This paper was originally published in the proceedings of the
17th International Conference on Engineering Design, ICED'09
by the Design Society: http://www.designsociety.org/

INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'09
24 - 27 AUGUST 2009, STANFORD UNIVERSITY, STANFORD, CA, USA

EXPLORING THE INTEGRATION OF SPATIAL GRAMMARS AND OPEN-SOURCE CAD SYSTEMS

Frank Hoisl and Kristina Shea
Virtual Product Development Group, Institute of Product Development, Technische Universität München, Germany

ABSTRACT
While Computer Aided Design (CAD) has made significant progress since its inception, CAD tools are still used primarily for design documentation rather than as active partners in the design process. Shape and spatial grammars provide methods for interactive, generative shape design but have yet to find general application within CAD systems. This paper investigates the potential for integrating spatial grammars and an open-source 3D modeling kernel and CAD system. The definition of a vocabulary of shapes based on a library of solid primitives and use of standard functions like geometric transformations and Boolean operations in rule applications are explored. Two examples illustrate the approach: the Kindergarten grammar and vehicle wheel rim designs. Advantages include (1) from an implementation viewpoint of spatial grammars, the coding effort decreases by making use of standard 3D modeling functionality, (2) from a usability viewpoint of spatial grammars, the combination improves the potential practical use of spatial grammars in CAD processes and (3) from a CAD viewpoint, spatial grammars provide an integrated means for automated, flexible generation of design alternatives.

Keywords: computer-aided design, generative design, spatial grammar, shape grammar, software implementation

1 INTRODUCTION
Today, standard software systems used by mechanical engineers in design are 3D Computer Aided Design (CAD) applications. The input and decisions needed to create a design are provided by the human designer who often uses the tool mainly to document a single pre-defined solution or its modifications. Traditional CAD systems on the market, therefore, can be considered passive [1].
Spatial and shape grammars are generative systems that generate shapes by starting from an initial shape, which exists within a defined vocabulary of shapes, and applying defined shape rules iteratively to an evolving shape. The language of shapes is defined by the set of rules and vocabulary. It can be calculated by applying all possible sequences of shape rules to all possible initial shapes. The shape rules, thus, provide a concise description of a design language of shape. The design language is often constrained to model a particular design style. This language is finite, although can be vast, and can represent a certain geometric style or a set of feasible shapes from an engineering viewpoint.
To extend on current CAD tools, shape grammars have potential to act as active partners in the design process. They can provide support for routine design tasks and, even more interesting, for the creation of spatially novel solutions beyond what a designer might think of. They can begin the generative process from an initial shape of a known design or from scratch. However, their application and mainstream uptake has been hindered because "grammar systems that are really useful are difficult to implement" [2].
Deak et al. [3] state: "Traditional shape grammar systems are not able to deal with CAD primitives directly. Using a design from a CAD application in a shape grammar system would require conversion of the design's representation to be compatible with the components of the specific system. It would be desirable if the representation does not have to be altered from the one used in CAD software." Therefore, the useful and practical application of shape grammars in the mechanical engineering design domain is tightly connected to making them available within CAD systems. These are the computer systems mechanical engineering designers work with daily. Implantation of spatial grammars in CAD systems has the potential to make the grammar approach to generative shape design

available more generally as well as providing improved CAD support for spatial design creation and exploration.

The aim of this paper is to investigate two research questions: (1) can we use 3D modeling and CAD kernels to ease the implementation of shape grammars while at the same time providing a generalized implementation with wider applicability? and (2) how can we better integrate shape grammars with CAD systems and achieve greater user interaction within the generative design process? To address these questions, this paper starts with a brief background on shape and spatial grammars and the implementation of spatial grammars to date. Next, it investigates the requirements for spatial grammars with regard to their application in mechanical engineering design and discusses the potential for implementation using an open source 3D modeling kernel and an open-source CAD system to fulfill those requirements. To demonstrate the practical use of the approach, two examples are shown: the Kindergarten grammar [4] and vehicle wheel rim designs. The paper concludes with a discussion of the benefits and limitations of the approach.

2 BACKGROUND – SHAPE AND SPATIAL GRAMMARS

Spatial synthesis is a fundamental task within mechanical design and can be thought of as creating form, or product structure, to fulfill desired behavior and function [5]. The approach for computational design synthesis used in this paper is based on shape grammars, which are a form of production systems. A uniform characterization of production systems can be found in [6]. Using shape grammars as a generative approach to shape design was first introduced by Stiny and Gips [7] and further detailed in Stiny [8]. To date there has been significant research on shape grammars and they have been successfully used in the domains of paintings, industrial design, decorative arts and, above all, architecture [9]. However, the exploration into the application of shape grammars in engineering design has been more limited [10]. Only during the last fifteen years an increasing interest in the development of engineering shape grammars can be seen [9]. Formulation of shape grammars for generation of diverse products from MEMS devices to household products to styling of cars has been carried out [10]. The foundation of the Genesis system at Boeing is based on a grammar for 3D solids originally developed to generate alternative Queen Anne style houses in architecture [11]. To capture and understand brand image, McCormack et al. [12] have developed a shape grammar that captures the style of the front view of Buick cars to both generate known and new designs that reflect the brand image. Shape grammars can also be developed that describe both form and implicit function, for example to describe a language of microresonators where form-function coupling is inherently strong [13]. Except for the case of the Genesis system that has been used in both aerospace and architecture, most implementations in engineering tend to be specific to a certain class of products and cannot be easily transferred between domains or used more generally for mechanical design, as for example, a CAD system can.

Formalism

A shape grammar is defined formally 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 shape where $I \subset (S, L)^0$.

The set of labeled shapes including the empty labeled shape is $(S,L)^0$ and is also called the vocabulary. Shape rules are defined in the form $A \rightarrow B$, where A and B are both shapes in the vocabulary. To apply a shape rule to a given working shape C, first, the shape A in the left-hand side (LHS) of a rule has to be detected in C. This shape matching process can make use of valid Euclidean transformations, t, e.g. translation and rotation, that are applied to the shape A, to find more possible matches of A in the working shape C. The transformed shape A then gets subtracted from C and the transformed shape B of the right-hand side (RHS) of the rule is added, thus resulting in the new working shape C' where $C' = C - t(A) + t(B)$.

Parametric shape grammars extend on shape grammars by including parameters in the rules to create shape rule schemas. Without the use of transformations, they can be matched to a wider range of shapes when a valid parameterization can be found [8].

Strictly speaking a shape grammar involves the use of a maximal shape representation. In contrast to CAD representations that represent shapes as a set of distinct elements, e.g., line, circle, block, a

maximal shape representation 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. This ability for re-representating 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*. Spatial grammars, on the other hand, is the more general term and includes all kinds of grammars that define languages of shape, e.g. string grammars, set grammars, graph grammars and shape grammars [2].

Spatial grammar implementations

In the research area of spatial grammars, of the many that exist on paper only a minority have been computationally implemented. Both 2D and 3D software implementations exist. An overview of software implementations until 2002 can be found in [9]. However, mostly the focus is on spatial design generation only, e.g. for applications in art, industrial design and architecture, rather than for mechanical engineering design purposes. Interesting previous work related to this paper includes the following systems:

- "CAD grammars" by Deak et al. [3], who discuss the automation of spatial design in combination with standard CAD systems, however, the implementation is restricted to 2D.
- A 3D shape grammar implementation by Chau et al. [9] that has a strong focus on style and therefore deals with curvilinear wireframes and their related surfaces.
- The "3DShaper" by Wang and Duarte [14] that allows for the upfront definition of parameters to define the dimensions of two blocks as well as their spatial relation parameters (translation, rotation) for a maximum of two rules. According to the number of iterations, which also is defined manually, the generation of the design is executed internally and the resulting 3D model file can be opened in an external viewer.
- The "shape grammar interpreter" by Piazzalunga and Fitzhorn [15] that is built on the commercial ACIS kernel[1] and makes use of much of the kernel's functionality. The examples shown, however, focus on the usage of blocks as primitives and translational transformations only.

Limited implementations in or related to 3D CAD systems can also be found including one based on AutoCAD/AutoLisp by Celani for educational purposes [16][2]. Interesting work concerning parametric shape grammars was done by McCormack and Cagan [17]. In general, there has been limited success in the development of useful and generic computerized grammar-based design-tools [1].

3 REQUIREMENTS FOR SPATIAL GRAMMARS IN MECHANICAL ENGINEERING DESIGN

Applying spatial grammars to mechanical engineering design places special requirements on each component of the grammar formalism, as described in Section 2. Further, to enable spatial grammars to become a partner for the engineering designer, they must allow for certain interactions between a human designer and the generative system. Both aspects will be discussed in this section.

3.1 Vocabulary

Stiny [18] defines a hierarchy of shape algebras, U_{ij}, where $0 \leq i \leq j$. The first index, i, corresponds to the dimension of the basic shape elements, e.g. 0 corresponds to points, 1 corresponds to lines, 2 corresponds to planes and 3 corresponds to solids. The second index, j, corresponds to the dimension of the space where the shapes are composed and transformed. A typical 2D shape grammar, for example, is the product of the U_{02} and U_{12} algebras.

To create a general spatial grammar implementation for engineering purposes, a wide range of 2D and 3D shapes must be available within the vocabulary of shapes to represent standard and complex mechanical parts. 3D surfaces and solids must be included so that properties such as volume and weight can be calculated and to enable links to simulation, e.g., FEM, and fabrication, e.g., rapid prototyping (RP) and computer-aided manufacturing (CAM). Thus, U_{33} algebras are required [10]. However, considering generality, 3D parts can also be created from 2D shapes and include points and lines. Therefore, the full range of U_{i3} grammars ($0 \leq i \leq 3$) must be provided to support general mechanical design. If less is provided, acceptance may be lower as it also may seem that CAD is

[1] http://www.spatial.com
[2] See also: http://web.mit.edu/4.184/www/autogrammar.htm

moving back to its 2D origins where 3D objects were only represented through a set of 2D views.

3.2 Geometric transformations and subshape detection

In this section we return to the shape grammar formalism presented in Section 2. To more generally detect the shape defined in the LHS of a rule in the working shape, geometric transformations are required. They determine both where rules apply in the working shape as well as carrying out the shape calculation, $C' = C - t(A) + t(B)$. The most commonly used Euclidean transformations, t, include translation, rotation, and scaling. For example, to generalize the application of a rule, rather than requiring only its application to a certain orientation of a cube, all possible cubes under rotations of 90^0 can be considered. Thus, transformations can also be constrained. Translation is almost always required for positioning of the LHS shape in the working shape. Thus, the basic transformations of translation, rotation, and scaling in 3D must be available.

If a shape grammar system works automatically (see discussion in Section 3.3), rather than by requiring the user to determine where in a design a rule applies, the topic of subshape detection, or how to generally find embedded shapes within the working shape becomes an important issue. To achieve this in the most flexible way, the use of the maximal line representation [8] and the restriction to rational shapes [19] are required for general shape grammar implementations. However, the general subshape detection problem in three dimensions is still unresolved [9] and as a consequence, most of the three dimensional shape grammar implementations do not make use of subshape detection [20].

It can be said that the true power of spatial and shape grammars will be only seen through the availability of automatic subshape detection that in turn enables detection of emergent shapes that leads to more creative shape generation. However, for mechanical engineering purposes there can still be large potential for their generative power alone and generating designs beyond what designers can think of. In this paper, we do not take subshape detection as an initial requirement for a general 3D spatial grammar implementation since the ability to work in 3D is prioritized. However, we leave the issue to future consideration.

3.3 Generative design process

According to the shape grammar formalism, Chase [1] defines the steps in the application of a grammar as follows:

- the determination of a rule to apply,
- the determination of an object the rule is applied to, and
- the determination of a matching condition (cf. Section 3.2).

To create a spatial grammar implementation that can be integrated within typical CAD processes, consideration of user involvement in the generative process is imperative. Considering the interaction between designer/user and computer, the execution of the steps above can generally be done either (1) fully manual, (2) semi-automatic, or (3) fully automatic. A general spatial grammar implementation for mechanical engineering should incorporate all three modes.

For the semi-automatic mode there are several scenarios. The two most relevant ones are:

1. The user chooses a rule, the computer detects all locations where the rule applies in the current design under allowable transformations and the user selects which location to apply the rule, or
2. the user selects an object, the computer provides all the existing rules that can be applied to the object under allowable transformations and the user decides which rule to apply.

In an engineering context a spatial grammar system should allow for both scenarios. In the first scenario, the user is assisted in exploring how a rule can be applied. In the second, the user is assisted in exploring possible transformations to a chosen object.

Further, modes that use automatic rule-determination need an additional control mechanism for rule ordering, in the case that not only one but a series of rules gets applied. This comprises the sequence of rules and the number of times rules repeat and can be triggered either rule-internal, e.g. using labels, or by an external control structure such as numbering or indexing rules [21].

In general, fully automatic systems, or systems with minimal user interaction, require optimization and search techniques to direct the selection and application of rules to generate optimal or satisfying designs [1]. This requires a different type of user interaction as the designers must encode their design objectives and constraints within an optimization model that is used to guide the search. This mode is also potentially useful, as has been shown in previous implementations, e.g. in [5], but will not be considered in this paper and is left to future work.

3.4 Manual adaptability of designs within the generative process

In the practical use of spatial grammars, engineers will likely want to manually adapt the solution(s) the system has generated at some point. While a spatial grammar encodes a particular style or common design guidelines, sometimes "breaking the rules" is necessary either to transform the style or meet further constraints. Further, it can be the case that the system does not provide all rules to create complete parts, but rather assists the designer only in certain stages of the design process and the designer carries out the rest of the shape generation tasks manually.
For better integration within the design process and typical CAD processes, the system should not only enable manual modification of designs after the generation process, but provide a possibility to stop the automated generation so that the user can manually work on the geometry and restart the automated generation that then also works from the new basis of manual changes. Adaptation of designs is strongly interconnected with the different ways of controlling the generative process presented in Section 3.3.

4 APPROACH

To meet the requirements discussed in Section 3, the approach taken here is to use an open-source 3D solid modeling CAD kernel and an open-source CAD system as the basis for implementing integrated 3D spatial grammars. The approach aims at using as much functionality as possible commonly provided by both. In general, the functionality of CAD systems is intertwined with the underlying geometric modeling kernel, which provides the needed geometric modeling abilities and often some of the modeling operations.

4.1 Vocabulary

Initially we use a set grammar formulation of a spatial grammar where the vocabulary is based on parameterized 2D and 3D CAD primitive objects. These solid primitives are, in the most basic form, block, torus, wedge, cone, cylinder and sphere and can directly be used as the vocabulary for the definition of shape rules. A different possibility for the definition of vocabulary is the use of sweeping that creates 3D volumes from 2D sketches, a common way of working in CAD. Depending on the kind of curve used as a sweeping path, prisms (linear curve), rotational solids (circular curve) or even complex solids (curvilinear curve) can be created and used as vocabulary. Moreover, this approach provides a link to 2D grammar implementations that are more mature than any 3D implementations to date.
To add further complexity to the shapes defined in a vocabulary, Boolean operations can be used, that provide the capability to combine solids using the three operations union, difference and intersection. The use of Boolean operations to define more complex shapes and within shape rules significantly reduces the coding effort for implementing the spatial grammar, because the functionality is included in the solid modeling kernel. The type of spatial representation used for the vocabulary impacts the way rules can be implemented as well as the expressive power of rules.

4.2 Detection of shapes

Using set grammars for spatial applications provides the advantage that the search for possible locations for rule applications is reduced to searching a set of primitives. Within the current working shape, or set of primitives, a primitive shape type under certain transformations and considering additional parameters, e.g. related to the size of the primitive, is found. The required transformations can be calculated as a series of standard transformations.
For the detection of the LHS of a rule, most of the implemented systems do not include an automatic approach for searching even basic shapes. Piazzalunga and Fitzhorn [15], for example, only provide a manual selection. A way to, at least, perform a very basic search is to make use of the standard, but unique (internal) names many CAD systems create for designed shapes and the modifications. This makes it easy to find for example, all blocks in a given design. The search can then be further constrained by comparing parameters (see Section 4.4) of the found blocks against those specified in the rule.

4.3 Application of rules

Among the three different modes to execute the steps of applying a rule (see Section 3.3) only the fully automatic scenario does not require user interaction at all. For the other ones it is inevitable to

allow for user input to the system, but also give feedback to the user. For practical and user-friendly use of a computer system this is normally done via a Graphical User Interface (GUI) in which traditional CAD tools have their edge [1]. Besides visualization and rendering capabilities, they provide a wide range of standard and expected functionality, e.g. very simple functionality like "undo", the selection of different views or dialogues and pop-up windows. Making use of as many of these standard capabilities of a CAD system as possible decreases the coding effort for developing a spatial grammar system that includes a GUI, e.g. existing toolbars only have to be adapted rather than created from scratch, and this significantly increases the practical usability.
To allow for the application of different sets of rules to handle more than only one design task on one and the same system, rules can be saved in data files, loaded whenever they are needed and finally applied. The underlying functionality and needed graphical interface can also be directly taken from the CAD system.

4.4 Adaptability of designs and parametric rules

In modern mechanical engineering CAD, shapes are created in parameterized form, which allows for further modifications of generated designs. This concept is being made use of in our approach so that grammatically created shapes become adaptable instead of generating "static", unchangeable geometry only for visualization, e.g., the grammar systems by Wang and Duarte [14] or Starling and Shea [5]. Manual changes become possible using parametric primitives as a base vocabulary for the spatial grammar. Once created, the parameters can be easily accessed and changed via the GUI. Moreover geometrical transformations can be defined in a parameterized way, which allows for the modification of a shape's spatial orientation.
It is also possible to draw upon the parameterized form of shapes for the definition of parametric grammars [8]. Furthermore, it can be useful to define rules that only change parameters of shapes. This means that one does not have to delete the shape in the LHS of a rule and create the RHS from scratch, but rather only change parameters. In computational exhaustive generations this can help to save computational time.

4.5 Implementation

A first prototype was developed to explore and demonstrate the approach discussed. It uses an open source 3D mechanical engineering CAD system, FreeCAD[3], that is built on top of the open source geometric modeling kernel Open CASCADE[4] and integrates several other powerful open source libraries and applications, e.g. QT[5] for the development of the GUI. Additionally, during runtime, it internally uses the scripting language Python[6] for the control of the system, most interestingly for the definition of geometry.
An open source approach was taken for the general advantages of maximum freedom of adaptation and use. Furthermore, since the chosen system provides separated processes for the computation of the geometry and the displaying on the screen, the GUI can be deactivated in case of computationally exhaustive automatic design generations. Further, open source software is most often easier to integrate with other code and software, e.g. FEM for structural analysis. It is planned in the context of the broader computational design synthesis approach to integrate this system with a graph grammar for mechanical design synthesis at the product structure, behavioral and function levels [22] to provide complete synthesis support for mechanical design.

5 EXAMPLES

Two examples are now given to illustrate the approach.

5.1 Kindergarten grammars

Stiny [4] exhaustively described rules using the Froebel building blocks from the Kindergarten method as a constructive approach to the definition of design languages. The majority of the shape rules defined use blocks as the vocabulary of shapes that are orthogonally combined in many different ways.

[3] http://apps.sourceforge.net/mediawiki/free-cad/
[4] http://www.opencascade.org
[5] http://www.qtsoftware.com
[6] http://www.python.org/

This is also the basis for the designs generated with the system described in [15] and similarly in [14] to serve as comparison to other implementations.
The grammar shown in *Figure 40 (d)* in Stiny's paper (*Kindergarten grammar 40 (d)*) is depicted here in Figure 1 to illustrate implementation issues. The original definition of the rule is shown in the left of Figure 1. To generate the related design, a fairly simple implementation could be used. The block primitives all have a size of 20 x 10 x 5 mm. The dimension of block 1 (rule LHS) along the x-axis is set to 20 mm and along the y-axis is set to 10 mm. The second block (rule RHS top) is defined using the measurements exactly the other way around, 10 mm along the x-axis and 20 mm along the y-axis. Repeated application of this procedure to the subsequent blocks, while additionally translating the origin of the blocks by the block-height along the z-axis, leads to the design shown in the right of Figure 1.

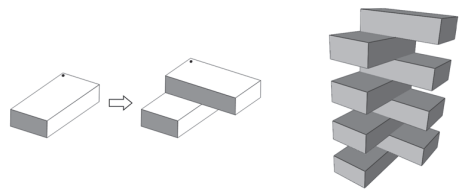

Figure 1. Kindergarten grammar 40 (d)

This process visually generates the design presented by Stiny, but internally it does not represent the grammar rule as it was intended since the spatial relation between the blocks is rather embedded in the setting of parameters defining the block dimensions. As a consequence the rule is virtually not applied in relation to a previously inserted block and also cannot be applied to a (manually) modified block, e.g., changing the angle of a block to a different value than 90°.
A more general implementation approach is now taken that uses transformations:

1. Manually or automatically select a block.
2. Insert a new block at the exact position, including all performed transformations, and with the exact measurements as the selected one. (Comparable to a "copy" command.)
3. Translate the new block by the height of the previous one along the z-axis.
4. Rotate the new block by 90° degrees around a defined axis. (Here, an axis parallel to the z-axis with its origin at x = 5mm and y = 5mm.)

To insert the second block in *Kindergarten grammar 40 (d)* the rule has to apply the rotation in step four above counterclockwise. The original definition of the rule by Stiny makes use of a mirror transformation; see the marker in the left of Figure 1. In a CAD system, however, every object is defined unambiguously concerning its position and alignment. Therefore, one rule can be implemented if a mirror transformation exists, or, a second rule has to be defined to handle the case of the mirrored block shown on top in the rule RHS of Figure 1.
Figure 2 shows two modified designs generated using the rule in Figure 1. For the design on the left the automatic generation was stopped after adding block five, the block was turned manually by 25° counterclockwise and the automatic process was continued. For the right hand design the automatic process completely ran once, afterwards block three was manually changed and, based on this modification, the rule was applied several times again. The implementation chosen here adds some additional coding effort for the rules, but it takes them to a more general level that is needed to apply further modifications to designs and rules.

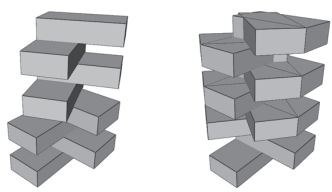

Figure 2. Modified Kindergarten grammar designs

5.2 Vehicle wheel rims

To demonstrate a mechanical engineering design example, the generation of wheel rims is chosen, because single piece rims combine mechanical issues, e.g. strength, with the need to be aesthetically pleasing. This grammar uses a variety of primitives as vocabulary:

- blocks, defined by three parameters to determine the position in x-, y- and z-direction according to their local coordinate system and three parameters to set the length (along x-axis), width (along y-axis) and height (along z-axis)
- cylinders, defined by three parameters to determine the position of the center in x-, y- and z-direction according to their local coordinate system and two parameters to set the height (along z-axis) and the radius (in xy-plane)
- wedges, defined by three parameters to determine the position in x-, y- and z-direction according to their local coordinate system, three parameters to set the length (along x-axis), width (along y-axis) and height (along z-axis) and one optional parameter to define the length of a flat section on top of the wedge

For every rim generation in the following examples, the first four rules are the same (Figure 3). Beginning with a starting symbol, Φ, a cylinder is inserted by rule (1). A second cylinder is positioned in relation to the center – symbolically marked "x" in the figure – of the first cylinder (2) and subtracted using the Boolean difference operation in rule (3) to generate the felly, or part of the wheel rim into which spokes are inserted. Using rule (4), another cylinder is added in the center of the felly representing the hub of the rim.

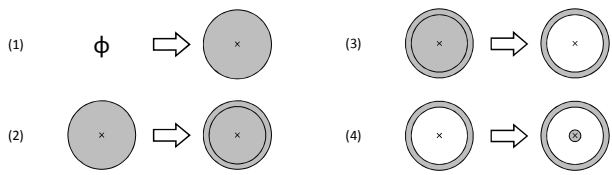

Figure 3. Rule (1)-(4)

To generate a first basic rim, two further rules need to be defined (Figure 4) in the x-y plane. The first one, rule (5), places the origin of a block in the center of a cylinder, which is the hub in this case. In analogy to the process described in section 5.1, rule (6) inserts a block at the exact position and with the exact measurements of a selected block and turns it counterclockwise by a defined angle, θ, around the origin of the selected block (marked "x"). Applying the latter rule a certain number of times, manually or automatically defined, a full rim is generated; see the right hand side of Figure 4.

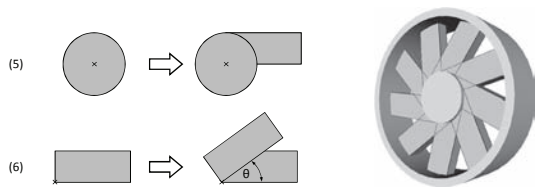

Figure 4. Rule (5) and (6) and a basic rim

To illustrate that the design is being generated in 3D and not only in plane, another rule, rule (7) shown in Figure 5, is defined that inserts a wedge on top of a block in the z-direction. Therefore a block is selected automatically, by searching the internal solid names. A new wedge is then inserted and the same transformation that was applied to the block is applied to the wedge. The width parameter of the wedge is set to the same value as the block. Finally, the value for the height of the block is used to translate the wedge in the positive z-direction, so that it is placed on top of the block. The rule and result of applying it to all existing blocks in the first example can be seen in Figure 5.

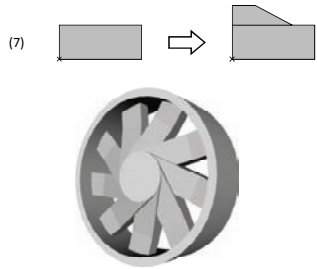

Figure 5. Rule (7) and rim with additional wedges

To extend the basic rim presented above, a further rule is defined (see Figure 6) that directly modifies the parameters of the solid primitive instead of deleting the LHS and adding a new solid in the RHS. It decreases the width *w* of a given spoke by changing the parameter of the block primitive by a certain percentage of the original value to *w'* and translates it in negative z-direction at the same time.

Figure 6. Rule (8)

The design presented in Figure 7 (left side) is generated by applying each of the rules (1) to (5) once, followed by a mixture of rules (6), (8) and manual modifications: First, rules (6) and (8) are applied once, resulting in the addition of the second spoke and the adaptation of its width as well as translating it in negative z-direction. Based on the second spoke, rule (6) is applied twice, inserting two more spokes of the same size and in the same z-direction as the second spoke. The parameters of spoke four are then manually modified to make them the same as spoke one. The described procedure restarts and is performed another two times to complete the rim.
The rim on the right hand side in Figure 7 is one of the examples generated using a very simple parameterized rule. Instead of setting the turning angle, θ, in rule (6) to a static value, as in the basic rim, it gets randomly set to values between 5° and 60° every time before the rule is applied. This demonstrates that a language of rim designs can be defined through a simple set of general rules that can be combined in a multitude of sequences and parameterizations and applied through manual, semi-automatic and automatic generative processes.

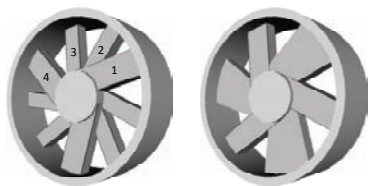

Figure 7. Examples for an extended basic rim

The actual generative power of grammars becomes more obvious by extending the set of rules which causes an extension of the design language so that it includes less predictable designs. To demonstrate this, rules (5) and (6) are used as a basis to create new rules (9) and (10) in Figure 8. Additionally, two further rules are defined: Rule (11) inserts a new block at the exact position and with the exact measurements of a selected block and translates it by the distance *d*. Rule (12) inserts two new blocks at the exact position and with the exact measurements of a selected block, rotates one of them by 45°, the other one by -45° and finally deletes the selected block. Note that rules (10)-(12) all have the same LHS but different RHSs.

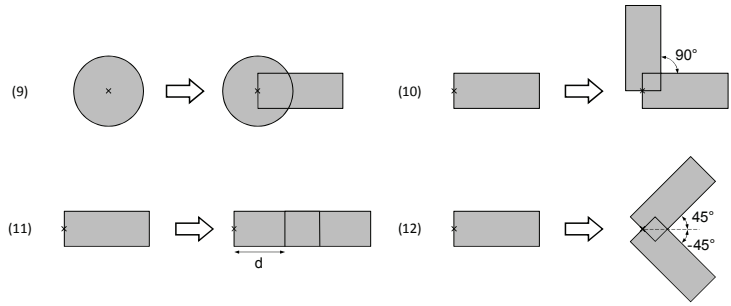

Figure 8. Rules (9)-(12)

Figure 9 shows four examples that were generated by applying rules (9)-(12) in different sequences.

Figure 9. Examples for rims generated applying different rule sequences

6 DISCUSSION

There is much functionality in geometric modeling kernels and CAD systems that can be made use of for implementing and applying spatial grammar systems in mechanical engineering design. The advantages especially stem from the combination of a strong geometric kernel with many standard functions required for implementing spatial grammars and a robust GUI that combined have the potential for creating more general 3D spatial grammar systems than found in the literature to date.
The implementation shown in Section 5 is a first, proof-of-concept prototype for this research, and initial benefits can already be seen:

- the use of a set of basic, parameterized primitive solids (blocks, cylinders, wedges) in combination,
- manual modification of generated designs,
- use of Boolean operations in rules to subtract and add shapes,
- use of transformation algorithms provided by the kernel, e.g. arbitrary rotations and translations,
- possibility for a basic shape detection to find the LHS of a rule in a design,
- approaches to save computational power by only adding the difference between LHS and RHS, where possible, and modifying shapes directly for parametric rules, and finally
- reduced coding effort by using the already existing functionalities provided by the kernel.

Future work on basic capabilities includes the integration of 2D spatial grammar approaches via sweeping operations, the usage of features in the CAD system to create more complex geometries and the application of multiple transformations, e.g. several rotations around more than one axis, to solids in three dimensions.
Compared to current CAD systems, integrated spatial grammars enable a more flexible and interactive means for generative shape design compared to scripting or parametric modeling. Compared to scripting, generating alternatives only requires applying a set of generic rules in a different sequence rather than changing a pre-defined script. With spatial grammars more interactivity is achieved through the three modes discussed. Compared to parametric modeling, parameterized shapes can be added and removed rather than only changing their dimensions.
Despite the advantages pointed out before, the approach of using a 3D modeling kernel and a CAD system also comes along with some limitations. One is that, since the shape representation in a CAD system is unambiguous in terms of positions and orientations of shapes, a robust way to find all possible transformations for matching a shape in the LHS of a rule to the working shape is needed that

includes at least general translation, rotation, mirror and scaling. Otherwise, more specific rules to cover special transformation cases are needed. Further, due to the internal CAD representations used, only set grammars can be defined and implemented. This type of representation conflicts with the widely used approach of maximal representation for general, automatic subshape detection in shape grammars and recognition of emergent shapes. However, without the ability for the detection of subshapes, a generative shape system can still have great value for exploring a wide range of spatial styles and alternatives. Recent research that could be helpful for dealing with the (sub-)shape recognition problem in two dimensions without using maximal representation is presented by Jowers et al. [23] and uses a pixel based representation. A future research issue remains to investigate other possible methods for achieving some level of re-interpretation of shapes through combinations of standard geometric modeling representations used in CAD and other, more flexible representations.
A further important issue is the definition and implementation of shape rules. This is currently done by hard coding in most approaches so far, including the implementation shown in this paper. While in the implementation in this paper Python, an easy to learn scripting language is used, this is typically not feasible for an engineering designer. For increased use and acceptance of spatial grammar systems in mechanical engineering practice, it is crucial to develop a visual grammar interpreter in 3D to conveniently define new shape rules, or a shape grammar interpreter. The open-source approach taken in combination with a general interpreter also gives potential for the sharing and re-use of shape rules between designers. This will be explored in future research.

7 CONCLUSION

The implementation and use of 3D spatial grammars in combination with an open-source solid modeling and CAD system provides several advantages. From a software developer's point of view, the coding effort decreases by making use of standard geometric modeling functions provided by the underlying kernel. From an engineering designer, or usability viewpoint, the combination improves the potential practical use of spatial grammars for mechanical engineering by providing the possibility for manual changes to a design at any point in the generative process and to use the common advantages and commands of a CAD GUI. Finally, the combination brings more generative power to CAD systems turning them away from only documenting data and at the same time, through the encoding of shape rules, preserving design knowledge related to geometry and spatial relations in a more general, flexible and reusable way than today's CAD systems provide. In the future, further advantages are expected through the development of new spatial grammar interpreters that will allow designers to encode their own shape rules and are crucial to turn spatial grammars into interactive generative systems that are designer-friendly.

REFERENCES

[1] Chase, S.C. A model for user interaction in grammar-based design systems. *Automation in Construction*, 2002, 11, 161-172.
[2] Krishnamurti, R. and Stouffs, R. Spatial grammars: motivation, comparison, and new results. In *5th International conference on computer-aided architectural design futures*, Pittsburgh, USA, 1993, pp. 57-74 (North-Holland Publishing Co., Amsterdam, The Netherlands).
[3] Deak, P., Rowe, G. and Reed, C. CAD grammars. In *Design Computing and Cognition 06*, Vol. 2, Eindhoven, Netherlands, 2006, pp. 503-520 (Springer, Dordrecht).
[4] Stiny, G. Kindergarten grammars: designing with Froebel's building gifts. *Environment and Planning B: Planning and Design*, 1980, 7(4), 409-462.
[5] Starling, A. and Shea, K. A parallel grammar for simulation-driven mechanical design synthesis. In *ASME IDETC/CIE Conference*, Long Beach, CA, USA, September 24-28, 2005 (ASME).
[6] Gips, J. and Stiny, G. Production systems and grammars: a uniform characterization. *Environment and Planning B: Planning and Design*, 1980, 7(4), 399-408.
[7] Stiny, G. and Gips, J. Shape grammars and the generative specification of painting and sculpture. In *Information Processing 71*, 1972, pp. 1460-1465 (North-Holland Publishing Company).
[8] Stiny, G. Introduction to shape and shape grammars. *Environment and Planning B: Planning and Design*, 1980, 7(3), 343-351.
[9] Chau, H.H., Chen, X.J., McKay, A. and dePennington, A. Evaluation of a 3D Shape Grammar Implementation. In *Design Computing and Cognition*, Cambridge, USA, 2004, pp. 357-376 (Kluwer Academic Publishers, Dordrecht).

[10] Cagan, J. Engineering Shape Grammars: Where We Have Been and Where We Are Going. In Antonsson, E.K. and Cagan, J., eds. *Formal Engineering Design Synthesis*, 2001, pp. 65-91 (Cambridge University Press, Cambridge, England).
[11] Heisserman, J. Generative geometric design. *Computer Graphics and Applications, IEEE*, 1994, 14(2), 37-45.
[12] McCormack, J.P., Cagan, J. and Vogel, C.M. Speaking the Buick language: capturing, understanding, and exploring brand identity with shape grammars *Design Studies*, 2004, 25(1), 1-29.
[13] Agarwal, M., Cagan, J. and Stiny, G. A Micro Language: Generating MEMS Resonators using a Coupled Form-Function Shape Grammar. *Environment and Planning B: Planning and Design*, 2000, 27(4), 615-626.
[14] Wang, Y. and Duarte, J. Automatic generation and fabrication of designs. *Automation in Construction*, 2002, 11(3), 291-302.
[15] Piazzalunga, U. and Fitzhorn, P. Note on a three-dimensional shape grammar interpreter. *Environment and Planning B: Planning and Design*, 1998, 25, 11-30.
[16] Celani, M.G.C. Beyond analysis and representation in CAD: a new computational approach to design education. *Department of Architecture*, 2002 (Massachusetts Institute of Technology, Cambridge, USA).
[17] McCormack, J.P. and Cagan, J. Curve-based shape matching: supporting designers' hierarchies through parametric shape recognition of arbitrary geometry. *Environment and Planning B: Planning and Design*, 2006, 33(4), 523-540.
[18] Stiny, G. Weights. *Environment and Planning B: Planning and Design*, 1992, 19(4), 413–430.
[19] Krishnamurti, R. SGI: A Shape Grammar Interpreter. 1982 (Design Discipline, The Open University, Walton Hall, Milton Keynes MK7 6AA).
[20] Gips, J. Computer implementation of shape grammars. In *NSF/MIT Workshop on Shape Computation*, Cambridge, USA, 1999.
[21] Knight, T. Shape grammars: six types. *Environment and Planning B: Planning and Design*, 1999, 26, 15-32.
[22] Helms, B., Shea, K. and Hoisl, F. A framework for computational design synthesis based on graph-grammars and function-behavior-structure. In *ASME IDETC/CIE Conference*, San Diego, CA, USA, August 30-September 2, 2009 (ASME).
[23] Jowers, I., Prats, M., Lim, S., McKay, A., Garner, S. and Chase, S. Supporting Reinterpretation in Computer-Aided Conceptual Design. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*, Annecy, France, 2008, pp. 151-158.

Contact: Frank Hoisl
Institute of Product Development, Virtual Product Development Group
TU München
Boltzmannstrasse 15
85748 Garching
Germany
+49-89-289-15137
+49-89-289-15144
frank.hoisl@pe.mw.tum.de
www.pe.mw.tum.de

Frank Hoisl is a doctoral student within the Institute of Product Development, Technische Universität München, where he also graduated in mechanical engineering (2006).

Prof. Dr. Kristina Shea is professor for Virtual Product Development within the Institute of Product Development, Technische Universität München. She studied Mechanical Engineering at Carnegie Mellon University (USA) where she completed her B.S. with University Honors (1993), M. S. (1995) and Ph.D. (1997).