EINDHOVEN UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING
Design Automation Group

CHAOS THEORY, FRACTAL GEOMETRY AND A GENETIC APPROACH TO IFS-ENCODING

by E.C.G.J. van Duren (322325)

Thesis submitted to fullfill the requirements for obtaining the degree of Master of Information Engineering, performed from May 11th till December 24th 1992, by order of and supervised by Prof. Dr. Ing. J.A.G. Jess.

To those whom I respect.

I want to know how God created this world. I am not interested in this or that phenomenon. I want to know His thoughts, the rest are details.

A. Einstein [ROB90]

Hofstadter's Law:

It always takes longer than you expect it will take, even if you take into account Hofstadter's Law.

D.R. Hofstadter [HOF85]

Abstract

To many scientists the words chaos and fractals either make no sense or are only associated with nice pictures. Up till now, only relatively few of them seem to fully understand the fundamental and universal properties of this new theory. One of the goals of this master thesis is to comprehend the concepts of chaos theory and fractal geometry in a compact and readable format. This is achieved by a bottom up approach and an extensive use of examples.

Fractal geometry is an extension of classical geometry and appears to be very suitable to describe natural structures, such as mountains, trees, clouds, etc. Scale-invariance of the amount of detail is the key concept.

Chaos theory provides powerful tools to analyze the behaviour of complex dynamical systems. On one hand it is an extension of the conventional mathematical analysis, which is based on numerical approximation, on the other hand it ties together aspects from various fields of science, such as physics, information theory and electronics.

Fractal geometry and chaos theory are closely related to each other by the concepts of scaling, and iteration (recursive composition). Furthermore, chaos theory has a dual character. On one hand it puts an end to the dream of full predictability of deterministic dynamical systems, on the other hand it gives some clues about the up till now supposed unpredictability of nondeterministic dynamical systems. Chaos theory is also closely related to already known concepts, such as Lyapunov exponents and entropy.

A dynamical system can be suitably represented by a set of iterated transformations which operate on a metric space. If the transformations have the property of being affine (i.e. map parallellograms to parallellograms), and are each applied with an assigned probability, then such dynamical system is called an Iterated Function System (IFS). The parameters of an IFS can be represented in a highly compact IFS-code. In this sense, an IFS-code establishes an extensive compressed representation of the (behaviour of) the dynamical system it encodes. Compression ratio's of 10000:1 are possible.

The process of decoding an IFS-code into the associated picture is trivial and fast. The rendered picture is called the attractor and represents the behaviour of the according dynamical system. If an attractor has fractal properties, then the associated dynamical system may possess chaotic behaviour under certain conditions.

The inverse process of encoding a target image into an IFS-code is not trivial at all, but, instead, is proven to be NP-complete. The reason we are very much interested in IFS-encoding is the fact that it implies a tool for enormous data compression and good control of real world images, which could have applications in the defence industry (visual systems of weapon platform simulators, storage of satellite data) and in new commercial applications (virtual reality, film, VLSI-design). Maybe IFS-encoding will appear to be a powerful tool for modelling dynamical systems of which only the attractor is known.

The practical component of this graduation work is mainly concerned with the implementation of a toolbox for applying interactive and automatic IFS-encoding. The toolbox is embedded in the OSF/Motif ® X Window System® and it fully supports IFS-decoding (both deterministic and nondeterministic) as well as interactive IFS-encoding. A genetic algorithm is implemented to achieve automatic IFS-encoding. Due to the enormous degree of freedom which exists in the tuning of the parameterset with which the genetic algorithm is initialized, we have only reached results up to 88% approximation quality in obtaining a collage (which implies 80% approximation quality for the rendered attractor). Symmetry group transformations (mainly used for validation of the genetic algorithm) have been found up to 99% approximation quality. However we feel that the genetic approach is a promising one and further investigations on this area will certainly improve its performance.

Acknowledgements

I acknowledge and thank Professor Jochen Jess for giving me the opportunity to fullfill this master thesis and to use the powerful computer facilities of his special group. It was his open mind and willingness to enter new unexplored trails that encouraged me to ask him to supervise this work. I thank Hans Fleurkens for introducing me to the OSF/Motif® X Window System®. I thank Marc Heijligers and Luc Cluitmans for introducing me to genetic algorithms. I thank the Royal Netherlands Air Force for providing me with the resources necessary to fullfill my study in Information Engineering. I thank my girl friend Heidy for her continuous and unconditional support throughout the whole study period. I thank my parents who stood at the basis of my personal development and who taught me that hard working is a good property. Last but not least, I thank God, not only for letting me fullfill this study, but also for creating the world as it is, especially its beautiful fractal geometry and its surprising chaotic dynamics.

Emile van Duren

Contents

The great morning of the world when first God dawned on chaos.

Percy Bysshe Shelley (1792 - 1882) [DAR80]

Chapter 1

Introduction

Newton stated that the world unfolded along a deterministic path, rule-bounded like the planets, predictable like eclipses and tides. Laplace imagined a supreme intelligence and stated: "Such an intelligence would embrace in the same formula the movements of the greatest bodies of the universe and those of the lightest atom; for it, nothing would be uncertain and the future, as well as the past, would be present to its eyes" [GLE87].

With chaos theory the era of the deterministic dream came to an end. Scientists who have understood the essence of chaos, will no longer hunt for full predictability anymore.

1.1. The birth of a new science

In the last few decennia, it has become clear that the language of classical geometry is insufficient to describe the natural geometry and the behaviour of dynamical systems. Therefore fractal geometry and chaos theory were developed. Strange enough they were developed almost independently (fractal geometry from mathematics and chaos theory from physics), despite the fact that they are closely related to each other. Scaling and iteration are the key concepts of both theories. In particular, it appears that a fractal is a "picture" of the behaviour of a dynamical system.

Despite the work a few early twentieth century mathematicians, like Cantor, Von Koch and Sierpinski, put in developing geometrical structures of infinite detail, it was Mandelbrot who gave the starting shot for the building of the theory of fractal geometry, with his famous paper "How long is the coast of Britain ?" [MAN67]. Four years earlier, Lorenz had done the same for the building of chaos theory with his historical paper: "Deterministic nonperiodic flow" [LOR63]. Here too, the man who gave the starting shot, was not the first one who got a glimpse of what was going on. In the late nineteenth century, the french mathematician Poincaré proved that it is in general impossible to give a simple prescription for the orbit of one body influenced by two other bodies (the three-body problem).

The concept of chaotic behaviour now pervades virtually all the sciences. In simple terms, we can say that a dynamical system exhibits chaos if its behaviour is not random, but there are no predictable patterns to it. Chaotic dynamics is the study of such behaviour, and despite the unpredictability of chaotic system behaviour, mathematical analysis can often describe it.

Chaos theory has been successfully applied to scientific fields which study a variety of topics such as earthquakes, heart attacks, infectuous diseases, stock markets, the weather, chemical reactions, electronic circuits, organizations, etc.

Geometry is concerned with making our spatial intuitions objective. Classical geometry provides a first approximation to the structure of physical objects; it is the language which we use to communicate the designs of technological products and, very approximately, the forms of natural creations. Fractal geometry is an extension of classical geometry. It can be used to make precise models of all existing natural structures, such as clouds, mountains, trees, bird feathers, galaxies, etc.

Alltogether, chaos theory and fractal geometry are the first languages that are powerful enough to describe how the world is really (dis)organized, both with respect to its behaviour as to its geometry. Both theories contain universal properties, and it has been said that those new theories form the third scientific revolution of this century, after relativity and quantum mechanics [GLE87].

1.2. What is new about chaos ?

Engineers have always known about chaos. It was called noise or turbulence or something else, and safety factors were used to design around these apparent random unknowns that seem to crop up in every technical device. So what is new about chaos ?

First, the recognition that chaotic behaviour can already arise in low-order, nonlinear deterministic systems, raises the hope of understanding the source of randomlike noise and doing something about it. Second, the new discoveries in nonlinear dynamics bring with them new concepts and tools for detecting chaotic dynamics in physical systems and for quantifying this "deterministic noise" with new measures, such as fractal dimension and Lyapunov exponents. Third, fractal geometry seems to be a promising tool for enormous data compression and computer animation, since it is able to use iteration as a decoding mechanism, which transforms very few bytes of information into complex geometrical structures.

Concluding, what is new about chaotic dynamics is the discovery of a seemingly underlying order which holds out the promise of being able to predict certain properties of noisy behaviour, which occurs when some strong nonlinearity exists. Even a chaotic system will at last complete a deterministic periodic cycle, but this can take centuries. Poincaré already stated that water at room temperature could spontaniously turn into ice, but the chance that it does is very small.

1.3. Thesis goal

The goal of this thesis is threefold. First it should give a thorough and compact overview of the essences of chaos theory and fractal geometry. Second it should document the developed user-interactive program IFS-CODEC, which facilitates application of all important concepts of fractal geometry. Third it should describe our attempt of solving an unsolved fundamental problem of fractal geometry, i.e. automatic IFS-encoding. This problem has applications in data compression and computer animation (flight simulators, virtual reality, etc.).

1.4. Thesis outline

The first goal of this thesis is achieved by the chapters 1 through 7, the second goal by the appendices A and B and the third goal by the chapters 8 through 11. We have tried to illustrate the theoretical concepts as much as possible with representative examples, to facilitate a quick understanding. The reader who is not familiar with either chaos theory or fractal geometry, is advised to study this thesis in chronological order, since it is written from a bottom-up point of view, i.e. each chapter introduces elements that will be used in successive chapters. The text of this thesis contains theorems, definitions and examples, the termination of which is respectively denoted by ■, ♦ and ★. Predicates are sometimes written in the "Dijkstra" form, sometimes in the classical form and sometimes in a mixed form. For example the following predicates are equivalent, but the left one is written in the "Dijkstra" form, while the right one is written in the classical form:

$$(\underline{A}\, n : 1 \le n : (\underline{S}\, i : 1 \le i \le n : i) = \tfrac{1}{2}(n^2+n)) \Leftrightarrow \forall_{n \in \mathbb{N}} (\sum_{i=1}^{n} i = \tfrac{1}{2}(n^2+n))$$

Despite the fact that this introduces an inconsequency, we decided to use both notations, because both have (dis)advantages. The "Dijkstra" form is very wel suited to use in regular textlines, but can be rather lengthy, while the classical form is not at all suited to use in textlines, but is very compact.

Chapter 2 basically deals with geometrical definitions, the most of which will be used throughout this thesis.

Chapter 3 gives a brief overview on fractals and their properties.

Chapter 4 introduces dynamical systems. To stress the close relationship of fractal geometry and chaos theory, dynamical systems are defined as (iterated) transformations in a geometrical space.

Chapter 5 introduces a formal definition of chaos and continues with a description of its separate elements.

Chapter 6 provides the theory behind some well known fractals, i.e. Julia sets and Mandelbrot sets.

In chapter 7, a special kind of dynamical system is introduced, namely the two-dimensional affine transformation. It is this transformation which forms the basis for operations on geometrical two-dimensional images.

Chapter 8 introduces an Iterated Function System (IFS) as a set of affine transformations together with a set of associated probabilities. Furthermore, IFS-decoding is treated from an algorithmic viewpoint as well as from a viewpoint of finite automata.

Chapter 9 is basically an introduction to the next chapter and deals with measures, Markov operators and moment theory.

Chapter 10 addresses the inverse problem of chapter 8, i.e. IFS-encoding. This topic is approached in a fundamental way. Computational time complexity is considered, as well as an approach to tackle the problem in closed form, based on moment theory. Furthermore, the IFS-encoding problem is proven to be NP-complete. Several quality measures are discussed, and the Pixset distance is introduced as a practical distance function, which is also proven to be a metric.

Chapter 11 deals about a genetic approach to solve the IFS-encoding problem. The principles of genetic algorithms are explained and the chromosome encoding process is illustrated. The developed genetic operators are discussed as well as the evaluation function. Finally, some results obtained with the genetic algorithm on the IFS-encoding problem are given.

Chapter 12 contains some conclusions and recommendations. Separate recommendations are made on the user interface, on the genetic algorithm as well as on IFS-encoding.

Although there are spaces in which I find nothing which excites my senses, I must not from that conclude that these spaces contain no body.

Descartes (1596 - 1650) [ADL77]

Chapter 2

Metric Spaces

Geometry is concerned with the description, classification, analysis and observation of subsets of metric spaces. Metric spaces are usually, but not always, of an inherently "simple" geometrical character but the subsets of these spaces are typically geometrically "complicated". There are a number of general properties of such subsets, which occur over and over again, which are very basic, and which form part of the vocabulary for describing fractal sets and other subsets of metric spaces. That is the reason that those properties will be investigated.

2.1. Basic definitions

Definition 2.1. A ***space*** X is a set. The ***points*** of X are the elements of the set [BAR88a]. ♦

Example 2.1. Let $X = \mathbf{R}^2$ be the ***Euclidian plane***, the coordinate plane of calculus. A ***point*** or ***vector*** $x \in X$ can be represented in several equivalent ways:

$$x = (x_1,x_2) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \vec{x}$$

★

Example 2.2. Let $X = \mathbf{C}$ be the ***complex plane*** (also called vector space), where any point $x \in X$ is represented by the complex number $x = x_1 + ix_2$, where $i = \sqrt{-1}$, for some pair of real numbers $x_1,x_2 \in \mathbf{R}$. Any pair of numbers $x_1,x_2 \in \mathbf{R}$ determine a point of $\mathbf{C}$. It is obvious that $\mathbf{C}$ is essentially the same as $\mathbf{R}^2$, but there is an implied distinction. In $\mathbf{C}$ we can multiply two points x and y and obtain a new point in $\mathbf{C}$:

$$xy = (x_1 + ix_2)(y_1 + iy_2) = (x_1y_1 - x_2y_2) + i(x_1y_2 + x_2y_1)$$

★

Example 2.3. Let $X = \Sigma$ be the ***code space*** on N symbols, where $N \in \mathbf{N}$. The symbols are integers $\{0,1,2,\ldots,N-1\}$. A typical point in Σ is a word such as $x = 2\ 17\ 0\ 0\ 1\ 21\ 15\ (N-1)\ 30 \ldots$ There are infinitely many symbols in this sequence. In general, for a given element $x \in \Sigma$ we can write:

$$x = x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7\, x_8 \ldots, \text{ where } x_i \in \{0,1,2,\ldots,N\}$$

★

Suppose that X_1 and X_2 are spaces. Then a new space can be created by taking the Carthesian product of X_1 and X_2, denoted by $X_1 \times X_2$. For example, $\mathbf{R}^2$ is the Carthesian product of $\mathbf{R}$ and $\mathbf{R}$, so: $\mathbf{R}^2 = \mathbf{R} \times \mathbf{R}$.

Definition 2.2. A ***metric space*** (X,d) is a space X together with a real-valued function d: $X \times X \rightarrow \mathbf{R}$, which measures the ***distance*** between pairs of points x and y in X. The function d is required to obey the following four axioms [BAR88a]:

(i) $(\underline{A}\, x : x \in X : d(x,x) = 0)$;

(ii) $(\underline{A}\, x,y : x,y \in X : d(x,y) = d(y,x))$;

(iii) $(\underline{A}\, x,y : x,y \in X, x \neq y : 0 < d(x,y) < \infty)$;

(iv) $(\underline{A}\, x,y,z : x,y,z \in X : d(x,y) \leq d(x,z) + d(z,y))$.

Such a function d is called a ***metric***. ♦

A distance function will not automatically be a metric, but a metric always defines a distance. The concept of shortest paths between points in a space, ***geodesics***, is dependent on the metric. The metric may determine a ***geodesic structure*** of the space. Geodesics on a sphere are great circles, geodesics in the Euclidian plane are straight lines. The following two metrics in the space $X = \mathbf{R}^2$ are frequently used:

Euclidian metric: $d_E(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$

Manhattan metric: $d_M(x,y) = |x_1 - y_1| + |x_2 - y_2|$

Definition 2.3. Two metrics d_1 and d_2 on a space X are ***equivalent*** if there exist constants $0 < c_1 < c_2 < \infty$ such that [BAR88a]:

$(\underline{A}\, x,y : x,y \in X \times X : c_1 d_1(x,y) \leq d_2(x,y) \leq c_2 d_1(x,y))$ ♦

The underlying concept is that equivalent metrics give the same notion of which points are close together and which are far apart. For example the Manhattan metric and the Euclidian metric are equivalent on $\mathbf{R}^2$ since:

$(\underline{A}\, x,y : x,y \in \mathbf{R}^2 \times \mathbf{R}^2 : 1 \cdot d_E(x,y) \leq d_M(x,y) \leq \sqrt{2} \cdot d_E(x,y))$

and also:

$(\underline{A}\, x,y : x,y \in \mathbf{R}^2 \times \mathbf{R}^2 : ½\sqrt{2} \cdot d_M(x,y) \leq d_E(x,y) \leq 1 \cdot d_M(x,y))$

Definition 2.4. Two metric spaces (X_1,d_1) and (X_2,d_2) are ***equivalent*** if there is a function $g: X_1 \to X_2$ which is one-to-one and onto (i.e. it is invertible), such that the metric $\underline{d}_1$ on X_1 is equivalent to d_1, where $\underline{d}_1$ is defined by [BAR88a]:

$$(\underline{A}\, x,y : x,y \in X_1 : \underline{d}_1(x,y) = d_2(g(x),g(y))) \qquad \blacklozenge$$

It follows that $(\mathbf{C},d_E)$ and $(\mathbf{R}^2,d_E)$ are equivalent metric spaces, since $g_1 : \mathbf{R}^2 \to \mathbf{C}$ and $g_2 : \mathbf{C} \to \mathbf{R}^2$ can be respectively defined as:

$$(\underline{A}\, x,y : x \in \mathbf{R}^2, y \in \mathbf{C} : y = x_1 + ix_2) \text{ and}$$

$$(\underline{A}\, x,y : x \in \mathbf{R}^2, y \in \mathbf{C} : x = (\mathrm{Re}(y),\mathrm{Im}(y)))$$

Definition 2.5. A sequence $\{x_n : n = 1,2,3,\ldots,\infty\}$ of points in a metric space (X,d) is called a ***Cauchy sequence*** if, for any given number $\epsilon > 0$, there is an $N \in \mathbf{N}$ such that [GUL92]:

$$(\underline{A}\, n,m : n,m \geq N : d(x_n,x_m) < \epsilon) \qquad \blacklozenge$$

In other words, the further along the sequence one goes, the closer together become the points in the sequence. However just because a sequence of points moves closer together as one goes along the sequence, does not imply that the sequence is approaching a fixed point. Perhaps it is trying to approach a point that isn't there.

Definition 2.6. A sequence $\{x_n : n = 1,2,3,\ldots,\infty\}$ of points in a metric space (X,d) is said to ***converge*** to a point $x \in X$ if, for any given number $\epsilon > 0$, there is an $N \in \mathbf{N}$ such that [GUL92]:

$$(\underline{A}\, n : n \geq N : d(x_n,x) < \epsilon)$$

In this case the point $x \in X$, to which the sequence converges, is called the ***limit*** of the sequence and we use the notation:

$$x = \lim_{n \to \infty} x_n \qquad \blacklozenge$$

Clearly, if a sequence of points $\{x_n : n = 1,2,3,\ldots,\infty\}$ in a metric space (X,d) converges to a point $x \in X$, then $\{x_n : n = 1,2,3,\ldots,\infty\}$ is a Cauchy sequence.

Definition 2.7. A metric space (X,d) is ***complete*** if every Cauchy sequence $\{x_n : n = 1,2,3,\ldots,\infty\}$ in X has a limit $x \in X$ [BAR88a]. ♦

In other words, in the space X actually exists a point x to which the Cauchy sequence is converging. This point x is of course the limit of the sequence. For example $(\mathbf{R}^2,d_E)$ and $(\mathbf{C},d_M)$ are complete metric spaces, but also (by equivalence) are $(\mathbf{R}^2,d_M)$ and $(\mathbf{C},d_E)$.

Definition 2.8. Let $S \subset X$ be a subset of a metric space (X,d). A point $x \in X$ is called a ***limit point*** of S if there is a sequence $\{x_n : n = 1,2,3,\ldots,\infty\}$ of points $x_n \in S \backslash \{x\}$ such that the limit of this sequence equals x [BAR88a]. ♦

Definition 2.9. Let $S \subset X$ be a subset of a metric space (X,d). The ***closure*** of S, denoted by S_c, is defined to be $S_c = S \cup \{\mathit{limitpoints\ of\ } S\}$. S is ***closed*** if it contains all of its limit points, i.e. $S = S_c$. S is ***perfect*** if it is equal to the set of all its limit points [BAR88a]. ♦

Definition 2.10. Let $S \subset X$ be a subset of a metric space (X,d). S is ***compact*** if every infinite sequence $\{x_n : n = 1,2,3,\ldots,\infty\}$ in S contains a subsequence having a limit in S [BAR88a]. ♦

Definition 2.11. Let $S \subset X$ be a subset of a metric space (X,d). S is ***R-bounded*** if there is a point $a \in X$ and a number $R>0$ such that [BAR88a]:

$$(\underline{\mathrm{A}}\, x : x \in X : d(a,x) < R)$$ ♦

Definition 2.12. Let $S \subset X$ be a subset of a metric space (X,d). S is ***totally bounded*** if, for each $\epsilon>0$, there is a finite set of points $\{y_1,y_2,\ldots,y_n\} \subset S$ such that whenever $x \in X$, $d(x,y_i)<\epsilon$ for some $y_i \in \{y_1,y_2,\ldots,y_n\}$. This set of points $\{y_1,y_2,\ldots,y_n\}$ is called an ***ϵ-net*** [BAR88a]. ♦

Theorem 2.1. Let (X,d) be a complete metric space and let $S \subset X$. Then S is compact if and only if it is closed and totally bounded.

Proof. See [BAR88a]. ■

Definition 2.13. Let $S \subset X$ be a subset of a metric space (X,d). S is ***open*** if for each $x \in X$ there is an $\epsilon>0$ such that $B(x,\epsilon) = \{y \in X : d(x,y) \leq \epsilon\} \subset S$, where $B(x,\epsilon)$ is a closed ball of radius $\epsilon>0$ centered at x (see fig. 2.1) [BAR88a]. ♦

Definition 2.14. Let $S \subset X$ be a subset of a metric space (X,d). A point $x \in X$ is a ***boundary point*** of S if for every number $\epsilon>0$, $B(x,\epsilon)$ contains a point in $X \backslash S$ and a point in S. The set of all boundary points of S is called the ***boundary*** of S, and is denoted by ∂S (see fig. 2.1) [BAR88a]. ♦

Definition 2.15. Let $S \subset X$ be a subset of a metric space (X,d). A point $x \in S$ is called an ***interior point*** of S if there is a number $\epsilon>0$ such that $B(x,\epsilon) \subset S$. The set of all interior points of S is called the ***interior*** of S, and is denoted by S^0 (see fig. 2.1) [BAR88a]. ♦

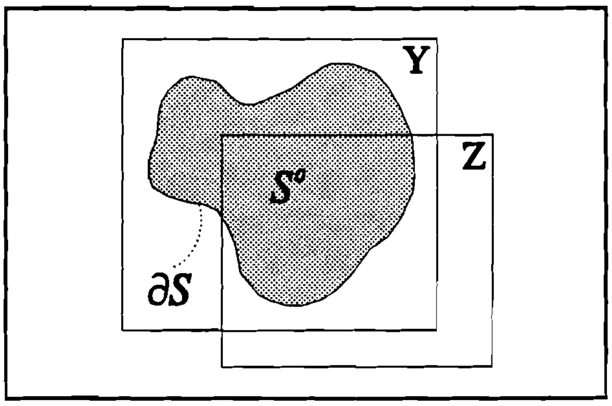


Figure 2.1. The interior of S is a closed set in the metric space (Y,d_E) but an open set in the metric space (Z,d_E).

2.2. The metric space $(\mathcal{H}(X),h)$

Definition 2.16. Let (X,d) be a complete metric space. Then $\mathcal{H}(X)$ denotes the space whose points are compact subsets of X, other than $\varnothing$ [BAR88a]. ♦

Definition 2.17. Let (X,d) be a complete metric space, and let $A,B \in \mathcal{H}(X)$. Define:

$$d(x,B) = (\ \underline{\text{Min}} : x \in X,\ y \in B : d(x,y)\ )$$

Then $d(x,B)$ is the ***distance from the point x to the set B*** [BAR88a]. In the same way we can define the ***distance from the set A to the point y*** as:

$$d(A,y) = (\ \underline{\text{Max}} : x \in A,\ y \in X : d(x,y)\ )$$ ♦

Definition 2.18. Let (X,d) be a complete metric space, and let $A,B \in \mathcal{H}(X)$. The ***distance from the set A to the set B*** is defined as [BAR88a,GUL92]:

$$d(A,B) = (\ \underline{\text{Max}} : x \in A : d(x,B)\ )$$ ♦

Example 2.4. Let $(X,d) = (\mathbb{R},d_M)$, $x = ½$ and

$$B = \{\ x_n = 3 + \frac{n(-1)^n}{n^2+1} : n = 1,2,3,\ldots\ \} \cup \{3\}$$

Then we can calculate B and $d_M(x,B)$ as follows:

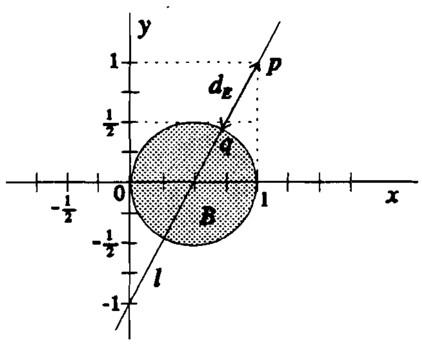
$$B = \{2\tfrac{1}{2},3\tfrac{2}{5},2\tfrac{7}{10},3\tfrac{4}{17},2\tfrac{21}{26},3\tfrac{6}{37},\ldots\} \cup \{3\}$$

$$d_M(x,B) = (\ \underline{\text{Min}} : y \in B : d(x,y)\ ) = d_M(½,2½) = |½ - 2½| = 2$$ ★

__*Example 2.5.*__ Let $(X,d) = (\mathbf{R}^2,d_E)$ and $p,q \in \mathbf{R}^2$. The point p is given by $(1,1)$ and B is a closed disk of radius ½ centered at the point (½,0). We can calculate $d_E(p,B)$ as follows (see figure):

$$d_E(p,B) = (\ \underline{\text{Min}} : q \in B : d_E(p,q)\ )$$

$$= \{\ d_E((1,1),q) : q \in B\ \}$$



q is the intersection point of the line l: $y = 2x - 1$ and the disk B: $y^2 + (x - ½)^2 = ¼$, so we can solve:

$$(2x - 1)^2 + (x - ½)^2 = ¼$$

$$\Leftrightarrow\ 5x^2 - 5x + 1 = 0$$

$$x_{1,2} = \frac{5 \pm \sqrt{25 - 20}}{10} \Rightarrow x_1 \approx 0.72,\ x_2 \approx 0.28$$

So the intersection point is given by $q \approx (0.72,0.45)$ and now we can continue with:

$$d_E((1,1),(0.72,0.45)) = \sqrt{(1 - 0.72)^2 + (1 - 0.45)^2} \approx 0.62$$ ★

__*Example 2.6.*__ Let $(X,d) = (\mathbf{R}^2,d_E)$ on a roadmap. Define *FRANCE* and *BRITAIN* as subsets of X, and let *Calais,Marseille,Paris* ∈ *FRANCE*, and let *Dover,Inverness,London* ∈ *BRITAIN*. A travelling salesman travels between *FRANCE* and *BRITAIN* and the starting and terminating points of his travels are always big cities. First suppose he travels back and forth between *London* and *Paris*. Then he must travel a fixed distance of:

$$d_E(London,Paris) = d_E(Paris,London) \approx 366\ [km]$$

If our travelling salesman travels from *London* to *FRANCE* he must at least travel a distance of:

$$d_E(London,FRANCE) = (\ \underline{\text{Min}} : y \in FRANCE : d_E(London,y)\ )$$
$$= d_E(London,Calais) \approx 154\ [km]$$

In the same way we obtain the distances from *FRANCE* to *London*, from *Paris* to *BRITAIN* and from *BRITAIN* to *Paris*:

$$d_E(FRANCE,London) = (\ \underline{\text{Max}} : x \in FRANCE : d_E(x,London)\ )$$
$$= d_E(Marseille,London) \approx 1049\ [km]$$

$$d_E(Paris,BRITAIN) = (\ \underline{\text{Min}} : y \in BRITAIN : d_E(Paris,y)\ )$$
$$= d_E(Paris,Dover) \approx 280\ [km]$$

$$d_E(BRITAIN,Paris) = (\ \underline{\text{Max}} : x \in BRITAIN : d_E(x,Paris)\ )$$
$$= d_E(Inverness,Paris) \approx 1112\ [km]$$

Suppose that the only thing we know is that the travelling salesman travels back and forth between *BRITAIN* and *FRANCE*, then he must travel at most distances of:

$$\begin{aligned} d_E(BRITAIN,FRANCE) &= (\ \underline{\mathrm{Max}} : x \in BRITAIN : d_E(x,FRANCE)\ ) \\ &= d_E(Inverness,FRANCE) \\ &= (\ \underline{\mathrm{Min}} : y \in FRANCE : d_E(Inverness,y)\ ) \\ &= d_E(Inverness,Calais) \approx 582\ [km] \end{aligned}$$

$$\begin{aligned} d_E(FRANCE,BRITAIN) &= (\ \underline{\mathrm{Max}} : x \in FRANCE : d_E(x,BRITAIN)\ ) \\ &= d_E(Marseille,BRITAIN) \\ &= (\ \underline{\mathrm{Min}} : y \in BRITAIN : d_E(Marseille,y)\ ) \\ &= d_E(Marseille,Dover) \approx 962\ [km] \end{aligned}$$ ★

From the above examples it becomes clear that as soon as sets are involved, distances such as $d_E(x,B)$ and $d_E(A,B)$ are not metrics, since not all rules of definition 2.2 are obeyed. What we need is a robust distance function on sets, which is also a metric.

Definition 2.19. Let (X,d) be a complete metric space. Then the ***Hausdorff distance*** between points A and B in $\mathcal{H}(X)$, which are sets in X, is defined by [BAR88a]:

$$h(A,B) = (\ \underline{\mathrm{Max}} : A,B \in \mathcal{H}(X) : (d(A,B),d(B,A))\ )$$ ♦

The Hausdorff distance appears to be a robust distance function for calculations on sets. The association of the name Hausdorff refers to the German mathematician Felix Hausdorff, who was one of the pioneers in the area of geometry called topology during the early part of this century.

Theorem 2.2. h is a metric on the space $\mathcal{H}(X)$.

Proof. We have to prove that h obeys the four axioms of definition 2.2.

(i) To be proven: $(\ \underline{\mathrm{A}}\ A : A \in \mathcal{H}(X) : h(A,A) = 0\ )$.

Let $A \in \mathcal{H}(X)$, then

$$\begin{aligned} & h(A,A) \\ = & \ \langle \text{ definition 2.19 } \rangle \\ & (\ \underline{\mathrm{Max}} : A \in \mathcal{H}(X) : (d(A,A),d(A,A))\ ) \\ = & \ \langle \text{ definition 2.18 } \rangle \\ & (\ \underline{\mathrm{Max}} : x \in A : d(x,A)\ ) \\ = & \ \langle \text{ definition 2.17 } \rangle \\ & (\ \underline{\mathrm{Max}} : x \in A : (\underline{\mathrm{Min}} : y \in A : d(x,y)\ )\ ) \\ = & \ \langle \text{ calculus } \rangle \\ & 0 \end{aligned}$$

(ii) To be proven: ($\underline{\mathrm{A}}\ A,B : A,B \in \mathcal{H}(X) : h(A,B) = h(B,A) = 0$).

Let $A,B \in \mathcal{H}(X)$, then

$h(A,B)$
$= \langle$ definition 2.19 $\rangle$
($\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X) : (d(A,B),d(B,A))$)
$= \langle$ symmetry of $\underline{\mathrm{Max}}$ operator $\rangle$
($\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X) : (d(B,A),d(A,B))$)
$= \langle$ definition 2.19 $\rangle$
$h(B,A)$

(iii) To be proven: ($\underline{\mathrm{A}}\ A,B : A,B \in \mathcal{H}(X), A \neq B : 0 < h(A,B) < \infty$).

Let $A,B \in \mathcal{H}(X)$ and $A \neq B$, then

$\Rightarrow \langle$ A and B are compact $\rangle$
($\underline{\mathrm{E}}\ a,b : a \in A,\ b \in B : h(A,B) = d(a,b)$)
$\Rightarrow \langle$ theorem 2.1, definition 2.12 $\rangle$
$0 \leq h(A,B) < \infty$
$\Rightarrow \langle A \neq B \rangle$
($\underline{\mathrm{E}}\ a : a \in A : a \notin B$) $\wedge$ ($0 \leq h(A,B) < \infty$)
$\Rightarrow \langle$ definitions 2.17, 2.18, 2.19 $\rangle$
$0 < d(a,B) \leq d(A,B) \leq h(A,B) < \infty$
$\Rightarrow \langle$ calculus $\rangle$
$0 < h(A,B) < \infty$

iv) To be proven: ($\underline{\mathrm{A}}\ A,B,C : A,B,C \in (X) : h(A,B) \leq h(A,C) + h(C,B)$).

Let $A,B,C \in \mathcal{H}(X)$, $a \in A$, $b \in B$, $c \in C$, $a \notin B$, then

$\Rightarrow \langle$ proof part (iii) $\rangle$
$0 < d(a,B) \leq d(A,B) \leq h(A,B)$
$\Rightarrow \langle$ definition 2.17 $\rangle$
$d(a,B) \leq d(a,b)$
$\Rightarrow \langle$ distances are not negative $\rangle$
$d(a,B) \leq d(a,b) + d(c,b)$
$\Rightarrow \langle$ $a \notin B$, triangle inequality $\rangle$
$d(a,B) \leq d(a,c) + d(c,B) \leq d(a,c) + h(C,B)$
$\Rightarrow \langle$ definitions 2.17, 2.18 $\rangle$
$d(a,B) \leq d(a,C) + h(C,B) \leq h(A,C) + h(C,B)$
$\Rightarrow \langle a \in A \rangle$
$d(A,B) \leq h(A,C) + h(C,B)$
$\Rightarrow \langle$ similarly $\rangle$
$d(B,A) \leq h(A,C) + h(C,B)$
$\Rightarrow \langle$ calculus $\rangle$
($\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X) : (d(A,B),d(B,A))$) $\leq h(A,C) + h(C,B)$

So $(\mathcal{H}(X),h)$ is a complete metric space. ■

Because the Hausdorff distance is a metric on $\mathcal{H}(X)$, it is also referred to as the ***Hausdorff metric***. The space $(\mathcal{H}(X),h)$ is referred to as ***fractal space***.

__*Example 2.7.*__ Consider example 2.5 once more. Now we know that the travelling salesman, who travels back and forth between *BRITAIN* and *FRANCE*, has to travel a distance of at most:

$h_E(BRITAIN,FRANCE)$

$= h_E(FRANCE,BRITAIN)$

$= (\underline{\text{Max}} : BRITAIN,FRANCE \in \mathcal{H}(X) : (d_E(BRITAIN,FRANCE),d_E(FRANCE,BRITAIN))$

$= (\underline{\text{Max}} : : (582\ [km],962\ [km]))$

$= 962\ [km]$ ★

Some properties of set distances which can easily been proven are [BAR89]:

(i) $B \subset C \Rightarrow d(A,C) \leq d(A,B)$;

(ii) $d(A \cup B,C) = (\underline{\text{Max}} : A,B,C \in \mathcal{H}(X) : (d(A,C),d(B,C)))$;

(iii) $d(A \cup B,C \cup D) \leq (\underline{\text{Max}} : A,B,C,D \in \mathcal{H}(X) : (d(A,C),d(B,D)))$;

(iv) $h(A \cup B,C \cup D) \leq (\underline{\text{Max}} : A,B,C,D \in \mathcal{H}(X) : (h(A,C),h(B,D)))$.

Nature flies from the infinite, for the infinite is unending or imperfect, and Nature ever seeks an end.

Aristotle (384 - 322 B.C.) [ADL77]

Chapter 3

Fractals

Fractal geometry is concerned with the description, classification, analysis and observation of subsets of the metric space $(\mathcal{H}(X),h)$. When we want to measure the length of a coastline, for instance of Britain, we will not succeed in finding a fixed length, because the length depends on the scale we are using. Suppose we use a unity of measure or ***yardstick*** ϵ of 1 [*mm*] to measure the coastline length of Britain on scaled maps. When we use a $1{:}10^4$ map, we will find a coastline length that exceeds the length of 100 times the measured length on a $1{:}10^6$ map. Mandelbrot found that the approximate coastline length of Britain versus the inverse yardstick length ϵ^{-1}, as plotted on a double logarithmic scale, fall on a straight line when ϵ tends to zero (see fig. 3.3). The reason for this is the fact that the amount of detail is scale-invariant. This appears to be not only the case for coastlines, but also for a lot of other natural geometries, like mountains, clouds and trees [MAN83].

3.1. Fractal dimension

Structures in classical Euclidian geometry have an integer dimension. For example a point is zero-dimensional (0*D*), a line is one-dimensional (1*D*), a plane is two-dimensional (2*D*) and a cube is three-dimensional (3*D*). We could also say that the ***topological dimension*** D of these structures has an integer value, hence: D(point) = 0, D(line) = 1, D(plane) = 2 and D(cube) = 3. If we look at fig. 3.1 we see three geometrical structures: a ***Cantor set***, a ***Sierpinski triangle*** and a ***Menger sponge***.

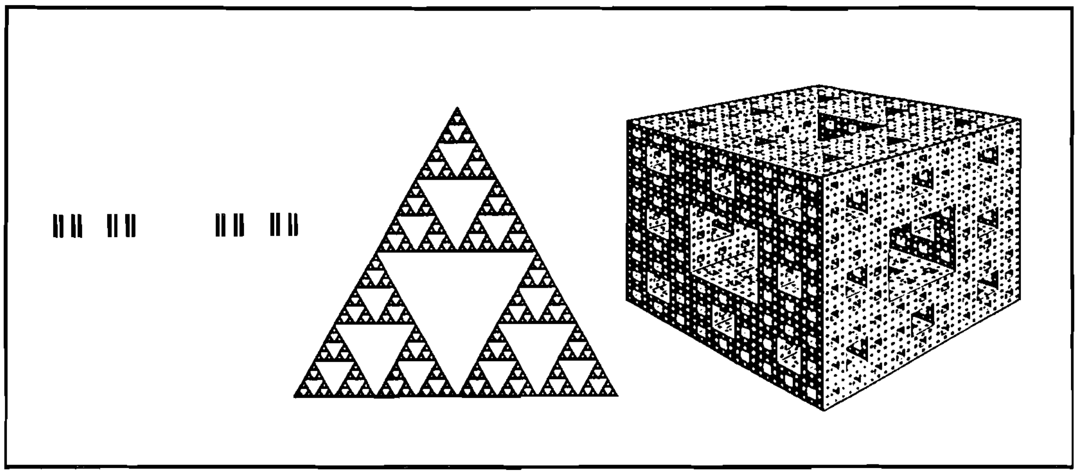


Figure 3.1. From left to right: a Cantor set, a Sierpinski triangle and a Menger sponge [GLE87,MAN83].

By looking at fig. 3.1 we could intuitively say that the Cantor set is more than a point, but less than a line, the Sierpinski triangle is more than a line, but less than a plane and the Menger sponge is more than a plane, but less than a cube. In other words:

$$0 < D(\mathit{Cantor\ set}) < 1$$
$$1 < D(\mathit{Sierpinski\ triangle}) < 2$$
$$2 < D(\mathit{Menger\ sponge}) < 3$$

This means that the topological dimension of those structures is not integer, but real valued, or a fractional integer value. Therefore Mandelbrot defined those geometrical structures as ***fractal structures*** or ***fractals*** [MAN83].

Definition 3.1. Let $(\mathbf{R}^m, d_E)$ with $m \in \mathbf{N}$ denote a complete metric space and let $S \in \mathcal{H}(\mathbf{R}^m)$ be a nonempty subset of $\mathbf{R}^m$. For each $\epsilon > 0$ we denote by $N(S,\epsilon)$ the smallest number of m-dimensional just touching boxes of side length ϵ, required to completely cover S. By an ***m-dimensional box*** in $\mathbf{R}^m$ we mean a line segment if $m=1$, a square if $m=2$ and a cube if $m=3$. Then the ***capacity dimension*** or ***box (counting) dimension*** of S is given by [BAR88a,GUL92]:

$$D_C(S) = \lim_{\epsilon \to 0} \frac{\ln N(S,\epsilon)}{\ln(1/\epsilon)}$$

If $D_C(S)$ exists then $N(S,\epsilon)$ ***scales*** as $(1/\epsilon)^D$. Moreover, if $D_C(S)$ exists and is non-integer, then S is said to have ***fractal dimension***. In that case $D_C(S)$ is called the fractal dimension of S and S is referred to as a fractal. ♦

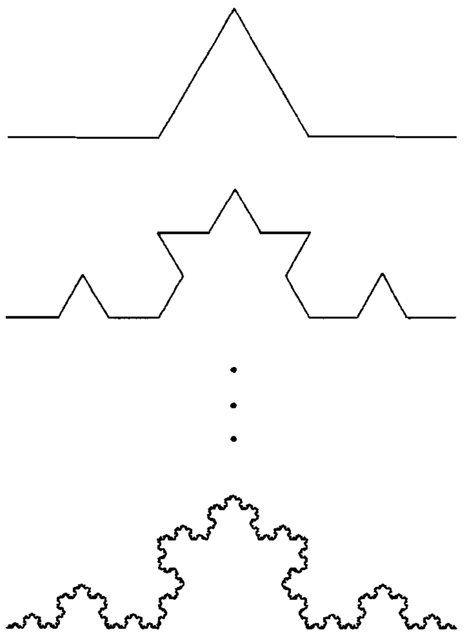
For a line segment of length N, a square with sidelength N and a cube with sidelength N, the capacity dimension yields respectively 1, 2 and 3. So those structures are not fractal. One way to construct fractals is by means of a context-free grammar. In that case the building process makes use of ***base-elements***, ***initiators*** and ***generators*** [KAA87].

Definition 3.2. Define (conform [LEW81]) a ***context-free grammar*** G as a 4-tuple (V,Σ,R,S) where V is the alphabet, Σ is the set of terminals (a subset of V), R is the set of (production) rules and S is the startsymbol (an element of V). A ***fractal generating context-free grammar*** F can accordingly be defined as:

$$\begin{aligned}
F &= (V,\Sigma,R,S) \text{ with} \\
V &= \{\ \mathit{initiator, generator, base\text{-}element}\ \} \cup \Sigma \\
\Sigma &= \{\ \mathit{fractal}\ \} \\
R &= \{\ \mathit{initiator} \rightarrow \mathit{base\text{-}element(s)}, \\
&\qquad \mathit{base\text{-}element} \rightarrow \mathit{generator}, \\
&\qquad \mathit{generator} \rightarrow \mathit{base\text{-}element(s)}, \\
&\qquad \mathit{generator} \rightarrow \mathit{fractal}\ \} \\
S &= \mathit{initiator}
\end{aligned}$$

♦

<u>*Example 3.1.*</u> Let the initiator be equal to the base-element, i.e. a line segment with a length of 1 [*unit*], let the generator be the 4-line segment of step 1 below and let the fractal be a ***Von Koch curve***. The derivation for n steps in F from S is the sequence:



	N	ϵ	$L=N\epsilon$
step 0	1	1	1
$F \Downarrow$			
step 1	4	$\frac{1}{3}$	$\frac{4}{3}$
$F \Downarrow$			
step 2	4^2	$(\frac{1}{3})^2$	$(\frac{4}{3})^2$
$F \Downarrow$			
⋮			
$F \Downarrow$			
step n	4^n	$(\frac{1}{3})^n$	$(\frac{4}{3})^n$

In the above derivation is N the amount of base-elements, each of length ϵ, and is L the total length of all base-elements the structure contains, so $L = N\epsilon$.

The capacity dimension of the Von Koch curve (the structure that appears after $n \to \infty$ steps) can be calculated as follows:

$$D_C(S) = \lim_{\epsilon \to 0} \frac{\ln N(\textit{Von Koch curve},\epsilon)}{\ln(1/\epsilon)} = \lim_{n \to \infty} \frac{\ln 4^n}{\ln(\frac{1}{3})^{-n}} = \frac{\ln 4}{\ln 3} \approx 1.26$$

Hence the Von Koch curve has fractal dimension. ★

Now we can also determine the capacity dimensions of the Cantor set, the Sierpinski triangle and the Menger sponge (see fig. 3.1). We follow the same procedure as in example 3.1 (see fig. 3.2). The generator of the Cantor set is made of N=2 base-elements (lines). The successive derivation steps are obtained by contractions of ratio ϵ=⅓, so the capacity dimension is:

$$D_C(\textit{Cantor set}) = \lim_{n \to \infty} \frac{\ln 2^n}{\ln 3^n} = \frac{\ln 2}{\ln 3} \approx 0.63$$

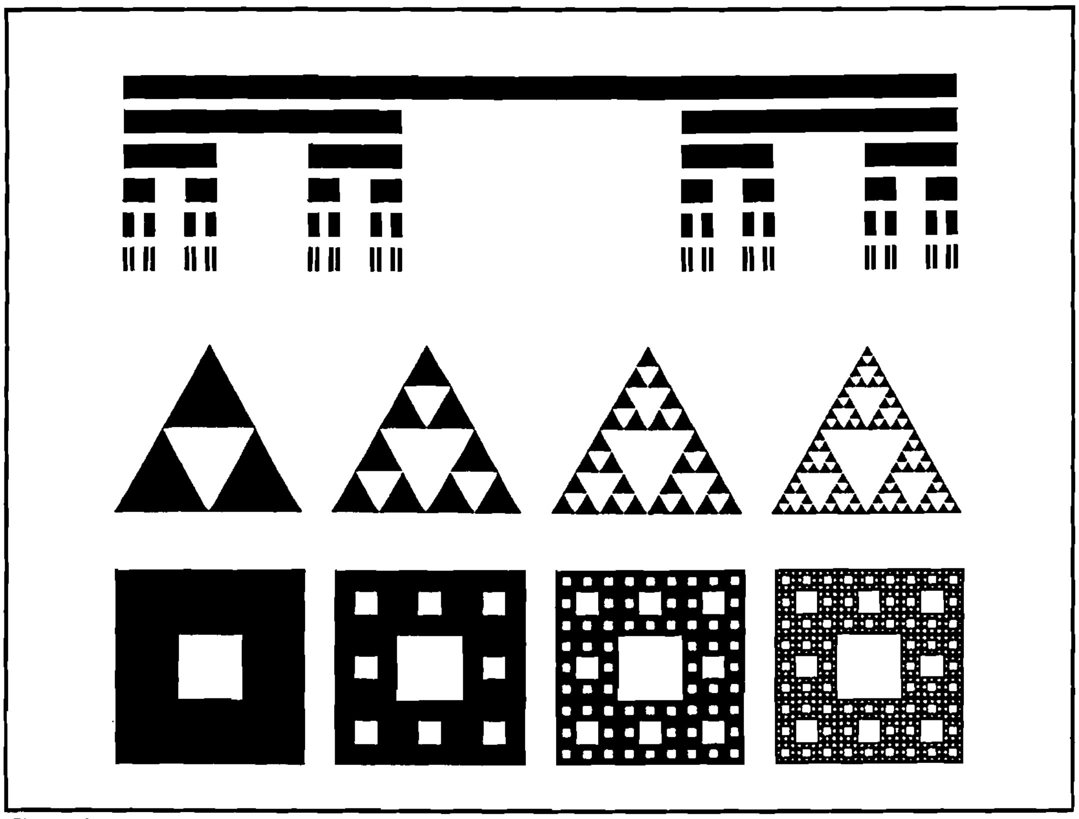


Figure 3.2. Derivation of the Cantor set, the Sierpinski triangle and the Menger sponge (front view).

The generator of the Sierpinski triangle is made of N=3 base-elements (triangle planes). The successive derivation steps are obtained by contractions of ratio ϵ=½, so the capacity dimension is:

$$D_C(\textit{Sierpinski triangle}) = \lim_{n\to\infty} \frac{\ln 3^n}{\ln 2^n} = \frac{\ln 3}{\ln 2} \approx 1.58$$

The generator of the Menger sponge is made of N=20 base-elements (cubes). The successive derivation steps are obtained by contractions of ratio ϵ=⅓, so the capacity dimension is:

$$D_C(\textit{Menger sponge}) = \lim_{n\to\infty} \frac{\ln 20^n}{\ln 3^n} = \frac{\ln 20}{\ln 3} \approx 2.73$$

And indeed, we see that all structures have fractal dimension and:

$$0 < D_C(\textit{Cantor set}) \approx 0.63 < 1$$
$$1 < D_C(\textit{Sierpinski triangle}) \approx 1.58 < 2$$
$$2 < D_C(\textit{Menger sponge}) \approx 2.73 < 3$$

Consider example 3.1 once more. If we had choosen the initiator to be a triangle in case of a line segment, we would have obtained a geometric structure which is known as the ***Von Koch snowflake*** and which has the remarkable property that its finite surface area has a boundary of infinite length [GIP90a].

There is another interpretation of the fractal dimension, namely the slope of the log-log plot of the amount of base elements $N(\epsilon)$ versus the inverse (yardstick) length ϵ^{-1}. If we make those plots for the structures of fig. 3.1 and example 3.1, as well as for the coastline of Britain, we obtain fig. 3.3.

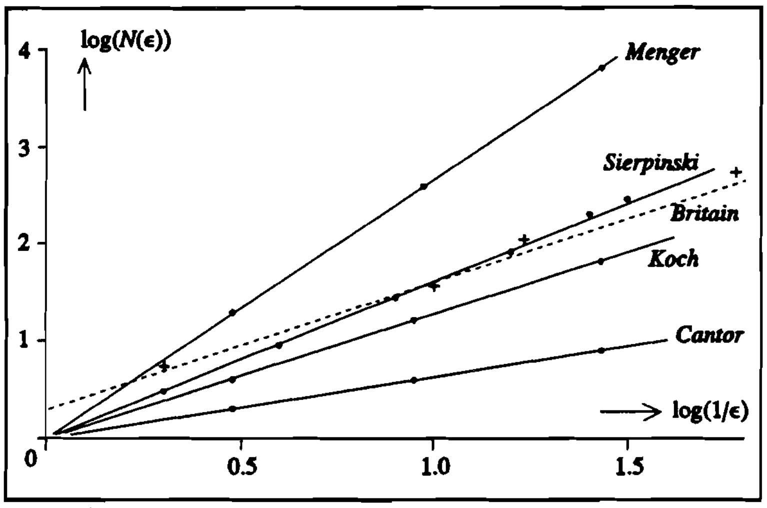


Figure 3.3. Log-log plot of amount of covering segments $N(\epsilon)$ versus the inverse yardstick length ϵ^{-1} for some earlier mentioned structures.

From fig. 3.3 we conclude that the fractal dimension of coastline of Britain is approximately 1.31. Several algorithms for computing fractal dimensions of images are developed [CRE89,DUB87, KEL87,VEP89]. In [KAN88] a ***fractal matrix model*** is introduced, which should give a better characterization of the fractal properties of structures than the fractal dimension does.

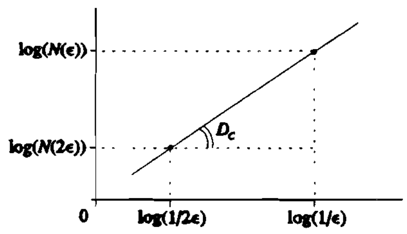
Example 3.2. Part of this graduation work was the development of a user interface (IFS-CODEC) to provide several facilities which are related with fractal geometry (see appendix A). One option is to calculate fractal dimensions of geometrical structures. The IFS-CODEC algorithm does this as follows. The structure is first covered with $N(\epsilon)$ boxes of sidelength ϵ. Then the structure is covered with $N(2\epsilon)$ boxes of sidelength 2ϵ. Then the fractal dimension is calculated as the slope of the line through the points $(\log(1/2\epsilon),\log(N(2\epsilon)))$ and $(\log(1/\epsilon),\log(N(\epsilon)))$, as follows (see figure below):

$$D_c \approx \frac{\log(N(\epsilon)) - \log(N(2\epsilon))}{\log(1/\epsilon) - \log(1/2\epsilon)} = \frac{\log\left(N(\epsilon)N(2\epsilon)^{-1}\right)}{\log 2}$$

In practice we choose $\epsilon = 1$ [*pixel*]. With the IFS-CODEC algorithm we obtained for the structures given in appendix C the fractal dimensions as given in table 3.1.

Table 3.1. Approximated fractal dimensions of structures of appendix C.

structure	D_c	structure	D_c	structure	D_c
chain	1.48	koch	1.20	tree	1.46
dragon	1.92	leaf	1.96	triangle	1.47
fern	1.78	rect	1.99	mandel	1.92
france	1.98	square	1.87	twig	1.17



★

3.2. Hausdorff dimension

The Hausdorff dimension (also called Hausdorff-Besicovitch dimension) is another real number which can be used to characterize the geometrical complexity of bounded subsets of $\mathbf{R}^m$. Its definition is more complex and subtle than that of the fractal dimension. One of the reasons of its importance is the fact that it is associated with a method for comparing the "sizes" of sets whose fractal dimensions are the same. It is harder to work with than the fractal dimension and therefore less often used.

Up till now we have used m-dimensional boxes to cover a set. But in fact we may also use m-dimensional spheres. In that case the ***geometrical factor*** γ can be defined as the unit length, area or volume of the m-dimensional boxes or spheres. Thus we have $\gamma=1$ [*unity length*] for lines, $\gamma=1$ [*unity area*] for surfaces and $\gamma=1$ [*unity volume*] for cubes. In the same way we have $\gamma=\pi/4$ [*unity surface*] for disks and $\gamma=\pi/6$ [*unity volume*] for spheres.

Definition 3.4. Let $(\mathbf{R}^m,d_E)$ for $m \in \mathbf{N}$ be a complete metric space and let S be a bounded subset of this space. The covering of S is now performed by a ***covering function*** or ***measure*** [FAL90,FED89]:

$$M(S,p,\epsilon) = \sum_{i=1}^{N} \gamma\epsilon^p = \gamma\epsilon^p N(S,\epsilon) \qquad \text{for } p \in [0,\infty)$$

with p the dimension of the measure. The ***Hausdorff p-dimensional measure*** is defined as:

$$M(S,p) = \lim_{\epsilon\to 0} M(S,p,\epsilon) = \begin{cases} \infty & \text{if } p < D_H(S) \text{ and } p \in [0,\infty) \\ 0 & \text{if } p > D_H(S) \text{ and } p \in [0,\infty) \end{cases}$$

where $D_H(S)$ denotes the ***Hausdorff dimension*** (or ***Hausdorff-Besicovitch dimension***) of S. Thus:

$$D_H(S) = (\,\underline{\mathrm{Inf}} : M(S,p) = 0 : p\,) = (\,\underline{\mathrm{Sup}} : M(S,p) = \infty : p\,)$$

Moreover, the following inequality holds: $0 \le D_H(S) \le D_C(S) \le m$. ♦

The Hausdorff p-dimensional measure $M(S,p)$ as a function of $p \in [0,\infty)$ consists of only one, two or three values, i.e. zero, a finite number and infinity. In [GAR88] and [STA89] algorithms are presented for calculating the Hausdorff dimension of images.

Example 3.3. Let S be the Sierpinski triangle (see fig. 3.2). For $p=1$ we cover S in n steps with one-dimensional lines and calculate $M(S,1)$ as (#lines) • (line length). Then for $p=2$ we cover S in n steps with two-dimensional triangles and calculate $M(S,2)$ as (#triangles) • (triangle area). The result is given in table 3.2.

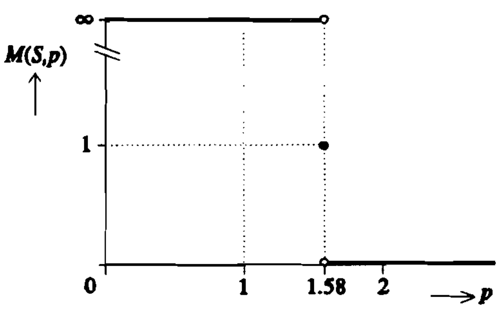
Table 3.2. Covering of S with lines and triangle areas.

step	#lines	line length	$M(S,1)$	#triangles	triangle area	$M(S,2)$
1	3^1	$(½)^0$	3.000	3^0	$(¼)^0$	1.000
2	3^2	$(½)^1$	4.500	3^1	$(¼)^1$	0.750
3	3^3	$(½)^2$	6.750	3^2	$(¼)^2$	0.563
4	3^4	$(½)^3$	10.125	3^4	$(¼)^3$	0.422
.	.	.	.	.	.	.
.	.	.	.	.	.	.
n	3^n	$(½)^{n-1}$	∞	3^{n-1}	$(¼)^{n-1}$	0.000

Let us see what happens if we choose $p = D_C(S) = (\ln 3)/(\ln 2) \approx 1.58$. Now we cover S in n steps with p-dimensional structures, actually smaller copies of S, and calculate $M(S,1.58)$. From this result (see table 3.3) we draw $M(S,p)$ as function of p (see figure below).

Table 3.3. Covering of S with copies of itself.

step	#copies	area covering	$M(S,1.58)$
1	3^1	$(⅓)^1$	1.000
2	3^2	$(⅓)^2$	1.000
3	3^3	$(⅓)^3$	1.000
4	3^4	$(⅓)^4$	1.000
.	.	.	.
.	.	.	.
n	3^n	$(⅓)^n$	1.000



It follows that in this specific case $D_H(S) = D_C(S) \approx 1.58$. ★

3.3. Self similarity

If each piece of a geometric structure is geometrically similar to the whole, then such structure is called ***self similar***. One could say that the structure is recursively built of down scaled copies of itself. The class of self similar fractals is also known as ***linear fractals*** [MAN83].

Self similar fractals have the property that their geometric structure is scale-invariant. If we keep zooming in on such fractal, we will see the same on all scales.

No more by the law of reason than by the law of nature can anything occur without a cause.

Jean-Jacques Rousseau (1712 - 1778) [ADL77]

Chapter 4

Dynamical systems

A dynamical system can informally be described as a mechanism that develops itself deterministically in time [BRO90]. Such systems are characterized by three properties [BEC90]:

(i) they are dynamic;

(ii) they are complex;

(iii) they are iterative.

Examples of dynamical systems are found in various fields, like electronics, biology, chemistry, economics, etc.

A discrete dynamical system may be thought of as a formula relating the value of a quantity at successive discrete time intervals. If the time interval is allowed to tend to zero, then the formula becomes a differential equation in the usual way [FAL90].

The problem with differential equations is the fact that they cannot always be solved in a closed form. In those cases the solution must be approximated by discretization, that is, we get a iteration equation which is initialized with a starting point, and the behaviour of the system is simulated over a certain period of time. This behaviour can either be periodic or nonperiodic, and can depend strongly on the initialized starting point. We will formally define a dynamical system in such way that it logically matches the concepts that we have introduced so far.

4.1. Transformations

Definition 4.1. Let (X,d) be a metric space. A ***transformation*** on X is a function $f : X \rightarrow X$, which assigns exactly one point $f(x) \in X$ to each point $x \in X$. If $S \subset X$ then $f(S) = \{ f(x) : x \in S \}$, and:

(i) f is ***one-to-one*** if $(\underline{A}\, x,y : x,y \in X : f(x) = f(y) \Rightarrow x = y)$;

(ii) f is ***onto*** if $(\underline{A}\, x : x \in X : (\underline{E}\, y : y \in X : f(x) = y))$;

(iii) f is ***invertible*** if f is one-to-one and onto. In this case it is possible to define an ***inverse transformation*** $f^{-1} : X \rightarrow X$ by $(\underline{A}\, x,y : x,y \in X : y = f(x) \Leftrightarrow x = f^{-1}(y))$. ♦

Definition 4.2. Let $f : X \to X$ be a transformation on a metric space (X,d). The ***forward n*th *iterate*** of f is denoted by $f^{(n)}$ and inductively defined by [LEW81]:

(i) $f^{(0)}(x) = I(x)$ (i.e. the identity);

(ii) $f^{(1)}(x) = f(x)$;

(iii) $(\underline{\mathrm{A}}\, n : n \in \mathbf{N} : f^{(n+1)}(x) = f \circ f^{(n)}(x) = f(f^{(n)}(x)))$.

If f is invertible then the ***backward n*th *iterate*** of f are transformations $f^{(-n)}$, inductively defined by:

(i) $f^{(-1)}(x) = f^{-1}(x)$;

(ii) $(\underline{\mathrm{A}}\, n : n \in \mathbf{N} : f^{(-n-1)}(x) = f^{(-1)} \circ f^{(-n)}(x) = f^{-1} \circ (f^{(n)})^{-1}(x) = f^{-1}((f^{(n)})^{-1}(x)))$;

(iii) $(\underline{\mathrm{A}}\, m,n : m,n \in \mathbb{Z} : f^{(m)} \circ f^{(n)}(x) = f^{(m+n)}(x))$.

Where $\circ$ denotes the usual composition operation. Note: $f^{(n)}(x)$ is also denoted as $f(x_n)$ or as x_{n+1} and $f^{(-n)}(x)$ is also denoted as $f^{-1}(x_n)$ or x_{n-1}. ♦

Definition 4.3. Let $f : X \to X$ be a transformation on a metric space (X,d). A point $x_f \in X$ such that $f(x_f) = x_f$ is called a ***fixed point*** of the transformation [BAR88a]. ♦

Definition 4.4. Let $f : X \to X$ be a transformation on a metric space (X,d). Then the ***Lipschitz constant*** of f is defined by [HOR90,HUT81]:

$$\mathrm{Lip}(f) = (\underline{\mathrm{A}}\, x,y : x,y \in X,\ x \neq y : \frac{d(f(x),f(y))}{d(x,y)})$$

$$= (\underline{\mathrm{Min}}\ s : s \in \mathbf{R} : (\underline{\mathrm{A}}\, t : t \in \frac{d(f(x),f(y))}{d(x,y)},\ x \neq y : s \geq t))$$

If there is no real number $s \geq t$ then the supremum (or least upper bound) equals $+\infty$. ♦

Definition 4.5. Let (X,d) be a metric space and let $f : X \to X$ be a transformation. If $\mathrm{Lip}(f) = s$ then:

$$(\underline{\mathrm{A}}\, x,y : x,y \in X : d(f(x),f(y)) \leq s d(x,y))$$

If $s < \infty$ then f is ***Lipschitz***, if $s < 1$ then f is a ***contraction*** with ***contractivity factor*** s [HUT81]. ♦

Theorem 4.1. Let $f : X \to X$ be a contraction mapping on a complete metric space (X,d). Then f possesses exactly one fixed point $x_f \in X$ and the sequence $\{f^{(n)}(x) : n = 0,1,2,\ldots\}$ converges to x_f:

$$(\underline{\mathrm{A}}\, x : x \in X : \lim_{n\to\infty} f^{(n)}(x) = x_f)$$

Proof. See [BAR88a]. ■

Theorem 4.2. Let (X,d) be a complete metric space and let $f: X \to X$ be a contraction mapping with contractivity factor $s \in [0,1)$ and let the fixed point of f be $x_f \in X$. Then [BAR88a]:

$$(\underline{\mathbf{A}}\, x : x \in X : d(x,x_f) \leq (1-s)^{-1} d(x,f(x)))$$

Proof. The distance function $d(a,b)$ for fixed $a \in X$ is continuous in $b \in X$, hence:

$d(x,x_f)$
$= \langle$ theorem 4.1 $\rangle$
$d(x, \lim_{n\to\infty} f^{(n)}(x))$
$= \langle$ calculus $\rangle$
$\lim_{n\to\infty} d(x, f^{(n)}(x))$
$= \langle$ definition 4.2 $\rangle$
$\lim_{n\to\infty} \sum_{i=1}^{n} d(f^{(i-1)}(x), f^{(i)}(x))$
$\leq \langle$ definition 4.5 $\rangle$
$d(x,f(x)) \lim_{n\to\infty} \sum_{i=1}^{n} s^{(i-1)}$
$= \langle\ |s| < 1$, sum for infinite geometric series $\rangle$
$(1-s)^{-1} d(x,f(x))$

Definition 4.6. A ***dynamical system*** is a 2-tuple (X,f), where $f: X \to X$ is a transformation on a metric space (X,d). The ***orbit*** of a point $x \in X$ is the sequence $\{ f^{(n)}(x) : n = 0,1,2,\ldots,\infty \}$. ♦

Example 4.1. We calculate some points of the orbits of the following functions:

$f(x) = x_{n+1}$	startpoint x_0	orbit of x_0
$\sqrt{x_n}$	0.3	{0.3,0.547..,0.740..,0.860..,...}
$\sqrt{x_n}$	3	{3,1.732..,1.316..,1.147..,...}
x_n^2	0.3	{0.3,0.09,0.0081,0.00006561,...}
x_n^2	3	{3{9,81,6561,43046721,...}
x_n^2-1	-1	{-1,0,-1,0,-1,0,-1,0,...}
x_n^2+1	-2	{-2,5,26,677,458330,...}
$x_n - \frac{g(x)}{g^{[1]}(x)},\ g(x) = x^2-7$	3	{3,2.66666..,2.64583..,2.64575..,...}

We see that $x_f = 1$ and $x_f = 0$ are fixed points of both $\sqrt{x}$ and x^2. The lower formula constitutes the Newton-Raphson method, which is often used for the approximation of a zero of a given function (in this case $g(x)$). We will use the notation $g^{[k]}(x)$ for the k^{th} ***derivative*** of the function $g(x)$. ★

In dynamical systems theory one is interested in what happens when a typical orbit is followed. The orbit of a single point may be a geometrically complex set or, in other words, a fractal.

Definition 4.7. Let (X,f) be a dynamical system. A ***periodic point*** of f is a point $x \in X$ such that $f^{(n)}(x) = x$ for some $n \in \mathbf{N}$. If x is a periodic point of f then the ***period*** of x is the least $n \in \mathbf{N}$ such that $f^{(n)}(x) = x$ and the first n points of the orbit of x are all distinct. If $x \in X$ has period n, then the orbit of x, which is given by:

$$\{ x, f(x), f^{(2)}(x), f^{(3)}(x), \ldots, f^{(n-1)}(x) \}$$

is a ***periodic orbit*** and is called an ***n-cycle*** [GUL92]. ♦

In example 4.1 the function $\sqrt{x}$ possesses a trivial 1-cycle $\{1\}$ or fixed point 1 for $x \in (0,\infty)$. The function x^2 possesses three trivial 1-cycles $\{0\},\{1\},\{+\infty\}$ or fixed points $0,1,+\infty$ for respectively $x \in (-1,1)$, $x \in \{-1,1\}$ and $x \in \mathbf{R}\backslash[-1,1]$. The function $f(x) = x^2 - 1$ possesses a 2-cycle $\{-1,0\}$ for $x \in \mathbf{R}$ and the function $x^2 + 1$ possesses a fixed point $+\infty$ for $x \in \mathbf{R}$. The last function in example 4.1 possesses a fixed point 2.6457.. which is the zero of the function $x^2 - 7$.

Definition 4.8. Let (X,f) be a dynamical system and let $x_f \in X$ be a fixed point of f. Let B be a closed unit ball with center x_f and radius ϵ, thus $B(x_f,\epsilon) = \{ y \in X : d(x_f,y) \leq \epsilon \}$. The point x_f is called an ***attractive fixed point*** of f if there is a number $\epsilon > 0$ such that f is a contraction mapping on $B(x_f,\epsilon)$. The point is called a ***repulsive fixed point*** of f if there are numbers $\epsilon > 0$ and $C > 1$ such that:

$$(\underline{\mathbf{A}}\, y : y \in B(x_f,\epsilon) : d(f(x_f),f(y)) \geq C \cdot d(x_f,y))$$

If f is differentiable at x_f then the following holds [GUL92]:

(i) if $|f^{[1]}(x_f)| < 1$ then x_f is attracting (stability);

(ii) if $|f^{[1]}(x_f)| > 1$ then x_f is repelling (instability);

(iii) if $|f^{[1]}(x_f)| = 1$ then x_f can be attracting, repelling or neither.

4.2. Graphical analysis of iterates

If we are able to render a reasonable precise graph of a given function f, then we may be able to analyze graphically the orbits of various members of the domain of f. First we draw the graph of f, along with the line $f(x) = x$. To obtain the orbit of x_0, first locate x_0 on the x-axis, then go in vertical direction to the point $(x_0,f(x_0))$. From there proceed in horizontal direction towards the line $f(x) = x$, to arrive at the point $(f(x_0), f(x_0))$, which we will denote (x_1, x_1). From then on we apply recursively the same process, i.e. we obtain the orbit of x_0 as the sequence $\{ f^{(n)}(x_0) : n = 0,1,2,\ldots,\infty \}$.

Example 4.2. Consider the so called *(eco)logistic map*: $f(x) = \mu x(1-x)$, $x \in [0,1]$, $\mu \in (0,\infty)$. This parametrized quadratic function has often be used by biologists to study the variation of population sizes of biological organisms for which there is a constant supply of food and space. We can calculate the fixed point(s) of $f(x)$ as follows:

$$x_f = \mu x_f(1-x_f) \Leftrightarrow 1 = \mu - \mu x_f \Leftrightarrow x_f = 1-1/\mu$$

If $\mu \in (0,1]$ then we have $1-1/\mu \in (-\infty,0]$, so in this case there is only one fixed point in the interval $[0,1]$, namely $x_f = 0$. By contrast, when $\mu > 1$, there are two distinct fixed points in $[0,1]$, namely $x_f = 0$ and $x_f = 1-1/\mu$.

Next we will determine which fixed points are attracting and which are repelling. Since $f^{[1]}(x) = \mu - 2\mu x$, it follows that $f^{[1]}(0) = \mu$ and $f^{[1]}(1-1/\mu) = \mu - 2\mu(1-1/\mu) = 2-\mu$. Fixed point 0 is attracting if $\mu \in (0,1)$ and is repelling if $\mu \in (1,\infty)$. Fixed point $1-1/\mu$ is attracting if $\mu \in (1,3)$ and is repelling if $\mu \in (3,\infty)$. In fig. 4.1 this is shown graphically. The orbits of a point x_0 are shown for three μ-values: $\mu = 0.8 \in (0,1)$, $\mu = 2.9 \in (1,3)$ and $\mu = 4 \in (3,\infty)$.

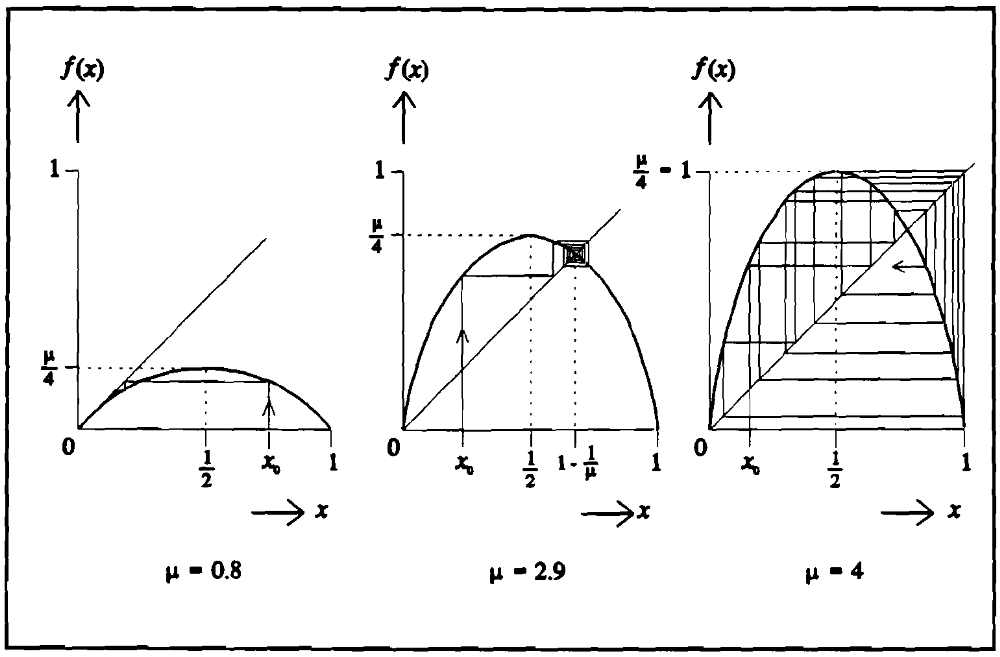


Figure 4.1. Graphical analysis of the logistic map for different μ-values.

As we can see, the left map possesses an attracting fixed point 0, the middle map possesses an attracting fixed point $1-1/\mu \approx 0.6551$ and a repelling fixed point 0 and the right map possesses two repelling fixed points 0 and $1-1/\mu = ¾$. ★

4.3. Attractors and bifurcations

We have seen that the amount and the behaviour of the fixed points can change, when the function parameter (μ in case of the logistic map) is changed. Before we are going to investigate this phenomenon, we will give some more definitions that are necessary to describe it.

Definition 4.9. Let (X,f) be a dynamical system and let $A_f \subset X$. We call A_f the ***attractor*** of f if A_f is a closed set that is invariant under f, i.e. $f(A_f) = A_f$, such that [FAL90,GUL92]:

$$\lim_{n\to\infty} h(f^{(n)}(A_f),A_f) = 0 \qquad \blacklozenge$$

For example the attractor of the function $f(x) = x^2-1$ is the set $\{-1,0\}$, because $f(-1) = 0$ and $f(0) = -1$ (see also example 4.1). So $f(\{-1,0\}) = \{-1,0\}$ and, in addition, $\{-1,0\}$ is a 2-cycle. Roughly speeking one can say that the attractor is a set to which all nearby orbits converge.

In example 4.1 we saw that for $\mu = 0.8$ and $\mu = 2.9$ the attractor of f equals $\{0\}$ and $\{0.6551..\}$ respectively. So it seems that the dynamics of f changes as the parameter changes. Up till now we have only discussed attractive and repelling fixed points, but not the periodic points that are neither attractive nor repelling.

Let us investigate the dynamics of the logistic map in a numerical way. We will try to find the periodic points of the logistic map for some increasing values of $\mu \in (0,4]$. In table 4.1 we iterate the logistic map 82 times for 7 values of $\mu \in (0,4]$. Each time we choose startvalue $x_0 = 0.883$. In fact, we may have choosen any startvalue $x_0 \in (0,1)$. This doesn't make difference for the outcome of the iteration, but may only influence the amount of necessary iterations to let the periodic points converge to the same amount of decimals.

Table 4.1. Orbits of $x_0 = 0.883$ as successive iterates of $x_{n+1} = \mu x_n(1-x_n)$ for $n = 0,1,..,82$ and different $\mu \in (0,4]$.

n	μ=0.5	μ=1.5	μ=2.5	μ=3.295	μ=3.514	μ=3.557	μ=3.568
0	0.8830..	0.8830..	0.8830..	0.8830..	0.8830..	0.8830..	0.8830..
1	0.0516..	0.1549..	0.2582..	0.3404..	0.3630..	0.3674..	0.3686..
2	0.0244..	0.1964..	0.4789..	0.7398..	0.8125..	0.8267..	0.8304..
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
66	0.0000..	0.3333..	0.6000..	0.8225..	0.8243..	0.8318..	0.8303..
67	0.0000..	0.3333..	0.6000..	0.4809..	0.5089..	0.4974..	0.5025..
68	0.0000..	0.3333..	0.6000..	0.8225..	0.8782..	0.8892..	0.8919..
69	0.0000..	0.3333..	0.6000..	0.4809..	0.3758..	0.3503..	0.3437..
70	0.0000..	0.3333..	0.6000..	0.8225..	0.8243..	0.8096..	0.8049..
71	0.0000..	0.3333..	0.6000..	0.4809..	0.5089..	0.5482..	0.5602..
72	0.0000..	0.3333..	0.6000..	0.8225..	0.8782..	0.8809..	0.8790..
73	0.0000..	0.3333..	0.6000..	0.4809..	0.3758..	0.3730..	0.3793..
74	0.0000..	0.3333..	0.6000..	0.8225..	0.8243..	0.8318..	0.8400..
75	0.0000..	0.3333..	0.6000..	0.4809..	0.5089..	0.4974..	0.4794..
76	0.0000..	0.3333..	0.6000..	0.8225..	0.8782..	0.8892..	0.8904..
77	0.0000..	0.3333..	0.6000..	0.4809..	0.3758..	0.3503..	0.3479..
78	0.0000..	0.3333..	0.6000..	0.8225..	0.8243..	0.8096..	0.8095..
79	0.0000..	0.3333..	0.6000..	0.4809..	0.5089..	0.5482..	0.5502..
80	0.0000..	0.3333..	0.6000..	0.8225..	0.8782..	0.8809..	0.8830..
81	0.0000..	0.3333..	0.6000..	0.4809..	0.3758..	0.3730..	0.3685..
82	0.0000..	0.3333..	0.6000..	0.8225..	0.8243..	0.8318..	0.8303..
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

From table 4.1 we see that the logistic map has the following attractors:

(i) for $\mu = 0.5$ an attractive fixed point $\{0\}$;

(ii) for $\mu = 1.5$ an attractive fixed point $1-1/(1.5) = \{0.3333..\}$;

(iii) for $\mu = 2.5$ an attractive fixed point $1-1/(2.5) = \{0.6000..\}$;

(iv) for $\mu = 3.295$ a 2-cycle $\{0.8225..,0.4809..\}$;

(v) for $\mu = 3.514$ a 4-cycle $\{0.8243..,0.5089..,0.8782..,0.3758..\}$;

(vi) for $\mu = 3.557$ an 8-cycle $\{0.8318..,0.4974..,0.8892..,0.3503..,0.8096..,0.5482.., 0.8809..,0.3730..\}$;

(vii) for $\mu = 3.568$ a 16-cycle $\{0.8303..,0.5025..,0.8919..,0.3437..,0.8049..,0.5602.., 0.8790..,0.3793..,0.8400..,0.4794..,0.8904..,0.3479..,0.8095..,0.5502..,0.8830.., 0.3685..\}$.

It is clear that when μ is increased, for certain values μ_k the period doubles from 2^k to 2^{k+1}. Also, when μ is increased, it takes longer to converge to the associated 2^k-cycle.

Definition 4.10. A one-parameter function family $f(x,\mu)$ has a ***bifurcation*** at μ_k, or bifurcates at μ_k, if the number or nature (attracting vs. repelling) of periodic points of $f(x,\mu)$ changes as μ passes through μ_k. In this case μ_k is said to be a ***bifurcation point*** for that family of functions [GUL92].♦

Feigenbaum found for the logistic map the following successive bifurcation points [FEI80]:

$$\begin{aligned}
\mu_0 &= 3.000000.. \\
\mu_1 &= 3.449499.. \\
\mu_2 &= 3.544090.. \\
\mu_3 &= 3.564407.. \\
&\cdot \\
&\cdot \\
\mu_\infty &= 3.569946..
\end{aligned}$$

The sequence has a limit $\mu_\infty = 3.569946..$ which is called the ***chaos criterium*** for the system [MOO87]. Feigenbaum also noticed the following convergence rate in the sequence μ_k [FEI80]:

$$\delta = \lim_{k\to\infty} \frac{\mu_k - \mu_{k-1}}{\mu_{k+1} - \mu_k} = 4.669201..$$

The constant δ is referred to as the ***Feigenbaum constant*** or the ***Feigenbaum ratio*** and has universal validity. This means that it also holds for unimodal functions (i.e. functions with a single maximum) as for instance $f(x) = \mu\sin\pi x$, $f(x) = xe^{\mu(1-x)}$, $f(x) = x(1+\mu(1-x))$, etc. The Feigenbaum ratio is as characteristic for the period doubling scenarios as the number π is for the relation of the circumference to the diameter of the circle [FAL90]. It must be noticed that the chaos criterium is system dependent (and therefore not universal), while the Feigenbaum constant is system independent (and therefore universal).

One method of displaying the points at which a parametrized family of functions bifurcates, is by means of a ***bifurcation diagram***. This is a graph for which the horizontal axis represents the parameter values μ and the vertical axis represents the elements of the attractor A_f, i.e. the periodic points. The bifurcation diagram for the μ-values of table 4.1 and the bifurcation points Feigenbaum found, is given in fig. 4.2.

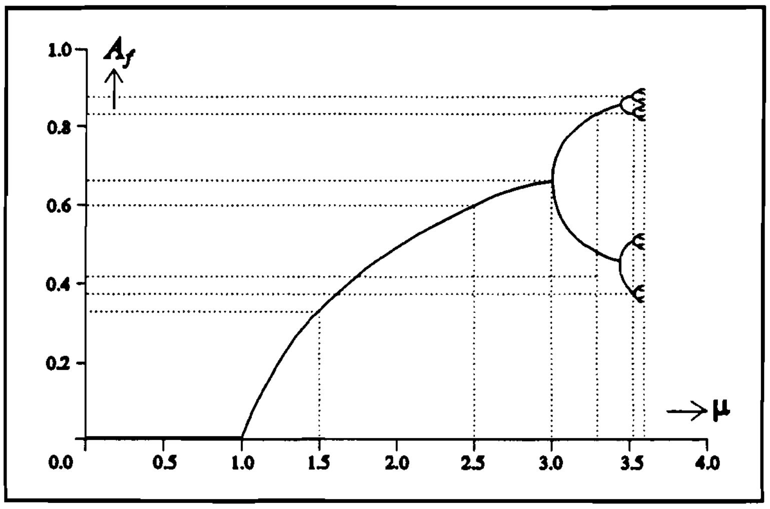


Figure 4.2. Construction of bifurcation diagram for the logistic map.

In case we are going to perform the iterations for small μ-increments, we will obtain the bifurcation diagram for $f(x)$ as shown in fig. 4.3. This diagram contains the whole attractor A_f of the logistic map for $\mu \in (0,4]$.

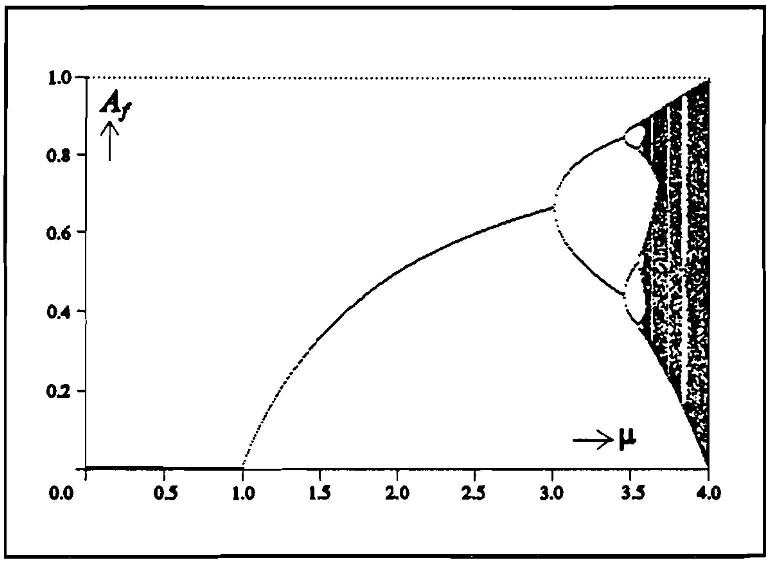


Figure 4.3. Bifurcation diagram of the logistic map.

Zooming in on the bifurcation diagram shows that it possesses a self similar fractal structure, which is scale-invariant. The attractor of a dynamical system is a "picture" of its behaviour. The periodic fixed points are the solutions of the differential equation(s) which describe the system in the usual way. From an attractor we can directly see for which μ-values these differential equations have a solution and for which they have not.

4.4. Rescaling function dynamics

In the former paragraph we have seen that the dynamics of the logistic map f changes as μ is varied. More specific, f undergoes period doubling, i.e. instead of having a stable k-cycle, the system gets a stable $2k$-cycle. Feigenbaum found that if f has a k-cycle $\{x_1,\ldots,x_k\}$ then $f^{(n)}$ (for $n \in [1,1+\log_2 k]$) possesses 2^{n-1} cycles of period $2^{1-n}k$ which form a partition on the k-cycle of f. Eventually $f^{(1+\log_2 k)}$ possesses k fixed points $x_1,\ldots,x_k$.

In general $f^{(2^{n+1})}$ can be recursively computed by $f^{(2^{n+1})} = f^{(2^n)} \circ f^{(2^n)}$. Suppose $f^{(2^n)}$ has a finite cycle that contains a fixed point $x_f = ½$ for $\mu = \mu_1$. Now we are going to increase μ to μ_2 so that the fixed point of the cycle of $f^{(2^{n+1})}$ nearest to $x = ½$ is again at $x_f = ½$ with slope 0 (see fig. 4.4).

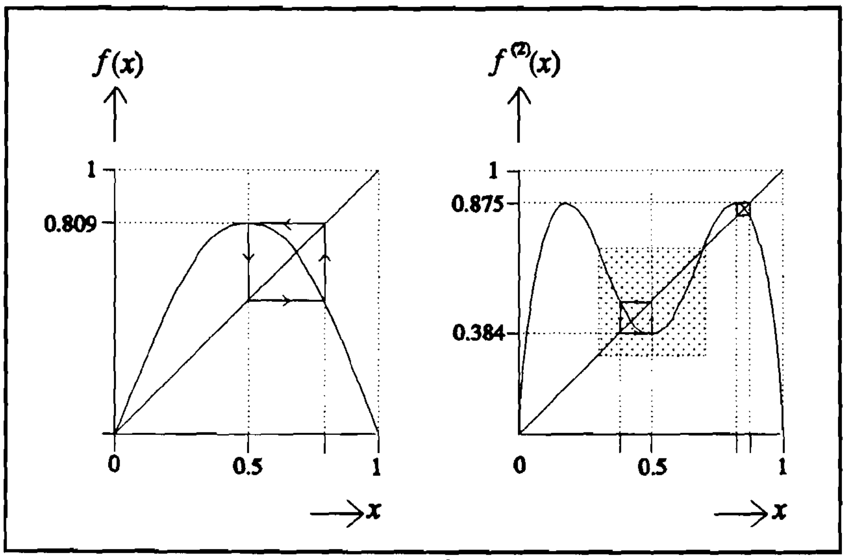


Figure 4.4. Left: f for $\mu_1 = 3.2360$, right: $f^{(2)}$ for $\mu_2 = 3.4986$.

Consider fig. 4.4. At the left we plotted $f = \mu x(1-x)$ for $\mu_1 = 3.2360$ such that f possesses a 2-cycle $\{0.5000..,0.8089..\}$, which contains $x_f = ½$ as one of its two fixed points. Now we increase μ until f possesses a 4-cycle $\{0.3835..,0.8272..,0.5000..,0.8746..\}$, which again contains $x_f = ½$ as one of its (in this case) four fixed points. This happens at $\mu = \mu_2 = 3.4986$. At the right we plotted $f^{(2)} = \mu^2 x(1-x)(1-\mu x(1-x))$ for $\mu_2 = 3.4986$. The maximum of f can be calculated as follows: $f(0.5) = 3.2360 \cdot 0.5(1-0.5) \approx 0.809$. The right maximum of $f^{(2)}$ can be calculated as follows: $f^{(2)}(0.8272) = (3.4986)^2 \cdot 0.8272(1-0.8272)(1-3.4986 \cdot 0.8272(1-0.8272)) \approx 0.875$. Also, the local minimum can be calculated as $f^{(2)}(0.5) = (3.4686)^2 \cdot 0.5(1-0.5)(1-3.4986 \cdot 0.5(1-0.5)) \approx 0.384$.

Next consider the contents of the dotted rectangle in the right graph of fig. 4.4 and compare this to the whole left picture of fig. 4.4. We see that if the contents of the dotted rectangle are reflected through $(x,f(x)) = (½,½)$ and then magnified with a scaling factor α, these contents equal the whole left picture. This rescaling is determined only by functional composition, so there is some function g that, composed with itself, will reproduce itself in scale by $-\alpha$ and reflected through $(x,f(x)) = (½,½)$. This function has a quadratic maximum at $x = ½$, is symmetric about $x = ½$ and can be scaled by hand to equal 1 as $x = ½$. Shifting coordinates so that $x = ½ \rightarrow x = 0$ we have:

$$-\alpha g(g(x/\alpha)) = g(x)$$

Substituting $g(0) = 1$ we have $g(1) = -1/\alpha$. There is a unique smooth solution to this equation which (obtained numerically) determines [PEI80a]:

$$\alpha = 2.502907..$$

Like the Feigenbaum ratio δ, this so called ***renormalization factor*** (or rescaling factor) α is a number that can be measured in any phenomenon exhibiting period doubling, and thus also has universal property. So we see that the relevant operation upon functions that underlies period doubling is functional composition followed by magnification [FEI79,FEI80]. In general, the following ***rescaling law*** holds [MOO87]:

$$\mu_k \approx \mu^\infty - \alpha\delta^{-k}$$

Thus, knowing that two successive bifurcation points can give only an estimate of the chaos criterium μ_∞, we obtain [MOO87]:

$$\mu_\infty \approx \frac{\delta\mu_{k+1} - \mu_k}{\delta - 1}$$

Example 4.3. Consider the logistic map. We first calculate the successive bifurcation points μ_0, μ_1, μ_2 and μ_3 from the known values of α, δ and μ_∞. Then we calculate the chaos criterium μ_∞ from the known values of δ, μ_2 and μ_3:

$$\mu_0 \approx (3.569946..) - (2.502907..)(4.669201..)^{-1} \approx 3.033899..$$

$$\mu_1 \approx (3.569946..) - (2.502907..)(4.669201..)^{-2} \approx 3.455141..$$

$$\mu_2 \approx (3.569946..) - (2.502907..)(4.669201..)^{-3} \approx 3.545358..$$

$$\mu_3 \approx (3.569946..) - (2.502907..)(4.669201..)^{-4} \approx 3.564680..$$

$$\vdots$$

$$\mu_\infty \approx ((4.669201..)(3.564407..) - (3.544090..))(4.669201.. - 1)^{-1} \approx 3.569944..$$

These values are close to the values Feigenbaum found (see below definition 4.10). ★

4.5. Poincaré maps

Up till now we have been dealing with discrete dynamical systems, which are in fact approximations of continuous dynamical systems. Thus, a continuous dynamical system is a generalization of a (discrete) dynamical system (X,f) as defined in definition 4.6. Instead of iteration equations of the form:

$$x_{n+1} = f^{(n)}(x) = f(x_n)$$

continuous dynamical systems are described by differential equations of the form:

$$x^{[1]}(t) = \frac{dx}{dt} = g(x(t))$$

As in the discrete case, continuous dynamical systems give rise to attractors and repellers. When $X = \mathbf{R}^2$, the range of attractors for continuous dynamical systems is rather limited. The only attractors possible are isolated points (x for which $g(x) = 0$) or closed loops. More complicated attractors can not occur [FAL90]. Consequently, to find continuous dynamical systems with fractal attractors (or strange attractors, see par. 5.2), we need to look at systems in three or more dimensions.

Linear differential equations (with $g(x)$ a linear function of x) can be solved completely by classical methods, the solutions involving periodic or exponential terms. However, even simple nonlinear systems can lead to orbits of a highly intricate form. Nonlinear differential equations, particularly in higher dimensions, are notoriously difficult to analyze. This must be done by a combination of qualitative mathematical analysis and numerical approximation.

A standard approach in chaos theory to look at a three-dimensional continuous dynamical system g is by means of a two-dimensional ***phase plane*** $\Phi(t) = x(t) \times x^{[1]}(t)$, which is actually a "cross section" of the three dimensional attractor. If the phase plane Φ is a plane region perpendicular to the orbit of the system, we may define the "first return map" $f\colon \Phi \to \Phi$ by taking $f(x)$ as the point at which the orbit through $x = (x(t_n), x^{[1]}(t_n))$ next intersects Φ (see fig. 4.5), so:

$$(x(t_{n+1}), x^{[1]}(t_{n+1})) = f(x(t_n), x^{[1]}(t_n))$$

Then f is a discrete dynamical system on Φ. If f has an attractor A_f in Φ it follows that the union of the trajectories through the points of A_f is an attractor A_g of g. Locally, A_g looks like a product of A_f and a line segment, and typically:

$$D_H(A_g) = 1 + D_H(A_f)$$

Definition 4.11. Let (X,g) with $X \subset \mathbf{R}^3$ be a continuous three-dimensional dynamical system with attactor A_g. Then the ***Poincaré map*** of g is defined as the two-dimensional (attractor A_f of the) discrete dynamical system $(x(t) \times x^{[1]}(t), f)$, where f and g are respectively defined by:

$$(x(t_{n+1}), g(x(t_{n+1}))) = f(x(t_n), g(x(t_n))) \quad \text{with} \quad g(x(t)) = x^{[1]}(t) \qquad \blacklozenge$$

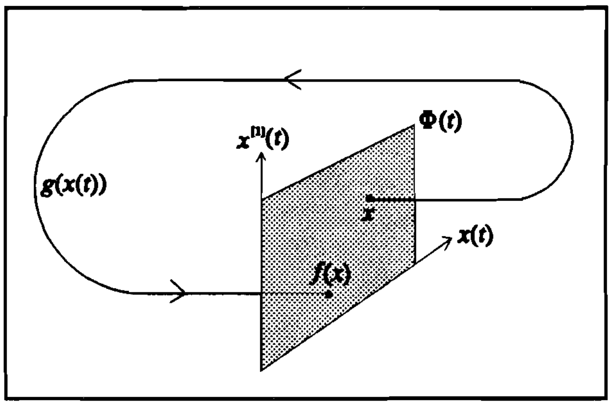
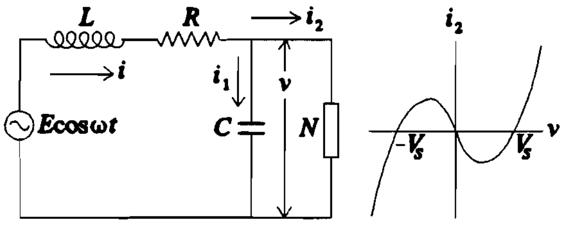


Figure 4.5. The continuous dynamical system (X,g) induces a discrete dynamical system (X,f) on a phase plane Φ.

__Example 4.4.__ Consider the forced nonlinear negative resistance oscillator with periodic excitation, for which the circuit schematic and the voltage-current characteristic of the negative resistance are given below [UED81]:



The differential equations of the system in terms of the variables as shown in the figure are given by:

$$\begin{cases} L\dfrac{di}{dt} + Ri + v = E\cos\omega t \\ i = i_1 + i_2 \\ i_1 = C\dfrac{dv}{dt} \\ i_2 = \dfrac{v}{R}(\dfrac{v^2}{V_s^2} - 1) \end{cases}$$

We introduce new quantities conform [UED81]:

$$\gamma = \frac{3L}{L - R^2C} \qquad \mu = \sqrt{\frac{3L}{R^2} - 3} \qquad \nu = \omega\sqrt{\gamma LC} \qquad \tau = \frac{t}{\sqrt{\gamma LC}} \qquad B = \frac{\gamma E\sqrt{\gamma}}{V_s} \qquad x = \frac{v\sqrt{\gamma}}{V_s}$$

Now we can rewrite the differential equations as a ***Van der Pol equation***:

$$\frac{d^2x}{d\tau^2} + \frac{dx}{d\tau}\mu(x^2-1) + x^3 = B\cos\nu\tau$$

which can be rewritten as:

$$x^{[2]} + x^{[1]}\mu(x^2 - 1) + x^3 = B\cos\nu\tau$$

By defining $z = \nu\tau + 2k\pi$ the system can be transformed to the three-dimensional state space $\mathbf{R}^2 \times \mathbf{R}$ (mod 2π), with coordinates x,y and z:

$$\begin{cases} x^{[1]} = y \\ y^{[1]} = y\mu(1 - x^2) - x^3 + B\cos z \\ z^{[1]} = \nu \end{cases}$$

This system can be thought of as a cylindrical state space (a torus), where the values of z are restricted to $z \in [0,2\pi]$. We can define the Poincaré map as:

$$(x_{n+1}, x^{[1]}_{n+1}) = f(x_n, y_n)$$

A picture of a part of the attractor of this system, together with Poincaré maps for some different z-values, is given in fig. 4.6. Poincaré maps for some (B,ν) pairs are given in fig. 4.7. ★

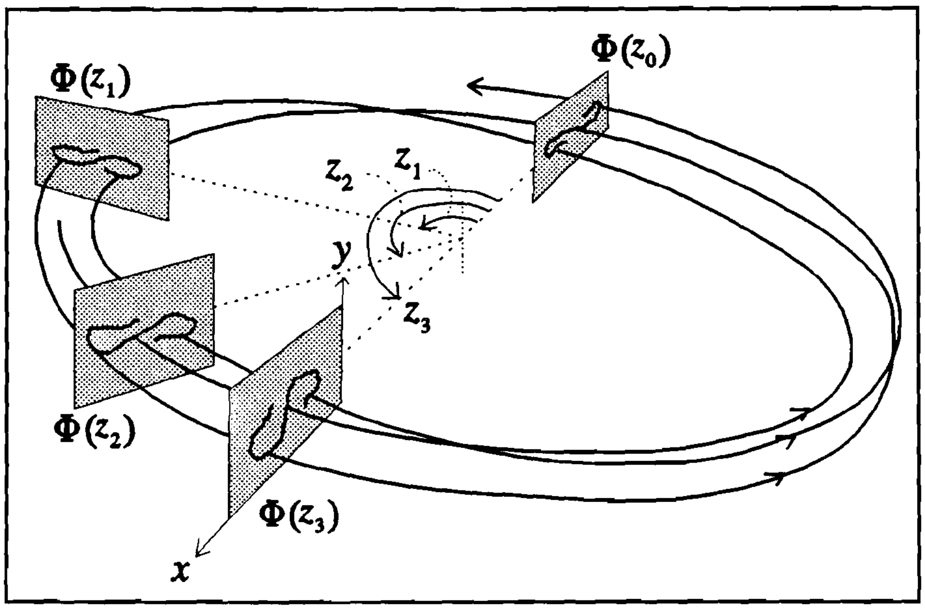


Figure 4.6. Sketch of the attractor of a forced non-linear oscillator with some Poincaré maps.

The more "beautiful" the attractors, the less energy the dynamical system dissipates. The time ordered sequence of phase plane filling points is nondeterministic, but the (fractal) structure which appears (the attractor) is deterministic.

More information about chaotic dynamics in electronic circuits can be found in [BUS85,UED81]. Especially [BUS85] also deals with the period doubling scenario, visualized by bifurcation diagrams. It appears to be the case that this type of electronic circuits has a period doubling scenario that obeys the route to chaos, which is governed by the Feigenbaum constant δ. This is also valid for coupled circuits.

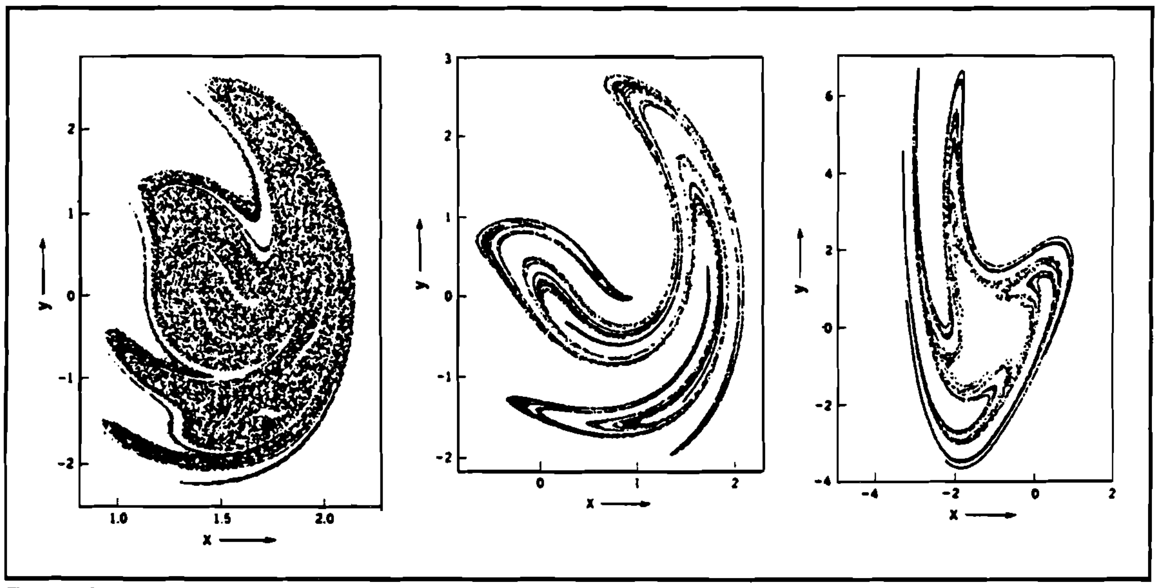


Figure 4.7. Poincaré maps for circuit of example 4.4 for (B,ν) is respectively (2.0,0.6), (2.4,0.7) and (17.0,4.0) [UED81].

Had Cleopatra's nose been shorter, the whole face of the world would have changed.

Blaise Pascal (1623 - 1662) [ADL77]

Chapter 5

Chaos

From fig. 4.3 we see that when μ passes through $\mu_\infty = 3.569946..$, the dynamical behaviour of the logistic map becomes chaotic, i.e. unpredictable. This means that there is no unique ordered finite n-cycle anymore. We also see that the chaotic region $(\mu_\infty, 4]$ contains "windows of order", or ***periodic windows***, where finite cycles occur. We can define this chaotic behaviour formally.

Definition 5.1. Let (X,f) be a dynamical system. Then $f : X \to X$ is ***chaotic*** if it satisfies at least one of the following conditions:

(i) f has a positive Lyapunov exponent at each point in its domain that is not eventually periodic [GUL92,MOO87];

(ii) f has a strange attractor [MOO87];

(iii) f has positive entropy [BRO90];

(iv) f has sensitive dependence on initial conditions [FAL90,GUL92];

(v) f has a 3-cycle [GUL92]. ♦

We will investigate all five criteria for chaos as mentioned in the definition above.

5.1. Lyapunov exponents

Definition 5.2. Let (X,f) be a dynamical system and let $X \subset \mathbb{R}$ be a bounded interval. Furtermore, let $f : X \to X$ be a one-dimensional differentiable function on X. Define the Lyapunov exponent of f at $x \in X$ as [BRO90,GUL92]:

$$\lambda(x) = \lim_{n\to\infty} \frac{1}{n} \ln|(f^{(n)})^{[1]}(x)| = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} \ln|f^{[1]}(x_k)|$$

If $\lambda(x)$ is dependent of x, then the common value of $\lambda(x)$ is denoted by λ and is called the ***Lyapunov exponent*** of f. ♦

The Lyapunov exponent of x measures whether and how strong x is repelling or attracting the orbit of f [BRO90].

Example 5.1. From example 4.2 we know that for $\mu \in (1,3)$ the logistic map possesses an attracting fixed point $x_f = 1 - 1/\mu$. This means that in case n increases without bound, all points $x \in (0,1)$ will eventually be equal to x_f. In other words:

$$(\underline{A}\, x : x \in (0,1) : \lim_{n\to\infty} f^{(n)}(x) = 1 - \frac{1}{\mu})$$

Since $f^{[1]}(1 - 1/\mu) = 2 - \mu$, we get:

$$(\underline{A}\, x : x \in (0,1) : \lim_{n\to\infty} (f^{(n)})^{[1]}(x) = 2-\mu)$$

and thus:

$$(\underline{A}\, x : x \in (0,1) : \lim_{n\to\infty} \frac{1}{n}\sum_{k=0}^{n-1} \ln|f^{[1]}(x_k)|) \Leftrightarrow (\underline{A}\, x : x \in (0,1) : \ln|2 - \mu|)$$

Since the final expression is independent of the number $x \in (0,1)$, we conclude that the Lyapunov exponent for $f(x) = \mu x(1 - x)$ and $\mu \in (1,3)$ equals $\lambda = \ln |2 - \mu|$. ★

If the Lyapunov exponents for all values $\mu \in (0,4]$ are calculated, we get the graph of fig. 5.1.

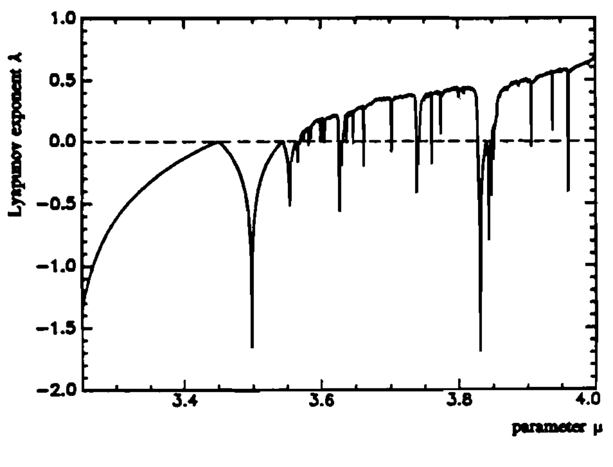


Figure 5.1. Lyapunov exponents for the logistic map [MOO87].

If we compare fig. 5.1 with fig. 4.3 we notice that beyond $\mu_\infty = 3.569946..$ the Lyapunov exponent becomes positive, except for the μ-values where the bifurcation diagram shows periodic windows. So the logistic map is chaotic for $\mu \geq \mu_\infty$.

Definition 5.3. Let (X,f) be a dynamical system and let $X \subset \mathbb{R}^2$ be a bounded region. Furthermore, let $f: X \to X$ be a continuously differentiable two-dimensional function on X. Also assume that f has an attractor A_f and that $B(x,\epsilon) = \{ y \in X : d(x,y) \leq \epsilon \}$ is a unit ball with center $x \in A_f$ and radius ϵ. The first derivative $(f^{(n)})^{[1]}(x)$ is a linear mapping which maps the unit ball in n steps into an ellipse with major and minor radii of lengts $\epsilon a_n(x)$ and $\epsilon b_n(x)$ respectively (see fig. 5.2). We define the Lyapunov exponents of f at $x \in X$ as the average logarithmic rate of growth of these radii lengths as n increases, so [FAL90,GUL92]:

$$\lambda_1(x) = \lim_{n\to\infty} \frac{1}{n}\ln a_n(x) \quad \text{and} \quad \lambda_2(x) = \lim_{n\to\infty} \frac{1}{n}\ln b_n(x)$$

If $\lambda_1(x)$ and $\lambda_2(x)$ are independent of x, then the common values of $\lambda_1(x)$ and $\lambda_2(x)$ are denoted by λ_1 and λ_2 respectively and are called the ***Lyapunov exponents*** of f. ♦

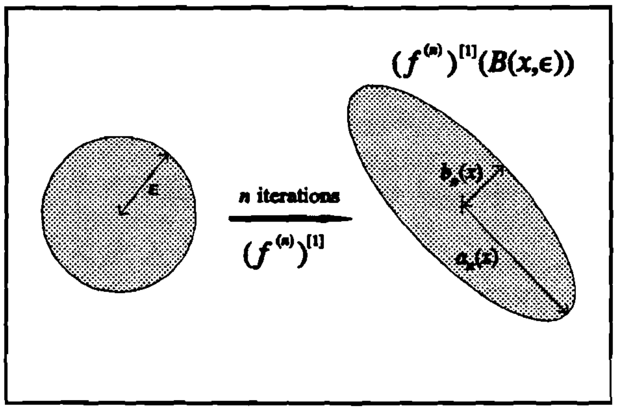


Figure 5.2. The unit ball is deformed into an ellipse by iteration under f.

As $\epsilon \to 0$ then $(f^{(n)})^{[1]}(B(x,\epsilon))$ will typically be close to an ellipse with radii of lengths $\epsilon e^{n\lambda_1}$ and $\epsilon e^{n\lambda_2}$ respectively [FAL90].

Definition 5.4. Let (X,f) be a dynamical system with attractor A_f and Lyapunov exponents λ_1 and λ_2. Assume that $\lambda_1 > 0 > \lambda_2$. We define the ***Lyapunov dimension*** of the attractor A_f as [GUL92,MOO87]:

$$D_L(A_f) = 1 - \frac{\lambda_1}{\lambda_2}$$ ♦

Example 5.2. Consider the so called ***bakers' map***, in a slightly different form also known as the ***horseshoe map***, $f: [0,1]^2 \to [0,1]^2$ defined by:

$$f(x,y) = \begin{cases} (2x, \mu(y-1)+1) & x \in [0,\tfrac{1}{2}],\ \mu \in (0,\tfrac{1}{2}) \\ (2x-1, \mu y) & x \in (\tfrac{1}{2},1],\ \mu \in (0,\tfrac{1}{2}) \end{cases}$$

This transformation may be thought of as stretching the unit square $[0,1] \times [0,1]$ into a $2 \times \mu$ rectangle, cutting it into two $1 \times \mu$ rectangles and placing these above each other with a gap of $1-2\mu$ in between (see fig. 5.3). Then the attractor $A_f = f^{(n)}([0,1] \times [0,1])$ of the transformation is an increasing sequence of sets with $[0,1] \times [0,1]$ comprising into 2^n horizontal strips of height μ^n separated by gaps of at least $(1-2\mu)^n$.

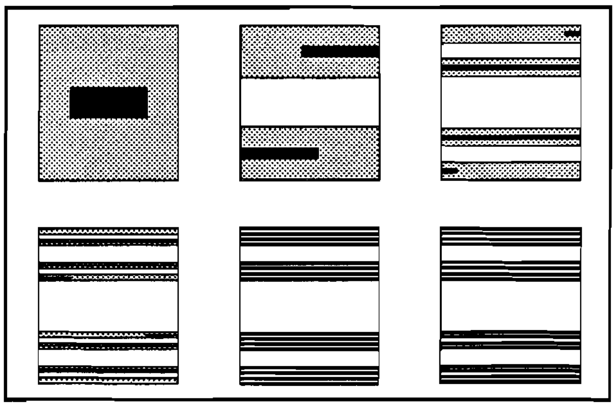


Figure 5.3. From upper left to lower right: six successive iterations of the baker's map for $\mu = \frac{1}{3}$.

The first derivative of $f(x,y)$, provided that $x \neq \frac{1}{2}$, is given by:

$$f^{[1]} = \begin{bmatrix} 2 & 0 \\ 0 & \mu \end{bmatrix} \Rightarrow (f^{(n)})^{[1]}(x,y) = \begin{bmatrix} 2^n & 0 \\ 0 & \mu^n \end{bmatrix}$$

Hence $a_n(x,y) = 2^n$ and $b_n(x,y) = \mu^n$. The Lyapunov exponents of f are given by:

$$\lambda_1(x,y) = \lim_{n\to\infty} \frac{1}{n}\ln a_n(x,y) = \lim_{n\to\infty} \frac{1}{n}\ln 2^n = \ln 2$$

$$\lambda_2(x,y) = \lim_{n\to\infty} \frac{1}{n}\ln b_n(x,y) = \lim_{n\to\infty} \frac{1}{n}\ln \mu^n = \ln \mu$$

Because $\mu \in (0,\frac{1}{2})$, the inequality $\lambda_1 > 0 > \lambda_2$ holds, and the Lyapunov dimension of the attractor A_f of the bakers' map can be calculated as follows:

$$D_L(A_f) = 1 - \frac{\lambda_1}{\lambda_2} = 1 - \frac{\ln 2}{\ln \mu} \in (1,2)$$

★

The Lyapunov dimension reflects the frequency with which points in the attractor are visited by orbits of various starting points [GUL92].

5.2. Strange attractors

<u>*Definition 5.5.*</u> Let (X,f) be a dynamical system with attractor A_f. We say that A_f is a ***strange attractor*** if A_f has a non-integer Lyapunov dimension and we say that A_f is a ***chaotic attractor*** if A_f has a Lyapunov dimension which exceeds 1 [GUL92]. ♦

In example 5.2 we found for the bakers' map that $D_L(A_f) \in (1,2)$, so the bakers' map has a strange attractor (which is also chaotic) and thus the bakers' map is chaotic.

5.3. Entropy

In chapter 3 We have seen that the capacity dimension D_C is a measure for the space-filling properties of an attractor. In the same way the Lyapunov dimension D_L is a measure for the frequency with which points in the attractor are visited by orbits of various starting points. We will introduce another dimension that is similar to the capacity dimension, but also tries to account for the frequency with which the trajectory visits each point of the attractor.

To calculate this entropy dimension, one counts the number of points in each of N just-touching boxes which cover the attractor and the probability of finding a point in each particular box.

Definition 5.6. Let (X,f) be a dynamical system with attractor A_f. We denote by $N(A_f,\epsilon)$ the smallest number of two-dimensional just-touching boxes of sidelength ϵ, required to completely cover A_f. We number each box with a unique code c_i and denote the ***alphabet*** of f by the sequence $C = \{c_0,\ldots,c_{N-1}\}$ [MOO87]. We represent the orbit of f by a sequence $\{ c_i : i \in \{0,\ldots,N-1\} \}$ where N denotes the amount of different points in the orbit, i.e. the total amount of code symbols. We define the ***self-information*** or ***surprise value*** of symbol c_i as [GAL68]:

$$I(c_i) = -\log_2 p_i \ [bits] \quad \text{with} \quad p_i = \frac{\#c_i}{N} \quad \text{and} \quad \sum_{i=0}^{N-1} p_i = 1$$

So p_i denotes the probability that the orbit of f visits the box with code symbol c_i. ♦

We are also free to use the natural logarithm (ln) in case of the 2-based logarithm ($\log_2$), but in that case the unity of $I(c_i)$ becomes natural digits ([*nats*]) in case of binary digits ([*bits*]).

The self-information is a measure for the predictability of the distribution of the points $x \in A_f$ over A_f. The more predictable the system, the higher the self-information. If A_f consists of a single point then the self-information becomes $-\log_2 1 = 0$ [*bits*]. So the behaviour is fully predictable or, in other words, we are not at all surprised by this outcome.

Definition 5.7. Let (X,f) be a dynamical system and let its attractor A_f be encoded in a similar way as described in definition 5.6. The average value of the self-information of all code symbols c_i is called the ***entropy*** of f and is denoted by [BRO90,FAL90,FED90,MOO87]:

$$H(f) = -\sum_{i=0}^{N-1} p_i \log_2 p_i \quad [bits/(code\ symbol)]$$

♦

The entropy is a measure for the unpredictability of a system. That is, for a uniform probability in each box, p_i equals $1/N$ and thus $H(f)$ reaches its maximum:

$$H(f) \leq -\sum_{i=0}^{N-1} N \log_2 \frac{1}{N} = -N\frac{1}{N}\log_2 N = \log_2 N \quad [bits/(code\ symbol)]$$

If all points are located in one box then $H(f) = 0$. One can also look upon the entropy as the amount of freedom the system has. The more freedom, the less predictable (deterministic) the system behaves.

Our alphabet consists of N code symbols, so the theoretical amount of possible different orbits of length k is equal to N^k. But the system doesn't necessarily have the capability to generate all this orbits. The smaller the entropy of a system, the less different orbits it can generate. A fully predictable system will generate only one possible orbit of length N. Let $H_k(\epsilon)$ denote the amount of orbits of length k that the system is able to generate, then clearly $N \leq H_k(\epsilon) \leq N^k$. We will now determine how $H_k(\epsilon)$ is related to k.

Definition 5.8. Let (X,f) be a dynamical system and let its attractor A_f be encoded in a similar way as described in definition 5.6. The ***ε-entropy*** of f is defined as [BRO90]:

$$H(f,\epsilon) = \lim_{n\to\infty}\frac{\log_2 H_k(\epsilon)}{k} \quad [bits]$$

♦

Thus, only if $H_k(\epsilon) = N^k$ then $H(f,\epsilon) = H(f) = \log_2 N$. Without proof we state that the following equation holds (see [BRO90,MOO87]):

$$H_{k+1}(\epsilon) \approx H_k(\epsilon)2^{H(f,\epsilon)}$$

or, in case the natural logarithm is used in definition 5.8 for the ε-entropy:

$$H_{k+1}(\epsilon) \approx H_k(\epsilon)e^{H(f,\epsilon)}$$

Definition 5.9. Let (X,f) be a dynamical system and let its attractor A_f be encoded in a similar way as described in definition 5.6. The ***entropy dimension*** (also called the information dimension) of f is defined as [FED89,MOO87]:

$$D_E(f) = \lim_{\epsilon\to 0}\frac{H(f)}{\log_2(1/\epsilon)} = \lim_{\epsilon\to 0}\frac{\sum_{i=0}^{N-1} p_i\log_2 p_i}{\log_2\epsilon}$$

♦

Example 5.3. Consider the bakers' map (see example 5.2). The attractor $A_f = f^{(\infty)}([0,1] \times [0,1])$ of this map consists of 2^n horizontal strips of length 1 and height μ^n. So the amount of possible orbits of length k is $H_k(\epsilon) = 2^n$ and the amount of possible orbits of length $k+1$ is $H_{k+1}(\epsilon) = 2^{n+1} = 2H_k(\epsilon)$ for $k \geq 2$. Since we know that $H_1(\epsilon) = 2^n$, it follows that $H_k(\epsilon) = 2^{n+k-1}$. So the ε-entropy yields:

$$H(f,\epsilon) = \lim_{k\to\infty}\frac{\log_2 H_k(\epsilon)}{k} = \lim_{k\to\infty}\frac{\log_2 2^{n+k-1}}{k} = \lim_{k\to\infty}\frac{(n+k-1)\log_2 2}{k}$$

$$= (\lim_{k\to\infty}\frac{n-1}{k} + 1)\log_2 2 = \log_2 2 = 1 \quad [bit]$$

Thus, each time the amount of possible orbits (states) doubles, we loose 1 [*bit*] of information about the predictability of the system behaviour. If we cover all 2^n horizontal strips of the bakers' map with boxes of sidelength $\epsilon = \mu^n$, then we need $N = 2^n\mu^n$ boxes to completely cover the attractor A_f. In this case $p_i = 1/N = 2^{-n}\mu^n$, so we can calculate the entropy of f as:

$$H(f) = -\sum_{i=0}^{N-1} p_i \log_2 p_i = -\sum_{i=0}^{2^n\mu^{-n}-1} 2^{-n}\mu^n \log_2(2^{-n}\mu^n) = -2^n\mu^{-n}2^{-n}\mu^n\log_2(\tfrac{\mu}{2})^n = n\log_2(\tfrac{2}{\mu})$$

Because $H(f) > 0$ for $\mu \in (0,½)$, the bakers' map has positive entropy and thus is chaotic. We can also calculate the information dimension of f as:

$$D_E(f) = \lim_{\epsilon\to 0}\frac{H(f)}{\log_2(1/\epsilon)} = \lim_{n\to\infty}\frac{n\log_2(2/\mu)}{\log_2\mu^{-n}} = \lim_{n\to\infty}\frac{-n\log_2(\mu/2)}{-n\log_2\mu} = \frac{\log_2\mu - \log_2 2}{\log_2\mu} = 1 - \frac{\log_2 2}{\log_2\mu}$$

In case $\mu = ⅓$ this yields $D_E(f) \approx 1.63$, and we see that $D_E(f) = D_C([0,1] \times \text{Cantor set}) = D_C([0,1]) + D_C(\text{Cantor set}) \approx 1 + 0.63$. ★

5.4. Sensitive dependence on initial conditions

We have seen that the logistic map becomes chaotic for $\mu > \mu_\infty$ which means that (except for the periodic windows) there are no unique ordered finite n-cycles anymore. But there is more to say.

<u>*Example 5.4.*</u> Consider the logistic map (see example 4.2). We will investigate the orbits for $\mu = 3.557$ and $\mu = 4$, each for two different starting values.

Table 5.1. Orbits of the logistic map for $\mu = 3.557$ and $\mu = 4$, for different starting values.

n	μ=3.557	μ=3.557	μ=4	μ=4
0	0.7300..	0.8830..	0.88300000000..	0.88300000001..
1	0.7010..	0.3674..	0.41324400000..	0.41324399997..
2	0.7454..	0.8267..	0.96989358586..	0.96989358584..
.	.	.	.	.
.	.	.	.	.
32	0.8809..	0.8809..	0.01195142608..	0.00186792168..
33	0.3731..	0.3730..	0.04723435801..	0.00745773022..
34	0.8319..	0.8318..	0.18001309374..	0.02960844992..
.	.	.	.	.
.	.	.	.	.
42	0.8318..	0.8318..	0.63954498650..	0.08079533875..
43	0.4974..	0.4974..	0.92210878698..	0.29706980796..
44	0.8892..	0.8892..	0.28729668783..	0.83527734864..
45	0.3503..	0.3503..	0.81902920397..	0.55035639796..
46	0.8096..	0.8096..	0.59288146805..	0.98985693274..
47	0.5482..	0.5482..	0.96549213157..	0.04016074180..
48	0.8809..	0.8809..	0.13326830179..	0.15419142648..
49	0.3730..	0.3730..	0.46203144610..	0.52166572192..
.	.	.	.	.
.	.	.	.	.

We see that for $\mu = 3.557$ and totally different startvalues 0.8830 and 0.7300, the logistic map settles down to a stable 8-cycle {0.8318..,0.4974..,0.8892..,0.3503..,0.8096..,0.5482..,0.8809..,0.3730..} as we also noticed from table 4.1. But for $\mu = 4$ and almost identical startvalues 0.88300000000 and 0.88300000001 the logistic map doesn't converge at all. Besides chaotic behaviour, we see that the small difference of 10^{-10} in startvalues causes the system to behave totally different after relatively few iterations. ★

The first one who noticed this kind of behaviour was Lorenz when he simulated a simple weather model, consisting of only three differential equations [LOR63]. This effect was called the ***butterfly effect***, referring to the small scale wing movement of a butterfly in Brazil that is iterated by a dynamical weather system and could thus eventually cause a large scale tornado in Texas. Thus we see a propagation of the same behaviour through all scales. The butterfly effect acquired a technical name: sensitive dependence on initial conditions.

Definition 5.10. Let (X,f) be a dynamical system. Then $f : X \to X$ has ***sensitive dependence on initial conditions***, or just sensitive dependence, if [GUL92]:

$$(\underline{A}\, \epsilon : \epsilon > 0 : (\underline{E}\, x,y,n : x,y \in X,\, n \in \mathbb{N} : |x-y| < \epsilon \wedge |f^{(n)}(x) - f^{(n)}(y)| > \epsilon)) \quad ♦$$

Suppose that an iteration map $x_{n+1} = f(x_n)$ has an initial error Δx_0. So we get:

$$x_{n+1} + \Delta x_{n+1} = f(x_n + \Delta x_n)$$

The right part of this equality can be approximated with a Taylor sequence:

$$f(x_n) + \Delta x_n f^{[1]}(x_n) + \frac{\Delta x_n^2}{2!} f^{[2]}(x_n) + \frac{\Delta x_n^3}{3!} f^{[3]}(x_n) + \ldots + \frac{\Delta x_n^n}{n!} f^{[n]}(x_n) + \frac{\Delta x_n^{n+1}}{(n+1)!} f^{[n+1]}(\xi)$$

With $x_{n+1} = f(x_n)$ we get:

$$\Delta x_{n+1} \approx \Delta x_n f^{[1]}(x_n)$$

So an initial error Δx_0 causes for x_n an error:

$$\Delta x_n = \Delta x_0 \prod_{i \to 1}^{n} f^{[1]}(x_{i-1})$$

We can rewrite this equation in the form:

$$\Delta x_n = \Delta x_0 e^{n\lambda} \qquad \text{with} \qquad \lambda = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \ln |f^{[1]}(x_{i-1})|$$

The Lyapunov exponent λ is a measure for the average increase of the error per iteration. If $\Delta x_n = 1$ then the calculated value x_n has become totally unreliable. This moment will occur from iteration:

$$n = \frac{1}{\lambda} \ln \Delta x_0$$

The logistic map for $\mu = 4$ is chaotic, since it has sensitive dependence on initial conditions:

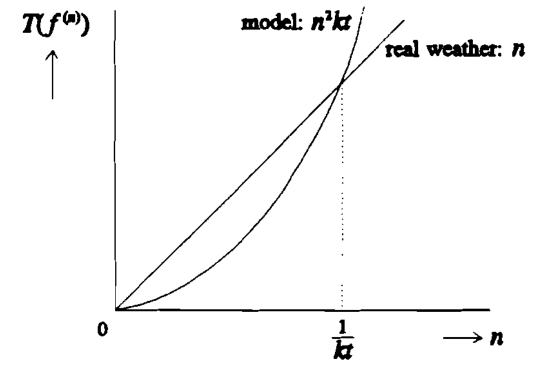
$$|x - y| < 10^{-10} \wedge |f^{(32)}(x) - f^{(32)}(y)| > 10^{-10}$$

Example 5.5. Consider the logistic map for $\mu = 4$. From example 5.1 we know that $\lambda = \ln|2-\mu| = \ln 2$. From table 5.1 we see that for $\mu = 4$ from iteration $n = 34$ and onwards the values of the orbits for $x_0 = 0.88300000000..$ and $x_0 = 0.88300000001..$ differ in all decimals. So at iteration $n = 34$, all significant decimals have been lost. This means that the computing device with which those iterations were performed, stores floating point numbers with an accuracy of:

$$\Delta x_0 = e^{-n\lambda} = e^{-34\ln 2} = 2^{-34} \approx 10^{-10} \quad [decimals]$$ ★

So we loose 1 [*bit*] of accuracy per iteration. This means that for the accurate prediction of the value of x_n one needs to have x_0 with an accuracy of at least n [*bits*]. So the calculation time per iteration is $O(n)$ and thus the prediction of x_n is $O(n^2)$. The far future (large n) is unpredictable; there exists a ***prediction horizon***. By the time we have calculated x_n, the system is already beyond this point.

Example 5.6. Assume that we want to make a weather forecast. We have available a computerized weather model and a sensor which measures the barometric pressure. Our computer has an execution time of t [*µs/instruction*] and the sensor is able to give readings of an infinite accuracy. For each iteration of the weather model, k [*instructions*] must be calculated, so this takes a calculation time of kt [*µs/iteration*]. For the prediction of x_n (the weather behaviour after n iterations), we need to initialize the weather model with a sensor reading x_0 of n [*decimals*]. The computational time complexity is $O(n^2)$, where n is the amount of decimals, i.e. the input length. So, to calculate x_n we need n^2kt [*µs*] of computation time. Meanwhile, the real weather is "iterating" with a constant rate and performs with a linear time complexity $O(n)$. We can draw the graphs of the time complexity function $T(f^{(n)})$ of the real weather system and of our model (see figure below).



There is a unique solution for the intersection point of the two graphs: $n = n^2kt \Rightarrow n = (kt)^{-1}$. This is the prediction horizon of our weather forecast, because at the time our model is calculating this value, we can no longer speak of a prediction, since the real weather system has also arrived at this point. Since there will never be computers with zero execution time per instruction and each computer program contains at least one instruction, there will always be a finite prediction horizon, which prevents the future beyond to be predictable. ★

Example 5.7. Lorenz estimated the prediction horizon of the real weather system to be 2 [*weeks*], but at present the weather forecasters estimate it to be at most 7 [*days*]. Several European countries participate in the European Centre for Medium-range Weather Forecasts (ECMWF), which is located in Reading (UK). The computers of the ECMWF perform calculations of the weather model on different grids which lay on concentric spheres. These spheres are aproximately 1 [*km*] apart and the points within a grid are approximately 100 [*km*] apart. The model is periodically initialized by sensordata like pressure, velocity, temperature and humidity of the air, which are measured at the same time all over the world at the gridpoints. This is done by means of radio probes, weather balloons and satellites. After the model is initialized, the computers start calculating the "orbit" of the weather with discrete steps of 10 [*minutes*].

Experience learns that after 5 to 6 [*days*] the errors have grown so big that the forecast has become useless and the model has to be re-initialized with up-do-date sensordata. There are meteorologists who think that the situation can be improved by increasing the amount of grid points, which implies that the computer power and the amount of sensors have to be increased correspondingly. Let us see if they are right.

It is true that the weather model is blind as to the space between the grid points. The smallest weather structure the model takes into account is a depression, which has a size of about 1000 [*km*] and exists for about 5 [*days*]. This is the reason we can only make reliable predictions up to 5 [*days*]. The formation of a depression is usually caused by a shower complex on a smaller scale, which is amplified under iteration of the weather system. A shower complex has a typical size of at most 100 [*km*] and a duration of about 1 [*day*]. If we want to take into account such shower complexes, we must be able to measure at a grid of 10 $\times$ 10 [*km*]. This requires both the computer power and the amount of sensor stations to be increased with a factor 100. This will cost an enormous amount of money and the gain in prediction time will be equal to the duration of existence of a shower complex, namely 1 [*day*]. So, we now can make reliable predictions up to 6 [*days*]. The formation of a shower complex is caused by coherent structures on an even smaller scale of 10 [*km*]. If we increase the computer power and amount of grid points again with a factor 100, we will be able to predict those coherent structures. This time the gain in prediction time will be equal to the duration of existence of a coherent structure: 5 [*hours*]. So, we now can make reliable predictions up to 6.2 [*days*]. We can continue this process until we reach the very small scale on which a butterfly lives. Its wing flaps cause little turbulences of about 10 [*cm*] with a duration of about 1 [*sec*].

It is obvious that the result of this adventure will be that we have spent an astronomical amount of money and we still are not able to make reliable weather predictions over more than 7 [*days*]. ★

5.5. Period 3 implies chaos

In fig. 4.3 we see that the logistic map becomes chaotic for $\mu > \mu_\infty$, but nevertheless, there are periodic windows for some $\mu \in (\mu_\infty, 4]$.

Theorem 5.1. ***(Li-Yorke theorem).*** Let (X,f) be a dynamical system and suppose that f is continuous on the closed interval X with $f(X) \subseteq X$. If f has a 3-cycle then f has n-cycles for all $n \in \mathbb{N}^+$.

Proof. We will not give the detailed proof which can be found in [LIY75]. The proof is based on the following idea. Assume that $a<b<c$ and that $f(a) = b$, $f(b) = c$ and $f(c) = a$. It can be proven that f must have a 1-cycle and also a 2-cycle. Let $n>3$ then f has all n-cycles. The kernel of this proof is to show that there is a point $p \in [b,c]$ such that:

(i) $f^{(k)}(p) \in [b,c]$ for $k = 1,2,3,\ldots,n-2$;

(ii) $f^{(n-1)}(p) \in (a,b)$;

(iii) $f^{(n)} = p$ and $p \in [b,c]$;

then automatically p will have period n. ■

The Li-Yorke theorem says that if f has a period-3 point, then it has points of all other periods. But suppose we can only show that f has for example a period-5 point. Then must f still have points of all periods ?

Definition 5.11. The ***Sharkovsky ordering*** of the positive integers is defined as [GUL92,PEI92]:

$$\underbrace{3 \rhd 5 \rhd 7 \rhd \ldots\ldots}_{\text{odd integers}} \rhd \underbrace{2\cdot3 \rhd 2\cdot5 \rhd 2\cdot7 \rhd \ldots\ldots}_{2\cdot(\text{odd integers})} \rhd \underbrace{2^2\cdot3 \rhd 2^2\cdot5 \rhd 2^2\cdot7 \rhd \ldots\ldots}_{2^2\cdot(\text{odd integers})} \rhd \underbrace{2^3 \rhd 2^2 \rhd 2 \rhd 1}_{\text{powers of } 2}$$

So $m \rhd n$ indicates that m appears before n in the Sharkovsky ordering. Thus $17 \rhd 14$ (because $14 = 2\cdot7$) and $40 \rhd 64$ (because $40 = 2^3\cdot5$ and $64 = 2^6$). Since every positive integer can be written as $2^k\cdot$(odd integer) for a suitable nonnegative integer k and a suitable odd integer, the Sharkovsky ordering is a permutation on $\mathbf{N}^+$. ♦

Theorem 5.2. (***Sharkovsky theorem***). Let (X,f) be a dynamical system and let f be continuous on the closed interval X with $f(X) \subseteq X$. If f has an m-cycle, then f has n-cycles for all n such that $m \rhd n$.

Proof. See [BRO90]. ■

By letting $m = 3$, we see that the Li-Yorke theorem is a direct implication of the Sharkovsky theorem. Moreover, from the Sharkovsky ordering we can imagine why the period-5 window in fig. 4.3 is located to the left of the large period-3 window.

The logistic map for $\mu = 3.829$ possesses an attracting 3-cycle $\{0.1576..,0.5085..,0.9569..\}$, because $f(0.1576..) = 0.5085..$ and $f(0.5085..) = 0.9569..$ and $f(0.9569..) = 0.1576..$, so the logistic map is chaotic for $\mu \in (\mu_\infty,4]$ (see fig. 5.4).

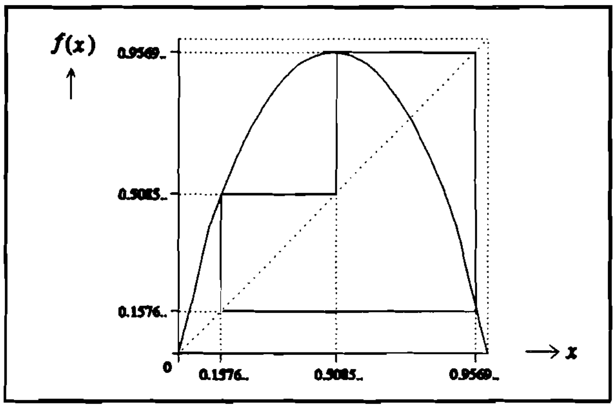


Figure 5.4. The logistic map possesses an attracting 3-cycle for $\mu = 3.829$.

Whence is it that Nature doth nothing in vain and whence arises all that order and beauty which we see in the world ?

Sir Isaac Newton (1642 - 1727) [ADL77]

Chapter 6

Attractor basin boundaries

Many of the beautiful exotic sets that have appeared on calendars and covers of magazines are sets which can be described in terms of iterates of complex functions. In this chapter we will describe the most famous of such sets, because we feel that we would be incomplete if we didn't.

Definition 6.1. Let (X,f) be a dynamical system with attractor A_f. The ***basin of attraction*** B_f of the attractor A_f consists of all $x \in X$ such that [GUL92]:

$$\lim_{n\to\infty} f^{(n)}(x) \in A_f \qquad \blacklozenge$$

Definition 6.2. Let (X,f) be a dynamical system with attractor A_f. The ***basin boundary*** of f is the set that lies between the basin of attraction of A_f and infinity [GUL92]. ♦

Example 6.1. Consider the dynamical system $(\mathbb{R}, f(x) = x^2)$. This system has three fixed points 0,1 and ∞. We know that:

$$\lim_{n\to\infty} f(x) = \begin{cases} 0 & \text{for } x \in (-1,1) \\ 1 & \text{for } x \in \{-1,1\} \\ \infty & \text{for } x \in \mathbb{R}\backslash[-1,1] \end{cases}$$

The attractor of f is given by $A_f = \{0,\infty\}$. The basin of attraction of 0 is the interval (-1,1) and the basin of attraction of ∞ is given by $\mathbb{R}\backslash[-1,1]$. The basin boundary of f is given by $\{-1,1\}$. ★

6.1. Julia Sets

Definition 6.3. Let $(\mathbb{C},f)$ be a dynamical system. The basin boundary of f is called the ***Julia set*** and is denoted by $J(f)$. The Julia set can also be defined as the smallest closed set that contains all repelling periodic points of f [DEV90,PEI88]. ♦

For example $J(z^2)$ with $z \in \mathbb{C}$ is a circle of radius 1 centered around the origin of the complex plane.

Example 6.2. Consider the transformation $f: \mathbf{C} \rightarrow \mathbf{C}$ given by $f(z) = z^2 + c$. In fig. 6.1 eight Julia sets of f have been plotted in the complex z-plane for different values of c.

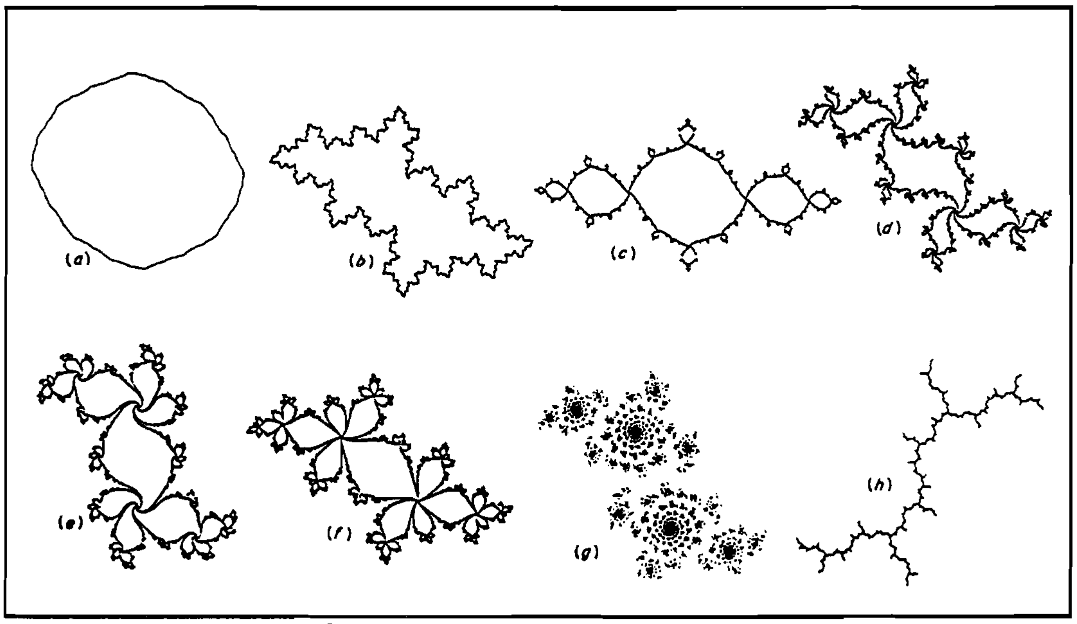


Figure 6.1. Julia sets of $f(z) = z^2 + c$ for different c, plotted in the complex z-plane [FAL90].

The eight Julia sets $J(f)$ in fig. 6.1 have the following characteristics:

a) $c = -0.1 + 0.1i$, f has an attractive fixed point and J is a quasi-circle;

b) $c = -0.5 + 0.5i$, f has an attractive fixed point;

c) $c = -1 + 0.1i$, f has an attractive 2-cycle;

d) $c = -0.2 + 0.75i$, f has an attractive 3-cycle;

e) $c = 0.25 + 0.55i$, f has an attractive 4-cycle;

f) $c = -0.5 + 0.55i$, f has an attractive 5-cycle;

g) $c = 0.66i$, f is chaotic and J is totally disconnected;

h) $c = -i$, $f(0)$ induces a 2-cycle and J is a dendrite. ★

The Julia set is named after the french mathematician Julia, who first studied this kind of sets in the early twentieth century. The Julia-Fatou dichotomy for quadratic polynomials establishes that for any choice of c, the resulting attractor is classified by one of only two possible characteristic cases: Either the attractor is mathematically connected (one piece) or mathematically a cloud of fractal dust (a cloud of infinitely many points, like a Cantor set) [PEI89]. Julia sets of other complex functions, such as: $f(z) = e^{z-1}$, $f(z) = (e^{-1} + 0.1)e^z$ and $f(z) = (1 + ci)\sin z$, can be found in [PEI88].

6.2. Mandelbrot sets

We now specialize to the case of quadratic functions on C. We study Julia sets of polynomials of the form $f(z) = z^2 + c$.

Definition 6.4. Let (C,f) be a dynamical system. The set of parameters $c \in$ C for which the Julia set of f is connected is called the ***Mandelbrot set*** and denoted by $M(f)$. Thus:

$$M(f) = \{ c \in \mathbf{C} : J(f) \text{ is connected} \}$$

$$= \{ c \in \mathbf{C} : \{ f^{(n)}(0) : n = 1,2,3,\ldots,\infty \} \text{ is bounded} \} \qquad \blacklozenge$$

The Mandelbrot set is named after the french mathematician Mandelbrot, who was the first to render this set with a computer at March 1st 1980. His ingenuity was to look at complex numbers (a plane) instead of real numbers (a line) to study iteration processes.

From now on we will refer to J and M as respectively the Julia set and the Mandelbrot set of the function $f(z) = z^2 + c$. In fig. 6.2 M is plotted in the complex c-plane.

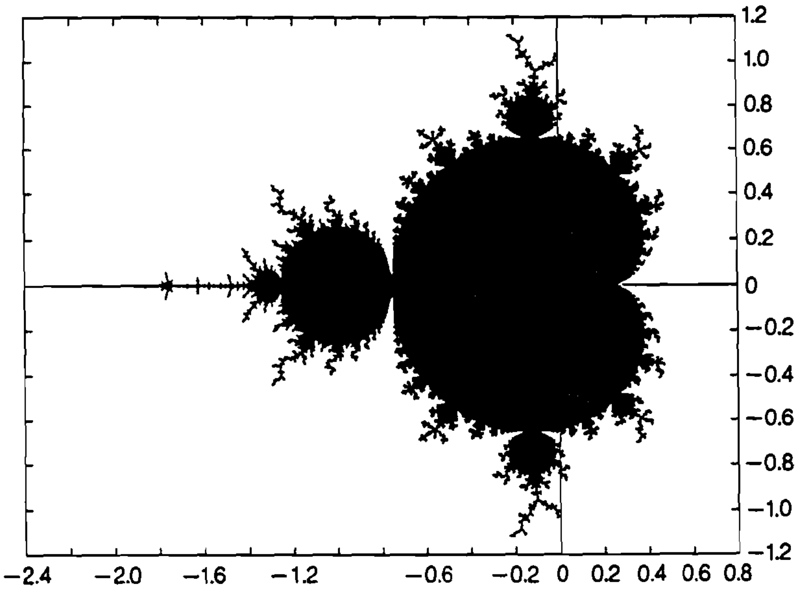


Figure 6.2. Mandelbrot set for $f(z) = z^2 + c$, plotted in the complex c-plane [PEI86].

The Mandelbrot set is a fractal and it combines aspects of self-similarity with the properties of infinite change. The basin boundary of M lies between the stable (black) and instable (white) regions. This border can be zoomed in upon endlessly and still new details will appear. Each time we zoom in on M we are looking at this set at a smaller scale. Here again, we see that the amount of detail is scale-invariant, which is true for all fractals. But if we want to determine the exact basin boundary of M, we will not succeed. This is exactly where chaos is all about and shows what sensitive dependence on initial conditions means. In fig. 6.3 successive zoomings of M are plotted.

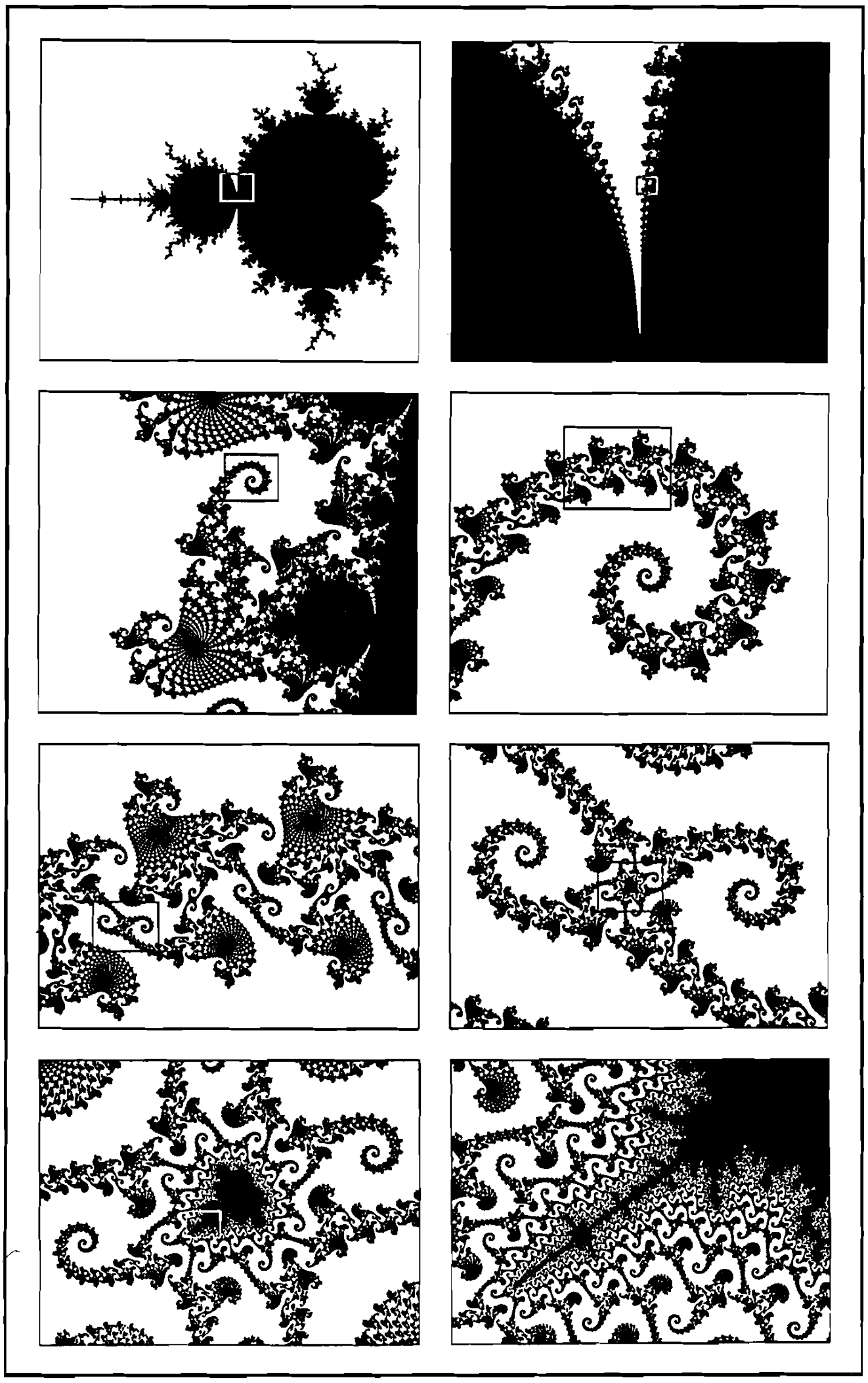


Figure 6.3. Successive zooms into the Mandelbrot set M [PEI92].

Since M consists of all c-values for which J is bounded, M can be considered as a map of the Julia sets. This is illustrated in fig. 6.4, where The Julia sets for various points $c \in M$ are plotted in more detail. In fact, these Julia sets are the same as in fig. 6.1.

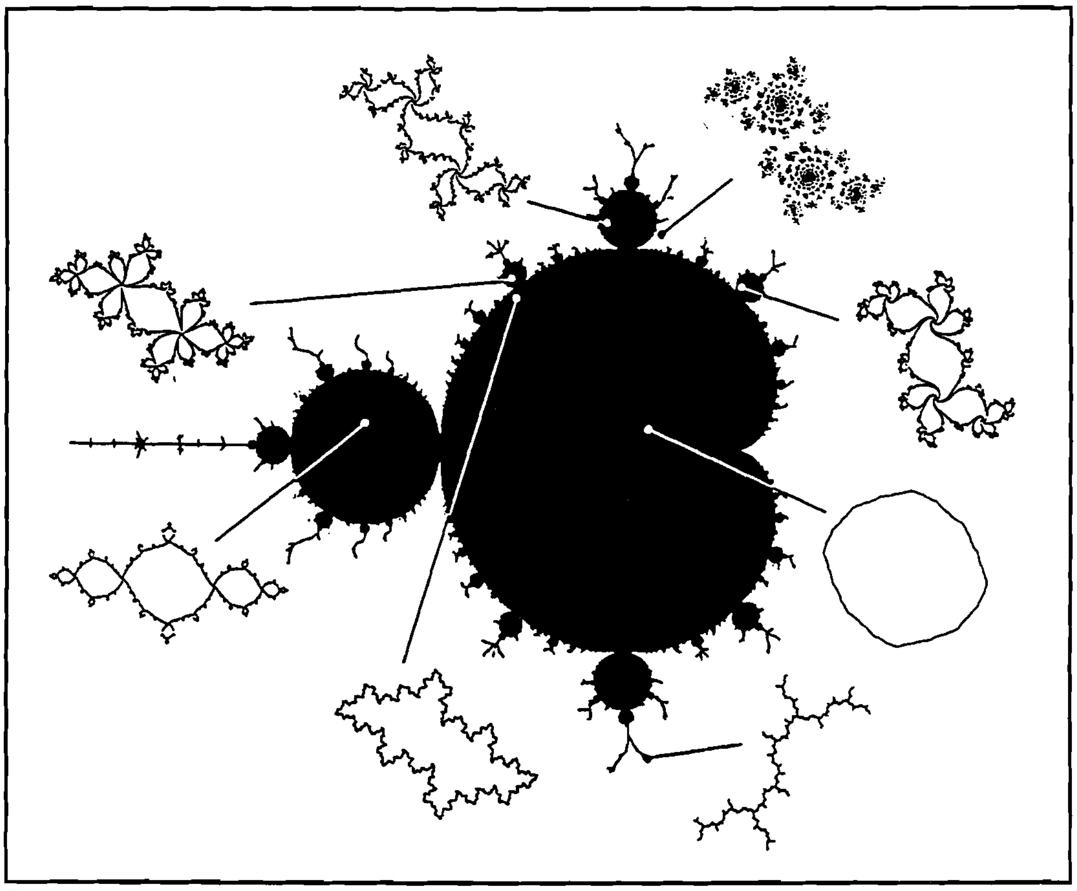


Figure 6.4. Julia sets for various points $c \in M$ [FAL90].

Theorem 6.1. The dynamical systems $(\mathbf{C}, f(z) = z^2 + c)$ and $([0,1] \times [0,1], f(x) = \mu x(1 - x))$ are equivalent.

Proof. Let $f(x) = \mu x(1 - x)$, let $z = -\mu(x - \frac{1}{2})$ and let $f(z) = -\mu(f(x) - \frac{1}{2})$. Then clearly $x = -z/\mu + \frac{1}{2}$ and:

$f(z)$
$= \langle$ substitution $\rangle$
$-\mu(\mu x(1 - x) - \frac{1}{2})$
$= \langle$ substitution $\rangle$
$-\mu(\mu(-z/\mu + \frac{1}{2})(1 - (-z/\mu + \frac{1}{2})) - \frac{1}{2})$
$= \langle$ calculus $\rangle$
$-\mu((-z + \frac{1}{2}\mu)(z/\mu + \frac{1}{2}) - \frac{1}{2})$
$= \langle$ calculus $\rangle$
$z^2 + (-\frac{1}{4}\mu^2 + \frac{1}{2}\mu)$
$= \langle$ substitution $\rangle$
$z^2 + c$ with $c = -\frac{1}{4}\mu^2 + \frac{1}{2}\mu$ ■

So $f(x) = \mu x(1 - x)$ is equivalent to $f(z) = x^2 + c$ with $x = -z/\mu + ½$ and $c = -¼\mu^2 + ½\mu$. If we plot the bifurcation diagram for $f(z) = x^2 + c$ for real parameter values c and compare this diagram to the Mandelbrot set M, we get fig. 6.5.

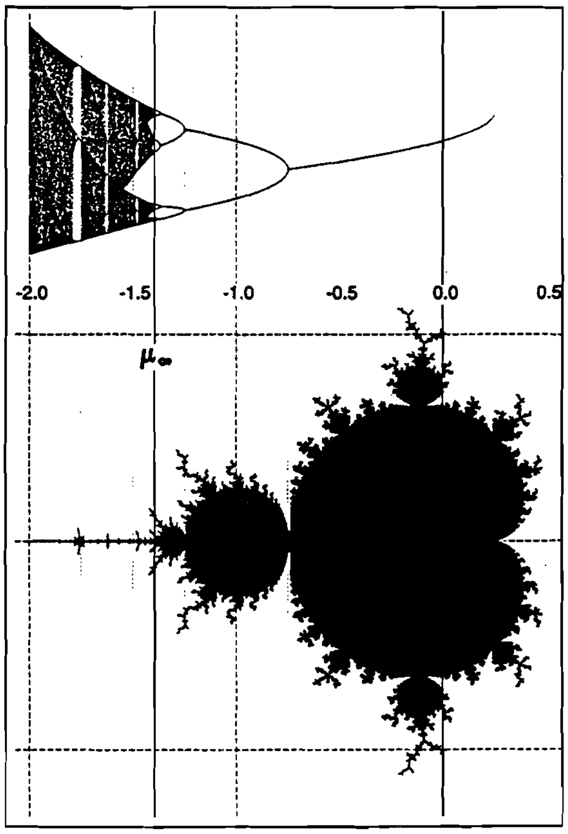


Figure 6.5. Relation between M and the bifurcation diagram of f when c is varied as a real parameter [PEI92].

From fig. 6.5 we see that bifurcations correspond to budding in M. Periodic windows interrupting the chaotic veil of the bifurcation diagram correspond to small copies of M located on the "main antenna". Also notice that the ratio of the radii of the successive buds of M is equal to the Feigenbaum constant $\delta = 4.669201..$ Mandelbrot sets have been observed to appear in many nonlinear dynamical systems, where there is competition between several points of attraction and where the competitive situation depends on a parameter. In this sense the Mandelbrot set is a universal mathematical constant, like the Feigenbaum constant.

Each Mandelbrot set can be classified as a grammar for the corresponding Julia-set language. It can be seen as a dictionary of all possible elements of a dynamical system $(\mathbf{C}, f)$. Magnifying the Mandelbrot set on a point $c \in M$ by about 10^6, we see a structure that is similar to the corresponding Julia set for that particular c-value. In this sense the Mandelbrot set can be seen an an image compressor of infinitely many Julia sets, the entire Julia-set language [PEI89].

In 1982 Douady and Hubbard proved that the Mandelbrot set for the family of dynamical systems $\{ (\mathbf{C}, z^2 + c) : c \in \mathbf{C} \}$ is connected [DOU82].

There is no royal road to geometry.

Euclid (300 B.C.) [DAR80]

Chapter 7

Affine transformations

We have defined a dynamical system to be a 2-tuple (X,f), where f is an arbitrary transformation. We will now focus on a special kind of two-dimensional transformation, which is important and which is frequently used in image processing applications, namely the affine transformation w.

7.1. Geometrical definition

Definition 7.1. Let (X,w) be a dynamical system. A transformation $w : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ of the form:

$$w(x_1,x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f)$$

with $a,b,c,d,e,f \in \mathbf{R}$, is called a (two-dimensional) ***affine transformation***. Often the following equivalent notations are used:

$$w(x) = w(x_1,x_2) = w\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = Ax + t$$

where the 2 × 2 real-valued matrix A deforms Euclidian space relative to the origin and where the vector t denotes a translation or shift, also relative to the origin. The matrix A can always be written in the form:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} r\cos\theta & -s\sin\psi \\ r\sin\theta & s\cos\psi \end{bmatrix}$$

where (r,θ) and $(s,\psi+\frac{1}{2}\pi)$ are the polar coordinates of respectively the points (a,c) and (b,d). ♦

Affine transformations can rotate, scale, translate and shear an image. Parallel lines are invariant under affine transformations and thus an affine transformation maps parallellograms to parallellograms [SNA89]. If an affine transformation is written in the form:

$$w\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} r\cos\theta & -s\sin\psi \\ r\sin\theta & s\cos\psi \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} h \\ k \end{bmatrix}$$

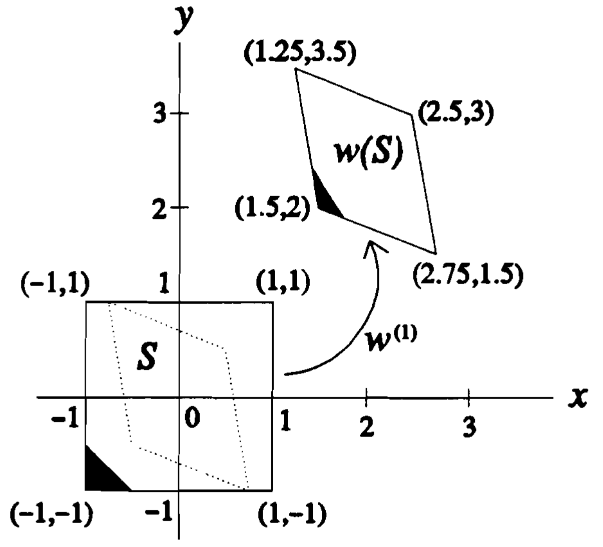
then we will refer to the parameters by the operation they actually perform, thus:

r = x scale factor (*xscaling*);
s = y scale factor (*yscaling*);
θ = x rotation angle (*xrotation*);
ψ = y rotation angle (*yrotation*);
h = x translation (*xtranslation*);
k = y translation (*ytranslation*).

Example 7.1. Consider the affine transformation $w : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, which is given by:

$$w(x_1,x_2) = (0.625x_1 - 0.125x_2 + 2, -0.25x_1 + 0.75x_2 + 2.5)$$

A graphical illustration of this transformation applied on a square is given below:



In case of Carthesian coordiates, we can also describe the transformation in polar coordinates, which can be calculated as follows:

$$r = \sqrt{a^2 + c^2} = \sqrt{(0.625)^2 + (0.25)^2} \approx 0.67$$

$$s = \sqrt{b^2 + d^2} = \sqrt{(-0.125)^2 + (0.75)^2} \approx 0.76$$

$$\theta = \cos^{-1}\left[\frac{a}{r}\right] = \cos^{-1}\left[\frac{0.625}{0.67}\right] \approx -21.8° \approx \sin^{-1}\left[\frac{-0.25}{0.67}\right] = \sin^{-1}\left[\frac{c}{r}\right]$$

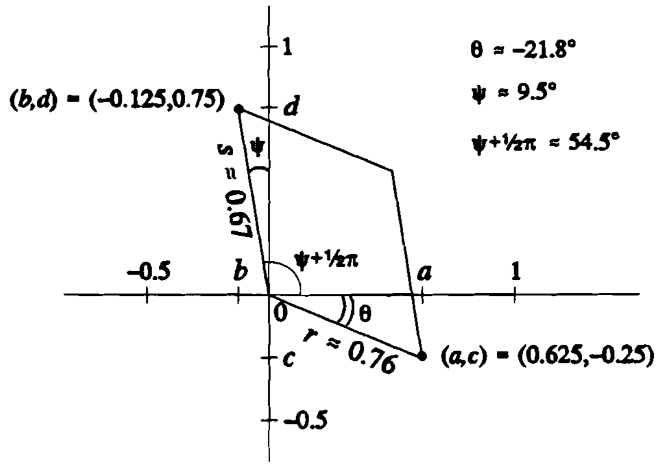
$$\psi = \cos^{-1}\left[\frac{d}{s}\right] = \cos^{-1}\left[\frac{0.75}{0.76}\right] \approx 9.5° \approx \sin^{-1}\left[\frac{0.125}{0.76}\right] = \sin^{-1}\left[\frac{-b}{s}\right]$$

So we can express the affine transformation both in Carthesian coordinates as in polar coordinates:

$w(x_1,x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f) = (0.625x_1 - 0.125x_2 + 2, -0.25x_1 + 0.75x_2 + 2.5)$

$w(x_1,x_2) = (rx_1\cos\theta - sx_2\sin\psi + e, rx_1\sin\theta + sx_2\cos\psi + f)$

$= (0.67x_1\cos(-21.8°) - 0.76x_2\sin(9.5°) + e, 0.67x_1\sin(-21.8°) + 0.76x_2\cos(9.5°) + f)$



Thus, the following points are equivalent (see figure above):

$(a,c) = (0.625,-0.25) \Leftrightarrow (r,\theta) = (0.67,21.8°)$
$(b,d) = (-0.125,0.75) \Leftrightarrow (s,\psi + ½\pi) = (0.76,54.5°)$ ★

Theorem 7.1. Let $w_1 : \mathbb{R}^2 \to \mathbb{R}^2$ and $w_2 : \mathbb{R}^2 \to \mathbb{R}^2$ both be affine transformations then so is $w_3 = w_1 \circ w_2$.

Proof. Let $w_1(x)$ and $w_2(x)$ be affine transformations, which are given by:

$w_1(x) = w_1(x_1,x_2) = (a_1x_1 + b_1x_2 + e_1, c_1x_1 + d_1x_2 + f_1)$
$w_2(x) = w_2(x_1,x_2) = (a_2x_1 + b_2x_2 + e_2, c_2x_1 + d_2x_2 + f_2)$

Then $w_3(x)$
= ⟨ substitution ⟩
$w_1 \circ w_2(x)$
= ⟨ definition of composition ⟩
$w_1(w_2(x))$
= ⟨ substitution ⟩
$w_1(a_2x_1 + b_2x_2 + e_2, c_2x_1 + d_2x_2 + f_2)$
= ⟨ substitution ⟩
$(a_1(a_2x_1 + b_2x_2 + e_2) + b_1(c_2x_1 + d_2x_2 + f_2) + e_1, c_1(a_2x_1 + b_2x_2 + e_2) + d_1(c_2x_1 + d_2x_2 + f_2) + f_1)$
= ⟨ calculus ⟩
$((a_1a_2+b_1c_2)x_1+(a_1b_2+b_1d_2)x_2+(a_1e_2+b_1f_2+e_1), (c_1a_2+d_1c_2)x_1+(c_1b_2+d_1d_2)x_2+(c_1e_2+d_1f_2+f_1))$
= ⟨ substitution ⟩
$(a_3x_1 + b_3x_2 + e_3, c_3x_1 + d_3x_2 + f_3)$

Since a_3,b_3,c_3,d_3,e_3,f_3 are real valued numbers, w_3 is also an affine transformation. ■

7.2. Fixed point analysis

The fixed points of a transformation (see definition 4.3) are very important. They tell us which parts of the space are position-invariant under the transformation. But also, fixed points of transformations correspond to equilibrium solutions of systems of differential equations, i.e. the attributes of attracting and repelling fixed points of transformations relate to the notion of stability of the system [GUL92].

Theorem 7.2. Let $(\mathbf{R}^2, w)$ be a dynamical system with $w : \mathbf{R}^2 \to \mathbf{R}^2$ an affine transformation given by:

$$w(x_1, x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f) = Ax + t$$

Then the fixed point of w is given by:

$$x_f = \left[\frac{bf + (1-d)e}{(1-a)(1-d) - bc}, \frac{ce + (1-a)f}{(1-a)(1-d) - bc}\right]$$

Proof. Suppose $\det(A) \neq 0$ and I is the identity matrix, then:

x_f

= ⟨ substitution ⟩

$Ax_f + t$

= ⟨ calculus ⟩

$(I - A)^{-1}t$

= ⟨ substitution ⟩

$$\left(\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} - \begin{bmatrix}a & b\\c & d\end{bmatrix}\right)^{-1}\begin{bmatrix}e\\f\end{bmatrix}$$

= ⟨ calculus ⟩

$$\begin{bmatrix}1-a & -b\\-c & 1-d\end{bmatrix}^{-1}\begin{bmatrix}e\\f\end{bmatrix}$$

= ⟨ calculus ⟩

$$\frac{1}{(1-a)(1-d) - bc}\begin{bmatrix}1-d & b\\c & 1-a\end{bmatrix}\begin{bmatrix}e\\f\end{bmatrix}$$

= ⟨ calculus ⟩

$$\frac{1}{(1-a)(1-d) - bc}\begin{bmatrix}(1-d)e & bf\\ce & (1-a)f\end{bmatrix}$$

= ⟨ rewrite ⟩

$$\left[\frac{bf + (1-d)e}{(1-a)(1-d) - bc}, \frac{ce + (1-a)f}{(1-a)(1-d) - bc}\right]$$ ■

From the above theorem we see that in case the translation vector t equals zero and w is a contraction, the fixed point is always the origin.

Example 7.2. Consider the affine transformation $w : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ of example 7.1:

$$w(x_1,x_2) = (0.625x_1 - 0.125x_2 + 2, -0.25x_1 + 0.75x_2 + 2.5)$$

The fixed point of w is given by:

$$x_f = \left[\frac{(-0.25)(2.5) + (1-0.75)(2)}{(1-0.625)(1-0.75) - (-0.125)(-0.25)}, \frac{(-0.25)(2) + (1-0.625)(2.5)}{(1-0.625)(1-0.75) - (-0.125)(-0.25)}\right]$$

$$= (3,7)$$ ★

7.3. Affine matrix properties

Definition 7.2. Let $(\mathbf{R}^2,w)$ be a dynamical system and let $w : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ be an affine transformation which is given by $w(x) = Ax + t$. Then the (Euclidian) ***vector norm*** of a vector (point) $x \in \mathbf{R}^2$ is defined as [BAR86b,HOR88]:

$$\|x\| = \sqrt{x_1^2 + x_2^2}$$

We define the ***matrix norm*** of the linear transformation matrix A as [HOR88]:

$$\|A\| = (\ \underline{\text{Max}}\ : x \in \mathbf{R}^2, x \neq 0 : \frac{\|Ax\|}{\|x\|}\ )$$ ♦

Theorem 7.3. Let $(\mathbf{R}^2,w)$ be a dynamical system and let $w : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ be an affine transformation which is defined by $w(x) = Ax + t$. If $\|A\|$ exists then [BAR88a]:

$$(\ \underline{\text{A}}\, x : x \in \mathbf{R}^2 : \|Ax\| \leq \|A\| \cdot \|x\|\ )$$

Proof. Let $x \in \mathbf{R}^2$, $x \neq 0$ and suppose that $\|A\|$ exists, then:

$\|A\| \cdot \|x\|$
$= \langle$ definition 7.2 $\rangle$
$\|A\| = (\ \underline{\text{Max}}\ : x \in \mathbf{R}^2, x \neq 0 : \frac{\|Ax\|}{\|x\|}) \cdot \|x\|$
$= \langle$ calculus $\rangle$
$(\ \underline{\text{Max}}\ : x \in \mathbf{R}^2, x \neq 0 : \|Ax\|\ )$
$\geq \langle$ definition of $\underline{\text{Max}}$ operator $\rangle$
$\|Ax\|$ for all $x \in \mathbf{R}^2$ ■

Theorem 7.4. Let $(\mathbf{R}^2,w)$ be a dynamical system where the affine transformation $w : \mathbf{R}^2 \rightarrow \mathbf{R}^2$ is given by $w(x) = Ax + t$ and where the real-valued matrix A is the same as defined in definition 7.1. Then the area of the parallellogram with adjacent sides (a,b) and (c,d) is equal to $|\det(A)|$.

Proof. In $\mathbb{R}^3$ let $x = (a,b,e)$ and $y = (c,d,f)$. The required area is equal to the norm of the cross product (or vector product) of x and y [BAX86], so:

$\| x \times y \|$
$= \langle$ definition of cross product $\rangle$
$\| (bf - ed, ec - af, ad - bc) \|$
$= \langle$ definition 7.2 $\rangle$
$\sqrt{(bf - ed)^2 + (ec - af)^2 + (ad - bc)^2}$
$= \langle$ switch to $\mathbb{R}^2$, so e=0 and f=0 $\rangle$
$\sqrt{(ad - bc)^2}$
$= \langle$ calculus $\rangle$
$|ad - bc|$
$= \langle$ definition of determinant $\rangle$
$|\det(A)|$ ■

Definition 7.3. Let $(\mathbb{R}^2,w)$ be a dynamical system and let $w : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ be an affine transformation which is given by $w(x) = Ax + t$. Then we define the ***spectrum*** of the matrix A to be the set of all eigenvalues of A, which set is denoted $\sigma(A)$. The ***spectral radius*** of A is defined as [HOR88]:

$$\rho(A) = (\underline{\text{Max}} : \lambda \in \sigma(A) : |\lambda|) \quad ♦$$

The spectral radius of a matrix A is just the radius of the smallest disk centered at the origin in the complex plane, that contains all the eigenvalues of A. So an important area of application of matrix norms is in giving bounds for the spectrum of a matrix.

Theorem 7.5. Let $(\mathbb{R}^2,w)$ be a dynamical system, where the affine transformation $w : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is given by $w(x) = Ax + t$, Then the ***trace*** of A and the ***determinant*** of A can be expressed in terms of the ***eigenvalues*** λ_1 and λ_2 of A [HOR88,KRA87]:

$$\operatorname{tr}(A) = \sum_{i=1}^{2} \lambda_i \qquad \det(A) = \prod_{i=1}^{2} \lambda_i$$

Proof. Suppose I is the identity matrix, then:

$\det(A - \lambda I)$
$= \langle$ substitution $\rangle$
$$\det\left[\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right]$$
$= \langle$ calculus $\rangle$
$$\det\begin{bmatrix} a-\lambda & b \\ c & d-\lambda \end{bmatrix}$$
$= \langle$ calculus $\rangle$
$(a - \lambda)(d - \lambda) - bc$
$= \langle$ calculus $\rangle$
$\lambda^2 - (a + d)\lambda + (ad - bc)$

We can determine the eigenvalues by setting this characteristic polynomial equal to zero, and solve the thus obtained characteristic equation:

$$\lambda^2 - (a + d)\lambda + (ad - bc) = 0$$
$$\Leftrightarrow \langle \text{ square root formula } \rangle$$
$$\lambda_{1,2} = \frac{\frac{1}{2}(a + d) \pm \sqrt{(a + d)^2 - 4(ad - bc)}}{2}$$
$$\Leftrightarrow \langle \text{ calculus } \rangle$$
$$\lambda_{1,2} = \tfrac{1}{2}(a + d) \pm \tfrac{1}{2}\sqrt{(a - d)^2 + 4bc}$$

Now we can calculate:

$$\sum_{i=1}^{2} \lambda_i = \lambda_1 + \lambda_2 = a + d = \mathrm{tr}(A)$$

$$\prod_{i=1}^{2} \lambda_i = \lambda_1 \cdot \lambda_2 = ad - bc = \det(A)$$

■

From this theorem and definition 4.5 it is clear that an affine transformation in $\mathbb{R}^2$ is contractive if and only if $|\lambda_1| < 1$ and $|\lambda_2| < 1$. In general (with an extension of theorem 7.4 and with use of theorem 7.5 and definition 7.1) the following properties hold, where $S \subset \mathbb{R}^2$ is an arbitrary bounded region and where $w : \mathbb{R}^2 \to \mathbb{R}^2$ is given by $w(x) = Ax + t$:

$$\begin{aligned}
\text{area of } w(S) &= s \cdot (\text{area of } S) \\
&= |\lambda_1| \cdot |\lambda_2| \cdot (\text{area of } S) \\
&= |\det(A)| \cdot (\text{area of } S) \\
&= |ad - bc| \cdot (\text{area of } S) \\
&= |rs(\cos\theta\cos\psi + \sin\theta\sin\psi)| \cdot (\text{area of } S) \\
&= |rs\cos(\theta-\psi)| \cdot (\text{area of } S)
\end{aligned}$$

Example 7.3. Consider the affine transformation of example 7.1:

$$w(x_1,x_2) = (0.625x_1 - 0.125x_2 + 2, -0.25x_1 + 0.75x_2 + 2.5) = Ax + t$$

According to theorem 7.5 the eigenvalues of the matrix A can be calculated as follows:

$$\lambda_{1,2} = \tfrac{1}{2}(0.625 + 0.75) \pm \tfrac{1}{2}\sqrt{(0.625 - 0.75)^2 + 4(-0.125)(0.75)} = 0.6875 \pm 0.1875$$

which implies that $\lambda_1 = 0.5$ and $\lambda_2 = 0.875$. The determinant of A can be calculated in two ways:

$$\det(A) = ad - bc = (0.625)(0.75) - (-0.125)(-0.25) \approx 0.437$$

$$\det(A) = rs\cos(\theta-\psi) = 0.67 \cdot 0.76 \cdot \cos(-21.8° - 9{,}5°) \approx 0.437$$

The area of S (the untransformed square) is 4 [*units*] (see figure in example 7.1), so the area of $w(S)$ (the transformed parallellogram) is given by:

$$\begin{aligned} \text{area of } w(S) &= |\lambda_1| \cdot |\lambda_2| \cdot 4 = (0.5)(0.875) \cdot 4 \approx 1.75 \text{ [}\textit{units}\text{]} \\ &= |\det(A)| \cdot 4 = (0.437) \cdot 4 \approx 1.75 \text{ [}\textit{units}\text{]} \\ &= |r s \cos(\theta - \psi)| \cdot 4 \approx (0.437) \cdot 4 \approx 1.75 \text{ [}\textit{units}\text{]} \end{aligned}$$

Notice that the ratio's of the lengths of the axes of the transformed parallellogram and the untransformed square are respectively given by $|\lambda_1|$ and $|\lambda_2|$. ★

7.4. Similitudes

Definition 7.4. Let $(\mathbf{R}^2, w)$ be a dynamical system. The affine transformation $w : \mathbf{R}^2 \to \mathbf{R}^2$ is called a ***similitude*** if it is an affine transformation with ψ set equal to θ and with s set equal to $\pm r$. ♦

A similitude not only preserves parallel lines (as all affine transformations do), but also preserves angles.

Theorem 7.6. Every affine transformation $w : \mathbf{R}^2 \to \mathbf{R}^2$ that is also a similitude which doesn't involve a reflection, can be written as a transformation $f : \mathbf{C} \to \mathbf{C}$ of the form $f(z) = sz + t$, with $s, t \in \mathbf{C}$.

Proof. Let $w : \mathbf{R}^2 \to \mathbf{R}^2$ be a similitude, which is given by:

$$w(x) = w\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} r\cos\theta & -r\sin\theta \\ r\sin\theta & r\cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \qquad \text{with } r, t_1, t_2 \in \mathbf{R},\ \theta \in [0, 2\pi]$$

Since $(\mathbf{R}^2, d_E)$ and $(\mathbf{C}, d_E)$ are equivalent metric spaces (see definition 2.4), we can define an equivalent transformation $f : \mathbf{C} \to \mathbf{C}$ as follows:

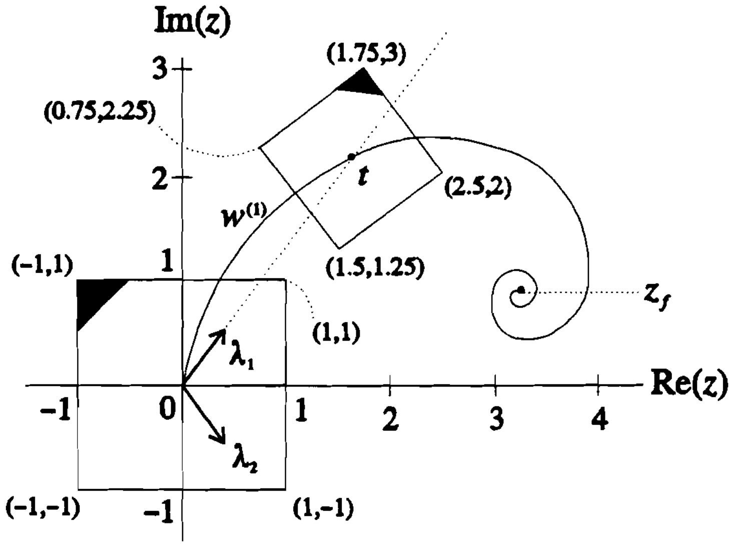
$f(z)$
$= \langle\ z \in \mathbf{C}\ \rangle$
$f(z_1 + iz_2)$
$= \langle$ substitution of $s = \pm r$ and switch to $\mathbf{C}$ $\rangle$
$$\begin{bmatrix} r\cos\theta & ir\sin\theta \\ ir\sin\theta & r\cos\theta \end{bmatrix} \begin{bmatrix} z_1 \\ iz_2 \end{bmatrix} + \begin{bmatrix} t_1 \\ it_2 \end{bmatrix}$$
$= \langle$ calculus $\rangle$
$rz_1(\cos\theta + i\sin\theta) + irz_2(\cos\theta + i\sin\theta) + (t_1 + it_2)$
$= \langle$ calculus $\rangle$
$r(\cos\theta + i\sin\theta)(z_1 + iz_2) + (t_1 + it_2)$
$= \langle$ substitution of $s = r(\cos\theta + i\sin\theta) = re^{\theta i}$, $s, z, t \in \mathbf{C}$ $\rangle$
$sz + t$ ■

It is clear that the fixed point of $f(z) = sz + t$ is given by $z_f = t(1-s)^{-1}$. So $t = (1-s)z_f$ and we can write $f(z) = sz + (1-s)z_f$. The advantage of this notation is that we can express the parameters of the transformation in terms of fixed point z_f and direction s, where $|s|$ is the scaling factor.

<u>*Example 7.4.*</u> Consider the similitude $w : \mathbf{C} \to \mathbf{C}$, which is given by:

$$w(z) = (0.375\,\mathrm{Re}(z) + 0.5\,\mathrm{Im}(z) + 1.625, -0.5\,\mathrm{Re}(z) + 0.375\,\mathrm{Im}(z) + 2.125)$$

The effect of this transformation on a square is given in the figure below:



The fixed point determines the point in space that is invariant under w and is given by:

$$z_f = \frac{t}{1-s} = \frac{1.625 + 2.125i}{1 - (0.375 - 0.5i)} = \frac{(1.625 + 2.125i)(0.625 - 0.5i)}{(0.625 + 0.5i)(0.625 - 0.5i)} \approx 3.244 + 0.805i$$

The (complex) eigenvalues determine the direction of the transformation and are given by:

$$\lambda_{1,2} = \frac{1}{2}(a + d) \pm \frac{1}{2}\sqrt{(a - d)^2 + 4bc} = 0.375 \pm 0.5i$$

The modulus of the eigenvalues determines the scaling of the transformation and is given by:

$$|\lambda_1| = |\lambda_2| = \sqrt{(0.375)^2 + (0.5i)^2} = 0.625$$

Furthermore, we can write:

$$f(z) = sz + t = \lambda_2 z + (e + if) = (0.375 - 0.5i)z + (1.625 + 2.125i)$$

$$= sz + (1-s)z_f \text{ with } s = 0.375 - 0.5i \text{ and } z_f = 3.244 + 0.805i.$$ ★

7.5. Contraction mappings

Theorem 7.7. Let $w : X \to X$ be a contractive affine transformation on the complete metric space (X,d) with contractivity factor $s \in [0,1)$. Then $w : \mathcal{H}(X) \to \mathcal{H}(X)$, which is defined by:

$w(K) = (\,\underline{A}\,K : K \in \mathcal{H}(X), x \in K : w(x)\,)$

is a ***contraction mapping*** on $(\mathcal{H}(X),h)$ with contractivity factor s [BAR88a].

Proof. Let $K,L \in \mathcal{H}(X)$, then:

$d(w(K),w(L))$
$= \langle$ definition 2.18 $\rangle$
$(\,\underline{\mathrm{Max}} : x \in K : d(w(x),w(L))\,)$
$= \langle$ definition 2.17 $\rangle$
$(\,\underline{\mathrm{Max}} : x \in K : (\,\underline{\mathrm{Min}} : y \in L : d(w(x),w(y))\,)\,)$
$\leq \langle$ w is contractive, definition 4.5 $\rangle$
$(\,\underline{\mathrm{Max}} : x \in K : (\,\underline{\mathrm{Min}} : y \in L : sd(x,y)\,)\,)$
$= \langle$ definition 2.18 $\rangle$
$sd(K,L)$
$\Rightarrow \langle$ similarly $\rangle$
$d(w(L),w(K)) \leq sd(L,K)$

Hence, we can now complete the proof for the Hausdorff distance:

$h(w(K),w(L))$
$= \langle$definition 2.19 $\rangle$
$(\,\underline{\mathrm{Max}} : K,L \in \mathcal{H}(X) : (d(w(K),w(L)), d(w(L),w(K)))\,)$
$\leq \langle$ first part of this proof $\rangle$
$(\,\underline{\mathrm{Max}} : K,L \in \mathcal{H}(X) : (sd(K,L), sd(L,K))\,)$
$= \langle$ definition 2.19 $\rangle$
$sh(K,L)$ ■

All really efficient power depends, and is really to be deduced from the sources of forms.

Francis Bacon (1561 - 1626) [ADL77]

Chapter 8

IFS-decoding

In this chapter we will address dynamical systems whose behaviour is governed by a set of affine transformations, a so called IFS. This concept can be considered as the main key for all applications of fractal operations on images.

8.1. Iterated Function Systems

Definition 8.1. An ***Iterated Function System (IFS)*** is a 3-tuple (X,W,P), where $X = (X,d)$ is a complete metric space and $W = \{ w_i : i = 1,2,3,\dots,N \}$ is a finite set of affine transformations and $P = \{ p_i : i = 1,2,3,\dots,N \}$ is a finite set of assigned probabilities, such that probability p_i is assigned to affine transformation w_i and [BAR88a,STA91]:

$$(\underline{A}\ i : 1 \le i \le N : p_i > 0) \quad \wedge \quad \sum_{i=1}^{N} p_i = 1$$

The contractivity factor of the IFS is given by $s = (\underline{\mathrm{Max}} : 1 \le i \le N : s_i)$. If $s \in [0,1)$ then the IFS is called ***hyperbolic***. If the probabilities are neglected then the IFS is ***deterministic***. ♦

For comparative reasons we will have to use the term ***nondeterministic IFS*** with which we state that either the IFS is known to be nondeterministic or we simply don't (want to) know whether or not the IFS is deterministic. From now on, if we talk just about an IFS we mean a nondeterministic IFS.

Definition 8.2. Let (X,W,P) be a hyperbolic IFS, then a ***collage*** is a transformation $w : \mathcal{H}(X) \rightarrow \mathcal{H}(X)$ which is defined by:

$$(\underline{A}\ w,K : w \in W,\ K \in \mathcal{H}(X) : w(K) = \bigcup_{i=1}^{N} w_i(K)) \qquad ♦$$

Theorem 8.1. Let (X,W,P) be a hyperbolic IFS with contractivity factor s, then the collage $w : \mathcal{H}(X) \rightarrow \mathcal{H}(X)$ is a contraction mapping on the complete metric space $(\mathcal{H}(X),h)$ with contractivity factor s, that is:

$$(\underline{A}\ K,L : K,L \in \mathcal{H}(X) : h(w(K),w(L)) \le s h(K,L))$$

Proof. The proof is based on mathematical induction. The induction hypothesis is given by:

$$\mathrm{IH}(N) : h(\bigcup_{i=1}^{N} w_i(K), \bigcup_{i=1}^{N} w_i(L)) \le (\,\underline{\mathrm{Max}} : 1 \le i \le N : s_i\,)h(K,L)$$

Base: $h(\bigcup_{i=1}^{N} w_i(K), \bigcup_{i=1}^{N} w_i(L))$
$= \langle\ \mathrm{IH}(1)\ \rangle$
$h(\bigcup_{i=1}^{1} w_i(K), \bigcup_{i=1}^{1} w_i(L))$
$= \langle$ calculus $\rangle$
$h(w_1(K), w_1(L))$
$\le \langle$ theorem 8.1 $\rangle$
$s_1 h(K,L)$

Step: $h(\bigcup_{i=1}^{N+1} w_i(K), \bigcup_{i=1}^{N+1} w_i(L))$
$= \langle$ definition of $\underline{\text{Union}}$ operator $\rangle$
$h(\bigcup_{i=1}^{N} w_i(K) \cup w_{N+1}(K), \bigcup_{i=1}^{N} w_i(L) \cup w_{N+1}(L))$
$\le \langle\ h(B \cup C, D \cup E) \le (\,\underline{\mathrm{Max}}\ (h(B,D), h(C,E)))$, property of Hausdorff distance [BAR88a] $\rangle$
$h(\,\underline{\mathrm{Max}} :: (\ h(\bigcup_{i=1}^{N} w_i(K), \bigcup_{i=1}^{N} w_i(L)), h(w_{N+1}(K), w_{N+1}(L))\ )\ )$
$\le \langle\ \mathrm{IH}(N)$ holds $\rangle$
$(\,\underline{\mathrm{Max}} :: ((\,\underline{\mathrm{Max}} : 1 \le i \le N : s_i\,)h(K,L), h(w_{N+1}(K), w_{N+1}(L)))\ )$
$\le \langle$ theorem 8.1 $\rangle$
$(\,\underline{\mathrm{Max}} :: ((\,\underline{\mathrm{Max}} : 1 \le i \le N : s_i\,)h(K,L), s_{N+1}h(K,L))\ )$
$= \langle$ definition of $\underline{\mathrm{Max}}$ operator $\rangle$
$(\,\underline{\mathrm{Max}} : 1 \le i \le N : s_i\,)h(K,L)$
$= \langle$ definition 8.1 $\rangle$
$sh(K,L)$ ■

Definition 8.3. Let (X, W, P) be a hyperbolic IFS with contractivity factor s_i. Then we say that the IFS obeys the ***average contractivity condition*** if [BAR88d]:

$$\prod_{i=1}^{N} s_i^{p_i} < 1$$
♦

Definition 8.4. Let (X, W, P) be a hyperbolic IFS, then the associated ***IFS-code*** is a finite set $\{\, a_i, b_i, c_i, d_i, e_i, f_i, p_i : i = 1,2,3,\ldots,N \,\}$ such that the average contractivity condition is obeyed [BAR88d]. The first six elements of the IFS-code are the parameters of affine transformations as defined in definition 7.1. ♦

Theorem 8.2. Let (X,W,P) be a hyperbolic IFS, then its unique attractor $A \in \mathcal{H}(X)$ obeys:

$$A = w(A) = \bigcup_{i=1}^{N} w_i(A)$$

and is given by $A = \lim_{n\to\infty} W^{(n)}(B)$ for any $B \subset A$ or in case of a deterministic IFS, for any $B \subset \mathcal{H}(X)$.

Proof. This is a direct result of definition 8.1 together with theorem 8.1. ■

This means that, in case of a deterministic IFS, it doesn't matter with which value B we initialize the IFS, the attractor will always be the same. In case of a nondeterministic IFS we initialize the IFS with a compact set $A_0 \subset \mathcal{H}(X)$ and then compute successively:

$$A_{n+1} = \bigcup_{i=1}^{N} w_i(A_n) \quad \text{for} \quad n = 1,2,3,...$$

Thus, we construct a sequence $\{ A_n : n = 0,1,2,3,... \} \subset \mathcal{H}(X)$ which converges to the attractor of the IFS in the Hausdorff metric [CUL90].

In case of a nondeterministic IFS we initialize the IFS with a point $x_0 \in X$ and then choose recursively and independently:

$$x_{n+1} \in \{ w_i(x_n) : i = 1,2,3,...,N \} \quad \text{for} \quad n = 1,2,3,...$$

where the probability of the event $x_{n+1} = w_i(x_n)$ is p_i [CUL90]. Thus we construct a sequence $\{ x_n : n = 0,1,2,3,... \} \subset X$ which converges to the attractor of the IFS in the Hausdorff metric.

Example 8.1. An IFS whose attractor is a Sierpinski triangle (see fig. 3.2) can be given by the 3-tuple: $(\mathbb{R}^2,\{w_1,w_2,w_3\},\{p_1,p_2,p_3\})$, where:

$$w_1\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix}, p_1 = 0.33$$

$$w_2\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}1\\50\end{bmatrix}, p_2 = 0.33$$

$$w_3\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}50\\50\end{bmatrix}, p_3 = 0.34$$

The IFS-code is given in table 8.1 below:

Table 8.1. IFS-code for a Sierpinski triangle.

i	*a*	*b*	*c*	*d*	*e*	*f*	*p*
1	0.5	0	0	0.5	1	1	0.33
2	0.5	0	0	0.5	1	50	0.33
3	0.5	0	0	0.5	50	50	0.34

We disregard the probabilities, so the IFS becomes deterministic. We initialize the IFS with a compact set $A_0 \subset \mathbb{R}^2$ and then compute successively:

$$A_{n+1} = \bigcup_{i=1}^{N} w_i(A_n) \qquad \text{for} \quad n = 1,2,3,\ldots$$

In fig. 8.1 below, this operation is illustrated. Left we choose A_0 to be a rectangle, right we choose A_0 to be a filled in triangle (we may also have choosen a point). In both cases we get the same (unique) attractor of the IFS.

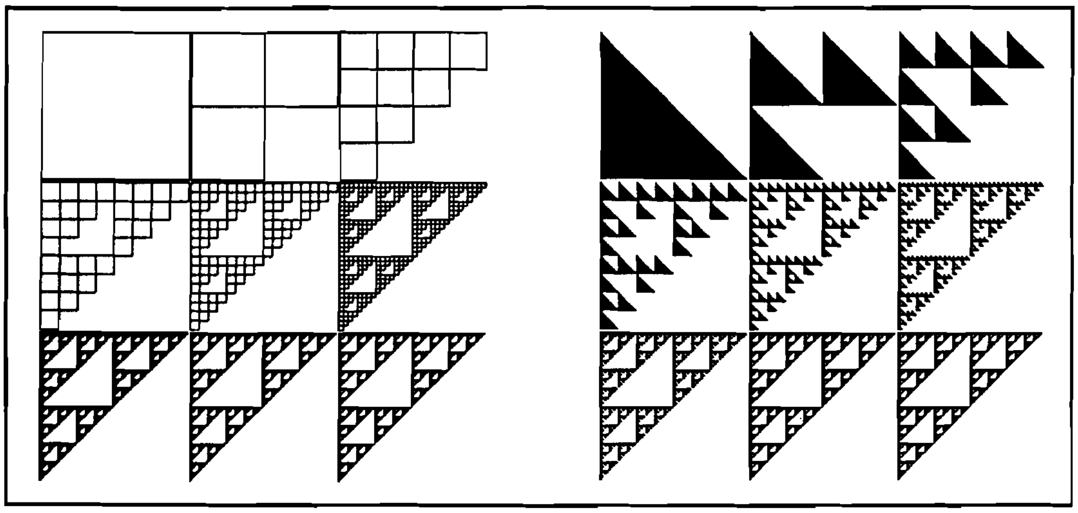


Figure 8.1. Decoding of IFS-code of table 8.1 for A_0 is a rectangle and a filled in triangle [BAR88a]. ★

In case of a nondeterministic IFS not only the affine transformations, but also the associated probabilities influence the structure of the attractor. We can show this by the next example.

Example 8.2. Consider four IFS-codes which only differ in their probabilities and which all generate a Sierpinki triangle (all put in table 8.2 below).

Table 8.2. IFS-codes for 4 Sierpinski triangles with different associated probabilities.

i	*a*	*b*	*c*	*d*	*e*	*f*	*p1*	*p2*	*p3*	*p4*
1	0.5	0	0	0.5	-50	-50	0.33	0.20	0.40	0.40
2	0.5	0	0	0.5	50	-50	0.33	0.40	0.20	0.40
3	0.5	0	0	0.5	0	50	0.34	0.40	0.40	0.20

For each IFS-code we render the attractor of the associated IFS, by initializing the IFS with a point $x_0 \in \mathbb{R}^2$ and then choosing recursively and independently:

$$x_{n+1} \in \{\, w_i(x_n) : i = 1,2,3,\ldots,N \,\} \text{ for } n = 1,2,3,\ldots$$

In fig. 8.2, these operations are illustrated. In each case the computation was halted after relatively few iterations, to prevent the image from getting "saturated". From the result we see that the rendered attractors are the same sets, but that the distribution of the points within the sets is different.

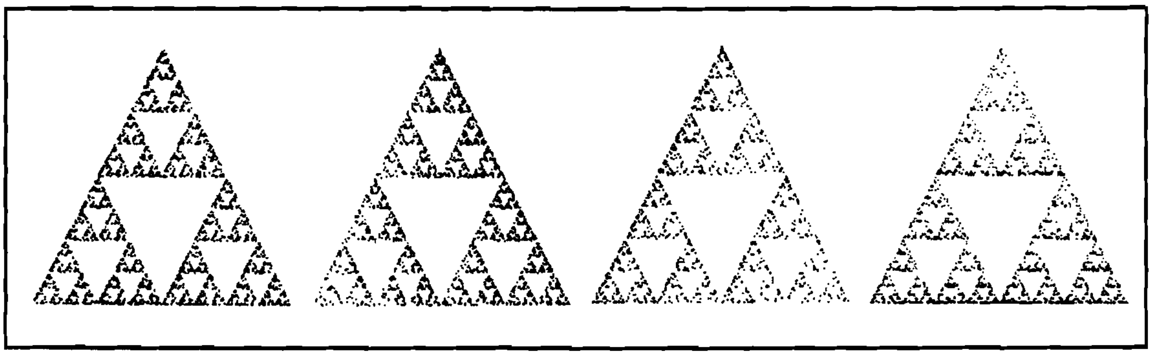


Figure 8.2. Attractors of IFS-codes of table 8.2, from left to right: p1, p2, p3 and p4. ★

8.2. Condensation

Definition 8.5. Let (X,W,P) be a hyperbolic IFS which is initialized by a compact set $C \subset \mathcal{H}(X)$. Define the ***condensation set*** as:

$$\begin{aligned} C_n &= C \cup w^{(1)}(C) \cup w^{(2)}(C) \cup \ldots \cup w^{(n)}(C) \\ &= C \cup \bigcup_{j=1}^{n} w^{(j)}(C) \\ &= w_0(C) \cup \bigcup_{j=1}^{n} w^{(j)}(C) \end{aligned}$$

Then we call $w_0 : \mathcal{H}(X) \rightarrow \mathcal{H}(X)$ a ***condensation transformation*** (which is not an affine transformation) and we call (X,W_C,P_C) with $W_C = w_0 \cup W$ and $P_C = p_0 \cup P$ and $(\underline{S}\ i : 1 \leq i \leq N : p_i)$ and $p_0 = 1$ a ***hyperbolic IFS with condensation*** [BAR85,BAR88d]. ♦

Observe that a condensation transformation $w_0 : \mathcal{H}(X) \rightarrow \mathcal{H}(X)$ is an identity mapping on the metric space $(\mathcal{H}(X),h)$ with contractivity factor equal to zero and scaling factor equal to one. It possesses a unique fixed point, namely the condensation set. In practice, condensation means that we don't clear the computer screen between two successive iterations. It is obvious that the condensation set $\{ C_n : n = 0,1,2,\ldots,n \}$ is a Cauchy sequence in $\mathcal{H}(X)$ which converges to the attractor of the IFS, with contractivity factor s equal to the contractivity factor of the associated hyperbolic IFS without condensation. Furthermore, all definitions and theorems about iterated function systems also hold for iterated function systems with condensation.

Example 8.3. Consider the IFS-code of table 8.3 below.

Table 8.3. IFS-code with condensation transformation.

i	a	b	c	d	e	f	p
0	1	0	0	1	0	0	1
1	0.55	0	0	0.55	150	0	1

We choose $C \subset \mathbb{R}^2$ to be a Sierpinski triangle and we render the attractor of the associated IFS with condensation. The result is given in fig. 8.3 below.

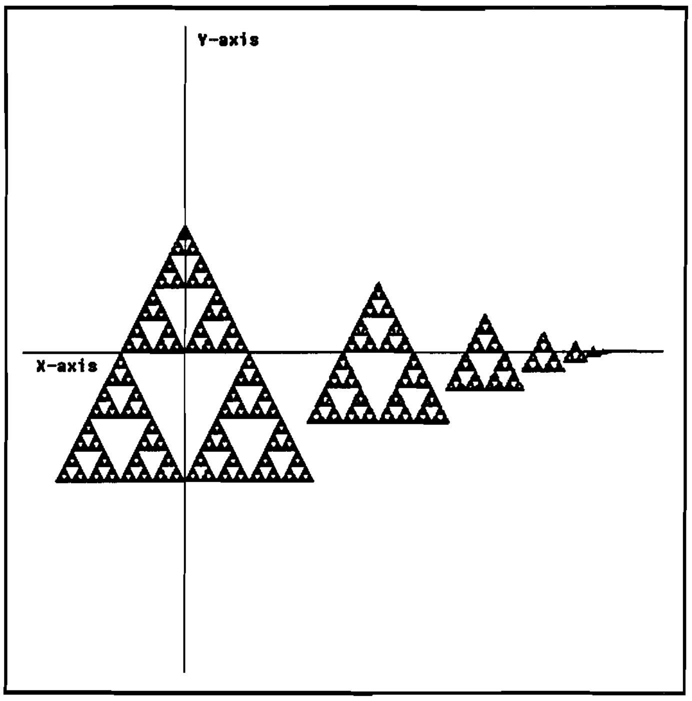


Figure 8.3. Attractor of the IFS-code of table 8.3. ★

We can immediately deduce the fact that if we initialize a hyperbolic IFS with condensation with a compact set $C \subset A$, the condensation set C_n equals the attractor A of the IFS.

8.3. An algorithmic approach

As we have seen in the preceding paragraphs, we can distinguish between deterministic and nondeterministic IFS-decoding. Both methods can either be applied on an IFS with or without condensation. We will give simple algorithmic descriptions of those methods, which do not take into account any efficiency or implementation considerations.

Definition 8.6. Let $(\mathbb{R}^2,d)$ be a complete metric space. A ***pixel*** (picture element) is a discretization $(x,y,z) \in N^3$ of a real valued point $(x,y,z) \in \mathbb{R}^3$, i.e. a mapping $\pi : \mathbb{R}^3 \rightarrow N^3$, where (x,y) denotes the Carthesian coordinate and z denotes the intensity of that point [FOL90]. ♦

Definition 8.7. Let (X,d) be a complete metric space. If $X = [x_{left},x_{right}] \times [y_{upper},y_{lower}] \subset N^2$, then we refer to (X,d) as a ***pixel space***. The elements of a pixel space are pixels [FOL90]. ♦

Definition 8.8. Let (X,d) be a pixel space, i.e. $X = [x_{left},x_{right}] \times [y_{upper},y_{lower}] \subset N^2$. A ***raster image*** can be defined as an assignment Π which obeys:

$$(\underline{A}\, x,y : x,y \in X : (\underline{E}\, b,z : b \in \mathbb{N}, z \in [0,2^b-1] : \Pi(x,y) = z))$$

where b denotes the amount of bitplanes of the framebuffer. If $b = 0$ then the image is called a ***monochrome image***, else it is called a ***color image*** or a ***greytone (greylevel) image***. ♦

So a monochrome image can be considered to be a special case of a color image (namely for $b = 0$). Typical values of b are 0, 4 and 8. The algorithms which are presented hereafter, operate on the pixel spaces (***Screen***,d) and (***Pixmap***,d) and generate raster images. ***Screen*** is the graphics computer screen and ***Pixmap*** is a framebuffer, i.e. a memory storage or buffer that can contain pixels. Typical values for ***Screen*** and ***Pixmap*** are [STE89]:

$[0,319] \times [0,199]$	(CGA)
$[0,639] \times [0,349]$	(EGA)
$[0,639] \times [0,479]$	(VGA)
$[0,719] \times [0,347]$	(HERCULES)
$[0,1257] \times [0,985]$	(HIGH RESOLUTION GRAPHICS)

In the following descriptions, C denotes an initializing set consisting of at least one pixel.

(i) Deterministic IFS-decoding with or without condensation:

```
plot C on screen
for iteration := 1 to maxiterations do
    [ clear pixmap
      copy screen to pixmap
      if not condensation then clear screen
      for all pixels in pixmap do
          [ for transformation := 1 to N
                [ pixel := transformed pixel
                  plot pixel on screen
                ]
          ]
    ]
```

(ii) Nondeterministic IFS-decoding with condensation:

```
plot C on screen
for iteration := 1 to maxiterations do
    [ clear pixmap
      copy screen to pixmap
      for all pixels in pixmap do
          [ choose a transformation according to its probability
            pixel := transformed pixel
            plot pixel on screen
          ]
    ]
```

(iii) Nondeterministic IFS-decoding without condensation:

```
plot C on screen
for iteration := 1 to maxiterations do
    [ choose a transformation according to its probability
      pixel := transformed pixel
      plot pixel on screen
    ]
```

In [MON90] a ***minimum point plotting algorithm*** is presented, which plots a pixel not more than once. It must be noticed that images which are rendered by IFS-decoding contain infinite detail, that is, the amount of detail is scale-invariant, which can be seen if we keep zooming in on the image. The reason for this is the fact that IFS-attactors are fractal structures.

It is clear that the different described IFS-decoding algorithms each yield a different amount of new pixels per iteration. Suppose we have N transformations and in all cases we initialize the IFS with a set C consisting of one pixel (point). In table 8.4 below the maximal amount of plotted pixels after each iteration is given. We stress the word "maximal", because it may be possible that a pixel is plotted exactly on an already existing pixel.

In case the framebuffer contains more than one bitplane, the nondeterministic IFS-decoding algorithm can create real color or greytone images by using this property. Each time a pixel is plotted exactly on an already existing pixel, we increase the color or greylevel value of this pixel with one [*bit*]. This means that if the framebuffer contains b [*bitplanes*], we can plot 2^b-1 [*pixels*] on the same location before that pixel gets saturated. So we could in this case extend the nondeterministic IFS-decoding algorithm with a guard that provides the rendered image from getting saturated by comparing the maximum rendered greylevel so far with the number 2^b-1.

Table 8.4. Pixel yields for the four IFS-decoding algorithms (subscript C denotes condensation).

	deterministic		nondeterministic	
iteration	(X,W_C,P_C)	(X,W,P)	(X,W_C,P_C)	(X,W,P)
0	$\|C\|$	1	$\|C\|$	1
1	$N+\|C\|$	N	$2\|C\|$	2
2	$(N+\|C\|)^2$	N^2	$4\|C\|$	3
3	$(N+\|C\|)^2$	N^3	$8\|C\|$	4
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
n	$(N+\|C\|)^n$	N^n	$2^n\|C\|$	$n+1$

From table 8.4 it is clear that from right to left the amount of produced pixels per iteration step increases. So, in case of flexible parallellization of the decoding process, the overall decoding speed of all four methods will not be the same anymore. In that case, deterministic IFS-decoding with condensation is the fastest method and nondeterministic IFS-decoding without condensation is the slowest method. The parallel computational time complexities of the four methods are respectively given by (from left to right according to table 8.4) $O(\log_{N+|C|}n)$, $O(\log_N n)$, $O(|C|^{-1}\log_2 n)$ and $O(n)$.

In fact, the serial computations will also differ if pixmaps are used to store the intermediate iteration results, since plotting pixels into a pixmap is much faster than plotting pixels on a screen. Moreover, if $|C| > 1$ [*pixels*] then IFS-decoding with condensation is really faster than IFS-decoding without condensation. Nondeterministic IFS-decoding is also slower than deterministic IFS-decoding because some transformations can have very low probabilities, which implies that only relatively few pixels will be transformed according to these transformations. In [HOR90] some more comments are made about parallellization of the IFS-decoding process on an array processor.

8.4. Data compression

It is obvious that IFS-codes constitute data in a highly compressed format and therefore IFS-encoding can be considered as a powerful data compression tool: the (redundant) pixel information of the target image can be represented by an IFS-code. Although IFS-encoding is subject of chapter 10, we now are already able to compare the data compression performance of IFS-encoding with that of other encoding schemata, such as ***Huffman-encoding***, ***runlength-encoding***, ***programmed compression*** and ***adaptive compression***. The performance of each method strongly depends on the type of data to be compressed. In [WEL84] a comparision is made between compression of English text, Cobol files, floating point arrays, formatted scientific data, system log data, program source code and object code. Compression of image data is mostly achieved by runlength-encoding, which has a typical average compression ratio of 10:1. In [REG81] some more data compression techniques are evaluated.

Under UNIX™ there exists an advanced data compression program, which can be invoked with the command: "`compress -v filename.ext`". This results in a file `filename.ext.Z` which contains the data of the file `filename.ext` in a compressed format. The decompression is invoked with the command: "`uncompress -v filename.ext`". For practical reasons, we will compare the data compression performance of IFS-encoding only to that of this UNIX™ compression program. It must be noticed that an IFS-code itself (which already represents a compressed image) can be further compressed by the UNIX™ compression program, which yields an average extra compression ratio of 2:1. In table 8.5 below, the data compression performances on several images are listed. The according images and their IFS-codes are contained in appendix C. The image sources contain the coordinate pairs for the pixels belonging to the "on"-set of the image. In table 8.5 the amount of affine transformations the IFS-code contains is denoted by N and the amount of necessary iterations to render a "good" picture by deterministic IFS-decoding is denoted by n.

Table 8.5. Data compression performances of UNIX™ compress and IFS-encoding.

	source	UNIX™ compress		IFS-encoding			
image	[*bytes*]	[*bytes*]	ratio	N	n	[*bytes*]	ratio
chain	106036	38613	2.7:1	9	6	413	256.7:1
dragon	703615	243225	2.9:1	2	21	72	9772.4:1
fern	221917	75313	2.9:1	4	18	174	1275.4:1
france	672111	227611	3.0:1	31	6	482	453.5:1
koch	16539	6060	2.7:1	4	8	180	91.9:1
leaf	632208	214409	2.9:1	4	8	186	3399.0:1
rect	316808	102225	3.1:1	4	8	180	1760.0:1
square	219664	70761	3.1:1	8	6	354	620.5:1
tree	117641	41823	2.8:1	5	8	238	494.3:1
triangle	50720	17130	3.0:1	3	9	134	378.5:1
twig	18042	6996	2.6:1	3	10	144	125.3:1

From table 8.5 we see that IFS-encoding always yields much better compression ratios than UNIX™ compress, but the latter one yields more constant compression ratios and therefore its performance is better predictable. Since there is no unique IFS-code for a target image, IFS-encoding can result in different IFS-codes. In general, the amount N of affine transformations in the IFS-code must be an integer between 2 and the amount of pixels a source image A contains, thus $2 \leq N \leq |A|$.

The amount of necessary iterations n to render an image consisting of $|A|$ by means of IFS-decoding, depends strongly on N. Suppose we apply deterministic IFS-decoding without condensation. If we decrease N, we have to increase n to render enough pixels. Suppose we have to encode the image "dragon" (see table 8.5 and appendix C). We therefore need to render 89587 [*pixels*]. So the IFS-code must contain at least 2 and at most 89587 affine transformations. The amount of necessary iterations n as function of N can be expressed by the following relationship (see also fig. 8.4 below):

$$n = \frac{\ln|A|}{\ln N} \quad [\textit{iterations}]$$

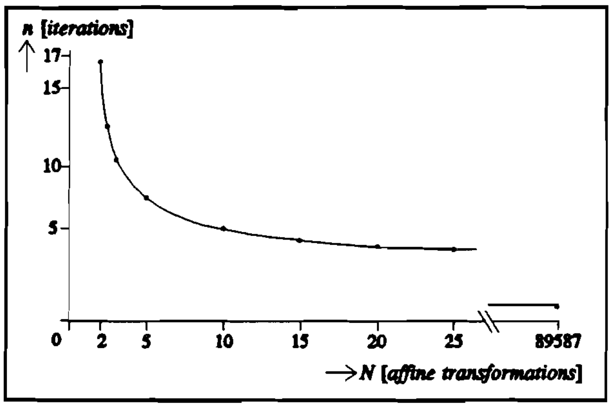


Figure 8.4. Relationship between N and n for $|A|$ = 89587.

So, if we are able to parallellize the IFS-decoding process by distributing the affine transformations over the array processors, the fastest IFS-decoding will be achieved for the minimal amount of iterations n, thus for maximal N. This implies a ***compression-iteration trade off***, i.e. the compression ratio and the amount of iterations are competitive factors.

Finally, we will briefly comment on a comparision between IFS-encoding (also known as the ***fractal transform*** [BAR90]) and the ***Discrete Cosine Transform (DCT)***. When making comparisions, it is important to remember that image compression methods only work well with image data. Geometric images such as those produced by CAD-systems are most efficiently stored as geometric data: as polygons or other primitives.

The DCT can achieve typical compression ratios around 40:1 and can only achieve higher compression ratios by displaying the image as bigger and bigger blocks. In extrema, a highly compressed image becomes an unrecognizeable mass of squares. By contrast, an overcompressed fractal transform picture just looks fuzzy [WRI92].

Aestetic and psychological factors aside, fractals have other advantages. Because a fractal is like an infinite series, decompression is independent of display resolution. It doesn't matter how closely we zoom in onto the image, the level of detail remains the same, because it is scale-invariant. This illustrates an important point. There is no fundamental limit to the effectiveness of the fractal transform. While DCT and similar methods run into solid barriers, the fractal transform can go on improving idefinitely. Image quality can be improved by simply increasing the amount of iterations of the IFS-decoding process. Also, enlargement of the image by a factor k can simply be achieved by multiplying the translation vector t of each affine transformation in the IFS by this factor k. This doesn't increase the IFS-code. All other methods would in this case result in more code.

Thus, the fractal transform has two main advantages above <u>all</u> other data compression techniques: its ability to benefit from improved processors and its resolution independence. The latter makes it automatically compatible with ever-improving graphics display standards.

Despite these benefits, there are situations where fractal data compression remains unsatisfactory. It doesn't work well for images with a low information content, such as engineering drawings or text, which may limit its embedded applications such as scanners or fax-devices [WRI92]. In general, promising applications for the fractal transform seem to be the defence industry (flight simulators, sattelite data) and the chip industry (recursive VLSI-design).

Some more information about the fractal transform applied as a ***block encoding scheme*** (compression ratio of about 40:1) can be found in [BEA91,JAC90,WAI90]. Another interesting approach is that of a ***yardstick traversal scheme*** (compression ratio of about 12:1), which encodes the information in each scanline of a picture [WAL86,YAN88,ZHA91].

8.5. Finite affine automata

We have seen an algorithmic approach of the IFS-decoding process. But we can also use an approach that is similar to the one used in par. 3.1. We can define the IFS as a language generator or a finite automaton.

Definition 8.9. Let (X,d) be a complete metric space and let (X,W,P) be a deterministic IFS operating on this space. Define (conform [CUL90,LEW81]) a ***deterministic finite affine automaton*** M_D as a 5-tuple (Q,W,δ,s,F), where:

Q is a finite set of states;

W is a finite set of affine transformations (i.e. the alphabet);

$\delta : Q \times W \rightarrow Q$ is the transition function;

$s \in Q$ is the initial state (an initialization for the IFS);

$F \subset Q$ is the set of final states (attractors). ♦

Example 8.4. Consider the deterministic finite affine automaton $M_D = (Q,W,\delta,s,F)$, where:

Q	=	$\{C,A_1,A_2,A_3,A_4\}$
W	=	$\{w_1,w_2,w_3\}$
s	=	C
F	=	$\{A_3\}$
δ	=	tabulated in table 8.6

The state transition diagram of M_D is a digraph $G_D = (V,E)$ with $V = Q$ and $E \subset Q \times Q$ and is given in fig. 8.5.

Table 8.6. The transition function δ of M_D.

q	w	$\delta(q,w)$
C	w_1	A_1
C	w_2	A_4
C	w_3	A_4
A_1	w_1	A_4
A_1	w_2	A_2
A_1	w_3	A_4
A_2	w_1	A_4
A_2	w_2	A_4
A_2	w_3	A_3
A_3	w_1	A_1
A_3	w_2	A_4
A_3	w_3	A_4
A_4	w_1	A_4
A_4	w_2	A_4
A_4	w_3	A_4

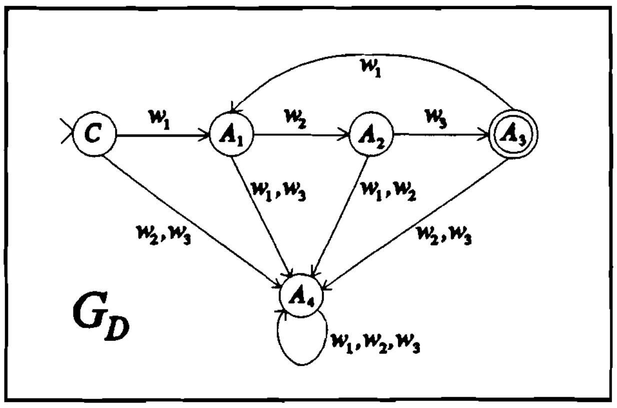


Figure 8.5. State transition diagram of M_D.

★

Definition 8.10. Let (X,d) be a complete metric space and let (X,W,P) be a nondeterministic IFS operating on this space. Define (conform [CUL90,LEW81]) a ***nondeterministic finite affine automaton*** M_N as a 6-tuple (Q,W,P,Δ,s,F), where:

Q is a finite set of states;

W is a finite set of affine transformations (i.e. the alphabet);

P is a finite set of probabilities;

$\Delta : Q \times (W \times P)^* \times Q$ is the transition relation, where $(q_i,(w_r,p_s),q_j)$ is the transition from state $q_i \in Q$ to state $q_j \in Q$ by transformation w_r with probability p_s ;

$s \in Q$ is the initial state (an initialization of the IFS);

$F \subset Q$ is the set of final states (attractors). ♦

Example 8.5. Consider the nondeterministic finite affine automaton $M_N = (Q,W,P,\Delta,s,F)$, where:

$$
\begin{array}{lcl}
Q & = & \{C,A\} \\
W & = & \{w_1,w_2,w_3\} \\
P & = & \{p_1,p_2,p_3\} \\
s & = & C \\
F & = & \{A\} \\
\Delta & = & \{(C,w_1,p_1,A),(C,w_2,p_2,A),(C,w_3,p_3,A),(A,w_1,p_1,A),(A,w_2,p_2,A),(A,w_3,p_3,A)\}
\end{array}
$$

The state transition diagram of M_N is a digraph $G_N = (V,E)$ with $V = Q$ and $E \subset Q \times Q$ (fig. 8.6).

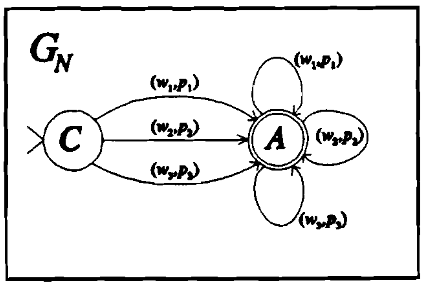


Figure 8.6. State transition diagram of M_N. ★

Theorem 8.3. Let (X,W,P) be an IFS. Then there exists an equivalent finite affine automaton M, which defines the same image (both as compact set and measure) [CUL90].

Proof. The proof follows from the simple observation that a deterministic IFS is just a special case of a determinisitc finite affine automaton and that a nondeterministic IFS is just a special case of a nondeterministic finite affine automaton. ■

Example 8.4 constitutes a deterministic IFS with $W = \{w_1,w_2,w_3\}$ and example 8.5 constitutes a nondeterministic IFS with $W = \{w_1,w_2,w_3\}$ and $P = \{p_1,p_2,p_3\}$.

Definition 8.11. Let M be a finite affine automaton. A string $\sigma \in \Sigma^*$ is ***accepted*** by M if and only if there is a state $q \in F$ such that $(s,\sigma) \vdash_{M^*} (q,e)$, where e denotes the empty string and $\vdash_{M^*}$ is the reflexive, transitive closure of a derivation step $\vdash_M$. The language L accepted by M is denoted by $L(M)$ and consists of the set of all strings accepted by M [LEW81]. ♦

Example 8.6. Consider the deterministic finite affine automaton M_D of example 8.4. If M_D is given the input string $w_1w_2w_3w_1w_2w_3$, its initial configuration is $(C,w_1w_2w_3w_1w_2w_3)$. Then:

$$\begin{aligned}(C,w_1w_2w_3w_1w_2w_3) &\vdash_{M_D} (A_1,w_2w_3w_1w_2w_3)\\ &\vdash_{M_D} (A_2,w_3w_1w_2w_3)\\ &\vdash_{M_D} (A_3,w_1w_2w_3)\\ &\vdash_{M_D} (A_1,w_2w_3)\\ &\vdash_{M_D} (A_2,w_3)\\ &\vdash_{M_D} (A_3,e)\end{aligned}$$

Therefore $(C,w_1w_2w_3w_1w_2w_3) \vdash_{M_D^*} (A_3,e)$ and the string $w_1w_2w_3w_1w_2w_3$ is accepted by M_D. ★

Example 8.7. Consider the nondeterministic finite affine automaton M_N of example 8.5. If M_N is given the input string $(w_2,p_2)(w_3,p_3)(w_1,p_1)$, its initial configuration is $(C,(w_2,p_2)(w_3,p_3)(w_1,p_1))$, and then:

$$(C,(w_2,p_2)(w_3,p_3)(w_1,p_1)) \vdash_{M_N} (A,(w_3,p_3)(w_2,p_2)) \vdash_{M_N} (A,(w_2,p_2)) \vdash_{M_N} (A,e)$$

Therefore $(C,(w_2,p_2)(w_3,p_3)(w_1,p_1)) \vdash_{M_N^*} (A,e)$ and the string $(w_2,p_2)(w_3,p_3)(w_1,p_1)$ is accepted by M_N. ★

God does not play dice.

Albert Einstein (1879 - 1955) [DAR80]

Chapter 9

Invariant measures

In example 8.2 we have seen that not only the affine transformations of an IFS, but also their assigned probabilities influence the structure of the attractor. The mathematical concept to describe the distribution or density of the points within the attractor is that of a measure. Measures can be used to describe intricate distributions of mass on metric spaces. Therefore we need to go through some more definitions.

9.1. Fields

Definition 9.1. Let X be a space and let $\mathbb{S}$ be a nonempty set of subsets of X. Furthermore, let $\mathcal{F}$ be the set of subsets of X which can be built from sets in $\mathbb{S}$ using the operations union, intersection and complementation, with respect to X, such that [BAR88a]:

(i) $A,B \in \mathcal{F} \Rightarrow A \cup B \in \mathcal{F}$;

(ii) $A \in \mathcal{F} \Rightarrow X \backslash A \in \mathcal{F}$.

Then $(\mathcal{F}, \cup, \cap)$ is called a ***field*** generated by $\mathbb{S}$ and denoted $\mathcal{F}$ (according to the De Morgan laws it is not really necessary to use the intersection operation, because $A \cap B = (A \cup B) \backslash (A \backslash B \cup B \backslash A)$). ♦

Notice that the above definition is fundamentally different from the standard algebraic definition of a field. For example, according to the algebraic definition $(\mathcal{F}, \cup, \cap)$ is a field if and only if $(\mathcal{F}, \cup)$ is an Abelian group on X and $(\mathcal{F} \backslash \{\varnothing\}, \cap)$ is a group on X and $(\mathcal{F}, \cup, \cap)$ is distributive on X [DUR85]. We can therefore easily check that $(\mathcal{F}, \cup, \cap)$ is not a field in algebraic terms. The unity elements of $\cup$ and $\cap$ are respectively $\varnothing$ and X, because $A \cup \varnothing = A = \varnothing \cup A$ and $A \cap X = A = X \cap A$. But there are no inverse elements $A^{-1} \subset X$ such that $A \cup A^{-1} = \varnothing = A^{-1} \cup A$ and $A \cap A^{-1} = X = A^{-1} \cap A$. Thus $(\mathcal{F}, \cup)$ and $(\mathcal{F}, \cap)$ are not even groups and $(\mathcal{F}, \cup, \cap)$ is not even a ring.

Example 9.1. Let X be a space and let $A \subset X$. Then $\mathcal{F} = \{ X, X \backslash A, A, \varnothing \}$ is a field. ★

It is obvious that if $\mathcal{F}$ is a field of subsets of a space X, then $X \in \mathcal{F}$ and also $\varnothing \in \mathcal{F}$. Suppose $A \in \mathcal{F}$, then $X \backslash A \in \mathcal{F}$ and hence $A \cup X \backslash A \in \mathcal{F} \Rightarrow X \in \mathcal{F}$. This implies $X \backslash X \in \mathcal{F}$, thus $\varnothing \in \mathcal{F}$.

Example 9.2. Let *Screen* denote the set of pixels corresponding to a certain computer graphics display. The set of all monochrome images which can be produced on this screen forms a field. An example of a field of subsets of *Screen* is shown in fig. 9.1 below. This field is generated by the pair of images A (the Sierpinski triangle in the left row, lower column) and B (the black ellipse in the third row, second column), together with the set *Screen* (the black rectangle). So $\mathbb{S} = \{A,B\}$, but there are more possible generating pairs. The empty set is represented by the white rectangle.

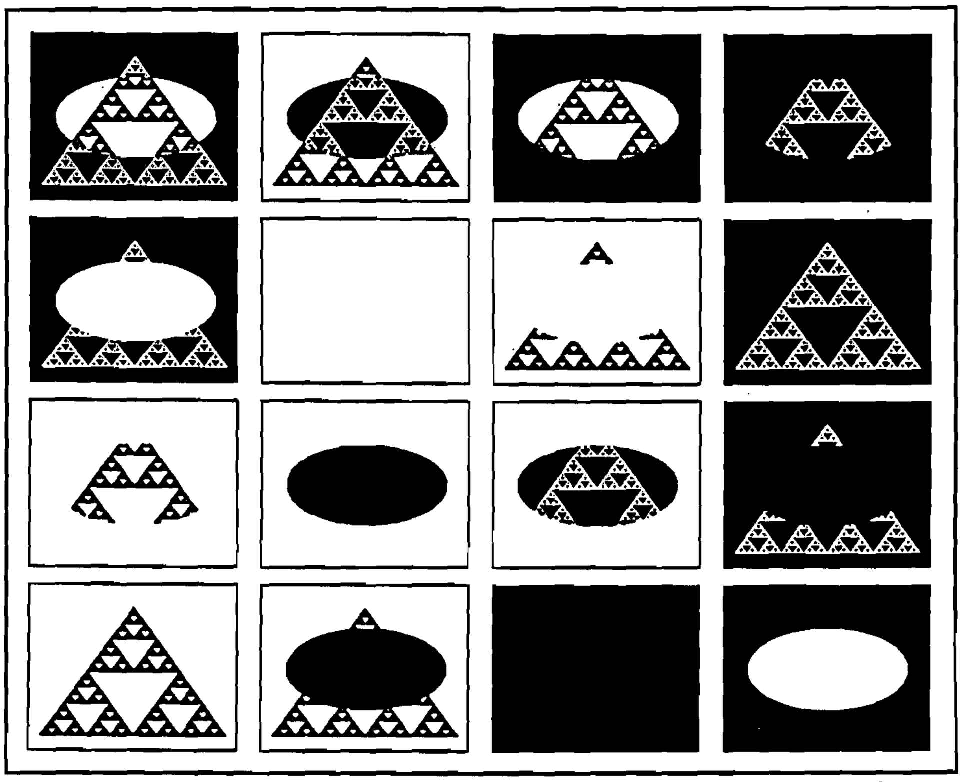


Figure 9.1. The field $\mathcal{F}$ whose elements are sets of pixels of the space *Screen* [BAR88a].

If we enumerate the elements of this field in row first order from upper left to lower right, we get an equivalent representation (where $A\Delta B = A\backslash B \cup B\backslash A = (A\cup B)\backslash(A\cap B)$):

$$\begin{aligned}\mathcal{F} \;=\; &\{\, \mathit{Screen}\backslash(A\Delta B),\ A\Delta B,\ A\cup(\mathit{Screen}\backslash B),\ \mathit{Screen}\backslash(A\cap B),\ \mathit{Screen}\backslash(A\cup B),\ \varnothing,\\ &A\backslash B,\ \mathit{Screen}\backslash A,\ A\cap B,\ B,\ B\backslash A,\ B\cup(\mathit{Screen}\backslash A),\ A,\ A\cup B,\ \mathit{Screen},\ \mathit{Screen}\backslash B \,\}\end{aligned}$$

Notice that $\mathcal{F}$ contains $2^4 = 16$ elements, since $|\{\, \mathit{Screen}, A, B, \varnothing \,\}| = 4$. ★

Definition 9.2. Let $\mathbb{S}$ be a set of subsets of a space X and let $\mathcal{F}$ be a field such that [BAR88a]:

$$(\,\underline{\mathrm{E}}\ i : i = 1,2,3,\ldots : A_i \in \mathcal{F} \Rightarrow \bigcup_{i=1}^{\infty} A_i \in \mathcal{F}\,)$$

then $\mathcal{F}$ is called a **σ-field.** Given any field, there always exists a minimal (or smallest) σ-field, which contains the entire field. The minimal σ-field which contains $\mathbb{S}$ is called the σ-field generated by $\mathbb{S}$ and contains the elements of $\mathbb{S}$, together with X and $\varnothing$. ♦

Example 9.3. Consider example 9.2, where ***Screen*** is the pixelspace on a computer screen. Let $\mathbb{S} = \{A,B\}$, i.e. respectively the black Sierpinski triangle and the black ellipse, and let $\mathcal{F}$ be the field generated by $\mathbb{S}$, hence:

$$\mathcal{F} = \{ Screen\backslash(A\Delta B),\ A\Delta B,\ A\cup(Screen\backslash B),\ Screen\backslash(A\cap B),\ Screen\backslash(A\cup B),\ \varnothing,\ A\backslash B,\ Screen\backslash A,\ A\cap B,\ B,\ B\backslash A,\ B\cup(Screen\backslash A),\ A,\ A\cup B,\ Screen,\ Screen\backslash B \}$$

Let $\mathcal{F}_1$ be the σ-field generated by $\mathbb{S}$ and let $\mathcal{F}_2$ be the σ-field generated by $\mathcal{F}$, then:

$$\mathcal{F}_1 = \mathcal{F}_2 = \{ Screen,\ A,\ B,\ \varnothing \}$$ ★

In general, if the σ-field $\mathcal{F}\sigma$ generated by $\mathbb{S}$ contains k elements, then the field $\mathcal{F}$ generated by $\mathbb{S}$ contains 2^k elements: $|\mathcal{F}| = 2^{|\mathcal{F}\sigma|}$. Thus a field can be considered as the powerset of its σ-field.

Definition 9.3. Let (X,d) be a metric space and let $\mathbb{B}$ denote the σ-field generated by open subsets of X, then $\mathbb{B}$ is called the ***Borel field*** associated with the metric space. An element of $\mathbb{B}$ is called a Borel subset of X and $\mathbb{B}$ is also called the ***Borel set*** [BAR88a]. ♦

In other words, the class of Borel sets is the smallest collection of subsets of X which do have a mass and which have the following properties [FAL90]:

(i) every open set and every closed set is a Borel set;

(ii) the union as well as the intersection and the complementation of every finite or countable collection of Borel sets is a Borel set.

Without proof we state that the associated Borel field $\mathbb{B}$ of a compact metric space (X,d) is generated by a countable set of unit balls (the proof can be found in [BAR88a]).

9.2. Measures

Definition 9.4. A ***measure*** μ on a field $\mathcal{F}$ is a real non-negative function $\mu : \mathcal{F} \rightarrow [0,\infty)$, such that whenever $\mathcal{F}$ is a σ-field with $A_i \cap A_j = \varnothing$ for $i \neq j$ we have [FAL90]:

$$\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i)$$ ♦

A measure is in fact a mass distribution and we can visualize this concept by taking a finite mass of sand and spread it in some way across a set X. Then $\mu(X)$ is a measure or mass distribution on X and $\mu(A)$ is the mass of the set $A \subset X$. For a measure μ certain properties hold. The most important (and also obvious) one is given by: $A \subset X \Rightarrow \mu(X) = \mu(X\backslash A) + \mu(A)$.

Definition 9.5. Let (X,d) be a metric space and let $\mathscr{B}$ denote the Borel subsets of X. Let μ be a measure on $\mathscr{B}$, then μ is called a ***Borel measure*** on X. If $\mu(X)=1$ then μ is said to be ***normalized*** [BAR88a]. ♦

Definition 9.6. Let (X,d) be a metric space, let μ be a Borel measure and let $B(x,\epsilon)$ be the closed unit ball with radius $\epsilon>0$, centered at $x \in X$, i.e. $B(x,\epsilon) = \{ y \in X : d(x,y)<\epsilon \}$. Then the ***support*** of μ is the closed set of points $x \in X$ such that $(\underline{A}\, \epsilon : \epsilon>0 : \mu(B(x,\epsilon))>0)$ or, in other words, the support of a measure is the set on which the measure lives [BAR88a]. ♦

Definition 9.7. Let $(\mathbf{R}^m,d)$ be a complete metric space and let $A = \{ (x_1,\ldots,x_n) \in \mathbf{R}^m : x_i \in [a_i,b_i] \}$ be a coordinate parallelepiped in $\mathbf{R}^m$. Then the m-dimensional volume of A is given by [FAL90]:

$$vol^m(A) = \prod_{i=1}^{m} (b_i - a_i)$$

Define the set S of all coverings of A by countable collections of m-dimensional volumes as [FAL90]:

$$S = \{ \sum_{i=1}^{m} vol^m(A_i) : A \subset \bigcup_{i=1}^{\infty} A_i \}$$

The m-dimensional ***Lebesgue measure*** $\mathscr{L}^m$ of the set A is defined as [FAL90]:

$$\mathscr{L}^m = (\underline{\mathrm{Inf}} : s \in S : s) = (\underline{\mathrm{Max}} : x \in \mathbf{R} \wedge (\underline{A}\, s : s \in S : x \leq s) : x)$$ ♦

So we look at all coverings of A by countable collections of m-dimensional volumes and take the smallest total volume possible.

Example 9.4. Consider the metric space $(\mathbf{R},d_E)$ end let $A \subset \mathbf{R}$ be given by:

$$A = \bigcup_{i=1}^{7} A_i$$

where $A_1 = [0,2]$, $A_2 = [1,5]$, $A_3 = [2,3]$, $A_4 = [3,4]$, $A_5 = [4,8]$, $A_6 = (5,6)$ and $A_7 = [7,10]$. Then there are 10 possible coverings of A, given by:

$$\{ A,\ A\backslash A_2,\ A\backslash A_3,\ A\backslash A_4,\ A\backslash A_6,\ A\backslash\{A_2,A_6\},\ A\backslash\{A_3,A_4\},\ A\backslash\{A_3,A_6\},\ A\backslash\{A_4,A_6\},\ A\backslash\{A_3,A_4,A_6\} \}$$

The according set S of 1-dimensional volumes $vol^1(A)$ for the above set is given by:

$$S = \{16,12,15,15,15,11,14,14,14,13\} \Leftrightarrow \{11,12,13,14,15,16\}$$

Finally we determine the 1-dimensional Lebesgue measure as:

$$\mathscr{L}^1(A) = (\underline{\mathrm{Inf}} : s \in S : s) = 11$$

In fig. 9.2 a graphical representation of the 10 possible coverings is given. From this we can see that the Lebesgue measure is represented by $A\backslash\{A_2,A_6\}$.

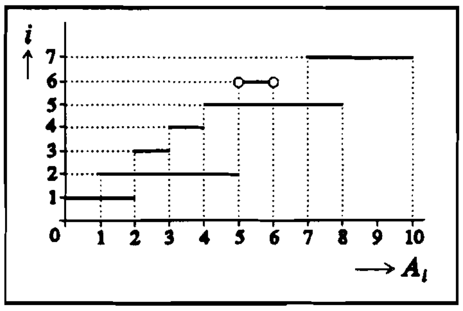


Figure 9.2. The 10 possible coverings of A. ★

Definition 9.8. A function $f: X \to \mathbf{R}$ is called ***simple*** if it can be written in the form [BAR88a]:

$$f(x) = \sum_{i=1}^{N} y_i \chi_{B_i}(x)$$

where $N \in \mathbf{N}$, $B_i \in \mathbb{B}$, $y_i \in \mathbf{R}$, (Union i : $1 \le i \le N$: B_i) = X and $B_i \cap B_j = \emptyset$ for $i \neq j$. The function χ_B is the ***characteristic function*** of the set $B \subset X$ and is defined by [LEW81]:

$$\chi_B(x) = \begin{cases} 1 & \text{for } x \in B \\ 0 & \text{for } x \in X \backslash B \end{cases}$$ ♦

A simple function takes only finitely many values $y_1,\dots,y_N$. In fig. 9.3 the graph of a simple function on a Sierpinski triangle is shown. The domain is a Sierpinski triangle in the x-y-plane. The function values are represented by the z-coordinates. If the function values were represented by colors, a painted Sierpinski triangle would replace the graph of fig. 9.3.

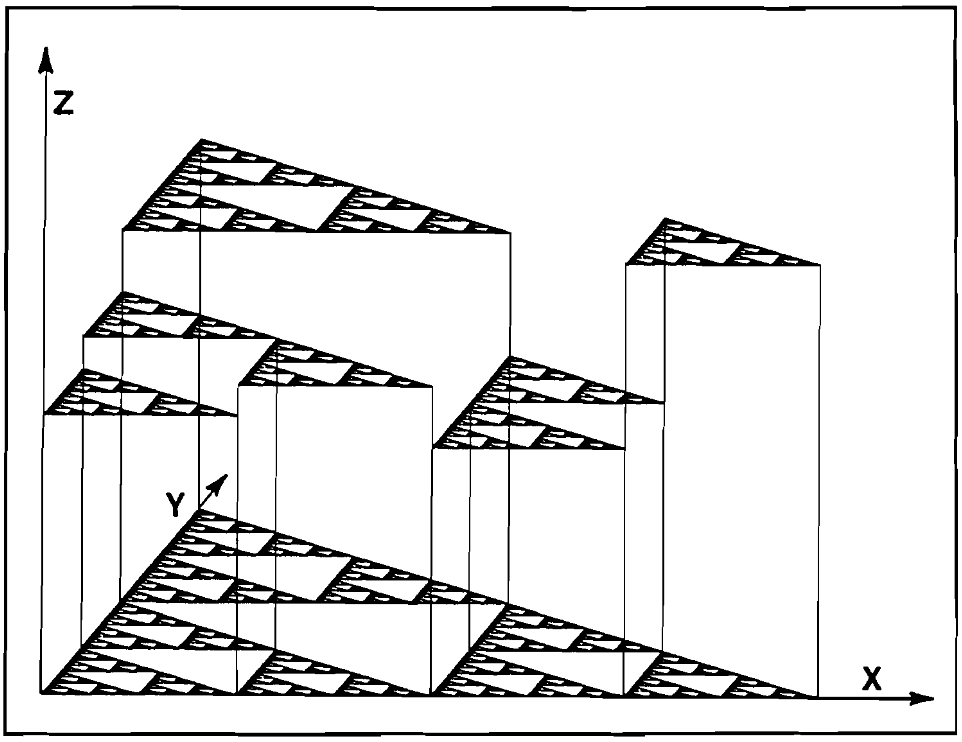


Figure 9.3. The graph of a simple function on a Sierpinski triangle [BAR88a].

Definition 9.9. Let X be a space then we define the ***integral with respect to the measure*** μ of a non-negative simple function f as [FAL90]:

$$\int_X f(x)\,\mathrm{d}\mu(x) = \sum_{i=1}^{N} y_i \mu\{\, x : f(x) = y_i \,\} \qquad \blacklozenge$$

Example 9.5. Let $X = [0.5,12] \in \mathbb{R}$ and let $A \subset X$ be given by:

$$A = \bigcup_{i=1}^{3} [i^2, i^2+2] = [1,3] \cup [4,6] \cup [9,11]$$

The normalized Borel measure μ is defined as:

$$\mu(x) = \frac{4}{x} \quad \text{with} \quad \mu(X) = \int_{0.5}^{12} \frac{4}{x}\mathrm{d}x = 4[\ln|x|]_{0.5}^{12} = 4\ln 12 - 4\ln 0.5 \approx 12.71$$

Next we define the non-negative simple function $f : X \to \mathbb{R}$ as $f(x) = \sqrt{x}$. The mass of the set A is given by:

$$\mu(A) = \mu(\bigcup_{i=1}^{3} A_i) = \sum_{i=1}^{3} \mu(A_i) = \mu([1,3]) + \mu([4,6]) + \mu([9,11])$$

$$= \int_1^3 \frac{4}{x}\mathrm{d}x + \int_4^6 \frac{4}{x}\mathrm{d}x + \int_9^{11} \frac{4}{x}\mathrm{d}x = 4[\ln|x|]_1^3 + 4[\ln|x|]_4^6 + 4[\ln|x|]_9^{11}$$

$$= 4(\ln 3 - \ln 1 + \ln 6 - \ln 4 + \ln 11 - \ln 9) \approx 6.82$$

So the normalized $\mu(A)$ is given by (6.82)/(12.71) $\approx$ 0.54. We can also calculate the integral of f with respect to μ as:

$$\int_A f(x)\,\mathrm{d}\mu(x) = \int_X f(x)\chi_A(x)\,\mathrm{d}\mu(x)$$

$$= \int_X f(x)\chi_A(x)\mu(x)\,\mathrm{d}x$$

$$= \int_{0.5}^{12} \frac{4}{\sqrt{x}}\chi_A(x)\,\mathrm{d}x = \int_1^3 \frac{4}{\sqrt{x}}\mathrm{d}x + \int_4^6 \frac{4}{\sqrt{x}}\mathrm{d}x + \int_9^{11} \frac{4}{\sqrt{x}}\mathrm{d}x$$

$$= [8\sqrt{x}]_1^3 + [8\sqrt{x}]_4^6 + [8\sqrt{x}]_9^{11} \approx 11.98$$

Since $\int_X f(x)\mu(x)\,\mathrm{d}x = \int_{0.5}^{12} \frac{4}{\sqrt{x}}\mathrm{d}x = [8\sqrt{x}]_{0.5}^{12} \approx 22.06$, the normalized value equals $\frac{11.98}{22.06} \approx 0.54$.

In fig. 9.4 the situation is graphically visualized.

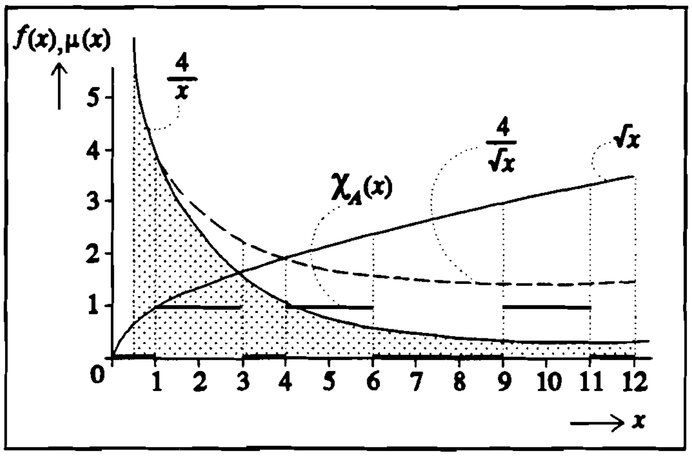


Figure 9.4. The shaded area represents the Borel measure. ★

9.3. Markov operators

Let (X,d) denote a compact metric space and let $\mathcal{B}$ denote the Borel subsets of X. Let $f: X \to X$ be continuous, then it can be proven that $f^{-1} : \mathcal{B} \to \mathcal{B}$ [BAR88a]. It follows that if μ is a normalized Borel measure on X, then so is $\mu \circ f^{-1}$. In turn, this implies that the function defined next, indeed takes $\mathcal{P}(X)$ into itself.

Definition 9.10. Let (X,d) be a compact metric space and let $\mathcal{P}(X)$ denote the space of normalized Borel measures on X. Let $(X,\{ (w_i,p_i) : i = 1,2,\dots,N \})$ be a hyperbolic IFS. The ***Markov operator*** associated with the IFS is the function $M : \mathcal{P}(X) \to \mathcal{P}(X)$ defined by [CRI91,STA91a]:

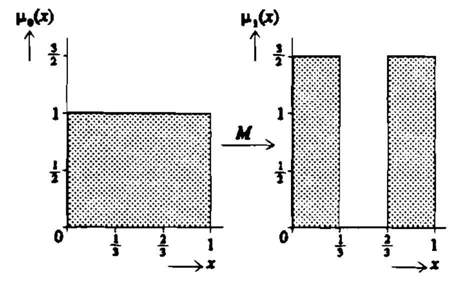
$$M(\mu(x)) = (\underline{A}\, x,\mu : x \in X,\ \mu \in \mathcal{P}(X) : (\underline{S}\, i : 1 \le i \le N : p_i\,\mu(w_i^{-1}(x))\,)\,) \qquad \blacklozenge$$

Example 9.6. Consider the IFS $([0,1],\{⅓x, ⅓x + ⅔\},\{½,½\})$. The attractor of the IFS is the Cantor set. Let M denote the associated Markov operator and let $\mu_0 \in \mathcal{P}([0,1])$ be the uniform normalized measure on $[0,1]$, i.e. $\mu_0([0,1]) = 1$. We look at the sequence of measures $\{ \mu_n = M^{(n)}(\mu_0) \}$, where $M^{(n)}(\mu(x)) = p_1\mu(w_1^{-1}(x)) + p_2\mu(w_2^{-1}(x))$. With $w_1^{-1}(x) = 3x$ and $w_2^{-1}(x) = 3x-2$ we get:

$$M(\mu_0([0,⅓])) = ½\mu_0(w_1^{-1}[0,⅓]) + ½\mu_0(w_2^{-1}[0,⅓]) = ½\mu_0([0,1]) + ½\mu_0([-2,-1]) = ½ + 0 = ½$$

$$M(\mu_0([⅓,⅔])) = ½\mu_0(w_1^{-1}[⅓,⅔]) + ½\mu_0(w_2^{-1}[⅓,⅔]) = ½\mu_0([1,2]) + ½\mu_0([-1,0]) = 0 + 0 = 0$$

$$M(\mu_0([⅔,1])) = ½\mu_0(w_1^{-1}[⅔,1]) + ½\mu_0(w_2^{-1}[⅔,1]) = ½\mu_0([2,3]) + ½\mu_0([0,1]) = 0 + ½ = ½$$



So after one iteration step the total mass of $[0,1]$ is spread out equally over $[0,⅓] \cup [⅔,1]$ and both subintervals contain $½\mu_0$. After n iterations all mass is concentrated in $(⅔)^n$ part of the original space $[0,1]$. The relative density of the mass in that part is $(⅔)^{-n}$ times the original density, but the total mass in the space remains equal (see figure left). ★

Theorem 9.1. Let (X,d) be a compact metric space and let $\mathcal{P}(X)$ denote the space of normalized Borel measures on X. Let M denote the Markov operator associated with the hyperbolic IFS with condensation (X,W_C,P_C) Let μ be the unique invariant measure of this IFS, where μ_0 denotes the condensation measure of (X,W_0,P_0). Furthermore, let $f : X \to \mathbb{R}$ be either a simple function or a continuous function, then [BAR85,BAR88a,FRE90]:

$$\int_X f(x)\,\mathrm{d}(M(\mu(x))) = \sum_{i=1}^{N} p_i \int_X f \circ w_i(x)\mathrm{d}\mu(x) + p_0 \int_X f(x)\mathrm{d}\mu_0(x)$$

If $p_0 = 0$ then we simply get an IFS (X,W,P) without condensation.

Proof. See [BAR88a]. ■

Definition 9.11. Let (X,d) be a compact metric space. Let $\mathcal{P}(X)$ denote the set of normalized Borel measures on X. Then the ***Hutchinson metric*** (or ***Hutchinson distance***) d_H is defined by [CRI91,HUT81]:

$$d_H(\mu,\nu) = (\,\underline{\mathrm{Sup}} : \mu,\nu \in \mathcal{P}(X), f : X \to \mathbb{R},\ (\,\underline{\mathrm{A}}\, x,y : x,y \in X : |f(x) - f(y)| \le d(x,y)) : \int_X f(x)\mathrm{d}\mu(x) - \int_X f(x)\mathrm{d}\nu(x)\,)$$

The supremum is taken over all continuous real-valued functions satisfying the stated Lipschitz condition. Without proof we state that $(\mathcal{P}(X),d_H)$ is a compact metric space. The proof can be found in [HUT81]. ♦

Theorem 9.2. Let (X,d) be a compact metric space. Let (X,W,P) be a hyperbolic IFS with contractivity factor $s \in [0,1)$ and associated Markov operator $M : \mathcal{P}(X) \to \mathcal{P}(X)$. Then M is a contraction mapping with contractivity factor s, with respect to the Hutchinson metric on $\mathcal{P}(X)$, that is [BAR88a,STA91a]:

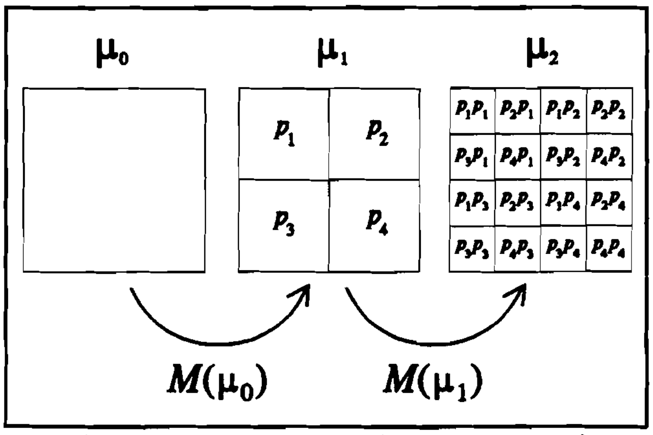
$$(\,\underline{\mathrm{A}} : \mu,\nu \in \mathcal{P}(X) : d_H(M(\mu),M(\nu)) \le s d_H(\mu,\nu)\,)$$

In particular, there is a unique measure $\mu_f \in \mathcal{P}(X)$ such that $M(\mu_f) = \mu_f$. This unique measure μ_f is the fixed point of the Markov operator and is called the ***invariant (probability) measure*** of the IFS. Moreover, the support of μ_f is the attractor of the IFS.

Proof. See [BAR88a]. ■

Definition 9.12. Let μ be the invariant measure of the hyperbolic IFS with condensation (X,W_C,P_C). If $p_0 > 0$ then μ is a ***p-balanced measure*** for the IFS (X,W_C,P_C). If $p_0 = 0$ then μ is a p-balanced measure for the IFS (X,W,P) [DEM85]. ♦

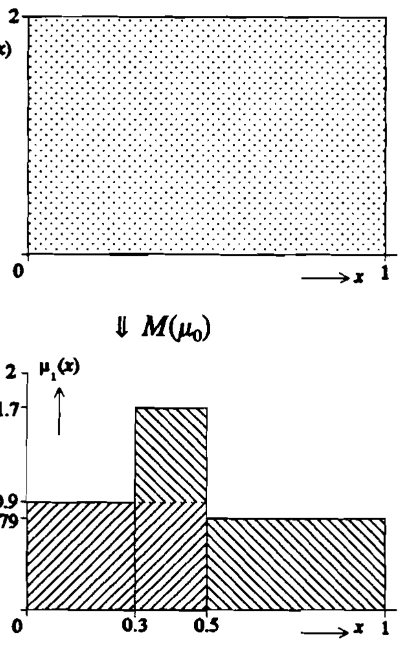
So, if an IFS (X,W,P) is initialized with a measure μ_0 then we get a sequence of measures $\{\ \mu_n = M^{(n)}(\mu_0)\ \}$. The measure $\mu_1 = M(\mu_0)$ is such that $\mu(w_i(X)) = p_i$ for $i = 1,2,\ldots,N$. In the same way the measure $\mu_2 = M^{(2)}(\mu_0)$ is such that $\mu(w_{i1} \circ w_{i2}(X)) = p_{i1}p_{i2}$ for $i = 1,2,\ldots,N$. Thus, in general the measure $\mu_n = M^{(n)}(\mu_0)$ is such that $\mu(w_{i1}\circ w_{i2}\circ\ldots\circ w_{in}(X)) = p_{i1}p_{i2}\ldots p_{in}$ (see fig. 9.5). So when the Markov operator is applied, the mass in a cell $A \subset X$ is redistributed among N cells $w_1(A), w_2(A), \ldots, w_n(A)$. Also, mass from other cells is mapped into subcells of A in such way that the total mass of A remains the same as before the Markov operator was applied. In this manner the distribution of mass is defined on finer and finer scales as the Markov operator is repeatedly applied.



μ_0
μ_1
μ_2
p_1 p_2 p_3 p_4
p_1p_1 p_2p_1 p_1p_2 p_2p_2
p_3p_1 p_4p_1 p_3p_2 p_4p_2
p_1p_3 p_2p_3 p_1p_4 p_2p_4
p_3p_3 p_4p_3 p_3p_4 p_4p_4
$M(\mu_0)$
$M(\mu_1)$

Figure 9.5. An IFS $([0,1]\times[0,1]\subset\mathbb{R},\{(w_i,p_i),\ i = 1,2,3,4\})$ with associated Markov operator at work.

Example 9.7. Consider the IFS $([0,1],\{0.5x, 0.7x+0.3\},\{0.45, 0.55\})$. The attractor of the IFS is $[0,1]$. Let M denote the associated Markov operator and let $\mu_0 \in \mathcal{P}([0,1])$ be the uniform normalized measure on $[0,1]$, so $\mu_0([0,1])=1$. We look at the sequence of measures $\{\ \mu_n = M^{(n)}(\mu_0)\ \}$ where $M(\mu(x)) = p_1\mu(w_1^{-1}(x)) + p_2\mu(w_2^{-1}(x))$. The successive iterates $M(\mu_0)$ and $M^{(2)}(\mu_0)$ are represented in the figure below. Each measure is represented by a collection of rectangles whose bases are contained in the interval $[0,1]$. The area of each rectangle equals the measure of its base. For each iteration the result of $w_1^{-1}(x) = 2x$ is shaded /// and the result of $w_2^{-1}(x) = (x - 0.3)/0.7$ is shaded \\\\\\.



$\mu_0([0,1]) = 1$

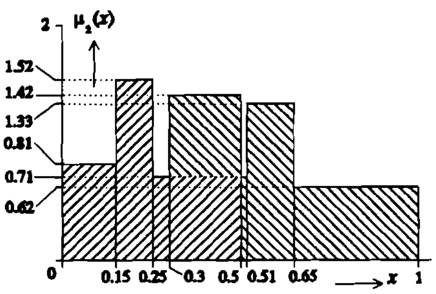
$$\begin{aligned}M(\mu_0([0.3,0.5])) &= 0.45\mu_0 w_1^{-1}([0.3,0.5]) + 0.55\mu_0 w_2^{-1}([0.3,0.5])\\ &= 0.45\mu_0([0.6,1]) + 0.55\mu_0([0,0.29])\\ &= 0.45\cdot 0.4 + 0.55\cdot 0.29\\ &= 0.34\end{aligned}$$

So $\mu_1(x) = 0.34/0.2 = 1.70$ for $x \in [0.3,0.5)$, thus alltogether:

$$\mu_1 = \begin{cases} 0.90 & \text{for } x \in [0,0.3)\\ 1.70 & \text{for } x \in [0.3,0.5)\\ 0.79 & \text{for } x \in [0.5,1]\end{cases}$$

$$\begin{aligned}M^{(2)}(\mu_0([0.3,0.5])) &= 0.45\mu_1 w_1^{-1}([0.3,0.5]) + 0.55\mu_1 w_2^{-1}([0.3,0.5])\\ &= 0.45\mu_1([0.6,1]) + 0.55\mu_1([0,0.29])\\ &= 0.45(0.45\mu_0 w_1^{-1}([0.6,1]) + 0.55(\mu_0 w_2^{-1}([0.6,1])))\\ &\quad + 0.55(0.45\mu_0 w_1^{-1}([0,0.29]) + 0.55(\mu_0 w_2^{-1}([0,0.29])))\\ &= 0.45(0.45\mu_0([1.2,2]) + 0.55\mu_0([0.43,1]))\\ &\quad + 0.55(0.45\mu_0([0,0.58]) + 0.55\mu_0([0.43,1]))\end{aligned}$$

$\Downarrow M^{(2)}(\mu_1)$



$=0.45(0.45\cdot 0+0.55\cdot 0.57)+0.55(0.45\cdot 0.58+0.55\cdot 0)$
$=0.28$
So $\mu_2(x) = 0.28/0.2 = 1.42$ for $x \in [0.3,0.5)$, thus alltogether:

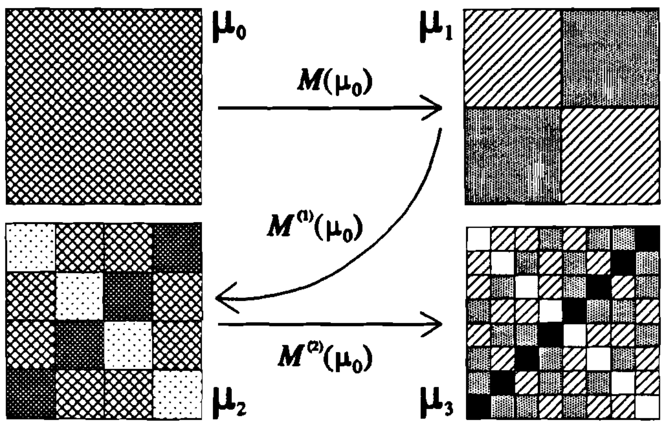
$$\mu_2 = \begin{cases} 0.81 & \text{for } x \in [0,0.15) \\ 1.52 & \text{for } x \in [0.15,0.25) \\ 0.71 & \text{for } x \in [0.25,0.3) \\ 1.42 & \text{for } x \in [0.3,0.5) \\ 0.71 & \text{for } x \in [0.5,0.51) \\ 1.33 & \text{for } x \in [0.51,0.65) \\ 0.62 & \text{for } x \in [0.65,1] \end{cases}$$

Although the sequence of measures converges to $\{ M^{(n)}(\mu_0) \}$ in the metric space $(\mathcal{P}([0,1]),d_H)$, some of the rectangles would become infinitely tall as n tends to infinity. ★

<u>*Example 9.8.*</u> Consider the IFS $([0,1]\times[0,1],\{w_1,w_2,w_3,w_4\},\{p_1,p_2,p_3,p_4\})$ with:

$w_1(x) = (0.5x_1, 0.5x_2+0.5)$, $p_1 = 0.2$
$w_2(x) = (0.5x_1+0.5, 0.5x_2+0.5)$, $p_2 = 0.3$
$w_3(x) = (0.5x_1, 0.5x_2)$, $p_3 = 0.3$
$w_4(x) = (0.5x_1+0.5, 0.5x_2)$, $p_4 = 0.2$

The attractor of the IFS is $[0,1]\times[0,1]$. Let M denote the associated Markov operator. Let $\mu_0 \in \mathcal{P}([0,1]\times[0,1])$ be a uniform measure on $[0,1]\times[0,1]$, i.e. a rectangle with a certain grey value, say 1000 [*pixels*]. We look at the sequence of measures $\{ \mu_n = M^{(n)}(\mu_0) \}$, where $M(\mu(x)) = p_1\mu(w_1^{-1}(x)) + p_2\mu(w_2^{-1}(x)) + p_3\mu(w_3^{-1}(x))$. With $\mu(w^{-1}(x))$ we actually mean that the mass is multiplied by the inverse determinant of the linear transformation matrix A of the map w. The successive iterates $M(\mu_0)$, $M^{(2)}(\mu_0)$ and $M^{(3)}(\mu_0)$ are represented below.



Each measure is represented by a collection of squares, whose areas are contained in $[0,1]\times[0,1]$. The grey value of a square equals the measure of the area of that square, i.e. the more mass lays on a square, the more pixels the square contains and the darker it appears. Notice that the total number of pixels (or average grey value) is constant. Suppose the original square contains 1000 [*pixels*]. Each time the Markov operator $M(\mu_0)$ is applied, the amount of subsquares is multiplied by the amount of affine transformations N=4. According to the latter figure, we can calculate the successive amounts of pixels each subsquare contains as:

			8,12,12,18,12,18,18,27
			12,8,18,12,18,12,27,18
		40,60,60,90	12,18,8,12,18,27,12,18
1000	200,300	60,40,90,60	18,12,12,8,27,18,18,12
	300,200	60,90,40,60	12,18,18,27,8,12,12,18
		90,60,60,40	18,12,27,18,12,8,18,12
			18,27,12,18,12,18,8,12
			27,18,18,12,18,12,12,8

As we can easily check, the total mass (amount of pixels) stays equal, i.e. 1000. ★

__Theorem 9.3.__ Let (X,d) be a compact metric space, let (X,W,P) be a hyperbolic IFS with unique invariant measure μ and let $f : X \to \mathbf{R}$ be a bounded continuous function. If we initialize the IFS with a point $x_0 \in X$ and choose the maps w_i recursively and independently according to their assigned probabilities p_i then [BAR88e,BAR89,ELT87]:

$$\lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n} f(w_{ik} \circ w_{i(k-1)} \dots \circ w_{i1}(x_0)) = \int_X f(x)\,\mathrm{d}\mu(x)$$

__Proof.__ See [ELT87]. ■

In other words, starting at any $x_0 \in X$, the empirical distribution of an orbit $\{ x_n : n = 1,2,\dots,\infty \}$ converges with probability one to the invariant measure μ. This means that each time we apply nondeterministic IFS-decoding without condensation (see chapter 8), we can not say what orbit will be followed, but with probability one, each time the same attractor will appear. A direct result from the above theorem is stated in the next definition.

__Definition 9.13.__ Let (X,d) be a compact metric space and let (X,W,P) be a hyperbolic IFS with unique invariant measure μ. Let B be a Borel subset of X and let $\mu(\partial B) = 0$. Let $N(B,n)$ be the number of points in the orbit $\{ x_n : n = 1,2,\dots,\infty \}$ which land precisely in B, then [BAR88a]:

$$N(B,n) = (\underline{N}\, n : 1 \leq n : x_n \in B)$$

and for all starting points $x_0 \in X$ the mass of B is given by:

$$\mu(B) = \lim_{n\to\infty}\frac{N(B,n)}{n}$$

Thus, the mass of B is the proportion of iteration steps which produce points in B. ♦

9.4. Moment theory

Definition 9.14. Let (X,d) be a compact metric space, where X has the dimension q and let (X,W,P) be a hyperbolic IFS with unique invariant measure μ. The ***moments*** of $\mu \in \mathcal{P}(X)$ for points $x \in X$ and $k_i \in \mathbf{N}$ is defined as [BEC64,HOR88,LAY90,RED89]:

$$m(\mu,k_i : i = 1..q) = \int_X \prod_{i=1}^{q} x_i^{k_i} \mathrm{d}\mu(x)$$

If $k_1 = k_2 = \dots = k_q = k$ then with $m(\mu,k)$ we refer to the k^{th} moment of μ, which we denote m_k. In that case the sequence $\{ m_k : k = 0,1,2,\dots \}$ is called the ***Hausdorff moment sequence*** of μ. ♦

Theorem 9.4. Let (X,d) be a compact metric space and let (X,W_C,P_C) be a hyperbolic IFS with condensation and with unique invariant p-balanced measure μ and condensation measure μ_0. Then the moments $\{ m(\mu,k) : k = 0,1,2\dots \}$ can be calculated uniquely explicitly recursively, in terms of the parameters which define the IFS and the moments $\{ m(\mu_0,k) : k = 0,1,2,\dots \}$ [BAR84,BAR85].

Proof. Let μ_0 be any probability measure whose support is C and suppose X is a one-dimensional space, i.e. q=1, then:

$$m(\mu,k)$$
$$= \langle \text{ definition 9.14 } \rangle$$
$$\int_X x^k \mathrm{d}\mu(x)$$
$$= \langle\, X \text{ is bounded, } k \neq 0, \text{ substitute } x^k := f \,\rangle$$
$$\int_X f(x)\,\mathrm{d}\mu(x)$$
$$= \langle \text{ theorem 9.1, theorem 9.2 } \rangle$$
$$\sum_{i=1}^{N} p_i \int_X f \circ w_i(x)\,\mathrm{d}\mu(x) + p_0 \int_X f(x)\,\mathrm{d}\mu_0(x)$$
$$= \langle\, k \neq 0, \text{ substitute } f := x^k \,\rangle$$
$$\sum_{i=1}^{N} p_i \int_X x^k \circ w_i(x)\,\mathrm{d}\mu(x) + p_0 \int_X x^k \mathrm{d}\mu_0(x)$$
$$= \langle \text{ definition 9.14}, f(x) = x \Rightarrow f \circ g(x) = g(x) \,\rangle$$
$$\sum_{i=1}^{N} p_i \int_X (w_i(x))^k \mathrm{d}\mu(x) + p_0 m(\mu_0,k)$$
$$= \langle \text{ suppose for brevity that } p_0 = 0 \text{ (no condensation), } x = z \text{ and } w_i = s_i z + b_i \text{ with } b_i, s_i, z \in \mathbf{C} \,\rangle$$
$$\sum_{i=1}^{N} p_i \int_X (s_i z + b_i)^k \mathrm{d}\mu(z)$$
$$= \langle \text{ calculus } \rangle$$
$$\sum_{i=1}^{N} p_i \int_X \sum_{j=0}^{k} \binom{k}{j} (s_i z)^j b_i^{k-j} \mathrm{d}\mu(z)$$
$$= \langle \text{ calculus } \rangle$$

$$\sum_{i=1}^{N} p_i \sum_{j=0}^{k} \binom{k}{j} s_i^j b_i^{k-j} \int_X z^j \mathrm{d}\mu(z)$$

$= \langle$ split off $j = k$ $\rangle$

$$\sum_{i=1}^{N} p_i \left[\sum_{j=1}^{k-1} \binom{k}{j} s_i^j b_i^{k-j} \int_X z^j \mathrm{d}\mu(z) + \binom{k}{k} s_i^k b_i^{k-k} \int_X z^k \mathrm{d}\mu(z) \right]$$

$= \langle$ calculus $\rangle$

$$\sum_{i=1}^{N} \sum_{j=0}^{k-1} \binom{k}{j} s_i^j b_i^{k-j} p_i \boldsymbol{m}(\mu,j) + \boldsymbol{m}(\mu,k) \sum_{i=1}^{N} p_i s_i^k$$

$= \langle$ calculus $\rangle$

$$\left[1 - \sum_{i=1}^{N} p_i s_i^k \right]^{-1} \sum_{i=1}^{N} \sum_{j=0}^{k-1} \binom{k}{j} s_i^j b_i^{k-j} p_i \boldsymbol{m}(\mu,j)$$

$\Rightarrow \langle$ $\boldsymbol{m}(\mu,0) = \int_X \mathrm{d}\mu(z) = 0$ $\rangle$

$\boldsymbol{m}(\mu,k)$ can be calculated uniquely explicitly recursively

We can prove the same for an affine transformation $w(x,y) = (ax + by + e,\ cx + dy + f)$:

$\boldsymbol{m}(\mu,k_1,k_2)$

$= \langle$ definition 9.14, theorem 9.1, theorem 9.2, conform first 6 steps of former part $\rangle$

$$\sum_{i=1}^{N} p_i \int_X (a_i x + b_i y + e_i)^{k_1} (c_i x + d_i y + f_i)^{k_2} \mathrm{d}\mu(x,y)$$

$= \langle$ assume $k_1 = k_2 = k$ $\rangle$

$$\sum_{i=1}^{N} p_i \int_X ((a_i x + b_i y + e_i)(c_i x + d_i y + f_i))^k \mathrm{d}\mu(x,y)$$

$= \langle$ calculus $\rangle$

$$\sum_{i=1}^{N} p_i \int_X (a_i c_i x^2 + a_i d_i xy + a_i f_i x + b_i c_i xy + b_i d_i y^2 + b_i f_i y + c_i e_i x + d_i e_i y + e_i f_i)^k \mathrm{d}\mu(x,y)$$

$= \langle$ multinomium $\rangle$

$$\sum_{i=1}^{N} p_i \int_X \sum_{j_1+\ldots+j_9=k} \binom{j}{j_1\, j_2 \ldots j_9} (a_i c_i x^2)^{j_1} (a_i d_i xy)^{j_2} (a_i f_i x)^{j_3} (b_i c_i xy)^{j_4} (b_i d_i y^2)^{j_5} (b_i f_i y)^{j_6} (c_i e_i x)^{j_7} (d_i e_i y)^{j_8} (e_i f_i)^{j_9} \mathrm{d}\mu(x,y)$$

$= \langle$ calculus $\rangle$

$$\sum_{i=1}^{N} p_i \sum_{j_1+\ldots+j_9=k} \binom{k}{j_1\, j_2 \ldots j_9} a_i^{j_1+j_2+j_3} b_i^{j_4+j_5+j_6} c_i^{j_1+j_4+j_7} d_i^{j_2+j_5+j_8} e_i^{j_7+j_8+j_9} f_i^{j_3+j_6+j_9} \int_X x^{2j_1+j_2+j_3+j_4+j_7} y^{j_2+j_4+2j_5+j_6+j_8} \mathrm{d}\mu(x,y)$$

$= \langle$ calculus $\rangle$

$$\sum_{i=1}^{N} p_i \sum_{j_1+\ldots+j_9=k} \frac{j!}{j_1! j_2! \cdots j_9!} a_i^{j_1+j_2+j_3} b_i^{j_4+j_5+j_6} c_i^{j_1+j_4+j_7} d_i^{j_2+j_5+j_8} e_i^{j_7+j_8+j_9} f_i^{j_3+j_6+j_9} M(\mu, 2j_1+j_2+j_3+j_4+j_7, j_2+j_4+2j_5+j_6+j_8)$$

$\Rightarrow \langle$ $\boldsymbol{m}(\mu,0,0) = \int_X \mathrm{d}\mu(x,y) = 0$ $\rangle$

$\boldsymbol{m}(\mu,k_1,k_2)$ can be calculated uniquely explicitly recursively ■

__*Example 9.9.*__ Consider the IFS $(\mathbb{R}^2, W, P)$ where W and P are given by:

$$w_1(x_1,x_2) = (\tfrac12 x_1, \tfrac12 x_2) \quad w_2(x_1,x_2) = (\tfrac12 x_1 + \tfrac12, \tfrac12 x_2) \quad w_3(x_1,x_2) = (\tfrac12 x_1, \tfrac12 x_2 + \tfrac12)$$

We can calculate the moments as follows:

$$\begin{aligned}
&\boldsymbol{m}(k_1,k_2)\\
&= \int_X x_1^{k_1} x_2^{k_2}\,\mathrm{d}\mu(x_1,x_2)\\
&= \sum_{i=1}^{3} p_i \int_X (w_i(x_1))^{k_1}(w_i(x_2))^{k_2}\,\mathrm{d}\mu(x_1,x_2)\\
&= \frac13\int_X (\tfrac12 x_1)^{k_1}(\tfrac12 x_2)^{k_2}\mathrm{d}\mu(x_1,x_2) + \frac13\int_X (\tfrac12 x_1+\tfrac12)^{k_1}(\tfrac12 x_2)^{k_2}\mathrm{d}\mu(x_1,x_2) + \frac13\int_X (\tfrac12 x_1)^{k_1}(\tfrac12 x_2+\tfrac12)^{k_2}\mathrm{d}\mu(x_1,x_2)\\
&= \left(1-\frac13(\tfrac12)^{k_1+k_2}\right)^{-1}\frac13(\tfrac12)^{k_1+k_2}\left[\sum_{j=0}^{k_1}\binom{k_1}{j}M(j,k_2) + \sum_{r=0}^{k_2}\binom{k_2}{r}M(k_1,r)\right]\\
&= (3\cdot 2^{k_1+k_2}-1)^{-1}\left[\sum_{j=0}^{k_1}\binom{k_1}{j}M(j,k_2) + \sum_{r=0}^{k_2}\binom{k_2}{r}M(k_1,r)\right]
\end{aligned}$$

Since $\boldsymbol{m}(0,0) = 0$, we can compute the moments uniquely explicitly recursively. ★

Duration is the indefinite continuation of existence.

Benedict de Spinoza (1632 - 1677) [ADL77]

Chapter 10

IFS-encoding: the inverse problem

In chapter 8 we were concerned with IFS-decoding. Here we address the inverse problem, i.e. given a set $A \in \mathcal{H}(X)$ (or a set $B \in \mathcal{P}(X)$), determine an IFS whose attractor is approximately A (or B). It is obvious that this inverse problem is less trivial than IFS-decoding.

10.1. The collage theorem

__*Theorem 10.1*__ (***collage theorem***). Let (X,d) be a complete metric space and let $A \in \mathcal{H}(X)$ and $\epsilon \geq 0$ be given. Choose a deterministic hyperbolic IFS (X,W,P) with or without condensation and with contractivity factor $s \in [0,1)$, such that:

$$h(A, \bigcup_{i\in\{0,1\}}^{N} w_i(A)) \leq \epsilon$$

where h denotes the Hausdorff metric, then:

$$h(A,A_f) \leq \frac{\epsilon}{1-s}$$

where A_f is the attractor of the IFS [BAR86a]. Equivalently:

$$(\underline{\mathrm{A}}\, A : A \in \mathcal{H}(X) : h(A,A_f) \leq (1-s)^{-1}h(A, \bigcup_{i\in\{0,1\}}^{N} w_i(A))$$

__*Proof.*__ Let $w(A) = \bigcup_{i\in\{0,1\}}^{N} w_i(A)$, then:

$h(A,A_f)$

$\leq \langle$ definition 4.9 $\rangle$

$\lim_{n\to\infty} \sum_{i=1}^{n} h(w^{(i-1)}(A),w^{(i)}(A))$

$\leq \langle$ calculus $\rangle$

$h(A,w(A)) \lim_{n\to\infty} \sum_{i=1}^{n} s^{i-1}$

$\leq \langle$ $|s| < 1$, sum for an infinite geometric series $\rangle$

$(1-s)^{-1}h(A,w(A))$ ■

The collage theorem tells us that in order to find an IFS whose attractor is "close to" or "looks like" a given set $A \in \mathcal{H}(X)$, one must find a set of transformations W such that the union or ***collage*** of the transformations $\{ w \in W : w^{(1)}(A) \}$ is near to the given set A. In other words, we perform a "lazy tiling" of A by affine transformed copies of itself [GIP90b]. The less overlap between the separate tiles exists, the more efficient the IFS-encoding. Nearness is measured using the Hausdorff metric. In fig. 10.1 the effects of a "good" and a "bad" collage are shown. The ideal case is to find a set of affine transformations W such that the transformations $\{ w \in W : w(A) \}$ form a partition on A, thus:

$$\bigcup_{i=1}^{N} w_i(A) = A \;\wedge\; (\underline{A}\, i,j : 1 \leq i,j \leq N,\, i \neq j : w_i(A) \cap w_j(A) = \emptyset\,)$$

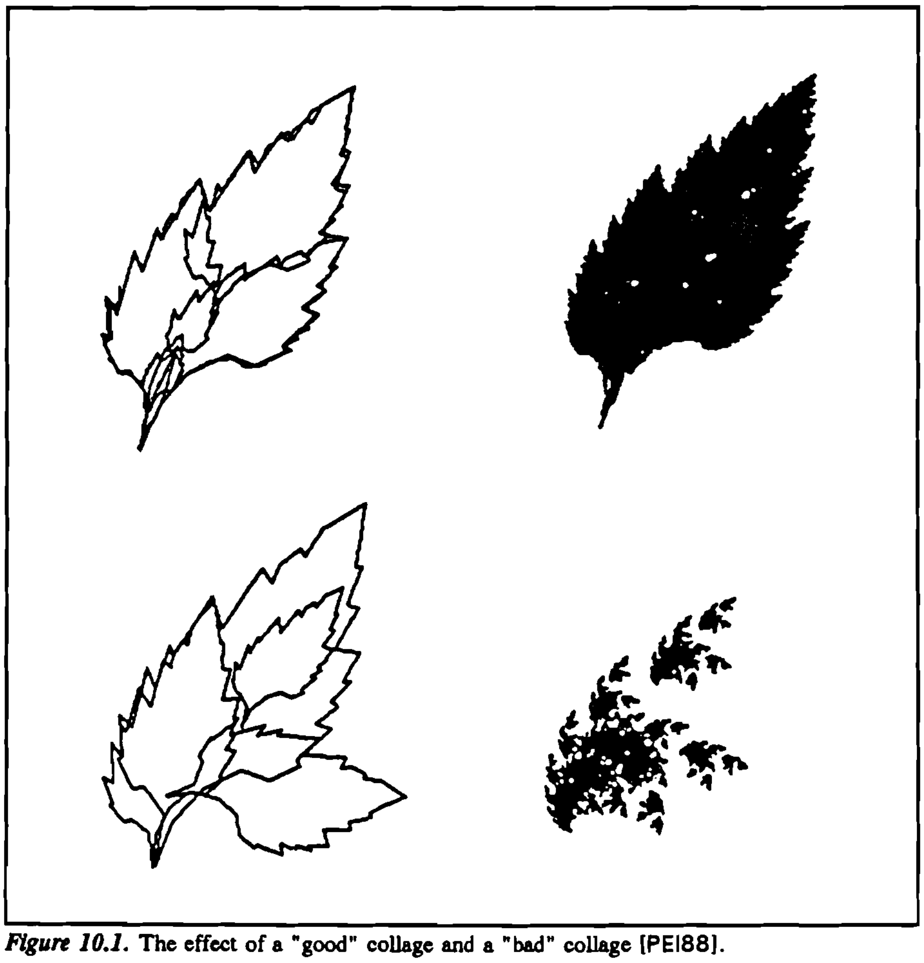


Figure 10.1. The effect of a "good" collage and a "bad" collage [PEI88].

Example 10.1. Suppose we are using a trial-and-error procedure to adjust the coefficients in two affine transformations $w_1(x) = ax + b$ and $w_2(x) = cx + d$, where $a,b,c,d \in \mathbf{R}$, to search for a deterministic hyperbolic IFS $(\mathbf{R},\{w_1,w_2\},P)$, with attractor $A_f = [0,1]$. Of course, since the IFS is deterministic, we disregard the probabilities P. We might come up with $w_1(x) = 0.51x - 0.01$ and $w_2(x) = 0.47x + 0.53$. We can calculate the quality of this approximation by:

$$\begin{aligned}
& h(A,\bigcup_{i=1}^{N} w_i(A)) \\
&= h([0,1],\bigcup_{i=1}^{2} w_i([0,1])) \\
&= h([0,1],[-0.01,0.5]\cup[0.53,1]) \\
&= (\ \underline{\mathrm{Max}} :: (d([0,1],[-0.01,0.5]\cup[0.53,1]),d([-0.01,0.5]\cup[0.53,1],[0,1]))\ ) \\
&= (\ \underline{\mathrm{Max}} :: ((\ \underline{\mathrm{Max}} :: x \in [0,1] : d(x,[-0.01,0.5]\cup[0.53,1])), \\
&\qquad (\ \underline{\mathrm{Max}} :: x \in [-0.01,0.5]\cup[0.53,1] : d(x,[0,1])))\ ) \\
&= (\ \underline{\mathrm{Max}} :: (d(0.515,[-0.01,0.5]\cup[0.53,1]),d(-0.01,[0,1]))\ ) \\
&= (\ \underline{\mathrm{Max}} :: (0.015,0.01)\ ) \\
&= 0.015
\end{aligned}$$

The contractivity factor for the IFS is (according to definition 8.1) given by:

$$s = (\ \underline{\mathrm{Max}} :: (s_1,s_2)\ ) = (\ \underline{\mathrm{Max}} :: (0.51,0.47)\ ) = 0.51$$

So by theorem 10.1 :

$$h([0,1],A_f) \le (0.015)(1-s)^{-1} = (0.015)/(1-0.51) \approx 0.031 \qquad \bigstar$$

Suppose that the collage exists of N affine transformations, then to use nondeterministic IFS-decoding it is necessary to assign a probability to each transformation. To assure that each transformation is randomly selected with a chance directly proportional to the information it contributes to the attractor, its probability must be set equal to the relative determinant value of its transformation matrix A, with respect to the sum of the determinant values of the transformation matrices of all transformations in the IFS [KOC89a]. In other words:

$$p_i = \frac{|\det(A_i)|}{\sum_{i=1}^{N} |\det(A_i)|} = \frac{|a_i d_i - b_i c_i|}{\sum_{i=1}^{N} |a_i d_i - b_i c_i|} \quad \wedge \quad \sum_{i=1}^{N} p_i = 1$$

If some p_i appears to be zero, we should assign a small value to it. In the IFS-CODEC algorithm (see appendix B) we assign the value 0.001 to p_i in such cases.

Theorem 10.2. Let (X,d) be a compact metric space and let (X,W,P) be a nondeterministic hyperbolic IFS with or without condensation and with contractivity factor $s \in [0,1)$. Let $\nu = M(\mu)$ be the associated invariant measure and let $M : \mathcal{P}(X) \to \mathcal{P}(X)$ be the associated Markov operator and let $\mu \in \mathcal{P}(X)$. Then [BAR88a]:

$$d_H(\mu,\nu) \le \frac{d_H(\mu,M(\mu))}{1-s}$$

where d_H denotes the Hutchinson metric.

Proof. This is a corollary of theorem 10.1. ■

Theorem 10.1 together with theorem 10.2 tells us that to find an IFS whose invariant measure is "close to" or "looks like" a given set $B \in \mathcal{P}(X)$, one must find a set of transformations W such that the support of its invariant measure is almost equal to B. Nearness is measured by the Hutchinson metric. Thus concluding:

(i) For IFS-encoding of subsets of the complete metric space $(\mathcal{H}(X),h)$, i.e. monochrome images, we should cover the target image $A \in \mathcal{H}(X)$ with N affine transformed copies of itself. The quality of the collage is measured by the Hausdorff metric h. In particular, for given $\epsilon \geq 0$ and $A,A_f \in \mathcal{H}(X)$:

$$h(A,\bigcup_{i=1}^{N} w_i(A)) \leq \epsilon \Rightarrow h(A,A_f) \leq \frac{\epsilon}{1-s}$$

where A_f is the attractor and s is the contractivity factor of the encoded IFS.

(ii) For IFS-encoding of subsets of the complete metric space $(\mathcal{P}(X),d_H)$, i.e. color or greytone images, we should cover the target image $B \in \mathcal{P}(X)$ with N affine transformed copies of itself and adjust the probabilities. The quality of the collage is measured by the Hutchinson metric d_H. In particular, for given $\epsilon \geq 0$ and $\mu,\mu_f \in \mathcal{P}(X)$:

$$d_H(\mu,M(\mu)) \leq \epsilon \Rightarrow d_H(\mu,\mu_f) \leq \frac{\epsilon}{1-s}$$

where μ_f is the invariant measure (whose support is the attractor) and s is the contractivity factor of the encoded IFS.

We mention that a collage can be formalized by a ***collage grammar*** [HAB91], but we will not discuss this possibility here. Furthermore, notice that the inverse problem has no unique solution, there are many valid collages.

10.2. Computational time complexity

If we want to solve the IFS-encoding problem by computing and evaluating all possible combinations of affine transformations and their probabilities, we will not succeed in polynomial time, since the IFS-encoding problem appears to be intractable.

We will restrict ourselves to the case of monochrome images, which implies that finding a collage can be done by tiling of the target image with affine transformed copies of itself. So, we only prove the case for IFS-encoding of subsets of the complete metric space $(\mathcal{H}(X),h)$ (see former paragraph).

The proof will not succeed if we use the Hausdorff metric h to measure the approximation quality, since the Hausdorff metric is not powerful enough to quantify the proposition: $A \cap B = \varnothing$ (see later on). Therefore we will use the Pixset metric d_P to measure the approximation quality. The reader should not be concerned about the details of d_P until par. 10.4, where we will introduce this metric formally. A quick glance at definition 10.1 should do for the moment.

Theorem 10.3. The IFS-encoding problem is NP-complete.

Proof. For brevity we omit the probabilities. Within this proof we formally define the IFS-encoding problem as follows (conform [GAR79]):

NAME:	IFS-ENCODING.
INSTANCE:	A finite set $A \in \mathcal{H}(X)$, a positive integer $N \leq \lvert A \rvert$ and a real-valued number $\epsilon \in [0,1)$.
QUESTION:	Is there a set $W = \{w_1,\dots,w_N\}$ of affine transformations such that:

$$d_p(A, \bigcup_{i=1}^{N} w_i(A)) \leq \epsilon \qquad \text{(strong constraint)}$$

$$(\underline{A}\, i,j : 1 \leq i,j \leq N,\ i \neq j : h(w_i(A), w_j(A)) \geq 1-\epsilon\,) \qquad \text{(weak constraint)}$$

The proof exists of two parts conform [COR90,GAR79,LEW81]:

(i) To be proven: IFS-ENCODING $\in$ NP.

Let T be a nondeterministic Turing machine which generates candidate solutions W for IFS-ENCODING. A checking algorithm which checks whether W is a valid solution must basically do three things. First, the probabilities for each $w_i \in W$ must be calculated from the determinants of the affine transformations, which can be done in $O(N)$ time, where N denotes the amount of affine transformations in W. Second, the attractor A_f of the thus formed IFS (X, W, P) must be rendered, which can be done in $O(n)$ time, where n denotes the amount of iterations (see par. 8.3). Third, the approximation quality of the rendered attractor A_f as to the instantiated set A must be calculated, which can be done in $O(\lvert A \rvert^2)$ time if we use the Hausdorff metric and in $O(\lvert A \rvert)$ time if we use the Pixset metric (see par. 10.4), where $\lvert A \rvert$ denotes the cardinality of the instantiated set A, i.e. the amount of pixels A contains. For each candidate solution W generated by the nondeterministic Turing machine T, the checking algorithm can determine in a time that is a polynomial function of the input length $\lvert A \rvert + N + n$, whether this candidate solution is valid or not. In other words, IFS-ENCODING can be solved Nondeterministic Polynomially and thus belongs to the class NP, which is defined to be the class of all problems that can be solved by nondeterministic algorithms that run in polynomial time [GAR79].

(ii) The second part of the proof consists of showing that some suitably choosen known NP-complete problem (in this case SET PACKING) can be considered as a special case of IFS-ENCODING. We must therefore prove that each instance of SET PACKING can be transformed in polynomial time into an instance of IFS-ENCODING. In other words, we must prove that there exists a polynomial transformation f from SET PACKING to IFS-ENCODING, which we denote with SET PACKING $\propto$ IFS-ENCODING.

To be proven: SET PACKING $\propto$ IFS-ENCODING.

The SET PACKING problem is defined as follows [GAR79]. Does an instantiated collection C of finite sets contain at least an instantiated number of $1 \leq N \leq |C|$ mutually disjoint sets?

We define a transformation f that transforms each instance $c_i \in C$ from SET PACKING into an instance $w_i \in W$ from IFS-ENCODING. First, the transformation f sorts the elements of each c_i. We denote the successive elements of c_i with $c_i(1),\ldots,c_i(|c_i|)$, where $c_i(1)$ contains the smallest element. Then the transformation f assigns to each element of C a pixel in a monochrome image A which consists of a horizontal line of adjacent pixels and which resides in some pixel space, thus $|C| = |A|$. Each pixel is initially marked "unassigned". Furthermore, elements that appear in more than one c_i are stored in a register.

It is clear that f must transform each instance $c_i \in C$ into an instance $f(c_i) = w_i(A)$, and it does so by applying the following (naive, but effective) algorithm:

```
for (position := 1 to |A|) pixel(position) := "unassigned";
register := ∅;
for ( i := 1 to N )
  {
    count := 0;
    for ( j := 1 to |$c_i$| )
      {
        position := 1;
        while (pixel(position) ≠ "unassigned")
          {
            if (pixel(position) = $c_i$(j))
              {
                register := register ∪ $c_i$(j);
                count := count + 1;
              }
            position := position + 1;
          }
      }
    a := d := |$c_i$|/|C|;
    e := position - count;
    b := c := f := 0;
    $w_i$ := {a,b,c,d,e,f};
    W := W ∪ $w_i$;
    for (j := 1 to |$c_i$|) pixel(e+j) := $c_i$(j);
  }
```

Only if *register* $= \varnothing \wedge (\underline{N}\, i : 1 \leq i \leq |A| : pixel(i) = \text{"unassigned"}) = 0$ is true, the thus obtained collage is valid, since this means that the transformed images $w_i(A)$ form a partition on A. To avoid problems with the implicit discretization which is induced by applying a transformation w_i we let $a,b,c,d \in \mathbb{Q}$. Now we must prove two things:

a) To be proven: $f(n)$ is $O(p(n))$ for some polynomial function p. Clearly, f can be computed in polynomial time $O(\underline{N}\, i,j : 1 \leq i \leq N, j \in c_1 \cup \ldots \cup c_{i-1} \cap c_i : j)$, since f first assigns to each element of C a pixel in A and then applies the above algorithm, which contains $O(|C| + |\{c_i : 1 \leq i \leq N\}|)$ steps. Because f assigns to each element of c_i a pixel in $w_i(A)$, it is clear that $f(c_i) = w_i(A)$ and $|c_i| = |w_i(A)|$.

b) To be proven: ($\underline{\mathrm{A}}\, c_i : c_i \in C : c_i \in$ SET PACKING $\Leftrightarrow f(c_i) \in$ IFS-ENCODING) or in other words: (C contains N mutually disjoint sets) $\Leftrightarrow$ ($|W| = N$). Let $\epsilon = 0$.

"$\Rightarrow$": Let C contain N mutually disjoint sets, then:

$\bigcup_{i=1}^{N} c_i = C \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : c_i \cap c_j = \varnothing)$

$\Rightarrow \langle$ apply $f \rangle$

$\bigcup_{i=1}^{N} w_i(A) = A \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : w_i(A) \cap w_j(A) = \varnothing)$

$\Rightarrow \langle$ definition 10.1, $A \cap B = \varnothing \Rightarrow d_p(A,B) = 1 \rangle$

$d_p(A, \bigcup_{i=1}^{N} w_i(A)) = 0 \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : d_p(w_i(A), w_j(A)) = 1)$

$\Rightarrow \langle \epsilon = 0 \rangle$

$d_p(A, \bigcup_{i=1}^{N} w_i(A)) \le \epsilon \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : d_p(w_i(A), w_j(A)) \ge 1-\epsilon)$

"$\Leftarrow$": Let $|W| = N$, then:

$d_p(A, \bigcup_{i=1}^{N} w_i(A)) \le \epsilon \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : d_p(w_i(A), w_j(A)) \ge 1-\epsilon)$

$\Rightarrow \langle$ let $\epsilon = 0 \rangle$

$d_p(A, \bigcup_{i=1}^{N} w_i(A)) = 0 \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : d_p(w_i(A), w_j(A)) = 1)$

$\Rightarrow \langle$ definition 10.1, $d_p(A,B) = 1 \Rightarrow A \cap B = \varnothing \rangle$

$\bigcup_{i=1}^{N} w_i(A) = A \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : w_i(A) \cap w_j(A) = \varnothing)$

$\Rightarrow \langle$ apply $f^{-1} \rangle$

$\bigcup_{i=1}^{N} c_i = C \ \wedge \ (\underline{\mathrm{A}}\, i,j : 1 \le i,j \le N, i \ne j : c_i \cap c_j = \varnothing)$

Thus we see that SET PACKING can be considered as a special case of IFS-ENCODING, namely for $\epsilon = 0$. Remember that $\epsilon \in [0,1)$ is also instantiated together with the set A. Since SET PACKING is known to be NP-complete, IFS-ENCODING must also be NP-complete. ■

It is remarkable that we have not been able to find an NP-completeness proof for the IFS-encoding problem in the available literature on the subject. Even the most recent fundamental book on the field ([PEI92]) doesn't contain such proof, although it states that IFS-encoding is probably computationally complex. It is strange that despite the lack of such NP-completeness proof in literature, all known attempts to implement automatic IFS-encoding so far have been heuristic ones.

Since we have proven IFS-ENCODING to be NP-complete, we know that any combinatorial search algorithm will fail to generate the (optimal) solution within reasonable time. Even if we bound the parameter values of the affine transformations $w_i \in W$ to certain discrete values, the problem still remains NP-complete.

Let (X,W,P) be an IFS and assume that the parameters of the according IFS-code are bounded by $a,b,c,d,e,f \in \{ \pm x/n : x = 1,2,\dots,n \}$. As said before, we omit the probabilities. Then the amount of possible affine transformations is given by the amount of possible combinations (a,b,c,d,e,f). This can be calculated as $(2n)^6 = 64n^6$. An IFS-code consists of a collage of N different affine transformations. The amount of possible combinations of N affine transformations is equal to the amount of possible combinations of N affine transformations from a language of $64n^6$ affine transformations, i.e. $(64n^6)^N$. A deterministic algorithm which generates all possible IFS-codes will therefore have running time $T(n,N) = 64n^{6N}$. Clearly, IFS-encoding is polynomial in n, but exponential in N. This is illustrated in fig. 10.2 for N=2 (left) and n=2 (right).

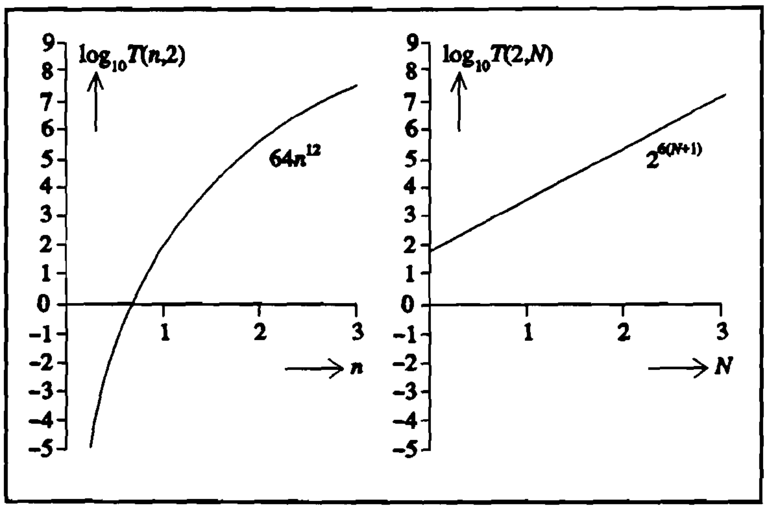


Figure 10.2. Time complexity function $T(n,N)$ for N=2 (left) and n=2 (right).

The growth rate of the polynomial time complexity $T(n,2) = 64n^{12}$ is less than the growth rate of the exponential time complexity $T(2,N) = 2^{6(N+1)}$. In the special case $n = N$, the polynomial time complexity $T(n,2)$ is only worse then the exponential time complexity $T(2,N)$ for values in the interval (2,4), since:

$$64n^{12} = \begin{cases} < 2^{6(N+1)} & \text{for } n=N \in [0,2] \\ \geq 2^{6(N+1)} & \text{for } n=N \in (2,4) \\ < 2^{6(N+1)} & \text{for } n=N \in [4,\infty) \end{cases}$$

Realistic values for n and N are given by $n = 100$ and $4 \leq N \leq 200$ [BAR88c,BAR90]. We see that even for the smallest realistic value of N the amount of possible IFS-codes equals the amount of atoms contained in the whole universe, which latter amount is roughly estimated to be 10^{50}. This can be shown as follows:

$$T(100,4) = 64 \cdot 100^{6*4} \approx 10^{1.8}(10^2)^{24} = 10^{49.8} \approx 10^{50}$$

The state space for the average IFS-encoding problem consists of an amount of elements somewhere in between $10^{49.8}$ and $10^{2401.8}$. This means that a (super) computer with a calculation speed of 200 [*GFlops*] would at least be kept busy for $5 \cdot 10^{-12} \cdot 10^{49.8} = 5 \cdot 10^{37.8}$ [*seconds*] or roughly 10^{29} [*centuries*] to apply a single floating point operation on each possible collage. If we take into account the probabilities, things are even worse, since these probabilities introduce another factor $n = 100$. In that case we get a time complexity function $T(100,N) = 64(100)^{7N} = 10^{14N+1.8}$. If the algorithm has to check the quality of each generated IFS-code, the running time will increase with a factor $|A|$, i.e.

the amount of pixels of the target image, and we get $T(|A|,100,N) = |A| \cdot 10^{14N+1.8}$. The trivial cases $|A| = 1$ [*pixel*] and $N = |A|$ [*affine transformations*] can of course be solved in linear time, but the problem is that there exists no algorithm which can solve the IFS-encoding problem for all $N \leq |A|$ in polynomial time. The same considerations are valid for the ϵ-values. If we want a collage for $\epsilon = 0.99$ then the problem can be solved very quickly, but the problem is that we want an algorithm which can generate solutions for all ϵ-values $\in [0,1)$, thus also for $\epsilon = 0$. The above considerations force us to develop an algorithm that is not based on elementary state space search, but rather on some other approach.

10.3. A first approach: moment theory

In par. 9.4 we have seen that it is possible to calculate the moments of an IFS uniquely, explicitly and recursively, in terms of the moments and the parameters which define the IFS (see theorem 9.4).

Theorem 10.4. Under the conditions of theorem 9.4 the associated moment problem is determinate. In particular, the corresponding p-balanced measure μ is unique.

Proof. See [BAR85]. ■

This means that, given the first k moments of a target image, which can be calculated in $O(kn)$ time, finding the invariant measure μ which defines the image, has a unique solution.

Example 10.2. Consider the fractal structure below, known as the ***twin dragon*** fractal [MAN83], and which we denote A.

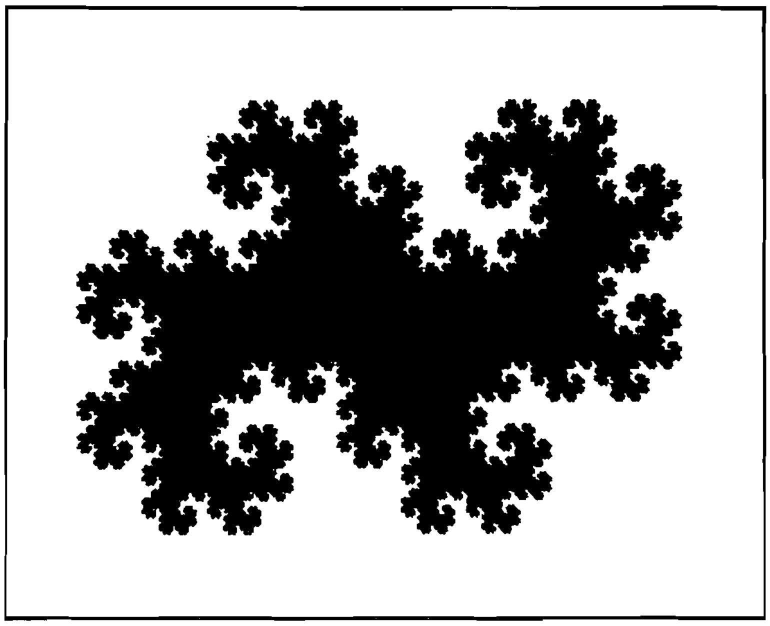


Figure 10.3. The twin dragon fractal.

Suppose this is the target image which we want to encode into an IFS-code. As A is symmetric about its centre, which we represent as the origin in the complex z-plane, we approximate μ by a p-balanced measure $\mu(s,a)$ for the IFS $(\mathbf{C},\{w_1(z),w_2(z)\},\{0.5,0.5\})$, with $w_1(z) = sz+(1-s)a$ and $w_2(z) = sz-(1-s)a$. Referring to par. 7.4 it is clear that $w_1(z)$ has fixed point $a \in \mathbf{C}$ and $w_2(z)$ has fixed point $-a \in \mathbf{C}$. The IFS has a contractivity factor $s \in \mathbf{C}$ such that $|s| < 1$. First we calculate the two-parameter family of moments:

$$\boldsymbol{m}(\mu,k) = \int_{\mathbf{C}} z^k \mathrm{d}\mu(z)$$

Since A is symmetric about the origin, we know that $\boldsymbol{m}(\mu,1) = \boldsymbol{m}(\mu,3) = 0$. So we only have to calculate $\boldsymbol{m}(\mu,1)$ and $\boldsymbol{m}(\mu,4)$.

$\boldsymbol{m}(\mu,2)$

= ⟨ definition 9.14 ⟩

$\int_{\mathbf{C}} z^2 \mathrm{d}\mu(z)$

= ⟨ theorem 9.1, theorem 9.2 ⟩

$\sum_{i=1}^{2} p_i \int_{\mathbf{C}} (w_i(z))^2 \mathrm{d}\mu(z)$

= ⟨ calculus ⟩

$\frac{1}{2}\int_{\mathbf{C}} [(w_1(z))^2+(w_2(z))^2]\,\mathrm{d}\mu(z)$

= ⟨ calculus ⟩

$\frac{1}{2}\int_{\mathbf{C}} [(sz+(1-s)a)^2+(sz-(1-s)a)^2]\,\mathrm{d}\mu(z)$

= ⟨ calculus ⟩

$\frac{1}{2}\int_{\mathbf{C}} (2s^2z^2+2(1-s)^2a^2)\mathrm{d}\mu(z)$

= ⟨ calculus ⟩

$s^2\int_{\mathbf{C}} z^2\mathrm{d}\mu(z) + (1-s)^2a^2\int_{\mathbf{C}} z^0\mathrm{d}\mu(z)$

= ⟨ definition 9.14 ⟩

$s^2\boldsymbol{m}(\mu,2) + (1-\mathrm{s})^2a^2\boldsymbol{m}(\mu,0)$

= ⟨ $\boldsymbol{m}(\mu,0) = 1$, calculus ⟩

$\frac{(1-s)^2a^2}{1-s^2}$

= ⟨ calculus ⟩

$\frac{(1-s)a^2}{1+s}$

$\boldsymbol{m}(\mu,4)$

= ⟨ definition 9.14 ⟩

$\int_{\mathbf{C}} z^4\mathrm{d}\mu(z)$

= ⟨ theorem 9.1, theorem 9.2 ⟩

$$\sum_{i=1}^{2} p_i \int_C (w_i(z))^4 \mathrm{d}\mu(z)$$
$= \langle$ calculus $\rangle$
$$\frac{1}{2}\int_C [(w_1(z))^4+(w_2(z))^4]\,\mathrm{d}\mu(z)$$
$= \langle$ calculus $\rangle$
$$\frac{1}{2}\int_C (2s^4z^4+12s^2z^2(1-s)^2a^2+2(1-s)^4a^4)\mathrm{d}\mu(z)$$
$= \langle$ calculus $\rangle$
$$s^4m(\mu,4) + 6s^2(1-s)^2a^2m(\mu,2) + (1-s)^4a^4m(\mu,0)$$
$= \langle\, m(\mu,0) = 0$, calculus $\rangle$
$$\frac{6s^2(1-s)^2a^2m(\mu,2) + (1-s)^4a^4}{1-s^4}$$
$= \langle$ substitute $m(\mu,2)$ $\rangle$
$$\frac{\dfrac{6s^2(1-s)^3a^4}{1+s} + (1-s)^4a^4}{(1-s)(1+s)(1+s^2)}$$
$= \langle$ calculus $\rangle$
$$\frac{6s^2(1-s)^2a^4 + (1+s)(1-s)^3a^4}{(1+s)^2(1+s^2)}$$
$= \langle$ calculus $\rangle$
$$\frac{(1-s)^2a^4[6s^2+(1+s)(1-s)]}{(1+s)^2(1+s^2)}$$
$= \langle$ substitute $m(\mu,2)$, calculus $\rangle$
$$\frac{(1+5s^2)}{(1+s^2)}(m(\mu,2))^2$$

Now we solve the latter equation for s^2:

$$(1+s^2)m(\mu,4) = (1+5s^2)(m(\mu,2))^2$$
$\Rightarrow \langle$ calculus $\rangle$
$$s^2(m(\mu,4) - 5(m(\mu,2))^2) = (m(\mu,2))^2 - m(\mu,4)$$
$\Rightarrow \langle$ calculus $\rangle$
$$s^2 = \frac{(m(\mu,2))^2 - m(\mu,4)}{m(\mu,4) - 5(m(\mu,2))^2}$$

To each pixel in A we attach a weight of $|A|^{-1}$, where $|A|$ denotes the cardinality or total amount of pixels in A. With the IFS-CODEC algorithm (see appendix A and B) We obtain $|A| = 89587$ [*pixels*].

With the IFS-CODEC algorithm, we compute the Hausdorff moment sequence $\{m_0,m_1,m_2,m_3,m_4\}$ (see definition 9.14), which results in:

$$
\begin{aligned}
m_0 &= 1+i\\
m_1 &= 0+0\mathrm{i}\\
m_2 &= 8.063819237\cdot 10^3+4.160132117\cdot 10^3 i\\
m_3 &= 0+0i\\
m_4 &= -2.027113949\cdot 10^7+1.968592598\cdot 10^8 i
\end{aligned}
$$

We next substitute the computed values m_2 and m_4 for respectively $m(\mu,2)$ and $m(\mu,4)$:

$$s^2 = \frac{(8.063819237\cdot 10^3 + 4.160132117\cdot 10^3 i)^2 - (-2.027113949\cdot 10^7 + 1.968592598\cdot 10^8 i)}{(-2.027113949\cdot 10^7 + 1.968592598\cdot 10^8 i) - 5(8.063819237\cdot 10^3 + 4.160132117\cdot 10^3 i)^2}$$

$$\approx 0.004481 + 0.498892i$$

This yields two roots: $s_1 \approx 0.502 + 0.497i$ and $s_2 \approx -0.502 - 0.497i$. There correspond four possible solution pairs (s,a), however, each IFS that is assigned to a solution pair yields exactly the same measure and approximate fractal attractor. The target image was rendered from the IFS $(\mathbb{R}^2,\{w_1(x),w_2(x)\},\{0.5,0.5\})$, with

$$
\begin{aligned}
w_1(x) &= (0.5x_1 - 0.5x_2 - 100,\ 0.5x_1 + 0.5x_2)\\
w_2(x) &= (0.5x_1 - 0.5x_2 + 100,\ 0.5x_1 + 0.5x_2)
\end{aligned}
$$

Since the the affine transformations are similitudes, this IFS can be transformed in an IFS $(\mathbf{C},\{w_1(z),w_2(z)\},\{0.5,0.5\})$, with transformations:

$$
\begin{aligned}
w_1(z) &= (0.5 + 0.5i)z - 100 = sz + t \quad \text{with fixed point } -100 + 0i\\
w_2(z) &= (0.5 + 0.5i)z + 100 = sz + t \quad \text{with fixed point } 100 + 0i
\end{aligned}
$$

The image in fig. 10.3 was rendered with $s = 0.5 + 0.5i$. From the moment calculation we obtained as one of the roots: $s_1 \approx 0.502 + 0.497i$. If we want to use this value to render the according attractor, we first go to $\mathbb{R}^2$ and obtain the IFS: $(\mathbb{R}^2,\{w_1(x),w_2(x)\},\{0.5,0.5\})$, with:

$$
\begin{aligned}
w_1(x) &= (0.502x_1 - 0.502x_2 - 100.498,\ 0.497x_1 + 0.497x_2 - 0.373)\\
w_2(x) &= (0.502x_1 - 0.502x_2 + 100.498,\ 0.497x_1 + 0.497x_2 + 0.373)
\end{aligned}
$$

For this transformation from $\mathbf{C}$ to $\mathbb{R}^2$ we used:

$$a^2 = \frac{(1+s)m(\mu,2)}{(1-s)} \qquad t = (1-s)a \approx 100.498 - 0.373 \qquad e = \mathrm{Re}(t) \qquad f = \mathrm{Im}(t)$$

$$\mathrm{Re}(s) = \pm\tfrac{1}{2}\sqrt{2}\,\sqrt{0.0004481 + \sqrt{(0.004481)^2 + (0.498892)^2}} \approx 0.502$$

$$\mathrm{Im}(s) = \frac{0.498892}{\mathrm{Re}(s)} \approx 0.497$$

The attractor of this IFS is given in fig. 10.4 and as we can see, it is a good approximate of the target image in fig. 10.3. The pixsetdistance between the two structures was calculated as $d_p = 0.053713$, which means an approximation quality of about 95%. ★

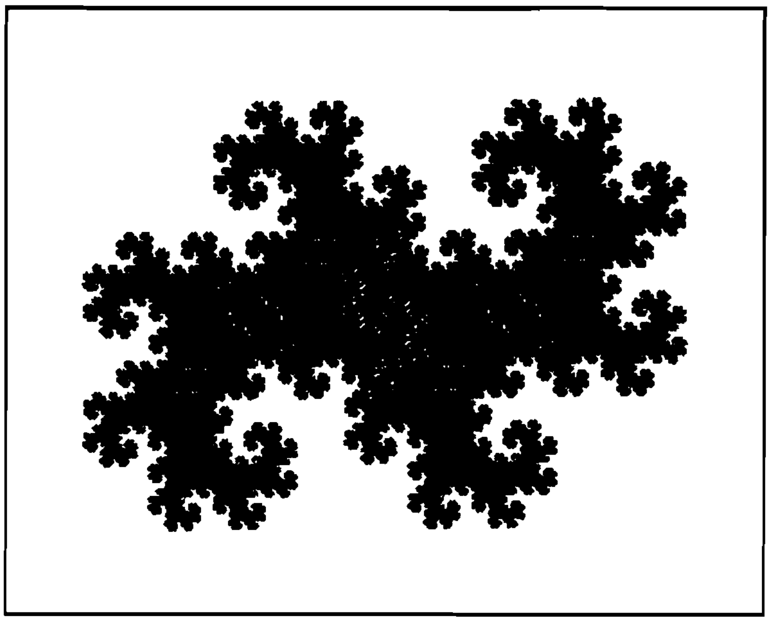


Figure 10.4. Approximation of fig. 10.3 calculated from moment theory.

From the above considerations, we could conclude that theorem 9.4 supplies us with a powerful tool to obtain IFS-encoding in closed form. But this method only works under certain hypotheses:

(i) The invariant measure μ should be uniform upon A, which is only true when all affine transformations $w_i(A)$ of the IFS are mutually disjoint.

(ii) Except information about the moments of A, useful application of theorem 9.4 also expects information about the symmetry of A, which requires certain pattern recognition techniques. But pattern recognition is a severe unsolved problem which we do not intend to solve first before solving the IFS-encoding problem. Moreover, we not only have to know how many transformations we need, but also their associated probabilities.

(iii) The structure in fig. 10.3 constitutes the attractor of an IFS of the simple form $(\mathbb{C}, w(z) = s_i z \pm (1-s_i)a_i, p_i)$, which only can generate similitudes of A, as shown in par. 7.4). But to encode real world images, the operations scaling, rotation and translation alone are not powerful enough. Therefore we also need to be able to shear the image, which implies that we have to use a more powerful affine IFS of the form $(\mathbb{R}^2, w_i(x) = (ax+by+e, bx+cy+f), p_i)$.

The above considerations definitely force us to develop an algorithm that is based on a heuristic approach. This means that we must first be concerned about a quick and reliable method for evaluating the heuristic generated candidate solutions.

10.4. The quality measure

If we want to measure the quality of an approximation A_f of a given monochrome image A in a pixel space, we should use the Hausdorff distance h. The Hausdorff distance is calculated by first finding the minimum of all distances between a point $x \in A$ and a point $y \in A_f$. Denote the two points where between the minimum distance exists as x_{min} and y_{min}. Then the maximum of all distances between a point $x \in A$ and $y_{min} \in A_f$ must be found. Denote the found point $x \in A$ as x_{max}. In the same way the maximum of all distances between a point $y \in A_f$ and $x_{min} \in A$ must be found and the thus obtained point $y \in A_f$ is denoted y_{max}. Finally, the Hausdorff distance is the maximum of the distance between x_{min} and y_{max} and the distance between x_{max} and y_{min} (see also par. 2.2).

Assume that A and A_f both contain n points, then the computational time complexity of an algorithm to compute $h(A,A_f)$ is clearly $T(n^2+2n+1)$ or $O(n^2)$. Since the quality measure will become part of the evaluation function of the IFS-encoding algorithm, and the evaluation time is a critical factor, the Hausdorff distance is insufficient because of its computational time complexity.

The Hausdorff distance can give strange results, which do not match with our intuitive notion of closeness. For example, if A is a disk and B is the same disk as A, but with an outside connected thin line segment with length equal to the diameter of B and perpendicular to its boundary, then $h(A,B)$ equals the length of the line segment. The structures A and B appear visually very close, but the Hausdorff distance doesn't imply this at all.

Moreover, the Hausdorff distance is insufficient to model the proposition $A \cap B = \varnothing$ which we will need hereafter. Therefore we need a quality measure that obeys the following criteria:

(i) it must match with our intuitive notion of closeness;

(ii) it must be able to model the proposition $A \cap B = \varnothing$;

(iii) it must have low computational complexity, i.e. less than $O(n^2)$;

(iv) it must be a metric on $\mathcal{H}(X)$.

In [KOC89b] the following measure is proposed:

$$rdiff(A,A_f) = \sqrt{\frac{|A \backslash A_f|}{|A|}}$$

but this is not a metric, since it doesn't obey axiom (ii) of definition 2.2.

__Definition 10.1.__ Let (X,d) be a complete metric pixel space. We define the ***Pixset distance*** between points A and B in $\mathcal{H}(X)$, which are sets in X, as:

$$d_P = (\ \underline{\text{Max}}\ : A,B \in \mathcal{H}(X),\ A,B \neq \varnothing : \left\{\frac{|A \backslash B|}{|A|}, \frac{|B \backslash A|}{|B|}\right\})$$

where $|A|$ denotes the ***cardinality*** of A, i.e. the amount of pixels contained in A. ♦

Consider the former example of the two disks A and B, where the Hausdorff distance gave a wrong notion of closeness. The pixset distance would in this case only yield a distance which equals the quotient of the amount of pixels in the line segment and the amount of pixels in B, which will be a small number. Clearly, the Pixset distance is able to model the proposition $A \cap B = \varnothing$ (see theorem 10.7). Assume that two structures A and A_f both contain n points, then the computational time complexity of an algorithm to compute $d_P(A,A_f)$ is clearly $O(n)$. Before we are able to prove that d_P is a metric, we must introduce three theorems, which we need for the proof.

Theorem 10.5. Let (X,d) be a complete metric pixel space, then:

$$(\underline{\mathrm{A}}\ A,B : A,B \in \mathcal{H}(X) : |A \cap B| \leq (\underline{\mathrm{Min}} :: (|A|,|B|)))$$

Proof. Let $A,B \in \mathcal{H}(X)$ and $|A| \geq |B|$, then:

$|A \cap B|$
$\leq$ ⟨ definition of Intersection ⟩
$|B|$

Let $A,B \in \mathcal{H}(X)$ and $|A| \leq |B|$, then:

$|A \cap B|$
$\leq$ ⟨ definition of Intersection ⟩
$|A|$ ■

Theorem 10.6. Let (X,d) be a complete metric pixel space, then:

$$(\underline{\mathrm{A}}\ A,B : A,B \in \mathcal{H}(X) : |A| \geq |B| \Rightarrow d_P(A,B) = 1 - |A \cap B|/|A|)$$

Proof. Let $A,B \in \mathcal{H}(X)$ and $|A| \geq |B|$ and $A,B \neq \varnothing$, then:

$d_P(A,B)$
$=$ ⟨ definition 10.1 ⟩
$(\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : (|A \backslash B|/|A|, |B \backslash A|/|B|))$
$=$ ⟨ calculus ⟩
$(\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : ((|A|-|A \cap B|)/|A|, (|B|-|B \cap A|)/|B|))$
$=$ ⟨ $|A| \geq |B|$, $A,B \neq \varnothing$, theorem 10.5 ⟩
$(|A|-|A \cap B|)/|A|$
$=$ ⟨ $A \neq \varnothing$, calculus ⟩
$1 - |A \cap B|/|A|$ ■

Theorem 10.7. Let (X,d) be a complete metric pixel space, then:

$$(\underline{\mathrm{A}}\ A,B : A,B \in \mathcal{H}(X) : d_P(A,B) = 1 \Leftrightarrow A \cap B = \varnothing)$$

Proof. Let $A,B \in \mathcal{H}(X)$ and $A \cap B = \varnothing$ and $A,B \neq \varnothing$, then:

$d_P(A,B) = 1$
$\Rightarrow$ ⟨ definition 10.1 ⟩
$(\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : (|A \backslash B|/|A|, |B \backslash A|/|B|)) = 1$
$\Rightarrow$ ⟨ calculus ⟩
$(\underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : ((|A|-|A \cap B|)/|A|, (|B|-|B \cap A|)/|B|)) = 1$
$\Rightarrow$ ⟨ definition of Max operator ⟩

$(|A|-|A\cap B|)/|A| = 1 \vee (|B|-|B\cap A|)/|B| = 1$
$\Rightarrow \langle A,B \neq \varnothing$, calculus $\rangle$
$1-|A\cap B|/|A| = 1 \vee 1-|B\cap A|/|B| = 1$
$\Rightarrow \langle A,B \neq \varnothing$, calculus $\rangle$
$A\cap B = \varnothing$

$A\cap B = \varnothing$
$\Rightarrow \langle A,B \neq \varnothing$, calculus $\rangle$
$1-|A\cap B|/|A| = 1 \vee 1-|B\cap A|/|B| = 1$
$\Rightarrow \langle A,B \neq \varnothing$, calculus $\rangle$
$(|A|-|A\cap B|)/|A| = 1 \vee (|B|-|B\cap A|)/|B| = 1$
$\Rightarrow \langle$ definition of $\underline{\mathrm{Max}}$ operator $\rangle$
$(\ \underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : ((|A|-|A\cap B|)/|A|,(|B|-|B\cap A|)/|B|)\ ) = 1$
$\Rightarrow \langle$ calculus $\rangle$
$(\ \underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : (|A\backslash B|/|A|,|B\backslash A|/|B|)\ ) = 1$
$\Rightarrow \langle$ definition 10.1 $\rangle$
$d_P(A,B) = 1$ ■

Theorem 10.8. d_P is a metric on $\mathcal{H}(X)$.

Proof. We have to prove that d_P obeys the four axioms of definition 2.2.

(i) To be proven: $(\ \underline{\mathrm{A}}\ A : A \in \mathcal{H}(X) : d_P(A,A) = 0\ )$

Let $A \in \mathcal{H}(X)$ and $A \neq \varnothing$, then:

$d_P(A,A)$
$= \langle$ definition 10.1 $\rangle$
$(\ \underline{\mathrm{Max}} : A \in \mathcal{H}(X), A \neq \varnothing : (|A\backslash A|/|A|,|A\backslash A|/|A|)\ )$
$= \langle A \neq \varnothing$, calculus $\rangle$
$(\ \underline{\mathrm{Max}} :: (0,0)\ )$
$= \langle$ calculus $\rangle$
0

(ii) To be proven: $(\ \underline{\mathrm{A}}\ A,B : A,B \in \mathcal{H}(X) : d_P(A,B) = d_P(B,A)\ )$

Let $A,B \in \mathcal{H}(X)$ and $A,B \neq \varnothing$, then:

$d_P(A,B)$
$= \langle$ definition 10.1 $\rangle$
$(\ \underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : (|A\backslash B|/|A|,|B\backslash A|/|B|)\ )$
$= \langle$ symmetry of $\underline{\mathrm{Max}}$ operator $\rangle$
$(\ \underline{\mathrm{Max}} : A,B \in \mathcal{H}(X), A,B \neq \varnothing : (|B\backslash A|/|B|,|A\backslash B|/|A|)\ )$
$= \langle$ definition 10.1 $\rangle$
$d_P(B,A)$

(iii) To be proven: $(\underline{\text{A}}\ A,B : A,B \in \mathcal{H}(X),\ A \neq B : 0 < d_P(A,B) < \infty)$

Let $A,B \in \mathcal{H}(X)$ and $A \neq B$, then:

$d_P(A,B)$
$= \langle$ definition 10.1 $\rangle$
$(\underline{\text{Max}} : A,B \in \mathcal{H}(X),\ A,B \neq \varnothing : (|A \backslash B|/|A|, |B \backslash A|/|B|))$
$= \langle$ calculus $\rangle$
$(\underline{\text{Max}} : A,B \in \mathcal{H}(X),\ A,B \neq \varnothing : ((|A|-|A \cap B|)/|A|, (|B|-|B \cap A|)/|B|))$

case iiia) $= \langle\ |A| > |B|$, theorem 10.6 $\rangle$
$1 - |A \cap B|/|A|$
$\Rightarrow \langle\ |A| > |B|,\ A,B \neq \varnothing \rangle$
$0 < 1 - |A \cap B|/|A| \leq 1$

case iiib) $= \langle\ |A| < |B|$, theorem 10.6 $\rangle$
$1 - |A \cap B|/|B|$
$\Rightarrow \langle\ |A| < |B|,\ A,B \neq \varnothing \rangle$
$0 < 1 - |A \cap B|/|B| \leq 1$

Hence: $0 < (\underline{\text{Max}} : A,B \in \mathcal{H}(X),\ A,B \neq \varnothing : (|A \backslash B|/|A|, |B \backslash A|/|B|)) < \infty$

(iv) To be proven: $(\underline{\text{A}}\ A,B,C : A,B,C \in \mathcal{H}(X),\ A,B,C \neq \varnothing : d_P(A,B) \leq d_P(A,C) + d_P(C,B))$

Let $A,B,C \in \mathcal{H}(X)$ and $A,B,C \neq \varnothing$, then:

case iva)

$= \langle\ |A| \geq |B|,\ |B| \geq |C|$, theorem 10.6 $\rangle$
$(\underline{\text{A}} A,B,C : A,B,C \in \mathcal{H}(X), A,B,C \neq \varnothing : 1-|A \cap B|/|A| \leq 2-|A \cap C|/|A|-|B \cap C|/|B|)$
$= \langle$ calculus $\rangle$
$(\underline{\text{A}} A,B,C : A,B,C \in \mathcal{H}(X), A,B,C \neq \varnothing : |B|\,|A \cap B| \geq |B|\,|A \cap C| + |A|\,|B \cap C| - |A|\,|B|)$
$\Rightarrow \langle$ theorem 10.5 $\rangle$
$(\underline{\text{A}}\ A,B,C : A,B,C \in \mathcal{H}(X), A,B,C \neq \varnothing : |B|\,|B| \geq |B|\,|A \cap C| + |A|\,|B \cap C| - |A|\,|B|)$
$= \langle$ calculus $\rangle$
$(\underline{\text{A}}\ A,B,C : A,B,C \in \mathcal{H}(X), A,B,C \neq \varnothing : |B|(|B|-|A \cap C|) \geq |A|(|B \cap C|-|B|))$
$= \langle\ |A| \geq |B|,\ |B| \geq |C|$, theorem 10.5 $\Rightarrow |B|-|A \cap C| \geq 0 \wedge |B \cap C|-|B| \leq 0 \rangle$
true

case ivb)

$= \langle\ |A| \geq |B|,\ |B| < |C|\ \rangle$

case ivb1) $= \langle\ |A| \geq |C|$, similarly case iva $\rangle$
true
case ivb2) $= \langle\ |A| < |C|$, similarly case iva $\rangle$
true

case ivc)

$= \langle\ |A| < |B|,\ |B| \leq |C|\ \rangle$

case ivc1) $= \langle\ |A| \geq |C|$, similarly case iva $\rangle$
true

case ivc2) $= \langle\ |A| < |C|$, similarly case iva $\rangle$
true

case ivd)

$= \langle\ |A| < |B|,\ |B| < |C|$, similarly case iva $\rangle$
true

Hence, d_P is a metric on $\mathcal{H}(X)$. ■

We might also have choosen a Pixset distance of for example $d_P' = (1 - |A \cap B| / |A \cup B|)$, but this function is not linear. Suppose we convolute two identical rectangles and we plot the percentage overlap against d_P then we get a straight line, but if we plot the percentage overlap agains d_P' we get a nonlinear graph.

Despite the fact that solving the IFS-encoding problem in closed form with help of the moment theory (see par. 10.3) will not succeed in polynomial time (see par. 10.2), theorem 9.4 still holds. This implies that we can compare the k^{th} moment sequences of a target invariant measure μ and its approximated invariant measure $\nu = M(\mu)$, which can both be calculated in linear time [RED89].

<u>*Definition 10.5.*</u> Let $m(\mu,k)$ and $m(\nu,k)$ be the k^{th} moments of resp. a target invariant measure μ and its approximated invariant measure $\nu = M(\mu)$, then we define the ***k*th moment distance** as [SHO91]

$$d_m(\mu,\nu) = \sum_{i=1}^{k} \left(|m(\mu,k) - m(\nu,k)|^2 \right)^{1/2}$$ ♦

The reader can easily verify that d_m is a metric on $\mathcal{P}(X)$ (see definition 2.2). Unfortunately, the k^{th} moment distance appears to be very unsatisfactory as quality measure. As many as 60 moments of two totally different 1-dimensional invariant measures can agree to each other up to 10 decimal places. Even more disturbing is the fact that tuning the associated IFS-code very slightly, gives rise to another IFS, which attractor is close to the attractor of the original IFS, but their moment sequences may differ considerably [SHO91].

The above considerations lead us to step away from the k^{th} moment distance as practical quality measure. If we want to measure the quality of an approximation ν of the invariant measure μ of a given color or greytone image A in a pixel space, then we can also use the Hutchinson metric d_H (see definition 9.11). The Hutchinson metric can be computed for a $|\mathit{Screen}| = (x_{right} - x_{left})(y_{lower} - y_{upper})$ cell discretized measure, as follows. First the nondeterministic IFS-decoding algorithm (see par. 8.3) must be applied to generate ν, which can be done in $O(n)$ time for generating n pixels. In this way a "histogram" is created, which is stored in the frame buffer. If this frame buffer contains b bitplanes, then we have 2^b discrete histogram values. Of course we must not "saturate" the image by applying too much iterations, to get a reliable result (if we iterate long enough, all histogram values will be either 0 or 2^b-1). Finally, the Hutchinson metric can be computed in $O(|Screen|)$ time by:

$$d_H = \sum_{i=1}^{|Screen|} |\mu(i) - \nu(i)| \quad \text{for } \mu(i),\nu(i) \in \{0..2^b\}$$

So the computation of the Hutchinson metric requires $O(n+|Screen|)$ computation time, which is linear in the amount of pixels. In [SHO91] it is reported that the Hutchinson metric works splendidly as quality measure for the approximation quality of 1-dimensional invariant measures.

If we are going to apply an algorithm for automatic IFS-encoding of a target image $A \in \mathcal{H}(X)$, we will have to evaluate candidate solutions which are represented by an IFS consisting of N affine transformations. The most straight forward way to do this is by deterministically rendering the attractor A_f of the IFS and calculate $d_P(A,A_f)$. It is obvious that the (sequential) rendering process is a time consuming step. This can be avoided by applying the following method. An approximate attractor $\check{A}_f$ of the IFS can be obtained by applying 1 iteration of deterministic IFS-decoding with condensation set A. It is obvious that $\check{A}_f = w_1(A) \cup w_2(A) \cup \ldots \cup w_N(A)$.

Concluding, the quality measure for the approximation of:

(i) a set $A \in \mathcal{H}(X)$ must be done as follows:

First, apply 1 iteration of deterministic IFS-decoding with condensation set A to render an approximate attractor $\check{A}_f = w_1(A) \cup w_2(A) \cup \ldots \cup w_N(A)$, which can be done in $O(\log(N+|A|))$ time for sequential computation. Second, calculate the Pixset distance $d_P(A,\check{A}_f)$, which can be done in $O(|A|)$ time.

(ii) an invariant measure $\mu \in \mathcal{P}(X)$ must be done as follows:

First, apply a sufficient amount of n iterations of nondeterministic IFS-decoding to render the attractor ν, which can be done in $O(n)$ time for sequential computation. Second, calculate the Hutchinson distance $d_H(\mu,\nu)$, which can be done in $O(|Screen|)$ time.

I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection.

Charles Darwin (1809 - 1882) [DAR80]

Chapter 11

Genetic IFS-encoding

From chapter 10 we know that IFS-encoding is NP-complete, which enforces us to use a heuristic approach to tackle the problem. There are several possibilities, for example greedy algorithms, simulated annealing, neural networks and genetic algorithms.

Greedy algorithms seem to be the least powerful of the four mentioned alternatives. Simulated annealing takes decisions based on information which is memorized by means of the evaluation value of the last evaluated member. It can therefore only perform single steps through the state space, based on a comparision of the evaluations of the current and the former candidate solutions. Nevertheless, good results have been reported with applying this method on the IFS-encoding problem [LEM81,LEV87,LEV88,MAN89]. Neural networks seem to be very promising to the IFS-encoding problem, but a major drawback is the fact that all pixels have to be represented by separate neurons, which also have to be connected to form a network [STA91a,STA91b]. It is obvious that this method implies an extreme memory load. By the way, it is noteworthy that in [BRE91] the idea is ventilated to use the collage theorem to develop supervised learning schemes for neural networks.

Based upon the former discussion, and also upon its promising results according to 1-dimensional IFS-encoding [SHO91], we have choosen for a genetic approach to tackle the IFS-encoding problem. We decided to concentrate upon finding IFS-codes for 2-dimensional monochrome images, thus omitting the probabilities.

11.1. Genetic algorithms

Genetic algorithms were invented by Holland to mimic the process of natural selection according to the natural evaluation theory of Darwin [HOL75]. The mechanisms that drive the evolution are not yet fully understood, but some of its features are known. The evaluation process operates on chromosomes, organic molecules built up from an alphabet of the 4 digits Adenine (A), Thymine (T), Guanine (G) and Cytosine (C), which are molecules themselves [SIN91]. Chromosomes encode the structure and properties of living organisms, and natural selection is the link between chromosomes (genotypes) and the performance of their decoded structures (phenotypes). The process of natural selection causes those chromosomes which encode successful structures, to reproduce more often than those which do not.

The process of reproduction is the point where evolution takes place. Mutations may cause the chromosomes of biological children to be different from those of their biological parents, and recombination processes may create quite different chromosomes in the children by combining material from the chromosomes of two parents. Biological evolution has no explicit memory. Whatever it knows about producing individuals that will function well in their environment, is contained implicitly in the chromosome set of the current generation of individuals [DAV91].

The way a genetic algorithm works is as follows. Candidate solutions of the problem to be solved are encoded into chromosomes. A first generation of chromosomes is (randomly) initialized. An evaluation function is defined to evaluate the fitness of each member of the polulation, by decoding their genotypes (chromosomes) into phenotypes. The better the candidate solution, the better its assigned fitness. The next step is to select two members from the current generation, which will act as parents. The probability that a certain parent is selected, is directly proportional to its fitness. In this way the fittest members are more frequently choosen than less fit members. After two parents have been selected, a genetic operator is selected. There can be several genetic operators and each one has an assigned probability of being choosen. Then the selected operator is applied to the selected parent pair, which yields a pair of so called children. Those children are placed in the next generation. As soon as the next generation contains as many members as the current generation, the next generation becomes the current generation and a certain best percentage of the current generation is merged into the next generation to avoid loss of good genetic material. This process is repeated until a satisfactory solution (best population member of the current generation) is found.

In our case the genetic algorithm is embedded within a while loop which increments the number of the affine transformation N we search for at each iteration. The loop guard compares the maximum allowed error with the Pixset distance between the target image and the current collage. In this way we avoid to repeatedly search for a collage consisting of a fixed amount of affine transformations. In fig. 11.1 the general outline of our genetic algorithm is given in pseudo code.

```
genotype gen[POP_SIZE],nxtgen[POP_SIZE];
int c1,c2,i,j,k,n,p1,p2,N=1;
while (PixsetDistance(image, $\bigcup_{i=1}^{N}$ w_i(image)) > MAX_ERROR)
 {
  InitializePopulation(gen);
  while (gen[0].fitness < 1 - MAX_ERROR)
   {
    for n:=1 to POP_SIZE/2
      {
       p1 := SelectParent(gen);
       p2 := SelectParent(gen);
       k  := SelectGeneticOperator;
       Operator[k](gen[p1],gen[p2],nxtgen[c1],nxtgen[c2]);
      }
    EvaluateAndSort(nxtgen);
    gen := nxtgen ∪ { gen[j] : 0 ≤ j ≤ POP_SIZE/10 ];
   }
  Transformation[N] := Decode(gen[0]);
  N++;
 }
```

Figure 11.1. Pseudo code for genetic IFS-encoding algorithm.

Like nature, genetic algorithms solve the problem of finding good chromosomes by manipulating the material in the chromosomes blindly, and they have no information about the problem they are actually solving. The only information they have, consists of evaluation values of each produced chromosome and their only use of that evaluation is to bias the selection of chromosomes such that those with the best evaluations tend to reproduce more often than those with bad evaluations.

11.2. Chromosome encoding

The first step in developing a genetic algorithm is to design an appropriate scheme for encoding phenotypes (candidate solutions) into genotypes (chromosomes). In our case a phenotype consists of an affine transformed copy of the target image, i.e. the image we want to encode into an IFS. The corresponding affine transformation w can be represented by a ***state space vector***, i.e. the 6-tuple (a,b,c,d,e,f), with $a,b,c,d,e,f \in [-1,1]$. The reader may have noticed that some IFS-codes in chapter 8 contain values $e,f \notin [-1,1]$, but this is just a machine dependency. In our case we use a pixel space ***Pixmap*** = $[-250,250] \times [-250,250]$, so for rendering of attractors we just apply the multiplications $e \cdot x_{right}$ and $f \cdot y_{lower}$. Thus an affine transformation can be represented by the above introduced 6-tuple state space vector. We can encode such state space vector into a chromosome, which consists of genes.

__Definition 11.1.__ A ***chromosome*** is a 6-tuple $(g_a,g_b,g_c,g_d,g_e,g_f)$ where $g_a,..,g_f$ are called ***genes*** with

$$(\underline{A}\, i : a \leq i \leq f : 1 \leq g_i \leq 255)$$

So a gene g contains 8 bits $g_0,..,g_7$, each of which is called a ***nucleotide***. ◆

In this way we obtain a chromosome of 48 [*bits*] which contains 6 genes of each 8 [*bits*], each of which bit is called a nucleotide, and such that the MSB g_7 is the sign bit and the 7 other bits are the data bits. So the numbers 1, 0 and -1 are represented by respectively 127, 128 and 255. The chromosome is the genotype which encodes the phenotype, i.e. the affine transformed copy of the target image, and each gene encodes a property of this phenotype, i.e. a parameter of the affine transformation (xscaling, yscaling, xrotation, yrotation, xtranslation, ytranslation). The in this way obtained encoding resolution is $(127)^{-1}$ or about 0.8 [%]. For the translation parameters e and f this implies a resolution of $250 \cdot (127)^{-1} \approx 2$ [*pixels*]. In fig. 11.2 the chromosome encoding process is represented schematically. It is obvious that this way of chromosome encoding implicitly places constraints on the phenotypes as to the scaling, rotation and translation intervals.

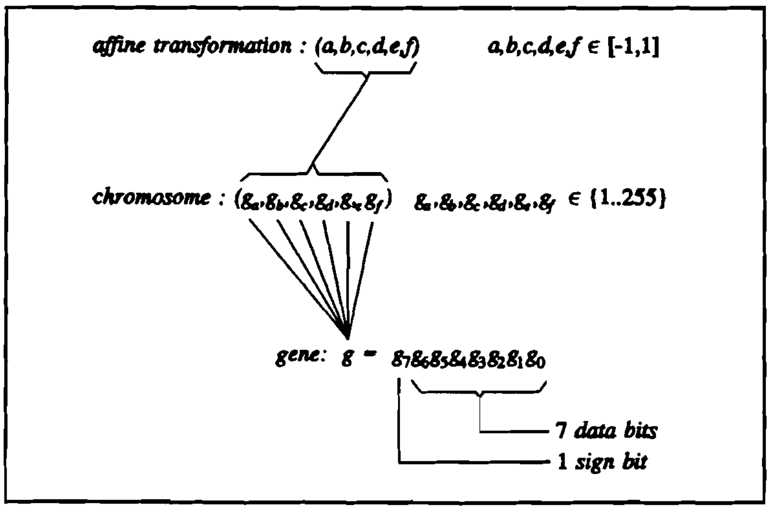

affine transformation : (a,b,c,d,e,f)
a,b,c,d,e,f ∈ [-1,1]
chromosome : (g_a,g_b,g_c,g_d,g_e,g_f)
g_a,g_b,g_c,g_d,g_e,g_f ∈ {1..255}
gene: g = g_7g_6g_5g_4g_3g_2g_1g_0
7 data bits
1 sign bit

Figure 11.2. Chromosome encoding.

11.3. The genetic operators

The most common genetic operators are ***crossover*** and ***mutation***. Crossover operates on two parent chromosomes from which it creates two child chromosomes in such way that each child chromosome contains material from both parent chromosomes. Mutation operates on one parent chromosome from which it creates one child chromosome. In our case, applying only those two operators appeared to be not powerful enough (see next chapter), so we designed some additional genetic operators. Alltogether, our genetic algorithm contains 8 genetic operators (*MacroCrossover*, *MicroCrossover*, *MixedCrossover*, *Mutate*, *Jump*, *Scale*, *Rotate*, *Translate*), each of which will be explained hereafter.

MacroCrossover operates on genes. It takes as input two parent chromosomes and generates a random *mask* as element of {1,..,63} which represents a bitstring of length 6. The bits from *mask* indicate gene locations on the parent chromosomes. If the MSB of *mask* is not set, then the chromosome of *child1* gets the gene of *parent1* at the leftmost position. In the same way *child2* gets the according gene of *parent2*. If the MSB of *mask* is set, then the genes in the leftmost position are crossed, i.e. *child1* gets the gene of *parent2* and vice versa. This process is applied on all 6 genes within the chromosome (see fig. 11.3).

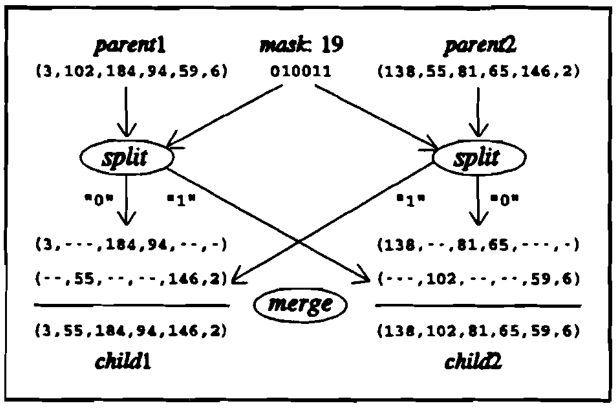


Figure 11.3. The genetic operator *MacroCrossover*.

MicroCrossover operates on nucleotides. It takes as input two parent chromosomes and generates a random *mask* as element of {1,..,255}, which represents a bitstring of length 8. The bits of *mask* indicate nucleotide locations on the parent genes. If the MSB of *mask* is not set, then each gene of the chromosome of *child1* gets the nucleotide of the according gene of *parent1* at the leftmost position, i.e. the gene's sign bit. In the same way the genes of *child2* get at their leftmost position the sign bit (leftmost nucleotide) of the genes of *parent2*. If the MSB of the mask is set, then the nucleotides in the leftmost positions are crossed, i.e. *child1* gets the nucleotide of *parent2* and vice versa. This process is applied on all 8 nucleotides within all 6 genes (see fig. 11.4).

MixedCrossover operates on genes and nucleotides. It takes as input two parent chromosomes and generates a random *mask1* as element of {1,..,63} and a random *mask2* as element of {1,..,255}, which respectively represent bitstrings of length 6 and length 8. The bits of *mask1* indicate gene locations on the parent chromosomes, and the bits of *mask2* indicate nucleotide locations on the parent genes. *MixedCrossover* basically does what its name already suggests. It only applies *MicroCrossover* on the genes which are selected by *MacroCrossover*. In other words, *mask1* determines on which genes *MicroCrossover* is going to be applied according to *mask2* (see fig. 11.5).

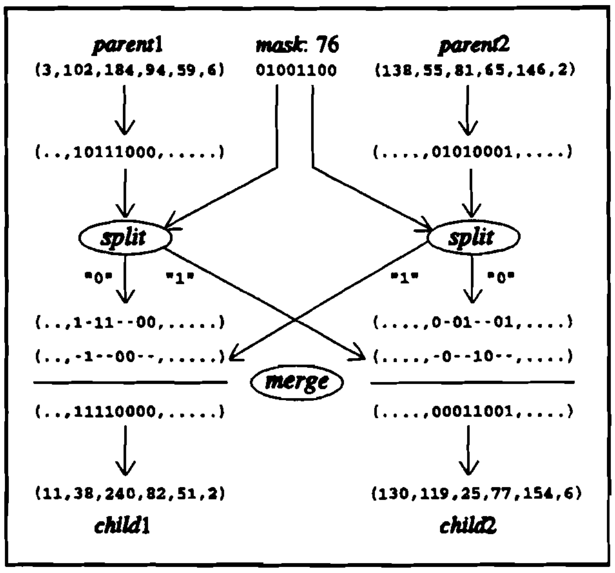


Figure 11.4. The genetic operator *MicroCrossover*.

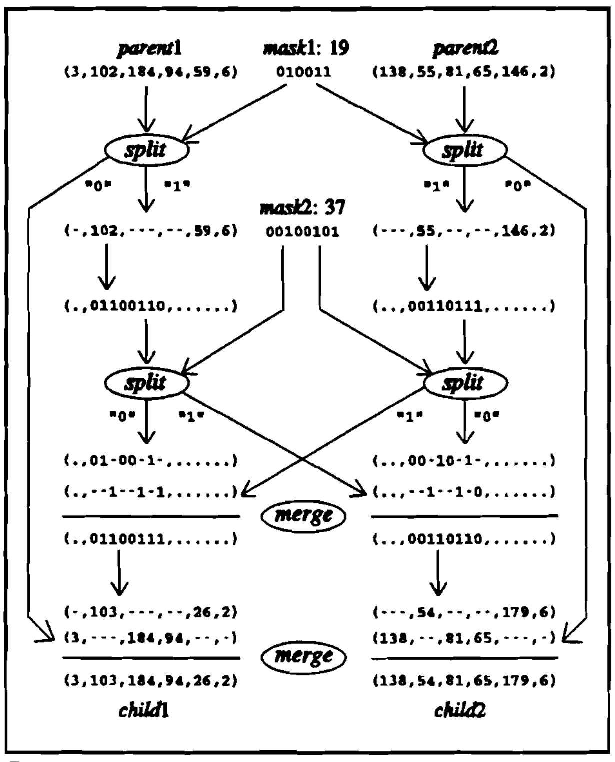


Figure 11.5. The genetic operator *MixedCrossover*.

Mutate operates on genes and nucleotides. It takes as input one parent chromosome and generates a random *mask1* as element of {1,..,6} and a random *mask2* as element of {1,2,4,8,16,32,64,128}. *Mask1* represents just a gene location on a chromosome (1 represents the leftmost gene, 6 represents the rightmost gene) and *mask2* consists of a bitstring of length 8, of which exactly one bit is set. The position of this bit corresponds to the position of a nucleotide in the gene which is pointed to by *mask1*. The only action *Mutate* takes, is to copy the parent chromosome into the child chromosome and invert the nucleotide bit at position *mask2* in the gene at position *mask1* (see fig. 11.6).

Jump operates on genes. It takes as input one parent chromosome and generates a random *mask1* and a random *mask2* as element of respectively {1,..,6} and {1,..,6}*mask1*. These masks represent two different gene locations within the parent chromosome. *Jump* creates a child by copying its chromosome from the parent chromosome and swapping the genes at the positions *mask1* and *mask2* (see fig. 11.7).

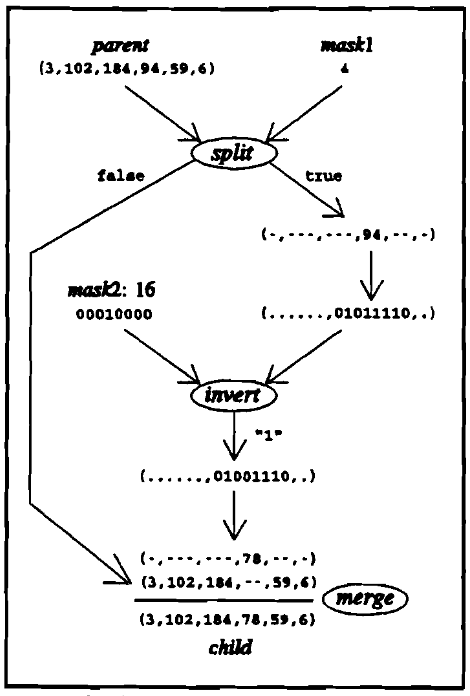


Figure 11.6. The genetic operator *Mutate*.

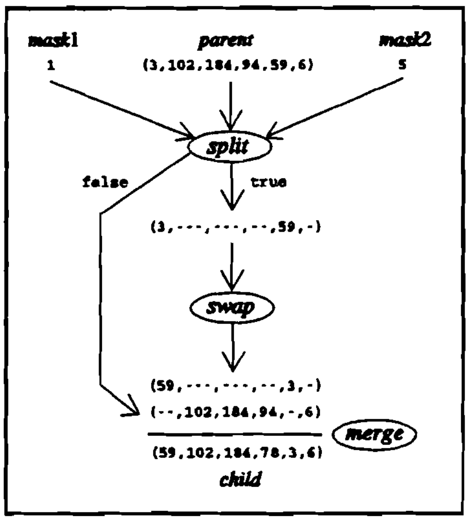


Figure 11.7. The genetic operator *Jump*.

Scale operates on genes. It takes as input one parent chromosome and generates a random *mask* as element of {1,..,126}, which represents a scaling factor in the interval [1/127,126/127]. *Scale* copies the parent chromosome into the child chromosome and multiplies the leftmost four genes - which represent the matrix *A* of the affine transformation - with *mask* (see fig. 11.8).

Rotate operates on genes. It takes as input one parent chromosome and generates a random *mask* as element of {1,..,359}, which represents a rotation angle in degrees. The mask is converted into an angle φ [*radians*]. *Rotate* copies the parent chromosome into the child chromosome and multiplies the leftmost four genes - which represent the matrix *A* of the affine transformation - with the standard rotation matrix (see fig. 11.9):

$$\begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$$

Translate operates on genes. It takes as input one parent chromosome and generates a random *mask* as element of {1,..,255}, which represents a translation vector $[mask, mask]^T$. *Translate* copies the parent chromosome into the child chromosome and adds (*mask* mod 255) to the rightmost two genes, which represent the translation vector *t* of the affine transformation (see fig. 11.10).

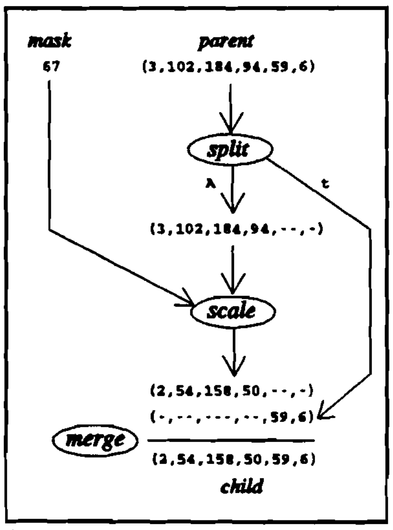


Figure 11.8. The genetic operator *Scale*.

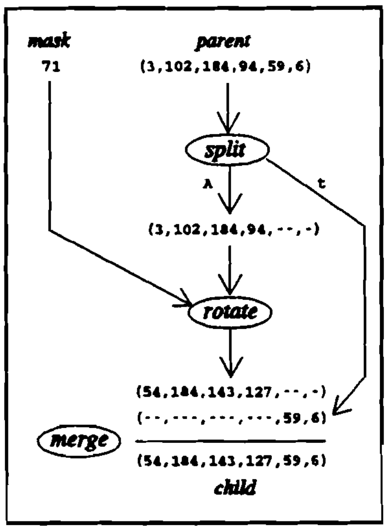


Figure 11.9. The genetic operator *Rotate*.

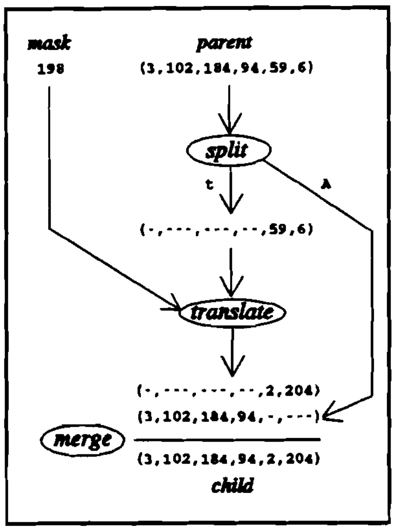


Figure 11.10. The genetic operator *Translate*.

11.4. The evaluation function

To evaluate the fitness of population members, we need an evaluation function. Depending on the kind of problem, we have to design an evaluation function which assigns penalties and rewards to certain properties of population members. Basically we have defined two main problems, each of which asks for a different approach. Those two main problems are finding the identity of a target image and finding a collage of a target image. Finding the identity is important to validate the genetic algorithm, but also to investigate its problem solving power in this field. Finding the collage is of course directly related to our IFS-encoding problem. Before we can proceed, we need to go through some more definitions.

Definition 11.2. Let ***Pixset*** be a pixel space and let $\{target, member, collage\} \subset$ ***Pixset***, where:

target	=	the to be encoded target image;
member	=	$w(target)$, the affine transformed copy of *target* according to the chromosome of *member*;
collage	=	$w_1(target) \cup w_2(target) \cup \ldots \cup w_N(target)$, the union of all affine transformed copies of *target* which are found so far, i.e. each best *member* of the last generation of the preceding runs of the genetic algorithm.

With $|target|$, $|member|$ and $|collage|$ we denote the cardinalities in [*pixels*] of the respective images and with N the number of the current affine transformation. ♦

Definition 11.3. A ***reward*** is a positive contribute to the fitness of *member* and a ***penalty*** is a negative contribute to the fitness of *member*. We define the following penalties and rewards: *strongpenalty* (*SP*), *weakpenalty* (*WP*), *borderreward* (*BR*), *fillreward* (*FR*) and *sizereward* (*SR*), which are quantified as follows:

$$SP = |member \backslash target| / |member|$$

$$WP = |target \cap member \cap collage| / |member|$$

$$BR = |(target \backslash collage) \cap member| / |member|$$

$$FR = |(target \backslash collage) \cap member| / |target \backslash collage|$$

$$SR = \begin{cases} ((N+1)\,|member|)/|target| & \text{for } |member|/|target| \in [0, 1-\frac{N}{N+q}] \\ ((1+\frac{1}{N})(1-|member|/|target|) & \text{for } |member|/|target| \in [1-\frac{N}{N+q}, 1] \end{cases}$$

with $N \in \mathbf{N}^+$ the number of the affine transformation and $q \in \mathbf{N}$ an offset (in our case 1). ♦

The penalties and rewards which are introduced in definition 11.3 are derived from the following considerations (see fig. 11.11).

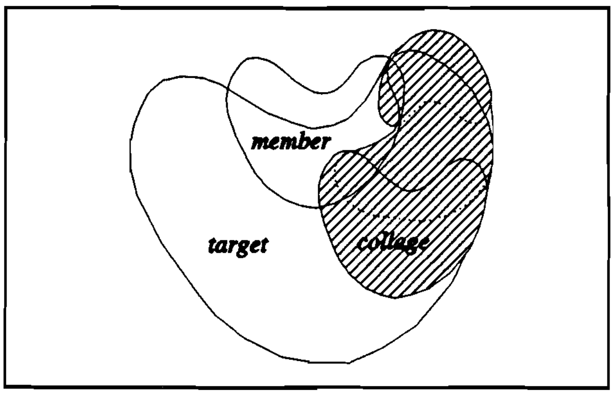


Figure 11.11. An impression of *target*, *member* and *collage*.

In order to get a good fitness, *member* must cover a reasonable part of the up till now uncovered part of *target* and must be located totally within *target*, covering as less as possible of *collage*. The amount of covering of the uncovered part of *target* is measured by *fillreward* (*FR*). The relative part of *member* which accounts for *fillreward* is measured by *borderreward* (*BR*). The relative part of *member* which lays outside *target* is measured by *strongpenalty* (*SP*) and the relative part of *member* which covers *collage* is measured by *weakpenalty* (*WP*). It makes sense to assume that when the tiling proceeds, we want to have smaller and smaller affine transformed copies (tiles) in order to be able to achieve an accurate collage. Therefore *sizereward* (*SR*) keeps track of the size of the tiles. This reward is adjusted each time as N (the number of the current affine transformation) is increased. Suppose that $q=1$, then if $N=1$ the optimal size of a tile *member* is half the size of *target*. When $N=2$ the optimal size of *member* is one third of the size of *target*, etc. (see fig. 11.12).

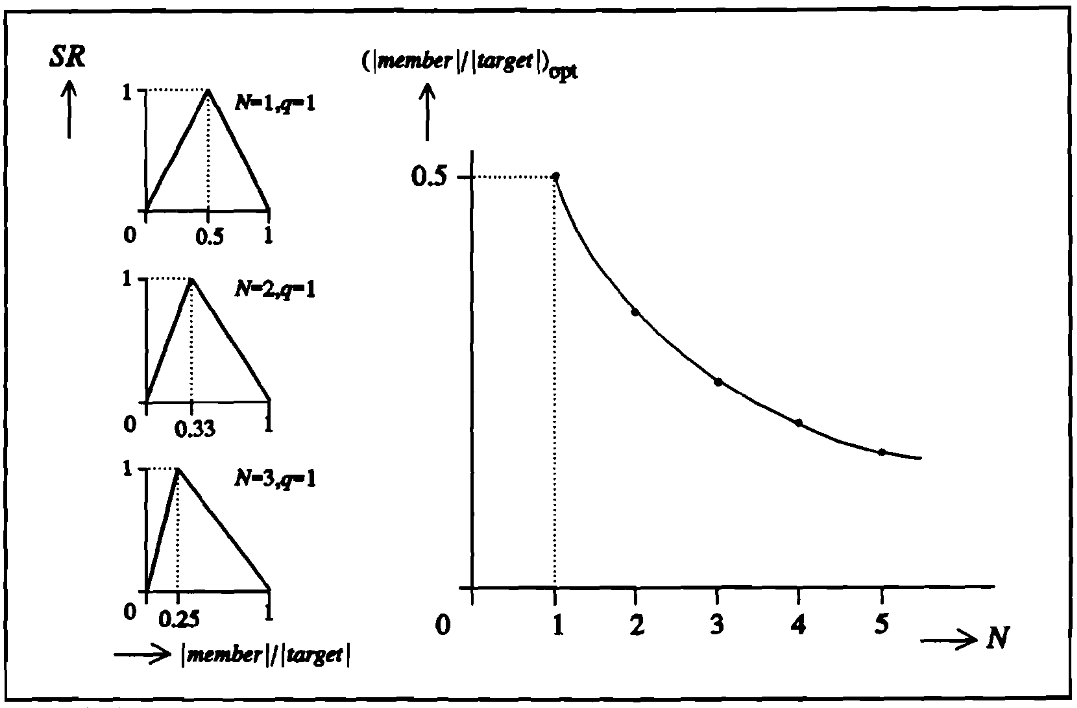


Figure 11.12. The sizereward *SR* is biased to the affine transformation number *N*.

Definition 11.4. We define the ***evaluation function*** $E : \mathbb{R}^+ \cup \{0\} \rightarrow (0,1)$ as:

$$E = (k_0 + k_1 BR + k_2 FR + k_3 SR - k_4 SP - k_5 WP)/k_6$$

where $k_0,k_1,k_2,k_3,k_4,k_5 \in \mathbb{R}^+ \cup \{0\}$ and $k_6 \in \mathbb{R}^+$ and $BR,FR,SR,SP,WP \in [0,1]$. ♦

The value of the evaluation function will be assigned directly to the *member* fitness. Whatever values we choose for $k_0,..,k_6$ we must assure that the obtained member fitness is always contained in the interval (0,1). In the genetic algorithm this is done by the following C-statement:

```
if ((int) 1000000*(member[i].fitness) <= 0)
    {
        member[i].fitness = 0.000001;
    }
if ((int) 1000000*(member[i].fitness) >= 1000000)
    {
        member[i].fitness = 0.999999;
    }
```

The tuning of the parameters k_i is of great influence on the performance of the genetic algorithm. A guide for tuning this parameters may be fig. 11.13, where a precedence relation is given on some possible covering situations.

Not only tuning of the parameters k_i, but also of the other parameters, such as population size, amount of generations, etc., influence the performance of the genetic algorithm. In par. 11.6 we will further comment on this subject. In our case we kept the offset q in SR equal to 1.

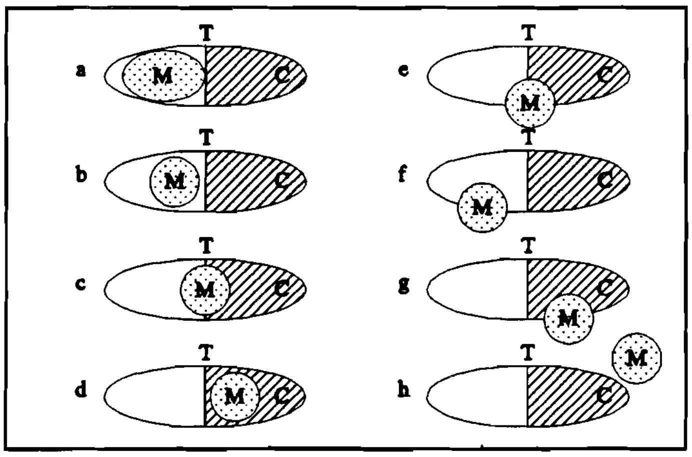


Figure 11.13. Precedence relation of member fitnesses (T = *target*, M = *member* and C = *collage*). Fitness(M): a>b>c>d>e>f>g>h.

Definition 11.5. A ***Genetic Parameter Setting (GPS)*** is a 7-tuple (N,G,S,T,P,K,ϵ), where:

N $\in \mathbf{N}^+$ is the maximum amount of affine transformations in *collage*;

G $\in \mathbf{N}^+$ is the maximum amount of generations;

S $\in \mathbf{N}^+$ is the population size;

T $\in [0,1]$ is the percentage of transferred best members to the next generation;

P $= \{ p_i : 1 \leq i \leq 8 \}$ is the set of operator probabilites, where $p_1 = p(MacroCrossover)$, $p_2 = p(MicroCrossover)$, $p_3 = p(MixedCrossover)$, $p_4 = p(Mutate)$, $p_5 = p(Jump)$, $p_6 = p(Scale)$, $p_7 = p(Rotate)$, $p_8 = p(Translate)$;

K $= \{ k_i : 0 \leq i \leq 6 \}$ is the set of evaluation function parameters, where $k_6 \in \mathbf{R}^+$ and $k_0,k_1,k_2,k_3,k_4,k_5 \in \mathbf{R}^+ \cup \{0\}$;

ϵ $\in [0,1)$ is the maximum encoding error. ♦

11.5. Initialization

The initial population (generation 0) the genetic algorithm starts with, can be created either randomly or by means of a heuristic. We have choosen for the latter option, and we will use the information provided by inverse fixed point analysis (see par. 7.2). The input of the process is a target image A, which is represented in some pixel space *Pixmap*. From the collage theorem (theorem 10.1) we can deduce the fact that the fixed points x_f of all IFS-transformations must be elements of the target image, hence: $(\underline{\mathrm{A}}\, i : 1 \leq i \leq N : x_{f_i} \in A)$. Each member of the initial population is created as follows. First assign a random number in the interval $\{1,..,255\}$ to each of the four leftmost genes g_a,g_b,g_c,g_d in the chromosome. Second, convert these chromosome values to transformation values a,b,c,d. Third, take a random point $(x_f,y_f) \in A$. Next, calculate the transformation values $e = (1-a)x_f - by_f$

and $f = (1-d)y_f - cx_f$. Finally, convert the transformation values e end f into the respective chromosome values g_e and g_f and create the chromosome $(g_a,g_b,g_c,g_d,g_e,g_f)$. The former calculation of the transformation values e and f is based on theorem 11.1 which is stated below.

Theorem 11.1. Let $(\mathbb{R}^2,w)$ be a dynamical system with $w : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ an affine transformation given by: $w(x_1,x_2) = (ax_1 + bx_2 + e,\ cx_1 + dx_2 + f)$ with fixed point (x_f,y_f), then the transformation values of the translation vector $t = (e,f)^T$ are given by $e = (1-a)x_f - by_f$ and $f = (1-d)y_f - cx_f$.

Proof. Let the above affine transformation be given, then:

(x_f,y_f)

$= \langle$ theorem 7.2 $\rangle$

$$\left[\frac{bf + (1-d)e}{(1-a)(1-d) - bc},\ \frac{ce + (1-a)f}{(1-a)(1-d) - bc}\right]$$

$\Rightarrow \langle$ calculus $\rangle$

$$e = \frac{x_f[(1-a)(1-d) - bc] - bf}{(1-d)} \quad \wedge \quad f = \frac{y_f[(1-a)(1-d) - bc] - ce}{(1-a)}$$

$\Leftrightarrow \langle$ substitute f in left equation, calculus $\rangle$

$$e = \frac{x_f[(1-a)(1-d) - bc]}{(1-d)} - \frac{by_f[(1-a)(1-d) - bc]}{(1-a)(1-d)} + \frac{bce}{(1-a)(1-d)}$$

$\Leftrightarrow \langle$ calculus $\rangle$

$$e = \frac{(1-a)(1-d)}{(1-a)(1-d) - bc}\left[\frac{x_f[(1-a)(1-d) - bc]}{(1-d)} - \frac{by_f[(1-a)(1-d) - bc]}{(1-a)(1-d)}\right]$$

$\Leftrightarrow \langle$ calculus $\rangle$

$$e = \frac{(1-a)x_f[(1-a)(1-d) - bc] - by_f[(1-a)(1-d) - bc]}{(1-a)(1-d) - bc}$$

$\Leftrightarrow \langle$ calculus $\rangle$

$e = (1-a)x_f - by_f$

(x_f,y_f)

$= \langle$ theorem 7.2 $\rangle$

$$\left[\frac{bf + (1-d)e}{(1-a)(1-d) - bc},\ \frac{ce + (1-a)f}{(1-a)(1-d) - bc}\right]$$

$\Rightarrow \langle$ calculus $\rangle$

$$f = \frac{y_f[(1-a)(1-d) - bc] - ce}{(1-a)} \quad \wedge \quad e = \frac{x_f[(1-a)(1-d) - bc] - bf}{(1-d)}$$

$\Leftrightarrow \langle$ substitute e in left equation, similarly first part of proof $\rangle$

$f = (1-d)y_f - cx_f$ ■

11.6. Some results

Not all genetic operators were developed at the same time, but for clearity we will keep using the *GPS* notation as introduced in definition 11.5. Thus, whenever an operator probability is set to zero, this can either indicate that the operator was not yet developed or that it is just not selected, the effect is the same. The first error free version of our genetic algorithm only possessed five genetic operators, i.e. *MacroCrossover, MicroCrossover, MixedCrossover, Mutate* and *Jump*. The algorithm was tasked to find the identity of a black square and was initialized with the genetic parameter setting:

$$GPS_1 = (1,10,10,0.1,\{0.3,0.3,0.2,0.1,0.1,0.0,0.0,0.0\},\{1\text{-}d_P,0,0,0,0,0,1\},0)$$

After 10 generations it found a best member fitness of 0.653090. An initialization with the genetic parameter setting:

$$GPS_2 = (1,15,50,0.1,\{0.2,0.2,0.2,0.2,0.2,0.0,0.0,0.0\},\{1\text{-}d_P,0,0,0,0,0,1\},0)$$

took about 2 [*hours*] and achieved a best member fitness of 0.784854. Those two runs were mainly made to validate the correct working of the genetic algorithm. The problem of finding the identity is not as trivial as it looks, because the genetic algorithm has no visual information, but only the evaluation values which are based on the Pixset distance d_P.

Each time the algorithm is run, it creates two files: `monitor.dat` and `filename.aut`. The first file contains all chromosomes and their assigned fitnesses for generation 0 (initialization) and the last generation. From the other generations only the best members are default put in this file. Optional (by setting the global variable *extended* = 1 in the file `all.h`) all generations are completely logged and also the specific parent and operator selections as well as the resulting childs. Also default, some other data, like operator probabilities and the amount of operator selections is logged. In appendix D a typical result of a genetic run is given. The file `filename.aut` contains the affine transformations which are created from the best members of the last generation of each while loop (see fig. 11.1).

The main problem of this early version was its time consuming member evaluation of about 9.5 [*seconds*] per member. Despite the fact that we are forced to run through all pixels of *target* and *member*, we have been able to speed up the evaluation time with a factor 3.5, thus resulting in an average member evaluation time of 2.7 [*seconds*]. This speeding up was mainly achieved by improving the data handling within and in between the pixmaps and by combining the rendering and pixel counting processes. Also, the search area for target pixels has been limited from the whole pixmap to the bounding box of *target*.

We have first tasked this improved version to find the identity of some images. Even for population sizes of 100 members over 15 generations the best member fitness did not get better than about 0.83. By displaying the successive generated members on screen, we were able to monitor the progress in the state space search. This real time monitoring takes about 50% more CPU time, but made clear that there was need for a scaling operator. Therefore we designed the opertor *Scale*. An initialization with the genetic parameter setting:

$$GPS_3 = (1,15,75,0.1,\{0.2,0.2,0.2,0.15,0.15,0.1,0.0,0.0\},\{1\text{-}d_P,0,0,0,0,0,1\},0)$$

resulted in a best member fitness of 0.907376. Additional runs, also for other seeds of the random number generator, showed that the best member fitness did not get better than about 0.90. Again the state space search process was monitored and it became clear that a rotation operator was needed. So we designed and implemented the operator *Rotate*. We tasked the algorithm to find the identity of some of the images as given in appendix C. Each time the algorithm was initialized with the following parameter setting:

$$GPS_4 = (1,25,S,0.1,\{0.2,0.2,0.2,0.1,0.1,0.1,0.1,0.0\},\{1\text{-}d_p,0,0,0,0,0,1\},0)$$

The results for different population sizes S are given in table 11.2. It is obvious that the searched for identity mapping has an IFS-code of {1,0,0,1,0,0}, but for symmetric structures, such as *rect* or *dragon* (see appendix C), also other transformations than the identity are valid. In particular all transformations belonging to the symmetry group of the structure. For example, the symmetry group of a square (or the structure *rect*) contains 8 transformations (see table 11.1):

i) ι = identity, IFS-code {1,0,0,1,0,0};
ii) ρ_1 = rotation over 90 [*degrees*], IFS-code {0,-1,1,0,0,0};
iii) ρ_2 = rotation over 180 [*degrees*], IFS-code {-1,0,0,-1,0,0};
iv) ρ_3 = rotation over 270 [*degrees*], IFS-code {0,1,-1,0,0,0};
v) σ_1 = reflection through x=0, IFS-code {-1,0,0,1,0,0};
vi) σ_2 = reflection through y=0, IFS-code {1,0,0,-1,0,0};
vii) σ_3 = reflection through y=$-x$, IFS-code {0,-1,-1,0,0,0};
viii) σ_4 = reflection through y=x, IFS-code {0,1,1,0,0,0}.

Table 11.1. Cayley table for symmetry group of the square for the composition operator.

$\circ$	ι	ρ_1	ρ_2	ρ_3	σ_1	σ_2	σ_3	σ_4
ι	ι	ρ_1	ρ_2	ρ_3	σ_1	σ_2	σ_3	σ_4
ρ_1	ρ_1	ρ_2	ρ_3	ι	σ_3	σ_4	σ_2	σ_1
ρ_2	ρ_2	ρ_3	ι	ρ_1	σ_2	σ_1	σ_4	σ_3
ρ_3	ρ_3	ι	ρ_1	ρ_2	σ_4	σ_3	σ_1	σ_2
σ_1	σ_1	σ_4	σ_2	σ_3	ι	ρ_2	ρ_3	ρ_1
σ_2	σ_2	σ_3	σ_1	σ_4	ρ_2	ι	ρ_1	ρ_3
σ_3	σ_3	σ_1	σ_4	σ_2	ρ_1	ρ_3	ι	ρ_2
σ_4	σ_4	σ_2	σ_3	σ_1	ρ_3	ρ_1	ρ_2	ι

As a result from the above considerations, we will let the genetic algorithm search for any valid transformation instead of just the identity. For clearity, table 11.2 also contains the found IFS-codes (i.e. the decoded chromome of the best member of each last generation). In fig. 11.14 the graphs of the best members from the structure *fern* are given. From these figures we see that the fitness is strongly related to the visual closeness, i.e. the better the fitness the better the visual closeness of *target* and *member* (see appendix C). Before we are going to comment on the results of table 11.2, we have to explain some more about its contents. Although we have speeded up the evaluation time, the whole process is still time consuming. This is a bottle neck in obtaining enough data to apply a reliable statistical analysis. Since the problem of finding a symmetry transformation is only relevant to validate the correct working and the intrinsic problem solving power of the genetic algorithm, we have choosen a different approach. All runs which are tabulated in table 11.2 have been done again

Table 11.2. Results of symmetry transformation search for some raster images of appendix C for GPS_4 (time in [*hours*]).

image	S	fitness	chromosome	IFS-code	time
dragon	10	0.756734	(124,11,9,127,0,139)	0.976 0.087 0.071 1.000 0.000 -0.087	0.17
	25	0.859271	(253,131,6,256,129,132)	-0.984 -0.024 0.047 -1.008 -0.008 -0.031	0.64
	50	0.897574	(127,7,1,127,129,129)	1.000 0.055 0.008 1.000 -1.969 -1.969	1.04
	100	0.899707	(125,2,132,127,128,128)	0.984 0.016 -0.031 1.000 0.000 0.000	3.42
fern	10	0.560379	(109,11,111,255,134,148)	0.858 0.087 0.874 -1.000 -0.047 -0.157	0.14
	25	0.774714	(127,134,10,125,3,129)	1.000 -0.047 0.079 0.984 5.906 -1.969	0.38
	50	0.780387	(127,132,145,125,129,2)	1.000 -0.031 -0.134 0.984 -0.008 0.016	0.93
	100	0.836296	(127,131,132,125,129,128)	1.000 -0.024 -0.031 0.984 -1.969 0.000	1.39
france	10	0.882267	(125,136,16,124,128,8)	0.984 -0.063 0.126 0.976 0.000 0.063	0.23
	25	0.898903	(127,2,147,127,2,136)	1.000 0.016 -0.150 1.000 0.016 -0.063	0.46
	50	0.923906	(127,144,1,127,130,128)	1.000 -0.126 0.008 1.000 -3.937 0.000	1.23
	100	0.962674	(126,128,133,127,128,128)	0.992 0.000 -0.039 1.000 0.000 0.000	2.47
leaf	10	0.697666	(162,243,120,193,128,128)	-0.268 -0.906 0.945 -0.512 0.000 0.000	0.19
	25	0.841518	(127,2,133,111,129,130)	1.000 0.016 -0.039 0.874 -1.969 -3.937	0.62
	50	0.868127	(254,72,6,121,7,132)	-0.992 0.567 0.047 0.953 13.780 -7.874	1.13
	100	0.946545	(127,131,2,124,128,129)	1.000 -0.024 0.016 0.976 0.000 -1.969	2.03
mandel	10	0.760231	(119,180,141,253,136,130)	0.937 -0.409 -0.102 -0.984 -0.063 -0.016	0.22
	25	0.896908	(127,2,137,125,128,6)	1.000 0.016 -0.071 0.984 0.000 0.047	0.41
	50	0.902132	(127,128,138,127,3,4)	1.000 0.000 -0.079 1.000 5.906 7.874	1.05
	100	0.905018	(127,131,12,126,1,2)	1.000 -0.024 0.094 0.992 1.969 3.937	1.64
rect	10	0.846164	(10,255,126,17,6,10)	0.079 -1.000 0.992 0.134 0.047 0.079	0.16
	25	0.879834	(127,130,20,125,128,139)	1.000 -0.016 0.157 0.984 0.000 -0.087	0.35
	50	0.931589	(130,254,254,137,131,131)	-0.016 -0.992 -0.992 -0.071 -0.024 -0.024	1.06
	100	0.987552	(127,129,1,127,1,128)	1.000 -0.008 0.008 1.000 0.008 0.000	1.83

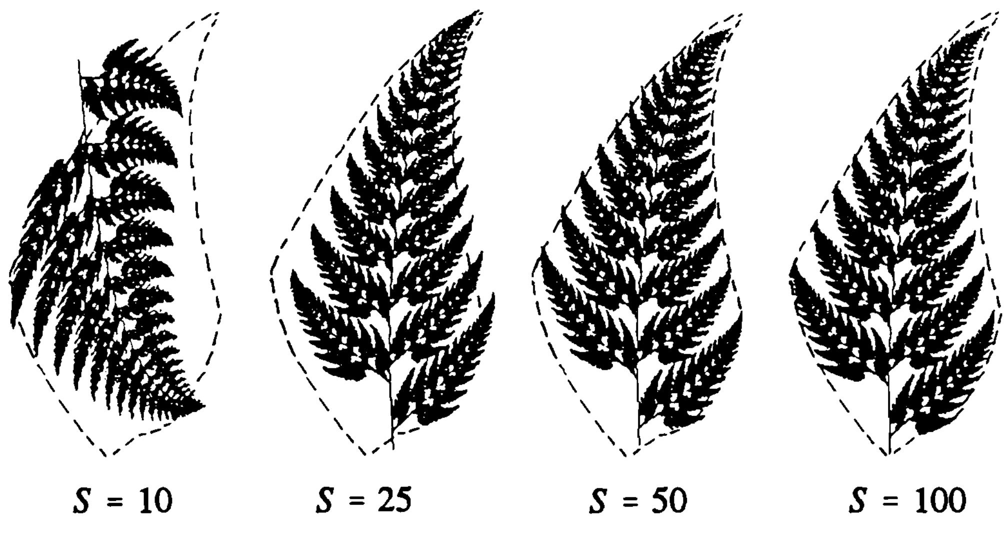


Figure 11.14. Visualized results of table 11.2 for the structure *fern* (dotted borderline corresponds to target image).

for different seedings of the random number generator, which is involved in the parent and operator selection processes. These seedings were obtained from the UNIX™ timing module, which keeps track of the amount of seconds that have passed since 1970. The best fitness obtained with runs on different seedings were about in the same range as those tabulated in table 11.2. From this we can conclude that the developed genetic algorithm is powerful enough to find a good approximate for an affine transformation for (one of the) symmetry transformations of a raster image in $O(n)$ time, where n denotes the amount of pixels within the bounding box of the image.

Definition 11.6. The ***performance graph*** of a genetic algorithm is a graph of the fitness of the best member of a generation as function of the number g of that generation. In other words, a graph of the evaluation function $g \rightarrow E(best\ member(g))$ for $0 \leq g \leq G$. ♦

In our case a performance graph is a typical non-declining function, since the best member fitness in two successive generations can not get worse, because the best 10% of a generation will be transferred to the next generation. In fig. 11.15 the performance graphs of the genetic algorithm are given for the task of finding transformations belonging to the symmetry groups of the structures from table 11.2. The plotted values are also tabulated in table 11.3. We have plotted four graphs, each for a different population size S. Genetic algorithms are stochastic, which implies that their performance usually varies from run to run and thus a curve showing the behaviour of a single run doesn't supply us with much information. Therefore we took the average performance of the algorithm on the six structures of table 11.2. Although these structures are different, finding transformations of their symmetry groups belongs to the same class of problems, and therefore it is allowed to take the average. Since the genetic algorithm has no visual information available and the state space has a lot of local optima (for example we get a strong oscillating pixsetdistance when convoluting two identical images of a fern), the obtained results must be classified as satisfying, since it appears to be possible to find a 99% best case approximation within polynomial time (see last line of table 11.2).

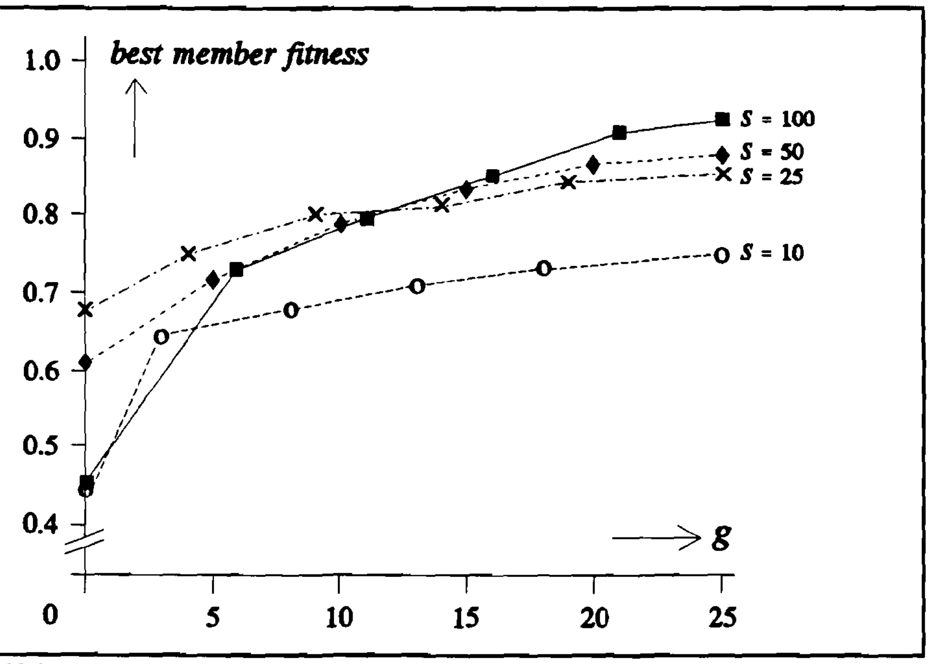


Figure 11.15. Performance graph of genetic algorithm for the structures of table 11.2.

Table 11.3. Average best member fitnesses for generation *g*.

g	*S* = 10	*S* = 25	*S* = 50	*S* = 100
0	0.459020	0.688917	0.614644	0.465150
1	0.568170	0.735361	0.662815	0.630836
2	0.584604	0.735361	0.666538	0.651159
3	0.642196	0.740391	0.682728	0.690090
4	0.658090	0.747115	0.715669	0.695425
5	0.670982	0.750775	0.724533	0.702619
6	0.670982	0.763626	0.742966	0.730205
7	0.680799	0.763626	0.747261	0.741330
8	0.685975	0.775617	0.760759	0.778864
9	0.694192	0.803186	0.781674	0.781044
10	0.694192	0.811193	0.790659	0.786643
11	0.703241	0.814121	0.796624	0.798677
12	0.708475	0.817182	0.799066	0.802216
13	0.710768	0.818466	0.817631	0.811568
14	0.714144	0.818466	0.825628	0.830453
15	0.714981	0.823499	0.832908	0.836047
16	0.723983	0.825582	0.838229	0.856836
17	0.723983	0.840374	0.842677	0.873441
18	0.732532	0.841276	0.849532	0.878968
19	0.734884	0.843150	0.860350	0.882357
20	0.739376	0.844854	0.863135	0.909210
21	0.748661	0.850121	0.867863	0.909907
22	0.749784	0.850146	0.871945	0.912160
23	0.750074	0.856041	0.875738	0.912549
24	0.750574	0.856861	0.878970	0.915981
25	0.750574	0.858525	0.883953	0.922965

From fig. 11.15 we conclude that a population size of 10 or 25 members is too small, but a population size of 100 members implies a lot of redundant calculations, since the result is hardly any better than with a population size of 50 members. Thus from now on we will fix the population size to 50 members, unless stated otherwise.

Now we are ready to run the genetic algorithm on the IFS-encoding problem, i.e. to let it search for a valid collage. Due to time constraints, we have only focussed on trying to find an IFS-code for the structure *france* (see appendix C). We have choosen for this structure because it is the most difficult one to encode into an IFS. The reason is the fact that the manually obtained (and thus very efficient) IFS-code, already contains 31 affine transformations, while the IFS-codes of the other structures in appendix C contain at most 9 affine transformations. Therefore we may carefully state that whenever the genetic algorithm is powerful enough to find an IFS-code for the structure *france* it must be able to find IFS-codes for all other structures.

Before we put the algorithm at work, we must be concerned about the evaluation function for the first member we put in the collage, because if we take no special precautions, the algorithm will start hunting for a symmetry group transformation, since this is the best first transformation it can find. In an empirical way we obtained as a satisfying parameter set for the evaluation of the first transformation $K = \{1,0,0,1,1,0,2\}$. Basically, this evaluation function searches for copies which are half the size of the target image and entirely contained within this target image. In fig. 11.16 some typical first found transformations are given for the structure *france*.

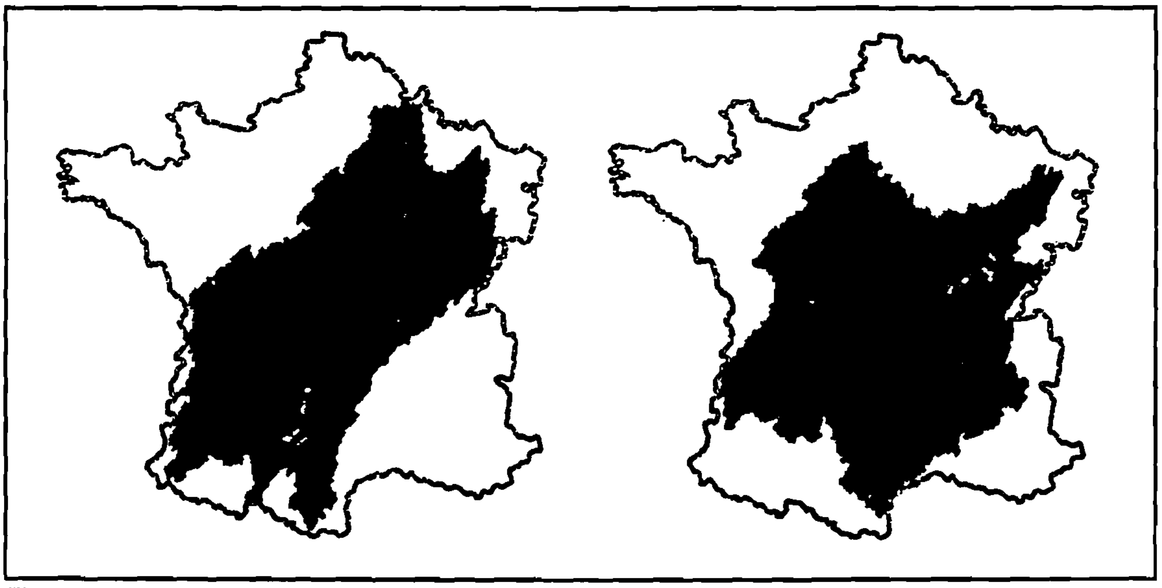


Figure 11.16. Some typical first collage tiles for the structure *france*, found by the genetic algorithm.

From the second tile and onwards, we need a different evaluation function parameter set. Since we cannot predict the outcome of the genetic algorithm we had to use a trial-and-error method to tune these parameters. First we tried the genetic parameter setting:

$$GPS_5 = (5,15,16,0.1,\{0.2,0.2,0.2,0.1,0.1,0.1,0.1,0.0\},\{0,1,0,1,2,0,2\},0.05)$$

The result is given in fig. 11.17. As we can see from this figure, the problem is that we get very long and flat tiles which exceed the target image boundary. The first property gives rise to an IFS-code which can not be decoded into the target image. We can understand this as follows. Suppose we encode a circle by tiling it with ellipses such that the center of all ellipses is equal to the center of the circle. If the circle is fully covered then this would certainly be a valid collage. However, if we want to decode it, we never get a circle, but instead, we get a bunch of pixels around the center of the circle. This is due to the fact that all transformations have the same fixed point. Thus an additional requirement for a valid collage should be added.

Theorem 11.2. The collage theorem (see theorem 10.1) is only valid if the fixed points of all transformations in the collage (see definition 8.2) are different.

Proof. Suppose we define a collage, such that the fixed points of all transformations it contains are the same, then clearly the attractor of the according IFS equals that fixed point and not the tiled target image. ■

Although we have not been able to find any literature where this problem is mentioned, it is clear that it is a real and severe problem in the automatic IFS-encoding process. In addition to the above

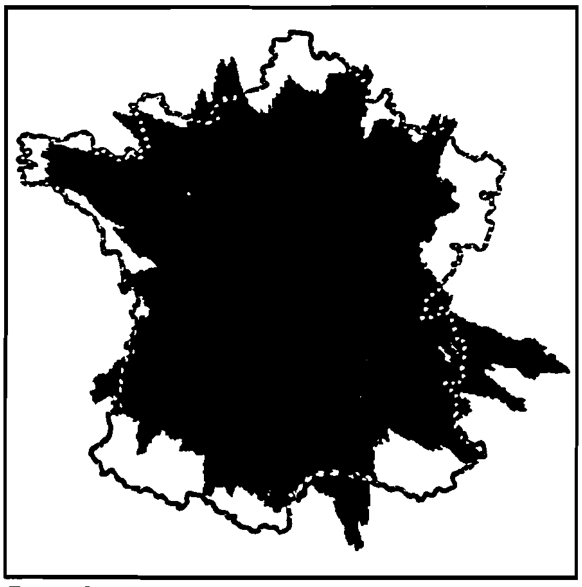


Figure 11.17. Result for GPS_5.

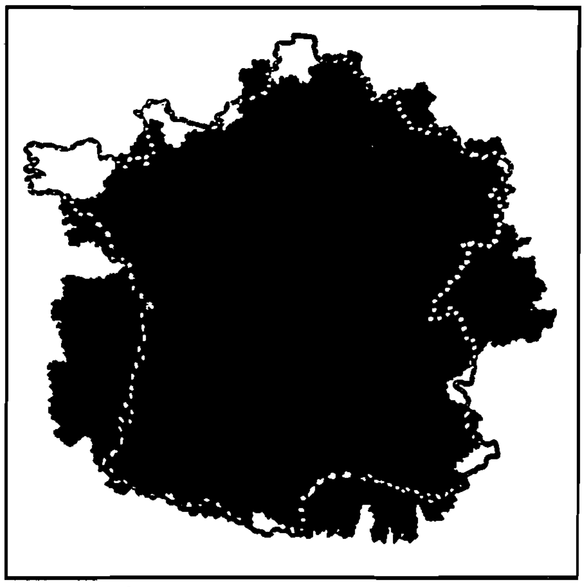


Figure 11.18. Result for GPS_6.

theorem, we can state that the further the fixed points of the separate transformations in the collage are away from each other, the faster the decoding process goes, since the attractor boundary will be reached earlier. By analyzing the progress of the state space search by real time monitoring the successive generated population members, we came to the conclusion that a translation operator was needed to tackle this mutual fixed point problem. This is because it was observed that the further the algorithm proceeded, the more the members were placed in the center of the target image. In some cases, any of the 7 operators implemented so far, would yield a worse evaluation value, except for a translation operator. Therefore we designed and implemented the operator *Translate*. We initialized the algorithm with the following genetic parameter setting:

$$GPS_6 = (6,25,100,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{0,1,0,1,2,0,2\},0.02)$$

The result with this new added operator, the redefined operator probabilities and the same evaluation parameter set is given in fig. 11.18. Despite the fact that GPS_6 yields a more spread out collage, which is good, the problem that the collage exceeds the boundary of the target, still remains. So we decided to attach more weight to the penalty which guards exceeding of the target image border (*SP*). We initialized the genetic algorithm with the genetic parameter setting:

$$GPS_7 = (10,25,100,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{8,1,0,1,8,0,10\},0.02)$$

As we can see from fig. 11.19, the result of GPS_7 is better than of GPS_6, as far as the exceeding of the boundary of the target image is concerned. But still the boundary of the target image is exceeded. Also we see that GPS_7 gives rise to very long and flat structures, from which we must conclude that each new tile covers a relatively great part of the collage so far. This is something we do not want because of theorem 11.2, but also because of efficiency considerations. Therefore we decided to adjust the gain of the penalty which guards this specific property. We therefore came up with the following genetic parameter setting:

$$GPS_8 = (5,15,20,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{0,6,0,4,99,6,10\},0.02)$$

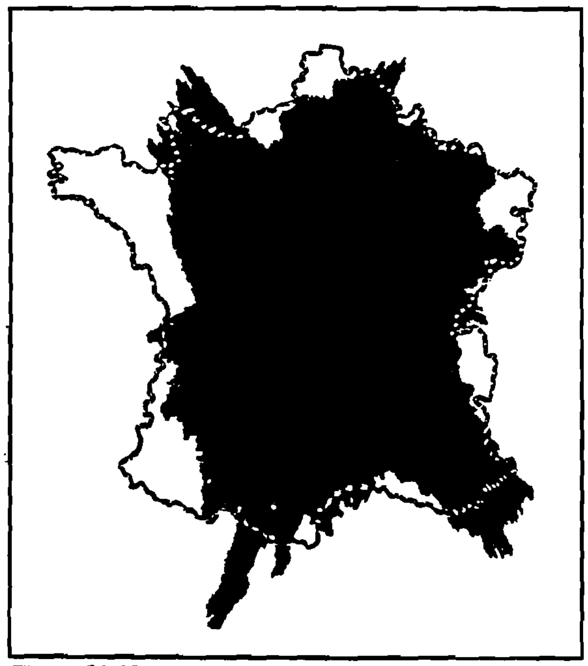


Figure 11.19. Result for GPS_7.

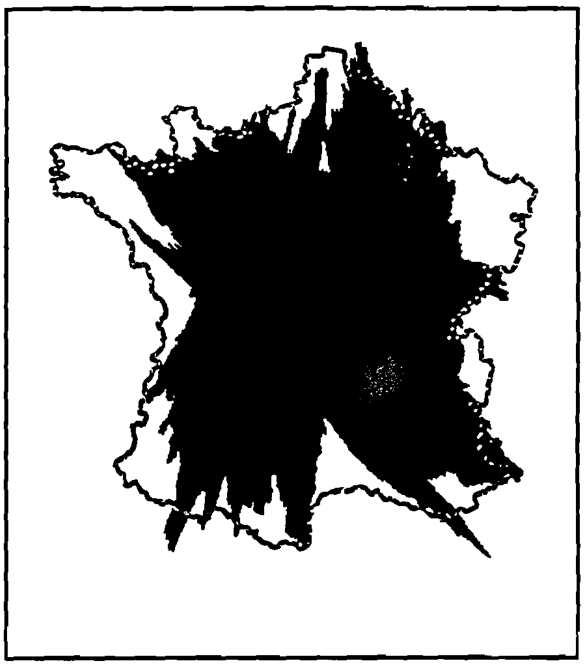


Figure 11.20. Result of GPS_8.

The outcome of GPS_8 is given in fig. 11.20, from which we can see that the result is still unsatisfactory. From the outputfile `monitor.dat` we learned that the reason for this phenomenon was the fact that from transformation 3 and onwards only the best 15% of the members from each last generation had a fitness which was significantly greater than 0.000001. So 85% of the population had a fitness of 0.000001 due to the relatively strong gain of parameter k_4. This means that the diversity of the population is very restricted, because each member has a chance of being choosen which is directly proportional to its fitness. Concluding, we can state that this evaluation function induces an "overshoot" because of the gain of the guard *SP* (see definition 11.3). We therefore adjusted the parameters of the evaluation function once more and tried the genetic parameter setting:

$$GPS_9 = (7,15,50,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{0,1,0,1,10,1,2\},0.02)$$

The results of GPS_9 are given in fig. 11.21 from which we conclude that the guard *WP* leads to a "noisy" evaluation function, which confuses the algorithm. This is because it wants to cover as much of the uncovered area as possible, so it produces tiles which cross the collage and cover uncovered area's on both sides of it. Therefore we changed our strategy drastically. Not only was parameter k_5 set to zero, but also we choose for an adaptive evaluation function. We modified the guard *SR* from a static one (i.e. $N=1,q=1$) into an adaptive one as described in par. 11.4 (the fact that the description is already covered in par. 11.4 does not imply that we have used this strategy from the beginning). In this way we were able to get flexible control over the size of the tiles, which we want to decrease as the collage contains more and more tiles. The reason for this is of course that the uncovered areas become smaller and smaller, so we can encode more effective (more accurate), but also more efficient (less overlap). The other main adjustment we made on the evaluation function was to introduce a variable parameter set. Some of the parameters k_i were tied to the current transformation number N and were thus made adaptive to the encoding process. Finally we obtained the evaluation parameter set $K = \{0,1,0,N,N+1,0,N+1\}$. Next we initialized the algorithm with the genetic parameter setting:

$$GPS_{10} = (12,25,50,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{0,1,0,N,N+1,0,N+1\},0.02)$$

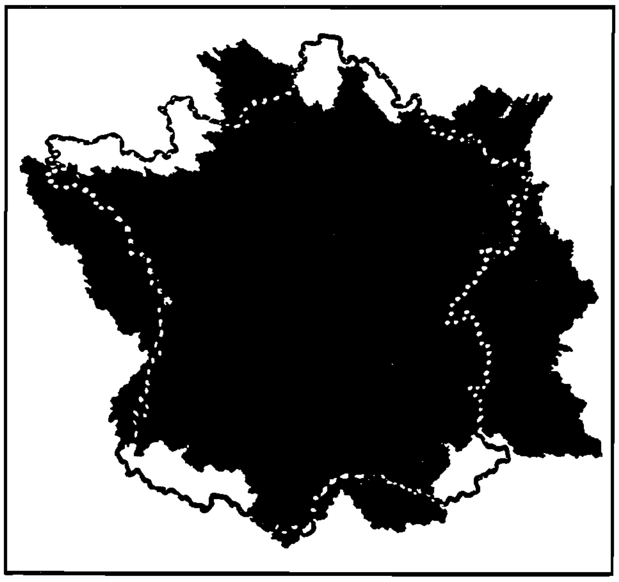


Figure 11.21. Result for GPS_9.

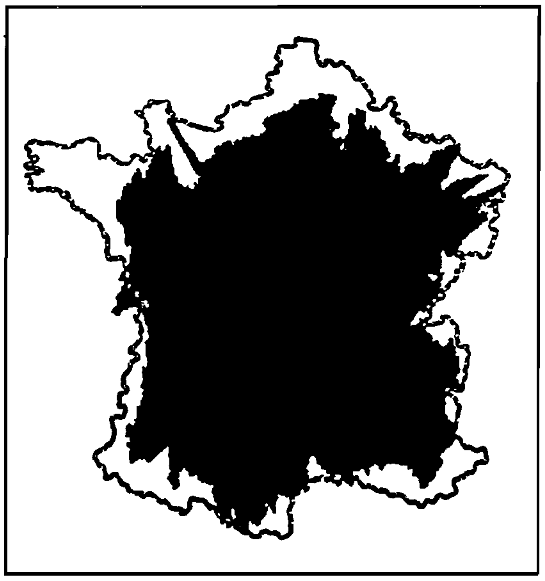


Figure 11.22. Result of GPS_{10}, $d_p = 0.231388$.

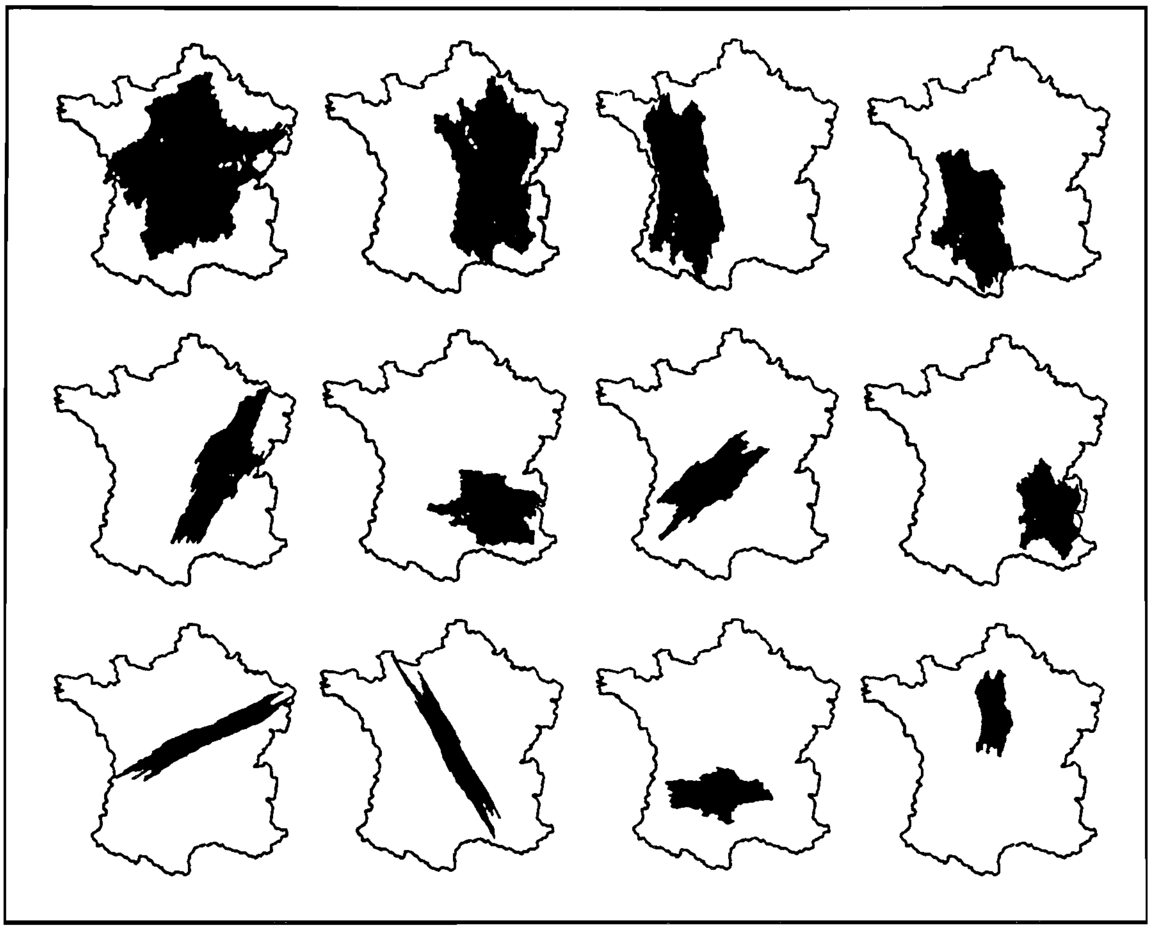


Figure 11.23. GPS_{10}: sizes of successive tiles of the collage decrease proportionally to the transformation number *N*.

The result of GPS_{10} is given in fig. 11.22 from which we conclude that all transformations lay within the target image boundary. Since this is the first image where the target boundary is not exceeded, it is also the first image about which it makes sense to assign a quality in terms of the Pixset distance. For GPS_{10} the Pixset distance yields d_P = 0.231388 which indicates an approximation quality of 77%. The successive transformations the algorithm found are given in fig. 11.23, from which we see that the successive tile sizes decrease proportionally to the transformation number N (from 1 upper left to 12 lower right). From this result we may expect that if we use the genetic parameter setting of GPS_{10} with an increased amount of transformations, the collage will get better. Therefore we initialized the algorithm with a different seed and the genetic parameter setting:

$$GPS_{11} = (17,25,50,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{0,1,0,N,N+1,0,N+1\},0.02)$$

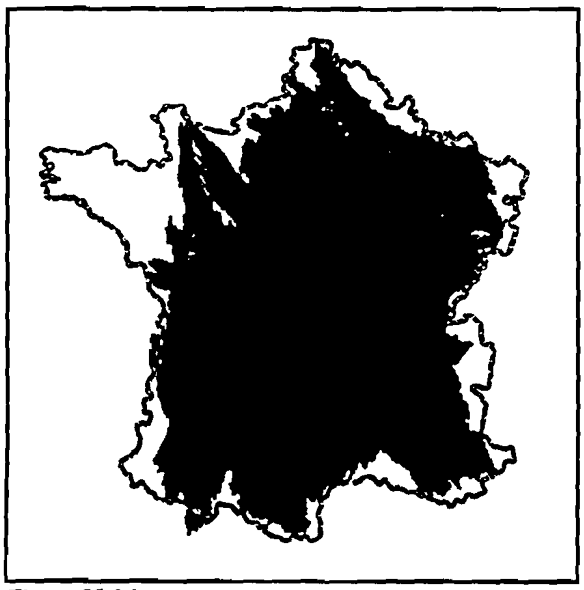


Figure 11.24. Result for GPS_{11}, d_P = 0.194123.

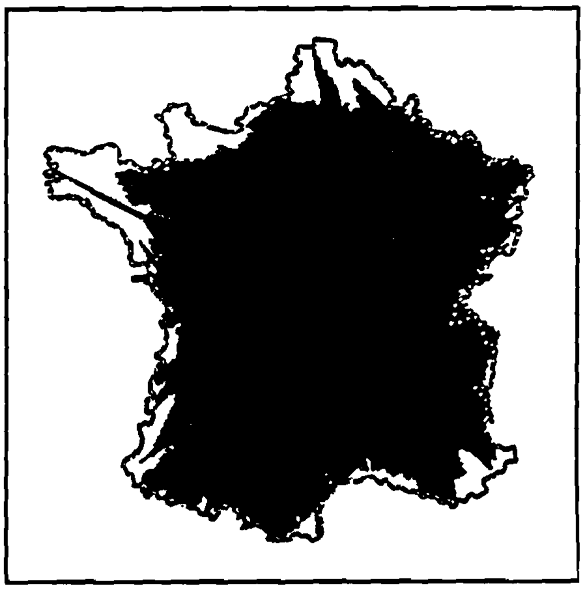


Figure 11.25. Result for GPS_{12}, d_P = 0.124390.

The result of GPS_{11} is given in fig. 11.24, from which we conclude that the increased amount of transformations from 12 in GPS_{10} to 17 in GPS_{11} results in a decrease of the Pixset distance from d_P = 0.231388 to d_P = 0.194123, which implies an increase in approximation quality from 77% to 81%. From this we may expect that a further increase of N will also increase the approximation quality. Therefore we initialized the algorithm with a different seed and the genetic parameter setting:

$$GPS_{12} = (50,25,50,0.1,\{0.15,0.15,0.15,0.08,0.11,0.12,0.12,0.12\},\{0,1,0,N,N+1,0,N+1\},0.02)$$

The result of GPS_{12} is given in fig. 11.25. We see that a further increase of N from 17 to 50 results in a decreasing Pixset distance from d_P = 0.194123 to d_P = 0.124390 and thus in an increasing approximation quality from 81% to 88%. Due to time constraints we have not been able to perform successive runs. It must be noticed that a complete run for GPS_{12} takes about 58 [*hours*] CPU-time. Thus a major concern for following research must be to further speed up the evaluation time. However, the result must be considered as satisfactory as far as the control over the process is concerned. After all we have been able to reach an approximation quality of 88% with 1.6 times as much transformations than were needed for the (most efficient) interactive encoding of the (most difficult) structure *france*. One must not forget that the results are obtained with a "blind" system. No attempts at all have been made to apply any pattern recognition techniques.

In tables 11.4 to 11.6 we have tabulated some intermediate results about the genetic runs which were initialized with respectively GPS_{10}, GPS_{11} and GPS_{12}. From this we can see that the algorithm is able to keep finding good fitnesses, even for high transformation numbers. This is important to notice, because it indicates that the algorithm does not get stuck in local optima. Also we can conclude that the first transformation is found within a few generations, while the following transformations all need the full amount of 25 generations, except for the last few ones. An explanation about those three tables must be given as to the parameter G. In each of the cases the maximum amount of generations was set to $G = 25$, but because the maximum encoding error was set to $\epsilon = 0.02$, in some cases the full amount of generations were not needed. As soon as a member fitness of $1-\epsilon$ or more was found, then the result was satisfactory for that transformation. In the first column i denotes the transformation number. In the second column G denotes the amount of generations. We conclude that we get better than average initial populations ($g = 0$), due to our inverse fixed point method (see par. 11.5).

Table 11.4. Intermediate results of GPS_{10} (CPU-time: 13.8 [*hours*]).

i	G	best member fitness		chromosome	affine transformation
		$g = 0$	$g = G$		
1	7	0.734840	0.981560	(1,199,47,199,192,133,6)	-0.559 0.370 -0.559 -0.504 -9.843 11.811
2	25	0.628592	0.773593	(55,129,129,97,24,129)	0.433 -0.008 -0.008 0.764 47.244 -1.969
3	25	0.726888	0.833530	(40,17,8,226,166,139)	0.315 0.134 0.063 -0.772 -74.803 -21.654
4	25	0.696179	0.830470	(45,7,146,200,152,161)	0.354 0.055 -0.142 -0.567 -47.244 -64.961
5	25	0.614980	0.849640	(177,147,174,198,25,130)	-0.386 -0.150 -0.362 -0.551 49.213 -3.937
6	25	0.663673	0.877555	(56,141,148,163,27,166)	0.441 -0.102 -0.157 -0.276 53.150 -74.803
7	25	0.680854	0.880781	(58,17,47,147,144,144)	0.457 0.134 0.370 -0.150 -31.496 -31.496
8	25	0.671484	0.897895	(9,161,48,21,41,161)	0.071 -0.260 0.378 0.165 80.709 -64.961
9	25	0.650132	0.914799	(7,219,19,170,14,16)	0.055 -0.717 0.150 -0.331 27.559 31.496
10	25	0.667832	0.911150	(55,136,222,140,130,130)	0.433 -0.063 -0.740 -0.094 -3.937 -3.937
11	8	0.610407	0.981027	(164,172,21,143,140,162)	-0.283 -0.346 0.165 -0.118 -23.622 -66.929
12	5	0.652295	0.994165	(20,6,12,172,131,32)	0.157 0.047 0.094 -0.346 -5.906 62.992

Table 11.5. Intermediate results of GPS_{11} (CPU-time: 19.6 [*hours*]).

i	G	best member fitness		chromosome	affine transformation
		$g = 0$	$g = G$		
1	1	0.689007	0.999999	(16,19,144,18,128,2)	0.126 0.150 -0.126 0.142 0.000 3.937
2	25	0.694666	0.963624	(147,185,213,26,2,128)	-0.150 -0.449 -0.669 0.205 3.937 0.000
3	25	0.850848	0.877129	(128,165,236,1,164,147)	0.000 -0.291 -0.850 0.008 -70.866 -37.402
4	25	0.653063	0.925333	(135,193,176,31,25,48)	-0.055 -0.512 -0.378 0.244 49.213 94.488
5	25	0.653169	0.912883	(51,15,147,184,31,160)	0.402 0.118 -0.150 -0.441 61.024 -62.992
6	25	0.636166	0.887283	(173,2,189,176,14,154)	-0.354 0.016 -0.480 -0.378 27.559 -51.181
7	25	0.702181	0.896465	(178,184,78,50,2,5)	-0.394 -0.441 0.614 0.394 3.937 9.843
8	25	0.706789	0.897279	(153,14,141,191,136,30)	-0.197 0.110 -0.102 -0.496 -15.748 59.055
9	25	0.645808	0.907286	(47,161,59,136,30,135)	0.370 -0.260 0.465 -0.063 59.055 -13.780
10	25	0.643598	0.911141	(46,15,1,31,25,157)	0.362 0.118 0.008 0.244 49.213 -57.087
11	25	0.624137	0.924444	(22,67,12,149,142,3)	0.173 0.528 0.094 -0.165 -27.559 5.906
12	25	0.715309	0.939591	(25,146,161,150,144,190)	0.197 -0.142 -0.26 -0.173 -31.496 -122.047
13	25	0.741563	0.929956	(131,18,58,22,133,130)	-0.024 0.142 0.457 0.173 -9.843 -3.937
14	25	0.683205	0.945376	(150,147,130,173,170,170)	-0.173 -0.15 -0.016 -0.354 -82.677 -82.677
15	25	0.632042	0.942288	(2,21,174,148,43,155)	0.016 0.165 -0.362 -0.157 84.646 -53.150
16	25	0.705903	0.949995	(14,14,163,20,149,128)	0.110 0.110 -0.276 0.157 -41.339 0.000
17	25	0.556917	0.949913	(13,14,168,18,2,2)	0.102 0.110 -0.315 0.142 3.937 3.937

Table 11.6. Intermediate results of GPS_{12} (CPU-time: 57.6 [*hours*]).

i	G	best member fitness		chromosome	affine transformation
		$g = 0$	$g = G$		
1	22	0.628568	0.987870	(27,219,199,189,128,152)	0.213 -0.717 -0.559 -0.480 0.000 -47.244
2	25	0.696261	0.855267	(245,128,1,47,132,32)	-0.921 0.000 0.008 0.370 -7,874 62.992
3	25	0.621325	0.795060	(146,197,170,63,15,149)	-0.142 -0.543 -0.331 0.496 29.528 -41.339
4	25	0.723438	0.819435	(168,59,30,35,141,46)	-0.315 0.465 0.236 0.276 -25.591 90.551
5	25	0.710251	0.838349	(157,26,183,168,135,3)	-0.228 0.205 -0.433 -0.315 -13.780 5.906
6	25	0.662953	0.867403	(50,57,59,22,25,128)	0.394 0.449 0.465 0.173 49.213 0.000
7	25	0.694442	0.884074	(141,59,173,55,152,130)	-0.102 0.465 -0.354 0.433 -47.244 -3.937
8	25	0.726687	0.890841	(37,140,28,39,31,31)	0.291 -0.094 0.220 0.307 61.024 61.024
9	25	0.619281	0.900479	(143,28,59,132,136,144)	-0.118 0.220 0.465 -0.031 -15.748 -31.496
10	25	0.728897	0.909925	(32,24,157,23,154,144)	0.252 0.189 -0.228 0.181 -51.181 -31.496
11	25	0.732075	0.919191	(168,182,177,163,9,129)	-0.315 -0.425 -0.386 -0.276 17.717 -1.969
12	25	0.696687	0.930419	(145,142,23,84,128,128)	-0.314 -0.110 0.181 0.661 0.000 0.000
13	25	0.619281	0.929861	(22,28,169,130,136,128)	0.173 0.220 -0.323 -0.016 -15.748 0.000
14	25	0.728897	0.939952	(16,132,170,175,138,152)	0.126 -0.031 -0.331 -0.370 -19.685 -47.244
15	25	0.732075	0.943619	(34,20,144,19,43,47)	0.268 0.157 -0.126 0.150 84.646 92.520
16	25	0.696687	0.950136	(156,77,128,155,136,128)	-0.220 0.606 0.000 -0.213 -15.748 0.000
17	25	0.619281	0.953109	(156,129,3,28,38,180)	-0.220 -0.008 0.024 0.220 74.803 -102.362
18	25	0.728897	0.949865	(183,142,144,10,136,136)	-0.433 -0.110 -0.126 0.079 -15.748 -15.748
19	25	0.732075	0.949854	(161,10,135,150,4,1)	-0.260 0.079 -0.055 -0.173 7.874 1.969
20	25	0.696687	0.960482	(128,148,160,128,27,174)	0.000 -0.157 -0.252 0.000 53.150 -90.551
21	25	0.619281	0.960513	(24,153,156,54,168,152)	0.189 -0.197 -0.220 0.425 -78.740 -47.244
22	25	0.728897	0.959826	(168,10,151,21,138,144)	-0.315 0.079 -0.181 0.165 -19.685 -31.496
23	25	0.732075	0.960612	(180,165,164,165,152,152)	-0.41 -0.291 -0.283 -0.291 -47.244 -47.244
24	25	0.696687	0.959944	(3,187,9,25,128,131)	0.024 -0.465 0.071 0.197 0.000 -5.906
25	25	0.619281	0.974438	(14,30,155,151,38,58)	0.110 0.236 -0.213 -0.181 74.803 114.173
26	25	0.728897	0.969763	(156,8,173,131,22,2)	-0.220 0.063 -0.354 -0.024 43.307 3.937
27	25	0.732075	0.970202	(21,153,130,24,51,42)	0.165 -0.197 -0.016 0.189 100.394 82.677
28	25	0.696687	0.970406	(133,5,85,128,137,149)	-0.039 0.039 0.669 0.000 -17.717 -41.339
29	25	0.619281	0.969964	(8,10,160,16,17,41)	0.063 0.079 -0.252 0.126 33.465 80.709
30	25	0.728897	0.969988	(165,146,131,142,2,2)	-0.291 -0.142 -0.024 -0.110 3.937 3.937
31	25	0.732075	0.974855	(144,129,45,31,8,57)	-0.126 -0.008 0.354 0.244 15.748 112.205
32	25	0.696687	0.969976	(64,161,149,4,128,128)	0.504 -0.260 -0.165 0.031 0.000 0.000
33	6	0.619281	0.981267	(139,156,150,155,24,176)	-0.087 -0.220 -0.173 -0.213 47.244 -94.488
34	25	0.573731	0.979819	(135,20,8,20,12,39)	-0.055 0.157 0.063 0.157 23.622 76.772
35	25	0.643031	0.979972	(143,142,19,131,128,128)	-0.118 -0.110 0.150 -0.024 0.000 0.000
36	8	0.722575	0.982489	(65,72,164,164,157,13)	0.512 0.567 -0.283 -0.283 -57.087 25.591
37	25	0.737806	0.979937	((25,154,26,166,130,163)	0.197 -0.205 0.205 -0.299 -3.937 -68.898
38	19	0.680204	0.980226	(5,136,13,38,30,169)	0.039 -0.063 0.102 0.299 59.055 -80.709
39	25	0.762241	0.979984	(19,136,153,26,6,4)	0.150 -0.063 -0.197 0.205 11.811 7.874
40	15	0.722197	0.980621	(151,140,19,23,178,11)	-0.181 -0.094 0.150 0.181 -98.425 21.654
41	10	0.669912	0.980017	(1,24,13,17,17,58)	0.008 0.189 0.102 0.134 33.465 114.173
42	25	0.672086	0.979996	(139,139,144,11,148,151)	-0.087 -0.087 -0.126 0.087 -39.370 -45.276
43	25	0.639545	0.979960	(41,144,139,131,8,132)	0.323 -0.126 -0.087 -0.024 15.748 -7.874
44	25	0.704970	0.979996	(18,7,151,8,131,135)	0.142 0.055 -0.1810.063 -5.906 -13.780
45	25	0.582427	0.979996	(174,150,145,130,5,145)	-0.362 -0.173 -0.134 -0.016 9.843 -33.465
46	25	0.697455	0.979913	(133,145,19,4,139,12)	-0.039 -0.134 0.150 0.031 -21.654 23.622
47	25	0.662468	0.979925	(7,128,2,42,16,132)	0.055 0.000 0.016 0.331 31.496 -7.874
48	25	0.663780	0.979913	(9,26,142,135,19,137)	0.071 0.205 -0.110 -0.055 37.402 -17.717
49	10	0.621786	0.980278	(3,137,156,143,178,135)	0.024 -0.017 -0.220 -0.118 -98.425 -13.780
50	2	0.604417	0.988725	(10,19,140,131,180,27)	0.079 0.150 -0.094 -0.024 -102.362 53.150

From the fact that our genetic algorithm is able to find collages with an approximation error of $\epsilon = 0.124390$ (measured by the Pixset distance d_P), we must not conclude that the rendered attractor also approximates the target image with an approximation error of ϵ.

The collage theorem (theorem 10.1) states that a collage approximation error of at most ϵ implies an attractor approximation error of at most $\epsilon(1-s)^{-1}$, where s is the contractivity factor of the IFS. According to definition 8.1 the contractivity factor of the IFS is given by $s = (\,\underline{\text{Max}}\ i : 1 \le i \le N : s_i\,)$, where s_i is the contractivity factor of affine transformation w_i in the IFS. The contractivity factor of an affine transformation can be calculated as $s_i = |a_i d_i - b_i c_i|$. Table 11.7 contains the absolute determinant values for the affine transformations of table 11.6. From table 11.7 we obtain:

$$s = (\,\underline{\text{Max}}\ i : 1 \le i \le N : |a_i d_i - b_i c_i|\,) \approx 0.503043$$

From this we conclude that we can expect an attractor approximation error of at most:

$$\frac{\epsilon}{1-s} = \frac{0.124390}{1 - 0.503043} \approx 0.25$$

Rendering of the attractor of the IFS of table 11.6 yields fig. 11.26, from which we calculated:

$$d_P(france_f, \bigcup_{i=1}^{N} w_i(france)) \approx 0.107570$$

$$d_P(france, france_f) \approx 0.202205$$

From this we see that indeed the attractor approximation error is less than the predicted maximum error of 0.25. This is a nice result as far as the validity of d_P as distance measure is concerned, since we have not proven that it is allowed to replace h by d_P in theorem 10.1. Notice that the attractor (fig. 11.26) not only differs from the collage (fig. 11.25) with respect to the approximation quality of the target image, but also with respect to its own geometry, i.e. the attractor is a fractal with self similarity properties, while the collage is not.

In fig. 11.27 the determinant values of the successive affine transformations of table 11.7 are plotted against the transformation number i. Since the relative determinant value also gives a notion of the area of the transformed tile (relative to the target image), we can conclude that the tile size control part in our evaluation function is working very well (compare fig. 11.27 to fig. 11.12).

Table 11.7. Successive tile sizes s of found affine transformations by GPS_{12} according to table 11.6.

i	s	i	s	i	s	i	s	i	s
1	0.503043	11	0.077110	21	0.036985	31	0.027912	41	0.018206
2	0.340770	12	0.068664	22	0.037676	32	0.027276	42	0.018531
3	0.250165	13	0.068292	23	0.036666	33	0.019527	43	0.018714
4	0.196680	14	0.056881	24	0.037743	34	0.018526	44	0.018901
5	0.160585	15	0.059982	25	0.030358	35	0.019332	45	0.017390
6	0.140623	16	0.046860	26	0.027582	36	0.015568	46	0.018891
7	0.120444	17	0.048208	27	0.028033	37	0.016878	47	0.018205
8	0.110017	18	0.048067	28	0.026091	38	0.018087	48	0.018645
9	0.098642	19	0.049325	29	0.027846	39	0.018239	49	0.018425
10	0.088704	20	0.039564	30	0.028602	40	0.018661	50	0.012204

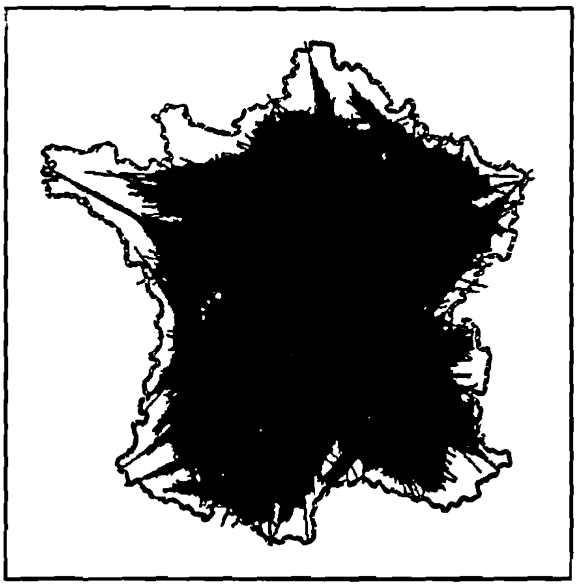


Figure 11.26. Rendered attractor france_f for GPS_{12}.

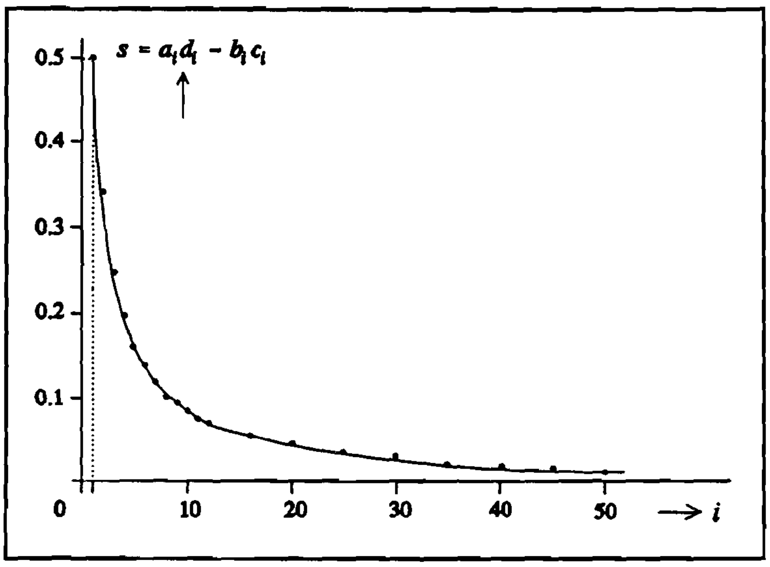


Figure 11.27. Graph of tilesizes s of GPS_{12} as function affine transformation i.

It is a capital mistake to theorize before one has data.

Sir Arthur Conan Doyle (1859 - 1930) [DAR80]

Chapter 12

Conclusions and recommendations

In this final chapter we will draw conclusions on the former chapters and also present some recommendations for further research. These recommendations are split up as to the subject on which they reflect, i.e. recommendations on the user interface, on the genetic algorithm and on IFS-encoding.

12.1. Conclusions

We have presented the concepts of chaos theory and fractal geometry as well as their relationship from a bottom up approach. We have therefore defined a dynamical system as a transformation on a metric space. This definition is extended to the concept of the Iterated Function System (IFS), of which a dynamical system is a special case. An IFS can be encoded by an IFS-code, which appears to be an extremely compact representation for the attractor of the IFS. An attractor can be looked upon as a (complex real world) image, but also as a Poincaré map of the behaviour of the system.

We have presented algorithms for efficient deterministic and nondeterministic IFS-decoding with and without condensation. The data compression performance of IFS-coding is considered, also in comparision with other encoding schemata. Furthermore it is shown that an IFS can be considered as a special case of a finite affine automaton.

We have seen that the attractor of a nondeterministic IFS is an invariant measure. Although the orbit of the system can not be predicted, with probability one the same attractor will appear when the Markov operator is recursively applied.

IFS-encoding is proven to be NP-complete. Despite its promising expectations, moment theory appears to be not suitable to solve the IFS-encoding problem in closed form (due to pattern recognition considerations). A practical try out of the moment approach to a simple structure, resulted in an approximation quality of 95% (measured by the Pixset distance). This shows that moment theory is suitable to act as a reliable quality measure for evaluating candidate solutions. However, the analytical way in which this result was obtained, had nothing to do with automatic IFS-encoding.

Although the Hausdorff distance is known to be a reliable quality measure for evaluating candidate solutions, it has a practical drawback, i.e. it has a computational time complexity of $O(n^2)$, where n denotes the amount of pixels in the target image. We developed the Pixset distance as useful alternative and proved that it is indeed a metric.

As far as the practical part of this graduation work is concerned, we developed the program IFS-CODEC, which constitutes a user interface as to IFS-decoding and interactive IFS-encoding. With this toolbox, which is embedded in the OSF/Motif® X Window System®, it is possible to draw and edit images. Collages can be obtained in an interactive way and rendering of the attractors of the resulting IFS-codes is possible (deterministic or nondeterministic, with or without condensation). Furthermore, attractors of manually entered IFS-codes can be rendered for a variable amount of iterations. Moments of images, fractal dimensions of images, as well as Pixset distances between images can be calculated. The Mandelbrot set can be rendered for a variable amount of iterations and the bifurcation diagram for the logistic map can be drawn.

Furthermore, we developed a genetic algorithm as an approach to solve the problem of automatic IFS-encoding. The chromosome encoding of transformations into chromosomes is achieved in a straight forward way and has shown to be robust. Also, eight genetic operators have been developed and tested. Two evaluation function parameter sets have been empirically derived. One for finding a symmetry group transformation of a target image (GPS_4) and one for finding a collage of a target image (GPS_{12}). The genetic algorithm has proven to be able to find symmetry group transformations with an approximation quality up to 99% and collages with an approximation quality up to 88%, which is a good result compared to the relatively few runs we could perform.

Our genetic algorithm runs for about 57.6 [*hours*] to encode a 500 × 500 pixel target image into a 2-dimensional IFS, existing of 50 affine transformations. The collage from which this IFS was extracted had an approximation quality of 88% and the attractor of the IFS had an approximation quality of 80%. Since the average CPU-assignment was 15%, this means that the effective calculation time for this collage was about 8.64 [*hours*]. In [ZOR88] it is reported that an interactive system, operated by a trained user, took about 100 [*hours*] to compress a 780 × 1024 pixel image. In [LEV88] it is reported that a simulated annealing based parallellized algorithm took 4 [*hours*] to find a 2-dimensional IFS of 3 affine transformations for a 256 × 256 pixel image, with an approximation quality of 95%. In [SHO91] it is reported that a genetic algorithm took at least 10 [*hours*] to find a 1-dimensional IFS of 3 transformations. In this latter case the collage problem has been simplified to find the IFS of an image consisting of three separate Cantor sets. Of course this is comparable to our symmetry group transformation approach, with which we obtained results up to 99%.

This leads us to the conclusion that our genetic algorithm as it is now, is already performing reasonably well, compared to the other results in the field.

12.2. Recommendations on the user interface

A zoom facility should be implemented in order to be able to zoom in onto rendered fractal attractors and onto the Mandelbrot set. Some preparations in this direction have already been made (see remark section of source code headers).

The genetic parameter settings are currently controlled via global variables in the file `all.h`. A sliderbox widget should be implemented with which this genetic parameter setting can be tuned.

12.3. Recommendations on the genetic algorithm

The genetic algorithm has been validated and found to be robust. All runs described in chapter 11 were successfully terminated. The main recommendation in this area would be to apply a great number of runs on different genetic parameter settings in order to get enough data to perform a thorough statistical analysis. The empirically obtained parameter setting GPS_4 is able to get symmetry group transformations of images with an approximation quality up to 99%. The empirically obtained parameter setting GPS_{12} is able to perform IFS-encoding of a target image with an approximation quality up to 88%. So it seems to be the case that GPS_4 is sufficient for its task (finding of symmetry group transformations) but GPS_{12} can further be improved as far as the task of finding IFS-codes of images is concerned. When tuning GPS_{12} the main area of concern should be the parameter set of the evaluation function. We feel that the operator probabilities as defined in GPS_{12} are good values. Besides, different try outs of the intuitively felt to be "problem specific" genetic operator probability set $P = \{0.0,0.0,0.0,0.25,0.0,0.25,0.25,0.25\}$ on the symmetry group transformation task, gave results with a maximum approximation quality of 79%. This implies that those problem specific operators *Scale*, *Rotate* and *Translate*, together with *Mutate* are not powerful enough. Thus we need the crossover and mutation parameters, which are in fact the basis of a genetic algorithm (see also [SCH91] for an interesting theory about crossover-mutation trade off). Also, from par. 11.6 it has become clear that there is certainly need for the problem specific parameters.

Inserting ***introns*** (non-functional genes) into the chromosomes may be considered to increase the success rate of the genetic algorithm [LEV91].

The current average member fitness evaluation time is 2.7 [*seconds*]. It is necessary to speed up this evaluation time in order to get faster runs of the genetic algorithm. Parallellization may be considered, such that each processor operates on a specific part of the pixel space.

In [WHI91] ***delta coding*** is introduced as a new search strategy for genetic algorithms, which allows iterative searches with complete reinitialization of the population, preserving the progress already made towards solving the optimization task. Delta coding is considered to effectively avoid what is usually a difficult trade off between achieving fast search and sustained diversity (and thereby avoiding premature convergence). Delta coding basically adjusts the population size by monitoring its diversity. When the diversity decreases, the population size is accordingly decreased. Iteration is halted when a certain minimum diversity has been reached. From then on the best solution is saved as a starting point for the next delta iteration. Delta coding treats chromosomes not as parameters, but as delta values ($\pm\delta$), which are added to a partial solution before being evaluated. So the next delta iteration is restarted with an initial population which forms a hypercube of $\pm\delta$ around the previous solution. After each iteration δ is increased. In this case a more and more accurate search around the optimum is performed. This is in fact comparable with the concept of simulated annealing. We would recomment to implement the following variant of delta coding in our genetic algorithm. The masks which are generated within the genetic operator functions *Scale*, *Rotate* and *Translate* are to be divided by the number of the generation which is currently build. In this way we obtain relatively more precise adjustments to the affine transformations as the search process proceeds. Of course this implies a trade off between diversity and accuracy.

Implementation of variable chromosome lengths may be considered, where the chromosome encodes a complete IFS instead of just one affine transformation. In [DAV91] some concepts of variable chromosome encoding are treated.

In our genetic algorithm the value of the evaluation function E(child) is directly assigned to the fitness of that child. The probability that this child will be choosen as parent in the next generation, will be directly proportional to its relative fitness in that next generation. It might be worth trying to apply a different way of fitness assignment. In case of directly assigning child[i].fitness = E(child[i]), we could sort the children by their fitness (which is already done by merging in our genetic algorithm) and assign fitnesses according to child[i].fitness = $S - i$, where S is the population size. In this way each member has a chance of being choosen which is equal to:

$$\frac{S-i}{\sum_{i=1}^{S} i} = \frac{(S-i)}{\frac{1}{2}(S^2+S)} = \frac{2(S-i)}{S^2+S}$$

In case of big populations this would increase the relative probability of best members to be choosen, otherwise they might be "lost" in the diversity of the population. This is because otherwise their probability is simply equal to their fitness divided by the population size, and since the fitness is at most 1, this probability of being choosen can never exceed S^{-1}, which can be a rather small number.

An interesting experiment as to genetic algorithms in general would be to vary the population size S as function of the current generation g. Since genetic algorithms mimic certain properties of evolution, why should we not mimic this additional property of evolution. Referring to chaos theory it might in this case be interesting to use the logistic relationship $S_{g+1} = \mu S_g(1-S_g)$ for some appropriate μ.

12.4. Recommendations on IFS-encoding

During the practical stage of this graduation work, several ideas have grown about improving the IFS-encoding process. Due to time considerations, we have not been able to implement those ideas. Some of the recommended ideas can be carried into the genetic algorithm, others ask for a different approach.

The most important recommendation on this field would be to search for a direct relationship between an affine transformation w_i and its fitness, without first applying the time consuming step of transforming the target image according to this transformation w_i. This would make it possible to speed up the member evaluation time tremendously.

According to [BAR88b,MEN91] it might be considered to use a ***Recurrent Iterated Function System*** (***RIFS***) to model deterministic images. An RIFS is an extension of the IFS as defined in par. 8.1. Unlike IFS-attractors, attractors of recurrent structures are not globally self affine, i.e. one does not find smaller and smaller copies of the entire fractal model buried in it at every scale. Instead, data redundancy is exploited through partial self-affinity.

It is worth to investigate whether the encoding of color and greytone images with use of the Hutchinson distance as part of the evaluation function, will be successful.

It may be considered to try to solve the IFS-encoding problem with a ***neural network***. The choice between a genetic algorithm and a neural network is basically a time-space trade off, since a genetic algorithm uses relatively much time and few memory, while a neural network uses relatively few time and much memory.

In this case it may be efficient to apply image segmentation to break down the image into segments which have (almost) the same color or greytone. Each such segment could be encoded with the collage approach. Of course the IFS-code of the whole image will in this case contain an offset for each encoded segment. If the image is recursively divided into 4 symmetric segments, which are placed in a ***quad-tree*** [HUN79,MAN90], then an alphabet of domain blocks could be constructed, from which the image will be encoded. Since this method does not use the fractal properties which are resident in almost every real world image, the compression performance will not be better than that of the Discrete Cosine Transform [WAI90]. Some more interesting articles about the image segmentation approach are given by [BHA91,KEL89,PEN84,RIG88].

Also, a collage for a monochrome image could be obtained by matching of the image boundary with affine transformed boundary segments. An interesting article about this subject is [WIT89].

In [LIB87] an algorithm is presented to obtain the ***morphological skeleton*** of an image. A skeleton $S(x)$ of an image A returns the radius r of the maximal disk centered at the point $x \in A$. A maximal disk is the largest possible disk which doesn't exceed the boundary of A. The r^{th} skeleton subset of A, denoted by $S_r(A)$, is defined to be the set of all skeleton points $x \in A$, such that $S(x) = r$. An important property of the skeleton function is the fact that it contains all the information necessary to reconstruct the original image. The skeleton of an image will contain certain branch points, and one of them can be considered as the central branch point, i.e. the center of the image. The vectors of the central branch point to the other branch points represent the affine transformations of the sought for IFS. If some parts of the image are still uncovered, they can be covered with maximal disks, which are found by locating skeleton points x in this uncovered region, and calculating $S(x)$ to obtain the radius of the maximal disk. Since the skeleton of an image can be computed in $O(n)$ time (with n the amount of skeleton subsets), this approach looks very promising at a first glance, but it works only for 2-dimensional monochrome images with high fractal dimension, i.e. close to 2. This can be made clear by comparing for instance the structures *leaf* and *fern* (see appendix C). In other words, we could state that this approach will only work satisfactory for images which can be represented as simple polygons, i.e. polygons which do not cut themselves. The structure *leaf* has a fractal dimension of about 1.96, while the structure *fern* has a fractal dimension of about 1.78 (see table 3.1). As we can easily check, the skeleton of the structure *leaf* will be equal to its nerve structure (compare to real leaves), but the skeleton of the structure *fern* will be equal to the structure itself, because the maximal disks in all points of the image have a diameter of 1 [*pixel*]. It is obvious that a skeleton which is equal to the structure itself is of no use.

It may also be considered to first scan the boundary of a target image and represent it as a (simple) polygon. In [ATA91] an $O(n)$ algorithm (with n the amount of polygon vertices) is presented. Maybe the current time consuming steps of transforming and evaluating all pixels can be speeded up by only applying those operations on the polygon vertices.

An approach to tackle the IFS-encoding problem would be the following. Suppose the target image A contains $|A|$ [*pixels*]. Then build an IFS consisting of $|A|$ affine transformations, i.e.:

$$([x_{left}, x_{right}] \times [y_{upper}, y_{lower}], \{\ w_i(x,y) = (a_i x + b_i y + e,\ c_i x + d_i y + f) : i = 1,\ldots,|A|\ \}, P)$$

such that:

$$(\ \underline{A}\ i,j : 1 \le i \le |A|, j \in A : a_i, b_i, c_i, d_i = 0 \wedge (e_i, f_i) = (x_j, y_j)\ )$$

This is of course a trivial IFS-encoding which doesn't establish any data compression at all. Then enlarge the parameters a_i ,b_i ,c_i and d_i of each transformation w_i and transform the target image according to this transformation w_i. Next remove the transformations from the IFS which are completely covered by w_i. This could be done in a recursive way. According to the collage theorem we can control the maximal tile sizes in order to get a small contractivity factor, which will assure us a good approximation quality. Of course this is a trade off between encoding accuracy and data compression.

In [BUD90] it is shown that recursive techniques in VLSI-design lead in a very natural way to certain fractals which reflect the asymptotic geometric properties of the design if its degree of recursiveness tends to infinity. Recursive design consists of breaking a problem into smaller problems in such way that from solutions to the smaller problems one can easily combine a solution to the entire problem. The mentioned article is based on a ***REcursive LAyout Computing System*** (***RELACS***). It may be interesting to investigate if and in which way the IFS-principle can be used in VLSI-design processes which are currently studied.

Appendix A

IFS-CODEC users guide

As part of the work involved with this master thesis, an integrated algorithm was developed. The goal of this algorithm is to:

(i) facilitate deterministic IFS-decoding (with and without condensation);

(ii) facilitate nondeterministic IFS-decoding (with and without condensation);

(iii) facilitate interactive IFS-encoding of an image;

(iv) facilitate automatic IFS-encoding of an image.

The IFS-CODEC algorithm is written in C, is embedded in the OSF/Motif® X Window System® and runs under the UNIX™ operating system. IFS-CODEC consists of about 7500 lines of source code (see appendix B) and is implemented on a HP 9000/750 system.

A.1. The user interface

We have choosen for the OSF/Motif® X Window System® because it guarantees a good-looking software system, but also because it is an industry standard. The development of the user interface has mainly been ruled by information ergonomic considerations. The most important one is the first law of information ergonomics: developer $\neq$ user. But since the users of the algorithm are also UNIX™ users, we felt that the freedom supported by UNIX™ should also be carried into the IFS-CODEC algorithm. Besides, making an algorithm "idiot proof" implies severe limitations and doesn't at all imply user friendlyness.

A.2. Getting started

Make sure that the following files are resident in the same directory: `all.h`, `ifs*`, `boxes.o`, `collage.o`, `control.o`, `draw.o`, `exposures.o`, `filehandler.o`, `genetic.o`, `graphics.o`, `handle.o`, `ifs.o`, `menubars.o`, `messages.o`, `operators.o`, `render.o`, `specials.o`, `strings.o`, `tracker.o`, `writer.o`.

After typing "ifs" from the invocation screen, the IFS-CODEC main window will pop up, consisting of three menubars, two (upper) text windows and two (lower) pixel windows. The grids of the pixel windows are default organized as follows:

(i) origin (0,0) in center of window;

(ii) y increasing upwards;

(iii) x increasing to the right.

After the IFS-CODEC window has popped up, the menu selections can either be made by mouse or by keyboard. If it is done by mouse, then the mouse pointer must be placed on the desired part of the menubar and the left mouse button must be pressed. When this button is released, the selection is made. If selection is established by keyboard, the traversal is performed by the direction keys and the selection is done by hitting the return key. Pressing an (underlined) mnemonic for a menu item in the most recently posted menu, selects that item. You can press the "Help" button to get more information. From then on the system should be self explanatory.

First specify a filename without an extension, because the program will determine the appropriate extension by itself. By looking at the file extensions (with UNIX™ command "ls") you can see what kind of data each file contains:

(i) "filename.img": a sequence of (x,y) coordinates which specify a user drawn image;

(ii) "filename.col": a sequence of (x,y) coordinates which specify a collage image;

(iii) "filename.man": IFS-code generated with interactive IFS-encoding;

(iv) "filename.aut": IFS-code generated with automatic IFS-encoding;

(v) "filename.ifs": IFS-code generated with a source editor.

Of course, each filename and extension can be modified under UNIX™. If settings of program parameters are changed interactively, the current values will be automatically updated in the according text window.

A.3. Viewing/editing source images

With this facility you can operate upon images. First select a file by the "Select file" option. Then apply each of the following steps as many times as desired:

(i) To load an existing image, contained in "filename.img", select "Options → Viewedit source image → Load file";

(ii) To put the cursor in draw mode, select "Options → Viewedit source image → Draw (toggle)". Now you can draw in the pixel window by pushing the left mouse button and using the mouse pointer as pencil. Each time the former selection is applied, the mode toggles between draw mode and normal mode (the current mode is written to the invocation screen);

(iii) To put the cursor in erase mode, select "`Options → Viewedit source image → Erase (toggle)`". Now you can erase in the pixel window by pushing the left mouse button and using the mouse pointer as erasor. Each time the former selection is applied, the mode toggles between erase mode and normal mode (the current mode is written to the invocation screen);

(iv) To save the current image, which is contained in the pixel window, to the file "`filename.img`", select "`Options → Viewedit source image → Save file`".

A.4. Viewing/editing IFS-codes

It is possible to view the contents of an IFS-code file, both textual and graphical. First select a file by the "`Select file`" option. Then apply one of the following three steps:

(i) To achieve a graphical view, select "`Options → Viewedit IFS-code → Graphical view → .ext`", where "`.ext`" is one of the three files (`.aut`, `.man`, `.ifs`) which contain an IFS-code. On the pixel window the graphical effect of the affine transformations is shown by applying them on a rectangle;

(ii) To achieve a textual view, select "`Options → Viewedit IFS-code → Textual view → .ext`", where "`.ext`" is one of the three files (`.aut`, `.man`, `.ifs`) which contain an IFS-code. On the pixel window the parameters *a,b,c,d,e,f,p* of all affine transformations in the file are displayed. If the IFS doesn't fit completely in the pixel window, then a scrollbar will automatically pop up, with which you can scroll the IFS up and down;

(iii) If you select "`Options → Viewedit IFS-code → Textual edit → .ext`", where "`.ext`" is one of the three files (`.aut`, `.man`, `.ifs`) which contain an IFS-code, then a message will pop up. This message tells you that editing such file can be done with VI or Emacs and which format has to be obeyed if you do so.

A.5. IFS-decoding

The IFS-decoding facility operates on files that contain IFS-codes, i.e. files with extensions "`.ifs`", "`.man`" or "`.aut`". After a filename (without an extension) is selected by the "`Select file`" option, the following actions must be applied:

(i) First decide whether you want to decode with (default) or without condensation. You can toggle between those options by pressing "`Render control → Condensation`". The current selection is written to the invocation screen. If decoding with condensation is selected, then the initial condensation set consists of all contents (except coordinate axes) of the according pixel window. These contents may be influenced by the use of

"`Display → Clear`" and "`Options → Viewedit source image`". From then on you can either load a file or draw/erase a picture yourself with the mouse;

(ii) Next determine the amount of iterations by "`Render control → Define #iterations`". A sliderbar will pop up with which you can vary the amount of desired iterations between 1 and 100;

(iii) Next determine if and how the startpixel has to be defined by "`Render control → Define startpixel`". This is not really essential for the decoding result, but only influences the fact whether or not the pixels which are generated by the first few iterations will be part of the attractor. The file "`codes`" (see appendix C) contains some examples of IFS-codes and the appropriate startpixel settings;

(iv) Next determine if and how the origin has to be redefined by "`Display → Redefine origin`". The origin is default defined to (0,0) which is the center of the pixel window. You can check where the origin is actually located by selecting "`Display → Axes (toggle)`" which will toggle the coordinate axes on the pixel window;

(v) Finally invoke the IFS-decoding process by selecting one of the following two options:

"`Options → Deterministic IFS-decoding → .ext`"

"`Options → Nondeterministic IFS-decoding → .ext`"

From chapter 8 we know that deterministic IFS-decoding will neglect the probabilities, and nondeterministic IFS-decoding will not.

If you want to verify the effect of a single affine transformation on a certain image, then load the image, set the amount of iterations equal to 1, set the condensation mode to FALSE and invoke the deterministic IFS-decoding process according to point (v) above.

A.6. Interactive IFS-encoding

The interactive IFS-encoding facility generates files that contain IFS-codes (with extension "`.man`") and files that contain graphical collages (with extension "`.col`"). After a filename (without extension) is selected by the "`Select file`" option, the following actions must be applied:

(i) First make sure that the pixel window contains the image which must be encoded. If this is done by loading an "`.img`" file it must be considered wheather or not the current filename has to be changed, because it may be the case that there already exists a "`.man`" file with the current filename. In this case the output of the IFS-encoding session will be appended to that file;

(ii) Next determine if and how the origin has to be redefined by "`Display → Redefine origin`". Note: while making a collage it is not possible to redefine the origin anymore and any attempt to do so will result in a warning;

(iii) Next add a similitude by selecting "`Options → Interactive IFS-encoding → Add similitude`". In the pixel window appears a 80% down scaled copy of the original target image, which is still visible;

(iv) Next select "`Options → Interactive IFS-encoding → Transform similitude`". A box with six sliderbars will pop up. You can move the sliderbars by selecting them with the mouse pointer and then dragging them to the left or to the right while the left mouse button is kept pressed. Also, you can put the mouse pointer at the far left or at the far right of the sliderbar and move the slider by clicking the left mouse button. Each click will move the slider a distance of 10% of the sliderbar width. If the similitude is to be just rotated, then *X-rotation* and *Y-rotation* sliders have to be moved the same amount. If the similitude is just to be scaled the the *X-scaling* and *Y-scaling* sliders have to be moved the same amount;

(v) If the similitude is to be fixed, select "`Options → Interactive IFS-encoding → Fix similitude`". The program will automatically save the parameters *a,b,c,d,e* and *f* (which are calculated from the slider values) to the file "`filename.man`";

(vi) If the original image is not yet totally covered with similitudes (i.e. the collage is not yet ready), new similitudes can be added, transformed and fixed, according to respectively steps (iii), (iv) and (v). When the collage is finished, select "`Options → Interactive IFS-encoding → Save collage`". The program now calculates the probabilities of the saved affine transformations and adds them to the file "`filename.man`".

If you want to add a transformation to an already existing collage, then simply select its filename and apply all steps (iv), (v) and (vi). Of course you can also create an IFS-code directly from a text editor. Such a file should be saved with the extension "`.ifs`". if you want to let the program take care of the probabilities, just apply "`mv filename.ifs filename.man`" under UNIX™ and select the filename. Then just apply step (vi), which will calculate the probabilities and add them to the file. Note: make sure that an editor-created IFS-code file obeys the format as displayed in the message which appears when selecting "`Options → Viewedit IFS-code → Textual edit`".

A.7. Automatic IFS-encoding

When applying an automatic IFS-encoding run, a file "`monitor.dat`" will be automatically created. This monitor file contains all kinds of information about a genetic run. In appendix D a typical output of this file is given. There are a lot of parameter settings involved with this option. Due to time constraint considerations, most of these parameters can not yet be interactively controlled by means of the user interface. For the time being, those parameters must be directly set in the file "`all.h`" by applying the following steps:

(i) Decide whether you want to watch the state space search or not. If so, set the global constant *DISPLAY* to the value 1, else to the value 0. If *DISPLAY* is set to 1, the original target image will get lost from the screen and the state space search of the genetic algorithm is displayed on the screen by means of the successive population members. The drawback of this option is the fact that it takes about 50% more CPU-time on an IFS-encoding run;

(ii) Decide whether you want to run the algorithm in the debug mode or not. If the global constant *DEBUG* is set to 1, additional information will be generated in the monitor file. This additional information basically deals with the processes of parent and operator selection, mask generation and child creation;

(iii) Decide what the amount of generations and the population size should be. Those values are set by resp. the global constants *MAX_GENERATIONS* and *POPULATION_SIZE*;

(iv) Decide if you want to change the probabilities with which the genetic operators *MacroCrossover, MicroCrossover, MixedCrossover, Mutate, Jump, Scale, Rotate* and *Translate* are applied. If so, adjust the values of the according global constants *P_MACRO, P_MICRO, P_MIXED, P_MUTATE, P_JUMP, P_SCALE, P_ROTATE and P_TRANSLATE*. It is important to make sure that the sum of those probabilities is 1;

(v) Decide what the maximum encoding error should be. This value is represented by the global variable *EPSILON* which must have an integer value in the interval [0%,100%). The default value is 2%. Of course it is not certain that the encoding will be established with a maximum encoding error *EPSILON*. It may for instance be the case that the amount of generations or the population size is not large enough.

After the above global constants in the file "`all.h`" have been adjusted, the IFS-encoding session can be started. To encode an image, first select a filename and make sure that the image which is to be encoded, is contained in the according pixel window. Then apply the following steps:

(vi) If you have set the global constant *DISPLAY* to 1 (see step (i)), then the original image will get lost from the screen and the state space search of the genetic algorithm will be displayed on the screen by means of the successive population members. If you want to keep the target image continuously visible during the search process, you must load the image into the opposite window from the one you want to use for the encoding;

(vii) Select "`Options → Automatic IFS-encoding → Identity (toggle)`" if you want to search for the identity transformation, which maps the target image onto itself. In case of a symmetric image (like a square) it is possible that a symmetry group transformation will be found (for example a rotation by 90 [*degrees*]). The current selection is written to the invocation screen. This option is basically meant for validation purposes with respect to the genetic algorithm;

(viii) Select "`Options → Automatic IFS-encoding → Calculate`" to start the encoding process. If *IDENTITY* is set to TRUE then only one transformation (the identity or another member of the symmetry group of the image) will be sought for. If *IDENTITY* is set to FALSE (default) then a collage will be sought for. The algorithm will in this case terminate as soon as at least one of the following conditions holds:

- all members of generation *MAX_GENERATIONS* have been evaluated;
- the current encoding error is less than *EPSILON*;
- the collage consists of *MAX_TRANSFORMATIONS* affine transformations.

During the encoding process, the number of the affine transformation which is encoded, is displayed on the invocation screen. In the text window the field "*iteration*" displays the current generation. After termination of the program the pixel window will contain the collage of all found affine transformations applied on the target image. The file "`filename.aut`" will contain these affine transformations.

A.8. Special utilities

The special utilities contain six options:

(i) "`Pixset distance`" calculates the Pixset distance d_p between the images contained in the left and right pixel windows. The result will be written to the invocation screen;

(ii) "`Specials → Rectangle` draws a rectangle on the according pixel window, which can be used for making collages such as the Menger sponge or the Sierpinski triangle. In this case one doesn't need to have a target image of those structures. For example to create a collage of a Sierpinski triangle one does only have to add three similitudes, each with *X-scaling* and *Y-scaling* both set to 50%. Two of those similitudes must cover the lower half of the rectangle, while the third is placed centered on the top of the two others;

(iii) "`Specials → Bifurcation diagram`" draws the bifurcation diagram of the logistic map: $f(x) = ax(1-x)$. For more information see chapter 4. There are no parameter settings involved with this option;

(iv) "`Specials → Mandelbrot set`" draws the Mandelbrot set M of the function $f(z) = z^2+c$ in the complex c-plane (see chapter 6). If "`Display → Axes (toggle)`" was selected first, then the real and imaginary axis will be displayed. Before you draw M, first select the amount of iterations that must be applied to each pixel. This can be done with "`Render control → Define #iterations`". The higher the amount of iterations, the more accurate (and slower!) M is rendered. Due to screen resolution, more than 25 iterations will not visibly increase the picture quality;

(v) "`Specials → Moment sequence`" calculates the first 5 complex moments of the image which is displayed in the according window (see par. 9.4 and 10.3). The moments are calculated with respect to the current origin setting. The result is written both to the invocation screen and to the file "`filename.mom`";

(vi) "`Specials → Fractal dimension`" calculates the fractal dimension of the image which is displayed in the according window. The calculation is done according to example 3.2.

Appendix B

IFS-CODEC source code

```
#******************************************************************************
# Copyright(c)    : Eindhoven University of Technology (TUE,NL)
#                 : Faculty of Electrical Engineering (E)
#                 : Design Automation Group (ES)
# Mail address    : emilevd@viper.es.ele.tue.nl
# Project         : Fractal imagecompression with Iterated Function Systems
# Supervisor      : Prof. Dr. Ing. J.A.G. Jess
# File            : Makefile
# Purpose         : Contains IFS-CODEC makefile
# Part of         : Not applicable
# Created on      : April 12, 1992
# Created by      : Emile van Duren
# Last modified on: December 17, 1992
# Modified by     : Emile van Duren
# Remarks         : None
#******************************************************************************

#
# name of executable file
#
NAME = ifs
#
# libraries to be linked (when debugging: change option -O into -g)
#
CFLAGS = -O -I/usr/include/X11R4 -I/usr/include/Motif1.1
LDFLAGS = -O -L/usr/lib/X11R4 -L/usr/lib/Motif1.1
#
# sourcefiles
#
SOURCES = \
        boxes.c       \
        collage.c     \
        control.c     \
        draw.c        \
        exposures.c   \
        filehandler.c \
        genetic.c     \
        graphics.c    \
        handle.c      \
        ifs.c         \
        menubars.c    \
        messages.c    \
        operators.c   \
        render.c      \
        specials.c    \
        strings.c     \
        tracker.c     \
        writer.c
#
# convert sourcefiles into objectfiles
#
OBJECTS = $(SOURCES:.c=.o)
#
# libraries to be linked
#
LIBS = -lXm -lXt -lX11 -lPW -lm
```

```
#
# actual compiler command, to be invoked with "make"
#
$(NAME): $(OBJECTS)
	cc $(LDFLAGS) -o $(NAME) $(OBJECTS) $(LIBS)
#
# clean all directories, to be invoked with "make clean"
#
clean:
	rm -f $(OBJECTS)
#
# make backup of sourcefiles, all.h and this file,
# to be invoked with "make backup"
#
backup:
	cp /users1/emilevd/work/codes /users1/emilevd/codes/
	cp /users1/emilevd/work/*.ifs /users1/emilevd/codes/
	cp /users1/emilevd/work/*.img /users1/emilevd/images/
	cp /users1/emilevd/work/*.c /users1/emilevd/sources/
	cp /users1/emilevd/work/*.h /users1/emilevd/lib/
	cp /users1/emilevd/work/Makefile /users1/emilevd/sources/
```

```
/******************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : all.h
 * Purpose         : Contains all necessary libraryfiles for IFS-CODEC
 * Part of         : IFS-CODEC
 * Created on      : May 08, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 02, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 ******************************************************************************/

/*
 * includefiles
 */
#include<math.h>
#include<stdio.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<X11/StringDefs.h>
#include<Xm/Xm.h>
#include<Xm/BulletinB.h>
#include<Xm/CascadeB.h>
#include<Xm/DrawingA.h>
#include<Xm/Form.h>
#include<Xm/Frame.h>
#include<Xm/Label.h>
#include<Xm/MessageB.h>
#include<Xm/PushB.h>
#include<Xm/RowColumn.h>
#include<Xm/ScrollBar.h>
#include<Xm/SelectioB.h>
#include<Xm/Separator.h>
#include<Xm/Text.h>
/*
 * global constants
 */
#define AREA_WIDTH 500
#define DEBUG 0
#define DISPLAY 1
#define EPSILON 2
#define EXTENDED 0
#define FALSE 0
#define IDENTITY 0
#define MAX_ARGS 15
#define MAX_GENERATIONS 10
#define MAXLINES 500
#define MAXLINESIZE 60
#define MAX_TRANSFORMATIONS 2
#define PI 3.141592654
#define PIXEL_AREA_HEIGHT 500
#define P_JUMP 0.11
#define P_MACRO 0.15
#define P_MICRO 0.15
#define P_MIXED 0.15
#define P_MUTATE 0.011
#define POPULATION_SIZE 10
#define P_ROTATE 0.12
#define P_SCALE 0.12
#define P_TRANSLATE 0.12
#define SCALEBOX_WIDTH 250
#define TEXT_AREA_HEIGHT 215
#define TRUE 1
/*
 * global structures
 */
typedef struct {
  int xrotation,yrotation,xscaling,yscaling,xshift,yshift,probability;
} Affine;
```

```
typedef struct {
  float fitness;
  int a,b,c,d,e,f;
} Chromosome;

typedef struct {
  Boolean filehelp,fileopenwarning,originwarning,tablemessage;
} Guard;

typedef struct {
  float a,b,c,d,e,f,p;
} IFScode;

typedef struct {
  Widget box,slider,okbutton;
} Numberbox;

typedef struct {
  Widget box,okbutton,xslider,yslider;
} Positionbox;

typedef struct {
  Widget box,scrollbar,okbutton;
} Scrollbox;

typedef struct {
  Widget box,okbutton,xscslider,yscslider,xrtslider,yrtslider,xshslider,yshslider;
} Transformbox;

typedef struct {
  Widget box,okbutton,zoomnowbutton,xoffsetslider,yoffsetslider,zoomslider;
} Zoombox;

typedef struct {
  char          mode,*filename;
  int           ncolors,oldpointx,oldpointy,xoffset,yoffset,zoomfactor;
  int           niterations,xorigin,yorigin,xstart,ystart;
  long int      npixels;
  Affine        transformation;
  Boolean       axes,condensation,drawgrab,erasegrab,firsttime,identity,unzoom;
  Dimension     width,height;
  GC            gc;
  Guard         first;
  Numberbox     iterationbox;
  Pixmap        pix,sim,col;
  Positionbox   originbox,startpixelbox;
  Scrollbox     scrollbox;
  Transformbox  similitudebox;
  Zoombox       zoombox;
} Drawdata;

typedef struct {
  char        *ext,*chars[MAXLINES];
  int         descent,fontheight,nlines,top;
  int         length[MAXLINES],rbearing[MAXLINES];
  XFontStruct *font;
} Scrolldata;

typedef struct {
  char *mode,*srce,*size,*cont,*save,*orgn,*init,*amit,*iter,*zoom;
} Textdata;

/*
 * global variables
 */
int ac,maxiterations;
Arg al[MAX_ARGS];
Boolean firsthelp,printmessage,copyrightmessage,debug;
Drawdata leftpixeldata,rightpixeldata,lefttextdata,righttextdata;
Textdata LText,RText;
Scrolldata LScroll,RScroll;
Widget form,leftpixelarea,rightpixelarea,lefttextarea,righttextarea;
Widget lefttextframe,righttextframe,leftpixelframe,rightpixelframe;
```

```
/****************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : boxes.c
 * Purpose         : Creates scalebox and promptbox widgets
 * Part of         : IFS-CODEC
 * Created on      : June 04, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 10, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 ****************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void AmountOfIterationsHandler();     /* contained in handle.c      */
void ChangeStartPixelHandler();       /* contained in handle.c      */
void ClearPixelArea();                /* contained in draw.c        */
void DrawAxes();                      /* contained in draw.c        */
void FileHelpCallback();              /* contained in messages.c    */
void ManageSimilitude();              /* contained in transform.c   */
void OriginHandler();                 /* contained in handle.c      */
void ScrollBarMovedHandler();         /* contained in handle.c      */
void SelectFile();                    /* contained in filehandler.c */
void Unmanage();                      /* contained in handle.c      */
void ZoomController();                /* contained in control.c     */
void ZoomHandler();                   /* contained in handle.c      */
Drawdata *GetWindowData();            /* contained in control.c     */
Scrolldata *GetScrollData();          /* contained in control.c     */


Widget CreatePromptBox(parent,location)
     Widget parent;
     char location;
{
  Widget promptbox;
  XmString xmstr;
  /*
   * create promptbox to appear when fileselectionbutton in filemenupane of
   * uppermenu is pressed
   */
  ac = 0;
  if (location == 'l')
    {
      xmstr = XmStringCreate("select leftfile",XmSTRING_DEFAULT_CHARSET);
    }
  if (location == 'r')
    {
      xmstr = XmStringCreate("select rightfile",XmSTRING_DEFAULT_CHARSET);
    }
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  promptbox = XmCreatePromptDialog(parent,"PromptBox",al,ac);
  XtAddCallback(promptbox,XmNhelpCallback,FileHelpCallback,location);
  XtAddCallback(promptbox,XmNokCallback,SelectFile,location);
  XmStringFree(xmstr);
  return(promptbox);
}


Widget CreateSimilitudeTransformBox(parent,location)
     Widget parent;
     char location;
{
  int mode;
  Drawdata *data;
  XmString xmstr;
  Affine *transformation;
```

```
/*
 * parameter initialization
 */
data = GetWindowData(location,'p');
/*
 * create transformbox to appear when transform similitudebutton in viewedit
 * ifs table menu of optionsmenu of left or rightmenu is pressed
 */
ac = 0;
if (location == 'l')
  {
    mode = 0;
    xmstr = XmStringCreate("transform left similitude",XmSTRING_DEFAULT_CHARSET);
  }
if (location == 'r')
  {
    mode = 1;
    xmstr = XmStringCreate("transform right similitude",XmSTRING_DEFAULT_CHARSET);
  }
XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
data->similitudebox.box = XmCreateFormDialog(parent,"TransformBox",al,ac);
XmStringFree(xmstr);
/*
 * create a X-scaling scrollbar with range 0 to 100 [%]
 */
ac = 0;
xmstr = XmStringCreate(" X-scaling [percent]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,0); ac++;
XtSetArg(al[ac],XmNmaximum,100); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,80); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
data->similitudebox.xscslider = XmCreateScale(data->similitudebox.box,
                                              "XscSlider",al,ac);
XtAddCallback(data->similitudebox.xscslider,XmNdragCallback,
          ManageSimilitude,mode);
XtAddCallback(data->similitudebox.xscslider,XmNvalueChangedCallback,
          ManageSimilitude,mode);
XtManageChild(data->similitudebox.xscslider);
XmStringFree(xmstr);
/*
 * create a Y-scaling scrollbar with range 0 to 100 [%]
 */
ac = 0;
xmstr = XmStringCreate(" Y-scaling [percent]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,0); ac++;
XtSetArg(al[ac],XmNmaximum,100); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,80); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,data->similitudebox.xscslider); ac++;
data->similitudebox.yscslider = XmCreateScale(data->similitudebox.box,
                                              "YscSlider",al,ac);
XtAddCallback(data->similitudebox.yscslider,XmNdragCallback,
          ManageSimilitude,mode+2);
XtAddCallback(data->similitudebox.yscslider,XmNvalueChangedCallback,
          ManageSimilitude,mode+2);
XtManageChild(data->similitudebox.yscslider);
XmStringFree(xmstr);
/*
 * create a X-rotation scrollbar with range -180 to 180 [degrees]
 */
ac = 0;
xmstr = XmStringCreate(" X-rotation [degrees]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,-180); ac++;
XtSetArg(al[ac],XmNmaximum,180); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,0); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,data->similitudebox.yscslider); ac++;
data->similitudebox.xrtslider = XmCreateScale(data->similitudebox.box,
                                    "XrtSlider",al,ac);
XtAddCallback(data->similitudebox.xrtslider,XmNdragCallback,
          ManageSimilitude,mode+4);
XtAddCallback(data->similitudebox.xrtslider,XmNvalueChangedCallback,
          ManageSimilitude,mode+4);
XtManageChild(data->similitudebox.xrtslider);
XmStringFree(xmstr);
/*
 * create a Y-rotation scrollbar with range -180 to 180 [degrees]
 */
ac = 0;
xmstr = XmStringCreate(" Y-rotation [degrees]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,-180); ac++;
XtSetArg(al[ac],XmNmaximum,180); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,0); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,data->similitudebox.xrtslider); ac++;
data->similitudebox.yrtslider = XmCreateScale(data->similitudebox.box,
                                    "YrtSlider",al,ac);
XtAddCallback(data->similitudebox.yrtslider,XmNdragCallback,
          ManageSimilitude,mode+6);
XtAddCallback(data->similitudebox.yrtslider,XmNvalueChangedCallback,
          ManageSimilitude,mode+6);
XtManageChild(data->similitudebox.yrtslider);
XmStringFree(xmstr);
/*
 * create a X-translation scrollbar with range -250 to 250 [pixels]
 */
ac = 0;
xmstr = XmStringCreate(" X-translation [pixels]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,-250); ac++;
XtSetArg(al[ac],XmNmaximum,250); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,0); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,data->similitudebox.yrtslider); ac++;
data->similitudebox.xshslider = XmCreateScale(data->similitudebox.box,
                                    "XshSlider",al,ac);
XtAddCallback(data->similitudebox.xshslider,XmNdragCallback,
          ManageSimilitude,mode+8);
XtAddCallback(data->similitudebox.xshslider,XmNvalueChangedCallback,
          ManageSimilitude,mode+8);
XtManageChild(data->similitudebox.xshslider);
XmStringFree(xmstr);
/*
 * create a Y-translation scrollbar with range -250 to 250 [pixels]
 */
ac = 0;
xmstr = XmStringCreate(" Y-translation [pixels]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,-250); ac++;
XtSetArg(al[ac],XmNmaximum,250); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
```

```
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNvalue,0); ac++;
  XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->similitudebox.xshslider); ac++;
  data->similitudebox.yshslider = XmCreateScale(data->similitudebox.box,
                                        "YshSlider",al,ac);
  XtAddCallback(data->similitudebox.yshslider,XmNdragCallback,
            ManageSimilitude,mode+10);
  XtAddCallback(data->similitudebox.yshslider,XmNvalueChangedCallback,
            ManageSimilitude,mode+10);
  XtManageChild(data->similitudebox.yshslider);
  XmStringFree(xmstr);
  /*
   * create OK-button
   */
  ac = 0;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->similitudebox.yshslider); ac++;
  data->similitudebox.okbutton = XmCreatePushButtonGadget(data->similitudebox.box,
                                                  "OK",al,ac);
  XtAddCallback(data->similitudebox.okbutton,XmNactivateCallback,Unmanage,
            data->similitudebox.box);
  XtManageChild(data->similitudebox.okbutton);
  /*
   * return value of this function
   */
  return(data->similitudebox.box);
}


Widget CreateOriginBox(parent,location)
     Widget parent;
     char location;
{
  int mode;
  Drawdata *data;
  XmString xmstr;
  int xvalue,yvalue;
  /*
   * parameter initialization
   */
  data = GetWindowData(location,'p');
  xvalue = data->xorigin - data->width/2;
  yvalue = data->yorigin - data->height/2;
  if (location == 'l')
    {
      mode = 0;
      xmstr = XmStringCreate("redefine left origin",XmSTRING_DEFAULT_CHARSET);
    }
  if (location == 'r')
    {
      mode = 1;
      xmstr = XmStringCreate("redefine right origin",XmSTRING_DEFAULT_CHARSET);
    }
  /*
   * create originbox to appear when redefine originbutton in display
   * menu of left or rightmenu is pressed
   */
  ac = 0;
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  data->originbox.box = XmCreateFormDialog(parent,"OriginBox",al,ac);
  XmStringFree(xmstr);
  /*
   * create a X-origin scrollbar with range -250 to 250 [pixels]
   */
  ac = 0;
  xmstr = XmStringCreate(" X-origin [pixels]",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
  XtSetArg(al[ac],XmNminimum,-250); ac++;
  XtSetArg(al[ac],XmNmaximum,250); ac++;
  XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNvalue,xvalue); ac++;
  XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  data->originbox.xslider = XmCreateScale(data->originbox.box,"XorSlider",al,ac);
  XtAddCallback(data->originbox.xslider,XmNvalueChangedCallback,OriginHandler,mode);
  XtAddCallback(data->originbox.xslider,XmNvalueChangedCallback,DrawAxes,location);
  XtManageChild(data->originbox.xslider);
  XmStringFree(xmstr);
  /*
   * create a Y-origin scrollbar with range -250 to 250 [pixels]
   */
  ac = 0;
  xmstr = XmStringCreate(" Y-origin [pixels]",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
  XtSetArg(al[ac],XmNminimum,-250); ac++;
  XtSetArg(al[ac],XmNmaximum,250); ac++;
  XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNvalue,yvalue); ac++;
  XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->originbox.xslider); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  data->originbox.yslider = XmCreateScale(data->originbox.box,"YorSlider",al,ac);
  XtAddCallback(data->originbox.yslider,XmNvalueChangedCallback,OriginHandler,mode+2);
  XtAddCallback(data->originbox.yslider,XmNvalueChangedCallback,DrawAxes,location);
  XtManageChild(data->originbox.yslider);
  XmStringFree(xmstr);
  /*
   * create OK-button
   */
  ac = 0;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->originbox.yslider); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
  data->originbox.okbutton = XmCreatePushButtonGadget(data->originbox.box,"OK",al,ac);
  XtAddCallback(data->originbox.okbutton,XmNactivateCallback,Unmanage,
           data->originbox.box);
  XtManageChild(data->originbox.okbutton);
  /*
   * return value of this function
   */
  return(data->originbox.box);
}


Widget CreateStartPixelBox(parent,location)
    Widget parent;
    char location;
{
  int mode;
  Drawdata *data;
  XmString xmstr;
  int xvalue,yvalue;
  /*
   * parameter initialization
   */
  data = GetWindowData(location,'p');
  xvalue = data->xstart - data->width/2;
  yvalue = data->ystart - data->height/2;
  if (location == 'l')
    {
    mode = 0;
    xmstr = XmStringCreate("define left startpixel",XmSTRING_DEFAULT_CHARSET);
    }
  if (location == 'r')
    {
      mode = 1;
      xmstr = XmStringCreate("define right startpixel",XmSTRING_DEFAULT_CHARSET);
    }
  /*
   * create startpixelbox to appear when define startpixelbutton in
   * deterministic IFS-decoding or nondeterministic IFS-decoding is
   * pressed
   */
  ac = 0;
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  data->startpixelbox.box = XmCreateFormDialog(parent,"StartPixelBox",al,ac);
  XmStringFree(xmstr);
  /*
   * create a X-origin scrollbar with range -250 to 250 [pixels]
   */
  ac = 0;
  xmstr = XmStringCreate(" X-origin [pixels]",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
  XtSetArg(al[ac],XmNminimum,-250); ac++;
  XtSetArg(al[ac],XmNmaximum,250); ac++;
  XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNvalue,xvalue); ac++;
  XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  data->startpixelbox.xslider = XmCreateScale(data->startpixelbox.box,"XorSlider",al,ac);
  XtAddCallback(data->startpixelbox.xslider,XmNvalueChangedCallback,
           ChangeStartPixelHandler,mode);
  XtManageChild(data->startpixelbox.xslider);
  XmStringFree(xmstr);
  /*
   * create a Y-origin scrollbar with range -250 to 250 [pixels]
   */
  ac = 0;
  xmstr = XmStringCreate(" Y-origin [pixels]",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
  XtSetArg(al[ac],XmNminimum,-250); ac++;
  XtSetArg(al[ac],XmNmaximum,250); ac++;
  XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNvalue,yvalue); ac++;
  XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->startpixelbox.xslider); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  data->startpixelbox.yslider = XmCreateScale(data->startpixelbox.box,"YorSlider",al,ac);
  XtAddCallback(data->startpixelbox.yslider,XmNvalueChangedCallback,
             ChangeStartPixelHandler,mode+2);
  XtManageChild(data->startpixelbox.yslider);
  XmStringFree(xmstr);
  /*
   * create OK-button
   */
  ac = 0;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->startpixelbox.yslider); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
  data->startpixelbox.okbutton = XmCreatePushButtonGadget(data->startpixelbox.box,
                                                       "OK",al,ac);
  XtAddCallback(data->startpixelbox.okbutton,XmNactivateCallback,Unmanage,
             data->startpixelbox.box);
  XtManageChild(data->startpixelbox.okbutton);
  /*
   * return value of this function
   */
  return(data->startpixelbox.box);
}
```

```
Widget CreateIterationBox(parent,location)
     Widget parent;
     char location;
{
  int value;
  Drawdata *data;
  XmString xmstr;
  /*
   * parameter initialization
   */
  data = GetWindowData(location,'p');
  value = data->niterations;
  if (location == 'l') xmstr = XmStringCreate("left #iterations",XmSTRING_DEFAULT_CHARSET);
  if (location == 'r') xmstr = XmStringCreate("right #iterations",XmSTRING_DEFAULT_CHARSET);
  /*
   * create iterationbox to appear when define #iterations button in
   * rendercontrolpulldownmenu is pressed
   */
  ac = 0;
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  data->iterationbox.box = XmCreateFormDialog(parent,"IterationBox",al,ac);
  XmStringFree(xmstr);
  /*
   * create an iteration scrollbar with range 1 to 100 [iterations]
   */
  ac = 0;
  xmstr = XmStringCreate(" #iterations [-]",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
  XtSetArg(al[ac],XmNminimum,1); ac++;
  XtSetArg(al[ac],XmNmaximum,100); ac++;
  XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNvalue,value); ac++;
  XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  data->iterationbox.slider = XmCreateScale(data->iterationbox.box,"Slider",al,ac);
  XtAddCallback(data->iterationbox.slider,XmNvalueChangedCallback,
           AmountOfIterationsHandler,location);
  XtManageChild(data->iterationbox.slider);
  XmStringFree(xmstr);
  /*
   * create OK-button
   */
  ac = 0;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->iterationbox.slider); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
  data->iterationbox.okbutton = XmCreatePushButtonGadget(data->iterationbox.box,
                                                     "OK",al,ac);
  XtAddCallback(data->iterationbox.okbutton,XmNactivateCallback,Unmanage,
           data->iterationbox.box);
  XtManageChild(data->iterationbox.okbutton);
  /*
   * return value of this function
   */
  return(data->iterationbox.box);
}


Widget CreateZoomBox(parent,location)
     Widget parent;
     char location;
{
  int mode,xvalue,yvalue,zvalue;
  Drawdata *data;
  XmString xmstr;
  /*
   * parameter initialization
   */
  data = GetWindowData(location,'p');
  xvalue = data->xoffset - data->width/2;
```

```
yvalue = data->yoffset - data->height/2;
zvalue = 10;
if (location == 'l')
  {
    mode = 0;
    xmstr = XmStringCreate("left area zooming",XmSTRING_DEFAULT_CHARSET);
  }
if (location == 'r')
  {
    mode = 1;
    xmstr = XmStringCreate("right area zooming",XmSTRING_DEFAULT_CHARSET);
  }
/*
 * create zoombox to appear when define zoom area button in render control
 * pulldown pane is pressed
 */
ac = 0;
XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
data->zoombox.box = XmCreateFormDialog(parent,"ZoomBox",al,ac);
XmStringFree(xmstr);
/*
 * create a X-offset scrollbar with range -250 to 250 [pixels]
 */
ac = 0;
xmstr = XmStringCreate(" X-offset [pixels]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,-250); ac++;
XtSetArg(al[ac],XmNmaximum,250); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,xvalue); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
data->zoombox.xoffsetslider = XmCreateScale(data->zoombox.box,"XoffsetSlider",al,ac);
XtAddCallback(data->zoombox.xoffsetslider,XmNdragCallback,ZoomHandler,mode);
XtAddCallback(data->zoombox.xoffsetslider,XmNvalueChangedCallback,ZoomHandler,mode);
XtManageChild(data->zoombox.xoffsetslider);
XmStringFree(xmstr);
/*
 * create a Y-offset scrollbar with range -250 to 250 [pixels]
 */
ac = 0;
xmstr = XmStringCreate(" Y-offset [pixels]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,-250); ac++;
XtSetArg(al[ac],XmNmaximum,250); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,yvalue); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,data->zoombox.xoffsetslider); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
data->zoombox.yoffsetslider = XmCreateScale(data->zoombox.box,"YoffsetSlider",al,ac);
XtAddCallback(data->zoombox.yoffsetslider,XmNdragCallback,ZoomHandler,mode+2);
XtAddCallback(data->zoombox.yoffsetslider,XmNvalueChangedCallback,ZoomHandler,mode+2);
XtManageChild(data->zoombox.yoffsetslider);
XmStringFree(xmstr);
/*
 * create a zoomfactor scrollbar with range 1 to 100 [x]
 */
ac = 0;
xmstr = XmStringCreate(" Zoomfactor [x 100/value]",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNtitleString,xmstr); ac++;
XtSetArg(al[ac],XmNminimum,1); ac++;
XtSetArg(al[ac],XmNmaximum,100); ac++;
XtSetArg(al[ac],XmNwidth,SCALEBOX_WIDTH); ac++;
XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
XtSetArg(al[ac],XmNvalue,zvalue); ac++;
XtSetArg(al[ac],XmNshowValue,TRUE); ac++;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,data->zoombox.yoffsetslider); ac++;
```

```
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  data->zoombox.zoomslider = XmCreateScale(data->zoombox.box,"ZoomSlider",al,ac);
  XtAddCallback(data->zoombox.zoomslider,XmNdragCallback,ZoomHandler,mode+4);
  XtAddCallback(data->zoombox.zoomslider,XmNvalueChangedCallback,ZoomHandler,mode+4);
  XtManageChild(data->zoombox.zoomslider);
  XmStringFree(xmstr);
  /*
   * create OK-button
   */
  ac = 0;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->zoombox.zoomslider); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
  data->zoombox.okbutton = XmCreatePushButtonGadget(data->zoombox.box,"OK",al,ac);
  XtAddCallback(data->zoombox.okbutton,XmNactivateCallback,Unmanage,data->zoombox.box);
  XtAddCallback(data->zoombox.okbutton,XmNactivateCallback,ZoomController,location);
  XtManageChild(data->zoombox.okbutton);
  /*
   * return value of this function
   */
  return(data->zoombox.box);
}


Widget CreateScrollBox(parent,location)
     Widget parent;
     char location;
{
  int mode;
  Drawdata *data;
  Scrolldata *scroll;
  XmString xmstr;
  /*
   * parameter initialization
   */
  data = GetWindowData(location,'p');
  scroll = GetScrollData(location);
  if (location == 'l')
    {
      mode = 0;
      xmstr = XmStringCreate("left text",XmSTRING_DEFAULT_CHARSET);
    }
  if (location == 'r')
    {
      mode = 1;
      xmstr = XmStringCreate("right text",XmSTRING_DEFAULT_CHARSET);
    }
  /*
   * create scrollbox to appear when textual view button in viewedit IFS code
   * pulldown pane is pressed
   */
  ac = 0;
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  data->scrollbox.box = XmCreateFormDialog(parent,"ScrollBox",al,ac);
  XmStringFree(xmstr);
  /*
   * create scrollbar widget
   */
  ac = 0;
  XtSetArg(al[ac],XmNorientation,XmHORIZONTAL); ac++;
  XtSetArg(al[ac],XmNworkWindow,GetWindow(location,'p'));
  data->scrollbox.scrollbar = XmCreateScrollBar(data->scrollbox.box,"ScrollBar",al,ac);
  XtAddCallback(data->scrollbox.scrollbar,XmNvalueChangedCallback,
             ScrollBarMovedHandler,location);
  XtAddCallback(data->scrollbox.scrollbar,XmNdragCallback,
             ScrollBarMovedHandler,location);
  XtManageChild(data->scrollbox.scrollbar);
  /*
   * create OK-button
   */
  ac = 0;
  XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
  XtSetArg(al[ac],XmNtopWidget,data->scrollbox.scrollbar); ac++;
  XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
  XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
  data->scrollbox.okbutton = XmCreatePushButtonGadget(data->scrollbox.box,"OK",al,ac);
  XtAddCallback(data->scrollbox.okbutton,XmNactivateCallback,Unmanage,data->scrollbox.box);
  XtAddCallback(data->scrollbox.okbutton,XmNactivateCallback,ClearPixelArea,location);
  XtManageChild(data->scrollbox.okbutton);
  /*
   * return value of this function
   */
  return(data->scrollbox.box);
}
```

```
/*************************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : collage.c
 * Purpose         : Callback procedure for interactive collage generation
 * Part of         : IFS-CODEC
 * Created on      : June 09, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 *************************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void FileOpenWarning();                /* contained in messages.c    */
void DrawAxes();                       /* contained in draw.c        */
void SaveImageToFile();                /* contained in filehandler.c */
void SaveTransformationToFile();       /* contained in filehandler.c */
Drawdata *GetWindowData();             /* contained in control.c     */
Widget GetWindow();                    /* contained in control.c     */


void ResetTransformSliders(location)
     char location;
{
  Affine *transformation;
  Drawdata *data;
  /*
   * initialize data
   */
  data = GetWindowData(location,'p');
  transformation = &(data->transformation);
  transformation->xscaling = 80;
  transformation->yscaling = 80;
  transformation->xrotation = 0;
  transformation->yrotation = 0;
  transformation->xshift = 0;
  transformation->yshift = 0;
  /*
   * set transformsliders back to initial position
   */
  ac = 0;
  XtSetArg(al[ac],XmNvalue,transformation->xscaling); ac++;
  XtSetValues(data->similitudebox.xscslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,transformation->yscaling); ac++;
  XtSetValues(data->similitudebox.yscslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,transformation->xrotation); ac++;
  XtSetValues(data->similitudebox.xrtslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,transformation->yrotation); ac++;
  XtSetValues(data->similitudebox.yrtslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,transformation->xshift); ac++;
  XtSetValues(data->similitudebox.xshslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,transformation->yshift); ac++;
  XtSetValues(data->similitudebox.yshslider,al,ac);
}


void DetermineProbabilities(location,ext)
     char location,*ext;
{
  char *savefile,*filename;
  float a,b,c,d,det,e,f,maxdet = 0,p,pi,sum = 0;
```

```
int amount = 0,pdum,pmax = 0,tot = 0,zerocount = 0;
FILE *file,*dummyfile;
Drawdata *data;
Widget w;
/*
 * initialize data
 */
pi = PI;
w = GetWindow(location,'p');
data = GetWindowData(location,'p');
savefile = calloc(strlen(data->filename)+5,sizeof(char));
strcpy(savefile,data->filename);
filename = strcat(savefile,ext);
if ((file = fopen(filename,"r")) == NULL)
  {
    FileOpenWarning(w,location);
  }
else
  {
    /*
     * if file could be successfully opened
     */
    dummyfile = fopen("dummy","w");
    while (!feof(file))
     {
       /*
        * get affine transformation from file
        */
       fscanf(file,"%f %f %f %f %f %f %f\n",&a,&b,&c,&d,&e,&f,&p);
       fprintf(dummyfile,"%.3f %.3f %.3f %.3f %.3f %.3f %.3f\n",a,b,c,d,e,f,p);
       /*
        * calculate determinant of transformationmatrix
        */
       det = a*d - b*c;
       /*
        * make determinant absolute
        */
       if (det < 0) det = -det;
       /*
        * totalize all determinantvalues and get maximum determinantvalue
        */
       sum += det;
       if (det > maxdet) maxdet = det;
     }
    fclose(file);
    fclose(dummyfile);
    dummyfile = fopen("dummy","r");
    while (!feof(dummyfile))
         {
           fscanf(dummyfile,"%f %f %f %f %f %f %f\n",&a,&b,&c,&d,&e,&f,&p);
           /*
            * calculate determinant of transformationmatrix
            */
           det = a*d - b*c;
           /*
            * make determinant absolute
            */
           if (det < 0) det = -det;
           /*
            * count total amount of transformations in file
            */
           amount++;
           /*
            * count amount of determinants that are smaller than 0.1% of
            * maximum determinantvalue
            */
           if (det <= 0.001*maxdet) zerocount++;
         }
    file = fopen(filename,"w");
    rewind(dummyfile);
    while (!feof(dummyfile))
     {
       /*
        * get affine transformation from dummyfile
        */
       fscanf(dummyfile,"%f %f %f %f %f %f %f\n",&a,&b,&c,&d,&e,&f,&p);
      /*
       * calculate determinant of transformation matrix
       */
      det = a*d - b*c;
      /*
       * make determinant absolute
       */
      if (det < 0) det = -det;
      /*
       * if determinantvalue is relatively small then set probability to
       * minimum, else calculate probability as ratio of determinantvalue
       * and total of sum of determinants and amount of minimum probability
       * values
       */
      if ((det <= 0.001*maxdet) && (amount != zerocount))
        {
          p = 0.001;
        }
      else
        {
          p = det/(sum + 0.001*zerocount);
        }
      /*
       * determine the maximum p-value and the sum of all p-values
       */
      if (pmax < 1000*p) pmax = 1000*p;
      tot += (int) (1000*p + 0.5);
      /*
       * put affine transformation together with probability in file
       */
      fprintf(file,"%.3f %.3f %.3f %.3f %.3f %.3f %.3f\n",a,b,c,d,e,f,p);
    }
  /*
   * if the sum of the probabilities is not equal to 1.000, adjust the
   * maximum probability, so the relative effect is minimal: thus add or
   * subtract 0.001, since this is the greatest possible round off error
   * that can occur
   */
  if (tot != 1000)
  {
      fclose(file);
      fclose(dummyfile);
       file = fopen(filename,"r");
      dummyfile = fopen("dummy","w");
       while (!feof(file))
        {
          fscanf(file,"%f %f %f %f %f %f %f\n",&a,&b,&c,&d,&e,&f,&p);
          pdum = (int) (1000*p + 0.5);
          if ((pdum >= pmax) && (tot < 1000))
            {
              p += 0.001;
              tot++;
            }
          if ((pdum >= pmax) && (tot > 1000))
            {
              p -= 0.001;
              tot--;
            }
          fprintf(dummyfile,"%.3f %.3f %.3f %.3f %.3f %.3f %.3f\n",a,b,c,d,e,f,p);
        }
      /*
       * copy dummyfile to file
       */
      fclose(file);
      fclose(dummyfile);
       file = fopen(filename,"w");
      dummyfile = fopen("dummy","r");
      while (!feof(dummyfile))
        {
          fscanf(dummyfile,"%f %f %f %f %f %f %f\n",&a,&b,&c,&d,&e,&f,&p);
          fprintf(file,"%.3f %.3f %.3f %.3f %.3f %.3f %.3f\n",a,b,c,d,e,f,p);
        }
  }
  fclose(dummyfile);
```

```
    }
  fclose(file);
  free(savefile);
  savefile = NULL;
}


void TransformSimilitude(location)
     char location;
{
  char *savefile;
  int x,y,xt,yt,h,k;
  double theta,psi;
  float r,s,pi;
  Affine *transformation;
  Drawdata *data;
  FILE *file;
  Widget w;
  /*
   * initialize parameters and try to open file
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  transformation = &(data->transformation);
  pi = PI;
  r = (transformation->xscaling)/100.0;
  s = (transformation->yscaling)/100.0;
  theta = (data->transformation.xrotation)*2*pi/360;
  psi = (data->transformation.yrotation)*2*pi/360;
  h = data->transformation.xshift;
  k = data->transformation.yshift;
  savefile = calloc(strlen(data->filename)+5,sizeof(char));
  strcpy(savefile,data->filename);
  if ((file = fopen(strcat(savefile,".img"),"r")) == NULL)
    {
      FileOpenWarning(w,location);
    }
  else
    {
      /*
       * set foreground to black
       */
      XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
      while (!feof(file))
       {
         /*
          * if file could be sucessfully opened
          */
         fscanf(file,"%d %d ",&xt,&yt);
         /*
          * transform coordinate to virtual origin
          */
         xt = xt - data->xorigin;
         yt = yt - data->yorigin;
         /*
          * convert y from screen coordinates to Carthesian coordinates
          */
         yt = - yt;
         /*
          * apply affine transformation
          */
         x = xt*r*cos(theta)-yt*s*sin(psi)+h;
         y = xt*r*sin(theta)+yt*s*cos(psi)+k;
         /*
          * convert y from Carthesian coordinates to screen coordinates
          */
         y = - y;
         /*
          * transform coordinate to its original position
          */
         x += data->xorigin;
         y += data->yorigin;
         /*
          * set foreground to black and plot point in screen space
          */
```

```
        XDrawPoint(XtDisplay(w),data->pix,data->gc,x,y);
      }
    }
  fclose(file);
  free(savefile);
  savefile = NULL;
}


void ManageSimilitude(w,code,call_data)
     Widget w;
     int code;
     XmScaleCallbackStruct *call_data;
{
  char location;
  int x,y;
  Affine transformation;
  Drawdata *data;
  if (code % 2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  /*
   * determine which slider was moved
   */
  switch(code)
  {
    case 0:
      data->transformation.xscaling = call_data->value;
      break;
    case 2:
      data->transformation.yscaling = call_data->value;
      break;
    case 4:
      data->transformation.xrotation = call_data->value;
      break;
    case 6:
      data->transformation.yrotation = call_data->value;
      break;
    case 8:
      data->transformation.xshift = call_data->value;
      break;
    case 10:
      data->transformation.yshift = call_data->value;
      break;
  }
  /*
   * clear out data->pix
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,
               data->width,data->height);
  /*
   * copy data->col to data->pix
   */
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  XCopyArea(XtDisplay(w),data->col,data->pix,data->gc,0,0,
          data->width,data->height,0,0);
  /*
   * Transform similitude and add it to data->pix
   */
  TransformSimilitude(location);
  /*
   * copy data->pix to screen
   */
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
          data->width,data->height,0,0);
  /*
```

```
   * determine if axes have to be drawn
   */
  if (data->axes) DrawAxes(w,location);
}


void AddSimilitude(w,location)
     Widget w;
     char location;
{
  int x,y;
  unsigned long pixel;
  Affine *transformation;
  Drawdata *data;
  XImage *image;
  /*
   * initialize data
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  transformation = &(data->transformation);
  transformation->xscaling = 80;
  transformation->yscaling = 80;
  transformation->xrotation = 0;
  transformation->yrotation = 0;
  transformation->xshift = 0;
  transformation->yshift = 0;
  /*
   * set sliders back to initial position
   */
  ResetTransformSliders(location);
  /*
   * pixmaps should only be created once
   */
  if (data->firsttime)
    {
      data->firsttime = FALSE;
      /*
       * similitude must be made equal to initial image that is in data->pix
       */
      XCopyArea(XtDisplay(w),data->pix,data->sim,data->gc,0,0,
             data->width,data->height,0,0);
      /*
       * put current contains of data->pix into data->col
       */
      XCopyArea(XtDisplay(w),data->pix,data->col,data->gc,0,0,
             data->width,data->height,0,0);
      /*
       * create image from original drawing located in data->pix
       */
      SaveImageToFile(location,".img");
    }
  /*
   * set foreground to black, transform similitude and draw axes
   */
  TransformSimilitude(location);
  DrawAxes(w,location);
}


void FixSimilitude(w,location)
     Widget w;
     char location;
{
  Affine *transformation;
  Drawdata *data;
  /*
   * initialize data
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  transformation = &(data->transformation);
  /*
   * write transformation to file data->filename.man
   */
  SaveTransformationToFile(location,".man");
  /*
   * copy data->pix to data->col in order to save the collage
   */
  XCopyArea(XtDisplay(w),data->pix,data->col,data->gc,0,0,
          data->width,data->height,0,0);
}


void SaveCollage(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  /*
   * initialize data
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  if (!data->firsttime)
    {
      /*
       * set data->firsttime to TRUE
       */
      data->firsttime = TRUE;
    }
  /*
   * determine probabilities of transformations
   */
  DetermineProbabilities(location,".man");
  /*
   * clear out pixmap
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,
              data->width,data->height);
  /*
   * copy data->col to data->pix in order to be able to do the next step
   */
  XCopyArea(XtDisplay(w),data->col,data->pix,data->gc,0,0,
          data->width,data->height,0,0);
  /*
   * save data->col to data->filename.col
   */
  SaveImageToFile(location,".col");
  /*
   * clear data->pix, data->col, data->sim and screen
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,
              data->width,data->height);
  XFillRectangle(XtDisplay(w),data->col,data->gc,0,0,
              data->width,data->height);
  XFillRectangle(XtDisplay(w),data->sim,data->gc,0,0,
              data->width,data->height);
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
          data->width,data->height,0,0);
  /*
   * determine if axes have to be drawn
   */
  if (data->axes) DrawAxes(w,location);
  /*
   * finally reset transformsliders
   */
  ResetTransformSliders(location);
}
```

```
/*****************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : control.c
 * Purpose         : Controls the operating environment
 * Part of         : IFS-CODEC
 * Created on      : June 18, 1992
 * Created by      : Emile van Duren
 * Last modified on: November 20, 1992
 * Modified by     : Emile van Duren
 * Remarks         : Zoomcontroller and Unzoomcontroller have to be completed
 *****************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void DrawMandelbrotSet();       /* contained in specials.c   */
void DrawText();                /* contained in writer.c     */


void Manage(w,client_data,call_data)
     Widget  w;
     XtPointer client_data;
     XtPointer call_data;
{
  XtManageChild(client_data);
}


void Unmanage(w,client_data,call_data)
     Widget  w;
     XtPointer client_data;
     XtPointer call_data;
{
  XtUnmanageChild(client_data);
}


Textdata *GetTextData(location)
     char location;
{
  Textdata *text;
  if (location == 'l') text = &LText;
  if (location == 'r') text = &RText;
  return(text);
}


Drawdata *GetWindowData(location,which)
     char location,which;
{
  char buffer[10];
  Drawdata *data;
  /*
   * if which == 't' then selected drawable is textarea
   * if which == 'p' then selected drawable is pixelarea
   * if location == 'l' then leftarea
   * if location == 'r' then rightarea
   */
  if ((location == 'l') && (which == 'p')) data = &leftpixeldata;
  if ((location == 'r') && (which == 'p')) data = &rightpixeldata;
  if ((location == 'l') && (which == 't')) data = &lefttextdata;
  if ((location == 'r') && (which == 't')) data = &righttextdata;
  return(data);
}
```

```
Scrolldata *GetScrollData(location)
     char location;
{
  Scrolldata *scroll;
  if (location == 'l') scroll = &LScroll;
  if (location == 'r') scroll = &RScroll;
  return(scroll);
}


Widget GetWindow(location,which)
     char location,which;
{
  Widget w;
  char buffer[10];
  /*
   * if which == 'p' then selected drawable is pixelarea
   * if which == 't' then selected drawable is textarea
   * if location == 'l' then leftarea
   * if location == 'r' then rightarea
   */
  if ((location == 'l') && (which == 'p')) w = leftpixelarea;
  if ((location == 'r') && (which == 'p')) w = rightpixelarea;
  if ((location == 'l') && (which == 't')) w = lefttextarea;
  if ((location == 'r') && (which == 't')) w = righttextarea;
  return(w);
}


void ModeController(w,code)
     Widget  w;
     int code;
{
  Drawdata *data;
  /*
   * if code is even then left, if code is odd then right
   */
  if (code % 2 == 0)
    {
      data = &leftpixeldata;
    }
  else
    {
      data = &rightpixeldata;
    }
  if (code == 0 || code == 1)  data->mode = 'v';
  if (code == 2 || code == 3)  data->mode = 'w';
  if (code == 4 || code == 5)  data->mode = 'i';
  if (code == 6 || code == 7)  data->mode = 'a';
  if (code == 8 || code == 9)  data->mode = 'd';
  if (code == 10 || code == 11) data->mode = 'n';
  if (code == 12 || code == 13) data->mode = 'p';
  if (code == 14 || code == 15) data->mode = 'b';
  if (code == 16 || code == 17) data->mode = 'm';
}


void ClearTextController(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  data = GetWindowData(location,'p');
  if (data->mode == 'w') ClearTextHandler(w,location,14);
  if (data->mode == 'i') ClearTextHandler(w,location,990);
  if (data->mode == 'a') ClearTextHandler(w,location,990);
}


void AxesController(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  /*
```

```
   * toggle axes button is activated, so invert axes operation mode
   * (i.e. if axes are currently displayed then erase them and if
   *  axes are currently not displayed then draw them)
   */
  data = GetWindowData(location,'p');
  data->axes = !data->axes;
}


void Passage(w,code)
     int code;
{
  char location;
  Drawdata *data;
  if (code % 2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  data = GetWindowData(location,'p');
  switch(code)
    {
    case 0:
      firsthelp = TRUE;
      break;
    case 2:
      data->first.filehelp = TRUE;
      break;
    case 4:
      data->first.fileopenwarning = TRUE;
      break;
    case 6:
      data->first.originwarning = TRUE;
      break;
    case 8:
      data->first.tablemessage = TRUE;
      break;
    case 10:
      printmessage = TRUE;
      break;
    case 12:
      copyrightmessage = TRUE;
      break;
    }
}


void ZoomController(w,location)
     Widget w;
     char location;
{
 Drawdata *data;
 data = GetWindowData(location,'p');
 /*
  * set data->unzoom to FALSE and redraw appropriate picture
  */
 data->unzoom = FALSE;
 switch(data->mode)
   {
   case 'd':
     break;
   case 'n':
     break;
   case 'm':
     DrawMandelbrotSet(w,location);
     break;
   }
}
```

```
void UnzoomController(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  data->unzoom = TRUE;
  data->xoffset = 0;
  data->yoffset = 0;
  data->zoomfactor = 1;
  /*
   * clear screen and copy data->pix to screen in order to remove zoombox
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,
              data->width,data->height);
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
          data->width,data->height,0,0);
  /*
   * reset zoomboxsliders
   */
  ac = 0;
  XtSetArg(al[ac],XmNvalue,data->xoffset); ac++;
  XtSetValues(data->zoombox.xoffsetslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,data->yoffset); ac++;
  XtSetValues(data->zoombox.yoffsetslider,al,ac);
  ac = 0;
  XtSetArg(al[ac],XmNvalue,data->zoomfactor); ac++;
  XtSetValues(data->zoombox.zoomslider,al,ac);
  /*
   * determine operation mode and redraw appropriate picture
   */
 switch(data->mode)
  {
  case 'd':
    break;
  case 'n':
    break;
  case 'm':
    DrawMandelbrotSet(w,location);
    break;
  }
}


void CondensationController(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  data = GetWindowData(location,'p');
  data->condensation = !(data->condensation);
  if (data->condensation)
    {
      printf("condensation = TRUE\n",NULL);
    }
  else
    {
      printf("condensation = FALSE\n",NULL);
    }
}


void DrawController(w,location)
     Widget w;
     char location;
{
  Textdata *text;
  Drawdata *data;
  w = GetWindow(location,'t');
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  /*
```

```
   * set erasegrab to false, toggle drawgrab and mode
   */
  data->erasegrab = FALSE;
  data->drawgrab = !data->drawgrab;
  if (data->drawgrab)
    {
      text->mode = "mode         : Viewedit source file (draw)";
      printf("drawmode = TRUE\n",NULL);
    }
  if (!data->drawgrab)
    {
      text->mode = "mode         : Viewedit source file";
      printf("drawmode = FALSE\n",NULL);
    }
  DrawText(location);
}


void EraseController(w,location)
     Widget w;
     char location;
{
  Textdata *text;
  Drawdata *data;
  w = GetWindow(location,'t');
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  /*
   * set drawgrab to false, toggle erasegrab and mode
   */
  data->drawgrab = FALSE;
  data->erasegrab = !data->erasegrab;
  if (data->erasegrab)
    {
      text->mode = "mode         : Viewedit source file (erase)";
      printf("erasemode = TRUE\n",NULL);
    }
  if (!data->erasegrab)
    {
      text->mode = "mode         : Viewedit source file";
      printf("erasemode = FALSE\n",NULL);
    }
  DrawText(location);
}


void ScrollController(w,code)
     Widget w;
     int code;
{
  char location;
  Scrolldata *scroll;
  if (code %2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  scroll = GetScrollData(location);
  if (code == 0) scroll->ext = ".ifs";
  if (code == 2) scroll->ext = ".man";
  if (code == 4) scroll->ext = ".aut";
}


void IdentityController(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  data = GetWindowData(location,'p');
  data->identity = !data->identity;
  if (data->identity)
    {
      printf("identity = TRUE\n",NULL);
    }
  else
    {
      printf("identity = FALSE\n",NULL);
    }
}


void Quit(w,client_data,call_data)
     Widget  w;
     XtPointer client_data;
     XtPointer call_data;
{
  XtCloseDisplay(XtDisplay(w));
  exit(0);
}
```

```
/*****************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : draw.c
 * Purpose          : Program to draw in pixelareas
 * Part of          : IFS-CODEC
 * Created on       : June 11, 1992
 * Created by       : Emile van Duren
 * Last modified on: November 20, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 *****************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void ClearTextHandler();     /* contained in handle.c    */
void DrawText();             /* contained in writer.c    */
Drawdata *GetWindowData();   /* contained in control.c   */
Widget GetWindow();          /* contained in control.c   */


void DrawAxes(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  data = GetWindowData(location,'p');
  w = GetWindow(location,'p');
  /*
   * whipe out screen and restore pixmap to screen
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,
             data->width,data->height);
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
           data->width,data->height,0,0);
  DrawText(location);
  /*
   * only draw axes if data->axes is TRUE and also we are not dealing with
   * either a bifurcation diagram or a mandelbrot set
   */
  if (data->axes && !(data->mode == 'b' || data->mode == 'm'))
    {
      /*
       * set foreground to black
       */
      XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
      /*
       * draw X-axis
       */
      XDrawLine(XtDisplay(w),XtWindow(w),data->gc,0,data->yorigin,
            data->width,data->yorigin);
      XDrawString(XtDisplay(w),XtWindow(w),data->gc,10,
              data->yorigin+15,"X-axis",6);
      /*
       * draw Y-axis
       */
      XDrawLine(XtDisplay(w),XtWindow(w),data->gc,data->xorigin,0,
            data->xorigin,data->height);
      XDrawString(XtDisplay(w),XtWindow(w),data->gc,data->xorigin+10,15,
              "Y-axis",6);
    }
}
```

```
void ClearPixelArea(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  /*
   * initialize data and window
   */
  data = GetWindowData(location,'p');
  w = GetWindow(location,'p');
  /*
   * clear pixelarea and its pixmap, but also the temporary pixmaps that
   * contain the similitudes and collages (data->sim and data->col will
   * be automatically cleared if AddSimilitude is invoked)
   */
  if (XtIsRealized(w))
    {
      data->firsttime = TRUE;
      data->unzoom = TRUE;
      XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
      XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,
                     data->width,data->height);
      XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,
                     data->width,data->height);
    }
  /*
   * draw axes if necessary
   */
  if (data->axes) DrawAxes(w,location);
}


void ClearAll(w,location)
     Widget w;
     char location;
{
  Drawdata *data;
  /*
   * initialize data and window
   */
  data = GetWindowData(location,'p');
  w = GetWindow(location,'p');
  /*
   * clear the pixel area and reset data->npixels
   */
  ClearPixelArea(w,location);
  data->npixels = 0;
  /*
   * clear text in textarea (except origin, startpixel, #iterations and
   * zoomfactor) and draw axes if necessary
   */
  ClearTextHandler(w,location,287);
  DrawText(location);
  if (data->axes) DrawAxes(w,location);
}


void DrawRectangle(w,location,mode)
     Widget w;
     char location,mode;
{
  int left,upper,width,height;
  Drawdata *data;
  /*
   * initialize data and window
   */
  data = GetWindowData(location,'p');
  w = GetWindow(location,'p');
  /*
   * set left upper coordinates and dimensions of rectangle
   */
  width = 2*(data->width)/5;
  height = 2*(data->height)/5;
  left = data->xorigin - width/2;
  upper = data->yorigin - height/2;
  /*
   * set foreground to black, draw rectangle in data->pix and on screen
   */
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  if (mode == 'd')
    {
      /*
       * mode is "dashed" so draw a dashed rectangle
       */
      XSetLineAttributes(XtDisplay(w),data->gc,1,LineOnOffDash,CapNotLast,JoinMiter);
      XDrawRectangle(XtDisplay(w),data->pix,data->gc,left,upper,width,height);
      XDrawRectangle(XtDisplay(w),XtWindow(w),data->gc,left,upper,width,height);
      XSetLineAttributes(XtDisplay(w),data->gc,1,LineSolid,CapNotLast,JoinMiter);
    }
  else
    {
      XDrawRectangle(XtDisplay(w),data->pix,data->gc,left,upper,width,height);
      XDrawRectangle(XtDisplay(w),XtWindow(w),data->gc,left,upper,width,height);
    }
  DrawText(location);
}
```

```
/*************************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : exposures.c
 * Purpose          : Contains function Redisplay
 * Part of          : IFS-CODEC
 * Created on       : June 10, 1992
 * Created by       : Emile van Duren
 * Last modified on: October 05, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 *************************************************************************************/

#include "all.h"


void Redisplay(w,data,call_data)
     Widget  w;
     Drawdata *data;
     XmDrawingAreaCallbackStruct *call_data;
{
  XExposeEvent *event = (XExposeEvent *) call_data->event;
  /*
   * extract the exposed area from the event and copy from the saved pixmap
   * to the drawingarea
   */
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,event->x,event->y,
          event->width,event->height,event->x,event->y);
}
```

```
/*******************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : filehandler.c
 * Purpose         : Contains fileselection, filesave and fileloadprocedures
 * Part of         : IFS-CODEC
 * Created on      : June 18, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 ******************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
char *GetStringFromXmString();     /* contained in strings.c      */
void DrawAxes();                   /* contained in draw.c         */
void FileOpenWarning();            /* contained in messages.c     */
void SaveImageToFile();            /* contained in filehandler.c  */
Drawdata *GetWindowData();         /* contained in control.c      */
Widget GetWindow();                /* contained in control.c      */


void SelectFile(w,location,call_data)
     Widget w;
     char location;
     XmSelectionBoxCallbackStruct *call_data;
{
  Drawdata *data;
  XmString xmstr;
  /*
   * get selected file, convert it from XmString to characterstring
   * and put it into global variable "leftfile" or "rightfile"
   */
  xmstr = call_data->value;
  data = GetWindowData(location,'p');
  data->filename = GetStringFromXmString(xmstr);
}


void SaveImageToFile(location,ext)
     char location,*ext;
{
  char *savefile;
  int x,y;
  unsigned long pixel;
  Drawdata *data;
  FILE *file;
  Widget w;
  XImage *image;
  /*
   * initialize data
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  /*
   * save image to data->filename.ext
   */
  if (data->filename == NULL)
    {
      FileOpenWarning(w,location);
    }
  else
    {
      /*
       * create image from current drawing located in data->pix
       */
      image = XGetImage(XtDisplay(w),data->pix,0,0,data->width,data->height,
                     1,XYPixmap);
      /*
       * determine filename.ext to save image
       */
      savefile = calloc(strlen(data->filename)+5,sizeof(char));
      strcpy(savefile,data->filename);
      if ((file = fopen(strcat(savefile,ext),"w")) == NULL)
       {
         FileOpenWarning(w,location);
         data->firsttime = TRUE;
       }
      else
       {
         /*
          * for all pixels in image
          */
         for (y = 0; y < data->height; y++)
           {
             for (x = 0; x < data->width; x++)
              {
                /*
                 * get pixel and put its coordinates in file if pixel is black
                 */
                pixel = XGetPixel(image,x,y);
                if (pixel == 0) fprintf(file,"%d %d ",x,y);
              }
           }
       }
      fclose(file);
      free(savefile);
      savefile = NULL;
      XDestroyImage(image);
      image = NULL;
    }
}


void SaveFile(w,location)
     Widget w;
     char location;
{
  SaveImageToFile(location,".img");
}


void LoadImageFromFile(location,ext)
     char location,*ext;
{
  char *savefile;
  int x,y;
  unsigned long pixel;
  Drawdata *data;
  FILE *file;
  Widget w;
  XImage *image;
  /*
   * initialize data
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  data->npixels = 0;
  /*
   * save image to data->filename.img
   */
  if (data->filename == NULL)
    {
      FileOpenWarning(w,location);
    }
  else
    {
      /*
       * determine filename.ext to load image from
       */
      savefile = calloc(strlen(data->filename)+5,sizeof(char));
```

```
        strcpy(savefile,data->filename);
        if ((file = fopen(strcat(savefile,ext),"r")) == NULL)
         {
           FileOpenWarning(w,location);
         }
        else
         {
           XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
           while (!feof(file))
             {
               fscanf(file,"%d %d ",&x,&y);
               (data->npixels)++;
               XDrawPoint(XtDisplay(w),data->pix,data->gc,x,y);
             }
           /*
            * copy contents of pixmap to window
            */
           XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
                     data->width,data->height,0,0);
           /*
            * determine if axes have to be drawn
            */
           if (data->axes) DrawAxes(w,location);
         }
        fclose(file);
        free(savefile);
        savefile = NULL;
      }
}


void LoadFile(w,location)
     Widget w;
     char location;
{
  LoadImageFromFile(location,".img");
}


void SaveTransformationToFile(location,ext)
     char location,*ext;
{
  char *savefile;
  double psi,theta;
  float a,b,c,d,e,f,p,pi;
  static Boolean firsttime = TRUE;
  Affine *transformation;
  Drawdata *data;
  FILE *file;
  Widget w;
  /*
   * initialize data
   */
  pi = PI;
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  transformation = &(data->transformation);
  /*
   * if no filename is specified then invoke file open warning
   */
  if (data->filename == NULL)
    {
      FileOpenWarning(w,location);
    }
  else
    {
     /*
      * save transformation to data->filename.ext
      */
      savefile = calloc(strlen(data->filename)+5,sizeof(char));
      strcpy(savefile,data->filename);
      if ((file = fopen(strcat(savefile,ext),"a")) == NULL)
       {
         FileOpenWarning(w,location);
       }
      else
      {
        /*
         * rewind file if this is the first transformation
         * (we want to avoid appending transformations to an
         *  existing file) -------- this doesn't work, why not?
         */
        if (firsttime)
          {
            rewind(file);
            firsttime = FALSE;
          }
        /*
         * transform angles from degrees to radians
         */
        theta = (transformation->xrotation)*2*pi/360;
        psi = (transformation->yrotation)*2*pi/360;
        /*
         * append transformation to savefile
         */
        a = ((transformation->xscaling)/100.0)*cos(theta);
        b = ((transformation->yscaling)/100.0)*sin(-psi);
        c = ((transformation->xscaling)/100.0)*sin(theta);
        d = ((transformation->yscaling)/100.0)*cos(psi);
        e = (transformation->xshift)/1.0;
        f = (transformation->yshift)/1.0;
        p = (transformation->probability)/100.0;
        fprintf(file,"%.3f ",a);
        fprintf(file,"%.3f ",b);
        fprintf(file,"%.3f ",c);
        fprintf(file,"%.3f ",d);
        fprintf(file,"%.3f ",e);
        fprintf(file,"%.3f ",f);
        fprintf(file,"%.3f\n",p);
      }
      fclose(file);
      free(savefile);
      savefile = NULL;
    }
}
```

```
/*****************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : genetic.c
 * Purpose          : Performs automatic IFS-encoding with a genetic algorithm
 * Part of          : IFS-CODEC
 * Created on       : September 17, 1992
 * Created by       : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 *****************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
int SelectGeneticOperator();        /* contained in operators.c    */
void FileOpenWarning();             /* contained in messages.c     */
void IterationHandler();            /* contained in handle.c       */
void MacroCrossOver();              /* contained in operators.c    */
void MicroCrossOver();              /* contained in operators.c    */
void MixedCrossOver();              /* contained in operators.c    */
void Mutate();                      /* contained in operators.c    */
void Jump();                        /* contained in operators.c    */
void Rotate();                      /* contained in operators.c    */
void SaveFileHandler();             /* contained in handle.c       */
void Scale();                       /* contained in operators.c    */
void Translate();                   /* contained in operators.c    */
Drawdata *GetWindowData();          /* contained in control.c      */
Widget GetWindow();                 /* contained in control.c      */


float PixsetDistance(w,mode)
     Widget w;
     int mode;
{
  int col,row;
  float AnotB,BnotA,pixsetdistance;
  long int intersection=0,A=0,B=0;
  unsigned long pixelA,pixelB;
  Drawdata *data;
  XImage *imageA,*imageB;
  /*
   * initialization:
   *
   * mode 1: A in leftdata->pix,  B in rightdata->pix
   * mode 2: A in leftdata->pix,  B in leftdata->sim
   * mode 3: A in leftdata->pix,  B in leftdata->col
   * mode 4: A in rightdata->pix, B in rightdata->sim
   * mode 5: A in rightdata->pix, B in rightdata->col
   */
  switch(mode)
  {
    case 1:
      w = GetWindow('l','p');
      data = GetWindowData('l','p');
      imageA = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                    data->height,1,XYPixmap);
      w = GetWindow('r','p');
      data = GetWindowData('r','p');
      imageB = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                    data->height,1,XYPixmap);
      break;
    case 2:
      w = GetWindow('l','p');
      data = GetWindowData('l','p');
      imageA = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                    data->height,1,XYPixmap);
      imageB = XGetImage(XtDisplay(w),data->sim,0,0,data->width,
                     data->height,1,XYPixmap);
      break;
    case 3:
      w = GetWindow('l','p');
      data = GetWindowData('l','p');
      imageA = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                     data->height,1,XYPixmap);
      imageB = XGetImage(XtDisplay(w),data->col,0,0,data->width,
                     data->height,1,XYPixmap);
      break;
    case 4:
      w = GetWindow('r','p');
      data = GetWindowData('r','p');
      imageA = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                     data->height,1,XYPixmap);
      imageB = XGetImage(XtDisplay(w),data->sim,0,0,data->width,
                     data->height,1,XYPixmap);
      break;
    case 5:
      w = GetWindow('r','p');
      data = GetWindowData('r','p');
      imageA = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                     data->height,1,XYPixmap);
      imageB = XGetImage(XtDisplay(w),data->col,0,0,data->width,
                     data->height,1,XYPixmap);
      break;
  }
  /*
   * for all pixels in image A and image B count #pixels in A, B and AandB
   */
  for (col = 0; col < data->width; col++)
    {
      for (row = 0; row < data->height; row++)
        {
          /*
           * get pixel and if pixel is black
           */
          pixelA = XGetPixel(imageA,col,row);
          pixelB = XGetPixel(imageB,col,row);
          if (pixelA == 0)
            {
              A++;
            }
          if (pixelB == 0)
            {
              B++;
            }
          if (pixelA == 0 && pixelB == 0)
            {
              intersection++;
            }
        }
    }
  /*
   * destroy the images
   */
  XDestroyImage(imageA);
  XDestroyImage(imageB);
  imageA = NULL;
  imageB = NULL;
  /*
   * calculate pixsetdistance as max(|A\B|/|A|,|B\A|/|B|)
   */
  if (A == 0 || B == 0)
    {
      if (A == 0 && B == 0)
        {
          pixsetdistance = -1;
        }
      else
        {
          pixsetdistance = 1;
        }
    }
  else
    {
      /*
       * watch overflow !
       */
      AnotB = (float) (A-intersection)/A;
      BnotA = (float) (B-intersection)/B;
      if  ((int) 1000000*AnotB < (int) 1000000*BnotA)
       {
         pixsetdistance = BnotA;
       }
      else
       {
         pixsetdistance = AnotB;
       }
    }
  /*
   * return value of this function
   */
  return(pixsetdistance);
}


float ChromosomeToTransformation(gene)
      int gene;
{
  float parameter;
  /*
   * convert chromosome values to IFS code; if gene >= 128 then the according
   * IFS code parameter is negative
   */
  if (gene >= 128) gene = 128 - gene;
  parameter = (float) gene/127;
  /*
   * return value of this function
   */
  return(parameter);
}


int TransformationToChromosome(parameter)
      float parameter;
{
  int gene = 0;
  /*
   * convert transformation values to chromosome values as element of {1..255},
   * i.e. if parameter <= 0 then the MSB of the chromosome will be set
   */
  if ((int) 1000000*parameter <= 0)
    {
      parameter = -parameter;
      gene = 128;
    }
  gene += (int) (127*parameter + 0.5);
  /*
   * return value of this function
   */
  return(gene);
}


void TransformAndEvaluateIdentity(location,code,xleft,xright,yupper,ylower,imageA,display)
      char location;
      int xleft,xright,yupper,ylower;
      Boolean display;
      Chromosome *code;
      XImage *imageA;
{
  float a,b,c,d,e,f,xt,yt,AnotB,BnotA,pixsetdistance;
  int col,i,k,row,x,y;
  long int intersection=0,A=0,B=0;
  unsigned long pixelA,pixelB;
  Drawdata *data;
  Widget w;
  XImage *imageB;
```

```
/*
 * initialization
 */
w = GetWindow(location,'p');
data = GetWindowData(location,'p');
XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
/*
 * convert chromosome values to IFS codes
 */
a = ChromosomeToTransformation(code->a);
b = ChromosomeToTransformation(code->b);
c = ChromosomeToTransformation(code->c);
d = ChromosomeToTransformation(code->d);
e = ChromosomeToTransformation(code->e);
f = ChromosomeToTransformation(code->f);
if (display)
  {
    /*
     * clear out data->sim
     */
    XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
    XFillRectangle(XtDisplay(w),data->sim,data->gc,0,0,data->width,
                   data->height);
    XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  }
/*
 * get image from data->sim
 */
imageB = XGetImage(XtDisplay(w),data->sim,D,0,data->width,data->height,
                   1,XYPixmap);
/*
 * for all pixels in image
 */
for (row = yupper; row < ylower; row++)
  {
    for (col = xleft; col < xright; col++)
      {
        /*
         * get pixel and if pixel is black
         */
        pixelA = XGetPixel(imageA,col,row);
        if (pixelA == 0)
          {
            /*
             * update amount of pixels in target image
             */
            A++;
            /*
             * transform coordinate to virtual origin
             */
            xt = col - data->xorigin;
            yt = row - data->yorigin;
            /*
             * convert y from screen coordinates to Carthesian coordinates
             */
            yt = -yt;
            /*
             * apply affine transformation
             */
            x = a*xt + b*yt + e*250;
            y = c*xt + d*yt + f*250;
            /*
             * convert y from Carthesian coordinates to screen coordinates
             */
            y = -y;
            /*
             * transform coordinate to original position
             */
            x += data->xorigin;
            y += data->yorigin;
            /*
             * only plot pixel in data->sim if it is located within window
             * and it is not already plotted (i.e. it is possible that in
             * case of shrinking several pixels are plotted on the same
             * coordinate)
```

```
           */
           if (x >= 0 && x < data->width && y >= 0 && y < data->height)
           {
             pixelA = XGetPixel(imageA,x,y);
             pixelB = XGetPixel(imageB,x,y);
             if (pixelB !=0)
               {
                 if (display) XDrawPoint(XtDisplay(w),data->sim,data->gc,x,y);
                 XPutPixel(imageB,x,y,0);
                 /*
                  * update amount of pixels in approximation image
                  */
                 B++;
                 if (pixelA == 0) intersection++;
               }
           }
         }
     }
   }
 if (display)
   {
     /*
      * copy data->sim to screen
      */
     XCopyArea(XtDisplay(w),data->sim,XtWindow(w),data->gc,0,0,
           data->width,data->height,0,0);
   }
 /*
  * destroy imageB
  */
 XDestroyImage(imageB);
 imageB = NULL;
 /*
  * calculate pixsetdistance as max(|A\B|/|A|,|B\A|/|B|), where the target
  * image A is located in data->pix and the approximation image B in data->sim
  */
 if (A == 0 || B == 0)
   {
     if (A == 0 && B == 0)
     {
       pixsetdistance = -1;
     }
     else
     {
       pixsetdistance = 1;
     }
   }
 else
   {
     /*
      * watch overflow !
      */
     AnotB = (float) (A-intersection)/A;
     BnotA = (float) (B-intersection)/B;
     if  ((int) 1000000*AnotB < (int) 1000000*BnotA)
     {
       pixsetdistance = BnotA;
     }
     else
     {
       pixsetdistance = AnotB;
     }
   }
 /*
  * calculate fitness
  */
 code->fitness = 1 - pixsetdistance;
}


void TransformAndEvaluateCollage(location,code,xleft,xright,yupper,ylower,imageA,imageC,N,display)
     char location;
     int xleft,xright,yupper,ylower,N;
     Boolean display;
     Chromosome *code;
```

```
    XImage *imageA,*imageC;
{
  float borderreward,fillreward,sizereward,strongpenalty,weakpenalty;
  float a,b,c,d,e,f,xt,yt;
  int col,offset=1,row,x,y;
  unsigned long A=0,B=0,C=0,AandBandC=0,AnotC=0,AnotCandB=0,BnotA=0;
  unsigned long pixelA,pixelB,pixelC;
  Drawdata *data;
  Widget w;
  XImage *imageB;
  /*
   * initialization
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  /*
   * convert chromosome values to IFS codes
   */
  a = ChromosomeToTransformation(code->a);
  b = ChromosomeToTransformation(code->b);
  c = ChromosomeToTransformation(code->c);
  d = ChromosomeToTransformation(code->d);
  e = 250*ChromosomeToTransformation(code->e);
  f = 250*ChromosomeToTransformation(code->f);
  if (display),
    {
      /*
       * clear out data->sim
       */
      XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
      XFillRectangle(XtDisplay(w),data->sim,data->gc,0,0,data->width,
                     data->height);
      XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
    }
  /*
   * A = target image contained in data->pix
   * B = current similitude contained in data->sim
   * C = current collage contained in data->col
   *
   * borderreward   = ((A-C)andB)/B
   * fillreward     = ((A-C)andB)/(A-C)
   * sizereward     = ((N+1)*B)/A      for B/A in [0,1-(N/(N+1)))
   *                = (1+1/N)*(1-B/A) for B/A in [(N/(N+1)),1]
   * strongpenalty  = (B-A)/B
   * weakpenalty    = (AandBandC)/B
   *
   * get image from data->sim
   */
  imageB = XGetImage(XtDisplay(w),data->sim,0,0,data->width,data->height,
                     1,XYPixmap);
  /*
   * for all pixels in image
   */
  for (row = yupper; row < ylower; row++)
    {
      for (col = xleft; col < xright; col++)
        {
          pixelA = XGetPixel(imageA,col,row);
          if (pixelA == 0)
            {
              /*
               * update amount of pixels in target image
               */
              A++;
              /*
               * transform coordinate to virtual origin
               */
              xt = col - data->xorigin;
              yt = row - data->yorigin;
              /*
               * convert y from screen coordinates to Carthesian coordinates
               */
              yt = -yt;
              /*
            * apply affine transformation
            */
           x = a*xt + b*yt + e;
           y = c*xt + d*yt + f;
           /*
            * convert y from Carthesian coordinates to screen coordinates
            */
           y = -y;
           /*
            * transform coordinate to original position
            */
           x += data->xorigin;
           y += data->yorigin;
           /*
            * only plot pixel in data->sim if it is located within window
            * and it is not already plotted (i.e. it is possible that in
            * case of shrinking several pixels are plotted on the same
            * coordinate)
            */
           if (x >= 0 && x < data->width && y >= 0 && y < data->height)
            {
              /*
               * update amount of pixels left to be covered
               */
              pixelC = XGetPixel(imageC,col,row);
              if (pixelC != 0) AnotC++;
              pixelA = XGetPixel(imageA,x,y);
              pixelB = XGetPixel(imageB,x,y);
              pixelC = XGetPixel(imageC,x,y);
              if (pixelB != 0)
                {
                  if (display) XDrawPoint(XtDisplay(w),data->sim,data->gc,x,y);
                  XPutPixel(imageB,x,y,0);
                  /*
                   * update amount of pixels in approximation image
                   */
                  B++;
                  if (pixelA == 0)
                   {
                     if (pixelC == 0)
                       {
                         AandBandC++;
                       }
                     else
                       {
                         AnotCandB++;
                       }
                   }
                  else
                   {
                     BnotA++;
                   }
                }
            }
         }
     }
   }
 if (display)
   {
     /*
      * copy data->sim to screen
      */
     XCopyArea(XtDisplay(w),data->sim,XtWindow(w),data->gc,0,0,
             data->width,data->height,0,0);
   }
 /*
  * destroy imageB
  */
 XDestroyImage(imageB);
 imageB = NULL;
 /*
  * calculate penalties and rewards (watch overflow !)
  */
 if (A == 0)
   {
        sizereward = 0.0;
      }
    else
      {
        /*
         * we want smaller and smaller copies as the process proceeds (to
         * influence the size of the pictures, adjust offset)
         */
        if ((int) 100*B/A < (int) (100.0/(N+offset)))
        {
          sizereward = (float) ((N+1)*B)/A;
        }
        else
        {
          if ((int) 100*B/A < 100)
            {
              sizereward = (float) (1.0 + 1.0/N)*(1.0 - (float) B/A);
            }
          else
            {
              sizereward = 0.0;
            }
        }
      }
    if (AnotC == 0)
      {
        fillreward = 0.0;
      }
    else
      {
        fillreward = (float) (AnotCandB)/AnotC;
      }
    if (B == 0)
      {
        borderreward  = 0.0;
        strongpenalty = 1.0;
        weakpenalty   = 1.0;
      }
    else
      {
        borderreward  = (float) (AnotCandB)/B;
        strongpenalty = (float) (BnotA)/B;
        weakpenalty   = (float) (AandBandC)/B;
      }
    /*
     * calculate fitness (prevent the first transformation to become the
     * identity mapping by sizereward and force the following similitudes
     * to be smaller and smaller copies to be able to get a good tiling
     */
    if (N==1)
      {
        code->fitness = (1.0 + sizereward - strongpenalty)/2.0;
      }
    else
      {
        code->fitness = (N*sizereward + borderreward - (N+1)*strongpenalty)/((float) N + 1.0);
      }
    /*
     * just to make sure the fitness is an element of (0,1) (this is necessary
     * because the fitness will be used to represent the relative probability
     * with which the member is choosen to be parent for the next generation)
     */
    if ((int) 1000000*(code->fitness) <= 0)
      {
        code->fitness = 0.000001;
      }
    if ((int) 1000000*(code->fitness) >= 1000000)
      {
        code->fitness = 0.999999;
      }
}
```

```
void MergeMemberIntoPopulation(array,member)
     Chromosome *member,array[];
{
  int i = POPULATION_SIZE-1;
  /*
   * merge member into population such that all members within population
   * remain sorted in best fitness order
   */
  while (i > 0 && (int) 1000000*member->fitness > (int) 1000000*array[i-1].fitness)
    {
      array[i].a = array[i-1].a;
      array[i].b = array[i-1].b;
      array[i].c = array[i-1].c;
      array[i].d = array[i-1].d;
      array[i].e = array[i-1].e;
      array[i].f = array[i-1].f;
      array[i].fitness = array[i-1].fitness;
      i--;
    }
  array[i].a = member->a;
  array[i].b = member->b;
  array[i].c = member->c;
  array[i].d = member->d;
  array[i].e = member->e;
  array[i].f = member->f;
  array[i].fitness = member->fitness;
}


void InitializePopulation(location,wtemp,wnext,xleft,xright,yupper,ylower,imageA,display)
     char location;
     int xleft,xright,yupper,ylower;
     Boolean display;
     Chromosome wtemp[],wnext[];
     XImage *imageA;
{
  int col,i,k,mode,row,start,startpoint[POPULATION_SIZE];
  int count,xfixed[POPULATION_SIZE],yfixed[POPULATION_SIZE];
  float a,b,c,d,e,f;
  unsigned long pixelA;
  Drawdata *data;
  Widget w;
  /*
   * initialization
   */
  if (location == 'l')
    {
      mode = 2;
      w = GetWindow('l','p');
      data = GetWindowData('l','p');
    }
  if (location == 'r')
    {
      mode = 4;
      w = GetWindow('r','p');
      data = GetWindowData('r','p');
    }
  for (i = 0; i < POPULATION_SIZE; i++)
    {
      startpoint[i] = 0;
      wtemp[i].fitness = 0.0;
      xfixed[i] = 0;
      yfixed[i] = 0;
    }
  /*
   * take POPULATION_SIZE random points from target_image T and make those
   * points the fixed points of the according affine transformations of the
   * initial population
   */
  for (k = 0; k < POPULATION_SIZE; k++)
    {
      start = (int) (rand())*(data->npixels)/32767;
      i = POPULATION_SIZE-1;
      while (i > 0 && start > startpoint[i-1])
       {
        startpoint[i] = startpoint[i-1];
        i--;
      }
      startpoint[i] = start;
    }
  /*
   * determine the coordinates of the fixed points which must be elements
   * of the pixelset of the target image
   */
  count = data->npixels;
  i=0;
  for (row = yupper; row < ylower; row++)
    {
      for (col = xleft; col < xright; col++)
       {
         pixelA = XGetPixel(imageA,col,row);
         /*
          * fixed point must be element of target_image
          */
         if (pixelA == 0)
           {
             count--;
             if (startpoint[i] == count)
              {
                xfixed[i] = col - data->xorigin;
                yfixed[i] = data->yorigin - row;
                i++;
              }
           }
       }
    }
  /*
   * generate the transformation parameters for the POPULATION_SIZE chromosomes
   * of the first generation
   *
   * each time generate 4 transformation parameters a,b,c,d each as element of
   * {1..255} and calculate the parameters e,f from a,b,c,d and the fixed
   * points xfixed[i] and yfixed[i]
   */
  for (i = 0; i < POPULATION_SIZE; i++)
    {
      wnext[i].a = (int) (rand()/129.0 + 0.5);
      wnext[i].b = (int) (rand()/129.0 + 0.5);
      wnext[i].c = (int) (rand()/129.0 + 0.5);
      wnext[i].d = (int) (rand()/129.0 + 0.5);
      a = ChromosomeToTransformation(wnext[i].a);
      b = ChromosomeToTransformation(wnext[i].b);
      c = ChromosomeToTransformation(wnext[i].c);
      d = ChromosomeToTransformation(wnext[i].d);
      e = (float) (((1 - a)*xfixed[i] - b*yfixed[i])/250.0);
      f = (float) (((1 - d)*yfixed[i] - c*xfixed[i])/250.0);
      wnext[i].e = TransformationToChromosome(e);
      wnext[i].f = TransformationToChromosome(f);
      if (wnext[i].a == 0) wnext[i].a = 128;
      if (wnext[i].b == 0) wnext[i].b = 128;
      if (wnext[i].c == 0) wnext[i].c = 128;
      if (wnext[i].d == 0) wnext[i].d = 128;
      if (wnext[i].e == 0) wnext[i].e = 128;
      if (wnext[i].f == 0) wnext[i].f = 128;
      /*
       * transform target image by current member, evaluate the result and
       * merge the member into the population according to its fitness
       */
      TransformAndEvaluateIdentity(location,&wnext[i],xleft,xright,yupper,ylower,imageA,display);
      MergeMemberIntoPopulation(wtemp,&wnext[i]);
    }
  /*
   * reset the random number generator with a new seed
   */
  srand(wnext[i].a);
}
```

```
int SelectParent(wtemp)
     Chromosome wtemp[];
{
  float totalfitness = 0;
  int k;
  unsigned long p=0,seed;
  Boolean found = FALSE;
  /*
   * determine total fitness
   */
  for (k = 0; k < POPULATION_SIZE; k++)
    {
      totalfitness += wtemp[k].fitness;
    }
  /*
   * roulette wheel parent selection
   */
  seed = rand();
  k = 0;
  while (k < POPULATION_SIZE && !found)
    {
      p += 32767*((wtemp[k].fitness)/totalfitness);
      if (seed <= p) found = TRUE;
      k++;
    }
  /*
   * return value of this function
   */
  return(k-1);
}


void AutomaticIFSEncoding(w,location)
     Widget w;
     char location;
{
  /*
   * chromosome : (a,b,c,d,e,f,fitness);
   * genes      : a,b,c,d,e,f (elements of {1..255})
   * nucleotides: 8 binary digits within genes (1 sign bit, 7 data bits)
   */
  char *savefile,*monitor = "monitor.dat",*ext = ".aut",fbuf[BUFSIZ];
  float a,b,c,d,e,f,p,xt,yt,pixsetdistance;
  float p1 = P_MACRO,p2 = P_MICRO,p3 = P_MIXED,p4 = P_MUTATE,p5 = P_JUMP;
  float p6 = P_SCALE,p7 = P_ROTATE,p8 = P_TRANSLATE,totalfitness,hours;
  int generation,i,index,j,k,mode,N=1,nmax,x,xleft,xright,y,yupper,ylower;
  int col,row,parent1,parent2,selection,n,seconds;
  int c1=0,c2=0,c3=0,c4=0,c5=0,c6=0,c7=0,c8=0;
  unsigned long pixelA,starttime,endtime;
  Boolean debug = DEBUG,display = DISPLAY,extended = EXTENDED,skip = FALSE;
  Chromosome child[2],wtemp[POPULATION_SIZE],wnext[POPULATION_SIZE];
  Drawdata *data;
  FILE *file;
  XImage *imageA,*imageC;
  /*
   * put current amount of seconds since 1970 in starttime
   */
  time(&starttime);
  /*
   * initialization (next line must be outcommented for debugging purposes,
   * because then each run will generate the same sequence of random numbers)
   */
  srand(starttime);
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  data->npixels = 0;
  /*
   * if we want to calculate the identity mapping of a target image then
   * nmax must be set to 1, i.e. only 1 affine transformation
   */
  if (data->identity)
    {
      nmax = 1;
    }
```

```
else
  {
    nmax = MAX_TRANSFORMATIONS;
  }
xleft = data->width;
xright = 0;
yupper = data->height;
ylower = 0;
if (location == 'l') mode = 2;
if (location == 'r') mode = 4;
/*
 * only proceed further if monitor file could be successfully opened
 */
if ((file = fopen(monitor,"w")) == NULL)
  {
    FileOpenWarning(w,location);
    data->firsttime = TRUE;
    skip = TRUE;
  }
else
  {
    /*
     * set vbuffer to force periodic flushing to file monitor.dat and
     * put savefile in text window
     */
    setvbuf(file,fbuf,_IOLBF,BUFSIZ);
    fclose(file);
    SaveFileHandler(w,location,ext);
  }
/*
 * data->pix: contains target_image
 * data->sim: contains evaluated_image (transformed target image under
 *            currently evaluated transformation)
 * data->col: contains collage_image (union of transformations selected to be
 *            part of the IFS code and applied on target_image so far)
 * clear out data->sim and data->col
 */
XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
XFillRectangle(XtDisplay(w),data->sim,data->gc,0,0,data->width,
          data->height);
XFillRectangle(XtDisplay(w),data->col,data->gc,0,0,data->width,
          data->height);
XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
/*
 * get target image from data->pix and initialize collage image to zero
 */
imageA = XGetImage(XtDisplay(w),data->pix,0,0,data->width,data->height,
               1,XYPixmap);
imageC = XGetImage(XtDisplay(w),data->col,0,0,data->width,data->height,
               1,XYPixmap);
/*
 * if target image is the empty set then quit immediately
 */
if (!skip && (int) PixsetDistance(w,mode) < 0)
  {
    skip = TRUE;
  }
else
  {
    /*
     * for all pixels in image
     */
    for (row = 0; row < data->height; row++)
    {
      for (col = 0; col < data->width; col++)
        {
          /*
           * get pixel and if pixel is black
           */
          pixelA = XGetPixel(imageA,col,row);
          if (pixelA == 0)
           {
             /*
              * update amount of pixels in target image
              */
          data->npixels++;
          /*
           * calculate bounding box of target image
           */
          if (col < xleft) xleft = col;
          if (col > xright) xright = col;
          if (row < yupper) yupper = row;
          if (row > ylower) ylower = row;
        }
      }
    }
    /*
     * update file monitor.dat
     */
    file = fopen(monitor,"a");
    fprintf(file,"file name                                    : %s\n",data->filename);
    if (data->identity)
    {
      fprintf(file,"searching to find                           : identity\n",NULL);
    }
    else
    {
      fprintf(file,"searching to find                           : collage\n",NULL);
    }
    fprintf(file,"operator probabilities: MacroCrossover: %3.2f\n",p1);
    fprintf(file,"                        MicroCrossover: %3.2f\n",p2);
    fprintf(file,"                        MixedCrossover: %3.2f\n",p3);
    fprintf(file,"                        Mutate        : %3.2f\n",p4);
    fprintf(file,"                        Jump          : %3.2f\n",p5);
    fprintf(file,"                        Scale         : %3.2f\n",p6);
    fprintf(file,"                        Rotate        : %3.2f\n",p7);
    fprintf(file,"                        Translate     : %3.2f\n",p8);
    fprintf(file,"maximum encoding error (epsilon)      : %d [percent]\n",EPSILON);
    fprintf(file,"maximum amount of generations         : %d\n",MAX_GENERATIONS);
    fprintf(file,"population size                       : %d\n\n",POPULATION_SIZE);
    fclose(file);
  }
/*
 * while approximation has not been accomplished with required accuracy,
 * i.e. the pixset distance between the target image T in data->pix and
 * the union of the transformed similitudes of T according to the trans-
 * formations found so far in data->col exceeds epsilon
 *
 * since data->col is empty and data->pix is not empty if T is not the
 * empty set, the pixsetdistance will always be 1, and thus greater than
 * EPSILON, which is an element of [0,1), so the while loop will always
 * be entered
 */
while (N<=nmax && !skip && (int) (100*PixsetDistance(w,mode+1)) > EPSILON)
  {
    /*
     * update amount of generations in text window as if it is an iteration
     */
    IterationHandler(location,-1);
    printf("affine transformation number %d\n",N);
    generation = 0;
    totalfitness = 0.0;
    /*
     * initialize an amount of POPULATION_SIZE transformations wtemp[i]
     * randomly (or with a heuristic) to form POPULATION N
     */
    InitializePopulation(location,wtemp,wnext,xleft,xright,yupper,ylower,imageA,display);
    /*
     * update file monitor.dat
     */
    file = fopen(monitor,"a");
    fprintf(file,"transformation: %d\n",N);
    fprintf(file,"generation    : %d (initialization)\n",generation);
    fprintf(file,"member    a   b   c   d   e   f    fitness\n",NULL);
    for (k = 0; k < POPULATION_SIZE; k++)
    {
      fprintf(file,"%3d      %3d %3d %3d %3d %3d %3d   %8f\n",k+1,wtemp[k].a,
            wtemp[k].b,wtemp[k].c,wtemp[k].d,wtemp[k].e,wtemp[k].f,
            wtemp[k].fitness);
      totalfitness += wtemp[k].fitness;
    }
    fprintf(file,"total fitness = %f\n\n",totalfitness);
    fclose(file);
    generation++;
    /*
     * initialize next cycle of genetic simulation to obtain an additional
     * transformation for the IFS code
     */
    while (generation <= MAX_GENERATIONS && (int) 100*wtemp[0].fitness < (100 - EPSILON))
     {
       totalfitness = 0.0;
       /*
        * update amount of generations in text window as if it is an iteration
        */
       IterationHandler(location,generation-1);
       /*
        * clear out next generation array
        */
       for (k = 0; k < POPULATION_SIZE; k++)
         {
           wnext[k].fitness = 0.0;
         }
       /*
        * apply all transformations in wtemp to target image in data->pix
        * and put them in data->sim
        */
       for (k = 0; k < POPULATION_SIZE; k += 2)
         {
           /*
            * select 2 parents according to roulette wheel selection
            */
           parent1 = SelectParent(wtemp);
           parent2 = SelectParent(wtemp);
           /*
            * select a genetic operator to be applied on both parents
            */
           selection = SelectGeneticOperator();
           /*
            * update file monitor.dat (next 3 lines are meant for debugging
            * purposes)
            */
           if (debug)
            {
              file = fopen(monitor,"a");
              fprintf(file,"parent1:%3d  parent2:%3d  operator:%2d  ",parent1+1,parent2+1,selection);
              fclose(file);
            }
           /*
            * apply selected genetic operator and create 2 childs
            */
           switch(selection)
           {
             case 1:
              MacroCrossOver(wtemp[parent1],wtemp[parent2],child[0],child[1]);
              c1++;
              break;
             case 2:
              MicroCrossOver(wtemp[parent1],wtemp[parent2],child[0],child[1]);
              c2++;
              break;
             case 3:
              MixedCrossOver(wtemp[parent1],wtemp[parent2],child[0],child[1]);
              c3++;
              break;
             case 4:
              Mutate(wtemp[parent1],child[0]);
              Mutate(wtemp[parent2],child[1]);
              c4++;
              break;
             case 5:
              Jump(wtemp[parent1],child[0]);
              Jump(wtemp[parent2],child[1]);
              c5++;
              break;
             case 6:
```

```
       Scale(wtemp[parent1],child[0]);
       Scale(wtemp[parent2],child[1]);
       c6++;
       break;
      case 7:
       Rotate(wtemp[parent1],child[0]);
       Rotate(wtemp[parent2],child[1]);
       c7++;
       break;
      case 8:
       Translate(wtemp[parent1],child[0]);
       Translate(wtemp[parent2],child[1]);
       c8++;
       break;
    }
    /*
     * transform target image according to current child chromosome,
     * evaluate quality of this child chromosome and assign fitness
     */
    for (j = 0; j < 2; j++)
     {
       if (data->identity)
         {
           TransformAndEvaluateIdentity(location,&child[j],xleft,xright,
                                  yupper,ylower,imageA,display);
         }
       else
         {
           TransformAndEvaluateCollage(location,&child[j],xleft,xright,
                                 yupper,ylower,imageA,imageC,N,display);
         }
       /*
        * update file monitor.dat (next 9 lines are meant for
        * debugging purposes)
        */
       if (debug)
         {
           file = fopen(monitor,"a");
           if ( j == 0)
            {
              fprintf(file,"child    a   b   c   d   e   f    fitness\n",NULL);
            }
           fprintf(file,"%3d      %3d %3d %3d %3d %3d %3d   %8f\n",
                 k+j+1,child[j].a,child[j].b,child[j].c,child[j].d,
                 child[j].e,child[j].f,child[j].fitness);
           fclose(file);
         }
       /*
        * merge children in sorted list
        */
       MergeMemberIntoPopulation(wnext,&child[j]);
     }
  }
/*
 * transfer best 10% of current population unmodified to next generation
 * (the +1 is added to assure that at least one member will be
 * transferred, which is important for populations < 10)
 */
for (i = 0; i < (int) (POPULATION_SIZE/10 + 1); i++)
  {
    MergeMemberIntoPopulation(wnext,&wtemp[i]);
  }
/*
 * make children in wnext the new generation in wtemp and
 * update file monitor.dat
 */
file = fopen(monitor,"a");
if (generation <= MAX_GENERATIONS)
  {
    fprintf(file,"generation : %d\n",generation);
    fprintf(file,"member   a   b   c   d   e   f    fitness\n",NULL);
  }
for (k = 0; k < POPULATION_SIZE; k++)
  {
    wtemp[k].a = wnext[k].a;
```

```
        wtemp[k].b = wnext[k].b;
        wtemp[k].c = wnext[k].c;
        wtemp[k].d = wnext[k].d;
        wtemp[k].e = wnext[k].e;
        wtemp[k].f = wnext[k].f;
        wtemp[k].fitness = wnext[k].fitness;
        if (extended || k == 0 || generation == MAX_GENERATIONS)
         {
          fprintf(file,"%3d     %3d %3d %3d %3d %3d %3d    %8f\n",k+1,
                wnext[k].a,wnext[k].b,wnext[k].c,wnext[k].d,
                wnext[k].e,wnext[k].f,wnext[k].fitness);
         }
        totalfitness += wnext[k].fitness;
      }
    fprintf(file,"total fitness = %f\n\n",totalfitness);
    fclose(file);
    generation++;
  }
  file = fopen(monitor,"a");
  fprintf(file,"#operator selections  : MacroCrossover: %d\n",c1);
  fprintf(file,"                        MicroCrossover: %d\n",c2);
  fprintf(file,"                        MixedCrossover: %d\n",c3);
  fprintf(file,"                        Mutate        : %d\n",c4);
  fprintf(file,"                        Jump          : %d\n",c5);
  fprintf(file,"                        Scale         : %d\n",c6);
  fprintf(file,"                        Rotate        : %d\n",c7);
  fprintf(file,"                        Translate     : %d\n\n",c8);
  fclose(file);
  c1=0;c2=0;c3=0;c4=0;c5=0;c6=0;c7=0;c8=0;
  /*
   * add best found transformation of this cycle of generations as
   * transformation number N in filename.aut
   */
  savefile = calloc(strlen(data->filename)+5,sizeof(char));
  strcpy(savefile,data->filename);
  if ((file = fopen(strcat(savefile,ext),"a")) == NULL)
   {
    FileOpenWarning(w,location);
    data->firsttime = TRUE;
   }
  else
   {
    a = ChromosomeToTransformation(wtemp[0].a);
    b = ChromosomeToTransformation(wtemp[0].b);
    c = ChromosomeToTransformation(wtemp[0].c);
    d = ChromosomeToTransformation(wtemp[0].d);
    e = 250*ChromosomeToTransformation(wtemp[0].e);
    f = 250*ChromosomeToTransformation(wtemp[0].f);
    p = 0.0;
    fprintf(file,"%.3f %.3f %.3f %.3f %.3f %.3f %.3f\n",a,b,c,d,e,f,p);
    fclose(file);
    /*
     * transform target image in data->pix under best transformation found in
     * generation MAX_GENERATIONS of this N-cycle, and add the results to the
     * collage in data->col
     */
    for (row = yupper; row < ylower; row++)
      {
        for (col =  xleft; col < xright; col++)
         {
          /*
           * get pixel and if pixel is black
           */
          pixelA = XGetPixel(imageA,col,row);
          if (pixelA == 0)
            {
             /*
              * transform coordinate to virtual origin
              */
             xt = col - data->xorigin;
             yt = row - data->yorigin;
             /*
              * convert y from screen coordinates to Carthesian coordinates
              */
             yt = -yt;
```

```
                    /*
                     * apply affine transformation
                     */
                    x = a*xt + b*yt + e;
                    y = c*xt + d*yt + f;
                    /*
                     * convert y from Carthesian coordinates to screen coordinates
                     */
                    y = -y;
                    /*
                     * transform coordinate to original position
                     */
                    x += data->xorigin;
                    y += data->yorigin;
                    /*
                     * draw point in collage
                     */
                    XDrawPoint(XtDisplay(w),data->col,data->gc,x,y);
                  }
              }
            }
      }
      /*
       * free the savefile
       */
      free(savefile);
      savefile = NULL;
      /*
       * get updated collage image from data->col
       */
      imageC = XGetImage(XtDisplay(w),data->col,0,0,data->width,data->height,
                  1,XYPixmap);
      /*
       * increment the current affine transformation that is sought for
       */
      N++;
    }
  /*
   * destroy images
   */
  XDestroyImage(imageA);
  XDestroyImage(imageC);
  imageA = NULL;
  imageC = NULL;
  /*
   * calculate pixsetdistance between target image and collage
   */
  pixsetdistance = PixsetDistance(w,mode+1);
  /*
   * copy best found member to data->pix and to screen
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,data->width,
            data->height);
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  XCopyArea(XtDisplay(w),data->col,data->pix,data->gc,0,0,
        data->width,data->height,0,0);
  XCopyArea(XtDisplay(w),data->col,XtWindow(w),data->gc,0,0,
        data->width,data->height,0,0);
  /*
   * put pixsetdistance, amount of affine transformations and cpu time used in
   * monitor file
   */
  if (!skip)
    {
      time(&endtime);
      seconds = (int) (endtime - starttime);
      hours = seconds/3600.0;
      file = fopen(monitor,"a");
      fprintf(file,"pixsetdistance(target,best member)     : %7.6f\n\n",pixsetdistance);
      fprintf(file,"#affine transformations in IFS-code    : %d\n\n",N-1);
      fprintf(file,"total CPU time used = %d [seconds] = %2.2f [hours]\n\n",seconds,hours);
      fclose(file);
    }
}
```

```
/******************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : graphics.c
 * Purpose         : Creates graphics contexts for drawingareas
 * Part of         : IFS-CODEC
 * Created on      : June 10, 1992
 * Created by      : Emile van Duren
 * Last modified on: November 18, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 ******************************************************************************/

#include "all.h"


void InitGraphics(w,data)
     Widget  w;
     Drawdata *data;
{
  /*
   * determine amount of available colors
   */
  data->ncolors = XDisplayCells(XtDisplay(w),XDefaultScreen(XtDisplay(w)));
  /*
   * get the size of the drawingara
   */
  ac = 0;
  XtSetArg(al[ac],XtNwidth,&data->width); ac++;
  XtSetArg(al[ac],XtNheight,&data->height); ac++;
  XtGetValues(w,al,ac);
  /*
   * create a graphics context
   */
  data->gc = XCreateGC(XtDisplay(w),DefaultRootWindow(XtDisplay(w)),NULL,NULL);
  /*
   * initialize pixmap to 500 x 500 pixels
   */
  data->width = AREA_WIDTH;
  data->height = PIXEL_AREA_HEIGHT;
  if (data == &lefttextdata || data == &righttextdata) data->height = TEXT_AREA_HEIGHT;
  /*
   * define origins, amount of iterations and zoomfactor
   */
  data->xorigin = data->xstart = data->width/2;
  data->yorigin = data->ystart = data->height/2;
  data->zoomfactor = 10;
  data->xoffset = 0;
  data->yoffset = 0;
  /*
   * create drawingarea pixmap and set its foreground to white
   */
  data->pix = XCreatePixmap(XtDisplay(w),DefaultRootWindow(XtDisplay(w)),
                            data->width,data->height,
                            DefaultDepthOfScreen(XtScreen(w)));
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,data->width,data->height);
  /*
   * create bufferpixmaps for similitude and collage manipulations
   * (this has only to bo done for pixelarea's)
   */
  if (data == &leftpixeldata || data == &rightpixeldata)
    {
      data->sim = XCreatePixmap(XtDisplay(w),DefaultRootWindow(XtDisplay(w)),data->width,
                                data->height,DefaultDepthOfScreen(XtScreen(w)));
      XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
      XFillRectangle(XtDisplay(w),data->sim,data->gc,0,0,data->width,data->height);
      data->col = XCreatePixmap(XtDisplay(w),DefaultRootWindow(XtDisplay(w)),data->width,
                                data->height,DefaultDepthOfScreen(XtScreen(w)));
      XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
      XFillRectangle(XtDisplay(w),data->col,data->gc,0,0,data->width,data->height);
    }
  /*
   * initialize transformation data
   */
  data->transformation.xscaling = 80;
  data->transformation.yscaling = 80;
  data->transformation.xrotation = 0;
  data->transformation.yrotation = 0;
  data->transformation.xshift = 0;
  data->transformation.yshift = 0;
  data->transformation.probability = 100;
  /*
   * set controlflags
   */
  data->drawgrab = FALSE;
  data->erasegrab = FALSE;
  data->firsttime = TRUE;
  data->axes = FALSE;
  data->unzoom = TRUE;
  data->condensation = TRUE;
  data->identity = IDENTITY;
  /*
   * initially no mode selected
   */
  data->mode = '0';
  /*
   * set help/warning popup box guards to TRUE
   */
  data->first.filehelp = TRUE;
  data->first.originwarning = TRUE;
  data->first.fileopenwarning = TRUE;
  data->first.tablemessage = TRUE;
}
```

```
/***********************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : handle.c
 * Purpose         : Handles text to be written in text window
 * Part of         : IFS-CODEC
 * Created on      : June 01, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by     : Emile van Duren
 * Remarks         : Zoomhandler has to be altered
 ***********************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void ClearPixelArea();              /* contained in draw.c      */
void DrawText();                    /* contained in writer.c    */
void FileOpenWarning();             /* contained in messages.c  */
void OriginWarning();               /* contained in messages.c  */
Drawdata *GetWindowData();          /* contained in control.c   */
Scrolldata *GetScrollData();        /* contained in control.c   */
Textdata *GetTextData();            /* contained in control.c   */
Widget GetWindow();                 /* contained in control.c   */


void ClearTextHandler(w,location,lines)
     Widget w;
     char location;
     short int lines;
{
  Textdata *text;
  text = GetTextData(location);
  /*
   * from textlines 1 till 10 only those lines will be cleared whose according
   * position in the binary number "lines" contains a "1"
   * example: if lines = 53 (decimal) then lines = 110101 (binary) so clear
   * out lines 1,3,5,6
   */
  if (lines %2 != 0 && lines != 0)
    {
     lines--;
     text->mode = "mode        : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
     lines--;
     text->srce = "sourcefile  : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
     lines--;
     text->size = "filesize    : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
     lines--;
     text->cont = "contains    : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
     lines--;
     text->save = "savefile(s) : ";
  }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
      lines--;
      text->orgn = "origin      : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
      lines--;
      text->init = "startpixel  : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
      lines--;
      text->amit = "#iterations : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
      lines--;
      text->iter = "iteration   : ";
    }
  lines /= 2;
  if (lines %2 != 0 && lines != 0)
    {
      lines--;
      text->zoom = "zoomfactor  : ";
    }
}


void ModeHandler(w,location)
     Widget  w;
     char location;
{
  Textdata *text;
  Drawdata *data;
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  if (data->mode == 'v') text->mode = "mode        : Viewedit source file";
  if (data->mode == 'w') text->mode = "mode        : Viewedit IFS-code";
  if (data->mode == 'i') text->mode = "mode        : Interactive IFS encoding";
  if (data->mode == 'a') text->mode = "mode        : Automatic IFS encoding";
  if (data->mode == 'd') text->mode = "mode        : Deterministic IFS decoding";
  if (data->mode == 'n') text->mode = "mode        : Nondeterministic IFS decoding";
  if (data->mode == 'p') text->mode = "mode        : Performance analysis";
  if (data->mode == 'b') text->mode = "mode        : Bifurcation diagram of logistic map";
  if (data->mode == 'm') text->mode = "mode        : Mandelbrot set";
  DrawText(location);
}


void FileSizeHandler(location,ext)
     char location,*ext;
{
  char *savefile;
  static char string1[60],string2[8],string3[] = "filesize    : ",string4[] = " [bytes]";
  struct stat nfile;
  Drawdata *data;
  FILE *file;
  Textdata *text;
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  strcpy(string1,string3);
  savefile = calloc(strlen(data->filename)+5,sizeof(char));
  strcpy(savefile,data->filename);
  file = strcat(savefile,ext);
  if (stat(file,&nfile) == 0)
    {
      /*
       * print filesize in bytes
       */
      sprintf(string2,"%d",nfile.st_size);
      strcat(string1,string2);
      text->size = strcat(string1,string4);
    }
  free(savefile);
  savefile = NULL;
}


void FileContainmentHandler(location,ext)
     char location,*ext;
{
  static char string1[60],string2[4],string3[] = "contains    : ";
  static char string4[50],string5[50];
  static char string6[] = "collage-generated IFS code table";
  static char string7[] = "automatic generated IFS code table";
  static char string8[] = "manually entered IFS code table";
  Textdata *text;
  Drawdata *data;
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  sprintf(string4,"%d [pixels] of source image",data->npixels);
  sprintf(string5,"%d [pixels] of collage image",data->npixels);
  strcpy(string2,ext);
  strcpy(string1,string3);
  if (string2[2] == 'm') text->cont = strcat(string1,string4);
  if (string2[2] == 'o') text->cont = strcat(string1,string5);
  if (string2[2] == 'a') text->cont = strcat(string1,string6);
  if (string2[2] == 'u') text->cont = strcat(string1,string7);
  if (string2[2] == 'f') text->cont = strcat(string1,string8);
}


void SaveFileHandler(w,location,ext)
     Widget w;
     char location,*ext;
{
  static char string1[60],string2[20],string3[] = "savefile(s) : ";
  Textdata *text;
  Drawdata *data;
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  strcpy(string1,string3);
  sprintf(string2,"%s",data->filename);
  if (data->mode == 'v' || data->mode == 'd' || data->mode == 'n') ext = ".img";
  if (data->mode == 'i') ext = " (.man & .col)";
  if (data->mode == 'a') ext = ".aut";
  strcat(string2,ext);
  text->save = strcat(string1,string2);
  DrawText(location);
}

void SourceFileHandler(w,location,ext)
     Widget w;
     char location,*ext;
{
  static char string1[60],string2[20],string3[] = "sourcefile  : ";
  Textdata *text;
  Drawdata *data;
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  strcpy(string1,string3);
  sprintf(string2,"%s",data->filename);
  if (data->mode == 'v' || data->mode == 'a') ext = ".img";
  strcat(string2,ext);
  text->srce = strcat(string1,string2);
  /*
   * filesize and filecontainment are only connected to the sourcefile
   * so call the relevant functions from here (and from nowhere else)
   */
  FileSizeHandler(location,ext);
  FileContainmentHandler(location,ext);
  /*
   * if sourcefile exists than also savefile exists
   */
  SaveFileHandler(w,location,ext);
}
```

```
void OriginHandler(w,code,call_data)
     Widget w;
     int code;
     XmScaleCallbackStruct *call_data;
{
  char location;
  Drawdata *data;
  Textdata *text;
  static char string1[60],string2[9],string3[] = "origin      : ";
  if (code % 2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  if (data->firsttime)
    {
      /*
       * origin will be stored in screen coordinates, i.e. (0,0) is upper
       * left corner
       */
      if (code == 0) data->xorigin = call_data->value + data->width/2;
      if (code == 2) data->yorigin = data->height/2 - call_data->value;
      strcpy(string1,string3);
      sprintf(string2,"(%d,%d)",data->xorigin - data->width/2,
             data->height/2 - data->yorigin);
      text->orgn = strcat(string1,string2);
      DrawText(location);
    }
  else
    {
      ac = 0;
      XtSetArg(al[ac],XmNvalue,data->xorigin - data->width/2); ac++;
      XtSetValues(data->originbox.xslider,al,ac);
      ac = 0;
      XtSetArg(al[ac],XmNvalue,data->height/2 - data->yorigin); ac++;
      XtSetValues(data->originbox.yslider,al,ac);
      OriginWarning(w,location);
    }
}


void ChangeStartPixelHandler(w,code,call_data)
     Widget w;
     int code;
     XmScaleCallbackStruct *call_data;
{
  char location;
  Drawdata *data;
  Textdata *text;
  static char string1[60],string2[9],string3[] = "startpixel  : ";
  if (code % 2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  /*
   * startpixel will be stored in screen coordinates, i.e. (0,0) is upper
   * left corner
   */
  if (code == 0) data->xstart = call_data->value + data->width/2;
  if (code == 2) data->ystart = data->height/2 - call_data->value;
  strcpy(string1,string3);
  sprintf(string2,"(%d,%d)",data->xstart - data->width/2,data->height/2 - data->ystart);
  text->init = strcat(string1,string2);
  DrawText(location);
}


void AmountOfIterationsHandler(w,location,call_data)
     Widget w;
     char location;
     XmScaleCallbackStruct *call_data;
{
  Drawdata *data;
  Textdata *text;
  static char string1[60],string2[9],string3[] = "#iterations : ";
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  data->niterations = call_data->value;
  strcpy(string1,string3);
  sprintf(string2,"%d",data->niterations);
  text->amit = strcat(string1,string2);
  DrawText(location);
}


void IterationHandler(location,iteration)
     char location;
     unsigned long iteration;
{
  static char string1[60],string2[9],string3[] = "iteration   : ";
  Textdata *text;
  text = GetTextData(location);
  strcpy(string1,string3);
  sprintf(string2,"%d",iteration+1);
  text->iter = strcat(string1,string2);
  DrawText(location);
}


void ZoomHandler(w,code,call_data)
     Widget w;
     int code;
     XmScaleCallbackStruct *call_data;
{
  char location;
  int height = 0,width = 0,xleft,yupper;
  static char string1[60],string2[9],string3[] = "zoomfactor  : ";
  Drawdata *data;
  Textdata *text;
  if (code % 2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  data->unzoom = FALSE;
  /*
   * offset will be stored in absolute pixel offset, i.e. (0,0) is center
   * of window
   */
  if (code == 0) data->xoffset = call_data->value;
  if (code == 2) data->yoffset = call_data->value;
  if (code == 4) data->zoomfactor = call_data->value;
  width = (data->width)*(data->zoomfactor)/100.0;
  height = (data->height)*(data->zoomfactor)/100.0;
  /*
   * calculate left upper zoombox corner in screen coordinates
   */
  xleft = data->xoffset + (data->width - width)/2;
  yupper = data->yoffset + (data->height - height)/2;
  if (xleft < 0)
```

```
      {
        xleft = 0;
        data->xoffset = (int) (width - data->width)/2;
        ac = 0;
        XtSetArg(al[ac],XmNvalue,data->xoffset); ac++;
        XtSetValues(data->zoombox.xoffsetslider,al,ac);
      }
    if (xleft > (data->width - width))
      {
        xleft = data->width - width;
        data->xoffset = (int) (data->width - width)/2;
        ac = 0;
        XtSetArg(al[ac],XmNvalue,data->xoffset); ac++;
        XtSetValues(data->zoombox.xoffsetslider,al,ac);
      }
    if (yupper < 0)
      {
        yupper = 0;
        data->yoffset = (int) (height - data->height)/2;
        ac = 0;
        XtSetArg(al[ac],XmNvalue,data->yoffset); ac++;
        XtSetValues(data->zoombox.yoffsetslider,al,ac);
      }
    if (yupper > (data->height - height))
      {
        yupper = data->height - height;
        data->yoffset = (int) (height - data->height)/2;
        ac = 0;
        XtSetArg(al[ac],XmNvalue,-(data->yoffset)); ac++;
        XtSetValues(data->zoombox.yoffsetslider,al,ac);
      }
    /*
     * convert yupper from Carthesian coordinates to screen coordinates
     */
    yupper = data->height - yupper - width;
    /*
     * copy data->pix to screen
     */
    XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
            data->width,data->height,0,0);
    /*
     * set foreground top black and draw an dashed rectangle (both the white
     * dashes and the black dashes are drawn)
     */
    XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
    XSetLineAttributes(XtDisplay(w),data->gc,1,LineDoubleDash,CapNotLast,JoinMiter);
    XDrawRectangle(XtDisplay(w),XtWindow(w),data->gc,xleft,yupper,width,height);
    XSetLineAttributes(XtDisplay(w),data->gc,1,LineSolid,CapNotLast,JoinMiter);
    /*
     * only redraw text if zoomfactor is changed
     */
    if (code == 4)
      {
        strcpy(string1,string3);
        sprintf(string2,"%d",data->zoomfactor);
        text->zoom = strcat(string1,string2);
        DrawText(location);
      }
}


void ScrollBarMovedHandler(w,location,call_data)
     Widget w;
     char location;
     XmScrollBarCallbackStruct *call_data;
{
  int index,yloc = 0;
  Drawdata *data;
  Scrolldata *scroll;
  /*
   * initialization
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  scroll = GetScrollData(location);
  scroll->top = call_data->value;
  index = scroll->top;
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,data->width,data->height);
  XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,data->width,data->height);
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  while (index < scroll->nlines)
    {
      yloc+= 15;
      XDrawImageString(XtDisplay(w),data->pix,data->gc,20,yloc,
                 scroll->chars[index],scroll->length[index]);
      index++;
    }
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,data->width,data->height,0,0);
}


void ScrollHandler(w,location)
     Widget w;
     char location;
{
  char *filename,*savefile,line[MAXLINESIZE];
  float a,b,c,d,e,f,p;
  int ascent,desc,dir,i = 0;
  Drawdata *data;
  FILE *file;
  Scrolldata *scroll;
  Textdata *text;
  Widget parent;
  XCharStruct char_info;
  /*
   * initialization
   */
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  scroll = GetScrollData(location);
  ClearPixelArea(w,location);
  savefile = calloc(strlen(data->filename)+5,sizeof(char));
  strcpy(savefile,data->filename);
  if ((file = fopen(strcat(savefile,scroll->ext),"r")) == NULL)
    {
      FileOpenWarning(w,location);
    }
  else
    {
      /*
       * clear out all text in textwindow
       */
      ClearTextHandler(w,location,287);
      text->mode = "mode       : Viewedit IFS-code (textual view)";
      SourceFileHandler(w,location,scroll->ext);
      /*
       * put header in buffer "line"
       */
      for (i=0 ; i<3; i++)
      {
        if (i == 0 || i == 2) sprintf(line,"%s","     \n");
        if (i == 1) sprintf(line,"%s","    a      b      c      d      e      f      p\n");
        scroll->chars[i] = XtMalloc(strlen(line) + 1);
        line[strlen(line)-1] = '\0';
        strcpy(scroll->chars[i],line);
        scroll->length[i] = strlen(scroll->chars[i]);
        XTextExtents(scroll->font,scroll->chars[i],scroll->length[i],&dir,
                   &ascent,&desc,&char_info);
        scroll->rbearing[i] = char_info.rbearing;
        scroll->descent = desc;
        scroll->fontheight = ascent + desc;
      }
      /*
       * calculate amount of transformations
       */
      while (!feof(file))
      {
        /*
        * read each line of the file into the buffer "line", calculating
        * and caching the extents fo each line
        */
       fscanf(file,"%f %f %f %f %f %f %f\n",&a,&b,&c,&d,&e,&f,&p);
       sprintf(line,"%7.3f %7.3f %7.3f %7.3f %8.3f %8.3f %7.3f\n",a,b,c,d,e,f,p);
       scroll->chars[i] = XtMalloc(strlen(line) + 1);
       line[strlen(line)-1] = '\0';
       strcpy(scroll->chars[i],line);
       scroll->length[i] = strlen(scroll->chars[i]);
       XTextExtents(scroll->font,scroll->chars[i],scroll->length[i],&dir,
                 &ascent,&desc,&char_info);
       scroll->rbearing[i] = char_info.rbearing;
       scroll->descent = desc;
       scroll->fontheight = ascent + desc;
       i++;
    }
    /*
     * close file, initialize amount of lines and current line number
     */
    fclose(file);
    scroll->nlines = i;
    scroll->top = 0;
    /*
     * only manage scrollslider if text doesn't fit in actual window
     */
    if (15*scroll->nlines > data->height)
    {
       Manage(w,data->scrollbox.box);
       ac = 0;
       XtSetArg(al[ac],XmNmaximum,scroll->nlines); ac++;
       XtSetArg(al[ac],XmNsliderSize,data->height/15); ac++;
       XtSetArg(al[ac],XmNpageIncrement,data->height/15); ac++;
       XtSetValues(data->scrollbox.scrollbar,al,ac);
    }
    /*
     * call ScrollBarMovedHandler to display text for the first time
     */
    ScrollBarMovedHandler(w,location,NULL);
  }
  free(savefile);
  savefile = NULL;
}
```

```
/*****************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : ifs.c
 * Purpose         : Main body for IFS-CODEC
 * Part of         : IFS-CODEC
 * Created on      : May 08, 1992
 * Created by      : Emile van Duren
 * Last modified on: November 20, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 *****************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void DrawAxes();               /* contained in draw.c      */
void Redisplay();              /* contained in exposures.c */
Widget CreateUpperMenu();      /* contained in menubars.c  */
Widget CreateLeftRightMenu();  /* contained in menubars.c  */


/*
 * main programbody
 */
void main(argc,argv)
     int  argc;
     char *argv[];
{
  Widget toplevel,uppermenu;
  Widget leftmenu,rightmenu;
  XmString xmstr;
  /*
   * initialize textdata
   */
  LText.mode = RText.mode = "mode          : ";
  LText.srce = RText.srce = "sourcefile    : ";
  LText.size = RText.size = "filesize      : ";
  LText.cont = RText.cont = "contains      : ";
  LText.save = RText.save = "savefile(s)   : ";
  LText.orgn = RText.orgn = "origin        : ";
  LText.init = RText.init = "startpixel    : ";
  LText.amit = RText.amit = "#iterations   : ";
  LText.iter = RText.iter = "iteration     : ";
  LText.zoom = RText.zoom = "zoomfactor    : ";
  /*
   * initialize main guards
   */
  firsthelp = TRUE;
  printmessage = TRUE;
  copyrightmessage = TRUE;
  /*
   * initialize toolkit and create toplevelshell
   */
  ac = 0;
  toplevel = XtInitialize(argv[0],"TopLevel",NULL,0,al,ac);
  /*
   * create form with size of 1030 x 830 pixels
   */
  ac = 0;
  XtSetArg(al[ac],XmNwidth,1030); ac++;
  XtSetArg(al[ac],XmNheight,830); ac++;
  xmstr = XmStringCreate("IFS-CODEC",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  XtSetArg(al[ac],XmNhorizontalSpacing,10); ac++;
  XtSetArg(al[ac],XmNverticalSpacing,10); ac++;
  XtSetArg(al[ac],XmNfractionBase,1000); ac++;
  form = XmCreateForm(toplevel,"Form",al,ac);
  XtManageChild(form);
```

```
XmStringFree(xmstr);
/*
 * create upper pulldownmenu
 */
uppermenu = CreateUpperMenu(form);
ac = 0;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetValues(uppermenu,al,ac);
/*
 * create left pulldownmenu
 */
ac = 0;
leftmenu = CreateLeftRightMenu(form,'l');
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,uppermenu); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_POSITION); ac++;
XtSetArg(al[ac],XmNrightPosition,495); ac++;
XtSetValues(leftmenu,al,ac);
/*
 * create right pulldownmenu
 */
ac = 0;
rightmenu = CreateLeftRightMenu(form,'r');
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,uppermenu); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_POSITION); ac++;
XtSetArg(al[ac],XmNleftPosition,505); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetValues(rightmenu,al,ac);
/*
 * create lefttextframe for lefttextarea (size = 200 x 500 pixels)
 */
ac = 0;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,leftmenu); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_POSITION); ac++;
XtSetArg(al[ac],XmNrightPosition,495); ac++;
XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_POSITION); ac++;
XtSetArg(al[ac],XmNbottomPosition,374); ac++;
XtSetArg(al[ac],XmNshadowType,XmSHADOW_IN); ac++;
lefttextframe = XmCreateFrame(form,"LeftTextFrame",al,ac);
XtManageChild(lefttextframe);
/*
 * create left textdrawingarea
 */
ac = 0;
lefttextarea = XmCreateDrawingArea(lefttextframe,"LeftTextArea",al,ac);
InitGraphics(lefttextarea,&lefttextdata);
XtAddCallback(lefttextarea,XmNexposeCallback,Redisplay,&lefttextdata);
XtManageChild(lefttextarea);
/*
 * create righttextframe for righttextarea (size = 200 x 500 pixels)
 */
ac = 0;
XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
XtSetArg(al[ac],XmNtopWidget,rightmenu); ac++;
XtSetArg(al[ac],XmNleftAttachment,XmATTACH_POSITION); ac++;
XtSetArg(al[ac],XmNleftPosition,505); ac++;
XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_POSITION); ac++;
XtSetArg(al[ac],XmNbottomPosition,374); ac++;
XtSetArg(al[ac],XmNshadowType,XmSHADOW_IN); ac++;
righttextframe = XmCreateFrame(form,"RightTextFrame",al,ac);
XtManageChild(righttextframe);
/*
 * create right textdrawingarea
 */
ac = 0;
righttextarea = XmCreateDrawingArea(righttextframe,"RightTextArea",al,ac);
InitGraphics(righttextarea,&righttextdata);
XtAddCallback(righttextarea,XmNexposeCallback,Redisplay,&righttextdata);
```

```
    XtManageChild(righttextarea);
    /*
     * create leftpixelframe for leftpixelarea (size = 500 x 500 pixels)
     */
    ac = 0;
    XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
    XtSetArg(al[ac],XmNtopWidget,lefttextframe); ac++;
    XtSetArg(al[ac],XmNleftAttachment,XmATTACH_FORM); ac++;
    XtSetArg(al[ac],XmNrightAttachment,XmATTACH_POSITION); ac++;
    XtSetArg(al[ac],XmNrightPosition,495); ac++;
    XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
    XtSetArg(al[ac],XmNshadowType,XmSHADOW_IN); ac++;
    leftpixelframe = XmCreateFrame(form,"LeftPixelFrame",al,ac);
    XtManageChild(leftpixelframe);
    /*
     * create left pixeldrawingarea
     */
    ac = 0;
    leftpixelarea = XmCreateDrawingArea(leftpixelframe,"LeftPixelArea",al,ac);
    InitGraphics(leftpixelarea,&leftpixeldata);
    InitMouseTracker(leftpixelarea);
    XtAddCallback(leftpixelarea,XmNexposeCallback,Redisplay,&leftpixeldata);
    XtAddCallback(leftpixelarea,XmNexposeCallback,DrawAxes,'l');
    XtManageChild(leftpixelarea);
    /*
     * create rightpixelframe for rightpixelarea (size = 500 x 500 pixels)
     */
    ac = 0;
    XtSetArg(al[ac],XmNtopAttachment,XmATTACH_WIDGET); ac++;
    XtSetArg(al[ac],XmNtopWidget,righttextframe); ac++;
    XtSetArg(al[ac],XmNleftAttachment,XmATTACH_POSITION); ac++;
    XtSetArg(al[ac],XmNleftPosition,505); ac++;
    XtSetArg(al[ac],XmNrightAttachment,XmATTACH_FORM); ac++;
    XtSetArg(al[ac],XmNbottomAttachment,XmATTACH_FORM); ac++;
    XtSetArg(al[ac],XmNshadowType,XmSHADOW_IN); ac++;
    rightpixelframe = XmCreateFrame(form,"RightPixelFrame",al,ac);
    XtManageChild(rightpixelframe);
    /*
     * create right pixeldrawingarea
     */
    ac = 0;
    rightpixelarea = XmCreateDrawingArea(rightpixelframe,"RightPixelArea",al,ac);
    InitGraphics(rightpixelarea,&rightpixeldata);
    InitMouseTracker(rightpixelarea);
    XtAddCallback(rightpixelarea,XmNexposeCallback,Redisplay,&rightpixeldata);
    XtAddCallback(rightpixelarea,XmNexposeCallback,DrawAxes,'r');
    XtManageChild(rightpixelarea);
    /*
     * realize widget
     */
    XtRealizeWidget(toplevel);
    XtMainLoop();
}
```

```
/*********************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : menubars.c
 * Purpose          : Contains functions CreateUpperMenu and CreateLeftRightMenu
 * Part of          : IFS-CODEC
 * Created on       : May 12, 1992
 * Created by       : Emile van Duren
 * Last modified on: December 16, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 *********************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void AddSimilitude();                    /* contained in collage.c       */
void AutomaticIFSEncoding();             /* contained in genetic.c       */
void AxesController();                   /* contained in control.c       */
void CalculateFractalDimension();        /* contained in specials.c      */
void CalculateMoments();                 /* contained in specials.c      */
void ClearAll();                         /* contained in draw.c          */
void ClearTextController();              /* contained in control.c       */
void CondensationController();           /* contained in control.c       */
void CopyrightMessage();                 /* contained in messages.c      */
void DeterministicRendering();           /* contained in render.c        */
void DrawController();                   /* contained in control.c       */
void DrawAxes();                         /* contained in draw.c          */
void DrawBifurcationDiagram();           /* contained in specials.c      */
void DrawMandelbrotSet();                /* contained in specials.c      */
void DrawRectangle();                    /* contained in draw.c          */
void EraseController();                  /* contained in control.c       */
void FixSimilitude();                    /* contained in collage.c       */
void GraphicalView();                    /* contained in render.c        */
void IdentityController();               /* contained in control.c       */
void LoadFile();                         /* contained in filehandler.c   */
void MainHelpCallback();                 /* contained in messages.c      */
void Manage();                           /* contained in control.c       */
void ModeController();                   /* contained in control.c       */
void ModeHandler();                      /* contained in handle.c        */
void NondeterministicRendering();        /* contained in render.c        */
void PixsetMessage();                    /* contained in messages.c      */
void PrintMessage();                     /* contained in messages.c      */
void Quit();                             /* contained in control.c       */
void SaveCollage();                      /* contained in collage.c       */
void SaveFile();                         /* contained in filehandler.c   */
void SaveFileHandler();                  /* contained in handle.c        */
void ScrollController();                 /* contained in control.c       */
void ScrollHandler();                    /* contained in handle.c        */
void SourceFileHandler();                /* contained in handle.c        */
void Unmanage();                         /* contained in control.c       */
void UnzoomController();                 /* contained in control.c       */
void VieweditIFSTableMessage();          /* contained in messages.c      */
Widget CreateIterationBox();             /* contained in boxes.c         */
Widget CreateOriginBox();                /* contained in boxes.c         */
Widget CreateScrollBox();                /* contained in boxes.c         */
Widget CreateSimilitudeTransformBox();   /* contained in boxes.c         */
Widget CreateStartPixelBox();            /* contained in boxes.c         */
Widget CreatePromptBox();                /* contained in boxes.c         */
Widget CreateZoomBox();                  /* contained in boxes.c         */


Widget CreateUpperMenu(parent)
     Widget parent;
{
  Widget uppermenu,filepulldown,helpbutton;
  Widget quitprogram,printbutton,pixsetbutton,copyrightbutton;
  XmString xmstr;
  /*
```

```
   * create uppermenubar
   */
  uppermenu = XmCreateMenuBar(parent,"UpperMenu",NULL,0);
  XtManageChild(uppermenu);
  /*
   * create quit program button in file pulldown pane
   */
  ac = 0;
  xmstr = XmStringCreate("Quit",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
  XtSetArg(al[ac],XmNmnemonic,'Q'); ac++;
  quitprogram = XmCreateCascadeButton(uppermenu,"QuitProgram",al,ac);
  XtAddCallback(quitprogram,XmNactivateCallback,Quit,NULL);
  XtManageChild(quitprogram);
  XmStringFree(xmstr);
  /*
   * create helpbutton in uppermenu
   */
  ac = 0;
  xmstr = XmStringCreate("Help",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
  XtSetArg(al[ac],XmNmnemonic,'H'); ac++;
  helpbutton = XmCreateCascadeButton(uppermenu,"HelpButton",al,ac);
  XtAddCallback(helpbutton,XmNactivateCallback,MainHelpCallback,NULL);
  XtManageChild(helpbutton);
  XmStringFree(xmstr);
  /*
   * create printbutton in uppermenu
   */
  ac = 0;
  xmstr = XmStringCreate("Print",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
  XtSetArg(al[ac],XmNmnemonic,'P'); ac++;
  printbutton = XmCreateCascadeButton(uppermenu,"PrintButton",al,ac);
  XtAddCallback(printbutton,XmNactivateCallback,PrintMessage,NULL);
  XtManageChild(printbutton);
  XmStringFree(xmstr);
  /*
   * create pixsetbutton in uppermenu
   */
  ac = 0;
  xmstr = XmStringCreate("Pixset distance",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
  XtSetArg(al[ac],XmNmnemonic,'d'); ac++;
  pixsetbutton = XmCreateCascadeButton(uppermenu,"PixsetButton",al,ac);
  XtAddCallback(pixsetbutton,XmNactivateCallback,PixsetMessage,NULL);
  XtManageChild(pixsetbutton);
  XmStringFree(xmstr);
  /*
   * create copyrightbutton in uppermenu
   */
  ac = 0;
  xmstr = XmStringCreate("Copyright(c)",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
  XtSetArg(al[ac],XmNmnemonic,'C'); ac++;
  copyrightbutton = XmCreateCascadeButton(uppermenu,"CopyrightButton",al,ac);
  XtAddCallback(copyrightbutton,XmNactivateCallback,CopyrightMessage,NULL);
  XtManageChild(copyrightbutton);
  XmStringFree(xmstr);
  /*
   * return value of this function
   */
  return(uppermenu);
}


Widget CreateLeftRightMenu(parent,location)
    Widget parent;
    char   location;
{
  Widget abutton,apulldown,asbutton,bdbutton,cabutton,clearbutton,ctbutton;
  Widget dabutton,dbutton,dibutton,displaybutton,displaypulldown,dmbutton;
  Widget dpulldown,drawaxes,drbutton,erbutton,fdbutton,fsbutton,gabutton;
  Widget gibutton,gmbutton,gvbutton,ibutton,idbutton,ipulldown,itbutton;
  Widget iterationbox,leftrightmenu,lobutton,mobutton,msbutton,nabutton;
Widget nbutton,nibutton,nmbutton,npulldown,optionsbutton,optionspulldown;
Widget orbutton,promptbox,rendercontrolbutton,rendercontrolpulldown,rtbutton;
Widget scbutton,sebutton,selectfile,spbutton,specialsbutton,specialspulldown;
Widget tabutton,tibutton,tmbutton,transformbox,trbutton,tsbutton,tvbutton;
Widget uzbutton,vbutton,vpulldown,vsbutton,wbutton,webutton,wgpulldown;
Widget wpulldown,wtpulldown,zmbutton;
XmString xmstr;
int mode;
Drawdata *data;
data = GetWindowData(location,'p');
if (location == 'l') mode = 0;
if (location == 'r') mode = 1;
/*
 * create leftrightmenubar
 */
leftrightmenu = XmCreateMenuBar(parent,"LeftRightMenu",NULL,0);
XtManageChild(leftrightmenu);
/*
 * create select file button in file pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Select file",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'f'); ac++;
selectfile = XmCreateCascadeButton(leftrightmenu,"SelectFile",al,ac);
promptbox = CreatePromptBox(selectfile,location);
XtAddCallback(selectfile,XmNactivateCallback,Manage,promptbox);
XtManageChild(selectfile);
XmStringFree(xmstr);
/*
 * create displaypulldown menupane as submenu of leftrightmenu
 */
ac = 0;
displaypulldown = XmCreatePulldownMenu(leftrightmenu,"DisplayPulldown",al,ac);
/*
 * create displaybutton for display pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Display",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'D'); ac++;
XtSetArg(al[ac],XmNsubMenuId,displaypulldown); ac++;
displaybutton = XmCreateCascadeButton(leftrightmenu,"DisplayButton",al,ac);
XtManageChild(displaybutton);
XmStringFree(xmstr);
/*
 * create clearbutton in display pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Clear",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'C'); ac++;
clearbutton = XmCreatePushButtonGadget(displaypulldown,"Clear",al,ac);
XtAddCallback(clearbutton,XmNactivateCallback,ClearAll,location);
XtManageChild(clearbutton);
XmStringFree(xmstr);
/*
 * create drawaxesbutton in display pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Axes (toggle)",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'A'); ac++;
drawaxes = XmCreatePushButtonGadget(displaypulldown,"DrawAxes",al,ac);
XtAddCallback(drawaxes,XmNactivateCallback,AxesController,location);
XtAddCallback(drawaxes,XmNactivateCallback,DrawAxes,location);
XtManageChild(drawaxes);
XmStringFree(xmstr);
/*
 * create redefine origin button in display pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Redefine origin",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'R'); ac++;
```

```
orbutton = XmCreateCascadeButton(displaypulldown,"OrButton",al,ac);
data->originbox.box = CreateOriginBox(orbutton,location);
XtAddCallback(orbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(orbutton,XmNactivateCallback,Manage,data->originbox.box);
XtManageChild(orbutton);
XmStringFree(xmstr);
/*
 * create optionspulldown menupane as submenu of leftrightmenu
 */
ac = 0;
optionspulldown =XmCreatePulldownMenu(leftrightmenu,"OptionsPulldown",al,ac);
/*
 * create optionsbutton for optionspulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Options",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'O'); ac++;
XtSetArg(al[ac],XmNsubMenuId,optionspulldown); ac++;
optionsbutton = XmCreateCascadeButton(leftrightmenu,"OptionsButton",al,ac);
XtManageChild(optionsbutton);
XmStringFree(xmstr);
/*
 * create viewedit ifs source image pulldown menupane (vpulldown pane)
 */
vpulldown = XmCreatePulldownMenu(optionspulldown,"VPulldown",NULL,0);
/*
 * create viewedit ifs source image button for vpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Viewedit source image",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'V'); ac++;
XtSetArg(al[ac],XmNsubMenuId,vpulldown); ac++;
vbutton = XmCreateCascadeButton(optionspulldown,"VButton",al,ac);
XtManageChild(vbutton);
XmStringFree(xmstr);
/*
 * create load file button in vpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Load file",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'L'); ac++;
lobutton = XmCreatePushButtonGadget(vpulldown,"LoButton",al,ac);
XtAddCallback(lobutton,XmNactivateCallback,ModeController,mode);
XtAddCallback(lobutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(lobutton,XmNactivateCallback,LoadFile,location);
XtAddCallback(lobutton,XmNactivateCallback,SourceFileHandler,location);
XtManageChild(lobutton);
XmStringFree(xmstr);
/*
 * create draw toggle button in vpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Draw (toggle)",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'D'); ac++;
drbutton = XmCreatePushButtonGadget(vpulldown,"DrButton",al,ac);
XtAddCallback(drbutton,XmNactivateCallback,ModeController,mode);
XtAddCallback(drbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(drbutton,XmNactivateCallback,SaveFileHandler,location);
XtAddCallback(drbutton,XmNactivateCallback,DrawController,location);
XtManageChild(drbutton);
XmStringFree(xmstr);
/*
 * create erase (toggle) button in vpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Erase (toggle)",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'E'); ac++;
erbutton = XmCreatePushButtonGadget(vpulldown,"ErButton",al,ac);
XtAddCallback(erbutton,XmNactivateCallback,ModeController,mode);
XtAddCallback(erbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(erbutton,XmNactivateCallback,SaveFileHandler,location);
XtAddCallback(erbutton,XmNactivateCallback,EraseController,location);
XtManageChild(erbutton);
XmStringFree(xmstr);
/*
 * create save file button in vpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Save file",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'S'); ac++;
vsbutton = XmCreatePushButtonGadget(vpulldown,"VsButton",al,ac);
XtAddCallback(vsbutton,XmNactivateCallback,ModeController,mode);
XtAddCallback(vsbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(vsbutton,XmNactivateCallback,SaveFileHandler,location);
XtAddCallback(vsbutton,XmNactivateCallback,SaveFile,location);
XtManageChild(vsbutton);
XmStringFree(xmstr);
/*
 * create viewedit ifs code pulldown menupane (wpulldown pane)
 */
wpulldown = XmCreatePulldownMenu(optionspulldown,"WPulldown",NULL,0);
/*
 * create viewedit ifs code button for wpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Viewedit IFS-code",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'w'); ac++;
XtSetArg(al[ac],XmNsubMenuId,wpulldown); ac++;
wbutton = XmCreateCascadeButton(optionspulldown,"WButton",al,ac);
XtManageChild(wbutton);
XmStringFree(xmstr);
/*
 * create graphical view pulldown menupane (wgpulldown pane)
 */
wgpulldown = XmCreatePulldownMenu(wpulldown,"WgPulldown",NULL,0);
/*
 * create graphical view button for wgpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Graphical view",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'G'); ac++;
XtSetArg(al[ac],XmNsubMenuId,wgpulldown); ac++;
gvbutton = XmCreateCascadeButton(wpulldown,"GvButton",al,ac);
XtManageChild(gvbutton);
XmStringFree(xmstr);
/*
 * create .ifs button in wgpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".ifs",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'i'); ac++;
gibutton = XmCreatePushButtonGadget(wgpulldown,"GiButton",al,ac);
XtAddCallback(gibutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(gibutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(gibutton,XmNactivateCallback,GraphicalView,mode);
XtManageChild(gibutton);
XmStringFree(xmstr);
/*
 * create .man button in wgpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".man",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'m'); ac++;
gmbutton = XmCreatePushButtonGadget(wgpulldown,"GmButton",al,ac);
XtAddCallback(gmbutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(gmbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(gmbutton,XmNactivateCallback,GraphicalView,mode+2);
XtManageChild(gmbutton);
XmStringFree(xmstr);
/*
```

```
 * create .aut button in wgpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".aut",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'a'); ac++;
gabutton = XmCreatePushButtonGadget(wgpulldown,"GaButton",al,ac);
XtAddCallback(gabutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(gabutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(gabutton,XmNactivateCallback,GraphicalView,mode+4);
XtManageChild(gabutton);
XmStringFree(xmstr);
/*
 * create textual view pulldown menupane (wtpulldown pane)
 */
wtpulldown = XmCreatePulldownMenu(wpulldown,"WtPulldown",NULL,0);
/*
 * create textual view button for wtpulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Textual view",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'T'); ac++;
XtSetArg(al[ac],XmNsubMenuId,wtpulldown); ac++;
tvbutton = XmCreateCascadeButton(wpulldown,"GvButton",al,ac);
XtManageChild(tvbutton);
XmStringFree(xmstr);
/*
 * create .ifs button in wtpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".ifs",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'i'); ac++;
tibutton = XmCreateCascadeButton(wtpulldown,"TiButton",al,ac);
data->scrollbox.box = CreateScrollBox(tibutton,location);
XtAddCallback(tibutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(tibutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(tibutton,XmNactivateCallback,ScrollController,mode);
XtAddCallback(tibutton,XmNactivateCallback,ScrollHandler,location);
XtManageChild(tibutton);
XmStringFree(xmstr);
/*
 * create .man button in wtpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".man",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'m'); ac++;
tmbutton = XmCreateCascadeButton(wtpulldown,"TmButton",al,ac);
data->scrollbox.box = CreateScrollBox(tmbutton,location);
XtAddCallback(tmbutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(tmbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(tmbutton,XmNactivateCallback,ScrollController,mode+2);
XtAddCallback(tmbutton,XmNactivateCallback,ScrollHandler,location);
XtManageChild(tmbutton);
XmStringFree(xmstr);
/*
 * create .aut button in wtpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".aut",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'a'); ac++;
tabutton = XmCreateCascadeButton(wtpulldown,"TaButton",al,ac);
data->scrollbox.box = CreateScrollBox(tabutton,location);
XtAddCallback(tabutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(tabutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(tabutton,XmNactivateCallback,ScrollController,mode+4);
XtAddCallback(tabutton,XmNactivateCallback,ScrollHandler,location);
XtManageChild(tabutton);
XmStringFree(xmstr);
/*
 * create textual edit button in viewedit IFS-code pulldown pane
 */
```

```
ac = 0;
xmstr = XmStringCreate("Textual edit",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'e'); ac++;
webutton = XmCreateCascadeButton(wpulldown,"WeButton",al,ac);
XtAddCallback(webutton,XmNactivateCallback,ModeController,mode+2);
XtAddCallback(webutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(webutton,XmNactivateCallback,ClearTextController,location);
XtAddCallback(webutton,XmNactivateCallback,VieweditIFSTableMessage,location);
XtManageChild(webutton);
XmStringFree(xmstr);
/*
 * create interactive IFS-encoding pulldown menupane (ipulldown pane)
 */
ipulldown = XmCreatePulldownMenu(optionspulldown,"IPulldown",NULL,0);
/*
 * create interactive IFS-encoding button for ipulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Interactive IFS-encoding",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'I'); ac++;
XtSetArg(al[ac],XmNsubMenuId,ipulldown); ac++;
ibutton = XmCreateCascadeButton(optionspulldown,"IButton",al,ac);
XtManageChild(ibutton);
XmStringFree(xmstr);
/*
 * create add similitude button in ipulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Add similitude",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'A'); ac++;
asbutton = XmCreatePushButtonGadget(ipulldown,"AsButton",al,ac);
XtAddCallback(asbutton,XmNactivateCallback,ModeController,mode+4);
XtAddCallback(asbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(asbutton,XmNactivateCallback,ClearTextController,location);
XtAddCallback(asbutton,XmNactivateCallback,SaveFileHandler,location);
XtAddCallback(asbutton,XmNactivateCallback,AddSimilitude,location);
XtManageChild(asbutton);
XmStringFree(xmstr);
/*
 * create transform similitude button in ipulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Transform similitude",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'T'); ac++;
tsbutton = XmCreateCascadeButton(ipulldown,"TsButton",al,ac);
transformbox = CreateSimilitudeTransformBox(tsbutton,location);
XtAddCallback(tsbutton,XmNactivateCallback,ModeController,mode+4);
XtAddCallback(tsbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(tsbutton,XmNactivateCallback,ClearTextController,location);
XtAddCallback(tsbutton,XmNactivateCallback,SaveFileHandler,location);
XtAddCallback(tsbutton,XmNactivateCallback,Manage,transformbox);
XtManageChild(tsbutton);
XmStringFree(xmstr);
/*
 * create fix similitude button in ipulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Fix similitude",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'F'); ac++;
fsbutton = XmCreatePushButtonGadget(ipulldown,"FsButton",al,ac);
XtAddCallback(fsbutton,XmNactivateCallback,ModeController,mode+4);
XtAddCallback(fsbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(fsbutton,XmNactivateCallback,ClearTextController,location);
XtAddCallback(fsbutton,XmNactivateCallback,FixSimilitude,location);
XtManageChild(fsbutton);
XmStringFree(xmstr);
/*
 * create save collage button in ipulldown pane
 */
ac = 0;
```

```
xmstr = XmStringCreate("Save collage",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'S'); ac++;
scbutton = XmCreatePushButtonGadget(ipulldown,"ScButton",al,ac);
XtAddCallback(scbutton,XmNactivateCallback,ModeController,mode+4);
XtAddCallback(scbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(scbutton,XmNactivateCallback,ClearTextController,location);
XtAddCallback(scbutton,XmNactivateCallback,SaveFileHandler,location);
XtAddCallback(scbutton,XmNactivateCallback,SaveCollage,location);
XtAddCallback(scbutton,XmNactivateCallback,Unmanage,transformbox);
XtManageChild(scbutton);
XmStringFree(xmstr);
/*
 * create automatic IFS-encoding pulldown menupane (apulldown pane)
 */
apulldown = XmCreatePulldownMenu(optionspulldown,"APulldown",NULL,0);
/*
 * create automatic IFS-encoding button for apulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Automatic IFS-encoding",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'A'); ac++;
XtSetArg(al[ac],XmNsubMenuId,apulldown); ac++;
abutton = XmCreateCascadeButton(optionspulldown,"AButton",al,ac);
XtManageChild(abutton);
XmStringFree(xmstr);
/*
 * create identity button in apulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Identity (toggle)",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'I'); ac++;
idbutton = XmCreatePushButtonGadget(apulldown,"IdButton",al,ac);
XtAddCallback(idbutton,XmNactivateCallback,IdentityController,location);
XtManageChild(idbutton);
XmStringFree(xmstr);
/*
 * create collage button in apulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Calculate",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'C'); ac++;
cabutton = XmCreatePushButtonGadget(apulldown,"CaButton",al,ac);
XtAddCallback(cabutton,XmNactivateCallback,ModeController,mode+6);
XtAddCallback(cabutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(cabutton,XmNactivateCallback,ClearTextController,location);
XtAddCallback(cabutton,XmNactivateCallback,AutomaticIFSEncoding,location);
XtManageChild(cabutton);
XmStringFree(xmstr);
/*
 * create deterministic IFS-decoding pulldown menupane (dpulldown pane)
 */
dpulldown = XmCreatePulldownMenu(optionspulldown,"DPulldown",NULL,0);
/*
 * create deterministic IFS-decoding button for dpulldown pane
 */
ac = 0;
xmstr =XmStringCreate("Deterministic IFS-decoding",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'D'); ac++;
XtSetArg(al[ac],XmNsubMenuId,dpulldown); ac++;
dbutton = XmCreateCascadeButton(optionspulldown,"DButton",al,ac);
XtManageChild(dbutton);
XmStringFree(xmstr);
/*
 * create .ifs button in dpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".ifs",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'i'); ac++;
dibutton = XmCreatePushButtonGadget(dpulldown,"DiButton",al,ac);
```

```
XtAddCallback(dibutton,XmNactivateCallback,ModeController,mode+8);
XtAddCallback(dibutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(dibutton,XmNactivateCallback,DeterministicRendering,mode);
XtManageChild(dibutton);
XmStringFree(xmstr);
/*
 * create .man button in dpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".man",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'m'); ac++;
dmbutton = XmCreatePushButtonGadget(dpulldown,"DmButton",al,ac);
XtAddCallback(dmbutton,XmNactivateCallback,ModeController,mode+8);
XtAddCallback(dmbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(dmbutton,XmNactivateCallback,DeterministicRendering,mode+2);
XtManageChild(dmbutton);
XmStringFree(xmstr);
/*
 * create .aut button in dpulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".aut",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'a'); ac++;
dabutton = XmCreatePushButtonGadget(dpulldown,"DaButton",al,ac);
XtAddCallback(dabutton,XmNactivateCallback,ModeController,mode+8);
XtAddCallback(dabutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(dabutton,XmNactivateCallback,DeterministicRendering,mode+4);
XtManageChild(dabutton);
XmStringFree(xmstr);
/*
 * create nondeterministic IFS-decoding pulldown menupane (npulldown pane)
 */
npulldown = XmCreatePulldownMenu(optionspulldown,"NPulldown",NULL,0);
/*
 * create nondeterministic IFS-decoding button for npulldown pane
 */
ac = 0;
xmstr =XmStringCreate("Noneterministic IFS-decoding",
                  XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'N'); ac++;
XtSetArg(al[ac],XmNsubMenuId,npulldown); ac++;
nbutton = XmCreateCascadeButton(optionspulldown,"NButton",al,ac);
XtManageChild(nbutton);
XmStringFree(xmstr);
/*
 * create .ifs button in npulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".ifs",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'i'); ac++;
nibutton = XmCreatePushButtonGadget(npulldown,"NiButton",al,ac);
XtAddCallback(nibutton,XmNactivateCallback,ModeController,mode+10);
XtAddCallback(nibutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(nibutton,XmNactivateCallback,NondeterministicRendering,mode);
XtManageChild(nibutton);
XmStringFree(xmstr);
/*
 * create .man button in npulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".man",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'m'); ac++;
nmbutton = XmCreatePushButtonGadget(npulldown,"NmButton",al,ac);
XtAddCallback(nmbutton,XmNactivateCallback,ModeController,mode+10);
XtAddCallback(nmbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(nmbutton,XmNactivateCallback,NondeterministicRendering,mode+2);
XtManageChild(nmbutton);
XmStringFree(xmstr);
/*
 * create .aut button in npulldownpane
 */
ac = 0;
xmstr = XmStringCreate(".aut",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'a'); ac++;
nabutton = XmCreatePushButtonGadget(npulldown,"NaButton",al,ac);
XtAddCallback(nabutton,XmNactivateCallback,ModeController,mode+10);
XtAddCallback(nabutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(nabutton,XmNactivateCallback,NondeterministicRendering,mode+4);
XtManageChild(nabutton);
XmStringFree(xmstr);
/*
 * create rendercontrol pulldown menupane as submenu of leftrightmenu
 */
ac = 0;
rendercontrolpulldown = XmCreatePulldownMenu(leftrightmenu,"RenderControlPulldown",al,ac);
/*
 * create rendercontrolbutton for rendercontrol pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Render control",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'R'); ac++;
XtSetArg(al[ac],XmNsubMenuId,rendercontrolpulldown); ac++;
rendercontrolbutton = XmCreateCascadeButton(leftrightmenu,"RenderControlButton",al,ac);
XtManageChild(rendercontrolbutton);
XmStringFree(xmstr);
/*
 * create define startpixel button in rendercontrol pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Define startpixel",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'s'); ac++;
spbutton = XmCreateCascadeButton(rendercontrolpulldown,"SpButton",al,ac);
data->startpixelbox.box = CreateStartPixelBox(spbutton,location);
XtAddCallback(spbutton,XmNactivateCallback,Manage,data->startpixelbox.box);
XtManageChild(spbutton);
XmStringFree(xmstr);
/*
 * create define amount of iterations button in rendercontrol pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Define #iterations",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'i'); ac++;
itbutton = XmCreateCascadeButton(rendercontrolpulldown,"ItButton",al,ac);
data->niterations = 1;
data->iterationbox.box = CreateIterationBox(itbutton,location);
XtAddCallback(itbutton,XmNactivateCallback,Manage,data->iterationbox.box);
XtManageChild(itbutton);
XmStringFree(xmstr);
/*
 * create zoom button in rendercontrol pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Zoom area",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'Z'); ac++;
zmbutton = XmCreateCascadeButton(rendercontrolpulldown,"ZmButton",al,ac);
data->zoombox.box = CreateZoomBox(zmbutton,location);
XtAddCallback(zmbutton,XmNactivateCallback,Manage,data->zoombox.box);
XtManageChild(zmbutton);
XmStringFree(xmstr);
/*
 * create unzoom button in rendercontrol pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Unzoom area",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'U'); ac++;
uzbutton = XmCreateCascadeButton(rendercontrolpulldown,"UzButton",al,ac);
XtAddCallback(uzbutton,XmNactivateCallback,UnzoomController,location);
XtManageChild(uzbutton);
XmStringFree(xmstr);
```

```
/*
 * create condensation (toggle) button in rendercontrol pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Condensation (toggle)",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'C'); ac++;
ctbutton = XmCreateCascadeButton(rendercontrolpulldown,"CtButton",al,ac);
XtAddCallback(ctbutton,XmNactivateCallback,CondensationController,location);
XtManageChild(ctbutton);
XmStringFree(xmstr);
/*
 * create specials pulldown menupane as submenu of leftrightmenu
 */
ac = 0;
specialspulldown = XmCreatePulldownMenu(leftrightmenu,"SpecialsPulldown",al,ac);
/*
 * create specials button for specials pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Specials",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'S'); ac++;
XtSetArg(al[ac],XmNsubMenuId,specialspulldown); ac++;
specialsbutton = XmCreateCascadeButton(leftrightmenu,"SpecialsButton",al,ac);
XtManageChild(specialsbutton);
XmStringFree(xmstr);
/*
 * create rectangle button in specials pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Rectangle",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'R'); ac++;
rtbutton = XmCreateCascadeButton(specialspulldown,"RtButton",al,ac);
XtAddCallback(rtbutton,XmNactivateCallback,DrawRectangle,location);
XtManageChild(rtbutton);
XmStringFree(xmstr);
/*
 * create bifurcation diagram button in specials pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Bifurcation diagram",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'B'); ac++;
bdbutton = XmCreateCascadeButton(specialspulldown,"BdButton",al,ac);
XtAddCallback(bdbutton,XmNactivateCallback,ModeController,mode+14);
XtAddCallback(bdbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(bdbutton,XmNactivateCallback,DrawBifurcationDiagram,location);
XtManageChild(bdbutton);
XmStringFree(xmstr);
/*
 * create mandelbrot set button in specials pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Mandelbrot set",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'M'); ac++;
msbutton = XmCreateCascadeButton(specialspulldown,"MsButton",al,ac);
XtAddCallback(msbutton,XmNactivateCallback,ModeController,mode+16);
XtAddCallback(msbutton,XmNactivateCallback,ModeHandler,location);
XtAddCallback(msbutton,XmNactivateCallback,DrawMandelbrotSet,location);
XtManageChild(msbutton);
XmStringFree(xmstr);
/*
 * create moment sequence button in specials pulldown pane
 */
ac = 0;
xmstr = XmStringCreate("Moment sequence",XmSTRING_DEFAULT_CHARSET);
XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
XtSetArg(al[ac],XmNmnemonic,'s'); ac++;
mobutton = XmCreateCascadeButton(specialspulldown,"MoButton",al,ac);
XtAddCallback(mobutton,XmNactivateCallback,CalculateMoments,location);
XtManageChild(mobutton);
XmStringFree(xmstr);
```

```
  /*
   * create fractal dimension button in specials pulldown pane
   */
  ac = 0;
  xmstr = XmStringCreate("Fractal dimension",XmSTRING_DEFAULT_CHARSET);
  XtSetArg(al[ac],XmNlabelString,xmstr); ac++;
  XtSetArg(al[ac],XmNmnemonic,'F'); ac++;
  fdbutton = XmCreateCascadeButton(specialspulldown,"FdButton",al,ac);
  XtAddCallback(fdbutton,XmNactivateCallback,CalculateFractalDimension,location);
  XtManageChild(fdbutton);
  XmStringFree(xmstr);
  /*
   * return value of this function
   */
  return(leftrightmenu);
}
```

```
/**************************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : messages.c
 * Purpose          : Contains callbacks for helpmessages and warnings
 * Part of          : IFS-CODEC
 * Created on       : June 02, 1992
 * Created by       : Emile van Duren
 * Last modified on: November 20, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 **************************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
float PixsetDistance();        /* contained in genetic.c  */
void Passage();                /* contained in control.c  */
void Unmanage();               /* contained in control.c  */
Drawdata *GetWindowData();     /* contained in control.c  */
Widget GetWindow();            /* contained in control.c  */


void CreateMessageBox(w,help_str,what,location)
     Widget w;
     char   *help_str[];
     char   what,location;
{
  Widget      messagebox,messagetext;
  XmString    xmstr;
  int         i,code;
  /*
   * create messagebox
   */
  ac = 0;
  switch(what)
  {
    case 'A':
      xmstr = XmStringCreate("help information",XmSTRING_DEFAULT_CHARSET);
      code = 0;
      break;
    case 'B':
      xmstr = XmStringCreate("help information",XmSTRING_DEFAULT_CHARSET);
      if (location == 'l') code = 2;
      if (location == 'r') code = 3;
      break;
    case 'C':
      xmstr = XmStringCreate("warning",XmSTRING_DEFAULT_CHARSET);
      if (location == 'l') code = 4;
      if (location == 'r') code = 5;
      break;
    case 'D':
      xmstr = XmStringCreate("warning",XmSTRING_DEFAULT_CHARSET);
      if (location == 'l') code = 6;
      if (location == 'r') code = 7;
      break;
    case 'E':
      xmstr = XmStringCreate("message",XmSTRING_DEFAULT_CHARSET);
      if (location == 'l') code = 8;
      if (location == 'r') code = 9;
      break;
    case 'F':
      xmstr = XmStringCreate("message",XmSTRING_DEFAULT_CHARSET);
      code = 10;
      break;
    case 'G':
      xmstr = XmStringCreate("Copyright(c)",XmSTRING_DEFAULT_CHARSET);
      code = 12;
      break;
  }
  XtSetArg(al[ac],XmNdialogTitle,xmstr); ac++;
  XtSetArg(al[ac],XmNautoUnmanage,FALSE); ac++;
  messagebox = XmCreateMessageDialog(w,"MessageBox",al,ac);
  XmStringFree(xmstr);
  /*
   * we don't need the helpbutton and the cancelbutton, so unmanage them
   */
  XtUnmanageChild(XmMessageBoxGetChild(messagebox,XmDIALOG_HELP_BUTTON));
  XtUnmanageChild(XmMessageBoxGetChild(messagebox,XmDIALOG_CANCEL_BUTTON));
  /*
   * retreive the message text widget and left justify the text
   */
  ac = 0;
  XtSetArg(al[ac],XmNalignment,XmALIGNMENT_BEGINNING); ac++;
  messagetext = XmMessageBoxGetChild(messagebox,XmDIALOG_MESSAGE_LABEL);
  /*
   * add OK-callback
   */
  XtAddCallback(messagebox,XmNokCallback,Passage,code);
  XtAddCallback(messagebox,XmNokCallback,Unmanage,messagebox);
  /*
   * count the text up to the first NULL string
   */
  for (i=0;help_str[i][0]!='\0';i++);
  /*
   * convert the stringarray to an XmString array and set it as the
   * messagetext
   */
  ac = 0;
  xmstr = StringArrayToXmString(help_str,i);
  XtSetArg(al[ac],XmNmessageString,xmstr); ac++;
  XtSetValues(messagebox,al,ac);
  /*
   * display the helpmessage
   */
  XtManageChild(messagebox);
  XmStringFree(xmstr);
}


void MainHelpCallback(w)
     Widget    w;
{
  static char *help_str[] = {
    "1) The upper two screens are textareas",
    "2) The lower two screens are pixeldrawingareas",
    "3) The left and rightwindow contain the same utilities",
    "4) The initial origins are set to center of areas, i.e. (0,0)",
    ""};
  if (firsthelp)
    {
      CreateMessageBox(w,help_str,'A',NULL);
      firsthelp = FALSE;
    }
}


void FileHelpCallback(w,location)
     Widget    w;
     char      location;
{
  Drawdata *data;
  static char *help_str[] = {
    "Specify a filename WITHOUT extension, because the program",
    "will determine the appropriate extension itself. By looking",
    "at the file extensions (with UNIX command ls) you can see",
    "what kind of data each file contains:",
    " ",
    "filename.img contains a sequence of integer (x,y) coordinates,",
    "             which specify a user drawn image.",
    "filename.col contains a sequence of integer (x,y) coordinates,",
    "             which specify the image of a user created collage.",
    "filename.man contains the code of an IFS that is specified by",
    "             the corresponding collage.",
    "filename.aut contains the code of an IFS that is generated by",
    "            the program to approximate an image.",
    "filename.ifs contains the code of an IFS that is created by the",
    "            user with a source editor. For more information,",
    "            push the Viewedit IFS code button.",
    " ",
    "If you want to save a rendered image, just push the savebutton",
    "in the viewedit source image menu. You will find the rendered",
    "image in filename.img which you can rename in UNIX if you wish.",
    ""};
  data = GetWindowData(location,'p');
  if (data->first.filehelp)
    {
      CreateMessageBox(w,help_str,'B',location);
      data->first.filehelp = FALSE;
    }
}


void FileOpenWarning(w,location)
    Widget    w;
    char location;
{
  Drawdata *data;
  static char *help_str[] = {
    "WARNING: file could not be opened",""};
  data = GetWindowData(location,'p');
  if (data->first.fileopenwarning)
    {
      CreateMessageBox(w,help_str,'C',location);
      data->first.fileopenwarning = FALSE;
    }
}


void OriginWarning(w,location)
    Widget w;
    char location;
{
  Drawdata *data;
  static char *help_str[] = {
    "WARNING: changing coordinates while tiling is not very,",
    "smart, because this will lead to an improper IFS-code",""};
  data = GetWindowData(location,'p');
  if (data->first.originwarning)
    {
      CreateMessageBox(w,help_str,'D',location);
      data->first.originwarning = FALSE;
    }
}


void VieweditIFSTableMessage(w,location)
    Widget    w;
    char      location;
{
  Drawdata *data;
  static char *help_str[] = {
    "You can do this with VI or Emacs, (obey the following format !)",
    " ",
    "Each line in the transformation table contains 7 elements, (i.e.",
    "a,b,c,d,e,f,p), contained in the following floating intervals:",
    " ",
    "a,b,c,d in interval [-1.000,1.000]   [dimensionless]",
    "e,f     in interval [-250.00,250.00] [pixels]",
    "p       in interval [0.000,1.000]    [x0.01 percent]",
    " ",
    "Each line must be terminated by a carriage return. If you want",
    "the program to determine the probabilities p, just set the file",
    "extension to .man, select the filename and push the save collage",
    "button in the interactive IFS encoding menupane.",
    ""};
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  if (data->first.tablemessage)
```

```
    {
      CreateMessageBox(w,help_str,'E',location);
      data->first.tablemessage = FALSE;
    }
}


void PrintMessage(w)
     Widget w;
{
  static char *help_str[] = {
    "You can do this by opening an xterm window",
    "and applying the following procedure:",
    " ",
    "xwd > filename_1",
    " ",
    "After the cursor has changed from an arrow into a",
    "crosshair, put the cursor in the desired window and",
    "push the left mousebutton. After the second beep type:",
    " ",
    "xpr -dev ps filename_1 > filename_2",
    " ",
    "You can view the printfile with:",
    " ",
    "gs filename_2",
    " ",
    "You can print the printfile with:",
    " ",
    "lp -dhpps filename_2",
    ""};
  if (printmessage)
    {
      CreateMessageBox(w,help_str,'F');
      printmessage = FALSE;
    }
}


void PixsetMessage(w)
     Widget w;
{
  float distance;
  distance = PixsetDistance(w,1);
  printf("pixsetdistance = %f\n",distance);
}


void CopyrightMessage(w)
     Widget w;
{
  static char *help_str[] = {
    "Copyright(c): Eindhoven University of Technology (TUE,NL)",
    "              Faculty of Electrical Engineering (E)",
    "              Design Automation Group (ES)",
    " ",
    "Author      : Emile van Duren",
    " ",
    "All rights reserved. No part of this software shall be",
    "reproduced, stored in a retrieval system, or transmitted",
    "by any means without written permission from NL/TUE/E/ES.",
    ""};
  if (copyrightmessage)
    {
      CreateMessageBox(w,help_str,'G');
      copyrightmessage = FALSE;
    }
}
```

```
/*****************************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : operators.c
 * Purpose         : Contains genetic operators for genetic algorithm
 * Part of         : IFS-CODEC
 * Created on      : September 30, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 01, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 *****************************************************************************************/

#include "all.h"


int SelectGeneticOperator()
{
  /*
   * initialize operator selection probabilities
   */
  static float operator[] = { P_MACRO, P_MICRO, P_MIXED, P_MUTATE, P_JUMP, P_SCALE, P_ROTATE, P_TRANSLATE
};
  int i=0,p=0;
  unsigned long seed;
  Boolean found = FALSE;
  /*
   * roulette wheel operator selection, so the chance of each operator to be
   * selected is directly proportional to its probability as initialized above
   *
   */
  seed = rand();
  while (i < 8 && !found)
    {
      p += 32767*(operator[i]);
      if (seed <= p) found = TRUE;
      i++;
    }
  /*
   * return value of this function
   */
  return(i);
}

void MacroCrossOver(parent1,parent2,child1,child2)
     Chromosome *parent1,*parent2,*child1,*child2;
{
  char *monitor = "monitor.dat";
  int mask;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * initialize child to be equal to parent
   */
  child1->a = parent1->a;
  child1->b = parent1->b;
  child1->c = parent1->c;
  child1->d = parent1->d;
  child1->e = parent1->e;
  child1->f = parent1->f;
  child2->a = parent2->a;
  child2->b = parent2->b;
  child2->c = parent2->c;
  child2->d = parent2->d;
  child2->e = parent2->e;
  child2->f = parent2->f;
  /*
   * generate random mask as element of {1..63}
   */
  mask = (int) (rand()/528.5 + 1);
  /*
```

```
     * update file monitor.dat (next 3 lines are meant for debugging purposes)
     */
    if (debug)
      {
        file = fopen(monitor,"a");
        fprintf(file,"mask:%2d\n",mask);
        fclose(file);
      }
    /*
     * if mask contains a binary 0 at position i then child1 gets the according
     * gene of parent1 and the same holds for child2 and parent2
     *
     * if mask contains a binary 1 at position i then child1 gets the according
     * gene of parent2 and the same holds for child2 and parent1
     *
     * example:  parent1  a1,b1,c1,d1,e1,f1
     *           parent2  a2,b2,c2,d2,e2,f2
     *           mask      0  1  1  0  1  0
     *           child1   a1,b2,c2,d1,e2,f1
     *           child2   a2,b1,c1,d2,e1,f2
     */
    if (mask%2 != 0 && mask != 0)
      {
        mask--;
        child1->f = parent2->f;
        child2->f = parent1->f;
      }
    mask /= 2;
    if (mask%2 != 0 && mask != 0)
      {
        mask--;
        child1->e = parent2->e;
        child2->e = parent1->e;
      }
    mask /= 2;
    if (mask%2 != 0 && mask != 0)
      {
        mask--;
        child1->d = parent2->d;
        child2->d = parent1->d;
      }
    mask /= 2;
    if (mask%2 != 0 && mask != 0)
      {
        mask--;
        child1->c = parent2->c;
        child2->c = parent1->c;
      }
    mask /= 2;
    if (mask%2 != 0 && mask != 0)
      {
        mask--;
        child1->b = parent2->b;
        child2->b = parent1->b;
      }
    mask /= 2;
    if (mask%2 != 0 && mask != 0)
      {
        mask--;
        child1->a = parent2->a;
        child2->a = parent1->a;
      }
}

void MicroCrossOver(parent1,parent2,child1,child2)
     Chromosome *parent1,*parent2,*child1,*child2;
{
  char *monitor = "monitor.dat";
  int mask;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * generate random mask as element of {1..255}
   */
  mask = (int) (rand()/129.0 + 1);
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"mask:%2d\n",mask);
      fclose(file);
    }
  /*
   * if mask contains a binary 0 at position i then child1 gets the same
   * nucleotide as parent1 at position i in the concerning gene and child2
   * gets the same nucleotide as parent2 at position i in the concerning gene
   *
   * if mask contains a binary 1 at position i then child1 gets the same
   * nucleotide as parent2 at position i in the concerning gene and child2
   * gets the same nucleotide as parent1 at position i in the concerning gene
   *
   * example:    mask               parent1(gene i)     parent2(gene i)
   *             0 0 1 1 1 0 1 1    1 1 0 0 1 0 0 1     0 1 1 0 1 1 1 0
   *
   *             "0"-mask           1 1 . . . 0 . .     0 1 . . . 1 . .
   *             "1"-mask           . . 1 0 1 . 1 0     . . 0 0 1 . 0 1
   *
   *             merge              1 1 1 0 1 0 1 0     0 1 0 0 1 1 0 1
   *                                child1(gene i)      child2(gene i)
   */
  child1->a = (~mask & parent1->a) | (mask & parent2->a);
  child1->b = (~mask & parent1->b) | (mask & parent2->b);
  child1->c = (~mask & parent1->c) | (mask & parent2->c);
  child1->d = (~mask & parent1->d) | (mask & parent2->d);
  child1->e = (~mask & parent1->e) | (mask & parent2->e);
  child1->f = (~mask & parent1->f) | (mask & parent2->f);
  child2->a = (~mask & parent2->a) | (mask & parent1->a);
  child2->b = (~mask & parent2->b) | (mask & parent1->b);
  child2->c = (~mask & parent2->c) | (mask & parent1->c);
  child2->d = (~mask & parent2->d) | (mask & parent1->d);
  child2->e = (~mask & parent2->e) | (mask & parent1->e);
  child2->f = (~mask & parent2->f) | (mask & parent1->f);
  if (child1->a == 0) child1->a = 128;
  if (child1->b == 0) child1->b = 128;
  if (child1->c == 0) child1->c = 128;
  if (child1->d == 0) child1->d = 128;
  if (child1->e == 0) child1->e = 128;
  if (child1->f == 0) child1->f = 128;
  if (child2->a == 0) child2->a = 128;
  if (child2->b == 0) child2->b = 128;
  if (child2->c == 0) child2->c = 128;
  if (child2->d == 0) child2->d = 128;
  if (child2->e == 0) child2->e = 128;
  if (child2->f == 0) child2->f = 128;
}

void MixedCrossOver(parent1,parent2,child1,child2)
     Chromosome *parent1,*parent2,*child1,*child2;
{
  char *monitor = "monitor.dat";
  int mask1,mask2;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * initialize child to be equal to parent
   */
  child1->a = parent1->a;
  child1->b = parent1->b;
  child1->c = parent1->c;
  child1->d = parent1->d;
  child1->e = parent1->e;
  child1->f = parent1->f;
  child2->a = parent2->a;
  child2->b = parent2->b;
  child2->c = parent2->c;
  child2->d = parent2->d;
  child2->e = parent2->e;
  child2->f = parent2->f;
  /*
   * generate random mask1 as element of {1..63} and random mask2 as element
   * of {1..255}
   */
  mask1 = (int) (rand()/528.5 + 1);
  mask2 = (int) (rand()/129.0 + 1);
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"mask1:%2d  mask2:%2d\n",mask1,mask2);
      fclose(file);
    }
  /*
   * mixed crossover combines the effect of macro crossover and micro
   * crossover, i.e. mask2 is the mask for micro crossover which is applied
   * to the genes according to the ones in mask1 (see also MacroCrossOver()
   * and MicroCrossOver()).
   */
  if (mask1%2 != 0 && mask1 !=0)
    {
      mask1--;
      child1->f = (~mask2 & parent1->f) | (mask2 & parent2->f);
      child2->f = (~mask2 & parent2->f) | (mask2 & parent1->f);
      if (child1->f == 0) child1->f = 128;
      if (child2->f == 0) child2->f = 128;
    }
  mask1 /= 2;
  if (mask1%2 != 0 && mask1 !=0)
    {
      mask1--;
      child1->e = (~mask2 & parent1->e) | (mask2 & parent2->e);
      child2->e = (~mask2 & parent2->e) | (mask2 & parent1->e);
      if (child1->e == 0) child1->e = 128;
      if (child2->e == 0) child2->e = 128;
    }
  mask1 /= 2;
  if (mask1%2 != 0 && mask1 !=0)
    {
      mask1--;
      child1->d = (~mask2 & parent1->d) | (mask2 & parent2->d);
      child2->d = (~mask2 & parent2->d) | (mask2 & parent1->d);
      if (child1->d == 0) child1->d = 128;
      if (child2->d == 0) child2->d = 128;
    }
  mask1 /= 2;
  if (mask1%2 != 0 && mask1 !=0)
    {
      mask1--;
      child1->c = (~mask2 & parent1->c) | (mask2 & parent2->c);
      child2->c = (~mask2 & parent2->c) | (mask2 & parent1->c);
      if (child1->c == 0) child1->c = 128;
      if (child2->c == 0) child2->c = 128;
    }
  mask1 /= 2;
  if (mask1%2 != 0 && mask1 !=0)
    {
      mask1--;
      child1->b = (~mask2 & parent1->b) | (mask2 & parent2->b);
      child2->b = (~mask2 & parent2->b) | (mask2 & parent1->b);
      if (child1->b == 0) child1->b = 128;
      if (child2->b == 0) child2->b = 128;
    }
  mask1 /= 2;
  if (mask1%2 != 0 && mask1 !=0)
    {
      mask1--;
      child1->a = (~mask2 & parent1->a) | (mask2 & parent2->a);
      child2->a = (~mask2 & parent2->a) | (mask2 & parent1->a);
      if (child1->a == 0) child1->a = 128;
      if (child2->a == 0) child2->a = 128;
    }
}
```

```
void Mutate(parent,child)
    Chromosome *parent,*child;
{
  char *monitor = "monitor.dat";
  int i,mask1,mask2=1;
  unsigned long seed;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * initialize child to be equal to parent
   */
  child->a = parent->a;
  child->b = parent->b;
  child->c = parent->c;
  child->d = parent->d;
  child->e = parent->e;
  child->f = parent->f;
  /*
   * generate random mask1 as element of {1..6}
   */
  mask1 = (int) (rand()/6553.4 + 1);
  /*
   * generate random mask2 as element of {1,2,4,8,16,32,64,128}, therefore
   * first generate a random seed as element of {0..7} and calculate mask2
   * as 2 exp seed.
   */
  seed = (int) (rand()/4681.0);
  for (i = 0; i < seed; i++)
    {
      mask2 *= 2;
    }
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"mask1:%2d  mask2:%2d\n",mask1,mask2);
      fclose(file);
    }
  /*
   * mask2 contains a single 1 digit at position i which will force the
   * nucleotide at position i in the gene at position mask1 to be inverted
   */
  switch(mask1)
  {
    case 1:
      child->a = (mask2)^(parent->a);
      if (child->a == 0) child->a = 128;
      break;
    case 2:
      child->b = (mask2)^(parent->b);
      if (child->b == 0) child->b = 128;
      break;
    case 3:
      child->c = (mask2)^(parent->c);
      if (child->c == 0) child->c = 128;
      break;
    case 4:
      child->d = (mask2)^(parent->d);
      if (child->d == 0) child->d = 128;
      break;
    case 5:
      child->e = (mask2)^(parent->e);
      if (child->e == 0) child->e = 128;
      break;
    case 6:
      child->f = (mask2)^(parent->f);
      if (child->f == 0) child->f = 128;
      break;
  }
}
```

```
void Jump(parent,child)
    Chromosome *parent,*child;
{
  char *monitor = "monitor.dat";
  float array[6],dummy;
  int mask1 = 0,mask2 = 0;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * generate random mask1 as element of {1..6} and mask2 as element of
   * {1..6}\mask1, since mask1 and mask2 must differ
   */
  while (mask1 == mask2)
    {
      mask1 = (int) (rand()/6553.4 + 1);
      mask2 = (int) (rand()/6553.4 + 1);
    }
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"mask1:%2d  mask2:%2d\n",mask1,mask2);
      fclose(file);
    }
  /*
   * child becomes equal to parent with genes at positions mask1 and mask2 swapped
   *
   * example:     parent        = a,b,c,d,e,f
   *              (mask1,mask2) = (2,6)
   *              child         = a,f,c,d,e,b
   */
  array[0] = parent->a;
  array[1] = parent->b;
  array[2] = parent->c;
  array[3] = parent->d;
  array[4] = parent->e;
  array[5] = parent->f;
  dummy = array[mask1-1];
  array[mask1-1] = array[mask2-1];
  array[mask2-1] = dummy;
  child->a = array[0];
  child->b = array[1];
  child->c = array[2];
  child->d = array[3];
  child->e = array[4];
  child->f = array[5];
}


void Scale(parent,child)
    Chromosome *parent,*child;
{
  char *monitor = "monitor.dat";
  int mask = 0;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * generate random mask as element of {1..126}
   */
  mask = (int) (rand()/262.1 + 1);
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"mask:%2d\n",mask);
      fclose(file);
    }
  /*
   * child will become a scaled version of the parent, such that the signs of
   * the parameters a,b,c, and d remains unchanged, thus for x in {a,b,c,d}:
   * bit 0 of child->x = bit 0 of parent->x
   * bit 1-7 of child->x = ((bit 1-7 of parent->x) + mask) mod 127
   *
   * example:      parent->a = 11001011 = -75
   *               parent->b = 00110010 = 50
   *               parent->c = 10000101 = -5
   *               parent->d = 01110111 = 119
   *               mask      =  1000011 = 67
   *               child->a  = 10001111 = -40
   *               child->b  = 01110101 =  26
   *               child->c  = 11001000 =  -3
   *               child->d  = 00111011 =  63
   */
  if ((parent->a - 128) > 0)
    {
      child->a = (int) (((parent->a - 128)*mask)/127.0 + 128.5);
    }
  else
    {
      child->a = (int) (((parent->a)*mask)/127.0 + 0.5);
    }
  if ((parent->b - 128) > 0)
    {
      child->b = (int) (((parent->b - 128)*mask)/127.0 + 128.5);
    }
  else
    {
      child->b = (int) (((parent->b)*mask)/127.0 + 0.5);
    }
  if ((parent->c - 128) > 0)
    {
      child->c = (int) (((parent->c - 128)*mask)/127.0 + 128.5);
    }
  else
    {
      child->c = (int) (((parent->c)*mask)/127.0 + 0.5);
    }
  if ((parent->d - 128) > 0)
    {
      child->d = (int) (((parent->d - 128)*mask)/127.0 + 128.5);
    }
  else
    {
      child->d = (int) (((parent->d)*mask)/127.0 + 0.5);
    }
  if (child->a == 0) child->a = 128;
  if (child->b == 0) child->b = 128;
  if (child->c == 0) child->c = 128;
  if (child->d == 0) child->d = 128;
  child->e = parent->e;
  child->f = parent->f;
}


void Rotate(parent,child)
     Chromosome *parent,*child;
{
  char *monitor = "monitor.dat";
  double phi;
  int mask;
  float dummy,pi = PI,acos,asin,bcos,bsin,ccos,csin,dcos,dsin;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * generate random mask as element of {1..359} degrees
   */
  mask = (int) (rand()/91.527 + 1);
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"phi: %3.2f\n",phi);
      fclose(file);
    }
```

```
/*
 * calculate rotation angle phi in radians
 */
phi = (double) mask*2*pi/360;
/*
 * child will become a rotated version of the parent, such that the
 * transformation matrix is multiplied by the standard rotation matrix:
 *
 * | x' |   | cos(phi) - sin(phi) |   | a b |   | x |   | e |
 * |    | = |                     | x |     | x |   | + |   |
 * | y' |   | sin(phi) + cos(phi) |   | c d |   | y |   | f |
 *
 * example: phi = 77 [degrees] = 1.34 [radians]
 *          parent->a = 203 = -75
 *          parent->b =  50 =  50
 *          parent->c = 133 = -5
 *          parent->d = 119 = 119
 *          child->a  = (int) -75*cos(77) -  -5*sin(77) =  -12 = 140
 *          child->b  = (int)  50*cos(77) - 119*sin(77) = -105 = 233
 *          child->c  = (int) -75*sin(77) +  -5*cos(77) =  -74 = 202
 *          child->d  = (int)  50*sin(77) + 119*cos(77) =   75 =  75
 */
if ((parent->a - 128) > 0)
  {
    acos = (128 - parent->a)*cos(phi);
    asin = (128 - parent->a)*sin(phi);
  }
else
  {
    acos = (parent->a)*cos(phi);
    asin = (parent->a)*sin(phi);
  }
if ((parent->b - 128) > 0)
  {
    bcos = (128 - parent->b)*cos(phi);
    bsin = (128 - parent->b)*sin(phi);
  }
else
  {
    bcos = (parent->b)*cos(phi);
    bsin = (parent->b)*sin(phi);
  }
if ((parent->c - 128) > 0)
  {
    ccos = (128 - parent->c)*cos(phi);
    csin = (128 - parent->c)*sin(phi);
  }
else
  {
    ccos = (parent->c)*cos(phi);
    csin = (parent->c)*sin(phi);
  }
if ((parent->d - 128) > 0)
  {
    dcos = (128 - parent->d)*cos(phi);
    dsin = (128 - parent->d)*sin(phi);
  }
else
  {
    dcos = (parent->d)*cos(phi);
    dsin = (parent->d)*sin(phi);
  }
/*
 * calculate children
 */
if ((int) (acos - csin) < 0)
  {
    child->a = ((int) ((csin - acos) + 128.5))%255;
  }
else
  {
    child->a = ((int) ((acos - csin) + 0.5))%255;
  }
if ((int) (bcos - dsin) < 0)
  {
      child->b = ((int) ((dsin - bcos) + 128.5))%255;
    }
  else
    {
      child->b = ((int) ((bcos - dsin) + 0.5))%255;
    }
  if ((int) (asin + ccos) < 0)
    {
      child->c = ((int) (-(asin + ccos) + 128.5))%255;
    }
  else
    {
      child->c = ((int) ((asin + ccos) + 0.5))%255;
    }
  if ((int) (bsin + dcos) < 0)
    {
      child->d = ((int) (-(bsin + dcos) + 128.5))%255;
    }
  else
    {
      child->d = ((int) ((bsin + dcos) + 0.5))%255;
    }
  if (child->a == 0) child->a = 128;
  if (child->b == 0) child->b = 128;
  if (child->c == 0) child->c = 128;
  if (child->d == 0) child->d = 128;
  child->e = parent->e;
  child->f = parent->f;
}


void Translate(parent,child)
     Chromosome *parent,*child;
{
  char *monitor = "monitor.dat";
  int i,mask;
  unsigned long seed;
  Boolean debug = DEBUG;
  FILE *file;
  /*
   * generate random mask as element of {1..255}
   */
  mask = (int) (rand()/129.0 + 1);
  /*
   * update file monitor.dat (next 3 lines are meant for debugging purposes)
   */
  if (debug)
    {
      file = fopen(monitor,"a");
      fprintf(file,"mask:%2d\n",mask);
      fclose(file);
    }
  /*
   * mask is added modulo 255 with the shift vector
   */
  child->a = parent->a;
  child->b = parent->b;
  child->c = parent->c;
  child->d = parent->d;
  child->e = (parent->e + mask)%255;
  child->f = (parent->f + mask)%255;
  if (child->e == 0) child->e = 128;
  if (child->f == 0) child->f = 128;
}
```

```
/*******************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : render.c
 * Purpose          : Contains procedures for rendering IFS-codes
 * Part of          : IFS-CODEC
 * Created on       : July 13, 1992
 * Created by       : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by      : Emile van Duren
 * Remarks          : Zoomingfacility must be added between lines 280 and 300
 ******************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void ClearTextHandler();        /* contained in handle.c    */
void DrawAxes();                /* contained in draw.c      */
void DrawRectangle();           /* contained in draw.c      */
void FileOpenWarning();         /* contained in messages.c  */
void IterationHandler();        /* contained in handle.c    */
void SourceFileHandler();       /* contained in handle.c    */
Drawdata *GetWindowData();      /* contained in control.c   */
Textdata *GetTextData();        /* contained in control.c   */
Widget GetWindow();             /* contained in control.c   */


void GraphicalView(w,code)
     Widget w;
     int code;
{
  char location,*ext,*filename,*savefile;
  int x1,x2,x3,x4,y1,y2,y3,y4;
  float a,b,c,d,e,f,p,xleft,xright,yupper,ylower,width,height;
  float xaleft,xaright,ybupper,yblower;
  unsigned long pixel;
  Drawdata *data;
  FILE *file;
  Textdata *text;
  if (code %2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  if (code == 0) ext = ".ifs";
  if (code == 2) ext = ".man";
  if (code == 4) ext = ".aut";
  savefile = calloc(strlen(data->filename)+5,sizeof(char));
  strcpy(savefile,data->filename);
  if ((file = fopen(strcat(savefile,ext),"r")) == NULL)
    {
      FileOpenWarning(w,location);
    }
  else
    {
      /*
       * clear out all text except origin in textwindow
       */
      ClearPixelArea(w,location);
      ClearTextHandler(w,location,287);
      text->mode = "mode        : Viewedit IFS-code (graphical view)";
      SourceFileHandler(w,location,ext);
      /*
       * set foreground to black and draw an untransformed dashed rectangle
       */
      XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
      DrawRectangle(w,location,'d');
      /*
       * determine size and position (screen coordinates) of base rectangle
       */
      width = 2*(data->width)/5;
      height = 2*(data->height)/5;
      xleft = data->xorigin - width/2;
      yupper = data->yorigin - height/2;
      xright = xleft + width;
      ylower = yupper + height;
      xaleft = xleft - data->xorigin;
      xaright = xright - data->xorigin;
      ybupper = data->yorigin - yupper;
      yblower = data->yorigin - ylower;
      /*
       * for all transformations in file
       */
      while (!feof(file))
       {
         /*
          * get transformation from file
          */
         fscanf(file,"%f ",&a);
         fscanf(file,"%f ",&b);
         fscanf(file,"%f ",&c);
         fscanf(file,"%f ",&d);
         fscanf(file,"%f ",&e);
         fscanf(file,"%f ",&f);
         fscanf(file,"%f\n",&p);
         /*
          * apply all transformations once the vertices of the rectangle
          * in data->pix
          */
         x1 = a*xaleft + b*ybupper + e + data->xorigin;
         y1 = -c*xaleft - d*ybupper - f + data->yorigin;
         x2 = a*xaright + b*ybupper + e + data->xorigin;
         y2 = -c*xaright - d*ybupper - f + data->yorigin;
         x3 = a*xaright + b*yblower + e + data->xorigin;
         y3 = -c*xaright - d*yblower - f + data->yorigin;
         x4 = a*xaleft + b*yblower + e + data->xorigin;
         y4 = -c*xaleft - d*yblower - f + data->yorigin;
         /*
          * and draw lines between the transformed vertices
          */
         XDrawLine(XtDisplay(w),data->pix,data->gc,x1,y1,x2,y2);
         XDrawLine(XtDisplay(w),data->pix,data->gc,x2,y2,x3,y3);
         XDrawLine(XtDisplay(w),data->pix,data->gc,x3,y3,x4,y4);
         XDrawLine(XtDisplay(w),data->pix,data->gc,x4,y4,x1,y1);
       }
      /*
       * finally, copy data->pix to screen and draw axes if necessary
       */
      XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
             data->width,data->height,0,0);
      if (data->axes) DrawAxes(w,location);
    }
  free(savefile);
  savefile = NULL;
}


void DeterministicRendering(w,code)
     Widget w;
     int code;
{
  char location,*ext,*filename,*savefile;
  float pi,xt,yt;
  int col,count = 0,index,iteration,row,x,y = 0;
  unsigned long pixel;
  Boolean skipflag = FALSE;
  Drawdata *data;
```

```
FILE *file;
IFScode *pointer;
XImage *image;
/*
 * initialization
 */
pi = PI;
pointer = NULL;
if (code %2 == 0)
  {
    location = 'l';
  }
else
  {
    location = 'r';
    code--;
  }
w = GetWindow(location,'p');
data = GetWindowData(location,'p');
/*
 * determine file extension
 */
if (code == 0) ext = ".ifs";
if (code == 2) ext = ".man";
if (code == 4) ext = ".aut";
savefile = calloc(strlen(data->filename)+5,sizeof(char));
strcpy(savefile,data->filename);
if ((file = fopen(strcat(savefile,ext),"r")) == NULL)
  {
    FileOpenWarning(w,location);
    skipflag = TRUE;
  }
else
  {
    /*
     * update sourcefile and savefile name in textwindow
     */
    SourceFileHandler(w,location,ext);
    /*
     * if file could be opened
     */
    while (!feof(file))
      {
        /*
         * allocate memory for transformations
         */
        if (pointer == NULL)
          {
            pointer = (IFScode *) malloc(sizeof(IFScode));
          }
        else
          {
            pointer = (IFScode *) realloc(pointer,(count+1)*sizeof(IFScode));
          }
        /*
         * get transformation from file and store it in memory
         */
        fscanf(file,"%f ",&((pointer + count)->a));
        fscanf(file,"%f ",&((pointer + count)->b));
        fscanf(file,"%f ",&((pointer + count)->c));
        fscanf(file,"%f ",&((pointer + count)->d));
        fscanf(file,"%f ",&((pointer + count)->e));
        fscanf(file,"%f ",&((pointer + count)->f));
        fscanf(file,"%f\n",&((pointer + count)->p));
        count++;
      }
  }
fclose(file);
free(savefile);
savefile = NULL;
if (!skipflag)
  {
    /*
     * set foreground to black and plot startpixel
     */
```

```
XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
XDrawPoint(XtDisplay(w),data->pix,data->gc,data->xstart,data->ystart);
/*
 * for each iteration
 */
for (iteration = 0; iteration < data->niterations; iteration++)
 {
   /*
    * deterministic rendering with condensation, so don't clear the
    * screen between successive iterations
    */
   if (data->condensation)
     {
       /*
        * copy data->pix to screen an get image from data->pix;
        */
       XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
             data->width,data->height,0,0);
     }
   image = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                 data->height,1,XYPixmap);
   /*
    * deterministic rendering without condensation, so clear the
    * screen between successive iterations
    */
   if (!data->condensation)
     {
       XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
       XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,
                  data->width,data->height);
       XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
             data->width,data->height,0,0);
       XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,
                  data->width,data->height);
       XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
     }
   /*
    * for all pixels in image
    */
   for (row = 0; row < data->height; row++)
     {
       for (col = 0; col < data->width; col++)
        {
          /*
           * get pixel and if pixel is black
           */
          pixel = XGetPixel(image,col,row);
          if (pixel == 0)
            {
              /*
               * apply all transformations sequentially
               */
              for (index = 0; index < count; index++)
               {
                 /*
                  * transform coordinate to virtual origin
                  */
                 xt = col - data->xorigin;
                 yt = row - data->yorigin;
                 /*
                  * convert y from screen coordinates to Carthesian coordinates
                  */
                 yt = -yt;
                 /*
                  * apply affine transformation
                  */
                 x = ((pointer + index)->a)*xt +
                     ((pointer + index)->b)*yt +
                     ((pointer + index)->e);
                 y = ((pointer + index)->c)*xt +
                     ((pointer + index)->d)*yt +
                     ((pointer + index)->f);
                 /*
                  * rescale coordinate with data->zoom and data->offset
                  */
```

```
                        /*
                         * convert y from Carthesian coordinates to screen coordinates
                         */
                        y = -y;
                        /*
                         * transform coordinate to original position and plot it
                         */
                        x += data->xorigin;
                        y += data->yorigin;
                        XDrawPoint(XtDisplay(w),data->pix,data->gc,x,y);
                      }
                    }
                  }
                }
            /*
             * update amount of iterations in textdisplay
             */
            IterationHandler(location,iteration);
            /*
             * destroy image and draw axes if necessary
             */
            XDestroyImage(image);
            image = NULL;
          }
        XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
              data->width,data->height,0,0);
        if (data->axes) DrawAxes(w,location);
      }
  free(pointer);
  pointer = NULL;
}


void NondeterministicRendering(w,code)
     Widget w;
     int code;
{
  char location,*ext,*savefile;
  float pi,xt,yt;
  int col,count = 0,i,row,x,y = 0;
  unsigned long iteration,p,pixel,seed;
  Boolean found,skipflag = FALSE;
  Drawdata *data;
  FILE *file;
  IFScode *ptr;
  XImage *image;
  /*
   * initialization
   */
  pi = PI;
  ptr = NULL;
  if (code %2 == 0)
    {
      location = 'l';
    }
  else
    {
      location = 'r';
      code--;
    }
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  /*
   * determine file extension
   */
  if (code == 0) ext = ".ifs";
  if (code == 2) ext = ".man";
  if (code == 4) ext = ".aut";
  savefile = calloc(strlen(data->filename)+5,sizeof(char));
  strcpy(savefile,data->filename);
  if ((file = fopen(strcat(savefile,ext),"r")) == NULL)
    {
      FileOpenWarning(w,location);
      skipflag = TRUE;
```

```
 }
else
 {
   /*
    * update sourcefile and savefile name in textwindow
    */
   SourceFileHandler(w,location,ext);
   /*
    * if file could be opened
    */
   while (!feof(file))
    {
      /*
       * allocate memory for transformations
       */
      if (ptr == NULL)
        {
          ptr = (IFScode *) malloc(sizeof(IFScode));
        }
      else
        {
          ptr = (IFScode *) realloc(ptr,(count+1)*sizeof(IFScode));
        }
      /*
       * get transformation from file and store it in memory
       */
      fscanf(file,"%f ",&((ptr + count)->a));
      fscanf(file,"%f ",&((ptr + count)->b));
      fscanf(file,"%f ",&((ptr + count)->c));
      fscanf(file,"%f ",&((ptr + count)->d));
      fscanf(file,"%f ",&((ptr + count)->e));
      fscanf(file,"%f ",&((ptr + count)->f));
      fscanf(file,"%f\n",&((ptr + count)->p));
      count++;
    }
 }
fclose(file);
free(savefile);
savefile = NULL;
if (!skipflag)
 {
   /*
    * set foreground to black and plot startpixel
    */
   XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
   XDrawPoint(XtDisplay(w),data->pix,data->gc,data->xstart,data->ystart);
   /*
    * nondeterministic rendering with condensation
    */
   if (data->condensation)
    {
      for (iteration = 0; iteration < data->niterations; iteration++)
        {
          /*
           * copy data->pix to screen and get image from data->pix
           */
          XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
                data->width,data->height,0,0);
          image = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                      data->height,1,XYPixmap);
          /*
           * for all pixels in image
           */
          for (row = 0; row < data->height; row++)
           {
             for (col = 0; col < data->width; col++)
               {
                 /*
                  * get pixel and if pixel is black
                  */
                 pixel = XGetPixel(image,col,row);
                 if (pixel == 0)
                  {
                    /*
                     * because rand() returns an integer between 0 and 32767,
                 * rescale the fractional probabilities for each
                 * transformation appropriately to this returnvalue and
                 * select a transformation according to its assigned
                 * probability
                 */
                p = 0;i = 0;
                found = FALSE;
                seed = rand();
                while (i < count && !found)
                  {
                    p += 32767*((ptr + i)->p);
                    if (seed <= p) found = TRUE;
                    i++;
                  }
                i--;
                /*
                 * transform coordinate to virtual origin
                 */
                xt = col - data->xorigin;
                yt = row - data->yorigin;
                /*
                 * convert y from screen coordinates to Carthesian
                 * coordinates
                 */
                yt = -yt;
                /*
                 * apply affine transformation
                 */
                x = ((ptr + i)->a)*xt + ((ptr + i)->b)*yt + (ptr + i)->e;
                y = ((ptr + i)->c)*xt + ((ptr + i)->d)*yt + (ptr + i)->f;
                /*
                 * rescale coordinate with data->zoom and data->offset
                 */


                /*
                 * convert y from Carthesian coordinates to screen
                 * coordinates
                 */
                y = -y;
                /*
                 * convert coordinate to its original position and plot it
                 */
                x += data->xorigin;
                y += data->yorigin;
                XDrawPoint(XtDisplay(w),data->pix,data->gc,x,y);
              }
            }
        }
        /*
         * update amount of iterations in textdisplay
         */
        IterationHandler(location,iteration);
        /*
         * destroy image and draw axes if necessary
         */
        XDestroyImage(image);
        image = NULL;
      }
    XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
            data->width,data->height,0,0);
    if (data->axes) DrawAxes(w,location);
  }
  /*
   * nondeterministic rendering without condensation (orbit calculation)
   */
  if (!data->condensation)
  {
    /*
     * initialize starting point of orbit
     */
    x = data->xstart;
    y = data->ystart;
    /*
     * apply 1000 times selected amount of iterations
```

```
       */
      for (iteration = 0; iteration < 1000*data->niterations; iteration++)
        {
          /*
           * update amount of iterations in textdisplay
           */
          if ((iteration+1) % 1000 == 0) IterationHandler(location,iteration);
          /*
           * because rand() returns an integer between 0 and 32767,
           * rescale the fractional probabilities for each
           * transformation appropriately to this returnvalue and
           * select a transformation according to its assigned
           * probability
           */
          p = 0;i = 0;
          found = FALSE;
          seed = rand();
          while (i < count && !found)
           {
             p += 32767*((ptr + i)->p);
             if (seed <= p) found = TRUE;
             i++;
           }
          i--;
          /*
           * transform coordinate to virtual origin
           */
          xt = x - data->xorigin;
          yt = y - data->yorigin;
          /*
           * convert y from screen coordinates to Carthesian coordinates
           */
          yt = -yt;
          /*
           * apply affine transformation
           */
          x = ((ptr + i)->a)*xt + ((ptr + i)->b)*yt + (ptr + i)->e;
          y = ((ptr + i)->c)*xt + ((ptr + i)->d)*yt + (ptr + i)->f;
          /*
           * rescale coordinate with data->zoom and data->offset
           */


          /*
           * convert y from Carthesian coordinates to screen coordinates
           */
          y = -y;
          /*
           * convert coordinate to its original position and plot it
           */
          x += data->xorigin;
          y += data->yorigin;
          /*
           * draw point in data->pix and also on screen
           */
          XDrawPoint(XtDisplay(w),data->pix,data->gc,x,y);
          XDrawPoint(XtDisplay(w),XtWindow(w),data->gc,x,y);
        }
    }
    free(ptr);
    ptr = NULL;
  }
}
```

```
/*************************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : specials.c
 * Purpose          : Rendering of Bifurcation diagram and Mandelbrot set
 * Part of          : IFS-CODEC
 * Created on       : July 31, 1992
 * Created by       : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by      : Emile van Duren
 * Remarks          : Zoomfacility must be added to DrawMandelbrotSet
 ************************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
void DrawText();                /* contained in writer.c    */
void ClearPixelArea();          /* contained in draw.c      */
void ClearTextHandler();        /* contained in handle.c    */
void FileOpenWarning();         /* contained in messages.c  */
Drawdata *GetWindowData();      /* contained in control.c   */
Textdata *GetTextData();        /* contained in control.c   */
Widget GetWindow();             /* contained in control.c   */


void DrawBifurcationDiagram(w,location)
     Widget w;
     char location;
{
  float a,value;
  int i,x,y,xleft,xright,yupper,ylower;
  Drawdata *data;
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  ClearPixelArea(w,location);
  ClearTextHandler(w,location,1022);
  DrawText(location);
  xleft = (data->width)/10;
  xright = 9*(data->width)/10;
  yupper = (data->height)/4;
  ylower = 3*(data->height)/4;
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  XDrawRectangle(XtDisplay(w),data->pix,data->gc,xleft,yupper,8*xleft,2*yupper);
  XDrawString(XtDisplay(w),data->pix,data->gc,xleft,yupper-70,
           "Attractor A (vertical) of f(x) = ax(1-x) against",48);
  XDrawString(XtDisplay(w),data->pix,data->gc,xleft,yupper-50,
           "control parameter a (horizontal)",32);
  XDrawString(XtDisplay(w),data->pix,data->gc,xleft-10,ylower+15,"0.0",3);
  XDrawString(XtDisplay(w),data->pix,data->gc,xleft-10,yupper-5,"1.0",3);
  XDrawString(XtDisplay(w),data->pix,data->gc,xright-10,ylower+15,"4.0",3);
  for (x=xleft; x<xright; x++)
    {
      value = 0.5;
      a = (x - xleft)/100.0;
      for (i=0; i<1000; i++)
       {
         value *= a*(1-value);
         if (i>750)
           {
             y = ylower - 2*yupper*value;
             XDrawPoint(XtDisplay(w),data->pix,data->gc,x,y);
           }
       }
      XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
             data->width,data->height,0,0);
    }
}
```

```
void DrawMandelbrotSet(w,location)
     Widget w;
     char location;
{
  double c1,c2,r,x,y,xmax,xmin,ymax,ymin,z1,z2;
  int col,row,maxsize = 4,n;
  Drawdata *data;
  Textdata *text;
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  text = GetTextData(location);
  /*
   * initialize first window and set foreground to black
   */
  ClearPixelArea(w,location);
  ClearTextHandler(w,location,894);
  DrawText(location);
  xmin = -2.2;xmax = 1.0; ymin = -1.6; ymax = 1.6;
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  /*
   * if data->unzoom is TRUE then we zoom for the first time, else the
   * amount of iterations must increase to keep constant detail
   */
  if ((maxiterations < 10000) && !(data->unzoom)) maxiterations *= 2;
  if (data->unzoom)
    {
      data->unzoom = FALSE;
      maxiterations = data->niterations;
      /*
       * if data->axes is TRUE do this in case of DrawAxes routine
       */
      if (data->axes)
       {
         col = (int) ((xmin/(xmin - xmax))*(data->width));
         row = (int) ((ymax/(ymax - ymin))*(data->height));
         XDrawString(XtDisplay(w),XtWindow(w),data->gc,20,20,
                    "Attractor basin boundary",24);
         XDrawString(XtDisplay(w),XtWindow(w),data->gc,20,40,
                    "of the complex function:",24);
         XDrawString(XtDisplay(w),XtWindow(w),data->gc,20,60,
                    "f(z) = z*z + c",14);
         /*
          * draw Re(c) axis
          */
         XDrawLine(XtDisplay(w),XtWindow(w),data->gc,0,row,data->width,row);
         XDrawString(XtDisplay(w),XtWindow(w),data->gc,10,row+15,"Re(c)",5);
         /*
          * draw Im(c) axis
          */
         XDrawLine(XtDisplay(w),XtWindow(w),data->gc,col,0,col,data->height);
         XDrawString(XtDisplay(w),XtWindow(w),data->gc,col+10,15,"Im(c)",5);
       }
    }
  for (col = 0; col < data->width; col++)
    {
      c1 = xmin + (xmax - xmin)*col/(data->width);
      for (row = 0; row < data->height; row++)
       {
         c2 = ymax - (ymax - ymin)*row/(data->height);
         x = c1;
         y = c2;
         r = 0;
         /*
          * escape-time algorithm (if orbit doesn't escape to infinity
          * then draw point, i.e. modulus of point remains below 4)
          */
         for (n = 0; n < maxiterations; n++)
           {
             if (r < maxsize)
              {
                z1 = x*x - y*y + c1;
                z2 = 2*x*y + c2;
                r = z1*z1 + z2*z2;
                x = z1;
                y = z2;
              }
            }
          if (r < maxsize)
            {
              /*
               * we choose for (slow) drawing on the screen instead of
               * (fast) drawing in data->pix and then copy to screen,
               * because the user must be able to see that something
               * is happening
               */
              XDrawPoint(XtDisplay(w),XtWindow(w),data->gc,col,row);
            }
        }
    }
  /*
   * copy screen to data->pix for redraw purposes
   */
  XCopyArea(XtDisplay(w),XtWindow(w),data->pix,data->gc,0,0,
          data->width,data->height,0,0);
}


void CalculateMoments(w,location)
     Widget w;
     char location;
{
  /*
   * complex moments !
   */
  char *savefile,*ext = ".mom";
  double sxupper,syupper,xm2sq,ym2sq,xupper,xlower,yupper,ylower;
  double sx2,sy2,sxlower;
  double xmoment[10],ymoment[10],weight,x,y,xtemp,ytemp,xtemp1,ytemp1;
  int i,col,n,row;
  unsigned long amount = 0,pixel;
  Drawdata *data;
  FILE *file;
  XImage *image;
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  n = 5;
  /*
   * initialize image and moment array
   */
  image = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                  data->height,1,XYPixmap);
  xmoment[0] = 1;
  ymoment[0] = 1;
  for (i = 1; i < n; i++)
    {
      xmoment[i] = 0;
      ymoment[i] = 0;
    }
  /*
   * calculate pixel weight
   */
  for (col = 0; col < data->width; col++)
    {
      for (row = 0; row < data->height; row++)
        {
          pixel = XGetPixel(image,col,row);
          if (pixel == 0) amount++;
        }
    }
  if (amount == 0) amount = 1;
  weight = (1.0/((double)amount));
  /*
   * calculate n-th moment sequence
   */
  for (col = 0; col < data->width; col++)
    {
      for (row = 0; row < data->height; row++)
        {
          pixel = XGetPixel(image,col,row);
          if (pixel == 0)
          {
            /*
             * calculate normalized complex pixel coordinates
             */
            x = (double) (col - data->xorigin);
            y = (double) (data->yorigin - row);
            /*
             * calculate first n moments (omit 0)
             */
            xtemp = x;
            ytemp = y;
            for (i = 1; i < n; i++)
             {
               xtemp1 = xtemp;
               ytemp1 = ytemp;
               xmoment[i] += xtemp;
               ymoment[i] += ytemp;
               xtemp = x*xtemp1 - y*ytemp1;
               ytemp = x*ytemp1 + y*xtemp1;
             }
          }
      }
  }
XDestroyImage(image);
image = NULL;
/*
 * correct for relative pixel weights
 */
xmoment[2] *= weight;
ymoment[2] *= weight;
xmoment[4] *= weight;
ymoment[4] *= weight;
/*
 * calculate s*s
 */
xm2sq = xmoment[2]*xmoment[2]-ymoment[2]*ymoment[2];
ym2sq =  2*xmoment[2]*ymoment[2];
xupper = xm2sq - xmoment[4];
yupper = ym2sq - ymoment[4];
xlower = xmoment[4] - 5*xm2sq;
ylower = ymoment[4] - 5*ym2sq;
/*
 * multiply nominator and denominator with complex conjugate
 */
sxupper = xlower*xupper + ylower*yupper;
syupper = xlower*yupper - xupper*ylower;
sxlower = xlower*xlower + ylower*ylower;
if (sxlower != 0)
  {
    sx2 = (float) sxupper/sxlower;
    sy2 = (float) syupper/sxlower;
  }
else
  {
    sx2 = 0;
    sy2 = 0;
  }
/*
 * save image to data->filename.mom
 */
savefile = calloc(strlen(data->filename)+5,sizeof(char));
strcpy(savefile,data->filename);
if ((file = fopen(strcat(savefile,ext),"w")) == NULL)
  {
    FileOpenWarning(w,location);
    data->firsttime = TRUE;
  }
else
  {
    printf("amount,weight = (%d,%f)\n",amount,weight);
    printf("(x-origin,y-origin) =  (%d,%d)\n",data->xorigin,data->yorigin);
    printf(file,"(x-origin,y-origin) =  (%d,%d)\n",data->xorigin,data->yorigin);
    fprintf(file,"file         : %s%s\n",data->filename,ext);
    fprintf(file,"amount       = %d\n",amount);
    fprintf(file,"weight       = %f\n",weight);
```

```
      for (i = 0; i < n; i++)
       {
        printf("xmoment[%d] = %16.9e\n",i,xmoment[i]);
        fprintf(file,"xmoment[%d] = %16.9e\n",i,xmoment[i]);
        printf("ymoment[%d] = %16.9e\n",i,ymoment[i]);
        fprintf(file,"ymoment[%d] = %16.9e\n",i,ymoment[i]);
       }
      printf("s*s       = %f + %fi\n",sx2,sy2);
      fprintf(file,"s*s       = %f + %fi\n",sx2,sy2);
      printf("%s\n","");
      fclose(file);
      free(savefile);
      savefile = NULL;
    }
}


void CalculateFractalDimension(w,location)
     Widget w;
     char location;
{
  int i,col,row,N1=0,N2=0;
  float fdim;
  unsigned long amount = 0,pixel1,pixel2,pixel3,pixel4;
  Drawdata *data;
  XImage *image;
  w = GetWindow(location,'p');
  data = GetWindowData(location,'p');
  /*
   * initialize image
   */
  image = XGetImage(XtDisplay(w),data->pix,0,0,data->width,
                   data->height,1,XYPixmap);
  /*
   * calculate fractal dimension as slope of log-log plot, thus:
   *
   * fractal dimension = log(N1/N2)/log2
   *
   * where N1 is the amount of boxes of size 1/500 which contain at least 1
   * pixel and where N2 is the amount of boxes of size 1/250 which contain
   * at least 1 pixel (notice that N1 is just equal to the amount of pixels
   * the image contains, since our pixmap size is 500 pixels)
   */
  for (col = 0; col < data->width; col += 2)
    {
     for (row = 0; row < data->height; row += 2)
      {
       pixel1 = XGetPixel(image,col,row);
       if (pixel1 == 0) N1++;
       pixel2 = XGetPixel(image,col+1,row);
       if (pixel2 == 0) N1++;
       pixel3 = XGetPixel(image,col,row+1);
       if (pixel3 == 0) N1++;
       pixel4 = XGetPixel(image,col+1,row+1);
       if (pixel4 == 0) N1++;
       if (pixel1 == 0 || pixel2 == 0 || pixel3 == 0 || pixel4 == 0) N2++;
      }
    }
  if (N2 != 0)
    {
     fdim = (float) (log((double) N1/N2)/log(2.0));
    }
  else
    {
     fdim = 0;
    }
  if (location == 'l') printf("fractal dimension of left image  = %f\n",fdim);
  if (location == 'r') printf("fractal dimension of right image = %f\n",fdim);
  XDestroyImage(image);
  image = NULL;
}
```

```
/*****************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : strings.c
 * Purpose         : Contains string conversion operations
 * Part of         : IFS-CODEC
 * Created on      : June 02, 1992
 * Created by      : Emile van Duren
 * Last modified on: November 20, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 *****************************************************************************/

#include "all.h"

XmString StringArrayToXmString(cs,n)
     char *cs[];
     int  n;
{
  int      i;
  XmString xmstr;
  /*
   * if the array is empty just return an empty string
   */
  if (n<=0) return (XmStringCreate("",XmSTRING_DEFAULT_CHARSET));
  xmstr = (XmString) NULL;
  for (i=0;i<n;i++)
    {
      if (i>0) xmstr = XmStringConcat(xmstr,XmStringSeparatorCreate());
      xmstr=XmStringConcat(xmstr,XmStringCreate(cs[i],XmSTRING_DEFAULT_CHARSET));
    }
  return (xmstr);
}

char *GetStringFromXmString(string)
     XmString string;
{
  char               *text,*buf = NULL;
  Boolean            done = FALSE,separator;
  XtPointer          context ;
  XmStringCharSet    charset;
  XmStringDirection  dir;
  /*
   * initialize context
   */
  XmStringInitContext(&context,string);
  while (!done)
    {
      if (XmStringGetNextSegment(context,&text,&charset,&dir,&separator))
       {
         if (separator) done = TRUE;
         if (buf)
           {
             buf = XtRealloc(buf,strlen(buf)+strlen(text)+2);
             strcat(buf,text);
           }
         else
           {
             buf = (char *) XtMalloc(strlen(text)+1);
             strcpy(buf,text);
           }
         XtFree(text);
       }
      else
       {
         done = TRUE;
       }
    }
  XmStringFreeContext(context);
  return(buf);
}
```

```
/***************************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : tracker.c
 * Purpose          : Controls conditional tracking of mousepointer
 * Part of          : IFS-CODEC
 * Created on       : June 17, 1992
 * Created by       : Emile van Duren
 * Last modified on: November 23, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 ***************************************************************************************/

#include "all.h"


void ShowMousePosition(w,data,event)
     Widget   w;
     Drawdata *data;
     XEvent   *event;
{
  /*
   * extract sprite position from event and draw point at that position,
   * both in drawingarea and its pixmap
   *
   * (must be altered to if-condition on mode zoom, draw or origin)
   */
  if (data->drawgrab)
    {
     XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
     XDrawPoint(XtDisplay(w),data->pix,data->gc,
             event->xbutton.x,event->xbutton.y);
     XDrawPoint(XtDisplay(w),XtWindow(w),data->gc,
             event->xbutton.x,event->xbutton.y);
     data->oldpointx = event->xbutton.x;
     data->oldpointy = event->xbutton.y;
    }
}


void TrackMousePosition(w,data,event)
     Widget   w;
     Drawdata *data;
     XEvent   *event;
{
  /*
   * extract sprite position from event and draw point at that position,
   * both in drawingarea and its pixmap
   *
   * (must be altered to if-condition on mode zoom, draw, erase, origin)
   */
  if (data->drawgrab)
    {
     XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
     XDrawPoint(XtDisplay(w),data->pix,data->gc,
             event->xmotion.x,event->xmotion.y);
     XDrawPoint(XtDisplay(w),XtWindow(w),data->gc,
             event->xmotion.x,event->xmotion.y);
     XDrawLine(XtDisplay(w),data->pix,data->gc,data->oldpointx,
            data->oldpointy,event->xmotion.x,event->xmotion.y);
     XDrawLine(XtDisplay(w),XtWindow(w),data->gc,data->oldpointx,
            data->oldpointy,event->xmotion.x,event->xmotion.y);
     data->oldpointx = event->xmotion.x;
     data->oldpointy = event->xmotion.y;
    }
  if (data->erasegrab)
    {
     XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
     XFillRectangle(XtDisplay(w),data->pix,data->gc,
                event->xmotion.x,event->xmotion.y,10,10);
     XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,
                 event->xmotion.x,event->xmotion.y,10,10);
    }
}


void LeaveWindowHandler(w,data,event)
     Widget   w;
     Drawdata *data;
     XEvent   *event;
{
  /*
   * remove the dynamically installed eventhandlers
   */
  if (!data->drawgrab && !data->erasegrab)
    {
      XtRemoveEventHandler(w,ButtonPressMask,FALSE,ShowMousePosition,data);
      XtRemoveEventHandler(w,ButtonMotionMask,FALSE,TrackMousePosition,data);
      XtRemoveEventHandler(w,ButtonReleaseMask | LeaveWindowMask,FALSE,
                           LeaveWindowHandler,data);
    }
}


void EnterWindowHandler(w,data,event)
     Widget   w;
     Drawdata *data;
     XEvent   *event;
{
  /*
   * if grab is TRUE and the sprite enters the window, then install the
   * additional eventhandlers
   */
  if (data->drawgrab | data->erasegrab)
    {
      XtAddEventHandler(w,ButtonPressMask,FALSE,ShowMousePosition,data);
      XtAddEventHandler(w,ButtonMotionMask,FALSE,TrackMousePosition,data);
      XtAddEventHandler(w,ButtonReleaseMask | LeaveWindowMask,FALSE,
                        LeaveWindowHandler,data);
    }
}


void InitMouseTracker(target)
     Widget   target;
{
  /*
   * install initial eventhandlers
   */
  if (target == leftpixelarea)
    {
      XtAddEventHandler(target,EnterWindowMask,FALSE,
                        EnterWindowHandler,leftpixeldata);
    }
  if (target == rightpixelarea)
    {
      XtAddEventHandler(target,EnterWindowMask,FALSE,
                        EnterWindowHandler,rightpixeldata);
    }
}
```

```
/*****************************************************************************
 * Copyright(c)     : Eindhoven University of Technology (TUE,NL)
 *                  : Faculty of Electrical Engineering (E)
 *                  : Design Automation Group (ES)
 * Mail address     : emilevd@viper.es.ele.tue.nl
 * Project          : Fractal imagecompression with Iterated Function Systems
 * Supervisor       : Prof. Dr. Ing. J.A.G. Jess
 * File             : writer.c
 * Purpose          : Contains texthandlers for writing in textareas
 * Part of          : IFS-CODEC
 * Created on       : June 18, 1992
 * Created by       : Emile van Duren
 * Last modified on: November 20, 1992
 * Modified by      : Emile van Duren
 * Remarks          : None
 *****************************************************************************/

#include "all.h"

/*
 * forward function declarations
 */
int StringLength();             /* contained in strings.c     */
Drawdata *GetWindowData();      /* contained in control.c     */
Textdata *GetTextData();        /* contained in control.c     */
Widget GetWindow();             /* contained in control.c     */


void DrawText(location)
     char location;
{
  Drawdata *data;
  Textdata *text;
  Widget w;
  w = GetWindow(location,'t');
  data = GetWindowData(location,'t');
  text = GetTextData(location);
  /*
   * clear textarea
   */
  XSetForeground(XtDisplay(w),data->gc,WhitePixelOfScreen(XtScreen(w)));
  XFillRectangle(XtDisplay(w),data->pix,data->gc,0,0,data->width,data->height);
  XFillRectangle(XtDisplay(w),XtWindow(w),data->gc,0,0,
             data->width,data->height);
  /*
   * draw in textarea
   */
  XSetForeground(XtDisplay(w),data->gc,BlackPixelOfScreen(XtScreen(w)));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,20,text->mode,
           strlen(text->mode));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,40,text->srce,
           strlen(text->srce));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,60,text->size,
           strlen(text->size));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,80,text->cont,
           strlen(text->cont));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,100,text->save,
           strlen(text->save));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,120,text->orgn,
           strlen(text->orgn));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,140,text->init,
           strlen(text->init));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,160,text->amit,
           strlen(text->amit));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,180,text->iter,
           strlen(text->iter));
  XDrawString(XtDisplay(w),data->pix,data->gc,10,200,text->zoom,
           strlen(text->zoom));
  XCopyArea(XtDisplay(w),data->pix,XtWindow(w),data->gc,0,0,
         data->width,data->height,0,0);
}
```

Appendix C

Some IFS-codes and their attractors

```
/*****************************************************************************
 * Copyright(c)    : Eindhoven University of Technology (TUE,NL)
 *                 : Faculty of Electrical Engineering (E)
 *                 : Design Automation Group (ES)
 * Mail address    : emilevd@viper.es.ele.tue.nl
 * Project         : Fractal imagecompression with Iterated Function Systems
 * Supervisor      : Prof. Dr. Ing. J.A.G. Jess
 * File            : codes
 * Purpose         : Contains IFS-codes for various fractal structures
 * Part of         : IFS-CODEC
 * Created on      : August 06, 1992
 * Created by      : Emile van Duren
 * Last modified on: December 17, 1992
 * Modified by     : Emile van Duren
 * Remarks         : None
 *****************************************************************************/

chain.ifs [origin (-250,-250) startpixel (-78,-183)]

 0.132  0.000  0.000  0.420   64.000  115.000 0.127
 0.405  0.000  0.000  0.109  143.000   54.000 0.101
 0.411  0.000  0.000  0.115  136.000  375.000 0.109
 0.140  0.000  0.000  0.457  356.000  125.000 0.147
 0.000  0.000  0.000  0.002  101.000  277.000 0.001
 0.125 -0.211  0.104  0.248  138.000   18.000 0.122
 0.121  0.230 -0.113  0.246  289.000   72.000 0.128
 0.123  0.192 -0.104  0.230   41.000  325.000 0.111
 0.146 -0.192  0.115  0.303  365.000  251.000 0.153

dragon.ifs [origin (0,0) startpixel (0,0)]

 0.557 -0.366  0.366  0.557 -100.000    0.000 0.500
 0.557 -0.366  0.366  0.557  100.000    0.000 0.500

fern.ifs [origin (0,-200) startpixel (0,0)]

 0.000  0.000  0.000  0.160    0.000    0.000 0.001
 0.850  0.040 -0.040  0.850    0.000   64.000 0.772
 0.200 -0.260  0.230  0.220    0.000   64.000 0.111
-0.150  0.280  0.260  0.240    0.000   18.000 0.116

france.ifs [origin (0,100) startpixel (0,0)]

 0.000 -0.033  0.033  0.000 -204.000   14.000 0.001
 0.036  0.000  0.000  0.036 -199.000   -8.000 0.001
 0.086 -0.043  0.043  0.086 -189.000   10.000 0.005
-0.045 -0.122  0.122 -0.045 -179.000   -6.000 0.009
-0.174 -0.037  0.037 -0.174 -136.000  -40.000 0.016
 0.098 -0.106  0.106  0.098 -178.000    4.000 0.011
 0.084  0.209 -0.209  0.084  -83.000  -48.000 0.026
-0.130  0.002 -0.002 -0.130  -95.000    0.000 0.009
 0.063 -0.024  0.024  0.063 -110.000   46.000 0.002
-0.193 -0.456  0.456 -0.193  -87.000 -166.000 0.124
-0.235  0.046 -0.046 -0.235  -86.000 -242.000 0.029
 0.097  0.081 -0.081  0.097 -118.000 -212.000 0.008
-0.660 -0.253  0.253 -0.660  -10.000 -142.000 0.252
-0.161 -0.192  0.192 -0.161  -34.000 -242.000 0.032
```

```
-0.250  0.013 -0.013 -0.250  -45.000 -248.000 0.032
 0.125  0.000  0.000  0.125  -23.000 -248.000 0.008
-0.310  0.172 -0.172 -0.310   62.000 -212.000 0.064
 0.250  0.004 -0.004  0.250   90.000 -182.000 0.032
-0.254  0.068 -0.068 -0.254   98.000 -234.000 0.035
 0.142 -0.099  0.099  0.142   99.000 -208.000 0.015
-0.036 -0.187  0.187 -0.036   83.000 -150.000 0.019
 0.000  0.354 -0.354  0.000   98.000  -72.000 0.063
-0.045 -0.256  0.256 -0.045   84.000  -40.000 0.034
 0.091  0.011 -0.011  0.091  150.000   -2.000 0.004
 0.113  0.053 -0.053  0.113  115.000   14.000 0.008
-0.219 -0.121  0.121 -0.219   -9.000   20.000 0.032
-0.219 -0.121  0.121 -0.219    0.000   20.000 0.032
-0.024  0.123 -0.123 -0.024   71.000   28.000 0.008
-0.106  0.021 -0.021 -0.106    5.000   68.000 0.006
-0.354 -0.006  0.006 -0.354   -9.000  -42.000 0.063
-0.220  0.008 -0.008 -0.220  -54.000  -24.000 0.024
```

koch.ifs [origin (0,0) startpixel (0,110)]

```
 0.330  0.000  0.000  0.330 -134.000    0.000 0.250
 0.330  0.000  0.000  0.330  134.000    0.000 0.250
 0.165 -0.286  0.286  0.165  -34.000   56.000 0.250
 0.165  0.286 -0.286  0.165   34.000   56.000 0.250
```

leaf.ifs [origin (-250,150) startpixel (0,0)]

```
 0.400 -0.380  0.250  0.440  -25.000 -209.000 0.189
 0.590  0.000  0.001  0.600  126.000   16.000 0.247
 0.400  0.350 -0.260  0.420  309.000  -73.000 0.181
 0.740 -0.003  0.002  0.740   67.000  -80.000 0.383
```

rect.ifs [origin (0,0) startpixel (0,0)]

```
 0.500  0.000  0.000  0.500  -50.000   50.000 0.250
 0.500  0.000  0.000  0.500   50.000   50.000 0.250
 0.500  0.000  0.000  0.500  -50.000  -50.000 0.250
 0.500  0.000  0.000  0.500   50.000  -50.000 0.250
```

square.ifs [origin (0,0) startpixel (0,35)]

```
 0.330  0.000  0.000  0.330  -66.000    0.000 0.125
 0.330  0.000  0.000  0.330    0.000   66.000 0.125
 0.330  0.000  0.000  0.330  -66.000   66.000 0.125
 0.330  0.000  0.000  0.330  -66.000  -66.000 0.125
 0.330  0.000  0.000  0.330   66.000   66.000 0.125
 0.330  0.000  0.000  0.330    0.000  -66.000 0.125
 0.330  0.000  0.000  0.330   66.000  -66.000 0.125
 0.330  0.000  0.000  0.330   66.000    0.000 0.125
```

tree.ifs [origin (-225,-200) startpixel (0,0)]

```
 0.195 -0.488  0.344  0.433  199.395  110.340 0.276
 0.462  0.414 -0.252  0.361  112.995  256.005 0.297
-0.058 -0.070  0.453 -0.111  268.920   43.605 0.042
-0.035  0.070 -0.469 -0.022  219.780  228.105 0.037
-0.637  0.000  0.000  0.501  385.290  113.085 0.348
```

triangle.ifs [origin (0,0) startpixel (0,0)]

```
 0.500  0.000  0.000  0.500  -50.000  -50.000 0.334
 0.500  0.000  0.000  0.500   50.000  -50.000 0.333
 0.500  0.000  0.000  0.500    0.000   50.000 0.333
```

twig.ifs [origin (-225,-225) startpixel (0,0)]

```
 0.387  0.430  0.430 -0.387  128.000  261.000 0.694
 0.441 -0.091 -0.009 -0.322  210.950  252.950 0.296
-0.461  0.020 -0.113  0.015  200.000  200.000 0.010
```

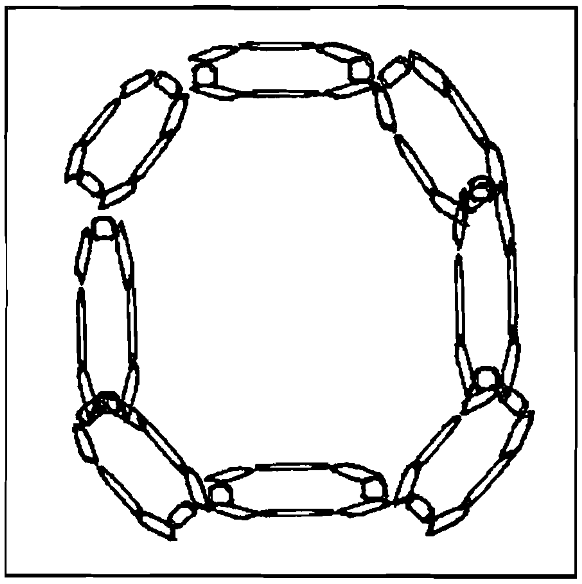


Figure C.1. Attractor of structure *chain.*

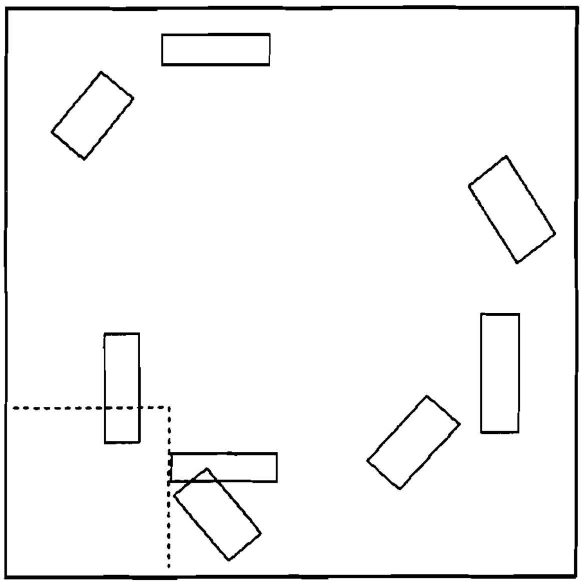


Figure C.2. Graphical view of IFS for structure *chain.*

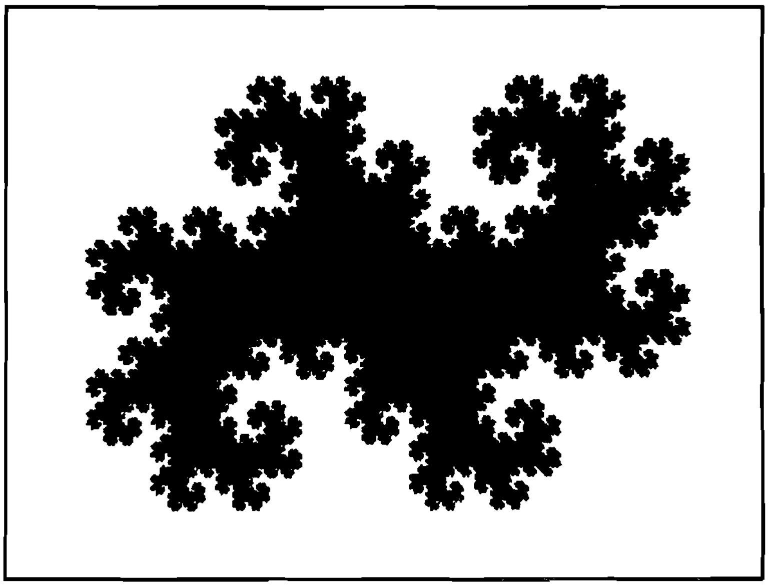


Figure C.3. Attractor of structure *dragon.*

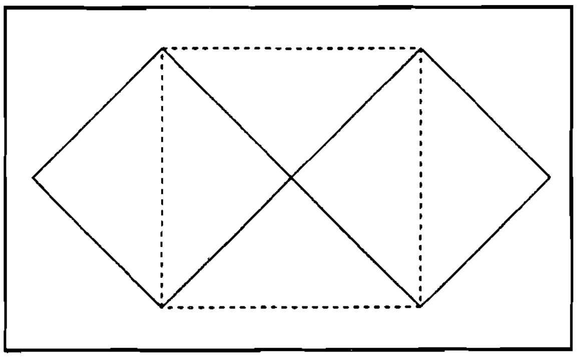


Figure C.4. Graphical view of IFS for structure *dragon.*

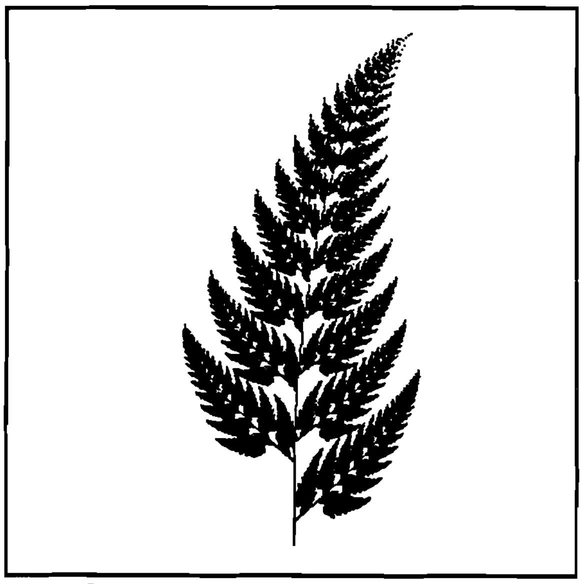


Figure C.5. Attractor of structure *fern*.

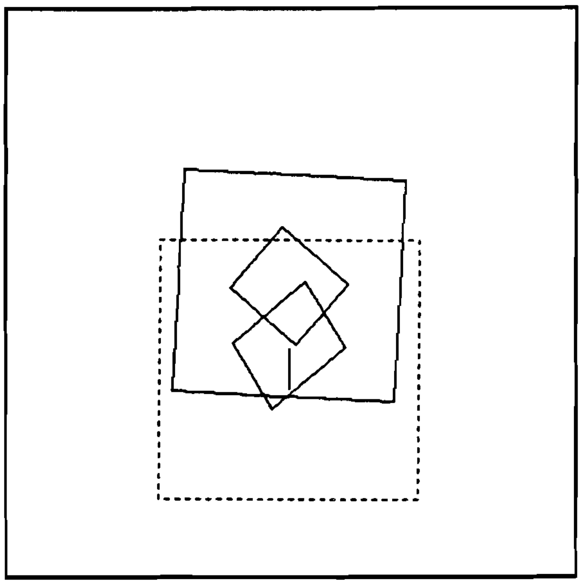


Figure C.6. Graphical view of IFS for structure *fern*.

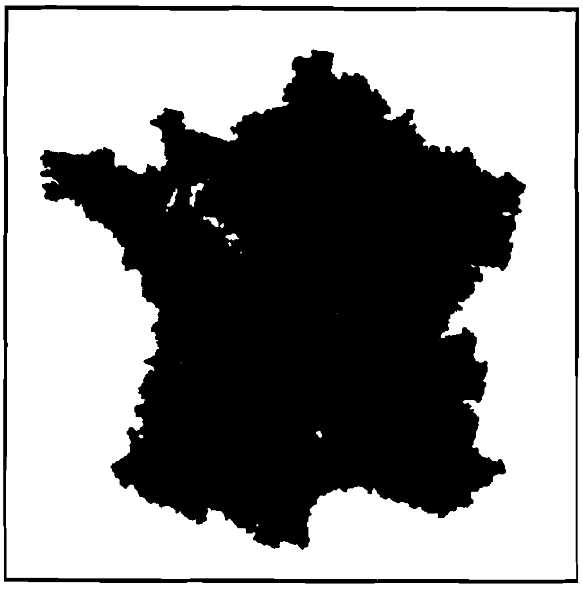


Figure C.7. Attractor of structure *france*.

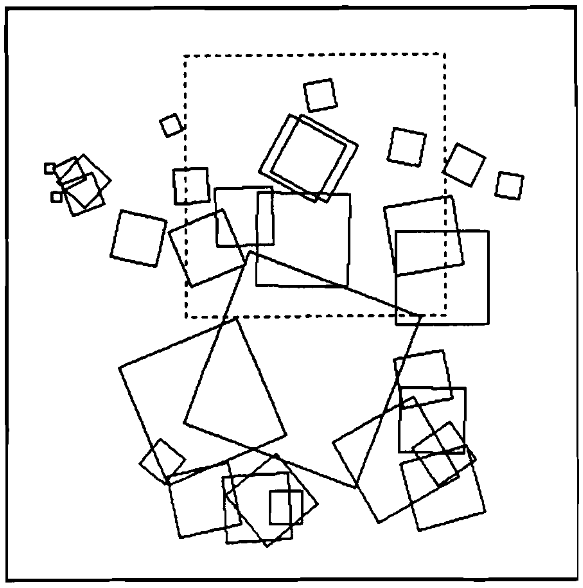


Figure C.8. Graphical view of IFS for structure *france*.

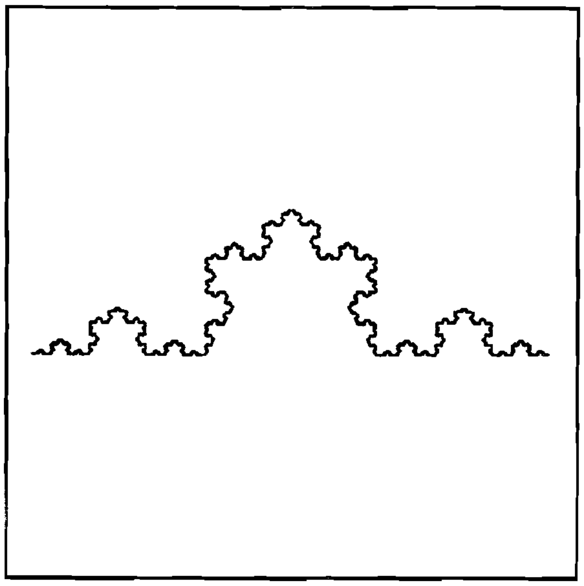


Figure C.9. Attractor of structure *koch*.

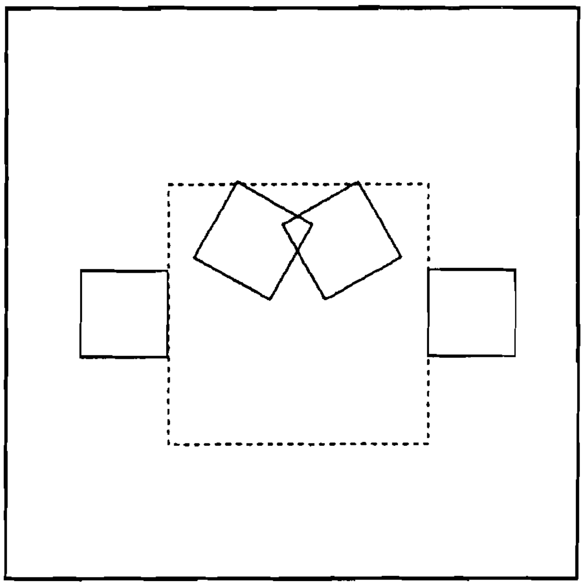


Figure C.10. Graphical view of IFS for structure *koch*.

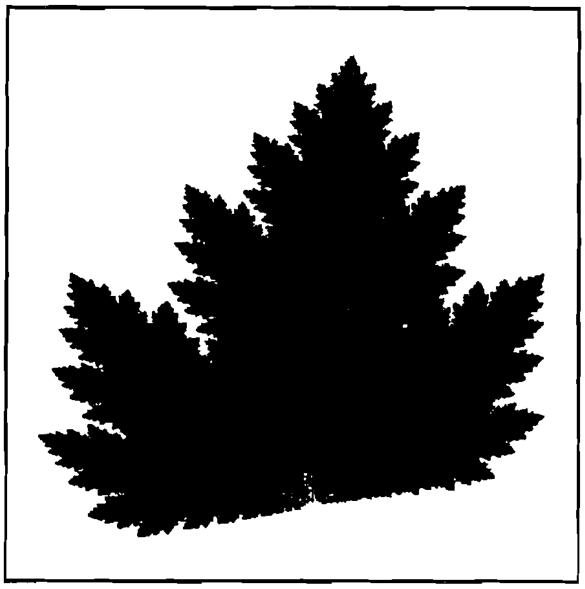


Figure C.11. Attractor of structure *leaf*.

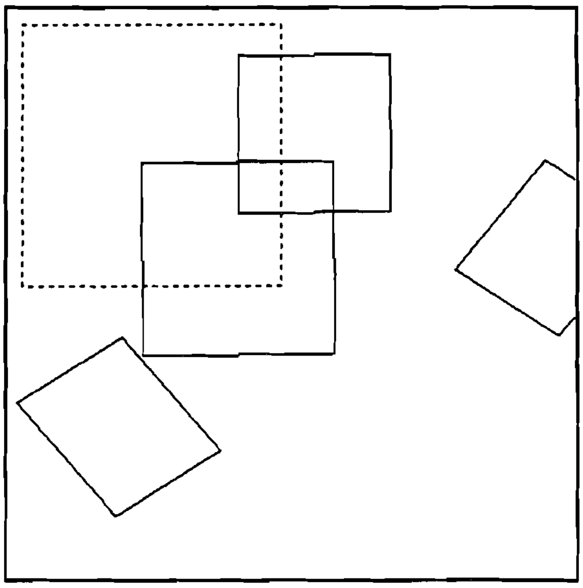


Figure C.12. Graphical view of IFS for structure *leaf*.

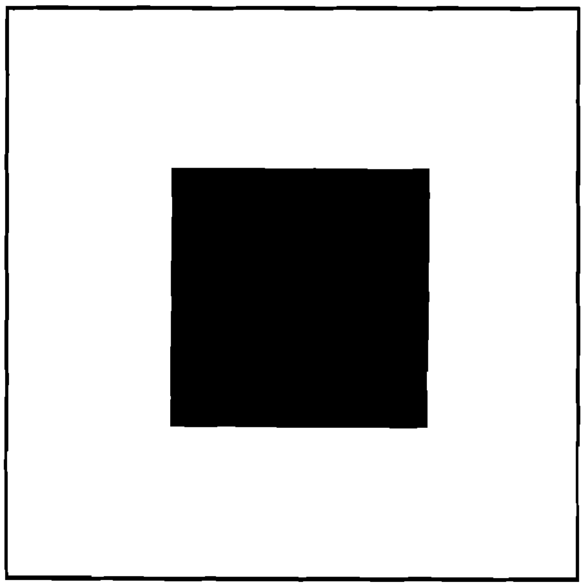


Figure C.13. Attractor of structure *rect*.

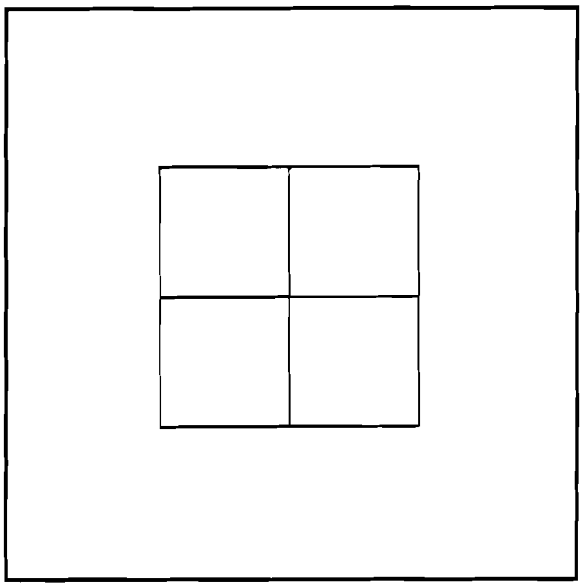


Figure C.14. Graphical view of IFS for structure *rect*.

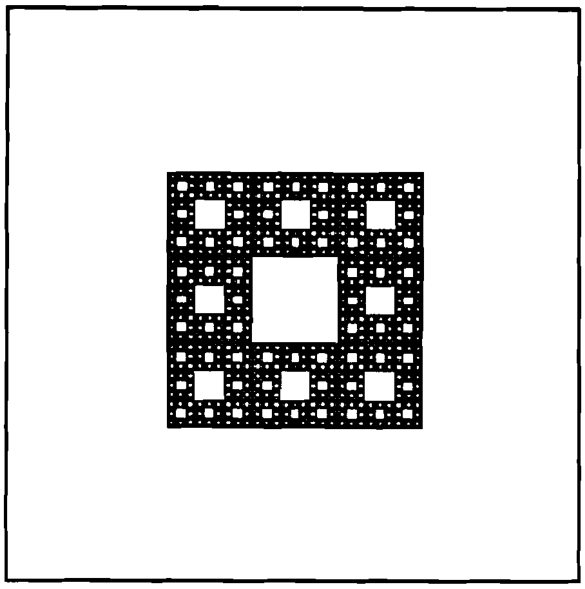


Figure C.15. Attractor of structure *square*.

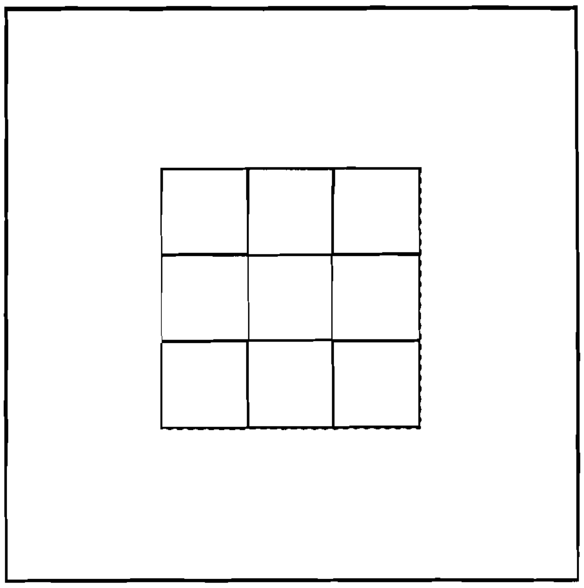


Figure C.16. Graphical view of IFS for structure *square*.

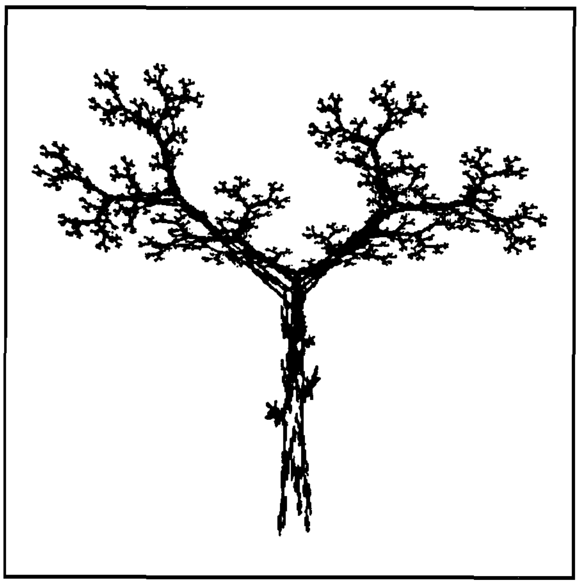


Figure C.17. Attractor of structure *tree*.

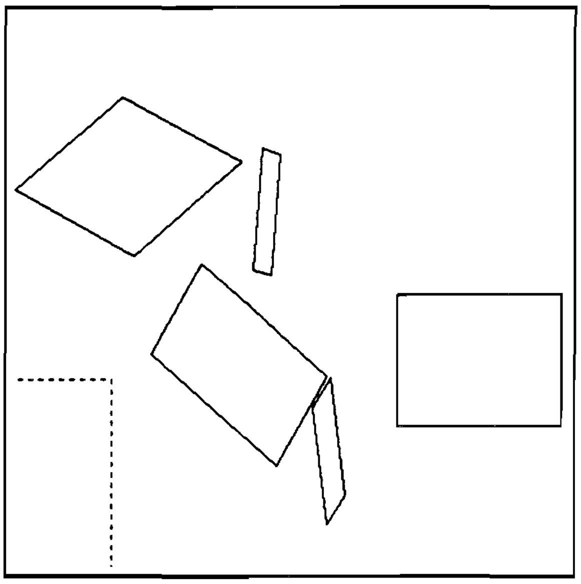


Figure C.18. Graphical view of IFS for structure *tree*.

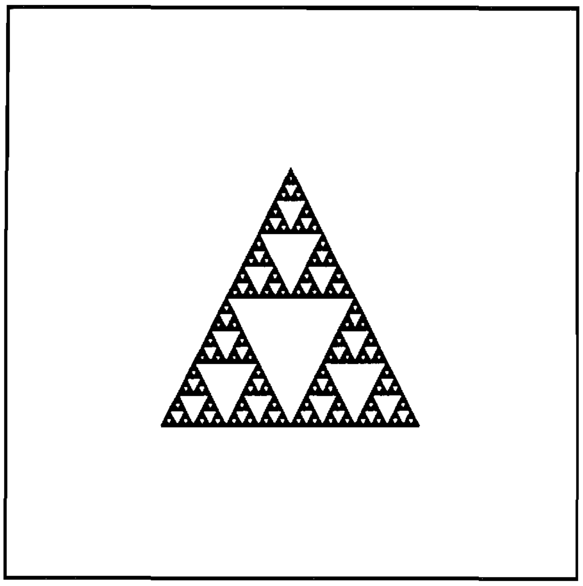


Figure C.19. Attractor of structure *triangle*.

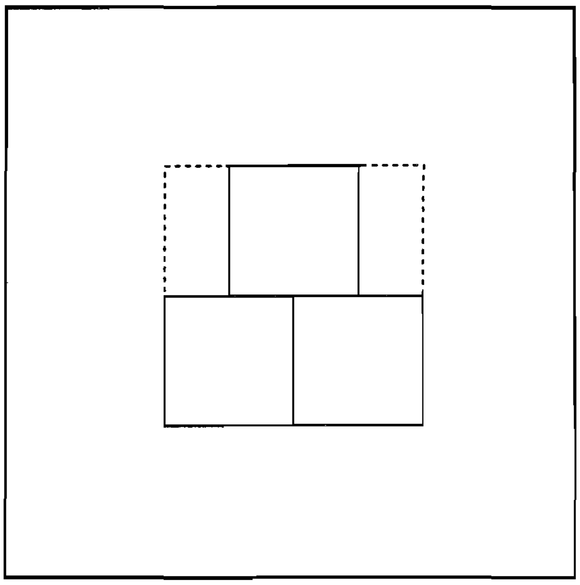


Figure C.20. Graphical view of IFS for structure *triangle*.

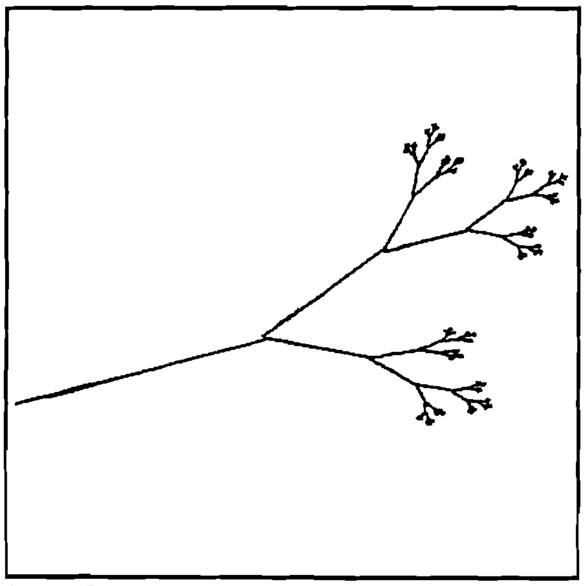


Figure C.21. Attractor of structure *twig*.

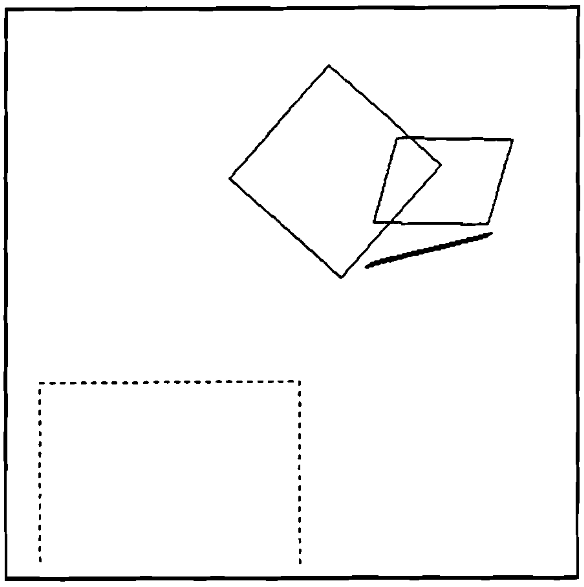


Figure C.22. Graphical view of IFS for structure *twig*.

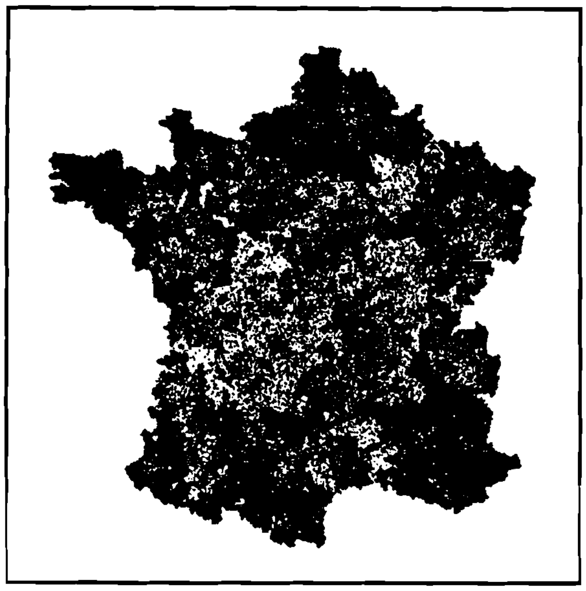


Figure C.23. $france_f$, nondeterministically rendered.

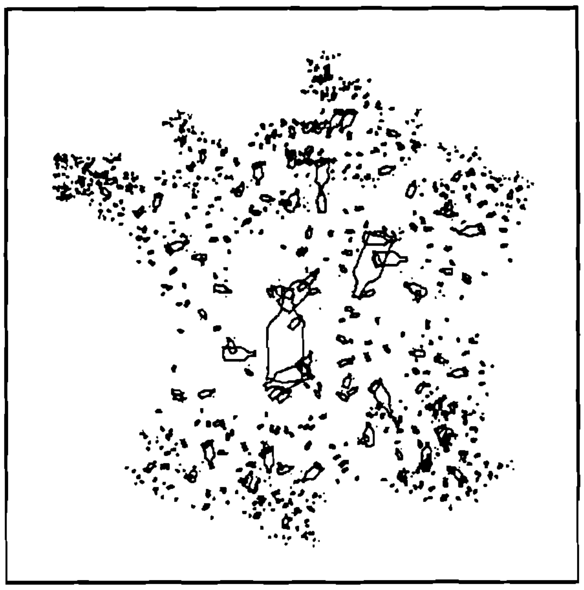


Figure C.24. $france_f$ with a bottle as condensation set.

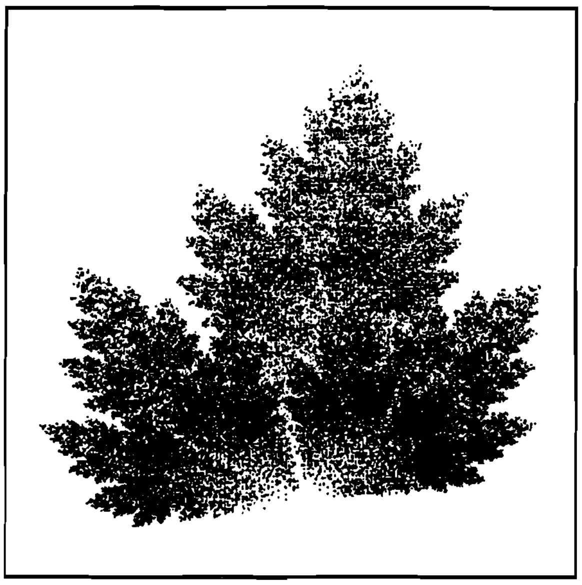


Figure C.25. *leaf$_f$* for $P = \{0.289, 0.147, 0.381, 0.183\}$.

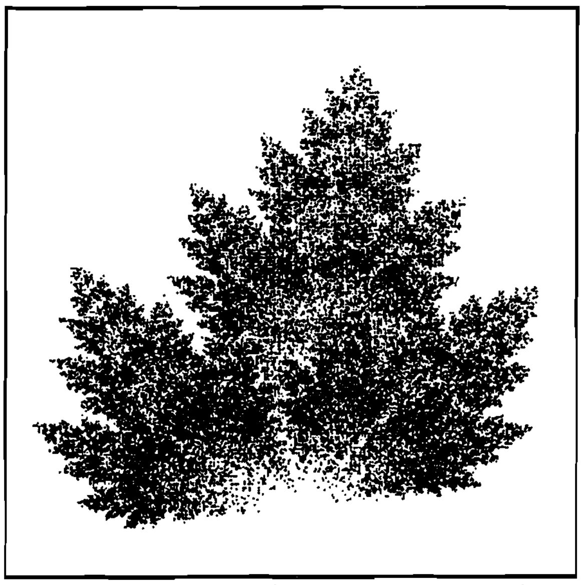


Figure C.26. *leaf$_f$* for $P = \{0.289, 0.247, 0.281, 0.183\}$.

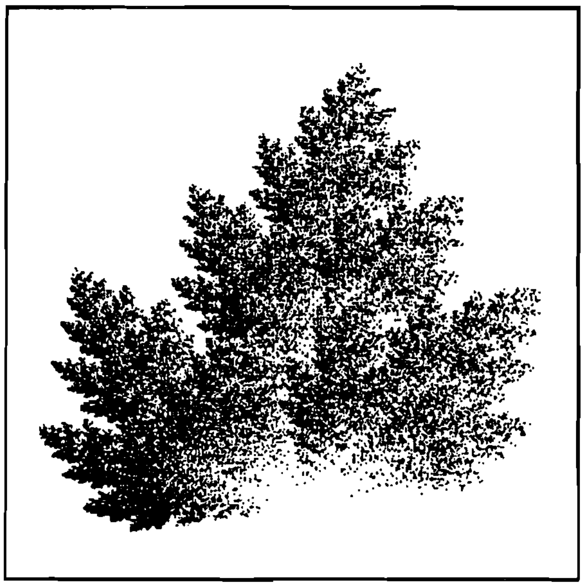


Figure C.27. *leaf$_f$* for $P = \{0.5, 0.25, 0.125, 0.125\}$.

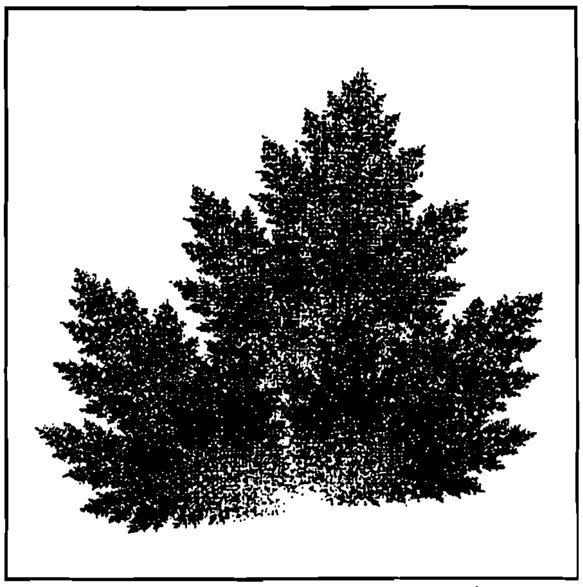


Figure C.28. *leaf$_f$* for $P = \{0.25, 0.25, 0.25, 0.25\}$.

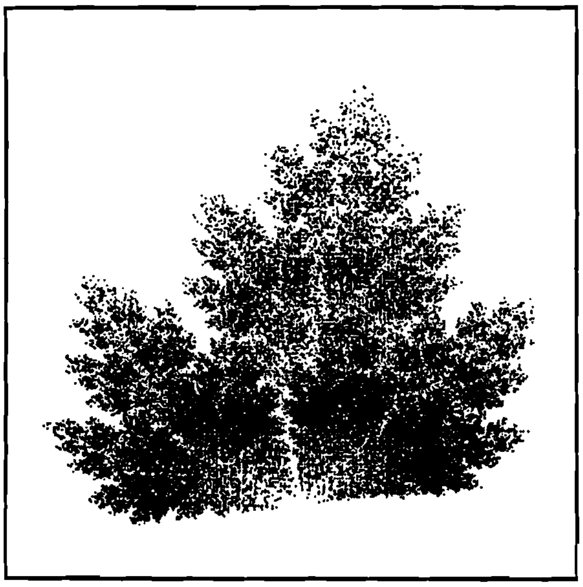


Figure C.29. *leaf$_f$* for $P = \{0.3,0.1,0.4,0.2\}$.

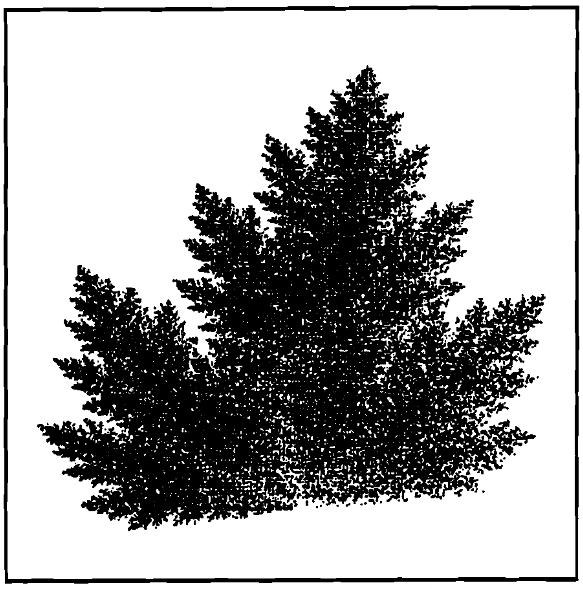


Figure C.30. *leaf$_f$* for $P = \{0.2,0.3,0.1,0.4\}$.

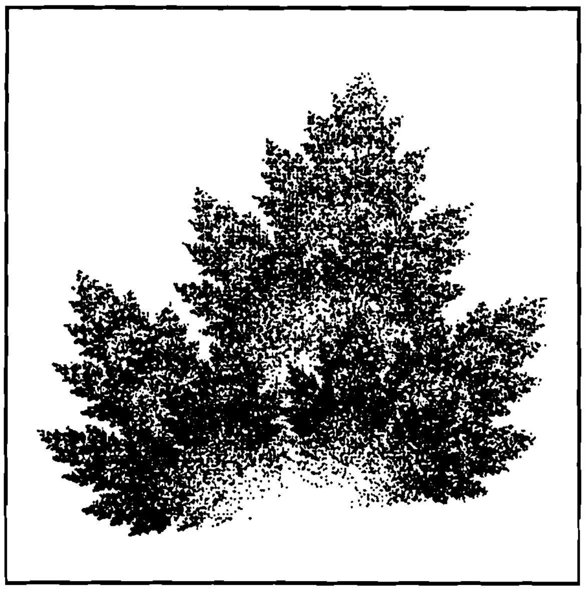


Figure C.31. *leaf$_f$* for $P = \{0.4,0.2,0.3,0.1\}$.

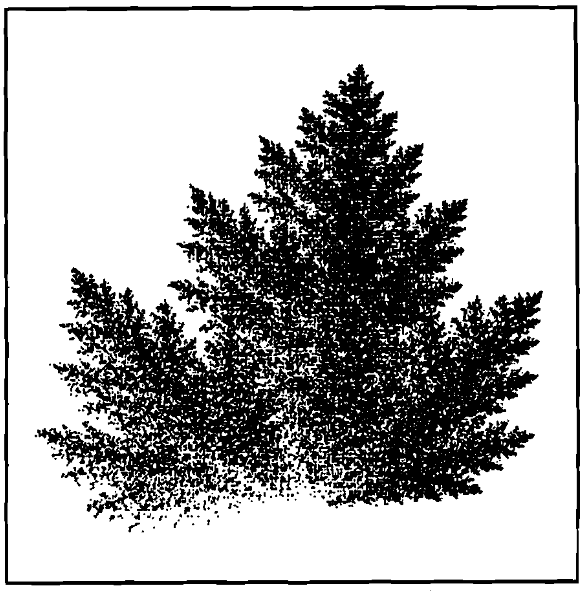


Figure C.32. *leaf$_f$* for $P = \{0.1,0.4,0.2,0.3\}$.

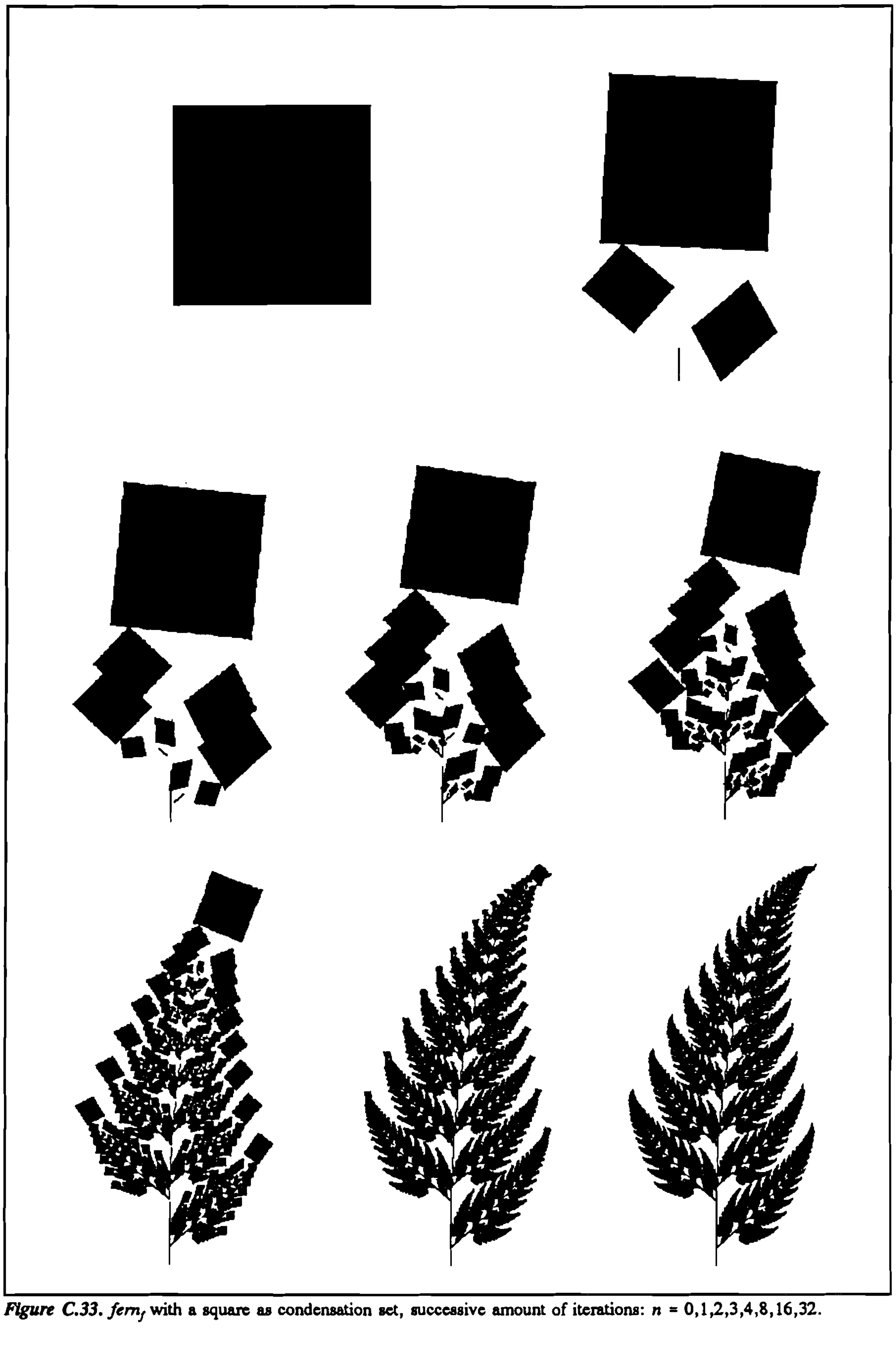


Figure C.33. $fern_f$ with a square as condensation set, successive amount of iterations: $n = 0,1,2,3,4,8,16,32$.

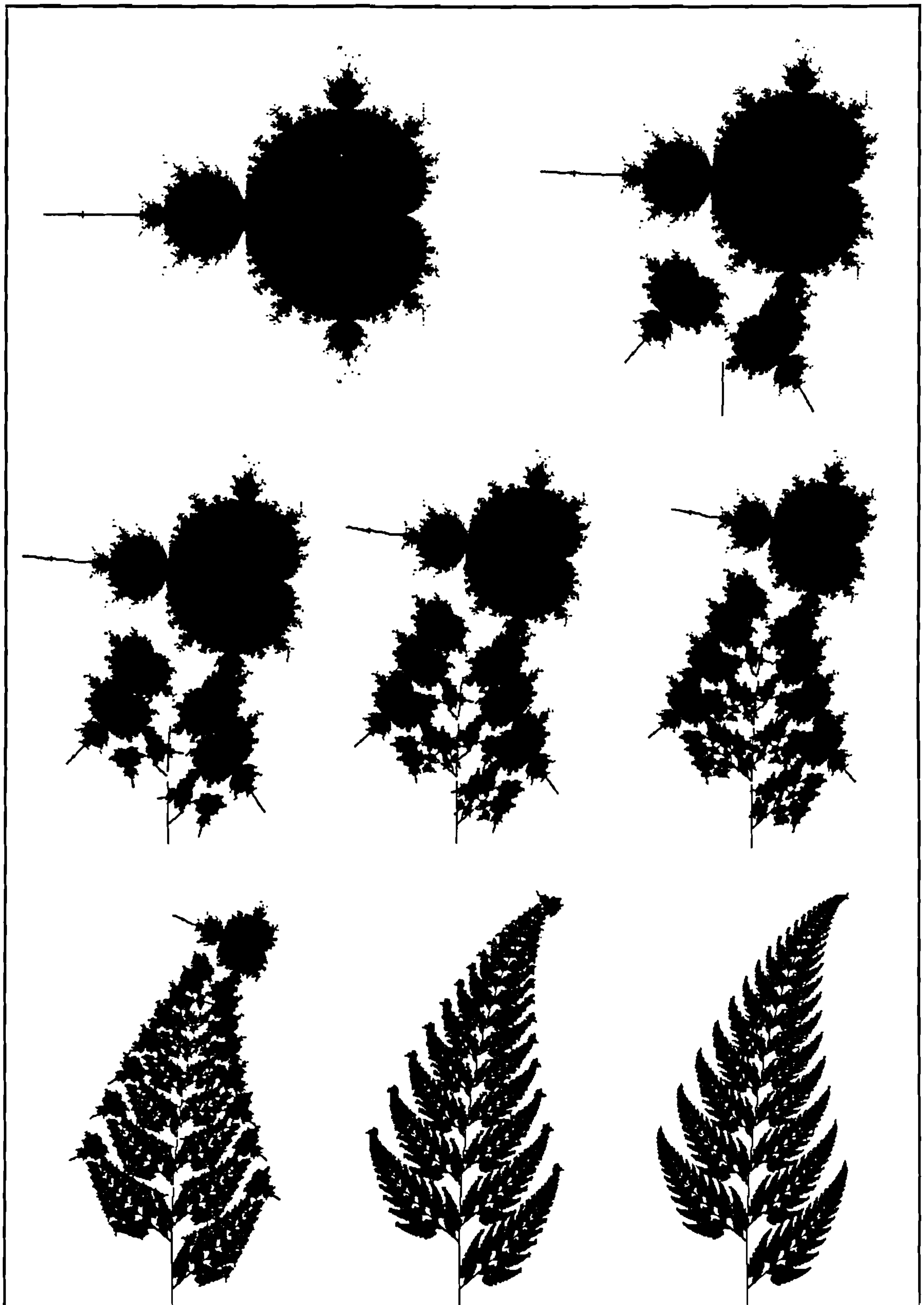
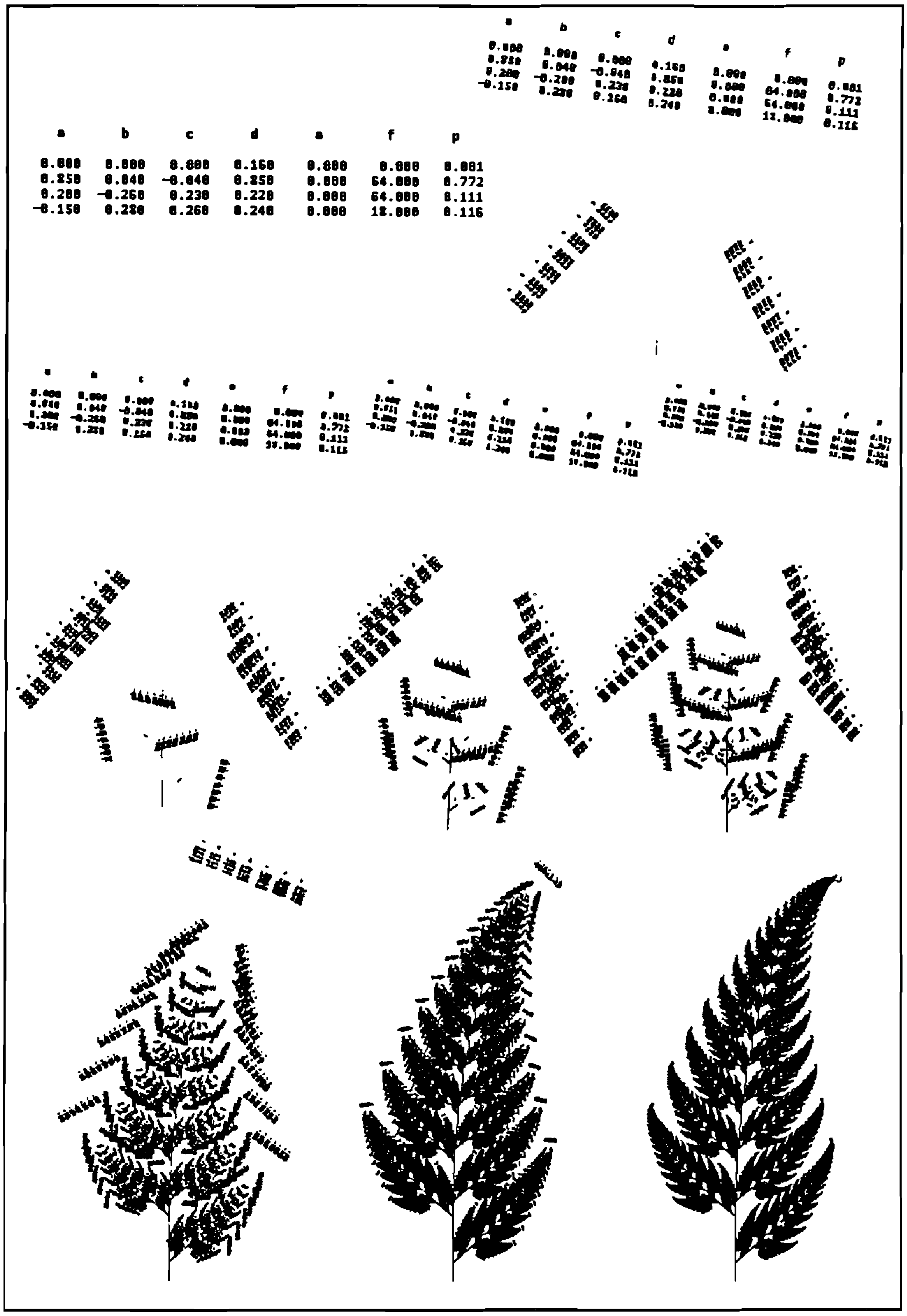


Figure C.34. *fern*$_f$ with the Mandelbrot set as condensation set, successive amount of iterations: n = 0,1,2,3,4,8,16,32.



a	b	c	d	e	f	p
0.000	0.000	0.000	0.160	0.000	0.000	0.001
0.850	0.040	-0.040	0.850	0.000	64.000	0.772
0.200	-0.260	0.230	0.220	0.000	64.000	0.111
-0.150	0.280	0.260	0.240	0.000	18.000	0.116

Figure C.35. $fern_f$ with its own IFS-code as condensation set, successive amount of iterations: n = 0,1,2,3,4,8,16,32.

Appendix D

Typical genetic monitor output

```
file name                              : france
searching to find                      : identity
operator probabilities: MacroCrossover: 0.20
                        MicroCrossover: 0.20
                        MixedCrossover: 0.20
                        Mutate        : 0.10
                        Jump          : 0.10
                        Scale         : 0.10
                        Rotate        : 0.10
maximum encoding error (epsilon)       : 0 [percent]
maximum amount of generations          : 25
population size                        : 10

transformation: 1
generation    : 0 (initialization)
member     a   b   c   d   e   f     fitness
  1      109 136 143  68 128 128    0.445742
  2       73  74  19 114 153   8    0.352467
  3      201 149  98 184 179  40    0.262182
  4       21  99  70  93 155   9    0.236376
  5       55 110  50  24  15 149    0.211504
  6       67  73 131 242  43 241    0.113952
  7      204  84 204 107  18 133    0.093416
  8      232 249  28  47  25 170    0.070576
  9      130 167 149 108  19   3    0.068520
 10      210 217  22   7 189 166    0.051151
total fitness = 1.905887

generation : 1
member     a   b   c   d   e   f     fitness
  1      109 136 167 124 178 128    0.507279
  2      109 136 143  68 128 128    0.445742
  3      109 136 143  68  32 128    0.362960
  4       73  74  19 114 153   8    0.352467
  5      113  64  35  58 155   9    0.197892
  6      249 129  61  46 189 166    0.184375
  7      138 167 149 108  19   3    0.121503
  8       67  73 131 242 139 241    0.114945
  9       53  99  70  93 155   9    0.095200
 10      201 149  74 128 129  40    0.088855
total fitness = 2.471216

generation : 2
member     a   b   c   d   e   f     fitness
  1      109 136 167 124 178 128    0.507279
  2      107 134 165 124 178 130    0.504466
  3      109 136 143  68 128 128    0.445742
  4      109 136 167  92 178 128    0.410672
  5       77  74  19 114 153   8    0.371124
  6       73 157 130  72 129 160    0.307531
  7      237 128 111 180 178   8    0.259996
  8      113  96  35  58 155   9    0.144981
  9      140 169 151 108  19   1    0.141826
 10      139 167 148 109  19   3    0.126465
total fitness = 3.220082
```

```
generation : 3
member    a   b   c   d   e   f     fitness
 1      109 136  19 124 153   8    0.687612
 2      109 136 167 124 178 128    0.507279
 3      107 134 165 124 178 130    0.504466
 4      107 134 165 124 178 130    0.504466
 5      107 134 165 124 178 130    0.504466
 6       77  74 167 114 178 128    0.433926
 7       97  96  35 122 153   9    0.364460
 8       73 157 130  72 131 160    0.306610
 9      111 128 237 180 178   8    0.262182
10      113  32  35  58 155   9    0.260493
total fitness = 4.335960

generation : 4
member    a   b   c   d   e   f     fitness
 1      109 153  19 124 136   8    0.780331
 2      109 136  17 124 153   8    0.688357
 3      109 136  19 124 153   8    0.687612
 4      109 136 167 124 162 128    0.596630
 5      109  72 167 120 178 128    0.538023
 6      107 165 134 124 178 130    0.512182
 7      109 136 167 124 178 128    0.507279
 8      107 134 165 125 178 130    0.506381
 9      107 132 173 244 178   8    0.500508
10       77  74 167 112 178 128    0.432071
total fitness = 5.749373

generation : 5
member    a   b   c   d   e   f     fitness
 1      109 153  19 124 136   8    0.780331
 2      109 136  19 124 146   0    0.734663
 3      109 136 167 124 136   8    0.706376
 4      109 136  17 124 153   8    0.688357
 5      109 136  19 124 153   8    0.687612
 6      109 153  19 124 162 128    0.641696
 7      107 132 173 244 144   8    0.640597
 8      109 136 167 124 162 130    0.598332
 9      109 136 167 124 162 128    0.596630
10      107 165 134 124 178 128    0.515018
total fitness = 6.589612

generation : 6
member    a   b   c   d   e   f     fitness
 1      109 153  19 124 136   8    0.780331
 2      109 136 131 124 136   8    0.779965
 3      109 153  55 124 136   8    0.753013
 4      109 136  17 124 144 128    0.750614
 5      109 136  19 124 146 128    0.734663
 6      109 153 136 124  19   8    0.702417
 7      109 136  19 124 155   8    0.673481
 8      111 128 173 244 144   8    0.648797
 9      107 134 173 244 144   8    0.644697
10      105 140 167 124 162 128    0.574995
total fitness = 7.042974

generation : 7
member    a   b   c   d   e   f     fitness
 1      109 136  19 124 136   8    0.798055
 2        2 250 122  35 136   8    0.786499
 3      109 153  19 124 136   8    0.780331
 4      109 136 131 124 136   8    0.779965
 5      109 136 131 124 140   8    0.766247
 6      109 153  55 124   8   8    0.745155
 7      109 153 136 124 147   8    0.711185
 8      109 153  19 124 155   8    0.686277
 9      109 136  19 126 155   8    0.680830
10       33 221 239 197 144   8    0.675347
total fitness = 7.409892
```

```
generation : 8
member     a   b   c   d   e   f    fitness
  1      109 136  19 124 136   8   0.798055
  2        2 250 122  35 136   8   0.786499
  3      109 136 131 124 140   8   0.766247
  4      109 136 131 124 140   8   0.766247
  5       14 250 114  47 136   8   0.749244
  6      109 136  51 124 136   8   0.745108
  7      109 185  19 124 136   8   0.724608
  8      232  49 161 242 136   8   0.705596
  9      107 148  29 125 155   8   0.684091
 10      101 184  40 111 136   8   0.681633
total fitness = 7.407328

generation : 9
member     a   b   c   d   e   f    fitness
  1      109 136  19 124 136   8   0.798055
  2      109 138  19 124 136   8   0.797571
  3        2 250 122  35 136   8   0.786499
  4       14 250 122  47 136   8   0.785282
  5      109 136 131 124 136   8   0.779965
  6      109 137 131 124 140   8   0.765656
  7      109 136  51 125 136   8   0.748003
  8       14 250 114  47 140   8   0.733966
  9      109 185  19 124 152   8   0.686029
 10      107 148  29 124 155   8   0.681161
total fitness = 7.562187

generation : 10
member     a   b   c   d   e   f    fitness
  1      109 136  19 124 136   8   0.798055
  2      109 138  19 124 136   8   0.797571
  3      124 136 131 109 136   8   0.774742
  4       13 248 115  44 136   8   0.756014
  5      109 168  19 124 136   8   0.752245
  6      110 146  29 111 140   8   0.712579
  7      109 153  19 124 152   8   0.707735
  8       98 138 138 115 136   8   0.651491
  9       11 252 114  60 155   8   0.641814
 10      109 136 114  47 140   8   0.332924
total fitness = 6.925170

generation : 11
member     a   b   c   d   e   f    fitness
  1      109 136  19 124 136   8   0.798055
  2      109 136  19 124 136   8   0.798055
  3      109 138  19 124 136   8   0.797571
  4      109 138  19 124 136   8   0.797571
  5      109 138  19 124 136   8   0.797571
  6      109 168  19 124 136   8   0.752245
  7        8 248 114  41 136   8   0.750898
  8       15 248 114  44 139   8   0.738042
  9      103 138 139 118 136   8   0.695293
 10      168 248 103 162 136   8   0.682839
total fitness = 7.608139

generation : 12
member     a   b   c   d   e   f    fitness
  1      109 136  19 124 136   8   0.798055
  2      109 136  19 124 136   8   0.798055
  3      109 136  19 124 136   8   0.798055
  4      109 140  19 124 136   8   0.796519
  5      109 168  19 124 136   8   0.752245
  6       15 248 114  44 136   8   0.750012
  7        8 248 114  41 139   8   0.741800
  8      105 168  19 124 136   8   0.737097
  9      232 168  19 250 136   8   0.702087
 10       45 248 103  36 136   8   0.690318
total fitness = 7.564243
```

```
generation : 13
member    a   b   c   d   e   f    fitness
  1     109 136  19 124 136   8   0.798055
  2     109 136  19 124 136   8   0.798055
  3     109 168  19 124 136   8   0.752245
  4      11 248 114  44 136   8   0.751595
  5     231  34 169 248 136   8   0.728519
  6     232 168  19 251 136   8   0.704214
  7      45 248 103  37 136   8   0.690235
  8     179 234  98 204 136   8   0.666934
  9     109 216  83 124 136   8   0.625815
 10     109 248 114 124 136   8   0.546685
total fitness = 7.062352

generation : 14
member    a   b   c   d   e   f    fitness
  1     109 136  19 124 136   8   0.798055
  2     109 136  19 124 136   8   0.798055
  3     124 136  19 109 136   8   0.766389
  4     109  19 168 124 136   8   0.754880
  5     109 168  19 124 136   8   0.752245
  6     109  34  19 248 136   8   0.736873
  7     109 152  83 124 136   8   0.730776
  8     231 136 169 124 136   8   0.717672
  9     109 200  19 124 136   8   0.695777
 10     109 216  87 124 136   8   0.621822
total fitness = 7.372542

generation : 15
member    a   b   c   d   e   f    fitness
  1     109 136  19 124 136   8   0.798055
  2     109 136  19 124 136   8   0.798055
  3     109 136  23 124 136   8   0.792608
  4     109 136  19 124 140   8   0.780875
  5     125  34  19 248 136   8   0.775132
  6     231 136 169 124 136   8   0.717672
  7     231 136 169 124 136   8   0.717672
  8     109 216  83 124 136   8   0.625815
  9     105 244 128  90 136   8   0.460689
 10     168 109  19 124 136   8   0.346228
total fitness = 6.812801

generation : 16
member    a   b   c   d   e   f    fitness
  1     109 136  19 124 136   8   0.798055
  2     109 136  19 124 136   8   0.798055
  3     109 152  19 124 136   8   0.781785
  4     109 136  19 124 140   8   0.780875
  5     109 136 145 124 136   8   0.750354
  6     231 136 169 124 136   8   0.717672
  7     109 136  83 124 140   8   0.691145
  8     125 136  19 248  34   8   0.674024
  9     109 216  19 124 136   8   0.669818
 10     231 136  43 124 136   8   0.655733
total fitness = 7.317515

generation : 17
member    a   b   c   d   e   f    fitness
  1     109 136  19 124 136   8   0.798055
  2     109 136  19 124 136   8   0.798055
  3     109 136  19 124 136   8   0.798055
  4     109 152  19 124 136   8   0.781785
  5     109 136  19 124 140   8   0.780875
  6     109 136  19 124 140   8   0.780875
  7     239 136   3 124 136   8   0.767086
  8     231 136 169 124 128   8   0.724726
  9     101 136  59 124 136   8   0.696982
 10     125 136  19 248  34   8   0.674024
total fitness = 7.600517
```

```
generation : 18
member    a   b   c   d   e   f    fitness
 1      125 136  19 120 136   8   0.831754
 2      109 136  19 124 136   8   0.798055
 3      109 136  19 124 136   8   0.798055
 4      109 136  19 124 136   8   0.798055
 5      109 136  19 124 136   8   0.798055
 6      109 136   3 124 140   8   0.776964
 7      101 136  59 124 136   8   0.696982
 8      101 136  59 124 136   8   0.696982
 9      101 136  59 124 152   8   0.637395
10      109 136  19 252  34   8   0.624220
total fitness = 7.456518

generation : 19
member    a   b   c   d   e   f    fitness
 1      125 136   3 120 136   8   0.845862
 2      125 136  19 120 136   8   0.831754
 3      109 136  19 124 128   8   0.802734
 4      109 136  19 124 136   8   0.798055
 5      109 136  19 252 136   8   0.718168
 6      101 136  59 252 136   8   0.667124
 7       93 128  51 116 136   8   0.622483
 8      109 136  19 124  34   8   0.616363
 9      109 136  19 124  34   8   0.616363
10       25 187 110  48 136   8   0.458066
total fitness = 6.976971

generation : 20
member    a   b   c   d   e   f    fitness
 1      125 136  19 124 136   8   0.853070
 2      125 136   3 120 136   8   0.845862
 3      125 136  19 120 136   8   0.831754
 4      125 138  19 120 136   8   0.831340
 5       99 237 104  94 136   8   0.632728
 6       93 128  51 112 136   8   0.606839
 7       77 136  19 124 136   8   0.579308
 8       25 187 110  48 152   8   0.440803
 9       25 179 110  48 136   8   0.409727
10       40 202 125 191 136   8   0.373972
total fitness = 6.405402

generation : 21
member    a   b   c   d   e   f    fitness
 1      125 136  19 124 128   8   0.876997
 2      125 136  19 124 136   8   0.853070
 3      125 136  19 124 136   8   0.853070
 4      125 136   3 120 136   8   0.845862
 5      125 138  27 120 136   8   0.819418
 6      109 136  19 124 136   8   0.798055
 7       93 136  19 124 136   8   0.696143
 8      225  80 207 218 136   8   0.663390
 9      115 253 104  94 152   8   0.598072
10       77 138  19 120 136   8   0.566039
total fitness = 7.570115

generation : 22
member    a   b   c   d   e   f    fitness
 1      125 136  17 124 128   8   0.880364
 2      125 136  19 124 128   8   0.876997
 3      125 136  19 124 128   8   0.876997
 4      125 136  19 124 128   8   0.876997
 5      125 136  19 124 136   8   0.853070
 6       26 247 124  36 128   8   0.798776
 7       97 144  15 120 136   8   0.702571
 8       93 136  17 124 136   8   0.694938
 9      237  72 211 222 136   8   0.679648
10      115 253 104  94 152   8   0.598072
total fitness = 7.838429
```

```
generation : 23
member    a   b   c   d   e    f    fitness
  1     125 136  17 124 128   8   0.880364
  2     125 136  17 124 128   8   0.880364
  3     125 136  17 124 128   8   0.880364
  4     125 136  19 124 128   8   0.876997
  5     109 152   1 124 128   8   0.773620
  6     113 128  31 120 136   8   0.771221
  7     115 144 104 120 152   8   0.642440
  8      91 229  80  70 152   8   0.609864
  9     216 125 246 238 136   8   0.594220
 10      97 253  15  94 136   8   0.517027
total fitness = 7.426481

generation : 24
member    a   b   c   d   e    f    fitness
  1     125 136  16 124 128   8   0.882267
  2     125 136  17 124 128   8   0.880364
  3     125 136  17 124 128   8   0.880364
  4     125 136  17 124 128   8   0.880364
  5     125 136  17 124 128   8   0.880364
  6     125 136  23 124 128   8   0.868194
  7     125 136  19 120 128   8   0.857784
  8     125 136  17 124 136   8   0.856154
  9     113 128  31 124 136   8   0.787858
 10     114 195  40 103 152   8   0.664500
total fitness = 8.438215

generation : 25
member    a   b   c   d   e    f    fitness
  1     125 136  16 124 128   8   0.882267
  2     125 136  17 124 128   8   0.880364
  3     125 136  17 124 128   8   0.880364
  4     125 136  17 124 128   8   0.880364
  5     125 136  17 124 128   8   0.880364
  6     125 136  19 124 128   8   0.876997
  7     125 136  21 124 128   8   0.872861
  8     125 136  23 124 128   8   0.868194
  9     249 173  37 244 128   8   0.758011
 10     125 136 128 120  19   8   0.755790
total fitness = 8.535578

#operator selections  : MacroCrossover: 23
                        MicroCrossover: 26
                        MixedCrossover: 23
                        Mutate        : 24
                        Jump          : 12
                        Scale         : 7
                        Rotate        : 10

pixsetdistance(target,best member)     : 0.117785

#affine transformations in IFS-code    : 1

total CPU time used = 845 [seconds] = 0.23 [hours]
```

References

[ADL77] Adler, M.J. and Van Doren, C.L.
GREAT TREASURY OF WESTERN THOUGHT.
New York, NY, USA: R.R. Bowker Company, 1977.

[ATA91] Atallah, M.J.
COMPUTING SOME DISTANCE FUNCTIONS BETWEEN POLYGONS.
Pattern Recognition (USA), vol.24(1991), no.8, p.775-81.

[BAR84] Barnsley, M.F. and Demko, S.G.
RATIONAL APPROXIMATION OF FRACTALS.
Proc. Rational Approximation and Interpolation, Tampa, FL, USA, 12-16 Dec. 1983, p.73-88.
New York, NY, USA: Springer-Verlag Inc., 1984.

[BAR85] Barnsley, M.F. and Demko, S.G.
ITERATED FUNCTION SYSTEMS AND THE GLOBAL CONSTRUCTION OF FRACTALS.
Proc. Royal Society of London, vol.A399(1985), no.1817, p.243-75.
London, UK: Royal Society, 1985.

[BAR86a] Barnsley, M.F. et al.
SOLUTION OF AN INVERSE PROBLEM FOR FRACTALS AND OTHER SETS.
Proc. Natl. Acad. Sci. (USA), vol.83(1986), p.1975-7.
Washington, DC, USA: NAS, 1986.

[BAR86b] Barnsley, M.F.
FRACTAL FUNCTIONS AND INTERPOLATION.
Constructive Approximation, vol.2(1986), p.303-29.

[BAR88a] Barnsley, M.F.
FRACTALS EVERYWHERE.
London, UK: Academic Press Inc. Ltd., 1988.

[BAR88b] Barnsley, M.F. and Jacquin, A.E.
APPLICATION OF RECURRENT ITERATED FUNCTION SYSTEMS TO IMAGES.
Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.1001(1988), p.122-31.

[BAR88c] Barnsley, M.F. and Sloan, A.D.
A BETTER WAY TO COMPRESS IMAGES.
Byte (USA), vol.13(1988), p.215-23.

[BAR88d] Barnsley, M.F. et al.
HARNESSING CHAOS FOR IMAGE SYNTHESIS.
Computer Graphics (USA), vol.22(1988), no.4, p.131-40.

[BAR88e] Barnsley, M.F. and Elton, J.H.
A NEW CLASS OF MARKOV PROCESSES FOR IMAGE ENCODING.
Advances in Applied Probability, vol.20(1988), p.14-32.

[BAR89] Barnsley, M.F. et al.
RECURRENT ITERATED FUNCTION SYSTEMS.
Constructive Approximation, vol.5(1989), p.3-31.

[BAR90] Barnsley, M.F.
IMAGE COMPRESSION USING THE FRACTAL TRANSFORM.
Proc. Image processing 90 - The Key Issues, London, UK, 9-11 Oct. 1990.
London, UK: Blenheim Online, 1990.

[BAX86] Baxandall, P.R. and Liebeck, H.
VECTOR CALCULUS.
Oxford, UK: Oxford University Press, 1986.

[BEA91] Beaumont, J.M.
IMAGE DATA COMPRESSION USING FRACTAL TECHNIQUES.
BT Technol. J., vol.9(1991), no.4, p.93-109.

[BEC64] Beckenbach, E.F. et. al.
APPLIED COMBINATORIAL MATHEMATICS.
New York, NY, USA: John Wiley & Sons, Inc., 1964.

[BEC90] Becker, K-H. and Dörfler, M.
DYNAMICAL SYSTEMS AND FRACTALS: COMPUTER GRAPHICS EXPERIMENTS IN PASCAL, reprint.
New York, NY, USA: Cambridge University Press, 1990.

[BHA91] Bhanu, B. et.al.
SELF-OPTIMIZING IMAGE SEGMENTATION SYSTEM USING A GENETIC ALGORITHM.
Proc. 4th Int. Conf. on Genetic Algorithms, San Diego, CA, USA, 13 - 16 Jul. 1991, p.362-9.
San Mateo, CA, USA: Morgan Kaufmann Publishers, Inc., 1991.

[BRE91] Bressloff, P.C. and Taylor, J.G.
DISCRETE TIME LEAKY INTEGRATOR NETWORK WITH SYNAPTIC NOISE.
Neural Networks (USA), vol.4(1991), p.789-801.

[BRO90] Broer, H.W. et al.
DYNAMISCHE SYSTEMEN EN CHAOS: EEN REVOLUTIE VANUIT DE WISKUNDE.
Utrecht, Netherlands: Epsilon Uitgaven, 1990.

[BUD90] Budach, L.
RECURSIVE VLSI-DESIGN AND FRACTALS.
Syst. Anal. Model. Simul. (East Germany), vol.7(1990), no.11-12, p.825-51.

[BUS85] Van Buskirk, R. and Jeffries, C.
OBSERVATION OF CHAOTIC DYNAMICS OF COUPLED NONLINEAR OSCILLATORS.
Physical Review A, vol.31(1985), no.5, p.3332-57.

[COR90] Cormen, T.H. et al.
INTRODUCTION TO ALGORITHMS, 2nd printing.
London, UK: The MIT Press, 1990.

[CRE89] Creutzburg, R. and Ivanov, E.
FAST ALGORITHM FOR COMPUTING FRACTAL DIMENSIONS OF IMAGE SEGMENTS.
Proc. 5th Int. Conf. - Artificial Intelligence and Information-
Control Systems of Robots, Strbske Pleso, Chechoslovakia, 6-10 Nov. 1989, p.397-9.
Amsterdam, Netherlands: North-Holland, 1989.

[CRI91] Crilly, A.J. et. al.
FRACTALS AND CHAOS.
Berlin, Germany: Springer-Verlag, 1991.

[CUL90] Culik, K. and Dube, S.
METHODS FOR GENERATING DETERMINISTIC FRACTALS AND IMAGE COMPRESSION.
Proc. 6th Int. Meeting of Young Computer Scientists, Smolenice,
Chechoslovakia, 19-23 Nov. 1990, p.2-28.
Berlin, Germany: Springer-Verlag, 1990.

[DAR80] Darwin, B.
THE OXFORD DICTIONARY OF QUOTATIONS.
Oxford, UK: Oxford University Press, 1980.

[DAV91] Davis, L.
HANDBOOK OF GENETIC ALGORITHMS.
New York, NY, USA: Van Nostrand Reinhold, 1991.

[DEM85] Demko, S.G. et al.
CONSTRUCTION OF FRACTAL OBJECTS WITH ITERATED FUNCTION SYSTEMS.
Comput. Graphics (USA), vol.19(1985), no.3, p.271-8.

[DEV90] Devaney, R.L.
CHAOS, FRACTALS AND DYNAMICS: COMPUTER EXPERIMENTS IN MATHEMATICS.
Reading, MA, USA: Addison-Wesley Publishing Company Inc., 1990.

[DOU82] Douady, A. and Hubbard, J.
ITÉRATION DES POLYNÔMES QUADRATIQUES COMPLEXES.
Comptes Rendus Acad. Sc. (Paris), vol.294(1982), p.123-6.

[DUB87] Dubuc, B. et al.
THE VARIATION METHOD: A TECHNIQUE TO ESTIMATE THE FRACTAL DIMENSION OF SURFACES.
Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.845(1987), p.241-8.

[DUR85] Durbin, J.R.
MODERN ALGEBRA, 2nd ed.
New York, NY, USA: John Wiley & Sons, Inc., 1985.

[ELT87] Elton, J.
AN ERGODIC THEOREM FOR ITERATED MAPS.
J. of Ergodic Theory and Dynamical Syst., vol.7(1987), p.481-8.

[FAL90] Falconer, K.J.
FRACTAL GEOMETRY: MATHEMATICAL FOUNDATIONS AND APPLICATIONS.
Chicester, UK: John Wiley & Sons Ltd., 1990.

[FED89] Feder, J.
FRACTALS, 4th printing.
New York, NY, USA: Plenum Press, 1989

[FEI79] Feigenbaum, M.J.
THE UNIVERSAL METRIC PROPERTIES OF NONLINEAR TRANSFORMATIONS.
J. of Stat. Phys., vol.21(1979), p.669-706.

[FEI80] Feigenbaum, M.J.
UNIVERSAL BEHAVIOR IN NONLINEAR SYSTEMS.
Los Alamos Science, vol.1(1980), p.4-27.

[FOL90] Foley, J.D. et al.
COMPUTER GRAPHICS: PRINCIPLES AND PRACTICE, 2nd ed.
Reading, MA, USA: Addison-Wesley Publishing Company, 1990.

[FRE90] Freeland, G.C. and Durani, T.S.
IFS FRACTALS AND THE WAVELET TRANSFORM.
Proc. Int. Conf. on Acoustics, Speech and Signal Processing
(ICASSP 90), Albuquerque, NM, USA, 3-6 Apr. 1990, vol.4(1990), p.2345-8.
New York, NY, USA: IEEE, 1990.

[GAL68] Gallager, R.G.
INFORMATION THEORY AND RELIABLE COMMUNICATION.
New York, NY, USA: John Wiley & Sons, Ltd., 1986.

[GAR79] Garey, M.R. and Johnson, D.S.
COMPUTERS AND INTRACTABILITY: A GUIDE TO THE THEORY OF NP-COMPLETENESS.
New York, NY, USA: W.H. Freeman and Company, 1979.

[GAR88] Garnett, L.
A COMPUTER ALGORITHM FOR DETERMINING THE HAUSDORFF DIMENSION OF CERTAIN FRACTALS.
Math. Comput. (USA), vol.51(1988), no.183, p.291-300.

[GIP90a] Gipser, T.
FRAKTALE - EINE NEUE SPRACHE DER WISSENSCHAFT.
Bull. Schweiz. Elektrotech. Ver. Verb. Schweiz. Elektr. werke (Switzerland), vol.81(1990), no.9, p.43-8.

[GIP90b] Gipser, T.
DATENKOMPRESSION MITELS ITERIERTER FUNKTIONENSYSTEME (IFS).
Bull. Schweiz. Elektrotech. Ver. Verb. Schweiz. Elektr. werke (Switzerland), vol.81(1990), no.9, p.49-54.

[GLE87] Gleick, J.
CHAOS: MAKING A NEW SCIENCE.
New York, NY, USA: Penguin Books Inc., 1987.

[GUL92] Gulick, D.
ENCOUNTERS WITH CHAOS.
New York, NY, USA: McGraw-Hill, Inc., 1992.

[HAB91] Habel, A. and Kreowski, H-J.
COLLAGE GRAMMARS.
Proc. 4th Int. Workshop on Graph Grammars and their application to Computer Science, Bremen, Germany, 5 - 9 Mar. 1990, p.411-29.
Berlin, Germany: Springer-Verlag, 1991.

[HOF85] Hofstadter, D.R.
METAMAGICAL THEMA'S: QUESTING FOR THE ESSENCE OF MIND AND PATTERN.
New York, NY, USA: Basic Books, Inc., 1985.

[HOL75] Holland, J.H.
ADAPTATION IN NATURAL AND ARTIFICIAL SYSTEMS.
Ann Arbor, MI, USA: University of Michigan Press, 1975.

[HOR88] Horn, R.A. and Johnson, C.R.
MATRIX ANALYSIS.
New York, NY, USA: Cambridge University Press, 1988.

[HOR90] Horn, A.N.
IFSs AND INTERACTIVE IMAGE SYNTHESIS.
Comput. Graph. Forum (NL), vol.9(1990), no.2, p.127-38.

[HUN79] Hunter, G.M. and Steiglitz, K.
OPERATIONS ON IMAGES USING QUAD TREES.
IEEE Trans. Pattern Anal. Mach. Intell. (USA), vol.PAMI-1(1979), no.2, p.145-53.
New York, NY, USA: IEEE, 1979.

[HUT81] Hutchinson, J.E.
FRACTALS AND SELF SIMILARITY.
Indiana Univ. Mathematics J., vol.30(1981), no.5, p.713-47.

[JAC90] Jacquin, A.E.
FRACTAL IMAGE CODING BASED ON A THEORY OF ITERATED CONTRACTIVE IMAGE TRANSFORMATIONS.
Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.1360(1990), p.227-39.

[KAA87] Kaandorp, J.A.
INTERACTIVE GENERATION OF FRACTAL OBJECTS.
Proc. Eur. Comp. Graph. Conf. and Exhibition (EUROGRAPHICS '87), Amsterdam, Netherlands, 24-28 Aug 1987, p.181-96.
Amsterdam, Netherlands: North-Holland, 1987.

[KAN88] Kaneko, H.
FRACTAL MATRIX MODEL AND ITS APPLICATION TO TEXTURE ANALYSIS.
Trans. Inst. Electron. Inf. Commun. Eng. E (Japan), vol.E71(1988), no.12, p.1221-8.

[KEL87] Keller, J.M. et al.
CHARACTERISTICS OF NATURAL SCENES RELATED TO THE FRACTAL DIMENSION.
IEEE Trans. Pattern Anal. Mach. Intell. (USA), vol.PAMI-9(1987), no.5, p.621-7.
New York, NY, USA: IEEE, 1987.

[KEL89] Keller, J.M. et al.
TEXTURE DESCRIPTION AND SEGMENTATION THROUGH FRACTAL GEOMETRY.
Comput. Vis. Graph. Image Process. (USA), vol.45(1989), no.2, p.150-66.

[KOC89a] Kocsis, S.M.
DIGITAL COMPRESSION AND ITERATED FUNCTION SYSTEMS.
Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.1153(1989), p.19-27.

[KOC89b] Kocsis, S.M.
FRACTAL-BASED IMAGE COMPRESSION.
Record 23rd Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 30 Oct - 1 Nov 1989, vol.1(1989) p.177-81.
San Jose, CA, USA: Maple Press, 1989.

[KRA87] Kraus, A.D.
MATRICES FOR ENGINEERS.
Berlin, Germany: Springer-Verlag, 1987.

[LAY90] Layman, J.W. and Womack, T.E.
LINEAR MARKOV ITERATED FUNCTION SYSTEMS.
Comput. & Graphics, vol.14(1990), no. 2, p.343-53.

[LEM81] Lemaréchal, C. et al.
ON A BUNDLING ALGORITHM FOR NONSMOOTH OPTIMIZATION.
In: Nonlinear Programming 4, ed. by O.L. Mangasarian et al., p.245-82.
London, UK: Academic Press, 1981.

[LEV87] Levy-Vehel, J. and Gagalowicz, A.
SHAPE APPROXIMATION BY A FRACTAL MODEL.
Proc. Eur. Comp. Graph. Conf. and Exhibition (EUROGRAPHICS '87), Amsterdam, Netherlands, 24-28 Aug 1987, p.159-80.
Amsterdam, Netherlands: North-Holland, 1987.

[LEV88] Levy-Vehel, J. and Gagalowicz, A.
FRACTAL APPROXIMATION OF 2D OBJECT.
Proc. European Computer Graphics Conference and Exhibition (EUROGRAPHICS '88), Nice, France, 12-16 Sep. 1988, p.313-26.
Amsterdam, Netherlands: North-Holland, 1988.

[LEV91] Levenick, J.R.
INSERTING INTRONS IMPROVES GENETIC ALGORITHM SUCCESS RATE: TAKING A CUE FROM BIOLOGY.
Proc. 4th Int. Conf. on Genetic Algorithms, San Diego, CA, USA, 13 - 16 Jul. 1991, p.123-7.
San Mateo, CA, USA: Morgan Kaufmann Publishers, Inc., 1991.

[LEW81] Lewis, H.R. and Papadimitriou, C.H.
ELEMENTS OF THE THEORY OF COMPUTATION.
Englewood Cliffs, NJ, USA: Prentice-Hall, Inc., 1981.

[LIB87] Libeskind-Hadas, R. and Maragos, P.
APPLICATION OF ITERATED FUNCTION SYSTEMS AND SKELETONIZATION TO SYNTHESIS OF FRACTAL IMAGES.
Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.845(1987), p.276-84.

[LOR63] Lorenz, E.N.
DETERMINISTIC NONPERIODIC FLOW.
J. of the Athmospheric Sciences, vol.20(1963), p.130-41.

[MAN67] Mandelbrot, B.B.
HOW LONG IS THE COAST OF BRITAIN?
Science, vol.155(1967), p.636-8.

[MAN83] Mandelbrot, B.B.
THE FRACTAL GEOMETRY OF NATURE.
New York, NY, USA: W.H. Freeman and Company, 1983.

[MAN89] Mantica, G. and Sloan, A.
CHAOTIC OPTIMIZATION AND THE CONSTRUCTION OF FRACTALS: SOLUTION OF AN INVERSE PROBLEM.
Complex Systems, vol.3(1989), no.1, p.37-62.

[MAN90] Manohar, M. et. al.
TEMPLATE QUADTREES FOR REPRESENTING REGION AND LINE DATA PRESENT IN BINARY IMAGES.
Comput. Vis. Graph. Image Process. (USA), vol.51(1990), p.338-54.

[MEN91] Meneghini, R. and Stutz, P.
WIE BILDER AUS SICH SELBST ENSTEHEN.
Bull. Schweiz. Elektrotech. Ver. Verb. Schweiz. Elektr. werke (Switzerland), vol.82(1991), no.21, p.11-5.

[MON90] Monro, D.M. et al.
DETERMINISTIC RENDERING OF SELF-AFFINE FRACTALS.
IEE Colloquium on "Applications of Fractal Techniques in Image Processing" (Digest no.171), London, UK, 3 Dec 1990, p.5/1-4.
London, UK: IEE, 1990.

[MOO87] Moon, F.C.
CHAOTIC VIBRATIONS: AN INTRODUCTION FOR APPLIED SCIENTISTS AND ENGINEERS.
New York, NY, USA: John Wiley & Sons, Inc., 1987.

[PEI86] Peitgen, H-O. and Richter, P.H.
THE BEAUTY OF FRACTALS: IMAGES OF COMPLEX DYNAMICAL SYSTEMS.
Berlin, Germany: Springer-Verlag, 1986.

[PEI88] Peitgen, H-O. et al.
THE SCIENCE OF FRACTAL IMAGES.
New York, NY, USA: Springer-Verlag Inc., 1988.

[PEI89] Peitgen, H-O. and Jürgens, H.
FRACTALS: A NEW CHALLENGE TO MODEL REALITY.
Proc. IFIP 11th World Computer Congress, San Francisco, CA, USA, 28 Aug - 1 Sep 1989, p.581-8.
Amsterdam, Netherlands: North-Holland, 1989.

[PEI92] Peitgen, H-O. et. al.
CHAOS AND FRACTALS: NEW FRONTIERS OF SCIENCE.
Berlin, Germany: Springer-Verlag, 1992.

[PEN84] Pentland, A.P.
FRACTAL-BASED DESCRIPTION OF NATURAL SCENES.
IEEE Trans. Pattern Anal. Mach. Intell. (USA), vol.PAMI-6(1984), no.6, p.661-74.
New York, NY, USA: IEEE, 1984.

[RED89] Reddaway, S.F. et al.
FRACTAL GRAPHICS AND IMAGE COMPRESSION ON A SIMD PROCESSOR.
Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation, Fairfax, VA, USA, 10-12 Oct 1988, p.265-74.
Washington DC, USA: IEEE Computer Society Press, 1989.

[REG81] Reghbati, H.K.
AN OVERVIEW OF DATA COMPRESSION TECHNIQUES.
Computer, vol.14(1981), no.4, p.71-5.

[RIG88] Rigaut, J.P.
AUTOMATED IMAGE SEGMENTATION BY MATHEMATICAL MORPHOLOGY AND FRACTAL GEOMETRY.
Journal of Microscopy (UK), vol.150(1988), pt.1, p.21-30.

[ROB90] Robinson, E.A.
EINSTEIN'S RELATIVITY IN METAPHOR AND MATHEMATICS.
Englewood Cliffs, NJ, USA: Prentice-Hall, Inc., 1990.

[SCH91] Schaffer, J.D. and Eshelman, L.J.
ON CROSSOVER AS AN EVOLUTIONARY VIABLE STRATEGY.
Proc. 4th Int. Conf. on Genetic Algorithms, San Diego, CA, USA, 13 - 16 Jul. 1991, p.61-8.
San Mateo, CA, USA: Morgan Kaufmann Publishers, Inc., 1991.

[SHO91] Shonkwiler, R. et. al.
GENETIC ALGORITHMS FOR THE 1-D FRACTAL INVERSE PROBLEM.
Proc. 4th Int. Conf. on Genetic Algorithms, San Diego, CA, USA, 13 - 16 Jul. 1991, p.495-501.
San Mateo, CA, USA: Morgan Kaufmann Publishers, Inc., 1991.

[SIN91] Singer, M. and Berg, P.
GENES & GENOMES.
Mill Valley, CA, USA: University Science Books, 1991.

[SNA89] Snapper, E. and Troyer, R.J.
METRIC AFFINE GEOMETRY.
Mineola, NY, USA: Dover Publications, Inc., 1989.

[STA89] Staiger, L.
QUADTREES AND THE HAUSDORFF DIMENSION OF PICTURES.
Report P-MATH-06/89, p.1-13.
Berlin, DDR: Akademie der Wissenschaften der DDR,
Karl-Weierstrass-institut für Mathematik, 1989.

[STA91a] Stark, J.
ITERATED FUNCTION SYSTEMS AS NEURAL NETWORKS.
Neural Networks, vol.4(1991), p.679-90.

[STA91b] Stark, J.
A NEURAL NETWORK TO COMPUTE THE HUTCHINSON METRIC IN FRACTAL IMAGE PROCESSING.
IEEE Trans. Neural Networks, vol.2(1991), no.1, p.156-8.

[STE89] Stevens, R.T.
FRACTAL PROGRAMMING IN C.
Redwood City, CA, USA: M&T Publishing Inc., 1989.

[UED81] Ueda, Y. and Akamatsu, N.
CHAOTICALLY TRANSITIONAL PHENOMENA IN THE FORCED NEGATIVE-RESISTANCE OSCILLATOR.
IEEE Trans. Circ. Syst., vol.CAS-28(1981), no.3, p.217-24.

[VEP89] Vepsalainen, A.M. and Ma, J.
ESTIMATING OF FRACTAL AND CORRELATION DIMENSIONS FROM 2D AND 3D-IMAGES.
Proc. SPIE - Int. Soc. Opt. Eng. (USA), vol.1199(1989), p.431-8.

[WAI90] Waite, J.
A REVIEW OF ITERATED FUNCTION SYSTEM THEORY FOR IMAGE COMPRESSION.
IEEE Colloquium on "Applications of Fractal Techniques in Image Processing" (Digest no.171), London, UK, 3 Dec 1990, p.1/1-8.
London, UK: IEEE, 1990.

[WAL86] Walach, E. and Karnin, E.
A FRACTAL BASED APPROACH TO IMAGE COMPRESSION.
Proc. Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP 86), Tokyo, Japan, 7-11 Apr., vol.1(1986), p.529-32.
New York, NY, USA: IEEE, 1986.

[WEL84] Welch, T.A.
A TECHNIQUE FOR HIGH PERFORMANCE DATA COMPRESSION.
Computer, vol.17(1984), no.6, p.8-19.

[WHI91] Whitley, D. et. al.
DELTA CODING: AN ITERATIVE SEARCH STRATEGY FOR GENETIC ALGORITHMS.
Proc. 4th Int. Conf. on Genetic Algorithms, San Diego, CA, USA, 13 - 16 Jul. 1991, p.77-84.
San Mateo, CA, USA: Morgan Kaufmann Publishers, Inc., 1991.

[WIT89] Withers, W.D.
NEWTON'S METHOD FOR FRACTAL APPROXIMATION.
Constructive Approximation, vol.5(1989), p.151-70.

[WRI92] Wright, A.
FRACTALS: TRANSFORM IMAGE COMPRESSION.
Electronics World + Wireless World (UK), vol.3(1992), p.208-211.

[YAN88] Yang, K-M. et al.
FRACTAL BASED IMAGE CODING SCHEME USING PEANO SCAN.
Proc. IEEE Int. Symposium on Circuits and Systems, Espoo, Finland,
7-9 Jun 1988, vol.3(1988), p.2301-4.
New York, NY, USA: IEEE 1988.

[ZHA91] Zhang, N. and Yan, H.
HYBRID IMAGE COMPRESSION METHOD BASED ON FRACTAL GEOMETRY.
Electron. Lett. (UK), vol.27(1991), no.5, p.406-8.

[ZOR88] Zorpette, G.
FRACTALS: NOT JUST ANOTHER PRETTY PICTURE.
IEEE Spectr. (USA), vol.25(1988), no.10, p.29-31.

Index