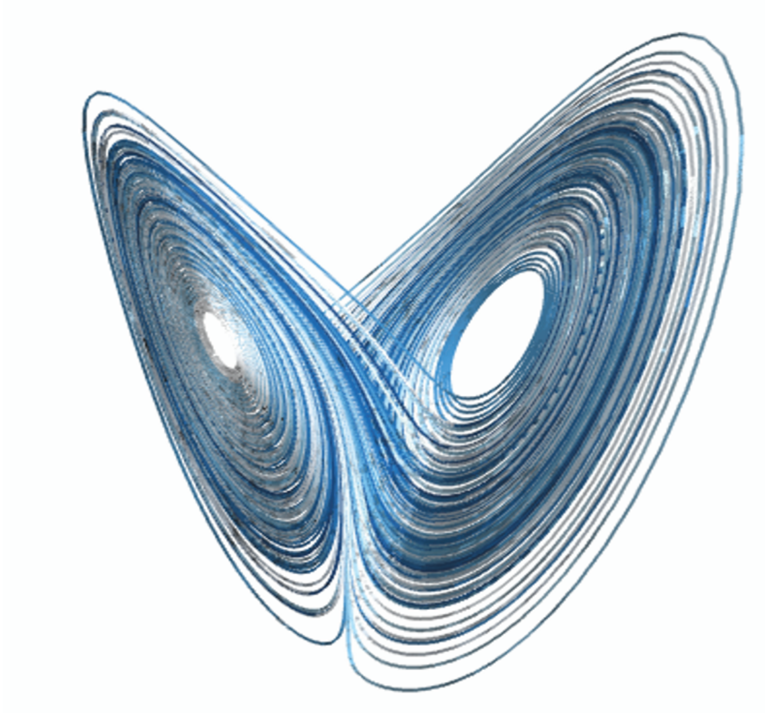
St. Xavier's College, Kolkata (Autonomous)

A DISSERTATION ON THE

VISUALIZATION OF LORENZ ATTRACTOR



submitted by

Amlan Saha Kundu

Semester: VI, Roll No.: 12

DEPARTMENT OF MATHEMATICS,

under the supervision of

Prof. Diptiman Saha

Associate Professor

DEPARTMENT OF MATHEMATICS,

The Visualization
of
Lorenz Attractor

———————————————————————————————————————

A Dissertation

presented to

The Department of Mathematics

St. Xavier's College, Kolkata

———————————————————————————————————————

by

Amlan Saha Kundu

MTMA, Roll. 12

under the supervision of

Prof. Diptiman Saha

April 2020

Acknowledgements

My thanks go out to all in the Mathematics department. Specially to Prof. Diptiman Saha for guiding me with the idea behind this dissertation, to Prof. Sucharita Roy, Prof. Gaurav Tripathi and Prof. Anindya Dey for tolerating me with my numerous queries and inspiring me to work hard in this field, to my subterranean colleagues for allowing me to distract them from their work, for their camaraderie, and for sharing in my excitement of the completed thesis, to my parents for granting me the opportunity to attend this fine institution, to Benjamin L. Kurian, dear hostel mate, my table tennis partner and my senior, who constantly help me out from this topic; to Sweety Mandal, a polymath and dear classmate, who constantly supplies me her gadget for adding the final-touch to this project and also helping me out, the hefty printing work of the synopsis of this project; as well as I also want to thank my dear, exciting, intriguing, lovely and estimable friends: Aniket Bhattacharyea, for round the clock technical support, as well as Anunoy Chakraborty and Sayanti Maity, for constantly sending push notifications to stay in the field of Mathematics instead of roaming around in the world of distraction, specially Computer Science. I am also very thankful to some amazing persons on the Internet, who has enlightened me with their vast knowledge, specially to Prof. David P. Feldman, for his awesome book "Chaos and Fractals: An Elementary Introduction" and also for the free course on this topic on complexity explorer, to Prof. Daniel Shiffman of Coding Train for his beautiful explanation and encouragement for the code; and all whom, whom I have known and loved; who have made this whole experience worthwhile.

Declaration

I affirm that I have identified all my resources and no part of my dissertation paper uses unacknowledged materials. I don't claim the works done here are solely original. Various books and internet resources have been used to carry out this project and have been referred in the Bibliography section.

Amlan Saha Kundu

B.Sc. Mathematics (H)
Sem: VI, Roll No.:12

St. Xavier's College (Autonomous), Kolkata

Index

Abstract

The Lorenz Attractor is a system of differential equations derived from simple models of weather phenomena. The beauty of it lies both in the mathematics and in the visualization of the model. Mathematically, the Lorenz Attractor is simple yet results in chaotic and emergent behaviour. Visually, when the values given by the equations are plotted in two- or three-dimensional space, the behaviour is reminiscent of an orbit of an object around two central origins; when looked at from the appropriate dimension the pattern appears similar to a butterfly or infinity or figure eight. The main crux of project entitled as "A Dissertation on the Visualization of Lorenz Attractor" is to build a real-time simulation to visualize the effect of changing of the three positive parameters σ, ρ and β in the Lorenz equations, as follows:

$$\dot{x} = \sigma\,(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = \dot{x}y - \beta\dot{z}$$

where, where, x, y and z are variables

The Lorenz Attractor, with its characteristic butterfly shape, has become a much-published symbol of chaos. It can be found by simply integrating almost any initial point. However, it is much more difficult to understand how the Lorenz attractor organizes the dynamics in a global way.

In this era of modern programming languages, one can easily visualize the Lorenz attractor with the help of Python, MATLAB, etc. but, it results a few drawbacks as follows:

- First, all of these languages need a lot of computing power and some specialized pre-installed software, to execute the code. Even, in some cases, like MATLAB, one has to purchase the software to get a taste of visualization.
- Secondly, the pre-compiled library functions can not provide proper understanding about what is going on inside them.
- Thirdly, the script must be refreshed each and every time to visualize the change, which indeed breaks the continuity of visualization.
- Last but not least, you can not customize the output as you imagine.

Keeping, above points in my mind, I have created such a software, which is completely free, has ability of simulate in real-time for continuous visualization, supports full customization and above all can be run in any system with minimal hardware. I called it as Lorenz Attractor Visualizer, abbreviated as LAVis and in this dissertation, I will discuss the logic and ideas behind this visualization in such a way, so that a reader, who is a complete stranger to this field can understand.

A Historical Introduction of Chaos

Most people who have never studied chaos theory, suffer from a gross ambiguation of the word “chaos”. In our common usage, the term refers to something that completely lacks any sort of order. We hear always of a chaotic desk being littered with disorganized papers, or we read of a riotous group of people breaking out in chaos. In fact, the word originates from the Greek notion of the disorderly mess of the elements which composed the universe before order was imposed, before the Earth came into existence and spawned life. Another misconception is that chaos refers to random behavior. However, in the physical science “Chaos” refers not to a system in which disorder reigns superior, but in one, in which the order is extremely intricate. Anyone, who has gazed upon a fractal image has seen the extent to which the order in a chaotic system is present. So, before we begin, let us not be confused by our previous understanding of the word “Chaos”. We now mean it to imply the supremely intricate workings of a system which abides fully by the basic laws of physics. Chaos theory is often considered as one of three greatest revolutions in twentieth century physics along with quantum mechanics and relativity. Before chaos theory, people thought that every problem of classical mechanics was predictable, provided you had the tools to take the necessary measurements or the mathematical skills to solve the system. The belief was that Newton’s laws governed these systems in a deterministic way and we simply had to pay close attention to all the minute details in order to be able to predict the behavior of the system. Unpredictable systems were those that, while governed buy the deterministic laws, had too many variables to measure and keep track of, such as turbulent motion in fluids and weather. We now know that, despite the deterministic laws, which govern certain phenomena, their behavior can be utterly unpredictable. No matter, how many measurements we make or how much we reduce the noise in certain systems, knowledge of the parameters and information concerning a system may never allow us to fully predict the outcomes of what have come to be known as chaotic systems. The first inklings of chaos theory were encountered by Henri Poincare at the turn of the twentieth century in his attempts to address the famous three-body problem. This problem consists of a set of differential equations concerning the motion and interaction of three massive bodies in space under the influence of Newtonian mechanics. In 1887, King Oscar II of Sweden threw a contest in which anyone who could solve the problem or come closest to solving it and in the mean time make significant contribution to classical mechanics, would win a monetary prize. Poincare tried his hand at the three-body problem and encountered solutions that no one has previously dealt with or discussed: he found orbits that were non-periodic, yet neither soared off to infinity, nor settled into a fixed point. He came across a formation known as Poincare’s nightmare: the first glimpses of a chaotic solution. Unfortunately, he had only pen and paper, and approximating solution techniques, such as perturbation theory at his disposal. Although, Poincare didn’t actually solve the problem, he did better than anyone else in trying and won King Oscar’s prize. After the competition however, Poincare did not know how to solve the three-body monster completely, lacking the aid of computational angel, which is nothing but modern computer. Chaos theory lay dormant for decades. It is most commonly thought that Edward Lorenz was the man who re-introduced Chaos Theory to the scientific community in the middle of twentieth century, after a revelation of his own. In the early 1960s, Edward Lorenz was studying the peculiarities of the weather at MIT. It was the dawn of the age of computers and this new technology was about to open up a field of study virtually off limits to those, who came before it. Lorenz designed a computer program to simulate a very simple model of the weather which was essentially composed of 12 variables, a set of rules to abide by, and a “go” button. The program would spit out reels and reels of numbers that corresponded to various changing weather variables, such as atmospheric pressure, wind speeds, temperatures, etc. Lorenz would run these simulations without being able to predict the outcome, despite the deterministic laws he had imposed on the system. The system was deterministic, yet unpredictable. One day, while running his simulations, Lorenz decided to re-create on of the runs. He looked back at the printout of some previous data and recorded the numbers for each parameter somewhere in the middle of the run, typed them into his computer and let his simulation go. As the story goes, he walked down to the end of the hallway to fill his coffee cup; and when he returned, he was stunned to see that the simulation has not re enacted the old simulation, but had set his fanciful weather on a vastly different course. What he has expected to observe was the same progression of events that he had witnessed earlier with the same set of states for each parameter.

Confounded, he shifted through the details and working of his program, looking for the bugs that must have caused this ambiguation. After finding none, he finally realized that while the program printed out data to only three decimal places, the internal machinery of his computer was working with six decimal places, a very small difference. When he thought he had typed in the exact same parameters as his previous run, he had actually dropped a few more decimal places and was off by just a few hundred-thousandths. Looking back at his newly acquired data, he saw that, the simulation repeated the old course of events for a short while, but then widely diverged. The system seemed to be incredibly sensitive to small changes in initial conditions. After some years of work, Lorenz simplified his system down to a set of three non-linear equations that described the currents induced in a convective cell of liquid when heated from the bottom. Although these equations had only three variables and were governed by deterministic laws, they behaved in an unpredictable, aperiodic way and retained an extreme sensitivity to initial conditions. The unpredictability was not due to an inability to measure all of the associative variable and parameters, but rather some inherent quality of the system itself. We will visualize that randomness in our later context, but before that, we will visualize the chaos theory with a comparatively easy real-life simulation.

A few words about Chaos

A dynamical system is chaotic if it possesses all of the following properties:

(1) The dynamical rule is deterministic.
(2) The orbits are aperiodic.
(3) The orbits are bounded.
(4) The dynamical system has sensitive dependence on initial conditions.

Let us consider each of these in turn. But before that, allow me to loosely discuss the mathematical jargons like "orbit", "aperiodic", "bounded" etc. I think, the reader has a clear idea about the function. We may consider function as certain rule that gives definite output for some given input. For example, a function, say $f(x) = 5x$ will give output 10 if we give input as 2.

Now, to understand the concept of "orbit", I will introduce another type of function, known as *iterative function.* Iteration entails doing the same thing again and again using the previous step's output as the next step's input. In other words, we start with a number and apply a function to it to get a new number. Then we take that new number and apply the function to it to get yet another number. Then we apply the function to this new number, and so on. Let's understand the concept with an example.

Suppose our function is the tripling function, $f(x) = 3x$. Let us use 2 as our input. We then apply the function and get $f(2) = 3\times 2 = 6$. We then take 6, our output, and use it as input for the function to get $f(6) = 3\times 6 = 18$. We then repeat the process; i.e., $f(18) = 3\times 18 = 54$ and so on.
Here is another, perhaps clearer, way to see this process:

$$2 \xrightarrow{f} 6 \xrightarrow{f} 18 \xrightarrow{f} 54 \xrightarrow{f} 162 \cdots$$

If we started with a different number, we would get a different series of outputs:

$$0.5 \xrightarrow{f} 1.5 \xrightarrow{f} 4.5 \xrightarrow{f} 13.5 \xrightarrow{f} 40.5 \cdots$$

I think, the idea of iteration is clear to the reader. Now, we will discuss some mathematical notations. For that, we will introduce another function, say, $g(x) = x^2$. If we start, say, with 3, we will get 9 and 81 and 6561 and so on.

$$3 \xrightarrow{g} 9 \xrightarrow{g} 81 \xrightarrow{g} 6561 \cdots .$$

The number we start with, 3 in this particular case, is known as the initial condition or the seed. Very often the initial condition is denoted as x_0, the next value is denoted as x_1, and then x_2, and so on. Iterating a function produces a sequence of numbers.

A sequence is simply a list with an order to it. This sequence is often called an itinerary. This is consistent with the everyday usage of the term; an itinerary is a list, in order, of all the places visited along a journey. The mathematical usage of itinerary is similar; the itinerary of 3 is a list of the results, in order, that one gets from applying the function again and again. Another word for itinerary is orbit. Orbit and itinerary, in their mathematical usages, are synonymous.

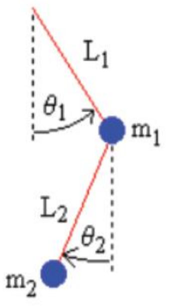
Now, in some cases, for some particular function and seed, we could keep iterating forever—literally forever—and we would never encounter exactly the same number in the itinerary. Such behaviour is said to be aperiodic. Also, for each values of the function in the orbit, if we can get any real number M, such that, $\forall\ i$, $|\ f(x_i)\ | \leq$ M, then we call that, the orbit is bounded by an upper-bound M.

Now, let's back to the chaos again. By dynamical rule I mean the rule that determines the orbit of the dynamical system. In this case, the rule is just the function that iterate. A deterministic function is one in which the input determines the output. That is, if you give the function the same input numerous times, the function will always return the same value again and again. But there is a slight twist in chaos. The all three conditions I have discussed before can be found in non-chaotic systems too. But, in chaos, there is a very special criteria, named as SDIC or sensitive dependence on initial conditions. This is a phenomenon that we have not yet encountered and which we will visualize shortly with an example. In brief, though, a system that has SDIC has the property that a very small change in the initial condition will lead to a very large change in the orbit in a relatively short time. sensitive dependence on initial conditions is more colloquially known as the butterfly effect. Enough of theory, let's visualize SDIC with real simulation

Visualize SDIC with Double Pendulum

In this context we will program two simulation models of double pendulum in p5.js using the following set of equations.

Now, lets simulate the model using p5.js with the help of following code.



$$x_1 = L_1 \sin\theta_1$$

$$y_1 = -L_1 \cos\theta_1$$

$$x_2 = x_1 + L_2 \sin\theta_2$$

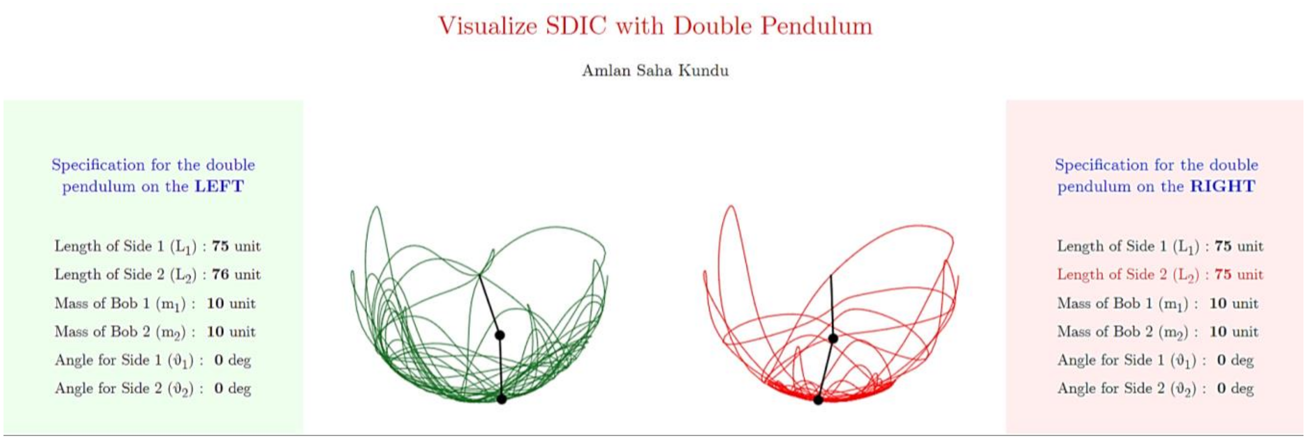
$$y_2 = y_1 - L_2 \cos\theta_2$$

$$\theta_1'' = \frac{-g\,(2\,m_1 + m_2)\sin\theta_1 - m_2\,g\sin(\theta_1 - 2\,\theta_2) - 2\sin(\theta_1 - \theta_2)\,m_2\,(\theta_2'^2\,L_2 + \theta_1'^2\,L_1\cos(\theta_1 - \theta_2))}{L_1\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

$$\theta_2'' = \frac{2\sin(\theta_1 - \theta_2)\,(\theta_1'^2\,L_1\,(m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + \theta_2'^2\,L_2\,m_2\cos(\theta_1 - \theta_2))}{L_2\,(2\,m_1 + m_2 - m_2\cos(2\,\theta_1 - 2\,\theta_2))}$$

```
//Double Pendulum Simulation  by Amlan Saha Kundu
let r1 = 75;  //Here r1 is same as first length in above equation
let r2 = 75;  // Here r2 is same as second length in above equation
let m1 = 10;
let m2 = 10;
let a1 = 0; //For reducing complexity, I replace first angle as a1 instead of ϑ1
let a2 = 0;
let a1_v = 0;
let a2_v = 0;
let g = 1;

let px2 = -1;
let py2 = -1;
let cx, cy;

let buffer;

function setup() {
  createCanvas(400, 370);
   pixelDensity(1);
  a1 = PI / 2;
  a2 = PI / 2;
  cx = width / 2;
  cy = 200;
  buffer = createGraphics(width, height);
  buffer.background(255);
  buffer.translate(cx, cy);
}

function draw() {
  background(175);
  imageMode(CORNER);
  image(buffer, 0, 0, width, height);

  let num1 = -g * (2 * m1 + m2) * sin(a1);
  let num2 = -m2 * g * sin(a1 - 2 * a2);
  let num3 = -2 * sin(a1 - a2) * m2;
  let num4 = a2_v * a2_v * r2 + a1_v * a1_v * r1 * cos(a1 - a2);
  let den = r1 * (2 * m1 + m2 - m2 * cos(2 * a1 - 2 * a2));
  let a1_a = (num1 + num2 + num3 * num4) / den;

  num1 = 2 * sin(a1 - a2);
  num2 = a1_v * a1_v * r1 * (m1 + m2);
  num3 = g * (m1 + m2) * cos(a1);
  num4 = a2_v * a2_v * r2 * m2 * cos(a1 - a2);
  den = r2 * (2 * m1 + m2 - m2 * cos(2 * a1 - 2 * a2));
  let a2_a = (num1 * (num2 + num3 + num4)) / den;

  translate(cx, cy);
  stroke(0);
  strokeWeight(2)
  let x1 = r1 * sin(a1);
  let y1 = r1 * cos(a1);
  let x2 = x1 + r2 * sin(a2);
  let y2 = y1 + r2 * cos(a2);

  line(0, 0, x1, y1);
  fill(0);
  ellipse(x1, y1, m1, m1);

  line(x1, y1, x2, y2);
  fill(0);
  ellipse(x2, y2, m2, m2);

  a1_v += a1_a;
  a2_v += a2_a;
  a1 += a1_v;
  a2 += a2_v;
  buffer.stroke(255,0,0);
  if (frameCount > 1) {
    buffer.line(px2, py2, x2, y2);
  }

  px2 = x2;
  py2 = y2;
}
```

Now, we will create another model similarly and modify one of the arm lengths slightly. For this example, I choose two sets of arm length as (75,75) and (75,76). Now after embedding this .js file into an .html web, we get the simulation as follows:



Visualize SDIC with Double Pendulum

Amlan Saha Kundu

Specification for the double pendulum on the **LEFT**

Length of Side 1 (L_1) : **75** unit
Length of Side 2 (L_2) : **76** unit
Mass of Bob 1 (m_1) : **10** unit
Mass of Bob 2 (m_2) : **10** unit
Angle for Side 1 (ϑ_1) : **0** deg
Angle for Side 2 (ϑ_2) : **0** deg

Specification for the double pendulum on the **RIGHT**

Length of Side 1 (L_1) : **75** unit
Length of Side 2 (L_2) : **75** unit
Mass of Bob 1 (m_1) : **10** unit
Mass of Bob 2 (m_2) : **10** unit
Angle for Side 1 (ϑ_1) : **0** deg
Angle for Side 2 (ϑ_2) : **0** deg

Just visualize, how a small change in the initial condition can cause drastic difference in oscillation pattern. This is known as sensitive dependence on initial conditions (SDIC).

Online simulation can be visualized at https://yoursamlan.github.io/SDIC_dp/

The Lorenz System

The Lorenz system is a system of Ordinary Differential Equation, first studied and developed by Edward Lorenz. It is notable for having Chaotic solutions for certain parameter values and initial conditions. In particular, the Lorenz attractor is a set of chaotic solutions of the Lorenz system. In popular media the "the Butterfly Effect" stems from the real-world implications of the Lorenz attractor, i.e. that in any physical system, in the absence of perfect knowledge of the initial conditions (even the minuscule disturbance of the air due to a butterfly flapping its wings), our ability to predict its future course will always fail. This underscores that physical systems can be completely deterministic and yet still be inherently unpredictable even in the absence of quantum effects. The shape of the Lorenz attractor itself, when plotted graphically, may also be seen to resemble a butterfly.

Anyway, Lorenz equations are:

$$\begin{aligned} \dot{x} &= \sigma\,(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= \dot{x}y - \beta\dot{z} \end{aligned}$$

Where, x, y, and z are variables; and, σ, ρ, and β are constant parameters greater than zero. Here, σ is called the "Prandtl number", ρ is the "Rayleigh number", and β is some quantity without a name.

Now, basing up on above equations, we will visualize Lorenz Attractor in our next context, but before that, let's understand what is an attractor.

In the mathematical field of dynamical system, an attractor is a set of numerical values toward which a system tends to evolve, for a wide variety of starting conditions of the system. System values that get close enough to the attractor values remain close even if slightly disturbed. Mathematically, it is defined as follows:

Let t represent time and let $f(t, \circ)$ be a function which specifies the dynamics of the system. Then, an attractor is a subset A of the phase space characterized by the following three conditions:

- A is *forward invariant* under f: if a is an element of A then so is $f(t,a)$, for all $t > 0$.
- There exists a neighbourhood of A, called the basin of attraction for A and denoted $B(A)$, which consists of all points b that "enter A in the limit $t \to \infty$". More formally, $B(A)$ is the set of all points b in the phase space with the following property:
  For any open neighbourhood N of A, there is a positive constant T such that $f(t,b) \in N$ for all real $t > T$.

- There is no proper (non-empty) subset of A having the first two properties.

Visualize Lorenz Attractor

Now, we are entering the core of the dissertation, i.e., the visualization of Lorenz attractor. We will chunk this topic into following blocks.

A. Logic behind the Code

In this part we will understand the working principle of our code. As per from our previous discussion, I think it is now clear that, to get the attractor, we need the tuple of (x, y, z) co-ordinates using the Lorenz Equation. For that purpose, we will simplify the equation in following manner, i.e.:

$$dx = [\ \sigma\ (y - x)\ ]\ dt$$
$$dy = [\ \rho x - y - xz\ ]\ dt$$
$$dz = [\ \dot{x}y - \beta\dot{z}\ \ ]\ dt$$

As we can see, there are 2 sets of parameters now, which are location parameters, (namely dx, dy, dz) and temporal parameter (i.e., dt). So, we have to initialize that first. Here, one thing I have to mention, that is, dt is fixed and can be used as framing interval through the entire simulation.

Now, after parameters, lets coming to the constants. Here all of the constants will be user input and can take any value greater than zero. So, it is better to add a slider for betterment of the input.

Now, after planning the working procedure, lets add the file hierarchy of our project. Since, I have chosen to use GLSL i.e., OpenGL Shading Language for the simulation, So I need an index.html file to mark-up the result and couple .js file to control the entire system.

Also, I have made custom .vert and .frag files, which are the vertex and fragment shaders for GLSL so that I can render the entire program in 3-Dimension.

B. Driver Code

Since, the entire project consists of more than 5000 lines of code, I have decided to upload the repository into GitHub and here is the link of the entire program:

https://github.com/yoursamlan/lavis.

[Anyway, the program is licensed under GNU 2.0 and MIT by Amlan Saha Kundu]

Now, lets drive into the driver code. The code is self-explanatory though I have added comments to the code for better understanding.

```
/**
 * @param {canvas} HTMLCanvasElement
 * @returns {Lorenz}
 */
function Lorenz(canvas) {
    var gl = canvas.getContext('webgl') ||
            canvas.getContext('experimental-webgl');
    if (gl == null)
        throw new Error('Could not create WebGL context.');
    this.gl = gl;
    gl.clearColor(0.1, 0.1, 0.1, 1);
    gl.enable(gl.BLEND);
    gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);

    this.params = {
        sigma: 10,
        beta: 8 / 3,
        rho: 28,
        step_size: 0.002,
        steps_per_frame: 3,
        paused: false
    };
    this.display = {
        scale: 1 / 25,
        rotation: [1.65, 3.08, -0.93],
        rotationd: [0, 0, 0],
        translation: [0, 0.075, 1.81],
        draw_heads: false,
        damping: true,
        _length: 512 // change through length getter/setter
    };

    this.solutions = [];
    this.tail = new Float32Array(0);
    this.tail_buffer = gl.createBuffer();
    this.tail_index = 0;
    this.tail_colors = new Float32Array(0);
    this.tail_colors_buffer = gl.createBuffer();
    var length = this.display._length;
    this.tail_index_buffer = Lorenz.create_index_array(gl, length);
    this.tail_element_buffer = Lorenz.create_element_array(gl, length);
    this.head = new Float32Array(0);
    this.head_buffer = gl.createBuffer();
    this.tail_length = new Float32Array(0);
    this.programs = {};
    var shaders = [
        'glsl/project.vert',
        'glsl/tail.vert',
        'glsl/tail.frag',
        'glsl/head.vert',
        'glsl/head.frag'
    ];
    Lorenz.fetch(shaders, function(project, tail_v, tail_f, head_v, head_f) {
        this.programs.tail = Lorenz.compile(gl, project + tail_v, tail_f);
        this.programs.head = Lorenz.compile(gl, project + head_v, head_f);
        /* Both use two attrib arrays, so just turn them on now. */
        gl.enableVertexAttribArray(0);
        gl.enableVertexAttribArray(1);
        this.ready = true;
    }.bind(this));

    this.frame = 0;
    this.fps = 0;
    this.accum = 0;
    this.second = Math.floor(Date.now() / 1000);
    this.ready = false;
}

/**
 * Fetch the content for each URL and invoke the callback with the results.
 * @param {String[]} urls
 * @param {Function} callback called with one argument per URL
 * @returns {Array} array that will contain the results
 */
Lorenz.fetch = function(urls, callback) {
    var results = [];
    var countdown = urls.length;
    for (var i = 0; i < urls.length; i++) {
        results.push(null);
        (function(i) {
            var xhr = new XMLHttpRequest();
            xhr.open('GET', urls[i], true);
            xhr.onload = function() {
                results[i] = xhr.responseText;
                if (--countdown === 0)
                    callback.apply(results, results);
            };
            xhr.send();
        }(i));
    }
    return results;
```

```
};

/**
 * @param {WebGLRenderingContext} gl
 * @param {string} vert
 * @param {string} frag
 * @returns {Object}
 */
Lorenz.compile = function(gl, vert, frag) {
    var v = gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(v, vert);
    var f = gl.createShader(gl.FRAGMENT_SHADER);
    gl.shaderSource(f, frag);
    gl.compileShader(v);
    if (!gl.getShaderParameter(v, gl.COMPILE_STATUS))
        throw new Error(gl.getShaderInfoLog(v));
    gl.compileShader(f);
    if (!gl.getShaderParameter(f, gl.COMPILE_STATUS))
        throw new Error(gl.getShaderInfoLog(f));
    var p = gl.createProgram();
    gl.attachShader(p, v);
    gl.attachShader(p, f);
    gl.linkProgram(p);
    if (!gl.getProgramParameter(p, gl.LINK_STATUS))
        throw new Error(gl.getProgramInfoLog(p));
    var result = {
        program: p,
        attrib: {},
        uniform: {}
    };
    var nattrib = gl.getProgramParameter(p, gl.ACTIVE_ATTRIBUTES);
    for (var a = 0; a < nattrib; a++) {
        var name = gl.getActiveAttrib(p, a).name;
        var location = gl.getAttribLocation(p, name);
        result.attrib[name] = location;
    }
    var nuniform = gl.getProgramParameter(p, gl.ACTIVE_UNIFORMS);
    for (var u = 0; u < nuniform; u++) {
        name = gl.getActiveUniform(p, u).name;
        location = gl.getUniformLocation(p, name);
        result.uniform[name] = location;
    }
    return result;
};

/**
 * @returns {WebGLBuffer}
 */
```

```
Lorenz.create_element_array = function(gl, length) {
    var data = new Uint16Array(length * 2);
    for (var i = 0; i < data.length; i++)
        data[i] = (length * 2 - i - 1) % length;
    var buffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, buffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, data, gl.STATIC_DRAW);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);
    return buffer;
};

/**
 * @returns {WebGLBuffer}
 */
Lorenz.create_index_array = function(gl, length) {
    var data = new Float32Array(length * 2);
    for (var i = 0; i < data.length; i++)
        data[i] = (length * 2 - i - 1) % length;
    var buffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);
    gl.bindBuffer(gl.ARRAY_BUFFER, null);
    return buffer;
};

/**
 * @returns {number[3]}
 */
Lorenz.generate = function() {
    return [
        (Math.random() - 0.5) * 50,
        (Math.random() - 0.5) * 50,
        (Math.random() - 0.5) * 50,
    ];
};

/**
 * @returns {number[3]}
 */
Lorenz.color = function(i) {
    var colors = [
        0x8d, 0xd3, 0xc7,
        0xff, 0xff, 0xb3,
        0xbe, 0xba, 0xda,
        0xfb, 0x80, 0x72,
        0x80, 0xb1, 0xd3,
        0xfd, 0xb4, 0x62,
        0xb3, 0xde, 0x69,
        0xfc, 0xcd, 0xe5,
        0xd9, 0xd9, 0xd9,
        0xbc, 0x80, 0xbd,
        0xcc, 0xeb, 0xc5,
        0xff, 0xed, 0x6f,
        0xff, 0xff, 0xff
    ];
    var base = (i * 3) % colors.length;
    return colors.slice(base, base + 3).map(function(x) { return x / 255; });
};


/**
 * Update s to the next Lorenz state using RK4.
 * Performs no allocations and hopefully JITs very effectively.
 * @param {!number[3]} s
 * @param {!number}    dt
 * @param {!number}    σ
 * @param {!number}    β
 * @param {!number}    ρ
 * @returns {undefined}
 */
Lorenz.lorenz = function(s, dt, σ, β, ρ) {
    function dx(x, y, z) { return σ * (y - x); }
    function dy(x, y, z) { return x * (ρ - z) - y; }
    function dz(x, y, z) { return x * y - β * z; }

    var x = s[0];
    var y = s[1];
    var z = s[2];

    var k1dx = dx(x, y, z);
    var k1dy = dy(x, y, z);
    var k1dz = dz(x, y, z);

    var k2x = x + k1dx * dt / 2;
    var k2y = y + k1dy * dt / 2;
    var k2z = z + k1dz * dt / 2;

    var k2dx = dx(k2x, k2y, k2z);
    var k2dy = dy(k2x, k2y, k2z);
    var k2dz = dz(k2x, k2y, k2z);

    var k3x = x + k2dx * dt / 2;
    var k3y = y + k2dy * dt / 2;
    var k3z = z + k2dz * dt / 2;

    var k3dx = dx(k3x, k3y, k3z);
    var k3dy = dy(k3x, k3y, k3z);
    var k3dz = dz(k3x, k3y, k3z);

    var k4x = x + k3dx * dt;
    var k4y = y + k3dy * dt;
    var k4z = z + k3dz * dt;

    var k4dx = dx(k4x, k4y, k4z);
    var k4dy = dy(k4x, k4y, k4z);
    var k4dz = dz(k4x, k4y, k4z);

    s[0] = x + (k1dx + 2*k2dx + 2*k3dx + k4dx) * dt / 6;
    s[1] = y + (k1dy + 2*k2dy + 2*k3dy + k4dy) * dt / 6;
    s[2] = z + (k1dz + 2*k2dz + 2*k3dz + k4dz) * dt / 6;
};

/**
 * Update the tail WebGL buffer between two indexes.
 * @param {number} a, with a <= b
 * @param {number} b
 */
Lorenz.prototype._update = function(a, b) {
    var gl = this.gl;
    var length = this.display._length;
    var buffer = this.tail.buffer;
    gl.bindBuffer(gl.ARRAY_BUFFER, this.tail_buffer);
    if (a == 0 && b == length - 1) {
        gl.bufferSubData(gl.ARRAY_BUFFER, 0, this.tail);
    } else {
        var sublength = b - a + 1;
        for (var s = 0; s < this.solutions.length; s++)  {
            var offset = s * 3 * length * 4 + 3 * a * 4;
            /* As far as I can tell, this buffer view is optimized out.
             * Therefore no allocation actually happens. Whew!
             */
            var view = new Float32Array(buffer, offset, sublength * 3);
            gl.bufferSubData(gl.ARRAY_BUFFER, offset, view);
        }
    }
};

/**
 * Advance the system state by one frame.
 * @returns {Lorenz} this
 */
Lorenz.prototype.step = function() {
    if (!this.ready)
        return this;
```

```
if (!this.params.paused) {
    var σ = this.params.sigma;
    var β = this.params.beta;
    var ρ = this.params.rho;
    var dt = this.params.step_size;
    var length = this.display._length;
    var tail = this.tail;
    var start_index = this.tail_index;
    var stop_index = 0;
    for (var s = 0; s < this.params.steps_per_frame; s++) {
        var tail_index = this.tail_index;
        this.tail_index = (this.tail_index + 1) % length;
        for (var i = 0; i < this.solutions.length; i++)  {
            Lorenz.lorenz(this.solutions[i], dt, σ, β, ρ);
            var base = i * length * 3 + tail_index * 3;
            tail[base + 0] = this.solutions[i][0];
            tail[base + 1] = this.solutions[i][1];
            tail[base + 2] = this.solutions[i][2];
            var next = this.tail_length[i] + 1;
            this.tail_length[i] = Math.min(next, length);
        }
        stop_index  = tail_index;
    }
    if (stop_index >= start_index) {
        this._update(start_index, stop_index);
    } else {
        this._update(start_index, length - 1);
        this._update(0, stop_index);
    }
}
this.display.rotation[0] += this.display.rotationd[0];
this.display.rotation[1] += this.display.rotationd[1];
this.display.rotation[2] += this.display.rotationd[2];
if (this.display.damping) {
    var damping = 0.96;
    this.display.rotationd[0] *= damping;
    this.display.rotationd[1] *= damping;
    this.display.rotationd[2] *= damping;
}
this.frame++;
var second = Math.floor(Date.now() / 1000);
if (second !== this.second) {
    this.fps = this.accum;
    this.accum = 1;
    this.second = second;
} else {
    this.accum++;
}
```

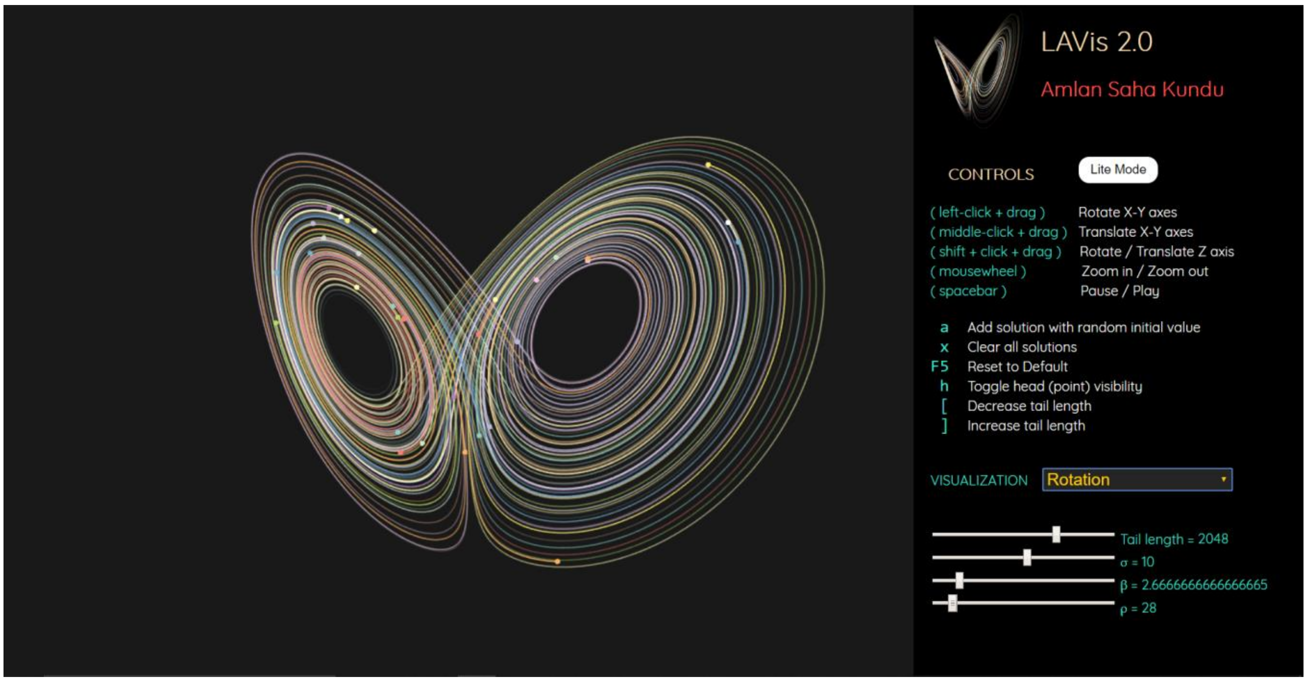
```
    return this;
};

/**
 * Renders the current state to the associated WebGL canvas.
 * @returns {Lorenz} this
 */
Lorenz.prototype.draw = function() {
    if (!this.ready)
        return this;

    var gl = this.gl;
    var width = gl.canvas.clientWidth;
    var height = gl.canvas.clientHeight;
    if (gl.canvas.width != width || gl.canvas.height != height) {
        gl.canvas.width = width;
        gl.canvas.height = height;
        gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);
    }
    gl.clear(gl.COLOR_BUFFER_BIT);

    var count = this.solutions.length;
    if (count == 0)
        return this;

    var aspect = gl.canvas.width / gl.canvas.height;
    var length = this.display._length;
    var scale = this.display.scale;
    var rotation = this.display.rotation;
    var translation = this.display.translation;
    var rho = this.params.rho;
    var start = (this.tail_index - 1 + length) % length;

    gl.useProgram(this.programs.tail.program);
    var attrib = this.programs.tail.attrib;
    var uniform = this.programs.tail.uniform;
    gl.bindBuffer(gl.ARRAY_BUFFER, this.tail_index_buffer);
    gl.vertexAttribPointer(attrib.index, 1, gl.FLOAT, false, 0,
                           (length - start - 1) * 4);
    gl.uniform1f(uniform.aspect, aspect);
    gl.uniform1f(uniform.scale, scale);
    gl.uniform3fv(uniform.rotation, rotation);
    gl.uniform3fv(uniform.translation, translation);
    gl.uniform1f(uniform.rho, rho);
    gl.uniform1f(uniform.max_length, length);
    gl.bindBuffer(gl.ARRAY_BUFFER, this.tail_buffer);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.tail_element_buffer);
    for (var i = 0; i < count; i++) {
        var r = this.tail_colors[i * 3 + 0];
        var g = this.tail_colors[i * 3 + 1];
        var b = this.tail_colors[i * 3 + 2];
        var offset = i * length * 4 * 3;
        gl.uniform3f(uniform.color, r, g, b);
        gl.uniform1f(uniform.tail_length, this.tail_length[i]);
        gl.vertexAttribPointer(attrib.point, 3, gl.FLOAT, false, 0, offset);
        gl.drawElements(gl.LINE_STRIP, length, gl.UNSIGNED_SHORT,
                        (length - start - 1) * 2);
    }

    if (this.display.draw_heads) {
        gl.useProgram(this.programs.head.program);
        attrib = this.programs.head.attrib;
        uniform = this.programs.head.uniform;
        for (var s = 0; s < count; s++) {
            var base = s * length * 3 + start * 3;
            this.head[s * 3 + 0] = this.tail[base + 0];
            this.head[s * 3 + 1] = this.tail[base + 1];
            this.head[s * 3 + 2] = this.tail[base + 2];
        }
        gl.bindBuffer(gl.ARRAY_BUFFER, this.head_buffer);
        gl.bufferSubData(gl.ARRAY_BUFFER, 0, this.head);
        gl.vertexAttribPointer(attrib.point, 3, gl.FLOAT, false, 0, 0);
        gl.bindBuffer(gl.ARRAY_BUFFER, this.tail_colors_buffer);
        gl.vertexAttribPointer(attrib.color, 3, gl.FLOAT, false, 0, 0);
        gl.uniform1f(uniform.aspect, aspect);
        gl.uniform1f(uniform.scale, scale);
        gl.uniform3fv(uniform.rotation, rotation);
        gl.uniform3fv(uniform.translation, translation);
        gl.uniform1f(uniform.rho, rho);
        gl.drawArrays(gl.POINTS, 0, count);
    }

    return this;
};

/**
 * Adjust all buffer sizes if needed.
 */
Lorenz.prototype._grow_buffers = function() {
    function next2(x) {
        return Math.pow(2, Math.ceil(Math.log(x) * Math.LOG2E));
    }
    var gl = this.gl;
    var count = next2(this.solutions.length);
    var length = this.display._length;
    if (this.tail.length < count * length * 3) {
        var old_tail = this.tail;
        this.tail = new Float32Array(count * length * 3);
        this.tail.set(old_tail);
        gl.bindBuffer(gl.ARRAY_BUFFER, this.tail_buffer);
        gl.bufferData(gl.ARRAY_BUFFER, count * length * 4 * 3, gl.DYNAMIC_DRAW);
        this._update(0, length - 1);
    }
    if (this.tail_length.length < count) {
        var old_tail_length = this.tail_length;
        this.tail_length = new Float32Array(count);
        this.tail_length.set(old_tail_length);
    }
    if (this.tail_colors.length < count * 3) {
        this.tail_colors = new Float32Array(count * 3);
        for (var i = 0; i < this.tail_colors.length; i++) {
            var color = Lorenz.color(i);
            this.tail_colors[i * 3 + 0] = color[0];
            this.tail_colors[i * 3 + 1] = color[1];
            this.tail_colors[i * 3 + 2] = color[2];
        }
        gl.bindBuffer(gl.ARRAY_BUFFER, this.tail_colors_buffer);
        gl.bufferData(gl.ARRAY_BUFFER, count * 4 * 3, gl.STATIC_DRAW);
        gl.bufferSubData(gl.ARRAY_BUFFER, 0, this.tail_colors);
    }
    if (this.head.length < count * 3) {
        // No copy needed since it's always set right before draw.
        this.head = new Float32Array(count * 3);
        gl.bindBuffer(gl.ARRAY_BUFFER, this.head_buffer);
        gl.bufferData(gl.ARRAY_BUFFER, count * 3 * 4, gl.DYNAMIC_DRAW);
    }
};

/**
 * Add a new solution to the system.
 * @param {number[3]} s
 * @returns {Lorenz} this
 */
Lorenz.prototype.add = function(s) {
    var gl = this.gl;
    var length = this.display._length;
    this.solutions.push(s.slice(0));
    this._grow_buffers();
    return this;
};

/**
 * Change the tail lengths.
 * @param {number} length
 * @returns {Lorenz} this
 */
Lorenz.prototype._trim = function(length) {
    function mod(x, y) { // properly handles negatives
        return x - y * Math.floor(x / y);
    }
    var count = this.solutions.length;
    var oldlength = this.display._length;
    this.display._length = length;
    var old_tail = new Float32Array(this.tail.length);
    old_tail.set(this.tail);
    this._grow_buffers();
    var actual = Math.min(length, oldlength);
    for (var s = 0; s < count; s++) {
        for (var n = 0; n < actual; n++) {
            var i = mod(this.tail_index - n - 1, oldlength);
            var o = actual - n - 1;
            var obase = s * length * 3 + o * 3;
            var ibase = s * oldlength * 3 + i * 3;
            this.tail[obase + 0] = old_tail[ibase + 0];
            this.tail[obase + 1] = old_tail[ibase + 1];
            this.tail[obase + 2] = old_tail[ibase + 2];
        }
        this.tail_length[s] = Math.min(this.tail_length[s], actual);
    }
    this.tail_index = actual % length;
    this.tail_index_buffer = Lorenz.create_index_array(this.gl, length);
    this.tail_element_buffer = Lorenz.create_element_array(this.gl, length);
    this._update(0, length - 1);
    return this;
};

/**
 * Remove all solutions.
 * @returns {Lorenz} this
 */
Lorenz.prototype.empty = function() {
    this.solutions = [];
    this.tail = new Float32Array(0);
    this.tail_index = 0;
    this.tail_colors = new Float32Array(0);
    this.head = new Float32Array(0);
    this.tail_length = new Float32Array(0);
    return this;
};

Object.defineProperty(Lorenz.prototype, 'length', {
    get: function() {
        return this.display._length;
    },
    set: function(v) {
        this._trim(v);
        return this.display._length;
    }
});

/**
 * Initialize and start running a demo.
 * @returns {Lorenz}
 */
Lorenz.run = function(canvas) {
    var lorenz = new Lorenz(canvas);
    for (var i = 0; i < 13; i++)
        lorenz.add(Lorenz.generate());
    function go() {
        lorenz.step();
        lorenz.draw();
        requestAnimationFrame(go);
    }
    requestAnimationFrame(go);
    return lorenz;
};
```

C. Execution

I gave the name of the program as LAVis (Lorenz Attractor Visualizer), which can be used by clicking here:

https://yoursamlan.github.io/lavisloader/

Let's see the screenshot of our executions:



LAVis 2.0

Amlan Saha Kundu

CONTROLS Lite Mode

(left-click + drag)	Rotate X-Y axes
(middle-click + drag)	Translate X-Y axes
(shift + click + drag)	Rotate / Translate Z axis
(mousewheel)	Zoom in / Zoom out
(spacebar)	Pause / Play

a	Add solution with random initial value
x	Clear all solutions
F5	Reset to Default
h	Toggle head (point) visibility
[	Decrease tail length
]	Increase tail length

VISUALIZATION Rotation

Tail length = 2048

σ = 10

β = 2.6666666666666665

ρ = 28

Allow me to give a brief description about this app:

On the right-hand side, there is the control centre, where the user can change the values of the constant parameters and can perform other controls. On the left-hand side, there is the canvas, where the user can see the output in real-time. Now, let's discuss its feature briefly:

1. It can capable of generate 3D model with rotation
2. User can visualize the result instantly, no re-run/ re-fresh is required
3. User can customizations the result as his wish
4. No hefty hardware, as well as specialised software is required. It can run on any browser on any operating system, varies from mobile to desktop
5. The software is completely free to use.

Conclusion

Throughout the entire project I'm trying to give a clear and understandable description of Chaos and hope that the thesis is approachable and understandable to any person with minimum knowledge and maximum interest to this field. My main objective to this project was to create an application that is free and can be run on any system despite of its hardware, and I'm satisfied with my work.

It is often said, *"A visual explanation can create pleasure and understanding at the same time"*, and throughout the entire project I keep emphasis to it. I think the simulation of double pendulum as well as the Lorenz Attractor Visualizer application will certainly attract the reader to dig deeper into the subject. Also, this dissertation has helped to discover a new field of mathematics, that was completely unknown to me. The goal of my thesis or dissertation was to familiarize myself and the reader with chaos theory with the visualization of relatively simple system, viz. Lorenz System. Now, we have acquired the tools, necessary to examine such chaotic systems, we can move onward to further educate ourselves in more advanced chaos theory.

Bibliography

- Chaos and Fractals: An Elementary Introduction by David P. Feldman
- Milnor, J. (1985). "On the Concept of Attractor: 177–195
- A Study of Lorenz Attractor and its Chaotic Behaviour by Benjamin L. Kurian
- https://www.complexityexplorer.org/
- https://www.youtube.com/watch?v=fDek6cYijxI&t=4s
- https://www.myphysicslab.com/pendulum/double-pendulum-en.html
- https://www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1_aw
- https://en.wikipedia.org/
- https://github.com/