Procedural Modeling in Theory and Practice

T. Ullrich, C. Schinko, D. W. Fellner
Institut für ComputerGraphik und WissensVisualisierung, TU Graz, Austria
Fraunhofer Austria Research, Visual Computing Devision, Graz, Austria
Fraunhofer Institute for Computer Research and Technical University of Darmstadt, Germany

ABSTRACT

Procedural modeling is a technique to describe 3D objects by a constructive, generative description. In order to tap the full potential of this technique the content creator needs to be familiar with two worlds – procedural modeling techniques and computer graphics on the one hand as well as domain-specific expertise and specialized knowledge on the other hand.
This article presents a JavaScript-based approach to combine both worlds. It describes a modeling tool for generative modeling whose target audience consists of beginners and intermediate learners of procedural modeling techniques.
Our approach will be beneficial in various contexts. JavaScript is a wide-spread, easy-to-use language. With our tool procedural models can be translated from JavaScript to various generative modeling and rendering systems.

Keywords: Computational Geometry and Object Modeling, Computer Graphics Methodology and Techniques, Modeling Languages

1 INTRODUCTION

Within the last few years generative modeling techniques have gained attention. In order to accelerate the modeling process, many researchers enforced research on procedural modeling. All models with well-organized structures and repetitive forms benefit from procedural model descriptions. In these cases generative modeling is superior to conventional approaches.

Its strength lies in a compact description [1], which does not depend on the counter of primitives but on the model's complexity itself. Especially large scale models and scenes – such as plants, cities, and landscapes – can be created efficiently. In this way generative descriptions make complex models manageable as they allow to identify a shape's high-level parameters [7].

A characteristic of generative modeling is its explicit analogy of 3D modeling and programming. This analogy has two negative aspects. First of all, the need to use a programming language is a significant inhibition threshold especially for architects, designers, etc. who are seldom experts in computer science and programming. Secondly, a programming language introduces a new dimension of complexity and further dependencies.

Furthermore, it has never been easy to convert 3D models between various file formats and with generative modeling techniques the the situation will probably become worse. If a 3D model does not only contain static geometry but algorithmic descriptions, the file format also depends on the languages and the interpreter that is able to execute the script.

In this paper we investigate generative modeling approaches concerning these aspects and present a new possibility to create procedural models in a beginner-friendly way. Additionally, we address the file format problem and present a solution to reduce the dependencies to scripting and rendering engines.

WSCG'2010, February 1 – February 4, 2010
Plzen, Czech Republic.

2 PROCEDURAL MODELING

In today's procedural modeling systems, grammars are often used as a set of rules to achieve a description. Early systems based on grammars were Lindenmayer systems [17], or L-systems for short. They were successfully applied to model plants. Given a set of string rewriting rules, complex strings are created by applying these rules to simpler strings. Starting with an initial string the predefined set of rules form a new, possibly larger string. The L-systems approach reflects a biological motivation. In order to use L-systems to model geometry an interpretation of the generated strings is necessary. The modeling power of these early geometric interpretations of L-systems was limited to creating fractals and plant-like branching structures. This lead to the introduction of parametric L-systems. The idea is to associate numerical parameters with L-system symbols to address continuous phenomena which were not covered satisfactorily by L-systems alone.

CGA Shape

Later on, L-systems and shape grammars were successfully used in procedural modeling of cities [16]. Parish and Müller presented a system that, given a number of image maps as input, generates a street map including geometry for buildings. For that purpose L-systems have been extended to allow the definition of global objectives as well as local constraints. However, the use of procedurally generated textures to represent facades of

buildings limits the level of detail in the results. In later work, Müller et al. describe a system [14] to create detailed facades based on the split grammar called CGA shape. A framework called the *CityEngine* provides a modeling environment for CGA shape. It relies on different views to guide an iterative modeling process.

Another modeling approach presented by Lipp et al. [11] following the notation of Müller [13] deals with the aspects of more direct local control of the underlying grammar by introducing visual editing. The idea is to allow modification of elements selected directly in a 3D-view, rather than editing rules in a text based environment. Therefore principles of semantic and geometric selection are combined as well as functionality to store local changes persistently over global modifications.

Model Graphs
Lintermann et al. proposed a modeling method as well as a graphical user interface (GUI) for the creation of natural branching structures [10]. A structure tree represents the modeling process and can be altered using specialized components describing geometry as well as structure. Another type of components can be used for defining global and partial constraints. Components are described procedurally using creation rules which include recursion. The generation of geometric data according to the structure tree is done via a tree traversal where the components generate their geometrical output.

The procedural modeling approach [4] proposed by Ganster et al. describes an integrated framework based on structure trees in a visual language. The infix notation of the language requires the use of variables which are stored on a heap. A graph structure represents the rules used to create an object. Special nodes allow the creation of geometry, the application of operators as well as the usage of control structures. Various attributes can be set for nodes used in a graph. Directed edges between nodes define the order of execution, in contrast to a visual data flow pipeline (VDFP) where data is transported between the different stages.

Hierarchical Description
Finkenzeller presented another approach for detailed building facades [3] called ProcMod. It features a hierarchical description for an entire building. The user provides a coarse outline as well as a basic style of the building including distinguished parts and the system generates a graph representing the building. In the next step, the system traverses the graph and generates geometry for every element of the graph. This results in a generated, detailed scene graph, in which each element can be modified afterwards. The current version has some limitations: for example, organic structures and inclined walls cannot be modeled.

Scripted Modelers
3D modeling software packages like Autodesk Maya™ provide a variety of tools for the modeling process. In addition to a graphical user interface, a scripting language is supplied to extend its functionality. It enables tasks that cannot be achieved easily using the GUI and speeds up complicated or repetitive tasks.

When using parametric tools in modern CAD software products, geometric validity is a subject. For a given parametric model certain combinations of parameter values may not result in valid shapes. Hoffmann and Kim propose an algorithm [8] that computes valid parameter ranges for geometric elements in a plane, given a set of constraints.

Postfix Expressions
Havemann proposes a stack based language for creating polygonal meshes called Generative Modeling Language (GML). The postfix notation of the language is very similar to that of Adobe's Postscript. It allows the creation of high-level shape operators from low-level shape operators. The GML serves as a platform for a number of applications because it is extensible and comes with an integrated visualization engine.

An extended system presented by Mendez et al. combines semantic scene-graph markups with generative modeling in the context of generating semantic three dimensional models of underground infrastructure [12]. The idea is to connect a geospatial database and a rendering engine in order to create an interactive application. The GML is used for on-the-fly generation of procedural models in combination with a conventional scene graph with semantic markup. An augmented reality view of underground infrastructure like water or gas distribution systems serves as a demo application.

WebGL and O3D
WebGL is a JavaScript binding to OpenGL ES 2.0 which enables rich 3D graphics within browsers on platforms supporting the OpenGL or OpenGL ES graphics standards [9]. A main advantage of this upcoming standard is its plugin-free realization within the browser. The WebGL standard will benefit from recent developments in Web technology like the HTML5 specification or the JavaScript performance increases across all major browsers.

Another "Web"-approach is O3D. This is a combination of a browser plugin and a JavaScript API which enables a web developer to create and display 3D scenes [5]. In order to have more control over the performance of the display routines a plugin is used. The JavaScript part is responsible for the control of the plugin.

Both techniques are still under development so that they can hardly be discussed in detail. However, due to

the fact that JavaScript is used as scripting environment, we will be able to support these techniques as soon as they reach a stable status.

3 LANGUAGE ELEMENTS FOR MODELING

When trying to combine different approaches for procedural modeling, a question arises: Is it possible to achieve a conversion between file formats, respectively languages? The simple answer is: Yes, but only with considerable expenditure. Because of differences in the intended purpose of the languages as well as paradigmatic variations it is a rather difficult task. In order to be able to cover a variety of approaches it would be necessary to implement converters that differ in the source as well as in the target language. Therefore a central solution representing a common ground for the procedural modeling approaches is desirable. This solution enables the user to create procedural models represented by a single language, but allows a variety of output representations to be generated.

Consequently, the motivation for this work is to establish a beginner friendly programming language for procedural modeling that serves as a basis to generate code for different target language. In particular the requirements for such a language can be summarized as follows:

- The language should support a user in typical modeling tasks; i.e. it should provide often needed data structures, algorithms and routines for geometric modeling – such as vectors, matrices, etc.
- As our target group is composed mainly of non-computer scientists and creative coders, the language should be beginner friendly and yet powerful. A user with little experience in programming should be able to read the language and use it in a short period of time [6]. More advanced users should not be limited by the language.
- Languages using error-prone techniques (pointers, memory management, etc. [2]) should be omitted; as Niklaus Wirth stated: "The most important decision in language design concerns what is to be left out."

Currently high-level programming languages can be classified in scripting- and in system-languages. The main difference – according to John K. Ousterhout [15] – is the style of programming: scripts are designed for gluing programs and algorithms together, whereas system languages are designed for complex algorithms and data structures. As a result scripting-languages are mostly type-less and more dynamic than system-languages. Of course, due to just-in-time compilation and even more advanced techniques this separation is not clear-cut.

In the domain of procedural modeling we favor a scripting language, as we believe that system languages tend to slow down the creative coding process by programming overhead. Scripting languages tend to be more fault-tolerant and the explanatory power of the source code is promoted. The result of these aspects is presented in the next section.

4 MODELING WITH JAVASCRIPT

The programming language JavaScript meets the requirements mentioned above. It is a structured programming language featuring a rather intuitive syntax, which is easy to read and to understand. As source code is more often read than written, a comprehensible, well-arranged syntax is sensible. JavaScript incorporates features like dynamic typing and first-class functions.

But the most important feature of JavaScript is: it is already in use by non-computer scientists – namely designers and creative coders. JavaScript dialects are used in Adobe Flash (called ActionScript), in the Adobe Creative Suite, in interactive PDF files, in Apple's Dashboard Widgets, in Microsoft's Active Scripting technology, in the VRML97, in the Re-Animator framework, etc. Consequently, a lot of documentation and tutorials to introduce the language exist.

However, in order to be used for procedural modeling, JavaScript is missing some functionality, which will be added by libraries.

Data structures and libraries for modeling

In the specification of JavaScript no data types representing vectors and matrices are defined. These types are an essential part of computer graphics. Therefore, the *Euclides* compiler includes a mathematical library to correct this drawback.

Modifications of the Language

While using JavaScript for procedural models it is our aim to be compliant to the standard ECMAScript (ECMA 262). Hence, we try to support this standard and do not add new language constructs and features, which would result in errors when using a standard JavaScript engine. During the development process the compiler's conformance with the JavaScript standard is tested with JavaScript engines of various web browsers.

5 A GENERATIVE META-MODELER

Our meta-modeler approach differs from other modeling environments in a very important aspect: target independence.

A "normal" generative modeling environment consists of a script interpreter and 3D rendering engine. A generative model (3D data structures with functionality) is interpreted directly to generate geometry, which is then visualized by the rendering engine. In our system a model's source code is not interpreted but parsed into an intermediate representation. After a validation

process it is translated into the target language. The process of

$$\text{parsing} \rightarrow \text{validating} \rightarrow \text{translating}$$

offers many advantages.

The validation step involves syntax and consistency checks. These checks are performed to ensure a correct intermediate representation and to provide meaningful error messages as early as possible within the processing pipeline. Sensible error messages are one of the most – if not *the* most – important aspect of a beginner-friendly development environment.

The consistent intermediate representation serves as a basis for backend exporters to different languages. As our compiler has been designed to export and translate JavaScript to other languages, it includes mechanisms to map JavaScript methods and data types to the target language as well as mechanisms to wrap already existing libraries of a rendering engine. The *Euclides* compiler uses annotation techniques to control this mapping and wrapping process. These annotations are placed in JavaScript comments to ensure 100% compliance with the JavaScript standard. In this way low-level, platform dependent functions – such as a method to draw a single triangle – are wrapped platform independently. During the bootstrapping process of a new exporter a few low-level functions need to be wrapped in this way. All other functions, methods, etc built upon these low-level routines are converted and translated automatically.

6 CONCLUSION AND FUTURE WORK

The analysis of existing procedural modeling tools shows similarities and differences. While some approaches are all-purpose modelers, others are specialized on certain subjects.

A common subset of data types and language constructs to describe 3D geometry has been identified. We integrated this common subset in the scripting language JavaScript and developed a corresponding compiler called *Euclides*. It is suited for procedural modeling, has a beginner-friendly syntax and is able to generate and export procedural code for various, different generative modeling or rendering engines.

This meta-modeler concept allows a user to export generative model to other platforms without losing its main feature – the procedural paradigm. The source code does not need to be interpreted or unfolded, it is translated. Therefore it can still be a very compact representation of a complex model.

The target audience of this approach consists of beginners and intermediate learners of procedural modeling techniques and addresses creative designers who are seldom computer scientists. These experts are needed to tap the full potential of generative techniques.

In the future we will support further target platforms and will concentrate on advanced compiler features.

ACKNOWLEDGMENT

The authors gratefully acknowledge the generous support from the European Commission for the integrated project 3D-COFORM (3D COllection FORMation, www.3D-coform.eu) under grant number FP7 ICT 231809, as well as from the Austrian Research Promotion Agency (FFG) for the research project METADESIGNER (Meta-Design Builder: A framework for the definition of enduser interfaces for product mass-customization), grant number 820925/18236.

REFERENCES

[1] René Berndt, Dieter W. Fellner, and Sven Havemann. Generative 3D Models: a Key to More Information within less Bandwidth at Higher Quality. *Proceeding of the 10th International Conference on 3D Web Technology*, 1:111–121, 2005.

[2] Matt Bishop and Deb Frincke. Teaching Robust Programming. *IEEE Security and Privacy*, 2:54–57, 2004.

[3] Dieter Finkenzeller. Detailed Building Facades. *IEEE Computer Graphics and Applications*, 28(3):58–66, 2008.

[4] Björn Ganster and Reinhard Klein. An Integrated Framework for Procedural Modeling. *Proceedings of Spring Conference on Computer Graphics 2007 (SCCG 2007)*, 23:150–157, 2007.

[5] Google. O3D API. online: http://code.google.com/apis/o3d/, 2009.

[6] David Gries. What have we not learned about Teaching Programming. *IEEE Computer*, 39:81–82, 2006.

[7] Sven Havemann and Dieter W. Fellner. Generative Parametric Design of Gothic Window Tracery. *Proceedings of the 5th International Symposium on Virtual Reality, Archeology, and Cultural Heritage*, 1:193–201, 2004.

[8] Christoph M. Hoffmann and Ku-Jin Kim. Towards valid parametric CAD models. *Computer Aided Design*, 33:81–90, 2001.

[9] Khronos Group. Khronos Details WebGL Initiative to Bring Hardware-Accelerated 3D Graphics to the Internet. online: http://www.khronos.org/news/press/releases/khronos-webgl-initiative-hardware-accelerated-3d-graphics-internet/, 2009.

[10] Bernd Lintermann and Oliver Deussen. A Modelling Method and User Interface for Creating Plants. *Computer Graphics Forum*, 17(1):73–82, 1998.

[11] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive Visual Editing of Grammars for Procedural Architecture. *ACM Transactions on Graphics*, 27(3):1–10, 2008.

[12] Erick Mendez, Gerhard Schall, Sven Havemann, Dieter W. Fellner, Dieter Schmalstieg, and Sebastian Junghanns. Generating Semantic 3D Models of Underground Infrastructure. *IEEE Computer Graphics and Applications*, 28:48–57, 2008.

[13] Pascal Müller, Peter Wonka, Simon Haegler, Ulmer Andreas, and Luc Van Gool. Procedural Modeling of Buildings. *Proceedings of 2006 ACM Siggraph*, 25(3):614–623, 2006.

[14] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based Procedural Modeling of Facades. *ACM Transactions on Graphics*, 28(3):1–9, 2007.

[15] John K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. *IEEE Computer Magazine*, 31(3):23–30, 1998.

[16] Yogi Parish and Pascal Mueller. Procedural Modeling of Cities. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 28:301–308, 2001.

[17] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.