The 3D Chaos Game

Joey Bloom

December 15 2015

Abstract

The Chaos Game is a method of fractal image generation. It is a particle system that draws points one by one according to a set of rules, and the points drawn will approach an attractor which may be a fractal. Many well known fractals in the euclidean plane can be produced by the Chaos Game. I developed an RTICA using HTML5 Canvas and Javascript to display these fractals and allow the user to manipulate the initial conditions of the Chaos Game to produce approximations of the attractors of Iterated Function Systems

1 The Rules of the Game

1.1 Basic/simple/accessible version

One definition of the Chaos Game at a minimum requires a finite set of points $S = \{A_0, A_1, A_2, ..., A_k\}$ and a ratio r where $0 < r < 1$.

1. Choose at random a point $A \in S$.
2. Choose at random a point $B \in S$.
3. Let d represent the distance from A to B. Move point A a distance of rd in the direction of B. That is, $A := rA + (1 - r)B$.
4. Draw the new point A.
5. Repeat from step 2 until a a good enough approximation of the attractor is produced.

1.2 Formal definition

An affine transformation maps each point in a space R to another point in R such that colinearity and distance ratios are preserved. The former means that any three or more points that lay on a line before the transformation will lay on a line after the transformation. The latter means that for any three or more points on a line, the distance between the points stays in proportion. Given three arbitrary colinear points P_1, P_2, P_3, a distance function d, and an arbitrary affine transformation T, we have the following:

$$\frac{d(P_1, P_2)}{d(P_2, P_3)} = \frac{d(T(P_1), T(P_2))}{d(T(P_2), T(P_3))}$$

Examples of affine transformations are translations, rotations, dilations, skews, and compositions thereof. Mathematically, an affine transformation T has the form $T(x) = Ax + t$ where A is a matrix and t is a column vector.

An *iterated function system*, or IFS, is a finite set of affine transformations that are all *contractive*. An affine transformation $T(x)$ is contractive if for all displacement vectors a and b, $|T(a) - T(b)| < k|a - b|$ where $0 < k < 1$. In English, any two points will be strictly closer to each other after T is applied to them. Furthermore, when T is applied iteratively on any two points, the distance between them approaches zero.

1.3 Visualizing IFS

There are two main methods for visualizing an IFS. The first is the *deterministic* method. Given a starting point, each of the transformations in the IFS are applied to the point and the result points are drawn. Then, for each result point from the previous step, each of the transformations in the IFS are applied to the point and those result points are drawn. This tree-like pattern continues to apply all possible sequences of transformations in the IFS of an arbitrary length.

The second method of visualizing an IFS is the Chaos Game. Where the deterministic method carefully enumerated all possible sequences of transformations, the Chaos Game randomly chooses one transformation from the IFS with equal probability, applies it to the starting point, and then chooses another transformation at random and applies it to that. The Chaos Game continues to iteratively apply randomly chosen transformations, often producing a desirable image more efficiently than the deterministic approach.

2 RTICA Design

2.1 Code Files

My code is divided into four files. Three bear the extension `.js` and contain only Javascript; these are `cube3D.js, AffineTransFinal.js,` and `IFS.js`. Each of these contains code relating to a type of object (almost like a class, but Javascript does not have formal class structure). The fourth of my code files, `final.html`, contains both Javascript and HTML, and contains all of the user interface code for the RTICA and uses the other three files to do the math for the chaos game. `final.html` is the main page for the RTICA.

2.2 cube3D.js

This file defines `cube3D` objects. `cube3D` objects have eight vertices and twelve edges. The vertices and edges have colors associated with them.

`cube3D` objects have a function `draw` that draws the cube on a canvas. By setting the `numbers` parameter to `true`, this function will also number each vertex in the drawing. This is useful to see the effect of a rotation on a cube.

2.3 AffineTransFinal.js

This file defines `AffineTrans` objects. `AffineTrans` objects contain two arrays, `A` and `t`, analogous to the components of an affine transformation when represented in $Ax + t$ form.

`AffineTrans` objects contain the function `transform(x)`, which returns the result of calculating $Ax + t$ for A and t from the `AffineTrans` object and x passed in as a parameter.

The function `compose(other)` composes two affine transformations together and returns the result. The algebra here is the following, where `this` is represented by $T_1(x) = A_1x + t_1$ and `other` is represented by $T_2(x) = A_2x + t_2$.

$$T_2(T_1(x)) = A_2A_1x + T_2(t_1)$$

At the bottom of this file are several global functions to create specific types of `AffineTrans` objects; namely, the identity transformation, translations, skews, dilations, and rotations.

2.4 IFS.js

This file defines `IFS` objects. `IFS` objects have an array of `AffineTrans` objects, analogous to the manifestation of a mathematical IFS as a finite set of affine transformations

The function `chaosGame` contains the code to perform the Chaos Game using the IFS object `this`. This function uses the origin as its initial point, setting the variable `point` to contain the array `[0,0,0]`. Given the positive integer parameter `iter`, `chaosGame` loops `iter` times. For each iteration of the loop, it chooses at random with equal probability one of the `AffineTrans` objects in `this.transArr`, calls `transform(point)` to apply that affine transformation to `point`, and stores the result in `point` for the next iteration and pushes the result onto `points` so that we can draw the attractor later.

The function `drawIFS` draws the precomputed points of the attractor of this IFS. It loops through each point in `points` and draws it on the canvas, optionally using the colors for each point from `pointColors` as well. This function also draws a black unit cube for scale.

The function `drawTrans` creates and draws a unit cube with colored and numbered vertices. Then for each `AffineTrans` object in `transArr`, (but only if the corresponding boolean in the `trans` boolean array parameter is `true`) creates another colored and numbered unit cube, transforms each vertex of the unit cube using `this.transform(x)`, and draws the transformed cube.

2.5 final.html

This file is the RTICA itself. It contains the HTML for the user interface to interact with `IFS` and `AffineTrans` objects.

I draw the IFS using the functions from the file IFS.js. The camera for the IFS is controlled by the 6 movement buttons. These buttons change the spherical coordinates of the camera.

The smaller canvas towards the bottom of the page shows a representation of each affine transformation. The black omnipresent cube is the unit cube. The other cubes that are drawn represent each checked affine transformation. The color of the cube's edges correspond to the color around each checkbox. If no other cubes are drawn, that means no boxes are checked in the list of affine transformations.

The RTICA allows the user to edit affine transformations in two ways. The first is numeric mode, where the user can directly manipulate the matri-

ces A and t for the `AffineTrans` object. The second is composite mode. The user can create a composition of one or more basic affine transformations (translations, skews, dilations, rotations). My code takes care of calculating the correct values for A and t and displays them in the matrices on the page.

3 Bibliography

Barnsley, Michael F. "Fractals Everywhere." Academic Press, Inc, 1988. Call Number 516 B267f at Grainger Engineering Library, UIUC.

Barnsley, Michael F. and Andrew Vince. "The Chaos Game on a General Iterated Function System." `http://arxiv.org/pdf/1005.0322v1.pdf`.

Francis, George. "BASIC Pocket Graphics Programs" UIUC Math 198 Hypergraphics 2001 Class Notes. `http://new.math.uiuc.edu/public198/pcPocketry/gasketry/gasket.pdf`.

Hart, John C. "Fractal Modeling." Lecture Slides for CS 418 at UIUC.

Weisstein, Eric W. "Affine Transformation." From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/AffineTransformation.html`.