

PYCON INDONESIA 2021

The Joy of Creating Art with Code.

Presented by Neeraj Pandey

@NEERAJP99

NEERAJ PANDEY

@NEERAJP99

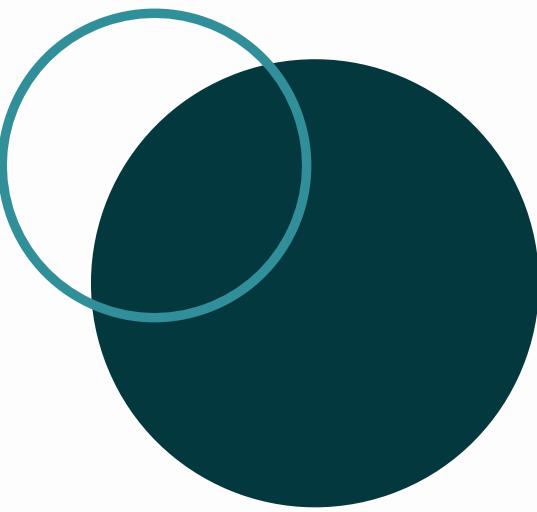
GENERATIVE ARTIST

Senior student at Ashoka University.

Computational Arts, Quantitative Finance,
Full Stack Web and Data Science



@NEERAJP99



POINTS FOR DISCUSSION

Generative Art

History behind Generative Art

Intro to Processing.py

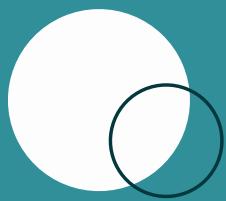
Geometry, Algorithms and Randomness

Examples using Processing.py

Art created through the use of an autonomous system.

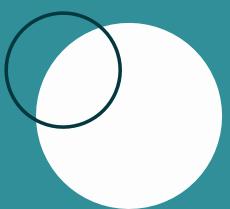
ALGORITHMS, MATHEMATICS, GENETIC SEQUENCES

“



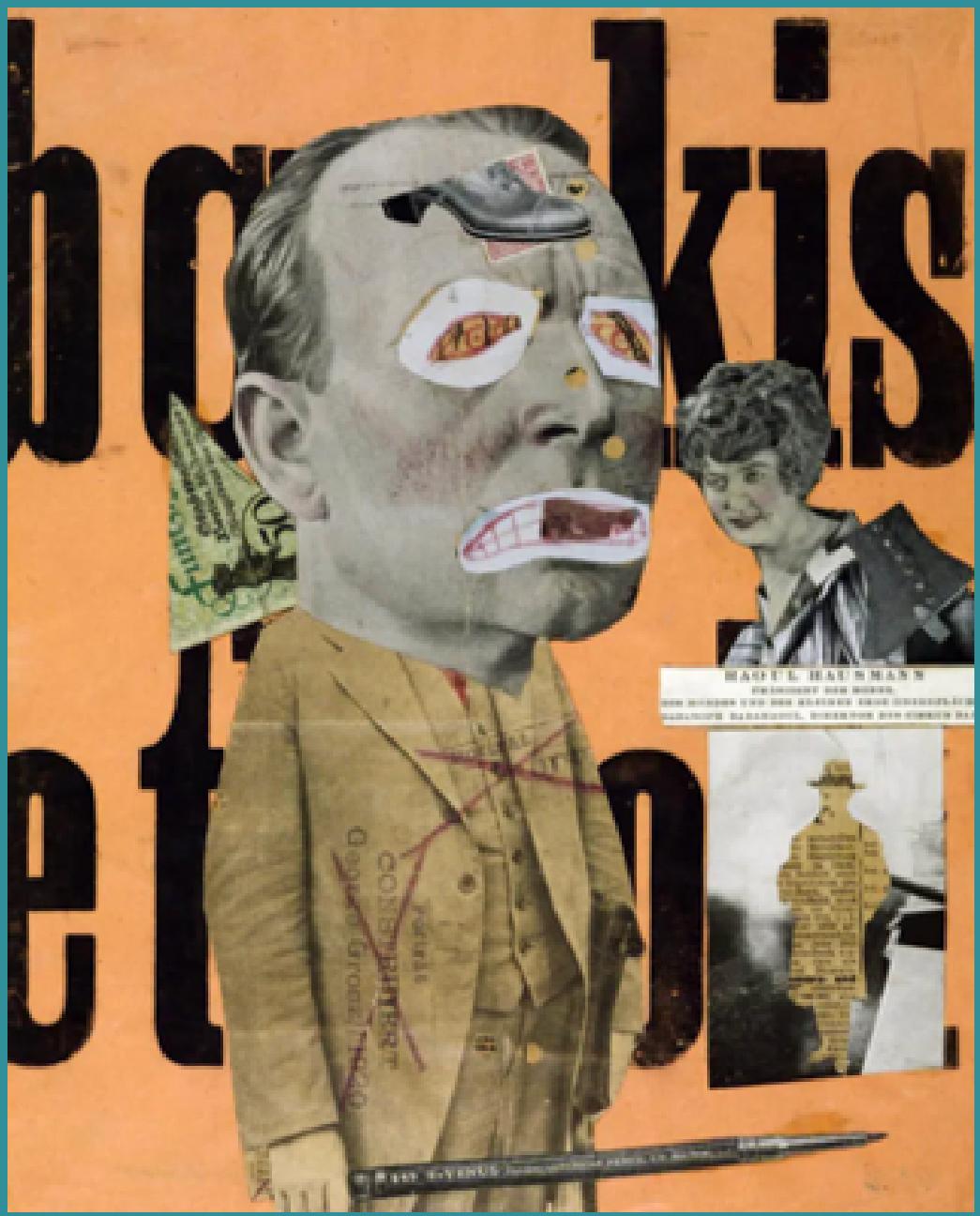
CREATIVITY

“I saw the angel in the marble and carved
until I set him free”



DADAISM

FOUNTAIN (DUCHAMP), 1917

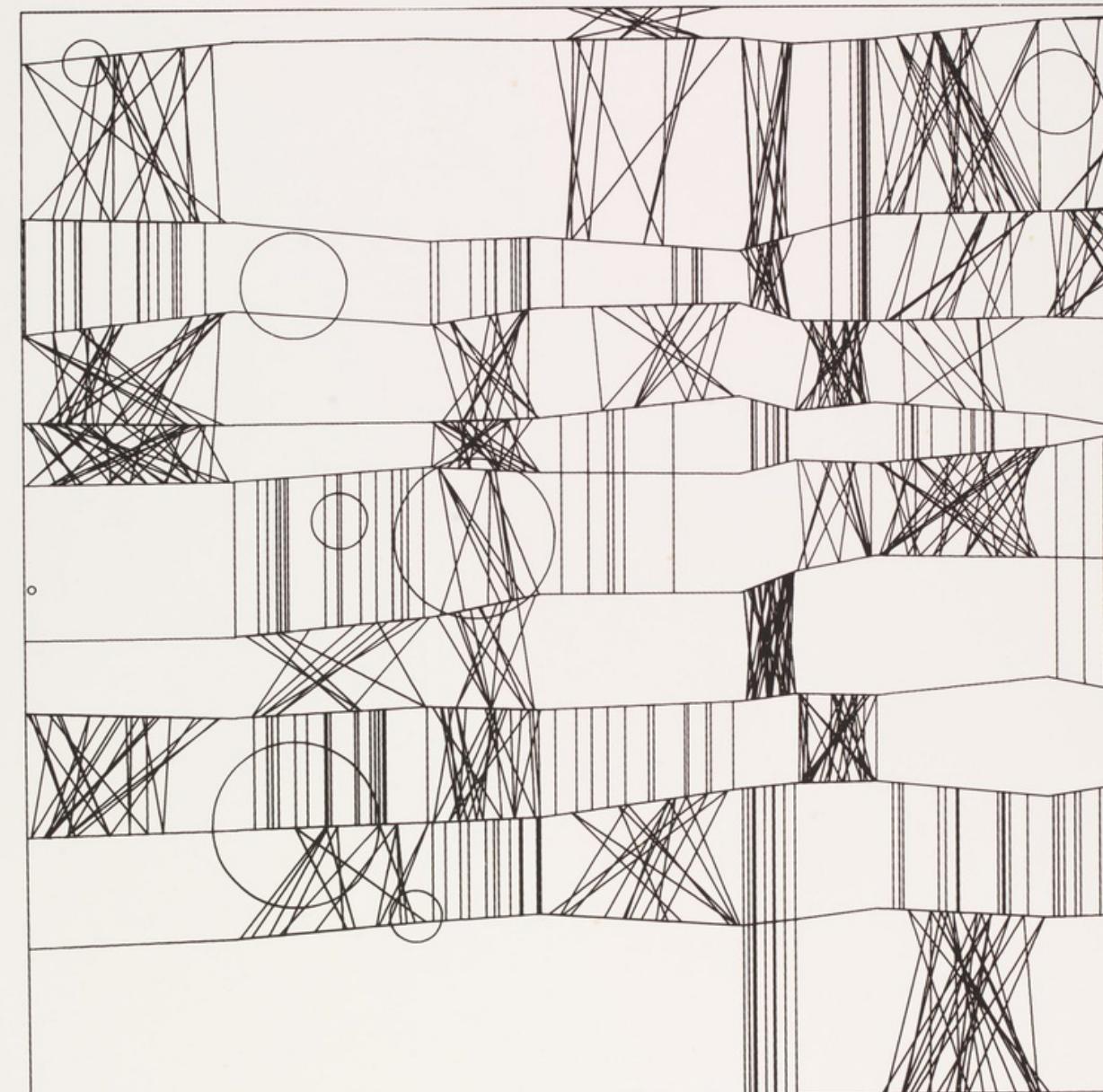


THE ART CRITIC, 1919-20

THE MECHANICAL HEAD (1920)



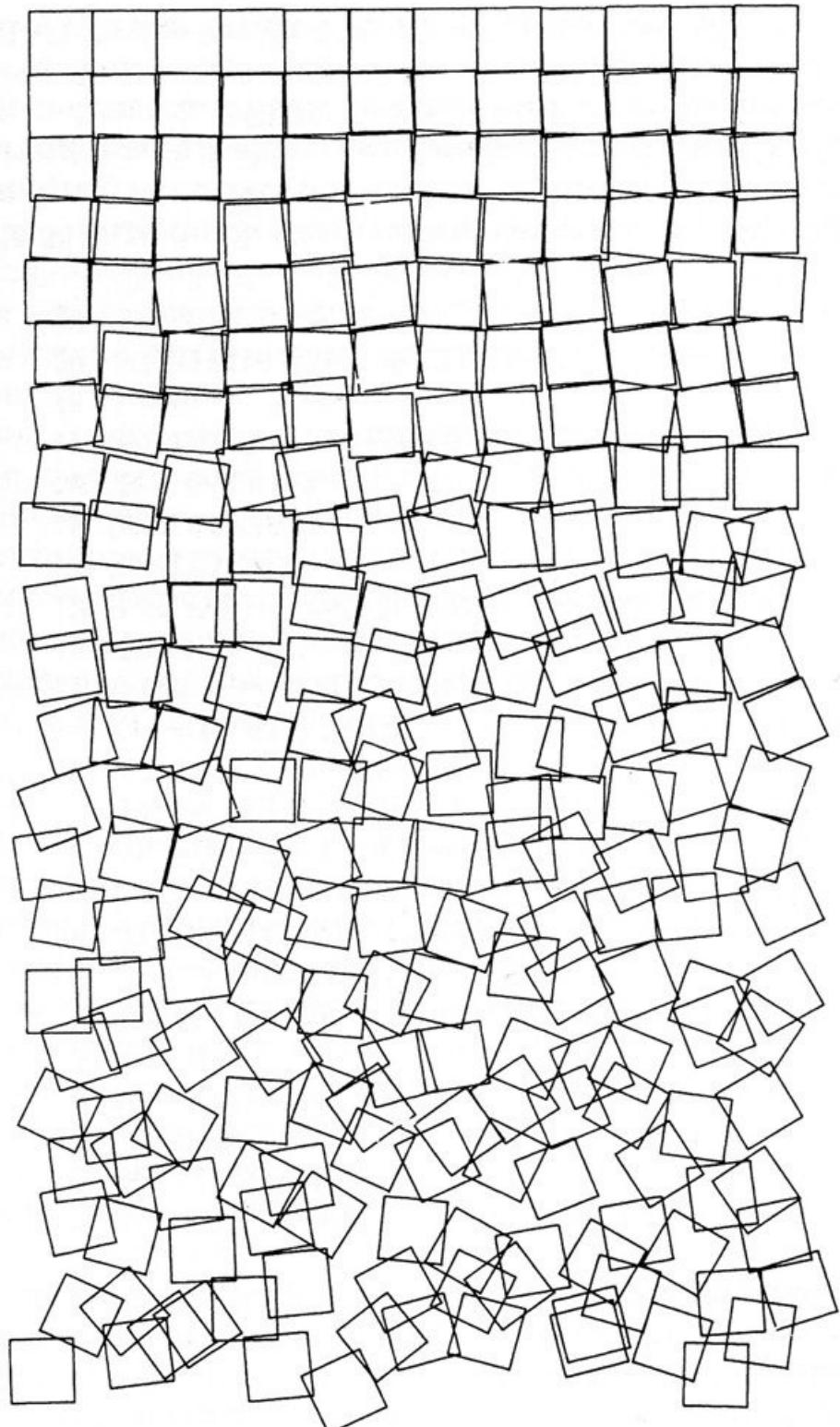
HISTORY OF GENERATIVE ART



Hommage à Paul Klee - Frieder Nake, 1965

One of the earliest and best-known pieces of generative art.

By Georg Nees, 1968



PROCESSING FOUNDATION

Processing

P5.js

Processing.py

Processing for Pi

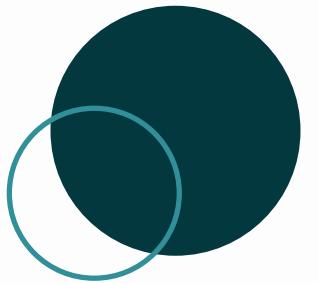
Processing for Android



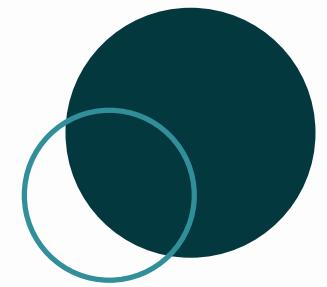
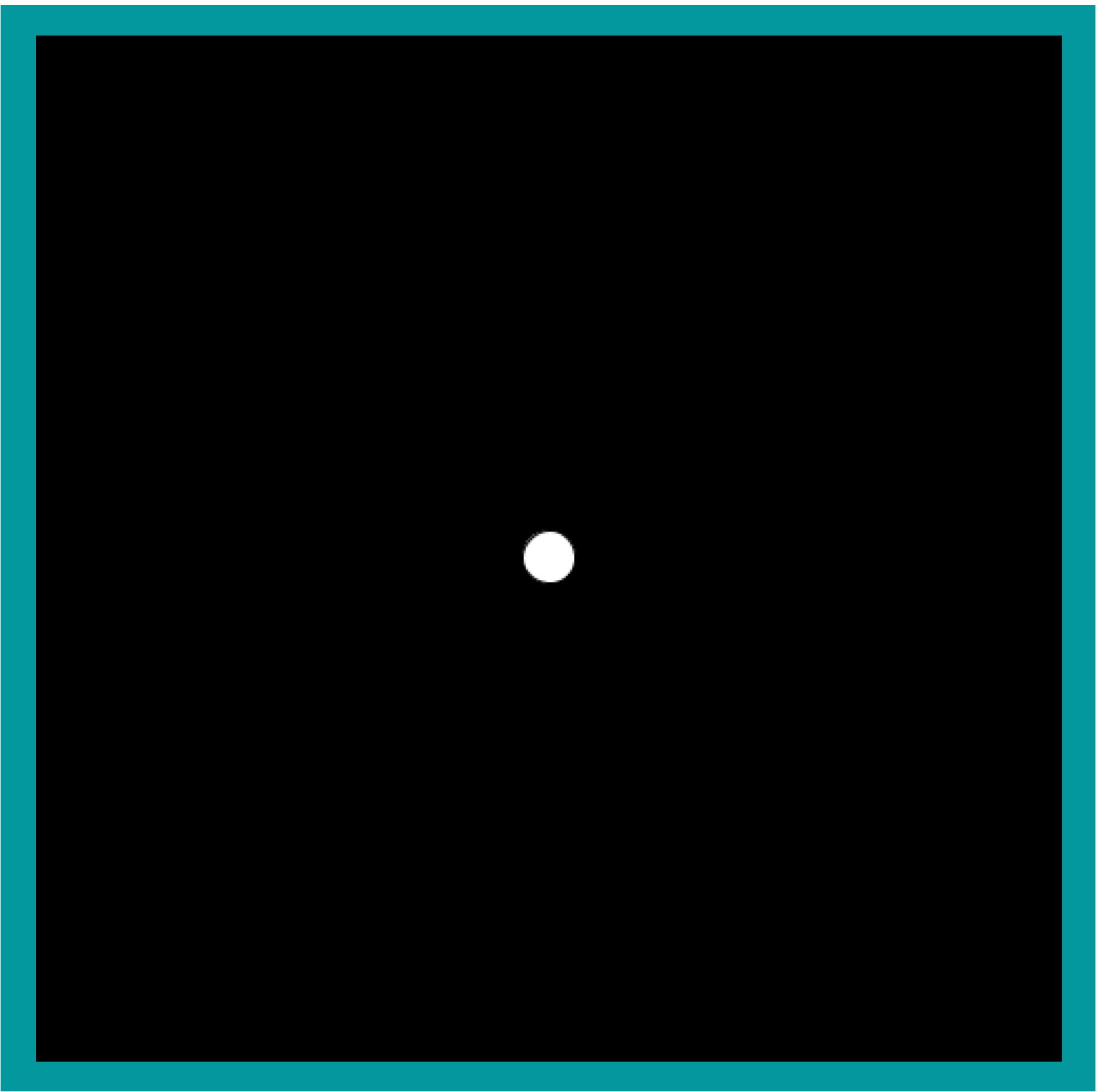
PROCESSING.PY

@NEERAJP99

OVERVIEW

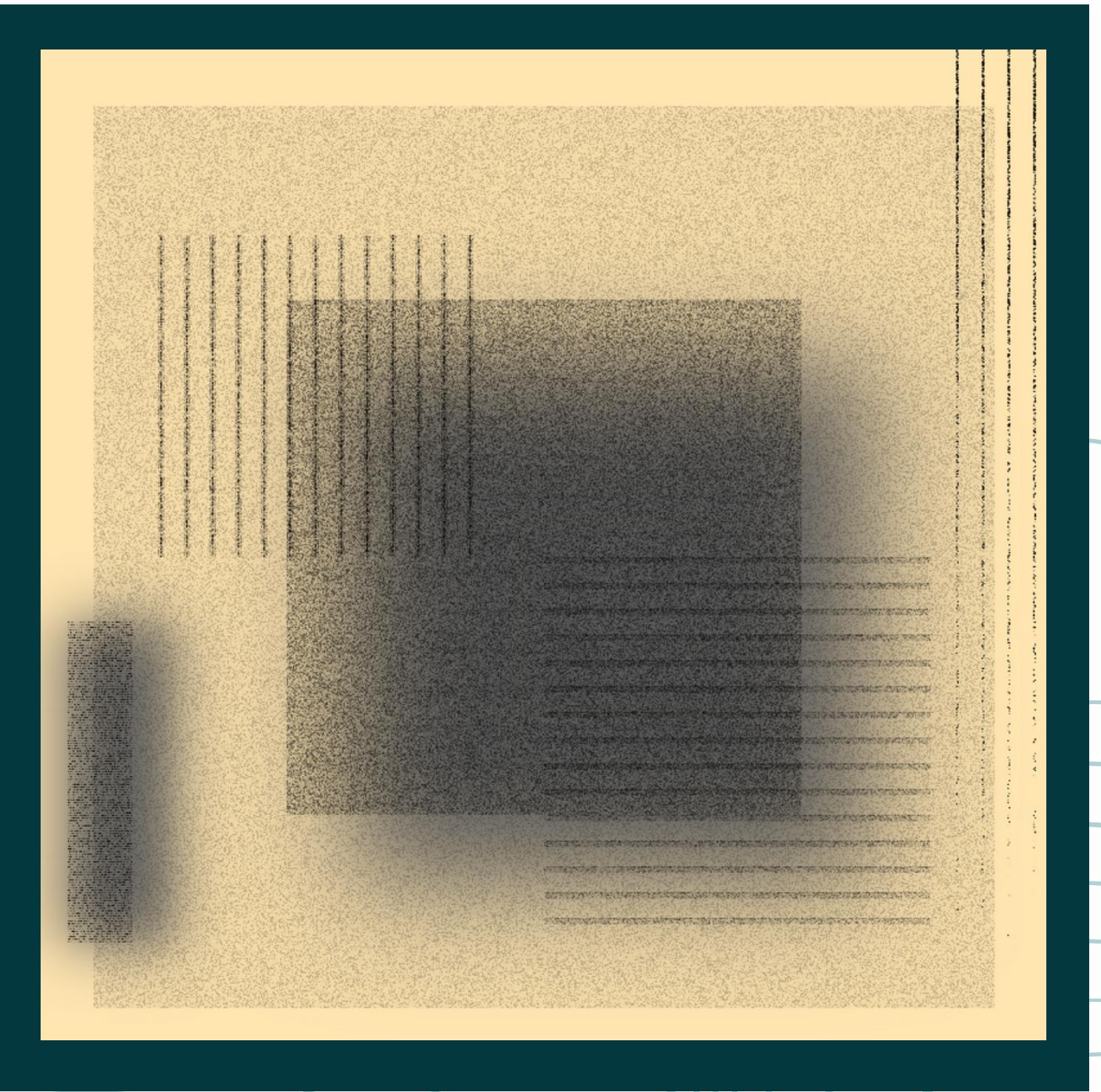


```
● ● ●  
1 x, y = 250, 250  
2 def setup():  
3     size(500, 500)  
4     background(0)  
5  
6 def draw():  
7     global x, y  
8     fill(255)  
9     ellipse(x, y, 25, 25)
```



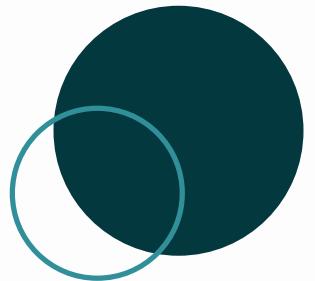
@NEERAJP99

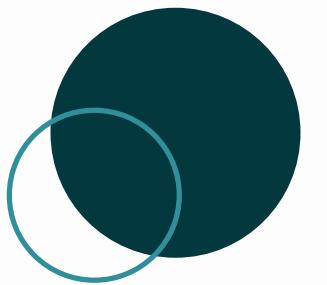
USING MATHEMATICS AND ALGORITHMS



RANDOM

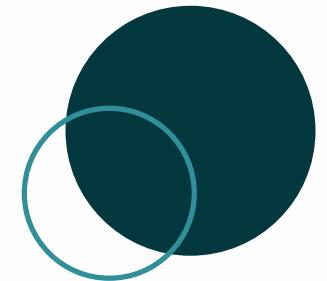
Generates random floating point numbers .





@NEERAJP99

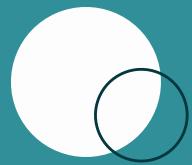
RANDOM



Generates random floating point numbers .

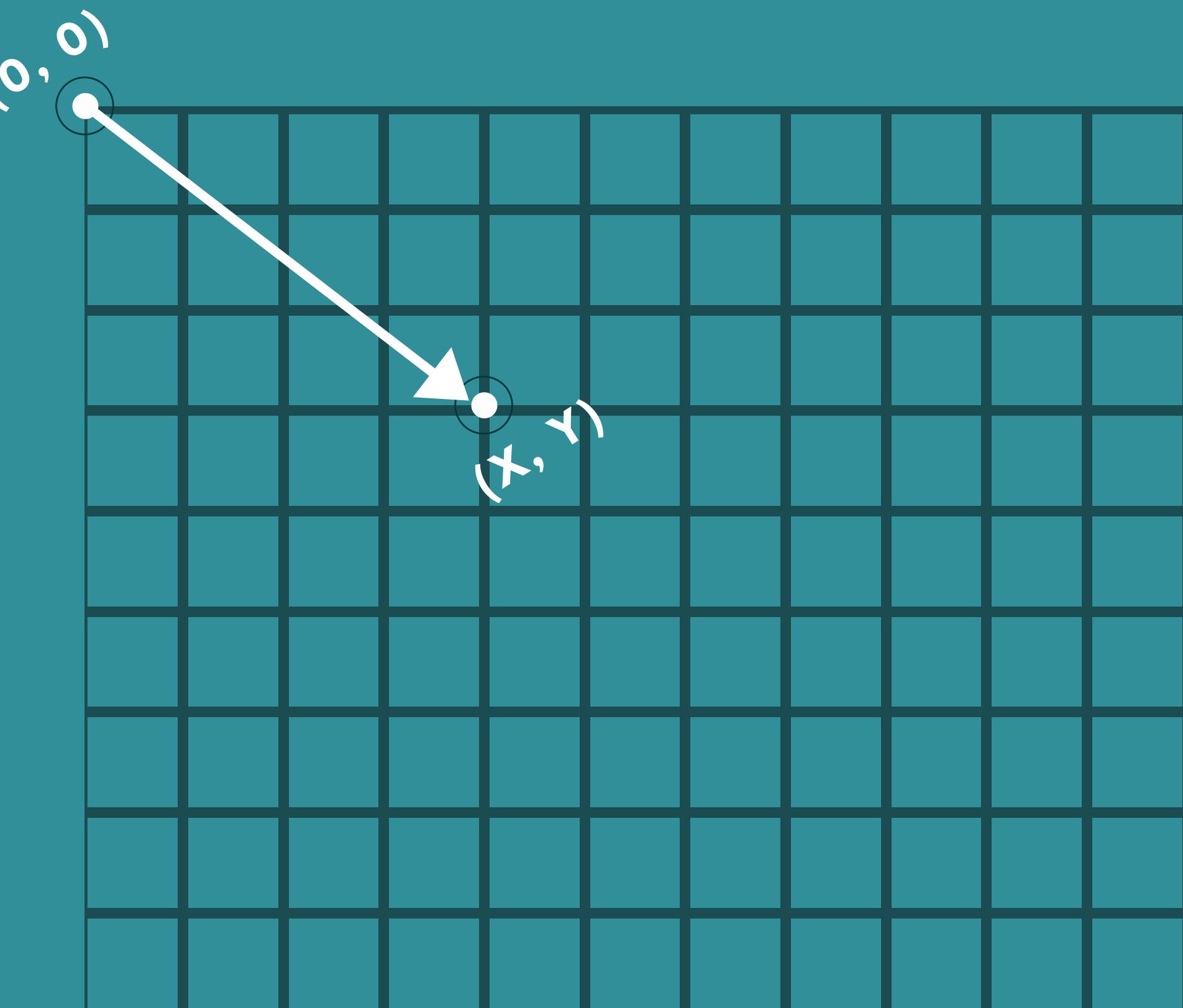
```
● ● ●  
1 random()  
2 # floating value between [0, 1)  
3 random([min], [max])  
4 # Random value between [min, max)  
5 random(x)  
6 # Random value between [0, x)
```

Without processing, use random module in Python.
(<https://docs.python.org/3/library/random.html>)



DRAWING A POINT

We consider a 2-D cartesian plane, and each point as a vector.

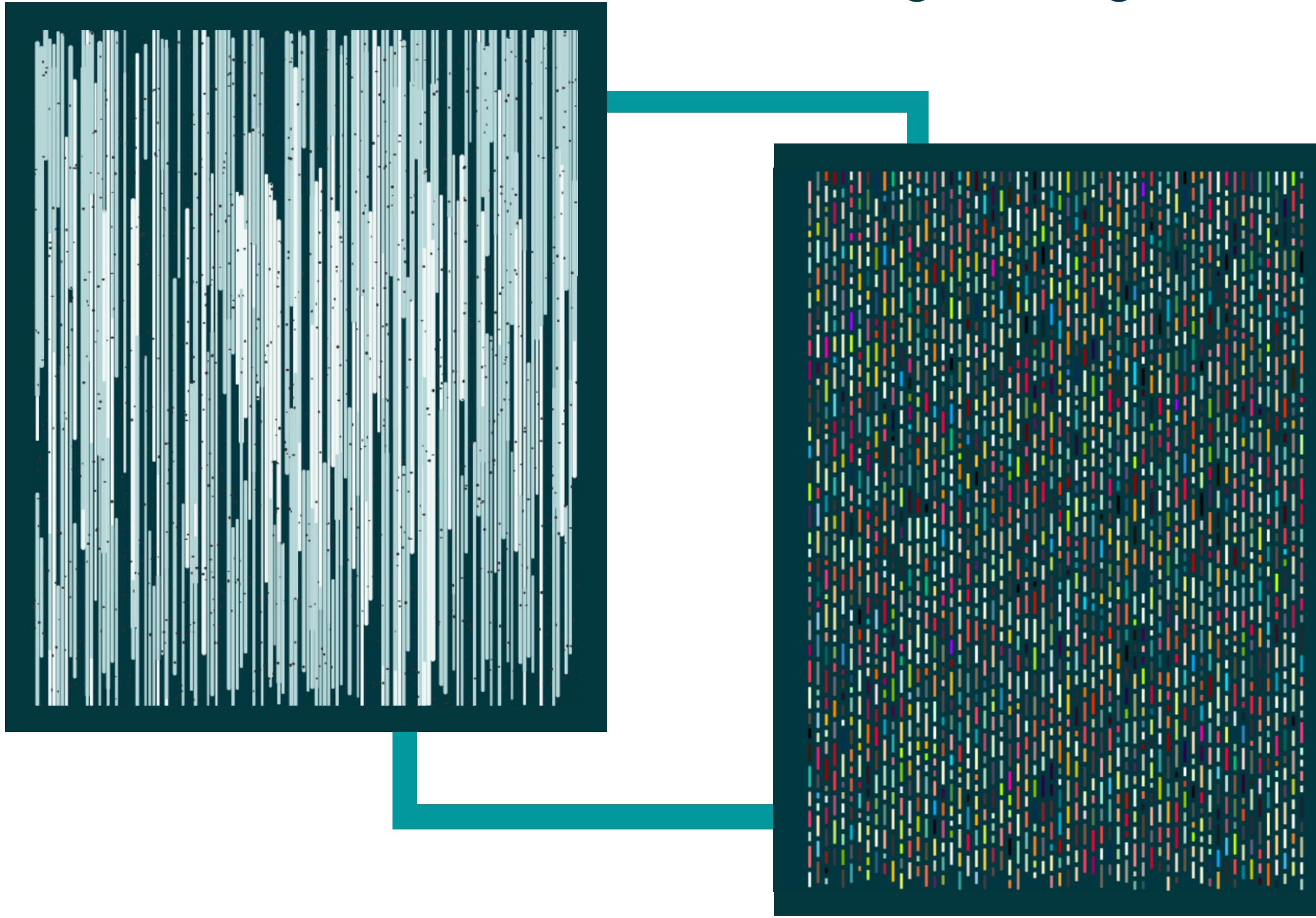


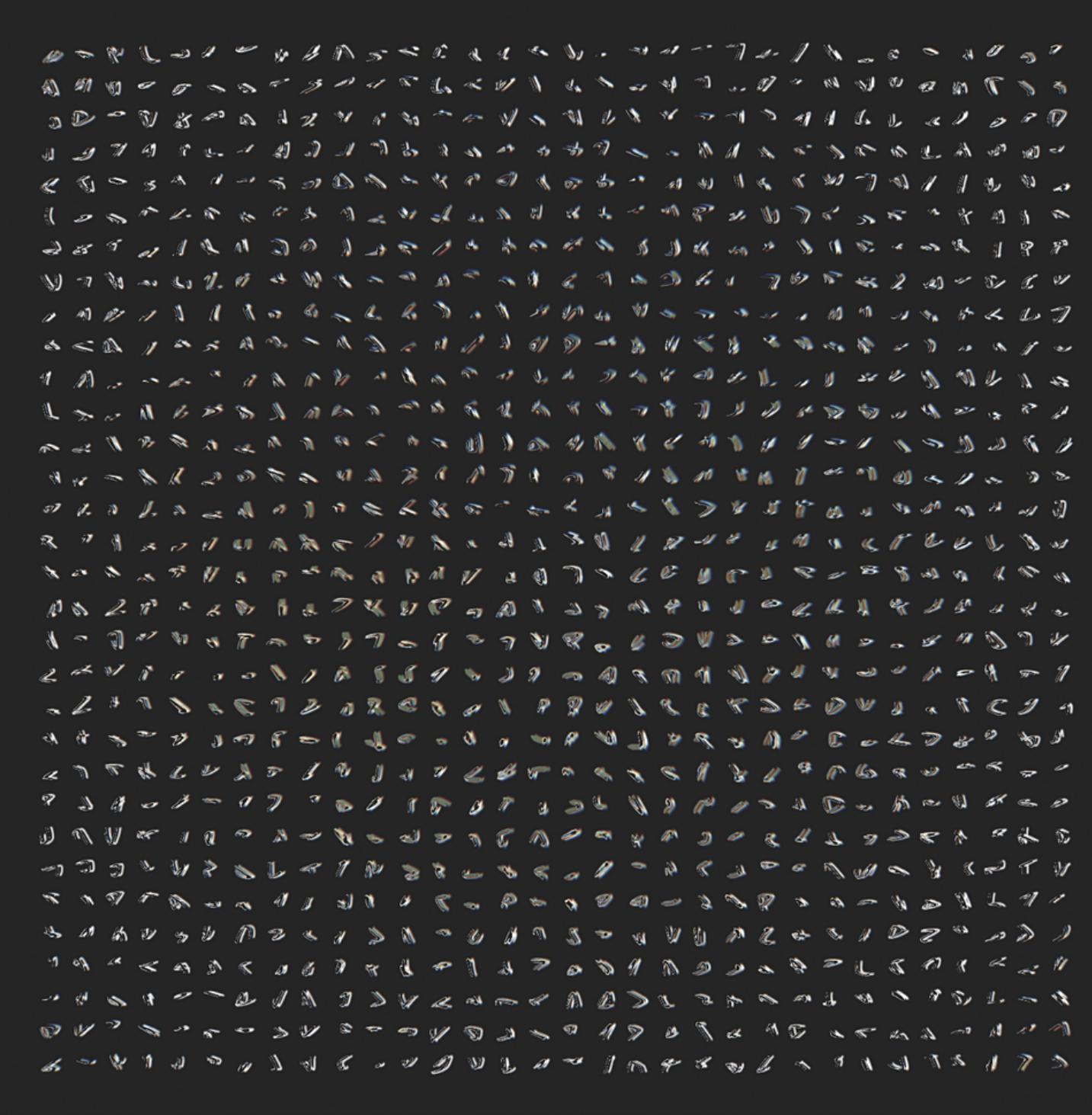
CREATING A POINT AND LINE

```
● ● ●  
1 point(x, y)  
2 # Creates a point at (x, y) from the origin.  
3 point(x, y, z)  
4 # Creates a point at (x, y, z).  
5 dist(x1, y1, x2, y2)  
6 # Calculates the distance between 2 points.  
7 line(x1, y1, x2, y2)  
8 # Create a line by joining 2 points.  
9  
10 stroke(color)  
11 # Sets the color used to draw lines  
12 # and borders around shapes.  
13 strokeWeight(x)  
14 # Sets the thickness of the stroke.
```

```
● ● ●  
1 context.set_line_width(0.02)  
2 context.move_to(x, y)  
3 context.line_to(x1, y1)  
4 context.stroke()
```

Using straight lines





CURVE VERTEX & BEZIER CURVES

Curve Vertex: It specifies the vertex coordinates for curves.

Bezier Curves: It is a versatile mathematical curve in vector graphics.

CREATING BEZIER CURVES

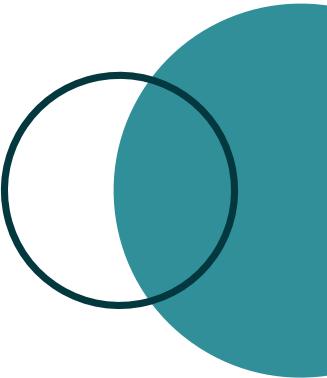
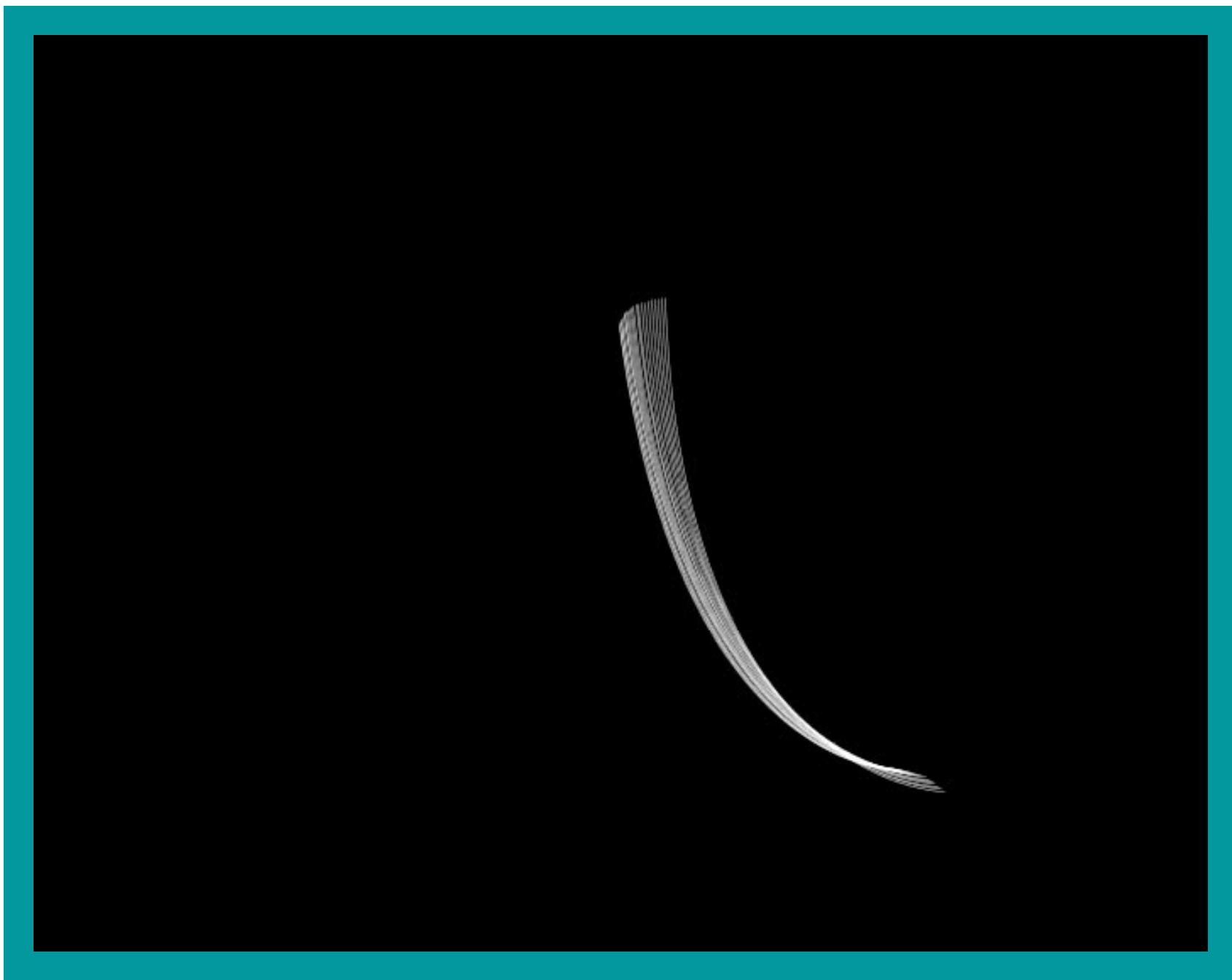


```
1 # Draws the bezier curvers on the screen.  
2 # Provide the control point of the anchor points  
3 bezier(x1, y1, x2, y2, x3, y3, x4, y4)  
4 bezier(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4)
```



```
1 # Sets the current point to a specific point A.  
2 ctx.move_to(ax, ay)  
3 # It draws a curve from the current point A to the point D,  
4 # using B and C as handles.  
5 ctx.curve_to(bx, by, cx, cy, dx, dy)
```

USING BEZIER CURVES TO CREATE WAVES



CREATING BEZIER CURVES

```
● ● ●

1 def setup():
2     global offset
3     offset = random(100)
4     size(700, 700)
5     background(0)
6     stroke(255)
7     strokeWeight(0.5)
8     noFill()
9
10
11 def draw():
12     global offset
13     x,y = [], []
14     m , n = 20, 70
15     if frameCount <= 600:
16         for i in range(4):
17             x.append(width * noise(offset + m))
18             y.append(height * noise(offset + n))
19             m += 10
20             n += 10
21     bezier(x[0], y[0], x[1], y[1], x[2], y[2], x[3], y[3]);
22     offset += 0.005;
```

USING BEZIER CURVES TO CREATE WAVES



@NEERAJP99

USING CURVEVERTEX()

```
● ● ●  
1 # An implementation of Catmull-Rom splines.  
2 # Provide x, y, z coordinates  
3 curveVertex(x, y)  
4 curveVertex(x, y, z)
```


EXAMPLE USING CURVEVERTEX()

```
● ● ●

1 def setup():
2     size(700, 700)
3     background("#013840");
4
5
6 def draw():
7     for i in range(40, width - 30, 40):
8         for j in range(40, height - 30, 40):
9             strokeWeight(1.3);
10            stroke(255);
11            noFill();
12            beginShape();
13            curveVertex(i + random(-10, 10), j + random(-10, 10));
14            endShape();
15    noLoop();
```



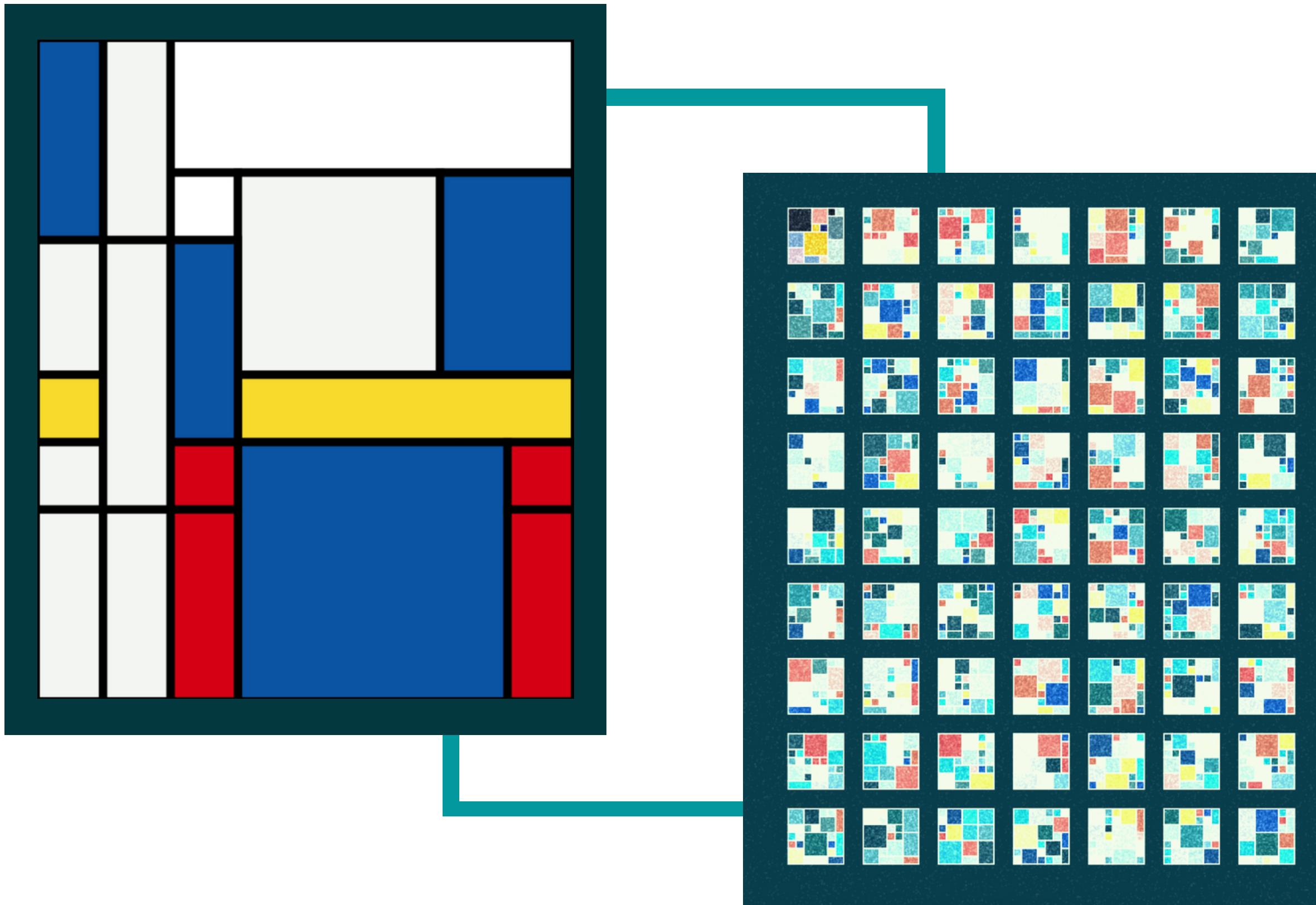
```
1 def setup():
2     size(700, 700)
3     background("#013840");
4
5
6 def draw():
7     for i in range(40, width - 30, 40):
8         for j in range(40, height - 30, 40):
9             strokeWeight(1.3);
10            stroke(255);
11            noFill();
12            beginShape();
13            curveVertex(i + random(-10, 10), j + random(-10, 10));
14            curveVertex(i + random(-10, 10), j + random(-10, 10));
15            curveVertex(i + random(-10, 10), j + random(-10, 10));
16            curveVertex(i + random(-10, 10), j + random(-10, 10));
17            curveVertex(i + random(-10, 10), j + random(-10, 10));
18            curveVertex(i + random(-10, 10), j + random(-10, 10));
19            endShape();
20    noLoop();
```


CREATING BASIC SHAPES

```
● ● ●  
1 ellipse(a, b, c, d)  
2 # Creates an ellipse at point a,b  
3 # with "c" width and "d" height  
4 rect(a, b, c, d, tl, tr, br, bl))  
5 # Creates a rectangle at point a,b  
6 # width width "c" and height "d"  
7 square(x, y, c)  
8 # Creates a square
```

```
● ● ●  
1 context.rectangle(a, b, c, d)  
2 # Creates a rectangle  
3 context.arc(x, y, radius, start_angle, stop_angle)  
4 # Creates an arc, with angles in radians  
5 # with stop angle as math.pi*2 for ellipse
```

Piet Mondrian Experiments



@NEERAJP99

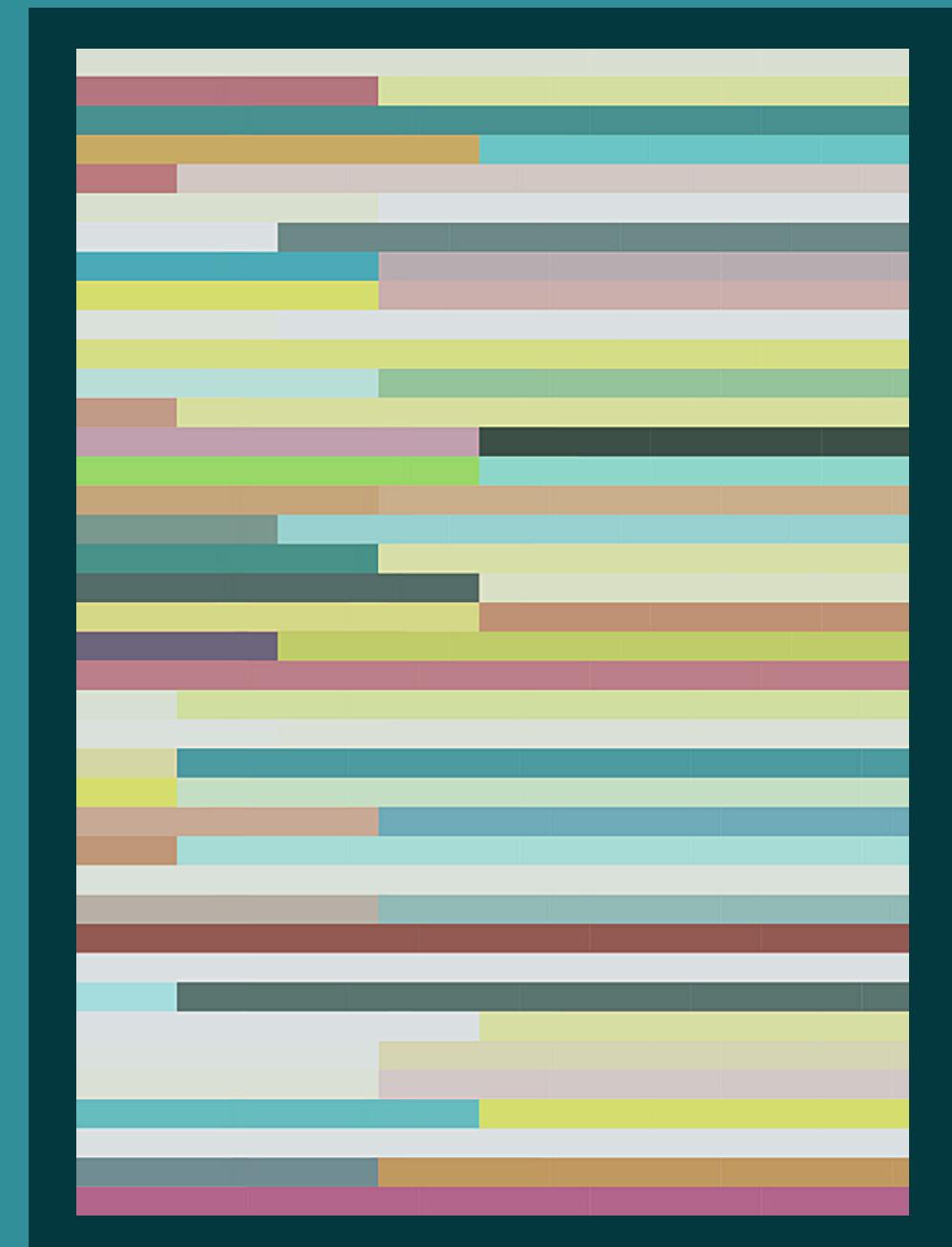
USING RECTANGLE SHAPE





```
1 def setup():
2     size(500, 700)
3     background(0)
4
5 def draw():
6     for y in range(0, 700, 10):
7         x1 = random(0, 500)
8         x2 = 500 - x1
9         noStroke()
10        fill(random(255))
11        rect(0, y, int(x1), 15)
12        fill(random(255))
13        rect(int(x1), y, 500, 15)
14    noLoop()
```

Using Shapes



LINEAR INTERPOLATION

This function interpolates within the range [start..end] based on the amount parameter, where amount parameter is typically within a [0..1] range.

```
● ● ●  
1 lerp(start, stop, amt)  
2 # Calculates a number between two numbers  
3 # at a specific increment.
```



EXAMPLE USING LERPCOLOR()

```
● ● ●

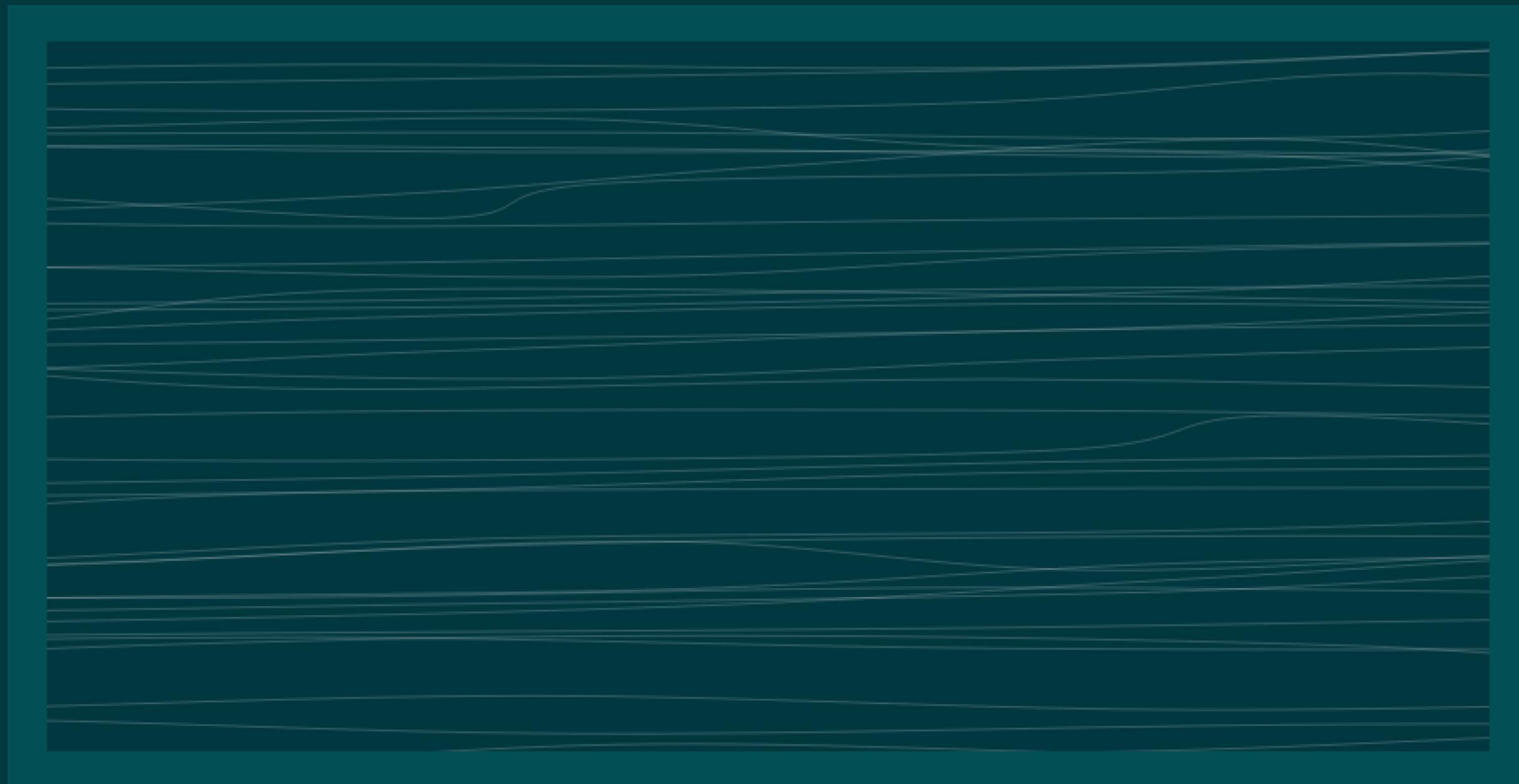
1 def setup():
2     size(600, 600)
3     background(255)
4     colorMode(HSB, 360, 100, 100);
5
6 def draw():
7     initialColor = color(0,50,100);
8     finalColor = color(45, 80, 100);
9     noStroke()
10    for i in range(width):
11        linearIntValue = map(i, 0, width, 0, 1.0)
12        # Calculates a color between two colors at a specific increment.
13        linearintColor = lerpColor(initialColor, finalColor, linearIntValue)
14        fill(linearintColor)
15        rect(i, 0, 25, height)
16    noLoop()
```

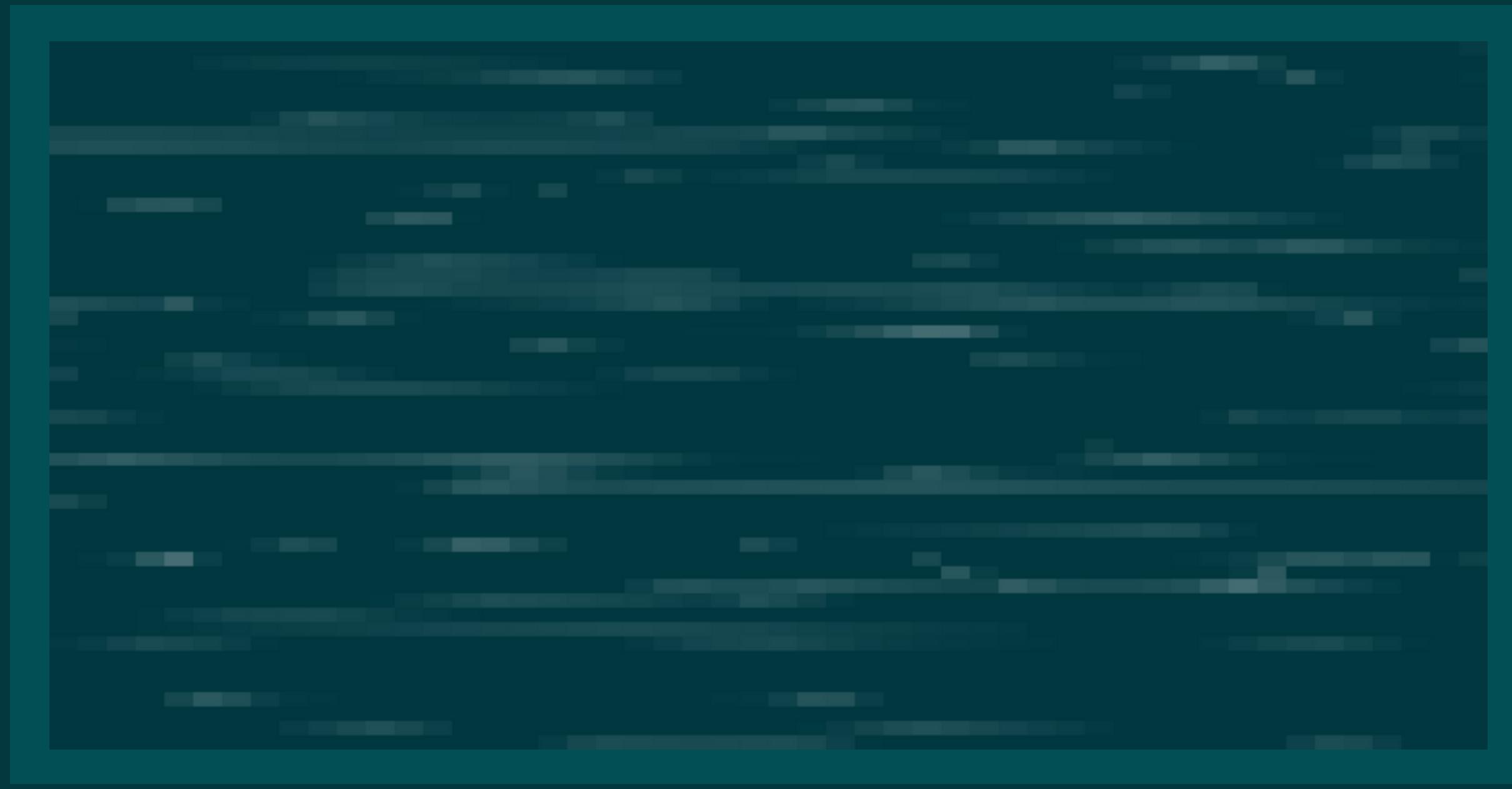




[CHROMA, NEERAJ PANDEY]

PERLIN NOISE / SIMPLEX NOISE





@NEERAJP99

PERLIN NOISE / SIMPLEX NOISE

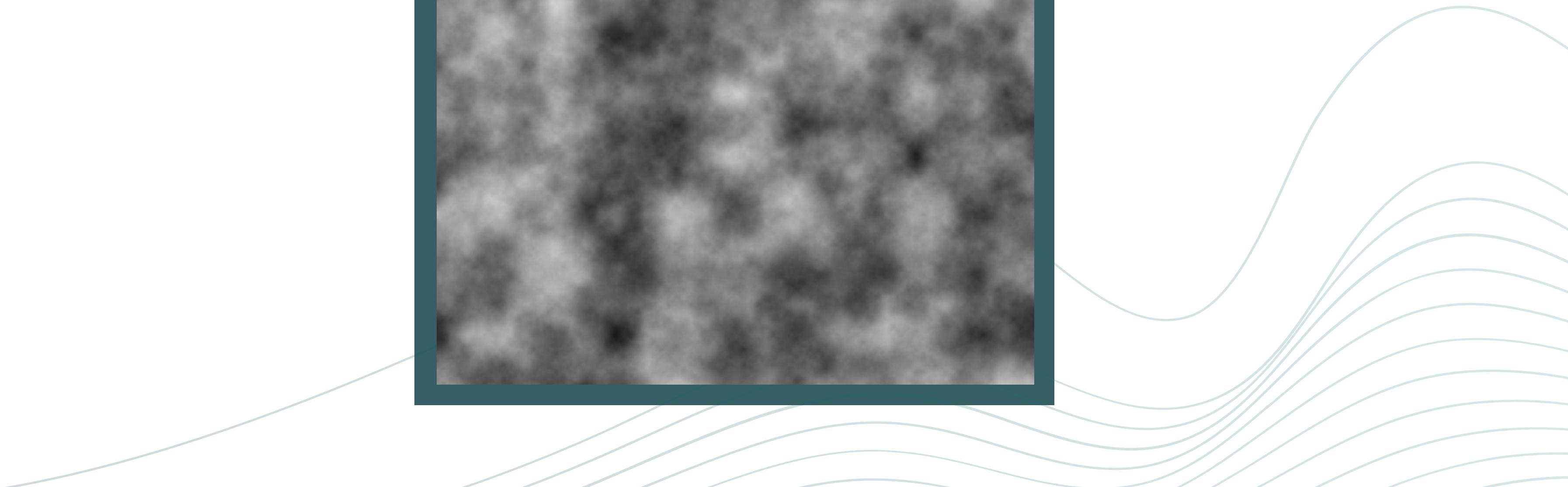
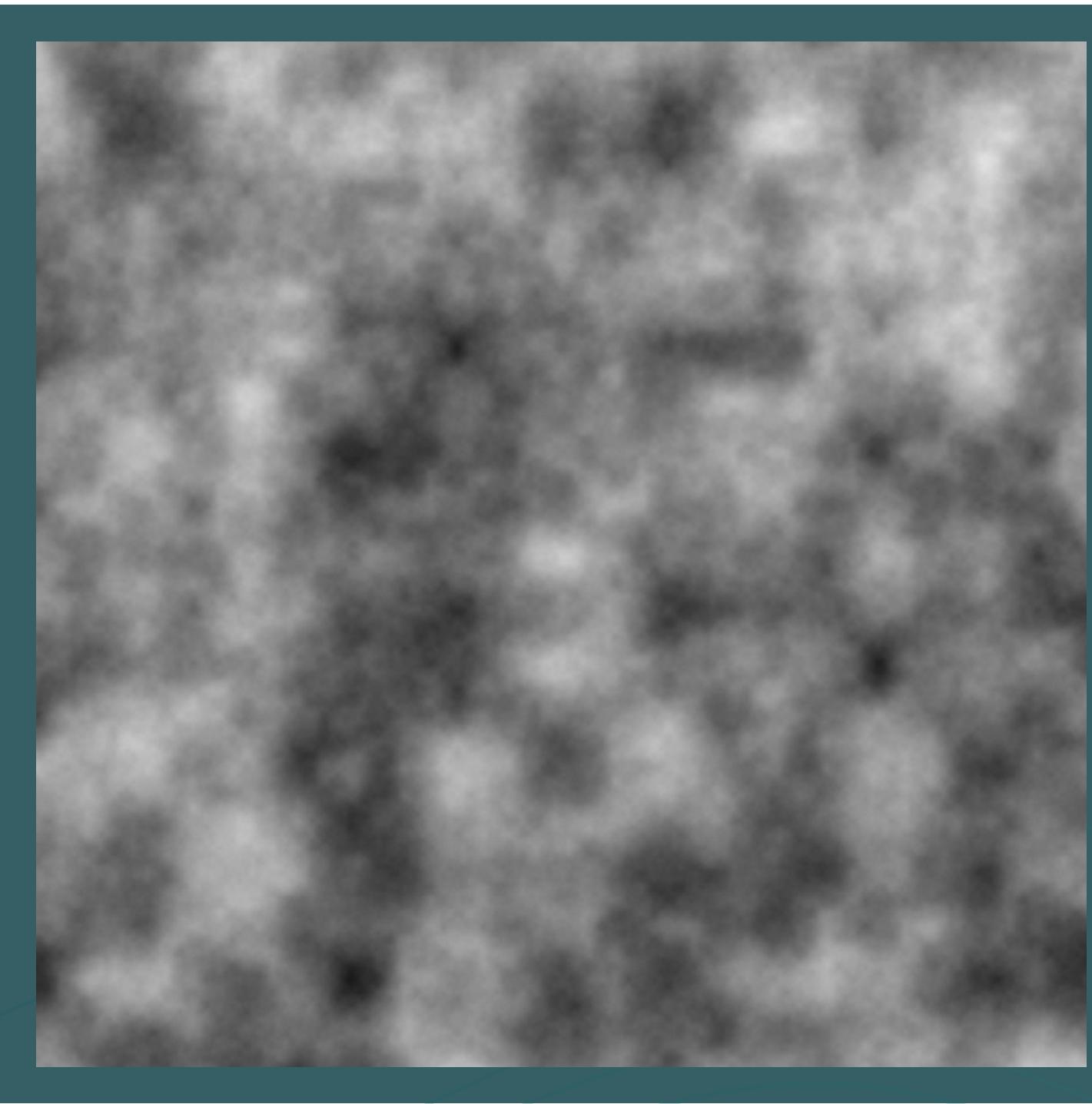


```
1 noise(x, [y], [z])
2 # Returns the Perlin noise value
3 noiseSeed(seed)
4 # Sets the sees value for noise()
5 noiseDetail(lod, falloff)
6 # Adjusts the level of details produced
7 # by perlin noise
```

Without processing, use noise module in Python.

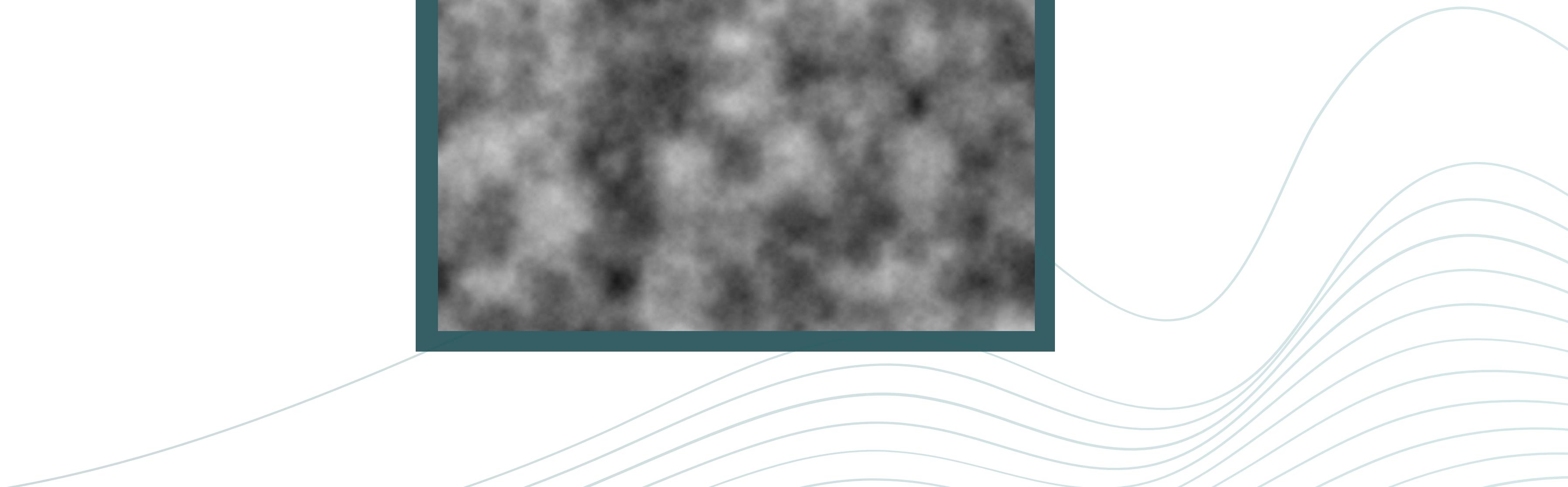
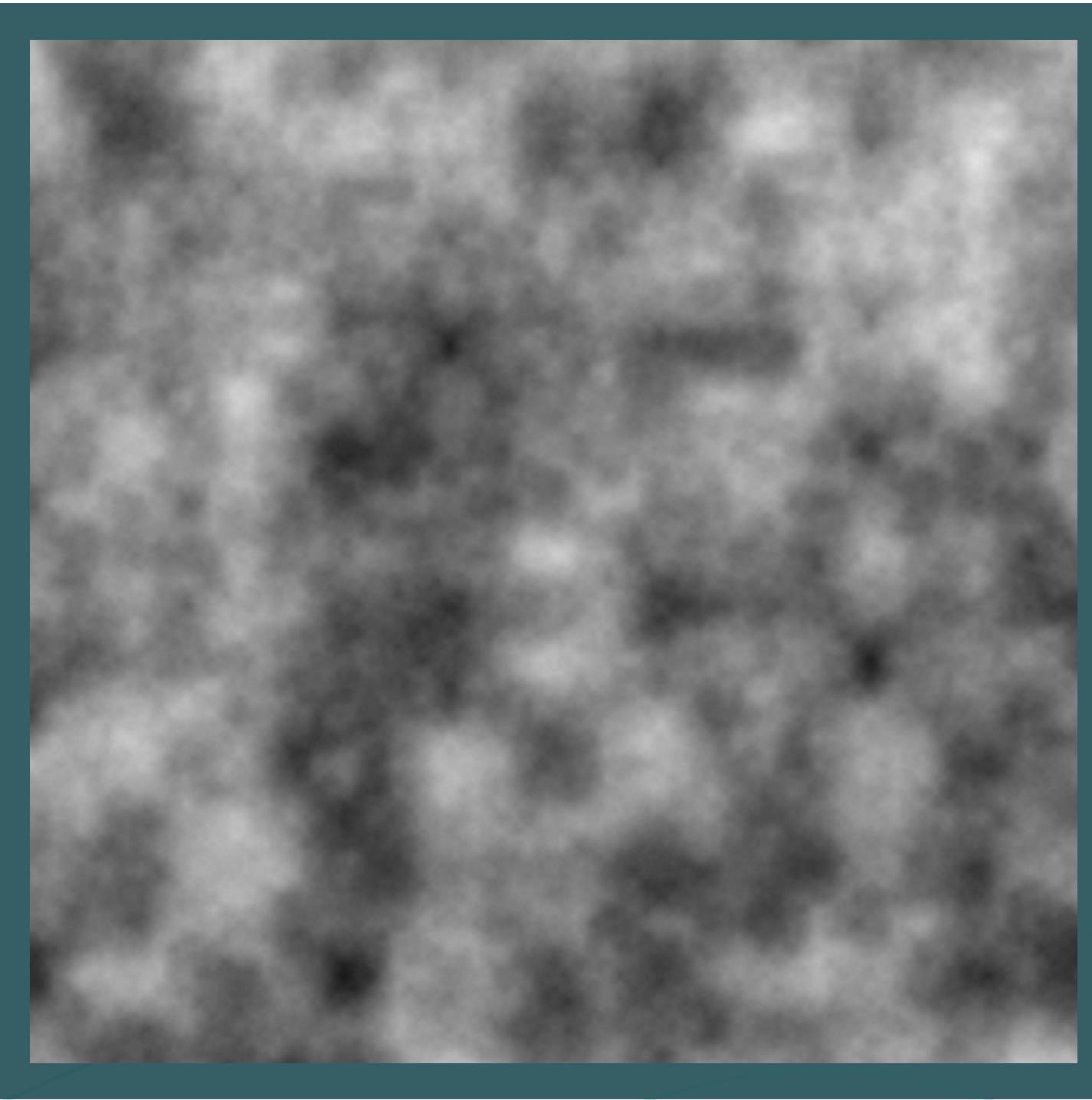
(<https://pypi.org/project/noise/>)

PERLIN NOISE / SIMPLEX NOISE

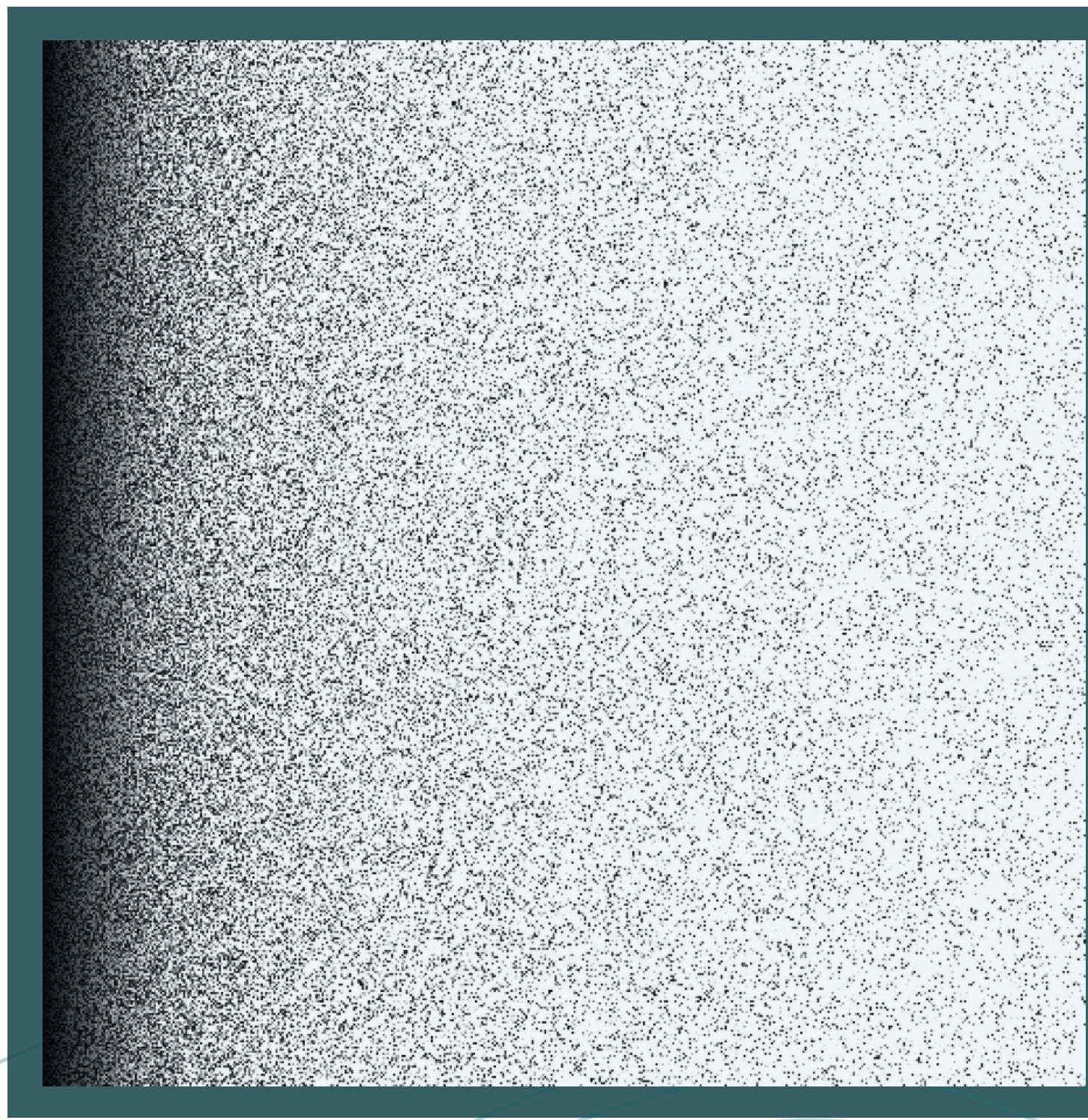


ADD NOISE TO THE PIXEL COLORS

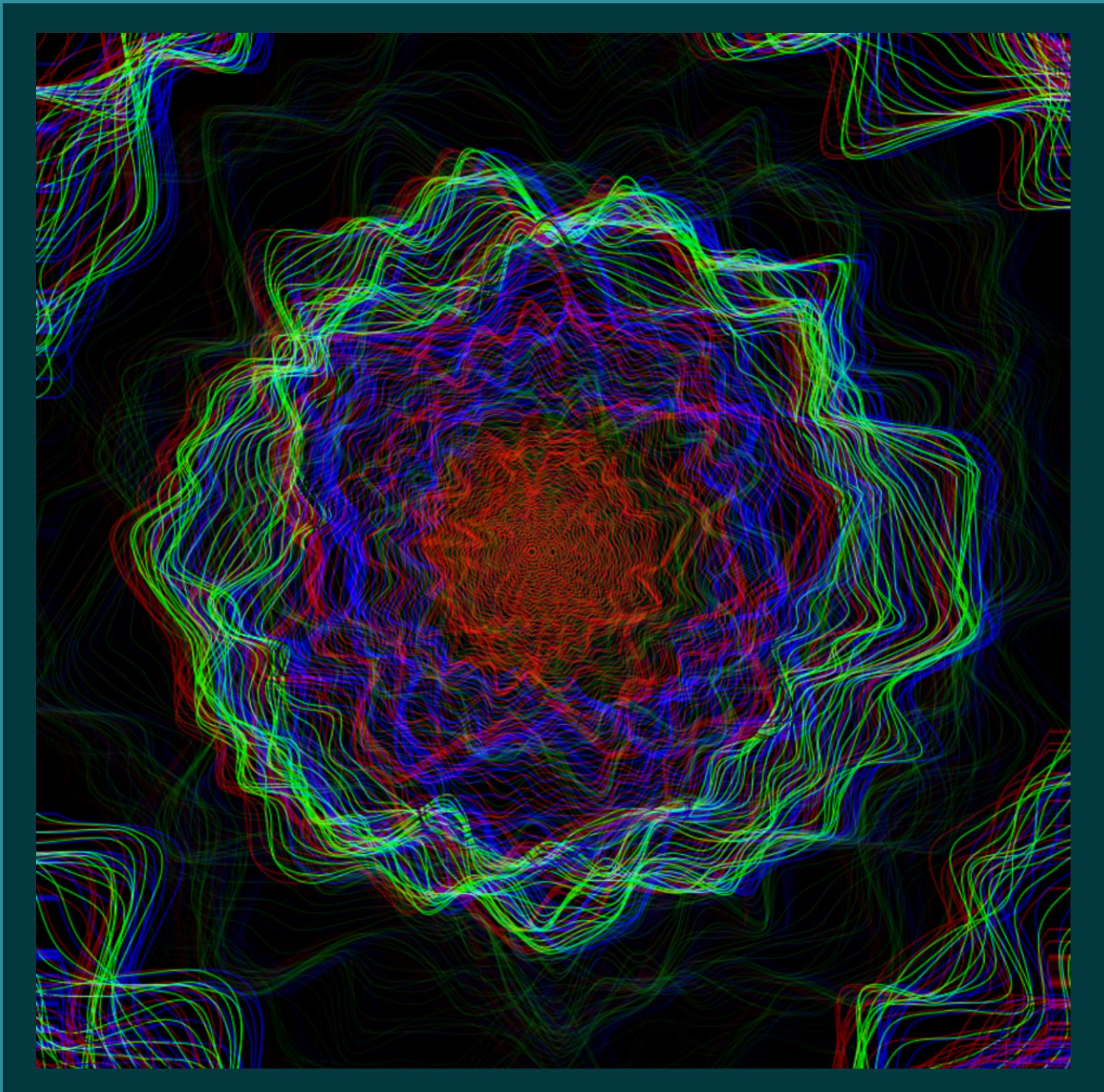
```
1 def setup():
2     size(500, 500)
3     pixelDensity(1)
4     noiseDetail(40)
5
6 def draw():
7     xoff = 0
8     loadPixels()
9     for x in range(0, width, 1):
10         yoff = 0
11         for y in range(0, height, 1):
12             index = (x + y * width)
13             r = noise(xoff, yoff) * 255
14             if random(1) > 0:
15                 pixels[index] = color(r)
16             yoff += 0.02
17         xoff += 0.02
18     updatePixels()
19     noLoop()
```



ADD RANDOM TO THE PIXEL COLORS



USING NOISE AND TRIGONOMETRY



USING NOISE AND TRIGONOMETRY

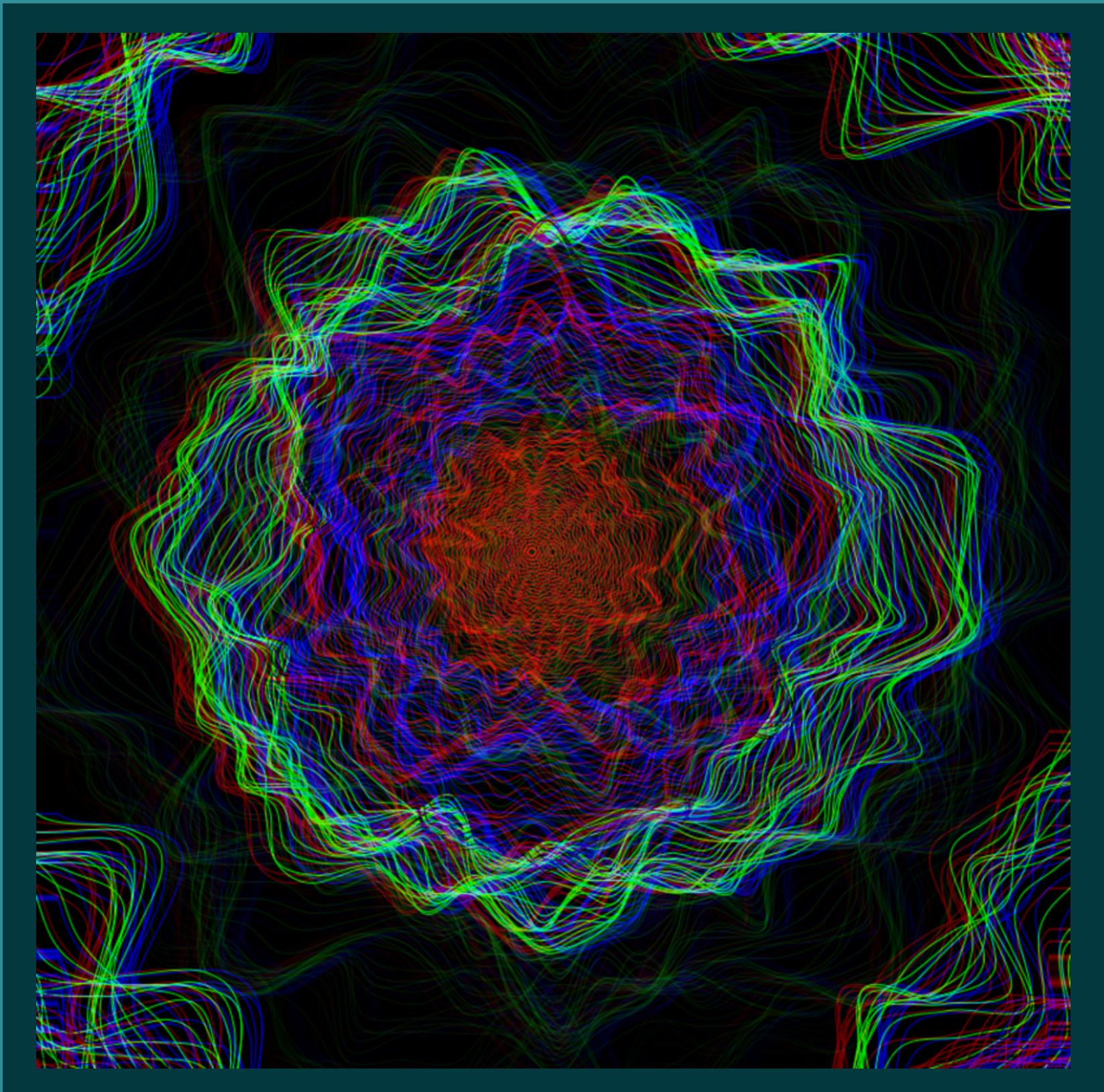
```
1 def draw():
2     global rad, r, g, b, translateX, translateY, opacity, zoff
3     noFill()
4     stroke(r, g, b, opacity)
5     beginShape()
6     a = 0
7     while a < TWO_PI:
8         xoff = map(cos(a), -1, 1, 0, 10)
9         yoff = map(sin(a), -1, 1, 0, 10)
10        noiseFactor = map(noise(xoff, yoff, zoff), 0, 1, 100, 150)
11        x = width / 2 + rad * noiseFactor * cos(a)
12        y = height / 2 + rad * noiseFactor * sin(a)
13        curveVertex(x, y)
14        a += 0.1
15    endShape(CLOSE)
16    zoff += 0.1
17    rad -= 0.02
```

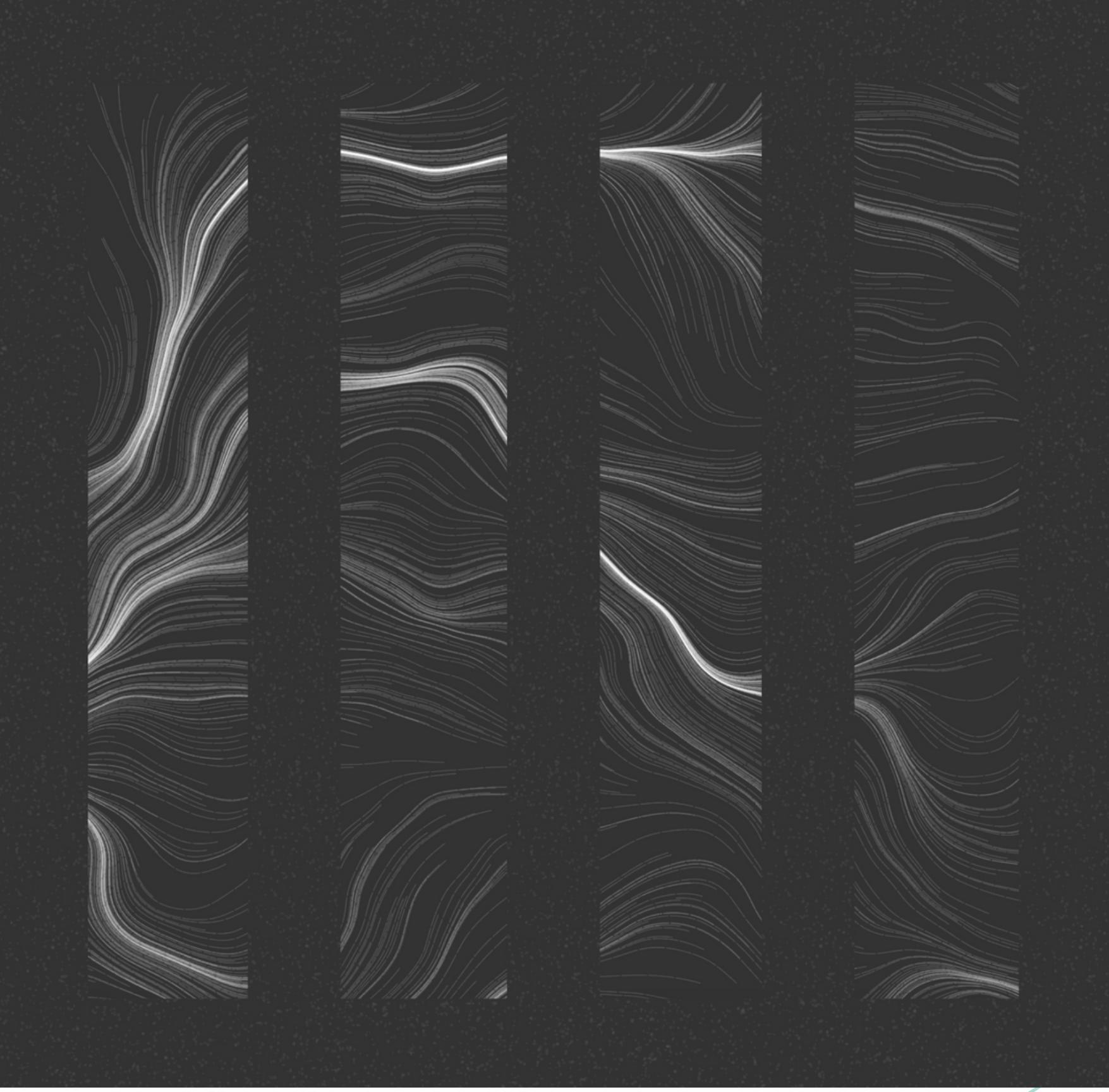
USING NOISE AND TRIGNOMETRY

```
● ● ●

1  if r > 255:
2      r = 0
3  if g > 255:
4      g = 0
5  if b > 255:
6      b = 0
7  if opacity == 0:
8      opacity = 255
9  opacity -= 1
10 r += 1
11 g += 1
12 b += 1
```

USING NOISE AND TRIGONOMETRY

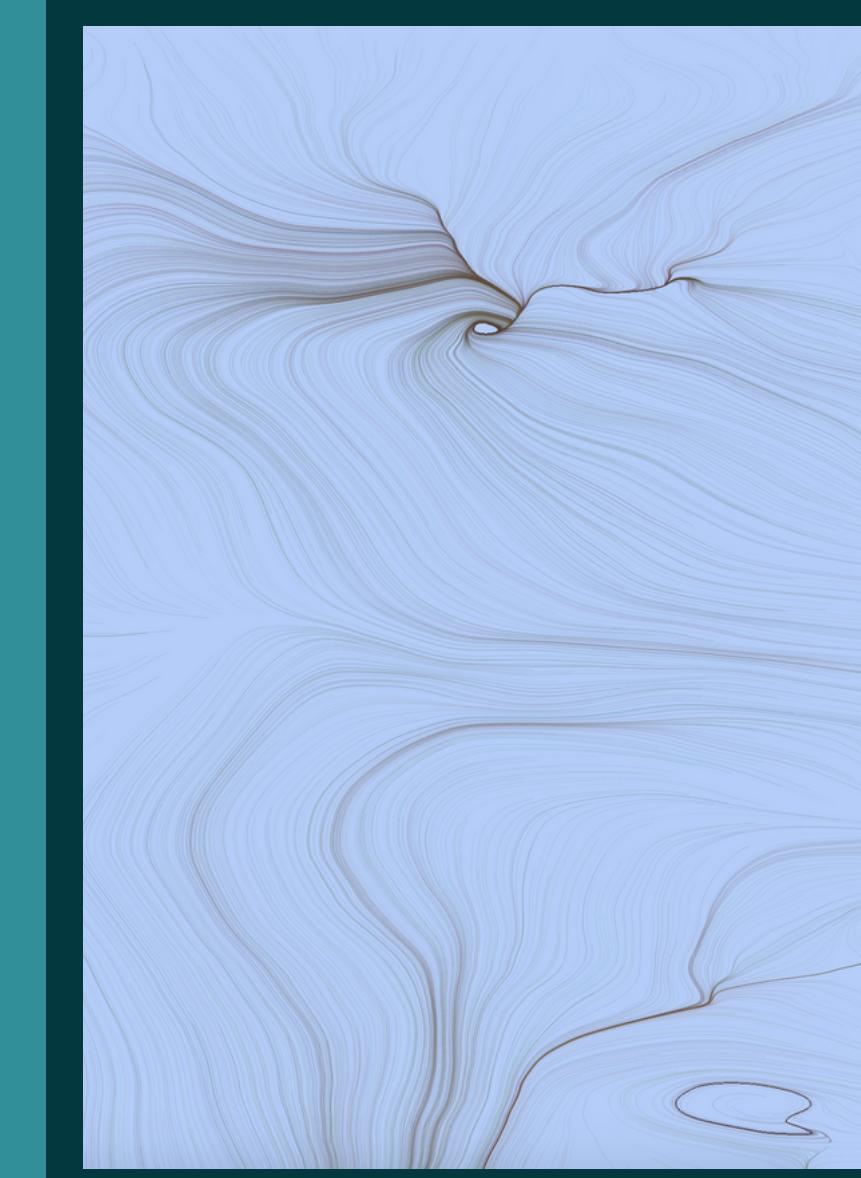
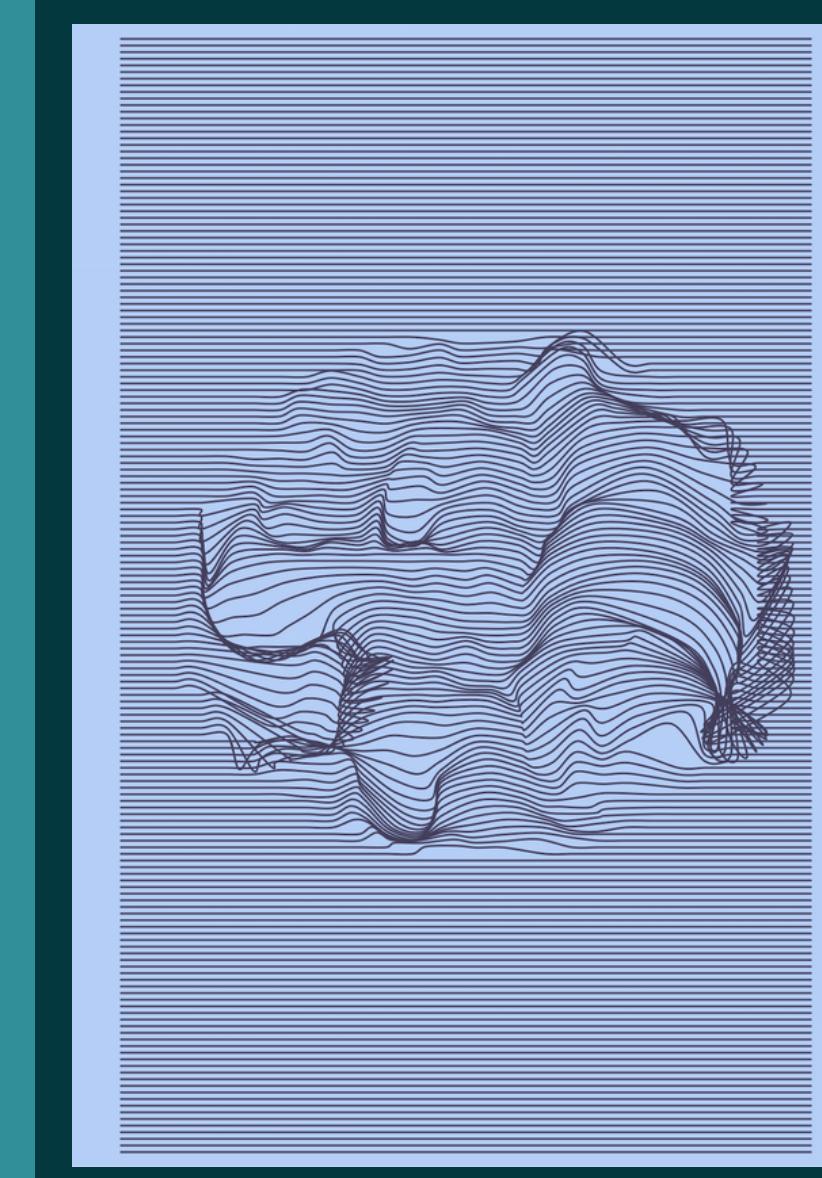




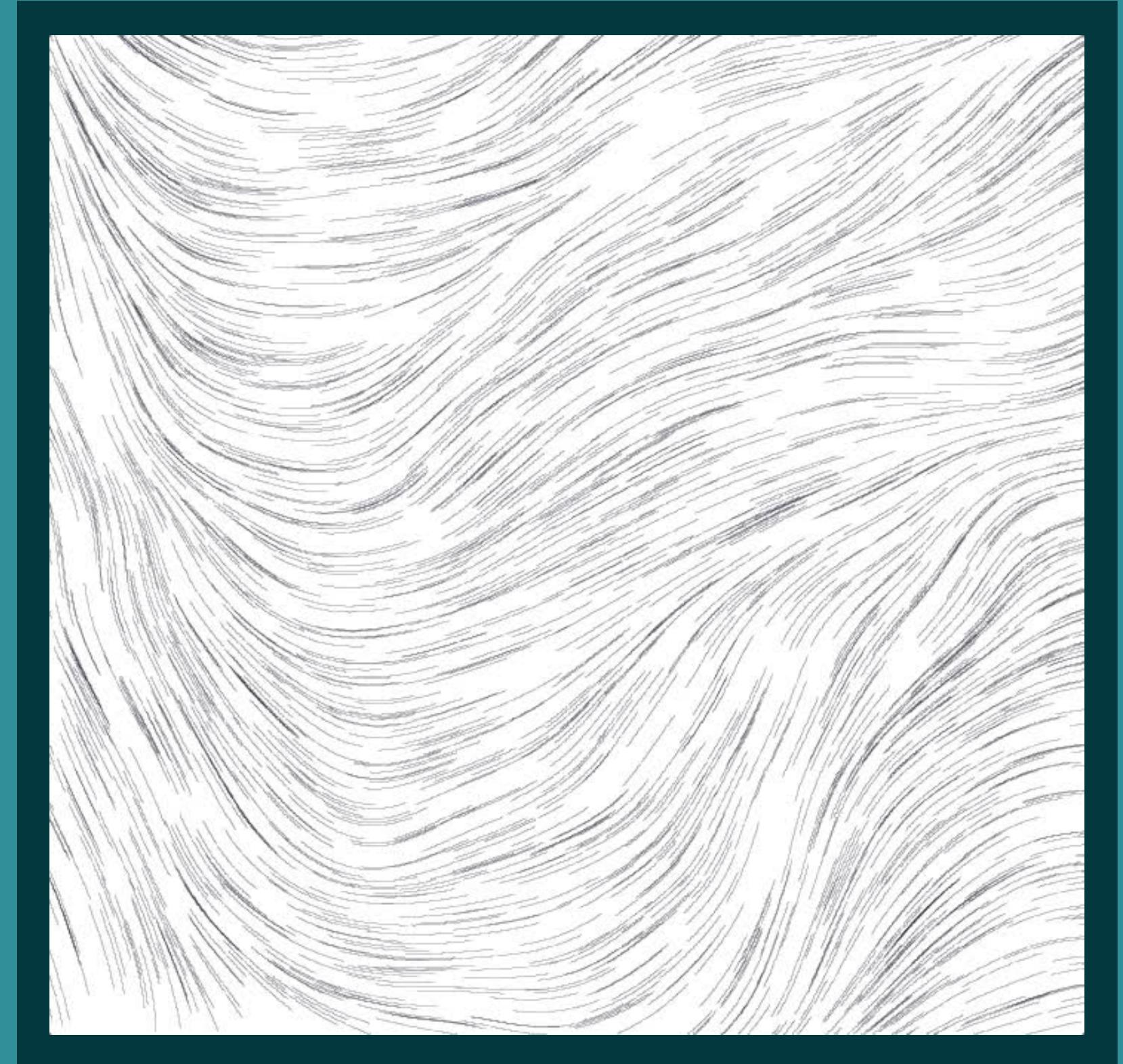
PERLIN NOISE FIELDS

Using Perlin Noise Field and Perlin Noise generated random noise points(grain like texture).

Perlin Noise Field



PERLIN NOISE FIELD



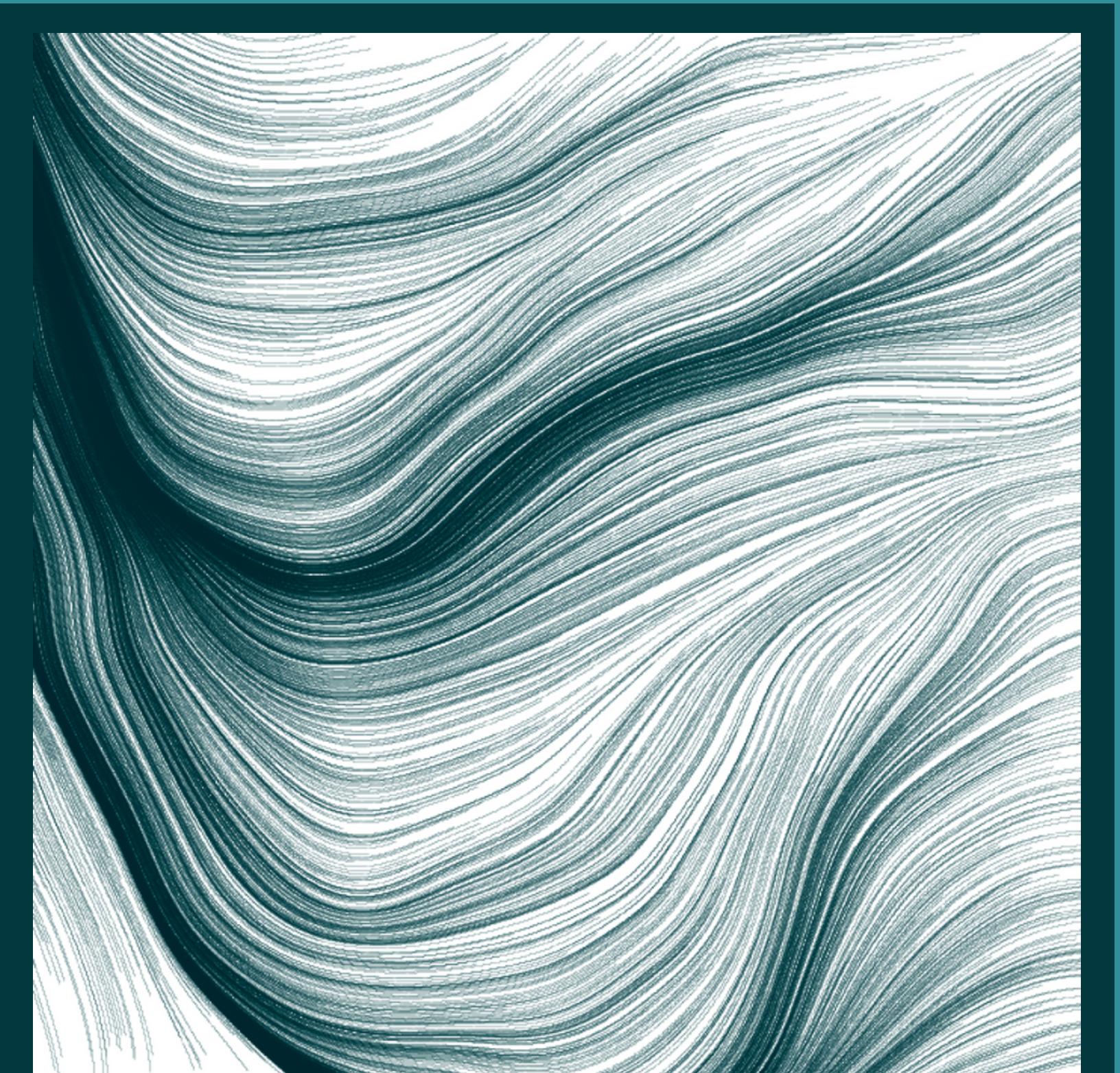
PERLIN NOISE FIELD

```
1 def setup():
2     global points
3     size(700, 700)
4     background(255)
5     points = []
6     # Loop over and create random x, y vectors
7     for i in range(2000):
8         newVector = PVector(random(width + 100), random(height))
9         # Push the newly created vector points to the points array
10        points.append(newVector)
```

PERLIN NOISE FIELD

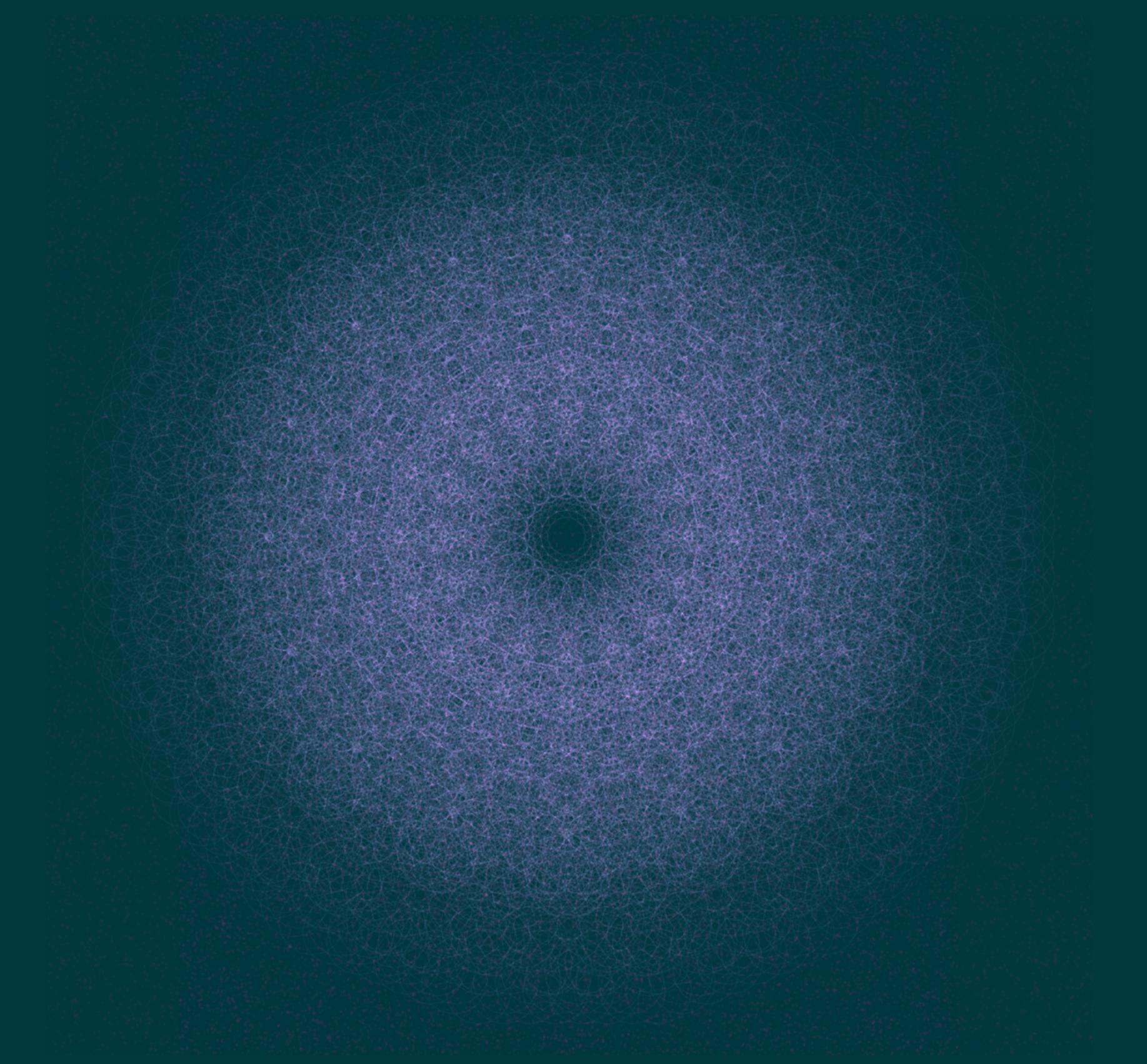
```
● ● ●  
1 def draw():  
2     global points  
3     for i in range(len(points)):  
4         noFill()  
5         noiseSeed(2)  
6         vectorObject = points[i]  
7         stroke("#333");  
8         strokeWeight(0.3);  
9         beginShape()  
10        for i in range(20):  
11            # Generate noise values and map it between 0 and 2*PI radians  
12            noiseValue = map(  
13                noise(vectorObject.x / 500, vectorObject.y / 500),  
14                0,  
15                1,  
16                0,  
17                2*PI  
18            )  
19            x1 = vectorObject.x  
20            y1 = vectorObject.y  
21            x2 = x1 + cos(noiseValue)  
22            y2 = y1 + sin(noiseValue)  
23            vertex(x1, y1)  
24            vectorObject.x = x2  
25            vectorObject.y = y2  
26        endShape(OPEN)
```

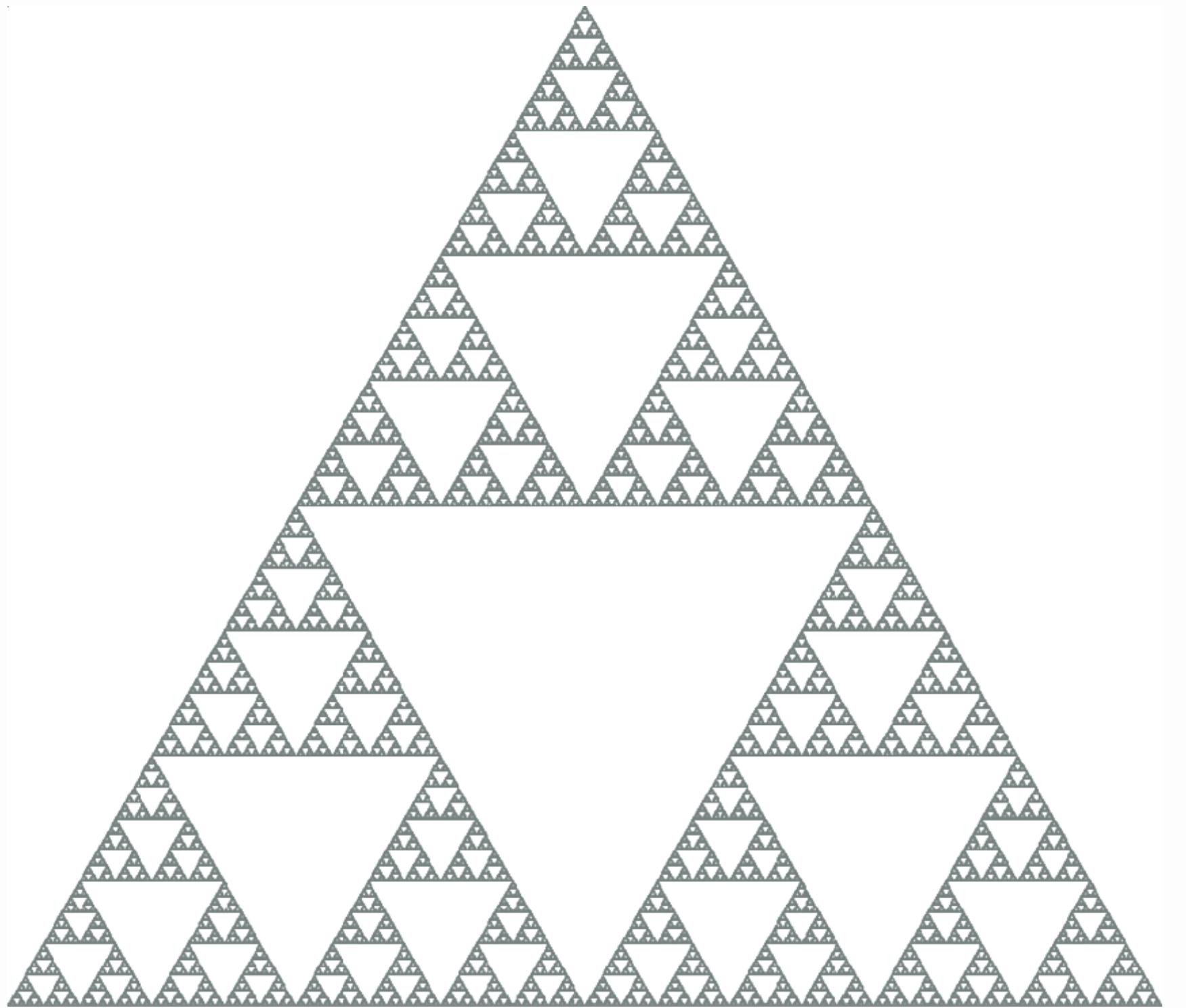
PERLIN NOISE FIELD



GEOMETRY FRACTALS CHAOS

Using Geometrical patterns, fractals and chaos theory to generate aesthetic art pieces

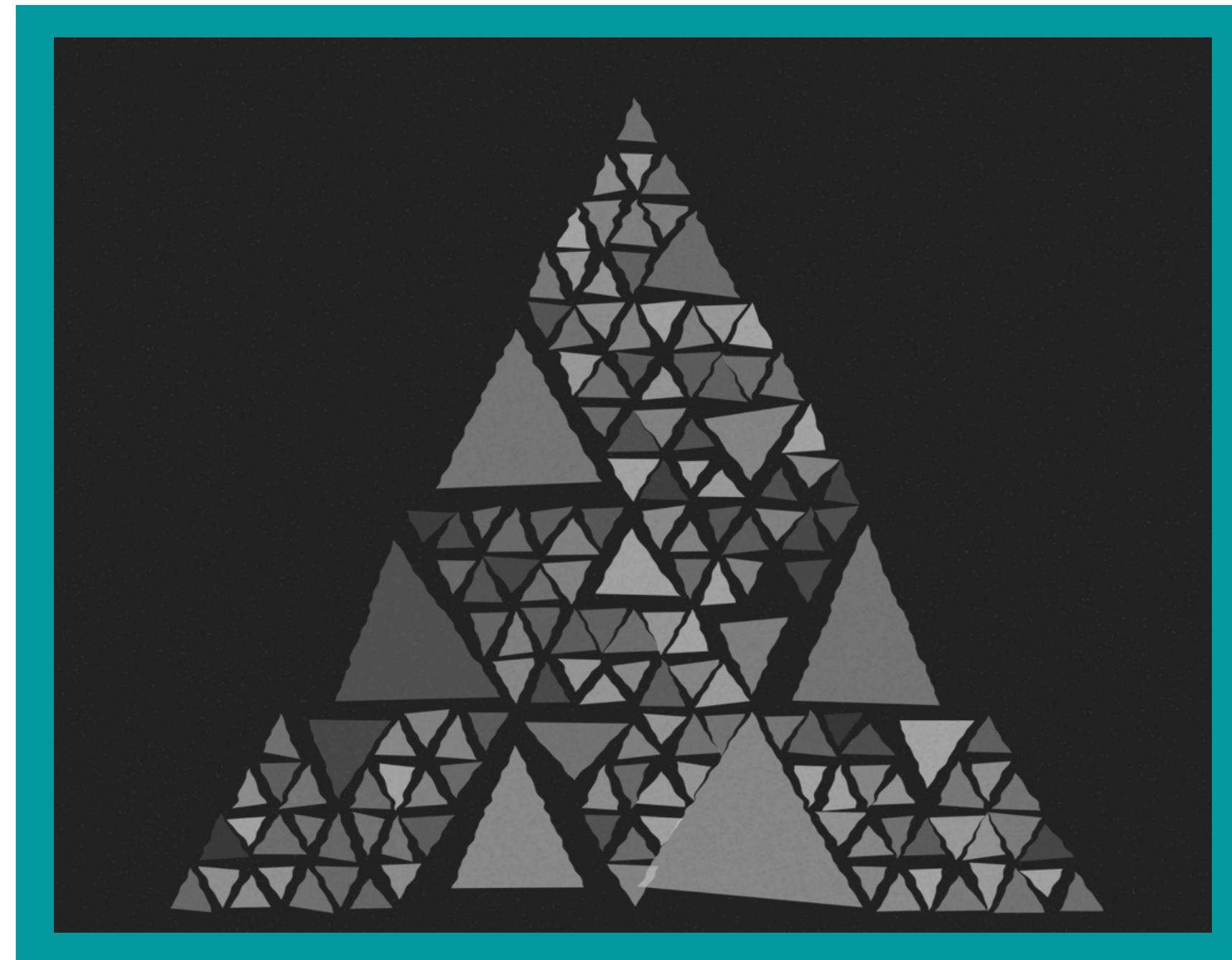




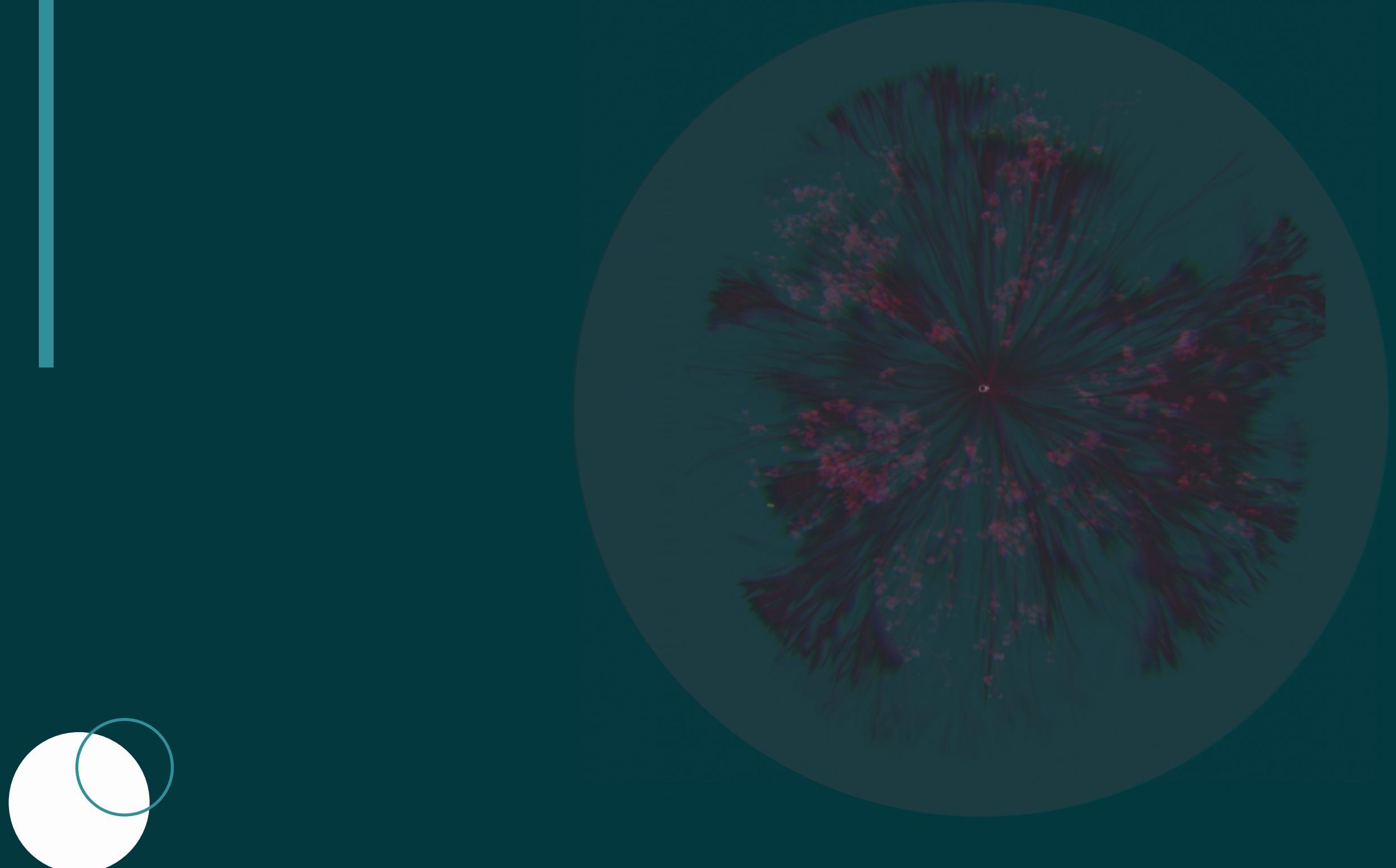
SIERPINSKI TRIANGLE

An equilateral triangle, subdivided recursively into smaller equilateral triangles with one recursive call each time.

Modified Sierpinski Triangle

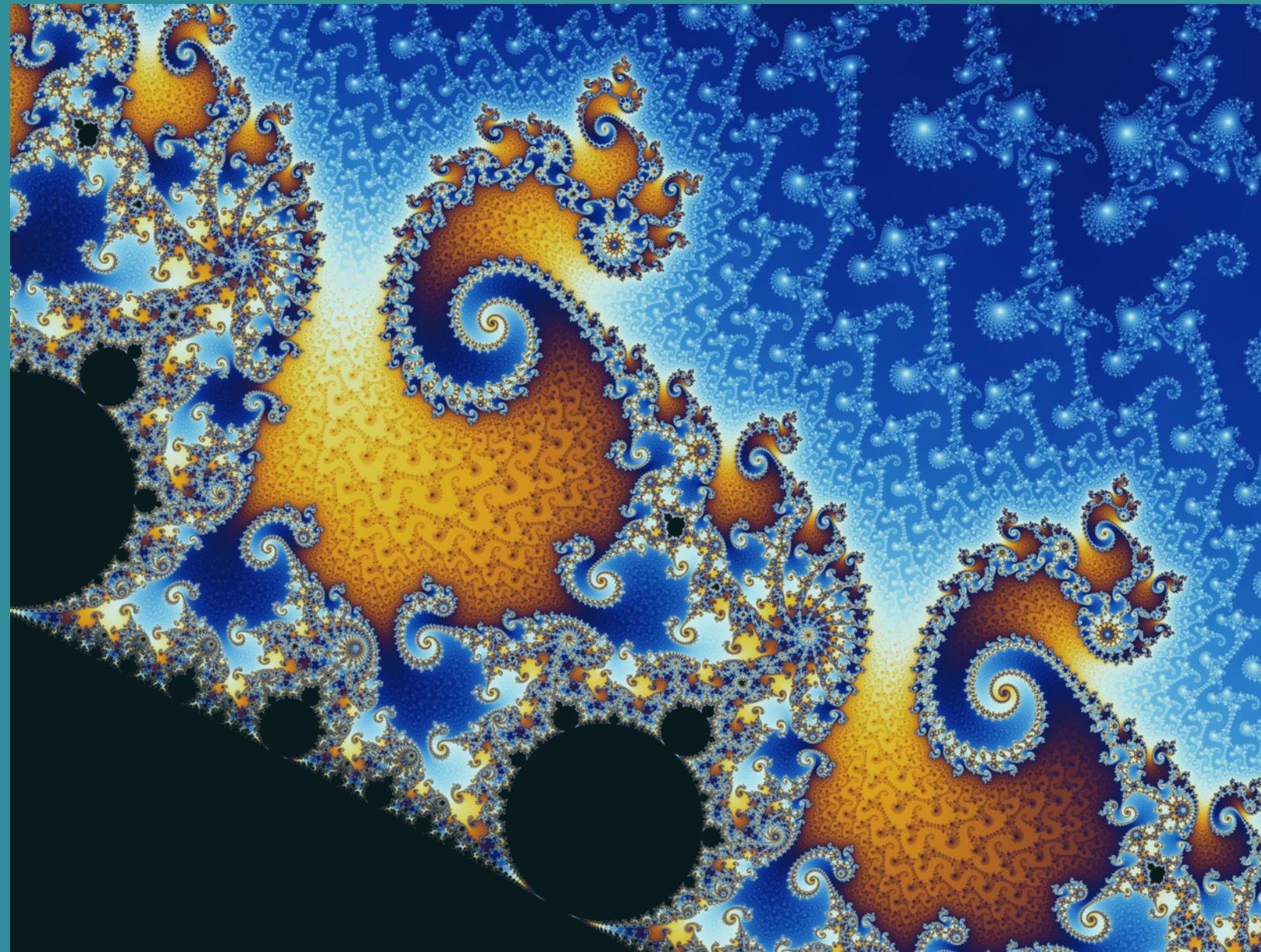


Fractal Flower



@NEERAJP99

Mandelbrot Set



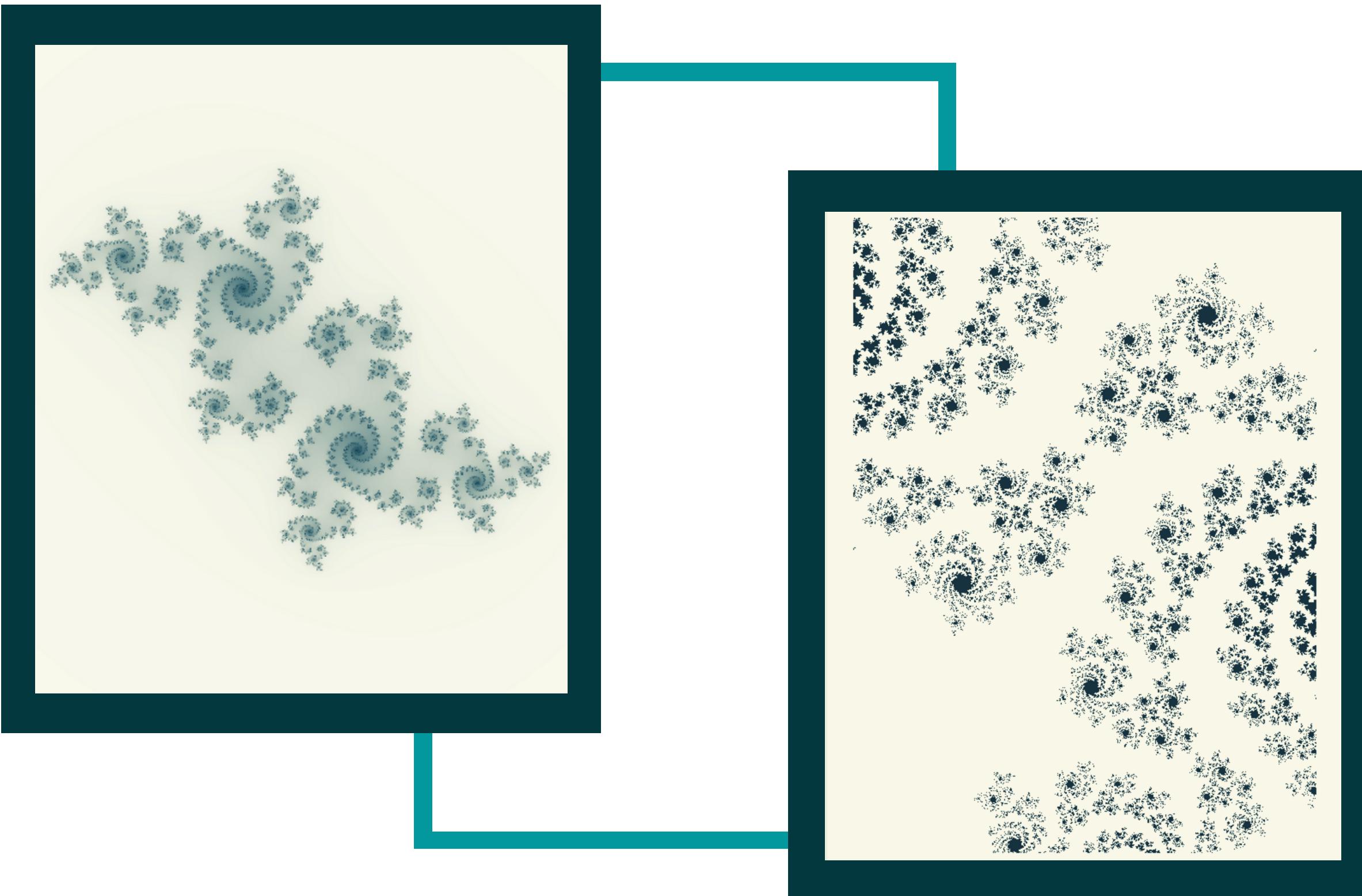
A geometrical figure where each part has the same statistical characters.

$$z = x + yi$$

$$z_{n+1} = z_n^2 + C$$

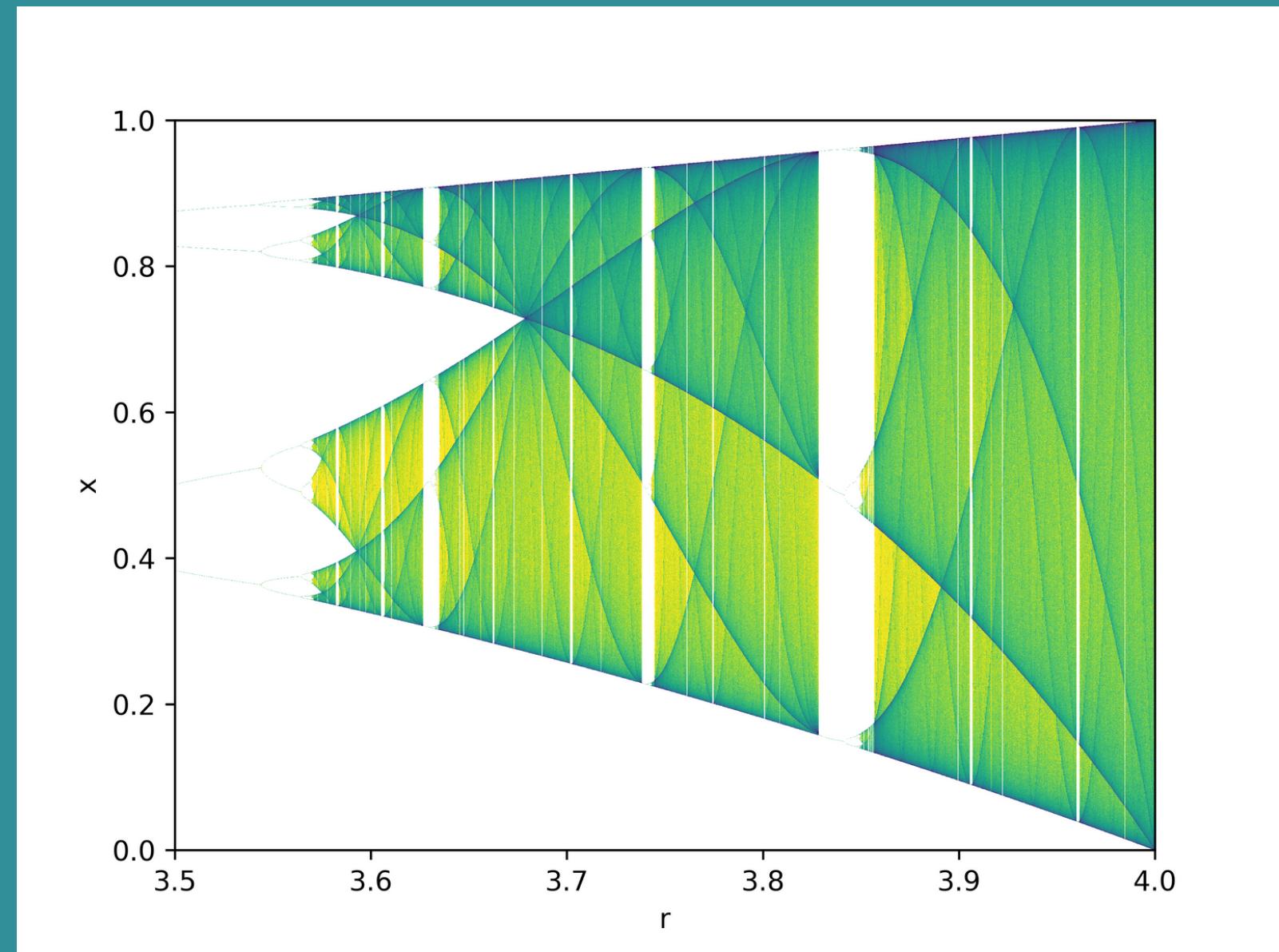


Julia Set



@NEERAJP99

The Logistic Map

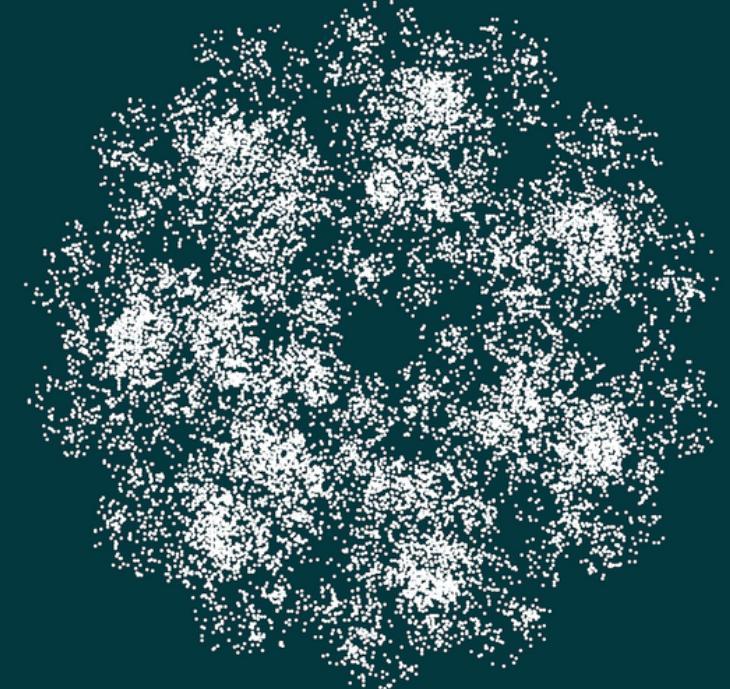
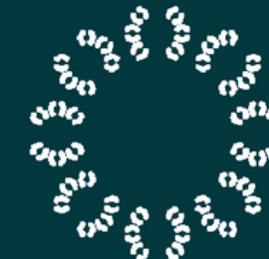
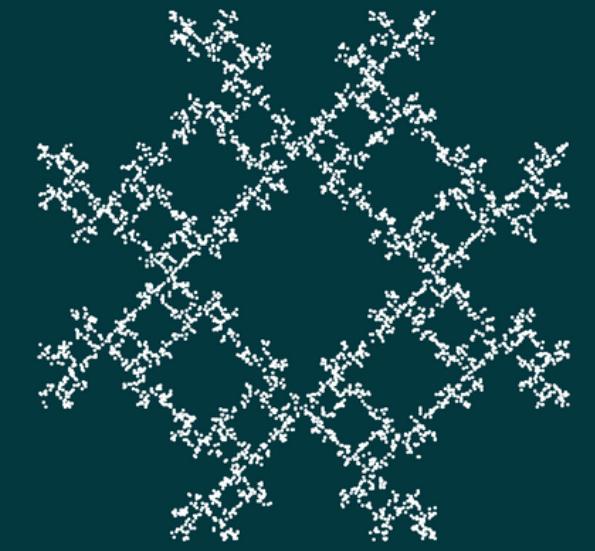
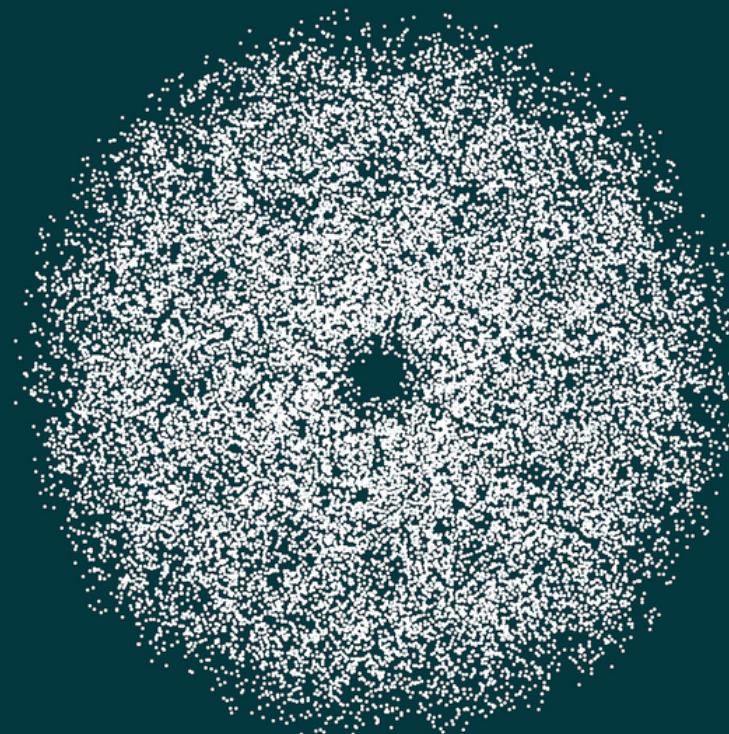


Source: Wikimedia

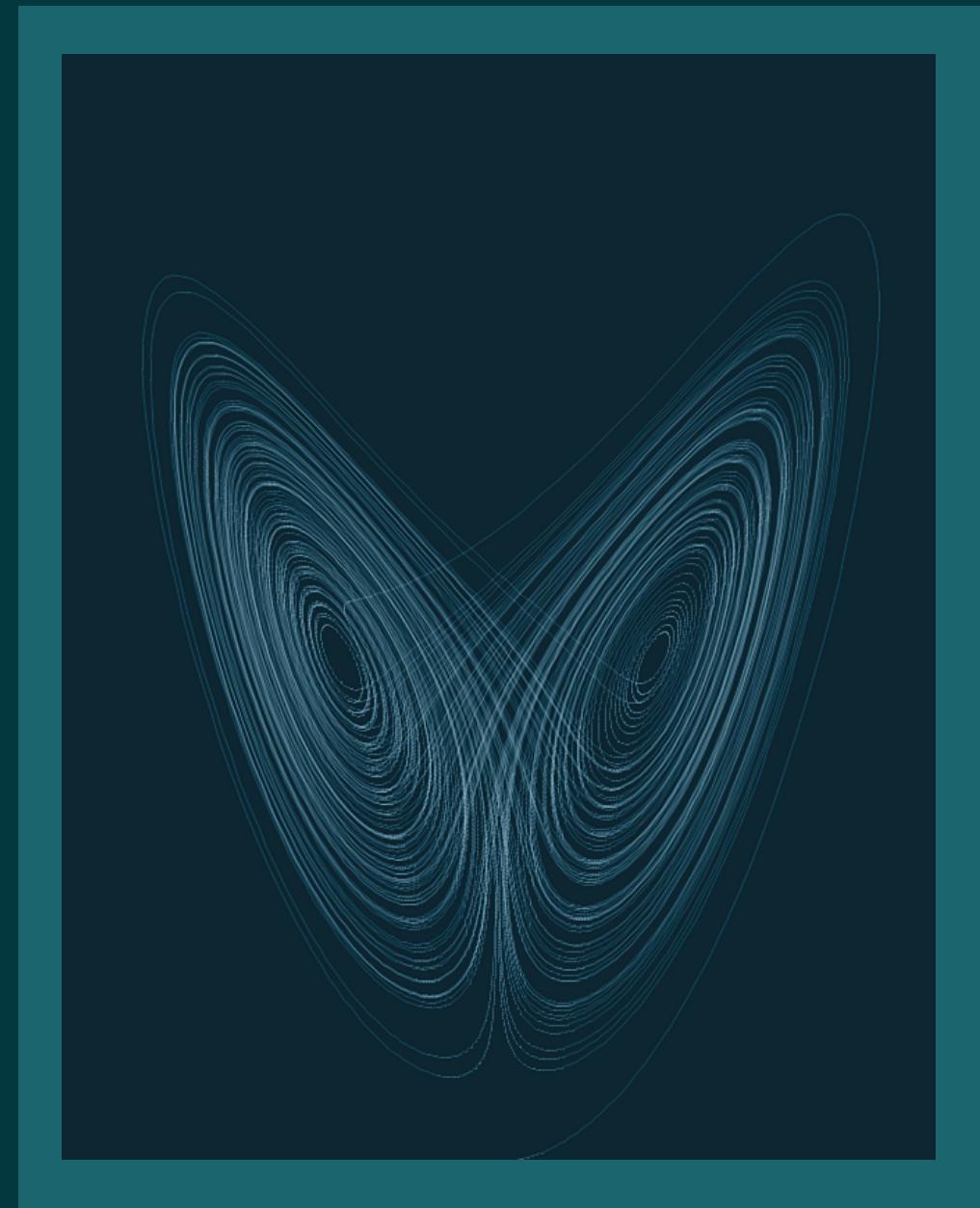
CHAOS THEORY

Deterministic, unpredictable

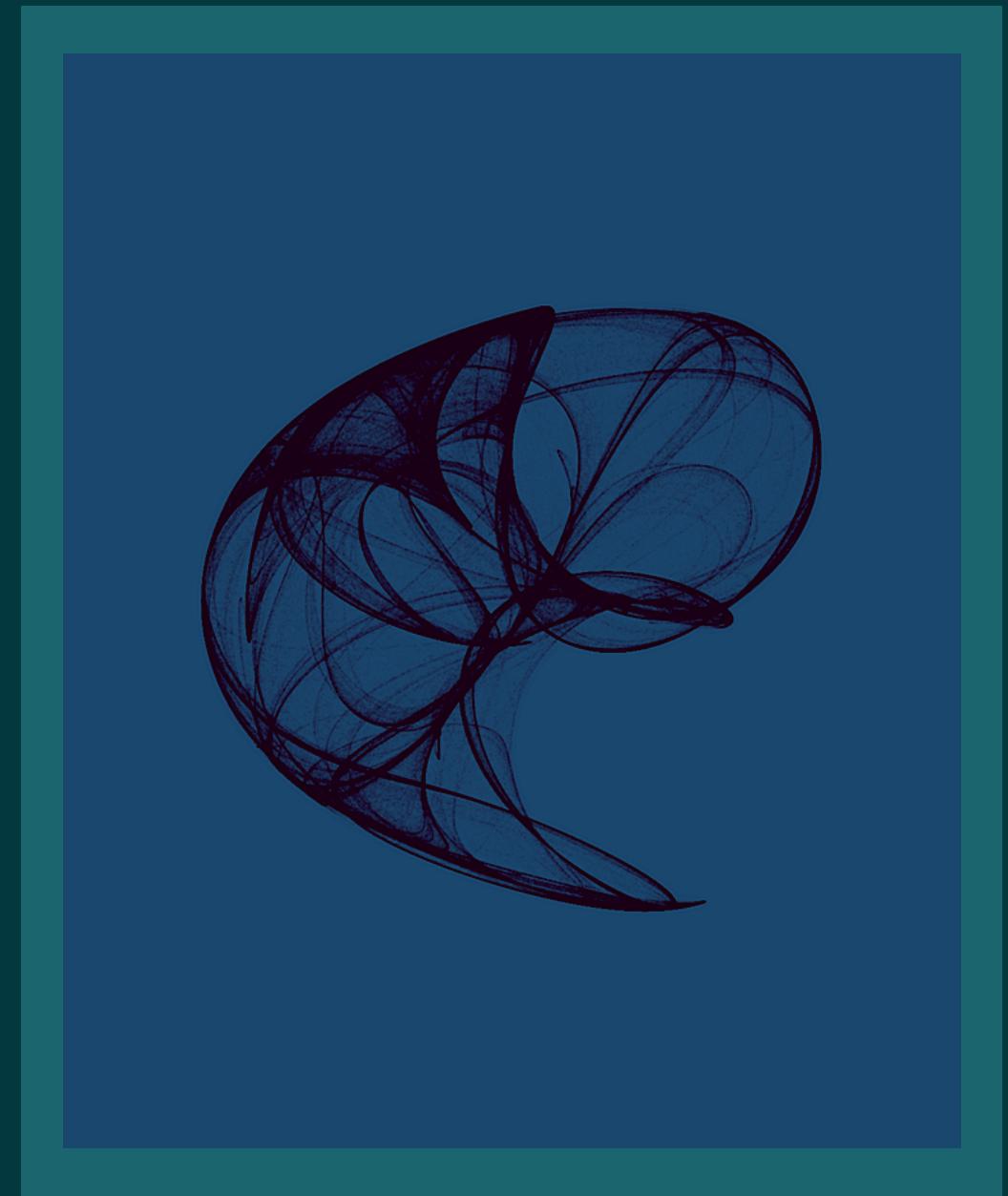
A small change in the initial state can result
in very large difference in the final outcome



Attractors



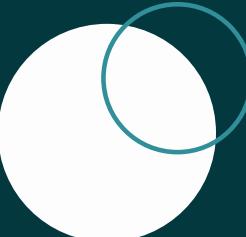
Lorenz system



De Jong Attractor

$$x_{t+1} = \sin(a * yt) - \cos(b * xt)$$

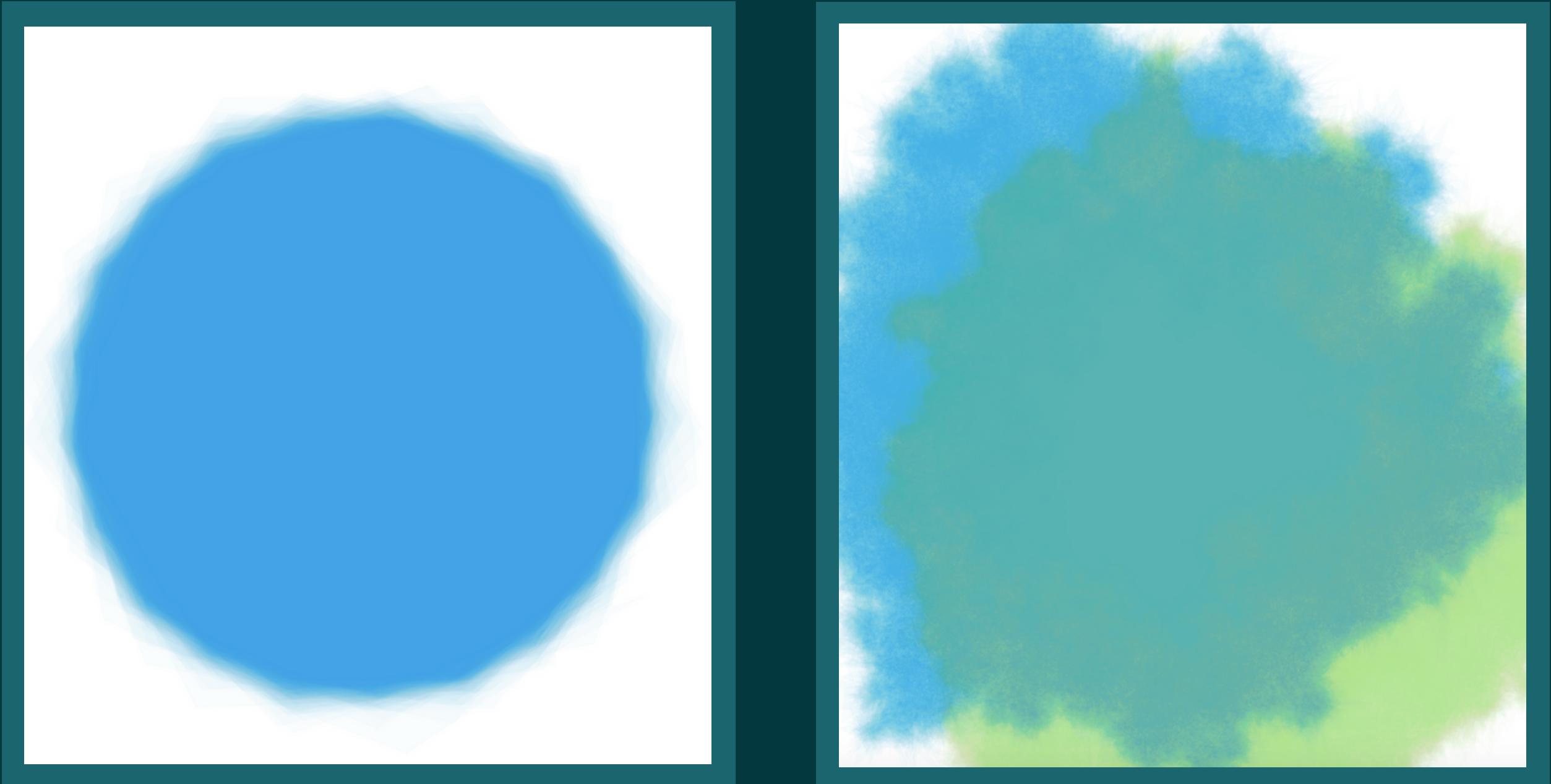
$$y_{t+1} = \sin(c * xt) - \cos(d * yt)$$





SIMULATING PAINT

Creating oil, water color paint effects on
our 2D/3D canvas.



@NEERAJP99

CREATE POLYGON, DIVIDE SIDES

```
● ● ●

1 def setup():
2     global polygon
3     size(800, 800)
4     background(255)
5     colorMode(HSB, 360, 200, 150, 1)
6     # Creating a polygon
7     sideAngle = 0.5
8     while sideAngle < 20:
9         sideX = sin(sideAngle) * width / 4
10        sideY = cos(sideAngle) * width / 4
11        polygon.append({"sideX": sideX, "sideY": sideY})
12        sideAngle += 0.5
13    # Midpoint using Gaussian Distribution
14    mid = 50
15    while mid > 5:
16        for i in range(3):
17            polygon = getMid(polygon, mid)
18            mid /= 2
```

FUNCTIONS TO DIVIDE THE SIDE USING RANDOM GAUSSIAN

```
● ● ●

1 def getCenterPoint(previous, current, standardDeviation):
2     sideX = random_gauss(previous["sideX"] +
3                           (current["sideX"] -
4                            previous["sideX"]) / 2,
5                           standardDeviation)
6     sideY = random_gauss(previous["sideY"] +
7                           (current["sideY"] -
8                            previous["sideY"]) / 2,
9                           standardDeviation)
10    return { "sideX": sideX, "sideY": sideY }
11
12 def getMid(mean, standardDeviation):
13     newVector = [mean[0]]
14     for i in range(1, len(mean), 1):
15         previousValue = mean[i - 1]
16         currentValue = mean[i]
17         midPoint = getCenterPoint(previousValue, currentValue, standardDeviation)
18         newVector.append(previousValue)
19         newVector.append(midPoint)
20     newVector.append(mean[len(mean) - 1])
21     return newVector
```

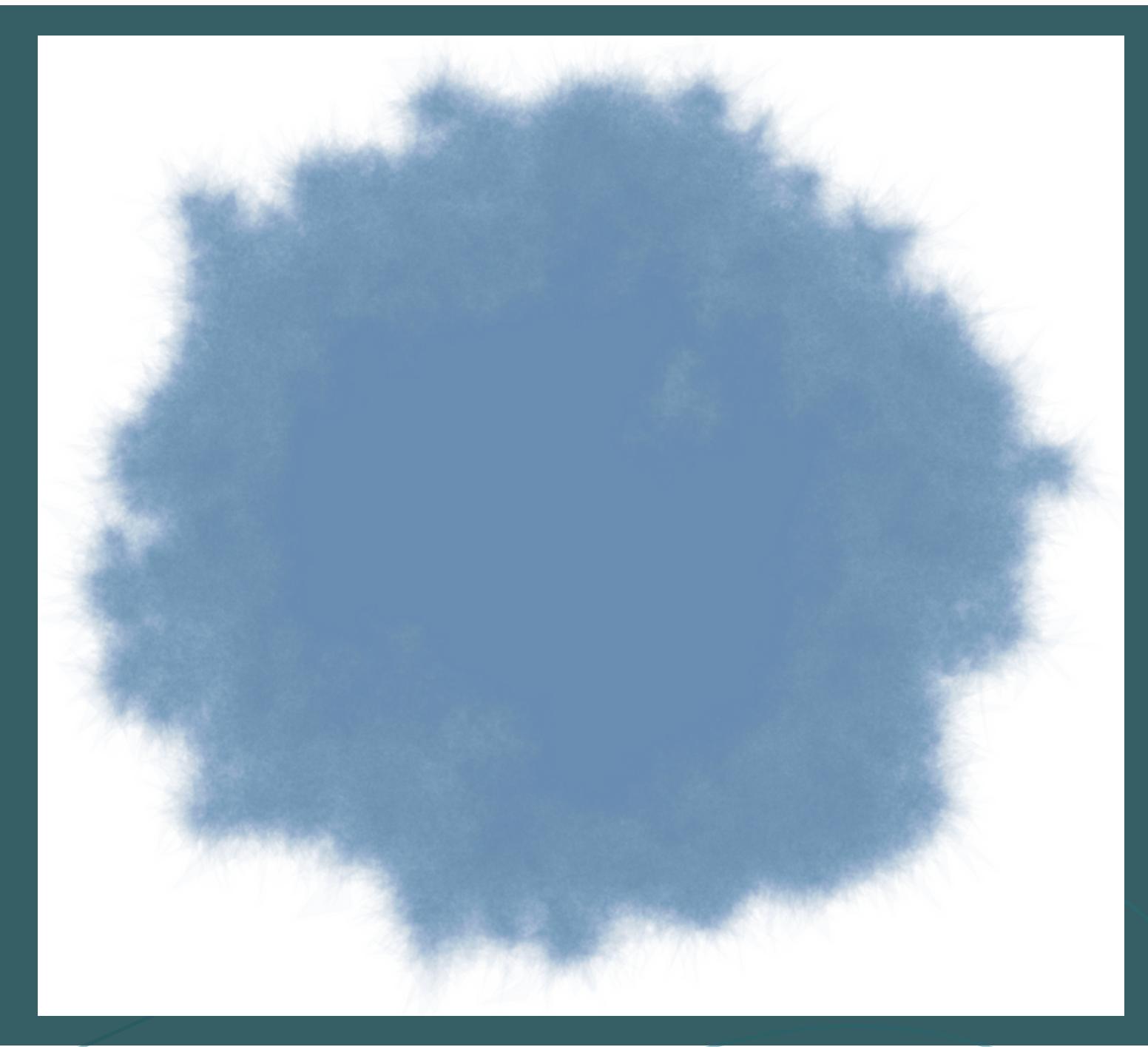
ADD VERTEX TO THE NEW SIDES

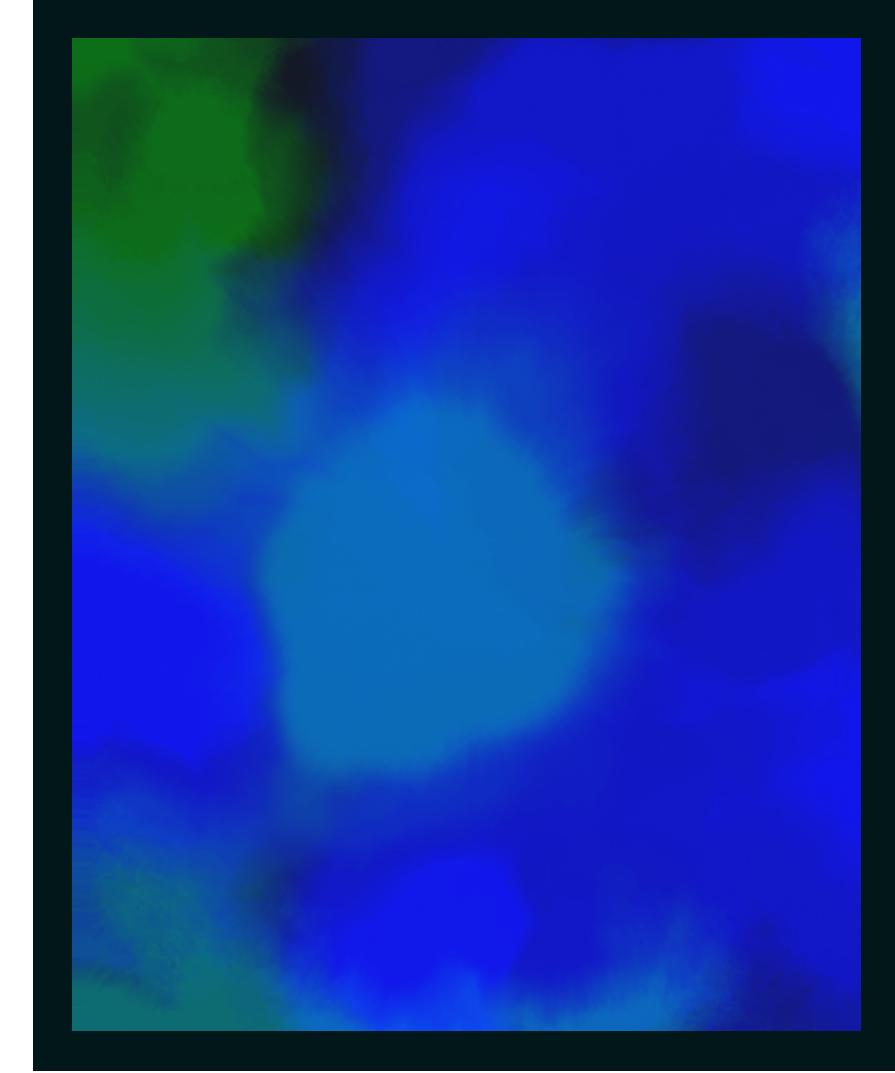
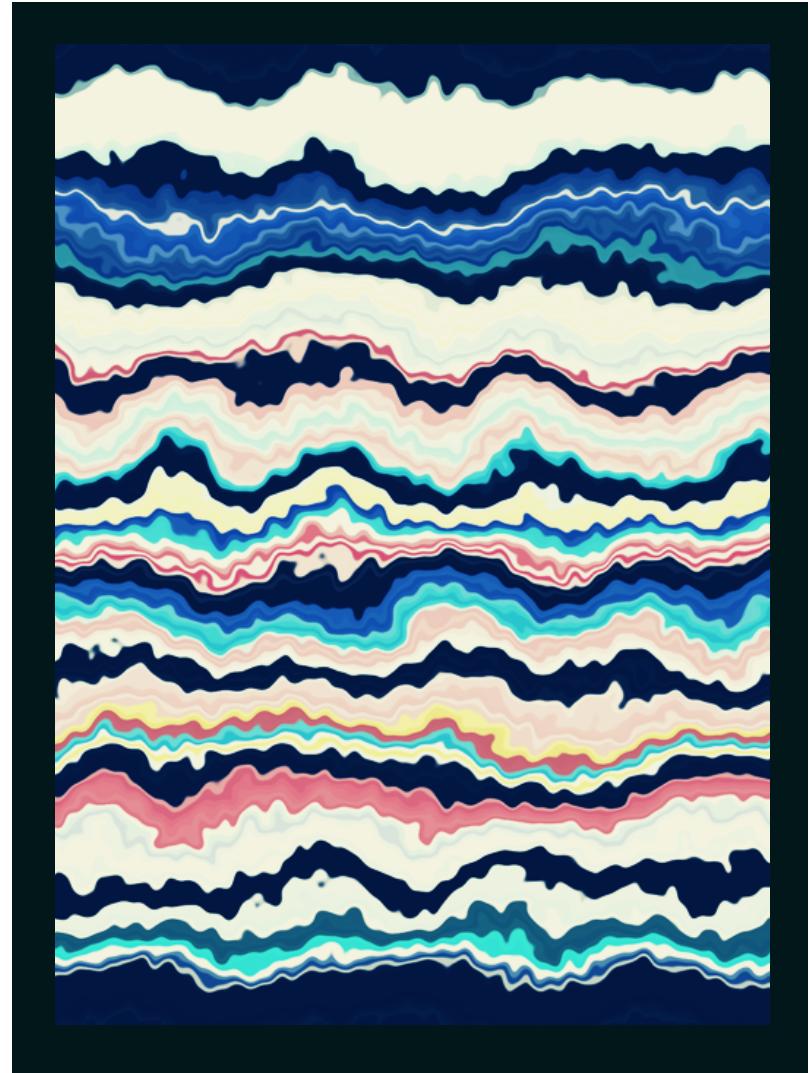
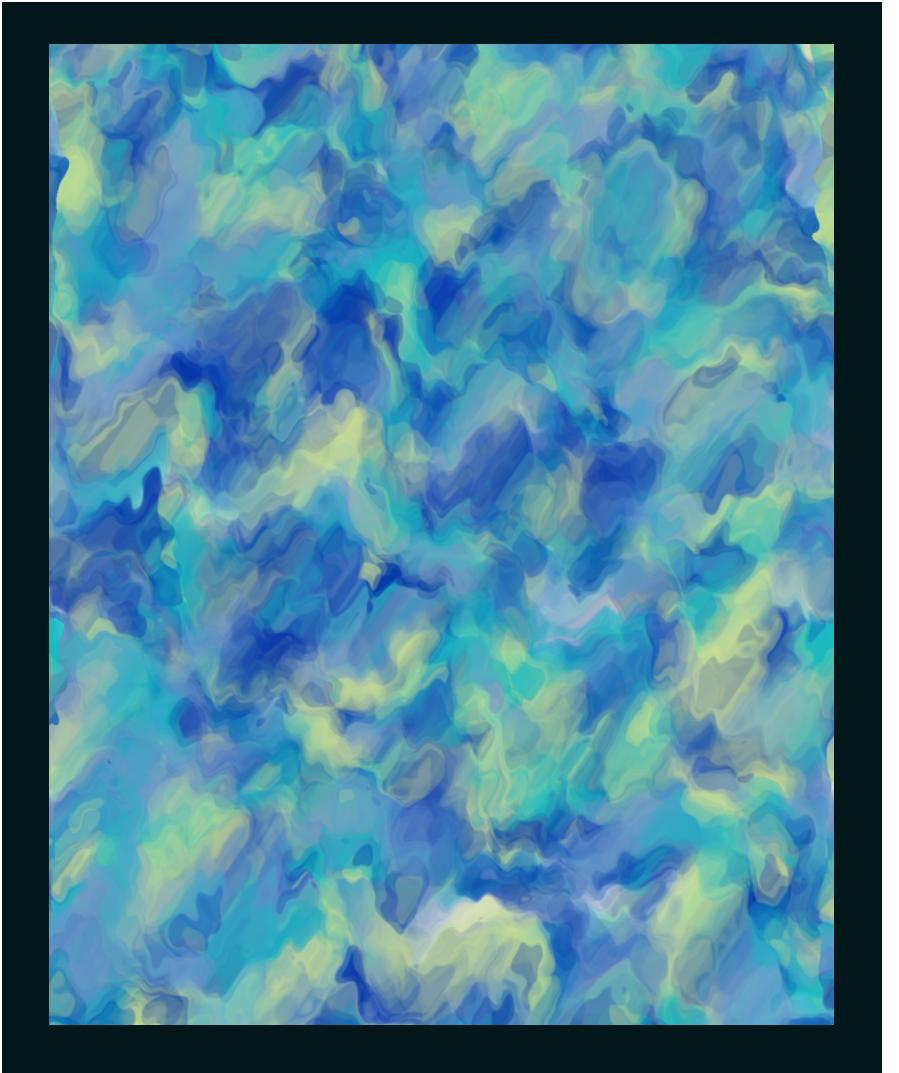
```
● ● ●

1 def draw():
2     global polygon, randomGauss
3     for i in range(20):
4         push()
5         noStroke()
6         fill(200, 100, 80, 0.02)
7         beginShape()
8         translate(width * 0.5, height * 0.5);
9         for j in range(len(polygon)):
10            currentVector = polygon[j]
11            x = random_gauss(currentVector['sideX'], random(25))
12            y = random_gauss(currentVector['sideY'], random(25))
13            vertex(x, y)
14        endShape(CLOSE)
15        pop()
16    noLoop()
```

ADD A CUSTOM RANDOM GAUSSIAN METHOD

```
1 # Custom function to return a random number fitting a Gaussian distribution.
2 def random_gauss(m, sd):
3     value = 1
4     while value >= 1:
5         x1 = random(2) - 1
6         x2 = random(2) - 1
7         value = x1 * x1 + x2 * x2
8         value = math.sqrt(-2 * math.log(value) / value)
9         y1 = x1 * value
10        y2 = x2 * value
11        mean = m or 0
12        standardDeviation = sd or 1
13        return y1 * standardDeviation + mean
```

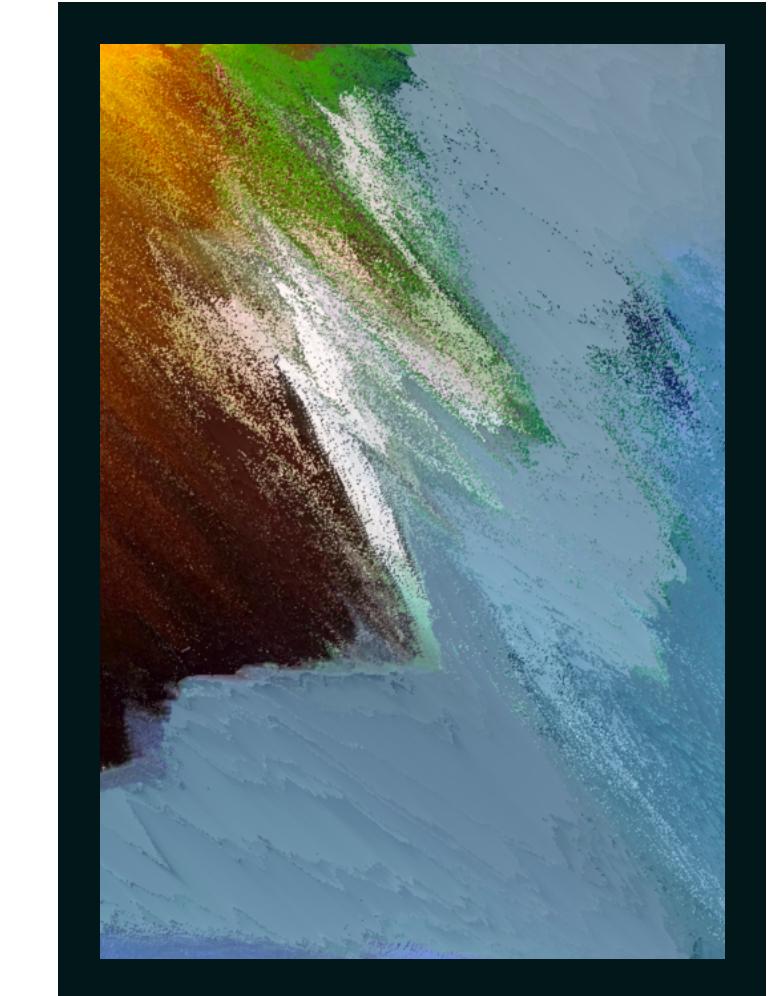
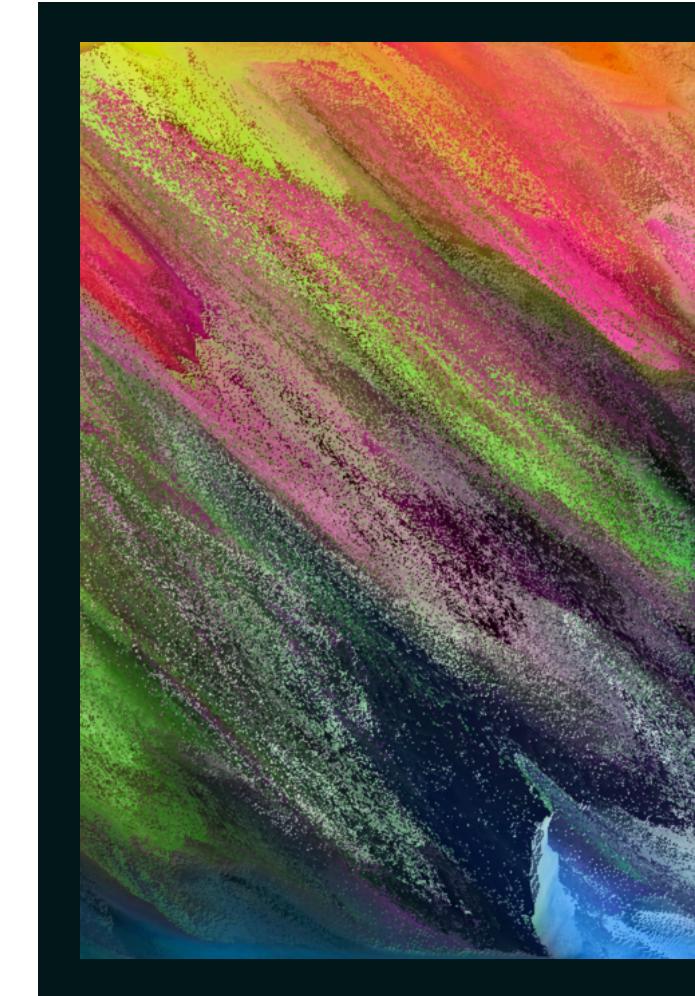
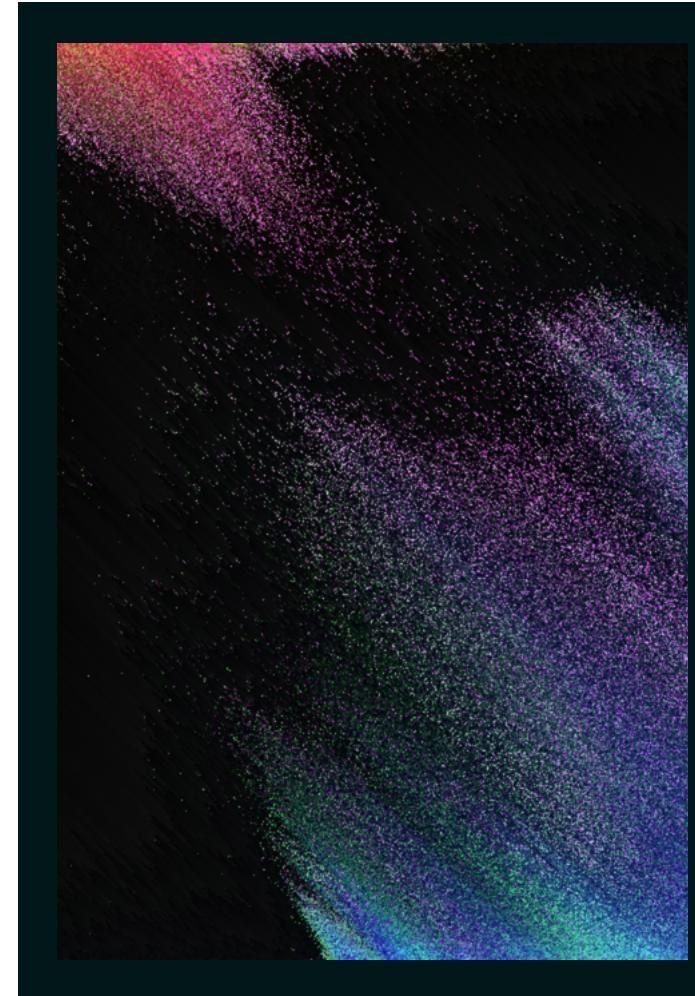
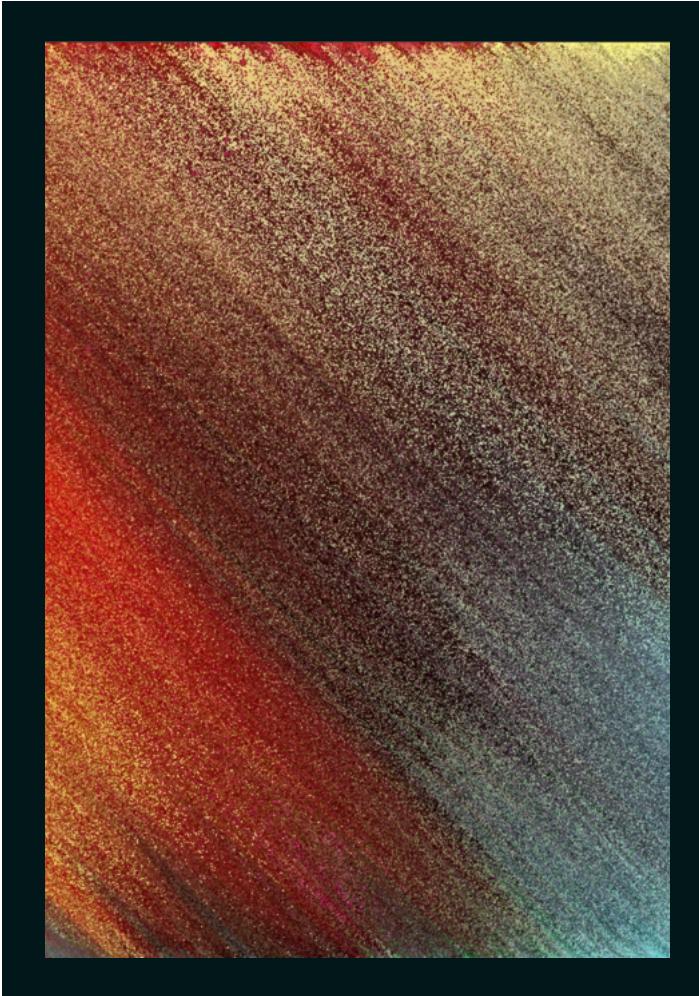




@NEERAJP99

Pixel Sorting Algorithms

v1.0



SIMPLE IMPLEMENTATION OF SORTING PIXELS

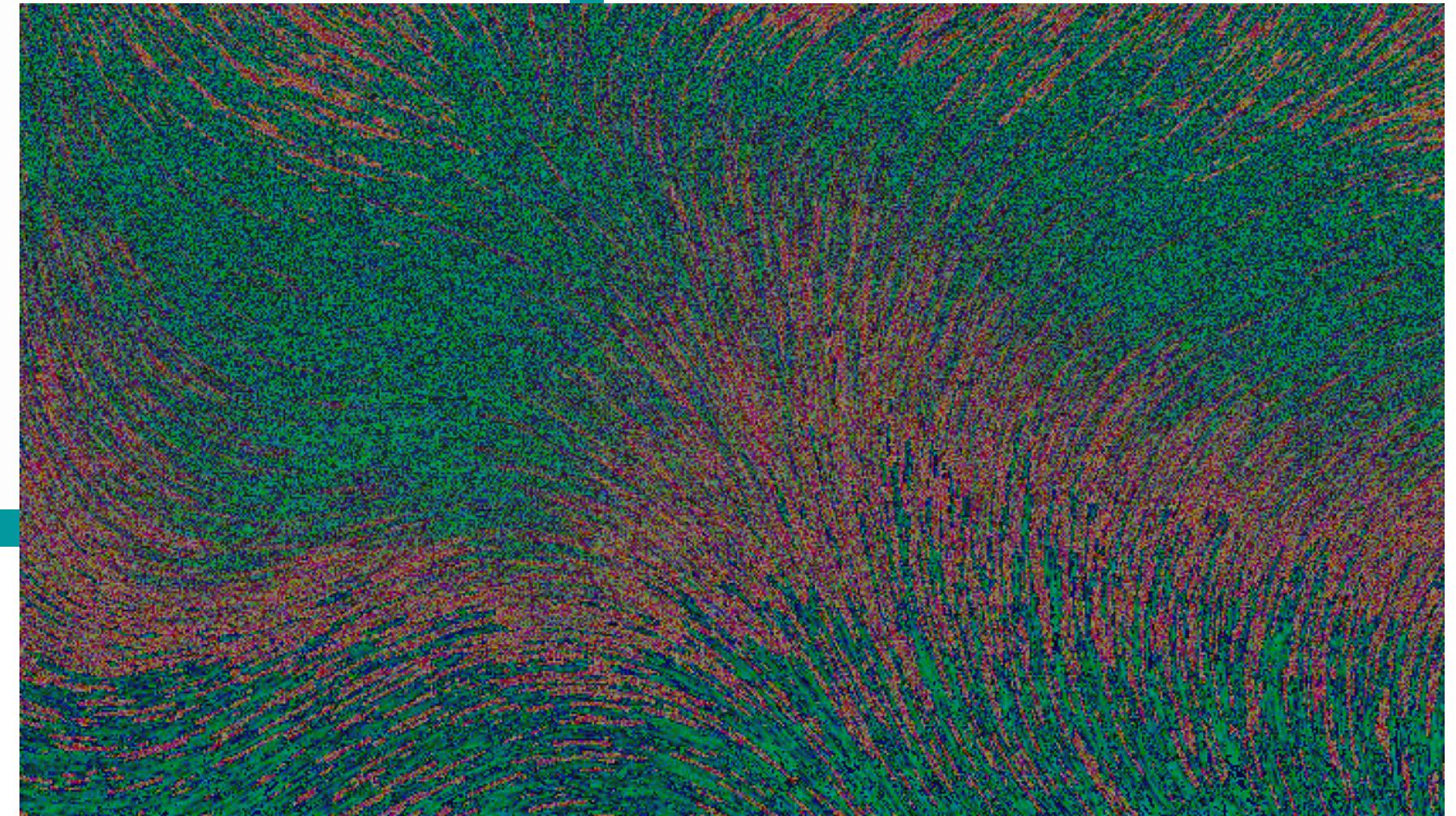
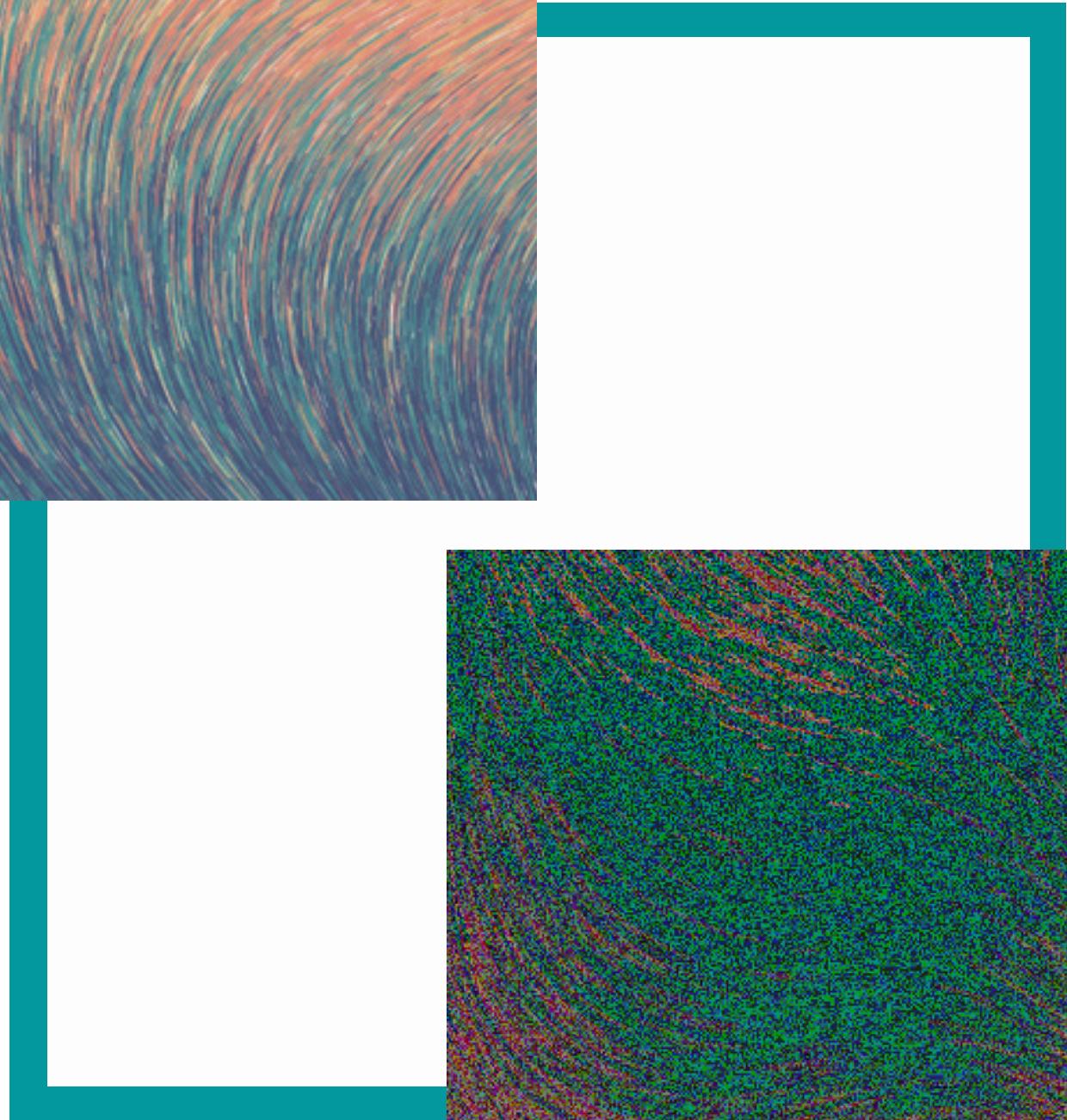
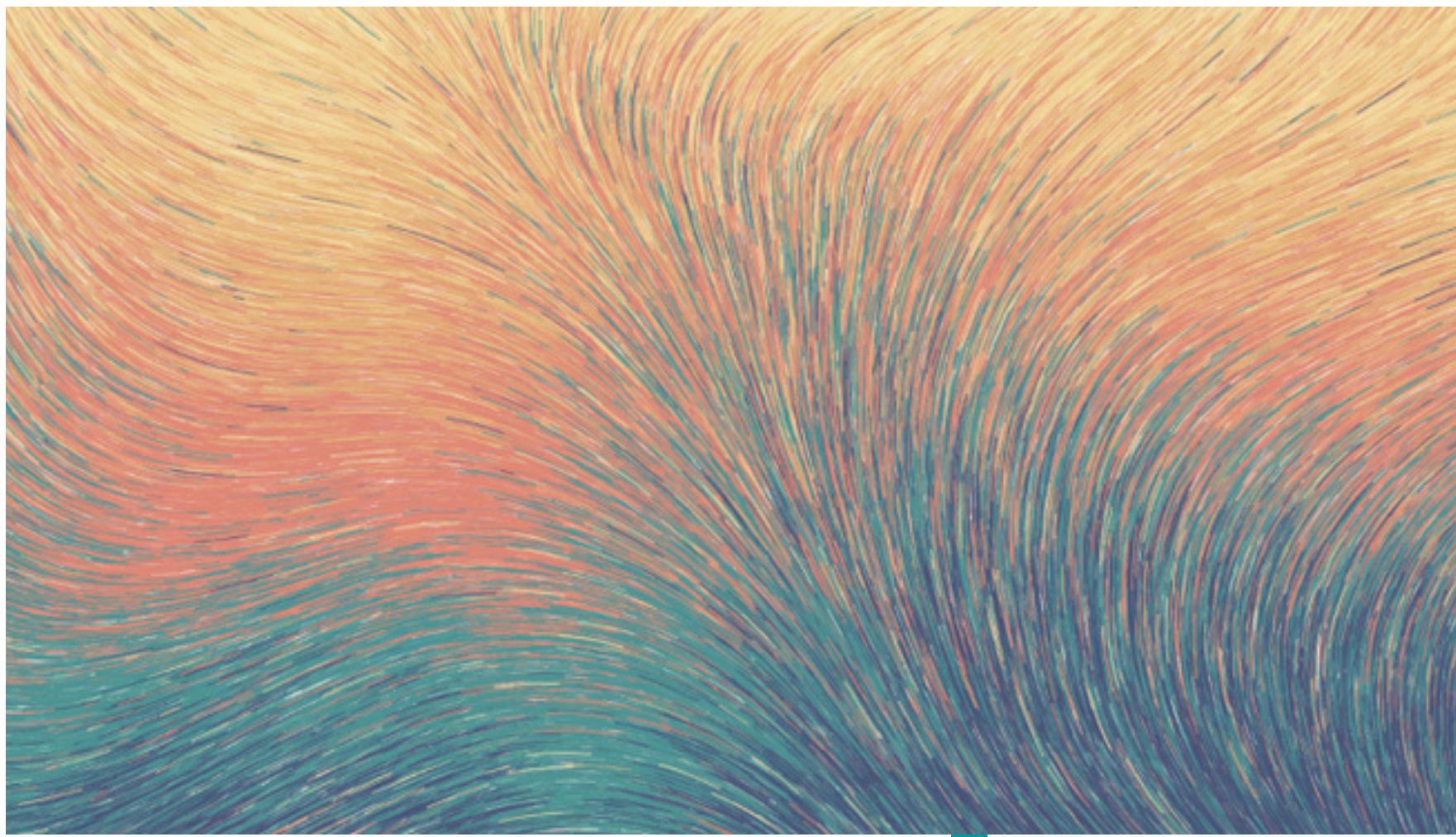
```
● ● ●

1 def draw():
2     global img, imgNew
3     image(img, 0, 0)
4     pixelArray = img.pixels
5     # Looping through rows and columns
6     for x in range(img.height):
7         newPixelArray = list()
8         for y in range(img.width):
9             # Get pixel positions
10            pixel = img.get(y, x)
11            # Get the pixel hex value and convert it into rgb
12            rgbPixel = convert_to_rgb(str(pixel))
13            rgbPixel.append(255)
14            newPixelArray.append(rgbPixel)
15        # Sort the new pixel array
16        for j in range(len(newPixelArray) - 2):
17            # for j in range (len(newPixelArray) - 2):
18            if (((newPixelArray[j][0] + newPixelArray[j][1] + newPixelArray[j][2])) >
19                (newPixelArray[j+1][0] + newPixelArray[j+1][1] + newPixelArray[j+1][2])):
20                newPixelArray[j], newPixelArray[j+1] = newPixelArray[j+1], newPixelArray[j]
21        # Set the image pixels to the second image
22        for i in range(img.width):
23            a = newPixelArray[i][0]
24            b = newPixelArray[i][1]
25            c = newPixelArray[i][2]
26            cl = color(a, b, c)
27            imgNew.set(i, x, cl)
28        imgNew.updatePixels();
29        image(imgNew,0,0);
30        noLoop()
```

SIMPLE IMPLEMENTATION OF SORTING PIXELS

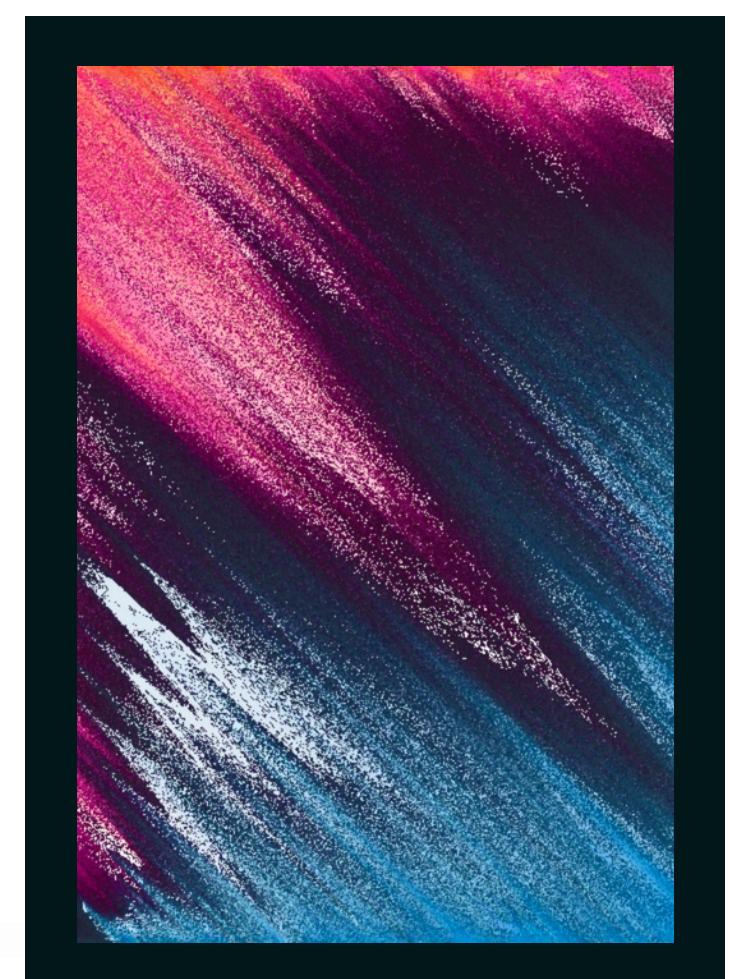
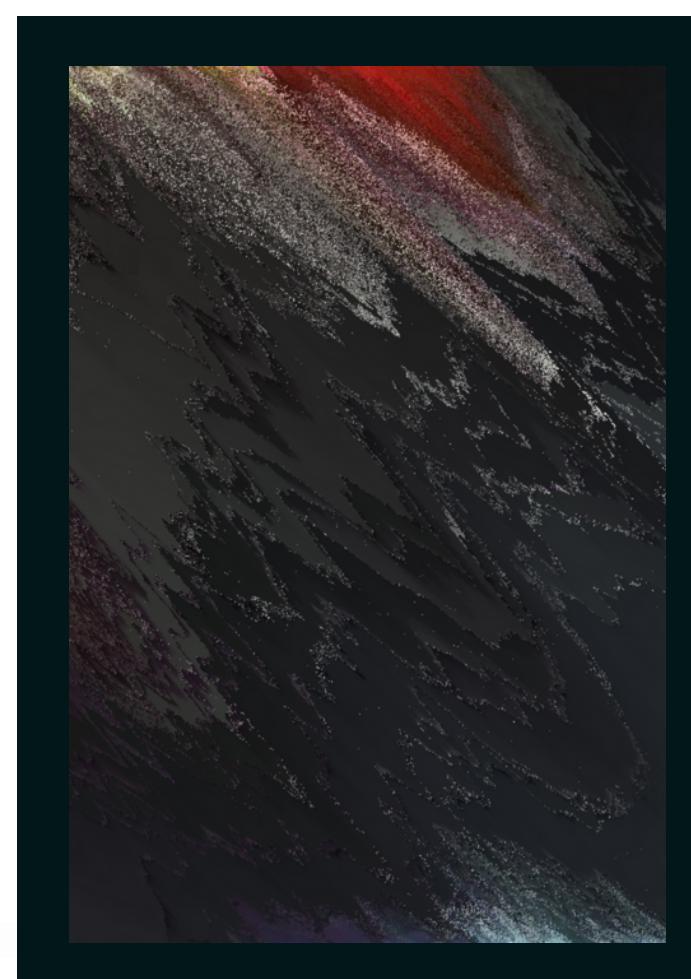
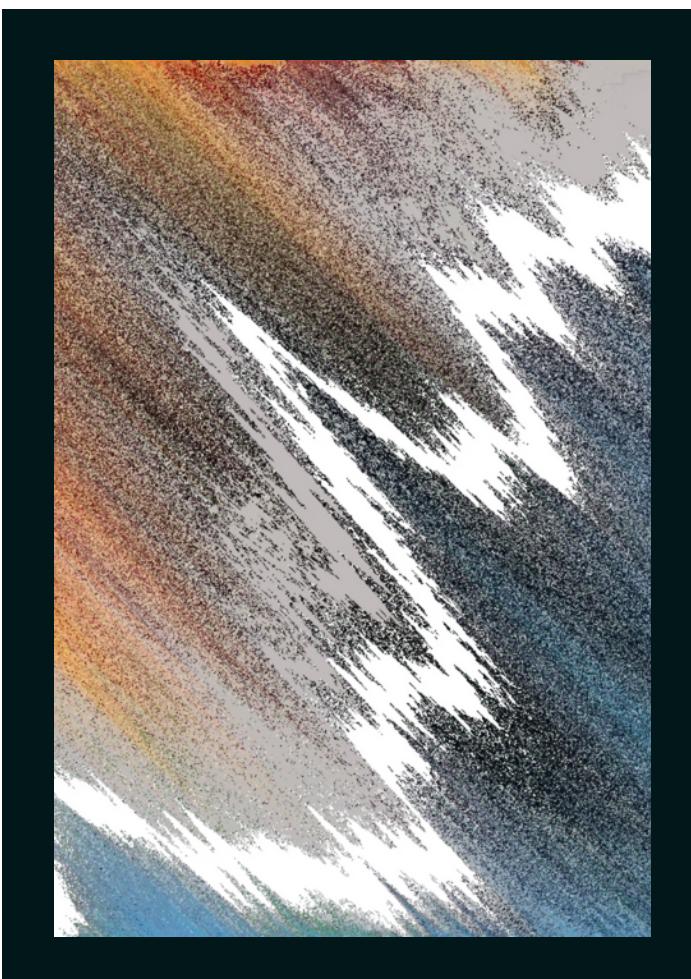
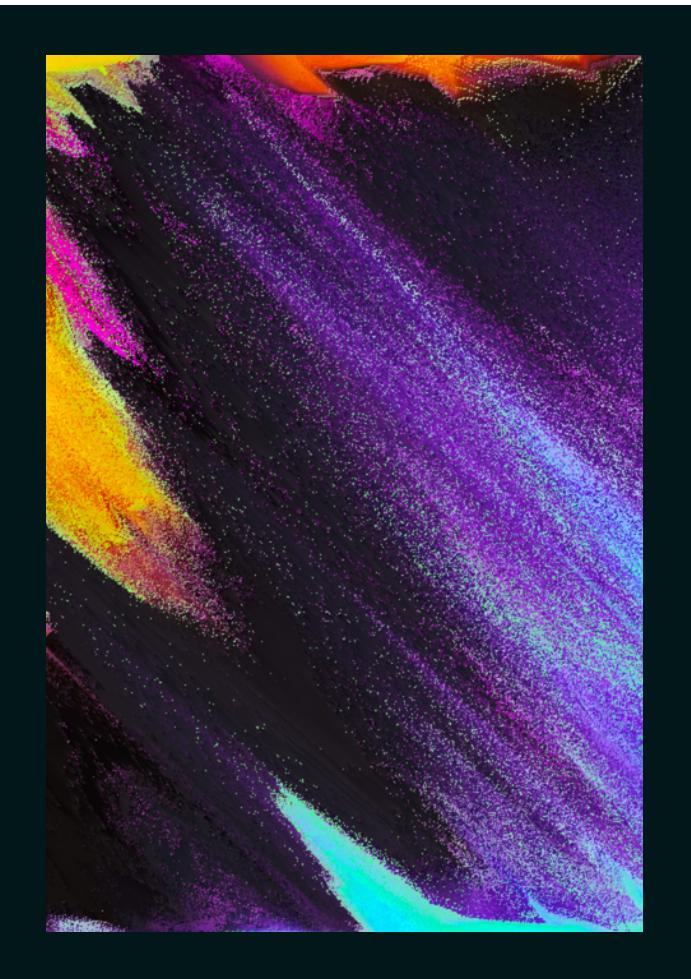
```
● ● ●

1 def setup():
2     global img, imgNew
3     size(700, 400)
4     background(0)
5     img = loadImage("sketch.jpg")
6     imgNew = loadImage("sketch.jpg")
7     imgNew.resize(700, 500)
8     img.resize(700, 400)
9     img.loadPixels()
10    colorMode(RGB)
11
12 def convert_to_rgb(hexcode):
13     hexcode = hexcode.lstrip('-')
14     return list(int(hexcode[i:i+2], 16) for i in (0, 2, 4))
```



Pixel Sorting Algorithms

v2.0



@NEERAJP99

THANK YOU



neerajp99

