Mesh-Subdivision Methods in CGA Shape Grammars

Frederico Dusberger*
Institute of Computer Graphics and Algorithms
Vienna University of Technology

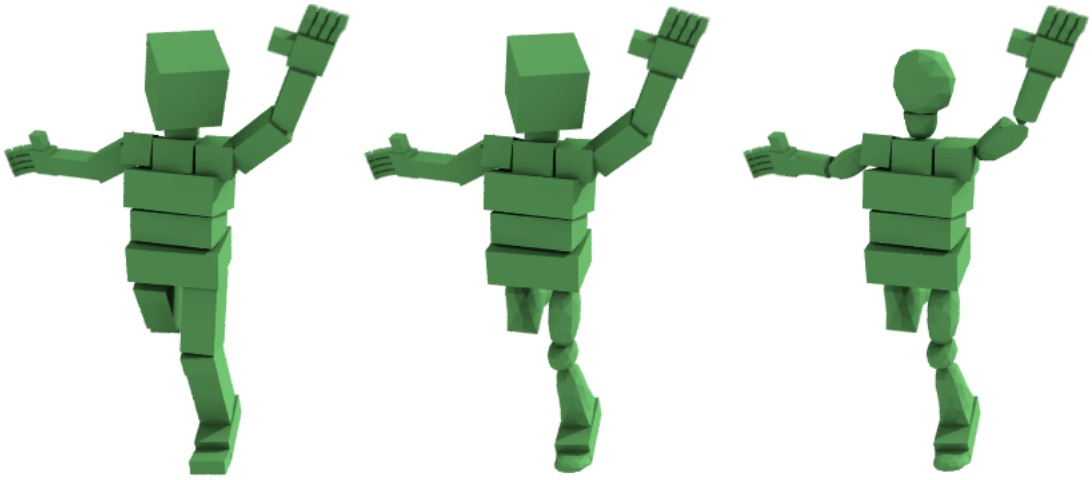


Figure 1: A model of a robot demonstrating the refinement of selected body parts by mesh-subdivision.

Abstract

This report describes the application of mesh-subdivision algorithms in the context of CGA shape, a shape grammar for Computer Generated Architecture. Constituting a procedural modeling approach, CGA shape allows the creation of models under moderate effort, which can furthermore be varied easily by the possibility to parametrize the rules of CGA shape.
At the Vienna University of Technology a CGA shape implementation is being designed and extended by a skeleton system, allowing an easier variation of poses for humanoid models. In this report we follow this idea and want to further extend CGA shape by mesh-subdivision in order to achieve a higher quality and complexity in the resulting models. We discuss mesh-subdivision in general and evaluate the results of an implementation in the existing software system of the Vienna University of Technology.

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

Keywords: procedural modeling, shape grammars, mesh-subdivision

1 Introduction

In this report we discuss the possibilities of using mesh-subdivision techniques in connection with procedural modeling, more precisely a shape grammar. The goal is to further refine automated generation of highly detailed models and thereby enhance their quality to a level at which the results are comparable to manually created models, i.e. models that were developed in a mainly-human guided process under application of conventional digital modeling techniques.
Furthermore, we describe our integration of mesh-subdivision in the CGA Shape grammar implementation that is currently being developed at the Vienna University of Technology

1.1 Motivation

The standards expected from modeling artists regarding the complexity of their work are rising continuously. More and more details have to be modeled in order to tap the full potential of current computer hardware and produce state-of-the-art content for the movies and computer games of tomorrow. It is understood, that in the course of this trend the costs and effort necessary to create this content are likewise increasing. Thus techniques allowing the automated generation of highly detailed models are required to help artists to keep up with this development. There is a variety of these so-called procedural modeling techniques (e.g. fractals or grammars) which currently form a very active and interesting field of

*e-mail: fdusberger@gmail.com

research.
Maybe the most promising approach is the use of shape grammars [Stiny 1975] which in general work like an L-system [Lindenmayer 1968] with the difference of operating on geometric shapes instead of strings. Using only a small set of production rules it is possible to create models with a highly complex geometry. It has been shown that realistic, highly detailed models of cities can be generated by the use of shape grammars [Müller et al. 2006]. However there are some drawbacks to this approach:

- No dynamic models:
  The models are static and it is not possible to add movable parts to them allowing them to appear in different poses.
- Automated process:
  As powerful as this automated method of model generation may be, in some occasions we want to edit the results manually. Imagine a highly complex city with thousands of buildings, created by a shape grammar. Maybe there were enough production rules to define about 30 different types of buildings. The result would still not be as satisfying as it would have been with a conventional modeling process. What is missing are particular details and characteristics that make the buildings unique. To create a realistic model, we would probably like to reshape some parts here and there to give the ensemble a convincing look. Sadly such a reshaping is not possible without writing an immense amount of production rules and forfeiting the advantages of procedural modeling.
- Difficulties with refinement of the model:
  We can easily write production rules to generate high-resolution models consisting of simple primitives. But as soon as we want to add fine smooth details to our model it will become very difficult or even impossible to represent these by using only basic primitives. We need a possibility to smoothen models with a relatively simple method, if we want preserve the strengths of shape grammars.

Regarding the former issue, there is already a solution which we present shortly in the following sections. This report describes our approach that aims to cope with the third issue. By means of mesh-subdivision we extend the current CGA Shape implementation, making it possible to refine and smoothen a mesh enhancing the optical quality.

1.2 Overview

The report is structured as follows: In the following section we list the related work and show how it contributed to our work and this report. Section 3 shortly talks about conventional modeling and its disadvantages, in order to present CGA shape, a procedural modeling approach, as a possible remedy. We then summarize the functionality of CGA shape using the current work on this approach at the Vienna University of Technology as an example of an actual implementation.
The main part, Section 4, is concerned with the discussion on mesh-subdivision, its possible use in shape grammars, followed by a presentation and discussion of an integration into our system.
The last section concludes this report with a summarizing discussion on the examined methods.

2 Related Work

The related work, our implementation and this report are based on, can be divided into two different fields of computer graphics. On the one hand, there is procedural modeling, the general approach our CGA Shape implementation is following. And on the other hand, there is of course mesh-subdivision.

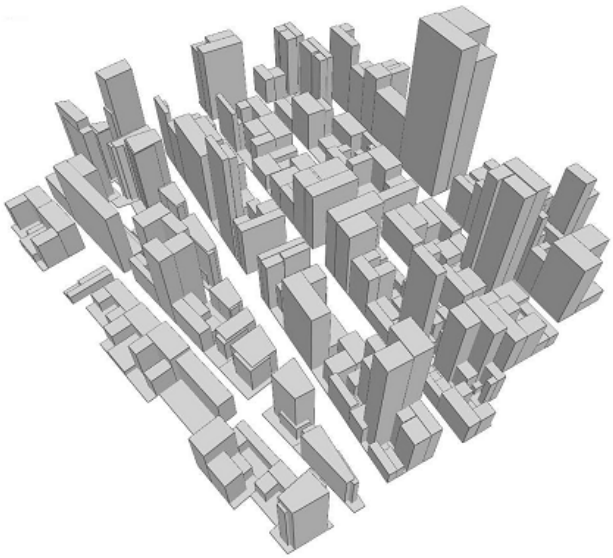


Figure 2: Building mass models generated with only four rules (starting with the building lot as axiom). *Source: [Mueller et al. 2006]*

2.1 Procedural Modeling

Since procedural modeling is such an active field of research there is of course a great number of publications on different approaches to the issue. In addition to those mentioned in the introduction there are some more that have to be mentioned as a basis for this paper. The original L-systems have produced impressive results in the modeling of plants. Being parallel grammars they were perfectly suitable considering the main objective of growth over time [Prusinkiewicz and Lindenmayer 1996]. When dealing with man-made constructions like buildings it is more convenient to design the modeling procedure as a sequence of partitioning steps, gradually refining the model and allowing to structure it. In this context split grammars, which are based on the previously mentioned shape grammars, were introduced and applied on the modeling of buildings. They consist of a large database of production rules using a stochastic process to give the resulting buildings a certain randomness, hence a more realistic appearance [Wonka et al. 2003]. A further development in that direction is CGA shape which was used in generating large scale models of cities. It showed in a very impressive way, that procedural modeling can also be a proper approach for mass modeling [Müller et al. 2006]. Recent works address usability improvement of CGA shape. As before manual editing of rules in text form was required, an interactive editor has been developed, enabling easy creation of models without having to cope with the grammar syntax [Lipp 2007; Lipp et al. 2008]. Furthermore, there are already techniques in development allowing to introduce physical constraints into models (especially buildings) generated by procedural modeling. Based upon grammars that admit the parametrization of production rules a statics analyzer calculates these parameters leading to an appropriate shape, satisfying the desired architectural style, as well as being structurally sound, which leads to even better results regarding realistically sound models [Whiting et al. 2009].

The most recent development is motivated by the desire for dynamic human and humanoid models. Further extending the promising CGA shape grammar our group at the Vienna University of Technology is currently working on procedural skeletons. In this approach a kinematic skeletal system is created during the generation of the model through the production rules. The use of a skeleton in order to describe poses and movement (i.e. being able to animate the models) is a common approach for complex models [Maestri 1999]. The approach of kinematic skeletons was first introduced as a multi-layered hierarchy of mesh-deformation [Chadwick et al. 1989]. Like it has been proposed before [Bloomenthal and Lim 1999] our implementation follows the idea of automatically deriving the skeleton from the model [Ilcik et al. 2010]. There is, as well, a first report on the implementation [Fiedler 2009].

2.2 Mesh-Subdivision

The beginnings of mesh-subdivision can be found in the two classical algorithms that are still in use today [Catmull and Clark 1978; Doo and Sabin 1978]. Both of them are applicable to arbitrary meshes, i.e. the faces of the model do not have any restrictions regarding their valence. A more restricted approach was presented later [Loop 1987], where only triangular faces are allowed in the mesh. Among the many other approaches that have been developed in the following, there is the $\sqrt{3}$ - Subdivision [Kobbelt 2000]. Here the topological refinement of the mesh is applied in a more natural way and advances slightly slower during the successive iterations. The paper furthermore describes adaptive refinement strategies, allowing to define different refinement levels for different parts of the mesh, as well as a way to define mesh boundaries, resulting in smooth boundary curves. A newer interesting approach to subdivision was described more recently [Velho 2003]. The author's idea is the description of mesh-subdivision as a graph grammar formalism, where the production rules describe the geometrical and topological transformations applied to the mesh.

3 CGA shape

In the Introduction we mentioned conventional modeling techniques. By this term we refer to manual modeling, the models are generally created by means of 3D modeling suites like 3D Studio Max, Maya or Blender. Before being able to use these tools it usually requires a long time of orientation and practice. Only skilled artists are able to create sophisticated results. Mastering the design of human models is even more challenging [Ratner 2003]. In the following we discuss some concrete issues of conventional modeling and what solutions CGA shape offers so far to overcome them.

3.1 Why procedural modeling?

- Conventional modeling is time consuming:
  Let us begin with the most basic and obvious drawback. As has been mentioned before, the manual modeling of realistic models is extremely time-consuming. The greatest relief CGA shape can offer in modeling is the automation of the process.
- Repetition:
  When designing a model that contains the same part several times, it is necessary to perform a lot of redundant work. Consider the facade of a building where the same type of window has to be placed across it with a fixed spacing between each window. It can be quite tedious to take care for the exact placement manually. A more convenient and elegant approach is the definition of some production rules, that generate the desired spatial distribution of objects.
- Post-editing:
  If the artist, after finishing the model, wants to change something basic e.g. the size or the shape, a redistribution of all the model's components is necessary. Using a grammar, the approach would be much easier: A basic change is always easily achievable, since the production rules will then generate and distribute the model's component based on its new shape.

After having thoroughly discussed the benefits of CGA shape, we now explain its basic concepts.

3.2 CGA shape basics

As mentioned CGA shape is in general a split grammar. It works in a sequential order because this allows better coping with structuring of the model. Nevertheless the notation of the general production rules is similar to the one in L-systems.

Definition. A grammar $G = (V, \Sigma, P, A)$ consists of the set of nonterminal shapes V, the set of terminal shapes Σ, an initial (set of) shape(s) $A \subseteq (V \cup \Sigma)^+$, called axiom, and a set of production rules P. $V \cup \Sigma = U$, the vocabulary of the grammar.

Additionally, when talking about CGA shape, we need to clarify the key elements this specific type of grammar is based on:

Shape
First of all there is the notion of shape. A shape is the equivalent to a nonterminal or terminal in an ordinary grammar. It consists of a symbol, as well as certain geometric and numeric attributes. The symbol serves as an identifier for the shape, it is either element of the nonterminal or the terminal symbols. The geometric attributes describe the spatial position, the orientation (a coordinate system composed by three orthogonal vectors) and the size (vector) of the shape, defining an oriented bounding box, called scope. Finally the numeric attributes are used for parametrization of rules allowing a more specific derivation process. Shapes can have three or less dimensions.

Production Rules

Definition. A production rule is denoted in the following way:

$$id : predecessor : cond; \rightsquigarrow successor : prob$$

where *id* is a unique identifier for the production rule, *predecessor* is a symbol $v \in V$, *successor* one or more symbols $\in (V \cup Sigma)^+$, *cond* specifying a condition necessary for this rule to fire and *prob* the probability for this rule to be selected.

This means that the application of a production rule p replaces the shape with symbol v given in the predecessor part of the rule by the shapes with symbols $u_1, ..., u_n$, given in the successor part.

For instance

$$1 : facade(width) : width > 9; \\ \rightsquigarrow wall(h/3)window(h/3)wall(h/3) : 0.8$$

denotes the rule with identifier 1, which describes the following: The shape $facade$ is replaced with the three shapes *wall*, *window*

and *wall*, if its parameter *width* is greater than 9. The rule however will only be selected by a chance of 80%. Otherwise some other defined rule will be chosen.

Derivation
The derivation of a model from an axiom is a simple iterative process. For an arbitrary configuration, i.e. a finite set of basic shapes the procedure works as follows:

Algorithm 1 Derivation Process

While the configuration contains nonterminal symbols:

(1) Choose a shape with symbol S in the set.

(2) Select all production rules that are applicable, i.e. where S is on the left hand side and the condition, if existent, evaluates to true.

(3) Pick one of these rules at random (considering their probabilities) and fire it: Replace The shape with symbol S by the shapes whose symbols are listed on the right hand side of the rule. Then continue with step (1).

Special Rules
Along with the previously mentioned basic rule syntax of CGA shape, there are also some special rules to specify the successor shapes. The rule

$$1 : floor; \rightsquigarrow Repeat(X,2)\{B\}$$

for example can be applied on the floor shape and replaces it with the result of a repeat rule, with the specified parameters.

In general CGA shape consists of five such rules: First there are the scope rules, which allow us to change the scope and the contained geometry by means of translation, rotation or scaling. The most significant rule is the split rule. It splits the shape's scope along one or more of its axes, creating two or more shapes of the same dimensionality. Similarly the repeat rule is used to tile a shape with another one. The original shape will be replaced by as many shapes of the successor shape as will fit in its place. Last but not least there are two antagonistic rules, the component split rule and the extrusion rule[1]. These two allow a lowering or an elevation of dimensionality, respectively. The following figure shows the relationship between these rules.

shape (dimension)	component split results	extrusion results
polyhedron (3D)	faces, edges, vertices	-
face (2D)	edges, vertices	polyhedron
edge (1D)	vertices	face
vertex (0D)	-	edge

In addition to these rules, the original CGA shape grammar further includes two more rules, namely occlusion rule and snap rule. The former one is used to check for intersections between shapes, the latter one to align faces and edges of shapes to common lines or planes.

Figure 3 shows a simple example grammar with 1 nonterminal, 2 terminals and 2 production rules.

[1] in the original grammar expressed as scaling along a scope axis

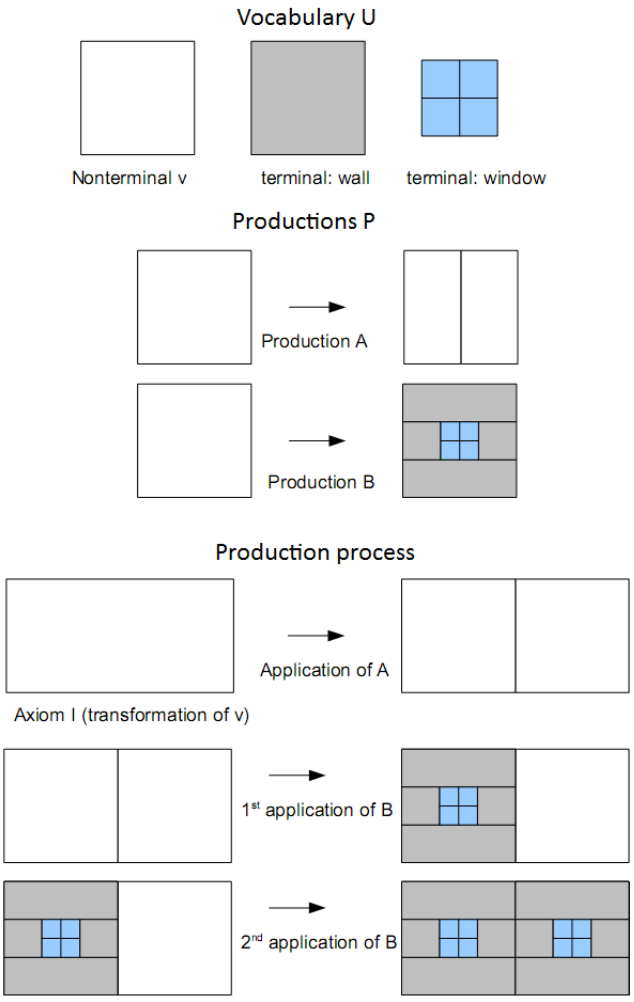


Figure 3: Simple infinite grammar consisting of 1 nonterminal, 2 terminals and 2 production rules. This is one possible production process this grammar allows, applying rule *A* once. On each resulting shape rule *B* is then applied. As there are only terminal shapes left the production process is finished here. *Source: [Lipp 2007]*

3.3 Kinematic extensions

We now describe, how our CGA shape grammar is extended by a skeletal system allowing us to create dynamic models. First of all we have to introduce a new key element.

Joints and Kinematic Sections
Every basic skeletal system consists of rigid bones, which define the particular moveable parts of the model, and rotational joints which link these parts to each other [Maestri 1999]. Just as well these are the elemental parts of our procedural skeletons. In our case joints and bones are represented by the notion of kinematic section. Each scope is now part of such a section. Furthermore the section stores the joint position and limitations, the current kinematic translation and rotation, as well as the links to the kinematic parent and children sections (see Figure 4 for an illustration).

We add some new kinematic rules to our existing set of production rules in order to define pose synthesis separately from model

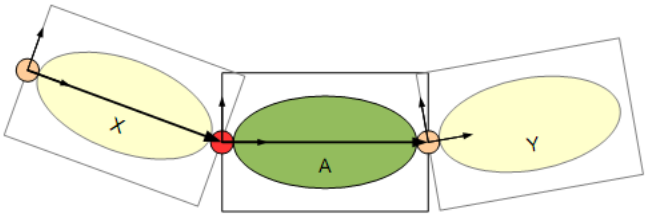


Figure 4: An elliptic shape, located in its rectangular bounding box representing its scope. This scope is bound to a kinematic section whose joint is shown as a circle. The section has a parent on the left and a child on the right. The sections are connected by their joints, represented by thick black arrows (i.e. the rigid bones of the skeleton). *Source: [Ilcik et al. 2010]*

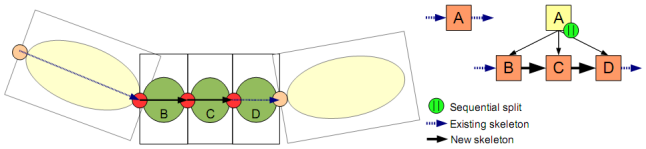


Figure 5: *Source: A sequential split is applied on the second shape of Figure 4. [Ilcik et al. 2010]*

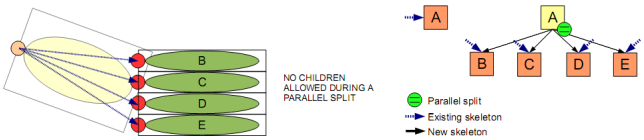


Figure 6: A parallel split applied on the second shape of Figure 4. Note that here the child of this shape has been removed in advance, otherwise the parallel split would have been undefined. *Source: [Ilcik et al. 2010]*

generation. Since the bones are rigid, the new kinematic rules have to operate on the joints. To begin with, there is the joint placement rule that allows us to position the joint of a kinematic section at an arbitrary position relative to the connected shape geometry. This sets the boundaries, within which the joint is able to rotate. Additionally there are limitation and transformation rules, controlling the actual rotation of the joints with respect to the defined movement limits.

In order to construct a skeleton hierarchy during modeling process, we also need to add some modifications to the original CGA shape rules defined in section 3.2. The joints that hold the individual kinematic sections of our model together have to be placed at some initial position during model generation. Thus we have to address the question how this can be achieved in a way, the resulting skeletal system is valid. A valid skeleton has the following properties:

- (i) Each kinematic section belongs to a leaf node of the parse tree
- (ii) Existence of a unique skeleton root
- (iii) Each kinematic section has exactly one parent section
- (iv) Kinematic independence of parse sub-trees [Ilcik et al. 2010], i.e. the outgoing kinematic links of a parse node, remain the same after the derivation.

Changes to the skeletal structure of the model can only occur if new shapes are added to it. This obviously excludes the transformation rules, which thus remain unchanged.
The split rules, however, have to be adapted since they introduce new shapes that have to be integrated in the skeletal hierarchy.

We first deal with the basic split rule. Since splits can basically occur in two different ways, we have to do a case distinction yielding two new sub-rules.
First the split can create a sequential structure, by connecting pairs of adjacent shapes with joints. In this case the solution is simple, since the original shape O is just replaced by the new sequence of shapes $N_1, ..., N_n$ (see Figure 5). One shape of this sequence is now placed in the same section as O has been before, carrying over its parent and another shape becomes the new parent of the original children. Between these two the other shapes are connected in a sequence. Although there is, in general, no restriction which shape becomes the first and which one the last of this sequence, it is the most common approach to just organize them in the split direction, i.e. N_1 to N_n.
The second case is a parallel split or fork. Here the situation is more complicated. Before a parallel split can be applied, it has to be assured, that the respective shape has a kinematic parent. If this was not the case, the split would produce a number of shapes, all of them constituting a root of the skeleton hierarchy. This would, in turn, violate property (ii), we postulated before. As a further restriction the respective shape may not have any kinematic children, to prevent the children from having multiple parents and thus violating property (iii). We further avoid any assignment confusions, where a decision would have to be made, which child becomes linked to which parent. As a consequence of these constraints all the new shapes $N_1, ..., N_n$ simply become direct children of the former parent of the original shape O.
This is shown in Figure 6.

The repeat split rule does not need any special consideration since it can be reduced to a special case of the split rule. Therefore the very same applies.

When a component split is performed, the original shape has to be copied to the result as well. This shape serves as a parent for all lower dimensional components.

Another aspect, we have to take care for, is the ε symbol. As in common grammars ε represents the empty word, in the case of shape grammars it symbolizes an empty geometry. In order to preserve the consistency of the skeleton we cannot just destroy every shape that is substituted by ε. In case the shape does not have any kinematic children, it can in fact be destroyed. Otherwise it has to remain in the configuration to avoid broken links in the skeletal hierarchy.

Connectivity
One last important point we have to take care for in the context of kinematics is the notion of connectivity. In regard to drawing and filling algorithms it has to be clear which parts are connected in order to clarify what is inside and what is outside the model. When dealing with a model that is based on an underlying skeleton, we need to think about the connectivity between the separate skeleton bones. So in our case the problem is, that we have to define between which kinematic sections there is or there is not a connectivity relation.
Let us look at a sequentially split set of shapes, for example. Most easily one could say that each shape has its kinematic parent and children, so naturally these shapes can be defined as connected to each other. In most of the cases this may be a correct and sufficient approach. However, this is not always what we want to have.

Consider the situation where at some point during the derivation process the model should be split into two or more separate independent parts. Imagine the model shown in Figure 2 is to be created starting from a single cubic shape. It will be necessary to split this shape in order to obtain the separate buildings which are naturally not connected to each other. The next problem making connectivity considerations necessary is the possibility to translate and rotate each shape. After splitting a shape, we of course know which parts of the shapes (more precisely, which vertices) are supposed to be connected to each other. But as soon as we apply transformations to the shapes this is no longer self-evident. We therefore extend our shape data structure by a connectivity map which maps the vertices of a shape to the vertices of its parent shape, specifying the connectivity between these vertices. Every vertex that is not mapped to a vertex of the parent shape is thus not connected.

Summarizing our work so far we have a highly-expressive procedural modeling approach at hand. It is possible to create a vast variety of models using a relatively easy grammar system. We have further shown how this system can be extended to create dynamic models, by deriving the skeletal hierarchy during model generation, storing the resulting skeleton in the leaf nodes of the derivation tree.
We now present our approach to enhance the optical quality of the created models by refinement through mesh-subdivision.

4 Mesh-Subdivision

We first want to discuss the theory behind the smoothing of the shapes we generate with the grammar. Since we have to implement the refinement of the shapes by a new production rule in order to integrate it into the shape grammar, we need a general procedure that allows smoothing of an arbitrary shape.
This is where mesh-subdivision comes into play. Concisely mesh-subdivision is an approximation method. Starting from a relatively coarse mesh, it gradually rises the number of its faces, successively refining it in order to get a more and more precise approximation of a smooth surface. What makes it suitable for our purposes is that this happens according to certain rules which normally do not affect the geometry as a whole. Usually there is a scheme describing subdivision rules for each triangle or quadriliteral (or set of triangles/quadriliterals) of the shape. This makes it possible to integrate the method as a production rule which can be applied to an arbitrary shape. It is not necessary that any additional information about the shape's topology has to be provided.

We begin with a short introduction to subdivision schemes in order to demonstrate how subdivision in general works. While there are many subdivision schemes, we have decided to use the $\sqrt{3}$ scheme for our approach, which will be described in the course of this introduction. Note, however, that we can only provide a very rough and simplified overview of it. A more sophisticated approach explaining the mathematical backgrounds and explaining the details and extensions (e.g. adaptive subdivision, special considerations for boundary polygons) would go beyond the scope of this report.
After this introduction we describe our implementation of a mesh-subdivision approach using this scheme. We provide a description of our approach, as well as an explanation of the three major issues that occurred and how they have been dealt with.

4.1 Subdivision schemes

As mentioned before, there are many different subdivision schemes. There are, however, certain features that are common to most of these schemes. Roughly spoken there are three general steps a scheme can be broken down to.

1. Generation of new vertices:
   According to certain criteria, for each face and/or edge of a mesh a new vertex is inserted at a specified spatial position.
2. Shifting of vertices:
   In order to create a smoother surface the existing vertices of the mesh are moved to a new spatial position.
3. Connecting the vertices / flipping of existing edges:
   Each scheme defines rules, how the newly inserted vertices are to be connected to each other and moreover, how they should be connected to the original vertices. Other additional measures define the flipping of existing edges, such that they connect to different vertices from the ones they have connected before.

In the following subsection we describe the $\sqrt{3}$ scheme, showing how these three steps look like in an actual subdivision scheme. Furthermore it serves as a rough technical overview over the approach our implementation is based on.

4.1.1 $\sqrt{3}$ Scheme

The $\sqrt{3}$ subdivision is based on triangular meshes. [Kobbelt 2000]. This means that the input meshes may only be composed of triangular faces. The output is of course a triangular mesh, too. Compared to the traditional dyadic splits (every original edge is split into two sections, thus every triangle is split into four new triangles) as it can be seen for example in the Loop subdivision scheme [Loop 1987], here the basic idea was to generalize this approach to n-adic splits. In every subdivision step a 1-to-3 split is performed for each triangle by inserting a vertex in the center. Connecting the original vertices of the triangle with this new vertex then yields three new triangles replacing the original one. In the end every original edge, that connects two old vertices is flipped in order to re-balance the valence of the vertices. This procedure is depicted in Figure 7.
Applying this scheme twice yields a 1-to-9 refinement of the mesh, which is, in fact, a triadic split. In a way one single iteration can then be considered the square root of a triadic split, which gives this scheme its name.

The simplified algorithm for an arbitrary triangular mesh M is shown in Algorithm 2.

Note that the $\sqrt{3}$ subdivision describes a slower refinement process than other conventional subdivision schemes. While there each iteration in general creates four new quadriliterals or triangles (in the case of a dyadic split, here we obtain only three new faces with every step.

4.2 Implemented Approach

As previously mentioned there are many other different subdivision schemes that can be applied to an arbitrary mesh and generate an appropriate output. There are basically no additional shape-specific parameters needed, since the subdivision always has the same goal: The smoothing of the shape.
For our implementation the $\sqrt{3}$ scheme is chosen out of several reasons:

- Since the boundary representation of our shapes consists of triangles it is more appropriate to directly work on triangles instead of making a detour to quadrilateral faces which have to be triangularized afterward.

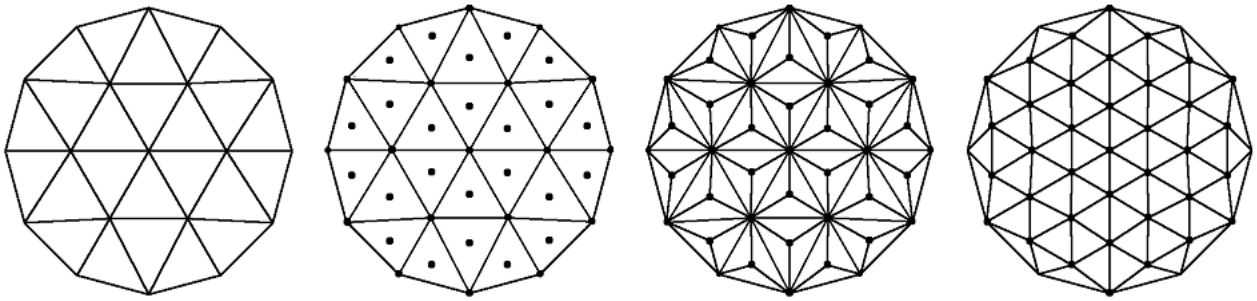


Figure 7: Step-by-step application of the $\sqrt{3}$ scheme. First a new vertex is inserted at the centroid of each face. Then the centroids are connected to the original vertices of the face. Finally the edges between the original vertices are flipped. *Source: [Kobbelt 2000]*

Algorithm 2 $\sqrt{3}$ Scheme

(1) Face points: for each face $F \in M$: Calculate the centroid of F and add a new vertex there.

(2) Vertex shift: for each original vertex $V \in M$: Let n be the valence of V. Let V_0,..., V_{n-1} be the directly adjacent vertices. Then the new position of the vertex V is given by

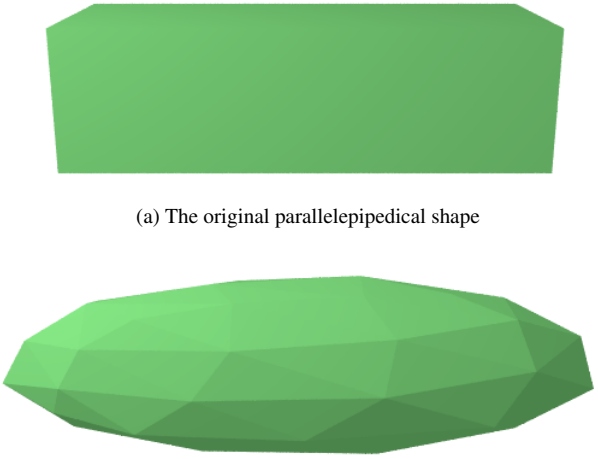
$$(1-\alpha_n)\cdot V + \alpha_n \frac{1}{n}\sum_{i=0}^{n-1} V_i \tag{4.1}$$

where

$$a_n = \frac{4-2\cdot\cos\frac{2\pi}{n}}{9} \tag{4.2}$$

(3) Connect each original vertex to the face points of the adjacent faces by adding a new edge between these.

(4) For each edge $E \in M$ that connects two original vertices: Flip the edge such that it connects the two newly inserted face points of the faces that were separated by it.



(a) The original parallelepipedical shape

(b) The same shape after the subdivision rule (2 iterations) has been applied to it.

Figure 8

- The subdivision of triangular faces at their center is a more natural way than splitting all three edges [Kobbelt 2000].
- As the $\sqrt{3}$ subdivision refines the input shape more slowly than the other algorithms it is easier to control the level of refinement. This allows us to only refine the mesh as far as necessary and thus avoid unnecessary calculations.
- It is one of the algorithms that is already implemented in CGAL, the Computational Geometric Algorithms Library (www.cgal.org), which is used by our shape grammar implementation.

Based on this subdivision algorithm we can now design a new production rule implementing mesh subdivision.

$$id : S; \rightsquigarrow Subdivide(iterations)\{S\}$$

Our first definition of this rule takes the number of iterations for the subdivision as the only parameter. When applied on a shape S the $\sqrt{3}$ subdivision, as described in Algorithm 2, is applied to it. The stored geometry for S is then replaced by the new smoothed one.
If we consider single shapes this first approach already yields the desired result (see Figure 8a and Figure 8b. However, there are certain further problems and special cases that have to be addressed.

Recall again the nature of production rules in a shape grammar. Each rule is supposed to be applied to a specific shape. There are no global effects, in particular no other shapes than the target shape of the rule are to be manipulated.
With subdivision, however, we have reached a point where it is difficult to preserve this basic property of production rules. In the following we present the main concerns that had to be dealt with in the course of the implementation. Basically, all of the first three issues are in some way related to the connectivity between the shapes.
The latter aspects then deal with more elaborated, wider modifications of the grammar. At this point we cross the boundaries of our implementation, since these are to a great part merely ideas that have come up in the course of our work, although parts of it have already found their way into the current implementation. During the planning of our work different possibilities for improvement have come up and we will describe the most promising ones, that form the upcoming features of our grammar and will be implemented in the future.

4.2.1 Connectivity Preservation

The first flaw of our new rule becomes evident, when we have several shapes that are connected to each other. At the moment the only possibility for smoothing them is to apply the subdivision rule

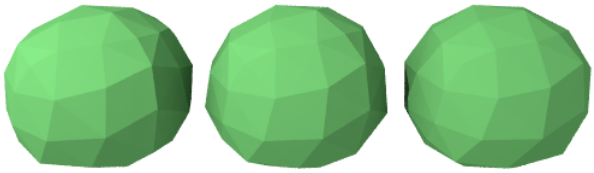


Figure 9: Three shapes, that have been subdivided separately. There is no visible connectivity between them.

on each of them. This way each shape is separately smoothed, yet there is no connectivity between the new smoothed geometry of these shapes. But consider connectivity is defined between these shapes. Then such a result is clearly not what we want. In this case it would be helpful to look at all the shapes as a whole and perform the subdivision on the union as if it was a single shape. The desired result, under assumption that the three shapes are connected, is the same as it would have been, if the subdivision had been applied to a single shape of the same size and form.
In other words, the application of the split rule and the subdivision rule on each of the resulting shapes should yield the same result as if the subdivision rule had been applied first before splitting its result into a set of new shapes.
Figure 9 shows the current result for three shapes after application of the subdivision rule on each of them. However, the correct result for three connected shape would look as shown in Figure 8b, where each of the shapes is assigned its respective part of the whole shape.

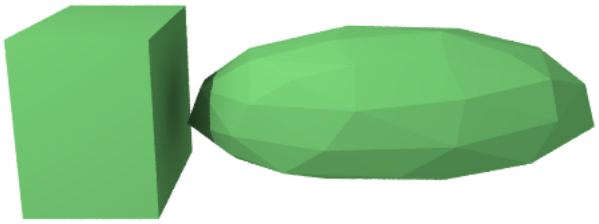
For this reason we have to introduce a pseudo-global effect to our new rule. Whenever a the subdivision rule is applied to a shape S, we have to browse through the kinematic ancestors and children of S. All shapes that have been previously subdivided have to be joined to S, such that the whole union becomes the target of the subdivision algorithm.

Smoothing Group
We therefore introduce the notion of a smoothing group, containing the mentioned set of shapes. This concept serves as a first approach to the problem and we will see how it can be further improved in section 4.3. The smoothing group G_S of a shape S can be easily obtained in an inductive way.

1. $S \in G_S$.
2. If a shape P is the kinematic parent of a shape $\phi \in G_S$ and P is connected to ϕ, then $P \in G_S$.
3. If a shape C is a kinematic child of a shape $\phi \in G_S$ and C is connected to ϕ, then $C \in G_S$.

After application of the subdivision, each of the shapes that were part of the union is then assigned its respective slice of the subdivision. To achieve this, after the subdivision is applied, for each shape of the smoothing group the geometric difference between the whole result and the current (pre-subdivision) geometry of all other shape is calculated. This way each section containing a shape of the union, is assigned a corresponding slice of the result after the subdivision.

Whenever the subdivision rule is applied to a shape, we have to apply the subdivision to its smoothing group as a whole in order to preserve the connectivity between the shapes.



(a) Subdivision result without connectivity to unsmoothed shape

(b) Subdivision result with connectivity to unsmoothed shape.

Figure 10: Comparison of subdivision results regarding connectivity of boundary shape

4.2.2 Boundary Shapes

This leads to the next issue we have to address. If we do not want to apply the subdivision to the complete shape configuration, there obviously has to be at least one shape that is at the boundary to another shape which has not been smoothed. This fact by itself would not cause any trouble, since the smoothing group joins only the necessary shapes not including the non-smoothed ones.
The problem that occurs now is again an issue of connectivity. There is no visible connection between the smoothed shapes and the first shape that is not in the smoothing group, even though a connection between these shapes may have been defined. Similarly to the former issue we need to add further modifications in order to make the connectivity between these shapes graphically visible.

To solve this problem the subdivision algorithm is modified, such that it is possible to define a set of faces (for the 3-dimensional case) or a set of vertices (for the 2-dimensional case) that should be excluded from subdivision. This allows us to exclude the faces (or vertices, respectively) at the boundary of the smoothing group, that are part of shapes which are connected to shapes outside the smoothing group.

In order to determine which parts of a shape have to be excluded from subdivision we introduce the notion of protected faces and protected vertices. After the collection of all the shapes that are part of the smoothing group, each shape of the union has to be examined for connections to shapes that are not in this union (i.e. that are not part of the same smoothing group). This can be done by simply checking for each shape if it has a parent or a child, that is not in this union. For each shape for which this is the case, the according parts, that form the boundary between this shape and the shape(s) outside the union, have to be protected. Regarding 2-dimensional shapes, the vertices which are connected to vertices of a neighboring shape (i.e. the connectivity relation for them is defined in the connectivity map) are exactly the vertices that can be marked as protected vertices. With 3-dimensional shapes, it is necessary to further check if there are faces of which all composing vertices are protected. If this is the case, these faces are marked as protected.

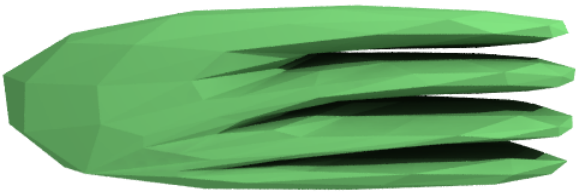


Figure 11: Soft Connectivity: Two shapes. The right shape has been split by a parallel split into four new shapes that are not connected to each other. The whole set of shapes has then been subdivided.

A list containing all the protected faces (or vertices, respectively) is then passed as an additional parameter to the modified subdivision algorithm and will therefore not be subdivided.

To clarify the way the smoothing works now, using the example of 3-dimensional shapes, see Algorithm 3, which describes, how the protected faces are determined. After this is done the resulting list of protected faces is passed to the modified subdivision algorithm. This algorithm then basically works the same way as it did without the modification for protected faces. The only difference is that none of the steps 1 to 4 (see Algorithm 2) is applied to the protected parts of the mesh:

- (1) For protected faces no face point is calculated.
- (2) If a vertex is part of a protected face, it is not shifted.
- (3) Since there are no new face points for protected faces, no new edges have to be created for these faces.
- (4) If one of the faces that is separated by an edge is protected, then this edge is not flipped.

Algorithm 3 Determines the faces, that are to be protected (*protected_faces*) for a given smoothing group S_G

```
protected_vertices = ∅
for all shapes S ∈ S_G do
  if parent P of S with P ∉ S_G then
    for all vertices V ∈ S that are connected to a vertex of P do
      Add V to protected_vertices;
    end for
  end if
  for all children C of S do
    if C ∉ S_G then
      for all vertices V ∈ S that are connected to a vertex of C
      do
        add V to protected_vertices;
      end for
    end if
  end for
end for
for all faces F ∈ S do
  if each vertex of F is contained in protected_vertices then
    Add F to protected_faces;
  end if
end for
```

Now it is possible to properly visualize the connectivity between these shapes. Figure 10a and Figure 10b show the different results with and without connectivity between boundary shape and unsmoothed shape.

4.2.3 Soft Connectivity

Finally there is one significant detail that has to be taken care of. Consider two shapes, A and B, that are connected to each other. A parallel split is applied on shape B, yielding several new shapes, that are all connected to shape A. Now, the whole set of shapes is to be smoothed. The question that arises now is, if the shapes obtained from the parallel split are connected among each other or not. We call this kind of connectivity *soft connectivity*.
According to whether there is soft connectivity between these shapes or not is important for determining the proper subdivision result. If the shapes are connected the result we obtain would look like the configuration shown in Figure 8b. On the other hand, if the shapes are not connected to each other, we would want to see this in the resulting model. For a parallel split that yields four new shapes, the desired result without connectivity is shown in Figure 11.

In order to decide if shapes after a parallel split are soft-connected or not, we have to further modify our shape data structure and the split rule.
When a parallel split is applied on a shape the rule needs a list of boolean values as an additional parameter. This list has to define for each neighboring two shapes if they are soft-connected or not. The shape itself receives an additional field to store this information, such that it can be taken into consideration when the subdivision rule is applied on it.

We implemented a straightforward and easy solution to this issue. The soft-connectivity is only defined for shapes that are part of a parallel split. Each shape that has come into being out of a parallel split is assigned a boolean value defining if it is soft-connected to the next shape of the parallel split or not.
Now, in the course of the application of the subdivision rule on a shape, the only case where an adjustment has to be made is, if a shape is encountered in the smoothing group, that has its soft-connectivity set to false. In this case a very small slice of the shape is cut away. This way the subdivision scheme will also recognize a gap between this shape and its sibling and smoothen the two shapes accordingly.
The only other cases are, that the two siblings are soft-connected or that there has not been a parallel split at all. In both cases, we do not need to introduce any further changes to the algorithm.

4.3 Further ideas

As already mentioned, this last section deals with the further ideas that have been developed in the course of our work. We will give an overview on two important issues that have occurred and how the intended further developments of the grammar can help overcoming them.

The key element on which the following considerations are based on, is a general modification of our grammar. In order to achieve a higher semantic expressiveness, context-sensitivity has been introduced into the grammar.
This was done by modifying the way rules can be applied to a shape. As we have described before, subdivision causes the need for rules that have more than local effects, since neighboring shapes have to be taken into consideration. For this reason a production can now not only modify the shape it is applied on, but also other shapes. This makes it possible to define rules that work differently according to the context in which a shape is embedded.

Another aspect is the modification of the shape data structure, more specifically the symbol that identifies each shape. The symbol is now no longer represented by just a simple string. It is now rather

a structure, that, along with the string identifying the shape, now stores two additional data structures.
The first one is the rules semantics dictionary that maps strings to strings. The first string is used to define a certain semantic attribute that a shape can have. The second one than defines the value this certain attribute has for the respective shape.
The second structure is a list of rules actions. It is used to define methods that should be triggered after a certain event occurs, e.g. the change of some semantics, the application of a certain rule on a shape, etc.

This allows some new powerful possibilities of manipulating the shapes and highly increases the expressiveness of the grammar.

4.3.1 The Semantics of Smoothing Groups

The first issue is related to the concept of smoothing groups, that has been introduced as a remedy to preserve the connectivity of multiple shapes, that are to be smoothed (see Section 4.2.1). However, this concept is still subject to improvements.
As it is, the subdivision algorithm has to be completely evaluated for every shape. Every time the subdivision rule is called for a shape, the whole smoothing group of this shape is affected, involving many join and difference operations in order to assign the correct slice of the subdivision result to each affected shape. The result we obtain this way can of course be discarded, as soon as the subdivision rule is applied on the next connected shape whose smoothing group again includes these shapes. The only result that remains valid is the last one, which is obtained after the subdivision rule is applied on the last shape of this smoothing group.

This can be realized by making use of the rules semantics dictionary. We define a new attribute: *smoothingGroup*. Each shape can now be assigned a specific value for this attribute.
This gives us the possibility to semantically define which shape belongs to which smoothing group. Thus, instead of having to iteratively recalculate the smoothing result in the course of the evaluation of the different shapes, as described above, we can now precisely define which shapes belong together, and should be smoothed as a union.

As there is now only one subdivision operation which is applied to a number of shapes a new problem occurs. We have to determine the correct time when the subdivision should take place for the union of shapes, since we do not know if, when the subdivision is executed, all shapes of the smoothing group are ready to be smoothed. According to the previously described implementation the Subdivide rule is called for every shape separately, which of course can occur at different times during the derivation process.
The shape on which the subdivision is then executed can be any arbitrary shape, since, as stated above, the grammar allows us to manipulate shapes on which a rule is not directly applied on, too, in the course of the evaluation of that rule.

Synchronization of Subdivision
We therefore develop a different approach for evaluating the rules that are applied on the shapes. For this reason an additional evaluation queue is introduced in which shapes are placed, that are *waiting*.
Now, whenever the subdivision rule is to be evaluated we have to perform a ready check. We iterate over all shapes that are part of the same smoothing group and check if they are either terminal shapes or if the evaluation of the subdivision rule has already been tried for them before, moving them to the waiting queue. Only if all shapes of the smoothing groups are waiting or terminals (i.e. all shapes are ready to become smoothed) the subdivision is applied. See Algorithm 4 for a compact description of the algorithm.

Algorithm 4 synchronization check for evaluation

```
Let sg be the smoothing group of the shape s.
for all other shapes s_other in sg do
  if s_other is in waiting queue or s_other is a terminal shape
  then
    continue;
  else
    add s to waiting queue
    return false;
  end if
end for
return true;
```

This way we ensure that the subdivision takes place that the subdivision is performed at the right moment, when all shapes are expecting it. As described before, which shape becomes the actual target of the subdivision does not matter, since the whole smoothing group is affected in any case. After the subdivision is executed, each shape is placed in the normal evaluation queue again (if it is not already a terminal shape, of course), such that further rules can be processed.

4.3.2 Refinement of Subdivision Results

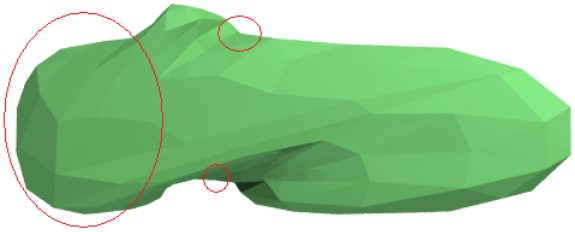
The last issue we want to address here is related to the assignment of the proper slices after the subdivision to the respective shapes. When using the method we have described before in Section 4.2.1 there are certain special cases where these assigned slices are not completely correct. This means that parts of the subdivision result get assigned to a section, that logically are not part of this section. As Figures 12a and 12b show, this issue typically occurs when shapes that are shifted are subdivided together. The subdivision algorithm creates smooth transitions between the shapes, which causes the extent of the resulting model to be greater than the extent of the pre-subdivision model. Clearly our previously described method using the pre-subdivision shapes cannot eliminate these parts.

For this reason it is necessary to implement a post-processing step that is executed after the assignment of the respective slices is done. In this step we try to further subtract these parts that should not be part of the shape.
To achieve this, some simple additional geometric difference operations are sufficient. For each shape, the connecting faces (or edges respectively) with the neighboring shapes are examined. In every case the halfspace that is facing away from the shape is subtracted from the shape's current geometry. This way every part of the geometry that is not directly connected to the *main*-part of the shape will be eliminated from its geometry. The result is that each shape now represents the correct slice of the subdivision.

5 Conclusion

Mesh subdivision is a method that per se can be easily included into a shape grammar. Since the application of a subdivision scheme is so general that no information about the shape itself has to be provided, subdivision is very well-suited to be described by a production rule.
Unfortunately, the main effort behind this integration lies in the detail. We have showed that the connectivity between the shapes is a major issue that has to be taken care of. Therefore a shape has to contain connectivity information with respect to its neighboring

(a) A union of several shapes before smoothing is applied to them.The red circle symbolically indicates the area of the first section.



(b) The same union after it has been smoothed. The red circles indicate the parts that are now forming the shape that is assigned to the first section. Note the two additional small circles, that are also assigned to this section.

Figure 12: Incorrect assignments in a subdivision result.

shapes. Furthermore, once this is assured, the algorithm has to be adapted, such that it takes this information into consideration when being applied to a shape.

Apart from these difficulties there is no reason why subdivision should not be included into procedural modeling, since the smoothing of the models contributes a big part to enhancing the realism of the results.
Of course this is just one possible approach and depending on the shape grammar implementation other subdivision schemes may be more suitable. Depending on the implementation another subdivision scheme may be more suitable, or additional issues may occur. However, this would go beyond the scope of this report, and we think, that we have provided a basic illustration of the approach.

References

BLOOMENTHAL, J., AND LIM, C. 1999. Skeletal methods of shape manipulation. In *SMI '99: Proceedings of the International Conference on Shape Modeling and Applications*, IEEE Computer Society, Washington, DC, USA, 44.

CATMULL, E., AND CLARK, J. 1978. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer Aided Design 16*.

CHADWICK, J. E., HAUMANN, D. R., AND PARENT, R. E. 1989. Layered construction for deformable animated characters. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 243–252.

DOO, D., AND SABIN, M. 1978. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design 10*.

FIEDLER, S. 2009. Procedural human posing using CGA grammars. Tech. rep., Technical University of Vienna.

ILCIK, M., FIEDLER, S., PURGATHOFER, W., AND WIMMER, M. 2010. Procedural skeletons: Kinematic extensions to CGA-shape grammars. In *Proceedings of the Spring Conference on Computer Graphics 2010*, Comenius University, Bratislava, 177–184.

KOBBELT, L. 2000. Sqrt(3)-subdivision. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 103–112.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development: Parts i and ii. *Journal of Theoretical Biology 18*.

LIPP, M., WONKA, P., AND WIMMER, M. 2008. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics 27, No. 3*, 10.

LIPP, M. 2007. *Interactive Computer Generated Architecture*. Master's thesis, Technical University of Vienna.

LOOP, C. T. 1987. *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis, University of Utah.

MAESTRI, G. 1999. *Digital Character Animation 2, Volume 1: Essential Techniques*. New Riders Publishing.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L., 2006. Procedural modeling of buildings.

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

RATNER, P. 2003. *3-D Human Modeling and Animation*. John Wiley & Sons, Inc.

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

VELHO, L. 2003. Stellar subdivision grammars. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 188–199.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, ACM, New York, NY, USA, 1–9.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 669–677.