Finite Algorithms for Decoding Recurrent Iterated Function Systems

Abbas Edalat

Imperial College Technical Report DoC 95/1

Abstract

We present two finite algorithms, the recurrent probabilistic domain algorithm for decoding a recurrent iterated function system (IFS) and the vector recurrent probabilistic domain algorithm for decoding a vector recurrent IFS on the digitised screen. Recurrent IFSs and vector recurrent IFSs are used for fractal image compression and our algorithms are the first finite algorithms in the state of art. They have the following advantages compared to the previous two known algorithms in the field.

- Our algorithms terminate in finite time on any digitised screen without needing to fix a number of iterations in advance.
- There is a simple complexity analysis for the algorithms.
- The algorithms produce a good quality image up to several times faster than the other algorithms.

1 Introduction

Recurrent iterated function systems (IFSs) and vector recurrent IFSs are the basis of the fractal image compressor VRIFS$^{\text{TM}}$ developed by M. F. Barnsley's Iterated Systems Inc. in the states [2, 5]. They encode an image as a measure on the screen exploiting the self-similarities of the image. Recurrent IFSs and vector recurrent IFSs generalise IFSs and it is useful to describe IFSs before presenting these two more general cases. We will, therefore, first explain how images can be regarded as measures and how an IFS encodes an image and how it is decoded.

1.1 Images as Measures

In fractal image compression, one seeks to encode and decode a colour or a greyscale digitised image on the computer screen. Using a colour map, a colour or a greyscale image can be presented by assigning a number to each pixel indicating its brightness or intensity. We can think of this number as the weight of the pixel. Any section of the image will then have a total weight equal to the sum of the weights of its pixels. It is technically convenient to rescale the weights of pixels so that the screen as a whole, consisting of say $r \times r$ pixels (where r is the resolution of the screen), has unit weight in total. In this way, we say that the image is represented by a *normalised measure* on the screen, which we can regard as the unit square. Conversely any normalised measure on the screen, i.e any distribution of weights on pixels which add up to one corresponds to an image. We can therefore identify (colour or greyscale) images on the screen and normalised measures on the unit square. Given an image, the set of all pixels which have non-zero weights is called the *support* of the image.

1.2 Iterated Function Systems with Probabilities

An *affine transformation* of the plane is a combination of a rotation, a rescaling, a sheer and a translation in the plane. Any affine transformation $f : \mathbb{R}^2 \to \mathbb{R}^2$ of the plane has the form

$$(x', y') = f(x, y) = (ax + by + g, cx + dy + h)$$

where $(x, y) \in \mathbb{R}^2$ is any point on the plane. It is convenient to express this in matrix notation as

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} g \\ h \end{pmatrix}.$$

The matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

is the linear part of the transformation; it is the combination of a rotation, a rescaling and a sheer. The vector

$$\begin{pmatrix} e \\ f \end{pmatrix}$$

is the translation part of the transformation.

An affine transformation is *contracting* if there is a number s with $0 \leq s < 1$ such that the distance between any two points (x_1, y_1) and (x_2, y_2) on the plane is contracted by at least a factor s, i.e.

$$\sqrt{(x_1' - x_2')^2 + (y_1' - y_2')^2} \leq s(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}).$$

For a contracting transformation f as above, the least number s with this property is called the *contractivity* of the affine transformation and is given by

$$s = \sqrt{\alpha + \beta + \sqrt{(\alpha - \beta)^2 + \gamma^2}}$$

where $\alpha = (a^2 + c^2)/2$, $\beta = (b^2 + d^2)/2$, and $\gamma = ab + cd$.

An *Iterated Function System (IFS)* on the plane is given by a finite collection of contracting affine transformations f_i $(i = 1, 2, \ldots, N)$ of the plane. An *IFS with probabilities* is an IFS such that each f_i is assigned a probability p_i with $0 < p_i < 1$ and $p_1 + p_2 + \ldots + p_N = 1$. Given an IFS with probabilities

$$(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$$

we can store the coefficients of the maps f_i and the probabilities in a table, which gives the code of the IFS with probabilities. For example the following table is the code of an IFS with probabilities with four maps $(N = 4)$.

f	a	b	c	d	g	h	p
1	0.76	0.00	0.00	0.76	0.12	0.25	0.72
2	0.00	0.00	0.00	0.26	0.51	−0.00	0.01
3	0.20	0.26	−0.23	0.22	0.41	0.24	0.13
4	−0.15	−0.28	−0.26	0.24	0.59	0.24	0.24

Table 1.

An IFS with probabilities determines a unique normalised measure, i.e. a unique image on the plane, called the *invariant measure* of the IFS. Therefore, we can say that the IFS encodes this measure or image. The existence and uniqueness of the invariant measure is shown in [2, page 351]. For example the IFS with probabilities with the code given in Table 1 above determines the image of a fern shown in Figure 1. The picture in Figure 2 is gives the support of the image in white.

We will now describe the two prior art algorithms which have been used up to now to generate, i.e. to decode, the image corresponding to an IFS with probabilities. These algorithms should be compared with the new algorithm presented in Section 2.

We refer the reader to [1, 5] for the theoretical basis of the following algorithms. We can assume, by a change of coordinates and rescaling if necessary, that each contracting affine transformation f_i $(i = 1, 2, \ldots, N)$ maps the unit square, denoted by X, into itself (i.e. each transformation maps the unit square to a parallelogram whose sides fall entirely within the unit square). Any point x in the unit square is represented by its closest pixel z_{kl}, for some k and l with $1 \leq k, l \leq r$ on the screen, where r is the resolution.

1.2.1 The Greyscale Photo Copy Algorithm

Given an IFS with probabilities

$$(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$$

the greyscale photo copy algorithm is equipped with an operator M, called the *Markov* operator, which given any image μ as input produces a new image $M(\mu)$ as output. Suppose μ is a digitised image, i.e. for each pixel z_{kl} $(1 \leq k, l \leq r)$ we have $\mu(z_{kl}) \geq 0$ with

$$\sum_{k,l=1}^{r} \mu(z_{kl}) = 1.$$

Then the new image $M(\mu)$ is obtained, by assigning the following weight to pixel z_{kl}:

$$M(\mu)(z_{kl}) = \sum_{i=1}^{n} p_i \mu(f_k^{-1}(z_{kl}))$$

where $f_i^{-1}(z_{kl})$ is the set of all pixels which are mapped to z_{kl} by f_i.

The invariant measure μ^* of the IFS is in fact the unique fixed point of M, i.e. μ^* is the unique normalised measure which satisfies $M(\mu^*) = \mu^*$. Moreover for any initial image μ the sequence of iterates of M, i.e.

$$\mu, M(\mu), M^2(\mu), M^3(\mu), \ldots, M^n(\mu), \ldots$$

always tends to μ^*. This means that if n is sufficiently large, the image represented by $M^n(\mu)$ is visually close to μ^*.

Therefore the algorithm starts with an initial digitised image μ and repeatedly applies M until a convergence to an image takes place. This limiting image is then the required image: It is the digitised approximation to μ^*. In most practical embodiments, the initial image is taken to be a uniformly grey image.

The algorithm eventually produces a good approximation to μ^* for large n. However, as the above formula for $M(\mu)$ shows, calculation of each new iterate of M is computational intensive and it requires a significant amount of memory (as it needs to store $2r^2$ numbers): beyond the capacity of a PC. Furthermore, there is no criteria defining how many iterations be performed, and the number n of iterations has to be fixed in advance by trial and error depending on the IFS and the resolution of the screen.

1.2.2 The Random Iteration Algorithm

The random iteration algorithm generates the image corresponding to an IFS with probabilities

$$(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$$

as follows. Initially, a point x_0 of the unit square is chosen. Then an integer between 1 and N is chosen at random with p_i being the probability of choosing i. Suppose i_1 is chosen. This determines a new point namely $x_1 = f_{i_1}(x_0)$. We then repeat this procedure to choose, say, i_2 and put $x_2 = f_{i_2}(x_{i_1}) = f_{i_2} \circ f_{i_1}(x_0)$. In this way we find a sequence of points

$$x_0, x_1, x_2, x_3, \ldots, x_n, \ldots$$

where

$$x_n = f_{i_n} \circ f_{i_{n+1}} \circ \ldots \circ f_{i_1}(x_0)$$

When n is large enough, the distribution of the first n terms of the above sequence on the screen approximates the required image μ^*, i.e. the invariant measure of the IFS. More precisely, let $L(n, z_{kl})$ be the total number of the first n terms of the above sequence which are represented by the pixel z_{kl}. Then we have

$$\mu^*(z_{kl}) = \lim_{n\to\infty} \frac{L(n, z_{kl})}{n+1}$$

i.e. for large n, the fraction

$$\frac{L(n, z_{kl})}{n+1}$$

is an approximation to $\mu(z_{kl})$.

Like the greysale photo copy algorithm, to use the random iteration algorithm one has to fix a number n of iterations in advance, which for each IFS with probabilities is found by trial and error for any given resolution of the screen. Furthermore, the selection of the integer i from $1, 2, \ldots, N$ with probability p_i is quite time consuming as it takes about $\log_2 N$ computational steps. Last but not least, the algorithm, as it is random, may fail, during the n iterations specified, to visit some of the pixels which have non-zero μ^* value, i.e. which in fact lie on the support of the original image.

1.3 Compression via Self-tiling

We now describe how compression of an image is done, i.e. given a digitised image μ on the screen how to find an IFS with probabilities

$$(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$$

which represents that image closely. Details of the method can be found in [5, page 113].

The aim is to find N contracting affine transformations f_i and probabilities p_i $(i = 1, 2, \ldots, N)$ such that the image represented by the measure

$$M(\mu) = p_1 \mu \circ f_1^{-1} + p_2 \mu \circ f_2^{-1} + \ldots + p_N \mu \circ f_N^{-1}$$

where M is the Markov operator, is visually close to the original image μ. Suppose for now that we have achieved this, i.e. assume we have an IFS with probabilities

$$(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$$

such that $M(\mu) \approx \mu$. Since the unique invariant measure μ^* of the IFS satisfies $M(\mu^*) = \mu^*$, one can infer that $\mu^* \approx \mu$. This means that the invariant measure of the IFS is close to the original image, in other words the above IFS with probabilities gives a good encoding of μ.

Therefore, we have to find

$$(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$$

such that

$$\mu \approx p_1 \mu \circ f_1^{-1} + p_2 \mu \circ f_2^{-1} + \ldots + p_N \mu \circ f_N^{-1}.$$

Each term $p_i \mu \circ f_i^{-1}$ in the above sum represents a copy of the original image transformed by the map f_i and attenuated in brightness by the factor p_i. Therefore the above equation represents a self-tiling, or a *collage*, of μ. Therefore, one proceeds by first assuming all probabilities are zero except p_1. Then f_1 and p_1 are chosen such that $p_1 \mu \circ f_1^{-1}$ approximates a part of the image μ. Next p_2 is allowed to be non-zero, and f_2 and p_2 are chosen so that $p_1 \mu \circ f_1^{-1} + p_2 \mu \circ f_2^{-1}$ approximates a larger part of the image. The procedure is repeated until the whole image μ is covered by such tiles. Then the probabilities are normalised so that they add up to one. The resulting $(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$ gives an encoding of the image μ.

This technique for image compression was used originally in a software developed by M. F. Barnsley, A. Sloan and L. Reuter. (See [1, page 337]). Their system contains two subsystems, *Collage* and *Seurat*. *Collage* finds the IFS code of a given image using the above method. *Seurat* then decodes the image by starting with the IFS code; for this it uses the random iteration algorithm described previously.

2 The Probabilistic Domain Algorithm

The probabilistic domain algorithm generates the invariant measure of an IFS with probabilities, i.e. it decodes an image. It is therefore an alternative to the greyscale photo copy algorithm and the random iteration algorithm described in the last section. It was presented in [7]. In order to understand the vector recurrent probabilistic domain algorithm, it is useful to recall how the probabilistic domain algorithm works as follows.

Suppose $(f_1, f_2, \ldots, f_N; p_1, p_2, \ldots, p_N)$ is an IFS with probabilities. We want to generate the image μ^* corresponding to the IFS by computing $\mu^*(z)$ for each pixel z.

Assume that the contractivity of f_i is s_i $(0 \leq s_i < 1)$, which can be calculated in terms of the coefficients of f_i as given in the previous section. Let s be the maximum value of s_i for $i = 1, \ldots, N$. i.e. $s = \max_{1 \leq i \leq N} s_i$. Then s also satisfies $0 \leq s < 1$. As before, we can assume that each contracting affine transformation f_i $(i = 1, 2, \ldots, N)$ maps the unit square X into itself.

Consider the tree, shown in Figure 3, which for convenience has been depicted upside down. We call it the *IFS tree with transitional probabilities.*

The root of the tree is the unit square X on depth zero. The immediate descendents or *children* of X, situated on depth one, are the images of X under the maps $f_1, f_2, \ldots, f_N$, which we denote simply by $f_1X, f_2X, \ldots, f_NX$. Since an affine transformation sends parallel lines to parallel lines, each f_iX is a parallelogram, contained in X by assumption; and its sides are at most of length s_i. Note that these parallelograms may intersect. On the next depth, the children of f_iX (for each $i = 1, \dots, N$) are the images of the depth one parallelograms, $f_1X, f_2X, \dots, f_NX$, under the map f_i. These are denoted therefore by $f_if_1X, f_if_2X, \dots, f_if_NX$. Each f_if_jX is a smaller parallelogram with sides at most of length s_is_j and is contained in f_iX. Similarly the next levels of the tree are constructed. A typical node of the tree on depth level n is a small parallelogram denoted by $f_{i_1}f_{i_2}\dots f_{i_n}X$, with sides at most of length $s_{i_1}s_{i_2}\dots s_{i_n}$, which is contained in its parent parallelogram $f_{i_1}f_{i_2}\dots f_{i_{n-1}}X$.

We give each of the parallelogram in the tree a weight. The unit square X is given weight one. This weight is distributed among the parallelograms on the next level by giving each f_iX weight p_i. The weight p_i is then redistributed among the children of f_iX by giving f_if_jX weight p_ip_j. Note that

$$p_ip_1 + p_ip_2 + \dots + p_ip_N = p_i(p_1 + p_2 + \dots p_n) = p_i$$

so that the total weight is always preserved. Generally, the parallelogram $f_{i_1}f_{i_2}\dots f_{i_n}X$ is assigned the weight $p_{i_1}p_{i_2}\dots p_{i_n}$.

Each branch of the tree eventually shrinks to a pixel. In fact, if $s_{i_1}s_{i_2}\dots s_{i_n}$ is of the order of the size of a pixel, i.e. of the order of $1/r$ where r is the resolution of the screen, then we can identify the node $f_{i_1}f_{i_2}\dots f_{i_n}X$ with a pixel, and the branch shown in Figure 4 then terminates at this node, since a child of any pixel is clearly the same pixel. This terminating node, represented by that pixel, is therefore a *leaf* of the tree and it has weight $p_{i_1}p_{i_2}\dots p_{i_n}$. Clearly when $n = [-\log r/\log s] + 1$, where $[a]$ is the greatest integer less than or equal to a, all parallelograms on level n have already shrunk to pixels, i.e. the tree will have finite depth n and all the leaves of tree will have depth less than or equal to n.

We can now explain how μ^* is computed. The value $\mu^*(z)$ for a pixel z is given simply by the sum of the weights of all the leaves which are represented by z.

The algorithm therefore traverses the finite tree in some specified way in order to find the leaves and their weights. Then, for each pixel the weights of the leaves represented by that pixel are summed up to give the total weight of each pixel, which determines the normalised measure μ^*.

Clearly then, the algorithm terminates, without needing to specify any fixed number of iterations as one has to do in the case of the other two prior art algorithms.

We can obtain a simple complexity analysis for the algorithm. If $s = \max_{1\le i\le N} s_i$ and $s' = \min_{1\le i\le N} s_i$, then it is easy to see that the height of the tree is between h and h' with

$$h = [-\frac{\log r}{\log s}] + 1 \quad \text{and} \quad h' = [-\frac{\log r}{\log s'}] + 1$$

where $[a]$ is the greatest integer less than or equal to a. A simple calculation shows that at each stage of computation 10 arithmetic operations have to be performed. It follows that the total number of computations performed by the algorithm before it stops is between $10(N + N^2 + \ldots + N^h)$ and $10(N + N^2 + \ldots + N^{h'})$, i.e between

$$10N\frac{N^h - 1}{N - 1} \approx 10N^h \qquad \text{and} \qquad 10N\frac{N^{h'} - 1}{N - 1} \approx 10N^{h'}.$$

Since one cannot have a complexity analysis for the greyscale photo copy algorithm and the random iteration algorithm (as in both cases one has to specify the number of iterations by trial and error), we cannot make a direct analytical comparison between the probabilistic power domain algorithm and the other two. However, on all inputs we have tried, the probabilistic domain algorithm is, often up to several times, faster than the random iteration algorithm in producing a good quality image.

3 Recurrent IFSs

Recurrent IFSs, which are generalisations of IFSs, were introduced in [4] and their application in image compression was further developed in [3, 6]. The flexibility of a recurrent IFS permits the construction of more general measures which do not have to exhibit the strict self-similarity of the IFS case.

Let X be the unit square as before, and suppose $\{f_1, f_2, \ldots, f_N\}$ an IFS acting on X. Let (p_{ij}) be an irreducible $N \times N$ row-stochastic matrix, i.e.

- $\sum_{j=1}^{N} p_{ij} = 1$ for all i,
- $p_{ij} \geq 0$ for all i, j, and
- for all i, j there exist $i_1, i_2, \ldots, i_n$ with $i_1 = i$ and $i_n = j$ such that $p_{i_1 i_2} p_{i_2 i_3} \cdots p_{i_{n-1} i_n} > 0$.

Then $\{f_j; p_{ij}; i, j = 1, 2, \ldots, N\}$ is called a recurrent IFS.

A recurrent IFS decodes N images on X which can be superimposed to produce a single image. Suppose we have N copies of the unit square X, which we denote by $X \times j$ with $j = 1, \ldots, N$. Then, for each $j = 1, \ldots, N$, we obtain an image μ_j^*. In vector notation, we can write an N-image by

$$\overline{\mu}^* = (\mu_j^*)_j = (\mu_1^*, \ldots, \mu_N^*),$$

which we call a *vector image.* We also let $z_{kl} \times j$ be the pixel z_{kl} in the copy $X \times j$. There are two decompression techniques in the state of art to generate $(\mu_j)_j$ [2].

3.1 The Recurrent Greyscale Photo Copy Algorithm

The recurrent greyscale photo copy algorithm generalises the greyscale photo copy algorithm. The idea is that starting with an N-image $\overline{\mu}$ we generate the sequence of N-images $\overline{\mu}$, $\overline{M}(\overline{\mu}$, $\overline{M}^2(\overline{\mu}$, $\overline{M}^3(\overline{\mu}$, ..., using the *recurrent Markov operator* $\overline{M}$, which takes a vector image $\overline{\mu}$ to a vector image $\overline{M}(\overline{\mu})$, whose jth component $(\overline{M}(\overline{\mu}))_j$ is defined by

$$(\overline{M}(\overline{\mu}))_j(z_{kl} \times j) = \sum_{i=1}^{N} p_{ij}\mu_i(f_j^{-1}(z_{kl} \times j)).$$

If n is sufficiently large, $\overline{M}^n(\overline{\mu})$ will be an approximation to $\overline{M}(\overline{\mu}^*)$. This method suffers from the same weaknesses as the greyscale photo copy algorithm described before.

3.2 Recurrent Random Iteration Algorithm

The recurrent random iteration algorithm generalises the random iteration algorithm. Take a copy $X \times i_0$ and a point $x_{i_0} \in X \times i_0$ in this copy. Now select i_1 from 1 to N such that the probability that j is chosen be $p_{i_0 j}$ and put $x_{i_1} = f_{i_1}(x_{i_0}) \in X \times i_1$. Therefore, we have moved from the initial point x_{i_0} in $X \times i_0$ to the point x_{i_1} in $X \times i_1$. Next, select i_2 from 1 to N such that the probability that j is chosen be $p_{i_1 j}$ and put $x_{i_2} = f_{i_2}(x_{i_1}) \in X \times i_2$. Repeat to obtain the sequence $x_{i_0}, x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_n}, \ldots$. If n is large, then those points of finite sequence $x_{i_0}, x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_n}$ which lie on $X \times j$ define the image μ_j^* for each $j = 1, \ldots, N$. More precisely, let $L(n, z_{kl} \times j)$ be the total number of the first n terms of the above sequence which are represented by the pixel $z_{kl} \times j$ in the copy $X \times j$. Then we have

$$\mu_j^*(z_{kl} \times j) = \lim_{n\to\infty} \frac{L(n, z_{kl} \times j)}{n+1}$$

i.e. for large n, the fraction

$$\frac{L(n, z_{kl} \times j)}{n+1}$$

is an approximation to $\mu_j^*(z_{kl} \times j)$. This method has the same shortcomings as the random iteration algorithm which we have already mentioned.

4 The Recurrent Probabilistic Domain Algorithm

We now present our first new algorithm. Let $\{f_j; p_{ij}; i, j = 1, 2, \ldots, N\}$ be a recurrent IFS on N copies $X \times j$ $(j = 1, \ldots, N)$ of the unit square X. We would like to find $\mu_j^*(z_{kl} \times j)$ for each $j = 1, \ldots, N$ and each pixel z_{kl}.

Since (p_{ij}) is an irreducible row-stochastic matrix, there exists a unique vector $(m_j)_j = (m_1, \ldots, m_N)$ with $m_j > 0$ $(1 \le j \le N)$ and $\sum_{j=1}^N m_j = 1$ which satisfies $m_j = \sum_{i=1}^N m_i p_{ij}$ [9, page 100]. The vector $(m_j)_j$ is called the *probability vector associated* with the transition matrix (p_{ij}) and can be found using the Gaussian elimination method. We then use the *inverse* transitional probability matrix [8, page

414] (q_{ij}) which is defined as follows:

$$q_{ij} = \frac{m_j}{m_i} p_{ji}. \tag{1}$$

Since $m_j > 0$ for $1 \le j \le N$ and therefore (q_{ij}) is well-defined; it is again row-stochastic, irreducible, and satisfies $\sum_{i=1}^{N} m_i q_{ij} = m_j$ for $j = 1, \ldots, N$.

We now consider the *recurrent IFS forest with transitional probabilities* in Figure 5, which consists of N trees with roots $X \times j$ for $j = 1, \ldots, N$. The recurrent probabilistic domain algorithm is based on this forest in the same way that the probabilistic domain algorithm was based on the IFS tree with probabilities of Figure 3.

The algorithm first computes the unique stationary initial distribution $(m_j)_j$, by solving the equations $m_j = \sum_{i=1}^{N} m_i p_{ij}$ for m_j $(1 \le j \le N)$ with the Gaussian elimination method, and determines the inverse transition probability matrix (q_{ij}) given by Equation (1). The number of arithmetic computations for this is of the order of N^3. Initially, the set $X \times \{j\}$ is given mass m_j, which is then distributed amongst the nodes of the jth tree according to the inverse transitional probability matrix (q_{tl}). In order to determine $\mu_j^*(z_{kl} \times j)$, where the number j and the pixel z_{kl} are fixed, the algorithm proceeds, similar to the probabilistic domain algorithm, to compute the sum of the weights $m_j q_{ji_2} \ldots q_{i_{n-1}i_n}$ of the leaves $f_j f_{i_2} \ldots f_{i_n} X$ of the jth tree which occupy the pixel $z_{kl} \times j$, i.e.

$$\mu_j^*(z_{kl} \times j) = \sum_{f_j f_{i_2} \cdots f_{i_n} X \subseteq z_{kl}} m_j q_{ji_2} \cdots q_{i_{n-1}i_n}.$$

The recurrent probabilistic domain algorithm, by efficiently traversing the finite forest, uses the least number of computations to make the best possible digitised approximation to the invariant vector measure of a recurrent IFS with probabilities.

The number of computations for the latter is of the order of N^h as before, where h is the height of the IFS tree. Therefore, the complexity of the algorithm is of the order of N^k where $k = \max(h, 3)$.

In Figure 6, the four images of a recurrent IFS with $N = 4$ is shown. The arrows indicate which p_{ij}'s are non-zero.

Compression is done using a generalisation of the collage technique described in Section 1.3. Details can be found in [2].

5 Vector Recurrent IFSs

Vector recurrent IFSs, which further generalise recurrent IFSs, are the basis of VRIFS™, which is a fractal image compressor. A vector recurrent IFS encodes and decodes N images μ_j^* on N *different* unit squares X_j for $j = 1, \ldots, N$. The precise definition is as follows.

Assume we have N unit squares, or computer screens, X_j for $j = 1, \ldots, N$. For each pair $i, j = 1, \ldots, N$, let $f_{ijn} : X_i \to X_j$, $1 \leq n \leq N_{ij}$, be a collection of contracting affine mappings, each with a probability weight $p_{ijn} > 0$, such that

$$\sum_{n=1}^{N_{ij}} p_{ijn} = 1,$$

for each pair $i, j = 1, \ldots, N$. Let p_{ij} be an irreducible row-stochastic $N \times N$ matrix. Then

$$(f_{ijn}; p_{ijn}; p_{ij}; 1 \leq i, j \leq N, 1 \leq n \leq N_{ij})$$

defines a *vector recurrent IFS* on the unit squares X_j for $j = 1, \ldots, N$. A recurrent IFS and an IFS are special cases of this. If $X_i = X$ for all $i = 1, \ldots, N$, $N_{ij} = 1$ and $f_{ijn} = f_j$ for all $i, j = 1, \ldots, N$, then we have a recurrent IFS $(f_j; p_{ij}; i, j = 1, \ldots, N)$. If, furthermore, $p_{ij} = p_j$ for all $i = 1, \ldots, N$, then we have, in effect, an IFS $(f_j; p_j; j = 1, \ldots, N)$.

5.1 Vector Recurrent Greyscale Photo Copy Algorithm

The recurrent greyscale photo copy algorithm can be generalised to the vector recurrent greyscale photo copy algorithm to generate the N images μ_j^* for $j = 1, \ldots, N$. As before, we will use the vector notation

$$\overline{\mu} = (\mu_j)_j = (\mu_1, \ldots, \mu_N).$$

Let $z_{kl} \times j$ be the pixel z_{kl} on the unit screen X_j. Any image μ_i on X_i induces an image $M_{ij}(\mu_i)$ on X_j defined by

$$M_{ij}(\mu_i)(z_{kl} \times j) = \sum_{n=1}^{N_{ij}} p_{ijn} \mu_i(f_{ijn}^{-1}(z_{kl} \times j)).$$

We now define the *vector recurrent Markov operator* $\overline{\overline{M}}$ which takes a vector image $\overline{\mu}$ to a vector image $\overline{\overline{M}}(\overline{\mu})$, whose jth component $(\overline{\overline{M}}(\overline{\mu}))_j$, is given by

$$(\overline{\overline{M}}(\overline{\mu}))_j(z_{kl} \times j) = \sum_{i=1}^{N} p_{ij} M_{ij}(\nu_i)(z_{kl} \times j).$$

One then starts with any vector image $\overline{\mu}$ and generates the sequence $\overline{\mu}$, $\overline{\overline{M}}(\overline{\mu})$, $\overline{\overline{M}}^2(\overline{\mu})$, …, $\overline{\overline{M}}^n(\overline{\mu})$, …. For large enough n, $\overline{\overline{M}}^n(\overline{\mu})$ approximates $\overline{\mu}^*$. This method has the same shortcomings as the greyscale photocopy algorithm.

It is also possible to obtain a vector recurrent random iteration algorithm to generate $\overline{\mu}^*$ which will share the weaknesses of the random iteration algorithm.

6 The Vector Recurrent Probabilistic Domain Algorithm

We will now present our second new algorithm. Let $(f_{ijn}; p_{ijn}; p_{ij}; 1 \leq i, j \leq N, 1 \leq n \leq N_{ij})$ be a vector recurrent IFS. We will now describe a finite algorithm to generate $\overline{\mu}^*$ on N digitised screens. Let $(m_j)_j$ be the unique probability vector associated with the irreducible row-stochastic matrix (p_{ij}) and let $q_{ij} = \frac{m_j}{m_i} p_{ji}$ be the inverse transitional matrix. We will consider a forest of N trees whose jth tree has a typical branch depicted in Figure 7.

Note that each node is a subset of its parent node. The root of the tree, at level 0, is X_j. On level one, a typical node is $f_{i_1 j n_1} X_{i_1}$ for $i_1 = 1, \ldots, N$ and $n_1 = 1, \ldots, N_{i_1 j}$. Similarly on level two we have the nodes $f_{i_1 j n_1} f_{i_2 i_1 n_2} X_{i_2}$ for $i_1, i_2 = 1, \ldots, N$, $n_1 = 1, \ldots, N_{i_1 j}$ and $n_2 = 1, \ldots, N_{i_2 i_1}$. On level t, we have the nodes $f_{i_1 j n_1} f_{i_2 i_1 n_2} f_{i_3 i_2 n_3} \cdots f_{i_{t-1} i_{t-2} n_{t-1}} f_{i_t i_{t-1} n_t} X_{i_t}$ for $i_1, \ldots, i_t = 1, \ldots, N$, $n_1 = 1, \ldots, N_{i_1 j}$, $\ldots$, $n_t = 1, \ldots, N_{i_t i_{t-1}}$. Each of the nodes of jth tree is given a weight. The root node X_j is given weight m_j, the node $f_{i_1 j n_1} X_{i_1}$ on level one weight $m_j q_{j i_1} p_{i_1 j n_1}$, the node $f_{i_1 j n_1} f_{i_2 i_1 n_2} X_{i_2}$ on level two weight $m_j q_{j i_1} q_{i_1 i_2} p_{i_1 j n_1} p_{i_2 i_1 n_2}$, and generally the node $f_{i_1 j n_1} f_{i_2 i_1 n_2} f_{i_3 i_2 n_3} \cdots f_{i_{t-1} i_{t-2} n_{t-1}} f_{i_t i_{t-1} n_t} X_{i_t}$ on level t is given the weight $m_j q_{j i_1} q_{i_1 i_2} \cdots q_{i_{t-1} i_t} p_{i_1 j n_1} p_{i_2 i_1 n_2} \cdots p_{i_t i_{t-1} n_t}$. These weights are shown on Figure 8.

The vector recurrent probabilistic domain algorithm finds the leaves of the jth tree (for each $j = 1, \ldots, N$) and for each pixel $z_{kl} \times j$ sums up the weights of those leaves which are occupied by the pixel. In other words,

$$\mu_j^*(z_{kl} \times j) = \sum_{f_{i_1 j n_1} f_{i_2 i_1 n_2} \cdots f_{i_t i_{t-1} n_t} X_{i_t} \subseteq z_{kl}} m_j q_{j i_1} q_{i_1 i_2} \cdots q_{i_{t-1} i_t} p_{i_1 j n_1} p_{i_2 i_1 n_2} \cdots p_{i_t i_{t-1} n_t}.$$

Like the probabilistic power domain algorithm and its recurrent version, the vecrtor recurrent probabilistic domain algorithm uses the least number of computations to make the best possible digitised approximation to the invariant vector measure of a vector recurrent IFS with probabilities. A simple calculation will determine the maximum number of arithmetic operations in the algorithm. Let s_{ijn} be the contractivity of f_{ijn} and put $s = \max_{ijn} s_{ijn}$. Then the height h of forest is $h = [-\frac{\log r}{\log s}] + 1$, where r is the resolution of the screen. It can be shown that the number of arithmetic computations needed in the algorithm is less than $12 N^{h+1} (\prod_{i,j=1}^{N} N_{ij})^h$. In practice N is chosen small and for most pairs i, j we have $N_{ij} = 1$. A selection of images encoded by vector recurrent IFSs can be found in [2, 5].

References

[1] M. F. Barnsley. *Fractals Everywhere.* Academic Press, 1988.

[2] M. F. Barnsley. *Fractals Everywhere.* Academic Press, second edition, 1993.

[3] M. F. Barnsley, M. A. Berger, and H. M. Soner. Mixing Markov chains and their images. *Prob. Eng. Inf. Sci.*, 2:387–414, 1988.

[4] M. F. Barnsley, J. H. Elton, and D. P. Hardin. Recurrent iterated function systems. *Constructive Approximation*, 5:3–31, 1989.

[5] M. F. Barnsley and L. P. Hurd. *Fractal Image Compression.* AK Peters, Ltd, 1993.

[6] M. A. Berger. Images generated by orbits of 2-D Markov chains. *Chance*, 2(2):18–28, 1989.

[7] A. Edalat. Power domain algorithms for fractal image decompression. Technical Report Doc 93/44, Department of Computing, Imperial College, 1993. British Patent Application no. P17444GB filed by Imperial College on 17-9-93.

[8] W. Feller. *An Introduction to Probability Theory and Its Applications.* Wiley, London, 3rd edition, 1968.

[9] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* D. Van Nostrand, 1960.

Figure 1

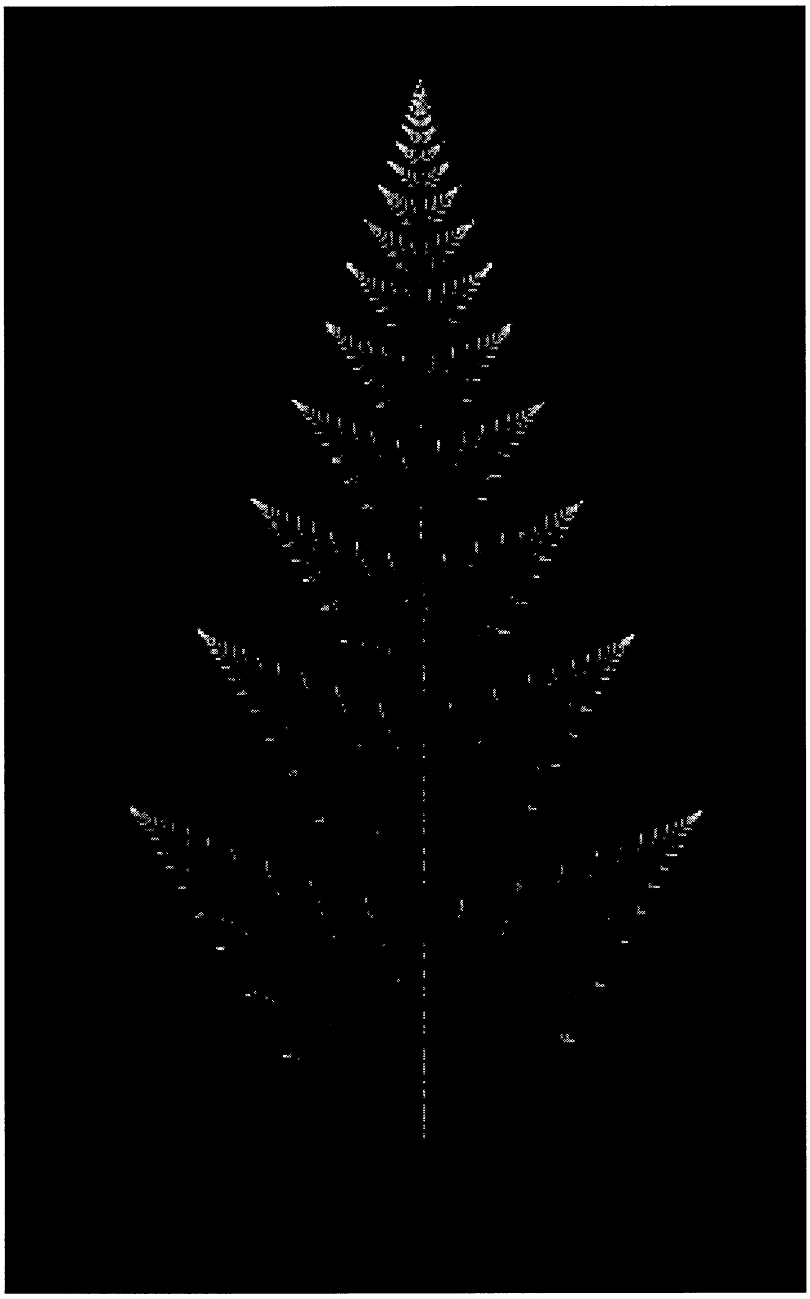

Figure 1. The greyscale image encoded by the IFS with probabilities given in Table 1.

Figure 2

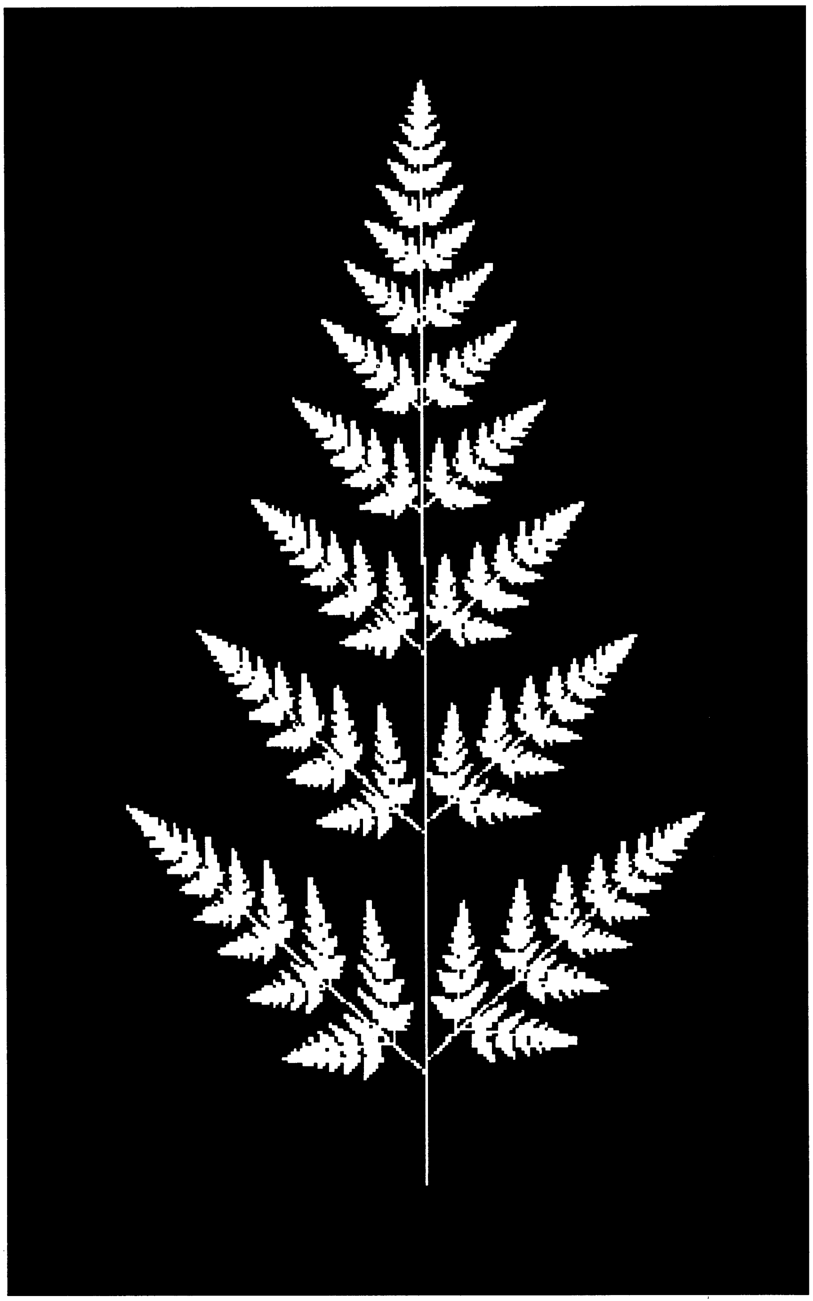

Figure 2. The support of the image in Figure 1.

Figure 3

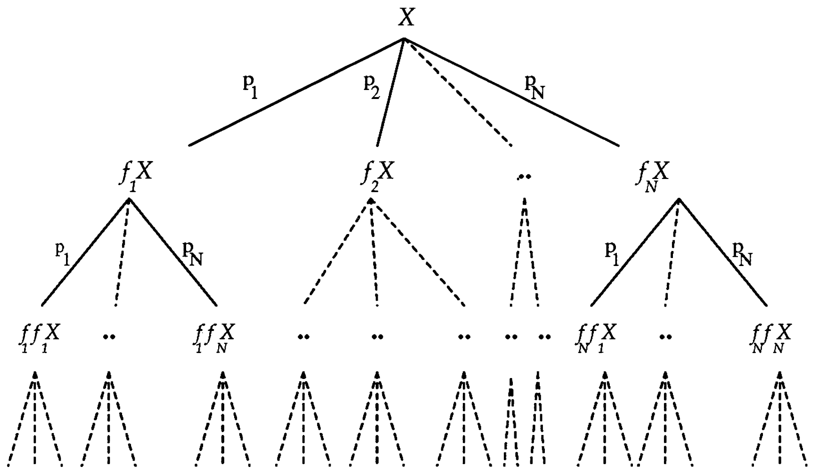

Figure 3. The IFS tree with transitional probabilities.

Figure 4

$$\begin{tikzcd}
X \arrow[d, no head] \\
f_{i_1}X \arrow[d, no head] \\
f_{i_1}f_{i_2}X \arrow[d, no head] \\
\cdot \arrow[d, no head] \\
f_{i_1}f_{i_2}\ldots f_{i_n}X
\end{tikzcd}$$

Figure 4. A complete branch of the IFS tree with transitional probabilities.

Figure 5

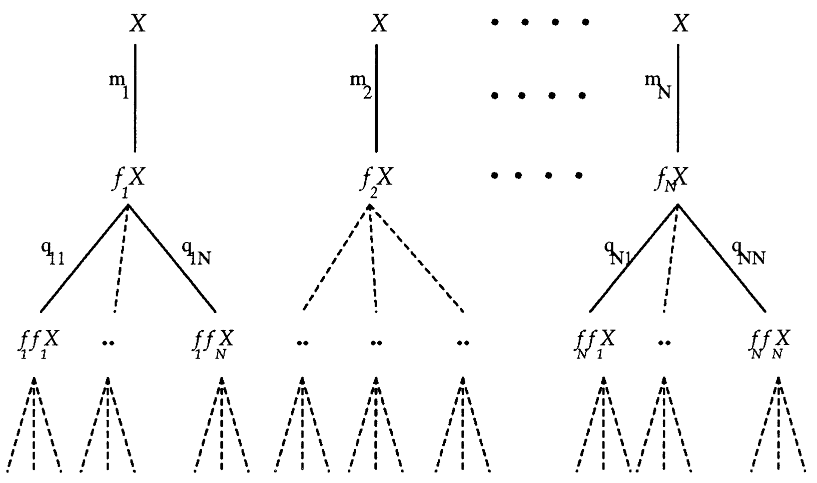

Figure 5. The recurrent IFS forest with transitional probabilities.

Figure 6

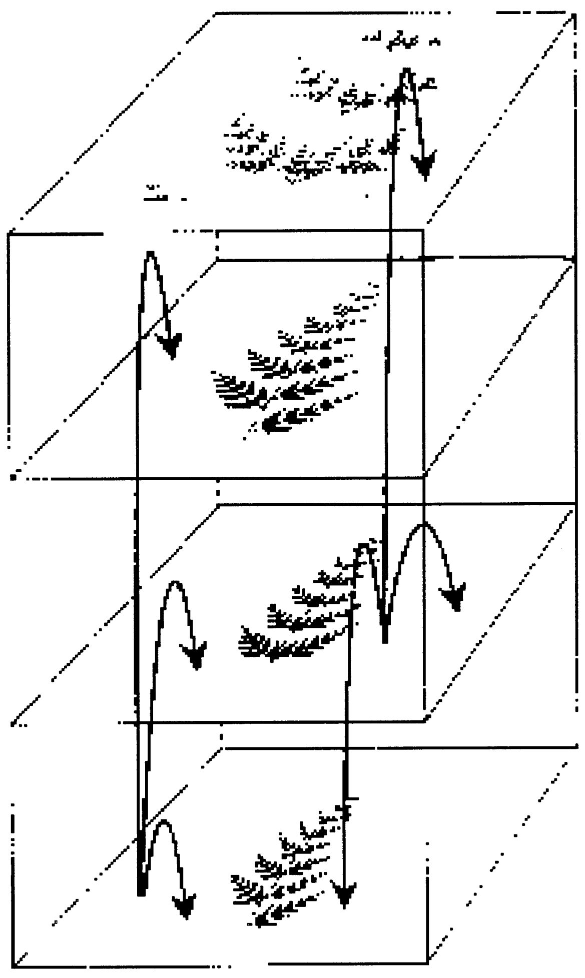

Figure 6. A vector of four images generated by a recurrent IFS.

Figure 7

$$\begin{tikzcd}
X_j \arrow[d, dash] \\
f_{i_1 j n_1} X_{i_1} \arrow[d, dash] \\
f_{i_1 j n_1} f_{i_2 i_1 n_2} X_{i_2} \arrow[d, dash] \\
f_{i_1 j n_1} f_{i_2 i_1 n_2} f_{i_3 i_2 n_3} X_{i_3} \arrow[d, dash] \\
\cdots \arrow[d, dash] \\
f_{i_1 j n_1} f_{i_2 i_1 n_2} f_{i_3 i_2 n_3} \cdots f_{i_{t-1} i_{t-2} n_{t-1}} f_{i_t i_{t-1} n_t} X_{i_t} \arrow[d, dash] \\
\cdots
\end{tikzcd}$$

Figure 7. A branch of the jth tree of the forest of a vector recurrent IFS.

Figure 8

$$\begin{tikzcd}
m_j \arrow[d, dash] \\
m_j q_{ji_1} p_{i_1 j n_1} \arrow[d, dash] \\
m_j q_{ji_1} q_{i_1 i_2} p_{i_1 j n_1} p_{i_2 i_1 n_2} \arrow[d, dash] \\
m_j q_{ji_1} q_{i_1 i_2} q_{i_2 i_3} p_{i_1 j n_1} p_{i_2 i_1 n_2} p_{i_3 i_2 n_3} \arrow[d, dash] \\
\cdots \arrow[d, dash] \\
m_j q_{ji_1} q_{i_1 i_2} \cdots q_{i_{t-1} i_t} p_{i_1 j n_1} p_{i_2 i_1 n_2} \cdots p_{i_t i_{t-1} n_t} \arrow[d, dash] \\
\cdots
\end{tikzcd}$$

Figure 8. The weights given to the nodes of the branch in Figure 7 of the jth tree of the forest of a vector recurrent IFS.