International Journal of Shape Modeling

MESH-BASED PARAMETRIZED L-SYSTEMS AND GENERALIZED SUBDIVISION FOR GENERATING COMPLEX GEOMETRY

ROBERT F. TOBLER STEFAN MAIERHOFER
VRVis Research Center
*Vienna, Austria**

and

ALEXANDER WILKIE
Institute of Computer Graphics and Algorithms
Vienna University of Technology, Austria†

We propose two mechanisms that can be used to generate complex geometry: *generalized subdivision* and *mesh-based parametrized L-Systems*. Instead of using standard subdivision, which uses the same subdivision rule at each level of the subdivision process, in order to converge to a limit surface, we employ a generalized approach, that allows different subdivision rules at each level. By limiting the variations at each level, it is possible to ensure convergence. Mesh-based parametrized L-Systems represent an extension to L-systems which associates symbols to the faces in a mesh. Thereby complex geometry can be introduced into an existing mesh by using procedural substitution rules. Combining both these mechanisms, a wide variety of complex models can be easily generated from very compact representations.

Keywords: meshes, procedural modelling, fractals, subdivision surfaces.

1. Introduction

Complex objects which exhibit different features at different resolution levels are difficult to model and render with conventional, manual modeling techniques. In order to facilitate such tasks it is necessary to resort to procedural modeling. Currently three different developments concentrate on different aspects of procedural generation of models:

- *Subdivision surfaces*: a sequence of subdivision steps applied to a base mesh generates a series of meshes that converges to a limit surface. The objective of this development is to generate smooth models with G^1 or — even better — G^2 continuity.
- *Fractal surfaces*: certain natural phenomena like terrains or wrinkled tissues can be nicely described by recursively adding random displacements to the

*Donau-City-Str. 1/3, A-1220 Vienna, Austria.
†Favoritenstr. 9/5, A-1040 Vienna, Austria.

vertices in a subdivision scheme. The goal of this development is to generate randomized models based on the variations that appear at different levels of resolution.

- *Parametrized L-systems*: plants and other branching structures are best described by a procedural definition that emulates the growing process of a biological system. Here the goal is to develop rule systems that mimic the behaviour of real plants.

In order to generate models of plants, terrains, and other natural phenomena that are convincing at all different scales, we introduce a combination of these three developments which makes it possible to choose which of them should be used at each level of resolution. Since the whole description of such multi-resolution models is procedural, their representation of such models is very compact and can be exploited by level-of-detail renderers that only generate surface detail where it is visible.

After giving an overview of the previous work in section , we will introduce a generalized view of the subdivision process that includes fractal surfaces in section , a new application for L-Systems in section , and a combination of these two techniques in section . Section details the rendering techniques that were used, and Section contains some results of this combination.

2. Previous Work

Mesh subdivision is a technique for generating smooth surfaces that has been introduced quite some time ago by Catmull and Clark [3] and Doo and Sabin [9]. For a long time the theoretical foundation of the subdivision process was not as thorough as for other modeling techniques such as BSplines and the more general NURBS, and thus it took a while for subdivision methods to become widely known and used. Recently this has been rectified by the introduction of methods to analyse and evaluate subdivision surfaces at any point [22] [24], a method for extending subdivision surfaces for emulating NURBS [23], the addition of normal control to subdivision surfaces [1], and a method to closely approximate Catmull-Clark subdivision surfaces using BSpline patches [19]. A number of other extensions to subdivision surfaces [7], [14], [15] have established them as the modeling tool of choice for generating topologically complex, smooth surfaces.

While the main goal of most subdivision surface techniques is the use of recursive refinement to obtain smooth surfaces, the field of procedural modeling uses various similar principles to add detail to surfaces at different levels of resolution. One example for such a procedural modeling strategy is the generation of fractal surfaces by adding random variations at each level of recursive refinement [10], [25]. These surfaces have been demonstrated to be very useful for modeling natural phenomena like terrains [18] and other complex geometry.

In order to generate more complex objects with branching structures, like trees

and other plants, another kind of procedural modeling strategy had to be used. Lindenmeyer introduced string-rewriting systems [16], later called L-systems, that are useful for describing biological processes. In order to generate realistically looking plants Prusinkiewicz et al. [21] associated parametrized geometry to the symbols in these L-systems.

By extending this to so-called open PL-systems [17] that interact with an environment, Měch and Prusinkiewicz were able to simulate the appearance of various plants and their reactions to environmental influences. A set of rules that can be used to generate models of a number of different tree species were presented by Weber and Penn [26]. Efficient methods for ray-tracing such systems were introduced by Gervautz and Traxler [11], efficient ray-tracing of complex scenes was demonstrated by Pharr et al. [20].

3. Generalized subdivision

The standard subdivision process starts out with a mesh $M^{(0)}$ composed of vertices, edges, and faces that is the base for a sequence of refined meshes $M^{(0)}$, $M^{(1)}$, $M^{(2)}$, ... which converges to a limit surface, called the subdivision surface.

The process for generating submesh $M^{(n+1)}$ of a specific mesh $M^{(n)}$ in the sequence can be split up into two operations. The first operation, which we will call *mesh refinement*, is the logical introduction of all the vertices in the submesh. This operation yields all the connectivity information for the vertices of the submesh without specifying the positions of these newly introduced vertices. The second operation, which we will call *vertex placement*, is the calculation of the actual vertex positions. Standard subdivision schemes use specific rules for generating the new vertex positions, that ensure that the limit surface of the subdivision process satisfies certain continuity constraints, e.g. G^1 or G^2 continuity.

In order to break these continuity constraints at user specified locations, different rules for vertex placement have been introduced [7], that maintain discontinuities at user specified edges. These rules fix the location of edge vertices in place for a user-specified number of subdivision steps. Thus this number can be viewed as a measure of edge-sharpness.

From a more general viewpoint, fractal surfaces [10] can be viewed as a type of subdivision surface where the vertex placement rules at each subdivision step have been chosen to maintain only G^0 continuity.

In order to obtain maximum flexibility in generating subdivision surfaces, we propose to separate the two operations of mesh refinement and vertex placement, and make it possible for the user to independently specify both of these operations.

3.1. *Mesh refinement*

As the *mesh refinement operation* generates the connectivity information for the submesh, it determines if the subdivision process generates quadrilateral meshes, such as Catmull-Clark subdivision, or triangular meshes such as Doo-Sabin or $\sqrt{3}$-

Subdivison [14]. In order to demonstrate the viability of our new approach, we implemented mesh refinement based on Catmull-Clark subdivision (see figure 1).

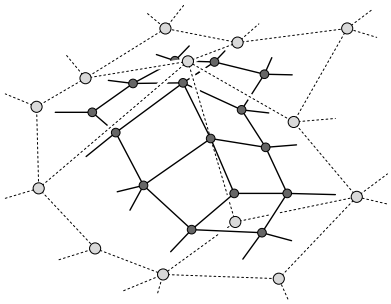

Fig. 1. A subdivision step in the Catmull-Clark subdivision scheme.

3.2. *Vertex placement*

Standard *vertex placement rules* consist of taking weighted averages of the vertex positions of mesh $M^{(n)}$ in order to calculate the vertex positions of mesh $M^{(n+1)}$. For standard subdivision surfaces these rules have been designed to smooth the cusps and edges of the input mesh $M^{(0)}$. Although this is desirable in a number of situations, we want to add more flexibility in the rules for vertex placement.

In order to introduce variations at any point in the subdivision process, we introduce two geometric properties that can be used to specify a vertex placement rule at each subdivision level. The first of these properties, the *local normal vector*, is an approximation of the surface normal of mesh $M^{(n+1)}$ at a given vertex. Although there are multiple methods for estimating this local normal vector, for simplicity we used the weighted average of the normals of all faces meeting at the vertex under consideration. The second property is the *local scale factor* of the surface, a scalar indicating the average face size at each vertex of a mesh in the sequence. Again this can be estimated with various methods. We chose the average diagonal length of all faces meeting at the vertex as a measure that can be easily computed. Both these parameters are provided in order to facilitate multi-resolution specification of these displacements.

By using these two properties, it is possible to specify a vertex placement rule, by an equation similar to a procedural texturing rule. Instead of a colour at each position in space, we generate a displacement vector for each vertex in a mesh. If these displacement vectors are chosen to be colinear with the local normal vectors at each vertex position, the resulting vertex placement rule can be viewed as a generalized form of displacement mapping [5], [6].

As an example (see figure 2), if random displacements in the direction of the local normal vector are added to the vertices of a surface, and the size of the displacements is proportional to the local scale factor, the resulting surface will be a fractal with the standard $1/f$ frequency characteristic. This example however,

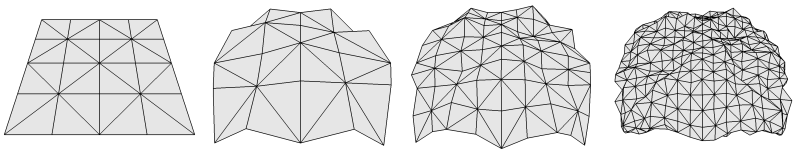

Fig. 2. A few subdivision steps using a fractal displacement rule with $1/f$ characteristic.

uses the same rule at each level of the subdivision process.

3.3. *Alternating between different vertex placement rules*

By specifying different vertex placement rules at different resolution levels, it now becomes possible to model a desired surface in a true multi-resolution fashion. At each scale of the model different variations can be introduced in order to approximate the desired result. This is similar to normal meshes [12] and displaced subdivision surfaces [15], but our scheme is a generalization as there is no limitation on the type of rules that can be used at each level of subdivision. A drawback is, that we cannot automatically generate the rules at each level to approximate a given shape.

The two properties of the local normal vector and local scale factor are provided, in order to facilitate simple and easily specifyable changes. It is also possible to add variations to the vertex placement rules, without regard to these properties. Using the fractal surface as an example again, instead of moving the vertices in the direction of the local normal vector, all vertex movements could be performed into the same global direction. In this way it is possible to generate a fractal height field.

The process so far makes it possible to modify the vertex positions at each level of subdivision either in a globally chosen direction, or locally in the direction of an estimated normal vector. Sometimes it may be necessary to change the position of a vertex locally not only in the direction of the normal vector, but with respect to a local coordinate frame. For this purpose, a similar concept to frame-mapping [13] can be employed (see figure 3).

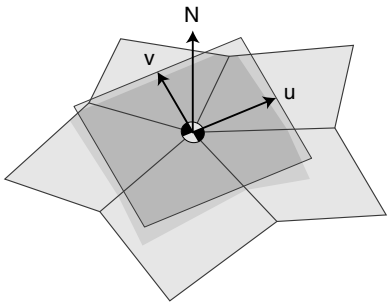

Fig. 3. Local coordinate frame at a vertex.

This allows the modification of the vertex position at each subdivision level, both in the direction of the local normal vector, and along the local tangent plane.

As long as the modified subdivision rules are only used a finite number of times, with the rest of the rules being applications of the standard smoothing rules, the algorithms for evaluation of subdivision surfaces at any point [24] can still be used. An example for such a model is the chair in figure 12: at a certain subdivision level random vertex displacements were added to simulate the folds in the cushion, but it is still possible to calculate the exact limit surface, as all subsequent subdivision steps are just standard Catmull-Clark steps.

If the introduced vertex displacements are always bounded by the local scale factor, the resulting surface is a fractal surface which can be approximated by terminating the subdivision process after a finite number of steps (for smooth surfaces, additional constraints have to be met [4]). The resulting error in vertex positions is on the order of the local scale factor. In this case the resulting surface is only G^0 continuous, and there is no good way of approximating the normal vector of the surface. Such surfaces are however still valuable modeling primitives, as there are a number of natural phenomena, e.g. terrains and wrinkled tissues, which can be approximated by such $1/f$ fractal surfaces.

4. Rule based mesh growing

Although the generalized subdivision method introduced in the previous section is a very powerful modeling tool, the resulting meshes will always have the same mesh-connectivity except at a small number of extraordinary vertices that were present in the original mesh. As an example, if Catmull-Clark subdivision is used, each mesh in the subdivision sequence will always be composed of quadrilaterals. Introducing local variations can then only be performed by locally expanding the subdivision mesh. This can lead to arbitrarily large distortions in the quadrilaterals.

If some vertices are shifted by large vectors (considerably longer than the local scale factor), and all other vertices are left unchanged, the four quadrilaterals meeting in that one vertex will be severely distorted (see figure 4). The resulting texture coordinate space of the affected vertices is severely stretched, which can lead to problems in such applications as texture mapping and finite-element methods like radiosity. In order to overcome this deficiency, we will introduce rule based mesh growing.

4.1. *Parametrized L-systems*

L-systems [16] are defined by a number of symbols that represent components of a plant, and a set of rules giving a string of replacement symbols for each of the available symbols. In order to simulate a biological system, a start symbol is taken, and in each replacement step, all symbols are transformed according to the given rules. Thus the start symbol can be thought of as a seed, and the ruleset encodes the growth of the plant.

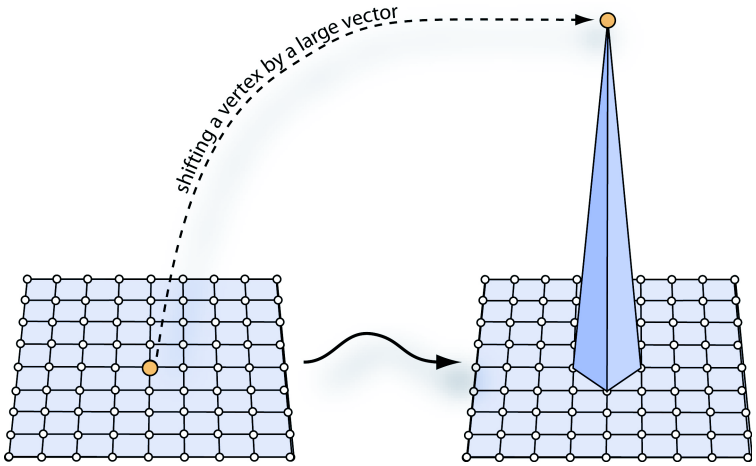

Fig. 4. Shifting a vertex by a large vector gives rise to severely distorted faces.

In order to generate three-dimensional models, L-systems have been extended by three significant concepts [21]:

- *Parametrized symbols*: for placing the parts of a plant in space and generating parts of different sizes, it is necessary to associate parameters with each symbol, that encode the properties of each part of a plant.

- *Parametrized rule expansion*: in order to modify the parameters, it is necessary to calculate new values for the parameters at each rule expansion step. These calculations are associated with each expansion rule.

- *Encoding of a hierarchical structure*: L-systems only operate on one-dimensional strings. In order to represent hierarchical structures, such as trees, it is necessary to introduce grouping symbols. With these symbols it is possible to encode branching structures.

4.2. *Mesh-based PL-systems*

In order to use parametrized L-systems in the context of a mesh-based modeling system, we introduce *mesh-based PL-systems* by associating each parametrized symbol of the system with a face in a mesh. Thus the right-hand side of each production rule is not a linear sequence of symbols, but a template mesh with each face representing a symbol.

Thereby the topological structure of an object generated with such a mesh-based PL-system is automatically encoded in the connectivity information of the mesh, and we do not need to introduce grouping symbols in order to encode the hierarchical structure.

It is also very easy to avoid producing degenerate meshes that contain T-vertices, or malformed faces: if the template meshes contained in the rules are well-formed, they will not introduce any degeneracies into the growing mesh.

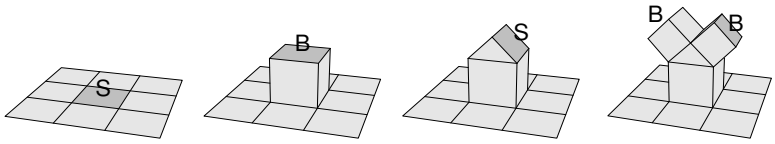

Fig. 5. Mesh growing by associating a symbol with each face.

Figure 5 demonstrates how a mesh can be grown from simple building blocks. Each rule in such a system consists of a symbol associated with a face and a replacement mesh, where each face is again associated with a symbol. In our example in figure 5 the replacement geometry is either a cube-shaped mesh (in the first and third expansion step) or a tent-shaped mesh (in the second expansion step).

Summing up, a mesh based PL-system consists of the following parts (we forego a completely formal specification, since this would be needlessly complex):

- *an initial mesh*: a symbol is associated with each face of this initial mesh.
- *a set of rules*: each of these rules consists of a symbol on the left side, and a replacement mesh on the right side. A symbol is again associated with each face of the replacement mesh.
- *parameters*: in order to parametrize the L system, an environment of variable bindings (parameters) is maintained.
- *calculations*: Each rule can be augmented with arbitrary calculations that modify the parameters.
- *conditionals*: each rule can be augmented by conditionals that allow the definition of alternative replacement geometries based on the result of arbitrary calcuations.

Mesh growing consists of taking the inital mesh, and applying all replacement rules in parallel. Each face that is associated with the left-hand symbol of a replacement rule is replaced by the mesh specified on the right side of the rule, subject to the calculations and conditions that are part of the rule. Symbols that do not appear as the left hand side of any rule are terminal symbols and denote faces that will not be changed anymore. After all rules have been applied in parallel a new mesh is generated, again with symbols associated with each face. Successive rule expansion steps are applied until only terminal symbols are left in the mesh. Figure 6 shows an example of a rule set for growing a mesh.

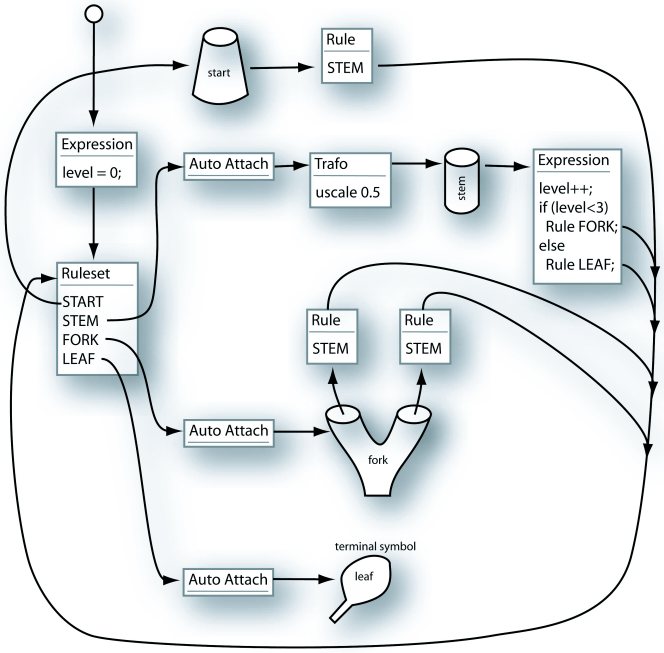

Fig. 6. An example rule set that generates a small tree-like structure.

4.3. *Attaching meshes in a rule*

In order to connect the mesh in a rule to the growing mesh it is necessary to transform the mesh in the rule in such a way that both meshes can be joined without creating degenerated geometry. This is done by specifying a suitable transformation between both meshes. Defining this transformation by hand is straightforward only for the most simple of configurations. For arbitrary meshes it is a sumptuous and error-prone job. In order to relieve the designer from this burden, we introduced an auto-attach operator. This operator can be encoded as a variable transformation in the rule set, that adapts to the current environment each time it is traversed. This means, that each time the auto-attach operator is traversed, it takes a look at the current expansion face which is stored in the traversal environment and also gathers information about the connecting face of the subsequent mesh in the rule. Using this data, a transformation is constructed which will scale, translate and rotate the mesh in the rule to fit onto the current expansion face. Another advantage of using the auto-attach operator is, that meshes can be altered without the risk of breaking subsequent transformations or the need of re-calculating transformations by hand. Figure 7 shows how the auto-attach operator transforms and attaches a leaf mesh multiple times to a stem mesh. For a more formal definition see appendix 1.

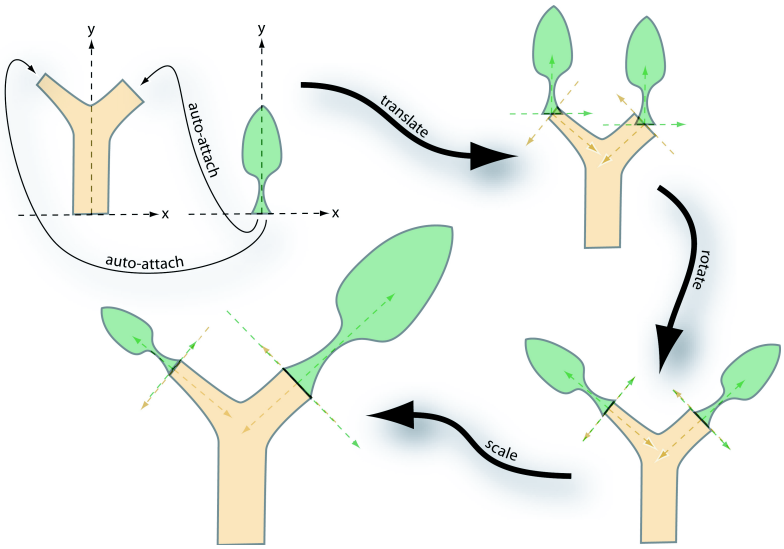

Fig. 7. A leaf mesh is placed on two faces of the mesh of a stem by using the auto-attach operator.

4.4. *Joining arbitrary faces during the growing process*

The standard application of mesh-based PL-sytems will always generate a mesh of genus 0. In order to generate meshes of higher genus, it is necessary to introduce loops or holes. This can easily be done by adding replacement geometry that contains loops or holes. Although this is simple, this limits the introduction of holes and loops by requiring them to be completely represented in a replacement mesh. More complex structures, such as steel frameworks cannot be easily specified in such a way.

In order to overcome this limitation we introduce a generalization of the mesh replacement steps that allows the introduction of arbitrary loops (and thereby implicitly holes):

- *multiple symbol replacement*: instead of only one symbol-carrying face being replaced by the replacement mesh, we allow for multiple faces being replaced by one rule

- *tagging of symbols*: in order to identify which incarnations of the same symbol should be replaced by one step replacing multiple symbols, each symbol carries an (optional) tag. By requiring all the symbols that are replaced in one application of a rule to carry the same tag, the tags are used to identify matching symbols.

- *tag computation*: control of the tagging mechanism is introduced by allowing the tags to be generated by arbitrary computation rules.

A simple example of this type of generalized replacement rule is shown in figure 8. Here the computed tags just represent the generation counter of the growing process. This results in a rule set where matching symbols are always part of the geometry generated in the same generation.

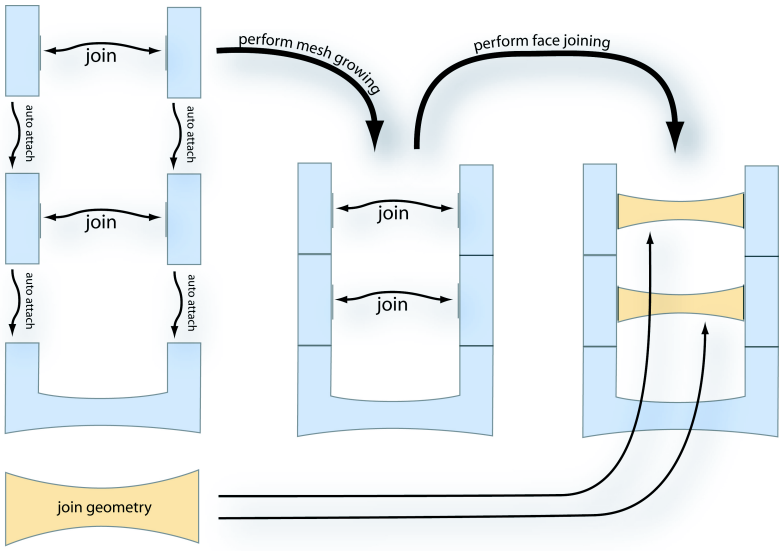

Fig. 8. Application of a generalized replacement rule that replaces two symbols with the same tag with a simple join geometry.

This type of generalized replacement rule can be used to generate geometry with very high genus with very simple rules. As an example we generated a section of a steel framework using this technique (see figure 9).

4.5. *Implementation of Mesh Growing*

We implemented a typed variable binding environment that provides parameters of the following types: integer, floating point, two-dimensional, and three-dimensional vector. Although typed parameters are not strictly necessary, they provide some convenience for implementing parametrized rules.

Currently we have no provision for avoiding intersection of the geometry of neighbouring replacement meshes. It turns out that this does not pose any problem, since it is very easy to design the rules in such a way, that no neighbouring faces will be replaced. In order to solve the self-intersection in a general, global way, open PL-systems have to be used, that store space occupancy in an environment [17].

Using this mesh growing procedure we implemented a PL-system that incorporates the parametrized tree model developed by Weber and Penn [26]. Figure 15 shows various different branches generated with this method.

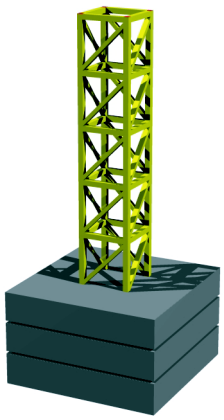

Fig. 9. A section of a steel framework generated by generalized replacement rules.

5. Combining both techniques

Both presented techniques, *generalized subdivision* and *rule based mesh growing* can be combined by using a rule based growing step as an even more generalized subdivision rule. Strictly speaking, no actual subdivision takes place, but since new geometry is introduced there is some similarity between a normal subdivision step and such a mesh growing step. Thus it is possible to alternate between subdivision steps that use texturing functions for vertex placement, and mesh growing steps that expand the geometry in places where it is necessary to add more detail.

Using this combined scheme we can now generate complex geometry that uses the advantages of both of these schemes. As an example, figure 10 shows a forking trunk of a tree.

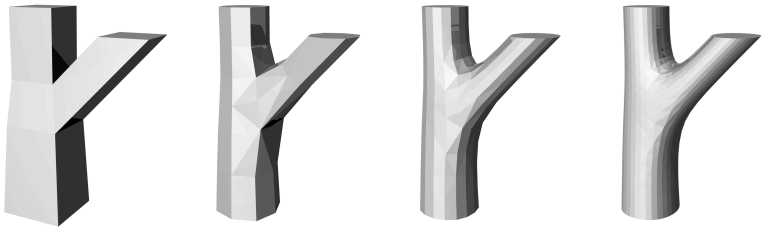

Fig. 10. Generating a branching structure of a tree by combining both techniques.

The branching structure of the tree was modeled using a mesh growing step. Afterwards a number of smoothing steps were used to make the structure look more natural.

The resulting structure, although convincing in its overall form, still lacks detail. After assigning suitable texture coordinates to the vertices in the original mesh, the

method of DeRose et al.[7] can be used to generate texture coordinates at each subdivision level.

In figure 11 these texture coordinates were used to modulate the displacement in the vertex placement step of the following subdivision steps. The actual texture coordinates can be found in appendix B. The left image shows the unmodified groove structure, in the right image, some random displacement of the ridges was added to obtain a more natural look.

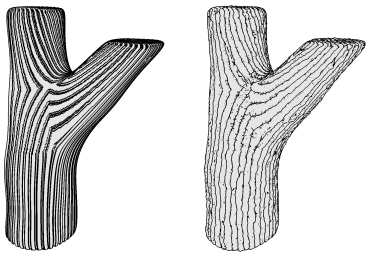

Fig. 11. Ridges generated by vertex placement based on texture coordinates.

6. Rendering

All the methods for generating the generalized subdivision meshes, and for growing meshes have been implemented in a photorealistic renderer based on ray-tracing. Currently the renderer expands all meshes before starting to render the images, however this is not an inherent limitation of our new method. In order to be able to render more complex scenes with our renderer we are working on a caching scheme that only expands the subdivision surfaces in those places where they are visible, only to that extent, that the geometry can be approximated well enough. Optimally a modeler such as the one by Deussen and Lintermann [8] could be modified to directly export our compact representation, thus making it possible to represent even very complex plants within a few kilobytes.

Based on the structure of our generalized subdivision scheme, it is possible to implement levels-of-detail for rendering in two ways: subdivision steps that generate fine detail can be omitted in order to generate simpler models, but it is also feasible to modify the meshgrowing steps to omit generating parts of the geometry that would lead to aliasing if seen from afar. As an example for this, consider the spikes of the cactus in figure 13. These could be omitted altogether for simpler models of the same object.

Although our renderer is based on ray-tracing and thus rather slow, the expansion of the subdivision meshes takes only a fraction of the rendering time (i.e. less than a second); rendering times were on the order of about one hour for a 1000 by 1000 pixel image of the dense bush in figure 15.

7. Results

We used the new methods for generating a number of complex models. The chair in figure 12 demonstrates the use of a random vertex displacement at one level of subdivision in order to model the folds of the cushion.

The overall shape of the cactus in figure 13 has been grown with a simple L-system. After some smooth subdivision steps with a bit of random variation added to generate a more natural look, another mesh growing step was used to generate the spikes. The position of the spikes was chosen by randomly associating the start symbol for the spikes with a small percentage of the faces at a fine subdivision level. Subjecting the resulting structure to more smooth subdivision steps yielded the nicely rounded spikes.

The forking branch (figure 14) shows the ridges that have been generated with our scheme of assigning texture coordinates.

While figure 15 demonstrates the use of our system for trees, the wrought-iron candle-holder figure 16 shows that it is not limited to vegetation scenes (note the rust that has eaten into the iron).

Fig. 12. A chair with folds in the cushion.

8. Conclusion and future work

A general approach for procedural mesh definition has been introduced, which combines multiple techniques derived from subdivision surfaces to parametrized L-

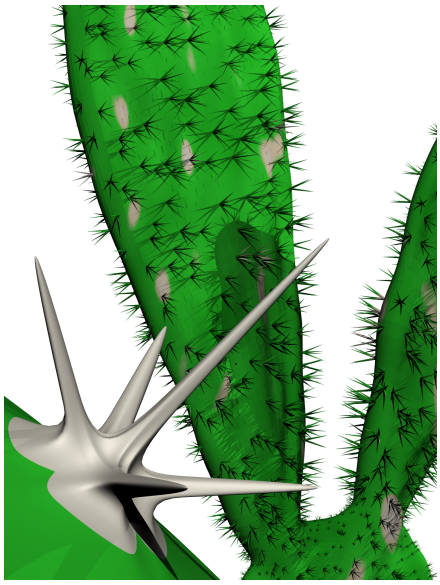

Fig. 13. A cactus generated by alternating between growing and subsivision steps.

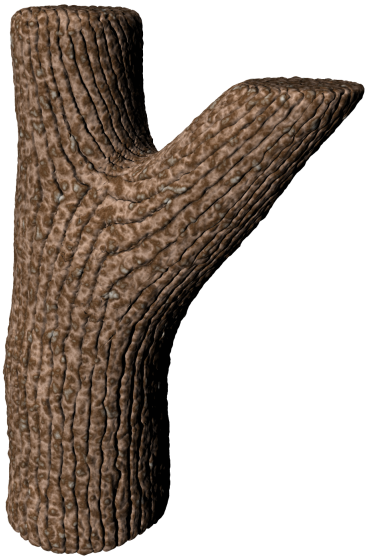

Fig. 14. The ridges on a forking branch.

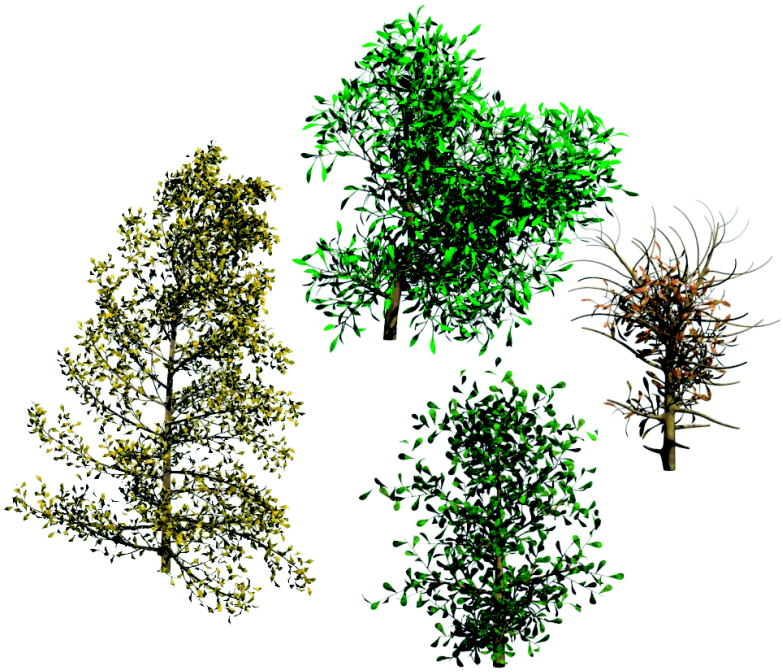

Fig. 15. Different tree branches.

Fig. 16. A wrought iron candle holder with rust stains.

systems. This combined approach yields exceptional modeling power that we used to efficiently define highly complex geometry, such as trees and plants. This combined approach easily copes with smooth branching structures, and makes it possible to use L-systems for defininig geometric structure at widely differeing levels of detail.

By implementing all these procedural mesh generation methods inside the renderer, it is possible to define highly complex scenes using as little as a few kilobytes. Although the actual renderer used for producing the images in this paper generated the complete meshes at each subdivision level, the proposed approach could be used to generate all necessary geometry on the fly.

We are currently working on a rendering system for both interactive and realistic rendering, that only generates these parts of the geometry that are visible, and disposes of the mesh parts that have already been rendered.

Acknowledgements

We would like to thank Anton L. Fuhrman for his suggestions that helped to improve this paper. Most of this work has been done at the VRVis research center, Vienna, Austria (`http://www.vrvis.at`), which is partly funded by the Austrian government research program Kplus.

References

1. Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 113–120. ACM SIGGRAPH, Addison Wesley, 2000.
2. J. Bloomenthal. Modeling the mighty maple. *Computer Graphics*, 19(3):305–311, July 1985.
3. E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978.
4. A. Cavaretta, W. Dahmen, and C. Micchelli. Subdivision for Computer Aided Geometric Design. *Memoirs Amer. Math. Soc.*, 93, 1991.
5. R. L. Cook. Shade trees. *Computer Graphics*, 18(3):223–231, July 1984.
6. Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. *Computer Graphics*, 21(4):95–102, July 1987.
7. Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. *Computer Graphics*, 32(Annual Conference Series):85–94, August 1998.
8. Oliver Deussen and Bernd Lintermann. A modelling method and user interface for creating plants. In Wayne A. Davis, Marilyn Mantei, and R. Victor Klassen, editors, *Graphics Interface '97*, pages 189–198. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 1997. ISBN 0-9695338-6-1 ISSN 0713-5424.
9. D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10:356–360, September 1978.
10. A. Fournier, D. Fussell, and L. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, June 1982.

11. Michael Gervautz and Christoph Traxler. Representation and realistic rendering of natural phenomena with cyclic CSG graphs. *The Visual Computer*, 12(2):62–71, 1996. ISSN 0178-2789.
12. Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. *Computer Graphics*, 33(Annual Conference Series):325–334, 1999.
13. J. T. Kajiya. Anisotropic reflection models. *Computer Graphics*, 19(3):15–21, July 1985.
14. Leif Kobbelt. $\sqrt{3}$ subdivision. In Kurt Akeley, editor, *SIGGRAPH 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 103–112. ACM SIGGRAPH, Addison Wesley, 2000.
15. Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surfaces. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 85–94. ACM SIGGRAPH, Addison Wesley, 2000.
16. A. Lindenmayer. Mathematical models for cellular interactions in development, I & II. *Journal of Theoretical Biology*, 18:280–315, 1968.
17. Radomír Měch and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In Holly Rushmeier, editor, *SIGGRAPH 96, Computer Graphics Proceedings*, Annual Conference Series, pages 397–410. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.
18. F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics*, 23(3):41–50, July 1989.
19. Jörg Peters. Patching catmull-clark meshes. In Kurt Akeley, editor, *SIGGRAPH 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 255–258. ACM SIGGRAPH, Addison Wesley, 2000.
20. Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 101–108. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.
21. Przemyslaw Prusinkiewicz, Aristid Lindenmayer, and James Hanan. Developmental models of herbaceous plants for computer imagery purposes. In John Dill, editor, *SIGGRAPH 88, Computer Graphics Proceedings*, volume 22, pages 141–150, August 1988.
22. Ulrich Reif. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design*, 12(2):153–174, 1995. ISSN 0167-8396.
23. Thomas W. Sederberg, Jianmin Zheng, David Sewell, and Malcolm Sabin. Non-uniform recursive subdivision surfaces. *Computer Graphics*, 32(Annual Conference Series):387–394, August 1998.
24. Jos Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In Michael Cohen, editor, *SIGGRAPH 98, Computer Graphics Proceedings*, Annual Conference Series, pages 395–404. ACM SIGGRAPH, Addison Wesley, 1998.
25. R. P. Voss. Fractal forgeries. In R. A. Earnshaw, editor, *Fundamental Algorithms for Computer Graphics*. Springer-Verlag, 1985.
26. Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In Robert Cook, editor, *SIGGRAPH 95, Computer Graphics Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

Appendix A Auto-Attach Operator

Given two polygons $A = [v_i^A]$ and $B = [v_j^B]$ with $v_i^A \in R^3$, $v_j^B \in R^3$ and

$0 \leq i < n$, $0 \leq j < n$ and surface normals denoted as $\vec{n}_{...}$.

Find a transformation M with $B^{\otimes} = M \times B$ such that

$$-\frac{\vec{n}_A}{|\vec{n}_A|} = \frac{\vec{n}_{B^{\otimes}}}{|\vec{n}_{B^{\otimes}}|}$$

and the sum

$$\sum_{0}^{n-1} |v_n^{B^{\otimes}} - v_n^A|^2$$

is minimized.

Appendix B Assignment of Texture Coordinates in a Branching Structure

We map the ridges of the bark onto the u-coordinate of our texture coordinate system. The v-coordinate is just mapped linearly along the length of the branch, splitting at each fork. At each point where a vertex is shared by the expansion of two separate symbols (faces), the texture coordinate for this vertex is taken to be just the average of the texture coordinates generated by each of the expansions. As an example, figure B.1 shows all the u-coordinates of the vertices in the original mesh of a forking branch.

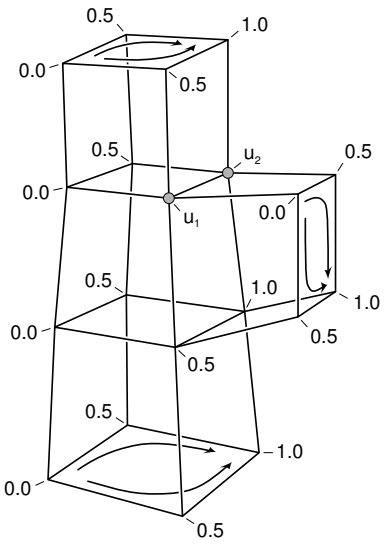

Fig. B.1. u-coordinates at a forking branch: $u_1 = \frac{0.0+0.5}{2}$, $u_2 = \frac{0.5+1.0}{2}$.

In general schemes similar to those developed by Bloomenthal [2] can be devised to cover all the possible surface structures that can be found in nature.