

TECHNISCHE UNIVERSITÄT MÜNCHEN  
Lehrstuhl für Produktentwicklung

# **Visual, Interactive 3D Spatial Grammars in CAD for Computational Design Synthesis**

**Frank Rainer Hoisl**

Vollständiger Abdruck der von der Fakultät für Maschinenwesen der Technischen Universität  
München zur Erlangung des akademischen Grades eines

**Doktor-Ingenieurs**

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing., Dr.-Ing. habil. Heinz Ulbrich  
Prüfer der Dissertation: 1. Univ.-Prof. Kristina Shea, Ph. D.  
2. Prof. Alison McKay, University of Leeds / UK

Die Dissertation wurde am 22.02.2012 bei der Technischen Universität München  
eingereicht und durch die Fakultät für Maschinenwesen  
am 09.07.2012 angenommen.

© Frank Hoisl, München 2012

Die Informationen in diesem Werk wurden mit großer Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Der Autor übernimmt keinerlei juristische Verantwortung oder Haftung jeglicher Art für eventuell verbliebene fehlerhafte Angaben und deren Folgen.

Alle Rechte vorbehalten.

**ISBN 978-3-00-040283-8**

## ABSTRACT

While Computer-Aided Design (CAD) has made significant progress since its inception, CAD tools are still used primarily for the creation, modification and documentation of designs rather than as ‘active’ partners in the design process. Spatial grammars are an approach for design synthesis having the potential to computationally support the human designer. They provide a method for rule-based, generative shape design but have yet to find general application within CAD systems. While many spatial grammars exist on paper, only a few have been computationally implemented to date. This is especially the case for three-dimensional systems. Most spatial grammars are hard-coded, i.e., once implemented, the vocabulary and rules cannot be changed without re-programming.

This work presents a new approach for creating a general three-dimensional spatial grammar system based on mechanical engineering CAD that enables visual, interactive definition and application of grammar rules for computational design synthesis. The method is based on a set grammar that uses a set of parameterized primitives and supports the definition of non-parametric and parametric rules, as well as their automatic application. It comprises a method for the automatic matching of the left hand side of a rule in a current working shape, including defining parametric relations.

Enhancements to the set grammar approach enable the use of Boolean operations, sweeping objects and three-dimensional labels to significantly increase the expressiveness of the geometry that can be generated with regard to mechanical engineering solutions. In addition to parametric relations and labels that are used as instruments to embed constraints in the rules, a collision detection mechanism is introduced to prevent generation of overlapping parts and to constrain the possible design space. A prototype implementation based on an open-source 3D modeling kernel and CAD system is presented and used to illustrate the approach through several mechanical engineering design examples such as the generation of vehicle wheel rims, robot arm concepts or gear systems.

The main contribution that results from this approach is that instead of being exclusively created for a specific example or domain, the software prototype provides a flexible platform supporting designers with visual, interactive definition and application of their own spatial grammar rules in a familiar CAD environment without programming. It puts the creation and use of three-dimensional spatial grammars on a more general level and enables their usability in mechanical engineering CAD systems to provide for more ‘active’ support of the engineering designer.

## ZUSAMMENFASSUNG

Trotz der erheblichen Fortschritte die Computer-Aided Design (CAD) seit seinen Anfängen gemacht hat, werden CAD Werkzeuge noch immer fast ausschließlich zur Erzeugung, Modifizierung und Dokumentation von Konstruktionen verwendet und nicht als ‚aktive‘ Partner im Konstruktionsprozess. ‚Spatial Grammars‘ stellen einen Ansatz zur Synthese von Geometrie dar, der das Potential besitzt den Konstrukteur computerbasiert zu unterstützen. Sie bieten eine Methode zur regelbasierten Generierung von Geometrie, welche bis dato keine Verbreitung im Bereich von CAD Systemen gefunden hat. Dies gilt insbesondere für dreidimensionale Systeme. Die überwiegende Anzahl an ‚Spatial Grammas‘ sind hard-coded, d.h. einmal implementiert können die Vokabeln und Regeln nicht ohne Umprogrammieren des Systems geändert werden.

Die vorliegende Arbeit präsentiert einen neuen Ansatz zur Umsetzung eines allgemeinen, dreidimensionalen ‚Spatial Grammar‘ Systems, das, basierend auf maschinenbaulichem CAD, die visuelle, interaktive Definition und Anwendung von Grammatikregeln zur rechnerbasierten Synthese von Geometrie ermöglicht. Die Basis der vorgestellten Methode bildet eine ‚Set Grammar‘. Unter Verwendung eines Satzes parametrisierter Primitive unterstützt diese Methode die Definition nicht-parametrischer und parametrischer Regeln sowie deren Anwendung. Sie umfasst einen Ansatz zur automatischen Erkennung der linken Seite einer Regel in einer existierenden Geometrie und berücksichtigt dabei auch vorhandene parametrische Beziehungen.

Durch Erweiterungen des ‚Set Grammar‘ Ansatzes wird die Verwendung von durch Boolesche und Sweeping Operationen erzeugten Volumenkörpern sowie der Einsatz von dreidimensionalen ‚Labels‘ ermöglicht. Mit Blick auf die Generierung maschinenbaulicher Lösungen wird dadurch die Aussagekraft der erzeugbaren Geometrie signifikant erhöht. Zusätzlich zu parametrischen Beziehungen und ‚Labels‘ als Instrumente zur Festlegung von Randbedingungen wird ein Mechanismus zur Kollisionserkennung eingeführt, durch den die Generierung überlappender Bauteile vermieden sowie der zur Verfügung stehende Bauraum eingeschränkt werden kann. Unter Verwendung eines Softwareprototyps, der auf Grundlage eines Open Source 3D Modellierungskernels und CAD Systems implementiert wurde, wird der beschriebene Ansatz anhand einiger maschinenbaulicher Konstruktionsbeispiele wie z.B. der Generierung von Autofelgen, Roboterarmen oder Getriebesystemen veranschaulicht.

Der Hauptbeitrag des vorgestellten Ansatzes, ist, dass der Softwareprototyp, anstatt als eigenständiges System für einen einzigen, spezifischen Anwendungsfall programmiert zu sein, eine flexible Plattform bietet, die Konstrukteure bei der visuellen, interaktiven Definition und Anwendung ihrer eigenen ‚Spatial Grammars‘ in gewohnter CAD Umgebung unterstützt ohne dabei Programmierkenntnisse zu erfordern. Der Ansatz ermöglicht eine generellere Erzeugung und Verwendung von dreidimensionalen ‚Spatial Grammars‘ sowie deren Nutzung in maschinenbaulichen CAD Systemen, um dem Konstrukteur eine ‚aktivere‘ Unterstützung bieten zu können.

## ACKNOWLEDGEMENT

This work results from my tenure as a doctoral student and scientific research assistant in the Virtual Product Development Group at the Institute of Product Development at the Technische Universität München from August 2006 until August 2011.

First of all I would like to thank my doctoral advisor Prof. Kristina Shea for believing in my abilities and for supporting me to work on this topic, which was the original motivation for me to do my Ph.D. Our many fruitful, challenging discussions and the detailed instructions on how to work to high research standards, especially with regard to scientific writing, helped me a lot in writing this doctoral thesis.

I would like to acknowledge Prof. Alison McKay for taking the role of my second supervisor and Prof. Heinz Ulbrich who spontaneously accepted to chair the examination board as Prof. Tim C. Lüth unfortunately fell ill the day of my defense.

I would also like to thank Prof. Udo Lindemann und Dr. Markus Mörtl for the very productive and friendly collaboration and for making me feel part of the overall Institute of Product Development.

Special thanks to all of my former colleagues. The many good discussions, great collaboration in projects and teaching, as well as friendship and laughs always made life at the chair enjoyable, even during the harder times of (doctoral student) life. In particular I would like to thank (alphabetically) Frank Deubzer, Bergen Helms, Andreas Kohn and Stefan Langer for their great support and the many great memories, such as forming the ‘greatest office’ at the chair, sharing rooms at conferences, going out together, helping to organize the formal administrative work around the submission of my doctoral thesis while I already lived abroad, listening to my complaints and problems, being able to see and enjoy the irony in life and work, sharing a great sense of humor, and many more.

Further, I would like to thank my student assistants, for example, Dong Li, Robert Schröder and René Neumann, to name a few. They helped me evolve my research and contributed to or improved many parts of the software prototype implemented for my research approach.

Special thanks go to my parents Ursula and Gerhard and my sister Katja who have always tried to make my life as easy as possible and thus enabled me to gain this high level of education.

Last but not least, I would like to thank my beloved girl-friend Sarah who, moving to Germany, decided to live in a weekend relationship for two years for the sake of me being able to do my Ph.D. Besides that, she often had to deal with a ‘grumpy man’ who would sometimes spend more time with his computer implementing a software prototype than with her. Finally, I would like to thank her for doing the tedious proof reading work for my thesis.



The following publications are part of the work presented in this thesis:

Hoisl, F.; Shea, K.: An Interactive, Visual Approach to Developing and Applying Parametric 3D Spatial Grammars. *AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing)* 25 (2011) 04, p. 333-356.

Hoisl, F.; Shea, K.: Interactive, Visual 3D Spatial Grammars, Fourth International Conference on Design Computing and Cognition DCC'10. Stuttgart, Germany, 2010.

Hoisl, F.; Shea, K.: Exploring the Integration of Spatial Grammars and Open-Source CAD Systems, 17th International Conference on Engineering Design (ICED'09). Stanford University, California, USA, 2009.

Hoisl, F.: Integrating Computer Aided Design and Shape Grammars for Computational Design Synthesis. In: *forum 3D 2009 3D-Technologien an der Technischen Universität München*, München, Shaker Verlag, 2009

Hoisl, F.; Shea, K.; Helms, B.: Towards Representing, Evolving and Networking Engineering Knowledge for Computational Design Synthesis, International Design Conference DESIGN 2008. Dubrovnik, Croatia, 2008.





# TABLE OF CONTENTS

|  |           |
|--|-----------|
| <b>1. Introduction</b>   | <b>1</b>  |
| 1.1 Current situation and motivation                             | 1         |
| 1.2 Research goals   | 2         |
| 1.3 Thesis outline   | 3         |
| <b>2. Background</b>   | <b>5</b>  |
| 2.1 Computational design synthesis                               | 5         |
| 2.2 Spatial grammars: theoretical fundamentals                   | 5         |
| 2.2.1 Formalism  | 6         |
| 2.2.2 Labels   | 8         |
| 2.2.3 Parametric grammars  | 11        |
| 2.3 Grammar interpreters and user interaction                    | 11        |
| 2.4 Existing spatial grammars                                    | 12        |
| 2.5 Relation to Knowledge-Based Engineering in CAD               | 20        |
| <b>3. Benefits and challenges</b>                                | <b>25</b> |
| <b>4. Approach</b>   | <b>29</b> |
| 4.1 Basics and non-parametric rules                              | 29        |
| 4.1.1 Development of rules and transformations of objects        | 30        |
| 4.1.2 Application of rules and LHS matching                      | 31        |
| 4.2 Parametric rules   | 36        |
| 4.2.1 Development of parametric rules                            | 36        |
| 4.2.2 Application of parametric rules                            | 39        |
| 4.3 Objects based on Boolean operations and swept objects        | 44        |
| 4.3.1 Boolean operations   | 44        |
| 4.3.2 Sweeping   | 50        |
| 4.4 Three-dimensional labels                                     | 53        |
| 4.4.1 General uses for three-dimensional labels                  | 54        |
| 4.4.2 Using three-dimensional labels for simplified LHS matching | 56        |

---

|           |   |           |
|-----------|---|-----------|
| 4.5       | Collision detection                       | 59        |
| 4.5.1     | Part collision avoidance                  | 60        |
| 4.5.2     | Design space restriction                  | 61        |
| 4.6       | Summary                                   | 62        |
| <b>5.</b> | <b>Software prototype</b>                 | <b>63</b> |
| 5.1       | System basics                             | 63        |
| 5.2       | Development of grammar rules              | 64        |
| 5.3       | Application of grammar rules              | 69        |
| 5.4       | Summary                                   | 71        |
| <b>6.</b> | <b>Examples</b>                           | <b>73</b> |
| 6.1       | Vehicle wheel rims                        | 73        |
| 6.2       | Cylinder cooling fins                     | 75        |
| 6.3       | High voltage insulators                   | 78        |
| 6.4       | Robot arm concepts                        | 81        |
| 6.5       | Gear systems                              | 84        |
| 6.6       | Summary                                   | 90        |
| <b>7.</b> | <b>Discussion and future work</b>         | <b>91</b> |
| 7.1       | Research contributions                    | 91        |
| 7.2       | Limitations and potential for improvement | 92        |
| 7.3       | Future extensions                         | 95        |
| <b>8.</b> | <b>Conclusion</b>                         | <b>97</b> |
| <b>9.</b> | <b>References</b>                         | <b>99</b> |

# 1. Introduction

## 1.1 Current situation and motivation

Remaining competitive in global markets is the crucial challenge for companies today. Above all, companies are forced to develop better products faster. In the engineering domain the computer has become a key tool, as it provides support and can automate a great variety of tasks involved in the creation of products. Its use significantly improves the ability of companies to meet the time, cost and quality requirements of competition. With continuously increasing computational resources, it is possible to perform routine tasks quicker and handle growing amounts of data more efficiently.

Today, a vast range of software tools are available to engineers covering areas such as design, analysis, simulation, optimization, manufacturing or visualization, to name a few. Engineering design, which is one of the central tasks of product development, is greatly supported by the use of Computer-Aided Design (CAD) systems. Since the development of the first concepts in the early 1960s (SUTHERLAND 1963), CAD has made significant progress. Over the decades, CAD systems have evolved from two-dimensional computational drawing applications to three-dimensional parametric, feature-based modeling systems – the standard systems used by mechanical engineers today. Based on a wide range of functionality, they provide high generality with regard to the geometric models that can be designed. Over many years of development, the interfaces of CAD tools have also evolved and become quite sophisticated (CHASE 2002), resulting in systems that are easy and intuitive to use via a graphical user interface (GUI).

The input and decisions about how to perform modeling operations are provided by the human designer. CAD systems on the market are used primarily for the creation, modification and documentation of designs and, therefore, can be considered passive (HEISSERMAN et al. 2004, CHASE 2002). Especially automatic support for the actual act of synthesis, i.e. the creation of non-predefined, possibly new, design solutions, is hardly provided by CAD tools. Consequently, the powerful computational systems that exist today are used below their capacities in this area (CELANI 2002).

Grammar-based design synthesis systems are considered to be potentially powerful tools for the generation of designs with little or no user-input (CHASE 2002). Especially spatial grammars provide a method for rule-based, generative shape design but have yet to find general application within CAD. To extend on current CAD tools, spatial grammars have the potential to act as an active partner through the synthesis of designs, thereby further supporting the human designer. They can support routine design tasks, assist in the generation of alternative designs and, even more interesting, create spatially novel solutions beyond what a designer might think of.

More than 40 years ago, STINY & GIPS (1972) introduced shape grammars as a generative approach to shape design. In following research, STINY (1980a) further detailed this concept and subsequently distinguished set grammars as a simple variant of shape grammars (STINY

1982). Since then, a significant amount of research has been published on shape and set grammars, both of which can be classified under the more general term spatial grammars (KRISHNAMURTI & STOUFFS 1993). Originally presented for paintings and sculptures, this concept has also been successfully applied in other domains, such as architecture, industrial design, decorative arts and engineering (cf. overviews in CHAU et al. 2004 and CAGAN 2001).

While many spatial grammars exist on paper, only a few, limited spatial grammar systems have been computationally implemented to date. This is especially true for three-dimensional systems. Of those that have been implemented, many are restricted to one specific design domain or application example. Further, the majority of implemented examples do not provide for a visual way to edit an existing grammar or to develop a completely new grammar. Instead, they usually require coding of grammar rules in textual form (CHASE 2002), meaning that, once implemented, the vocabulary and rules cannot be changed without re-programming. This makes at least some programming knowledge necessary. Practicing designers, however, tend to think spatially. They are used to working in a graphical environment, for example, using the GUI of a CAD system, or are often not willing or able to program due to limited programming experience. Instead, they want to focus on designing.

## 1.2 Research goals

The aim of the research presented in this thesis is the development of a new approach for creating a general three-dimensional spatial grammar system. Instead of being exclusively created for a specific example or domain, the more general system will provide a flexible platform supporting designers with visual, interactive definition and application of their own grammar rules in a familiar CAD environment without programming. Aiming to overcome the limitations of currently available three-dimensional spatial grammar systems, the approach taken here seeks to make the advantages of spatial grammars available in CAD systems. Enhancing CAD systems with computational design synthesis capabilities transforms such systems into a more active partner for the human designer.

The thesis is written from a mechanical engineering perspective, but the approach described is applicable to other domains, such as architecture, as well. The aim is to provide design support through the integration of a spatial grammar interpreter in a CAD system. Therefore, the developed approach is implemented as a prototype software system based on a three-dimensional solid CAD system. This reflects a statement by GIPS (1999) who describes the idea of developing a shape grammar plug-in for a traditional CAD program that would assist in creating a shape grammar, which, in turn, would help the practicing designer.

The term ‘visual’ is used to distinguish from grammar approaches that require programming for the development of new rules. The direct visual manipulation of displayed geometric objects using the computer mouse is not within the focus, as the extent to which this is possible is highly dependent on the CAD system used and only marginally affects the actual approach. As is common in mechanical engineering design, the need to define parametric relations and exact measurements is realized using numerical inputs that trigger an immediate update of the geometry and its visualization.

## 1.3 Thesis outline

The thesis starts with a background chapter that introduces computational design synthesis and the set and shape grammar formalism. It explains the meaning of interpreters and user interaction, presents and characterizes existing spatial grammar implementations and takes a look at related approaches in the area of Knowledge-Based Engineering (KBE).

Building on the background chapter, Chapter 3 discusses the potential benefits of using a spatial grammar approach in CAD and the challenges that are involved in creating a general, three-dimensional grammar interpreter.

Taking the various challenges highlighted in the previous chapter into consideration, Chapter 4 outlines this research's approach, starting with a clarification of the concepts used in the development of non-parametric and parametric grammar rules and their automatic application. Enhancements to the approach in terms of using Boolean operations, simple swept objects and three-dimensional labels to significantly increase the expressiveness of the geometry that can be generated are described subsequently. Finally, a collision detection mechanism adding further possibilities to constrain design generation is introduced.

The implementation of the approach in a prototype software system is presented in Chapter 5. The chapter demonstrates the functionality and the use of the system as well as some important issues concerning the internal realization of the implementation. Several mechanical engineering examples that cover the different aspects of the presented approach are given in Chapter 6.

The thesis ends with a summary of the research contributions, a discussion of the limitations, potential areas of improvement, suggestions for future research and a conclusion.



## 2. Background

This chapter discusses relevant research topics and existing work related to this thesis. It provides the basis for the derivation of benefits and challenges presented in Chapter 3 and enables the reader to gain an understanding of the context of this thesis.

The chapter starts with an introduction to the general area of computational design synthesis, followed by an explanation of the formalism that underlies the approach of spatial grammars including the related terminology. The meaning of grammar interpreters as well as aspects of user interaction are described, before an overview of important existing spatial grammars is given. Here, the focus is on implemented grammar systems; these are characterized in more detail. The final part of this chapter compares the spatial grammar approach to KBE approaches used in commercial CAD software.

### 2.1 Computational design synthesis

The approach presented in this thesis aims at computationally supporting the human CAD designer in the design synthesis phase. Design synthesis can be thought of as creating form, or product structure, to fulfill desired behavior and function (STARLING & SHEA 2005) based on the creation and combination of building blocks. It is the creative step itself, the conception and postulation of possibly new solutions to solve a problem, which in engineering design is mostly performed by creative human minds (ANTONSSON & CAGAN 2001). Formalization of the design synthesis process enables the computer-based, automated or semi-automated generation of designs making use of the capacities that computers provide.

Several methods for formal, computational design synthesis have been successfully developed during the last decades. Three major themes among these methods are function-based synthesis, grammar-based synthesis and analogy-based design (CHAKRABARTI et al. 2011). Overviews of existing examples can be found in CHAKRABARTI et al. (2011), ANTONSSON & CAGAN (2001) and CHAKRABARTI (2002). They cover a wide range of applications, for example, structures (e.g. SHEA & CAGAN 1999), consumer products (e.g. AGARWAL & CAGAN 1998), mechanical systems (e.g. STARLING & SHEA 2005), automotive styling (e.g. MCCORMACK et al. 2004), and microelectromechanical systems (e.g. BOLOGNINI et al. 2006).

The approach in this thesis relies on grammar-based synthesis, namely spatial grammars, as they have the potential to be directly used in a CAD environment that is familiar to designers.

### 2.2 Spatial grammars: theoretical fundamentals

‘Spatial grammars’ is a general term that includes all kinds of grammars that define languages of shape, for example, string grammars, set grammars, graph grammars and shape grammars (KRISHNAMURTI & STOUFFS 1993). This thesis focuses on set and shape grammars because the presented approach is based on a set grammar formulation and among the related existing implementations, there are also several shape grammar systems.

## 2.2.1 Formalism

The formalisms of set and shape grammars are similar. Both are generative systems that generate shapes applying defined rules iteratively starting from an initial set or shape that exists within a defined vocabulary of shapes. A set grammar is formally defined as  $G = (S, L, R, I)$  where:

- $S$  is a finite set of shapes
- $L$  is a finite set of labels
- $R$  is a finite set of rules, and
- $I$  is the initial set, where  $I$  is a subset of  $(S, L)^0$

The set of labeled shapes, including the empty labeled shape, is  $(S, L)^0$  and is also called the vocabulary. The rewriting rules are defined in the form  $A \rightarrow B$ , where  $A$  and  $B$  are both subsets of the vocabulary. An example for a rule is given in Figure 2-1(a). It is demonstrated on a simple, two-dimensional level to make it easily understandable to the reader and it does not make use of labels which are explained in more detail in the next subparagraph.

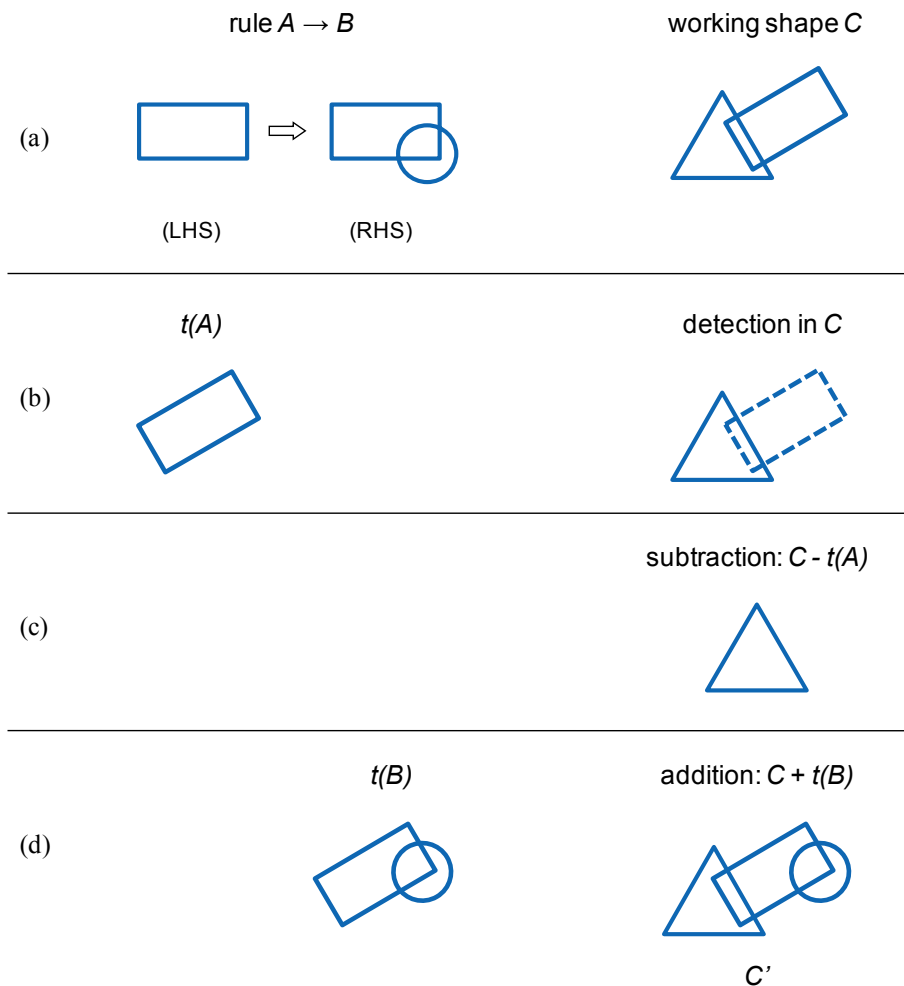


Figure 2-1: Example for the application of a rule according to the grammar formalism



To apply a rule to a given set of shapes (Figure 2-1(a), right), which is called the working shape  $C$  or current working shape (CWS), first,  $A$  in the left hand side (LHS) of a rule has to be detected in  $C$ . This matching process can make use of valid Euclidean transformations,  $t$ , that are applied to  $A$ , to find more possible matches of  $A$  in the working shape  $C$  (Figure 2-1(b)). The transformed subset  $A$  is then subtracted from  $C$  (Figure 2-1(c)) and the transformed subset  $B$  of the right hand side (RHS) of the rule is added (Figure 2-1(d)), thus resulting in a new set of shapes  $C'$  where  $C' = C - t(A) + t(B)$ .

Defining more than one rule, for example the two shown in the upper part of Figure 2-2, and applying the rules using different application sequences enables the generation of alternative solutions. Two example solutions and the according rule application sequences are shown in the lower part of Figure 2-2, both starting with the current working shape as shown in Figure 2-1(a), right.

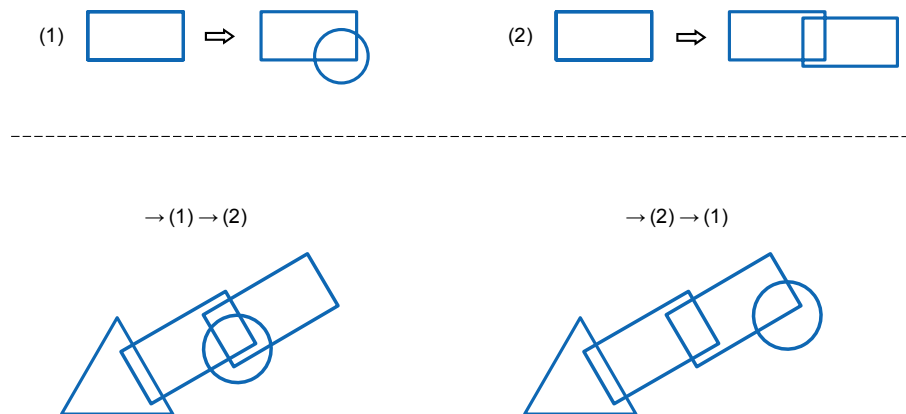


Figure 2-2: Two rules and two solutions created using different rule application sequences

The formalism of shape grammars is basically the same as the set grammar formalism, where  $I$  is the initial shape,  $A$  and  $B$  are shapes in the vocabulary and rules apply to subshapes of labeled shapes to produce other labeled shapes. In comparison, set grammar rules apply to subsets of sets of shapes to produce other such sets (STINY 1982). Therefore, designs generated by a set grammar consist of shapes in  $S$ . Designs defined by shape grammars, instead, consist of shapes and subshapes of shapes in  $S$ , because the compositional units in designs can be decomposed and recombined in different ways (STINY 1982). Shape grammars work directly on spatial forms (KRISHNAMURTI & STOUFFS 1993), while set grammars treat designs as symbolic objects with geometric properties, which makes them more amenable to computer implementation (STINY 1982). Strictly speaking, a shape grammar involves the use of a maximal line representation, which can be broken down and re-represented in a large number of ways. For example, a line can be broken up into smaller line segments to form subshapes. This ability for re-representing shapes in a number of ways enables for wider matching of the shape  $A$  in the LHS of a rule to embedded subshapes in the working shape  $C$ .

## 2.2.2 Labels

The second component of the grammar formalism presented above is the set of labels. Labels have been applied in a wide range of scopes (see overview in Figure 2-3) and are therefore explained in more detail in this subparagraph. The most common way to represent a label is a dot, or small circle, that might be filled or empty. In addition characters, or letters, and numbers have been used, often in combination with a geometrical symbol.

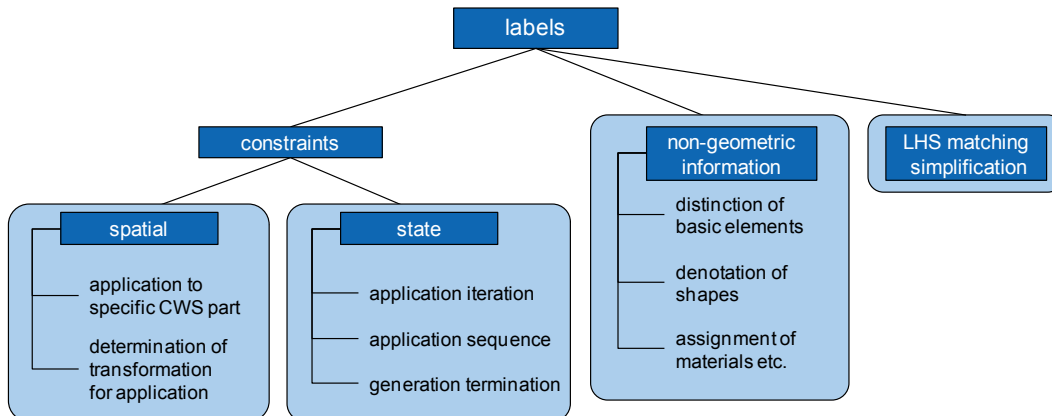


Figure 2-3: Concise overview of the use of labels

Spatial labels are used to control *where*, i.e. the relative position in relation to a distinct shape, and *how*, i.e. under what specific Euclidean transformations, rules apply to a CWS (KNIGHT 1994). In this context, spatial labels can also remove symmetries (rotational, reflective) of a shape. Often spatial labels are used to prevent application of rules to the same shape more than once or to make rules applicable only to the most recently added shape in a CWS. Spatial labels have a specific location but are invariant if a rule is applied under rotation or scale transformations.

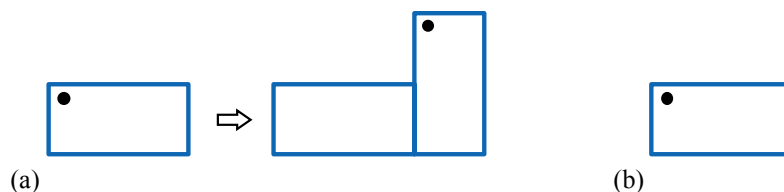


Figure 2-4: Rule using spatial labels

Figure 2-4(a) shows an example for a rule using spatial labels that combine the described purposes. Applying the rule to the initial set in Figure 2-4(b), the label depicts the one transformation under which the rule is to be applied. Without using a label, there would be several different rotation or reflection transformations to match the rectangle. Applying the rule a second time, the label forces the LHS to match under a combined reflection and rotation transformation. Further, the LHS can only be matched to the rectangle with the label,

i.e. the most recently added shape, and therefore the rule can only be applied once to any shape that is added to the CWS.

The second category of labels is state labels (KNIGHT 1994). They are used to define *when* rules apply or, more precisely, the sequence and the repetition, i.e. (implicitly) the number of times a rule could be applied. This can also include the termination of the generation process, for example, if there are no further labels in a CWS and therefore no further rule is applicable. State labels are completely non-spatial, i.e., in comparison to spatial labels, not even the location matters. Often they are represented using numbers.

Figure 2-5 illustrates several rules using state labels. If only rule (b) is defined in a grammar, it can be applied once to the initial set (a), whereas the state changes from “1” to “2”. This prevents further application of the rule, restricting the number of possible iterations. Adding a second rule (c) to the grammar with a state label “2” in its LHS, the application is no longer restricted to a single iteration. As the application of the rule changes the state back to “1”, the rule sequence is determined to apply rule (b) and (c) repeatedly one after the other. To terminate the generation process, rule (d) can be applied that changes state “1” into state “3”. This state is not part of any of the other rules and, thus, no further rule can be applied. Similarly, rule (e) can be applied. This rule does not change the state, but instead completely removes the label.

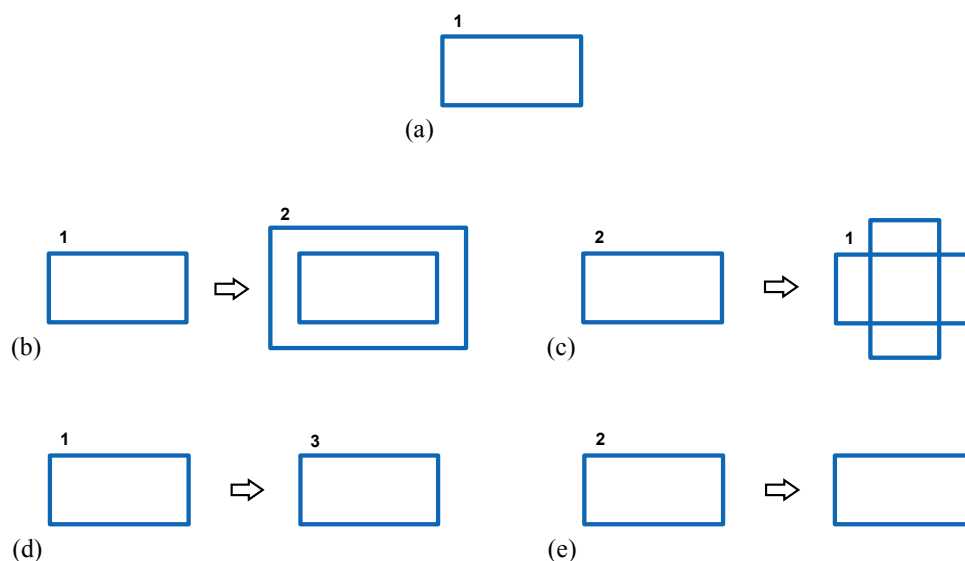


Figure 2-5: Rules using state labels

More than one label and also different kinds of labels (spatial and state), can be defined in either side of a rule.

The uses of labels described so far introduce constraints restricting the application of rules. However, labels can also be used to augment shapes with additional, non-geometric information or data. For example, they can distinguish or classify basic elements and parts of shapes, or they may have their own semantics to introduce other kinds of properties (STINY 2006). Other examples are the use of text labels for the denotation of shapes, which can also

be used to assign semantic or functionality to shapes. In the same way, labels can assign, for example, materials to solids that can be used to calculate the mass or render the texture and color of a solid (HEISSERMAN 1994).

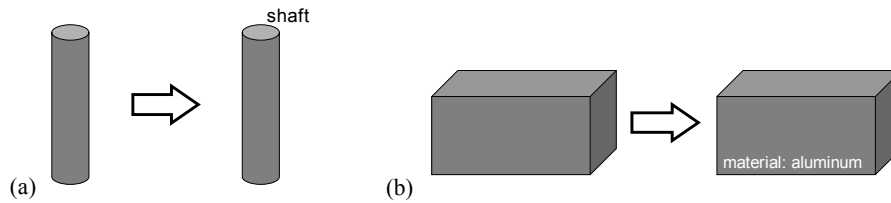


Figure 2-6: Rules with labels adding non-geometric information to geometry

For example, rule (a) in Figure 2-6 adds the textual label ‘shaft’ to a cylinder so that it differs from other cylindrical objects, and gives semantic meaning to the geometry. A further rule could be defined that only applies to a ‘shaft’ and no other cylinder. Similarly, rule (b) augments the block with information about its material.

The last category of labels shown in Figure 2-3 is the simplification of the LHS matching of a rule in the CWS. For the computational implementation of grammars the matching problem is a central challenge (KRISHNAMURTI & STOUFFS 1993), as the computer relies on formal representations of geometry that are not sufficiently suited to detect visual similarity of geometry under Euclidean transformations in a general way. This is especially true for curved shapes (JOWERS & EARL 2011). A few examples for spatial grammars exist that, while using curves, avoid shape-matching entirely by relying on labels (MCCORMACK & CAGAN 2006). In the example in Figure 2-7, the problem of matching curved geometry is circumvented using a combination of a straight line and a label in the LHS of rule (a). Instead of having to match curved geometry in the CWS (b), only the label and the straight line have to be matched.

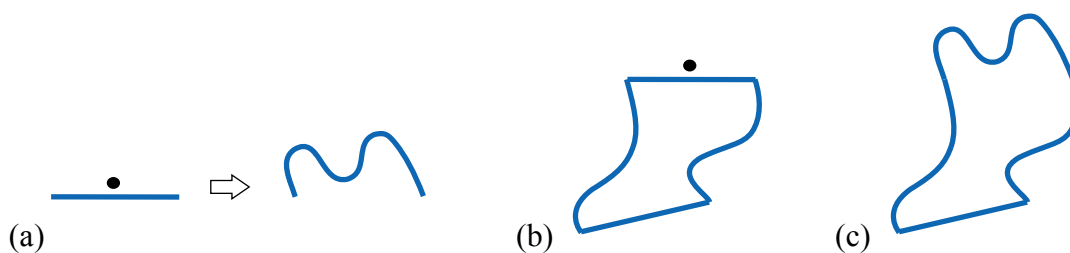


Figure 2-7: Example using simplified LHS matching

The label in Figure 2-7(b) indicates that a further rule has to be applied, i.e. the CWS with the straight line is an intermediate stage. The straight line acts as a reference, as only using the label in the LHS would not depict the rotation under which the rule has to be applied. In general, the reference does not necessarily have to be a line but multiple options using points and lines are possible to achieve this goal. The application of the rule results in the curvilinear design in Figure 2-7(c). It can be seen that this label type is an extension of the spatial label.

### 2.2.3 Parametric grammars

As an extension to the basic formalism STINY (1977, 1980a) describes parametric grammars where the rules are based on parameterized shapes and some or all of the parameter values are not predefined in the rule. Thus, a rule schema is defined that describes many different but related rules in one generalized rule. This allows for a wider variety of possible designs to be generated by fewer rules. Figure 2-8(a) shows a simple example of a parametric rule. The LHS is defined to be a parameterized quadrilateral, i.e. either the coordinates of its vertices or the lengths of its edges and the angles between them are not predefined. This allows for matching not only the quadrilateral shown in the LHS but any other quadrilateral, for example, the one shown in Figure 2-8(b).

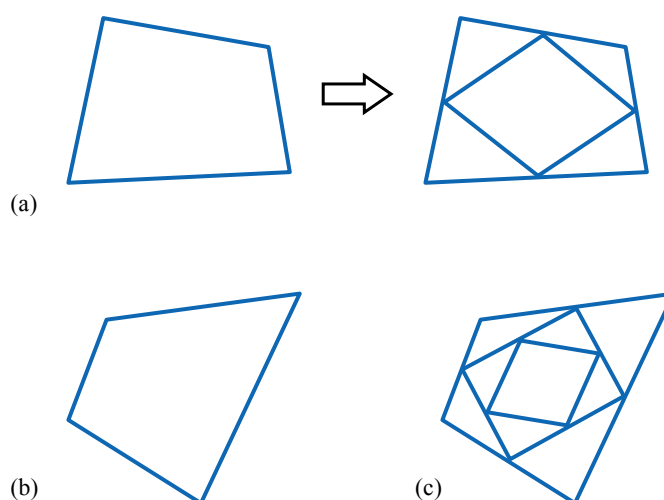


Figure 2-8: Example for a parametric grammar (adapted from STINY 1980a)

The RHS inscribes a new quadrilateral inside the matched one, whereas the vertices of the new quadrilateral are constrained to coincide with the midpoints of the edges. Starting with the shape in Figure 2-8(b) and applying the rule twice, generates the result shown in Figure 2-8(c).

## 2.3 Grammar interpreters and user interaction

GIPS (1999) provides a general definition of shape grammar interpreters, calling them a “program for shape grammar generation”. According to CHASE (2002) “the complete process of generating a design with a grammar involves two main stages”: the development of the grammar and its application. The development of a grammar mainly comprises the definition and modification of the vocabulary and the rules as well as the design of the initial set. The tasks in the application phase primarily include the selection of a rule to be applied, the determination of an object, or set of objects, to apply the rule to, as well as the detection of the matching condition under which the LHS of a rule matches an object in the CWS. These three tasks, which are often interdependent, are illustrated in Figure 2-9 together with the development phase shown on the left hand side. The figure further shows several exemplary

scenarios for user interaction with an interpreter system. The single tasks can be performed by different entities, namely the developer, the user, i.e. the designer using the grammar rules, or the computer. Depending on the degree of computer involvement, it can be differed between manual, semi-automatic and automatic application modes. Non-automatic modes are needed in cases where it is difficult to embed certain constraints in the grammar (CHASE 1989), so that the user can interactively direct the rule application to generate suitable solutions.

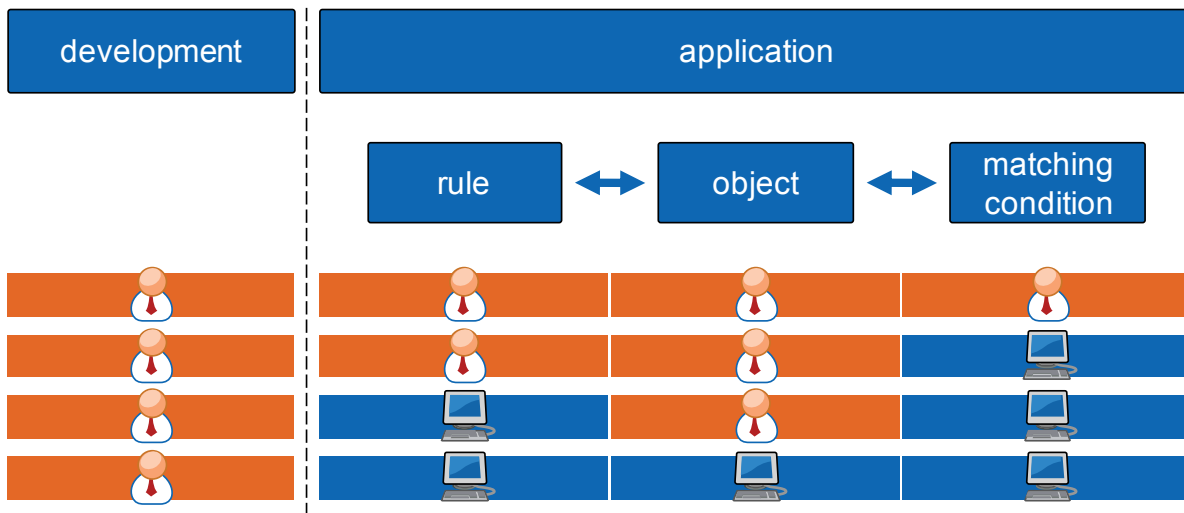


Figure 2-9: Development and application of grammar rules including user interaction scenarios (adapted from CHASE 2002)

CHASE (2002) differs between the two roles ‘developer’ and ‘user-designer’. For many existing implementations the development of the grammar was rather difficult and generally included hard-coding to implement the system. For a more general use of grammars, however, it is inevitable that designers themselves are enabled to cover the development phase and define their own rules in a familiar environment without requiring programming. CHAKRABARTI et al. (2011) suitably define a grammar interpreter as a software program that interactively and graphically supports the tasks involved in the development and the application phase, without the need for extensive programming.

## 2.4 Existing spatial grammars

The theoretical aspects of shape and set grammars have been discussed in a variety of publications. The basic ideas were introduced by STINY and GIPS (STINY & GIPS 1972, STINY 1980a, STINY 1982) and have further been evolved and explored in several other papers. Some of the important examples are the use of colors in grammars (KNIGHT 1989), the introduction of weights (STINY 1992), the determination of algebras of shapes (STINY 1991), and the distinction of different grammar types in KNIGHT (1999). It was also KNIGHT (1994) who first provided a comprehensive overview about shape grammars and so does STINY’s (2006) most recently published book *Shape*. The examples used to explain the theoretical

aspects of grammars are mostly kept on a simple level focusing on illustrating particular issues.

Other authors focused their work on mathematical or algorithmic aspects of grammars to create the foundation for computational implementations. It was especially Krishnamurti who published a series of papers in this area, for example about an algorithm for shape rule application (KRISHNAMURTI 1981) and about procedures for the recognition of three-dimensional line shapes (KRISHNAMURTI & EARL 1992). Noteworthy are also the contributions by GIPS (1999) discussing general issues with regard to the implementation of grammars, for example, user interfaces and parametric grammars, and by CHASE (2002) analyzing user interaction in grammar-based design systems.

While the research presented in the above publications has mainly a fundamental character, there exist many papers that concern specific spatial grammars for the generation of concrete artifacts. The majority of them stem from the areas of decorative arts and architecture. Early examples are the *ice-ray grammar* that generates Chinese lattice designs (STINY 1977) and a grammar for the design of Hepplewhite chair backs (KNIGHT 1980), both of which are based on parametric shape grammars. The often cited *kindergarten grammar* (STINY 1980b) illustrates the definition of design languages based on grammar rules using the three-dimensional Froebel building blocks from the kindergarten method. Further prominent grammars exist for the creation of different ground plans or house designs, for example, Palladian villas (STINY & MITCHELL 1978), prairie houses in the style of Frank Lloyd Wright (KONING & EIZENBERG 1981), bungalows from suburban developments in Buffalo (DOWNING & FLEMMING 1981) and houses in the Queen Anne style (FLEMMING 1987).

All of the examples mentioned in the previous paragraph are paper-based, i.e. the designs are created manually. Paper-based spatial grammars can serve as useful guides for the creation of designs (MCCORMACK & CAGAN 2002) and can easily be applied as most of the computationally difficult tasks, especially the recognition of the LHS of a rule in a CWS, are performed by the human grammarian. This, however, also makes the rule application error-prone and the creation of many different solutions by hand can be an extremely time consuming task, especially in 3D. To make spatial grammars more valuable by using them in computational design synthesis systems requires their implementation. To date, however, only a few spatial grammar systems have been computationally implemented. Nearly all of these have been developed in academia as experimental prototypes or for educational purposes. An overview of implemented systems up until 2002 can be found in CHAU et al. (2004). A few more have been implemented since; an overview of the latest systems is presented in MCKAY et al. (2012). In the remainder of this section, the most noteworthy implementations are presented. From these systems, the ones that are most relevant to this thesis are characterized in more detail in Table 2-1 at the end of this paragraph.

Many of the existing systems were implemented for one specific grammar and, from a user-point of view, they only support the application of grammar rules, not their development. They are typically hard-coded meaning that “a computer program was created entirely to implement a single set of rules” (MCCORMACK & CAGAN 2003). Some were first developed on paper and later implemented, for example, the *coffee maker grammar* (AGARWAL & CAGAN 1998) which was implemented as a two-dimensional system that generates three

orthographic views to derive a three-dimensional representation of a coffee maker from. It consists of 100 rules and makes extensive use of labels, for example to control the generation process and to circumvent the need to match curvilinear lines (MCCORMACK & CAGAN 2006). It also relies on many parameters that the user can assign concrete values to during rule application. LI & KUEN (2004) implemented a system<sup>1</sup> that provides a user interface for the application of rules that generate sections of two-dimensional Chinese wood-frame buildings represented by straight lines. The grammar consists of 48 rules that can be applied semi-automatically by the user or, for example for the roof, automatically by the system. In the first case, the user chooses the rules manually, whereas the system only activates the rules that are currently applicable to the CWS after every application step. To restrict the application to certain areas of the CWS, a variety of different labels are used and 10 rules are available exclusively to remove remaining labels from a building design. Another example that uses labels to guide the application process is a grammar for the generation of microelectromechanical (MEMS) devices (AGARWAL et al. 2000). Using a form-function coupling, labels are further used to ensure that some indispensable functional entities exist in any final design solution. The 2D grammar consists of 15 rules and was implemented as a prototype software system.

During the last fifteen years, an increasing interest in implementing more general interpreter systems, which can provide user support for both the development and the application phase, can be seen. To date, the majority of these systems are two-dimensional systems. *GEdit* (TAPIA 1999) allows for the visual development of rules as well as their interactive application, which includes subshape recognition. New shapes based on straight two-dimensional lines can be created in an external drawing program and are converted to a maximal line representation once they are imported to *GEdit*. Shape rules can also contain labels to restrict the application of rules to a certain shape or to deal with reflection symmetries. They are drawn in the external program as ovals which can be filled with patterns to define different labels. *Shaper2D<sup>2</sup>* (MCGILL & KNIGHT 2004) was implemented for educational purposes and allows for the direct visual manipulation of two shapes that are either rectangle, square, isosceles triangle or equilateral triangle by changing size, location or orientation. A maximum of two rules are iteratively and fully automatically applied where the resulting design is shown in real-time. Labels are used exclusively to remove symmetry of shapes. Their position can be changed to different predefined locations that are dependent on the degrees of the symmetry of a particular shape. DEAK et al. (2006) developed a software system named *Spadesys*, which can be used for what they call ‘CAD grammars’. CAD grammars as shown in the publication are a two-dimensional set grammar approach based on straight lines. Lines can be distinguished by assigning different line-types which can be seen as augmenting labels. Parametric grammar rules can be defined using the GUI of the system. They can be applied semi-automatically or automatically based on using scripting. Additionally, constraints and requirements can be defined importing an external chunk of code to the system.

---

<sup>1</sup> <http://andrew.li/research/Gram%20sim%20w%20descr.swf> (accessed January 14<sup>th</sup> 2012)

<sup>2</sup> <http://designmasala.com/miri/shaper2d/> (accessed January 14<sup>th</sup> 2012)



Most recent research makes an effort to extend on interpreters that can also handle curvilinear 2D shapes. MCCORMACK & CAGAN (2006) presented an implementation that is able to match curve-based shapes, including parametric shape recognition. The focus of this system is not on the definition or modification of grammars, but on the application and, especially, the matching of the LHS of rules in a CWS. It was used to implement several different examples, among them a grammar originally presented in MCCORMACK et al. (2004) that is able to generate front views of Buick vehicles. The system provides a more general approach as it is not implemented for one specific example. However, new grammars have to be defined using a text-based interface. Jowers implemented a shape grammar interpreter, *QI*, for shapes based on quadric Bézier curves (JOWERS & EARL 2010, JOWERS & EARL 2011). New rules can be visually designed by defining or manipulating the positions of the curves' control points. The system automatically detects the LHS of a rule under similarity transformations and includes subshape recognition. The *SubShapeDetector*<sup>3</sup>, also developed by JOWERS et al. (2008), allows for the import of hand-drawn sketches that act as the basis for the interactive development and application of rules. It includes a pixel-based approach for the detection of subshapes, enabling the use of curved shape grammars. The second version of this system, *SD2*<sup>4</sup> (JOWERS et al. 2010), additionally provides for the direct computerized drawing of shapes within the software and the definition of an arbitrary number of rules. A system with a similar range of functionality, however restricted to straight lines, but instead using maximal line representation for subshape detection and providing the possibility to define spatial labels in rules, was published by TRESCAK et al. (2009)<sup>5</sup>.

Up to now, only a few implementations of three-dimensional spatial grammars exist and the ones that do exist are mainly designed to deal with specific or restricted problems. Their focus is on the application of rules and the generation of design solutions. Therefore, the rules, or rule schemata, are pre-implemented using programming or scripting languages. SHEA & CAGAN (1997) implemented *eifForm* which provides for the automatic or semi-automatic application of grammar rules. It is based on the shape annealing method which combines shape grammars with simulated annealing to generate optimally directed designs. Using a labeled boundary graph as an internal representation for the structural shapes, a two-dimensional grammar for the generation of planar trusses and a three-dimensional grammar for the generation of domes and free-forms are developed. The system was later used in combination with an associative modeling system for the generation of cantilever roof trusses (SHEA et al. 2005). A further three-dimensional example is an implementation that uses a parallel grammar, i.e. a combination of graph and set grammars, for the generation of gear systems (STARLING & SHEA 2002). The geometric objects are represented by cylindrical primitives. The rule application is performed (semi-) automatically and the solutions are output as VRML<sup>6</sup> files that can be opened in an external viewer. The software prototype

---

<sup>3</sup> <http://www.engineering.leeds.ac.uk/dssg/downloads/requestForm.php> (accessed January 14<sup>th</sup> 2012)

<sup>4</sup> <http://www.engineering.leeds.ac.uk/dssg/downloads/requestForm.php> (accessed January 14<sup>th</sup> 2012)

<sup>5</sup> <http://sourceforge.net/projects/sginterpreter/> (accessed January 14<sup>th</sup> 2012)

<sup>6</sup> „Virtual Reality Modeling Language“, a standard file format for representing 3D vector graphics

additionally includes simulation-based evaluation and optimization methods to generate optimally directed solutions (STARLING & SHEA 2005). *MALAG* is a system that is implemented for the grammar-based design of mass-customized housing using the example of Alvaro Siza's patio houses at Malagueira (DUARTE 2005). It provides a GUI window, in which the user can select different requirements, e.g. the number of people living in the house, the number of different kinds of rooms, spaciousness, capacity, etc., and can assign weights to some of them. Based on that data, a design brief, i.e. a symbolic description of a house, is generated. The design brief is used as input for the shape grammar based generation of a house design that fulfills the given requirements as good as possible. The output is a 3D model that can be displayed in a web-based design viewer (DUARTE & CORREIA 2006).

Besides the latter systems, a few 3D implementations exist that are not designed to deal with one specific problem. Their focus is also on the application side and the rules have to be pre-implemented using programming or scripting languages. *Genesis* is currently the only known commercially used implementation of a spatial grammar system and, consequently, can be considered very mature. It was implemented at Boeing (see e.g. HEISSERMAN et al. 2004) based on the original system that Heisserman developed in his PhD thesis to generate, for example, alternative Queen Anne houses, Sierpinski sponges, and fractal-like mountains (HEISSERMAN 1994). At Boeing it is used to support the development and interactive application of rules for tubing designs in aircrafts, but, generally speaking, it is not restricted to this application area and could work in different domains. It enables designers to explore solution spaces as well as to evaluate, compare and merge design alternatives. Rather than as replacement rules, i.e. replacing the match to the LHS of a rule with the RHS, the design rules are formulated through logical match conditions and design transformations. The geometric objects as well as the rules are implemented upfront for later use by designers. PIAZZALUNGA & FITZHORN (1998) used a commercial solid modeling kernel to develop a three-dimensional 'interpreter'. In this system the rules are defined using a programming language that makes most of the kernel's capabilities accessible. The application of these rules requires user interaction, where the user chooses a rule and an object to apply the rule to. Labels represented by single letters are used to restrict the application of rules to a certain geometric object, to remove symmetries and in some cases to define a certain rule sequence. CHAU et al. (2004) presented an approach called SGS that can handle rectilinear and curvilinear basic elements in 3D space. The shapes and rules are created and edited in an external text file and applied to generate wire frame models. It uses points in combination with different single letters as labels to restrict rule application to certain parts of the CWS and to remove symmetries. Based on the use of letters, the labels can also be used to realize state labels. The implemented example of a Coca-Cola bottle grammar uses straight lines and labels in the LHSs of the rules to circumvent problems with matching curvilinear lines.

In addition to the systems described so far, a few three-dimensional implementations provide limited capabilities to customize rules. However, they do not allow for new rules to be defined without programming. The rules are realized as pre-implemented rule schemata to which the user can assign specific values. The sizes of shapes and their spatial relations, as well as the number of rule applications, can be defined before the application of a rule; the rule application itself is executed automatically. WONG & CHO (2004) started from the concept of *Shaper2D* (see above) and extended it to simulate three-dimensional blocks into a

single rule. In the revised version of this system, *Shape Designer V.2* (WONG et al. 2005), the user can choose from several different predefined rule schemata. A rule is applied by typing a short command into a text input window. Commands consist of a short name for the chosen rule schema and the required values for the parameters. Depending on the rule schema, these include some or all of the following: number of iterations, translations, rotations, scale factor and sizes. Some rule schemas also include labels whose location is represented as one of the parameters that values can be assigned to. *3DShaper* (WANG & DUARTE 2002) provides a dialog window for a pre-implemented rule schema to define the sizes and the spatial relationships of two blocks by typing in the required values. In doing so, one or two additive rules are specified which are immediately applied and saved in data files. A visual representation of the rules, as well as of the generated design, is available after opening the created files in an external viewer. In order to eliminate the ambiguity in shape rule applications that arises due to the symmetry of blocks, labels are defined in the rule schema. The position of a label can be set to one of the corners of a specified block.

The latest prototype implementation of a grammar system that provides capabilities for the (re-)definition of rules, *Grammar Environment* (LI et al. 2009a, LI et al. 2009b), is based on the system by CHAU et al. (2004) but extends it with the capability to define new shapes and non-parametric, (labeled) rules in a graphical environment. The application of rules generates wire frames, like the original system, however, it is restricted to the use of points and straight lines. 2D shapes and rules can be directly created in the *Grammar Environment* system, but for the design of 3D shapes an external shape editor is needed. For this purpose, the commercial CAD system AutoCAD is used. The data of the designed geometry can be imported to the *Grammar Environment* system to be used for rule definition and application.

Table 2-1 presents an overview of the characteristics of the most relevant spatial grammar implementations to this thesis. Their relevance was determined by their three-dimensional capabilities and/or the possibility to visually define and modify rules, as these are the foci of this thesis. The data was gathered from published papers and system tutorials, by testing a working copy of the corresponding implementation and also in direct correspondence with the authors.

Table 2-1: Relevant spatial grammar implementations and their characteristics

| name                                     | GEEdit  | Shaper2D   | QI   | SubShapeDetector  | Shape Grammar Interpreter                                   | SD2  |
|--|---|--|--|---|---|--|
| reference                                | Tapia (1999)                                    | McGill & Knight (2004)   | Jowers & Earl (2011)                                       | Jowers et al. (2008)  | Trescak et al. (2009)                                       | Jowers et al. (2010)   |
| dimension(s)                             | 2D  | 2D   | 2D   | 2D  | 2D  | 2D   |
| shape types                              | straight lines                                  | rectangle, square, isosceles/ equilateral triangle; customized shape(s) (to replace one of the standard shapes per rule)   | quadratic Bézier curves                                    | arbitrary (pixel-based)   | straight (poly-)lines                                       | arbitrary (pixel-based)  |
| max. number of shapes                    | unrestricted                                    | 1 (LHS), 2 (RHS)   | unrestricted   | unrestricted (no explicit single shapes)                        | unrestricted  | unrestricted (no explicit single shapes)   |
| max. number of rules                     | unrestricted                                    | 2  | unrestricted   | unrestricted (one loaded at a time)                             | unrestricted  | unrestricted   |
| rule format                              | additive, subtractive, replacing                | additive   | additive, subtractive, replacing                           | additive, subtractive, replacing                                | additive, subtractive, replacing                            | additive, subtractive, replacing   |
| parametric rules                         | no  | no   | no   | no  | no  | no   |
| labels                                   | spatial   | spatial  | no   | no  | spatial   | no   |
| definition/editing/manipulation of rules | visual, interactive (restricted to definition)  | direct visual manipulation of shape sizes/location/ orientation  | visual, interactive  | visual, interactive (restricted to copy& paste of (sub-)shapes) | visual, interactive   | visual, interactive  |
| LHS matching                             | automatic, including subshapes                  | rules are always applied to most recently added shape; LHS is always subset of RHS of previously applied rule; automatic detection of the corresponding transformation | automatic, including subshapes                             | automatic, including subshapes                                  | automatic, including subshapes                              | automatic, including subshapes   |
| transformations for matching             | translation, rotation, scale, reflection        | translation, rotation, reflection  | translation, rotation, scale, reflection                   | translation, vertical reflection                                | translation, rotation, scale, reflection                    | translation, rotation, scale (manual specification of factor), vertical reflection |
| application mode                         | semi-automatic                                  | automatic  | semi-automatic   | semi-automatic  | (semi-)automatic  | semi-automatic   |
| max. number of applications              | no explicit restriction                         | 25   | no explicit restriction                                    | no explicit restriction   | 100   | no explicit restriction  |
| one single, integrated system            | external system needed to create new shapes     | external file for customized shape necessary   | yes  | import of sketched shape(s) needed                              | yes   | yes  |
| unique characteristic(s)                 | preview of all possible results applying a rule | real time design generation and display; direct visual manipulation of shapes  | based on parametric curves; curvilinear subshape detection | pixel-based, curvilinear subshape detection                     | generation chain preview; generates all possible next steps | pixel-based, curvilinear subshape detection; preview of possible replacements      |

Note:

- *reference*: one of the latest publications about the implementation
- *dimension(s)*: dimension of the space in which the shapes/rules are used
- *shape types*: types of shapes that are used in the given rules or that are provided in the implementation for the definition of rules; e.g. some systems provide straight lines to define other, more complex shapes, others are restricted to a given set of predefined shapes, etc.
- *max. number of shapes*: maximum number of shapes that can be used in the definition of a rule or that are given in a rule schema
- *max. number of rules*: maximum number of rules that can be defined in a grammar; some systems are pre-implemented rule schemata that are restricted to a certain number of rules
- *rule format*: the types of rules the implementation supports/provides – these can be “additive“ if only part of the RHS equals the LHS, “subtractive“ if the RHS equals only part of the LHS or “replacing“ if the complete LHS is substituted by the RHS
- *parametric rules*: depicts whether the implementation uses or allows for the definition of parametric rules as described in STINY 1980a
- *labels*: depicts whether the implementation uses or allows for the definition of labels and the kind of labels (according to the overview in Figure 2-3)
- *definition/editing/manipulation of rules*: depicts the means of user-interaction for the definition of new rules, editing existing rules or the manipulation of rule schemata
- *LHS matching*: characterization of how the LHS (shapes and transformations) of a rule is matched in the CWS
- *transformations for matching*: the kinds of transformations that can be used to find matches of the LHS in the CWS
- *application mode*: the level of human intervention required or allowed in application steps, e.g. the selection of a rule or an object to which a rule is applied
- *max. number of applications*: some implementations are restricted to a certain number of rule applications (or iterations)
- *one single, integrated system*: some implementations require external systems, e.g. for the definition of new shapes or to view generated designs
- *unique characteristic(s)*: characteristics that can only be found in the corresponding implementation

| name                                     | Genesis  | 3D grammar interpreter   | 3DShaper   | SGS  | Shape Designer   | Shape Designer V.2  | Grammar Environment  |
|--|--|--|--|--|--|---|--|
| reference                                | Heisserman et al. (2004)   | Piazzalunga & Fitzhorn (1998)  | Wang & Duarte (2002)   | Chau et al. (2004)   | Wong & Cho (2004)  | Wong et al. (2005)  | Li et al. (2009)   |
| dimension(s)                             | 3D   | 3D   | 3D   | 3D   | 2D or 3D (2D methods simulating 3D objects)  | 2D or 3D  | 3D   |
| shape types                              | 3D polyhedral and 3D swept solids  | blocks   | blocks (cube, oblong, pillar, square); possibly substitutable by a customized shape  | straight lines and circular arcs   | triangle, rectangle, square, block (2D simulating 3D)  | straight lines, triangle, rectangle, pentagon, block, triangular pyramid; in 3D only one single type used at a time for a rule schema   | straight lines   |
| max. number of shapes                    | unrestricted (rules can operate on other representations than shapes as well)  | 1 (LHS), 5 (RHS, in the given examples, theoretically expandable)  | 1 (LHS), 2 (RHS)   | unrestricted   | 1 (LHS), 2 (RHS)   | 1 (LHS), 20 (RHS, in the given schemata, theoretically expandable)  | unrestricted   |
| max. number of rules                     | unrestricted   | theoretically unrestricted (new ones could be coded)   | 2  | unrestricted   | 2  | 3 (2D) and 1 (3D) in the given schemata   | unrestricted   |
| rule format                              | described in terms of logical match conditions and design transformations (additive, subtractive or replacing/modifying) | additive, replacing  | additive   | additive, subtractive, replacing   | additive   | additive, replacing   | additive, subtractive, replacing   |
| parametric rules                         | yes  | no   | system represents a rule schema: pre-implemented shapes and their spatial relations are parametric   | no   | no   | rule schemata: depending on a particular schema, the spatial relations and scaling factor or some of the sizes are parametric   | no   |
| labels                                   | spatial, state, non-geometric information  | spatial, state   | spatial  | spatial, LHS matching simplification   | spatial  | spatial   | spatial, state, non-geometric information  |
| definition/editing/manipulation of rules | hard coding (high-level language)  | hard coding  | numerical input form for the assignment of concrete values to the given parameters for the derivation of a rule instance   | text file editing  | manipulation of a shape's location, translation or scaling using sliders; immediate update of the displayed geometry | command line input for the assignment of concrete values to the parameters of a loaded schema for the derivation of a rule instance; definition of new schemata programming Prolog scripts theoretically possible | visual, interactive (within a restricted design space)   |
| LHS matching                             | design rules encode the logical match conditions - for applying the design transformations                               | manual (sub-)shape selection   | no explicit matching; transformation for each new shape is separately calculated including the transformations of all previously added shapes and regarding the label position | automatic (sub-)shape recognition based on user-specified transformation (cf. "transformations for matching")                        | based on the "Shaper2D" approach   | not clear:<br>- 2D: mainly based on the "Shaper2D" approach<br>- 3D: similar to 3DShaper including scaling; rule is recursively applied to all existing blocks  | automatic, including subshapes   |
| transformations for matching             | (rules can transform shapes using) affine transformations, multiple transformations                                      | calculated based on the manual (sub-)shape selection; considering translation, rotation, scale, reflection | cf. "LHS matching"   | manually specified by the user selecting a set of point triples in LHS and CWS; can realize translation, rotation, scale, reflection | cf. "LHS matching"   | cf. "LHS matching"  | based on the "SGS" approach; extension to automatic detection of all relevant pairs of point triples |
| application mode                         | semi-automatic (interactive) or automatic  | semi-automatic   | automatic  | semi-automatic   | automatic  | automatic   | semi-automatic   |
| max. number of applications              | no explicit restriction  | no explicit restriction  | no explicit restriction  | no explicit restriction  | 14   | no explicit restriction   | no explicit restriction  |
| one single, integrated system            | yes (import of surrounding geometry from external CAD systems possible)  | yes (several different windows/views for coding and graphical output)                                      | external viewer needed to display the actual rules and the resulting design; external file for customized shape necessary  | text file editor needed  | yes  | yes (including an internal editor for editing rule schemata)  | external CAD applet needed for the creation of 3D shapes   |
| unique characteristic(s)                 | only known commercially used implementation  | based on a commercial solid modeling kernel  |  | using circular arcs in 3D space  | real time design generation and display as in "Shaper2D"   |   | preview of all possible results applying one or all rules  |

Table 2-1: Relevant spatial grammar implementations and their characteristics (continued)

## 2.5 Relation to Knowledge-Based Engineering in CAD

Aiming at making CAD systems more active based on a spatial grammar approach, it should not be neglected that there has been significant progress since the inception of the first CAD approaches. Many enhancements have been introduced with a focus on supporting designers with easier creation and modification of designs. This is an important aspect in engineering as the modeling process usually is iterative taking many rework-cycles influenced by different requirements. The changeability of existing geometric models in CAD systems was significantly improved with the introduction of feature-based, parametric modeling in the beginning of the 1990s (SHAH & MÄNTYLÄ 1995). Even since, approaches in CAD have further evolved. Especially techniques attempting to preserve and reuse design knowledge have been developed. The approaches that are most closely related to the spatial grammar approach are commonly summarized under the term Knowledge-Based Engineering (KBE).

Generally speaking, KBE is about the “use of advanced software techniques to capture and re-use product and process knowledge in an integrated way” (STOKES 2001). The main aim is to assist or automate routine, repetitive and time consuming design tasks. The term KBE has been used in two different ways in both academic research and industry. The original use stems from expert systems where symbolic systems were created to encode domain and problem-solving knowledge that could automatically be reasoned about using an inference engine to solve synthesis, diagnosis, analysis and planning tasks (LEVITT 1991). Research activities are, for example, about capturing, re-using and sharing engineering knowledge or improving the methodological support (VERHAGEN et al. 2011). For the implementation, stand-alone KBE development packages<sup>7</sup> exist, which are not dependent on any proprietary CAD system. They are often language-based (COOPER & LA ROCCA 2007), i.e. they provide a programming language to create KBE applications for specific problems. An example is the BAe Airbus wing rib design KBE tool<sup>8</sup>, which generates wing ribs automatically in any location on an airplane wing (COOPER et al. 2001). Based on a programming language, these KBE development tools are very general and can be very powerful, but at the same time they also require a skilled KBE development engineer (COOPER et al. 2001). Due to their generality, it might be possible to implement spatial grammar approaches based on some of these languages as well.

However, as the focus of this thesis is to create a general spatial grammar system that does not require programming, the original use of the term KBE and the mentioned development packages are not discussed in detail here. Instead, the focus of the discussion is on tools that are ‘visually’ usable by the design engineers while they work in a familiar software environment, not needing any further stakeholder like a programmer or a knowledge engineer. These tools, claiming KBE capabilities from the CAD vendors’ perspective, can nowadays be found as integrated parts of mainstream commercial CAD software<sup>9</sup> (COOPER & LA ROCCA 2007). They usually contain knowledge in limited forms and some of them directly evolved

---

<sup>7</sup> for example Genworks, <http://www.genworks.com/> (accessed January 14<sup>th</sup> 2012)

<sup>8</sup> implemented using ICAD which does not exist anymore

<sup>9</sup> popular examples are NX Knowledge Fusion or CATIA V5 Knowledgeware

from existing CAD functionality, e.g. parametric modeling. Their categorization corresponds to the second, more recent use of the term KBE, which is less restrictive and can refer to any process-related or reference knowledge used within CAx tools (FENVES et al. 2005).

Keeping in mind that the detailed capabilities of each KBE approach can differ considerably, Table 2-2 gives an overview of the general characteristics of the most relevant KBE approaches in CAD systems in comparison to spatial grammars. The fundamental characteristics of each approach address the questions as to whether or how the geometry of an existing model is changed by using single approaches (*‘Geometry change’*) and if its topology stays the same or not (*‘Influence on topology’*). The *‘Character’* indicates the way the approaches alter an existing model, for example, whether they add in new geometry or update the existing one. Further, it is specified how the location of the modification in an existing model is determined when using the single approaches (*‘Location of application’*) as well as the number of solutions that can be created and to what extent they differ (*‘Solutions’*).



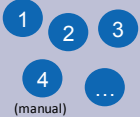
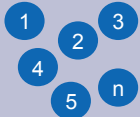

|                                | Pure Automation   | Parametric Models   | User-Def. Features  | Rule Scripting  | Spatial Grammars  |
|--------------------------------|---|---|---|---|---|
| <i>Geometry change</i>         | no  | parametric relations  | parametric relations  | rule-based  | rule-based / parametric   |
| <i>Influence on topology</i>   | if used with other model  | very limited  | model it is inserted into   | only predefined   | different topologies  |
| <i>Character</i>               | additive  | updating  | additive / updating   | modifying   | substituting  |
| <i>Location of application</i> | static  | --  | manually defined  | determined in rules   | geometry dependent  |
| <i>Solutions</i>               | always identical  | different dimensions  | user dependent  | predictable   | possibly unexpected   |
|                                |  |  |  |  |  |

Table 2-2: Characteristics of different KBE approaches in comparison to spatial grammars

The first approach, ‘pure’ design automation, is about the generation of a model or a part of a model used to automate frequently recurring design tasks. It is usually based on the execution of a script<sup>10</sup>, whereas scripts do not necessarily have to be written manually. The systems provide functionality to record the user’s modeling command sequence and save it in a script file commonly called ‘macro’ (CHOI & HAN 2002). A macro can be executed to automatically

<sup>10</sup> for example Visual Basic is used as a script language in many CAD systems

recreate the exact same geometry and topology, i.e. one single solution that is always identical. Only if used in combination with another geometric model, it can have an influence on the topology of the overall result. A macro can usually only add the geometry as saved according to the modeling sequence and not, for example, modify existing geometry. A modification is only possible applying the macro to the exact same model it was created with or rewriting parts of the recorded script of the macro to generalize it. The geometry created by a macro is always inserted in the exact same location as recorded, no matter what the surrounding or interpenetrating geometry is.

Parametric CAD models allow for the creation of geometry that can be rapidly changed and updated. Once a model is designed, user-defined mathematical equations can be used to define parametric relations between different parameters (LEE 1999). Changing the value of only one or a few input parameters, a new instance of the parametric model is automatically calculated (SHAH & MÄNTYLÄ 1995). The geometry of the model is updated according to the defined parametric relations. The solutions created are instances of the initially designed model and generally only differ in terms of their dimensions, whereas this is not restricted to scaled versions only. Changes of the topology can be realized in very limited form, e.g. by making the number of instances of a pattern dependent on other parameters.

A further KBE concept that can often be found in CAD packages are templates. There are several different kinds of templates, e.g. scripting templates that lower the coding effort for writing scripts or templates that contain a basic structure and geometric elements as the basis for the design of different detailed variants of a part. An advanced kind of geometry template is User-Defined Features (UDFs). Features are high level parametric modeling entities that have some engineering significance and ease the design of recurring shapes (SHAH & MÄNTYLÄ 1995). The 'standard' features most commonly provided by mechanical engineering CAD systems are, for example, fillets, chamfers, pockets, holes, etc. UDFs enable designers to define their own customized compound features (e.g. see HOFFMANN & JOAN-ARINYO 1998). These help to facilitate the design of frequently occurring, very similar subparts of geometric models as needed, for example, in product variants. A UDF can be composed of different standard features and might contain parametric relations and parameters that can be changed to instantiate the geometry of the UDF once it is inserted into a design. In conventional CAD design, the designer has to position the UDF in an existing model and manually define the required reference elements. The topology of a UDF itself always stays the same but it changes the topology of the model it is inserted into. Theoretically an unlimited number of different solutions can be created depending on the different UDFs used and the locations in which they are manually added to an existing design by the user.

The last KBE functionality considered is the possibility to define a limited form of rules. Some CAD systems provide a wizard for their definition so that no or hardly any programming knowledge is needed. The user can compile different rules combining predefined script statements with geometric parameters. They typically consist of if-else statements and relational operators. In comparison to higher level scripting languages, no advanced constructs like loops are available. Depending on the fulfillment of the defined conditions, the geometric model is modified in the according location(s). Usually the



modifications are limited to showing or hiding predesigned geometric entities and changing their parameters or the execution of macros (see pure automation). Depending on what parts are currently set visible, different topologies can be represented and in combination with different parameter settings, a wide number of solutions can be achieved. However, as every used component has to be predesigned, the solutions are all predictable.

In comparison to the presented KBE approaches in CAD, spatial grammar approaches are based on rules that can be parametric but are not expressed by if-else branches or other programming statements. Instead, they directly work on shapes whereas the location of their application depends on where the LHS of a rule can be detected in an existing design. The rules 'know' where they can be applied in comparison to, for example, UDFs which require manual specification of their position. The detected LHS in the existing design is substituted by the geometry in the RHS of the rule. This allows for the definition of rules with not only 'additive', but also 'subtractive' or 'replacing' character. In combination with different rule sequences, it is possible to generate a wide range of topologically different solutions that are not predesigned, as in case of rule scripting. Therefore, the solution space that is defined by a grammar often contains unexpected solutions that were not thought of upfront during the definition of the grammar.



### 3. Benefits and challenges

In theory, the computational use of spatial grammars can be very beneficial to support designers. The advantages of such systems, however, have not yet been realized because the current grammar approaches and implementations face a number of challenges that have yet to be adequately solved, several of which have already been mentioned throughout the previous chapters. This chapter summarizes the theoretical benefits of using a spatial grammar approach in CAD design and the challenges that exist for the development of an ideal 3D spatial grammar system.

Spatial grammars are an approach that provides the possibility of computationally synthesizing designs. Consequently, they have the potential to help make CAD systems a more active design partner. The rules that are defined for a grammar are often much less complicated than the actual designs they produce (STINY 1980b). Grammar-based synthesis of designs involves several aspects that are beneficial with regard to designer support. Routine design tasks, which are often time-consuming and tedious, can be automated, thus, leaving more time for creative activities and helping to reduce errors (CHAKRABARTI et al. 2011). Especially in mechanical engineering design it is often the case that many revisions are needed before the final version of a design is achieved. In case of major changes, for example because of changing requirements for a product, a grammar-based approach means that the design does not have to be completely reworked; rather, it means that a few rules will have to be modified, added or deleted. Based on such changes, the quick generation of a new solution is possible.

Besides saving time in the creation of a single solution, one of the most powerful aspects of spatial grammars is their ability to rapidly generate a range of design alternatives. In changing a few simple rules, grammar-based approaches allow for the construction of a multiplicity of complex designs (STINY 1980b), helping with regard to the creation of product variants. Grammar-based approaches can, therefore, be highly supportive in meeting growing customer demand for one-of-a-kind design within a class of products (CAGAN 2001). There is no need to specify every single solution when higher-level descriptions, i.e. a small number of abstract rules, can generate them automatically (MCCULLOUGH 1996). Even in those cases where only one solution is needed, the generation of different variants can be helpful in finding better or even optimal solutions.

Spatial grammars can also help to enhance design creativity by offering spatially novel, inventive solutions beyond what a designer might think of or that may not be obvious to the designer due to limitations of knowledge or fixation (CHASE 1989, CHAKRABARTI et al. 2011). Thus, spatial grammar systems usually aim to assist and not to replace the designer (HEISSERMAN et al. 2004, JOWERS et al. 2008).

A grammar-based approach helps to encode design knowledge, saving knowledge chunks and making them reusable. Formalizing knowledge can help designers who develop their own rules to move away from intuitive methods of design towards a more structured, methodological procedure (CHASE 1989). This can lead to a better understanding of a design

problem (CHAKRABARTI et al. 2011), as designers have to actively and consciously think about their “ideas and knowledge about possibilities for design in an explicit and detailed way” (STINY 1980b).

The majority of the benefits discussed can only be derived through a general computational implementation of spatial grammars. As a method that follows a strict formalism, they seem to lend themselves well to computer implementations (CHASE 1989). However, as can be seen in the previous chapter, many of the existing systems are restricted in some form or another. There are a lot of different challenges to implementing a grammar interpreter, especially regarding the issues of generality and usability. Many challenges have been identified in earlier work, for example, by GIPS (1999), CHAU et al. (2004), LI (2005) or, most recently, by CHAKRABARTI et al. (2011) and MCKAY et al. (2012). The most relevant challenges with regard to the approach presented in this thesis are summarized in the following.

Using programming for the development of rules and their application provides high flexibility for the implementation of various geometry or vocabulary, especially in expressing relations between, or constraints on, geometric objects. However, as described in the introduction, programming in a design environment has several drawbacks. A system that designers really want and need to use has to make trying out grammars easier than trying them by hand (GIPS 1999). With regard to better acceptance, interpreter implementations should also be better integrated with the software toolset designers are used to working with, for example with CAD systems.

For increased usability, many existing interpreters focus on the application of rules. However, a generalized interpreter that provides for facilitated use of grammars without programming must support the tasks in both stages – development and application – in an interactive, visual manner. Especially in three-dimensional space, where spatial thinking is more demanding, it is important to have direct visualization of the rules as they are developed. Without direct visualization, the designer must first write code, and possibly compile it, to see whether the intended geometric objects and spatial relations are generated.

Once defined, it should also be possible to modify existing rules in an easy way, especially because defining a grammar “often tends to be a ‘generate and test’ cycle” (CHASE 2002). As a new grammar is often tested and modified several times before it describes the intended design language, this demands proper user support not only for the development of rules, but also for their application. The possibility to easily edit rules becomes even more important with regard to the use of a grammar system in mechanical engineering, as the rules can be seen as chunks of knowledge that continuously evolve during the product development process, and are influenced by external requirements stemming from other domains, e.g. manufacturing constraints.

A visual, interactive approach has to provide a set of standard commands whose functionality works on a higher level than programming, but visually usable via a user interface, therefore making it easier to work with. The interplay of the different single commands needs to provide high flexibility to enable a generalized design and use of grammars. In that regard, specific challenges arise, especially in conjunction with a three-dimensional approach. Being the basis for the definition of a grammar, the vocabulary should be as flexible as possible to allow for the creation of a wide variety of geometric objects. For the definition of rules, these

geometric objects have to be graphically represented to allow for direct manipulation and positioning in 3D space. The latter requires the robust handling of 3D transformation operations, at least for translation and rotation. For the application of a rule, the geometric objects as well as their location and orientation have to be defined in a way that enables the automatic matching of the LHS. Realization of automatic LHS matching, however, should not restrict the complexity of geometric objects that can be used for the definition of rules. In general, computer-based recognition, or matching of 3D objects under transformations and parametric relations, is a known and difficult problem. This is mainly because, to date, a general technique to computationally perform the same functions as the human visual perception system has not been developed (IYER et al. 2005).

To allow for a wider variety of possible designs, not only the definition of non-parametric but also the definition of parametric rules should be possible along with their automatic application. To enable the possibility of better directing the generation of the solutions, it should not only be possible to automatically, but also manually or semi-automatically, apply rules. Further, the number of geometric objects in a rule should not be restricted. The same applies to the number of rules that can be defined and applied since the interaction of several different rules generally leads to a wider variety of alternative solutions.

In summary, the requirements for an ideal 3D spatial grammar system are:

- general, i.e. not restricted to a specific problem,
- an unrestricted vocabulary allowing a wide variety of complex, 3D geometric objects, but at the same time
- automatic matching of the left hand side of rules under transformations and considering parametric relations,
- graphical representation and direct, visual manipulation of objects in rules,
- robust handling of transformation operations in 3D,
- definition of parametric rules,
- mechanisms for the visual incorporation of constraints including labels,
- an unlimited number of rules,
- an unlimited number of objects that can be used in rules,
- interactive application of rules (automatic, semi-automatic, manual),
- support for both definition of new rules and editing/modification of existing rules,
- better integration with other design software, e.g. CAD systems, and
- an intuitive user interface requiring little to no programming.

With regard to the overall goal of this research and the challenges discussed in this chapter, the following table gives an overview of the contributions of this thesis and the chapters they are addressed in.

| <b>New approach for creating a general, visual, interactive 3D spatial grammar system</b>               |                             |
|---|-----------------------------|
| Visual definition and modification of rules   | 4.1.1 and 5.2               |
| Definition of additive, subtractive and substituting rules  | 4.1.1                       |
| Wide range of shapes for rule definition<br>(parameterized primitives, Boolean and sweeping operations) | 4.1.1, 4.3.1,<br>4.3.2      |
| Parametric rules  | 4.2                         |
| Consolidated concept for labels   | 4.4                         |
| Unrestricted number of rules, shapes in rules and applications of rules                                 | 4 and 5                     |
| Interactive rule application  | 5.3                         |
| Automatic LHS matching  | 4.1.2, 4.2.2,<br>4.3, 4.4.2 |
| Collision detection   | 4.5                         |
| Integration into CAD  | 5                           |

*Table 3-1: Overall goal, research contributions and the related chapters of this thesis*

## 4. Approach

The approach developed in this thesis addresses the challenges that exist for three-dimensional spatial grammar systems as discussed in the previous chapter. It is conceptualized in a way that allows for developing and applying grammars in a visual, interactive manner. This chapter first introduces the basics of the proposed approach, as well as the development and application of non-parametric rules. The second part describes the enhancement of the approach to parametric rules, followed by an extension using objects based on Boolean and sweeping operations. The concept and usage of 3D labels is presented in another sub-chapter. Using collision detection in this spatial grammar approach concludes the chapter.

### 4.1 Basics and non-parametric rules

The approach described in this thesis is based on a set grammar formulation of spatial grammars. The vocabulary consists of a set of parameterized three-dimensional primitives, namely, box, torus, cone, cylinder, sphere and ellipsoid. Defining more than one primitive in a side of a grammar rule, as well as variations of the defined parameters, allows for the description of a wide range of geometry. Figure 4-1 shows a base version (left) and a version with alternative parameter values (right) for each primitive used, including the parameterization.

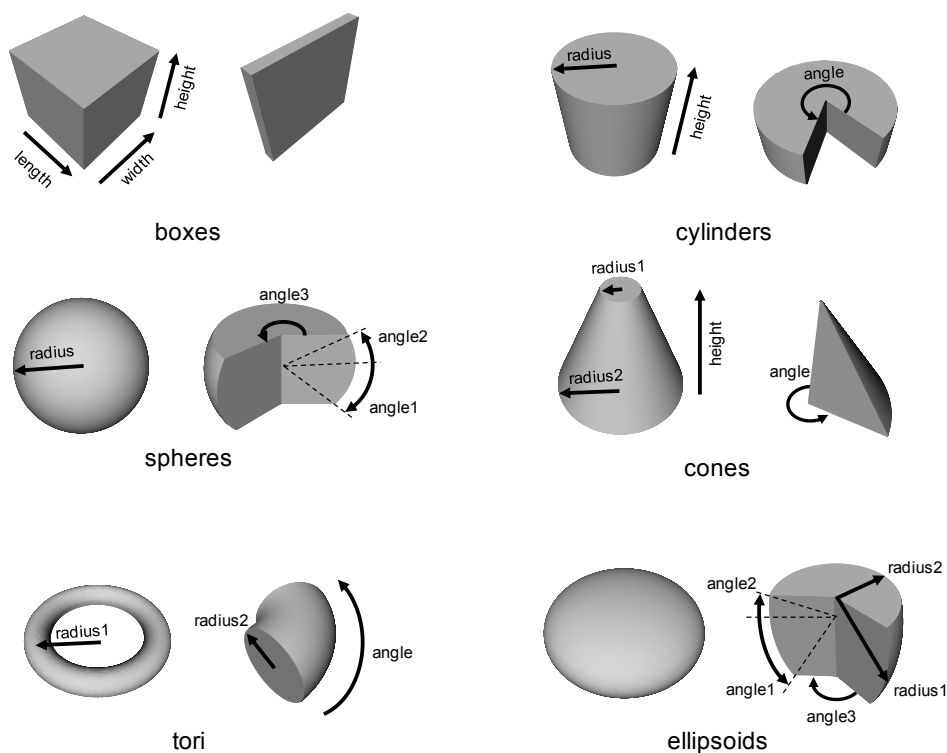


Figure 4-1: Primitives used illustrated by two different instances

This section introduces the basics of the approach in conjunction with non-parametric grammar rules. Following the definition by CHASE (2002), it is subdivided in two parts: the development and the application of a spatial grammar.

#### 4.1.1 Development of rules and transformations of objects

To define rules in the form  $A \rightarrow B$  (cf. 2.2.1), one needs to define the geometric object(s) in  $A$  and  $B$  as well as their spatial relations. The fundamental aspects for the visual definition of spatial grammar rules are the creation and the positioning of geometric objects in 3D space. The basis for the definition of geometric objects is the given set of parameterized 3D primitives shown in Figure 4-1. An object is created by choosing one of these primitives and by assigning values to its parameters. As is usually the case in the design of solids in mechanical engineering CAD, the input is realized numerically so that exact values can be assigned to the parameters. The parameters describe the size as well as the location and orientation of the objects. In the process of designing a rule, the assigned values can be modified until the intended rule is achieved. The final parameter values defined remain static in the later application of the rule.

Every primitive geometric object that is created has its own local coordinate system. This coordinate system and, therefore, the attached object itself, can be translated and rotated in relation to the global coordinate system labeled 'x, y, z' in Figure 4-2 (left).

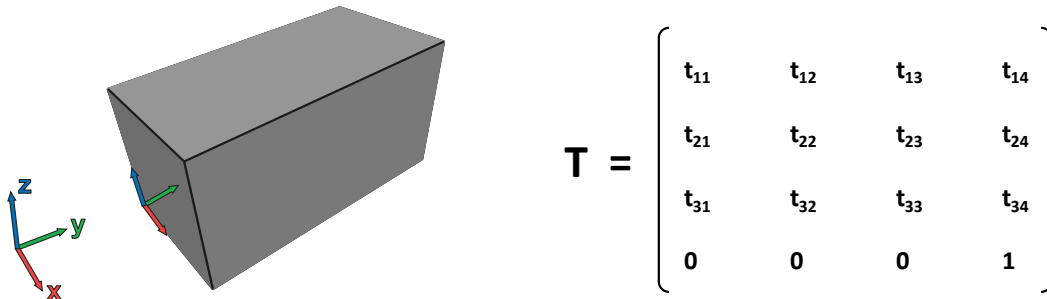


Figure 4-2: Global coordinate system and object with local coordinate system (left) and general transformation matrix (right)

Through this coordinate system, the location and orientation of the object is defined or changed in three-dimensional space. The information about the position is kept in a transformation matrix. As is common in 3D applications, this is a 4x4 matrix containing homogeneous coordinates (Figure 4-2, right). It enables the calculation of different kinds of transformations in one single matrix. Several transformations can be easily concatenated by multiplying the corresponding matrices. This is especially useful for the application of spatial grammar rules, as they require the replacement of the transformed LHS of a rule,  $t(A)$ , by the transformed RHS,  $t(B)$ .

For simplicity in defining the spatial relations of objects and easy access to the transformation information, a rule is designed such that there is one reference object,  $L_0$ , in the LHS. The



reference object is located in the global origin and must not be rotated. It is the basis for the detection of the spatially related objects in the rule's LHS. The transformation matrix of the reference object,  $T_{L_0}$ , always equals the identity matrix  $I$ . Any further geometric objects in the LHS are then positioned in relation to this reference object and, therefore, in relation to the global origin. Thus, the relative spatial relations of the single objects are implicitly defined via the reference object. For an arbitrary number,  $m$ , of objects in the LHS, the transformation matrices of additionally added objects are denoted  $T_{L_j}$  for  $j = 1, \dots, m-1$ .

The objects in the RHS of a rule are also positioned in relation to the global origin and, as a result, are implicitly positioned in relation to the reference object in the LHS. There is no need for a reference object in the RHS, i.e. any object can be arbitrarily located and rotated. The transformation matrices of an arbitrary number,  $n$ , of objects,  $R_i$ , in the RHS of a rule are denoted  $T_{R_i}$  for  $i = 0, \dots, n-1$ . Figure 4-3 shows an example for a rule consisting of several objects including the reference object,  $L_0$ , located at the global origin in the LHS.

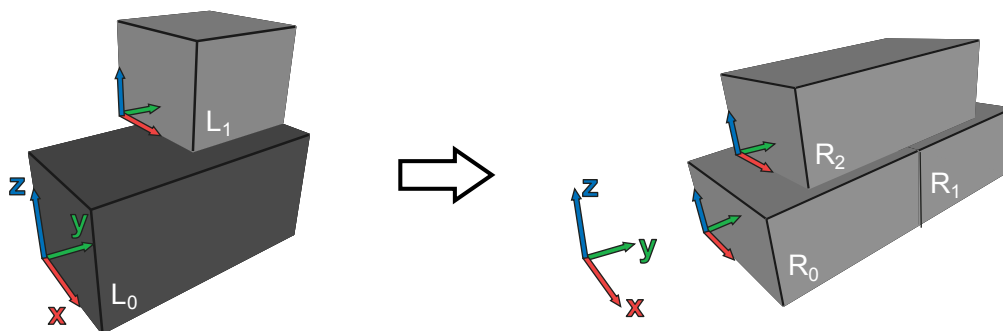


Figure 4-3: Example for a non-parametric rule

Choosing the objects to be inserted in the LHS and RHS, defining the values for their size parameters and their transformations and making sure that one of the objects in the LHS fulfills the reference object preconditions is all that is needed to visually develop an arbitrary range of non-parametric grammar rules.

#### 4.1.2 Application of rules and LHS matching

As shown in section 2.3, the application of grammar rules can be subdivided into the three steps of (1) the determination of a rule to apply, (2) the determination of an object to which the rule is applied and (3) the determination of a matching condition. The latter two steps concern the matching of the LHS of a rule in the CWS. Once a match is found, a further step is needed. According to the grammar formalism, the match has to be subtracted from the CWS and then be replaced by the RHS of the rule under the matching transformation.

In the approach presented here, the selection of rules to apply, as well as the number of rule applications, is assumed to be done either manually by the user or randomly by the system. The remaining steps are supported automatically. The semi-automatic application of rules in this approach is restricted to the scenario of “manually selecting a rule and automatically detecting all objects to which it can be applied”. The alternative scenario of “manually

selecting an object and automatically finding all rules that can be applied to it” is not included here.

### Matching – detection of the LHS of a rule in the CWS

Once a rule is selected for application, according to the equation  $C' = C - t(A) + t(B)$  (cf. formalism in section 2.2), the first step is to detect the LHS of the rule under a certain transformation,  $t(A)$ , in the CWS. Generally, the automatic matching of the LHS of a rule in a current design is a difficult problem, especially in three-dimensional systems. Existing 3D grammar systems require, for example, manual matching by the user (e.g. PIAZZALUNGA & FITZHORN 1998) or they circumvent the problem due to the nature of the provided rules and the way the transformation for every single shape is calculated (e.g. WANG & DUARTE 2002). The aim of the approach presented in this thesis is to automatically match the LHS of a rule in the CWS. As the approach is based on basic primitives, it can benefit from the fact that for any primitive it is explicitly known which class it is derived from, i.e. the ‘type of primitive’, for example, box, cylinder, etc.

Several conditions have to be fulfilled so that a LHS with an arbitrary number of objects can be detected in a CWS. The procedure consists of four main steps, which are illustrated in Figure 4-4 and explained below.

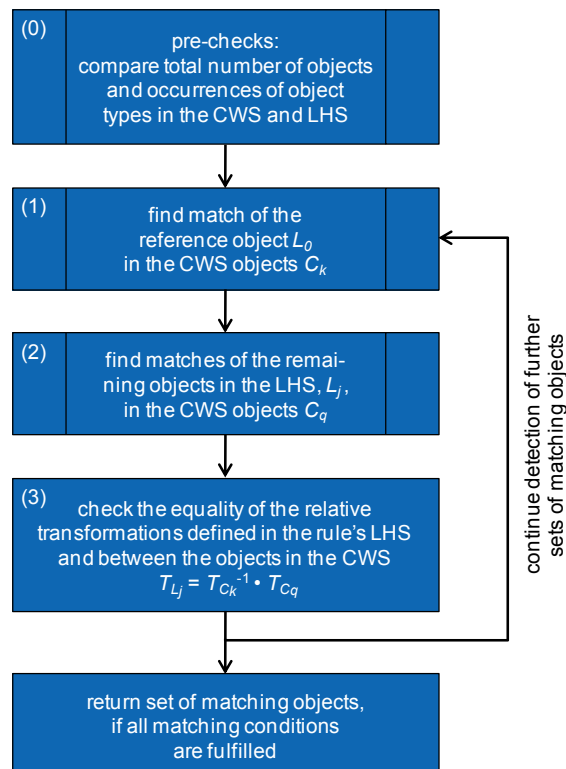


Figure 4-4: General steps for matching a LHS with an arbitrary number of objects in a CWS

Before the detailed detection is performed, a rough pre-check (step 0) is useful to examine whether matching is possible, in general, illustrating an advantage of a set grammar approach.

First, the geometric objects in the LHS of the rule are counted and the result is compared to the total number of objects in the CWS. If the latter is smaller than the number of objects in the LHS, a match is not possible. The second part of the pre-check treats the LHS objects as isolated, single objects. It determines the primitive type of every object in the LHS and checks whether an object with the same type exists in the CWS. If there are more objects of a primitive type in the LHS than in the CWS, matching is also not possible. For example, if there are two boxes and one cylinder in the CWS but three boxes in the LHS, the rule cannot be matched. These simple pre-checks can help save considerable computational time.

The approach for the detailed matching finds all possible matches of the LHS of a rule in the CWS. Beforehand, one reference object,  $L_0$ , in the LHS is identified, as it is theoretically possible that there is more than one object located in the global origin without any rotation. (step 1) The first step is to find matches of  $L_0$  within all the CWS objects,  $C_k$ , for  $k = 0, \dots, p-1$ , where  $p$  is the total number of objects in the CWS. This consists of a comparison of the objects' primitive types with the primitive type of  $L_0$  and an equality check of the size parameters.

(step 2) For every match of the reference object,  $L_0$ , the remaining objects in the LHS,  $L_j$  for  $j = 1, \dots, m-1$ , are checked for matching in the CWS. Every  $L_j$  is therefore compared to the CWS objects,  $C_q$ , starting with  $q = 0$  and incrementing  $q$  until  $q = p-1$ , excluding the case where  $C_q$  is represented by the same object as the considered match of the reference object. The procedure is the same as in (step 1), except that  $L_j$  is checked instead of  $L_0$ .

(step 3) If a certain object,  $L_j$ , matches a CWS object,  $C_q$ , the spatial relation between  $L_j$  and  $L_0$  has to additionally be equal to the spatial relation between  $C_q$  and the considered  $C_k$ . This is checked by comparing the relative transformation matrices of the two pairs of objects. The condition that has to be fulfilled is:  $T_{L_j} = T_{C_k}^{-1} \cdot T_{C_q}$ .

If all matching conditions are fulfilled for all objects in the LHS of a rule, the set of matched objects is returned. Even if a complete set of matched objects in accordance to a reference object match was found, still the remaining CWS objects are checked, because overlapping objects are generally allowed. This means that two identical objects can have the exact same transformation, therefore resulting in two matches with one single reference object match. The outcome of the complete matching procedure is the set of all matches of the LHS in the CWS, whereas every single match itself is a set of objects or at least one object.

An illustrative example of the procedure is given in Figure 4-5. All size parameters of boxes in the LHS as well as in the CWS that are not explicitly shown are 10 mm. The upper part of the figure shows a rule with two boxes in the LHS and a cone in the RHS. The pre-check for matching the LHS in the CWS (lower part of Figure 4-5) is successful, as the number of objects, or boxes, in the LHS of the rule (two) is lower than the number of objects (six) or boxes (five) in the CWS.  $L_0$  is located in the global origin without any rotation and is, therefore, the reference object of the LHS. It is checked for matches to all objects of the CWS. The first object,  $C_0$ , does not match, as it is of a different primitive type (cylinder).  $C_1$ , instead, is of the same type and can be further checked for matching of the parameters. Width, length and height of both the objects equal 10 mm and, therefore,  $C_1$  is the first match of the reference object  $L_0$ . Based on this, the remaining objects of the LHS, in this case only  $L_1$ , have to be checked for possible matches in the CWS.  $C_0$ , again, does not match because of the

different primitive type. Object  $C_I$  is the match for the reference object,  $L_0$ , and must not be checked again.

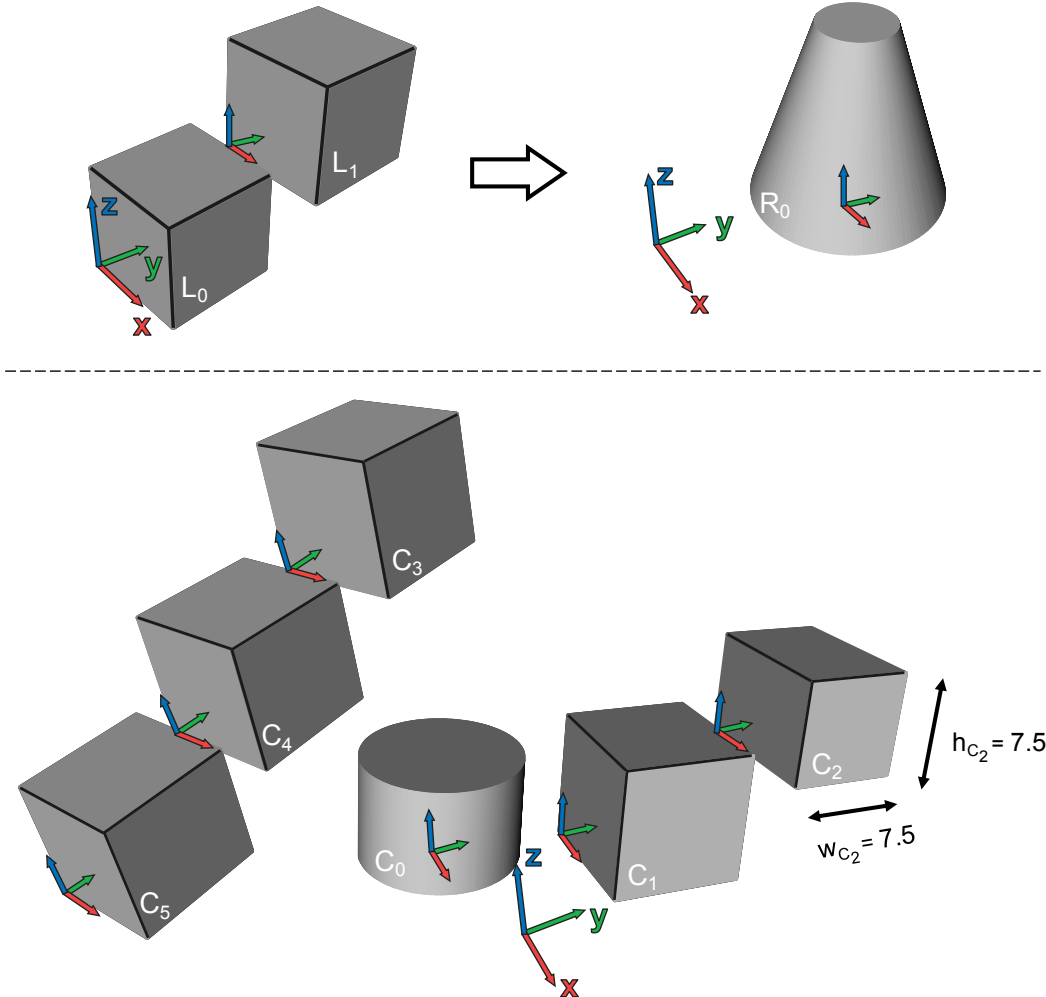


Figure 4-5: Example for the matching procedure: rule above, CWS below

$C_2$  has the same primitive type as  $L_1$ , but, as the width and height both are 7.5 mm, the size parameter check fails. The matching procedure continues identifying the reference object's next match in the CWS.  $C_2$  cannot be matched, again because of the different size parameters. Therefore, the next object in the CWS that fulfills all conditions is  $C_3$ . Candidates for matching the second object in the LHS are the boxes  $C_1$ ,  $C_4$  and  $C_5$ . However, taking a look at the spatial relations between the objects in all possible pairs,  $(C_3, C_1)$ ,  $(C_3, C_4)$  and  $(C_3, C_5)$ , shows that none of the relative transformations are identical to that of  $(L_0, L_1)$  in the LHS of the rule, i.e. the equation  $T_{L_1} = T_{C_3}^{-1} \cdot T_{C_1/4/5}$  is not fulfilled. In the case of the pair  $(C_3, C_4)$ , the absolute values in the relative transformation matrix are the same, but the translation values in the matrix are negative and, therefore, no match is detected. With  $C_4$  as the next match of the reference object, the pair  $(C_4, C_3)$  results in a set of matched objects, as now, in comparison to the pair  $(C_3, C_4)$ , also the relative transformation is the same. The two remaining matching possibilities on the basis of  $C_4$ , the pairs  $(C_4, C_1)$  and  $(C_4, C_5)$ , both fail

the relative transformation check. The matching procedure for the last object,  $C_5$ , is basically the same as for  $C_4$  and results in another set of matched objects  $(C_5, C_4)$ . In summary, the set of all matches of the LHS in the CWS that are given in the example shown in Figure 4-5 consists of  $((C_4, C_3), (C_5, C_4))$ .

### Replacing – calculation of the transformations of the RHS objects

Following the equation  $C' = C - t(A) + t(B)$  (cf. formalism in section 2.2), all of the rules are realized as replacement rules in this approach, i.e. the matched LHS is always fully subtracted from the CWS and replaced by the transformed RHS. While this is not always computationally efficient, for example in case of additive rules, it is the most general way to create a spatial grammar interpreter.

The starting point for the replacement is the set of all possible LHS matches detected. In a parallel grammar approach, the rule would be applied to all these matches simultaneously (GIPS 1975). However, as is the case with most existing grammars, the approach described here is based on a serial application of rules. Therefore, a rule is always applied to only one match out of all found matches. This match can either be determined automatically by a randomized selection or manually chosen by the user.

The RHS objects that will be inserted into the CWS to replace the detected LHS objects are denoted  $C'_i$  for  $i = 0, \dots, n-1$ , where  $n$  is the number of objects in the RHS, as introduced in 4.1.1. The transformation information,  $T_{C_k}$ , of the object that was matched to the LHS reference object is explicitly available, as are the transformation matrices  $T_{R_i}$  of the objects in the RHS of the rule. The new position of every object,  $C'_i$ , is determined in two steps: (1) add the object to the CWS under the transformation  $T_{R_i}$ , as defined in the RHS of the rule and, in order to fulfill the equation in the grammar formalism, (2) apply the transformation under which the LHS was detected ' $t(A)$ ' to the RHS object ' $t(B)$ '. This means that the transformation of  $R_i$  is additionally multiplied with the transformation of the object  $C_k$ . Eventually, for all objects in the RHS, this results in the equation:  $T_{C'_i} = T_{C_k} \cdot T_{R_i}$  for  $i = 0, \dots, n-1$ . After all transformations are calculated, the objects of the selected match can be subtracted from the CWS and replaced by the RHS objects under the calculated transformations.

Coming back to the illustrative example in Figure 4-5, the depicted replacement procedure is as follows: Out of the set of all found matches, for example, the match  $(C_4, C_3)$  is chosen. In this case,  $C_4$  is identified as the match of the LHS reference object. The RHS of the rule consists of only one object,  $R_0$ . It will be the new object  $C'_0$  in the CWS after the replacement and its position is calculated according to the equation:  $T_{C'_0} = T_{C_4} \cdot T_{R_0}$ . Lastly,  $C_4$  and  $C_3$  are subtracted, resulting in the new CWS shown in Figure 4-6.

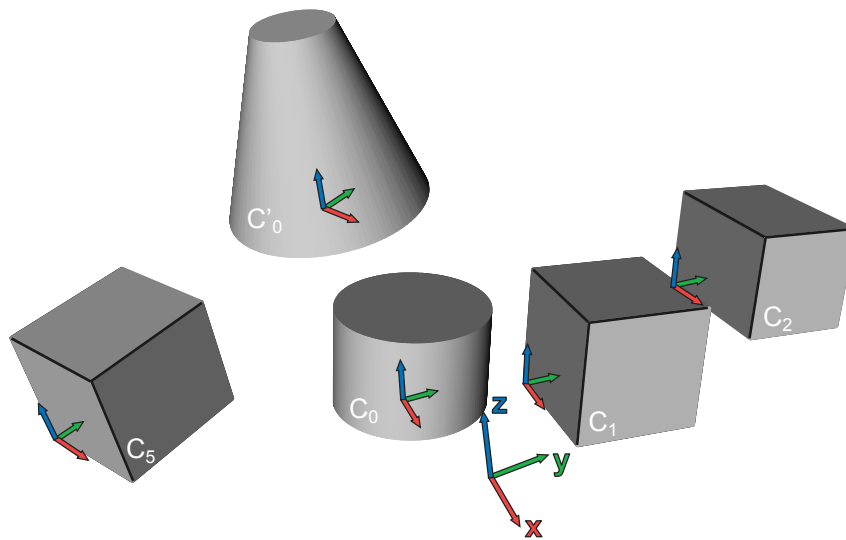


Figure 4-6: New CWS after application of the rule in Figure 4-5 to the object pair ( $C_4$ ,  $C_3$ )

## 4.2 Parametric rules

In the previous section, the used parameterized primitives are assumed to be fully defined, i.e. specific values are assigned to all parameters. The outcome is rules that are based on fixed, fully determined objects. In this section, the approach is extended to the use of parametric rules (cf. 2.2.3) to enable the generation of a wider variety of design solutions by fewer rules and incorporate constraints into rules. On paper, defining parametric rules is straight forward. Parametric relations and constraints are often implied using additional descriptive text (e.g. STINY 1977). However, a general computational implementation cannot be as easily created. One of the crucial points is to automatically match a general parameterized shape in the LHS of a rule to an existing design. Partially adapted from the original definition by STINY (1977, 1980a) to fit the approach based on parameterized primitives, in the following aspects for the development and application of parametric rules are elaborated on.

### 4.2.1 Development of parametric rules

The set grammar formulation of spatial grammars chosen in this approach, which is based on the use of parameterized primitives, enables the definition of parametric rules, as the parameters are explicitly defined for each object. Parametric rules require all the basics needed for the development of non-parametric rules (cf. 4.1.1). Even though the intention is to define a parametric rule, initially the geometric objects have to be fully determined, i.e. specific values have to be assigned to all parameters. This is because a visual grammar system, in comparison to a grammar on paper or a hard-coded grammar, would not be able to create and display objects that initially have one or more unspecified parameters. Once the initial, non-parametric state of an object is designed, one or more of its parameters can be

‘unlocked’, denoted as ‘free parameters’, to make it parametric. Parameters that can be unlocked are not only the ones that define geometric dimensions, called ‘size’ parameters, like width, length, radius, etc., but also those that determine the position of an object in 3D space, called ‘transformation’ or, more specifically, ‘location’ and ‘orientation’ parameters. Location parameters determine the translation of an object in the x-, y- and z-directions, denoted ‘translateX, translateY and translateZ’; orientation parameters determine the rotation defined as ‘yaw, pitch and roll’.

If a parameter is unlocked, by default it is completely unrestricted, i.e. any real value can be assigned to it. However, there are two ways to constrain free parameters:

(1) The values that are allowed to be assigned to a free parameter can be restricted to a certain range. An example rule with restricted ranges of possible values for the width  $w$  of a box and its rotation angle  $roll$  around the x-axis is shown in Figure 4-7.

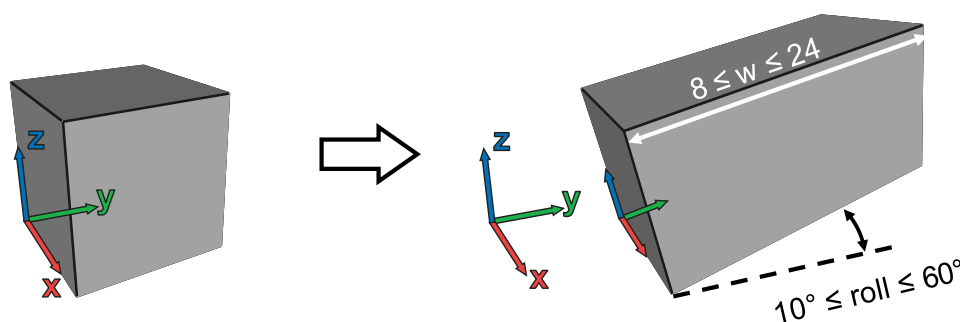


Figure 4-7: Parametric rule with free parameters that are restricted to certain ranges

(2) Parametric relations can be defined between free parameters. To make a free parameter dependent on one or more other free parameters, mathematical equations can be defined. These equations consist of different operators and mathematical functions used with either free parameters as operands and/or numeric operands. For example, in Figure 4-8 the radius  $r_{R_0}$  of the cylinder is dependent on the height of the box using the equation  $r_{R_0} = \frac{1}{4} * h_{R_1}^2 - \pi$ , whereas  $h_{R_1}$  is restricted to a certain range.

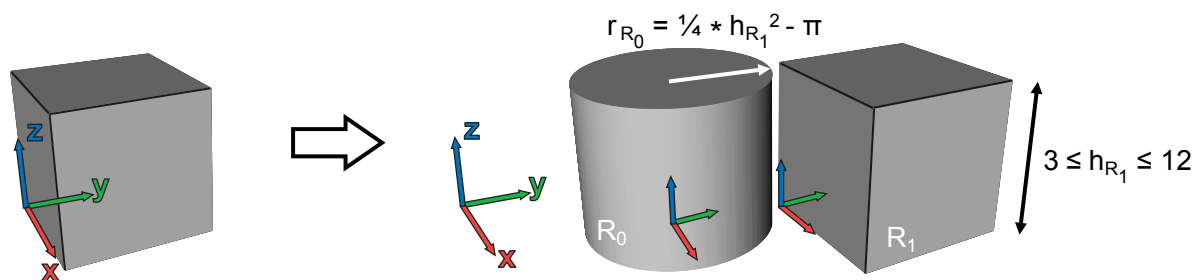


Figure 4-8: Rule with parametric relation between different objects in the RHS

This is similar to the approach taken in many commercial mechanical CAD systems for the definition of parametric relations in or between geometric models. In a CAD system, this is primarily used to lower the effort needed for the modification of a model. Changing the value of just one parameter subsequently triggers the change of one or more other parameters in the same model. The concept for grammar rules used here is basically the same; however, the main purpose is not the easier adaptation or modification of the geometry, but the definition of size or spatial relations within one or between different objects.

To incorporate certain constraints based on range restrictions, it is often not sufficient to only define real values for the limits. Therefore, it is also possible to use equations containing parametric relations to define the minimum or maximum limit. Additionally, if a restriction in only one of the directions is needed, either minimum or maximum can be left blank. In Figure 4-9 for example, the height of the cylinder in the RHS has to be at least twice as big as the radius, which is left unrestricted, but there is no maximum limit.

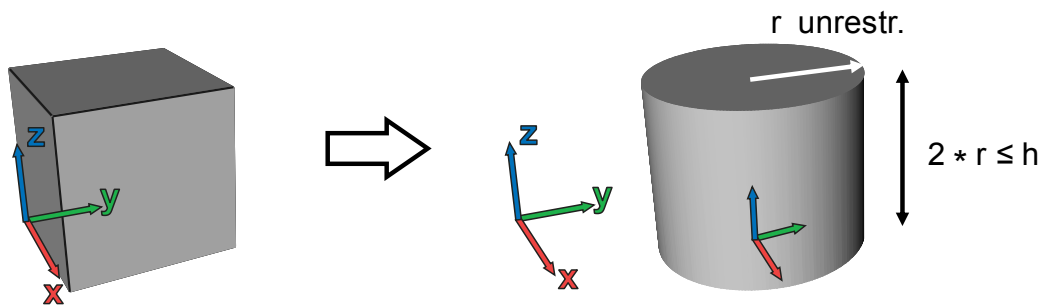


Figure 4-9: 'Range parameter' with only one limit which is defined using a parametric relation

The examples given so far have considered only objects in the RHS of a rule, but the described specification of free parameters can also be used for geometric objects in the LHS. LHS parameters can only be dependent on other parameters in the LHS. Parameters in the RHS, instead, can be dependent on parameters in both the RHS and the LHS. Figure 4-10 illustrates an example in which the length of the box in the LHS is defined as being completely unrestricted. The width and the height are both dependent on the length. Additionally, a parametric relation between the height of the first box in the RHS,  $h_{R0}$ , and the height of the second box,  $h_{R1}$ , is defined, whereas  $h_{R1}$  itself is dependent on the width of the box in the LHS.



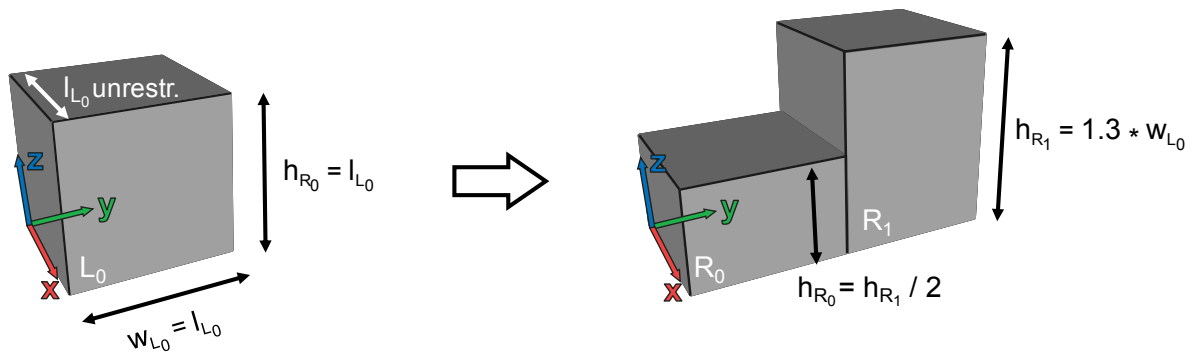


Figure 4-10: Rule with parametric relations in the LHS and RHS

Free parameters can be defined by the user as needed. However, in a computational system it has to be ensured that they, especially the parametric relations, are evaluated in a distinct sequence, so that all equations in a rule are resolvable during the application, independent of the sequence in which they are defined. For example, in the rule shown in Figure 4-10 the free parameter  $h_{R1}$  inevitably has to be evaluated first in the RHS, so that it is possible to resolve the equation for  $h_{R0}$ . To determine the correct order of the parameters for their evaluation, a topological sort algorithm can be used after transforming the parametric relations in a rule into a dependency graph. If this step is already carried out in the development of a rule, it only has to be done once and not each time the rule is applied. It is further possible to use the same algorithm for the detection of cyclic relations between free parameters, e.g. *width = length*, *length = height* and *height = width*. Dependencies like this cannot be resolved during the application of a rule. Therefore, the detection of cyclic relations allows for avoiding their definition.

### 4.2.2 Application of parametric rules

Applying parametric rules, the impact of the free parameters is slightly different depending on the side of the rule in which they are defined. In the LHS a parametric object allows for matching a wider range of objects, so that the same rule can be applied in more cases. In the RHS, a parametric object provides for the generation of a wider variety of differently sized and transformed objects.

For example, in the parametric rule in Figure 4-11(a) the length of the box in the LHS is restricted to a range with lower limit  $a$  and upper limit  $b$ . The effect is that it can not only match boxes with the (designed) length  $l$ , but also boxes with length  $a$ , length  $b$  or any other length in between the two limits (Figure 4-11(b)), providing, of course, that the non-parametric width and height match in each case. Once a match is found, the actual value of the length of the detected box is assigned to the height of the cylinder in the RHS because of the parametric relation  $h_{R0} = l_{L0}$ . The radius of the cylinder, instead, is not dependent on any other parameter but completely unrestricted, so that a virtually unrestricted variety of cylinders can be generated applying this rule (Figure 4-11(c)).

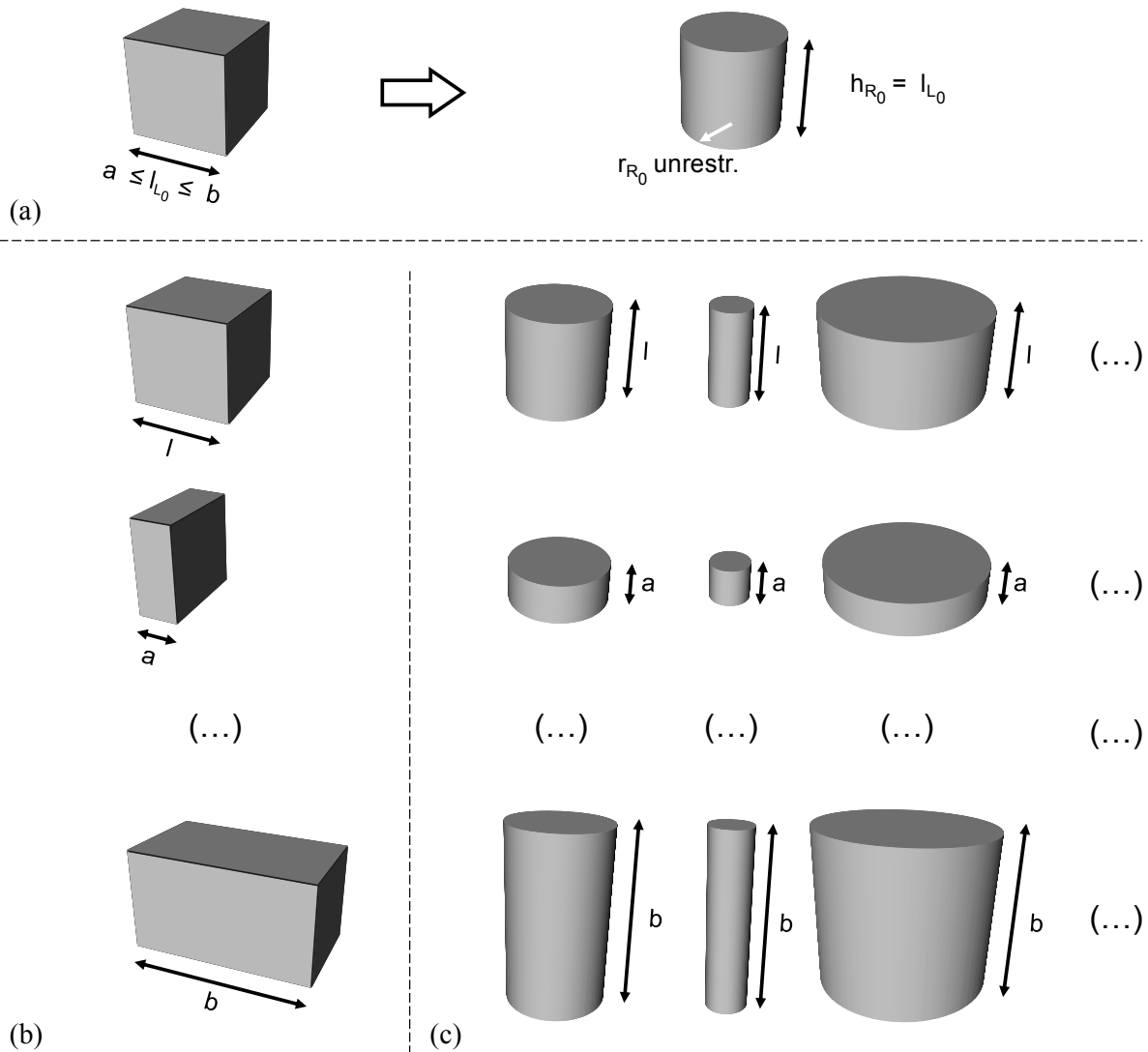


Figure 4-11: Impact of free parameters in the LHS and RHS applying a rule

A wider variety of LHS matches can be detected if a parameter is not only restricted to a range but defined as completely unrestricted. Obtaining an even higher degree of matching generality is feasible if more than one parameter is ‘unlocked’. For the special case in which all size parameters of an object in the LHS are ‘unlocked’ and completely unrestricted, all objects of the same primitive class can be found, e.g. all boxes, no matter what length, width or height. This is the most general definition of a parametric rule’s LHS.

Further, the use of parametric relations in the LHS enables the matching of objects under a certain ratio of dependent size parameters. For example, a parametric relation between length and width of a box can be defined as  $l = 3 * w$ , so that the matching is restricted to objects with a ratio of three between these two parameters. This concept can also be used for the detection of scaled versions of an object. The LHS of the rule in Figure 4-10 represents an example for this: the length of the box,  $l_{L_0}$ , is completely unrestricted but its width and height

are restricted by the parametric relations  $w_{L_0} = l_{L_0}$  and  $w_{L_0} = l_{L_0}$ . The matching process therefore detects all cubes, no matter which size, but no other boxes.

The procedure for matching parametric rules is basically the same as in the non-parametric case (Figure 4-12). First, the pre-checks are performed and the reference object in the LHS is identified as described in Section 4.1.2. The first step (step 1), which is about finding matches of the reference object in the CWS, is performed identically, except that any defined free parameters are additionally checked. The same applies to the second step (step 2) that checks the remaining objects in the LHS for matches in the CWS. As explained for the non-parametric case, one object after another is checked according to the list of objects.

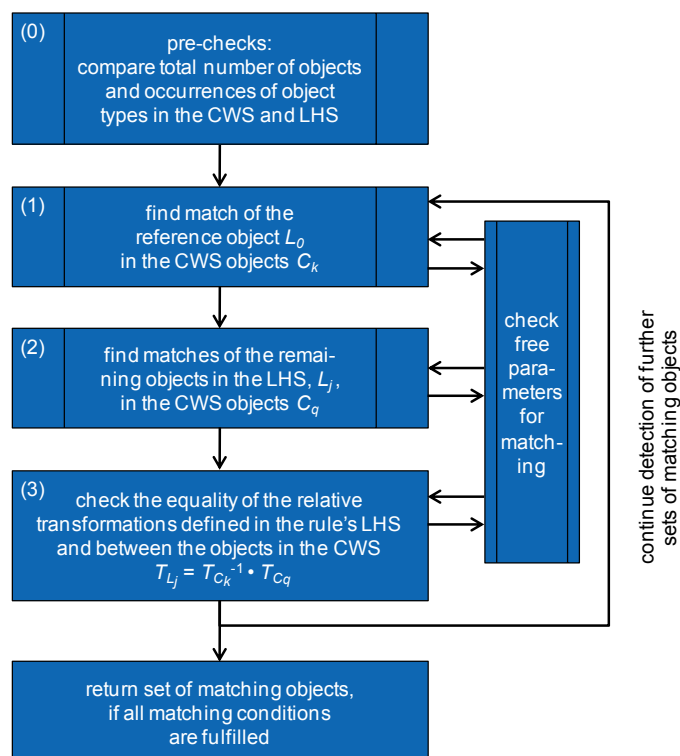


Figure 4-12: LHS matching process including free parameter evaluation

Due to the sequence of the list of objects, however, the following issue can occur: If there is more than one object defined in the LHS, by the time a free ‘equation’ parameter is supposed to be evaluated, it is possible that the actual value of one or more of the parameters on which it is dependent has not yet been evaluated. For example, by the time the matching process reaches the box  $L_1$  in the LHS of the rule in Figure 4-13(a) to check it for matching the CWS (Figure 4-13(b)), its free parameters  $h_{L_1}$  and  $w_{L_1}$  cannot be evaluated as they are directly or indirectly dependent on the free height-parameter of the cylinder  $L_2$ .

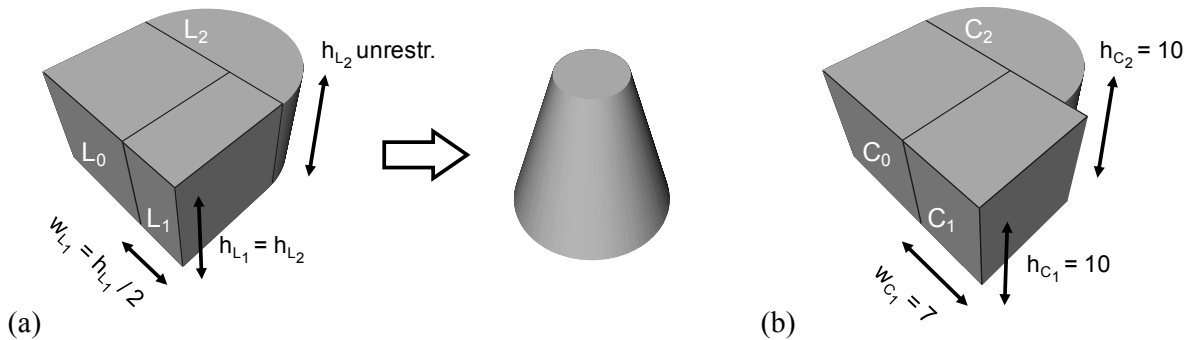


Figure 4-13: Rule (a) with the need to postpone the evaluation of certain parameters ( $h_{L_1}$  and  $w_{L_1}$ ) and a CWS example (b) to match it to

In such cases, where parameters cannot be evaluated right away, the evaluation of the affected free parameters has to be postponed and it is assumed that the object is a match. Once the last object in the LHS is checked for matching, all postponed free parameters can be evaluated according to the parameter sequence that was determined during the development of the rule (see 4.2.1). If only one of them does not match, there is no need to go on evaluating the remaining parameters, as one single parameter-mismatch is sufficient to indicate that the complete LHS does not match. In Figure 4-13, for example, it turns out that the postponed parameter  $w_{C_1}$  does not fulfill the condition defined in the rule and, therefore, the temporary assumption of having a match is retracted.

Postponing the evaluation of parameters is applicable to matching the reference object as well. Further, it also applies to any free translation or rotation parameters. The evaluation of translation or rotation parameters starts off with the calculation of the relative transformation as explained for non-parametric rules. However, to be able to perform the check for free transformation parameters, the calculated relative transformation matrix is converted into the representation using ‘translateX, translateY, translateZ’ and ‘yaw, pitch, roll’. This is the representation used for the definition of equations during the rule development phase and it makes it possible to evaluate these parameters like any other size-parameters.

Once the set of all possible LHS matches is detected, one is chosen and replaced by the transformed RHS of the rule as described for the non-parametric case in 4.1.2. If the RHS of the rule is also parametric, all free parameters are evaluated, again in the sequence that was determined during the development of the rule (see 4.2.1). This includes the assignment of concrete values to parameters that are either completely unrestricted or restricted by a range. They can be chosen either manually by the user or randomly in an automatic mode. Values that are assigned to completely unrestricted parameters or to ‘range’ parameters with only one limit are additionally checked for the fulfillment of implicit restrictions to generate valid geometry, e.g. the values for length, width and height have to be positive.

The rule shown in Figure 4-14 revisits the illustrative example given in Figure 4-5, but this time it is augmented by free parameters to define a parametric rule. As the object  $C_0$  is a cylinder, the first candidate in the CWS to match the reference object  $L_0$  is the box  $C_1$ . Width

and height fulfill the matching conditions, the evaluation of the length, however, is dependent on the length of the second object according to the parametric relation  $l_{L_0} = l_{L_1}$  in the LHS of the rule. As the second object that will be checked and its parameters are not yet known, the evaluation is postponed and it is preliminarily assumed that the object  $C_1$  is a match of the reference object  $L_0$ .

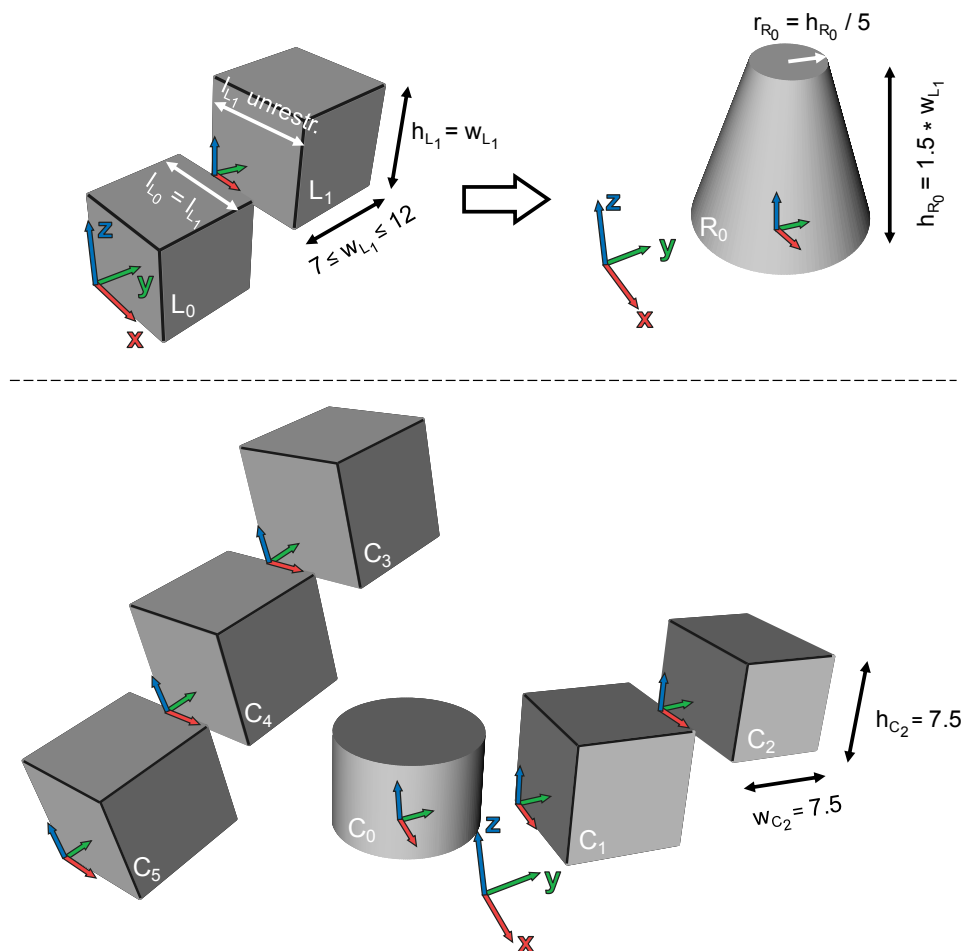


Figure 4-14: Parametric version of the example in Figure 4-5

Now possible matches for the object  $L_1$  are searched for in the CWS.  $C_0$  does not match and  $C_1$  was the match of the reference object, so  $C_2$  is the first object that comes into consideration. The parameter for the length of the object is defined as unrestricted in the LHS of the rule and, therefore, this condition is fulfilled. According to the evaluation sequence determined during the development of the rule, the next parameter to be evaluated is the width  $w_{C_2}$ , which equals 7.5 mm and is, therefore, within the given range of 7 mm to 12 mm. The parametric relation of the last free parameter, the height  $h_{C_2}$ , is also fulfilled as it equals the width of the object. After the LHS is completely checked, the postponed parameter, i.e. the length of the first box, can be evaluated. As required, its value equals the length of the second object and, therefore, the assumption of having a match is approved. The last step is to compare the relative transformation between the object pairs  $(L_0, L_1)$  and  $(C_1, C_2)$ , which is

also successful. As in the non-parametric version of the example, the further matching procedure results in matches for the pairs  $(C_4, C_3)$  and  $(C_5, C_4)$ . The set of all matches of the LHS in the CWS, therefore, consists of  $((C_1, C_2), (C_4, C_3), (C_5, C_4))$ .

Choosing, for example, the first match  $(C_1, C_2)$ , results in the replacement of the two boxes by the cone in the RHS of the rule. The transformation is not dependent on any free parameters and is calculated as in the case of a non-parametric rule. The calculation of the free parameters in the RHS is again done in the sequence that was determined during the development of the rule. That means that, first, the new height is determined. It is dependent on the value that was detected for the width of the second box,  $w_{C_2}$ , and results in 11.25 mm according to the given equation. The new upper radius of the cone is in turn dependent on the height. The result of the evaluation of the according parametric relation is 2.25 mm.

### 4.3 Objects based on Boolean operations and swept objects

The spatial grammar approach described so far allows for a wide variety of different geometries to be generated. This is achieved by not restricting the number of geometric objects that can be defined on either side of a rule and allowing parameterized primitives that can be used with different parameter configurations and in conjunction with parametric relations. Nevertheless, there are restrictions that prevent the generation of more complex solid models, which are often needed in mechanical engineering. What if, for example, a round hole in a block is required? This cannot be accomplished with the approach described so far because it is based on a primitive instancing modeling approach, i.e. the objects in the LHS and RHS of a rule are exclusively parametric primitives. In the following subsections the approach is, therefore, enhanced to include further solid modeling functions, namely Boolean operations and sweeping.

#### 4.3.1 Boolean operations

Combining the parameterized primitives used in this approach with each other helps to considerably increase the amount of different shapes that can be modeled in rules and thus the complexity of shapes that can be generated. Solid modeling systems usually support the combination of geometric objects based on Boolean set operations; the resulting objects will be denoted as ‘Boolean objects’ in the following. The common Boolean operations are union ( $\cup$ ), intersection ( $\cap$ ) and difference ( $-$ ). Figure 4-15 illustrates simple examples for each of these operations combining the given cylinder (object  $X$ ) and box (object  $Y$ ). When performing a union operation it is not relevant whether object  $X$  is added to  $Y$  ( $X \cup Y$ ) or the other way around ( $Y \cup X$ ), the resulting geometry is identical. The same applies to intersection operations, where the intersecting volume of the two solids defines the new object. In the case of using a difference operation, however, the geometric result is dependent on what object is subtracted from the other one, i.e.  $X - Y$  or  $Y - X$ .

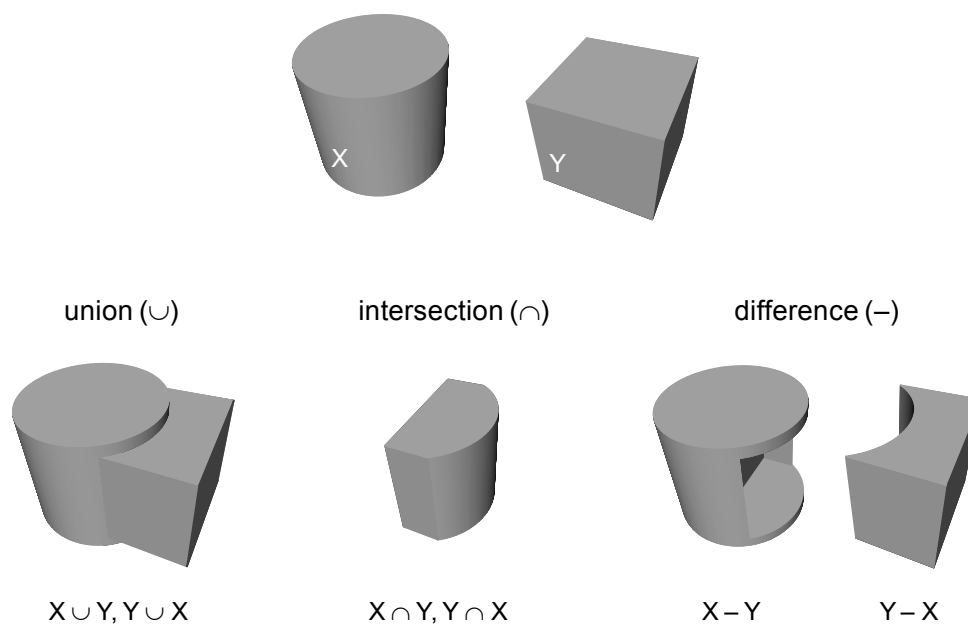


Figure 4-15: Different Boolean operations performed on the primitives  $X$  and  $Y$

The use of the three types of Boolean operations shown in Figure 4-15 is not restricted to combining primitives only. Any object can be based on other Boolean objects to an arbitrary depth. This allows for the definition of very complex objects. The correlations between the single objects are kept in a tree structure, which is commonly known as the Constructive Solid Geometry (CSG) representation. An example of an object based on the combination of several other Boolean objects is shown in the tree structure in Figure 4-16. Three cylinders with different spatial relations to each other are combined to one object using two union operations. The resulting object is then subtracted from a box using a difference operation to produce the intended design.

All primitives a Boolean object is based on exist within the solid model and are still accessible after the Boolean operations are performed but only the resulting object is visible. Therefore, the underlying parameterized primitives, i.e. their sizes and transformations, can be modified whenever needed and the resulting Boolean object is updated accordingly. Because of that precondition, the basics and methods for the spatial grammar approach described so far can be completely transferred to the usage of Boolean objects. Consequently, rules including these kinds of objects can also be parametric.

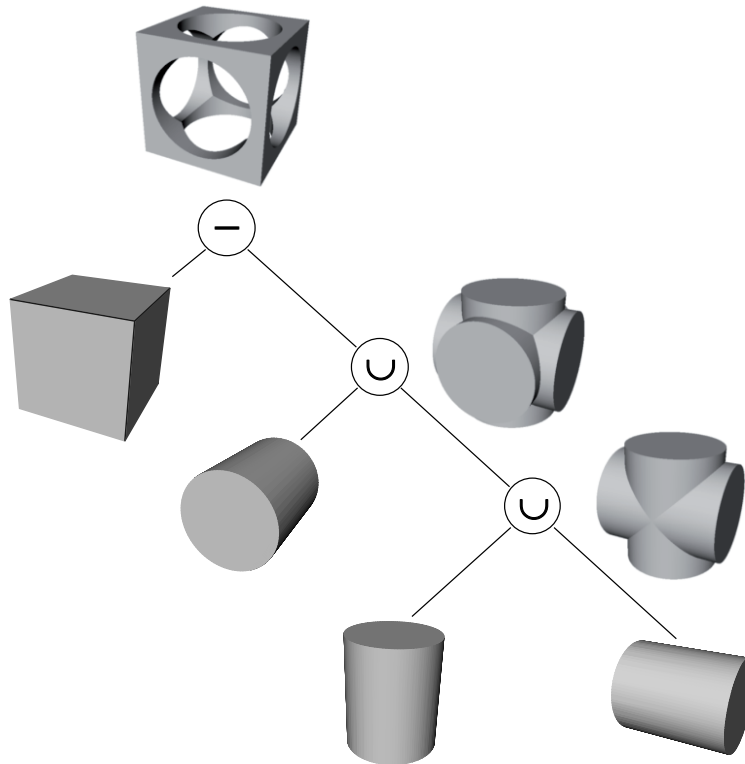


Figure 4-16: Example CSG tree for an object created using several Boolean operations

The definition of grammar rules is carried out directly on the underlying primitives. In this context, the Boolean objects can be seen as a ‘hull’ whose shape is defined only indirectly but represents the final geometric object. In a rule, the reference object cannot directly be represented by a Boolean object. Instead it has to be ensured that one of the underlying primitives is positioned in the global origin to define the reference object, unless there are other ‘conventional’ primitives in the LHS that fulfill the condition of being a reference object. Apart from that, the development of rules follows exactly the approach described for non-parametric (cf. 4.1.1) and parametric rules (cf. 4.2.1).

The overall process for matching the LHS in the CWS generally stays the same as described in Chapter 4.2.2 for parametric rules. The only difference is that in addition to the object types for the geometric primitives (‘box’, ‘cylinder’, etc.), objects of the types ‘union’, ‘intersection’ and ‘difference’ can occur. Once the process reaches an object in the LHS that has one of these types, it tries to find an object of the same type in the CWS. If this search is successful, a sub-process for matching the underlying parameterized primitives is initiated (Figure 4-17). For Boolean objects that are not only based on two primitives but on further Boolean objects, i.e. using a CSG tree like the example given in Figure 4-16, this sub-processes is performed in a recursive manner. This means that if one of the underlying objects is a Boolean object itself, it is resolved using the same procedure until all Boolean objects are compared based on their underlying parameterized primitives for matching in the tree.



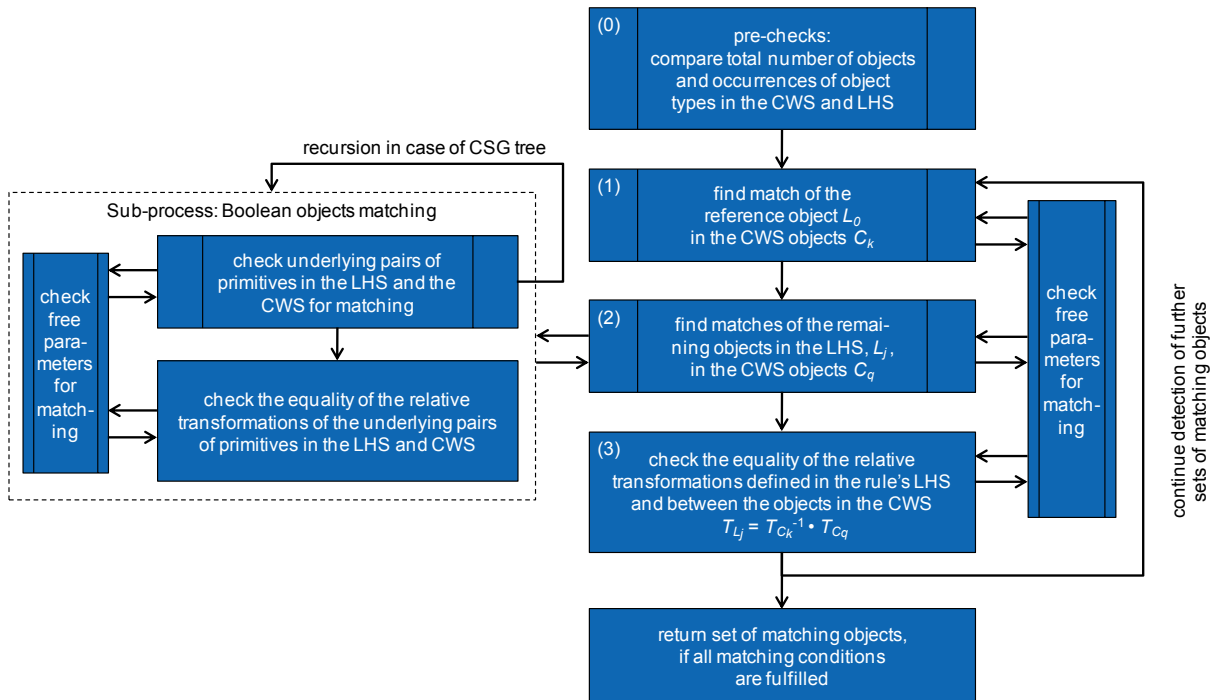


Figure 4-17: LHS matching process including a sub-process for matching Boolean objects

A simple example kept on the level of one single Boolean operation, i.e. only two geometric primitives and no further Boolean objects are combined, is shown in Figure 4-18. This example is used to illustrate the steps needed for this sub-process. The LHS of the rule consists of an object,  $L_0$ , created using a Boolean union operation that combines the box  $L_{0a}$  with the sphere  $L_{0b}$  (Figure 4-18(a)). The CWS (Figure 4-18(b)) is nearly identical to the LHS of the rule except that the order of the primitives that the object  $C_0$  is based on, namely the sphere  $C_{0a}$  and the box  $C_{0b}$ , is the other way around.

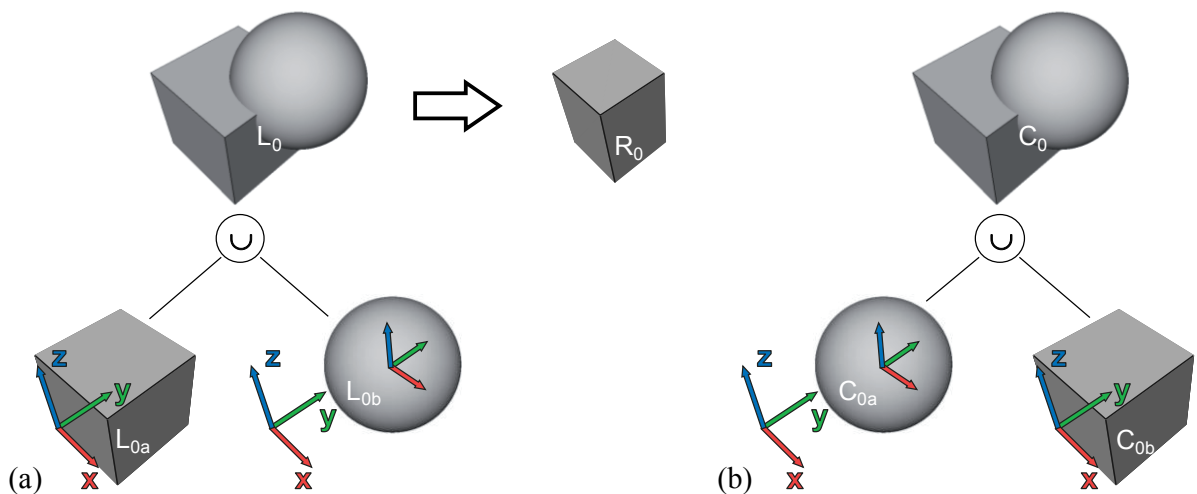


Figure 4-18: Example for a rule (a) and CWS (b) containing an object created using a Boolean union operation

Evidently the two Boolean objects  $L_0$  and  $C_0$  are the same. That means that the procedure has to be able to detect a match even if the order of the underlying primitives differs. For objects based on union operations like the one shown in the above example, as well as for objects based on intersection operations, the pairs with the order of the objects  $(L_{0a}, L_{0b}) \Leftrightarrow (C_{0a}, C_{0b})$  and the pairs with the inverted order of objects for the LHS  $(L_{0b}, L_{0a}) \Leftrightarrow (C_{0a}, C_{0b})$  are checked for matching. The comparison is done using the same methods as described for ‘conventional’ matching: First the type of the objects is checked, followed by the size parameters and the relative transformation between the objects. If needed, it also includes the check of free parameters. In the first case above, the comparison of the object types of  $L_{0a}$  (box) and  $C_{0a}$  (sphere) fails and therefore it cannot result in a match. Even if the two spheres,  $L_{0b}$  and  $C_{0a}$ , were boxes of the same exact sizes as  $L_{0a}$  and  $C_{0b}$ , no match would be detected because the equality check of the relative transformation matrices would result in the negative version of the translations for the pair  $(C_{0a}, C_{0b})$  in comparison to the pair  $(L_{0a}, L_{0b})$ . In the second case, the types and measurements of the compared objects  $L_{0b}/C_{0a}$  and  $L_{0a}/C_{0b}$  as well as the relative transformations of the object pairs  $(L_{0b}, L_{0a})$  and  $(C_{0a}, C_{0b})$  are identical. Due to this successful check, the superior Boolean objects  $L_0$  and  $C_0$  are identified as matches.

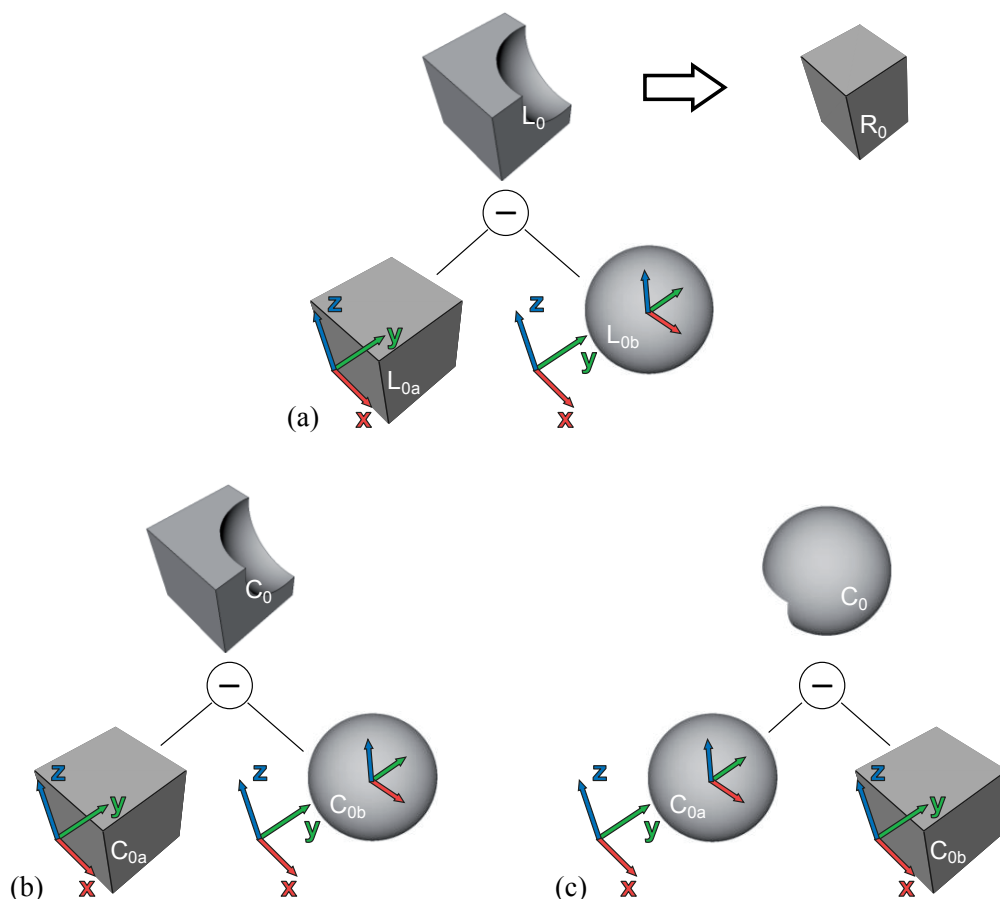


Figure 4-19: Example for a rule (a) and two different CWSs (b) and (c) containing an object created using a Boolean difference operation each

In contrast to objects based on Boolean union or intersection operations, the order of the underlying primitives of Boolean difference objects plays a crucial role. Therefore only one distinct check corresponding to the exact order of the underlying primitives in the LHS Boolean object is allowed to be performed. Trying to match the LHS of the rule in Figure 4-19(a) with each of the two CWS examples in (b) and (c), the required check compares the pairs with the order of the objects ( $L_{0a}, L_{0b}$ )  $\Leftrightarrow$  ( $C_{0a}, C_{0b}$ ). For the CWS in (b) this results in a match, whereas for (c) the check fails because the object  $L_{0a}$  and  $C_{0a}$  are of different primitive types. That is exactly the intended behavior of the check because the geometry of the Boolean object in (c) clearly differs from that in (a) and, therefore, must not result in a match.

The last step in the application of a rule, the replacement of the matched objects in the CWS by the objects in the RHS of the rule, including the calculation of the related measurements and transformations, is identical to that explained in 4.1.2 and 4.2.2.

A concluding example for the use of Boolean operations in a parametric rule is given in Figure 4-20. It is based on the ‘die’ shown in Figure 4-16. The underlying primitives are not explicitly shown in the figure, but their free parameters are denoted with index  $a$  for the box primitive and  $b$  to  $d$  for the three cylinder primitives. For reasons of clarity, the parametric relations for all transformation parameters as well as the ones of the object  $R_0$ , which are defined as being equal to the corresponding size parameters of the dice in the LHS, are not shown.

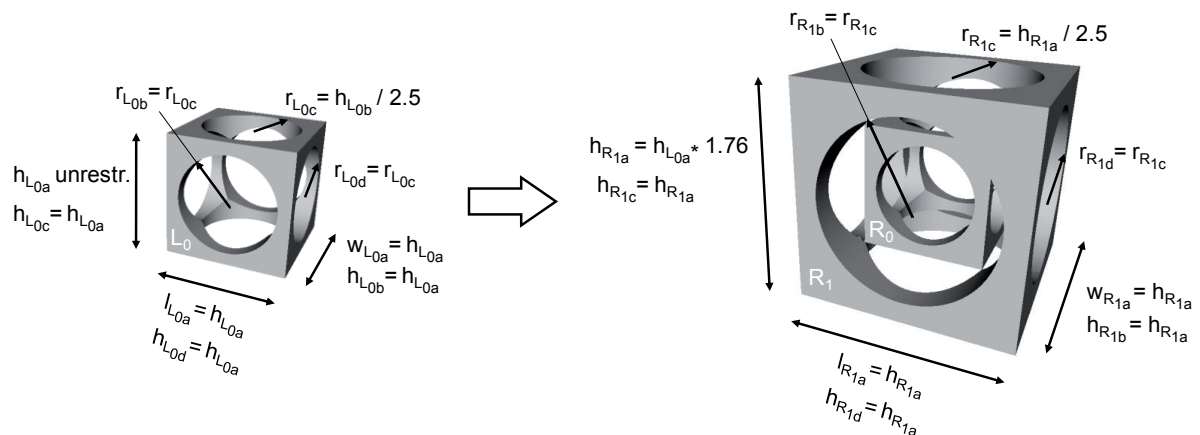


Figure 4-20: Example for a parametric rule containing Boolean objects

The rule finds all scaled versions of the die shown in the LHS and adds a second die to an existing one with an edge length that is 1.76 times larger than that of the matched Boolean object. Applying the rule four times to the last previously added die, starting with the die shown in Figure 4-16 as the initial set, results in the geometric model shown in Figure 4-21.

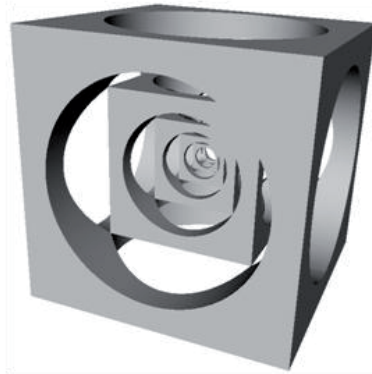


Figure 4-21: Example solution applying the die rule four times

### 4.3.2 Sweeping

Using Boolean operations for the combination of parameterized primitives as described allows for a wide range of different shapes that can be modeled in rules. The predominantly used three-dimensional modeling technique in today's mechanical engineering CAD systems, however, is to design geometric models based on sweeping operations. This technique often allows for easier modification of existing models and enables the design of geometric objects that cannot be created using Boolean operations or would require very high modeling effort. Therefore, aspects of sweeping are also incorporated into the spatial grammar approach.

The basic idea of sweeping is to move a planar, two-dimensional cross-section along a guiding line to describe a volume in 3D space. In general, the guiding line can be linear ('extrusion'), a circle or axis ('revolve'), or an arbitrary three-dimensional curve. In the approach in this work, only the first two possibilities are considered, as the general automatic matching of a rule's LHS containing an arbitrary trajectory is a very difficult issue.

A cross-section in a grammar rule is created by means of defining a number of vertices. The result is a wire-profile made out of straight lines. To be able to derive a valid solid based on sweeping operations in the approach described here, this profile has to be closed and the wire has to be converted into a face. An example is shown in Figure 4-22. To facilitate the LHS matching process, the profile must be created on the xy-plane of the local coordinate system, one of the vertices has to be located at the origin, one of the edges of the profile has to be coincident with the x-axis and the profile must be created in the positive y-direction.

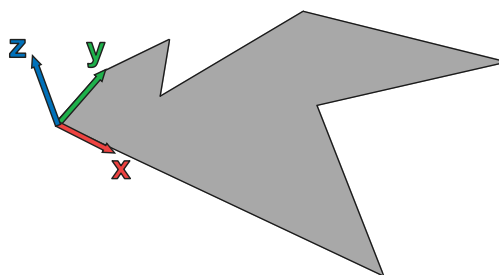


Figure 4-22: Example for a two-dimensional cross-section

An extruded object is created based on a cross-section by defining its extrusion in the x, y or z-direction. That means that a vector is determined in relation to the local coordinate system and the cross-section is expanded along that vector. The related parameters are denoted ‘dirX, dirY and dirZ’. The definition of a revolve object is similar, except that the defined vector represents the axis that the cross-section is rotated around. The related parameters for the latter are denoted ‘axisX, axisY and axisZ’. An example for a grammar rule using different kinds of swept objects based on the cross-section in Figure 4-22 is given in Figure 4-23. The LHS consists of a revolve object, whose rotation-axis is defined by the parameters  $axisX = 1$  and  $axisY = 5$  and, therefore, slightly deviates from the y-axis. Additionally, it is not fully revolved; the angle parameter is set to 230 degrees. The object in the RHS is created using an extrusion along a vector that is defined by the parameters:  $dirX = 3$ ,  $dirY = 3$  and  $dirZ = 2$ .

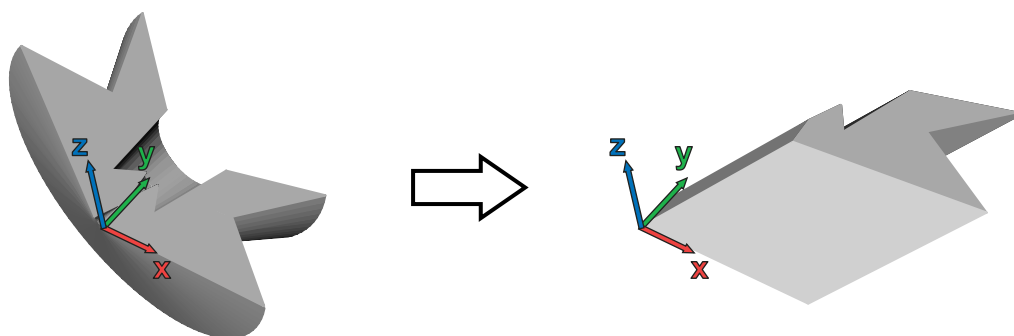


Figure 4-23: Example for a grammar rule using different kinds of swept objects

As is the case with ‘conventional’ primitives, the parameters of swept objects can be used to define parametric grammar rules. The underlying profiles explained above, however, are not parameterized and, therefore, cannot be used for the definition of parametric rules. To circumvent this, cross-sections can alternatively be defined on the basis of a parameterized, two-dimensional plane primitive that possesses the two parameters length and width. These parameters can be used for the definition of parametric rules. It is also possible to modify a plane primitive combining it with other primitives using Boolean operations. In conjunction with, for example, cylinder primitives, this approach allows for the creation of cross-sections

that are not solely based on straight lines. At the same time these cross-sections are fully parameterized. Figure 4-24(a) illustrates a parametric example created using two Boolean difference operations. The first one subtracts a cylinder,  $L_{0c}$ , from a plane primitive,  $L_{0b}$ . From the resulting object, a further plane primitive,  $L_{0a}$ , is subtracted. The parametric relations for the location parameters of the objects  $L_{0a}$  and  $L_{0c}$  are not shown for clarity. Figure 4-24(b) shows a swept revolve object that is generated based on the cross-section in (a).

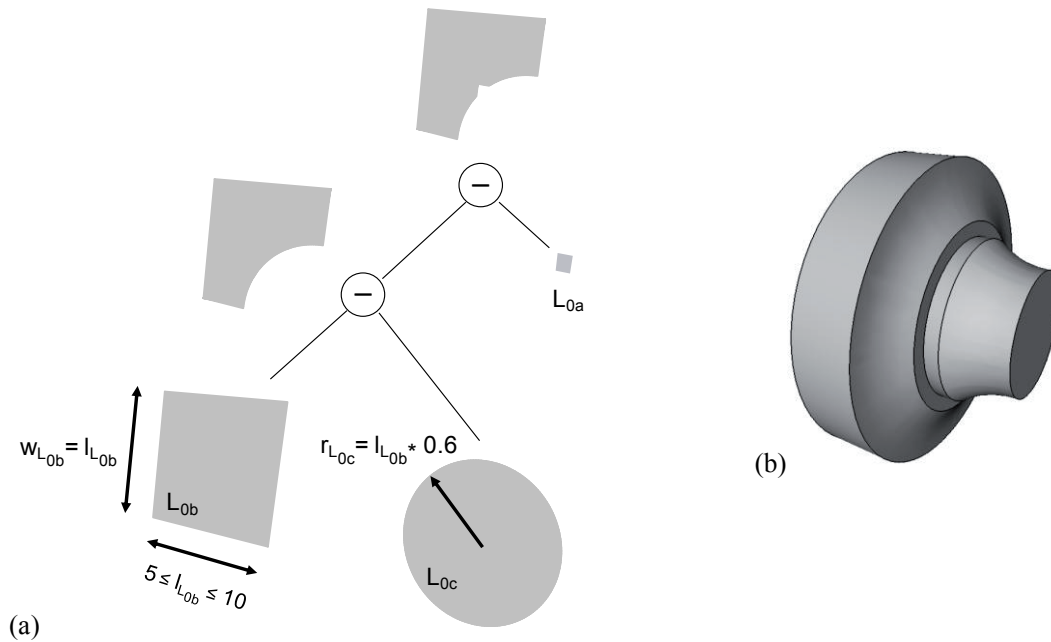


Figure 4-24: Example for a cross-section based on a plane primitive modified using Boolean operations (a) and revolved object derived from it (b)

Except for the fact that they are based on only one single object, swept objects are technically similar to Boolean objects. For example, changes to both the size or transformation parameters of the underlying cross-section directly affect the geometric shape represented by the swept object. The transformation of a swept object in 3D space cannot directly be defined but is implicitly given by the underlying cross-section. Therefore, as is the case for Boolean objects, the reference object in the LHS of a rule cannot be represented directly by a swept object. Instead, it has to be ensured that the cross-section is positioned in the global origin.

The process for matching the LHS in the CWS is also similar to the one for Boolean operations. Once an object of type ‘extrusion’ or ‘revolve’ in the LHS is reached during the process, first an object of the same type in the CWS has to be found. If this search is successful, the parameters, e.g. dirZ, axisX, angle, etc., are checked. If these parameters are all identical, a further sub-process for matching the underlying cross-section is initiated. A single plane primitive is handled like any other ‘conventional’ primitive (cf. 4.1.2 and 4.2.2). For a plane primitive that is additionally modified using Boolean operations, like the one shown in Figure 4-24(a), this sub-process is identical to the process for matching Boolean objects as described in 4.3.1.

The last possible kind of cross-section matching checks for the equality of the faces that are derived from profiles made out of straight lines, like the example in Figure 4-22. This sub-process is comprised of several steps. (1) First, a pre-check is performed that compares the number of edges of a profile and the size of the area enclosed by the profile. Only if both are identical, matching is generally possible. (2) Since the vertices and, therefore, implicitly the edges of a profile can be defined in an arbitrary sequence during the development of a rule, they are resorted so that they create a continuous path. (3) Based on the result of (2), the lengths of the single edges and the angles between adjacent edges are calculated. (4) With the ‘first’ edge in the profile in the LHS acting as the starting edge, edge after edge is compared to the CWS profile with respect to the calculated length and the angles between the adjacent edges. If one single length or angle does not match, the next edge in the LHS profile is chosen as the starting edge and the same procedure as described before is again performed. This is done clockwise and counterclockwise until a match is found or until none of the possibilities results in a match.

Replacing the matched objects in the CWS with the objects in the RHS is, again, identical to the methods described in 4.1.2 and 4.2.2.

## 4.4 Three-dimensional labels

With regard to the spatial grammar formalism (cf. 2.2.1), the focus of the approach described so far is on providing for the definition of shapes and rules in a way that is as general as possible. However, one component of the formalism, the set of labels, is not addressed so far. Labels can be used to better guide the shape generation process, i.e. constrain the application of rules, to help generating more meaningful or valid solutions. Further, they can be used to introduce additional, non-geometric data to rules and they provide the possibility to simplify LHS matching.

This chapter introduces labels to the spatial grammar approach integrating the different uses of labels (cf. Figure 2-3) in a single concept. As with the other geometric primitives that can be used in this spatial grammar approach, a label is specified such that it has its own local coordinate system (Figure 4-25). This enables the determination of the location of a label using translation transformations. Additionally, however, it enriches the label with information about its orientation in three-dimensional space based on the possibility to change the rotation parameters. This has an influence on the detection of the transformation required to apply a rule. Due to the fact that these labels carry information about location and rotation in three-dimensional space, they are denoted ‘three-dimensional labels’ or ‘3D labels’.

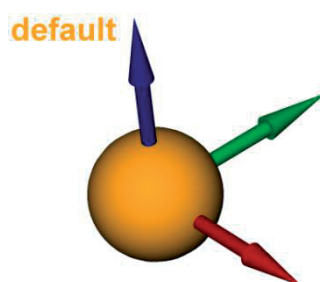


Figure 4-25: Three-dimensional label

3D labels do not have any geometric properties, but only carry transformation information. The sphere that can be seen in Figure 4-25 only exists for visualization purposes, i.e. it is merely displayed to ease the work with labels for the user in a visual, interactive grammar environment. Hence, the 3D label also cannot be detected as a match of a sphere object by the automatic LHS search during rule application. Without having any geometric properties, it is not possible to differentiate 3D labels by means of their shapes. Instead, the approach provides the possibility to assign different colors to a label. The effect is the same as using different shapes. At the same time, it simplifies the automatic matching of a label, as no shape needs to be matched, but only the symbolic numbers of the color values. Besides the color, a ‘label name’ can be defined as an additional property. This allows for assigning a letter or number to further distinguish labels. The label name is not restricted to one single character or number but can consist of a string of arbitrary length and therefore enables the possibility to introduce a wider range of additional information in a grammar rule in textual form. The standard name of a label is defined to be ‘default’ (cf. Figure 4-25), whereas it is also possible to define labels without using a name.

In principle, 3D labels have the same behavior as any other primitive in the approach. For example, they can be the reference object in the LHS of a rule or define different spatial relations together with other geometric objects. The application, including the matching of several objects and their relative transformations, stays the same as described in the previous chapters. It is also possible to define free parameters, the same way as explained for the definition of parametric grammar rules (cf. 4.2.1). Since labels do not have any geometric parameters, only the translation or rotation parameters can be ‘unlocked’.

A 3D label is not directly assigned to a shape, i.e. there is no explicit connection defined between the two. Instead, an affiliation can be implicitly given by the spatial relation between a label and a shape. A spatial relation other than the one defined in the LHS of a rule, i.e. a different relative transformation, would not be detected as a match in the CWS.

#### 4.4.1 General uses for three-dimensional labels

The concept of 3D labels introduced above can be used for all common applications of labels (cf. Figure 2-3). For use as a spatial label, only the information about the location but not the orientation is relevant. Unlocking all three rotation parameters and making them unrestricted makes a 3D label a ‘conventional’ spatial label. Using a label in this way, the application of rules can be restricted to a specific part of the CWS, for example, to the most recently added box as illustrated in Figure 4-26. Without labels it is necessary to manually choose one of the matches to avoid applying the rule to one specific box more than once. Using labels in the definition of the rule (Figure 4-26(a)) enables the automatic generation of the design in Figure 4-26(c). The application starts by matching the LHS of the rule to the initial shape shown in Figure 4-26(b), which contains a randomly oriented 3D label. It can be matched since the orientation for a spatial label is not important, i.e. the rotation parameters of the label in the LHS of the rule are all unlocked, or free.



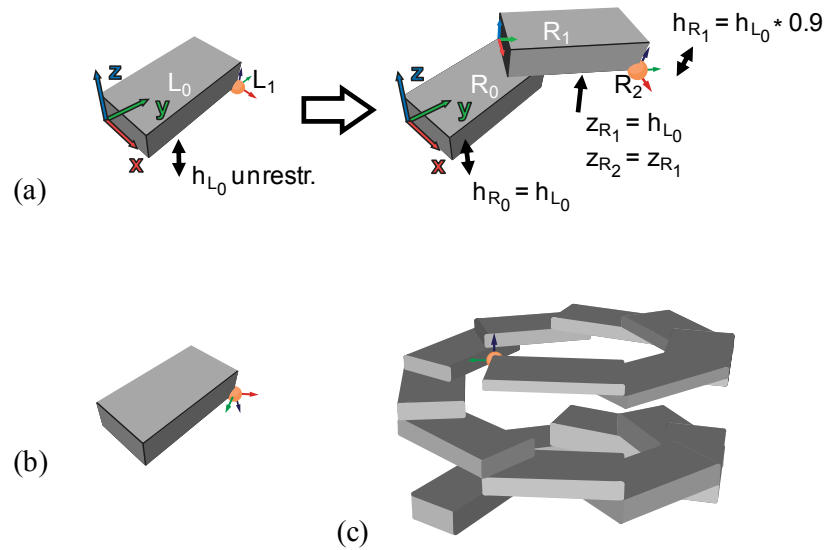


Figure 4-26: Three-dimensional label used as spatial label to restrict rule application to the most recently added shape in the CWS

For use as a state label, in addition to the rotation parameters, the location parameters are set as unrestricted to make the label completely ‘non-spatial’, i.e. neither the location nor the orientation matter, but only that the label exists. The label is then assigned to a rule side rather than to a specific shape. Using 3D labels as state labels, rules can be defined in a way that the labels implicitly determine the rule application sequence if different kinds of colors and/or label names are used. Also the number of rule iterations or the termination of the shape generation process using a rule that erases the existing label(s) can be controlled. In Figure 4-27, for example, the rules can only be applied iteratively according to the sequence (a)-(b)-(c) due to the labels. Rule (d) can be used to abort the generation process after three, six, nine, etc. rule applications.

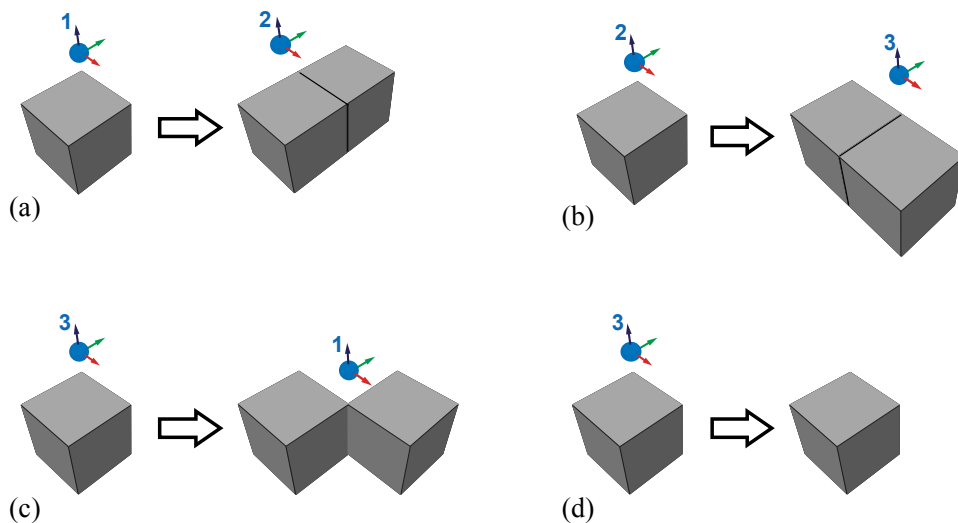


Figure 4-27: 3D labels used as state labels

3D labels can also be used to add non-spatial information to rules. Developing a new rule, one has to decide whether the additional information is relevant for the LHS matching process during the application of the rule or not. Depending on this choice, the different transformation parameters can be set as unrestricted or restricted. Most important for the augmentation of shapes is the additional ‘label name’. It can be used to classify or distinguish shapes or to add in a semantic meaning for a shape. The label name can also represent pure textual information that does not have any influence on the rule application but rather remains in the geometric model after design generation is finished. In this regard, such labels can be informative by adding information about design intent, manufacturing, for example, tolerances or welding information, measurement details, materials, texture, and colors.

The last possible use of labels, the simplification of the LHS matching, is discussed in more detail in the next subsection.

#### 4.4.2 Using three-dimensional labels for simplified LHS matching

Despite the possibility to use Boolean operations on objects and simple swept objects for the definition of parametric rules as described in Section 4.3, the complexity of the generated solutions can sometimes still be insufficient. Solids commonly used in mechanical engineering can be topologically more complex or even derived from free-form surfaces and include a wide range of detailing geometry, e.g. fillets and chamfers. Spatial grammar rules based on such complex geometry are difficult to handle generally in an implemented system because of the issues regarding automatic matching of the LHS.

3D labels provide the possibility to be used as a ‘substitute’ for geometry in the LHS and therefore circumvent the difficulties with the automatic matching of the LHS. In comparison to the example shown in Figure 2-7 where the straight line is an implicit reference to determine the orientation under which the RHS of the rule has to be added to the CWS, it is not obligatory to have any shape in the LHS in addition to the 3D label. This is because the 3D label, as introduced above, also carries information about orientation. The abstract additive rules in Figure 4-28 symbolize the difference between a ‘conventional’ rule (a) and a rule that contains only a 3D label in its LHS (b). Application of both rules would result in the same solution assuming that rule (b) is applied one more time than rule (a).

For a 3D label that might exist in the RHS, like the one shown in the example in Figure 4-28 (b), additionally a parametric relation can be defined, so that, for example, its location is dependent on an unrestricted parameter of one of the shapes in the RHS. It then implicitly depicts the location of the next rule to be applied in relation to the geometry in the RHS of the currently applied rule.

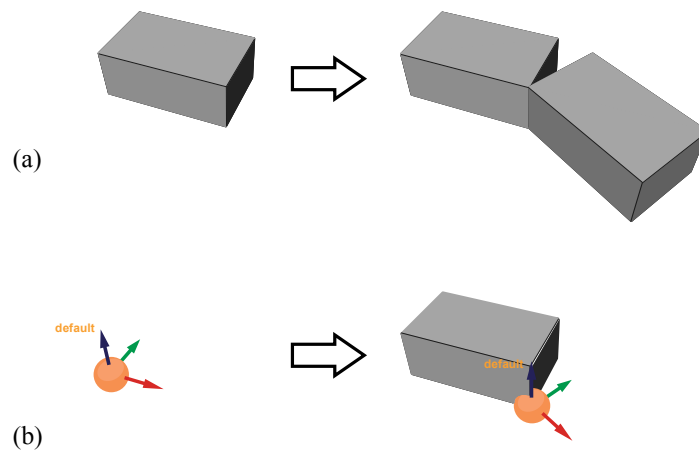


Figure 4-28: Conventional rule (a) and rule with only a 3D label in the LHS (b)

Rules developed based on this approach can contain an arbitrarily complex, parametric RHS and can be applied as long as the 3D label in the LHS can be matched. To generate meaningful mechanical engineering parts using three-dimensional labels for simplified LHS matching, it is necessary that the interfaces between the shapes in the different rules are ensured to fit once they are inserted in a design. Shapes with specified interfaces are denoted ‘segments’ in this thesis. They are the subparts or building blocks used to generate mechanical engineering parts as symbolically illustrated in Figure 4-29.

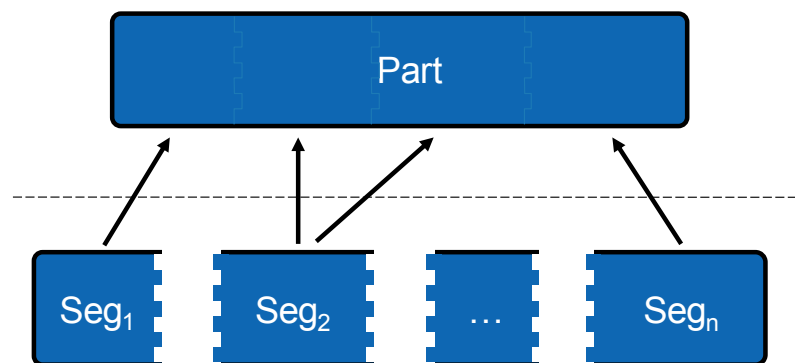


Figure 4-29: Segments and their interfaces used to compose a mechanical engineering part (symbolic)

This concept has an analogy to an approach often used in variant management, called modular construction or assembly systems (see e.g. ULRICH 1995 or PAHL et al. 2007). In modular systems the aim is to define a limited number of basic building blocks, or modules, with well-defined interfaces so that they can be combined to define a large range of product variants. Often these building blocks are derived from existing related product variants by extracting their commonalities. The extraction and definition of building blocks from existing designs is also applicable for the definition of segments. Ideally, in modular systems, the parts or subassemblies are used more than once in one product and so are segments in a part (see

Figure 4-29). In comparison to modular systems that operate on the assembly or product level, the segment concept is used to generate parts through a series of interfaced segments. To allow for more complex part compositions, it is useful to use more than one type of interface together with different kinds of labels for the definition of rules.

Three example rules that use 3D labels in the LHS and segments with complex curved geometry in the RHS are shown in Figure 4-30(a)-(c). The initial set Figure 4-30(d) contains another segment and two 3D labels. The interfaces of all segments are identical. An example for a solution generated applying the rules is shown in Figure 4-30(e).

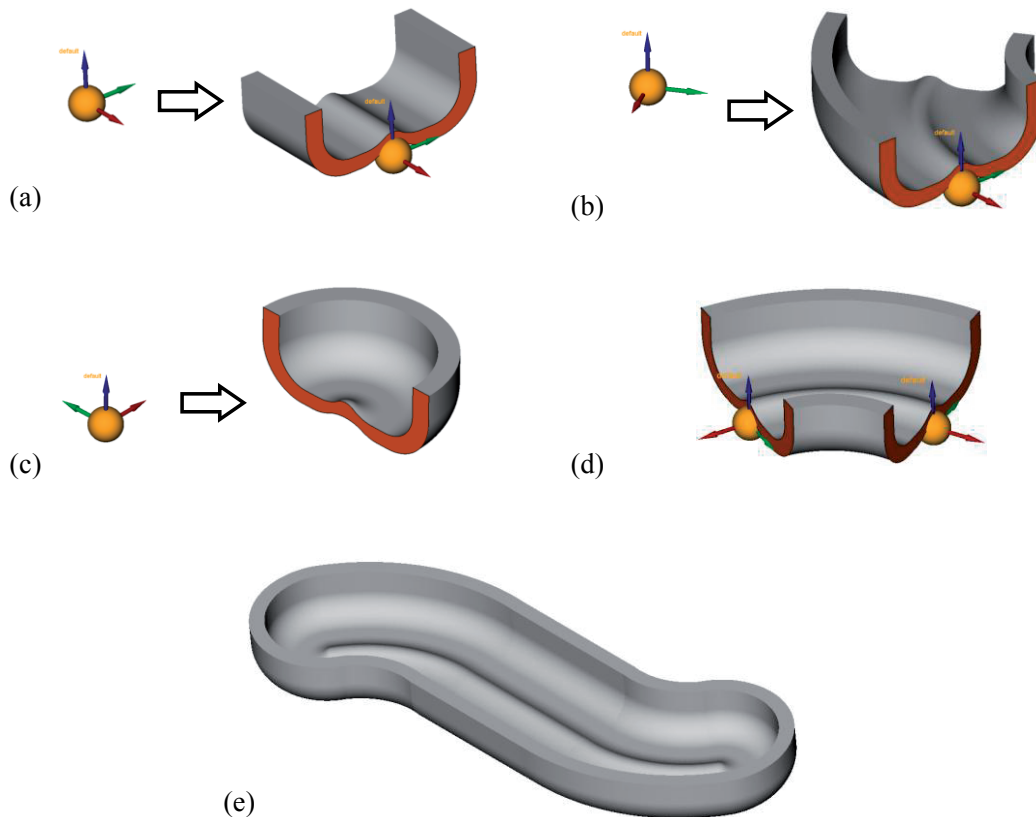


Figure 4-30: Example for a grammar using 3D labels and segments

As the geometry in the RHS is of no relevance for the further application of rules in a CWS, there are basically no restrictions on how segments are created. CAD systems usually provide many modeling concepts that can be used for designing segments. They can be modeled using different features (cf. 2.5) together with parametric primitives, Boolean objects or other advanced parametric modeling techniques like UDFs, sweeps along three-dimensional trajectories, lofts for the generation of solids or parametric free form surfaces to derive solids. Further, it is possible to import solid geometry from another modeling system or 3D model catalogues, for example using standard exchange formats like STEP. Since geometry that is imported using an exchange format is static, i.e. non-parametric, it cannot be used for the definition of parametric rules on its own. However, in conjunction with, for example, parameterized primitives a partly parametric RHS can be defined.

## 4.5 Collision detection

Mechanical engineering parts or products usually have to fulfill different requirements and underlay certain constraints. Incorporation of all constraints needed in grammar rules is often only possible by developing grammar systems for one specific example, or purpose, and hard-coding them. These grammars, however, are often static and are not easily adapted once they are created. The specification of constraints on a general level is a more difficult task, especially in an approach as the one presented in this work that provides for a more general platform and allows for the visual definition of rules. Parametric relations and parameters that are restricted to certain ranges (cf. 4.2) or the use of 3D labels (cf. 4.4) are mechanisms that allow for the visual incorporation of constraints in spatial grammars. These techniques are defined and used internally within rules.

In this section collision detection is introduced to the approach as a further, rule-external possibility for constraining the design generation. A simple principle for the detection of a collision between two objects is used: The objects are intersected and the result is checked for the existence of vertices. If vertices exist, a collision is detected. For example, the intersection of the box and the cone in Figure 4-31(a) results in the shape shown in (b). A collision is detected because the resulting object contains several vertices (shown in (c) using a wireframe visualization of the object).

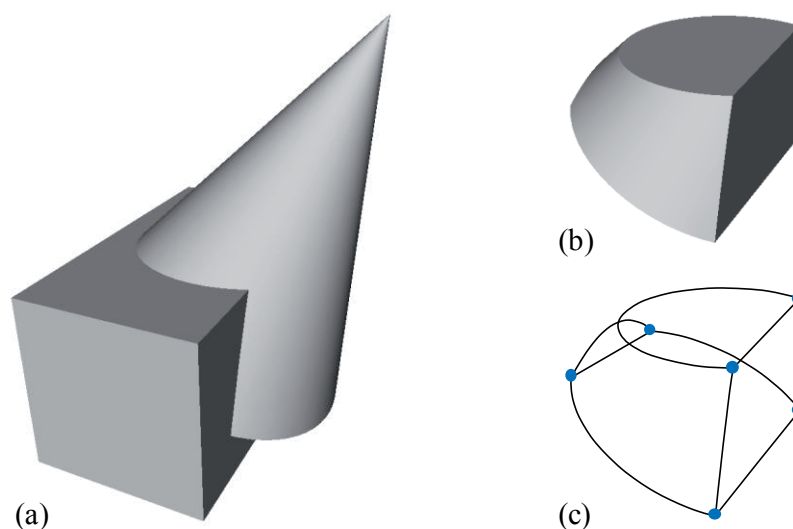


Figure 4-31: Principle for the detection of a collision between two objects

This collision detection mechanism can be used for two different purposes during rule application, first to avoid collision of parts in assemblies and second to restrict the design space, i.e. the boundary within which design generation can occur.

### 4.5.1 Part collision avoidance

In many cases of grammar-based design generation it is not relevant whether the single geometric objects interpenetrate each other in the virtual model or not. This is especially true in cases where one single part is created, i.e. a ‘mechanical’ part that would be one continuous chunk of material when physically produced. Interpenetration is often explicitly wanted because it can support the generation of more unexpected designs. In fact, it can be a valuable technique in design and designers often conceive of designs in terms of interpenetrating masses or volumes (STINY 1980b).

However, there are also cases where interpenetration of parts is unintended. Especially if the geometric objects represent different separated parts that could be physically assembled into a product, it has to be ensured that they do not collide with each other. In cases like this, collision detection can be used to avoid the overlap of single parts.

As collision detection is a rule-external mechanism, it is not part of the rule development step. Instead, it can be activated before the application of a rule is started. The effect is that, once the application process reaches the step of substituting the LHS with the RHS of a rule and therefore inserts the RHS objects into the CWS, all of the RHS objects are checked for collisions with any other object that already exists in the CWS, except the ones that are in the LHS match of the rule in the CWS. If any of the objects collides, the attempt to apply the rule is retracted, the already inserted RHS objects are removed from the CWS and the overall process continues trying to apply the next rule.

In the example given in Figure 4-32, two bushings as shown in (a) are to be placed on the shaft in (b). The rule to accomplish this task is given in (c). It is parametric so that it can match shafts of arbitrary length. Further, the translation of the bushing that is added in the RHS of the rule is restricted to a range to ensure that it is not placed beyond the height of the shaft. The first application of the rule on the shaft in (b) places a bushing with a randomly chosen translation within the given ranges resulting in (d<sub>1</sub>). To insert the second bushing, the rule is applied again. This time, the newly added bushing, shown transparently in (d<sub>2</sub>), collides with the already existing one. Therefore, the attempt to apply the rule is retracted and the bushing is removed from the CWS. The rule is tried another time. Now, a different location is randomly chosen, which does not result in a collision creating the solution in (d<sub>3</sub>).

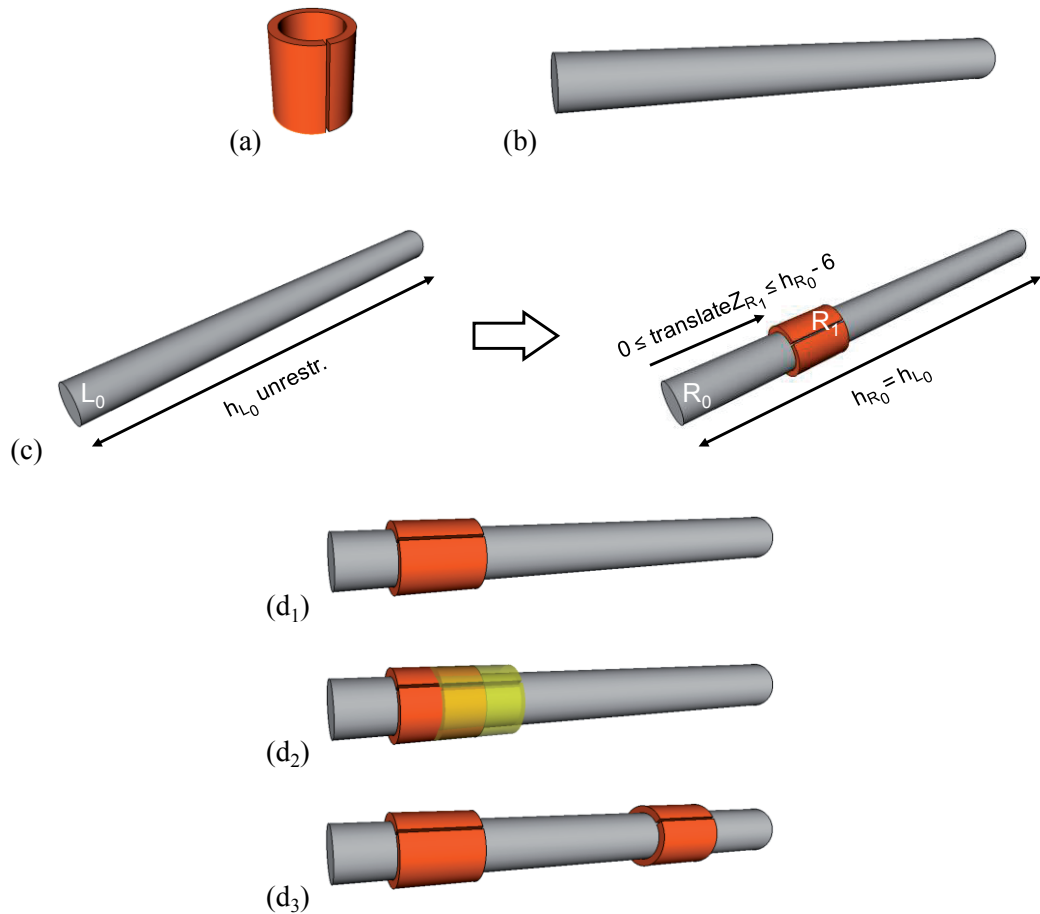


Figure 4-32: Example for a grammar rule application avoiding collision of parts

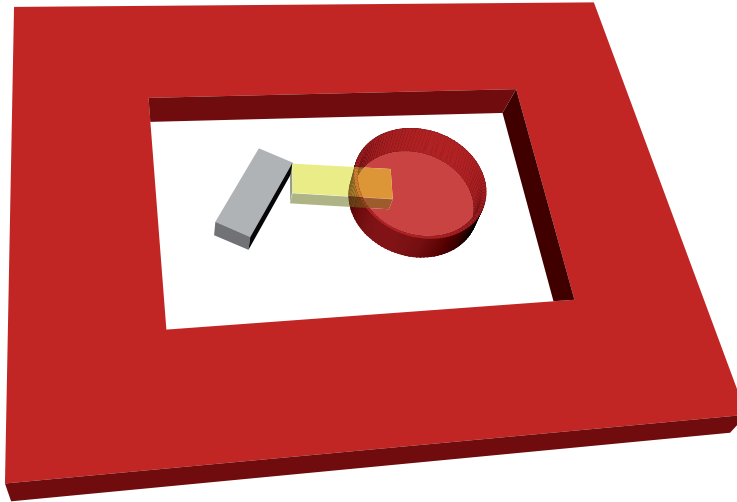
### 4.5.2 Design space restriction

In addition to avoiding overlapping parts, collision detection can also be used for the restriction of the available design space. This can be an outer border that must not be exceeded by the generated design. Or, it can be any other obstacle to define a certain space to be kept clear by the design generation process.

A design space restriction can be designed like any other geometric model, but the objects used have to be converted to a state denoted ‘obstacles’ so that they can be differed from the objects used in the grammar rules. Consequently, obstacles can only be defined in the initial set and not in rules. Applying rules in a semi-automatic mode, it is theoretically also possible to manually select different objects and convert them to obstacles before the next rule is applied. With regard to the grammar formalism, however, this would mean that after every rule application a new initial set and, therefore, implicitly a new grammar is defined.

If there are obstacles defined, they are checked for collisions once other objects are inserted. Figure 4-33 shows an example for a restricted design space made out of an outer border and an additional cylinder within the border. A rule that is applied on the box located within the

design space results in the collision of the newly added box shown transparently with the cylinder obstacle. The attempt to apply the rule is retracted and the already inserted box is removed from the design.



*Figure 4-33: Example for a restricted design space with an object detected colliding with an obstacle*

For clarity, the restriction of the design space in the figure above is represented by simple 2,5D obstacles. In general, however, the design space restriction can be arbitrarily complex. For example it is possible to use already existing parts as obstacles. This allows for a grammar-based design generation of a part within a given design space that is defined implicitly by the surrounding parts. If a sub-assembly consisting of more than one single part is to be generated within the given design space, additionally the part collision avoidance (4.5.1) can be activated.

## 4.6 Summary

A new three-dimensional spatial grammar approach was presented in this chapter. The basics of the approach include the use of parameterized primitives and automatic matching of the LHS of a rule. To allow for a wider variety of solutions to be generated on the basis of only a few rules, parametric rules were introduced. To increase the complexity of shapes that can be defined in rules, the approach was enhanced by the possibility to use Boolean and sweeping operations in grammar rules. The concept of 3D labels was developed which can be used to constrain the application of rules, add additional information to rules and provide the possibility to simplify LHS matching which enables the use of geometrically complex segments. A mechanism to avoid part collisions and to be able to constrain the design space was added in the last section. The different components of the approach provide the basis for the implementation of a general, three-dimensional spatial grammar system that enables the development and application of grammar rules in a visual, interactive manner.



## 5. Software prototype

A prototype software system of the approach described in Chapter 4 has been implemented and published as open source software<sup>11</sup>. Figure 5-1 shows a screenshot of this system during the definition of a rule.

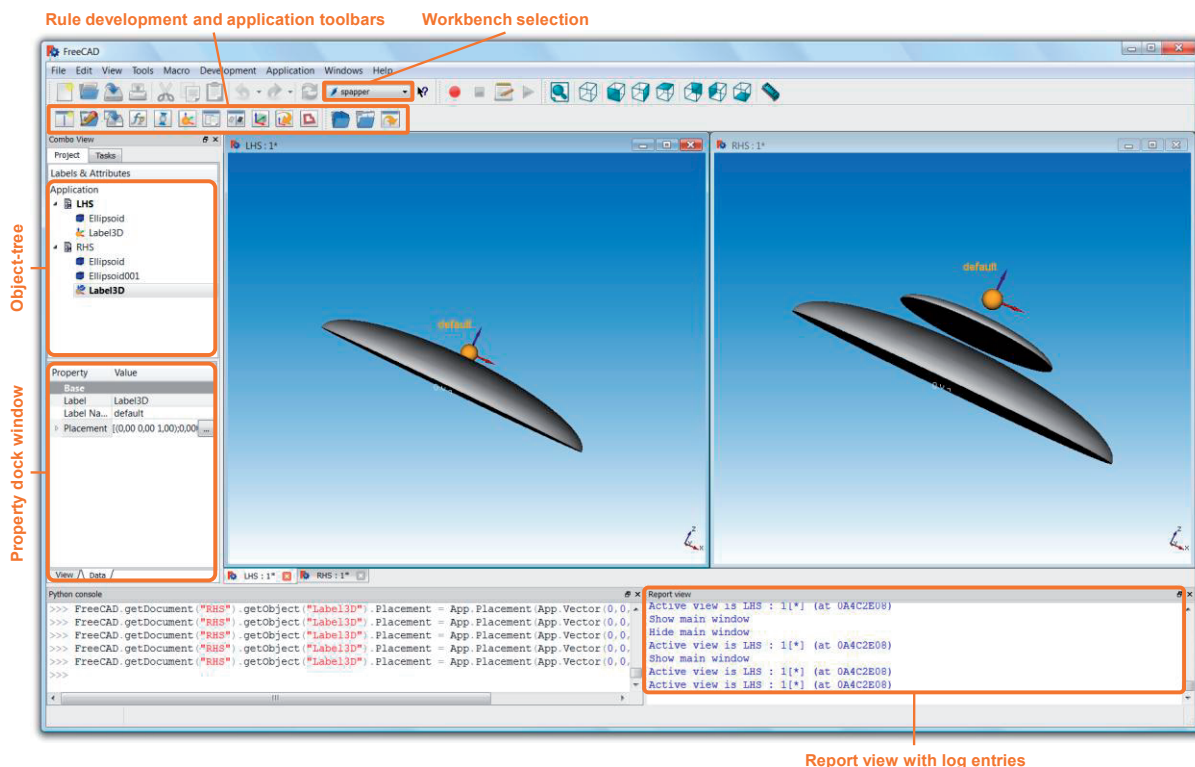


Figure 5-1: Screenshot of the prototype software system

This chapter elucidates the most important aspects of the software prototype. It starts with the underlying basics of the system and then moves on to explain how rules can be developed and applied using the provided functionality. Therefore, this chapter aims to not only explain how one can work with the system, but also how the system works.

### 5.1 System basics

The software prototype is based on an open source 3D mechanical engineering CAD system<sup>12</sup> that, in turn, is built using an open source geometric modeling kernel<sup>13</sup>. By using the existing

<sup>11</sup> <http://sourceforge.net/projects/spapper/> (accessed January 14<sup>th</sup> 2012)

<sup>12</sup> FreeCAD, <http://sourceforge.net/apps/mediawiki/free-cad/> (accessed January 14<sup>th</sup> 2012)

<sup>13</sup> OpenCASCADE, <http://www.opencascade.org/> (accessed January 14<sup>th</sup> 2012)

user interface and the functionalities for geometry generation and manipulation provided by the kernel and the CAD system respectively, the coding effort was kept within reasonable limits. The CAD system uses Python<sup>14</sup> as an internal scripting language and for the realization of an application programming interface (API). This allows the full range of CAD functionality to be addressed, for example for the definition and manipulation of geometry, but also to directly modify the GUI of the system<sup>15</sup>. The approach for the interpreter is, therefore, realized as a Python module that is integrated at the startup of the program. It adds an additional workbench to the CAD system, including two special toolbars for the development and application of spatial grammar rules.

## 5.2 Development of grammar rules

The toolbar for the development of grammar rules comprises eleven different buttons whose functionality is explained in this section.



For the development of a new rule, two empty windows, containing a visualization of the global origin, are opened for the design of the geometric objects and their relations in the LHS and RHS, as shown in Figure 5-1. For the definition of new objects, the functionality of the existing ‘Part’-workbench of the CAD system can be used. This workbench provides a toolbar for the definition of the different primitives. However, it is recommended to use the ‘Geometric Primitives’ dialog window accessible via the main menu (‘Part > Create Primitives...’), because it contains a few additional primitives not included in the toolbar. The values for the parameters of the primitives, including size, location and orientation parameters, can be changed in the ‘Property’ dock window located on the left hand side of the program window (see Figure 5-1). Location and orientation parameters can be found under the entry ‘Placement’. Changing parameter values is realized using numerical inputs, upon which the visualization of the geometry is immediately updated according to these inputs. The ‘Placement’ can alternatively be changed visually by directly manipulating the translation and rotation of a geometric object with the mouse, after double-clicking it in the objects-tree in the upper left hand side of the CAD system. The ‘Part’ workbench further includes two toolbars that have the added functionalities to perform Boolean operations on geometric objects and to create extruded and revolve objects. The only functionality for the design of objects in grammar rules that is not directly available in the ‘Part’ workbench is for the creation of two-dimensional cross-sections made out of straight lines (cf. 4.3.2). This can be found in the ‘2d Drafting’ workbench.


Using the functionalities described so far, it is already possible to define any kind of non-parametric grammar rules. Furthermore, for the definition of 3D labels, which is not part of the standard geometric primitives’ toolbar, an extra button is available. Once inserted into a rule, a 3D label can be handled like any other geometric primitive. Location and orientation can be changed using the ‘Placement’ property. The further parameters of a 3D label, color and label name, can be found and modified in the two tabs of ‘Property’ dock




<sup>14</sup> <http://www.python.org/> (accessed January 14<sup>th</sup> 2012)

<sup>15</sup> Via PyQt, <http://www.riverbankcomputing.co.uk/> (accessed January 14<sup>th</sup> 2012)

window. If a larger sized model is designed defining a rule, the 3D label's size can be increased using a scaling factor. This helps to ensure the visibility of the label and, therefore, the ability to conveniently work with it. The size of the label does not have any influence on the actual rule and its application, as only the existence of the label, its transformation information, its color and its name are of relevance.

The grammar-based design generation does not necessarily have to begin with an initial set that contains geometric objects. If the generation is supposed to start from scratch, an initial set cannot contain any geometry. However, the LHS of a rule must not be completely empty, as for the application of the rule it would not be clear what to search for and under what transformation to insert the RHS into the CWS. For this reason, a special object, called  'starting symbol', can be inserted into the LHS side of a rule or into an initial set. Generally a starting symbol is a special kind of label and similarly does not have any internal geometrical representation. While the symbol is visible in the GUI for increased usability, it is only relevant for the application of a grammar rule; what it looks like is irrelevant. In comparison to a 3D label, it does not have any parameters and there is no possibility to translate or rotate it in 3D space, i.e. it can only be inserted in the global origin. The use of a starting symbol is especially helpful if the RHS of the first rule to be applied is parametric or if there is more than one rule containing a starting symbol in the LHS.

For the development of parametric rules the possibility to define free parameters is provided.

 By selecting a geometric object and pressing the 'fp'-button, a dialog window is opened that lists all size, location and orientation parameters of the object, as well as all relevant free parameters that have been defined through other objects in a rule (Figure 5-2). A parameter is unlocked, or 'freed', by ticking the according check box in the 'Unlock'-column. Once unlocked, the parameter appears in the list of free parameters on the right hand side of the dialog window.

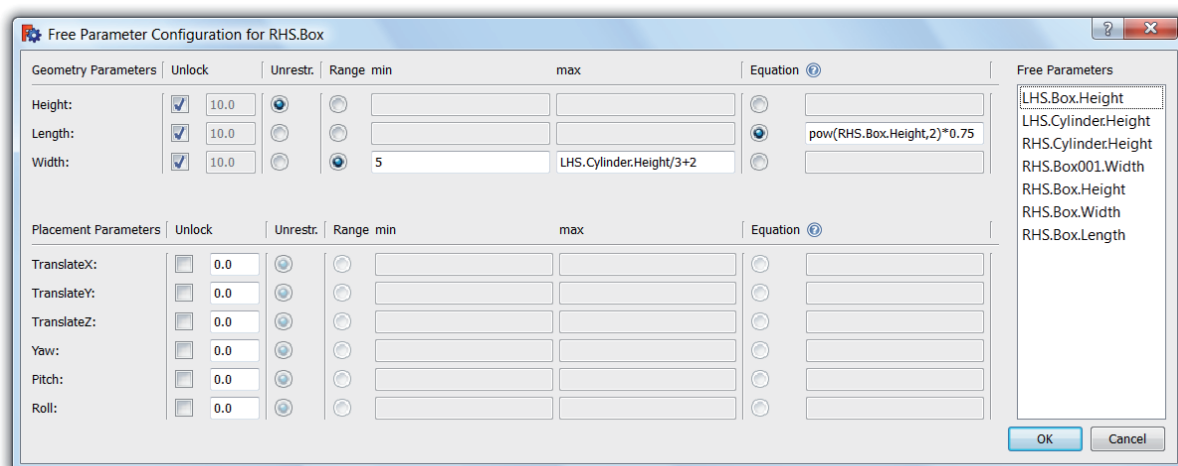


Figure 5-2: Dialog window for the definition of free parameters

To make it clear to which object(s) free parameters belong and to make them system-internally accessible, the following convention for a free parameter's full name is used: the first part represents the name of the *rule side*, followed by the name of the *object* and the name of the actual *parameter*, all separated by points, for example, *RHS.Box001.Width*.

As described in the approach section of this thesis, a free parameter can be completely unrestricted, restricted by a certain range or defined by a parametric relation. After unlocking a parameter, by default it is completely unrestricted. This can be seen in the dialog window through the radio button set in the 'Unrestr.'-column. Choosing a different option for the radio button allows for the definition of a range or an equation. In the latter case, an equation-string is defined by dragging and dropping parameter names from the free parameters list to the input field; the system automatically prevents making a parameter dependent on itself. The parameters in the input field can be extended to an equation by typing in additional operators and operands or mathematical functions. The full range of functions provided by the Python math library<sup>16</sup> can be used. Free parameters of the selected object but also of other objects can be included in the definition of an equation. For the definition of equations in the LHS, only the LHS parameters are provided in the free parameters list, for equations in the RHS, parameters from both sides are available. To restrict a free parameter to a certain range, numbers can be typed into the input fields that are provided for the definition of the minimum and the maximum. Or, instead of numbers, equations can be created the same way as described for parametric relations. In the case that only one range limit is needed, one of the input fields can be kept blank.


Once the OK button is pressed, the system internally performs several checks to ensure the validity of the defined free parameters. As mentioned in Chapter 4.2.1, the parametric relations are transformed into a dependency graph and, using a topological sort algorithm, possible cycles between parameters are detected. Further, if the radio button is either set to range or equation, it is checked whether the related input fields are filled in. In the case of range parameters, if only numeric values are used, their validity is checked with regard to the related parameter. This includes a check to ensure that the maximum is bigger than the minimum and that certain geometric, or internal boundaries, are not violated, for example, the minimum for size parameters must not be negative. Further, it is checked whether the objects of all parameters used in an equation still exist, because the related objects might have been deleted since the equation was defined.





To facilitate the development of rules, a few auxiliary functions are provided. One is the possibility to copy selected objects from one rule side to the other. This is especially helpful for defining additive or subtractive rules, because in these cases either the LHS is part of the RHS or vice versa and, thus, the related objects do not have to be designed twice. It is also convenient for copying Boolean objects that are based on several other primitives because only the resulting, and not every single underlying object, has to be selected in the object-tree.


---

<sup>16</sup> <http://docs.python.org/library/math.html> (accessed January 14<sup>th</sup> 2012)

Sometimes it is difficult to recognize and correctly modify the spatial relations of objects displayed in the LHS with respect to the ones in the RHS. Especially due to the standard functionality of the underlying CAD system that allows zooming and changing the view, a wrong impression of the real sizes of the displayed objects can be created. For this reason, a  shaded, half-transparent version of the complete LHS of a rule can be displayed in the RHS. Internally, the displayed representation of the LHS, not its geometrical representation, is added to the RHS. This is done by integrating the scene graph<sup>17</sup> of the LHS window into the scene graph of the RHS as a sub-graph.

The definition and application of a rule is based on the relative transformations between the different objects. The location and orientation in the 3D space are defined via the local coordinate system of every single object (cf. 4.1.1). During the visual development of a grammar rule it can sometimes be difficult to visually identify the current ‘positions’ of the local coordinate systems. For example, if a cube is rotated 90 degrees it visually looks the same as a completely un-rotated cube, but the spatial relation to other objects would be  different. Therefore, it is possible to show the local coordinate systems of selected objects to easily identify their orientation. Regarding the implementation, the representation of the local coordinate system is realized as a special sub-graph inserted into the scene graph.

As explained in the approach section (cf. 4.1.1), there is always one reference object to be defined in the global origin in the LHS. All remaining objects in the LHS and RHS are positioned in relation to this object and, thus, implicitly in relation to the global origin. If the decision is made to define a different object as the reference object, it can be a tedious task to re-position the remaining objects accordingly. To facilitate changing the reference object, a  function is provided to recalculate the position of a selected object to the global origin and relatively transform the remaining objects. This functionality can also be helpful in cases where it is easier to define a spatial relationship between objects directly and not in accordance to the reference object. For example, it is easy to define a primitive rotated 15 degrees around the x-axis and a second primitive rotated 15 degrees around the y-axis. Recalculation of the first object back to the global origin to make it the reference object results in values for rotation and translation of the second object whose ‘manual’ calculation would be rather tedious. The functionality provided by this button is not only restricted to deal with reference objects, but it can also be used in the RHS of a rule if needed.

 Once fully defined, rules can be saved in a folder on the computer system. A rule is saved in an archive file<sup>18</sup> with the filename extension ‘.rule’. Internally, this archive consists of four files, shown in the overview in Figure 5-3. Two of them are the CAD files of the LHS and RHS that are saved in the standard format<sup>19</sup> of the underlying CAD system. This standard format is an archive itself containing the designed geometry as boundary

---

<sup>17</sup> FreeCAD comprises Coin3D, an Open Inventor clone, which is built on OpenGL and uses scene graph data structures to render 3D graphics, <http://www.coin3d.org/> (accessed January 14<sup>th</sup> 2012)

<sup>18</sup> The used format for the archive is the ZIP file format

<sup>19</sup> The filename extension of files saved in the standard format of FreeCAD is ‘.FCStd’

representation (B-rep) data and an additional XML-file<sup>20</sup> that saves the data as defined in the ‘Property’ dock window (see Figure 5-1).

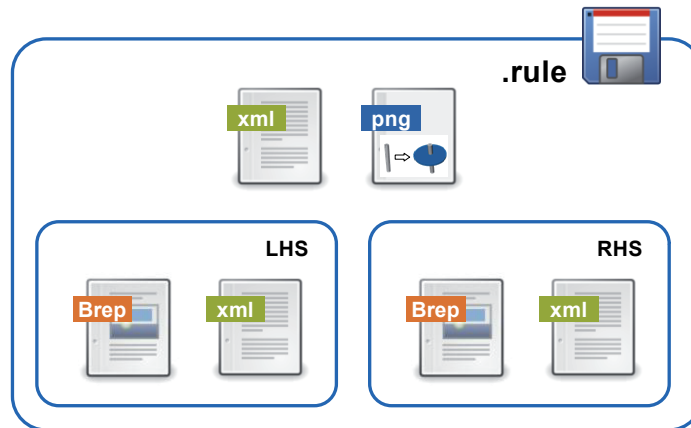


Figure 5-3: Content of a rule file

In addition to the two CAD files, a thumbnail (‘.png’-format), that is, a preview image which is shown once a rule is loaded for application, is created and saved in the archive. The last file contains all free parameters that are defined for a rule, again in XML-format. The specification of a free parameter comprises its name, its type (unrestricted, range or equation), and, if needed, the minimum and maximum or the defined equation. An example is given in Figure 5-4. To save the parameters in the correct sequence, so that it is ensured that they are evaluable during rule application, they are sorted using the topological sort algorithm as described for the definition of free parameters (cf. 4.2.1).

```
<Parameters>
  <Parameter fptype="unrestricted" name="LHS.Cylinder.Height" />
  <Parameter fptype="range" max="LHS.Cylinder.Height/3+2" min="5" name="RHS.Box.Width" />
  <Parameter equation="LHS.Cylinder.Height" fptype="equation" name="RHS.Cylinder.Height" />
  <Parameter fptype="unrestricted" name="LHS.Box.Height" />
  <Parameter fptype="unrestricted" name="RHS.Box001.Width" />
  <Parameter fptype="unrestricted" name="RHS.Box.Height" />
  <Parameter equation="pow(RHS.Box.Height,2)*0.75" fptype="equation" name="RHS.Box.Length" />
</Parameters>
```

Figure 5-4: Snippet of an XML-file containing the definition of free parameters

Before a rule is finally saved, several checks are performed to ensure its validity. For example, in the LHS at least one object must be defined, a reference object must exist and, in case a starting symbol is used, no further objects are allowed to be inserted.




Any rule that has been saved can be opened for modification. In this case, the system internally extracts the archive file described above, the two CAD files are opened and


<sup>20</sup> XML = EXtensible Markup Language

the XML-file containing the definition of the free parameters is parsed and delivered back to the program.

Besides the rules, an initial set has to be defined to develop a complete grammar. This is done using a standard file in the CAD system and the functionality of the 'Part'-workbench that is already described for the definition of rules in the beginning of this chapter. An initial set can consist of geometric objects, 3D labels or a starting symbol. Further, it can contain design space restrictions as described in Chapter 4.5.2. Any geometric object, also imported geometry<sup>21</sup>, can be selected and converted to an obstacle. These objects are moved to a special group called 'Obstacles' that can be seen in the object-tree of the CAD system. At the same time they are colored red and are shown transparently in the design window. If needed, they can also be converted back to 'normal' geometric objects.

### 5.3 Application of grammar rules

The toolbar for the application of grammar rules consists of three buttons. To apply rules, first the initial set of the grammar is opened. As the initial set is saved in the file format of the CAD system, it can be opened using the standard open dialog. However, using the  button in the grammar application toolbar additionally adds a visualization of the global origin<sup>22</sup>. This makes it easier to identify the position of the object in 3D space, which is often helpful in cases where rules do not generate expected results and need to be analyzed in order to understand how to adapt them.

 After the initial set, rules can be selected and loaded into the system. Loading rules opens a new dock window (Figure 5-5) on the right hand side of the program window. The chosen rules are displayed in a re-sortable list that represents the application sequence of the rules. Every rule-entry in the list consists of the rule's file-name and the preview image of the rule that was generated when the rule was saved (cf. 5.2). The latter makes it easier for the user to visually distinguish the rules, especially if they are applied manually. As described in the approach chapter (4.1.2), for every rule that is applied, one out of all found matches of its LHS in the CWS has to be selected. It is possible to determine for every rule in the list whether this selection has to be done manually by the user or whether it is done randomly by the system. Further, if a rule contains free parameters in its RHS that are completely unrestricted or restricted by a range and, therefore, require assignment of a concrete value, it can be depicted whether this value will be randomly chosen by the system or has to be manually defined by the user. The checkbox at the beginning of every list entry can be ticked or un-ticked to determine whether the rule is active for the application or not. This makes it easy to explore the impact of single rules in a grammar on the generated solutions. Clicking on the arrows on the right hand side of a list entry moves the position of a rule up or down resulting in a change of the rule application sequence. In case a special rule sequence is needed that, for example, requires one and the same rule to be applied more than once, rules can also be loaded into the list more than once.

---

<sup>21</sup> For example STEP files can be imported from other CAD systems

<sup>22</sup> The current version of FreeCAD does not show the global origin

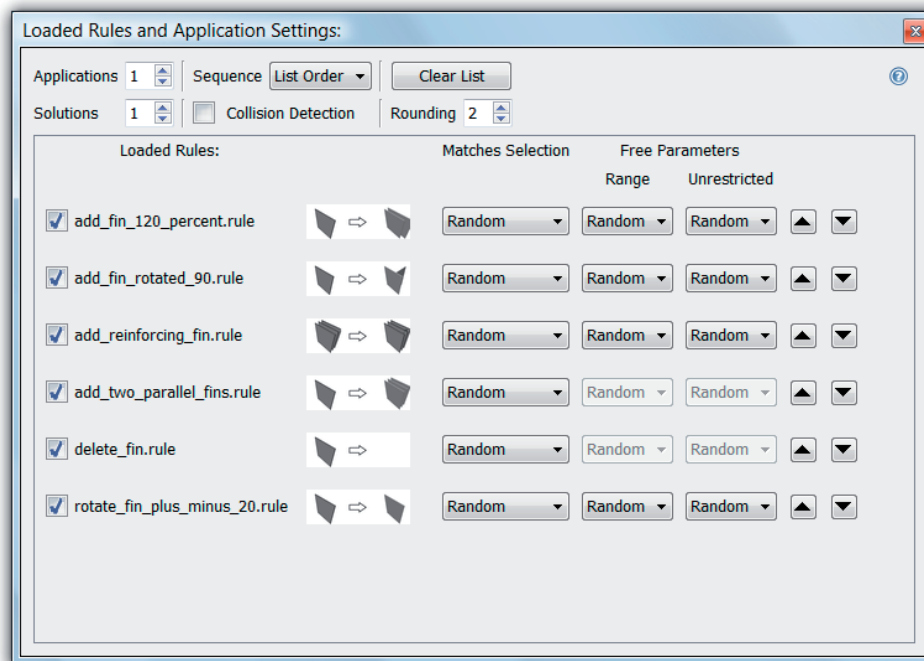


Figure 5-5: Dock window with list of loaded rules and application settings

In addition to the settings for each rule, several general configuration options concerning the application of rules are available. The field ‘Applications’ depicts the number of rules to be applied. This is a rule-external mechanism to abort the rule application even if further rules would be applicable. On the other side, as it is also possible that no more rules in the list are applicable, the system stops the rule application even if the depicted number of applications has not been reached. If the number of rule applications selected is higher than the actual number of active rules in the list, the application continues at the beginning of the list once the end of the list has been reached. Rules can not only be applied in accordance to the sequence defined in the list, they can also be randomly chosen by the system. In conjunction with the possibility to load a rule into the list more than once, this can be used to implicitly define a weighting of the rules. Taking, for example, a grammar with two rules and loading the first rule three times, but the second one only once, would create a 75-percent chance that the first rule is applied.

If more than one solution is needed, the number of solutions generated can be increased. Generally this makes sense only if more than one rule is loaded in conjunction with random rule selection or if the loaded rule(s) are parametric. Otherwise the same result would just be generated several times. Further, the option for collision detection can be activated to avoid overlap of single parts or shapes (cf. 4.5.1). The last configuration option allows for depicting the number of digits after the decimal point that is used to round floating-point numbers when parameters in the LHS and the CWS are checked for equality during the LHS matching process. This is mainly relevant for the calculation of the relative transformation between objects in rules that have at least two objects in the LHS and, predominantly, if rotational



transformations are used because they involve sine and cosine functions. The standard value, which is set to 2, is appropriate in most cases. Clicking the button on the upper right hand side clears the current list of loaded rules and resets all application settings to the default configuration.



Once all rules are loaded and the configuration options for the application are set as needed, the application process can be started by clicking the 'Apply rules'- button in the application toolbar. The system applies the rules trying to match the LHS according to the method described in Chapter 4. If the option to manually select one of the found matches is chosen, a dialog window listing all found matches is opened after the LHS matching process is finished (see example in Figure 5-6).

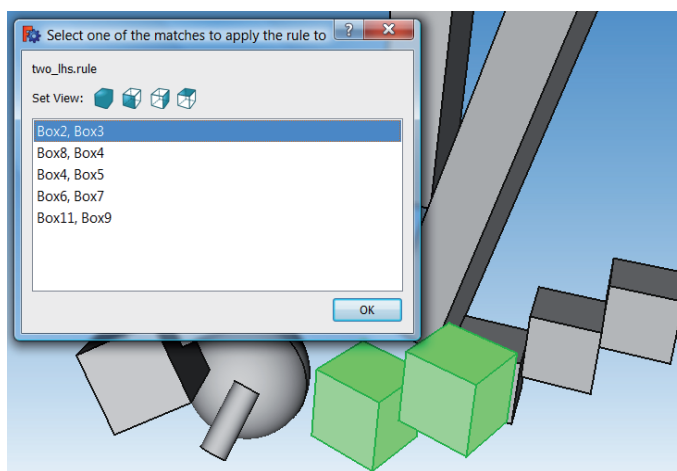


Figure 5-6: Dialog window to select one of the matches with the according CWS in the background

Selecting an entry in the list highlights the related match in the CWS. Once the OK button is pressed, the system replaces the selected match with the RHS of the rule as explained in Chapter 4. If the rule includes free parameters in the RHS that have to be chosen manually, another dialog pops up to allow the user to input the actual value. The system checks whether the user input satisfies any implicit boundaries or defined range limits.

To make it easier to comprehend why a rule has not been matched or why its application delivered an unexpected result, the system records log entries about internally performed checks and calculations (see lower right in Figure 5-1). Through this log the user can better track in what ways a rule has to be modified to reach the desired behavior.

## 5.4 Summary

A prototype system for a spatial grammar interpreter was described that implements the single components of the approach presented in Chapter 4. Based on an open source CAD system, it enables the visual, interactive development and application of three-dimensional spatial grammar rules. Additional functionalities were integrated that make the system more user-friendly and automatically check for errors or inconsistencies, for example, to ensure the

validity of parametric rules. An overview of the most important implementation issues and an instruction to the functionalities of the system were given in this chapter.

To summarize, the different settings allow the user to specify whether the application of rules should be done manually, semi-automatically or automatically (cf. Figure 2-9). In the case of fully automatic rule application, the system selects a rule, either according to the rule list or randomly, it automatically determines the matching conditions and objects in the CWS that match the LHS, and it randomly selects one of the found matches and assigns random values to any free parameters that require the assignment of a concrete value. For the manual application of rules, the user makes the decisions and gives the needed input data. For example, manual rule selection can be realized by activating only one of the rules in the rule set and setting the number of rule applications to one. Additionally, the LHS match selection can be set to manual as well as the determination of free parameters. The only step that is always performed by the system is the matching of the LHS.

## 6. Examples

To evaluate the approach, several spatial grammars are designed using the prototype system described in the previous chapter. The main aim of the examples is to test and validate the methods presented in Chapter 4. The focus is not on the grammars themselves in terms of their validity or the solutions spaces they create but rather the visual, interactive development and application of rules and the generation of alternative mechanical engineering design solutions including complex geometry.

### 6.1 Vehicle wheel rims

*Parts of the approach evaluated in this example: non-parametric rules (additive, replacing), design space restriction*

The generation of vehicle wheel rims is chosen as a mechanical engineering design example since single piece rims combine mechanical issues, e.g. strength, with the need to be aesthetically pleasing. The rim rules are all non-parametric. The four rules for the generation of the spokes are shown in Figure 6-1. Rule (a) inserts the first spoke in relation to the hub; rule (b) adds a second spoke translated in the x-direction in relation to an existing spoke; rule (c) replaces an existing spoke by two new spokes that are rotated by 45 and -45 degrees respectively; and, rule (d) adds a new spoke rotated 90 degrees in relation to an existing one.

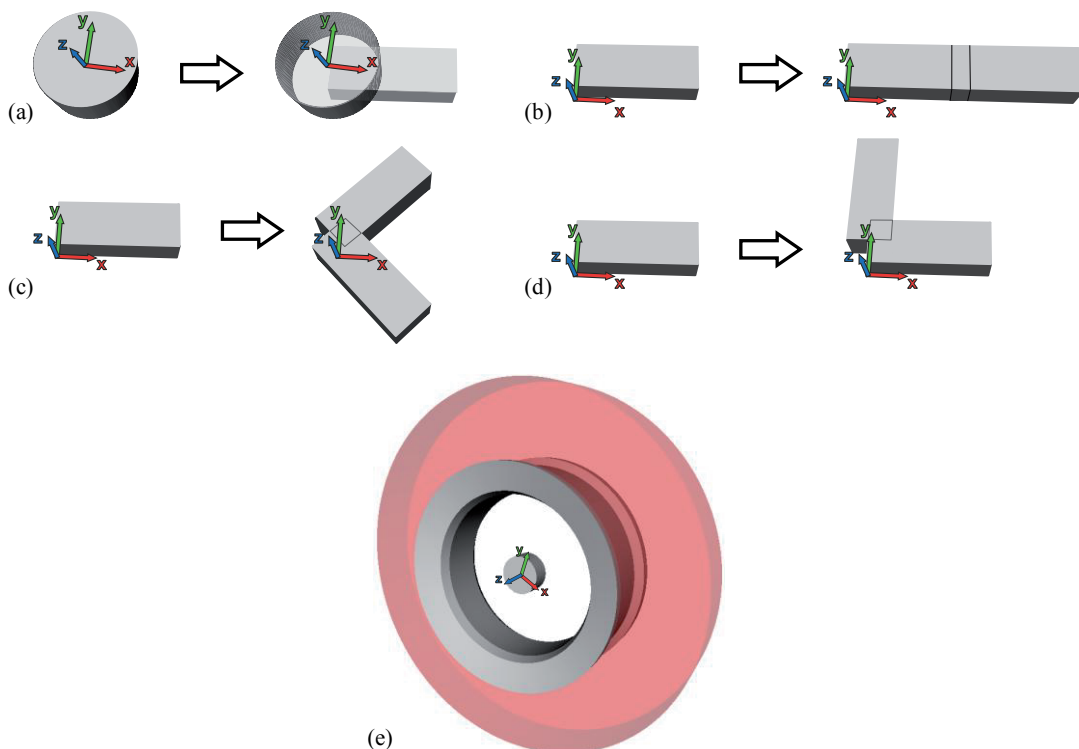


Figure 6-1: Rules for the generation of rim spokes (a)-(d) and the initial set (e)

The rules above allow the single objects to overlap, which can help to generate more unexpected designs. The starting design for the grammar, consisting of the felly and the hub of the rim (Figure 6-1(e)), is designed manually using the standard functionality of the underlying CAD system. In addition to the felly and the hub, the initial set contains a design space restriction that prevents from generating spokes outside the outer diameter of the rim (shown transparently in Figure 6-1(e)). To generate acceptable design solutions, the rules are applied in a semi-automatic mode where the user decides how many rules to apply and in each application step manually chooses one of the found LHS matches. A selection of different created solutions is shown in Figure 6-2. While some of the results were predictable, others were unexpected, e.g. the two on the right hand side.

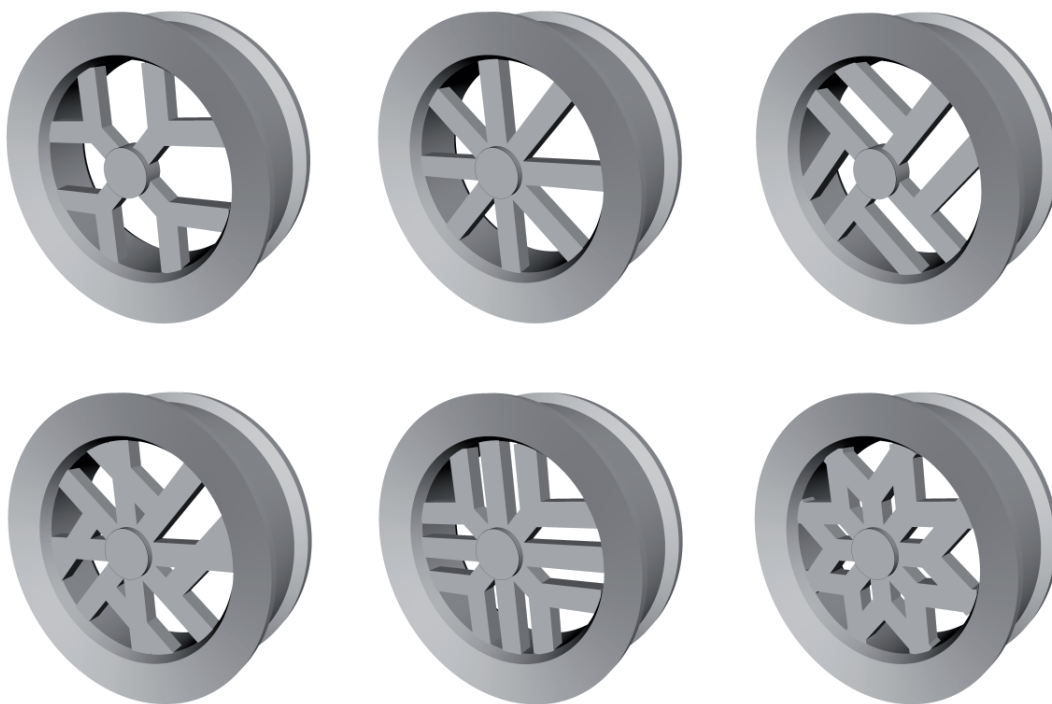


Figure 6-2: Generated rim solutions

The creation of engineering designs is always influenced by many constraints not only in the designs themselves, but also stemming from other domains like customer requirements, manufacturing, costs, laws and standards, etc. Production capabilities have an especially large influence on the geometry of a design. The spokes of customized rims, like the ones generated with the grammar above, are sometimes manufactured on milling machines due to the high flexibility and the low volumes produced. The spokes of the designs shown in Figure 6-2 are all in one plane. The necessity to have the spokes on one plane could be required due to restrictions of the available production capabilities. Extending these capabilities to more complex manufacturable 3D geometry directly influences the range of valid rules that can be defined and, therefore, the resulting solutions. Generally speaking, it is possible to modify rules systematically to define new design languages that reflect changing circumstances (STINY 1980b).

As an example, rules (a) and (d) in Figure 6-1 are modified. In the first case, the spoke is additionally rotated 10 degrees around the y-axis and translated slightly in z-direction (Figure 6-3(a)). In the second case, both spokes are rotated 10 degrees around their y-axes. In a subsequent step, object  $R_0$  is set back to the global origin and, accordingly, the position of  $R_1$  is transformed so that the RHS of the rule is in the correct spatial relation to the reference object in the LHS (Figure 6-3(b)). Figure 6-3(c) shows three examples for rim solutions generated using the modified set of rules.

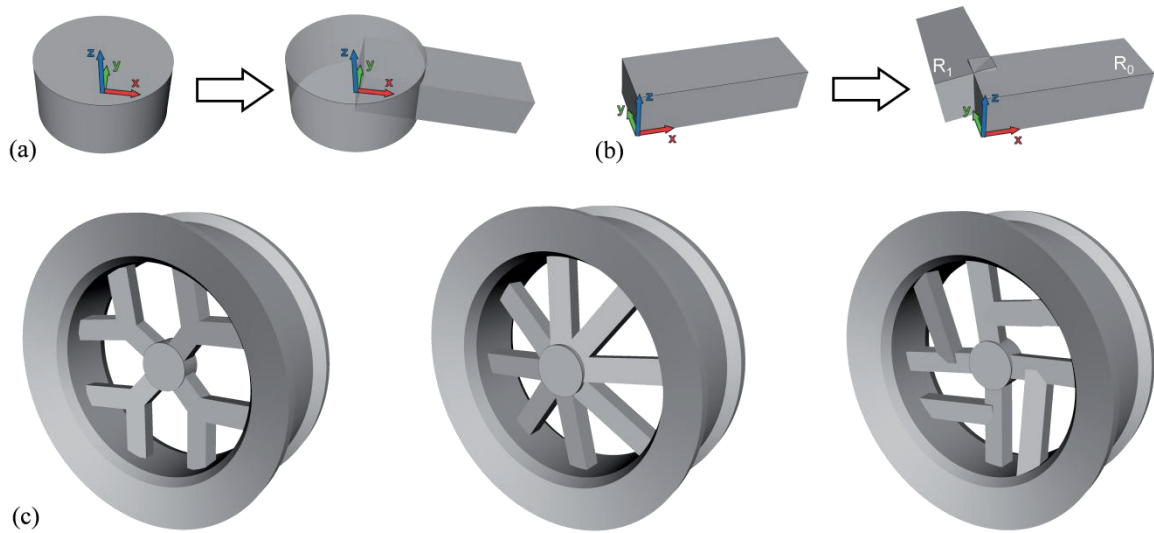


Figure 6-3: Adapted rules and newly generated rim solutions

This example for spatial grammars was previously presented in HOISL & SHEA (2009). However, in previous work the rules were implemented directly as Python scripts, or hard-coded, and restricted to 2.5D geometry. In comparison to using the interactive, visual approach presented here, defining the rules by writing the scripts was rather tedious, as it was often not clear whether the intended rules and effects were realized until their later application. This required many time-consuming test and modification cycles.

## 6.2 Cylinder cooling fins

*Parts of the approach evaluated in this example: parametric rules (additive, subtractive) including parametric relations and ranges with static limits*

As the second example, the generation of cooling fins for the cylinder of motorcycle engines is considered. In comparison to vehicle wheel rims, cooling fins do not have to be aesthetically pleasing but have to fulfill requirements concerning, primarily, the avoidance of overheating, manufacturability and fitting into the available space.

The rules were derived by analyzing the cooling fins of the cylinder of a Kreidler Florett motorcycle and are shown in Figure 6-4. The majority of the rules in this example are parametric. Rule (a) inserts the first two fins, translated in the x- and y-direction in relation to the cylinder; applying rule (b), two new fins are added to an existing one at a distance of 15

mm from each other; rule (c) matches an existing fin of arbitrary length and adds another fin, which is rotated 90 degrees around the z-axis, translated in x-direction by 80% of the length of the matched fin and is assigned a length within the range of 30 to 80 mm; rule (d) finds an existing fin of arbitrary length and adds a new fin whose length is 80% of the length of the matched fin; rule (e) is similar to rule (d) but instead of decreasing the length of the new fin, it increases it by 20%; rule (f) matches an existing fin of arbitrary length and deletes it; rule (g) matches an existing fin of arbitrary length and rotates it around the z-axis by a value within the range of -20 to 20 degrees; and, rule (h) tries to find three fins that are at least 60 mm long and adds a reinforcing fin, which is realized using a cylinder primitive segment rotated 90 degrees around the y-axis and translated along the x-axis by 80% of the average length of the three matched fins. The last rule (h) is intended to help prevent fins from excessive vibrations if they are too long. In rules (c), (d), (e) and (g), the length of the object  $R_0$  in the RHS is set to the same length as the object that is matched to  $L_0$ . This parametric relation is not explicitly shown in the rule figures.

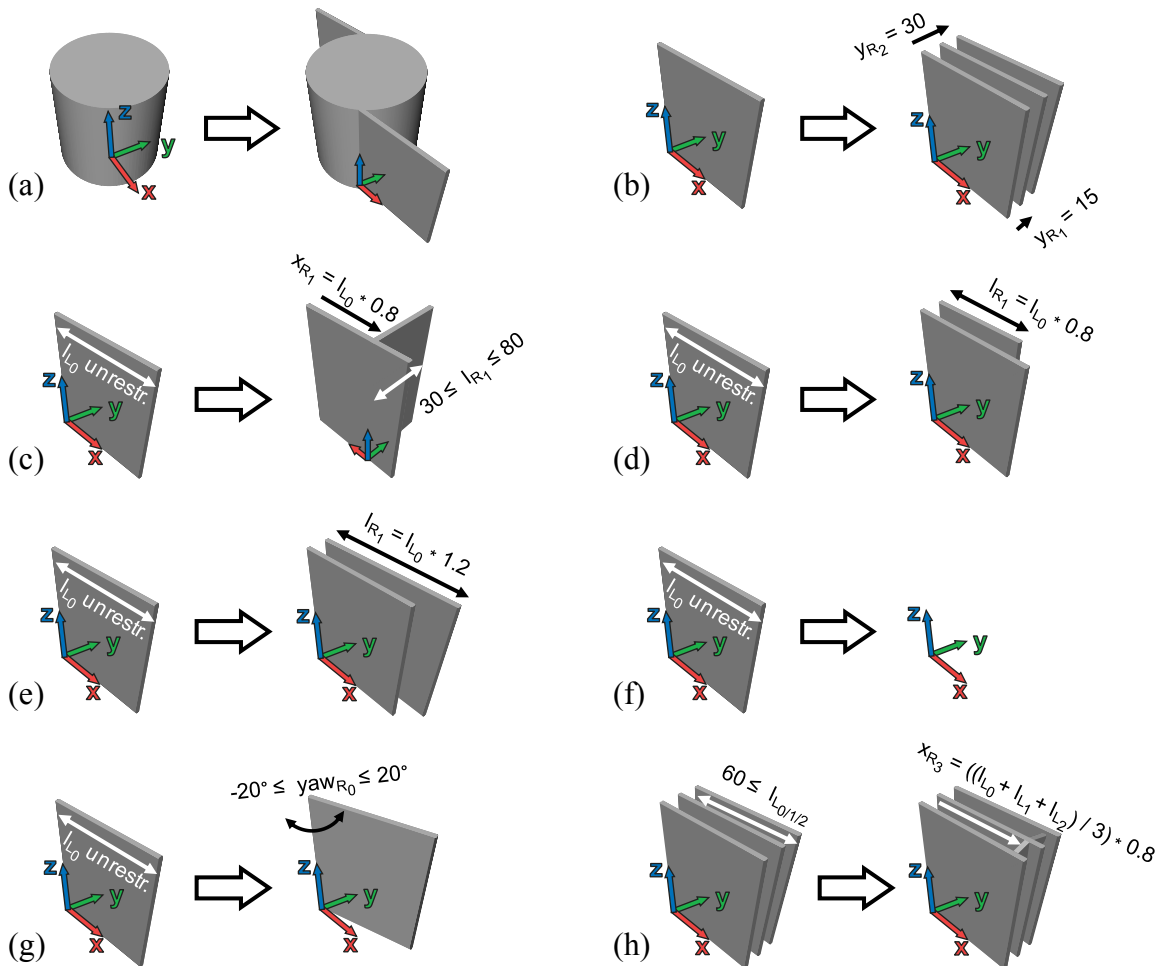
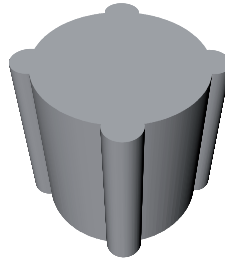


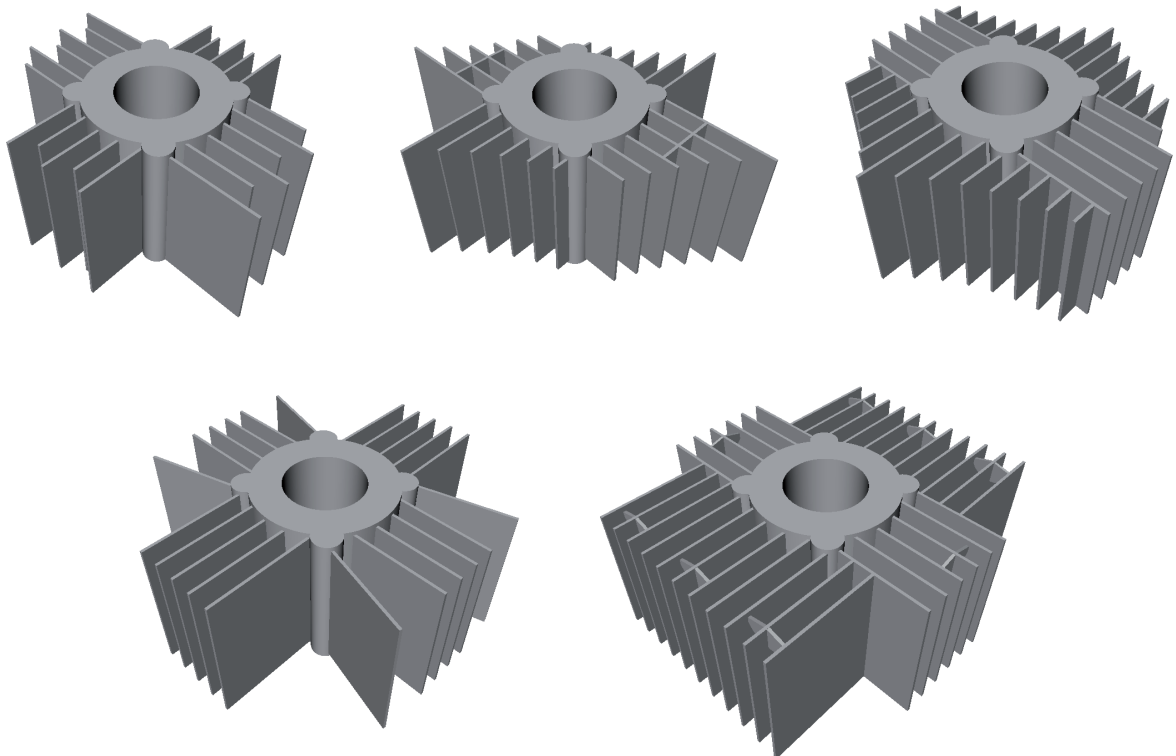
Figure 6-4: Rules for the generation of cooling fins

The initial set is designed manually. It consists of the cylinder, without the bore-hole, and four attached smaller cylinders, which provide the material needed to fix the cylinder head with screws.



*Figure 6-5: Initial set for the cooling fin grammar*

Figure 6-6 shows several examples of solutions generated using the rules above. The rules are applied in a semi-automatic mode to generate meaningful design solutions. The cylinder bore-hole is inserted in a manual step using a Boolean operation after the application of rules is finished.



*Figure 6-6: Several designs generated by applying the cooling fin grammar rules (Figure 6-4)*

### 6.3 High voltage insulators

*Parts of the approach evaluated in this example: swept objects based on parameterized plane primitives using Boolean operations, 3D labels used as spatial and state labels, parametric rules (parametric relations and ranges with parametric limits)*

High voltage insulators are used in a wide variety of applications. The specific example chosen here is derived from the design of centenary insulators for high-speed railways. This kind of insulator has to fulfill several requirements that vary depending on the particular application scenario. The requirements concern both mechanical and electrical engineering and are highly dependent on the geometric design. Examples are the mechanical resistance an insulator has to provide, prevention of electrical arcs and water bridges caused by rain fall, minimization of contamination, or the magnitude of leakage current. The basic element of an insulator is a cylindrical shank that is covered by several sheds. Important geometric parameters influencing the design are the spacing between the single sheds, the diameters of the sheds, their thickness and the shed body angle.

The initial set for this grammar does not contain any geometric objects. Instead it comprises the starting symbol to allow for a wide variety of different insulators to be generated from scratch. The basis is created in the first rule shown in Figure 6-7(a). It replaces the starting symbol with the cylindrical shank object,  $R_0$ , whose height can vary from 200 mm to 400 mm. The radius is also unlocked and restricted to a range between 15 mm and 30 mm. In addition to the shank, the first shed of the insulator,  $R_1$ , is introduced. It is represented by a revolve object whose cross-section is rotated around the z-axis. The cross-section is defined by two parameterized plane primitives (Figure 6-7(b)), whereas  $R_{1b}$  is subtracted from  $R_{1a}$  using a Boolean difference operation. The width,  $w_{R_{1a}}$ , of the first plane represents the thickness of the shed and can be set to values between 10 mm and 20 mm. The length corresponds to the radius of the shed. Its lower limit is defined to be at least 25 mm longer than the radius,  $r_{R_0}$ , of the shank, whereas the upper limit is set to 100 mm. The second plane primitive is rotated by 180 degrees with respect to the first one. Additionally it is translated to the outer corner of the first plane according to the equation  $transX_{R_{1b}} = l_{R_{1a}}$  and placed 3mm higher on the z-axis. The *pitch* angle of the plane can be varied within a defined range of 5 to 15 degrees. This angle correlates to the shed body angle. The last object,  $R_2$ , which is added applying the first rule, is a 3D label (see Figure 6-7(a)). Its purpose will become clear in the description of the remaining two rules of the grammar.

The second rule (Figure 6-7(c)) adds a new shed to the insulator in relation to an already existing shed. The LHS of the rule consists of the shank and a shed. The unlocked parameters of these two objects are the same as the ones shown in Figure 6-7(a) and (b). However, they are all set as unrestricted so that any shank-shed pair of arbitrary dimensions can be matched; for clarity these free parameters are not displayed in Figure 6-7(c). Additionally, the translation of the shed in z-direction is set as unrestricted to be able to match sheds that are positioned anywhere along the shank. This unrestricted translation is accomplished by the definitions  $transZ_{L_{1a}}$  *unrestr.* and  $transZ_{L_{1b}} = transZ_{L_{1a}} + 3$  for the two underlying plane primitives of the shed's cross-section. The only restriction for matching the LHS is given by the 3D label. Its translation on the z-axis has to be equal to the one of the shed determined by the first underlying plane primitive. This is expressed by the equation  $transZ_{L_2} = transZ_{L_{1a}}$ .



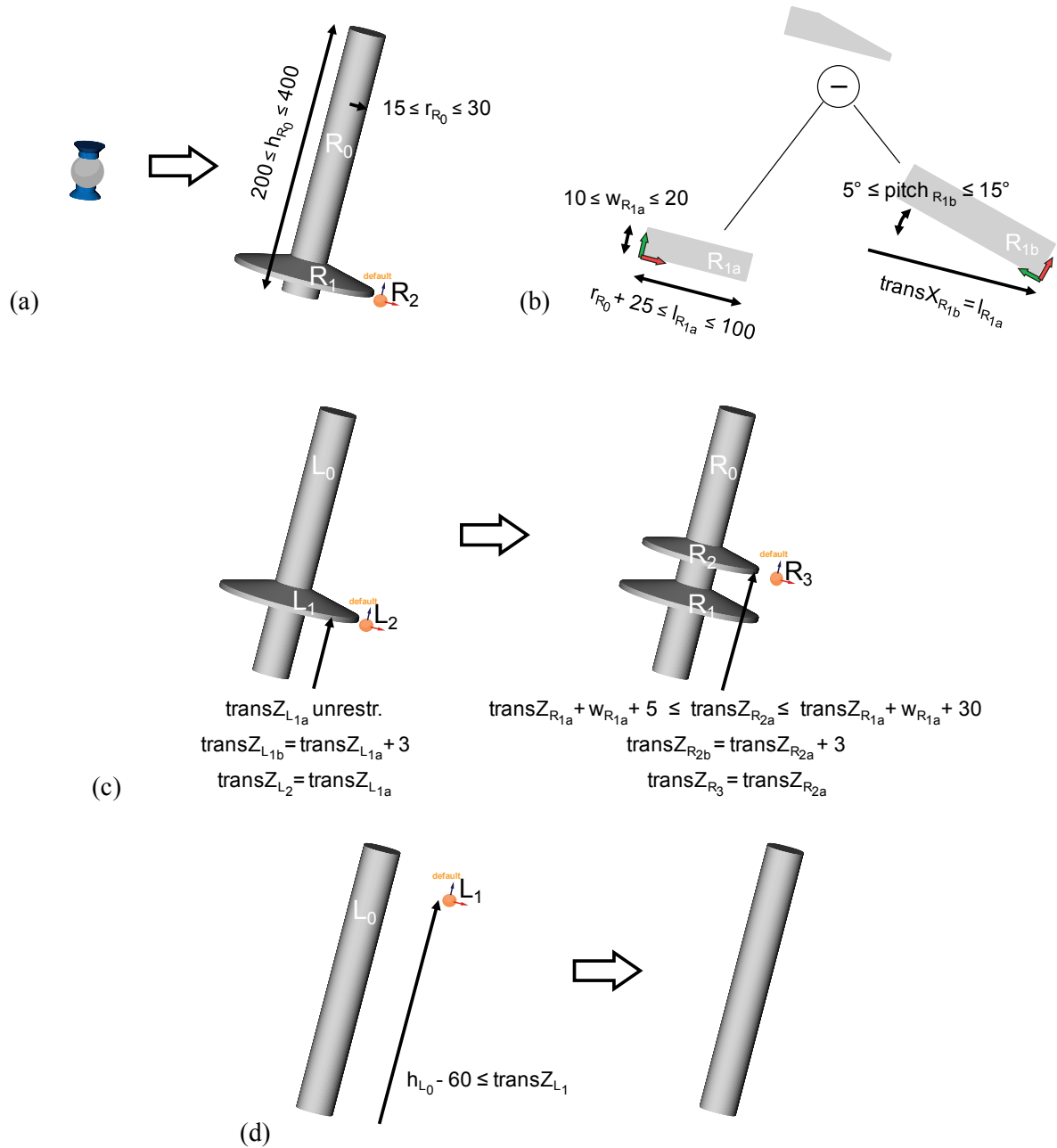


Figure 6-7: Rules for the generation of high voltage insulators

As the rule represents an additive rule, the geometric objects in the LHS are all part of the RHS as well. The same free parameters as the ones shown in Figure 6-7(a) and (b) are defined again for the shank,  $R_0$ , and the shed,  $R_1$ , and are set equal to the free parameters of the LHS's objects; again, these free parameters are not shown in the figure for clarity. Additionally, a new shed,  $R_2$ , is added to the insulator. It is positioned at least 5mm above the top of the matched shed,  $\text{trans}Z_{R_{1a}} + w_{R_{1a}} + 5 \leq \text{trans}Z_{R_{2a}}$ , and to the most 30mm above it. The last object introduced is the 3D label,  $R_3$ . It is allocated to the shed,  $R_2$ , via its translation in z-direction using the parametric relation  $\text{trans}Z_{R_3} = \text{trans}Z_{R_{2a}}$ . The 3D label used in both sides of the rule

acts as a spatial label, as it always restricts the application of the rule to the most recently added shed.

The last rule in Figure 6-7(d) is an abstract rule that does not change the geometry of the insulator. Instead it checks the position of the 3D label and, therefore, implicitly the position of the most recently added shed in relation to the shank. If it is located within 60mm from the top of the shank, a match of the LHS can be detected. The RHS of the rule keeps the shank but removes the 3D label with the effect that no further rules can be applied and the generation procedure terminates. Therefore, the 3D label also has state label character. The rule prevents sheds from being inserted in positions that are higher than the maximum height of the shank, i.e. without a ‘physical’ connection to the shank.

The automatic generation of solutions always starts with the application of the first rule that can only be applied once. Next, rules two and three are applied interchangeably. Rule two inserts a new shed every time it is applied. Matching the LHS of the third rule fails as long as the 3D label is not in the specified range. Once it is applicable, it results in the end of the generation process. For some of the solutions the values that have to be assigned to free parameters are randomly selected by the system, for others they are manually defined. Figure 6-8 shows several insulator solutions generated using the presented rules. The joints are manually added after the generation.

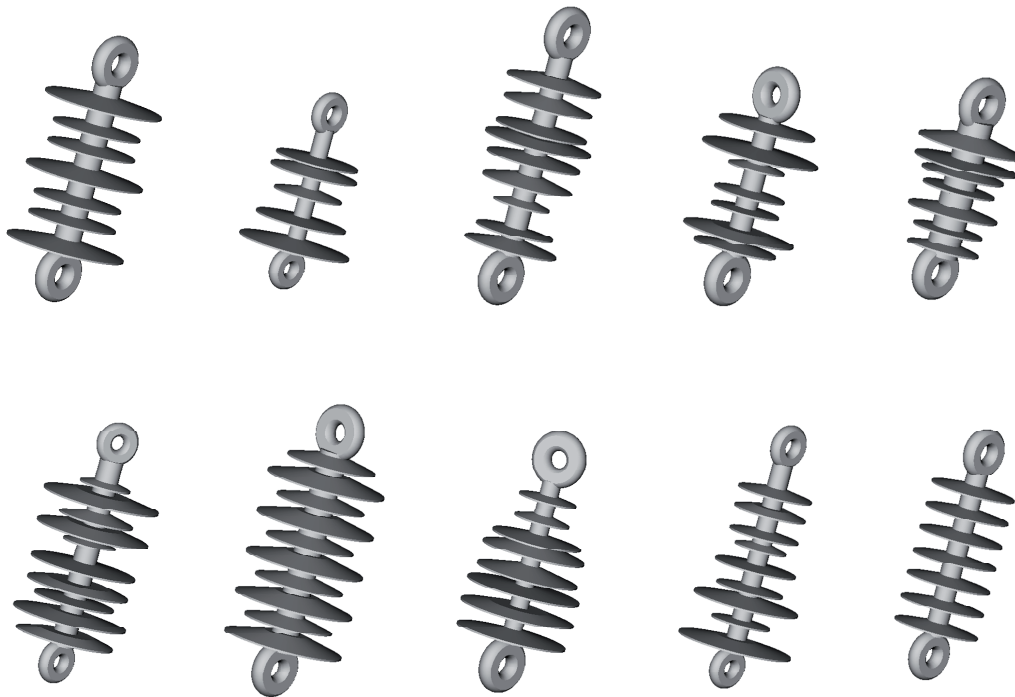


Figure 6-8: Several examples for generated insulator solutions

## 6.4 Robot arm concepts

*Parts of the approach evaluated in this example: simplified LHS matching using 3D labels and parametric segments, general uses for 3D labels*

The following spatial grammar example has a focus on simplified LHS matching using the 3D label concept described in Section 4.4. The example is able to generate different complex geometry solutions for robot arm concepts. A robot usually consists of several robot arms that can be rotated around different axes. These robot arms have to be designed in a way that they fulfill requirements concerning stability, weight, design space, etc. Here, the application is approached only conceptually as an example for complex 3D design generation in a mechanical engineering related area. The grammar is intended to illustrate and validate the 3D label approach rather than validate the grammar itself, i.e. in terms of the design space defined. Figure 6-9 shows the spatial grammar developed based on the approach of using three-dimensional labels for simplified LHS matching and different segments of robot arms. The rules were derived by analyzing existing industrial robots<sup>23</sup>.

Rule (a) is the first rule to be applied. It uses the starting symbol in the LHS to start from no geometry and adds a segment consisting of a parametric joint as well as the first part of the robot arm with a specified interface. Additionally a 'default' 3D label is defined in the RHS so that further rules can be applied. The radius,  $r$ , is defined to be within a range of 30 to 40 mm, so that the diameter of the joint can be adapted, e.g. depending on the load the arm has to resist. The translation,  $translateY$ , of the joint depicts the offset of the joint's centre from the label and is set depending on the radius. Rules (b)-(e) and (h) can be applied after this first rule because they all have a 'default' label in the LHS. The RHS of rule (b) consists of a bridge-segment that extends the length of the robot arm. It is based on an extrusion in the y-direction,  $dirY_b$ , which is restricted to a range between 30 and 100 mm. Additionally, it contains a stiffener that is positioned in the middle of the extruded part ( $translateY_s$ ) and can be rotated within a range of  $\pm 15^\circ$  around the x-axis ( $roll_s$ ) depending on the main direction of the force application on the robot arm. A second stiffener on the backside of the part is positioned according to the first one. Rule (c) adds another bridge-segment that includes a narrowed cross-section in the middle. It can help in cases where the weight of the robot arm has to be reduced or the load on the arm is not high. The length of this middle part of the segment is parametric and restricted by a range. Rule (d) allows for extrusions in three different directions,  $dirX$ ,  $dirY$  and  $dirZ$  that are all restricted to certain ranges. In cases where  $dirX$  and  $dirZ$  are zero, the rule is identical to rule (a) except with the addition of stiffeners. Application of the rule results in a deviation of the main robot arm direction from the y-axis. This can be needed in cases where the available space for the robot arm is restricted, for example to avoid collisions with other obstacles.

---

<sup>23</sup> See <http://www.fanucrobotics.de/> (accessed January 14<sup>th</sup> 2012)

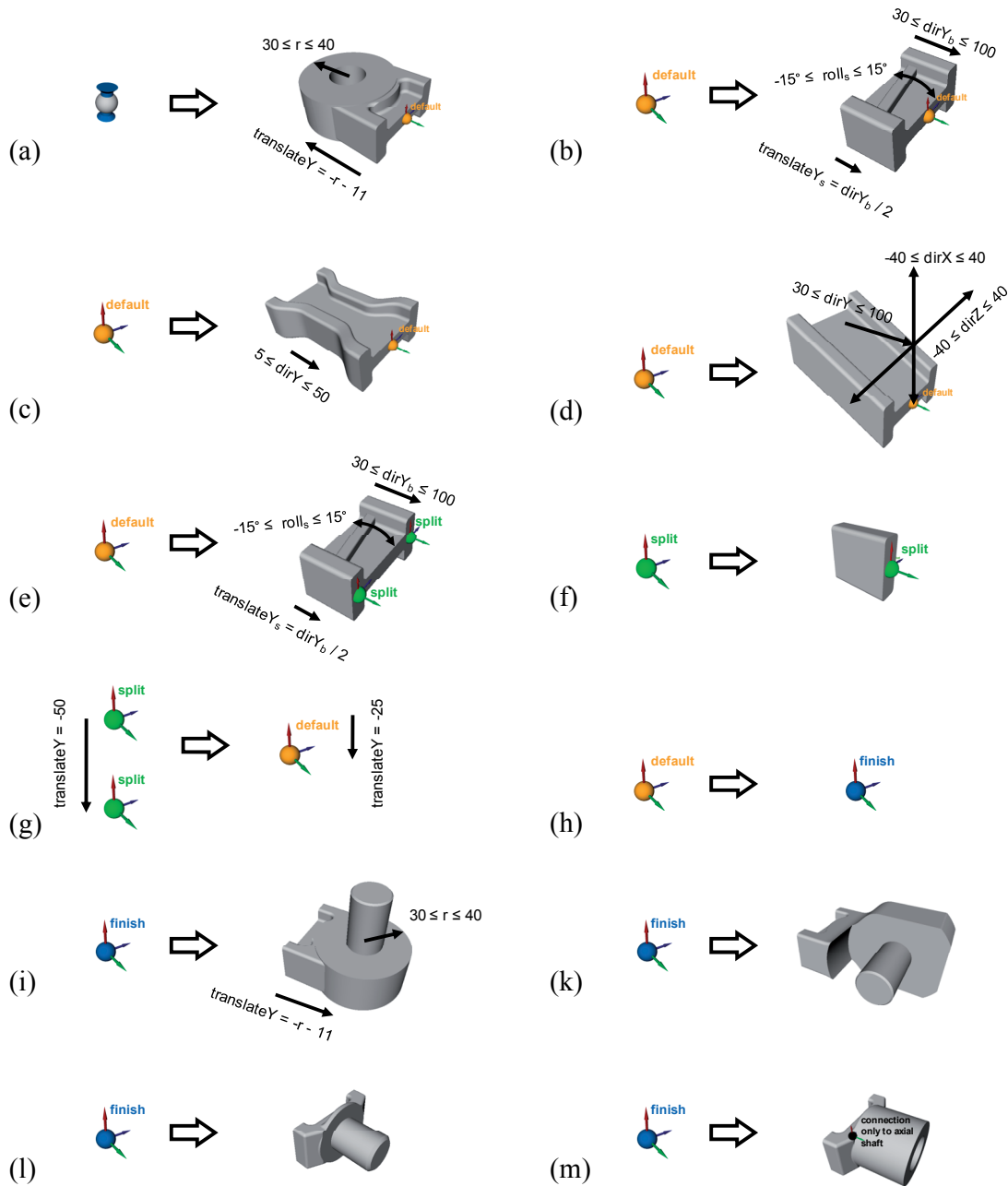


Figure 6-9: Spatial grammar for the generation of robot arm concepts

Rule (e) is geometrically and parametrically identical to rule (b). The difference is that it introduces a new kind of label that carries a different color and the name ‘split’ and therefore also changes the state. Additionally, not only one single label, but two of these labels are defined in the RHS of this rule. After applying this rule, none of the previously described rules or rule (h) can be applied, because the ‘default’ label in the LHSs, i.e. its name and color, can no longer be matched. Instead only the two rules (f) and (g) are applicable. The first one adds a smaller, non-parametric segment with a different interface, which nonetheless geometrically fits the interface used in the other rules. Application of this rule results in the

creation of a cutout in the robot arm. This can help to reduce weight or create space for attachment parts or electrical motors. Repeated application of this rule creates longer holes. When rule (g) is applied, the generation of a hole is terminated. Rule (g) is an abstract rule because it does not result in any geometrical changes, but instead substitutes two ‘split’ labels with a distance of 50 mm to each other by one ‘default’ label that is positioned in the center of the locations of the two ‘split’ labels. After rule (g) is applied, the rules with ‘default’ labels in the LHS can be applied again. To end the generation of a robot arm, a further abstract rule (h) is defined. It substitutes a ‘default’ label by a ‘finish’ label, again changing the state but also specifying the transformation for the next segment. The ‘finish’ label is a state label that restricts further application to one of the rules (i)-(m). Additionally, no 3D label exists in the RHSs of these rules, so that the generation process terminates after their application. Rule (i) finishes a robot arm with a joint that is the counterpart to the joint added by rule (a). Instead of a hole, it consists of a shaft in the centre. Similarly, rule (k) consists of a shaft with an axis twisted by  $90^\circ$  in comparison to the start joint in rule (a) to allow for the generation of robot arms with different kinematic characteristics. Rule (l) also adds a shaft but in this case axially aligned to the longitudinal direction of the robot arm. The last rule (m) inserts a joint with a hole instead of a shaft in the axial direction of the robot arm. It has a 3D label in the RHS carrying the non-geometric information that it should only be assembled to a particular axial shaft, i.e. only to the joint inserted by rule (l) and not the ones introduced by (i) or (k). The combination of these two parts realizes a joint that can rotate a robot arm around its own longitudinal axis. This is often needed to position the gripper on the top of the robot to reach an intended location. Figure 6-10 shows a selection of different robot arm designs automatically generated using the rules defined above.

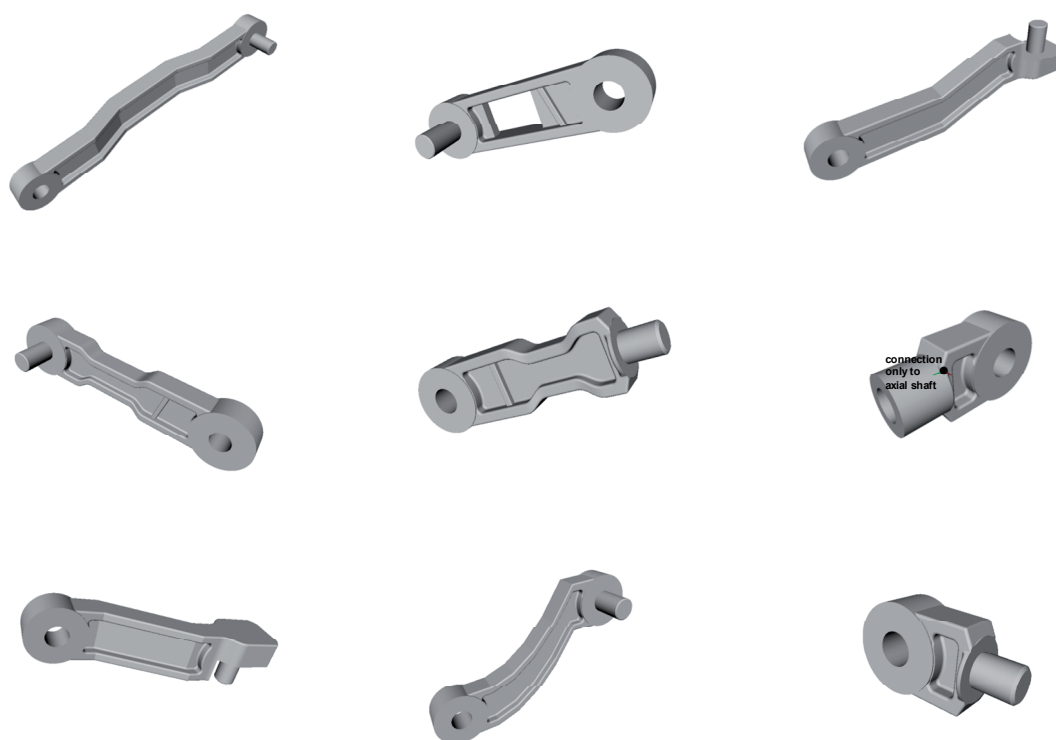


Figure 6-10: Several solutions for robot arms generated using the rules shown in Figure 6-9

A few examples for manually assembled robot arms are illustrated in Figure 6-11. The rotary base and the gripper are always identical and therefore designed manually and inserted after generation.

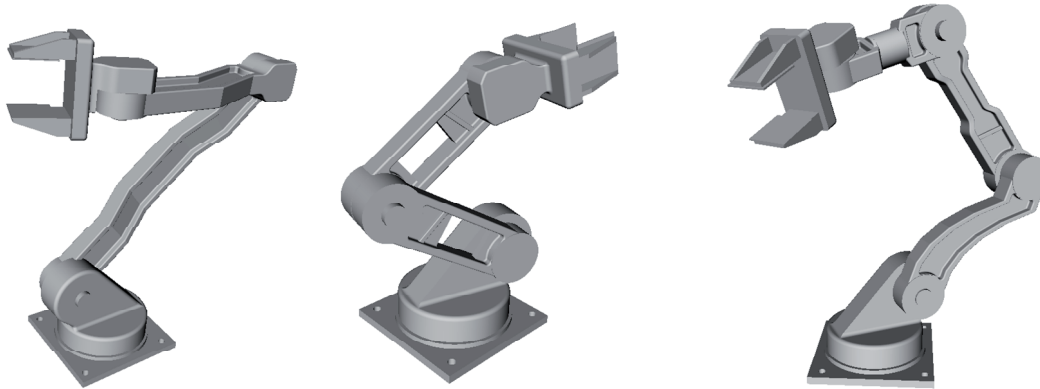


Figure 6-11: Examples for robot arms assembled using the concepts shown in Figure 6-10

## 6.5 Gear systems

*Parts of the approach evaluated in this example: Boolean operations, part collision avoidance, parametric rules (parametric relations and ranges with parametric limits) including advanced mathematical functions*

Gear systems exist in different forms in a variety of products. STARLING & SHEA (2002) presented gear systems as an example for using grammar-based synthesis to generate gear systems in clocks, power drills, winding mechanisms in photo cameras and automotive gear boxes. Based on this approach LIN et al. (2009) developed a refined method for automated gearbox synthesis incorporating simulated annealing. Gear systems have to fulfill several requirements concerning, for example, weight, available space or the number of output shafts as well as the related speed and torque. Further, the design of a gear system must include many constraints to ensure its validity. For example, it has to be ensured that gear pairs interact with each other but they must not overlap or a newly inserted gear has to sit on a shaft and both have to be coaxial. The manual design of gear systems is a tedious and time consuming task. Due to the numerous constraints often only very few alternative designs can be created and the best solution(s) might not be found. Therefore, gear systems are a suitable example for the use of a grammar approach, especially in early design phases where it is sufficient to use a simplified representation of gear disks in the form of plain cylinders (cf. STARLING & SHEA 2002).

Unlike the original system by STARLING & SHEA (2002) that additionally included an analysis of the performance of the created gear systems, the example presented here focuses on the synthesis of the geometry of gear system solutions. The rules for the original examples are all hard-coded, making them difficult to change, whereas the prototype spatial grammar system here allows for the flexible, visual definition of the gear rules.

In comparison to the grammar examples presented so far in this chapter, which all generate solutions consisting of a single part, this example acts on the assembly level treating the different geometric objects as separate parts that are not allowed to overlap. This requires the collision detection (cf. 4.5.1) to be activated for the application of the rules.

The initial set of the grammar consists of the starting symbol to allow for a greater variety of solutions. The first rule (Figure 6-12) inserts the first shaft of the gear system that can have a height,  $h_{R_0}$ , of 100 to 350 mm and decisively influences the outer measurements of the gear system in z-direction. The radius of the shaft,  $r_{R_0}$ , can be set within a range of 10 to 20 mm.

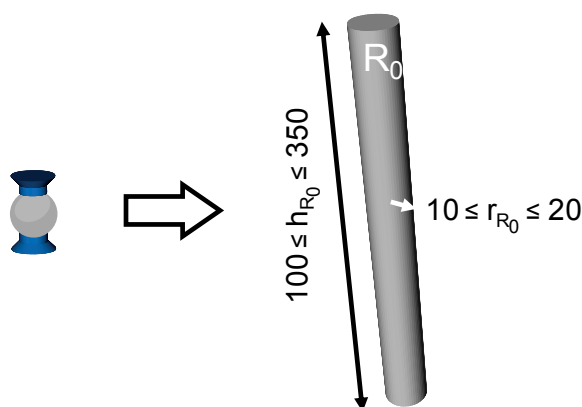


Figure 6-12: First rule for the generation of gear systems

The second rule (Figure 6-13(a)) adds a further shaft to an already existing one and inserts an interacting pair of gear disks. The height and radius parameters of the shaft in the LHS are set as unrestricted to be able to detect any shaft. The two shafts in the RHS,  $R_0$  and  $R_1$ , are defined to be equal to the values of height and radius in the LHS; for clarity these parameters are not shown in the figure.  $R_1$  is additionally translated within a range of -300 to 300 mm in x- and y-direction in relation to the position of  $R_0$ . The rule further inserts a pair of gear disks to connect the two shafts with each other. To avoid the detection of collisions between the gear disks and the shafts they are allocated to, the disks must not consist of a cylinder primitive only but have to contain a hole in the center. Consequently, they are created based on a Boolean difference operation of two cylinders (Figure 6-13(b)). The free parameters are defined on the basis of the underlying primitives. The first cylinder,  $R_{2a}$ , can have a height between 15 and 50 mm to be able to represent gear disks of different thicknesses. The radius,  $r_{R_{2a}}$ , of the cylinder is defined to be at least 3 mm larger than the radius of the shaft,  $r_{R_0}$ , it is attached to. The maximum limit is set to 200 mm. To ensure that the cylinder is positioned on the shaft, i.e. not above or under it, its translation in the z-direction is restricted to positive values with a maximum of the shaft's height minus its own height,  $0 \leq \text{trans}Z_{R_{2a}} \leq h_{R_0} - h_{R_{2a}}$  (see RHS of the rule in Figure 6-13(a)).

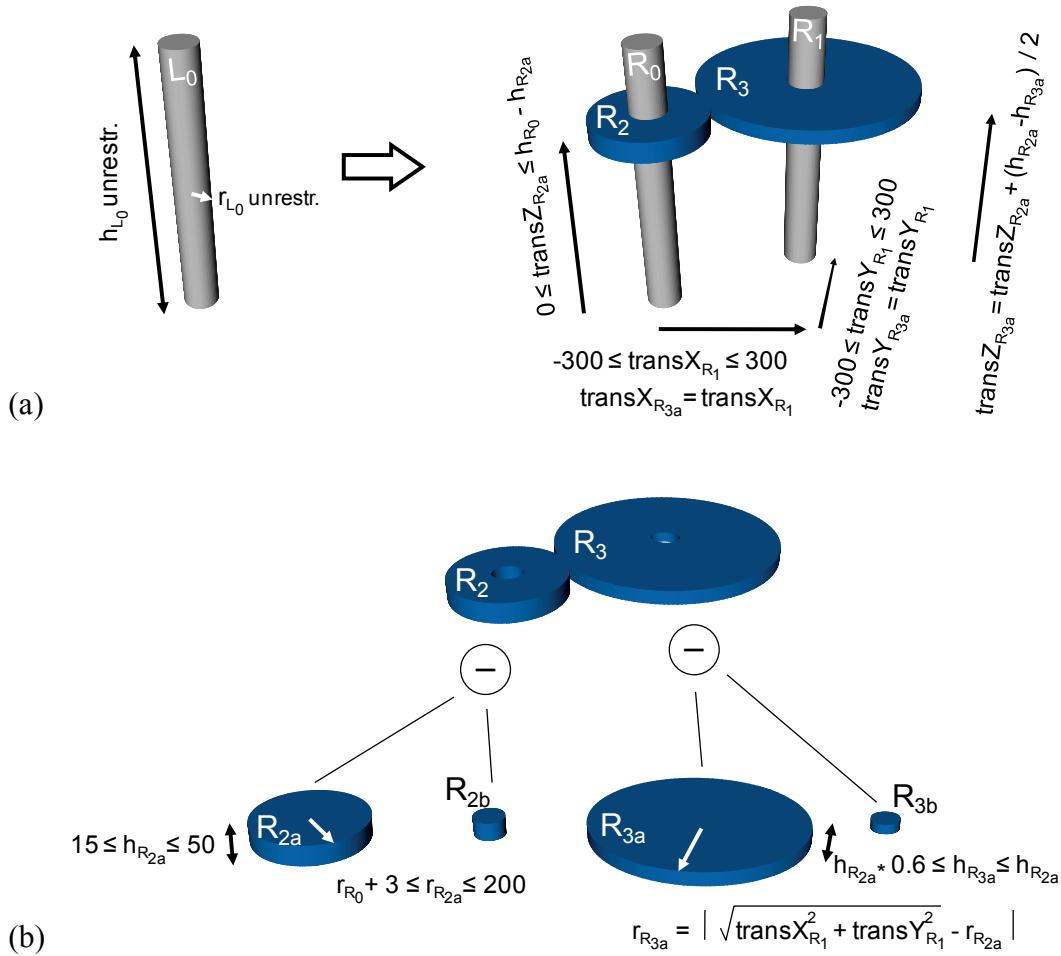


Figure 6-13: Second rule for the generation of gear systems

To position the second disk correctly, the translation in the x- and y-direction of its underlying cylinder,  $R_{3a}$ , is set to be equal to the translation of the shaft  $R_1$ . The height,  $h_{R_{3a}}$ , of the cylinder is defined within a range from 60% of the height of the first disk (Figure 6-13(b)). The translation in z-direction is calculated so that it is ‘centered’ with respect to the first disk according to the equation  $transZ_{R_{3a}} = transZ_{R_{2a}} + (h_{R_{2a}} - h_{R_{3a}}) / 2$ . To ensure that the two disks interact, the radius of the underlying cylinder of the second disk has to be equal to the absolute value of the distance between the two shafts minus the radius of the first disk:

$$r_{R_{3a}} = \left| \sqrt{transX_{R_1}^2 + transY_{R_1}^2} - r_{R_{2a}} \right|$$

The cylinders  $R_{2b}$  and  $R_{3b}$  are subtracted from the other two cylinders to create the holes in the center of the disks. The heights as well as the translations in z-direction are set to be equal to the heights and the translations of the cylinders  $R_{2a}$  and  $R_{3a}$ , the radii are set to be equal to the radii of the related shafts.  $R_{3b}$  is additionally translated in the x- and y-direction in accordance with the translation of the shaft  $R_1$ . For clarity, all of these parameters are not shown in the figure above.



A further rule is defined that is similar to the previous one. The main difference is that the LHS does not only contain a shaft but also a gear disk (Figure 6-14). The effect of applying the rule in comparison to the previous one is that no additional disk is inserted to the detected shaft, but the existing disk is connected to a new disk including a new shaft.

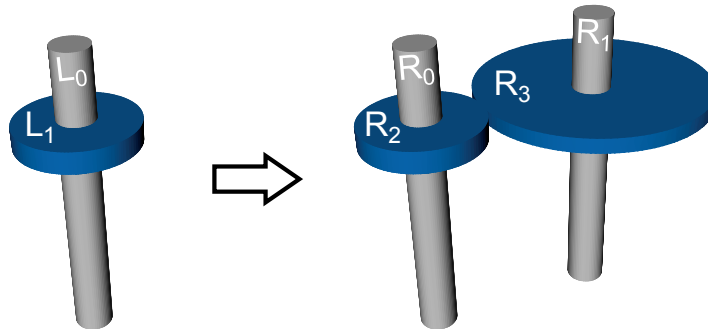


Figure 6-14: Third rule for the generation of gear systems

The disk,  $L_1$ , is generated the same way as the gear disks in Figure 6-13(b). To be able to detect any gear disk, the height, radius and transZ of the two underlying cylinders,  $L_{1a}$  and  $L_{1b}$ , are all set as unrestricted. The same applies to the height and radius parameters of the shaft  $L_0$ . As it is an additive rule, the values for these parameters are taken over for the geometric primitives  $R_0$ ,  $R_{2a}$  and  $R_{2b}$  in the RHS. All remaining parameters are defined in exactly the same way as in the rule described above (Figure 6-13).

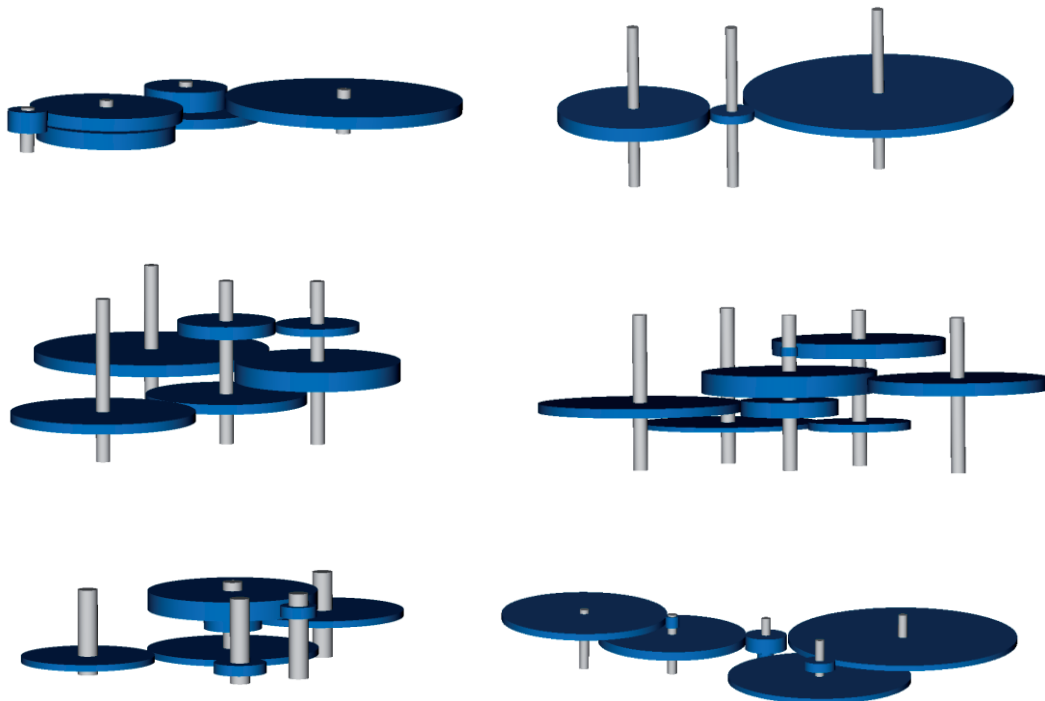


Figure 6-15: Solutions for gear systems generated using the rules shown in Figure 6-12 to Figure 6-14

Figure 6-15 shows several solutions generated automatically using the three rules and the starting symbol as the initial set. Any values that have to be assigned to free parameters are randomly selected by the system. The number of rule applications is manually chosen to generate solutions with different numbers of shafts. Some solutions, e.g. the first one, result in very flat designs. The second example is made of a simple chain of disks. Others, for example the fourth solution, contain a ‘central’ shaft that has several disks attached to serve several output shafts at the same time.

The rules defined so far as well as the ones originally used by STARLING & SHEA (2002) use parallel shafts only. With the visual approach developed in this thesis, it is easy to define rules that include shafts and gear disks whose axes are rotated with respect to each other, resulting in entirely three-dimensional gear systems.

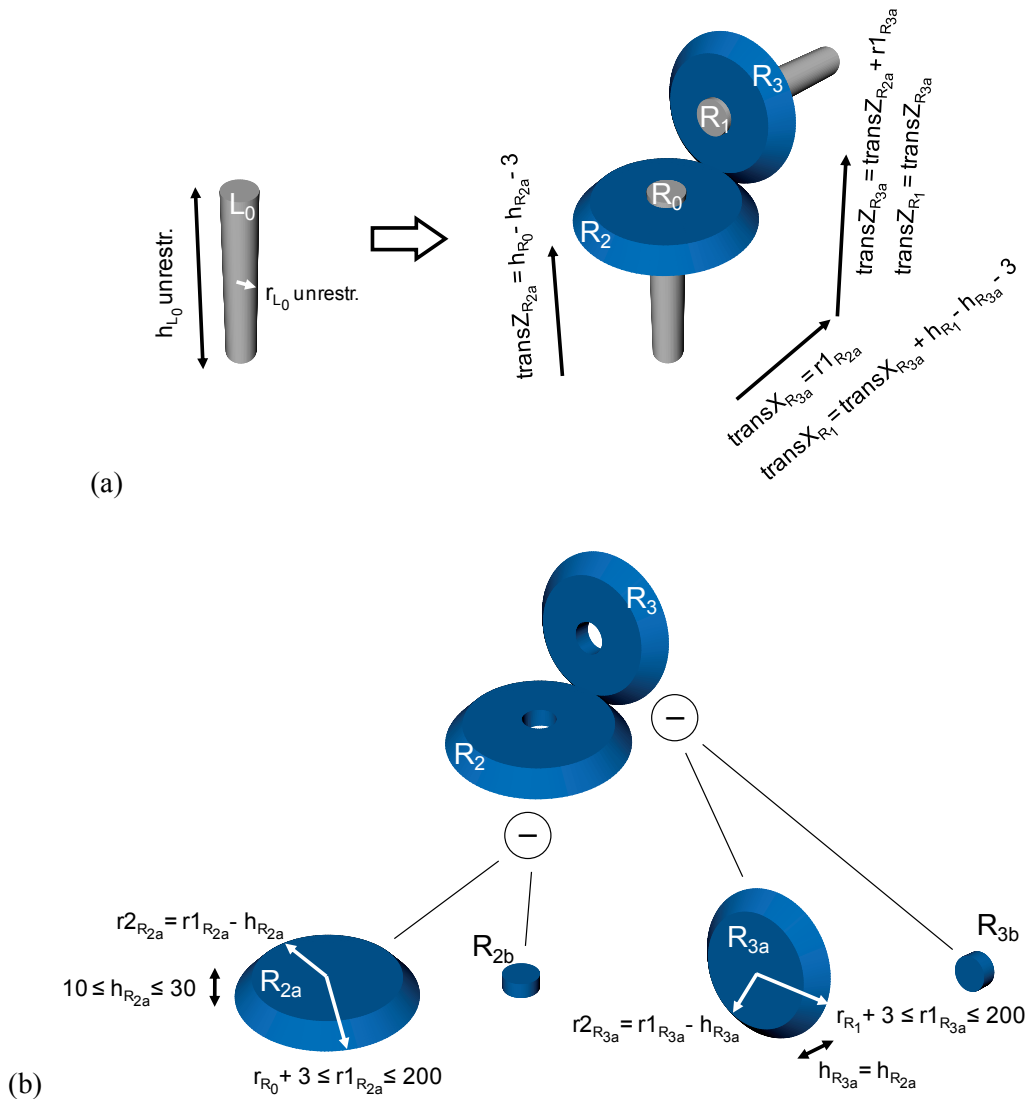


Figure 6-16: Rule for the creation of a pair of bevel gears

An example for such a rule is shown in Figure 6-16. In addition, it introduces cones as a further type of primitive to be able to represent bevel gears. The LHS of the rule contains a shaft with unrestricted height and radius. In the RHS this shaft is recreated assigning the same values to the height and radius of the cylinder  $R_0$ . The first bevel gear is added to this shaft. Like the other gears, it is created using a Boolean difference operation (Figure 6-16(b)). The gear's height,  $h_{R_{2a}}$ , can range between 10 and 30 mm. Its lower radius,  $r_{l_{R_{2a}}}$ , has to be at least 3 mm larger than the radius of the shaft and can be set to a maximum of 200 mm. The operating angle of the bevel gear in this example is chosen to be 45 degrees, achieved by defining the upper radius  $r_{2_{R_{2a}}} = r_{l_{R_{2a}}} - h_{R_{2a}}$ . The gear is positioned slightly below the upper top of the shaft using the equation  $transZ_{R_{2a}} = h_{R_0} - h_{R_{2a}} - 3$ . The second bevel gear is inserted rotated by 90 degrees around the y-axis. Its upper and lower radii are defined in the same way as described for the first gear; its height is set to be equal to the height of the first gear. To establish the correct interaction between the two gear disks, the second disk is translated by the value of the lower radius of the first disk,  $r_{l_{R_{2a}}}$ , in the x-direction and by the position of the first disk plus its own lower radius,  $transZ_{R_{2a}} + r_{l_{R_{3a}}}$ , in the z-direction.

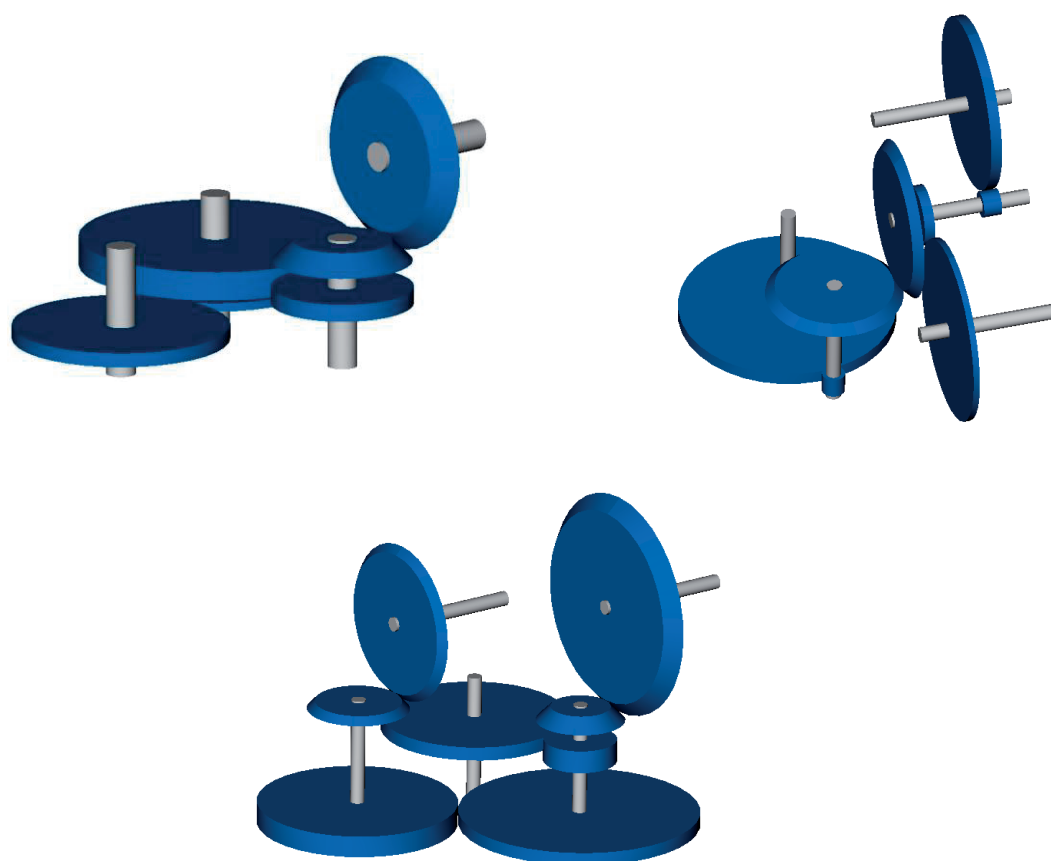


Figure 6-17: Solutions for gear systems including bevel gears

The last new object added is the second shaft,  $R_1$ , which is also rotated 90 degrees around the y-axis. The translation in the z-direction has to be equal to the translation of the second bevel gear. To create the correct spatial relation between gear and shaft, the shaft is additionally translated in the x-direction according to the equation  $transX_{R_1} = transX_{R_{3a}} + h_{R_1} - h_{R_{3a}} - 3$ .

Heights and translations of the two cylinders,  $R_{2b}$  and  $R_{3b}$ , that are subtracted from the cones,  $R_{2a}$  and  $R_{3a}$ , are set to be equal to the heights and translations of the cone parameters; the radii of the two cylinders are set to be equal to the radii of the shafts  $R_0$  and  $R_1$ . Figure 6-17 shows three gear systems solutions including bevel gears generated automatically using the rules presented in this example.

## 6.6 Summary

Five different examples for mechanical engineering grammars were developed to evaluate the approach and the prototype system. The examples cover all parts of the spatial grammar approach presented in this thesis. The rules were described in this chapter and different solutions generated by each of the grammars were shown. The system proved to be a general platform that allows for the development of different grammars without programming.

The rim and the cooling fin grammars were used to test the basics of the approach and semi-automatic rule application. They are based on rather unrestricted rules. This allows for a wide range of solutions to be generated. However, they can also generate invalid solutions, for example, if none of the spokes in a rim is connected to the felly or if cooling fins are created that are not connected to the cylinder or any other cooling fins. For that reason the grammar rules were applied in a semi-automatic mode. Introduction of further constraints, e.g. using labels, or the definition of further rules could help to ensure valid solutions if the rule application is performed automatically. While the application of the remaining three grammars results in geometrically valid solutions, the meaningfulness of the solutions must be further verified in terms of mechanical engineering requirements.

## 7. Discussion and future work

The approach for a spatial grammar interpreter presented in this thesis comprises different aspects of the visual, interactive development and application of three-dimensional spatial grammar rules. In the following the contributions of this approach to the research area of spatial grammars are summarized. In addition, its limitations as well as ideas for improvements and future work are discussed.

### 7.1 Research contributions

Integrated into a single system, the approach for the presented 3D spatial grammar interpreter has several advantages over existing 3D implementations:

- the possibility to visually, i.e. without programming, define and modify three-dimensional rules of
  - arbitrary rule format, i.e. additive, subtractive and replacing rules, which can
  - make use of a wide range of solid, parameterized primitives including Boolean and sweeping operations in rule definition;
- development and application of both non-parametric and parametric rules;
- definition of an unrestricted number of rules, shapes in rules and applications of rules;
- a concept for the consolidated use of labels;
- automatic matching of the LHS of a rule in a CWS, including checking of parametric relations
- the interactive application of rules (automatic, semi-automatic, manual) in combination with adhering to defined parametric relations;
- integration into a CAD system including a user-friendly user interface;
- and, collision detection for the restriction of the design space and to avoid collisions of parts.

Combining all features, the approach provides a general platform for the development and application of spatial grammars and is not restricted to or implemented for one specific example. Integrated in a CAD system, it helps to make CAD a more active design partner through computational design synthesis. With regard to engineering design, this work is a step towards supporting engineers in formalizing their knowledge about design while they work in a familiar software environment, i.e. a CAD tool. Even when using automated design generation, the approach does not intend to replace the designer but rather to support time-consuming, tedious design tasks and the generation of a wider variety of alternative design solutions. While the approach is limited to the use of a set grammar formulation of spatial grammars, a wide range of designs can be rapidly generated yielding some creative and unexpected outcomes.

The approach is not a substitution for but an enhancement of CAD systems. The prototype is implemented as a plug-in providing an additional, specialized workbench within a CAD system. It can be used in cases that are amenable to grammatical generation, for example, for designs that contain several recurring subparts or patterns. After the grammatical generation process is finished, the designer is able to change to other CAD workbenches to go on working with the created solutions.

## 7.2 Limitations and potential for improvement

Despite the advantages the presented approach provides, the method and implementation have some limitations. These and potential for improvement are discussed in the following.

### *Transformations and LHS matching*

Unlike the original grammar formalism, the current approach does not consider scaling and reflection transformations for matching the LHS of a rule. The use of scaling transformations can be problematic in combination with parametric rules. This is especially the case when parametric relations between objects are defined that include mathematical operators and static values since these would not be ‘scaled’ in the equations in accordance to the geometry. Besides, parametric rules as described in this thesis can also be used to recognize scaled versions of an object (see 4.2.2). This allows the user to explicitly decide whether scaled versions should be matched or not.

The approach for matching the LHS relies on the local coordinate systems of the geometric objects defined in the rules. This makes it easy to determine the transformation under which a rule applies to the CWS. However, it also prevents from matching shapes under several symmetry transformations, especially concerning reflections. This is due to the fact that the local coordinate system of each shape acts as an implicit label that unambiguously determines the transformation for the rule application. In the broader context, this is related to the issue of symmetries of objects, the automatic detection of symmetry- or reflection planes, as well as the question of how to deal with rotationally symmetric objects, as they match under an infinite number of reflection or rotation transformations. Symmetries help to generate a wider variety of solutions with only one rule, where labels are often used to restrict the application to a few specific symmetries. Due to the neglecting symmetries, with the current approach it is sometimes necessary to define more rules to be able to generate certain design solutions. From a usability point of view, defining a rule for every single symmetry case is often more transparent, since spatial thinking can be very challenging with rules that rely on symmetries, especially in 3D.

Taking a longer term perspective, alternative possibilities for resolving the LHS matching problem need to be investigated. The current approach can automatically match parametric rules based on primitives and Boolean or simple swept objects. On this basis, a wide range of geometries can be created. Further, 3D labels allow circumventing the matching problem to generate complex geometries. However, using the 3D label concept, only rules that add geometry to a CWS can be defined. Once a segment is added to the CWS, there is no possibility to replace it anymore, unless parts of it are based on parameterized primitives or objects based on Boolean or simple sweeping operations. Complex geometry in the LHS

cannot be matched so far and even for simple geometry the matching can fail if the ‘internal’ structure of two objects is not identical. For example, an L-shaped block could be designed using sweeping, Boolean difference or Boolean union operations. Even if the resulting geometry is visually identical in all three cases, the objects are built in different ways and can therefore not be matched with each other.

In the generalization of the approach, it may be promising to investigate the incorporation of more general existing three-dimensional shape searching methods that are based on the comparison of, for example, voxels and boundary representations (see e.g. IYER et al. 2005). Concepts stemming from tools to compare 3D CAD models could also prove helpful since they provide the possibility to compare very complex solids. Such tools, which can be found as proprietary software<sup>24</sup> or within commercial CAD packages<sup>25</sup>, are most often used to calculate the deviations between two versions of one and the same part. The checks they perform include, for example, comparisons of the number of lines and faces of objects and the angles between all surface normals. To deliver meaningful results, the two parts that are compared, usually have to be properly positioned and oriented with respect to each other so that their common areas overlap. Comparing geometry without knowing the transformation information upfront or at least a reference like the local coordinate system can result in very complex and time intensive calculations. In any case, the tools to compare 3D CAD models cannot be used for parametric grammar rules, since they would only enable non-parametric matching of the LHS.

### ***Sweeping operations***

Using sweeping operations based on 2D cross-sections that are defined using straight lines is generally possible but restricted in the current state of the approach. The user has to design a cross-section carefully in accordance with the instructions given in Chapter 4.3.2. Otherwise the matching might fail or deliver wrong results. The approach for matching cross-sections has to be enhanced in this regard to become more general. In comparison to cross-sections that are based on plane primitives and modified using Boolean operations, it is easier to create more complex cross-sections using the line-based approach. However, the lines in the cross-sections are not parameterized and therefore cannot be incorporated in the definition of parametric rules. The approach could be enhanced in that regard based on using a sketcher that enables the definition of parameterized lines. As an extension, the cross-sections could also be created based on a 2D grammar approach and subsequently used in 3D grammar rules.

Further, it would also be interesting to use sweeping operations based on a trajectory. However, this is difficult to be incorporated in the grammar approach as curvilinear matching remains an open issue.

---

<sup>24</sup> e.g. CompareVidia, <http://www.capvidia.com/products/comparevidia/> (accessed January 14<sup>th</sup> 2012)

<sup>25</sup> e.g. CATIA V5 workbench ‘Healing Assistant’

### ***3D labels***

3D labels do not have any geometric parameters, but their translation or rotation parameters can be used for the definition of parametric spatial relations. The remaining parameters, color and name, can only be set and matched statically. Making colors and label names also available as parametric properties would enable a generalization with regard to matching a wider range of different labels. For example, the color of a label could be restricted by a range of colors where the upper and lower limits are defined by certain color values. Using numbers as label names would further allow for the definition of equations. Assume, for example, that the match of a rule's LHS delivers a label with name '2' and the currently applied rule contains an equation adding '3' to the detected name. This would mean that the next rule that can be applied would have to contain a label with name '5' in its LHS. This opens up further possibilities especially with regard to state labels. On an even more general level, the use of regular expressions could be considered to not only match numbers but also strings and substrings.

In the long term perspective examples using the 3D labels approach to circumvent the LHS matching problem could be developed using other, more advanced 3D modeling techniques for the design of parametric segments. Especially UDFs (cf. 2.5) are interesting for further investigation. They have also the potential to be used in the LHS of a rule as parameterized geometry to enable the definition of more complex parametric subtractive or replacing rules. Assigning a unique internal 'name' or rather 'object type' to every UDF that is created, UDFs could be treated the same way as any of the parameterized primitives for LHS matching, i.e. checking the type, the free parameters and the relative transformation. This would be an enhanced concept for conventional set grammar rules. This concept could also be seen as an enhancement of the UDF concept in CAD as the UDFs do not have to be placed manually anymore but 'know' where they can be inserted in an existing design, thus enabling a higher degree of design automation.

### ***Constraints***

The generation of meaningful or valid designs often requires the introduction of different constraints to a grammar. The current approach provides three different possibilities to constrain rule application without requiring programming: (1) the definition of parametric relations or restriction of parameters to values within a certain range, (2) the use of labels and (3) collision detection of objects including obstacles and design space restrictions. These, however, are not sufficient to cover all possible cases of constraints that might be required. What if, for example, only symmetrical design solutions are supposed to be generated or a certain kind of object is not allowed to be positioned within a certain distance to another object, but others are? Further mechanisms for constraint incorporation or ideally a concept to define constraints on a more general level are therefore needed.

### ***Prototype implementation***

In addition to the issues discussed so far, improvements to the software prototype are also needed. For example, in the current version of the system it is possible that the same solution is generated more than once during the application of a grammar. This should be avoided in



future versions. Detecting identical solutions that are generated by the exact same rule sequence is easier than those generated using different rule sequences. Further, in the current implementation collision detection is based on Boolean operations which are generally computationally intensive. Using a more advanced algorithm could help speeding up rule application that uses collision detection. Another functionality that would increase the usability of the system is the direct visualization of free parameters in the graphical representation of a rule. So far, they can only be seen by opening the dialog window for the free parameter definition. Finally, in manual or semi-automatic rule application modes, a helpful feature for the user would be a preview of the resulting design before finally applying a rule or the possibility to backtrack, i.e. to return to previous states in the rule application sequence or undo a rule application.

### 7.3 Future extensions

Taking a longer term perspective, more general extensions to the approach and the implementation are envisioned.

Automatic derivation of a script from the visual definition of a grammar rule using a scripting language, such as Python, would allow for the potential editing of the ‘rule’ code by designers for enhanced rule definition and customization, especially for the incorporation of constraints. This could be based, for example, on the macro-recording mechanism that is available for conventional designs in the underlying CAD system.

The spatial grammar interpreter can be used for examples where hundreds or even more design solutions are generated. In these cases, the system can only be beneficial for the designer if it also supports the possibility to automatically find good, or ideally the best, solution(s). Manual investigation of hundreds of solutions would take too long. Therefore, it is important to provide an interface to simulation in order to evaluate generated designs according to certain requirements, e.g. stresses in the vehicle wheel rims or cooling performance of the cooling fin designs (cf. examples in 6.1 and 6.2). Also, more difficult evaluation questions like fulfilling certain functionality with the least number of parts, the least complex parts or finding the least expensive solution could be subject to investigation in future work.

Using synthesis and analysis, or simulation, in a closed loop can also provide for more intelligent automatic rule selection and parameter definition enabling the incorporation of optimization and search methods to generate optimally directed designs using a generative grammar (see e.g. STARLING & SHEA 2005).

With regard to downstream processes like manufacturing, the spatial grammar approach can be combined with automated machining planning approaches to completely automate the design-to-fabrication process for customized products (see e.g. SHEA et al. 2010), and it can be investigated to what extent labels can be used to encode fabrication information and act as the connection to hand over additional information to the downstream fabrication system.

Another important issue is the further validation of the approach and the prototype implementation. So far it has only been used by a few students with CAD background. However, more tests of the system are needed including CAD designers to evaluate its

acceptance and to what extent it helps to effectively use spatial grammars in mechanical design. Also the spatial grammar examples that were developed based on the theoretical inspection of existing products have to be further validated, ideally in case studies in collaboration with industry partners to discuss the significance of the rules and the generated solutions and to further refine them. On a more general level, mechanisms to validate the set of rules in a grammar, e.g. to avoid contradicting rules, or to ensure that the evaluation of parametric equations delivers valid result, e.g. to avoid negative values for size parameters, should be investigated.

Finally, besides the discussed limitations and the proposed extensions, an important question remains how to conceive grammar rules in general (see also CHAKRABARTI et al. 2011 and MCKAY et al. 2012). The development of a general theoretical method to support designers in the process of defining ‘useful’ grammar rules is required. ‘Useful’ rules are defined if their application results in meaningful solutions, generating not only one or few solutions but solution spaces that ideally comprise creative, new solutions. This is a difficult issue that has not sufficiently been addressed in literature yet. Existing examples, also the ones in this thesis, are developed intuitively and in a generate and test manner, i.e. rules are defined and applied to check the outcome and subsequently the rules are adapted to reach the intended solution. This issue needs more investigation also to answer the question whether the development of grammars is systematically learnable or whether it is up to the intuition and experience of the user.

## 8. Conclusion

Spatial grammars have been successfully applied in various domains to describe languages of shapes and generate alternative designs. For increased use and acceptance of spatial grammar systems, the development of visual, interactive grammar interpreters that are designer friendly is crucial. In an intuitive way, they allow designers to define their own rules and apply them interactively to generate different design alternatives. This is important as the design of spatial grammars is most often an iterative process where the language and full impact of the rules is often not known until they are applied. Further, integration of spatial grammars into CAD systems, including all of their functionality, is important to help encourage designers to utilize and benefit from grammatical design methods.

This thesis presents a new approach for a three-dimensional spatial grammar interpreter. The main contribution of this approach is that it provides a flexible platform supporting designers with visual, interactive definition and application of their own spatial grammar rules in a familiar CAD environment without programming. It puts the creation and use of three-dimensional spatial grammars on a more general level and enables their usability in mechanical engineering CAD systems to provide for more ‘active’ support of the engineering designer. For the rule development phase, this includes the creation and positioning of geometric objects and labels in 3D space for the definition of non-parametric and parametric rules. Geometric objects can consist of parameterized primitives that can be combined to more complex objects using Boolean and sweeping operations. For the rule application phase, automatic matching of the left hand side of a rule in a current working shape is carried out along with the calculation of the positions and sizes of the objects in the right hand side according to the defined parametric relations.

While the main research goals of the thesis are achieved, several improvements to the developed approach are possible, for example, the generalization of the automatic LHS matching or further mechanisms to incorporate constraints into grammar rules. Future work towards more general extensions should include issues like the automatic derivation of a script from the visual definition of a grammar rule and linking the system with simulation and optimization software.



## 9. References

AGARWAL & CAGAN 1998

Agarwal, M.; Cagan, J.: A blend of different tastes: the language of coffeemakers. *Environment and Planning B: Planning and Design* 25 (1998) 2, p. 205-226.

AGARWAL et al. 2000

Agarwal, M.; Cagan, J.; Stiny, G.: A Micro Language: Generating MEMS Resonators using a Coupled Form-Function Shape Grammar. *Environment and Planning B: Planning and Design* 27 (2000) 4, p. 615-626.

ANTONSSON & CAGAN 2001

Antonsson, E. K.; Cagan, J. (Eds.): *Formal engineering design synthesis*. Cambridge, England: Cambridge University Press 2001.

BOLOGNINI et al. 2006

Bolognini, F.; Shea, K.; Vale, C. W.; Seshia, A. A.: A Multicriteria System-Based Method for Simulation-Driven Design Synthesis, ASME IDETC/CIE Conference. Philadelphia, USA, 2006.

CAGAN 2001

Cagan, J.: Engineering Shape Grammars: Where We Have Been and Where We Are Going. In: Antonsson, E. K.; Cagan, J. (Eds.): *Formal engineering design synthesis*. Cambridge, England: Cambridge University Press 2001, p. 65-91.

CELANI 2002

Celani, M. G. C.: CAD - The Creative Side: An Educational Experiment that Aims at Changing Students' Attitude in the Use of Computer-Aided Design, SIGraDi 2002 - Proceedings of the 6th Iberoamerican Congress of Digital Graphics. Caracas, Venezuela, 27.-29. November 2002.

CHAKRABARTI 2002

Chakrabarti, A. (Ed.): *Engineering Design Synthesis: Understanding, Approaches and Tools*. London: Springer 2002.

CHAKRABARTI et al. 2011

Chakrabarti, A.; Shea, K.; Stone, R.; Cagan, J.; Campbell, M.; Hernandez, N. V.; Wood, K. L.: Computer-Based Design Synthesis Research: An Overview. *Journal of Computing and Information Science in Engineering* 11 (2011) 2, p. 021003-1 - 021003-10.

CHASE 1989

Chase, S. C.: Shapes and shape grammars: From mathematical model to computer implementation. *Environment and Planning B: Planning and Design* 16 (1989) 2, p. 215-242.

## CHASE 2002

Chase, S. C.: A model for user interaction in grammar-based design systems. *Automation in Construction* 11 (2002) 2, p. 161-172.

## CHAU et al. 2004

Chau, H. H.; Chen, X. J.; McKay, A.; de Pennington, A.: Evaluation of a 3D Shape Grammar Implementation. In: Gero, J. S. (Ed.): *Design Computing and Cognition 04*. Cambridge, USA: Kluwer Academic Publishers 2004, p. 357-376.

## CHOI &amp; HAN 2002

Choi, G.-H. M., Duhwan; Han, S.: Exchange of CAD part models based on the macro-parametric approach. *International Journal of CAD/CAM* 2 (2002) 1, p. 13–21.

## COOPER &amp; LA ROCCA 2007

Cooper, D. J.; La Rocca, G.: Knowledge-based techniques for developing engineering applications in the 21st century, *Proceedings of the 7th AIAA Aviation Technology, Integration and Operations Conference*. Belfast, Northern Ireland, 2007.

## COOPER et al. 2001

Cooper, S.; Fan, I.; Li, G.: Achieving competitive advantage through knowledge-based engineering: A best practice guide. Prepared for the Department of Trade and Industry by: Department of Enterprise Integration, Cranfield University, Bedford, UK (2001).

## DEAK et al. 2006

Deak, P.; Rowe, G.; Reed, C.: CAD grammars, *Design Computing and Cognition 06*. Eindhoven, Netherlands, 2006.

## DOWNING &amp; FLEMMING 1981

Downing, F.; Flemming, U.: The bungalows of Buffalo. *Environment and Planning B: Planning and Design* 8 (1981) 3, p. 269-293.

## DUARTE 2005

Duarte, J. P.: A discursive grammar for customizing mass housing: The case of Siza's houses at Malagueira. *Automation in Construction* 14 (2005) 2, p. 265-275.

## DUARTE &amp; CORREIA 2006

Duarte, J. P.; Correia, R.: Implementing a description grammar: Generating housing programs online. *Construction Innovation: Information, Process, Management* 6 (2006) 4, p. 203-216.

## FENVES et al. 2005

Fenves, S. J.; Sriram, R. D.; Subrahmanian, E.; Rachuri, S.: Product Information Exchange: Practices and Standards. *Journal of Computing and Information Science in Engineering* 5 (2005) 3, p. 238-246.

## FLEMMING 1987

Flemming, U.: More than the sum of parts: The grammar of Queen Anne houses. *Environment and Planning B: Planning and Design* 14 (1987) 3, p. 323-350.

## GIPS 1975

Gips, J.: Shape Grammars and Their Uses: Artificial Perception, Shape Generation and Computer Aesthetics (Interdisciplinary Systems Research). Basel: Birkhäuser 1975.

## GIPS 1999

Gips, J.: Computer implementation of shape grammars, NSF/MIT Workshop on Shape Computation. Cambridge, USA, 1999.

## HEISSERMAN 1994

Heisserman, J.: Generative geometric design. Computer Graphics and Applications, IEEE 14 (1994) 2, p. 37-45.

## HEISSERMAN et al. 2004

Heisserman, J.; Mattikalli, R.; Callahan, S.: A grammatical approach to design generation and its application to aircraft systems, Proceedings of Generative CAD Systems Symposium '04. Pittsburgh, Pennsylvania, July 12-14 2004.

## HOFFMANN &amp; JOAN-ARINYO 1998

Hoffmann, C.; Joan-Arinyo, R.: On User-Defined Features. Computer-Aided Design 30 (1998) 5, p. 321-332.

## HOISL &amp; SHEA 2009

Hoisl, F.; Shea, K.: Exploring the Integration of Spatial Grammars and Open-Source CAD Systems, 17th International Conference on Engineering Design (ICED'09). Stanford University, California, USA, 24.-27.08.2009.

## IYER et al. 2005

Iyer, N.; Jayanti, S.; Lou, K.; Kalyanaraman, Y.; Ramani, K.: Three-dimensional shape searching: State-of-the-art review and future trends. Computer-Aided Design 37 (2005) 5, p. 509-530.

## JOWERS &amp; EARL 2010

Jowers, I.; Earl, C.: The construction of curved shapes. Environment and Planning B: Planning and Design 37 (2010) 1, p. 42-58.

## JOWERS &amp; EARL 2011

Jowers, I.; Earl, C.: Implementation of curved shape grammars. Environment and Planning B: Planning and Design 38 (2011) 4, p. 616-635.

## JOWERS et al. 2008

Jowers, I.; Prats, M.; Lim, S.; McKay, A.; Garner, S.; Chase, S.: Supporting Reinterpretation in Computer-Aided Conceptual Design, EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling. Annecy, France, 2008.

## JOWERS et al. 2010

Jowers, I.; Hogg, D.; McKay, A.; Chau, H.; de Pennington, A.: Shape detection with vision: Implementing shape grammars in conceptual design. Research in Engineering Design 21 (2010) 4, p. 235-247.

## KNIGHT 1980

Knight, T. W.: The generation of Hepplewhite-style chair-back designs. *Environment and Planning B: Planning and Design* 7 (1980) 2, p. 227-238.

## KNIGHT 1989

Knight, T. W.: Color grammars: Designing with lines and colors. *Environment and Planning B: Planning and Design* 16 (1989) 4, p. 417-449.

## KNIGHT 1994

Knight, T. W.: *Transformations in Design: A Formal Approach to Stylistic Change and Innovation in the Visual Arts*. Cambridge University Press 1994.

## KNIGHT 1999

Knight, T. W.: Shape grammars: Six types. *Environment and Planning B: Planning and Design* 26 (1999) 1, p. 15-32.

## KONING &amp; EIZENBERG 1981

Koning, H.; Eizenberg, J.: The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B: Planning and Design* 8 (1981) 3, p. 295-323.

## KRISHNAMURTI 1981

Krishnamurti, R.: The construction of shapes. *Environment and Planning B: Planning and Design* 8 (1981) 1, p. 5-40.

## KRISHNAMURTI &amp; EARL 1992

Krishnamurti, R.; Earl, C. F.: Shape recognition in three dimensions. *Environment and Planning B: Planning and Design* 19 (1992) 5, p. 585-603.

## KRISHNAMURTI &amp; STOUFFS 1993

Krishnamurti, R.; Stouffs, R.: Spatial grammars: Motivation, comparison, and new results, 5th International conference on computer-aided architectural design futures. Pittsburgh, USA, 1993.

## LEE 1999

Lee, K.: *Principles of CAD/CAM/CAE Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. 1999.

## LEVITT 1991

Levitt, R. E.: *Knowledge-Based Systems in Engineering*. McGraw-Hill, Inc. 1991.

## LI 2005

Li, A. I.-k.: Thoughts on a designer-friendly shape grammar interpreter, Proceedings of the 23th conference of education and research in computer aided architectural design (eCAADe 2005). Lisbon, Portugal, 2005.

## LI et al. 2009a

Li, A. I.-k.; Chau, H. H.; Chen, L.; Wang, Y.: A Prototype for developing Two- and Three-Dimensional Shape Grammars, CAADRIA 2009: 14th International Conference on Computer-aided Architecture Design Research in Asia. Toulieu, Taiwan, 2009.



LI et al. 2009b

Li, A. I.-k.; Chen, L.; Wang, Y.; Chau, H. H.: Editing Shapes in a Prototype Two- and Three-Dimensional Shape Grammar Environment, *Computation: The new realm of architectural design - proceedings of the 27th conference of education and research in computer aided architectural design (eCAADe 2009)*. Istanbul, Turkey, 2009.

LI & KUEN 2004

Li, A. I.-k.; Kuen, L. M.: A set-based shape grammar interpreter, with thoughts on emergence, *First International Conference on Design Computing and Cognition DCC'04 (Paper read at Workshop 3, "Implementation issues in generative design systems")*. Massachusetts Institute of Technology, 17–19 July 2004.

LIN et al. 2009

Lin, Y.-s.; Shea, K.; Johnson, A.; Coultate, J.; Pears, J.: A Method and Software Tool for Automated Gearbox Synthesis, *ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. San Diego, USA, 30.08.-02.09. 2009.

MCCORMACK & CAGAN 2002

McCormack, J. P.; Cagan, J.: Designing inner hood panels through a shape grammar based framework. *AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing)* 16 (2002) 04, p. 273-290.

MCCORMACK & CAGAN 2003

McCormack, J. P.; Cagan, J.: Increasing the Scope of Implemented Shape Grammars: A Shape Grammar Interpreter for Curved Shapes, *ASME Conference Proceedings*. 2003.

MCCORMACK & CAGAN 2006

McCormack, J. P.; Cagan, J.: Curve-based shape matching: Supporting designers' hierarchies through parametric shape recognition of arbitrary geometry. *Environment and Planning B: Planning and Design* 33 (2006) 4, p. 523-540.

MCCORMACK et al. 2004

McCormack, J. P.; Cagan, J.; Vogel, C. M.: Speaking the Buick language: Capturing, understanding, and exploring brand identity with shape grammars. *Design Studies* 25 (2004) 1, p. 1-29.

MCCULLOUGH 1996

McCullough, M.: *Abstracting Craft: The Practiced Digital Hand*. MIT Press 1996.

MCGILL & KNIGHT 2004

McGill, M.; Knight, T.: *Designing Design-Mediating Software: The Development of Shaper2D*, *Proceedings of eCAADe 2004*. Copenhagen, Denmark, 2004.

MCKAY et al. 2012

McKay, A.; Chase, S. C.; Shea, K.; Chau, H. H.: Spatial grammar implementation: From theory to useable software. *AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing)* 26 (2012) 02, p. 143-159.

PAHL et al. 2007

Pahl, G.; Beitz, W.; Feldhusen, J.; Grote, K. H.: Engineering Design: A Systematic Approach. London: Springer 2007.

PIAZZALUNGA & FITZHORN 1998

Piazzalunga, U.; Fitzhorn, P.: Note on a three-dimensional shape grammar interpreter. Environment and Planning B: Planning and Design 25 (1998) 1, p. 11-30.

SHAH & MÄNTYLÄ 1995

Shah, J. J.; Mäntylä, M.: Parametric and feature-based CAD/CAM: Concepts, techniques, and applications. New York: John Wiley & Sons 1995.

SHEA & CAGAN 1997

Shea, K.; Cagan, J.: Innovative dome design: Applying geodesic patterns with shape annealing. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing) 11 (1997) 05, p. 379-394.

SHEA & CAGAN 1999

Shea, K.; Cagan, J.: Languages and semantics of grammatical discrete structures. AI EDAM (Artificial Intelligence for Engineering Design, Analysis and Manufacturing) 13 (1999) 04, p. 241-251.

SHEA et al. 2005

Shea, K.; Aish, R.; Gourtovaia, M.: Towards integrated performance-driven generative design tools. Automation in Construction 14 (2005) 2, p. 253-264.

SHEA et al. 2010

Shea, K.; Ertelt, C.; Gmeiner, T.; Ameri, F.: Design-to-fabrication automation for the cognitive machine shop. Advanced Engineering Informatics 24 (2010) 3, p. 251-268.

STARLING & SHEA 2002

Starling, A.; Shea, K.: A Clock Grammar: The Use of a Parallel Grammar in Performance-Based Mechanical Design Synthesis, ASME DETC Conference. Montreal, Canada, September 29 - October 2 2002.

STARLING & SHEA 2005

Starling, A.; Shea, K.: A parallel grammar for simulation-driven mechanical design synthesis, ASME IDETC/CIE Conference. Long Beach, CA, USA, September 24-28 2005.

STINY 1977

Stiny, G.: Ice-ray: A note on the generation of Chinese lattice designs. Environment and Planning B: Planning and Design 4 (1977) 1, p. 89-98.

STINY 1980a

Stiny, G.: Introduction to shape and shape grammars. Environment and Planning B: Planning and Design 7 (1980) 3, p. 343-351.

## STINY 1980b

Stiny, G.: Kindergarten grammars: Designing with Froebel's building gifts. *Environment and Planning B: Planning and Design* 7 (1980) 4, p. 409-462.

## STINY 1982

Stiny, G.: Spatial relations and grammars. *Environment and Planning B: Planning and Design* 9 (1982) 1, p. 313–314.

## STINY 1991

Stiny, G.: The algebras of design. *Research in Engineering Design* 2 (1991) 3, p. 171-181.

## STINY 1992

Stiny, G.: Weights. *Environment and Planning B: Planning and Design* 19 (1992) 4, p. 413–430.

## STINY 2006

Stiny, G.: *Shape: Talking about Seeing and Doing*. Cambridge, Massachusetts: MIT Press 2006.

## STINY &amp; GIPS 1972

Stiny, G.; Gips, J.: Shape grammars and the generative specification of painting and sculpture, *Information Processing* 71. 1972. (*Information Processing 71: Proceedings of IFIP Congress 71*)

## STINY &amp; MITCHELL 1978

Stiny, G.; Mitchell, W. J.: The Palladian grammar. *Environment and Planning B: Planning and Design* 5 (1978) 1, p. 5-18.

## STOKES 2001

Stokes, M.: *Managing engineering knowledge: MOKA Methodology for knowledge based engineering applications*. London: Professional Engineering Publishing 2001.

## SUTHERLAND 1963

Sutherland, I. E.: *SketchPad: A man-machine graphical communication system*, AFIPS Spring Joint Computer Conference. 1963.

## TAPIA 1999

Tapia, M.: A visual implementation of a shape grammar system. *Environment and Planning B: Planning and Design* 26 (1999) 1, p. 59-73.

## TRESCAK et al. 2009

Trescak, T.; Esteva, M.; Rodriguez, I.: *General Shape Grammar Interpreter for Intelligent Designs Generations, Computer Graphics, Imaging and Visualization, CGIV'09*. Tianjin, China, 2009.

## ULRICH 1995

Ulrich, K.: The role of product architecture in the manufacturing firm. *Research Policy* 24 (1995) 3, p. 419-440.

## VERHAGEN et al. 2011

Verhagen, W. J. C.; Bermell-Garcia, P.; van Dijk, R. E. C.; Curran, R.: A critical review of Knowledge-Based Engineering: An identification of research challenges. *Advanced Engineering Informatics* 26 (2011) 1, p. 5-15.

## WANG &amp; DUARTE 2002

Wang, Y.; Duarte, J.: Automatic generation and fabrication of designs. *Automation in Construction* 11 (2002) 3, p. 291-302.

## WONG &amp; CHO 2004

Wong, W.-K.; Cho, C. T.: A Computational Environment for Learning Basic Shape Grammars, *International Conference on Computers in Education 2004*. Melbourne, Australia, 2004.

## WONG et al. 2005

Wong, W.-K.; Wang, W.-Y.; Chen, B.-Y.; Yin, S.-K.: Designing 2D and 3D Shape Grammars with Logic Programming, *10th Conference on Artificial Intelligence and Applications*. Kaohsiung, Taiwan, 2005.