SHAPE GRAMMAR/DESIGN KNOWLEDGE-BASED SYSTEM (7B)

A PROTOTYPE SYSTEM FOR DEVELOPING TWO- AND THREE-DIMENSIONAL SHAPE GRAMMARS

ANDREW I-KANG LI
Department of Architecture, The Chinese University of Hong Kong
andrewili@cuhk.edu.hk

HAU HING CHAU
School of Mechanical Engineering, University of Leeds
h.h.chau@leeds.ac.uk

LIANG CHEN
Department of Architecture, The Chinese University of Hong Kong
chenliang@cuhk.edu.hk

AND

YANG WANG
Department of Architecture, National University of Singapore
akiwangy@nus.edu.sg

Abstract. A number of researchers have developed shape grammar systems, with a variety of aims. These systems all help users (to varying degrees) to run grammars, but not to develop grammars. However, we believe that developing grammars is also work and needs to be supported. A system to do this would make it easier and more convenient for people using grammars to do design work. Following the generate-test model, we design and implement a prototype system that supports the user in editing grammars, testing grammars, and switching easily between the two types of activity. We emphasize the graphic nature of the task: the user is all the time working with graphic objects, namely shapes.

Keywords. Shape grammar; interpreters; development; systems.

1. Introduction

Since Stiny and Gips's (1972) seminal paper more than three decades ago, researchers have steadily developed the theoretical aspects of shape grammars. On the practical side, researchers have also developed systems for supporting or implementing shape grammars. (For a recent list, see Chau et al. (2004). To this may be added work by Celani (2001), Romão (2005), McCormack and Cagan (2006), and the Design synthesis and shape generation project (2008).) These systems can be grouped roughly as follows:

- **Single-grammar systems**. These implement a single, fixed grammar. Users use these to explore the single, fixed language of designs defined by the grammar. Some examples are by Flemming (1987a; 1987b), Duarte (2001), Li (2002), Soman et al. (2003), and Wu (2004).
- **Shape addition systems**. These allow users to define a small number of rules of the form $A \rightarrow A + B$, where A and B are usually simple shapes like rectangles. These systems help users explore the effects of a single rule, especially when applied recursively. Some examples are by Wang and Duarte (2002) and McGill (2002).
- **General systems**. These allow users to specify and run any grammar within broad technical limits. Thus users can explore many grammars and the languages they define. Some examples are by Krishnamurti (n.d.), Krishnamurti and Giraud (1986), Chase (1987; 1989), Tapia (1999), and Chau et al. (2004).

One type of work for a user of grammars is exploring a language of designs defined by a finished grammar. This type of work is well studied: it is supported, in varying fashion, by systems in all three groups, and a model of user interaction for this type of work has been proposed by Chase (2002).

However, there is another type of work, namely developing a new grammar. This is often done by designers who use grammars as tools for creating designs. It has been less well studied.

Development is supported by systems in the second group, but only for a small proportion of grammars, namely those consisting of a few shape addition rules. The systems in the third group all provide some kind of support to users in creating shapes, rules, and grammars. Indeed, Tapia's (1999) GEdit is frequently commended for its user-friendliness. However, we believe that existing systems could do more to support users who design with grammars.

This subject is little treated in the literature. One rare discussion is by Knight (1991). It is also mentioned by Chase (2002), but not included in his model, which applies only to finished grammars.

If we understood grammar development better, we could create a better system for supporting this type of work. And, as a result, users designing with grammars could work more effectively.

We propose that systems for developing grammars should fulfill two requirements:

- They should be domain-specific; and
- They should support the generate-test model of work.

From these two general requirements, we infer specific practical guidelines and implement a prototype system, called Grammar Environment, which we describe here.

2. Guidelines

In designing a system for users, we think it helpful to consider the users' work (Mirel 2004). By identifying what users do, or want to do, we can deduce what the system should do; roughly put, this is anything that users do not want or need to do.

2.1. THE DOMAIN

What users want to do varies with the domain, hence our first requirement, domain-specificity. What does this mean for grammar development? We start from a simple point, well put by Gips (1999, 1): "Shape grammars are intended to form a basis for purely visual computation." That is, the objects that users handle and reason about are shapes, rules, and grammars. These are the domain objects, and their essential characteristic is that they are graphic. To implement this "graphicness", we propose the following guidelines:

- Domain objects should be displayed graphically.
- They should be directly manipulable.

So, for example, to define a shape, users should simply draw it. (The Design synthesis and shape generation project (2008) is unique in allowing users to draw shapes freehand.) They should not have to "encode" a shape by typing coordinates into a text file. This is the type of work that Tapia (1999, 59) calls "bookkeeping"; Flemming (1987a, 349) calls it "tedious and error-prone."

2.2. THE GENERATE-AND-TEST CYCLE

Our second requirement is that the system should support the generate-and-test cycle or, to propose a more grammar-specific term, the edit-and-run cycle. (Tapia (1999), in an implicit analogy with computer programming, includes "compile" in the cycle.) As Chase (2002) and Knight (1991) point out, this is how users usually develop grammars, and it is consistent with Schön's (1983) discussion of how designers design.

For a system to support the generate-and-test cycle, we propose two guidelines:

- The system should make it easy for users to: 1) edit a grammar; 2) test it (i.e, create output for evaluation); and, perhaps most important, 3) switch back and forth between the two activities.
- It should support emergence.

It is sometimes argued that emergence is unnecessary (Piazzalunga and Fitzhorn, 1998). This is often true if the grammar is fixed, i.e., if users are only exploring the language defined by the grammar and do not want or need to modify the grammar. However, if users are developing a grammar, then they may change it in an unpredictable way. Emergence is called for.

3. User interaction

3.1. DISPLAYING DOMAIN OBJECTS

Grammar Environment has three windows (fig. 1):

- The main window, which displays the current shape, a scrollable list of initial shapes, and a scrollable list of rules;
- The preview window, which displays a scrollable list of the possible next shapes; and
- The console window, which displays the system's operations for debugging purposes.

The grammar, its constituent objects (initial shapes, rules), and related objects (current shape, next shapes) are all displayed graphically. Users perform an action on an initial shape or rule by selecting it (by clicking on it) and choosing the action from a menu. Users can control the 3D display of the current shape with buttons for resizing and for choosing a view, a projection, and rotations.

3.2. EDITING A GRAMMAR

3.2.1. Manipulating rules

By editing a grammar, we mean more than just editing shapes; we mean manipulating all types of domain objects. We think that users might, for example, generate many rules and want to organize them. We offer this possibility indirectly, by allowing users to name initial shapes and rules; these are then listed in order by name. In this way, users can control the display, at least in one dimension.

Users can select an initial shape, a rule, or a shape in a rule, and then an

operation like duplicate, edit, delete, rename. They can also import and export shapes and rules.

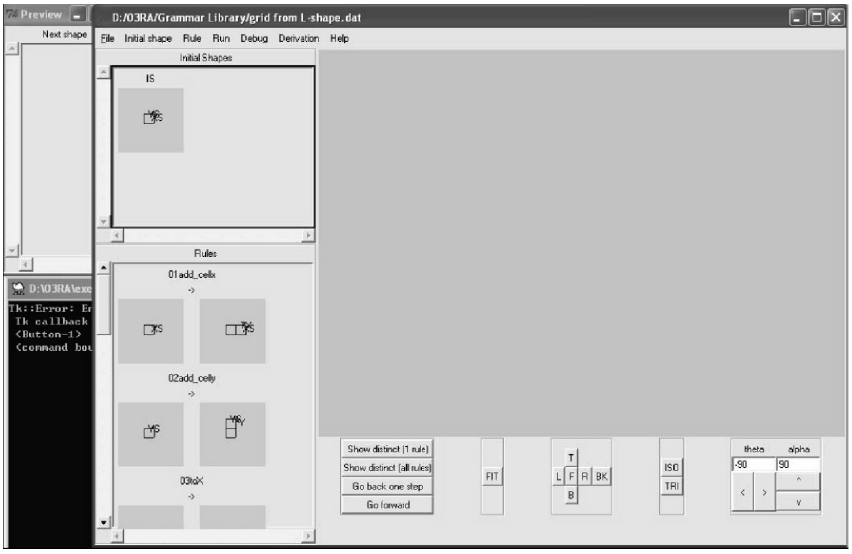

Figure 1. The main window, which contains a scrollable display of initial shapes (upper left), a scrollable display of rules (lower left), and a display of the current shape (large gray area). The shapes seen here consist of straight lines and labeled points. At the lower right are buttons to run the grammar and to control the display of the current shape. At the left, partially obscured, are the preview window, which contains a scrollable display of possible next shapes, and the console window, which displays the system's operations.

3.2.2. Editing shapes

Given our stated guidelines, our ideal shape editor would allow users to draw shapes, construct rules, and assemble grammars by directly manipulating those objects. For instance, users would be able to edit a shape wherever they see it, for example, on the left side of a rule in the rule list. And the editor would be as powerful and easy to use as the commercial applications that users are used to.

However, with the resources at hand, this is difficult. Instead, we implement two shape editors – one internal and one external – with complementary advantages and disadvantages.

The internal shape editor. The internal editor appears in a pop-up window when users select a shape and an edit command. When users are finished editing, the window is closed. Thus the principal advantage of the internal editor is that it is embedded in the generate-test work flow. Users can perform basic drawing operations: draw, move, and delete lines, rectangles, and labeled points. This also is handy for users who do not have other drawing software. However, its limited capabilities make it difficult to draw complex shapes.

The external shape editor. The external shape editor is an Autocad applet. With it, users can create new shapes and rules, edit them, and import and export them to and from Grammar Environment. Its main advantages are Autocad's power and ease of use in creating and editing shapes, especially for experienced users. In addition, users can import and export shapes in DWG and DXF formats, which allows them to integrate work with processes both up- and downstream. For example, shapes produced elsewhere can be imported into the grammar, and shapes produced by the grammar can be exported for rapid prototyping.

An important disadvantage of the external editor is that it introduces a second application into the work flow. Users must divert their attention to switching between applications and transferring domain objects back and forth. This is distracting and clearly degrades both the generate-and-test cycle and direct manipulation.

3.3. RUNNING A GRAMMAR

As mentioned above, we are making a system for users who use grammars to create satisfactory designs. So we think that users run a grammar to generate designs that they can evaluate. Each run iteration is what Schön (1987) calls an "on-the-spot experiment". The results of the experiment motivate users either to choose one design for further transformation (by running the grammar again) or to modify the grammar (by editing it). With this in mind, we provide three types of interaction.

First, given a current (or initial) shape, users click one button ("Show next shapes – all rules"), the system calculates and displays all possible next shapes, and the users evaluate the shapes.

Or, if users are interested in only one rule, they select that rule and click one button ("Show next shapes – one rule"), and the system calculates and displays all possible next shapes.

A third possibility is a derivation that involves a sequence where each shape has only one possible next shape. This is called a deterministic chain, and there is nothing for users to do until the chain branches, i.e., until there are several next shapes for users to choose among. In this case, users click one button ("Show deterministic chain"), and the system generates each successive next shape until the branch point, when it stops and displays the possible next shapes.

Still other interactions are possible. For example, the user may be interested in a particular subarea of the current shape and want to have the system limit its search to that subarea (Tapia 1999; Liew 2004). We save this for future versions of the system.

In addition, users can move backward and forward in the derivation between the initial shape and the last current shape.

3.4. SWITCHING BETWEEN EDITING AND RUNNING A GRAMMAR

Grammar Environment is modeless. The initial shapes, rules, and current shape are always displayed. If users are neither editing nor running the grammar, then the system is standing by for either an edit or a run command. Thus, users can move freely through the generate-and-test cycle.

4. Implementation

Since our ideas are more about user interaction and less about technical shape grammar functionality, we chose to build on the system by Chau et al. (2004), whose engine provides essential capabilities, namely support for U_{03}, U_{13}, and V_{03} algebras (i.e., points, lines, and labeled points in 3-space) and support for emergence. Their system was implemented in the Perl programming language, with the Perl Data Language (PDL) used for matrix manipulation, and Perl/Tk for the graphical user interface.

We made two major modifications to their system. One was to implement the graphical shape editor. The other was to improve the run interaction. Both were coded in Perl, tightly coupled, and integrated with the original engine. This combined script was then compiled into a Windows-based executable file using the PAR model in CPAN.

4.1. THE GRAPHICAL SHAPE EDITOR

Chau et al.'s system graphically displays only two domain objects: the current shape and one rule. Users create and edit domain objects by creating and editing a text file. We created a graphical interface for this text file, and made a package of graphical interaction methods mimicking desktop file operations, such as open, close, rename, and delete. In addition, we created the pop-up editor for drawing and editing shapes directly.

4.2. INTERPRETER

In terms of running a grammar, Chau et al.'s system does two main things. First, given 1) a rule A ? B, where A and B are shapes, 2) the current (or initial) shape C, and 3) a transformation t specified by the user, the system tests whether the shape A, under the transformation t, is a subshape of the shape C, i.e., whether $t(A) = C$. And second, if so, it calculates and displays the corresponding next shape $C? = [C - t(A)] + t(B)$.

The input to the first task is two sets of point triples: one from the left shape A and one from the current shape C. The points in the triples are labeled points,

end points, or intersections of lines. The transformation t is implied by the two triples, and the triples are input by the user. In other words, the user specifies the transformation t, and the system calculates and displays the next shape C, if there is one.

For us, most of this work is bookkeeping, and our system handles it in the background in the following way. It generates all relevant pairs of point triples, removes duplicates, and feeds the remaining pairs to the engine. It collects all the resulting next shapes and displays them in the preview window. The user can then choose one to replace the current shape, run a different rule, or edit the grammar.

5. Discussion

The interaction model that we have described has not previously been applied to shape grammar development or implemented in any shape grammar system. We have developed Grammar Environment as a prototype system for assessing the value of this interaction model.

We tried out Grammar Environment informally with a range of users: shape grammar novices and experts; high school, university, and graduate students; and teachers. After taking a short time to become familiar with the system, they seemed to spend most of their time and effort on their shape grammar work, not on "bookkeeping."

The problems as well suggest that the interaction model is appropriate. For instance, because of a quirk in the GUI toolkit, an initial shape or rule can sense clicks only in a small part of the area that it occupies on the screen. But users clicked all over that area, apparently expecting direct manipulation. This suggests to us that our attention to domain objects and interaction is well-placed, and that it is our implementation which is imperfect.

Nevertheless, and needless to say, there are many things which Grammar Environment does not do, especially when measured against an ideal, fully developed system. Such a system might, for example, support users developing grammars with large numbers of rules organized in groups using labels.

How do we see Grammar Environment with respect to such a standard? Labels are one technical feature of shape grammars, and are supported by Grammar Environment. There are other technical features, such as descriptions and parallel grammars (Stiny 1981; 1990), which are potentially powerful tools but have not, to the best of our knowledge, been supported by previous systems. Unfortunately, we cannot implement all of these features at once, and can only speculate and strategize about the sequence of implementation.

Dealing with large numbers of rules is a serious issue, already mentioned by Gips (1999, 5). Our first step has been to test the interaction model. The

next step is to deal with interface issues, such as the apparent contradiction between direct manipulation and a large number of rules.

Ultimately, a system like Grammar Environment should help users create designs with grammars. Grammar Environment has not yet been put to such a test, and it probably does not accomplish this goal. However, we believe that the questions we have raised – such as how users do their work, in terms of domain objects and operations – will help researchers develop such a system. For example, it would help frame a basic interface question: what information does the user want to see, and how can it be presented on the screen in an "intuitive" way? Indeed, it would help us think about what we mean by intuitive. Further work along these lines would move us closer to "purely visual computation."

Acknowledgements

This work was supported by a Competitive Earmarked Research Grant from the Hong Kong Research Grants Council, which we acknowledge with thanks.

References

Celani, G.: 2001, MIT-MIYAGI workshop 2001: an educational experiment with shape grammars and computer applications, *International journal of design computing,* http://faculty.arch.usyd.edu.au/kcdc/journal/vol3/celani/abstract.htm, accessed 20 November 2008.

Chase, S.C.: 1989, Shapes and shape grammars: from mathematical model to computer implementation, *Environment and planning B: planning & design,* **16**, 215–42.

Chase, S.C.: 2002, A model for user interaction in grammar-based design systems, *Automation in construction,* **11**, 161–72.

Chase, S.C.: 1987, "Computer implementations of shape grammars." MA thesis, University of California, Los Angeles.

Chau, H.H. Chen, X. McKay, A. and de Pennington, A.: 2004, Evaluation of a 3D shape grammar implementation, in J.S. Gero (ed.), *Design computing and cognition '04,* Kluwer, Dordrecht.

Design synthesis and shape generation: 2008, http://www.engineering.leeds.ac.uk/dssg/, accessed 12 December 2008.

Duarte, J.P.: 2001, "Customizing mass housing: a discursive grammar for Siza's Malaguiera houses." PhD dissertation, Massachusetts Institute of Technology, Cambridge, Mass.

Flemming, U.: 1987a, More than the sum of parts: the grammar of Queen Anne houses, *Environment and planning B: planning & design,* **14**, 323–50.

Flemming, U.: 1987b, The role of shape grammars in the analysis and creation of designs, in Y.E. Kalay (ed.), *Computability of design,* John Wiley, New York.

Gips, J.: 1999, Computer implementation of shape grammars, paper read at NSF/MIT Workshop on Shape Computation, at Massachusetts Institute of Technology.

Knight, T.W.: 1991, Designing with grammars, in G.N. Schmitt (ed.), *Computer aided architectural design futures: education, research, applications,* Vieweg, Braunschweig.

Krishnamurti, R.: n.d., SGI user manual, n.p., n.p.

Krishnamurti, R. and Giraud, C.: 1986, Towards a shape editor: the implementation of a shape generation system, *Environment & planning B: planning & design,* 13, 391–404.

Li, A.I.: 2002, A prototype simulated interactive shape grammar, in K. Koszewski and S. Wrona (eds.), *Design e-ducation: connecting the real and the virtual, Proceedings of the 20th conference on education in computer aided architectural design in Europe, Education in Computer Aided Architectural Design in Europe,* Warsaw.

Liew, H.: 2004, "SGML: a meta-language for shape grammars." PhD dissertation, Massachusetts Institute of Technology, Cambridge, Mass.

McCormack, J.P. and Cagan, J.: 2006, Parametric shape grammar interpreter, US patent 7,050,051 B1.

McGill, M.C.: 2002, Shaper2D: visual software for learning shape grammars, in K. Koszewski and S. Wrona (eds.), *Design e-ducation: connecting the real and the virtual, Proceedings of the 20th Conference on Education in Computer Aided Architectural Design in Europe,* eCAADe, Warsaw.

Mirel, B.: 2004, *Interaction design for complex problem solving: developing useful and usable software,* Morgan Kaufman, San Francisco.

Piazzalunga, U. and Fitzhorn, P.: 1998, Note on a three-dimensional shape grammar interpreter, *Environment and planning B: planning and design,* 25, 11–30.

Romão, L.: 2005, SGtools: a computer tool for exploring designs with set grammars, in B. Martens and A. Brown (eds.), *Learning from the past: a foundation for the future: special publication of papers presented at the CAAD futures 2005 conference held at the Vienna University of Technology, Vienna, Austria, on June 2002–22, 2005,* Österreichischer Kunst- und Kulturverlag, Vienna.

Schön, D.A.: 1983, *The reflective practitioner: how professionals think in action,* Basic Books, New York.

Schön, D.A.: 1987, *Educating the reflective practitioner: toward a new design for teaching and learning in the professions,* Jossey-Bass, San Francisco.

Soman, A. Padhye, S. and Campbell, M.I.: 2003, Toward an automated approach to the design of sheet metal components, *Artificial intelligence for engineering design, analysis and manufacturing,* **17**, 187–204.

Stiny, G.: 1981, A note on the description of designs, *Environment and planning B: planning & design,* **8**, 257–67.

Stiny, G.: 1990, What is a design?, *Environment and planning B: planning & design,* **17**, 97–103.

Stiny, G. and Gips, J.: 1972, Shape grammars and the generative specification of painting and sculpture, *Information processing,* **71**, 1460–65.

Tapia, M.: 1999, A visual implementation of a shape grammar system, *Environment & planning B: planning & design,* **26**, 59–73.

Wang, Y. and Duarte, J.P.: 2002, Automatic generation and fabrication of designs, *Automation in construction,* **11**, 291–302.

Wu, Q.: 2004, Bracket teaching program: a shape grammar interpreter, paper read at GCAD '04, Generative CAD Systems Symposium, at Carnegie Mellon University.