Construction of Fractal Objects with Iterated Function Systems

Nicolas Dutil

1-Introduction:

Suppose you had no ideas what fractals were, and you were asked to model objects like clouds, grass, plants, etc.. What else could you use to define the geometries of these objects? Although polynomials can easily define objects with smooth geometry, they are pretty useless if you want to model complex objects like grass or plants, since those possess infinitely non smooth, highly structured geometries. In the early 1980s, mathematicians have become concerned with non-smooth sets, that is, sets where the method of classical calculus couldn't be applied. It was the fundamental work of Mandelbrot that opened up a new way to model natural phenomena. What is a fractal? Many definitions exists, and mathematicians have not yet agreed on one. Benoit Mandelbrot refers to the word "fractals" as objects who possess self-similarity. For the rest of this paper, we will adopt this as our definition of a fractal.

It is important for the reader to understand that fractals don't really exists in nature. They are used as a model for studying complex natural phenomena. If you look closer and closer at a leaf of a tree, you will eventually arrive at the cells that constitute the leaf. This property of self-similarity that natural objects possess is maintained as long as we don't look to close at them. However, with fractals, there is no limit as too how far we can look before this property disappears. As we keep looking deeper and deeper in the structure, additional details will be revealed and you will probably have a strong feeling of déjà vu.

This brings us to our main discussion: how can we generate these fractals? Besides using IFS, which is our main topic, other techniques exists. Lindenmayer systems, or L-systems for short, where invented in 1968 by Aristid Lindenmayer [Lind68] as a way to model biological growth. The technique works as follows : we have an alphabet V consisting of various symbols and an initiator (a string of symbols from V) to which we apply a list of production rules. By applying all the rules, we create a new word. If we repeat this scheme over and over again, a global pattern will emerge. This global process will often have the self-similarity property that we were talking about. This technique is very similar to an iterated function system. Both of them are dynamical systems (defined later).

An Iterated Function Systems is a set of contraction mappings $W=\{w_1,w_2,...w_n\}$ acting on a space X. Associated with this set of mappings W, is a set of probabilities $P=\{P_1,P_2,...,P_n\}$. As we will see, these probabilities are used to generate a random walk in the space X. If we start with any point in X and apply these maps iteratively, we will come arbitrarily close to a set of points A in X called the attractor of the IFS. These attractors are very often fractal. (for the most part, we will assume attractors are fractal sets, and thus, use the words interchangeably) This forms the basis for creating an algorithm that will approximate the attractor of an IFS. We sometime call sets $\{W^k(A)\}$ whose limits are fractals, pre-fractals. These are sets the algorithm will be able to generate . Increasing the number of times we apply the maps will give us a more accurate picture of what the attractor looks like.

2-Mathematical Preliminaries

In order to understand what iterated function systems are and why the random iteration algorithm works, we need to be familiar with some mathematical concepts. Readers with a good background in analysis and algebra can move on to the next section.

For the rest of this section, a space X is simply a set of elements (points).

I- Metric Spaces

definition: A space X with a real-valued function d: $X \times X \rightarrow \Re$ is called a metric space (X,d) if d possess the following properties:

1.$d(x,y) \geq 0$ for $\forall\ x,y \in X$
2.$d(x,y) = d(y,x)\ \forall\ x,y \in X$
3.$d(x,y) \leq d(x,z) + d(z,y) \forall\ x,y,z \in X$. (triangle inequality)

For instance, $\Re$ with $d = |x-y|$ is a metric space. $\Re^2$ with the usual euclidian distance is also a metric space.

Open Sets

definition : A subset S of the metric space (X,d) is open if, for each point $x \in S$, we can find a $r > 0$ so that $\{y \in X : d(x,y) < r\}$ is contained in S.

Closed Sets

definition:A subset S of the metric space (X,d) is closed if,whenever a sequence$\{x_n\}$contained in S converges to a limit $x \in X$,then in fact this limit $x \in S$.

Bounded Sets

definition: A subset S of the metric space (X,d) is bounded if we can find an $x \in X$ and an $M \in \Re > 0$ so that $d(a,x) \leq M\ \forall\ a \in S$.

Cauchy Sequence

definition: A sequence $\{x_n\}$ in X is called a Cauchy sequence if given $\varepsilon > 0$, we can find an $N \in N > 0$ such that $d(x_n,x_m) < \varepsilon\ \forall\ n,m > N$

Note: A cauchy sequence need not have a limit in X. This stimulate the next definition.

Complete Metric Space

definition: A metric space (X,d) is complete if every Cauchy sequences in X converges in X.

Compact Sets

definition: A subset S of the metric space (X,d) is compact if every sequences in S has a subsequence which converges in S.

Since we are mostly concerned with metric spaces where the underlying space is $\Re^n$ or C^n , we can state the following :

theorem: If a subset $S \subset \Re^n$ (or C^n) is closed and bounded, then it is compact.

Now that we know what a compact set is, we can define the following space:

definition: Let X be a complete metric space.Then H(X) consists of the non-empty compact subsets of X.

To make H(X) into a metric space, we must find a real valued function h : H(X) x H(X) $\rightarrow \Re$ with the properties enumerated before. To construct this metric, we need to know what a δ-parallel body A_δ of a set A is :

definition : Let (X,d) be a metric space. A δ-parallel body A_δ of a set A is the set of points in X within distance d of A : $\{x \in X : d(x,y) \leq d \text{ for some } y \in A\}$

definition : Let $A,B \in H(X)$, and h(A,B) = infinimum{ $d : A \subset B_\delta$ and $B \subset A_\delta$ } . We call h(A,B) the Hausdorff metric.

The Hausdorff metric tells us how close two sets are to each other. If A and B are very close, then d will be small, and thus h(A,B) will be small also.
Before we can begin talking about iterated function systems,there's two more interesting mathematical notions we must be aware of.

II- Dynamical Systems

definition : A dynamical system is a transformation $S : X \rightarrow X$ on a metric space (X,d). It is denoted by {X ;S}.

definition : Let {X ;S} be a dynamical systems. A point $x \in X$ for which S(x) = x is called a fixed point of {X ;S}.

III- Contraction Mappings

definition : Let $S : X \rightarrow X$ be a transformation on the metric space (X,d). S is a contraction if $\exists\ s \in \Re$with $0 \leq s < 1$ such that $d(S(x),S(y)) \leq sd(x,y)\ \forall\ x,y \in X$. Any such number s is called a contractivity factor of S.

The following theorem will be very important for later on.
Contraction Theorem : Let $S : X \rightarrow X$ be a contraction on a complete metric space (X,d). Then S possess exactly one fixed point $x_s \in X$ and moreover for any point $x \in X$, the sequence $\{S^k(x): n = 0,1,2.....\}$ converges to x_s. That is, $\lim_{n\rightarrow\inf} S^k(x) = x_s$.
The proof can be found in [Barn93]

3-Iterated Function Systems

[Barn93] defines an iterated function systems in the following way :

definition : A (hyperbolic) iterated function system consists of a complete metric space (X,d) together with a finite set of contraction mappings $w_n : X \rightarrow X$, with respective contractivity factor s_n , for n = 1,2,....N. The abbreviation "IFS" is used for "iterated function systems". The notation for the IFS just announced is $\{X ; w_n : n = 1,2,...N\}$ and its contractivity factor is $s = \max\{s_n : n = 1,2,...N\}$.

The following theorem is extremely important and suggest an algorithm for computing the pre-fractals.

Theorem 3.1:
Let $\{H(X) ; w_1, w_2, ... w_N\}$ be an IFS with contractivity factor s. We define $W(B) = union(wi(B))$ for $i=1..N$, $B \in H(X)$. Then the following can be said :

a) W(B) is a contraction mapping with contractivity factor s

b) Its unique fixed point $A \in H(X)$ obeys $A = W(A) = union(wi(A))$ for $i = 1..N$
given by $A = \lim_{k \to infinity} W^k(B) = intersection(W^k(B))$ $k=1..infinity$, for any $B \in H(X)$.

The proof can be found in [Barn93].
definition : The fixed point $A \in H(X)$ as described in the theorem is called the attractor of the IFS.

Like we've said before, attractors of IFS are very often fractals. So , this theorem gives us a way to compute pre-fractals. All we have to do is take any compact subset A (e.g a square if $X = \Re^2$ would work} of X, and apply iteratively the contraction mapping W on A. As $k \to inf$, the set $\{ W^k(B) \}$ for any $B \in H(X)$ will give us better and better approximations to the attractor of the IFS.

The Deterministic Algorithm [Barn93]:
Let $\{X ; w_1, w_2, ... w_n)$ be an IFS. Choose any compact set $Bo \subset \Re^2$. Then compute successively

$B_{n+1} = union(w_i(B_n))$ for $i = 1..n$

This sequence $\{ B_n \}$ will converge to the attractor of the IFS.

4-The Random Iteration Algorithm

The Random Iteration Algorithm [Barn93]: Let $\{X ; w_1 , w_2 , ... w_N\}$ be an IFS, where probability $P_i > 0$ has been assigned to w_i for $i = 1,2,....n$, where $\Sigma P_i = 1$. Choose $x_o \in X$ and then choose recursively and independently

$x_n \in \{ w_1(x_{n-1}), w_2(x_{n-1}), ..., w_N(x_{n-1}) \}$ for $n = 1,2,3....$

where the probability of the event $x_n = w_i(x_{n-1})$ is Pi. Thus construct a sequence $\{x_n : n = 0,1,2....\} \subset X$. This sequence of points will come arbitrarely close to every point in the attractor of the IFS.

The amazing thing this algorithm tells us is that we don't even have to think about choosing a compact subset B of X and apply iteratively the contraction mapping W to this subset. The process of applying W to B is computationally expensive compare to applying one of the maps wi to a single point $x \in X$. So, given a k , the sequence $\{W^k(B)\}$ will take much longer to generate than the sequence $\{x_n : n = 0,1,2....k\}$. However, we immediately observed the familiar trade-off between time and space. There is much more points generated by the sequence $\{W^k(B)\}$ than the sequence $\{x_n : n = 0,1,2...k\}$ for the same

value of k. But the simplicity of the random iteration algorithm makes it an attractive solution for generating approximations of a fractal set.

Now, let's see why the algorithm works. Suppose we have a set $B \in H(X)$. Then for each k, we have

$W^k(B) = union(w_{i1}(w_{i2}(....(w_{ik}(B)))))$ where the union is over the set J_k of k-term sequences

$(i_1,i_2,...i_k)$ with $1 \le ij \le n$.

What we did is simply expand the set $W^k(B)$:

$W^k(B) = union(w_i(W^{k-1}(B)))$ for i=1..N = $union(w_i(union(w_j(W^{k-2}(B)))$ for j=1..N) for i = 1..N = $union$ $(w_i(w_j(W^{k-2}(B))))$ for i=1..N

We keep expanding the set $W^k(B)$ until it's the union of sets obtained by applying iteratively k compositions $w_{i1},w_{i2},w_{i3},...w_{ik}$ to the set B. It is not unlikely that some sets might contain the same elements.

Now, suppose that $w_i(B)$ is contained in B for every i. Then if $x \in A$ (the attractor), it follows from theorem 3.1 (statement b) that there is a (not necessarily unique) sequence $(i_1,i_2,...,i_k)$ such that $x \in w_{i1}(w_{i2}(...(w_{ik}(B))))$ for all k. A sequence of this kind is often called the address of a point. So $A = union\{x_{i1,i2,...,..}\}$ and $x_{i1,i2,...} = intersection(w_{i1}(w_{i2}(...w_{ik}(B))))$ k = 1...inf . This is the key to understanding why the random iteration algorithm works.

Let x_o Î B and $x_k = w_{ik}(w_{ik-1}(...w_{i1}(B)))$ after k steps. Then the address of xk will agree with one point in A up to the k-th term in the sequence. If we let k get bigger and bigger, xk will get closer and closer to a point in A. At the same time, the sequence $\{x_n\}$ gets closer and closer to all points in the attractor. Suppose after n iterations, we have a value xn. Then if we iterated k-th times more, $x_{n+k} = w_{i(k+n)}(w_{i(k+n-1)}....(w_{in}(x_n)))$ will be close to a point in A with an address beginning by $(i_n,i_{n+1},...i_{k+n})$. Thus, the more we iterate the more close we are to each point in A. If let B be the entire monitor, then B is obviously compact , and so any point chosen in this area will work. If we fall in A after a certain number of iterations, that is, $x_k = w_{ik}(x_{k-1})$ Î A, then we will never leave the attractor after that. We will simply hop from one point in the attractor to another. Since there are infinitely many points in A, we can only display an approximation of what the attractor looks like. The probabilities associated with each maps will determine the density of each portions of the attractor. If a map has a high probability of being chosen, then the region where the contraction mappings maps the point too will be more dense than the other regions.

5- Implementation

5.1 Restrictions

The definition of the IFS and the description of the random iteration algorithm are quite generic. We will restrict ourselves to IFS where the space is either $\Re,\Re^2,\Re^3$ or the complex numbers C.The contraction mappings $\{w_1,w_2,....w_n\}$ used for generating the pictures are affine transformations. An affine

transformation in $\Re^2$ is of the following form :

T[x,y] = A[x y]T + [e f]T , where A = $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$

5.2 The algorithm

As we've said before, the random iteration algorithm presented earlier was implemented. The initial point $x_o \in X$ is always the origin of the space : $x_o = (0,0)$ if $X \in \Re^2$ or $x_0 = 0 + 0i = (0,0)$ if $X \in C$ (complex numbers) . Similarly, If $X \in \Re^3$, then $x_o = (0,0,0)$. If we want to see the path the sequence $\{x_n\}$ takes before it falls in the attractor A of the IFS,we can choose an initial point far from the attractor, which usually lie in the unit square [0,0]x[1,1] if $X \in \Re^2$. if $X \in \Re^2$, choosing a point near the edges of the screen will work fine in most cases.

Since most computers don't have a built-in function that generates random numbers according to a given probability density function (e.g normal distribution) , the following algorithm was used to select the maps according to their probabilities P_i of being chosen :

Let $\{w_1,w_2,....,w_N\}$ be a set of contraction mappings and $\{P_1,P_2,....P_N\}$ the associated set of probabilities P_i, where P_i means the probability of choosing w_i. Let r be a number generated using a uniform distribution. r is chosen in the interval [0,1]. The algorithm is then :

```
let sum = p1 and let i = 1
while ( r > sum)
{
sum = sum + pi
i = i + 1
}
use map wi
```

The coloring scheme we used attibutes a color to each contraction mapping wi. The user can specify the color they want for each mappings, but the goal here is to identify what each mapping accomplishes, so it is best to associate a unique color to each mapping wi.

5.2 Complexity of the algorithm :

The complexity of the algorithm depends on the number of points in the sequence $\{x_n\}$ we wish to generate and the number of maps wi. But the former largely suppress the latter. So, the algorithm is Θ(number of iterations).

5.3 Results

Figure 5.3.1
A Spiral

Figure 5.3.2
A close-up of a 3-D fern

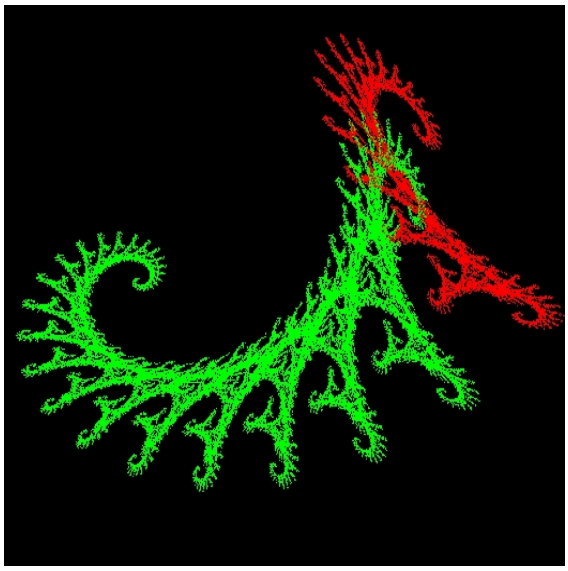

Figure 5.3.3
Dragon

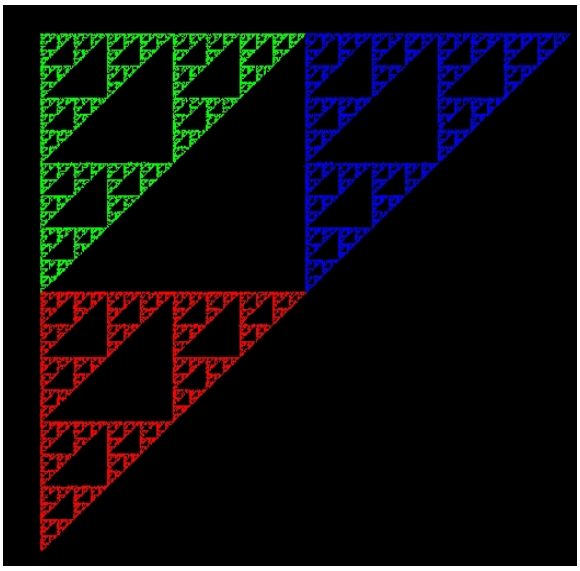

Figure 5.3.4
Sierpinski's triangle

Fractal Set	Number of Iterations	Scale Factor	Time taken to generate the picture(millisec)
Sierpinski's triangle	4400	0.024	1047.85
Dragon Curve	82000	0.202	1079.57
Spiral	219000	0.517	1214.45
3-D Fern (Close-Up)	343000	2.035	1447.575

Table 5.3

Table 5.3 gives us some quantitive informations about the pictures shown above. All pictures were generated using the random iteration algorithm. The time taken to generate a picture is not very useful by itself. As you can see by looking at the pictures, we tried to make every picture look equally good so that we can compare objectively the time taken for generating approximations to different fractals. The number of iterations (i.e number of points in the sequence $\{x_n\}$) is shown in table 5.3 for each of the pictures. The scale factor tells us by how much we contracted or expanded the original picture. The scaling is applied after we have multiplied the points $x \in X$ by the affine transformation wi.

The dragon and the sierpinski's triangle took about the same time to generate. The spiral took a little bit longer , and finally the three dimensional fern was the longest to compute. This is attributable to the fact that it's the only figure shown here that's in three dimensions. So some details of the picture are hidden by pixels closer to our viewpoint.

Finally, we can observe the coloring scheme at its work in these picture. We can clearly see the region of space a mapping wi maps points $x \in X$ into. The mappings used to generate the sierpinski's triangle seems to map points into different region of space, while the two mappings used to create the dragon seems to share a region in space. We emphasize the word seems here, since we should not forget that rounding errors occur when moving points belonging in a continuous 2-D space onto a discrete 2-D space (the screen).

5.4 Improvements

We talked about how the deterministic algorithm takes less iterations but more time for each iteration to compute the next set B_n. The random iteration algorithm is faster for one iteration, but it needs a lot more iterations for generating the same picture. A good compromise would be to start with more than one point $x_o \in X$ and build a sequence $\{x_k\}$ for each initial points $\{x_0,x_1,...x_m\} \in X$. Another improvement would be to use the scale factor as an estimate for how much iterations we will need to produce each time the picture is expanded (more points needed) or contracted (less points needed).

---

6. Conclusion

6.1 Critique

The random iteration algorithm is a pretty fast way to generate complex objects. It lends itself perfectly to objects which are self-similar since we can easily find contraction mappings wi that recreate approximately the original object. One way to achieve fractal compression is based on this idea. The detail at which we can see objects generated by an IFS is only limited by the amount of computer memory that's available. By looking again at the pictures in section 5.3, it would seem that we have applied some sort of shading technique to these pictures. However, this is not the case. The reason is that pixels with non-zero intensities in areas (near the set's boundaries) where there are surrounded by lots of black pixels appear to have less intensities than they truely have. In contrast, dense areas (lots of non-black pixels) appear brighter.

This bring the issue of how we could apply conventional shading techniques to these pictures. Applying the global illumination model for every point in the sequence $\{x_n\}$ would be extremely time consuming.

[Reev85] describes a method for shading objects,made using particle systems, that could be used here also at some extent. An other property of the random iteration algorithm that might be undesirable in some cases is that every time we run the algorithm, we will never quite get the same picture twice. So if we want to reproduce exactly an image that we generated a while ago, chances are, we will never be able to do so.This is due to the fact that we don't always choose the same mapping w_i for generating the point $x_n = w_i(x_{n-1})$ in the sequence $\{x_n\}$. However, in some cases we might like the fact that each mapping wi has a probability P_i of being chosen. If we are more interested in a certain region of the fractal, we might attribute higher probabilities $\{P_1,P_2,..P_k\}$ to the mappings $\{w_1,w_2,..w_k\}$ which maps points $x \in X$ in that region.

6.2 Summary

Two algorithms were presented : the deterministic algorithm and the random iteration algorithm. Although not implemented, the deterministic algorithm was discussed because it was a direct consequence of theorem 3.1. It turns out there is a simpler way of generating the attractor of the IFS. The random iteration algorithm takes as argument one point x_o in X,a set $\{w1,w2,...w_N\}$ of contraction mappings, with an associated probability set $\{P1,P2,...P_N\}$, and then generate a sequence $\{x_n\}$ where $x_n = w_i(x_{n-1})$, with the probability of chosing wi equal to P_i. As $n \rightarrow inf$, $\{x_n\}$ clusters around a set called the attractor of the IFS. The attractor is very often a fractal set. A fractal has the property of being self-similar.

6.3 References

Principal Reference :

S.Demko,L.Hodges, B.Naylor, "Construction of Fractal Objects with Iterated Function Systems", SIGGRAPH volume #3, July 22-26 1985, p271-276.

Other references :

[Barn93]
M.Barnsley, "Fractals Everywhere", Academic Press,1993.

[Lind68]
Lindenmayer, A. "Mathematical Models for Cellular Interactions in Development, Parts I and II", J.Theor. Biol., 18,1968,280-315

URLs

http://www.math.unl.edu/~webnotes/home/home.htm , "Analysis WebNotes" by John Lindsay
http://www.math.okstate.edu/mathdept/dynamics/lecnotes/node1.html, "Dynamical Systems and Fractals Lecture Notes" by David. J. Wright.

Go Visit the IFS Gallery