Transactions on Information and Communications Technologies vol 5, © 1993 WIT Press, www.witpress.com, ISSN 1743-3517

Invited Paper

Visualization and generation of iterated function systems

H. Jones,[a] A. Campa[b]

[a]*School of Mathematics, Statistics and Computing, Middlesex University, Bounds Green Road, London N11 2NQ, UK*

[b]*Centre for Advanced Studies in Computer Aided Art and Design, Middlesex University, Cat Hill, Barnet, EN4 8HT, UK*

ABSTRACT

Iterated Function Systems (IFSs) have been developed by Michael Barnsley for the generation of a class of fractal objects. The authors show how IFSs can also be used to generate three dimensional solid objects. A method for visualizing such systems is presented, using the standard Computer Graphics depth cueing and z-buffer algorithms. The method has potential for adding naturalistic features, such as trees, shrubs and clouds, to architectural drawings and some of the forms generated could be inspirational to architects and engineers in the generation of innovative structures.

ITERATED FUNCTION SYSTEMS

Iterated Function Systems (IFSs) were developed by Michael Barnsley [BARNSLEY88] and have been used by him to illustrate organic structures such as the ferns and leaves; figure 1 shows a leaf (after Barnsley) generated using an IFS. A feature of such systems is that complex structures can be generated using a straightforward algorithm applied to relatively simple data sets, giving major potential for compression of the data required for description of visual images. This feature is currently being investigated by Barnsley and his colleagues as a potential method for compression of High Density Television (HDTV) images, enabling more efficient transmission of such images in a narrower bandwidth than that normally used.

Most applications of IFSs generate images within a two dimensional space, generating objects that have fractal dimension between 1 and 2 [MANDELBROT82]. The authors have extended and illustrated a set of fractal objects lying in three dimensional space, based on the Platonic solids [JONES91b]. The Sierpinski tetrahedron and the Menger sponge were already well known, but the method of illustration using IFSs is novel and gives more correct illustrations of such fractal objects than the methods normally used.

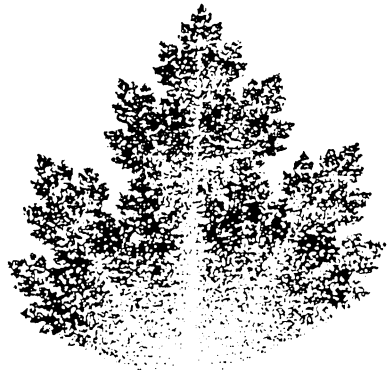

Figure 1. A leaf drawn using a four transform Iterated Function System

An Iterated Function System is defined by a set of n contractive affine transformations $\{T_1, T_2, \ldots T_n\}$ applied to some space and n associated probabilities $\{\pi_1, \pi_2, \ldots \pi_n\}$, such that the sum of the probabilities is unity,

$$\sum_{i=1}^{n} \pi_i = 1.$$

By definition all the π_i are non-negative as they represent probabilities.

An affine transformation is one in which lines are transformed into lines, parallel lines and only parallel lines are transformed into parallel lines [SELKIRK91]. Affine transformations can be created as combinations of the three standard Computer Graphics transformations of translation, scaling and rotation, which are implemented in most Computer Graphics systems using matrix operations on homogeneous coordinate representations of space [FOLEY90][FERRANTE91]. A contractive transformation diminishes the area of a two dimensional object or the volume of a three dimensional object. This can be tested by showing that the determinant of the square matrix used to represent the transformation has value less than one.

There are two major methods for depiction of an IFS [BARNSLEY88]. One involves recursive application of the set of transformations to a specified region of the plane. The points remaining after many iterations of the transformations fall into the attractor of the IFS, for example the leaf shape of figure 1. Different forms of attractor result from different sets of transformations. Interactive methods have been developed to enable designers empirically to change transformations to produce different effects in the image of the attractor [HORN91]. Barnsley describes how 'tiling' of fractal structures can be used to generate the required transformations. For most developers, the generation of transformations to produce particular shapes is an 'art'; Barnsley's research is to make it into a science, so that the process can be automated. This will enable the rapid algorithmic reduction of complex image data into a compact set of transformations.

The second form of depiction is the one that will be used in this study. The simple algorithm for such depiction is given below (algorithm 1). This generates an apparently random sequence of points, P_0, P_1, P_2, ..., which is attracted towards the shape defined by the IFS. If the first point chosen, P_0, is not in the attractor defined by the IFS, the first few points plotted will lie outside the attractor. However, convergence is usually rapid and this problem can be overcome by not plotting the first few points generated, say the first 20. The overall number of points needed for a reasonable result will generally depend on the precision of the display surface. With a medium precision VDU, no added detail should be noticed after about 20 000 points. If the probability π_i associated with the transformation T_i is proportional to the contraction ratio of the transformation (the area into which a unit area is transformed under T_i), the random scatter of points produced will have uniform density with regard to the different transformations throughout the image generation. However, if generation continues for a very large number of iterations with probabilities π_i being otherwise assigned, the whole of the attractor will eventually be visited, although the density of depiction will be uneven for the early stages of the image generation.

Given a set of transformations $\{T_1, T_2, \ldots T_n\}$ and a set of probabilities $\{\pi_1, \pi_2, \ldots \pi_n\}$;

Define an initial point P_0;

For k = 1 to a large number

Select a transformation T_i with probability π_i;

Apply the transformation T_i to point P_{k-1} to create point P_k;

Plot P_k;

End k loop.

Algorithm 1: A general IFS plotting algorithm

A particularly straightforward family of IFSs uses transformations consisting only of translations and uniform scalings with factor $\frac{1}{2}$ (this means the same scale factor is applied to all Cartesian coordinates). Such scalings can be represented as projections with scaling $\frac{1}{2}$ towards a given centre of projection. Figure 3 illustrates the affine transformation that shrinks the square ABCV to half size so that the vertex V is unchanged. This can be performed by applying the formula

$$P' = \tfrac{1}{2}(P + V)$$

to any point P in the original shape, generating the 'image' of P under the transformation as P'. Here, P', P and V are taken to be position vector representations of the points concerned, so that the formula should be applied separately to all Cartesian coordinates of these points.

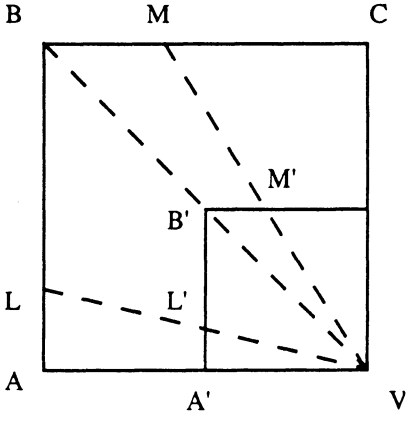

Figure 2. Effect of half scale shrinking of a square towards one of its vertices

The algorithm for generating such IFSs can now be reduced to the following form, called the 'chaos game' by Barnsley.

Given a set of vertices $\{V_1, V_2, \ldots V_n\}$;

Define an initial point P_0;

For k = 1 to a large number

Select a vertex V_i with probability $\frac{1}{n}$;

Set $P_k = \frac{1}{2}(P_{k-1} + V_i)$;

Plot P_k;

End k loop.

Algorithm 2: The 'chaos game' algorithm

If this algorithm is applied to the vertices V_1, V_2 and V_3 of an isosceles triangle, the result is as in figure 3, generating the 'classical' fractal shape of the Sierpinski triangle or gasket [SIERPINSKI16].

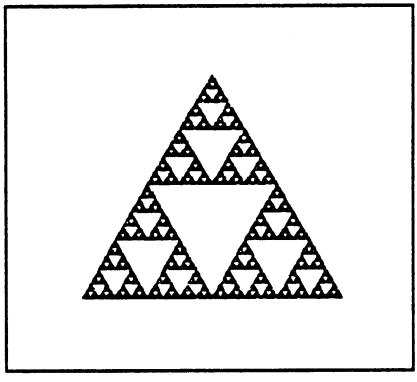

Figure 3. Chaos game depiction of a Sierpinski triangle

Sierpinski's original paper describes two ways of producing this shape. One involves an infinite recursion, drawing smaller and smaller triangles, the other fills the shape with a curve whose length approaches infinity and has nowhere a well defined tangent, as it is generated by an infinite set of zero length line segments. Both these methods must be drawn to a finite precision and they impose, at their limited precision, the shapes of triangles and straight lines where there should be none. The IFS method is intrinsically better for the representation of such fractal forms, as it imposes no superfluous information other than the shape of a pixel, the basic drawing element of a display surface. Thus the depiction is automatically to the greatest precision possible on the display surface used. Lines are seen in figure 3 only as the result of generating a scatter of points that happen to lie in the line. Completeness of the depiction depends on the number of repetitions used in the drawing algorithm; after a certain number of repetitions, all possible pixels that lie within the shape will have been visited and no further change to the image will be observed.

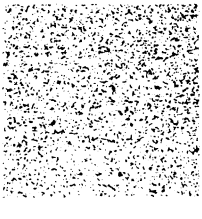

Figure 4. The 'chaos game' algorithm applied to the vertices of a square

Jones [JONES91a] has illustrated that the same algorithm, when applied to the four vertices of a square, evenly fills the square with a uniformly scattered set of random points (figure 4), as proved by Jarrett [JARRETT90]. This shows that an IFS can be used to 'fill' regions of two dimensional space.

Given a set of vertices $\{V_1, V_2, \ldots V_n\}$ and a shrink factor f;

Define an initial point P_0;

For k = 1 to a large number

Select a vertex V_i with probability $\frac{1}{n}$;

Set $P_k = fP_{k-1} + (1 - f)V_i$;

Plot P_k;

End k loop.

Algorithm 3. The adapted chaos game algorithm

Figure 5 shows the effect of applying algorithm 3 to the vertices of a regular pentagon and to a regular hexagon. In the case of the pentagon, the shrink factor is set to $\frac{1}{2+\rho}$, where ρ is the 'golden ratio' [CUNDY61], $\rho = \frac{1+\sqrt{5}}{2}$. For the hexagon, the shrink factor is $\frac{1}{3}$. These factors are calculated using simple trigonometry to enable the subportions of the figures to touch, but not to overlap.

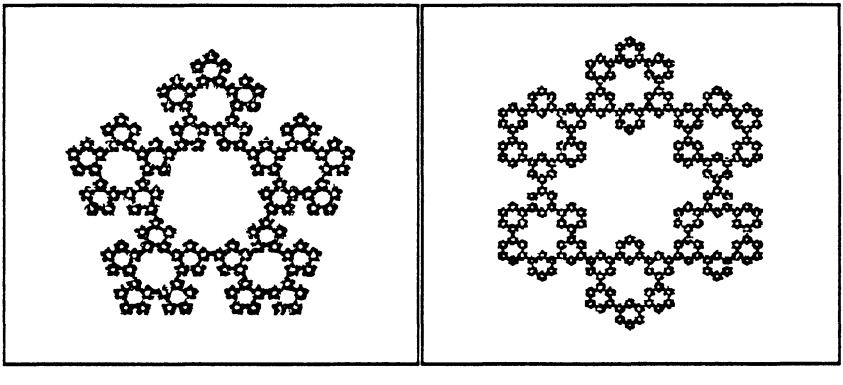

Figure 5. Algorithm 3 applied to a regular pentagon and hexagon.

ITERATED FUNCTION SYSTEMS APPLIED TO SETS OF THREE DIMENSIONAL VERTICES

Without changing the algorithm, the vertices and points generated can be interpreted as three dimensional Cartesian coordinate triads, (x, y, z), with formulae being applied to each of these coordinates in turn. Visualization of objects generated in this way can be enabled by using two algorithms commonly used in Computer Graphics, depth cueing and z-buffering [FOLEY90].

In many Computer Graphic systems, an intermediate stage of generating images of objects placed in a three dimensional Cartesian universe redefines the Cartesian space so that the origin is at the viewing position (equivalent to the eye of an observer or the lens of a notional camera). The z-coordinate of this 'view coordinate system' points away from the observer towards the scene to be viewed and thus represents the 'depth' of a point from the observer. Depth cueing shades points or lines according to this depth. To generate an image of an IFS using one of algorithms 1, 2 or 3 applied to a three dimensional set of vertices, points closer to the observer are given high colour intensity, the intensity diminishing for points further away. This device enables a real observer to distinguish the depths of points in the illustration generated. The authors have set the highest possible intensity of colour allowed on the display surface for points in the objects generated closest to the observer, diminishing linearly to the lowest possible intensity for points in the object furthest from the observer. This gives the largest possible intensity difference within the object's depth.

Images are synthesised on the 'raster' device of a graphics display screen. This contains a regular grid of 'pixels', or picture elements, each of which can be allocated a colour value and intensity. Projection from the three dimensional description of a point to the equivalent pixel location that will represent that point forms part of the process of image generation. A point known to occupy a particular pixel position should only be displayed if the pixel does not currently illustrate a point closer to the observer. Use of a z-buffer enables this check. A z-buffer is a dedicated section of computer memory containing one real valued variable for each pixel of the display device. A memory location in a z-buffer holds the depth (or z-coordinate value in view coordinates) of the object currently illustrated in its pixel. The z-buffer is initiated to hold a large 'background' z distance, beyond which nothing will be depicted. As elements are added to the image on the raster display, values in the z-buffer are upgraded to keep pace with the image. A new point generated is not drawn if its pixel address is already occupied by a point closer to the view position (one with lower z value). Otherwise, the new point is displayed, overwriting the previous occupant of the pixel, with the correct depth cued intensity, and the z-buffer is adjusted accordingly to contain the z-depth of the new point.

This technique has been applied to the vertices of regular polyhedra (or Platonic solids) [CUNDY61] to generate another family of fractals [JONES91b][JONES91c]. Shrink factors are calculated so that the shrunken versions of the object concerned will just touch each other when fitted inside the original object. Considerations of solid geometry give shrink factors of $\frac{1}{2}$ for the tetrahedron (figure 6) and octahedron (figure 7), $\frac{1}{(2+\rho)}$ for the dodecahedron

(figure 8) and $\frac{1}{(1+\rho)}$ for the icosahedron (figure 9), where ρ is the 'golden ratio' defined above.

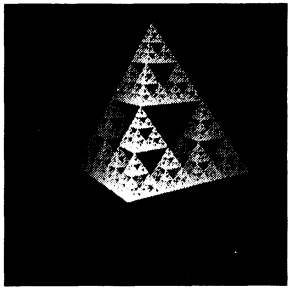

Figure 6. A Sierpinski tetrahedron illustrated using an IFS

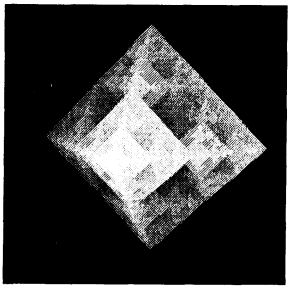

Figure 7. An octahedral based fractal , generated using an IFS

Note that the tetrahedron generated is the classical Sierpinski tetrahedron. Once more, the method of generation and depiction used here is purer than the normally used methods of recursively redrawing small tetrahedra. Recursive redrawing of tetrahedra imposes surface normals on the fractal object where none should exist, unless the subdivision is continued so that the sub-polyhedra depicted occupy less than the space of a single pixel in image space. The adapted chaos game method shows the object to the full precision of the display device without imposing any unnecessary and invalid surface information. The octahedral based form has a complex double-pyramid structure. The other forms illustrated have almost organic, coral like forms.

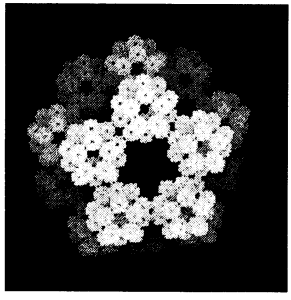

Figure 8. A dodecahedral based fractal, generated by an IFS

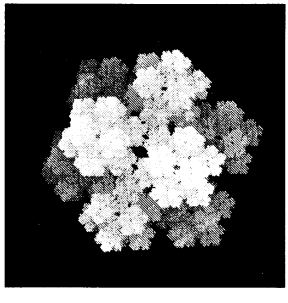

Figure 9. An icosahedral based fractal, generated by an IFS

Further interesting figures have been generated by taking narrow sections through this collection of objects, producing fractal patterns sometimes displaying several forms of symmetries, otherwise with some non-symmetric but self similar patterns. Figure 10 shows a cross section through an icosahedral based fractal that displays pentagonal symmetry. Other slices through the same object display different forms of symmetries. The lay outs of such patterns could, perhaps, have desirable properties for town planners; the objects based in three dimensions could give architects and designers a new source of potential shapes to explore.

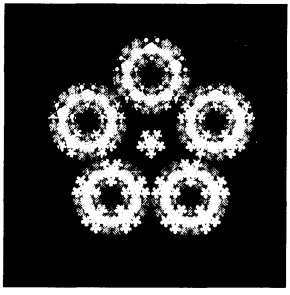

Figure10. A slice through an icosahedral fractal

The algorithm does not generate a fractal object from the vertices of a cube or hexahedron when the natural shrink factor of $\frac{1}{2}$ is applied, as the points generated evenly fill that original cube, in a manner analogous to the filling of a square described above. Different cube-based distributions of vertices with different shrink factors can, however, produce interesting effects. An IFS with 32 vertices based at the eight corners of a cube and at $\frac{1}{3}$ and $\frac{2}{3}$ distances along the cube's twelve edges with shrink factor of $\frac{1}{3}$ generates a fractal object similar to the classical Menger sponge, [MANDELBROT82] (figure 11).

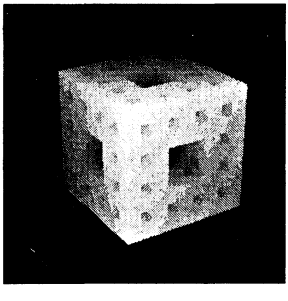

Figure 11. A new cube based fractal, the 'Jones/Campa cake'

This new form has been named the 'Jones/Campa cake' by John Lansdown [LANSDOWN91]. The Menger sponge itself can be generated by using the same shrink factor of $\frac{1}{3}$, but with 20 vertices at the cube's eight corners and at the mid points of each of its twelve edges [figure 12].

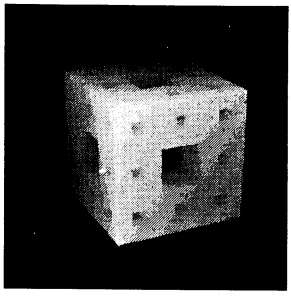

Figure 12. The Menger sponge illustrated using an IFS

In theory, the block shapes so formed have zero overall volume. It would be interesting to analyse if a limited precision approximation of these structures could produce light but strong building blocks. There is already an analogy of a light, but strong, structure in the similarity of the Sierpinski tetrahedron (figure 6) to the 'pylons' that carry high tension electrical transmission lines. A similar 'pylon' shape can be generated by applying a shrink factor of $\frac{1}{2}$ to the vertices of a square based pyramid. Could such a shape be of interest to structural engineers? Do the fractal cube structures illustrated have potential as the basis for building designs? The 'Défense' development in Paris shows that such structures are not totally outrageous for modern architectural development.

It has been demonstrated above that IFSs can be used to generate filled regions such as squares in two dimensions and cubes in three dimensions. Another organisation of vertices can be used to generate an apparently solid tetrahedron. Using a shrink factor of $\frac{1}{2}$ on eight vertices consisting of the corners and edge mid-points of a regular tetrahedron gives the image of figure 13. In this

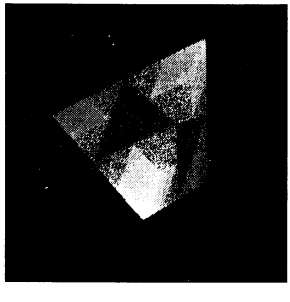

Figure 13. An apparently solid tetrahedron generated using an IFS

figure, points generated by attraction towards a corner vertex were coloured in yellow with points generated by attraction towards an edge mid-point vertex were coloured red, showing up as different tones in the monochrome reproduction shown here. The exterior surface of the tetrahedron is filled, although if the object generated is sliced through, interior holes are found. Combinations of cubes, tetrahedra and other IFSs of 'solid' appearance could, conceivably, be used to illustrate models of more complex solid objects. The combination of depth-cueing and z-buffer algorithms gives a relatively simple rendering method for such shapes.

It is worth considering the implications of the rendering method used. No surface information is presented to the observer. The objects generated are illustrated as clusters of points scattered on a two dimensional surface. The only form of shading is to vary the intensities of the shading of the points to match a concept of distance from an observer. There is no attempt to reproduce such lighting features as inverse square laws. However, the images are interpreted as having surfaces as in figure 13. This feature is also evident in fractal images based on the tetrahedron (figure 6), octahedron (figure 7) and cube (figures 11 and 12), where co-planar features can be distinguished. It is a matter of fascination to the authors that the eye and brain interpret the images as objects existing within a three dimensional space on such relatively minor clues.

The half-toned figures illustrated above are monochrome reproductions of images from a Silicon Graphics Personal Iris Workstation, with the capability of displaying up to 16.5 million colours. The process does not depend on equipment of this degree of sophistication to deliver useful results. Figure 14 gives an example of an IFS generated on a PC by David Scott, with a limited range of colours [SCOTT92]. This fern differs from those normally shown in

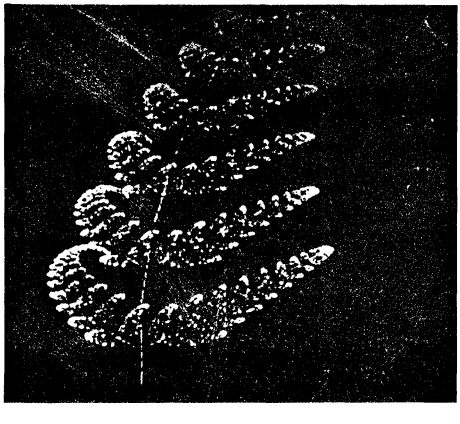

Figure 14. A three dimensional IFS fern

that it is a representation of a three dimensional structure based on a three dimensional IFS. Choosing different viewing directions will generate different views of the fern. On more limited equipment, a little subtlety in development of the display method can do away with the need for a z-buffer, which may be too costly in storage space for equipment with limited memory resources. Each pixel

holds implicit information on the depth of the point currently illustrated in that pixel, as the depth cueing has tied the intensity of pixel illumination to the z-depth. Using a 'read pixel' command to compare the current pixel intensity to that which would be allocated to a new point can enable the decision whether the new point should be displayed, or whether it is behind (less intense than) the point currently illustrated within the pixel.

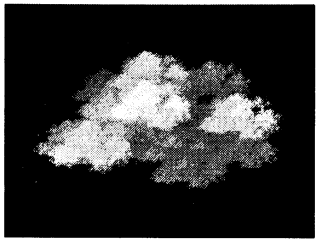

Figure 15. An IFS generated three dimensional cloud

Other natural forms such as clouds [LUCKINS92] and trees [SCOTT92] have been successfully illustrated by this method on devices with limited memory and limited display capacity. An example of an IFS generated cloud is shown in figure 15, although this picture used the full capabilities of the advanced Workstation cited above. Here the vertices used for algorithm 3 are scattered within an ellipsoid and the value of f is empirically determined to give a satisfactory image. The cloud generated here is a representation of a set of particles suspended in three dimensional space and is therefore different from the majority of computer generated cloud simulations [SAUPE88], which are shown as painted objects on a two dimensional backdrop sky. This form of illustration could be used in flight simulator systems, giving a flight officer an experience similar to that of flight through a cloud.

SUMMARY

The concept of generation of Iterated Function Systems has been developed, mainly within the context of a limited sub-class of such systems. A relatively straightforward method of visualizing IFSs that are based within three dimensional Cartesian space on a two dimensional display surface has been discussed. Although no real surface information is given in this method, images generated are interpreted by observers as having spatial properties such as co-planar sets of points and even planar surfaces. These even allow the depiction of apparently solid objects and could be used for generation of images of a wide class of objects, including realistic simulations of naturally occurring phenomena. The new fractal forms generated through IFSs could be inspirational to designers. IFSs have the property of encapsulating the forms of complex objects in a relatively limited data set and a simple reconstruction algorithm.

The monochrome images displayed in this publication do not do justice to the precision of the full colour images generated on the authors' workstation and displayed as colour slides.

REFERENCES

[BARNSLEY88] Barnsley, M.F., *Fractals Everywhere,* Academic Press, San Diego, 1988

[CUNDY61] Cundy, H.M. and Rollett, A.P., *Mathematical Models* (2 ed), Oxford University Press, Oxford, 1961

[FERRANTE91] Ferrante, A.J., Moreira, L.F.R., Boggio Videla, J.M. and Montagu, A., *Computer Graphics for Engineers and Architects*, Elsevier/Computational Mechanics Publications, Amsterdam, 1991

[FOLEY90] Foley, J.D., vanDam, A., Feiner, S.K. and Hughes J.F., *Computer Graphics, Principles and Practice* (2 ed), Addison Wesley, Reading, Mass, 1990

[HORN91] Horn A.N., *IFSs and the Interactive Design of Tiling Structures*, in Fractals and Chaos, Crilly A., Earnshaw R. and Jones H. (eds), 119 – 144, Springer-Verlag, New York, 1991

[JARRETT90] Jarrett, D., Personal correspondence, Middlesex University, 1990

[JONES91a] Jones, H., *Dürer, Gaskets and the Chaos Game,* Computer Graphics Forum, 9 (3), 327 – 332, 1991

[JONES91b] Jones, H. and Campa, A., *Fractals Based on Regular Polygons and Polyhedra*, in Scientific Visualization of Physical Phenomena, Patrikalakis, N.M. (ed), 299 – 314, Springer-Verlag, Tokyo, 1991

[JONES91c] Jones, H. and Campa, A., *Fractal Polyhedra – New Shapes from Old Figures*, in Compugraphics'91, Santo, H.P. (ed), 47 – 56, Conference Proceedings, Sesimbra, Portugal, September 1991

[LANSDOWN91] Lansdown, R.J., *Serendipity, Not only computing – also art*, The Computer Bulletin, Vol 3, Part 7, 12 – 13, September 1991

[LUCKINS92] Luckins, S., *Three Dimensional Cloud Modelling Using the Chaos Game*, MSc Applied Computing Technology Dissertation, Middlesex University, 1992

[MANDELBROT82] Mandelbrot, B.B., *The Fractal Geometry of Nature*, W.H.Freeman, 1982

[SAUPE88] Saupe, D., *Algorithms for Random Fractals*, in Peitgen, H.-O. and Saupe, D. (eds), The Science of Fractal Images, Springer-Verlag, New York, 1988

[SCOTT92] Scott, D. *Three Dimensional IFS Generation and Rendering of Botanical Images*,MSc Applied Computing Technology Dissertation, Middlesex University, 1992

[SELKIRK91] Selkirk, K., *Longman Mathematics Handbook*, 236, Longman, Harlow, 1991

[SIERPINSKI16] Sierpinski, W., *Sur Une Courbe dont tout point est un point de ramification*, in W.Sierpinski: Oeuvres Choisis, Tome II, PWN - Polish Scientific Publishers, Warsaw, Poland, 1975, first published 1916