Flows with Strange Attractors

Shunhua (Sam) Fu

December 11, 2013

Abstract

The notion of *flow* describes a system that changes with time. Mathematically, flows can be described by dynamical systems, which can be defined by differential equations or recurrence equations. Mathematicians have long been interested in a class of dynamical systems called strange attractors, in which systems evolve toward defined shapes with fractal structures. My project visualizes flow by showing various strange attractors and how they change with time. Dozens of randomly plotted points over time asymptotically converge toward a stable structure. I investigated 10 such dynamical systems in total.

1 Mathematics

How does one visualize flow? Let us take a look at a simple analogy. A river is a good parallel for the concept of flow. How can we tell that a river flows? Well, we could just look at the water. The water clearly moves and changes with time. Indeed, it seems obvious that a river flows. But in empty 3-space, there is no "water" to inform us whether there is flow. What would we do if we could not see the water in the river? One way is to drop a cork into the water. The cork would move over time, following the contours of the river. But this simply a trajectory; it does not necessarily mean that there is flow. Just because one cork moved does not tell us decisively that this invisible river really is flowing. But what if we randomly dropped many corks into this river? They would all start moving and outline the shape of the river. By watching the aggregate of trajectories of these corks, we would be able to "see" this river, without ever needing to see the water.

(On a side note, in the context of the river analogy, a strange attractor would be the equivalent of a whirlpool or vortex. The corks you drop in would be "attracted" by this whirlpool and converge upon its shape, even though we cannot see the water directly.)

All of the strange attractors I investigated could be defined by either differential equations or recurrence equations.

1.1 Solving Ordinary Differential Equations—Lorenz Strange Attractor

Solving the differential equations for the respective variables, I could obtain computationally simple equations for finding the coordinates of the next point. I will demonstrate this with the Lorenz equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \tag{1}$$

(Here, σ, ρ, and β are constants. We'll see later that changing their values changes the shape of the attractor.)

Multiply both sides by dt

$$\begin{aligned} dx &= (\sigma(y-x))dt \\ dy &= (x(\rho-z)-y)dt \\ dz &= (xy-\beta z)dt \end{aligned}$$

The left sides are now simply differences, and we can treat them as such. For example, $dx = \Delta x = x_f - x_0$. So,

$$\begin{aligned} x_f - x_0 &= (\sigma(y-x))dt \\ y_f - y_0 &= (x(\rho-z)-y)dt \\ z_f - z_0 &= (xy-\beta z)dt \end{aligned}$$

And we can now solve for the final coordinates.

$$\begin{aligned} x_f &= x + (\sigma(y-x))dt \\ y_f &= y + (x(\rho-z)-y)dt \\ z_f &= z + (xy-\beta z)dt \end{aligned} \tag{2}$$

And there we have it; we have found the x, y, and z coordinates for our next point. If we do this a few hundred times for a bunch of points, they will converge beautifully on the Lorenz strange attractor.

1.2 Rössler Attractor

$$\begin{aligned} \frac{dx}{dt} &= -y - z \\ \frac{dy}{dt} &= x + ay \\ \frac{dz}{dt} &= b + z(x-c) \end{aligned} \tag{3}$$

a, b, and c are constants.

Solving for x_f, y_f, and z_f, we get

$$\begin{aligned} x_f &= x + (-y-z)dt \\ y_f &= y + (x+ay)dt \\ z_f &= z + (b+z(x-c))dt \end{aligned} \tag{4}$$

1.3 Hénon Map

$$\begin{aligned} x_{n+1} &= y_n + 1 - ax_n^2 \\ y_{n+1} &= bx_n \end{aligned} \tag{5}$$

You might notice two things. First, this is not a differential equation but rather recurrence equation. There is no need for mathematical manipulation here; this equation is ready to go as-is. Second, you might note that there is no equation for the z coordinate. That is because this is a 2-D attractor. I simply set $z = 0$.

1.4 Chua's Attractor (Double Scroll Attractor)

This attractor is based on three ordinary differential equations that model the dynamics of Chua's circuit, which is a peculiar yet simple circuit which manifests the butterfly effect—i.e., it exhibits chaos theory behavior.

$$\begin{aligned}
\frac{dx}{dt} &= \alpha(y - x - f(x)) \\
\frac{dy}{dt} &= x - y + z \\
\frac{dz}{dt} &= -\beta y \\
f(x) &= m_1 x + \frac{m_0 - m_1}{2}|x+1| - |x-1|)
\end{aligned} \tag{6}$$

The equation for $f(x)$ was obtained from Valentin Siderskiy's matlab simulation. [1] α, β, m_0, and m_1 are constants. If we solve the differential equations, we get

$$\begin{aligned}
x_f &= x + (\alpha(y - x - f(x)))dt \\
y_f &= y + (x - y + z)dt \\
z_f &= z + (-\beta y)dt
\end{aligned} \tag{7}$$

1.5 Liénard-Van der Pol Oscillator

$$\begin{aligned}
\frac{dx}{dt} &= y \\
\frac{dy}{dt} &= \mu(1 - x^2)y - x
\end{aligned} \tag{8}$$

This oscillator is again 2-dimensional. I showed flow for various random values of μ, with z set to 0. It also looks nice if you set $z = \mu$.

$$\begin{aligned}
x_f &= x + (y)dt \\
y_f &= y + (\mu(1 - x^2)y - x)dt
\end{aligned} \tag{9}$$

1.6 Rabinovich-Fabrikant System

$$\begin{aligned}
\frac{dx}{dt} &= y(z - 1 + x^2) + \gamma x \\
\frac{dy}{dt} &= x(3z + 1 - x^2) + \gamma y \\
\frac{dz}{dt} &= -2z(\alpha + xy)
\end{aligned} \tag{10}$$

α and γ are, of course, constants that control how the system evolves over time.

$$\begin{aligned}
x_f &= x + (y(z - 1 + x^2) + \gamma x)dt \\
y_f &= y + (x(3z + 1 - x^2) + \gamma y)dt \\
z_f &= z + (-2z(\alpha + xy))dt
\end{aligned} \tag{11}$$

1.7 Tinkerbell Map

This is a fun dynamical system that might remind you of a certain Disney fairy.

$$\begin{aligned}
x_{n+1} &= x_n^2 - y_n^2 + ax_n + by_n \\
y_{n+1} &= 2x_n y_n + cx_n + dy_n
\end{aligned} \tag{12}$$

This is a 2-D dynamical system, so I just set $z = 0$. a, b, c, and d are constants.

1.8 Ikeda Map

$$\begin{aligned} x_{n+1} &= 1 + u(x_n cos(t_n) - y_n sin(t_n)) \\ y_{n+1} &= u(x_n sin(t_n) + y_n cos(t_n)) \\ t_n &= 0.4 - \frac{6}{1 + x_n^2 + y_n^2} \end{aligned} \tag{13}$$

Once more, this is a 2-dimensional dynamical system, and u is a constant. For certain values of u, the system has a chaotic attractor. I set $u = 0.85$, where interestingly there are two attractors fighting each other for control, much like nearby galaxies do.

1.9 Rainey Dynamical System

$$\begin{aligned} \frac{dx}{dt} &= yz \\ \frac{dy}{dt} &= xz \\ \frac{dz}{dt} &= xy \end{aligned} \tag{14}$$

Chris Rainey is a former REU/illiMATH student who made up his own set of ODEs for a dynamical system. In his honor, I have also included them into my project. Unfortunately, they are dynamical systems but do not contain attractors.

$$\begin{aligned} x_f &= x + (yz)dt \\ y_f &= y + (xz)dt \\ z_f &= z + (xy)dt \end{aligned} \tag{15}$$

1.10 Fu Attractor

I decided to reward my hard work on this project, so I created my own strange attractor. I think it looks like a genie lamp, and it has a cool jet with spirals going around it.

$$\begin{aligned} \frac{dx}{dt} &= \alpha\frac{y + x}{z} \\ \frac{dy}{dt} &= x(\beta - x - y - z) \\ \frac{dz}{dt} &= \frac{xyz}{\gamma} \end{aligned} \tag{16}$$

α, β, and γ are constants.

$$\begin{aligned} x_f &= x + (\alpha\frac{y + x}{z})dt \\ y_f &= y + (x(\beta - x - y - z))dt \\ z_f &= z + (\frac{xyz}{\gamma})dt \end{aligned} \tag{17}$$

2 WebGL and Three.js

I used WebGL for my project. WebGL is a JavaScript graphics library that implements OpenGL ES 2.0. The use of JavaScript and HTML5 moves this course into a new era, one in which the projects can be viewed

directly from a browser, without the need for the end user to install any additional dependencies. This allows my project to be easily displayed on the course website.

Three.js is a JavaScipt library that implements WebGL. It is very convenient and negates the need for much of the boilerplate code required by native WebGL. I used Three.js in my project because it allowed me to focus more on the mathematics and coding rather than treading through a landmine of WebGL initiation errors.

2.1 Basic Outline

I will give a quick summary on the *flow* of my code.

The user begins in `flows.html`. The Lorenz strange attractor begins to form upon opening the page. On the back end, I first detect whether the user's browser can display my program. Then, I declare a bunch of global variables. Finally, I create a canvas on which to draw.

```
if (!Detector.webgl) Detector.addGetWebGLMessage();

var container, stats;
var camera, scene, renderer, particles, geometry, material, parameters, i, h, color;
var coords = [];
var trajColors = [];
var colors = [];
var particleNum = 0;
var execute = true;
var which = 1;

var mouseX = 0, mouseY = 0;

var windowHalfX = window.innerWidth / 2;
var windowHalfY = window.innerHeight / 2;

container = document.createElement('div');
document.body.appendChild(container);
```

Next, I initialize some global objects such as the Three.js camera and the WebGL renderer. I also add some event listeners so I can detect the mouse.

```
camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 1, 3000);
scene = new THREE.Scene();
renderer = new THREE.WebGLRenderer();
stats = new Stats();

document.addEventListener('mousemove', onDocumentMouseMove, false);
document.addEventListener('touchstart', onDocumentTouchStart, false);
document.addEventListener('touchmove', onDocumentTouchMove, false);
window.addEventListener('resize', onWindowResize, false);
```

In this file, I also have scripts declaring my global functions:

```
function addPoint(x, y, z) { ... }
```

```
function removeAll() { ... }
function onWindowResize() { ... }
function onDocumentMouseMove(event) { ... }
function onDocumentTouchStart(event) { ... }
function onDocumentTouchMove(event) { ... }
```

There is a nice user interface built using jQuery UI. The panel can be moved by being dragged. This accordion allows the user to choose between the 10 different dynamical systems. The user can click a dynamical system name to view the equations, and then he or she can click the "Show me!" button to start generating the chosen attractor.

jQuery detects when the "Show me!" button is clicked, and certain JavaScript commands are run. These can be seen in the `controller.js` file. Namely, I stop generation of the current attractor, clear my WebGL buffers, clear my renderer, and then begin generation of the chosen dynamical system. The relevant code for the Lorenz "Show me!" button is as follows:

```
$("#lorenzButton").click(function() {
    execute = false;
    geometry.__webglVertexBuffer = null;
    geometry.__webglColorBuffer = null;
    renderer.clear();
    removeAll();
    execute = true;
    which = 1;
    lorenzInit();
    lorenzAnimate();
});
```

The last two functions begin generation of the Lorenz strange attractor. There are 4 important functions:

```
function lorenzInit() { ... }
function lorenzAnimate() { ... }
function lorenzRender() { ... }
function calcLorenz() { ... }
```

In `lorenzInit()`, I first reset my variables.

```
coords = [];
trajColors = [];
colors = [];
particleNum = 0;

camera.near = 1;
camera.far = 3000;
camera.position.x = 0;
camera.position.y = 0;
camera.position.z = 100;

geometry = new THREE.Geometry();
```

Then I randomly generate initial starting vertices, randomly generate a color for each trajectory, and then plot the starting vertices. Importantly, I collapse all unused vertices into a point way off screen. This will

be discussed later. I set my geometry color array and then add the Three.ParticleSystem to the scene.

I then animate and render the vertices. There is a call in `lorenzAnimate()` to `requestAnimationFrame(lorenzAnimate)`. This makes the method call itself until all the vertices have been plotted. Each vertex is calculated through the `calcLorenz()` function.

2.2 Sketchy Hacks

Many things did not work natively. I had to do some dirty hacks in order to get my program to work properly.

First, Three.js allocates memory for its vertex and color buffers with immutable sizes. But my dynamical system grows in size as more particles are plotted. To get around this, I must declare how many vertices I intend to have upon creating the geometry (which then creates the vertex buffer). I then collapse all my unused vertices to a point far off screen. Lastly, I declare "dirty vertices" with `geometry.verticesNeedUpdate = true`. [2] [3] [4] [5]

Next, I had to figure out how to get it all to work on one HTML page. To clear up memory, I needed to destroy the ParticleSystems when they no longer were needed. My solution was to create the `removeAll()` function. It is called when the "Show me!" buttons are pressed. Execution is stopped and all ParticleSystems in the scene are removed.

But the program still inexplicably lagged when the buttons were pressed. I spent much time debugging and finally resolved that the internal WebGL buffers were not being destroyed properly, so I needed to do so manually. I destroy the WebGL vertex buffer and the WebGL color buffer every time the buttons are pressed. [6] This is referred to as a "really ugly hack" by alteredq, one of the creators of Three.js. [7] I think that it is beautiful though, and it gave me significant boosts in performance.

3 Previous Versions

Before working on the final project, I made a few proof-of-concept projects. I will describe them briefly.

3.1 Trajectory Spline

I graphed the motion of one point along the Lorenz strange attractor. I used the Numeric Javascript library to aid me in solving the ordinary differential equations. It was neat because I implemented sliders to change the constants. But importantly, this pre-project did not depict flow. I had only picked one point to start with—I was simply showing trajectory. This would be the equivalent of dropping a single cork into the metaphorical invisible river. It might go into the whirlpool, but we can't say much about the whirlpool based only on the path of the single cork.

3.2 Canvas

I had made a prior version of the Lorenz strange attractor that showed flow using the HTML5 Canvas renderer. This program looks similar to the attractor that appears upon opening my final project. It has bright colors and the particles are circles; it was very pretty. Unfortunately, the Canvas renderer was not made for rendering the thousands of particles I had, and it quickly dropped in FPS after only about 1,000 particles. The Canvas API was made to be 2D, and it was very inefficient because there were no graphics buffers (I was adding and drawing particles on the fly).

References

[1] V. Siderskiy. *Matlab simulation code for Chua's circuit.* URL: http://www.chuacircuits.com/matlabsim.php.

[2] mrdoob. *How to update things.* URL: https://github.com/mrdoob/three.js/wiki/Updates.

[3] Reinmar. *Question about performance tuning.* URL: https://github.com/mrdoob/three.js/issues/167.

[4] Shawson. *Mutable Particle System.* URL: https://github.com/mrdoob/three.js/issues/172.

[5] Reinmar. *Balls.* URL: https://github.com/Reinmar/ideas/blob/master/ball/balls.js.

[6] mrdoob. *WebGLRenderer.* URL: https://github.com/mrdoob/three.js/blob/master/src/renderers/WebGLRenderer.js.

[7] Tjomas. *Dynamic faces.* URL: https://github.com/mrdoob/three.js/issues/342.