Procedural Architectural Facade Modeling

Jonathan Zweig*
Computer Science Department
Brown University

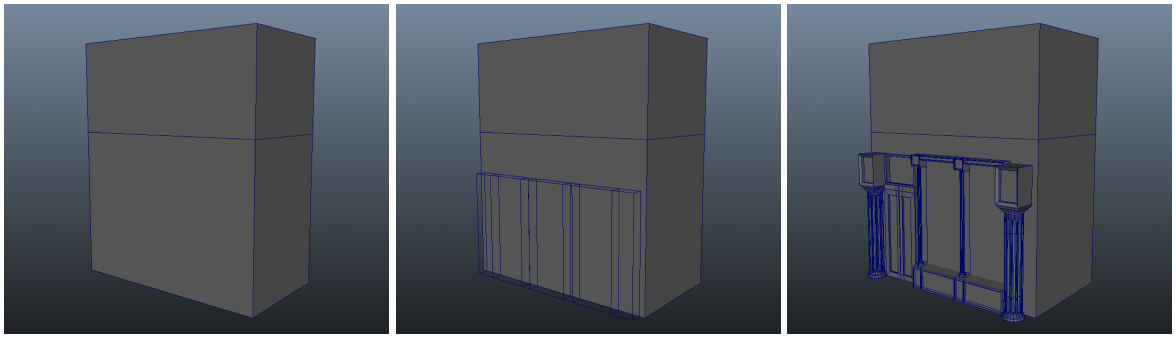

Figure 1: *This sequence of images, from left to right, shows the creation of an architectural facade using the plugin introduced by this paper.*

Abstract

This project is an implementation of the CGA shape grammar as a plugin for Autodesk Maya® to create a interactive environment for generating architectural facades. The CGA (Computer Generated Architecture) shape grammar, is a context-sensitive grammar created by Pascal Müller and Peter Wonka to procedurally generate buildings [Müller et al. 2006]. By implementing a subset of the shape grammar directly into an industry standard modeling program, this project demonstrates a way to streamline the workflow pipeline for technical artists at film and game studios when modeling multiple buildings that adhere to the same style.

Keywords: modeling, architecture, shape grammars, Autodesk Maya

Links: PDF

*e-mail: jmzweig@cs.brown.edu

1 Introduction

In the film and game industries, modelers are under tight deadlines to create detailed sets. Although these sets often have detailed concept art that the modeler can follow when creating them, if the set changes at all, whole buildings must be remodeled to fit them in the new dimensions.

Recently, Pixar encountered this problem when making the city of London for *Cars 2*. Because they had many buildings, and knew that the layout of the buildings could change dramatically depending on the needs of the creative directors, Pixar built an in-house procedural solution, wrapped around the commercial program Esri CityEngine™[Frederickson and Northrup 2011][Wonka et al. 2011]. Although this solution allowed Pixar to procedurally generate London, it was tailored for their specific use case.

The primary obstacle to the procedural modeling of facades is that elements in the facade require spatial and contextual awareness of the facade. The solution to this problem is complex. It requires awareness of the relationships between architectural elements and the dimensions of the building. For example, without awareness of location, elements meant for the ground floor could be incorrectly placed on upper floors. The solution must also maintain the relationship between multiple architectural elements. For example, a set of win-

dows should not necessarily be placed right next to another set of windows. To solve these issues, we used previous research in procedural architectural modeling and shape grammars.

1.1 Background

1.1.1 Shape Grammars

A shape grammar is a natural way of representing an architectural style so it may be reproduced on buildings of different sizes. Shape grammars were first introduced by George Stiny and James Gips in their paper "Shape Grammars and the Generative Specification of Painting and Sculpture" [Stiny and Gips 1971]. Shape grammars are differentiated from string grammars in that the a shape grammar is defined by an alphabet of shapes, a starting shape, and a set of rules that describe spatial relations between the shapes in the alphabet. For more on grammars, see [Chomsky 1957] and [Sipser 1996].

The only existing commercial program implementing shape grammars is Esri's CityEngine. Esri CityEngine has pioneered the use of shape grammars in the industry. The main focus of the program, however, is on the generation of entire 3D cities and not on the generation of detailed facades. Although there is support for facade generation, CityEngine does not allow in-depth modification of the facade, and thus it is not a replacement for a modeling environment, such as Autodesk Maya® [Parish and Müller 2001].

1.1.2 Maya

Nodes User projects in Maya are saved in a scene file. A scene file is organized as a node system, where each node stores its own set of attribute data. A node often depends on the attributes of other nodes to determine its own attribute values. Connecting attributes between nodes enables sharing and modification of the initial attribute data. In this way, a project in maya is a network of interconnecting nodes. Maya visualizes this network in its node editor interface (See Figure 2). Shape nodes hold geometry information, such as where vertices in a mesh are located spatially, in local space. By connecting a transform node to a shape node, a user can modify all of the vertices' location information in world space.

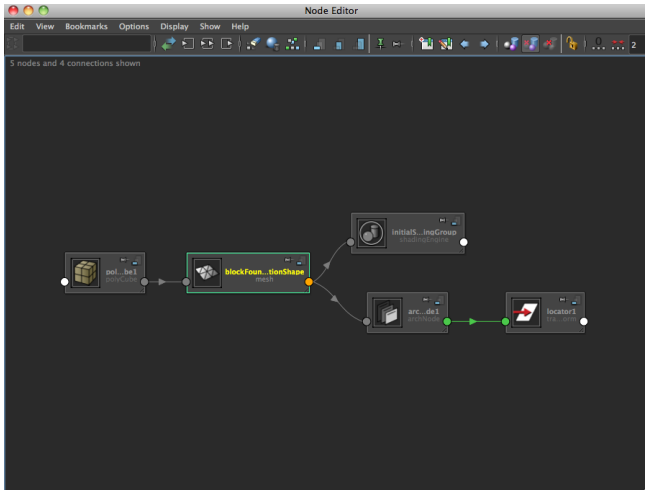

Figure 2: *A screenshot from Maya 2013's node editor, visualizing a project in maya as a network of interconnecting nodes.*

Scene Graph In Maya, objects in a scene are organized in a hierarchy known as a scene graph, (See Figure 3). A scene graph is a directed acyclic graph (DAG), where each non-leaf node contains geometric transformation information, such as translation, scale, or rotation. Children of these nodes are placed in the scene using their parent node's transformations in addition to their own transformations. Geometric shapes in Maya are implemented as leaf nodes in the scene graph, just as they are terminal symbols in the shape grammar [Autodesk 2013].

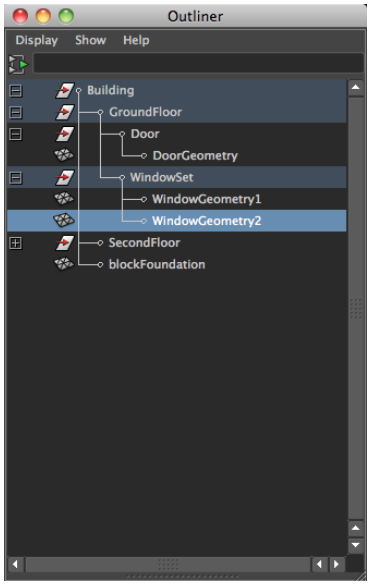

Figure 3: *A screenshot from Maya 2013's outliner, illustrating Maya's scene graph*

1.2 Previous Research

Peter Wonka, in his paper "Instant Architecture," proposes an automated approach to shape grammars. Wonka treats shapes as symbolic objects [Wonka et al. 2003]. He extends Stiny's work by adding the concept of attributes, such as the shape's width or height, to each shape in the alphabet. These attributes can be passed as a *context* to rules during evaluation. A rule's context is propagated to its children, distributed as evenly as possible to fit the constraints defined by the children's attributes. For non-terminal shapes (shapes that must still be evaluated), attributes can either be absolute, or relative to those in the evaluating rule's context. For terminal shapes, attributes must be absolute, however a terminal shape's size can be smaller than the context it is given during evaluation. In this way, Wonka's grammar dictates the atomic, pre-made piece that should be inserted at any given position on the building. By encoding the spacial information of the shapes into the grammar, Wonka generalizes Stiny's shape grammars, making it more suitable for a computer implementation.

Pascal Müller and Peter Wonka describe an implementation of Wonka's previous work in their "Procedural Modeling of Buildings," as the CGA (Computer Generated Architecture) shape grammar [Müller et al. 2006]. They introduce the concepts of *split rules* and *repeat rules*. As defined by Müller and Wonka, *split rules* divide the rule's context up between the exact set of products of the rule. *Repeat rules* are an extension of *split rules*, which allow a set of products to be repeated as many times as fits in the context of the rule during evaluation. We use these concepts in the implementation of a subset of the CGA shape grammar in Maya.

1.3 Overview

The rest of this paper is structured as follows: We start by outlining the high level overview of the program's design and architecture in section 2. In section 3, we explain our grammar implementation and how we integrate the shape grammar with Maya. In sections 4 and 5 we evaluate the workflow and functionality of our program and discuss the features and limitations of our implementation.

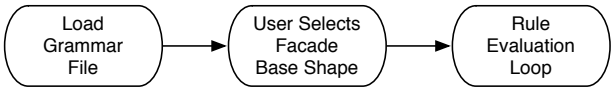

Figure 4: *Overview of the user flow. See section 2.2 for a detailed description of each step.*

2 Software Design and Architecture

When integrating shape grammars into Maya, we took advantage of the parallel between the hierarchical manner in which Maya organizes its scenes, and the parse tree created when rules are evaluated.

2.1 Design Requirements

The main architecture of the program takes into consideration the following. The program must:

- Have the ability to import pre-made shape grammars, since there is currently no way to create shape grammars inside Maya.
- Provide users with a controlled method for automating repetitive details. The implementation should take a block model of the building as input, and construct the facade on a specified portion of the model.
- Take advantage of Maya's interactive modeling environment. Maya users are accustomed to interactivity. The program should allow the user to use native Maya methods to modify the base shape.

2.2 Main User Flow

The main user flow (see Figure 4) of the plugin is as follows:

1. The user imports a grammar into the plugin from a text file. (Figure 5)
2. The user selects the base shape, start vertex, end vertex, and a vertex to track the facade height. The plugin uses this information to generate an initial context for the shape grammar. The initial position is set to that of the start vertex. The width is calculated as difference between the start and end vertices, and the height is calculated as the difference between the start and height vertices. (Figure 6)

3. When the user moves one of the vertices on the base shape, Maya signals the plugin to evaluate the grammar rules to check for any changes. The plugin then generates the new geometry for Maya to render. (Figure 7)

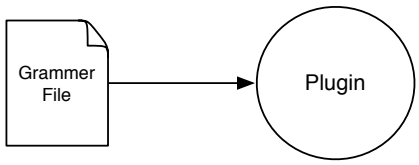

Figure 5: *Step 1 of the user flow*

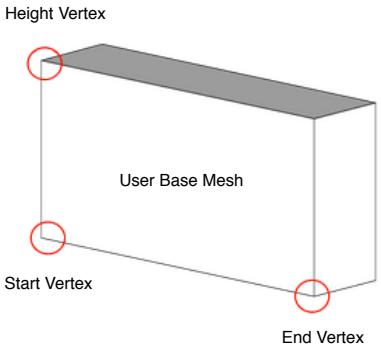

Figure 6: *Step 2 of the user flow*

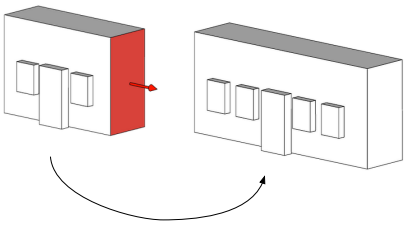

Figure 7: *Step 3 of the user flow*

3 Implementation

3.1 Grammar Implementation

This project's grammar is a simplified version of the CGA shape grammar. We keep the same principle of using a parametric grammar for determining spatial layout, but we simplified it for the scope of the project. The project's grammar is split into three parts: conditions, products, and rules.

3.1.1 Conditions

The conditions of the grammar consist of three parts: an attribute of the context, an equality or inequality operator, and a constant value against which to compare the attribute. (See Figure 8).

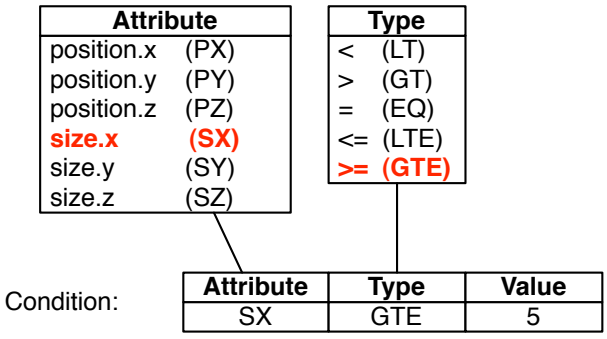

Figure 8: *Conditions are limited to position and size components.*

3.1.2 Products

Products in the grammar are either standard products, which represent non-terminal symbols in the grammar; or geometry products, which represent terminal symbols, and specific, renderable 3d model assets. (See Figure 9).

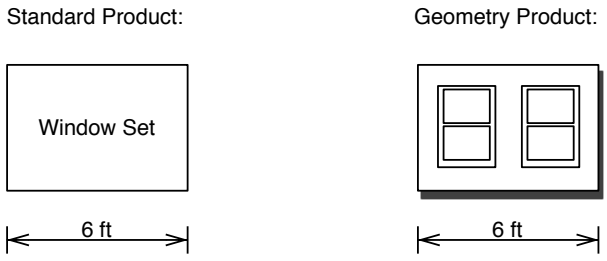

Figure 9: *Products are either non-terminal* ***standard*** *products or terminal* ***geometry*** *products.*

3.1.3 Rules

Rules consist of a list of conditions and a list of products. We implemented the two types of rules from the CGA shape grammar described previously: *split rules* and *repeat rules*. (See Figure 10).

In our implementation of *split rules*, we chose to fo-

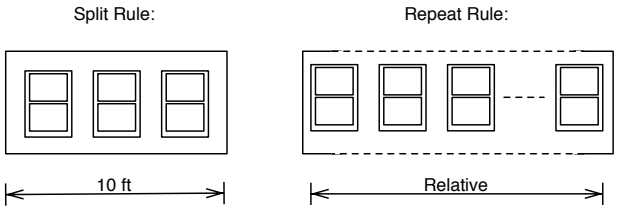

Figure 10: *Repeat rules repeat products as many times as can fit the context.*

cus only on horizontal splits. Instead of defining the size of the products of the splits in the rule, we let the products determine their own size, given a context. After absolutely-sized products are added, relatively-sized products are given equal portions of the remaining context.

The grammar is stored as a JSON string. JSON, which stands for JavaScript Object Notation, is a human-readable and writable text format that is efficient for computers to parse and create [Crockford 2006]. By storing our grammar as JSON, the grammar is integrated directly into Maya's node structure and is saved together with the Maya scene file.

3.2 Maya Integration

3.2.1 Custom Nodes

Two custom nodes are created to integrate with Maya: ArchManagerNode and ArchNode. The ArchManagerNode is responsible for the persistent storage of the grammar inside of Maya. When a user imports a grammar, the JSON file is read into a string buffer and is stored as an attribute in the Maya node. The ArchNode is responsible for updating the scene graph in the rule evaluation loop. This includes removing all nodes from the previous rule evaluation and creating the nodes for the new rule evaluation.

3.2.2 User Interface

The user interface uses a combination of MEL, (Maya's embedded scripting language), and Python calls into Maya's API. Bounding boxes (ghosts) show where architectural pieces will be placed. This is a visualization of the rule evaluation process for the user. When the base shape is modified, these bounding boxes update to reflect the changes in rule evaluation. The user

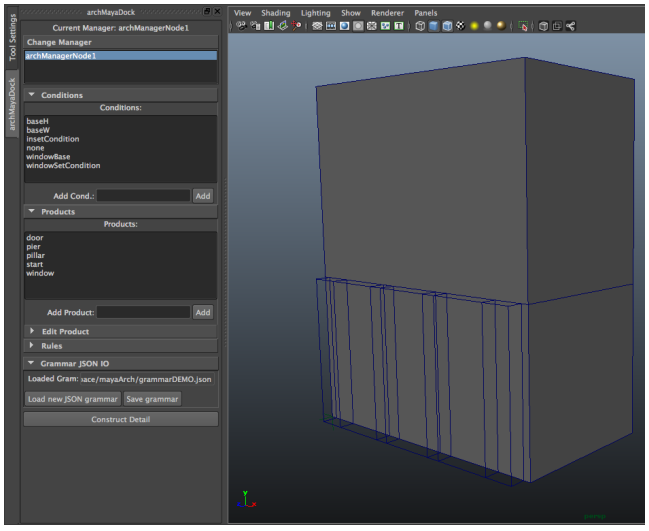

Figure 11: *A screen shot of the GUI of our plugin. Loaded conditions, products, and rules of the grammar are displayed.*

can request that detailed assets be imported and replace the bounding boxes at any time; however, the bounding boxes are used to ensure the interface stays responsive during modification.

4 Results

4.1 Implementation

As the main contribution of this paper is the implementation of shape grammars in Maya, an example of the use of our plugin is illustrated in Figure 12. (1) A grammar representing a facade is created in JSON imported using the plugin's native Maya UI. (2) A shape is selected and confirmed as the base shape to the plugin. (3) The start vertex, followed by the end and height vertices are specified. (4) The plugin evaluates the grammar and displays bounding boxes of the resulting facade. (5) When the base shape is modified, the resulting grammar evaluation is rendered. (6) The final detailed assets are imported by clicking the "construct detail" GUI button.

4.2 Performance

Response time is a crucial element of 3D modeling programs. As an interactive implementation of shape grammars, our solution is expected to perform with the same level of responsiveness as native modeling tools in Maya. For instance, when the user modifies the

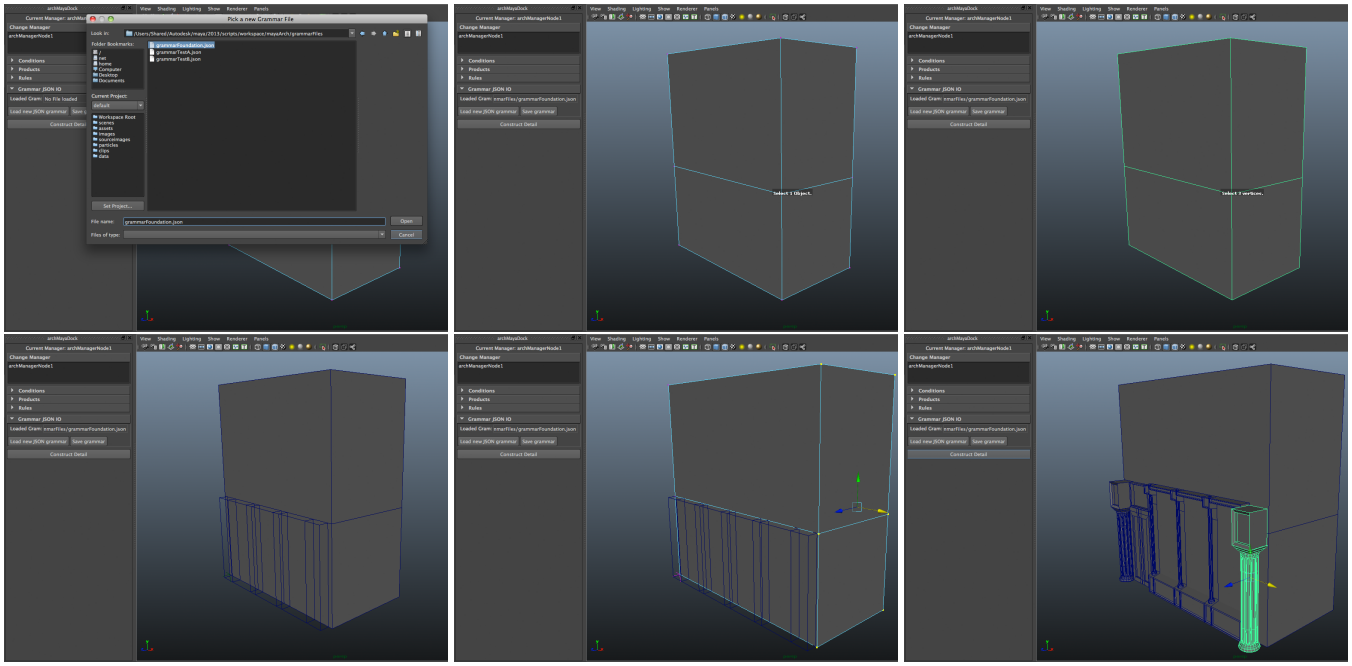

Figure 12: *Example workflow using our Maya plugin. A description of each screenshot is provided in section 4. The order of the images is from top left to bottom right.*

base shape in Maya, such as by dragging the vertices associated with the facade, the implementation of the shape grammar must be able to evaluate the rules without causing the user to wait. The implementation must delete and create new Maya nodes for each rule evaluation. However, since Maya has its own deferred memory management, the plugin stays responsive, and only starts to lag on a 2009 Macbook Pro equipped with a 2.8Ghz Intel Core 2 Duo processor when hundreds of nodes must be handled.

5 Discussion

5.1 Functionality

Using this plugin a technical artist can construct complex facades using a set of rules inside Maya. The user can import these rules using a pre-made JSON grammar. These grammars can be partially modified inside Maya using the GUI to add conditions. The user can export the modified grammar and edit the grammar even further outside of Maya. The plugin also handles real-time, interactive modification of the base shape.

5.2 Limitations

The plugin currently only handles rotations and not shearing of imported architectural models. Therefore, it will not detect if the resulting facade would continue off the edge of a slanted building. However, the user can still perform the shearing manually after the plugin loads the models into maya.

Furthermore, the plugin only functions along flat surfaces. However, curved surfaces along the facade, such as cylinders, can be accommodated by creating multiple facades using each flat section of the curve. This could be automated with a MEL script.

5.3 Challenges

Interactively evaluating rules in the grammar using Maya's API was difficult due to the way Maya handles and updates data. The conventional method for storing custom data in Maya is by creating a custom attribute in a Maya node. All information that is not stored as a Maya attribute is lost when the application closes. These attributes are limited, however, to primitive types, such as integers, floats, and strings. We overcame this issue by storing the grammar as a JSON string.

5.4 Future Work

Future work includes bringing grammar generation into Maya. Now that a framework for using shape rules in Maya has been implemented, it can be used to implement grammar generation, as well. Research by Markus Lipp, Peter Wonka, and Michael Wimmer outlines how to implement real-time interactive editing of shape grammars [Lipp et al. 2008], which could be implemented within Maya. Further research into Maya UI paradigms would need to be done, but having the grammar stored in Maya would make implementation of a GUI for creating rules easier.

Furthermore, the plugin can still be optimized for greater efficiency. Development on this project was done primarily in Python using Maya's Python wrapper on Maya's C++ library. Hence, the code can be easily converted into C++ for finer control of memory management during rule evaluation by calling the same functions in the API without the wrapper.

6 Conclusion

In this paper we have presented a plugin that combines the generative power of shape grammars with the robust modeling environment of Autodesk Maya. We have taken a complex, academic concept, and made it so that it can be applied in the production environment by technical artists. Writing rules for shape grammars takes some time; however, it is still easier and less time consuming than remodeling the entire facade. By implementing shape grammars in Maya, we have made it easier for modelers to make use of years of research on shape grammars.

Acknowledgements

I would like to thank Barbara Meier and John Hughes, my primary and secondary thesis advisors, respectively. I would also like to thank Tom Doeppner, the Director of Undergraduate Studies at the Brown University Computer Science department, and Michael Frederickson, a Technical Director at Pixar Animation Studios.

References

AUTODESK, 2013. Querying the scene graph. http://docs.autodesk.com/MAYAUL/2013/ENU/, January.

CHOMSKY, N. 1957. *Syntactic Structures*. Mouton & Co., The Hague. Reprinted 1985 by Springer, Berlin and New York.

CROCKFORD, D. 2006. The application/json media type for javascript object notation (json).

FREDERICKSON, M., AND NORTHRUP, J. 2011. Nuclear monkeys and talking cars. University talk, Brown University, September.

LIPP, M., WONKA, P., AND WIMMER, M. 2008. Interactive visual editing of grammars for procedural architecture. *ACM Trans. Graph. 27*, 3 (Aug.), 102:1–102:10.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (July), 614–623.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 301–308.

SIPSER, M. 1996. *Introduction to the Theory of Computation*. PWS Pub Co, Boston, MA.

STINY, G., AND GIPS, J. 1971. Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress (2)*, 1460–1465.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. Graph. 22*, 3 (July), 669–677.

WONKA, P., ALIAGA, D., MÜLLER, P., AND VANEGAS, C. 2011. Modeling 3d urban spaces using procedural and simulation-based techniques. In *ACM SIGGRAPH 2011 Courses*, ACM, New York, NY, USA, SIGGRAPH '11, 9:1–9:261.