CS142 Coursework 2

Mohammed Rafi

May 2020

Contents

1 Introduction

In this report I will discuss my methodology when creating my visualisation for my CS142 coursework which was to a system or algorithm of my choice.

I decided to visualize a system of differential equations called the Lorenz system which was first studied by a mathematician and meteorologist called Edward Lorenz who also was a leading contributor to the branch of mathematics called chaos theory[1].

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y - x)\\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho - z) - y\\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy - \beta z\end{aligned}$$

Figure 1: Lorenz System of differential equations

The equations shown in figure 1 show the rate of change of 3 variables, x, y and z over time t with given constants σ, ρ and β. The equations were derived from a simplified model of convection in the earths atmosphere, it also arises naturally in models lasers and dynamos[2].

For specific values of σ, ρ and β when the equations are plotted you can get beautiful patterns shown in figure 2.

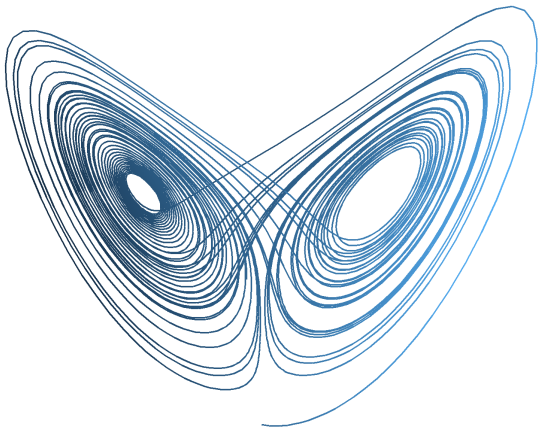

Figure 2: Image example of solution of Lorenz system where $\sigma = 10, \rho = 28$ and $\beta = \frac{8}{3}$ [3]

2 Design

When envisioning how my visualisation would look like I decided to look into existing visualisations of the Lorenz system. Most existing visualisations such as the one shown in figure 2 of the system was not interactive and showed a static image of the final product so I wanted the user to be able to interact with the system. An interactive simulation of the Lorenz attractor by Hendrik Wernecke[4] only allowed users to change the value of ρ and shows the graph in 2D (When there are 4 variables) so in my visualisation I would like to create a 3D representation where users can adjust the values of σ, ρ and β

As I was visualising 4 variables x, y, z and t in 3 dimensions I decided I wanted the color of the line to be dependent on time even if colour is not very accurate way to convey information[5] I believe that it was the cleanest way to represent the change in time without the use of other graphical methods such as volume.

In terms of interaction the main styles I wanted to use were Instructions in the form of buttons and Manipulation[6].I wanted the user to be able pause and play the graphing of the system and adjust the values of σ, ρ and β to allow the user to create many different visualisations of the system. I also wanted the user to be able to zoom, pan and rotate around the graph to add a extra layer to the interactivity of the visualisation.

Figure 3 is how I envisioned the layout of my visualisation with each square / rectangle representing a zone in my visualisation:

- Simulation: Main area where the graph is being drawn
- Playback Settings: Area where reset, pause and speed button are located
- Value adjuster: Area where sliders to adjust constant values are located
- Presets: Area where buttons that graph the system with different example values of constants.

With my design envisioned, I now needed to create it in Processing.

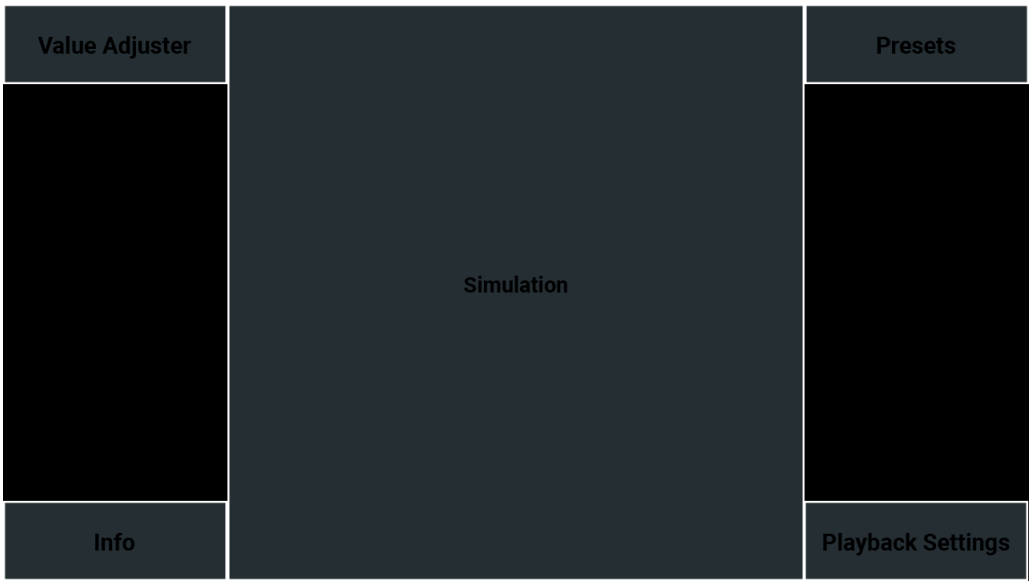

Figure 3: Prototype of my visualisation layout

3 Implementation

As most of the code has been heavily commented, I'll briefly describe the two sections I needed to code, how to simulate the graphing of the system and the interaction of the visualisation.

Simulation

To create the visualisation which would graph the Lorenz system of equations I first had to create a method `Lorenz()`, as shown in figure 4 which would run the system of equations and save the points which would be drawn.

It was fairly easy to get the value of each point as I didn't have to solve the equations to get each point but just increment the variables by their corresponding differential value. I then added the new x, y and z into an ArrayList of PVectors as a PVector can store a point of 3 dimensions.

The method would keep adding points into the ArrayList until we reached our max number of iterations or the visualisation has been paused (which will be discussed in the interaction section).

```
public void Lorenz() {
  //For a accurate representation of what the system will look like
  //we must use a small value of t to get the value of each point
  float dt = 0.005;
  float dx = (sigma * (y - x))*dt;
  float dy = (x * (rho - z) - y)*dt;
  float dz = (x * y - beta * z)*dt;

  // Checks whether the pause button has been pressed or our
  //current value of t is LEQ then our chosen max value of t. //
  if ((!stop_state) && (t <= t_max)) {
    // Increment our values of x, y, z and t with the corresponding
    //rate of change and add the coordinates to our ArrayList.
    x += dx;
    y += dy;
    z += dz;
    t += dt;
    points.add(new PVector(x, y, z));
  }
}
```

Figure 4: Lorenz System Method

With my array of points created I now needed to plot them, this was done by using `beginShape()` and looping through the array and creating a vertex between each point to make a continuous line between all the points, as shown in figure 5.

```
// Loops and draws a simultaneous line which connects all points in the ArrayList.//
  scale(5);
  noFill(); // Creates a line compared to filling the empty space with colour
  float color_t = 0;
  beginShape();
  for (int i = 0; i < points.size(); i++) {
    colorMode(RGB);
    strokeWeight(0.2);
    stroke(color(map_color(color_t)));
    vertex(points.get(i).x, points.get(i).y, points.get(i).z);
    color_t += 0.005; // Increment color_t for each frame drawn
  }
  endShape();
```

Figure 5: Code which plots the points

As I wanted to change the color of the point on the line based on the current value of t, I created a method `map_color()`, as shown in figure 6 which returns a colour the point should take by using `LerpColor()` and `map()` to get a interpolated colour between two chosen colours by mapping the current value of t and the max value of t between 0 and 1.

```
/**
 Calculates and returns a colour between two predetermined colours based on the current value of t,
 which is mapped to be turned into a float between 0 and 1 to be compatible with lerpColor().
 */
int map_color(float col) {
  return lerpColor(#F56200, #FFFFFF, map(col, 0, t_max, 0, 1));
}
```

Figure 6: mapcolor() Method

Interaction

For interactivity I decided to use two libraries, peasycam[7] which gave me the ability to move the camera based on mouse movements and controlP5[8] which automates the process of creating buttons and sliders. I created images in Photoshop for my buttons as I didn't like the look of controlP5 default buttons, however this lead to the buttons showing no feedback when hovered upon.

Playback

- "Normal Mode" and "Quick Mode" - As the speed of which the points are drawn are dependent of framerate of the visualisation, I decided to create a buttons which would change the framerate of the visualisation using the `frameRate()` method, changing the speed of which the points are drawn.
- "Reset" - This button would simply set the values of x, y, z and t back to its original values, clear the ArrayList of points and reset the cameras position.
- "Pause" and "Play" - This button would change a boolean value which is checked in the `Lorenz()` method and would only increment the variables and add to the array if the visualisation isn't paused.

Presets

These buttons would change the values of σ, ρ and β to their corresponding preset values and rerun the visualisation with these values.

Value Adjuster

The sliders would change the values σ, ρ and β to the users desired values. As I didn't want to change the values which were currently being used in the running simulation the sliders changed temporary values which would only switch the real values once the "Update" button was pressed.

Constraints

As suggested by Donald Norman[9], I added some constraints to the cameras movement such as the maximum and minimum zoom distance using `setMaximumDistance()` and `setMinimumDistance()` respectively as the camera would clip through the graph if zoomed in to much or the graph would disappear if zoomed out to far. I also used `keyPressed()` to check if the key "r" is being pressed to reset the cameras position.

Info

My information box was simple to create as I only needed to draw a box and text for each variable. As these variables change every frame I had to call the method within `draw()` to show the change, I also rounded each number to 3 decimal places to reduce visual clutter as the constantly changing numbers may take the users attention compared to the graph.

To stop all the GUI from moving with the camera I used peasycams HUD methods and set `cp5.setAutoDraw()` to stop the buttons and sliders from drawing in the 3D scene.

4 Resulting visualisation

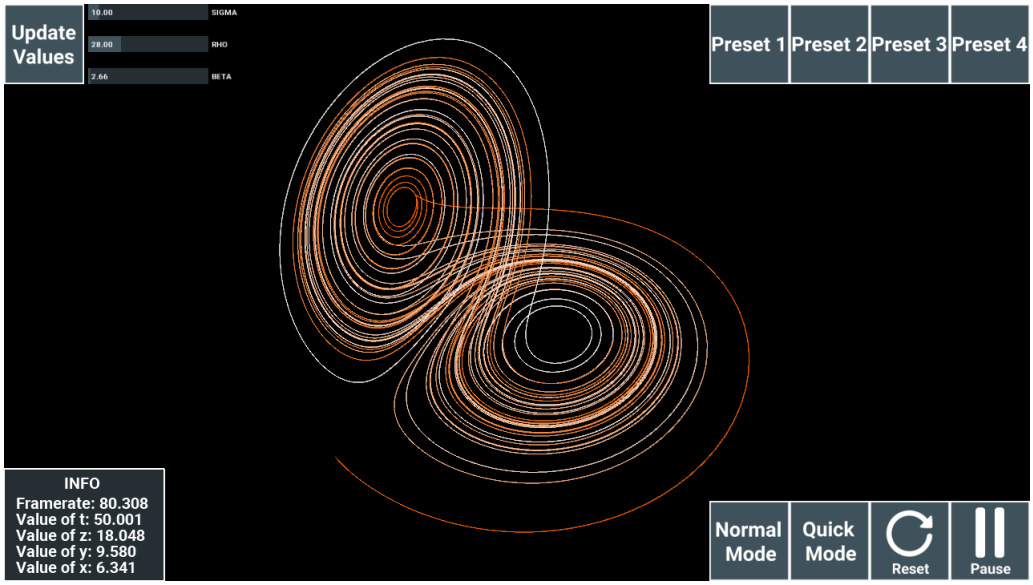

Figure 7: Screenshot 1 of final visualisation

Figure 7 shows my visualisation which shows the Lorenz system where $\sigma = 10, \rho = 28$ and $\beta = \frac{8}{3}$. The graph has a butterfly shape with the lines spiraling inwards in each "wing".

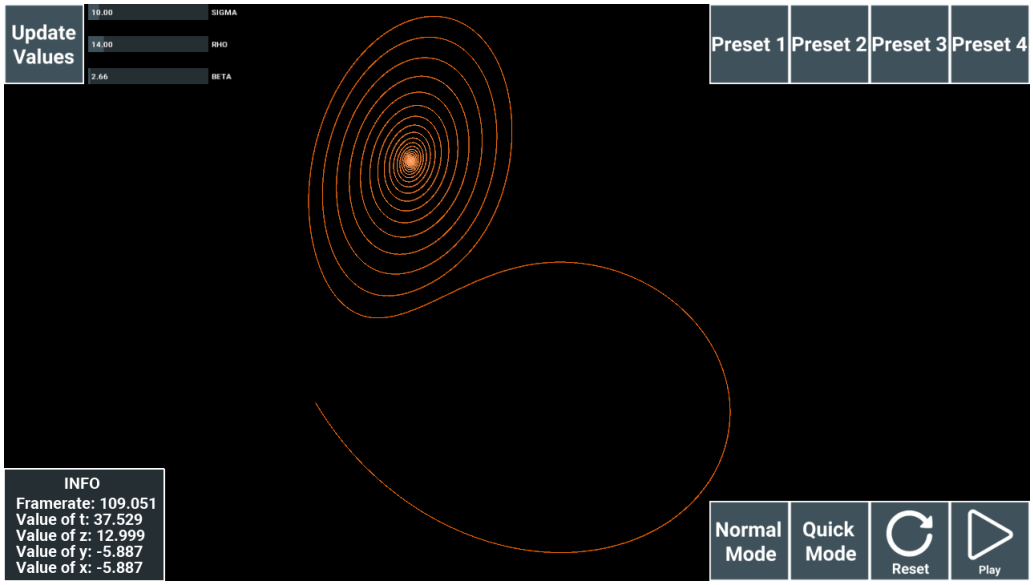

Figure 8: Screenshot 2 of final visualisation

Figure 8 shows the Lorenz system where $\sigma = 10, \rho = 14$ and $\beta = \frac{8}{3}$ which gives the final graph a much different shape to figure 8, showing how a small change can completely change the course of the system.

5 Evaluation

To evaluate my visualisation I decided to use some predetermined frameworks and highlighted how they stacked up against them.

4 Criteria

Using the 4 criteria evaluation[10] I believe that my final visualisation was successful at meeting these criteria:

- Informative: I believe that the visualisation is informative as it graphs how the Lorenz system changes over time with different values of σ, ρ and β.
- Efficient: As I visualised the system using a simple line that changes color based on time and showed an overview of the current points coordinates, I believe it is efficient at conveying how the Lorenz system changes through time and different constant values.
- Aesthetic: I used a consistent typeface, muted colour palette and layout for my GUI, allowing the Lorenz attractor to take center stage.
- Novel: As this is quite subjective, I believe that the visualisation is interesting, whether you are interested in mathematics or not, you can appreciate the beautiful patterns created by a system of differential equations.

Mazza 5 Criteria

Using the Mazza's 5 criteria evaluation[11] method I believe that my final visualisation successfully meets their criteria:

- Functionality - The final visualisation does meet the requirements of my design brief.
- Effectiveness - I believe that the visualisation does give better knowledge to the user about the Lorenz system and how different constant values can change the final result.
- Efficiency - With the use of my "Quick Mode" button and my information text it allows the user to rapidly see how the system is changing with time.
- Usability - The interaction in my visualisation is simple as it is mostly just simple camera movements and buttons which are explained how to use in the README file, however users can easily test the interaction as it is mostly self explanatory.
- Usefulness - I believe that the visualisation does serve a purpose in showing how beautiful mathematics can be and how small changes and decisions such as changing a constant value in an equation can have lasting effects.

In terms of accessibility for colour blind people I believe that my visualisation is successful as I chose colours which shouldn't alter the perception of the Lorenz system and the fact that this visualisation can still convey the same message without the use of colour.

6 Conclusion

In conclusion I believe that I have successfully visualised the Lorenz system of equations as I met most criteria in the evaluation frameworks used and converted the design envisioned into a working product.

However there are drawbacks with my final product such as they high CPU usage when the visualisation is in "Quick Mode" and the large memory usage as we had to create an ArrayList which could hold at least 10,000 points but this could be lowered by lowering our max iterations of t

or using a accurate differential equation solver such as the Runge-Kutta 4th Order Method[12][13].

I also heavily relied on libraries, especially controlP5 to create the buttons and sliders as I struggled to keep the buttons contained in the HUD when I created my own button class.

If I were to attempt this again I would of abstracted the Lorenz System plotting and solving methods to reduce clutter,Allow users to change the amount of iterations of t within the GUI and found a way to improve the performance of the visualisation (however this could be due to Processing 3 being inefficient), change the colour of the line and attempt to create my own button, slider and camera class to remove reliance from libraries.

References

[1] Stanford Encyclopedia of Philosophy. *Chaos*, 2015 (accessed May, 2020).

[2] Paul Bourke. *The Lorenz Attractor in 3D*, 2005 (accessed May, 2020).

[3] IMPERFECT LENS. *Growing a Lorenz Butterfly in R*, 2017 (accessed May, 2020).

[4] Hendrik Wernecke. *Lorenz system - An interactive simulation of a chaotic attractor*, 2018 (accessed May, 2020).

[5] Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[6] Preece, Rogers, and Sharp. *Interaction design: beyond human-computer interaction.* Wiley, 2019.

[7] Jonathan Feinberg. *peasycam v302*, 2018 (accessed May, 2020).

[8] Andreas Schlegel. *controlP5*, 2015 (accessed May, 2020).

[9] Donald A. Norman. *The design of everyday things.* The MIT Press, 2013.

[10] Julie Steele and Noah P. N. Iliinsky. *Beautiful visualization.* OReilly, 2010.

[11] Riccardo Mazza. *Introduction to information visualization.* 2009.

[12] GeeksforGeeks. *Runge-Kutta 4th Order Method to Solve Differential Equation*, 2020 (accessed May, 2020).

[13] Marc Kjerland. *The Lorenz attractor*, 2008 (accessed May, 2020).