Fourier Drawing

Kohki Hatori

Table of Contents

Analysis

Identification of Problem

I am creating a website that takes images as input from users' devices or URLS, and approximate and draw the edges of the image with a mathematical technique called Fourier Series. I chose this problem as the first time I saw a "Fourier drawing" video online, I was astonished and intrigued by the beauty of Fourier Series, and I believe people with interest in Maths, Physics, and/or Computer Science will find it interesting as well.

Intended users

The intended users of this website are keen students or individuals who are interested in Mathematics, Physics, and/or Computer Science, as Fourier series is relatively complicated and sophisticated, yet important idea in Mathematics and Computer Science. In addition, it may as well be used as a demonstration of Fourier Series in classrooms.

Research methods

I mainly used YouTube and websites on the internet to do research. URLs to the videos and websites are in the bibliography.

Dialogue with clients

Me: What functionalities would you want on the website?

Client: I would want to see the ability to upload my own data or image to be used. Also, The ability to customize the number of vectors used in the Fourier series to adjust the level of detail in the resulting image.
Me: Okay. Do you have any suggestions on the User Interface?
Client: It would be great if it had an intuitive user interface that is easy to navigate and use, even for those who are not familiar with Fourier series or SVG images.
Me: Okay. Thank you very much.

Having spoken to the client and considering his suggestions, I concluded the requirements below.

Rationale

I have decided to use Python for backend processing and JavaScript for frontend processing for following reasons. Firstly, I am most familiar with Python and it has libraries that are required to build this website. Also, for frontend processing, there is not many other options.

Requirements

The following are the high-level functional and non-functional requirements of the solution.
FR: Functional Requirement
NFR: Non-functional Requirement

FR 1 Users should be able to easily upload an image file from their devices.
FR 2 Users should be able to control the parameters such as number of vectors and speed of rotation
FR 3 The program must be able to display the drawing on a web browser.
NFR 1 The program should be able to display circles around the vectors and vectors themselves for clearer display.
NFR 2 Users should be able to stop and resume the animation at any point.
FR 4 The API server must be able to receive a request from and respond to the frontend.
FR 5 The API server must be able to run the model with given parameters.
FR 6 The program must be able to calculate the coefficients for each vector.
FR 7 The program must be able to detect edges in images.
FR 8 The program must be able to convert a bitmapped image to an SVG image file format.
FR 9 The program must be able to get Bezier curve equations from SVG files.
FR 10 The program must be able to obtain a parametric equation from the Bezier curves in SVG files.
FR 11 The program must be able to combine multiple edges to a single edge.
FR 12 The data server must be able to receive a request from the API server and return data specified by the API server.
NFR 3 The data (coefficients, original image, and selected edges) of each drawing is stored in a text-based database.

Mathematics behind the problem

The mathematics used in this program can be briefly decomposed to:

I. Complex Fourier Series
II. Bezier Curves
III. Integration by parts, the quick technique

Complex Fourier Series:

Given that:

$f(t)$ is a complex function which, as t ranges from 0 to 1, the points on the complex plane plotted by the function draw out one cycle of a particular enclosed path that is defined by the function. (e.g., $f(t) = e^{2\pi it}$ draws out one cycle of a unit circle as t ranges from 0 to 1)

Complex Fourier Series states that $f(t)$ can be approximated with the sum of complex vectors of integer frequencies with arbitrary coefficients,

n	…	-2	-1	0	1	2	3	…
$c_n e^{2\pi in}$		$c_{-2}e^{-2*2\pi}$	$c_{-1}e^{-1*2\pi}$	$c_0 e^{0*2\pi i}$	$c_1 e^{1*2\pi i}$	$c_2 e^{2*2\pi i}$	$c_3 e^{3*2\pi i}$	

$$f(t) \approx \sum_{n=-\left\lfloor \frac{N}{2} \right\rfloor + int(N\%2==0)}^{N} c_n e^{2\pi int}$$

Such that:

$$\lim_{N\to\infty} \sum_{n=-\left\lfloor \frac{N}{2} \right\rfloor + int(N\%2==0)}^{N} c_n e^{2\pi int} \equiv f(t)$$

Where …
n: frequency
t: input parametric variable ranging between 0-1.
(The value of t can exceed 1, it will just repeat the cycle over and over again.)

Using this theory, the only unknown is each coefficient for each frequency of complex vector.
⇔ Once the coefficients are calculated, the function can be approximated using complex vectors.

In order to find the coefficients, let's start from considering the case when n = 0.

If you think about this integral …

$$\int_0^1 f(t)dt$$

$$= \int_0^1 \left(\cdots + c_{-1}e^{-1\cdot 2\pi it} + c_0 e^{0\cdot 2\pi it} + c_1 e^{1\cdot 2\pi it} + \cdots\right)dt$$

$$= \cdots + \int_0^1 c_{-1}e^{-1\cdot 2\pi it}dt + \int_0^1 c_0 e^{0\cdot 2\pi it}dt + \int_0^1 c_1 e^{1\cdot 2\pi it}dt + \cdots$$

$$= \int_0^1 c_0 e^{0\cdot 2\pi it}dt = \int_0^1 c_0 dt = c_0$$

Each integral can be thought as the average of the vector inside as the value of t ranges from 0 to 1.

Since all rotating vectors have integer frequencies, they all have an average of 0, as t goes from 0 to 1, apart from when n = 0, where the value is not varying and is a constant.

Thus, the coefficient for the complex vector of the frequency of 0 can be calculated. The reason why it can be calculated is because change of t does not have significance on the value of the 0^{th} complex vector. If we could make other complex vectors constant so that it does not vary as t varies from 0 to 1, we would be able to calculate the value of coefficient for any frequency.

Therefore, when computing c_n, you want to make the nth term constant. You can achieve this by multiplying the whole integral by $e^{-n\cdot 2\pi it}$. This manipulation kind of "shifts" each rotating vector so that the nth vector holds still.

The general formula for coefficients:

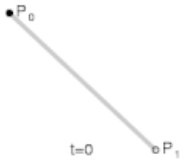
$$c_n = \int_0^1 e^{-2\pi int} f(t)dt$$

Bézier Curve:

A Bezier Curve is a parametric curve defined by a set of control points and a start and end point. The number of control points can vary.

Linear Bezier:

A Linear Bezier curve is a Bezier curve with no control points.

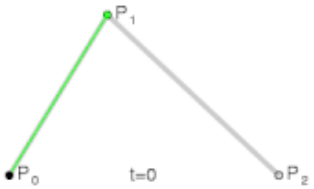


The point P for a linear Bezier curve as a function of t can be calculated using the linear interpolation function:

$$P(t) = P_0 + t(P_1 - P_0)$$

Quadratic Bezier:

A quadratic Bezier curve is a Bezier curve with one control point.



The point P for a quadratic Bezier curve as a function of t can be calculated by recursively applying the linear interpolation function, first between P0 and P1 and P1 and P2, and then bet ween the two points interpolated in the first interpolation.

$$P(t) = (1-t)[(1-t)P_0 + tP_1] + t[(1-t)P_1 + tP_2]$$

Higher-order curves:

Similarly, for higher-order Bezier curves, the number of control points increases and the number of recursive interpolation increases.

Poly-Bezier curve:

Kohki Hatori

Poly-Bezier curve is simply a Bezier curve that consists of multiple Bezier curves of which its first curve's end point is the second curve's start point, and its second curve's end point is the third curve's start point and so on.

Integration by parts, the quick technique

Using the grid method, integration by parts can be done immensely faster than the normal method.

Consider:

$f(x) = e^x x^2$

Where $h(x) = e^x$ and $g(x) = x^2$

$$\int f(x)dx$$

Writing down the integrals of $h(x)$and differentials of $g(x)$ until $g(x)$ becomes a constant.

sign	*Integrals*	*Differentials*
+	e^x	x^2
-	e^x	$2x$
+	e^x	2

The integral can be calculated by just adding the products of the left-hand side column and right-hand side column, with the alternating sings. In the above example, the solution is:

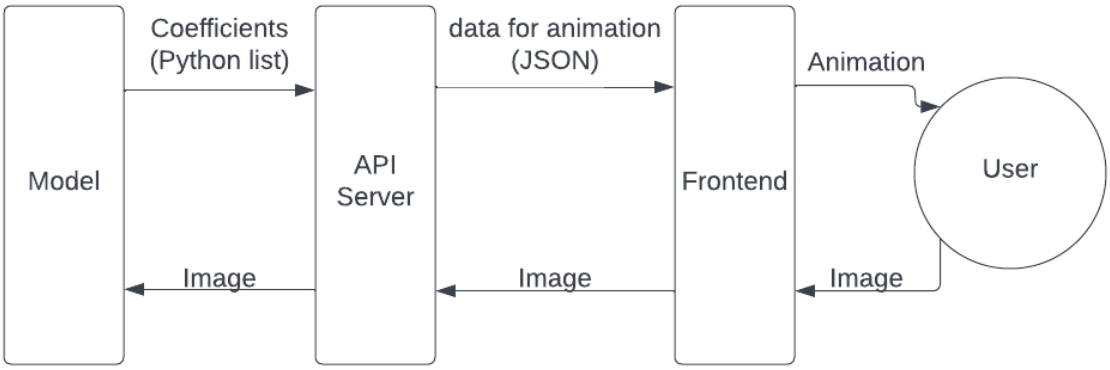
$$e^x x^2 - 2e^x x + 2e^x + c$$

API server

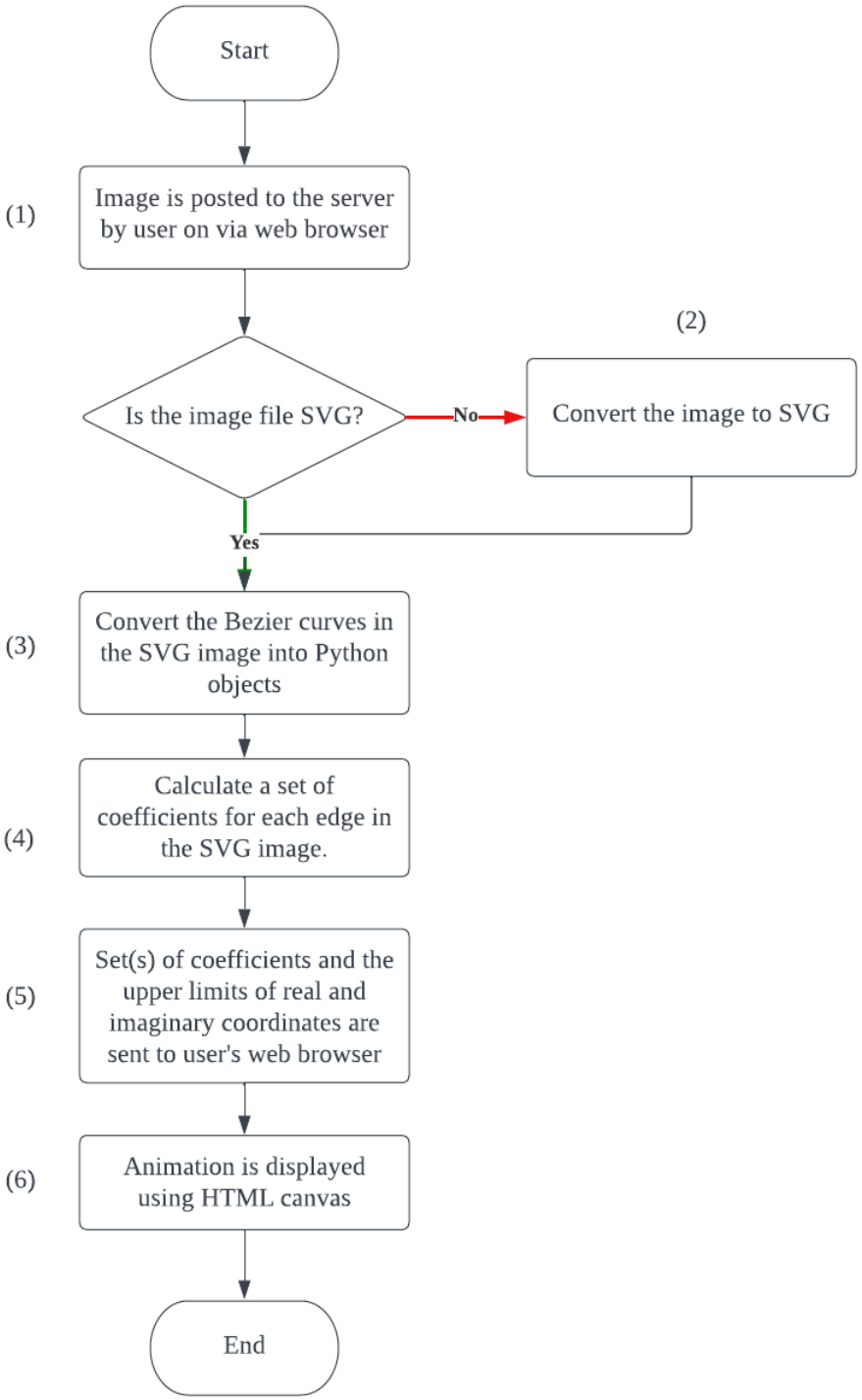
For the API server, I decided use FastAPI and uvicorn which are third-party Python libraries. Mainly because I have used them before in my programs and I am familiar with them more than any other server libraries in Python.

Documented Design

Structure



General flow of the program



1. Image posted to the server
   This is implemented using JavaScript's fetch API. The image is posted via a POST request, and the web browser waits for the drawing data (set(s) of coefficients and upper limits).

2. Image converted to SVG
   This is implemented using a third-party library called Potrace. It requires the input bitmapped image in the format of pnm.

3. Bezier curves converted to Python objects
   This is implemented using the SVG parser in the model.

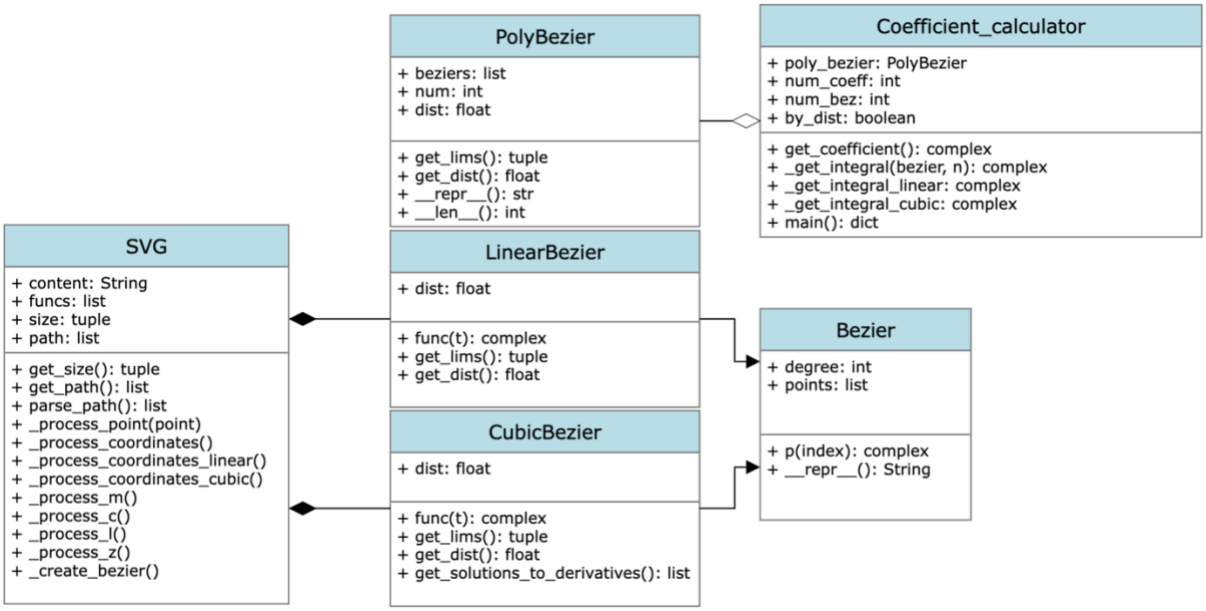
4. A set of coefficients is calculated for each edge in the SVG image
   This is implemented using the Coefficient_calculator class in the model.

5. Set(s) of coefficients and the upper limits are sent to user's web browser
   This is done using the FastAPI. It returns set(s) of coefficients and the upper limits of real and imaginary coordinates. The upper limits are used to adjust the size of the canvases on HTML.

6. Animation is displayed
   HTML canvas is used to display the animation. For mutability, two different canvases are used for animation path, and vectors and circles around them, displaying the magnitude of the vectors.

Class Diagrams

Backend:



Algorithms

Coefficient Calculator

The Coefficient Calculator computes coefficients for Poly-Bezier curves which consist of multiple Bezier curves. Each Bezier curve has a different range of t values of the parent Poly-Bezier curve while it's originally defined by its own parametric variable from 0 to 1.

Consequently, I will represent an arbitrary Bezier curve in general as $B(u)$ where u is a parametric variable varying from 0 to 1, and any arbitrary Poly-Bezier curve in general as $P(t)$ where t is a parametric variable varying from 0 to 1.

The general formula for nth coefficient is:

$$c_n = \int_0^1 e^{-2\pi i n t} f(t) dt$$

In my program's case, f(t) is the Poly-Bezier function, $P(t)$. What must to be considered is that $P(t)$ is a compound function, meaning that the as the value of t changes, the function itself changes as well.

$$P(t) = \begin{cases} B_0(u), & 0 \le t < t_1 \\ B_1(u), & t_1 \le t < t_2 \\ \vdots & \\ B_{N-1}(u), & t_{N-1} \le t < 1 \end{cases}$$

Where N is the number of Bezier curves in the Poly-Bezier curve.

Therefore, the general formula for nth coefficient becomes:

$$c_n = \int_0^{\frac{1}{N}} e^{-2\pi int} B_1\,(u)dt + \int_{\frac{1}{N}}^{\frac{2}{N}} e^{-2\pi int} B_2\,(u)dt + \cdots \int_{\frac{N-1}{N}}^{1} e^{-2\pi int} B_1\,(u)dt$$

Fortunately, only two cases need to be considered: Linear Bezier curve and Cubic Bezier curve.

In general:

$$\int_{\frac{x}{N}}^{\frac{x+1}{N}} e^{-2\pi int} B_x(u)dt$$

Where x is the order of the Bezier curve in the Poly-Bezier curves.

By definition, upper and lower limits are in terms of t, and is different to the limits of u. The limits of u is from 0 to 1.

$$u = Nt - x$$

Linear Bezier curve:

When $n \neq 0$

$$\int_{\frac{x}{N}}^{\frac{x+1}{N}} e^{-2\pi int}\{(P_1 - P_0)u + P_0\}dt$$

$$du = Ndt$$
$$dt = \frac{du}{N}$$
$$t = \frac{u+x}{N}$$
$$\frac{1}{N}\int_0^1 e^{-\frac{2\pi in(u+x)}{N}}\{(P_1 - P_0)u + P_0\}du$$

Integration by parts, using [the quick technique]

signs	*Integrals*	*Differentials*
+	$\frac{Ne^{-\frac{2\pi in(u+x)}{N}}}{-2\pi in}$	$(P_1 - P_0)u + P_0$
-	$\frac{N^2 e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^2}$	$P_1 - P_0$

$$= \frac{1}{N}\left[\frac{Ne^{-2\pi in(u+x)}}{-2\pi in}\{(P_1 - P_0)u + P_0\} + \frac{N^2 e^{-2\pi in(u+x)}}{(-2\pi in)^2}(P_1 - P_0)\right]_0^1$$
$$= \{P_1 - P_0 + P_0\}\frac{e^{-\frac{2\pi in(x+1)}{N}}}{-2\pi in} + \frac{e^{-\frac{2\pi in(x+1)}{N}}}{(-2\pi in)^2}\{N(P_1 - P_0)\}$$
$$- \left[P_0\frac{e^{-\frac{2\pi inx}{N}}}{-2\pi in} + \frac{e^{-\frac{2\pi inx}{N}}}{(-2\pi in)^2}\{N(P_1 - P_0)\}\right]$$

$$= \frac{P_1 e^{-\frac{2\pi in(x+1)}{N}} - P_0 e^{-\frac{2\pi inx}{N}}}{-2\pi in} + \{N(P_1 - P_0)\}\frac{e^{-\frac{2\pi in(x+1)}{N}} - e^{-\frac{2\pi inx}{N}}}{(-2\pi in)^2}$$

$$\frac{x+1}{N} : upper\ limit, \frac{x}{N} : lower\ limit, N : \frac{du}{dt}$$

In general:

$$\frac{P_1 e^{-2\pi in(upper\ limit)} - P_0 e^{-2\pi in(lower\ limit)}}{-2\pi in} + \left\{\frac{du}{dt}(P_1 - P_0)\right\}\frac{e^{-2\pi in(upper\ limit)} - e^{-2\pi in(lower\ limit)}}{(-2\pi in)^2}$$

Similarly, when *by_dist* is True,

Cumulative sum is the sum of the distances of previous Bezier curves.

$$Let\ cumulative\ sum\ be\ d_c, Bezier.dist\ d_b, PolyBezier.dist\ d_p.$$

$$\int_{\frac{d_c}{d_p}}^{\frac{d_c+d_b}{d_p}} e^{-2\pi int} B_x(u)dt$$

$$\int_{\frac{d_c}{d_p}}^{\frac{d_c+d_b}{d_p}} e^{-2\pi int}\{(P_1 - P_0)u + P_0\}dt$$

$$u = \frac{td_p - d_c}{d_b}$$
$$t = \frac{ud_b + d_c}{d_p}$$

$$du = \frac{d_p}{d_b}dt$$
$$dt = \frac{d_b}{d_p}du$$

$$\frac{d_b}{d_p}\int_0^1 e^{-\frac{2\pi in(ud_b+d_c)}{d_p}}\{(P_1-P_0)u+P_0\}du$$

Integration by parts, using [the quick technique]

signs	*Integrals*	*Differentials*
+	$\frac{d_p e^{-\frac{2\pi in(ud_b+d_c)}{d_p}}}{d_b(-2\pi in)}$	$(P_1-P_0)u+P_0$
-	$\frac{d_p^2 e^{-\frac{2\pi in(ud_b+d_c)}{d_p}}}{d_b^2(-2\pi in)^2}$	P_1-P_0

$$=\frac{d_b}{d_p}\left[\frac{d_p e^{-\frac{2\pi in(ud_p+d_c)}{d_p}}}{d_b(-2\pi in)}\{(P_1-P_0)u+P_0\}+\frac{d_p^2 e^{-\frac{2\pi in(ud_p+d_c)}{d_p}}}{d_b^2(-2\pi in)^2}(P_1-P_0)\right]_0^1$$

$$=\{P_1-P_0+P_0\}\frac{e^{-\frac{2\pi in(d_b+d_c)}{d_p}}}{-2\pi in}+\frac{e^{-\frac{2\pi in(d_b+d_c)}{d_p}}}{(-2\pi in)^2}\left\{\frac{d_p}{d_b}(P_1-P_0)\right\}$$
$$-\left[P_0\frac{e^{-\frac{2\pi ind_c}{d_p}}}{-2\pi in}+\frac{e^{-\frac{2\pi ind_c}{d_p}}}{(-2\pi in)^2}\left\{\frac{d_p}{d_b}(P_1-P_0)\right\}\right]$$

Kohki Hatori

$$= \frac{P_1 e^{-\frac{2\pi in(d_b+d_c)}{d_p}} - P_0 e^{-\frac{2\pi i n d_c}{d_p}}}{-2\pi in} + \left\{\frac{d_p}{d_b}(P_1 - P_0)\right\} \frac{e^{-\frac{2\pi in(d_b+d_c)}{d_p}} - e^{-\frac{2\pi i n d_c}{d_p}}}{(-2\pi in)^2}$$

This can be derived using the general formula:

$$\frac{P_1 e^{-2\pi in(upper\ limit)} - P_0 e^{-2\pi in(lower\ limit)}}{-2\pi in} + \left\{\frac{du}{dt}(P_1 - P_0)\right\} \frac{e^{-2\pi in(upper\ limit)} - e^{-2\pi in(lower\ limit)}}{(-2\pi in)^2}$$

$$upper\ limit: \frac{d_b + d_c}{d_p}, lower\ limit: \frac{d_c}{d_p}, \frac{du}{dt} : \frac{d_p}{d_b}$$

$$= \frac{P_1 e^{-\frac{2\pi in(d_b+d_c)}{d_p}} - P_0 e^{-\frac{2\pi i n d_c}{d_p}}}{-2\pi in} + \left\{\frac{d_p}{d_b}(P_1 - P_0)\right\} \frac{e^{-\frac{2\pi in(d_b+d_c)}{d_p}} - e^{-\frac{2\pi i n d_c}{d_p}}}{(-2\pi in)^2}$$

When $n = 0$

$$du = Ndt$$
$$dt = \frac{du}{N}$$
$$t = \frac{u + x}{N}$$

$$\int_{\frac{x}{N}}^{\frac{x+1}{N}} (P_1 - P_0)u + P_0 dt$$

$$= \frac{1}{N}\int_0^1 (P_1 - P_0)u + P_0 du$$

$$= \frac{1}{N}\left| \frac{(P_1 - P_0)u^2}{2} + P_0 u \right|_0^1$$

$$= \frac{1}{N}\left\{\frac{(P_1 - P_0)}{2} + P_0\right\}$$

$$\frac{du}{dt} = N$$

$$= \frac{dt}{du}\left\{\frac{(P_1 - P_0)}{2} + P_0\right\}$$

Similarly, when *by_dist* is True,

$$\int_{\frac{d_c}{d_p}}^{\frac{d_c+d_b}{d_p}} (P_1 - P_0)u + P_0 dt$$

$$= \frac{dt}{du}\left\{\frac{(P_1 - P_0)}{2} + P_0\right\}$$

$$t = \frac{ud_b + d_c}{d_p}$$

$$du = \frac{d_p}{d_b} dt$$

$$\frac{dt}{du} = \frac{d_b}{d_p}$$

$$= \frac{d_b}{d_p} \left\{ \frac{(P_1 - P_0)}{2} + P_0 \right\}$$

Cubic Bezier curve:

$$B(u) = (P_3 - 3P_2 + 3P_1 - P_0)u^3 + (3P_2 - 6P_1 + 3P_0)u^2 + (3P_1 - 3P_0)u + P_0$$
$$Let\ A = P_3 - 3P_2 + 3P_1 - P_0, B = 3P_2 - 6P_1 + 3P_0, C = 3P_1 - 3P_0, D = P_0$$

When $n \neq 0$

$$\int_{\frac{x}{N}}^{\frac{x+1}{N}} e^{-2\pi int}(Au^3 + Bu^2 + Cu + D)dt$$

$$u = Nt - x$$
$$t = \frac{u + x}{N}$$
$$du = Ndt$$
$$dt = \frac{du}{N}$$

Substituting t with u,

$$= \frac{1}{N} \int_0^1 e^{-\frac{2\pi in(u+x)}{N}} \left(Au^3 + Bu^2 + Cu + D\right) du$$

Integrating by parts, using the quick technique

sign	*Integrals*	*Differentials*

Kohki Hatori

+	$\frac{Ne^{-\frac{2\pi in(u+x)}{N}}}{-2\pi in}$	Au^3+Bu^2+Cu+D
-	$\frac{N^2e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^2}$	$3Au^2+2Bu+C$
+	$\frac{N^3\,e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^3}$	$6Au+2B$
-	$\frac{N^4e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^4}$	$6A$

$$= \frac{1}{N}\left[\frac{Ne^{-\frac{2\pi in(u+x)}{N}}}{-2\pi in}(Au^3+Bu^2+Cu+D) - \frac{N^2e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^2}(3Au^2+2Bu+C) + \frac{N^3\,e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^3}(6Au+2B) - \frac{N^4e^{-\frac{2\pi in(u+x)}{N}}}{(-2\pi in)^4}6A\right]_0^1$$

$$= \frac{e^{-\frac{2\pi in(x+1)}{N}}}{-2\pi in}(A+B+C+D) - \frac{Ne^{-\frac{2\pi in(x+1)}{N}}}{(-2\pi in)^2}(3A+2B+C) + \frac{N^2\,e^{-\frac{2\pi in(x+1)}{N}}}{(-2\pi in)^3}(6A+2B) - \frac{N^3e^{-\frac{2\pi in(x+1)}{N}}}{(-2\pi in)^4}6A - \{\frac{e^{-\frac{2\pi inx}{N}}}{-2\pi in}D - \frac{Ne^{-\frac{2\pi inx}{N}}}{(-2\pi in)^2}C + \frac{N^2\,e^{-\frac{2\pi inx}{N}}}{(-2\pi in)^3}2B - \frac{N^3e^{-\frac{2\pi inx}{N}}}{(-2\pi in)^4}6A\}$$

$$= \frac{e^{-\frac{2\pi in(x+1)}{N}}(A+B+C+D) - De^{-\frac{2\pi inx}{N}}}{-2\pi in} - \frac{N(e^{-\frac{2\pi in(x+1)}{N}}(3A+2B+C) + Ce^{-\frac{2\pi inx}{N}})}{(-2\pi in)^2} + \frac{N^2\,(e^{-\frac{2\pi in(x+1)}{N}}(6A+2B) - 2Be^{-\frac{2\pi inx}{N}})}{(-2\pi in)^3} - \frac{N^3(e^{-\frac{2\pi in(x+1)}{N}}6Ae^{-\frac{2\pi inx}{N}})}{(-2\pi in)^4}$$

$$= \frac{e^{-2\pi in(upper\ limit)}(A+B+C+D) - De^{-2\pi in(lower\ limit)}}{-2\pi in} - \frac{\frac{du}{dt}(e^{-2\pi in(upper\ limit)}(3A+2B+C) + Ce^{-2\pi in(lower\ limit)})}{(-2\pi in)^2} + \frac{\left(\frac{du}{dt}\right)^2(e^{-2\pi in(upper\ limit)}(6A+2B) - 2Be^{-2\pi in(lower\ limit)})}{(-2\pi in)^3} - \frac{\left(\frac{du}{dt}\right)^3(e^{-2\pi in(upper\ limit)}6Ae^{-2\pi in(lower\ limit)})}{(-2\pi in)^4}$$

Similarly, when *by_dist* is True, using the relationship above,

$$u = \frac{td_p - d_c}{d_b}$$

$$t = \frac{ud_b + d_c}{d_p}$$

$$du = \frac{d_p}{d_b}dt$$

$$= \frac{e^{-\frac{2\pi in(d_c+d_b)}{d_p}}(A+B+C+D) - De^{-\frac{2\pi ind_c}{d_p}}}{-2\pi in} - \frac{\frac{d_p}{d_b}(e^{-\frac{2\pi in(d_c+d_b)}{d_p}}(3A+2B+C) + Ce^{-\frac{2\pi ind_c}{d_p}})}{(-2\pi in)^2} + \frac{\left(\frac{d_p}{d_b}\right)^2(e^{-\frac{2\pi in(d_c+d_b)}{d_p}}(6A+2B) - 2Be^{-\frac{2\pi ind_c}{d_p}})}{(-2\pi in)^3} - \frac{\left(\frac{d_p}{d_b}\right)^3(e^{-\frac{2\pi in(d_c+d_b)}{d_p}}6Ae^{-\frac{2\pi ind_c}{d_p}})}{(-2\pi in)^4}$$

When $n = 0$

$$\int_{\frac{x}{N}}^{\frac{x+1}{N}} (Au^3 + Bu^2 + Cu + D)dt$$

$$du = Ndt$$

$$dt = \frac{du}{N}$$

$$t = \frac{u + x}{N}$$

$$= \frac{1}{N}\int_0^1 (Au^3 + Bu^2 + Cu + D)du$$

$$= \frac{1}{N}\left[\frac{A}{4}u^4 + \frac{B}{3}u^3 + \frac{C}{2}u^2 + Du\right]_0^1$$

$$= \frac{1}{N}\left(\frac{A}{4} + \frac{B}{3} + \frac{C}{2} + D\right)$$

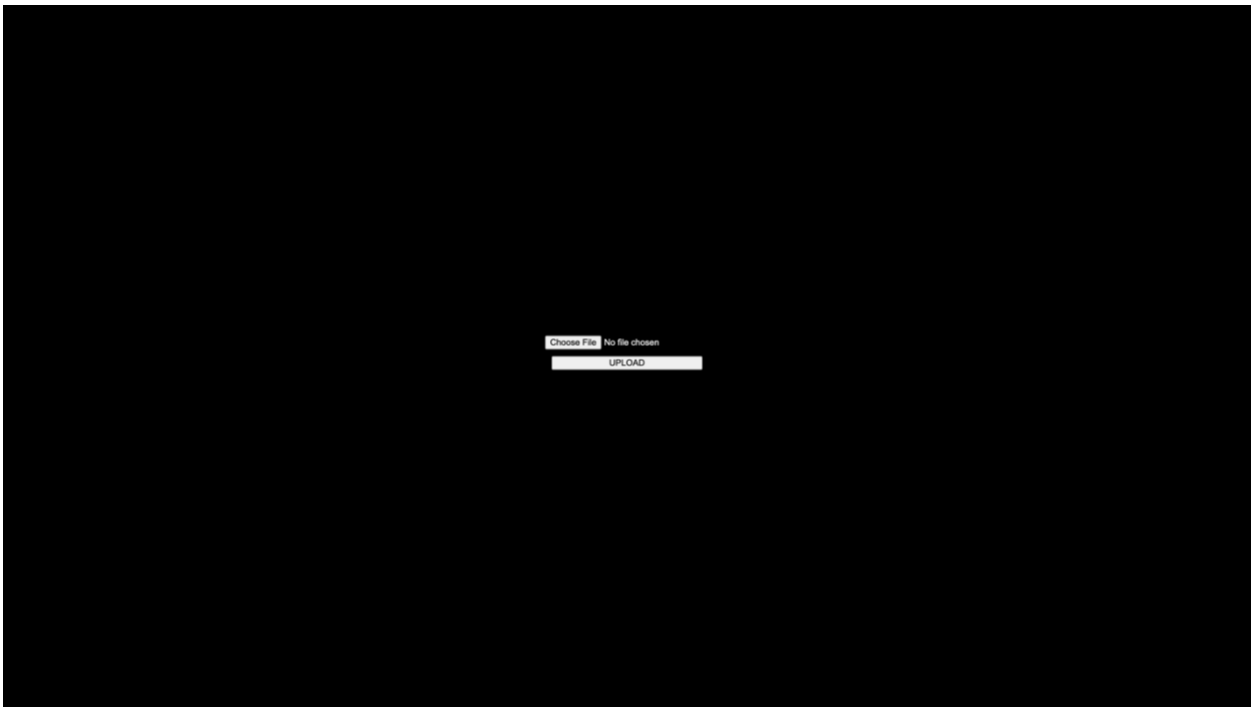
$$= \frac{dt}{du}\left(\frac{A}{4} + \frac{B}{3} + \frac{C}{2} + D\right)$$

Similarly, when *by_dist* is True,

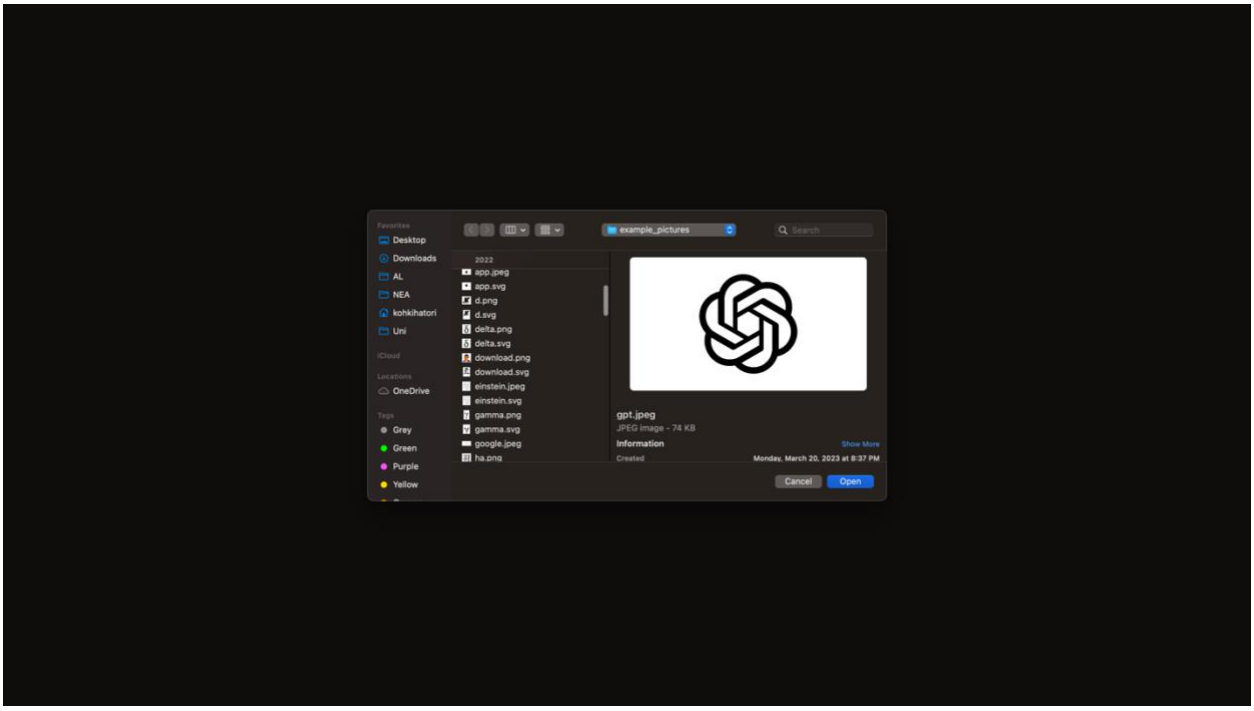
$$= \frac{dt}{du}\left(\frac{A}{4} + \frac{B}{3} + \frac{C}{2} + D\right)$$

$$= \frac{d_b}{d_p}\left(\frac{A}{4} + \frac{B}{3} + \frac{C}{2} + D\right)$$

User Interface



The initial page has a simple structure with just two buttons, “Choose File” and “UPLOAD”



Users can choose any image file on their local machine easily.

The animation can be begun immediately after the data is transmitted from the server to the client computer.

On the top left, there is the control panel. From left to right, there is:

1. QUIT button
   This gets user back to the initial page
2. CLEAR button
   This erases the entire animation.
3. START button
   This starts off the animation. Once it's started, it changes to PAUSE button. The animation will pause when it's pressed.
4. Circle toggle
   This shows and hides the circles around each vector.
5. Vector toggle
   This shows and hides the vectors
6. Number of vectors slider
   This changes the number of vectors used to draw the path(s)
7. Speed slider
   This changes the speed at which the vectors rotate.

JSON structure

Below is the structure of the JSON object that is passed on to the frontend from the backend upon a POST request from user's web browser.

{
"lim": {"x": xlim, "y": ylim},

Kohki Hatori

“sets_of_coeffs”: sets_of_coeffs
}

Testing

#	input	Test type	Expected Outcome	Actual Outcome	Result
1	.jpeg image	Normal	Animation	Animation	Pass
2	.png image	Normal	Animation	Animation	Pass
3	.svg image	Normal	Animation	Animation	Pass
4	.jpeg image with numerous edges	Erroneous	Very slow animation with not very clear edges.	Very slow animation with not very clear edges.	Pass
5	.pdf file	Erroneous	Error message	Error message	Pass

Evaluation

FR 1 Users should be able to easily upload an image file from their devices.
➔ This is done using HTML and JavaScript.

FR 2 Users should be able to control the parameters such as number of vectors and speed of rotation
➔ This is done using HTML and JavaScript.
FR 3 The program must be able to display the drawing on a web browser.
➔ This is done using HTML canvas and JavaScript.

NFR 1 The program should be able to display circles around the vectors and vectors themselves for clearer display.
➔ This is done using HTML canvas and JavaScript.
NFR 2 Users should be able to stop and resume the animation at any point.
➔ This is done using HTML and JavaScript.
FR 4 The API server must be able to receive a request from and respond to the frontend.
➔ This is done using FastAPI and Python.
FR 5 The API server must be able to run the model with given parameters.
➔ This is done using FastAPI and Python.
FR 6 The program must be able to calculate the coefficients for each vector.
➔ This is done using Coefficient_calculator in coeff.py.
FR 7 The program must be able to detect edges in images.
➔ This is done using Potrace library.
FR 8 The program must be able to convert a bitmapped image to an SVG image file format.
➔ This is done using Potrace library.
FR 9 The program must be able to get Bezier curve equations from SVG files.
➔ This is done using SVG class in svg.py
FR 10 The program must be able to obtain a parametric equation from the Bezier curves in SVG files.

➔ This is done using Bezier class in bezier.py,

FR 11 The program must be able to combine multiple edges to a single edge.

➔ This is not implemented, as the animation can display multiple edges using multiple sets of coefficients.

FR 12 The data server must be able to receive a request from the API server and return data specified by the API server.

➔ This is not implemented.

NFR 3 The data (coefficients, original image, and selected edges) of each drawing is stored in a text-based database.

➔ This is not implemented

Most of my objectives are met at decent performance level and User Interface.
My project can be improved by following:

1. Building a database of images that have been posted previously to the server and making them available to users.
2. Combining multiple edges to one edge for better visibility and simplicity.

Lastly, I was not able to host the API server on the internet so it's not accessible yet to everyone, and I hope to do so in the near future.

Bibliography

- SVG path documentation 1:
https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths
- SVG path documentation 2:
https://svgwg.org/specs/paths/#PathDataCubicBezierCommands
- Potrace documentation:
https://potrace.sourceforge.net/#version
- Imagemagick documentation:
http://www.imagemagick.org/script/download.php
- Bézier curve
https://en.wikipedia.org/wiki/B%C3%A9zier_curve
- De Casteljau's algorithm
https://en.wikipedia.org/wiki/De_Casteljau%27s_algorithm

Demonstration:

https://youtu.be/SP-9ka76OwI