Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 165/176

Chapter 14
Implementing Shape Grammars for Designers

Andrew I-kang Li

Abstract For the computational understanding of visual artifacts, shape grammar provides an important theoretical framework. In addition to the theory, there have been numerous computer implementations; these have tended to be proofs of concept. As such, they are essential steps in development, but do not directly help researchers do the kind of analyses seen in the literature, which were done by hand. That is to say, we have a theory but not yet a sturdy tool. We present a prototype implementation to help designers and design researchers work with shape grammars. This implementation allows users to focus on domain tasks—editing and testing grammars—by shielding them from sub-domain tasks—mechanical tasks like matching shapes and applying rules. A grammar is displayed as a collection of shapes in 3D space that users can manipulate directly; a commercial 3D modeling application is used for this purpose. The components of the implementation are designed to make it easy for users to switch between editing and testing their grammars. The implementation handles emergence and is general. We report on users' experiences with the implementation in workshops on grammatical design and analysis.

14.1 Introduction

For the computational understanding of visual artifacts, shape grammar [1, 2] provides an important theoretical framework. It has been used on subjects ranging from patterns on classical Greek pottery [3] to a twelfth-century Chinese text on buildings [4] to twentieth-century paintings [3].

In addition to the theory, there have been numerous computer implementations (see Chase [5] for a thorough overview); these have tended to be proofs of concept. As such, they are essential steps in development, but do not directly help

A.I. Li (✉)
Kyoto Institute of Technology, Kyoto, Japan
e-mail: andrewli@kit.ac.jp

J.-H. Lee (ed.), *Morphological Analysis of Cultural DNA*,
KAIST Research Series, DOI 10.1007/978-981-10-2329-3_14

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 166/176

researchers do the kind of analyses above, which were done by hand. That is to say, we have a theory but not yet a sturdy tool.

As an analogy to this situation, consider the financial spreadsheet. The idea is straightforward. There is a matrix of cells, and in each cell the financial analyst can enter either a number (an independent variable) or a formula that operates on numbers in other cells to produce a new number (a dependent variable). If he changes one or more of the independent variables, the dependent variables change accordingly.

There are two levels of work here. One involves formulating the independent variables and the formulas. This is the stuff of financial analysis, the domain level. The other level is the calculation of the dependent variables. This is straightforward execution of domain-level decisions and occurs at the sub-domain level.

In an implementation of a spreadsheet, the work is divided cleanly: the financial analyst works with the numbers and formulas, and the implementation takes care of the arithmetic. He is relieved of sub-domain work and can focus on domain work.

If there were no spreadsheet application, and the financial analyst had only grid paper, pencil, eraser, and perhaps a calculator, he could still do his work, but it would be slow and he would make many mistakes. Worse, he would be spending time and effort on arithmetic that he could be spending on financial analysis.

This is roughly the situation of design analysts who use shape grammars. Their domain work is creating grammars and evaluating the designs generated. [See Computing style [6]] The sub-domain work involves applying rules to existing shapes to produce new shapes; this is shape arithmetic. In studies like those mentioned above, the analysts worked by hand, doing large amounts of shape arithmetic while trying to remember the big picture.

In response, we have developed a prototype implementation for shape grammars in which design analysts draw shapes and rules directly, just as financial analysts enter numbers and formulas directly into the cells of a spreadsheet. The implementation then takes care of the shape arithmetic, just as a spreadsheet application takes care of the financial arithmetic.

14.2 The Implementation

The implementation has three components. The first is the commercial modeling application, **Rhinoceros3D**; users use it to create, edit, and save grammars.

The second component is a free-standing **shape grammar interpreter**.[1] With this, users run grammars and generate new shapes. It is general, which means that

[1]The interpreter is based on an engine by Chau et al. [7]. This engine transforms shapes, finds subshapes, applies rules, and displays the current shape (in 3D). It handles emergence and labeled points. We wrapped this engine in an interface that displays rules visually; displays next shapes; and imports and exports shapes, rules, grammars, and derivations [8].

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 | Time: 7:10 pm | Page: 167/176

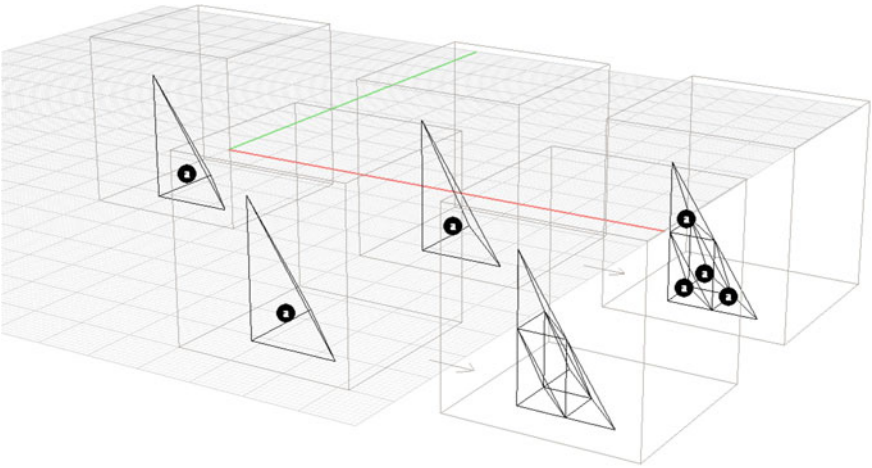

Fig. 14.1 A grammar for creating three-dimensional Sierpiński gaskets. It consists of an initial shape (*left*) and two rules (*right*)

users can make and use their own grammars. It also handles emergence, which, as Krishnamurti ([9]: 940, n. 9) says, is what makes an implementation "worthwhile".

The third component is a set of **scripts**, written in Python and RhinoScript. Users use these to transfer grammars and shapes between Rhino and the interpreter.

To demonstrate how a user would use the implementation, we show how to create and run a grammar that generates the fractal designs known as Sierpiński gaskets. The grammar consists of one initial shape (the labeled tetrahedron) and two rules. Each rule consists of a left shape, an arrow, and a right shape. Each shape is contained in a frame (Fig. 14.1).

To generate a new design, the user starts with the initial shape, applies one of the rules, and transforms the initial shape into a new shape.[2] She continues applying a rule, each time transforming the current shape into a new shape, until she is satisfied with the shape or until a rule cannot be applied (Fig. 14.2).

14.3 Using the Implementation

Using the implementation involves these steps:

1. Creating the grammar.
2. Testing the grammar; i.e., generating and evaluating designs.

[2]For the technical explanation of rule application, see Stiny [1].

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 168/176

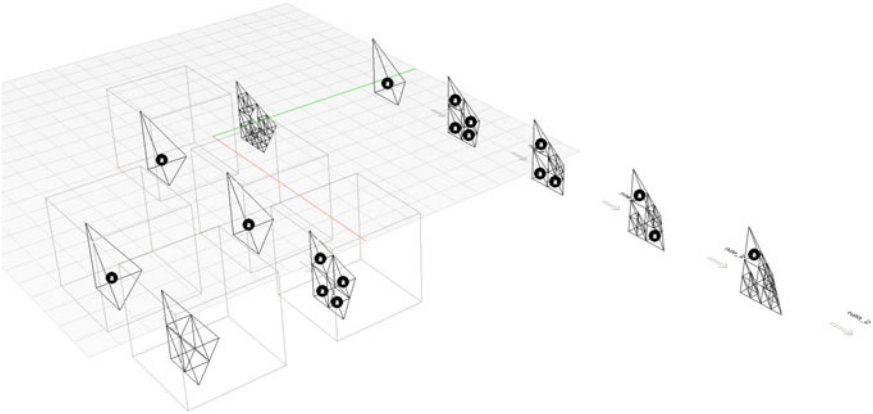

Fig. 14.2 The derivation of a new Sierpiński gasket, behind the original grammar. It consists of 6 shapes: the initial shape (at the back), 4 intermediate shapes (to the right of the initial shape), and the final shape (in front of the initial shape)

3. Revising and retesting the grammar until the results are satisfying; i.e., moving repeatedly through the edit–test cycle.
4. Using the results for some further purpose, i.e., converting the shapes from representations to more concrete artifacts like images for publication or 3D-printed objects.

14.4 Creating the Grammar

The user's first step is to set up an empty grammar document in a Rhino document; she does this with the *new grammar* script. The script creates two layers, one each for an initial shape and a rule, and draws three frames. Each frame delineates the volume inside which she can draw a shape (Fig. 14.3).

Now she can draw shapes with lines and annotation text dots.[3] Each shape should lie within the appropriate frame and on the appropriate layer. She can edit the grammar in any way at any time: she can add initial shapes and rules with the *new initial shape* and *new rule* scripts; she can delete initial shapes and rules; and she can revise initial shapes and rules. In this way, she can draw the two-rule grammar for generating three-dimensional Sierpiński gaskets (Fig. 14.1).

[3]Because the labeled points are implemented as annotation text dots, the labels can consist only of text. These are not as appealing as graphic labels, which are frequently seen in the literature, but they are functionally equivalent.

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 | Time: 7:10 pm | Page: 169/176

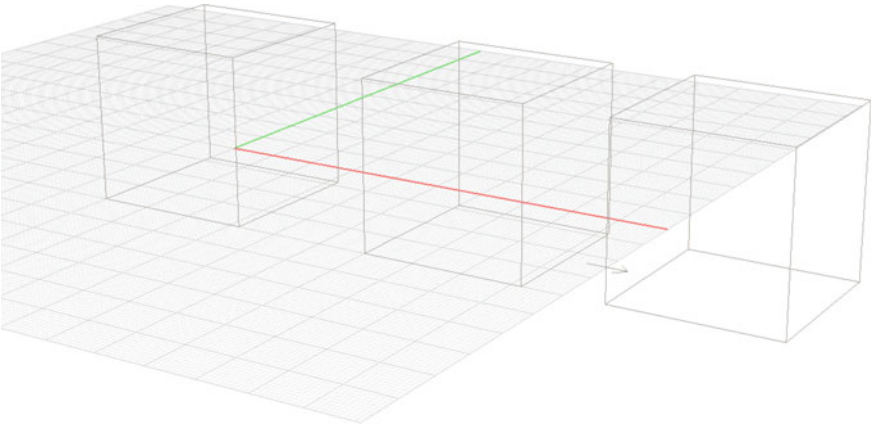

Fig. 14.3 The Rhino document, set up as an empty grammar document. The user draws an initial shape inside the *left frame*, and *left* and *right* shapes in the *middle* and *right frames*

The user can arrange the initial shapes and rules in any way she finds congenial, just as she can arrange icons on her computer desktop. This contributes to the sense that the grammar is an object for her to work with, to manipulate directly.

When she has finished drawing the grammar, she exports it with the *export grammar* script, which generates a file. This is a text file and is easy to understand, but in most cases she need not look at it.

14.5 Testing the Grammar

Now that the user has exported the grammar, she moves to the interpreter to generate shapes. The interpreter has three windows: the main window, in which the grammar and the current shape are displayed; the preview window, in which the possible next shapes are displayed; and a console window, for diagnostic purposes.

She opens the grammar file. The initial shape and rules are displayed in plan view on the left; the main canvas, on the right, is empty at first (Fig. 14.4).

To start generating shapes, the user selects the initial shape[4]; it then appears in the main canvas as the (first) current shape; it can be displayed in isometric, perspective, or orthographic views, and can be rotated in space (Fig. 14.5).

To see what next shapes can be obtained by applying any of the rules, the user clicks *show distinct (all rules).* To see those resulting from a single selected rule, she clicks *show distinct (1 rule).* The next shapes, if any, appear in plan view as a scrollable list in the preview window (Fig. 14.6). Calculating these next shapes is shape arithmetic, and it is done by the implementation, not by the user.

[4]It is necessary to select the first current shape because there may be more than one initial shape.

Layout: T1 Standard Book ID: 418385_1_En Book ISBN: 978-981-10-2328-6
Chapter No.: 14 Date: 1-9-2016 Time: 7:10 pm Page: 170/176

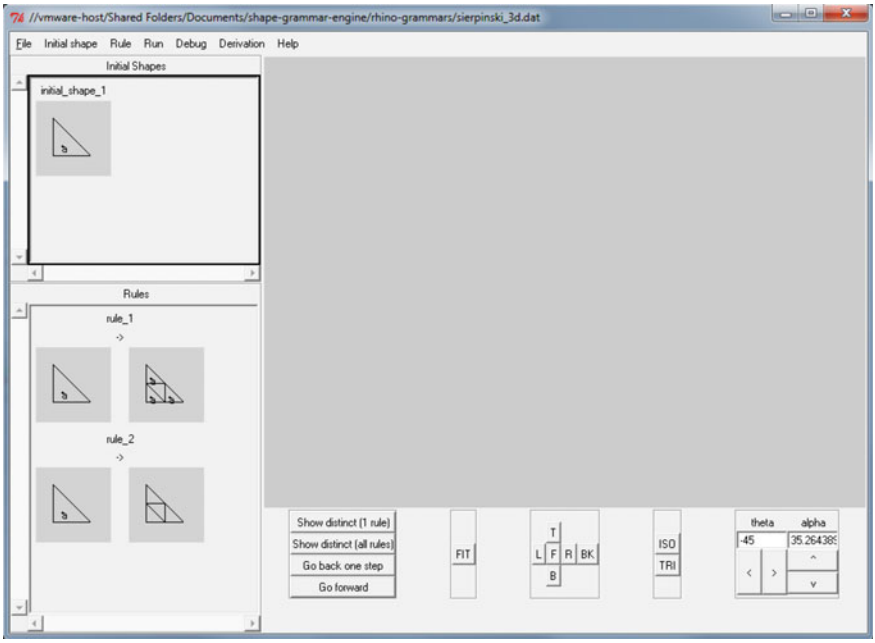

Fig. 14.4 The main window of the interpreter after importing the grammar. The initial shape is on the *upper left*. The rules are on the *lower left*. They are shown in plan view. No current shape has been selected, so the main canvas, on the *right*, is empty

In this case, there are two next shapes. The user can inspect either of these by selecting it; it then replaces the initial shape in the main canvas, becoming the new current shape (Fig. 14.7). She can continue applying rules and transforming the current shape until she is satisfied with the result. The interpreter retains a record of this history, known as a derivation.

If the user wants to undo or redo a rule application, she can step backward and forward through the derivation, as in a browser. It also may happen that she decides that she will not get a satisfactory result with the grammar as it is. In this case, she can simply return to the grammar document in Rhino, revise it, and run the revised version in the interpreter.

Let us assume that the user does not need to modify the grammar and has now finished a satisfactory Sierpiński gasket (Fig. 14.8).

She can save her results by selecting *Save derivation.* The interpreter generates a file which, like the grammar file, is an easily understood but ignorable text file. It includes the grammar, the initial shape, the final shape, all intermediate shapes, and the rule applied at each iteration.

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 171/176

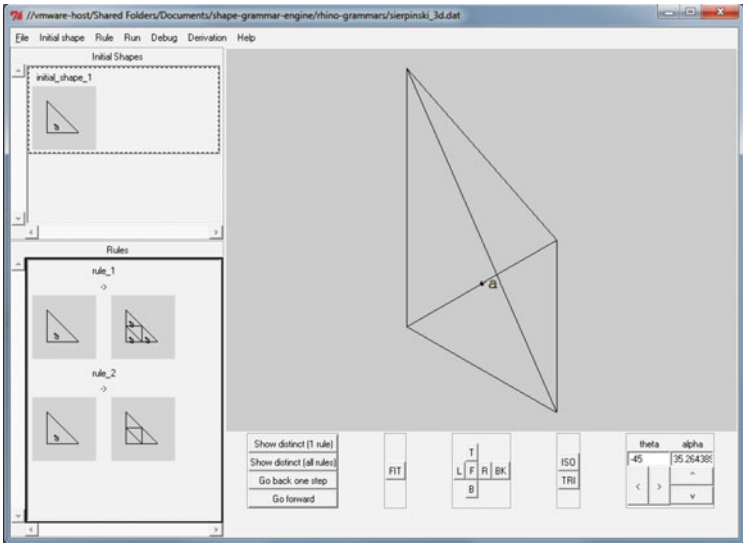

Fig. 14.5 The main window of the interpreter after selecting the initial shape as the (first) current shape. Since no rules have been applied yet, it is still the initial shape. It is shown here in isometric view

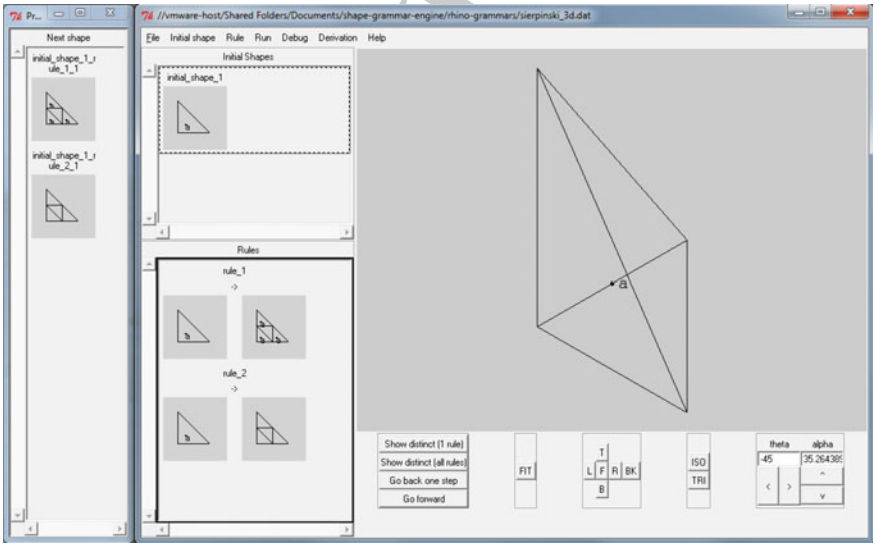

Fig. 14.6 Given the current shape and the rules, there are two possible next shapes. They are shown in plan view in the scrollable preview window on the *left*

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 172/176

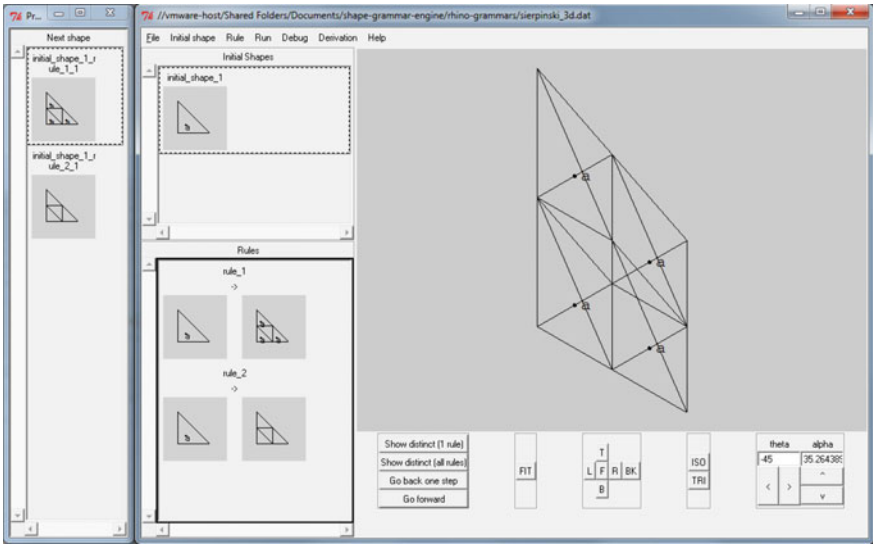

Fig. 14.7 The user selects one of the two possible next shapes (*left window*); it becomes the new current shape (*right window*, main canvas)

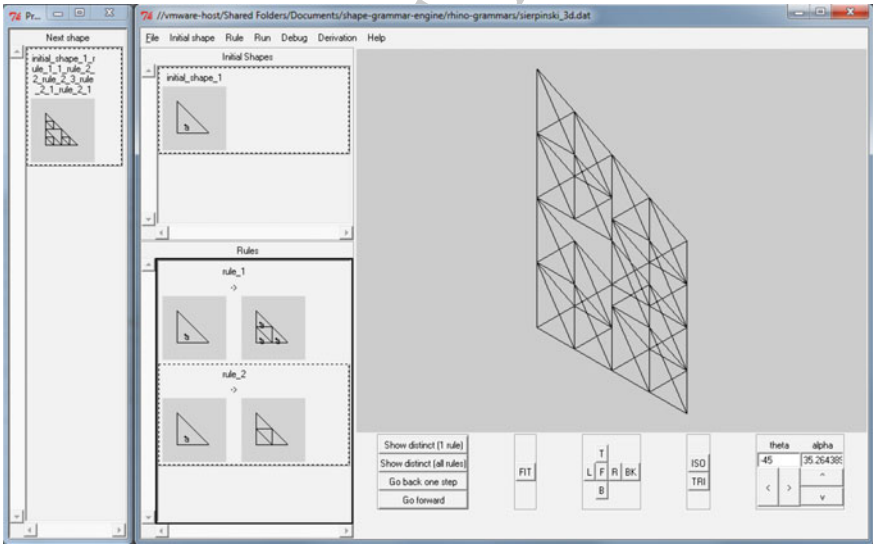

Fig. 14.8 The finished Sierpiński gasket

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 173/176

14.6 Using the Results

The user's work is not done: she has created new shapes, but now she needs to use them. That is, she needs to transform these representations into physical artifacts, whether fabricated objects or simply drawings on a page [10]. To begin this transformation, she uses the *import derivation* script, which uses the derivation file to draw the derivation in the Rhino document (Fig. 14.2). Now all the shapes in the derivation, from the initial shape to the final shape, appear in Rhino as wire frame models, which she can use like any others created in Rhino. She can convert them to surface and solid models; she can render and fabricate them.

14.7 Observations and Discussion

We have used our implementation in several workshops on mass customized design, ranging from one to five days in length. In the longer workshops, participants not only created designs with the implementation; they also converted those designs into physical objects through laser cutting and 3D printing. Some participants had a basic familiarity with Rhino or similar modeling application, and all were new to shape grammar. They were learning both shape grammar and the implementation at the same time.

The goal of the workshops was not to test the implementation per se; it was to give the participants the experience of accomplishing design tasks with shape grammars and the implementation. Thus, we were interested to see whether the implementation generally helped or hindered them, and in what ways.

14.8 Users Were Able to Focus on Domain-Level Work

We observed that participants learned the basics of both shape grammar and the implementation quickly, and were then doing design work easily, including downstream fabrication. For example, in the two-day workshop, participants were introduced to shape grammar on the first day, and to the implementation on the morning of the second day. By the end of the second day they had created and revised grammars, created designs, and fabricated some of those designs.

Once the participants had become familiar with the implementation, they focused on domain-level questions: What if I change this rule? How do I get that shape? In other words, they were successfully shielded from sub-domain-level tasks. We attribute this to several factors.

One is that the domain objects—the shapes and rules of the grammar—are presented in a domain-appropriate way, i.e., visually, and are directly manipulable. Most domain operations, such as 'find next shapes', are single commands, initiated by the user but executed by the implementation.

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 174/176

Another is that the implementation is general. Given the basic elements of lines and labeled points in three-dimensional space (using the U_{13} and V_{03} algebras, in grammar speak), users can create any shapes and rules. They can create grammars for their own purposes.

Finally, we suspect that being able to manipulate shapes directly and getting feedback quickly create a virtuous cycle that motivates users.

14.9 Users Were More Interested in Product Than Process

The derivation of a design is unique to that design, and specifies it within the design space. Grammars in the literature regularly include derivations. Given this significance, we provided the capability to move an entire derivation—not just the final shape—from the interpreter back into Rhino, and we expected users to use this capability. However, we observed that users showed little interest in the derivation. Instead, they were interested in accumulating new designs: product over process.

This discrepancy between theory and practise is, on reflection, not really surprising. As Woodbury [11, p. 17] has noted, designers tend to be pragmatic and indifferent to theory. 'Amateurs program because they have a task to complete for which programming is a good tool'. This applies to shape grammars too.

14.10 The Implementation Made Users Aware of the Conventions of Shape Grammar

The interpreter executes—and thereby enforces—the conventions of shape grammar, just as, say, an interpreter of the Python programming language executes and enforces those of Python. As a result, users who have not yet assimilated those conventions may be surprised by the results that the interpreter produces.

One case is when users could not visualize the results. This type of surprise is to be expected in shape grammar, because of emergence. (Recall [9].) The interpreter is doing what it should do.

Other cases, however, merit some thought. Sometimes users drew shapes imprecisely. For example, they drew rectangles with slightly different proportions and expected the interpreter to see them as similar. Or they drew shapes containing overlapping or abutting lines, which are invalid. In shape grammar theory, such shapes are reduced to their maximal form by replacing the overlapping and abutting lines with single lines. When the interpreter received non-maximal shapes, it sometimes produced incorrect results.

In such cases, we may wonder how to distinguish precise and imprecise, correct and incorrect. On the one hand, a formalism like shape grammar is a formalism, and therefore requires precision, including precision about imprecision. By requiring

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 175/176

precision, the interpreter helps users acquire the habit of precision. On the other hand, Woodbury's [11] remark about the pragmatism of designers suggests that we consider whether we can get by with less precision. This is a question addressed by Jowers et al. [12] with their 'fuzzy' implementation.

For our part, we will continue to implement shape grammars as they are defined formally, even as we are aware of the tension with designerly pragmatism.

14.11 Conclusion

Our experiences tend to confirm our basic approach: shield users from sub-domain tasks, support the edit–test cycle, and use a modeling application as an editing platform.

This approach has two general benefits. The first is that the sub-domain work is performed more accurately. The second is that the user can now concentrate her attention on domain tasks: editing and testing grammars.

There is additionally a third benefit, that three-dimensional grammars are usable. These have always been theoretically possible, but they are difficult to draw on paper and even more difficult to execute mechanically. It is probably for this reason that there is little three-dimensional work to be seen in the literature (Koning and Eizenberg [13] is one of the few examples). Now, with Rhino as the platform for viewing and editing grammars, it is easy to do three-dimensional work.

Needless to say, much remains to do, to ask, and to try before designers and design analysts have a robust tool for working with shape grammars. With respect to our implementation, the next step is to tighten the edit–test cycle by reduce the distance between the grammar-editing platform (Rhino) and the grammar-running platform (the interpreter). In the current version, switching between editing and testing requires exporting a file from one platform and importing it into the other. These steps can be eliminated by relocating the interpreter inside Rhino.

Other steps are less obvious. Having seen that users can be more interested in products than process, how should we develop the implementation so it best supports users? At the same time, the users that we observed were doing design work. Do users doing analysis work the same way? What about the many technical features of shape grammar that are still to be implemented: schemas, weights, parallel grammars, description, and so on? How will users want to interact with these features?

There is still much to learn about how designers and analysts work with grammars.

Layout: T1 Standard | Book ID: 418385_1_En | Book ISBN: 978-981-10-2328-6
Chapter No.: 14 | Date: 1-9-2016 Time: 7:10 pm | Page: 176/176

References

1. Stiny, G. (1980). Introduction to shape and shape grammars. *In Environment and planning B, 7*, 343–351.
2. Stiny, G., & James, G. (1972). Shape grammars and the generative specification of painting and sculpture. In C. V. Frieman (Ed.), *Information Processing '71* (pp. 1460–1465). Amsterdam: North-Holland.
3. Knight, T. W. (1994). *Transformations in design*. Cambridge: Cambridge University Press.
4. Li, A. I. (2001). A shape grammar for teaching the architectural style of the *Yingzao fashi*. PhD dissertation. Massachusetts: Massachusetts Institute of Technology.
5. Chase, S. (2010). *Shape grammar implementations: The last 35 years*. DCC workshop, Stuttgart. July 11, 2010.
6. Li, A. I. (2011). Computing style. *Nexus Network Journal, 13*(1), 183–193.
7. Chau, H. H., Xiaojuan, C., Alison, M., & de Alan, P. (2004). Evaluation of a 3D shape grammar implementation. In John S. Gero (Ed.), *Design computing and cognition '04* (pp. 357–376). Dordrecht: Kluwer
8. Li, A. I., Chau, H. H., Chen, L., & Wang, Y. (2009). A prototype system for developing two- and three-dimensional shape grammars. In *Proceedings of the 14th International Conference on Computer-Aided Architectural Design Research in Asia, CAADRIA 2009* (pp. 717–726).
9. Krishnamurti, R. (2015). Mulling over shapes, rules and numbers. *Nexus Network Journal, 17*(3), 925–945.
10. Knight, T. W. (2015). Shapes and other things. *Nexus Network Journal, 17*(3), 963–980.
11. Woodbury, R. (2010). *Elements of parametric design*. London: Routledge.
12. Jowers, I., Hogg, D. C., McKay, A., Chau, H. H., & de Pennington, A. (2010). Shape detection with vision: Implementing shape grammars in conceptual design. *Research in Engineering Design, 21*(4), 235–247.
13. Koning, H., & Eizenberg, J. (2010). The language of the prairie: Frank Lloyd wright's prairie houses. *Environment and planning B: Planning and design, 8*(3), 295–323.