Parametric Procedural Models for 3D Object Retrieval, Classification and Parameterization

dem Fachbereich Informatik
der Technischen Universität Darmstadt
vorzulegende

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

M.Sc. Roman Getto

geboren in Heidelberg, Deutschland

Referenten der Arbeit:	Prof. Dr. techn. Dieter W. Fellner Technische Universität Darmstadt Prof. Dr. rer. nat. Tobias Schreck Technische Universität Graz
Tag der Einreichung:	12/02/2019
Tag der mündlichen Prüfung:	05/04/2019

Darmstädter Dissertation
D 17

Getto, Roman: Parametric Procedural Models for 3D Object Retrieval, Classification and Parameterization.
Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2019
Tag der mündlichen Prüfung: 05.04.2019

Erklärung zur Dissertation

Hiermit versichere ich die vorliegende Dissertation selbständig nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 12/02/2019 Roman Getto

Abstract

The amount of 3D objects has grown over the last decades, but we can expect that it will grow much further in the future. 3D objects are also becoming more and more accessible to non-expert users. The growing amount of available 3D data is welcome for everyone working with this type of data, as the creation and acquisition of many 3D objects is still costly. However, the vast majority of available 3D objects are only present as pure polygon meshes. We arguably can not assume to get meta-data and additional semantics delivered together with 3D objects stemming from non-expert or 3D scans of real objects from automatic systems. For this reason content-based retrieval and classification techniques for 3D objects has been developed.

Many systems are based on the completely unsupervised case. However, previous work has shown that there are strong possibilities of highly increasing the performance of these tasks by using any type of previous knowledge. In this thesis I use procedural models as previous knowledge. Procedural models describe the construction process of a 3D object instead of explicitly describing the components of the surface. These models can include parameters into the construction process to generate variations of the resulting 3D object. Procedural representations are present in many domains, as these implicit representations are vastly superior to any explicit representation in terms of content generation, flexibility and reusability. Therefore, using a procedural representation always has the potential of outclassing other approaches in many aspects. The usage of procedural models in 3D object retrieval and classification is not highly researched as this powerful representation can be arbitrary complex to create and handle. In the 3D object domain, procedural models are mostly used for highly regularized structures like buildings and trees.

However, Procedural models can deeply improve 3D object retrieval and classification, as this representation is able to offer a persistent and reusable full description of a type of object. This description can be used for queries and class definitions without any additional data. Furthermore, the initial classification can be improved further by using a procedural model: A procedural model allows to completely parameterize an unknown object and further identify characteristics of different class members. The only drawback is that the manual design and creation of specialized procedural models itself is very costly. In this thesis I concentrate on the generalization and automation of procedural models for the application in 3D object retrieval and 3D object classification.

For the generalization and automation of procedural models I propose to offer different levels of interaction for a user to fulfill the possible needs of control and automation. This thesis presents new approaches for different levels of automation: the automatic generation of procedural models from a single exemplary 3D object. The semi-automatic creation of a procedural model with a sketch-based modeling tool. And the manual definition a procedural model with restricted variation space. The

second important step is the insertion of parameters into the procedural model, to define the variations of the resulting 3D object. For this step I also propose several possibilities for the optimal level of control and automation: An automatic parameter detection technique. A semi-automatic deformation based insertion. And an interface for manually inserting parameters by choosing one of the offered insertion principles. It is also possible to manually insert parameters into the procedures if the user needs the full control on the lowest level.

To enable the usage of procedural models directly for 3D object retrieval and classification techniques I propose descriptor-based and deep learning based approaches. Descriptors measure the difference of 3D objects. By using descriptors as comparison algorithm, we can define the distance between procedural models and other objects and order these by similarity. The procedural models are sampled and compared to retrieve an optimal object retrieval list. We can also directly use procedural models as data basis for a retraining of a convolutional neural network. By deep learning a set of procedural models we can directly classify new unknown objects without any further large learning database. Additionally, I propose a new multi-layered parameter estimation approach using three different comparison measures to parameterize an unknown object. Hence, an unknown object is not only classified with a procedural model but the approach is also able to gather new information about the characteristics of the object by using the procedural model for the parameterization of the unknown object.

As a result, the combination of procedural models with the tasks of 3D object retrieval and classification lead to a meta concept of a holistically seamless system of defining, generating, comparing, identifying, retrieving, recombining, editing and reusing 3D objects.

Zusammenfassung (Abstract – German)

Die Anzahl der 3D-Objekte ist in den letzten Jahrzehnten stark angewachsen, jedoch können wir erwarten, dass sie in Zukunft noch viel weiter wachsen wird. Auch für Laien werden 3D-Objekte immer zugänglicher. Die wachsende Menge an verfügbaren 3D-Daten ist für alle, die mit dieser Art von Daten arbeiten, sehr zu begrüßen, da die Erstellung und Beschaffung vieler 3D-Objekte immer noch aufwendig ist. Nach wie vor sind die überwiegende Mehrheit der verfügbaren 3D-Objekte nur als reine Polygonnetze vorhanden. Da diese Objekte auch zu großen Teilen von Laien stammen oder 3D-Scans von realen Objekten darstellen, die aus automatischen Systemen stammen, können wir nicht davon ausgehen, dass Metadaten und zusätzliche Semantiken mitgeliefert werden. Aus diesem Grund wurden inhaltsbasierte Such- und Klassifizierungstechniken für 3D-Objekte entwickelt.

Viele Systeme basieren auf dem unüberwachten Fall. Bisherige Arbeiten haben jedoch gezeigt, dass es Möglichkeiten gibt, die Präzision durch die Nutzung von Vorkenntnissen stark zu erhöhen. In dieser Arbeit verwende ich prozedurale Modelle als Vorkenntnis für das System. Prozedurale Modelle beschreiben den Konstruktionsprozess eines 3D-Objekts, anstatt die Komponenten der Oberfläche explizit zu beschreiben. Diese Modelle können Parameter in den Konstruktionsprozess einbeziehen, um Variationen des resultierenden 3D-Objekts zu erzeugen. Prozedurale Repräsentationen sind in vielen Bereichen vorhanden, da diese impliziten Darstellungen in Hinsicht auf automatische Erzeugung, Flexibilität und Wiederverwendbarkeit weit überlegen sind. Deshalb hat die Verwendung einer prozeduralen Repräsentation grundsätzlich das Potenzial, herkömmliche Ansätze in vielerlei Hinsicht zu übertreffen. Die Verwendung von prozeduralen Modellen bei der Suche und Klassifizierung von 3D-Objekten ist noch wenig erforscht, da diese leistungsstarke Darstellung beliebig komplex in der Erstellung und Handhabung sein kann. Im 3D-Objektbereich werden prozedurale Modelle vor allem für hochgradig reguläre Strukturen wie Gebäude und Bäume eingesetzt.

Prozedurale Modelle können jedoch die Suche und Klassifizierung von 3D-Objekten erheblich verbessern, da diese Darstellung eine dauerhafte und wiederverwendbare vollständige Beschreibung eines Objekttyps bietet. Diese Beschreibung kann für Suchanfragen und Klassendefinitionen ohne zusätzliche Daten verwendet werden. Darüber hinaus kann eine anfängliche Klassifizierung durch ein prozedurales Modell weiter verbessert werden: Ein prozedurales Modell ermöglicht es, ein unbekanntes Objekt vollständig zu parametrisieren und Merkmale verschiedener Klassenmitglieder zu identifizieren. Der einzige Nachteil ist, dass das manuelle Design und Erzeugung spezieller prozeduraler Modelle sehr aufwendig ist. In dieser Arbeit konzentriere ich mich auf die Generalisierung und Automatisierung von prozeduralen Modellen für die Verwendung in der 3D-Objektsuche und 3D-Objektklassifizierung.

Für die Generalisierung und Automatisierung von prozeduralen Modellen schlage ich vor, verschiedene Ebenen der Interaktion für einen Benutzer anzubieten, um die möglichen Anforderungen

der Kontrollierbarkeit und Automatisierung zu erfüllen. Diese Arbeit stellt neue Ansätze für verschiedene Automatisierungsstufen vor: die automatische Generierung von prozeduralen Modellen aus einem einzigen exemplarischen 3D-Objekt. Die halbautomatische Erstellung eines prozeduralen Modells mit einem zeichenbasierten Modellierungstool. Und die manuelle Definition eines prozeduralen Modells mit beschränktem Variationsraum. Der zweite wichtige Schritt ist das Einfügen von Parametern in das prozedurale Modell, um die Variationen des resultierenden 3D-Objekts zu definieren. Für diesen Schritt schlage ich auch mehrere Möglichkeiten für den optimalen Kontrollierbarkeits- und Automatisierungsgrad vor: Eine automatische Parametererkennungstechnik. Eine halbautomatische verformungsbasierte parametereinfügetechnik. Und ein Interface zum manuellen Einfügen von Parametern durch die Auswahl eines geeigneten Einfügeprinzips. Es ist auch möglich, Parameter manuell in die Prozeduren einzufügen, für den Fall, dass der Benutzer die volle Kontrolle auf der untersten Ebene benötigt.

Um die Verwendung von prozeduralen Modellen direkt für die Suche und Klassifizierung von 3D-Objekten zu ermöglichen, schlage ich deskriptorbasierte und tiefgehend lernende Ansätze vor. Deskriptoren messen die Differenz von 3D-Objekten. Durch die Verwendung von Deskriptoren als Vergleichsalgorithmus können wir den Abstand zwischen prozeduralen Modellen und anderen Objekten definieren und diese nach Ähnlichkeit ordnen. Die prozeduralen Modelle werden abgetastet und verglichen, um eine optimale Objektergebnisliste zu erhalten. Wir können auch direkt prozedurale Modelle als Datenbasis für ein umlernen eines neuronalen Faltungsnetzwerkes verwenden. Durch das tiefgehende Lernen einer Reihe von prozeduralen Modellen können wir neue unbekannte Objekte ohne weitere große Lerndatenbank direkt klassifizieren. Zusätzlich schlage ich einen neuen mehrschichtigen Parameterbestimmungsansatz vor, der drei verschiedene Vergleichsmaße zur Parametrisierung eines unbekannten Objekts verwendet. Somit wird ein unbekanntes Objekt nicht nur mit einem prozeduralen Modell klassifiziert, sondern der Ansatz ist auch in der Lage, neue Informationen über die Eigenschaften des Objekts zu sammeln, indem er das prozedurale Modell für die anschließende Parametrisierung verwendet.

Die Kombination von prozeduralen Modellen mit der 3D-Objektsuche und -Klassifizierung führt zu einem Metakonzept eines ganzheitlich nahtlosen Systems zum Definieren, Erzeugen, Vergleichen, Identifizieren, Suchen, Rekombinieren, Bearbeiten und Wiederverwenden von 3D-Objekten.

Preface and Acknowledgments

Several people worked with me on my ideas. For this reason, I use the "we" form throughout the main parts of this thesis. Also, the "we" form is the usual academic form in computer science. Only in the introduction and conclusion I use the first-person singular to emphasize my personal views and contributions.

In this context, I want to thank everyone that supported me during the work on my thesis and all co-authors of my papers. Especially I want to thank my supervisor Dieter W. Fellner and my second supervisor Arjan Kuijper, who guided me on my way. Also, a special thanks to my colleagues that supported me on a day by day basis and thoroughly discussed my ideas with me in the PhD Seminar: (In no specific order) Tatiana von Landesberger, Felix Brodkorb, Kathrin Ballweg, Marcel Wunderlich, David Kügler, Johannes Fauser, Marcelo Walter, Jürgen Bernard, Anirban Mukhopadhyay.

Prepublished Papers

The following papers include prepublished content of this PhD thesis. Within this thesis, parts of the papers are present verbatim or with minor changes.

[GKF18] GETTO R., KUIJPER A., FELLNER D. W.: Automatic Procedural Model Generation for 3D Object Variation. *The Visual Computer* (2018), 1–18.

[GFJ*18] GETTO R., FINA K., JARMS L., KUIJPER A., FELLNER D. W.: 3D Object Classification and Parameter Estimation based on Parametric Procedural Models. In *23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (2018).

[MGKF18] MERZ J., GETTO R., KUIJPER A., FELLNER D. W.: Simplified Definition of Parameter Spaces of a Procedural Model using Sketch-Based Interaction. In *Proceedings of the 13th International Conference on Computer Graphics Theory and Applications* (2018).

[GKF17] GETTO R., KUIJPER A., FELLNER D. W.: Unsupervised 3D Object Retrieval with Parameter-Free Hierarchical Clustering. In *Proceedings of the Computer Graphics International Conference* (2017), no. 7, ACM.

[GMKF17] GETTO R., MERZ J., KUIJPER A., FELLNER D. W.: 3D Meta Model Generation with Application in 3D Object Retrieval. In *Proceedings of the Computer Graphics International Conference* (2017), no. 6, ACM.

[GF15] GETTO R., FELLNER D. W.: 3D Object Retrieval with Parametric Templates. In *Proceedings of the Eurographics Workshop on 3D Object Retrieval* (2015), Eurographics Association, pp. 47–54.

[GKvL15] GETTO R., KUIJPER A., VON LANDESBERGER T.: Extended Surface Distance for Local Evaluation of 3D Medical Image Segmentations. *The Visual Computer 31*, 6-8 (2015), 989–999.

Contents

Part 1.

Introduction

1.1. Motivation

3D objects are used in numerous applications, ranging from rendering in digital art, films and video games, to cultural heritage analysis, to scientific physical simulation and visualization, to medical surgery assistance. The digital representation of objects as 3 dimensional surfaces or volumes has become more and more prevalent and accessible during the last decades and seems to raise even further in the future. 3D objects do not only emerge from the hands of professional artists. Many users are able to use systems like Google SketchUp [GL06] to create their own 3D objects. Also, 3D objects come from digitization of real objects which has become very accessible.

The ongoing growing of 3D object data in the world has clear benefits for everyone working with this data, since the acquisition becomes easier and cheaper. However, meta-data and additional semantics are seldom, when the objects are not created by experts. The vast majority of the novel data is a raw mesh of polygons, making it exceedingly difficult to find, sort and compare 3D object data.

For this reason, content-based techniques have been developed. Content-based means that the technique is not based on any meta-data (e.g. text-data) but solely on the content itself, i.e. the 3D data. The goal of all techniques is to support the user with additional information about the raw polygon meshes. The user might have different tasks in mind: searching for a specific 3D object in a database of objects, searching for similar objects in a database, exploring a database without any specific object in mind, searching for objects of a certain type/class/category, comparing and sorting objects of the same class or from different classes. In all cases the user needs additional information about the objects. The information can consist of a list sorted by similarity to a reference object, or it can be a class or category label. The information can also be numerical by quantifying the similarity of objects or by quantifying characteristics of the objects.

1.1.1. 3D Object Retrieval and Classification

All tasks can be generalized to two cases: Either we have a query and want to find an object or we already have an object and want to gather information about it. The first case corresponds to the task of 3D object retrieval: we have a query object and are searching for similar objects (See Figure 1.1.1). The second case corresponds to 3D object classification: we have an unknown object and want to gather information by classifying it (See Figure 1.1.2).

3D object retrieval (3DOR) covers the case of a dedicated search of a specific object but it can also be used for an explorative search. The query object can be a new 3D mesh, an example mesh from any database or even a random object from the same database to start an explorative search. It is also

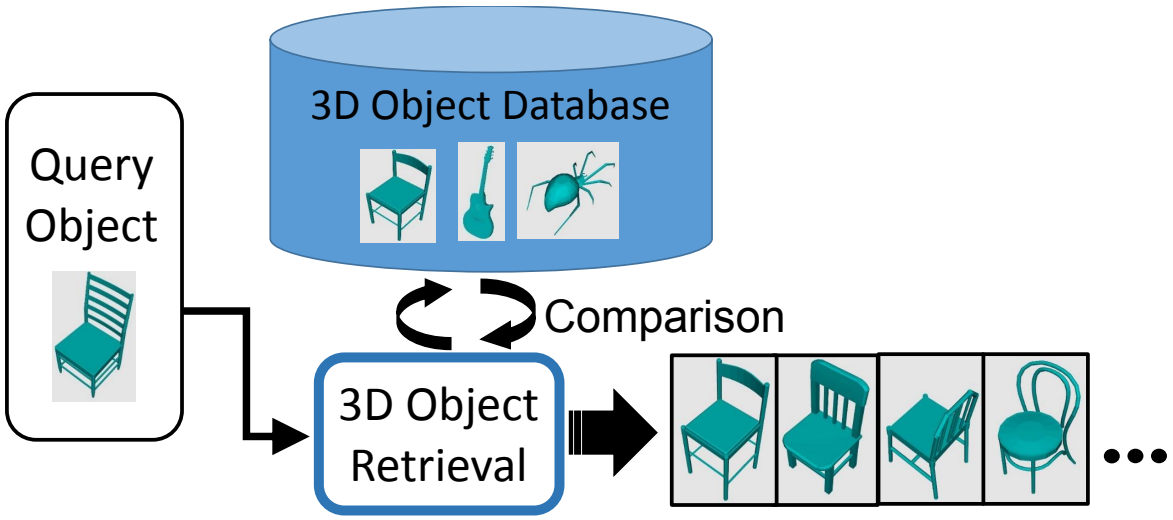


Figure 1.1.1.: 3D object retrieval: A single query is given. The query can be a 3D object itself but can also have another form. The query is compared to all objects of a database. The result is a list of object sorted by similarity. The first object is the most similar object.

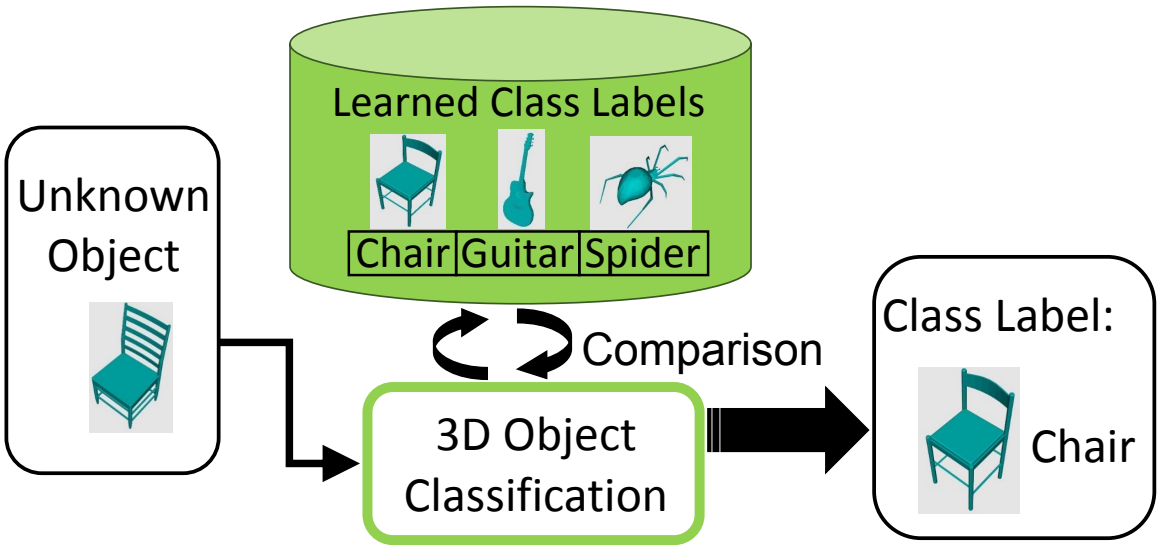


Figure 1.1.2.: 3D object classification: An unknown object is given. The unknown object is compared to all learned class labels and the object is labeled by the most similar class.

possible that the query is not a real 3D object but rather an image or topological graph if the comparison algorithm allows this representation.

3D object classification can be used for a single object but it can also be used to filter or cluster a complete database, by classifying all objects together. The class labels can be exclusive labels or can overlap, so that one object can have several labels. The class labels can have different semantic levels, e.g. a rather broad level (mammals, insects, furniture) or a more specific level (shorthair cats, longhair cats).

The problem of the classification task is that we need sufficient labeled data to learn the classes in advance. In a real scenario with a specific class in mind the creation and acquisition of learning data is difficult. Many object variations are needed to correctly define a single class. The results of the classification algorithms highly depend on the amount of adequate learning data.

For the 3DOR task no learning data is needed. Nevertheless, it highly profits from additional data for the query. In the technique of relevance feedback [LZYX15, ASYS10, LMT05] the user can make an initial query and then mark related retrievals as correct. Effectively this corresponds to a query refinement, so that the query consists of multiple objects. This technique leads to a highly increased accuracy. However, the success of the technique for the single case is dependent on the database, as the user has to mark related retrievals stemming from the database itself. In general, the single query represents an object that is similar to what the user is searching for. When the user can give several objects that are similar to the object he is searching for, the query itself is a more accurate representation of the search that the user has in mind. Therefore, specifying the search with several object variations instead of a single one leads to more accurate results.

Summarizing, additional object data and shape variations of a single object, are highly beneficial for 3D object retrieval and 3D object classification. Creating many object variations manually is a very time consuming and costly task. What we really need is the possibility of directly creating the whole object variation space for a complete class. This should be possible by only using a single example object as starting point.

1.1.2. Procedural Models

There are many methods that can produce shape variations automatically, but these always have strong limitations. Either it is only possible for specific object types [BWSK12] or the possibilities of variations are very limited [WXL*11] or the shape variations are only possible by recombination of objects, which only works if many different objects are available in advance [JTRS12].

A well known and popular method for content generation are procedural models [STBB14]. Procedural models are an implicit representation of an object. Instead of listing all components, the procedural model describes the construction of an object. The advantage of this representation is the huge flexibility. The construction process offers the possibility of directly including variations. Procedural models initially came from other domains, but started to grow in the domain of 3D objects [STBB14, GLXJ14]. Procedural models were used predominantly in areas with regular structures,

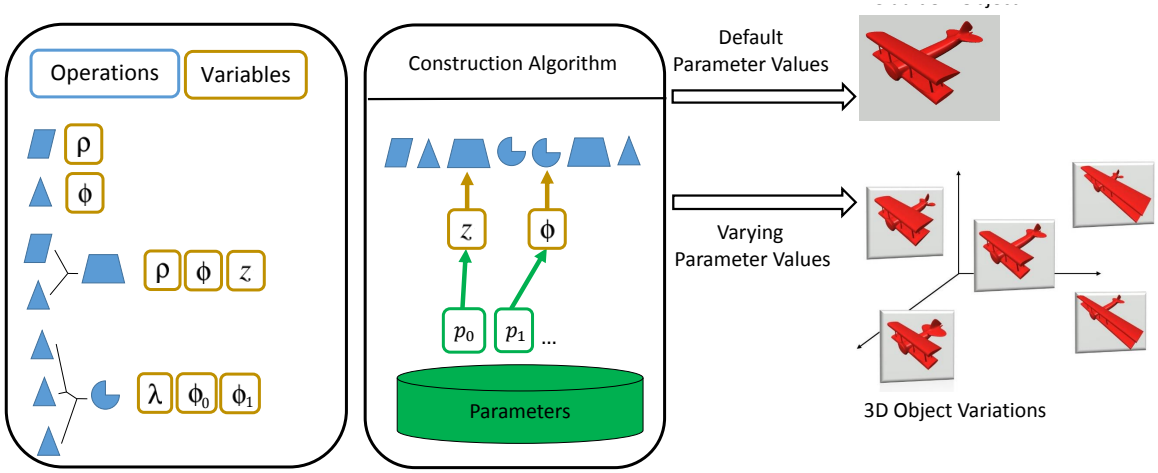


Figure 1.1.3.: Procedural models: A procedural model consists of a definition of operations, a construction algorithm and parameters. Higher operations can be can be composed of lower operations. The operations include variables which influence the outcome of the operations. A construction algorithm defines the sequence of operations which produce a 3D object like a biplane. The parameters influence the values of the variables and therefore influence the final shape of the resulting 3D object.

like trees [SPK*14, LRBP12, BSMM11] and buildings [NGDA*16, VGDA*12, TLL*11]. The construction process for regular structures can be define by a grammar [TLL*11] or l-system [SPK*14].

Havemann [HF05, HF03] proposed another representation of procedural models. A model is directly represented by a sequence of construction procedures. The procedures can include smaller procedures, so that procedures can be constructed hierarchically. Also, procedures can include parameters so that changing a parameter results in a variation of the final object. This concept is shown in Figure 1.1.3. Havemann called this models 'generative models', since the model directly generates a 3D object. In my work I preferred the initial more general term of procedural models, since the term 'generative model' is often used for other techniques. Also, 'procedural models' is a more well-known term for similar approaches.

The previous work based on Havemanns representation [UF11, MSH*08, HF04] is mainly concentrated on the manual construction of a detailed and elaborated generative/procedural model, consisting of special construction procedures defined by experts. For example this can be chair, a cup, an underground tube system, a Gothic window or a complete cathedral. The construction of a single object is time consuming. To bring this concept to a new level, we need to automate and generalize the construction, parameterization and variation generation of procedural models.

1.1.3. Thesis Goal and Structure

This thesis combines the topics of procedural models and 3DOR/3D object classification. In my contribution I concentrate on the automation and generalization of procedural models to make them more suitable for the retrieval and classification task. For 3DOR and 3D object classification I concentrate on finding the best possibilities of using procedural models for this tasks.

Furthermore, the use of procedural models in a 3D classification scenario even brings further benefits. Ullrich et al. [UF11] proposed to not only use procedural models for the classification of an unknown object but also generate additional information by determining the parameters of the procedural model which represents the class of the unknown object. This is an enhancement of the classification task. The unknown object is classified and parameterized based on the procedural model. I propose a new system to further extent the possibilities of parameterizing unknown objects.

In the following section 1.2, I introduce the meta concept which explains the relation of a all single contributions. Section 1.3 presents a detailed list of all contributions.

The following Part 2 presents all contributions related to the automation and generalization of procedural models. In Part 3, I present all contributions in the applications of 3DOR and 3D object classification, the integration of procedural models into the applications and the parameterization of objects based on procedural models. In the final Part 4, I summarize the thesis and outline future work.

1.2. Meta Concept

The goal of my concept is to automate the process of creating and parameterizing a procedural model and subsequently using it for 3DOR and classification. However, with the introduction of more automation, the user inevitably has less control over the resulting procedural model. The optimal level of interaction differs from case to case. For this reason, I introduce a meta concept including multiple options for the level of interaction. The user can decide about the level of control and automation.

In this thesis I propose several new algorithms for procedural models and their application in 3DOR and classification. The algorithms differ in the level of automation and control. In the following I first explain the processing sequence in the meta concept. Then I describe each step in further detail.

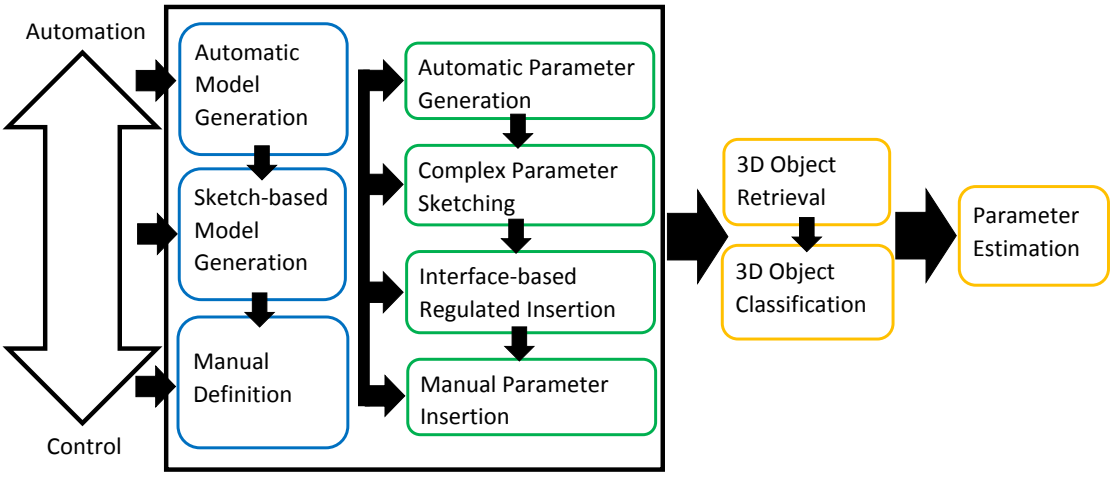


Figure 1.2.1.: Meta Concept: The user can choose the suitable level of automation and control for the creation and parameterization of a procedural model. Subsequently the procedural model can be used for 3D object retrieval and classification. The target object can then be parameterized by estimating the parameters of the procedural model, which corresponds to the target object.

Figure 1.2.1 illustrates the meta concept. The user can choose between manual, semi-automatic and fully automatic approaches to generate and parameterize the procedural model. Subsequently the user can use the generated procedural models for 3D object retrieval and 3D object classification tasks. In

the final step the parameters of the procedural model are estimated to further quantify the characteristics of the retrieved or classified object.

The blue and green boxes describe the complete process of generating a parametric procedural model. The blue boxes consist of the construction of the sequence of procedures, which correspond to the procedural model. The green boxes represent the process of inserting parameters into the procedural model to define all possible variations. For both, blue and green boxes, the higher levels offer more automation but less control and the lower levels offer a higher control but less automation.

Also, note the arrows pointing from a box of a higher level to a box of the same color in lower level. These demonstrate that it is possible to combine different levels for the construction of the parameterization of the procedural model. For example the initial model can be generated automatically and subsequently refined manually. Furthermore, any level of the construction (blue boxes) can be combined with any level of the parameterization (green boxes). A manually defined procedural model can be automatically parameterized. An automatically generated procedural model can be manually parameterized.

Independently of the chosen levels for the creation of the procedural models, these can be used for the subsequent tasks of information search and gathering. The yellow boxes represent these. The procedural models can be used for a 3D object retrieval or 3D object classification task. A retrieved object can also be classified subsequently, indicated by the arrow pointing downwards. In every case, the retrieved and/or classified object can be further quantified by the parameter estimation task.

Manual Definition (blue): The user can create everything manually. The user can define any procedures. (See Section 2.3 and [GF15])

Sketch-based Generation (blue): The user can use a sketch-based modeling tool to construct a single object which is automatically transformed to a parameterizable procedural model. The procedural model consists of modeling construction procedures. He has the full control which procedures are used for which parts. (See Section 2.4 and [GMKF17])

Automatic Model Generation (blue): From a single example 3D object as polygon mesh a procedural model is generated. The complete procedural model is a reverse construction of the single object and consists of parameterizable modeling construction procedures. (See Section 2.7 and [GKF18])

Manual Parameter Insertion (green): The user can define all parameters manually by directly exchanging the values of single procedures a parameter. (See 2.3 and [GF15]))

Interface-based Regulated Insertion (green): The user can use an interface giving several possibilities of inserting new parameters. The user has to choose where, with what parameter range and which insertion scheme he wants to use. The parameter is then included in the procedural model. (See Section 2.4 and [GMKF17])

Complex Parameter Sketching (green): The user can sketch complex parameters. The suited procedures and the location of the insertion are automatically detected and the parameter range is automatically defined. The parameter is integrated into the procedural model. (See Section 2.5 and [MGKF18])

Automatic Parameter Generation (green): Parameters are computed completely automatically. Using an importance measure all parameters are sorted by importance. Only the parameters with an importance higher than 1 are inserted into the procedural model. Alternatively the user can inspect all offered parameters and choose or discard parameters as desired. (See Section 2.6 and [GFJ*18])

3D Object Retrieval (yellow): Independently of the preceding chosen interaction levels, the user can use his procedural models to find models similar to any variation of the procedural model within databases. (See Section 3.5, [GF15] and [GMKF17])

3D Object Classification (yellow): The user can use several procedural models to learn a classifier using deep learning. Unknown objects can then be classified and associated to one of the available procedural models. (See Section 3.6 and [GFJ*18])

Parameter Estimation (yellow): When an unknown object is associated to a procedural model (via classification or retrieved by 3DOR), it is possible to gather additional information by parameterizing the unknown object with respect to the parameters of the procedural model. (See Section 3.7 and [GFJ*18])

1.3. Contributions

This PhD thesis contributes at many points in the domains of procedural models and 3D object retrieval and classification. While some contributions fluently transition from previous work other contributions are targeted towards completely novel directions. Here I explicitly list all major contributions.

Additionally, I split the contributions into three categories: Concepts, techniques and insights. Concepts guide the research. Techniques are developed to reach the research goals and gain new insights. Insights are answers to research questions.

1.3.1. Concepts

Concept 1: I propose the concept of generalizing procedural models as a concatenation of basic procedures to open up the possibilities of using procedural models for broader applications. Additionally, I propose to restrict the parameter ranges to create a well-defined multi dimensional space by the parameters of the procedural model. (See Section 2.7 and [GKF18], Section 2.4 and [GMKF17], Section 2.3 and [GF15])

Concept 2 : I propose to use different interaction levels at the creation or generation of procedural models to overcome the limitations of larger applications. In this context I propose to automate the construction with a semi-automatic modeling approach or by automatically generating a procedural model from a single example. Additionally, I propose to (semi-)automate the process of parameterization of procedural models. (Section 2.4 and [GMKF17], Section 2.7 and [GKF18], See Section 2.5 and [MGKF18], Section 2.6 and [GFJ*18])

Concept 3: I propose to use the generalized procedural models as data basis for 3D object retrieval and classification applications, enabling possibilities of directly learning, classifying and retrieving 3D objects with procedural models. (See Section 3.6 and [GFJ*18], Section 3.5, [GMKF17] and [GF15])

Concept 4: I propose to use a layered classification and parameterization system to retrieve information at a suitable level about every unknown object using procedural models (See Section 3.7 and [GFJ*18])

1.3.2. Insights

Insight to Concept 1: How to generalize and automate procedural model generation? Procedural models in the sense of a hierarchy or concatenation of parametric procedures are by definition composed of smaller units of construction. In the manual design process the smaller construction units are individ-

ually designed for each object. This is the point where the generalization can be applied. Generalized smaller units of construction can be put together to form arbitrary procedural models. Also, general units of construction can be automatically detected, generated and parameterized. (See Section 2.7 and [GKF18], Section 2.4 and [GMKF17], Section 2.3 and [GF15])

Insight 1 to Concept 2: Is the semantic context of procedural models lost in automated procedural models? No. One benefit of procedural models is the semantic context of the parameters, e.g a parameter can depict the 'length of the legs'. In the manual process the semantic connection is established additionally to the design process. With generalized construction units the semantic context of the parameters can still be established on top of the construction units. (See Section 2.7 and [GKF18], Section 2.6 and [GFJ*18])

Insight 2 to Concept 2: Can we automate the parameterization of the procedural model? Mostly yes, but not completely. Many variations of objects are from geometrical nature and varying parts of the object in length, size and form. However, more complicated parameters with complex semantic coherence are still exclusively available by design. Though, semi-automation of the parameter definition heavily eases the design process. The correct level of automation of the parameterization has to be dictated by the needs of the specific application. (Section 2.6 and [GFJ*18])

Insight 3 to Concept 2: Can the parameters of generalized procedures reproduce any object variation? Yes, a small set of generalized procedures is able to produce any possible shape. When complex parameters are inserted into the procedures, these also allow any complex variation. (See Section 2.5 and [MGKF18])

Insight 1 to Concept 3: Can procedural models be used for general 3D object retrieval and classification applications? Yes, procedural models are perfectly suited to classify and quantify an unknown object. However, the manual design of a single procedural model is very costly. An elaborated design can be desired and suited in single cases. Still, 3D object retrieval and classification applications are concerned about diverse objects in general. In order to make procedural models an accessible approach for these applications the process of generating procedural models has to be further generalized and automated. (See Section 3.5, [GF15] and [GMKF17], Section 3.6 and [GFJ*18])

Insight 2 to Concept 3: How precise is a classification and retrieval only based on procedural models? Very precise. The precision is similar to learning techniques using large learning databases tailored towards the specific set of classes. In real scenarios good and large learning databases are arguably not available or very costly to create. Hence, procedural models are superior in this case. The State-of-the-art recognition techniques achieve classification accuracies of $85-90\%$ in big databases. This includes unusual objects where even humans would struggle in classification. Though, the relative accuracy in comparison to human-possible recognition is even higher. This is similar to the circumstances in 2D image recognition where the neural networks already achieved higher recognition rates than the average human recognition rate [WYS*15]. (See Section 3.6 and [GFJ*18])

Insight to Concept 4: Is it possible to quantify an object of a specific class with parameters? In many cases yes, but it completely depends on the quality and degree of detail of the procedural model. In the best case, the correct parameters of the procedural model produce an object that corresponds

exactly to the given object. If the procedural object is not detailed enough or the unknown object is very extraordinary, their might not be any parameter set precisely describing the object. Still, at the procedural model can roughly represent the main characteristics of the object. Furthermore, the level of abstraction can be automatically detected, so that the suitability of the procedural model to the unknown object can be quantified. (See Section 3.7 and [GFJ*18])

1.3.3. Techniques

Techniques for Concept 1: A scheme for manually defining and constraining general procedural models (See Section 2.3 and [GF15]). An approach to model a procedural model from generalized sketch-based modeling operations (See Section 2.4 and [GMKF17]). An approach to automatically generate a procedural model only using a few generalized modeling operations (See Section 2.7 and [GKF18]).

Techniques for Concept 2: An approach to automatically generate a parameterizable procedural model from a single modeling session (See Section 2.4 and [GMKF17]). An approach to completely automatically generate a procedural model from from a single example (See Section 2.7 and [GKF18]). A regulated insertion technique to create a well-defined parameterization of a procedural model (See Section 2.4 and [GMKF17]). A technique to automatically generate parameters for a predefined procedural model (See Section 2.6 and [GFJ*18]). A deformation based technique to define complex procedural model parameters (See Section 2.5 and [MGKF18]).

Techniques for Concept 3: A technique for 3D object retrieval with procedural models (See Section 3.5, [GF15] and [GMKF17]). A deep learning and classification approach only using procedural models as data foundation (See Section 3.6 and [GFJ*18]). A hierarchical clustering based technique to improve 3D object retrieval (See Section 3.4 and [GKF17]). A local distance measure to analyze the surface differences of two object. (See Section 3.3 and [GKvL15]).

Techniques for Concept 4: A layered parameterization approach to estimate the best parameters and the level of suitability for an unknown object using a procedural model (See Section 3.7 and [GFJ*18]). A Z-Buffer distance measure for measuring pixel-wise differences (See Section 3.7 and [GFJ*18]).

Part 2.

Procedural Models

2.1. Overview

In Part 2 of this thesis we present several possibilities of automating and generalizing procedural models. In this context we speak of procedural models as a sequence of parameterizable procedures. There are two steps in the creation of procedural models. In the first step the construction process of the object is defined. When the construction process is processed with default values an explicit 3D object as polygon mesh is generated. This object can be seen as the default object of the procedural model. In the second step the variation possibilities of the object are defined. That means that parameters are inserted into the procedural model and the parameter ranges are determined.

We propose a method for a completely automatic generation of a procedural model. This generalized procedural model is perfectly suited for broader applications. However, the drawback is that the automatic creation gives less control. Therefore, the target is, to give the user all possibilities to choose the right level of automation and control that he needs for his application. Hence, we also propose semi-automatic methods supporting the user in the creation of a procedural model. We use a sketch-based modeling to define a new object, which is automatically transformed to a procedural model and subsequently parameterized. For the parameterization we also propose different levels of control. We propose a fully automatic method but also a scheme for manual insertion and a semi-automatic method for complex parameters.

In the following chapter we discuss work related to procedural models. Then we describe our initial constrained manual procedural model construction. In chapter 2.4 we present our sketch-based modeling approach for the creation of generalized procedural models, together with the manual insertion scheme of parameters. The following two chapters describe a semi-automated and a fully automated technique for the parameter insertions. The final chapter 2.7 presents the approach for the fully automatic generation of a procedural model on the basis of a single 3D object example.

2.2. Related Work

Procedural models are an higher implicit 3D object representation. As such there are several fields related to procedural models. First, procedural and generative models in general. Second, methods that create object variations with part-based recombinations, since object variations are a main target of procedural models. Third, any other higher 3D object representation, with similar purpose as procedural models. Fourth, we also review sketch-based modeling systems, as we want to make it possible for the user to directly create procedural models with sketch-based modeling.

2.2.1. Procedural Models

In general a procedural model is the description of a construction scheme for a class of objects, which allows to easily generate many different variations. Therefore, procedural models excel in content generation [STBB14]. A procedural model can be represented by a grammar [TLL*11] or an L-system [SBM*10], allowing to define the construction with production rules. L-systems were originally defined by Lindenmayer [Lin68], tough also called Lindenmayer-system. These are especially suited for fractals, which makes them popular in the reproduction of natural growing schemes. They have been used to define parametric models [HRL75, TMW02] or combined with shape grammars [PM01].

In the task of inverse procedural modeling, the system finds rules to describe given example objects as procedural models. Stava et al. [SBM*10] propose a system, which automatically generates l-systems from given examples. They show that this is possible for several domains. However, the system only works for 2D images and not for 3D objects.

The construction scheme of procedural models can also be represented by a concatenation of parameterized procedures [HF05]. The sequence describes the construction process and the parameterization allows the variation of the construction process.

In their survey, Smelik et al. [STBB14] state that procedural models typically are used to generate virtual worlds due to their regular structure but also their need for countless variations to appear realistic. Hence, the original domain of procedural domains are well-regulated structures like trees, buildings and whole cities.

Trees are a common domain, as realistically looking trees can be generated by growing rules taken from nature. The user can guide this process by sketching his preference of grow direction [LRBP12] or directly defining the grow area [BSMM11]. As it can be cumbersome to tweak tree parameters to actually get the tree you want, also inverse procedural modeling has been established to automatically derive parameters from a given example [SPK*14].

Urban models are also well researched. Haegler et al. [HMVG09] uses procedural modeling to describe different buildings which are equal in their kind of structure. Marvie et al. [MPB05] also construct different buildings with procedural modeling by using L-systems. The building construction is mostly realized in grammars and L-systems, as buildings usually also follow a very ordered scheme of construction.

Muller et al. [MWH*06] propose a shape grammar combining volumetric parts. Starting with an initial part, the building is resized and the style of all new parts are changed so that it fits to the style of the initial part, e.g. the new facade has a uniform look and windows do not intersect with the borders of the volume. However, the rules are given manually.

In the approach of Nishida et al. [NGDA*16] the user does not directly give rules, but roughly sketches the desired type of structure. As the procedural generation with grammars can be difficult to control also methods came up to control the production of the grammar [TLL*11, MPB05] or inversely derive city structures from examples [VGDA*12] or facade models from images [MZWVG07].

Besides the construction of virtual worlds, procedural models have been integrated into other domains of 3D objects. Other domains do not offer the same level of well-regulated structures that are repeatedly used, however, the need for variations is present in many domains. For example Guo et al. [GLXJ14] proposed a creature grammar to create recombinations and variations of creatures.

In the last century Ramamoorthi and Arvo [RA99] already created a procedural model for 3D objects. They represent a single object by a combination of procedures. When the procedures are executed subsequently the object surface is generated. Each procedure has a mathematical function as parameter, which can be changed, so that the resulting object shape looks different. Hence, they generate variations by changing the function of a procedural model.

Bokeloh et al. [BWSK12] propose a procedural modeling algorithm which works on regular structured polygon meshes. First, the mesh is split into detected regular patterns. Then, parameters are defined to control all parts of the shape of the object. When the parameters change, the object shape is changed while preserving the regular patterns optimally. This approach shows that procedural models are generally very powerful in terms of flexibility. The editing of objects and generation of variations is extraordinary simple with this approach. This approach also automatically builds the procedural model from an example object. Still, it only works for highly regular structures.

The process of inverse procedural modeling constructs the rules from existing objects, enabling the possibility to combine the rules in a new way to produce new objects. Bokeloh et al. [BWS10] use this approach to define castle-wall construction rules and show that it is possible to arbitrarily combine these rules to construct new wall-combinations. The approach is based on finding 'dockers' within an object so that parts of the objects can be recombined. Any fitting part can be connected to the 'docker'. Fisher et al. [FRS*12] also uses inverse procedural modeling and finds rules for office-desk arrangements and combines them to new arrangements. The approach of Milliez et al. [MWCS13] offers the possibility of interactively designing new procedural models. The models consists of connectable parts and the user can pull the model to resize it. The model automatically adapts, so that new parts are generated

and arranged in an optimal way to fit to the given length. Nevertheless, these approaches are always based on finding and reconnecting patterns.

Procedural models can make repeated use of single procedures and recombine them to higher order procedures [HF05]. Describing an object in operations can be a very cheap description when high-level-operations are constructed from low-level-operations and reused several times in a description. Berndt et al. [BFH05] have shown that this description of an object is also perfect for transmission, as it uses less bandwidth.

The construction of a procedural model can be very complex. Havemann et al. [HF04] demonstrate the parametric design of a gothic window. Mendez et al. creates a procedural model for a complete urban pipeline [MSH*08]. Ullrich et al. [UF11] define a procedural model with parameters and show that variations of the modeled object can be generated by varying the parameters. However, the construction of the procedural model is very complex as it is constructed completely manually by coding all procedures and describing the object construction algorithm as program code. To support this complex process, Schinko et al. [SSUF10] presented an approach to define a procedural model for a general 3D object trough a simple scripting language. Still, the procedural model has to be manually coded. This manual process remains highly effortful. A generalization of procedural models for 3D objects would enable the possibilities of further automating the creation of procedural models. The automation of procedural models makes this highly flexible method of content generation suitable for broader applications.

2.2.2. Part-based Recombination for Object Variation

Several approaches generate variations from existing objects by using part-based recombination. Xu et al. [XLZ*10] generate new objects by co-analyzing several objects, which are of the same type (chairs) but have a different style. They find corresponding parts in all example objects and recombine parts of several objects with the possibility of scale changes of single parts. This method gives a fast and easy way to create new objects if an adequate set of objects is available for the co-analysis. Jain et al. [JTRS12] present an approach which also recombines parts of objects, but they also allow a blending from one object to another by iteratively replacing small parts of one object by parts of another object. A similar approach from Kraevoy et al. [KS04] uses cross-parameterization to combine parts of different objects.

Averkiou et al. [AKZM14] allow an even more sophisticated stepwise navigation between objects by projecting all objects of a set into a 2D object space. The user can choose a point within this object space and retrieves a combined synthesized object.

Yumer et al. [YK12] compute a co-abstraction for a set of objects to achieve similar level of details for all objects of the collection. In a later work Yumer et al. [YCHK15] define a whole parameter space of objects, interpolating between single initial objects.

In general the approaches which generate new object variations by using existing objects excel at being a fast and easy way of creating new 3D objects, as the creation of 3D objects is a time consuming task. The big disadvantage is that a sophisticated set of example objects is always needed.

2.2.3. Higher 3D Object Representations

In their State-of-the-Art report 'Structure Aware Shape Processing' Mitra et al. [MWZ*13] discuss algorithms and approaches processing an object on a higher level of abstraction. Structure aware algorithms transform the object into a specialized representation. They identify that procedural models are structure aware, since procedural models can easily include parameters to let the user change the object on a higher level of abstraction, e.g. adding a parameter 'leg length' to the rules or procedures of the legs of an animal.

In general, higher 3D object representations promote a simple 3D object representation (e.g. a polygon mesh) to a higher level of abstraction. This makes it easier to use them for different purposes. Generally speaking, the higher abstraction level adds a kind of semantic to the representation. For example a segmentation of the mesh has a different part for the 'head' and a 'leg', while a polygon mesh only offers pure geometry.

Many approaches propose such higher representations. In many cases, these special representations are chosen for similar reasons as procedural models. To reuse, change and vary parts of the objects.

Wyvill et al. proposed so-called BlobTrees. These are combined CSG-Tree with implicit functions on the leaf nodes. These allow blending [BBCW10] and editing, swapping and removal.

Wang et al. [WXL*11] present a different approach that creates a symmetry hierarchy of man made 3D objects. The resulting hierarchy can be changed to edit an object and create new variations. This offers the possibility to directly create variations from a single object. However, the possibilities of variations are also limited to any rotational or translational symmetries, which limits the approach to specific cases.

In the work of Thiery et al. [TGB13], they compute a so-called sphere-mesh representation where differently sized spheres are connected by edges and triangles to represent a rough approximation of the 3D object surface. They show that the representation can be used to facilitate the editing of the object. However, the possibilities to define variations are very limited. The approach aims at supporting manual editing.

Baerentzen et al. [BAS14] convert a mesh into a Polar-Annular Mesh representation (PAM) to facilitate editing. The PAM representation is a more regularized polygon mesh making it possible to compute a more accurate skeleton for articulating the mesh. Still, the PAM is a polygon mesh itself, only specially useful for this purpose.

Attene at al. [AFS06] fit differently sized cylinders into a mesh to construct a hierarchy of segmentations. Similarly, Raab et al. [RGS04] produce 'virtual woodwork' by transforming a mesh into a cylinder-wire-model, which can be seen as a separation into parts. Zhou et al. [ZYH*15] generate an

over-complete generalized-cylinder representation of the mesh which is then optimized to segment the mesh. Summarizing, many representations are used for segmentation, which is very useful in itself. However, the intermediate representations for the segmentation are not directly suitable for other tasks.

Another approach similar to procedural models are box templates. Averkiou et al. [AKZM14] extract templates represented as a set of loosely arranged boxes. The boxes describe the position and size of different parts of the objects. Ovsjanikov et al. [OLGM11] also use box templates. The user can change the position of the boxes to find similar 3D objects in a collection of objects. Kim et al. [KLM*13] learn box templates for specific object types from a collection of similarly shaped 3D objects. Fish et al. [FAVK*14] propose a representation for shape families also resulting in shared box templates for each family. In general box templates are a very high level of abstraction and are useful to roughly structure and compare objects with a shared topology. Nevertheless, box templates describe the rough arrangement of parts and are not able to represent any form and shape changes on a lower level.

Summarizing, all reviewed representations do not fulfill the same needs as procedural models. The representations either do not offer possibilities for the definition of variations or the representation are only suitable for special types of objects or the possible variations are very limited.

2.2.4. Sketch-based Modeling

We also reviewed modeling tools, especially the modeling tools using sketch-based operations. Gossweiler and Limber [GL06] introduce Google SketchUp, which is a tool for non-expert users showing the advantages of sketch-based modeling as it is easy to use and fast. The user can draw 2D silhouettes and extrude them to 3D to create 3D parts. Schmidt et al. [SWSJ07] propose a CSG approach using a hierarchy of volume models as underlying surface representation. The user can draw whole objects which are directly transformed into a volume representation. Bein et al. [BHSF09] propose a tool for sketching subdivision surfaces. This means that the user only models the control mesh and flags the edges as sharp or smooth and the underlying subdivision surface is automatically generated. Takayama et al. [TPSHSH13] present a sophisticated tool for professionals for the generation and editing of quad meshes. The sketching operations allow a detailed control of the resulting surface patches.

Sketch-based modeling systems can be separated into two different categories: recognition based and construction based [KYZ14].

Recognition based systems use the sketches as shape descriptors to retrieve shapes from a database [EHBA10, ERB*12]. This means that the user does not actually model a new object, instead known object parts are used to recombine to a new object. Therefore, a suited database of object parts is needed. This is not the case for our applications.

Construction based systems directly generate, add and edit parts based on the sketched line or shape. The initial approaches were mostly targeting the construction of 3D objects by drawing 2D objects and subsequently inflating them [IMT99, NISA07]. The resulting 3D object is the natural completion of the 2D drawings.

The more sophisticated approach of Schmidt et al. [SWSJ05] uses hierarchical implicit volume models (BlobTrees) as underlying data structure. This gives further possibilities of modeling complex forms with multiple sketches. Another approach [SS08] introduced layered surface editing operations which is also based on a tree structure like BlobTrees. However, Jorge et al. [JS11] evaluated that using a hierarchical tree-view for the manipulation is not intuitive for designers.

Sketch-based interaction is also used for other interactions with objects. Bessmeltsev et al. [BVS16] used sketch-based interactions do define the silhouette of a target pose. The pose of the object is automatically fitted to the silhouette. Choi et al. [CiRL*16] proposed to use sketches for single joints of an object to guide the animation. The character animation is automatically fitted to the curves approximating the drawn sketches. Similar to these approaches, we propose to use sketch-based interactions to define parameters of procedural models.

2.3. Manual Definition of Procedural Models

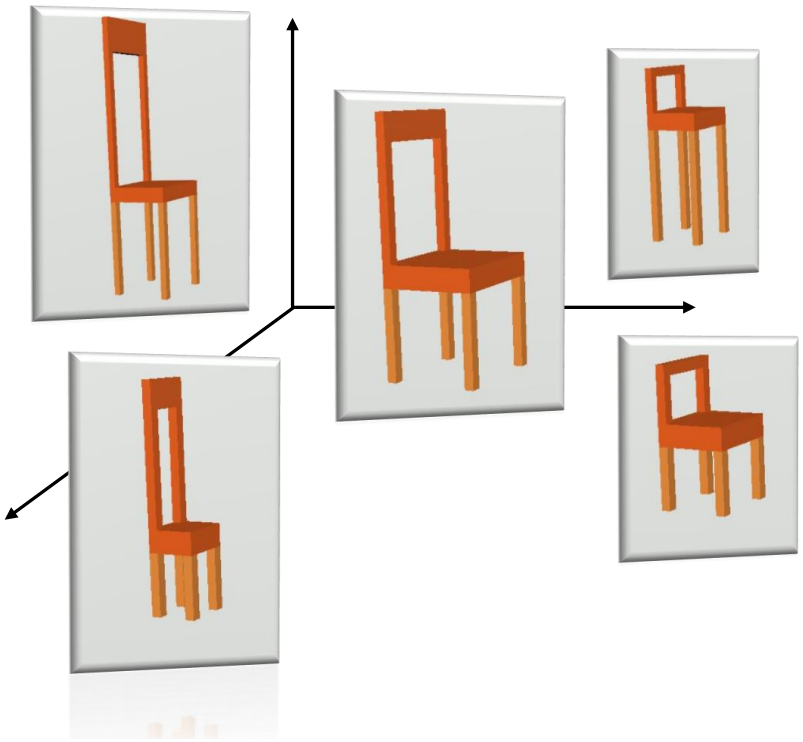


Figure 2.3.1.: The procedural model of a dining chair spans a multidimensional space of dining chair objects.

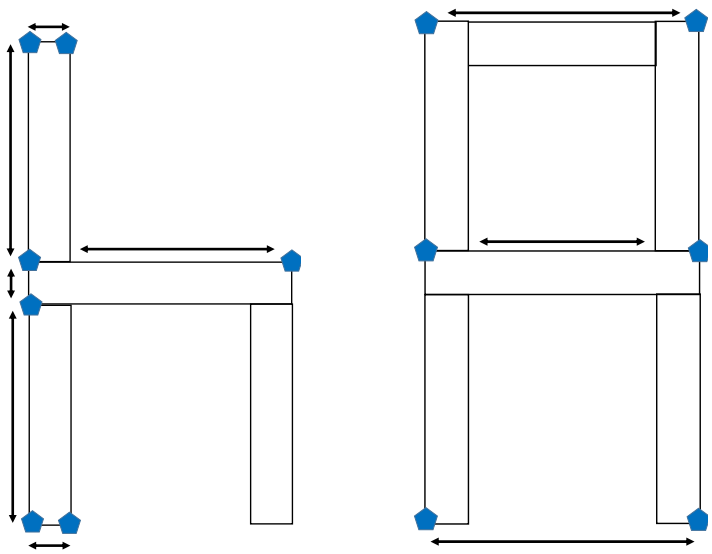
A procedural model consists of a chain of operations and a set of parameters. Executing the operations with the given parameters leads to a polygon mesh representation of an object. Such an object is called an instance of the procedural model. Hence, a procedural model can be understood as the definition of an object construction algorithm. Dependent on the parameters different instances are generated by the algorithm, therefore a procedural model spans a multidimensional space of objects (illustrated in Figure 2.3.1).

The construction algorithm can be arbitrarily complex and process any number of parameters. Hence, in general, any object can be created as procedural model. Nevertheless, an improperly complex high-dimensional representation is cumbersome to handle.

Therefore, when creating a procedural model object domain and a semantic concept like "chair","dining chair" or "biplane" one has to consider in which proprieties the objects should change and how much they should change, i.e. clearly defining and restricting the change axes of an object domain.

Indeed, if used correctly the resulting procedural model, i.e. object domain construction algorithm, can be a very compact representation and be very handy. The construction algorithm can consist of high-level-operations which are a reusable combinations of low-level-operations. Hence, the algorithm can be rather short. A representation of an object in operations can be more compact than a usual representation as a list of polygons [BFH05].

We use the Generative Modeling Language (GML: `//www.generative-modeling.org/`) created by Sven Havemann [HF05]. The language offers Euler operations working on a half-edge structure of a quad-based control mesh of a catmull-clark subdivision mesh.



(a) Side view (b) Front view

Figure 2.3.2.: The basic concept of a simple chair construction. The chair is represented by 10 points (3 points are shown in both views). The distance of the points to each other define the size of each part of the chair.

In Figure 2.3.2 we illustrate a basic concept for a procedural model of a chair. The parameters are 10 points represented by (x,y,z)-coordinates. The points define the size of all parts of the chair, but also where there chair is placed in Euclidean 3D space. Note, that this definition does not restrict how the

points are related one to another. Therefore, choosing bad values for the parameters could also result in a self-overlapping or degenerated mesh. Or the result could simply just not look like a chair.

For this reasons, procedural models should be represented in a restrictive space-invariant representation, which limit the possible change-axes of the chair. Also, the range of the possible parameters should be clearly defined. Therefore, we propose a restricted manual definition of a procedural model. With this restrictions every possible generated instance of the procedural model represents an actual object of the procedural model domain, e.g. a 'dining-chair'. Note that, only with this definition we can directly use any instance of the procedural model for applications like 3D object retrieval.

Furthermore, as we want to measure the difference of objects in a orientation, translation and scaling invariant form, we do not have to consider (x, y, z)-coordinates of the parameter points. We are much more interested in the ratios within the object, i.e. is the back of the chair much longer than the legs? Or are the legs of the chair thin or thick in relation to the height of the chair?

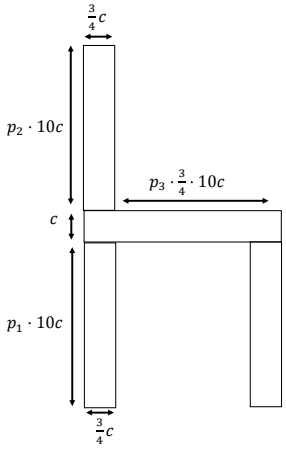


Figure 2.3.3.: This side view of the dining-chair definition shows the three lengths defined by the parameters p_1, p_2 and p_3 in relation to the constant c. All other parts are defined only in dependency of c.

For our procedural model of a dining chair we restrict one size c and introduce 3 parameters p_1, p_2, p_3, defining the ratios within the chair. Fig. 2.3.3 shows the definition of the 3 parameters. The thickness t

and the length l are defined as follows:

$$t_{legs} = \frac{3}{4}c \tag{2.3.1}$$

$$t_{back} = \frac{3}{4}c \tag{2.3.2}$$

$$t_{seat} = c \tag{2.3.3}$$

$$l_{legs} = p_1 \cdot 10c \tag{2.3.4}$$

$$l_{back} = p_2 \cdot 10c \tag{2.3.5}$$

$$l_{seat} = p_3 \cdot \frac{3}{4} \cdot 10c \tag{2.3.6}$$

We additionally restrict the possible change to a closed range:

$$p_1, p_2, p_3 \in [0.5, 1.0]$$

The seat is considered to be quadratic and therefore only has one size describing his side length. Note that the side length of the seat has a diminishing factor of $\frac{3}{4}$ since the size variance of the seat is smaller than the size variance of the legs and the back. The range $[0.5, 1.0]$ means that the length of the legs l_{legs} and the back l_{back} are 5 to 10 times the size of the thickness constant c. The side length of the seat l_{seat} is 3.75 to 7.5 times the size of the thickness constant c.

Choosing any value for the parameters p_1, p_2 and p_3 within their value range $[0.5, 1.0]$, results in the generation of a valid dining chair instance as polygon mesh. These can be used for 3D object retrieval applications. We use this definition in the retrieval with procedural models, described in Section 3.5

2.4. Semi-Automatic Model Generation

Manual procedural models include special procedures for every step of the construction. The procedures are explicitly tailored towards the type of object which is being generated. Our goal of generalization and automation includes the abstraction of special procedures to general procedures. We propose to use modeling procedures as general construction procedures. In modeling tools for the construction of 3D models several procedures are used to create a polygon mesh. Therefore, modeling procedures naturally are related to the construction of objects. We propose to translate the modeling procedures into procedures of a procedural model and parameterize these procedures. The result is the procedural model which corresponds to the parameterized construction process of an object.

The main problems with this approach lies in the translation of the modeling procedures. Just saving the raw values and the executed procedures would not lead to our goal. This would not allow a consistent parameterization. Inserting a parameter and changing the value of a raw modeling procedure would corrupt all the following procedures. This happens, because the correctness of all following procedures are explicitly dependent on the location and direction in the global coordinate system as well as any face IDs and local half-edge structures. For this reason we propose insertion principles and insertion techniques for the parameterization and translate the modeling procedures to suitable local procedure representations.

Our approach is based on the Generative Modeling Language (GML: http://www.generative-modeling.org/), created by Havemann [HF01] and we use the modeling tool and sketch-based modeling system proposed by Bein et al. [BHSF09].

We include the procedural model generation into the 3D sketch-based modeling process of the tool. The user can create a model in a given interface where he can use different operations on the vertices, faces and edges of a control mesh of a catmull-clark subdivision surface. The operations of the system are scale, rotate, insert, drag, connect, remove and flag. Additionally, the user can sketch face extrusions with two different sketching extrusion operations. We automatically generate the procedural model during the modeling process. We then insert parameters into the procedures of the model. The procedural model allows to generate variations of the initially modeled object by simply varying the parameters. We illustrate our concept in Figure 2.4.1.

We describe our approach in two subsections: the extraction of the procedural model and the insertion of the parameters into the procedural model.

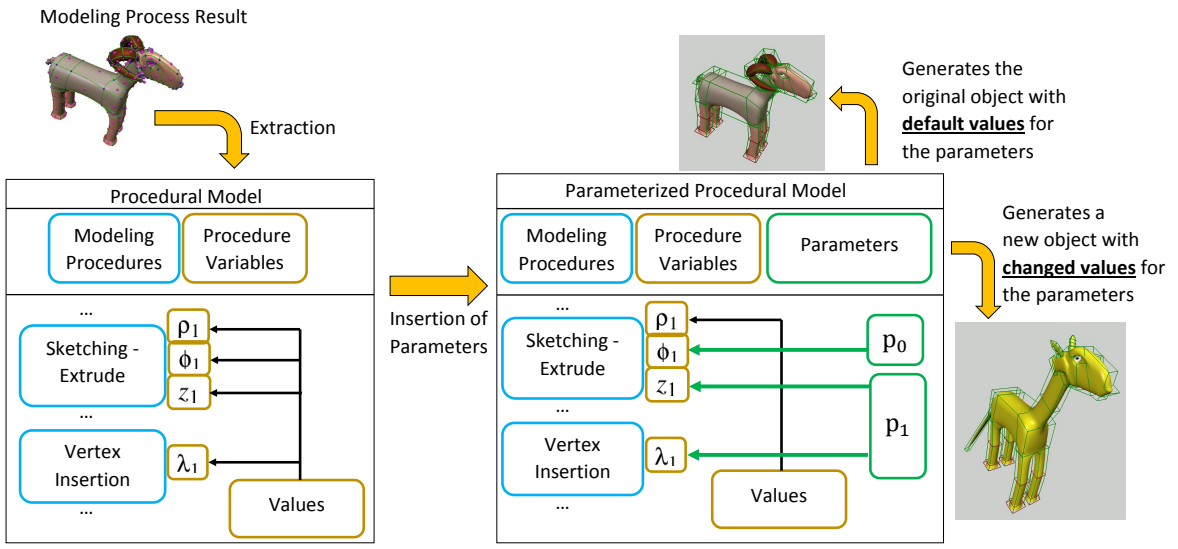


Figure 2.4.1.: We use a modeling tool to construct a single object. During the modeling phase we automatically extract a procedural model of the modeled object. By inserting parameters into the procedures we define the variations of the object.

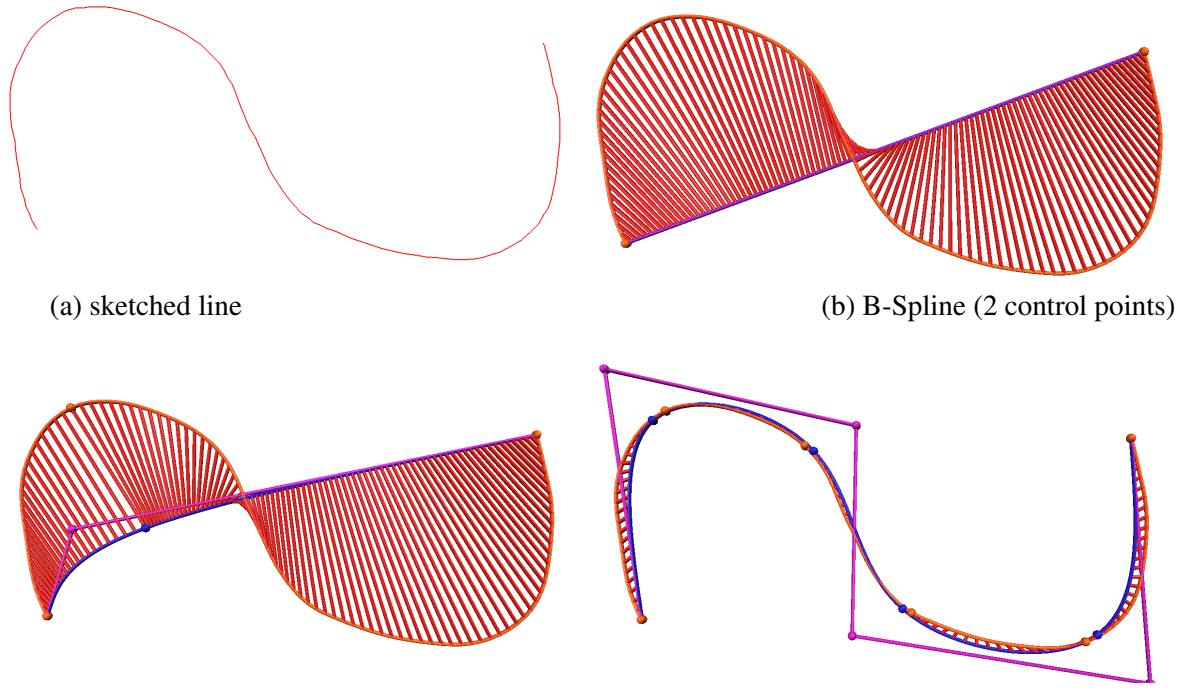
2.4.1. Extraction of a Procedural Model

To extract a procedural model we have to include our procedure generation directly into the 3D sketch-based modeling process. We first describe the relevant operations of the modeling tool, as this is important to understand how we create our procedures. We also describe the sketched line processing and the surface boundary representation used by the modeling tool.

The sketched line processing: The modeling system by Bein et al. [BHSF09] offers sketch-based modeling operations. Hence, the system needs to process a sketched line. Figure 2.4.2 shows the sketched line processing. Initially a B-Spline with 2 control points is constructed which interpolates the start and the end point of the line. Then the B-Spline control points are iteratively incremented until the distance between the curve of the B-Spline to the sketched curve is lower than a predefined ε. Note that the iterative approximation includes steps to define if the control points are tagged smooth or sharp. Hence, the output is a sequence of control points with sharpness tags.

The boundary representation and the modeling view: The boundary representation generated by the GML language (combined B-Rep [HF01]) is a control mesh where each edge is tagged as smooth or sharp. The final output is the subdivision surface of the mesh. In Figure 2.4.3 (a) we can see the boundary representation as green lines (smooth) and red lines (sharp) as well as the resulting subdivision surface. The tessellation of the subdivision surface is shown in Figure 2.4.3 (b).

The modeling view is shown in Figure 2.4.3 (c) and (d). During the modeling the user sees the resulting subdivision surface and the edges, vertices and faces of the control mesh. The faces are



(a) sketched line

(b) B-Spline (2 control points)

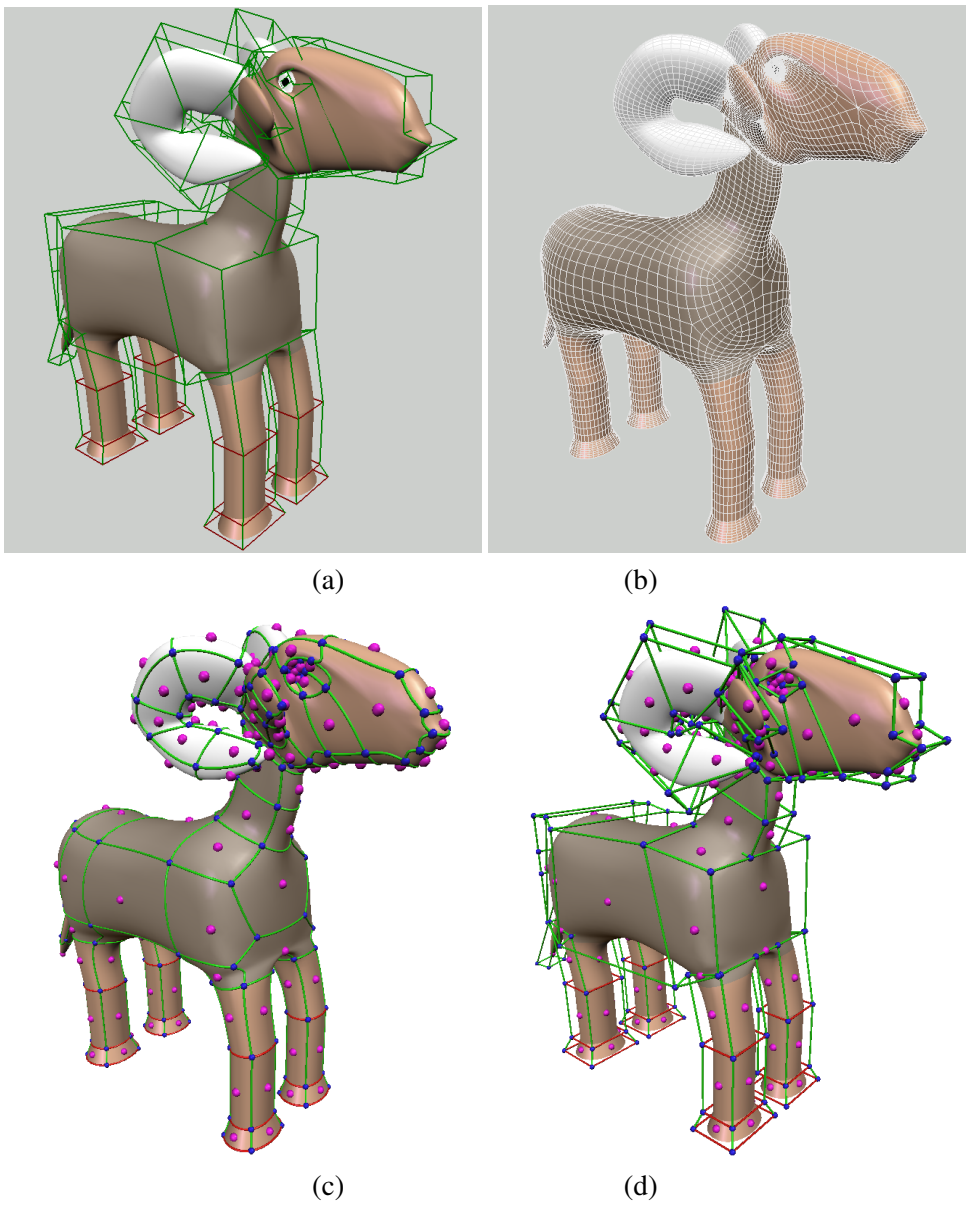
(c) B-Spline (3 control points)

(d) B-Spline (6 control points)

Figure 2.4.2.: The sketched line is translated to a sequence of control points by refining a B-Spline until the distance of the curve of the B-Spline (blue) to the sketched line (orange) is lower than a predefined ε.

represented by pink balls, the vertices by blue balls and the edges as green lines (smooth) or red lines (sharp). The user can manipulate the control mesh by clicking on the faces, vertices or edges of the control mesh. The different modeling operations are evoked by these actions. For example, a left click on a face activates the sketching extrude operation, which means that the user can sketch a line and the face is extruded on this path.

The modeling operations: To capture each modeling operation we need to define a set of variables for each operation that completely describe the specific operation. An important propriety, which we identified, is the **local relativity**. We define all variable-sets in a way that the values are relative to the position of the included faces, vertices or edges. This is important, as the position and orientation may change if we vary a preceding operation. For this reason we do not use any vector (x, y, z) dependent on the global coordinate system. We specify all spatial position changes as cylindrical coordinates (ρ, ϕ, z). Where ρ is the radial distance, ϕ the rotation angle, and z the height. The origin of the local system is the face midpoint, vertex position or edge midpoint. The cylinder axis is oriented according to the surface normal of the face, vertex or edge.



(a) (b)

(c) (d)

Figure 2.4.3.: The images show the used boundary representation as (a) the resulting control mesh and subdivision surface and (b) the resulting tessellation of the surface. (c) The view of the modeling tool with the control mesh projected onto the surface and (d) with the original control mesh. In (c) and (d) The pink balls represent the faces, the blue balls represent the vertices, the red lines represent the sharp edges and the green lines represent the smooth edges.

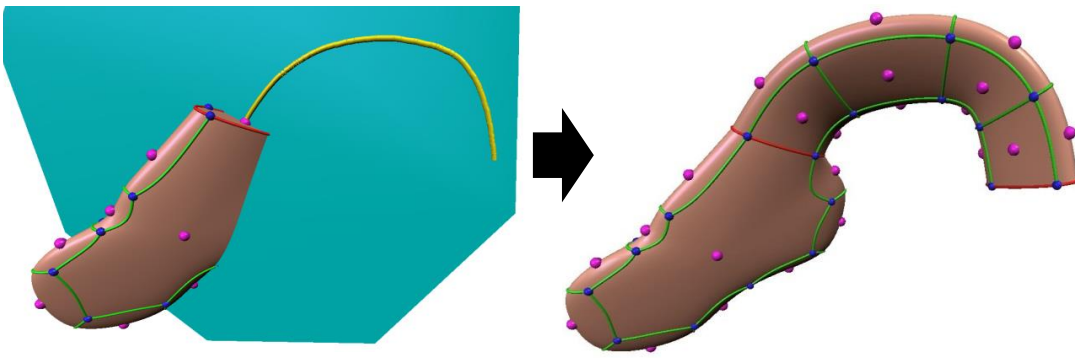


Figure 2.4.4.: A sketching extrude operation. On the left we see the sketching and on the right we see the resulting surface. The yellow line is the sketched curve for the extrusion and starts at the middle point of the face. The blue plane going through the 3D surface is shown during the sketching operation. The plane is calculated from the face midpoint, the face normal and the cross product of the viewing direction and the face normal.

Sketching - Extrude: In Figure 2.4.4 we can see a sketching extrude operation. This operation always starts at a single face mid point. For a better orientation in 3D a sketching plane (blue) is shown during the sketching. The line is sketched onto the sketching plane. The plane is constructed by the face normal, face midpoint and the cross product of the viewing direction and face normal.

If we look at the example shown in Figure 2.4.4, we can see that the sketching extrude operation consists of multiple face extrusions. In this example the face is extruded 4 times. As described before (See 'The sketched line processing'), the result of the sketching is a sequence of control points. Hence, a face extrusion is performed for each control point. The position of the control points marks the position of the subsequent face midpoint. The width of the face is defined by the width of the initial face.

For the sketching extrude operation we define the following variables:

$$([ID], [(\rho_1, \phi_1, z_1), (\rho_2, \phi_2, z_2), ...,], [\alpha_1, \alpha_2, ...], [s_1, s_2, ...,], [(\rho_n, \phi_n, z_n)])$$

ID is the id of the selected face. For each face extrusion we have the position of the control point relative to the preceding control point (ρ, ϕ, z) and the angle of the surface normal α after the extrusion (relative to the previous surface normal) and the smoothness of the control point s (0 or 1). Note that these 5 variables ρ, ϕ, z, α and s are needed for each control point of the extrusion path. The number of variables is defined by the number of control points. For the whole operation we also need (ρ_n, ϕ_n, z_n), which gives us the rotation normal. Together with the face normal and α the rotation is well defined.

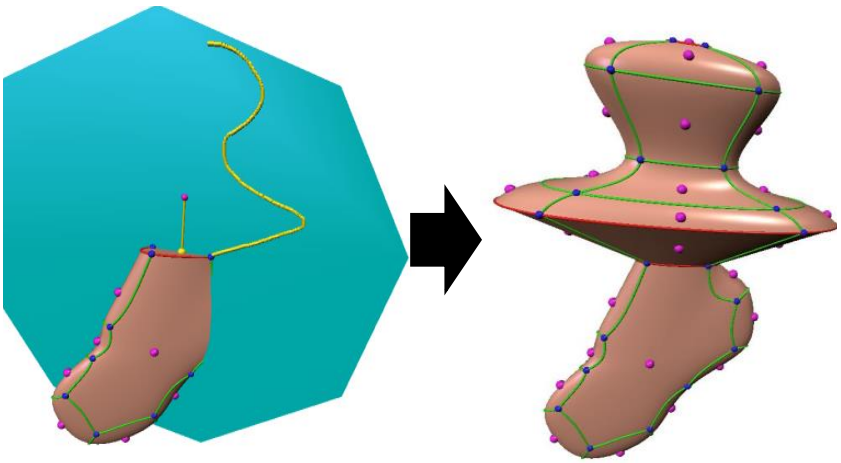


Figure 2.4.5.: The sketching rotation-extrude operation. On the left we see the sketching and on the right we see the resulting surface. The face which was selected for the operation is indicated by the yellow pin with a pink ball on top. The sketching starts at a vertex of the chosen face.

Sketching - Rotation-Extrude: The second possible sketching operation is the sketching rotation-extrude operation. It is a special case of the sketching extrude operation. As shown in Figure 2.4.5, this operation is also done on a single face, but in contrast to the normal sketching extrude operation the rotation-extrude starts at a vertex of the selected face.

The example shows that the rotation-extrude operation also consists of several individual extrusions. In this example we can see 5 face extrusions. A major difference is that the rotation-extrude is fixed in the direction of the extrusions but not fixed in the width. Each extrusion has the direction of the surface normal of the selected face. Therefore, we do not need the vector $\mathbf{n}$ or the angles α or the distance vector (ρ, ϕ, z). We only need the width and the length of each extrusion. This results in a very different variable-set for the rotation-extrude operation:

$$([ID], [w_1, w_2, ...,], [l_1, l_2, ...,], [s_1, s_2, ...,])$$

ID is the id of the selected face, w is the width of each individual extrusion and l is the distance to the preceding control point. s is the sharpness of the control point.

Drag: This operation allows dragging a single vertex, an edge or a face. The only variable we need for the drag operations are the following:

$$([ID], [(\rho, \phi, z)])$$

With *ID* being the face, vertex or edge id and (ρ, ϕ, z) being the local relative displacement as cylindrical coordinates.

Scale: This operation allows to resize a face. We need the face id *ID* and the relative size of the scaling σ:

$$([ID], [\sigma])$$

Rotate: This operation allows to rotate a face. We need the variable for the face id *ID* and the rotation angle α as well as the rotation normal defined by the relative offset (ρ_n, ϕ_n, z_n):

$$([ID], [\alpha], [(\rho_n, \phi_n, z_n)]])$$

Insert Vertex: The insert vertex operation allows to insert a new vertex on an edge. For the preservation of our local relativity concept, we do not use the coordinates of the inserted vertex, but only the relative positioning as barycentric coordinates on the edge:

$$([ID], [\lambda_1])$$

ID is the id of the edge. λ_1 is the first barycentric coordinate and the second barycentric coordinate is $\lambda_2 = 1 - \lambda_1$.

Paint: A face is painted with a chosen color. We need the the face id *ID* and the chosen color as RGB values:

$$([ID], [(r, g, b)])$$

Flag, Connect, Remove: The flag operation allows to flag a single edge as smooth or sharp. It is also possible to flag a vertex or face, which causes all adjacent edges to get flagged as smooth or sharp. With the connect operation it is possible to connect two vertices or two faces with each other. Connecting two vertices creates a new edge. Connecting two faces creates a bridge between the faces. The remove operation removes a vertex, edge or face. For the flag and remove operations we only need the id $([ID])$ of the vertex, edge or face. For the connection operations we need the two involved ids of the vertices or faces $([ID_1, ID_2])$.

2.4.2. Parameter Insertion

During the modeling process a procedural model is automatically extracted. The procedural model consists of all executed modeling operations with the according values for the variables of the operations. When the procedural model is executed the resulting object exactly corresponds to the modeled object. Now we want to define meaningful parameters which vary the construction algorithm defined by the procedural model and therefore vary the resulting 3D object. The parameterized procedural model makes it possible to generate different 3D objects by changing the parameters.

An inserted parameter influences the values of several modeling operations at once. In the end, we have a small list of parameters $(p_0, p_1, ...)$ and the variation of these parameters cause meaningful changes to the resulting object.

In the following we explain our principles for the insertion of parameters and then describe our insertion techniques. The principles ensure that the variation of the parameters lead to consistent results. For each inserted parameter the user can choose one of the three insertion techniques.

The parameter insertion principles:

Multiple insertions: We aim at creating a small set of parameters. For this reason each parameter can be inserted multiple times on different operations. Several operations are therefore varied with a single parameter. For example several extrude operations can be scaled with a single parameter.

Local relativity: Generally, we have to rely on the local relativity of the parameters, so that the variation of parameters in an operation does not evoke undesired changes to other subsequent operations.

Fixed Range: The variation of the resulting object should not be arbitrary as all possible variations should still be connected to the semantic of the defined class. For this reason all parameters have a fixed range of possible values.

Continuous parameters: We generally avoid discrete parameters and only use continuous parameters. A discrete parameter variation is difficult to define, as the range limits are usually unclear. For example, if we choose a face id of an operation as a parameter, it is unclear which other face ids would make sense to insert into this parameter. For this reason we do not change *ids* with parameters. Therefore, especially the flag, connect and remove operations are not used in parameters.

The techniques for parameter insertion:

Exchange: A variable of an operation is directly exchanged by a parameter. The value of the parameter is set as default value of the parameter.

Example: The Operation 'Insert Vertex' has the values $(5, 0.37)$ for the variables (ID, λ_1). We exchange 0.37 by the parameters p_0:

$$(5, 0.37) \rightarrow (5, p_0) \; with \; p_0 = 0.37$$

Multiplication: A variable of an operation is multiplied with a parameter. The default value of the parameter is 1.0.

Example: The Operation 'Sketching - Extrude' has the values $(1.2, 30, 0.3)$ for the cylindrical coordinates of the first extrusion (ρ_1, ϕ_1, z_1). We multiply ρ_1 and z_1 with the parameter:

$$(1.2, 30, 0.3) \rightarrow (1.2 \cdot p_0, 30, 0.3 \cdot p_0) \; with \; p_0 = 1.0$$

Interpolation: The parameter $p_0 \in [0, 1]$ interpolates between the original value of a variable of an operation and a new chosen value. The default value is 1.0.

Example: The Operation 'Sketching - Extrude' has the values $(1.8, 60, 0.8)$ for the cylindrical coordinates of the first extrusion (ρ_1, ϕ_1, z_1). We interpolate ϕ_1 from the original 60 to 180 degrees:

$$(1.8, 60, 0.8) \rightarrow (1.8, 60 \cdot p_0 + 180 \cdot (1 - p_0), 0.8) \; with \; p_0 = 1.0$$

Parameter variations within the modeling operations:

Sketching - Extrude: This operation is our main operation to introduce meaningful parameters. We can change the size of the extrusions by inserting parameters to ρ and z_1 and we can vary the direction of the extrusions in one direction by inserting a parameter to the angle ϕ. We can also change the direction by using the angle α of the rotation of the final face after an extrusion. Only changing the radial distance ρ also leads to a direction change.

Sketching - Rotation-Extrude: The rotation extrude can directly be varied by inserted parameters for the width w and length l of every single extrusion. For w and l different parameters can be inserted or a single parameter can vary both at once, so that the total size of the extrusion is varied.

Drag: A drag operation is suitable to define a variation. It offers ρ, ϕ and z and therefore can include any variation of the length and direction.

Scale: We can directly include a parameter for the scaling value σ. This can be very useful if a face is scaled before a sketch-extrude operation is used, because the size of the extruded faces dependents on the size of the initial face.

Rotate: The rotation angle α can be used for a parameter. Hence, the direction of a face can be varied before an extrusion is applied.

Insert Vertex: This operation is very useful for the insertion of parameters, as the barycentric coordinate λ_1 is very easy and intuitive to vary. The variation can either be used for a single vertex to define the width of a branching part or it can be used for multiple vertices of the same branching part of an object (e.g. a wing of a plane), so that the position of the branch within the face is changed.

Paint: The color can be varied by exchanging the (r, g, b) values with a parameter.

Flag, Connect, Remove: As we avoid discrete variations and especially variations of IDs, we do not vary these operations.

Parameter insertion interface: Our extracted procedural model corresponds to program code. Reviewing the complete code and manually inserting the parameters would be tedious. Hence, we include a simple user interface (See Figure 2.4.6) for the insertion of the parameters.

The user can select an operation from the list of operations. All variables of the operation and their values are then shown. The user can exchange any variable by clicking on the value and writing a name. The name is automatically mapped to a parameter with the defined name. If the parameter does not exist, a new one is generated. Then, the user can switch to the parameter view, where all parameters are listed. For each parameter the user can choose between the exchange, multiplication and interpolation technique and has to set the value range. The user interface does not only ease the insertion of parameters, we hereby also enforce that all insertions correspond to our principles and insertion techniques.

Additionally, the chosen operation is always highlighted on the resulting mesh (See Figure 2.4.7), so that the user knows which operation influences which faces.

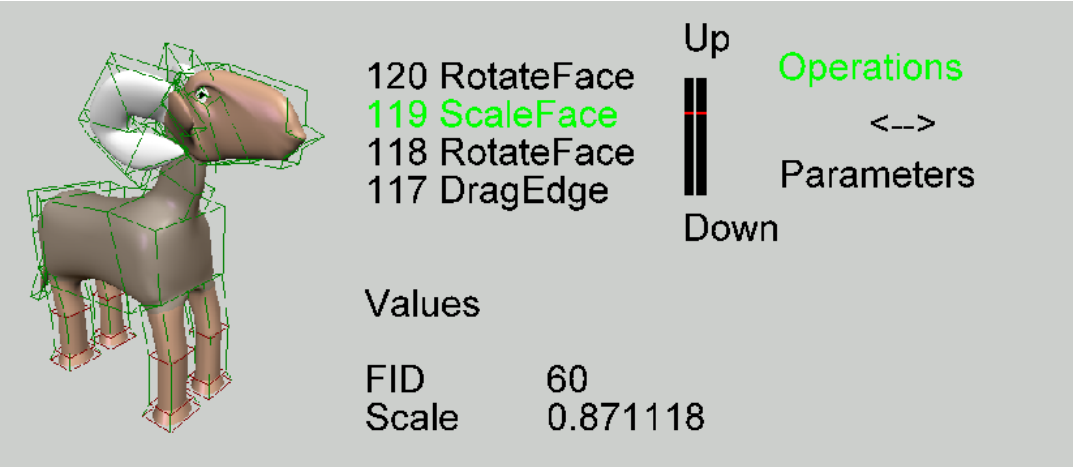


Figure 2.4.6.: The GUI to insert and edit the parameters.

2.4.3. Results and Discussion

In this section we present results and show how the parameter variation produces variations of objects. Then we discuss the further possibilities and limitations of our approach.

We show the possibilities of our proposed procedural modeling approach on two basic shapes: a quadruped and a biplane. By solely changing the parameters we obtain the 7 different resulting objects of the modeled quadruped and the 5 different objects of the modeled biplane in Figure 2.4.8.

We modeled the biplane and the quadruped with the sketch-based modeling tool from Bein et al. [BHSF09]. With the automatically extracted procedural model we inserted parameters with the insertion interface. The biplane procedural model includes 8 parameters for the following properties: The wings length, size and angle, the length of the plane, the back-wings size, the plane-nose length and width and the size of the wheels. The quadruped procedural model has 15 parameters: The body size and length, the neck length and direction, the length of the snout, the tail length, the leg size and length, the ear direction, size and length and the horn direction, size and length. Additionally, 3 parameters for the color of the model: one for the color of the body, one for the horns and one for the color of the residual model.

In the following we first discuss the resulting objects generally and then we take a closer look at the single variation possibilities to show how the parameters can be inserted and used for the variation.

In Figure 2.4.8 the first visible variation is the variation of the color. Each used color can be exchanged by a parameter and therefore we can vary the color value. The more interesting variations are

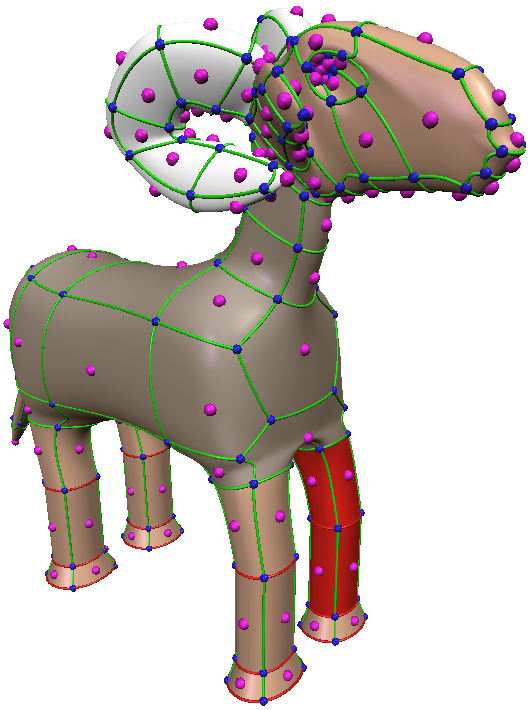


Figure 2.4.7.: For the insertion of parameters the user wants to know which operations influence which parts of the surface. In the shown case a single sketch extrude operation is highlighted in red.

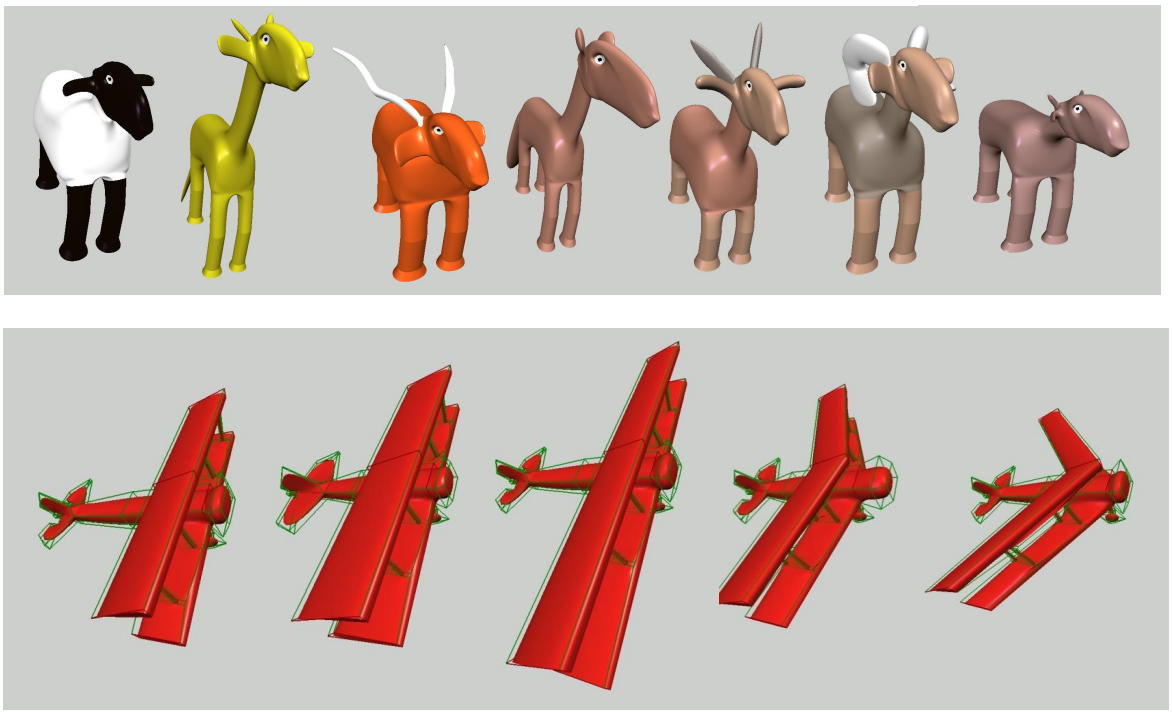


Figure 2.4.8.: The procedural model allows to introduce parameters into the object construction. Changing the parameters of the construction changes the surface of the resulting 3D object.

visible at the head of the quadruped. Not only the size of the head differs from one object to another, but also the width, length and form of the horns and of the ears. The horns can also be reduced to the size of zero. All other parts (legs, chest, tail, neck) are also varied in size.

Figure 2.4.8 also shows variations of a biplane object. We see that parameters control the length, width and direction of the wings. The size of the smaller wings at the back of the plane are also changing. Furthermore, the length of the whole plane as well as the length and width of the nose of the plane are varied.

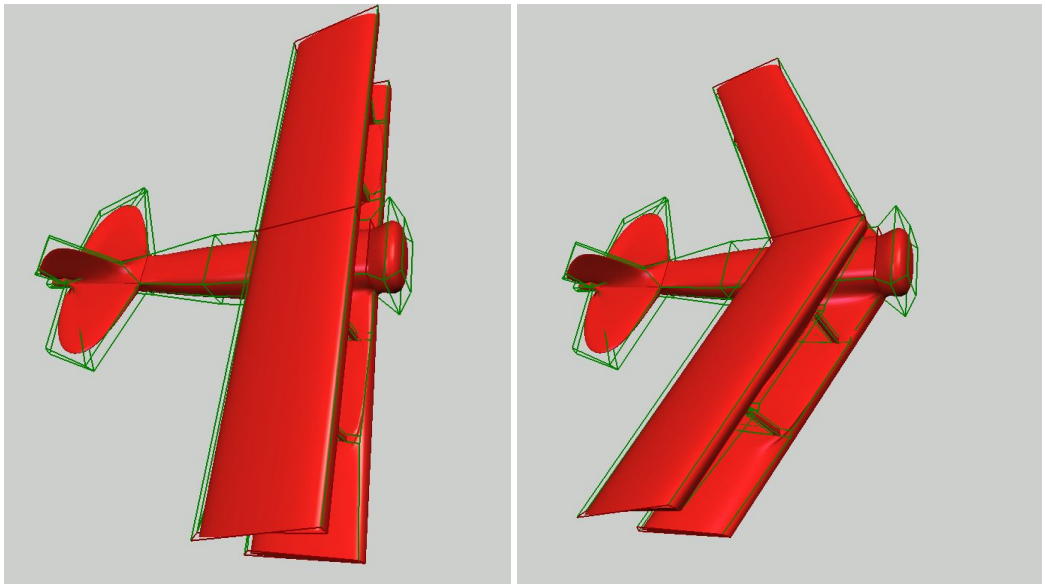


Figure 2.4.9.: The wings are defined by sketching extrude operations. By varying the radial distance ρ of these operations we can change the angle of the wings.

The variation of the angle of the wings, shown in Figure 2.4.9, is produced by inserting a parameter which is multiplied with the cylindrical coordinate radial distance ρ for all extrusion operations of all 4 wings. Note that the bars connecting the upper wing with the lower wing, are not explicitly changed with the parameter. Thanks to the principle of local relativity all modeling operations related to these bars still produce the expected result: The bars stay at their position on the wings, even if the angle of the wing is changed.

Figure 2.4.10 shows the variation of the front part of the biplane. This part was modeled with sketching rotation-extrude operation consisting of two face extrudes. Here we use two parameters: one for the length and one for the width: the first parameter is multiplied with the width of the first extrusion w_1. The second parameter is multiplied with the length l_1 and l_2 of both extrusions.

A parameter can also be included in several different operations. The example of Figure 2.4.11 shows the wheels of the biplane. The wheels are constructed with sketching extrude and sketching rotation-extrude operations. A single parameter controls the size of the wheels. The parameter is

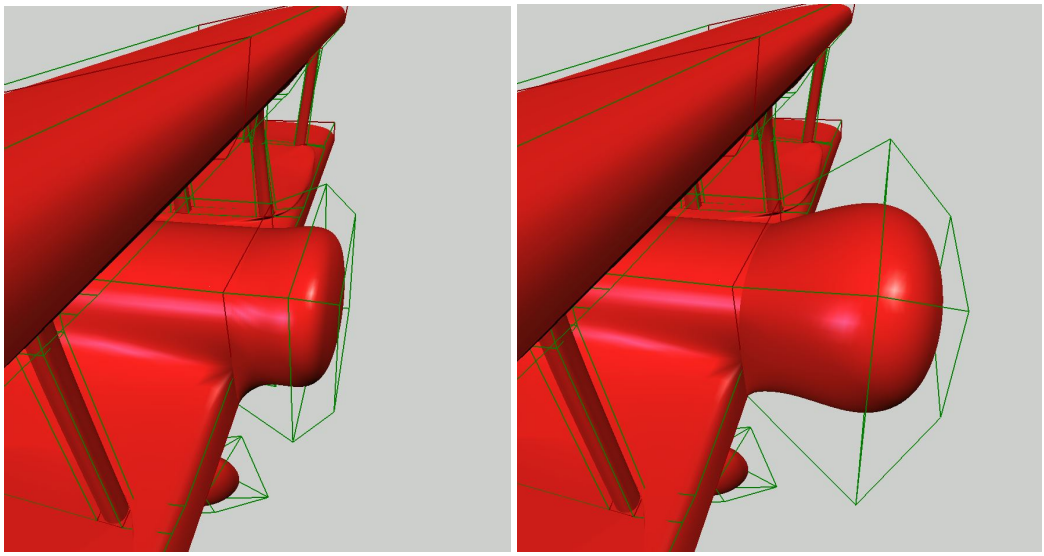


Figure 2.4.10.: Variations of the width and length variables of the sketching rotation-extrude operation lead to differently formed front parts of the biplane

multiplied with the length l and width w of the extrusion operations and with ρ, z and α of all extrusions of the sketching extrude operations. Allowing the extrusions to become zero is a setup that allows a part of the object to appear and disappear without including if-then-else branches into the code. These branches are problematic as the subsequent operations rely on a given topology of the mesh. If topology changing operations are simply skipped without replacement the subsequent operations may not work as expected. Additionally a boolean parameter would also contradict the continuous parameter principle.

The examples already have shown that inserting parameters into the two sketching operations lead to meaningful and easy to use parameters. It is also notable that the insertion of the parameters to the sketching operations does not have to be considered during the modeling. The variation with parameters can be considered and tested afterwards.

The drag operation differs in this characteristic. Figure 2.4.12 shows the result of a variation using drag operations. We dragged all edges of the front part of the head, so that it gets longer. Then we insert a parameter, which exchanges the length of the cylindrical coordinates ρ and z of all drag operations. Hence, all drags are sized equally according to the parameter. The variation works very well, however there is a difference between the variation through drag operations and the variation through sketching operations. The difference is that we executed new drag operations for the purpose of inserting new parameters, after thinking of the variation we want to produce. Hence, a limitation we identified is that the drag operation often has to be added afterwards for the use of parameters. Of course, already existent drag operations can also be used for variations.

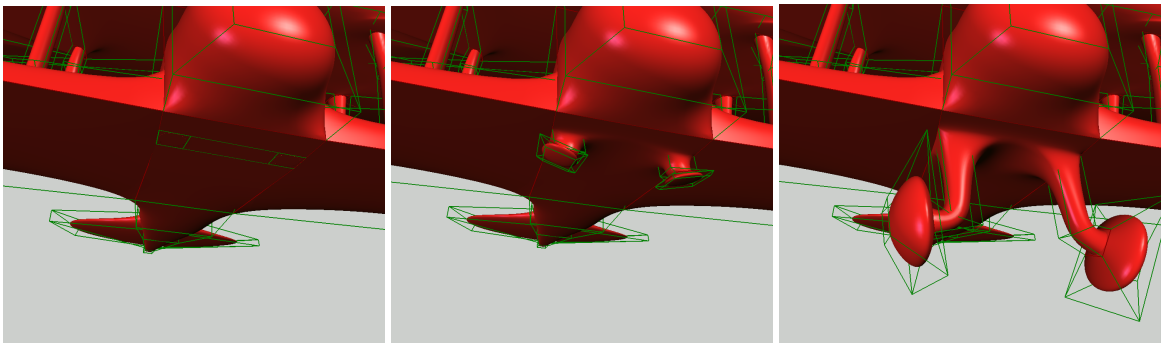


Figure 2.4.11.: The images show the resulting object for variations of the parameter of the wheels. The parameter is multiplied with l,w,ρ,z and α of all used extrusions.

Note that the scale and rotate operations work similarly. We can use existent operations in the procedural model but we can also execute new operations to include them in the procedural model and use them for the parameters and produce meaningful changes.

The insert vertex operation is usually used when a face is divided into smaller faces like it has to be done when adding legs to the chest of the quadruped (See Figure 2.4.13). Hence, the insert vertex operation appears naturally and does not have the problem that the drag operations has. Like the drag and scale operations it can also be used to define a variation of the size of a part of the object. This is the case because the size of the extrusions are dependent on the size of the extruded face. The insertion of a vertex on the edge decides how big the new face will be. In Figure 2.4.13 we can see such a variation with insert vertex operations. The inserted vertices are marked with red circles. We exchange λ_1 of the insert vertex operations of all legs with a parameter. The variation of the parameter changes the size of the whole legs.

Figure 2.4.14 shows the variation of the direction of the ear of the quadruped. The ear extrusion consists of a single sketching extrude operation with two face extrusions. We insert a parameter for the interpolation of the cylindrical coordinate angle ϕ, the radial distance ρ and also for the interpolation of the face rotation angle α for both extrusion operations. The result is an upwards rotation of the whole ear when the parameter is changed from 1.0 to 0.0.

Figure 2.4.15 shows a variation of the horns of the quadruped. We used a scaling of the base face and exchanged σ of the scale operations by a parameter defining the thickness of the horns. Then we used sketching extrude operations for the curved form of the horn. The second parameter is multiplied with ρ and z of all sketching extrude operations and therefore defines the length of all extrusions. The third parameter interpolates the rotation angle α from the original value to 0 and the radial distance from the original value to 0. When α and ρ are 0, the extrusion direction equals the face normal direction. Hence, the third parameter interpolates between the curved horn and the straight horn.

There are several possibilities of introducing variations with the insertion of parameters. The best operation for the variations is the sketching extrude operation as it offers many possibilities for easy to

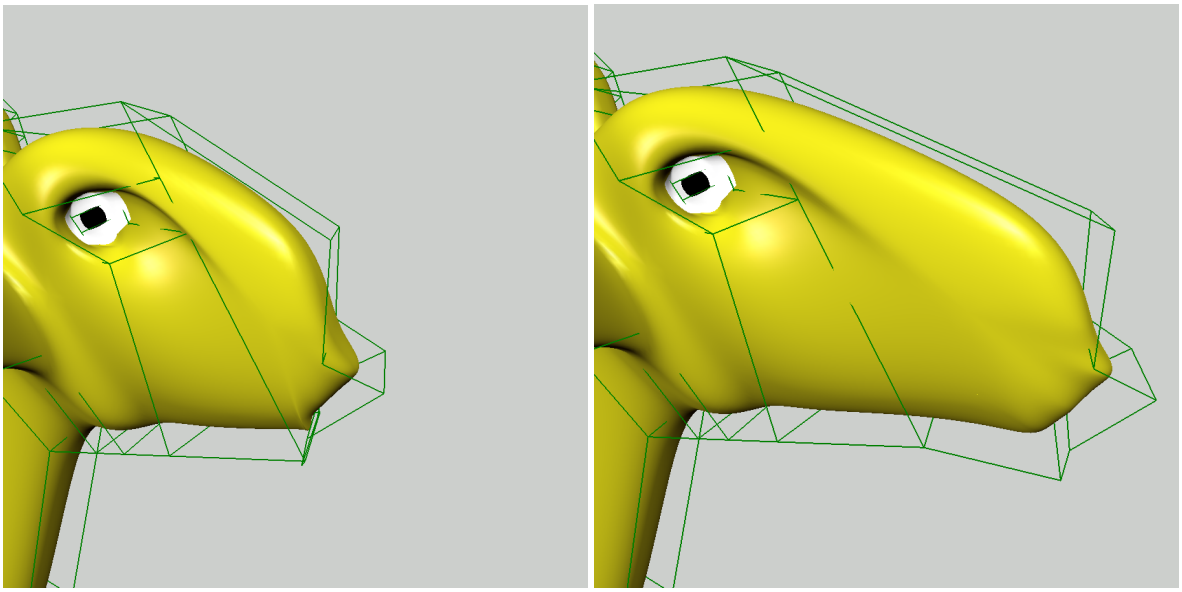


Figure 2.4.12.: The length of the head is varied with a few drag operations. The drag operations were used on all edges of the front part of the head. The inserted parameter exchanges all ρ and z of the drag operations, so that all drags are sized equally.

use parameters and the insertion of the parameters is always possible in the sketching extrude operation. The sketching rotation-extrude operation offers lower diversity as the direction of the extrusion cannot be changed but also offers easy ways of inserting parameters for meaningful variations. The drag, scale and rotate operations can easily be varied with parameters and also offer good possibilities of producing variations. However, these operations have the drawback that they are not always naturally present and in some cases they have to be additionally added to insert parameters. The insert vertex operation offers an alternative, since this operation is always used and can also change the scale and position of parts of the object.

Summarizing, semi-automatically created flexible procedural models like the shown quadruped have a high potential. However, our automatically created procedural model code is less efficient than manually created code, since it is linearly dependent on the number of used modeling operations. This could be further optimized, e.g. subsequent drag operations on the same face could be merged to a single drag operation. Nevertheless, the automatic creation of the procedural model and the semi-automatic insertion of parameters into the procedural model makes it possible to directly model a complex procedural parametric model.

A hybrid modeling is also possible with our approach. Hybrid means that the result of a normally modeled object is combined with our procedural modeling approach. The user starts with an initial model and all subsequent modeling operations are parameterized and available for parameter insertion.

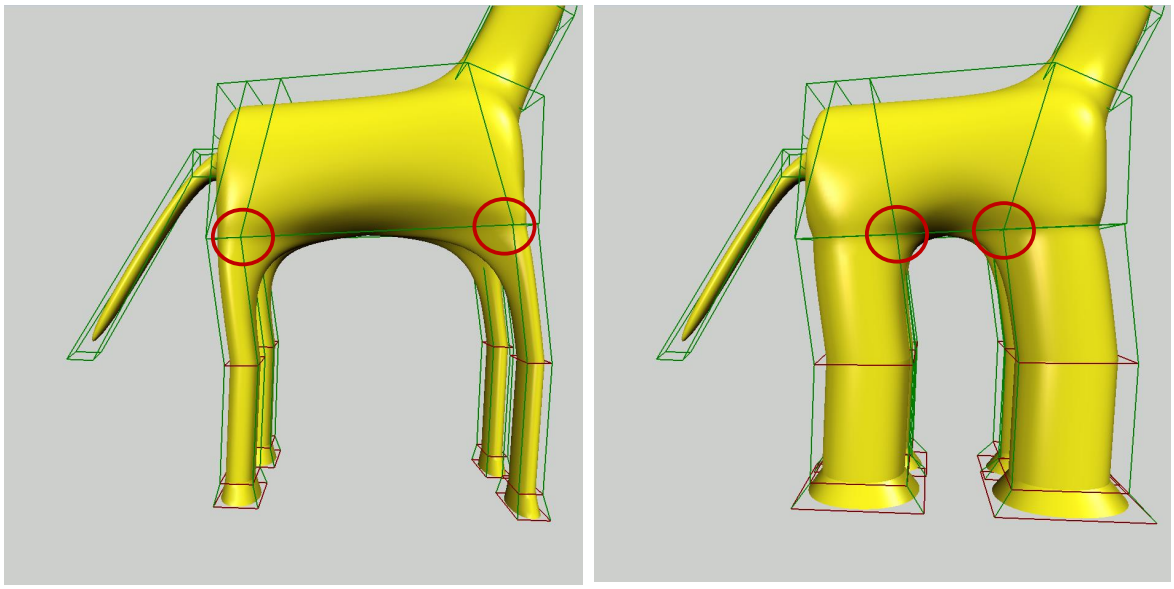


Figure 2.4.13.: The size of the legs is varied through a variation of λ_1 of the insert vertex operations used for the construction of the legs. The varied inserted vertices are marked with red circles.

It is also possible to reuse a procedural model to define new procedural models of related types. As our examples show we aim at variations which do not completely change the type of object. Hence, a procedural model could be understood as a class representation e.g. a 'quadruped' class or 'biplane' class. If much bigger changes are desired there is a straightforward way to do so by reusing a procedural model: An initial procedural model (with parameters) can serve as the basic model (e.g. a 'plane') and then each class specification could be modeled with the basic model as starting point. Hence, a procedural model for a 'biplane', a 'fighter-jet' and a 'commercial plane' could be created from an initial 'plane' model. The parameters of the 'plane' would be available in all models, but additional parameters could also be defined for every model.

The procedural model description is powerful, flexible and also is comparably fast and easy to create with a modeling tool (compared to the tedious process of programming a procedural model from scratch). Still the process of procedural model creation is non automatic: It does involve the modeling with a tool and the parameter insertion. Even though both are simple the user first needs to get used to the operations and the parameterization process. This process offers the user a high amount of control. Still, it is important to give the user more automated possibilities of inserting parameters. This problem is tackled in the next sections 2.5 and 2.6.

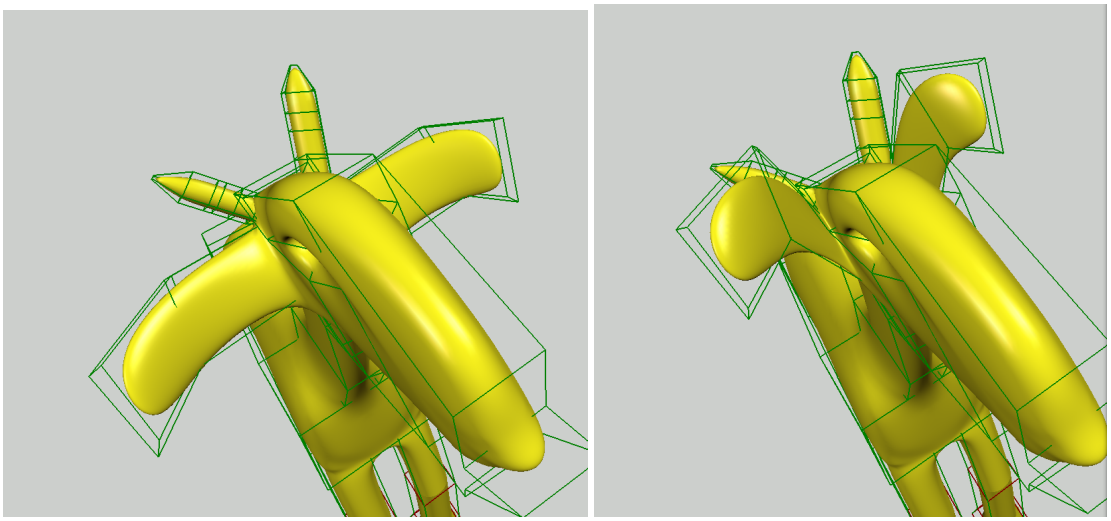


Figure 2.4.14.: The direction of the ear can be changed by a variation of the angles ϕ and α together with the radial distance ρ of the included sketching extrude operations.

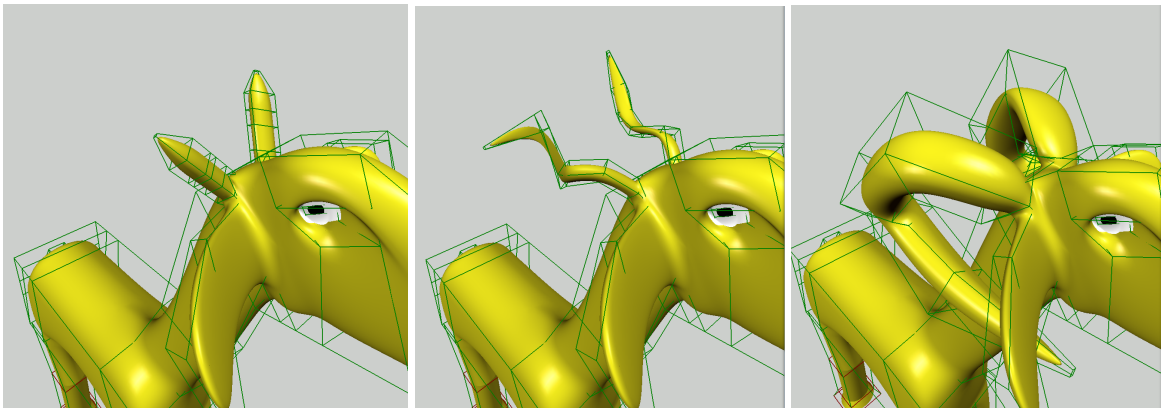
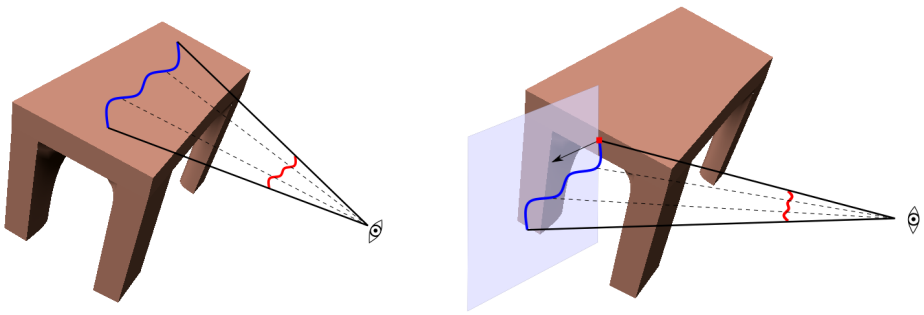


Figure 2.4.15.: The variation of the horns includes 3 parameters for the length, thickness and direction of the horns. The length changes ρ and z of the sketching extrude operations. The thickness is changed with the variable σ of a scale operation of the base face used for the horn extrusions. For the direction we change ρ and α of the extrusions

2.5. Complex Parameter Sketching

The sketch-based modeling system for semi-automatic procedural model generation allows the manual insertion of parameters through an interface. The user can choose between several insertion principles. Even though, the user has a high amount of control with the manual insertion with insertion principles, the effort for defining complex parameters is present. To further automate this process we propose the sketching of complex parameters. Instead of manually choosing procedures for the insertion of parameters, the user defines the region and the parameter influence with a sketch-based interface.

First, the user has to define the region of the object, which should be influenced by the parameter. For this step we segment the mesh into different parts. The user can refine the segmentation with additional sketch interactions. Then, the user sketches the path of a deformation of the object. Our system automatically calculates the deformation and identifies all procedures within the procedural model which are relevant for this deformation. Finally, a parameter is generated and inserted into the procedural model which varies the procedures, so that the sketched deformation is imitated by the parameter variation of the procedural model.



(a) (b)

Figure 2.5.1.: (a) The sketched line is projected onto the object. (b) the sketched line is projected on the minimal-skew viewplane.

For the interpretation of the sketched lines we use two different projections in the system. Figure 2.5.1 shows the two possible projections. The first possibility shown in (a) directly projects the sketched line onto the object by casting it into the viewing direction. This type of sketch is used to sketch on the surface of the object to refine the segmentations. The second possibility is presented in Figure 2.5.1

(b). The sketched line is projected on the minimal-skew viewplane. This projection is proposed by De Paoli et al. [DS15]. This type of projection is used for the sketch of the deformation of the object. Since the user always starts the deformation on the mesh we can use the surface normal of the object at the initial point of the sketch.

2.5.1. Preprocess

For the deformation we use the polygon mesh resulting from the procedural model with default parameter settings. Since this mesh is the result of a subdivision surface algorithm the tessellation generally offers enough degrees of freedom for the following deformation algorithm.

However, when all edges of a control polygon are marked as sharp the polygon is only separated into triangles and not subdivided further. This can result in large triangles not offering enough degrees of freedom. For this reason we include a preprocess refining all these triangles. For this process we need to find all large triangles and refine these. We use the refinement of [BK04], which takes a set of triangles as input.

For the selection of large triangles we choose a common technique used to detect outliers in explorative data analysis [Pag13]. We calculate the interquartile range (IQR) between the lower ($\frac{n}{4}$th element) and the upper quartile ($\frac{3n}{4}$th element). Then we apply the following heuristic:

$$D_{out} := \{D \mid D > Q_3 + 1.5 \cdot IQR\}. \tag{2.5.1}$$

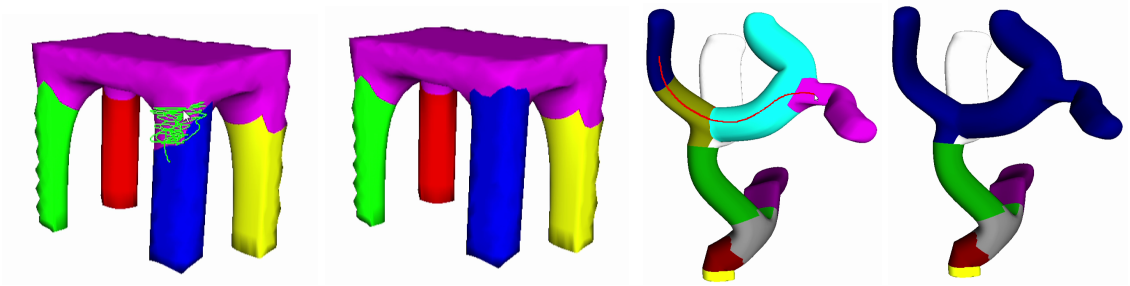
Note that we keep the correspondences between the refined mesh and the unrefined mesh for later steps.

2.5.2. Mesh Segmentation

To maximize the automation of our system we choose a deformation algorithm, which directly works on the polygon mesh. The user does not have to set up any control structure for the deformation. However, for proper deformations we need to set the fixed vertices for a deformation. These vertices stay at their position during the deformation. Letting the user manually define the fixed vertices for a single deformation would require too much effort. Therefore, we included a mesh segmentation and automatically set all vertices to fixed vertices, which are not part of the segment, which is deformed by the user.

We use the segmentation algorithm of Shapira et al. [SSCO08], since this algorithm is fully automated. Nevertheless, the number of segments and their extent is not always optimal for the deformation desired by the user. Therefore we choose a liberal parameter setting for the segmentation algorithm to provide a rather fine initial segmentation to the user. Then, we provide the user a simple sketch inter-

action to merge segments or even extent single segment. With these interactions the user can define the area which should be deformed with a few sketches.



(a) (b)

Figure 2.5.2.: (a) The green sketch lines define where the blue segment is extended (b) all segments hit by the red sketch line are merged.

The two interactions are shown in Figure 2.5.2. The extension of a segment is performed by starting a sketch on a segment and drawing over the area which should be included in this segment. The merging of segments is performed by starting on a segment and drawing a line over all segments that should be merged with the former segment.

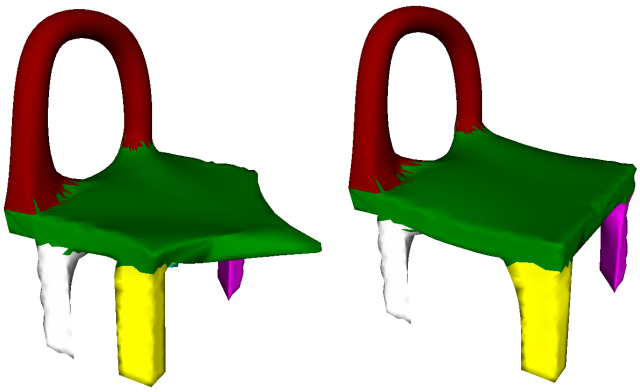
2.5.3. Mesh Deformation

For the deformation we choose the as-rigid-as-possible deformation of Sorkine et al. [SA07]. The deformation is applied to the polygon mesh and during the deformation we save one intermediate deformed mesh per sketch point. This sequence of deformation meshes provide us the target function for our parameter of the procedural model. In the following stages we identify suitable parameters to imitate the deformation sequence with parameter changes.

For the deformation the algorithm needs fixed vertices and handle vertices. All other vertices of the mesh are used for the deformation. The fixed vertices are provided by the segmentation. For the handle vertices we provide three different setups, which can be chosen by the user.

In the first setting the handle points are the three vertices of the triangle picked by the user. This is a very local deformation and might not always be the deformation desired by the user. Figure 2.5.3 shows a case where the user intended to stretch the whole chair instead of just stretching a single point of the front of the chair.

For this reason we provide a second deformation interaction: the segment-spanning deformation. In this deformation the handle points are spread out until a discontinuity in the face is detected. Additionally, the spreading is not limited to the initial segment. Therefore additional segments can be added to the initial segment during the process. This is also shown in Figure 2.5.3.



(a) (b)

Figure 2.5.3.: (a) The chair is deformed with the standard deformation handle. (b) The chair is deformed with the segment-spanning handle.

The third possible deformation interaction is the segment-bounded deformation and is similar to the segment-spanning deformation, with the difference that the spreading is limited to the initial segment.

2.5.4. Parameter Generation

For the generation of parameters we have to identify the parameter setting imitating the deformation process. The resulting parameter has the range of $[0, 1]$, with 0 being the setting of the initial mesh and 1 being the setting of the final deformed mesh. In between, the intermediate deformed meshes are approximated as closely as possible.

Procedure identification: In the first step of the parameter generation we have to identify which procedures of the procedural model are relevant for the deformation. We identify all relevant procedures by checking which procedures generated the vertices, which are deformed in the final mesh. For this identification, we execute every procedure of the procedural model step by step and check whether the vertices of the final mesh are already present in this step or not.

Instead of checking every single deformed vertex we create a reduced set of displaced points V. Empirically we identified that taking a much smaller set results in a relevant runtime improvement with no noticeable accuracy drawback. In all experiments the same set of procedures was identified with the highly reduced set.

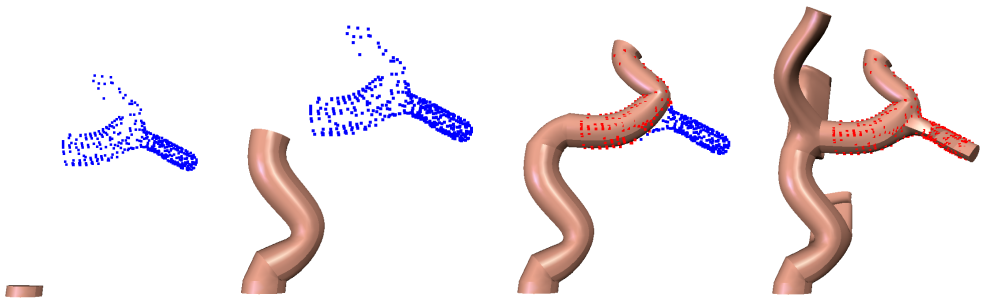
Therefore, the reduced set V contains all characteristics deformation vertices. We form the set by taking all deformed vertices (all vertices that are not fixed vertices or handle vertices) and apply two filters: First, we filter out all vertices not belonging to the original unrefined mesh. Then, we filter out

vertices with similar deformation offset vectors. We only take the top 20% of vertices with distinct deformation offset vectors. The algorithm of the generation of the set is shown in Algorithm 1.

Algorithm 1 Reduce set

```
1:  Sort V descending by displacement distance
2:  lastDist ← 0
3:  for all v ∈ V do
4:      if v.dist − lastDist > ε then
5:          V_final ← v
6:          lastDist ← v.dist()
7:      end if
8:      if V_final.size() == V.size() · 0.2 then
9:          exit
10:     end if
11: end for
```

After generating the set of characteristic deformation vertices V we iterate over all procedures of the procedural model and identify all relevant commands. This algorithm is presented in Algorithm 2 and illustrated in Figure 2.5.4.



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

Figure 2.5.4.: 4 steps of the construction process of the procedural model. The points show the characteristic deformation vertices. The blue points are not on the mesh in this step. The red points have been detected on the mesh in this step.

Each procedure is executed step by step and we check if any vertices of the set V are present on the current surface. If this is the case the last executed procedure is marked as relevant. Then the found vertices are removed from V to avoid false-positives in the following procedures.

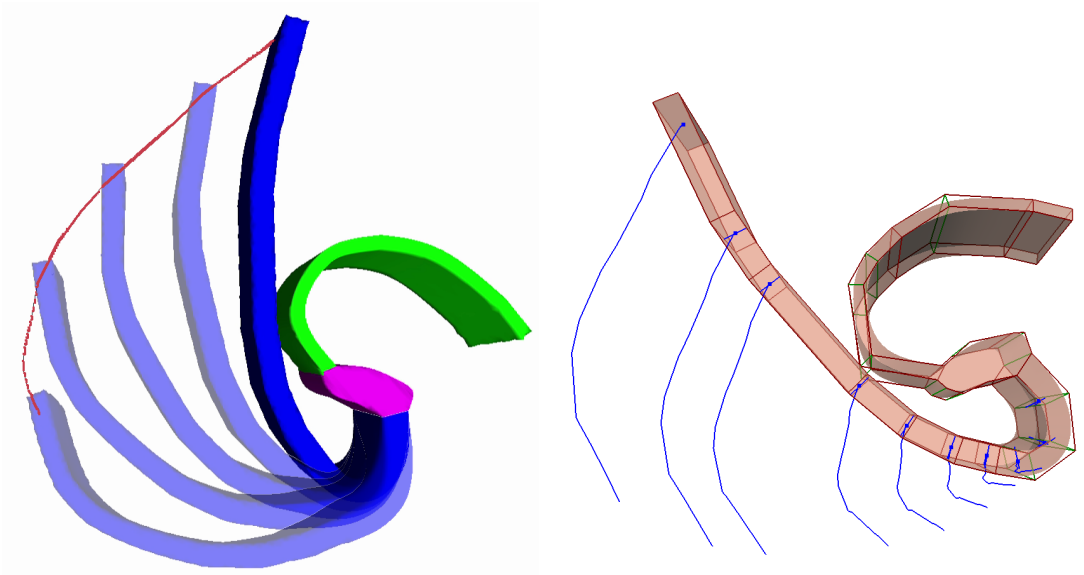
Additionally, we restrict the relevant procedures to all construction procedures. The sketched-based modeling tool has several procedures that do not create or change the surface or do not offer suitable

Algorithm 2 Identify Procedures

```
1:  function IDENTIFYPROCEDURES(V)
2:      ID := ∅
3:      for i = 0 to C.size() do
4:          if V = ∅ then
5:              exit
6:          end if
7:          EXECUTE(C[i])
8:          if ISCONSTRUCTIONPROCEDURE(C[i])
              ∧ POINTONOBJECT(V) then
9:              ID.insert(i)
10:         end if
11:     end for
12:     return ID
13: end function
14:
15: function POINTONOBJECT(V)
16:     pointOnObject ← False
17:     for all f ∈ F do
18:         if f is smooth then
19:             patch ← SUBDIVIDE(f)
20:         else
21:             patch ← f
22:         end if
23:         for all v ∈ V do
24:             if DISTANCE(v, patch) < ε then
25:                 pointOnObject ← True
26:                 V.remove(v)
27:             end if
28:         end for
29:     end for
30:     return pointOnObject
31: end function
```

parameters for our purpose. We identified the following three construction procedures as relevant: Sketching extrude, sketching rotation-extrude and drag. Note that a drag can be performed on a face, edge or vertex.

Offset Tracking: To imitate the deformation with the identified relevant procedures we calculate reference points r of the mesh and global offsets g for the procedures. The reference points r are located on the initial polygon mesh. Since the deformation does not change the mesh topology, the reference points can be calculated for every deformation step. A global offset g can be calculated from the reference points. A global offset describes the target location of a single procedure. Therefore, for each deformation step we calculate the global offsets from the given reference points and define a B-Spline function approximating the sequence of global offsets g. The B-Spline is generated with the method of Bein et al. [BHSF09], which is also used for the sketching of the lines. Figure 2.5.5 (a) shows the intermediate steps of a deformation and (b) shows the resulting B-Spline curves for each individual procedure. Note that when multiple parameters influence the global offset, all individual offsets are summed up to retrieve the final global offset.



(a) (b)

Figure 2.5.5.: (a) Intermediate steps of the deformation. (b) The resulting B-Splines for each procedure.

Procedures: For each procedure we need to define the reference points r and the global offset g:

Sketching extrusion: 2.5.6(a) shows the known components of a sketching extrusion procedure. n is the face normal and o is an orthogonal vector in the face plane corresponding to $\phi = 0$. ra is the rotation axis of the extrusion and α the rotation angle. The global offset g is the midpoint of the target face and

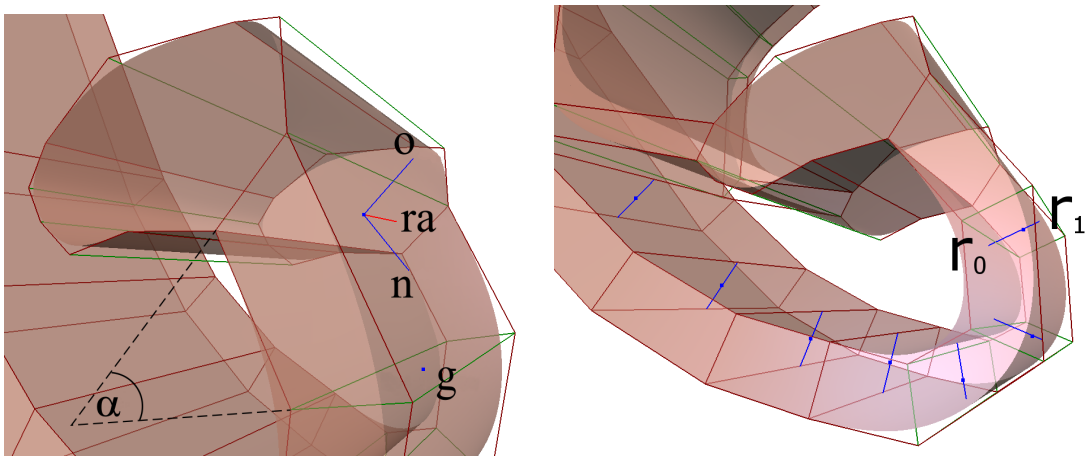
can be calculated with the midpoint of the given starting face of the extrusion and the given cylindrical coordinates.

The reference points r_0 and r_1 on the mesh are calculated by using g, o, α and ra. We calculate a direction vector corresponding to o rotated by α around ra. We shoot a ray from g in direction of this vector and in the opposite direction. The intersection points with the mesh are r_0 and r_1.

For the offset tracking of the sketching extrusion we need to be able to calculate g with given reference points r_0 and r_1. Therefore we store a barycentric coordinate λ, defining the position of g on the line from r_0 to r_1. 2.5.6(b) shows the reference lines and reference points together with the position of g. Hence, we are able to derive the global offset g:

$$\lambda_g = \frac{|r_1 - g|}{|r_1 - r_0|} \quad , r_1, r_2 \in D_0 \tag{2.5.2}$$

$$\Rightarrow g = r_1 - \lambda_g \cdot (r_1 - r_0) \quad , r_1, r_2 \in D_i. \tag{2.5.3}$$



(a) (b)

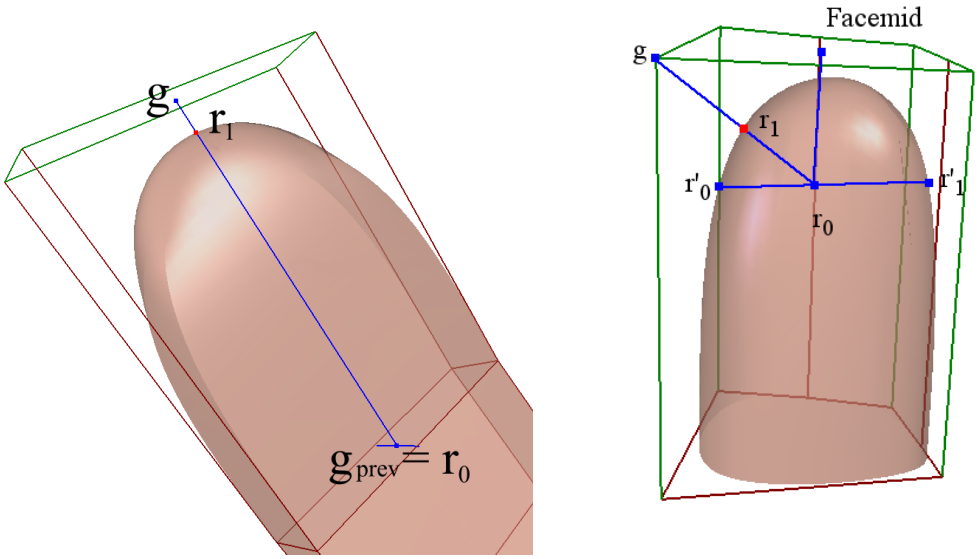
Figure 2.5.6.: (a) All components of a sketching extrusion. (b) the reference lines of a sketching extrusion

This scheme is valid for all sketching extrusions within a sequence of smooth extrusions. However, there are two exceptions: when the sketching extrusion models an end piece of the object and when the sketching extrusion lies within a sharp and discontinuous part of the object.

In the case of the end piece of an object the midpoint of the target face g does not lie inside the mesh. Therefore, the rays from g do not intersect the mesh and the reference points r_0 and r_1 can not be calculated. As Figure 2.5.7(a) shows, we use different reference points for g in this case. We take

the previous g_{prev} as reference point r_0 and shoot a ray from g_{prev} to g. The intersection point with the surface is the reference point r_1.

In the case of a discontinuous part of the object the rays shot from g to find r_0 and r_1 do not intersect the mesh near g. This can be detected as the reference line is of bad quality. This means that the reference line is very long and g not located near the middle of the line. In this case we rotate the direction vector of the ray along *ra* until a good quality is achieved for the reference line.



(a) (b)

Figure 2.5.7.: (a) The anchoring scheme of a sketching extrusion for the case of an end piece of the object. (b) The anchoring scheme of a drag vertex or drag edge.

For the sketching extrusion we also need to consider that local angles have to be recalculated when the offset changes. The new rotation angle α_i is the angle between the initial normal vector n_0 and the vector of the line from the previous g to the following g.

$$\phi(u,v) := \cos^{-1}\frac{u \cdot v}{|u| \cdot |v|} \tag{2.5.4}$$

$$l_i = \begin{cases} g_{i+1} - g_{i-1} & ,i \in \{1,\ldots,n-1\} \\ g_{i+1} - g_i & ,i = n \end{cases} \tag{2.5.5}$$

$$\alpha_i = \begin{cases} 2\pi - \phi(n_0, l_i) & , ra \cdot (n_0 \times l_i) < 0 \\ \phi(n_0, l_i) & , \text{else.} \end{cases} \tag{2.5.6}$$

Drag: There are three type of drag operations: drag face, drag edge and drag vertex. The calculation of the reference points and global offset for a drag face is similar to the sketching extrusion procedure, since the drag describes the displacement of one face position to another face position. The difference is that g_{prev} is not present. To solve this issue, we shoot a ray from g in the opposite direction of the face normal and define the intersection point with the surface of the mesh as r_1. r_1 lies on the surface of the mesh. We define a second reference point r_0 inside the mesh. We calculate r_0 by starting at r_1 and following the former ray direction by an additional epsilon. This r_0 is not on the surface of the mesh and therefore needs additional reference points itself. We define r'_0 and r'_1 by using the same technique as we used at defining the reference points at a normal sketching extrusion. i.e. We calculate a reference line across r_0 and take the two intersections points as r'_0 and r'_1.

For the drag edge and drag vertex we have to slightly adapt this technique. The resulting scheme is presented in Figure 2.5.7(b). The difference to the drag face procedure is, that these two procedures are defined by a given vertex and not by a face. We calculate r_0 as before by using the face midpoint and then we shoot an additional ray towards g to find the new reference point r_1.

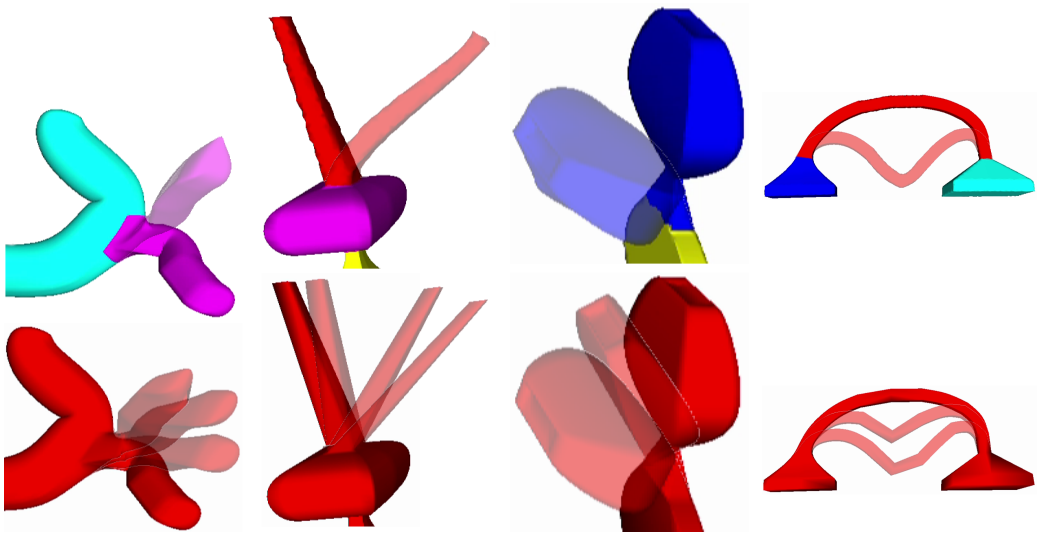
Sketching rotation-extrude: The sketching rotation-extrude has a very different parameter set than the sketching extrude or drag. This is the case because the rotation-extrude has a fixed direction. Therefore, no angle, rotation axis or cylindrical coordinates are present. The extrusions of the rotation-extrude are always in the direction of the face normal and the parameters vary the length and width of the extrusions. For this reason, the rotation-extrude is not naturally suited to reproduce deformations. Instead of using the parameters of the rotation-extrude we transform rotation-extrude procedures into normal sketching extrude procedures with an additional scaling of the faces. That means, that we define the target face midpoint of the rotation-extrude as global offset g and compute the angle, rotation axis and cylindrical coordinates for the sketching extrude. The cylindrical coordinates can be directly compute from g_{prev} and g. The only difficult parameter is the rotation axis. We compute a suited rotation axis for a specific deformation. We use the original g_{orig} and the deformed g_{def} to define the rotation axis ra:

$$ra = (g_{1,orig} - g_{0,orig}) \times (g_{1,def} - g_{0,def}). \tag{2.5.7}$$

After transforming a single sketching rotation-extrude procedure to a normal sketching extrude procedure we can use the technique for sketching extrudes to derive the reference points r_0 and r_1.

2.5.5. Evaluation and Discussion

We evaluated our approach by creating several examples and comparing the result of the parameterization of the procedural model with the original deformed meshes. Several results are shown in Figure2.5.8. (a), (b) and (c) show the result of parameter insertions into a single procedures. These demonstrate the effect of the parameter insertion on a single procedure at an end part. (d) also shows



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

Figure 2.5.8.: Examples of parameter insertions in a single: (a) sketching extrusion. (b) drag face. (c) sketching rotation-extrusion. (d) sketching extrusion in the middle of the object.

the insertion into a single procedure, but in the middle of the object. In all cases reasonable approximations are achieved even though only a single procedure is used. The results show that each procedure can achieve the desired deformation approximation by inserting parameters into the procedural model.

An important part of the evaluation is the combination of parameters. The user should be able to perform several deformations to combine parameters for the definition of complex variations of an object. For this reason we present a combination of parameters in Figure 2.5.9. The results show that the parameters perfectly combine the performed deformations resulting in a complex variation with two deformation parameters.

In Figure 2.5.10 we show the result of a segment-spanning deformation. The complete chair is stretched with the deformation. In this case multiple procedures have to be considered to achieve the complex deformation. Our approach is able to identify all procedures and insert a single complex parameter varying the object as shown in (b).

In our last evaluation we inserted multiple complex parameters of all types in a single object. The procedural model of the plane used in Figure 2.5.11 is composed of 76 modeling commands and includes five complex parameters that vary the angle of the cockpit, the wings and their tips. With the help of only five intuitive deformations, the user quickly defined the desired parameter spaces and created a procedural model that describes a range of different planes. Some of the deformations are visualized in Figure 2.5.11(a). By varying the values of the parameters, the plane changes its structure

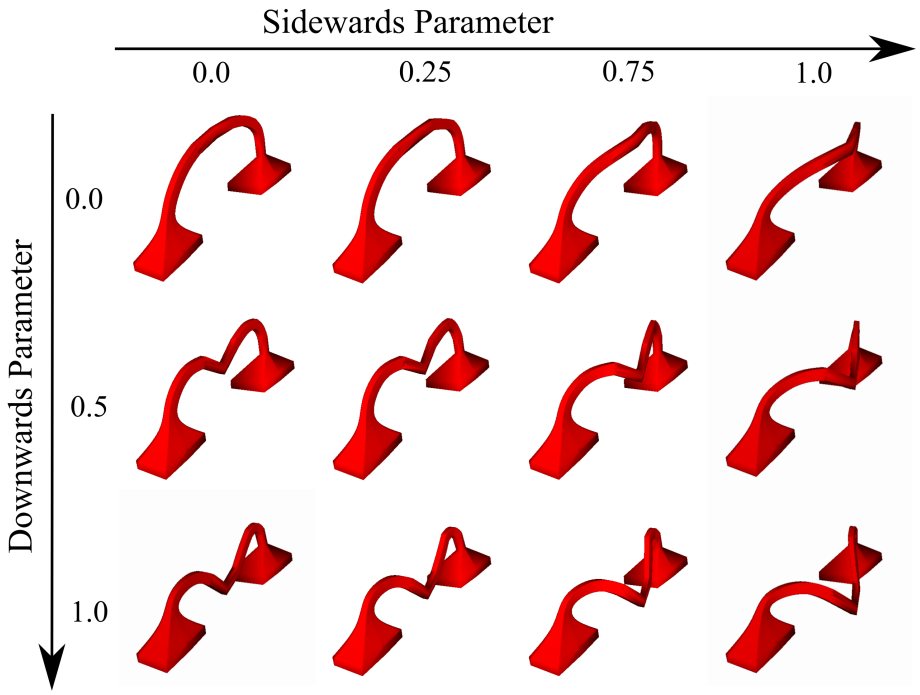
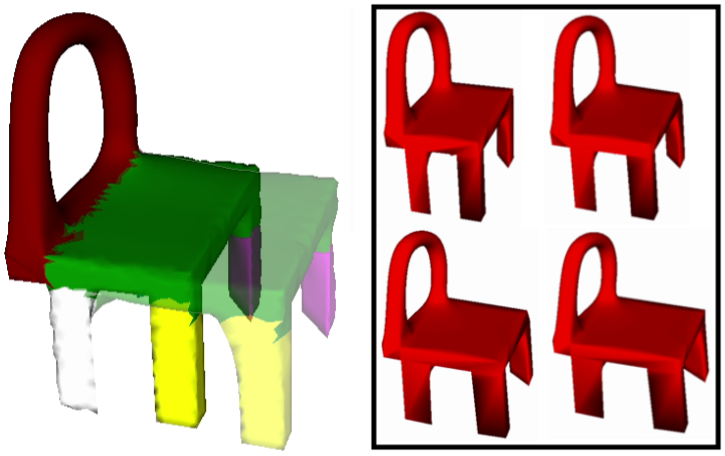
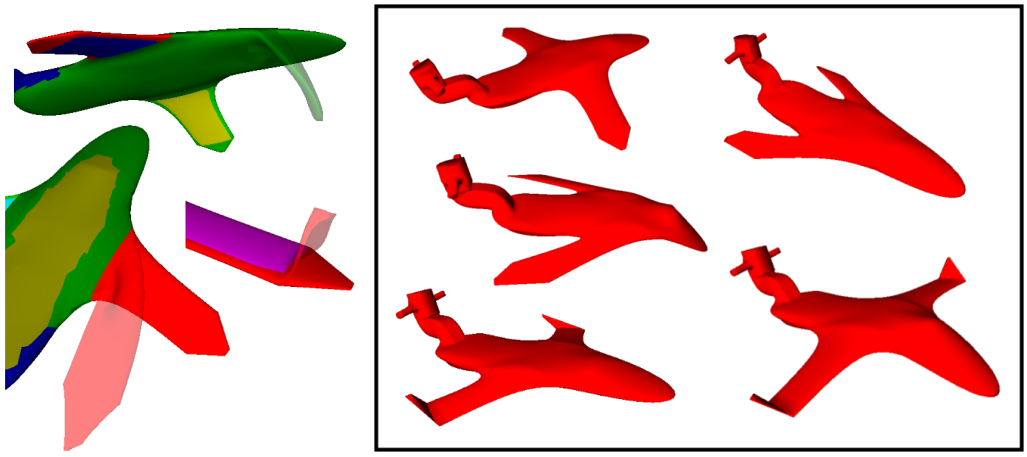


Figure 2.5.9.: The combination of two parameters.



(a) (b)

Figure 2.5.10.: (a) The deformation with the segment-spanning face handle. (b) resulting objects with different parameter values.



(a) (b)

Figure 2.5.11.: (a) Several deformations of the plane. (b) Resulting objects with multiple changed parameter values.

in the bounds of the defined object class. Figure 2.5.11(b) shows a number of samples, generated by randomly varying the parameters between 0 and 1.

Summarizing, the approach gives the user the possibility of defining complex parameters with simple and low effort sketch and deformation interactions. The results show that not only single deformations are well approximated by the automatically generated parameters but also the combination of several parameters is possible. Therefore, the user has a high amount of control to define and combine complex parameters.

Our approach is only limited by the degrees of freedom of the procedures of the procedural model. Since the procedures are directly used to imitate the deformation, it can happen that deformations in areas with very few procedures are not perfectly approximated. When there are enough procedures the approximation calculated by our system is highly accurate. Even though, the problematic cases are seldom, the degree of freedom should be increase artificially. A solution would be to split up single procedures of the procedural model into multiple new procedures resulting in the same final mesh. The additional procedures increase the degrees of freedom and therefor increases the accuracy of the approximations.

2.6. Automatic Parameterization

The parameterization of the semi-automatically generated procedural model (See Section 2.4) has to be done manually, by using the insertion interface. Even though, this offers many possibilities and a high level of control for the user, large parts of the parameter insertion process can be automated. Hence, we propose an automatic parameterization technique to enhance the process.

We do not omit the possibility of manually inserting parameters for very special variations of the objects. However, many parameters can be represented as length, width, depth and rotation changes of any part of the construction of the object.

With this target in mind, we developed parameter variations for all relevant modeling operations of the used sketching tool. Our goal is to compute several possible variations and evaluate these and insert all relevant parameters into the procedural model automatically. Additionally, we include the option of more control by showing all proposed parameters to the user and let the user decide which variations make sense. Therefore, the user can define all parameters with a few clicks.

The problem with this concept is that there might be hundreds of possible parameters. The user cannot inspect every possible parameter. Therefore we do not only generate possible variations but order them by 'importance' and furthermore automatically group related parameters together. The grouping has two reasons: primarily the parameter list is easier to inspect with groups, secondarily it makes sense to vary related parameters together. This is also related to symmetries. For example when we model a 3D object of a human, the model has 2 arms. Is is more reasonable to vary the thickness/length of both arms together.

2.6.1. Generation of Parameters

Table 2.6.1 shows the overview of the operations, the initial parameters, the variation and the subsequent evaluation of the importance of the new parameter.

The only relevant operations are extrude, rotation-extrude, insert-vertex, drag, scale and rotate. All other operations do not include numerical values that can be varied. Operations like remove only include id values. In our list we treat extrude and rotation-extrude as single operations. In the modeling tool a sketched extrude consists of several extrude operations bound together. For our parameter analysis we treat every single extrusion separately.

Procedure Variables: The modeling tool outputs every operation with predefined variables. An extrusion is described with cylindrical coordinates ρ, ϕ and z. A rotation-extrude is always in normal direction and therefore it is defined using the width w and length l of the extrusion. The insert ver-

Operation	Procedure Variables	Automatic Variation	Importance Evaluation Measure
Extrude (Sketching)	(ρ, ϕ, z)	$(2\rho\ , \phi, 2z)$	$0.1\delta_{volume} + 0.25\delta_{surface} + 0.1\delta_{BSvolume} + 0.2\delta_{projection}$
		$if\ \rho > 0.15z\ and\ \phi \in [90,270)$: $(\rho, \phi + 180(\sqrt{0.5} - 1), z)$	$0.75\delta_{projection} + 0.1\delta_{vertexdistance}$
		$if\ \rho > 0.15z\ and\ \phi \in [270,90)$: $(\rho, \phi + 180(\sqrt{1.5} - 1), z)$	
Rotation-Extrude (Sketching)	(w, l)	$(w, 2l)$	$0.15\delta_{volume} + 0.3\delta_{surface} + 0.1\delta_{BSvolume} + 0.2\delta_{projection}$
		$(0.5w, l)$	$0.7\delta_{volume} + 0.15\delta_{surface} + 0.05\delta_{projection}$
Insert Vertex	(λ_1) $with\ \lambda_2 = 1 - \lambda_1$	$if\ \lambda_1 \geq 0.5 : (0.5 + 0.5\lambda_1)$	$0.1\delta_{surface} + 0.3\delta_{projection} + 0.15\delta_{vertexdistance}$
		$if\ \lambda_1 < 0.5 : (0.5\lambda_1)$	
Drag	(ρ, ϕ, z)	$(2\rho, \phi, 2z)$	$0.1\delta_{surface} + 0.5\delta_{projection} + 0.15\delta_{vertexdistance}$
Scale	(σ)	$if\ \sigma > 1 :\ (2\sigma)$	$0.5\delta_{volume} + 0.25\delta_{surface} + 0.1\delta_{BSvolume} + 0.15\delta_{projection}$
		$if\ \sigma < 1 :\ (0.5\sigma)$	
Rotate	(α)	$(-\alpha)$	$0.7\delta_{projection} + 0.1\delta_{vertexdistance}$

ρ = radial distance, ϕ = angular coordinate, z = height , w = width, l = length, λ = baryzentric coordinate,
σ = relative scale, α = rotation angle
δ_{volume} = volume difference, $\delta_{surface}$ = surface area difference, $\delta_{BSvolume}$ = bounding sphere volume difference,
$\delta_{projection}$ = coordinate plane projection difference, $\delta_{vertexdistance}$ = average vertex distance difference

Table 2.6.1.: Overview of all relevant operations, the automatic variations and the corresponding evaluation measures used to automatically parameterize the procedural model.

tex operation only needs the position of the new vertex on the specified edge. This is described by the barycentric coordinate λ_1 ($\lambda_2 = 1 - \lambda_1$). The translation of a drag operation is also described by cylindrical coordinates ρ, ϕ and z. A scale has a relative size σ and a rotate has a rotation angle α.

Automatic Variations: For all these parameters, we define variations to evaluate the importance of these parameters. We mostly double or half values of the operations. The reasoning behind this, is that variations are not arbitrary large. Within a single object class many parts can differ in size, direction and form. In most cases the difference of two objects is not larger than doubling or halving values. We defined a total of 11 possible variations for the 6 operations.

The extrude operation can either vary in length (doubling ρ and z) or can vary in direction angle. The direction is only varied if the radial distance ρ is at least 15% of the size of the height z. If the radial distance is smaller, the angular coordinate only has a very small influence, and therefore the possible parameter is irrelevant. For the variation of the angular coordinate there are two cases. In both cases the rotation is varied towards 90.

The rotation-extrude includes a variation of the length and of the width of the extrusion. While the length is doubled we decided to half the width instead of doubling. The reason is, that doubling the width more often leads to self-intersections. These have a less reliable influence on the evaluation measures.

For the insert vertex operation the parameter λ_1 is varied in a way that the inserted vertex is placed nearer towards the nearest of the two involved vertices. This means if $\lambda_1 < 0.5$ it is moved towards the first vertex and if $\lambda_1 >= 0.5$ it is moved towards the second vertex.

There are two cases for the variation of the scale operation: Either the scaling enlarges or the scaling diminishes. In the first case we double the scale and in the second case we half the scale.

The rotation operation is varied by inverting the rotation. The inversion is preferred as it always leads to a reasonable variation. A doubling of the angle could lead to a twist in the object part.

Importance Evaluation Measures: To measure the importance of an operation we follow a simple rule: Small detail changes are less relevant. The bigger a change the higher is the importance. Hence, we measure how big the changes are geometrically. The difficulty is, that different operations change the mesh in a different way. While some operations change the volume of the mesh, other operations only change a direction and not the volume. For this reason, we include several measures and define the importance measure for each operation individually.

We generate a single mesh for every variation and compare the varied mesh to the original mesh taking into account 5 measures: The volume of the 3D object, the surface area of the mesh, the bounding sphere volume, the projection of the surface on to the coordinate planes and the average distance of the vertex positions from one mesh to the other.

For all measures the base mesh is normalized, so that the centroid is at the origin and the mesh is within a radius of 1. All variations are normalized equal to the base mesh. The volume is computed with the method of Zhang et al. [ZC01]. The surface area is the sum of all polygon areas. The bounding sphere has its center at the origin and its radius is the distance to the furthest vertex of the mesh.

The coordinate plane projection difference is computed by projecting the surface of the mesh on to the three coordinate planes: the x-y plane, the y-z plane and the z-x plane. We conduct this projection by creating an image of 64x64 pixels for each plane. A pixel is set to true if any part of the object is projected onto this pixel.

The average distance of the vertex positions can be computed as the base mesh and the varied mesh do not differ in the number of vertices. We average the Euclidean distance between all vertices.

The final value of the difference of the two meshes is calculated by the following equations:

$$\delta_p = \frac{p(v) - p(b)}{p(b)} \quad (2.6.1)$$

$$\delta_{projection} = \frac{\sum_{i=0}^{m} xor(v_{pixel\_i} - b_{pixel\_i})}{m} \quad (2.6.2)$$

$$\delta_{vertexdistance} = \frac{\sum_{i=0}^{n} |v_{vertex\_i} - b_{vertex\_i}|}{n} \quad (2.6.3)$$

$v = variation\ mesh\ \ b = base\ mesh$
$p \in \{volume, surface, BSvolume\}$
$m = number\ of\ pixels\ in\ projection\ planes$
$n = number\ of\ vertices\ in\ the\ mesh$

The overview Table 2.6.1 shows the final composition of the evaluation measure for each parameter variation. The exact compositions are determined empirically. Note that not all 5 differences are taken into account for every parameter. For example, in the rotation operation the surface area difference is irrelevant as it does not change with a rotation.

2.6.2. Parameter Grouping

Groups of parameters are new parameters themselves. When the group parameter is changed all underlying operations are changed respectively. Groups of parameters are formed by finding related operations with related parameters. This is generally the case if two operations are similar. Operations are similar if their values are similar. Therefore, we first define the similarity of two values x and y and two angles α and β:

$$similarity(x,y) = 1 - \frac{|x-y|}{max(1,|x|,|y|)} \quad (2.6.4)$$

$$similarity(\alpha,\beta) = 1 - \frac{|\alpha-\beta|}{c} \quad c \in \{45,90\} \quad (2.6.5)$$

For the similarity of angles we need to cover additional special cases since two angles of related operations should be considered similar if the one angle is the mirrored version of the other. Therefore, we do not only check the original angle but also all mirrored angles. In Figure 2.6.1 we show all mirrored versions of angles that are considered for the similarity. The red line indicates the original angle. The 7 blue lines indicate all possible mirrored angles. The angles with blue surrounding take the original equation for their similarity. The angles with yellow surrounding are more seldom and only take half of the range, which is achieved by exchanging the 90 by a 45 within the equation.

To calculate the similarity of two operations we multiply all similarities of their parameters. Finally, we set a threshold for the similarity of each operation as some operations are much more likely to have a higher similarity (e.g. insert vertex) than others (See Table 2.6.2).

After finding all similar operations we need to further process them to identify actual groups. Similar operations might be present throughout the procedural model, even though they are unlikely to be

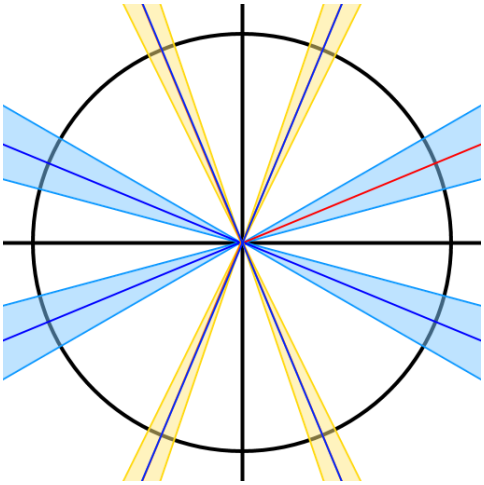


Figure 2.6.1.: The current angle is shown as red line. The other 7 blue lines represent the angles that are considered to be similar due to mirroring. The blue areas and smaller yellow areas show the range near the angles in which these are estimated as similar.

Operation	Threshold
Extrude	0.3
Rotation-Extrude	0.45
Insert Vertex	0.95
Drag	0.85
Scale	0.9
Rotate	0.7

Table 2.6.2.: The thresholds indicate at which distance two operations are considered to be similar.

related if one operation is at the very start and the other at the very end. The procedural model gives a structure to the construction process and includes related sequences of operations. Hence, we can deduct the relation of operations by their relative position in the sequence of operations. We identified that related instructions are either present in the pattern AA or with the pattern ABAB. This means they are not only similar but also subsequent, as in pattern AA. Or a combination of operations AB is subsequent, forming the pattern ABAB. Finally, we build groups of similar operations which are present in one of these two patterns within the sequence of operations of the procedural model.

2.6.3. User based Parameter Choice

By default a parameter is considered to be important if the importance value is bigger or equal than 1. The user phase can be skipped and all parameters chosen with the default threshold are considered to be important. However, which parameters actually make sense for the object class is a semantic definition. Therefore, we give additional control to the user by including the possibility of refining the parameter choices in this step and offer a simple interface to inspect all parameters and choose all important parameters. This interface is shown in Figure 2.6.2. The parameters are ordered by their importance and the user can click on every parameter. The mesh variation is shown in the interface when a parameter is chosen. The user can check or uncheck any parameter or parameter group with a single click. He can also expand a group node and check or uncheck individual operations. He can create a new group by choosing several parameters. Additionally he can rename parameters, since the parameters can be related to actual semantic characteristics.

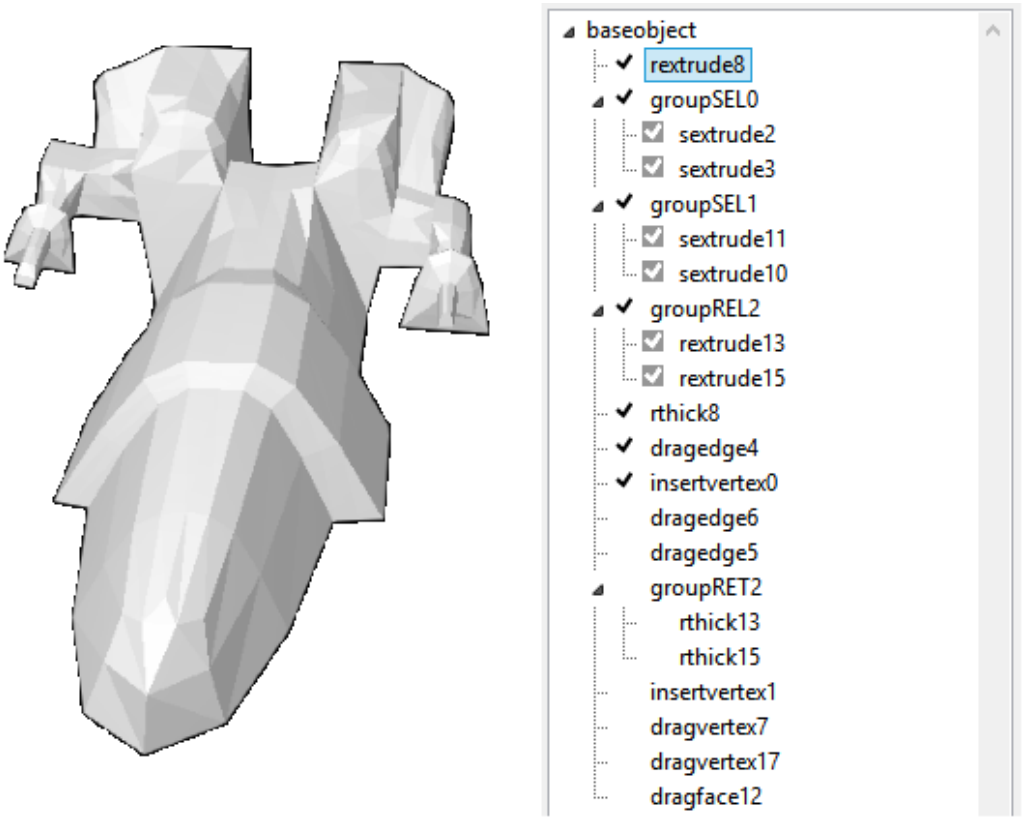


Figure 2.6.2.: The user interface of the parameter insertion.

Range Estimation: In the end, when a parameter is considered to be important, a new parameter x is inserted into the procedural model. With our final parameterized procedural model it should be possible to generate reasonable random variations of the original object. To enable this possibility, we need to additionally define a valid range for the parameter x. This is done automatically, even though the user can still change it individually. This corresponds to our concept to automate as much as possible and still give the user as much control as possible.

Operation	New Inserted Parameter x	Initial Range of x
Extrude (Sketching)	$(x\rho\ , \phi, xz)$	$[0.125,8]$
	$(\rho, \phi + 180(x-1), z)$	$[0,2.0]$
Rotation-Extrude (Sketching)	(w, xl)	$[0.125,8]$
	(xw, l)	$[-2,1]$
Insert Vertex	$(0.5 + x\lambda_1)$	$[1,1.9]$
	$(x\lambda_1)$	$[0.1,1.0]$
Drag	$(x\rho, \phi, xz)$	$[-8,8]$
Scale	$(x\sigma)$	$[-3,3]$
Rotate	$(x\alpha)$	$[0.125,5]$

Table 2.6.3.: Overview of the inserted parameter in all relevant operations used to automatically parameterize the procedural model.

The overview Table 2.6.3 shows the initial range estimations for every parameter type. However, we noticed that these large variations are not suited for every case. The maximal or minimal range values of a parameter should still lead to an object of the original object class. For each parameter we generate a mesh with the maximal and minimal value for the specific parameter and measure the difference to the base mesh using the panorama distance [PPTP10]. The panorama distance is a geometrical measure used for measuring the general difference of two 3D objects.

We set a maximal threshold $C \cdot t$. This threshold contains two parts. C is a constant multiplier that is set to 1 by default. The user can change C if he wants to allow a bigger range. t is the actual value of the maximal allowed panorama distance. It is obtained by measuring the panorama distance of the base mesh to all variations generated during the preceding phase. In this way, t is set to the maximal difference that has already been approved by the user.

If the panorama distance is bigger than the threshold $C \cdot t$ the range is diminished and reevaluated. The algorithm ends when the maximal as well as the minimal range values of all parameters have been checked to be lower than the threshold.

2.6.4. Evaluation and Discussion

We evaluate the correctness of the proposed automatic parameter insertion technique. Note that for the preceding modeling part we used the semi-automatically generated procedural model (See Section 2.4). Our insertion techniques is dependent on the importance evaluation measure and the parameter grouping. The importance and the grouping basically decide which parameters will be inserted in the model. Therefore, our evaluation focuses on two questions: how accurate is the importance evaluation measure and how accurate is the parameter grouping?

For the evaluation we created several example models with the modeling tool and manually annotated important parameters and appropriately grouped related parameters. The test models are shown in Figure 2.6.3. Then we started our algorithm and retrieved the automatically proposed parameters and the automatically detected groups. Note that we did not change any parameters during the user phase. We inserted all parameters that were automatically detected as important by the default threshold of 1.

	CP	FP	FN	CN	Accuracy
Airplane	9	1	1	11	90.91%
Ship	7	3	0	8	83.33%
Stool	6	3	0	12	85.71%
Animal	11	12	1	18	69.05%
Spaceship	6	1	0	6	92.31%
Tower	11	4	1	7	78.26%
Humanoid	10	8	1	12	70.97%
Chair	6	6	0	8	70.00%
Average	9.42	5.43	0.57	11.71	80.07%
CP = Correct Positive, FP = False Positive FN = False Negative, CN = Correct Negative					

Table 2.6.4.: The accuracy of the parameter importance evaluation measure

Figure 2.6.3.: All models that were used to test the parameter insertion.

	CG	FG	MG	Accuracy
Airplane	7	0	3	70.00%
Ship	3	1	0	66.67%
Stool	8	1	0	87.50%
Animal	7	0	0	100.00%
Spaceship	3	0	0	100.00%
Tower	4	0	1	80.00%
Humanoid	9	0	1	90.00%
Chair	7	1	0	85.71%
Average	6.00	0.38	0.63	84.98%
CG = Correct Group, FG = False Group MG = Missed Group				

Table 2.6.5.: The accuracy of the parameter grouping

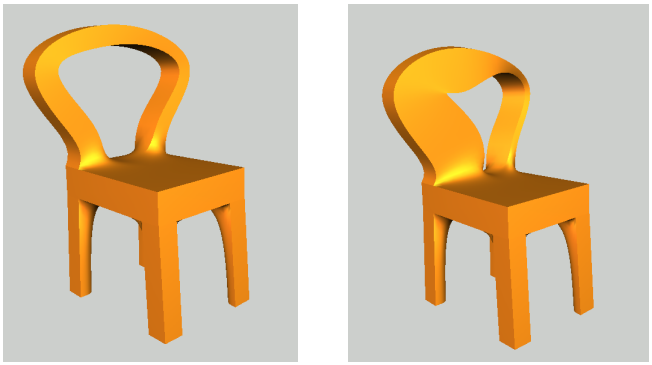


Figure 2.6.4.: False positive result of the parameter importance measure

We present our results in Table 2.6.4 and Table 2.6.5. Table 2.6.4 shows that the most relevant parameters are automatically categorized as important in 80% percent of the cases. Table 2.6.5 presents a similar result for the parameter grouping. 84% percent of the groups are correctly identified by the algorithm.

Discussion: The grouping accuracy as well as the importance measure accuracy are fairly high. However, there are still cases where a parameter have been missed or a group not detected.

Table 2.6.4 shows that false negatives are quite seldom. Our algorithm rather finds too many important parameters, so that false positives are the important error cases. In Figure 2.6.4 we show a false positive. This parameter was found to be important by our algorithm even though the change of the parameter does not lead to a semantically consistent outcome. In fact, our measures are not able to actually grasp the semantic meaning behind the parameters. We measure the geometric influence, since a high geometric influence most likely gives a parameter with a strong semantic influence. However, this is not always the case.

Note that our system includes a user phase where the user can refine parameters, rename the parameters and also annotate groups with a few clicks. Therefore, the user is able to correct semantic inconsistencies.

The parameter grouping also has erroneous cases. We analyzed these cases and found out that the majority of the missing groups and false groups are caused by the insert vertex operation. In contrast to other operations the insert vertex operation has a low variety of values and only includes a single parameter. The values are mostly within $[0.25, 0.75]$. Even thought we set a very tight threshold with 0.95 for this operation the similarity computation is less reliable for this operation. For this reason we get some false groups and missing groups for this operation. As there is no simple solution for this problem, we can only highlight the insert vertex groups for the user, so that he can decide in these cases.

Summarizing, the automatic parameterization is able to include the most relevant parameters into a procedural model with a high relevance accuracy. The offered user interface for fast inspection and choice of appropriate parameters gives the user the full control in complex cases. Furthermore, in this

interface the user can rename all parameters, so that it also offers the direct possibility of including semantics into the procedural model.

2.7. Automatic Procedural Model Generation

With our semi-automatic procedural model generation, based on sketch-based modeling the user can directly create and parameterize a procedural model from sketch. In several steps the user has a high amount of control during the modeling phase. Using procedural models for 3D object retrieval and classification is not targeting the completely unsupervised case. The user always has to invest some time to define or choose his procedural models for his task. However, our goal is to improve the task of 3D object retrieval and classification on any level of control that the user needs. Though, we also seek to cover the most uncontrolled case. The user is not interested in a high level of control and wants to create procedural models as automatically as possible.

In this section we propose a completely automatic generation of a procedural model. The user only has to provide a single example object. Then, the user can define the desired parameters with a few interactions and directly use the resulting procedural model for further applications.

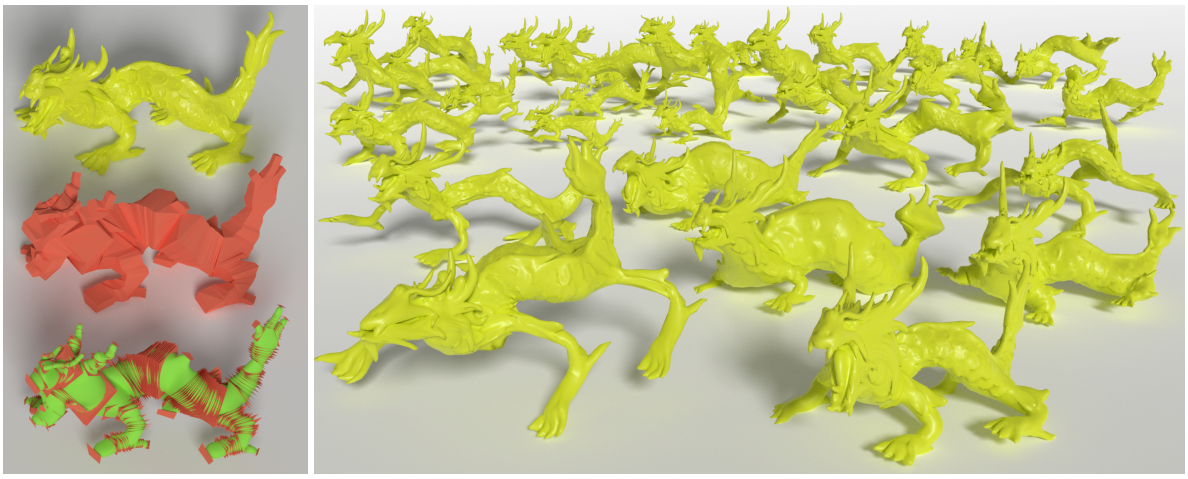


Figure 2.7.1.: The original mesh is decomposed into construction procedures. The red planes show the boundaries of each individual procedure and the green surface is the generated subdivision surface. We reconnect the generated surface to the original mesh surface, resulting in surface modifications when changing the parameters of the procedures.

2.7.1. Approach Concept and Goal

Our approach generates a procedural model from a single given polygon mesh. The procedural model consists of parameterizable procedures and represents the object construction process. The idea to achieve this is to reproduce the process of modeling an object using face extrusions. Only 2 additional procedures are needed: One procedure so start a new object and one procedure to connect faces ('bridge faces').

A face extrusion is a very common operation for modeling tools. Modeling tools usually refer to a face extrusion as an operation extruding a face in normal direction and not resizing or rotating it. When we extrude a face, the original face vertices keep their position and a new face is created where each vertex of the new face is connected to the corresponding vertex of the original face. Therefore, this operation always generates 4 new vertices and 4 new faces.

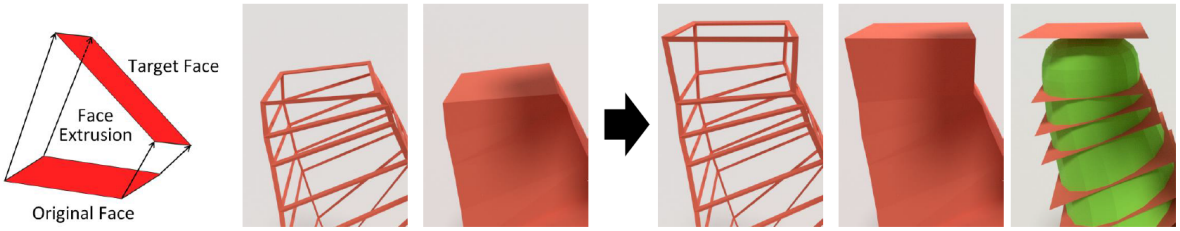


Figure 2.7.2.: The face extrusion is our basic construction procedure. An original face is extruded to a target face. The extrude face procedure is applied to a subdivision surface control structure. We show the control structure in red: As wireframe and with filled faces. In green we show the resulting subdivision surface.

In our case a face extrusion also includes a transformation of the resulting face. Therefore, the face extrusion is defined by an originating quad and a targeting quad. The operation extrudes the original face to the target face. We illustrate our basic operation in Figure 2.7.2. The extrude face procedure is applied to a catmull-clark subdivision surface control structure. The control structure corresponds to a quad mesh. A single face of the quad control structure is extruded to a target face. In the example of Figure 2.7.2 we show the control structure in red as filled quads and also as wireframe. The resulting subdivision surface is shown in green. It results from 2 iterations of catmull-clark subdivision.

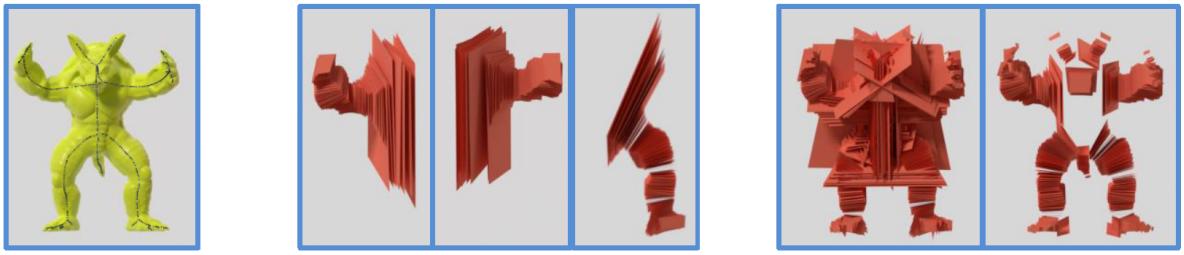
In the initial Figure 2.7.1 we see a green dragon penetrated by red quads. The red quads are the originating and targeting quads of all face extrusion procedures. The red dragon is the complete quad mesh after all extrusions have been executed. The green surface is the subdivision surface of the quad mesh performing 2 catmull-clark subdivisions. The green surface is also used to finally transfer the deformation of the procedural model to the original model.

The goal of our approach is to decompose the mesh into extrusions connected through a construction process of the object (the procedural model). Separating the object into a process of small parts enables the possibility of varying single parts of the constructions process to perform an automatized variation

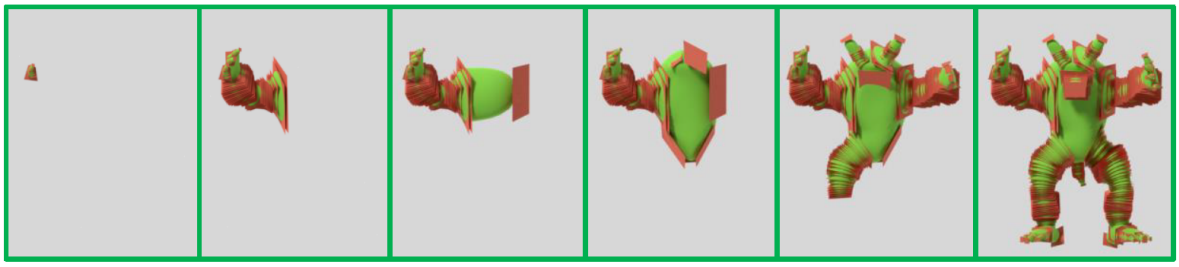
of the object shape. The extrude parts of the construction process naturally correspond to object parts and can for example vary in size, which corresponds to the thickness of the object part. Additionally our representation allows the user to define rules for the parameter changes to further refine the desired variation space.

In the following we first give an overview of the approach which includes 4 steps. Then, we explain the composition and the procedures of the procedural model. Lastly, we describe each step in full detail. We present pseudocode for the complete approach and further explain the operations in each subsection.

2.7.2. Overview



(a) Preprocessing (b) Skeleton Path Quad Fitting (c) Quad Decimation



(d) Procedural Model Generation

Figure 2.7.3.: The pipeline to generate a procedural model. The first 3 steps prepare a non-intersecting set of quads. In the final step the procedural model is generated. Step wise extrusions are performed with the prepared quads as targets.

Our approach consists of 4 steps. In the first 3 steps the mesh is analyzed, structured and a set of extrusion quads is precomputed. In the final 4th step the actual construction of the procedural model is performed. The process is illustrated in Figure 2.7.3.

In the first step we compute a skeleton for the object. The skeleton consists of vertices and edges. Most skeleton vertices have 2 edges connecting them to 2 neighboring vertices. These are path vertices. The other skeleton vertices can either be an end vertex (1 neighbor) or a split vertex (more than 2

neighbors). The skeleton guides our fitting and construction process. Also, in this step we perform several preprocessing steps which ensure a clean structure of the mesh and the skeleton for the following steps.

In the second step we separate the skeleton into paths. Each path starts and ends at a split vertex or end vertex. To guide the extrusions we fit quads for each skeleton vertex along the paths. These are later used as target quads for face extrusions. After this step the representation is over-complete as the quads intersect each other. The quads intersect where the paths meet each other: at split vertices.

To resolve this problem the third step decimates the quads iteratively around the split vertices. This results in a non intersecting set of quads. Finally, in the fourth step the procedural model is constructed by starting with an extrude part at a skeleton end vertex and step wise extruding the face to neighboring quads following the skeleton vertex path structure. At each step the following vertex can either be a path vertex, an end vertex or a split vertex. At the end vertex a final extrusion finishes the path. At the path vertex an extrusion is performed and the successor vertex is processed. At the split vertex the best extrusion sequence to connect all adjacent quads is determined and performed. The split vertex initiates several new paths. The procedural model is finished when all paths reached an end vertex.

2.7.3. Procedural Model Structure

Our procedural model includes 2 operations, 1 functional annotation and 2 structural annotation. The procedural model starts with an initial face creation operation and subsequently adds face extrusion operations. The functional annotation is used to connect two faces when topological loops occur. The structural annotations are included for additional structuring of the procedural model.

The initial face operation needs a quad Q as parameter. Q include the x, y and z-coordinates of 4 vertices:

$$InitialQuad(Q) : Creates\ the\ intial\ quad\ Q.$$

The face extrusion procedure takes a target quad Q and the face id of the originating quad, at the construction of the procedural model:

$$ExtrudeFace(Q, f) : Extrusion\ of\ face\ f\ to\ Q.$$

However, when changing the parameters of other extrusions the position of the face f might change, so that Q should change accordingly. Hence, it is important that the procedure actually processes the offset from the original face f to Q in the local coordinate system of f:

$$\begin{aligned} Q &= \{v_0, v_1, v_2, v_3\} \\ X_{local} &= v_1 - v_0 \\ Y_{local} &= X_{local} \cdot v_2 - v_0 \\ Z_{local} &= X_{local} \cdot Y_{local} \end{aligned}$$

When processing this operation the local coordinate system of the current f is calculated and the offset to a new Q' is determined. Finally, the face f is extruded to the new Q'.

The functional annotation 'face loop' is only needed if the surface is of genus 1 or higher (i.e. the skeleton includes loops). A surface incorporating a ring structure cannot be constructed with face extrusions only. Hence, we perform a special operation 'bridge faces' which connect two faces to close a loop. When the approach visits an already processed split vertex, we know that a loop has been found. A 'bridge faces' operation is performed. Though, we need to know which faces have to be bridged together to close the loop. For this reason we include the 'face loop' annotation. This annotation states that the preceding face extrusion generated a face that should be connected to another face. As there can be more than one loop in the skeleton we include a unique ID.

$$FaceLoop(ID) : Two\ Faces\ with\ ID\ are\ connected$$

The structural annotations 'begin path' and 'begin split' are included for the only reason to provide structural information.

$$BeginPath : The\ start\ of\ a\ skeleton\ path$$
$$BeginSplit : The\ start\ of\ processing\ a\ splitvertex$$

With this two annotations we can separate the structure into distinct paths and splits. We do not need explicit ending annotations as a 'begin' always marks the end of the preceding.

2.7.4. Step 1: Preprocessing and Skeletonization

In the first step we compute the skeleton which guides all following steps. We compute the skeletonization using the algorithm of Tagliasacchi et al. [TAOZ12].

The resulting skeleton equals a graph of vertices and edges. The edges are straight lines connecting vertices. The vertices are well distributed along the branches of the skeleton. Most skeleton vertices have exactly 2 edges, which connect them to their 2 neighbors. These are path vertices. End vertices only have 1 neighbor and split vertices have more than 2 neighbors.

Preprocessing of the mesh: The skeletonization needs a closed 2-manifold mesh. Therefore, it is needed to close holes and ensure a manifold polygonization of the mesh. In some cases it can happen that skeleton vertices are outside of the actual mesh. For the later steps we need the skeleton vertices to be inside the mesh. For this reason we move them into the mesh according to the skeleton structure. That means that we move them in the direction of the adjacent edge. If the vertex has more than 1 edge we choose the direction with the nearest surface intersection.

2.7.5. Step 2: Skeleton Path Quad Fitting

We separate the skeleton into distinct paths. Each path consists of a sequence of vertices with the start and end being either an end vertex or an split vertex. All inner vertices of the path have 2 neighbors and therefore are path vertices. We perform the path separation by starting at every end vertex and iterating through the skeleton until reaching a split vertex. Hence, every end vertex generates a single path. Then, we process all split vertices and create new paths for every branch that has not been visited before. These paths start at a split vertex and also end at a split vertex.

We continue, by computing quads for every skeleton vertex along every path, only excluding split vertices. Quads at split vertices are meaningless as we want to generate an intersection free representations and these quads naturally intersect with the quads of other paths ending at the same split vertex.

The quad fitting process is shown in Algorithm 3. We take the skeleton edges to construct a plane perpendicular to the edges and compute the intersection points of the plane with the object surface to determine the final quad.

In order to generate intersection free quads for every vertex of a path we iterate through the path and only generate quads that have no intersection with previous quads. Algorithm 4 describes this behavior: The initial quad is always fitted using the single edge as plane normal. For the following quads the initial quad is fitted using the average of the two adjacent edges as face normal. Though, when the fitted quad intersects the previous quad we step wise interpolate from the new face normal to the face normal of the previous face. If the face still intersects the previous quad (with an interpolation value of 0.1) it demonstrates that the previous fitted quad is incompatible with the path and deletes the previous quad. Then it restarts with an interpolation value of 1.0, which means that a quad is fitted with the initial normal. This is a very important part of the approach as the quad rotation, backtracking and deleting results in a complete intersection free quad sequence for every individual path.

2.7.6. Step 3: Quad Decimation

Merging all quads of all paths together results in various intersections of the quads. Explicitly, the path quads always intersect around the split vertices. Therefore, we process each split vertex and delete adjacent quads until all adjacent paths are intersection free to each other. Algorithm 5 describes this process. First, we make sure that each path starts at the split vertex, as we start deleting quads at the start of each path. If the path ends at the split vertex we reverse the path. Then, we handle empty paths. This can happen as we continue to delete quads. Empty paths that originally ended at an end vertex are completely deleted. A different case happens when the empty path connects two split vertices. In this case we merge the two split vertices to a single split vertex.

The actual deletion is located at ll. 15-17 of Algorithm 5. We check the starting quad of each adjacent path for any intersection with other quads of any other adjacent paths. From all these starting quads

Algorithm 3 Quad fitting for skeleton vertex

procedure FITQUAD(*skeleton vertex* v, *direction* n)
  $d^1_{plane} \leftarrow n \cdot (1,0,0)$
  if $n-(1,0,0)=0$ **then**
    $d^1_{plane} \leftarrow n \cdot (0,1,0)$
  end if
  $d^2_{plane} \leftarrow d^1_{plane} \cdot n$
  $d^1_{plane} \leftarrow d^2_{plane} \cdot n$
  Initiate fitted quad $Q \leftarrow$ *surface intersection of Ray* r *from* v *in direction* $v \pm d^2_{plane} \pm d^1_{plane}$
  for all α *from* 1 *to* 360 **do**
    Cast ray r *from* v *to* $v + d^1_{plane}$ *rotated* α *around* n
    $i_Q \leftarrow$ *intersection with edge of* Q
    $i_S \leftarrow$ *intersection with the surface*
    if $|i_Q - v| < |i_S - v|$ **then**
      Displace the edge with the according direction vector $(\pm d^2_{plane}$ *or* $\pm d^1_{plane})$ *to hit* i_S
    end if
  end for
  if v *is an endvertex* **then**
    Cast ray r *from* v *in direction* $(-n)$
    $i_S \leftarrow$ *intersection with the surface*
    Displace all vertices of Q *by* $(i_S - v)$
  end if
end procedure

Algorithm 4 Skeleton path quad fitting

```
1:  for all Skeleton paths do
2:      if path starting vertex v is an end vertex then
3:          v_s ← neighbor of v
4:          n ← (v_s − v)
5:          FitQuad(v, n)
6:      end if
7:      for all Skeleton vertices v from 1 to (path length) − 1 do
8:          i ← 1
9:          v_s ← successor of v on the path
10:         v_p ← predecessor of v on the path
11:         n ← ((v_s − v) + (v − v_p))/2
12:         n_i ← n
13:         repeat
14:             Q ← FitQuad(v, n_i)
15:             Q_p ← quad of predecessor
16:             if Q intersects Q_p then
17:                 i ← i − 0.1
18:                 if i = 0 then
19:                     delete Q_p and v_p from path
20:                     i ← 1
21:                 end if
22:                 Q_p ← quad of current predecessor
23:                 n_p ← direction of Q_p
24:                 n_i ← ((1 − i) · n + i · n_p)/2
25:             end if
26:         until Q does not intersect Q_p
27:     end for
28: end for
```

Algorithm 5 Quad decimation

```
1:  for all split vertices s do
2:      for all adjacent paths p do
3:          if start vertex of p ≠ s then
4:              reverse p
5:          end if
6:      end for
7:      repeat
8:          if ∃ adjacent path p containing 0 quads then
9:              if p ends at a split vertex s_e then
10:                 merge s with s_e
11:             else
12:                 delete the path
13:             end if
14:         end if
15:         if ∃ adjacent paths p and p' :
            the starting quad of p intersects
            with a quad of p' then
16:             From all starting quads with intersections :
            delete the quad with the longest total edge length
17:         end if
18:     until no split vertex or quad has been deleted
19: end for
```

having intersections we delete the one with the highest total edge length. We choose the edge length instead of the quad area as long thin quads should also be highly relevant.

As a results the total amount of quads is decimated. This guaranties that the quads can be used for subsequent face extrusions without overlapping faces. It is also important as the total structure of the procedural model is defined by this sequence.

2.7.7. Step 4: Procedural Model Generation

Algorithm 6 Procedural model generation

```
1:  New Procedural Model M
2:  Q ← intial quad of start vertex
3:  M add Annotation BeginPath
4:  M add Procedure InitialQuad(Q)
5:  List L_v ← add successor of start vertex
6:  List L_f ← add face id 0
7:  repeat
8:      v ← pop last element from L_v
9:      f ← pop last element from L_f
10:     if v is an end vertex then
11:         Q ← quad of v
12:         M add Procedure ExtrudeFace(Q, f)
13:     end if
14:     if v is a path vertex then
15:         if The last processed v was an end vertex
                    or splitvertex then
16:             M add Annotation BeginPath
17:         end if
18:         Q ← quad of v
19:         M add Procedure ExtrudeFace(Q, f)
20:         add successor v_s to the end of L_v
21:         add f+4 to the end of L_f
22:     end if
23:     if v is a splitvertex then
24:         SplitVertex(v)
25:     end if
26: until L_v is Empty
```

Algorithm 7 Split vertex processing

```
1:  procedure SPLITVERTEX(skeleton vertex v)
2:      if v was already visited then
3:          id ← new unique id
4:          M add Annotation FaceLoop(id)
5:          M insert Annotation FaceLoop(id) after
            the counterpart ExtrudeFace
6:          Stop the procedure
7:      end if
8:      M add Annotation BeginSplit
9:      List L_s ← insert all adjacent path vertices
10:     if number of adjacent paths > 7 then
11:         Compute 7! random permutations of L_s
12:     else
13:         Compute all permutations of L_s
14:     end if
15:     for all permutations of L_s do
16:         execute all face extrusions and compute the
              z−buffer difference of the subdivision
              surface to the original mesh
17:         L_s ← best permutation
18:     end for
19:     repeat
20:         v_n ← pop vertex from L_s
21:         f_n ← nearest intersecting face of the ray
              from v to v_n
22:         if f_n is empty then
23:             f_n ← nearest intersecting face of the ray
              from the predecessor of v to v_n
24:         end if
25:         Q ← quad of v_n
26:         M add Procedure ExtrudeFace(Q, f_n)
27:         add v_n to the start of L_v
28:         add f+4 to the start of L_f
29:     until L_s is empty
30: end procedure
```

In the final step of our approach we use the quads to generate the procedural model and construct our final quad mesh and subdivision surface. We start at a path with an end vertex as first vertex. With the first quad we initiate the procedural model.

Then, each skeleton vertex is processed individually. There are 3 cases possible: The skeleton vertex is an end vertex, a path vertex or a split vertex. The process is described in Algorithm 6 and Algorithm 7.

The skeleton vertex is an end vertex: We perform an extrusion of the current face f to the quad of the final vertex. The path is finished.

The skeleton vertex is a path vertex: We perform an extrusion to the next quad but also add the successor to the end of the processing list, which means that it is processed right after. This ensures that the 'begin path' comment actually separates complete paths.

The skeleton vertex is a split vertex: In this special case we perform several steps. If the split vertex has already been visited before we encountered a loop in the skeleton. For this special case we add the 'loop face' annotation to the procedural model and also add the same annotation for the face extrusion procedure that generated the face to which we are connecting.

In contrast to the well aligned quads of the paths the split vertices are more difficult to handle. We have several paths and want to connect our current face to the start quad of each of the paths. We need to find a sequence of extrusions beginning at the current quad and resulting in a quad for all adjacent paths.

For each extrusion we need to determine the originating quad and the target quad. The target quads are the start quads of the adjacent paths. The originating quads can be any quad of the split region. To determine these we cast a ray from the split vertex towards the target quad and take the face with the closest intersection. In seldom cases this ray has no hit point. For this case we cast a ray from the predecessor of the split vertex towards the target quad. As this vertex is inside the mesh it always has a hit point.

However, we also need to know in which sequence the target quads are chosen. There is no natural order of the adjacent paths at the split vertices. Also, for the resulting quad mesh and subdivision surface the order of extrusions makes a big difference. Therefore, we need to determine the best sequence of target quads. As this is a difficult problem we determine which connection sequence to all adjacent paths is the best by using an evaluation measure and evaluate all possibilities.

We compute all possible permutations for the connection sequence (if the number of adjacent paths are more than 7 at a single split vertex we take a randomized set of 7! permutations). We perform all extrusions of the split and compute the resulting subdivision surface for each of the possibilities. Then we measure a z-buffer difference between the original object and the subdivision surface. We take the sequence of extrusions with the lowest z-buffer difference.

Z-buffer differences are used to measure object differences. Our z-buffer difference consists of pixel-wise differences of 14 views. We use the 6 views from each of the directions of the main axis and additionally use the 8 diagonal views from the corners of a cube around the origin. For each of the 14 views we render an image of the size 256x256 with orthogonal projection and measure the difference

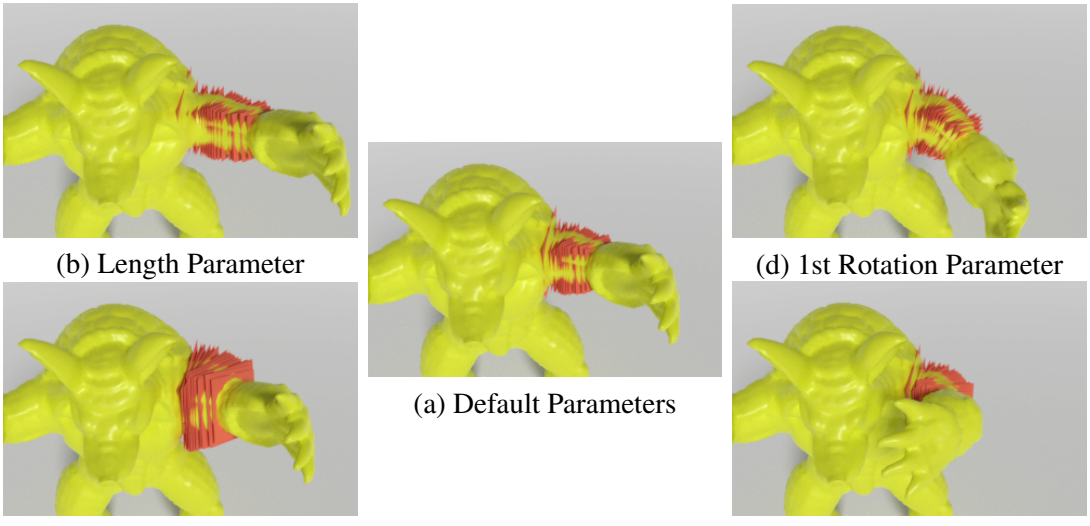
of the z-buffers. Additionally, we want to strongly punish sequences that produce a surface bigger than the actual object. Each pixel filled by the subdivision surface, which was a background pixel at the original mesh counts as a difference of 8, instead of 1.

Finally, for each performed extrusion we add the targeted vertex into the list, as these represent the start of a new path. After processing all vertices of the list, the procedural model is complete.

2.7.8. Parameter Definition for Object Variation

Our novel procedural representation has a high potential towards several applications as a procedural model naturally introduces flexibility and variability. To show the high potential of our approach we conducted a parameterization of the procedural model and created a setup that allows users to define parameters for groups of face extrusions or individual face extrusions. In the following we explain the parameterization and describe the possibilities of inserting and defining new parameters:

Object Parameters: To define the variations the user can assign one or multiple face extrusions, paths (i.e. groups of face extrusions) or splits to a parameter p_i. A parameter p_i has a size, length and 2 rotation values.



(b) Length Parameter (d) 1st Rotation Parameter

(a) Default Parameters

(c) Size Parameter (e) 2nd Rotation Parameter

Figure 2.7.4.: The influence of the 4 face extrusion parameters on the extrusion quads and the original surface.

Face Extrusion Parameters: Figure 2.7.4 illustrates the 4 parameters that we defined for each extrusion. Each extrusion can be varied in length, size and rotation. As the rotation includes two rotation axis using the local quad coordinate system, we have 2 parameters for the rotation. The size parameter increases the relative difference of all vertices to the middlepoint of the target quad. The

length parameter increases the distance between the originating quad and the target quad. We measure the distance between the middlepoint and increase it relatively with the parameter. Therefore, the length and size parameters are set to 1.0 by default. The rotation is performed by tilting the quad. We rotate the normal towards the local x or the z axis. These parameters are given as an angle and by default set to 0.0.

Split Parameters: A split consists of several face extrusion operations. As the split parts semantically differ from other parts we defined a special length and size parameter varying the complete split part. We calculate the midpoint of the split part as the average of all quad midpoints of the split. The length parameter changes the relative distance of all split quads to the split midpoint. The size parameter changes the relative size of all quads of the split. Both have 1.0 as default value.

Connection to the Original Object: To apply the changes of the parameters to the original object we take the subdivision surface of the resulting quad mesh generated by the procedural model as reference. For each vertex of the original mesh we compute a set of x influence points on the subdivision surface. For determining the influence points we compute a set of 20 000 points on the subdivision surface with a Poisson disk sampling [CCS12]. The influence points of a vertex of the original mesh are the nearest x points of the 20 000 points. When the parameters of the procedural model are changed the subdivision surface changes. To transfer these changes to the original mesh we compute the offset of all influence points. The offset is the difference of the original position to the new position. Each vertex of the original mesh is then updated by displacing the vertex by the average offset of all his influence points.

The user can set x to any value between 1 and 1000. By taking the average offset of all influence points, every influence point has the same weight for the final displacement. We considered to give nearer points a higher weight than the further points. However, when further points have less influence, the parameter x has a lower impact. Additional points become less relevant. The user only has the parameter x and the skeleton granularity value. Therefore, we give x a high impact and give the user the full control about the range of the influence. Hence, we take the average offset of all x points.

Granularity Value: The used skeletonization algorithm [TAOZ12] includes a ratio w_L/w_M controlling the smoothness of the skeleton. We refer to this ratio as 'granularity value'. A higher granularity value gives a more detailed skeleton with more vertices and more branches and a lower granularity gives a rougher but smoother approximation. The best value for this parameter is dependent on the mesh but also on the user. We compute a procedural model for 7 different values of the granularity value: $0.2, 0.5, 1.5, 2.0, 5.0, 10.0$. Before defining the parameters p_i the user can choose one of the 7 procedural models to match his desired granularity.

Random Object Variation: Besides the possibility of editing the mesh by directly changing the parameters of the defined p_i, the user can set a minimal and maximal range for the size, length and rotation values for each p_i. This possibility enables the generation of random object variations. The parameters p_i are randomly varied within the defined parameter range. If desired, the user can fix a p_i with the current values and only randomize the residual parameters.

2.7.9. Evaluation and Discussion

We evaluate our approach by computing results for various objects. We present and discuss the generation of the procedural model itself and the subsequent object variation with parameter definitions.

Object	Skeleton		Procedural Model	
Name	Granularity Value	Number Of Vertices	Number Of Extrusions	Average Surface Distance
Dragon	0.2	507	319	0.015
Frog	0.2	223	88	0.049
Dolphin	0.2	290	200	0.011
Fish	1.5	423	223	0.024
Cat	5.0	387	312	0.010
Camel	1.5	502	376	0.012
Teddy	0.2	267	115	0.033
Spring	0.2	612	588	0.006
Wolf	5.0	381	260	0.012
Ant	0.2	441	355	0.008
Hand	0.5	388	290	0.014
Vase	0.5	717	406	0.016
Chair	2.0	625	459	0.011
Bike	0.2	1059	976	0.004
Armadillo	1.5	576	364	0.023

Table 2.7.1.: Properties of the shown procedural model generation results. The average surface distance shows the difference from the original object to the subdivision surface of the procedural model.

Procedural Model Generation: We present the results of 15 different objects in Figure 2.7.5 and the initial example object in Figure 2.7.1. For the 15 objects we present several properties in Table 2.7.1: The specific skeleton granularity value, the number of skeleton vertices, the resulting number of extrusions of the procedural model and the average surface distance of the original object to the resulting subdivision surface. The surface distance was determined by computing the Euclidean distance of every vertex of the original object to the nearest point of the subdivision surface. The average surface distance gives a measure to determine how good the subdivision surface approximates the original object.

The examples show objects with more and with less paths of small extrusions. Split parts mostly include around 2-5 face extrusions. Split parts are necessary but generally the smaller extrusions at the paths give more control. We can see that the cat, the dolphin, the camel and the spring are mostly

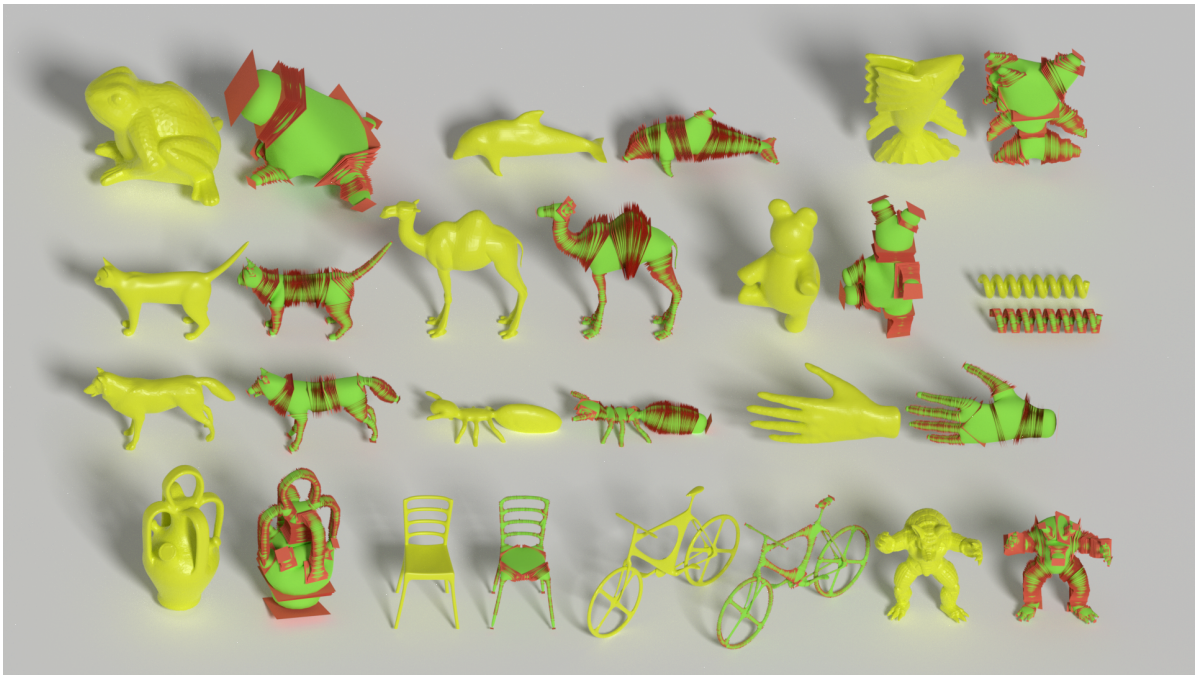


Figure 2.7.5.: The yellow objects are the original objects and the red quads and green objects show the generated procedural model. The red quads are target quads of the face extrusion operations and the green surface shows the subdivision surface of the resulting quad mesh.

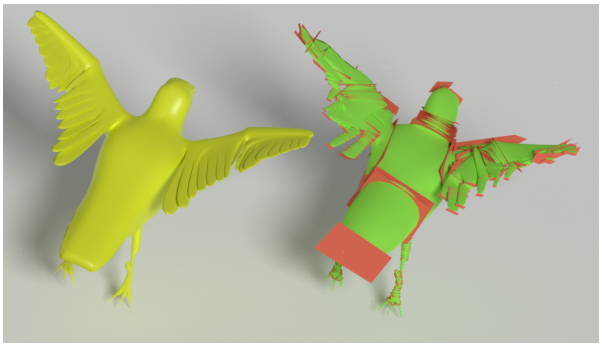


Figure 2.7.6.: In the original mesh the feathers intersect each other. At such unclean meshes our approach cannot uniquely reproduce the structure of the feathers.

densely covered by extrusions. They also have a low average surface distance. The frog and the teddy include less parts and larger split regions resulting in the highest average surface distance of all examples. The bicycle and the chair have many thin parts and are well approximated by the procedural model offering much control. Both have a very small average surface distance. Generally, bulky objects are more difficult to reconstruct with the procedural model as many quads intersect within the split parts. Loops in the skeleton cause no problems for the reconstruction, as the chair, vase and bicycle include many loops. The procedural model is suited for many different forms and could even reproduce the rather sharp angled seat of the chair.

In Figure 2.7.6 we show a limitation of the procedural model generation. The original mesh includes many thin feathers which also intersect each other. The skeletonization could not exactly reproduce a branch for every feather. Also the intersections of the feathers disturbed the quad generation. The approach is dependent on a clean mesh representation and structure. The skeletonization is also dependent on a closed and manifold mesh. Hence, a current drawback is that the approach is not completely robust against unclean meshes. As a solution we should include further preprocessing specifically preparing the mesh structure for the skeletonization and procedural model generation.

Randomly sampled split vertices: In our approach the best sequence of extrusions for split vertices is determined by evaluating all possible sequences. In thorough experiments we could not find any robust heuristic to directly calculate a good sequence. With n adjacent paths, there are $n!$ possible sequences. When the number of adjacent paths is ≤ 7 we compute all $7! = 5040$ sequences and evaluate them with the z-buffer difference. However, when $n > 7$ we compute 5040 random samples of all possible sequences instead. We evaluate if the randomly sampled split vertices with $n > 7$ achieve good results.

We analyzed the results of split vertices with $n \leq 7$ and observed that around 1% to 15% of all sequences have a sufficiently good result with a z-buffer difference similar to the best possible sequence. We reason that in general there are multiple options with equal or nearly equally good results. Therefore, randomly sampling the possible sequences is a valid option to find a reasonable good sequence.

For a further evaluation we analyzed a split vertex with $n = 9$ $(9! = 362880)$. The analyzed split vertex is the central split vertex of the vase object (See Figure 2.7.5). We analyzed the resulting z-buffer difference for 5 attempts with 5040, 1000 and 100 samples. The results are shown in Figure 2.7.7. We see that 4 of 5 attempts with 5040 samples achieved the optimal or near-optimal result. Also, 3 of 5 attempts with only 1000 samples achieved a near-optimal result.

In general, a sequence with a normalized z-buffer difference of > 0.90 was sufficient to achieve a reasonable good result. Therefore, all attempts with 5040, all attempts with 1000 samples and 3 of 5 attempts with only 100 samples achieved good results.

The presented results confirm our assumption that there are many possible sequences with good results. Therefore, it is valid to only take a sample of all possible sequences to achieve a reasonable good result.

Skeleton influence: Our approach is based on a preceding skeletonization. Therefore, we also analyzed the influence of the skeleton on the final result.

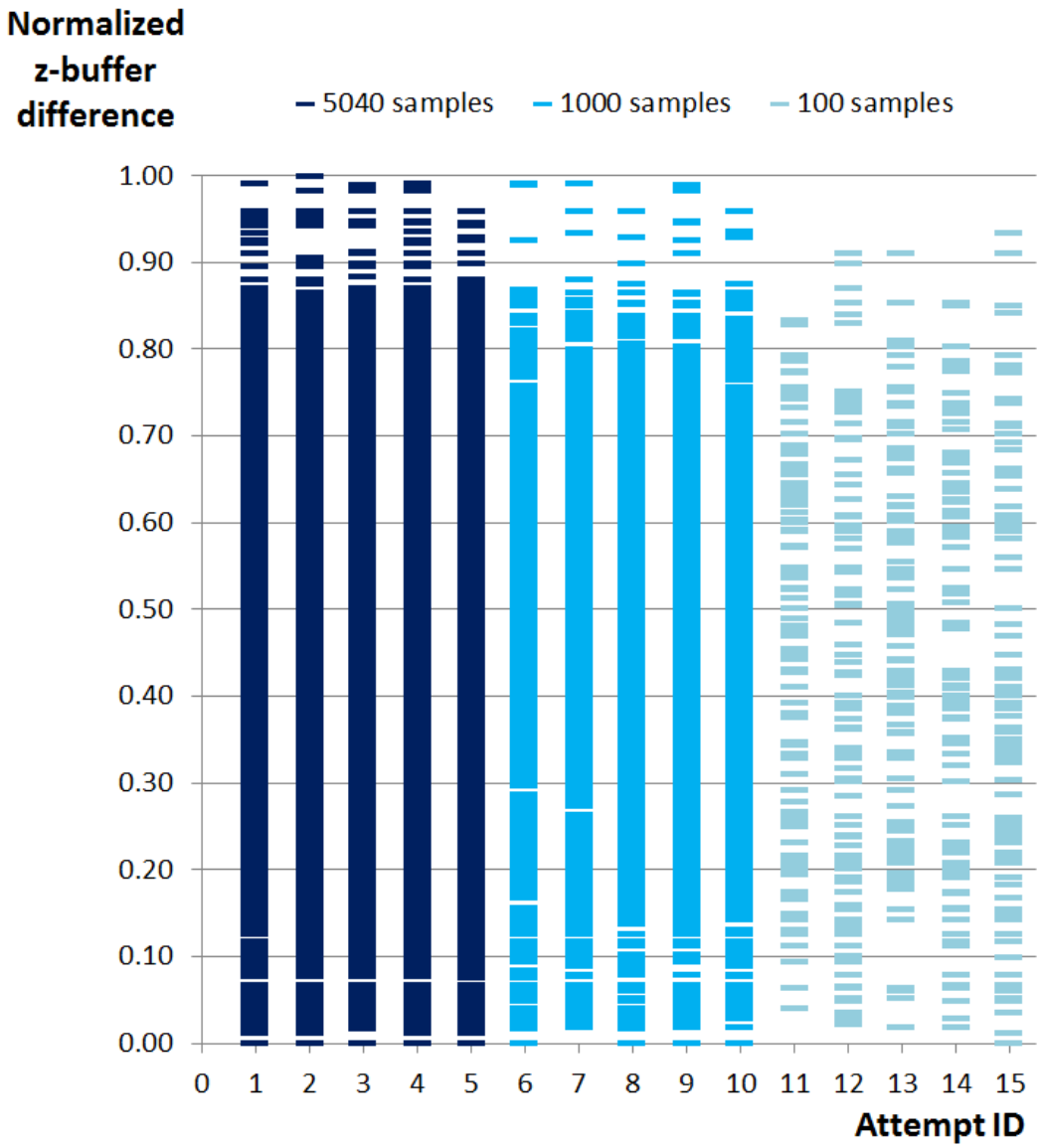


Figure 2.7.7.: The results of 5 attempts with 5040, 1000 and 100 samples of a randomly sampled split vertex with 9! possible sequences. The z-buffer difference is globally normalized on the basis of all results, so that the best found difference has the value 1.0 and the worst has the value 0.0.

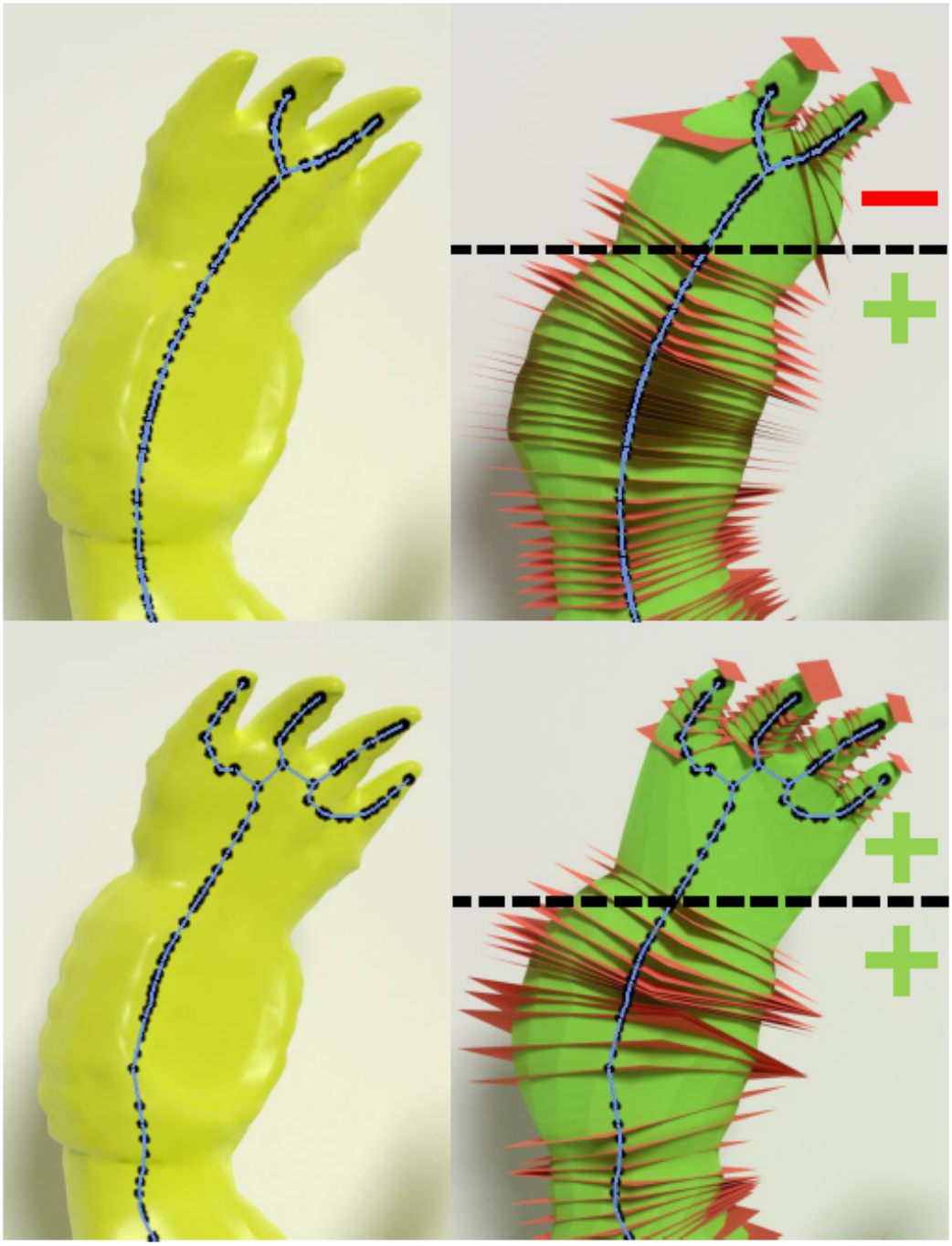


Figure 2.7.8.: An arm with a hand of the armadillo with different skeletons. The skeleton on the top was generated with a granularity value of 0.2 and the skeleton on the bottom with a granularity value of 10.

In general, our approach is limited to the cases with reasonable skeletons. An erroneous skeleton will not lead to a good procedural model. When the skeleton quality is reasonable well the level of precision may vary. In Figure 2.7.8 we present an example of different skeletonizations illustrating the properties of the influence of the skeleton on the resulting procedural model. The example shows an arm with a hand of the armadillo.

We can see that the hand is not reproduced correctly in the first case. We reason that the branching of the skeleton directly influences the resulting split procedures. The quad decimation step only reduces the number of quads, which can result in a reduction of splits. It never adds new splits. With the granularity of the skeleton we decide about the resulting granularity of the procedural model. Smaller details which are not represented by the skeleton are not considered for separate branches. In this case the granularity was chosen very low, so that 2 fingers of the armadillo were treated as minor details.

Nevertheless, the example also shows that the form of the arm of the armadillo is approximated correctly with both skeletons. Irrespective of the exact number and position of the skeleton vertices our algorithm is able to produce a good combination of quads to reproduce the form of the arm. Note that a severely sparse skeleton with very few vertices could result in a bad approximation. However, in all tested cases the number of vertices were sufficient. Therefore, this is not a major limitation. Furthermore, additional vertices could be generated on the skeleton edges if needed.

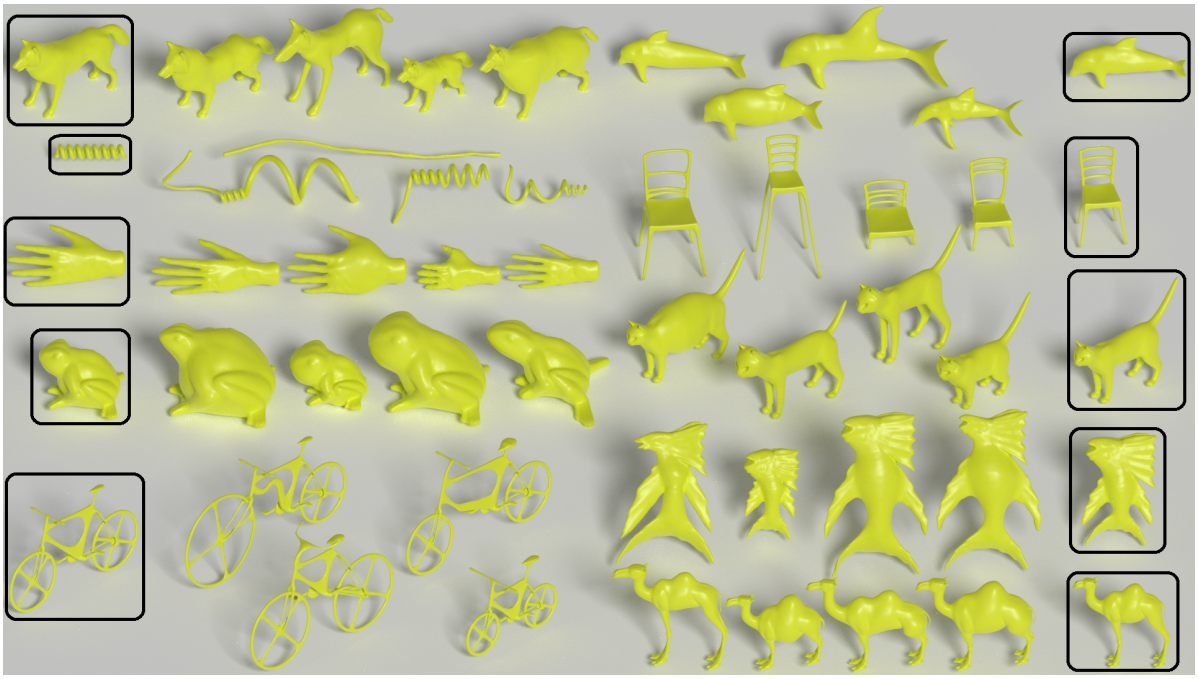


Figure 2.7.9.: On the left and right we show the original 3D objects without changes. In the middle we see variations of all objects produced by editing the procedure parameters of the procedural model.

Object variation: We show several examples of using the procedural models for object variations. In Figure 2.7.1 a total of 14 parameters have been defined. The resulting variations of the dragon are generated completely random. Figure 2.7.9 shows variations created by directly changing parameter values. Figure 2.7.10 shows examples of random variations using 4-5 parameters.

In Figure 2.7.9 we demonstrate the possibilities of the parameter definition by generating several variations for 10 of the example objects. For these examples we initially have chosen the skeleton granularity value and the number of influence points. Then we defined multiple parameters p_i. By changing the size, length and rotation values of the parameters p_i we could edit the form and shape of the objects.

In Figure 2.7.10 we show randomly produced variations of 4 of the example objects. We defined several parameters and randomly generated multiple objects. For each defined parameter we show all extrusions that are influenced by this parameter. We visualize the extrusions of a single parameter within the black bordered region: The original object is shown in yellow and all extrusions of the single parameter are shown as red quads.

In both examples the variations keep the characteristics of the original objects, while looking natural and well formed. Manually defining the values of the parameters gives a tight control. The random variation of defined parameters within a set range leads to a large variety of forms. Nevertheless, the

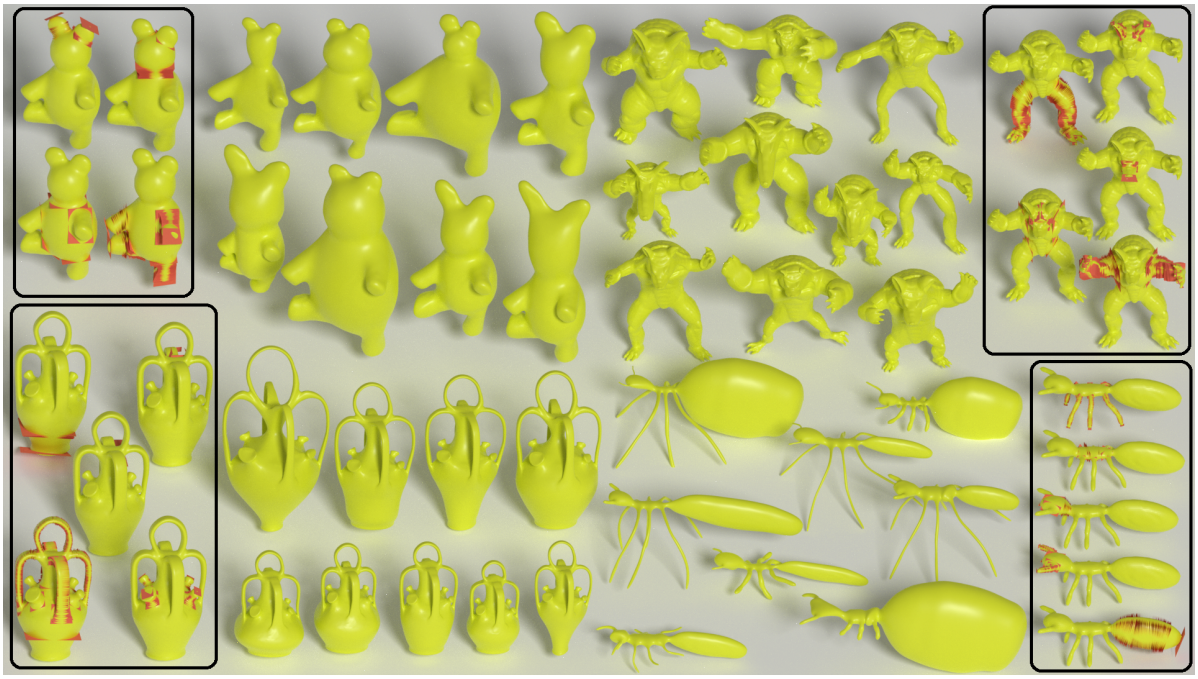


Figure 2.7.10.: On the left and right the yellow objects with intersecting red quads show the original 3D objects. The red quads represent all extrusions that are influenced by a single parameter. The teddy has 4 parameters and the residual objects have 5 parameters. In the middle we see random variations of all objects, generated with the defined parameters.

quality of the resulting variations are dependent on the parameters and the parameter ranges defined directly by the user. Bad parameter choices can lead to unnatural results. In Figure 2.7.11 we show a bad choice of the number of influence points. The left object shows the original object with the red quads representing the relevant extrusions. The size parameter of these extrusions were diminished to a small value. The object in the middle shows the result with 1 influence point. The right object shows the same size change with 400 influence points. The object in the middle has visible artifacts and an unnatural transition from the thick part to the thin part. In Figure 2.7.12 we show a bad parameter choice. The left object shows the original object with the red quads representing the relevant extrusions. In this case a split part. The size of the split part was diminished to a small amount. In the resulting object (shown on the right) the ears of the teddy intersect each other.

In general, the parameter definition does not hinder bad configurations. It provides much flexibility and many possibilities to the user. A solution for this problem is to provide a suggestion system to the user. Good parameter choices are suggested automatically. This can be done by measuring properties of the surface, e.g. smoothness or self-intersections.

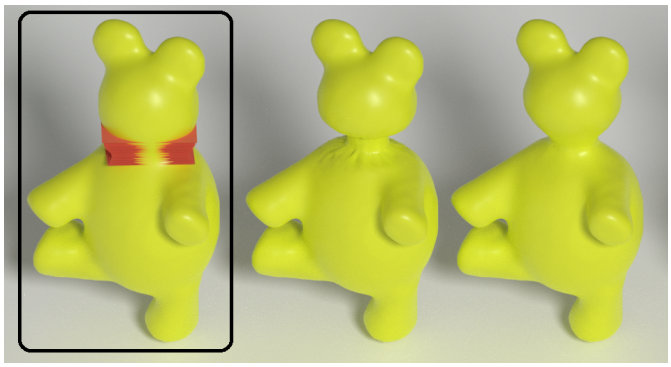


Figure 2.7.11.: For the resulting variation in the middle the number of influence points are set to 1. For the variation on the right the number of influence points are set to 400. The object in the middle has visible artifacts and an unnatural transition from the thick part to the thin part.

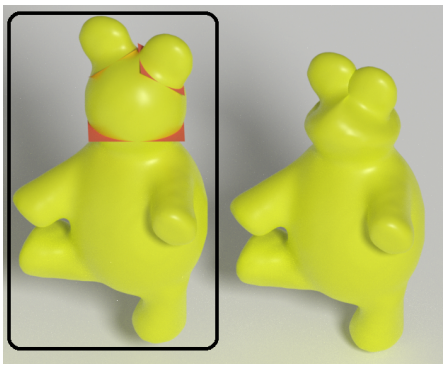


Figure 2.7.12.: The head size has been diminished heavily, resulting in an intersection of the ears of the teddy.

Comparison with other approaches: Our procedural model approach offers the possibility of directly inserting parameters and randomly varying the shape of different parts of the object. The advantage in comparison to other approaches is that our smallest construction part (a single extrusion) naturally defines a direction, a thickness and a length. The complete procedural representation results in a consistent and homogeneous construction of all parts of the object. This is different to sphere-mesh [TGB13], a cylinder decomposition [ZYH*15] or a part-based recombination [JTRS12].

The sphere-mesh [TGB13] represents an object with a loosely coupled aggregation of spheres. The user can manually drag a sphere to deform the original object. The dragging of the spheres and the deformation transfer to the original object is easy and intuitive. However, the spheres do not have an intrinsic definition of the structure of the object. Any sphere can be dragged to any direction. The approach is solely targeted towards the enhancement of manual editing. Therefore, the insertion of natural varying parameters is not directly possible.

The cylinder decomposition [ZYH*15] includes two steps. First, the object is decomposed into an over-complete set of cylinders. Then, a global optimization decomposes the object into distinct parts. Each part is related to his set of cylinders. In contrast to the sphere-mesh, the cylinder also have a natural definition of direction, thickness and length. However, the cylinder decomposition does not result in a complete consistent representation of the object, but instead separate the object into unrelated parts. In the first step the cylinder decomposition is over-complete and therefore the cylinders of all parts intersect each other at split parts. In the second representation the object is spitted into parts, resolving the intersections of the cylinders. The intersection in split parts is resolved by assigning the split part to one neighboring part and separating it from all other parts. Therefore the cylinders are only aligned to one neighboring part. In our approach the procedural model forms a complete consistent and intersection free representation of the object.

A part-based recombination offers the possibility of randomly generating new variations of objects. The results presented by Jain et al. [JTRS12] have a high quality and demonstrate versatile variations of objects. However, the approach is fully dependent on the availability of a versatile and high qualitative complete set of objects. All variations are recombined from the detected parts of the available objects. Arguably an extensive detailed set of objects is rarely available. Furthermore, the resulting variations can only include the parts of the initial set. This might not match the desired outcome. In our approach it is possible to define the desired outcome of the random variations and it is also possible to refine the parameters of the resulting object manually.

In sum, our approach offers natural variation possibilities that no other approach offers. It is possible to use the procedure parameters for random variations as well as for manual editing.

Variation possibilities: We have shown that other approaches are rather limited in the possibilities of parameter integration for random variations. Our approach offers the possibilities of directly using the naturally included parameters of the construction. Nevertheless, sophisticated editing tools using complex shape deformations give limitless variation possibilities. The automatically offered parameters of our approach come with the drawback that the variation is restricted to the offered parameters. The quads are oriented according to the natural direction of the object surface. Therefore, only varia-

tions within this directions are possible. Nevertheless, this is a fair limitation for the task of variation generation, as random variations not obeying the natural directions would rather lead to unnaturally arbitrary deformed shapes.

Parameter ranges: The range of the parameters for the random variations are directly defined by the user. The range is defined for every parameter individually. In general, it is possible that there might be parameter combinations that are not desired by the user. The user might select a maximum for two different parameters, with the intention that each might vary up to this range but these two parameters should not both be at the maximum together. We considered to let the user define conditions and dependency rules of the parameters. However, the definition of rules by the user includes a high effort for relatively low benefit. In our test cases it was sufficient to choose a relative large range for all parameters. New variations can be generated with a single click and the user can skip all undesired combinations. Also, as described in section 2.7.8 we offer the possibility of fixing single parameters during the random variation generation. When a variation has a undesired combination of parameter values, the user can fix one or multiple parameters and only continue to vary the residual to gather desired variations.

Deformation quality: In the final part of our evaluation we compare the meshes resulting from our procedural model approach with the result of a manual deformation using the as-rigid-as-possible deformation from Sorkine and Alexa [SA07] with the energy function for elastic deformation of Chao et al. [CPSS10].

For the deformation no additional structure is needed. It is directly computed on the mesh. However, the deformation needs the definition of a region of interest and control vertices. Both are a set of vertices of the mesh. The region of interest defines the region which is considered for the deformation. All vertices that are not part of the region of interest keep their position. The control vertices define the region which is picked for the deformation. The region defined by the control vertices can be dragged and rotated for the deformation.

The results of the deformations are shown in the Figures 2.7.13 and 2.7.14. The orange objects are the result of the manual deformation. For the comparison we show a result of our approach as yellow object.

Figure 2.7.13 shows the stretching of the left arm of the armadillo. The hand was picked for the control vertices and dragged to the right of the front view. For the deformation of the top armadillo the whole object was chosen as region of interest. For the deformation below, only the arm was chosen as region of interest. The yellow object presents the result of our approach by increasing the length parameters of the arm.

We can see that the choice of the region of interest is very important for the result. Only when choosing the arm as region of interest the stretching of the arm is produced as expected. However, choosing all vertices of the region of interest includes a relevant user effort. In our approach the user can pick the arm with a single click. The quality of the stretching of the arm is comparable with both methods. The difference is that our approach keeps the initial curvature of the arm when increasing the length parameter.

Figure 2.7.13.: The orange armadillos result from a deformation based method. For the top orange armadillo the arm was stretched with the whole object as region of interest. For the bottom orange armadillo only the arm was chosen as region of interest. The yellow armadillo is generated with our procedural model approach. The stretching of the arm is similar to the deformation based method having only the arm as region of interest. The difference is that our approach keeps the initial curvature of the arm.

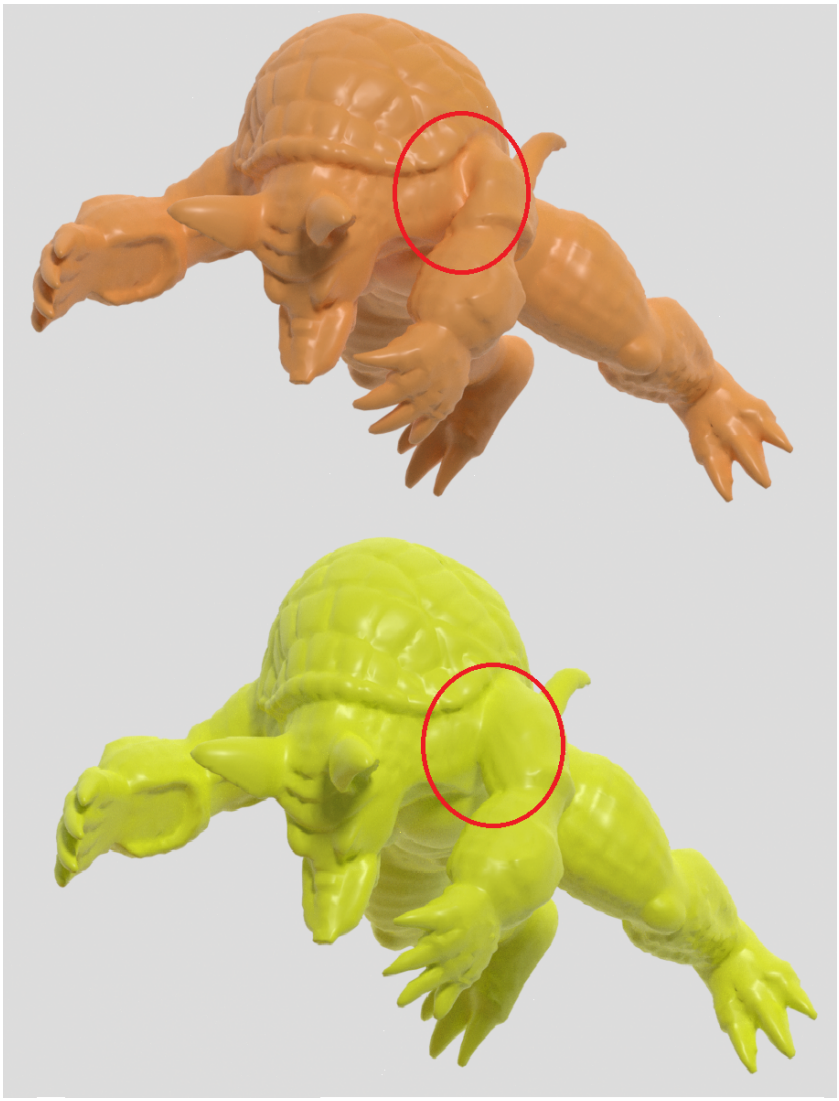


Figure 2.7.14.: The orange armadillo results from a rotation of the arm with a deformation based method. The yellow armadillo results from a parameter change with our procedural model approach. Within the red circle we can see a bending artifact of the deformation based method. Our approach produces a bending of the arm without artifacts.

Figure 2.7.14 shows a modification of the direction of the arm. For the deformation the lower part of the arm was chosen as control vertices and the upper part of the arm was chosen as region of interest. For our approach we modified the two rotation parameters of all procedures of the upper part of the arm.

The results show that the modified bending of the arm is correctly generated by our approach. The manual deformation method produces a bending artifact at the arm pit, marked with a red circle.

In sum, our approach can reproduce bending and stretching results from a deformation based method. Furthermore, our approach includes less user effort and the results are more robust.

2.8. Conclusion – Automation and Generalization of Procedural Models

In this part, we have introduced several possibilities of automating and generalizing the generation and parameterization of procedural models. These possibilities enable the usage of parametric procedural models for broader applications. Applications like 3D object retrieval and classification have multiple use cases where the burden of a very complex manual procedural model construction for each single case is non-viable. In fact, the amount of feasible user effort in these applications highly differ from case to case. For this reason, a flexible construction, generation and parameterization of procedural models is vital: To enable the full potential of the parametric procedural model representation the control of the user should be as high as possible. At the same time the automation should be as high as the user demands.

The first key concept is the generalization of the procedural models. A procedural model can be highly specialized, so that each procedure of the model is tailored towards a specific object class. We were able to describe full procedural models with a small set of general procedures. Complex shapes could be described as a combination of several general procedures. For the semi-automatic model generation we transformed a small set of modeling operations into general procedures, suitable to describe the construction process in a flexible way and therefore suitable to represent a procedural model. We were able to show that this general representation is able to produce arbitrary objects and also, the procedures are parameterizable in various ways. Furthermore, in the fully automatic procedural model generation we have shown that complex structures are reproducible with an even smaller set of procedures. Overall, we were able to proof that highly generalized procedural models are capable of unfolding the full potential of procedural models.

We have also generalized the parameters, as we defined fixed ranges and standard insertion principles. We could show that already in the manual definition of procedural models a fixed range with a fixed set of parameters gives a multi-dimensional object space for an object class. In the semi-automatic model generation as well as in the automatic model generation the fixed ranges and insertion principles allowed a regulated parameterization for a clear definition of the possible object variations. The complex parameter sketching has shown an important property of the generalized parameters: It shows that an arbitrary deformation can be translated into a parameter variation. Therefore, demonstrating that the parameters of general procedures are able to reproduce any shape variation.

The second key concept is the automation of the procedural models. The automation is based on the generalization, as only the generalized procedures allow to formulate an automatic process, which works for each procedure. We were able to formulate a completely automatic process to generate a pro-

cedural model from a single example. Also, we proposed an approach for automatic parameterization of a semi-automatically generated procedural model. Therefore, the generation of procedural models as well as the parameterization are suited for automation. The semi-automatic model generation and the complex parameter sketching have shown that the user can still have a high level of control, while the underlying procedural structures are generated and parameterized automatically. Overall, we have reached our goal as we have found good trade-offs for each level of automation.

Part 3.

3D Object Retrieval, Classification and Parameterization

3.1. Overview

The applications of 3D object retrieval and 3D object classification are both based on the comparison of 3D objects. In 3D object retrieval objects are sorted by similarity and in 3D object classification the similarity to a class is computed. The basis to these applications is the definition of similarity of 3D objects. This is a highly non trivial definition, as the similarity of arbitrary objects is not only a question of geometry, but also a question of the semantic of an object. The similarity of objects is lastly dependent on the human notion of similarity, which is not formalizable. Nevertheless, geometrical, visual and topological characteristics of an object are measurable and correlate with the human notion of similarity. Therefore, many successful approaches have shown that the deduction from the geometrical similarity to the semantic similarity is viable.

However, the requested notion of similarity can differ from one application to the other. Specifically, the degree of comparison can be different. In technical manufacturing the measurement of similarity of objects is concerned about small deviations from one object to the other. In zoology we might want to measure the similarity of different breeds of cats or dogs. But in a different scenario we might just want to recognize if the animal is a cat or dog at all, so that we just measure the general similarity of arbitrary animals.

Summarizing, for each specific application the chosen similarity measure has to be chosen appropriately. In the following, we review different 3D object similarity measures. We also review so-called descriptors, which can be described as higher-order similarity measures.

A descriptor describes an object in a short manner, so that objects can be compared simply by comparing their descriptors. Descriptors can be based on different proprieties of objects. They can be based on topological proprieties, geometrical features of the surface or just general appearance properties.

For the application of 3D object retrieval, descriptors are directly used to efficiently measure the difference between the query object and every other object. The resulting retrieval list, is a list of all objects sorted by their similarity. In the basic case the retrieval is performed completely unsupervised. However, several techniques (e.g. relevance feedback) have been developed to improve the retrieval results with additionally provided information. We analyzed the possibilities of using procedural models as additional information directly for the retrieval and propose a 3D object retrieval technique for procedural models based on descriptor comparisons. Furthermore, we propose a technique to use deep learning with procedural models for this task.

For 3D object classification we use procedural models as data basis for the deep learning of classes. Furthermore, we propose a new system to estimate the parameters of a procedural model to optimally

fit to an unknown 3D object of the same class. Hence, an unknown object is not only classified but also the characteristics of the object are quantified by the parameterization estimation.

In the following Part 3 of the thesis, we present all contributions related to the similarity of objects, 3D object retrieval and classification. We propose a new local distance measure, to analyze local surface deviations and propose a new technique for 3D object retrieval based on a hierarchical clustering. Then, we introduce our descriptor based 3D object retrieval technique for procedural models. Furthermore, we propose a deep learning based approach using procedural models as data basis. Here we show the usability for 3D object classification and 3D object retrieval. In the last section we present our novel parameter estimation system. For an unknown 3D object we are able to find an optimal parameter set of an associated procedural model.

3.2. Related Work

We review four related areas. First, we review 3D object surface similarity measures, focusing on direct surface to surface differences. Second, we review descriptors in general. In the third section we discuss methods related to 3D object retrieval. In the last section we review classification approaches for 3D objects.

3.2.1. 3D Object Surface Similarity

3D object surface similarity measures can be separated into global and local measures. Global measures provide a single distance value for the comparison of two surfaces. Local measures determine values per mesh vertex.

The most well-known local distance is the Surface Distance. The Surface Distance of a point is defined as the Euclidean distance to its nearest point of the other surface. Many global measures are based on the Surface Distance [Gue01]. They aggregate (maximum or average) the local Surface Distance values. The maximal Surface Distance from one surface to another is the Hausdorff Distance. If the Surface Distance is considered in both directions then the maximal value from both directions is called the symmetric Hausdorff Distance [ASCE02].

Global measures can also be based on the volume of the objects. Volume comparing measures include the Volumetric Overlap Error [VGHS07] and the Relative Volume Difference [HvGS*09]. They measure a ratio of the shared part to the non-shared parts of the two objects.

Other global measures focus on curvature and form differences of the meshes using, e.g., normal fields [CSAD04] or the energy metric [Hop96]. The result can only be compared per mesh and not local per vertex.

Ray-casting based measures [SvHVG*08,CK97] are local measures that calculate a *relative error* by casting rays from inside or outside the mesh. These measures are highly dependent on the form of the meshes. For example, the radial distance [CK97] is suitable only for sphere-like formed meshes. The distance by Strecha et al. [SvHVG*08] casts rays from rather randomly chosen points. The concept of ray-casting based local distances is similar to z-buffer distance measures, as the z-buffer distances basically measure the distance of a ray cast trough a pixel. The difference is that the final distance values are not present locally on the mesh but rather on the pixel positions. Therefore, z-buffer distances have to be aggregated to a global measure.

There are also local measures, which are based on a surface parameterization. Both surfaces are associated with a range of values and the distance of two points with the same values can be calculated,

e.g., The Fréchet distance [VH99]. A consistent parameterization depends on the mesh form. For every case, a different suitable parameterization method is needed.

Similar to the parameterization methods, other local distance measures rely on matching two surface points of both meshes. This problem is related to finding point-to-point mesh correspondences [VKZHCO11]. Often a dense point correspondence is computed from several corresponding feature points, which must be known in advance [VKZHCO11]. Kraevoy et al. [KS04] require user-defined basic corresponding points. Cates et al. [CMF*06] move particles along the surface for finding corresponding points. This is calculated automatically but restricted to smooth surfaces allowing for free particle movement.

Another concept to find correspondences is to deform one mesh into the other, so that the points lying on top of each other can be marked as corresponding. Non-rigid registrations are focused on small local deformations [BR07]. Other methods use additional registered scan data for global deformation [LSP08, HTB03].

A corresponding point can also be found by using the shape context [BMP01] or spin images [Joh97]. These describe the distribution of surrounding points. Corresponding points have similar surroundings. Corresponding points can also be detected by measuring the biharmonic distance [LRF10]. However, these methods only work when the surfaces are similar to each other [MGG15, WMWL15]. Otherwise the correspondence is not detected. When we want to measure the differences based on corresponding points, the correspondences cannot be based on similarity.

3.2.2. Descriptors

Descriptors are 3D object similarity measures themselves. The difference is that descriptors are computed for a single 3D object and not for two objects together. A descriptor should describe the characteristics of a 3D object. For the measurement of the similarity of two 3D objects, the descriptors of both objects are compared instead of directly comparing the objects.

The survey of Tangelder et al. [TV08] separates descriptors into feature based, graph based and geometry based descriptors.

Feature based descriptors include two different approaches. The first approach measures features over the whole surface and merge them to a global descriptor. The features can be based on, e.g., D2 distances [OFCD02], spherical harmonics [KFR03], oriented gradients [SWS10], 3D zernike polynomials [NK03] or density functions [ASYS09]. The second feature based approach only integrates (sparse) local features into the descriptor. This can happen in a bag-of-words manner [BBGO11, TCF09] or SIFT-based [DK12, MFK*10] or SURF-based [KPW*10].

Graph based descriptors describe the topology of an object as graph, e.g., as skeleton-graph [SSGD03] or a topology graph [MSF07].

Geometry based descriptors includes the residual, but the most relevant are the view-based descriptors which describes the object from 2D views, like the LightField descriptor [CTSO03]. There are

also hybrid approaches combining view-based with feature-based, like the DESIRE descriptor [Vra05] and the panorama descriptor [PPTP10]. In a comparison in 2015 [LLL*15] the panorama descriptor is considered to be one of the best geometrical descriptors. Therefore we use this descriptor for object comparisons with procedural models.

3.2.3. 3D Object Retrieval

In the completely unsupervised case a user chooses a query and the query is compared to all objects of a database. The result is a retrieval list including all objects of the database sorted by similarity. For the comparison any descriptor can be used. Therefore, improving the accuracy of 3D object retrieval often just means developing a better descriptor. However, there are also several techniques for improving the results of 3D object retrieval which can be used for any descriptor. Mostly these techniques include additional knowledge of any type. For example additional knowledge as class information, labeled data, user feedback or other additional algorithms with fine tuned parameters. In this section we focus on 3D object retrieval techniques outside of descriptor improvement.

Several techniques use user interaction for significant improvement of retrieval accuracy. A very popular technique is relevance feedback [LMT05, ASYS10, GWJ*14, LZYX15]. In relevance feedback the user can label results as correct or wrong to refine his query. The resulting retrieval list is recomputed based on the user feedback. The retrieval performance is raised tremendously by this technique. Effectively, the query of the user is much more accurate when he is able to give several examples showing what type of object he is searching for or not searching for.

User interaction can also be used in a different setting. Patterson et al. [PIMD08] lets the user choose a part of an object which should be retrieved. The part is then searched and retrieved within the object itself. In the work of Sunkel et al. [SJWS13] the user labels the searched objects in a range image training scene and afterwards the same objects are retrieved in a new scene.

Papadakis et al. [PPT*08] evaluate several relevance feedback techniques and also describe a 3DOR technique which is completely independent of the descriptor and does not require any additional knowledge. This technique is called PRF (Pseudo-Relevance-Feedback). In this technique the user feedback is simulated by taking the n nearest neighbors as correct, relevant results. They then refine the result list just like the user picked this n objects as correct. They show that the retrieval results are improved by taking the 4 nearest neighbors as correct results. However, 4 is not always a good choice for n. The parameter n has to be defined by the user and is much too dependent on the database and the single case.

Tatsuma and Aono [TA09] describe a 3DOR technique which is completely independent of the descriptor. They describe an approach where the database is clustered without any additional knowledge. The cluster information is then combined with the descriptor distances to compute the final distance to each object. They show that using the clustering with their descriptor has a significantly higher retrieval performance than only using the descriptor. They use a k-means clustering in spectral space and generate a single clustering of a whole database. For this reason they need to choose the number

of clusters k and also configure other parameters dependent on the database. Their results show that the accuracy strongly relies on a good choice for k. This approach is related to our proposed retrieval technique based on hierarchical clustering (Section 3.4). The major difference is that we resolve the problem of the parameter dependence by proposing a completely parameter-free hierarchical technique for the clustering and subsequent improvement of retrieval performance.

Gong et al. [GXLT09] describe a technique to boost 3D object retrieval by combining a descriptor with a new object flexibility descriptor. They show that the combination with their descriptor leads to a better performance of several descriptors. This is often the case when multiple descriptors are focused on different aspects of the object.

An important part of many descriptors is pose normalization, where the 3D object is normalized, so that the descriptor is translation, scale and rotation independent. Dutagaci et al. [DSY10] and Lian et al. [LRS10] have shown that the improvement of the normalization can effectively improve the performance with several descriptors. Though, this technique corresponds to an improvement of the descriptor itself.

Funkhouser et al. [FMK*03] change the query to a 2D sketch from the user. Therefore, this approach uses additional information by including more user interaction during the query definition. With this technique the user is able to express the type of object he has in mind more accurately. Still, the additional effort for sketching the object is not always desirable.

Other techniques use templates for the retrieval. Biasotti et al. [BGM*07] construct structural prototypes during a learning phase and use these to improve the retrieval. Template based approaches are not only developed for 3D object retrieval, but also for the exploration of collections and creation of new combined objects. Averkiou et al. [AKZM14] extract part-based templates from collections of objects and synthesize them to new combinations of object parts. Then, the user can not only retrieve the initial objects but also new combinations of them. Jain et al. [JTRS12] co-segment two objects and produce different combinations of the two objects. Xu et al. [XLZ*10] generates correspondences between parts of objects and then combines and scales these parts, deriving new objects. Osjanikov et al. [OLGM11] construct flexible part-based template models from a set of objects. During the retrieval the user can define the location of the parts of the template to retrieve the object with highest similarity to the chosen template configuration.

Part-based templates are typically more restricted than procedural models. Procedural models can also include complex geometrical properties. Therefore, our approach is a more expressive representation than part-based templates. This is especially interesting for higher semantic concepts which can include a broad variety of different parts.

Anther way of improving 3D object retrieval performance is using class information to learn classes in advance. The similarity of the query and a database object is then not only measured by a descriptor but combined with the classification of the object to improve the retrieval results. Hou et al. [HLR05] use a support vector machine to learn a model and then retrieve objects similar to this model. Note that this case represents an overlap of 3D object retrieval and 3D object classification.

Li and Johan [LJ13] propose a method called CBR (class based retrieval) which can be used with any descriptor. The method uses the class information of a database and measures the distance of an object to each class. The final distance to an object is a combination of the object distance and class distance. They show that the retrieval results are significantly improved by using CBR. Wang et al. [WLPL15] use a bag-of-words based descriptor and integrate the information from class labeled data into the computation of the descriptor. Depending on the class information different feature sets are chosen for the retrieval.

Summarizing, many approaches show that the integration of additional information into the 3D object retrieval process is able to tremendously raise the retrieval performance. Especially, because the additional information is specified or chosen by the user, which therefore can express more accurately what type of object he is searching for. Hence, our 3D object retrieval with procedural models correspond to the integration of additional knowledge chosen by the user.

A big advantage of procedural models in comparison to other approaches is the persistence of the additional knowledge. A procedural model can be created and parameterized once and reused for any query for any database. It is even possible to recombine and edit existent procedural models. Therefore, procedural models give huge possibilities of improving retrieval and classification in a long-lasting fundamental way. The vast majority of the available techniques are targeted towards a non-persistent single query improvement. Techniques, which are based on user interaction or parameter configuration during or after the result computation, generate non-persistent additional information, since this information is only valid for the specific ongoing query. Other techniques, which use additional information provided in advance, are non-persistent when the information is tailored towards the specific database. The additional information is not valid for other queries of other databases. Techniques that generate persistent additional information are typically targeted towards the formulation of a more informative query constructed in advance. This includes techniques of user based query formulation, like the 2D sketch from Funkhouser et al. [FMK*03] (since the 2D sketches can be reused and are non database dependent) and several part-based template techniques [BGM*07], [AKZM14], [OLGM11]. However, 2D sketches and part-based templates are only rough approximations. None of these techniques offer the intrinsic expressiveness and flexibility of a procedural model.

3.2.4. 3D Object Classification

In 3D object classification we use labeled data to learn class labels in advance. In general, the goal is to gather new information about an unknown object. In real a application this can be a classification but it could also include overlapping labels or categories. However, the most usual case treated in most works is the assignment of a single class label to an object with unknown semantic, only based on the polygon mesh.

Li et al. [LJ13] compute a descriptor and uses prior known class information so that objects can be classified to their nearest class. Xu and Li [XL07] directly use the labeled data with class labels for a neural network learning. New objects are then classified by the neural network. Ip et al. [IRSS03] also

use a machine learning approach with labeled data for the model classification. Hou et al. [HLR05] uses a clustering to improve the retrieval performance. Data with class labels is used to learn the classes and then cluster the database accordingly.

Van Kaick et al. [VKTS*11] analyze a set of labeled objects representing one class to learn a consistent segmentation and point-to-point correspondence over the training set. New objects of the same class can then be segmented accordingly and point-to-point correspondences can be established.

For the 3D object classification the descriptors have also been used to directly learn single classes of 3D objects [WBK08]. Also, bag-of-words approaches [WLPL15] achieved high accuracy in classification.

However, deep learning has mostly outclassed these approaches for the classification task. The drawback of deep learning being, that a large amount of data is needed for the specific learning task.

Maturana et al. [MS15] proposed voxnet, a 3D convolutional neural network for real-time object recognition. Their algorithm transforms 3D point clouds into voxel data within a grid of 32x32x32. This voxel grid is used as input for a convolutional neural network (CNN) which directly learns the shape of objects in 3D. Wu et al. [WSK*15] also directly use 3D data transformed into a voxel grid to learn a convolutional deep belief network with the resolution of 30x30x30.

Su et al. [SMKLM15] developed a multi-view CNN for 3D shape recognition. A CNN is learned on rendered images from various views of a 3D mesh. This approach uses images of 3D objects instead of working directly on the 3D data. With this approach they achieve higher accuracy than any comparable approach. The authors reason that currently the relative efficiency using 2D data is higher than using 3D representations. Using a 164 x 164 image instead of a sparse 30 x 30 x 30 voxel grid, leads to better performance when training the CNNs. Still, like many CNN approaches a large amount of data is needed. Szegedy et al. [SLJ*15] proposed the google inception network which is a very deep CNN for learning classes of images. The network was trained on a vast amount of images and achieves state-of-the-art accuracy in all image classification tasks. An additional benefit of this approach is that the last fully connected layer can easily be retrained for new classes since the amount of learned image features are numerous within the huge network. We use this network for the learning of procedural model variations.

3.3. Extended Surface Distance

The Surface Distance (SD) is the Euclidean distance from one point of the mesh to the nearest point on the other mesh. With this distance measure it is possible to locally define the difference between two polygon meshes. We can calculate an approximation of the SD for each vertex by using the vertices of both meshes. For each vertex of one mesh the SD is the distance to the nearest vertex of the other mesh.

The SD is a widely used local measure. The average and maximum (Hausdorff distance) of the SD is also often used to measure the deviation of two similar surfaces. The advantage of the SD is that it is easy and fast to calculate. Also, it is always possible to calculate the SD, even if the polygon mesh is not closed or 2-manifold.

However, the SD fails to be locally exact in some cases. It is dependent on the shape of the surface. The exact distance of two surfaces can be calculated when we have exactly corresponding points on both surfaces and measure the distance between the corresponding points. There are algorithms to compute corresponding points. However, these are dependent on specific similarities and properties of the surface and do not work for all object types.

Instead, we propose a new local distance measure to evaluate the distance of two objects locally. The new distance measure is a modification of the SD.

3.3.1. Problematic Cases of the Surface Distance

We describe the three main and two combined error *cases* of the SD, which we identified during our work. The order of the three main cases expresses their severity, with case-3 being the most severe. We explain the error cases on two meshes M^1 (orange) and M^2 (blue) in a 2D view for easier understanding (see Figure 3.3.1(a)).

We show the SD as a *distance vector* pointing from one vertex on one surface to its nearest vertex on the other surface. We show the SD for the blue surface M^2 in Figure 3.3.1(b) and the SD for the orange surface M^1 in Figure 3.3.1(c). The distance vectors pointing into the same direction of the surface normal of the vertex are colored in dark green, while the others are colored in light green. Note that we only show the vertices and distance vectors for the interesting part of the example and omit the rest.

In the examples, the SD is calculated by taking a discrete amount of vertices of the two surfaces (e.g., the vertices of a mesh). The distance of each vertex $v_i^1 \in M^1$ is determined by its nearest vertex in M^2 (see Figure 3.3.1). Note, the exact SD considers every possible point of the surface represented by

the mesh. Taking a discrete amount of vertices provides a good approximation for meshes with dense vertex coverage.

Case-1: Wrong Distance in One Comparison Direction: This problem of the SD results from the strong asymmetry (see P^1 in Figure 3.3.1). The distance vectors are correct in one of the two directions: from the blue to the orange surface. But, we see that the distance vectors are erroneous from the orange surface in the same region.

Case-2 : Nearest Vertex in an Unrelated Region: This error occurs when the nearest vertex is found in a region representing a "semantically" different part of the other mesh (see P^2 in Figure 3.3.1). The distance vectors from the blue surface are pointing to the wrong side of the other surface.

Case-3: Wrong Distance in Both Comparison Directions: The error of case-3 is shown in P^3 (see Figure 3.3.1). Since the surfaces differ only in the direction of the deformation (inward and outward), we can expect that the distance vectors connect these regions. However, the distance vectors from both sides only point to the border of the deformations. Note that in this case the distance vectors of both meshes are wrong in the erroneous region.

Case-2 errors overlaying case-3 or case-1 errors: It is possible that case-2 errors overlay case-3 or case-1 errors (see P^4 and P^5 in Figure 3.3.1). The distance vectors from the blue surface M^2 show an erroneous behavior of case-2 (pointing to an unrelated region). Case-2 error of P^4 (formed like P^3) overlays the case-3 error in this region. Analogously, P^5 is formed like P^1. In P^5, the errors of case-2 overlay the errors of case-1. These overlaying errors imply that we have to solve the errors of case-2 before identifying the residual errors.

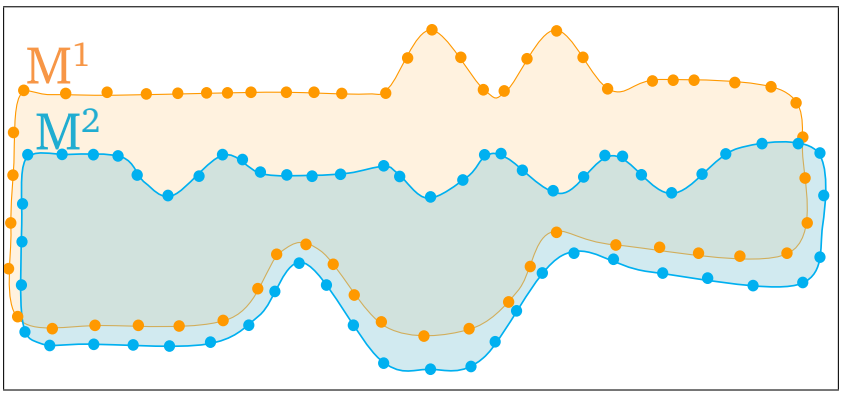
3.3.2. Extended Surface Distance Calculation

We now describe our algorithm for calculating the new distance measure. It is based on the **Surface Distance (SD)**, therefore, we refer to it as **Extended Surface Distance (ESD)**. We first present assumptions and the used notations. We then explain the concepts used in our algorithm. Then, the algorithm is explained step-by-step.

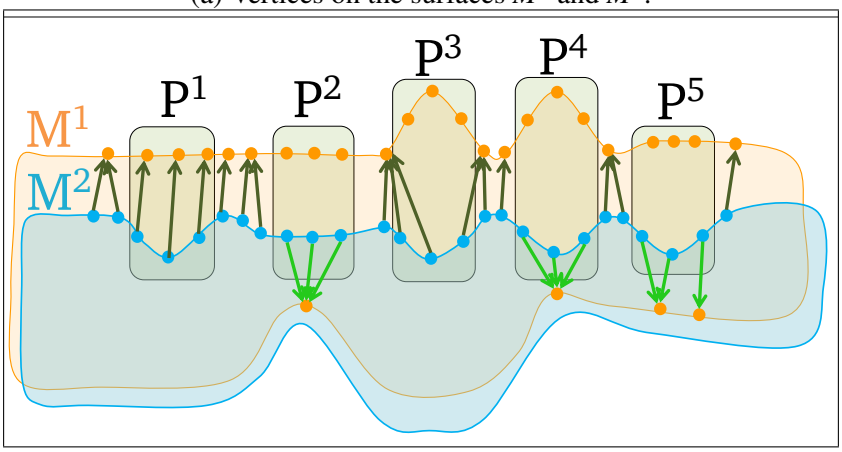
Assumptions: We assume two triangle meshes. We pose no constraints on mesh size and the number of mesh faces. The two meshes can differ in these properties. We assume that point correspondences between meshes are unknown. We also assume mesh closeness and 2-manifoldness as well as spatial alignment. These properties can also be established by preprocessing, e.g., 3D-registration [LH07].

Notations: We assume a comparison of two meshes M^1 and M^2. For clarifying that several calculations are done for both comparison directions $M^1 \rightarrow M^2$ and $M^2 \rightarrow M^1$ we also use the notation M^a and M^b with $\forall a,b \in \{1,2\}$ and $a \neq b$.

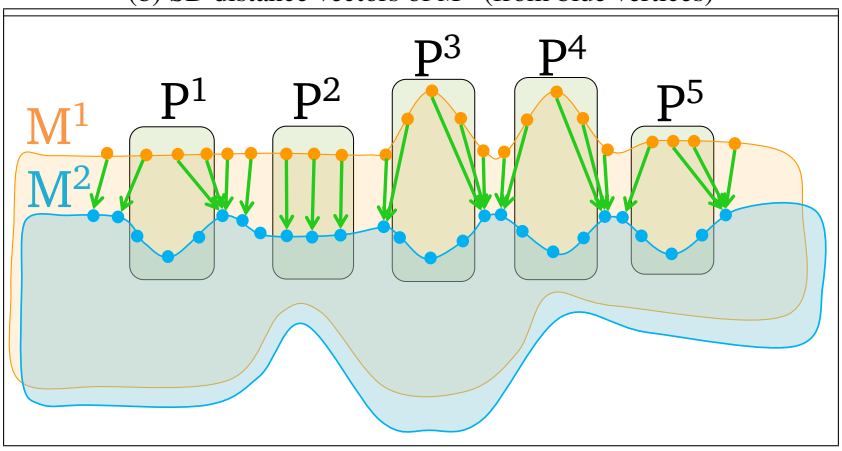
The vertices of M^a can lie inside or outside M^b. $v_i^a \in M^a$ is an inner vertex if the vertex lies inside the surface of M^b. Otherwise, v_i^a is outside. Every vertex v_k^a connected to v_i^a via an edge is in the set of neighbors $nb(v_i^a)$. We define a *region* R^a as a set of vertices $R^a = \bigcup v_i^a \in M^a$. A *region pair* P_i is a tuple (R_i^1, R_i^2) with $R_i^1 \subset M^1$ and $R_i^2 \subset M^2$



(a) Vertices on the surfaces M^1 and M^2.



(b) SD distance vectors of M^2 (from blue vertices)



(c) SD distance vectors of M^1 (from orange vertices)

Figure 3.3.1.: Example meshes and SD distance vectors.

A vertex v_i^a has exactly one *distance vector* $\vec{d}(v_i^a)$ targeting a vertex $v_j^b \in M^b$. Several distance vectors can end in the same vertex v_j^b. The vertex's distance value is the Euclidean length of its distance vector.

The faces of a Mesh M are denoted as f. The surface area of a face f is $Area(f)$. The number of vertices in a Mesh M is $N_v(M)$. The number of faces is $N_f(M)$.

A *neighborhood* of a region is $NB(R^a) = \bigcup v_k^a : v_k^a \notin R^a, \exists v_i^a : v_i^a \in R^a, v_k^a \in nb(v_i^a)$. The *centroid* of a region is $Centroid(R^a) = \frac{\sum v_i^a \in R^a}{N(R^a)}$.

The target vertex v^b of a *distance vector* $\vec{d}(v^a)$ is denoted as $Target(\vec{d}(v^a)) = v^b$. The set of all vertices with a distance vector targeting v^a is denoted as $To(v^a) := \{v^b : Target(\vec{d}(v^b)) = v^a\}$. Therefore $To(v_i^a) = \emptyset$ means that v^a is uncovered.

A vertex $v^a \in M^a$ lying inside M^b is an inner node $Inner(v^a) = true$ else v^a is an outer node $Inner(v^a) = false$. The surface Normal of a vertex v^a is denoted as $\vec{n}(v^a)$

When casting a ray from a point $v^a \in M^a$ to the direction $\vec{d}$, we denote the nearest point $v^b \in M^b$ as $v^b = Ray(v^a, \vec{d})^{M^b}$

A surface-point-set of a mesh M^a is denoted as M^{Ea}. Note that in stage 1 the distance vectors are always calculated from a normal mesh to a surface-point-set and vice versa. Therefore we have to be aware of a relation between vertices of a normal mesh and vertices of a surface-point-set. We denote the relation as follows: $Re(v_i^a) : v_i^a \in M^a$ is the set of all vertices $v_k^a \in M^{Ea}$ which has v_i^a as their nearest vertex in M^a in respect to the *geodesic distance*. The relation is bidirectional, therefore with $v_k^a \in M^{Ea}$ and $v_i^a \in M^a$ we define $Re(v_k^a) = v_i^a$. Additionally we define the same operator for Regions, which means that $Re(R^a) = \bigcup Re(v^a)$ with $R^a = \bigcup v^a \in M^a$.

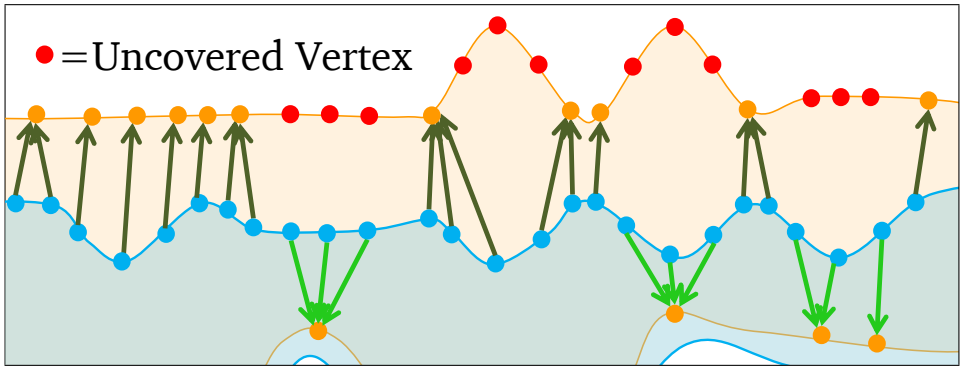
Concepts: The input for our algorithm are two meshes M^1 and M^2.

In our approach, we identify and correct cases in which the Surface Distance (SD) measure is *erroneous* (see Sec. 3.3.1). The identification uses the results of an initial SD calculation. It uses the concepts of the *surface-vertex-sets*, the *uncovered vertices*, the *erroneous region*, and *region pairs*. We explain them before we detail on the algorithm.

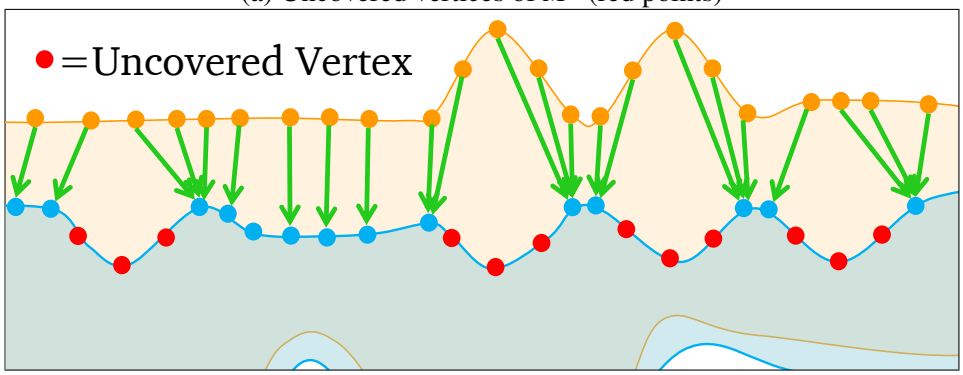
The SD distance vectors of a vertex $v_i^a \in M^a$ are calculated as a vector pointing from v_i^a to the nearest vertex of M^b. A more dense vertex distribution on the surface of the meshes lead to more accurate results. Therefore, we introduce the surface-vertex-sets.

A *surface-vertex-set* M^{Ea} is a set of vertices lying on the surface of M^a. M^{Ea} includes original vertices of M^a and additional (subdivision) vertices. The exact calculation is described in the following algorithm for ESD calculation. These dense vertex sets are needed for identification of uncovered vertices (see Figure 3.3.3).

Uncovered vertices in mesh M^a are vertices with no distance vectors pointing to them from mesh M^b. Figure 3.3.2 shows the uncovered vertices of our example as red dots. The uncovered vertices can only be correctly determined if enough distance vectors cover the surface of one mesh. We show this problem in Figure 3.3.3. Therefore, we always calculate the distance vectors between a surface-vertex-set M^{Ea} and an original mesh M^a. The uncovered vertices result from the SD asymmetry and thus indicate SD errors.

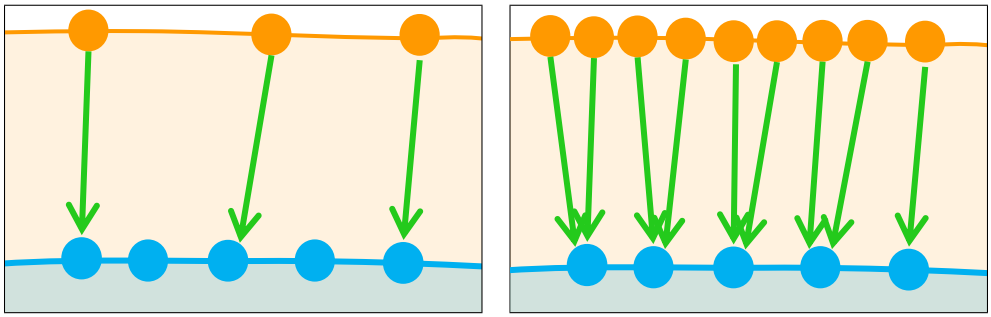


(a) Uncovered vertices of M^1 (red points)



(b) Uncovered vertices of M^2 (red points)

Figure 3.3.2.: Uncovered vertices: not targeted by any distance vector.



(a) $M^1 \rightarrow M^2$ (b) $M^{E1} \rightarrow M^2$

Figure 3.3.3.: The need for surface sets M^E for vertex coverage. $M^{E1} \rightarrow M^2$: enough distance vectors cover the surface.

Erroneous region is a region $R_i^a \subset M^a$ composed of erroneous vertices, i.e., an error case of the SD.

Region pair $P_i = (R_i^1, R_i^2)$ is composed of two regions $R_i^1 \subset M^1$ and $R_i^2 \subset M^2$, which are related. (see Figure 3.3.1, P^1–P^5). We need to identify all region pairs P_i for correcting the erroneous distance vectors of the SD.

Algorithm for Distance Calculation: We describe our algorithm for the computation of the extended distance measure. It consists of 4 stages.

S1 : Calculation of the **surface-vertex-sets** M^{E1} and M^{E2}.

S2 : Construction of **region pairs** and limitation of distance vectors within a region pair.

S3 : Classification of **erroneous** vertices.

S4 : Final correction of distance vectors.

Algorithm 8 Stage 1 : Calculation of Surface-Vertex-Sets and distance initialization

$d_1 \leftarrow \left(\sum_{f \in M^1} Area(f)\right) / N_f(M^1)$
$d_2 \leftarrow \left(\sum_{f \in M^2} Area(f)\right) / N_f(M^2)$
$threshold \leftarrow min(d1, d2)/4$
for all M^a **do**
    for all $f \in M^a$ **do**
        $F \leftarrow \{f\}$
        while $\exists f_i \in F : Area(f_i) > threshold$ **do**
            subdivide f_i
            add all $f_k \in subdivide(f_i)$ to F
            delete f_i from F
        end while
        for all $f_i \in F$ **do**
            add all $v^a \in f$ to M^{Ea}
        end for
    end for
end for
for all $(M^a, M^b) \in \{(M^1, M^{E2}), (M^2, M^{E1})(M^{E1}, M^2), (M^{E2}, M^1)\}$ **do**
    for all $v^a \in M^a$ **do**
        $Target(\vec{d}(v^a)) \leftarrow nearest\ v^b \in M^b$
    end for
end for

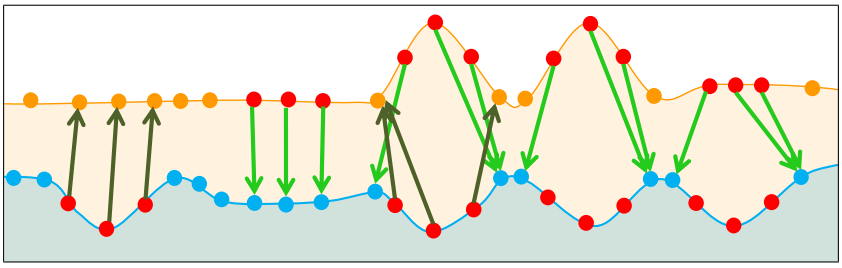
Stage 1: Calculation of Surface-Vertex-Sets: We calculate surface-vertex-sets M^{E1},M^{E2}, and the initial SD. This is shown in Algorithm 8

The surface-vertex-sets are constructed by iteratively subdividing mesh triangles. A triangle is divided into 4 triangles. We use the same scheme as Aspert et al. [ASCE02], because it leads to well

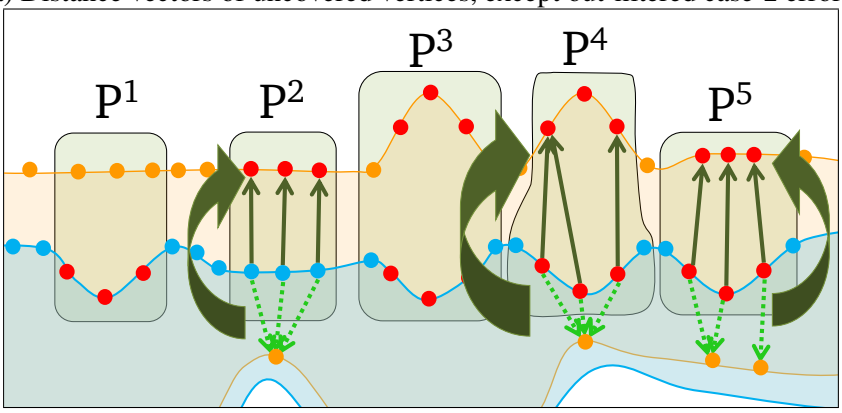
distributed vertices. The surface-vertex-set M^{Ea} consists of all vertices of M^a and all additionally obtained vertices from the subdivision steps.

The number of subdivision iterations depends on the area of the triangle. Each triangle is subdivided until its area is smaller than a defined threshold. We propose a threshold heuristic: We set the threshold to one quarter of the smaller average triangle area of the two meshes. This threshold assures at least one iteration for most triangles.

Afterwards, we compute the SD from a mesh to a surface-vertex-set & vice versa: $M^a \rightarrow M^{Eb}$, $M^{Ea} \rightarrow M^b$.



(a) Distance vectors of uncovered vertices, except out-filtered case-2 errors.



(b) Resulting regions pairs and reoriented distance vectors.

Figure 3.3.4.: Stage 2 identifies region pairs & corrects case-2 errors.

Stage 2: Construction of Region Pairs: This stage constructs region pairs $P_i = (R_i^1, R_i^2)$ using identification of uncovered vertices and regions. As case-2 errors cause that vertices are covered by an erroneous distance vector, we need to filter them out first. This step is presented in Algorithm 9.

The distance vectors not fulfilling the following criteria are filtered out. Note that these criteria are conservative, which means that the erroneous case-2 distance vectors are filtered out correctly but a few correct distance vectors could also be included. Actually, this does not cause problems since the correct distance vectors within differently formed regions are recognized as correct in the next stage. The criteria are:

1. A distance vector has to point from an inner vertex to an outer vertex or vice versa.
2. The surface normals of the start and end vertex of a distance vector should be similarly oriented (the angle between them is less than 90 degree).
3. A distance vector should not intersect any surface, since the distance vectors of the SD should mark the shortest connection between two vertices.

Then the uncovered vertices are identified. We construct regions R_i^a on each mesh separately: neighboring uncovered vertices form a region. We differentiate between inner and outer vertices, leading to inner and outer regions. A region pair always consists of an inner region on one mesh and an outer region on the other mesh.

We then construct region pairs P_i. For each region $R_i^a \subset M^a$ we need to find its counterpart $R_i^b \subset M^b$, $a \neq b$. If an uncovered vertex $v_i^a \in R_i^a$ has a (non case-2 erroneous) distance vector, the end vertex v_j^b is a part of the searched region R_i^b (see Figure 3.3.4(a)). If v_j^b has a neighbor v_k^b which belongs to a erroneous region R_k^b, then R_k^b and the vertex v_j^b is merged to the searched region R_i^b. This is done iteratively until no neighboring error vertices exist. The resulting region pairs are highlighted in Figure 3.3.4(b).

We now need to correct the filtered out case-2 erroneous vertices/regions. The distance vectors of vertices $v_i^a \in R_i^a \in P_i$ are reoriented, so that they target the nearest vertex in the counterpart region $v_i^b \in R_i^b \in P_i$ (see Figure 3.3.4 (b)). This solves case-2 errors overlaying case-1 or case-3 errors.

Stage 3: Classification of erroneous vertices: We now classify the errors of every region pair $P_i = (R_i^1, R_i^2)$ for their correction in Stage 4. As case-2 errors were already eliminated in stage 2, we need to distinguish only case-1 and case-3 errors. Case-3 errors are identified as residual erroneous vertices after case-1 identification.

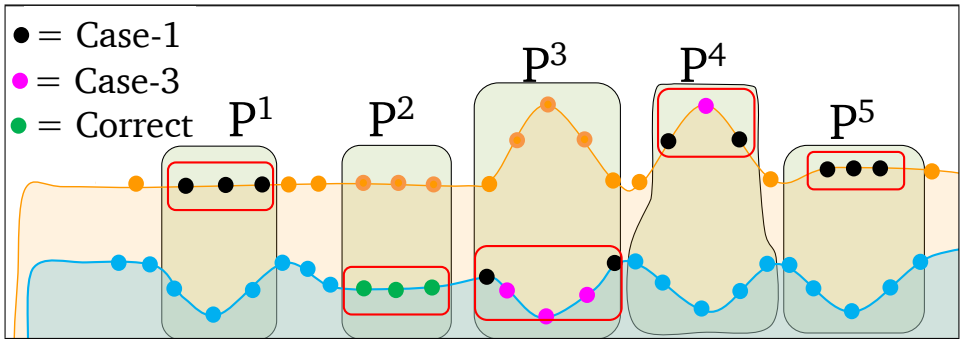


Figure 3.3.5.: Stage 3: The vertices in the erroneous regions (red boxes) of every region pair are classified.

We first identify case-1 errors (i.e., erroneous in one comparison direction). These can be easily identified, because correct distance vectors are present in one of the two regions $R_i^1 \in P_i$ or $R_i^2 \in P_i$. We have to identify which region R_i^a ($a \in \{1,2\}$) of the region pair P_i is the erroneous one. The other

Algorithm 9 Stage 2: Construction of Region Pairs

for all M^a **do**
  for all $v^a \in M^a$ **do**
    if $Inner(v^a) = Inner(Target(\vec{d}(v^a))) \vee \vec{d}(v^a)$ intersects M^1 or M^2
      $\vee\ \vec{n}(v^a) \cdot \vec{n}(Target(\vec{d}(v^a))) < 0$ **then**
      add v^a to $\{case\text{-}2\ error\}$
    end if
  end for
end for
for all M^a **do**
  if $\exists v_i^a \in M^a : To(v_i^a) \setminus \{case\text{-}2\ error\} = \emptyset, \nexists R_i^a : v_i^a \in R_i^a$ **then**
    new $R_i^a \leftarrow \{v_i^a\}$
    while $\exists v_k^a \in M^a : nb(v_k^a) \setminus R_i^a \neq \emptyset \wedge\ Inner(v_i^a) = Inner(v_k^a) \wedge\ v_k^a \notin R_i^a$ **do**
      add v_k^a to R_i^a
    end while
  end if
end for
for all R_i^1 and R_i^2 **do**
  Initiate $P_i = (R_i^1, \emptyset)$ or $P_i = (\emptyset, R_i^2)$
end for
while $\exists v_i^a \in Re(R_i^a) \in P_i : \exists v_i^b \notin P_i : Target(\vec{d}(v_i^a)) = v_i^b \wedge\ To(Re(v_i^a)) = \emptyset$
    $\wedge\ Inner(v_i^a) \neq Inner(v_i^b)$ **do**
  add v_i^b to P_i
  if $\exists v_k^b \in nb(v_i^b) : v_k^b \in R_k^b \in P_k \neq P_i$ **then**
    merge P_i and P_k
  end if
end while
for all P_i **do**
  for all $R_i^a \in P_i$ **do**
    for all $v^a \in R_i^a$ **do**
      $Target(\vec{d}(v^a)) \leftarrow nearest\ v^b \in R_i^b \in P_i$
    end for
  end for
end for

region R_i^b has correct distance vectors. These vectors can then be used for correcting the case-1 errors in the erroneous region (see Stage 4).

The erroneous region is identified by counting the number of *uncovered* vertices. Erroneous regions have less uncovered vertices, since correct distance vectors always cover more vertices of the surface than erroneous distance vectors (see Figure 3.3.5). We count the erroneous vertices of the surface-vertex-sets M^{Ea} and M^{Eb}, because they have similar number of vertices across region pairs. Original meshes M^a and M^b can differ much in the number of vertices. As shown in Figure 3.3.5 in P^3, region pairs of case-3 error can have the same count of erroneous vertices, in which case we can simply choose one of the two regions (we take the region in M^2) as erroneous, as both are erroneous and will be identified as case-3 in the following. Step 3 is shown in Algorithm 10.

The vertices of the erroneous region $R_i^a \in P_i$ are classified as follows (see Figure 3.3.5):

1. The vertex is *covered* by at least one distance vector from an uncovered vertex $\rightarrow$ case-1 error.
2. The vertex is only *covered* by distance vectors of covered vertices. This vertex is correct, so it is not classified.
3. The vertex is not *covered* at all $\rightarrow$ case-3 error.

Stage 4: Final correction of distance vectors: We now correct the distance vectors. We first correct the case-1 errors. This correction is used for the later correction of case-3 errors. Note that case-2 were corrected in stage 2. This step is shown in Algorithm 11.

The case-1 erroneous vertices are in the erroneous region $R_i^a \in P_i$. The other region $R_i^b \in P_i$ has correct distance vectors. We will now use the correct distance vectors of R_i^b for the reorientation of the distance vectors of R_i^a. The classification in stage 2 required that every vertex $v_i^a \in R_i^a$ of case-1 error has one or more *uncovered* $v_i^b \in R_i^b$, whose distance vector targets v_i^a. One of these correct distance vectors targeting v_i^a should be mirrored by v_i^a. We reorient the distance vector of v_i^a so that it points to the *furthest* vertex of all the vertices v_i^b, as defined above. By choosing the furthest vertex we rather slightly overestimate the distance – leading to a locally maximal distance. Figure 3.3.6 (a) shows the result.

We can now correct the possible residual case-3 errors in P_i. In contrast to case-1 errors, the case-3 errors are located on both regions of the region pair P_i. We denote the case-3 error regions as $R_k^a \subset R_i^a \in P_i$ and $R_k^b \subset R_i^b \in P_i$. We already identified the case-3 erroneous vertices of R_k^a within R_i^a as these are the vertices classified as case-3 in stage 3, but we now also need to identify the case-3 erroneous vertices of R_k^b within the counterpart R_i^b.

We use the case-1 correction for the identification of R_k^b, as these are closely related (see Figure 3.3.6(a), P^3 and P^4). The distance vectors of $v_i^a \in R_i^a$ of the former case-1 error vertices now target the case-3 erroneous region R_k^b within R_i^b.

The specific case-3 erroneous region R_k^a is defined as all neighboring vertices classified as case-3 in R_i^a. We identify its counterpart case-3 error region R_k^b: Every vertex of R_i^b which is pointing to one of the former case-1 error vertices surrounding R_k^a is now labeled as a *candidate vertex*, i.e., a candidate for the searched region R_k^b.

Algorithm 10 Stage 3: Classification of erroneous points

for all $P_i = (R_i^1, R_i^2)$ **do**
  for all $a \in \{1,2\}$ **do**
    $c^a \leftarrow 0$
    for all $v^a \in Re(R_i^a)$ **do**
      if $To(Re(v^a)) = \emptyset$ **then**
        $c^a \leftarrow c^a + 1$
      end if
    end for
  end for
  if $c^1 < c^2$ **then**
    $a \leftarrow 1, b \leftarrow 2$
  else
    $a \leftarrow 2, b \leftarrow 1$
  end if
  for all $v_i^a \in R_i^a$ **do**
    if $To(v_i^a) = \emptyset$ **then**
      add v_i^a to $\{case\text{-}3\ error\}$
    else if $\nexists v_k^a \in To(v_i^a) : To(Re(v_k^a)) = \emptyset$ **then**
      v_i^a is correct
    else
      add v_i^a to $\{case\text{-}1\ error\}$
    end if
  end for
end for

We now have to identify which *candidate vertices* of R_i^b are related to R_k^a and therefore are a part of R_k^b. For this identification we use a ray casting approach as case-3 regions can contain arbitrarily dissimilar deformations.

We first identify the direction of the rays: We compute the centroid of R_k^a and the centroid of all *candidate vertices* which are target vertices of a distance vector of the former case-1 error vertices neighbored to R_k^a. Our ray direction vector points from the first to the second centroid. We cast a ray from each vertex of R_k^a in this direction. The nearest *candidate vertex* of each ray belongs to R_k^b.

As a last step in case-3 region identification, we need to adapt some of former case-1 error vertices surrounding R_k^a, because these are pointing to R_k^b and possibly also belong to case-3: If a surrounding vertex of R_k^a is only *covered* by vertices of R_k^b, then it is only covered by case-3 erroneous vertices and therefore belongs to R_k^a. Otherwise, if the vertex is also covered by a vertex outside of R_k^b, then the distance vector has to be adapted, so that it targets the furthest *covering* vertex, outside of R_k^b.

The final reorientation of the distance vectors uses additional rays cast between both regions R_k^a and R_k^b. The rays are aligned as before: Every vertex of R_k^a and R_k^b casts a ray. The nearest vertex to the ray on the other region is the end vertex of vertex' distance vector (see Figure 3.3.6(b)).

3.3.3. Evaluation and Discussion

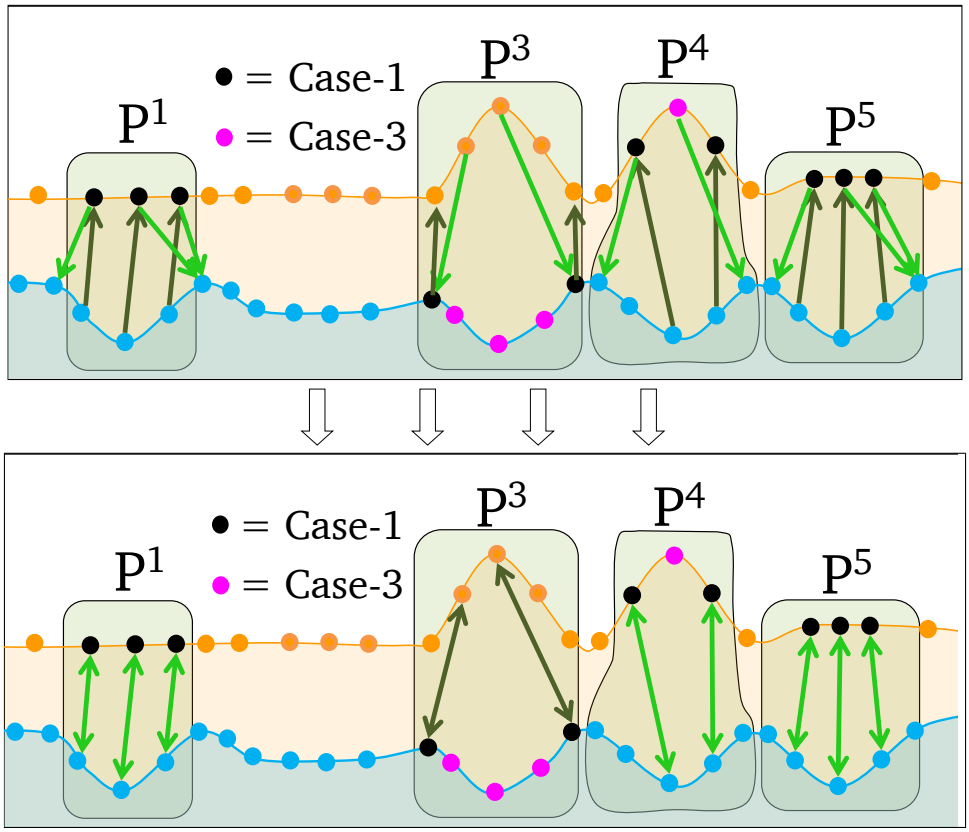
We evaluate our approach on the application of 3D medical image segmentation [KBW11, BKS14]. In this application polygon meshes are constructed from medical images. To evaluate this application a ground truth mesh (reference segmentation (RS)) is compared with an automatically segmented mesh (automatic segmentation (AS)). The focus of our evaluation is on the assessment of the quality measurement using our distance measure and using the SD.

We first describe the datasets and then show the qualitative and quantitative evaluation. Finally, we discuss the advantages and limitations of our approach.

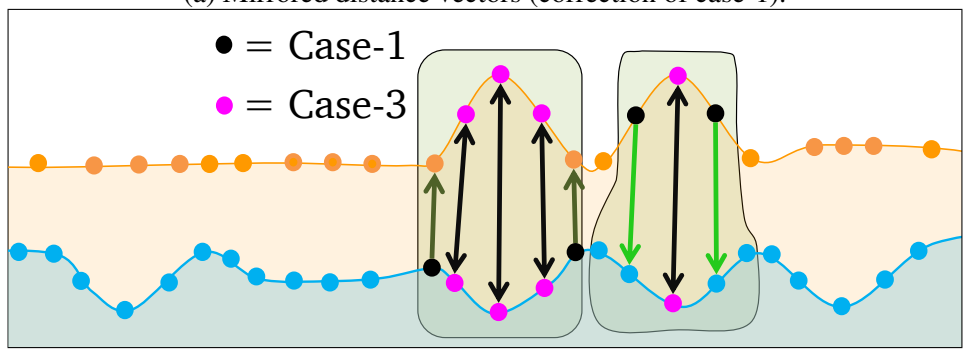
Data Description: We evaluated four datasets consisting of 20 livers, 42 nervus facialis, 42 semicircular canals and 20 cochleas. The real-world medical image data was extracted from CT scans. The liver segmentation algorithm is [KBW11] and the segmentation of the other datasets was done using [BKS14].

We evaluate various datasets in order to show the versatility of our approach. The new distance measure is independent of the mesh size and number of vertices and faces (see Table 3.3.1). It is able to evaluate the organs of different sizes. The cochleas are very small, while the livers are rather large. They also differ in other proprieties, as the nervus facialis are of genus 0 and the Semicircular canals are of genus 3. The livers have different genera within the dataset (some livers include tubular structures inside the organ)

Qualitative Evaluation: We assess how well the measure identifies regions of bad/good segmentation quality. We visually inspected the local distances on both automatic and reference segmentations for every comparison.



(a) Mirrored distance vectors (correction of case-1).

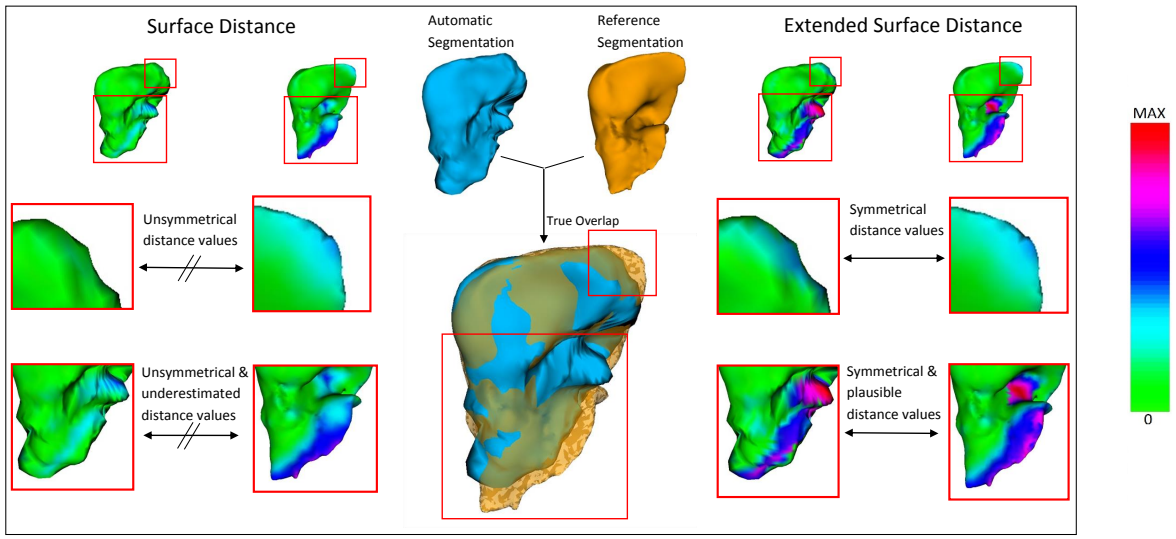


(b) Black arrows: corrected by rays.
Green Arrows: corrected by final case-1 adaption.

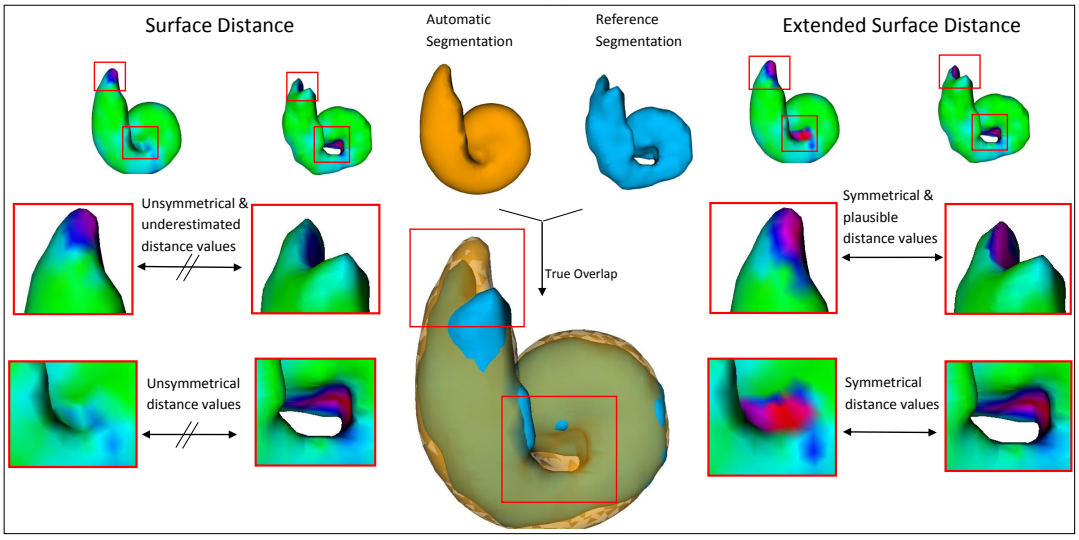
Figure 3.3.6.: In Stage 4 distance vectors of case-1 and case-3 erroneous vertices are reoriented. a) case-1 distance vectors are reoriented by mirroring correct distance vectors. b) case-3 distance vectors are reoriented with the help of rays cast between the erroneous regions.

Algorithm 11 Stage 4: Final correction of distance vectors

for all $P_i = (R_i^1, R_i^2)$ **do**
  a and b as defined in Stage 3
  for all $v^a \in R_i^a \in P_i$ **do**
    if $v^a \in \{case\text{-}1\ error\}$ **then**
      $Target(\vec{d}(v^a)) \leftarrow furthest\ v^b \in To(v^a)$
    end if
  end for
  while $\exists v_i^a \in R_i^a : v_i^a \in \{case\text{-}3\ error\} \setminus \{Finished\}$ **do**
    $R_k^a \leftarrow \{v_i^a\}, R_k^b \leftarrow \emptyset$
    while $\exists v_k^a \in R_i^a : \exists v_m^a \in R_k^a : v_k^a \in nb(v_m^a) \wedge v_k^a \in \{case\text{-}3\ error\} \wedge v_k^a \notin R_k^a$ **do**
      add v_k^a to R_k^a
    end while
    $\{Surroundings\} \leftarrow NB(R_k^a)$
    while $\exists v_k^a \in R_i^a : \exists v_m^a \in \{Surroundings\} : v_k^a \in nb(v_m^a) \wedge v_k^a \in \{case\text{-}1\ error\}$ **do**
      add v_k^a to $\{Surroundings\}$
    end while
    $\{Candidates\} \leftarrow \emptyset$
    while $\exists v^b \in Re(R_i^b) : Re(v^b) \notin \{Candidates\} \wedge Target(\vec{d}(v^b)) \in \{Surroundings\}$ **do**
      add $Re(v^b)$ to $\{Candidates\}$
    end while
    $\vec{d} \leftarrow$from $Centroid(R_k^a)$ to $Centroid(\{Candidates\})$
    for all $v_k^a \in Re(R_k^a)$ **do**
      add $v^b = Ray(v_k^a, \vec{d})^{\{Candidates\}}$ to R_k^b
    end for
    for all $v_k^a \in \{Surroundings\}$ **do**
      if $To(v_k^a) \setminus R_k^a = \emptyset$ **then**
        add v_k^a to R_k^a
      else
        $Target(\vec{d}(v_k^a)) \leftarrow furthest\ v^b \in To(v_k^a) \setminus R_k^a$
      end if
    end for
    for all $v_k^a \in R_k^a$ **do**
      $Target(\vec{d}(v_k^a)) \leftarrow Ray(v_k^a, \vec{d})^{R_k^b}$
    end for
    for all $v_k^b \in R_k^b$ **do**
      $Target(\vec{d}(v_k^b)) \leftarrow Ray(v_k^b, -\vec{d})^{R_k^a}$
    end for
    for all $v_k^a \in R_k^a$ **do**
      add v_k^a to $\{Finished\}$
    end for
  end while
end for

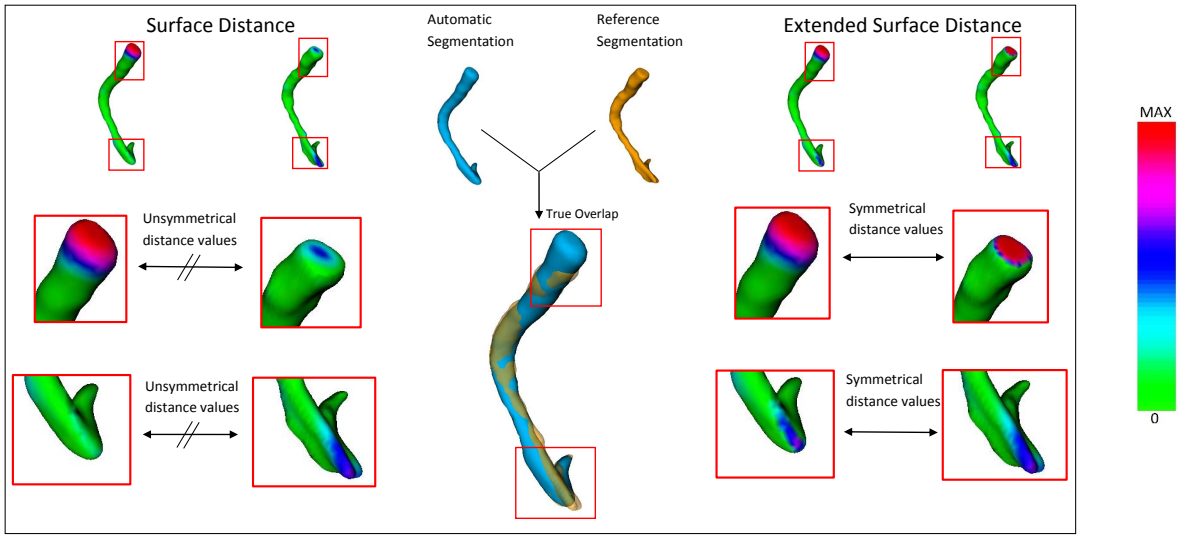


(a) Liver

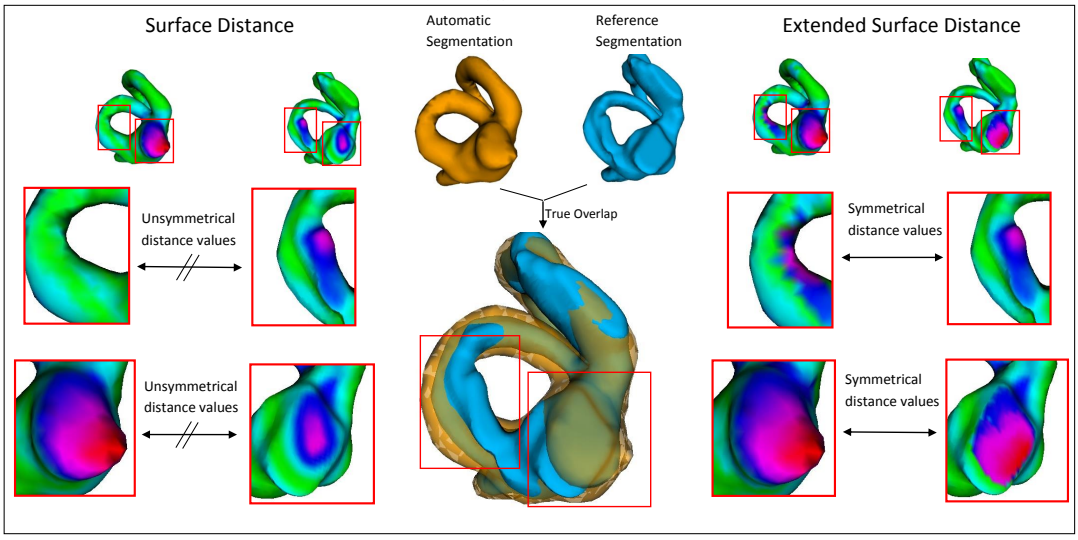


(b) Cochlea

Figure 3.3.7.: The widely used Surface Distance (SD) compared to our Extended Surface Distance (ESD). The measures are used for the local comparison of a (ground truth) reference segmentation mesh and an automatic segmentation mesh of a 3D medical image. The SD is asymmetric and underestimates the distances. The new measure shows better results.



(a) Nervus Facialis



(b) Semicircular Canals

Figure 3.3.8.: Qualitative comparison of SD (left) and ESD (right) for segmentation evaluation (middle). The ESD shows more accurate information on both meshes even in this difficult examples.

Dataset	Count	$\#F$: RS	$\#F$: AS
Liver	20	8000 - 20000	5120
Cochlea	20	5832 - 12626	1996
Semicircular canals	42	8162 - 23746	4008
Nervus facialis	42	4132 - 15386	2996

Table 3.3.1.: Dataset properties. $\#F$ = Number of faces in the mesh, RS= Reference segmentation, AS= Automatic segmentation.

One important benefit of the new measure is its symmetry. Symmetry enables to spot problems on both automatic (AS) and reference (RS) segmentations. The SD was not symmetric and thus it was not possible to detect all segmentation errors by analyzing the results on one mesh. In contrast, the new distance measure recognizes regions of bad segmentation that were either only identified on one of the two meshes or could not be identified with the SD at all (see Figure 3.3.7 and 3.3.8).
As an example, we look at the Semicircular canals and the Nervus facialis (see Figure 3.3.8). Both have two large erroneous regions (see red box). Here, one region is erroneously measured in the AS and the other region is erroneously measured in the RS. Thus for several erroneous regions the correct distance values can be scattered across both meshes. This means that there is no single mesh, which identifies correct distances on all regions. The ESD is able to detect correct distances for all regions on both meshes.

The Liver and the Cochlea (see Figure 3.3.7) examples show errors that are even more severe than the symmetry problem. Both examples have one large region (see red box), in which the SD underestimates the difference on both meshes. Therefore, the correct information is not available anywhere. The new distance measure indicates better and more plausible results, even on these very difficult and badly segmented examples.

Quantitative Evaluation: We quantitatively evaluate the new distance measure. This evaluation is difficult as there are no ground truth distances available. This is because the difference of meshes is not well defined, so that a ground truth could only be constructed with a similarity measure itself. Therefore, we propose a quality assessment method which uses a symmetric global measure for the comparison of the local measures.

We transform the two input meshes M^1 and M^2 to the deformed meshes M^{D1} and M^{D2} by moving each vertex of the mesh to the target vertex of its distance vector (see Figure 3.3.9). The target vertices of the distance vectors of M^1 are on the surface of M^2. Therefore, if the distance vectors are good, the deformed mesh M^{D1} should be very similar to the original mesh M^2. We measure this similarity using symmetric Hausdorff Distance. The lower this global distance the better the distance measure (SD or ESD) performs. We note that this method does not replace ground truth distances. Nevertheless, it provides a quality indicator. Figure 3.3.10 shows the results for all datasets.

- *Accuracy:* The mean value close to zero indicate high accuracy. Our measure has lower mean values for each direction in all datasets. ESD is more accurate than SD.

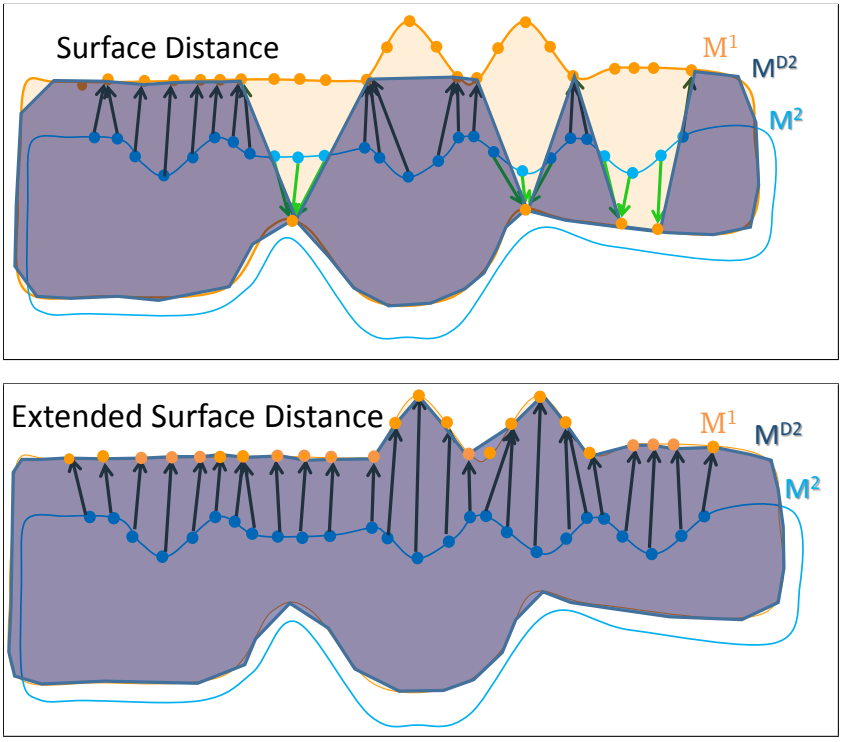


Figure 3.3.9.: Mesh transformation for quantitative evaluation.

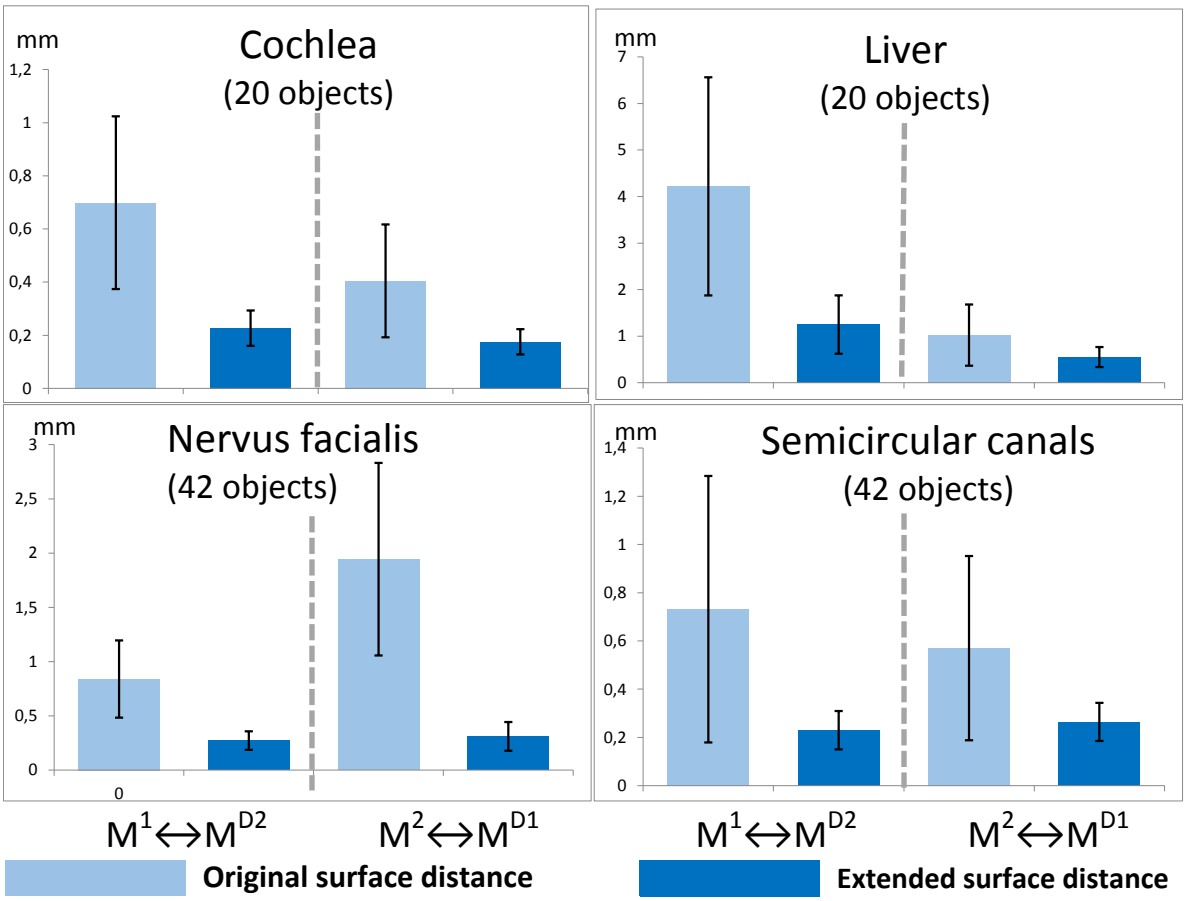


Figure 3.3.10.: Quality of the SD and the ESD for four real datasets. Low mean – better distance quality. Low standard deviation – robustness. ESD shows better results.

- *Reliability and robustness:* Low standard deviation indicates robustness (i.e., low measure quality variation). ESD has lower standard deviations then SD, thus ESD has lower variability and higher reliability.
- *Asymmetry:* The asymmetry of the distance measures can be assessed by looking at the difference between the first comparison $M^1 \leftrightarrow M^{D2}$ and the second comparison $M^2 \leftrightarrow M^{D1}$. Symmetric measures would have no differences. The SD is much more asymmetric than ours as it has larger differences.

Discussion: We presented the ESD for comparing two meshes resulting from medical image segmentations. Our new distance measure overcomes the shortcomings of the widely used SD. It provides more accurate and more symmetric distance values. The new measure does not need any user interaction or parameter setting. It can deal with various anatomical structures. It, however, has several limitations. We now discuss the main aspects of our approach: the assumptions and specifics of our algorithm. We discuss the relationship of our approach to finding correspondences. Finally, we touch upon the evaluation and runtime of our approach.

Our algorithm has several *assumptions* on the input data. The input meshes meshes should be closed 2-manifolds and should be spatially aligned. These properties can already be ensured by the results of 3D medical image segmentation algorithms. Alternatively, these properties can also be established by pre-processing. This, however, can introduce approximations errors. Therefore, mesh comparison measures (like the ESD) can be dependent on the quality and accuracy of the surface representation as mesh.

Although our measure provides generally better results than the SD, it can still *slightly overestimate* the distance values in the differently formed regions (due to the concept of local maxima). This characteristic, however, ensures a clear indication, where large differences exist. An underestimation could lead to false implications of segmentation quality.

Our work primarily concentrated on the improvement of comparison quality. Nevertheless, we also assessed the *runtime* on the example datasets (ca. 2k–23k faces). We used a computer with an Intel(R) Core(TM) i7-3930K Processor. Without parallelization, the calculation took 7–16 sec.

The problems of local distance measurement and the identification of *point-to-point correspondences* are related. However, the correspondence is generally very difficult to solve for arbitrarily formed meshes [VKZHCO11]. The algorithms for calculating correspondences often focus on specific cases. Therefore, they are also more restrictive concerning the preconditions and are less flexible and robust concerning the input meshes. Correspondence algorithms are mostly restricted to input meshes which fulfill assumptions about the form and semantic coherence. We aimed for a more general solution. In our case, the input meshes can have arbitrary formed surfaces and topologies, when our basic assumptions are met. Moreover, our algorithm also does not need any parameter setting for different sizes and types of meshes. If a well suited correspondence finding algorithm is available for a particular case at hand, this might be used instead. This could, however, reduce comparability of result quality across organs.

The *evaluation* of our approach on real data indicates that the new measure provides more expressive and more reliable results than the SD. It also reduces asymmetry problems. Thereby it allows for visual inspection of the results solely on one of the two meshes. This reduces the evaluation burden for larger datasets.

3.4. 3D Object Retrieval with Hierarchical Clustering

In many applications the accuracy of a 3D object retrieval is purely dependent on the accuracy of a descriptor. Improvements on top of the descriptor performance are often missed. As described in Section 3.2.3 there are several other possibilities to improve 3DOR applications. Nevertheless, these are dependent on adding additional knowledge [HLR05]. Only a few algorithms are able to improve 3DOR without any additional knowledge like user interaction or classification information.

Tatsuma and Aono [TA09] propose an approach which computes a clustering of the database without using any additional knowledge. They also show that the results of the retrieval can be significantly improved with the clustering information. However, besides other parameters, one needs to define how many cluster k are present in the database. k has a high impact on the accuracy of the results.

Indeed, a class and a cluster are similar concepts. Therefore, one could generate classes of objects by clustering the database.

The number of classes/clusters of a database is not easy to define. Even if we ask a human into how many classes the person would separate a set of objects, different persons would come up with different numbers of classes (e.g. a sheep object can belong to a class sheep or a class quadruped). This is the main drawback of many clustering algorithms (e.g. k-means) used for 3D object retrieval. The parameter setting is highly dependent on the database and needs additional knowledge itself.

Hence, we propose a completely parameter-free algorithm to improve 3DOR applications. In our approach we calculate a hierarchical clustering instead of a predefined set of clusters. Each object then does not only belong to a single cluster, but to one cluster on each hierarchy level. We propose an algorithm which uses the cluster hierarchy to directly improve the retrieval results for a single query.

Our approach is guided by the idea to use all known distances between all objects of a database to improve the retrieve results for a single object query. Usually only the distances from the query object itself are used.

We use all distances to compute a hierarchical clustering of the database and use the cluster-information together with the distance information to improve the results for a single query.

To compute the distance between two objects we use the panorama-descriptor [PPTP10]. In our approach the descriptor itself can be treated like a black-box, as we only use the measured distances between objects. A combination of different descriptors is also possible as these has the same input and output like a single descriptor. In fact, the combination of many descriptors is nothing else than a

new descriptor itself. In our approach the descriptor is generally interchangeable and it would also be possible to use a combination of descriptors.

For our approach we assume to have a database with n 3D objects O_i and a descriptor $desc(..)$ for the comparisons of the objects. The result of the computation of the descriptors for an object O_i is $desc(O_i)$. The comparison of two objects $\Delta[O_i, O_j]$ equals the comparison of their descriptors $\Delta[desc(O_i), desc(O_j)]$.

For a database of size n we can compute a distance matrix with all $\Delta[O_i, O_j] : i, j \in [0, n]$:

$$\begin{bmatrix} \Delta[O_0,O_0] & \Delta[O_0,O_1] & \dots & \Delta[O_0,O_n] \\ \Delta[O_1,O_0] & \Delta[O_1,O_1] & \dots & \Delta[O_1,O_n] \\ \vdots & \vdots & \ddots & \vdots \\ \Delta[O_n,O_0] & \Delta[O_n,O_1] & \dots & \Delta[O_n,O_n] \end{bmatrix}$$

Note that usually a descriptor has the proprieties: $\Delta[O_i, O_j] = \Delta[O_j, O_i]$ (symmetry) and $\Delta[O_i, O_i] = 0$ (identity).

For a single 3D object query O_{query} we want to compute a retrieval list. The retrieval list is a list containing all objects of the database, which is sorted by similarity to the O_{query}. The common method to compute the retrieval list is to compute the distances to all database objects $\Delta[O_{query}, O_i]$ and sort by distance. Therefore, the common method only uses one column (or row) $\Delta[O_{query}, O_0]$ to $\Delta[O_{query}, O_n]$ of the distance matrix for the calculation of the retrieval list for a single query.

We use the full distance information to primarily compute a hierarchy of clusters with a hierarchical agglomerative clustering (Section 3.4.1). Then we present our proposed algorithm, which combines the information of the computed distances with the hierarchical clustering to improve the retrieval list of a single query (Section 3.4.2). Note that in this case the query object is part of the clustering itself.

Another relevant possible case is that a new object which is not part of the clustering is used as query object. Also it is possible that new objects are inserted frequently into the database. For this case we do not want to recompute the clustering for the whole database every time. We propose an efficient solution to make use of our algorithm for this case (Section 3.4.3). Note that the agglomeration clustering strategy itself is well-known. Our main contribution is the combination of the clustering with the single distances to directly improve the retrieval results.

3.4.1. Hierarchical Clustering of a 3D Object Database

A hierarchical clustering has the purpose to produce a hierarchy of clusters. In the top level of the hierarchy, there is only a single cluster including all objects. In the first level of the hierarchy, each database object is a single cluster. In each hierarchy level two clusters are merged and therefore the number of clusters is diminished by 1. So the first level of the hierarchy has n clusters and the top level of the hierarchy consists of 1 cluster (the top level is $n-2$ if we assume that we start with level 0).

The agglomerative strategy for computing a hierarchical clustering is a 'bottom-up' strategy. We start on the lowest hierarchy level, where each cluster contains only a single object and than compute the upper hierarchy levels by merging the two most similar clusters in each step. Algorithm 12 shows this simple strategy.

Algorithm 12 General algorithm for the hierarchical agglomerative clustering of a 3D object database

1: **for all** $O_i \in$ Database **do**
2: $Initiate$ cluster $C_i = \{O_i\}$
3: **end for**
4: **for** $i = 0$ to $n-2$ **do**
5: $Compute$ all distances $\Delta[C_i, C_j]$
6: $Merge\ C_i\ and\ C_j$ $(i \neq j)$ with smallest distance
7: **end for**

The only part of the agglomerative clustering algorithm which we have to define is the distance function for computing the distance of two clusters $\Delta[C_i, C_j]$.

For our application many distance functions are possible, e.g. average-linkage, single-linkage, minimum-energy, average-group-linkage, complete-linkage and centroid distance. Each different function tend to produce a certain type of clusters: equal sized cluster, unequal sized clusters, small clusters, big cluster or clusters with similar density. For 3D object databases we experienced that the 3D Object clusters have a very high variation of size and density. Assuming a high variation of different clusters is also a good assumption for real (unclassified) databases. All functions strongly enforcing a certain type of cluster fail to produce such a variation of clusters.

For this reason we choose the most balanced distance function for our algorithm: the so called 'average-linkage': The distance of a cluster C_i to a cluster C_j is the average distance of each $O_i \in C_i$ to each $O_j \in C_j$.

3.4.2. Hierarchical Clustering based Retrieval Algorithm

In the cluster-hierarchy, each hierarchy level corresponds to a complete clustering of the database with a different number of clusters. The question arises how we can use several clusterings for the improvement of a single query. Tatsuma and Aono [TA09] use a single clustering of the database. They combine the distance to an object and the distance to the cluster of the object to compute a final distance to the object. With this technique they compensate the uncertainty of the clustering, resulting from the single clustering with a predefined number of clusters.

In a perfect clustering we could expect that each object of the same cluster is more similar to the query object than any object of another cluster. Of course, the hierarchical clustering also has a degree of uncertainty. What we know for sure is that the confidence in the lower hierarchy levels of the cluster-hierarchy is higher than the confidence in the higher hierarchy levels. For instance, in the first level

each cluster has only one object and therefore the confidence for the cluster members to belong to the same class is 1.

For this reason we propose a new algorithm iterating over the cluster-hierarchy starting with the first level (highest confidence) and ending with the top level (lowest confidence). In each level we only consider the objects which are in the same cluster as the query Object O_{query}. Note that in the highest level all objects of the database are within a single cluster and therefore in the highest level all objects of the database are considered.

The result of our method for a single query object O_{query} is a retrieval list. The retrieval list is a list containing all objects of the database sorted descending by similarity. Hence, the first object added to the retrieval list is the most similar and the last object added to the retrieval list is the most different.

Algorithm 13 Retrieval using a precomputed hierarchy of clusters

1: **for** $i = 0$ *to* $n-2$ **do**
2: $C_i =$ cluster of hierarchy level i with $O_{query} \in C_i$
3: *Find* all $O_i \in C_i$ *with* $O_i \notin$ retrieval list
4: *Sort* the list of all found O_i by distance $\Delta[O_{query}, O_i]$
5: *Add* all O_i to the retrieval list starting with the smallest distance $\Delta[O_{query}, O_i]$
6: **end for**

Algorithm 13 outlines our method for the retrieval. We iterate over the hierarchy levels staring with the lowest level. In each hierarchy level all objects O_i, that are in the same cluster as the query object are added to the retrieval list. However, they are ordered in a sorted way. The objects O_i are added to the retrieval list sorted by distance to the query object $\Delta[O_{query}, O_i]$. The object with the smallest distance is added first. We thus primarily sort by cluster/class membership and secondarily by distance to the query object.

We can compute the result lists for all objects of the database efficiently during the clustering. When two clusters are merged we only update the retrieval lists for the objects within these 2 clusters accordingly.

3.4.3. Retrieval for an Unknown Query Object

Our algorithm assumes that the query object is part of the clustering for the computation of the results. However, we need to consider the case that the query object is not part of the database and therefore is unknown in advance. Also, it is possible that new objects are inserted into the database and we do not want to recompute the clustering for every single object.

The first possibility we considered, is to find the N nearest neighbors of the query object and compute a new hierarchical clustering only for these N nearest neighbors and the query object itself. We tested this possibility with several sizes for N for each database and experienced that N can have a high influence on the precision of the results depending on the database. The clustering can be much less

reliable if less objects are used. We also do not want to introduce a parameter to our algorithm, as it should work for the unsupervised case. Hence, we discarded this possibility.

Nevertheless, during the tests we recognized that the clustering of the whole database does not change significantly if a single object is added or removed to the database. Hence, the clustering of the whole database with and without the unknown query object is pretty much equal in the majority of the cases. With this reasoning we discovered a simple algorithm which leads to very good results:

As shown in Algorithm 14, we compute the nearest neighbor of the unknown query object and use the precomputed clustering of the database. Then we just insert the unknown query object into all clusters in which the nearest neighbor is also present. This effectively adds the new object to the clustering. Therefore we can use our retrieval Algorithm (Algorithm 13) for the new object.

The big advantage is also, that this fast strategy can be used if new objects are inserted into the database, without the need for recomputing the clustering. The clustering is just updated by using Algorithm 14 and the adapted clustering is used for further retrieval queries.

Nevertheless, this strategy does not imply that the clustering should never be recomputed. If more and more objects are inserted and we do not recompute the clustering the retrieval algorithm degenerates to a standard retrieval. In the extreme case if the clustering only consists of a single cluster with all objects, then our retrieval Algorithm 13 exactly corresponds to the standard retrieval which only sorts by distance to other objects.

We could not identify any threshold which defines how many new objects can be inserted before the clustering should be recomputed. This completely depends on the specifically inserted objects. If the newly inserted objects are distributed well over all clusters the retrieval performance is barely diminished by adding new objects to the clustering.

Algorithm 14 Adaption for an unknown query object

1: *Find* Nearest Neighbor O_{NN} of O_{query}
2: **for** All clusters C of all hierarchy levels **do**
3: **if** $O_{NN} \in C$ **then**
4: *Insert* O_{query} into C
5: **end if**
6: **end for**

3.4.4. Evaluation and Discussion

We evaluated our approach with 6 databases including different kind of objects with a total of 13271 in 481 classes. Each database contains objects with class labels and each of the 13271 objects are taken as the query object. For a query object each retrieved object is considered to be a correct retrieval if the class label of the retrieved object corresponds to the class label of the query object. The databases differ in the types of objects and in the number of objects per class. We used the following databases:

	PAN	PAN + HC	Improvement
DCG	0.7644	**0.7780**	1.78%
NN	0.7974	**0.8537**	**7.06**%
FT	0.4228	**0.4751**	**12.37**%
ST	0.5462	**0.5738**	**5.05**%
E-M	0.3035	**0.3456**	**13.86**%
DCG-32	0.6462	**0.6726**	4.09%

(a) SHREC2014 (8987 Objects in 171 Classes)

	PAN	PAN + HC	Improvement
DCG	0.8843	**0.9221**	4.28%
NN	**0.9675**	0.9525	-1.55%
FT	0.6716	**0.8303**	**23.63**%
ST	0.7843	**0.8664**	**10.47**%
E-M	0.4825	**0.5492**	**13.80**%
DCG-32	0.7798	**0.8590**	**10.17**%

(b) SHREC2007 (400 Objects in 20 Classes)

	PAN	PAN + HC	Improvement
DCG	0.8499	**0.8824**	3.82%
NN	**0.9069**	0.9028	-0.46%
FT	0.6312	**0.7520**	**19.13**%
ST	0.7684	**0.8352**	**8.70**%
E-M	0.4552	**0.5154**	**13.21**%
DCG-32	0.7488	**0.8068**	**7.73**%

(c) NIST (720 Objects in 40 Classes)

	PAN	PAN + HC	Improvement
DCG	0.7317	**0.7378**	0.82%
NN	0.7814	**0.7832**	0.23%
FT	0.4861	**0.5351**	**10.09**%
ST	0.6094	**0.6249**	2.55%
E-M	0.3928	**0.4193**	**6.74**%
DCG-32	0.6108	**0.6174**	1.08%

(d) NTU (549 Objects in 47 Classes)

	PAN	PAN + HC	Improvement
DCG	0.7035	**0.7066**	0.45%
NN	**0.7466**	0.7289	-2.36%
FT	0.4662	**0.5120**	**9.81**%
ST	0.5903	**0.6066**	2.77%
E-M	0.3716	**0.5871**	**8.15**%
DCG-32	0.5778	**0.5871**	1.61%

(e) PSB (1815 Objects in 161 Classes)

	PAN	PAN + HC	Improvement
DCG	**0.8050**	0.8006	-0.54%
NN	0.9013	**0.9038**	0.28%
FT	0.5365	**0.5509**	2.68%
ST	**0.6871**	0.6648	-3.26%
E-M	**0.4246**	0.4227	-0.44%
DCG-32	0.6567	**0.6599**	0.49%

(f) ESB (800 Objects in 42 Classes)

Table 3.4.1.: Evaluation measure results for all tested databases: only using the panorama descriptor (PAN) and using our algorithm combining the panorama descriptor with a hierarchical clustering (PAN + HC). In bold the best performance and significant improvements ($> 5\%$). The biggest and most representative database (SHREC2014) is improved in all measures by using our algorithm. In some smaller databases we could find worsening in a few measures and analyzed the reasons in the evaluation. In general, our automatic algorithm shows a clear improvement on average over all results.

- SHREC2014: The evaluation database for the SHREC 2014 track [LLL*14]. Note that this is a combined database and includes several other databases and different kind of objects. It is by far the biggest tested database.
- SHREC2007: The evaluation database for the SHREC 2007 watertight Track [GBP07]. This database includes articulated objects classes (e.g. hand class).
- NIST: The shape benchmark from the national institute of standards and technology (NIST) [FGLW08], includes generic objects of good quality and similarly sized classes.
- NTU: Model retrieval database from the National Taiwan University (NTU) introduced as evaluation data set of the lightfield descriptor [CTSO03], includes generic objects of good quality and differently sized classes.
- PSB: The Princeton Shape Benchmark (PSB) database consists of the PSB 'train' set and the PSB 'test' set. While the original publication of the Panorama-descriptor only uses the 'test' set, we use **both** sets of the Princeton Shape Benchmark [SMKF04] , including generic objects of very differently sized classes and varying object detail and quality.
- ESB: The Engineering Shape Benchmark (ESB) consists of CAD objects. [JKIR06]. It is different from other databases as it does not include generic natural objects and generic man-made objects, but engineering-part objects, e.g. bearing blocks and machined blocks and the given classification is also more functionality oriented (i.e. engineering-parts with the same functionality belong to the same class).

For the evaluation we use the precision-recall plot (Figure 3.4.1) to show the overall performance of our algorithm taking into account all 13271 queries. Then we use 6 well-known measures (Table 3.4.1) to differentiate between the performances in different databases. The 6 measures used for our evaluation are the following:

- DCG: (Normalized) discounted cumulative gain. This measure is based on the gain of each retrieved object. The gain of a retrieved object is dependent on the position in the retrieval list. The first object has the highest gain and the gain decreases if more objects are retrieved. The measure sums the gain of all correctly retrieved objects. It measures the overall performance and considers the whole retrieval list.
- NN: Nearest neighbor. Measures if the first retrieved object (not equaling the query object) belongs to the correct class.
- FT: First tier. The percent of members belonging to the class of the query object retrieved within the first m objects, where m is the number of objects within the class of the query object.
- ST: Second tier. Similar to FT, with the difference that m is doubled.
- E-M: E-Measure. This measure is computed from the precision/recall values. It measures the overall performance and considers the whole retrieval list.
- DCG-32: DCG for the first 32 retrieved objects. The reasoning behind the DCG-32 is that the first search page can contain about 32 objects and every user is considered to be willing to look at the results of the first page. Hence, the DCG-32 is the 'first-page-measure'. The FT has a

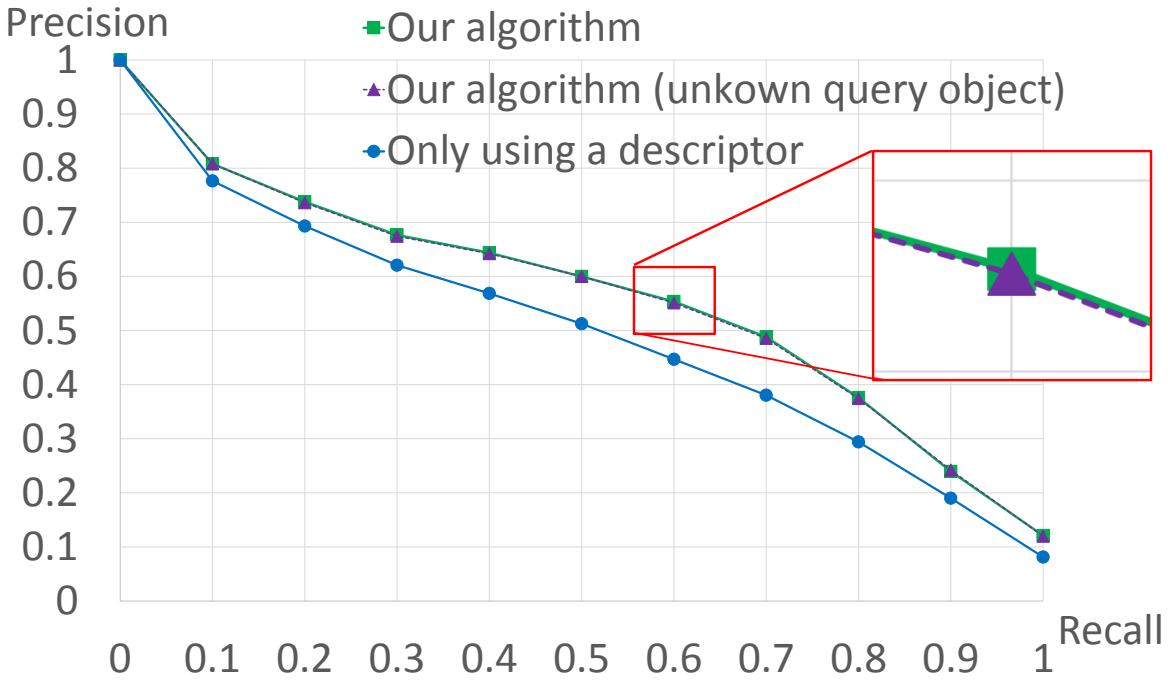


Figure 3.4.1.: The Precision-Recall plot for all queries of all 6 Databases. There are 3 curves present in this plot, while two curves mostly overlap each other. The blue curve with lower precision only uses the descriptor for the retrieval. The two overlapping curves with higher precision represent the two variants of our algorithm: with and without the query object being part of the clustering.

similar semantic, with the difference that the FT is directly dependent on the number of object belonging to the retrieved class.

Figure 3.4.1 shows the 11 point precision-recall plot averaged over all queries for all databases. The recall is the percent of the total class members retrieved and the precision is the percent of retrieved objects belonging to the correct class. For each of the 11 recall marks the precision over all 13271 queries is averaged. We can see that the hierarchical clustering improved the precision on average over the whole plot.

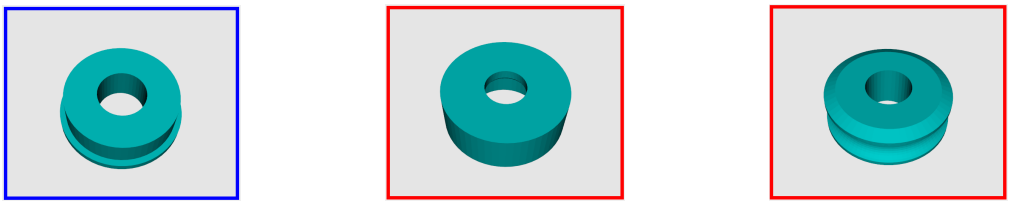
In this plot we also show the results for our algorithm for an unknown query object (described in Section 3.4.3). This evaluation was computed in a leave-one-out fashion: a hierarchical clustering of the database excluding the query object itself was used for every query object. As we can see the precision is only slightly lower if the query object is not part of the clustering.

The evaluation for each of the 6 databases are shown in detail in Table 3.4.1. In the Table we do not explicitly show the results for an unknown query object, as they are nearly the same if the object is part of the clustering. Again, for the SHREC2014 database the results are improved significantly within all measures. The results for the NTU database are also improved in all measures. For the SHREC2007, NIST and PSB database the results are only slightly worsened in the NN measure. All other measures are improved in every database except the ESB database. In the ESB database the ST is worsened and also the two overall-measures DCG and E-M. The 'first-page-measures' FT and DCG-32 are showing improvements in every database. Clearly the biggest database with the highest diversity of object types (SHREC2014) shows a significant overall improvement. Therefore, in the general unsupervised case our algorithm should be favored over the common method.

FT: The most interesting result is that the FT shows the most significant improvement of all measures and is even improved in the ESB database. As we stated in the description of the algorithm the confidence of the clustering is higher on the lower levels of the hierarchy (more clusters) and lower on the higher levels (less clusters). The primarily retrieved objects (measured with the FT) are therefore relying on the lower levels of the hierarchy and for this reason are always improving the result. The secondarily retrieved objects (measured with the ST) rely on the higher levels of the hierarchy and are therefore less reliable. One possibility to use this knowledge is to set a threshold for a maximal hierarchy level being used for the retrieval. Everything above this level is treated as a single cluster. Unfortunately, the exact level of the hierarchy differs much from object class to object class and is also dependent on the total number of objects in this class. Hence, this threshold must be set for each class individually.

NN: Other clustering methods have shown similar results concerning the nearest neighbor measure. The clustering method of Tatsuma et al. [TA09] could also improve FT ST and DCG, but the NN measure was worsened for every number of cluster k. Note that in contrast to our approach all measures, including FT ST and DCG are worsened if the number of clusters k are not chosen well. Again emphasizing the value of our parameter-free hierarchical clustering concept.

A clustering generally has the characteristic that similar objects are clustered together and are better separated from other objects, This is the purpose of a clustering. Hence, for a single object the quality



(a) Bearing like part (b) Cylindrical Part (c) Pulley like part

Figure 3.4.2.: The blue bordered object is of the class 'bearing like part' from the ESB database. The other two objects are of different classes 'cylindrical part' and 'pulley like part'. The high overlap of classes together with the high variation within classes makes the ESB database more difficult to cluster.

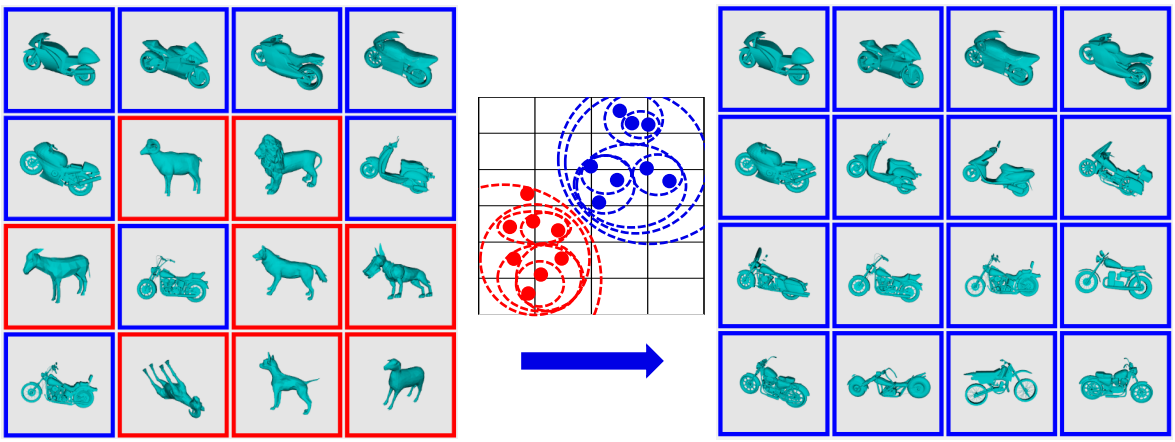


Figure 3.4.3.: Retrieval results from the NIST database for a motorcycle object query. The results on the left are computed by using only the distances of the Panorama-descriptor. The results on the right are computed by our algorithm using the additional information of the hierarchical clustering. In both result sets the top leftmost object is the query object and the results are ordered primarily from left to right and secondarily from top to bottom. In the clustering the motorcycle class and the quadruped class are separated from each other. Our algorithm combines this information with the original distances improving the retrieval results significantly.

of the clusters in which they are included explicitly depends on the quality of the surrounding objects. If the object has a wrong NN and no objects of the same class are anywhere near, the clustering will not resolve that. If the object has a correct NN and lies near several objects of the same class, the objects will belong to a shared cluster and the NN will probably stay correct.

The only case where the NN changes are border cases. The object lies at the border of a class, meaning that it belongs to a class but the appearance is not the class-typical appearance. In the border cases many circumstances decide if the object is correctly put into the own class or primarily put into a wrong but similar class. If it is put into a wrong class, the NN changes from a correct class member to a wrong class member. However, in the border case the correct class members are within a nearby cluster, hence the FT and DCG-32 improve nonetheless.

ESB: The measures for the ESB database generally show a low impact of the clustering since all measures only changed slightly. The FT is improved, but the ST is worse. Hence, we can conclude that in the ESB database the initial results (first-page-results) are better but the following retrieved objects are worse. The reason for the bad results after the first page is that the overlap of different classes in the ESB database is much higher and differs from any other database: There are shapes that reappear on different classes leading to a higher overlap and convolution of classes. One example are torus-like objects which can be found in the 'bearing like part' class but also in the 'cylindrical part' and 'pulley like part' class (See Figure 3.4.2). Note that overlapping classes with hierarchical dependency (e.g. sheep - quadruped) work well within the clustering, but a high overlap of independent classes is problematic. This high overlap of classes lead to bigger clusters of mixed classes. Even though the most similar objects of the correct class are primarily clustered together (FT improves), many following retrieved objects are of the overlapping class, which has the consequence that the correct class members are appearing later in the retrieval list (ST worsened).

Single Query Results: In this part we analyze 3 representative results for single queries, to show how and why our algorithm improves the results and in which case it is not improved.

Figure 3.4.3 shows a query of a motorcycle object in the NIST database. One can see that the retrieved objects belong to exactly 2 classes. Obviously, these 2 classes are very near to each other in high-dimensional panorama descriptor space. Only taking the direct distances of one object to another leads to the mixed result list on the left side. Using our clustering, the two classes are clustered and separated from each other, resulting in a perfect result list.

This example also shows the semantic advantage of a hierarchical clustering. A single clustering could lead to a cluster of 'motorcycles' without any further separation. If we take a closer look at the objects of the result list and their order using our algorithm, we can see that there is also an ordering within the 16 retrieved objects of the class 'motorcycle'. The first 5 objects look like racing motorcycles, while the next 2 objects are scooter like motorcycles, followed by motorcycles with a more chopper/cruiser like style. Therefore, the retrieved cluster 'motorcycle' includes smaller clusters/classes not directly addressed by the database class labels of the NIST database. Still, the hierarchical clustering naturally obeys this intrinsic order.

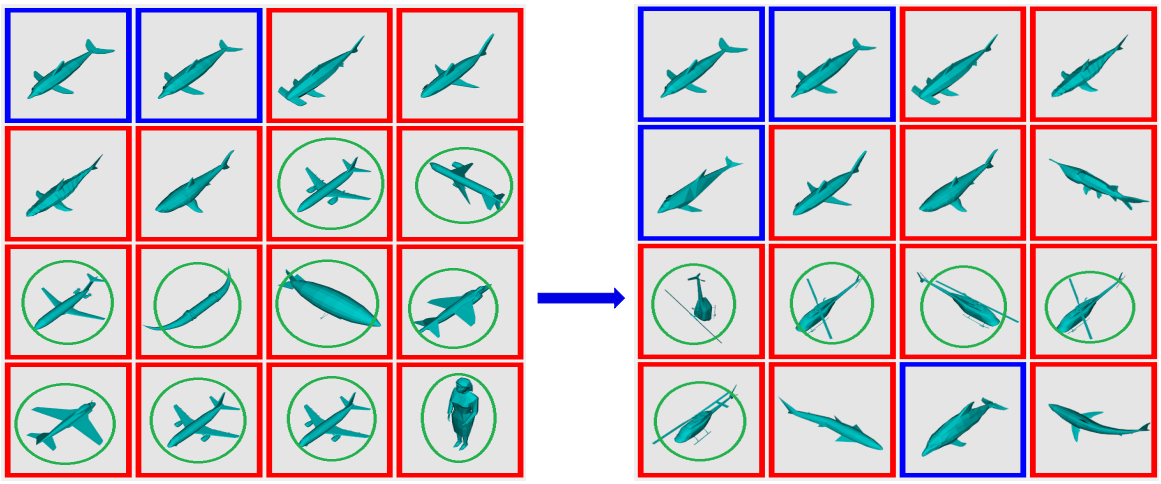


Figure 3.4.4.: The retrieval results for a dolphin object query in the PSB database. With green circles we marked objects semantically unrelated to the dolphin. We can see that our algorithm follows the hierarchical nature of classification of 3D objects. Using our algorithm improves the results for the 'dolphin' class, but also improves the result list in general, as more 'shark' objects and and less unrelated objects are present.

The example of Figure 3.4.4 shows an interesting propriety we experienced with many query objects: Even if the retrieval is only improved slightly in respect to the given classification the retrieval list is improved semantically. This happens because the databases used for testing usually has a single class for every object and it is hard to measure how wrong the classification is, if the class is not the correct one. However, the dolphin is semantically nearer to the shark than to other objects present in the result list. Therefore it is still an improvement that less unrelated objects are present in the result list and more sharks are visible.

Figure 3.4.5 shows an example for a query which was not improved. The query object belongs to the class 'rifle' from the PSB database. However, all other rifle objects are too different (or at least the descriptor could not catch this similarity). The result is that the rifle is inserted into wrong clusters (primarily pistol clusters).

Runtime: The runtime for computing the descriptor and the distances for each object is completely descriptor dependent. Note that for many descriptors (including the panorama descriptor) the descriptor computation for each object is time consuming but can be precomputed as this happens only once per object. The descriptor comparison is very fast for most descriptors as this is only a vector distance. The retrieval algorithm itself (with precomputed clustering) only involves descriptor comparisons. Furthermore, the retrieval algorithm results can also be precomputed during the clustering.

The hierarchical clustering however has a relevant runtime. In our implementation the runtime for the hierarchical agglomerative clustering is around 1 second for the PSB database and 35 seconds for the

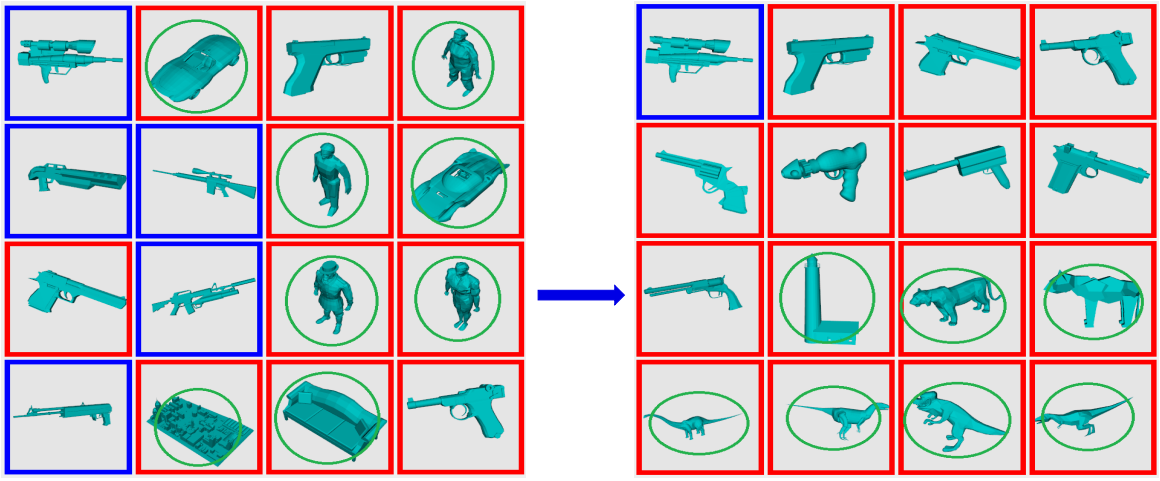


Figure 3.4.5.: The retrieval results for a rifle object query in the PSB database. With green circles we marked objects semantically unrelated to a rifle. This is an example for the worst case showing the limitations of our approach. The query rifle object does not look like other rifles and is put into the wrong clusters in the hierarchy. On the left we can see the results without our algorithm showing a mix of many object classes including humans, cars and others. Four correct rifle-class objects are included here besides the query object itself. On the right we see the results using our algorithm and no correct rifle object is retrieved even though the first 8 retrieved objects are all pistols.

biggest database (SHREC2014). This can be a major problem for much bigger databases and represents the only drawback of our approach. However, if an application cannot wait for the clustering to finish, it can use the unfinished clustering. Our algorithm allows to use the bottom levels of the clustering and simply treat the unfinished upper levels like the top level, where all objects belong to a single cluster. Also, to compute a fast approximation of the clustering, it is possible to change the clustering, so that it uses an approximate nearest neighbor calculation for the cluster merging instead of an exact nearest neighbor calculation.

3.5. Descriptor based Retrieval with Procedural Models

To use a procedural model $PM_{\{p\}}$ in a 3D object retrieval application we need to define a measure for the distance of the procedural model to a database object.

Comparing an object of the database with every possible instance generated from the procedural model is impossible, as these are infinite. Therefore we need to sample the space of possible objects and generate a set of instances $\{i : i \in PM_{\{p\}}\}$.

Then we can compare each sampled instance i to a database-object o. For this purpose we compute a descriptor $d(..)$ for both. The difference of the descriptors $\delta[d(i), d(o)]$ represents the difference between the objects.

For the final retrieval result of a procedural model $PM_{\{p\}}$ we aggregate the comparison of each instance. We define the difference of an object to the procedural model as the distance to his nearest neighbor in the object space of the procedural model:

$$\delta[PM_{\{p\}}, o] = min\{\delta[d(i), d(o)] : i \in PM_{\{p\}}\}$$

3.5.1. 3DOR with a Manual Procedural Model

To enable the generation of a reasonable small set of instances of the procedural model we sample the 3-parameter representation $PM_{p_1 p_2 p_3} \subset PM_{\{p\}}$ of the dining chair defined in Section 2.3. For each dimension we take one sample each 0.05 of the restricted value range $p_i \in [0.5, 1.0]$. This results in 11 sample values for each dimension and 1331 instances of the procedural model.

We use the Princeton Shape Benchmark (PSB) [SMKF04] for the evaluation of our approach. We combine the procedural model of the dining chair with an appropriate descriptor and retrieve the most similar objects from the PSB database (1815 models). We consider every object classified as "dining chair" within the database as a correct retrieval.

As our approach can be combined with any descriptor, We have considered different descriptors. To use the full advantage of the procedural models, we considered different proprieties of the descriptors.

Every type of descriptor has different advantages and disadvantages. Graph based descriptors focus on the topology of an object and therefore are very good for objects which mainly are related by their topology (e.g., quadrupeds). They give poor results if the meshes are disconnected, with holes and include self-overlaps or if the searched objects have a high variety of possible topologies (e.g., vases).

Global feature based descriptors are mostly good at discriminating at a high level of difference, while local feature based descriptors are better at discriminating objects based on small (local) differences. View-based and hybrid descriptors are in between in respect to the level of discrimination. Considering our setting (PSB database), which includes objects represented as arbitrary mesh soups, we looked for a state-of-the-art descriptor which can handle these meshes. Also, as our procedural model is less detailed than the retrieved objects, we excluded the local feature based methods. We preferred the view-based, hybrid and global feature descriptors.

We also considered that the descriptor has to be rotation-invariant, translation-invariant and scale-invariant for the usage with a procedural model. However, most state-of-the-art descriptors are either invariant by construction or include normalization for archiving invariances. Therefore this consideration does not exclude many descriptors.

The one state-of-the-art descriptor with the best results among these is currently the Panorama descriptor [PPTP10]. It is a hybrid descriptor including geometric and view-based features.

3.5.2. Evaluation with a Manual Procedural Model

We show results of the Panorama descriptor in Figure 3.5.1 and 3.5.2. Each Precision-Recall plot shows the retrieval of the "dining-chair" class, which has 22 Members. Besides the retrieval with the procedural model we show 5 single queries in each plot. The retrieval queries use the (dining chair) models 807, 808, 809, 812 and 825 from the Princeton Shape Benchmark (PSB) Database. We evaluated all plots with all 22 dining chair single queries and (for diminishing overplotting) have chosen 5 Models which include the best and the worst results in all plots.

The results show that the Panorama descriptor indeed works perfectly well together with the procedural models. Several single chair queries already give good results with the Panorama descriptor. The retrieval performance with the procedural model shows an even better result than any single query retrieval. Note that there is one chair which is hardly retrieved by any query as it differs much from the other chairs. Still, the procedural model retrieval retrieves this chair much better than any possible single query.

Our Results show that the retrieval with procedural models work very well with the right procedural model representation and the right descriptor.

As we only evaluated the procedural model of a chair we also have to investigate how much the retrieval gain changes with other classes and how the retrieval changes with more detailed procedural models.

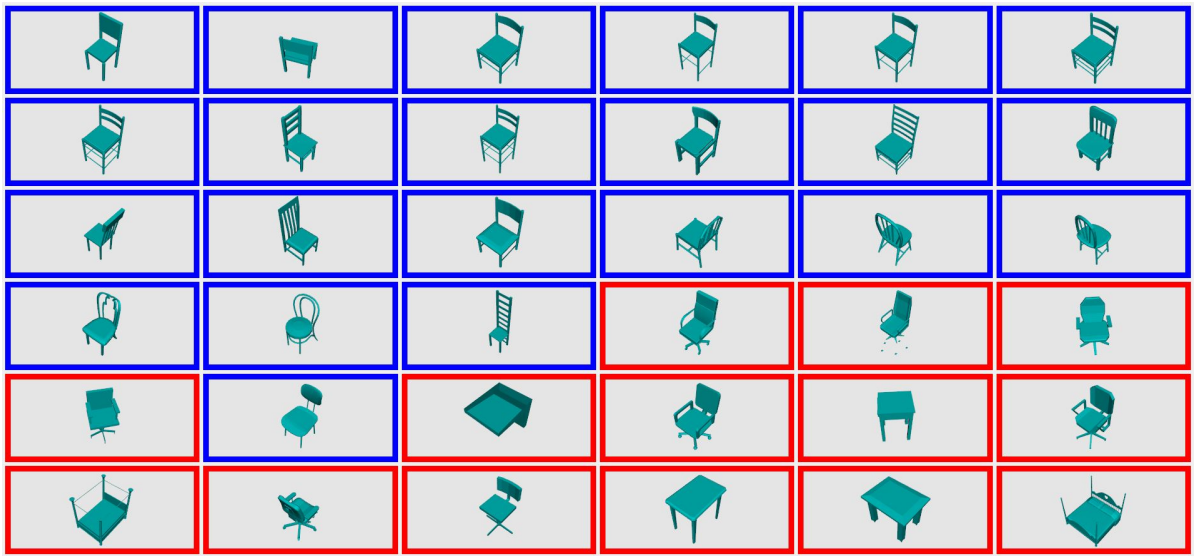


Figure 3.5.1.: We show the retrieval results using the procedural model "dining chair" in combination with the Panorama descriptor in the Princeton Shape Benchmark database. The first 36 retrieved objects are shown. All 22 members of the class "dining chair" are within the result. The results are ordered primarily from left to right and secondarily from top to bottom.

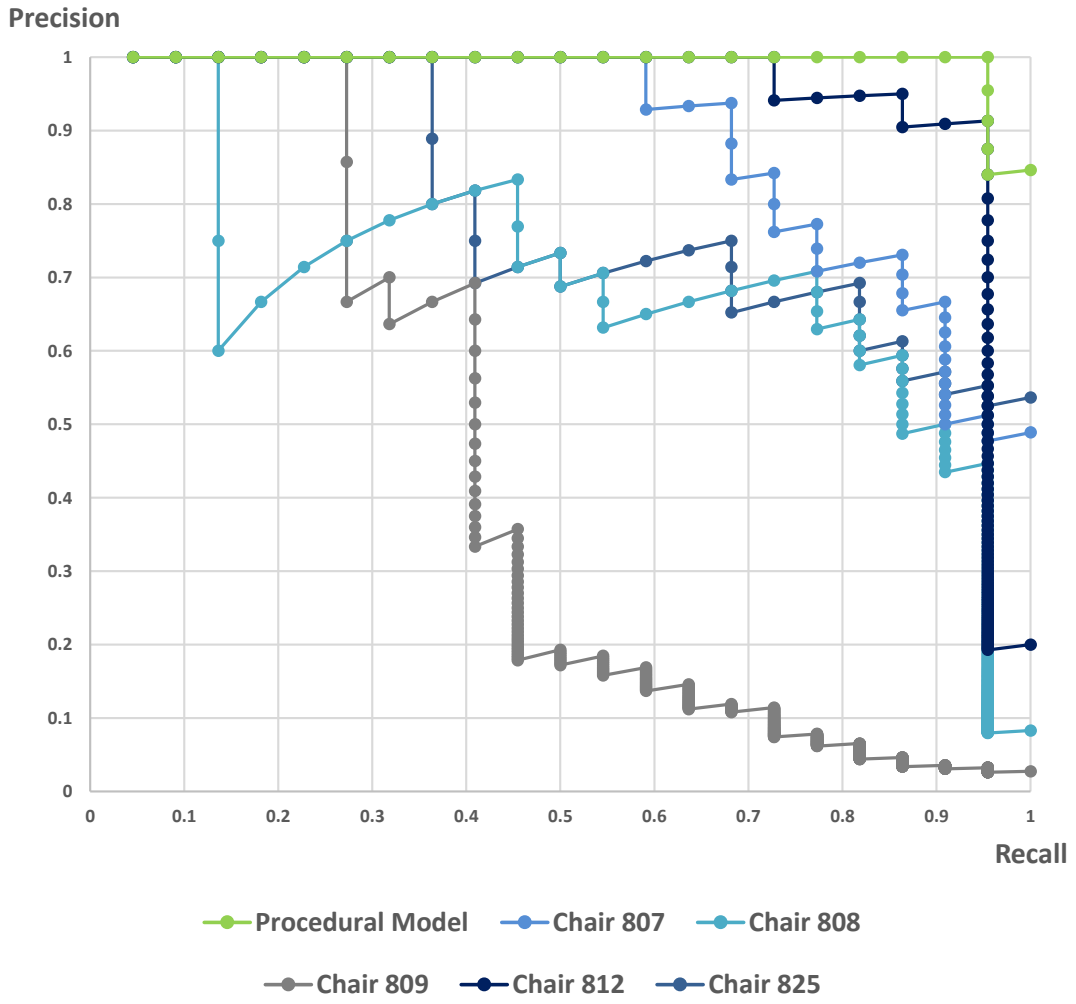


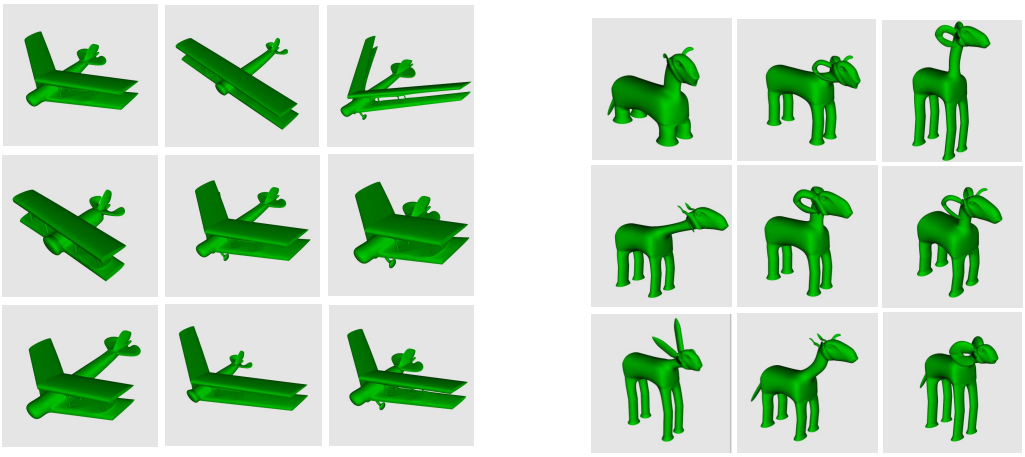
Figure 3.5.2.: Precision-Recall plot for the retrieval of the "dining chair" class (22 members) in the whole Princeton Shape Benchmark Database. The procedural model "dining chair" was combined with the Panorama descriptor. The single chair queries also used the Panorama descriptor. The 5 single queries plotted are chosen from the 22 possible queries and include the best and the worst results of the single queries.

3.5.3. 3DOR with a Semi-Automatic Procedural Model

In this section we evaluate our 3DOR approach with procedural models, using two more complex objects generated with our semi-automatic procedural model generation. The quadruped and biplane procedural models are introduced in Section 2.4.

These objects are not only more complex than the chair, they also have more parameters. It is possible to make a random sampling in the procedural model space as the range of the parameters is fixed. However, it is more reasonable to make a more regular well-defined sampling. We define several variations for different parts of the object and take every possible combination as sampling point. For the quadruped we take 3 variations of: the body, the legs, the ears, the neck and the snout. Additionally we take 2 variations of the tail and 6 variations of the horns. In total we have 2916 instances of the quadruped procedural model.

For the biplane model we take 3 variations of: the plane length, the wings length, the wings width, the wings angle, the back-wings size, and 2 variations of the plane-nose and the wheels. In total 972 instances of the biplane procedural model. For both procedural models we show several examples of the sampled instances in Figure 3.5.3.

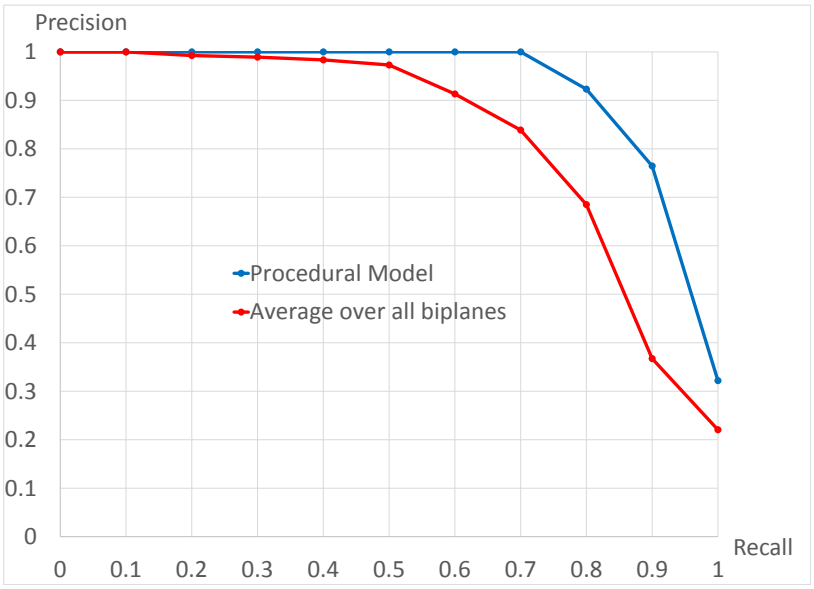


(a) biplane samples for 3DOR application

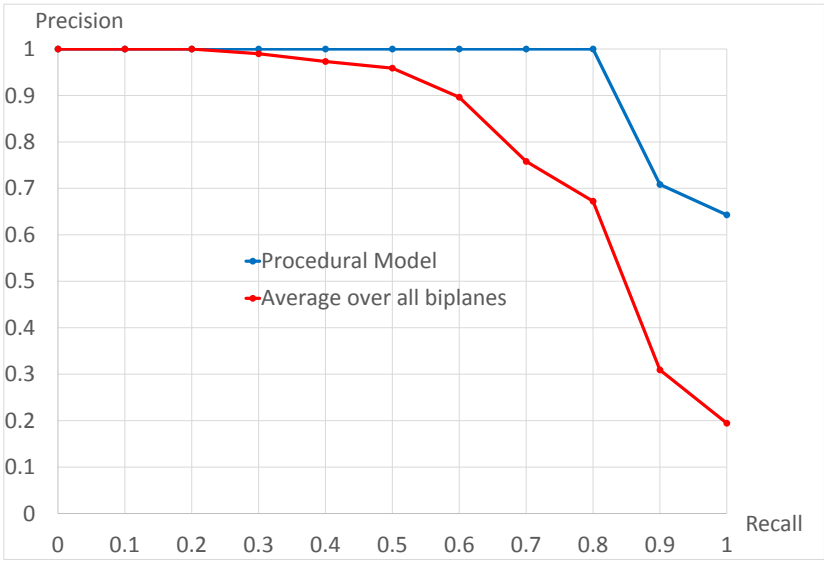
(b) quadruped samples for 3DOR application

Figure 3.5.3.: We sampled the procedural models for the 3D object retrieval. Every specific set of values for the parameters of a procedural model corresponds to a 3D object.

We use 2 databases which include quadrupeds and biplanes: the National Institute of Technology (NIST) database [FGLW08] and the Princeton Shape Benchmark (PSB) database [SMKF04]. The NIST database includes 18 biplanes and 18 quadrupeds in a total of 720 Objects in 40 Classes. The PSB database includes 28 Biplanes and 31 Quadrupeds in a total of 1815 objects in 161 or 53 classes: for the biplane case we take the 'base' classification and for the quadruped we take the 'coarse1'

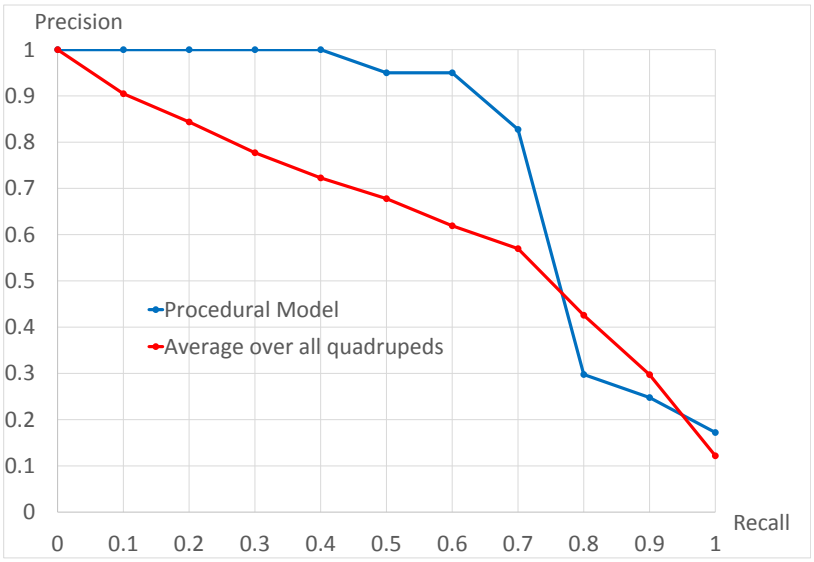


(a) PSB - biplane

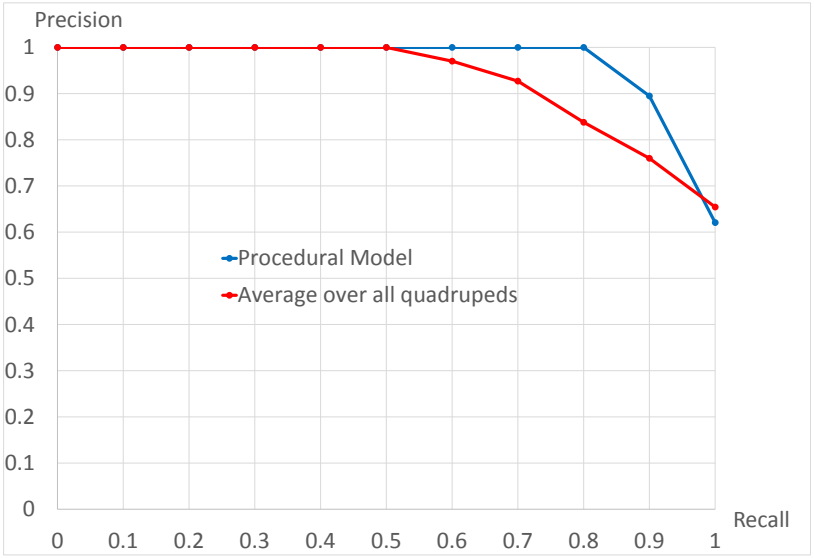


(b) NIST - biplane

Figure 3.5.4.: The precision-recall curves comparing the retrieval with the procedural model of a biplane to the retrieval with a single example. These are shown for two databases: the Princeton Shape Benchmark (PSB) in (a), the National Institute of Technology (NIST) database in (b).



(a) PSB - quadruped



(b) NIST - quadruped

Figure 3.5.5.: The precision-recall curves comparing the retrieval with the procedural model of a quadruped to the retrieval with a single example. These are shown for two databases: the Princeton Shape Benchmark (PSB) in (a), the National Institute of Technology (NIST) database in (b).

classification, as only the coarser classification includes the quadruped class. For the computation of the distance of objects we use the Panorama-Descriptor [PPTP10].

3.5.4. Evaluation with a Semi-Automatic Procedural Model

We evaluate our approach by comparing the retrieval with a single object to the retrieval using the procedural model. For the single object retrieval we compute the precision-recall for every single object belonging to the class. We then average the precision over all members.

In Figure 3.5.4 and Figure 3.5.5 we show the results as precision-recall curve for the procedural model retrieval and the averaged single example retrievals.

In Figure 3.5.4 we see the results for the biplanes. The procedural model retrieval raised the precision significantly for both databases. In Figure 3.5.5 the quadruped model could also increase the precision for both databases. However, we can see that the precision is lower in the area with recall > 0.7 in the PSB database.

The results also show a known difference between the NIST and the PSB database. The NIST database consists of equally sized classed with well limited variations within each class. The PSB database was designed to be very challenging and includes objects of varying quality, varying sizes of the classes and also a high variation of form within single classes. We show an example of this property in Figure 3.5.6.

For this reason the results are generally better for the NIST database. In this database the variations of the biplanes and quadrupeds introduced by the procedural models could almost catch all class members. In the PSB database there are cases where the quadruped but also the biplane procedural model does not include a variation covering the database object. This explains the lower precision in the area with high recall in In Figure 3.5.5 (a). So, even if the procedural model is able to incorporate a huge class variation it is explicitly limited to this variation. This is a limitation as the procedural model is not able to solve the semantic problem of infinite possibilities of variations of an object class. However, this can also be the exact desired behavior for the retrieval: the user searches for objects which are similar to any variation defined by the procedural model.

The results of the chair, the quadruped and biplane object show that the retrieval with procedural models using a descriptor gives very accurate results when the procedural models are defined appropriately. Therefore, we reason that procedural models generally have a very high potential for this application. Nevertheless, the quality is dependent on the quality of the procedural model itself.

(a) biplane from PSB database (b) quadruped from PSB database

Figure 3.5.6.: The PSB database intentionally includes uncommon class members and bad quality meshes.

3.6. Retrieval and Classification with Deep Learning of Procedural Models

In the previous Section 3.5 we introduced a comparison method to directly measure the difference of a procedural model to any 3D object. This can be used for 3D object retrieval but it could also be used for a classification task: an unknown 3D object is labeled with the class of the most similar procedural model. Using this method is always possible, since it is independent of the number of objects in the database or the number of procedural models or the number of instances generated with a procedural model.

Nevertheless, in presence of a large suitable learning database, which contains sufficient examples for all classes, deep learning methods have shown to outperform previous approaches [SMKLM15, MS15, WSK*15]. If we are able to use procedural models as data foundation for a learning process we could increase the accuracy of classifications. The only drawback is the time consumption for the preceding learning phase. Therefore, we propose a method combining deep learning with procedural models without any additional learning database.

We propose to use the very deep convolutional neural network (CNN) Google inception network [SLJ*15] to directly learn the 3D object with rendered images of the object. The last fully connected layer of the inception network can be retrained with a relative small amount of 3D data. Also the retraining is tremendously faster than training a network from scratch.

3.6.1. Deep Learning with Procedural Models

We retrain the last fully connected layer with a randomized set of images of rendered views of the 3D object. Also, we additionally generate random variations of the 3D object within these images.

Each procedural model represents an object class. For each class we generate 1000 variations (3D mesh). We vary each parameter of the procedural model completely random within the parameter range. For each of the 1000 variations we generate 10 images. In total, we use 10 000 images per class to train the network. In our evaluation we use 10 classes, therefore the network is trained with a total of 100 000 images.

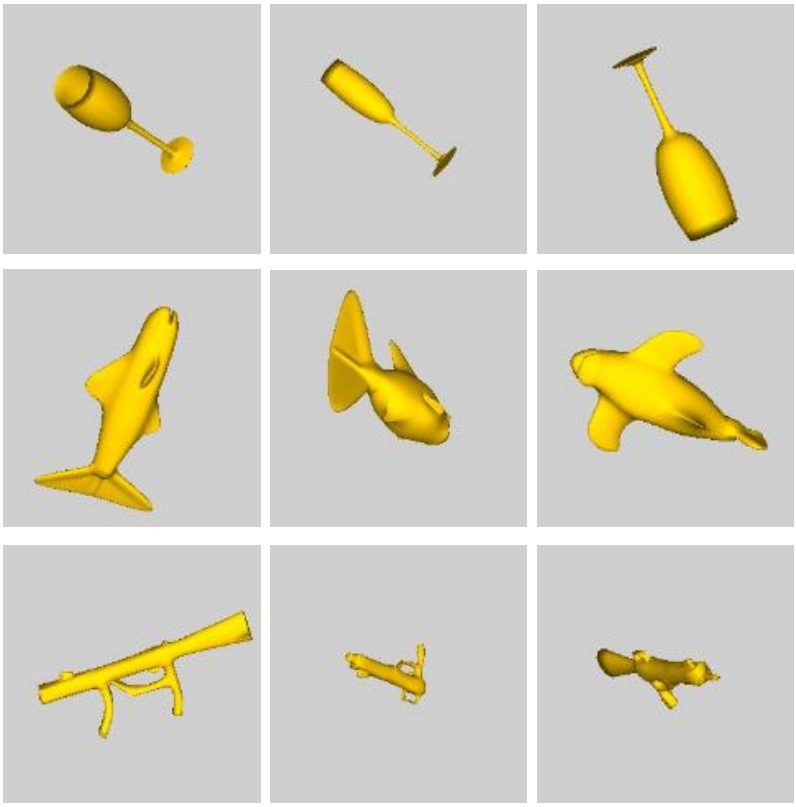


Figure 3.6.1.: Examples of randomly generated images for the learning process

3.6.2. Image Generation

We use rendered images of the generated 3D objects with random perspectives. The resolution of the images is 256x256, this is the input resolution for the CNN. We use an orthogonal projection (all boundary planes are set to $[-2, 2]$) with smooth shading. Like [SMKLM15] we noticed that different illumination setups and colors did not make significant differences in the results. We have chosen a light yellow color for the objects and a grey background.

Before generating the images the 3D object is first normalized in the global coordinate system. The average position of the vertices of the mesh (center of mass) is translated to the origin. The object is scaled, so that all coordinate values are within -1 and 1. Note that our orthogonal projection does not only include the cube of -1 to 1. The reason is that the object is normalized to fit into this cube at the start, but it might get out of the cube after the following random rotation and scaling.

Each image represents one 2D sample of the 3D object. Therefore, we include a random rotation and scaling of the object. The scaling of the object corresponds to a zooming of the image. The rotation

is chosen completely random and the scaling is between 0.8 and 2. This means that we allow more zooming in than zooming out. We noticed that the results get worse if very small scales are allowed. We reason that at some point if the object on the image gets too small the network learns less from the image.

3.6.3. Object Classification

With the retrained network we can classify any new image. The output of the network for a single image is a class probability for each trained class. To classify a new 3D object we generate 10 images of random views and average the classification values for each class. The 3D object is then put in the class with the highest value.

3.6.4. Evaluation and Discussion

To evaluate our deep learning approach on rendered images, we constructed 10 procedural models. Figure 3.6.2 shows the models and Table 3.6.1 shows statistics of the models.

	po	to-ops	par-ops	par
fish	1348	150	68	13
glass_with_stem	332	44	11	6
helicopter	1560	294	184	12
gun	888	161	54	15
table	614	87	51	6
spider	2976	197	137	9
sword	568	64	30	7
office_chair	1724	193	86	8
bird	2040	241	134	13
bicycle	4966	510	189	10
po = number of polygons to-ops = total number of operations par-ops = number of parameterizable operations par = number of parameters				

Table 3.6.1.: The properties of the procedural models

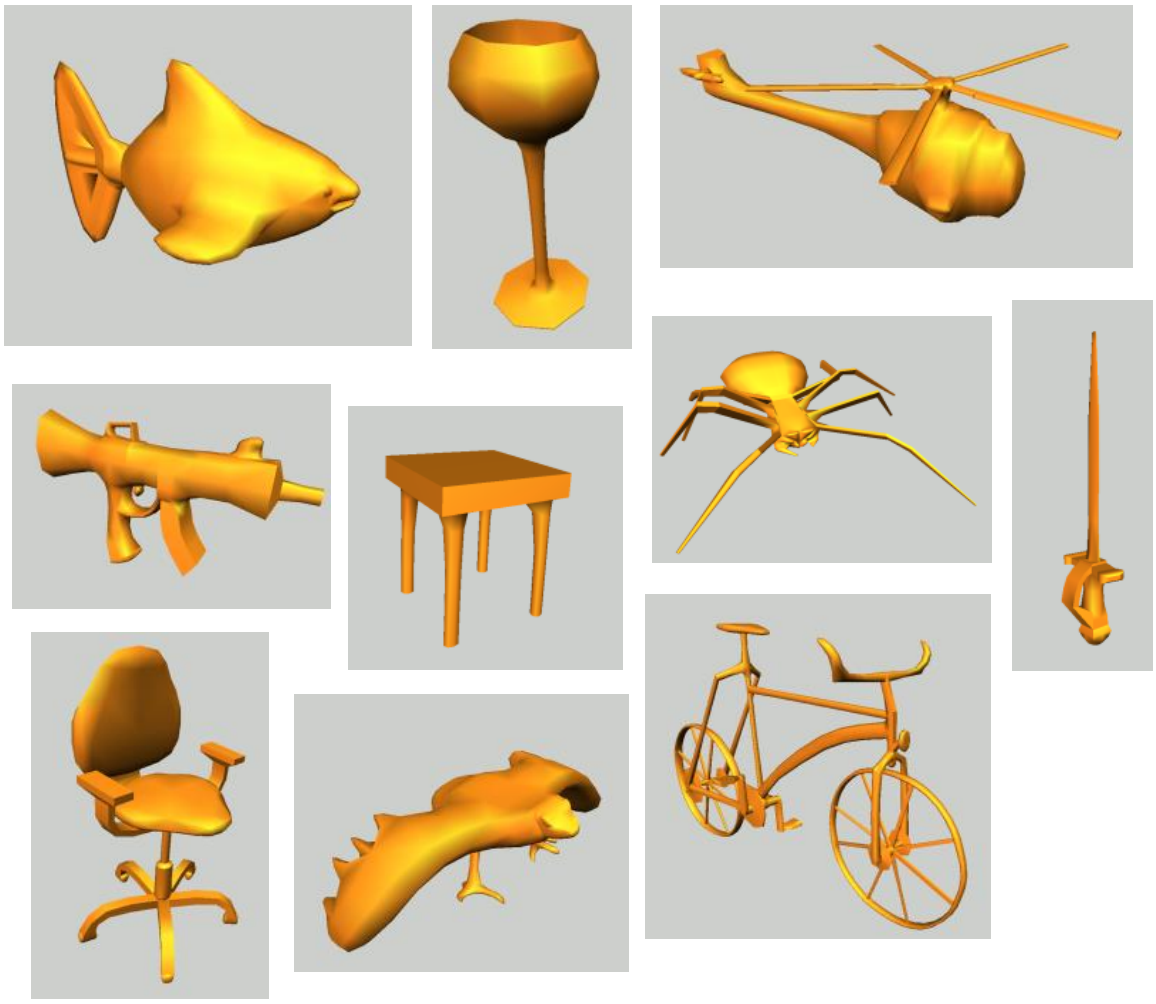


Figure 3.6.2.: All models that were used to evaluate the classification.

	NIST	PSB	Total
others	504	1562	2066
fish	18	17	35
glass_with_stem	18	9	27
helicopter	18	35	53
gun	36	39	75
table	36	63	99
spider	18	16	34
sword	18	31	49
office_chair	18	15	33
bird	18	21	39
bicycle	18	7	25
total within classes	216	253	469
total	720	1815	2535

Table 3.6.2.: The number of objects for each class in the PSB and the NIST database.

The 10 procedural models represent 10 different classes. To evaluate our approach we use the NIST database [FGLW08] and the princeton shape benchmark (PSB) [SMKF04]. Each of the 10 classes are present in both databases. Table 3.6.2 shows the number of objects of each class in the databases. We use all objects of both databases together as our input data for our evaluation. Note that we only took classes premade within the given databases and fitting to our objects. For this reason our spider class includes non flying insects (even though a spider is not an insect). The NIST only has a single class with spiders and insects. We trained our network with a total of 100000 images (10 classes $\cdot$ 10 images $\cdot$ 1000 variations). The retraining only needed about 10 hours on a casual pc (Intel i7-3930K 3.2GHz).

We used our algorithm for 2 different scenarios: First, the classification of all database objects of one of the 10 classes. Second, a classic 3D object retrieval scenario with all database objects.

Classification: In the classification scenario every classified object is sorted into the class with the highest probability. We classified every object of the databases that belong into one of the 10 learned classes and measured the overall classification accuracy. Figure 3.6.3 shows the accuracy for all 10 classes resulting in the average accuracy of 86.14% (404 of 469 objects are classified correctly).

3D object retrieval: The output of a 3D object retrieval query typically is a list of retrieved objects, ordered from the most similar to the least similar. The neural network outputs class probabilities when we input an unknown 3D object. We directly use these class probabilities to sort the list of retrieved objects. Hence, a class label is the query itself and the first object of the retrieval list is the 3D object with the highest class probability for this class. Here we include all objects of all databases, also the objects that do not belong into any of the 10 classes.

Figure 3.6.4 shows the precision recall curve for the 3D object retrieval using the 10 class labels as query. We compare this result with two other possible approaches: In the first case we use the panorama

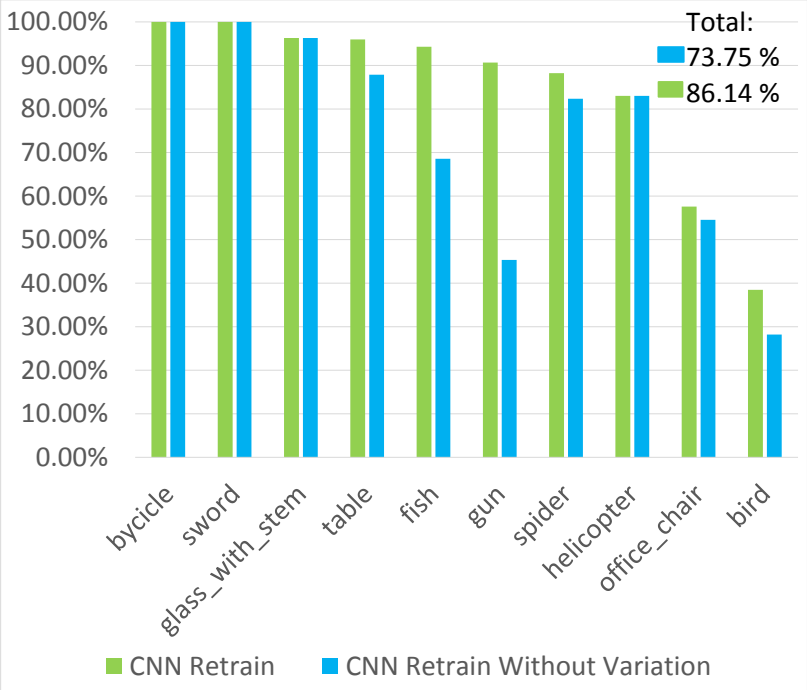


Figure 3.6.3.: The classification accuracy for all classes

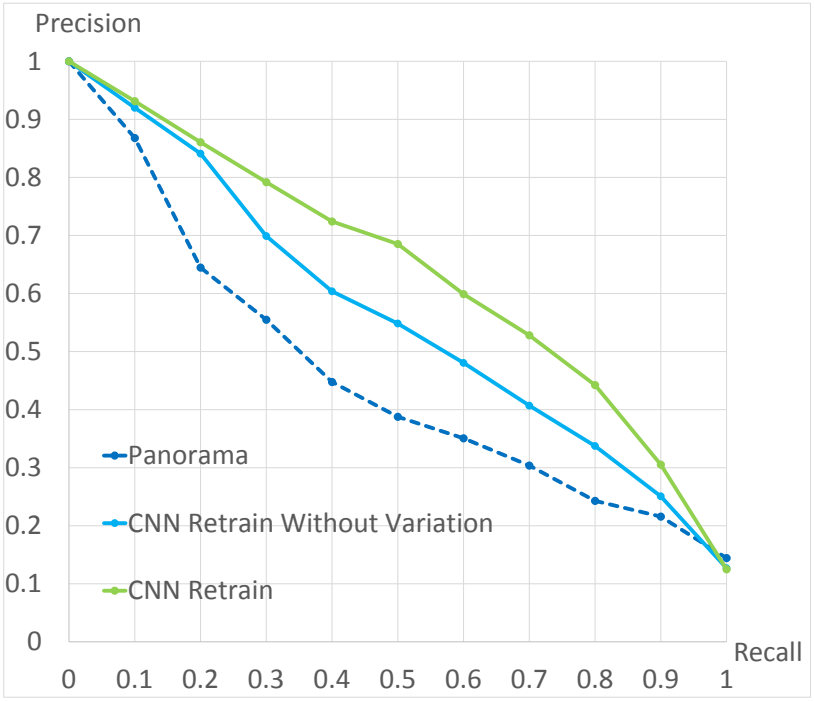


Figure 3.6.4.: The precision-recall curve for the 3D object retrieval scenario

Figure 3.6.5.: Images of falsely classified objects. The variations of the procedural model could not cover every possible type of variation.

distance directly for the 3D object retrieval. The panorama distance was originally developed for 3D object retrieval. We use the default object generated with a procedural model as query for the database. The distance from this object to all objects in the database is calculated and the retrieval list is sorted respectively. In the second case we user our deep learning algorithm but do not use any variations of the procedural model. We simply generate all images without changing any parameters and retrain the network with these images.

Discussion: The classification and the 3D object retrieval show several properties of the approach. An important insight is that including variations into the learning process generally leads to improvements. This is not as trivial as it might seem at first glance. We tested several other possibilities of image generation, including random translations and higher variations of scaling and found out that the results get worse when too much varying information is present in the images. It is easier for the neural network to learn the class when the images are more consistent. At the same time a good amount of variability is needed in the images to prevent overfitting and promote generalizability. Therefore it is a tradeoff. In this context it is more astonishing, that our results clearly show that the addition of object variations enhance the results in all cases. For every class and for every recall level in 3D object retrieval the precision is higher or equal when including the variations.

The average classification accuracy is 86%. This is comparable to state-of-the-art approaches like [SMKLM15] achieving 83-90% accuracy on the classification task. Only the office chair and bird classes achieved a lower accuracy. The simple reason for this is that the database objects are not similar to the initial procedural model at all. In Figure 3.6.5 we show some examples of falsely classified objects. This shows a general limitation of the method: the parameters and variations cannot generally compensate exceptional variations of the objects.

The precision-recall curve also shows a comparison with the panorama distance. Just like other deep learning approaches, our approach outperforms the classic geometrical approach, even though the

panorama distance is among the best geometrical distance measures. Summarizing, our deep learning retraining approach is fast and works with less data than a full network learning and still generates comparable results.

3.7. Parameter Estimation with Procedural Models

In the previous section we propose a method to classify unknown objects by only using procedural models as data foundation. In the classification task the user is interested in additional information about unlabeled unknown objects. The assignment of class labels enables the possibility to further organize the raw 3D object data to eventually inspect or gather the desired set of objects for further processing.

The procedural model allows to enhance the classification process with an additional parameter estimation to dramatically increase the amount of information gathered from unknown objects. This is one of the unique advantages of using procedural models for retrieval and classification. In addition to the general retrieval and classification, procedural models allow a much deeper structuring of unknown objects by estimating the parameters of the procedural model to fit to the unknown object. The result for a single unknown object is a class label and a set of parameter values describing the characteristics of the object (e.g. stem length of a glass). Also, with the specific parameter setting we can generate an instance of the object with the procedural model.

Figure 3.7.1 illustrates the full pipeline of using procedural models for the classification and the subsequent parameter estimation. Step 1 is the creation of the procedural Model. Step 2 is the classification with a deep learned CNN. The last step is the parameter estimation. We estimate the best parameters of a procedural model representing the same class as the unknown object. For example if the unknown object is classified as 'glass with stem', we use the procedural model of 'glass with stem' to estimate parameters of the procedural model, so that the generated instance has the highest similarity to the unknown object. The result of the example in Figure 3.7.1 is shown on the right side. We estimated the class, the parameters and also generated an instance of the procedural model (in yellow), with the best possible similarity to the unknown 3D object (in green).

3.7.1. Overview

In the parameter estimation we use the procedural model by extending the class label with additional information about the characteristics of the 3D object. The procedural model has several parameters to generate variations. We estimate those parameters for a new 3D object having the same class as the procedural model. The parameters can either be labeled by the user, e.g. 'wing length', or the influence

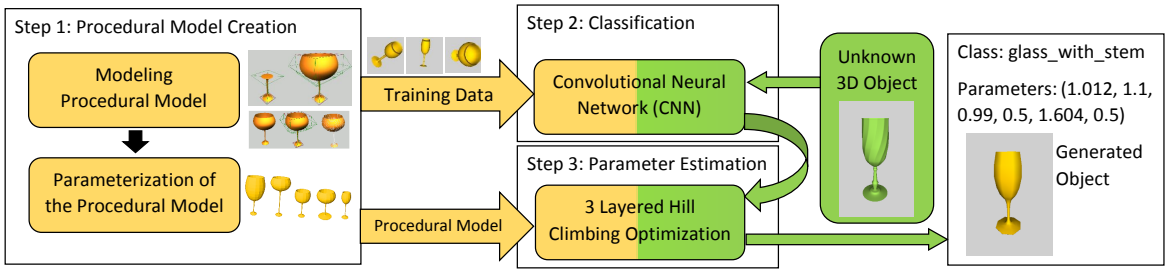


Figure 3.7.1.: The full pipeline: procedural models are created to represent a blueprints of an object class. These are used to classify and estimate the parameters of an unknown database object.

of the parameters can be shown visually to the user by generating exemplary objects for different values. In both cases estimating the parameters for an unknown object gives valuable information to the user.

The parameter estimation is based on geometrical similarities of the unknown 3D object and the objects generated by the procedural model. It is important to note that the procedural models cannot reproduce any object perfectly in full detail. We are not searching for a perfect geometric match as there might not be any parameter configuration leading to a perfect match. In fact, due to this reason we do not only use a single measure but we use 3 different measures with different degrees of detail. The panorama distance [PPTP10] as a more general measure, the surface distance as low-level distance measure and a z-buffer distance on the lowest level of detail.

Our algorithm includes 3 layers for these 3 measures. We use all measures subsequently in a hill climbing optimization to refine the estimated optimal parameters step by step. This process is illustrated in Figure 3.7.2. Additionally we measure if a layer is actually suitable for this specific object and decide whether the result of the preceding layer should be taken instead. As a result we do not only estimate the best parameters but actually are able to tell how well the procedural model represents the unknown 3D object. This means, if the procedural model can generate an object very similar on the highest layer of comparison the unknown 3D object can be represented well by the procedural model. If the unknown 3D object can only be represented on the first layer of comparison then the procedural model can only roughly represent the characteristics of the unknown 3D Object.

3.7.2. Distance Measures

Panorama Distance: The panorama distance is defined by the panorama descriptor [PPTP10], which is a hybrid descriptor based on geometrical features and image features of panoramic views of the object. We consider this descriptor to be the most high level distance of our three used distances.

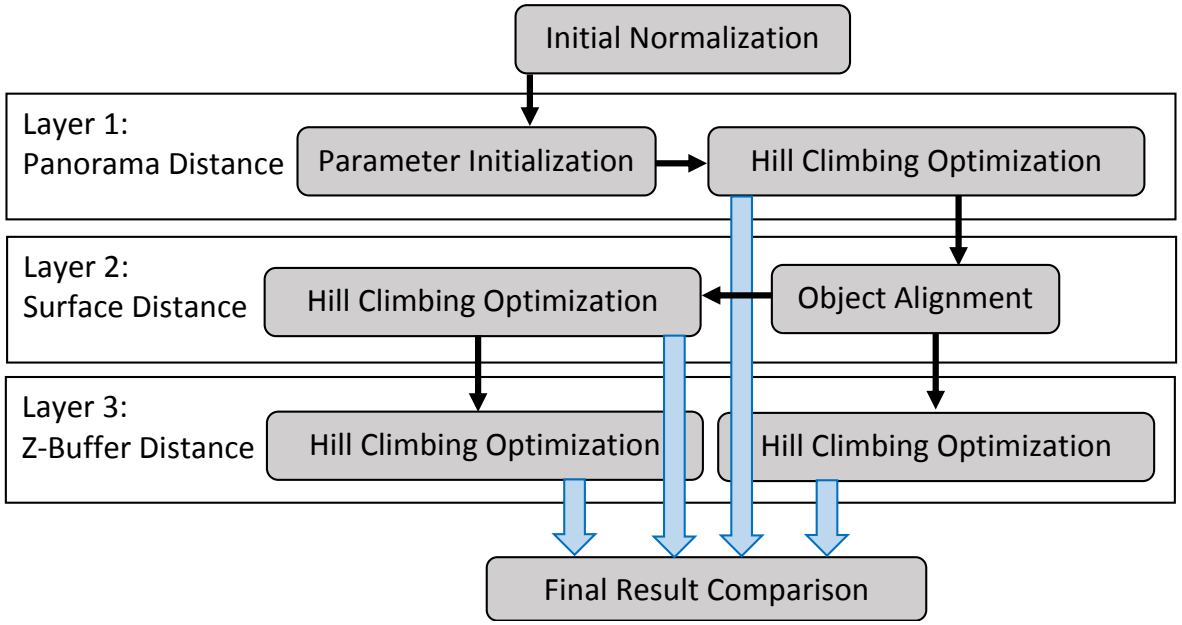


Figure 3.7.2.: The parameter estimation consists of 3 layers using 3 different levels of distance measures. The final result is the best result of 4 different optimizations.

Surface Distance: The surface distance, also known as point-to-surface-distance is based on the distance between the actual surface polygons of both meshes when aligned into a shared coordinate system. To compute the surface distance between an instance of the procedural model and the unknown object we generate a set of points for both meshes. For the unknown object we use the Poisson disk sampling [CCS12] with 2000 points. For the mesh generated by the procedural model we take all vertices of the mesh after 2 iterations of the subdivision. For each set of points the surface distance of a single point is the distance to the nearest point of the other set of points. We calculate the surface distance for every point of both sets. Then we average the distance within both sets, and finally we combine the two average distances of both sets to the final average surface distance.

Z-Buffer Distance: Our z-buffer distance is calculated on pixel level and is considered to be the most low-level distance measure. This distance also needs both objects to be aligned in one shared coordinate system. We compare the z (depth) information of both objects pixel wise. For this distance we generate a total of 14 views with 256x256 pixels. We use an orthogonal projection with $(left, right, top, bottom, near, far) = (-2, 2, -2, 2, -1.5, 1.5)$. The 14 views are the 6 views directly from the positive and negative coordinate axes and the 8 views from the corners of a cube around the origin.

We compare every view by comparing every pixel of the two images generated with the two different objects. For every pixel where both objects are present (i.e. $value \neq background$) we take the difference of the z value. If the procedural model object is present in a pixel but the unknown object not we pe-

nalize this pixel with the distance 1. This case means that the fitted object (generated by the procedural model) is oversized in comparison to the unknown object, since it fills a pixel that should not be filled. If the unknown object is present at one pixel but the procedural model object not than we double this penalty with a distance of 2. This means that the fitted object is undersized, not filling a pixel where the original object is present. We penalize this case more since we want to reward the algorithm to fill as much of the original object as possible. Therefore we reward the algorithm for growing, and penalize oversizing less than undersizing. This has a very remarkable effect. If both values are equally set to 1 the algorithm tend to get stuck in the search space with insufficiently small parts while the doubling of the undersize penalty leads to a positive reinforcement of growing into all regions of the object.

3.7.3. Algorithm for Parameter Estimation

The parameter estimation using the distances measures is performed with a greedy hill climbing algorithm. Note that these kinds of algorithms are explicitly dependent on a reasonable initialization. We ensure a good initialization of the procedural model parameters by using the panorama distance for initialization. Also, to ensure a good starting point we roughly align both objects before optimizing the surface distance and the z-buffer distance.

Our algorithm consists of 3 layers leading to results of different degree of detail for the parameter estimation. Each layer uses one of the 3 distance measures (panorama distance, surface distance, z-buffer distance).

Initial normalization: At the very start we initially bring both objects into a shared world coordinate system and normalized scale. The average position of the vertices of the mesh (center of mass) is translated to the origin. The object is scaled, so that all coordinate values are within -1 and 1.

Hill climbing algorithm: Each layer includes a hill climbing search with one of the distance measures. Each is structured in the same way: we initialize a step size with 1.0 and check all parameters. For each parameter of the procedural model we check if adding or subtracting the step size to the parameter value (clamped to the parameter range) improves the distance measure. If the distance measure is improved the parameter is changed.

In a single iteration of the algorithm all parameters are checked and changed when indicated. The algorithm stops when no parameter has been changed during an iteration. Then the step size is reduced and the algorithm is started again. The sequence of step sizes are: $1.0, 0.5, 0.25, 0.1, 0.01$. The final result is given when the algorithm ended with the smallest step size 0.01.

For the surface distance (layer 2) and the z-buffer distance (layer 3) an additional intermediate step has to be inserted: These 2 measures are directly dependent from the position of the two objects in the world coordinate system. Both objects are aligned to each other at the start of layer 2. However, when parameters of the procedural model change and the object shape changes, the position of the object is shifted respectively. Therefore after each change of a parameter value the objects have to be realigned before the distance measure is computed. Just like the parameter hill climbing algorithm itself, we perform a greedy search for the best translation, rotation and scaling. All of these 3 transformations

are checked with all possible directions and evaluated with the current used distance measure. The step size is fixed for this realignment: 0.01 for the translation, $0.01 \cdot 180$ for the rotation and 0.01 for the scaling. In special cases the scaling can lead to false intermediate optimization towards very low or big scales. To eliminate these errors we limit the scaling to a minimum of 0.5 and a maximum of 1.5 of the original scale.

3.7.4. Layer 1 – Panorama Distance

At the start of the first layer the initial normalization (scaling and translation) is performed. The initialization of the parameters of the procedural model is of major importance since the following greedy hill climbing algorithms can get stuck in a local extremum. The panorama distance generally measures the distance between two 3D objects and is optimally suited to fulfill this task. We generate a total of 10 000 objects from the procedural model with random parameterization, covering a large range of possible initialization values. We compute the panorama distance of each generated object to the unknown object. The parameters of the generated object with the smallest panorama distance are taken as our starting point.

Subsequently, we perform a hill climbing optimization of all parameters using the panorama distance. The result of this optimization is the final result of layer 1.

3.7.5. Layer 2 – Surface Distance

For the surface distance measure and the following z-buffer distance measure we need to align both objects with each other in the world coordinate system. To ensure an optimal initialization of the rotation we perform a brute-force search of 24 possible coordinate system rotations. The 24 rotations include all main rotation possibilities: the x-axis can be rotated to match one of the 6 possible positive or negative coordinate system axis and can be rotated around itself by $0, 90, 180$ or 270 degrees. Giving a total of $6 \cdot 4 = 24$ possibilities. For each of the 24 possibilities we perform the initial rotation and then further optimize the translation, rotation and scaling. Just like in the parameter hill climbing algorithm itself we adjust the transformation with a fixed step size of 0.01 until an optimum is reached. Finally, each of the 24 possibilities are evaluated with the surface distance and the case with the lowest surface distance is taken as the initial alignment.

After the alignment has been computed, we perform a hill climbing optimization of all parameters using the surface distance. The final result of layer 2 gives the optimal parameters with respect to the surface distance.

3.7.6. Layer 3 – Z-Buffer Distance

In the final layer we use the most low-level distance measure based on pixel wise z-buffer differences. In this layer we compute a hill climbing optimization of all parameters using the z-buffer distance. We compute this optimization for 2 cases: In the first case we use the output of layer 2 as input and in the second case we use the output of layer 1 (after the alignment in layer 2) as input. Hence, we compute optimal parameters for 2 different starting point. Then we compare the z-buffer distance of the two final optimization results and take the better solution as final result.

At the end of our parameter estimation we decide which layer result is the most adequate representation. As mentioned already, the procedural model might not be able to represent every object to a pixel wise degree, hence we set thresholds for the final results to decide to which extent the procedural model represents the unknown object. We analyzed several objects, results and distance measures values and identified shared thresholds for the distance measures. A z-buffer difference of lower than 0.7 and a surface distance of lower than 0.04 represents an adequate match. The final parameters correspond to the result of layer 3 if the z-buffer difference is below 0.7. Else it corresponds to the result of layer 2 if the surface distance is below 0.04. If both are not the case than the result of layer 1 gives the final parameters.

3.7.7. Evaluation and Discussion

For the evaluation we used the 10 procedural models classified by our deep learning approach in Section 3.6.

We take all correctly classified examples of our 10 classes and estimated the parameters of the procedural model for every unknown object. Figure 3.7.3 shows the distribution of the surface distance and z-buffer distance for the 4 different results in the 3 layers: the result of layer 1 (using the panorama distance), the results of layer 2 (using the surface distance) and both results of layer 3 (using the z-buffer distance with the output of layer 1 or layer 2). Figure 3.7.4 shows which final results were taken from which layers. Figure 3.7.5 presents several exemplary parameter estimations for all classes. The full parameter estimation of an unknown object took about 3 minutes.

Discussion: The results generally show that for the majority of objects fitting parameters could be found on the last layer. However, the plots also show that about one third of the objects could not achieve desirable precision at z-buffer level. Also about two tenth could neither achieve desirable precision at z-buffer level or surface distance level, so that the final result origins from layer 1. For this cases we proposed our back propagation system to previous layers. If the procedural model is not able to precisely represent the object the results of the higher layers can be misleading. Therefore, our approach recognizes these cases and provides a correct, but more general result from a previous layer.

In the plots of Figure 3.7.3 we can also detect another advantage of the layered optimization system. In the plots we present the two different results from layer 3 separately. Here, we can see that the distribution of the z-buffer distance in the final layer is better on average when the output of the 2nd

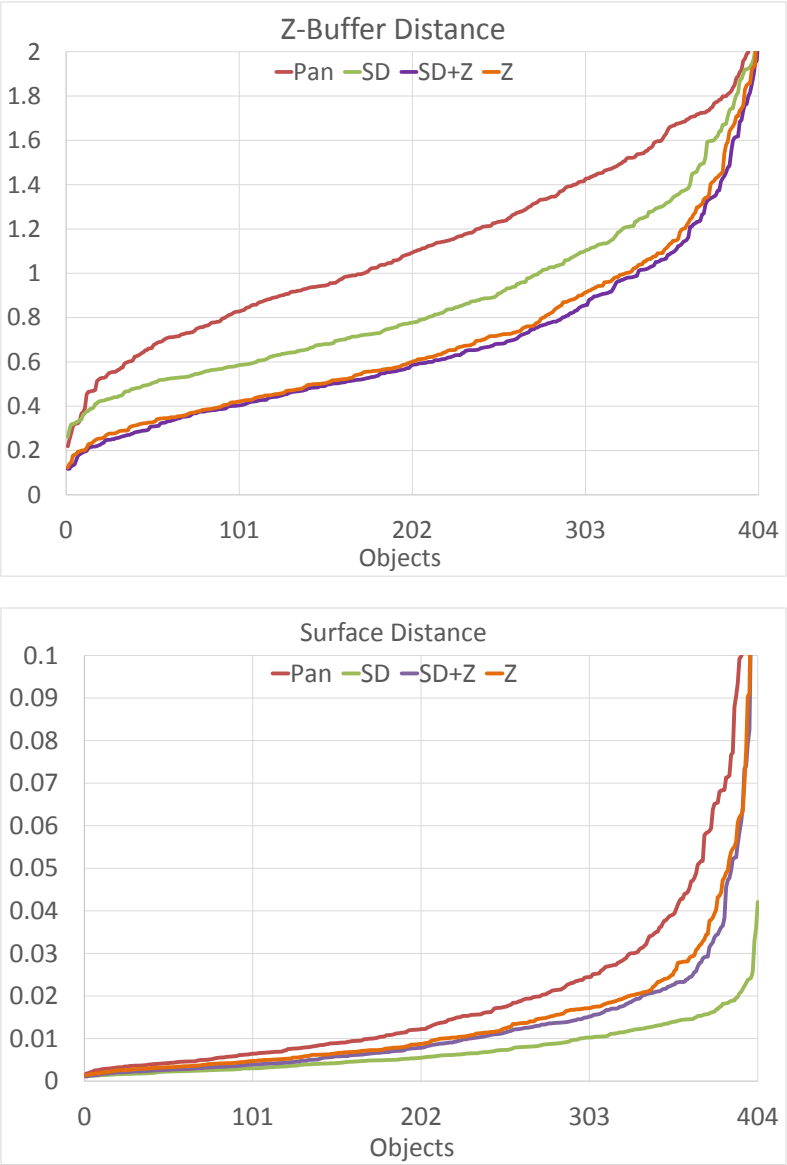


Figure 3.7.3.: The plots show the distribution of the surface distance and the Z-Buffer distance for all objects for the 4 different results from the 3 layers.

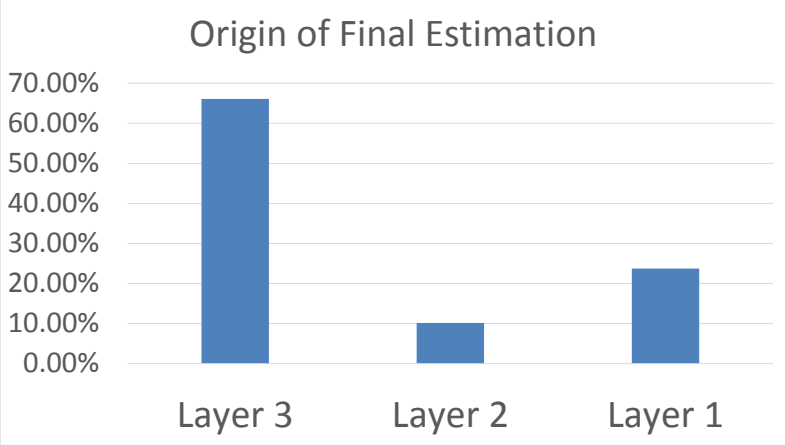


Figure 3.7.4.: In the final step the best result from all layers is taken as final result. This graph shows the origin of the final result.

layer is taken as input. As the hill climbing algorithm naturally profits from a good initialization, we can see that not only the initial setup of layer 1 improves the results, but also the intermediate optimization of layer 2 improves the results of the final layer 3.

The examples presented in Figure 3.7.5 show that the parameter estimations lead to generated objects with similar overall appearance compared to the unknown database objects. The examples also show that most object classes could be estimated on layer 3 (z-buffer). However, the bicycle, spider and helicopter class did not have enough flexibility to represent most of the objects on layer 3. Especially the rotors of the helicopter, the legs of the spiders and the thin spokes and connection bars of the bicycle could not be matched pixel-wise. More parameters could be added to increase the flexibility. Nevertheless, the results from layer 2 (surface distance) and layer 1 (panorama distance) are sufficient in most cases.

Figure 3.7.6 presents an object characteristic derived by the parameters. Here we order the objects by the ratio of the stem length to the bowl length. The parameters describe the main object characteristics quite well. Important to note in this context is that ratios and differences between parameters are more meaningful than single parameters. This is the case because all objects have to be normalized in the coordinate system and the fitting of the parameters includes a scaling of the whole objects. Therefore the size of the objects can differ and the length values of the parameters are not completely comparable. This effect is only present because the database objects are not related to real length values. This is not the case if we consider a scanned object. Here we have a real connection of values to millimeters. In this case the parameters can depict real millimeters and are more comparable individually.

Figure 3.7.7 shows two types of errors we found in the results. The bird is quite symmetrical, so that the objects happen to be misaligned in layer 2. The head and the tail are facing in the wrong direction.

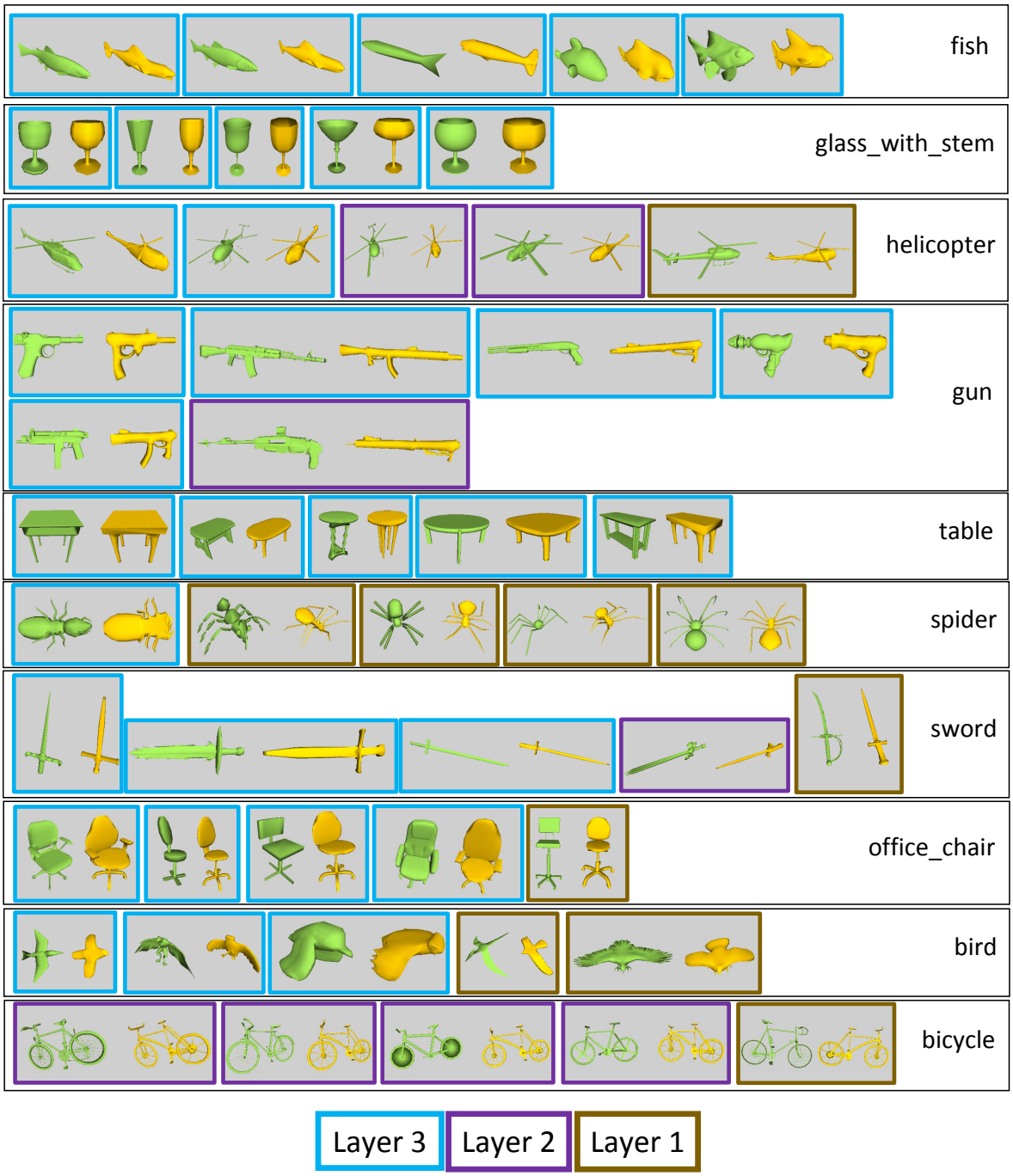


Figure 3.7.5.: Exemplary results for all classes. The colored borders show from which layer the result origins.

Figure 3.7.6.: Ten different glasses of the database sorted by the ratio of stem length to the bowl length.

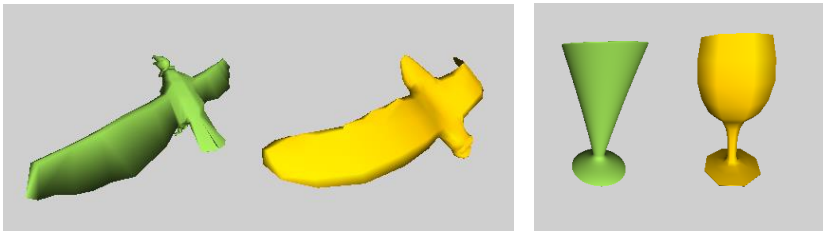


Figure 3.7.7.: Limitations of the parameter estimation: The bird is falsely aligned in layer 2. The glass has an estimation of the stem length even though the glass of the database has no stem.

These cases happened at some highly symmetrical objects of the bird, fish and gun class. In the future we will have to integrate an additional symmetry detection to handle these cases.

The second error type is represented by the glasses with stem. The database object does not have a real stem in Figure 3.7.7. The bowl is directly connected to the base. The procedural model does not include the case of a stem having 0 length. Even though this result comes from layer 3, the final parameters are distorted by the falsely estimated stem length. In general, the procedural model represents a blueprint of an object class and the parameters describe the basic characteristics. However, exceptional objects can still lead to false parameter estimations in layer 3. On average, exceptional objects tend to be shifted towards higher layers, making estimations from layer 3 more reliable.

Summarizing, our parameter estimation could not only achieve a high accuracy for the majority of objects but it has also shown to deliver viable additional information in the cases where the procedural model does not fit perfectly to the unknown object. Therefore, the classification and parameter estimation with procedural models excels in the task of information gathering for unknown object. The quality only depends on the quality of the procedural model itself. Hence, the user can choose the right level of control and automation for his task and thereby also control the level of quality in the subsequent 3D object retrieval, classification or parameter estimation.

3.8. Conclusion – Combination of 3DOR and Classification with Procedural Models

In this part, we introduced several approaches using procedural models for 3D object retrieval and classification. Parametric procedural models have a high potential for these two applications, since both profit from a more precise definition of the query or class.

A query in 3DOR is often underdefined when the query is represented by a single example object. The user wants to express what kind of object he is searching for and chooses an example within the range of objects he has in mind. With a procedural model a whole range of objects can be defined. Therefore, the query is defined more precisely.

We have developed an approach to directly integrate a procedural model into a 3D object retrieval application. We defined a method to use a descriptor for the direct measuring of the distance between a 3D object and a procedural model. On basis of this measurement method we could retrieve 3D objects from a database ordered by similarity to the procedural model. The integration of the procedural model works seamless and the quality of the results is very good. Note that the used descriptor is not specifically developed for procedural models. Therefore, good results could be achieved with a normal descriptor and a direct integration of procedural models. We reason, that a well defined procedural model is perfectly suitable for the integration in this application.

In 3D object classification many examples of a class of objects are needed to learn this class adequately. The available databases cover common objects like cats or dogs and are intended for evaluating new approaches. In a real scenario, the construction of a complete suitable object database for class description is effortful and expensive. The procedural model is able to directly serve as data basis for a whole object class definition.

We proposed to use procedural models as a data basis for deep learning. Here we have shown that procedural models are suitable for this task. The network was able to learn a new class only using variations provided by the procedural model. Note that this is not trivial, since the network does not generally profit from additional variation. For example, we tested the possibility of including further variations into the data by including random translations, however the results were highly inferior. Therefore, we can conclude that the object variations included by using the procedural model are directly beneficial for the deep learning task.

Furthermore, we proposed a new approach for automatic parameter estimation of a procedural model given an unknown object. This can be understood as an extension to the classification task. The class label is extended by additional quantified characteristics to improve the gained information. Our

evaluation shows that the additional information is highly beneficial to categorize and order unknown objects. Therefore, the procedural model is able to seamlessly enhance the basic task of classification.

Overall, the combination of 3DOR and classification with procedural models works perfectly well. In contrast to any other approach procedural models represent persistent additional knowledge, which can be reused, recombined and edited. Therefore, procedural models are not only perfect in terms of the quality of the results but even furthermore represent a superior persistent formulation of additional knowledge.

Part 4.

Conclusion and Future Work

4.1. Conclusion

In this thesis I proposed a generalization and automation of the creation and parameterization of procedural models to directly use them for 3D object retrieval and 3D object classification.

4.1.1. Generalization and Automation of Procedural Models

For the generalization of procedural models I have introduced several possibilities of using modeling procedures for the representation of the construction process of a 3D object. I have proposed an approach where the procedural model directly stems from a modeling tool translating modeling procedures to parametric procedures of a procedural model. In a second approach the modeling procedures are used to reversely derive a complete construction process from a single exemplary 3D object.

The automation of the generation of procedural models is always a trade-off. Higher automation leads to less specific control for the user. For this reason, i have proposed several levels of automation, which can be combined with each other. On the highest level a procedural models is generated automatically based on a single exemplary 3D object. On a lower level, a semi-automatic approach using a sketch-based modeling tool gives more control to the user. For the highest control the user can still define his own procedures and manually edit or construct the procedural model.

Besides the initial creation of the procedures a major part of parametric procedural models is the insertion of parameters. The generalized procedures are targeted towards inserting parameters to alter the construction process and therefore vary the resulting shape of the 3D object. Therefore, I also proposed several possibilities of inserting parameters into generalized procedural models with different levels of automation and control.

It is possible to automatically generate basic parameters varying the length, size and direction of the procedures. The approach automatically measures the importance of the parameters and offers the user the possibility of automatically including all important parameters or to inspect the parameters ordered by importance and only choose the parameters fitting to the semantic target of the user.

Complex parameters can be inserted by using a deformation based approach. The user can define the effect of the parameter by deforming the default mesh of the procedural model. The deformation is automatically translated to a parameter. For the sketch-based modeling approach the user can pick one of three insertion schemes in an insertion interface to manually control the insertion and extent of each parameter. Lastly, it is also possible to manually insert complex parameters into the procedures.

On the highest level of automation various types of objects can be used for the automatic generation and parameterization of procedural models. However, the automatic generation directly uses an

automatic skeletonization. Therefore, the major limitation of our approach concerns bulky objects with unclear skeletons, objects with complex topologies combined with thick structures, and objects with overlapping polygons, resulting in erroneous skeletons. An imprecise topology of the skeleton results in an imprecise procedural model. Therefore, right now the procedural model cannot be automatically generated for every possible type of object. However, it is always possible to use one of the less automated approaches to achieve sufficient control.

4.1.2. 3D Object Retrieval and Classification using Procedural Models

For the tasks of 3D object retrieval and classification I analyzed several possibilities of using descriptors and learning with procedural models.

These tasks always include a comparison measure for 3D objects. For the direct local comparison of similar meshes I propose a new distance measure: the extended surface distance. For the task of 3D object retrieval I propose a parameter-free hierarchical clustering. The hierarchical clustering can be used with procedural models but it also improves the results of any other unsupervised 3D object retrieval task.

For the usage of procedural models in the 3D object classification task I used a deep learning approach. The parameters of the procedural model are used to create variations of all objects. All variations of all procedural models are then used as data basis for a retraining of a deep neural network. The results confirm that the variations are learned by the network and directly improve the classification accuracy.

For the 3D object retrieval task I have shown the result enhancement with two different approaches: First, using a 3D descriptor in combination with the variation of a procedural model to measures the distance to all objects of a database and increase the accuracy in the 3D object retrieval task. Second, deep-learning classes by using several procedural models in advance and then translating the class probabilities to object distances to retrieve an improved 3D object retrieval list.

Summarizing, I have shown that it is possible to automatically generate generalized procedural models and effectively enhance 3D object retrieval and classification. The only drawback of the proposed concept in comparison to the completely unsupervised method is that the procedural models have to be generated in advance. The results have proven that adequate procedural models improve the accuracy of these tasks. The only approaches with comparable results are deep learning approaches trained on very large specialized training databases. However, a core advantage of the procedural model approach is that objects can directly be deep-learned with the generated procedural models. Therefore, I additionally avoid the need of large suitable training databases. These are arguably rarely available and costly to create in real scenarios. Furthermore, using the proposed parameter estimation technique based on procedural models, it was possible to directly estimate characteristic parameters of an unknown object. Therefore, procedural model are perfectly suitable for complex retrieval and classification tasks and are even more suitable for the representation and for the estimation of parameters of 3D objects.

4.2. Future Work

The thesis has shown that a generalization and automation of procedural models leads to an enhancement of 3D object retrieval and classification tasks. However, there are still limitations which should be tackled in the future. Especially the generalization as well as the automation should be increased even further to establish procedural models for all cases of the tasks. Also the deep learning of procedural models should be promoted to the next level of concept by creating procedural model databases. Furthermore, to ease the usage of procedural models in all tasks additional model space exploration and visualization techniques are needed.

4.2.1. Optimal Generalized set of Procedures for Special Cases

The generalization of procedural models does not work in special cases, where the generalized procedures are not adequate. Especially for objects with erroneous and imprecise skeletons, e.g. bulky objects with unclear skeletons or object with bulky parts combined with a complex topology. In order to increase the generalization of procedural models to cover these cases more adequate procedures have to be integrated into the set of modeling procedures. The generalized procedural models should be limited to a general set of procedures, but also the set of procedures should be able to cover all cases. Finding the complete optimal set of procedures, covering all special cases is the most important task for the future.

4.2.2. Automatic Suggestion System for Parameters

The automation of the generation of procedural models is an important step towards broader applications like 3D object retrieval. As we have seen we naturally partly trade control for automation. Nevertheless, further improvements of this trade-off should be accomplished to increase the usability for all possible tasks. Especially one step can profit from further researching the automation possibilities: The insertion of parameters into the procedural model. While basic parameters can be included automatically, more complex parameters include user-interactions. It is generally not possible to completely automate this step, as the choice of parameters corresponds to the definition of possible variations, which in fact is a semantic decision of the user. A further automation of the parameterization is the biggest possible improvement in the future of the concept. The generation of complex parameter describing the semantic variation of an object class as desired by the user, would lead to the next level of automation, opening up the concept even for casual users. A promising approach for

this step are suggestion systems. In a suggestion system the user interaction is limited to a minimum. The system offers the user several possibilities of variations and the user only confirms or declines the suitable variations. The system automatically learns what kind of variations the user wants to have and automatically generates suitable parameters for a procedural model.

4.2.3. Deep Learning of Combinations of Procedural Models

For the deep learning of procedural models, variations of several models are used in the learning process. The neural network is retrained with every new set of procedural models. However, to bring the concept of learning procedural models for classification and retrieval to a new level, a database of combinations of procedural models should be targeted. In this database procedural models of combined types could be included to further increase the classification possibilities. Furthermore, the databases could offer parts of procedural models for reusing and recombining with new models to even increase the flexibility and usability for the user. The user can create new procedural models by recombining parts or allow variation and substitution of parts during the learning process to detect different types of combinations within a class.

4.2.4. Procedural Model Visualization and Variation Space Exploration

Procedural models generally have proven to be a flexible and powerful concept in several domains. However, the complete variation possibilities of a large procedural model with multiple parameters can be hard to grasp for a new user. The parameters span a multi-dimensional space of possible objects. Showing several instances of the procedural model to represent the model is helpful, but it does not depict the full space. To make procedural models more approachable for all tasks and users, an important key consists of easing the access with a novel visualization and variation space exploration technique tailored towards procedural models of 3D objects. An elaborated way of exploring and visualizing the variation possibilities of a procedural model could also be used for the insertion of parameters to provide the user the full information about the current variation space for further parameter insertions.

Bibliography

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3 (Mar. 2006), 181–193. 24

[AKZM14] AVERKIOU M., KIM V. G., ZHENG Y., MITRA N. J.: Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 125–134. 23, 25, 112, 113

[ASCE02] ASPERT N., SANTA-CRUZ D., EBRAHIMI T.: Mesh: Measuring errors between surfaces using the hausdorff distance. In *Multimedia and Expo - ICME* (2002), vol. 1, IEEE, pp. 705–708. 109, 120

[ASYS09] AKGÜL C. B., SANKUR B., YEMEZ Y., SCHMITT F.: 3D model retrieval using probability density-based shape descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 31*, 6 (2009), 1117–1133. 110

[ASYS10] AKGÜL C. B., SANKUR B., YEMEZ Y., SCHMITT F.: Similarity Learning for 3D Object Retrieval Using Relevance Feedback and Risk Minimization. *International Journal of Computer Vision 89*, 2-3 (Sept. 2010), 392–407. 5, 111

[BAS14] BÆRENTZEN J. A., ABDRASHITOV R., SINGH K.: Interactive shape modeling using a skeleton-mesh co-representation. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 132. 24

[BBCW10] BERNHARDT A., BARTHE L., CANI M. P., WYVILL B.: Implicit blending revisited. *Computer Graphics Forum 29*, 2 (2010), 367–375. 24

[BBGO11] BRONSTEIN A. M., BRONSTEIN M. M., GUIBAS L. J., OVSJANIKOV M.: Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics (TOG) 30*, 1 (2011), 1. 110

[BFH05] BERNDT R., FELLNER D. W., HAVEMANN S.: Generative 3d models: a key to more information within less bandwidth at higher quality. In *Proceedings of the tenth international conference on 3D Web technology* (2005), ACM, pp. 111–121. 23, 28

[BGM*07] BIASOTTI S., GIORGI D., MARINI S., SPAGNUOLO M., FALCIDIENO B.: 3d classification via structural prototypes. In *Semantic Multimedia*. Springer, 2007, pp. 140–143. 112, 113

[BHSF09] BEIN M., HAVEMANN S., STORK A., FELLNER D. W.: Sketching subdivision surfaces. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2009), ACM, pp. 61–68. 25, 31, 32, 40, 55

[BK04] BOTSCH M., KOBBELT L.: A Remeshing Approach to Multiresolution Modeling. *Proceedings of the 2004 Eurographics ACM SIGGRAPH symposium on Geometry processing SGP 04* (2004), 185. 50

[BKS14] BECKER M., KIRSCHNER M., SAKAS G.: Segmentation of risk structures for otologic surgery using the probabilistic active shape model (pasm). In *SPIE Medical Imaging* (2014), International Society for Optics and Photonics, pp. 90360O–90360O. 126

[BMP01] BELONGIE S., MALIK J., PUZICHA J.: Matching shapes. In *Computer Vision - ICCV* (2001), vol. 1, IEEE, pp. 454–461. 110

[BR07] BROWN B. J., RUSINKIEWICZ S.: Global non-rigid alignment of 3-d scans. In *ACM T. Graphics (TOG)* (2007), vol. 26, ACM, p. 21. 110

[BSMM11] BENES B., STAVA O., MĚCH R., MILLER G.: Guided Procedural Modeling. *Computer Graphics Forum 30*, 2 (Apr. 2011), 325–334. 6, 21

[BVS16] BESSMELTSEV M., VINING N., SHEFFER A.: Gesture3D: Posing 3D Characters via Gesture Drawings. *ACM Transactions on Graphics 35*, 6 (2016), 165:1–165:13. 26

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 104. 22

[BWSK12] BOKELOH M., WAND M., SEIDEL H.-P., KOLTUN V.: An algebraic model for parameterized shape editing. *ACM Transactions on Graphics 31*, 4 (July 2012), 1–10. 5, 22

[CCS12] CORSINI M., CIGNONI P., SCOPIGNO R.: Efficient and flexible sampling with blue noise properties of triangular meshes. *IEEE Transactions on Visualization and Computer Graphics 18*, 6 (2012), 914–924. 88, 173

[CiRL*16] CHOI B., I RIBERA R. B., LEWIS J. P., SEOL Y., HONG S., EOM H., JUNG S., NOH J.: SketchiMo: Sketch-based Motion Editing for Articulated Characters. *ACM Transactions on Graphics 35*, 4 (jul 2016), 1–12. 26

[CK97] CHALANA V., KIM Y.: A methodology for evaluation of boundary detection algorithms on medical images. *IEEE T. Medical Imaging 16*, 5 (1997), 642–652. 109

[CMF*06] CATES J., MEYER M., FLETCHER T., WHITAKER R., ET AL.: Entropy-based particle systems for shape correspondence. In *1st MICCAI Workshop on Mathematical Foundations of Computational Anatomy: Geometrical, Statistical and Registration Methods for Modeling Biological Shape Variability* (2006), pp. 90–99. 110

[CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. In *ACM transactions on graphics (TOG)* (2010), vol. 29, ACM, p. 38. 99

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM T. Graphics 23*, 3 (2004), 905–914. 109

[CTSO03] CHEN D.-Y., TIAN X.-P., SHEN Y.-T., OUHYOUNG M.: On visual similarity based 3D model retrieval. In *Computer graphics forum* (2003), vol. 22, Wiley Online Library, pp. 223–232. 110, 143

[DK12] DAROM T., KELLER Y.: Scale-invariant features for 3-d mesh models. *IEEE Transactions on Image Processing 21*, 5 (2012), 2758–2769. 110

[DS15] DE PAOLI C., SINGH K.: SecondSkin: Sketch-based Construction of Layered 3D Models. *Acm Transactions on Graphics 34*, 4 (2015), 10. 50

[DSY10] DUTAGACI H., SANKUR B., YEMEZ Y.: Subspace methods for retrieval of general 3D models. *Computer Vision and Image Understanding 114*, 8 (Aug. 2010), 865–886. 112

[EHBA10] EITZ M., HILDEBRAND K., BOUBEKEUR T., ALEXA M.: Sketch-based 3D shape retrieval. In *ACM SIGGRAPH 2010 Talks on - SIGGRAPH '10* (New York, New York, USA, 2010), ACM Press, p. 1. 25

[ERB*12] EITZ M., RICHTER R., BOUBEKEUR T., HILDEBRAND K., ALEXA M.: Sketch-based shape retrieval. *ACM Transactions on Graphics 31*, 4 (2012), 1–10. 25

[FAVK*14] FISH N., AVERKIOU M., VAN KAICK O., SORKINE-HORNUNG O., COHEN-OR D., MITRA N. J.: Meta-representation of shape families. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 34. 25

[FGLW08] FANG R., GODIL A., LI X., WAGAN A.: A new shape benchmark for 3D object retrieval. *Advances in Visual Computing* (2008), 381–392. 143, 155, 165

[FMK*03] FUNKHOUSER T., MIN P., KAZHDAN M., CHEN J., HALDERMAN A., DOBKIN D., JACOBS D.: A search engine for 3d models. *ACM Transactions on Graphics (TOG) 22*, 1 (2003), 83–105. 112, 113

[FRS*12] FISHER M., RITCHIE D., SAVVA M., FUNKHOUSER T., HANRAHAN P.: Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 135. 22

[GBP07] GIORGI D., BIASOTTI S., PARABOSCHI L.: Shape retrieval contest 2007: Watertight models track. *SHREC competition 8* (2007). 143

[GF15] GETTO R., FELLNER D. W.: 3D Object Retrieval with Parametric Templates. In *Proceedings of the Eurographics Workshop on 3D Object Retrieval* (2015), Eurographics Association, pp. 47–54. vii, 10, 11, 13, 14, 15

[GFJ*18] GETTO R., FINA K., JARMS L., KUIJPER A., FELLNER D. W.: 3D Object Classification and Parameter Estimation based on Parametric Procedural Models. In *26th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (2018). vii, 11, 13, 14, 15

[GKF17] GETTO R., KUIJPER A., FELLNER D. W.: Unsupervised 3D Object Retrieval with Parameter-Free Hierarchical Clustering. In *Proceedings of the Computer Graphics International Conference* (2017), no. 7, ACM. vii, 15

[GKF18] GETTO R., KUIJPER A., FELLNER D. W.: Automatic Procedural Model Generation for 3D Object Variation. *The Visual Computer* (2018), 1–18. vii, 10, 13, 14, 15

[GKvL15] GETTO R., KUIJPER A., VON LANDESBERGER T.: Extended Surface Distance for Local Evaluation of 3D Medical Image Segmentations. *The Visual Computer 31*, 6-8 (2015), 989–999. vii, 15

[GL06] GOSSWEILER R., LIMBER M.: Sketchup: An easy-to-use 3d design tool that integrates with google earth. In *Adjunct Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology (UIST06)* (2006), vol. 19, p. 3. 3, 25

[GLXJ14] GUO X., LIN J., XU K., JIN X.: Creature grammar for creative modeling of 3d monsters. *Graphical Models 76*, 5 (2014), 376–389. 5, 22

[GMKF17] GETTO R., MERZ J., KUIJPER A., FELLNER D. W.: 3D Meta Model Generation with Application in 3D Object Retrieval. In *Proceedings of the Computer Graphics International Conference* (2017), no. 6, ACM. vii, 10, 11, 13, 14, 15

[Gue01] GUEZIEC A.: Meshsweeper: dynamic point-to-polygonal mesh distance and applications. *IEEE T. Visualization and Computer Graphics 7*, 1 (2001), 47–61. 109

[GWJ*14] GAO Y., WANG M., JI R., WU X., DAI Q.: 3-d object retrieval with hausdorff distance learning. *Industrial Electronics, IEEE Transactions on 61*, 4 (2014), 2088–2098. 111

[GXLT09] GONG B., XU C., LIU J., TANG X.: Boosting 3D object retrieval by object flexibility. In *Proceedings of the 17th ACM international conference on Multimedia* (2009), ACM, pp. 525–528. 112

[HF01] HAVEMANN S., FELLNER D. W.: A versatile 3d model representation for cultural reconstruction. In *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage* (2001), ACM, pp. 205–212. 31, 32

[HF03] HAVEMANN S., FELLNER D. W.: Technical Report TUBS-CG-2003-01 Generative Mesh Modeling. *Online* (2003). 6

[HF04] HAVEMANN S., FELLNER D. W.: Generative parametric design of Gothic window tracery. In *Shape Modeling Applications, Proceedings* (2004), IEEE, pp. 350–353. 6, 23

[HF05] HAVEMANN S., FELLNER D. W.: *Generative mesh modeling.* PhD thesis, University of Braunschweig-Institute of Technology, 2005. 6, 21, 23, 28

[HLR05] HOU S., LOU K., RAMANI K.: SVM-based semantic clustering and retrieval of a 3D model database. *Computer-Aided Design and Applications 2*, 1-4 (2005), 155–164. 112, 114, 137

[HMVG09] HAEGLER S., MÜLLER P., VAN GOOL L.: Procedural modeling for digital cultural heritage. *Journal on Image and Video Processing 2009* (2009), 7. 22

[Hop96] HOPPE H.: Progressive meshes. In *Computer graphics and interactive techniques* (1996), ACM, pp. 99–108. 109

[HRL75] HERMAN G. T., ROZENBERG G., LINDENMAYER A.: *Developmental systems and languages*. North-Holland Pub. Co., 1975. 21

[HTB03] HAEHNEL D., THRUN S., BURGARD W.: An extension of the icp algorithm for modeling nonrigid objects with mobile robots. In *IJCAI* (2003), pp. 915–920. 110

[HvGS*09] HEIMANN T., VAN GINNEKEN B., STYNER M. A., ARZHAEVA Y., AURICH V., BAUER C., BECK A., BECKER C., BEICHEL R., BEKES G., ET AL.: Comparison and evaluation of methods for liver segmentation from ct datasets. *IEEE T. Medical Imaging 28*, 8 (2009), 1251–1265. 109

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3D freeform design. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99* (1999), 409–416. 25

[IRSS03] IP C. Y., REGLI W. C., SIEGER L., SHOKOUFANDEH A.: Automated learning of model classifications. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), ACM, pp. 322–327. 113

[JKIR06] JAYANTI S., KALYANARAMAN Y., IYER N., RAMANI K.: Developing an engineering shape benchmark for CAD models. *Computer-Aided Design 38*, 9 (Sept. 2006), 939–953. 143

[Joh97] JOHNSON A. E.: *Spin-images: a representation for 3-D surface matching*. PhD thesis, Carnegie Mellon University, 1997. 110

[JS11] JORGE J., SAMAVATI F. (Eds.): *Sketch-based Interfaces and Modeling*. Springer London, London, 2011. 26

[JTRS12] JAIN A., THORMÄHLEN T., RITSCHEL T., SEIDEL H.-P.: Exploring Shape Variations by 3d-Model Decomposition and Part-based Recombination. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 631–640. 5, 23, 98, 112

[KBW11] KIRSCHNER M., BECKER M., WESARG S.: 3D active shape model segmentation with nonlinear shape priors. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI, LNCS 6892* (2011), pp. 492–499. 126

[KFR03] KAZHDAN M., FUNKHOUSER T., RUSINKIEWICZ S.: Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Symposium on geometry processing* (2003), vol. 6. 110

[KLM*13] KIM V. G., LI W., MITRA N. J., CHAUDHURI S., DIVERDI S., FUNKHOUSER T.: Learning part-based templates from large collections of 3D shapes. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 70. 25

[KPW*10] KNOPP J., PRASAD M., WILLEMS G., TIMOFTE R., VAN GOOL L.: Hough transform and 3d surf for robust three dimensional classification. In *Computer Vision–ECCV 2010*. Springer, 2010, pp. 589–602. 110

[KS04] KRAEVOY V., SHEFFER A.: Cross-parameterization and compatible remeshing of 3D models. In *ACM Transactions on Graphics (TOG)* (2004), vol. 23, ACM, pp. 861–869.

23, 110

[KYZ14] KAZMI I. K., YOU L., ZHANG J. J.: A Survey of Sketch Based Modeling Systems. *2014 11th International Conference on Computer Graphics, Imaging and Visualization* (2014), 27–36. 25

[LH07] LI H., HARTLEY R.: The 3D-3D registration problem revisited. In *Computer Vision - ICCV* (2007), IEEE, pp. 1–8. 116

[Lin68] LINDENMAYER A.: Mathematical models for cellular interactions in development. II. Simple and branching filaments with two-sided inputs. *Journal of theoretical biology 18*, 3 (1968), 300–315. 21

[LJ13] LI B., JOHAN H.: 3D model retrieval using hybrid features and class information. *Multimedia tools and applications 62*, 3 (2013), 821–846. 113

[LLL*14] LI B., LU Y., LI C., GODIL A., SCHRECK T., AONO M., CHEN Q., CHOWDHURY N. K., FANG B., FURUYA T., JOHAN H., KOSAKA R., KOYANAGI H., OHBUCHI R., TATSUMA A.: SHREC' 14 Track: Large Scale Comprehensive 3D Shape Retrieval. In *Eurographics Workshop on 3D Object Retrieval 2014 (3DOR 2014)* (2014), pp. 131–140. 143

[LLL*15] LI B., LU Y., LI C., GODIL A., SCHRECK T., AONO M., BURTSCHER M., CHEN Q., CHOWDHURY N. K., FANG B., ET AL.: A comparison of 3d shape retrieval methods based on a large-scale benchmark supporting multimodal queries. *Computer Vision and Image Understanding 131* (2015), 1–27. 111

[LMT05] LEIFMAN G., MEIR R., TAL A.: Semantic-oriented 3D shape retrieval using relevance feedback. *The Visual Computer 21*, 8-10 (2005), 865–875. 5, 111

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: Treesketch: interactive procedural modeling of trees on a tablet. In *Proceedings of the international symposium on sketch-based interfaces and modeling* (2012), Eurographics Association, pp. 107–120. 6, 21

[LRF10] LIPMAN Y., RUSTAMOV R. M., FUNKHOUSER T. A.: Biharmonic distance. *ACM T. Graphics 29*, 3 (2010), 27. 110

[LRS10] LIAN Z., ROSIN P. L., SUN X.: Rectilinearity of 3D Meshes. *International Journal of Computer Vision 89*, 2-3 (Sept. 2010), 130–151. 112

[LSP08] LI H., SUMNER R. W., PAULY M.: Global correspondence optimization for non-rigid registration of depth scans. In *Computer graphics forum* (2008), vol. 27, Wiley, pp. 1421–1430. 110

[LZYX15] LENG B., ZENG J., YAO M., XIONG Z.: 3D Object Retrieval With Multitopic Model Combining Relevance Feedback and LDA Model. *IEEE Transactions on Image Processing 24*, 1 (Jan. 2015), 94–105. 5, 111

[MFK*10] MAES C., FABRY T., KEUSTERMANS J., SMEETS D., SUETENS P., VANDERMEULEN D.: Feature detection on 3d face surfaces for pose normalisation and recog-

nition. In *Biometrics: Theory Applications and Systems (BTAS), 2010 Fourth IEEE International Conference on* (2010), IEEE, pp. 1–6. 110

[MGG15] MARTINEK M., GROSSO R., GREINER G.: Interactive partial 3d shape matching with geometric distance optimization. *The Visual Computer 31*, 2 (2015), 223–233. 110

[MGKF18] MERZ J., GETTO R., KUIJPER A., FELLNER D. W.: Simplified Definition of Parameter Spaces of a Procedural Model using Sketch-Based Interaction. In *Proceedings of the 13th International Conference on Computer Graphics Theory and Applications* (2018). vii, 10, 13, 14, 15

[MPB05] MARVIE J.-E., PERRET J., BOUATOUCH K.: The FL-system: a functional L-system for procedural geometric modeling. *The Visual Computer 21*, 5 (June 2005), 329–339. 22

[MS15] MATURANA D., SCHERER S.: Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on* (2015), IEEE, pp. 922–928. 114, 161

[MSF07] MARINI S., SPAGNUOLO M., FALCIDIENO B.: Structural shape prototypes for the automatic classification of 3d objects. *IEEE Computer Graphics and Applications 27*, 4 (2007), 28–37. 110

[MSH*08] MENDEZ E., SCHALL G., HAVEMANN S., FELLNER D. W., SCHMALSTIEG D., JUNGHANNS S.: Generating Semantic 3D Models of Underground Infrastructure. *IEEE Computer Graphics and Applications 28*, 3 (May 2008), 48–57. 6, 23

[MWCS13] MILLIEZ A., WAND M., CANI M.-P., SEIDEL H.-P.: Mutable elastic models for sculpting structured shapes. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 21–30. 22

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Transactions On Graphics (TOG) 25*, 3 (2006), 614–623. 22

[MWZ*13] MITRA N., WAND M., ZHANG H. R., COHEN-OR D., KIM V., HUANG Q.-X.: Structure-aware shape processing. In *SIGGRAPH Asia 2013 Courses* (2013), ACM Press, pp. 1–20. 24

[MZWVG07] MÜLLER P., ZENG G., WONKA P., VAN GOOL L.: Image-based procedural modeling of facades. *ACM Transactions on Graphics 26*, 99 (July 2007), 85. 22

[NGDA*16] NISHIDA G., GARCIA-DORADO I., ALIAGA D. G., BENES B., BOUSSEAU A.: Interactive sketching of urban procedural models. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 130. 6, 22

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: FiberMesh. *ACM Transactions on Graphics 26*, 99 (2007), 41. 25

[NK03] NOVOTNI M., KLEIN R.: 3D zernike descriptors for content based shape retrieval. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003),

ACM, pp. 216–225. 110

[OFCD02] OSADA R., FUNKHOUSER T., CHAZELLE B., DOBKIN D.: Shape distributions. *ACM Transactions on Graphics (TOG) 21*, 4 (2002), 807–832. 110

[OLGM11] OVSJANIKOV M., LI W., GUIBAS L., MITRA N. J.: Exploration of continuous variability in collections of 3D shapes. *ACM Transactions on Graphics (TOG) 30*, 4 (2011), 33. 25, 112, 113

[Pag13] PAGANO R. R.: *Understanding statistics*. Oxford University Press, 2013. 50

[PIMD08] PATTERSON IV A., MORDOHAI P., DANIILIDIS K.: Object detection from large-scale 3d datasets using bottom-up and top-down descriptors. In *Computer Vision–ECCV 2008*. Springer, 2008, pp. 553–566. 111

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural Modeling of Cities. *28th annual conference on Computer graphics and interactive techniques*, August (2001), 301–308. 21

[PPT*08] PAPADAKIS P., PRATIKAKIS I., TRAFALIS T., THEOHARIS T., PERANTONIS S.: Relevance feedback in content-based 3D object retrieval a comparative study. *Computer-Aided Design and Applications 5*, 5 (2008), 753–763. 111

[PPTP10] PAPADAKIS P., PRATIKAKIS I., THEOHARIS T., PERANTONIS S.: Panorama: A 3d shape descriptor based on panoramic views for unsupervised 3d object retrieval. *International Journal of Computer Vision 89*, 2 (2010), 177–192. 69, 111, 137, 152, 158, 172

[RA99] RAMAMOORTHI R., ARVO J.: Creating generative models from range images. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 195–204. 22

[RGS04] RAAB R., GOTSMAN C., SHEFFER A.: Virtual woodwork: Making toys from geometric models. *International Journal of Shape Modeling 10*, 01 (2004), 1–29. 24

[SA07] SORKINE O., ALEXA M.: As-Rigid-As-Possible Surface Modeling. *Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), 109–116. 51, 99

[SBM*10] STAVA O., BENES B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of l-systems. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 665–674. 21

[SJWS13] SUNKEL M., JANSEN S., WAND M., SEIDEL H.-P.: A correlated parts model for object detection in large 3d scans. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 205–214. 111

[SLJ*15] SZEGEDY C., LIU W., JIA Y., SERMANET P., REED S., ANGUELOV D., ERHAN D., VANHOUCKE V., RABINOVICH A.: Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 1–9. 114, 161

[SMKF04] SHILANE P., MIN P., KAZHDAN M., FUNKHOUSER T.: The Princeton Shape Benchmark. In *Shape modeling applications* (2004), IEEE, pp. 167–178. 143, 151, 155,

165

[SMKLM15] SU H., MAJI S., KALOGERAKIS E., LEARNED-MILLER E.: Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 945–953. 114, 161, 162, 168

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse Procedural Modelling of Trees: Inverse Procedural Modeling of Trees. *Computer Graphics Forum 33*, 6 (Sept. 2014), 118–131. 6, 21

[SS08] SCHMIDT R., SINGH K.: Sketch-based procedural surface modeling and compositing using surface trees. *Computer Graphics Forum 27*, 2 (2008), 321–330. 26

[SSCO08] SHAPIRA L., SHAMIR A., COHEN-OR D.: Consistent mesh partitioning and skeletonisation using the shape diameter function. *Visual Computer 24*, 4 (2008), 249–259. 50

[SSGD03] SUNDAR H., SILVER D., GAGVANI N., DICKINSON S.: Skeleton based shape matching and retrieval. In *Shape Modeling International* (2003), IEEE, pp. 130–139. 110

[SSUF10] SCHINKO C., STROBL M., ULLRICH T., FELLNER D. W.: Modeling procedural knowledge: a generative modeler for cultural heritage. In *Digital Heritage*. Springer, 2010, pp. 153–165. 23

[STBB14] SMELIK R. M., TUTENEL T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum 33*, 6 (2014), 31–50. 5, 21

[SvHVG*08] STRECHA C., VON HANSEN W., VAN GOOL L., FUA P., THOENNESSEN U.: On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *Computer Vision and Pattern Recognition - CVPR* (2008), IEEE, pp. 1–8. 109

[SWS10] SCHERER M., WALTER M., SCHRECK T.: Histograms of oriented gradients for 3d object retrieval. In *WSCG 2010, 18th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision. Full Papers Proceeding* (2010), pp. 41–48. 110

[SWSJ05] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: ShapeShop: sketch-based solid modeling with BlobTrees. *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling* (2005). 26

[SWSJ07] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *ACM SIGGRAPH 2007 courses* (2007), ACM, p. 43. 25

[TA09] TATSUMA A., AONO M.: Multi-Fourier spectra descriptor and augmentation with spectral clustering for 3D shape retrieval. *The Visual Computer 25*, 8 (Aug. 2009), 785–804. 111, 137, 139, 145

[TAOZ12] TAGLIASACCHI A., ALHASHIM I., OLSON M., ZHANG H.: Mean curvature skeletons. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 1735–1744. 79, 88

[TCF09] TOLDO R., CASTELLANI U., FUSIELLO A.: A bag of words approach for 3d object categorization. In *Computer Vision/Computer Graphics CollaborationTechniques*. Springer, 2009, pp. 116–127. 110

[TGB13] THIERY J.-M., GUY M., BOUBEKEUR T.: Sphere-Meshes: shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics 32*, 6 (Nov. 2013), 1–12. 24, 98

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Transactions on Graphics (TOG) 30*, 2 (2011), 1–14. 6, 21, 22

[TMW02] TOBLER R. F., MAIERHOFER S., WILKIE A.: Mesh-Based Parametrized L-Systems and Generalized Subdivision for Generating Complex Geometry. *International Journal of Shape Modeling 08*, 02 (2002), 173–191. 21

[TPSHSH13] TAKAYAMA K., PANOZZO D., SORKINE-HORNUNG A., SORKINE-HORNUNG O.: Sketch-based generation and editing of quad meshes. *ACM Transactions on Graphics 32*, 4 (2013), 1. 25

[TV08] TANGELDER J. W., VELTKAMP R. C.: A survey of content based 3d shape retrieval methods. *Multimedia tools and applications 39*, 3 (2008), 441–471. 110

[UF11] ULLRICH T., FELLNER D. W.: Generative object definition and semantic recognition. In *Proceedings of the 4th Eurographics conference on 3D Object Retrieval* (2011), Eurographics Association, pp. 1–8. 6, 7, 23

[VGDA*12] VANEGAS C. A., GARCIA-DORADO I., ALIAGA D. G., BENES B., WADDELL P.: Inverse design of urban procedural models. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 168. 6, 22

[VGHS07] VAN GINNEKEN B., HEIMANN T., STYNER M.: 3D segmentation in the clinic: A grand challenge. *3D segmentation in the clinic: a grand challenge* (2007), 7–15. 109

[VH99] VELTKAMP R. C., HAGEDOORN M.: *State-of-the-Art in Shape Matching*. Tech. rep., Principles of Visual Information Retrieval, 1999. 110

[VKTS*11] VAN KAICK O., TAGLIASACCHI A., SIDI O., ZHANG H., COHEN-OR D., WOLF L., HAMARNEH G.: Prior knowledge for part correspondence. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 553–562. 114

[VKZHCO11] VAN KAICK O., ZHANG H., HAMARNEH G., COHEN-OR D.: A survey on shape correspondence. In *Computer Graphics Forum* (2011), vol. 30, Wiley, pp. 1681–1707. 110, 134

[Vra05] VRANIC D. V.: Desire: a composite 3d-shape descriptor. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on* (2005), IEEE, pp. 4–pp. 111

[WBK08] WESSEL R., BARANOWSKI R., KLEIN R.: Learning distinctive local object characteristics for 3d shape retrieval. In *VMV* (2008), pp. 169–178. 114

[WLPL15] WANG Y., LIU Z., PANG F., LI H.: Boosting 3D model retrieval with class vocabularies and distance vector revision. In *TENCON 2015-2015 IEEE Region 10 Conference* (2015), IEEE, pp. 1–5. 113, 114

[WMWL15] WU H., MIAO Z., WANG Y., LIN M.: Optimized recognition with few instances based on semantic distance. *The Visual Computer 31*, 4 (2015), 367–375. 110

[WSK*15] WU Z., SONG S., KHOSLA A., YU F., ZHANG L., TANG X., XIAO J.: 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1912–1920. 114, 161

[WXL*11] WANG Y., XU K., LI J., ZHANG H., SHAMIR A., LIU L., CHENG Z., XIONG Y.: Symmetry Hierarchy of Man-Made Objects. In *Computer graphics forum* (2011), vol. 30, Wiley Online Library, pp. 287–296. 5, 24

[WYS*15] WU R., YAN S., SHAN Y., DANG Q., SUN G.: Deep image: Scaling up image recognition. *arXiv:1501.02876 7*, 8 (2015). 14

[XL07] XU D., LI H.: 3D shape retrieval integrated with classification information. In *Fourth International Conference on Image and Graphics.* (2007), IEEE, pp. 774–779. 113

[XLZ*10] XU K., LI H., ZHANG H., COHEN-OR D., XIONG Y., CHENG Z.-Q.: Style-content separation by anisotropic part scales. *ACM Transactions on Graphics (TOG) 29*, 6 (2010), 184. 23, 112

[YCHK15] YUMER M. E., CHAUDHURI S., HODGINS J. K., KARA L. B.: Semantic shape editing using deformation handles. *ACM Transactions on Graphics 34*, 4 (July 2015), 86:1–86:12. 23

[YK12] YUMER M. E., KARA L. B.: Co-abstraction of shape collections. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 166. 23

[ZC01] ZHANG C., CHEN T.: Efficient feature extraction for 2d/3d objects in mesh representation. In *International Conference on Image Processing* (2001), vol. 3, IEEE, pp. 935–938. 65

[ZYH*15] ZHOU Y., YIN K., HUANG H., ZHANG H., GONG M., COHEN-OR D.: Generalized cylinder decomposition. *ACM Transactions on Graphics (Special Issue of SIGGRAPH Asia) 34*, 6 (2015), 171. 24, 98