QSHAPER

A CAD Utility for Shape Grammars

Tuğrul Yazar[1], Birgul Colakoglu[2]
Yıldız Technical University, Design & Computation Unit, Turkey
http://www.bot.yildiz.edu.tr
[1]turuly@yahoo.com, [2]bigi@alum.mit.edu

This paper presents an ongoing research about a new computer-aided design tool named QShaper (QS). It is developed within a visualization software as a scripted utility. It aims to assist designers and students in creating and exploring rule-based designs.

Keywords. *Computational design tools; shape grammars; scripted utility; graphical user-interface.*

Introduction

Shape Grammars, invented by Stiny and Gips (1972) have shown to be powerful rule-based design systems. They have well developed mathematical foundation that provides formal mechanism for their computer implementations. Gips (1999) classifies possible computer tools that implement Shape Grammars in four groups: 1-Generation, 2-Parsing, 3-Inference and 4-Computer-aided design tools for Shape Grammars.

Several tools for Shape Grammars have been developed such as Shaper2D (McGill, 2002), GEdit (Tapia, 1999), and 3DShaper (Wang and Duarte, 2002). These applications explore two-dimensional and three-dimensional Shape Grammars and are powerful tools for learning its' fundamental concepts. Most of these tools have been used for education. However, their use in the design practice has been limited.

The fourth type of tools would assist (users) designers creating a Shape Grammar by providing sophisticated functions. This type of an application would be a plug-in for a computer-aided design software that would use Shape Grammars to help the designer (Gips, 1999). AutoGrammar (Celani, 2001), developed as plug-in under a common CAD software is of such kind that is used in design workshops as an auxiliray design tool.

This paper describes a new Shape Grammar tool, QShaper (QS) developed within a common visualization software as a scripted utility. QS is using Shape Grammars to help designers explore computationally generated rule-based design variations. It functions both as an experimental tool for practical use, and an educational platform for explaining the basic mechanics of Shape Grammars.

QS is not claiming to be a complete solution for the specific design paradigm. Therefore, it is open to add-ons or changes in its structure.

Quick Shaper

QShaper means "Quick Shaper", focusing on the user-friendliness. Main concern of this research is to develop a powerful and easy-to-use tool that provides fundamentals of Shape Grammars.

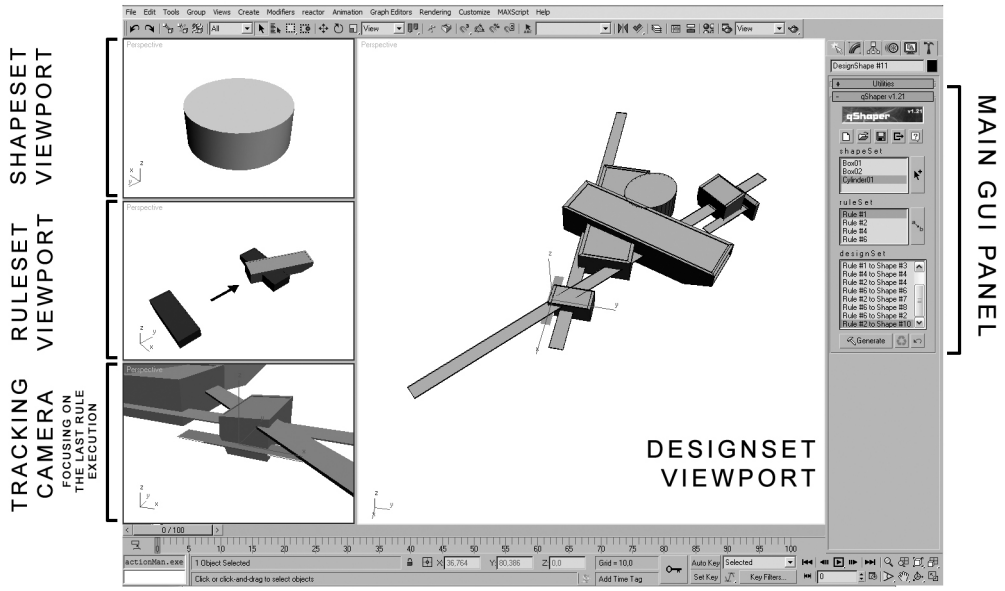


Figure 1
GUI layout of QS on a single-monitor computer

QS operates on maximum user-interaction. This involves a user choosing which rule to apply and how to apply it in each step of computation. In this case, the user's role approximates the role of a designer (Knight, 1999). In order to control different groups (shapes, rules, and the design) at the same time, graphical user-interface (GUI) of such applications require several different visual areas. In QS these areas are defined as;

1. A set of 3D views, showing the shapes and the rules, allowing a designer to see and manipulate them (figure 1). This structure of GUI viewports reflect the algorithm of QS.
2. A main 3D view, showing the current design composition, the desig nSet. In QS, this is central and the largest viewport (figure 1). A designer cannot manipulate on this viewport objects directly as it is controlled by the designSet section of the GUI panel (figure 2).
3. A GUI panel, including icons of the new functions added to the software (figure 2). QS maximizes

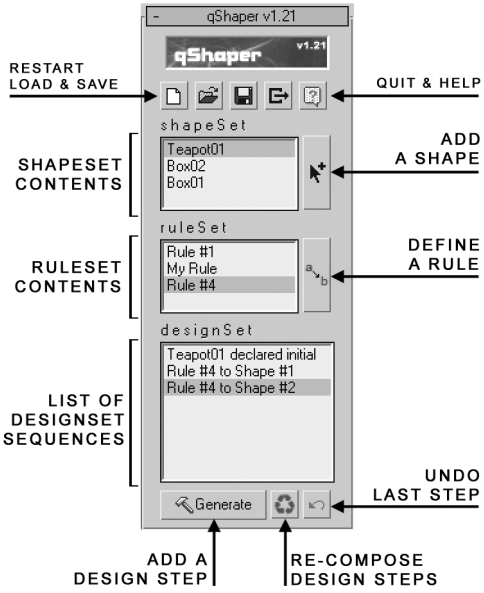


Figure 2
Main GUI panel of QS

visual interaction with the designer, operating on "see and point" method. Thus, a designer can use all functions of QS without any keyboard input.

The algorithm and the design strategy

QS is developed as a scripted utility using the software's built-in scripting language. The software is completely functional while QS is running (with some exceptions). The algorithm of QS introduces two collections of objects (shapeSet, ruleSet), and a sequential structure (designSet) to record and manipulate the design process (figure 3). Although these collections are named as "set", they are not closed structures. QS allows a designer to define shapes and rules, and to manipulate them in the design process synchronously.

ShapeSet represents the vocabulary of objects to be used in a grammar. QS does not include any object creation functions as the software already provides them. Any object created within the software can be assigned to shapeSet, regardless of its' class or geometry (2D & 3D primitives, objects with "modifiers", NURBS surfaces, boolean objects, lights, cameras etc.). There is no limit on the number of different objects that can be assigned to this collection. An object may be removed from the shapeSet if it is not used in any shape rule. In order to start a generative process, at least one object must be assigned to shapeSet and one rule must be defined in ruleSet.

RuleSet allows designers to define rules through spatial relations. Each rule in the ruleSet consists of two shapes instanced from shapeSet; the source shape represents left-hand side and the target shape represents right-hand side of the rule. Shape rules may be defined as additive (adding the target shape, keeping the source shape) or replacing (removing the source shape with target shape). There is no limit on the number of different rules that can be defined. A rule can be removed from the set, if it is not executed at any step of designSet.

QS operates on the local-space transformation matrix (position, rotation and scale) which is included in the software. Connecting source shape as the

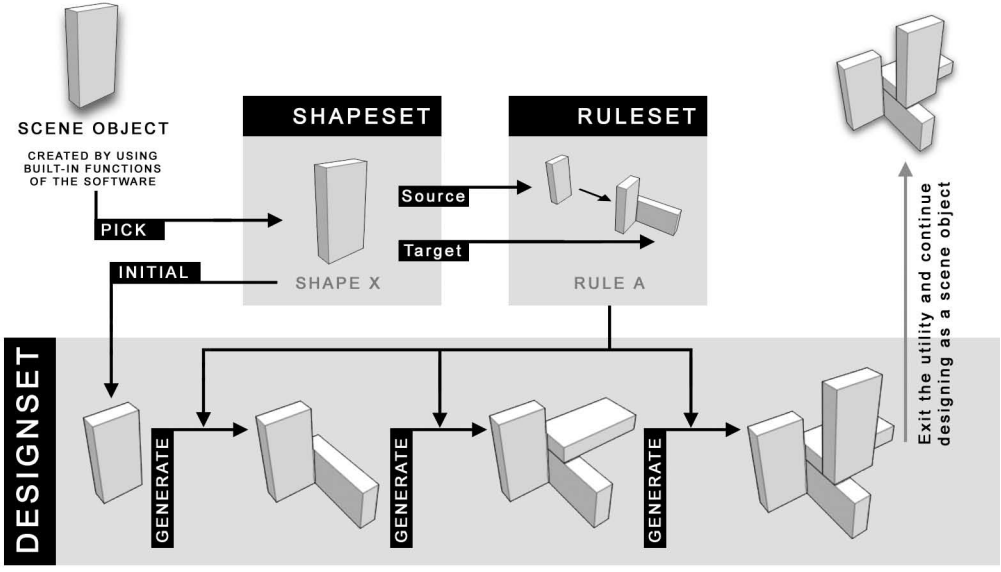


Figure 3
Basic generative process of QS, introducing object sets and sequential structure

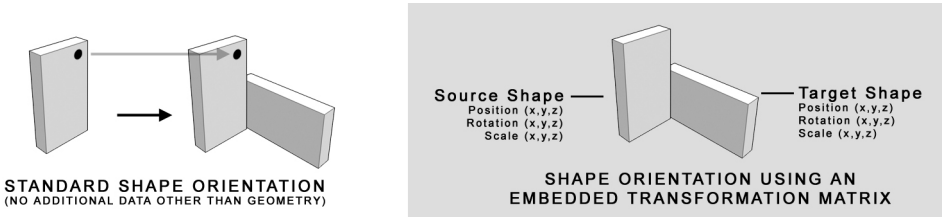


Figure 4
On the left; traditional shape rule definition. On the right; shape rule definition of QS

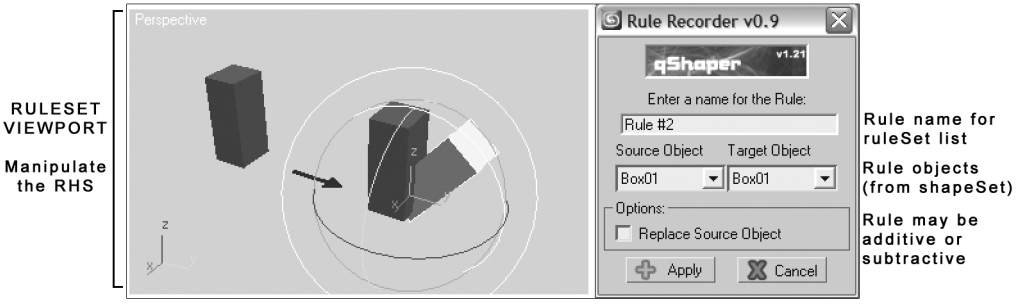


Figure 5
GUI of "Rule Recorder" in QS

parent object of target shape, their transformation matrixes will cooperate automatically. Using this technique, computation of a shape rule becomes a process of shape replacement (figure 4).

Creation of a shape rule brings additional functions and puts QS in a different mode. Thus, another GUI panel pops up (figure 5). After defining the shape rule, this panel closes and the new rule is added to ruleSet list. It is possible to change the rule by selecting it from the main GUI panel (figure 2) and manipulating it on the ruleSet viewport (figure 1) using built-in transformation functions of the software.

DesignSet represents the design space. Its' structure records an array of steps. These steps are generation sequences that designer creates and manipulates. Each member of this set includes an object selected from a previous step of designSet and a rule from ruleSet applied to it. It is possible to re-compose the designSet by applying the changes that a designer makes on shapeSet objects and/or shape rules. In order to help designer test different variations, sequences can be removed from the designSet. Also, the design sequences may be rewinded and re-executed step-by-step by the designer to see the process in a more perceivable way. Although the designSet is sequential, it is not necessary to generate shapes sequentially. A designer may choose to use any object created before in designSet (figure 6, 8).

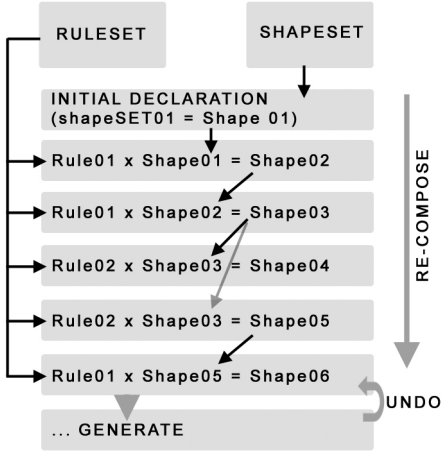
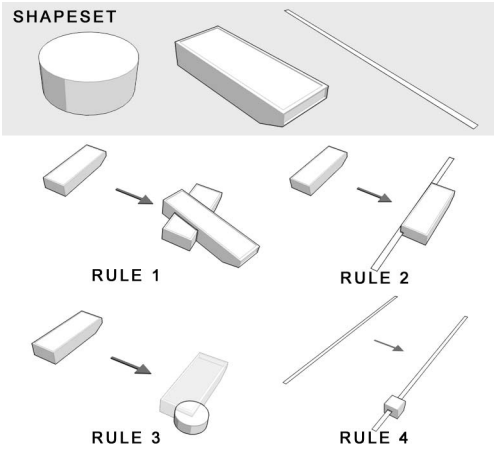


Figure 6
Logic of designSet recording

Figure 7
ShapeSet and ruleSet

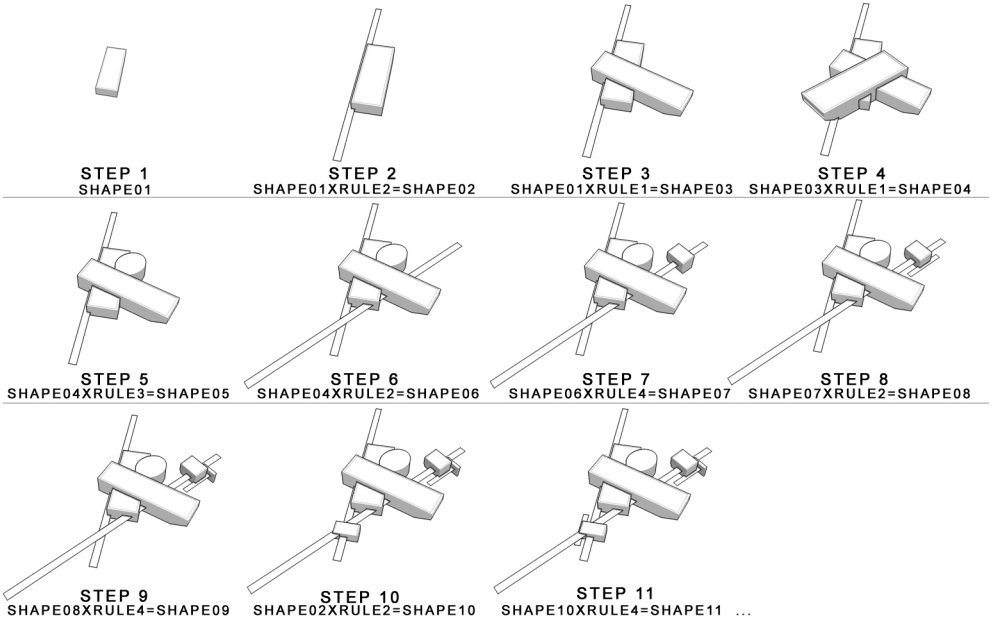


Testing QS

Scope of the QS test includes the form-finding process of a building complex. The test focuses on controlling the design process of the tool rather than creating a final design.

The test started by assigning some temporary shapes to the shapeSet. After defining some rules (figure 7), generation process started by declaring an initial shape from the shapeSet. Manipulating the shapes and rules, the design process immediately opens up new creative possibilities (figure 10, 11). Moreover, user may restart the process from the beginning, creating a different alternative using the same shapeSet and ruleSet (figure 9). All design sequences may be re-executed step-by-step by the designer (figure 8). Designer may chose to stop the process and continue form-finding without QS, or may save the whole process to a CAD file to restore and continue later.

Figure 8
DesignSet sequences



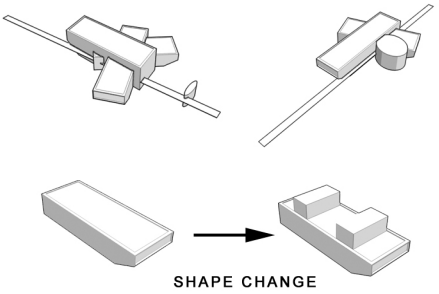


Figure 9
Some alternative designs created by using the same shapes and rules

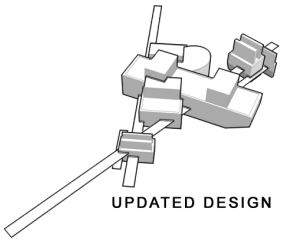
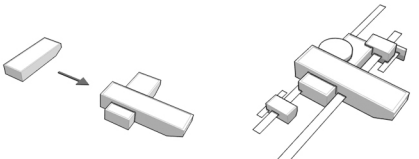


Figure 10
An alternative design created by changing the a shape and updating the designSet



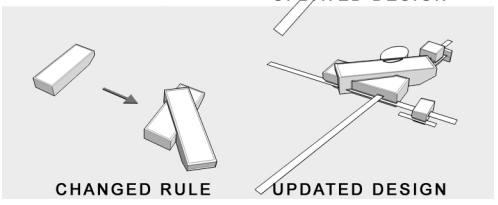


Figure 11
Some alternative designs created by changing a rule and updating the designSet

Conclusion and further research

Developmental potentials of QS includes the implementation of additional features the visualization software provides. Thus, using current version of QS as a basis, various specific add-ons can be developed. Some of these potentials are described below;

1. Emergent Shape Recognition: In order to maintain computational continuity in the generation process, emergent shapes are important. Using boolean techniques and a comparsion algorithm, emergent shape recognition module might be added to QS.
2. Time-based Features: In addition to designSet recording, QS might be improved using sophisticated animation features of the software, such as keyframing and inverse kinematics. These features might provide some analytical capabilities as well.
3. Automation Module: Instead of maximum user control, the system might provide a degree of automation. In this case, QS would allow users to define constraints and calculate multiple computations based on these given goals.

The tool described here is a part of an ongoing research project. According to first application tests, QS is stable and flexible enough to be used in quick form-finding exercises. Although a scripted utility is not a "real" stand-alone software, it is advantageous in terms of the balance between required programming effort and the expected benefit.

References

Celani, G., 2001, MIT-MIYAGI workshop 2001, http://faculty.arch.usyd.edu.au/kcdc/ijdc/vol03/papers/.

Gips, J.: 1999, Computer implementation of shape grammars, NFS/MIT Workshop on Shape Computation, http://www.shapegrammar.org/projects.html.

Knight, T.: 1999, Shape grammars in education and practice: history and prospects, International Journal of Design Computing 2.

McGill, M. C.: 2002, Shaper2D: visual software for learning shape grammars, in Proceedings of the 20th eCAADe Conference, Warsaw, pp. 148-151.

Stiny, G. and Gips, J.: 1972, Shape grammars and the generative specification of painting and sculpture, in C. V. Freiman (ed.), Information Processing 71, North Holland, Amsterdam, pp. 1460-1465.

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

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