Challenging Non-Euclidean Geometry of Irregular Objects in University Education: the Fractal Objects

Akemi Gálvez Tomida
Department of Applied Mathematics and Comp. Sciences
University of Cantabria, Avda. de los Castros
s/n, E-39005, Santander, Spain
galveza@unican.es

Abstract

The present work describes author's experience in developing a problem-solving environment (PSE) to analyze and display fractal objects. The PSE has been implemented by the author in the popular CAS *Matlab* as a module of an introductory course on nonlinear systems for undergraduate students of Mathematics, Physics and Engineering. Such a course has been designed to fully comply with Bologna's Declaration principles and regulations in the sense that it allows self-learning and promote active participation of students in the learning process. In this paper the architecture of this computer system along with a description of its main functionalities and some illustrative examples are briefly reported.

1 Introduction

Bologna's declaration - seen today as the well-known synonym for the whole process of reformation in the area of higher education - was signed in 1999 by 29 European countries with the objective to create *"a European space for higher education in order to enhance the employability and mobility of citizens and to increase the international competitiveness of European higher education"* [1]. Its upmost goal is the commitment freely taken by each signatory country to reform its own higher education system in order to create overall convergence at European level. This process encompasses the adoption of a common framework of readable and comparable degrees as well as the introduction of undergraduate and postgraduate levels in all countries along with ECTS (European Credit Transfer System) credit systems to ensure a smooth transition from one country's system to another one, thus enforcing free mobility of students, teachers and administrators among the European countries.

An important issue in this process is to provide students with a good collection of scholar materials that enable them to accomplish the learning process by themselves. During the last few years, the author has been involved in the development of computer software for an introductory course on nonlinear systems for undergraduate students of

Mathematics, Physics and Engineering. The course has been designed in a modular form, so that chapters describing different subjects are associated with different computer programs and materials. Some chapters (related to discrete and continuous chaotic dynamical systems) have already been described in [8, 10, 11]. This paper is specifically focused on fractal objects.

Although smooth curves and surfaces are the most usual graphical objects displayed in scientific documents, it is also interesting to represent graphically irregular mathematical objects, such as fractals [24]. Among them, the *Iterated Function Systems* (IFS) models, popularized by Barnsley in the 1980s, are particularly interesting due to their appealing combination of conceptual simplicity, computational efficiency and great ability to reproduce natural formations and complex phenomena [2]. For instance, the attractors of nonlinear chaotic systems exhibit a fractal structure [9, 13, 14, 18, 20, 21, 22, 26]. IFS fractals are typically made up of the union of several copies of themselves, where each copy is transformed by a function (function system). In the two-dimensional space such a function is mathematically a 2D affine transformation (see Section 3 for details), so the IFS is defined by a finite number of affine transformations (rotations, translations, and scalings), and therefore represented by a relatively small set of input data [15, 16, 17]. This fact has been advantageously used in the field of fractal image compression, an efficient image compression method that uses IFS fractals to store the compressed image as a collection of IFS codes [3, 6, 23]. The method, based on the idea that in images, certain parts of the image resemble other parts of the same image, has the great advantage that the final image becomes resolution independent. Moreover, this compression method is able to achieve higher compression ratios than conventional methods and still offer better visual quality.

In this paper, we describe a problem-solving environment (PSE) to analyze and display fractal objects. The PSE has been implemented by the author in the popular CAS *Matlab* as a module of an introductory course on nonlinear systems for undergraduate students of Mathematics, Physics and Engineering. Such a course has been designed to fully comply with Bologna's Declaration principles and regulations in the sense that it allows self-learning and promote active participation of students in the learning process. To this purpose, some specialized numerical libraries have been developed. Further, to provide end-users with a nice navigation and intuitive access to the main methods and routines, a powerful graphical user interface (also described in this paper) has been implemented. To show the good performance of this program, some illustrative examples discussing its use at the classroom are also reported.

The structure of this paper is as follows: in Section 2 the portfolio of the introductory course on nonlinear systems is portrayed. Then, some basic definitions and concepts about IFS are given in Section 3. The chaos game algorithm and the optimal choice of the probabilities required by the algorithm are also briefly discussed in that section. Section 4 describes the software introduced in this paper, including the description of system components, some implementation issues and a typical session workflow. Some illustrative examples showing the good performance of our program are reported in Section 5. The paper closes with the main conclusions and our future work.

2 Course Portfolio

As above-mentioned, the problem solving environment described in this paper is part of the material designed for the course *"Introduction to Nonlinear Systems"*, offered as elective course for sophomore, junior and senior students from Maths, Physics and Engineering (Civil, Naval, Mechanical, Chemical, Electrical, Electronical, Telecommunications, Mines and Computer Science) degrees. The course is scheduled to have 50 hours at the classroom, including theoretical classes, computer labs, final project discussion and evaluation, and 75 hours for students homework, practical assignments and final project. Main challenges for instructors are the high heterogeneity of students, as they come from different degrees and have different (although usually close) ages but also have different background, interests, goals and vision. It is an introductory course about the subject and hence students have no background on the field. The only requirements to access the course are some basic computer skills, such as a programming language and a fluent use of some CAS. The computer tool used in this course is the popular CAS *Matlab*, whose advantages for this course will be explained in Section 4. To teach this course, the author applies an educational strategy based on computer problem-solving environments specially designed to fulfill students' needs.

In this paper we focus on a particular topic of the course: the analysis of irregular objects and fractal geometry, with emphasis on the Iterated Function Systems (see Section 3 for details). This topic is explained in the course during three hours for the theoretical background, 3 hours for computer training with the program described in this paper and 7.5 hours of personal homework. The goal of this topic is to allow students to:

- understand the intrinsic complexity of fractal objects,
- capture the beauty of fractals objects in both Nature and Science,
- know some mathematical techniques to analyze them,
- discuss critically how to implement them on a CAS (pross and cons),
- identify relevant examples in their field,
- conduct further study by themselves about the topic and, finally
- integrate this topic into the knowledge core of the course.

3 Iterated Function Systems

3.1 Basic definitions

From the mathematical point of view, an *Iterated Function System (IFS)* is a finite set of contractive maps $w_i : X \longrightarrow X,\ i = 1, \dots, n$ defined on a complete metric space (X, d). We refer to the IFS as $\mathcal{W} = \{X; w_1, \dots, w_n\}$. In the two-dimensional case, the metric space (X, d) is typically $\mathbb{R}^2$ with the Euclidean distance d_2, which is a complete metric space, so the affine transformations w_i are of the form:

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = w_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} \tag{1}$$

or equivalently:

$$\mathbf{w}_i(\mathbf{x}) = \mathbf{A}_i.\mathbf{x} + \mathbf{b}_i$$

where $\mathbf{b}_i$ is a translation vector and $\mathbf{A}_i$ is a 2×2 matrix with eigenvalues λ_1, λ_2 such that $|\lambda_i| < 1$. In fact, $|det(\mathbf{A}_i)| < 1$ meaning that w_i shrinks distances between points. Let us now define a transformation, T, in the compact subsets of X, $\mathcal{H}(X)$, by

$$T(A) = \bigcup_{i=1}^{n} w_i(A). \tag{2}$$

If all the w_i are contractions, T is also a contraction in $\mathcal{H}(X)$ with the induced Hausdorff metric [2, 19]. Then, T has a unique fixed point, $|\mathcal{W}|$, called the *attractor of the IFS.*

3.2 Chaos game

Let us now consider a set of probabilities $\mathcal{P} = \{p_1, \dots, p_n\}$, with $\sum_{i=1}^{n} p_i = 1$. We refer to $\{\mathcal{W}, \mathcal{P}\} = \{X; w_1, \dots, w_N; p_1, \dots, p_n\}$ as an *IFS with Probabilities* (IFSP). Given $\mathcal{P}$, there exists a unique Borel regular measure $\nu \in \mathcal{M}(X)$, called the *invariant measure of the IFSP*, such that

$$\nu(S) = \sum_{i=1}^{n} p_i \nu(w_i^{-1}(S)), \quad S \in \mathcal{B}(X),$$

where $\mathcal{B}(X)$ denotes the Borel subsets of X. Using the Hutchinson metric on $\mathcal{M}(X)$, it is possible to show that M is a contraction with a unique fixed point, $\nu \in \mathcal{M}(X)$. Furthermore, *support*$(\nu) = |\mathcal{W}|$. Thus, given an arbitrary initial measure $\nu_0 \in \mathcal{M}(X)$ the sequence $\{\nu_k\}_{k=0,1,2,\dots}$ constructed as $\nu_{k+1} = M(\nu_k)$ converges to the invariant measure of the IFSP. Also, a similar iterative deterministic scheme can be derived from Eq.(2) to obtain $|\mathcal{W}|$.

However, there exists a more efficient method, known as *probabilistic algorithm*, for the generation of the attractor of an IFS. This algorithm follows from the result $\overline{\{x_k\}_{k>0}} = |\mathcal{W}|$ provided that $x_0 \in |\mathcal{W}|$, where (see, for instance, [4]):

$$x_k = w_i(x_{k-1}) \text{ with probability } p_i > 0. \tag{3}$$

Picking an initial point, one of the mappings in the set $\{w_1, \dots, w_n\}$ is chosen at random using the weigths $\{p_1, \dots, p_n\}$ according to Eq. (3). The selected map is then applied to generate a new point, and the same process is repeated again with the new point obtaining, as a result of this iterative process, a sequence of points. The sequence obtained using this stochastic process converge to the fractal as the number of points increases. This algorithm is known as probabilistic algorithm or *chaos game* [2] and generates a sequence of points that are randomly distributed over the fractal, according to the chosen set of probabilities. Thus, the larger the number of iterations (a parameter we can freely set up), the better the resolution of the resulting fractal image. As it will be shown later on, input data of main commands in our program include the number of iterations used to display the final image along with the method applied to generate the sequence of data points.

3.3 Optimal choice of probabilities

The fractal image is determined only by the set of contractive mappings; the set of probabilities gives the efficiency of the rendering process. Thus, a good choice for the probabilities is relevant for the efficiency of the rendering process, since the random sequence of points is generated according to these probabilities. One of the main problems of the chaos game algorithm is that of finding the optimal set of probabilities to render the fractal attractor associated with an IFS. Several different heuristic methods for choosing efficient sets of probabilities have been proposed in the literature [5, 7, 12]. The most standard of these methods was suggested by Barnsley [2] and has been widely used in the literature. For each of the mappings, this method (called *Barnsley's algorithm*) selectes a probability value that is proportional to the area of the figure associated with the mapping. Since the area filled by a linear mapping w_i is proportional to its contractive factor, s_i, this algorithm proposes to take:

$$p_i = \frac{s_i}{\sum\limits_{j=1}^{n} s_j} \quad ; \quad i = 1, \dots, n. \tag{4}$$

Another algorithm, proposed in 1996 and known as *multifractal algorithm* [16, 17], provides a method for obtaining the most efficient choice for the probabilities as: $log(p_i) = Dlog(w_i) \Leftrightarrow p_i = w_i^D$,(where $D$ is a real constant) along with $\sum\limits_{i=1}^{n} w_i^D = 1$. Then, the most efficient choice corresponds to

$$p_i = s_i^D \; ; \; i = 1, \dots, N \tag{5}$$

where $D$ denotes the *similarity dimension.* This method will be called *optimal algorithm* onwards.

4 Program Architecture and Implementation

4.1 Program Architecture

The problem solving environment introduced in this paper for generating and rendering IFS fractals consists basically of two major components:

1. a *computational library (toolbox)*: it contains a collection of commands, functions and routines implemented to perform the numerical and graphical tasks.

2. a *graphical user interface (GUI)*: this component is responsible for input/output windowing and smooth interaction with the user.

Our numerical functions have been implemented by using the native *Matlab* programming language. They take advantage of the large collection of numerical routines available in this system. Usually, these built-in *Matlab* routines provide extensive control on a number of different options and are fully optimized to offer the highest level of performance. In fact, this is one of the major strengths of the program and one of the main reasons to choose *Matlab* as a convenient programming environment.

On the other hand, the powerful *Matlab* graphical capabilities exceed those commonly available in other CAS such as *Mathematica* and *Maple*. Although our current needs do not require to apply them at full extent, they avoid us the tedious and time-consuming task to implement many routines for graphical output by ourselves. In fact, some nice viewing features such as 3D rotation, zooming in and out, labeling, scaling, coloring and others, which are automatically inherited from the *Matlab* windows system, have been intensively used in our system.

Although this toolbox is enough to meet all our computation needs, potential end-users might be challenged for using it properly unless they are really proficient on *Matlab*'s syntax and functionalities and the toolbox routines. This kind of limitations can be overcome by creating a GUI; a well-designed GUI uses readily recognizable visual cues to help the user navigate efficiently through information. Fortunately, *Matlab* provides a mechanism to generate GUIs by using the so-called *guide* (*GUI development environment*). This feature is not commonly available in many other CAS so far. Although its implementation requires - for complex interfaces - a high level of expertise, it allows end-users to deal with the toolbox with a minimal knowledge and input, thus facilitating its use and dissemination.

Based on this idea, a GUI for the already-generated toolbox has been implemented. Figure 1 shows an example of a typical window. It allows an effective use of powerful interface tools designed according to the type of values being displayed (e.g., drop-down menu for a choice list, radio buttons for single choice from multiple options, text boxes for displaying messages, list boxes for input/output user interaction, etc.). In general, functions associated with each topic under analysis have been grouped and arranged in a rectangular area with an indicative title on the top, such as: code, initial set, method and iterations (see Figure 1 and subsequent description in Section 4). They are also labeled with numbers in red from 1 to 4 in Figure 1 for prompt identification. Thus, in the upper part of area 1 you can see some boxes and buttons for input/output user interaction. Below those boxes, some buttons for additional tasks and a list box to display generated IFS code are also included (see Section 4 for further description). Additional functionalities are also provided in hidden menus or in separate windows which can be invoked at will if needed so as to keep the main window streamlined and uncluttered. For instance, all graphical output is displayed in separate windows so that the information is better organized and "flows" in a natural and intuitive way. As a result, this GUI presents an interface which is both aesthetic and very functional to the user.

4.2 Implementation Issues

Regarding the implementation, this program has been developed by the author in *Matlab* [25] v2008a on Windows XP operating system by using a PC with Intel Core 2 Duo processor at 2.4 GHz. and 2 GB of RAM. However, the program supports many different

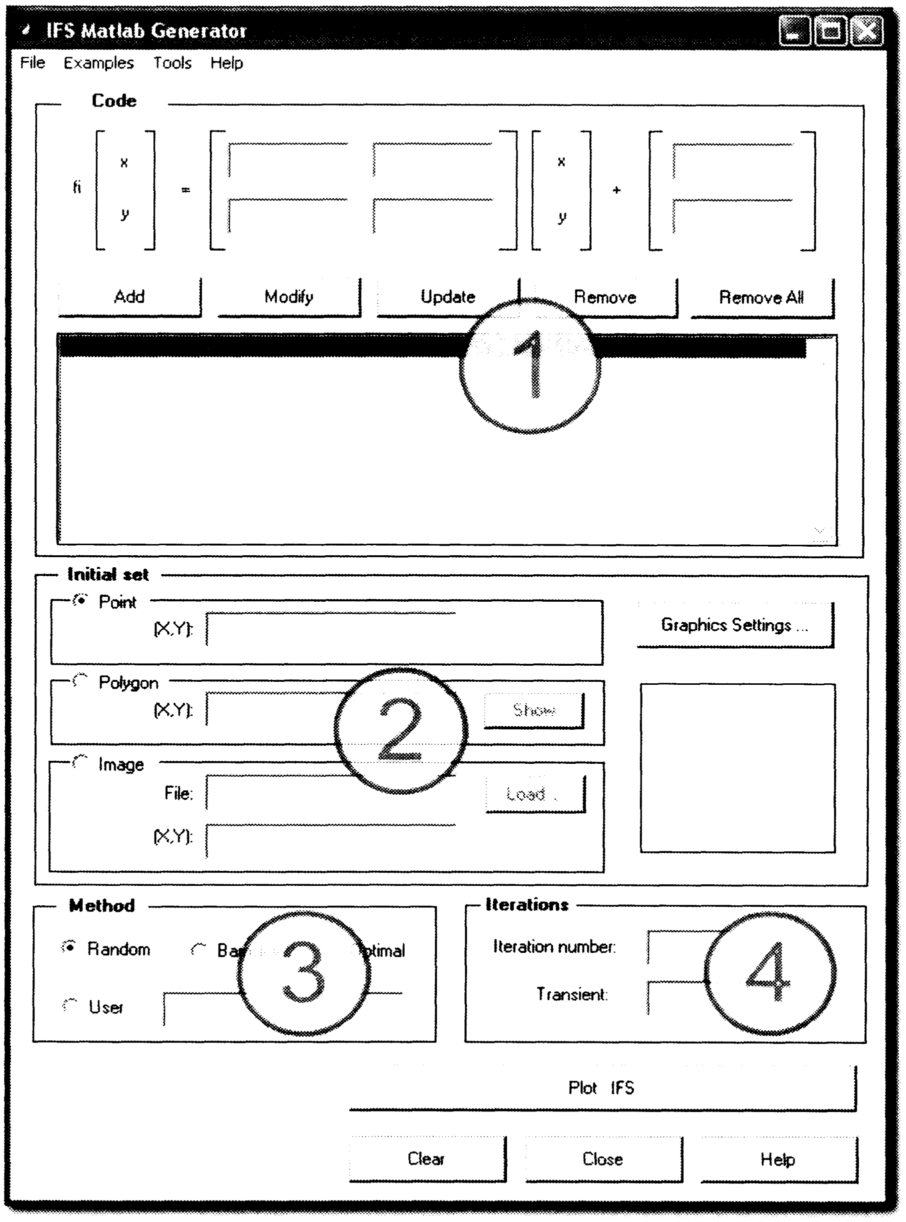


Figure 1: Screenshot of main window of our program

platforms, such as PCs (with Windows 9x, 2000, NT, Me, XP and Vista) and UNIX workstations. A version for Apple Macintosh with Mac OS X system is also available provided that X11 (the implementation of the X Window System that makes it possible to run X11-based applications in Mac OS X) is properly installed and configured. Figures in this paper correspond to the PC platform version.

The graphical tasks are performed by using the *Matlab* GUI for the higher-level functions (windowing, menus, or input) while the OpenGL-based built-in graphics *Matlab* commands are applied for rendering purposes. All our numerical functions have been implemented in the native *Matlab* programming language.

Table 1: IFS code of Barnsley's fern

	a	b	c	d	e	f
w_1	0.81	0.07	-0.4	0.84	0.12	0.195
w_2	0.18	-0.25	0.27	0.23	0.12	0.02
w_3	0.19	0.275	0.238	-0.14	0.16	0.12
w_4	0.0235	0.087	0.045	0.1666	0.11	0

5 Some Illustrative Examples

In this section the main features of the program are described through its application to generate and render several IFS fractals.

5.1 Session Workflow

A typical session in our problem solving environment for fractals begins by writing the IFS code of a fractal. To this purpose, some input boxes are provided in the IFS code area (labeled as 1 in Figure 1). They reproduce the typical structure of an IFS code, with input boxes for the coefficients of the IFS functions, namely, values for a_i, b_i, c_i, d_i, e_i and f_i arranged in a two-dimensional matrix and vector according to Eq. (1). Once a new function is inputted, it can be added to the list of iterated functions by pressing button "Add". This action is immediately reflected in the list box below such a button since the new iterated function shows up. For instance, Figure 3 displays the IFS code of the famous *Barnsley's fern*, given by Table 1. Such a fractal is represented graphically in Figure 2.

We can modify any already created function through the button "Modify". To this aim, it is enough to click on the list box at the location of the contractive function we wish to modify and its parameter values will be placed back onto the input boxes for further modification. Finally, clicking on the button "Update" returns the new function code to its former position at the list box. User is kindly warned at this point about the possible confusion between pressing button "Add" or "Update" once function parameters are modified. The former button adds a new function, so the IFS is enlarged regarding the number of iterated functions, while the latter one replaces the former function by the modified one, so the number of iterated functions for the IFS does not change. We can also remove any contractive function of an IFS at any time (button "Remove"). Alternatively, button "Remove All" removes the codes of all contractive functions of current IFS.

Once we know how an IFS describe a fractal image, the next step is introducing a rendering method. Equation (2) provides us with the simplest rendering algorithm, called *deterministic algorithm*. It works by generating a sequence of images obtained by iterating (2) starting on an initial set in the plane. This sequence will converge to the attractor of the IFS independently on the initial set meaning that we can start with any arbitrary set or image. Our program allows us to start with several initial sets, ranging from a single point to a polygon or a picture (see label 2 in Figure 1). Figure 3 shows an example of a picture as initial set. As the reader can see, the program allows us to choose any arbitrary picture from our storage units or devices by simply selecting or writing the

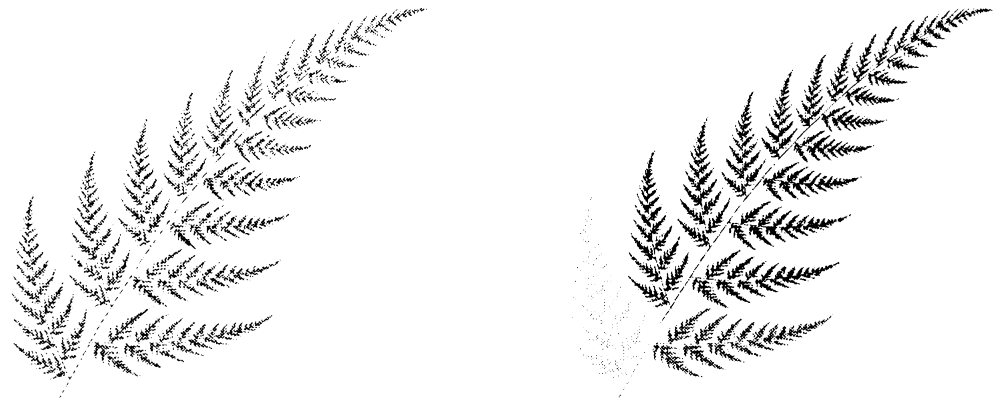


Figure 2: Bernsley's fern (left) and its contractive functions in color (right)

filename and its path from current directory and the initial region where such a picture is going to be displayed before iteration.

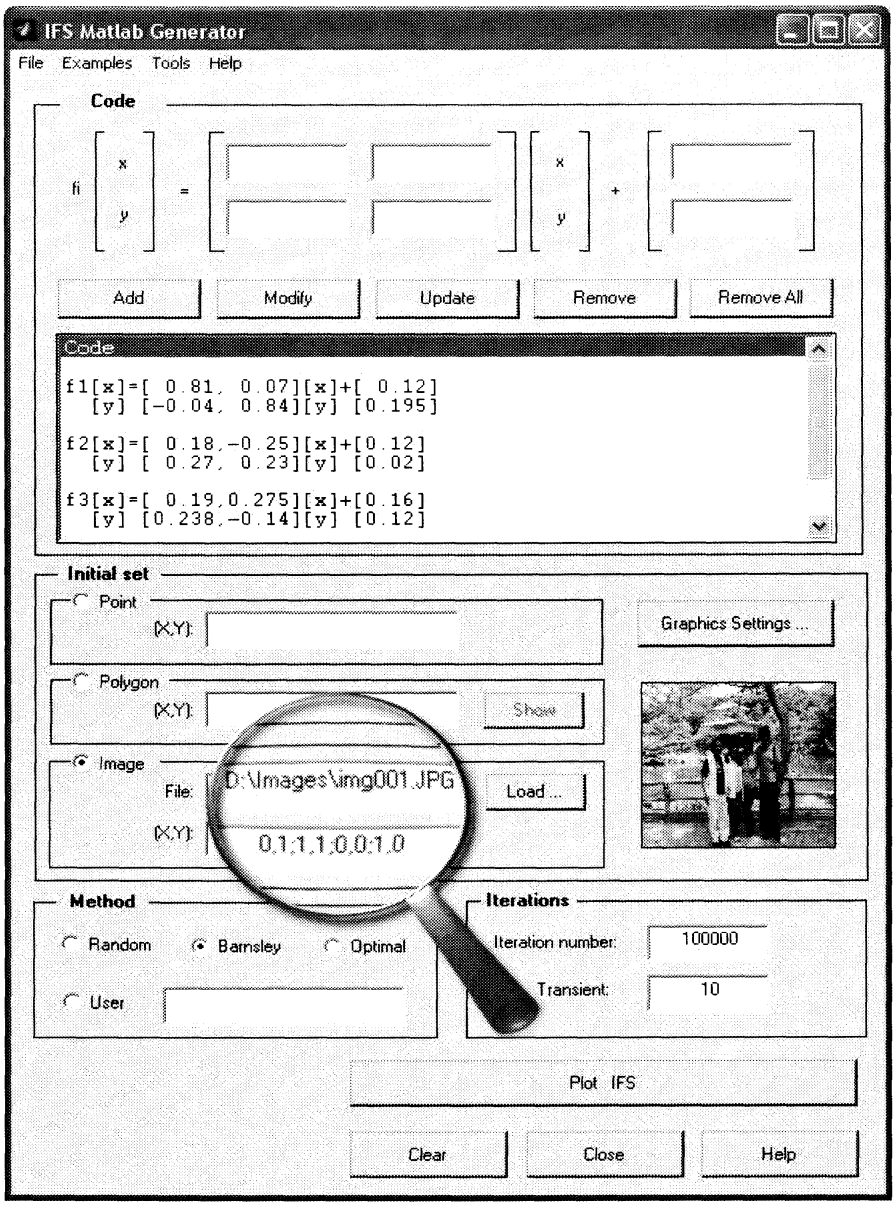
It is worthwhile to mention that, while the final image (the attractor, shown in Figure 2) will always be the same regardless the initial set we select, the computational complexity to get such an image will not. So, instead of dealing with pictures or complicated shapes as initial sets, it is more convenient to use very simple sets (preferably a single point, like in Figure 4) to that purpose.

Another factor to alleviate the computational load concerns the rendering algorithm. In spite of the beauty and simplicity of the deterministic algorithm, it is computationally expensive and, hence, practically useless. A more efficient algorithm can be obtained by attaching real weights p_i to each of the transformations in the IFS, such that:

$$\sum_{i=1}^{n} p_i = 1 \tag{6}$$

Picking an initial point, one of the mappings in the set $\{w_1, \dots, w_n\}$ is chosen at random using the weigths $\{p_1, \dots, p_n\}$ according to Eq. (3). The selected map is then applied to generate a new point, and the same process is repeated again with the new point obtaining, as a result of this iterative process, a sequence of points. The sequence obtained using this stochastic process converge to the fractal as the number of points increases. This algorithm is known as probabilistic algorithm or *chaos game* [2]. Third area in our main window (label 3 in Fig. 1) allows us to select the probabilities in four different modes: randomly or according to Barnsley's algorithm, multifractal algorithm or user's choice. In the later case, user writes the probabilities in an input box as a list of values separated by commas. In the three former cases, probabilities are normalized to ensure Eq. (6) holds.

The chaos game algorithm generates a sequence of points that are randomly distributed over the fractal, according to the chosen set of probabilities. Thus, the larger the number of iterations, the better the resolution of the resulting fractal image. Input data for our program also includes the number of iterations used to display the final



```
f1[x]=[ 0.81, 0.07][x]+[ 0.12]
  [y] [-0.04, 0.84][y] [0.195]

f2[x]=[ 0.18,-0.25][x]+[0.12]
  [y] [ 0.27, 0.23][y] [0.02]

f3[x]=[ 0.19,0.275][x]+[0.16]
  [y] [0.238,-0.14][y] [0.12]
```

Point (X,Y):

Polygon (X,Y):

[✓ Image] File: D:\Images\img001.JPG (X,Y): 0,1;1,1;0,0;1,0

Random [✓ Barnsley] Optimal User

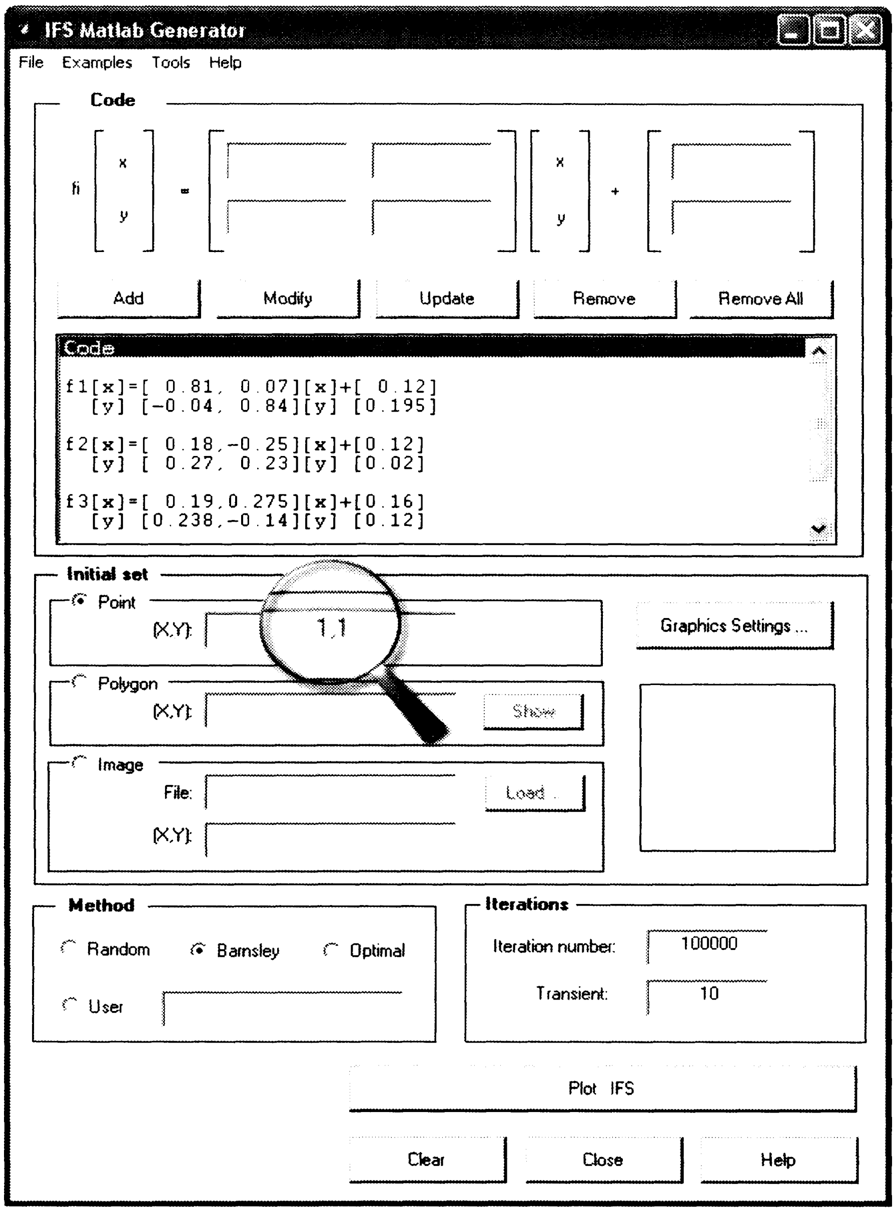
Iteration number: 100000

Transient: 10

Figure 3: Choosing a picture as the initial set

image (label 4 in Figure 1). Fractal images in this paper have been generated with about 2×10^5 iterations. It is also convenient to consider a transient of m initial iterations (set to $m = 10$ in all our pictures) that are not displayed in order to skip points that do not really belong to the attractor and might otherwise be displayed before convergence.

One of the most surprising properties of IFS models is their ability to capture the main features of some natural formations. For example, it is not difficult to realize at first glance that the fractal images given in Figure 5 resemble leaves (top) and trees (bottom), respectively. Note also the amazing realism of Figure 2.



```
Code
f1[x]=[ 0.81, 0.07][x]+[ 0.12]
  [y] [-0.04, 0.84][y] [0.195]

f2[x]=[ 0.18,-0.25][x]+[0.12]
  [y] [ 0.27, 0.23][y] [0.02]

f3[x]=[ 0.19,0.275][x]+[0.16]
  [y] [0.238,-0.14][y] [0.12]
```

Initial set: [✓ Point] (X,Y): 1,1 — Polygon (X,Y): — Image File: (X,Y):

Method: Random [✓ Barnsley] Optimal User

Iterations: Iteration number: 100000 Transient: 10

Figure 4: Choosing a point as the initial set

6 Conclusions and Further Remarks

In this paper a *Matlab*-based problem solving environment for analyzing and displaying IFS fractals is presented. The program allows us to display any two-dimensional IFS fractal through a wealth of graphical and numerical options. Additional options for the determination of the fractal dimension, the best probabilities for the IFS rendering algorithm and other numerical tasks have also been incorporated. The program have exhibited a very good performance in all our examples described here and many others not reported because of limitations of space. In author's opinion, this program has been very useful to introduce our students into the fractal geometry world, and instruct them

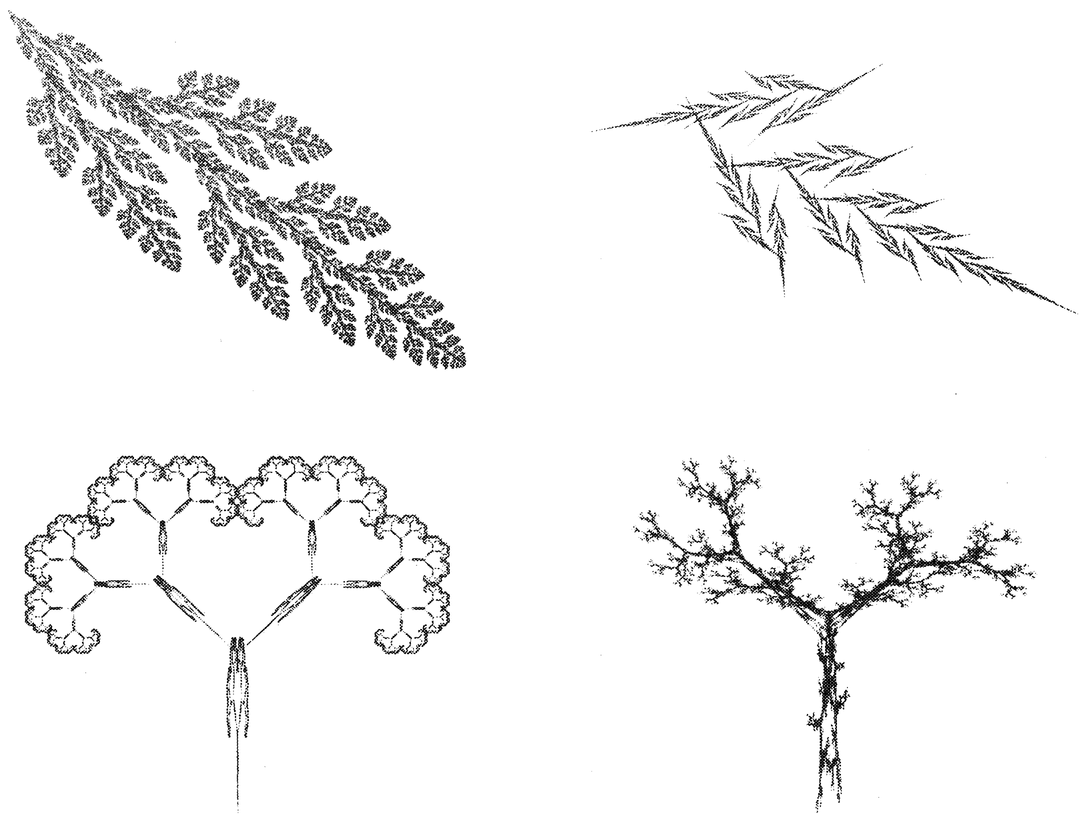


Figure 5: IFS models associated with two different natural formations

what for and how fractals are used.

Regarding the educational issues, the described PSE has been very well welcome by my students. They were enthusiastic about this computer-based approach and become engaged from the very beginning. On the other hand, students' projects and assignments showed that the software effectively help the students to grasp the main concepts and techniques in the subject under study. In general, it can be said that students feedback has been very positive and encourages me to keep creating new computer tools for this and other courses on similar topics. It is author's hope that this experience can also be useful to other teachers and educators pursuing to follow a similar approach. Feedback about those similar experiences would be kindly welcome.

Acknowledgments

The author would like to express her sincere acknowledgment and appreciation to Prof. Setsuo Takato for his kind invitation to participate in this RIMS workshop by delivering this talk and visit the lovely cities of Kyoto and Uji (and its impressive Byodo-in) and for creating such a wonderful atmosphere during all my stay in Kyoto. Special thanks are also due to Prof. Takato and all KETpic members for making my birthday such a special day (*Ookini!!!*).

This research has been supported by the Computer Science National Program of the Spanish Ministry of Education and Science, Project Ref. #TIN2006-13615 and the University of Cantabria.

References

[1] The Bologna Declaration on the European space for higher education: an explanation. Association of European Universities & EU Rectors' Conference (1999) pp. 4 (available at: *http://ec.europa.eu/education/policies/educ/bologna/bologna.pdf*).

[2] Barnsley, M.F.: *Fractals Everywhere, Second Edition.* Academic Press (1993)

[3] Barnsley, M.F., Hurd, L.P.: *Fractal Image Compression.* AK Peters, Wellesley, MA. (1993)

[4] Elton, J.H.: "An Ergodic Theorem for Iterated Maps," *Ergodic Theory Dynam. Syst.*, *7* (1987) 481-488

[5] Falconer, K.: *Fractal Geometry: Mathematical Foundations and Applications.* Wiley (1990)

[6] Fisher, Y.: *Fractal Image Compression: Theory and Applications.* Springer-Verlag (1995)

[7] Forte, B. Vrscay, E. R.: "Solving the inverse problem for measures using iterated function systems : a new approach". *Adv. Appl. Prob.*, **27**, (1995) 800-820

[8] Gálvez, A.: "Numerical-symbolic Matlab program for the analysis of three-dimensional chaotic systems". *Lectures Notes in Computer Science*, **4488** (2007) 211-218

[9] Gálvez, A., Iglesias, A.: "Symbolic/numeric analysis of chaotic synchronization with a CAS". *Future Generation Computer Systems* **25**(5) (2007) 727-733

[10] Gálvez, A.: "Matlab Toolbox and GUI for Analyzing One-dimensional Chaotic Maps, *Computational Science and its Applications-ICCSA'2008. IEEE Computer Society Press,* Los Alamitos, California, USA (2008) 321-330

[11] Gálvez, A.: A Matlab Problem-Solving Environment for Nonlinear Systems Education in Mathematics, Physics and Engineering, *RIMS Kokyuroku Journal Series*, **1624** (2009) 129-144.

[12] Graf, S.: "Barnsley's scheme for the fractal encoding of images". *Journal of Complexity*, **8** (1992) 72-78

[13] Gutiérrez, J.M., Iglesias, A., Rodríguez, M.A.: "Logistic Map Driven by Correlated Noise". Second Granada Lectures in Computational Physics. Garrido, P.L., Marro, J. (Eds.) *World Scientific*, Singapore (1993) 358-364

[14] Gutiérrez, J.M., Iglesias, A., Rodríguez, M.A.: “Logistic Map Driven by Dichotomous Noise”. *Physical Review E, 48*(4) (1993) 2507-2513

[15] Gutiérrez, J.M., Iglesias, A., Rodríguez, M.A., Rodríguez, V.J.: “Fractal Image Generation with Iterated Function Systems”. In ”Mathematics With Vision”. In: *Proceedings of the First International Symposium of Mathematica*, V. Keranen and P. Mitic (Eds.), Computational Mechanics Publications (1995) 175-182

[16] Gutiérrez, J.M., Iglesias, A., Rodríguez, M.A.: “A Multifractal Analysis of IFSP Invariant Measures with Application to Fractal Image Generation”. *Fractals*, **4**(1) (1996) 17-27

[17] Gutiérrez, J.M., Iglesias, A., Rodríguez, M.A., Rodríguez, V.J.: “Efficient Rendering in Fractal Images”. *The Mathematica Journal*, **7**(1) (1997) 7-14

[18] Gutiérrez, J.M., Iglesias, A.: “A Mathematica package for the analysis and control of chaos in nonlinear systems”. *Computers in Physics*, **12**(6) (1998) 608-619

[19] Hutchinson, J.: “Fractals and Self-Similarity”. *Indiana Univ. Math. Jour.*, **30** (1981) 713-747

[20] Iglesias, A., Gálvez, A.: “Analyzing the synchronization of chaotic dynamical systems with Mathematica: Part I”. *Lectures Notes in Computer Science* **3482** (2005) 472-481

[21] Iglesias, A., Gálvez, A.: “Analyzing the synchronization of chaotic dynamical systems with Mathematica: Part II”. *Lectures Notes in Computer Science* **3482** (2005) 482-491

[22] Iglesias, A., Gálvez, A.: “Revisiting some control schemes for chaotic synchronization with Mathematica”. *Lectures Notes in Computer Science* **3516** (2005) 651-658

[23] Jacquin, A.E.: “Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations”. *IEEE Transactions on Image Processing*, **1**(1) (1992) 18-30

[24] Mandelbrot, B. B.: *The Fractal Geometry of Nature.* W. H. Freeman and Co. (1982)

[25] The Mathworks Inc: *Using Matlab.* Natick, MA (1999)

[26] Peitgen, H. O., Jurgens, H. and Saupe, D.: *Chaos and Fractals. New Frontiers of Science.* Springer-Verlag (1993)