MATHEMATICS ANALYSIS AND APPROACHES HL
Producing the IB Logo with the Fourier Series

Candidate Code:
Session: May 2024
Page Count:

Contents

1 Rationale

I have shown interest in visual arts done through the means of software, with particular experience in 3D modelling and animation in Blender and Cinema 4D.

I never was experienced with drawing, therefore producing digital art on a 2D plane using artistic skill was not of interest to me. However, something that I came across online was the use of the Fourier Series in order to produce vector art, which instantly intrigued me.

While vector art files such as those with the file extension ".svg" relate to mathematics in the sense that it contains multiple graphed mathematical relationships in order to produce an image, the method of using the Fourier Series to produce similar art is more mathematically intriguing, as it proves use just one expression to produce the same result done by the numerous mathematical relationships.

2 Aim

The objective of this investigation is to link Fourier series with complex numbers to create a single series that is capable of reproducing the IB logo on the Argand plane.

3 Plan of Action

This exploration focuses on the following areas of math:

- Integral Calculus
- Series
- Trigonometry
- Complex Analysis
- Vectors

4 Background Information

4.1 Overarching idea of the Fourier Series

A periodic function is one where the output for a particular input equals to the output for the sum of the same input and the value of the function's period. This can be represented mathematically as:

$$f(x) = f(x + P)$$
$$\text{where } P = \text{ the period of the function}$$

The sine wave is widely known for being a periodic function for the ease of graphing a sinusoidal wave. However, there are periodic functions that are difficult to graph with an algebraic expression, such as one that alternates between 1 and -1 or one that is shaped as a zig-zag.

This is the motivation behind the Fourier Series, which is to be able to represent period functions that normally can't be represented by an algebraic function.

The idea behind the Fourier Series is to take an infinite sum of varying sinusoidal functions such that a desired periodic function is produced.

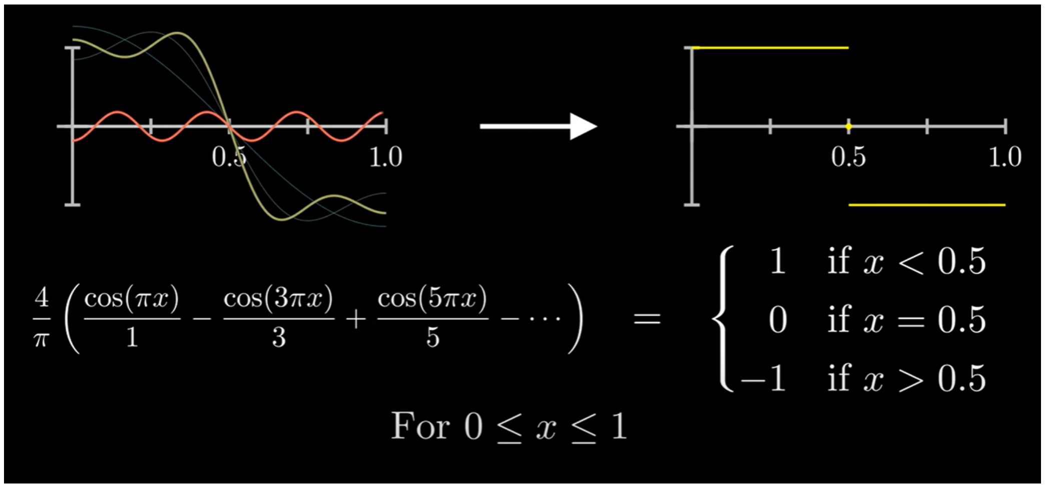


Figure 1: Visualization of the mechanism of the Fourier Series (Sanderson, 2019). The yellow line is the periodic function resulting from the previous iteration, the red line is the sinusoidal function to be added in the next iteration.

4.2 Idea of drawing with the Fourier series explained with the Cartesian Plane

The rule is for eligible drawings to be any that can be drawn by starting at one point on a cartesian plane and, without lifting the hypothetical plane throughout the entire sketch, return to the exact same point.

Defining the variable t as time, $t = 0$ will represent the point in time where the drawing began and $t = 1$ will represent the point in time where the drawing ended.

Each point of the drawing on the plane will be defined by $P(x(t), y(t))$, where $x(t)$ and $y(t)$ are both functions with an input of t that indicates the coordinates after some amount of time passed of the pen's progress through the drawing.

Assuming that any function can be represented as a Fourier Series, then $x(t)$ and $y(t)$ can be any function and therefore, by extracting the coordinates on a graph of any drawing obeying the rule described earlier, the Fourier Series of $x(t)$ and $y(t)$ can be determined and therefore produce the desired drawing for $t \in [0, 1]$

As a simple example that does not require a Fourier Series, we can take a unit circle defined by $x^2 + y^2 = 1$ as the drawing. It quickly becomes evident of what $x(t)$ and $y(t)$ are, as since it is a circle, then $x(t) = \cos(2\pi t)$ and $y(t) = \sin(2\pi t)$.

4.3 Enriched application to draw on the Argand Plane

Euler's formula is defined to be

$$e^{it} = \cos(t) + i\sin(t) \tag{1}$$

Given that both the cosine function and the sine function are included in this formula, a connection between this formula and the Fourier Series becomes evident. The two sinusoidal functions in the formula are the core behind applying the Fourier Series to the Argand Plane.

It is easier to think of Euler's formula to be a vector on the Argand Plane (Sanderson, 2019). With just the formula given above, we have a vector with a length of 1 that rotates counterclockwise when the value that e is raised to is positive and clockwise when the value is negative.

Let's incorporate n and 2π to the power in Equation (**1**) such that we have $e^{n \cdot 2\pi it}$. The purpose of 2π is to simplify what defines a revolution, as now, for every unit of time that t passes, a revolution will be completed. $n, n \in \mathbb{Z}$ will define the frequency and directionality of the rotation of the vector. If $n > 0$, then the vector will rotate counterclockwise and vice versa. If $n = 2$, then the vector would rotate by 2 revolutions for every unit of time that passes.

Lastly, let's multiply the entire power by the variable c_n to get $c_n e^{n \cdot 2\pi it}$. This will not only allow for the vector to be scaled but also for the starting rotation of the vector to be defined. Let's temporarily define c_n to be:

$$c_n = A \cdot z$$

A will indicate the factor to which the vector will be scaled by. Because the original length of the vector was 1, then the factor will directly indicate the length of the vector.

z will indicate the starting rotation of the vector, which would be $e^{i\theta}$, where θ is the starting rotation angle in radians.

It could be imagined that the final Fourier Series that produces a desired drawing is the sum of multiple vectors of different magnitudes rotating a different frequencies indicated by $n, n \in \mathbb{Z}$. Therefore, if $f(t)$ is defined to be the function that represents the drawing, this can be expressed mathematically as:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{n \cdot 2\pi it} \tag{2}$$

The main concern as of right now is how the value of c_n will be determined for a specific drawing.

The easiest way to start off is by first considering $c_0 e^{0 \cdot 2\pi it}$. This can be simplified into c_0, but what does this mean? This is the vector that is rotating at a frequency of 0, which means that it is static. It can therefore be defined to be the "centre of mass"(Sanderson, 2019) of the entire function.

If we take discrete intervals of t and obtain the value of $f(t)$, then by taking the average of all those values, we obtain a complex number close to c_0. This is illusrated by Figure 2.

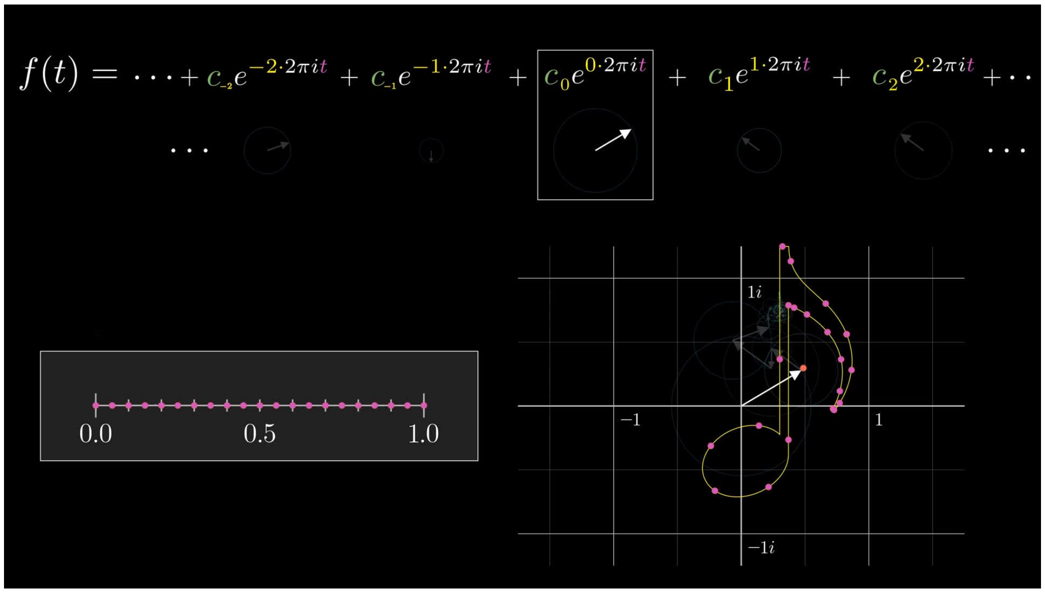

$f(t) = \cdots + c_{-2}e^{-2\cdot 2\pi i t} + c_{-1}e^{-1\cdot 2\pi i t} + c_0 e^{0\cdot 2\pi i t} + c_1 e^{1\cdot 2\pi i t} + c_2 e^{2\cdot 2\pi i t} + \cdots$
0.0
0.5
1.0
1i
−1
1
−1i

Figure 2: Averaging points throughout $f(t)$ illustrated (Sanderson, 2019). The number line represent t, and the red dots on the musical note represent the resulting values of $f(t)$

With finer and finer intervals of t, the result becomes more and more accurate to the value of c_0, therefore it can be said that:

$$c_0 = \lim_{\Delta t \to 0} \sum_{t=0}^{\frac{1}{\Delta t}} f(t \cdot \Delta t)\Delta t \tag{3}$$

From Equation **(3)**, c_0 can ultimately be expressed as:

$$c_0 = \int_0^1 f(t)\, dt \tag{4}$$

We can use the same technique for all other values of n, but the problem is for all other values of n that are not 0, the vector is rotating, therefore it doesn't make sense to take the average of the rotating vector.

Recalling the thorough definition of $f(t)$ stated earlier in Equation 2, we can substitute Equation 2 into Equation 4.

$$c_0 = \int_0^1 \sum_{n=-\infty}^{\infty} c_n e^{n\cdot 2\pi it}\, dt$$

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

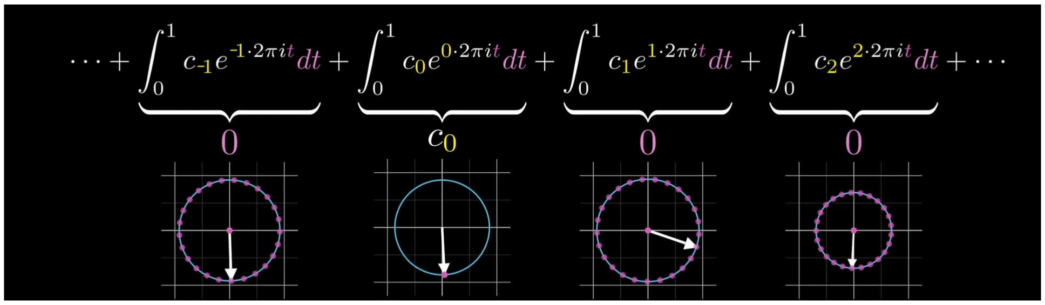
$$c_0 = \cdots + \int_0^1 c_{-1}e^{-1\cdot 2\pi it}\, dt + \int_0^1 c_0 e^{0\cdot 2\pi it}\, dt + \int_0^1 c_1 e^{1\cdot 2\pi it}\, dt + \cdots$$

Remember that $\int_0^1 c_0 e^{0\cdot 2\pi it}\, dt$ was easy to simplify as the power cancels out from being raised to 0. In turn, it can be further evaluated to be just c_0, as shown below:

$$\begin{aligned} &\int_0^1 c_0 e^{0\cdot 2\pi it}\, dt \\ &= \int_0^1 c_0\, dt \\ &= [c_0 t]_0^1 \\ &= c_0 \end{aligned}$$

Additionally, if all the other integrals were thought of as the the average of all the points produced when its vector rotates by one revolution, then it can be argued that each integral, when evaluated, would be 0.

This idea is illustrated in Figure 3.



$\cdots + \underbrace{\int_0^1 c_{-1}e^{-1\cdot 2\pi it}dt}_{0} + \underbrace{\int_0^1 c_0 e^{0\cdot 2\pi it}dt}_{c_0} + \underbrace{\int_0^1 c_1 e^{1\cdot 2\pi it}dt}_{0} + \underbrace{\int_0^1 c_2 e^{2\cdot 2\pi it}dt}_{0} + \cdots$

Figure 3: Illustration of averaging all the points on a circle (Sanderson, 2019).

If we were to multiply $f(t)$ by $e^{-n \cdot 2\pi it}$, then an effect can occur where all of the power's exponents will decrease by n. Ultimately, we will get a similar scenario in the summation of all the integrals where all integrals are evaluated to become 0 except for the integral where $n = 0$, which evaluates to c_0. However, the multiplication of the two powers will cause the integral of some n value to simplify to just c_n.

This mechanism is mathematically shown below.

$$
\begin{aligned}
&n = 1 \\
&c_1 = \int_0^1 f(t)e^{-1\cdot 2\pi it}\,dt \\
&c_1 = \cdots + \int_0^1 c_{-1}e^{-1\cdot 2\pi it} \cdot e^{-1\cdot 2\pi it}\,dt \\
&+ \int_0^1 c_0 e^{0\cdot 2\pi it} \cdot e^{-1\cdot 2\pi it}\,dt + \int_0^1 c_1 e^{1\cdot 2\pi it} \cdot e^{-1\cdot 2\pi it}\,dt + \cdots \\
&c_1 = \cdots + \int_0^1 c_{-1}e^{-2\cdot 2\pi it}\,dt + \int_0^1 c_0 e^{-1\cdot 2\pi it}\,dt + \int_0^1 c_1 e^{0\cdot 2\pi it}\,dt + \cdots \\
&c_1 = \cdots + 0 + 0 + c_1 + 0 + 0 + \cdots \\
&c_1 = c_1
\end{aligned}
$$

Therefore, c_n can be defined as:

$$c_n = \int_0^1 f(t)e^{-n\cdot 2\pi it}\,dt \tag{5}$$

Equation 5 is what will be used in order to determine each value of c_n

All concepts of this section come from (Sanderson, 2019).

4.4 Bezier Curves

4.4.1 Linear interpolation

Suppose there is a free moving point P on the line drawn between the stationary points P_0 and P_1 as illustrated on Figure 4.

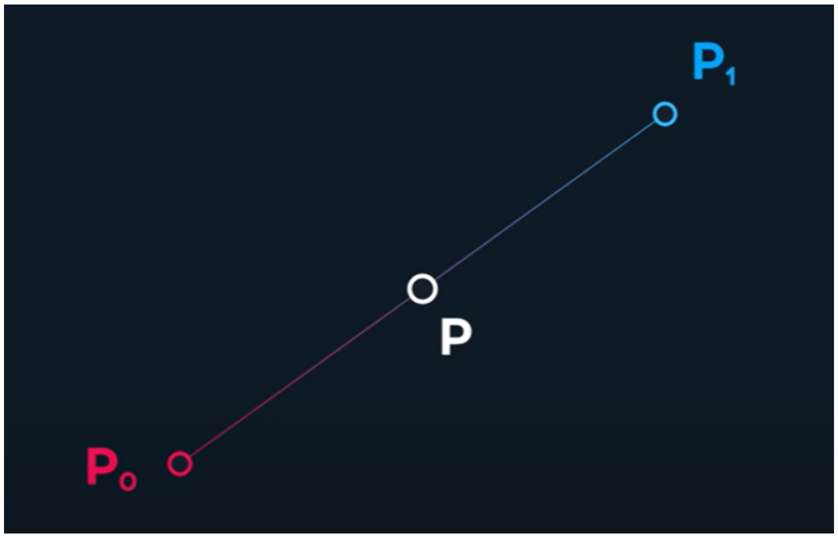


Figure 4: 2 point Bezier (Holmer, 2021).

The position of P along the line will be defined by t, which is thought of to be the percentage that t is along the line, where $t = 100\%$ is located at P_1 and $t = 0\%$ is located at P_0. This process is called linear interpolation (lerp) and can be expressed mathematically as:

$$P = lerp(P_0, P_1, t) = (1 - t)P_0 + tP_1 \tag{6}$$

where $lerp(P_0, P_1, t)$ is the function that represents the process of lerping.

4.4.2 Cubic Bezier Curves

While there are many types of Bezier Curves, this investigation will only focus on Cubic Bezier Curves as they are the only type of Bezier Curve used in the IB Logo.

Suppose that instead of just 2 points, there are 4 points on the plane (P_0, P_1, P_2, P_3). The lines are drawn so that P_0 connects to P_1, P_1 connects to P_2, and P_2 connects to P_3 such that P_0 and P_3 are both endpoints.

Each line segment has its own individual moving point, with all moving points "lerping" according to the same universal t value.

Lines can then be drawn between these new moving points, and these points lerp according to the same t value.

This process of adding points on lines and drawing new lines until just one moving point along a single line is created, which will be the ”pen” of the Bezier Curve.

This is all illustrated by Figures 5 and 6.

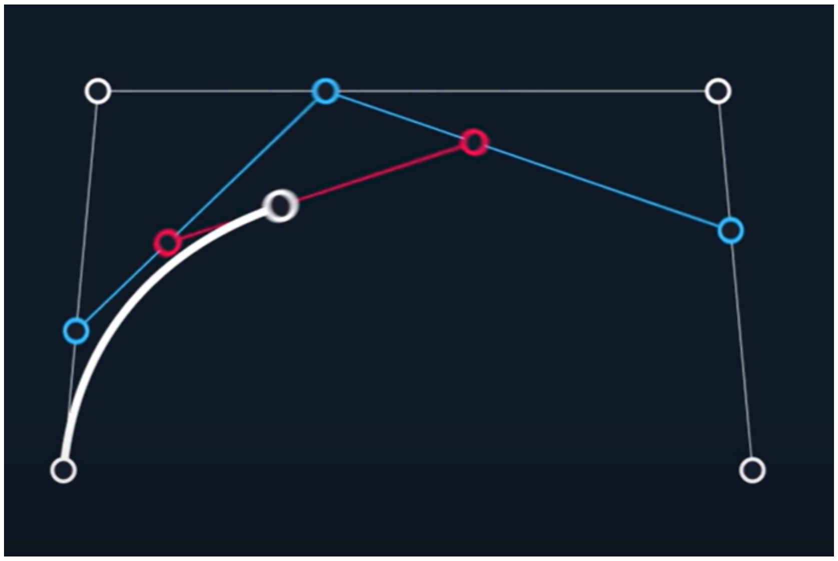


Figure 5: Cubic Bezier around the middle of its lerp (Holmer, 2021).

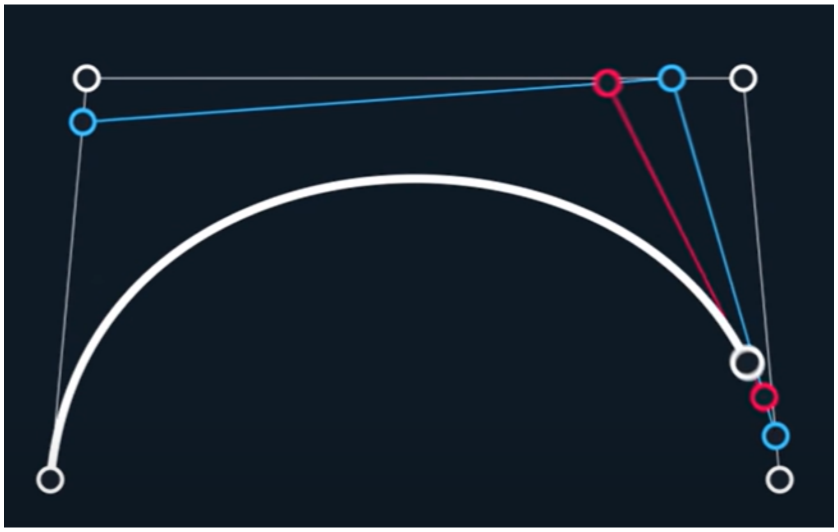


Figure 6: Cubic Bezier nearing the end of its lerp (Holmer, 2021).

4.4.3 Bernstein Polynomial Form

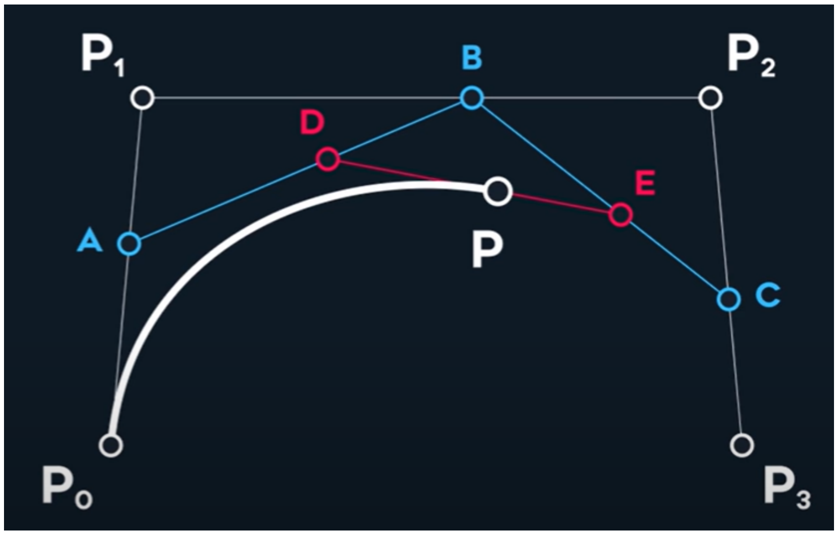


Figure 7: Cubic Bezier Curve with labelled points (Holmer, 2021).

We can derive a general formula for P in terms of P_0, P_1, P_2, P_3, t by expanding out all of the lerp functions for each point in a Cubic Bezier Curve and substituting when necessary, as shown below. Variables are with reference to Figure 7.

$$A = lerp(P_0, P_1, t)$$
$$B = lerp(P_1, P_2, t)$$
$$C = lerp(P_2, P_3, t)$$
$$D = lerp(A, B, t)$$
$$E = lerp(B, C, t)$$
$$P = lerp(D, E, t)$$

From Equation 6

$$lerp(P_0, P_1, t) = (1 - t)P_0 + tP_1$$

$$A = (1 - t)P_0 + tP_1$$

$$B = (1-t)P_1 + tP_2$$
$$C = (1-t)P_2 + tP_3$$
$$D = (1-t)A + tB$$
$$E = (1-t)B + tC$$
$$P = (1-t)D + tE$$

$$P = P_0(-t^3 + 3t^2 - 3t + 1) + P_1(3t^3 - 6t^2 + 3t)$$
$$+ P_2(-3t^3 + 3t^2) + P_3(t^3)$$

Therefore, the general formula for P in terms of $P_0, P_1, P_2, P_3, \text{and} t$ is

$$P = P_0(-t^3 + 3t^2 - 3t + 1) + P_1(3t^3 - 6t^2 + 3t) + P_2(-3t^3 + 3t^2) + P_3(t^3) \quad (7)$$

All information regarding Bezier Curves come from (Holmer, 2021).

4.5 Composition of ".svg" files

SVG files are "XML-based vector image format for defining two dimensional graphics"("SVG", 2023).

An SVG's two dimensional size is defined by the "viewBox" attribute, which will indicate the width and height of the canvas.

There are many attributes for drawing predefined shapes, but what we are concerned with is the "path" attribute, which allows for custom shapes and is what the IB logo will be composed of.

The "path" attribute will be defined by various curves that are indicated by letters. The letters that are of concern in this investigation are listed below.

- M: Move the pen to somewhere without drawing a line
- C: Cubic bezier defined by the starting location, 2 control points, and an indicated end location
- L: Draw a vertical line to some coordinate
- V: Draw a vertical line up to some y-coordinate
- Z: Draw a straight line back to the first point of the path

Note that when the letter is lowercase, the coordinates specified will be relative to the current position (ex. $m\ \Delta x\ \Delta y$). If the letter is uppercase, the coordinates specified will be absolute to the canvas (ex. $M\ x\ y$).

All information regarding paths come from (Mozilla Developer Network, 2023a).

5 Methodology

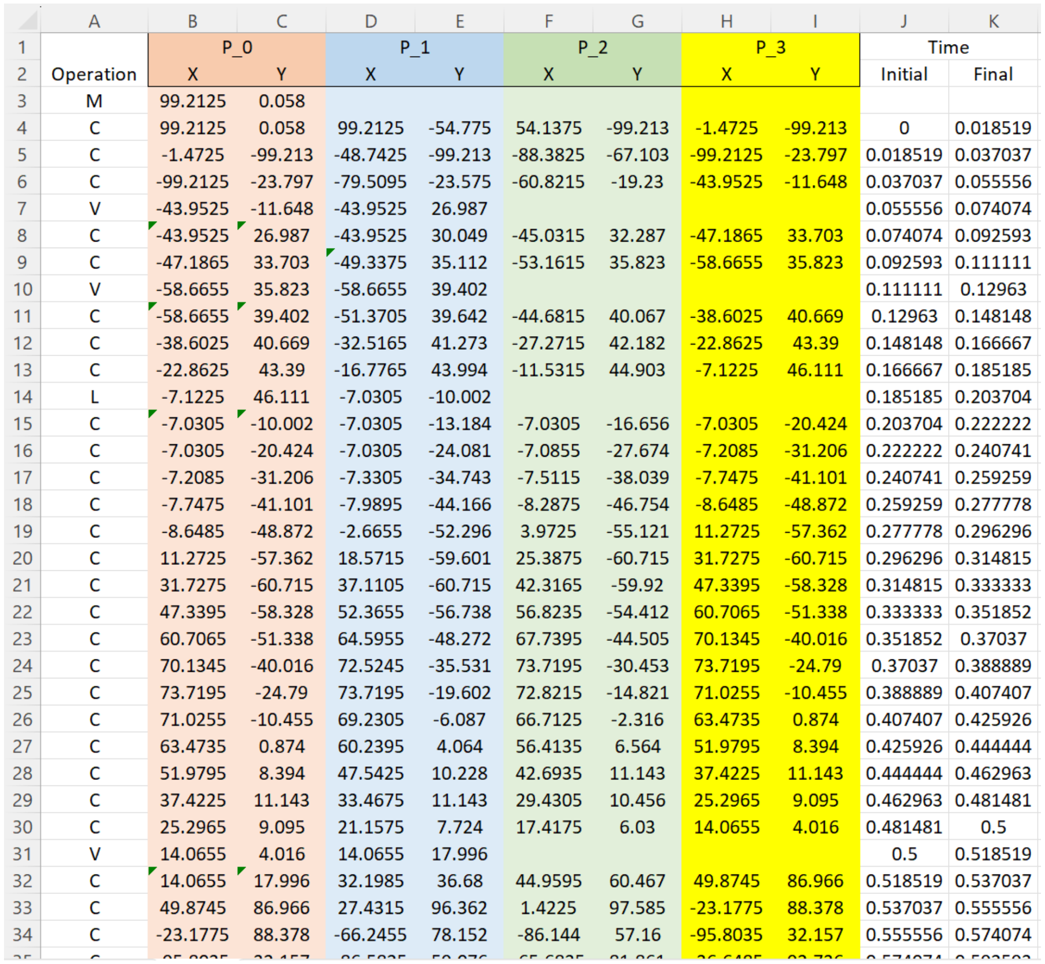
5.1 Analysis of the ".svg" File

On a canvas of height 198.426px and width 198.425px, a few lines of the path of the IB Logo is presented below, with the entire path presented in Section A of the appendix (International Baccalaureate Organisation, 2013). **Note: SVG files interpret positive y-values as downwards.**

```
<path fill="url(#SVGID_1_)" d="
M198.425,99.155
c 0,54.833 -45.075,99.271 -100.685,99.271
c -47.27, 0 -86.91 -32.11 -97.74 -75.416
c 19.703 -0.222, 38.391 -4.567, 55.26 -12.149
V 72.226
...
C 68.199, 48.705, 71.582, 47.359, 74.363, 44.669
z"/>
```

Disregarding any "Move" actions, there are 54 pen strokes in total for the IB Logo. This means that for the starting piecewise form of $f(t)$, each piece will have a domain of $\frac{1}{54}(n-1) \leq t < \frac{1}{54}n$, where n represents the index of the current pen stroke.

The contents of this path was moved into an Excel Spreadsheet for the sake of organization, with part of the spreadsheet shown in Figure 8.



	A	B	C	D	E	F	G	H	I	J	K
1		P_0		P_1		P_2		P_3		Time	
2	Operation	X	Y	X	Y	X	Y	X	Y	Initial	Final
3	M	99.2125	0.058								
4	C	99.2125	0.058	99.2125	-54.775	54.1375	-99.213	-1.4725	-99.213	0	0.018519
5	C	-1.4725	-99.213	-48.7425	-99.213	-88.3825	-67.103	-99.2125	-23.797	0.018519	0.037037
6	C	-99.2125	-23.797	-79.5095	-23.575	-60.8215	-19.23	-43.9525	-11.648	0.037037	0.055556
7	V	-43.9525	-11.648	-43.9525	26.987					0.055556	0.074074
8	C	-43.9525	26.987	-43.9525	30.049	-45.0315	32.287	-47.1865	33.703	0.074074	0.092593
9	C	-47.1865	33.703	-49.3375	35.112	-53.1615	35.823	-58.6655	35.823	0.092593	0.111111
10	V	-58.6655	35.823	-58.6655	39.402					0.111111	0.12963
11	C	-58.6655	39.402	-51.3705	39.642	-44.6815	40.067	-38.6025	40.669	0.12963	0.148148
12	C	-38.6025	40.669	-32.5165	41.273	-27.2715	42.182	-22.8625	43.39	0.148148	0.166667
13	C	-22.8625	43.39	-16.7765	43.994	-11.5315	44.903	-7.1225	46.111	0.166667	0.185185
14	L	-7.1225	46.111	-7.0305	-10.002					0.185185	0.203704
15	C	-7.0305	-10.002	-7.0305	-13.184	-7.0305	-16.656	-7.0305	-20.424	0.203704	0.222222
16	C	-7.0305	-20.424	-7.0305	-24.081	-7.0855	-27.674	-7.2085	-31.206	0.222222	0.240741
17	C	-7.2085	-31.206	-7.3305	-34.743	-7.5115	-38.039	-7.7475	-41.101	0.240741	0.259259
18	C	-7.7475	-41.101	-7.9895	-44.166	-8.2875	-46.754	-8.6485	-48.872	0.259259	0.277778
19	C	-8.6485	-48.872	-2.6655	-52.296	3.9725	-55.121	11.2725	-57.362	0.277778	0.296296
20	C	11.2725	-57.362	18.5715	-59.601	25.3875	-60.715	31.7275	-60.715	0.296296	0.314815
21	C	31.7275	-60.715	37.1105	-60.715	42.3165	-59.92	47.3395	-58.328	0.314815	0.333333
22	C	47.3395	-58.328	52.3655	-56.738	56.8235	-54.412	60.7065	-51.338	0.333333	0.351852
23	C	60.7065	-51.338	64.5955	-48.272	67.7395	-44.505	70.1345	-40.016	0.351852	0.37037
24	C	70.1345	-40.016	72.5245	-35.531	73.7195	-30.453	73.7195	-24.79	0.37037	0.388889
25	C	73.7195	-24.79	73.7195	-19.602	72.8215	-14.821	71.0255	-10.455	0.388889	0.407407
26	C	71.0255	-10.455	69.2305	-6.087	66.7125	-2.316	63.4735	0.874	0.407407	0.425926
27	C	63.4735	0.874	60.2395	4.064	56.4135	6.564	51.9795	8.394	0.425926	0.444444
28	C	51.9795	8.394	47.5425	10.228	42.6935	11.143	37.4225	11.143	0.444444	0.462963
29	C	37.4225	11.143	33.4675	11.143	29.4305	10.456	25.2965	9.095	0.462963	0.481481
30	C	25.2965	9.095	21.1575	7.724	17.4175	6.03	14.0655	4.016	0.481481	0.5
31	V	14.0655	4.016	14.0655	17.996					0.5	0.518519
32	C	14.0655	17.996	32.1985	36.68	44.9595	60.467	49.8745	86.966	0.518519	0.537037
33	C	49.8745	86.966	27.4315	96.362	1.4225	97.585	-23.1775	88.378	0.537037	0.555556
34	C	-23.1775	88.378	-66.2455	78.152	-86.144	57.16	-95.8035	32.157	0.555556	0.574074
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Figure 8: Path data imported in an Excel Spreadsheet

Using Javascript under the NodeJS Runtime, this spreadsheet can be read using the NPM Package "read-excel-file" (Kuchumov, n.d.). This package will parse the spreadsheet into an array of rows, with each row being an array of cells. Then, the code presented below will iterate through each row, with access to all the cells in that row indexed by zero-based numbering.

```
readXlsxFile('./SVG_Coordinate_Spreadsheet.xlsx')
.then((rows) => {
    // ‘rows‘ is an array of rows
    // each row being an array of cells.
    rows.forEach(row => {
        // iterate through each row, accessing ↩
           the
        // cells of each row
    });
})
```

For each row, the code checks if the operation for the current row is relevant for drawing. Only "Move" (M) is ignored, and all "Z" operations were written down as "line" (L) operations. Using the data within each row, the parameter of c_n can be computed for each value of n.

5.2 Computation of parameters

Given that $f(t)$ is able to be determined as a piecewise function from the previous section, the desired summation representing $f(t)$ can be determined.

Referring back to Section 4.3, c_n is defined as:

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

This means that every c_n must be determined individually. Therefore, it becomes evident that it is unreasonable to evaluate the infinite sum of $f(t) = \sum_{n=-\infty}^{\infty} c_n e^{n\cdot 2\pi it}$, and that the limits of the summation must be defined.

Let's rewrite $f(t)$ as:

$$f(t) = \sum_{n=-k}^{k} c_n e^{n\cdot 2\pi it}$$

so that k indicates the selected frequency of the two fastest vectors that are spinning in opposite directions to each other.

As $k \to \infty$, $f(t)$ becomes more and more accurate to the original drawing, which will be demonstrated by performing distinct analyses for various values of k.

5.2.1 Approach to integration

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

The integral above can be evaluated for some value of n by taking some small value to represent Δt, in which by summing up the values from $f(t)e^{-n\cdot 2\pi it}$ produced by each increment of t by Δt, a value close to the original integral can be determined (Sanderson, 2019). This is known as a Riemann Sum ("Riemann Sum", 2023).

$$c_n = \sum_{t=0}^{\frac{1}{\Delta t}} f(t\cdot\Delta t)e^{-n\cdot 2\pi i(t\cdot\Delta t)}\Delta t \tag{8}$$

Because a small value of Δt must be used, c_n must be evaluated through a computer. Additionally, since the original version of $f(t)$ can be thought of as a piecewise function composed of Bezier curves and lines, then Equation 8 must be split up into multiple sums, each accounting for the domain of each piece of the piecewise function ($\frac{1}{54}(n-1) \leq t < \frac{1}{54}n$).

Therefore, the approach of the code will be to iterate through each value of n from $-k \leq n \leq k$, and evaluate the Riemann Sum

5.2.2 Calculation for Cubic Bezier Curves

5.3 Rendering the final image

6 Results

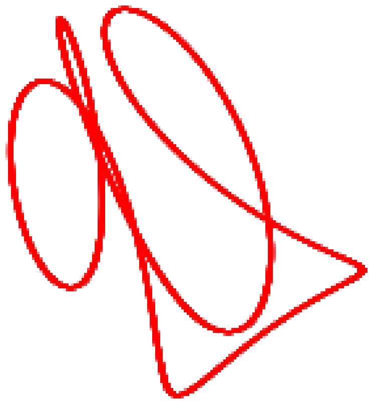


Figure 9: $\Delta t = 0.0001$, 10 spinning vectors, 11 total vectors

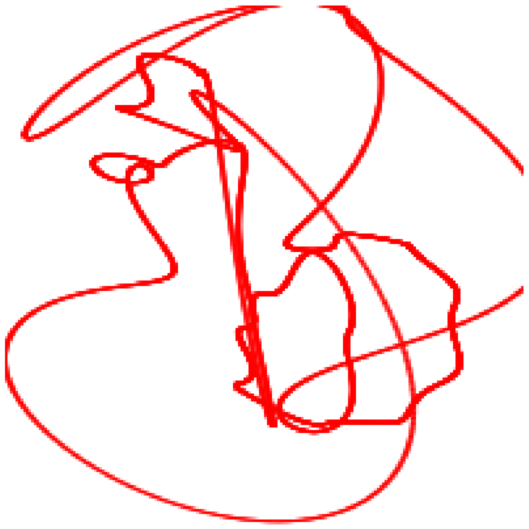


Figure 10: $\Delta t = 0.0001$, 50 spinning vectors, 51 total vectors

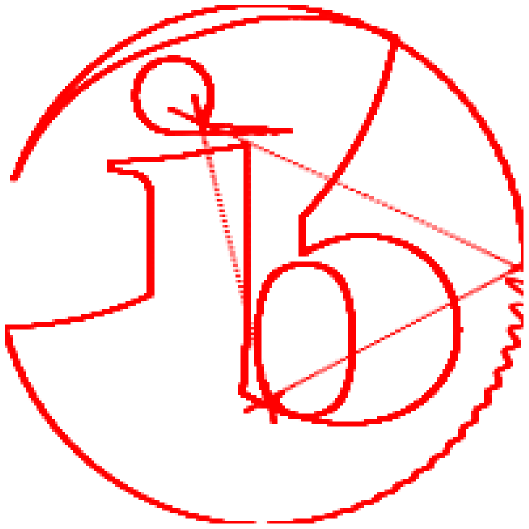


Figure 11: $\Delta t = 0.00001$, 2000 spinning vectors, 2001 total vectors

References

Automattic. (n.d.). *Canvas.*

Bazett, T. (n.d.). *Intro to FOURIER SERIES: The Big Idea - YouTube.* https://www.youtube.com/watch?v=wmCIrpLBFds.

Clastify. (n.d.). *IB Math AA IA examples — Clastify.* https://www.clastify.com/ia/math-aa.

DeCross, M., & Khim, J. (n.d.). *Fourier Series — Brilliant Math & Science Wiki.* https://brilliant.org/wiki/fourier-series/.

Euler's formula. (2023, December). *Wikipedia.*

Fireship. (2021, March). *SVG Explained in 100 Seconds.*

Fragomeni, P. (n.d.). *Prompt-sync.*

Holmer, F. (2021, August). *The Beauty of Bézier Curves.*

International Baccalaureate Organisation. (2013, November). *International Baccalaureate Logo.*

Jos de Jong. (n.d.). *Mathjs.*

Kuchumov, N. (n.d.). *Read-excel-file.*

Mozilla Developer Network. (2023a, November). *Paths - SVG: Scalable Vector Graphics — MDN.* https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths.

Mozilla Developer Network. (2023b, March). *Positions - SVG: Scalable Vector Graphics — MDN.* https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Positions.

Riemann sum. (2023, December). *Wikipedia.*

Sanderson, G. (2019, June). *But what is a Fourier series? From heat flow to drawing with circles.*

Sandlin, D. (2018, December). *What is a Fourier Series? (Explained by drawing circles) - Smarter Every Day 205.*

SVG. (2023, December). *Wikipedia.*

Tripathi, S. (2023, January). *IB Math IA (Ultimate Guide For 2023) - Nail IB.* https://nailib.com/blog/ib-math-ia.

A Full SVG Path

```
<path fill="url(#SVGID_1_)" d="
M198.425,99.155
c 0,54.833 -45.075,99.271 -100.685,99.271
c -47.27, 0 -86.91 -32.11 -97.74 -75.416
c 19.703 -0.222, 38.391 -4.567, 55.26 -12.149
V 72.226
c 0 -3.062 -1.079 -5.3 -3.234 -6.716
c -2.151 -1.409 -5.975 -2.12 -11.479 -2.12
```

```
v -3.579
c 7.295 -0.24, 13.984 -0.665, 20.063 -1.267
c 6.086 -0.604, 11.331 -1.513, 15.74 -2.721
l 0.092, 56.113
c 0, 3.182, 0, 6.654, 0, 10.422
c 0, 3.657 -0.055, 7.25 -0.178, 10.782
c -0.122, 3.537 -0.303, 6.833 -0.539, 9.895
c -0.242, 3.065 -0.54, 5.653 -0.901, 7.771
c 5.983, 3.424, 12.621, 6.249, 19.921, 8.49
c 7.299, 2.239, 14.115, 3.353, 20.455, 3.353
c 5.383, 0, 10.589 -0.795, 15.612 -2.387
c 5.026 -1.59, 9.484 -3.916, 13.367 -6.99
c 3.889 -3.066, 7.033 -6.833, 9.428 -11.322
c 2.39 -4.485, 3.585 -9.563, 3.585 -15.226
c 0 -5.188 -0.898 -9.969 -2.694 -14.335
c -1.795 -4.368 -4.313 -8.139 -7.552 -11.329
c -3.234 -3.19 -7.06 -5.69 -11.494 -7.52
c -4.437 -1.834 -9.286 -2.749 -14.557 -2.749
c -3.955, 0 -7.993, 0.687 -12.126, 2.048
c -4.139, 1.371 -7.879, 3.065 -11.231, 5.079
V 81.217
c 18.133 -18.684, 30.894 -42.471, 35.809 -68.97
c -22.443 -9.396 -48.452 -10.619 -73.052 -1.412
C 32.967, 21.061, 12.981, 42.053, 3.409, 67.056
c 9.221 -26.919, 30.12 -49.704, 59.155 -60.579
c 24.245 -9.068, 49.573 -8.27, 71.696, 0.248
c 0.002 -0.021, 0.011 -0.045, 0.014 -0.063
C 171.81, 21.082, 198.425, 57.046, 198.425, 99.155
z
M 100.598, 149.993
c 2.746, 4.661, 7.408, 6.986, 13.987, 6.986
c 6.564, 0, 11.233 -2.325, 13.985 -6.986
c 2.749 -4.652, 4.123 -11.988, 4.123 -22.013
c 0 -10.014 -1.374 -17.353 -4.123 -22.008
c -2.752 -4.657 -7.421 -6.979 -13.985 -6.979
c -6.58, 0 -11.242, 2.322 -13.987, 6.979
c -2.75, 4.655 -4.124, 11.994 -4.124, 22.008
C 96.474, 138.005, 97.848, 145.341, 100.598, 149.993
```

```
z
M 74.363, 44.669
c 2.793 -2.686, 4.185 -6.045, 4.185 -10.084
c 0 -4.037 -1.392 -7.397 -4.185 -10.086
c -2.781 -2.692 -6.165 -4.037 -10.129 -4.037
c -3.968, 0 -7.351, 1.345 -10.137, 4.037
c -2.79, 2.688 -4.181, 6.049 -4.181, 10.086
c 0, 4.039, 1.391, 7.398, 4.181, 10.084
c 2.786, 2.69, 6.168, 4.036, 10.137, 4.036
C 68.199, 48.705, 71.582, 47.359, 74.363, 44.669
z"/>
```

B Full Calculation and Graphing Program Code

```
const math = require('mathjs')
const prompt = require("prompt-sync")({ sigint: true ↩
  });
const readXlsxFile = require('read-excel-file/node')

const { createCanvas, loadImage } = require('canvas')
const canvas = createCanvas(198.425, 198.426)
const ctx = canvas.getContext('2d')


const dt = parseFloat(prompt("Enter \Delta t: "));
const k = parseFloat(prompt("Enter abs(k): "));

const sectionCount = 54;

function subTimeCalc(t){
    return sectionCount * t;
};

const fCubicCalc = (t, r) => {
    const P0 = math.complex(r[1], r[2]);
    const P1 = math.complex(r[3], r[4]);
    const P2 = math.complex(r[5], r[6]);
    const P3 = math.complex(r[7], r[8]);

    let subTime = subTimeCalc(t);
    const ret =
        math.add(
            math.add(
                math.multiply(P0, (-1*math.pow(↩
                   subTime,3) + 3*math.pow(subTime,2)↩
                    - 3*subTime + 1)),
                math.multiply(P1, (3*math.pow(subTime↩
                   ,3) - 6*math.pow(subTime,2) + 3*↩
                   subTime))
            ),
            math.add(
                math.multiply(P2, (-3*math.pow(↩
                   subTime,3) + 3*math.pow(subTime,2)↩
                   )),
                math.multiply(P3, (math.pow(subTime↩
                   ,3)))
            )
        )
    return ret;
};

const CnDict = {};
for(let n = -k; n <= k; n++){
    CnDict[n] = math.complex(0, 0);
}

function plotPoint(x, y){
    const adjX = x + 99.2125;
    const adjY = -y + 99.213;
    ctx.strokeStyle = 'rgba(255,0,0,1)';
    ctx.beginPath();
    ctx.lineTo(adjX-1, adjY);
    ctx.lineTo(adjX+1, adjY);
    ctx.stroke();
}
```

```
// console.log('<img src="' + canvas.toDataURL() + '"↩
   />');

readXlsxFile('./SVG_Coordinate_Spreadsheet.xlsx').↩
   then((rows) => {
    // `rows` is an array of rows
    // each row being an array of cells.
    rows.forEach(row => {
        if(row[0] == 'C'){
            for(let n = -k; n <= k; n++){
                //summation
                for(let t = 0; t < 1/sectionCount/dt;↩
                    t++){
                    const cube = fCubicCalc(t * dt, ↩
                       row);
                    const add = math.multiply(dt, ↩
                       math.multiply(cube, math.pow(↩
                       math.e, math.multiply(math.↩
                       complex(0, 1), -2 * n * math.↩
                       pi * (t * dt + row[9])))));
                    CnDict[n] = math.add(CnDict[n], ↩
                       add);
                }
            }
        }
        else if(row[0] == 'V'){
            for(let n = -k; n <= k; n++){
                for(let t = 0; t < 1/sectionCount/dt;↩
                    t++){
                    const f =
                    math.add(
                        row[1],
                        math.multiply(
                            math.complex(0, 1),
                            math.add(
                                row[2],
                                math.multiply(
                                    subTimeCalc(t * ↩
                                       dt),
                                    math.add(
                                        row[4],
                                        -1 * row[2]
                                    )
                                )
                            )
                        )
                    );
                    const add =
                    math.multiply(dt,
                        math.multiply(
                            f,
                            math.pow(math.e, math.↩
                               multiply(math.complex↩
                               (0, 1), -2 * n * math.↩
                               pi * (t * dt + row[9])↩
                               ))
                        )
                    );
                    CnDict[n] = math.add(CnDict[n], ↩
                       add);
                }
            }
        }
        else if(row[0] == 'L'){
            for(let n = -k; n <= k; n++){
                for(let t = 0; t < 1/sectionCount/dt;↩
                    t++){
                    const f = math.add(
                        math.add(
                            row[1],
                            math.multiply(
                                subTimeCalc(t * dt),
                                row[3] - row[1]
                            )
                        ),
                        math.multiply(
                            math.complex(0, 1),
                            math.add(
                                row[2],
                                math.multiply(
                                    subTimeCalc(t * ↩
                                       dt),
                                    math.add(
                                        row[4],
                                        -1 * row[2]
                                    )
                                )
                            )
                        )
                    );
                    const add =
                    math.multiply(
                        dt,
                        math.multiply(
                            math.pow(math.e, math.↩
                               multiply(math.complex↩
                               (0, 1), -2 * n * math.↩
                               pi * (t * dt + row[9])↩
                               )),
                            f
                        )
                    )
                    CnDict[n] = math.add(CnDict[n], ↩
                       add);
                }
            }
        }
});

let universalTime = 0;
function finalF(){
    while(universalTime <= 1){
        let curr = math.complex(0, 0);
```

```
            Object.entries(CnDict).forEach(pair => {
                [currN, cn] = pair;
                //for a specific time
                //calculate f(t)
                //plot
                curr = math.add(
                    curr,
                    math.multiply(
                        cn,
                        math.pow(
                            math.e,
                            math.multiply(
                                math.complex(0, 1),
                                currN * 2 * math.pi *↩
                                    universalTime
                            )
                        )
                    )
                )
            });
            plotPoint(curr.re, curr.im);
            universalTime += dt;
        }
    }

    finalF();
    console.log('<img src="' + canvas.toDataURL() + '↩
        " />');
})
```