EUROGRAPHICS '92 / A. Kilgour and L. Kjelldahl (Guest Editors), Blackwell Publishers

*Volume 11, (1992), number 3*

# Fractals and Solid Modeling

Eduard Gröller

Technische Universität Wien
Institut für Computergraphik
A-1040 Wien, Karlsplatz 13/186/2
Tel.: +43 (1) 58801-4582
FAX: +43 (1) 5874932
e-mail: groeller@eigv4.tuwien.ac.at

**Abstract**

*Trying to combine fractal geometry and solid modeling seems to be a contradiction in itself, In this paper a new type of 3D objects is presented that accomplishes this combination in a specific way. Objects with a fractal macro structure and a 3D solid micro structure can be specified and rendered efficiently by using context free, attribute, geometric grammars. This new object type can be incorporated into the CSG-modeling technique (Constructive Solid Geometry) in two ways: a) using CSG for the specification of the micro structure of the new object type, b) using these fractal like objects as a new type of primitive in the CSG model. Ray tracing is used for generating high quality images of these geometrically complex objects.*

**Keywords:** fractals, solid modeling, CSG, ray tracing, attribute grammar, L-system

## 1. Introduction

Fractal geometry has gained widespread attention only in recent years. Starting out from the pioneering work of Benoit Mandelbrot [Mand82] fractals have been applied in a variety of different scientific fields. Computer graphics techniques are used in fractal geometry for visualization and analysis of fractal dynamic systems, e.g., Mandelbrot sets, Julia sets, population growth models, "strange" attractors, etc. [PeRi86] , [PeSa88]. On the other hand concepts of fractal geometry are used for modeling various natural phenomena like mountains [FoFu82], [Kaji83], plants [PrLi90], clouds, waves, chemical reactions, physical discharge patterns and so on. High quality rendering (e.g., ray tracing) of geometrically very complex fractal or fractal like objects is characterized by excessive computational and storage requirements. Coherence techniques, e.g., space subdivision or bounding volumes, have to be used to keep the calculation cost down. In [HaFa91] object instancing is used for rendering objects given as 3D IFS (Iterated Function Systems). In this paper a new method is presented that uses attribute grammars to incorporate fractal and solid modeling techniques.

With this approach high quality image generation for very detailed three dimensional fractal like objects can be accomplished at relatively low computation and storage cost. Complex lighting effects that are usual for 3D models, e.g., shadowing, shading, reflection, transparency, can be thus calculated for these fractal like objects as well (see slides)."

## 2. Attribute grammars for fractal like objects

Lindenmayer systems (L-Systems)[PrLi90]have been used with great advantage for modeling various types of plants. An L-Systems consists of a grammar (context free, context sensitive, stochastic, parametric, **...)** and a geometric interpretation of the symbols of the grammar. Derivations of a start symbol (or start string) are calculated and interpreted geometrically by applying the production rules of a grammar in parallel. The number of parallel derivations determines the level of detail of the specified object and can be adjusted to minimize computation and storage cost: an object seen from a large distance does not need to have as much detail as the same object would need for a close-up look. The level of detail can thus be easily controlled by the number of derivations calculated. With the method presented in this paper, a fractal like object is specified as a context free, deterministic, attribute, geometric grammar. An attribute grammar $G = <\Sigma,P,S>$ is hereby defined as follows:

| | | |
|---|---|---|
| $\Sigma$ | alphabet of non-terminal symbols | |
| $P$ | $P \subseteq \Sigma \times (\Sigma)_{i,j,k}$ $1 \le i,j,k \le n$ | production rules |
| $S$ | $S \in \Sigma$, start symbol | |

Each symbol of $\Sigma$ represents a (fractal like) geometric object, that may be visualized by applying the production rules recursively and computing the geometric interpretation of the resulting array of symbols. There is exactly one production rule for every symbol, and each production rule specifies how a symbol is replaced by a 3D array of symbols of $\Sigma$. A symbol on the right side of a production rule may represent a simple 3D object or a fractal like object itself, that can be further specified by applying the corresponding production rules. Production rules have a simple geometric interpretation, so derivations can be calculated easily, and objects can be rendered efficiently.

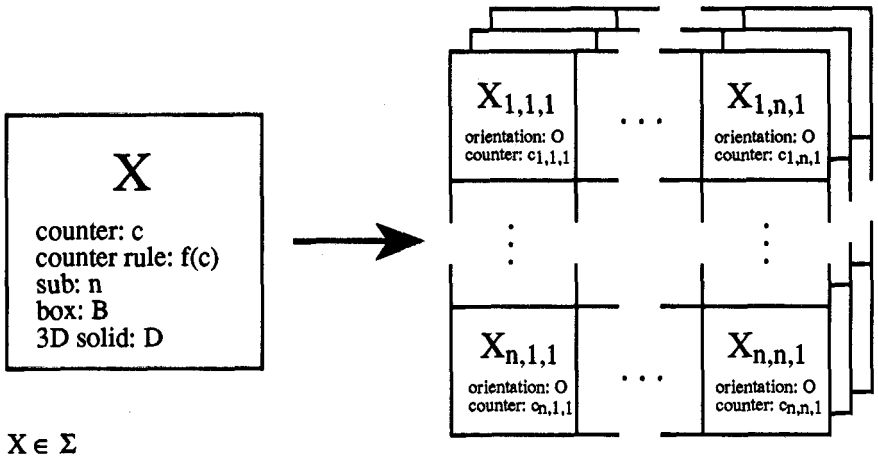
Attributes specified for each production rule are used to influence the calculation of derivations and the geometric interpretation of the resulting set of symbols. With these attributes it is very easy and straightforward to specify, among other things, the desired level of detail for the object represented. A production rule p is defined as shown in figure 1.

Each of the sub boxes on the right side of a production rule may contain a symbol $X_{i,j,k}$ or may be left empty if the sub box does not contain any parts of the object represented by X. The attributes of a production rule have the following meaning: The counter value c is an inherited attribute that determines whether symbol X is replaced by the array of symbols on the right side of production rule p ($c>0$) or the 3D solid D is used as a geometric interpretation of symbol X ($c=0$). The 3D solid D can be defined by using any of the standard 3D modeling techniques like Brep (Boundary Representation) or CSG (Constructive Solid Geometry) and may be arbitrarily complex.

By giving different initial values for the counter c of the start symbol S the same geometric grammar is used to produce an object at different levels of detail. Whenever a production rule is applied the counter values $c_{i,j,k}$ of the symbols $X_{i,j,k}$ are calculated by using the counter rule f(c) of the left side symbol X. These

* **See pages C–484 and C–485 for Slides 1, 2, 3 and 4.**

counter values determine the number of recursions and therefore the termination of the replacing process. An arbitrary function f() can be used as counter rule, e.g., f(c)=c-1, f(c)=c/2. A counter rule for a symbol X need not be strictly decreasing, i.e., f(c) < c. If non-decreasing counter rules are specified, the user however has to take care that all different recursion paths are eventually terminating.




$X \in \Sigma$

counter: $c \in N$

counter rule: function $f:N \rightarrow N$

box: $B \subseteq R^3$, bounding box

3D solid D, Brep or CSG object

$X_{i,j,k} \in \Sigma, \quad 1 \leq i,j,k \leq n$

orientation: O, 3x3 transformation matrix

counter: $c_{i,j,k} = f(c)$

figure 1.: production rule p with attributes

Parameter sub gives the size of the array $(X_{i,j,k})\ 1 \leq i,j,k \leq n$ of symbols of the right side of production rule p. B is a simple rectangular box enclosure of the object represented by symbol X. Whenever production rule p is used the box enclosure B is subdivided into a regular array of n*n*n sub boxes $B_{i,j,k}$ so that $B_{i,j,k}$ determines a box enclosure for the object represented by symbol $X_{i,j,k}$. Orientation O is a simple 3x3 transformation matrix where every row and column contains at most one non zero entry that is either +1 or -1. O is thus a combination of simple rotations (0°, 90°, 180' or 270' along x, y or z axis) and reflections (at xy-, xz- or yz-plane) and determines how the object of $X_{i,j,k}$ is placed within $B_{i,j,k}$ (48 different orientations or positions are possible). Considering only this restricted class of transformations enables a fast ray-object intersection procedure by avoiding costly matrix transformations. Subdividing box enclosure B into sub boxes $B_{i,j,k}$ results in a hierarchical bounding structure, that is used for exploiting spatial coherence properties in the rendering process.

In figure 2 an example of an attribute geometric grammar in two dimensions is shown. There are two production rules, u and v, with u defining the macro structure of the object. Because of the counter rule of production v (f(c)=0) the corresponding 2D solid is always taken as the geometric interpretation of symbol B, in this case the right hand side of production rule v is not of interest and is omitted. The derivation of start symbol A (counter c = 3) is illustrated, for reasons of clearness, by showing the replacing symbols without attributes. The attributes, e.g., counter, orientation, etc., are however taken into account when applying u and v.

attribute grammar G = <{A,B}, {u,v}, A>

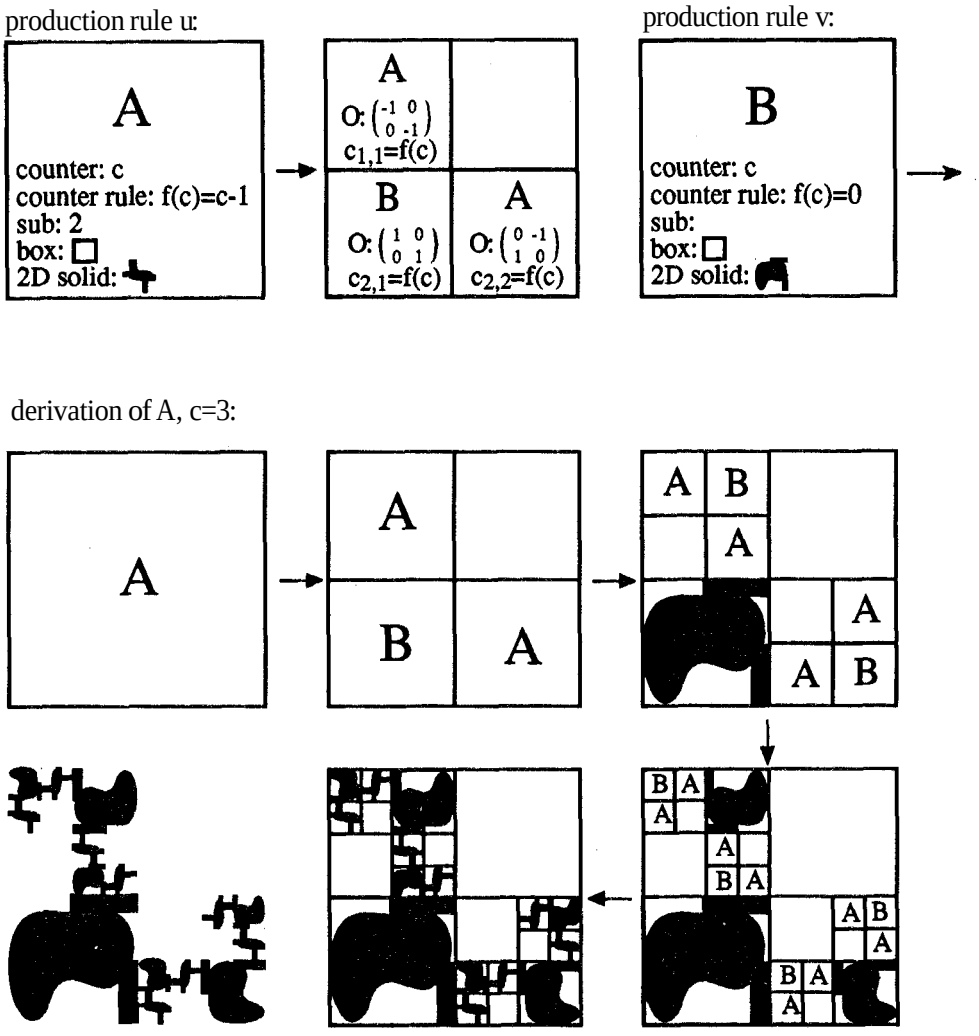



figure 2.: 2D attribute geometric grammar

The fractal like macro structure of objects defined by attribute geometric grammars is given by the production rules of the grammar whereas the 3D solids D specified for every symbol define the corresponding 3D micro structure.

## 3. A new primitive for CSG-modeling

A twofold connection between attribute geometric grammars as defined in section 2 and CSG-modeling can be established:

a) The 3D solid D that is the representation of a symbol X whenever counter c equals to zero may be given as a CSG-object. With this approach a geometrically highly complex object can be defined with very low storage requirements, as the attribute grammar itself and the pertinent CSG-trees do not need a lot of memory.

b) Objects specified by attribute grammars may be used as a new type of primitive (leaf node) in the CSG-model. In section 4 the efficient intersection of a ray with an object given as an attribute grammar is discussed, ray classification that is necessary for finding the intersections of a ray with a CSG-object can be done as usual [Roth82]. The algorithm in section 4 calculates only the first intersection of a ray with an object defined as an attribute grammar. If attribute grammars are used as primitives in the CSG model, more than one intersection point between a ray and a fractal like primitive may have to be calculated. Therefore in this case the intersection test of section 4 has to be modified slightly. In figure 3 the concept of using attribute grammars as leaf nodes in the CSG model is illustrated.

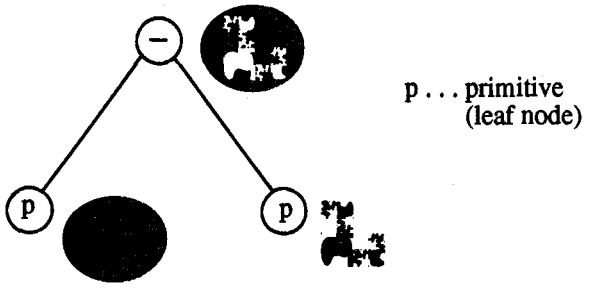



figure 3.: CSG-tree with attribute grammar as primitive

## 4. Rendering

The ray-tracing technique is used for rendering objects given as attribute geometric grammars. As ray-object intersection tests account for most of the computational cost of ray tracing, these tests have to be done efficiently to make ray tracing feasible for fractal like objects. The simple geometric interpretation of production rules, i.e., subdivision of a box enclosure into a regular array of sub boxes, defines an efficient hierarchical bounding structure that is used in the rendering process. With the data model defined above a fast and storage efficient intersection test is done as follows: The fractal like object is not constructed explicitly, only the small portion of the object that is relevant for a specific ray is kept in memory. First a ray is tested with the enclosing box B of start symbol S. If the ray intersects this bounding box B and the counter of S is not zero, symbol S is replaced by applying the relevant production rule. Only those sub boxes $\mathbf{B_{i,j,k}}$ that are intersected by ray r are considered further. Whenever a symbol with counter equal to zero is processed, a ray-object intersection test with the 3D solid specified for this symbol is done. The pseudo code for ray-object intersetion is given below:

```
ray_intersection(X:symbol, r:ray):intersection
begin
    if counter of X equal to zero
    then
        return (intersection of ray r with 3D solid D)
    else
        begin
            apply production rule X →...;
            put all symbols X_{i,j,k} whose sub boxes B_{i,j,k} are intersected
                by ray r in a queue Q in sorted order;
            finished = false;
            while not finished and Q not empty do
            begin
                take first symbol X_{i,j,k} of Q;
                use orientation matrix O to transform ray r, r' = O^{-1}*r;
                intersection = ray_intersection(X_{i,j,k}, r');
                if intersection found
                then
                    begin
                        return (intersection)
                        finished = true
                    end (* if *)
            end (* while *)
        end (* if *)
end (* ray_intersection *)
```

The procedure ***ray__intersection0*** calculates the first intersection point of a ray with a fractal like object. For the calculation of more than just the first intersection point (e.g., if attribute grammars are used as primitives in a CSG-model) the procedure ***ray__intersection()*** must be changed slightly.

To find all the sub boxes $B_{i,j,k}$ that are intersected by ray r the regular arrangement of these sub boxes can be exploited. By using a 3D grid traversal algorithm as in [FuTa86], that basically uses additions and subtractions to replace costly multiplications, the entries of queue Q can be found easily. If a symbol $X_{i,j,k}$ of sub box $B_{i,j,k}$ is processed the orientation O must be taken into account. We are interested in the intersection point of ray r with object $X_{i,j,k}$ transformed by matrix O. Instead of doing a costly transformation of the object represented by $X_{i,j,k}$ with matrix O, the inverse matrix $O^{-1}$ is applied to ray r to give a new ray r'. The intersection test is then done for ray r' and the untransformed object $X_{i,j,k}$. The matrix O has, as explained in section 2, a very simple specific structure. The inverse matrix $O^{-1}$ does have this simple structure as well, so applying matrix $O^{-1}$ means just changing signs and coordinates of the components of ray r accordingly, no costly matrix multiplication has to be done. Thus, because of the special nature of the production rules and their geometric interpretation, ray-object intersection for these geometrically complex objects can be done fast and at low storage cost.

The hierarchical bounding volumes allow the easy use of the inherent spatial coherence of these fractal like objects. Applications of production rules are always stopped if the counter of the corresponding symbol is finally equal to zero. The level of recursion can however be controlled adaptively to make sure that no calculations are performed that do not have any influence on the final image. If the bounding volume of a

symbol projects to a sub pixel area on the final image, recursive replacement of this symbol can be stopped even in case of a non zero counter value.

Most shading models make use of normal vectors to calculate light interaction at surface points. As no normal vectors are usually defined for fractal surfaces, shading such objects is a little bit more complicated. In [HaFa91] different methods are described to generate vectors that are used instead of non existing normal vectors in the shading process. Objects defined through attribute geometric grammars do have a 3D solid micro structure, so normal vectors are defined for any surface point. Normal vectors are calculated when a ray is intersected with a 3D solid D and shading can be done as usual to get sophisticated illumination effects like shadowing, reflection, transparency, and so on. Secondary rays (shadow rays, reflection and transparency rays) can be processed as easily as primary rays by using the hierarchical bounding boxes as well.

## 5. Extensions and modifications

In section 2 a rather simple type of grammar was used for object specification. Such a simple concept enables a fast and easy rendering algorithm. A lot of modifications and extensions are possible which would allow one to generate an even wider class of fractal like objects. However for every extension the implications on the rendering step have to be examined.

Specifying more than one production rule for a symbol and choosing one of the possible production rules at random, increases the class of representable objects with almost no cost in the rendering step. The object is not kept explicitly, so if random numbers are used, it must be assured that the same object (or part of it) is derived for different rays (internal consistency). This can be accomplished by tying the seed value of the random number generator to the spatial positions of the bounding boxes.

Using context sensitive grammars or synthesized attributes (inherited attributes are calculated top-down, synthesized attributes are calculated bottom-up), a larger part of the object will have to be generated explicitly for each ray in the rendering step, making ray-object intersection more costly.

Additional attributes can be specified that may affect the way derivations are calculated or may influence the geometric interpretation of symbols. The geometric interpretation of production rules may be modified so that a bounding box of a symbol is subdivided into an irregular grid of sub boxes. By gaining a new degree of freedom in specifying objects, the grid traversal step in the ray-object intersection procedure becomes more complicated.

Instead of having just one counter rule for a production, different counter rules, one for every symbol $X_{i,j,k}$ on the right side of the production rule, could be defined.

Orientation matrices O (see figure 2) may be selected according to some stochastic attributes. If a wider class of transformations (not only the simple transformations as described in section 2) shall be used for orientation matrices O, the ray-object intersection procedure again gets more complicated. Expensive matrix operations are necessary for ray transformation. In any case a hierarchical bounding volume structure should be used whenever possible. In general a trade-off between the functionality of the attribute grammar and the cost of rendering must be made.

## 6. Implementation

A test system was implemented in Pascal on a cluster of VAX machines (VAX station 2000 and 3200). A module for the intersection of a ray with an object defined as an attribute geometric grammar was incorporated into RISS (Realistic Images Synthesis System), a software package for the generation of realistic images that was developed at our department [GePu88]. Another module has been implemented for the easy interactive specification and modification of attribute grammars. Grammars and their attributes are defined interactively by the user and are stored as very short grammar specification files. These files act as interface to the rendering module of RISS which uses them for ray-object intersection. On the following slides some examples of fractal like objects defined by attribute grammars are shown with some statistics. Calculation times are given, but it has to be pointed out that the tests were done on a rather slow hardware. Most of the calculation time is spent on ray-3D solid intersection and shading.

slide 1:

resolution: 278x400
computation time: app. 117 min.
number of production rules: 8
number of parallel derivations (counter c of the start symbol) : 6

The so called Menger sponge (object in the middle of slide 1) is reflecting. Fractal like mirrors always guarantee an awful lot of secondary rays.

slide 2:

resolution: 311x400
computation time: app. 64 min.
number of production rules: 3
number of parallel derivations (counter c of the start symbol) : 5

slide 3:

resolution: 429x400
computation time: app. 20 min.
number of production rules: 2
number of parallel derivations (counter c of the start symbol) : 5

slide 4:

resolution: 429x400
computation time: app. 208 min.
number of production rules: 1
number of parallel derivations (counter c of the start symbol) : 7

The object is specified with just one production rule. The attribute sub of this rule is 3. With an initial counter c equal to 7 (7 parallel derivations are calculated) an explicit modeling and storing of this object would have resulted in a number of elements on the order of $(3x3x3)^7 \approx 1010$.

## 7. Summary

In this paper a method is described for defining highly detailed fractal like objects with attribute geometric grammars, These objects are characterized by a fractal macro structure and a 3D solid micro structure. Contrary to true fractals these objects do not have new details at all scales, properties of self-similarity are only valid for a certain range of magnification. Ray tracing is used for generating high quality images with sophisticated illumination effects. Spatial coherence properties (bounding volume hierarchies) are exploited for rendering these objects efficiently. Objects are given only implicitly through the attribute grammar, explicit modeling of these objects would not have been easily possible due to excessive requirements in storage space and computational cost. Even if a high level of detail is necessary the database is not increased but only the relevant attributes are modified. Fractal concepts are combined with 3D solid modeling techniques.

## 8. References

[Barn88] Bamsley, M.: Fractals Everywhere, Academic Press, 1988.

[Falc90] Falconer, K.: Fractal Geometry, J. Wiley & Sons, 1990.

[FoFu82] Fournier, A., Fussell, D., Carpenter, L.: Computer Rendering of Stochastic Models, Communications of the ACM, Vol. 25(6), pp. 371-384, June 1982.

[FuTa86] Fujimoto, A., Tanaka, T., Iwata, K.: ARTS: Accelerated Ray-Tracing System, IEEE Computer Graphics & Applications, pp. 16-26, April 1986.

[GePu88] Gervautz, M., Purgathofer, W.: RISS-Ein Entwicklungssystem zur Generierung realistischer Bilder. In Visualisirungstechnikenund Algorithmen (Hrsg. W. Barth) Informatik-Fachberichte 182, pp. 61-79, Springer Verlag, Berlin 1988.

[HaFa91] Hart, J.C., DeFanti, T.A.: Efficient Antialiased Rendering of 3-D Linear Fractals, Computer Graphics 25(4), pp. 91-100, 1991.

[Kaji83] Kajiya, J.: New Techniques for ray tracing procedurally defined objects, Computer Graphics Vol. 17, No 3, July 1983.

[Mand82] Mandelbrot, B.: The Fractal Geometry of Nature, W. H. Freeman and Company, New York, San Francisco, 1982.

[PeRi86] Peitgen, H.-O., Richter, P. H.: The Beauty of Fractals, Springer Verlag, 1986

[PeSa88] Peitgen, H.-O., Saupe, D.: The Science of Fractal Images, Springer Verlag, 1988.

[PrLi90] Prusinkiewicz, P., Lindenmayer, A.: The algorithmic beauty of plants, Springer Verlag, 1990.

[Roth82] Roth, S. D.: Ray Casting for Modeling Solids, Computer Graphics and Image Processing 18, pp. 109-144, 1982

## Acknowledgment

The author would like to thank Hannes Bienenstein and Peter Gassy for their help in implementing the test system. Thanks to my colleagues Michael Gervautz, Werner Purgathofer and Michael Zeiller for valuable suggestions during the preparation of this paper.