International Journal of Soft Computing 11 (3): 107-113, 2016
ISSN: 1816-9503
© Medwell Journals, 2016

On the Study of 3D Fractals

[1]Bulusu Rama and [2]Jibitesh Mishra
[1]Department of Computer Science and Engineering, MLR Institute of Technology, Hyderabad, India
[2]Department of Computer Science and Applications,
College of Engineering and Technology BPUT, Bhubaneswar, India

Abstract: Fractals provide an innovative method for generating 3D images of real-world objects by using computational modeling algorithms based on the imperatives of self-similarity, scale invariance and dimensionality. Images such as coastlines, terrains, clouds, mountains and most interestingly random shapes composed of curves, sets of curves, etc., present a multi-varied spectrum of fractals usage in domains ranging from multi-colored, multi-patterned fractal landscapes of natural geographic entities, image compression to even modeling of molecular ecosystems. Fractal geometry provides a basis for modeling the infinite detail found in nature. Many different types of fractals have come into limelight since their origin.

Key words: Fractals, 3D Images, Mandelbrot set, Mandelbulb, Julia set, Sierpinski gasket, 3D rendering, IFS

INTRODUCTION

Today fractal geometry is completely new area of research in the field of computer science and engineering. It has wide range of applications. Fractals in nature are so complicated and irregular that it is hopeless to model them by simply using classical geometry objects. A fractal is a rough or fragmented geometric shape that canbe subdivided into parts, each of which is (at least approximately) a reduced size copy of the whole or in otherwords is self-similar when compared with respect to the original shape. The term was coined by Benoit Mandelbrot in 1975 and was derived from the Latin word "fractus" meaning"broken" or "fractional".

Formally, the fractals are categorized in two types i.e., regular (geometric) and random fractals. Regular fractals consist of large and small structures that are exact copies of each other, except in size. One of the more well-known regular fractals is the Koch snowflake which is made up of small triangles added to the sides of larger triangles to an infinite degree. Random fractals are more apparent in nature as their small scale structures may differ in detail. It was this type of pattern that greatly influenced Mandelbrot, who gave these patterns the name fractal, from the Latin word fractus.The primary characteristic propertiesof fractals are self-similarity, scale invariance and gener alirregularity in shape due to which they tend to have asignificant detail even after magnification-the more the magnification the more the detail. In most cases, afractal canbe generated by a repeating pattern constructed by a recursiveor iterative process. Natural fractals possess statistical self-similarity whereas regular fractals such as Sierpinski gasket, Cantor set or Koch curve contain exact self-similarity. "Clouds are not spheres, mountains are not cones, coastlines are not circles and bark is not smooth, nor does lightning travel in a straight line" (Mandelbrot, 1982).

Natural phenomena of fractals: Many objects in the nature can be created by applying the concept of classical geometry like-lines, circles, conic sections, polygons, spheres, quadratic surface and so on. There are various objects of nature which cannot be modeled by applying Euclidean geometry hence, there is need to deal with such complicated and irregular object which can only be constructed by fractal geometry. To generate such complicated object iteration process is required which is called iterated function system.

MATERIALS AND METHODS

Process of generation: We present here the process to convert 2D fractals to 3D and give an example as to how 2D has been converted to 3D in the case of Mandelbulb (Rama and Mishra, 2011) and the cube based Sierpinski's gasket (Bulusu, 2012).

Process of generation of 2D fractals to 3D: The steps used in the process of generation of 2D fractals to 3D are as follows: For a given maximum number of iterations

Corresponding Author: Bulusu Rama, Department of Computer Science and Engineering, MLR Institute of Technology, Hyderabad, India

denotedby the constant maxit, plot the set defined by the (min, max) range of values of each point the axes (Xmin, Xmax), (Ymin, Ymax)

This is done by first generating a grid based on theline spacing values for each Xz and Yz. The linespacing is calculated using the distance of each suc hpoint based on (Xmin, Xmax) and (Ymin, Ymax), set tolower and upper limits from the origin of z, whos einitial value is always 0. For each co-ordinate axisinvolved, we get a line spacing value, e.g., Xline spacing, Yline spacing etc.

Then the actual fractal is plotted by selecting all points in the grid obtained from the step above and checking if it qualifies as a candidate for being part of the fractal. If the difference between these twoline spacing values is 0, then the corresponding point be longs to the fractal being generated. And the union of all such points gives the domain of the resulting fractal set.

The next step is to plot the actual fractal set based on the set of points obtained from step 3.This is done by iterating the above steps based on the max number of iterations (this is now an inputvariable) and the (Xmin, Xmax), (Ymin, Ymax) ranges, till all the iterations are complete. However, as uper limit for the maximum number of iteration sbeing input is fixed at value greater than maxit, toensure that the generated fractal doesn't blowup into an infinite space on the screen.

The output: The program runs as a Windows console applicationwith the 3D image of the fractal (Rama and Mishra, 2010) beingrendered as a colour-mapped image projected on to the 3D plane, using Matlab 3D Image Rendering Software.The use of Open G L algorithms and the standard Graphics library makes the implementation of the routines for construction of the initial grid and the subsequent fractal as also the colour-coding and 3D rendering very flexible and efficient.

Generation of the mandelbulb: The 3D Mandelbulb is generated by taking results obtained by applying above described method based on the defined parameters and then subsequent rendering of each image output in 3D using Matlab Software. The Mandelbulb is obtained by using an nth power of a 3D hyper-complex number, atrue 3D version of the same can be obtained and is often referredto as the Mandelbulb. This is obtained by using a rotationtrans formation away from the z-axis. The 3D Mandelbulb is generated using the iteration $z = z^n+c$ where z and c are 3-dimensional hyper-complex numbers with the power mapping $z \dashrightarrow z^n$, defined as stated above. For n>3, the result is a 3-dimensional bulb with a fractal

Fig. 1: 'True' 3D simulation of the Mandelbrot Setgenerated using n = 2 as part of the hyper-complexdimensional rotation around the z-axis (Rz (-θ)

Fig. 2: 'True" 3D simulation of the mandelbrot setgenerated using Rz (θ) as the hyper-complex dimensional rotation around the z-axis

surface and a number of protruding "lobes" that dependent on the parameter n. Multiple graphic renderings can be generated using $n \geq 2$.

The output: The program runs as a windows console application with the 3D image of the Mandelbulb being rendered as a colourmapped image projected onto the 3D plane (Rama and Mishra, 2011), using MATLAB 3D Image Rendering Software. The first 3D fractal of Mandelbulb is obtained by using a rotational transformation of Rz (-θ) as depicted in Fig. 1. The second one is obtained by a positive angle around the φ rotation. This gives a negative z-componenta round Rz (θ) resulting the fractal in Fig. 2. The use of Open GL algorithms and the standard

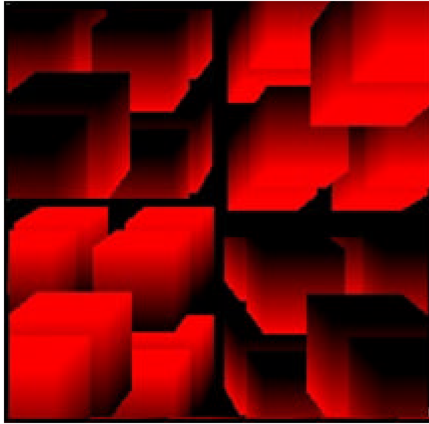

Fig. 3: The 3D Sierpinski gasket (Cube-based) having depth = 3

graphics library makes the implementation of the routines for construction of the initial grid and the subsequent Mandelbrot set as also the colourcoding and 3D rendering very flexible and efficient. Additional details and other techniques to generate the 3D Mandelbrot Set can be found in the book as in (Barnsley, 1988) and the publications (Rama and Mishra, 2011; Wijk and Saupe, 2004).

Generation of the 3D sierpinski gasket: The process of generation of the cubed-basedversion of the 3D Sierpinski gasket (Bulusu, 2012) is outlined.

We started with a cube to generate the 3D Sierpinski Gasket (by imitating the 3D Sierpinski triangle) and then recursively re-generated the 3D fractalto arrive a new 3D Sierpinski gasket version-that resembles an almost 360° self-similar 3D version of the original fractal but one that is constructed as a self-manipulated syndicate of successive iterated version-results, each of which represents a 3D fractalpore of the initial 3D Sierpinski gasket. The programwas written in the C++ programming language, the 3D "visual" rendering was done using the open Glut Graphics library. It runs as a Windows consoleapplication with the variable depth being input as the first command-line argument (any non-zero integer value, preferably from 1- 9).

The originally generated 2D Sierpinski gasket uses a square as initiator and the IFS as generator. The equivalent 3D version of the Sierpinski gasket uses a cube as initiator and the IFS as the generator all of them having an additional property called depth = 3. Figure 3-5 are the maneuvered versions obtained by recursive re-generation with the same depth property (i.e., depth = 3). Figure 6 is the maneuvered version of Fig. 5 obtained by recursive re-generation with the depth property equal to 2. Finally, Fig. 7 seemingly displays the starting 3D

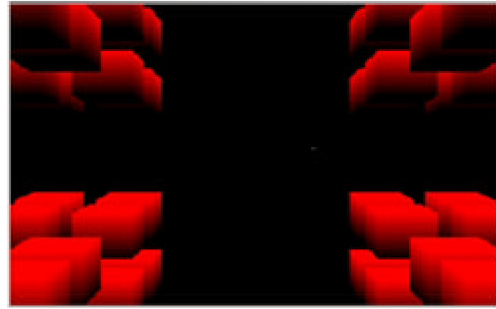

Fig. 4: The 3D Sierpinski Gasket (Cube-based) having depth = 3 obtained from recursive re-generation of Fig. 3

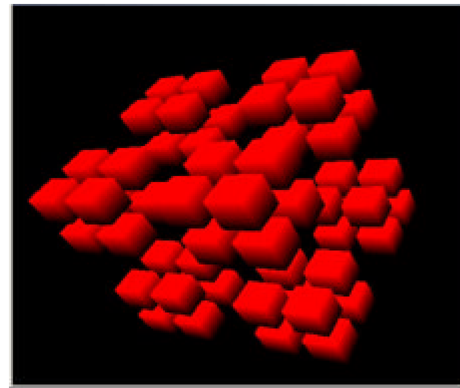

Fig. 5: The 3D Sierpinski gasket (Cube-based) having depth = 3 and applying the generator on Fig. 4

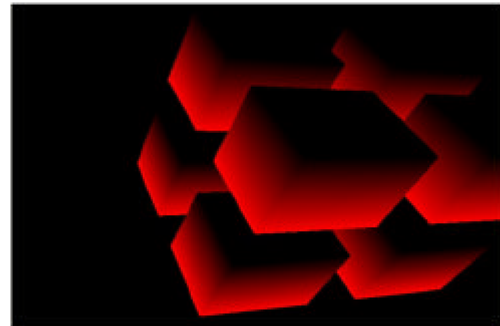

Fig. 6: The 3D Sierpinski gasket (Cube-based) Generatedusing depth = 2 applying the generator on Fig. 5

Sierpinski gasket as a 3D Cubed version (Mirtchovski. 2001) with the depth property changed to 1. The step-by-step procedure is as follows:

- Adding a new depth property that is input dependent to a chosen 3D cube image. The default depth is chosen to be 3

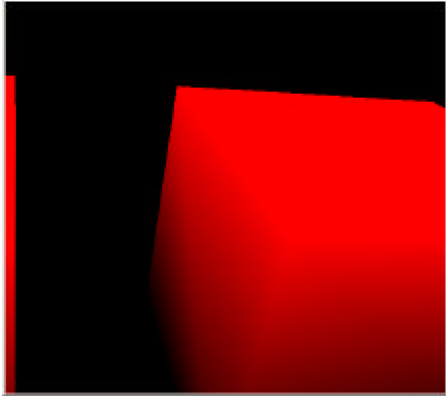

Fig. 7: The 3D Sierpinski Gasket (Cube-based) generated using depth = 1 applying the generator on Fig. 6

- The generating IFS consists of a (x,y,z) transformation that consists of an (x, y) rotationand a z-based zoom or the scaling factor
- Setting the reference frame by specifying a (height, width) pair to the enclosing (movable) window. The default colour scheme is red/blackfor each cube-based 3D fractal
- Using the IFS, auto-generate 8 self-similarfractal blocks and position them at the 8 different corners of its parent cube fractal. This creates a linked chain of smaller but self-similar 3D Sierpinski Gaskets-that are within theboundaries of the frame-of-reference. The dynamic variables involved in this method are:
- The depth property that represents anadditional new variant that defines aprojection
- The reference frame itself
- The (x, y, z) triple representing the translation forthe 3D fractal. Here z represents the zoom i.e., scaling factor

The dynamic translation is a chieved by auto-capturingthe variations in the scaling (the z-variable) value and the corresponding (x, y) rotation; and auto-adapting the Glprojection(s) to the conforming viewports.

The output: The program runs as a Windows console application using the Open Glut API with the 3D Sierpinski gasket (Bulusu, 2012) being generated as an IFS-based recursive version of the 2D Sierpinski carpet that uses a cube as the base initiatorfor the IFS. Successive fractal images are obtained by interactive mouse-based positional translation and context-aware zoom-based scaling in a dynamic fashion and varying the depth property from 3-1. Using Matlab 3D image rendering Software, the displayed graphics are processed for 'true' GUI compatibility.

RESULTS AND DISCUSSION

Discussion about different 3D Julia sets: The following section gives a discussion about the differences between the different 3D Julia sets as shown in Fig. 8-12. The Julia Set was invented by the French Mathematician Gaston Julia in 1918 while studying the iteration ofpolynomials and relational functions. It is very closely relatedto the Mandelbrot set. It is also obtained by iterating the equation $z = z^2+c$. The primary difference between the Julia set and the Mandelbrot set is the manner in whichthe function is iterated. The Mandelbrot set iterates as perz = $z^2 +c$ with z always starting at 0 and varying the cvalue. The Julia set iterates as per $z = z^2+c$, where c is constant and z is variable. In other words, the Mandelbrotset is in the parameter space or the c-plane, while the Julia set is in the dynamical space, i.e., the z-plane (Lee, 2011).

An Iterated Function System (IFS) based on a co-efficient cand the maximum number of iterations is iterated as manytimes as the maximum number of iterations. The resulting set of points can span an indeterminable amount of space that is afunction of the number of iterations involved. Now, for anyrandomly chosen point z from this set, it can either be located inside or outside of the generated area, depending on the valueof c and the co-ordinate-axes range. The Julia set comprisesall such points z, each of which lies outside of the bounded space before the IFS was applied. Applying the set of affinetransformations on the starting set of points, in each iterationstep, the resulting fractal is a self-similar shaped image that resembles an approximation to the original image. This effectis best visualized when rendered in 3D.

The corresponding program is written in C++ and built as a Visual C++ Project of type Windows Console Application. The resulting Julia Set is rendered as a 3D image using Matlab 3D image rendering Software. Multiple 3D Julia sets are generated by varying the number of iterations and therandom co-efficient c. Figure 8 shows the initial 2D image of the Julia set. Figure 9-11 show the different 3D Julia sets for Fig. 8.

The following Fig. 9 shows a 3D Julia set-the result obtained by successive iterations and changing various modeling parameters starting for every pixel, iterating znew = $zold^2+c$ on the complex plane until it leaves the circle around the origin with radius 2. The number of iterations it the color of the pixel.

For a Julia set, for each pixel apply an iterated complex function. Figure 12 This function is newz = $oldz^2+c$ with z and c both being complex numbers. The Z is initially the coordinates of the pixel and will then constantly be updated through every iteration: each

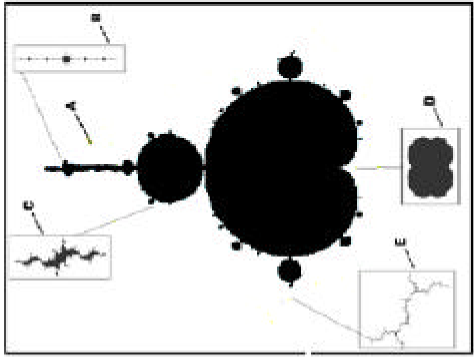

Fig. 8: The 2D Julia set-initial image

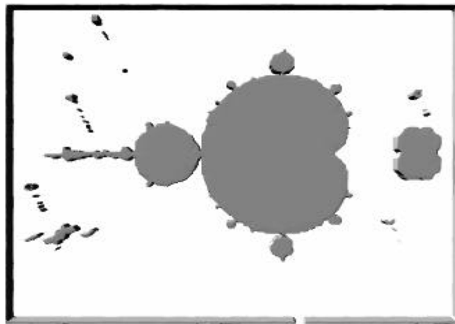

Fig. 9: First 3D Julia set for Fig. 5. 2D (zoomed-in)

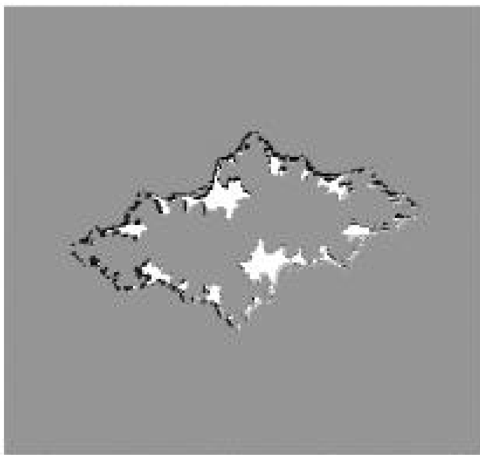

Fig. 10: The 2nd 3D Julia set from Fig. 5 (2D Julia set) obtained by changing thickness and width (zoomed-in)

iteration, the "newz" of the previous iteration is now used as "oldz". If we keep iterating this function, depending on the initial condition (the pixel), z will either go to infinity or remain in the circle with radius 2 around the origin of the complex plane forever. The points that remain in the circle forever are the ones that belong to the Julia set. So

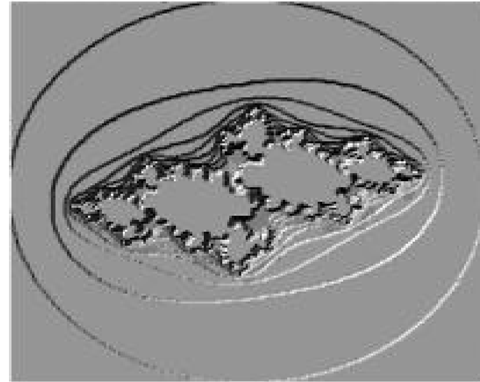

Fig. 11: 3rd 3D Julia thickness 17 mm, width 81 mm, grayscale defines thickness (zoomed-in)

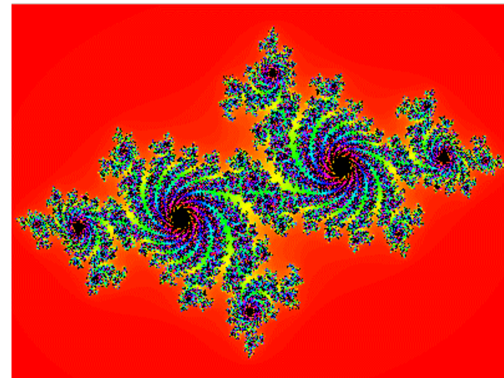

Fig. 12: Julia set for each pixel

we keep iterating the function until the distance of z to the origin (0, 0) is >2. And also give a maximum number of iterations, for example 256.

The color value of the pixel will then become the number of times we had to iterate the function before the distance of z to the origin got larger than 2. The constant c in the formula can be anything really as long as it's also inside the circle with radius 2. Different values of c give different Julia sets (Norton, 1982).

The more iterations, the more detailed the Julia set will look when zooming in deeply but the more calculations are needed. The higher the precision of the numbers, the longer we can zoom in without encountering blocky pixels.

We can change the values of zoom and position parameters to zoom in at certain positions. This can also be done in realtime while the program is running, for example by pressing the arrow keys to move and the keypad+and-keys to zoom in/out. Even better would be if you could use the keypad arrow keys to change the values of real and imaginary parts of the constant c, to change the shape of the Julia set in realtime.

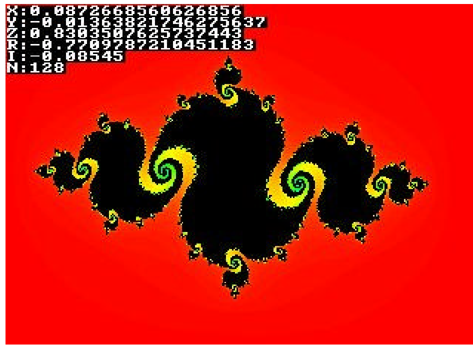

Fig. 13: Details of every possible Julia set

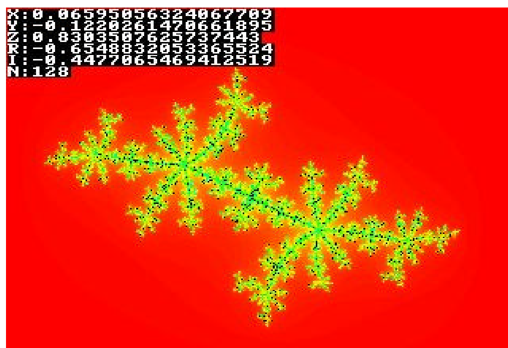

Fig. 14: Julia images zoomed

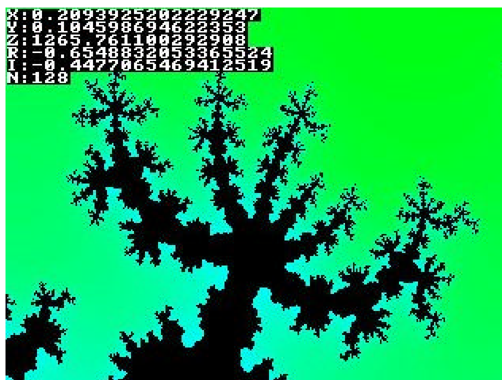

Fig. 15: Julia images after the increasing the number of iterations.

Programming something that can do that is pretty simple, just use the SDL keys to change the values of those variables and draw the Julia set every time again in a loop. By using the appropriate code and changing the required parameters at the appropriate place we can know exactly at what coordinates of the Julia set the nice things are. The result is that we can explore all the details of every possible Julia set, find a nice shape with the keypad numbers, then move with the arrow keys to a border or interesting spot of the Julia set and start zooming in to see more detail. The different shapes obtained are shown in Fig. 13-16.

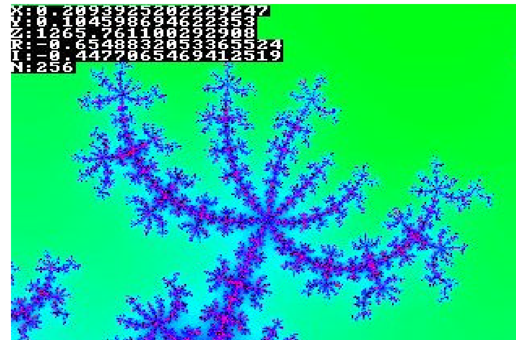

Fig. 16: Differences between different 3D Julia sets

Figure 14-16 show the differences between different 3D Julia sets obtained by using the appropriate code and rendering using 3D Matlab image rendering Software.

CONCLUSION

This study explained the study of 3D versions of some of the common fractals-the Mandelbulb and the Sierpinski gasket-the rendering of which gives a real-world look and feel in the world of fractal images, followed by the discussion of different 3D Julia sets. The two-dimensional (2D), 3D versions of the same have been realized based on the starting axioms/generators. The results demonstrate how by using theory, computation and experimentation fractal geometry is both an art and a science and also enables generation of more beautiful images and in higher dimensions too. In making further research on fractals, these fractals help us to explore the science of fractals in generation of new fractals based on the ideas presented here which can also enable us to have a real-world Imagineering of the same which can translate to examples like an entire coast-line set to dance in space by adding 3D-animation enabled elevation to the corresponding fractal image.

REFERENCES

Barnsley, M.F., 1988. Fractal Modeling of Real World Images. In: The Science of Fractal Images. Peitgen, H.O. and D. Saupe (Eds.). Springer New York, Berlin, Germany, ISBN: 978-1-4612-8349-2, pp: 219-242.

Bulusu, R.J.M., 2012. Using 3D Sierpinski gasket to generate and recursively re-generate 3D fractals-closing the self-similarity loop. Int. J. Graphics Vision Image Process., 12: 43-48.

Lei, T., 1990. Similarity between the Mandelbrot set and Julia sets. Commun. Math. Phys., 134: 587-617.

Mandelbrot, B.B., 1982. The Fractal Geometry of Nature. 1st Edn., W.H. Freeman, San Francisco, CA., ISBN: 0716711869.

Norton, A., 1982. Generation and display of geometric fractals in 3-D. ACM. SIGGRAPH. Comput. Graphics, 16: 61-67.

Rama, B. and J. Mishra, 2010. Generation of 3D fractal images for Mandelbrot and Julia Sets. Int. J. Comput. Commun. Technol., 1: 178-182.

Rama, B. and J. Mishra, 2011. Generation of 3D fractal images for Mandelbrot set. Proceedings of the 2011 International Conference on Communication, Computing and Security, February 12-14, 2011, ACM, ODISHA, India, ISBN: 978-1-4503-0464-1, pp: 235-238.

Wijk, V.J.J. and D. Saupe, 2004. Image based rendering of iterated function systems. Comput. Graphics, 28: 937-943.