EUROPYTHON 2020

The Joy of Creating Art with Code.

Presented by Neeraj Pandey

@NEERAJP99

NEERAJ PANDEY

@NEERAJP99

ASHOKA UNIVERSITY

Sophomore student at Ashoka University.

Software Development, Generative Art, Distributed Computing and Quantitative Finance.

POINTS FOR DISCUSSION

Generative Art - Principles and Elements

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

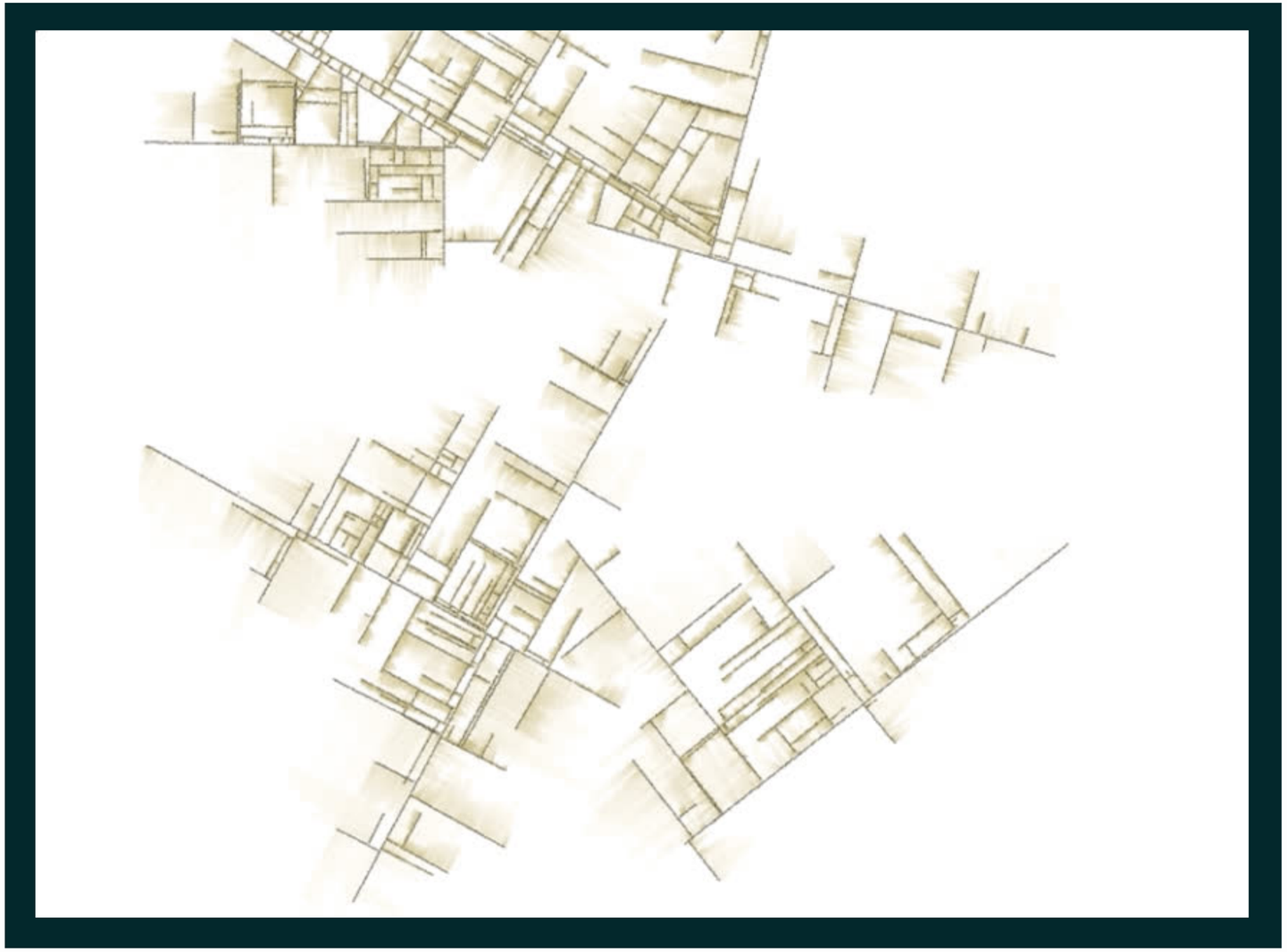
@NEERAJP99

PRINCIPLES AND ELEMENTS

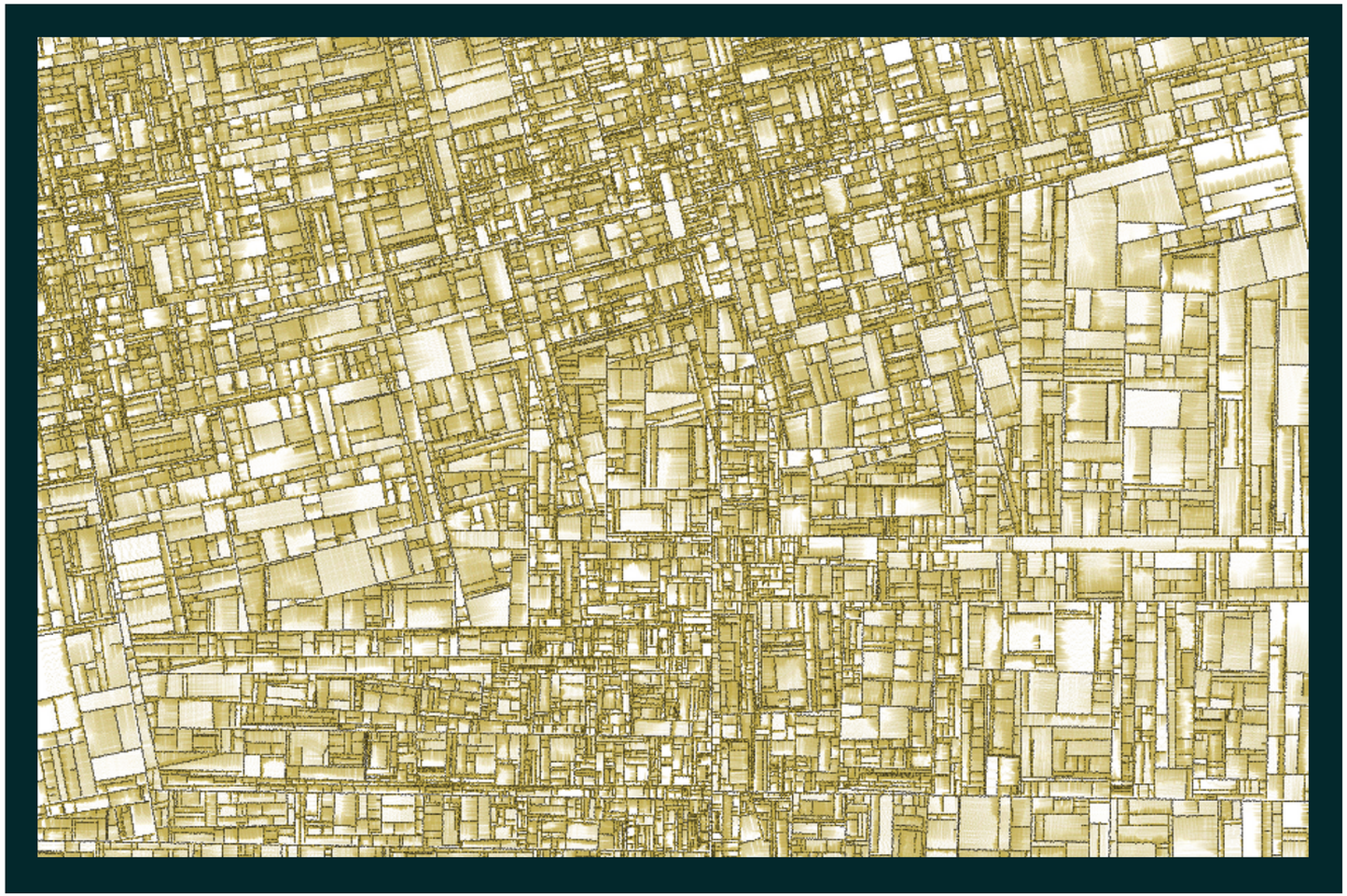
Elements: color, form, line, shape, space, the randomness and the texture.

Principles: rhythm, contrast, harmony, balance, movement, proportion

Substrate, Jared Tarbell



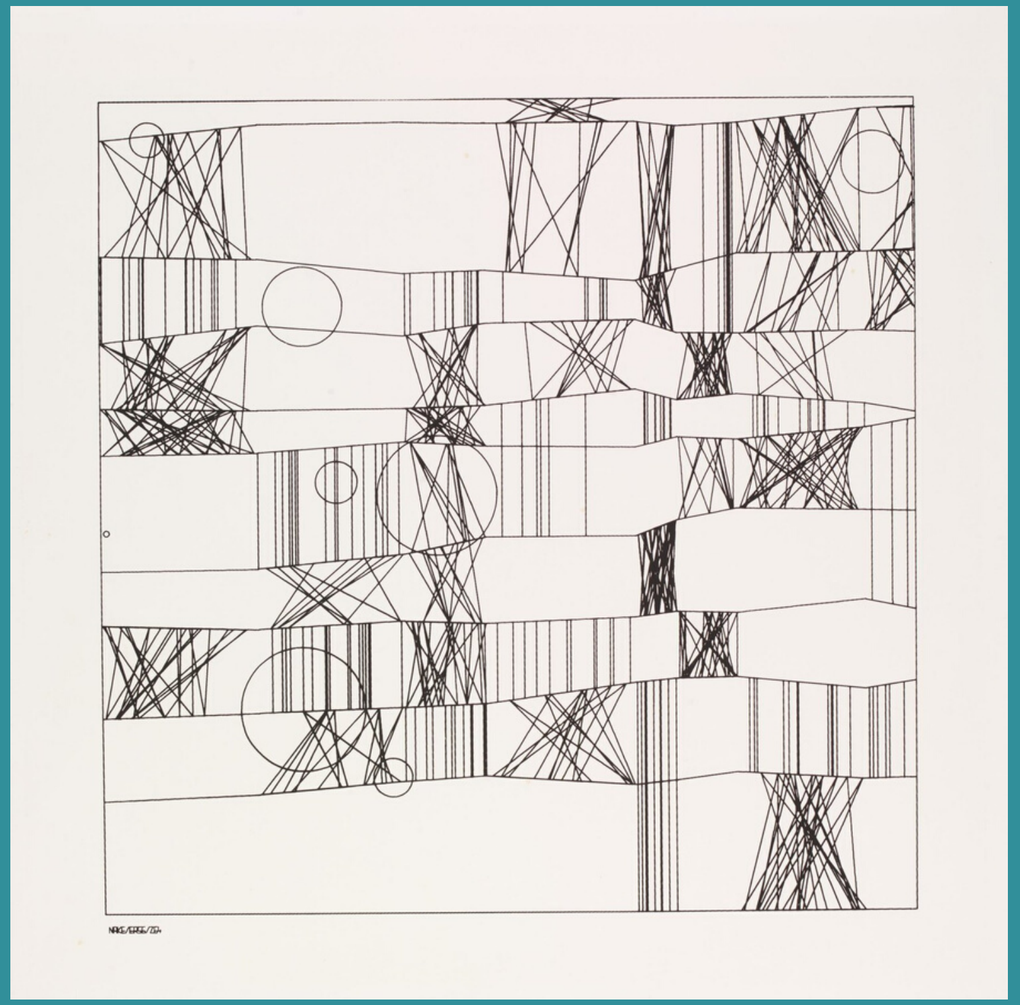
Created using Python mode for Processing



Created using Python mode for Processing

@NEERAJP99

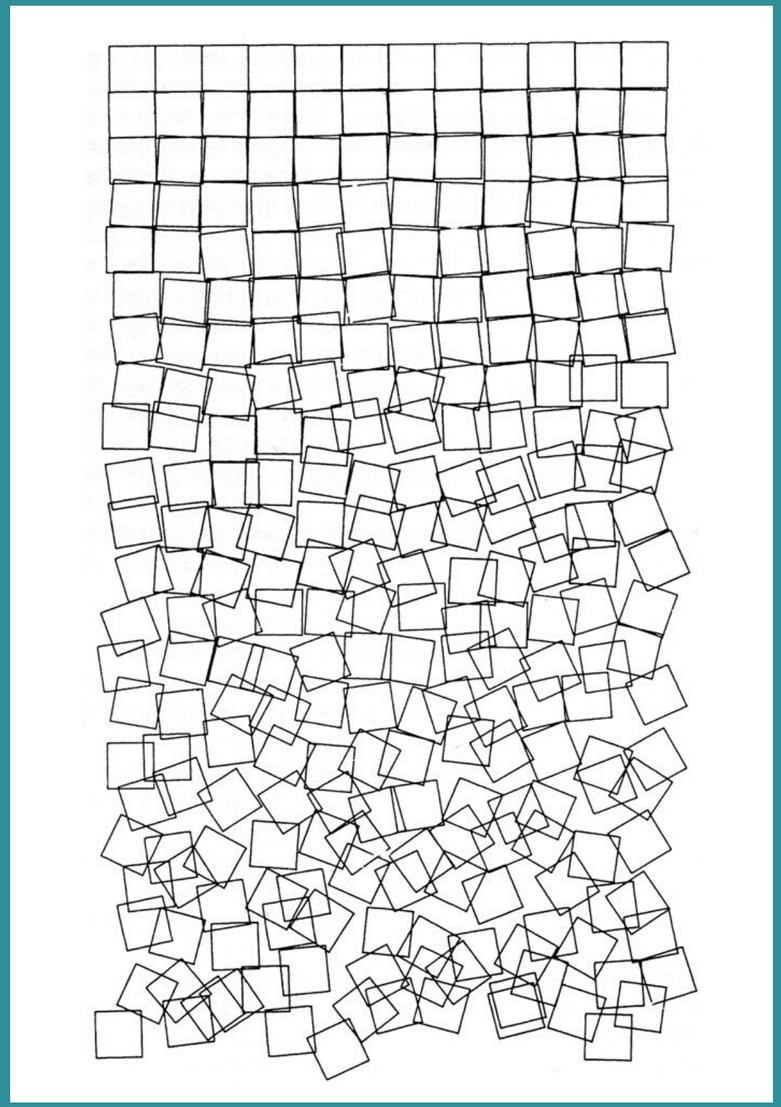
HISTORY OF GENERATIVE ART



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

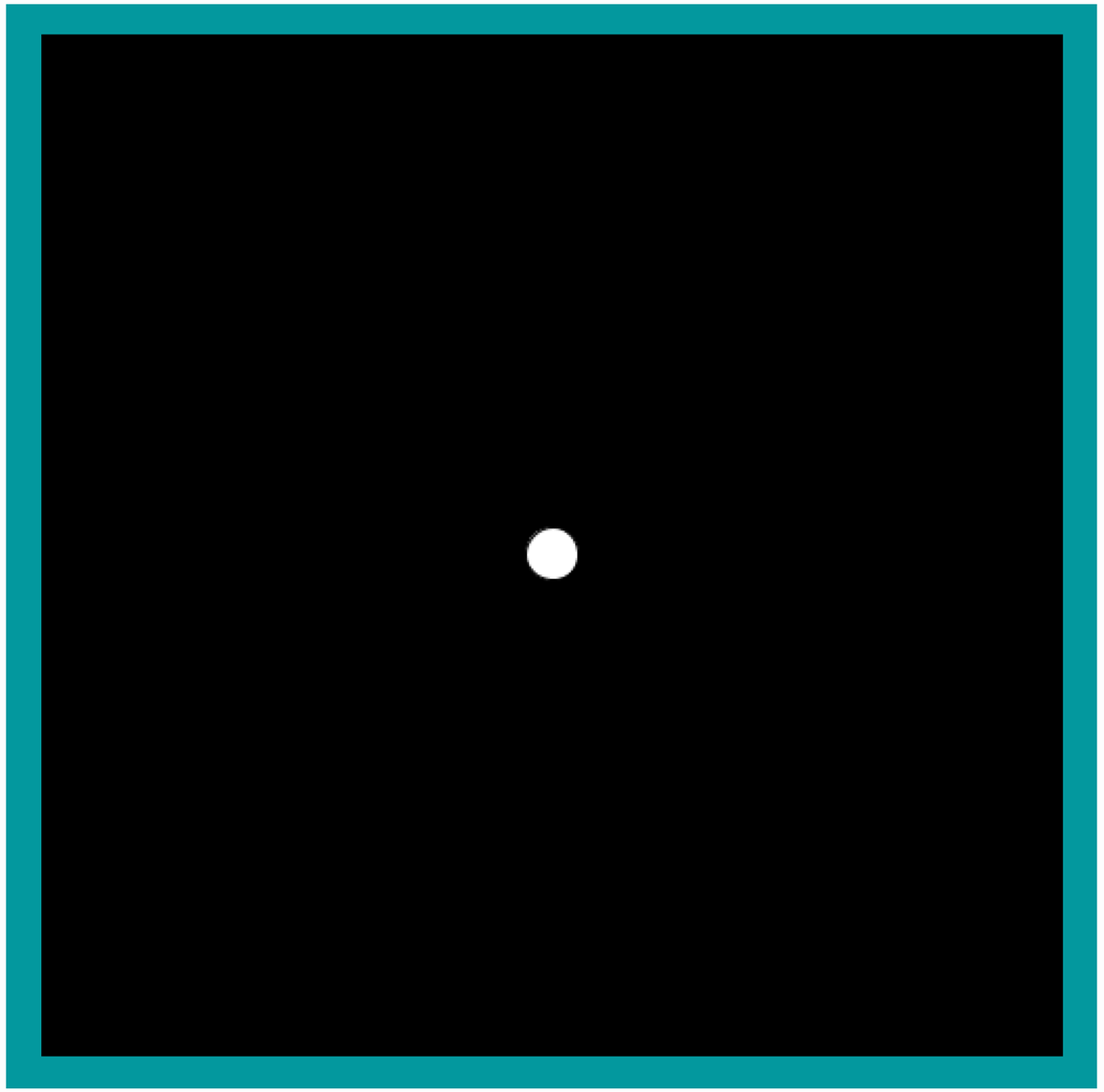
Processing for Android

PROCESSING.PY

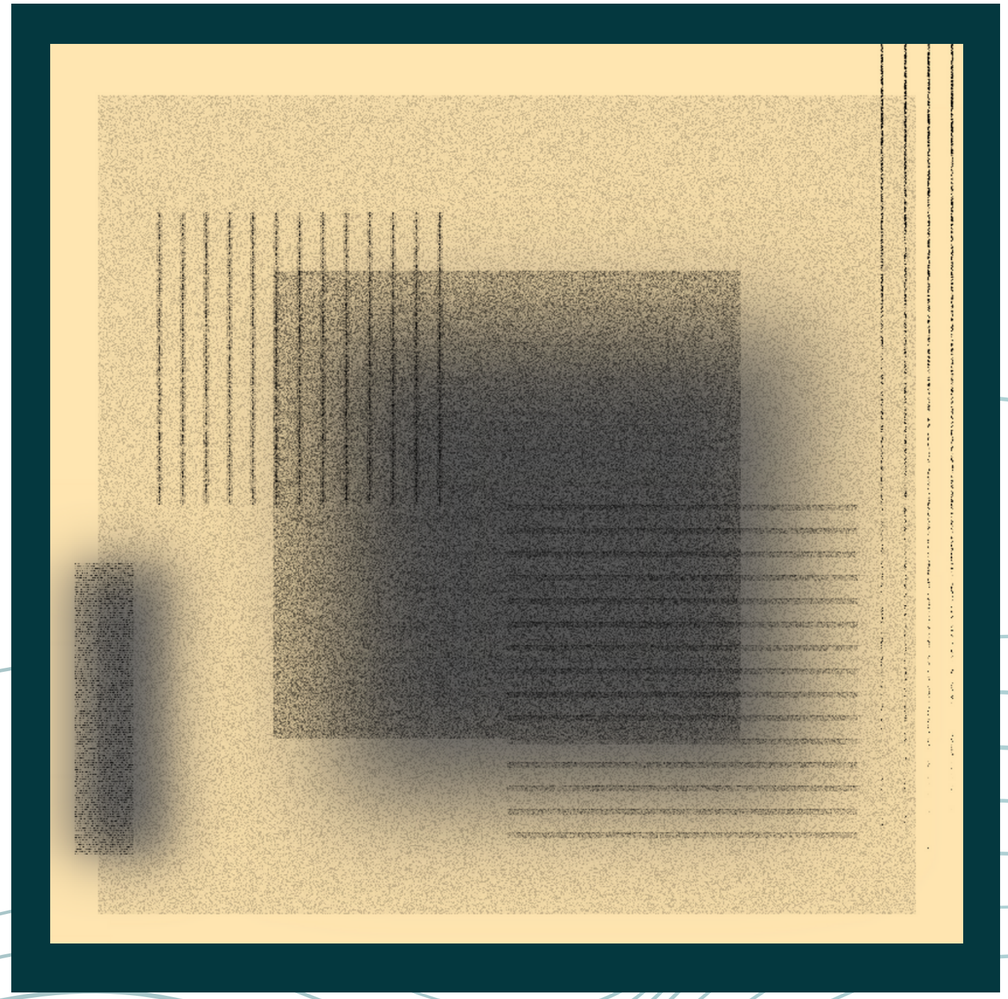
@NEERAJP99

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)
```



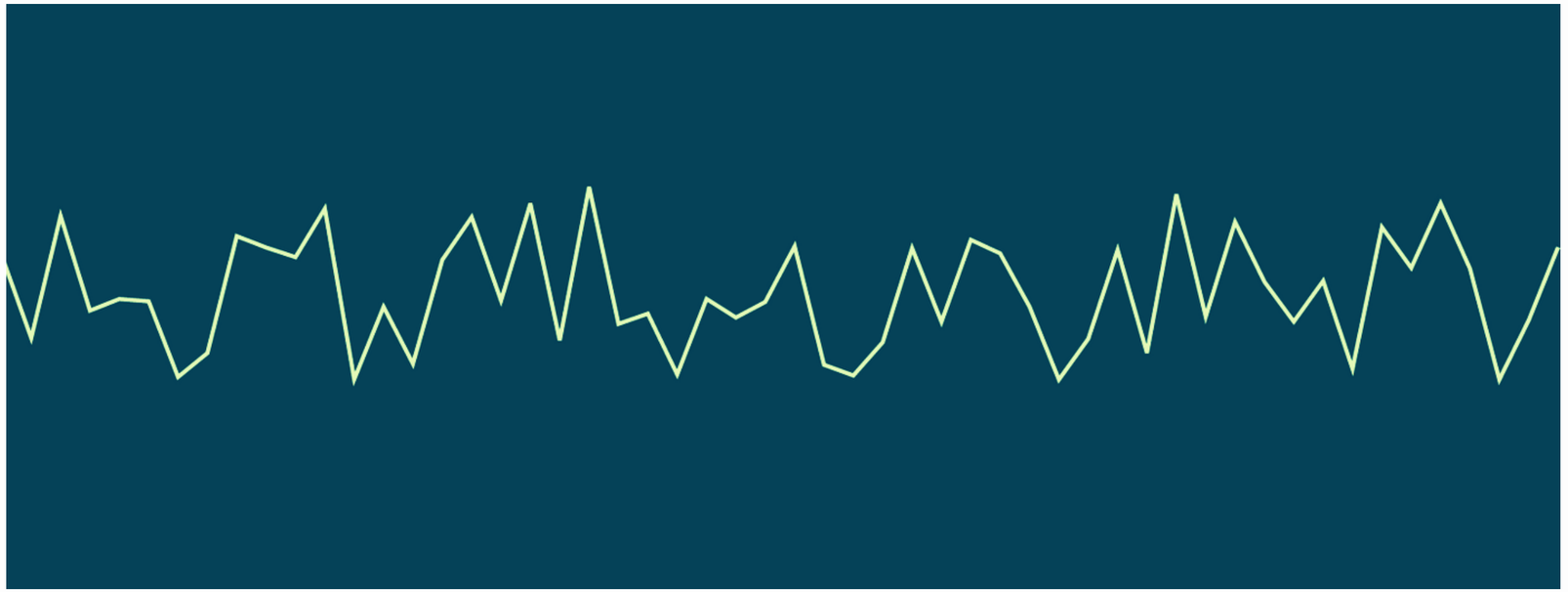
USING MATHEMATICS AND ALGORITHMS



@NEERAJP99

RANDOM

Generates random floating point numbers .



RANDOM

Generates random floating point numbers .

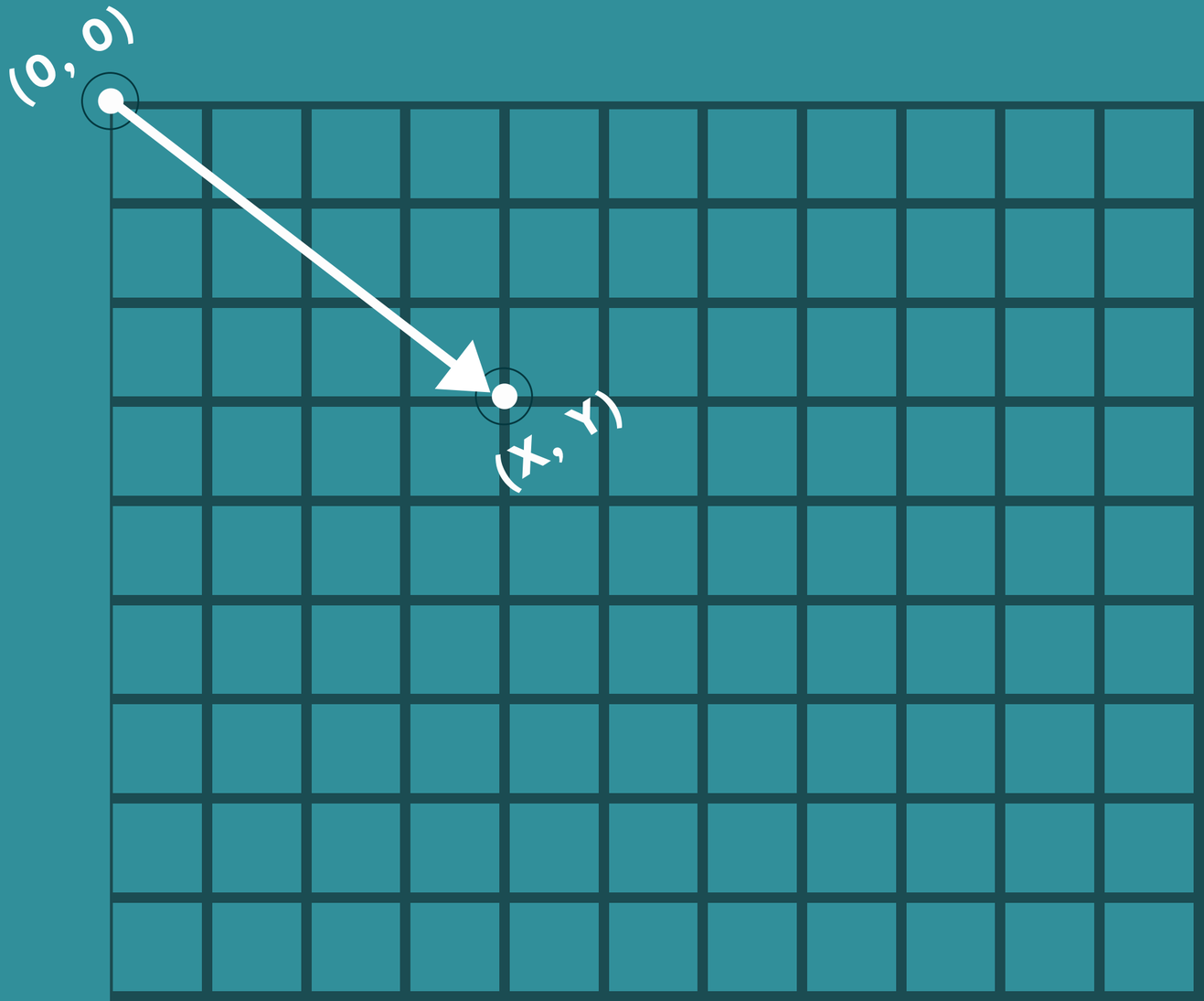
```
1 random()
2 # floating value between [0, 1)
3 random([min], [max])
4 # Random value between [min, max)
5 random(x)
6 # Random value between [0, x)
```

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

ARTIST'S CANVAS

DRAWING A POINT

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

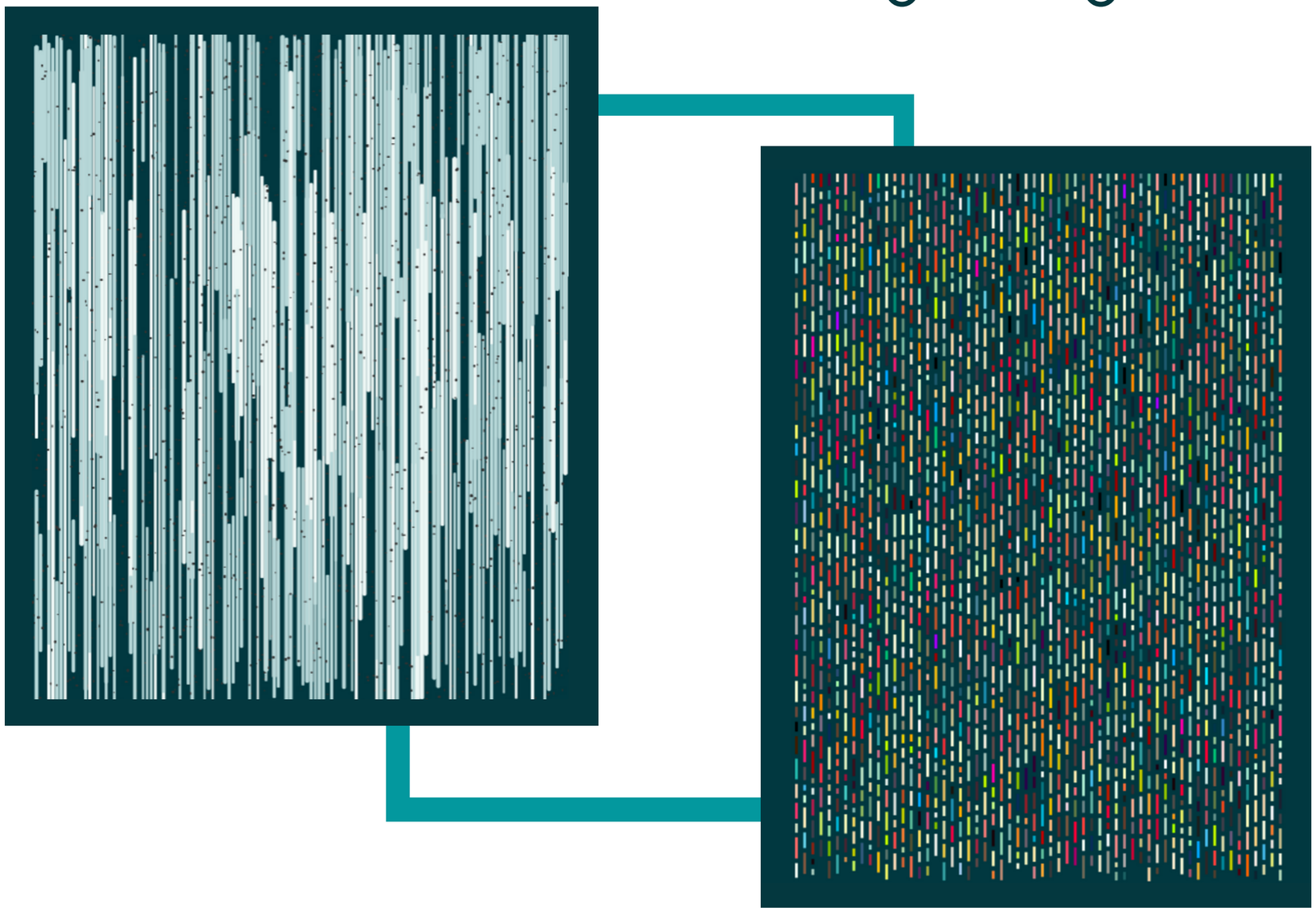


CREATING A POINT AND LINE

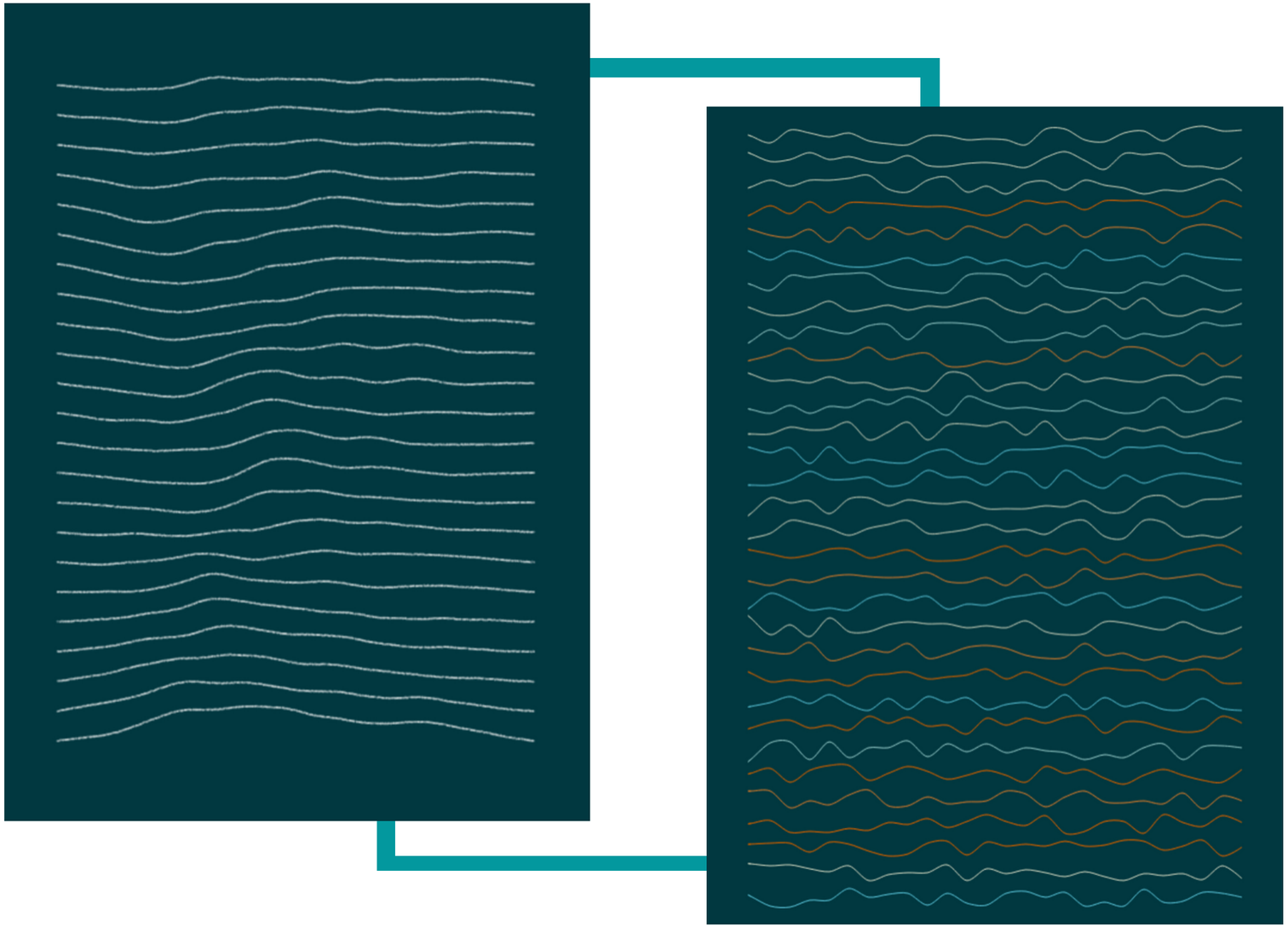
```
point(x, y)
# Creates a point at (x, y) from the origin.
point(x, y, z)
# Creates a point at (x, y, x).
dist(x1, y1, x2, y2)
# Calculates the distance between 2 points.
line(x1, y1, x2, y2)
# Create a line by joining 2 points.

stroke(color)
# Sets the color used to draw lines
# and borders around shapes.
strokeWeight(x)
# Sets the thickness of the stroke.
```

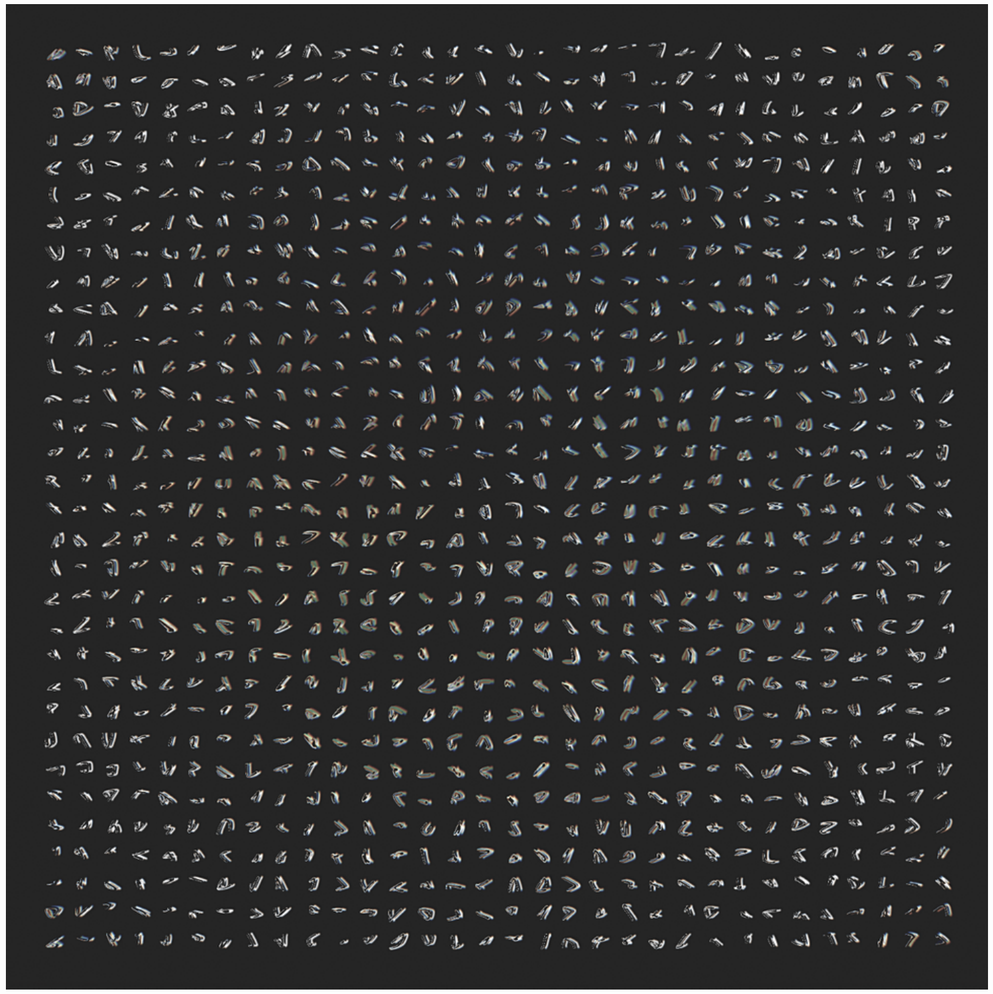
```
context.set_line_width(0.02)
context.move_to(x, y)
context.line_to(x1, y1)
context.stroke()
```

Using straight lines



Using vector operations





CURVE VERTEX & BEZIER CURVES

Curve Vertex: It specifies the vertex coordinates for curves.

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

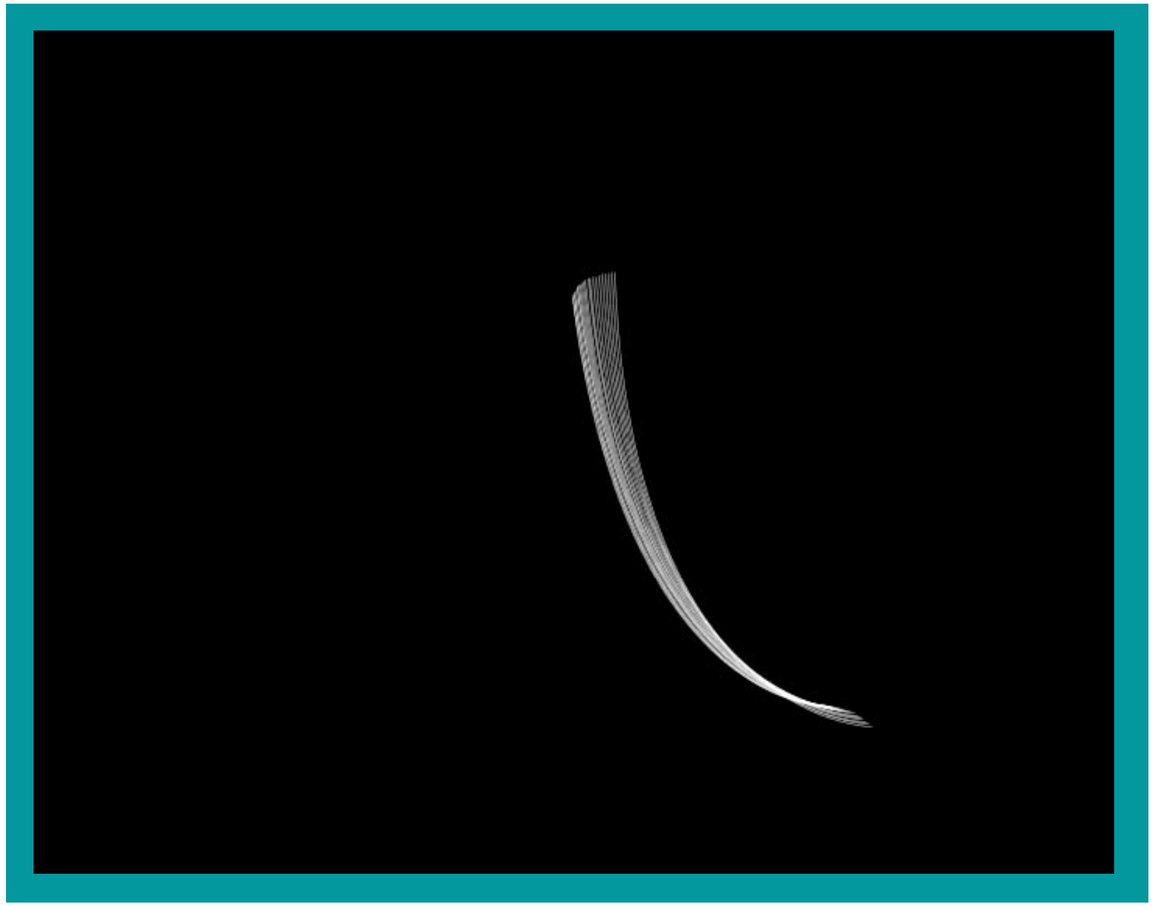
@NEERAJP99

CREATING BEZIER CURVES

```
1 # Draws the bezeir 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



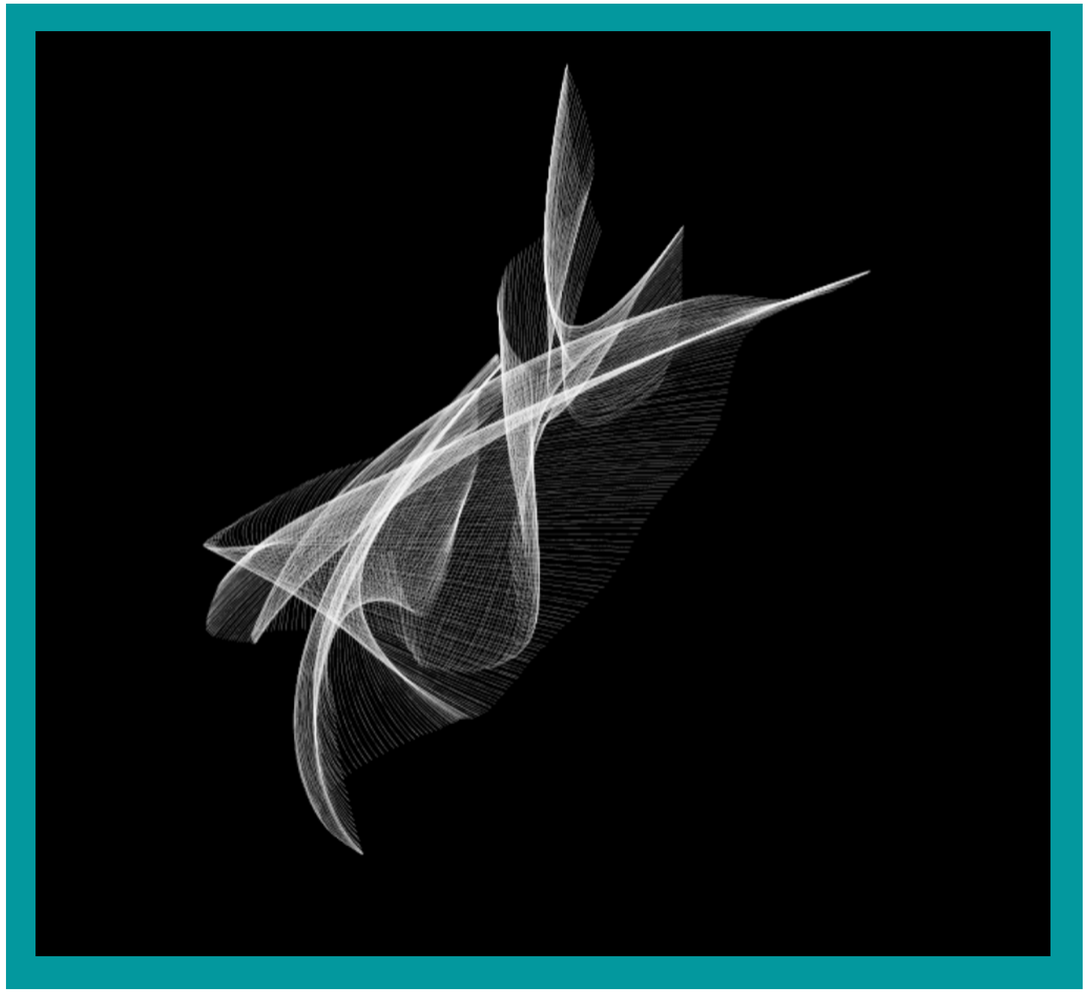
@NEERAJP99

CREATING BEZIER CURVES

```
def setup():
    global offset
    offset = random(100)
    size(700, 700)
    background(0)
    stroke(255)
    strokeWeight(0.5)
    noFill()


def draw():
    global offset
    x,y = [], []
    m , n = 20, 70
    if frameCount <= 600:
      for i in range(4):
          x.append(width * noise(offset + m))
          y.append(height * noise(offset + n))
          m += 10
          n += 10
    bezier(x[0], y[0], x[1], y[1], x[2], y[2], x[2], y[2]);
    offset += 0.005;
```

@NEERAJP99

USING BEZIER CURVES TO CREATE WAVES



@NEERAJP99

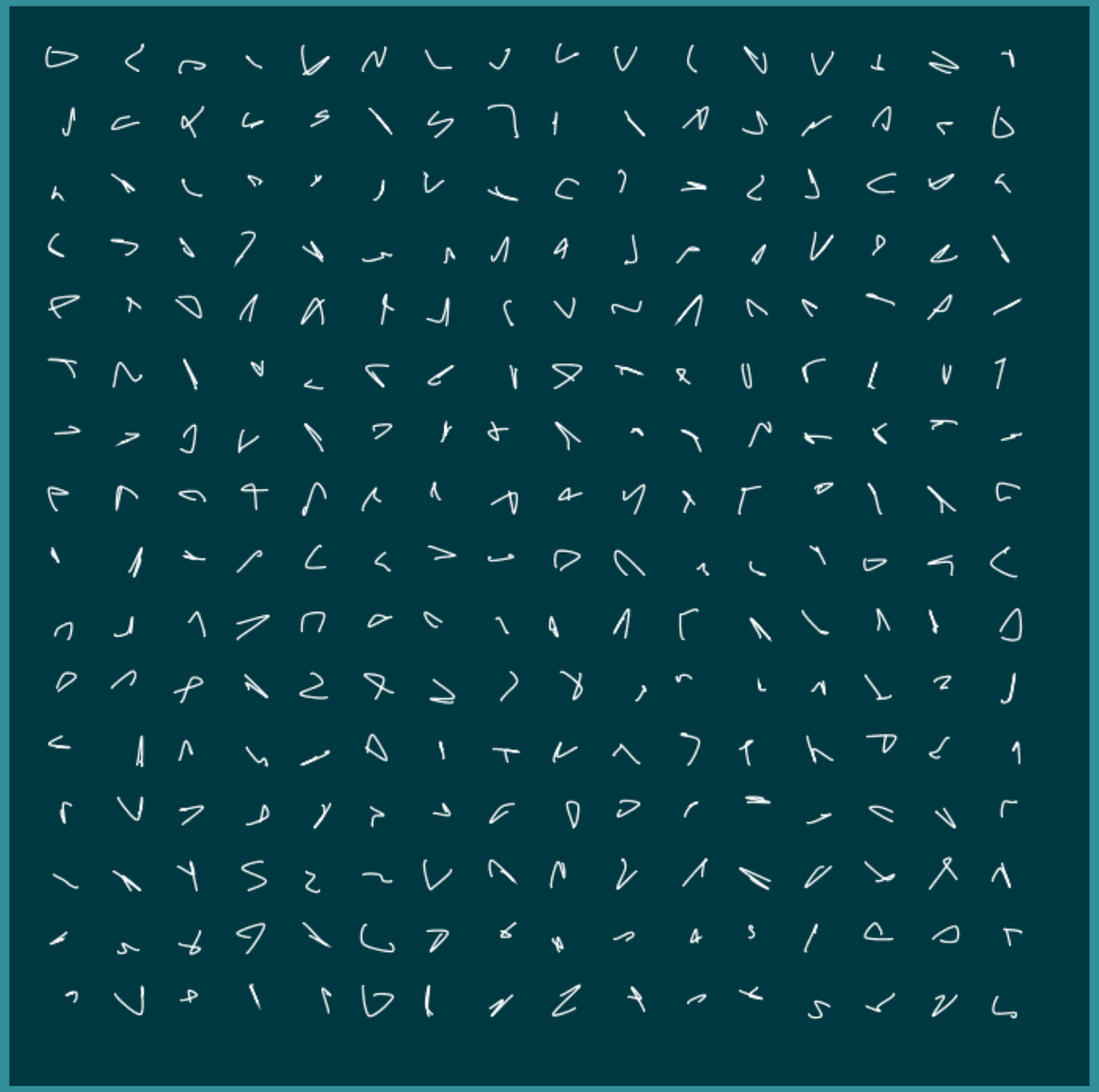
USING CURVEVERTEX()

```
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()

```
def setup():
    size(700, 700)
    background("#013840");


def draw():
    for i in range(40, width - 30, 40):
        for j in range(40, height - 30, 40):
            strokeWeight(1.3);
            stroke(255);
            noFill();
            beginShape();
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            endShape();
    noLoop();
```

```
def setup():
    size(700, 700)
    background("#013840");


def draw():
    for i in range(40, width - 30, 40):
        for j in range(40, height - 30, 40):
            strokeWeight(1.3);
            stroke(255);
            noFill();
            beginShape();
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            endShape();
    noLoop();
```



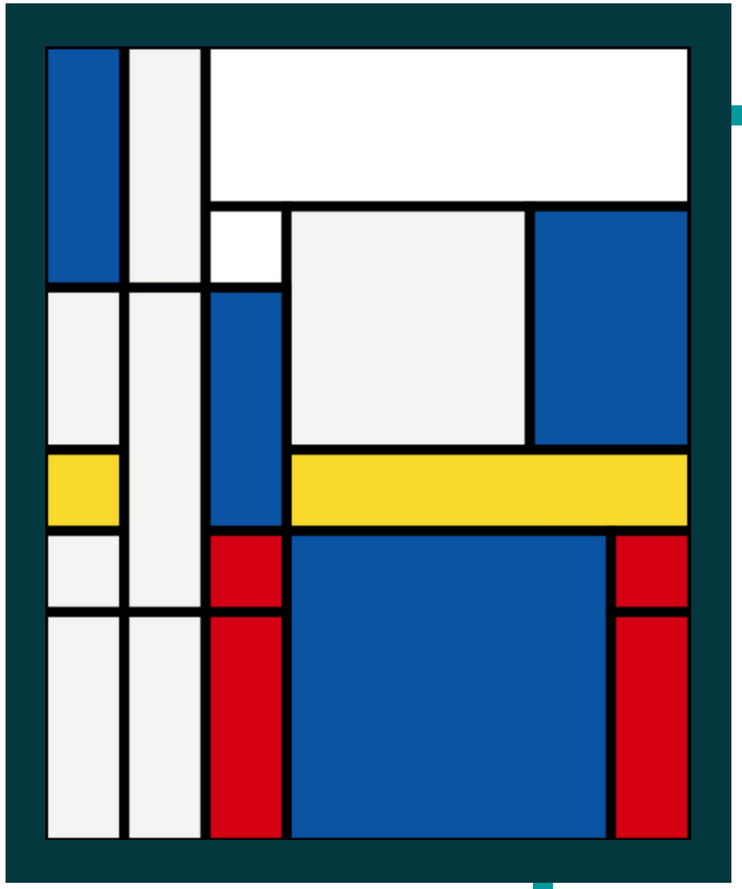
@NEERAJP99

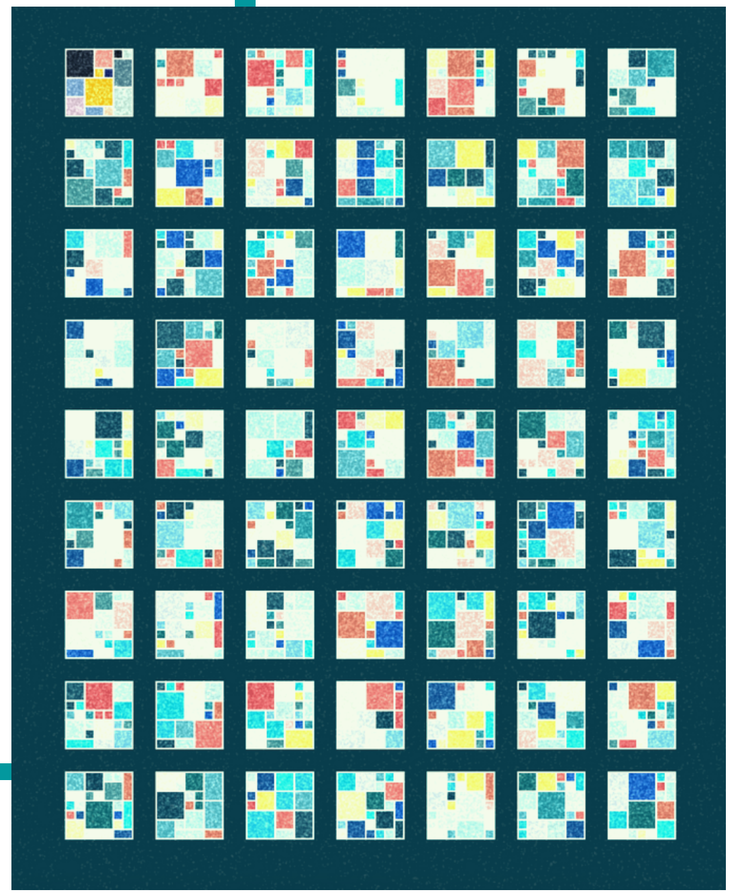
CREATING BASIC SHAPES

```
ellipse(a, b, c, d)
# Creates an ellipse at point a,b
# with "c" width and "d" height
rect(a, b, c, d, tl, tr, br, bl))
# Creates a rectangle at point a,b
# width width "c" and height "d"
square(x, y, c)
# Creates a square
```

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

Piet Mondrian Eperiments



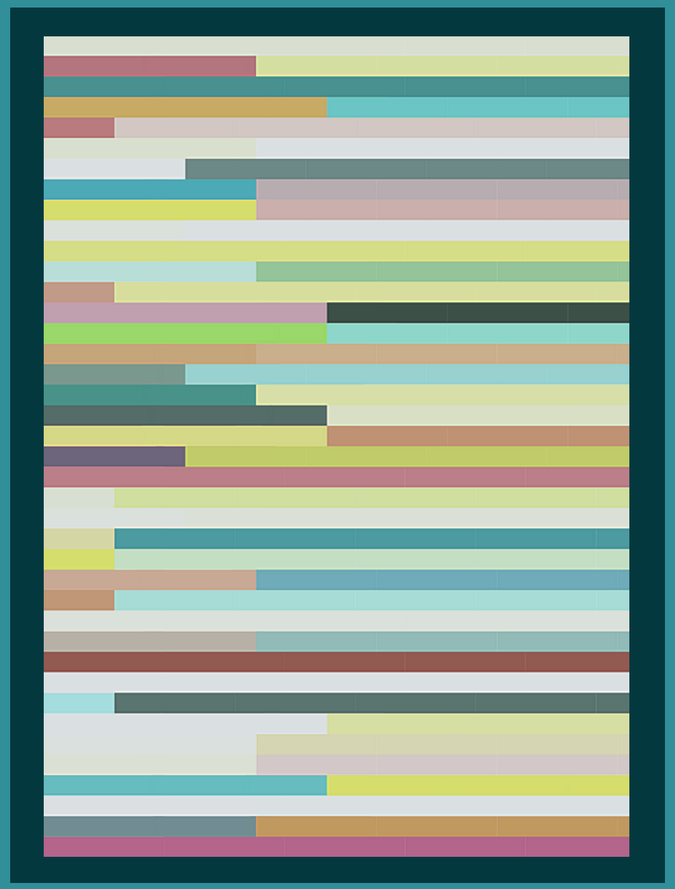


@NEERAJP99

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

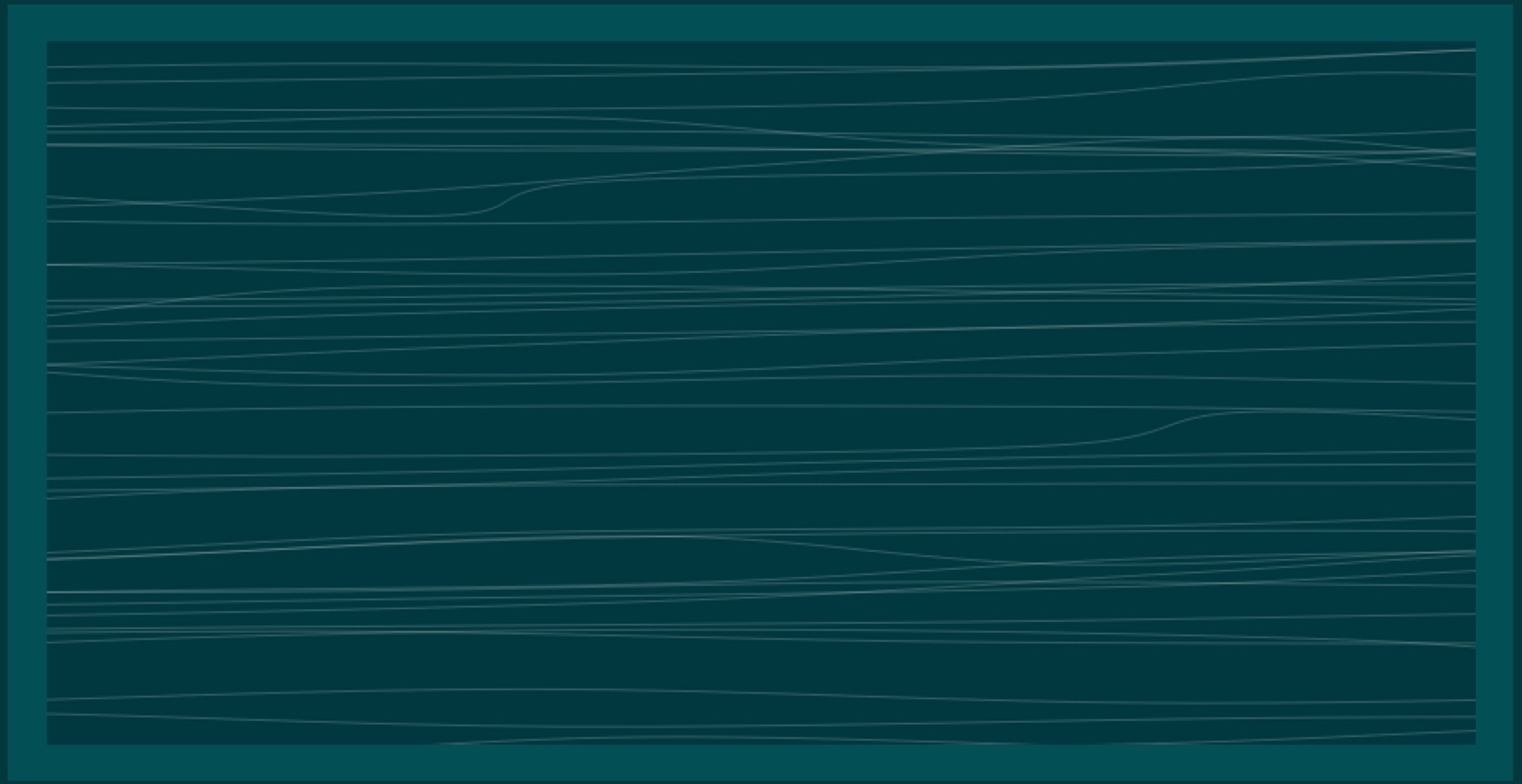
This function interpolates within the range [start..end] based on the amount parameter, where amount parameter is typically within a [0..1] range.

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

EXAMPLE USING LERPCOLOR()

```
def setup():
    size(600, 600)
    background(255)
    colorMode(HSB, 360, 100, 100);

def draw():
    initialColor = color(0,50,100);
    finalColor = color(45, 80, 100);
    noStroke()
    for i in range(width):
        linearIntValue = map(i, 0, width, 0, 1.0)
        # Calculates a color between two colors at a specific increment.
        linearIntColor = lerpColor(initialColor, finalColor, linearIntValue)
        fill(linearIntColor)
        rect(i, 0, 25, height)
    noLoop()
```

PERLIN NOISE / SIMPLEX NOISE



@NEERAJP99

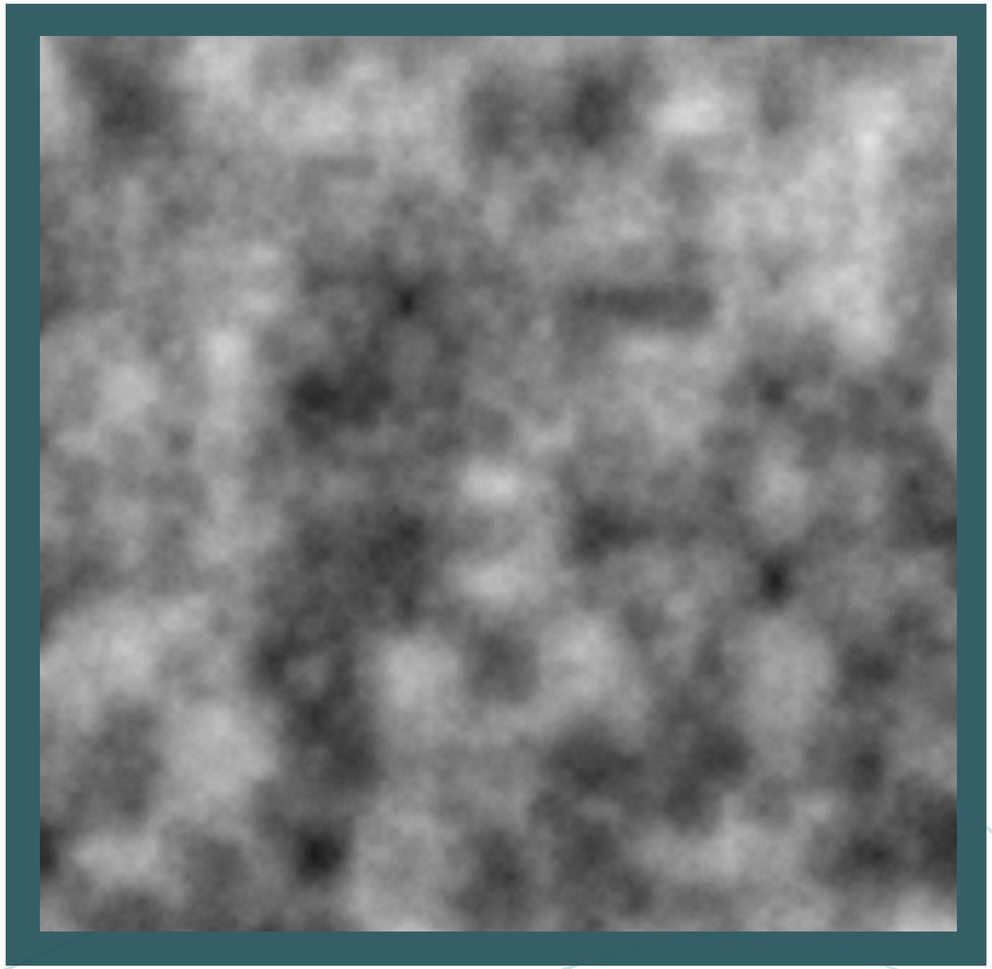
PERLIN NOISE / SIMPLEX NOISE

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

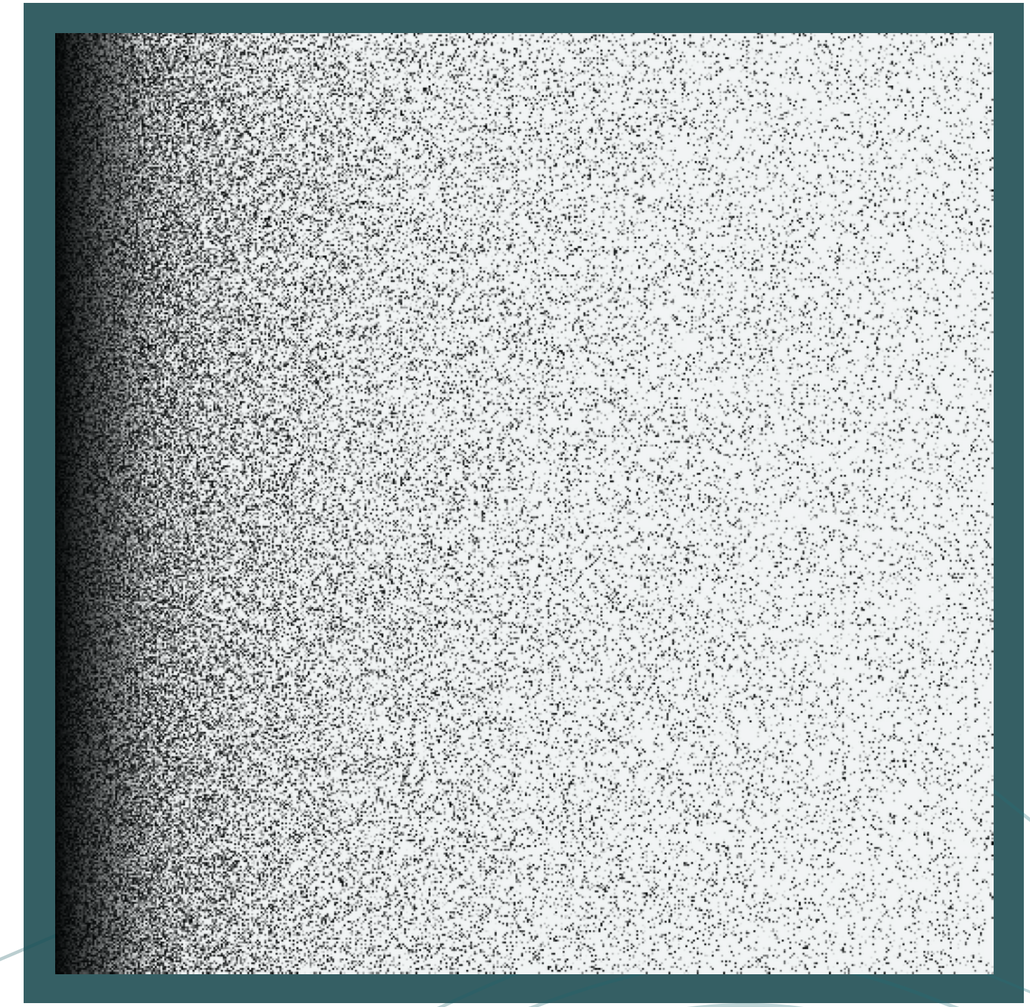
Without processing, use noise module in Python.
(https://pypi.org/project/noise/)

ADD NOISE TO THE PIXEL COLORS

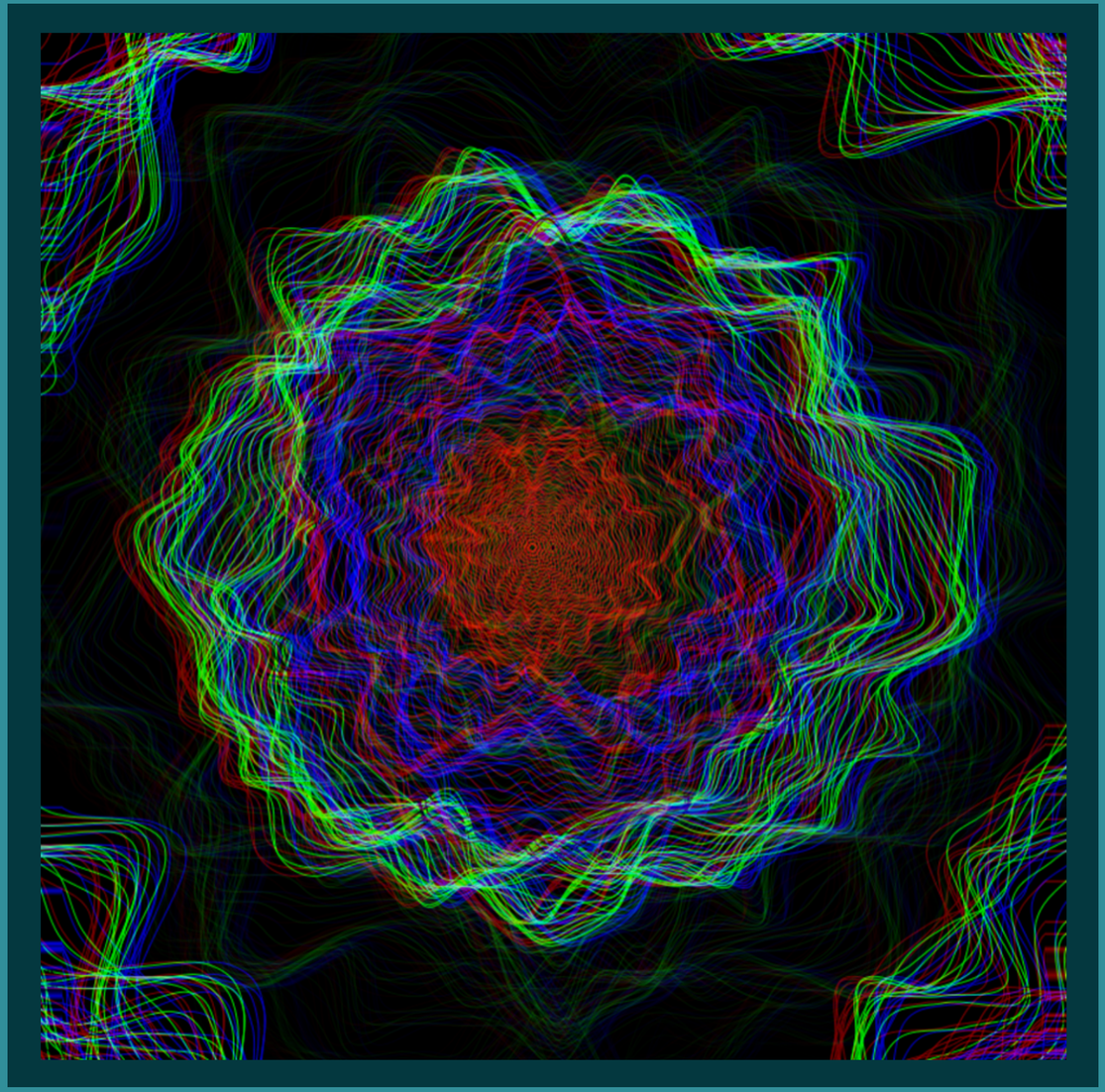
```
def setup():
    size(500, 500)
    pixelDensity(1)
    noiseDetail(40)

def draw():
    xoff = 0
    loadPixels()
    for x in range(0, width, 1):
        yoff = 0
        for y in range(0, height, 1):
            index = (x + y * width)
            r = noise(xoff, yoff) * 255
            if random(1) > 0:
                pixels[index] = color(r)
            yoff += 0.02
        xoff += 0.02
    updatePixels()
    noLoop()
```



ADD RANDOM TO THE PIXEL COLORS



USING NOISE AND TRIGONOMETRY



@NEERAJP99

USING NOISE AND TRIGONOMETRY

```
def draw():
  global rad, r, g, b, translateX, translateY, opacity, zoff
    noFill()
    stroke(r, g, b, opacity)
  beginShape()
    a = 0
    while a < TWO_PI:
        xoff = map(cos(a), -1, 1, 0, 10)
        yoff = map(sin(a), -1, 1, 0, 10)
        noiseFactor = map(noise(xoff, yoff, zoff), 0, 1, 100, 150)
        x = width / 2 + rad *  noiseFactor * cos(a)
        y = height / 2 + rad *  noiseFactor * sin(a)
        curveVertex(x, y)
        a += 0.1
    endShape(CLOSE)
    zoff += 0.1
    rad -= 0.02
```

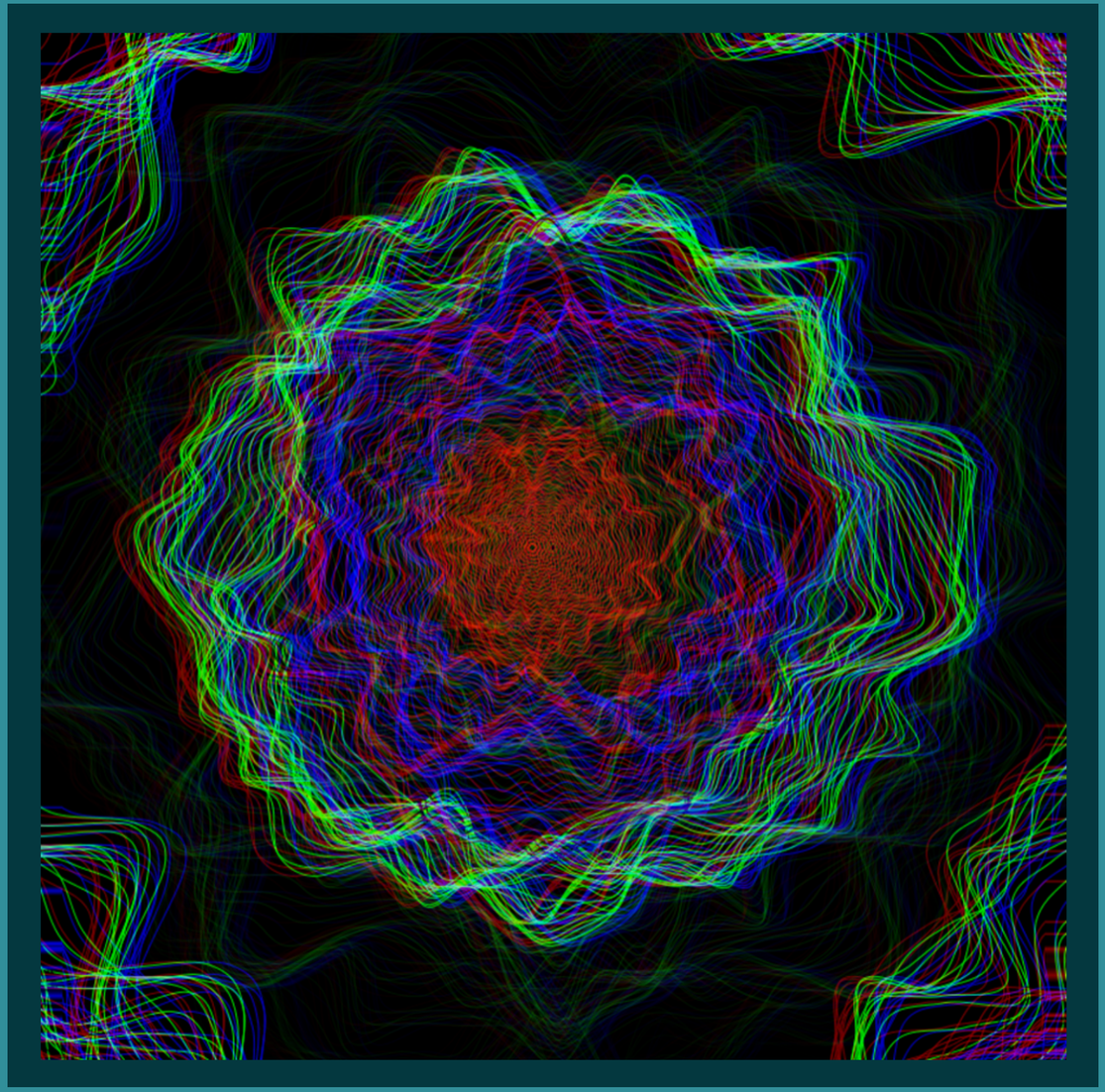
@NEERAJP99

USING NOISE AND TRIGNOMETRY

```
    if r > 255:
        r = 0
    if g > 255:
        g = 0
    if b > 255:
        b = 0
    if opacity == 0:
        opacity = 255
    opacity -= 1
    r += 1
    g += 1
    b += 1
```

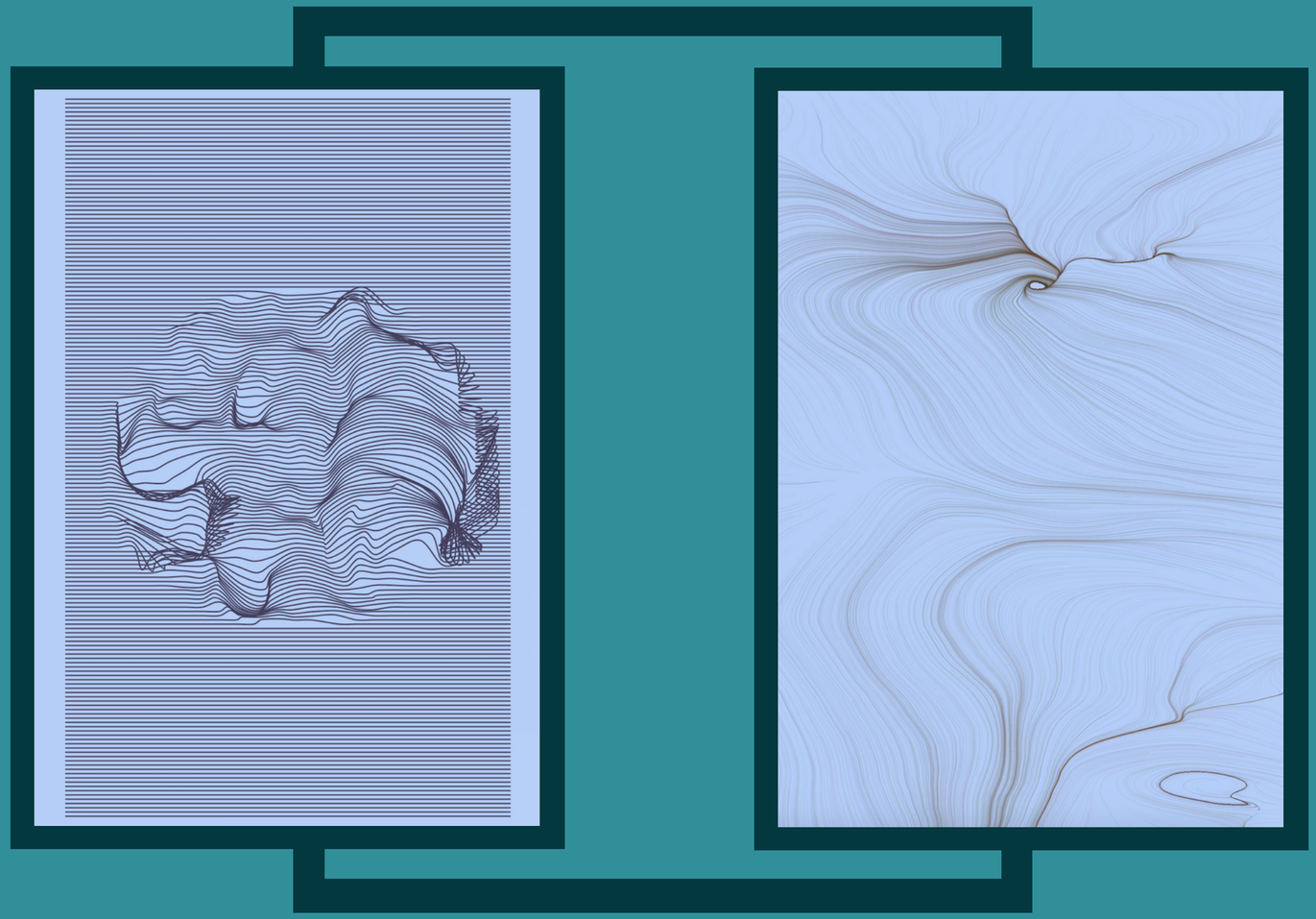
@NEERAJP99

USING NOISE AND TRIGONOMETRY

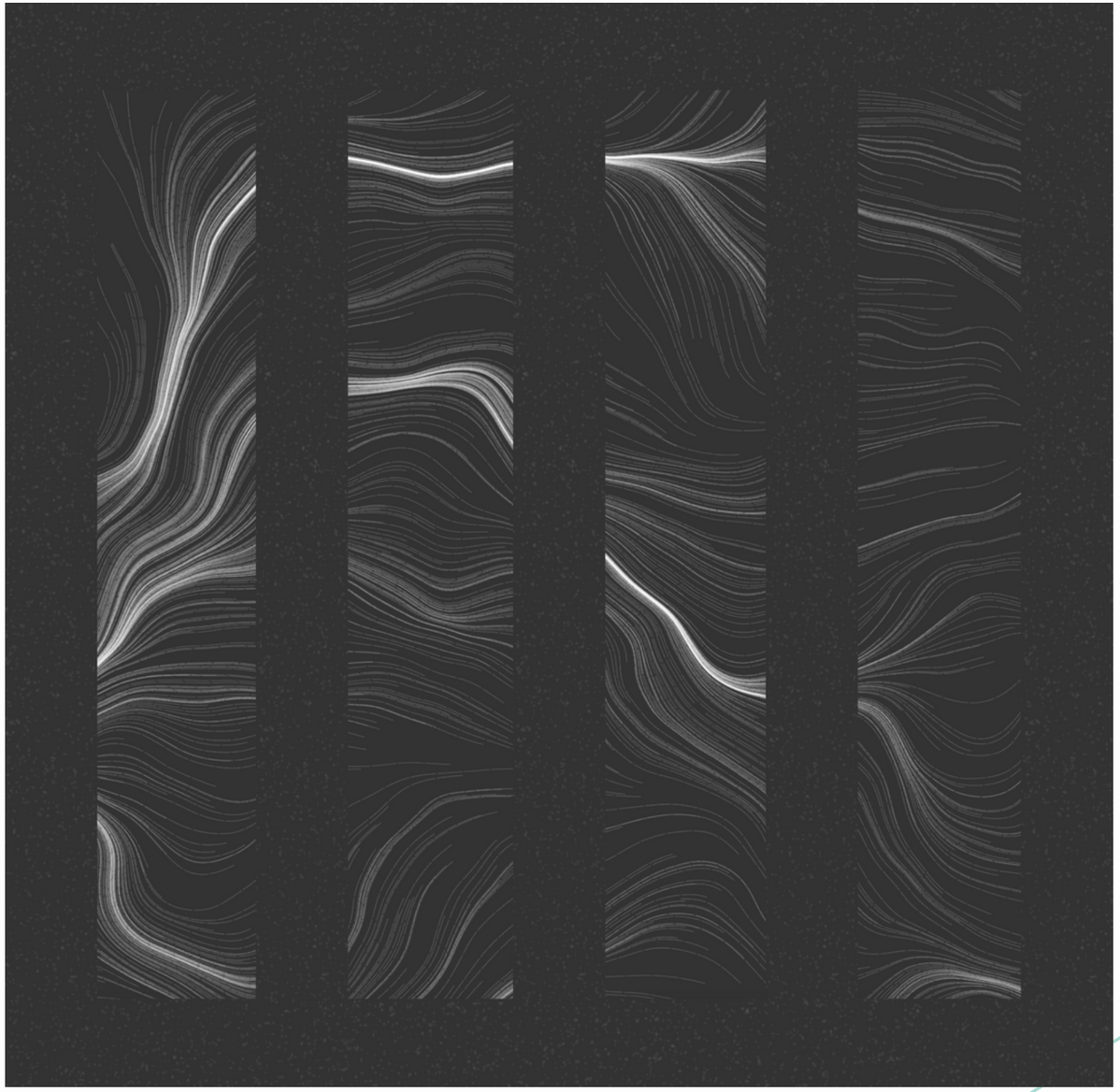


@NEERAJP99

Perlin Noise Field



@NEERAJP99

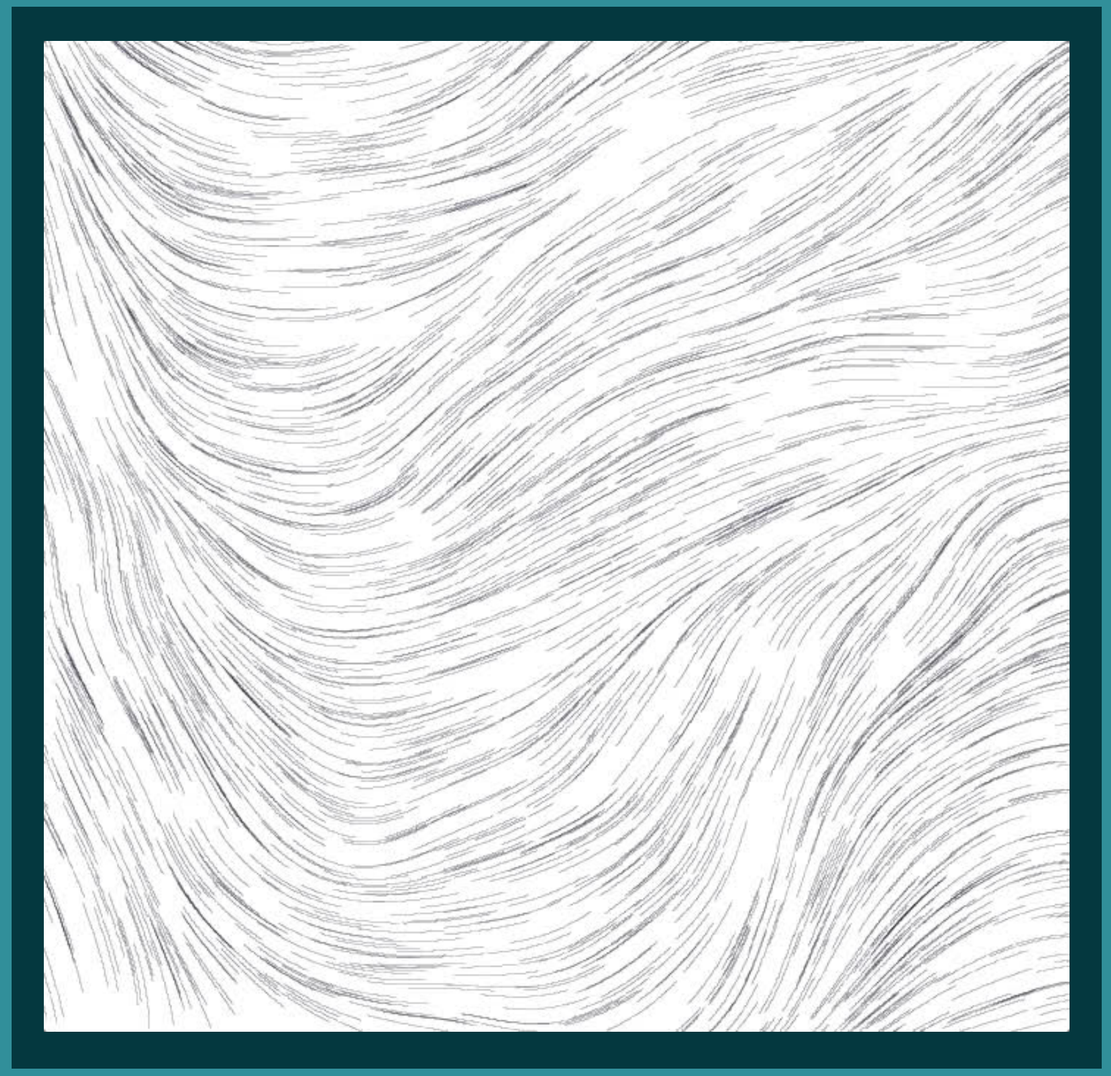


PERLIN NOISE

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

@NEERAJP99

PERLIN NOISE FIELD



@NEERAJP99

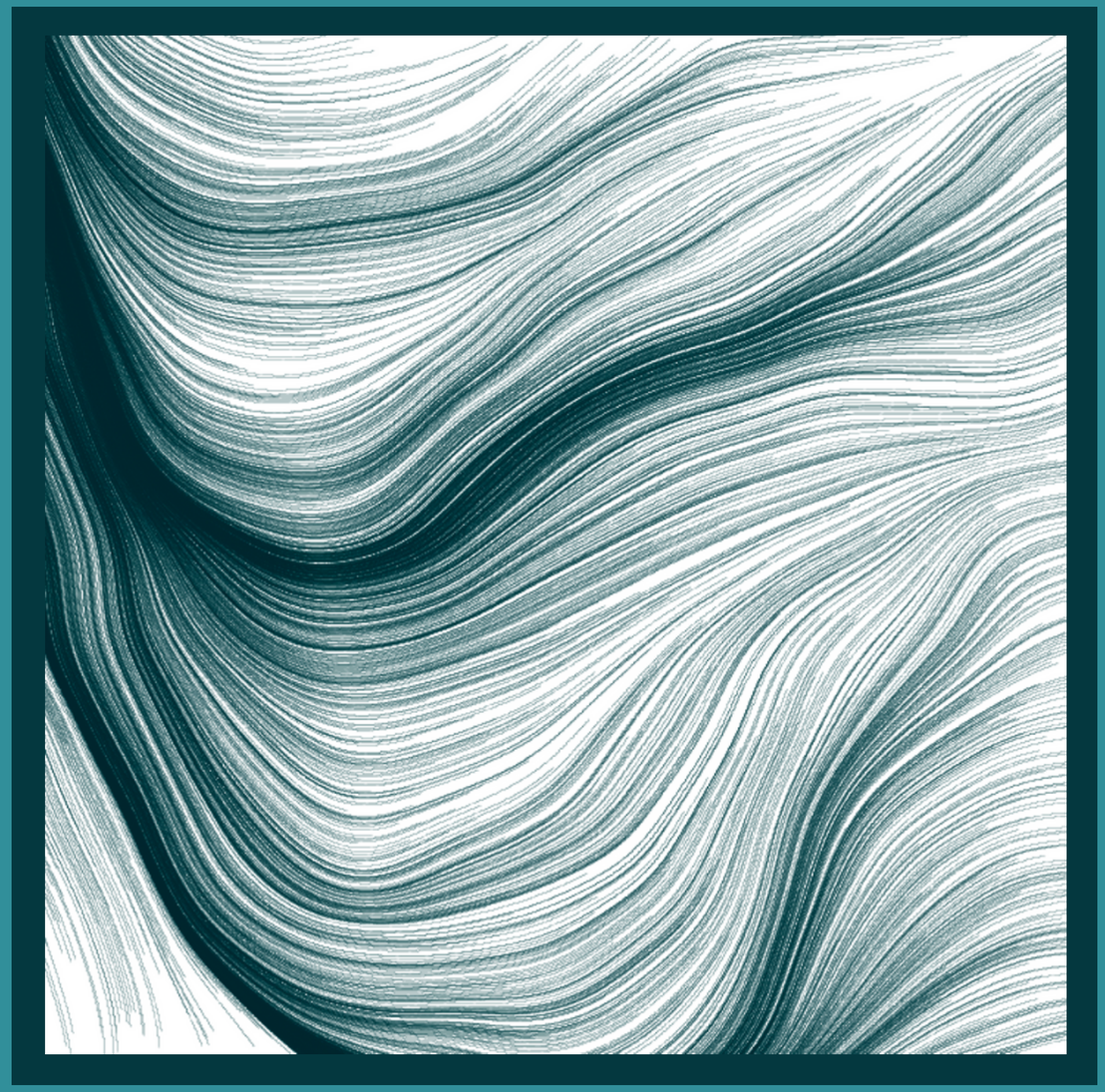
PERLIN NOISE FIELD

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

PERLIN NOISE FIELD

```
def draw():
    global points
    for i in range(len(points)):
        noFill()
        noiseSeed(2)
        vectorObject = points[i]
        stroke("#333");
        strokeWeight(0.3);
        beginShape()
        for i in range(20):
            # Generate noise values and map it between 0 and 2*PI radians
            noiseValue = map(
              noise(vectorObject.x / 500, vectorObject.y / 500),
              0,
              1,
              0,
              2*PI
            )
            x1 = vectorObject.x
            y1 = vectorObject.y
            x2 = x1 + cos(noiseValue)
            y2 = y1 + sin(noiseValue)
            vertex(x1, y1)
            vectorObject.x = x2
            vectorObject.y = y2
        endShape(OPEN)
```

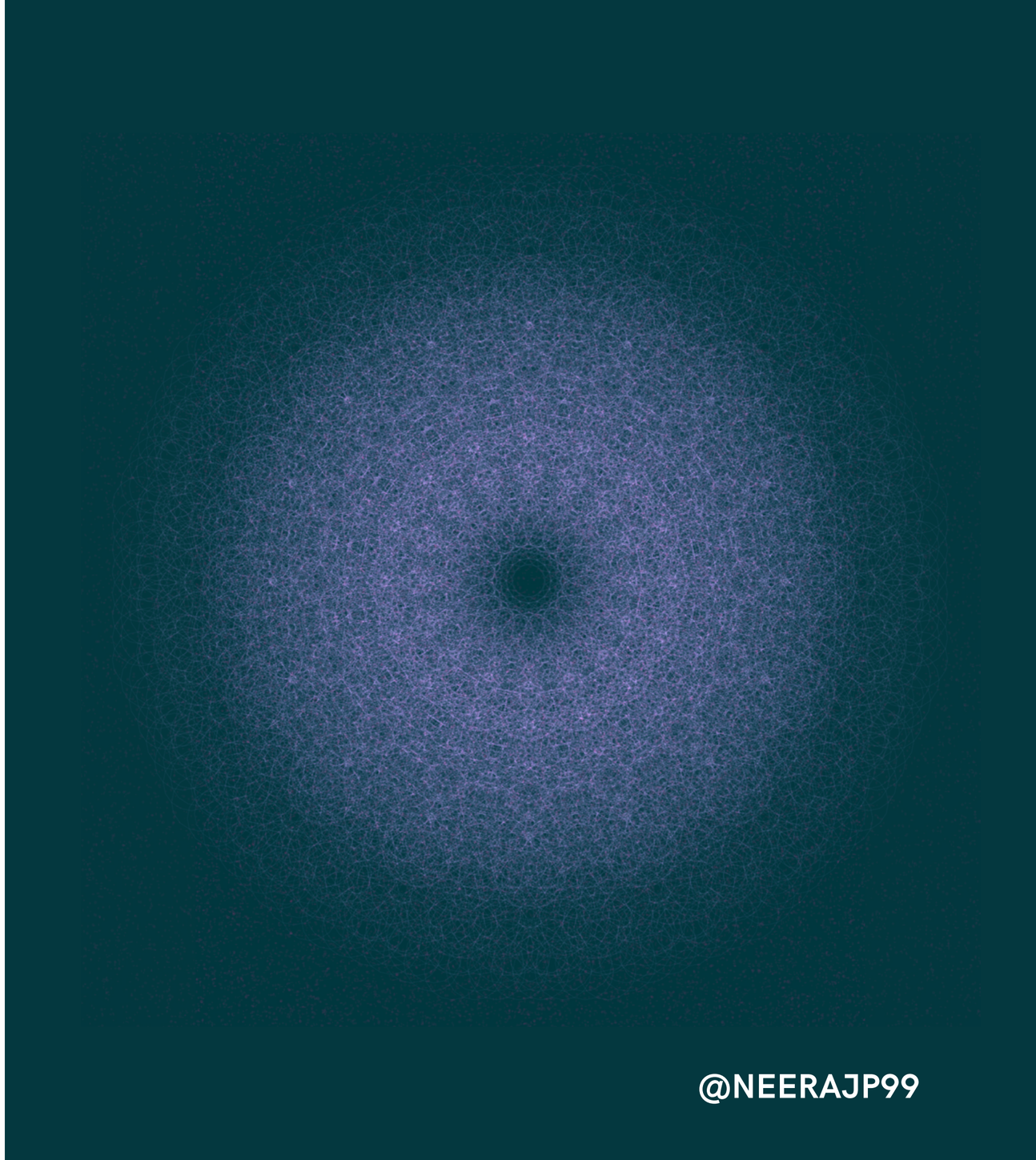
PERLIN NOISE FIELD



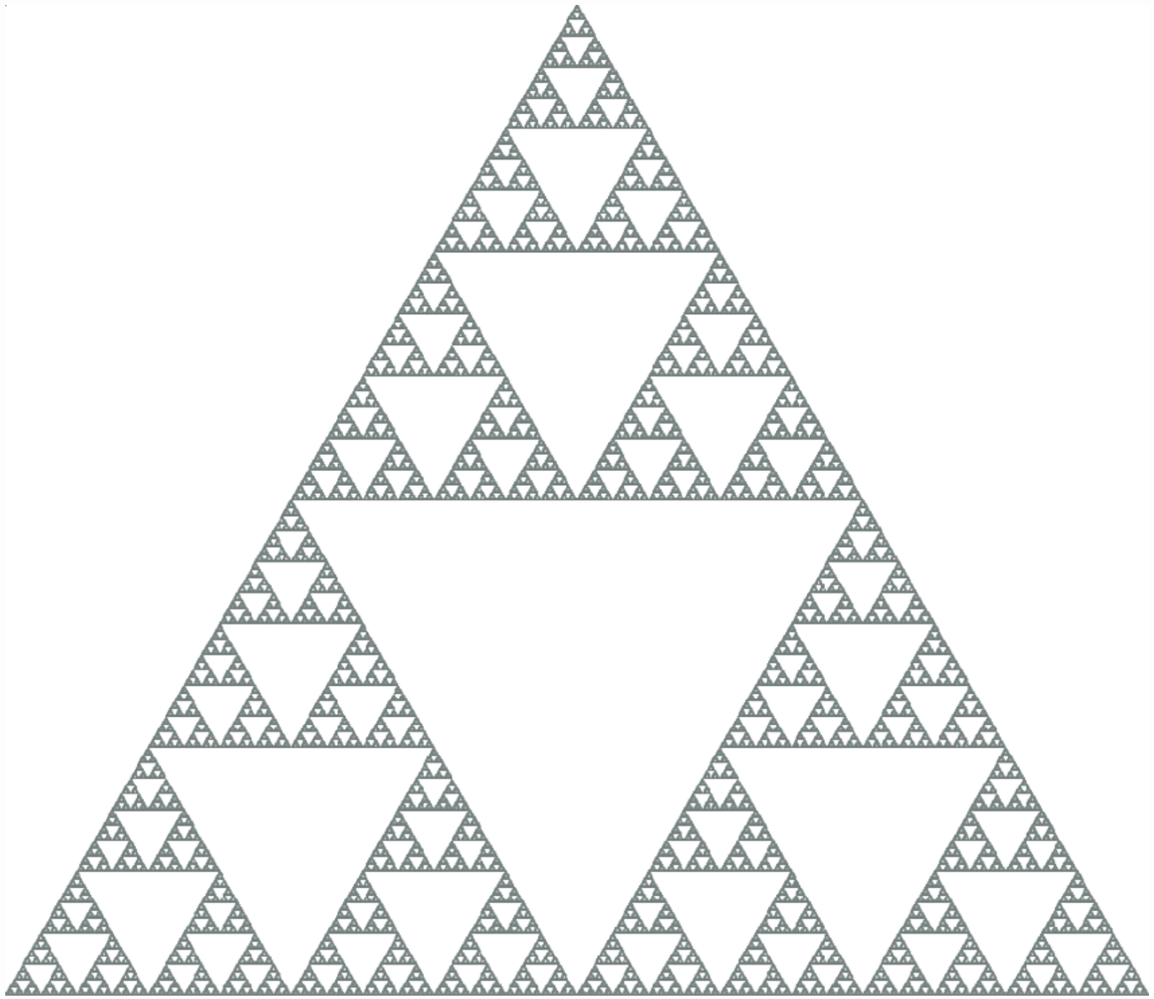
@NEERAJP99

GEOMETRY FRACTALS CHAOS

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



@NEERAJP99

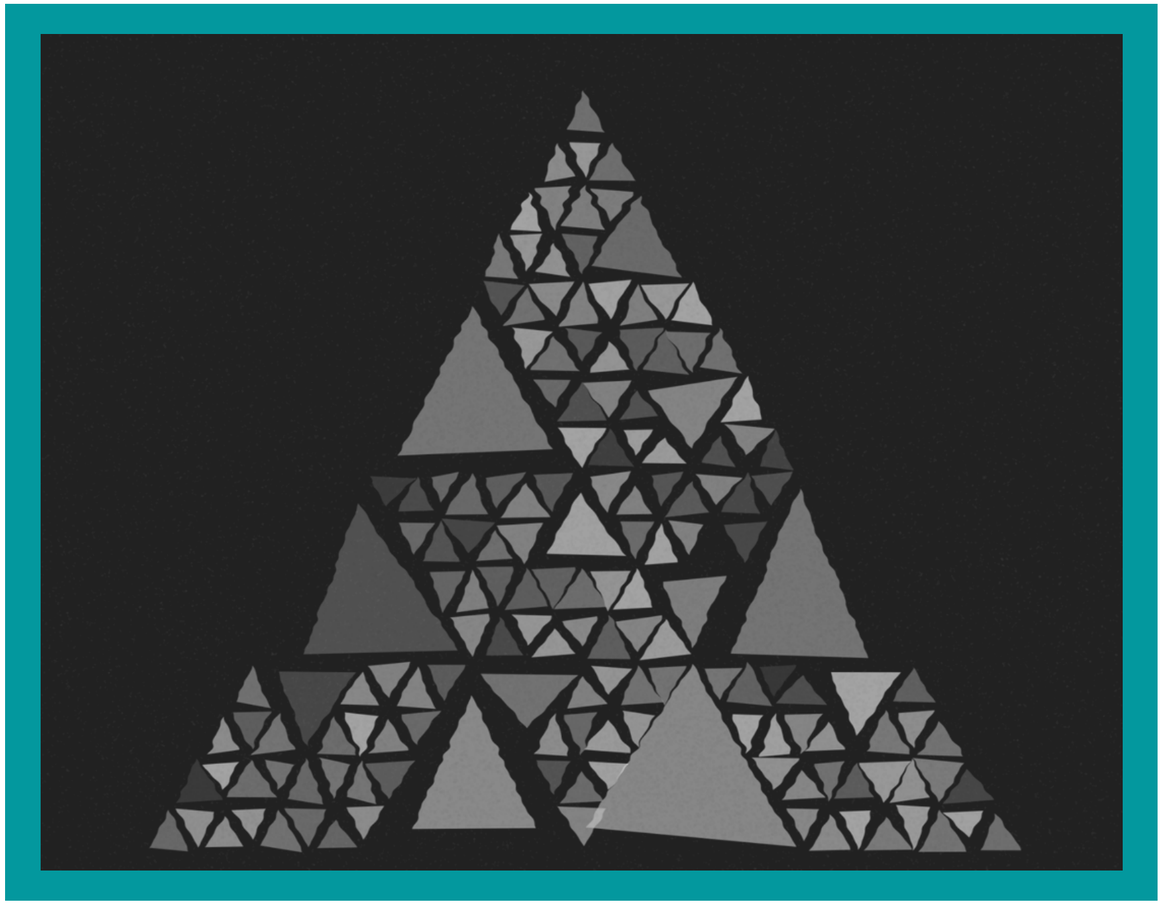


SIERPINSKI TRIANGLE

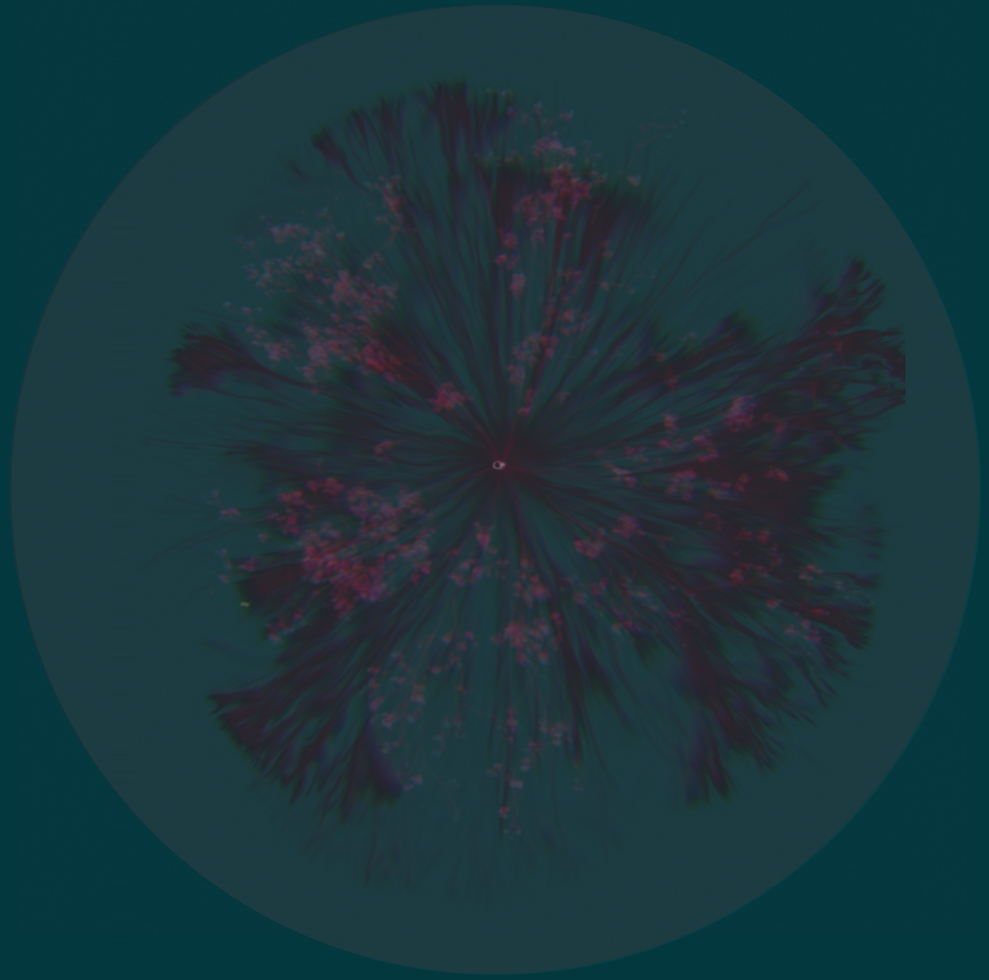
An equilateral triangle, subdivided recursively into smaller equilateral triangles with one recursive call each time.

@NEERAJP99

Modified Sierpinski Triangle

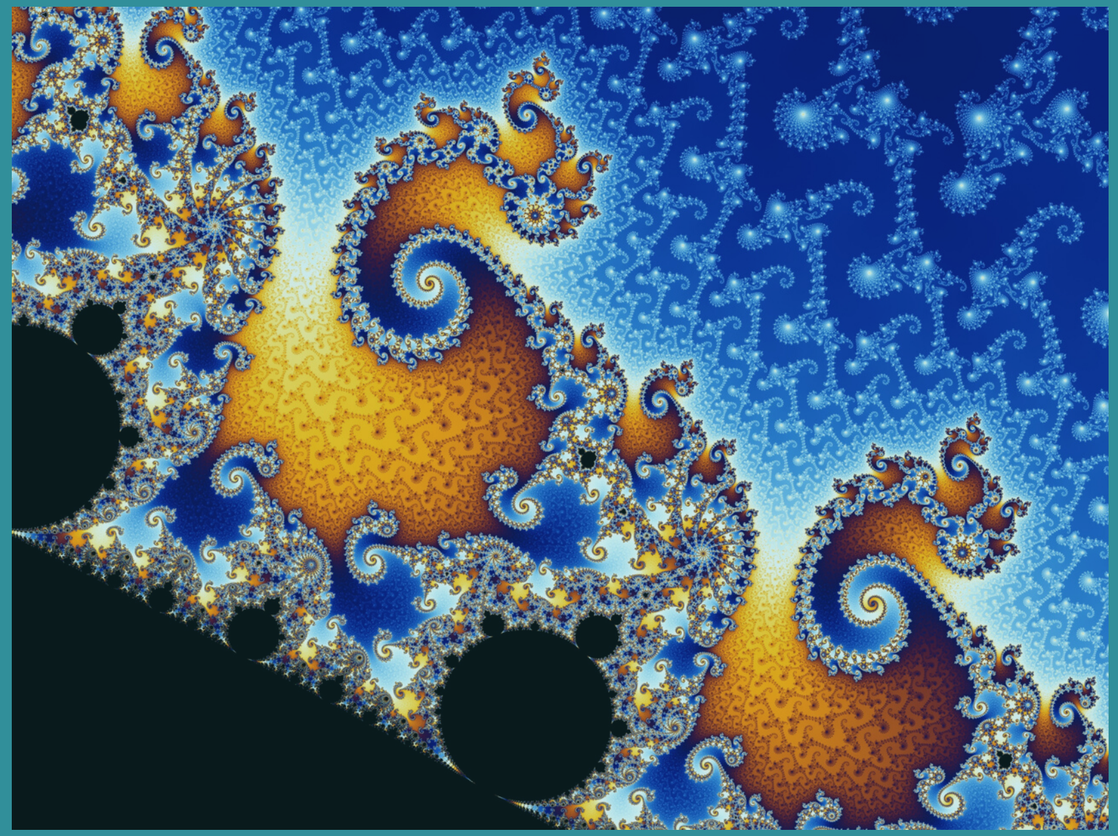


Fractal Flower



@NEERAJP99

Mandelbrot Set



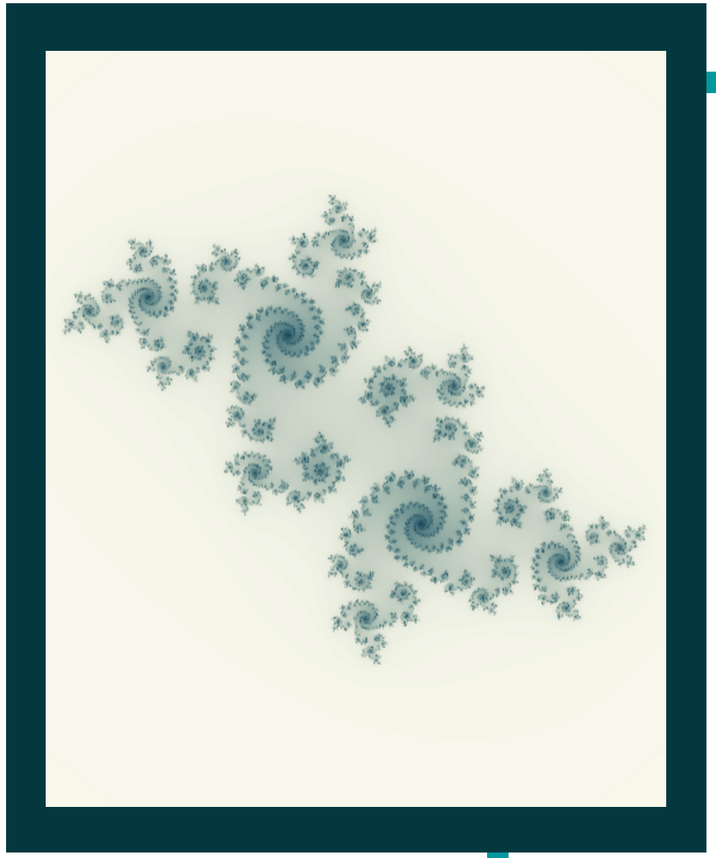
A geometrical figure where each part has the same statistical characters.

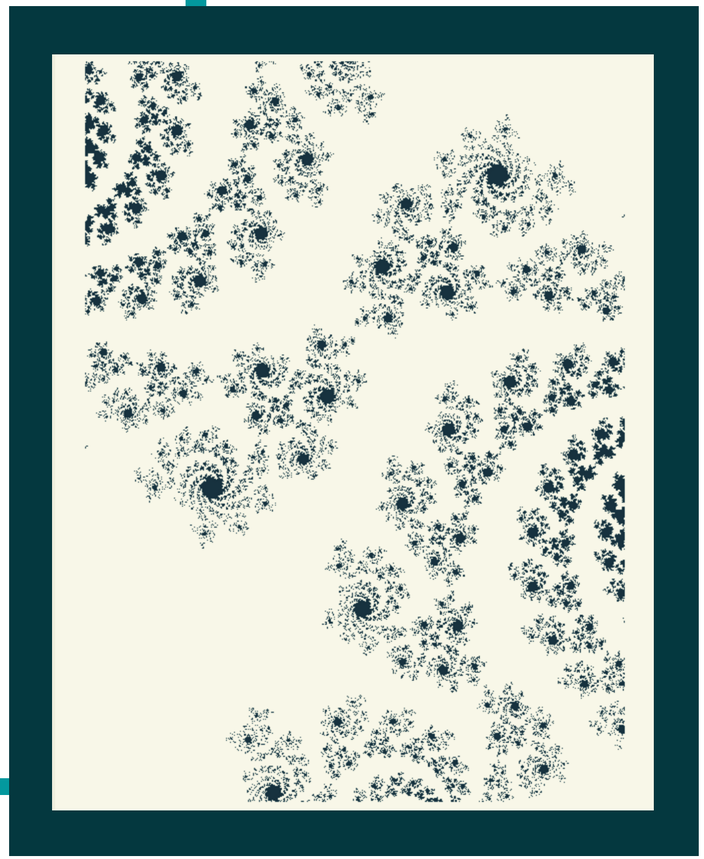
$$z = x + yi$$

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

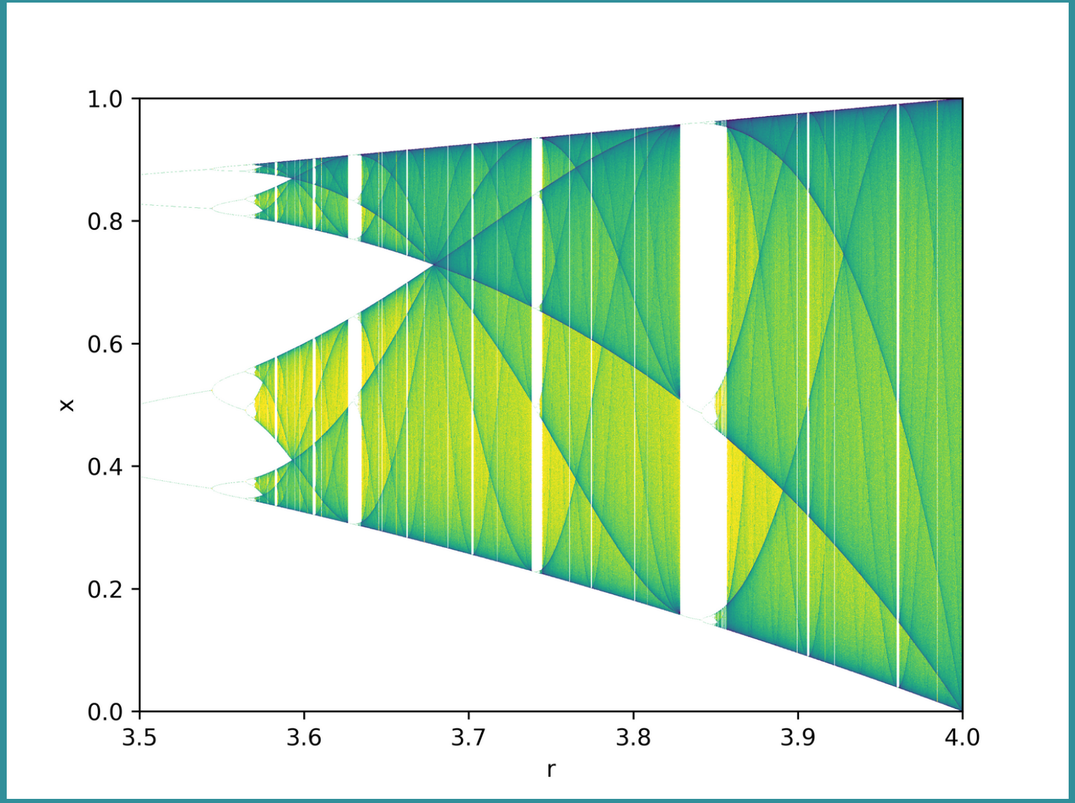
Source: Wikipedia

Julia Set





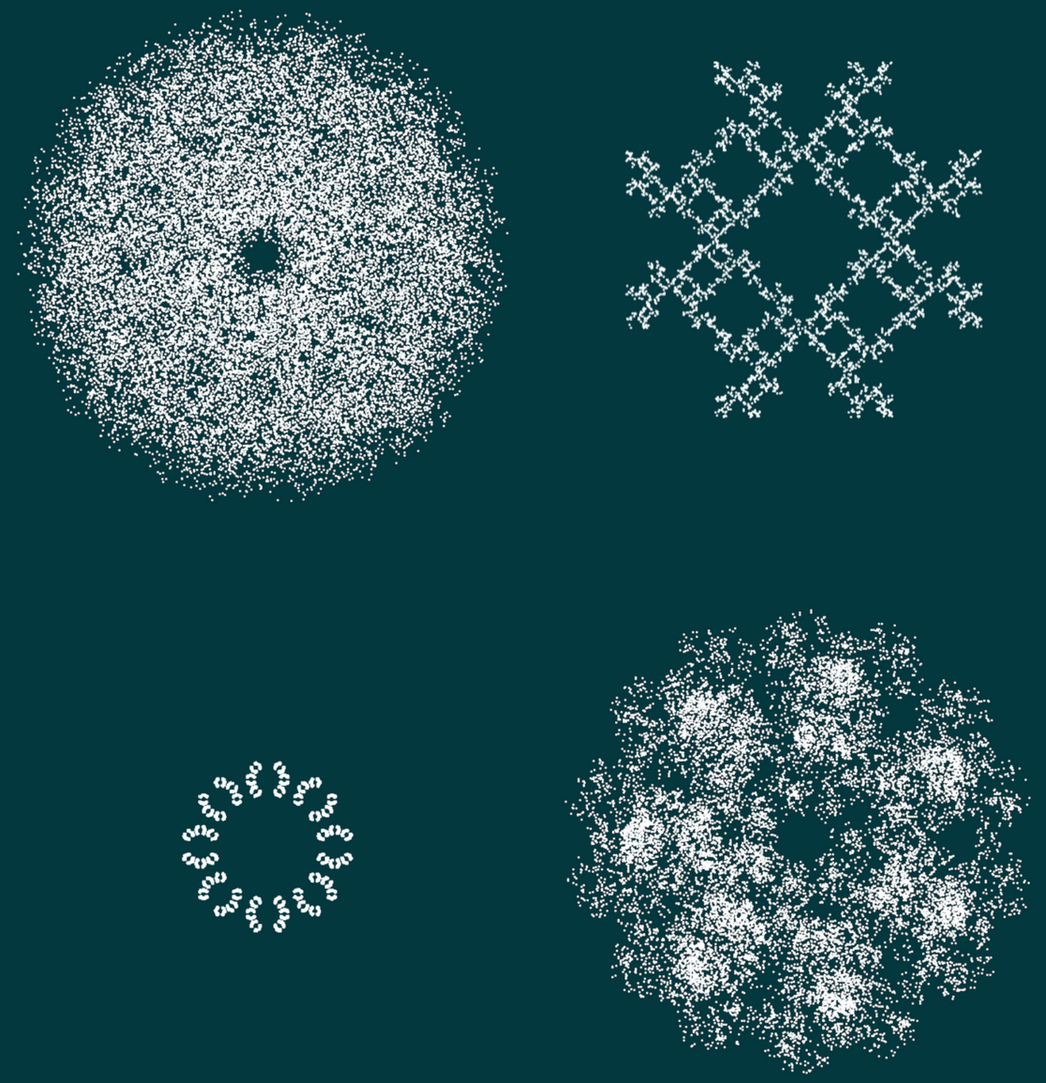
The Logistic Map



Source: Wikimedia

CHAOS THEORY

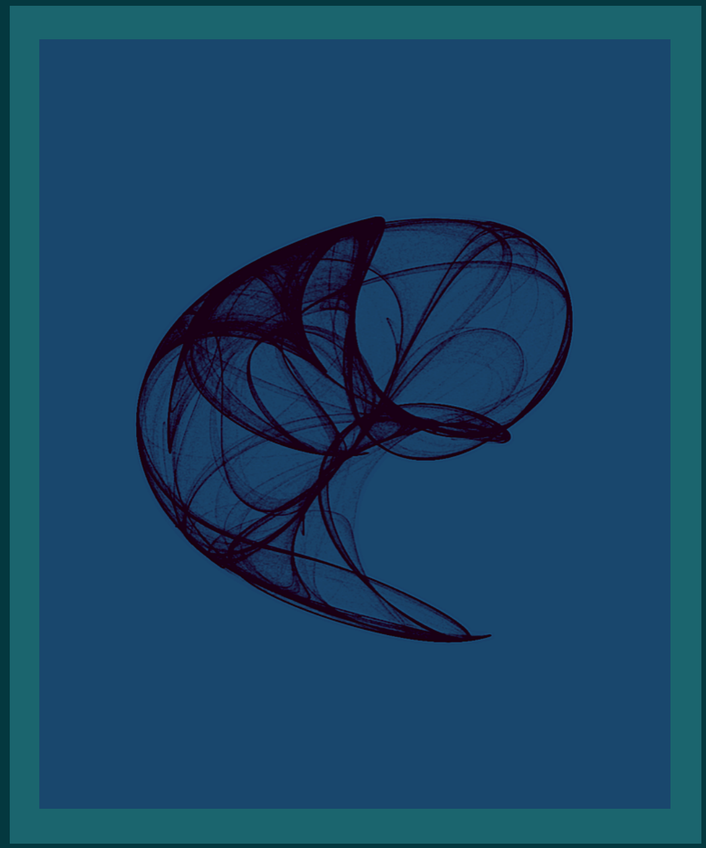
Deterministic, unpredictable
A small change in the initial state can result
in very large difference in the final outcome



@NEERAJP99

Attractors

Lorenz system



De Jong Attractor

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

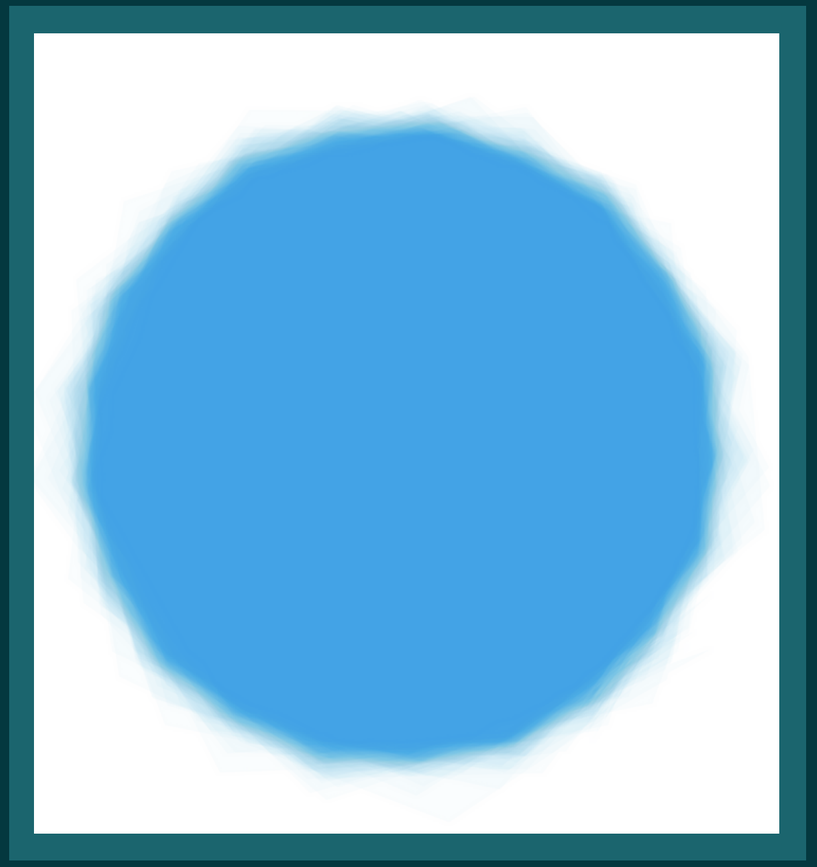
https://examples.pyviz.org/attractors/attractors.html

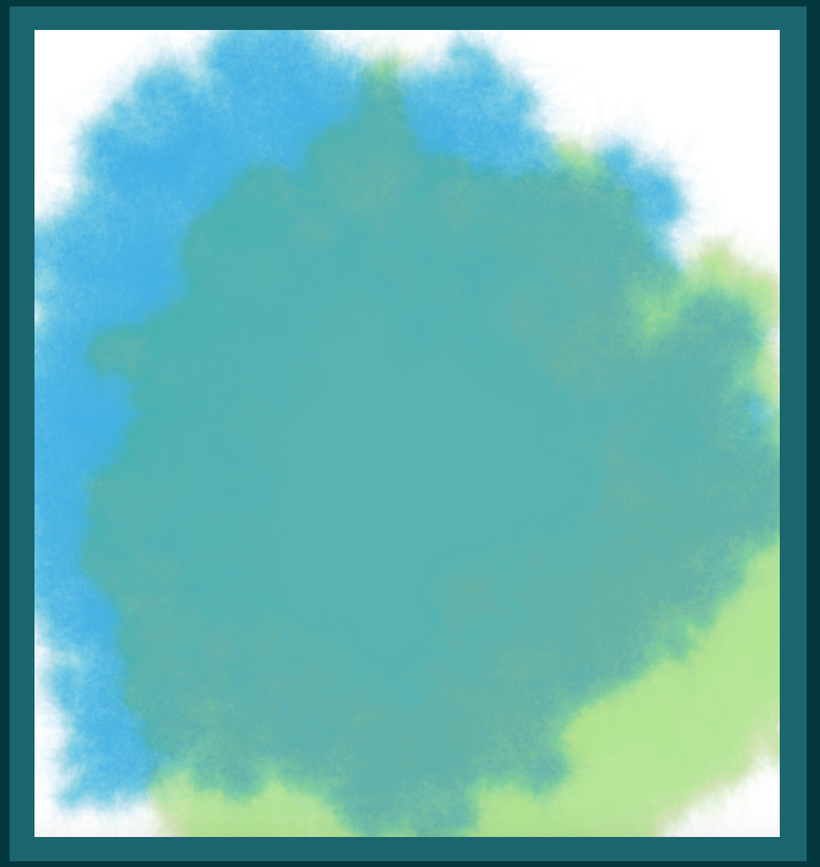
@NEERAJP99

SIMULATING PAINT

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

@NEERAJP99





CREATE POLYGON, DIVIDE SIDES

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

@NEERAJP99

FUNCTIONS TO DIVIDE THE SIDE USING RANDOM GAUSSIAN

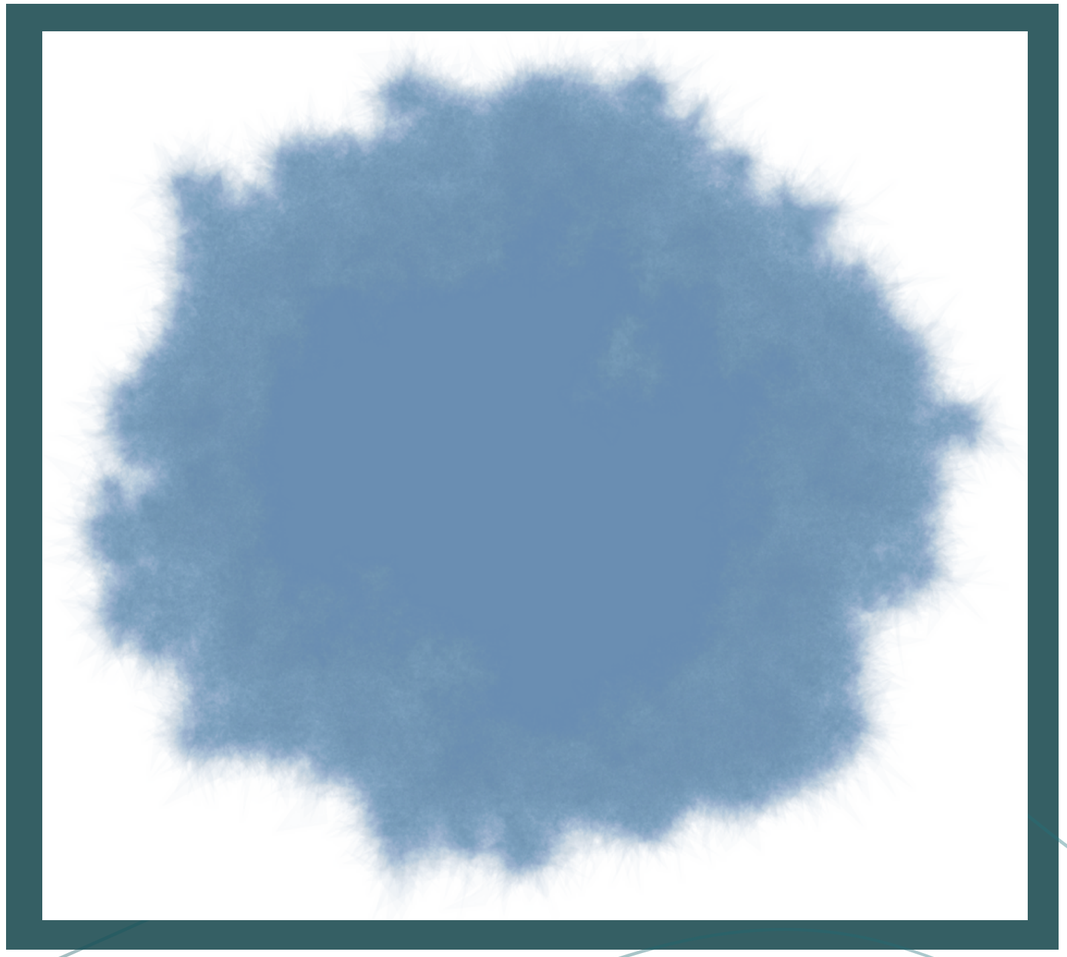
```python
def getCenterPoint(previous, current, standardDeviation):
        sideX = random_gauss(previous["sideX"] +
                             (current["sideX"] -
                              previous["sideX"]) / 2,
                             standardDeviation)
        sideY = random_gauss(previous["sideY"] +
                             (current["sideY"] -
                              previous["sideY"]) / 2,
                             standardDeviation)
        return { "sideX": sideX, "sideY": sideY }

def getMid(mean, standardDeviation):
        newVector = [mean[0]]
        for i in range(1, len(mean), 1):
            previousValue = mean[i - 1]
            currentValue = mean[i]
            midPoint = getCenterPoint(previousValue, currentValue, standardDeviation)
            newVector.append(previousValue)
            newVector.append(midPoint)
        newVector.append(mean[len(mean) - 1])
        return newVector
```

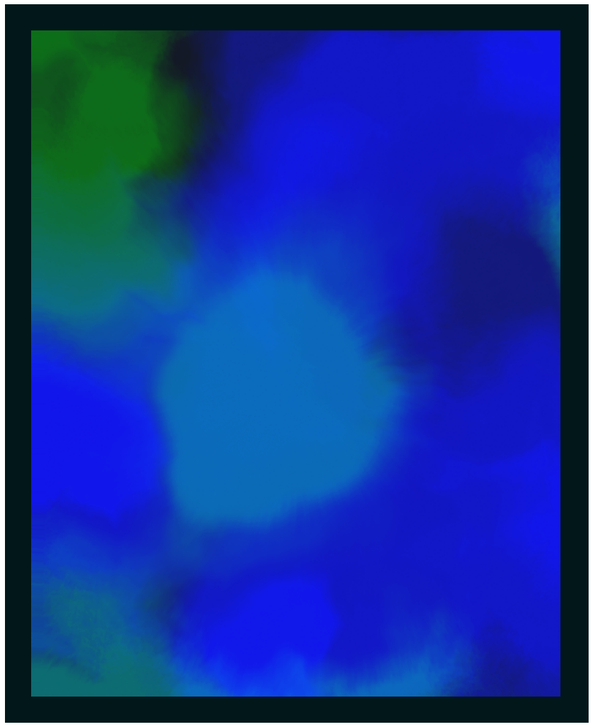
ADD VERTEX TO THE NEW SIDES

```
def draw():
    global polygon, randomGauss
    for i in range(20):
        push()
        noStroke()
        fill(200, 100, 80, 0.02)
        beginShape()
        translate(width * 0.5, height * 0.5);
        for j in range(len(polygon)):
            currentVector = polygon[j]
            x = random_gauss(currentVector['sideX'], random(25))
            y = random_gauss(currentVector['sideY'], random(25))
            vertex(x, y)
        endShape(CLOSE)
        pop()
    noLoop()
```

ADD A CUSTOM RANDOM GAUSSIAN METHOD

```python
# Custom function to return a random number fitting a Gaussian distribution.
def random_gauss(m, sd):
    value = 1
    while value >= 1:
        x1 = random(2) - 1
        x2 = random(2) - 1
        value = x1 * x1 + x2 * x2
    value = math.sqrt(-2 * math.log(value) / value)
    y1 = x1 * value
    y2 = x2 * value
    mean = m or 0
    standardDeviation = sd or 1
    return y1 * standardDeviation + mean
```

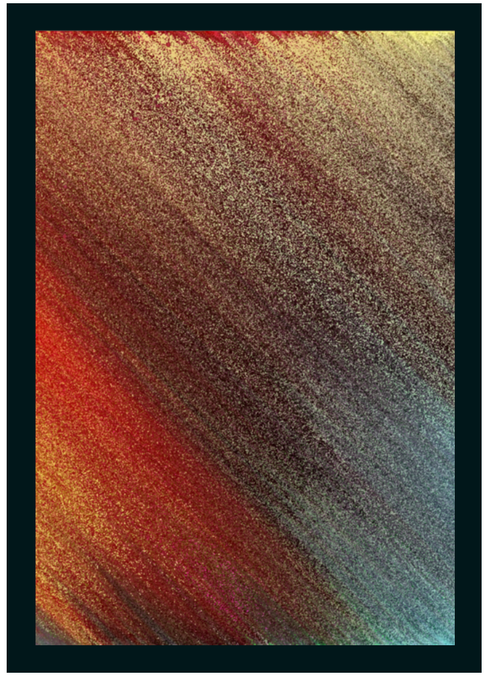
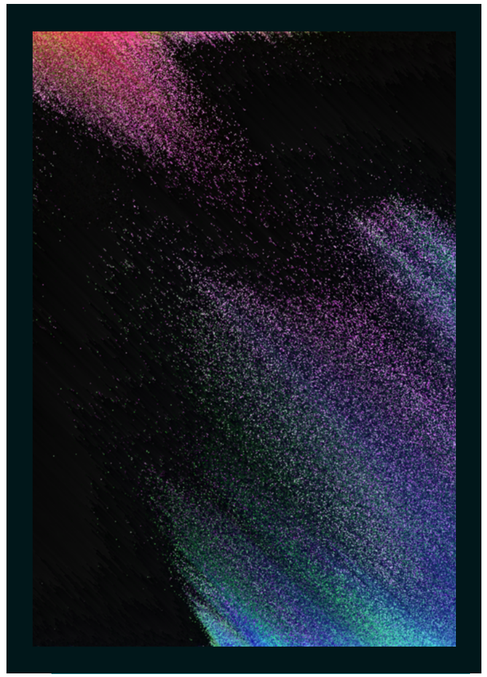
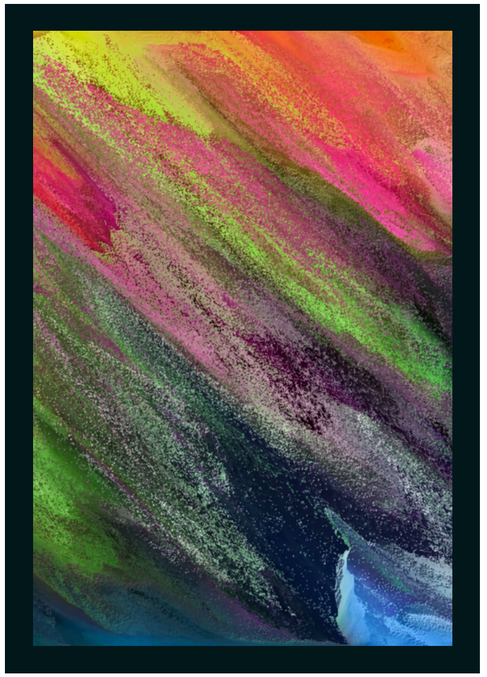
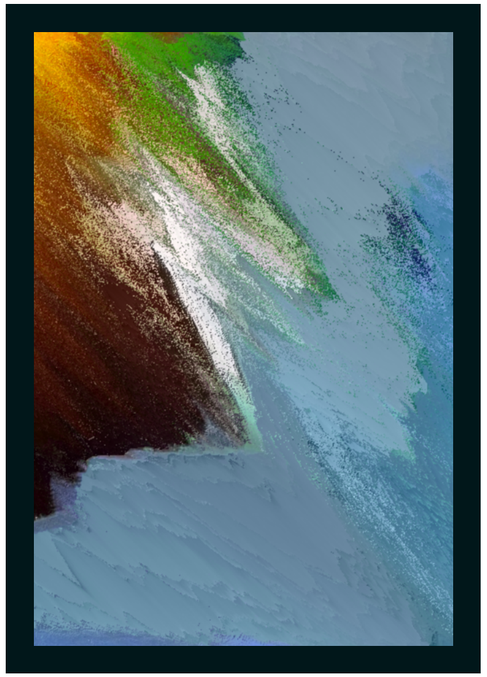




Inspiration: https://tylerxhobbs.com/essays/2017/a-generative-approach-to-simulating-watercolor-paints

@NEERAJP99

Pixel Sorting Algorithms v1.0

@NEERAJP99

SIMPLE IMPLEMENTATION OF SORTING PIXELS

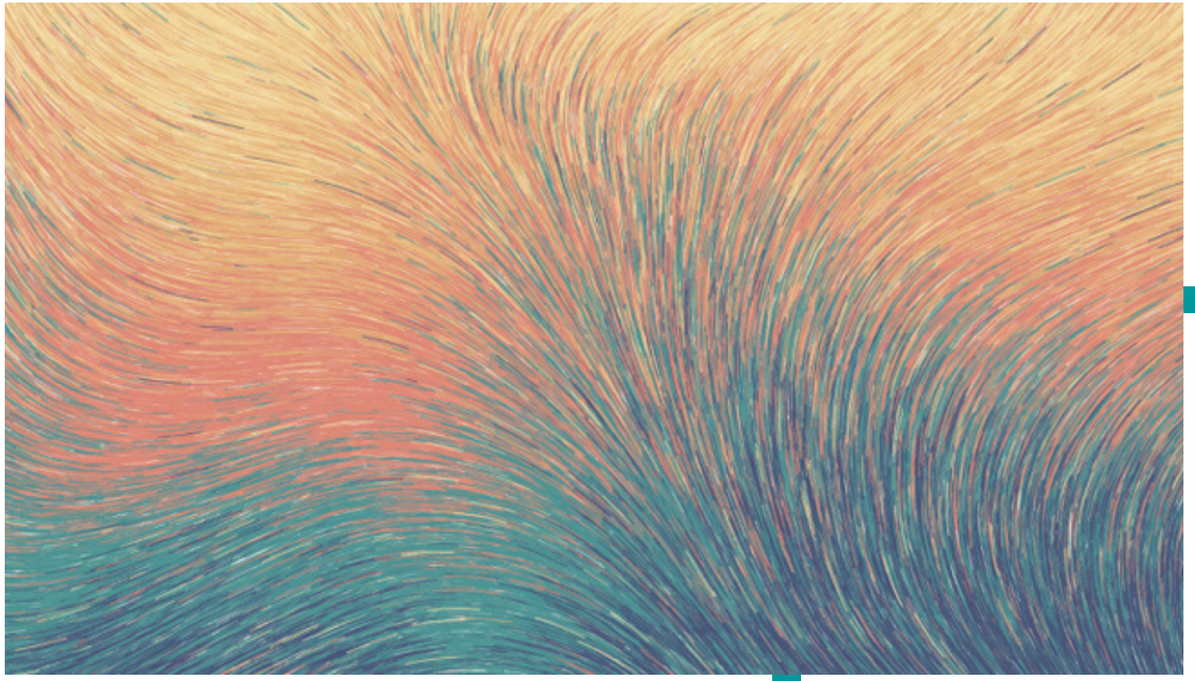
```
def draw():
    global img, imgNew
    image(img, 0, 0)
    pixelArray = img.pixels
    # Looping through rows and columns
    for x in range(img.height):
        newPixelArray = list()
        for y in range(img.width):
            # Get pixel positions
            pixel = img.get(y, x)
            # Get the pixel hex value and convert it into rgb
            rgbPixel = convert_to_rgb(str(pixel))
            rgbPixel.append(255)
            newPixelArray.append(rgbPixel)
        # Sort the new pixel array
        for j in range(len(newPixelArray) - 2):
            # for j in range (len(newPixelArray) - 2):
            if ((((newPixelArray[j][0] + newPixelArray[j][1] + newPixelArray[j][2])) >
                 (newPixelArray[j+1][0] + newPixelArray[j+1][1] + newPixelArray[j+1][2]))):
                newPixelArray[j], newPixelArray[j+1] = newPixelArray[j+1], newPixelArray[j]
        # Set the image pixels to the second image
        for i in range(img.width):
            a = newPixelArray[i][0]
            b = newPixelArray[i][1]
            c = newPixelArray[i][2]
            cl = color(a, b, c)
            imgNew.set(i, x, cl)
    imgNew.updatePixels();
    image(imgNew,0,0);
    noLoop()
```

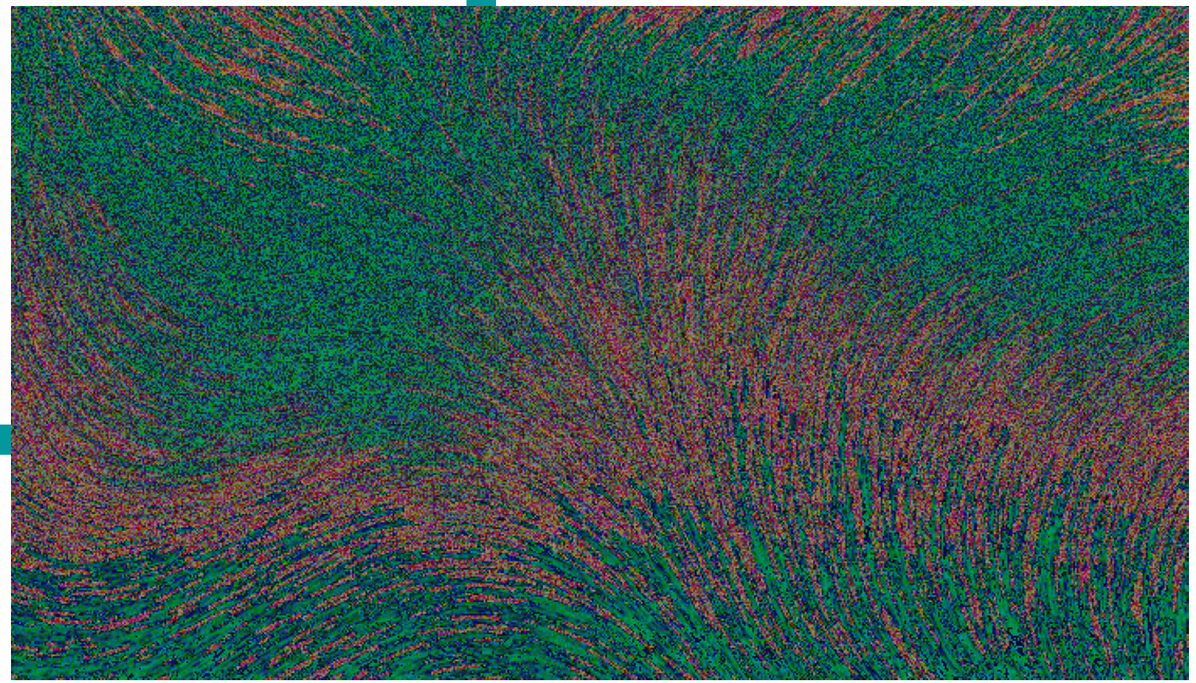
@NEERAJP99

SIMPLE IMPLEMENTATION OF SORTING PIXELS

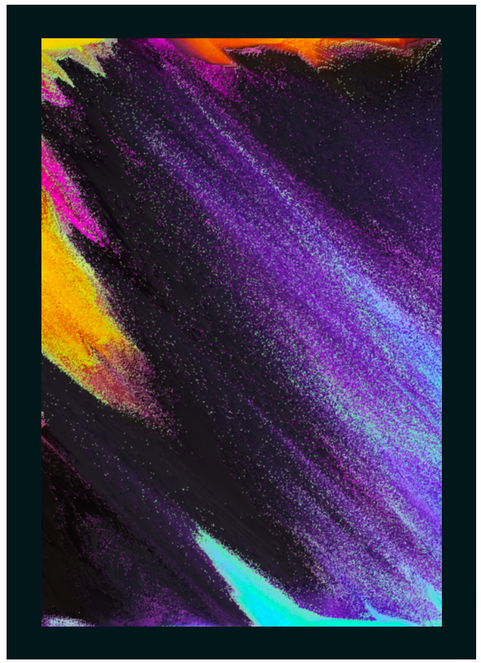
```
def setup():
    global img, imgNew
    size(700, 400)
    background(0)
    img = loadImage("sketch.jpg")
    imgNew = loadImage("sketch.jpg")
    imgNew.resize(700, 500)
    img.resize(700,400)
    img.loadPixels()
    colorMode(RGB)

def convert_to_rgb(hexcode):
    hexcode = hexcode.lstrip('-')
    return list(int(hexcode[i:i+2], 16) for i in (0, 2, 4))
```

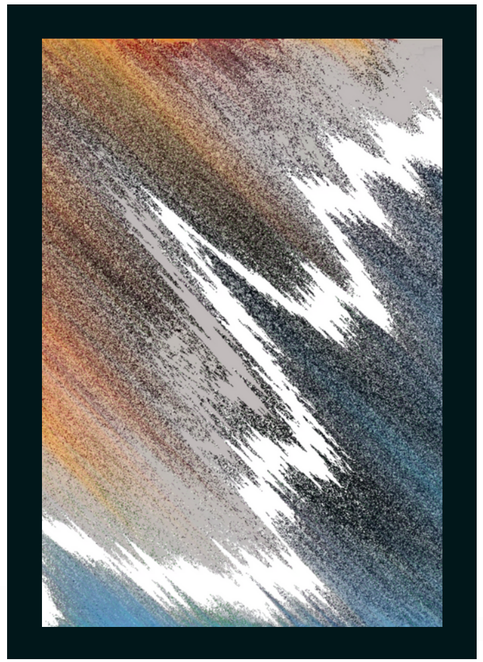
@NEERAJP99

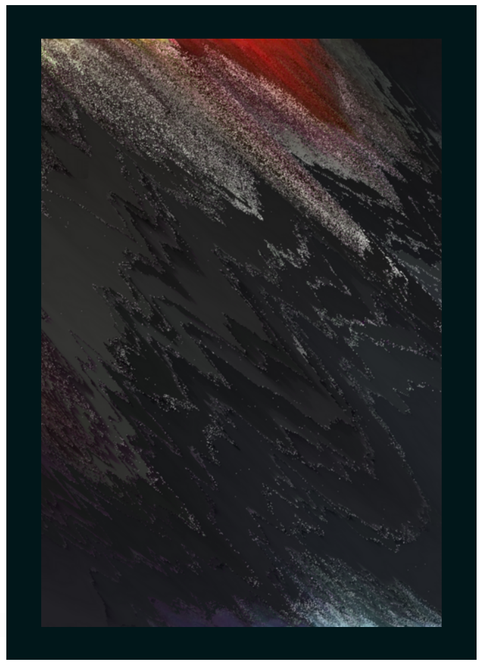


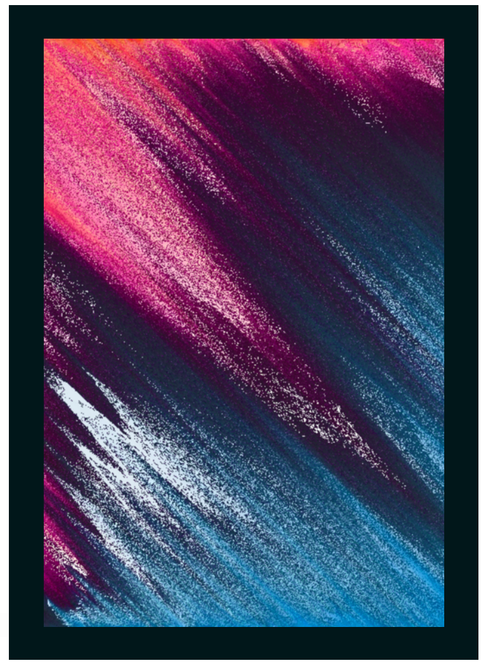


Pixel Sorting Algorithms v2.0









@NEERAJP99

THANK YOU

 neerajp99

Code: https://bit.ly/2WDxqsC