

**In Ordnung**

**Lara Stumpf**

## **In Ordnung**

Lara Stumpf

A Homage to Frieder Nake

*6/7/64 Nr. 20 Zufälliger Polygonzug*

**Das ist doch total billig!**

*Frieder*

<b>0</b>	8
<b>1</b>	12
<b>2</b>	14
<b>3</b>	18
<b>3.14</b>	24
<b>4</b>	40
<b>5</b>	48
<b>6</b>	58
<b>7</b>	70
<b>9</b>	84
<b>13</b>	102
<b>14</b>	110
<b>16</b>	120
<b>17</b>	134
<b>18</b>	150
<b>161238</b>	168
<b>19</b>	210
<b>20</b>	230
<b>2</b>	234
<b>1</b>	244
<b>0</b>	258



## The Zeroth Line

*In Awe*

What would become of us if there were no music.  
What would become of us if there were no poetry.  
What would become of us if there were no young students,  
students who give a damn when their professors remind them of  
modules and credit points and marks and final exams – occasional-  
ly, at least, giving a damn for what the systems require, and instead  
follow their desires their aspirations their intuition their love and  
their love. Occasionally at least.

For two hours today – before I finally sat down in the still young  
night to start writing these words dedicated to Lara Stumpf – for  
two hours today I had been listening to the powers of the music of  
Mikis Theodorakis and the poems of Pablo Neruda, for those two  
hours today I was exposing myself to the overwhelming sounds  
of those voices and those drums and those violins, those flutes,  
pianos, and harps into which Theodorakis had wrapped Neruda's  
words. My muscles and nerves were melting into tears running  
down my cheeks. The music-poetry stopped all my thinking of  
rational thoughts. It was creeping into my soul, my body-and-soul,  
and made me forget Chile and Greece and exiles and Victor Jara's  
deathly torture in the stadium of Santiago de Chile. Made me for-  
get? No, in no way! The poetry of the Canto General and its music  
and my body became one.

But then I took to Lara's book. A full-fledged book that she had  
done as her contribution to a seminar. A seminar offered to the  
undergraduate students of a seminar, and the seminar came under  
the title "The Algorithmic Dimension in Art."

Lara at that time decided to do a book. A book with a lot of draw-  
ings and little, mainly technical text, text that contained program  
text, it constituted a new kind of text, a unity, as Donald Knuth had  
requested long decades ago, a unity of plain and algorithmic text.  
And when Lara's algorithmic text was able to generate a closed pol-  
ygon of seven edges, then there came seven pages of seven-edged  
drawings. And they were clean and clear and geometric and, dare I  
say, beautiful. In a way they were beautiful.

They were far away, very far away from the storms of the mighty  
poetry and music that had made my tears flow. But far away as  
those lines and polygons were from the immediacy of the words  
and the sounds, they were beautiful. The beauty of those straight

sharp clear lines is a mediate beauty – only. But that frustrating  
word "only" is revealing. Why not confess that the lonely one  
line, the cosy two lines, the touching three lines, as all the other  
lines, live their life of purity, their absolute life that largely rests  
in itself. They don't grab you and shake you. They appear, silently,  
not demanding anything. They stand for themselves and if you are  
willing, you hear them whisper. There is a warm melancholia with  
the straight lines. No otherness, selfness only.

After 3 (three) in ordinary life comes 4 (four). But Lara's life is no  
ordinary life and, therefore, in Lara's book, that carries the title *In  
Ordnung*, after three and before four comes three point one four.  
Which in itself (would you not agree?) is a short little poem. It is a  
poetic text in a similar way as El Lissitzky's graphic works are poetic  
texts. Only they are no texts.

As all the other sections in *In Ordnung*, the section *The Tree Point  
One Fourth Line* also boasts of a number of drawings. How many  
edges, and how many of them? That must have been the  
toughest question to Lara in the entire book. All drawings of this  
section are composed of one triangle and one circle. And there are  
eight of them. A prize has been set for the first reader who explains  
the hidden riddle. We easily discover that in three of the drawings  
the circle and the triangle are intersecting, and in three more they  
are totally separate, whereas in one the triangle is inside the circle,  
and in one a tiny circle is inside its triangle. It looks forlorn, and  
arouses feelings in me of caressing.

After all this, you may not be surprised when you discover that an  
Eighth section is missing.

I don't think my task here is to give the contents of this book  
which is, obviously the first computer-generated graphic novel. But  
I should be very frank and tell you that the book whose preface I  
here have the great honor to write contains a lot more of surprises.  
They all come from a beautiful line of narration that runs through  
the book and culminates in an algorithmic text of the finest kind.  
But they also come as parts of an intriguing series of intricate line  
drawings that surprisingly also culminate in that same algorithmic  
text. How come?

Well, in algorithmic art, as well as in any other kind of applying  
computing principles to generate something, there hovers a dupli-  
cation of the world. The world now (where the word "now" means:  
"as the Algorithmic Revolution has already done a lot but is still  
going on") is a world of the semiotic animal meeting the semiotic  
machine. The semiotic animal, that's us humans. The semiotic

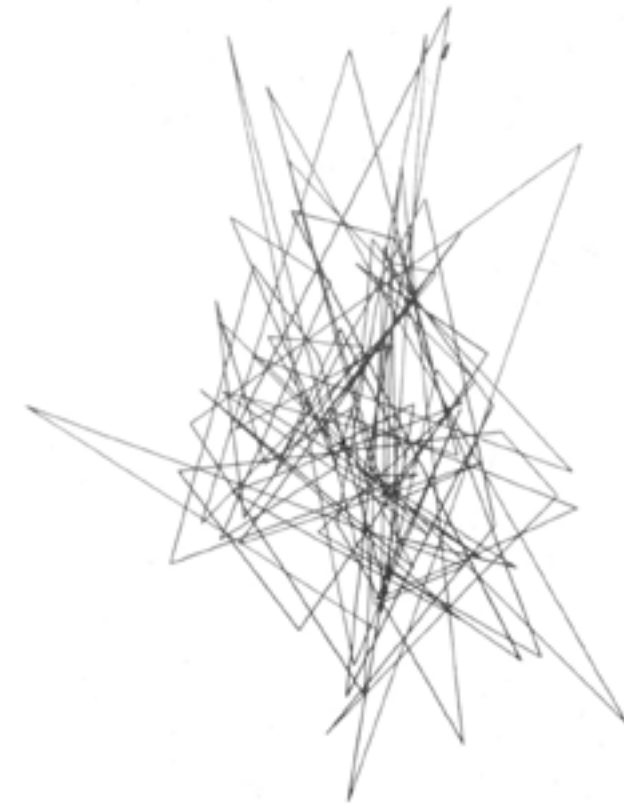
machine, that's them computers. They meet at interfaces, as one says. And those interfaces, in order to allow for the meeting, are a surface and a subface, tightly glued together. The subface is the machine's view, the surface is ours. They are meant to be manipulating and perceiving.

I stop this here. For digging deeper into such considerations is not really our business here. But I felt I should at least mention it since Lara Stumpf has set with her book a monument of this surface-subface view I have been propagating for a number of years. She has found this and then made it explicit independent of my telling her.

Her entire book with its fine text passages and with its rational drawings (that build an entire compendium) is a pleasure to hold in your hands, to not only flip pages, but to carefully wander from one page to the next. Lara Stumpf's book is a marble. It is a marble of taking up in light gestures a convincing theoretical foundation of computing. It is an Artist's book.

I stand in awe. It is as if I was listening to the Canto General. What would become of us if there were no algorithmics.

Frieder Nake, 30 April 2018



20/7/64 Nr. 20

## The First Line

A line segment is the shortest path between two points. Every line segment consists of an endless number of points. Use a ruler and a pen and connect two points on a piece of paper. Easy! Now we do the same on a computer. Easy? We can start out by selecting two points and connect them with dots.

```
point(5, 5); point(10, 5);  
point(6, 5); point(7, 5); point(8, 5); point(9, 5);
```

What did we do now? The screen on a computer consists of pixels. These pixels create a raster. We coloured six horizontal pixels next to each other and since there is (almost) no empty space between them, we will call this a line segment. We can do the same in the vertical direction.

```
point(5, 5); point(5, 6); point(5, 7);  
point(5, 8); point(5, 9); point(5, 10);
```

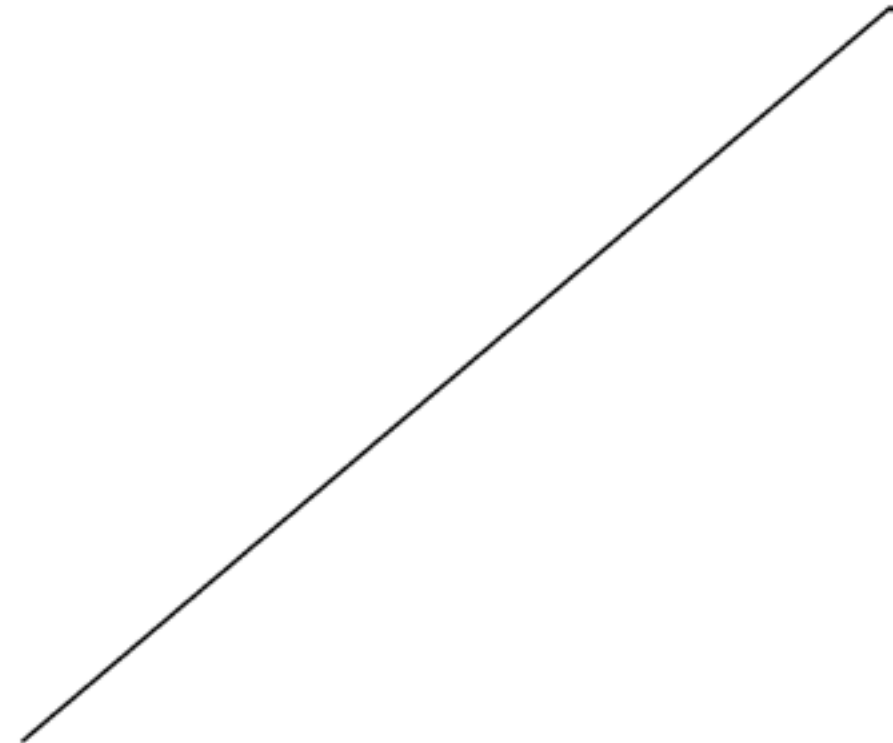
Still working. Let's draw a line segment that is neither horizontal nor vertical. But how? A line segment turned by forty-five degrees might, with a bit of goodwill, still make sense.

```
point(5, 5); point(6, 6); point(7, 7);  
point(8, 8); point(9, 9); point(10, 10);
```

Apart from that, however, most or maybe even all other line segments represented by pixels are no real line segments. Their graphical representation is only as close to the mathematical calculation of the computer as the raster allows. Since the pixels on screens are getting smaller and smaller, the represented line segments will get closer to the calculation. As long as a screen is a raster, the line segments will never be perfect though.

This chapter is about the first line. Line is the wrong term for line segment and these line segments cannot even be displayed on a screen properly. However, line sounds appealing. So we are going to call them lines! And since this work is done on a computer, we are going to use comfortable algorithms for calculating and drawing them.

```
line(random(0, width), random(0, height),  
random(0, width), random(0, height));
```

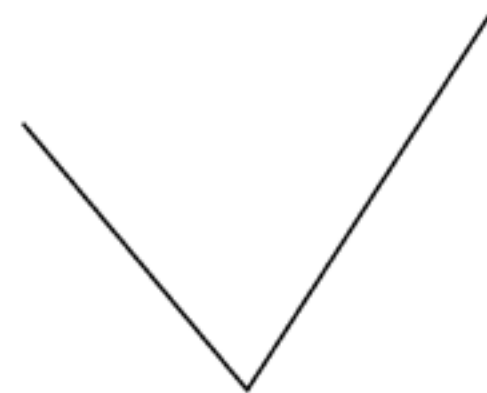


## The Second Line

What is a homage? Frieder himself will teach us. His work *Hommage à Paul Klee, 13/9/65 Nr. 2* is our guide. Comparing the homage to the original *Hauptweg und Nebenwege* reveals the closeness as well as the distance of both works at the same time. Frieder analysed the structure of Klee's work and picked it up in his own style: lines in black and white. He came up with details looking different but still giving the impression of belonging to the original picture. Furthermore he rotated the image, or rather the original structure, by ninety degrees and added some circles, even though there are no curvy objects in the original picture at all.

```
float x1 = random(0, width), y1 = random(0, height);  
float x2 = random(0, width), y2 = random(0, height);  
float x3 = random(0, width), y3 = random(0, height);  
line(x1, y1, x2, y2); line(x2, y2, x3, y3);
```

How to apply a personal style without having a personal style?



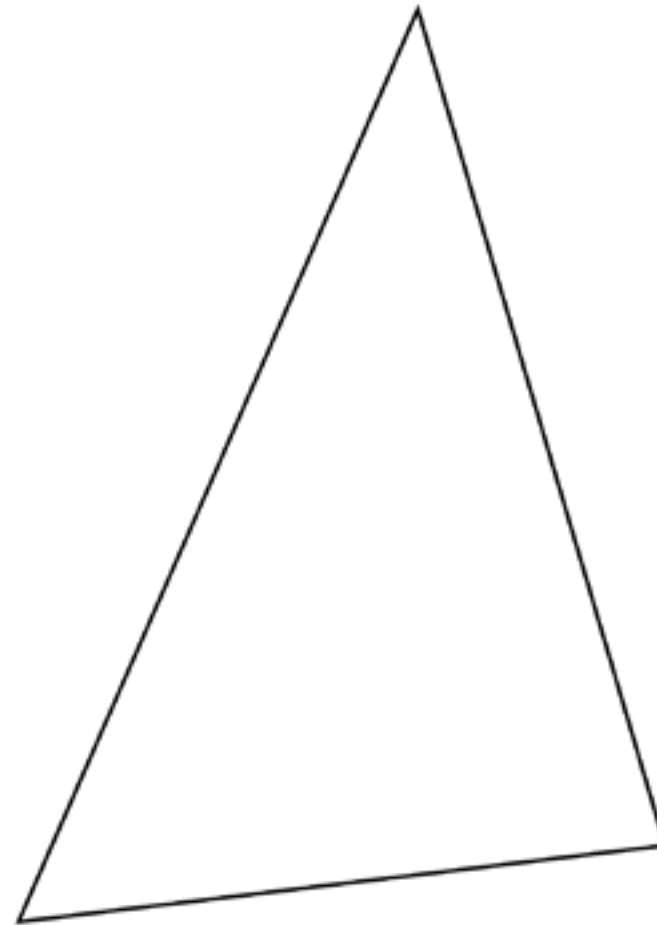


## The Third Line

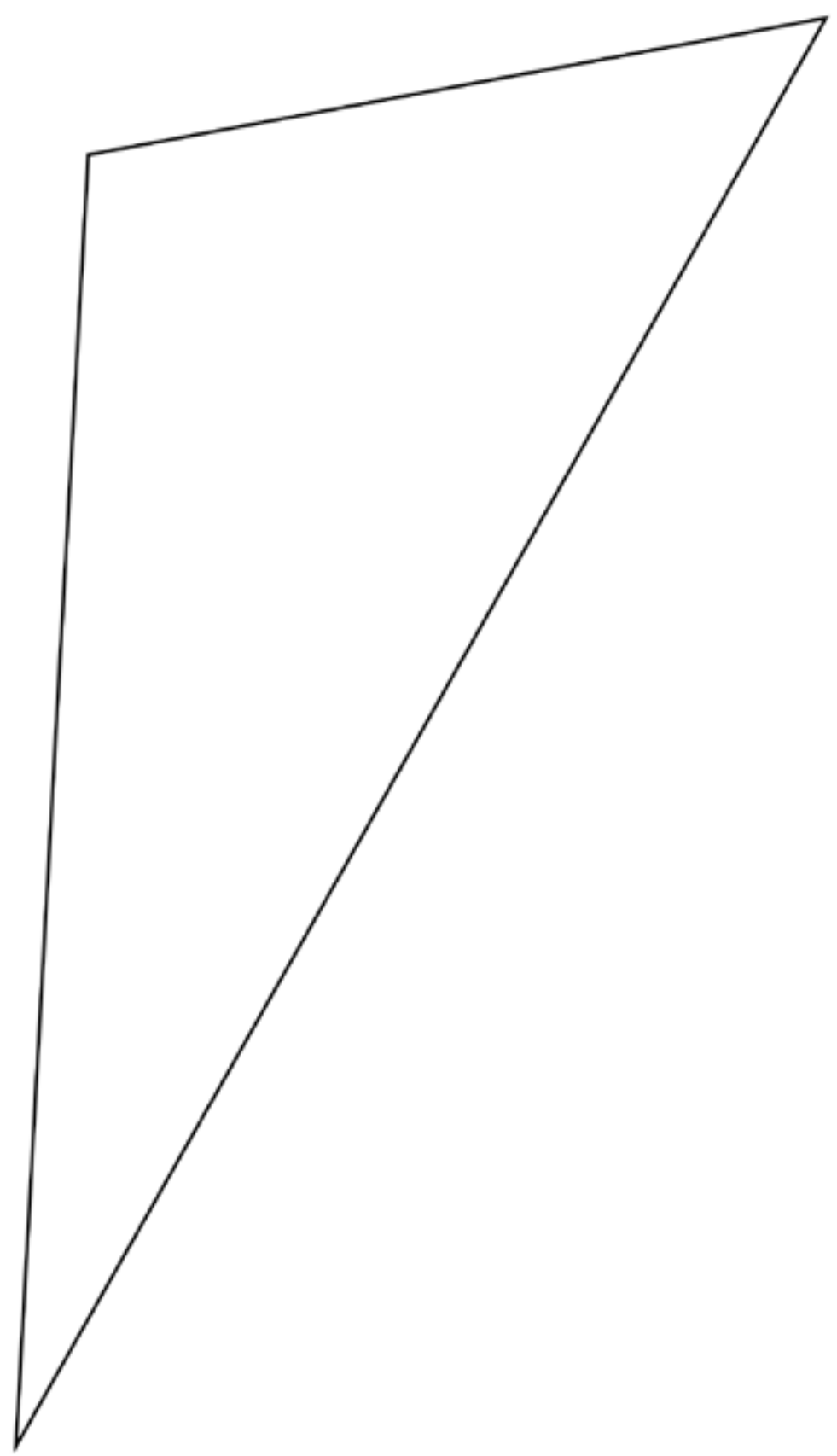
From an algorithmic point of view some things can be lines even though they would not be from a different, maybe artistic, point of view. Like a very short line with a heavy stroke. This line would rather look like a shape. When do lines look like lines and when do lines look like shapes? Are we used to several proportions? If the line with the heavy stroke became longer, we would think about a line again. Either way, with our third line we can start creating shapes with our set stroke weight. The second line already gave us a third point. All we need to do is connect all three points with each other and enjoy the beauty.

```
triangle(x1, y1, x2, y2, x3, y3);
```

Triangles, finally! Triangles are magic. Whenever a clever person finds out about something in this world, it has to do with triangles. Or at least they come up with some model in the shape of a triangle.





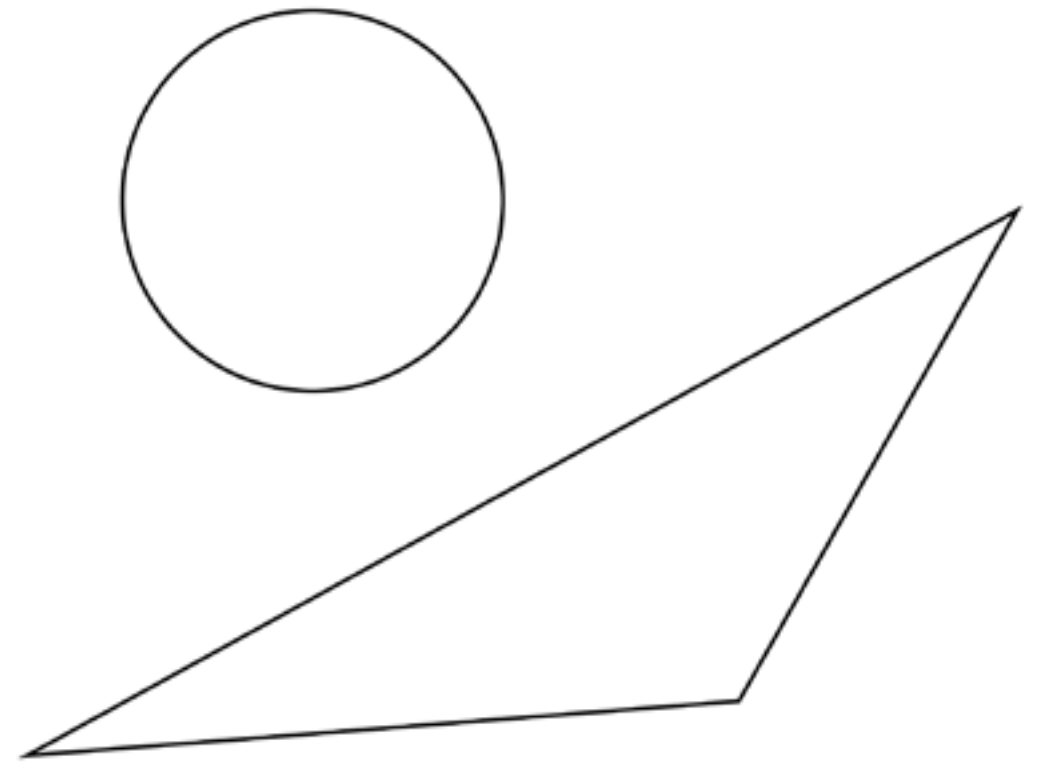


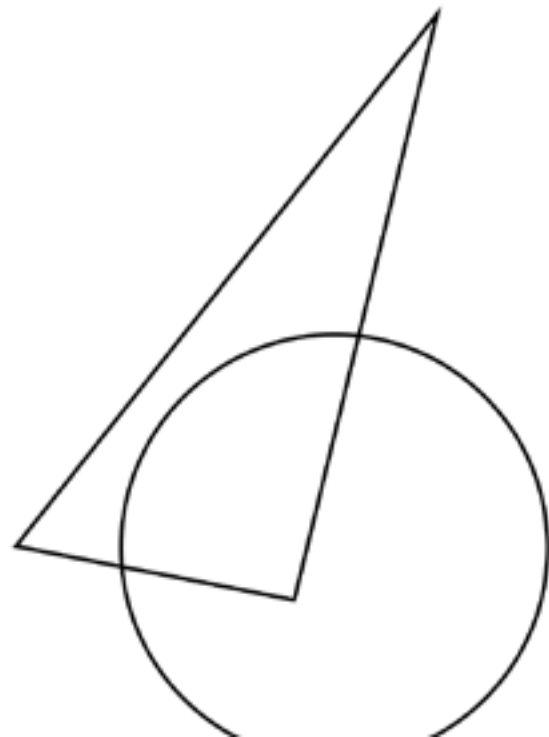
### The Three Point One Fourth Line

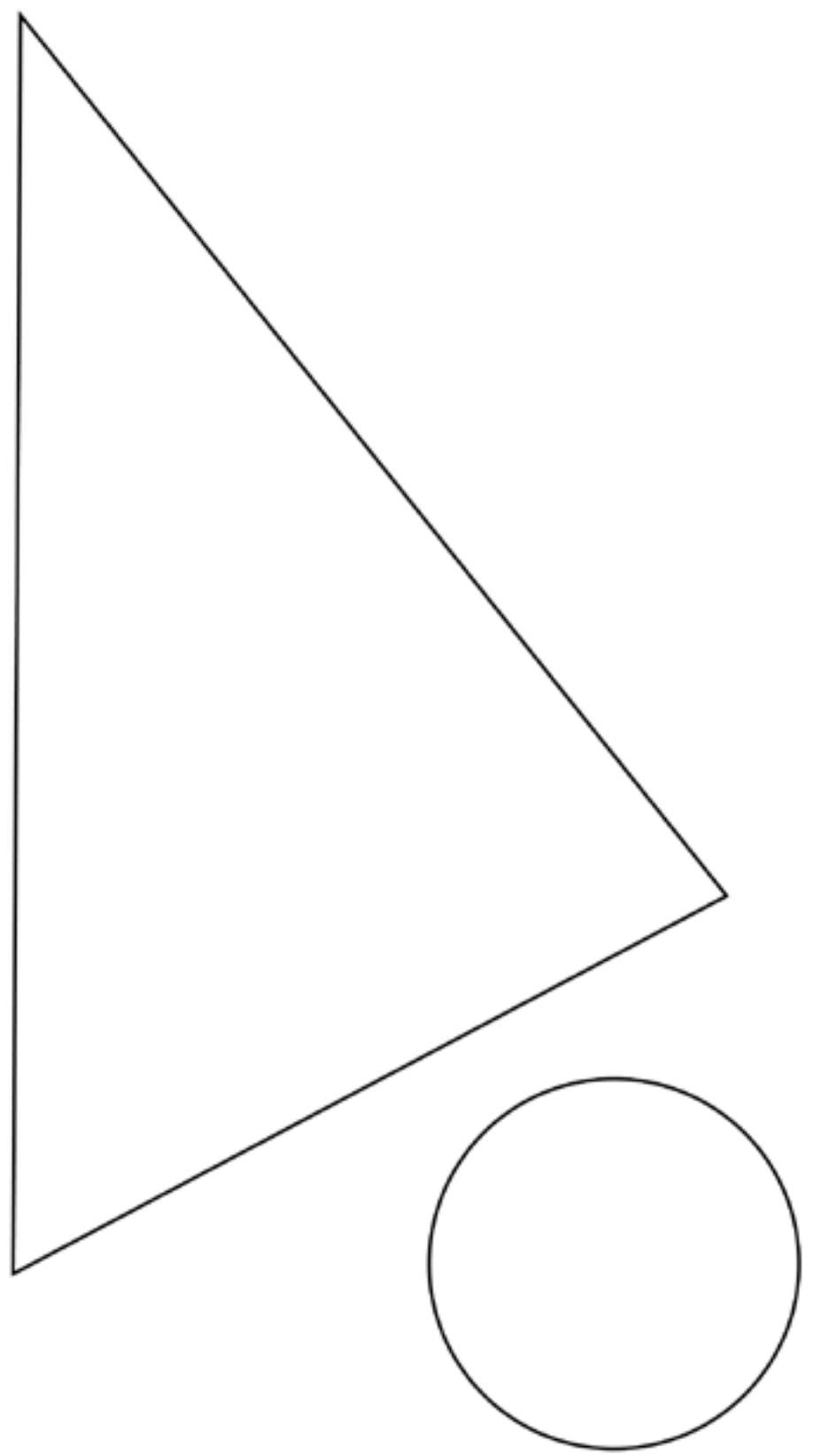
Circles are magic. Did you ever draw a perfect circle by just using a pen? Probably not. Thinking about the pixel raster again, circles on a computer are even worse than straight lines. It is very easy to create some shape representing a circle, though. Frieder taught us: If there are no curvy objects in the original picture, add some.

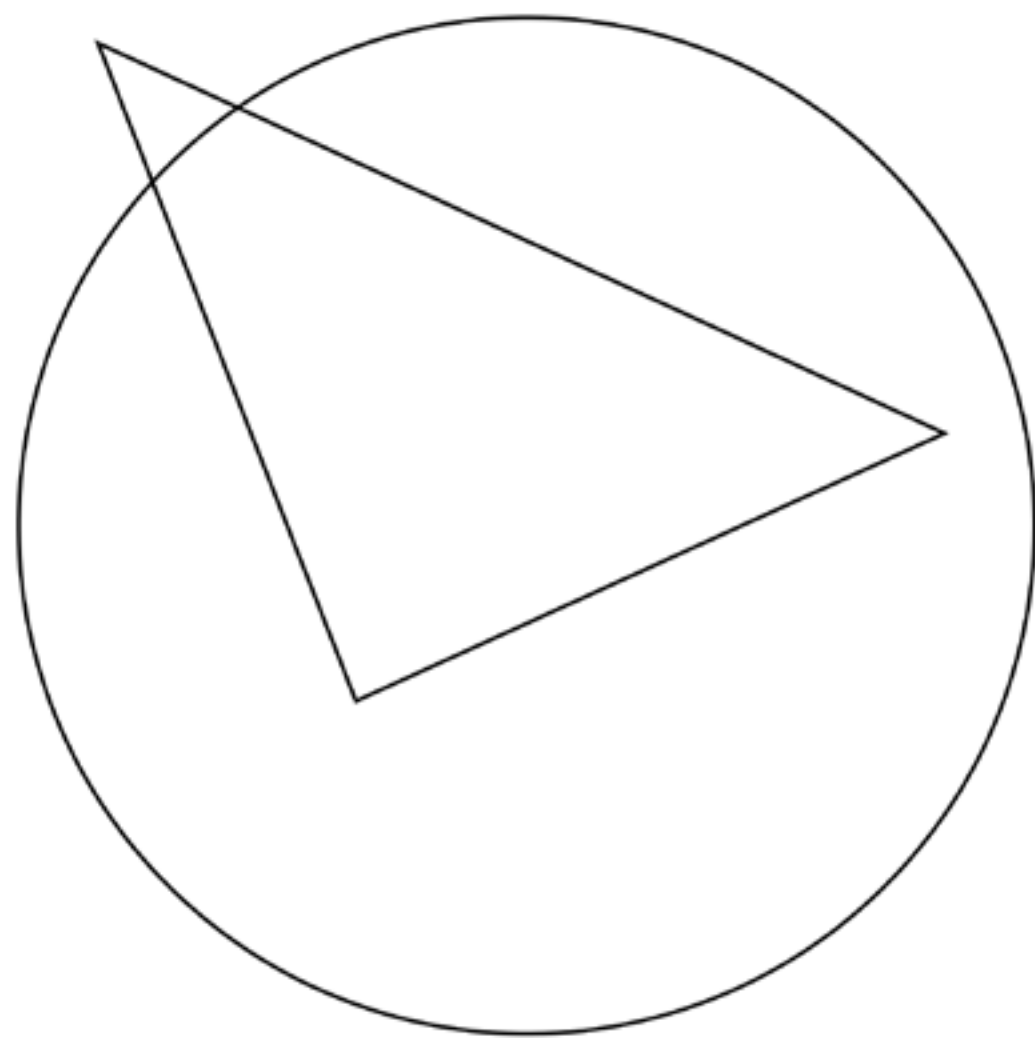
```
float d = random(0, width/2);  
ellipse(random(0, width), random(0, height), d, d);
```

I call it *The Interaction of Triangle and Circle*. Let us take a moment and enjoy the beauty of this simple combination. However, for now the circle will be left behind. Maybe there are going to be curvy objects in the end. Maybe not. At first we need to get the lines straight.

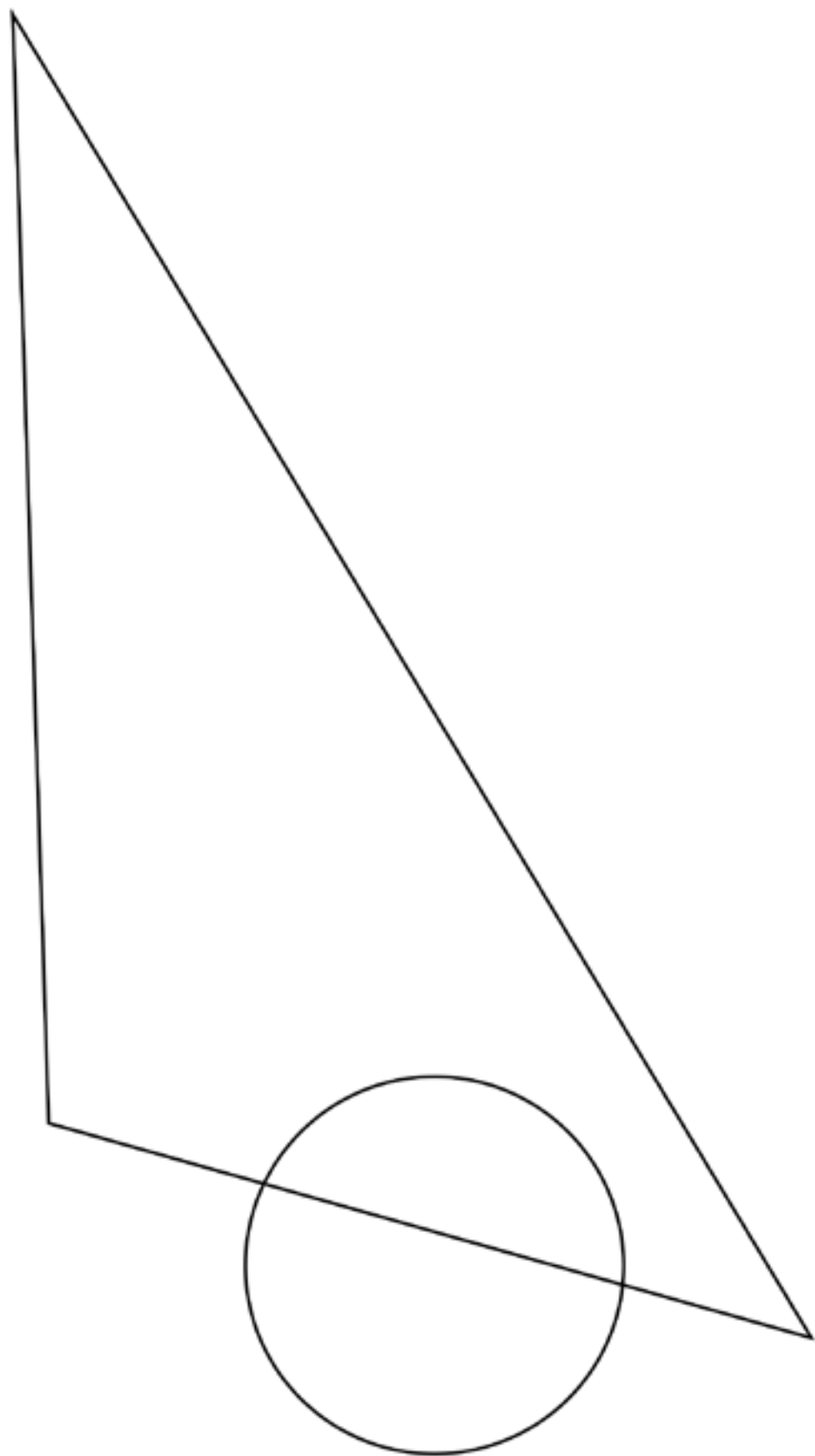




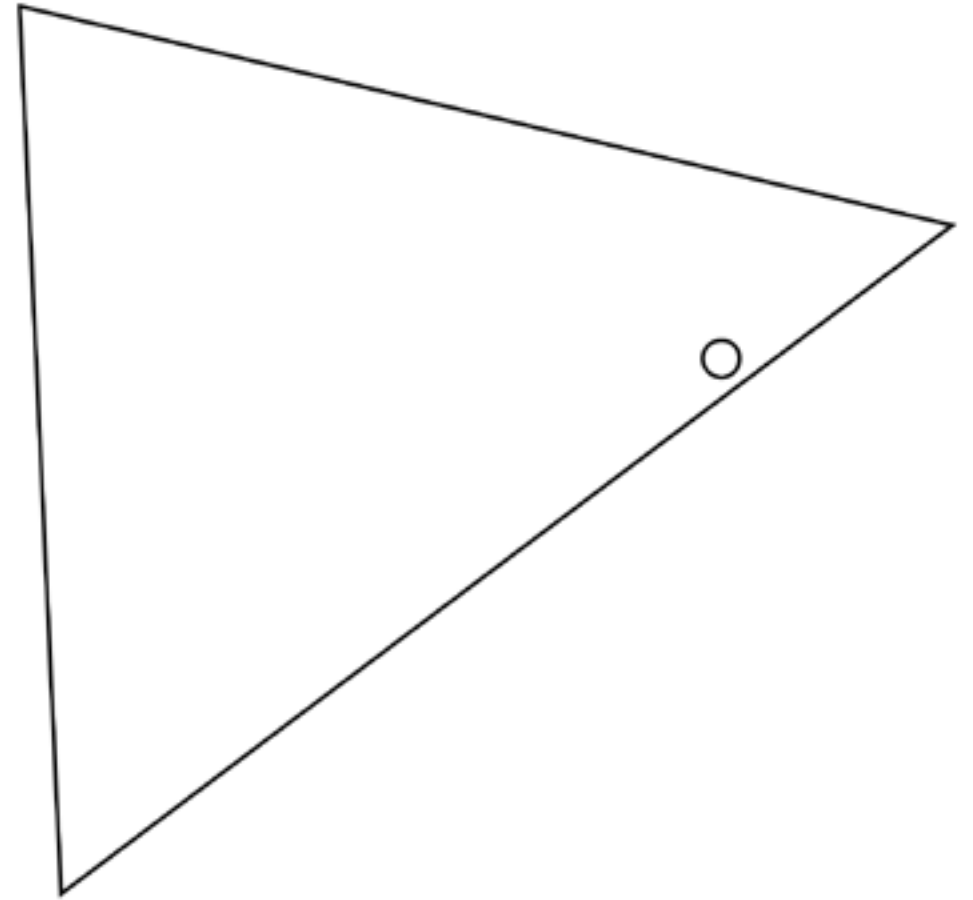












## The Fourth Line

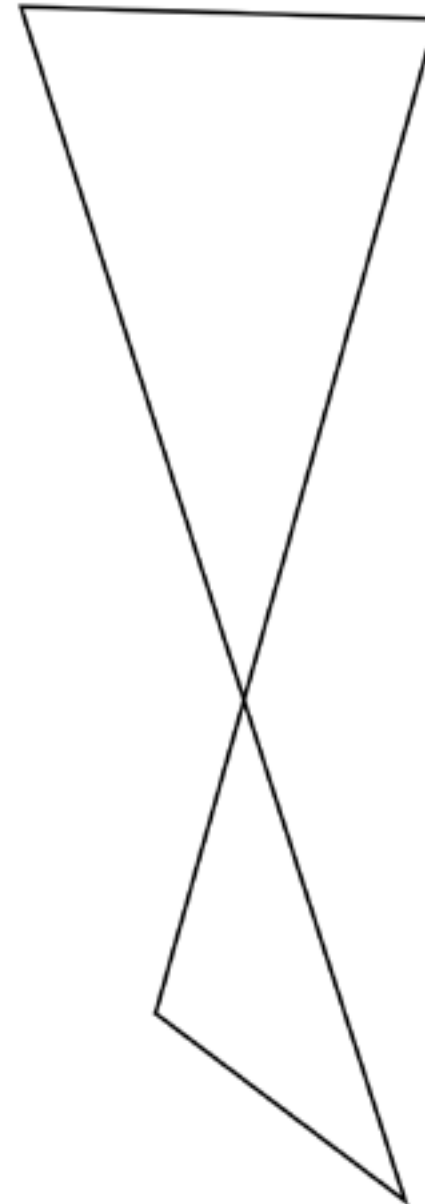
Four lines enable us to create a new shape: a quadrilateral, a parallelogram, a rectangle, a square. Josef Albers used rectangles and squares, stacking them up or aligning them. But his *Interaction of Color* contains, obviously, colours. Frieder works with black lines on a white ground. Maybe I could apply a bit of Lara by using colours?

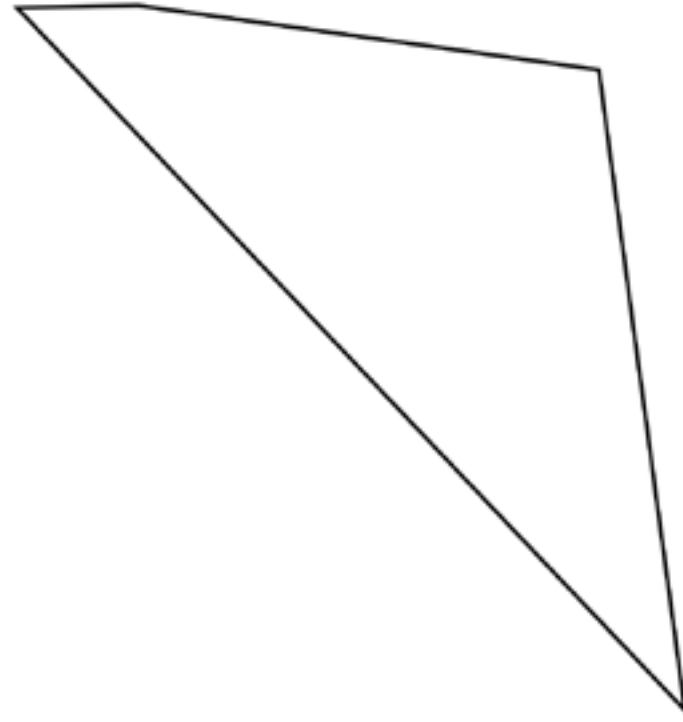
Let's take a closer look. Albers' pictures look very simple. Very few elements create a relation with each other using a few colours and probably some algebra. We can understand the basic idea of the pictures very easily and would rather question some details or a further meaning of the pictures. Yet, these simple looking pictures were probably very difficult to create. On the contrary, Frieder used an easy algorithm to create chaos in black and white. How many lines are there? How do they belong together? The picture is very complex and I have a lot of fundamental questions before even thinking about a further meaning.

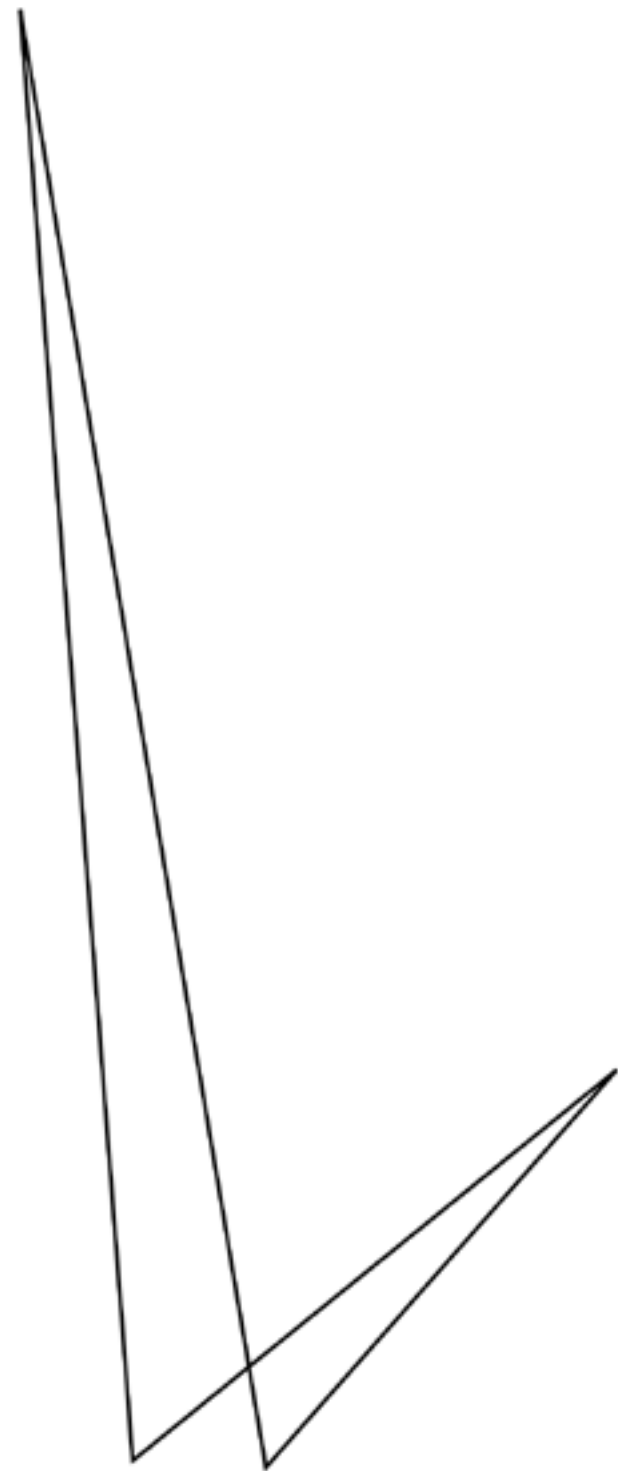
This is a homage to Frieder. At this point I need to stick with the idea of simple input creating chaotic output. I do not have a favourite colour nor any personal relation to colours. My experience of random and chaotic colours is rather ugly. Frieder is not ugly. I am not ugly. Ugly colours are forbidden.

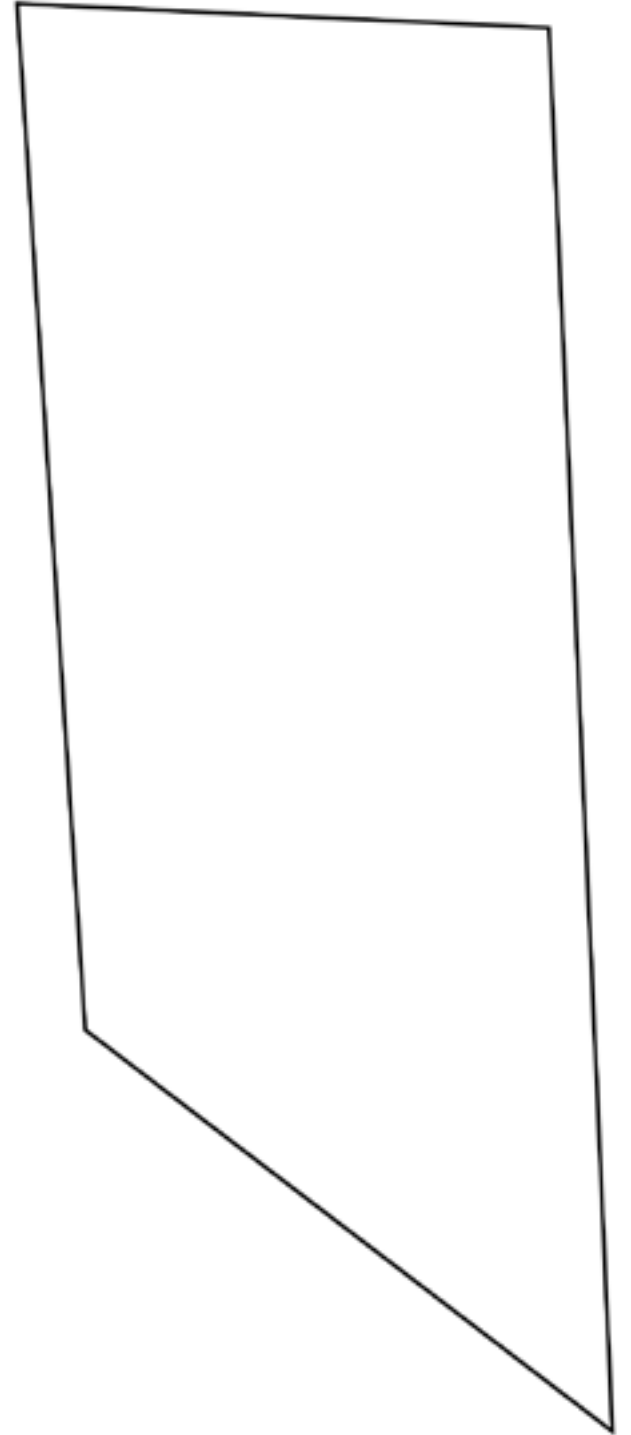
`quad(x1, y1, x2, y2, x3, y3, x4, y4);`

With four lines we can already create a bit of chaos. If we are lucky, they will create two shapes and thus look like six lines. This is getting more and more fun!







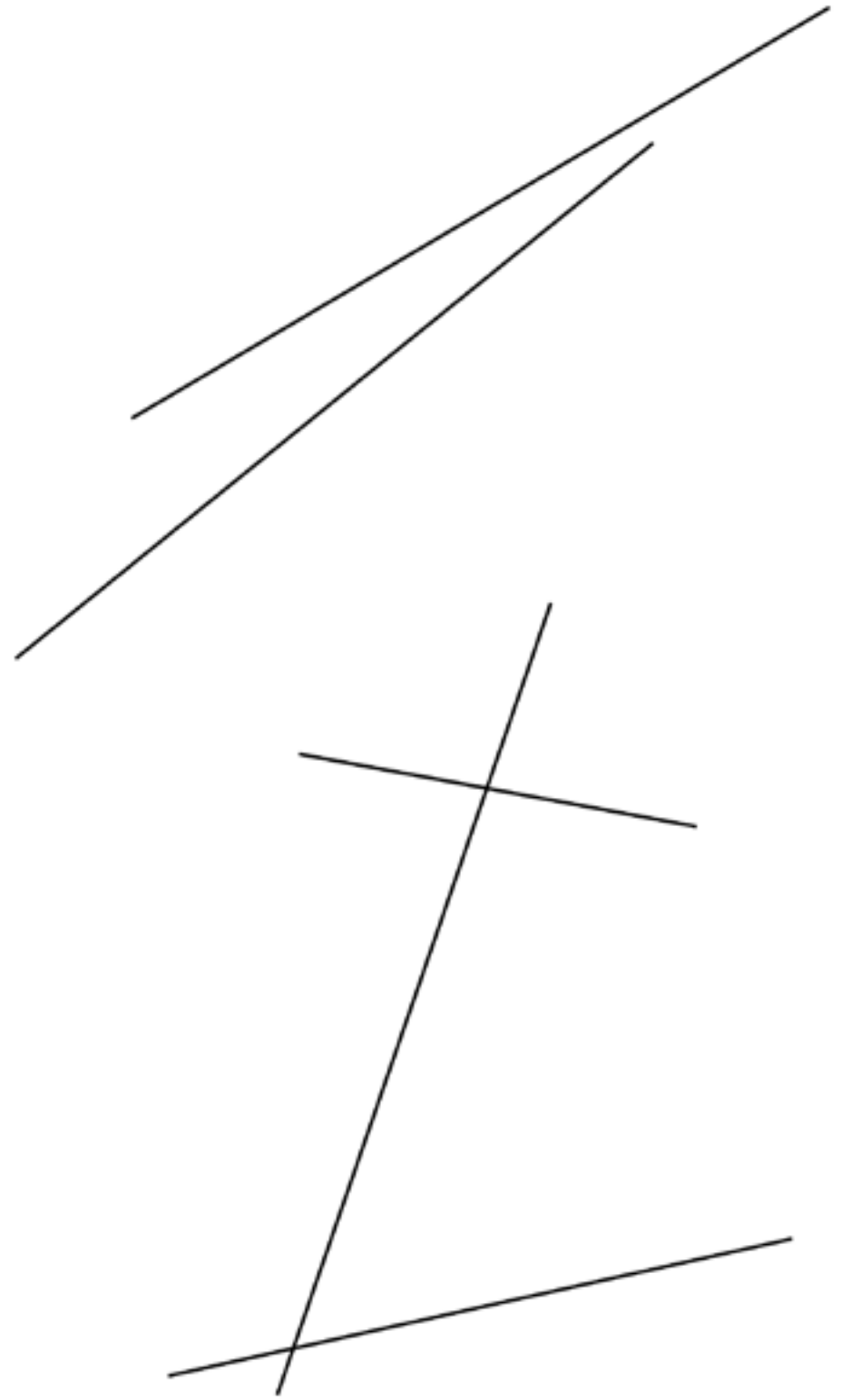


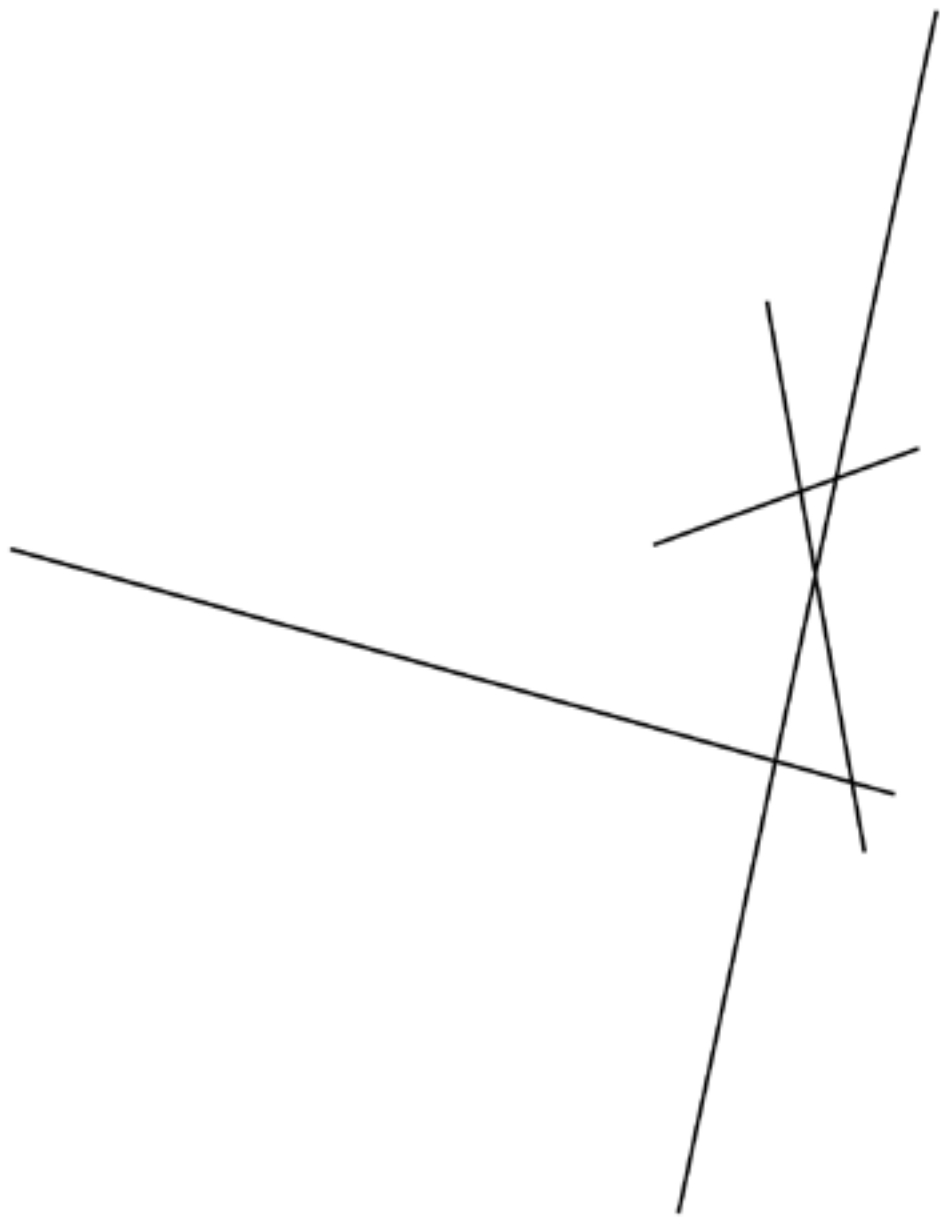
## The Fifth Line

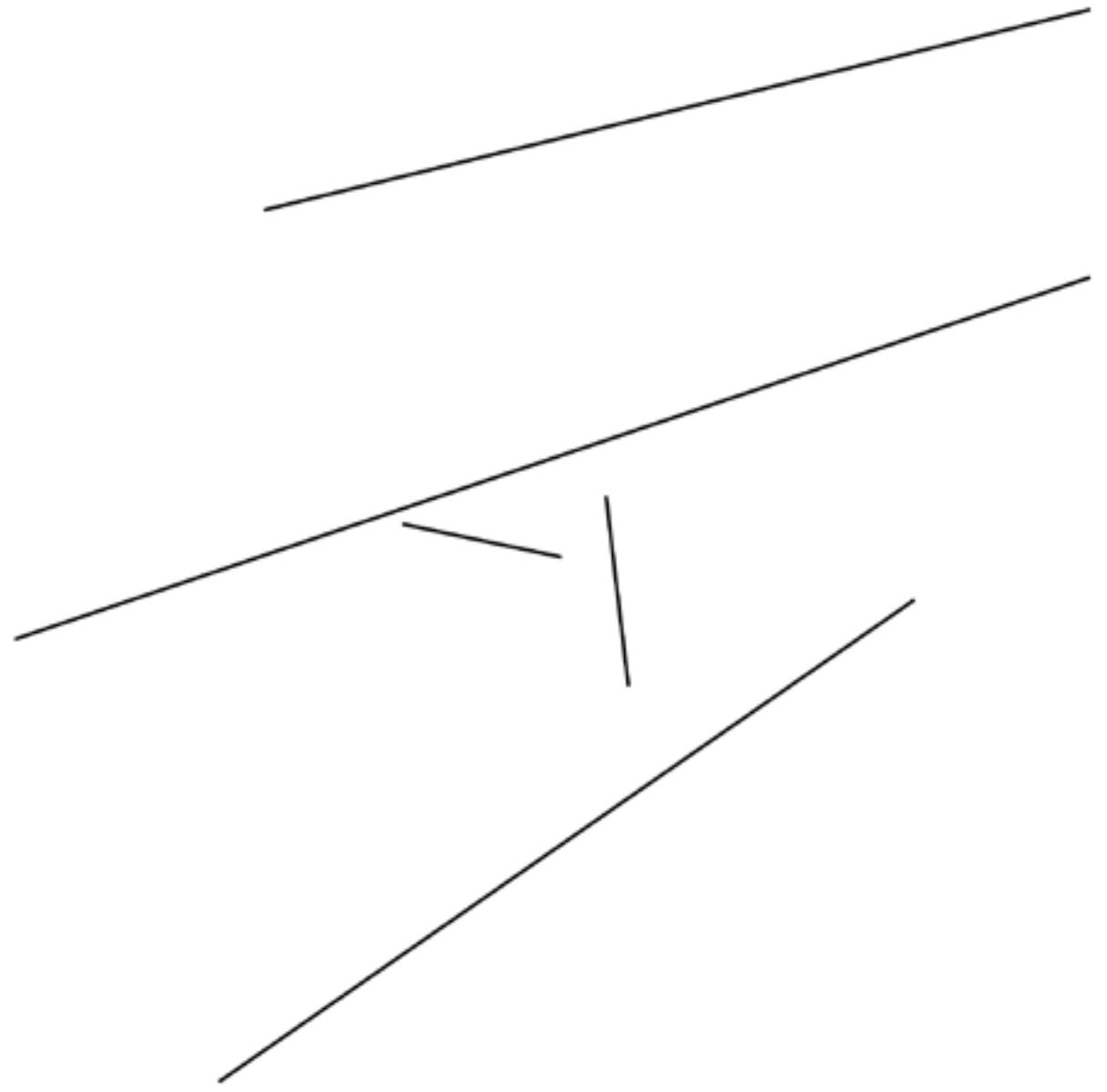
More lines always mean new shapes. Why stick to shapes? The lines can be anywhere on the canvas. Maybe touching each other, maybe not.

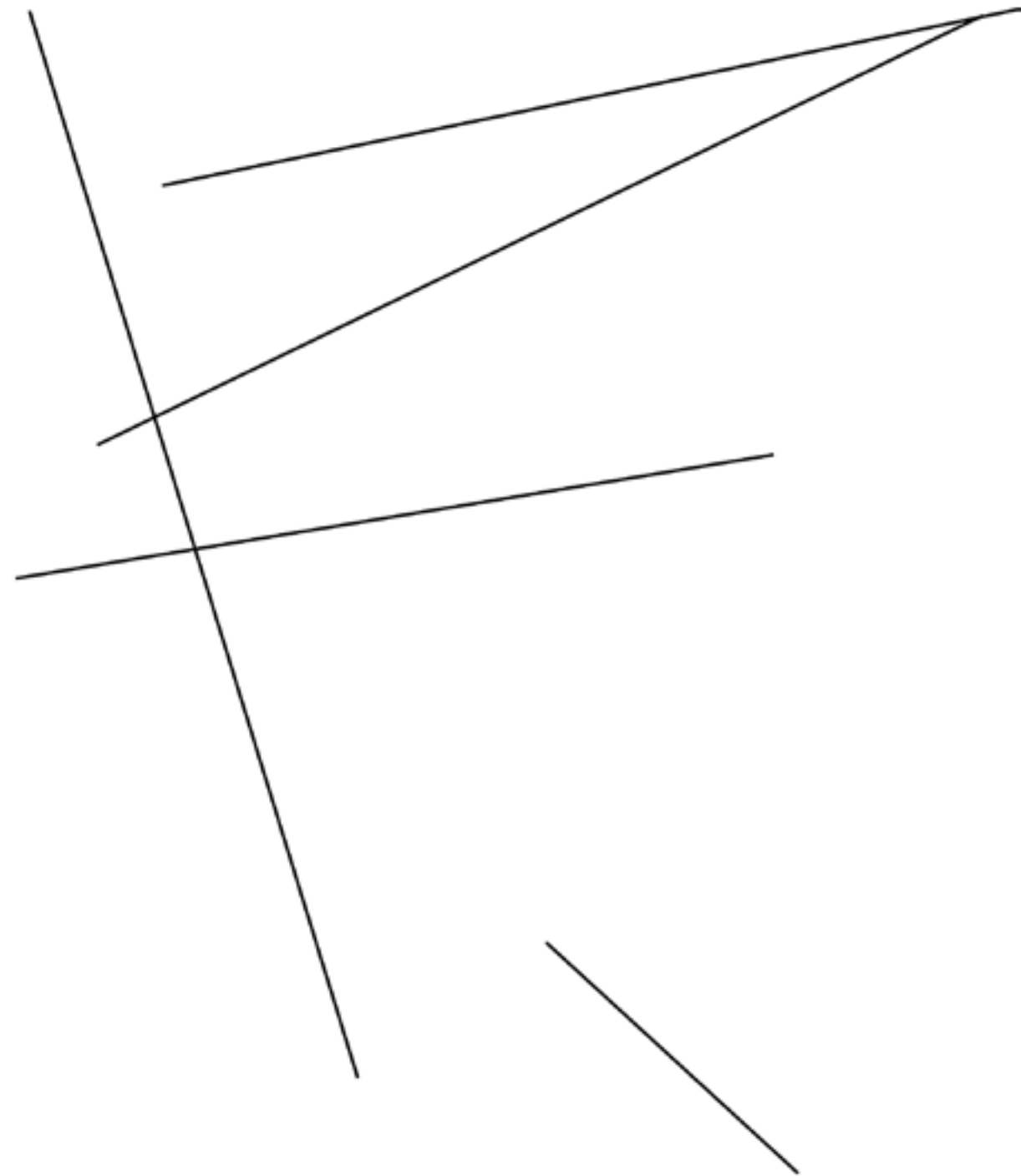
```
for (int i = 0; i < 5; i++)  
{  
  line(random(0, width), random(0, height),  
        random(0, width), random(0, height));  
}
```

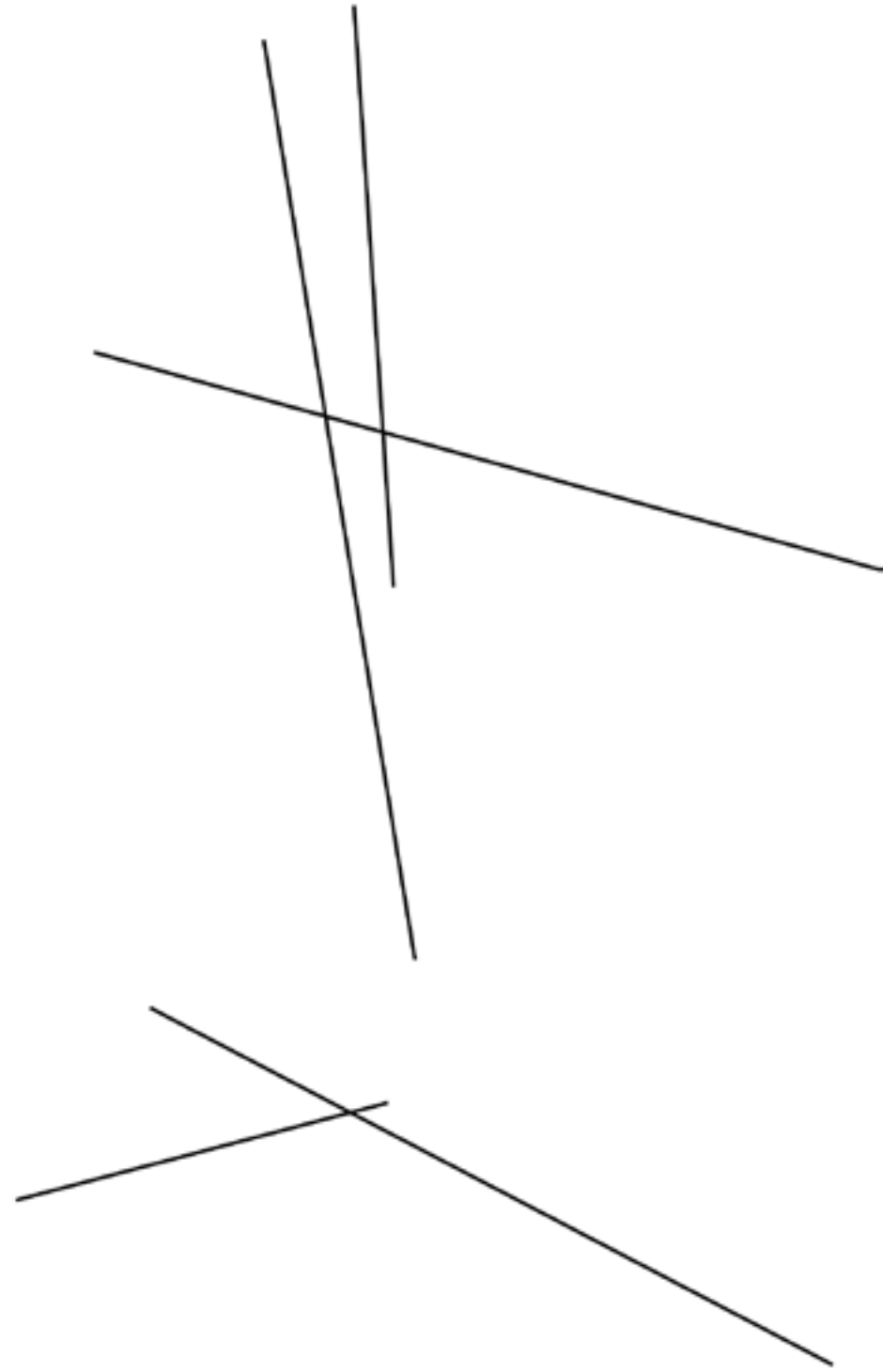
But why? Frieder connects his lines and makes them belong together. He even uses a mighty Friederline to connect his startingpoint with his endpoint. Unfortunately there seems to be no reason for the lines to be separated at this point. They seem to be quite chaotic, but that's it. What a shame, they look fantastic!











## The Sixth Line

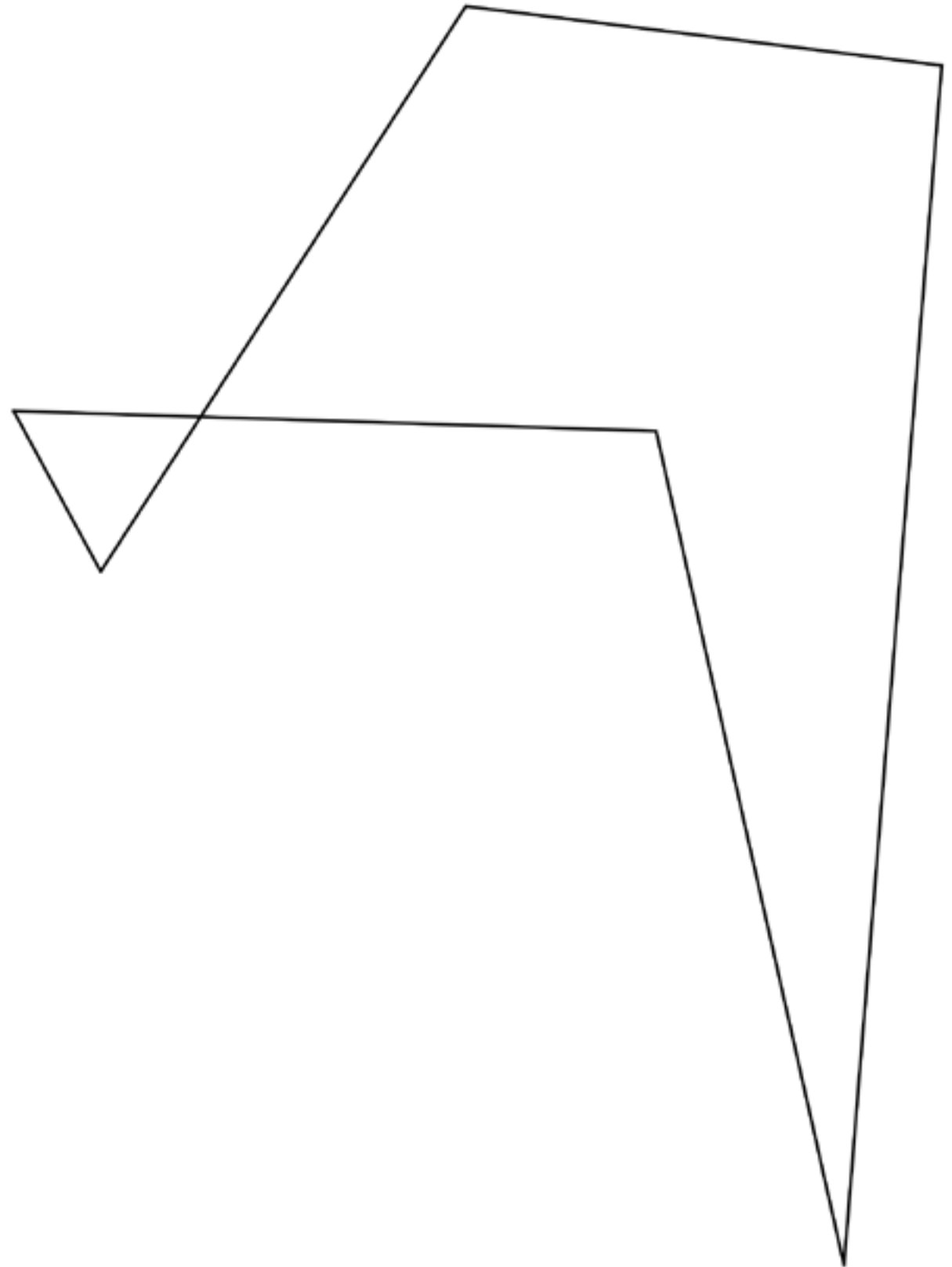
Input creates output. In this case the input consists of algorithms I have done using code someone else has developed and the output is generated by the computer. We need to stay with the computer creating the output, but what about different input? This algorithmic piece could be interactive! There are a lot of ways *you* could be a part of the input! And since art does not need to be printed anymore nowadays, it would even be available for everyone! That sounds great!

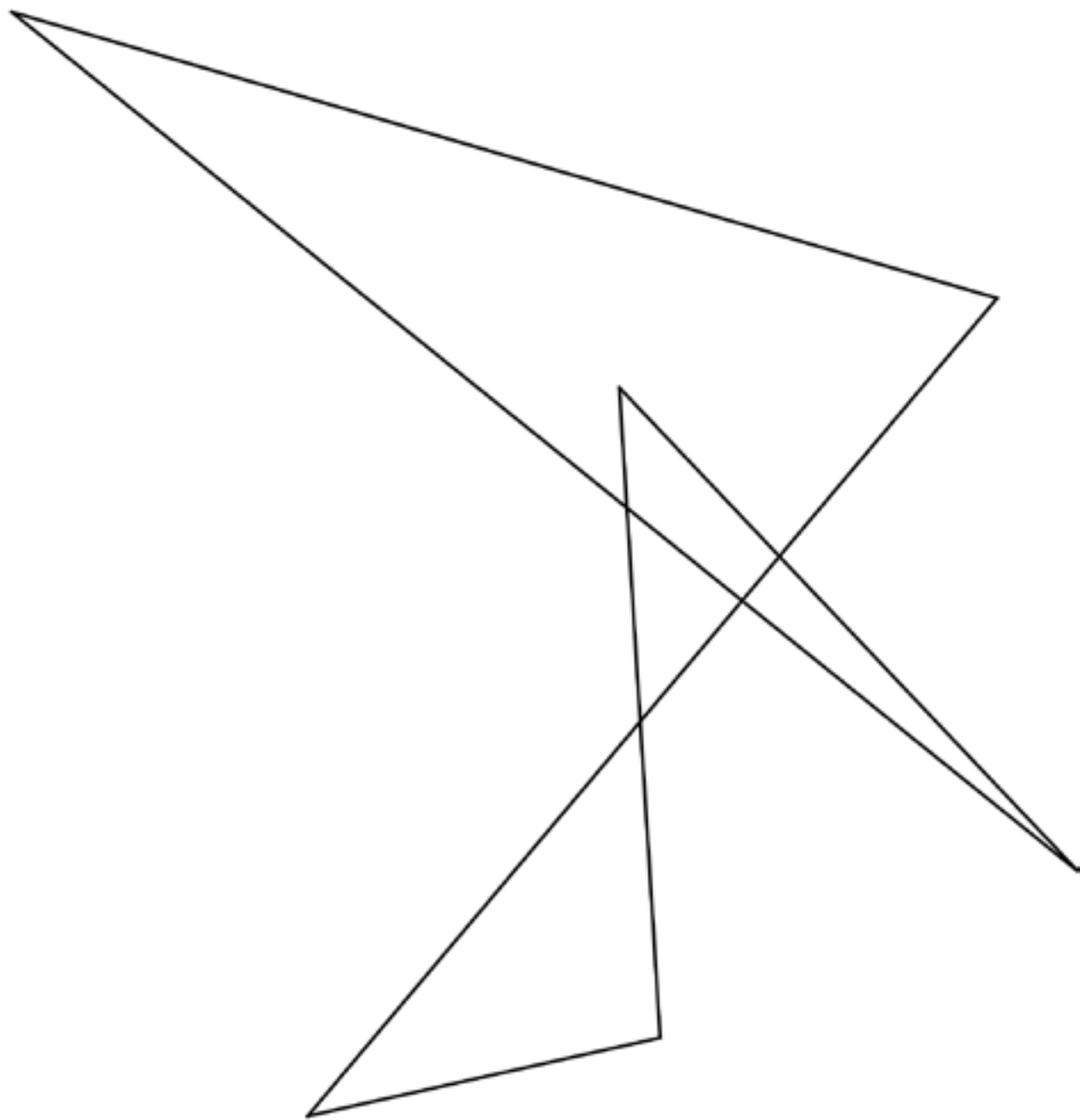
But there is a problem. I am the artist. And this is a homage to Frieder. Why should anybody else apart from the coders, the computer, Frieder and me be part of this? To be modern, fancy and cool? Pha! This is not funny at all. This is dead serious. Since when is art supposed to be fun? Nothing of this is.

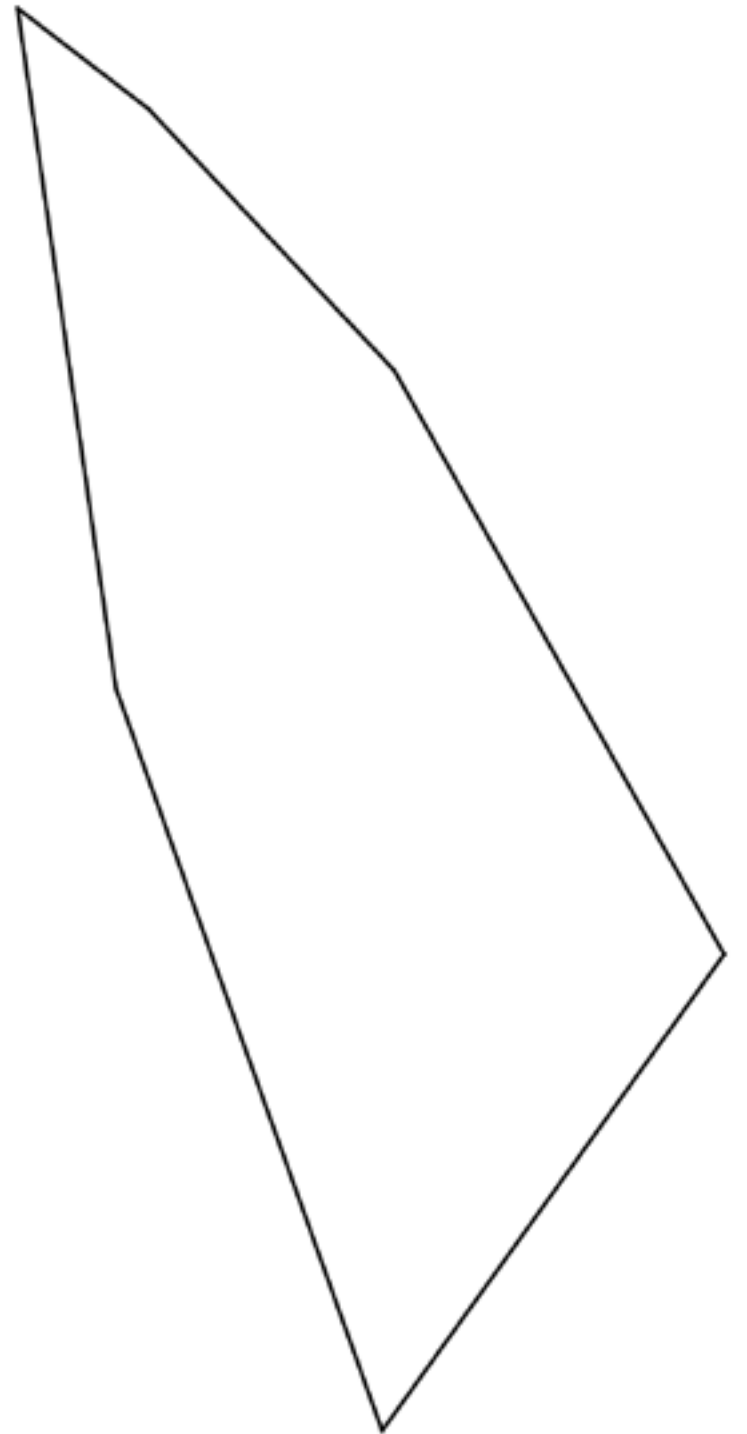
Just kidding. But still, this should be more than just a fun gadget for boring times. No input is going to be done by you.

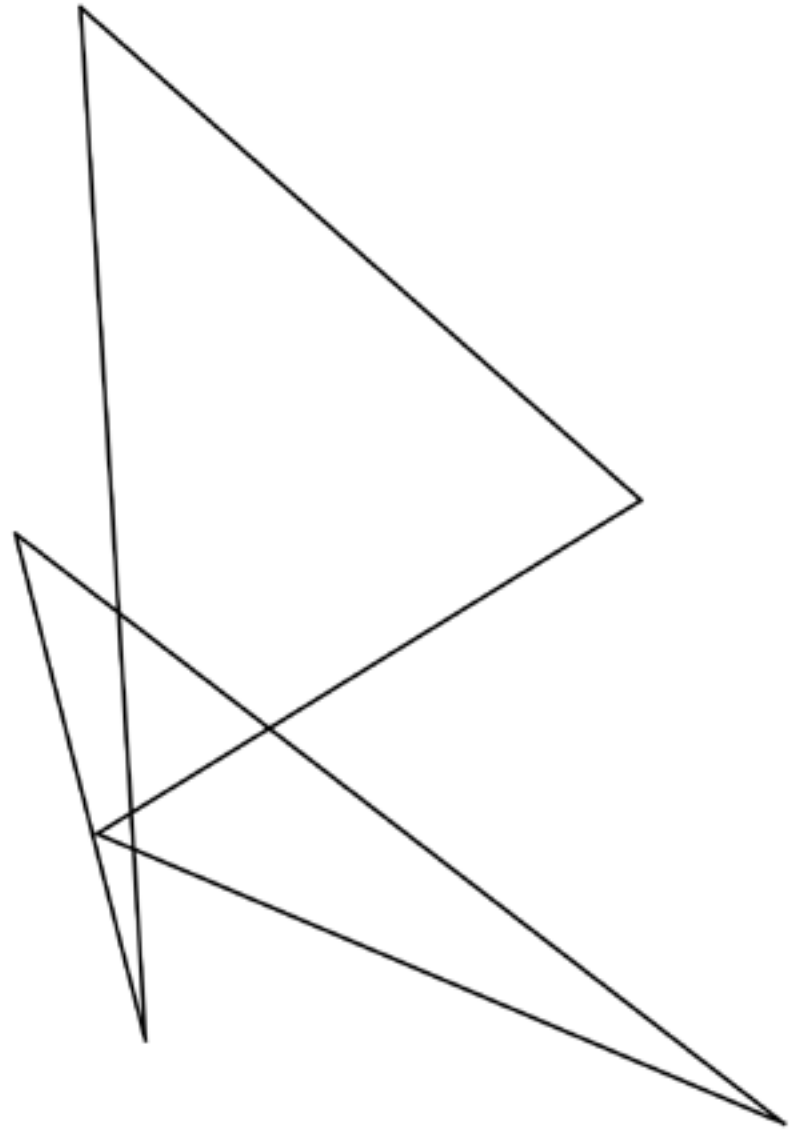
```
int howMany = 6;
float[] x = new float[howMany];
float[] y = new float[howMany];
for (int i = 0; i < howMany; i++)
{
    x[i] = random(0, width);
    y[i] = random(0, height);
}
for (int i = 1; i < howMany; i++)
{
    line(x[i-1], y[i-1], x[i], y[i]);
}
line(x[howMany-1], y[howMany-1], x[0], y[0]);
```

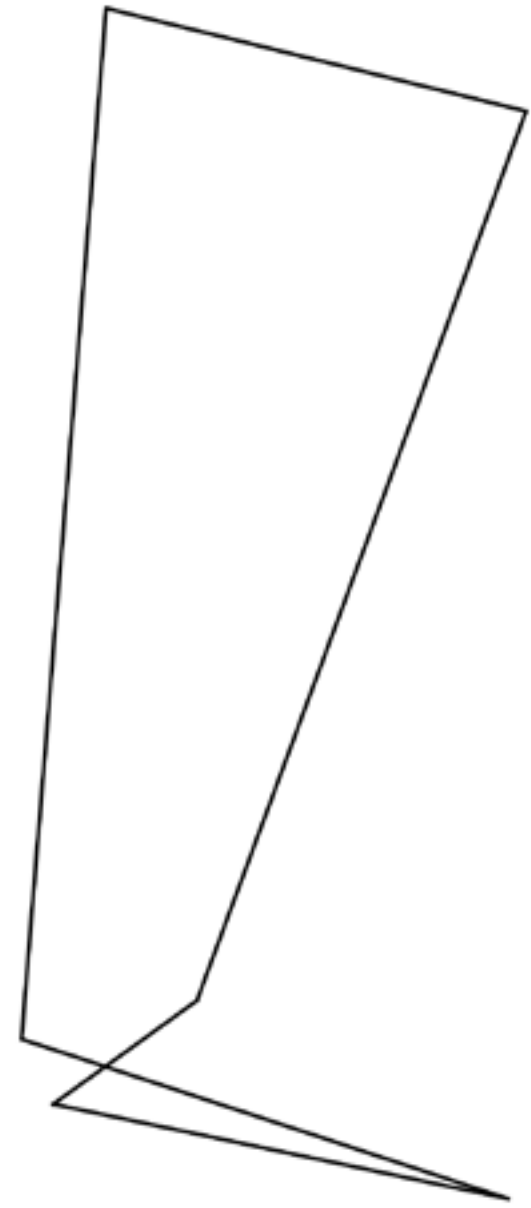
Not yet very chaotic, though. Reminds me more of Albers or Picasso. If this was a homage to Picasso in Frieder's style, I might actually be on a good way. But it is not.











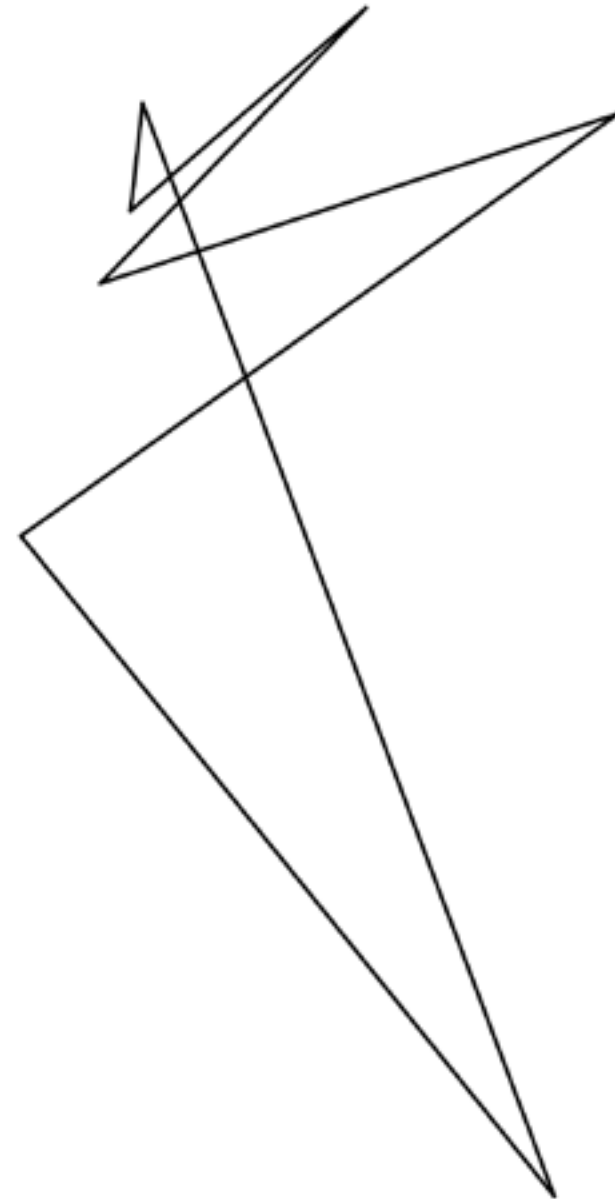


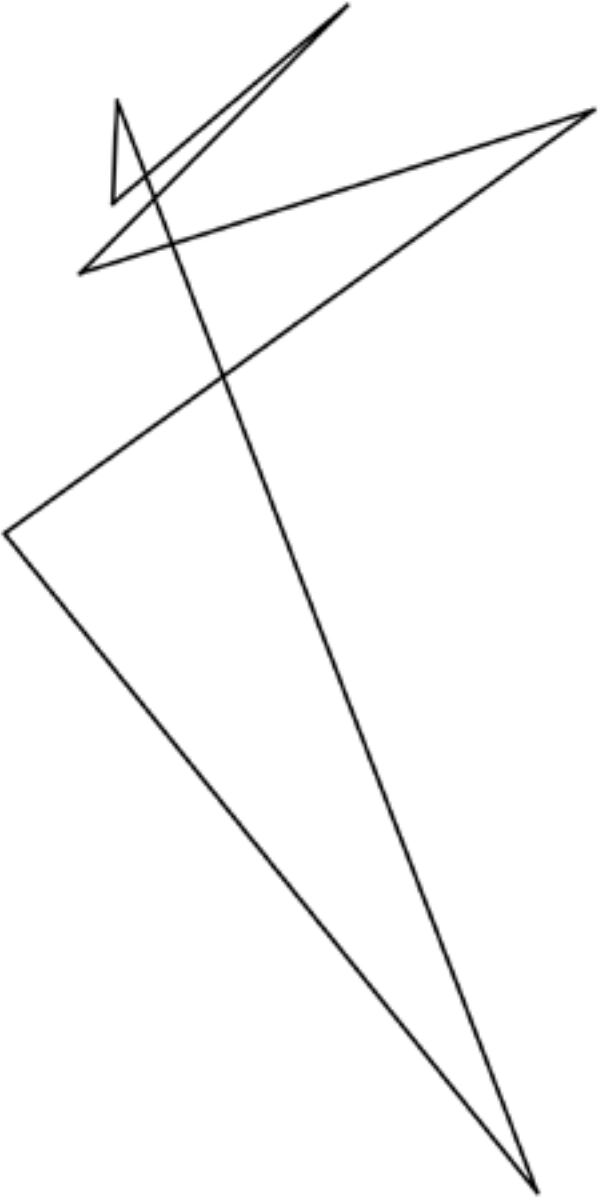
## The Seventh Line

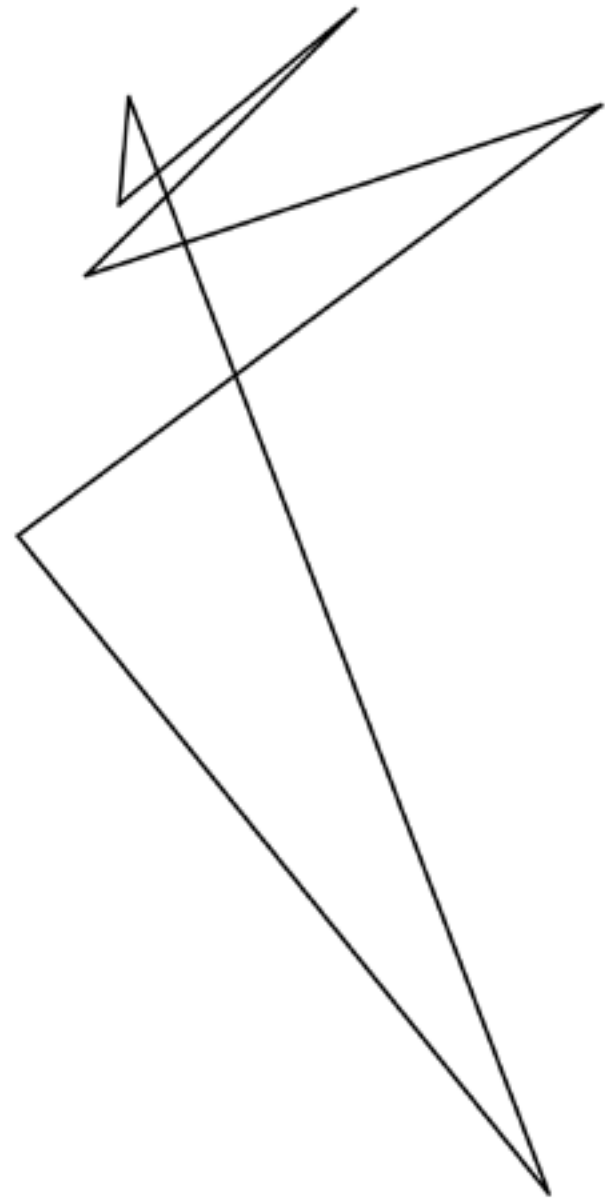
Instead of being interactive, the output could be moving. Shivering lines create chaos!

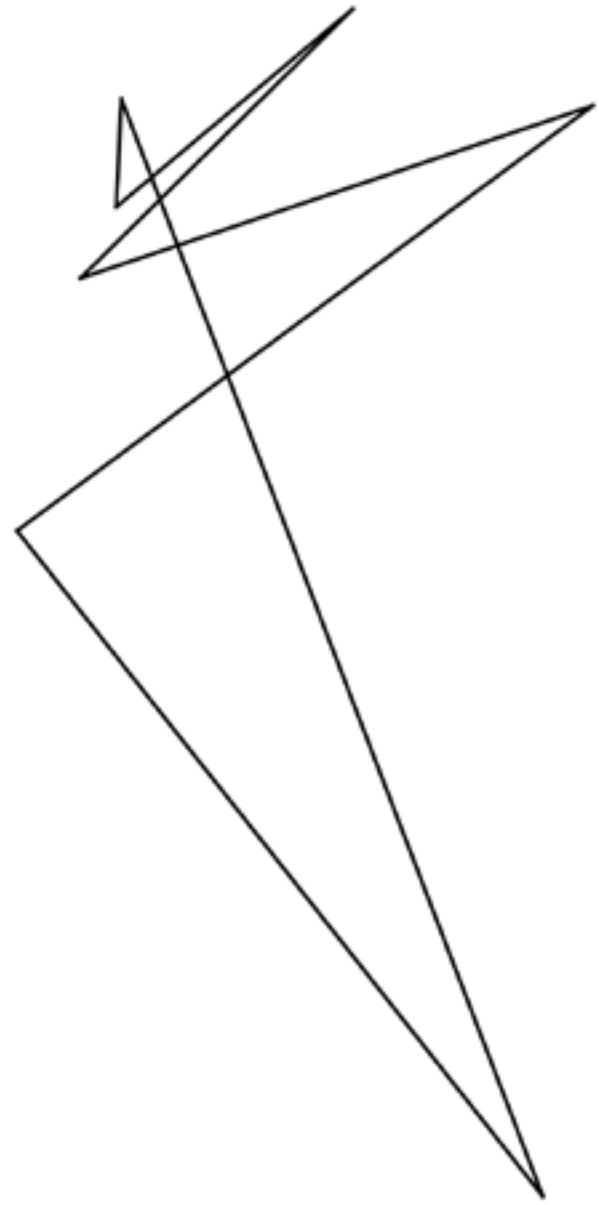
```
int shiver = 7;
for (int i = 0; i < howMany; i++)
{
    x[i] = x[i] + random(-shiver, shiver);
    y[i] = y[i] + random(-shiver, shiver);
}
```

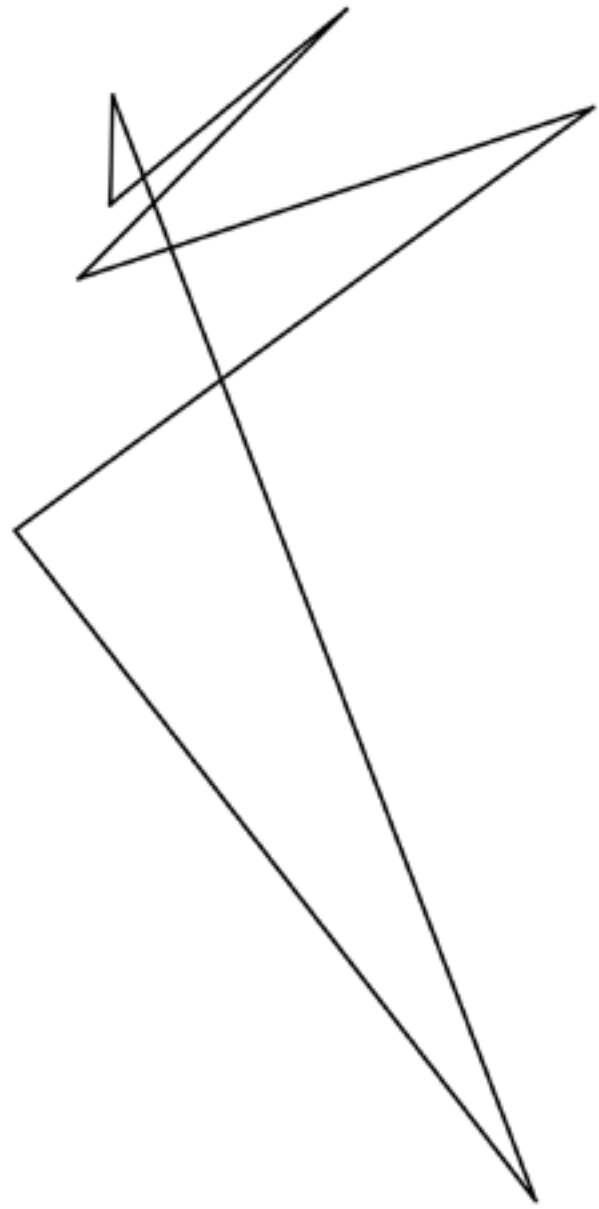
Fancy! It might even make sense, since I really like creating animations with lines. But is it enough?

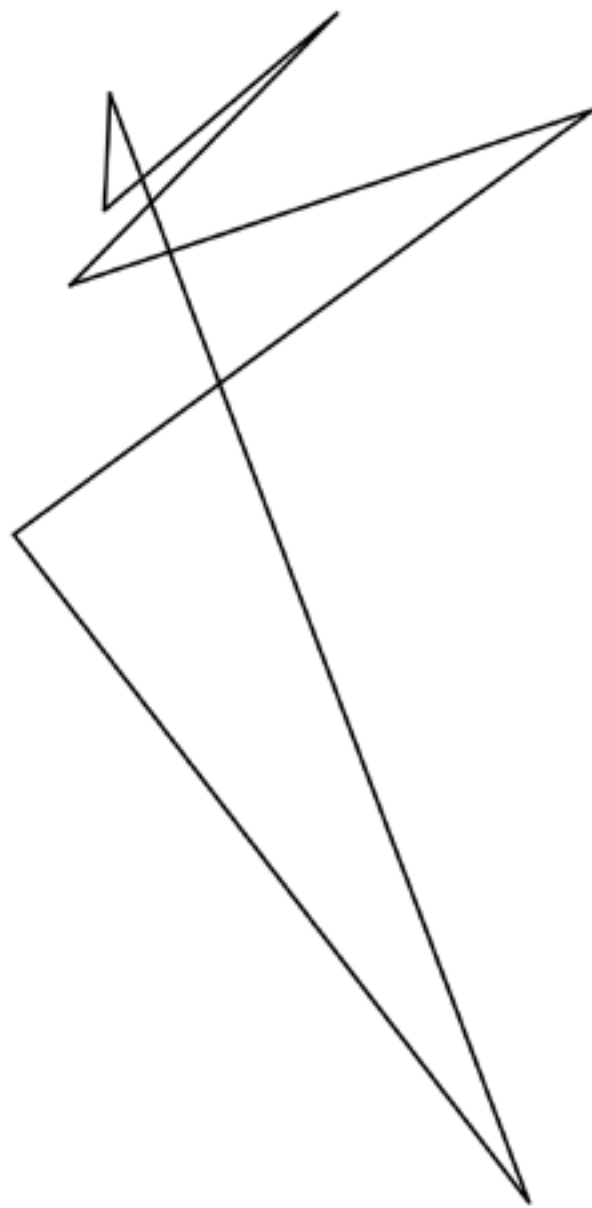


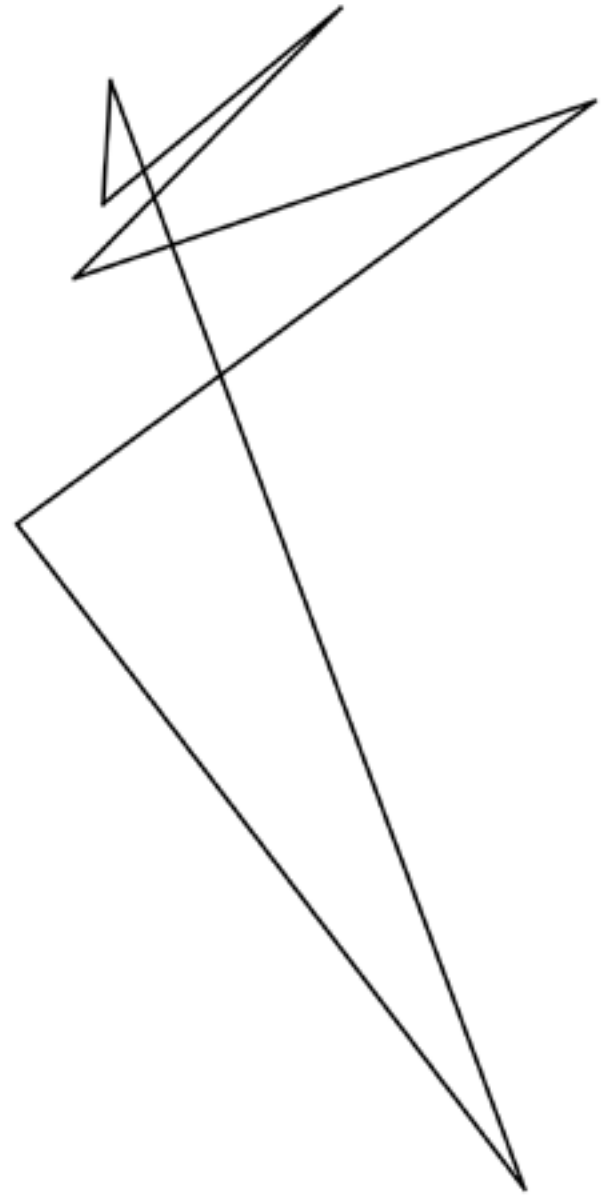










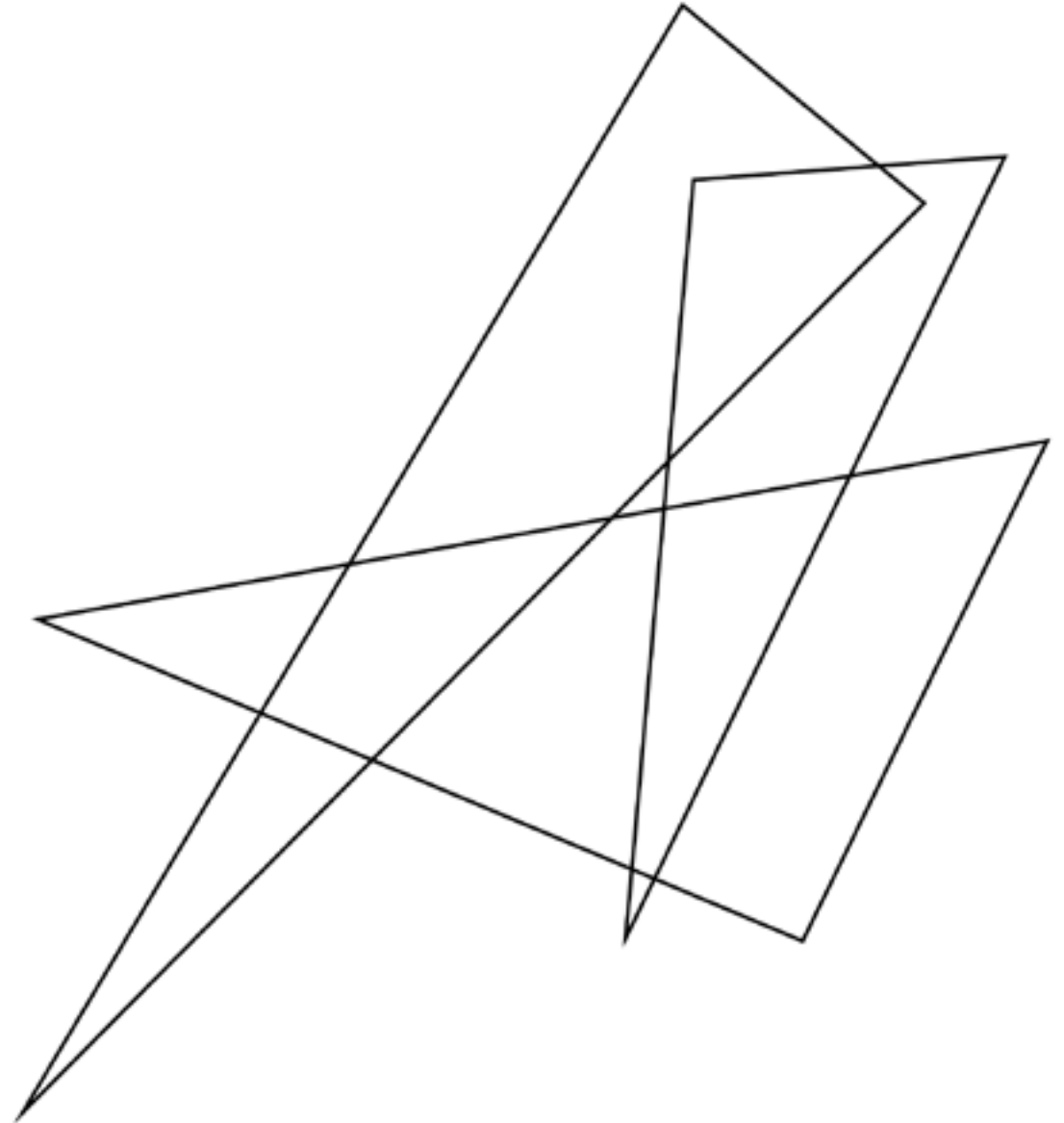


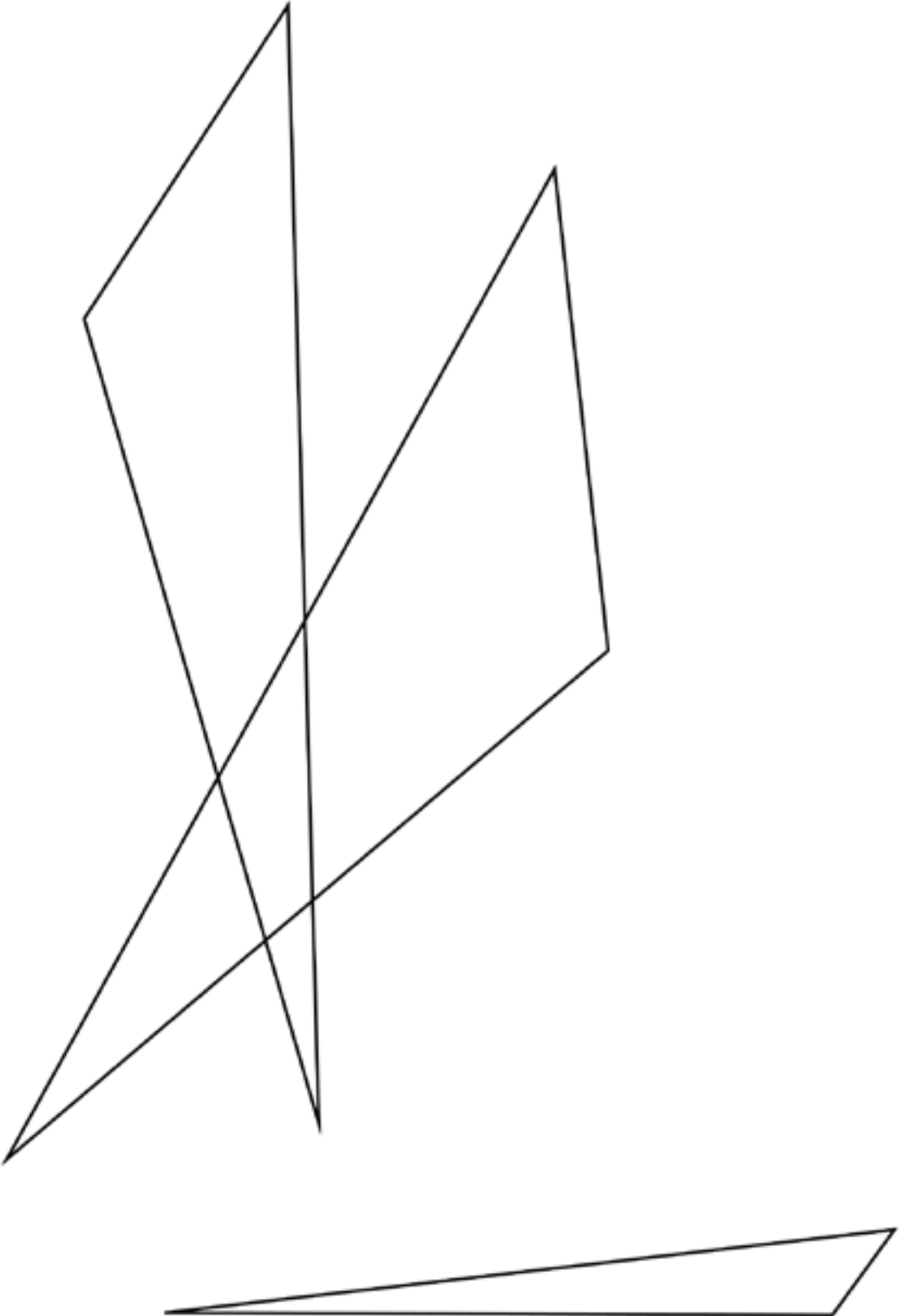
## The Ninth Line

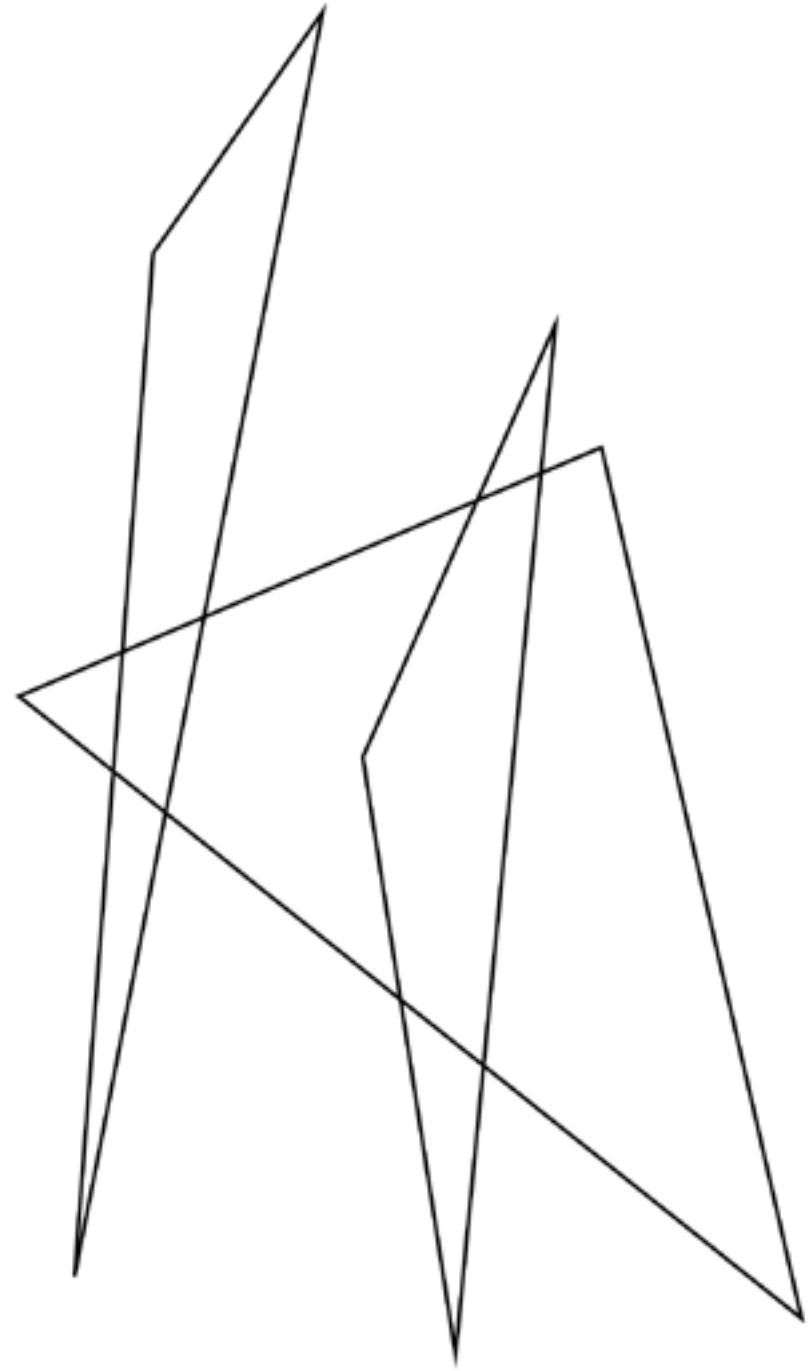
The square of three is nine. If we disconnect them partly (ha!), we can draw three beautiful triangles.

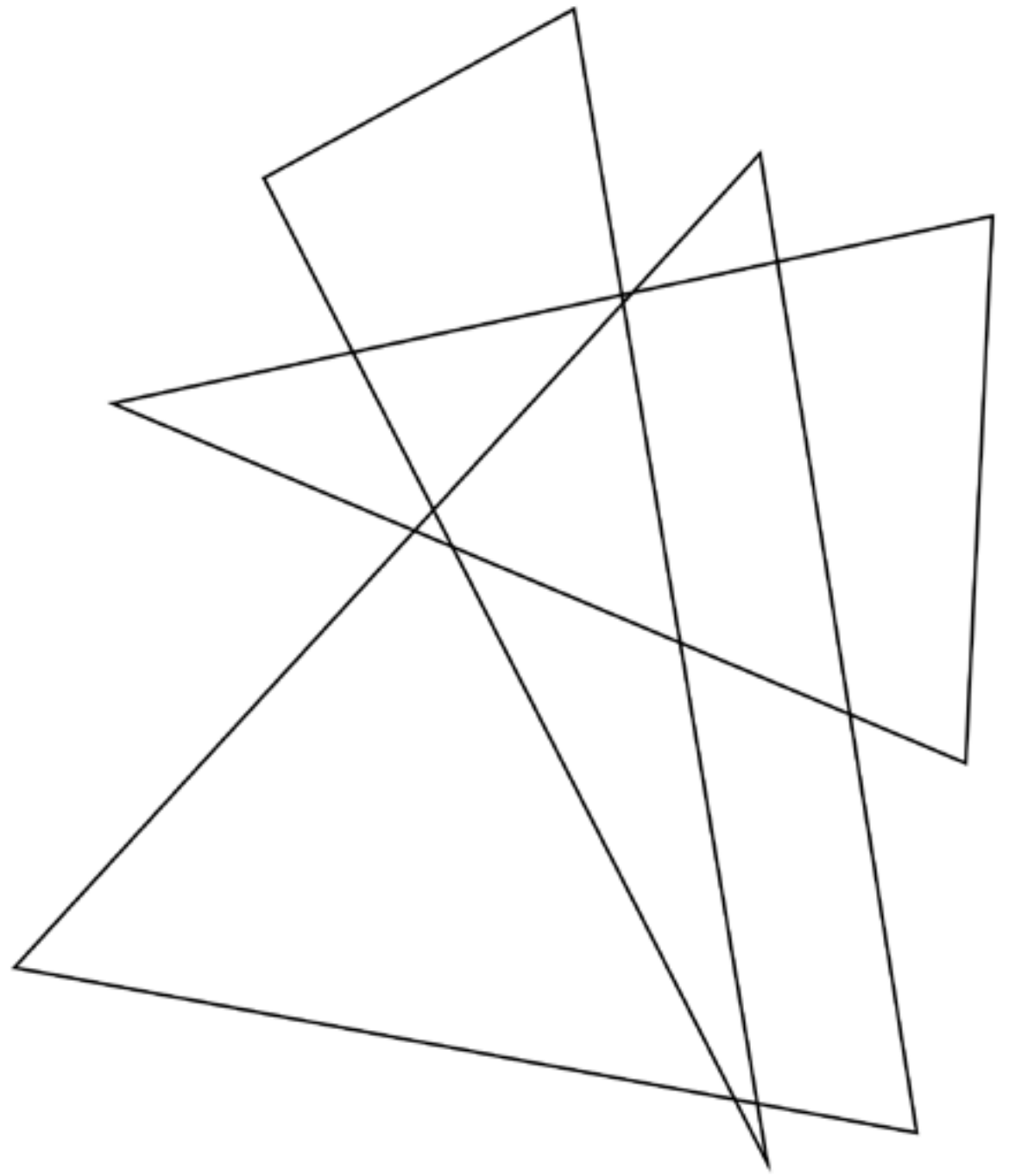
```
for (int i = 0; i < 3; i++)  
{  
    float x1 = random(0, width), y1 = random(0, height);  
    float x2 = random(0, width), y2 = random(0, height);  
    float x3 = random(0, width), y3 = random(0, height);  
    triangle(x1, y1, x2, y2, x3, y3);  
}
```

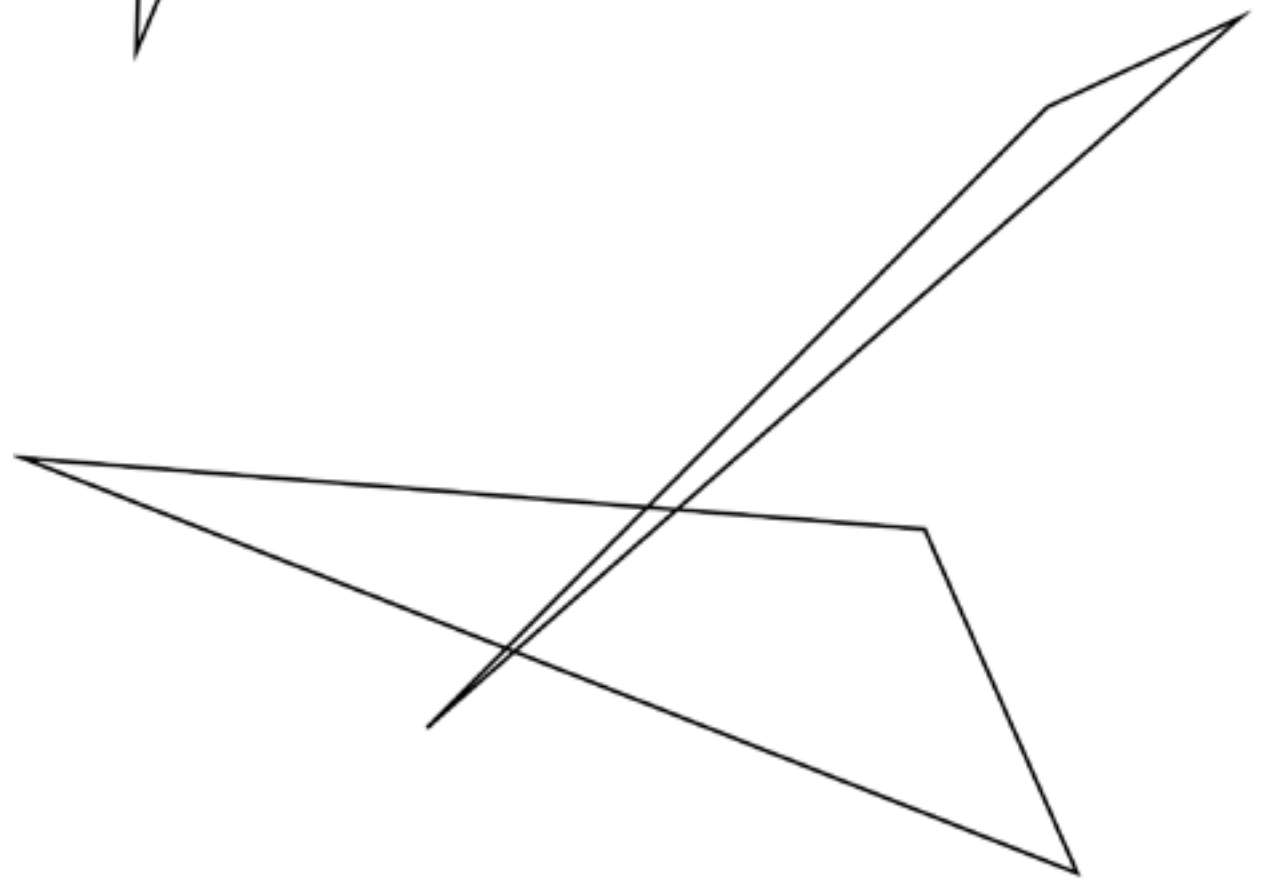
So far we have seen a point, some lines, triangles, quadrilaterals and even a few circles created with random choices inside our canvas. What are these random choices? Algorithms, obviously, creating pseudo random numbers. An examination of these algorithms might be interesting on a larger scale, but for now we will happily take these deterministic created numbers.

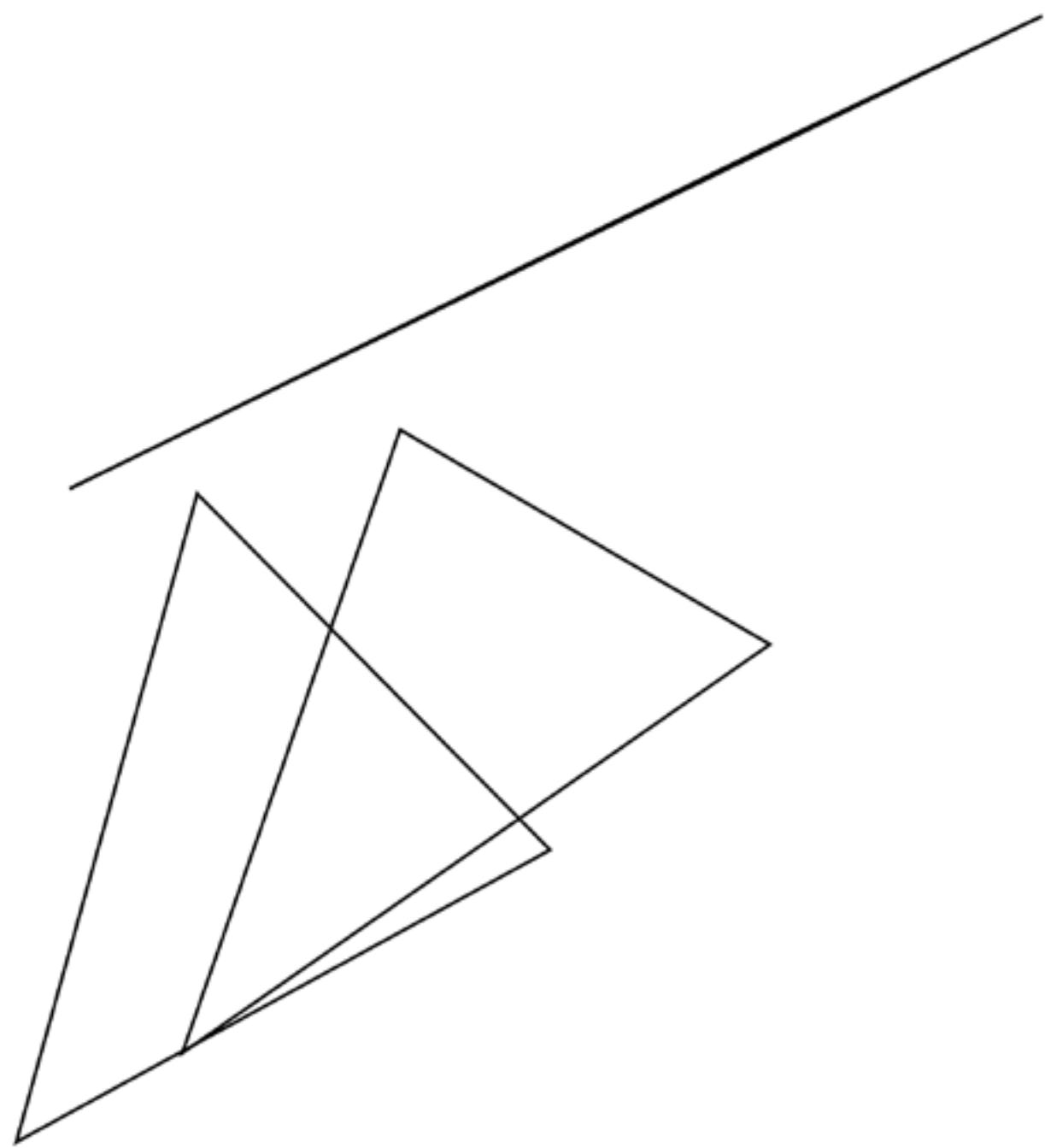


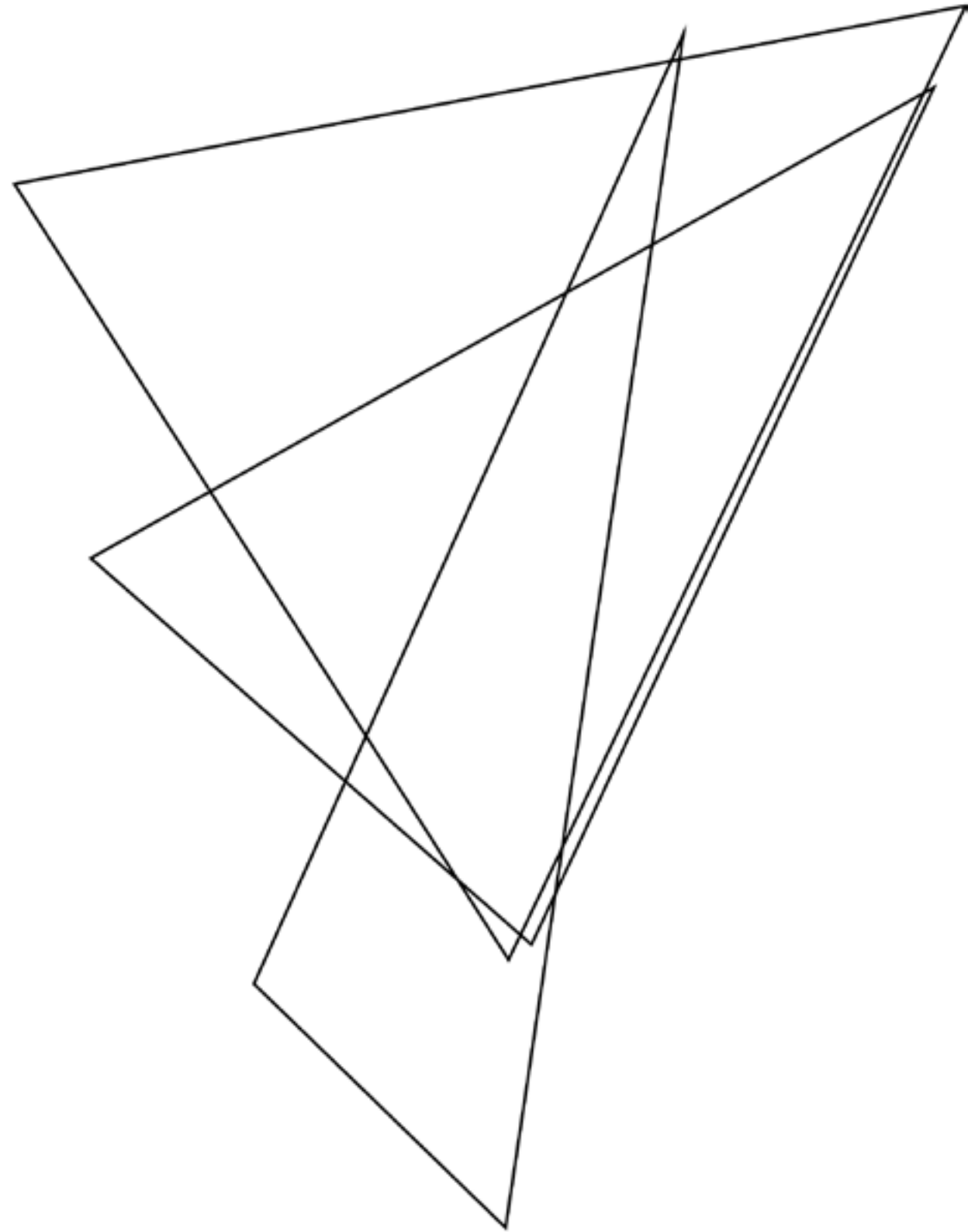


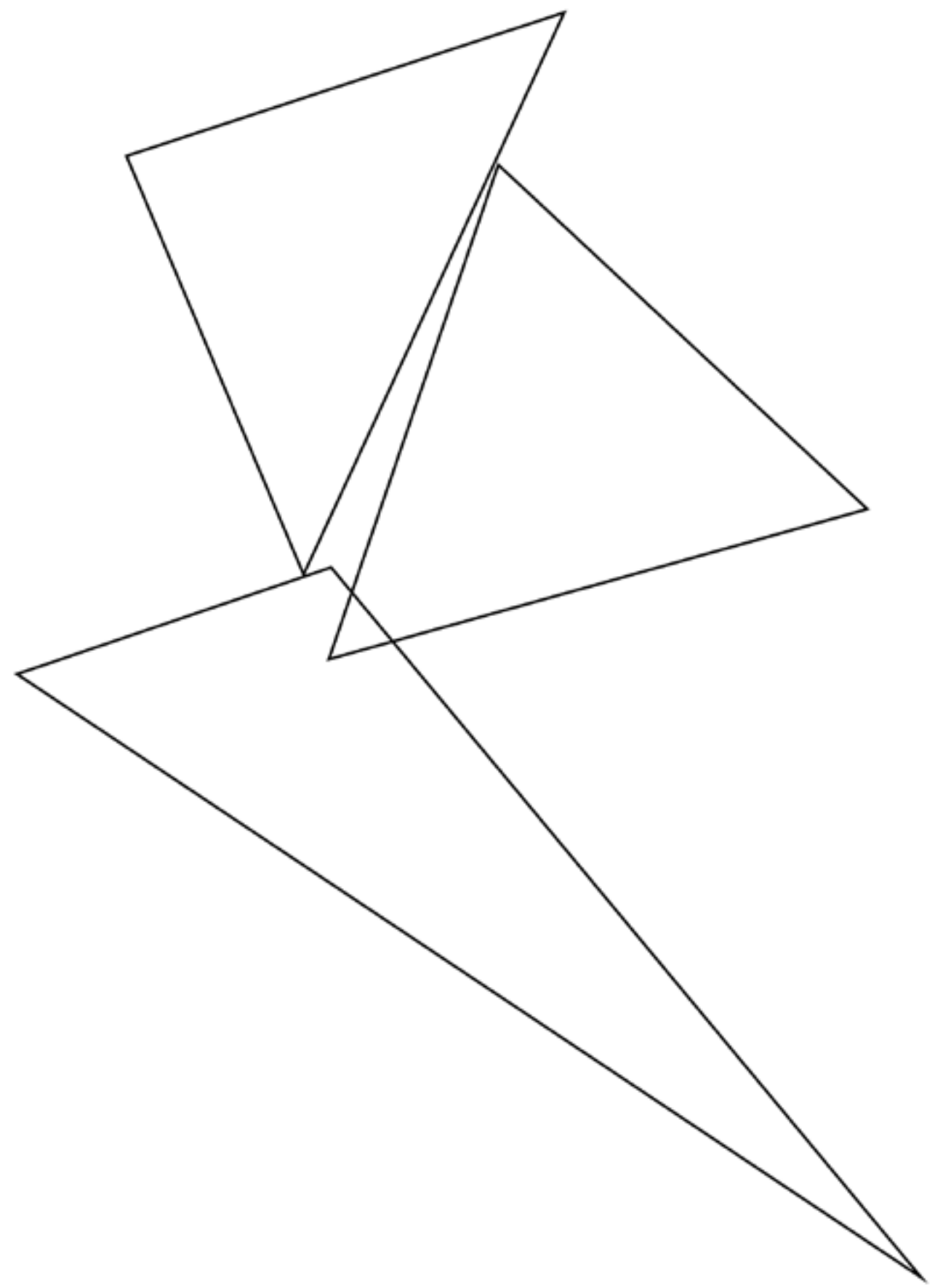


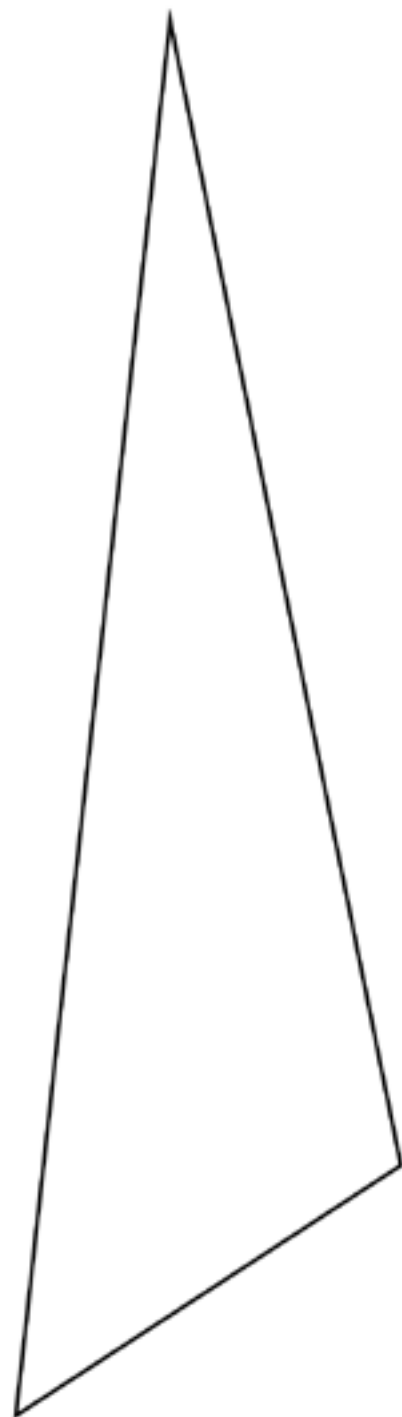
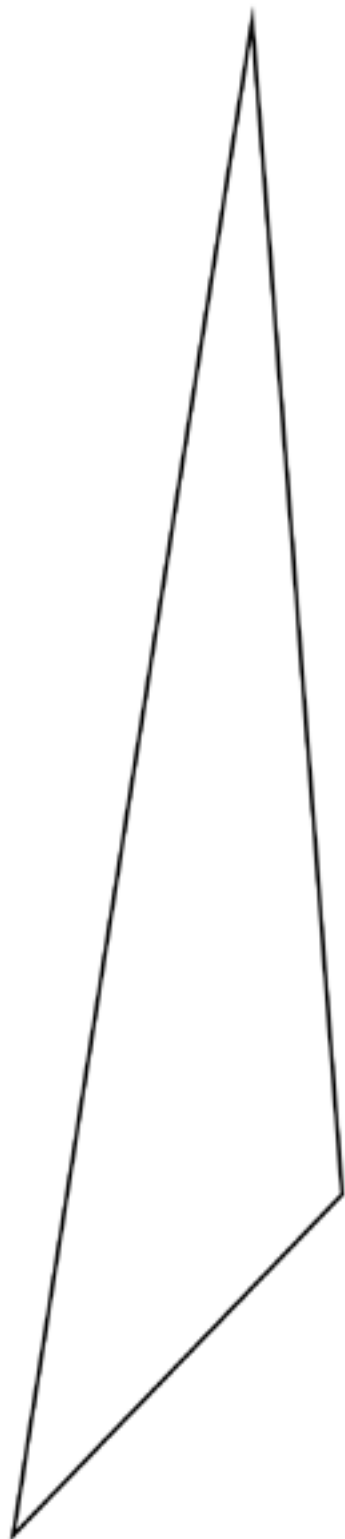






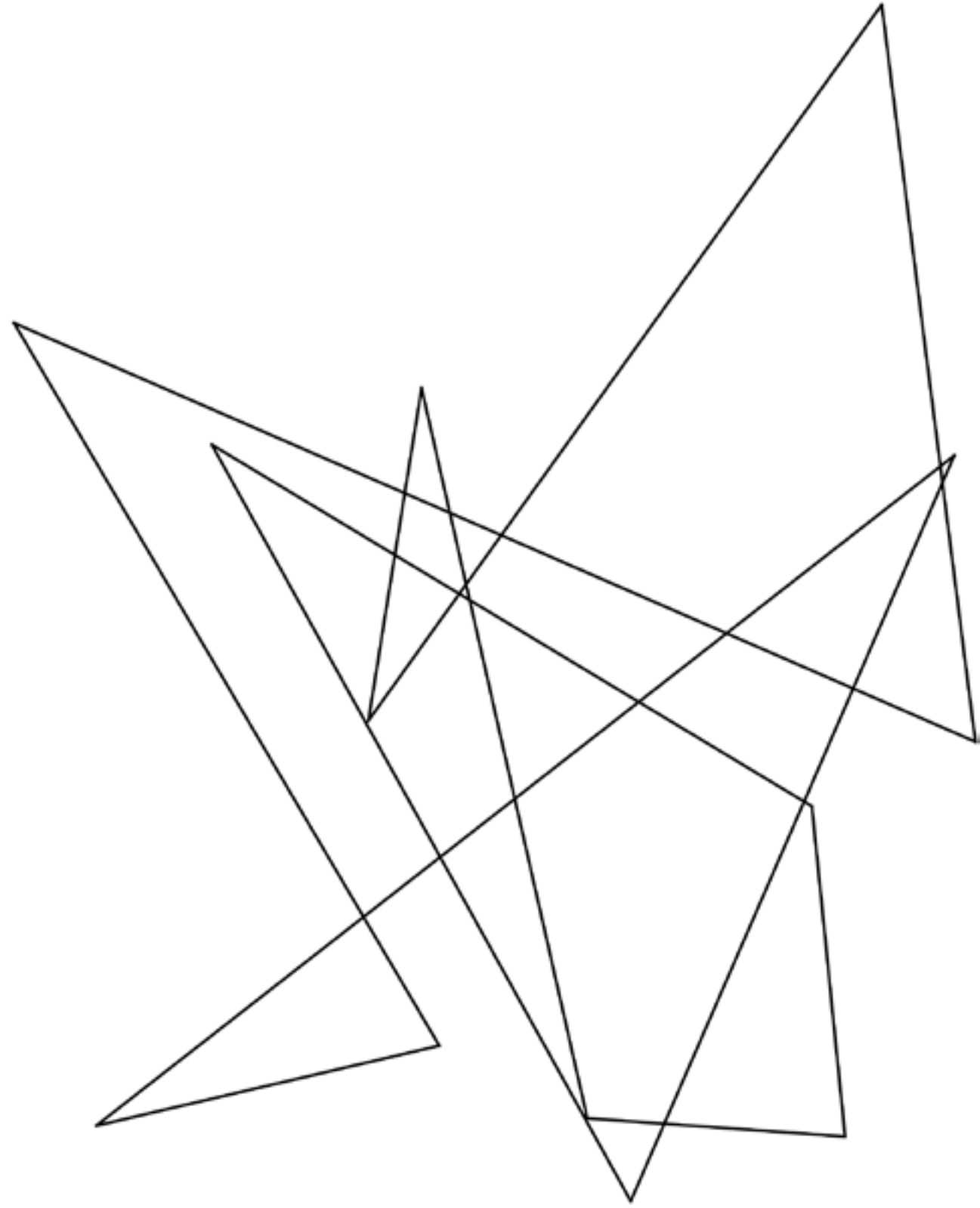


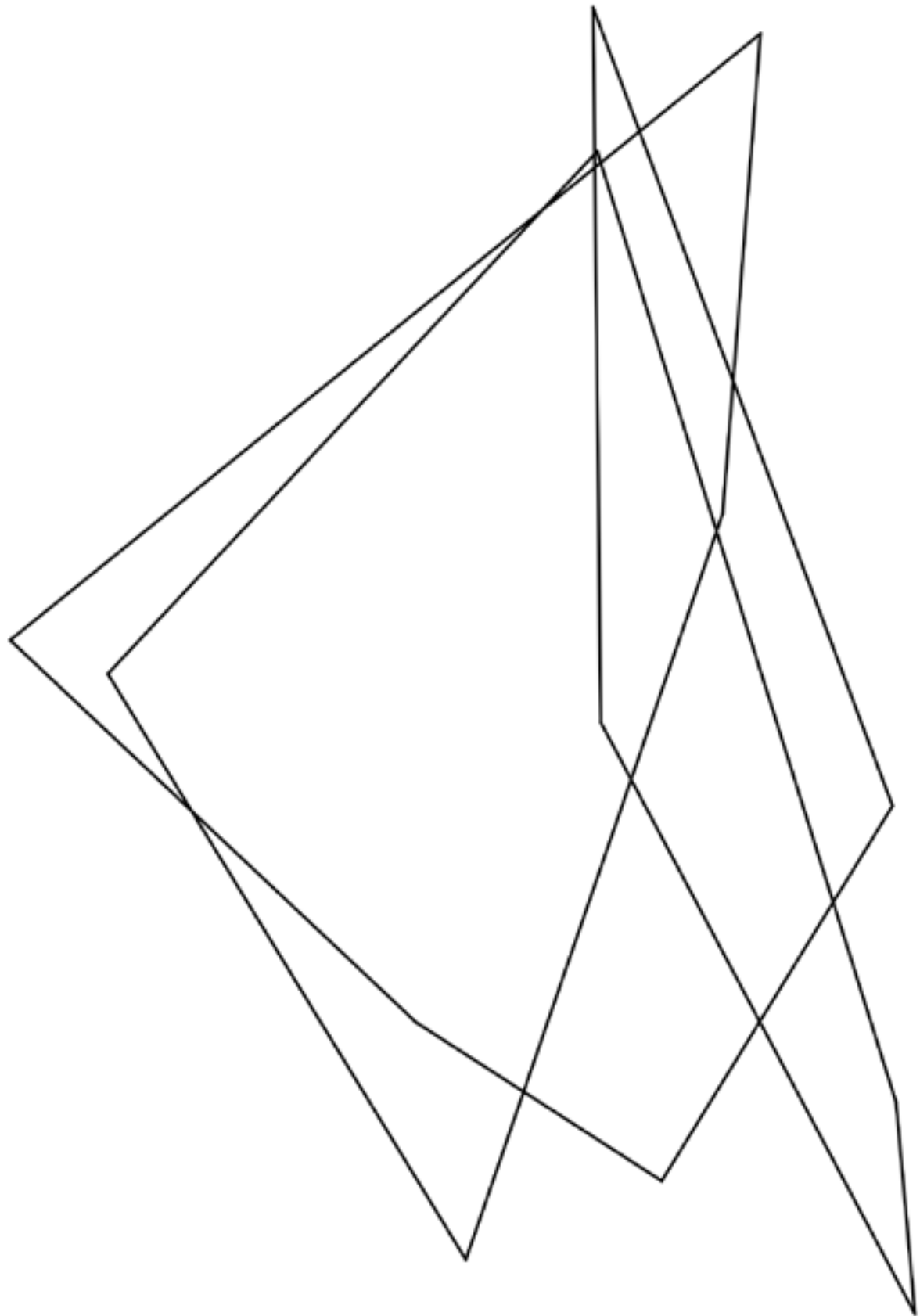


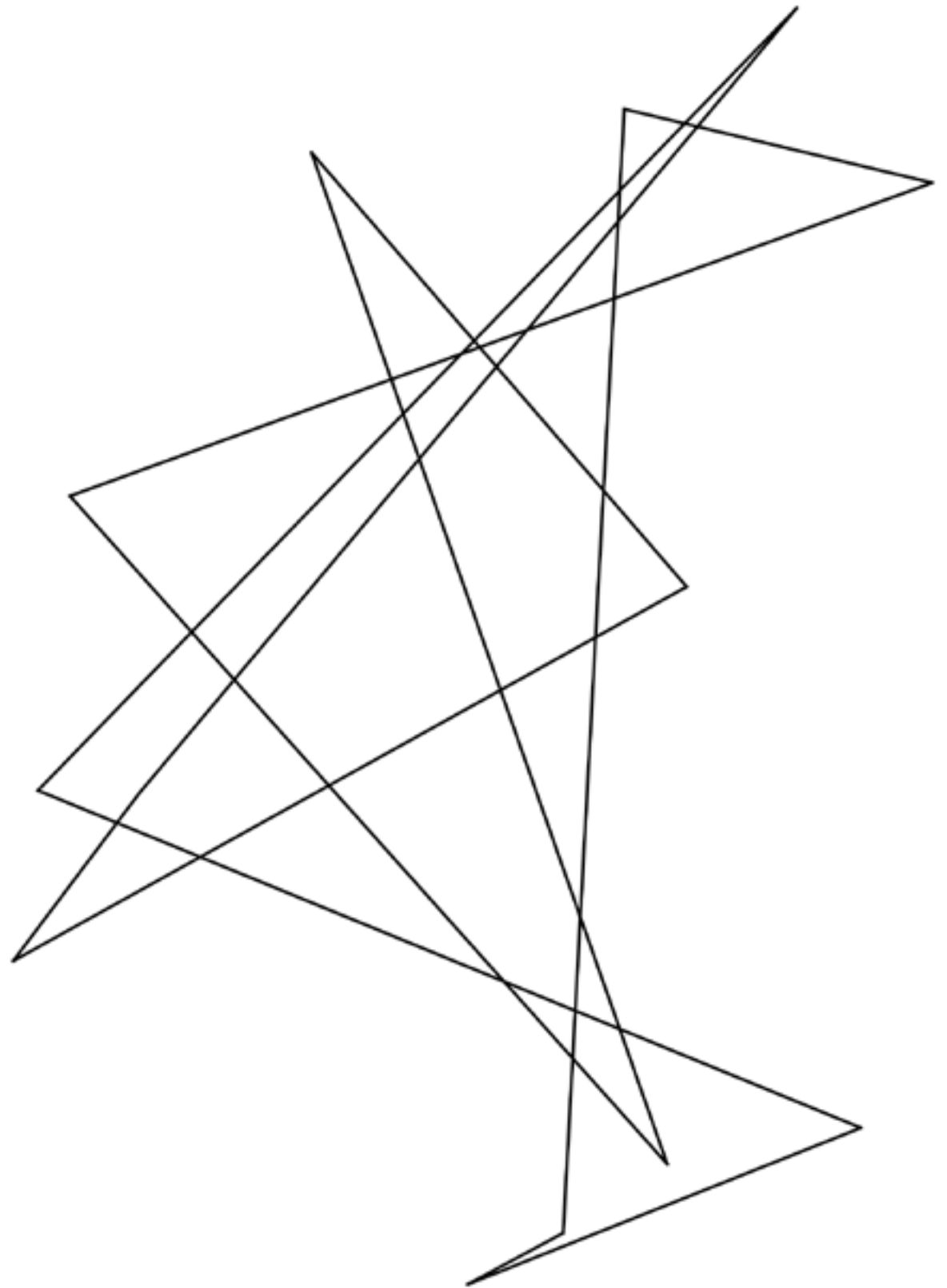


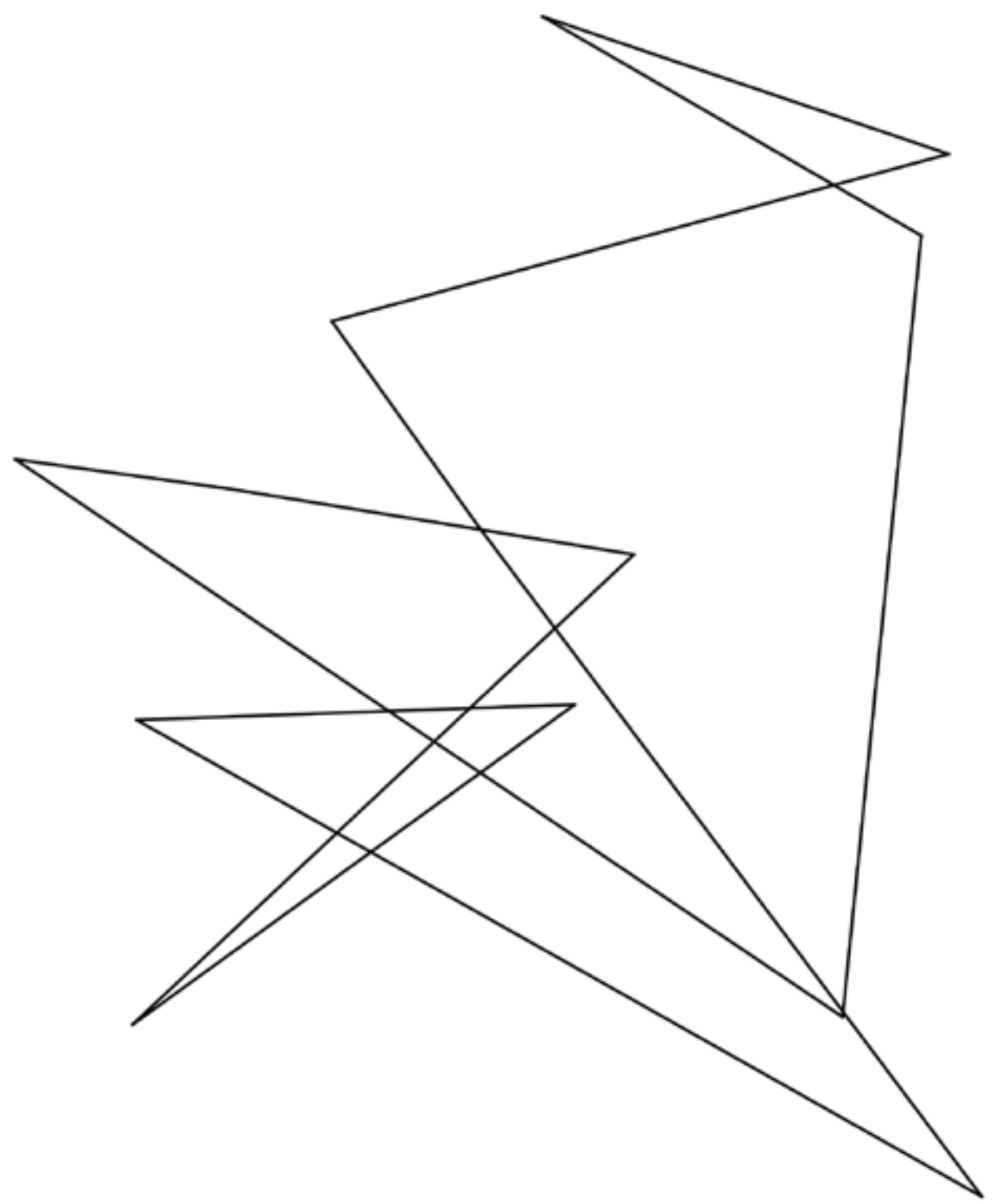
## The Thirteenth Line

Instead of being inspired by Frieder, we could use him as some kind of input. A deviation could be created with rectangles, a primitive would even include some curvy lines. Creating an integration, mirroring the function. Algebra offers a lot of possibilities. They seem quite funny and Frieder might even like the idea. Yet the question: Is this a homage? Or is this just playing around with Frieder's lines? Maybe it is the same. But simply using some random algebra is not enough.









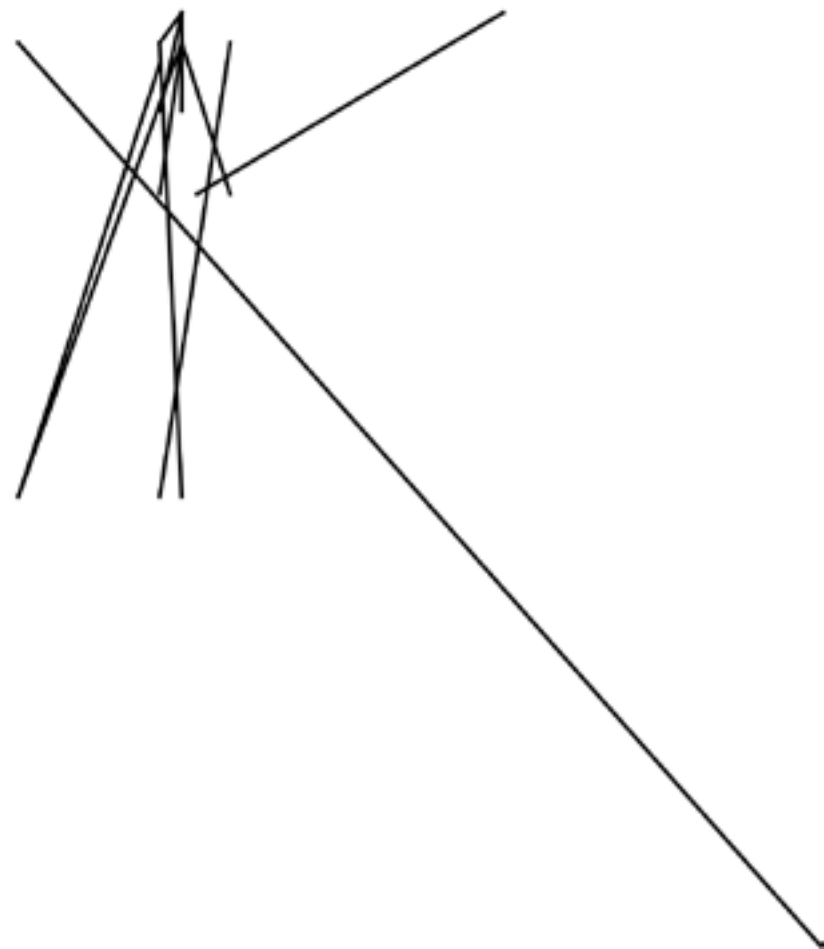
## The Fourteenth Line

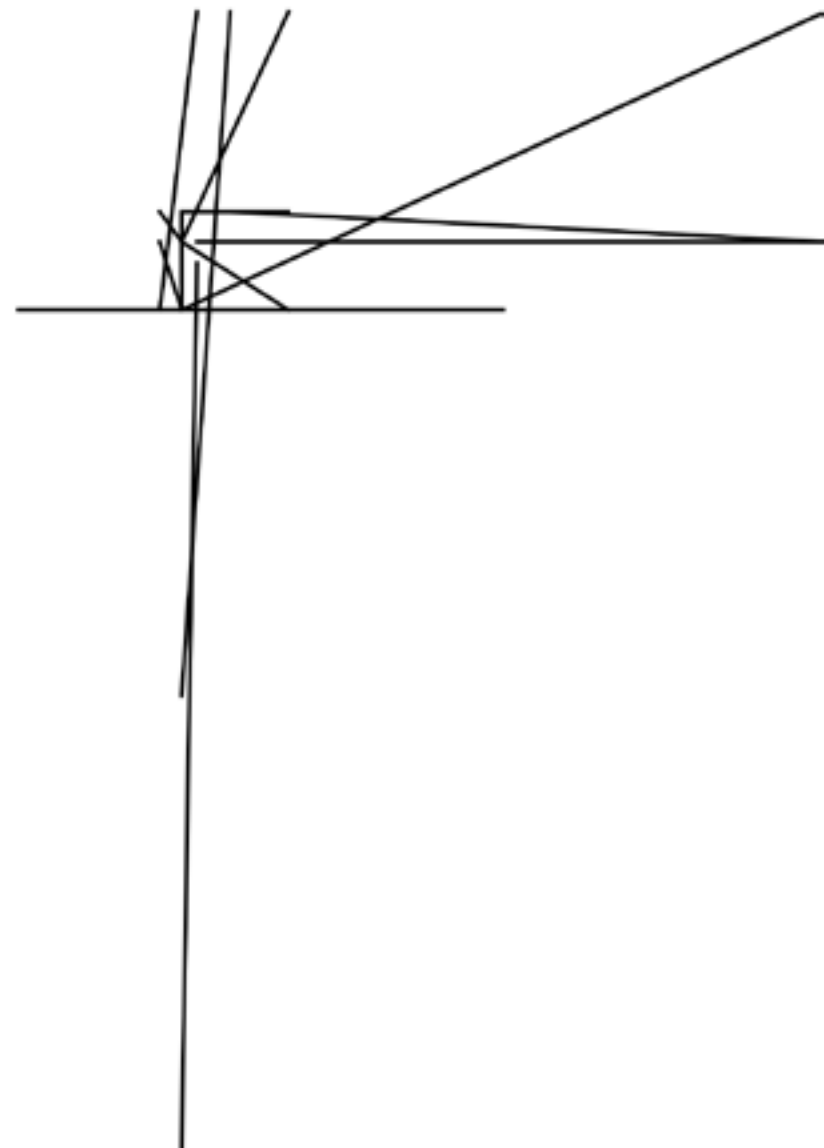
What does Frieder stand for in the algorithmic dimension? He stands for the beginning of algorithmic art. He *is* the beginning of algorithmic art? Thinking about beginnings and algorithms Ada Lovelace pops up. She is the beginning of modern algorithms. Her first algorithm was the calculation of the Bernoulli numbers. We could calculate them once and connect them over and over again. The outcomes will look different while still belonging together.

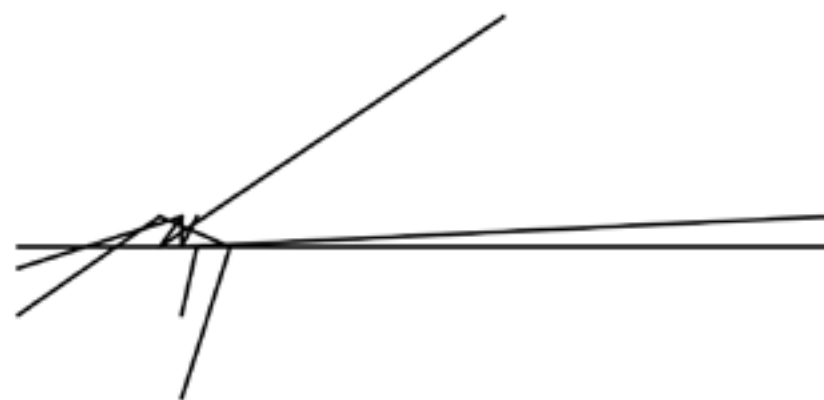
```
for (int i = 0; i < howMany; i++)  
{  
  x1 = bernoulliX[int(random(0, howMany-1))];  
  y1 = bernoulliY[int(random(0, howMany-1))];  
  x2 = bernoulliX[int(random(0, howMany-1))];  
  y2 = bernoulliY[int(random(0, howMany-1))];  
  line(x1, y1, x2, y2);  
}
```

This is a nice connection between Ada and Frieder!  
But where am I?

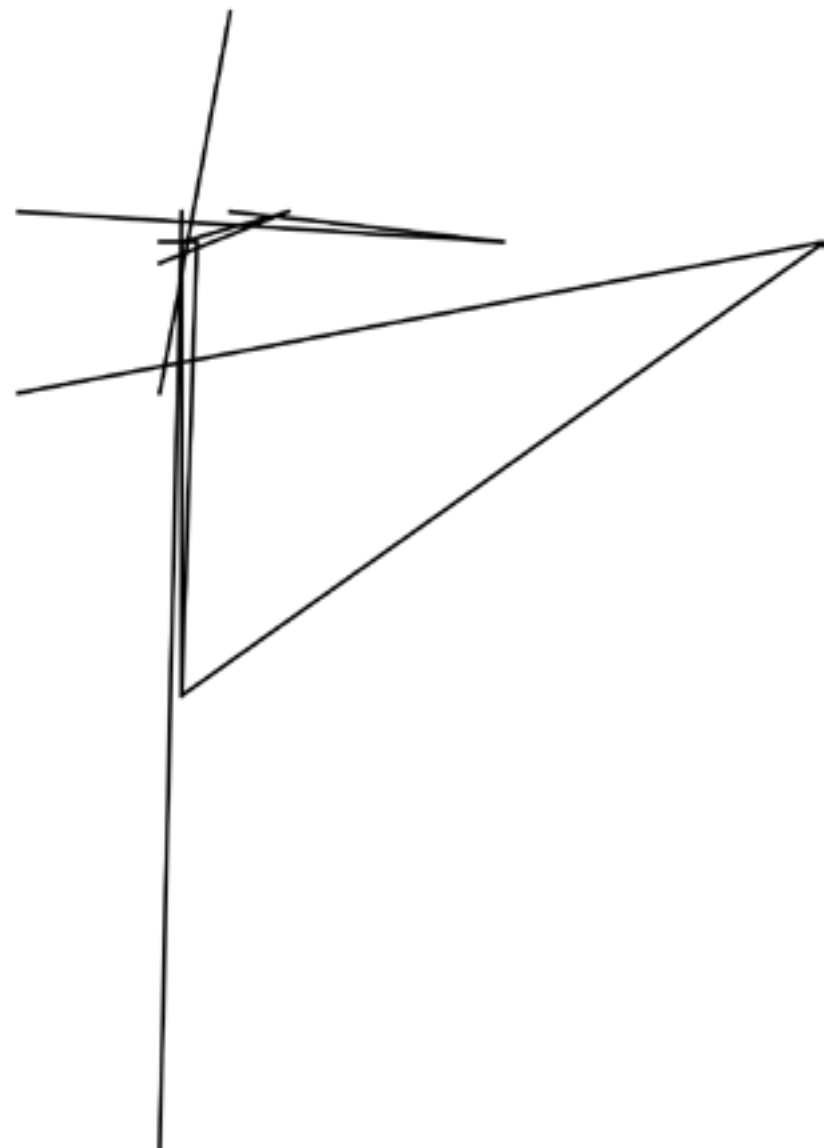








—

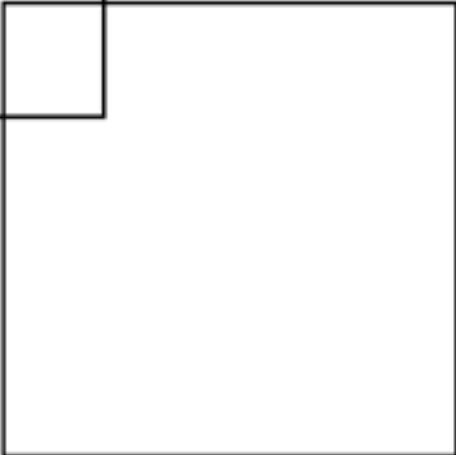
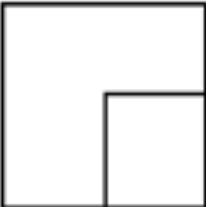


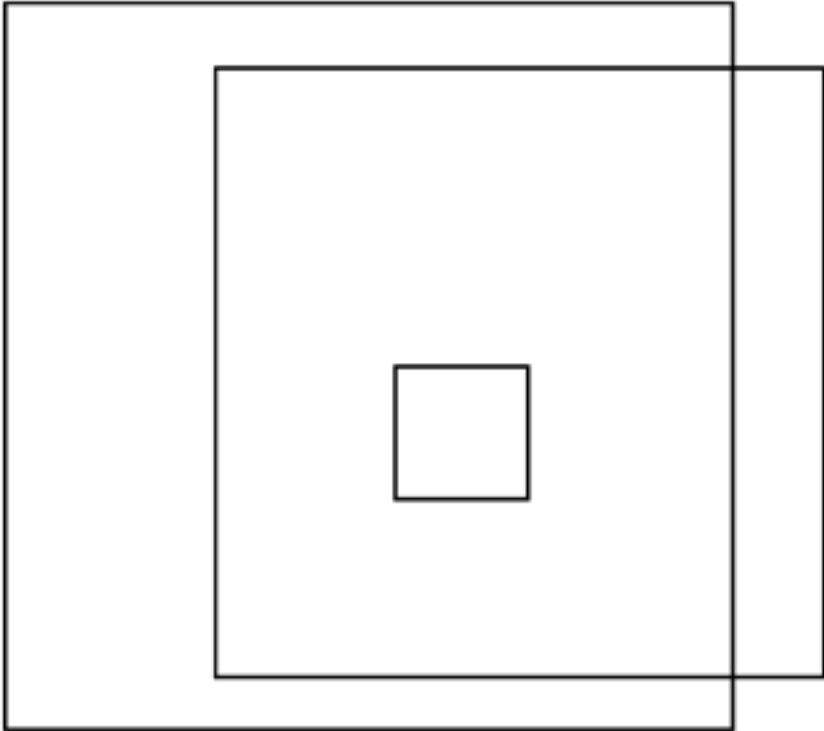
## The Sixteenth Line

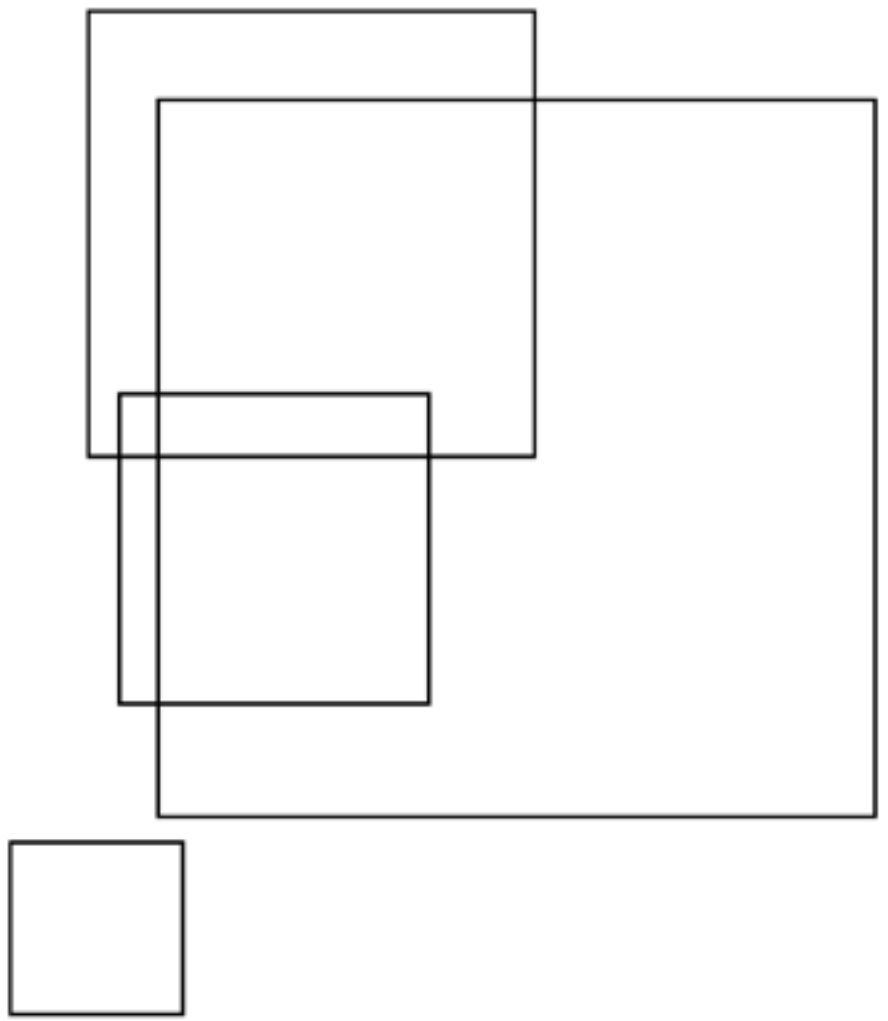
The square of two is one square. And the square of four is four squares. Wow!

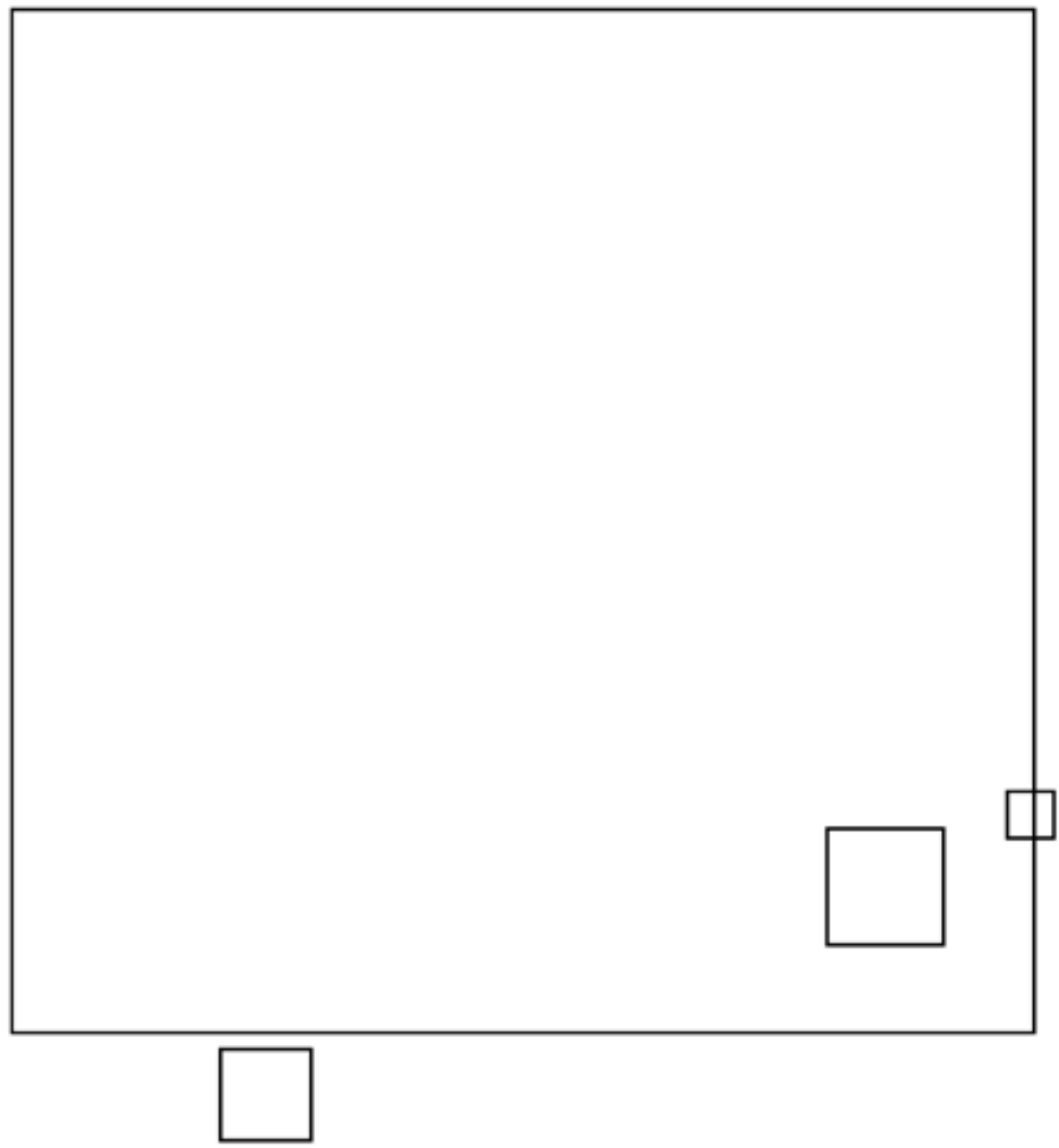
```
for (int i = 0; i < howMany; i++)  
{  
    float x = random(0, width);  
    float y = random(0, height);  
    float size;  
    if(width-x < height-y)  
    {  
        size = random(0, width-x);  
    }  
    else  
    {  
        size = random(0, height-y);  
    }  
    rect(x, y, size, size);  
}
```

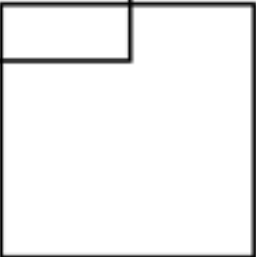
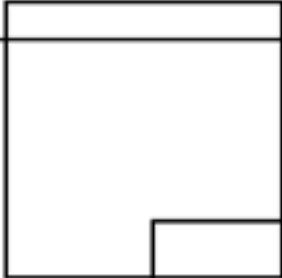
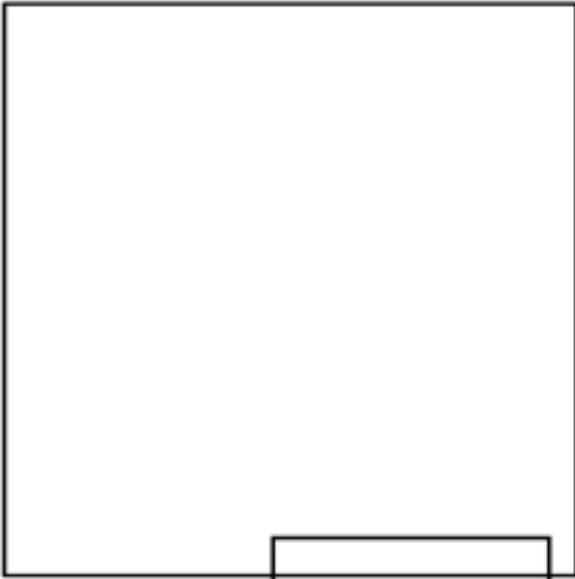


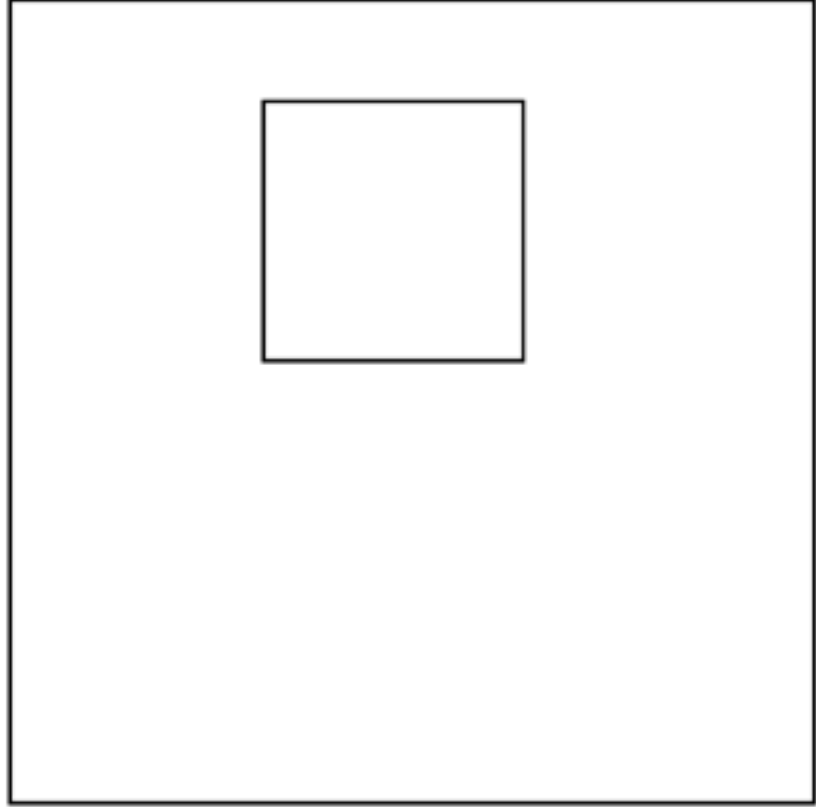










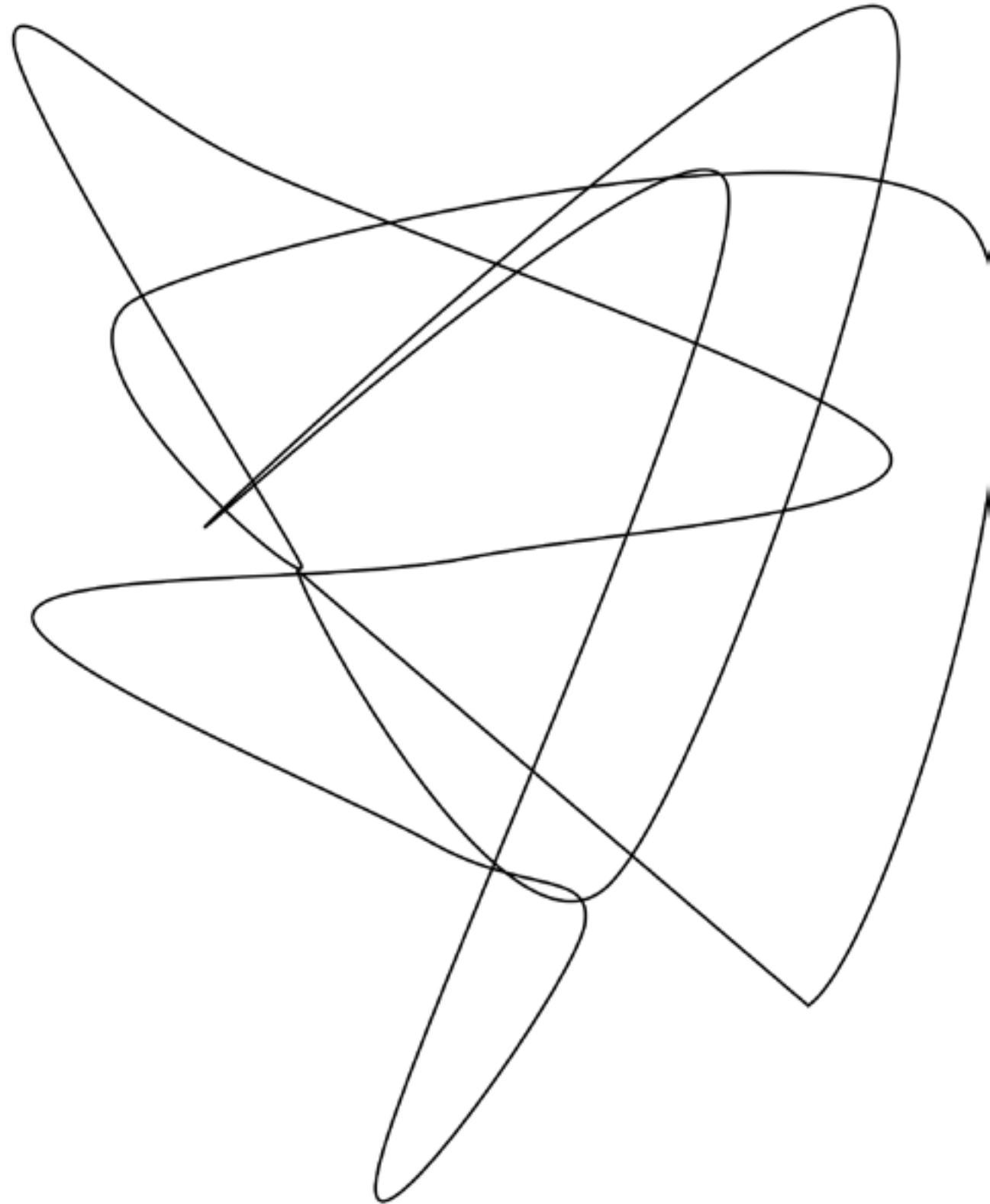


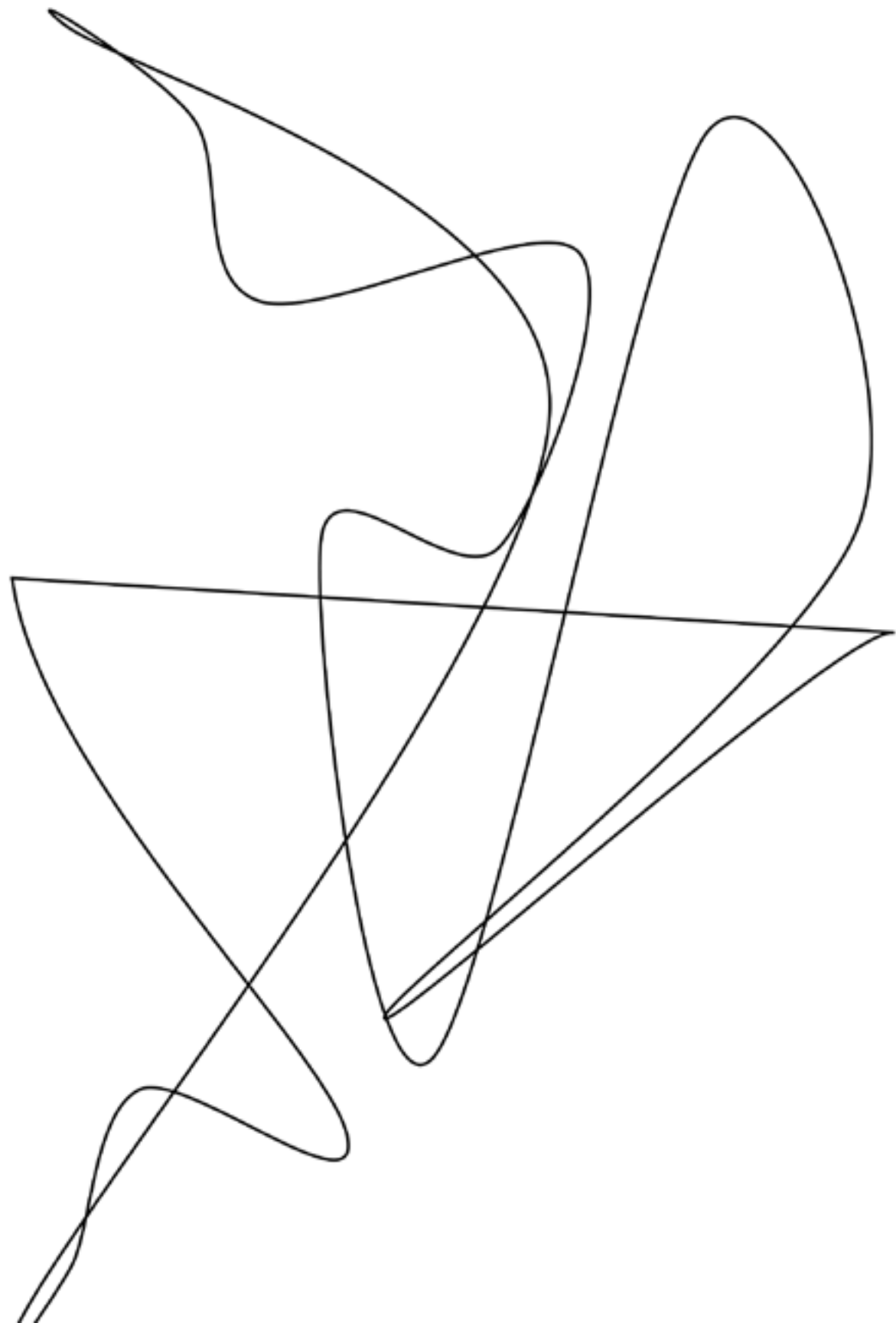
## The Seventeenth Line

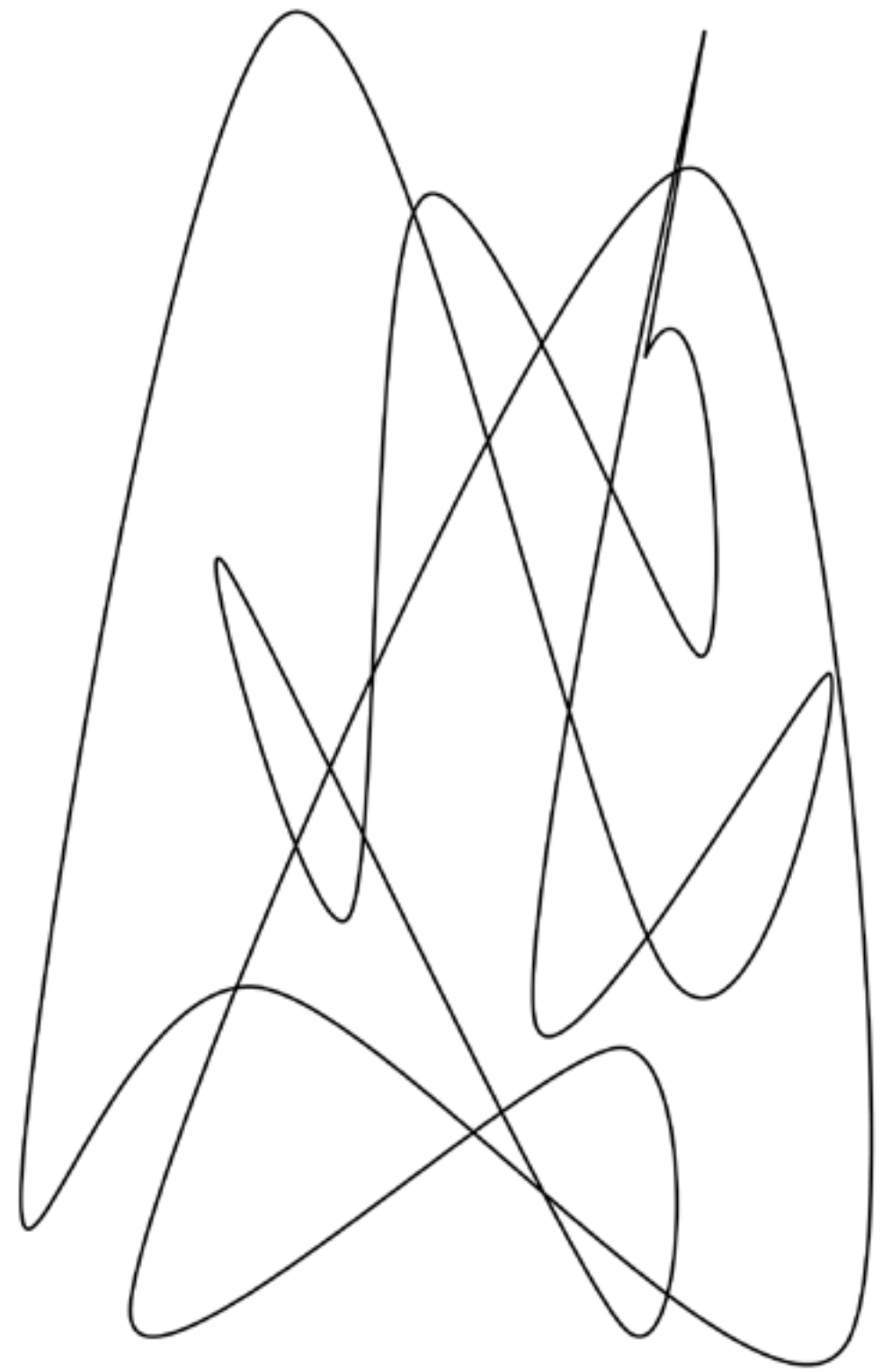
Since we left the circle behind, we did not have any other curvy objects yet.

```
for (int i = 3; i < howMany; i++)  
{  
    curve(x[i-3], y[i-3], x[i-2], y[i-2], x[i-1], y[i-1], x[i], y[i]);  
}  
line(x[1], y[1], x[howMany-2], y[howMany-2]);
```

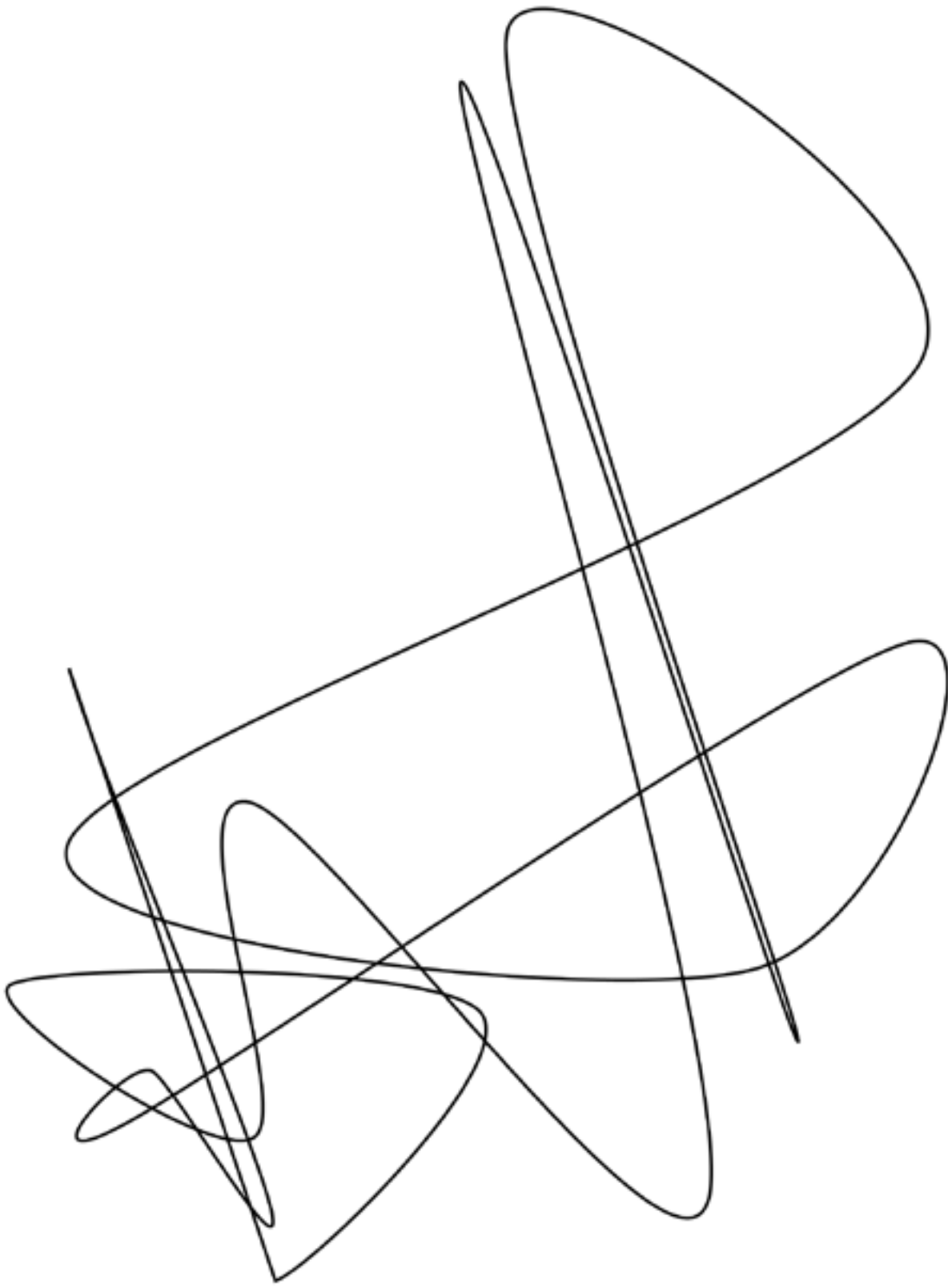
Now we do. At least for a moment. Hi Picasso!

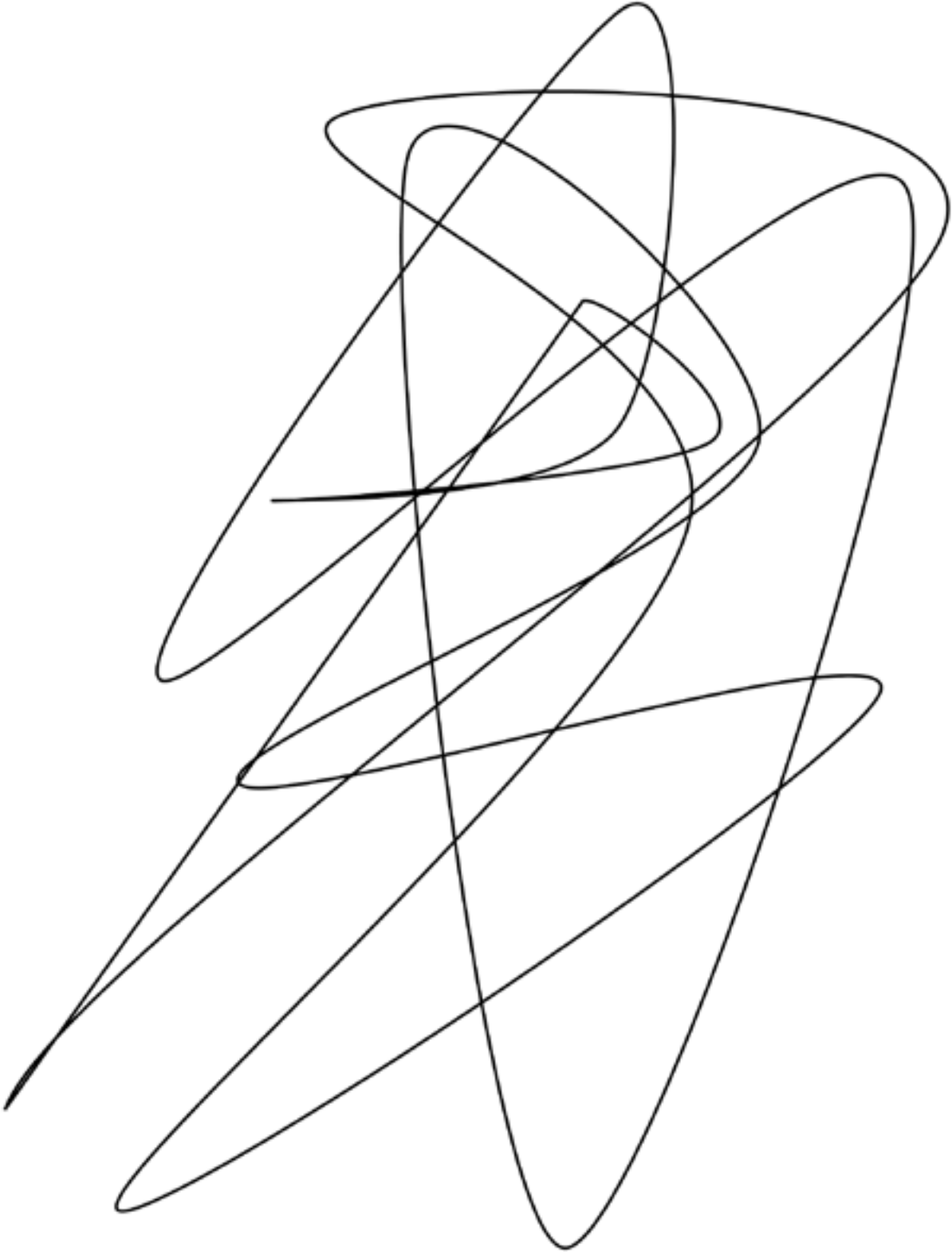


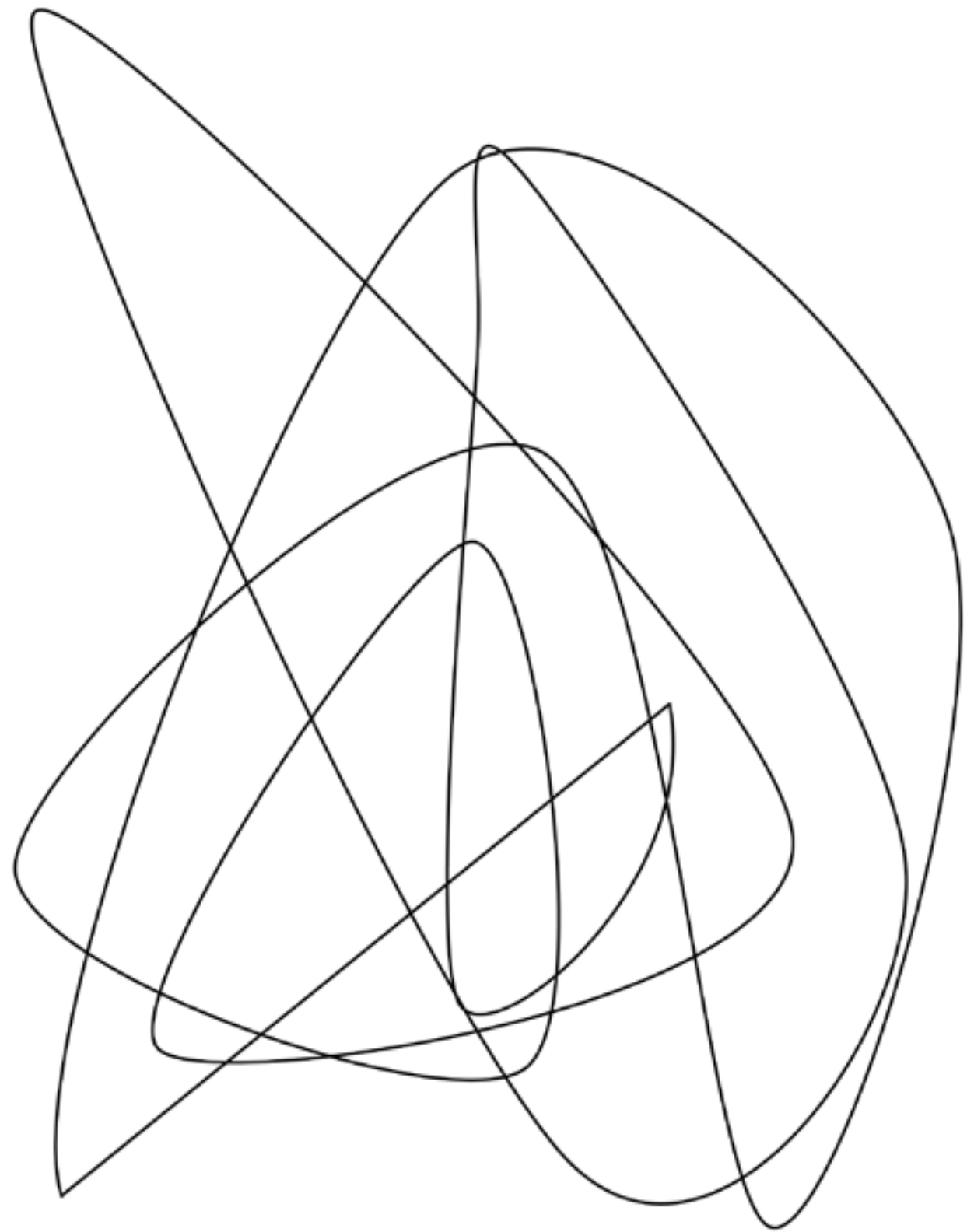


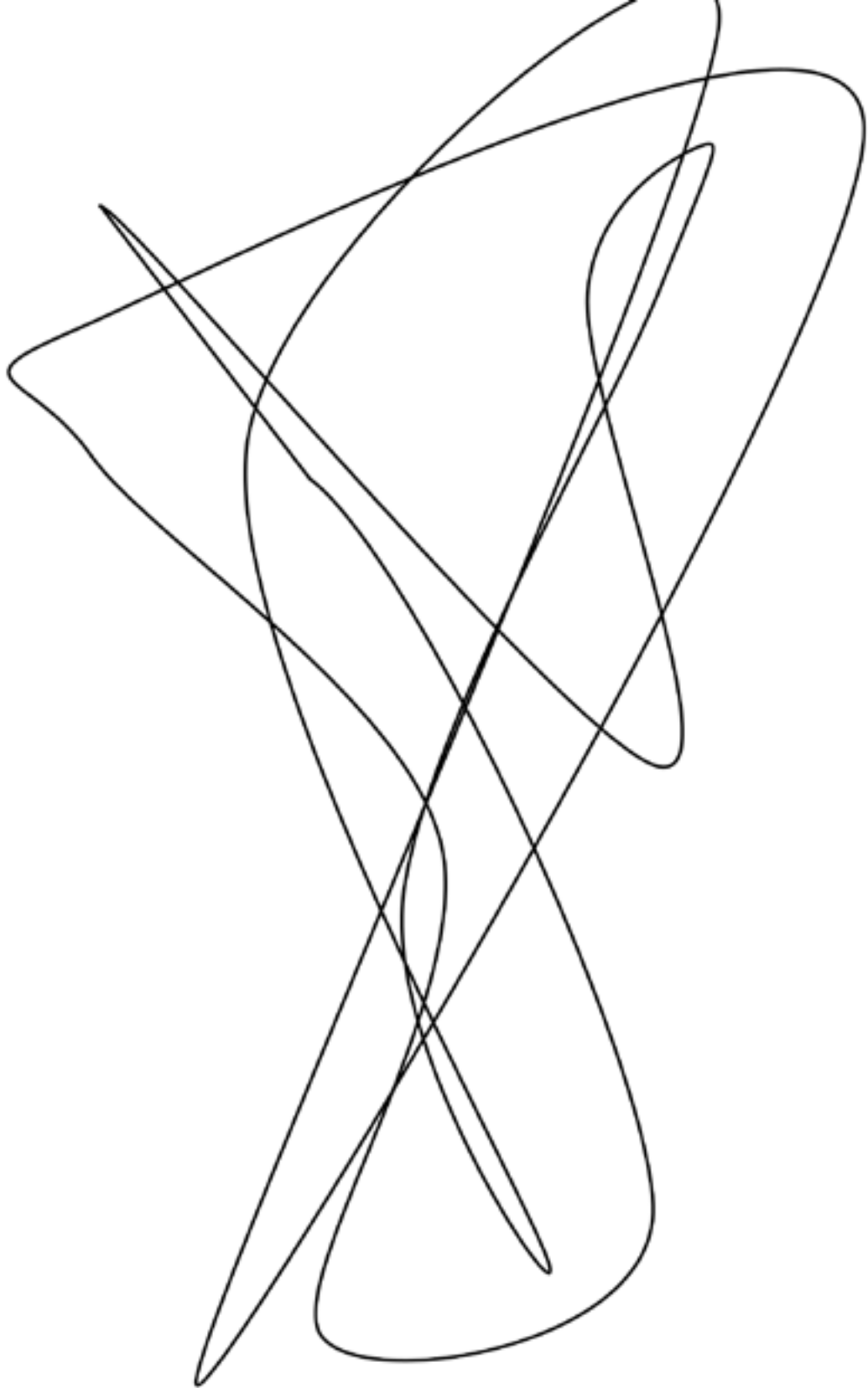








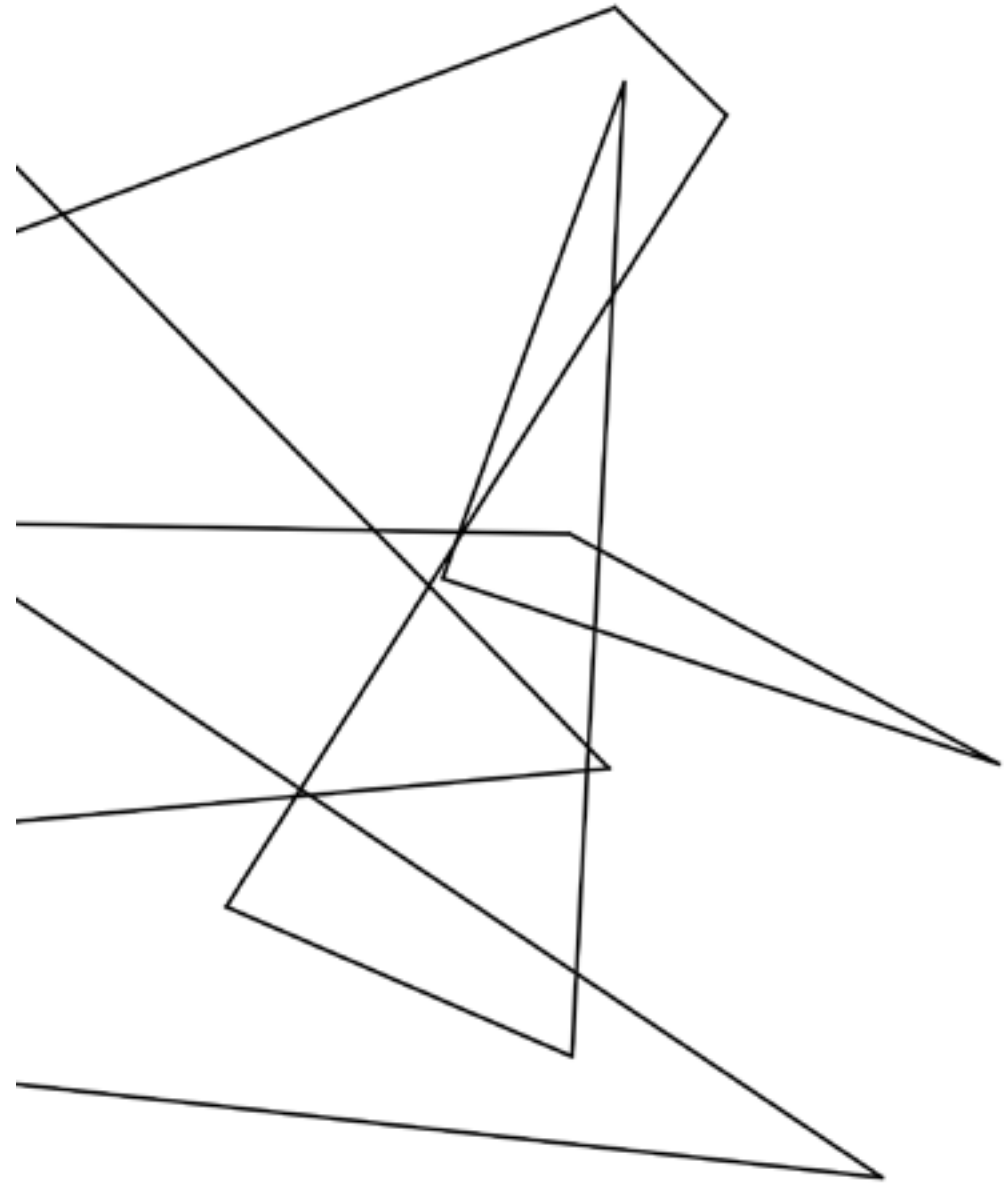


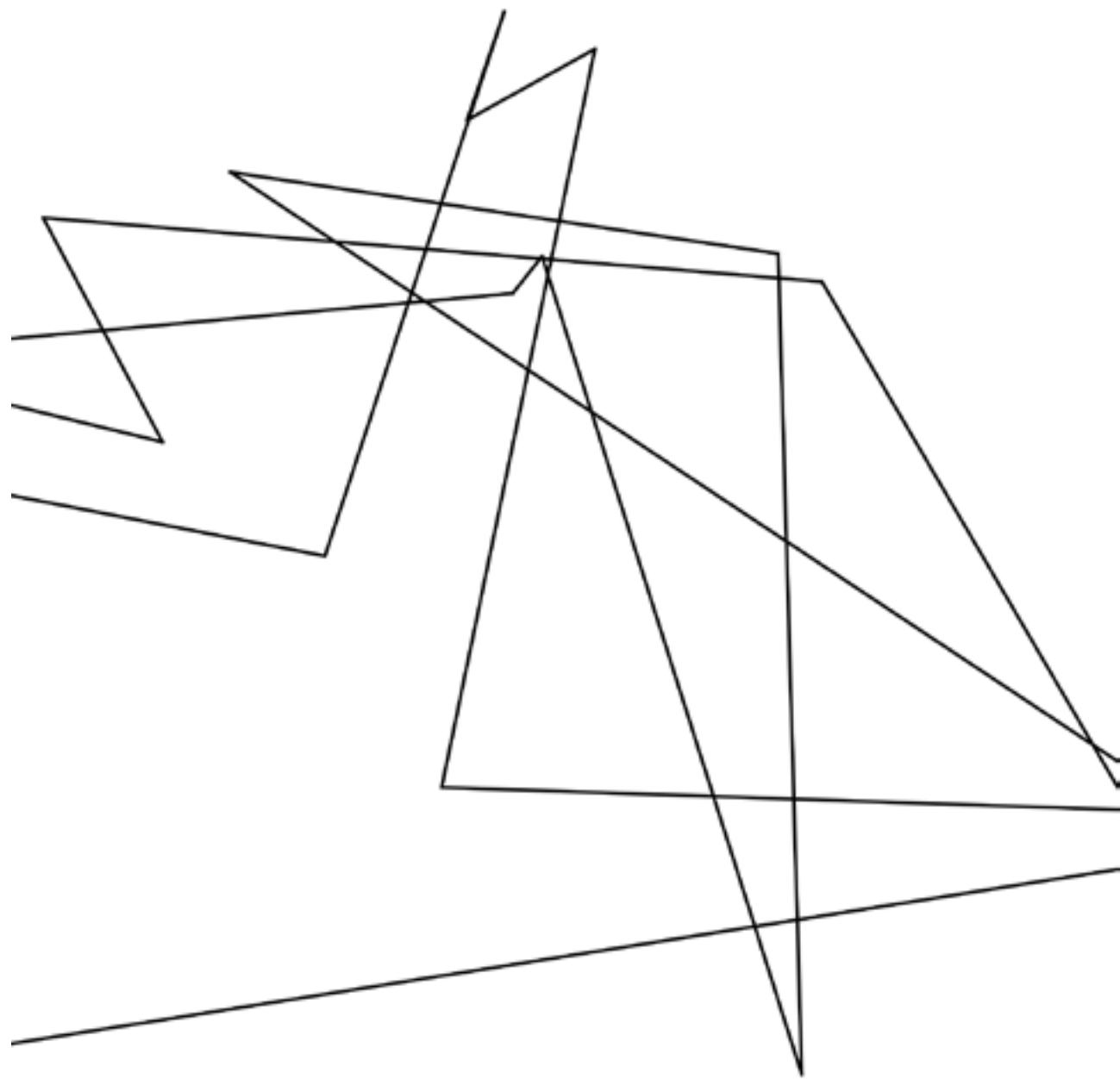


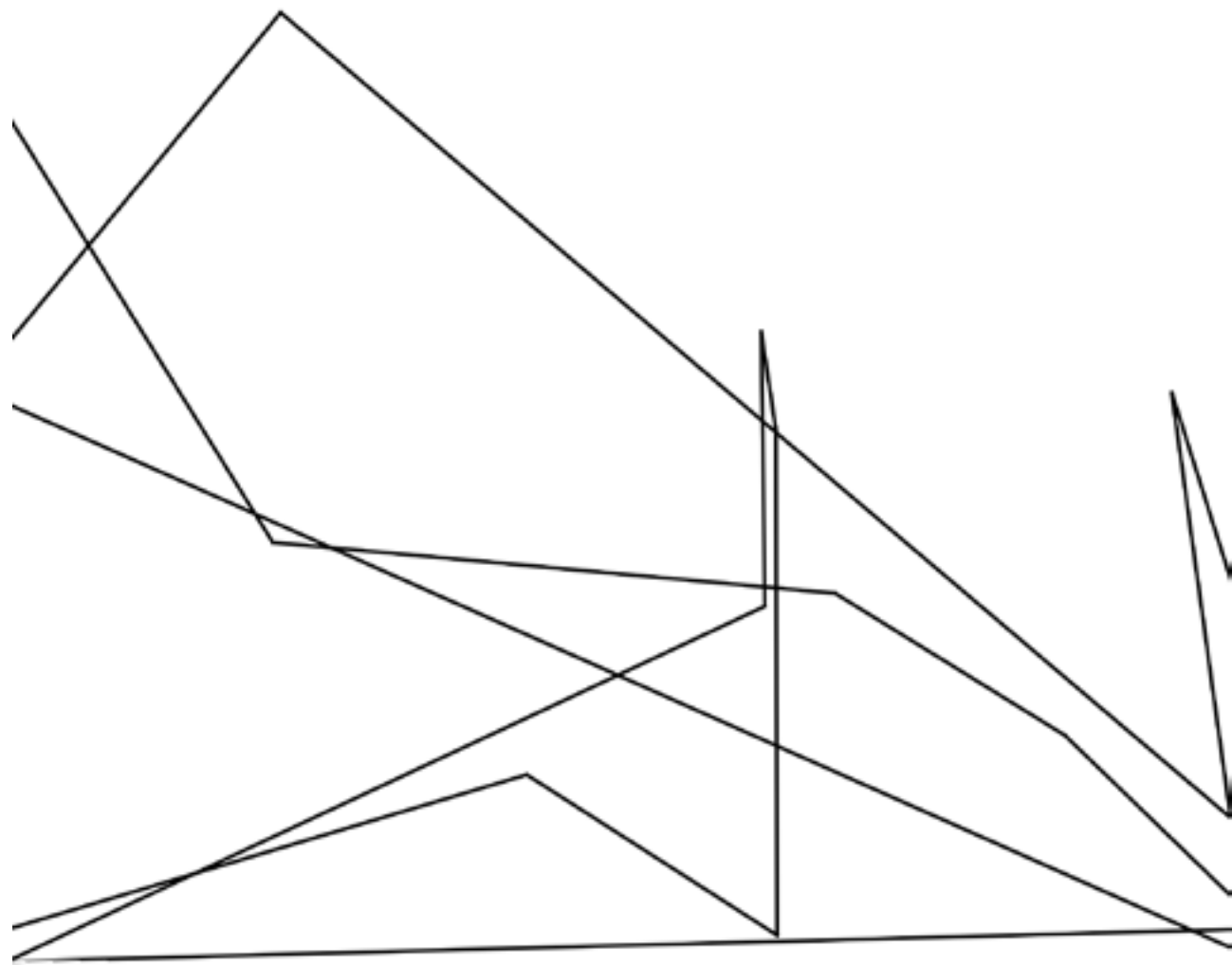
## The Eighteenth Line

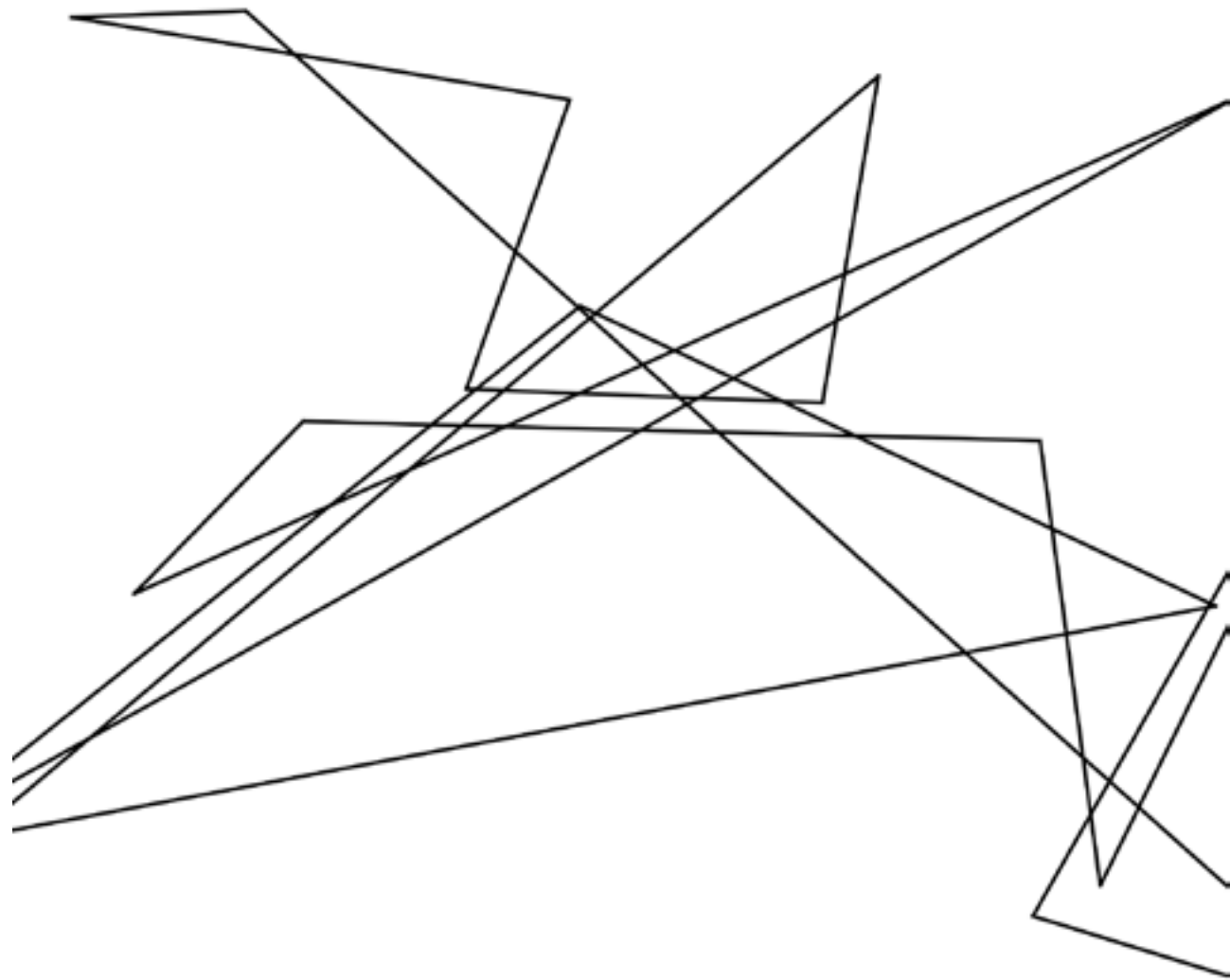
We also need to rotate the picture by ninety degrees like Frieder did.

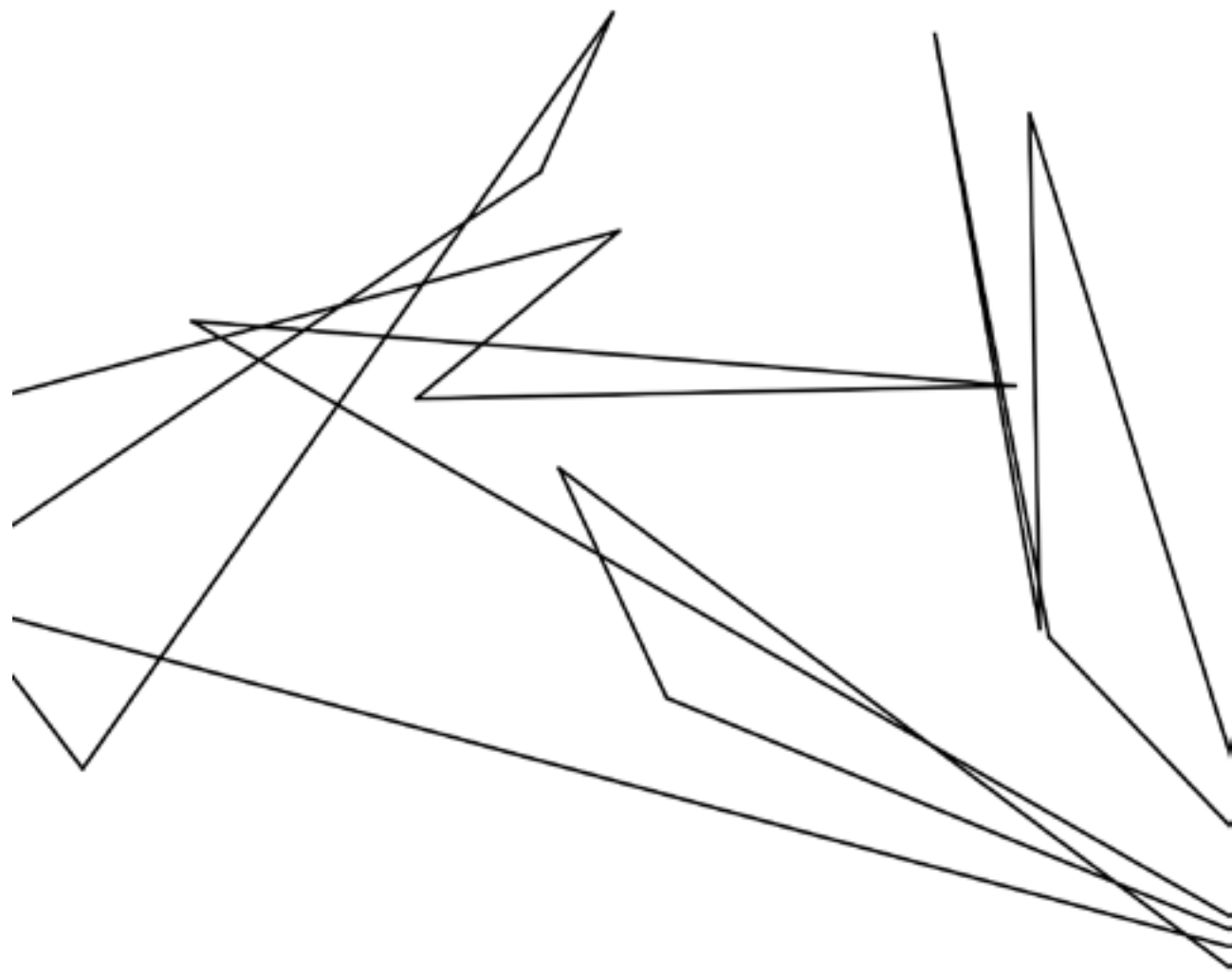
```
translate(width/2, height/2);  
rotate(radians(90));
```

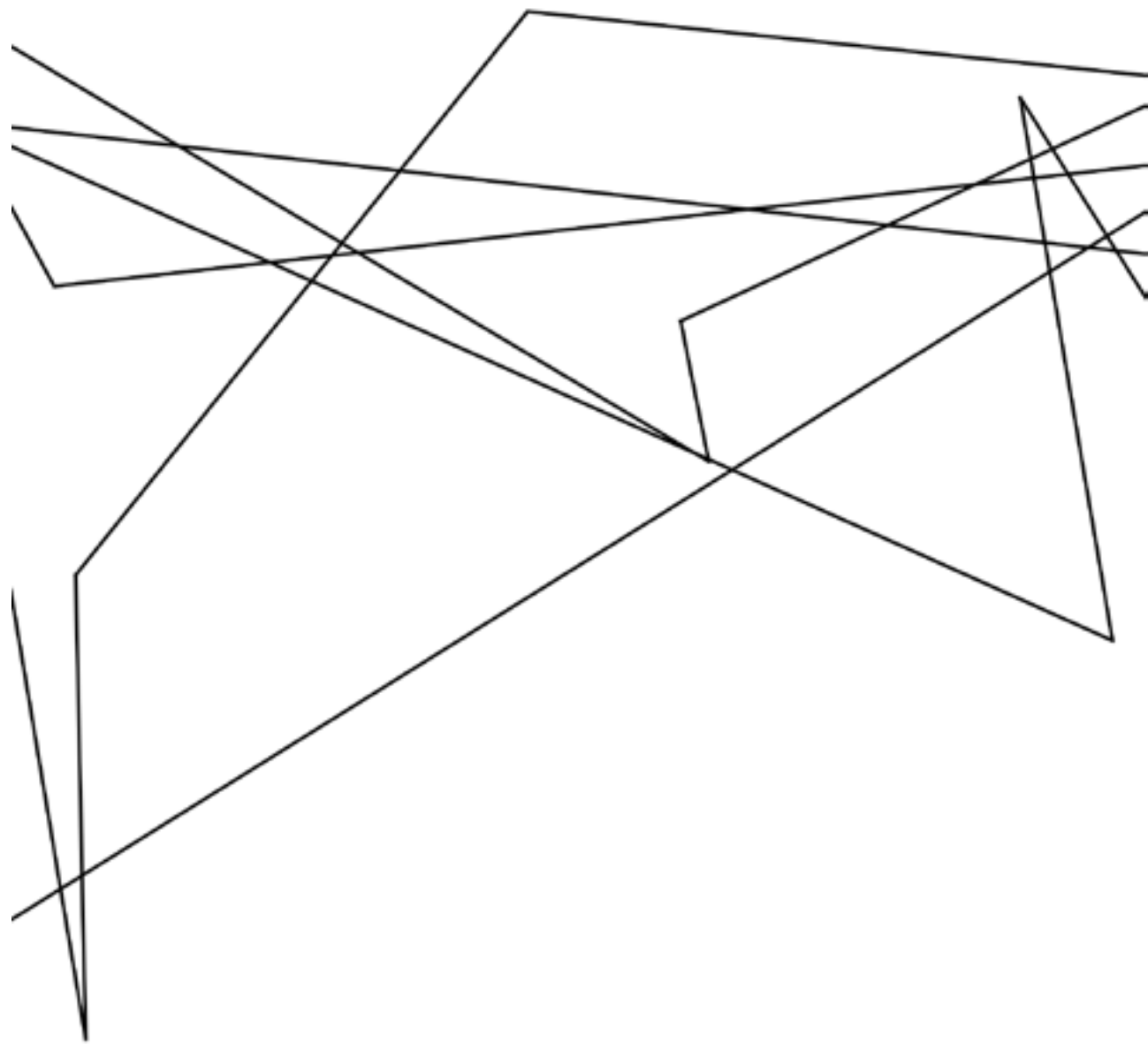


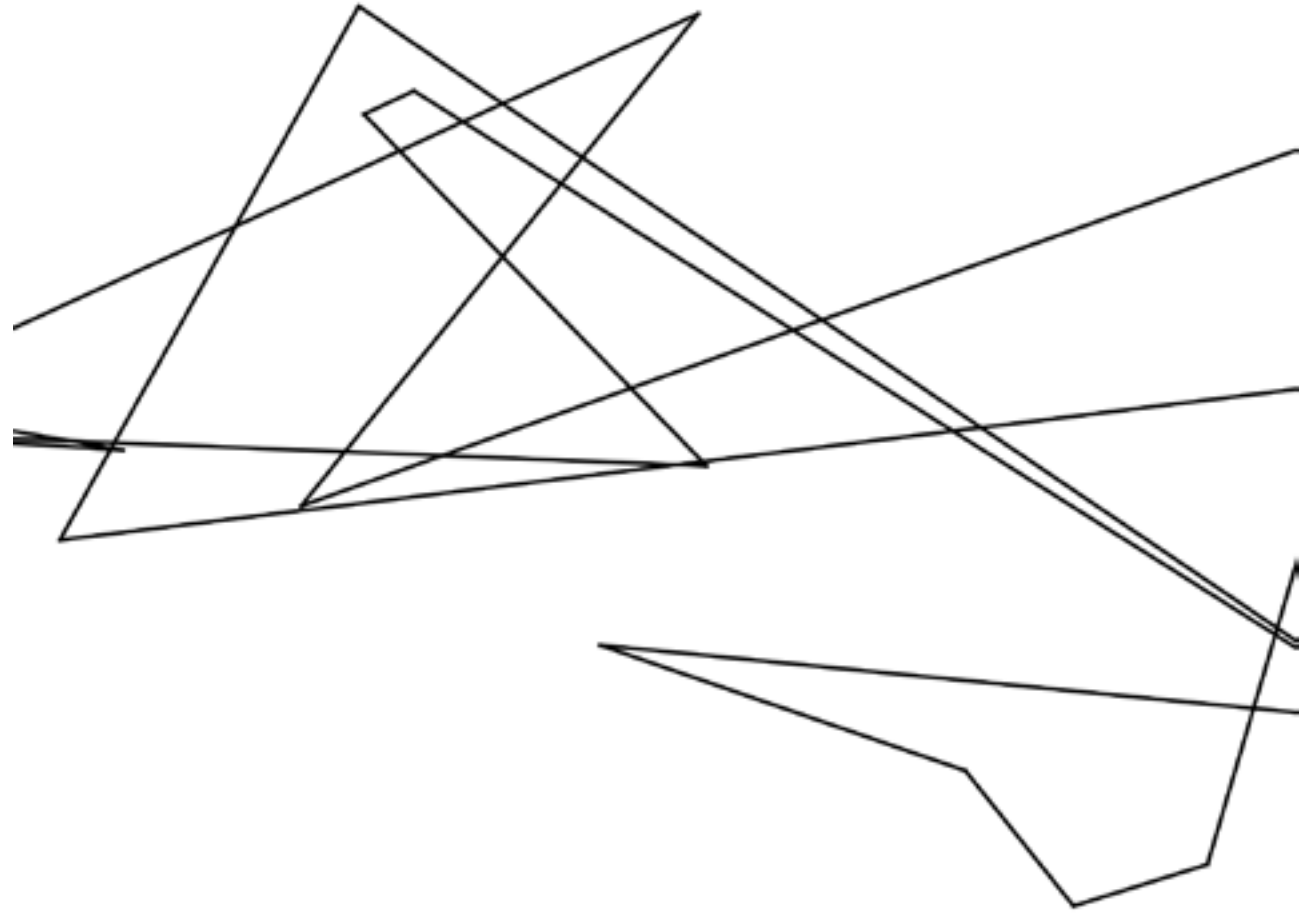


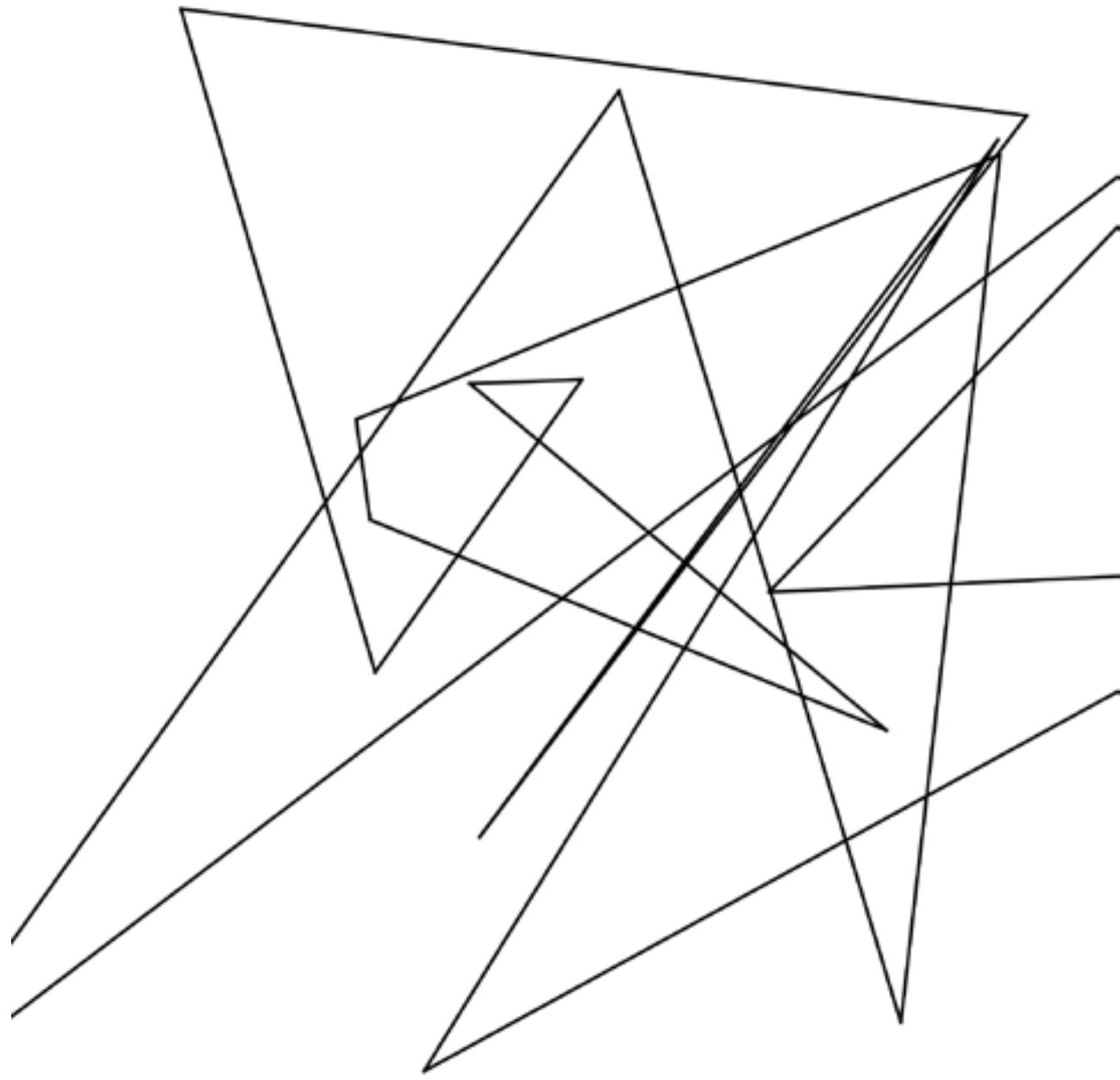


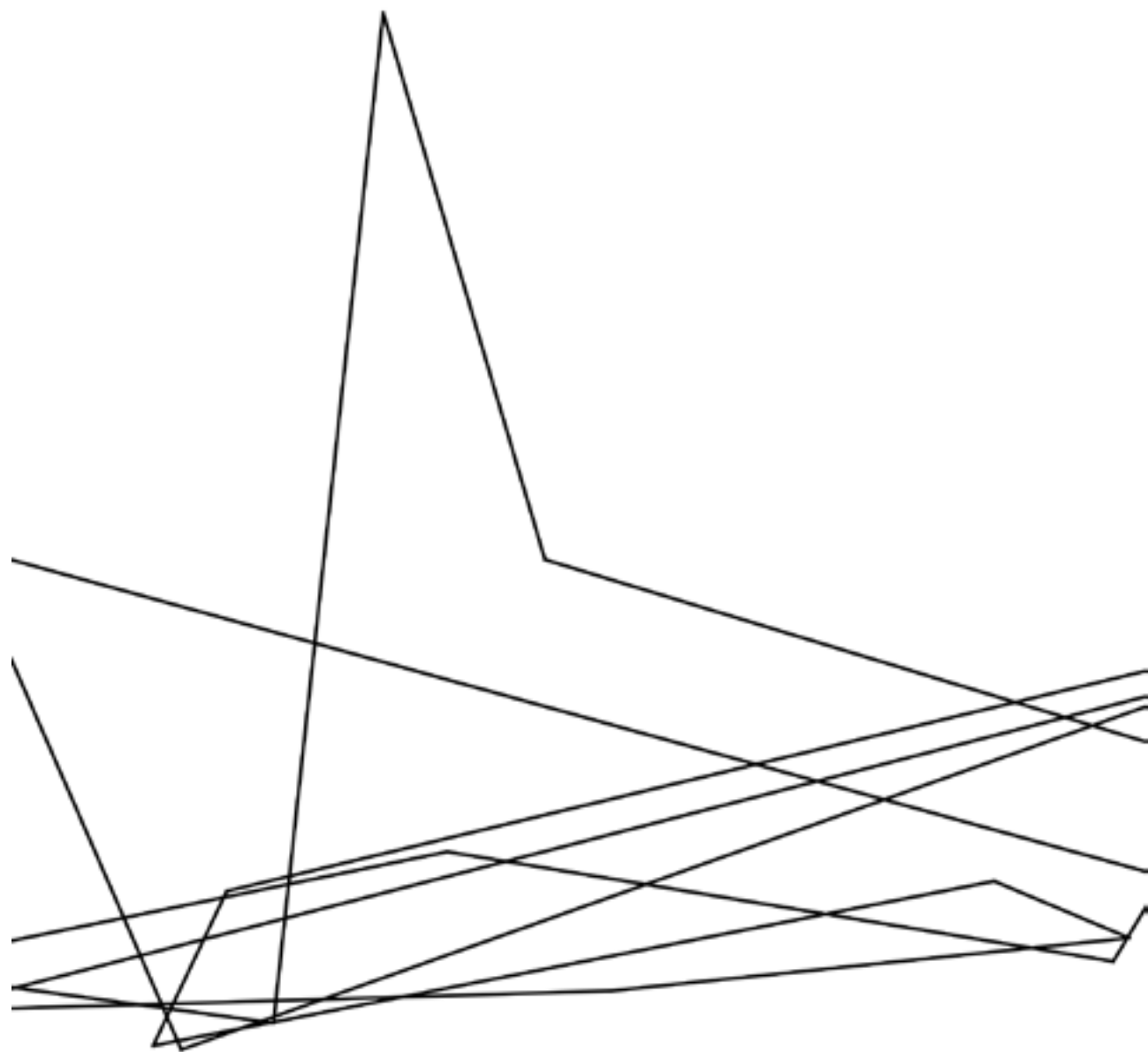






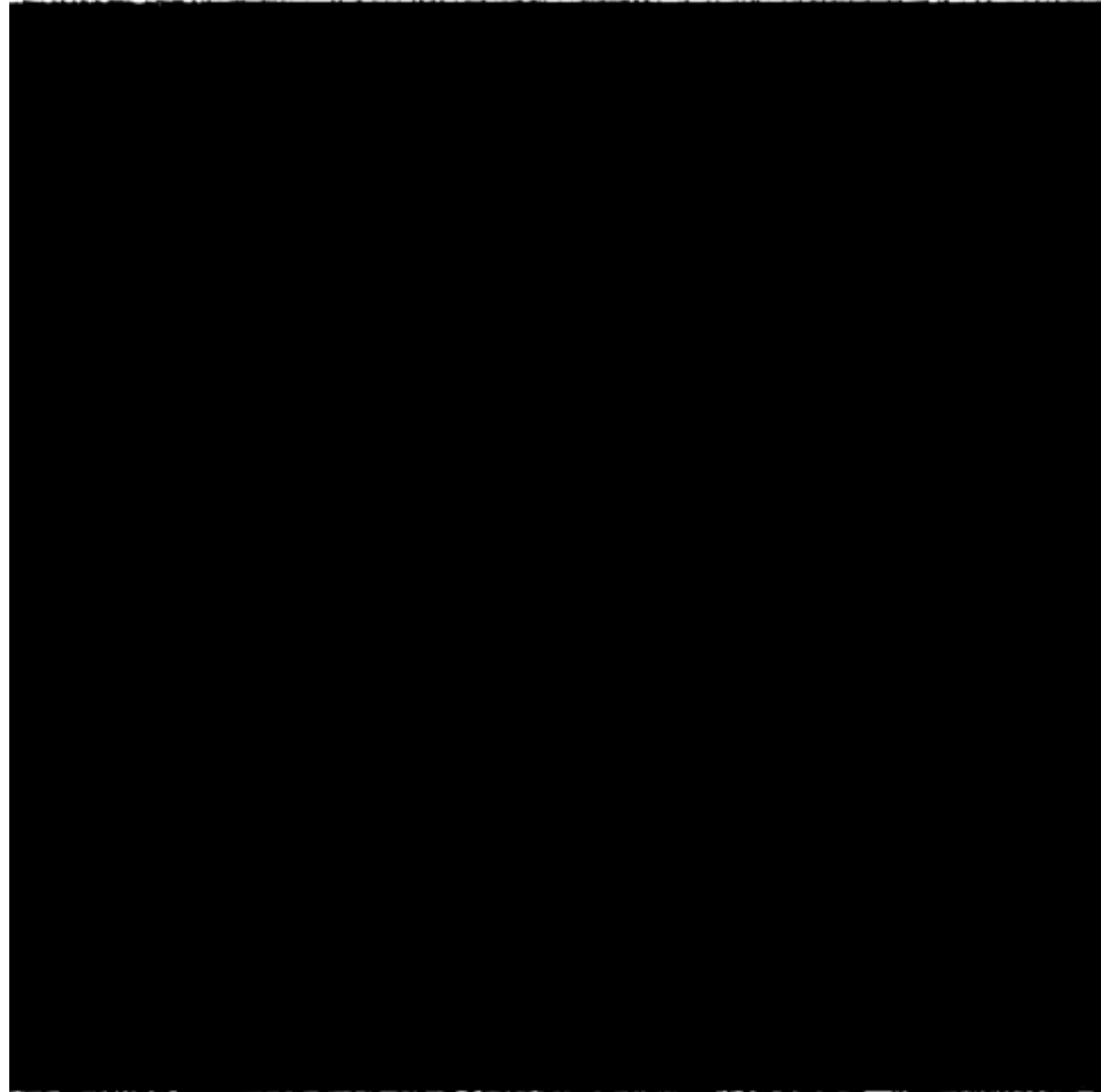


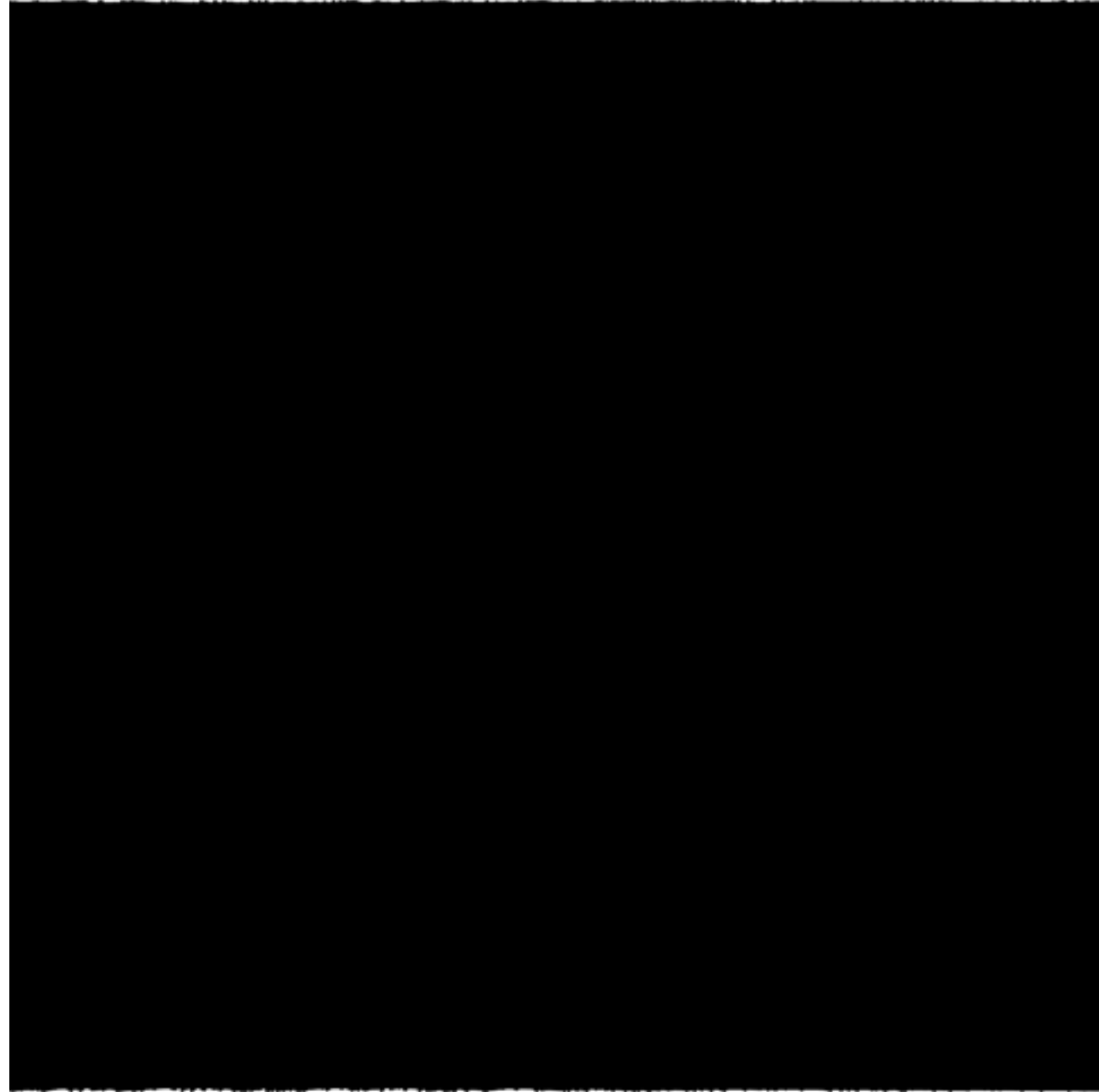


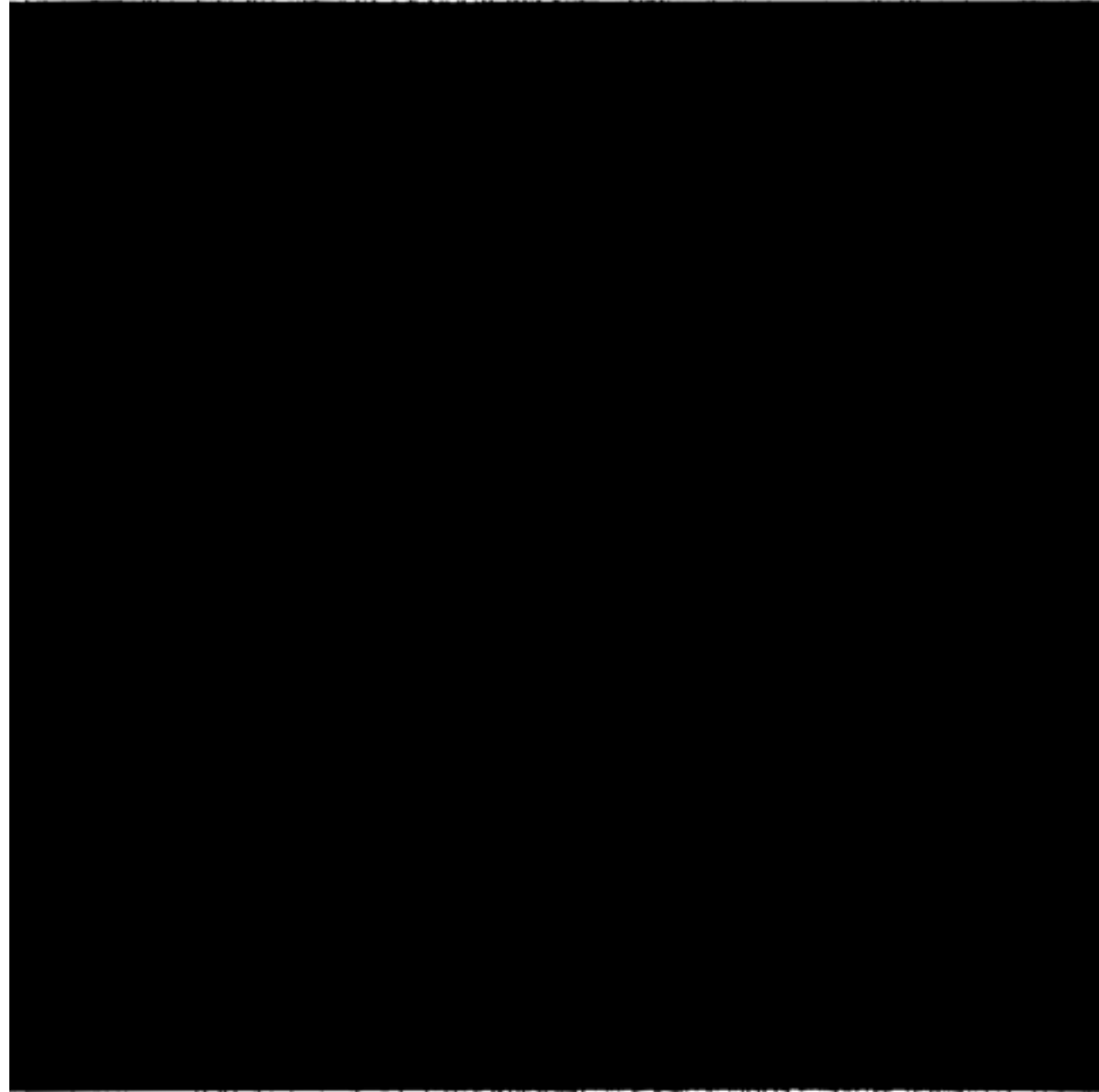


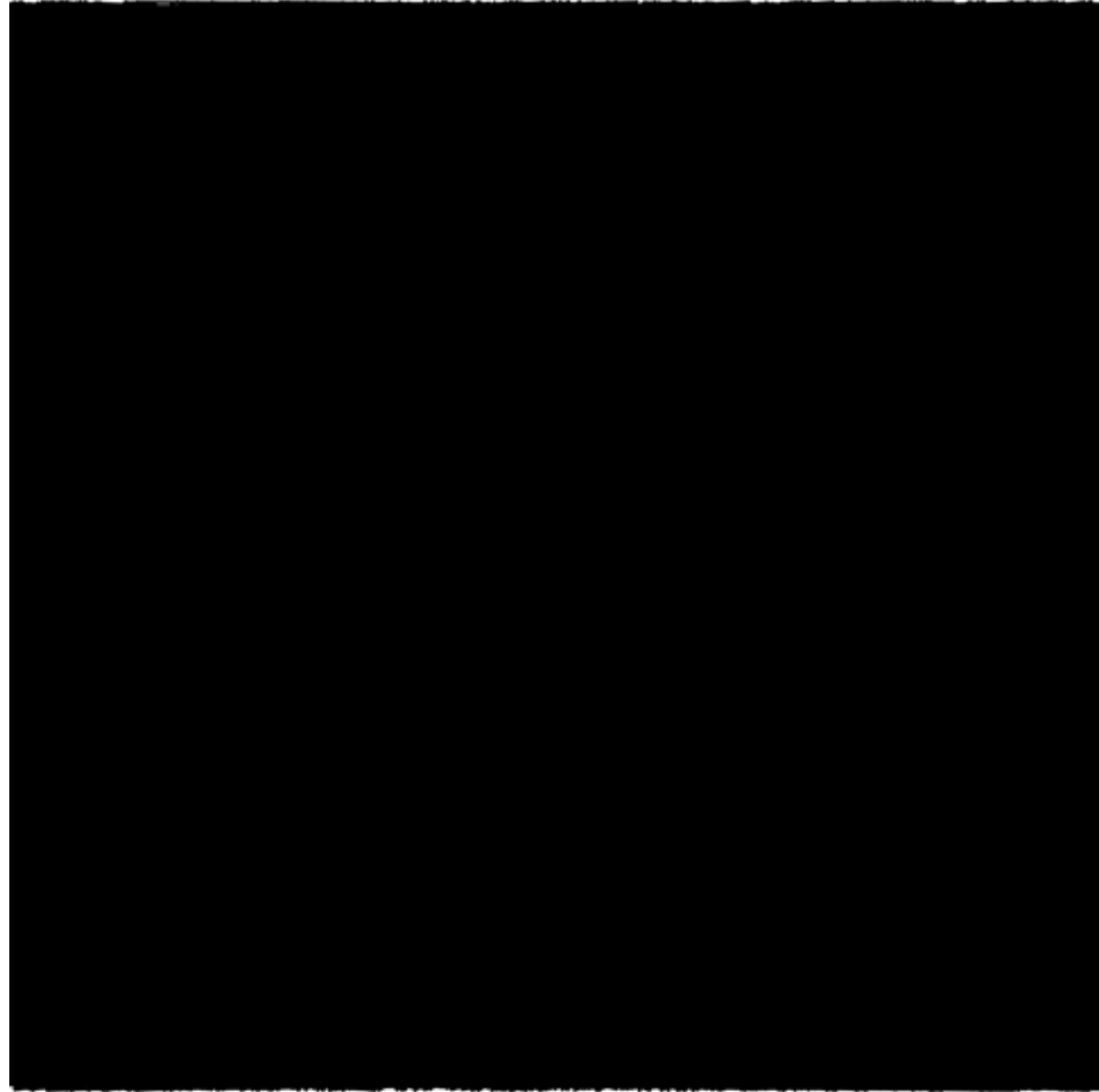
## The 161238th Line

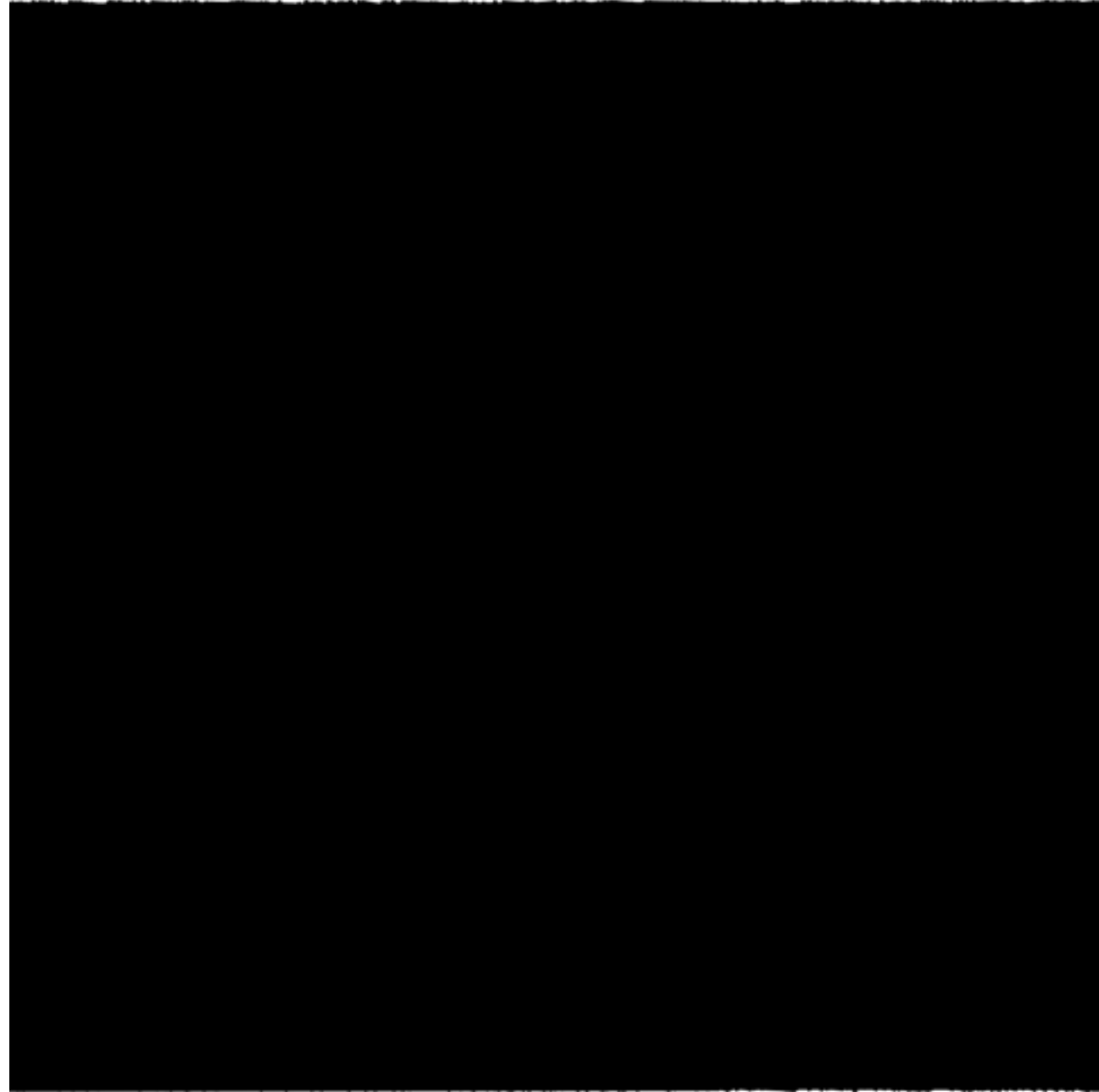
Oh, a black square! Maybe we need to stick with less lines.

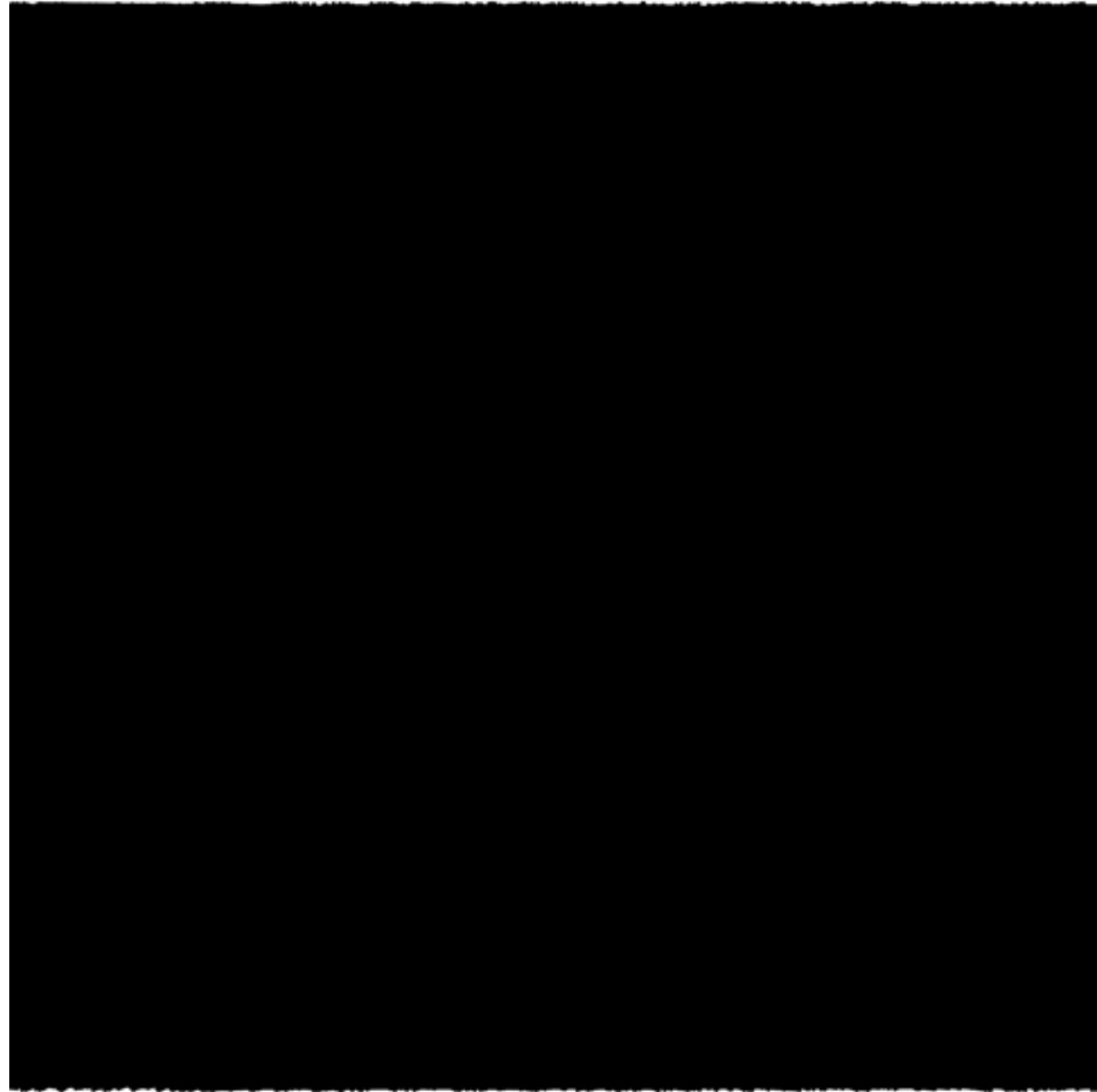


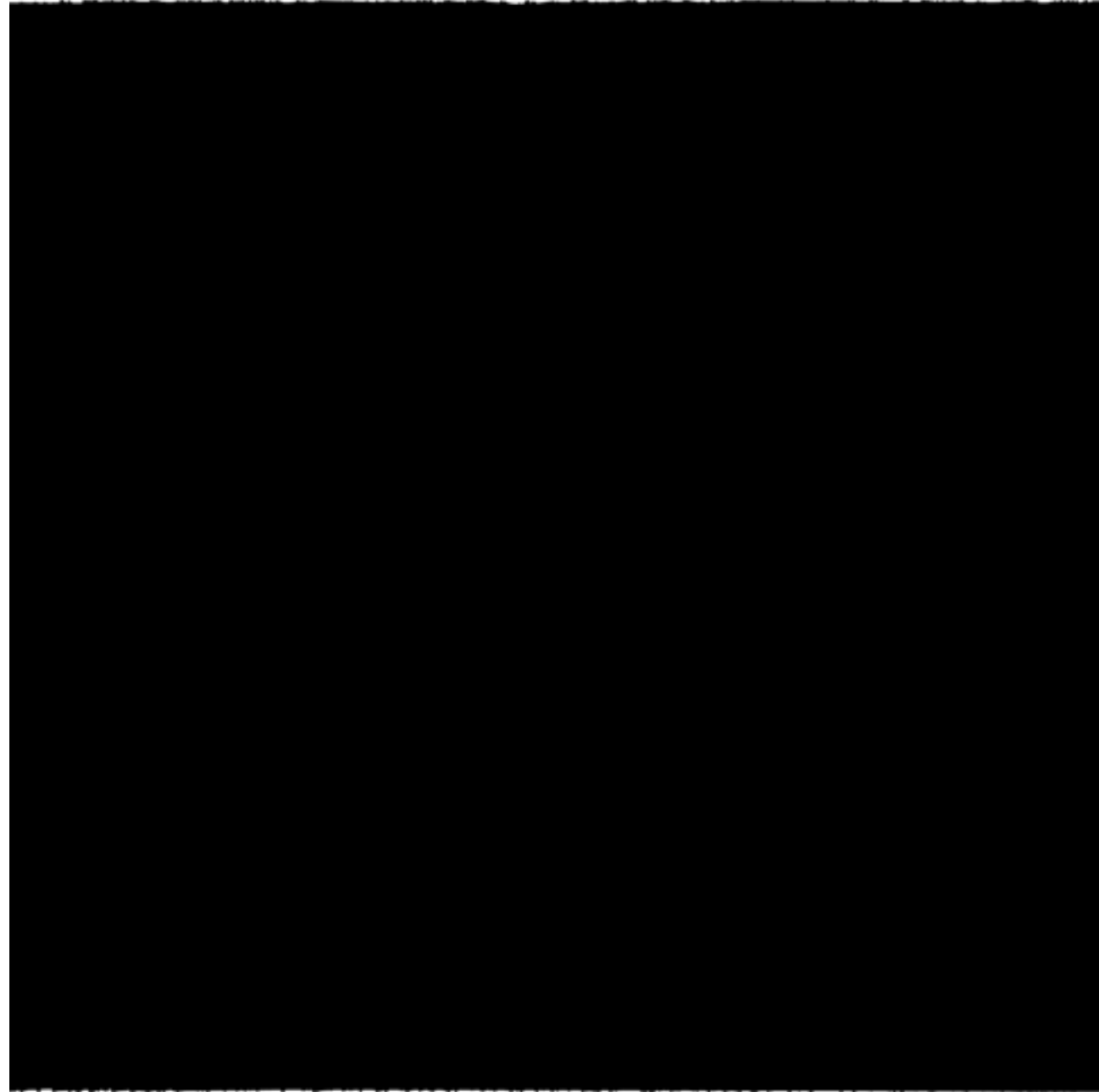




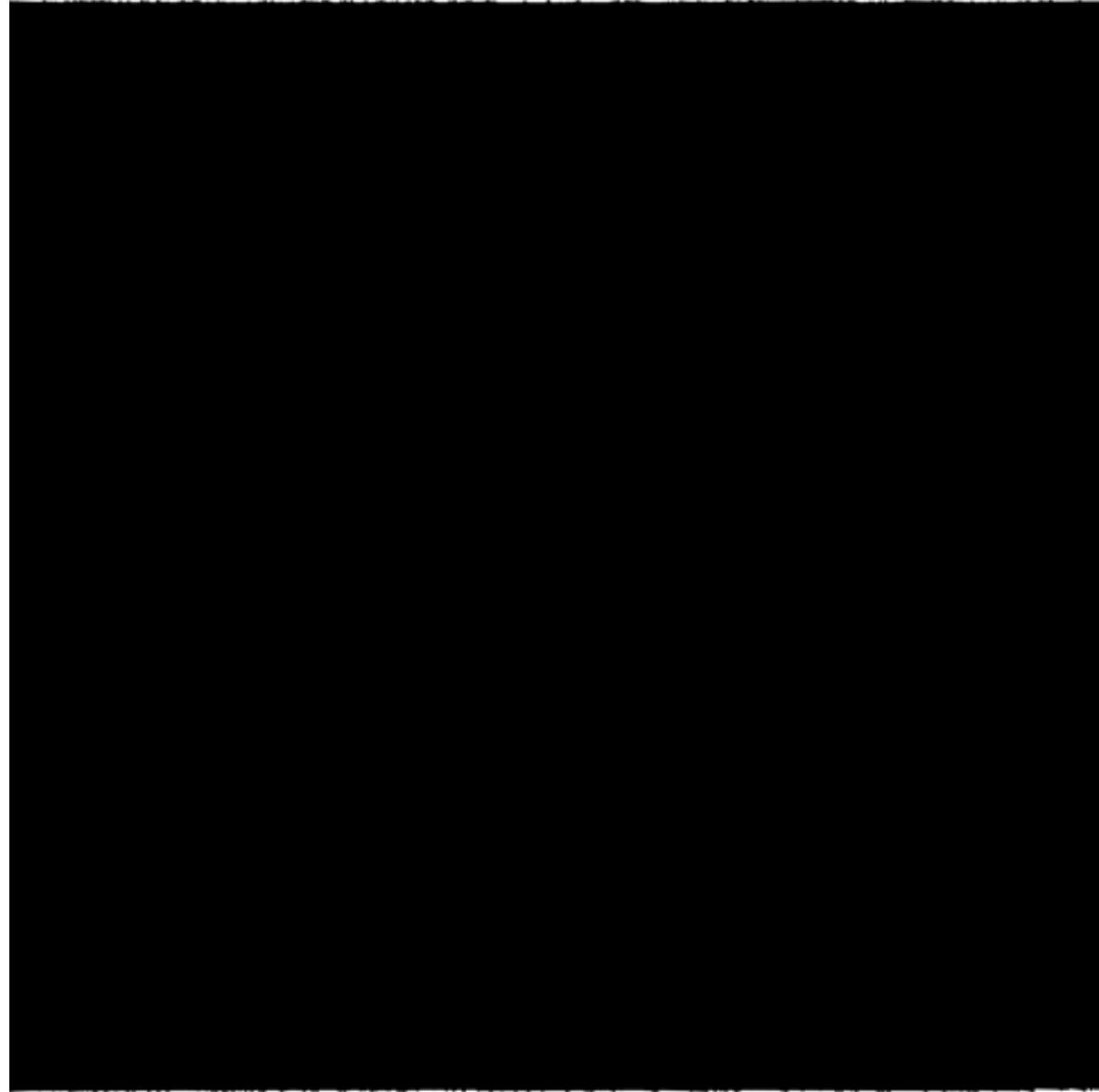


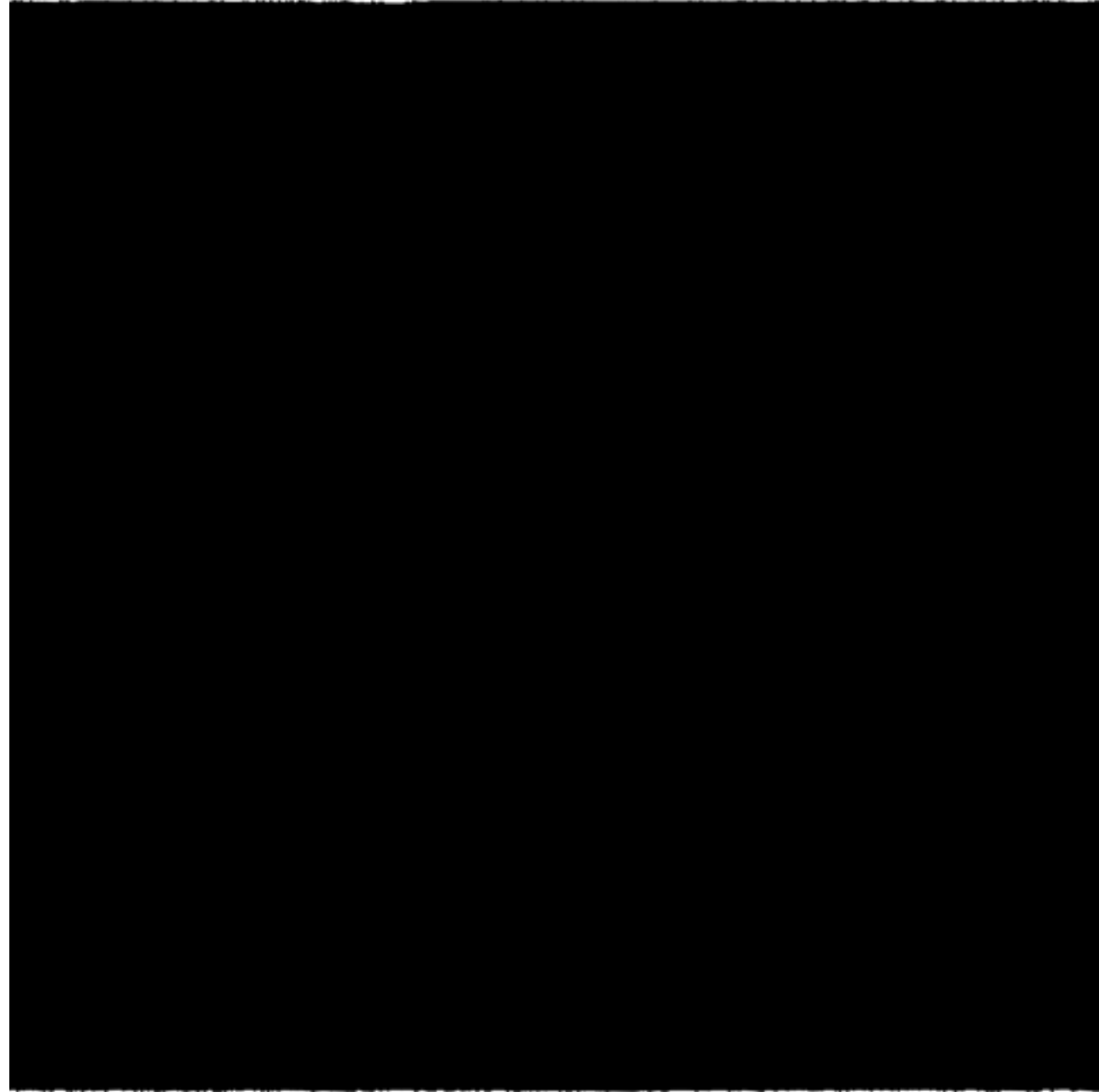


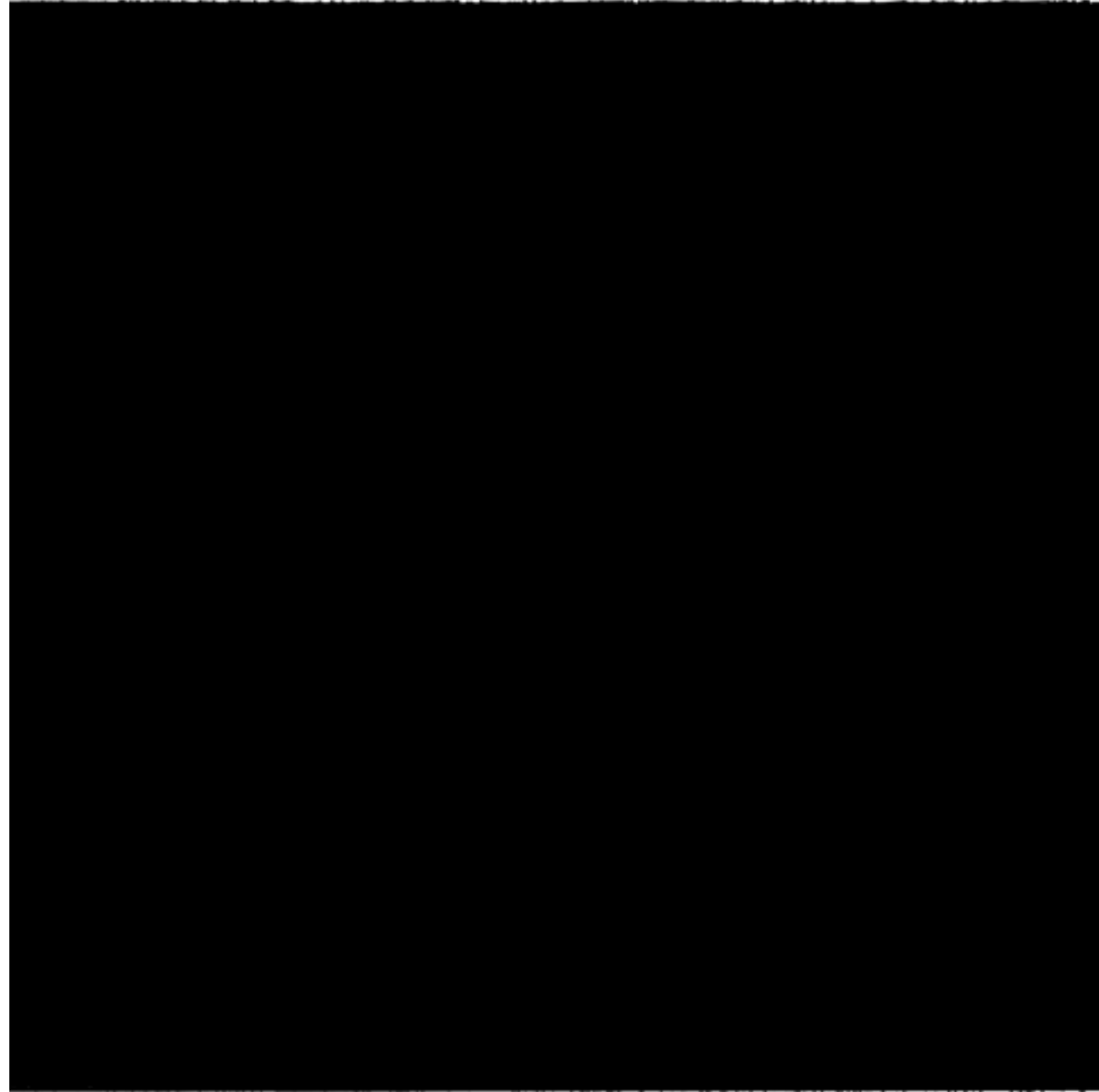


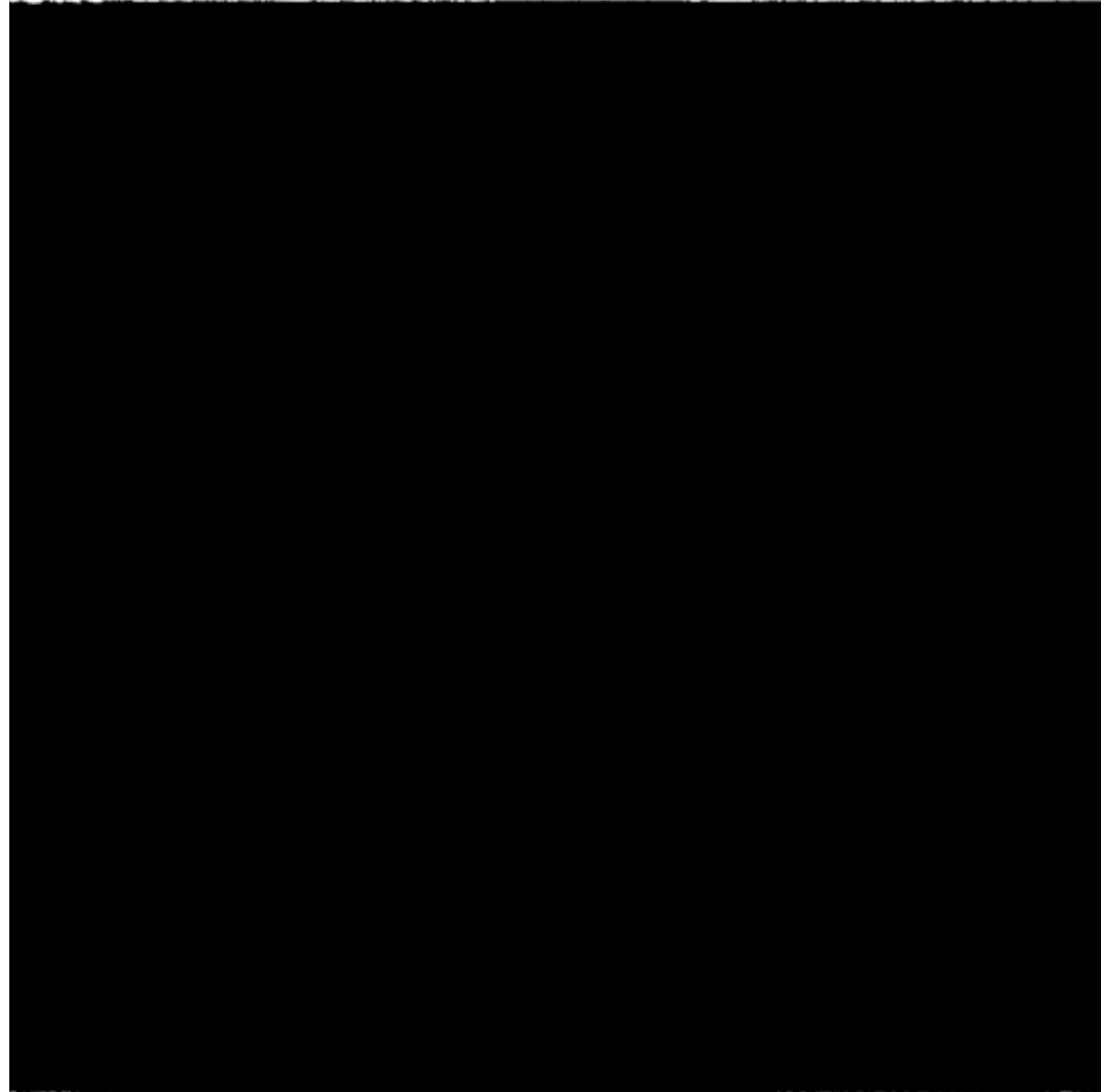


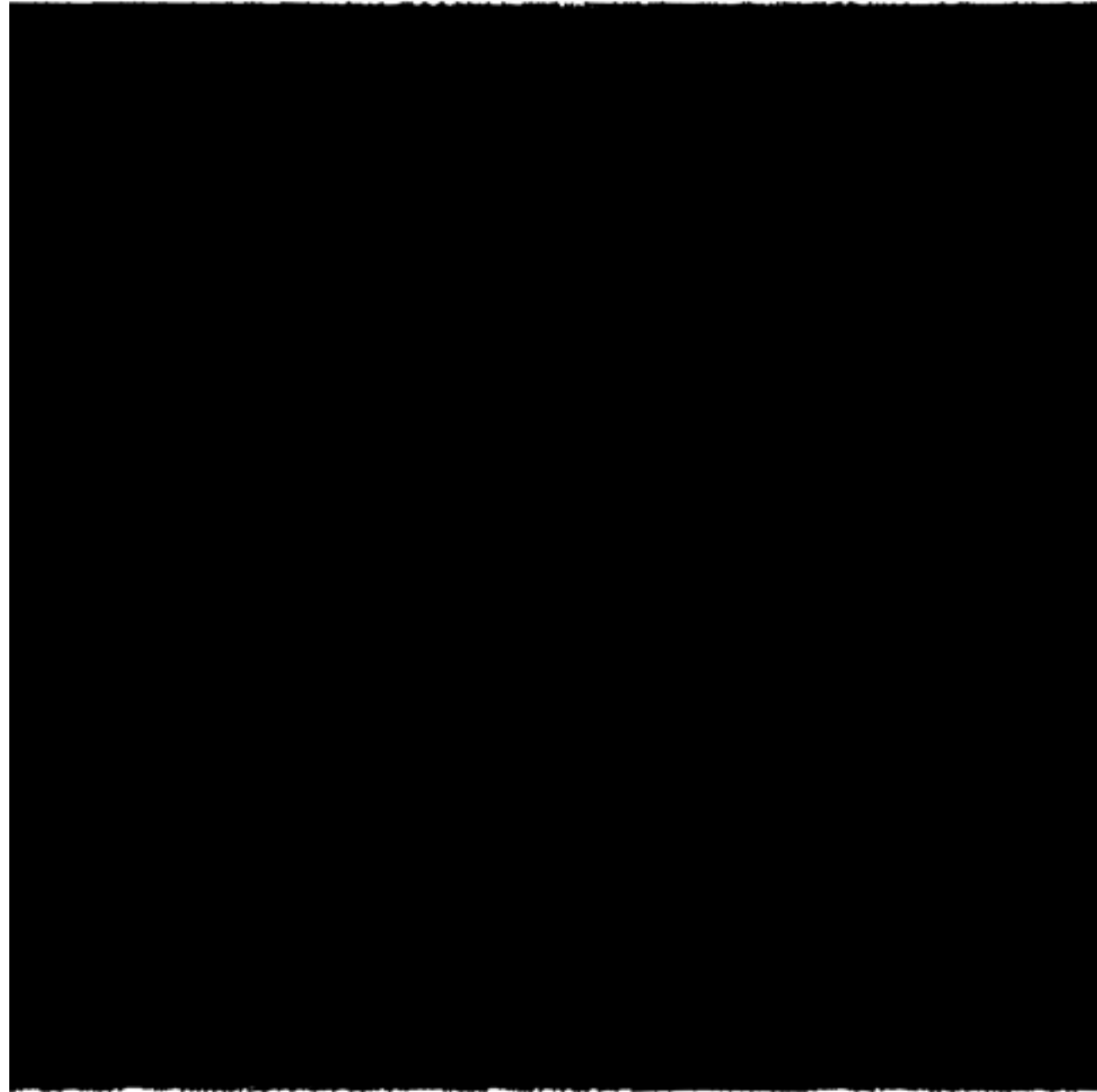


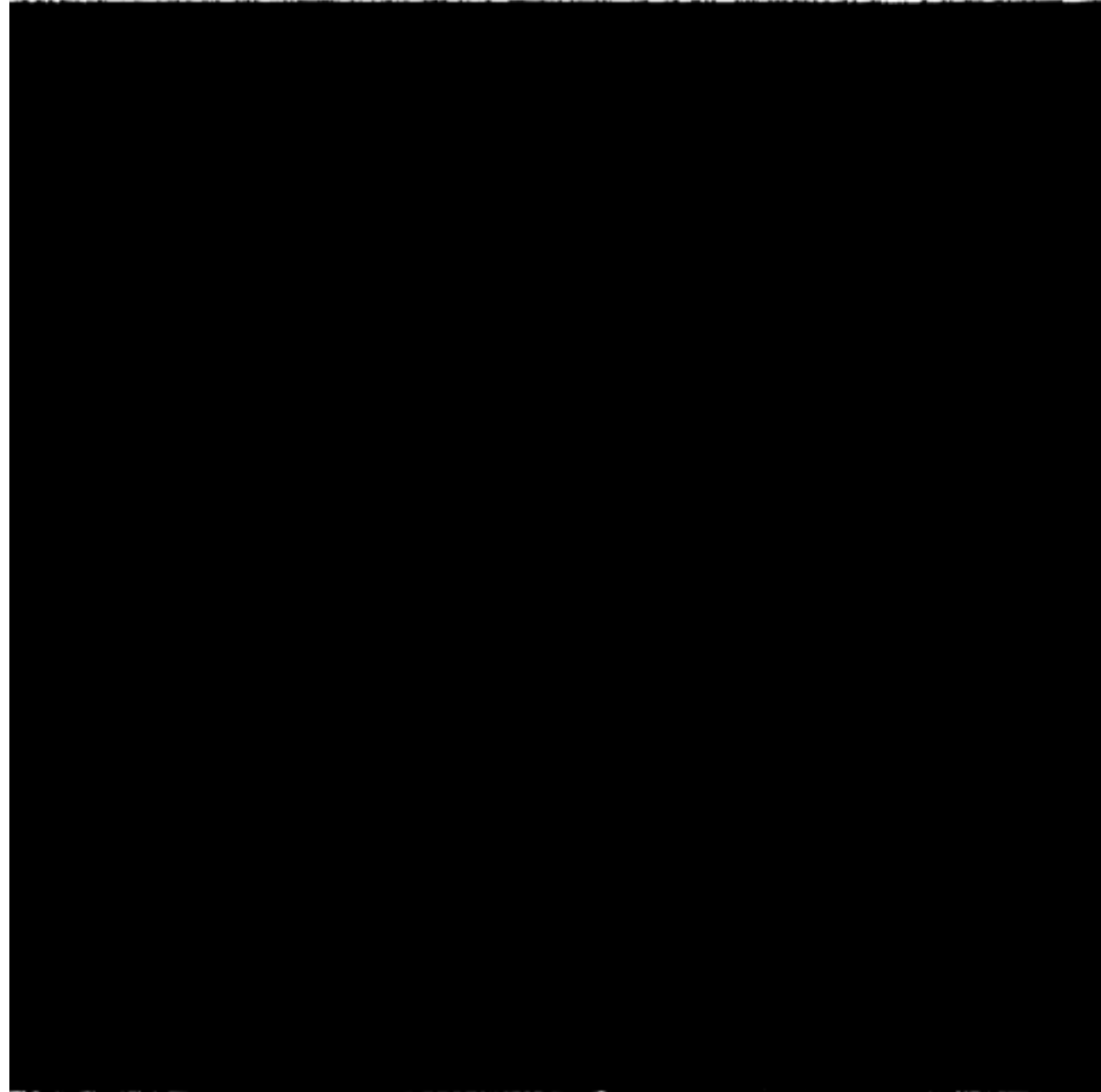


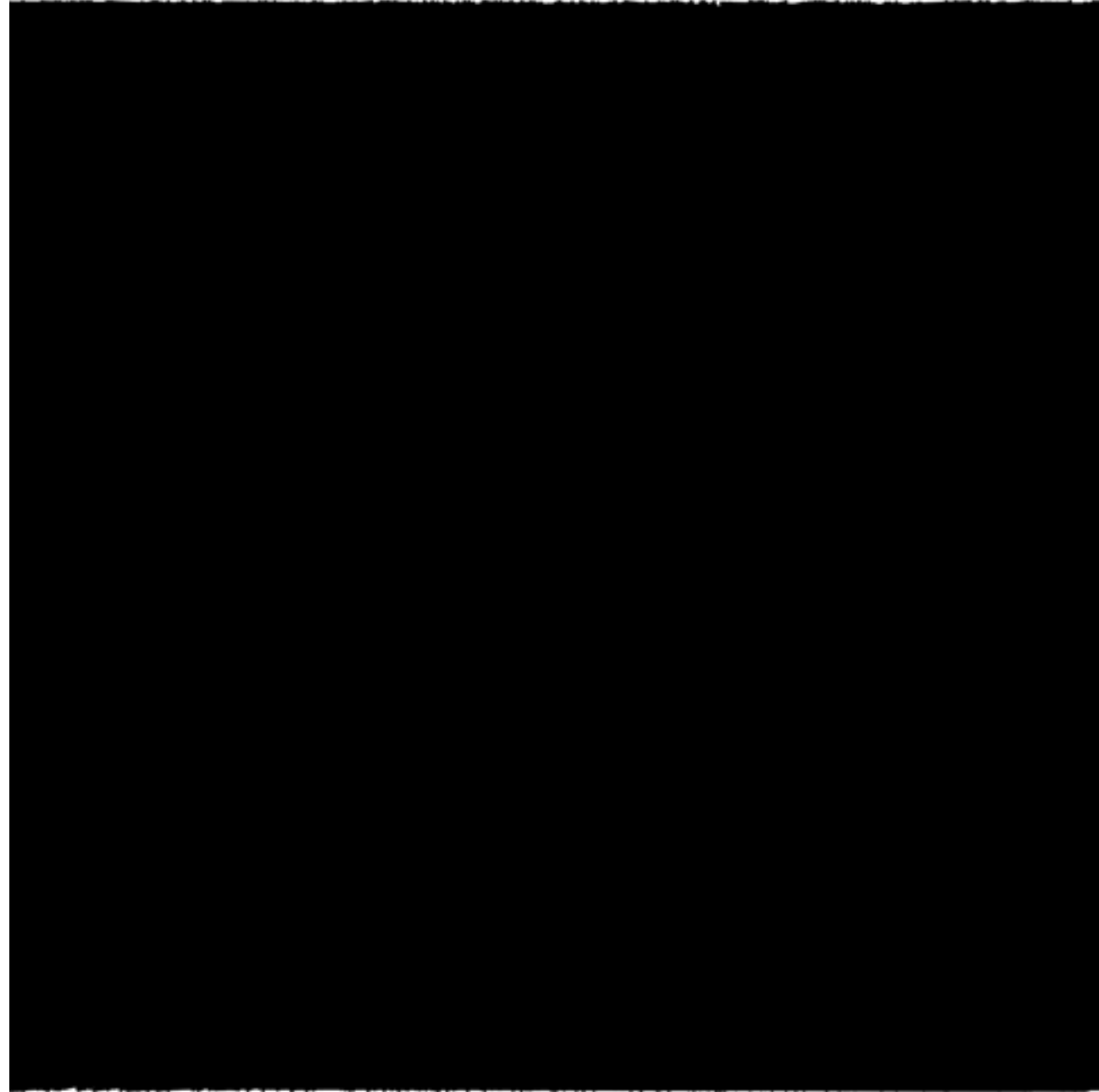


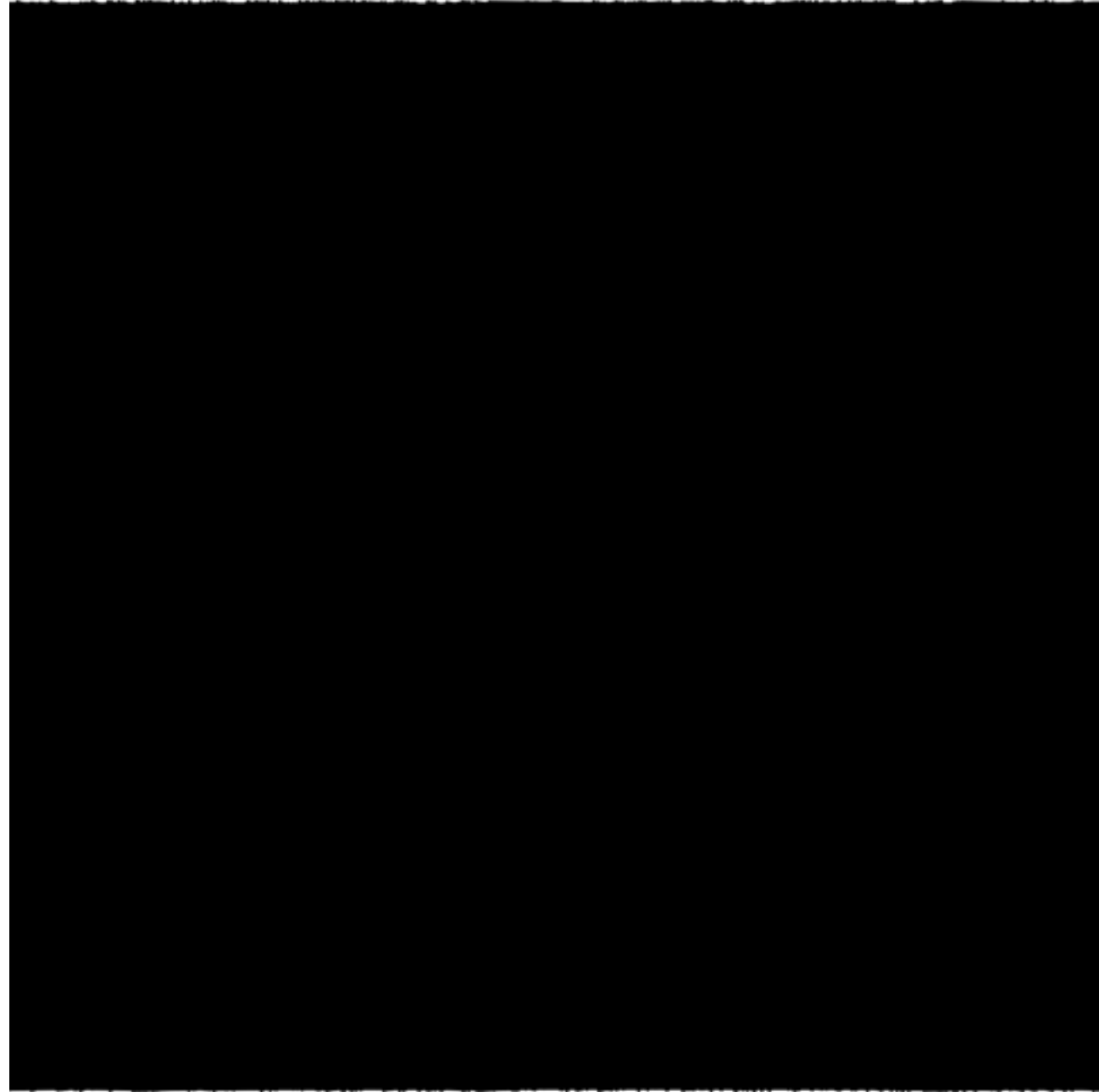


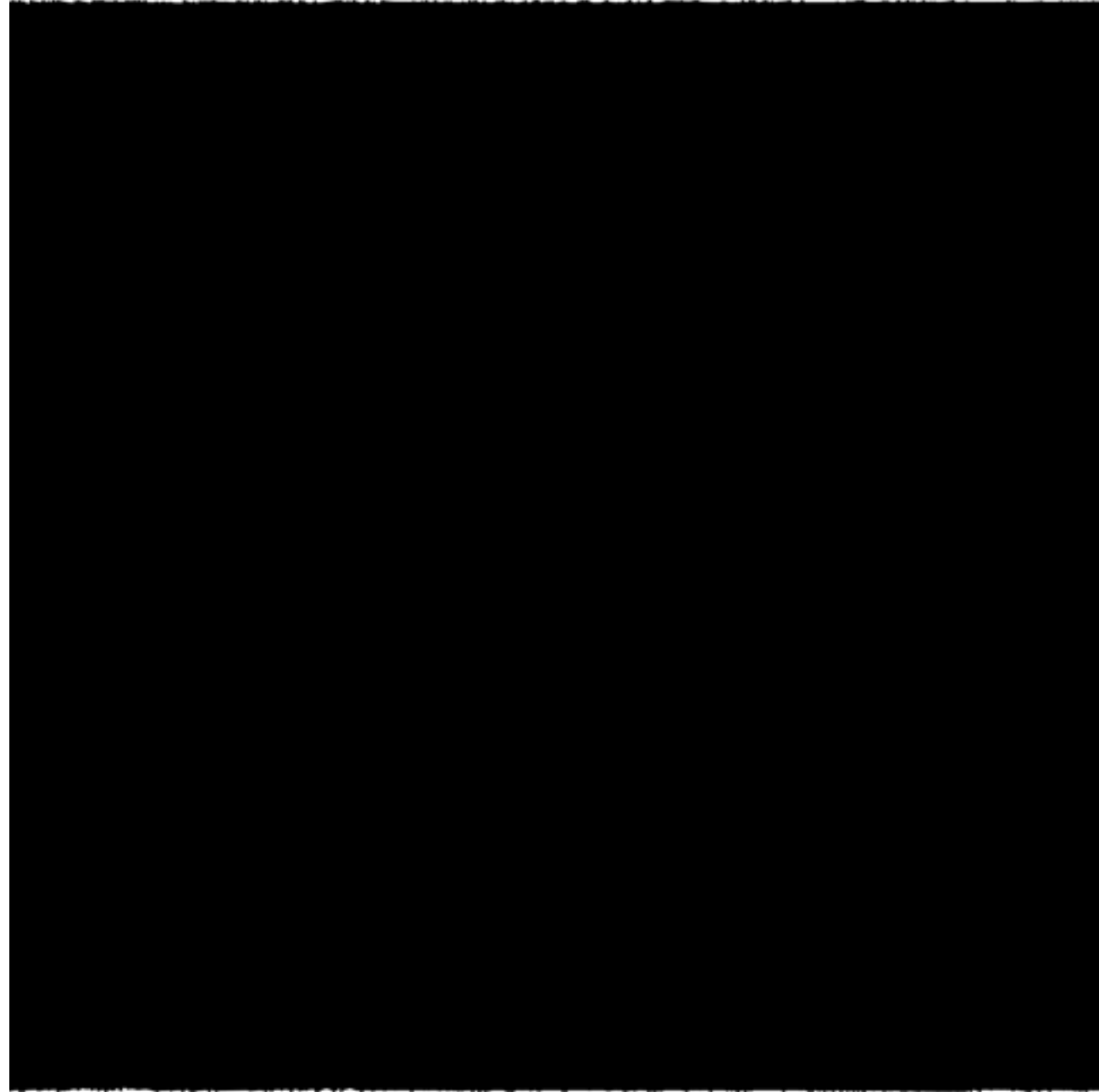


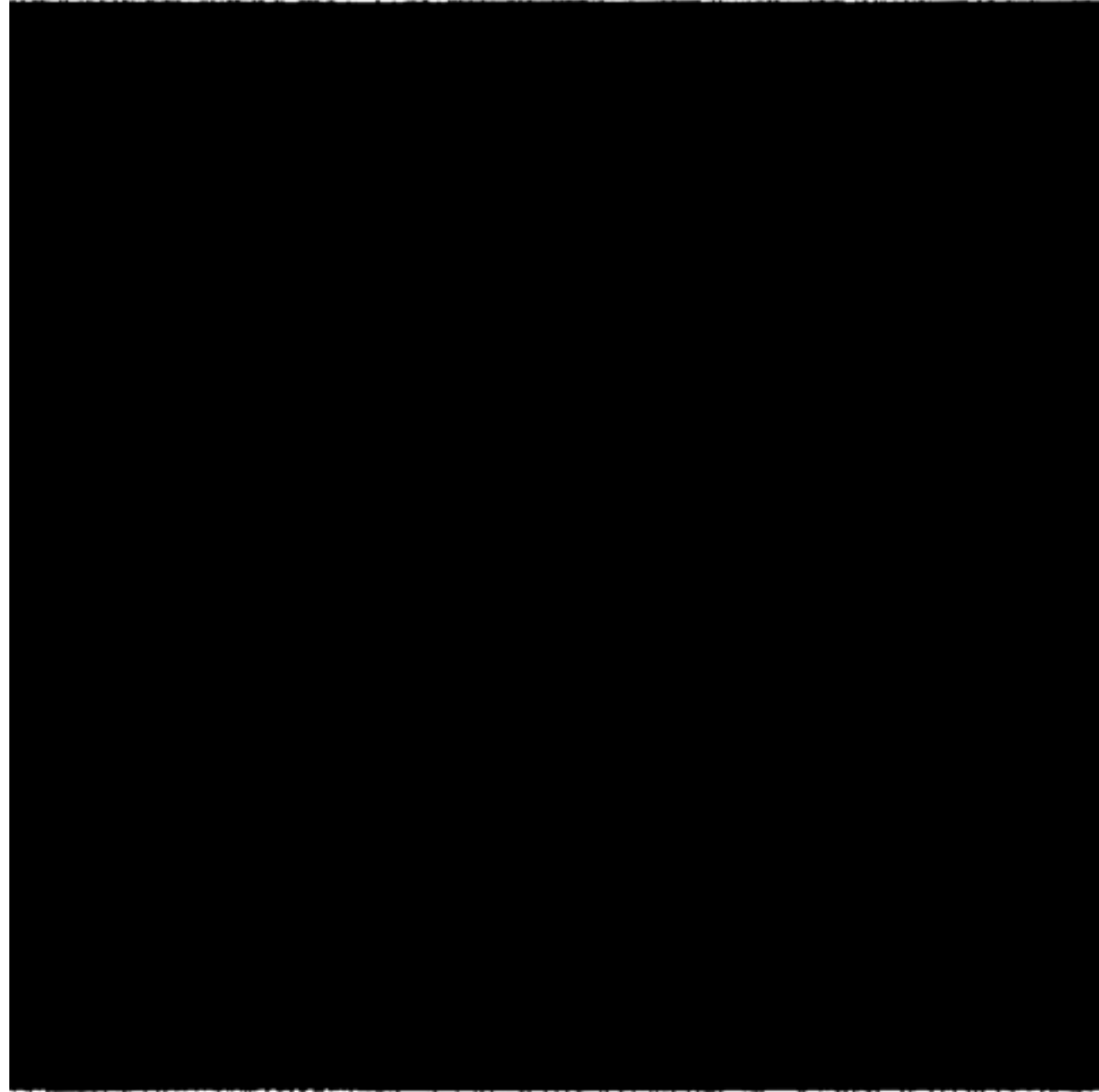


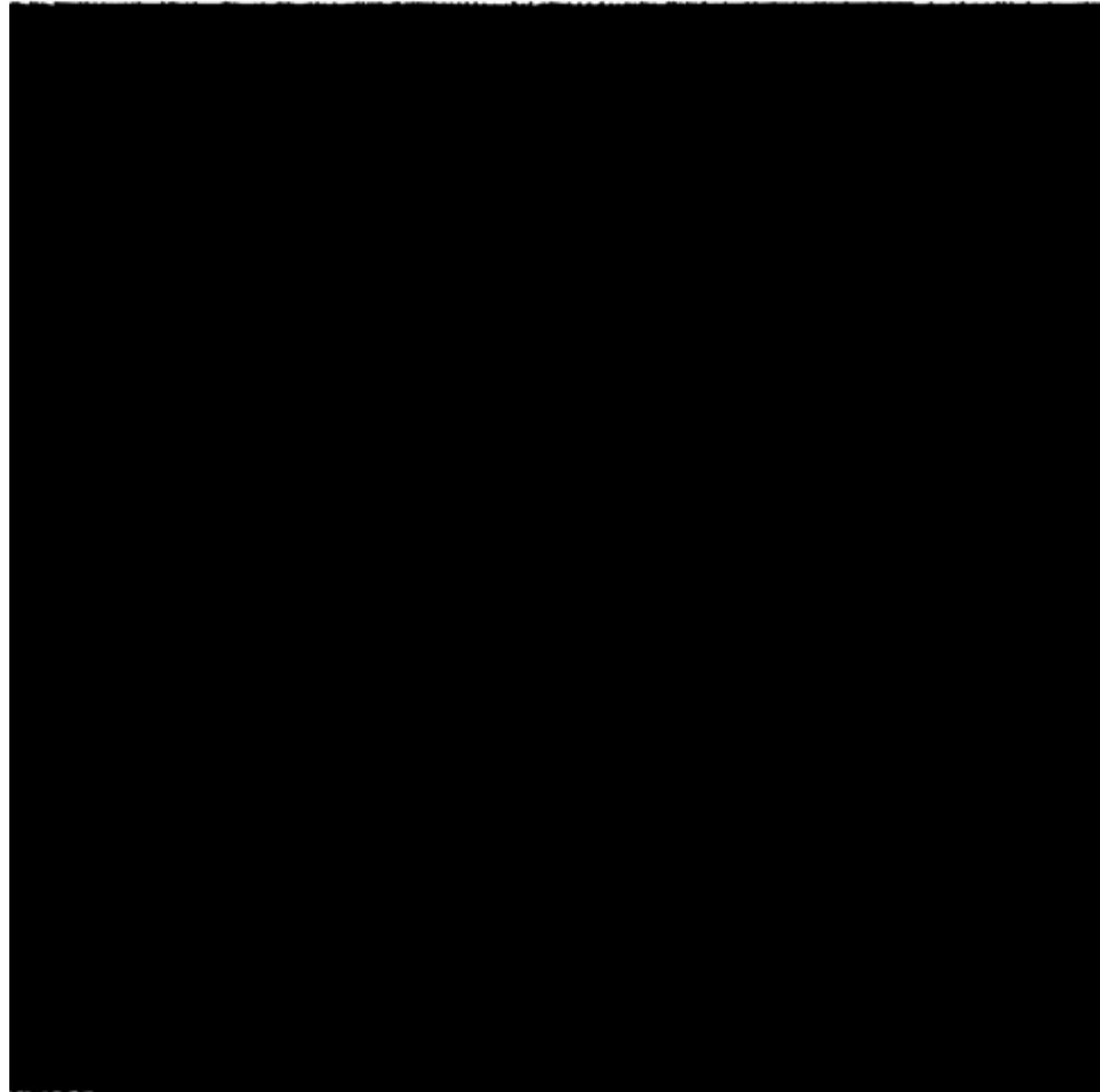


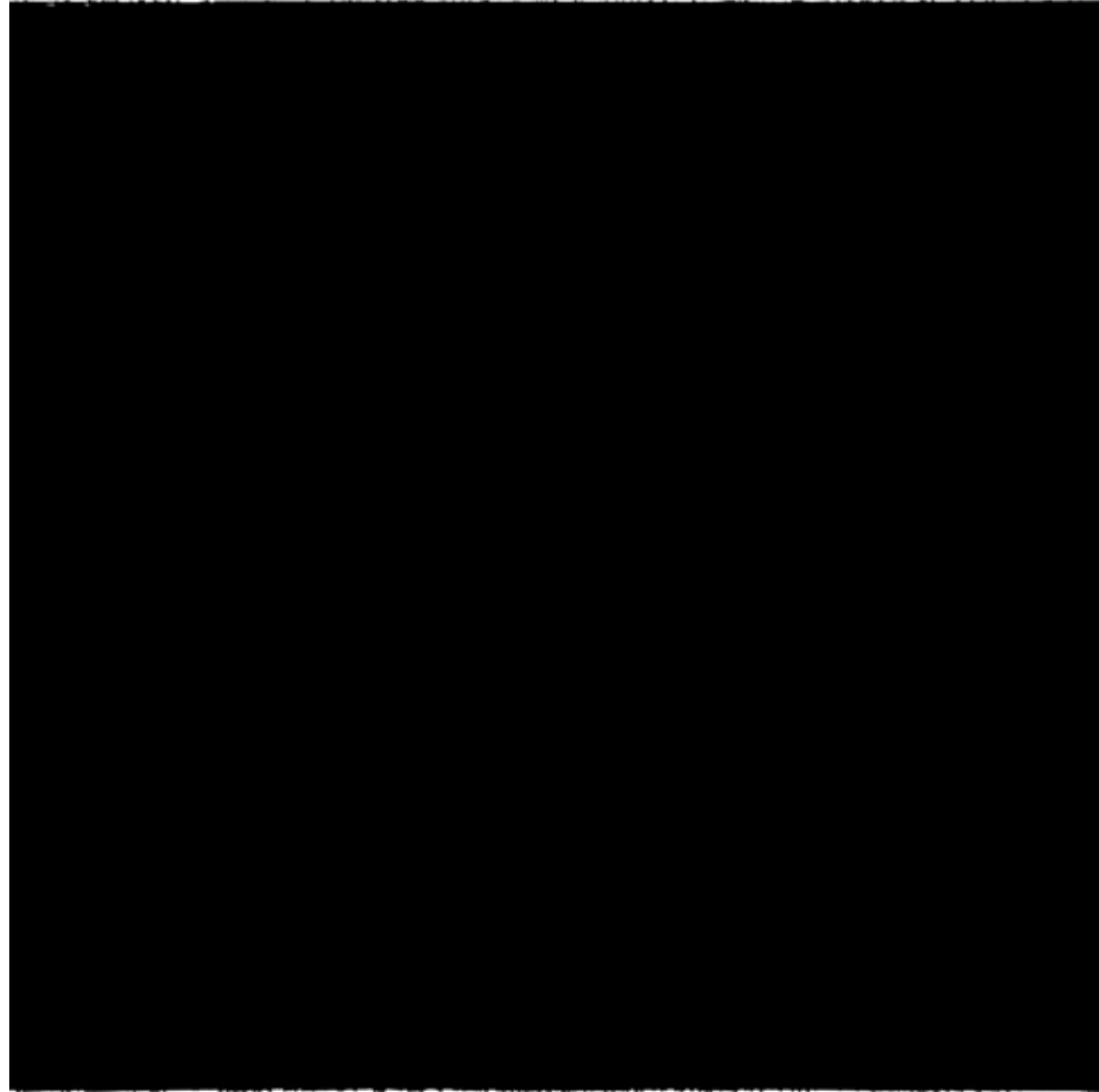


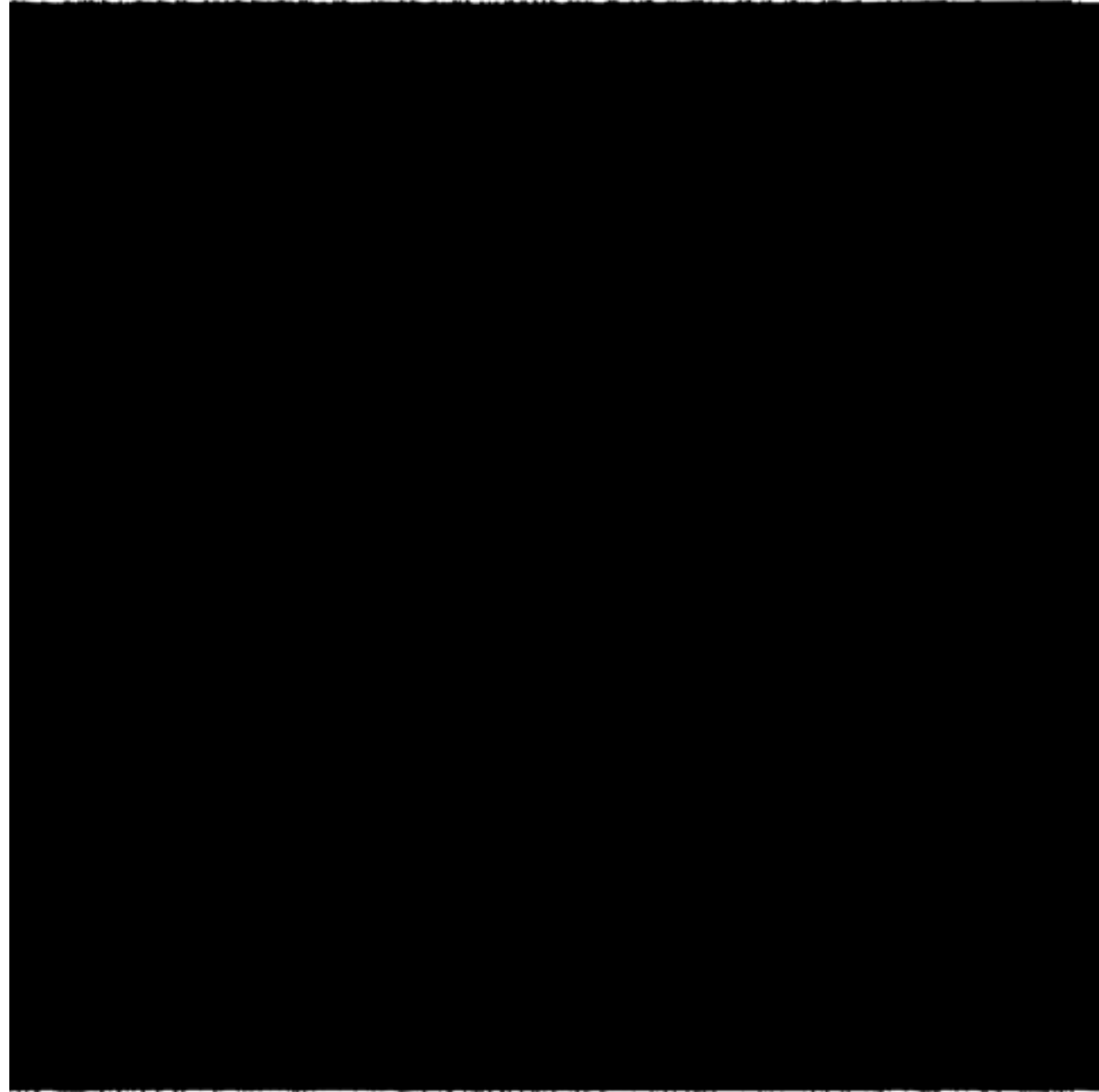










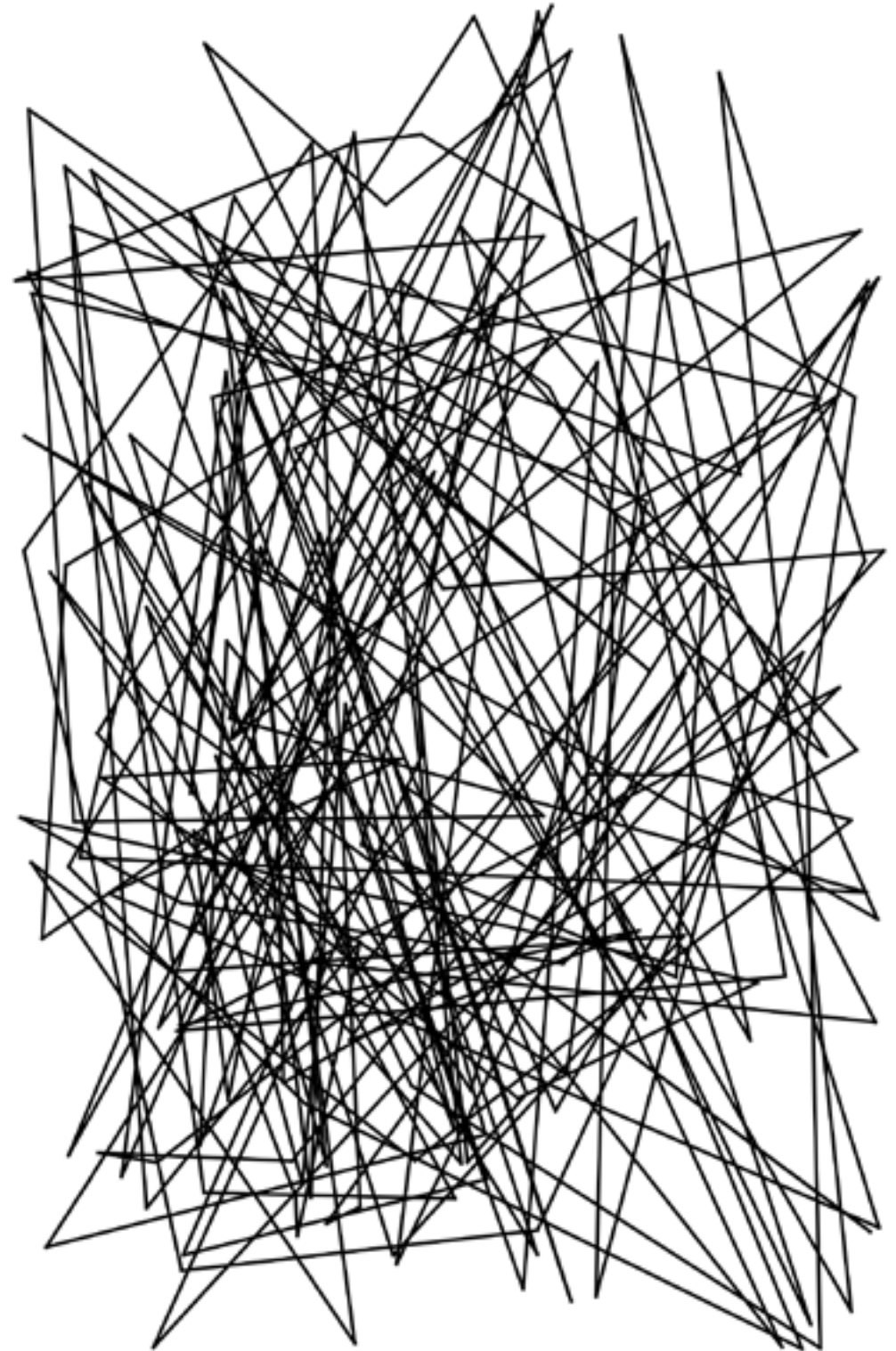


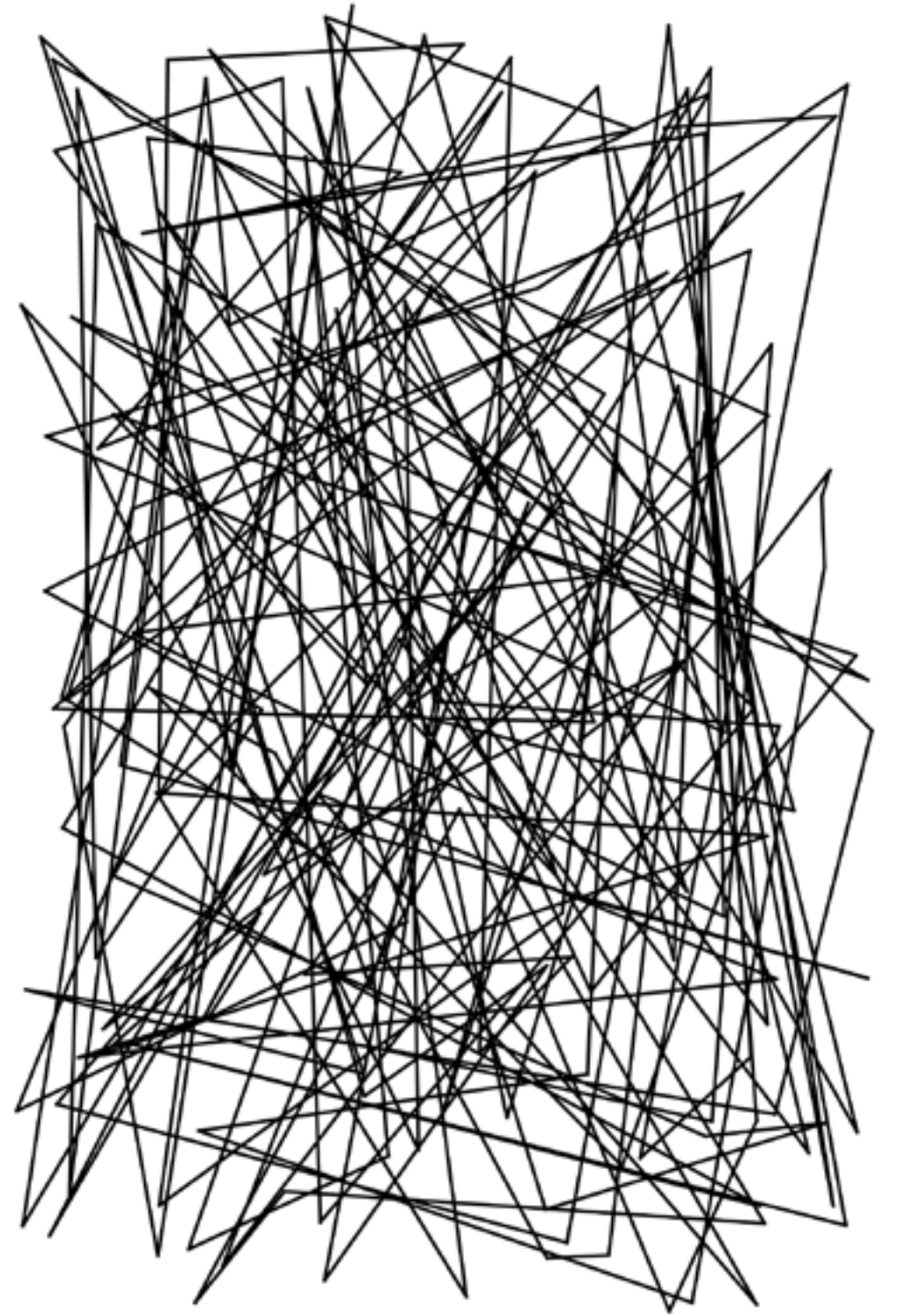
## The Nineteenth Line

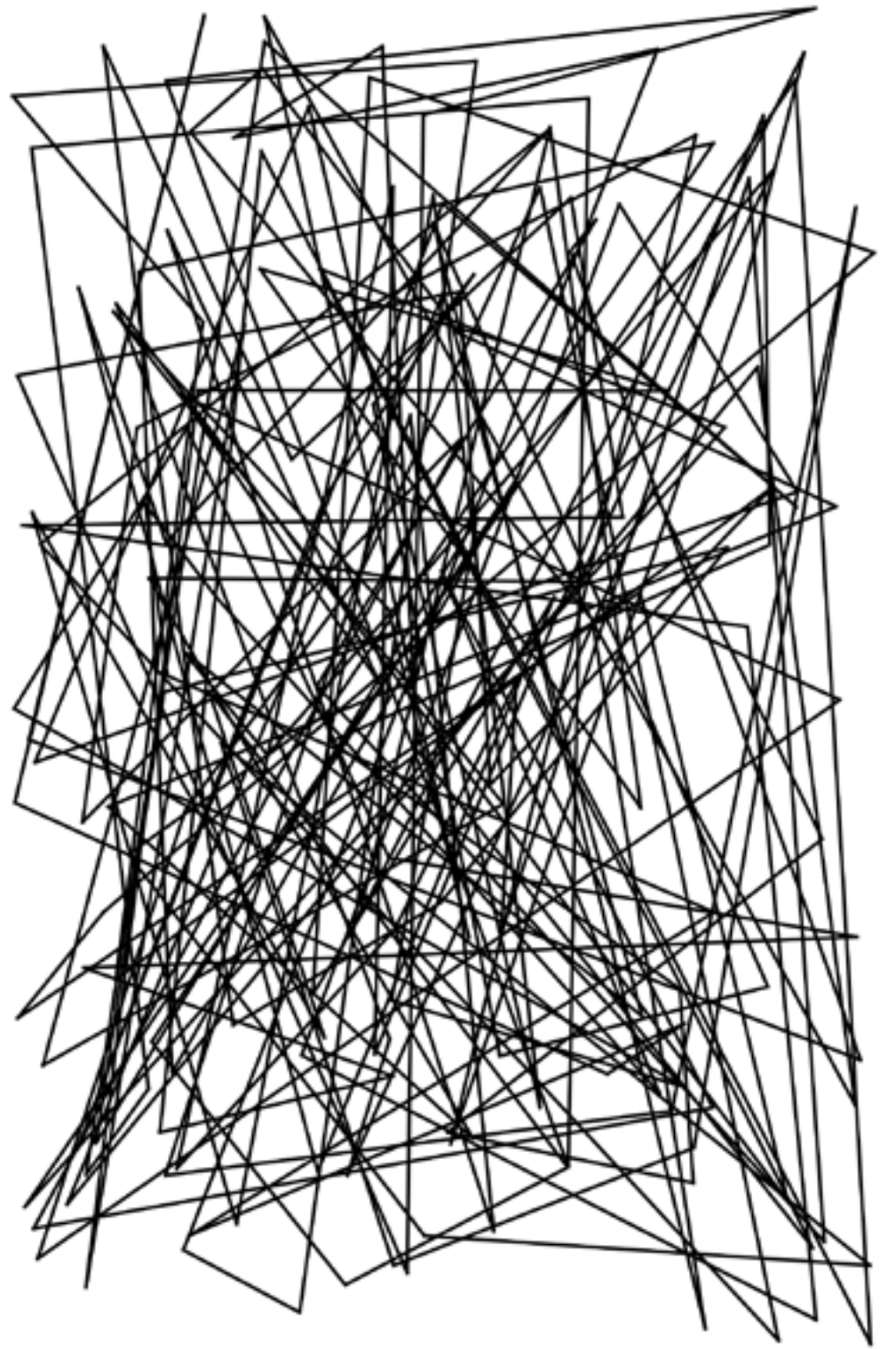
A homage is one's own work inspired by their interpretation of the original work. When Frieder talked about his algorithm, I wondered about the complexity. And even though he called it *cheap*, my first idea was *even cheaper*. Why not pick points randomly and connect them with each other? Put the homage on the algorithmical level instead of the visual one! That is what I have been trying to do. And the idea is still fantastic, my own style on the meta-level.

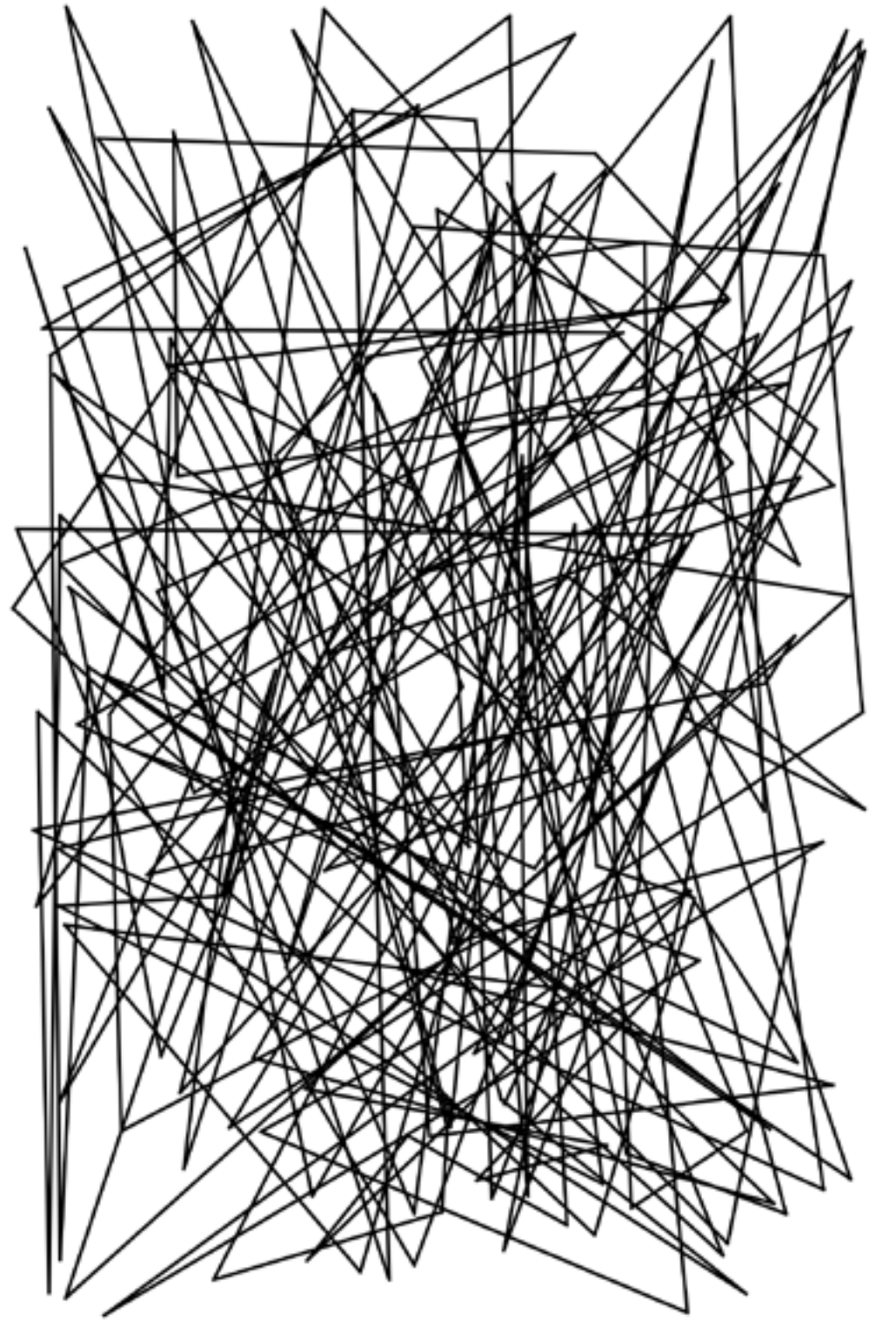
```
int frame = 200;
int howMany = 19*10;
float[] x = new float[howMany];
float[] y = new float[howMany];
for (int i = 0; i < howMany; i++)
{
    x[i] = random(frame, width-frame);
    y[i] = random(frame, height-frame);
}
for (int i = 1; i < howMany; i++)
{
    line (x[i-1], y[i-1], x[i], y[i]);
}
line (x[howMany-1], y[howMany-1], x[0], y[0]);
```

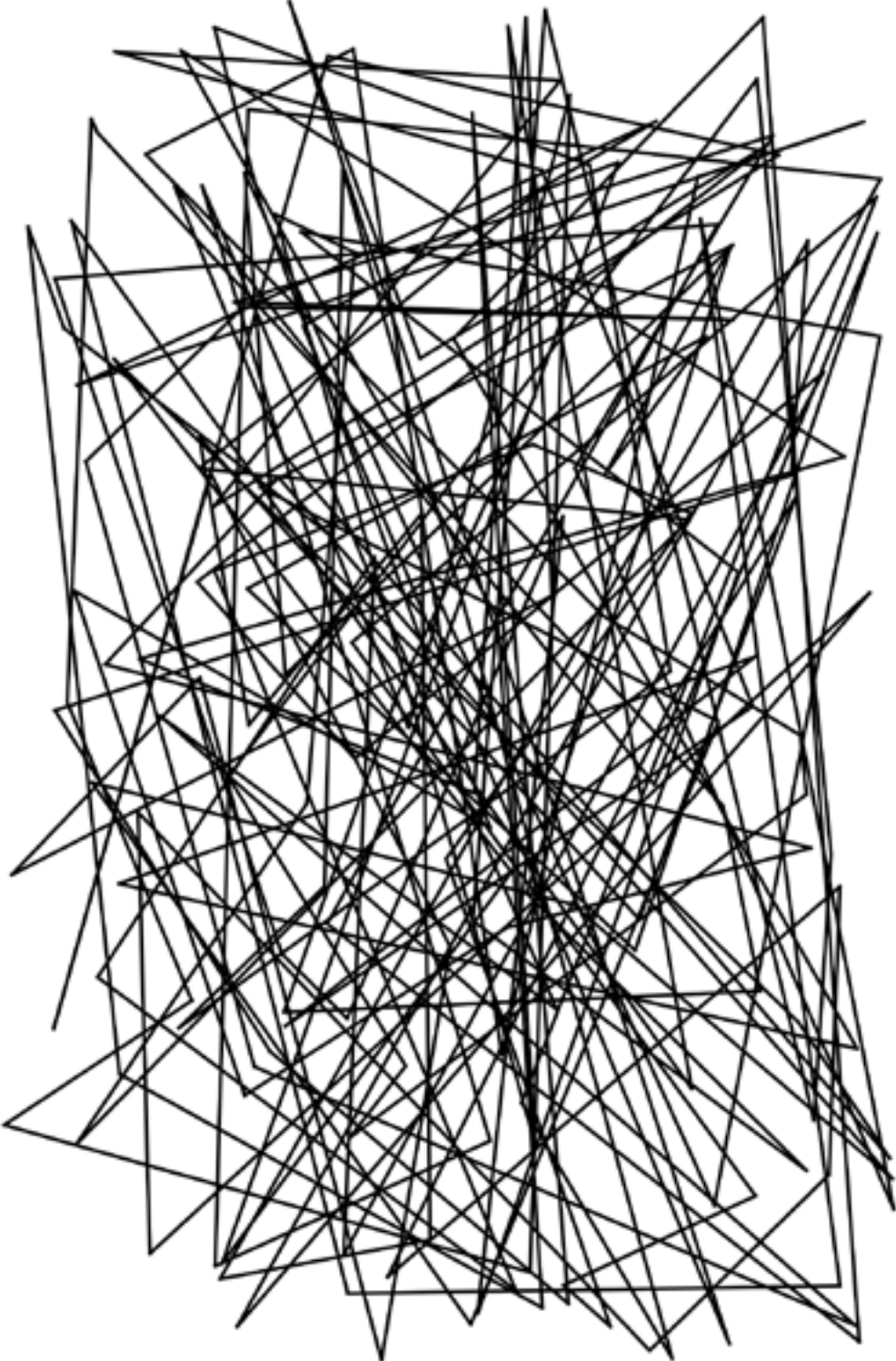
But there is a problem. Even though Frieder is denying it, there is more to his algorithm. We can see it! His result looks interesting, mine does not. Or at least there is a clear visual difference. My idea is useless, if there is a clear visual difference. Maybe it is time to take a closer look at Frieder's *cheap* algorithm.

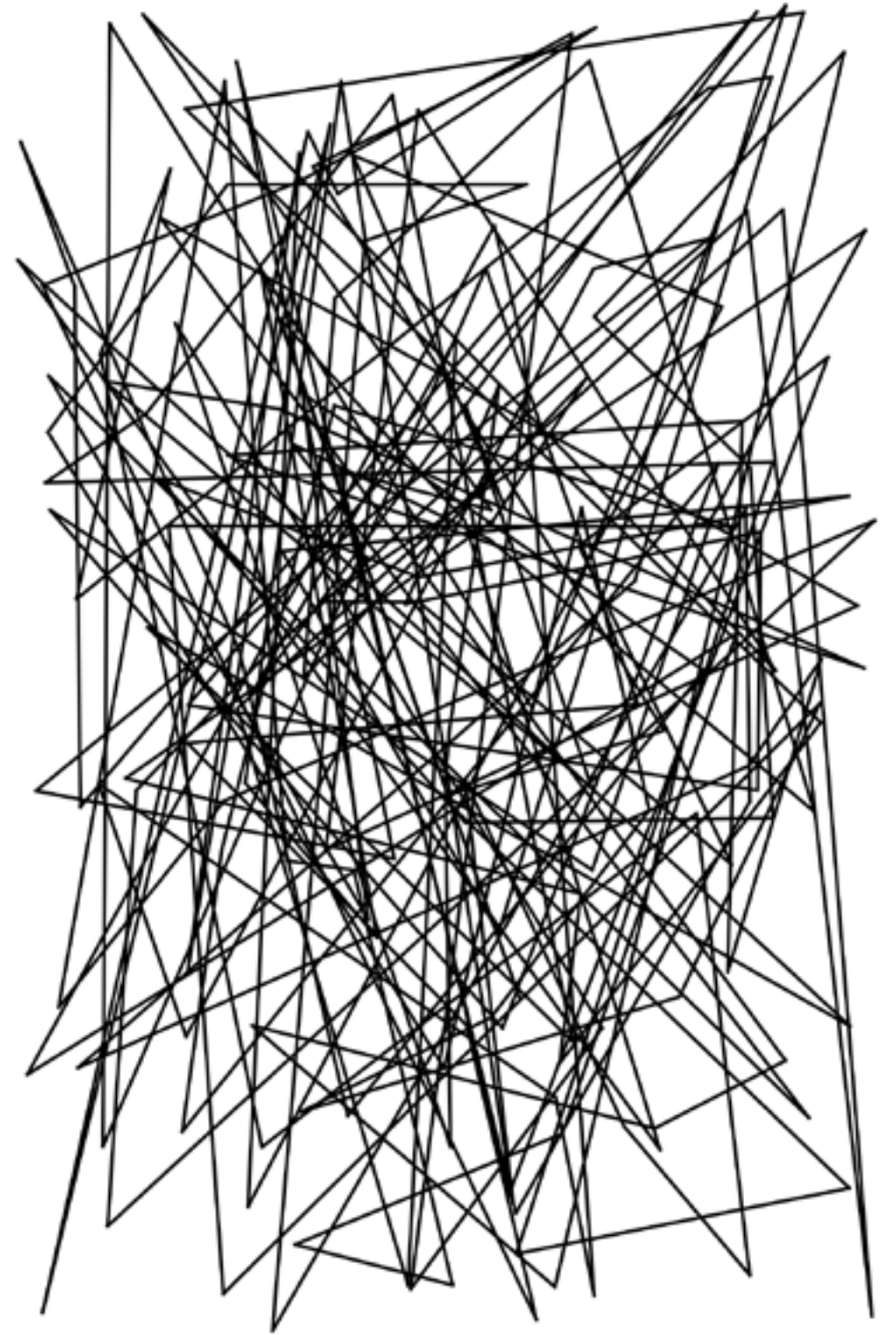


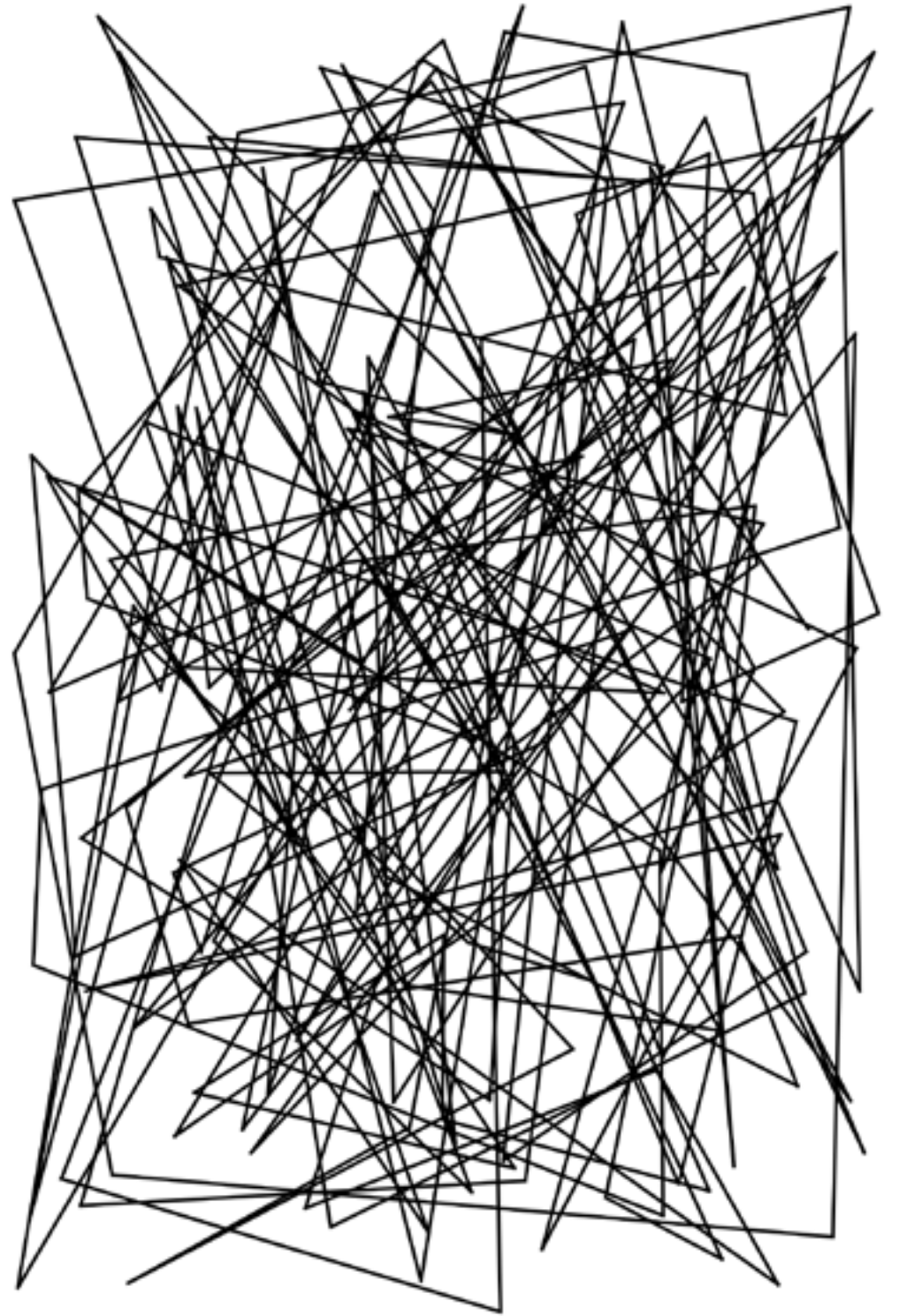


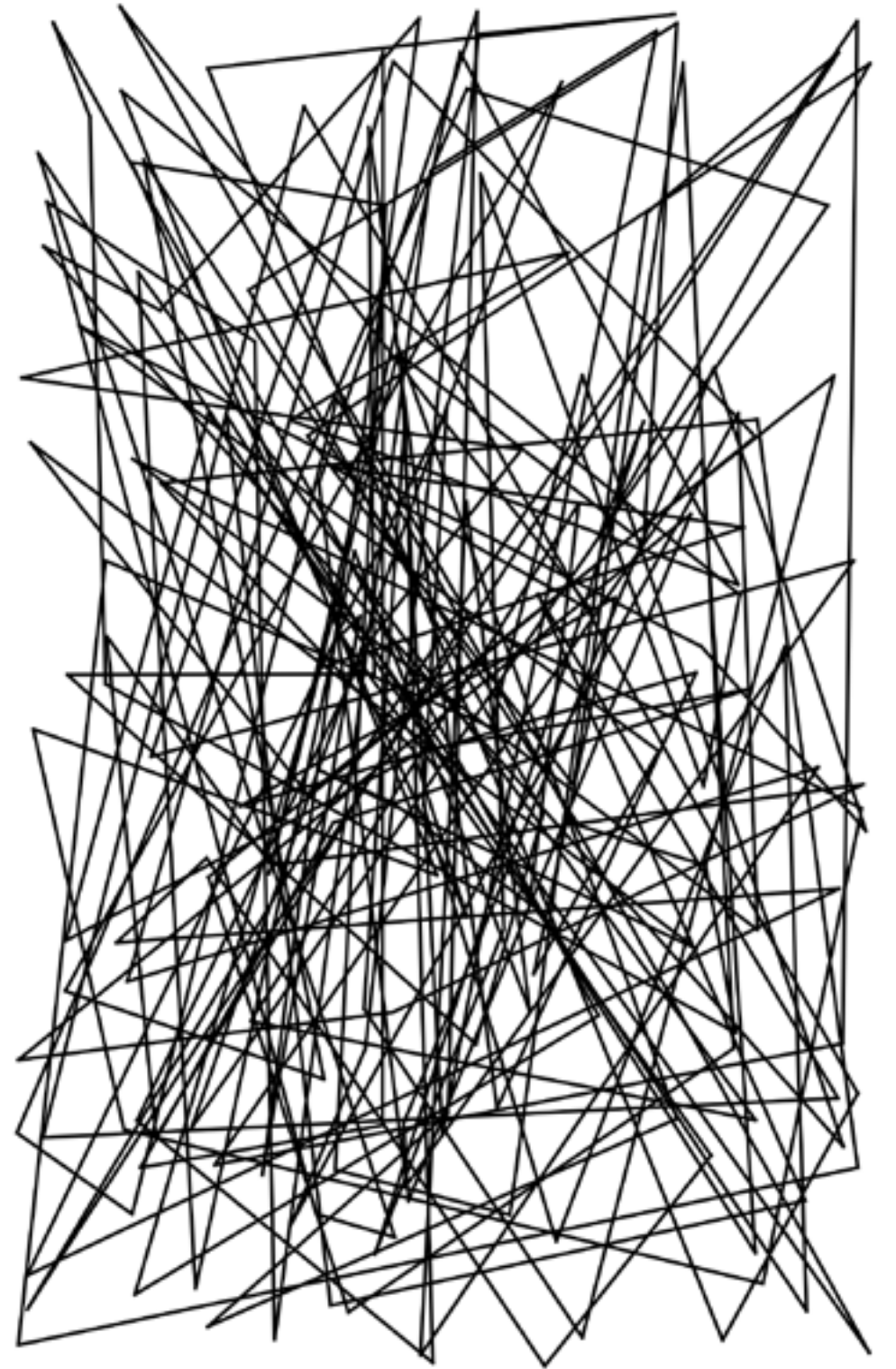


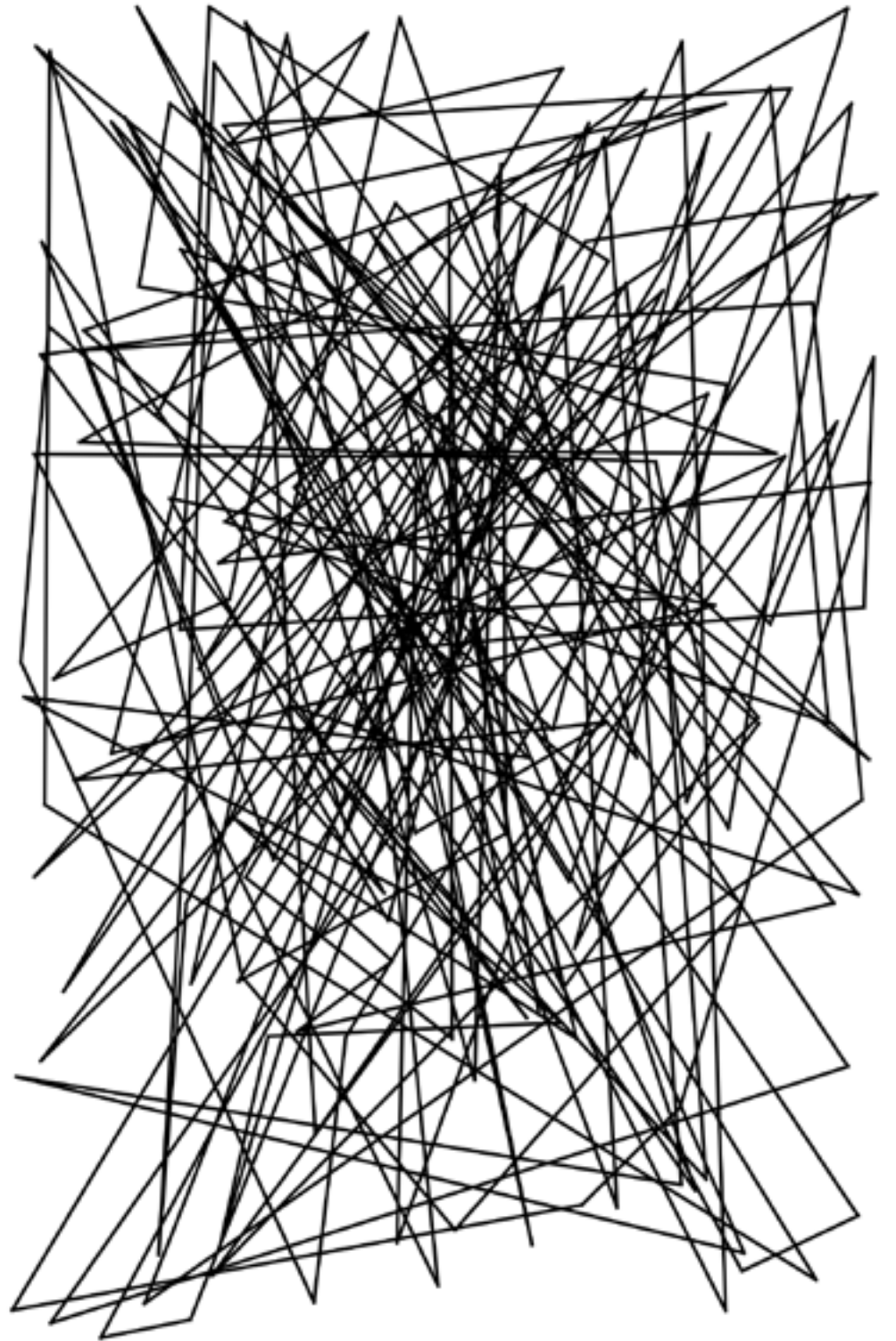


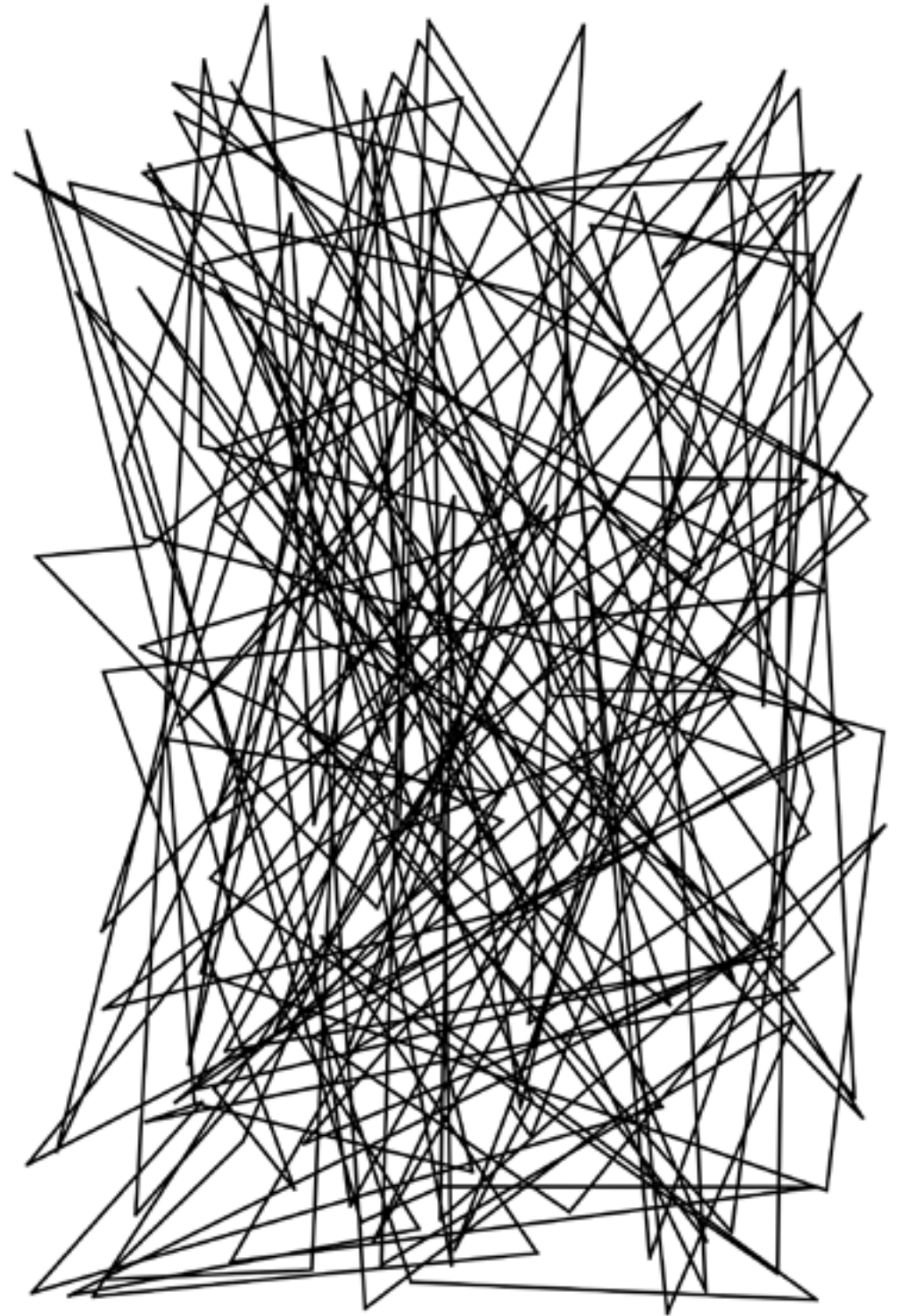












## The Twentieth Line

Reproducing Frieder's homage-style on himself did not really work.  
Creating the same visual output using my input did not really work.  
Finally the time has come to examine Frieder's original creation.

*6/7/64 Nr. 20 Zufälliger Polygonzug.* This piece is completely generated using an algorithm. While staying within the drawing's boundary, a random number of continuing lines is created using random directions and lengths. Finally, the starting and ending point are connected using the Friederline.

Aha! Someone has power over directions and lengths. The random-algorithm maybe?

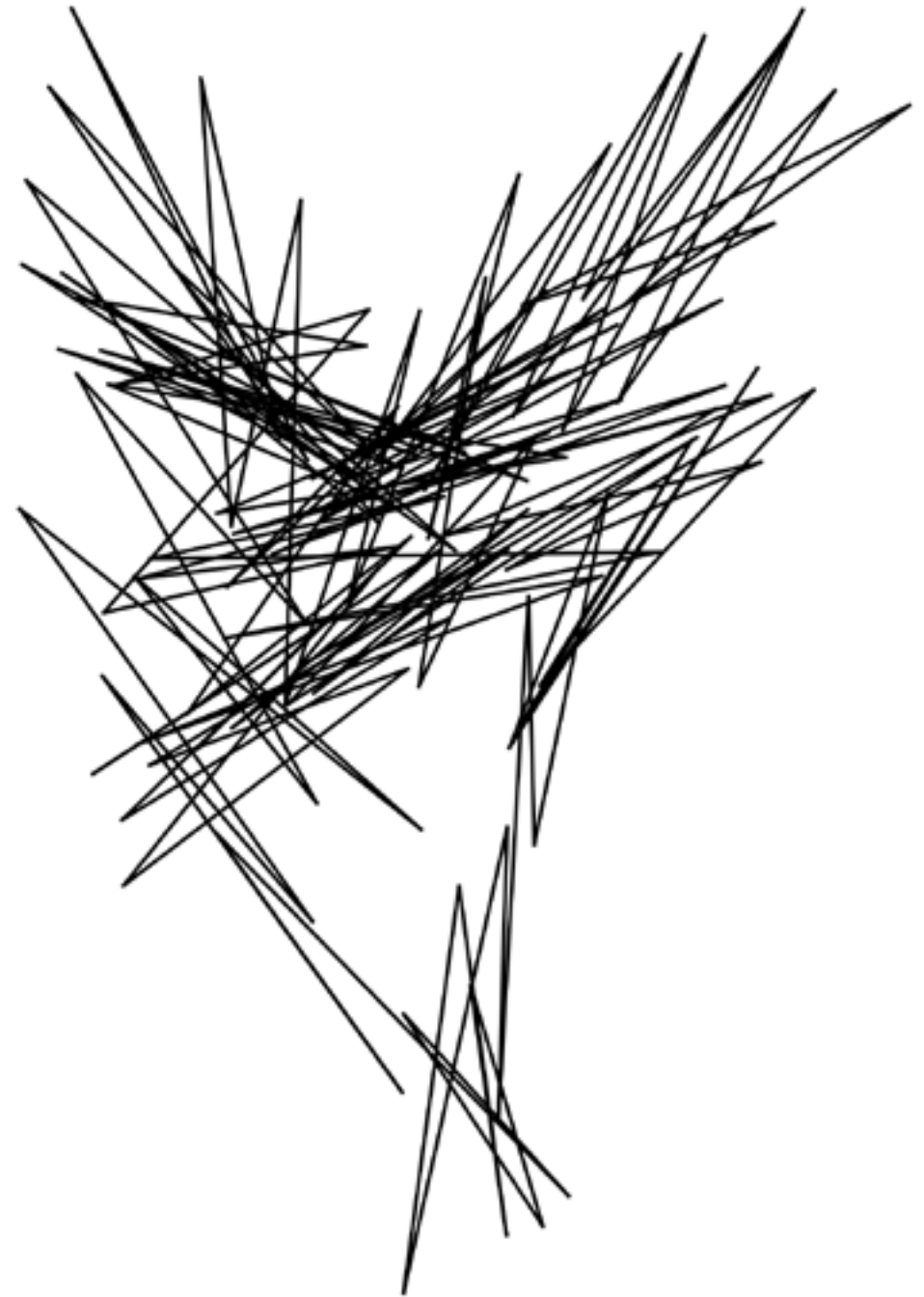


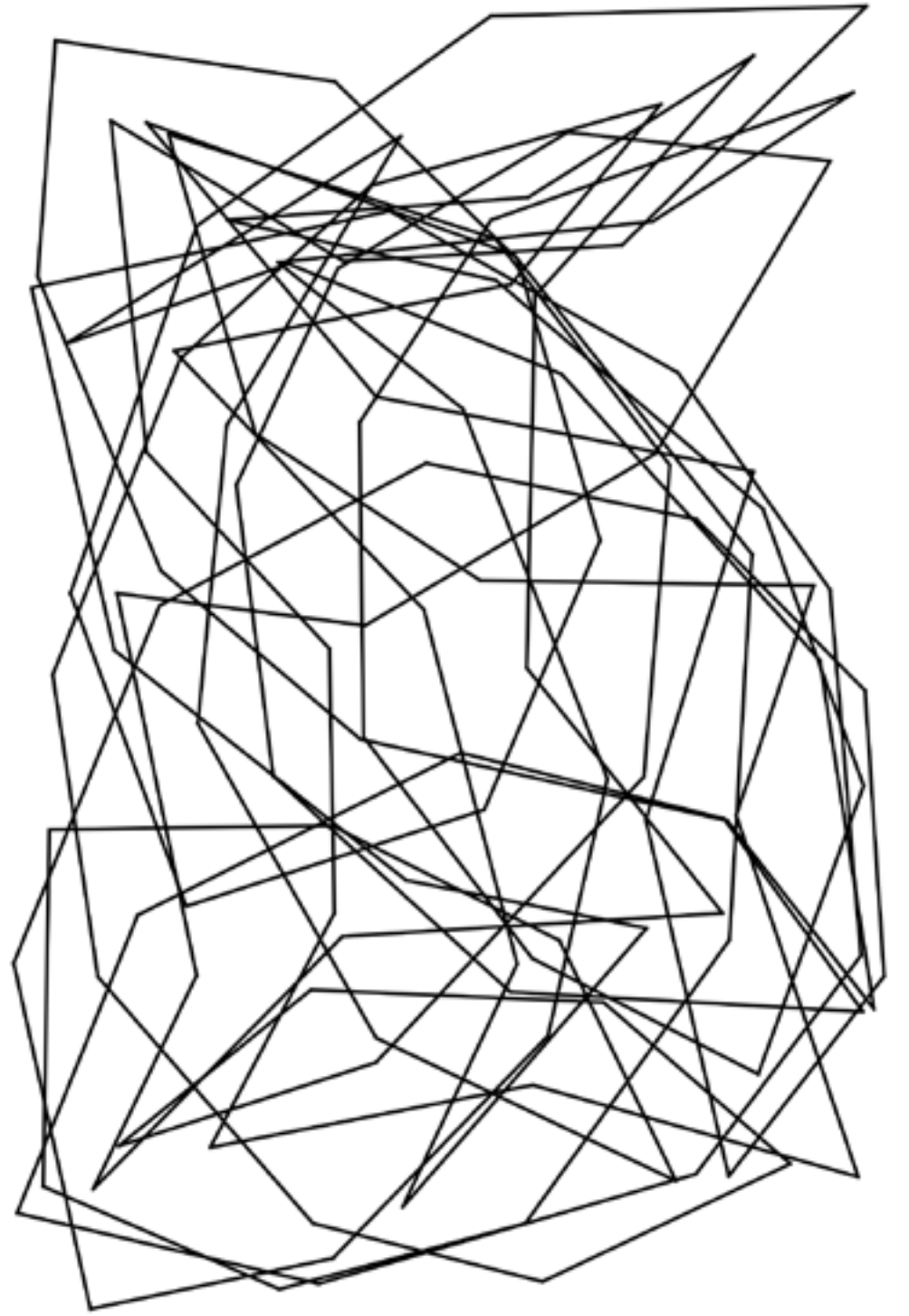
## The Second Last Line

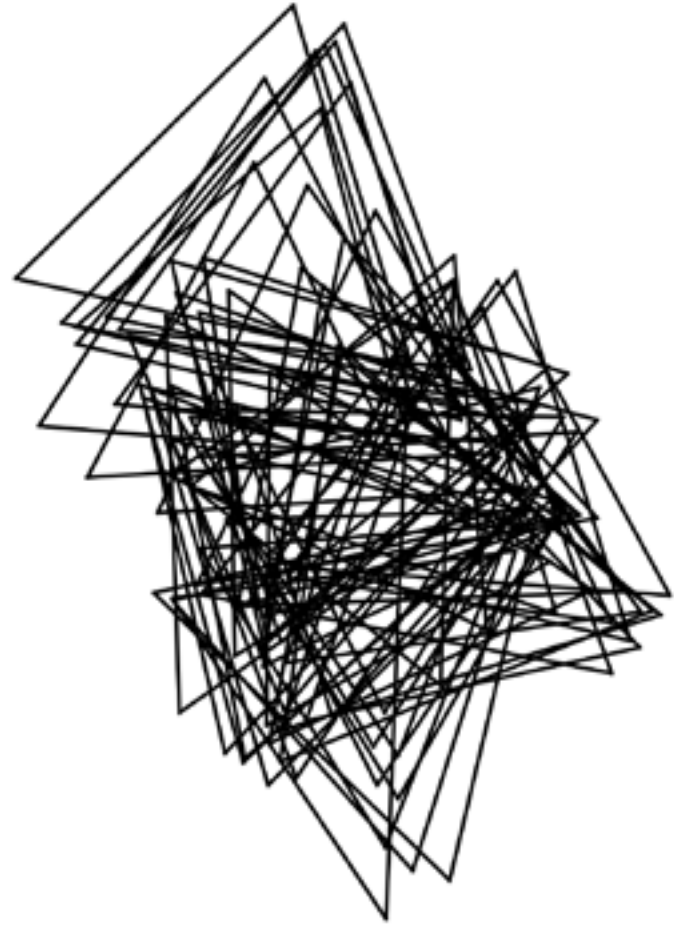
Trying to recreate Frieder's cheap algorithm. Trying. Apparently it is not as cheap as he claims. Or maybe the random-algorithm does not reproduce a picture like his. Or maybe Frieder also created an endless amount of pictures until he was satisfied.

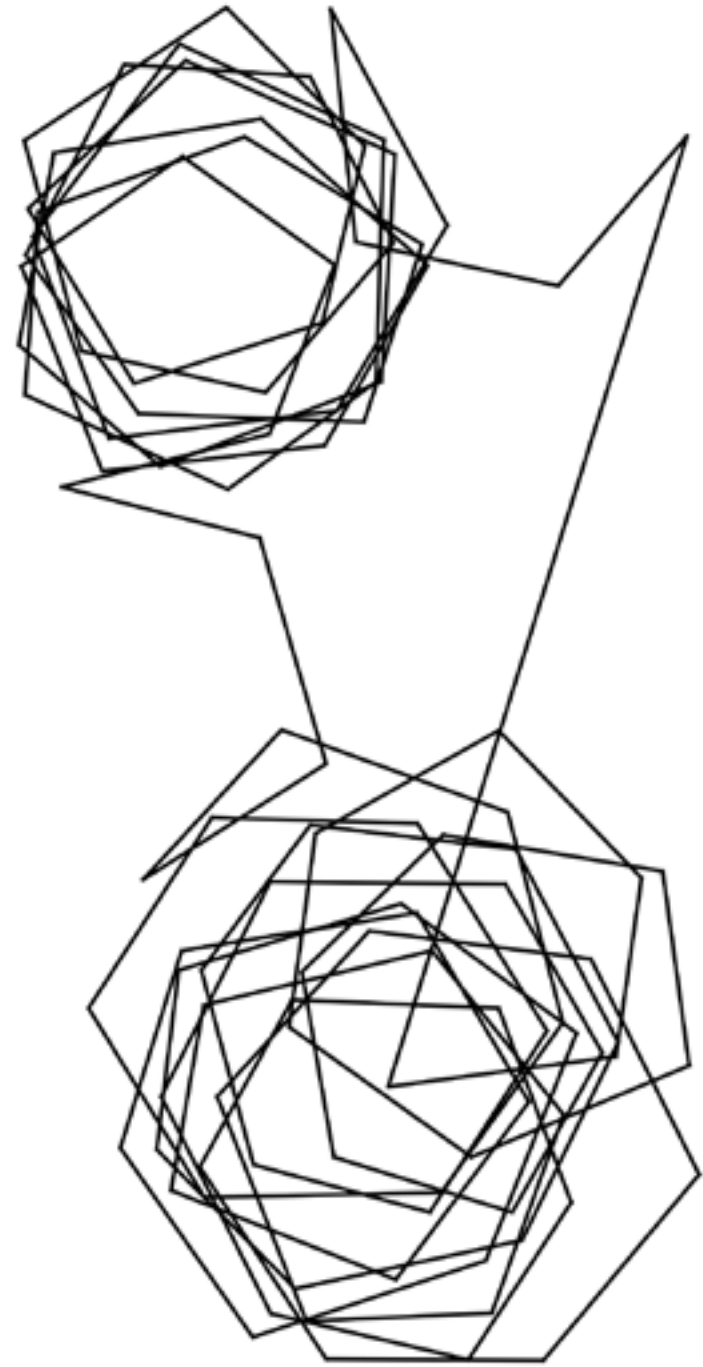
Unfortunately I cannot stop here. Trying to reproduce Frieder's algorithm is fun and it looks quite interesting. But the title of this work says *homage*. I like keeping it tidy.

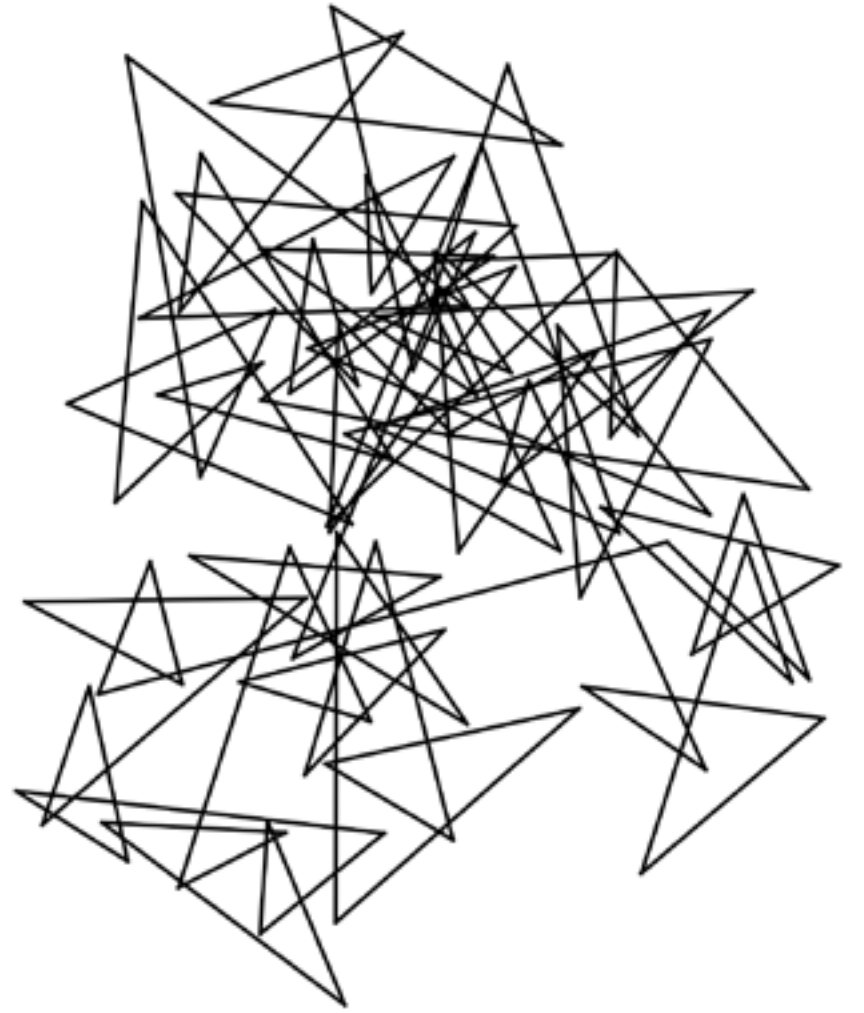
Unlike Frieder. Frieder is chaotic. I know him, I know his office, I have even seen his computer. He cannot even find his own books! I thought I need to keep the chaos. But maybe all I have to do is tidy it up!











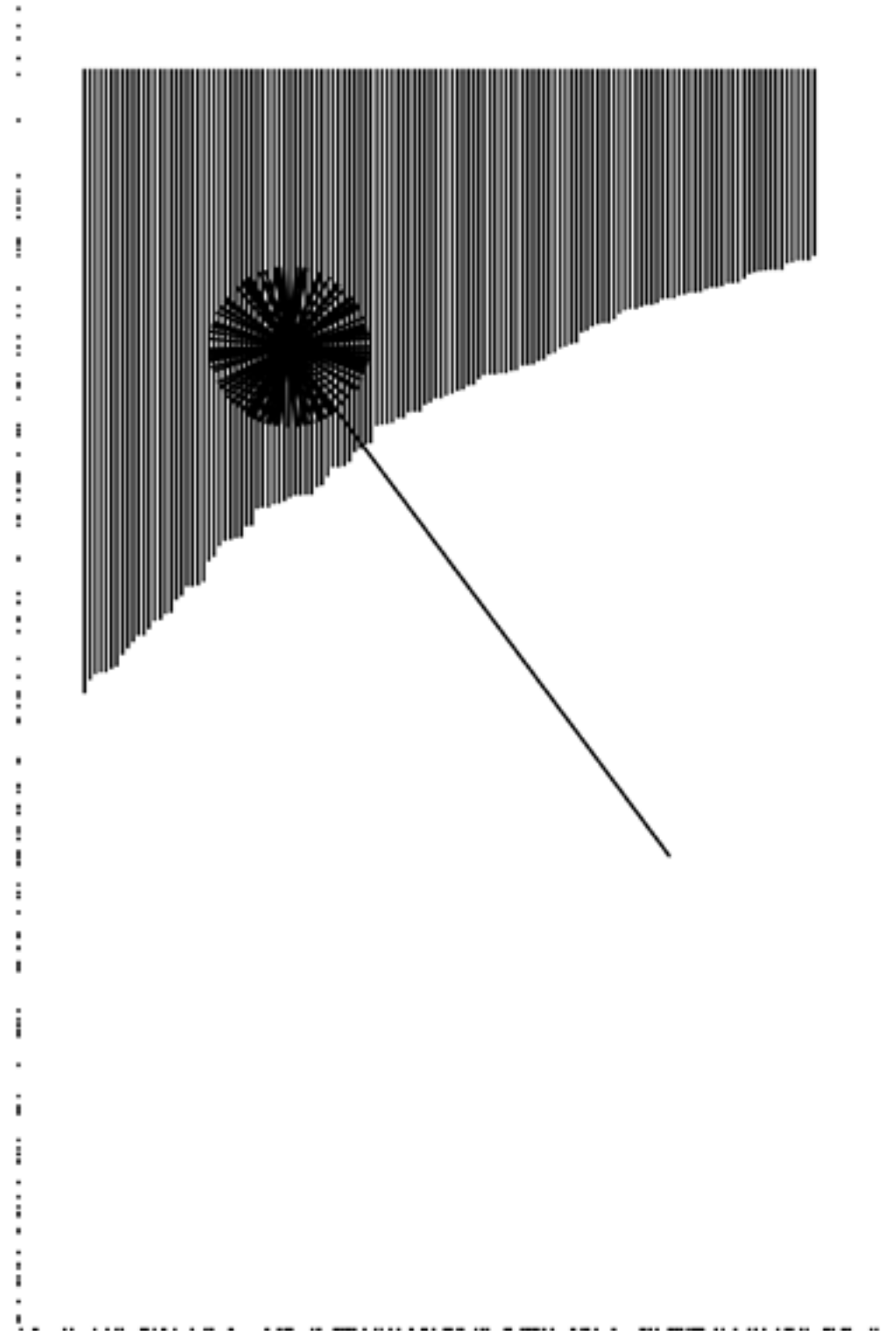
## The Last Line

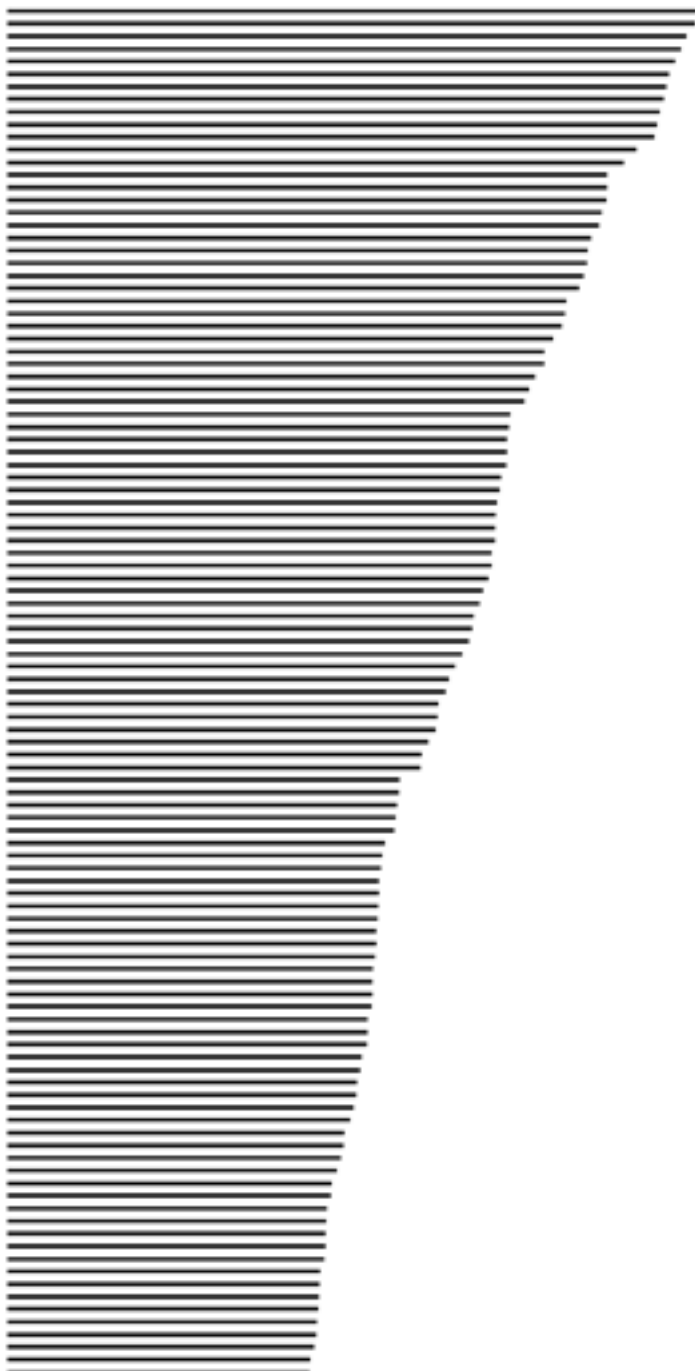
I am tidy. My flat is tidy. I like it tidy. Time to clean Frieder's mess, his artificial art, and put the computer back to what it is good at: calculating and tidying up.

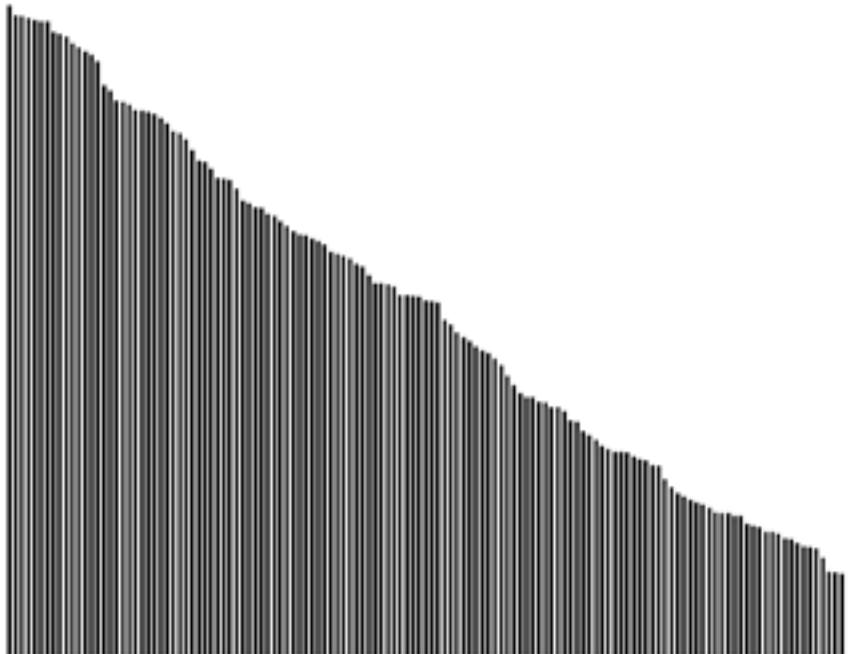
Let's take a look back. On the fourth line I decided to stick with simple input creating chaotic output. I changed my mind. Trying to recreate Frieder's algorithm was not as *cheap* as he claims. Even though my algorithm is still on a low level of complexity, tidying up is more exhausting than messing around. On the thirteenth line I wanted to use Frieder's algorithm as some kind of input, but I did not know how. Now I do. And it is fantastic! I am not changing the originally created algorithm. All I am doing is displaying it differently.

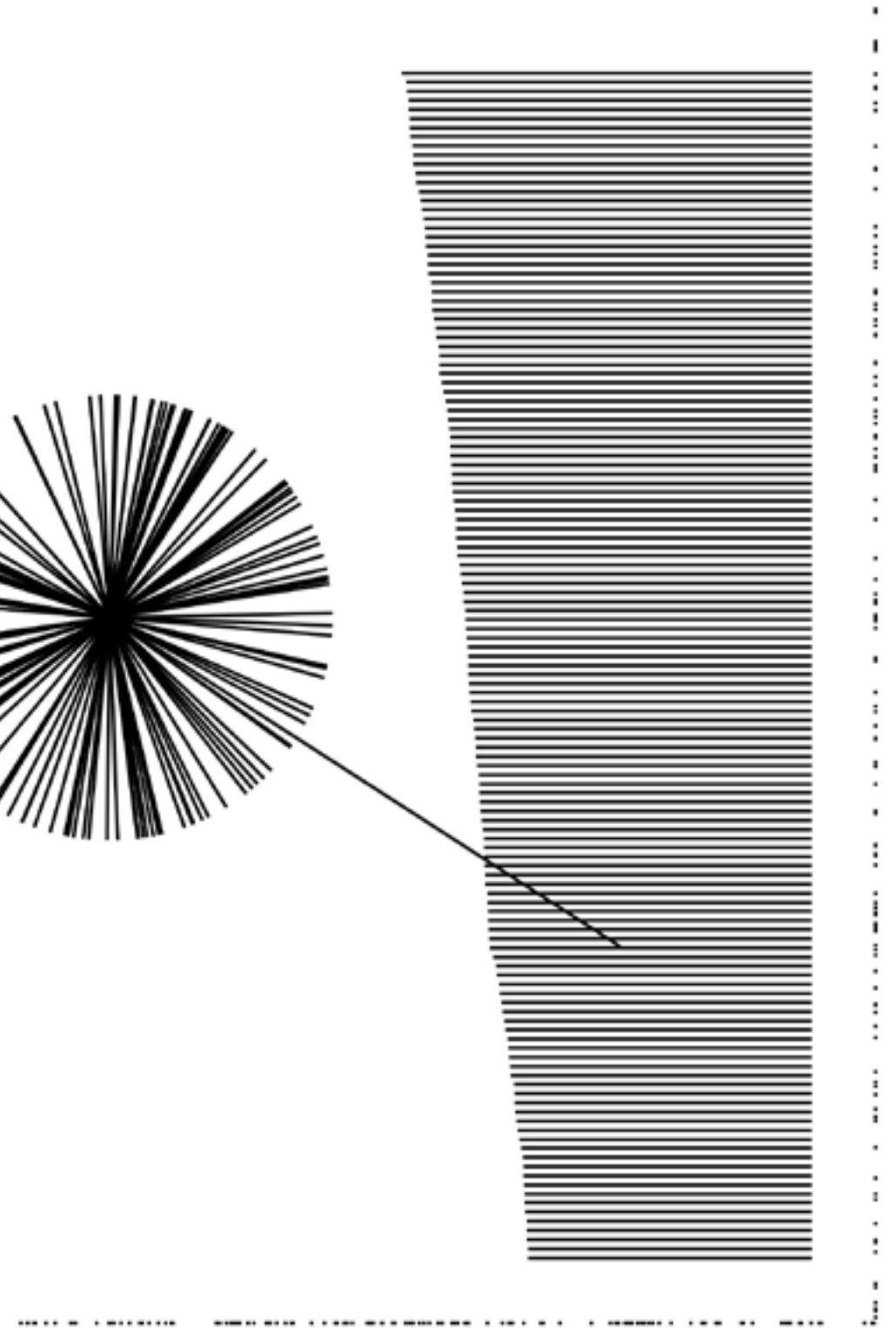
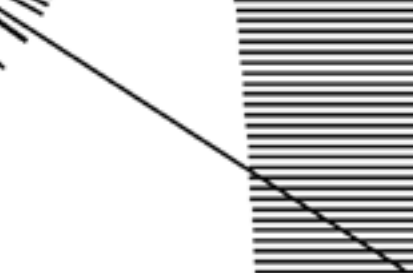
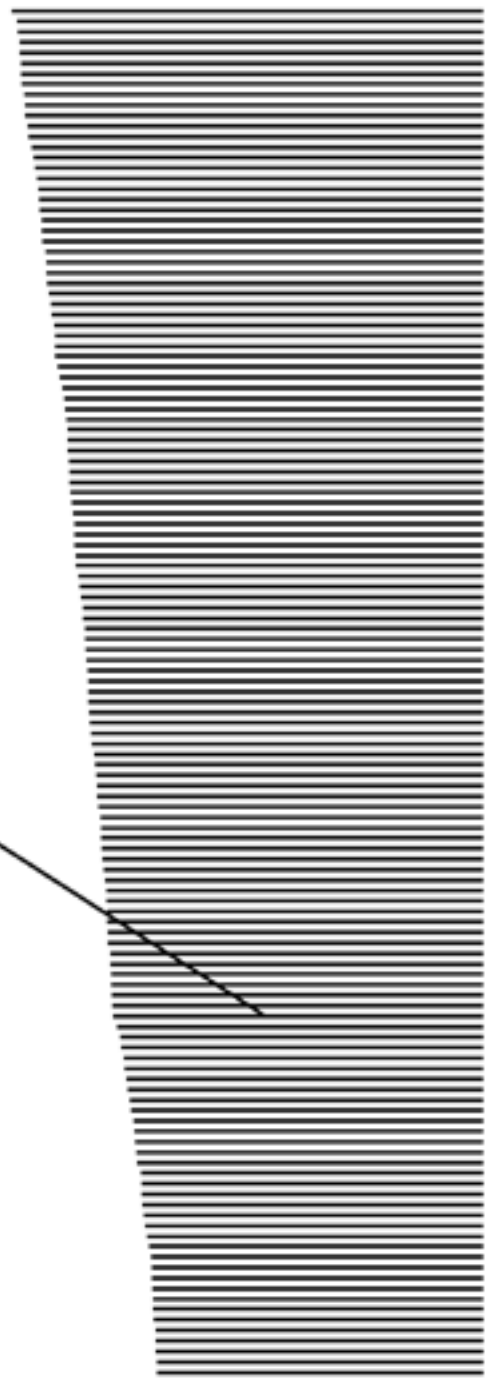
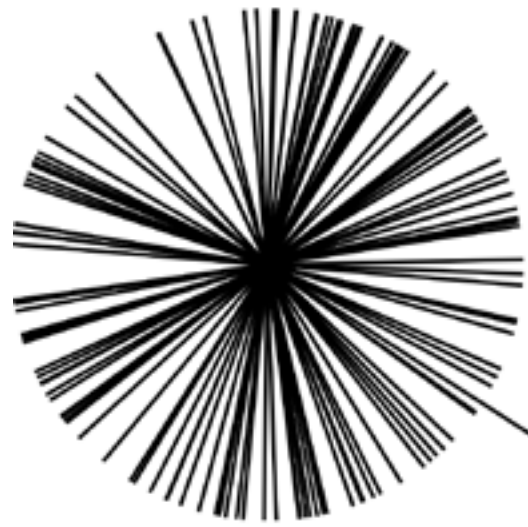
This algorithm does not have a lot of possibilities for creating differences in the pictures. But this is how tidying up works. If you do not invest into anything new, it will look the same.

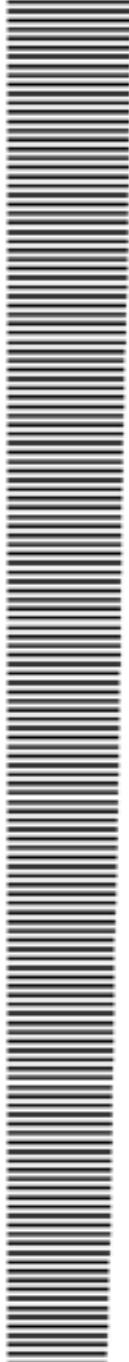
Oh, and never forget about the mighty Friederline. It keeps the picture alive.

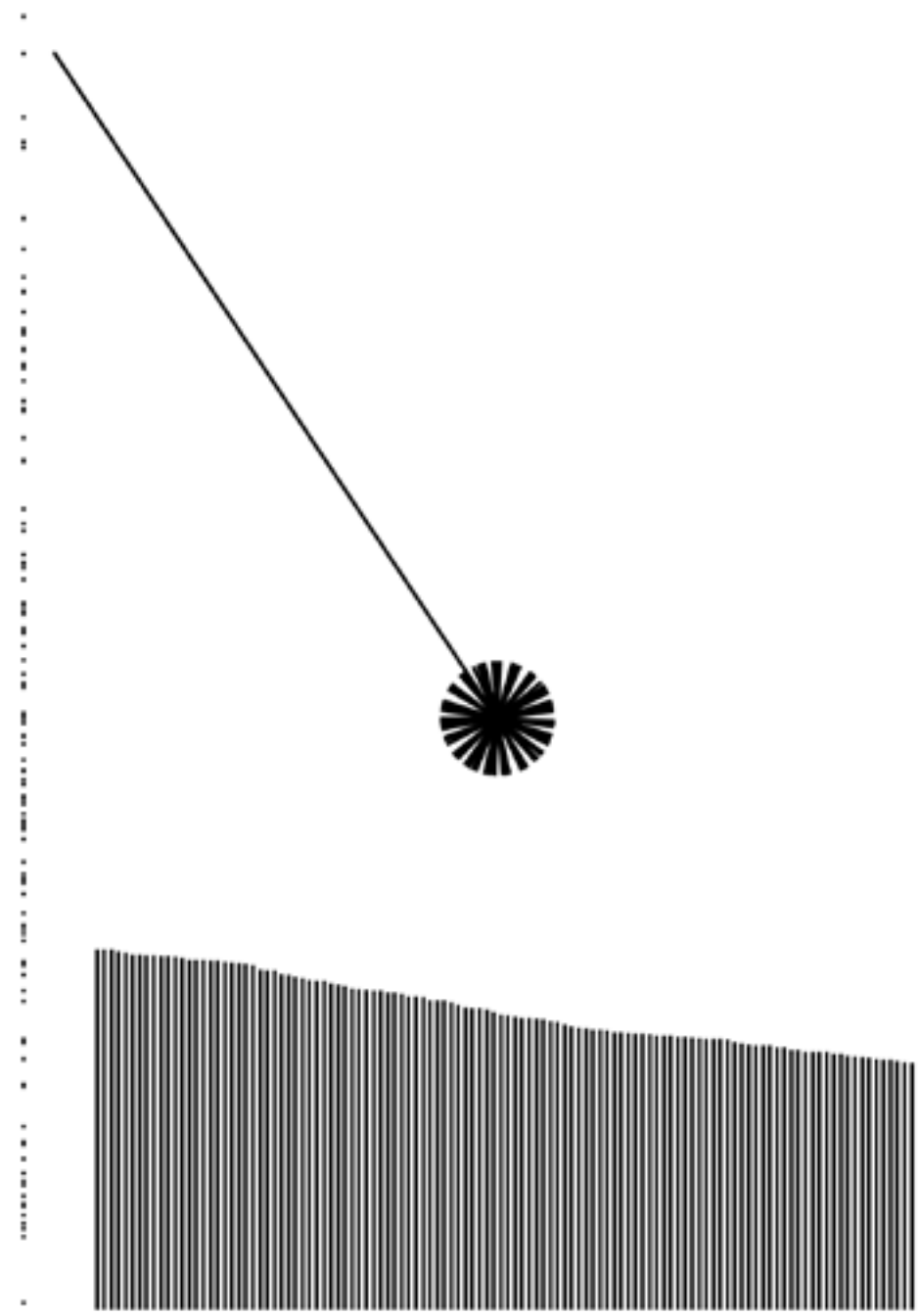






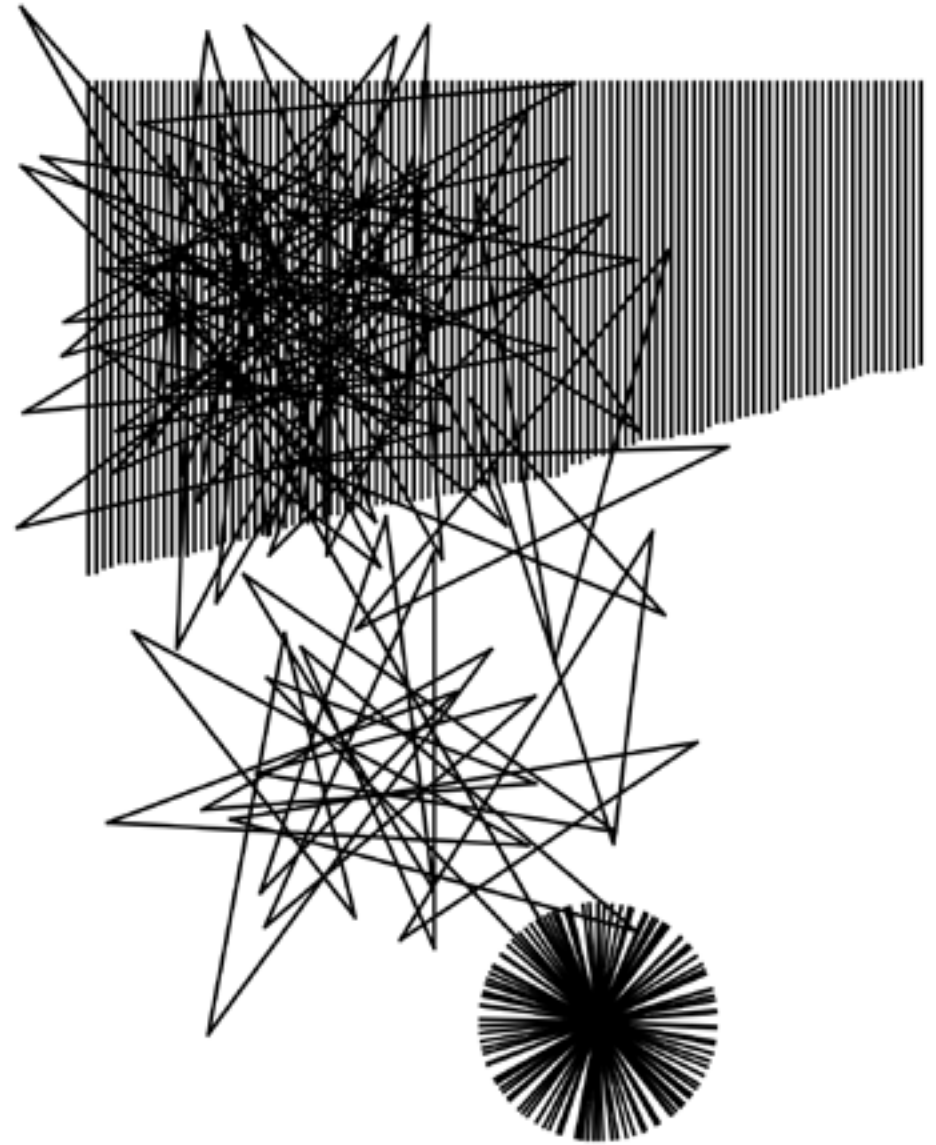








Let's share the beauty of both.





## In\_Ordnung.pde

```
/*
The Last Line. A try on reproducing

FRIEDER NAKE
6/7/64 Nr. 20 Zufälliger Polygonzug

and tidying it up.

By LARA STUMPF
*/

// PERSONALITY OF THE PICTURE
int frame = 200; // lines don't touch the frame

int minLines = 100;
int maxLines = 150;
int howMany = int(random(minLines, maxLines)); // number of lines

float minAngle = random(0, 360);
float maxAngle = minAngle + random(0, 40); // range of angles
int howManyTriesWithAngleAllowed = int(random(0, 30)); // how many tries for
combining angle+length

float minLength = random(100, 500);
float maxLength = random(minLength, 1000); // length of lines

// TIDY PERSONALITY
float pointX, pointY; // splitting point-coordinates and mixing them with new ones
float lengthFrame = frame*1.5; // frame for displaying lengths
boolean left, top, right, bottom; // new position for the lengths
float lengthX, lengthY; // new coordinates for the lengths
float lengthSpace; // how much space for one length
float angleX, angleY; // position for displaying angles
float angleLength; // length for lines on angleflower

// VISUAL
color cBackground = color(255, 255, 255);
color cStroke = color(0, 0, 0);
int wStroke = 5;

// LINE STORAGE
float[] xF = new float[howMany]; // x-coordinate for each line
float[] yF = new float[howMany]; // y-coordinate for each line
float[] angleF = new float[howMany]; // angle of each line
float[] lengthF = new float[howMany]; // length of each line
```

```
void setup()
{
  size(1700,2400);
  stroke(cStroke);
  strokeWeight(wStroke);
  strokeCap(PROJECT);
  noFill();

  // STEP 1: RECREATING FRIEDER'S CHAOS

  // the first line, add random (within boundaries) elements to array
  xF[0] = random(frame, width-1-frame);
  yF[0] = random(frame, height-1-frame);
  angleF[0] = random(minAngle, maxAngle);
  lengthF[0] = random(minLength, maxLength);

  // the next lines
  for (int i = 1; i < howMany; i++)
  {
    int howManyTriesWithAngle = 0; // reset for every line

    // while the calculated point is not inside the picture: try again
    while (xF[i] <= frame || xF[i] >= width-1-frame || yF[i] <= frame || yF[i] >= height-1-frame)
    {
      lengthF[i] = random(minLength, maxLength); // new length
      angleF[i] = angleF[i-1] + random(minAngle, maxAngle); // add new to old angle

      // maybe the angles and lengths don't work together on the current position
      // after a few tries completely dismiss our set boundaries
      if (howManyTriesWithAngle >= howManyTriesWithAngleAllowed)
      {
        angleF[i] = random(0, 360);
      }

      // calculate current point
      xF[i] = xF[i-1] + cos(radians(angleF[i])) * lengthF[i];
      yF[i] = yF[i-1] - sin(radians(angleF[i])) * lengthF[i];

      howManyTriesWithAngle++;
    }

    // cleaning angles over 360 degrees
    if (angleF[i] > 360)
    {
      angleF[i] = angleF[i] - 360;
    }
  }
}
```

```
// STEP 2: LARA TIDYING UP
```

```
// POINTS: splitting points and deciding about left/right and up/down
if (howMany % 2 == 0) // left/right depending on even/uneven of howMany
{
    pointX = frame;
}
else
{
    pointX = width-1-frame;
}
if (sin(howMany) >= 0) // up/down depending on pos/neg sin of howMany
{
    pointY = frame;
}
else
{
    pointY = height-1-frame;
}
```

```
// LENGTHS:
```

```
// deciding about the new positions and calculating the space between lines
// compare length of first old line to possible lengths (is it a quarter? and so on)
if (lengthF[0] < ((maxLength-minLength)/4) + minLength) // < 1/4
{
    left = true;
    lengthX = lengthFrame;
    lengthSpace = (height - 2*lengthFrame) / float(howMany-1);
}
else if (lengthF[0] < ((maxLength-minLength)/2) + minLength) // < 1/2
{
    top = true;
    lengthY = lengthFrame;
    lengthSpace = (width - 2*lengthFrame) / float(howMany-1);
}
else if (lengthF[0] < ((maxLength-minLength)*3/4) + minLength) // < 3/4
{
    right = true;
    lengthX = width-1-lengthFrame;
    lengthSpace = (height - 2*lengthFrame) / float(howMany-1);
}
else // <= 1
{
    bottom = true;
    lengthY = height-1-lengthFrame;
    lengthSpace = (width - 2*lengthFrame) / float(howMany-1);
}
```

```
// sorting the lines and reversing if the first length is an uneven number
lengthF = sort(lengthF);
if (lengthF[0] % 2 != 0)
{
    lengthF = reverse(lengthF);
}
```

```
// ANGLE: put angleflower onto beginning or ending of Friederline?
if (sin(angleF[0]) >= 0) // depending on pos/neg sin of first angle
{
    angleX = xF[0];
    angleY = yF[0];
}
else
{
    angleX = xF[howMany-1];
    angleY = yF[howMany-1];
}
```

```
angleLength = angleF[0]; // length of the lines
angleF = sort(angleF); // tidying up (even though, yes, there is no visual difference)
}
```

```
void draw()
{
    background(255);

    // DRAW FRIEDER
    // draw the lines
    for (int i = 1; i < howMany; i++)
    {
        line(xF[i-1], yF[i-1], xF[i], yF[i]);
    }
}
```

```
// DRAW LARA
for (int i = 0; i < howMany; i++)
{
    // POINTS
    point(pointX, yF[i]);
    point(xF[i], pointY);
}
```

```
// LENGTHS
if (left == true)
{
    line(lengthX, lengthFrame + lengthSpace*i,
        lengthX + lengthF[i], lengthFrame + lengthSpace*i);
}
```

```
else if (top == true)
{
  line(lengthFrame + lengthSpace*i, lengthY,
lengthFrame + lengthSpace*i, lengthY + lengthF[i]);
}
else if (right == true)
{
  line(lengthX, lengthFrame + lengthSpace*i,
lengthX - lengthF[i], lengthFrame + lengthSpace*i);
}
else if (bottom == true)
{
  line(lengthFrame + lengthSpace*i, lengthY,
lengthFrame + lengthSpace*i, lengthY - lengthF[i]);
}

// ANGLES
line(angleX, angleY, angleX + cos(radians(angleF[i])) * angleLength,
angleY - sin(radians(angleF[i])) * angleLength);
}

// but never forget the Friederline
line(xF[howMany-1], yF[howMany-1], xF[0], yF[0]);

save("In_Ordnung.tiff");
}
```

