Generic Graph Grammar: A Simple Grammar for Generic Procedural Modelling

Christiansen, Asger Nyman; Bærentzen, Jakob Andreas

Published in:
Proceedings of the 28th Spring conference on Computer Graphics

Link to article, DOI:
10.1145/2448531.2448542

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Christiansen, A. N., & Bærentzen, J. A. (2012). Generic Graph Grammar: A Simple Grammar for Generic Procedural Modelling. In *Proceedings of the 28th Spring conference on Computer Graphics* (pp. 85-92). Association for Computing Machinery. https://doi.org/10.1145/2448531.2448542

Generic Graph Grammar: A Simple Grammar for Generic Procedural Modelling

Asger Nyman Christiansen[*] Jakob Andreas Bærentzen[†]
Department of Informatics and Mathematical Modelling, Technical University of Denmark, Denmark

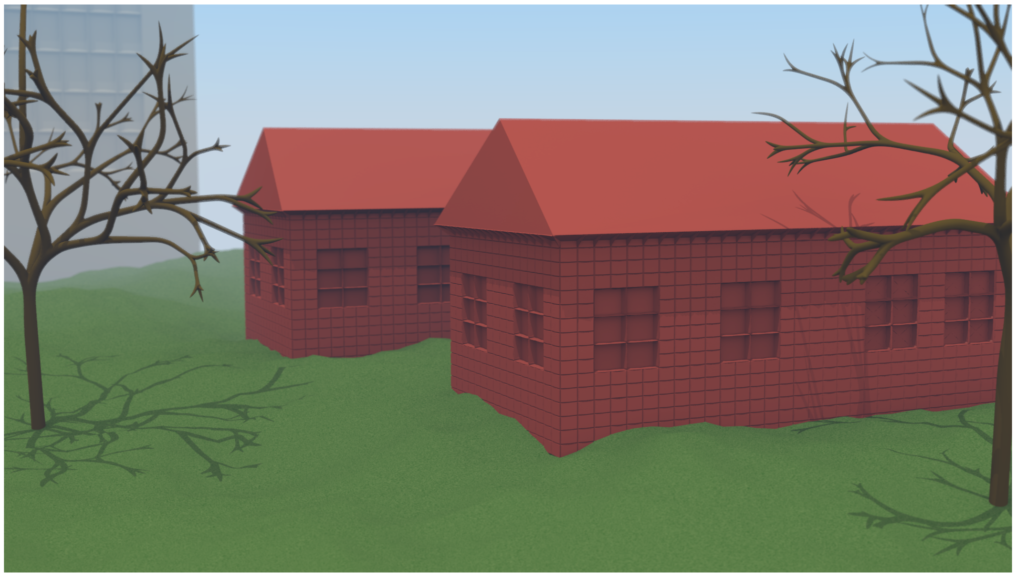


Figure 1: *Examples of objects created by Generic Graph Grammar.*

Abstract

Methods for procedural modelling tend to be designed either for organic objects, which are described well by skeletal structures, or for man-made objects, which are described well by surface primitives. Procedural methods, which allow for modelling of both kinds of objects, are few and usually of greater complexity. Consequently, there is a need for a simple, general method which is capable of generating both types of objects. *Generic Graph Grammar* has been developed to address this need. The production rules consist of a small set of basic productions which are applied directly onto primitives in a directed cyclic graph. Furthermore, the basic productions are chosen such that *Generic Graph Grammar* seamlessly combines the capabilities of L-systems to imitate biological growth (to model trees, animals, etc.) and those of split grammars to design structured objects (chairs, houses, etc.). This results in a highly expressive grammar capable of generating a wide range of types of models. Models which consist of skeletal structures or surfaces or any combination of these. Besides generic modelling capabilities, the focus has also been on usability, especially user-friendliness and efficiency. Therefore several steps have been taken to simplify the workflow as well as to make the modelling scheme interactive. As proof of concept, a generic procedural modelling tool based on *Generic Graph Grammar* has been developed.

[*]e-mail:asny@imm.dtu.dk
[†]e-mail:jab@imm.dtu.dk

CR Categories: F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems— [I.3.5]: Computer Graphics—Computational Geometry and Object Modeling I.6.3 [Simulation and Modeling]: Applications—;

Keywords: Procedural Modelling, Graph Grammars, Skeletons, Interactive Modelling.

1 Introduction

Modelling of 3D digital content, especially for the entertainment industry, is a severe economic challenge due to an increasing demand for realism and a boom in size and number of virtual models [Fletcher et al. 2010]. Automated or at least semi-automated content generation would increase the efficiency of a designer which would enhance the output quality and quantity.

For this reason, there has been an increasing interest in the semi-automated modelling approach called procedural modelling. The idea is to design procedures or rules which generate virtual models automatically, instead of designing models explicitly. Procedural modelling tools are therefore based on rules which, for example, could be defined in production systems or grammars.

Procedural modelling schemes have been developed for decades, but almost all are specialised in generating a single type of model, primarily vegetation or buildings. Some research seeks to develop a more general purpose grammar e.g. [Paoluzzi et al. 1995][Tobler et al. 2002][Havemann 2005][Ganster and Klein 2007][Krecklau et al. 2010]. This tendency has resulted in some expressive, but rather complex modelling schemes, which are often hard to utilise for people with little technical insight. Another trend in recent research is to focus on developing user-friendly procedural modelling tools [Smelik et al. 2010][Lipp et al. 2008][Lintermann and Deussen 1999][Ijiri et al. 2006]. The primary instrument to make procedural modelling user-friendly is to make it entirely interactive.

The main contribution of this research is a simple procedural modelling scheme which is simple yet quite general. The scheme consists of a novel grammar, *Generic Graph Grammar* presented in Section 2, for simple and efficient procedural modelling of a wide variety of 3D models. Furthermore, an interactive modelling tool called *GraphGen*, which is based on *Generic Graph Grammar*, has been developed as proof of concept. Some of the abilities of *GraphGen* are described in Section 3.

The ideas which distinguish *Generic Graph Grammar* from existing procedural techniques and enable general purpose and user-friendly modelling are: Representing an object by a *directed cyclic graph* (Section 2.1), using five basic graph manipulating operators called *commands* (Section 2.2), saving each derivation step in a new *generation* of the graph (Section 2.2.2) and using *conditions* to choose which parts of the graph a command is applied to (Section 2.2.3). Furthermore, *Generic Graph Grammar* is able to procedurally generate a diverse set of skeletal models, in addition to the more traditional surface models. Previously, procedural creation of skeletal structures has only been utilised to model botanical objects. Finally, as with other procedural methods, subdivision of a surface model towards a smooth or fractal limit surface is supported.

1.1 Related work

One of the most fundamental grammars used in procedural modelling is due to Lindenmayer and Prusinkiewicz. Lindenmayer developed a parallel string-based grammar in 1968 called L-systems [Lindenmayer 1968] as a theoretical framework for studying the development of simple multicellular organisms. Prusinkiewicz was the first to use L-systems in computer graphics [Prusinkiewicz 1987] and together Prusinkiewicz and Lindenmayer showed that L-systems can give impressive results when generating geometric plant models [Prusinkiewicz and Lindenmayer 1996].

Another fundamental grammar for procedural modelling is the sequential shape grammars introduced by Stiny in 1975 [Stiny 1975]. Shape grammars develop visual models called shapes and Stiny thereby introduces grammars to the field of modelling. The shape grammar approach often results in complex derivations. Nonetheless, the idea of letting shapes replace shapes in the same manner as symbols replace symbols in traditional string-based production systems has been used in a large variety of later developed grammars. Examples of variations of shape grammars are set grammars [Stiny 1982], split grammars [Wonka et al. 2003] and CGA shape grammars [Müller et al. 2006] which are suited for creating structured, often man-made, objects.

Another relevant extension of string-based production systems is graph grammars [Ehrig et al. 1991]. Graph grammars work directly on graphs, as the name suggests, by using the production rules to replace a part of the graph. This type of grammar has been applied in many areas since graphs are common in a wide range of applications. However, graph grammars have not yet been applied in the area of procedural modelling.

2 *Generic Graph Grammar*

In this section the different parts of *Generic Graph Grammar* (G^3) will be defined and explained along with motivations for the design choices. Consequently, it is then possible to define G^3.

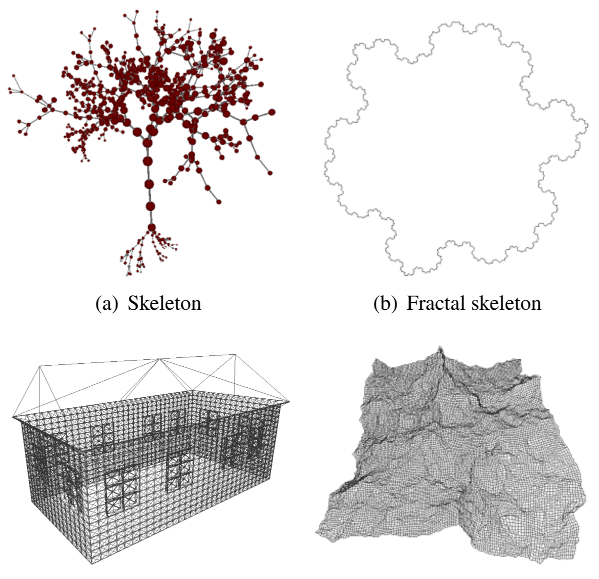


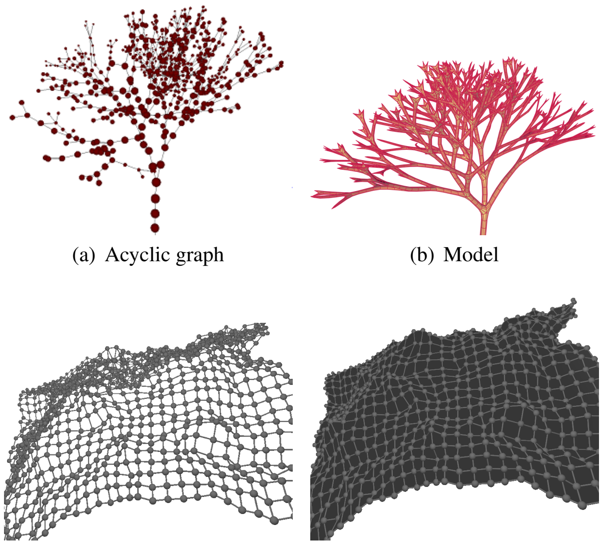
Figure 2: *Shows four examples of graphs created by G^3. The graph in Figure 2(a) represents the topological skeleton of a botanical tree while the graph in Figure 2(b) is a fractal skeleton called a Koch snowflake. Figure 2(c) displays a surface, in the form of a house, and a fractal surface, which resembles terrain, is seen in Figure 2(d).*

2.1 Object representation

Procedural methods have been applied in automatic generation of a wide variety of virtual models and effects. Examples are terrain, water, vegetation, road networks, gasses, liquids, fire, buildings and whole cities [Smelik et al. 2009][Ebert et al. 2002]. As mentioned, the aim of this work is to be able to create a variety of these models. In addition, models which have not previously been generated procedurally such as animals and furniture are sought included in the modelling capabilities of G^3. These capabilities can be achieved by being able to model two types of geometry, skeletal structures and surfaces, and combining these geometries. This claim is supported by the diversity of types of models displayed in Section 4. The geometries are illustrated in Figure 2 and described in the following.

Skeletal structures The natural illustrative example of a topological skeleton [Blum 1967], also known as the medial axis, is replacing the bones in a human with connected line pieces, which thereby constitute the shape of a human. Skeleton models are suited for representing organic shapes, such as trees, animals etc.

Surfaces Traditionally, a surface, usually in the form of a polygon mesh, has been used to represent a virtual model. This is a useful representation when modelling human-made objects, eg. houses and furniture. An important type of surface in computer graphics is fractal surfaces [Mandelbrot 1975]. The definition of fractal geometry is usually simplified in computer graphics to be an object which has a self repeating pattern. This means the pattern is similar at all scales. An example is the serrated look of a mountain, which is more or less the same from a distance, to close zoom on a piece of rock. Fractal geometry describes natural phenomena well and is therefore essential to model terrain and clouds, for example.


(a) Acyclic graph (b) Model

(c) Cyclic graph (d) Model

Figure 3: *Examples of conversion from graph to mesh. The skeletal structure in Figure 3(a) is converted into the mesh in Figure 3(b) by the SQM method. The cyclic graph in Figure 3(c) is directly converted into the mesh in Figure 3(d).*

A directed cyclic graph G (defined in Definition 1) is chosen to represent an object. The main reason for this is its ability to describe the desired types of geometry. Edges can describe skeletons, while faces are able to represent surfaces. Furthermore, an important feature of the graph representation is its ability to naturally combine these different types of geometry.

Definition 1 *A directed cyclic graph $G = \langle P \rangle$ consists of a set of primitives P. A primitive contains both topological and geometric information and is a common term for:*

Nodes *A node $n \in G$ contains a list of neighbouring edges, as well as the position $p \in \mathbb{R}^3$ and the radius of the node $r \in \mathbb{R}$.*

Directed edges *A directed edge $e \in G$ connects the node's tail n_t and head n_h; $e = (n_t, n_h)$. For the sake of fast lookup, it also contains a list of the faces which it is a part of.*

Faces *A face $f \in G$, representing a closed loop, defined by a set of ordered nodes $f = \{n_0, n_1 \ldots, n_n\}$, where two consecutive nodes are connected by an edge; (n_0, n_1) or (n_1, n_0) $\ldots (n_n, n_0)$ or (n_0, n_n). Note that a face is not enforced to be planar.*

It is chosen that all nodes in a graph are connected via a set of edges. This means, for any pair of nodes $n_i \in G$, $n_j \in G$, there exists at least one set of edges; $\{(n_i, n_1), (n_1, n_2), \ldots, (n_{j-1}, n_j)\}$, which connects n_i with n_j.

Another reason for choosing a graph representation is the relatively straightforward conversion from object to model. All acyclic parts of the graph are converted using, for example, the *SQM method* [Bærentzen et al. 2012] or the *B-mesh* approach [Ji et al. 2010]. These methods use the topological information as well as the geometric information (the position and radius of nodes), where the radius of the nodes is used to determine the thickness of the model. The cyclic parts of the graph are directly convertible into mesh by using the topological information and the position of the nodes. Note that in this case, the radii of the nodes are not used. The conversion of both acyclic and cyclic graphs to meshes is illustrated in Figure 3.

2.2 Productions

A directed cyclic graph is able to represent a wide range of models. It is, however, also necessary to be able to develop the topology of this graph in such a way that the wide range of models can be created procedurally. The strategy is to combine the abilities of L-systems [Lindenmayer 1968] in imitating biological growth and the abilities of split grammars [Wonka et al. 2003] in generating structured objects. Combining L-systems and shape grammars has been attempted before, both in the mesh-generation approach by Tobler et al. [Tobler et al. 2002] and in the *Generalized Grammar* framework by Krecklau et al. [Krecklau et al. 2010]. However, these attempts have not succeeded in a seamless integration and especially L-systems have been applied in an unnatural environment. As an example the *Generalized Grammar* approach uses cylinders as a replacement for line pieces when imitating L-systems.

To achieve a seamless imitation and combination of both modelling schemes, G^3 is applied directly onto the primitives of the graph. Furthermore, a set of productions which consists of both grow and split rules is chosen. The grow rules make it possible to model biological growth of organic objects, whereas the split rules allow for easy construction of models of man-made objects as well as natural phenomena such as terrain. These productions are called *commands* which is a basic notion of G^3. Commands are very basic operators, much like the Euler operators used by Havemann [Havemann 2005]. They are chosen such that they are the minimal set but still allows for general and simple development of skeleton structures and surfaces.

Commands are applied in parallel to all primitives of a specific type in G. For example, if the command *Split edge* is applied, it is applied to all edges in G simultaneously. The commands thereby resemble the productions of graph grammars [Ehrig et al. 1991] with the significant difference that the productions are applied to primitives only, not sub-graphs. All of the commands add new primitives to the graph and may change its topology, but they will not change the geometric information contained in the existing primitives.

To understand how the commands operate, one has to know the terms *parameters*, *conditions* and *generations of graphs*. These topics are described in the following, followed by a formal definition of the five commands and a description of how these commands are used for the derivations of G^3.

2.2.1 Parameters

A set of parameters $u = \{r, H, L, U, l, \sigma_r, \sigma_\rho, \sigma_\phi, \sigma_\theta, \sigma_l\}$ is given as input to each of the commands. These parameters are the same for all commands and consist of:

Radius r Determines the radius of the node created when applying the command.

Headings H**,** L **and** U The turtle representation in three dimensions [Prusinkiewicz and Lindenmayer 1996][diSessa and Abelson 1981] is adopted in this research, which is the reason for using a *directed* graph. The idea is that the turtle has a heading H, a direction to the left L and a direction up U. These directions are represented by unit vectors, which are perpendicular to each other and incorporated into the edge data structure. The headings are used when a command creates a new edge, both to determine the headings of the edge and to determine the position of the new head node.

Length l This is used to determine where to position the new node created by the applied command. The position is determined

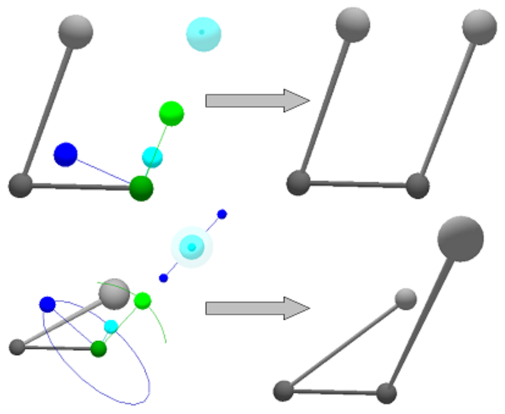


Figure 4: *Illustrates how the parameters are selected interactively by drag and drop. The dark green sphere represents the point of origin p_0, where the green, turquoise and blue edges represent the headings H, L and U respectively. The length l and radius r parameter can be determined by the position and radius of the turquoise sphere. In addition, the bottom figure illustrates how to set the variation parameters. The variation in radius σ_r is shown by three opaque spheres, where the medium size is a sphere with radius r, the largest is a sphere with radius $r + \sigma_r$ and the smallest is a sphere with radius $r - \sigma_r$. The variation in length σ_l is illustrated by a blue edge having minimum and maximum highlighted by a blue sphere. The variations in roll σ_ρ, pitch σ_ϕ and turn σ_θ are illustrated by blue, green and turquoise parts of circles respectively.*

by $p = p_0 + H \cdot l$, where p_0 is some starting point which depends on the command.

Variation in radius σ_r Introduces random variation in the radius r by utilising a uniformly distributed random variable $r = U(r - \sigma_r, r + \sigma_r)$ for each operation on a primitive.

Variations in roll σ_ρ**, pitch** σ_ϕ **and turn** σ_θ Introduce random variations in the headings H, L and U. A uniformly distributed random variable is utilised to find the roll $\rho = U(-\sigma_\rho, \sigma_\rho)$, pitch $\phi = U(-\sigma_\phi, \sigma_\phi)$ and turn $\theta = U(-\sigma_\theta, \sigma_\theta)$. The headings are then rotated ρ around H, ϕ around L and θ around U.

Variation in length σ_l Introduces random variation in the length l by utilising a uniformly distributed random variable $l = U(l - \sigma_l, l + \sigma_l)$.

How these parameters are set interactively is shown in Figure 4.

Sometimes it is preferred to define parameters of a command relative to the preceding primitive, instead of defining the parameters in an absolute manner. This is especially useful when imitating biological growth. For example the radius r_1 of a new node n_1 created by a command with parameter r, can be determined relative to the radius r_0 of the preceding node n_0. This is calculated by $r_1 = r \cdot r_0$ instead of absolute calculation $r_1 = r$. How to define r_0 and other preceding parameters is sometimes straight-forward and sometimes not obvious. In this implementation, a choice of averaging the parameters of the preceding primitives has been made, if several primitives precede the new primitive. Figure 5 illustrates the difference between absolute and relative parameters.

2.2.2 Generations of graphs

The term *generation* is a very simple and intuitive concept. The derivation of G^3 simply starts with a graph of generation zero G_0.

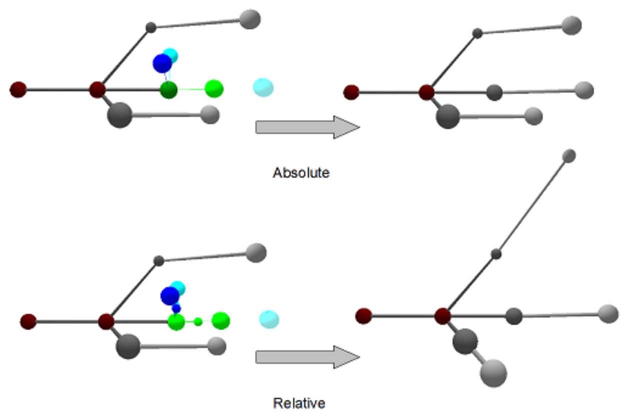


Figure 5: *The parameters of a Grow edge command applied to the green and the two dark gray nodes are set in an absolute (top) and relative (bottom) manner with respect to the green node. Notice that choosing relative headings can be used to grow the model in a fashion resembling L-systems.*

Then, each time a command is applied to a graph of generation g; G_g, this graph is copied into a new generation of the graph G_{g+1}. The operations are then performed only on graph G_{g+1}, where graph G_g is used to determine which primitives obey the conditions v. Therefore, n steps of a derivation of G^3 consist of a list of generations of graphs; $G_0, G_1, \dots, G_n$.

The generations of graphs are necessary to avoid order dependency, when applying conditions. Applying a command to a primitive $a \in G_g$, which fulfils the conditions v, may change a neighbouring primitive $b \in G_g$, resulting in b no longer fulfilling the conditions v. This will result in not applying the rule to b, even though it fulfilled conditions v. The resulting derivation would thereby be dependent on the order of application. These kind of problems are eliminated using the generation terminology and commands can thereby be applied in parallel.

2.2.3 Conditions

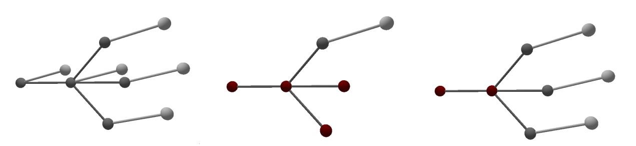
Conditions are introduced to be able to select to which primitives a command is applied. A condition is a Boolean statement that tells whether a primitive obeys some property (geometric, topological etc.). The types of conditions v are different for each primitive and is therefore divided into three sets: one for nodes v_n, edges v_e and faces v_f. Only if a primitive obeys all of the selected conditions will the command be applied to it. Furthermore, saving generations of the graph has made it possible to condition on primitives as they appeared in another generation of the graph $G_{g-i},\ i \geq 1$ than the current generation G_g. Examples of conditions are given below and illustrated in Figure 6.

Node conditions v_n**:** random r, age a, number of neighbouring edges $\#e_{ne}$, number of faces $\#f$, number of outgoing edges $\#e_{ou}$, number of incoming edges $\#e_{in}$, position p.

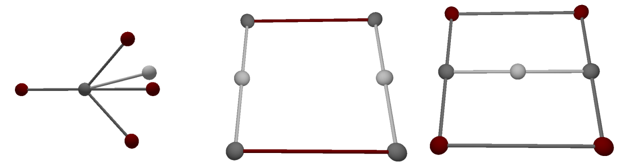
Edge conditions v_e**:** random r, age a, number of neighbouring edges $\#e_{ne}$, number of faces $\#f$, number of outgoing edges $\#e_{ou}$, number of incoming edges $\#e_{in}$, position p, heading H.

Face conditions v_f**:** random r, age a, number of nodes $\#n$, number of edges $\#e$, position p.

One should notice that the sets of conditions are not considered complete and, presumably, other conditions are needed to achieve



(a) $C_{CE}(u, v_n = \{\})$ (b) $C_{CE}(u, v_n = \{\#e_{in} > 0, H_y > 0\})$ (c) $C_{CE}(u, v_n = \{a < 3\})$



(d) $C_{CE}(u, v_n = \{\#e_{ou} = 0 \in G_{g-3}\})$ (e) $C_{SE}(u, v_e = \{H_x = 0\})$ (f) $C_{SF}(u, v_f = \{\#n = 6\} \cup v_n = \{a = 0\})$

Figure 6: *A command is applied simultaneously to all primitives which fulfil all of a set of conditions v. These primitives are colored dark gray, while the primitives that do not fulfil the chosen conditions are colored red. The new primitives, added to the graph after applying the command in the caption, is colored light gray. Figure 6(d) illustrates how to condition on the primitives in G_{g-3} instead of G_g. Figure 6(f) shows that it is necessary to condition on both nodes and faces, when applying a Grow face or Split face command.*

easy and generic modelling. Also, the conditions are currently limited to condition on local properties, but could be extended to be able to condition on the geometry or topology of larger parts of the graph. This is possible since all geometric and topological information are available. Consequently, G^3 would be able to imitate geometric matching as found in shape grammars.

2.2.4 Commands

Finally, it is possible to give a formal definition of a command.

Definition 2 *A command C is a parametric and conditional production, which transforms a graph from generation g to generation $g + 1$:*

$$C(u, v) : G_g \to G_{g+1}$$

A primitive $a \in G_g$, which obey the conditions v, are converted to a set of connected primitives $B \subseteq G_{g+1}$, according to the type of command and the parameters $u = \{r, H, L, U, l, \sigma_r, \sigma_\rho, \sigma_\phi, \sigma_\theta, \sigma_l\}$.

This is a very general definition and may not give the best intuition to how the five commands are applied in practice. Therefore, the five commands are described in detail below and illustrated in Figure 7. In the following, the set of parameters u will be as described in Section 2.2.1 and the set of conditions v_n, v_e and v_f are as described in Section 2.2.3. To simplify the description, variation in parameters is omitted.

Create edge Assume graph G_g contains a node n_0 situated at position p_0 and a *Create edge* command $C_{CE}(u, v_n)$ is applied to G_g. If the node n_0 obeys the conditions v_n, a node n_1 at position $p_1 = p_0 + H \cdot l$ with radius $r_1 = r$ and an edge $e = (n_0, n_1)$ are created.

Create face Assume graph G_g contains an edge $e_0 = (n_0, n_1)$ where the position of n_1 is p_1 and a *Create face* command $C_{CF}(u, v_e)$ is applied to G_g. If e_0 obeys the conditions v_e, a new node n_2 is created at $p_2 = p_1 + H \cdot l$ with radius $r_2 = r$. Furthermore, two new edges are created $e_1 = (n_1, n_2)$ and $e_2 = (n_2, n_0)$ as well as a face $f = (n_0, n_1, n_2)$.

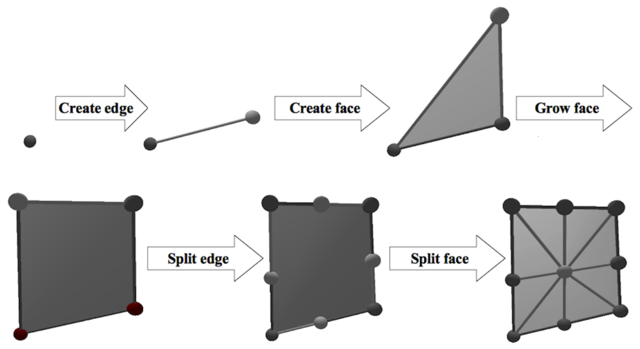


Figure 7: *Simple examples of all of the five basic commands. Notice that a command is applied to all primitives in a graph simultaneously, which is illustrated by the Split edge command. This figure also illustrates the concept of rules, since a rule consists of a set of commands applied in sequence. For example Create edge, Create face and Grow face could be a rule called Create square.*

Grow face Assume graph G_g contains a face $f = (n_0, \ldots, n_i, n_{i+1}, \ldots, n_n)$ where the position of n_i is p_i and a *Grow face* command $C_{GF}(u, v_n \cup v_f)$ is applied to G_g. If f obeys the conditions v_f, then for each node n_i which obeys the conditions v_n, a node n_{n+1} is created at position $p_{n+1} = p_i + H \cdot l$ with radius $r_{n+1} = r$. Furthermore, an edge $e_i = (n_i, n_{n+1})$ is created and the edge $e_{i+1} = (n_i, n_{i+1})$ is changed to $e_{i+1} = (n_{n+1}, n_{i+1})$. Also, the face f is changed to $f = (n_0, \ldots, n_i, n_{n+1}, n_{i+1}, \ldots, n_n)$.

Split edge Assume graph G_g contains an edge $e_0 = (n_0, n_1)$ where the position of n_0 is p_0 and the position of n_1 is p_1 and a *Split edge* command $C_{SE}(u, v_e)$ is applied to G_g. If e_0 obeys the conditions v_e, a new node n_2 is created at $p_2 = \frac{p_0 + p_1}{2} + H \cdot l$ with radius $r_2 = r$. In addition, an edge $e_1 = (n_2, n_1)$ is created and e_0 is changed to $e_0 = (n_0, n_2)$.

Split face Assume graph G_g contains a face $f = (n_0, \ldots, n_i, \ldots, n_n)$ where the position of n_i is p_i and a *Split face* command $C_{SF}(u, v_n \cup v_f)$ is applied to G_g. If f obeys the conditions v_f, a node n_{n+1} is created at position $p_{n+1} = \frac{1}{n} \sum_{i=0}^{n} p_i + H \cdot l$ with radius $r_{n+1} = r$. If n_i obeys conditions v_n, then an edge $e_i = (n_i, n_{n+1})$ is created. Furthermore, the face f is replaced by a set of new faces. The number of new faces is determined by the number of nodes which obey the conditions v_n. Each of the new faces is defined as $f_i = (n_i, \ldots, n_{i+m}, n_{n+1})$ where n_i and n_{i+m} obey v_n and the nodes $n_{i+1}, \ldots, n_{i+m-1}$ do not obey v_n.

2.3 Grammar

It is now possible to define G^3.

Definition 3 *Generic Graph Grammar $G^3 = \langle P, G_0, S_C \rangle$ consists of a set of primitives P and an initial graph G_0, which often consists of a single node $G_0 = \{n_0\}$. Furthermore, G^3 consists of the set of the five parametric and conditional commands S_C.*

Notice that G^3 does not include any terminals, since the primitives (nodes, edges and faces) to which the commands are applied, are always non-terminals.

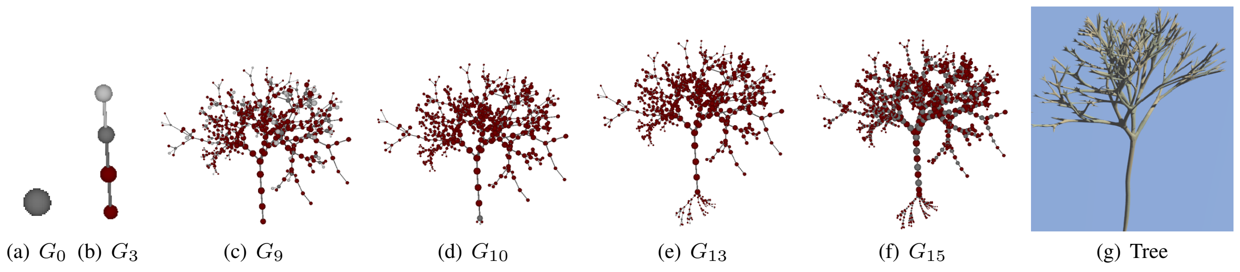

(a) G_0 (b) G_3 (c) G_9 (d) G_{10} (e) G_{13} (f) G_{15} (g) Tree

Figure 8: *Illustrates a derivation of G^3 which results in a skeleton model of a tree (note that the displayed skeleton model is not the same as the rendered. However, the same derivation has been used to create both models).*
Derivation: $[[R_{3 \cdot CE}(u^3, v^3) : G_0 \rightarrow G_3], [R_{6 \cdot CE}(u^6, v^6) : G_3 \rightarrow G_9], [C_{CE}(u, v_n) : G_9 \rightarrow G_{10}], [R_{3 \cdot CE}(u^3, v^3) : G_{10} \rightarrow G_{13}], [R_{SE,EN}(u^2, v^2) : G_{13} \rightarrow G_{15}]]$

In practice, a derivation of G^3 consists of an initial graph G_0 and an ordered list of commands $L_C = [[C_0(u_0, v_0)], [C_1(u_1, v_1)], \ldots, [C_n(u_n, v_n)]]$. To automatically generate the final graph G_n, the commands in L_C are applied in sequence to G_0; $[[C_0(u_0, v_0) : G_0 \rightarrow G_1], [C_0(u_1, v_1) : G_1 \rightarrow G_2], \ldots, [C_n(u_n, v_n) : G_{n-1} \rightarrow G_n]]$. The initial graph G_0 is predefined in the grammar, but the type and order of commands as well as the parameters and conditions for each of these commands are specified by the user. It is these specifications that determine how the final graph G_n (and model) will look like. The workflow for developing this type of derivation will be elaborated in Section 3.2.

It is worth noticing that G^3 is deterministic since the ordered set of commands and appurtenant parameters and conditions, uniquely defines the derivation.

2.4 Rules

To define each of the commands, parameters and conditions in a derivation can be trivial, even though the number of commands needed is usually low (n is small) and the parameters can be set interactively. Therefore, the notion of rules is introduced. A rule L_C is simply a derivation or a part of a derivation, i.e. an ordered list of commands $L_C = [[C_g(u_g, v_g)], [C_{g+1}(u_{g+1}, v_{g+1})], \ldots, [C_{g+m}(u_{g+m}, v_{g+m})]]$ where $0 \leq g \leq g+m \leq n$. This highly simplifies the construction of derivations, which is best illustrated by an example. If it is desired to create a derivation which generates a model of a house, then a derivation which generates a window can be created and saved in a rule. This rule can then be applied in the creation of different types of houses and consequently one does not need to apply the set of commands making up a window more than once. A formal definition of rules is given in Definition 4 and a example of a derivation utilising rules is displayed in Figure 8.

Definition 4 *A rule R is a parametric and conditional production which transforms a graph from generation g to generation $g+m$:*

$$R(u^m, v^m) : G_g \rightarrow G_{g+m}$$

A rule consists of m commands applied in sequence

$$[[C_1(u_1, v_1) : G_g \rightarrow G_{g+1}], [C_2(u_2, v_2) : G_{g+1} \rightarrow G_{g+2}],$$

$$\ldots,\ [C_m(u_m, v_m) : G_{g+m-1} \rightarrow G_{g+m}]]$$

where $v_i \in v^m$ and $u_i \in u^m$ for $i = 1, .., m$.

3 *GraphGen*

Interactive manual modelling is very intuitive, even though it often requires training to use the manual modelling tools existing today. Designing procedures on the other hand, is not intuitive and thereby requires even more training if the procedural tools are not implemented in a user-friendly manner. At worst a procedural tool is usable by persons having deep technical insight only, eg. if designing procedures involves grammar notation. An example of such a tool is the *Generalized Grammar* framework [Krecklau et al. 2010] which is usable by a programmer, but not by a designer. However, focus has recently been on usability of procedural tools and especially interactive procedural modelling has been a popular topic [Smelik et al. 2010][Lipp et al. 2008].

To show that G^3 is useful in practice, a modelling tool called *GraphGen* has been developed and is described in this section. When developing this tool, focus has been on usability and therefore interactive modelling has been incorporated. This implies that no grammar notation is necessary, but is still possible via editing of saved rules which may be preferred by users having technical insight. Interactive modelling is introduced in the form of drag and drop capabilities. This is used to set the parameters of commands, as described in Section 2.2.1, as well as to apply geometric editing, described in the following section. It is worth noticing that interactive modelling could not be achieved if the generations of graphs were not saved.

3.1 Geometric editing

Besides the grammar, which changes the topology of the graph, it is convenient to have geometric editing tools. Two such tools are a part of the implementation of G^3; one for manual editing and one for procedural editing. Both of these tools edit the geometric information, the position and the radius, of the nodes of the graph only.

The manual editing tool is an advantage since procedural modelling is inefficient when designing unique details in the model, in the same manner as manual modelling is inefficient when designing repeating patterns. Furthermore, when modelling procedurally, the user does not always have full control of local details in the model and some things may therefore be impossible to model satisfactory. Manual editing is therefore an easy and efficient way to create the finishing touch. Manual editing is implemented such that it is possible to drag and drop nodes to a new position and to set the radius of individual nodes interactively by again applying drag and drop To ensure consistency, manual changes are overwritten if the current or preceding commands are edited which is not a sustainable solution. Instead one could use instance locators as proposed by Lipp et al. [Lipp et al. 2008].

The procedural editing tool consists of a procedure called *Edit node*, which resembles a command in how it is applied. The difference is, however, that no topological changes are made when applying this tool. This tool is essential when subdividing a curve and, in addition, useful when creating a wide range of models. Since *Edit node* resembles a command, it is described in much the same manner. Furthermore, it can also be applied in an absolute or relative manner.

Edit node Assume graph G_g contains a node n_0 situated at position p_0 with radius r_0 and an *Edit node* procedure $C_{EN}(u, v_n)$ is applied to G_g. If the node n_0 obeys the conditions v_n, the node changes position $p_0 = p_0 + H \cdot l$ and radius $r_0 = r$.

3.2 Workflow

The basic concept behind modelling using *GraphGen* is to choose which commands to apply and in which order. Additionally, the parameters and conditions of the applied commands are chosen. These selections are made manually by the user to develop the desired model.

Since all generations of the graph, i.e. the whole derivation, are saved, it is possible to edit parameters and conditions in any derivation step. When a derivation step has been altered the derivation is regenerated from this step and onward which ensures fast and constant visualisation. Furthermore, saving the generations allows the user to jump through the growth of the model, thereby visualising any derivation step. This results in a detailed overview of all of the modelling steps.

The introduction of rules infers using a hierarchical modelling approach, which means to apply rules in sequence before saving the derivation in a new rule. This can be repeated to make an increasingly larger model, while still applying only few rules at a time. A rule can therefore both be a procedure, which generates a, possibly repeating, part of a model (e.g. a window) or an entire model (e.g. a house). This eliminates the need for time-consuming remodelling when a model or a part of a model is reused.

It is also possible to delete derivation steps, making it possible to edit which commands to apply or the order of applied commands. If a derivation step is deleted, all of the subsequent derivation steps are also deleted, which ensures the consistency of the model. This is a disadvantage, since, if the designer desires to delete a derivation step in the beginning of a derivation, the whole derivation has to be redesigned. This problem is minimised by the hierarchical modelling approach, since this scheme limits the number of rules to apply before the derivation is re-established.

4 Results

Results are depicted in Figures 1, 8 and 9. All graphs are generated using *GraphGen* and all images are rendered using Blender [Blender Foundation 2002]. Some graphs have been converted from graph to polygon mesh using the *SQM method* by Bærentzen et al. [Bærentzen et al. 2012] and all other graphs have been directly converted to a mesh as described in Section 2.1.

The generation of graphs, as well as the conversion from graph to mesh, is fast. The whole derivation, of even the most complicated structures, only takes a couple of seconds to generate and it takes less than a second to apply a new command to a complex graph, which enables interactive modelling.

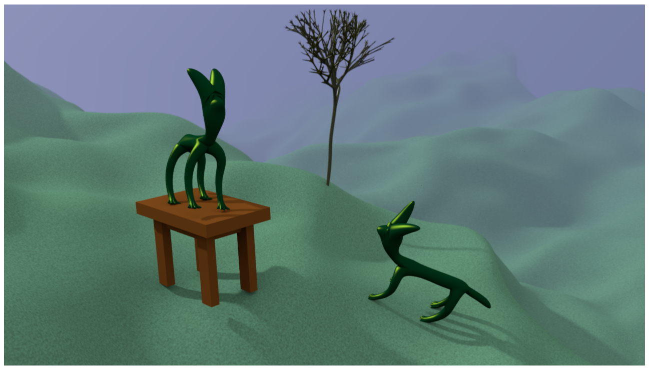


Figure 9: *Objects which are created by Generic Graph Grammar.*

5 Conclusion

A novel grammar G^3 for general purpose and user-friendly procedural modelling has been presented and the usefulness of this grammar has been shown in practice by the capabilities of the procedural modelling tool *GraphGen*.

Concerning the generic modelling capabilities, the presented results clearly show that a wide range of different types of models can be procedurally generated using G^3. G^3 therefore combines the modelling abilities of several specialised tools. In addition, the ability to procedurally generate some models, which cannot be procedurally generated by any existing specialised tool. Most generic tools can of course compete on this subject, but often these tools are not as simple as G^3 and the ability of G^3 to procedurally generate a large selection of varying skeleton models is quite unique.

The simple workflow and the interactive controls make *GraphGen* an overall user-friendly tool which primarily has one flaw only, namely conditions. These are cumbersome to apply in their current version and in some cases they even limit the expressiveness of G^3. These troubles are primarily due to the local nature of conditions, i.e. it is currently only possible to condition on local properties. Expanding the set of conditions to be able to also condition on parts of the graph, instead of just primitives, is therefore a key issue in future development of G^3.

The effectiveness of *GraphGen* in terms of modelling time is not at the level of specialised tools. This weakness is due to the manual selection of commands and setting of parameters and conditions for each command. This is obviously not as efficient as specifying the few parameters which are necessary to create a model using a specialised tool. It is, however, worth noticing that when a derivation of G^3 has been created and saved in a rule, this rule can be reused by changing only few, if any, parameters. The effectiveness in terms of runtime, on the other hand, is high since it takes a few seconds to generate large and complex models.

6 Future work

A limitation to G^3 is the lack of its ability to generate volumes. However a relatively straight-forward addition to the set of commands would make this possible. To follow the structure of the commands, these additions should include *Create volume*, *Grow volume* and *Split volume* commands. It is of course also necessary to expand the data structure to feature a volume primitive, which naturally would be a polyhedron.

Another limitation of G^3, which has already been mentioned, is the notion of conditions which are tedious to use in practice and limits the expressiveness. Adding the possibility to apply conditions globally instead of just locally would resolve this limitation. Note also that this will make G^3 able to imitate shape grammars, since this extension of conditions is equivalent to geometric matching. Furthermore, making the process of selecting conditions interactive would improve the usability of *GraphGen* immensely.

Other improvements of G^3 include preventing self-intersections and modelling additional properties such as normals, texture etc. In addition to the improvements of G^3, the improvements of *GraphGen* are plenty since *GraphGen* is proof of concept and not a fully developed commercial tool. Here, an interesting improvement is to optimise for speed, for example by utilising parallel computing, which possibly would make real-time editing feasible.

Acknowledgements

To Vaida Laganeckiene, for modelling efforts.

References

BÆRENTZEN, J. A., MISZTAL, M. K., AND WELNICKA, K. 2012. Converting skeletal structures to quad-dominant polygonal meshes (under review). In *Proceedings of the Shape Modeling International 2012 (SMI'02)*, IEEE Computer Society, Washington, DC, USA.

BLENDER FOUNDATION, 2002. Blender. Available from `http://www.blender.org/`.

BLUM, H. 1967. A Transformation for Extracting New Descriptors of Shape. In *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn, Ed. MIT Press, Cambridge, 362–380.

DISESSA, A. A., AND ABELSON, H. 1981. *Turtle Geometry: the computer as a medium for exploring mathematics*. MIT Press, Cambridge, MA.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

EHRIG, H., KREOWSKI, H.-J., AND ROZENBERG, G., Eds. 1991. Graph-Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5-9, 1990, Proceedings, vol. 532 of *Lecture Notes in Computer Science*, Springer.

FLETCHER, D., YUE, Y., AND KADER, M. A. 2010. Challenges and perspectives of procedural modelling and effects. In *Proceedings of the 2010 14th International Conference Information Visualisation*, IEEE Computer Society, Washington, DC, USA, IV '10, 543–550.

GANSTER, B., AND KLEIN, R. 2007. An integrated framework for procedural modeling. In *Spring Conference on Computer Graphics 2007 (SCCG 2007)*, Comenius University, Bratislava, M. Sbert, Ed., 150–157.

HAVEMANN, S. 2005. *Generative Mesh Modeling*. PhD thesis, Institute of Computer Graphics, Braunschweig Technical University.

IJIRI, T., OWADA, S., AND IGARASHI, T. 2006. The sketch l-system: Global control of tree modeling using free-form strokes. In *In Smart Graphics*, 138–146.

JI, Z., LIU, L., AND WANG, Y. 2010. B-mesh: A modeling system for base meshes of 3d articulated shapes. *Computer Graphics Forum (Proceedings of Pacific Graphics) 29*, 7, 2169–2178.

KRECKLAU, L., PAVIC, D., AND KOBBELT, L. 2010. Generalized use of non-terminal symbols for procedural modeling. *Computer Graphics Forum 29*, 8, 2291–2303.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development: Parts I and II. *Journal of Theoretical Biology 18*, 280–315.

LINTERMANN, B., AND DEUSSEN, O. 1999. Interactive modeling of plants. *IEEE Computer Graphics and Applications 19*, 1, 56–65.

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

MANDELBROT, B. B. 1975. Stochastic models for the earth's relief, the shape and the fractal dimension of the coastlines, and the Number-Area rule for islands. *Proceedings of the National Academy of Sciences of the United States of America 72*, 10, 3825–3828.

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

PAOLUZZI, A., PASCUCCI, V., AND VICENTINO, M. 1995. Geometric programming: a programming approach to geometric design. *ACM Trans. Graph. 14* (July), 266–306.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1996. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA.

PRUSINKIEWICZ, P. 1987. Applications of l-systems to computer imagery. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, Springer-Verlag, London, UK, 534–548.

SMELIK, R. M., TUTENEL, T., BIDARRA, R., DE KRAKER, K. J., AND GROENEWEGEN, S. A. 2009. A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*.

SMELIK, R. M., TUTENEL, T., DE KRAKER, K. J., AND BIDARRA, R. 2010. Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ACM, New York, NY, USA, PCGames '10, 2:1–2:8.

STINY, G. N. 1975. *Pictorial and formal aspects of shape and shape grammars and aesthetic systems.* PhD thesis, University of California.

STINY, G. N. 1982. Spatial relations and grammars. *Environment and Planning B: Planning and Design 9*, 1 (January), 113–114.

TOBLER, R. F., MAIERHOFER, S., AND WILKIE, A. 2002. A multiresolution mesh generation approach for procedural definition of complex geometry. In *Proceedings of the Shape Modeling International 2002 (SMI'02)*, IEEE Computer Society, Washington, DC, USA, 35–.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Transaction on Graphics 22*, 3 (July), 669–677. Proceedings ACM SIGGRAPH 2003.