J.A. Kaandorp

Interactive generation of fractal objects

Computer Science/Department of Interactive Systems Report CS-R8706 January

Interactive Generation of Fractal Objects

J. A. Kaandorp
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In this paper a description is given of the development of data structures by which a large class of fractal objects can be represented using production rules, together with an algorithm to generate these objects. These data structures and algorithm can be used to produce many of the classical examples of self-similar sets, mentioned for example by Mandelbrot [13] . The algorithm and data structures are part of a fractal system by which production rules can be designed; production rules can be retrieved from and stored in a fractal library.

1980 Math. Subject Classification : 58F13, 69K30, 69K34
Key Words & Phrases : fractals, self-similar sets, computer graphics, graphics utilities
Note : This report will be submitted for publication elsewhere.

1. INTRODUCTION

Fractals have received much attention recently; a large number of articles on the application of fractals in all kinds of different areas of science, are an indication of this. Some examples of those applications are: [19] (physical geography), [12] (biochemistry), [15] (biology), [16] (physics).
A description of fractals can be found in Mandelbrot [13] and in Barnsley and Demko [4] . Fractals are defined by Mandelbrot as sets whose Hausdorff-Besicovitch dimension exceeds its topological dimension. Instead of the Hausdorff dimension another dimension definition, the capacity [7] can be used. Fractals may be defined as sets with, in most cases, a fractional dimension. The definitions Hausdorff-Besicovitch dimension, capacity and fractional (fractal) dimension will be used as synonyms in this paper. Fractals show a self-similar structure, this phenomenon may be used as the guiding principle for a more general definition [8] A mathematical description of fractals is given by [3] .
Fractals form a powerful tool for creating "objects", which may look quite complicated at first sight, but can be defined and computated in a relatively simple way. This technique is especially important for creating objects which show a high resemblance with objects from nature like trees, clouds, mountains etc. The fractal dimension can be demonstrated for several objects from nature, for example the length of a coastline, surface of the brain, lung, vascular system (all examples from Mandelbrot [13]). Fractals seem to serve quite well as models of natural objects, an example of a study from mathematical biology in which fractals are used for modelling is Meakin [14] . Many processes, for example growth processes in biology, may be described as feed-back processes in which the output of one iteration is the input for the next one. The same feed-back process can be used for creating fractal objects.
The purpose of this paper is to present an interactive method for the generation of a large class of fractal objects. This method will only deal with self-similar sets [3] , this class of fractals is also known as linear fractals [13] . The fractals of this class, which contains many of the classical fractals, are self-similar and are built from pieces which are geometrically similar to the entire set but on a smaller scale. This property of self-similarity will be disturbed slightly, or in some cases almost completely, in the fractals which will be shown in this paper. One advantage of self-similar sets is that production rules, by which fractal objects can be created, may be formulated in a simple way. It is relatively easy to enter those production rules in an interactive fractal system. The property of self-similarity is a very important one, because it often can be related to the physical or mathematical properties of the objects being modelled by fractals. The importance of using an interactive system partly lies in the possibility to find the "right" kind of self-similarity for a certain modelling problem.
Other studies in which methods are described for generating fractal objects from the class of self-similar sets are: Demko [6] , in this study fractal objects are generated using Iterated Function Systems and Smith [18] , in which formal languages are used for creating fractals. Many examples of self-similar sets can be found in Mandelbrot [13] , a mathematical description of these sets can be found in Dekking [1] , Falconer [3] and

Report CS-R8706
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Hata [8] .
The fractal system, discussed in this paper, was implemented using 2D C-GKS [9] ,[5] ; 3D C-GKS [10] and C as programming language. The data structures used for representing the components of production rules for fractals refer to data structures defined in C-GKS.

2. Methods

2.1 Production rules and data structures necessary for representing fractals

In Mandelbrot [13] many examples are given of production rules for self-similar sets. The production rules in Mandelbrot are frequently augmented by non-grammatical rules [18] . A number of self-similar sets can be represented by an initial polygon (the initiator) and a polygon (the generator), which replaces sides of the initiator. One of the quadric Koch curves is represented by the production rule in fig. 1. The data structure necessary for representing the initiator (and all succeeding iteration steps) and the generator is very simple and consists only of two fields: the number of world coordinates and a pointer to the world coordinates.

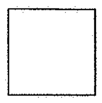
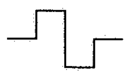
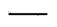
initiator	generator	base element
		

Figure 1. Production rule of a quadric Koch curve. The resulting fractal is shown in fig. 2.

The self-similar sets can be classified on the basis of the data structures minimally necessary for representing all components of the production rule. In fig. 2 a possible classification is shown of some self-similar sets. In the classification diagram of fig. 2 the quadric Koch curve is the most primitive fractal.
An additional rule may be introduced in the production rule. By defining some of the sides of the initiator "fertile" or "not-fertile", this rule controls which of the sides participate in the next iteration. In the production rule for a simple tree (fig. 3) it is necessary to define the fertilisation state of the sides of the initiator and generator. In the diagram (fig. 2) the data structure for representing the simple ramiform fractal is extended with a new field: a pointer to booleans values, which determine the fertilisation state of the sides of the two components of the production rule.
A new class of ramiform fractals may be created by introducing functions, which randomize the original generator in the production rule. A random processing function which will generate a ramiform fractal with a more irregular appearance is, for example, a function which allows the original generator to make random movements between two limits (see fig. 4). The data structure of the generator is extended by a field for a random processing function.
Another class of self-similar sets are the fractals (more accurately: approximants of fractals), which are able to seed new fractals. The data structure of the generator has to be extended to an array of different parts of the generator. The data structure of the initiator and next iteration steps is an array of fractals. The production rule for a self seeding square is shown in fig. 5.
The possibilities of the ramiform fractal from fig. 4 may be enlarged by using an array of generators, consisting of an array of parts of the generator. A selection function is used to determine which generator should be used for replacing line-pieces of the preceding fractal approximant by a new set of line-pieces. In this example the growing generator (gen0) is chosen as long as the "age" of the fractal approximant doesn't equate 3 iterations. On the third iteration the fractal starts seeding a new set of fractals (age equals 0) using the seeding generator (gen1). The production rule for a self reproducing ramiform fractal (see fig. 2 and fig. 7), in which two generators are used is shown in fig. 6. The original ramiform fractal has changed into a vegetation of fractals.

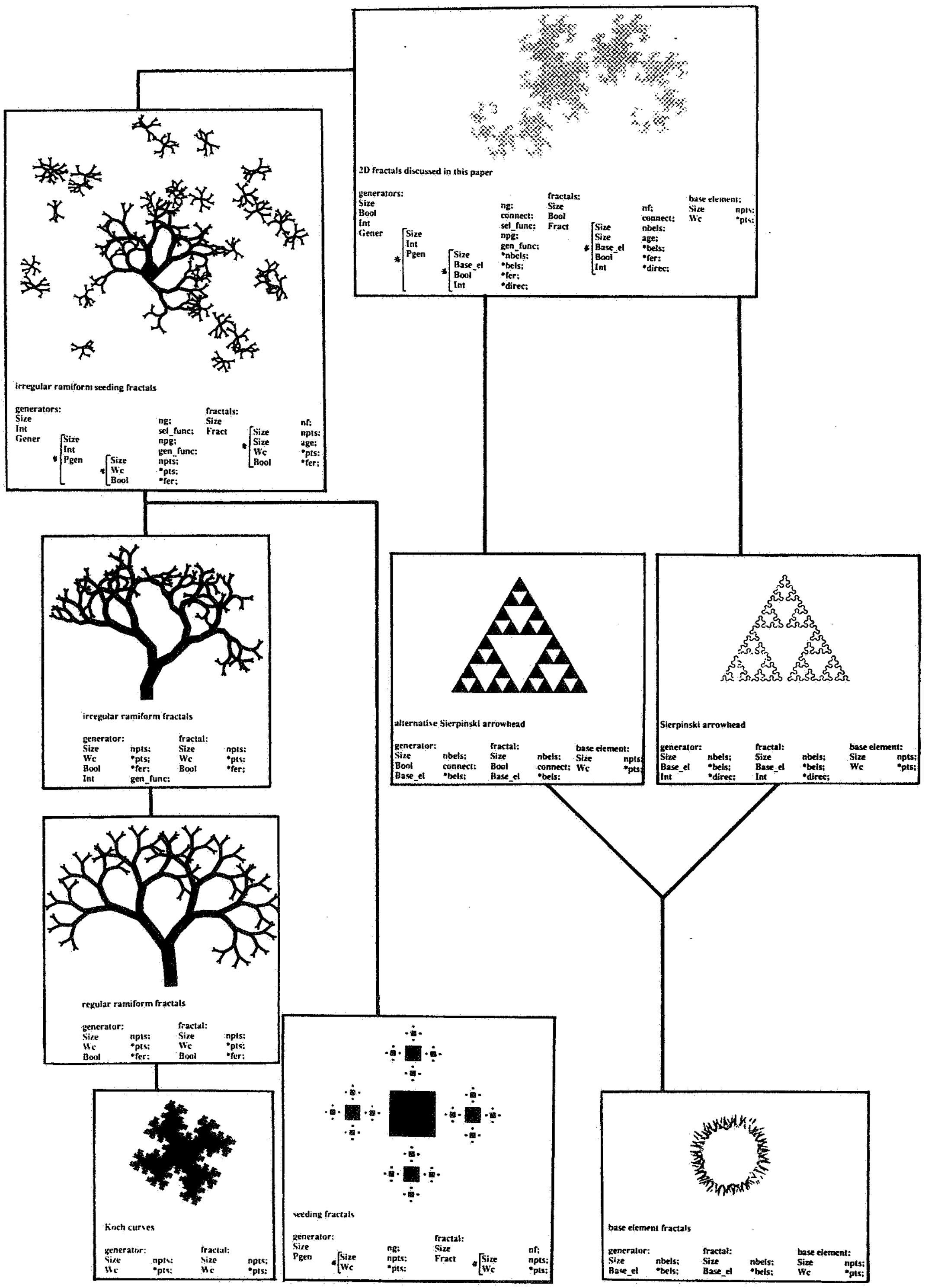

2D fractals discussed in this paper
generators: Size Bool Int Gener [Size Int * Pgen [Size * Base_el Bool Int]] ng; connect; sel_func; npg; gen_func; *nbels; *bels; *fer; *direc;
fractals: Size Bool Fract * [Size Size Base_el Bool Int] nf; connect; nbels; age; *bels; *fer; *direc;
base element: Size Wc npts; *pts;
irregular ramiform seeding fractals
generators: Size Int Gener [Size Int * Pgen * [Size Wc Bool]] ng; sel_func; npg; gen_func; npts; *pts; *fer;
fractals: Size Fract * [Size Size Wc Bool] nf; npts; age; *pts; *fer;
irregular ramiform fractals
generator: Size Wc Bool Int npts; *pts; *fer; gen_func;
fractal: Size Wc Bool npts; *pts; *fer;
regular ramiform fractals
generator: Size Wc Bool npts; *pts; *fer;
fractal: Size Wc Bool npts; *pts; *fer;
Koch curves
generator: Size Wc npts; *pts;
fractal: Size Wc npts; *pts;
seeding fractals
generator: Size Pgen * [Size Wc] ng; npts; *pts;
fractal: Size Fract * [Size Wc] nf; npts; *pts;
alternative Sierpinski arrowhead
generator: Size Bool Base_el nbels; connect; *bels;
fractal: Size Bool Base_el nbels; connect; *bels;
base element: Size Wc npts; *pts;
Sierpinski arrowhead
generator: Size Base_el Int nbels; *bels; *direc;
fractal: Size Base_el Int nbels; *bels; *direc;
base element: Size Wc npts; *pts;
base element fractals
generator: Size Base_el nbels; *bels;
fractal: Size Base_el nbels; *bels;
base element: Size Wc npts; *pts;

Figure 2. Classification diagram of self similar sets based on minimal data structures necessary for representing all components of the production rules. The data structure on top of the classification can be used for representing all fractals discussed in this paper. The resulting fractal is shown in fig. 2.

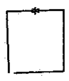
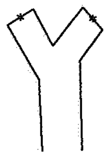
initiator	generator	base element
		

Figure 3. Production rule for a simple tree (Pythagoras tree). Fertile sides (sides of the preceding fractal approximant which will replaced in next iteration steps, are marked with asterisks. The resulting fractal is shown in fig. 2.

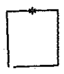
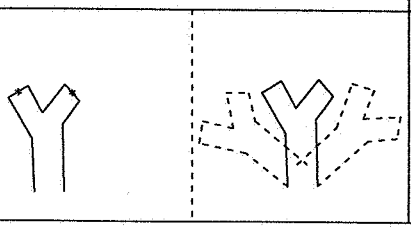
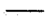
initiator	generator	base element
		

Figure 4. Production rule for a tree in which the original generator is processed by a function which allows random movements of the generator between two limits. The generator processing function is described in the right part of the generator component. The resulting fractal is shown in fig. 2.

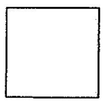
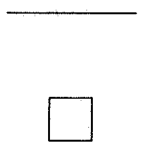
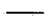
initiator	generator	base element
		

Figure 5. Production rule for a self-seeding square. The generator consists of two parts and is seeding new fractals during each iteration. The resulting fractal is shown in fig. 2.

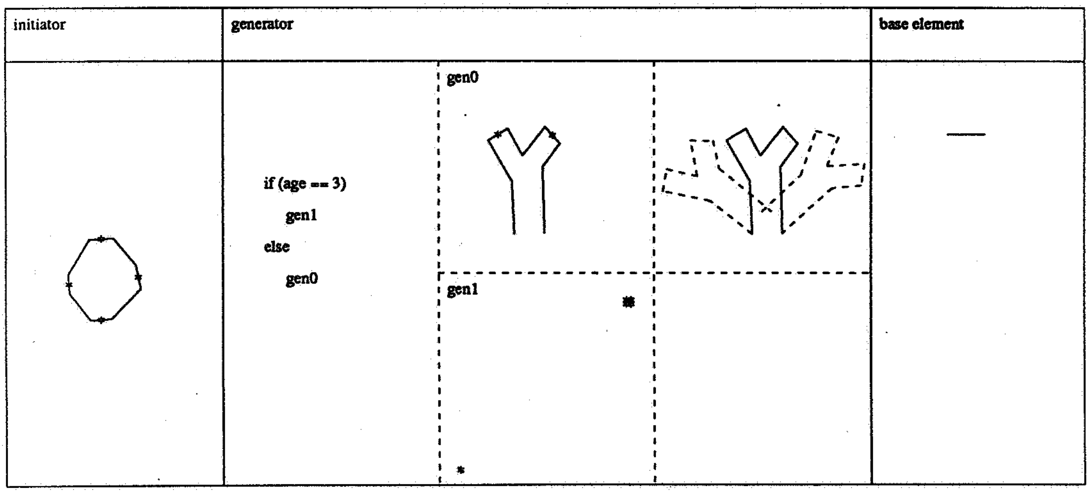


Figure 6. Production rule for a vegetation of fractals. The growing generator (gen0) is chosen by the selection function (left part of the generator component) as the age of the fractal doesn't equal 3 iterations, on the third iteration the fractal starts seeding (gen1). The resulting fractal is shown in fig. 7.

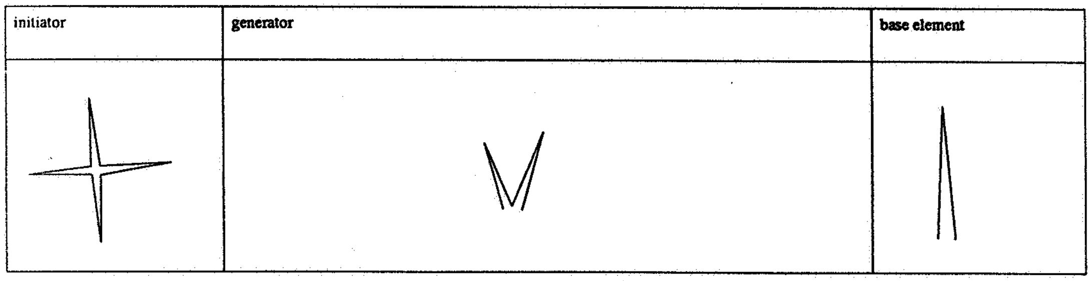


Figure 8. Production rule for a simple base element fractal. The resulting fractal is shown in fig. 2.

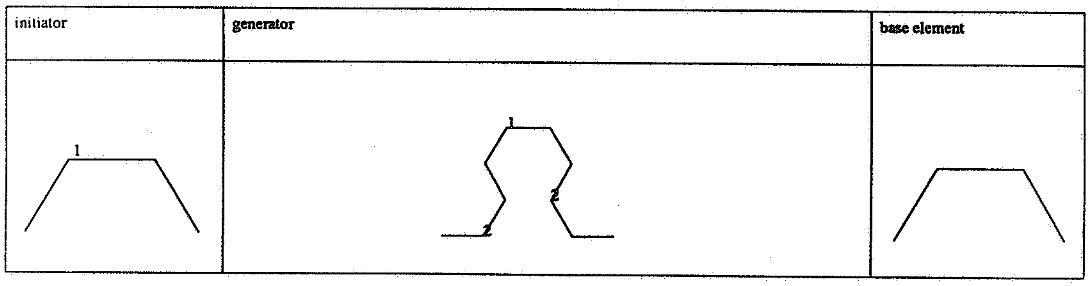


Figure 9. Production rule for a Sierpinski arrowhead. The direction of the base elements is denoted in the first two components with numbers: direction 1, normal base element; direction 2, y-coordinate has been reflected. The resulting fractal is shown in fig. 2.

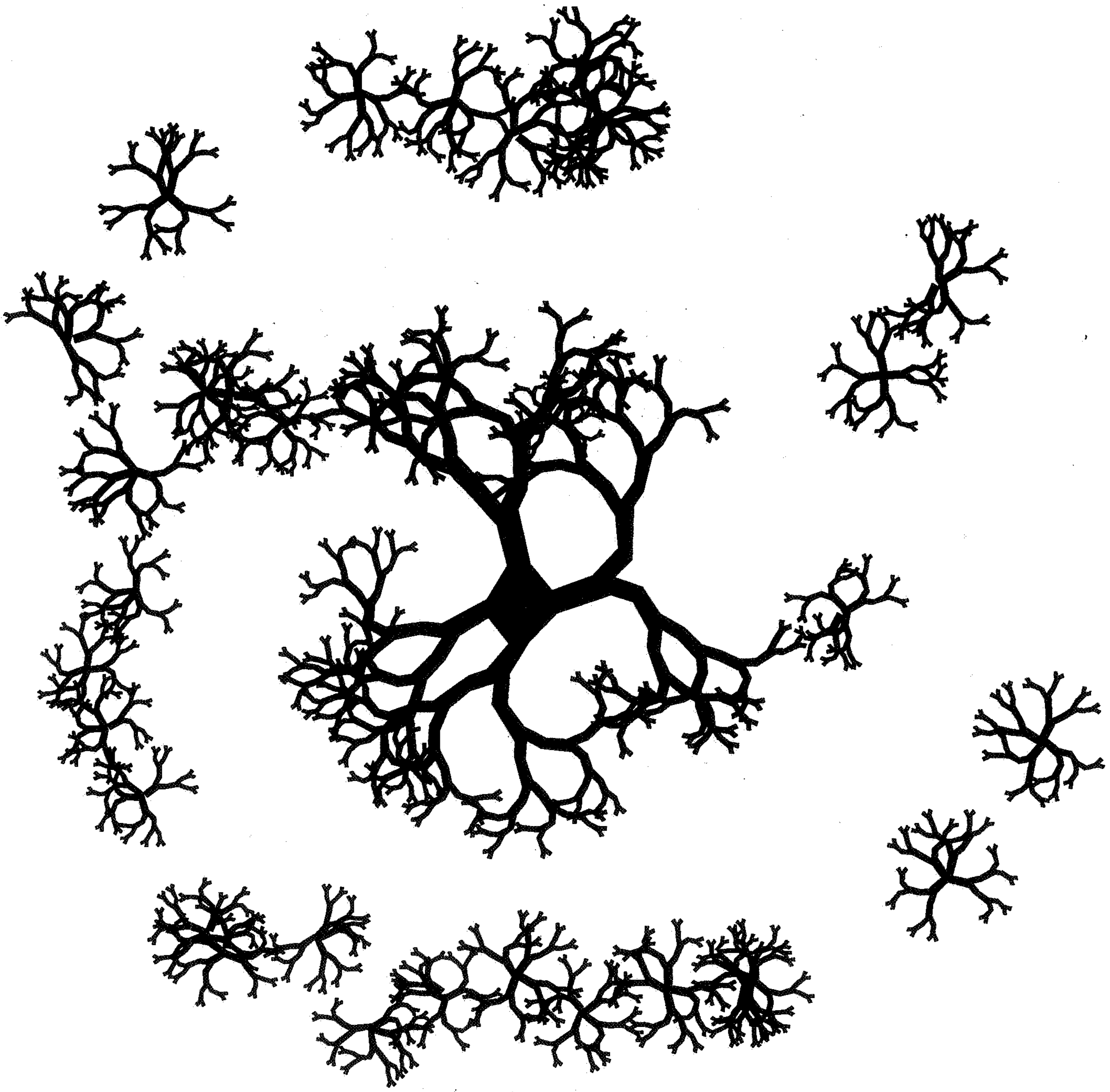


Figure 7. Vegetation of fractals, resulting from the production rule shown in fig. 6.

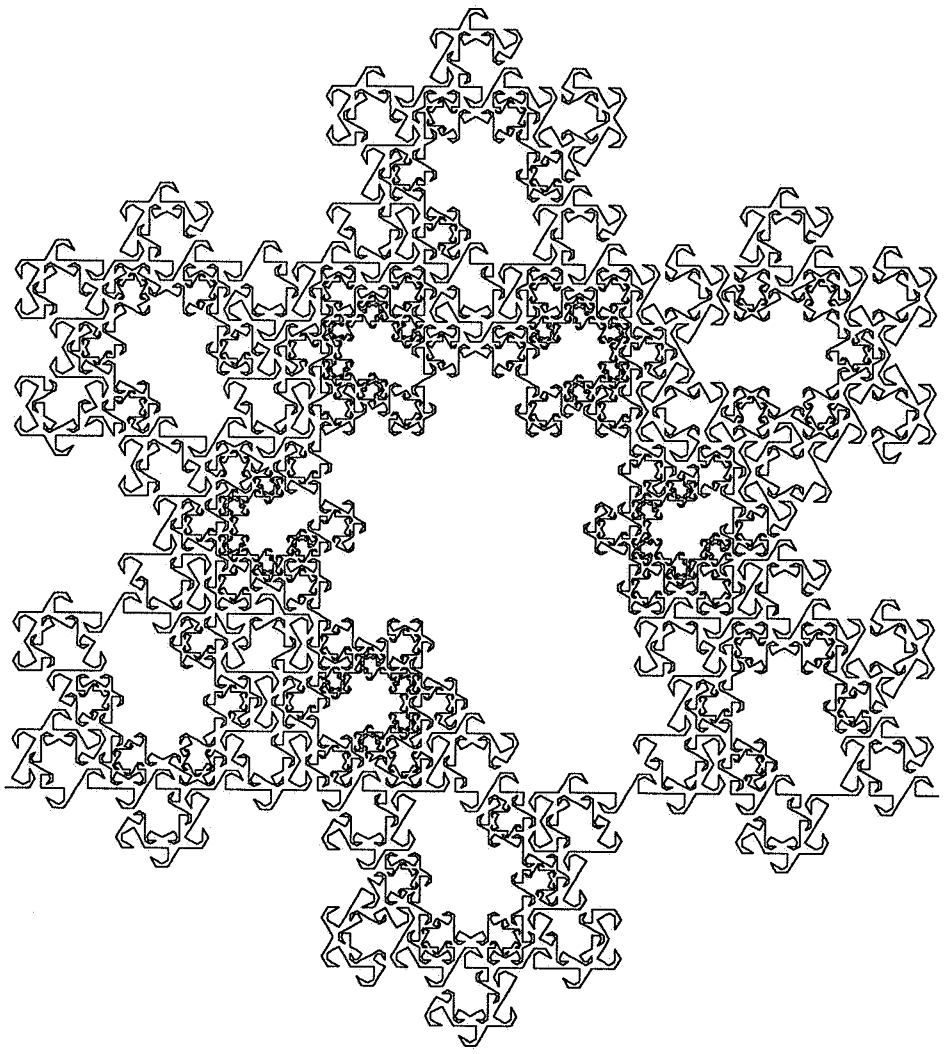


Figure 10. Monkeys tree (Mandelbrot, 1982), the production rule of this fractal can be found in fig. 11.

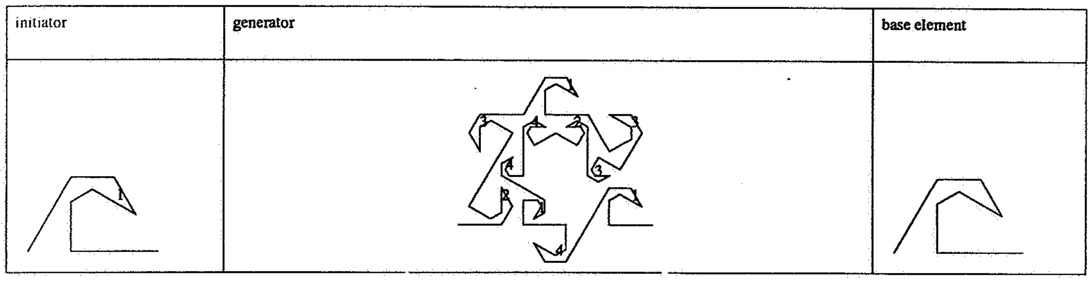


Figure 11. Production rule for the Monkeys tree shown in fig. 10. The direction of the base elements is denoted in the first two components with numbers: direction 1, normal base element; direction 2, y-coordinate has been reflected; direction 3, x-coordinate has been reflected; direction 4, x- and y-coordinate has been reflected.

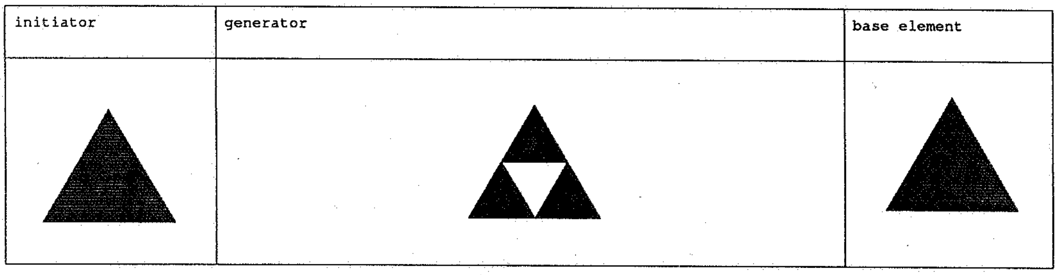


Figure 12. Production rule for an alternative way of constructing a Sierpinski arrowhead. The resulting fractal is shown in fig. 2.

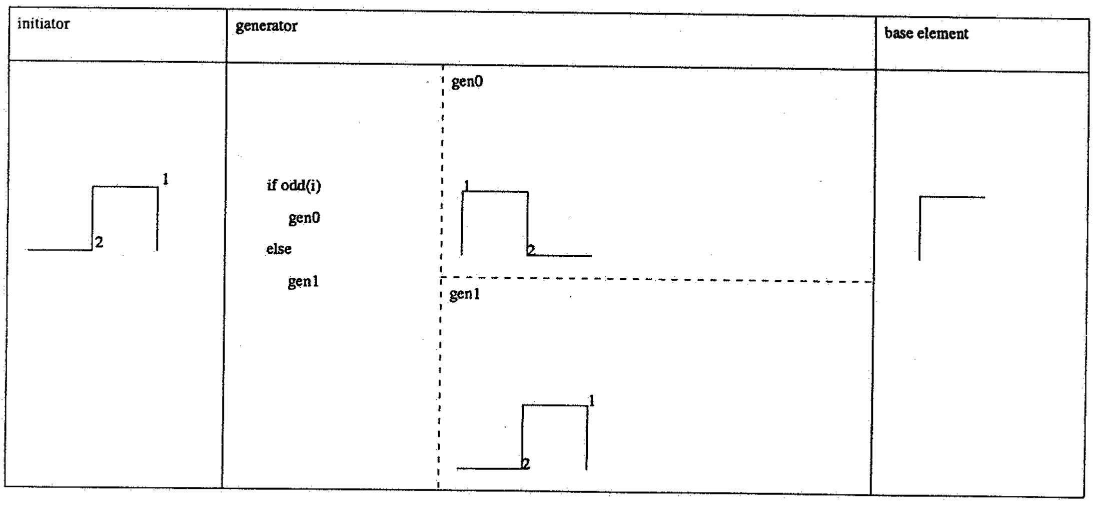


Figure 13. Production rule for constructing the Dragon sweep, odd base elements of the preceding fractal approximant are replaced by gen0, even base elements by gen1. The resulting fractal is shown in fig. 2.

In all fractals discussed so far fertile sides, consisting of one line piece, of the preceding approximant of the fractal were replaced by a new set of line pieces. It is possible to create a new class of fractals by the replacement of a polygon by a set of polygons. This polygon is called a "base element". The production rule for a simple fractal, built from base elements, is shown in fig. 8. The data structures for representing the production rule for this class of fractals have to be enlarged by base elements as new component.
The next step in creating new fractal objects is to add a new field in the data structure of the generator and fractal approximants, which defines the direction of the base elements in both components. In this paper for the 2D fractals four different directions have been used: direction 1, normal base element; direction 2, y-coordinate of base element is reflected; direction 3, x-coordinate of base element is reflected; direction 4, x- and y-coordinate of base element are reflected. As a result it is possible to construct, for example, the Sierpinski arrowhead (see diagram fig. 2, the production rule is given in fig. 9). Another example of a fractal which may be described with a production rule using base elements with different directions is the Monkeys tree of Mandelbrot [13] shown in fig. 10 (the production rule is given in fig. 11).
So far all base elements or line pieces of the fractals were connected. It is easy to define fractals with disconnected base elements by adding a boolean to the data structure of the generator and initiator, which indicates whether the base elements are connected or not. An example of a fractal using disconnected base elements is the alternative way to construct a Sierpinski arrowhead, using triangles as base elements. It is shown in fig. 2 (production rule is given in fig. 12).
When all different rules to generate fractal objects are put together in three data structures, it is possible to cover all production rules given above. The resulting data structures are shown in the top of fig. 2. With the resulting data structures it is possible to describe all, interactively defined, production rules in the fractal system. With these data structures, production rules of still more complex fractal objects can be defined. For example the components of the production rule (fig 13) of the Dragon sweep [13] may be represented by these data structures. The Dragon sweep (see fractal in top of fig. 2) is created by using an array of two generators which are alternating. The odd base elements of the preceding fractal approximant are replaced by gen0, the even base elements by gen1.

2.2 An algorithm for the generation of fractal objects

The algorithm described in this paper is suitable for generating all classes of fractals shown in fig. 2. This algorithm is suitable for 2D fractals, the extension to 3D will be discussed below. In order to allow dynamically growing fractal objects, during calculation, a dynamic data structure is used. The dynamic data structure allows random growth and seeding of fractal objects and consists of a list of pointers (the "entrances" to the fractals) to fractal objects. The fractal objects are represented by lists consisting of elements. Each element is a set of base elements together with the parameters (fertilisation, direction). The diagram of fig. 14 shows the dynamic data structure used in the fractal algorithm. The algorithm itself is described in a summarized form using pseudo code below:

```
fractal( old_fractals, new_fractals, generator, base_element ) {
  A: if ((no random generator is used) && (no selection function is used))
     a local copie of the generator is made and the translation vectors are derived from the generator;
  B: next fractal from old fractals is taken {
    C: next base_element from current old fractal is taken {
      D1: if (current old base_element is fertile) {
        D1a: first point is taken from the current old base element and temporarily stored, this value is
        the initial value used in G;
        D1b: if ((random function is used) || (selection function is used))
          In case a selection function is used a generator is selected from the array of generators, in
          case a random processing function is used a local copie (randomly changed by a function )
          is made of the original generator, translation vectors are derived from the local copie;
        D1c: the distance between the last and the first point of the current old base_element is calcu-
        lated;
        D1d: the orientation of the line through last and first point of the current old base_element is
        calculated;
        E: next part is taken from the generator {
          F: next base_element is taken from the generator {
            G: next point is taken from base_element of the generator {
              G1: translation is performed using the vectors from A / D1b and the result of D1c;
              G2: Rotation is performed using results of D1d;
              G3: if (fractal is seeding)
                calculated value is used for jumping to new point;
              else
                new point is added to the dynamic lists;
            }
            F1: if (fractal is not seeding)
              fertilisation status of new base_element (equal to fertilisation status of current
              base_element generator) is added to the dynamic lists, direction of the new
              base_element is evaluated from direction current base_element generator and
              current old base_element and added to the dynamic lists.
          }
          E1: if (fractal is seeding)
            a new entrance for a new fractal is added to the dynamic lists, the value from D1a is
            used as initial value;
        }
      }
      D2: if (current old base_element is not fertile)
        old base_element and its fertilisation and direction state is added to the dynamic lists;
    }
  }
  H: the new fractal stored in the dynamic lists is copied into the static data structure of the new fractal;
}
```

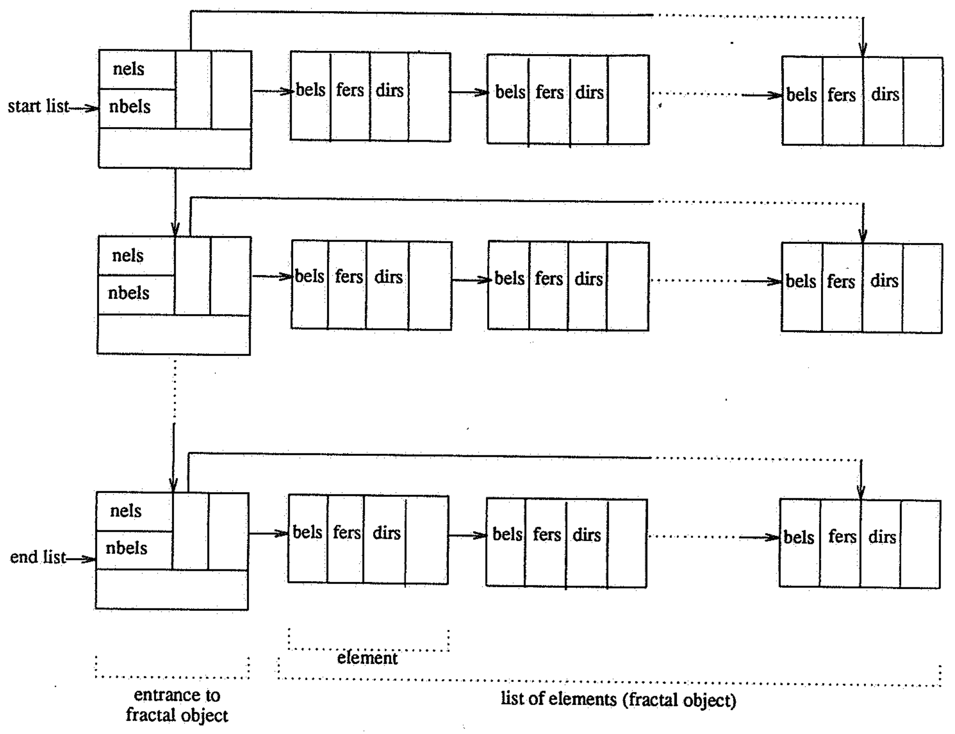


Figure 14. Diagram of the dynamic data structure used in the algorithm for calculating fractals. The first column of the diagram contains a list with pointers to the fractal objects (the entrances to the fractals), together with two parameters (nbels = total number of base elements, nels = number of elements in the connected list). The other colums consists of elements which may contain different sets of base elements (bels), together with the fertilisation (fers) and direction (dirs) state of the base elements.

2.3 Description of the fractal system

The fractal system is described in a diagram in fig. 15. In this diagram the fractal system is enclosed by a dotted rectangle. As discussed above, it is necessary to make production rules, built up from three components: initiator (which could also be denoted as the 0-approximant of the fractal), generators, and base elements. The three components can be designed interactively in the fractal system (the first interactive state in the system) or taken from an external st rage. This external storage con-ists of files, which contain the parameters of the data structures, which describe the three components. Newly designed components can be added to these three library files.

The generators may contain fields which refer to a special function which processes the original generator or to a selection function which selects, during the calculation of the fractal, a generator from an array of generators. Both the generator processing and the generator selection function are described internally in the fractal system.

The three components may be visualized using a special part of the fractal system for showing production rules. The main part of the fractal system is the fractal algorithm, which uses the output of one iteration as input for the next one. The fractal generation is the second interactive state in the system. After a production rule has been defined, next fractal approximants may be calculated. Interactively, within certain physical limits, is decided which fractal approximant yields a satisfying result.

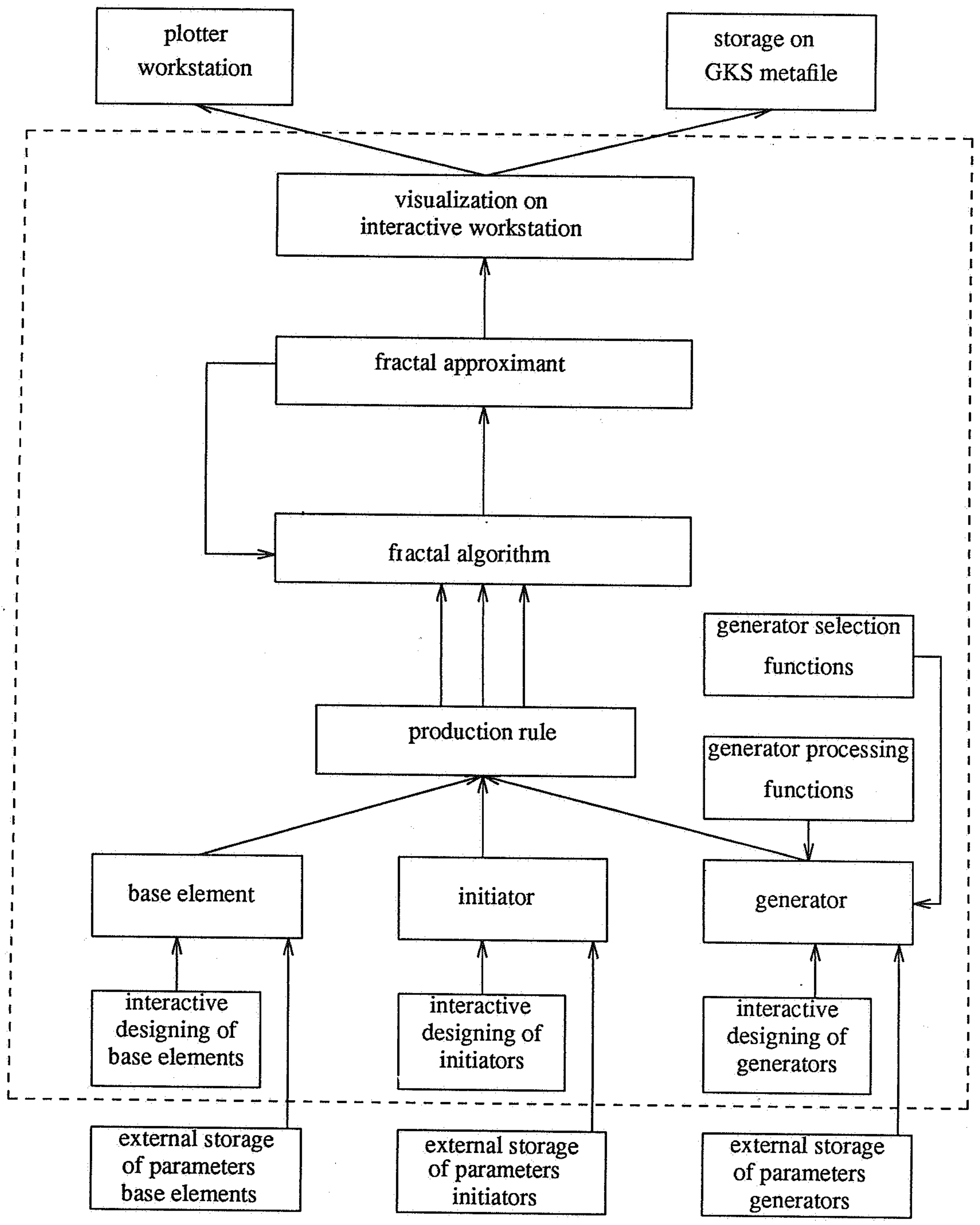

plotter workstation
storage on GKS metafile
visualization on interactive workstation
fractal approximant
fractal algorithm
generator selection functions
production rule
generator processing functions
base element
initiator
generator
interactive designing of base elements
interactive designing of initiators
interactive designing of generators
external storage of parameters base elements
external storage of parameters initiators
external storage of parameters generators

Figure 15. Diagram of the fractal system, the fractal system is enclosed by a dotted rectangle.

The third interactive state is the fractal manipulation. The visualization of the fractal may be influenced interactively by changing GKS attributes (colour, fill area interior style etc.) and by performing transformations on the fractal object. Except for the visualization of the fractal on the interactive workstation, the resulting fractal can be externally stored on a GKS metafile or send to a plotter workstation.
The three interactive states may be freely interleaved. It is possible to define new production rules during the generation of fractal objects, so the ultimate fractal object can be influenced interactively.

2.4 Extension to 3D GKS

For the extension to 3D GKS the fractal system is simplified partly. The part in the fractal system (fig. 15) in which the three components are designed interactively is skipped. The only way to introduce new production rules in the fractal system, so far, is to change the externally stored library files.
The data structure necessary for representing production rules is extended by a new level. Instead of Wc-coordinates in the base element a new component called "surface" is introduced. The surface consists of four fields: the number of Wc3 coordinates used in the surface, a pointer to the Wc3 coordinates, a field with the normal vector of the surface and a field with the colour index of the surface.
The algorithm for the generation of fractal objects (see above) is extended by a new level in part G in which the points are taken from the surfaces. Part D1d is changed into the calculation of the orientation of the surface and the necessary axis of rotation. The transformations in G1 and G2 are changed into 3D transformations.
The visualization in the fractal system (fig. 15) is less trivial for 3D objects. The fractal object has to be processed in several stages before it can actually be visualized in a proper way. In this 3D version the visualization is performed in about the most simple way. First all surfaces, with a normal vector indicating that they will never be visible, are removed. The next step is to colour the surfaces, which may be visible. The intensity of the colour of the surfaces is determined by the cosine of the angle between the normal vector of the surface and the ray from the light source on the surface (see for shading models Foley and van Dam [2]). After this a hidden surface removal is performed using a z-buffer algorithm [2] After this the surfaces are ready to be visualized by the 3D GKS system; all projections are done by 3D GKS.
Examples of 3D fractals, generated with the 3D extension of the fractal system, are shown in fig. 16[1] and fig. 17.

2.5 Fractal dimension and self-similarity

So far all "objects" created by the fractal system were called fractals. It is rather doubtful whether all "objects" satisfy the definitions mentioned in the introduction. The fractal dimension (D) may be calculated for a complete self-similar set made up of N equal sides of length r. (r is the similarity ratio between the length of a side of a fractal approximant and the preceding fractal approximant) using the formula from Mandelbrot [13] :

1) $D = \log N \,/\, log\,(1/r\ )$

This formula may be applied for the objects defined by the production rules described in, for example, fig. 1 (D = 3/2); fig. 5 (D = log 7 /log 3), when the fractal dimension of the complete fractal is considered; fig. 9 (D = log 3 /log 2); fig. 13 (D = 2).
The set described in fig. 3 is nearly self-similar, because in addition to the ramifying parts it includes a trunk. For this set, which is not composed of equal sides, the fractal dimension should be calculated in a different way. For a mathematical description of ramifying fractals, which are basically the same as Pythagoras trees, see Lauwerier [11] .
The fractal dimension can be determined analytically in a few special cases only. A method for estimating D is given by Mandelbrot [13] (see also: [17]). D of fractals in a plane can be estimated using the following method: Let ε be the length of the side of a small square and cover the set by a grid of such meshes. If $N(\varepsilon)$ is the number of squares needed for covering the set, then D can be calculated using formula 2:

2) $$D = \lim_{\varepsilon \downarrow 0} \frac{\log N(\varepsilon)}{\log 1/\varepsilon}$$

With the fractal system objects may be created, which satisfy the similarity condition, but do not have a fractal dimension. In fig. 18 the production rule of a spiral object shown in fig. 19 is given. This object is

1. Fig. 16 and fig. 17 are coloured photographs and are not shown in this report.

obviously not of a fractal dimension.
Self-similarity may be easily disturbed in the objects created by the fractal system, by using randomizing functions (see for example fig. 4). Objects generated in this way do not completely satisfy the self-similarity condition. In these cases, when the object is modified randomly during the generation, D must be determined in the experimental way. In [3] a definition is given of the dimension of statistically self-similar sets.
Many different objects may have an equal fractal dimension. In applications a fractal dimension is only useful in comparing objects from the same class. An example of this can be given from the class of ramifying fractals. When all proportions in the branches (see fig. 3) remain constant except for the value of r, which varies between 0.5 and 1.0, a series of fractals may be generated with an increasing D (see fig 20). In this example the ramifying fractal becomes more plane-filling, D may be used as a measure of plane-filling properties of the fractal. In other examples it is possible to use D as a measure of irregularity or fragmentation (in the class of seeding fractals).

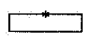
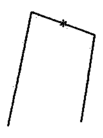
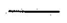
initiator	generator	base element
		

Fig. 18: Production rule for a spiral, the resulting spiral is shown in fig. 19.

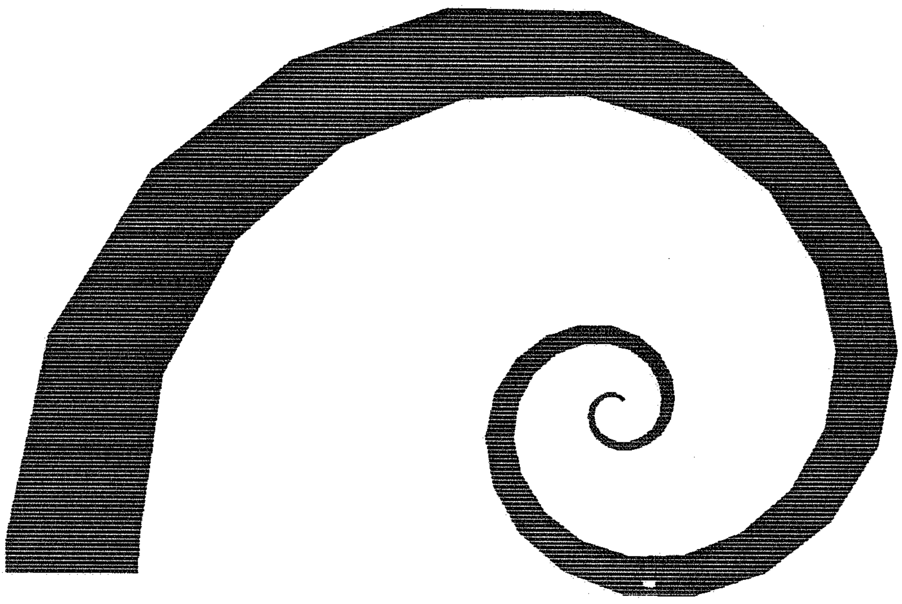


Fig. 19: Spiral resulting from the production rule shown in fig. 18.

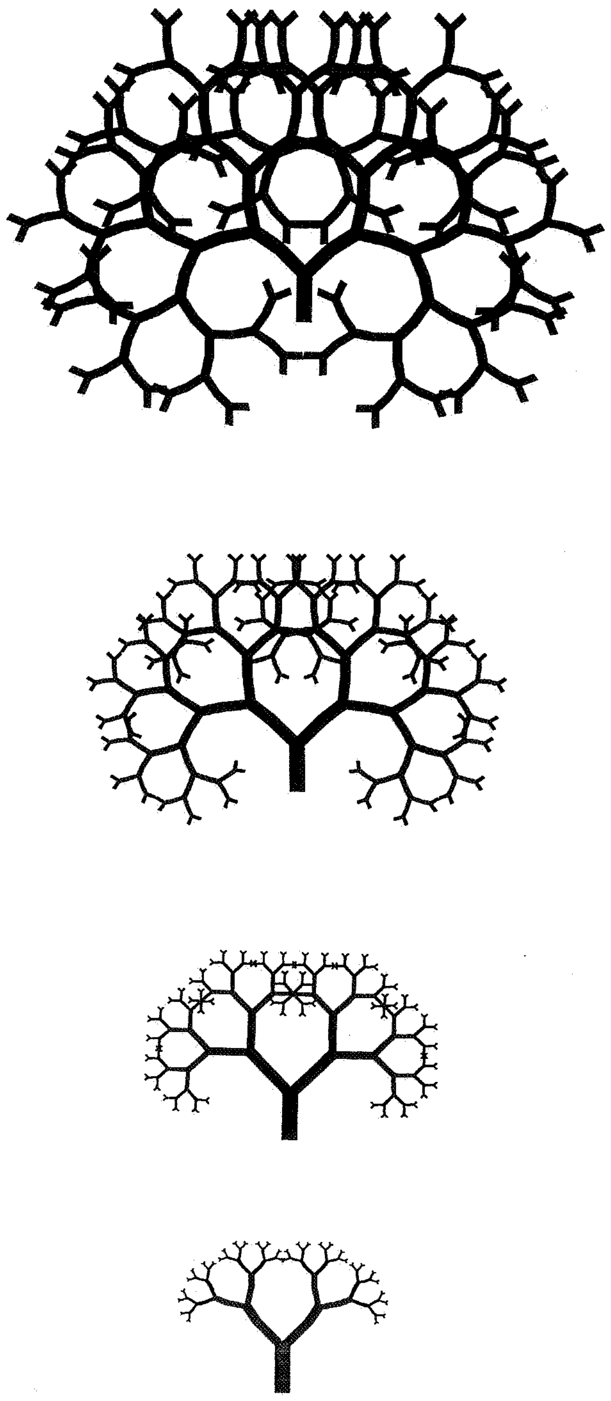


Fig. 20: Series of ramifying fractals in which D is increasing from the right to the left. On the right D is almost 1.0, at the left D is nearly 2.0. The production rule for those fractals is related to the one shown in fig. 3.

To summarize fractals and other objects generated by the fractal system may be characterized by their mathematical properties:
a) The fractal dimension and the topological dimension.
b) The degree of self-similarity and the transformations like dilation, rotation, translation and reflection; to attain this self-similarity.
In addition those objects are characterized by a production rule necessary for generating the object and the data structures for representing it.

3. Conclusions

With the fractal system presented in this paper it is possible to create a large number of self-similar sets. The data structures necessary for representing different fractal objects are put together in three base data structures. These data structures, combined with an algorithm suitable for several different types of fractals, offers also new combinations, like self-reproducing fractals. One consequence of these attempts to cover a large class of fractals is that the algorithm becomes less efficient for more "primitive" fractals, as for example the quadric Koch curves.
Fractals may be stored in an efficient way using the parameters from the 3 components of the production rule. In this way a library of fractals can be created. These library files may form a useful tool for entering production rules from outside the fractal system.

4. Future plans

The purpose of the fractal system is to develop a tool for modelling objects from nature. One important aim in the future is to make the system suitable for developing mathematical models for biological objects.

Acknowledgements

Drs. P.J.W. ten Hagen (Centre for mathematics and Computer Science (Amsterdam), department of Interactive Systems); prof. dr. H.A. Lauwerier and dr. J.B.T.M. Roerdink (both from the department of Applied Mathematics of the Centre for mathematics and Computer Science) are thanked for reviewing the first drafts of this paper.

References

1. F.M. Dekking (1982). Recurrent sets, *Advances in Mathematics*, 44, pp 78-104.
2. J.D. Foley and A. van Dam (1982). *Fundamentals of interactive computer graphics*, 664pp, Addison-Wesley publishing company, London.
3. K.J. Falconer (1985). *The geometry of fractal sets*, 162pp, Camebridge University Press, Cambridge.
4. M.F. Barnsley and S.G. Demko (1986). *Chaotic dynamics and fractals*, 292pp, Academic Press, New York,.
5. P.J.W. ten Hagen and M.M. de Ruiter (1986). *C-GKS the C implementation of GKS, the graphical Kernel System user guide*, 88pp, Centre for Mathematics and Computer science, Amsterdam.
6. S. Demko, L. Hodges, and B. Naylor (1985). Construction of fractal objects with iterated function systems, *Computer Graphics*, 19.3, pp 271-278, SIGGRAPH '85 proceedings..
7. Farmer, Kaplan, and Yorke (1983). The dimension of chaotic attractors, *Physica 7D*, pp 153-180.
8. M. Hata (1985). On the structure of self-similar sets, *Japan J. Appl. Math.*, 2, pp 381-414.
9. F.R.A. Hopgood, D.A. Duce, J.R. Gallop, and D.C. Sutcliffe (1983). *Introduction to the Graphical Kernel system (GKS), Academic*, 200pp, Academic Press, London.
10. ISO (1986). *Information processing systems - computer graphics- Graphical Kernel System for three dimensions (GKS-3D) functional description*, ISO/TC 97/SC21/DP8605.
11. H.A. Lauwerier (1985). The Pythagoras tree as a Julia set, *CWI Newsletter*, 6, pp 2-18.
12. M. Lewis and D.C. Rees (1985). Fractal surfaces of proteins, *Science*, 230, pp 1163-1165.
13. B.B. Mandelbrot (1982). *The fractal geometry of nature*, 460pp, Freeman, San Francisco.
14. P. Meakin (1986). A new model for biological pattern formation, *J. theor. Biol.*, 118, pp 101-113.
15. D.R. Morse, J.H. Lawton, M.M. Dodson, and M.H. Williamson (1985). Fractal dimension of vegetation and the distribution of arthropod body length, *Nature*, 314, pp 731-733.
16. J. Nittman, G. Daccord, and G. H.E. Stanley (1985). Fractal growth of viscous fingers: quantitative charararacterization of a fluid instability phenomenon, *Nature*, 314, pp 141-144.
17. C.A. Pickover (1986). A Monte Carlo approach for ε placement in fractal-dimension calculations for

waveform graphs, *Computer Graphics Forum*, 5, pp 203-210.

18. A.R. SMITH (1984). Plants, fractals and formal languages, *Computer graphics*, 18.3, pp 1-10, SIGGRAPH '84 proceedings.
19. D.L. TURCOTTE, R.F. SMALLEY JR, and S.A. SOLLA (1985). Collapse of loaded fractal trees, *Nature*, 313, pp 671-672.