

SHAPER 2D

An Interactive Shape Grammar Interpreter

MIRANDA C MCGILL

Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract. *Shaper2D* is a program that promotes the use of computers for learning about computational design by employing a dynamic, interactive interface.

1. Introduction

Shaper2D is a program that promotes the use of computers for learning about computational design. This program, created by a designer rather than a programmer, was developed to employ an intuitive, visual interface that encourages a “learning by designing” approach to shape grammar education.

Shaper2D is not the first application created for shape grammar exploration. Examining two previous computer implementations of shape grammars situates *Shaper2D* in its intended context. These software applications for exploring two-dimensional and three-dimensional shape grammars provide powerful tools for learning fundamental concepts, but their interfaces are not conducive to experimenting with shape grammars. While these programs demonstrate simple and accurate representations of the actual shape grammar computation, they do not provide an appropriate interface for designers.

2. Designing Design-Mediating Software

The development of *Shaper2D* responded to the need for an improved human-computer interface for exploring shape grammars. It has been widely agreed that hand computation is essential for a comprehensive understanding of the concepts involved in shape grammars. However, using the computer can facilitate designing with shape grammars, as designs tend to get very complicated very quickly. *Shaper2D* makes the shape grammar process as transparent as possible through interactivity and dynamic response. Designing

becomes more comprehensible; designers are encouraged to persevere with an idea rather than give up due to frustration or impatience. *Shaper2D* capitalizes on the advantages of computer power over hand computation, especially when it comes to more complex design generation.

2.1. A PROGRAMMER'S APPROACH TO PROGRAMMING

It is important to consider how a programmer develops a program, in order to contextualize the different approach taken with *Shaper2D*. While the focus on an intuitive interface and relevant functionality may seem an obvious approach to designing an application, this is not always the case. Often the "worse is better" philosophy, as discussed by Richard Gabriel, is used in software development.

"...It is more important for the [program's] implementation to be simple than the interface. Simplicity [of code] is the most important consideration in a design." (Gabriel, 1999)

While not all programmers fall prey to this idea, it is still the most typical approach. Essentially, "worse is better" programming sacrifices the interface and usability for simplicity of code design, that is:

- Simple code is more important than a usable interface, in other words, the programmer's convenience is more important than the user's convenience.
- The user is expected to learn to think about the problem the way the programmer thinks about the problem.
- The interface should reflect the internal structures of the program since they are the most fundamental description of the problem.

This approach usually results in a program with enormous functionality, which is difficult to use and has an unintuitive interface.

When a user has difficulty using a program, it is typical that they blame themselves rather than look to the inadequacies of the software they are using. This should not be the case.

2.2. A DESIGNER'S APPROACH TO PROGRAMMING

2.2.1. Interface

From the outset, *Shaper2D* was designed with the end user in mind. Rather than programming an application and then tacking on an interface at the end, developing a dynamic, intuitive, and simple interface was the driving force behind the program design. Three rules were steadfastly adhered to:

- Never choose to sacrifice the usability of the program for the sake of making the code simpler.

- If a help manual is required for learning to use the program then the interface is flawed.
- Remember that the end user is a designer not a programmer so the program should be intuitive, visual, and responsive.

Interfaces developed by programmers have a tendency to mirror the internal structure of a program without consideration for the end user. This approach is borne out of convenience and often results in powerful programs that sometimes include a token interface as an acknowledgment for the end user, but which usually require a lengthy help manual in order to understand how to run them.

2.2.2. *Functionality*

The deliberate restrictions placed on *Shaper2D* could be described as additional functions in their own right. *Shaper2D* was designed to force the user to use the program in a certain way in order to learn basic shape grammars. Ensuring simplicity of use made the interface design more problematic and restricting the program's functionality, such as forcing the user to manipulate the rule geometry and labels separately, complicated programming.

Programmers are often reluctant to restrict what a program can do, regardless of whether a simpler, less feature-heavy—or restricted—program would be more beneficial to the end user.

3. A New Shape Grammar Interpreter

3.1. PREVIOUS SHAPE GRAMMAR INTERPRETERS

To frame the motivation behind the development of *Shaper2D*, it is helpful to consider two recent shape grammar interpreters, *3D Shaper* (Wang 1999) and *GEdit* (Tapia 1999). These programs were developed to expedite the process of design using shape grammars and have been used on occasion for classroom teaching. However, their use in the design studio has been limited. Various reasons have been cited for this, including a completely unrestricted workspace that is only truly useful to more advanced uses of shape grammars, delayed feedback, and a designer-unfriendly interface. These are not criticisms of the software, as each application has an audience in current shape grammar pedagogy and research. However, the deficiencies listed above are those that *Shaper2D* seeks to address.

3.1.1. *3D Shaper*

3D Shaper was developed by Yufei Wang in 1999 to simulate the designs produced by manipulating wooden “Froebel” blocks. Compiled for the

UNIX/SGI operating system, it is a very powerful program that enables the user to experiment with three-dimensional grammars in ways that are difficult to visualize mentally, and often impossible to perform physically.

3D Shaper implements a static interface that requires the user to type in numerical parameters to determine the size and type of shapes, as well as the spatial relation between the shapes. The program then applies the rules and produces an *Open Inventor* file containing the three-dimensional design output. The user opens this file in an appropriate viewer for review. However, if a design change is needed then the user must return to the *3D Shaper* window, input the necessary shape and/or rule revisions, and run the program again in order to generate a new *Open Inventor* file.

3.1.2. *GEdit*

The first computer implementation of shape grammars that included a viable user interface was the Apple Macintosh-based *GEdit* developed by Mark Tapia (1999). This interpreter is used for generating two-dimensional, non-parametric shape grammars and emphasizes that the "...drawing is the computation" (Tapia 1999). Tapia sought to minimize user distraction, both in terms of obscuring parts of the program and perverting the design flow, by limiting the use of drop-down menus and dialog boxes. He instead implemented an object-specific radial menu, which only appears when needed to ensure that the user's focus is not removed from the design process. This is a program for shape grammar experts, since it is very open-ended and allows for the free exploration of almost any kind of non-parametric, two-dimensional shape grammar.

3.2. WHAT IS DIFFERENT ABOUT SHAPER2D?

Previous interpreters developed to expedite the process of design using shape grammars lacked an accessible user interface, were operating system-specific, were too general, or did not account for the dynamic, interactive environment sought by a designer.

With *3D Shaper*, the process of first inputting numbers into a static interface then opening up a separate viewer in order to see the design does not tally with the designer's habitual design process. While the parameters entered by the user fully describe the desired grammar, they do so in a way reflective of the internal computations of the software as opposed to the way the designer would manipulate shapes naturally. The interface was written for the ease of the programming, not for the benefit of the user. For many architects, conceptual designing is fluid and dynamic—*3D Shaper* is an invaluable tool for experimenting with three-dimensional shape grammars, but the interface is not analogous to the design process and hence does not facilitate quick, experimental design.

GEdit, while introducing the notion of an elegant interface, is a generalized, unrestricted example of shape grammar software. It is suitable for advanced shape grammarians wishing to visualize a particular derivation, but is impractical for novice users who require a more constrained, structured environment or for users wishing to use shape grammars for real-world design problems.

Shaper2D is a visually dynamic shape grammar application for generating designs using very restricted kinds of shape grammars. Changes to the grammar are immediately reflected in the design. It was written to overcome the platform-specific limitations imposed by previous interpreters. The application has been run successfully under many major operating systems (Microsoft Windows, Mac OS and Linux), and the applet runs under any web browser capable of running JavaTM2 (such as *Mozilla*, *Internet Explorer*, and *Opera*).

4. The Development of Shaper2D

4.1. INTERFACE AND INTERACTIVITY

The interface is the key aspect of *Shaper2D*'s development. It had been noted by several researchers in the field of shape grammars that while useful implementations of shape grammars have been developed, little attention had been focused on the interface (March, 1997; Knight, 1998; Tapia, 1999). This was a surprising oversight given the anticipated audience.

Visual seductiveness is very important to designers. So naturally, when developing *Shaper2D*, many graphic and interface design issues had to be considered. In particular, the interface needed to embrace the inherent visual uniqueness of shape grammars. The decision to exclude any need for manual typing-in of parameters or instructions was made at the program's inception—it was important to develop a program based on an interactive, visual interface to generate designs.

The most comprehensible way of doing shape grammars is when the computation is done by hand—indeed, hand computation using tracing paper played an important role in the development of *Shaper2D*. However, because the actual computations are concealed the program could be described as a “black box”: the rule application process is hidden from the user. Physically flipping and rotating drawings on tracing paper, or manipulating three-dimensional wooden blocks, personalizes the act of performing the spatial transformations by exposing the designer to direct contact with the processes involved.

The dynamic interface endeavors to offset the inherent black box disadvantages by informing the designer of the effect of her actions immediately. The user can experiment quickly with different rules and

iterations in order to compare the outcomes of different ideas and grammar applications.

4.2. HOW SHAPER2D WORKS

Shaper2D was developed as both a Java applet (figure 1) and a stand-alone application (figures 2 and 3). This decision was based on the advantages of using the Internet for enhancing portability and cross-platform compatibility.

The *Shaper2D* user interface is composed of a menu bar, five panels—“Spatial Relation” (1 and 2), “Rule” (1 and 2), and “Design”—and two toolbars for selecting different shapes. The default shapes are a rectangle, square, equilateral triangle, and isosceles triangle. The “Spatial Relation 2” and “Rule 2” panels are grayed out by default when the program is first run to emphasize the difference between using one rule and two rules.

Whenever the user manipulates a shape—which can be done in both the “Spatial Relation” and “Rule” panels—the other active panels update in real-time to reflect the change. This enables the user to get a dynamic response to her actions. Similarly, when the position of a label is changed the “Design” panel updates immediately.

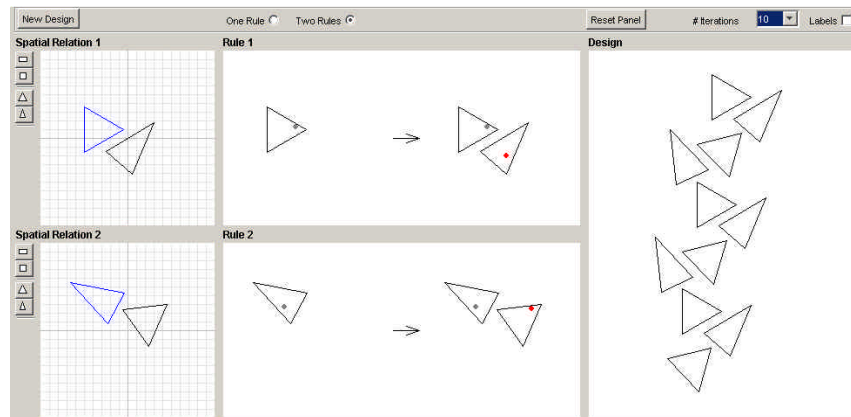


Figure 1. The Shaper2D Applet (version 1.0)

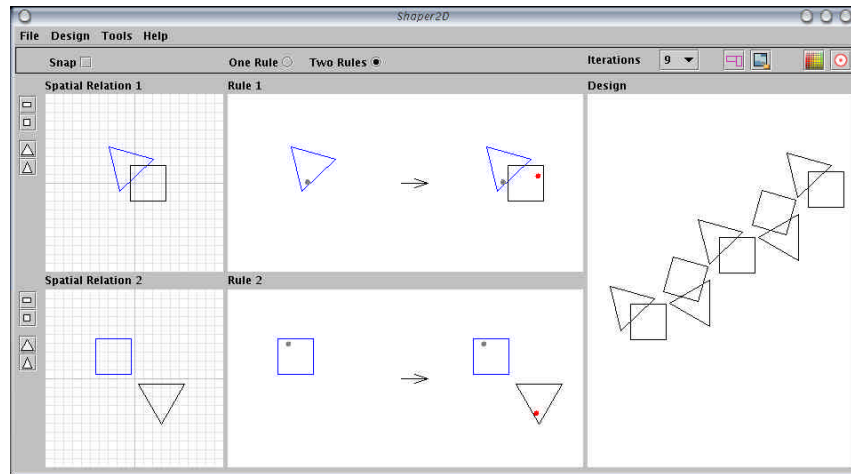


Figure 2. The Shaper2D Basic Application (version 3.1)

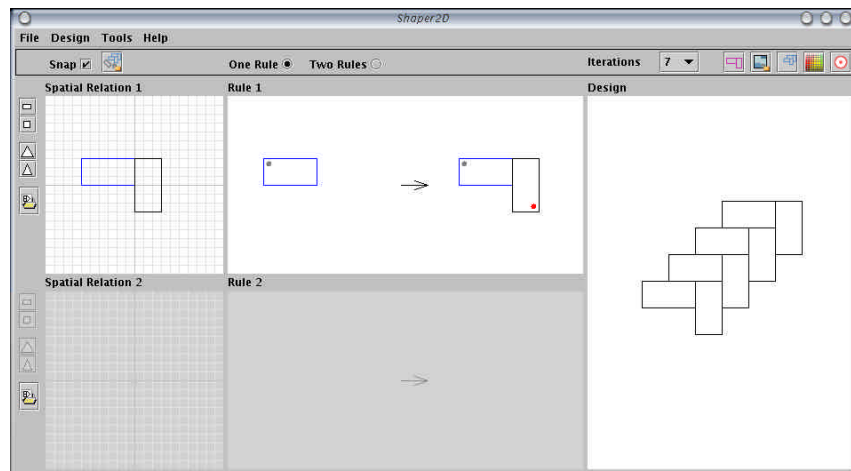


Figure 3. The Shaper2D Advanced Application (version 3.1)

The dynamic update is a key feature of the program and enables the user to see the implications of each design move instantaneously. The “Design” panel updates dynamically according to changes made in the “Spatial Relation” and “Rules” panels. Additionally, to review the construction of a design, the user has the option to display the labels (figure 4)—with the most recent shape addition highlighted in red. The label display feature is particularly helpful when the user gradually increments the number of iterations displayed, in other words, performs a ‘walk through’ of a design’s construction.

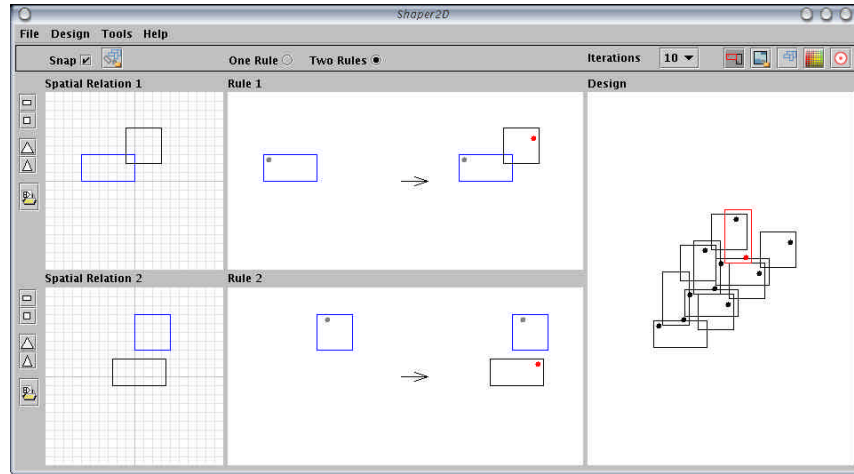


Figure 4. Label Display

4.2.1. Shaper2D Applet

The advantage of the applet (<http://www.mit.edu/~miri/shaper2d>) is that it can be run directly from an Internet browser. This allows users who may not be able to download and run the application to have access to the software. However, this version has limited functionality due to the security restrictions imposed on applets.

4.2.2. Shaper2D Application (Basic & Advanced)

The *Shaper2D* application is a more feature-rich tool than the applet. The main difference between the applet and application is the ability to save and load: the user is able to save and load rules and can export a design in the DXF file format for importing into another CAD application (such as *AutoCAD*), thus increasing the pedagogical and design value of the software.

The user is also able to import a background image into the design window, in order to place a design in a given context. In turn, the user can change the line color of the design to improve visibility (figure 5).

Two versions of the application were created, one for novices and the other for more experienced users. The decision to develop two versions was in response to *Shaper2D* being designed as a pedagogical tool. Rather than overwhelm beginners with an over-complex interface, an advanced version was created which includes additional features. Thus, when the user is confident enough to progress to more complex shape grammar investigations she can change from the basic to the advanced interpreter.

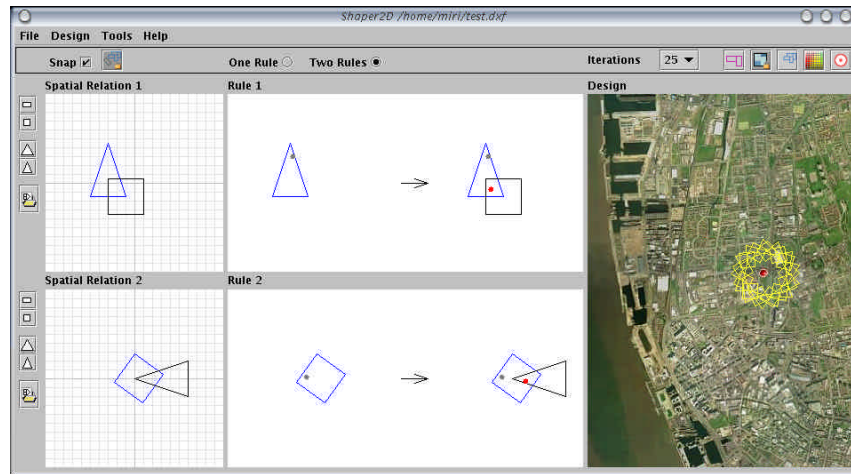


Figure 5. Importing a background image and changing line color

4.2.3. Shaper2D Application (Advanced)

This version has the advantage of allowing shape substitution (figure 6). This feature enables the user to import a substitute shape or design to be used in place of one of the four preexisting, or “reference”, shapes. The substituted shape retains the symmetry group of the reference shape, thus allowing the user to explore the implications of different symmetries.

More complex designs can be developed providing the user an opportunity to investigate how the different symmetries of the reference shape affect the rule application.

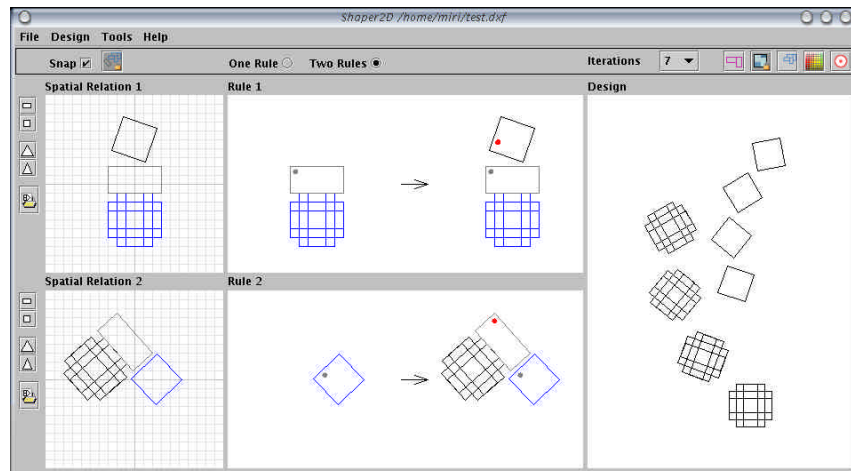


Figure 6. Shape Substitution

5. Conclusion

Shaper2D was developed by a designer rather than a programmer, and strives to promote a more fluid approach to software design for designers. Previous shape grammar programs—designed by programmers—do not offer the user the opportunity to quickly explore many designs, so fail to offer any advantages over hand computation. They tie the user into the “plan, calculate, examine, redesign” method of designing—in other words, an iterative and algorithmic approach—where each stage is seen as a distinct step. The user loses the ability to explore and experiment.

Shaper2D encourages exploration by providing the user with instant feedback and a purely visual, designer-friendly interface. Designing becomes more comprehensible; designers are encouraged to persevere with an idea rather than give up due to frustration or impatience. *Shaper2D* capitalizes on the advantages of computer power over hand computation, especially when it comes to more complex design generation.

Acknowledgements

Thanks to: Terry Knight, Oliver Dial, William Mitchell, Edith Ackermann, Mark Tapia, and everyone who has taken the time to test *Shaper2D* and provide me with valuable feedback.

References

- Gabriel, RP: 1991, Lisp: Good News, Bad News, How to Win Big, *Lucid Inc*
- Galitz, W.O.: 2002, *The Essential Guide to User Interface Design*, John Wiley & Sons, New York.
- Knight, TW: 1998, Shape Grammars, *Planning and Design, Anniversary Issue*: 86- 91.
- Knight, TW: 2000, Shape Grammars in Education and Practice: History and Prospects, *International Journal of Design Computing* **2**.
- Loy, M., Eckstein, R., Wood, D., Elliott, J. and Cole, B.: 2003, *Java™ Swing*, O’Reilly & Associates Inc., Sebastopol, MA.
- March, L: 1997, A White Paper: Shape Computation. In *Proceedings of the Symposium on Design and Computation*, University of California: Los Angeles.
- McGill, MC: 2001, *A Visual Approach for Exploring Computational Design*, SMArchS Thesis, Department of Architecture, Cambridge, MA: Massachusetts Institute of Technology.
- Resnick, M, Bruckman, A and Martin, F: 1996, Planos Not Stereos: Creating Computational Construction Kits, *Interactions* **3**(6).
- Tapia, MA: 1999, A Visual Implementation of a Shape Grammar System, *Environment and Planning B: Planning and Design* **26**: 59-73.
- Wang, Y: 1999, *3D Architecture Form Synthesizer*, SMArchS Thesis, Department of Architecture, Cambridge, MA: Massachusetts Institute of Technology.