Math 7H | Professor: Padraic Bartlett

Lecture 5: Drawing Fractals

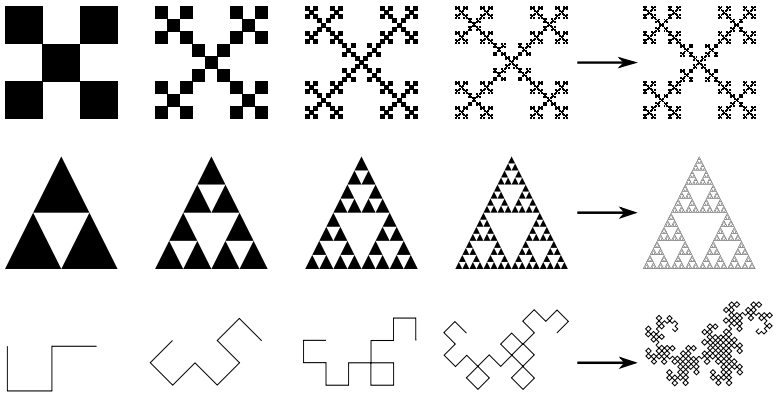
Week 5 | *UCSB 2015*

(This talk draws heavily on the Hasse Prize-winning paper "Fractals, Graphs and Fields" by F. Mendivil. It's a fantastic read, and is remarkably accessible; check it out!)

1 Fractals

1.1 Definitions and Examples

Intuitively, a "fractal" is a shape or set in mathematics that is "self-similar:" in other words, fractals are sets that "look the same" when we zoom in on them.



From top to bottom: the Vicsek fractal, the Sierpinski triangle, and the dragon curve. The fractal is drawn at right, with the first few steps of the "pattern" used to generate each fractal drawn at left.

"Looks the same," however, is not a particularly rigorous definition. In this talk, we are going to consider one specific notion for fractals: the concept of an **iterated function system.**

Before defining an iterated function system (IFS for short), we first give an example to illustrate the ideas here.

Example. Take the unit square $X = [0,1]^2 \subset \mathbb{R}^2$, and the following three functions $X \to X$:

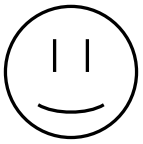
$$w_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right)$$
$$w_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right)$$
$$w_3(x,y) = \left(\frac{x}{2}, \frac{y}{2} + \frac{1}{2}\right).$$

Given any subset $A \subset X$, set $w_i(A) = \{(w_i(x, y) \mid (x, y) \in A\}$.

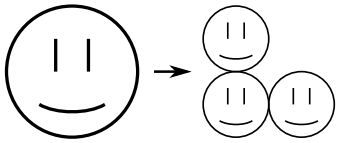
With all of this set up, we can finally define the **iterated function system** W on X associated to $\{w_1, w_2, w_3\}$ as follows: given any set A, define[1]

$$W(A) = w_1(A) \cup w_2(A) \cup w_3(A).$$

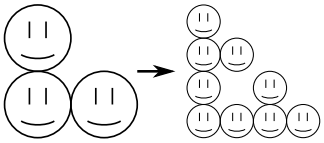
In words: the map W takes in any set A and outputs the three sets $w_1(A), w_2(A)$ and $w_3(A)$; each of which are $1/4$-scale copies of A in the top-left, bottom-left, and bottom-right quadrants of $[0, 1]^2$. To illustrate this idea, consider the following subset A of $[0, 1]^2$:



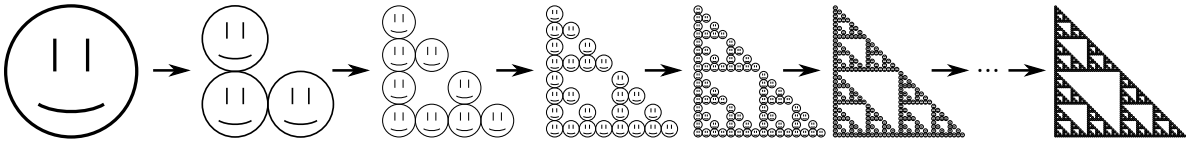
When we apply W to this "smiling-face" set, we get the following object:



We can apply W again to get the following:



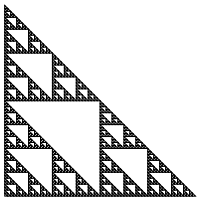
If we keep applying W, we get the following pattern:



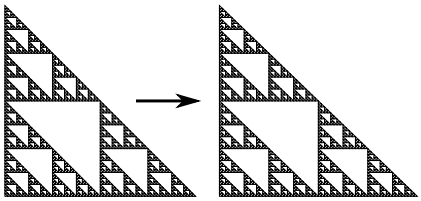
We call the set at the far right above the **limit set** L of our iterated function system W. Notice two things about this limit set:

- In the limit, it didn't matter what our initial set A was! We started with a smiley-face for A, but in the limit we can't tell the difference between this shape and (say) a frowning face, or really anything else.
- If we were to **start** with our initial set A equal to our limit set L: i.e. suppose that we let A be the following set:

[1] If you haven't seen this notation before: we write $A \cup B$ to denote the set that consists of all of the elements that are in either A or B (or both!) We call this the **union** of the two sets A and B.



Then, notice that applying W to this set doesn't change anything: i.e. we have



So our limit set, in a sense, is **fixed** under the map W! We will call such sets **fixed** sets in general.

In general, we define an iterated function system as follows:

Definition. A **iterated function system** consists of the following objects:

- A space X that we want to iterate our maps on. (Usually this will be $[0,1]^2$, though other spaces like $\mathbb{R}^2$ or $[0,1]^3$ are certainly used as well!)
- A collection of maps $\{w_1, w_2, \ldots w_k\}$, such that each w_i is a map $X \to X$.

With these two objects defined, we say that the IFS associated to $X, \{w_1, \ldots w_n\}$ is the following map W, defined on any subset A of X as follows:

$$W(A) = w_1(A) \cup w_2(A) \cup \ldots \cup w_n(A).$$

Definition. For any subset X of $\mathbb{R}^n$, a map $w : X \to X$ is called a **contraction** with contraction factor s if

- For any two points $a, b \in X$, we have that the distance between $w(a)$ and $w(b)$, $d(w(a), w(b))$, is at most $s \cdot d(a, b)$.
- s is the smallest real number such that the above point is true.

In other words, the map w "shrinks" space by a factor of s. For instance, all three of the maps w_1, w_2, w_3 from our example above are contractions with contraction factor $1/2$.

The reason that we like IFS is that in general, they always exhibit the "limit/fixed set" phenomena that we noticed in our example above! Specifically, we have the following theorem:

Theorem. (Banach Fixed-Point Theorem.) Suppose that W is an IFS with associated maps $\{w_1, w_2, \ldots w_n\}$, with the following property: there is some $s < 1$ such that all of the maps w_i are contractions with contraction factor no greater than s.

Then W has a unique limit set; in other words, there is some set L such that for any[2] subset $A \subset X$, we have

$$\lim_{n\to\infty} W^n(A) = L.$$

Moreover, this set L is the unique "fixed set" for the map W: that is, L is the only set such that

$$W(L) = L.$$

We're not going to prove this in this class (though we may come back for it later!) Instead, we're going to assume it's true (it certainly worked for our example above!) and focus instead on how we can **work** with an IFS!

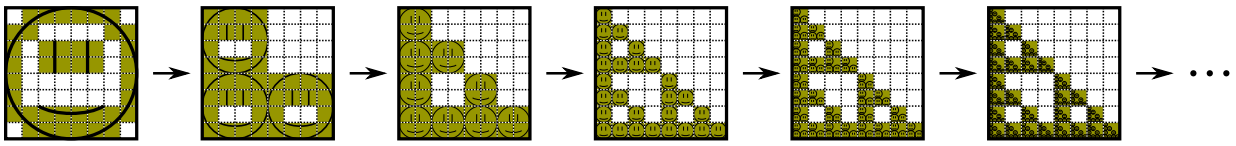
1.2 How to Draw a Fractal: A Deterministic Approach

In particular, I want to consider the following problem: suppose we are given an IFS, and we want to draw its limit set. How can we do this?

To make this problem more tractable, let's suppose that we're working with our example from earlier: that is, an IFS on $X = [0, 1]^2$, with three maps w_1, w_2, w_3, each of which have contraction factor $1/2$. Suppose that we want to display the limit set of this IFS on a monitor; to be particularly specific, let's suppose that we're trying to draw this limit set on a $n \times n$ grid of pixels.

How can we do this? Well, let's subdivide $[0, 1]^2$ into a grid of $\frac{1}{n} \times \frac{1}{n}$-sized cells, each of which we think of as a pixel. If any points from our limit set is contained within any such cell, we turn that pixel "on," and otherwise keep that cell turned "off."

So, for example, if we had a particularly small monitor — say, 8×8 pixels — we would display the following images when drawing the sets ☺, $W(☺), W^2(☺), \ldots$



Notice that when we do this, our image "stabilizes" after a few steps! In particular, the pixelated image we drew for $W^3(☺)$ was the same as for $W^4(☺)$ and $W^5(☺)$ (and in fact for any future iteration, if you draw it out!)

This is not too surprising, when you look at how the maps w_i work. In particular, notice the following property:

[2] In practice, there are some minor restrictions on what "any" means here that need a lot of real analysis/topology to talk about. Basically, any set you're going to ever look at will work here!

Lemma. Take any two points $\vec{a}, \vec{b} \in X$, and any sequence $w_{i_1}, \ldots w_{i_m}$ of our IFS maps. Then, we have that

$$d\left(w_{i_1}(w_{i_2}(\ldots(w_{i_m}(\vec{a})))), w_{i_1}(w_{i_2}(\ldots(w_{i_m}(\vec{b}))))\right) \leq \frac{d(\vec{a},\vec{b})}{2^m}.$$

Proof. This is just because each map w_i contracts space by a factor of $1/2$; therefore, if we apply m of these maps, we contract space by a factor of $1/2^m$! □

Suppose that $\frac{\sqrt{2}}{2^m} < \frac{1}{n}$. Then, on our $n \times n$ pixel grid, notice that this tells us that for **any** two points $\vec{a}, \vec{b}$, the two points

$$w_{i_1}(w_{i_2}(\ldots(w_{i_m}(\vec{a})))), w_{i_1}(w_{i_2}(\ldots(w_{i_m}(\vec{b}))))$$

are distance at most $1/n$ apart; in other words, the two points $\vec{a}, \vec{b}$ are "indistinguishable" on our monitor after m applications of our IFS maps!

In particular, this tells us that for any $\vec{a}$ and any $w_{i_{m+1}}$, we cannot distinguish $w_{i_1}(w_{i_2}(\ldots(w_{i_m}(\vec{a}))))$ and $w_{i_1}(w_{i_2}(\ldots(w_{i_m}(w_{i_{m+1}}(\vec{a})))))$ — i.e. if we want to draw our image, we only need to apply W to our set m times, instead of having to somehow apply it "infinitely" many times! In particular, only the most recently applied m maps matter for determining where $\vec{a}$ is sent; applying more maps doesn't matter!

This gives us our first answer for how to draw our image on a $n \times n$ grid of pixels: simply pick out m such that $\frac{\sqrt{2}}{2^m} < \frac{1}{n}$, take any set A, and create the set $W^m(A)$. To make our lives easier, because (as noticed before) it doesn't matter what set we start with, we can start with a one-element set $\{\vec{a}\}$ for any single point $\vec{a}$:

$$W^m(\{\vec{a}\}) = \{w_{i_1}(w_{i_2}(\ldots(w_{i_m}(\vec{a})))) \mid i_1, \ldots i_m \in \{1,2,3\}\}.$$

This is a finite set, containing at most 3^m many points (one for each sequence of length m from $\{1,2,3\}$); if we draw them all, we get our image!

1.3 How to Draw a Fractal: A Randomized Approach

This, however, is not the most efficient approach. In the above system, we calculate the two points

$$\overbrace{w_1(w_1(w_1\ldots(w_1(\vec{a}))\ldots))}^{m \text{ functions}},$$
$$\overbrace{w_2(w_1(w_1\ldots(w_1(\vec{a}))\ldots))}^{m \text{ functions}}$$

separately; this takes us $2m$ calculations. However, this is very inefficient! As we noted above, for our purposes, only the most recently applied m functions matter for determining where a point goes. So, if we know

$$\overbrace{w_1(w_1(w_1\ldots(w_1(\vec{a}))\ldots))}^{m \text{ functions}},$$

we can simply apply w_2 to this to get

$$\overbrace{w_2(w_1(w_1(w_1\ldots(w_1(\vec{a}))\ldots)))}^{m+1 \text{ functions}},$$

which by our argument before is the same as applying the most recent m maps: i.e.

$$\overbrace{w_2(w_1(w_1\ldots(w_1(\vec{a}))\ldots))}^{m \text{ functions}}!$$

So, if we are clever, we can save ourselves $m-1$ calculations here with this one point! This inefficiency generalizes, of course — there was nothing special about the all-w_1 sequence here — which makes us want to know how to consistently avoid this inefficiency.

One solution for this is to adopt a **randomized** algorithm. Specifically, consider the following process:

0. Take any point $\vec{a} \in X$, and any sequence of m maps $w_{i_1}, \ldots w_{i_{m-1}}$. Set $\vec{x_0} = w_{i_m}(w_{i_{m-1}}(\ldots w_1(\vec{x})))$.
1. Suppose that we have defined the point $\vec{x_i}$. Randomly pick a value $k \in \{1,2,3\}$, and define $\vec{x_i} = w_k(x)$.
2. Draw the point $\vec{x_i}$, and return to 1.

If we let the process above run long enough, we will eventually randomly list every possible sequence of values from $\{1,2,3\}$; consequently, because only the most recent m values matter for when we're applying maps to a point, the process above will eventually generate every point in $W^m(\vec{a})$!

1.4 How to Draw a Fractal: Graphs and Eulerian Circuits

This, however, is still not the most efficient process. Consider the task of simply drawing a limit set on a 4×4 grid, for which we would need to generate $W^2(\{\vec{a}\})$. This set potentially has 9 elements to consider:

$$w_1(w_1(\vec{a})), w_1(w_2(\vec{a})), w_1(w_3(\vec{a})), w_2(w_1(\vec{a})), w_2(w_2(\vec{a})), w_2(w_3(\vec{a})), w_3(w_1(\vec{a})), w_3(w_2(\vec{a})), w_3(w_3(\vec{a})).$$

These correspond to the three length-2 sequences of symbols from $\{1,2,3\}$, namely

$$11, 12, 13, 21, 22, 23, 31, 32, 33.$$

If you head over to random.org and generate a random list of these sequences, you'll be surprised at how long a random sequence might take to generate these numbers! For example, the first sequence of twenty numbers I tried,

$$32312123123211333112$$

is missing the 22 sequence. The second such sequence I tried worked,

$$22312113231332322123$$

but the third one I tried failed to have the 13 sequence:

$$23332321121223121233$$

As well, you can notice that these sequences are also inefficient in a number of ways. All of the three sequences above repeated several smaller strings multiple times; the first has 12 show up several times, for instance.

What we want, then is a way to create such a sequence that avoids repeating earlier sequences (as this is inefficient) and yet lists all of the sequences we want! This is an interesting combinatorial object, and as such mathematicians have given it a name:

Definition. A **DeBruijn** sequence on symbol set $\{1, 2, \ldots N\}$ with window length l is a sequence made out of listing symbols from $\{1, 2, \ldots N\}$, so that the following holds:

- Every possible string of l consecutive symbols comes up exactly once.

For example,

$$2231211\boxed{3\ 2}313\boxed{3\ 2}322123$$

is not a DeBruijn sequence on symbols $\{1, 2, 3\}$ with window 2: as boxed above, we have repeated a pair of length-2 strings. Conversely,

$$1122331321$$

is a DeBruijn sequence!

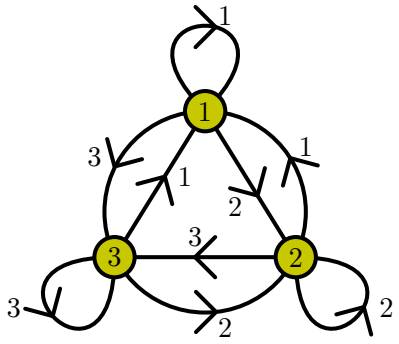
If we can find these DeBruijn sequences, we can use them as the "random" sequence in our algorithm earlier to efficiently find and draw our limit sets! So it suffices to figure out how to create these objects.

Surprisingly, this is where graphs come in! To see how this works, consider the following small example, which we can use to generate the DeBruijn sequences on $\{1, 2, 3\}$ with window size 2.

Draw the following directed[3] graph:

- Vertices: draw three vertices, one for each of $\{1, 2, 3\}$.
- Edges: draw an edge from every vertex to every other vertex, including itself. Label each edge with the vertex that it ends at.

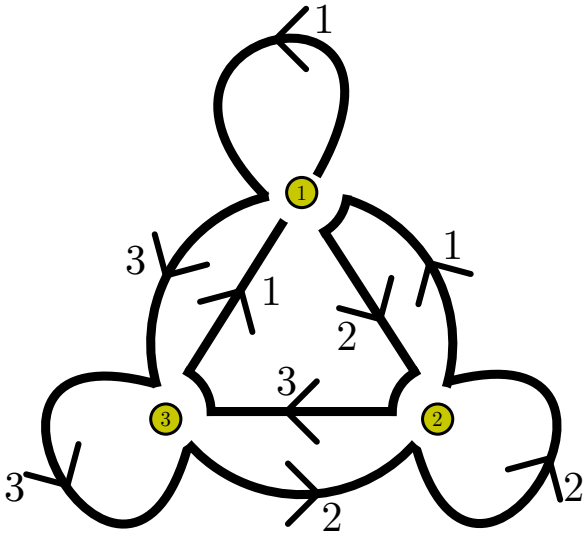
This gives us the following picture:



[3]This is a graph, where we add directions to all of our edges, and also allow a vertex to have a "loop-edge" that goes to itself.

In this situation, suppose we have something called an **Eulerian walk** of our graph G: a walk[4] on G so that every edge of G shows up exactly once.

We draw such a walk here:



Take any such walk, and do the following:

- Write down the name of the vertex our circuit starts at.
- Then, write down in order the names of each of the edges in our walk.

This gives us the following sequence:

$$1133221231.$$

Notice that this is a DeBruijn sequence! This is because for any pair of symbols (i, j) from $\{1, 2, 3\}$, there is a unique vertex-edge pair in our graph that is also named (i, j) (as we have three vertices $1, 2, 3$ and each has an edge leaving it for each possible symbol $1, 2, 3$, thus giving us all nine possible triples (i, j)! So, if our walk doesn't repeat any edge, then writing down the edges will give us a DeBruijn sequence.

We can generalize this to a DeBruijn sequence on $\{1, 2, \ldots N\}$ with window size l as follows:

- Vertices: draw a vertex for each element of $\{1, 2, \ldots N\}^{l-1}$: that is, a vertex for each sequence of $l - 1$ symbols of $\{1, 2, \ldots N\}$.
- Edges: for any two vertices of the form $a_1 a_2 \ldots a_{l-1}$, $a_2 a_3 \ldots a_{l-1} b$, draw an edge from $a_1 a_2 \ldots a_{l-1}$ to $a_2 a_3 \ldots a_{l-1} b$ and label it b.

This, too, gives us a DeBruijn sequence, and gives us a nice, efficient way to draw our fractals. Cool!

[4] A **walk** in a graph is a sequence of edges $e_1, e_2, \ldots e_n$ so that the endpoint of e_i is the start of e_{i+1} for every i: in a sense, a "walk" is describing a way to move around on the edges of our graph!