Visualising and Editing Graphical Representations for Procedurally Generated Designs using Shape Grammars

Miguel Féo e Torres, Filipe Santos
ISCTE - Instituto Universitário de Lisboa (ISCTE-IUL)
Information Sciences, Technologies and Architecture Research Center (ISTAR-IUL)
Lisboa, Portugal
{miguel_feo_torres, filipe.santos}@iscte-iul.pt

Pedro Santana
ISCTE - Instituto Universitário de Lisboa (ISCTE-IUL)
Instituto de Telecomunicações (IT)
Lisboa, Portugal
pedro.santana@iscte-iul.pt

Abstract—In the context of the Alternative Shaper model, this paper proposes a novel tool to visualise and edit procedural generation rules and processes associated with shape grammars. With this tool, shape grammar users are able to, using a graphical user interface, create a design specification by editing atomic and compound shapes, shape grammar rules and processes. The tool also automatically generates designs that comply with the specification designed by the user.

Keywords—shape grammar, procedural content generation

I. INTRODUCTION

Shape grammars have been around for more than 40 years, since their first introduction by Stiny and Gips [1]. Since then, many applications have been found for shape grammars, mainly in architectural design and in the creation of graphical designs. Shape grammars are usually defined by two main sets: (1) a set of symbols which contains all the graphical information that a finalised object constructed by a shape grammar can be made of, and (2) a set of rules, which can be used to replace a shape that exists in a design, the input shape, by another one, the output shape. With shape grammars, designs are created through derivation, that is, the continuous application of a rule or rules in a design (see Fig. 1). Alternative Shaper [2] supplements shape grammars with the concept of process with the goal of automating the generation of complex designs. Processes may be comprised of shape grammar rules, constraints, and other usual procedural constructs.

To facilitate the specification of designs within the Alternative Shaper model, ideally by non-expert users, this paper proposes a novel visualisation and edition tool that allows shape grammar users to, using a graphical interface, create a design specification by editing atomic and compound shapes, shape grammar rules and processes.

To accommodate processes, key to procedural generation [3][4] and absent in older shape grammar editors [5][6], the tool is based on a novel visual language. The tool also automatically generates designs that comply with the design specification created by the user. The tool is described in the following section.

II. THE PROPOSED VISUALISER AND EDITOR TOOL

The proposed tool allows the user to specify shapes (atomic or compound), shape rules (that transform a shape into another shape), and processes. It also allows the user to execute those processes in a given design to produce new designs and save them for later reuse.

A. User Interface

As depicted in Fig. 2, the interface includes a window through which the user is able to design shapes and store them in a library of shapes. These shapes can then be used by the user to create shape grammar rules, as exemplified in Fig. 3.

![Fig. 2. Shapes design interface. Left: interface with the library of shapes already designed by the user. Right: example of a shape being designed.](image)

The interface also provides the user with a visual language to handle processes. There is a visual representation for each possible type of process (see Fig. 4), namely: (1) shape rules; (2) named processes; (3) sequences; (4) options; (5) verifications; and (6) empty process. Sequences are sets of processes executed in order. Options are sets of mutually-exclusive processes. Verifications check whether certain conditions apply and block process execution otherwise.
B. Implementation Details

Fig. 5 presents the two main software components that compose the proposed tool. One of the components is responsible for the visual interface through which users are able to produce design specifications, whereas the other is responsible for the inference engine capable of generating designs that comply with the specification produced by the user.

The visual interface was developed in the Java programming language, using the AWT interface. The design specifications defined by the user in terms of shapes, shape grammar rules, and processes are stored in XML files, which can then be used to open and edit the specification.

The inference engine responsible for the generation of designs that comply with the design specification was built on top of a Prolog implementation of the Alternative Shaper. Both visual interface and inference engine interact via the SWI-Prolog Java toolkit. More specifically, the visual interface component asks Prolog to load the facts and predicates required for Alternative Shaper and then asserts all facts and predicates required to represent the design specification created by the user.

Alternative Shaper is responsible for executing a given process applied to a given design. The process applies repeatedly shape rules to the intermediate designs obtained so far according to the order established sequentially in the process. The shape grammar rules and options offer the possibility to generate a shape composition among different alternatives. Each time a verification fails or a shape grammar rule fails to apply, the system backwards trying to build a different design. The designs may thus be created automatically with no intervention of the user.

III. CONCLUSIONS

This paper proposed a novel tool to visualize and edit shape grammars supplemented with processes in the context of the Alternative Shaper model. Compared with older shape grammar editors, the presented tool supports, additionally, a visual language adequate for representing processes for design generation. Some informal tests have already been carried out to guide the development of the tool and to assess its usability. However, it is still necessary to execute a set of formal tests with target users (e.g., architects and designers) to extensively evaluate the system’s usability. These tests are expected to show the benefits of using a visual language to produce design specifications over direct text-based coding.

REFERENCES