FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Struktur-Erhaltende Manipulation von Geometrischen 3D Modellen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Visual Computing**

eingereicht von

**BSc. Bernhard Steiner**
Matrikelnummer 0825391

an der Fakultät für Informatik
der Technischen Universität Wien

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

Wien, 4. September 2014

| Bernhard Steiner | Michael Wimmer |
|---|---|

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Structure-Aware Manipulation of Geometric 3D Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Visual Computing**

by

**BSc. Bernhard Steiner**
Registration Number 0825391

to the Faculty of Informatics
at the Vienna University of Technology

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

Vienna, 4th September, 2014

Bernhard Steiner

Michael Wimmer

# Erklärung zur Verfassung der Arbeit

BSc. Bernhard Steiner
Minciostraße 4/13
1150 Wien

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

Wien, 4. September 2014

Bernhard Steiner

# Acknowledgements

Many thanks to Przemyslaw Musialski who gave me a lot of useful insights during the development of this framework. Thank you for pushing me in the right direction.

I would also like to thank Michael Wimmer for his helpful comments while writing my thesis, even when being on vacation.

Thanks to all the people who took part in the user study and allowed me to collect all the necessary data in less than a week. I also want to thank the Vienna University of Technology for funding my project, especially the touchscreen notebook for my user study, via the Förderungsstipendium.

Special thanks to my colleges Silvana Podaras and Simon Wallner for the lot of work they had with proofreading my thesis and for building me up when I nearly gave up.

# Kurzfassung

Durch die in den letzten Jahren steigende Verbreitung von 3D-Druckern und ähnlicher Hardware, werden Anwender immer öfter mit 3D-Modellierungsaufgaben konfrontiert. Dies kann besonders für ungeübte Benutzer zu einem Problem werden. Structure-Aware Shape-Processing bietet genau für solche Anwender die Möglichkeit 3D-Modelle anzupassen und zu verändern, ohne mit der Komplexität der 3D-Modellierung vertraut zu sein. Solche Systeme unterstützen den Anwender dadurch, dass sie die 3D-Inhalte an User-Änderungen anpassen wobei globale Charakteristika des Modells erhalten bleiben.

In dieser Arbeit wird eine neue Formulierung für Symmetrie in 3D-Geometry gezeigt, welche die Anordnung der symmetrischen Elemente entlang einer parametrischen Kurve beschreibt. Um dies umsetzen zu können wird außerdem das Part-Relation Model, welches als Basis für viele Modellierungs-Programme verwendet wird, so erweitert, dass es mittels eines Layered-Graphen solche Symmetrien besser darstellen kann. Des weiteren wird ein Algorithmus gezeigt, welcher mit Hilfe einer Connected-Component-Analyse solche direkten Relationen findet, die die Verbindung zwischen zwei Teilen des Modells am besten beschreiben.

Alle diese Algorithmen werden im Anschluss in einem, auf Übertragung basierendem, Framework umgesetzt, welches sowohl mit einer Maus als auch über einen Touchscreen bedient werden kann. Um die Qualität dieser Anwendung zu bestätigen wird danach eine User-Study gezeigt, welche im Speziellen überprüft ob das System von ungeübten Anwendern bedient werden kann.

# Abstract

Inexperienced users get more and more exposed to 3d-modelling applications due to the wider availability of 3d printers and other such hardware. Structure-aware shape manipulation is one way to help them adapting and extending 3d models without having to handle the complexity of a traditional 3d-modelling environment. Such systems support the user by adapting an object to changes introduced by the user while retaining global shape characteristics.

This thesis presents a new method for describing symmetries in geometry The method describes the arrangement of parts in terms of a parametrized curve. Additionally, the part-relation system used to describe models in structure-aware modelling applications is extended to a layered graph model that can describe symmetries better. In this thesis, also an extension for finding better binary relations between adjacent parts is shown, which uses a connected-component search to identify binary relations that approximate the connectivity between the parts best.

It is shown how all these methods are implemented in a propagation-based framework, which can handle user input via mouse and via touchscreen. Using this framework, a user study is presented, which came to the result that the system is well usable by inexperienced users.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation

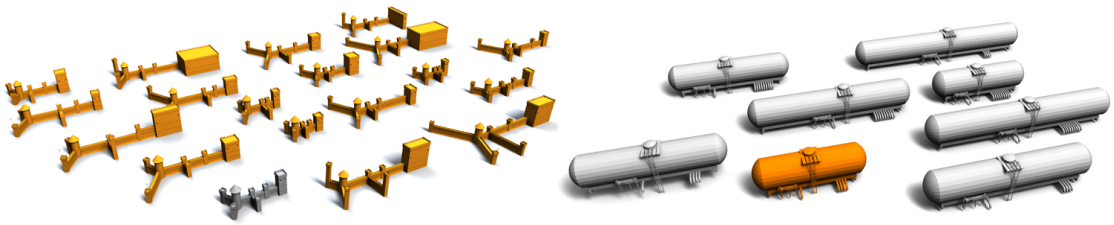


Figure 1.1: Structure-aware shape-manipulation systems are used to adapt 3d models while retaining global shape characteristics. Reprinted from [BWKS11] and [BWSK12].

In computer graphics, one of the most labor-intensive tasks is the creation of suitable digital content for specific application scenarios. While today, rendering is highly sophisticated and automated, 3d modelling is still a very challenging task due to the steep learning curve and the amount of manual work required.

From the beginning of digital content creation, methods and applications used have been very complex, and even after some years of working with them, it can be difficult to create exactly the result one wants. Since there is more and more digital 3d content available, users are faced not only with the challenge of creating but also of adapting and extending existing content. The average user is not interested in learning a traditional modelling application, since this is a time-consuming task. Moreover, although there exist several free model databases, e.g., 3D Warehouse[1], these models are in general not perfectly suited to the user's needs and have to be adapted. This task can be very challenging for inexperienced users. For this reason, more advanced methods for modelling

[1] `https://3dwarehouse.sketchup.com`

and modifying 3d content are required. Additionally, due to the wide availability of tablets and other modern hardware, especially these users are becoming the main focus group of such applications.

Over the last few years, a lot of scientific effort was made in order to develop methods for editing models by inexperienced users. Structure-aware model-manipulation systems allow these users to manipulate a model while the system updates the rest of the shape to fit to the changes made. Such approaches allow manipulating the model on a high abstraction level, where the user does not have to deal with vertices, edges and faces. Such algorithms are also required, since in future, there might be a 3d printer in every household, allowing everyone to fabricate everything at home. Although this is science fiction at the moment, the scientific community has started to think about this possibility. In this context, a big advantage of structure-aware model manipulation is that additional constraints can be enforced on the model. For example, physical stability could be tested, or only models could be created, which can be fabricated by a given machine.

It can also be interesting for more experienced users to use application-guided design methods. In game development, level designers can, in most systems, only apply rigid transformations to a model. In many cases, the level designer is not perfectly satisfied with the model, and has to send it back to the 3d artist. If structure-aware methods were present in the level-editor, the level designer would be able to handle minor changes to the model without having the designer rework it.

One direction to go is the so called *structure-aware shape manipulation*. These methods aim at creating variations, or adapting a given input model, while retaining global features like symmetry and connectivity (see Figure 1.1). Over the last few years, a lot of different approaches for performing such structural adaptions have been developed, in order to help the user to design exactly the model that she desires. But the majority of existing algorithms only try to preserve structural features [CLDD09], [GSMCO09], or have very limited support for adapting the overall structure of the model to changes made by the user [BWSK12], where structural changes can only happen along straight lines and on regular grids. The lack of complex structural adaption in models limits the amount of possible variations that can be achieved. For example, it is impossible to adjust the number of basements on a castle tower using only lines or regular grids, since they are aligned on a circle (see Figure 1.2).

## 1.2 Structure-Aware Model Manipulation

Most structure-aware model-manipulation applications follow the analyse and edit principle as shown in Figure 1.3 [GSMCO09, BWSK12, MWZ+13]: First the model is analysed, and information about the structure of the model is generated. After this, the user can manipulate the mesh and the system updates the model.

The information generated in the model analysis stage varies from approach to approach, but in all approaches there is some kind of most primitive structure that is identified. This can range from voxels, over interesting lines or polygons, to sub-models that group triangles with similar properties. In addition to this, also the relations

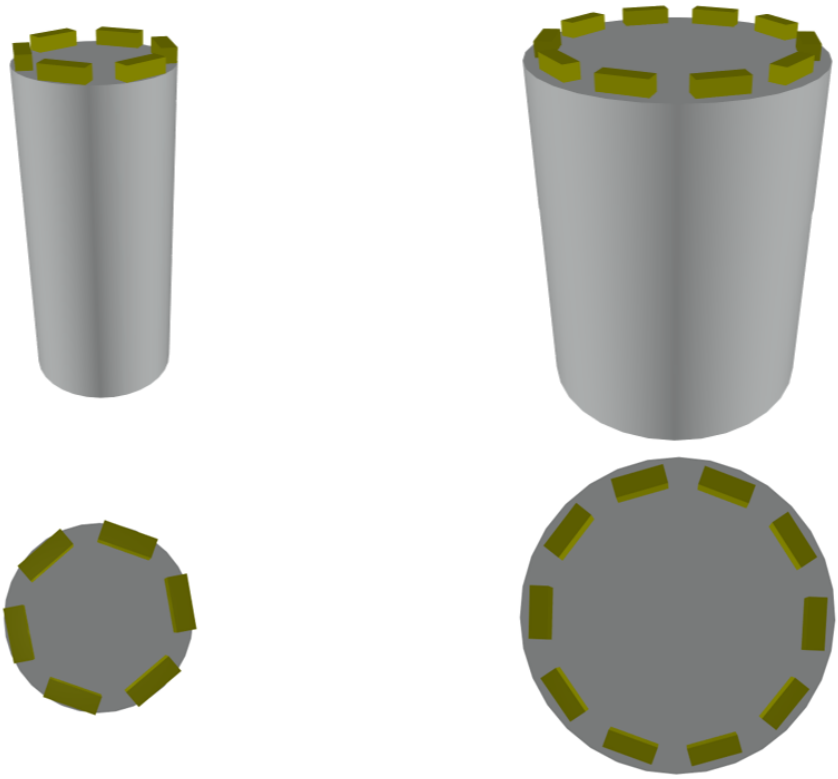


Figure 1.2: The number of basements should be increased when the tower changes its radius.

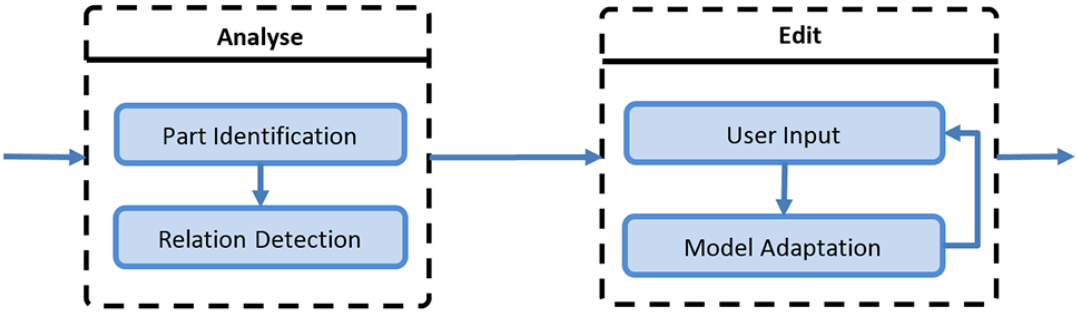



Figure 1.3: Stages in the general design of a structure-aware model manipulation system.

between these primitive elements (or parts) are identified. Relations can describe physical connectivity (so-called binary relations) or any other kind of relation, e.g., symmetries, which are called higher-order relations.

After the analysis stage, the user can modify the model. Whenever changes are

introduced, the system tries to adapt the model such that both, the structural features and the user input, are preserved. This includes modifications of the parts due to binary relations, but also the enforcement of global constraints like symmetries. This stage is repeated until the user has performed all necessary changes and a final model is produced. In our implementation, a propagation-based modification algorithm is used, which transfers changes through the parts and binary relations. Also, whenever the pre-conditions for updating a higher-order relation are fulfilled, the global constraints described by this relation are enforced.

## 1.3 Problem Definition

The aim of this work is to design a framework that handles, based on the part-relation model introduced by Mitra et al. [MWZ+13] and Zheng et al.[ZFCO+11], how global structures in a 3d model should react to changes made by the user. In this work, we will explicitly target inexperienced users, thus we will assume that the users of our framework do not have any prior knowledge in the field of 3d modelling.

We want to improve several sub-stages of the general design principle in order to support a wider variety of structural changes. First of all, we want to identify symmetric parts in the model that are aligned along a parametrized curve. For this, we have to find a new formulation for symmetry groups that can cover the positioning along such a curve as well as the orientation along the Frenet frame of the curve. In the manipulation stage, we will allow the system to adapt the number of repetitions of the elements in such a group when the user modifies the model. This method will allow us for example to adjust the number of basements of the previously mentioned tower without losing the local arrangement of the components. In addition to this, we will describe a multi-layered graph structure that is generated from the model, which stores information about the geometry as well as information about the logical arrangement of parts and how they are related to each other. Compared to the traditional connectivity graph, this data structure will allow us to express symmetry groups and their parameters in the same graph as parts and relations. These techniques will then be integrated in an as-intuitive-as-possible touch-based framework, where we will use a propagation-based algorithm to update parts, binary relations and symmetries to changes made by the user. All of these methods should be designed so that they work in real time, to provide the users with direct feedback about their interactions.

Using the developed application, a user study should be performed to check if our application can be intuitively handled by inexperienced users. We also want to find out if touch interaction is liked more or less by the users than mouse interaction. These questions will be answered by letting users perform a number of different tasks with our system, monitoring performance data and asking them which interaction method they prefer after each task.

## 1.4 Contribution

In the scope of this thesis, we develop an approach for structure-aware modelling based on the work of Zheng et al. [ZFCO+11]. We extend their method in order to cover a wider range of possible adaptions in the model. For this we contribute in the following fields:

- A new formulation for symmetry groups in terms of local arrangement in the Frenet-frame of a parametrized curve is presented. For this formulation, we prove that the mathematical body that is constructed by our formulation satisfies the constraints for a symmetry group, which allows us to apply the whole theory that is already developed for symmetry groups to our approach.
- A new method for binary relation detection is shown, where connected-component analysis is applied to an intersection point cloud reducing the number of binary relations between two parts by finding significant binary relations that approximate the connectivity.
- A structure-aware model-manipulation framework is presented, which includes both previously mentioned contributions and shows that it is possible to design a system based on our theory of symmetries. For this, we extend the part-relation model by using a layer graph to cover higher-order relations better.
- We present the results of our user study, which tests if our system can be handled by inexperienced users. We also show how prior knowledge influences the choice of the preferred interaction method.

## 1.5 Structure of the Work

This thesis continues with Chapter 2, which gives an overview of the state-of-the-art in structure-aware shape processing and shows the different approaches that can be used to build such systems.

Chapter 3 explains the basic concepts that are used in our work, starting with a description of a general structure that is used in most shape-manipulation systems, followed by a overview on the theory of parametric curves. After this, we describe how symmetry is defined and how it can be used in 3d geometry. We also give an overview on how such symmetries can be identified in geometric models. At the end of this chapter, an overview on quantitative usability testing is given, which is the basis for our user-study.

Our implementation of the structure-aware model-manipulation framework is presented in Chapter 4, together with the novel formulation for symmetries on curves. These algorithms and techniques are evaluated in Chapter 5, where also the user study is described in detail, and the findings of it are presented. Besides this, some performance statistics are shown, and the limitations of our system are described.

This thesis ends with a reflection of our work and an outlook on future work in Chapter 6.

CHAPTER 2

# State of the Art

Structure-Aware model manipulation deals with how models should be adapted to influences coming from outside. The methods used vary from approach to approach, but it is possible to group all methods in two categories: Methods which take an input and propagate changes through the mesh, and on the other hand approaches, which construct a constraint system for the whole mesh and try to solve this problem set by minimizing the error introduced. Beside this, there are some methods, that deal with the deformation of meshes directly. Most of the current methods follow the analyse and edit principle: First, the mesh is analysed and constraints are searched, then the user can edit the model while the system maintains these constraints.

## 2.1 Mesh Deformation

One approach for transforming geometry are the class of methods that work directly on this geometry. These techniques are often working without any high-level knowledge of the model and are only dealing with the local or global geometry itself. Since these methods are only far related to our approach, they will be discussed very briefly here.

In all these methods, the goal is to find a deformation of a mesh under a given constraint, which preserves the structure of the geometry. Laplacian coordinates (Sorkine et al. [SCOL$^+$04]) are describing each vertex as the relative deviation from the centroid of the vertice's neighbourhood. When vertices are moved, a linear system of equations describing these relations is constructed and solved in a least square sense, giving the final position for each vertex. A non linear iterative extension to this is proposed by Sorkine and Alexa [SA07] which is called *As-Rigid-As-Possible* surface deformation. The non-linearity comes from introducing a shape deformation energy, that adds a global constraint to the deformation.

Another shape based deformation method called *Green Coordinates* has been proposed by Lipman et al. [LLCO08], where the user input is not directly given on the geometry

but instead on a control cage. The major advantage here is, that green coordinates can be given in closed form in 2d and 3d, although the general formulation works also in $\mathbb{R}^n$.

When an underlying skeleton is present in the model, Jacobson et al. [JBK+12] are showing how the as-rigid-as-possible approach can be improved to run much faster while resulting in the same quality. Results of all these methods are presented in Figure 2.1 and Figure 2.2.

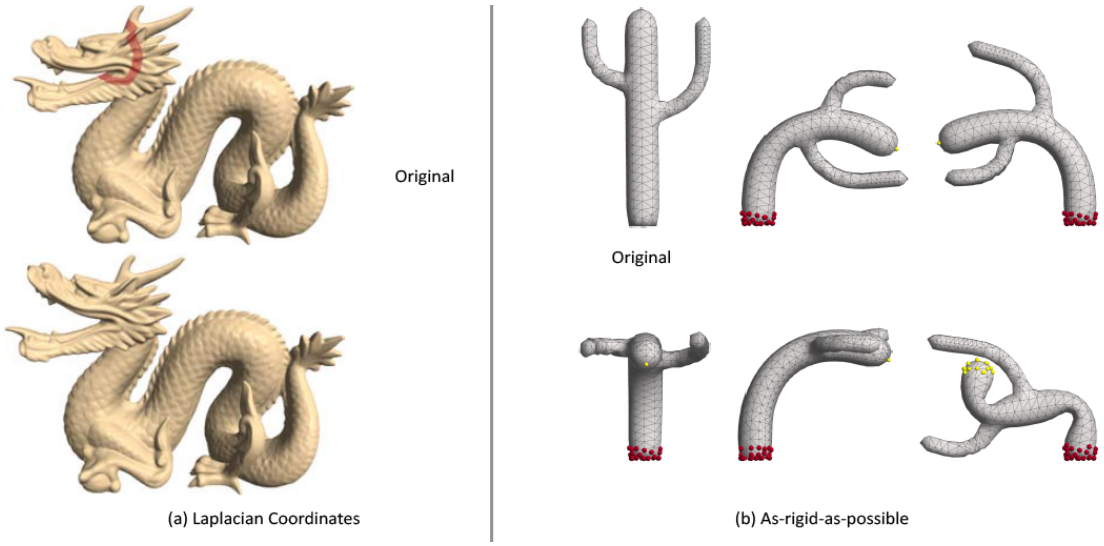



Figure 2.1: Pure geometric deformation examples: (a) Laplacian Coordinates [SCOL+04], (b) As-rigid-as-possible [SA07]

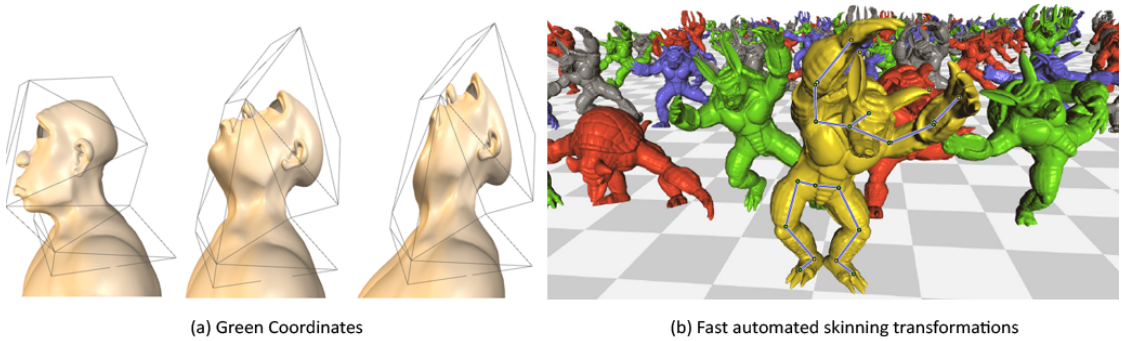



Figure 2.2: Pure geometric deformation examples: (a) Green Coordinates [LLCO08], (b) Fast automated skinning transformation [JBK+12]

As-rigid-as possible and all related approaches tend to deform the model in a non structure-aware manner when non-uniform scaling is applied. For example, there is, due to the missing global analysis of the shape, no guarantee that circles and other important features are preserved during modifications. Such a global analysis in a pure geometric

approach is shown by Kraevoy et al. [KSSCO08]: The whole model is embedded in a protective volumetric grid, where each cell and the underlying geometry is analysed in terms of vulnerability. In this context, the vulnerability of a cell describes if the geometry in a cell can be deformed non-uniformly. For each cell three parameters, which describe the vulnerability along each axis, are calculated. Due to the use of three values per cell, the algorithm can distinguish between e.g. spheres, which should not be scaled non-uniformly and cylinders, which can be scaled anisotropically along the height-axis, but not along the other axis. The vulnerability along each axis is a combination of the two geometric features slippage and normal curvature. Slippage (as defined by Gelfand and Guibas [GG04]) estimates how persistent a surface is with respect to a given transformation type. This can be seen as an indicator if a local patch would stay on the surface after applying the transformation. In terms of non-uniform scaling, it can be seen that a surface is only slippable if it's normal is orthogonal to the scaling axis. Using slippage alone would not be sufficient, since it does not take the scaling direction into account. Normal curvature on the other hand predicts how much a surface will bend when scaling in a given direction is applied. Together, normal curvature and slippage give a good estimation about the vulnerability in a cell.

When the user changes the control-cage, the algorithm has to update all cells using the previously calculated parameters. For each cell a scale-only gradient is calculated, such that they accumulate to the desired transformation. This gradients are then used to calculate the full per-cell transformation that has to be applied. In this step also the compatibility between neighbouring cells has to be enforced. For more details about the calculation we refer to the original paper. In general, this method tends to produce very good results, as shown in Figure 2.3, where the original model (a) is distorted by non-uniform scaling (b). In comparison, the non-homogeneous resizing method (c) preserves a lot more local features. The protective grid used here is shown before (d), and after the modification (e). Although results of this method are very promising, there are some situations where the algorithm does not perform well and needs additional user input to detect the vulnerability correctly.

## 2.2 Controller Based Propagation

Beside direct mesh manipulation, one of the first and probably the most influencing work in this field, is the iWires framework by Gal et al. [GSMCO09]. The general idea here is that most man-made objects can be described by a few interesting curves, the so called *wires*. Based on this observation, they present an analyse-and-edit framework that propagates user changes through the whole model.

In the analyse stage, the system tries to find wires in the input model. Although this does not apply to general models, most man-made objects contain sharp-creased lines, which are good candidates for this. These edges lie in general on the intersection of two smooth surfaces, thus they can be seen as "edges" of the mesh. The final wires are then found by a greedy algorithm, which starts with a crease edge, that has not served jet as seed. From this starting point, it walks along adjacent crease edges, choosing always the

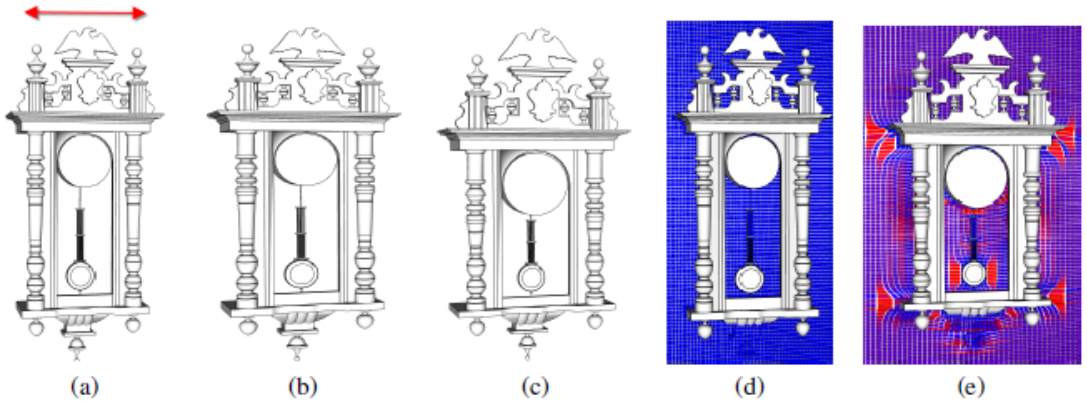



Figure 2.3: Results of the "Non-homogeneous resize" algorithm. (a) Original model, (b) Standard non-uniform scaling distorts a lot of features, that are preserved by the proposed method (c). (d) and (e) are showing the protective grid before and after the transformation. Reprinted from [KSSCO08].

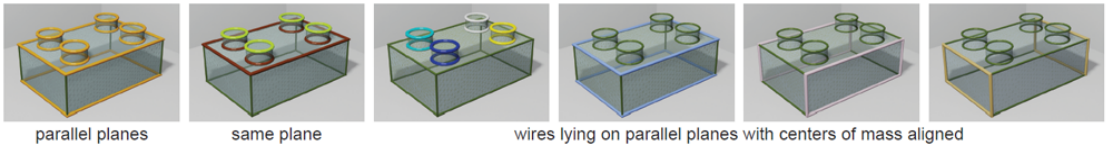



Figure 2.4: The constraints that iWires takes into account shown on a lego stone model. Reprinted from [GSMCO09].

edge that would keep the wire planar (if possible) when a crossing is reached. This is repeated until a closed wire is found, or no further edges could be added.

For each wire a number of parameters are captured: It is detected if the wire is planar or non-planar. In addition the atomic type, which describes if the wire can be approximated by a primitive type, has to be determined. In their application they use lines, (part of) circles, (part of) ellipses, and polynomial curves. If the wire cannot be approximated by such a structure, it is separated into several sub-wires by splitting at salient internal angles. When also these sub-wires cannot be approximated, the wire has no special type attached, and is treated further on as a polyline. For compound wires, an additional attribute is attached: The angle between the tangent vectors at the connection points.

When all wires are analysed, iWires searches for wire groups that share a common property, for example wires that lie on the same plane, wires on parallel planes and wires that lie on parallel planes and have approximately aligned center of mass (see Figure 2.4). Symmetries are also considered by testing all pairs of wires for them. All sets where at least five wires share a symmetry build together a symmetry group.

When all the wires, their relations and the symmetries are identified, editing can

start. The user has several possibilities to interact with the system, directly dragging surfaces or parts of them, deforming wires, or sketch based interaction (Figure 2.5 step 3). Due to this interactions, the wires will be deformed and previously identified constraints could eventually be violated. Thus, the system first optimizes the individual wires by modifying them such that the characteristics of the original shape are preserved, while the modifications made by the user are also maintained. For example, when the original wire has been a planar circle, than the system projects the modified wire to the best fitting plane and constructs a circle that has the same center of mass and radius as the modified wire. To update the rest of the model, changes made to wires are propagated to untouched adjacent wires, where the local optimization happens again, followed by a group optimization step. Group optimization handles all constraints that concerns multiple wires like symmetry or parallelity (see Figure 2.5 steps 4-7). These steps are repeated until all wires are updated, or until there exists no further wires that have to be updated. At the end of the propagation, the surfaces of the mesh are reconstructed from the wire groups, giving a complete and modified model (Figure 2.5 steps 8-10).

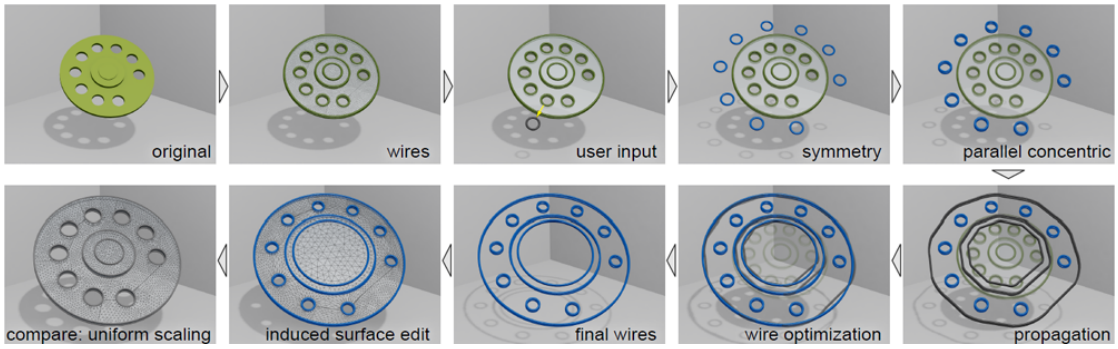



Figure 2.5: Stages of the iWires editing process, starting with a user interaction followed by propagation and optimization. At the end the final faces are reconstructed. Reprinted from [GSMCO09].

The iWires paper is important due to two mayor contributions: First, it is the first work that shows an analyse-and-edit approach for structure-aware shape manipulation. Second, and more important, this framework produced very good results for man-made shapes (see Figure 2.6 for results), showing that structure-aware shape editing can be applied to a wide variety of shapes without any prior knowledge needed.

In the iWires framework, the shape is characterized through a number of 1d controllers, the wires. These controllers have very limited support for constraining the allowed modifications. Zheng et al. [ZFCO+11] show in their work, that the usage of 3-dimensional controllers can improve the quality of a shape modification algorithm. The general technique used in this work is similar to the pre-described structure: It consists of a shape analysis stage and an editing stage.

Again, the focus of this work is on man-made objects, thus it assumes, that a model is composed from several meaningful parts, that can be decomposed and approximated

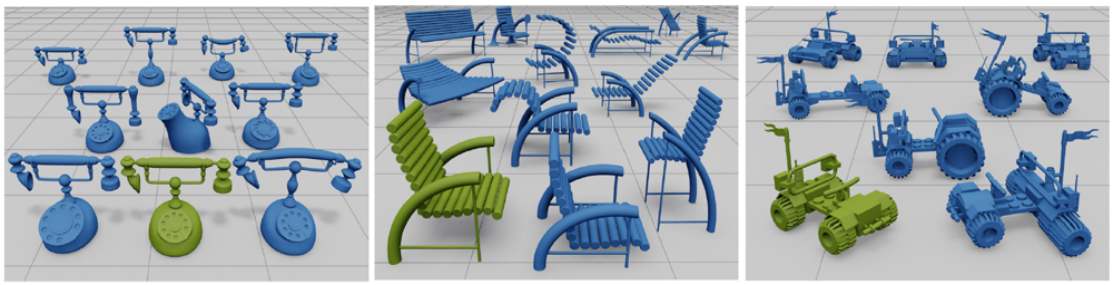


Figure 2.6: Results of the iWires framework. Adapted from [GSMCO09].

by controllers. The authors use a hierarchical segmentation algorithm, which performs a bottom-up clustering. Starting with the smallest cluster size, this algorithm merges clusters based on the cost that a merge would introduce. The main difficulty is to define a cost function for the merging: The function used here does not only consider the fitting error of different types of primitives, but is also influenced by the boundary concavity and the boundary length. Since it is computationally heavy to consider all possible types of controllers in the clustering, only planes, spheres and cylinders are used. After the decomposition the best fitting controller from the final controller list is found for each segment. For controllers, Zheng et al. propose to use four different types: Spheres, Cylinders, Cuboids and generalized Cylinders, since these controllers can well approximate the models, while still being fairly simple. Figure 2.7 shows the four controller types and their behaviour under deformation. With exception of the sphere, all controllers can be modified anisotropically. Each of these controllers has a pre-defined degree of freedom, which is described by feature curves (marked in yellow). For example, the sphere has only one degree of freedom (uniform scaling), which is given by a curve along the equator. Cylinders have two feature curves, one on each side of the cylinder.

As in iWires, it is not enough to know only the decomposition and the controllers. It is also required to know how these controllers are related to each other. Such relations like symmetry, parallelism and orthogonality are detected on the level of the feature curves, but can also be modified by the user to achieve different editing results.

Editing is done by manipulating individual controllers depending on their restrictions. All types of controllers can be translated and isotropically scaled, some of the controllers allow also other manipulations, e.g. on the cylinder frustum scaling can be applied along the cylinder axis. Changes made on controllers are then propagated through the whole controller structure: Starting from the modified controller, a breadth-first propagation is performed in a controller-to-controller manner. Whenever the system touches a controller that depends on more complex relations like parallelism, the algorithm also enforces these constraints. To ensure that the system reaches a final state and does not produce loops, each controller in the system is only touched once by the propagation, which is very similar to the way how iWires [GSMCO09] propagates changes. A stepwise example of the propagation schema used can be seen in Figure 2.8.

In contrast to iWires, this method does not deal with geometric features, but with

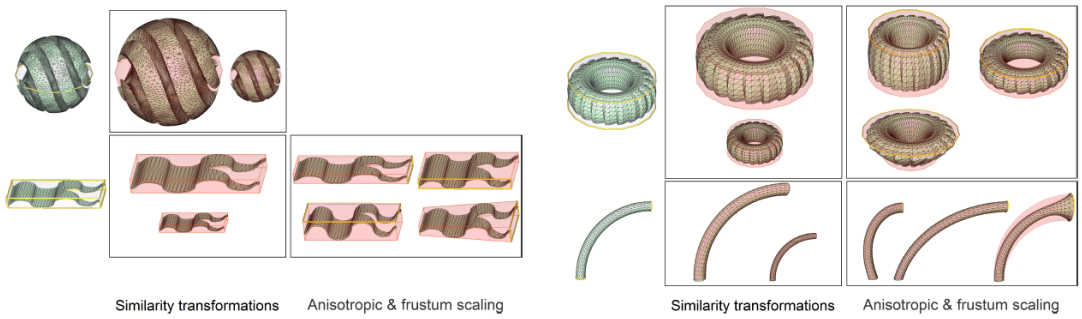



Figure 2.7: The four controller types used, and how they can be transformed. Adapted from [ZFCO+11].

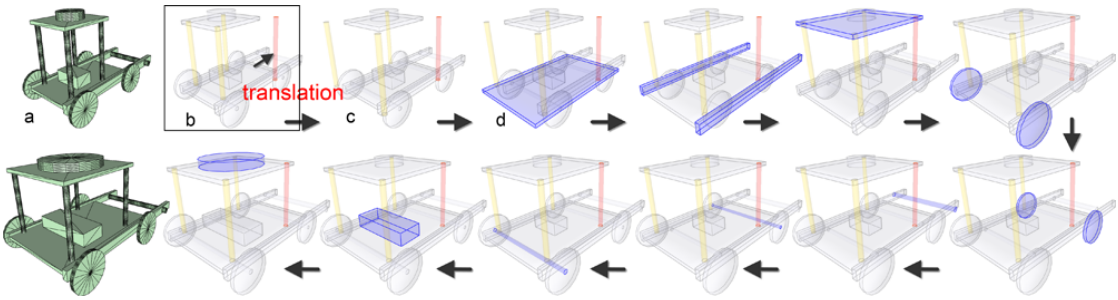



Figure 2.8: Propagation algorithm by Zheng et al. [ZFCO+11] in action. Reprinted from [ZFCO+11].

approximations instead, thus it has to update the geometry contained in the controllers after the propagation. For this, a surface-based deformation approach is used, where virtual edges are constructed between the controller and the geometry. This allows the system to overcome the problem of geometry that lies on the intersection of two controllers.

Compared to the iWires framework, this shape manipulation system can handle a larger number of models, since it does not directly rely on shape features, which can be problematic in iWires when not enough wires can be found. On the contrary, the controllers can introduce some problems, especially when the decomposition algorithm does not produce good clusters. Another problem are self-intersections of controllers after modification, which are not handled by the system. Results produced by the "Component-wise Controller" framework are shown in Figure 2.9.

## 2.3 Rule Based Deformation

In contrast to systems that try to modify a 3d model by directly manipulating its parts, rule based systems use a grammar that describes how models are composed. Such procedural modelling frameworks have been used for a while in computer graphics (see

Figure 2.9: Results from the "Component-wise Controllers for Structure-Preserving Shape Manipulation" paper. Adapted from [ZFCO+11].

[Ebe03, MWH+06, PM01]), and their advantages can clearly be seen in the large variety of shapes they can construct. But when using shape grammars directly, it can be very hard for the user to generate exactly the result that is wanted. Beside this, it is very time consuming to specify the rules for a complex model by hand. A near related research field, the inverse procedural modelling, derives grammars from existing shapes (see [vBM+10]), but the problem of getting a specific model stays the same.

An approach that should overcome the limitations of such inverse procedural modelling frameworks has been proposed by Bokeloh et al. [BWS10]: Starting with an input shape, this method tries to identify shape operations that can be applied to the model while keeping the result *r-similar* to the input. R-similarity can be defined as a special type of local similarity, where for each point in a shape $\mathcal{S}_1$ a point in $\mathcal{S}_2$ can be found, which has a similar r-neighbourhood $N_r(x) := \{y \in \mathcal{S} | dist(x, y) \leq r\}$. The same principle can also be used to define r-symmetry, specifying that two points and their r-neighbourhood are symmetric.

All operations that the system can perform are specified using so-called *docking sites*. According to Bokeloh et al. docking sites are defined as follows: A set of surface curves $\alpha \subseteq \mathcal{S}$ is called docking site with respect to a transformation T, if the following three properties hold:

- $\alpha$ is r-symmetric under T
- $\alpha$ partitions the model into two topological disconnected parts $\mathcal{D}_\alpha$ and $\mathcal{R}_\alpha$. $\mathcal{D}_\alpha$ contains geometry that is not symmetric under T, and is called a *docker* for the docking site $\alpha$.
- $T(\alpha)$ also partitions the model into two topological disconnected subsets $\mathcal{D}_{T(\alpha)}$ and $\mathcal{R}_{T(\alpha)}$. $T(\alpha)$ is called secondary docking site and $\mathcal{D}_{T(\alpha)}$ secondary docker.

These docking sites could be identified directly on the triangle mesh, but in practice this might be inaccurate due to sliver triangles. Thus a voxel-based method is used, which works with a fixed sampling resolution and gives more stable results.

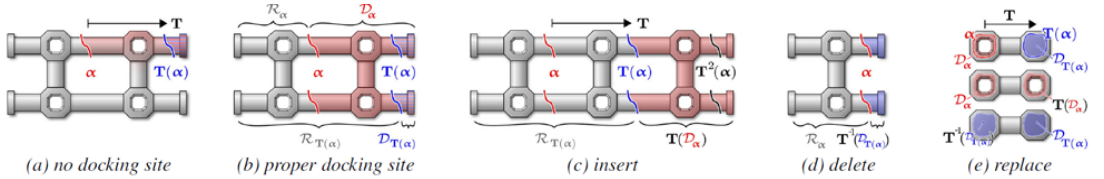



Figure 2.10: (a,b) Definition of docking sites and dockers (c-e) The three operations that can be applied. Reprinted from [BWS10].

On these docking sites, three operations can be applied: Insertion, where a new docker is added, deletion where an existing docker is removed, and replacement, that substitutes a docker by another one. Clearly, this can only work when the docking site splits the model into two parts (see Figure 2.10 a and b). Examples for these three operations can be seen in Figure 2.10 c-e. When applying such an operation to the input model, it has to be checked, that the modified part does not lead to any collisions in the modified object.

As mentioned before, it is also necessary to ensure that the result is r-similar to the input: Bokeloh et al. are showing that each operation that does not overlap will create a r-similar result. The last problem that has to be solved is, that after editing the model once, the situation might change, thus a formulation has to be found, that ensures the independence of the shape operations. This can be done by defining a general Chomsky type-0 grammar. Although this is the most general way, it might be a problem from the computational point of view. The authors propose to generate a context-free subset of the grammar, and show then how non context-free grid-based replication rules can be added to the system.

This whole grammar-based system is presented to the user in an interactive way: The user can select docking sites, and gets suggestions of dockers that can be modified. Thus the shape variation constructed can be controlled very easily by the user, allowing for the generation of exactly the model the user wants (see Figure 2.11a). Figure 2.11b-d are showing different variations of some input models (red).

When looking at these results, it is easy to see the major limitation: Changes to the model can only happen in discrete steps by inserting or deleting dockers. Since inserting (and deleting) uses only translations, but does not deform the parts itself, the fence in Figure 2.11c will always be aligned on a plane. Milliez et al. [MWCS13] also decompose an input model into disconnected parts that share a common boundary (which can be seen as dockers and docking sites), but instead of only applying grammar operations, the geometry itself is adjusted to changes. For each part in the model, several geometric representations (so called *rest positions*) are specified, that can be switched depending on the local context, which allows to choose the shape that matches the current situation best. In addition to selecting the best matching shape, the parts can be modified using an elastic deformation, allowing to exactly match the situation the user specifies. This deformation is based on the as-rigid-as-possible approach by Sorkine et al. [SA07], which models elastic shape deformation by minimizing an energy function that specifies the

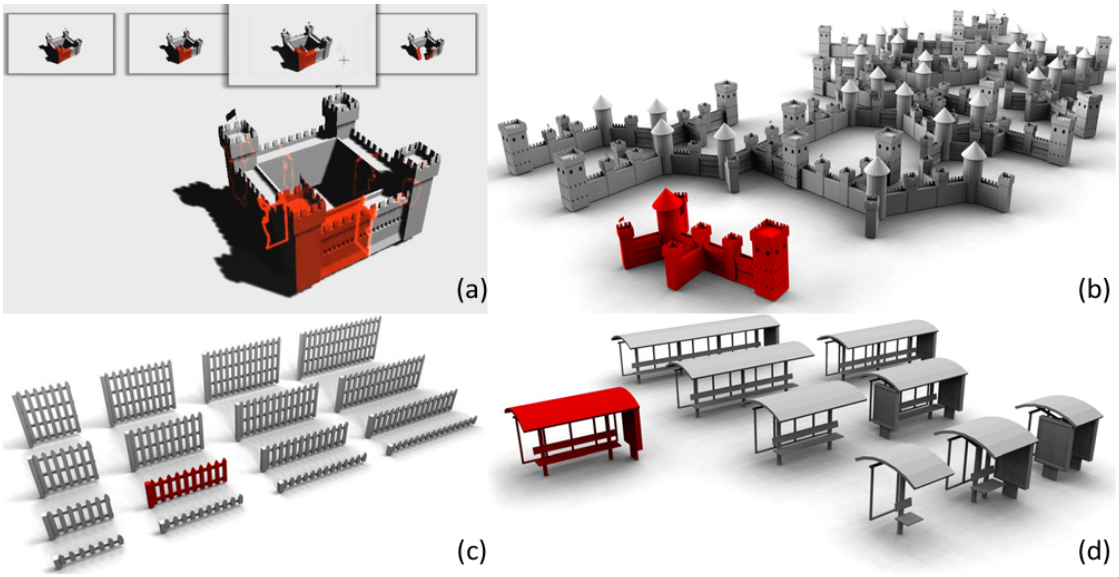



Figure 2.11: (a) When the user selects a docking site, the system displays possible variations at this site. (b) Automatically derived variations, (c,d) User-generated shape variations. Figure (a) adapted from the related video, Figure (b-d) adapted from [BWS10].

behaviour of a part. Depending on which rest positions are specified in the system, the way how a piece of geometry is changed can vary. For example, when just using one rest position for a straight street part, the object deforms "more elastic" as in the case where also a corner rest position is available (see Figure 2.12e).

Using the described methodology, the system supports four different shape operations. The method already described can be seen as a context aware replace operation. Changes in size can be achieved by stretching and shrinking (which is considered as one operation, since they are inverse). When the user drags unrelated geometry next to each other, the algorithm automatically tries to merge them together, forming a new connected part. In contrast to this, the cut operation allows the user to split up connected parts into unrelated ones. Note, that in both cases the context changes, and thus a replace operation might happen too. Figure 2.12 a-d shows these four operations on simple geometric parts.

This method can be seen as a mixture of two different model manipulation methods: On one hand, a grammar-like approach is used to decide which rest position is used for cutting and merging, while on the other hand a geometry manipulation is performed to adjust the mesh to a specific situation. Results of the "Mutable elastic model" algorithm are shown in Figure 2.13.

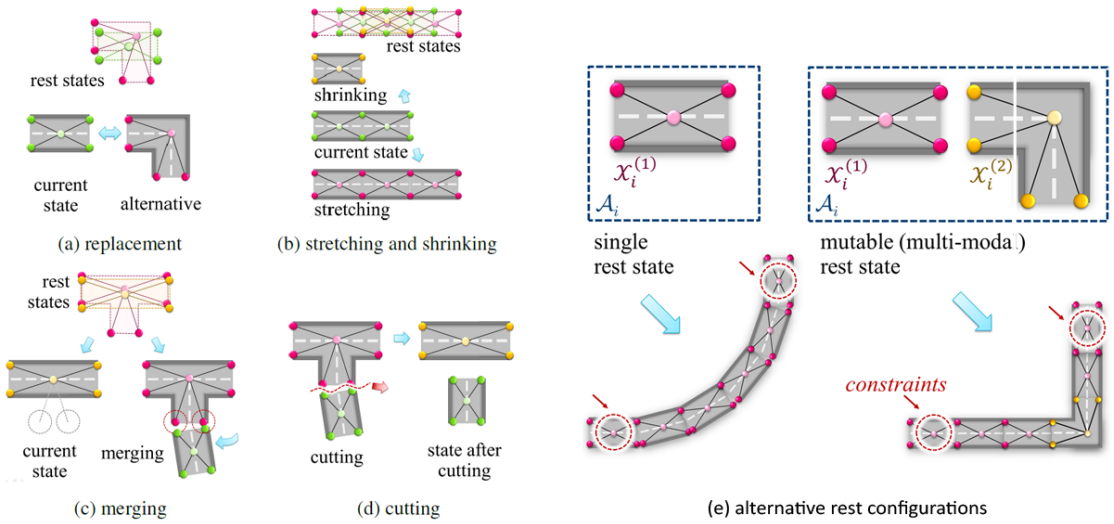



Figure 2.12: (a)-(d) Operations that can be applied by the user. (e) Models can behave differently depending on the available rest configurations. Adapted from [MWCS13].

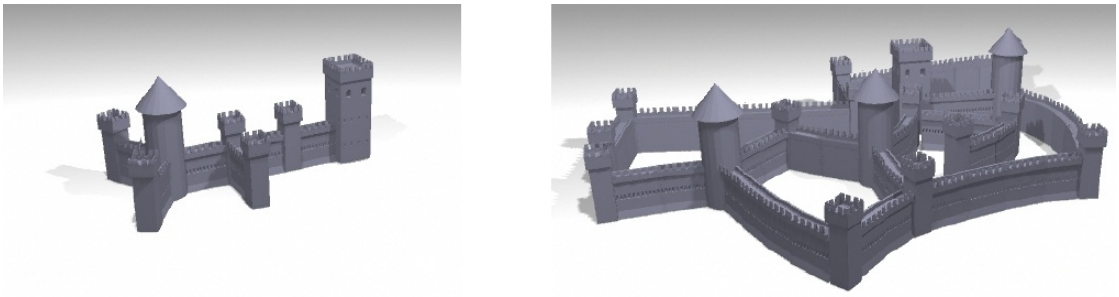


Figure 2.13: Original model (left) and variation of it produced by the mutable elastic deformation algorithm (right). Reprinted from [MWCS13].

## 2.4 Structure Adapting Systems

All of the algorithms presented until here are dealing with how models should be adapted to changes introduced by the user, but most of them are only modifying existing geometry without adapting the number of components present in the model. One example for a structure adapting algorithm has been presented in Section 2.3, but there are a lot more of them.

Bokeloh et al. [BWKS11] introduces the notion of *sliding dockers*: In many models there exists a number of regular patterns. When modifying such meshes, it would be appropriate to adjust the number of the elements in these patterns to better fit the changed situation. Again, the system consists of two phases: First the model is analysed to find symmetry groups. In case of sliding dockers, the system is not directly interested

in symmetry groups, but tries to find structures in these groups that can be described by a generating transformation. Assuming that a symmetry group $\mathcal{S}$ is already identified by the system, a subset $\mathcal{P} \subseteq \mathcal{S}$ can be taken, together with a transformation $T$ and a real interval $\mathcal{I} \in \mathbb{Z}$. If this subset fulfils the requirement $T^I(\mathcal{P}) \subseteq \mathcal{S}$ it is called a *discrete partial 1-parameter pattern*. Sliding dockers are then the patterns that can be decomposed into several subparts and thus allow for changes in the number of elements. Beside this discrete patterns, the model can also contain continuous partial 1-parameter patterns, which are defined similar to the discrete case, but with the interval $I$ being a subset of $\mathbb{R}$. These continuous features are found by using a slippage analysis, as already described in the work of Kraevoy et al. [KSSCO08]. For better understanding of these features Figure 2.14 shows examples of them, together with an example of a sliding docker group.

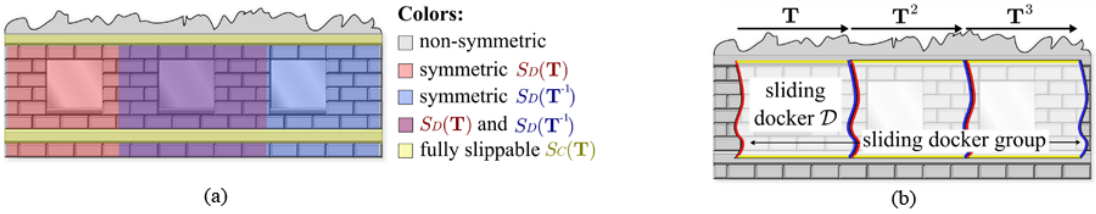



Figure 2.14: (a) Example of continuous and discrete patterns. (b) Sliding docker groups are composed of multiple sliding dockers with a common boundary. Adapted from [BWKS11].

The modification algorithm here is also based on a volumetric representation, but this time a transformation field is used to describe an elastic transformation for the underlying surface. This deformation field is found by defining an energy function, that is minimized by the optimal field $f$. The energy function used in this approach consists of four parts: The user constraints $E_u$, which describe the modification made by the user, and the base-regularize energy $E_r$ are used as in a standard elastic shape deformation approach. To account for discrete and continuous patterns in the model, the additional energy terms $E_d$ (discrete) and $E_c$ (continuous) are added. Using this energy terms the total energy is given as

$$E = E_u + \lambda_r E_r + \lambda_d E_d + \lambda_c E_c, \tag{2.1}$$

where the lambdas are algorithmic parameters, which adjust the influence of the different energy terms. For the definition of $E_u$ and $E_r$ have a look in the paper. The more interesting parts are the energies for continuous and discrete patterns: For both types the idea is similar: A constraint subspace has to be found, which describes the line or plane on which the pattern can be extended: Here, only the continuous case is shown, where for a part $P$ with two translational degrees of freedom $T_1$ and $T_2$ the constraint subspace for each point on the surface is given as

$$\mathcal{M}(y) = T_1^{\mathbb{R}} T_2^{\mathbb{R}}(y). \tag{2.2}$$

In this space two tangent vectors $t_1$ and $t_2$ can be found, which are used to define a quadric $Q_{\mathcal{M}}(y)$ that penalizes displacements that move the point $y$ away from the constraint space. The continuous energy function is then given by summing over the $k$ parts $\mathcal{P}_1$ - $\mathcal{P}_k$ and their corresponding points $q_j^{(i)}, j = 1, ..., n_i$:

$$E_c = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \left[ f(d_j^{(i)}) - d_j^{(i)} \right]^T Q_{\mathcal{M}}(q_j^{(i)}) \left[ f(d_j^{(i)}) - d_j^{(i)} \right]. \tag{2.3}$$

where $d_j$ is the original distance vector and $f(d_j)$ describes the transformed distance vector. The formulation for discrete patterns can be found similar to this one, for details please refer to the paper.

At all points in the distance field where a sliding docker is present, the energy system is modified to only support elastic motion, and an error quadric that constraints the motion to the direction of the docker group is used. The total energy system is then minimized to gain a motion field, which solves the constraint set best. In this field for each sliding docker the stretch factor is calculated, which gives the number of elements that should optimally be present in the docker group.

Together with some small adaptations to ensure that sliding dockers are never touched by the as-rigid-as-possible motion, this framework produces very nice results, and is one of the first systems that deals with repetitions in models. In addition, nearly all results of the previous algorithms can be achieved with this technique, but the newly added sliding dockers allow for a much wider range of possible modifications. Results produced by this algorithm can be seen in Figure 2.15.

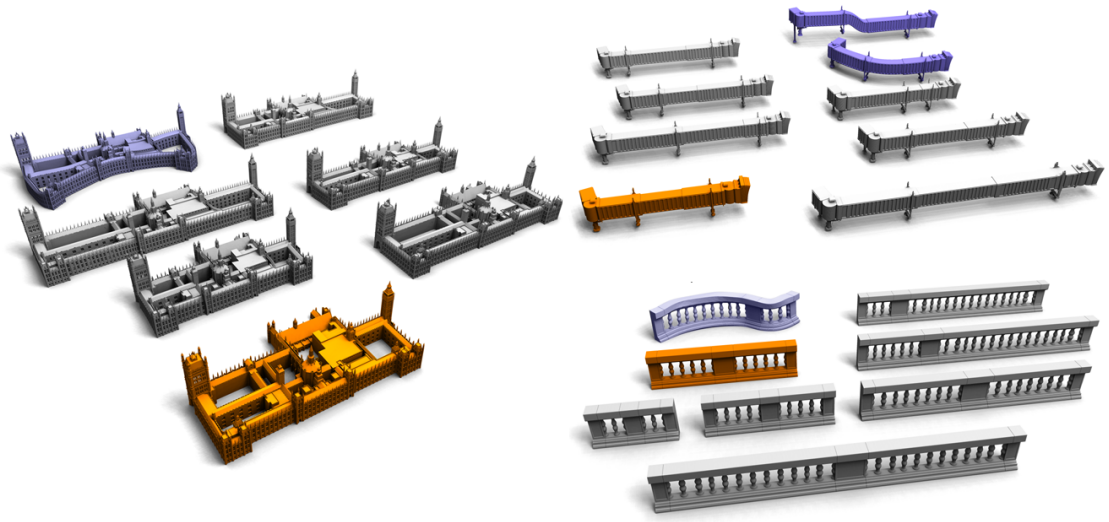


Figure 2.15: Results from the sliding dockers approach. Adapted from [BWKS11].

The follow-up work by Bokeloh et al. [BWSK12], called "Algebraic Model for Parameterized Shape Editing", improves the method of dealing with repetitions even further by

extending the theory to two-dimensional patterns and mixed patterns. In addition, the volumetric energy field approach is replaced by a new surface-based technique, which allows for a better formulation of the problem. To achieve this, the whole input geometry is segmented in patches of different type. The system identifies 6 different kinds of patches: rigid patches, which describe polygons where no resizing should happen, continuous line patches, which describe a line-geometry that can be stretched and discrete line patches, where repetitions of the input geometry can happen. Based on these three primitive cases, continuous area patches are defined, which describe a polygon where all edge-lines are continuous line patches. Discrete area patches work the same way, but this time discrete line patches are used to define the border. The last type of patch is the mixed-case patch, where continuous as well as discrete line patches are present. All these types of patches are shown in Figure 2.16 a-f.

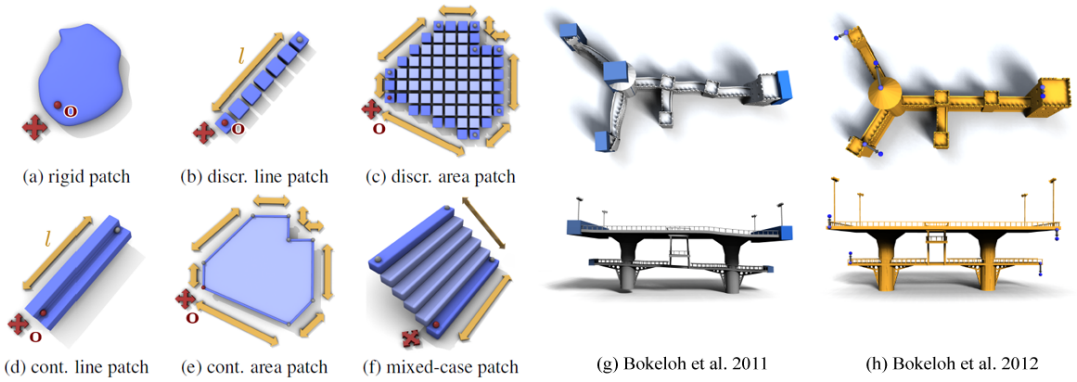



Figure 2.16: (a-f) The six patch types that can be used. (g-h) Comparison of the sliding dockers approach [BWKS11] with the patch approach [BWSK12]. Adapted from [BWSK12].

Between these patches in the input mesh, adjacencies are searched that are responsible for holding two surface patches together. All patches and the adjacencies are transformed into a large system of equations, where area patches are fully defined by the line patches that describe their boundary. In most cases, this system of equations will be heavily redundant and underdetermined, which yields a linear subspace that contains the solutions. Since not all solutions in this system of equations will be valid results, for example lines with negative length are not wanted, a number of inequalities has to be added to restrict the system to the desired solutions. Dealing with such large inequality systems is, as mentioned at the beginning, a big problem in real time applications. To overcome this, the problem is solved using null-space analysis, where the solutions can be described by an offset translation in relation to the basis of the kernel of the equation matrix. This results in general in a much smaller problem size and thus can be solved efficiently. When a valid solution for the current situation is found, all patterns are adapted. For continuous patches, this is simply done by scaling the geometry, while for discrete patches the number of elements is adapted according to the length of the patch. Looking at

the patch formulation for the input geometry, it can be noticed that since directions of line segments are never changed, the general structure of the mesh will be preserved. In contrast to the sliding docker algorithm (Bokeloh et al. 2011 [BWKS11]), where as-rigid-as-possible geometry adaptation was used, this results in a much cleaner solution without any distortions. This behaviour can be seen in Figure 2.16 g and h, where two models are shown with the same interactions performed. For example, the walls of the castle are no longer distorted when the patch technique is used. Other results for the Bokeloh et al. 2012 approach are presented in Figure 2.17, where especially the two dimensional discrete patch describing the windows of the apartment tower should be noticed, which was also not possible with the other approaches.

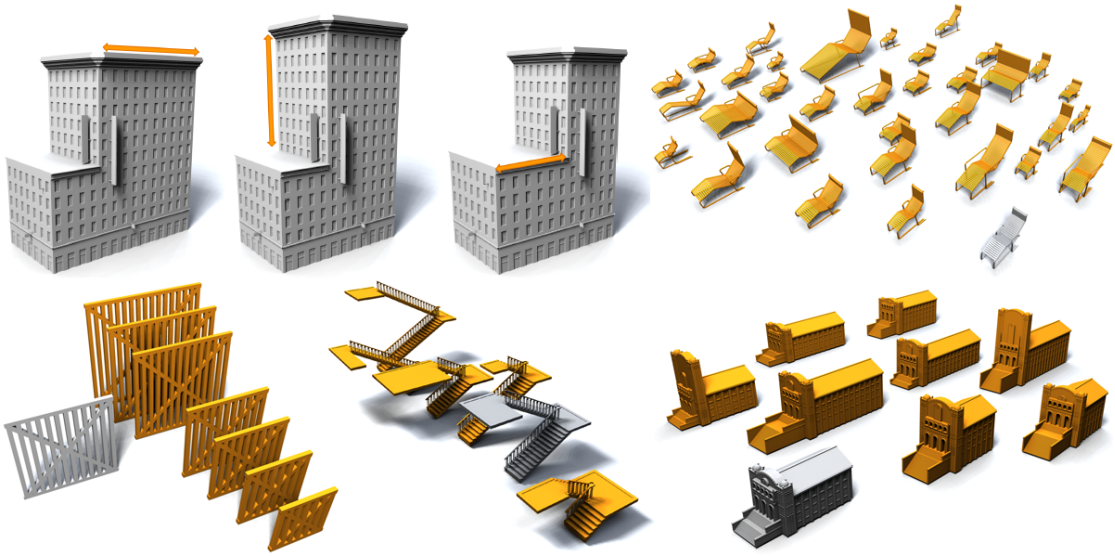


Figure 2.17: Results from the discrete and continuous patch approach. Adapted from [BWSK12].

A method that uses a similar approach for solving the energy system, but covers a totally different application field, is proposed by Schulz et al. [SSL+14]. This work shows how, based on a previously created template database, models can be designed such that they can be fabricated. The user can, using a design-by-example approach, combine parts of models from the database to new models. For our work, the most important part is how these parts or templates can be manipulated to allow a wider variety of producible shapes. Figure 2.18 (left) shows how the same combination of parts could be modified to produce different looking results, although the same constructive parts are used. Since the model here has not only the constraint to look correct, but also has to be physically stable, only the parameters of the parts can be adjusted. The parts are, in contrast to previous work, organised in a hierarchical data structure. At every none-leaf node in this tree, the constraints are given as the stacked vectors of the child nodes.

Since this system targets non-experienced users, there are a lot of methods imple-

mented that help the user to construct a model which can be build in real world. For example, when adding new templates to the model, the system automatically adjusts the size and snaps the new part to the existing model. Additional parts needed for the fabrication process are also added automatically, for example, hinges are automatically added when a door is placed in a shelf. With such structures, the system also checks if the door can be opened afterwards. When the user has finished the design of his model, a physical stability test can be performed to ensure that the model will be physically stable in real world. If this is not the case, the user can modify the object until it satisfies these constraints. To show that this analysis really works, Schulz et al. show that at least some of their models can be fabricated, which is shown in Figure 2.18 (right).

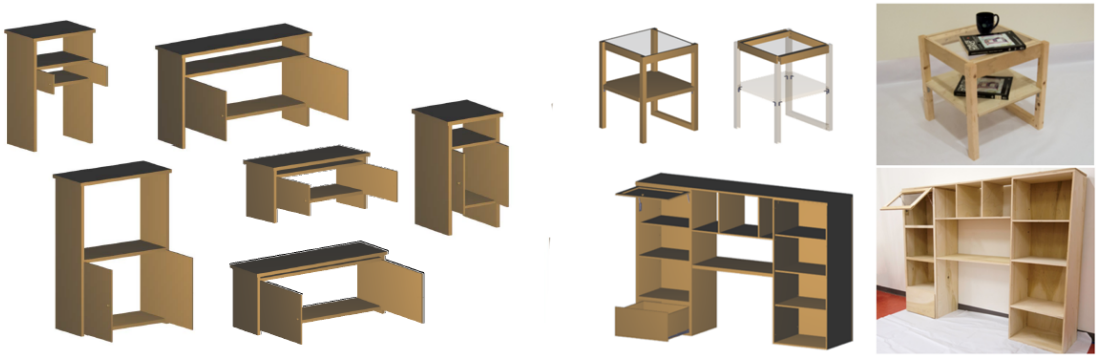


Figure 2.18: Left: The same constructive parts can be combined to a variety of different looking objects. Right: All models created with the "Design and Fabrication by Example" framework can be fabricated in real world. Adapted from [SSL+14].

CHAPTER 3

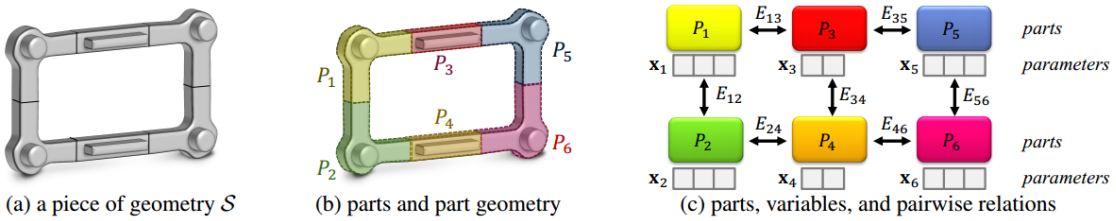
# Theoretical Foundations

This chapter gives an overview over the basic concepts that are used in this thesis. We first discuss how the general structure of a structure-aware model-manipulation application can be build up. After this, we give a short introduction to parametric curves, and give a brief explanation of symmetry and symmetry detection. At the end some theoretical background information on user studies is presented.

## 3.1 Structure Definition

When working with structure-aware model manipulation, sooner or later the question of what structure is will come up. Many man-made objects or natural forms will have a specific layout or form, which is mainly defined by its functionality and constraints. Thompson [Tho92] for example describes that horns and shells of animals are often spirally shaped due to their growth pattern. In man-made objects, such structures are often more visible, due to the constraints that arise from the costs and the manufacturing process.

A good definition for structure is given in the Eurographics 2013 STAR [MWZ+13]: Structure constitutes a collection of parts and how these parts are mutually related. This definition assumes that every object can be described by a number of *parts* that are not necessarily disjoint subsets of the whole shape. These parts are then connected by *relations*, which results in a unique composition. The combination of parts and relations is always unique to a special type of object, for example, many tables share the same structure. Beside parts and relations, there can also be a number of parameters that define the behaviour of the element they are attached to. Mitra et al. [MWZ+13] defines such parameters only on parts, but in this implementation we will also allow relations to be parametrized.




(a) a piece of geometry $\mathcal{S}$ (b) parts and part geometry (c) parts, variables, and pairwise relations

Figure 3.1: An example of a structured shape (a), which is decomposed into parts. Each part controls a portion of geometry (b), which we call part geometry. The parts (c) have parameters and a constraint energy that controls the parameters as well as the decomposition itself. The example shows binary relations (a connectivity graph). In this particular case, it has a Markovian structure (only neighbors interact). Reprinted from [MWZ+13].

### 3.1.1 Parts

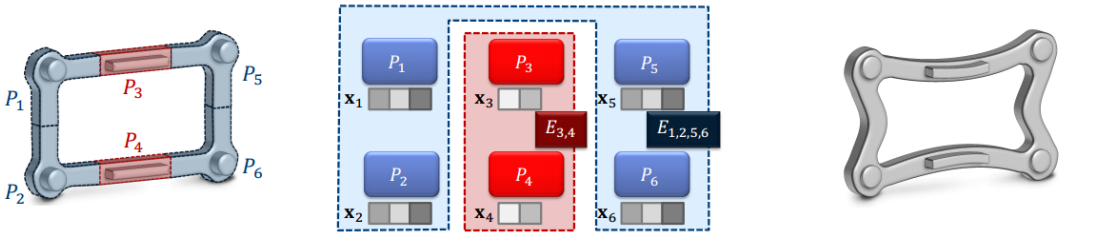
As described above, a model is composed of several parts. A part can be seen as a semantic abstraction that controls the behaviour and thus the visual appearance of a subset of the model. According to this definition, parts are not necessarily disjoint, but due to the part detection used (see Section 4.3), they are always disjoint in our application. In contrast to the traditional definition of a part as a surface patch after segmentation, parts can here be seen as an abstraction of a shape's region. This is shown in Figure 3.1, where a shape is decomposed in several parts. Depending on the application, parts can be approximated by different representations: Xu et al. [XLZ+10] are using bounding boxes, Zheng et al. [ZFCO+11] are additionally using generalized cylinders and spheres. It is also possible to use volumetric approximations [Sha08] or a general shape space.

On each part a set of parameters can be defined, which specify how the part should react and appear. For many applications, these parameters can be automatically determined by the choice of parts. Some applications, and also our framework, rely on the user to specify additional parameters manually. This can range from providing semantic parts as part of an input template, to our use case, the specification of how parts should react to changes. [MWZ+13]

### 3.1.2 Relations

Relations are used to describe how parts are associated to each other. In general two types of relations are considered: Binary relations and higher-order relations. Binary relations are always linking two parts together (as also seen in Figure 3.1). For example, physical connectivity could be expressed by such a binary relation. In contrast, higher-order relations are describing a correlation between a larger number of elements. Another option, shown in Figure 3.2, is to group all elements that should behave similarly. This leads to symmetrical results when manipulating the mesh (Figure 3.2c).

For each relation (binary and higher-order) an energy can be defined that is zero




(a) partitioning into parts (b) higher order graph (e.g, symmetries) (c) symmetry-preserving deformation

Figure 3.2: Higher-order relations can be used to group parts with similar behaviour. The four corners are related by a 4-ary symmetry energy (b). When modifying this object with a structure-preserving shape deformation algorithm, the result will always be symmetric. Reprinted from [MWZ+13].

for a valid structure. Depending on the application, these constraints can be treated as hard constraints (all energies have to be zero) or as soft constraints (which results in a minimization problem). [MWZ+13]

### 3.1.3 Detection of Parts, Relations and Parameters

Now that we know how parts, relations and parameters are related, the next task is to identify them in an input model. Assuming, that information about the parts (and maybe about the parameters) have to be obtained before relation detection makes sense, we start with the part detection. Methods for detecting parts can be classified in three categories [MWZ+13]:

#### User Specification

When no prior knowledge of the model exists, one of the simplest solutions is to let the user identify the parts. This can be done, for example, by selecting subsets of the input model, where each subset is then used as a part. Another possible method is, that the user creates primitive objects, which abstract the real data. In many cases it is also possible to take information about the model into account. When working with hand-modelled objects, the scene hierarchy of the input-file might give some information about the intention the designer had.

Manual parameter specification is often used in procedural modelling, where the user has to specify the whole scene via a shape grammar. In this context specifying parameters by hand is the natural interaction. But also in mesh-modelling, parameter specification by hand takes place, for example by assigning a specific color to parts of the mesh. Our application will, as described later, also rely on some user specified parameters.

#### Fixed Model

In this class of methods a fixed segmentation model is constructed, which allows the system to identify parts automatically using this model. iWires [GSMCO09], for example, identifies significant edges, which is exactly such a model. Identification of objects due to symmetries or other geometric properties assumes also the use of a fixed model.

Parameter detection using a fixed model happens very often. The arrangement of parts will in most cases happen by using homogeneous transformations. When approximation objects are used, the specification of the sizes these boxes and models have happens in a parametric model. We will see later, that our user interactions will fall in this category, since we allow the user only to manipulate bounding boxes.

#### Learning from Data

In contrast to the previous approach, no prior knowledge is used here. The parts and parameters are learned here by a machine learning technique, which derives a meta-model from a number of training objects. As always in machine learning, the user can either interact with the learning system (supervised learning) or the system works without any additional information, for example, by detecting clusters and finding subsets of the models that have a similar structure.

#### Identifying Relations

Identifying relations is challenging, since this strongly depends on the type of relation and what it should express. A relation can be of unary rank (i.e. symmetry connects the part to itself), or up to n-ary rank. Often used relations are the ones that connect two parts, which gives a general graph structure, thus probabilistic graph models can be used. When the relation describes contacts between parts, it can also be expressed as a Markov-random field (for more details see [MWZ+13]).

The basic concepts for part detection can also be applied to relation identification: The user could specify the relations, for example in a modelling tool by specifying a hierarchy, or by using the same mesh in multiple places, thus identifying a symmetry. When dealing with physical constraints, like connectivity or stress factors, a priori models can be used. Bokeloh et al. [BWSK12] uses a machine learning algorithm to identify repeatable patterns. Another example for learning base relation classification is the whole field of inverse procedural modelling, where the grammar has to be derived from an input model automatically.

## 3.2 Symmetry and Symmetry Detection

Mitra et al. [MPWC13] define symmetry as follows:

> *A symmetry preserves a certain property (e.g., geometric similarity) of an object under some operation applied to the object. This notion of invariance is formalized in an elegant branch of mathematics called group theory. In the*

*context of geometry, we will consider geometric transformations as the symmetry operatons, such as reflections, translations, rotations, or combinations thereof.*

Two objects M and N are symmetric, if there exists a transformation $T$ such that $N = T(M)$. In the mathematical context, all the symmetry operations that can be applied to a shape form the structure of a group. For a symmetry set $\mathcal{S}$ of transformations, it has to be proven that several axioms are fulfilled in order to form a symmetry group (to be more concrete, every set that satisfies these properties is a group). The group operation that is used in this definition is the composition operation.

The axioms are the following: First, there has to exist an **identity element**. When working with symmetries, the identity transformation $I$ is always a valid transformation, since every object stays unchanged when applying an identity transformation. Beside the identity element, there has to exist an **inverse element**, meaning that for every transformation $T \in \mathcal{S}$ the inverse transformation $T^{-1}$ has to be part of $\mathcal{S}$. For the inverse transformation $T^{-1}T = TT^{-1} = I$ has to hold. Each mathematical group has to have a **closure**, thus when a object is symmetric under two transformations $T_1$ and $T_2$, than the combined transformation $T_1T_2$ must also be part of the group. The last property, that has to be checked, is the **associativity**. In terms of transformations, this is easy to proof, since matrix products (and thus all transformations that are represented by matrices) are by definition associative (see [DGKP08]).

For some simple objects, we can exemplary define such a symmetry group. When looking at an equilateral triangle, it can be seen, that there exist three unique rotations, around 120°, 240° and 360° = 0° (which is also the identity element in this group). In addition to these rotations, there exist three reflections, one along each altitude. Together, this six transformations form the dihedral group $D_3$, that is shown in Figure 3.3a. Another group that is very similar to the $D_3$ group is the symmetry-group on a five-pointed star. This group $D_5$ (Figure 3.3b) consists of repeated application of 72° rotations and reflections. Beside the dihedral groups $D_n$ on regular n-gons, there exists a large number of other groups. Examples are the cyclic group $C_n$ (see Figure 3.3c for the $C_3$ group) that is generated by rotations around $\frac{360°}{n}$. In 2D, the infinite group $O(2)$, which is the group with elements that stay symmetric under rotations around arbitrary angles and reflections (in 2D the only object is the circle, see Figure 3.3d), is a super group of all the previous described ones. [MPWC13]

Symmetry is a long used concept in art and architecture: The probably most well known example for symmetry in arts is the "Vitruvian Man" by Leonardo da Vinci (around 1490), where the symmetries of the human body are shown (Figure 3.4 right). From the theoretical point of view, the ornaments in the spanish castle Alhambra have been really important. These ornaments were studied around 1830 by Owen Jones and Lewis Vuillamy. Later in 1958, Jones published his probably most important work "The Grammar of Ornament" [Jon68], which described patterns from all epochs all over the world and defines key principles for the decorative art, also showing how symmetry is used in this field (see Figure 3.4 left). Building up on this work, a mathematical classification for 2-dimensional patterns, which are repeatable in one direction, are given

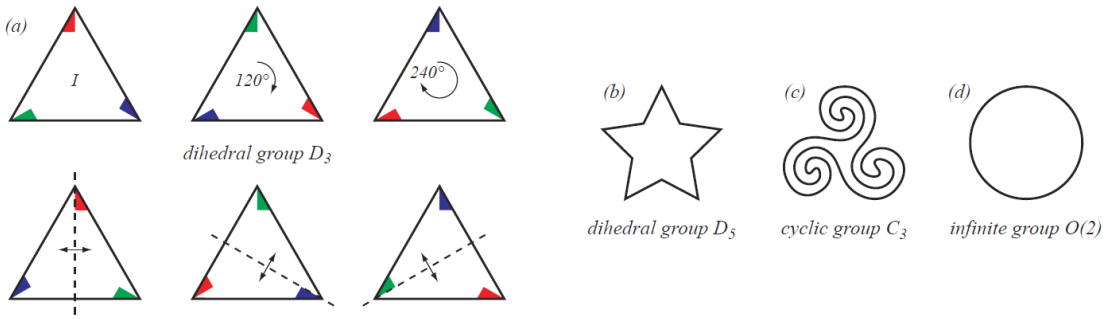



Figure 3.3: The dihedral group $D_3$ represents the symmetries of the equilateral triangle (the colored flags are added to illustrate the transformation), while $D_5$ is the symmetry group of the five-sided star. The triskelion has three rotational symmetries, but no reflectional symmetries and is represented by the cyclic group $C_3$. All of these finite point groups are subsets of the isometry group $O(2)$, which represents the symmetries of the circle. Adapted from [MPWC13].

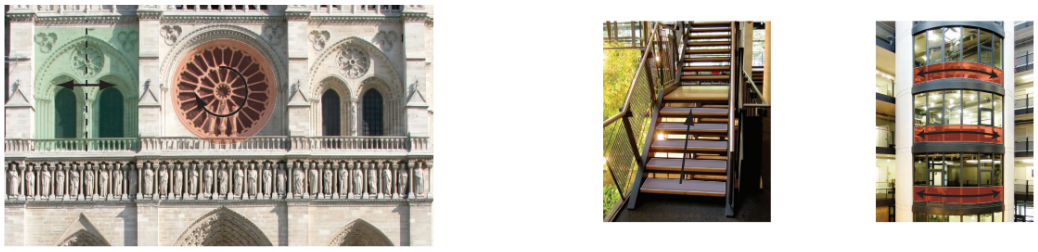
by the Frieze groups, where seven basic patterns are identified. When repetition can happen in two directions, this leads to the definition of wallpaper groups [Fed91], where seventeen distinct types can occur.


Figure 3.4: Left: Two plates with symmetric ornaments as identified by Jones et al. in "The Grammar of Ornament", adapted from [Jon68]. Right: The "Vitruvian Man" by Leonardo da Vinci. Adapted from Wikipedia.

### 3.2.1 Global and Partial Symmetries

Symmetries can be classified according to the type of symmetry used: We differentiate here between global and partial symmetries, where global symmetries try to map the whole object to itself. This leads in practice to a lot of simplifications, since we do not have to solve the segmentation problem. Global symmetry detection algorithms make also use of the fact, that for these symmetries the centroid of both objects has to be the same. Additionally, the centroid is also the only center of rotation used. Partial symmetries are searching for parts of the object which are symmetric. If we consider M as a subset of the object, then we can use the definition of the symmetry-groups as before, thus the whole symmetry detection is a segmentation followed by a global symmetry detection algorithm. One problem with partial symmetries is, that they are often not forming a complete symmetry group as defined above (Figure 3.5a). When looking at the steps of a staircase (Figure 3.5b, c), which are in general partial symmetric, we can for example find a transformation that maps one step to the next one. But we are not able to find a transformation, except of the identity transformation, which maps the whole set of steps to itself. This leads us to an adapted formulation for partial symmetries: A shape $S$ is partial symmetric under a transformation $T$, when there exist two subsets $M_1, M_2 \in S$, which fulfil the property that $M_1 = T(M_2)$. From this definition it can be seen, that global symmetries are just a subgroup of partial ones, to be exact, global symmetries are the one where $M_1 = M_2 = M$.

To summarize: Partial symmetries can describe much more cases than global symmetries, but they might not fulfil the criterions, which are required for a group structure, although we could fix this problem by also taking repetitions to infinity of the pattern into account.



*(a) complete symmetry group on parts of a shape*

*(b) partial translational symmetry*

*(c) partial rotational symmetry*

Figure 3.5: Examples for partial symmetries. Reprinted from [MPWC13].

### 3.2.2 Imperfect Symmetries

Until now, we only considered objects as symmetric, if the matching is perfect. Since we are dealing in this work mainly with man-made triangle meshes, it can happen due to imprecise modelling, or due to sampling artefacts of the triangulation process, that objects, which are considered symmetric by a user, are not symmetric in the mathematical

sense. Thus a definition for these *imperfect symmetries* is required. According to Mitra et al. [MPWC13], we can achieve this by transforming the original equation $M_1 = T(M_2)$ into an optimization problem, using the distance $d(M_1, T(M_2))$ between the object and the transformed object. Imperfect symmetry is given when

$$d(M_1, T(M_2)) \leq t. \tag{3.1}$$

If the threshold $t$ is set to zero, we get again the definition of perfect symmetry. The simplest form of a distance measurement would be to take the mean of the distances between each point in $M_1$ and the nearest neighbour in $M_2$. Many authors proposed other distance functions, for example, Zabrodsky et al. [ZPA95] replaced the distance between the nearest points by the distance between a point and the nearest point on the surface to the other object (in the case of meshes, this results in finding the nearest face for a given point). Another measurement, which is mainly used in computer vision, but can also be applied to symmetries, is the Hausdorff distance:

$$d(M_1, T(M_2)) = \max \sup_{x \in M_1} \inf_{y \in T(M_2)} \|x - y\|, \sup_{y \in T(M_2)} \inf_{x \in M_1} \|x - y\|. \tag{3.2}$$

The major problem with all this measurements for distances between meshes is, how the threshold should be defined. A larger threshold will allow for objects to be symmetric under a higher approximation error, but may instead declare parts as symmetric although they are different.

Two things should be noted: First, that at the moment where imperfect symmetries come into play, the algebraic structure is not a group any more. The reason for this is given by the fact, that the set is not closed with regard to composition any more, since $d(M, T_1(M)) < t$ and $d(M, T_2(M)) < t$ can both be below the threshold, while the composition of both $d(M_1, T_1T_2(M_2))$ can be above it. The second point is, that the quality and the possibilities that symmetries can describe strongly depends on the space that is used. Euclidean space (and the so called *extrinsic symmetries*), which are also used in our work, are well suited to rigid body transformations, but when for example working with humans or animals and their non-rigid behaviour (e.g. Figure 3.6) a better-fitting problem space has to be used. The group of such symmetries is called *intrinsic symmetries*.

### 3.2.3 Symmetry Detection

For the detection of symmetries in arbitrary meshes or in images, a lot of methods have been proposed. Global symmetries can for example be found using the iterative closest point methods by Besl et al. [BM92] and Chetverikov et al. [CSSK02]. For every point in a model the corresponding point in the other model is searched and the motion that minimizes the distances between corresponding points is calculated. The correspondence for a point is, in the simplest case, determined by searching for the point with the shortest distance. This can be improved by using the shortest distance to the surface of the other model. The ICP algorithm is explained later in Section 4.3.3, since our implementation

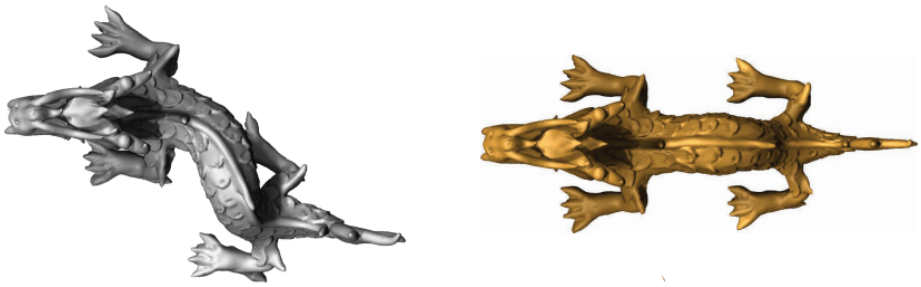


Figure 3.6: Example where the extrinsic symmetry changes under a non-rigid transformations. Reprinted from [MPWC13].

is based on this technique. The main limitation of the ICP-technique is, that it can only search for global symmetries.

Another approach, proposed by Mitra et al. [MGP06], searches for local shape features in the input 3d geometry (Figure 3.7 left). These features are then matched against each other, and if two features are matching, the corresponding reflection line (in case of reflective symmetry) is marked in the so-called *transformation space*. When multiple point pairs correspond to the same reflection line, these features build a cluster in the transformation space (Figure 3.7 middle). These clusters are found by a mean-shift algorithm, which returns the significant symmetries (Figure 3.7 right). For other types of symmetries, the algorithm works basically the same, although the transformation space might be of higher dimension. Since the marked values in transformation space do not have to match exactly due to the clustering algorithm used, this technique can, beside of partial symmetries, also find approximative symmetries.

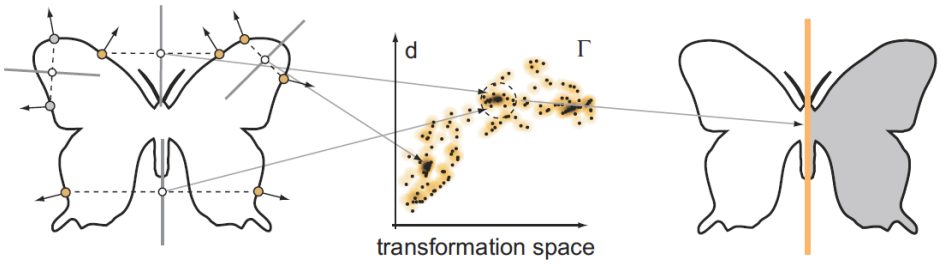



Figure 3.7: Basic transformation space algorithm for reflective symmetries: Matching shape features are searched (left), and transformed to transformation space (middle). Clusters in this space correspond to symmetries in the model (right). Reprinted from [MGP06].

Bokeloh et al. [BWKS11, BWSK12] are using a RANSAC-based detection schema. Generator transformations are sampled randomly for mesh features and the number of other features that belong to this transformation are counted. After a high number of repetitions (up to 500 times,) only the transformation with the highest number of

corresponding features is kept. This transformation is then removed from the feature set, and the algorithm is started again on the reduced set. Using this technique, it is possible to find global and local symmetries in a mesh, although it is only used for translational symmetries here. Jain et al. [JTRS12] improved this technique and extended it to reflective and rotational symmetries.

Beside the methods described in detail here, there exists a large number of other possible algorithms: Gelfand et al. [GG04] uses slippage analyses to determine symmetries, Gal et al. [GCO06] shows, how geometric hashing can be used in this application field. Pauly et al. [PMW+08] propose an approach that also uses transformation spaces to detect features, but deals explicitly with outliers and missing data, which arise when working with point clouds. A conceptually different method for symmetry detection is shown by Berner et al. [BBW+08], where feature graph matching is used as an input to an ICP-based segmentation method. For more details about symmetry detection methods, have a look at Mitra et al. [MPWC13].

### 3.2.4 Further Reading

Since the topic of symmetries has a long term history, we can give here only a short overview on this topic. For example, there is a strong relationship between geometric symmetries and the theory of Lie groups. We want to refer the interested reader for example to the theoretical work by Helgason [Hel79] for more information about this theory. Also the book "Geometric Symmetries", by Lockwood and MacMillan [LM78], might give deeper knowledge about the theory. The last work that should be mentioned here, is the state-of-the-art report by Mitra et al. [MPWC13], which presents beside the classical theory also a very good overview on application fields.

## 3.3 Parametric Curves

Since we want to extend the formulation of symmetries by using parametric curves to describe the local arrangement of parts, we first have to define what parametric curves are: Many functions and curves can be written in the form $x = f(y)$ or $y = f(x)$, but when thinking for example about a circle, we will see, that it is impossible to find such a representation for it. The equation that defines a circle is given by

$$x^2 + y^2 = r^2, \tag{3.3}$$

which we could solve for x resulting in the equation $x = \pm\sqrt{r^2 - y^2}$. But when looking at this in detail, it can be noticed, that these are two separate equations, one defining the top half of the circle, and a second one that gives the bottom half. Beside this type of curves, there exists also a large number of curves that cannot be expressed in terms of $x$ and $y$ only.

For some of these equations, we can solve the problem by using so called *parametric equations*, that define both $x$ and $y$ in terms of a new variable $t$, which is called parameter.

Using this formulation, we can define the circle by the following equation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = f(t) = \begin{pmatrix} sin(t) \\ cos(t) \end{pmatrix}, \tag{3.4}$$

where t is going from 0 to $2\pi$. Solving the equation for a special $t$ gives a point in space. A parametric curve is given as the set of points which are calculated by $f(t)$ for each possible value of $t$ [PT95]. In our application, we are using three different parametric curve types. One is the circle that was already explained above, the other ones are Bezier curves and NURBS.

### 3.3.1 Bezier Curves

Bezier curves have a long history in computer graphics. They were independently invented in the beginning of the 1960's by Pierre Bézier and Paul de Casteljau. Both worked in the field of computer aided design, and used this representation of curves for their CAD applications. Similar to polynomials, these curves can be defined for any degree. A Bezier curve of degree $n$ is defined by using a control-polygon with $n+1$ control points $P_0, P_1, ..., P_n$ as follows:

$$c(t) = \sum_{i=0}^{n} B_{i,n}(t) P_i, \tag{3.5}$$

where $B_{i,n}(t)$ is the i-th Bernstein polynomial of degree $n$ and $t$ has to be in $[0, 1]$. Bernstein polynomials can be defined recursively as

$$\begin{aligned} B_{i,n}(t) &= (1-t) \cdot B_{i,n-1}(t) + t \cdot B_{i-1,n-1}(t) \\ B_{0,0}(t) &= 1, \end{aligned} \tag{3.6}$$

with $B_{i,n}$ equals zero for $i < 0$ or $i > n$. It is also possible to calculate the Bernstein polynomial directly using binomial coefficients:

$$B_{i,n} = \binom{n}{i} t^i (1-t)^{n-i}. \tag{3.7}$$

Some examples for Bezier curves with varying degree are shown in Figure 3.8.

Bezier curves belong to the class of curves where the control points have global influence, which means that modifying one control point will change the whole curve. This property is in general not wanted, but for our application case we will see, that this is highly desirable. The major problem with Bezier curves is, that they cannot express the full spectrum of possible curves in space, for example it is impossible to define a circle using a Bezier curve. [PT95]

### 3.3.2 NURBS

NURBS, or Non Uniform Rational B-Splines, are a more complex class of curves. To be more concrete, they are a superclass to Bezier curves. They have also been used very

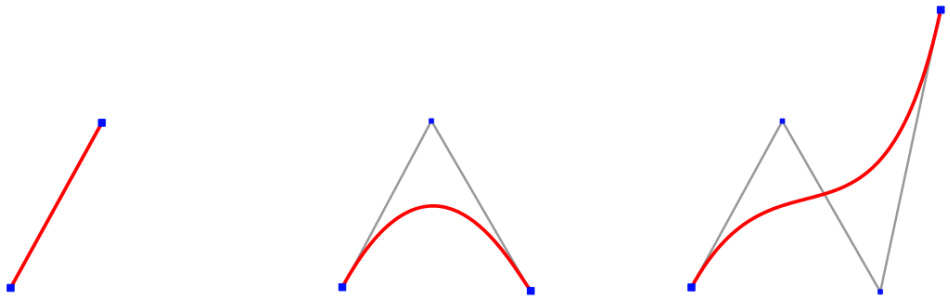


Figure 3.8: Bezier curves of degree 1, 2 and 3. Reprinted from Wikipedia.

successfully in CAD applications and, in contrast to Bezier curves, the control points only have local influence. Due to additional weighting factors, these curves can express many more space curves, including circles (see Figure 3.9a). [Pie91]

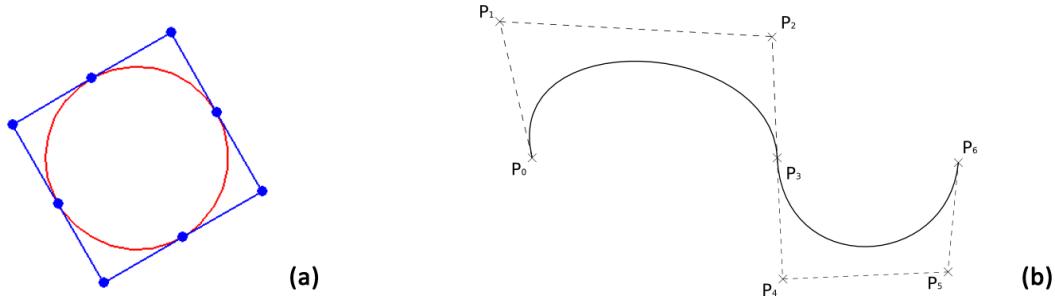



Figure 3.9: (a) Using NURBS it is possible to create a circle, (b) Two curves with 4 control-points can be joined together, where $P_0$ - $P_3$ define the first curve and $P_3$ - $P_6$ the second one. (a) Reprinted from staffweb.ncnu.edu.tw, (b) Adapted from andrewharvey4.wordpress.com.

Mathematically, NURBS are defined by the following equation:

$$c(t) = \frac{\sum_{i=1}^{k} N_{i,n} w_i P_i}{\sum_{i=1}^{k} N_{i,n} w_i}, \tag{3.8}$$

where $P_i$ is again the i-th control point, $w_i$ is a weighting factor for $P_i$ and $N_{i,n}$ is a b-spline basis function of degree $n$. Basis functions define in which range control points should influence the curve, and which influence they have at a given parameter. Examples for basis functions of degree 1 and 2 are shown in Figure 3.10.

In many application scenarios, it can be helpful not to model the whole model by using just one curve. Instead, multiple NURBS can be fitted together by using the last control point of the first curve as a starting point for the second curve. This can be done, since NURBS (and also Bezier curves) will always go through the first and the last point. In Figure 3.9b, an example for a stitched NURBS curve is shown, where control points $P_0$

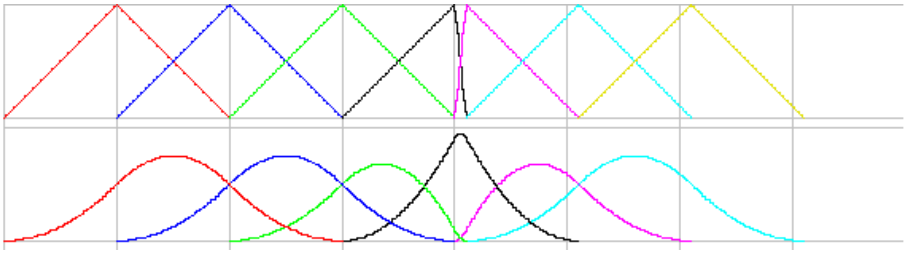


Figure 3.10: Linear (top) and quadratic (bottom) B-Spline basis functions. Reprinted from Wikipedia.

- $P_3$ define the first curve and $P_3$ - $P_6$ the second one. When stitching together multiple curves, we want to avoid in most cases that the stitching point is visible to the user. The way how this can be achieved, is mathematically defined by the concept of geometric continuity [Wer94]. Continuity of a given degree $n$, commonly written as $C_n$, specifies, that the n-th derivative of two curves is the same at the docking site. In practice, three levels of continuity are used:

$C_0$ **continuity** (also called positional continuity) defines, that the starting position of the second curve is equal to the end position of the first curve. As stated before, this is easy to achieve with NURBS. In this definition only the location plays a role, due to this, $C_0$ continuity is also given when the two curves meet at an angle.

$C_1$ **continuity**, or tangent continuity, rules out these sharp edges, by specifying that the vectors at the joining location of both curves have to point in the same direction and are parallel. When thinking in terms of illumination, using $C_0$ continuity will break the highlights due to the sharp edges resulting in the mesh. Using $C_1$ continuity, the highlights will be continuous, but may appear stretched. Since it will look natural in most cases, it can be sufficient for many applications to only require tangent continuity.

$C_2$ **continuity** solves the problem of highlights getting stretched, by not only requiring the two vectors to be parallel. Additionally, the two tangent vectors also have to have the same length, which leads to the same scaling of the highlights on both sides, giving a perfectly smooth curve.

In our application, we require all curves to have at least $C_1$ continuity, since we are requiring tangent directions of the curve, but not tangent length. Figure 3.9b is also an example for an at least $C_1$ continuous curve.

### 3.3.3 Spaces on Curves

On every space curve $c(t)$, we can according to Kreyszig [Kre13] define two vectors of interest at every point: The **tangent vector** describes the forward direction of a curve. When thinking about the curve as the path that a particle moves in space, the speed

of this particle at the point specified by the parameter $t$ is given by this tangent vector. Mathematically speaking this vector is given by the following formula:

$$c'(t) = \frac{d}{dt}c(t). \tag{3.9}$$

The normalized tangent vector $e_1$ is then given as

$$e_1 = \frac{c'(t)}{\|c'(t)\|}. \tag{3.10}$$

The second important vector is the **normal vector**, or sometimes called curvature vector. It is defined as

$$\begin{aligned} e_2(t) &= \frac{\overline{e_2}}{\|\overline{e_2}\|} \\ \overline{e_2} &= c''(t) - \langle c''(t), e_1(t)\rangle e_1(t) \end{aligned} \tag{3.11}$$

and indicates how much the curve diverges from being a line. The normal vector is always perpendicular to the tangent vector and spans together with the tangent vector a plane, which has a second order contact with the curve. This plane is called *osculating plane*. Figure 3.11a shows the tangent vector and the normal vector of a curve in 2d. The vector orthogonal to both, the tangent vector and the normal vector, is called **binormal vector**. In 3-dimensional space it can be calculated via the cross-product of these vectors:

$$e_3 = e_1 \times e_2. \tag{3.12}$$

Together, these three vectors define the **Frenet frame** in $\mathbb{R}^3$, an orthogonal moving frame on a curve, which is one of the major tools in differential geometry (see Figure 3.11 for an example of a Frenet frame moving along a curve). In $\mathbb{R}^n$, the Frenet frame consists of n orthogonal vectors, which can, according to the gram-schmidt orthonormalization process be calculated as follows:

$$\begin{aligned} e_j(t) &= \frac{\overline{e_j}}{\|\overline{e_j}\|} \\ \overline{e_j} &= c^{(j)}(t) - \sum_{i=1}^{j-1} \langle c^(j)(t), e_i(t)\rangle e_i(t). \end{aligned} \tag{3.13}$$

The initial value for $e_1$ is defined as in Equation 3.9. When working in 3d space, this formulation can be collapsed exactly to Equations 3.10 and 3.11.

The usage of the Frenet frame is also the reason why we require all our curves to be at least $C_1$ continuous, in order to have a smooth Frenet frame without discontinuities. Since we are only using the normalized tangent vector, $C_2$ continuity is not required.

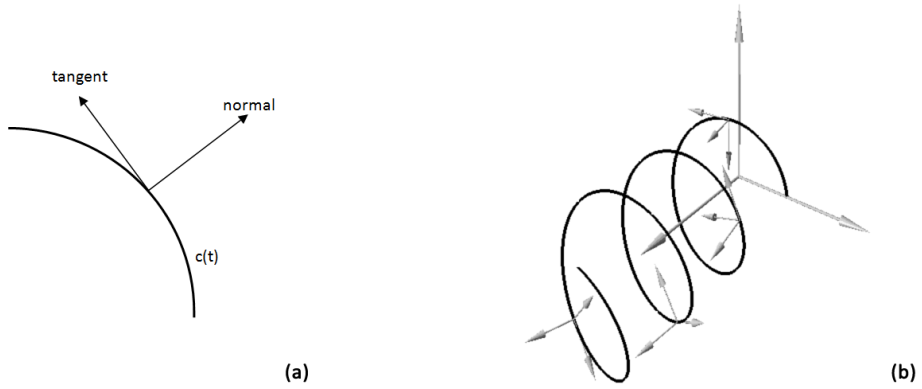



Figure 3.11: (a) Tangent and normal vector of a 2D-curve, (b) Frenet frame moving along a space curve. Reprinted from vjticontrolin.cluster2.hostgator.co.in.

### 3.3.4 Length of Curves

As already mentioned in the previous section, the tangent vector $c'(t)$ defines the velocity vector that a particle has when moving along the curve. Jia [Jia13] shows how this vector can be used to calculate the length of a curve: When looking at the curve, it can be noticed, that the curve between the parameters $t$ and $t + \Delta t$ can be approximated by a straight line. Thus the length in the interval $[t, t + \Delta t]$ is given by

$$\|c(t + \Delta t) - c(t)\|, \tag{3.14}$$

which can be approximated again using the tangent vector, giving us $\|c'(t)\|\Delta t$. The length is here calculated by splitting up the curve in a large number of segments, and then approximating the length in all these segments. The error made by this calculation will decrease when $\Delta t$ gets smaller, thus we would get the exact length when $\Delta t$ tends to zero. Using this knowledge the length of a curve $c(t)$ between the parameters $a$ and $b$ can be defined as

$$\int_a^b \|c'(t)\| dt. \tag{3.15}$$

## 3.4 Quantitative Usability Testing

Usability testing is being used for a long time in computer science, although in computer graphics the user is often neglected. Since our work is not only about the beauty of our framework, but also about how well it fits to the desired target group, a user study is the good indicator for that. User-based testing, as done here, is an empirical study to test if a product is usable by real users. This is achieved by letting them perform real tasks with the application. There are several parameters that can be tested by such a study: **Learnability** indicates how well users can cope with the system when they work with the product for the first time. **Efficiency** measures how fast users can handle the

system, when they are already aware of how the application works. In addition, **errors** can be measured by the system. Here not only the fact that the user made an error is interesting, most of the time it is more important to see if users are aware that they made a mistake, and if they can solve it or recover from it after the error happened. Another parameter is how pleased users are when using the framework, which can be called **satisfaction**.

When a new application is developed, user testing should be done as often and as early as possible. The reason for repeating such studies is, that in the first test the users will be distracted by the worst errors, thus minor ones will probably not be noticed at all. The better method is to start with a first study, which includes a few users, and then reworking the application and fix the problems encountered. Afterwards, a second user test is done, again with a few users, since the major problems are already solved, these users will now focus on the minor problems that would otherwise be neglected. This workflow can then be repeated until the application satisfies the desired quality criterias. According to Nilsen [Nil93], the best results in nearly all situations can be achieved when each test group consists of 5-8 users, since with this group size already 80-90 percent of the usability issues can be found. When testing more users at the same time, the problems-found to number-of-test-users ratio will decrease drastically, leading to not so well used resources (see Figure 3.12).

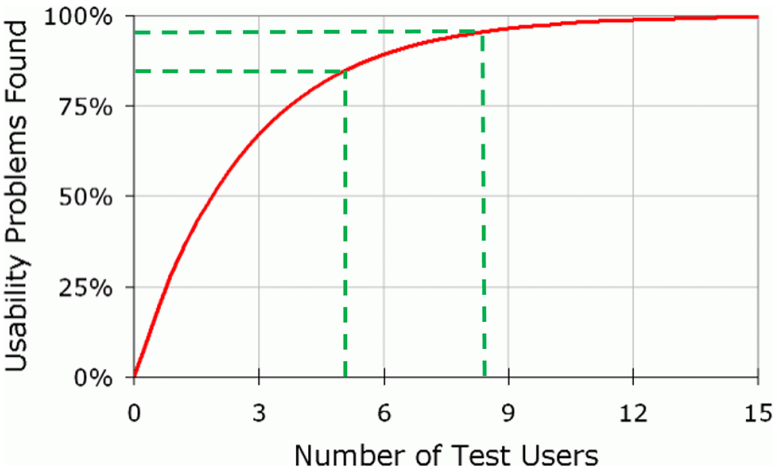



Figure 3.12: The optimal ratio between found usability issues and test group size is between 5 and 8 users. Adapted from [Mus14].

The user study itself should always follow the same plan: Optimally, the study is held in a quiet room with a computer inside and two or three chairs. Only the participant (the user that is subject to the study) and the moderator should be present during the study. Additional observers can follow the process from another room, or if this is not possible, from the side of the test room. It can also be a good advice to record all user sessions, since they can then be analysed later for criteria, which were not known during

the study.

During the study a lot of data is collected. In general, there are two categories of such information: Performance data, which can be quantitatively measured (everything that has a precise measurement), and preference data, which is qualitative (verbal descriptions, subjective feelings and opinions, preference scales). Both types of data can be analysed, but depending on its type different methods should be used: Quantitative data can be analysed by standard statistical methods, for example, mean value calculation, median, distribution etc. Qualitative data has, depending on the nature of question, to be reorganized before statistics can be applied. In the top-down approach, several predefined categories are used and the answers are sorted to them, while in the bottom-up approach all the answers are grouped and the categories are labelled according to the contained data. Beside qualifying the data as being either qualitative or quantitative, it can also be qualified into being either objective or subjective. Objective data is everything, that can be measured directly e.g. shoe size or yes/no questions. Subjective data contains everything, that depends on opinions of users and is not measurable. Examples here are questions that can be answered on a scale or questions that require a textual answer. A very good matrix that should clarify this, is given by Musialski [Mus14] in Figure 3.13.

The classifications determine which mathematical operations can be applied to the data. For qualitative data, which is of nominal type (e.g. gender), only equal/unequal operations should be used. When the data is of ordinal type (qualitative, e.g. scale from very good to very bad), the data can also be compared using greater/less operations. Quantitative data, which is given in intervals (temperature, dates), can be added and subtracted in addition to the comparison operations. The data with the widest number of possible operations is quantitative data, which is given by a ratio (e.g. age from 0 to 99 years) where also multiplications and divisions are allowed. We will later on use this knowledge about user-studies and data analysis to present the results of our user study, which is described in Section 5.2. [Mus14]

| | Quantitative | Qualitative |
|---|---|---|
| **Objective** | **Q:** What is the CPU speed of your computer?<br>***A:*** *2 GHz* | **Q:** Do you own a computer?<br>***A:*** *Yes, I own a computer.* |
| **Subjective** | **Q:** How easy is computer to use on a scale of 1 to 10?<br>***A:*** *7* | **Q:** Give your general opinion about the price of computers<br>***A:*** *They are too expensive.* |

Figure 3.13: Matrix showing the difference between qualitative/quantitative and objective/subjective data. Reprinted from [Mus14].

CHAPTER 4

# Framework for Structure-Aware Model Manipulation

In this chapter, our framework for structure-aware model manipulation is presented, starting with an overview of the used multi-layered graph representation, explaining all necessary components used in our propagation algorithm. Afterwards the algorithms used for detecting these parts in an input mesh are presented, followed by an description of the general idea of symmetries in curve spaces and how parameters for them are calculated. In the next section, the propagation algorithm used for updating the model is shown. At the end, a short description of the user interactions in our framework is given.

## 4.1 Overview

Our framework follows the general design principle for structure-aware mesh manipulation systems: First the model is analysed, and afterwards manipulations made by the user are handled interactively. A flow diagram showing the various stages of the system is given in Figure 4.1.

In the model-analysis stage, the parts and binary relations are identified automatically, giving a connectivity graph that describes how parts and relations are tied together. The parts are then analysed to find symmetry groups in the model, which are added to the graph structure as additional layers. In our work, we try to describe the arrangement of the parts in these symmetry groups by a control curve, which also allows us to repeat the elements along such a curve. While the detection of symmetry groups happens fully automatically, the user has to select the type of control curve used. Depending on the choice of control curve, the system automatically finds all necessary parameters. The behaviour of objects on such a curve is specified by the user. One can selected whether parts should only be rearranged when the curve is changed, or if the number of parts is changed to keep the distance between parts as constant as possible. For both types of

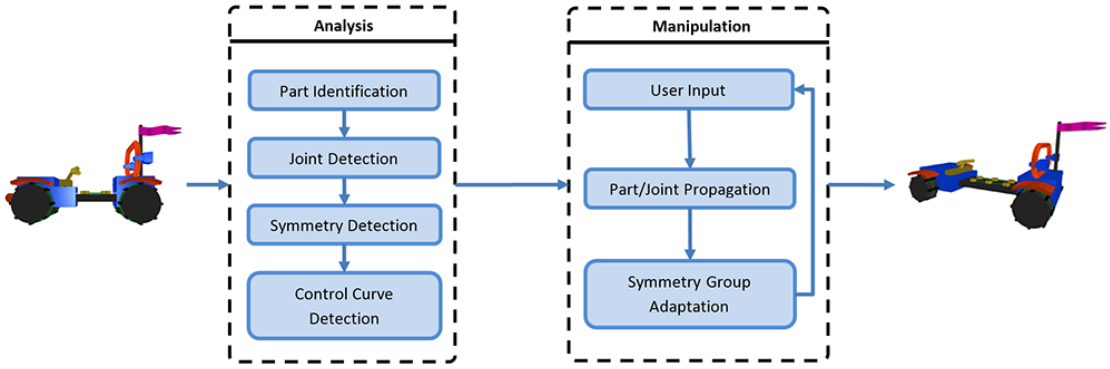



Figure 4.1: Flow diagram of the model manipulation framework.

modification behaviour, the system detects all required parameters, like the orientation of parts to the curve, or the distance between parts, automatically.

When mesh analysis is finished, the user can interact with the system by manipulating the parts. The resulting changes are propagated through the connectivity graph to modify other binary relations and parts, such that the overall structure of the model is maintained. Whenever necessary, higher-order relations are adapted to the changes introduced by the propagation. For example, the number of elements in such a group can be altered when the length of the control curve is changed. The system also maintains the position and the orientation of parts that are controlled by a curve by keeping both of them on the curve, or in the Frenet frame of the curve. The steps in the modification stage are repeated, starting with the user interaction, until the user is satisfied with the result.

Before starting with a detailed explanation of the analysis stage, we will describe the elements of which we compose models in our framework, and how these elements are stored.

## 4.2 Data Structure

To store all the informations about an input model, and especially the newly defined symmetry groups, it is highly desirable to have only one data structure that can describe the whole model in a unified way.

According to the definition of Mitra et al. [MWZ+13] (see Section 3.1), every model consists of several parts, which are connected by relations to form a unique composition. Relations can be grouped in two categories: Binary relations, which describe the physical connectivity between two parts, and higher-order relations, which handle all constraints where more than two parts are involved, e.g., the curve symmetries described later. Relations as well as parts can have parameters, which describe the appearance or the behaviour of the specific element. An example for such a decomposition of a ladder model is given in Figure 4.2, where on the left side (a) the original input model is given. The parts of the model are surrounded by blue boxes in (b), while (c) shows the binary

relations between the connected parts with yellow boxes. We will describe in this section how parts, binary relations and higher-order relation are defined, and how they can be stored in a unified data structure.

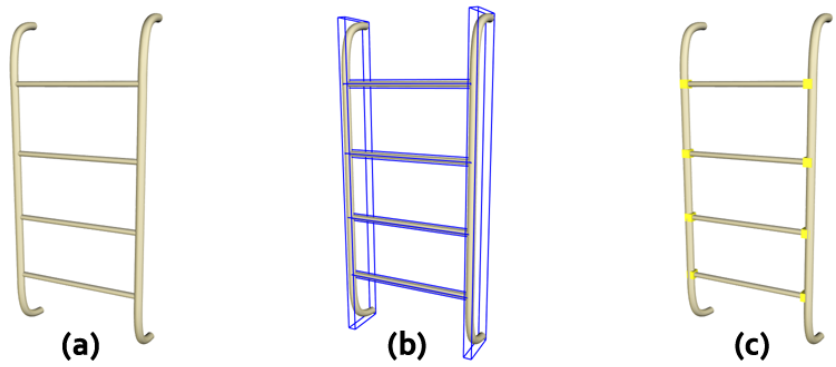



Figure 4.2: Overview on parts and binary relations: (a) Input model, (b) parts, (c) binary relations

**Parts:** In our framework, parts are geometric sub-meshes of the input model. We assume in our application that the input model is pre-segmented by the user, since mesh segmentation algorithms delivery very good results, and we do not want to contribute in this field. For every part that is present in the model, a bounding box is defined, which covers the underlying geometry. We will see in Section 4.3.1 how these boxes can be calculated. The bounding box is the most important parameter that a part has, since it defines the three principle axes, along which the part can be scaled or translated. For the automatic model adaptation, the user can specify along each of these axes whether this part can only be moved, or if it can change the size. Furthermore, one can specify that resizing can only happen uniformly along two or three axes. Using this, it is possible to maintain the circular cross-section in a cylinder by specifying that scaling along both axes of the cross-section is only allowed by the same scaling factor.

**Binary relations**: The physical connectivity between exactly two parts is described by a binary relation. Parts can be connected by multiple binary relations, as we will see in detail in Section 4.3.2, since parts can overlap in multiple disconnected regions. A binary relation is fully described by its position in the world and the two parts it connects. For each part that is connected by a binary relation, the relative position in the part's bounding box is stored. When two connected parts are in a valid position, then the relative orientations in box space describe the same point in world space. When one part is moved, the world space position of the binary relations changes with it. Since the two relative orientations no longer describe the same location in world space, the model has to be adapted to move the two parts again in a valid position. More on how parts and binary relations are updated is given in Section 4.5.2.

As already mentioned, a binary relation stores its relative position in both bounding boxes. These parameters can be found by calculating the relative position of the binary relation along each of the bounding box axes, describing the relative distance between


Figure 4.3: Binary relations, marked by yellow boxes, can be oriented relatively between two sides of the box (a) or with absolute offset to one side (b). The arrows show to which sides of the bounding box the binary relation is related.

the opposite sides of the bounding box (see Figure 4.3 left). We call this type of binary from now on *relative binary relation*. During the development it turned out that although relative binary relations alone should theoretically be sufficient, this is not the case with practical models. There can be some problems when two parts are touching, and the binary relation location is not calculated exactly at the side of the bounding box. This can happen due to imprecise modelling, but the algorithm used to identify significant binary relations also tends to produce relations that are slightly moved into the box when curved surfaces are present. Visually, the problem can be seen when a thin object is present, since the relative binary relation will, when one box is scaled, tend to move one part through the other one. This behaviour is shown in Figure 4.4, where in the initial situation (a), the bounding boxes overlap. When in (b) the blue box is scaled, the relative orientation of the binary relation is maintained, resulting in the blue box standing through the orange one.

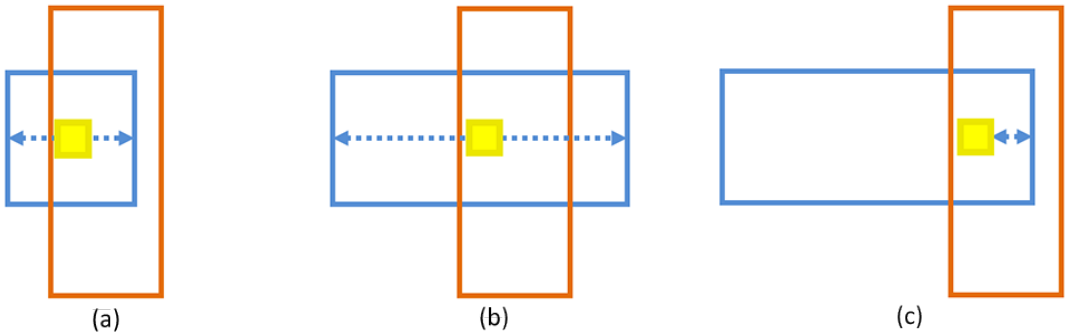



Figure 4.4: Relative binary relations can produce problems due to imprecise modelling. (a) Shows the initial model, where a relative relation connects the blue and the orange box. When the blue box is scaled (b), it can happen that the box is also visible on the other side of the orange box, since the relative position of the relation is maintained. To overcome this problem, we can tie the binary relation only to the right side of the blue box (c), which produces a better looking result.

To overcome this problem, we added a second binary relation type, the so-called

*absolute binary relations*, which are, in contrast to relative ones, tied with an absolute offset to one side of the bounding box. This type of relation is shown in Figure 4.3b, where the binary relation is only tied to the left side of the box. This helps to overcome imprecise locations due to not perfectly aligned geometry, as shown in Figure 4.4c. Binary relations are identified fully automatically by our implementation, which is described in Section 4.3.2.

**Higher-order relations**: A more general concept than binary relations are higher-order relations, which allow to define relations between more than two parts, such that each higher-order relation contains a set of parts. In our application this concept is used to describe curve symmetries: a concept that we explain in more detail in Section 4.4. Symmetric parts are automatically identified by the framework (see Section 4.3.3), but the user has to specify the type of control curve that should be used for a specific symmetry group. Similar to parts and binary relations, higher-order relations also have a number of parameters. These parameters depend mainly on the control curve used, and are detected automatically, which is presented in detail in Section 4.4.1. In addition to the control curve and their parameters, the user has to specify in the model analysis stage whether parts should only be rearranged to stay evenly spaced when the control curve is modified, or if the number of parts should be adapted to keep a constant distance between them.

**Connectivity graph**: As described by Mitra et al. [MWZ+13], the connectivity graph is an undirected graph, which stores parts and binary relations. Let $G = (V, E)$ be such a graph, the nodes $V$ represent the parts of the model, while the edges $E$ represent respective binary relations, linking the two parts the relation connects. The graph that corresponds to the model from Figure 4.2 is presented in Figure 4.5 on the left side, where parts are again marked blue and binary relations are given in yellow. We will see later on, how this connectivity graph is used in the propagation algorithm to determine in which order parts and binary relations are updated.

**Multi-layered graph representation**: Since we want to store the information about parts and binary relations in the same data structure as the higher-order relations, we have to extend the simple connectivity graph. For this, we will use a multi-layered graph representation, which was used successfully in other fields of computer science like for example network routing [XXS05] and feature matching [LLZ10]. A layered graph consists of a base graph, and some layers $\{L_0, L_1, \cdots, L_n\}$, which contain a subset of the nodes $V$, connected with layer-specific edges. Thus a layer for the base graph $G$ is defined as $L_i = (V_i \subseteq V, E_i)$.

To describe the structure of a model using a layered graph, our framework uses the connectivity graph, which is formed by the parts and binary relations as base graph. For each higher-order relation, we create a layer in the graph, which contains exactly the set of nodes that the higher-order relation operates on. When describing curve symmetries, the nodes are given by the parts that are aligned on the control curve. The edges in each layer define the ordering of the parts along the control curve, thus every part is connected with its predecessor and successor along the curve. How this order is calculated is presented in Section 4.4.1, since this strongly depends on the type of control curve

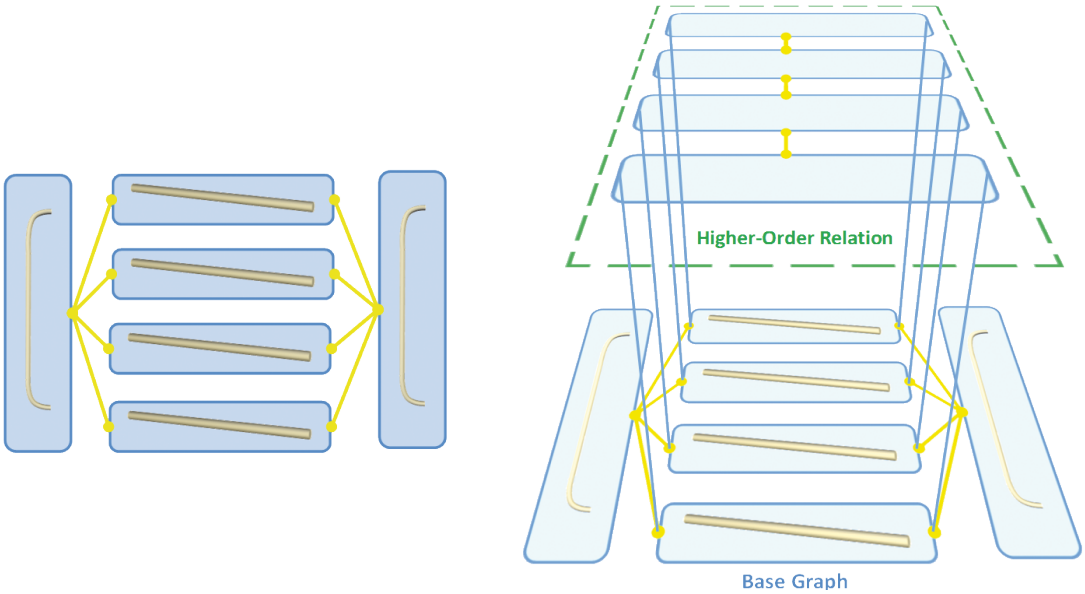



Figure 4.5: (left) Graph corresponding to the ladder model. (right) Layered graph with a higher-order relation grouping the steps.

used.

## 4.3 Model Analysis

Since only a model without additional information is given as input to our system, we need to find as many structures in the model as possible in a semi-automatic way. First, we have to detect parts in the input model, which are the most primitive elements we are dealing with. Parts can behave in two ways in the propagation algorithm: (1) They can be restricted to translations, or (2) they can change also their size depending on the surrounding situation. Due to this, the grouping of triangles into meshes should be such that triangles with different resize behaviour are never present in the same part. Since the user can only modify parts, but not triangles directly, a second criterion for the grouping is that all sub-meshes that should be modified independently are grouped in different parts. The problem of segmenting meshes is a well-known topic in the modelling community, and segmentation algorithms delivery very good results (see e.g. [AKM+06, CGF09]). In our application, we decided to assume that the input geometry is pre-segmented by the designer. Since most 3d applications have many settings that are specified per geometry part rather than per vertex, the part hierarchy used in the modelling application is often sufficient for use in our method.

For each part in the model, we have to find a tight fitting bounding box (see Section 4.3.1). Using these bounding boxes and the underlying geometry, the system detects binary relations between the parts and stores them together with the parts in the base

graph of our data structure (Section 4.3.2). The next step in the model analysis is to detect symmetries between parts in the input model (Section 4.3.3). Based on these automatically detected symmetry groups, the user can select which type of curved symmetry (see Section 4.4) should be present in a group. The user has to select only which type of control curve should be used, and whether parts should be repeated along this curve, while the system detects the parameters for these relations automatically. The algorithms used for this are described in Section 4.4.1.

### 4.3.1 Bounding Box Calculation

In the first step of the model analysis, we have to compute a bounding box for each of the geometric parts. The axis of this bounding box will be used to determine the directions along which a part can be modified. As described beforehand, these directions are also used to specify how binary relations are related to a part. The first algorithm we used to calculate bounding boxes was a PCA (principle component analysis) based approach that treats the input mesh as a point cloud and calculates the eigenvectors of the centered points. Given a set of points $\{p_0, p_1, \cdots, p_n\}$, the axis of a bounding box can be found by first centring the set around its center of mass. This results in a new set $PC$ with centered points $\{pc_0, pc_1, \cdots, pc_n\}$, where each point is calculated as follows:

$$pc_i = p_i - \frac{\sum_{j=0}^{n} p_j}{n}. \tag{4.1}$$

We then perform then a singular value decomposition (SVD) of the matrix containing the centered point set:

$$U\Sigma V^* = \begin{pmatrix} pc_0 \\ pc_1 \\ \cdots \\ pc_n \end{pmatrix}. \tag{4.2}$$

The directions of the axes for the bounding box can be found as the vectors in the matrix $V$ that correspond to the three largest singular values in $\Sigma$. To determine the size of the bounding box, the points have to be projected onto each axis, and the minimum and maximum has to be stored.

In practice, this algorithm works very well when the input mesh has a sufficiently large number of points and significant directions that can be found. In our application, it turned out that for example with 8-cornered boxes, the algorithm produces very different results depending on the orientation of the box. To overcome this, we also tried the DiTO algorithm by Larsson and Källberg [LK11], where a simple shape (a ditetrahedron) is constructed from the input mesh, which is then used to derive the orientation of the bounding box. According to Larsson and Källberg, a ditetrahedron is a mesh consisting of two irregular tetrahedra connected along a shared interior side. In order to construct such a structure, a set of extremal points is needed, which can be found by projecting a subset $S$ of the input vertices $P$ onto a set of given normal directions. The extremal points are selected as the two vertices in $S$ that have the maximum projected distance.

These points are used to construct the so-called *large base triangle*, by choosing the extremal point duple with maximum distance between them:

$$\{p_0, p_1\} = max\|a_j - b_j\|. \tag{4.3}$$

The triangle is constructed using the points $p_0$, $p_1$, and a third point $p_2$ that has maximum distance to the line going through $p_0$ and $p_1$. For each of the edges $e$ of this triangle with surface normal $n$, one oriented bounding box is generated with the following axes:

$$\begin{aligned} u_0 &= e/\|e\| \\ u_1 &= n/\|n\| \\ u_2 &= u_0 \times u_1. \end{aligned} \tag{4.4}$$

To select the best bounding box, the surface area of each of the candidate boxes is calculated, and the box with the smallest area is chosen. In order to ensure that the selected bounding box has a good quality, the axis-aligned bounding box is also calculated, and if the area of the AABB is smaller or equal to the area of the previously generated OBB, the axis-aligned bounding box is used instead.

This algorithm produces better-fitting bounding boxes than the PCA algorithm in less time, and in addition looks much more plausible to the user since it finds similar bounding boxes for similar meshes regardless of the orientation. Figure 4.6 shows results of the PCA algorithm on the left side and the same scene with the DiTO algorithm on the right side. The boxes where the difference can be seen best are marked in red.

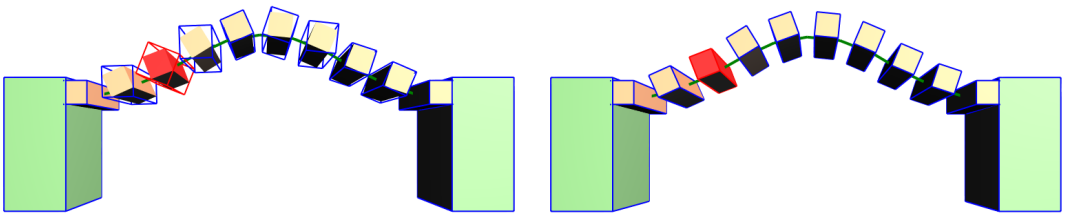


Figure 4.6: Bounding-box computation with PCA (left) and DiTO (right)

### 4.3.2 Binary Relation Detection

After all parts and their respective bounding boxes have been found, the next algorithmic step is to detect binary relations between these parts. A simple solution for this problem is to check for intersections between the bounding boxes and to compute the position of the binary relation by taking the center of the intersection region. In most cases, this solution would give very good results, but there are some problematic configurations: Assuming we have an object that consists of an outer ring and a cylinder going from one side of the ring to the other. Since the bounding box of the cylinder would be completely

inside the boundings of the ring, our primitive detection would only find one binary relation in the middle of the cylinder (see Figure 4.7a). This would lead to incorrect results after manipulating the model, since the cylinder will not be tied to both sides of the ring. The optimal result we are looking for in this case would be to detect two binary relations between the parts, one on each end of the cylinder (b).

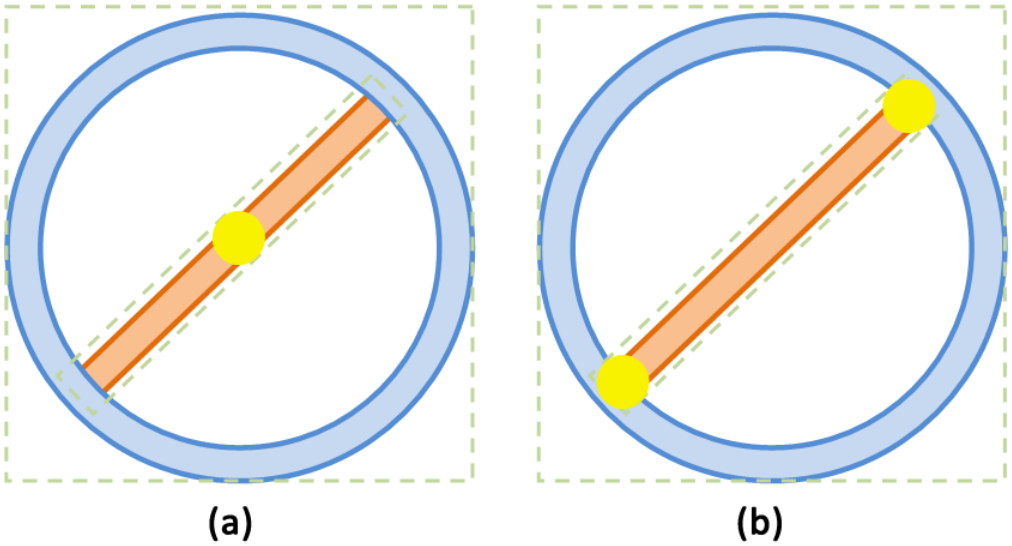



Figure 4.7: Binary relation detection using bounding box overlap does not work in cases where more than one relation between two parts is necessary.

Algorithms that can handle this cases will necessarily need to take the geometry contained in each part into account. For this, we propose a new algorithm that is based on the contact analysis in the work of Jain et al. [JTRS12]. The difference is that, instead of using an axis-aligned bounding-box tree, we use an oriented bounding-box tree [GLM96] for each part in our implementation.

### Part Intersection

An oriented bounding-box tree is a tree structure where the root node contains an oriented bounding box that encapsulates the whole 3d model. We will use this acceleration structure in future work also for other tasks, so we decided to use PCA bounding boxes here instead of the DiTo algorithm used for the bounding boxes that are used in the propagation algorithm. The reason for this is that many algorithms will rely on stable eigenvalues, and the DiTo algorithm would not calculate these. Since PCA bounding boxes have some drawbacks, as described in the last section, our implementation does not only take vertices into account, but instead it generates a dense point cloud on the the surface of the triangle-mesh. This is also required due to the unequal spacing of vertices when small and large triangles are present in the mesh. We implemented two methods for this point-cloud refinement, starting with a very simple one, where we project a grid on

each triangle, and generate a new point for each cell that contains parts of the triangle. The more advanced algorithm accomplishes the task by using Lloyd's algorihm [Llo82], but the differences in the bounding-box quality were negligible, so in our final version, the grid algorithm is used, due to performance considerations.

The resulting point cloud of the refinement algorithm is then used as an input to the PCA algorithm, which generates the root node bounding box. The tree is built up recursively by defining a splitting plane that is orthogonal to the longest axis of the bounding box. The position of the plane is calculated by taking the average over all points, and letting the plane go through it. The splitting algorithm then tries to partition the triangles in the node into two groups, one in front of the plane and one behind. If this is not possible, or does not produce good results, the whole step is performed on the second-longest axis first, and then on the shortest one. For both child nodes, the algorithm is called recursively, starting with the calculation of a new bounding box for the child point cloud. When a node cannot be subdivided along any axis, or when the number of triangles present is smaller than a specifiable threshold, the tree generation is finished.

Using this data structure, the binary relation detection works as follows: For each pair of parts, the two corresponding trees are intersected, following in each level of the tree all sub-nodes that intersect the other tree. In the end we get a list of intersecting leaf nodes, and from them a list of triangles that potentially intersect each other. These triangles are then tested against each other by using the triangle-triangle intersection algorithm by Möller [MÖ97]. This method calculates the plane equation of both triangle's, and first tests whether all points of one triangle lie on the same side of the other triangles plane. If this is the case, the triangle is marked as non-intersecting and the algorithm can stop. In all other cases, the intersection line is found and the intervals for each triangle are calculated.

### Binary Relation Clustering

As a result of this step, we get a list of intersection lines between the triangles. Since each binary relation is defined by one position, we have to derive the positions of significant binary relations from the line set. The easiest thing would be to represent every line by several evenly spaced points and use them as relation positions. But this would give a significant overhead, since we would have to take a large number of binary relations into account, which would have similar information about the connectivity. The overhead can be decreased dramatically by running a connected-component search algorithm, which is easier to design when working with a point cloud than when working with lines, since there are a lot of fully developed algorithms that can perform what we need. The result of these algorithms are just a few but very important binary relations. The component search algorithm in this implementation is based on the DBSCAN-algorithm [EKSX96], which starts with a list of points and an empty set of components. For each point, we check the shortest distance to each existing component, and if the distance is smaller than the threshold, the point is added to the component. If a point is below the threshold in more than one group, these two groups are joined together, producing one larger

component. This algorithm is exemplarily shown in Figure 4.8: At the beginning, (a) one component is already defined by the first point. A second point is tested (b), and since the distance is below the threshold, it is added to the component (c). After some iterations, three components have been identified, and the last point is compared (d). This point is below the threshold for the orange as well as for the green component. As a result, these two components form, together with the last point, one big component (e).

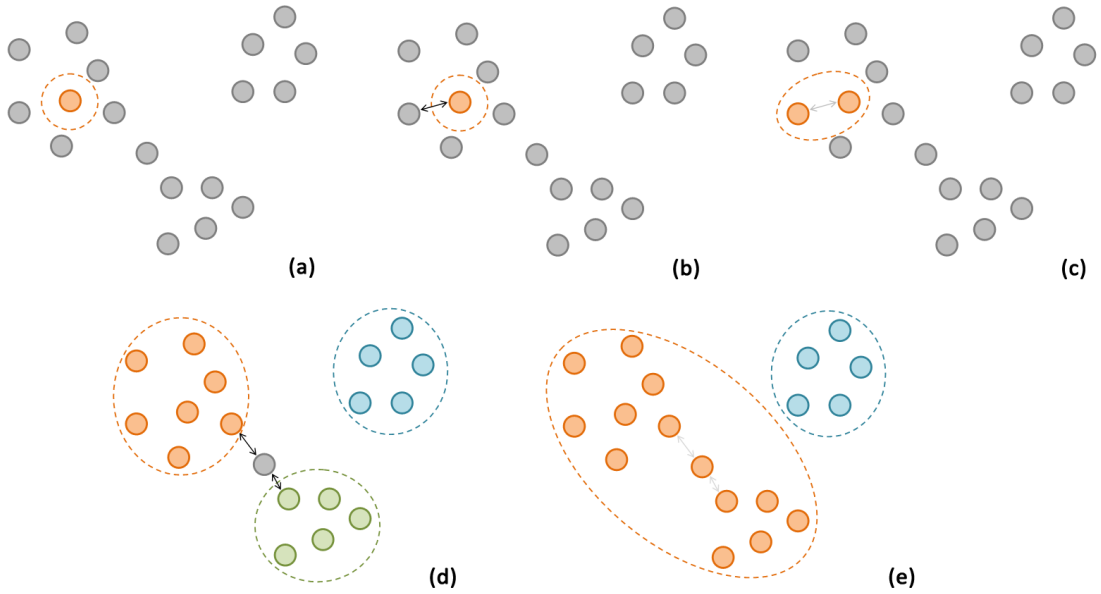



Figure 4.8: Five steps in a greedy component-search algorithm. (a) Initial state, (b) second point is compared to all components, (c) second point is added to the component, (d) last point is compared to the three components, (e) due to the last point two components are joined together.

For each cluster that is found by the connected-component search, a binary relation is created and the relation parameters are determined. This is done by checking along each axis of the part's bounding box if the relation position is located in the outer 10% of it. If this is the case, then the binary relation is marked as an absolute relation and the absolute offset to the next face is calculated. Otherwise, the binary relation is marked as relative and the relative orientation between the two faces of the box is calculated. Parts and binary relations together form the base graph of our data structure, where a node in the graph corresponds to a part in the model. For each binary relation an edge is constructed, that connects the parts that should be connected by this relation. This base graph will in the modification stage serve as basis for the propagation.

### 4.3.3 Symmetry Detection

Beside the detection of parts and their binary relations to each other, we also have to detect symmetry groups in the input model. As described in Section 3.2, there are two classes of symmetries, global symmetries and partial symmetries. In our approach, we only target symmetries between the parts of the mesh, since all our later algorithms work on these parts and not on the geometry directly. For a closer look on different algorithms for detecting symmetries (also partial ones) in meshes, please refer to Section 3.2.3. In our framework, symmetry detection happens in two steps: First, the pair-wise similarity between parts is detected by testing all parts against each other. Second, these pair-wise similarities are used to find clusters where all parts are symmetric to each other. The resulting cluster are then the symmetry groups we are looking for.

#### Pair-wise Similarity Detection

The general problem that arises when searching for similar parts in a mesh is the problem of finding a transformation that maps a point set $P = \{P_0, P_1, ..., P_i\}$ to another set $Q = \{Q_0, Q_1, ..., Q_j\}$, with not necessarily the same number of points in each set. In order to identify if two such sets are similar, the error $E$ that occurs when mapping $P$ with the best possible transformation $F$ to $Q$, has to be below a threshold. One method for solving this problem is the *iterative closest point algorithm (ICP)*, proposed by Besl et al. [BM92], which was improved over the years by many authors (e.g. [CSSK02]). Our implementation of the ICP algorithm restricts the possible transformations that can be applied to a point set to translations, rotations and uniform scaling. This is necessary since we want to find parts in the mesh, which would also look similar to a user. When, for example, non-uniform scaling is allowed, every box-shaped mesh could be mapped to every other box, which would lead in the model used in Figure 4.6 to symmetries between all twelve parts.

When comparing two point sets ($P$ and $Q$), the first thing that needs to be done is to transform all points to a space where the comparison can be performed. This can easily be achieved by centering both point sets around their center of mass $\bar{p}$ and $\bar{q}$, similar to the calculation performed in Equation 4.1. In this transformed space, for each point in $P$, a corresponding point in $Q$ has to be detected and the mean squared error for this configuration has to be calculated. The method for finding point correspondences in our implementation is simply to find the nearest point in $Q$, which gives a mapping $C : P \to Q$. The initial error is then calculated as

$$err = 1/n \sum_{i=0}^{n} (P_i - C(P_i))^2. \tag{4.5}$$

Using the mapping defined above, we have to find the best transformation, which can, according to Sorkine [Sor09], be computed by calculating a covariance matrix of the two pre-centered point sets:

$$S = PQ^T. \tag{4.6}$$

The optimal rotation can be calculated by performing a singular value decomposition of $S = U\Sigma V^T$ and is then given in matrix form as

$$R = V \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & det(VU^T) \end{pmatrix} U^T. \tag{4.7}$$

The optimal translation is given by

$$t = \bar{q} - R\bar{p}. \tag{4.8}$$

Starting with the correspondence calculation, this algorithm has to be performed several times, until the mean squared error is less or equal than the required threshold, or until the change in error between two iterations gets too small. If the final error after the last iteration is below a threshold, the two compared point sets $P$ and $Q$ can be treated as similar objects.

There are several problematic cases with this simple version of the ICP algorithm. The most important one is that depending on the initial orientation, there can be a stable situation between two similar objects although the alignment does not fit well. This problem occurs especially in cases when there are very few points in each set. For example, two boxes of similar size that are nearly orthogonal to each other would be identified as not symmetric, due to the equal forces pushing to both sides. Figure 4.9a shows the initial situation where the point correspondences are marked with gray arrows. After the first iteration the two boxes are perfectly 90° aligned, and all arrows are of equal size. Since these forces are cancelling out each other, no rotation is calculated any more.

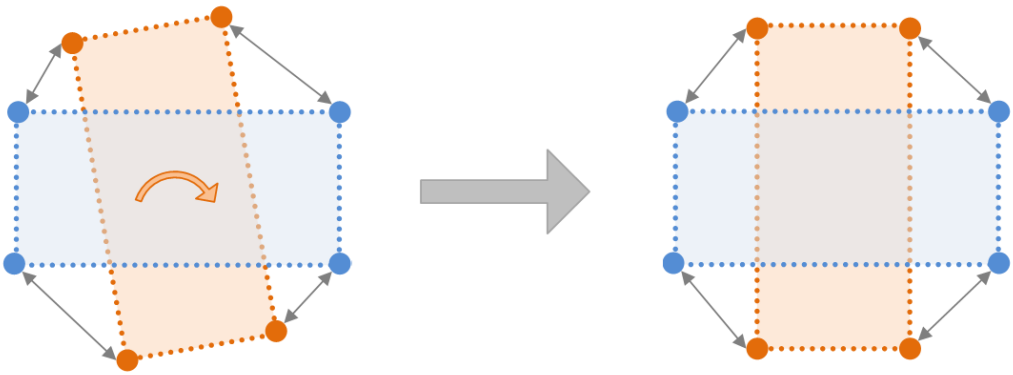


Figure 4.9: ICP algorithm producing incorrect results in sparse point sets

As a solution to this problem, our symmetry detection always rotates the bounding boxes of the mesh such that the longest axis points in the x-direction, the second largest

in y-direction, and the shortest along the z-axis. This produces in general better results and reduces in most cases the required number of iterations in the ICP algorithm, as long as the bounding boxes for two similar objects are also similar.

### Symmetry Group Clustering

The ICP algorithm identifies whether two parts in the model are symmetric to each other. The next step is to identify groups of objects, so-called symmetry groups, where all objects that are symmetric are grouped together. We identify these groups by an algorithm similar to the DBSCAN algorithm, which was used for binary relation clustering. The algorithm starts with an empty set of groups, and a list of parts that we want to group. For every part, we test if this part is symmetric to any object in an already existing group. If this is the case, the part is added to this group, if not, a new group is created. Since we are dealing with approximative symmetries here, where we mark parts as being symmetric when the error after the ICP algorithm is below a threshold, it can happen that a part is symmetric to parts in more than one group. In this case, our implementation joins the the two groups together, resulting in one large group.

Each symmetry group that has been found is added to our data structure as an additional layer. In this layer, all the parts that are contained in the symmetry group are present, together with the parameter that describe this higher-order relation (see Section 4.4.1). In the next section, we will see how the edges in these layers are found.

Another problem with this symmetry detection is that all equal objects in a scene are marked as being symmetric. For example, if the walls of a castle have two rows of battlements (one on each side of the wall, see Figure 4.10 left), all these battlements are symmetric, but we will see in the next section that this is not exactly what we are looking for. What we want to find are two groups of battlements, each of them constructed on a line (Figure 4.10 right). At the moment, our system does not take this into account, since finding related objects is a very active research field, and our contribution is not in this area.

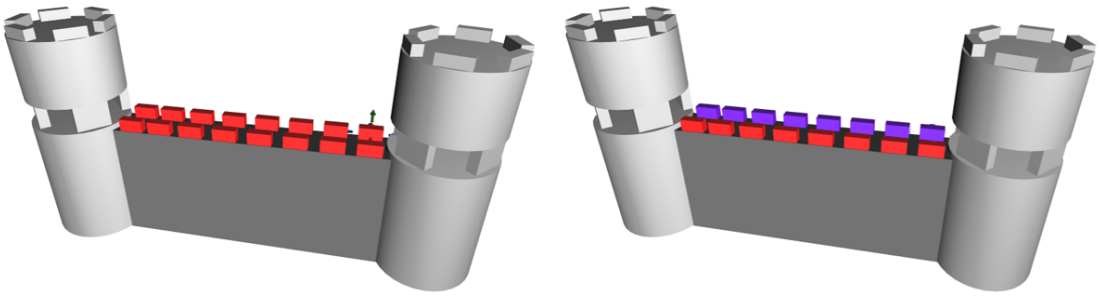


Figure 4.10: ICP finds all objects that are symmetric to each other, but in many cases this is not sufficient. In the castle wall example, all basements are symmetric, but should optimally be in two groups, each spanned by a line

## 4.4 Symmetry on Curves

Symmetry (as described in Section 3.2) is a very helpful theory when dealing with the structure of meshes. In most of today's applications, complex symmetric relations are only considered for maintaining the structure of an object [GSMCO09], [ZFCO+11]. But when it comes to adaptations, only very limited support is available, mostly 1d-translations (resulting in objects lying on a line, [BWS10, BWKS11]) and 2d-translations (objects lying on a grid, [BWSK12]). The approach used here for structural adaption should extend this to a model which can deal with a wider number of symmetries, while also working with these simple types.

To achieve this, we introduce so-called *control curves* for symmetries. The type of control curve used determines which type of symmetry is used. A control curve is in our case a parametric curve of the form $(x, y, z) = c(t)$, thus allowing for a large number of curves to be supported. We define a set of objects $\{P_0, P_1, \ldots, P_n\}$ to be *curve-symmetric* if there exists a control-curve $c(t)$ such that all objects are evenly spaced in the arc-length parametrized curve space. This means that for each object $P_i$, the center position can be found by evaluating the curve at the given parameter, thus $Center(P_i) = c(i \cdot \delta)$, where $\delta$ is the distance between two consecutive parts in the group. Assuming arc-length parametrization, we define a transformation between two objects $P_i$ and $P_j$ as the geometric transformation that maps from the point $c(i \cdot \delta)$ to the point $c(j \cdot \delta)$. Since we are dealing at the moment with translations only, the transformation between these two objects is a translation from $c(i \cdot \delta)$ to $c(j \cdot \delta)$, which can also be applied to the vertices of the objects. We then show that this transformation is a symmetry transformation, i.e., the set of the transformations between all pairs of objects forms a symmetry group. We will from now on use the notion $c(i \cdot \delta) \rightarrow c(j \cdot \delta)$ for a transformation that maps from the point $c(i \cdot \delta)$ to the point $c(j \cdot \delta)$. Using the fact that both objects are part of the set, we can rewrite the formulation by using $j = i + k$, resulting in the transformation $P_i \rightarrow P_{i+k} := c(i \cdot \delta) \rightarrow c((i + k) \cdot \delta)$. In practice, many curves are limited to a given parameter range, thus it is also assumed that $k \cdot \delta$ is always in a range that does not exceed the definition space of the curve. We have to check that all four axioms for a symmetry group are fulfilled to ensure that the structure we have defined here is valid:

**Identity element:** The identity element $I$ should map an object to itself. In the case of *curve symmetry*, this is given by the transformation with $k$ equals 0. $I : c(i \cdot \delta) \rightarrow c((i + 0) \cdot \delta) = c(i \cdot \delta)$

**Inverse element:** Assuming we have a transformation $c(i \cdot \delta) \rightarrow c((i + k) \cdot \delta)$, then the inverse transformation $c((i + k) \cdot \delta) \rightarrow c(i \cdot \delta)$ has to be part of the group. Since we can define a transformation with $k_2 = -k$, we get $c((i+k) \cdot \delta) \rightarrow c((i+k+k_2) \cdot \delta) = c((i + k - k) \cdot \delta) = c(i \cdot \delta)$, thus the inverse transformation is always part of the group.

**Closure:** Like partial symmetries, curve-symmetries have the problem that they exist mostly in a limited definition space. As before closure is in general not given, but

when the definition space of the control curve is extended to infinity, allowing for an infinite number of repetitions, it can theoretically be achieved.

**Associativity:** As proven in the theoretical section, all symmetries that can be expressed by matrices are by definition associative. When looking at the definition used here, it can be noticed that $c(i \cdot \delta) \rightarrow c(j \cdot \delta)$ is just a translation from one point in space to another one, and can thus be expressed by a matrix. Using this, the associativity of our definition is also given.

In summary, a curve-symmetric set is a group as long as repetitions to infinity can be expressed. Straight lines or circles would for example allow for this.

Up to now, we have only considered groups of objects where the positions of the objects (or rather the centers of the objects) are aligned on a control curve. When working with general curves, this can behave unintuitively for the user. Therefore, a more complex formulation is required, which can not only express the location of parts, but also their orientation. To keep the naming consistent, from now on the definition that only deals with positions is called *translational curve symmetry*, and the following definition is called *full curve symmetry*. The difference between these two types of curve symmetries are shown in Figure 4.11, where the same control curve is shown. On the left side, translational curve symmetry is used, while on the right side, full curve symmetry is applied.

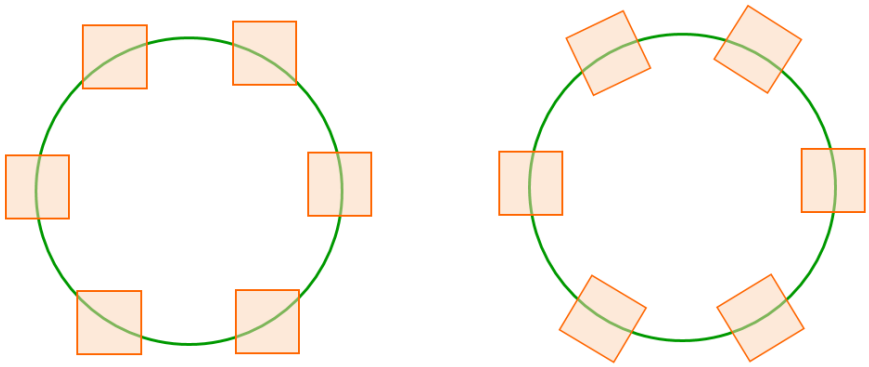


Figure 4.11: Curve-symmetries: translational curve symmetries (left) versus full curve symmetry (right).

Orientation on a curve is given by the Frenet frame (see Section 3.3.3). For a curve $c(t)$, the Frenet frame $e_c(t)$ is given by the tangent vector, the normal vector and the binormal vector. When the parts should not only be moved along the curve, but also be rotated along the Frenet frame, we need an extended definition for the symmetry transformation. For positioning, we can keep the transformation defined above, but we add an additional mapping from the Frenet frame orientation $e_c(i \cdot \delta)$ to the orientation specified by $e_c(j \cdot \delta)$. Since both mappings result in a transformation matrix, one translation matrix and one

rotation matrix, we can multiply them to get the final transformation. The symmetry transformation between $P_i$ and $P_j$ is thus defined as

$$\begin{aligned} P_i \to P_j &:= T(i,j)R(i,j) \\ T(i,j) = c(i \cdot \delta) &\to c(j \cdot \delta) \\ R(i,j) = e_c(i \cdot \delta) &\to e_c(j \cdot \delta). \end{aligned} \tag{4.9}$$

Again, we have to prove, that the group axioms are fulfilled, which is done similarly to the proofs given for translational curve symmetries. For the identity element, we have already shown that for a transformation with $k = 0$, the translation $T(i, j)$ will result in a identity matrix. The same hold for $R(i, j)$, since $e_c(i \cdot \delta) \to e_c((i+k) \cdot \delta) = e_c((i+0) \cdot \delta) = e_c(i \cdot \delta)$. The inverse element can again be found by using a transformation with $n_2 = -n$. As for translational curve symmetries, the closure is only given when the definition space of the curve is extended to infinity. Since the resulting transformation for full curve symmetry is a matrix, associativity is given by definition. Due to these proofs, full curve symmetry forms again a symmetry group, as long as the definition space is extended to infinity.

### 4.4.1 Curve Symmetry Parameter Calculation

Up to now, all steps in the model-analysis are working fully automatic, but for curve symmetries, the user has to specify additional input. Using the symmetry groups found in Section 4.3.3, the user can specify how the parts in such a group should be related to each other. In our framework, the primary method for doing this to specify a control curve along which the parts should be aligned. The type of curve that should be used has to be specified by the user, but the system identifies the parameters required for each type of curve automatically. At the moment, several control curves for curve symmetries are implemented, primitive ones like lines and circles as well as more complex ones like Bezier curves and NURBS. In addition to the type of control curve, the user has to specify how parts should react to changes of the curve. Parts can only be moved and stay evenly distributed along the curve, or the number of parts can be changed to keep the distance between them fixed. In both cases, translation-based curve symmetry is as well supported as full curve symmetry. We show in this section how the parameters of different control curves are calculated, and then describe how the relative orientation of parts to the Fourier frame of the curve is found.

#### Lines

In this most primitive case, all the elements in the group should be aligned on a straight line. Here, we only need the first and the last part on the control curve, which will then span the line. These two parts can be found by calculating the length between each component pair, and choosing the combination with the longest distance between them as start and end. All other elements are projected on the line and their parameters on the control curve are evaluated. In the data structure layer that describes the current

symmetry group, edges are added between consecutive parts on the line. This is done by always spanning edges from each part to the part with the next smaller parameter and to the part with the next bigger parameter. The start and the end part are only connected to one adjacent part. An example for such a configuration using a line control curve is shown in Figure 4.12, where the pickets of the fence are aligned on a line. The control curve is shown as a green line.

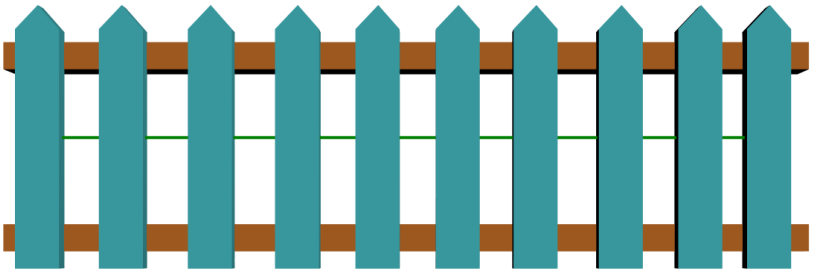


Figure 4.12: A fence, where the pickets are aligned on a control line (marked with a green line)

### Circle

In many man-made objects, there exist parts that are oriented along a circle. For this type of control curve, we need to find the best matching circle for a given set of parts. Circle detection happens in two steps: First, finding the best plane in which the circle should lie, then all the parts get projected to this plane and the best circle is found. The regression plane is calculated by first centering all parts and then computing the SVD. This is done in a similar way as in the PCA bounding box detection (Equations 4.1 and 4.2). In contrast to the bounding box detection, where we searched for the directions with the greatest amount of change, we are now looking for the direction with the fewest changes, which is given by the eigenvector corresponding to the smallest eigenvalue. This direction is used as a normal vector ($n$) for the plane we are looking for. After finding the regression plane, we project all points to this plane:

$$x_{proj} = x - dot(x, n). \tag{4.10}$$

Since the center of the original points went through the origin, we can calculate for each part a circle with the radius given by the distance of the point to the origin. The final radius for the control circle is then given by averaging over all these part radii. A circle controller generated by this method is shown in Figure 4.13. The control curve (the green line) connects all the spokes of the wheel.

Again we have to add edges describing the order along the curve in the data structure's layer. In case of circles, we calculate the angle that each element has in the circle. Since the absolute values of the angles are not important, an arbitrary part can be selected to have an angle of 0°. For every part we add two edges to the layer, that connect the part

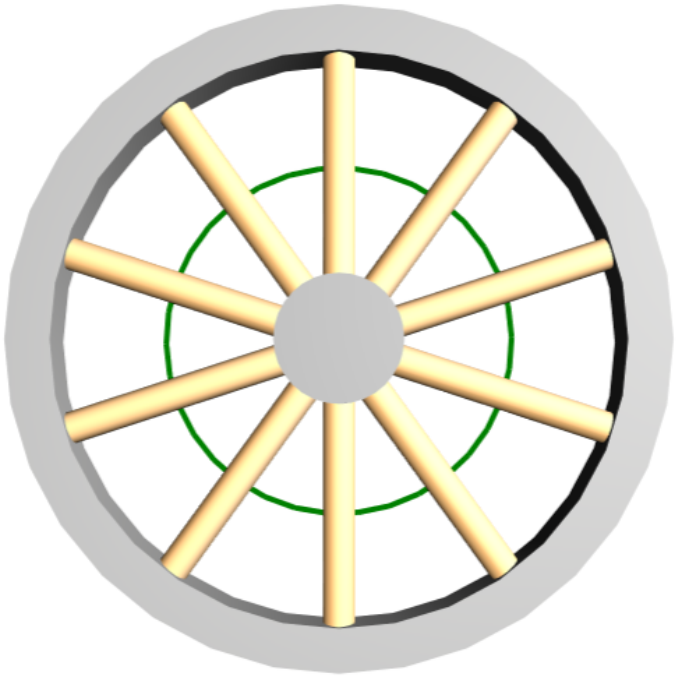


Figure 4.13: A wheel where the spokes are aligned on a control circle (marked with a green line).

with the two other parts that have the next higher angle and the next lower angle. In contrast to the line model, where we had a start and an end part, all parts along the circle are equal, thus the edges are also resulting in a circle in the graph.

### Bezier Curves

Bezier curves (as described in Section 3.3) are a special type of parametric curves. Again we have to find the best-fitting curve for a given set of parts (respectively for a given number of points defined by the centers of the parts). As an additional user input, we require the degree that the curve should have. While testing the framework, we observed that for structures that a user can identify, a maximum degree of four is sufficient, although we also allow the user to select a larger number.

Looking at the input points, the first information that has to be computed is an ordering of the points on a control polygon. Since there exists an infinite number of curves that go through a point set, this ordering will give us a constraint to find a well-usable curve. The ordering is calculated using a greedy algorithm, which starts with the two points that have the shortest distance between each other. Using this polygon, always the point is chosen which has the shortest distance to the a end of the polygon. This point is than connected to this end and the algorithm is redone until no points are left. Figure 4.14 shows some steps of this algorithm on a simple point set. This ordering is

also used to add the edges in the symmetry group's layer, since it already defines the correct ordering along the curve.

Figure 4.14: The greedy algorithm for control-polygon generation: (Left) Initial polygon, (Middle) Nearest point is joined to the polygon, (Right) Final polygon.

Next, we have to find the control points of the Bezier curve. For this, we define a system of linear equations of the form $Ax = b$ for each coordinate axis. The matrix is a $degree \times \#points$ matrix, which stores along each row the Bernstein polynomial for a point in the point set, thus forming a basis of the vector space of the polynomials:

$$\begin{pmatrix} B_{1,degree}(t_1) & B_{2,degree}(t_1) & \cdots & B_{degree,degree}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ B_{1,degree}(t_n) & B_{2,degree}(t_n) & \cdots & B_{degree,degree}(t_n) \end{pmatrix}. \tag{4.11}$$

In this matrix, the curve parameter $t_i$ for each point on the curve is required. From the theoretical point of view, we would get the best-fitting curve by using the distance on the ordering polygon. In our use case, we decided to use an equal spacing instead, since all algorithms that operate on this controller curve assume that the elements on the curve are similar to each other and have equal distance in curve space. The matrix $b$ contains the coordinates of the input point set along the corresponding axis. Solving this system of equations for all three axes results in a list of control points for a Bezier curve of the defined degree, which approximates the input point set best. An example of a Bezier curve control element is shown in Figure 4.15, where the cylinders of the bridge are oriented on such a curve.

### NURBS

The workflow for finding NURBS is generally the same as the one for Bezier curves. First, the ordering of the input points on the curve is calculated with the greedy polygon-finding algorithm. Again this ordering is used to add the edges to the layer. The input points, together with the ordering information, are then handed over to a NURBS-fitting algorithm, which calculates at first a NURBS curve of the desired order which has the same number of control points as the input set. This fine-grained curve is then handed over to a knot removal algorithm, which tries to reduce the number of control points without producing an error larger than a given threshold. Figure 4.16 shows the same bridge model as in Figure 4.15, but with a NURBS curve instead of a Bezier curve.

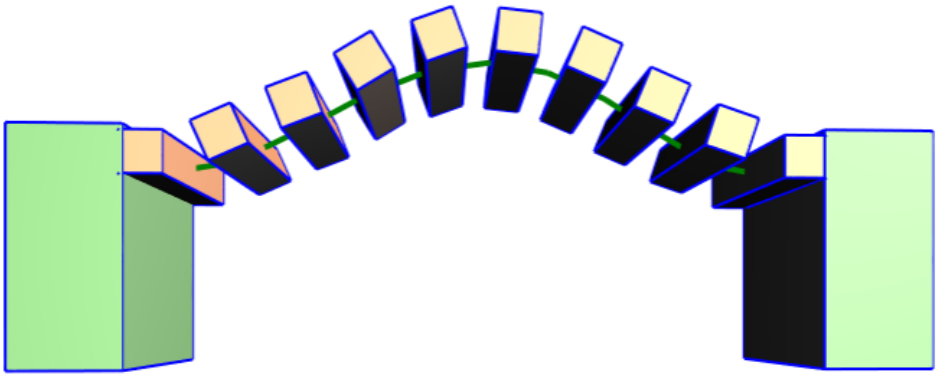


Figure 4.15: The orange cylinders of the bridge model are located on a Bezier curve.

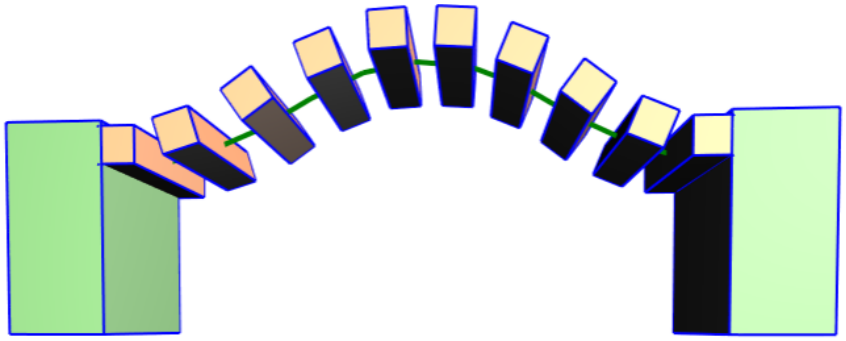


Figure 4.16: A bridge model where the cylinders are aligned by a NURBS curve.

### Base orientation

Beside the calculation of the control curves, the algorithm also requires the relative orientation of the curve parts to the coordinate system defined by this curve for handling full curve symmetries. For example, the part's x-axis is not necessarily the same as the x-axis in the curve's coordinate frame (see Figure 4.17a for an example). When using a line control curve, this task is trivial, since the coordinate system on a line has the same orientation in every point, which makes it unnecessary to touch the rotation component of the part's transformation matrix. Due to this, we do not need any base orientation here.

For the other three control curves, the coordinate frame can change over the curve, so for each part we have to find the matrix $T_{base}$ that specifies the necessary rotation to transform a part from the local coordinate frame defined by the curve to the final orientation. This transformation is visually shown in Figure 4.17b.

In order to calculate the base transformation, we first have to calculate the Frenet frame at the location of a part. In case of the circle controller, this is given by using the vector from the center to the part's position as z-axis, the normal of the plane in which

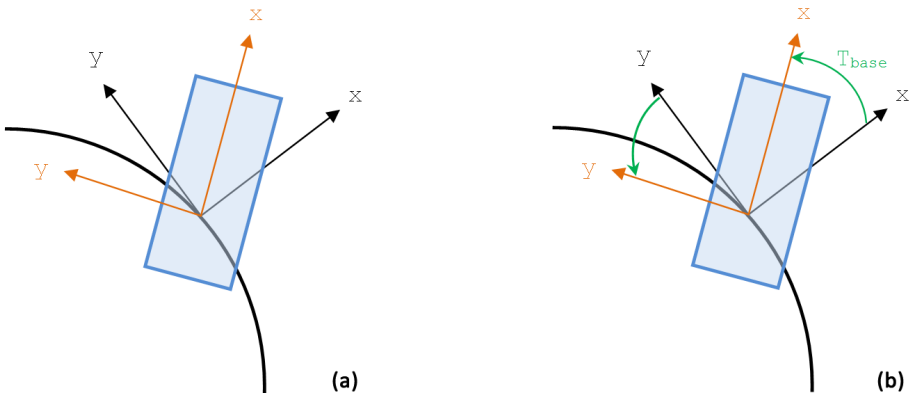



Figure 4.17: (a) Example where the coordinate system of a part (green) does not overlap with the coordinate frame of a control curve (black), (b) The transformation $T_{base}$ that maps the curve's coordinate frame to the part's coordinate system.

the circle is defined as y-axis, and the cross-product of y and z-axis as the x-axis, giving us the following matrix:

$$T_{curve} = \begin{pmatrix} \cdots & (Location_{Box} - center) \times normal & \cdots \\ \cdots & normal & \cdots \\ \cdots & Location_{Box} - center & \cdots \end{pmatrix}. \tag{4.12}$$

When looking at this formulation, it can be seen that for the circle $c$, this corresponds to the tangent (see Equation 3.10)

$$t = c', \tag{4.13}$$

the normal (see Equation 3.11)

$$n = \frac{\bar{n}}{\left\|\bar{n}\right\|}, \bar{n} = c'' - \langle c'', t \rangle t \tag{4.14}$$

and the binormal (or bitangent, see Equation 3.12)

$$b = t \times n. \tag{4.15}$$

This also gives us the solution for the local coordinate frame on a general curve, which is used in our implementation for Bezier curves and NURBS (for more details see Section 3.3.3):

$$T_{curve} = \begin{pmatrix} \cdots & t & \cdots \\ \cdots & n & \cdots \\ \cdots & b & \cdots \end{pmatrix}. \tag{4.16}$$

Using this definition for the transformation matrix $T_{curve}$ and the rotation matrix of the part $R_{box}$, the base-matrix $T_{base}$ that performs the transformation from curve frame to final position can be calculated:

$$T_{base} = R_{box} T'_{curve}. \tag{4.17}$$

## 4.5 Manipulation

As described above, the framework consists of two stages, model analysis and model manipulation, which is repeated until the user is satisfied with the result. In the model-analysis stage, we have calculated bounding boxes for each part and identified binary relations that connect the parts together. Symmetry groups were also identified by the framework and the user specified which type of control curve should be used and how parts should behave when the curve is changed. For all of these curved symmetries, the parameters were identified automatically by the system. In this section, we will show the manipulation stage and describe how user interactions are handled by the system, and how these changes are propagated through the whole model. At the beginning of each interaction step, the user modifies one part by manipulating its bounding box (Section 4.5.1), then the whole model is updated by using a propagation-based distribution algorithm, which is based on the work of Zheng et al. [ZFCO$^+$11] (Section 4.5.2). In this propagation scheme, the user modification is distributed using a breadth-first algorithm through the base graph. This propagation through the connectivity graph also modifies parts that are part of higher-order relations. Due to this changes, the control curves of these higher-order relations are modified. When all requirements for updating a higher-order relation are fulfilled, this relation is modified to fit into the changed environment. The update of higher-order relations is described in Section 4.5.4, where also the preconditions for each type of higher-order relation are explained.

### 4.5.1 User Manipulation

Users can manipulate the bounding box of parts in different ways. They can move them in space, scale them along their axes, or rotate them around their centers. When a part is selected by the user, it displays *handles* that allow the user to manipulate it. For translation and scaling, arrows are displayed on each side of the bounding box, for rotations, a ring-controller is displayed (see Figure 4.18).

In the translation and in the rotation mode, the manipulations made by the user can be applied directly to the mesh, e.g., moving a translation arrow some units away will move the model in the same way. When manipulating the mesh in scale mode, this is no longer working. The user should have the impression of moving one side of the box (see Figure 4.19), but applying the scaling to the part directly would not only move the chosen side, but also the opposite side. This problem can be solved by applying a translation factor ($t$) in addition to the scale factor ($s$). Assume that the user drags a scale arrow for a distance $x$ on the right side of the box. To decrease the complexity of the calculation, we transform all relevant values to the box space beforehand by using the inverse matrix of the normalized box axis

$$T_{World \to Box} = \begin{pmatrix} \cdots & Box_X & \cdots \\ \cdots & Box_Y & \cdots \\ \cdots & Box_Z & \cdots \end{pmatrix}^{-1} . \tag{4.18}$$

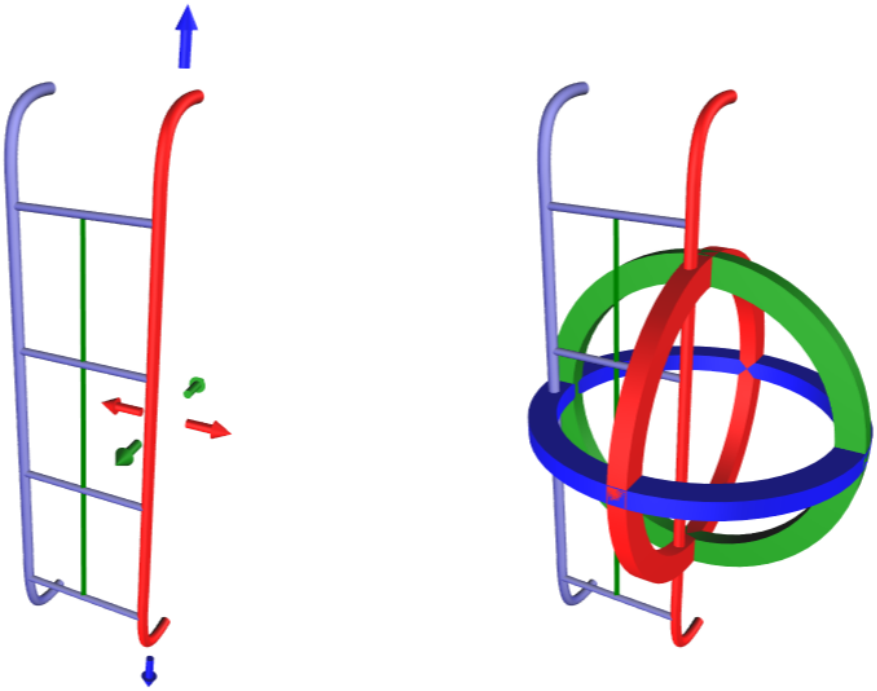


Figure 4.18: The user can manipulate a part of the model by dragging the arrows, or by rotation the ring controller.

In box space, the scaling and translation factors for the specific axis are given by

$$\begin{aligned} t &= \frac{x}{2} \\ s &= \frac{2 \cdot extent + x}{2 \cdot extent}. \end{aligned} \tag{4.19}$$

When these factors are applied to the bounding box, and to the underlying geometry, the modification will look, as if the user had dragged one side of the box. When the modification has happened on the opposite side of the box, the scaling factors stays the same, but the translation factor is given by $t = -x/2$.

### 4.5.2 Overview of Propagation Algorithm

Everytime the user has modified a part of the model, the other parts of the model have to be adapted to these changes. In general, there are two categories of methods that can achieve such results. On the one hand, there are methods that construct a global system of equations and minimize this system in a least squares sense. Examples for this are the work by Bokeloh et al. [BWKS11, BWSK12], Cabral et al. [CLDD09] und Xu et al. [XWY+09]. The major problem of these methods is that in the end, a large system of (often non-linear) equations has to be solved, which can, especially for large

Figure 4.19: Dragging a scale-arrow should look for the user as if he drags only one side of the box.

models, be a big performance bottleneck. The other class of methods tries to propagate changes starting from the modified source part through the whole mesh step-by-step (see [GSMCO09] and [ZFCO$^+$11]). For more details see Section 2.

Our approach follows the propagation-based idea: Starting from the part that the user has modified, we perform a breadth-first search through our base graph. In this graph, all parts of the model are stored as nodes, together with the binary relations that are described by the edges. The result of the breadth-first search is an ordered list of parts and binary relations that have to be updated. To avoid endless loops, we touch every object only once. The propagation starts at the first element of the list, and updates this object by keeping it as near as possible to the original, while fulfilling all constraints given by already manipulated objects. Since our parts are always connected by binary relations, we have two types of elements, parts and binary relations, that have to be updated. Every binary relation is constrained by the parts it connects and every part is constrained by the relations that are connected to it. This updating process is the basis for modifications to higher-order relations. As we already know, each higher-order relation is controlled by a parametric curve on which the parts are aligned. Due to changes to parts that are contained in such a higher-order relation, the control curve of this relation might change, and thus this relation has to be updated. We will show here first how the propagation through the base graph is handled, which updates parts and binary relations. Then, we give a short example of such a propagation process and show an extension of the used algorithm to support rotations. At the end, we show how higher-order relations are updated by the framework.

Our first implementation performed the propagation every time a part was modified, thus the propagation algorithm had to run multiple times during a drag operation. The big advantage of this was that the user had immediate feedback on how the model reacts. During the conception phase for the user study, we found out, that users have a lot of troubles with the delay that can happen in the propagation of more complex models. Due to this we added a second interaction mode, where propagation happens when the mouse is released or when the touch interaction ends.

### 4.5.3 Connectivity Graph Propagation

#### Binary Relation Modification

Binary relations are more easy to update than parts, since they always connect exactly two parts together. When a binary relation has to be updated, two cases can happen: The first case is that one of the connected parts is already updated, while the other is not. In the second case, both parts are already updated.

When only one part is updated, the optimal location ($l$) of the binary relation is calculated using the relation's type, the offset information ($o$) and the bounding box (axis $a$, center $c$ and extents $e$) of the updated part. Depending on the type of relation, different formulas have to be used: When the binary relation is a relative one, then the optimal location is given by

$$l_{i\in\{x,y,z\}} = c + a_i \cdot e_i \cdot (2 \cdot o_i - 1). \tag{4.20}$$

In case of an absolute relation this location is found by the following equation

$$l_{i\in\{x,y,z\}} = c + a_i \cdot (e_i - o_i) \cdot s, \tag{4.21}$$

where side $s$ is defined as

$$s = \begin{cases} 1 & \text{if bound to maximum side} \\ -1 & \text{if bound to minimum side.} \end{cases} \tag{4.22}$$

The final location can then be achieved by summing up the result of all three axis.

In the case where both connected parts are already updated, the best locations are calculated for each of the two parts using Formulas 4.20 - 4.22. The final result is then given by averaging these two values, which minimizes the error that is made by this calculation.

#### Part Modification

Due to the varying number of binary relations and the additional constraints, resolving parts is a bit more complicated than resolving binary relations. We construct a system of linear equations for each part, which holds all the necessary constraints for this object, and solve the system using a least-squares solver. Least-squares solving is necessary due to the fact that the constraints in the system could contradict, and we want to have the solution with the smallest error. The system of equations will be always solved for six unknown variables: Three translation factors ($t_{x,y,z}$) and three scaling factors ($s_{x,y,z}$). To keep the system uniquely solvable, we have to define at least six equations. In order to simplify the calculations, in this chapter we again transform all positions into the space of the box (see Section 4.5.1 for details).

As described in Section 4.2, for each axis of a part one can specify whether it should only move, or if it can stretch along this direction. Depending on this configuration,

different constraints have to be added to the system. In case of translation only, we first have to add an equation to the system to disable scaling and fix the scale-factor to 1

$$s_d = 1. \tag{4.23}$$

For each already updated binary relation connected to the part, an additional equation has to be added, that constraints the box to move in a way such that the location of the binary relation fulfils the relation type and the offset

$$relationBox_d + t_d = relationLocation_d. \tag{4.24}$$

In this equation, *relationLocation* is the location of the binary relation and *relationBox* is the location where the binary relation would be located in the current configuration of the box. *relationBox* can be computed using the formulas for optimal binary relation locations in Section 4.5.3. Note, that all of the equations in this chapter can be used along each axis of the bounding box. Thus $d$ can be replaced by $x$, $y$ or $z$.

When not only translation but also scaling is allowed along an axis, we have to specify a constraint that transforms the box in location and size so that the binary relation constraints again fulfil the current location of the binary relation

$$relationBox_d \cdot s_d + t_d = relationLocation_d. \tag{4.25}$$

It can easily be seen that if only one binary relation is updated, the scaling factor would never be used, since translation only would also lead to an error of zero. In most cases, this is not what we want, so if there is just one updated binary relation, we have to add more constraints to the system. The same situation occurs if all the binary relations present in the calculation are grouped on one side of the box. This happens when there are only absolute relations bound to the maximum and relative relations with a offset greater than 0.5, or if all absolute relations are bound to the minimum and all relative relation offsets are below 0.5. The best-working solution we found to overcome this problem was a similar approach to the one used for the box modification: We constrained the opposite side such that it tries to keep its location. Opposite side is here defined as the side where no binary relations are located. Since we are calculating in box space, this side can be found for each main axis separately. The equation that has to be added is then

$$extend \cdot side \cdot s_d + t_d = extend \cdot side, \tag{4.26}$$

where *side* is defined as in Equation 4.22.

When the system of equations is fully set up, it can be solved by any appropriate solver, returning the translation and scaling factors that should be applied to this part.

**Example**

To better visualize the propagation algorithm, Figure 4.20 shows an exemplary propagation through the ladder model, where each step shows the propagation for one depth layer in the breadth-first list. The user drags the part marked in red from the input

model (1) a bit to the left (2). According to the breadth-first search in the connectivity graph, the next elements that have to be updated are the four binary relations (yellow boxes) connecting the steps of the ladder with the modified part. Since these binary relations are at this time only constrained by the red part, they are translated to lie again on this part (3). In step (4), the four steps are adapted to fit to the already modified binary relation from step (3). Due to user configuration, the steps can only be translated, thus they are moved to the modified binary relation positions. After adapting the steps, the binary relations located on the left side also have to be modified. In step (5), these relations are moved to be located at the left side of the corresponding step. The last adaptation that happens is the adjustment of the left-most part, which has to fit to all four binary relations. The final model after all steps is shown in sub-figure (6).

**Extension to rotations**

Although this is not implemented in our framework, in some cases it might be helpful if parts could not only be moved and stretched, but also be rotated. We will give a short overview on how the propagation schema has to be adapted. For simplicity, only the two-dimensional case is shown, but extension to 3d would work in an analogous way.

In our implementation, the systems of equations for parts always search for six unknown parameters (three translations and three scaling factors). In 2D, we would have the following system of equations for each binary relation, where we are searching for two translations and two scaling-factors:

$$\begin{pmatrix} relationBox_x \\ relationBox_y \end{pmatrix} \begin{pmatrix} s_x \\ s_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} relationLocation_x \\ relationLocation_y \end{pmatrix}. \tag{4.27}$$

If the number of updated binary relations is too low for solving the system of equations uniquely, the unconstrained sides of the box are fixed, as also done when no rotations are available. When the system should be able to handle rotations, in the 2D-case, we get one additional unknown variable, the angle $\alpha$ that specifies how much the box is rotated. To include rotations in the system of equations, we have to multiply the whole equation by a rotation matrix $R(\alpha)$:

$$\begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \left[ \begin{pmatrix} relationBox_x \\ relationBox_y \end{pmatrix} \begin{pmatrix} s_x \\ s_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \right] = \begin{pmatrix} relationLocation_x \\ relationLocation_y \end{pmatrix}. \tag{4.28}$$

Since we are now using sine and cosine terms in our system of equations, we end up with a system of non-linear equations that is ways harder to solve than our previous system.

The second and more important problem that comes up when using rotations is, that now multiple solutions for the same binary relation locations would perfectly satisfy the constraints. In Figure 4.21 two solutions with equal quality are given for an input situation. Producing different results for the same input situation is, when working with users, never a good idea. In the optimal case, the system should provide exactly the solution that the user expects. When thinking about the 3d situation, the problem gets

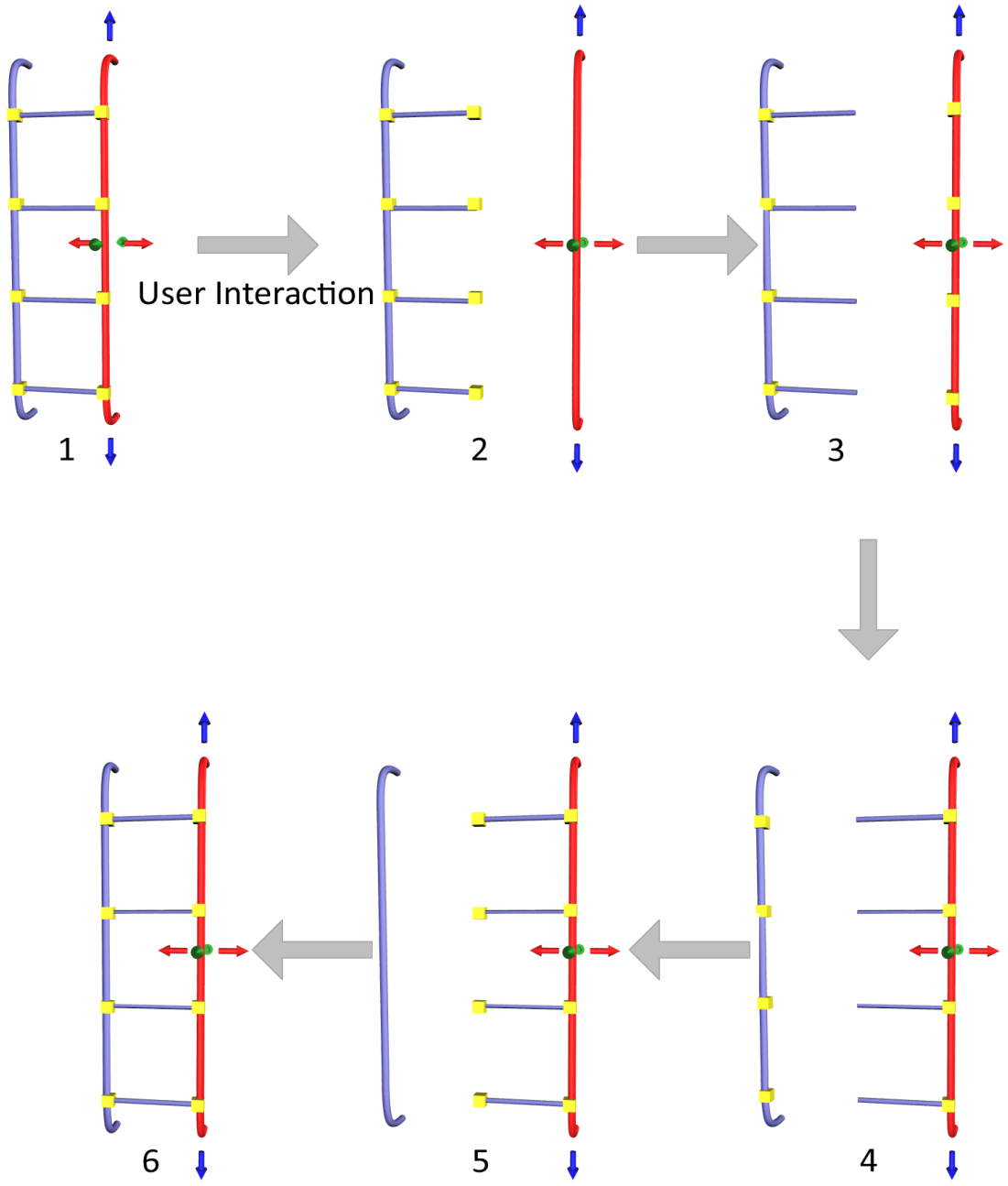



Figure 4.20: Step-wise example of the propagation algorithm.

even more complicated, since the third dimension further increases the number of possible solutions.

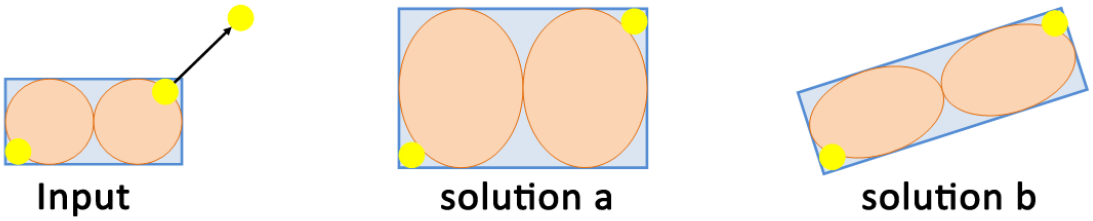



Figure 4.21: With rotations enabled, different solutions can be produced for an input situation.

The best idea we came up for this problem is to restrict the system in a way that we are not searching for 5 unknown variables any more. For example, we could leave out the whole translation term, because with rotations and scaling alone, every two fixed binary relation situation can be solved. When there are more than two already updated relation, the whole problem vanishes, since we then have 6 equations for five variables, and the system can be solved exactly.

### 4.5.4 Higher-Order Relation Propagation

The higher-order relations that were identified in Section 4.4.1 also have to be evaluated and updated in the modification process. During the propagation of parts and binary relations, the position of parts that are contained in higher-order relations are modified. According to these changes, the control curve is adapted, and the parts have to be repositioned. Changes already made to parts by the propagation algorithm are overwritten by the higher-order relation adaptation. During the propagation process, the system checks after the updating of each part if there are any higher-order relations that can also be updated. When the pre-conditions, which are specific to each type of control curve, are fulfilled, then the higher-order relation is updated immediately.

The steps that are necessary to update a higher-order relation depend strongly on the control curve that is used, but the general workflow is always the same: First, the control curve is updated and afterwards, the constraints that are defined for this element are enforced.

The simplest control curve for updating is the line controller: Since a line is always uniquely defined by two points, this controller is updated when both the first and the last element of the parts it handles are updated. The new line is then defined by the vector between these two parts.

Bezier curves and NURBS are updated when the propagation through the whole model is finished. For Bezier curves, updating consists of setting the first control point to the position of the starting part, and setting the last control point to the position of the ending part. Due to the global control of Bezier curves, modifying only the first and

the last control point influences the whole curve, thus giving more plausible results when modifying it. NURBS, in contrast to Bezier curves, have only local influence, which is a big advantage in most cases (e.g. designing meshes). In our application scenario, this means that moving the starting part of such a curve will only modify a small part of the curve, which can look implausible to the user. This behaviour can be seen in Figure 4.22 (top line), where only the last node of a NURBS curve of order five is transformed.

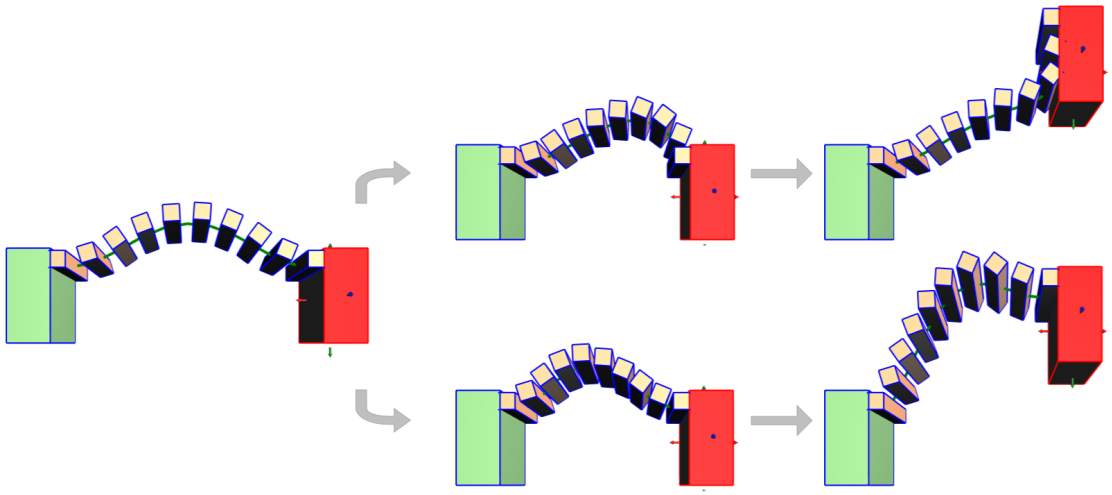


Figure 4.22: Different algorithms for updating NURBS. Top: Only the last control point is adapted, Bottom: All control points are adapted with a weighted motion vector.

Since we always want to produce results that the user expects, we have to come up with a more complex algorithm: When one end of the curve is moved, we determine the motion vector ($t_{motion}$), which is the vector between the last position of the corresponding part and the new location. All control points ($c_i$) of the NURBS curve ($c$) are updated by adding the weighted motion vector to them. The weighting factors are determined by the square root of the normalized curve distance to the modification point:

$$\begin{aligned} c_i &= w_i \cdot t_{motion} \\ w_i &= \sqrt{\frac{curveDistance(modified, i)}{curveLength(c)}}. \end{aligned} \tag{4.29}$$

For the definition of *curveDistance* and *curveLength* have a look at Section 3.3. Using this modification behaviour, the whole curve is influenced when one end is moved, as can be seen in the bottom line of Figure 4.22, giving better results for the user.

The circle controller can be updated when all elements it controls are updated. Since it is possible that some parts are never touched by the propagation algorithm, this controller is also updated when the propagation is finished. The algorithms for updating

the circle are the same as for the detection (see Section 4.4.1). This results in a circle that has the smallest error for a given set of modified parts.

After updating the controllers, the constraints on these controllers have to be applied. There are two types of constraints that are implemented in our framework: reposition parts on curve and repeat.

### Position Element on Curve

This class of constraints ensures that all parts that are controlled by a curve are always positioned evenly spaced on this curve. Repositioning can, depending on the settings the user chooses, only modify the position of the elements, or it can also orient the parts in the Frenet frame of curve. When updating the positions, we have to find points on the curve that are evenly spaced. Since NURBS and Bezier curves are not always parametrized by arc-length, especially when multiple curves are fitted together, the curve parameter has to be calculated from the curve length. The mathematically exact solution for calculating the distance on a curve is to calculate the following integral:

$$length(c) = \int_a^b c'(t)dt. \tag{4.30}$$

In most cases, such integrals are hard to solve, and not suited for real-time applications. Our solution is to calculate the position on the curve with small step size and store an array containing the length between consecutive polygon points. This gives us a piecewise linear polygon that approximates the curve. When the length between two parameters is searched, we only have to look up the two positions in the length array and sum up the length between them. In most cases, the start and end limits will not fall exactly on a polygon point, thus the first and the last position is calculated on the polygon by linear interpolation.

Updating the position itself is then done by querying the curve at the so found parameter, and moving the center of the part's bounding box to the curve position. Orientation is a bit more complex: First, the local frame of the curve ($T_{curve}$) is evaluated at the curve parameter where the part should be located. This is done in a similar way as in the base-orientation detection (Section 4.4.1). When applying the orientation to the part, we first have to extract the scaling matrix ($Box_{scale}$) from the part's transformation matrix, which is given by the length of the rows in the matrix. The resulting matrix is a matrix where the scaling factors are located at the diagonal elements. The final transformation is then calculated by

$$T_{final} = Box_{scale} \cdot T_{base} \cdot T_{curve} \cdot Box_{rotation}. \tag{4.31}$$

$T_{base}$ is the base transformation matrix that was calculated in the detection stage, allowing the parts to have another alignment than the curve frame. The difference between modification with orientation and without orientation can be seen in Figure 4.23, where the same user input is updated without orientation adaptation on the left side, and with orientation adaptation enabled on the right side.

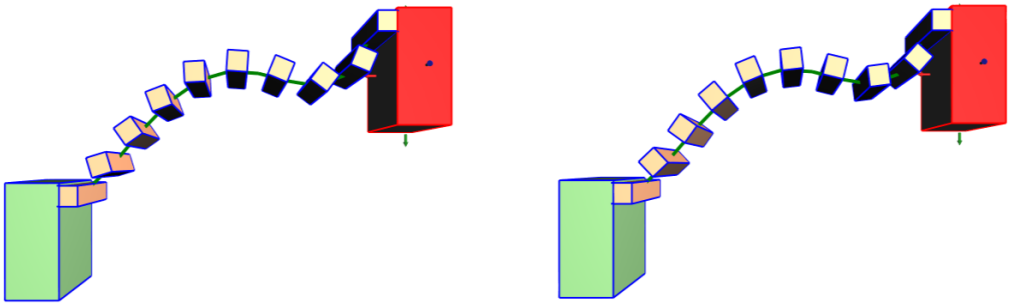


Figure 4.23: NURBS curve modification, without orientation adaptation (left) and with adaptation enabled (right).

### Repeat Elements on Curve

In the previous section, the parts that are controlled by a curve were moved according to the changes of the curve. But regardless of how the curve is transformed, the number of parts stays the same. The second constraint class solves exactly this problem: According to the length of the curve, new elements are inserted and old elements are deleted. In addition to the base orientation matrices, this constraint holds a scalar value that describes the optimal distance ($d$) between elements on the curve. Whenever the curve changes, the constraint first checks the new length of the control curve, and then calculates the optimal number of parts that should be placed on it:

$$OptimalCount = round\left(\frac{CurveLength}{d} + 1\right). \tag{4.32}$$

When the optimal element count is not the same as the actual element count, this group has to be adapted. In the case where less elements are required, elements are removed from the model. To keep the curve well defined, the first and the last element are never removed. In the other case, where elements have to be added, this is done by copying one of the currently present elements. For such newly created elements, an additional binary relation detection is performed, since these elements should also be connected to the rest of the model.

Note that every time the number of objects is changed, an update to the base graph is required, since new binary relations have to be added, and deleted ones have to be removed. In addition, the layer that corresponds to this symmetry group has to be updated to reflect the new ordering along the curve. After updating the element count, the workflow is the same as in the positioning case: For all elements, the new positions and optionally the orientations are evaluated, and the elements are moved on the curve. Figure 4.24 shows the same modifications as in Figure 4.22, but this time with repeating enabled.

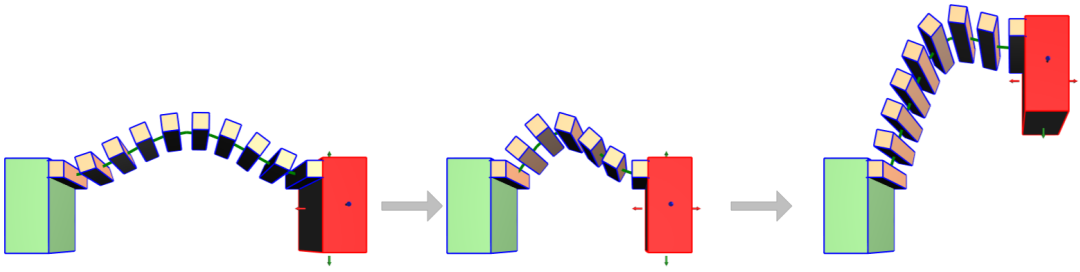


Figure 4.24: NURBS curve modification with repeating and orientation adaptation.

## 4.6 Interaction

We implemented two input methods for manipulating models in the framework. The first input device that can be used is the mouse: Users can rotate the camera around a fixed center via clicking and dragging the right mouse button. The rotation center can be changed by holding down the SHIFT-key and clicking and dragging the right mouse button. When scrolling with the mouse wheel, the camera moves closer to the object, or further away. This type of camera is commonly called orbit camera and is similar to the camera model implemented in most 3d modelling tools, e.g., Blender and Maya. Mathematically speaking, the camera is located on a sphere around the center point, where the azimuthal angle ($\theta$) and polar angle ($\varphi$) can be modified as well as the radius ($r$). Together, these three parameters form a spherical coordinate system $(r, \theta, \varphi)$, which is shown in Figure 4.25.

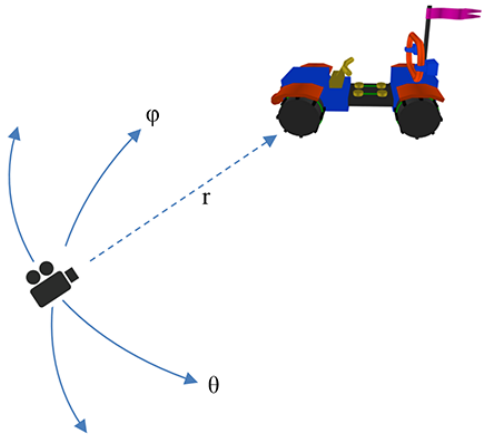



Figure 4.25: The orbit camera is located in the spherical coordinate system $(r, \theta, \varphi)$

When using the touch input, our second interaction method, the camera is modelled in the same way, but the interaction is a little bit different: When one finger touches the screen and is moved, the angles of the camera-coordinate system are changed, thus

rotating the camera. The center point can be moved by touching the screen with two fingers and moving them parallelly. When the distance between the two fingers is changing, the camera is zooming.

Interactions with the model are quite similar in both interaction methods. The user can select parts by either touching or left clicking them. The handles displayed can then be moved by a drag-operation with the left mouse button, or with one finger on the touchscreen. During the user-study it turned out that this interaction of first selecting and then moving with arrows did work, but could be improved. For example, some users suggested that moving parts by click-and-drag directly would be more intuitive, especially with a touchscreen. In addition, scaling could be done by touching the part with two fingers and moving them away from each other, which would also allow for rotations. For more details about how well mouse and touch interaction were accepted by the users, have a look at the results of the user study in Section 5.2.

CHAPTER 5

# Results and Evaluation

This section describes the implementation of our framework. After this, the user study is shown that we did at the end of our work to check if untrained users can handle the system. After this, selected results of our framework are presented, together with example application scenarios where the framework could be used. In addition to this, we will also describe the limitations of our current implementation and show some performance measurements.

## 5.1 Implementation

All the algorithms described in this thesis are implemented in a C#/WPF (Windows Presentation Foundation[1]) framework. For some specialized tasks, like rendering, we rely on external libraries, which are described here briefly. In general, our implementation follows the Model-View-ViewModel (MVVM) software-engineering pattern that Microsoft recommends to use when writing WPF applications for Windows. Even though WPF already has a very good 3d rendering support in its core implementation, we decided to use the Helix 3D Toolkit[2]. This framework is a high-level abstraction of the DirectX API, which is exposed to C# via the SharpDX library[3]. The reason why we preferred the Helix Toolkit over WPF 3D is that it has fewer limitations in terms of, e.g., number of objects that can be drawn at the same time. Besides this, the Helix Toolkit has more technical flexibility for possible future work.

For most mathematical tasks, the framework uses the MATLAB API for performing the calculations, which is especially helpful in the development process, since it offers very good debugging support. Another big advantage of MATLAB as computation backend is its library of fully tested, high-performance algorithms. Using Matlab allowed us to try

[1] http://msdn.microsoft.com/de-de/library/ms754130(v=vs.110).aspx
[2] http://helixtoolkit.codeplex.com/
[3] http://sharpdx.org

out different algorithms and ways of solving problems, without having to search for a new library that supports the required functionality everytime. Although the Matlab-API performs very well, we may in future replace it by other libraries, since the data transfer between our application and Matlab is becoming more and more a bottleneck. Besides this math framework, our program uses an external library for handling NURBS cures. The SISL library[4], which is developed by the SINTEF group[5], an independent, non-commercial organisation, can perform everything required for our implementation, starting from curve fitting, over curve complexity reduction, to measurements and Frenet frames. Since this library is written in C, we implemented a wrapper, that exposes the library's functionality to C#.

Parts of the binary relation detection, especially the bounding-box creation and the oriented bounding-box trees, were developed during the Master's thesis project by Sebastian Sippl at the Vienna UT. The library written by him offers the possibility to intersect two bounding box trees and their underlying geometry and returns a list of contact points, which we use as input to our binary relation reduction algorithm.

Up till now, we have only seen how the algorithms in our framework are working. On top of these algorithms, our system also provides the user with a user interface for configuring the input model (see Figure 5.1). For the user-study (see Section 5.2), we added an additional interface that only contains the elements for working with the model, but hides all configuration interfaces (Figure 5.2).

## 5.2 User Study

The user study was designed following the principles given in Section 3.4. Our main goal was to test if inexperienced users are able to work with the system. Beside this, we also wanted to see if the users like the mouse interaction or the touch interaction better. Some tasks and questions were also designed to check if users like Bezier curves or NURBS with our adaptation method more.

We performed the user study on a SONY notebook with touchscreen. When a user starts the study, we first introduce him to the system by explaining the interaction methods via mouse and touchscreen first. For this, we used the very simple ladder model, where also the basic behaviour (adaptation to the user input) was shown. The study itself consists of nine small tasks, ranging from basic interactions to complex models.

**Task 1** introduces basic selection and translation behaviour to the user by giving a simple task, where an object should only be moved. **Task 2** adds resizing to the interaction. Both tasks can be solved without turning the camera. In **Task 3**, the user has to rotate the camera for the first time, otherwise the task is not solvable since the objects in the scene are positioned behind each other in the initial situation. These basic

[4] http://www.sintef.no/Informasjons--og-kommunikasjonsteknologi-IKT/Anvendt-matematikk/Fagomrader/Geometri/Prosjekter/The-SISL-Nurbs-Library/SISL-Homepage/

[5] http://www.sintef.no/

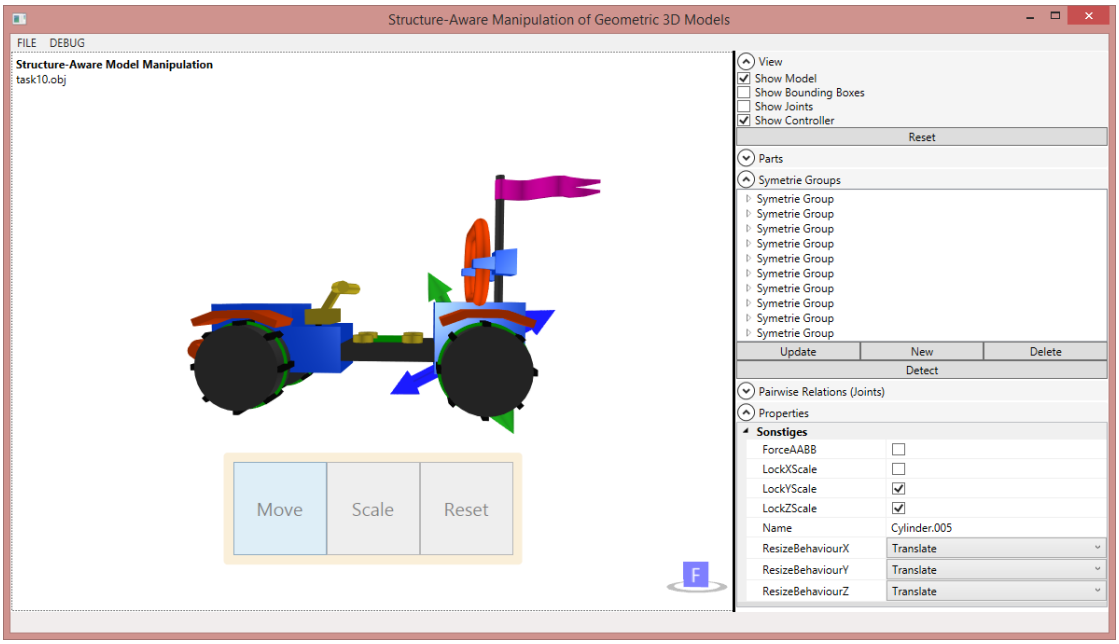



Figure 5.1: Screenshot of the Structure-Aware Manipulation of Geometric 3D Models framework

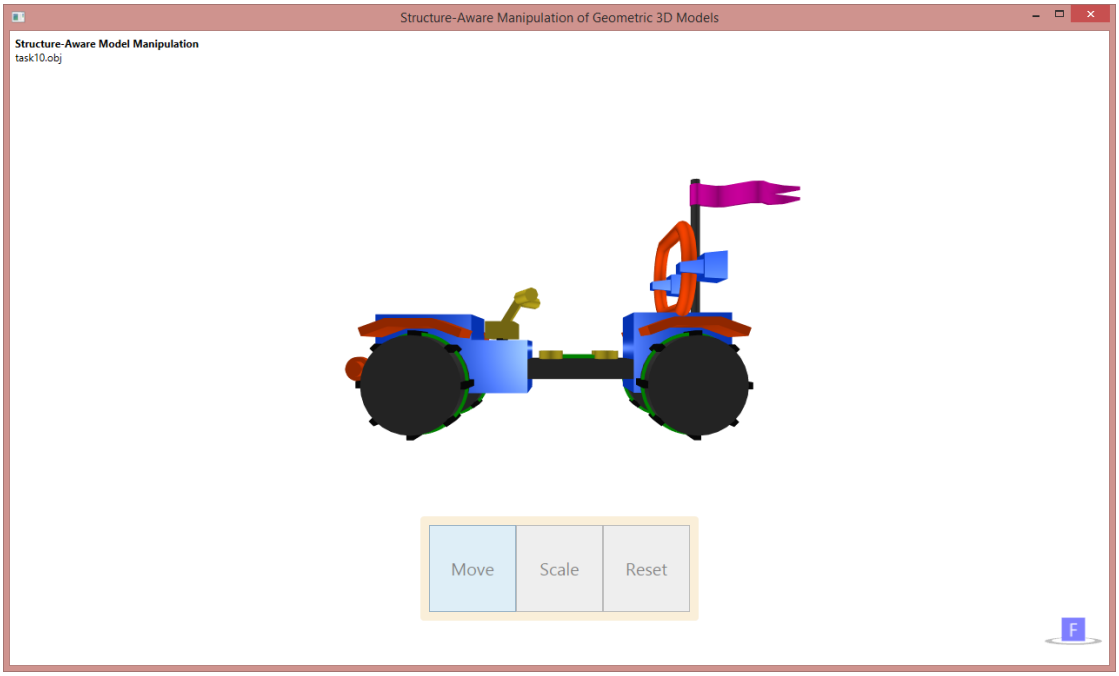



Figure 5.2: Screenshot of the user study mode.

tasks (Task 1-3) are only designed to teach the user how the system can be controlled. Due to this, the performance results of these tasks have not been used in our evaluation.

Starting with **Task 4**, the user has to work with more complex models, consisting of multiple connected elements and some higher-order relations. These tasks are designed to always test one specific feature. Task 4 presents line control curves to the users, where the elements (the red spheres, see Figure 5.3 left) are repeated when the user enlarges the model. **Task 5** also tests repeating behaviour, but this time, a circle control curve is used, and the user has to manipulate the view to see the results of his interactions (see Figure 5.3 right). In **Task 6** and **Task 7**, the same input model is given to the user with the same task to perform, but one time the elements are connected via a Bezier curve and the other time using a NURBS curve. For these two tasks, the user has to choose which type of behaviour seems more intuitive to him. This question is mainly asked to find out, whether it is necessary to support the more complex NURBS, or if Bezier curves alone would be sufficient. In the last two tasks, **Task 8** and **Task 9**, the users has to deal with complex objects. In these tasks, the user had to modify an input model such that it looks similar to a target model. The goal here is to test if the users can figure out how to modify complex models to get a specific result. The models used for those tasks are shown in Figure 5.4, together with the target model that should be modelled.

Figure 5.3: For Task 4 and Task 5, two models with repeatable control curves were used. In Task 4 (left), the red spheres are aligned on a line, in Task 5 (right), the purple boxes are aligned on a circle. The task here is to modify the models, until the number of repeatable parts has changed to a specific number.

Every user had to perform all tasks twice, one time using the mouse controls, and one time with touch interaction. The order in which mouse and touch interaction was given, was randomized for every user. Also, the order in which Task 6 and Task 7 were shown was randomized to ensure valid results. After performing each task twice, the question "Did you prefer mouse interaction or touch interaction more for this task" had to be answered. Furthermore, the time it took the user to solve the task was noted and the number of retries was also counted. After performing all tasks, five general questions were given to the users, on the one hand to check for their prior knowledge in 3d modelling and touch interaction, and on the other hand to get general feedback about the system. The questions that were used are the following:

1. How much experience do you have with touch devices (tablets, notebooks/PCs,

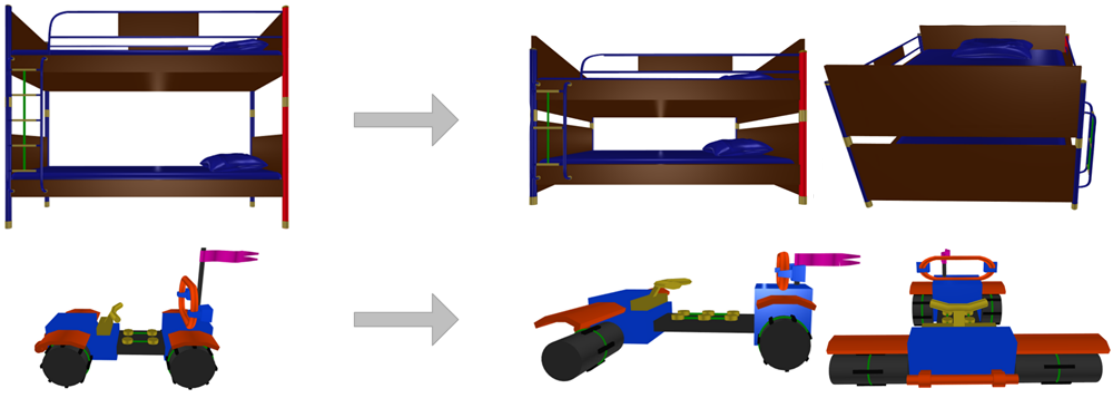


Figure 5.4: Task 8 (top) and Task 9 (bottom) were used to test if complex models can be handled by the users. In both tasks, the user had to modify the input model (left) to produce the target model (right).

mobile phones)? Answers ranging from 5 (very much) to 1 (none)

2. How much experience do you have with modelling applications (Blender/Maya/AutoCAD)? Answers ranging from 5 (very much) to 1 (none)

3. How much do you like the framework? Answers ranging from 5 (Liked it a lot) to 1 (Don't like it)

4. After all tasks: Would you prefer to use the mouse input or the touch input? Possible answers: Mouse only - Mouse more preferred - Both equal - Touch more preferred - Touch only

The detailed questionary for users together with the task descriptions can be found in Appendix A.

### 5.2.1 Results

We performed the user study with 11 participants, ranging from users that have absolutely no experience with modelling systems to users that use 3d modelling in their daily work (6 of the 11 participants answered the question about their modelling experience with three or higher). The same holds for the experience regarding touch input, where five of the 11 users had experience of three or less. Not surprising, this prior knowledge leads to very different results in the choice of the preferred input method: Users that have high experience with modelling tend to choose the mouse input more often (68.52% over all tasks) than users without prior knowledge (57.78%), see Figure 5.5 (left). This might be due to the fact that the trained participants use the same mouse input method as in our task also in the modelling tools they use in their work. Similar observations can also be made about the experience with touch input devices: It is more likely that experienced

users feel comfortable with the touch interaction (only 35.56% preferred mouse input over all tasks) while users without touch input experience tend to choose mouse input more often (55.56% preferred mouse input over all tasks), which is shown in Figure 5.5 (right).

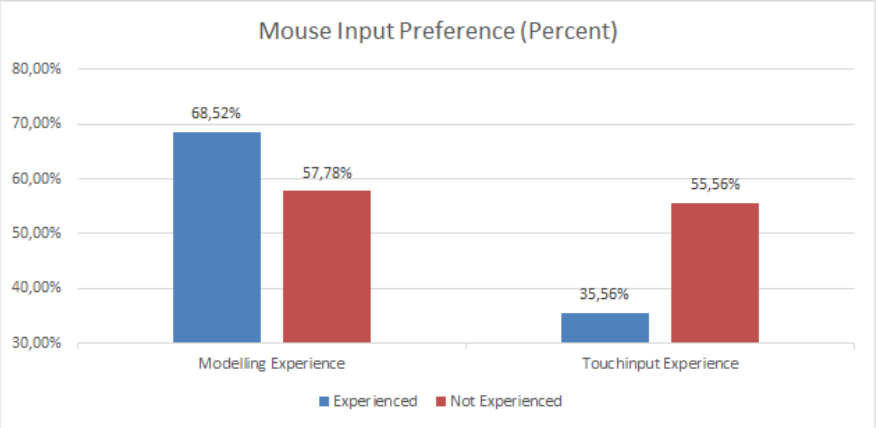



Figure 5.5: Users that have prior knowledge in modelling tend to select mouse input more often. The same holds for users without touch experience.

One of the most interesting facts we found out during the user study is that the acceptance of touch input gets higher the more complex and free the tasks are (see Figure 5.6). In the first three tasks, where the user had to move an object as exactly as possible to a target, most users prefer the mouse input, while in the last two assignments, where precision is not that important, the gap between mouse and touch input preference is not that high. This fact was also confirmed by the users in their comments, where they told us that touch interaction allowed them to work more easily in tasks where approximate results are desired, but mouse input allowed them to manipulate the objects more accurately.

The general low acceptance rate of touch input over the whole user study is the biggest problem that we currently have with our system, and might also be the reason why no user chose the "Touch more preferred" or the "Touch only" choice in the last question (see Figure 5.7 left). According to user comments, there are two major reasons for that: First, the touch interaction with the arrow controller did not work out very well for the users. Especially participants that have a lot of touch input experience found it very unintuitive that they had to move an object by dragging the arrows instead of dragging the object itself. Scaling would, according to the participants, also be more intuitive if it could be performed by touching the object with two fingers and moving them away from each other. The second reason is that for users with less touch experience, the latency introduced by touch input devices felt unintuitive. Since this is a problem that would occur with every application when it is used by such users, this is not considered as problem by us.

The more promising results of the study are that all eleven users were able to accomplish all tasks, although we did not answer any questions regarding the way how

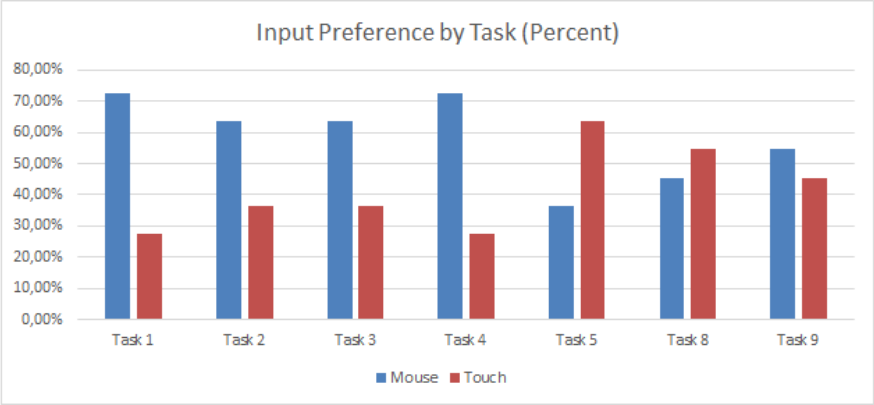



Figure 5.6: For tasks where exact modelling is necessary, the touch input acceptance is much lower than for tasks with more freedom.


Figure 5.7: Histogram of the answers of the last two questions.

to solve the assignment during the study. Nearly all users were also able to complete the tasks without resetting the model (only two users used this option in 5 assignments in total). We interpret this as an indicator that the framework is very stable at the moment and allows the users to solve their problems within the system. This is also confirmed by the users who told us that they were able to predict very easily how models will react to their interactions after the first three tasks. While performing the tasks, the learning effect was clearly visible and can also be seen in the time measurements (Figure 5.8). When users perform a task for the second time, they are (with few outliers) able to complete the task much faster. On average, the users are twice as fast in the second try (red bars) than they were in the first try (blue bars).

The question whether Bezier curves or NURBS with scale adaptation are better suited for our system did not lead to a significant preference: 6 users chose Bezier curves, while 5 users selected NURBS curves. Since it seems not make any difference for the user if Bezier curves are used instead of NURBS, we might drop the support for NURBS curves in the future.

In general, our feeling during the study, which is also shown in the results of the "How

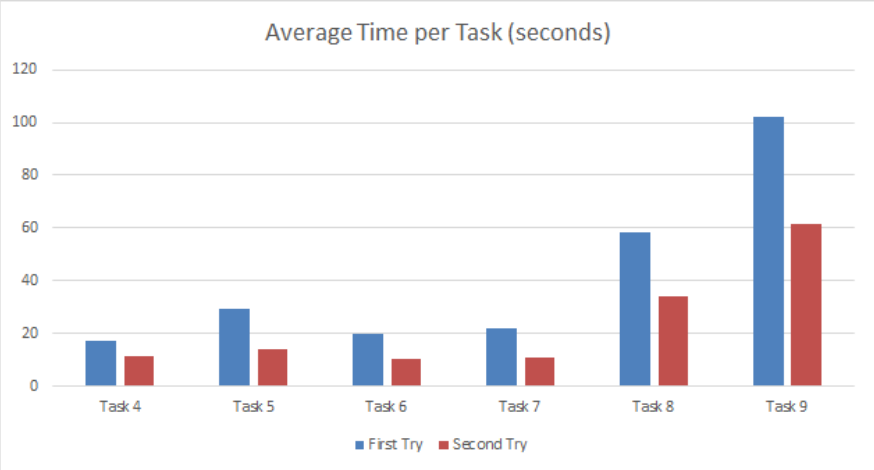



Figure 5.8: Average time it took the users to solve the tasks. Blue bars show the first attempt, red ones the second try.

much do you like the framework?" question (see Figure 5.7 right) is that the system is very well accepted by the users, regardless of their prior knowledge. This shows, that our way of designing such user-based systems is clearly on the correct path.

## 5.3 Results

We have implemented the presented techniques in a framework for structure-aware model manipulation, which allows the user to modify a given input model while the systems maintains and adapts shape features. To test our system, we designed and segmented models from the internet and from related-work papers. We specified five test models with varying complexity, which can be seen in Figure 5.9 and are described in Table 5.1. Since we have not implemented an automatic shape-segmentation method, all these models were manually segmented before they were used in our system. For all these models we were able to obtain good looking editing results without holes or other errors, as demonstrated in Figure 5.10. Our shape-analysis stage is able to identify the symmetries in all models correctly (although in some cases, the objects with the same generating transformation had to be selected manually, see Section 4.3.3).

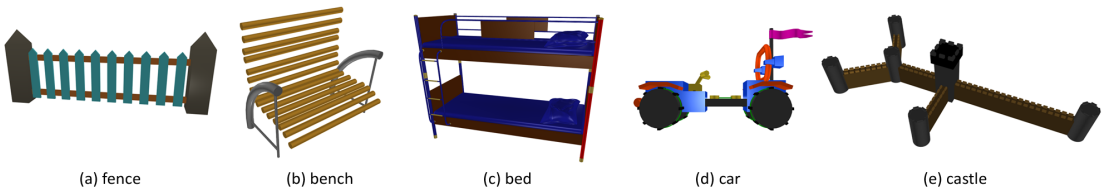



Figure 5.9: Test models used in our framework.

| model | parts | vertices | symmetries | symm. detection | binay relation detection |
|---|---|---|---|---|---|
| fence | 14 | 134 | 3 | 0.04 sec. | 0.01 sec. |
| bench | 21 | 1944 | 3 | 0.23 sec. | 0.20 sec. |
| bed | 48 | 14230 | 2 | 1.53 sec. | 2.44 sec. |
| car | 62 | 1387 | 6 | 0.38 sec. | 0.45 sec. |
| castle | 192 | 2096 | 19 | 2.02 sec. | 1.77 sec. |

Table 5.1: Performance statistics for our test models. In addition the number of parts, the number of vertices and the number of symmetry groups in the model are shown.

### 5.3.1 Performance

For most input models, our model analysis stage, which detects symmetry groups and binary relations, runs in a few seconds. Since the detection of symmetry groups is only necessary when starting to work on a new model, we did not optimize these algorithms. The binary relation detection, which is called whenever the model is changed, is at the moment the biggest bottleneck in our system when working with models that have a large number of parts. Time measurements from our test system (Intel i7x64 with 1.8GHz dual core, 16GB memory, NVIDIA Geforce GT 735M) are shown in Table 5.1.

Another and more general performance problem is the usage of Matlab as mathematical back-end. When a lot of data has to be transferred between our application and Matlab, this can stall the whole application for some time. At the moment this can only be noticed when working with small models, since with large models the binary relation detection itself is too slow.

### 5.3.2 Limitations

Although our framework for structure-aware model manipulation produces very good results with a lot of input models, there are some limitations that should be mentioned: The first and probably most important one is, that at the moment no segmentation stage is implemented, thus our framework only performs well when the model is pre-segmented by the user. A similar problem arises with the detection of curve-symmetric relations. At the moment, only the detection of symmetric parts is done automatically, while the type of control curve as well as the behaviour of the parts in these groups have to be specified by the user.

During our development, it turned out, that the propagation-based algorithm itself is also a limitation of our system. It is very hard to add global constraints like reflective symmetry between multiple parts in this formulation. The second problem is that the propagation algorithm can fail in some situations. Failures mainly happen when there are circles in the connectivity graph and the parts in the circle have different resize behaviours. An example for a propagation where the algorithm fails is shown in Figure 5.11. Here, the boxes are connected to the adjacent spheres. Green boxes indicate that they can be scaled while the blue box can only be translated. In the step-wise propagation, it can


Figure 5.10: Editing results of the test models. The original model is always shown in red.

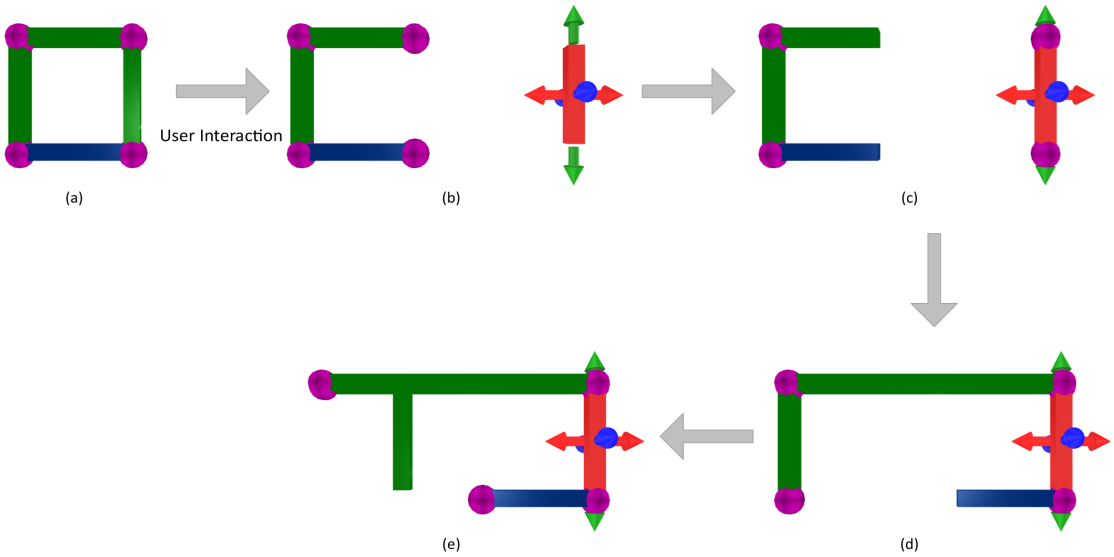



Figure 5.11: Step-wise visualization of a situation where the propagation fails. Green boxes can be scaled while blue boxes can only be translated. When the user modifies the model (b), the upper and the lower box are adjusted differently (d), and the last iteration can not be solved since the binary relations contradict (e).

be seen that in step (d), the upper and the lower box are adjusted differently due to their configuration. In step (e), the two binary relations for the left box contradict, since there is no configuration for this box where it can touch both adjacent spheres. Since the system always tries to minimize the error that would be introduced by an editing operation, the box is placed in the middle between the two spheres.

Binary relation detection can also be problematic when there is more than one binary relation required between two overlapping objects. When, for example, one box should propagate changes to its size to the next box, it is necessary to have at least four binary relations, two for each direction. At the moment, our connected-component search would only find one relation for this situation. Figure 5.12 shows in 2d how an object reacts when there is only one binary relation (left), or when there are two of them (left).

## 5.4 Applications

Structure-aware model-manipulation systems can be used in a wide variety of applications. As shown by Schulz et al. [SSL+14], one possible scenario is to support users by designing models that can be fabricated later on. Especially with the growing availability of 3d printers for untrained users, such systems become more important. Since these users will not be able to generate models by their own, application-supported modification of models from internet databases will allow them to use this hardware efficiently.

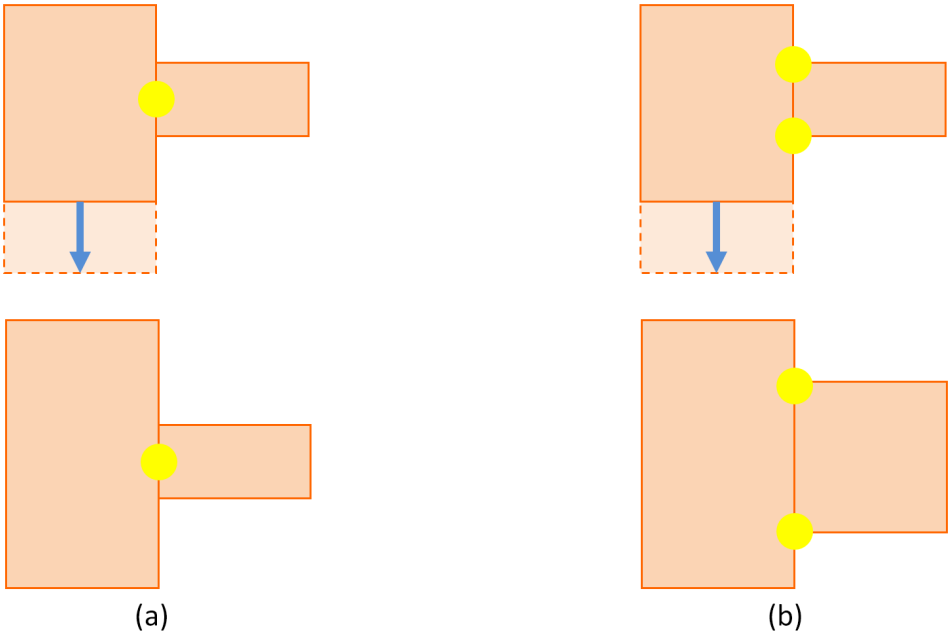



Figure 5.12: Models can behave differently depending on the choice of binary relations. (a) When only one relation is present (as in our implementation) changes in size are not propagated to nearby objects. (b) Using two binary relations make it possible to do this.

Another field where frameworks like ours can be used are games. When thinking for example about the next Sims-like game, it would be interesting to allow the user not only to place furnitures in their homes, but also give them the ability to modify these models. A wardrobe, for example, could be stretchable and adjust the number of doors it has to the size that the user specifies. Other games like roller-coaster construction games could also benefit from these methods. But there are not only opportunities for the gamers through such a system: Artists and level-designers could also take advantage of such a system. These two production stages are at the moment the most expensive parts when producing a AAA-game. Using structure-aware methods, this work could be sped up, thus reducing the costs.

CHAPTER 6

# Conclusion and Future Work

## 6.1 Synopsis

In this thesis, we presented a framework for structure-aware model manipulation. The framework consists of two main stages: analysis and modification, and we contribute to both of them. A model in our framework is a 3d geometric shape that is composed of parts that are connected by one or more relations. We identified two types of such relations, binary relations, which describe physical connections between parts, and higher-order relations, which are used to form symmetry groups. In this work, we developed a new algorithm for finding well-defined binary relations in 3d models by pairwise intersecting the geometry contained in the parts. This process is sped up by using oriented bounding-box trees for faster intersection testing. The resulting point cloud of overlap points is then refined using a greedy algorithm, which finds connected components and describes each component by a binary relation. Higher-order relations are found by an iterative closest-point algorithm (ICP), where a pre-transformation step is used that rotates the part until the axes of the bounding boxes overlap. This transformation improves the quality of the found results, since less situations are produced where stable forces appear in unmatched state.

For handling higher-order relations, we presented a novel way of defining symmetries on curves. This formulation covers a wider range of identifiable patterns in 3d models, while allowing all previous definitions for symmetry to be covered by our method. Our framework supports different types of control curves for curve symmetries: Straight lines, circles, Bezier curves and NURBS curves. We show for all of them how their parameters can be found automatically from a symmetry group. For each part that is related to such a control curve, the system also identifies the relative orientation to the Frenet frame of the curve.

Our application allows the user to modify the model by moving and resizing parts using the mouse or a touchscreen. Changes made by the user are propagated through the connectivity graph of the object in a breadth-first manner, touching every part only once.

This work shows how parts and binary relations have to be updated according to changed neighbouring parts, also giving an overview on how this propagation could be extended to support rotations. Higher-order relations are updated whenever the pre-conditions for them are fulfilled. For objects in such relations, we implemented two different behaviours: They can stay equally spaced on the curve, or they can be repeated to keep the distance between two parts as fixed as possible. Both techniques can either only modify the position of the affected parts, or they can also adapt the orientation based on the relative orientation and the changed Frenet frame.

In order to test the usability and to find out which input method is preferred by the users, we performed a user study at the end of our work. Every user had to do each task twice and select the preferred input method. Beside this, we also measured the time each user needed for a task. By evaluating the results of the study we found out that our system is in general very well accepted by the users, although there is space for improvements, especially for the touchscreen controls. Another finding is that touch interaction is better accepted by untrained users than by experienced ones, which is, regarding our target group, very good. At the end of this work, results and performance statistics for our framework are shown, also describing the limitations the system currently has.

## 6.2 Conclusion

The main target of this work was to develop a structure-aware model-processing system that explicitly targets untrained users, allowing them to easily modify 3d geometry via a touchscreen. This target was generally met, although some minor problems were identified. Our user study documents that the system is usable by our target group since all users were able to perform the given tasks. It turned out, that the interaction method via touchscreen was not that well accepted by the users, although all users were able to perform the tasks also with this interaction method.

The novel formulation for symmetries presented in this work allows us to describe more complex symmetries than lines and regular grids. The implementation in our framework shows that our method of defining curve symmetries performs well in such a structure-aware model processing system. One of the biggest challenges during the development was to set up a stable propagation-based algorithm, since small changes in the propagation order can cause the model to react totally unintuitive, or to be broken after an interaction. Especially when repeat behaviour is used, this gets more important, since newly inserted parts will influence further propagation immediately.

Another point we found out is that finding stable binary relations in the input model is one of the most important tasks in such a framework. Using badly placed binary relations can easily break the whole system by moving parts in a way that is unpredictable by the user, thus making the application unusable. Our method for finding good binary relations is a first approach to this problem field, although a lot of work can be done here. We also found out that using a propagation-based algorithm performs very well with simple constraints, but when more complex constraints should be added to the

system, it might be better to use another approach like a global system of equations, since integrating them here produces a lot of problems especially in the ordering of the update propagation.

The use of structure-aware shape manipulation will, due to better methods developed, increase in the future. Especially when inexperienced users have to manipulate 3d geometry, which is the case when working with 3d printers etc., such a system might be the best way to produce good-looking results that satisfy the application constraints. We presented here one possible implementation of such a system, making improvements in the use of repeatable patterns on curves and in the binary relation detection, which gives us the ability to produce a large number of good-looking results in very short time.

## 6.3 Future Work

There are a lot of improvements that we will hopefully be able to add in the future: Regarding the touchscreen interaction, it might improve the interaction quality when the users could drag the objects directly while arrows are used at the same time to scale these objects, as implemented in the Microsoft 3D Builder[1]. The user interface could also be improved by displaying the arrows always at the same size, regardless of the distance to the part, by displaying point sprites instead of real geometry. Another option would be to use the touch gestures described by Au et al. [ATF12], which allow the user to manipulate parts by selecting the manipulation axis.

From the algorithmic point of view, the whole system could be reformulated to use a global system of equations instead of the propagation-based technique. This will allow us to integrate constraints like parallelity or coplanarity in our system more easily. The detection of patterns in the input geometry can also be improved to identify such patterns automatically and assign the correct control curve to it. Here we might be able to benefit from the usage of finite elements. An algorithm that automatically finds out which resizing behaviour should be assigned to a part would also be highly desirable. Another interesting topic would be to extend the theory of symmetries on curves to two dimensions, which would allow us to use two-dimensional patterns, for example on a sphere or on a Bezier patch.

The binary relation detection, especially the connected-component search, is also a topic where we want to make improvements. For example, binary relations could be derived from the connectivity point cloud by grouping binary relations that are not only specially near, but also by the constraints they express. This would reduce problems when two objects require more than one binary relations between them, e.g., allowing the propagation of size changes between two objects.

[1] `http://apps.microsoft.com/windows/de-at/app/3d-builder/75f3f766-13b3-45e9-a62f-29590d5781f2`

APPENDIX A

# User Study Material

On the following pages, the materials for the user study are shown, starting with the questionnaire for participants, followed by the task descriptions for the users. For more details, refer to Section 5.2.

# Questionary for a User

## Task1

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

## Task2

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

## Task3

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

## Task4

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

## Task5

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

## Task6

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

**Task7**

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

More natural model ☐ Task 6 (Bezier) ☐ Task 7 (NURBS)

**Task8**

Time required:_____/_____

Number of retries:_____/_____

Preferred Interaction: ☐ Mouse Input ☐ Touch Input

**Task10**

Time required:_____/_____

Number of retries:_____/_____

Prefered Interaction: ☐ Mouse Input ☐ Touch Input

# After Study-Questions

How much experience do you have with touch input devices (Tablets, PCs, Mobile-Phones)?

Very much ☐——☐——☐——☐——☐ None

How much experience do you have with modelling applications (Blender/Maya/AutoCAD)?

Very much ☐——☐——☐——☐——☐ None

How much do you liked the framework?

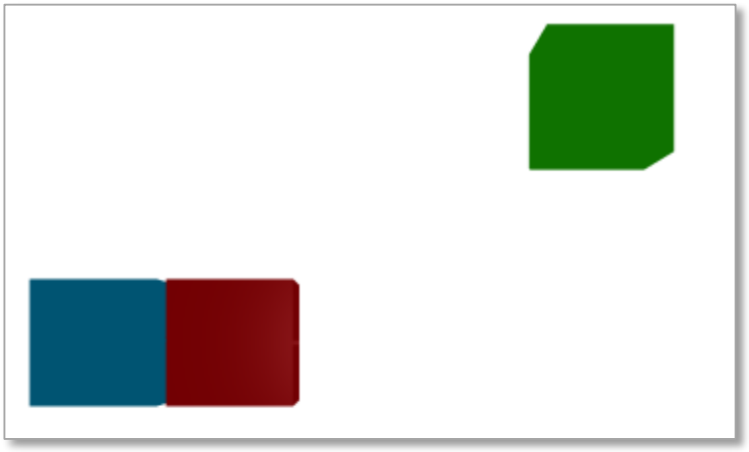
Liked it a lot ☐——☐——☐——☐——☐ Don’t like it

After all tasks: Would you prefer to use the mouse input or the touch input

Mouse ☐——☐——☐——☐——☐ Touch

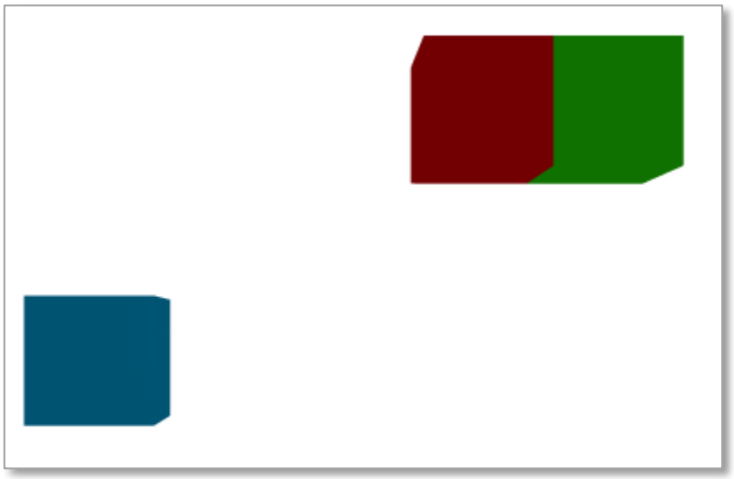
# Task 1

## Initial Situation



## Task

Move the red box to the green box so that the two boxes touch:

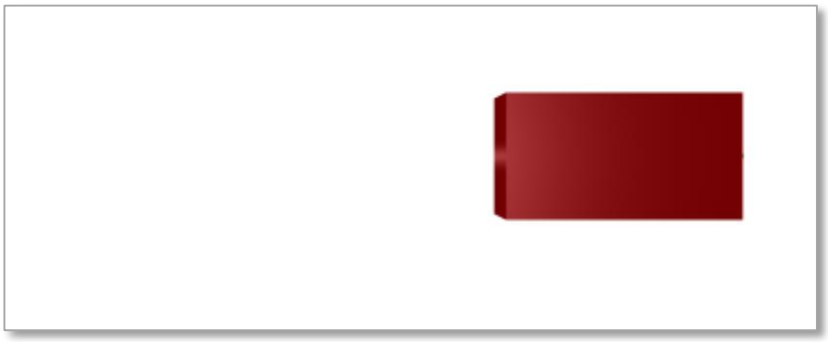


# Task 2

## Initial Situation

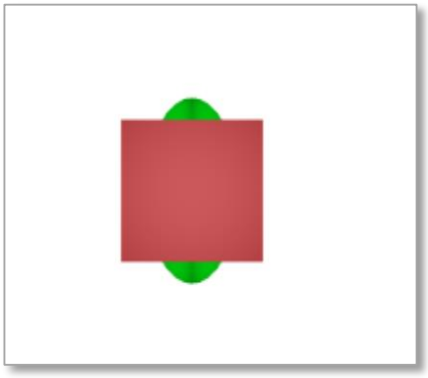
## Task

Move and scale the red box, so that it perfectly covers the green ellipsoid:



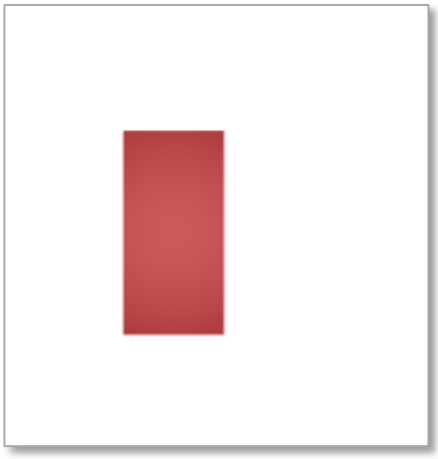
# Task 3

## Initial Situation

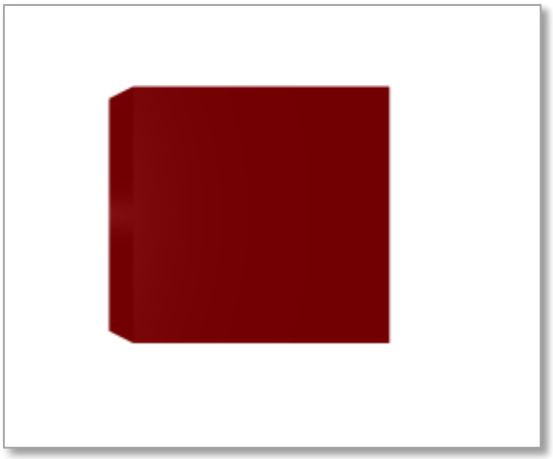


## Task

Move and scale the red box, so that it perfectly covers the green ellipsoid. In this task you will have to move the camera.



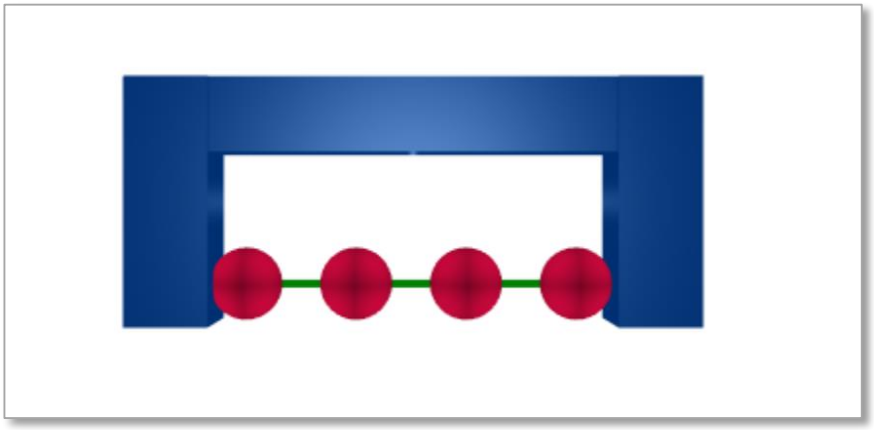
Front View



Side View

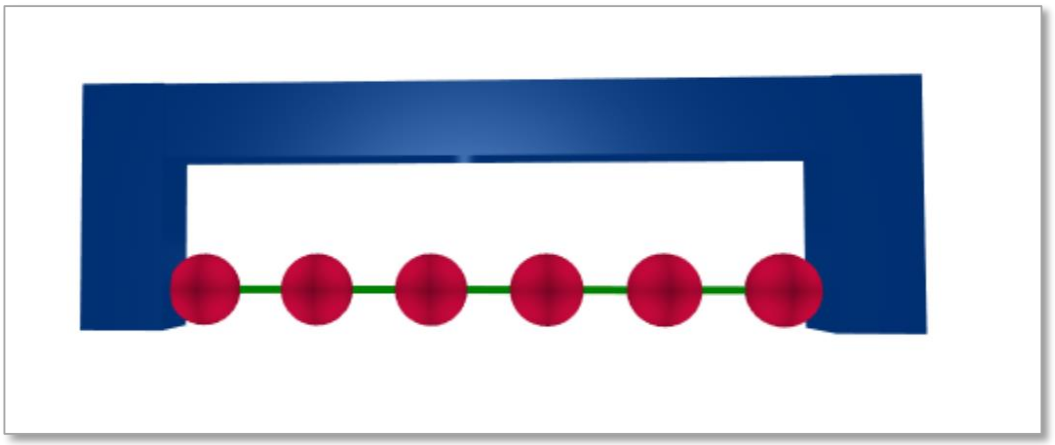
# Task 4

## Initial Situation



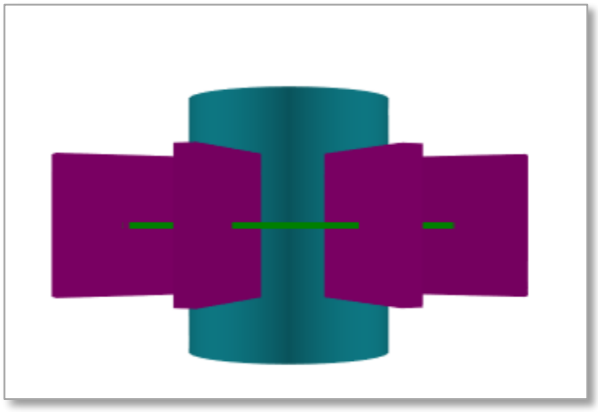
## Task

Enlarge the model until it has exactly six spheres:



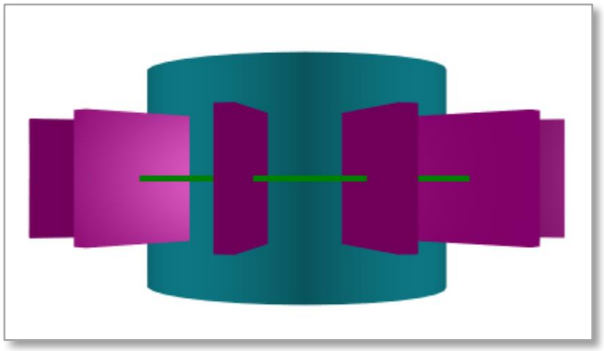
# Task 5

## Initial Situation



## Task

Enlarge the model until it has exactly 12 violet boxes

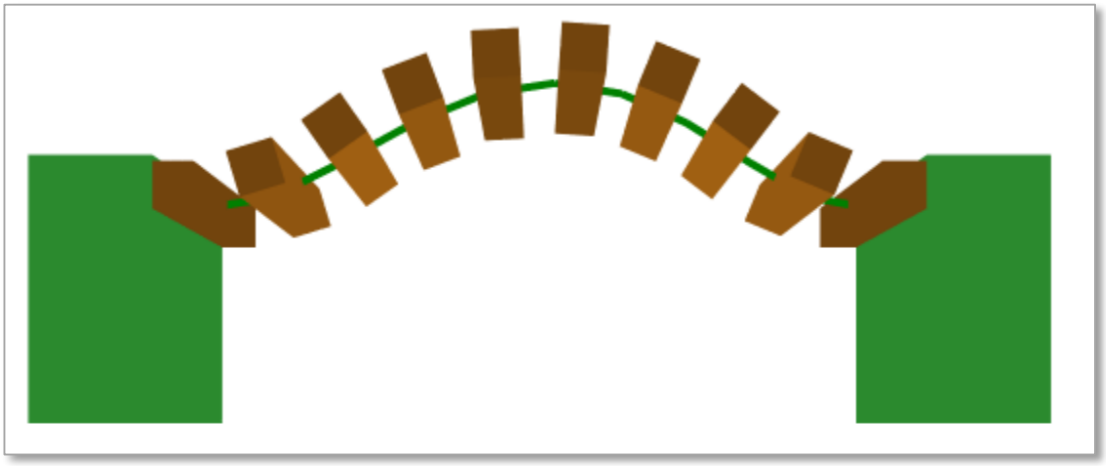


Front View

Top View

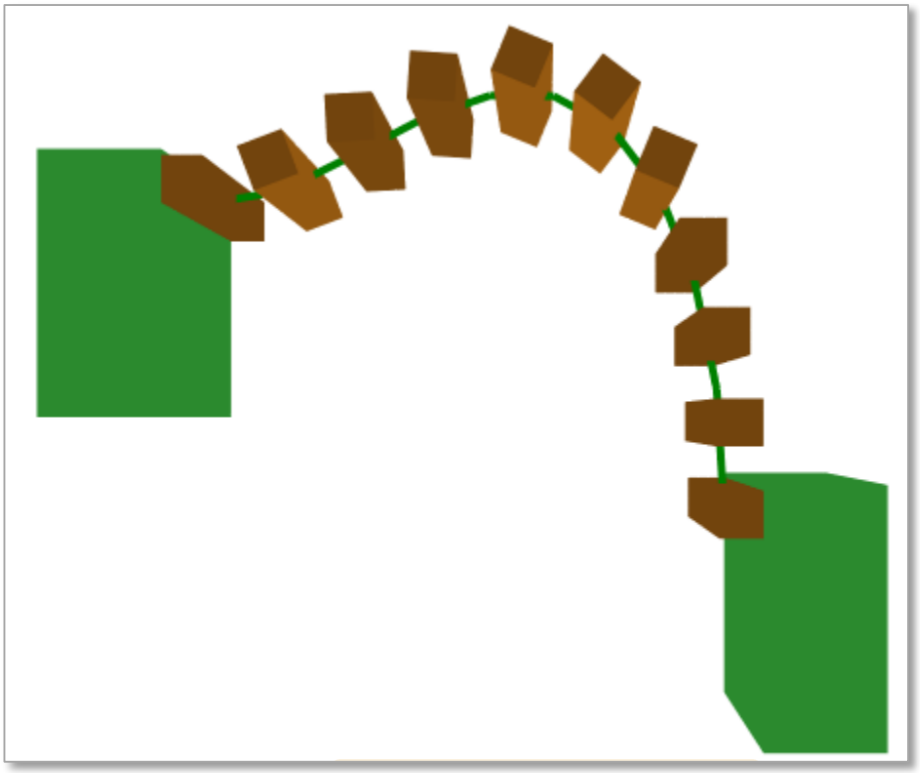
# Task 6

## Initial Situation



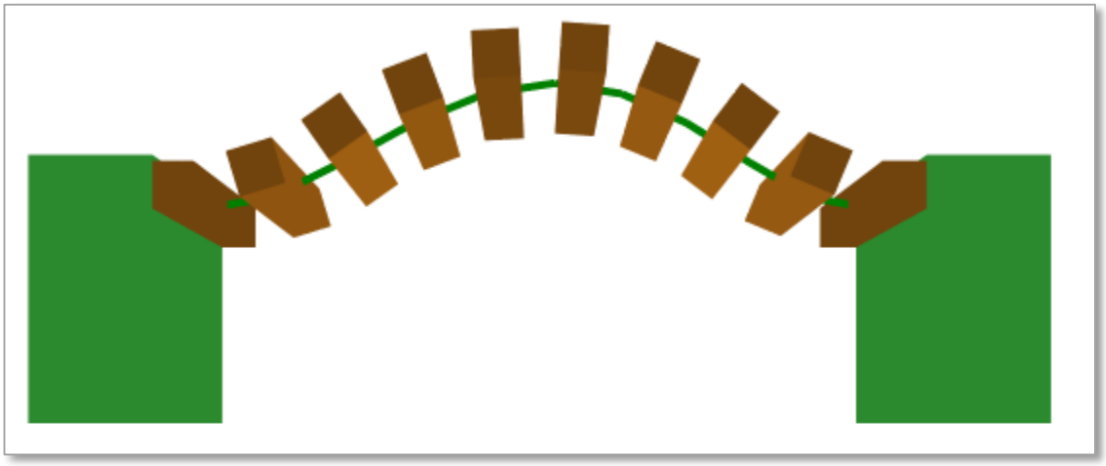
## Task

Modify the model until you have a feeling for it’s behavior, then try to achieve the following result



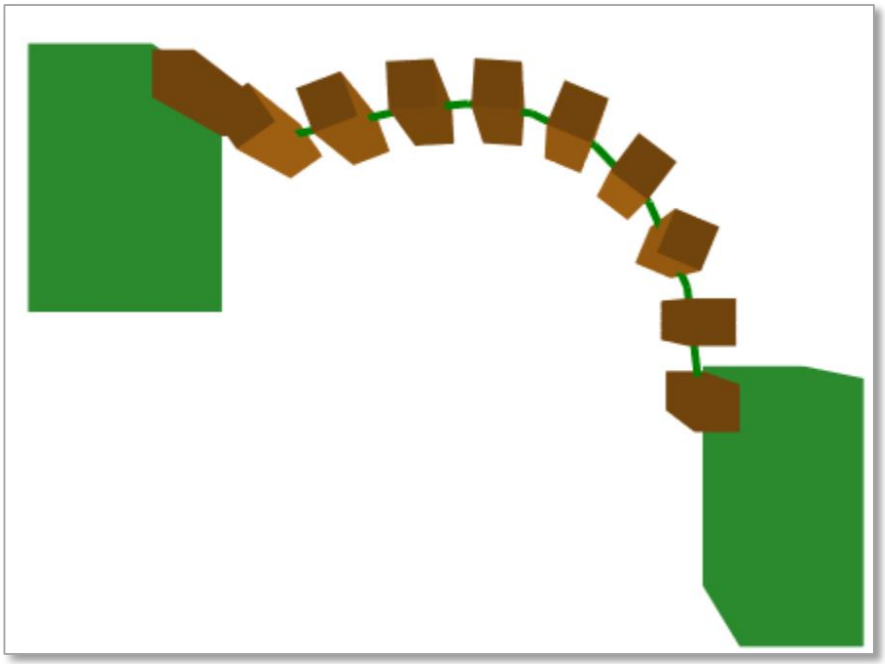
# Task 7

## Initial Situation



## Task

Modify the model until you have a feeling for it's behavior, then try to achieve the following result

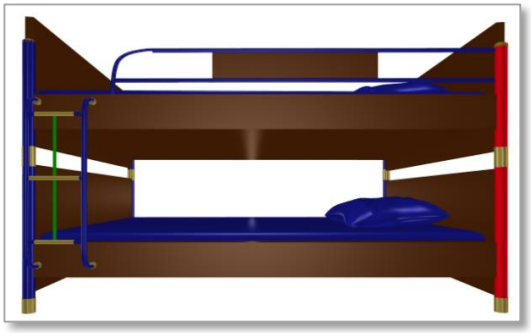


# Task 8

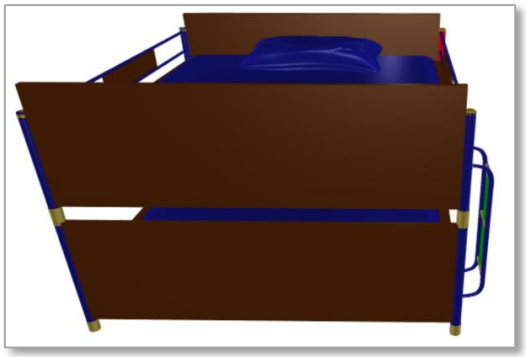
## Initial Situation

## Task

Modify the bed by moving and scaling only the red part to achieve the following result:



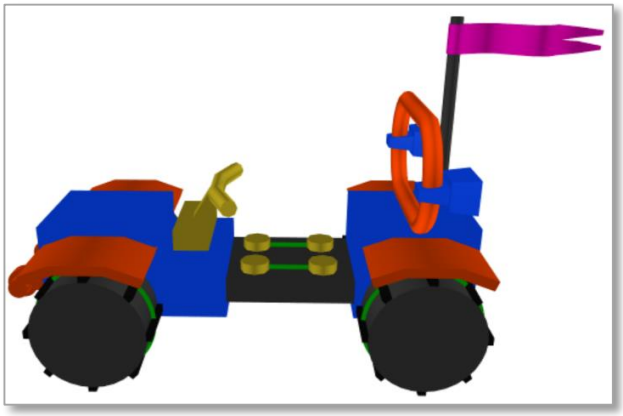
Front View



Side View

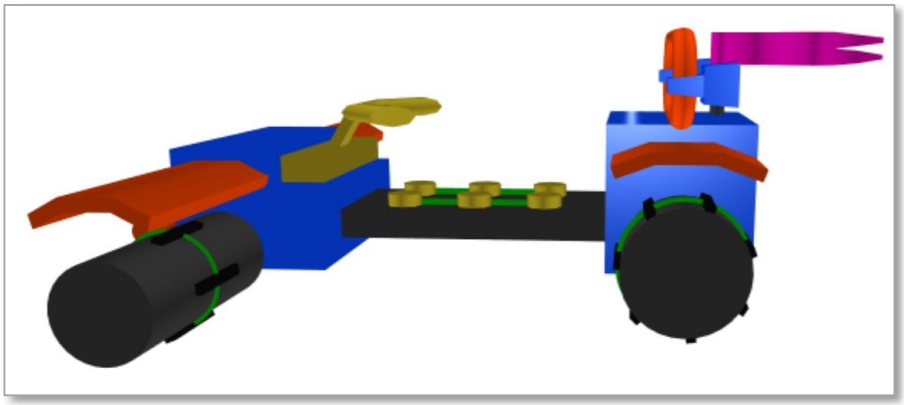
# Task 9

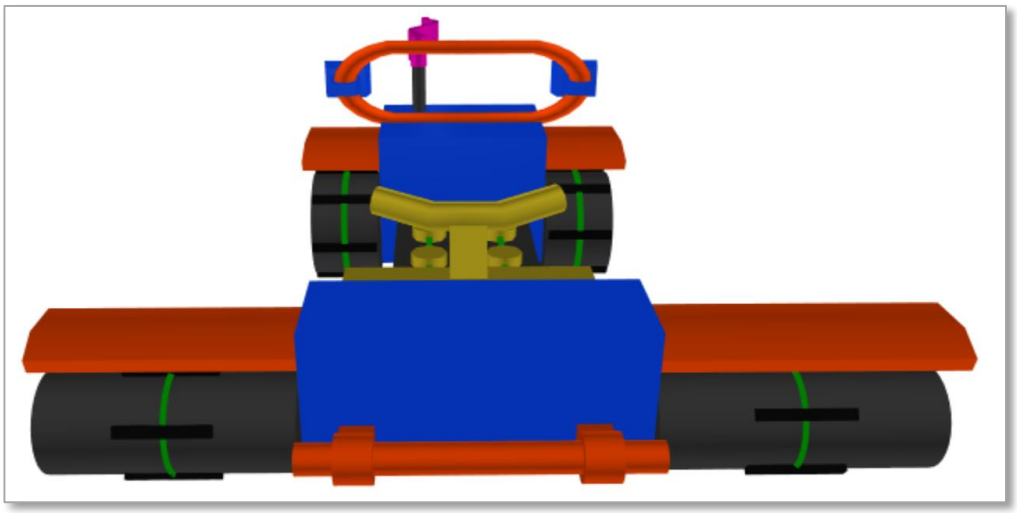
### Initial Situation



# Task

Modify the model to achieve the following result:





# Bibliography


[AKM+06] Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patané, Michela Spagnuolo, and Ayellet Tal. Mesh segmentation-a comparative study. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, page 7. IEEE, 2006. 46

[ATF12] Oscar Kin-Chung Au, Chiew-Lan Tai, and Hongbo Fu. Multitouch gestures for constrained transformation of 3d objects. In *Computer Graphics Forum*, volume 31, pages 651–660. Wiley Online Library, 2012. 91

[BBW+08] Alexander Berner, Martin Bokeloh, Michael Wand, Andreas Schilling, and H-P Seidel. A graph-based approach to symmetry detection. In *Proceedings of the Fifth Eurographics/IEEE VGTC conference on Point-Based Graphics*, pages 1–8. Eurographics Association, 2008. 32

[BM92] Paul J Besl and Neil D McKay. Method for registration of 3-D shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992. 30, 52

[BWKS11] Martin Bokeloh, Michael Wand, Vladlen Koltun, and Hans-Peter Seidel. Pattern-aware shape deformation using sliding dockers. *ACM Transactions on Graphics (TOG)*, 30(6):123, 2011. 1, 17, 18, 19, 20, 21, 31, 55, 64

[BWS10] Martin Bokeloh, Michael Wand, and Hans-Peter Seidel. A connection between partial symmetry and inverse procedural modeling. *ACM Transactions on Graphics*, 29(4):104–114, July 2010. 14, 15, 16, 55

[BWSK12] Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. An algebraic model for parameterized shape editing. *ACM Transactions on Graphics*, 31(4):1–10, 2012. 1, 2, 19, 20, 21, 26, 31, 55, 64

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3D mesh segmentation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 73. ACM, 2009. 46

[CLDD09] Marcio Cabral, Sylvain Lefebvre, Carsten Dachsbacher, and George Drettakis. Structure-Preserving Reshape for Textured Architectural Scenes. *Computer Graphics Forum*, 28(2):469–480, April 2009. 2, 64

[CSSK02] Dmitry Chetverikov, Dmitry Svirko, Dmitry Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 545–548. IEEE, 2002. 30, 52

[DGKP08] B. Drmota, G. Gittenberger, M. Karigl, and A. Panholzer. *Mathematik für Informatik (2nd Edition)*. Heldermann Verlag, Germany, 2008. 27

[Ebe03] David S Ebert. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003. 14

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. 50

[Fed91] E S Fedorov. The symmetry of regular systems of figures. *Zap. Mineralog. Obsc.(2)*, 28:1–146, 1891. 28

[GCO06] Ran Gal and Daniel Cohen-Or. Salient geometric features for partial shape matching and similarity. *ACM Transactions on Graphics (TOG)*, 25(1):130–150, 2006. 32

[GG04] Natasha Gelfand and Leonidas J Guibas. Shape segmentation using local slippage analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 214–223. ACM, 2004. 9, 32

[GLM96] Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM, 1996. 49

[GSMCO09] Ran Gal, Olga Sorkine, Niloy J Mitra, and Daniel Cohen-Or. iWIRES: An Analyze-and-Edit Approach to Shape Manipulation. *ACM Transactions on Graphics*, 28(3):1, 2009. 2, 9, 10, 11, 12, 26, 55, 65

[Hel79] Sigurdur Helgason. *Differential geometry, Lie groups, and symmetric spaces*, volume 80. Academic press, 1979. 32

[JBK+12] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Transactions on Graphics*, 31(4):1–10, July 2012. 8

[Jia13] Yan-Bin Jia. Parametric Curves, 2013. 37

[Jon68] Owen Jones. *The grammar of ornament*. B. Quaritch, 1868. 27, 28

[JTRS12] Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. Exploring Shape Variations by 3D-Model Decomposition and Part-based Recombination. *Computer Graphics Forum*, 31(2pt3):631–640, May 2012. 32, 49

[Kre13] Erwin Kreyszig. *Differential Geometry*. Courier Dover Publications, 2013. 35

[KSSCO08] Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. In *ACM Transactions on Graphics (TOG)*, volume 27, page 111. ACM, 2008. 9, 10, 18

[LK11] Thomas Larsson and Linus Källberg. Fast Computation of Tight-Fitting Oriented Bounding Boxes. In *Game Engine Gems 2*, pages 3–19. A K Peters, Ltd., 2011. 47

[LLCO08] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. In *ACM Transactions on Graphics (TOG)*, volume 27, page 78. ACM, 2008. 7, 8

[Llo82] Stuart Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982. 50

[LLZ10] Liang Lin, Xiaobai Liu, and Song-Chun Zhu. Layered graph matching with composite cluster sampling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(8):1426–1442, 2010. 45

[LM78] Edward Harrington Lockwood and Robert Hugh Macmillan. *Geometric symmetry*. CUP Archive, 1978. 32

[M̈97] Tomas Möller. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997. 50

[MGP06] Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Transactions on Graphics*, 25(3):560, 2006. 31

[MPWC13] Niloy J Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. In *Computer Graphics Forum*, volume 32, pages 1–23. Wiley Online Library, 2013. 26, 27, 28, 29, 30, 31, 32

[Mus14] Przemyslaw Musialski. Quantitative Usability Testing (in a Nutshell). Technical report, 2014. 38, 39, 40

[MWCS13] A. Milliez, M. Wand, M.-P. Cani, and H.-P. Seidel. Mutable elastic models for sculpting structured shapes. *Computer Graphics Forum*, 32(2pt1):21–30, May 2013. 15, 17

[MWH+06] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. *Procedural modeling of buildings*, volume 25. ACM, 2006. 14

[MWZ+13] Niloy J Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, and Martin Bokeloh. Structure-Aware Shape Processing. In *EUROGRAPHICS 2013 - State of the Art Reports*. Eurographics Association, 2013. 2, 4, 23, 24, 25, 26, 42, 45

[Nil93] Jakob Nilsen. Usability engineering. *San Francisco: Morgan Kaufmann*, 1:993, 1993. 38

[Pie91] Les Piegl. On NURBS: a survey. *IEEE Computer Graphics and Applications*, 11(1):55–71, 1991. 34

[PM01] Yoav I H Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001. 14

[PMW+08] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics*, 27(3):43–52, August 2008. 32

[PT95] Les Piegl and Wayne Tiller. *Curve and Surface Basics*. Springer, 1995. 33

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, 2007. 7, 8, 15

[SCOL+04] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184. ACM, 2004. 7, 8

[Sha08] Ariel Shamir. A survey on mesh segmentation techniques. In *Computer graphics forum*, volume 27, pages 1539–1556. Wiley Online Library, 2008. 24

[Sor09] Olga Sorkine. Least-squares rigid motion using svd. *Technical notes*, 120:3, 2009. 52

[SSL+14] Adriana Schulz, Ariel Shamir, David I W Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. Design and Fabrication by Example. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2014)*, 33(4), 2014. 21, 22, 87

[Tho92] D Wentworth Thompson. On growth and form: the complete revised edition. *Dover, New York. Vandiver, R. and Goriely, A.(2008). Tissue tension and axial growth of cylindrical structures in plants and elastic tissues. Europhys. Lett.(EPL)*, 84:58004, 1992. 23

[vBM+10] Ondrej Šťava, Bedrich Beneš, R Měch, Daniel G Aliaga, and Peter Krištof. Inverse Procedural Modeling by Automatic Generation of L-systems. In *Computer Graphics Forum*, volume 29, pages 665–674. Wiley Online Library, 2010. 14

[Wer94] Josie Wernecke. *The inventor mentor: programming object-oriented 3d graphics with open inventorTM, release 2.* Addison-Wesley, 1994. 35

[XLZ+10] Kai Xu, Honghua Li, Hao Zhang, Daniel Cohen-Or, Yueshan Xiong, and Zhi-Quan Cheng. Style-content separation by anisotropic part scales. In *ACM Transactions on Graphics (TOG)*, volume 29, page 184. ACM, 2010. 24

[XWY+09] Weiwei Xu, Jun Wang, KangKang Yin, Kun Zhou, Michiel van de Panne, Falai Chen, and Baining Guo. Joint-aware manipulation of deformable models. *ACM Transactions on Graphics*, 28(3):1, July 2009. 64

[XXS05] Chunsheng Xin, Bo Xie, and Chien-Chung Shen. A novel layered graph model for topology formation and routing in dynamic spectrum access networks. In *New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on*, pages 308–317. IEEE, 2005. 45

[ZFCO+11] Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. Component-wise Controllers for Structure-Preserving Shape Manipulation. *Computer Graphics Forum*, 30(2):563–572, April 2011. 4, 5, 11, 13, 14, 24, 55, 63, 65

[ZPA95] Hagit Zabrodsky, Shmuel Peleg, and David Avnir. Symmetry as a continuous feature. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(12):1154–1166, 1995. 30