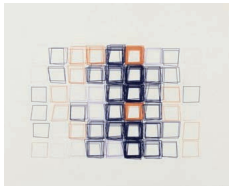
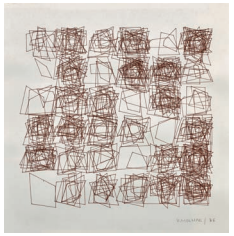
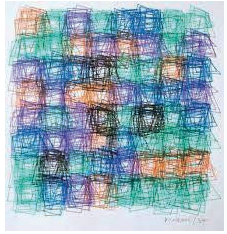
Recoding project algorithmic aesthetics

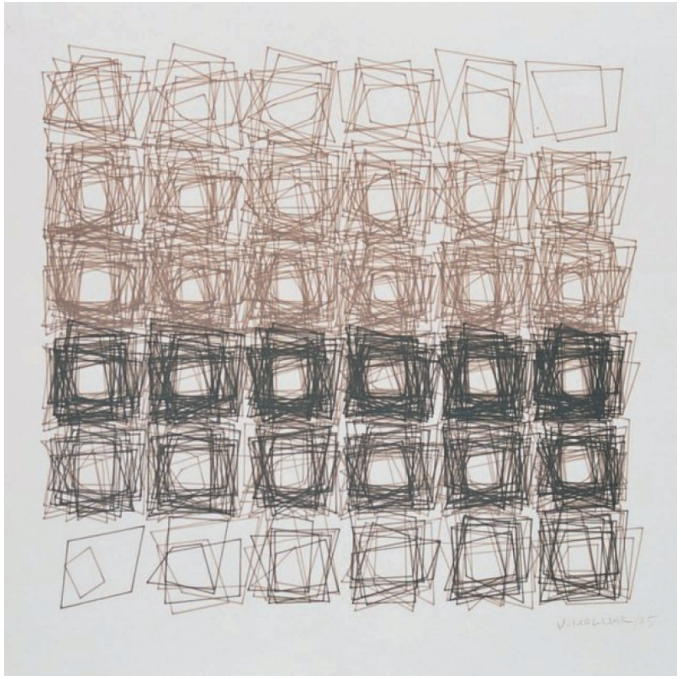
Tristan Debeaune & Wendy Gervais





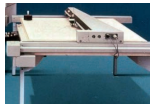


Chosen artwork



Véra Molnar, *Structure de quadrilatère*, 1985

Ink on paper
30 cm x 30 cm
Algorithm on computer + plotter



Vera Molnar is a pioneer in computer art, her work is experimental, she tries and makes programs in order to experiment randomness with shapes - oftenly squares.

Analysis

The version we chose :

- **squared light-grey canvas**
- **black and brown squared lined pattern**
- **6 per 6 grid**

Each cell :

- scribbled with **quadrangles**
- all cells are different, **randomly generated**
- can overlap its neighbours

They are **variable in numbers and colours :**

- **3 first rows** : brown, the density of lines grows row by row, right to left, from top right (2 quadrangles) to bottom left (about thirty).

- **3 last rows** : superposition of half brown and half black lines. Mirrored repartition (the density increases row by row from bottom left to top right).

Like Molnar, we have been very interested in the «ordered» chaos created by this grid, and its imbrication of alternative squares ; we are now trying to recreate it with an algorithm.

Pseudo-code

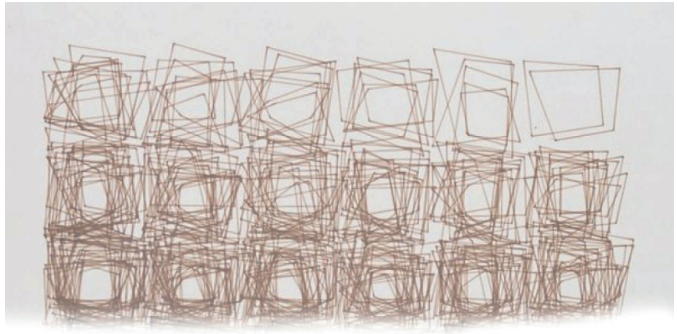
1. Take a brown pen.

2. Start on top-right corner. Draw two deformed superposed squares, their shapes are random.

3. On the left of this first shapes, start again drawing 3 more random quadrangles over the same position.

4. Repeat on the row until having 6 squares increasing the amount of lines each time.

5. Repeat this 2 more times, starting from the right every row.

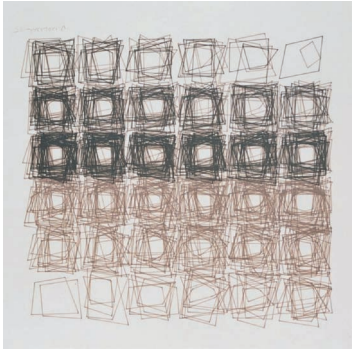


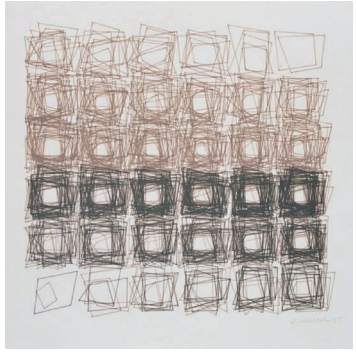
6. Rotate the canvas upside down.

7. Take also a black pen.

8. Repeat the 5 first steps. But, for each shape, begin by drawing the first half of the lines brown, then the second half black.

9. Rotate the canvas upside down again.





Pseudo-algorithm

```
function setup()
    create a 650*650 canvas
    DrawMesh(6, 6)
```

step 2

```
function DrawSquare(quad_amount, column, line)
    push()
        translate by 60*column on x axis, 60*line on y axis, 0 on z axis
        for k from 0 to quad_amount, with a step of 1
            dispersion <- []
            add 8 random relative integers between -10 and 10 to the dispersion array
            draw a quadrangle centered on the square generated by the 4 vertices (40,40),
(80,40), (80,80), (40,80) using the 8 integers of dispersion added to the 8 coordinates
        end for

    pop()
```

steps 1 to 5

```
function DrawMesh(width, height)
    for j from 0 to height/2, with a step of 1
        for i from width to 0, with a step of 1
          ijquad_amount <- 2*(width-i-1 + width*j + 1)
          draw in brown
            DrawSquare(ijquad_amount, i, j)
        end for
    end for
```

steps 6 to 8

```
    for j from 0 to height/2, with a step of 1
        for i from width to 0, with a step of 1
          ijquad_amount <- width-i-1 + width*j + 1
          drawn in brown
            DrawSquare(ijquad_amount, width-1-i, height-1-j)
          drawn in black
            DrawSquare(ijquad_amount, width-1-i, height-1-j)
        end for
    end for
```

p5 version

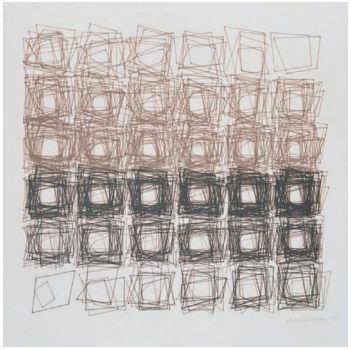
Code «STATIC»

```
// Recoding project 2022
// T. DEBEAUNE & W. GERVAIS
// Vera Molnar - Square Structures

// Static Version

function setup() {
  let img = createCanvas(650, 650);
  background(220);
  scale(1.5,1.5,0);
  DrawMesh (6,6);
  //saveCanvas('img', 'png');
}

// Draw n superposed quadrangles (based on 40*40 squares) position i-column, j-row in the Mesh
function DrawSquare(n, i, j) {
  push();

    noFill();
    translate(60*i, 60*j, 0); // put it on i-column, j-row, with a 20px "distance" between 2 quads
    for (let k=0; k<n; k++) { // n times
      let dispersion = [];
      for (let d=0; d<8; d++) { // generate the dispersion array, filled with 8 random relative integers which will be the dispersions around the 8 coordinates
        let disp = int (random(-12, 15));
        dispersion.push(disp);
      }

      // draw the quadrangle :
      //stroke(20*(i+j),20*i,20*j); // color gradient

quad(40+dispersion[0],40+dispersion[1],80+dispersion[2],40+dispersion[3],80+dispersion[4],80+dispersion[5],40+dispersion[6],80+dispersion[7]);
    }

  pop();

}
```

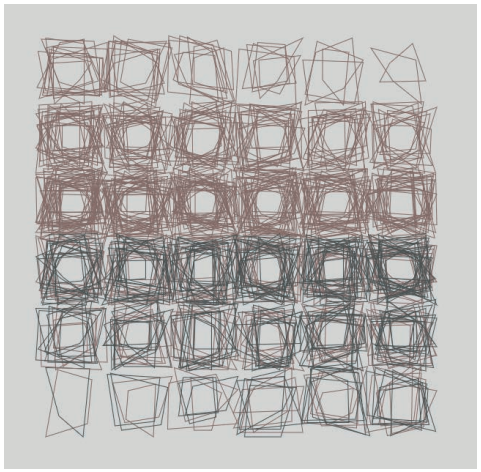
```
// Draw the i*j mesh of n superposed quadrangles
function DrawMesh(w,h) {
  let quad_amount;
  strokeWeight(0.7);

  for (let j=0; j<h/2; j++) {
    for (let i=w-1; i>=0; i--) {
      ijquad_amount=2*((w-i-1+w*j)+1); // increasing from right to left (-i), keep the number increasing from row to row (w*j)

      // 3 first lines (brown)
      stroke("#8f746d");
      DrawSquare(ijquad_amount,i,j);

    }
  }

  for (let j=0; j<h/2; j++) {
    for (let i=w-1; i>=0; i--) {
      ijquad_amount=(w-i-1+w*j)+1; // split in 2 the previous amount (/2) to draw half brown and half black

      // 3 last lines (brown)
      stroke("#8f746d");
      DrawSquare(ijquad_amount, w-1-i, h-1-j); // positions = start bottom right ("rotate")

      // 3 last lines (black)
      stroke("#515757");
      DrawSquare(ijquad_amount, w-1-i, h-1-j);

    }
  }

}
```

p5 editor - https://editor.p5js.org/kau.grv/sketches/OC2bP--qy

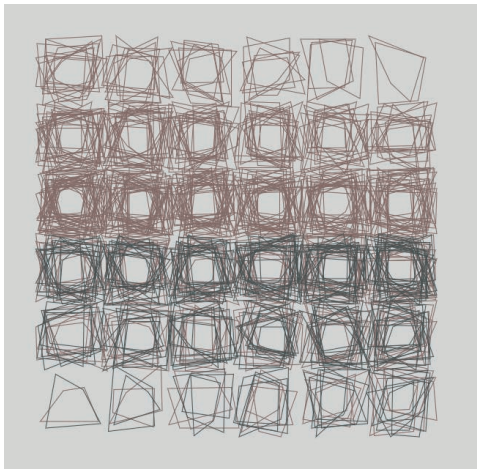
Outputs



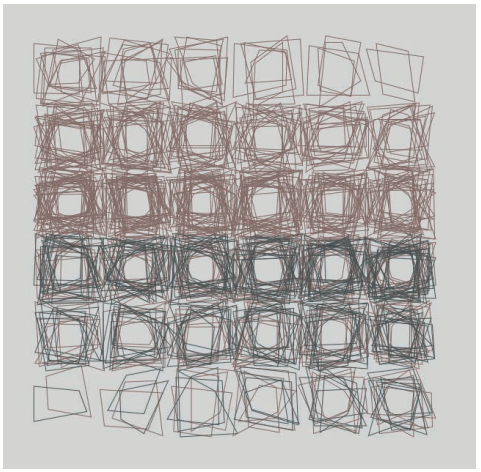
Original artwork



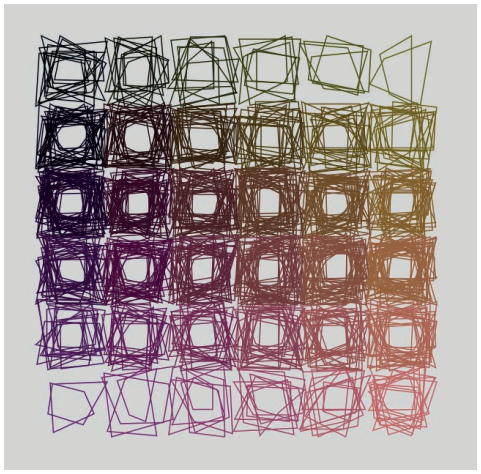
#1



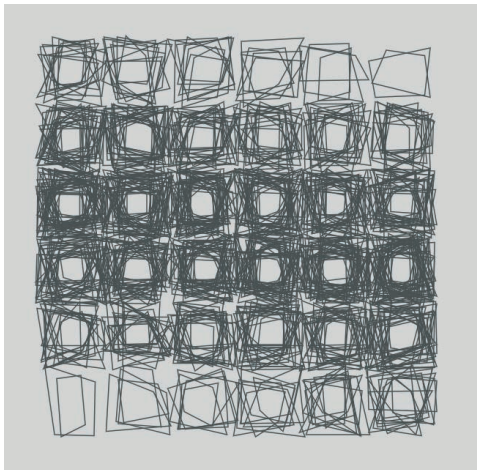
#2



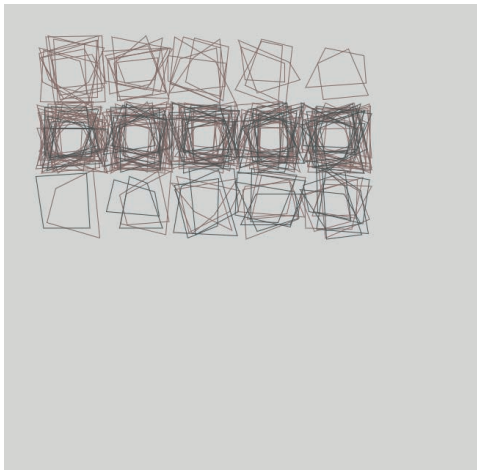
#3



with a color gradient



all black + 1px stroke weight

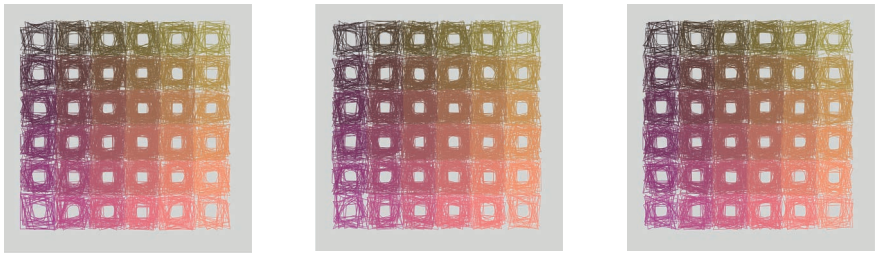


5*3

Variations «DYNAMIC»

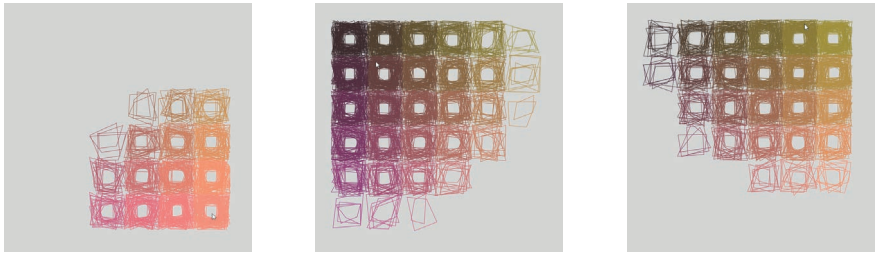
We implemented 3 major **interactive** and **animated** extensions to our algorithm **(please watch attached video!)**

1. Animation (shapes are shaking with frame rate of 10)



```
function draw() {
  frameRate(10);
  background(220);
  scale(1.5,1.5,0);
  DrawMesh (6,6);
}
```

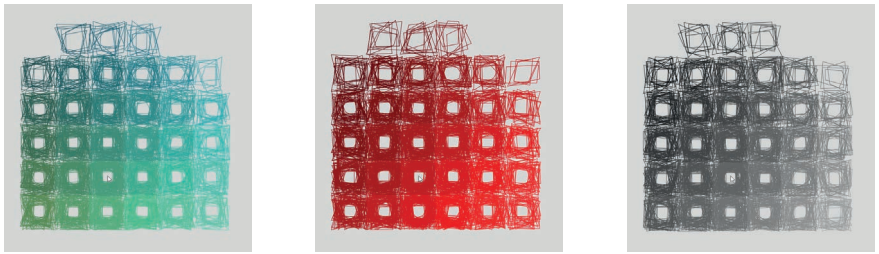
2. **Mouse tracking** (the density of line follows the cursor)



```
// Calculates the distance the mouse and the i,j cell's center
function DistIJ(i, j) {
  let X = i*90 + 90;
  let Y = j*90 + 90;
  return sqrt((X-mouseX)*(X-mouseX) + (Y-mouseY)*(Y-mouseY));
}
```

```
ijquad_amount=50-DistIJ(i,j)/8;
```

3. **Mouse click interaction** (the color changes when mouse is clicked)



```
// Updates s when clicked
function mouseClicked() {
  s++;
  if (s===8) {
    s=1;
  }
}
```

```
      if (s==1) {
         stroke(20*(i+j+1)+50,20*i+50,15*j+50);
      }
      if (s==2) {
         stroke(15*(j+1), 15*(i+j+1)+50, 20*(i+1)+60);
      }
      if (s==3) {
         stroke(0,0,0);
      }
      if (s==4) {
        stroke(20*(i+j+1)+50,0,0);
```

p5 - https://editor.p5js.org/kau.grv/sketches/tFSqdts4U

Conclusion

Thank you for reading our work!

You can visit our GitLab page, where you will find the codes, outputs and many different versions of the dynamic part :

GITLAB - https://gitlab.com/debeaunetristan/projet-recoding

Tristan Debeaune & Wendy Gervais