Application of Fractal Geometry in Generating Variations of East Kalimantan Batik Motifs Based on Julia Set

Razi Rachman Widyadhana - 13523004[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]rr.widyadhana@gmail.com, 13523004@std.stei.itb.ac.id

***Abstract*—Mathematics is not solely a field of theory and calculation; it also embodies a form of beautiful art. Indonesia is well known for its diverse backgrounds, cultures, and heritage. One of Indonesia's intangible cultural heritages is Batik. The creation of Batik motifs has advanced significantly, originating from traditional handcrafting methods and expanding to applying mathematical principles, particularly fractal geometry. This paper introduces Julia set as the fractal geometry implementation for developing a novel art to Batik called Fractal Batik. This paper opens up new creative possibilities for developing Batik motifs in Indonesia from a Mathematical perspective.**

Keywords—Batik, Complex Numbers, East Kalimantan, Fractal Geometry, Julia Set.

I. Introduction

Indonesia is well known for its diverse backgrounds, cultures, and heritage. One of Indonesia's intangible cultural heritages is Batik, officially recognized as UNESCO's Intangible Cultural Heritage of Humanity on October 2, 2009. Batik is a decorative pattern made by writing or applying wax to fabric. Etymologically, the word *batik* comes from the Javanese language, derived from the word *amba*, which means 'to draw,' and *tik*, which means 'dot' (a verb meaning to make dots). It later developed into the term Batik [1]. Indonesia features numerous Batik motifs characteristic of various regions, including East Kalimantan, representing the province's rich cultural identity.

Fig. 1. Examples of East Kalimantan Batik. Adapted from [2]

East Kalimantan is a province in Indonesia known for its rich natural resources and cultural diversity. The region is home to various ethnicities, such as the Dayak and Kutai, whose traditions and artistic expressions have significantly influenced local craftsmanship, particularly in Batik.

The creation of Batik motifs has advanced significantly, originating from traditional handcrafting methods and expanding to applying mathematical principles, particularly fractal geometry. Fractal geometry is a branch of mathematics that studies the properties and behaviors of various fractals. Unlike most conventional mathematical forms, fractals generally have irregular shapes that do not follow linearity.

Batik and fractals represent two distinct domains. Batik is a form of artistic expression, whereas fractals are mathematical objects related to iteration and self-similarity processes by recursive or iterative algorithms. Despite the differences, these concepts introduce a novel art to Batik called Fractal Batik. It utilizes fractals to generate and redesign patterns using computational techniques.

Therefore, this paper reveals how Python programming and Photo Editor generate creative variations of East Kalimantan Batik motifs. This paper introduces Julia set as the fractal geometry implementation for developing the Batik motifs.

II. Theoretical Foundations

A. Complex Numbers

The largest set of numbers in mathematics is the set of complex numbers ($\mathbb{C}$). The set of real numbers ($\mathbb{R}$), commonly used in everyday life, is a subset of complex numbers. Any number, k such that $k^2 \geq 0$ is a real number and belongs to the set of real numbers ($\mathbb{R}$). To deal with a number k such that $k^2 < 0$ requires the introduction of a new symbol (number). In 1777, Leonhard Euler introduced the "imaginary" number, denoted by i to stand in for $i = \sqrt{-1}$ which gave birth to complex numbers [3].

A complex number z is an expression with two terms (or components) of real numbers. Represented as

$$z = a + bi$$

where a is the real part of z, b is the imaginary part of z, and i is the imaginary unit that satisfies the equation $i^2 = -1$.

The notation $\mathrm{Re}(z)$ or $\Re(z)$ represents the real part of a complex number z, whereas $\mathrm{Im}(z)$ sor $\Im(z)$ represents its imaginary part. Thus

$$\mathrm{Re}(z) = (4 - 3i) = 4 \quad \text{and} \quad \mathrm{Im}(4 - 3i) = -3$$

A real number is a complex number $a + 0i$ whose imaginary part is equal to 0. A purely imaginary number bi is a complex number $0 + bi$ whose real part is zero. It is common to represent a for $a + 0i$ and bi for $0 + bi$. In this way, the complex numbers contain the ordinary real numbers while extending them to solve problems that cannot be solved with real numbers alone.

Since a complex number is a unique specification based on an ordered pair of real numbers (a, b), complex numbers have a one-to-one correspondence with points on a plane defined as the complex plane. Complex numbers expand the idea of the 1D number line into the 2D complex plane, where the horizontal axis represents the real part and the vertical axis represents the imaginary part. A complex number $a + bi$ corresponds to the point (a, b) in the complex plane. The resulting plot is known as an Argand Diagram [4].

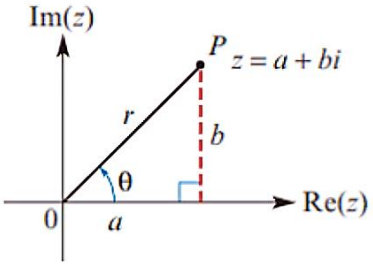


Fig. 2. Argand Diagram. Adapted from [5]

Hence, in terms of its real and imaginary parts, a complex number z is equal to $\mathrm{Re}(z) + i \cdot \mathrm{Im}(z)$. The magnitude of z is referred to as its modulus, denoted in (1)

$$r = |z| = \sqrt{a^2 + b^2} \tag{1}$$

The angle θ is called the argument of z, given as in (2)

$$\theta = \begin{cases} \tan^{-1}(b/a), & a > 0 \\ 180° + \tan^{-1}(b/a), & a < 0 \end{cases} \tag{2}$$

Letting $|z| = r$ expresses the complex number in (3)

$$z = a + bi = r(\cos\theta + i\sin\theta) \tag{3}$$

This form defines the polar representation of z. Applying Euler's formula further simplifies the equation to (4)

$$z = r \cdot e^{i\theta} \tag{4}$$

Standard arithmetic operations manipulate complex numbers in the same way as real numbers. Operations with complex numbers follow algebraic properties such as associativity, commutativity, and distributivity, along with the defining equation $i^2 = -1$.

1) Addition and Subtraction: Performed by adding or subtracting the real part and imaginary part separately, expressed with

$$(a + bi) \pm (c + di) = (a \pm c) + (b \pm d)i$$

2) Multiplication: Defined in the following formula

$$(a + bi) \cdot (c + di) = ac + adi + cbi + bdi^2$$

By $i^2 = -1$, it simplifies the equation to

$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + cb)i$$

3) Division: The division of complex numbers is more complicated than real numbers because of the imaginary component. However, the division of complex numbers can be simplified with the help of the conjugate of complex numbers. The conjugate of a complex number $z = a + bi$ is defined as

$$z^* = a - bi$$

The product of a complex number and its conjugate has a unique result:

$$zz^* = (a + bi)(a - bi) = a^2 + b^2$$

Shown that the product of a number and its conjugate will remove the imaginary component. From it, the division of complex numbers is defined as

$$\frac{(a + bi)}{(c + di)} = \frac{(a + bi)}{(c + di)} \cdot \frac{(c - di)}{(c - di)} = \left(\frac{ac + bd}{c^2 + d^2}\right) + \left(\frac{bc - ad}{c^2 + d^2}\right) i$$

Unique properties further distinguish complex numbers from real numbers, which every polynomial equation in complex numbers guarantees at least one solution, providing a more comprehensive approach to solving algebraic equations.

B. Fractal

The term fractal originates from the Latin word *fractus*, which relates to the verb *frangere*, meaning 'to break,' and describes the formation of irregular or fragmented shapes. In mathematics, fractals represent geometric forms that display infinite repetition of patterns. These repeating structures show self-similarity, with each part resembling the whole, regardless of scale. The arrangement of these patterns varies without restriction to a single orientation and often appears irregular or twisted, ranging in size from tiny to massive.

Initially studied as mathematical objects [5], fractals demonstrate the property of self-similarity, classified into two types: regular and random. Regular fractals display exact self-similarity, where each part of the fractal object precisely mirrors the entire structure when observed at different scales. Examples of regular fractals include the Barnsley Fern, the Sierpinski Triangle, and the Cantor set. In contrast, random fractals display statistical self-similarity, where patterns resemble each other probabilistically rather than exactly. Examples of random fractals include the Julia set and the Mandelbrot set [6]

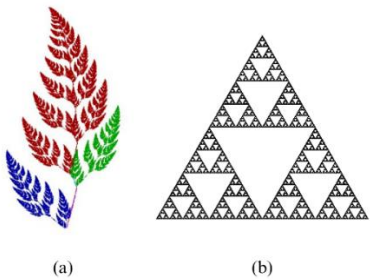


Fig. 3. Examples of regular fractals: (a) The Barnsley Fern and (b) The Sierpinski Triangle. Adapted from [7]

C. Julia Set

The Julia set is created by defining specific boundaries within the area used for its representation. The Julia set consists of two main components: the Julia set itself and the filled-in Julia set. The Julia set and the Filled-In Julia set are related mathematical concepts, each with distinct characteristics. The Filled-In Julia set represents the entire stable region, whereas the Julia set highlights the intricate patterns along its edge.

In 1915, Gaston Julia, a French mathematician, discovered the Julia set and studied fractals by examining rational polynomial expressions of varying degrees. Julia introduced the iterative mapping function $f_c(z) : \mathbb{C} \to \mathbb{C}$, which defines the general form of the Julia set as shown in (5)

$$f(z) = z^2 + c \tag{5}$$

In (4), the variable z takes the form $a + bi$ and it can represent all values in the complex plane. The quantity c defines a complex number that remains constant for any given Julia set, making it a parameter. In other words, infinitely many Julia sets exist, each corresponding to a unique value of c. Smaller values of c (e.g.,$|c| < 2$) often generate visually appealing patterns.

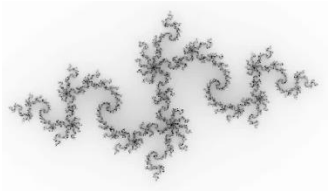


Fig. 4. Julia set with $c = -0.835 - 0.2321i$. Adapted from [8]

When used only once, (5) has a very low chance of producing anything visually interesting. However, repeatedly iterating this equation can generate a Julia set. By feeding the output of the expression $f(z)$ back into the equation as a new value for z, the process becomes an iteration, functioning like a feedback loop. Thus, for any arbitrary value of n, the iteration is defined as in (6)

$$f_{n+1}(z) = {z_n}^2 + c \tag{6}$$

Each new value of $f(z)$ obtained from the calculation serves as the next input for z through a feedback iteration process.

In the special case where $c = 0 + 0i$, the Julia set takes the form of a circle (not a fractal) with a radius of 1. For any z, this transformation involves a contraction (when $|z| < 1$) caused by multiplication with $|z|$, as well as a doubling of the polar angle of z, followed by a translation by c.

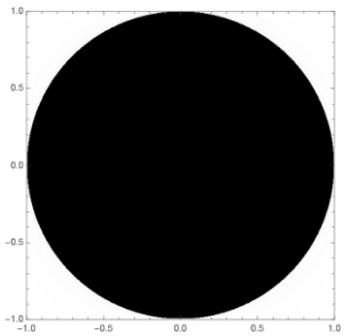


Fig. 5. Julia set with $c = 0 + 0i$. Adapted from [9]

In mathematics, several types of connectivity exist. For Julia sets, the relevant type is path connectivity, meaning a path can be traced from one point in the set to another without leaving the set. As demonstrated independently by Julia and Fatou, connected Julia sets are globally connected, not just locally connected. Topologically, connected Julia sets form a highly deformed circle or a curve with infinitely many branches and sub-branches, referred to as a dendrite, as shown in Figure 3.

Benoît B. Mandelbrot, known as the Father of Fractal Geometry, referred to disconnected Julia sets as "dust" or "Fatou dust," a term named after Pierre Fatou (1878–1929). This terminology is appropriate because disconnected Julia sets consist of individual points scattered across the complex plane, resembling dust particles spread over a surface, with no connections between them.

Another term used to describe such sets is Cantor dust. A Cantor set is a completely disconnected set formed by repeatedly dividing the line segment $[0,1]$ into three parts, removing the middle segment and leaving the segments $[0, 1/3]$ and $[2/3, 1]$. This process is repeated infinitely for each remaining segment.

The distribution of points in a disconnected Julia set qualitatively resembles the appearance of Cantor dust, as both are entirely disconnected, with each point isolated from the others. Disconnected sets are completely unconnected, forming an infinite collection of points that are isolated from one another.

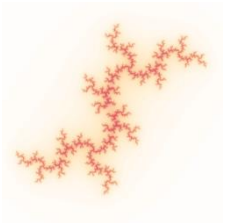


Fig. 6. Disconnected Julia set with $c = 0 + 0.8i$. Adapted from [10]

III. Implementation

The instruments used in this paper consist of two main components, namely hardware and software. The hardware specifications used are: Intel i7-8550U Processor, Intel UHD Graphics 620 GPU, and 8 GB RAM. Then, the software specifications used are Python 3.12.4 programming language for generating the Julia set, Jupyter Notebook for the program kernel, and Figma as Photo Editor for designing and integrating the fractal patterns into Batik motif variations.

A. Creating the Julia Set

The process starts by defining the necessary parameters, including the complex constant c and the maximum number of iterations N. The algorithm divides the complex plane into a grid of points, each corresponding to a complex number z. These points initialize a value for the iterative process. The algorithm repeatedly computes transformations for each point by using (2).

In each iteration, the algorithm checks whether the magnitude of z exceeds a predefined threshold, typically set to 2, or whether it reaches the maximum number of iterations N. The threshold value of 2 ensures mathematical correctness, as points with

magnitudes greater than two always escape to infinity under iteration. For instance, if $|z| > 2$, subsequent iterations of (2) continue growing without bounds. This property allows the algorithm to identify the escape points.

Finally, the algorithm processes the iterations and masks the data to generate visual representations of the Julia set. It includes a custom color palette parameter that enables customization of the fractal's appearance, supporting its use in generating variations of Batik motifs. The output provides flexibility for further design modifications and integration. The Python code below demonstrates the implementation of this process.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap

class JuliaSet:
    def __init__(self, c: complex, N: int = 200, dimension: int = 1000):
        self.c = c
        self.N = N
        self.dimension = dimension
        self.escapeThreshold = 2.0

    def compute(self):

        # Define range and grid
        xMin, xMax, yMin, yMax = -2, 2, -2, 2
        x, y = np.meshgrid(np.linspace(xMin, xMax, self.dimension),
                           np.linspace(yMin, yMax, self.dimension) * 1j)

        # Combine real and imaginary parts into a single complex grid
        z = x + y

        # Create a mask and track points that never escaped
        mask = np.full(z.shape, True, dtype=bool)
        neverEscaped = np.ones([self.dimension, self.dimension], dtype=bool)

        # Initialize F with zeros
        F = np.zeros([self.dimension, self.dimension], dtype=float)

        # Iterative Julia set algorithm
        for j in range(self.N):
            z[mask] = z[mask] ** 2 + self.c
            escaped = np.abs(z) > self.escapeThreshold
            neverEscaped &= ~escaped
            F[escaped & mask] = j
            mask = ~escaped

            if not np.any(mask):
                break

        # Handle |c| ≤ 2 (Connected Julia set)
        if abs(self.c) <= 2:
            F[neverEscaped] = self.N

        # Normalize escape times
        self.x = np.linspace(xMin, xMax, self.dimension)
        self.y = np.linspace(yMin, yMax, self.dimension)
        self.F = F / self.N
        self.mask = np.ma.masked_where(self.F < 0.01, self.F)

    def visualize(self, ax=None, N: int = 50, palette: str = 'BLACK'):

        # Set up plotting and palette
        if ax is None: ax = plt.gca()

        chosenPalette = [list(color) for color in COLOR_PALETTES[palette]]
        cmap = LinearSegmentedColormap.from_list('custom', chosenPalette, N=N)

        # Plot on the specified axis
        ax.pcolormesh(self.x, self.y, self.mask, cmap=cmap)
        ax.axis('equal')
        ax.axis('off')

    def save(self, index: int, N: int = 50, palette: str = 'BLACK'):

        # Set up plotting and palette
        plt.figure(figsize=(7, 7))

        chosenPalette = [list(color) for color in COLOR_PALETTES[palette]]
        cmap = LinearSegmentedColormap.from_list('custom', chosenPalette, N=N)

        # Plot the image
        plt.pcolormesh(self.x, self.y, self.mask, cmap=cmap)
        plt.axis('equal')
        plt.axis('off')

        plt.savefig(f'juliaSets/julia-{index:02d}.png', format='png', transparent=True,
                    bbox_inches='tight', pad_inches=0, dpi=300)
        plt.close()

        DATABASE = DATABASE.append({'c': self.c, 'palette': palette})
        DATABASE.to_csv('JuliaSet.csv', index=False)
```

Fig. 7. Julia Set Class and Algorithm in Python

Selecting the value of c is crucial for determining the desired shape of the Julia set. Combinations of c values fall into three forms:

- c with only real values ($\mathrm{Re}(c) \neq 0, \mathrm{Im}(c) = 0$)
- c with only imaginary values ($\mathrm{Re}(c) = 0, \mathrm{Im}(c) \neq 0$)
- c with both real and imaginary values ($\mathrm{Re}(c) \neq 0, \mathrm{Im}(c) \neq 0$)

For the first combination, the range $-1{,}5 \leq \mathrm{Re}(c) \leq 0.5$ provides a variety of Julia set structures suitable for motif generation. Values beyond this range often produce patterns that lack the desired structural coherence, less effective as motifs. Focusing on this range ensures that the resulting designs retain aesthetic appeal, suitable for developing the Batik motifs.

```python
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

values = [complex(0.3, 0), complex(-0.6, 0), complex(-1.2, 0), complex(-1.5, 0)]
palletes = ['BLACK', 'BLACK', 'GOLD', 'BLACK']
labels = ['a', 'b', 'c', 'd']

for i, (c, ax) in enumerate(zip(values, axes)):
    julia = JuliaSet(c)
    julia.compute()
    julia.visualize(ax=ax, palette=palletes[i])
    julia.save(i+1, palette=palletes[i])
    ax.set_title(getTitle(c, labels[i]), y=-0.1, fontsize=14)

plt.show()
```

Fig. 8. First Julia Set Combination Creation in Python Code

Fig. 8. creates selected samples with each defined color used for this combination.

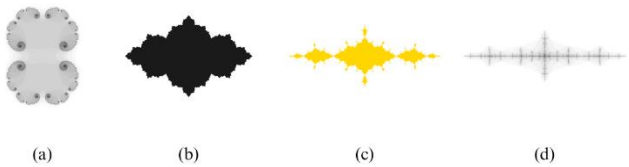


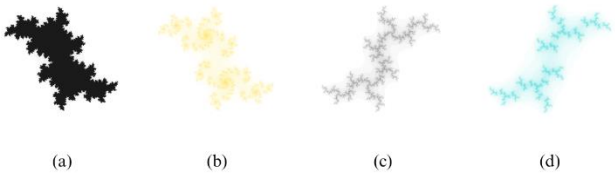
Fig. 9. Julia set with (a) $c = 0.3$, (b) $c = -0.6$, (c) $c = -1.2$, (d) $c = -1.5$

In the second combination, the Julia set appears as Cantor dust when the imaginary part of c falls within the range $0.63 < |\mathrm{Im}(c)| \leq 0.9$. As it evolves, the shape becomes thicker and eventually returns to a circular form when $\mathrm{Im}(c) = 0$. The shapes observed in the range $-2 \leq \mathrm{Im}(c) < 0$ differ horizontally from those in the range $0 < \mathrm{Im}(c) \leq 2$. Similar to the first combination, focusing on a specific range of $\mathrm{Im}(c)$ enables the generation of structured and harmonious patterns that reflect the detailed and artistic elements commonly found in Batik motifs.

```python
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

values = [complex(0, 0.63), complex(0, 0.7), complex(0, -0.8), complex(0, -0.9)]
palletes = ['BLACK', 'GOLD', 'BLACK', 'TEAL']
labels = ['a', 'b', 'c', 'd']

for i, (c, ax) in enumerate(zip(values, axes)):
    julia = JuliaSet(c)
    julia.compute()
    julia.visualize(ax=ax, palette=palletes[i])
    julia.save(i+5, palette=palletes[i])
    ax.set_title(getTitle(c, labels[i]), y=-0.1, fontsize=14)

plt.show()
```

Fig. 10. Second Julia Set Combination Creation in Python Code

In Fig. 10., the algorithm creates selected samples with each defined color used for this combination.



(a) (b) (c) (d)

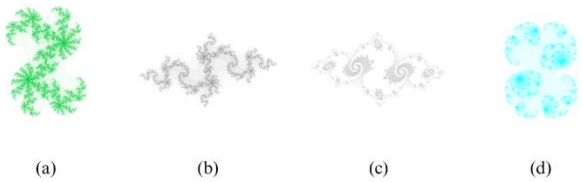
Fig. 11. Julia set with (a) $c = 0.63i$, (b) $c = 0.7i$, (c) $c = -0.8i$, (d) $c = -0.9i$

For the last combination, several samples were selected from [11], [12]. The selection was based on their proportion and clarity with the Batik motifs.

```
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

values = [complex(0.37, 0.1), complex(-0.835,-0.2321),
          complex(-0.772, 0.103), complex(0.285, 0.01)]
palletes = ['GREEN', 'BLACK', 'WHITE', 'CYAN']
labels = ['a', 'b', 'c', 'd']

for i, (c, ax) in enumerate(zip(values, axes)):
    julia = JuliaSet(c)
    julia.compute()
    julia.visualize(ax=ax, palette=palletes[i])
    julia.save(i+9, palette=palletes[i])
    ax.set_title(getTitle(c, labels[i]), y=-0.1, fontsize=14)

plt.show()
```

Fig. 12. Last Julia Set Combination Creation in Python Code

Fig. 12. creates selected samples with each defined color for the last combination



(a) (b) (c) (d)

Fig. 13. Julia set with (a) $c = 0.37 + 0.1i$, (b) $c = -0.835 - 0.2321i$, (c) $c = -0.772 + 0.103i$, (d) $c = 0.285 + 0.01$.

B. Motifs Used

The Batik motifs of East Kalimantan often draw inspiration from nature, featuring patterns representing flora, fauna, and traditional symbols. These motifs serve as decorative art and share cultural narratives and philosophical values deeply rooted in the province's heritage. This paper selects several types of East Kalimantan Batik motifs, as shown in Fig. 14.

(a) (b) (c) (d) (e)

Fig. 15. Selected East Kalimantan Batik Motifs (a) Hiu Taliyasan, (b) Rutun Penyu, (c) Kuntul Perak, (d) The first Tengkawang Ampiek Motif, and (e) The second Tengkawan Ampiek Motif.

C. Generating the Batik Motifs Variations

Before generating the variations, certain elements will be combined to make more visually appealing motifs. The combining process begins by establishing the layering order of the East Kalimantan Batik motif and the selected Julia Set. The first layer is the background, followed by the second layer as the foreground. This order ensures that both elements maintain their visual importance. Ensure both images have proper alignment, achieving a precise and visually perfect combination. The Python code below demonstrates the implementation of this combining process.

```
from PIL import Image

background = Image.open('background.png')
foreground = Image.open('foreground.png')
combined = Image.alpha_composite(background, foreground)
combined.save('combined.png')
```

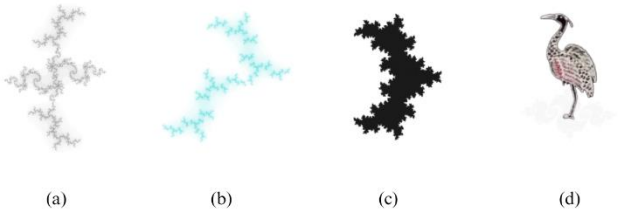
Fig. 14. Combining Algorithm in Python Code

The algorithm in Fig. 14. creates element combinations from selected East Kalimantan Motifs and Julia Sets.

(a) (b) (c) (d)

Fig. 15. Combination of (a) Hiu Taliyasan Motif and the first Julia Set in Fig. 9., (b) The first and the fourth Julia Set in Fig. 13., (c) Rutun Penyu Motif and the second Julia Set in Fig. 11., (d) The fourth Julia Set in Fig. 9. And the second Julia Set in Fig. 13.

Unfortunately, some complex motifs cannot be combined effectively due to the complexity requiring Machine Learning models for precise and perfect combinations. As a result, a manual combination in Figma is used for such cases.



(a) (b) (c) (d)

Fig. 15. Manual Combination of (a) The second Julia Set in Fig. 9., (b) The third Julia Set in Fig. 11., (c) The first Julia Set in Fig. 11., (d) Kuntul Perak Motif and the third Julia Set in Fig. 13.

After creating new beautiful elements, it's time to generate the Batik Motifs Variations. The layout for the generation is shown in Fig. 17.

	0	1	2	
0	1st Image	2nd Image	1st Image	...
1	3rd Image	4th Image	3rd Image	...
2	1st Image	2nd Image	1st Image	...
	...	...	...	and so on ...

Fig. 17. Layout for generate the Batik Motifs Variations

Then, the Python code below implements this generating process by using the predefined layout with four images defined in order corresponding to the layout.

```python
def generateBatikVariations(images, rows, cols, width, height):
    motifWidth = cols * width
    motifHeight = rows * height
    pattern = Image.new('RGBA', (motifWidth, motifHeight), (255, 245, 182, 255))

    for row in range(rows):
        for col in range(cols):
            if (col + 1) % 2 == 1:
                img = images[0] if (row + 1) % 2 == 1 else images[2]
            else:
                img = images[1] if (row + 1) % 2 == 1 else images[3]

            flip = False
            if col % 2 == 1:
                flip = (col // 2) % 2 == 1
            else:
                flip = ((col-1) // 2) % 2 == 1

            if flip:
                img = img.transpose(Image.FLIP_LEFT_RIGHT)

            resizedImage = img.resize((width, height))
            pattern.paste(resizedImage, (col * width, row * height), resizedImage)

    return pattern
```

Fig. 18. Batik Motif Variations Generator Algorithm in Python Code

The first variation uses the first Tengkawang Ampiek motif, the second Julia Set in Fig. 9, the first element combination in Fig. 16., and the first element combination in Fig. 15.

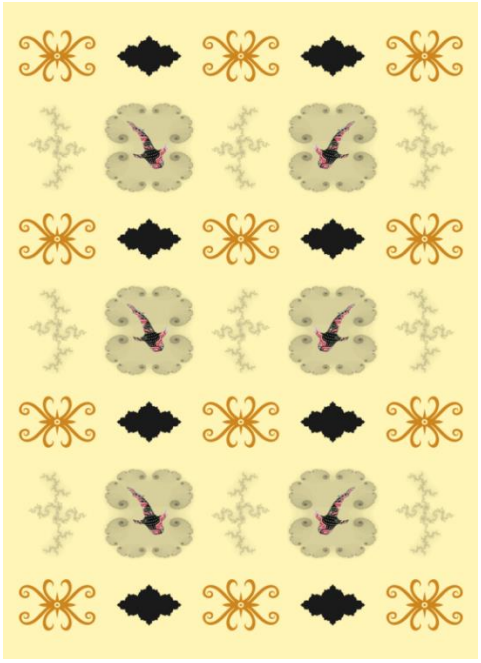


Fig. 19. The first Batik Motifs Variation

For the second variation, the second combination in Fig. 15., the third Julia Set in Fig. 9., the second and the third combination in Fig. 15. will be used to the generator. The generator creates the variation as shown in Fig. 20.

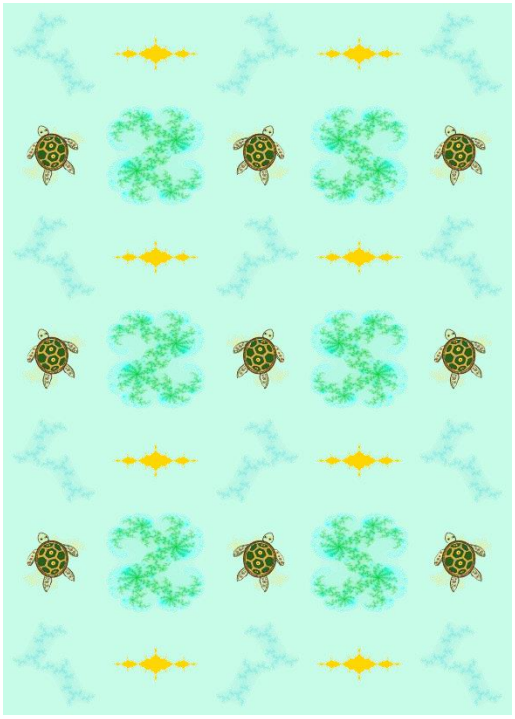


Fig. 20. The second Batik Motifs Variation

The last variation uses the second Tengkawang Ampiek, the fourth combination in Fig. 15., the third and the fourth combination in Fig. 16.

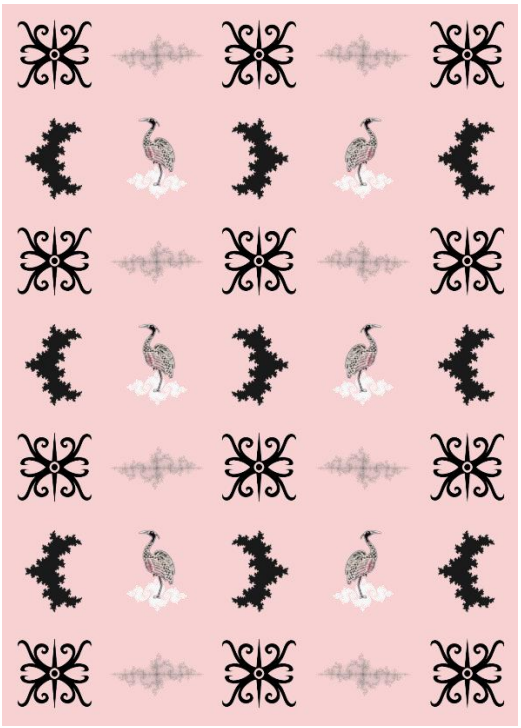


Fig. 21. The second Batik Motifs Variation

IV. Conclusion

This paper presents a method for generating variations of East Kalimantan Batik motifs by applying the principles of fractal geometry, specifically using the Julia Set. The proposed approach successfully merges mathematical concepts with traditional art to create intricate, visually fascinating patterns that respect cultural identity while introducing modern computational techniques.

This approach introduces new creative possibilities for developing Batik motifs in Indonesia from a mathematical perspective. The results showed that fractal-based patterns, such as those obtained from the Julia Set, have enormous potential to develop Batik motif designs. This method bridges the gap between art and mathematics, providing a valuable tool for designers and artists, which will further enhance the richness of Indonesia's cultural heritage.

Along this line, further research may extend this approach to deal with more complex motif geometries, such as overlapping or nested patterns, and possibly integrate Machine Learning models with the aim of optimizing motif combinations.

V. Appendix

The methods and experiments presented in this paper are implemented in the following GitHub repository: https://github.com/zirachw/Algeo-JuliaSet

Further explanations of the implementation in this paper are available in the following YouTube video: https://youtu.be/0__2fKRfFb8

VI. Acknowledgment

The author expresses heartfelt gratitude to God Almighty for His blessings and grace throughout the writing process, allowing the smooth completion of this paper. The author would also like to sincerely thank Dr. Ir. Rinaldi Munir, M.T., the IF2123 Linear Algebra and Geometry K02 course lecturer, for his invaluable guidance and knowledge that significantly contributed to completing this work.

Additionally, the author is deeply grateful to their parents for their moral and spiritual support as the source of inspiration and motivation during the writing journey. Finally, the author wishes to thank the friends in the CIPHER cohort for their mutual support during lectures and for learning together throughout the semester.

May the Almighty God reward all the kindness that has been bestowed. May this paper serve as a valuable contribution to the academic community and be well-received.

References

[1] A. Kamsyakawuni, K. D. Purnomo, and E. K. Wulandari, "Pengembangan Desain Batik Labako dengan Menggabungkan Geometri Fraktal Kurva Naga dan Corak dari Daun Tembakau," *Jurnal Ilmu Dasar*, vol. 18, no. 2, pp. 125-132, Jul. 2017, Accessed: Dec. 29, 2024, doi: 10.19184/jid.v18i2.5650. [Online]. Available: http://repository.unej.ac.id/handle/123456789/102055

[2] Jurnal Kaltim. Accessed: Dec. 28, 2024. [Photo]. Available: https://jurnalkaltim.com/gaya-hidup/batik-kalimantan-timur-ciri-khas-motif/

[3] R. Munir. (2023*). Aljabar Kompleks (Update 2023)*. Accessed: Dec. 28, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-24-Aljabar-Kompleks-2023.pdf

[4] J. Vince, *Geometric Algebra for Computer Graphics*. Ringwood, UK: Springer-Verlag, 2008, ch. 3.

[5] S. Hasang and S. Supardjo. "GEOMETRI FRAKTAL DALAM RANCANGAN ARSITEKTUR," *MEDIA MATRASAIN*, vol. 9, no. 2, pp. 111–124, Aug. 2012. Accessed: Dec. 29, 2024, doi: 10.35793/matrasain.v9i2.665B. [Online]. Available: https://ejournal.unsrat.ac.id/v3/index.php/jmm/article/view/665/540

[6] P. S. Addison, *Fractal and chaos*. Bristol, UK; Philadelphia, USA : Institute of Physics Pub., 1997.

[7] R. Munir. (2023). *Deretan, Rekursi, dan Relasi Rekurens Bagian 1* (Update 2024), pg. 17. Accessed: Dec. 31, 2024. [Photo]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/10-Deretan,%20rekursi-dan-relasi-rekurens-(Bagian1)-2024.pdf

[8] Medium. Accessed: Dec. 31, 2024. [Photo]. Available: https://www.cantorsparadise.com/the-julia-set-e03c29bed3d0

[9] Imgur. Accessed: Dec. 31, 2024. [Photo]. Available: https://imgur.com/gallery/julia-manelbrot-sets-VxdTaQ8

[10] Paul Bourke. Accessed: Dec. 31, 2024. [Photo]. Available: https://paulbourke.net/fractals/juliaset/

Statement

In this statement, I declare that this paper I have written is my own work, not a duplication or translation of someone else's paper, and is not plagiarized.

Bandung, 2 January 2025

Razi Rachman Widyadhana
13523004