General Shape Grammar Interpreter for Intelligent Designs Generations

T. Trescak
Artificial Intelligence Research Institute
Spanish Council for Scientific Research
Barcelona, Spain
ttrescak@iiia.csic.es

I. Rodriguez
Applied Mathematics Department
University of Barcelona
Barcelona, Spain
inma@maia.ub.es

M. Esteva
Artificial Intelligence Research Institute
Spanish Council for Scientific Research
Barcelona, Spain
marc@iiia.csic.es

***Abstract*—Shape grammars play an important role in a new generation of tools for the analysis and design of products. In this work we present a general tool named Shape Grammar Interpreter (SGI) for the automatic generation of designs. The developed shape grammar framework allows designers to obtain automatically generated designs and to participate in the design process. In that way the generated design complies with both the desired functionality and an attractive aspect. Great effort has been devoted on having a comfortable way of defining shapes and later using them in shape grammar rules and designs' generation process. We have also implemented and incorporated in the tool an optimized subshape detection algorithm. Hence, subshapes of the existing shapes can be detected in the generation process obtaining more appealing designs.**

***Keywords*-shape grammars; electronic institution; virtual world; 3D; interpreter;**

I. Introduction

In recent years, ever-increasing advances in both 3D computer visualization and artificial intelligent technologies have motivated the evolution of traditional computer aided design tools to more generic, useful and sophisticated ones. Shape grammars play an important role in a new generation of tools for the analysis and design of products.

Shape grammars in computation are a specific class of production systems that generate geometric shapes or designs [18][10]. Instead of using sequenced instructions as basic unit of computation, production systems use unordered and data-sensitive rules called production rules. Shape grammars are capable to represent knowledge about both the functionality and the form/shape of a product. Additionally, shape grammars generate forms not defined previously, i.e emergent shapes.

Production systems can be used for synthesis and classification tasks. Our interest is centered on synthesis problems which involve the generation of geometrical designs. Shape grammars can be also used for analysis as the resulting design can be used for purposes of explanation and what-if analysis. In computer aided design, shape grammars accelerate and simplify the development of prototypes.

In this research, we present a powerful tool named SGI (Shape Grammar Interpreter) that allows users to create, modify shape grammars and generate designs in an interactive way. Generated designs are two dimensional but the framework will incorporate a transformation mechanism in charge of creating the 3D counterpart and export it to several 3D formats. The main contribution of the framework is the generic nature of the interpreter but other features, such as usability, modularity and performance, are particularly important from the user's point of view. Potential applications of our research can be found in the educational field (i.e architecture and arts) and in the automatic generation of designs. We also present the use of this tool for the design of the floor plan of Virtual Institutions, which are virtual worlds where participants activities are regulated by the defined institutional rules [1][9][4].

This paper is organized as follows. Section II presents shape grammars and their history. Section III reviews the related work on shape grammars and presents some state-of-the-art solutions. Section IV gives an overview of the general shape grammar interpreter including a detailed description of the graphical user interface. Section V presents conclusions and future work.

II. Shape Grammars

Shape grammar is a method of generating designs by using primitive shapes and the rules of interaction between them. Shape Grammar rules are composed of left-side shapes and right-side shapes. Right-side shapes are either transformed left-side shapes or new additional shapes. Grammar is executed as follows:

1) Recognition of a particular shape from the left-side
2) Replacement of the recognized shape to the right-side shape of the rule.

As an example from we can take simple *addition* rule, where we add a square to the top-right part of original rectangle. In section IV-C we present this example. Later this rule can be applied to this newly added rectangle and to all its symmetries. To control how the rule is applied to the left-side of the shape *markers* can be applied.

Shape Grammars were introduced by George Stiny and James Gips in the early 1970s as a way of describing and creating paintings and sculptures [18]. This first attempt led to wide-spread use of grammars into multiple artistic, scientific and industrial fields. It has been used to analyze and recreate works of various artists like Piet Mondrian, Georges Vantongerloo and Fritz Glarner and for many of these artworks with great accuracy. In architecture, Frank Lloyd Wright's prairie houses [11], Palladio's villas [20] or Mughul gardens [21] were analyzed by the shape grammars and new studies were presented according to original design.

The inventors of shape grammars showed that by using simple geometrical shapes we can analyze and recreate visual styles of complex original designs. We can regard it as a way of an encapsulation of styles. It can be used for educational purposes to better understand structure and composition of original design and it also helps us to discover principles behind the design. It is possible to also generate new designs in the standard-defined style [14].

III. Related work

Shape grammars had their first applications in architecture [19] but soon they had applications in engineering such as in coffee-makers [2] and process plans [5] researches. Shape grammars started to spread into lots of the fields working with visual representations. Jose Duarte used them to generate Siza's Malagueira houses [7] and created an online application that rendered houses depending on user preferences. But shape grammars also entered non traditional fields when they were used to derive cellular automata rule patterns [22]. Sometimes those grammars were managed manually (i.e with pencil and paper) or the computerized approaches were done ad-hoc for a specific grammar. The framework we present in this paper has been designed to be a general tool to create and work with any 2D shape grammar.

An early generic shape grammar system was implemented in prolog [6]. Nevertheless, first attempts produced results not visually attractive. Nowadays, shape grammars continue being an interesting research topic and trends are oriented to provide the designer both an attractive visual interface and realistic results. Cityengine system was inspired by Lindenmayer systems [17] where rules are codified using symbols. This system introduced the CGA shape concept [16] [15]. Rather than building on string replacement as L-systems do, cityengine rules replace shapes with shapes and we could say it is an hybrid symbol-shape grammar. In contrast, our approach is a pure shape grammar framework where the rules and the process of rules application is performed using directly geometrical shapes. A recent research, also using CGA shape, has presented a real-time visual editor which works on usability issues such as the possibility of doing interactively local modifications on buildings [13].

IV. General Shape Grammar Interpreter (SGI)

Up until now there have been numerous attempts to create a general shape grammar interpreter but most of existing tools are either very specific on its purpose, had only limited functionality, were programmed to one operational system or had any other limitations such as speed of generation or impossibility to detect sub-shapes.

We have focused on bringing complete and robust tool that would allow user comfortably specify any shapes and rules and also have complete control over grammar rendering process. Another important aspect of this tool is the object-oriented design that allows future programmers easily extend current functionality.

A. Framework description

SGI is programmed in JAVA and we plan to offer this software as open-source to provide possibilities of joint development in different shape grammar communities. Great effort has been devoted on having a comfortable way of defining shapes and later using them in shape grammar rules.

User creates new shapes by drawing them on canvas using a mouse. Rules are operated in similar manner, the user creates rules by specifying the spatial relation either parametrically or by mouse. All modifications of current grammar are persisted in XML file for future use.

There exist predefined shapes such as rectangle or triangle. It is also possible to use curves. Existing shapes are used to create rules of a shape grammar. Currently supported types of rules are:

- *addition*: adds new shape in spatial dependency to another shape;
- *substitution*: substitutes existing shape by the new shape; and
- *modification*: modifies existing shape to new proportions.

We contemplate undeterministic shape grammars, characterized by the possibility of applying several rules in one generation step. Several mechanisms are implemented to select a candidate shape and rule to proceed in generation process. They are separated in two groups:

1) *Tree-search* mechanism stores state of current generation process in a tree structure and uses traditional tree-search algorithms to find the next rule to apply (e.g. breadth first, depth first).
2) *Sub-shape* detection mechanism detects sub-shapes emerging from current generation process. This means, that after each step of design generation all sub-shapes that exist on the left side of any rule are detected so they can be used in the next step.

User has the possibility to either select next rule of current generation process or lets computer to decide by selecting different levels (e.g. low, medium, high) of randomization. This allows user to generate either all possible designs in the next step or select random design processed in n steps. Implementation of tree-search mechanism is trivial and it does not result into shape emergence. Because of this we will focus on sub-shape detection meachnism which brings much more possibilities into shape generation process.

B. Subshapes detection algorithm

Shape Grammar Interpreter implements modified version of Ramesh Krishnamurti algorithm for subshape detection [12]. This algorithm processes rules as the euclidean transformations of a left side of the rule to the right side of the rule.

Krishnamurtis algorithm is capable of solving most of the problems related with subshape detection but it also has some limitations. Due to missing detection of infinite subshape (such as finding line subshapes in line segment), it processes only shapes with at least three points. However, the biggest drawback of this algorithm is performance. To overcome this problem we have modified the algorithm by allowing detection of subshapes with at least one intersection. This provides real-time capability of rendering designs using subshape detection. Algorithm structure is presented next. As it works with specific terms of *maximal shape*, *maximal line* and *viable intersections*, we explain them next.

1) Maximal shape and maximal line: Maximal shape is a new shape created from original one using its maximal lines, that is the minimum set of lines maintaining the original form of the shape (function $CreateMaxLns$ in the algorithm). Figure 1 shows how the maximal shape is created by joining lines into maximal lines. In original shape a) lines a and b, or c and d are joined by algorithm to maximal line e and f to create maximal shape b). This allows the algorithm to work with minimal set of intersections and also to correctly detect if subshape is within boundaries of original shape (as explained in section IV-B3 below).

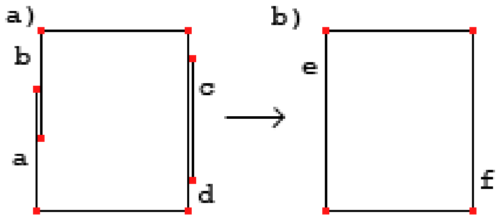


Figure 1. Maximal lines: a) original shape b) maximal shape

2) Viable intersections: Viable intersection is any internal or external intersection of the two segments of the shape (this process is done by function *CreateInts* in the algorithm). By outer intersection we mean intersection which is positioned on the line containing the segment, but outside of its boundaries.

In Figure 2a) we see all viable intersections of the shape. The internal intersections are shown as black boxes, external as white boxes and dotted line displays containing line of segments. Figure 2b) emphasizes the importance of finding external intersections as without them the subshape (represented in grey) would not be detected in input shape (black).

3) Algorithm: Below we present naive representation of the Krishnamurtis subshape detection algorithm.

Algorithm 1: Subshape detection algorithm

```
Input: inputShape, subShape
Output: Collection of subshapes
begin
    maxLines ← CreateMaxLns (inputShape)
    subMaxLines ← CreateMaxLns (subShape)
    inters ← CreateInts (maxLines)
    transfs ← FindTransfs (subshape, inters)
    forall transfs do
        if $\forall$subMaxLines $\subseteq$ maxLines then
            subshapes ← TransShape (subshape)
end
```

Let us explain step by step, over a simple example, the execution of the algorithm and what modifications we have performed. In this example we will be detecting $subShape$ b) in $inputShape$ b) displayed in Figure 3.

In the first step, maximal shapes (sets of maximal lines) are created from both of the shapes as presented in previous section. When this process is finished we find all the viable intersections in both shapes as shown in Figure 4.

The most time-consuming part of the algorithm is finding correct transformations (function *FindTransfs*) of the $subShape$ to the $inputShape$. In the original version of the algorithm, any three points are taken from $subShape$ to create the transformation to any three points in the $inputShape$. This transformation is used to check if remaining points of the $subShape$ are transformed to some points of the

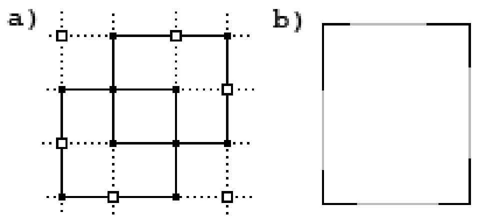


Figure 2. Intersections

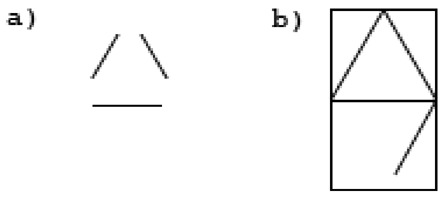


Figure 3. Algorithm input: a) $subShape$ b) $inputShape$

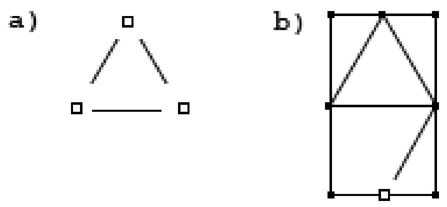


Figure 4. Intersections: a) $subShape$ b) $inputshape$

$inputShape$. This search space is exponential to the amount of intersections in $inputShape$. Our proposal reduces this search space by using *intersection triplets*, that is a structure containing intersection point, two guiding points, the angle and the ratio of lengths between related segments. Figure 5 shows an example of intersection triplet. The intersection point is represented as a white box and guiding points are represented as black boxes. The angle is created by containing lines of the two segments and ratio is obtained from the lengths of the segments.

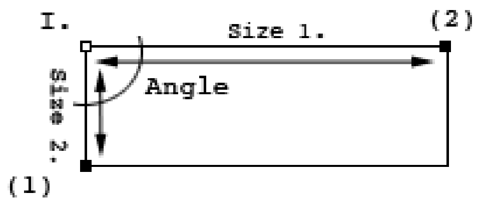


Figure 5. Intersection triplet

In a first step we find triplets in the $subShape$, order them by the angle and select the one with the smallest angle. Second we start expanding triplets in the $inputShape$. All triplets that do not have the same angle and the same ratio are thrown away, the good ones are stored. For each of the stored triplet we create affine transformation, and check if remaining points of the $subShape$ will fall onto points of the $inputShape$. We store all passing transformations and discard the rest.

In the last step, we check if transformed maximal lines of the $subShape$ fall within the boundaries of maximal lines of the $inputShape$ (function $TransShape$ in the algorithm). Figure 6 shows the passing condition a) in the left part of the shape and a failing condition b) in the right part. We see that we could find the transformation of the points of the $subShape$ to the lower part of the $inputshape$, but the test on the boundaries fails. The white boxes represent intersection points and the black boxes are guiding points.

We have significantly improved the performance of the subshape detection algorithm using *intersection triplets*. In the original version, we would have to test $k!\binom{n}{k}$ combinations, where k is the number of points used to create the transformation and n is the number of intersections of the

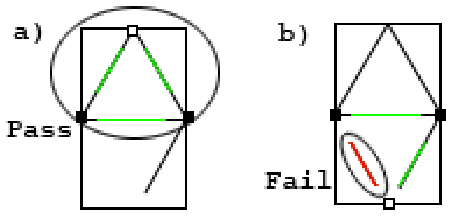
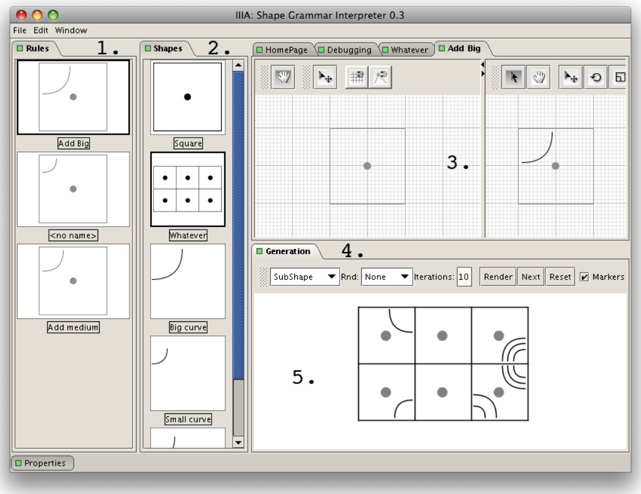


Figure 6. Boundary detection: a) Passing detection b) Failing detection, missing boundary



SubShape Rnd: None Iterations: 10 ✓ Markers

Figure 7. SGI User Interface

$inputShape$. Usually, the transformation is created using three points, so k is equal to 3. In the provided simple example, it would be 504 combinations to test. With the proposed modification we have to test only 4 combinations.

C. SGI User Interface

Figure 7 displays SGI graphical user interface. It was designed to provide comfortable user experience. Tabbed framework lets the user to customize the interface. Focus was stressed on comfortable definition of shapes and rules by using mouse. Parts and rules are defined on separate canvases giving the user complete control and overview of ongoing work. This figure also points to most important parts of the SGI. Part 1) is used to show all current rules. Part 2) shows all current shapes. The list is created by the thumbnails of given objects. Part 3) is the edit area. Here it is where most of the actions are performed, including all shape and rule modifications. Part 4) and 5) are used for grammar rendering. Part 4) is used to define what *protocol* and *randomization* we want to use for rendering as well as how many iterations we want to perform in one rendering. This allows us to render grammar step by step and see how the shape emerges.

Let us present some results of generations using SGI. Figure 8a) represents a very simple spatial rule, where we add another square to original square, creating new square in their intersection. Figure 8b) shows the result of the generation using simple tree search protocol. This simple generation respects original shape and its orientation always creating the very simple design.

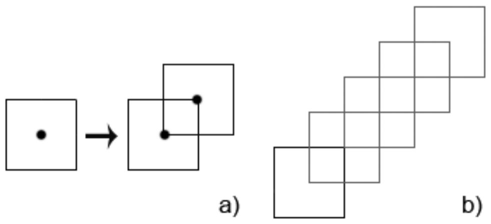


Figure 8. a) Definition of rule b) Generated design using tree search protocol

Let us see what will happen when we allow the generator to use subshape detection. Figure 9a) shows the result of generation using simple rule from figure 8a) with active subshape detection and active markers. Figure 9b) depicts the result of generation with active subshape detection but without markers. As we see we have generated much more shapes creating very interesting designs from this one simple rule. Presented technology of markers limits subshape detection for emerging shapes. When placing marker on some shape, generator not only has to find subshape in current shape but also has to find this marker on the same position in the detecting shape. In Figure 9a), when smaller unmarked rectangles, such as those created by intersection of rectangles are ignored, the detection of possible shapes is limited to the originally marked ones.

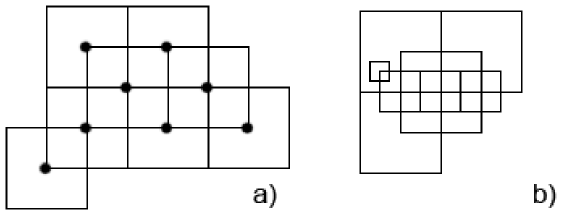


Figure 9. Generated design using rule from figure 8 with activated subshape detection. a) With markers b) Without markers

V. Conclusion and future works

Shape grammars are very promising and usable mechanism for analysis of existing designs and smart generation of new designs. Up until now there did not exist a tool that would comfortably allow to process it. In this research we have presented a generic shape grammars tool named SGI. The framework allows to interactively create shapes and rules and several mechanisms have been implemented to select a candidate shape and rule to proceed in generation process. Contributions of the research are 1) the generic nature of the interpreter, 2) the inclusion of an optimal algorithm for subshapes detection and 3) the user-friendly design.

Although SGI can be used in several fields of applications, we plan to use it in the development of virtual institutions [1][4]. Virtual institutions are interaction environments where participants can be human and software agents. It is a normative environment where software and human agents can participate and collaborate in a joint 3D virtual world [3]. Communication between participants is done by avatars (user's or generally agents computer representation of himself/herself or alter ego). Institutional rules structure valid interactions within the institution providing controllable and normative environment. For the *specification of the institutional rules*, we use electronic institutions (EI), a well-known MAS methodology. The institution specification [8] defines among other issues the activities (interactions protocols) participants can engage in and their connections, that is the role flow policy among them. It also establishes the maximum number of participants for each activity.

Construction of such Virtual World (VW) will be done in two phases:

1) Specification of institutional rules using Islander Tool
2) Generation of the Virtual World

We plan to use the Shape Grammar Interpreter in the second phase to automatically generate the floor plan of the building taking into account the institution specification. For this purpose, we will extend the tool to load existing specification of an EI and extract needed metadata such as: how many rooms are to be generated, how they are linked or what is the needed size of the room. These parameters will be later used as input for a defined *floor plan* grammar to create initial 2D layout of the virtual environment. Another shape grammar will solve automatic population of the functional and non-functional objects into the VW (e.g. interaction objects such as reception desk, furniture ...). We are also working on the mechanism of transforming 2D representation into a 3D one and then exporting it into an open standard format for interactive 3D applications like COLLADA or X3D.

Acknowledgment

Partially funded by projects IEA (TIN2006-15662-C02-01), AT (CONSOLIDER CSD2007-0022) and by the Generalitat de Catalunya under the grant 2005-SGR-00093. M.Esteva enjoys a Ramon y Cajal contract from the Spanish Government. We want to thank to Joan Carles Naranjo, Yestyn Jowers and Miquel Prats for their extensive support on this research.

References

[1] A.Bogdanovych, M.Esteva, S.Simoff, C.Sierra, and H.Berger. A methodology for developing multiagent systems as 3d electronic institutions. In *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in CS*, pages 103–117. Springer, 2008.

[2] M. Agarwal and J. Cagan. A blend of different tastes: The language of coffee makers. *Environment and Planning B: Planning and Design*, 25(2):205–226, 1998.

[3] A. Bogdanovych, M. Esteva, S. Simoff, C. Sierra, and H. Berger. A methodology for 3d electronic institutions. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, New York, NY, USA, 2007. ACM.

[4] Anton Bogdanovych. *Virtual Institutions*. PhD thesis, University of Technology, Sydney, Australia, 2007.

[5] K. N. Browna, C. A. McMahon, and J. H. Sims Williams. Describing process plans as the formal semantics of a language of shape. *Artificial Intelligence in Engineering*, 10(2):153–169, 1996.

[6] Chase S C. Shapes and shape grammars: from mathematical model to computer implementation. *Environment and Planning B: Planning and Design*, 16:215–242, 1989.

[7] J. P. Duarte. *Customizing mass housing : A discursive grammar for Siza's Malagueira houses*. PhD thesis, Cambridge (MA): Massachusetts Institute of Technology, 2001.

[8] Marc Esteva, David de la Cruz, and Carles Sierra. Islander: en electronic institutions editor. In W. Lewis Johnson Cristiano Castelfranchi, editor, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1045–1052, Bologna, Italy, July 2002. ACM PRESS.

[9] I.Rodriguez, A.Puig, M.Esteva, C.Sierra, A.Bogdanovych, and S.Simoff. Intelligent objects to facilitate human participation in virtual institutions. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 196–199, 2008.

[10] T. W. Knight. Shape grammars: six types. *Environment and Planning B: Planning and Design*, 26(1):15–31, 1999.

[11] H Koning and J Eizenberg. The language of the prairie: Frank lloyd wright's prairie houses. *Environment and Planning B*, 8(3):295–323, 1981.

[12] R. Krishnamurti. The construction of shapes. *Environment and Planning B: Planning and Design*, 8:5–40, 1981.

[13] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. *ACM Trans. Graph.*, 27(3):1–10, 2008.

[14] Junsik Moon. *Shape grammar for Mies van der Rohe's highrise apartment*. PhD thesis, S.M. Massachusetts Institute of Technology, 2007.

[15] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, 2006.

[16] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA, 2001. ACM Press.

[17] Radoslaw Karwowski Brendan Lane Przemyslaw Prusinkiewicz, Lars Mndermann. The use of positional information in the modeling of plants. In *Proceedings of ACM SIGGRAPH 2001*, page 289300, 2001.

[18] G. Stiny and J. Gips. *Shape grammars and the generative specification of painting and sculpture.* IFIP Congress 1971. North Holland Publishing Co., 1971.

[19] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. In C. V. Friedman, editor, *Information Processing '71*, pages 1460–1465, Amsterdam, 1972.

[20] G Stiny and W J Mitchell. The palladian grammar. *Environment and Planning B*, 5(1):5–18, 1978.

[21] G Stiny and W J Mitchell. The grammar of paradise: on the generation of mughul gardens. *Environment and Planning B*, 7(2):209–226, 1980.

[22] E. Crawley T. Speller, D. Whitney. Using shape grammar to derive cellular automata rule patterns. *Complex Systems*, 17:79102, 2007.