



**Revised and
Updated Edition**

Genera- tive Design

**Visualize,
Program, and
Create with
Javascript in p5.js**

**Benedikt Groß
Hartmut Bohnacker
Julia Laub
Claudius Lazzeroni**

**with contributions from
Joey Lee
Niels Poldervaart**

Revised and Updated Edition

Generative Design

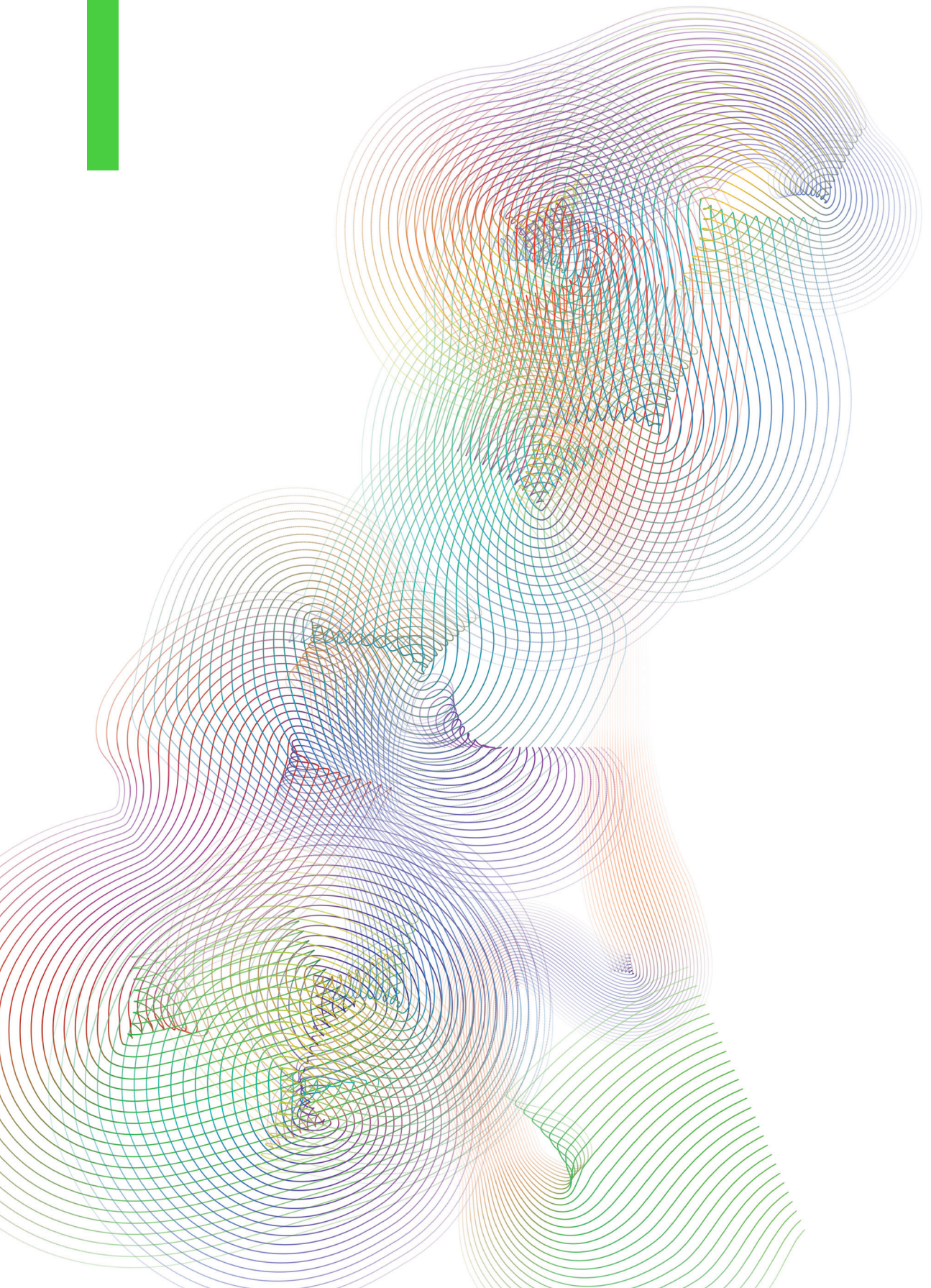
**Visualize, Program,
and Create
with JavaScript in p5.js**

**Benedikt Groß
Hartmut Bohnacker
Julia Laub
Claudius Lazzeroni**

**with contributions by
Joey Lee and
Niels Poldervaart**

Translated by Marie Frohling

Princeton Architectural Press, New York



Generative design has long ceased to be a trade secret among design students; in some universities, it is now firmly integrated into the curriculum. From infographics to the visualization of sound, from the fine arts to architecture, and especially in the realm of communication design and media installations, generative design allows for dynamic, stunning, and fascinating applications.

Processing and vvvv have for many years been the programming environments of choice for artists and designers. However, more recently there has been a shift toward more web-centric applications, giving rise to new coding environments such as p5.js, a JavaScript library that is especially programmed for and by artists, designers, and other web users.

The first edition of *Generative Design* was written almost a decade ago, and its acclaimed underlying teaching methods are still unrivaled. In this updated edition, the authors create an even more accessible and up-to-date entry point into creative coding with the use of JavaScript. In the spirit of the first edition of *Generative Design*, the goal is to remove any fear of programming and demonstrate how existing program snippets can be manipulated and tweaked to achieve amazing results almost at the click of a button.

Generative design fundamentally changes the design process: the designer shifts from being a performer of

tasks to being a conductor, effectively orchestrating the decision-making process of the computer. This is what generative design is all about: iteratively developing different processes and then selecting those that produce the most visually compelling results. Designers and artists no longer have to use the tools dictated by computers and powerful but prescriptive design software and can now create their own tools, which generate amazing results independently, as many of the examples in the book demonstrate.

In four simple lessons, Color, Shape, Type, and Image, users learn to influence their results and to improve them by either varying parameters—as explained in detail in each step—or by changing entire algorithms. The explanations are easy to understand, and their execution requires little or even no programming; with p5.js and its rapidly growing community, it is becoming easier to lay the groundwork for advanced technologies and trends, from 3D to augmented reality. The p5.js community is very active and constantly provides new updates and plug-ins for extending the functionality of p5.js. This book shows what can be done with this knowledge and how to dive deeper into generative design and its community. After experimenting with the sketches in the book and completing initial tasks with the online p5.js editor, users can venture forth independently and explore and expand on the creative output of the p5.js community and beyond.

With the success of *Generative Design*, which has been translated into several languages, the authors realized that the key to teaching artists and designers how to code was to empower them through simple but satisfying incremental successes. Students could then build increasing complexity into their work based on these basic principles.

The book is supplemented by a website where users can download all of the programs (sketches) for free and start experimenting immediately. After completing the four tutorials, you will be able to visualize data, create infographics, visualize text analyses, and much more.

Have fun with creative coding.

Karin and Bertram Schmidt-Friderichs

I	Introduction	3–41
I.0	Preface	3
I.1	Contents	6
I.2	Website	8
I.3	Projects	10
P	Basic Principles	42–225
P.0	Introduction to p5.js	42
	P.0.0 p5.js, JavaScript, and Processing	44
	P.0.1 The development environment	46
	P.0.2 Language elements	48
	P.0.3 Programming beautifully	56
P.1	Color	58
	P.1.0 Hello, color	60
	P.1.1 Color spectrum	62
	P.1.1.1 Color spectrum in a grid	62
	P.1.1.2 Color spectrum in a circle	64
	P.1.2 Color palettes	66
	P.1.2.1 Color palettes through interpolation	66
	P.1.2.2 Color palettes from images	68
	P.1.2.3 Color palettes from rules	72
P.2	Shape	78
	P.2.0 Hello, shape	80
	P.2.1 Grid	82
	P.2.1.1 Alignment in a grid	82
	P.2.1.2 Movement in a grid	86
	P.2.1.3 Complex modules in a grid	90
	P.2.1.4 Checkboxes in a grid	94
	P.2.1.5 From grid to moiré	98
	P.2.2 Agents	102
	P.2.2.1 Dumb agents	102
	P.2.2.2 Intelligent agents	104
	P.2.2.3 Shapes from agents	108
	P.2.2.4 Growth structure from agents	112
	P.2.2.5 Structural density from agents	116
	P.2.2.6 Agents on a pendulum	120
	P.2.3 Drawing	126
	P.2.3.1 Drawing with animated brushes	126
	P.2.3.2 Relation and distance in drawing	130
	P.2.3.3 Drawing with type	132
	P.2.3.4 Drawing with dynamic brushes	134
	P.2.3.5 Drawing with the pen tablet	138
	P.2.3.6 Drawing with complex modules	142
	P.2.3.7 Drawing with multiple brushes	146

P.3	Type	150
P.3.0	Hello, type	152
P.3.1	Text	154
P.3.1.1	Writing time-based text	154
P.3.1.2	Text as blueprint	156
P.3.1.3	Text image	160
P.3.1.4	Text diagram	166
P.3.2	Font outline	170
P.3.2.1	Dissolving the font outline	170
P.3.2.2	Varying the font outline	174
P.3.2.3	Font outline from agents	178
P.3.2.4	Parallel font outlines	180
P.3.2.5	Kinetic font	184
P.4	Image	188
P.4.0	Hello, image	190
P.4.1	Image cutouts	192
P.4.1.1	Image cutouts in a grid	192
P.4.1.2	Feedback of image cutouts	196
P.4.2	Image collection	198
P.4.2.1	Collage from image collection	198
P.4.2.2	Time-based image collection	202
P.4.3	Pixel values	204
P.4.3.1	Graphics from pixel values	204
P.4.3.2	Type from pixel values	210
P.4.3.3	Real-time pixel values	214
P.4.3.4	Emojis from pixel values	220

A	Appendix	226-256
----------	-----------------	----------------

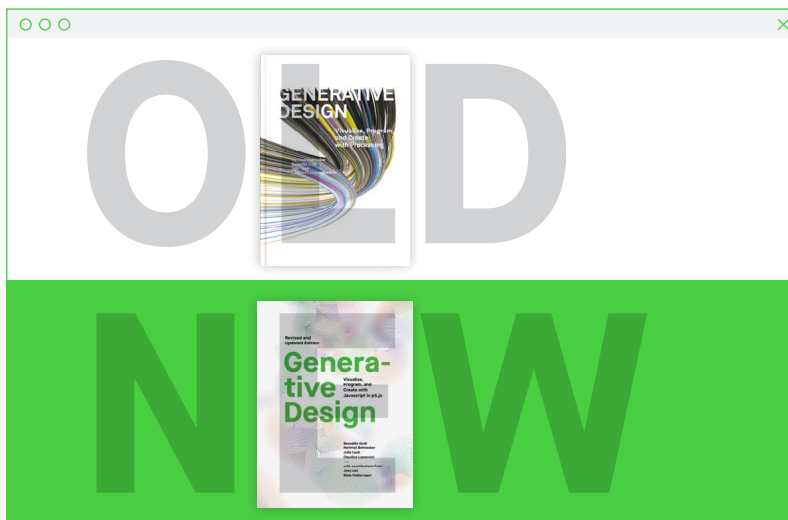
A.1	Looking ahead	228
A.2	Reflection	244
A.3	Bibliography	250
A.4	The authors	252
A.5	We thank	253
A.6	Copyright	254
A.7	Farewell	256

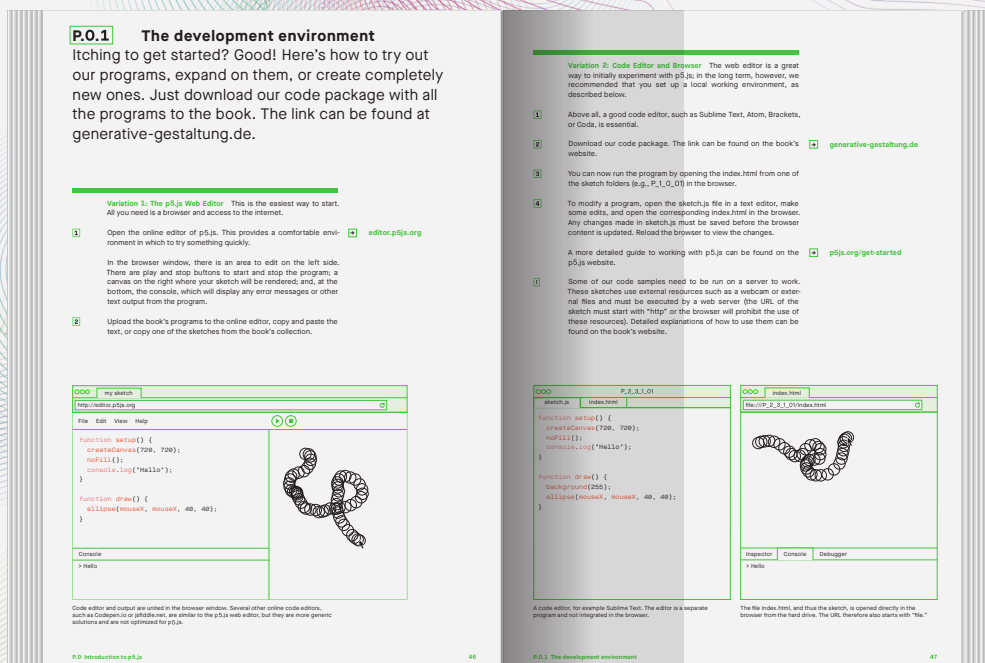
I.2

www.generative-gestaltung.de

Generative Design: Visualize, Program, and Create with Javascript in p5.js is a tried and tested tutorial that allows people to design with p5.js. It is not necessary to type in any code: all the programs in the book, called “sketches,” can be downloaded for free from the book’s website for experimentation. This symbol → indicates the name of a sketch in the download package.

The code summary page shows the main features of the code and how it affects the program output. The book explains how the parameters of the code impact the outcome and how users can interact with the sketch to develop their own visual solutions.



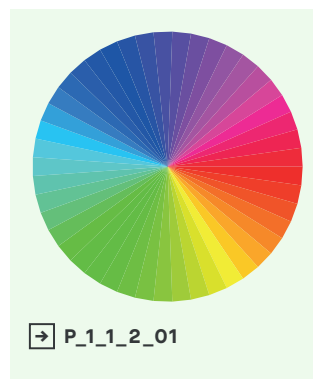
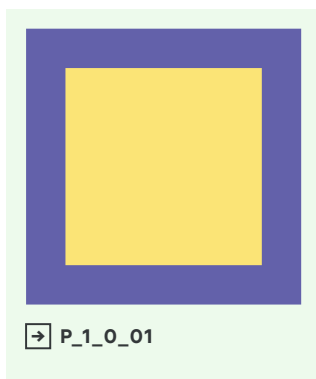


Hello and welcome to Generative Design. Here, you will find all of the sketches from the book and their associated code. Run the sketches directly in the browser with the p5.js-web-editor or locally on your machine by downloading the code package below.

 **Download Code Package**

Library

P.1. Color



I.3

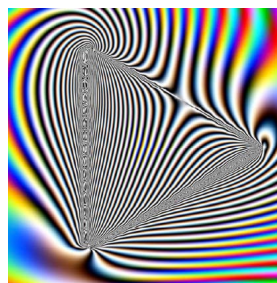
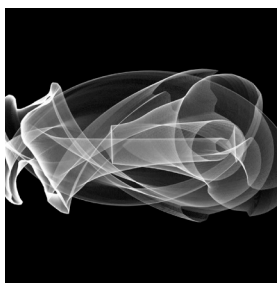
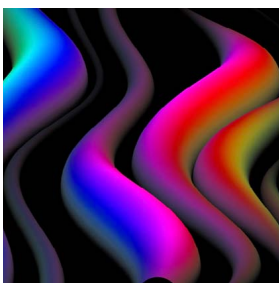
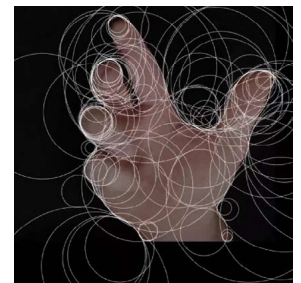
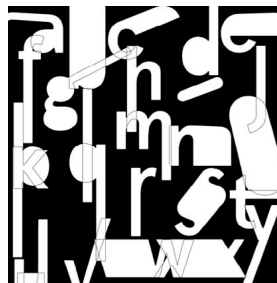
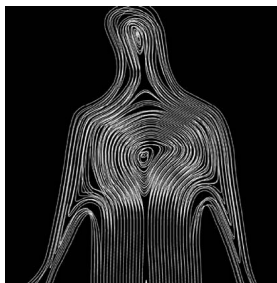
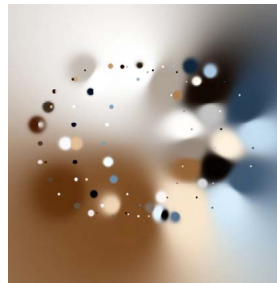
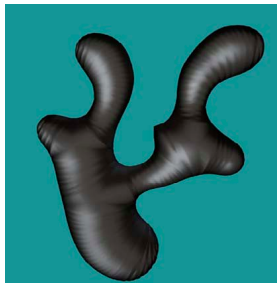
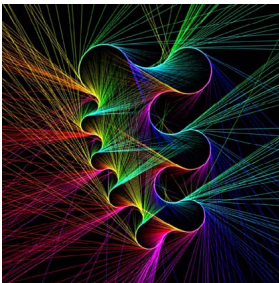
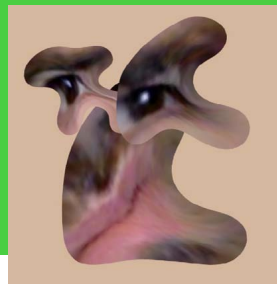
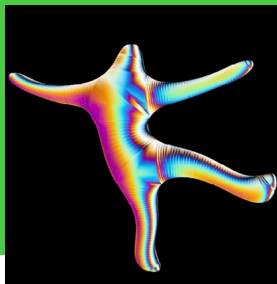
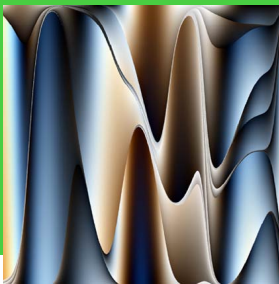
Projects

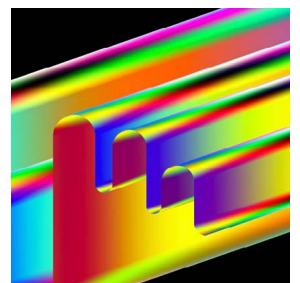
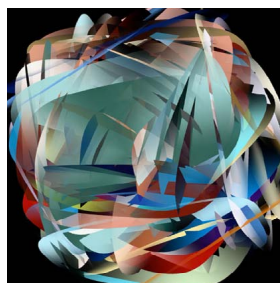
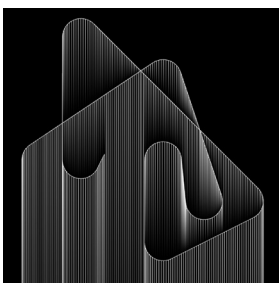
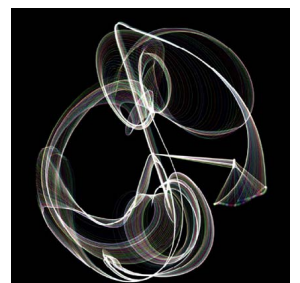
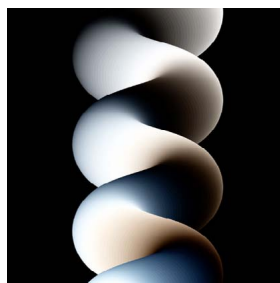
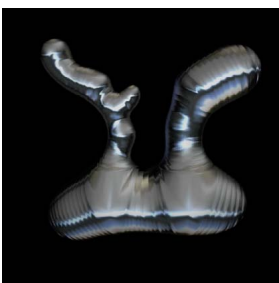
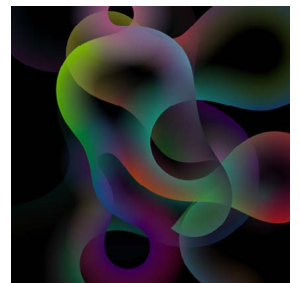
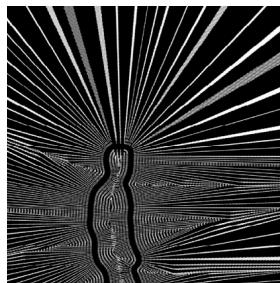
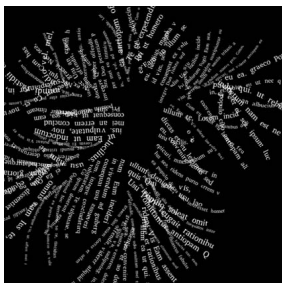
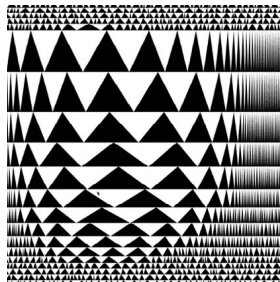
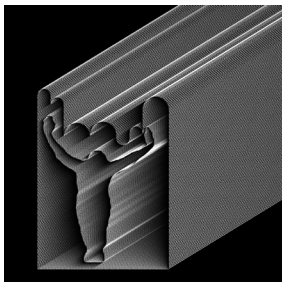
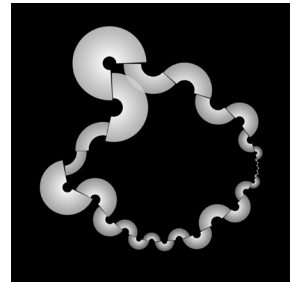
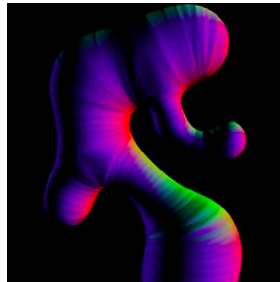
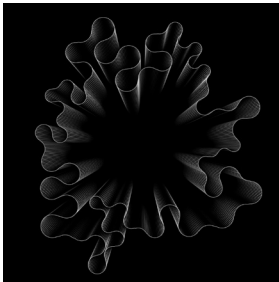
These thirteen works by various media artists, designers, and architects active in the field of generative design are intended to serve as a representative overview of the topic and a source of inspiration.

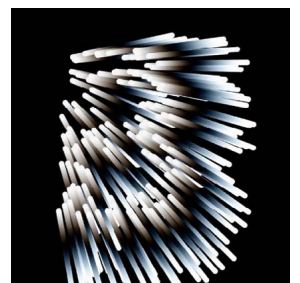
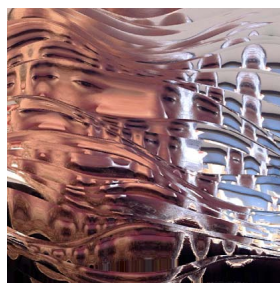
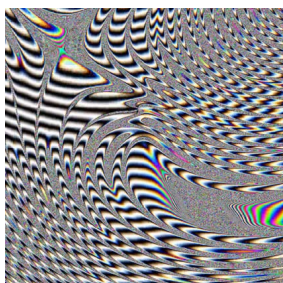
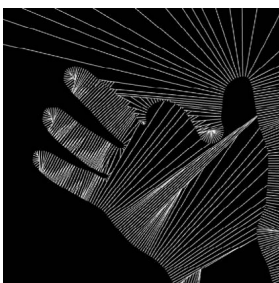
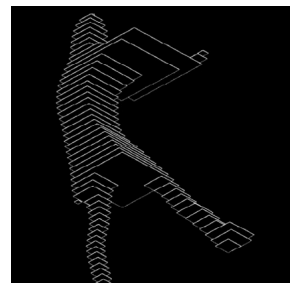
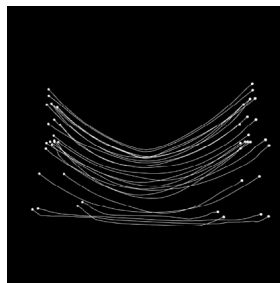
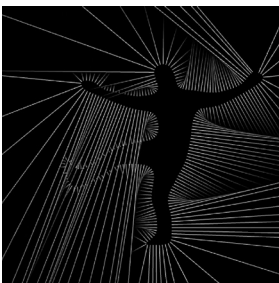
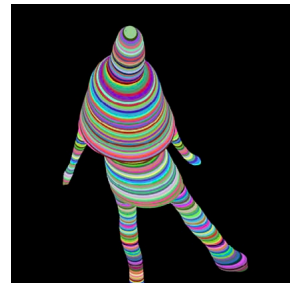
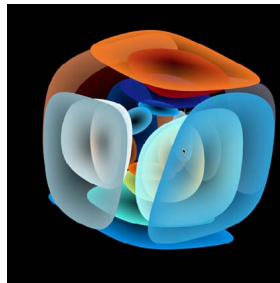
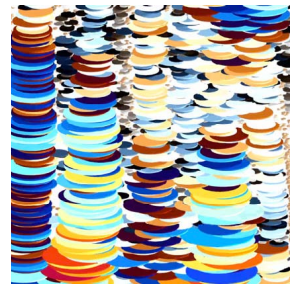
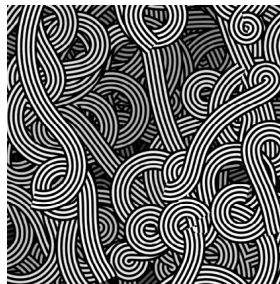
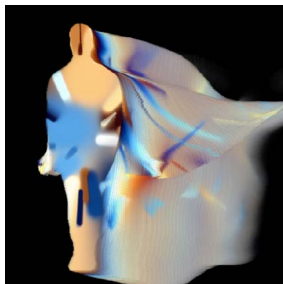
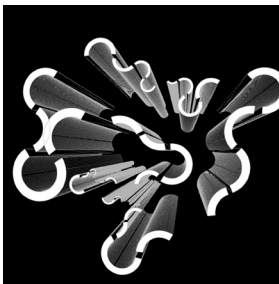
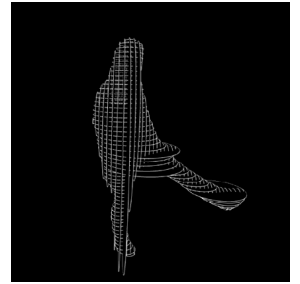
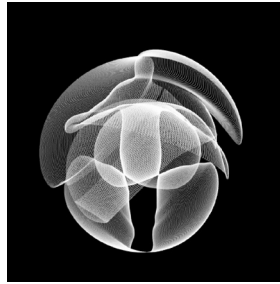
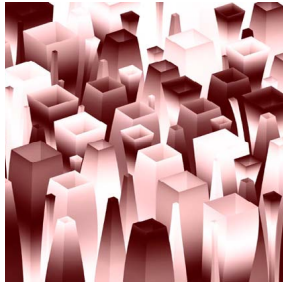
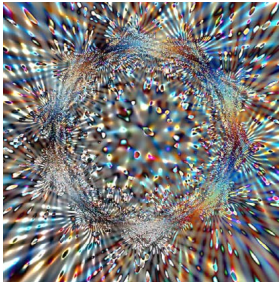
01

2016/17 Daily Sketches Zach Lieberman

Daily Sketches is a series of short generative animations shared daily with the world by Zach Lieberman on social media for fast feedback. The sketches show the process of exploring new visual concepts using geometry, animation, gesture, form, and code. Lieberman describes his sketches of the day as follows: "A lot of times, as artists, we feel like we're struggling to find our frequencies and what resonates with the frequencies of the world. This act of sketching is a kind of tuning of these frequencies."



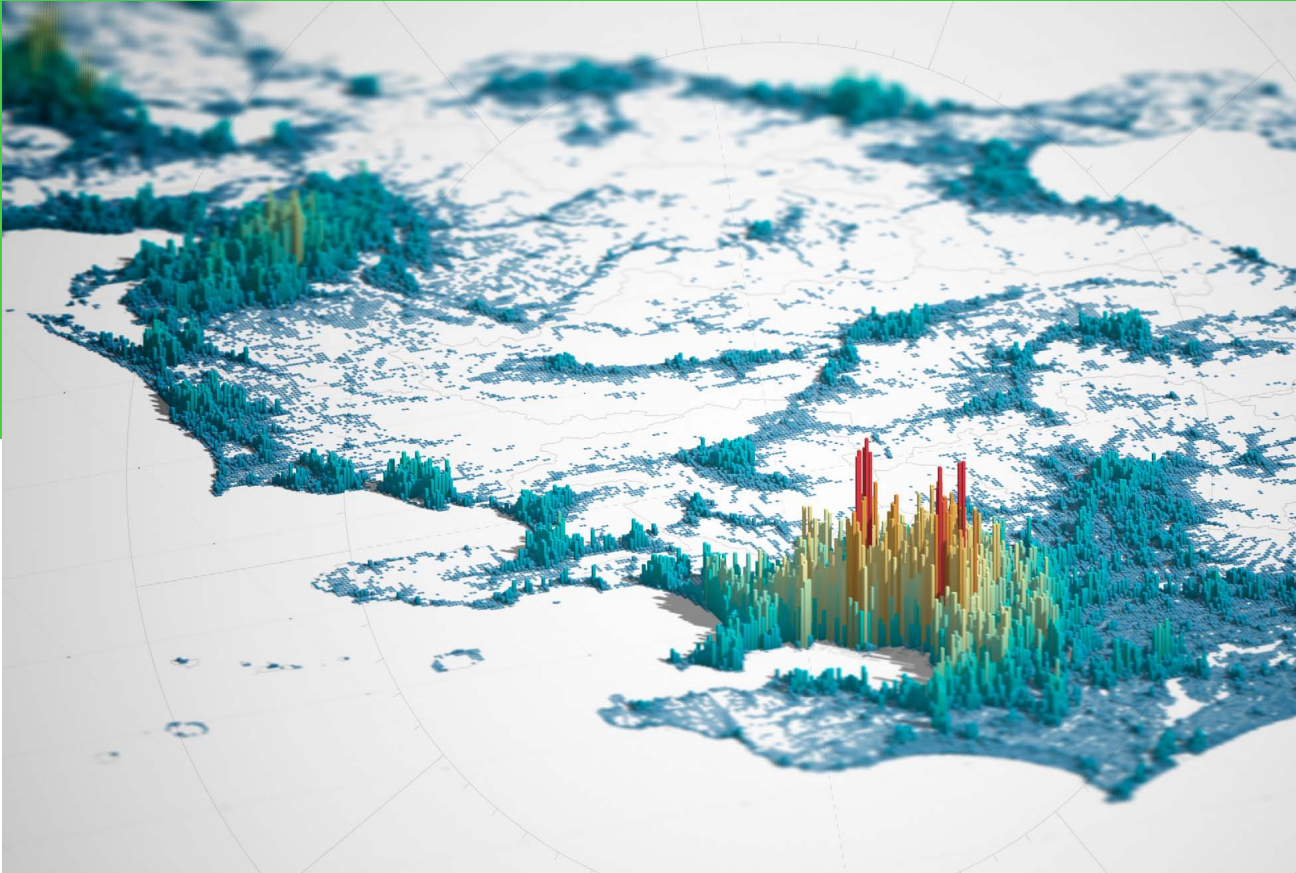


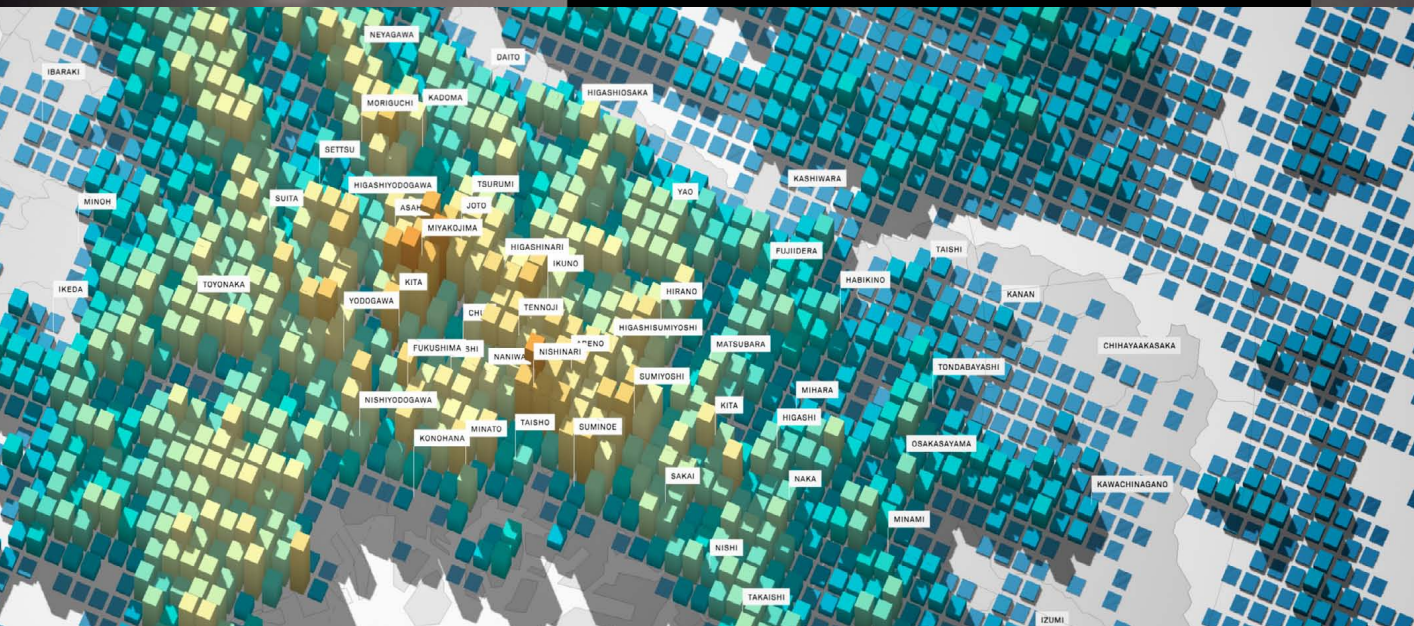


Design and Development
Shota Matsuda (Takram)

Photo Credits
Koki Nagahama

Planck is a web browser-based framework developed by Takram for the visualization of large geographical data sets. The framework is designed to achieve both analytical and immersive visual experience by using various techniques, such as parallel projection and depth of field. Three visualizations were created for Media Ambition Tokyo 2017, presenting data on Japan's estimated population in 2050, the languages people have tweeted in, and world air traffic.





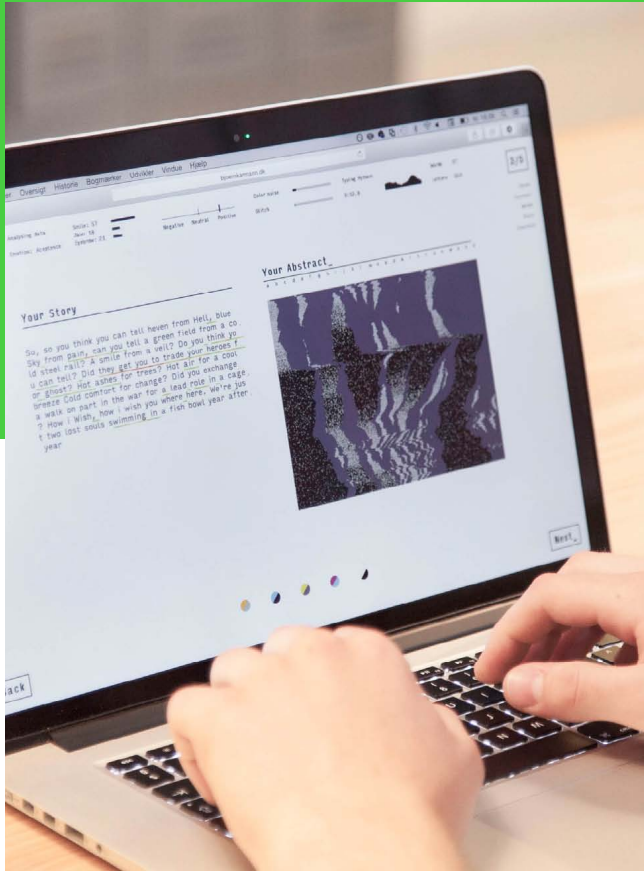
Interaction Design
Bjørn Karmann

Fashion Design
Julie Helles

Textile Design
Kristine Boesen

**Bachelor's Degree
Graduation Project**
Kolding Design School, DK

Fashion has always been a means of self-expression, but **Abstract_** takes individuality a step further by enabling a customer to write themselves into a piece of clothing. Abstract_'s interactive platform prompts the customer to write a personal story and uses the computer's camera to capture facial expressions. Data collected from the story, the rhythm of the keystrokes, and the customer's expression are then transformed into a visual representation and mapped onto a textile for clothing.



Bjørn Karmann

Julie Helles

Kristine Boesen

04

2016

Rottlace – Björk

MIT Media Matter Group

Christoph Bader

Dominik Kolb

Prof. Neri Oxman

Stratasys Ltd.

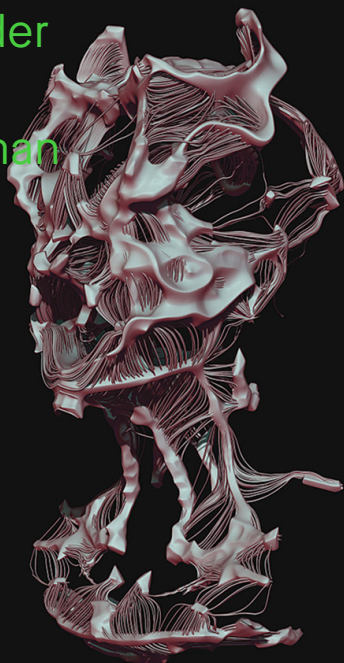
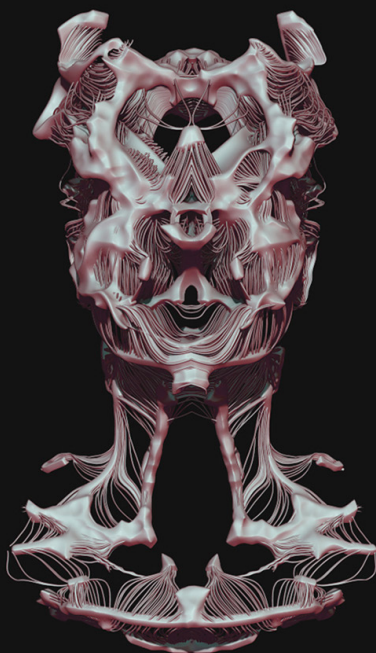


Photo Credits
Santiago Felipe

Rottlace is part of a family of masks designed for the Icelandic singer-songwriter Björk. The design is informed by the geometric and material logic that underlies the human musculoskeletal system. The masks can be understood as “muscle textile”: bundled, continuous multimaterial structures providing formal and structural integrity as well as movement to the face and neck, resulting in an object that is designed as a synthetic whole without parts.

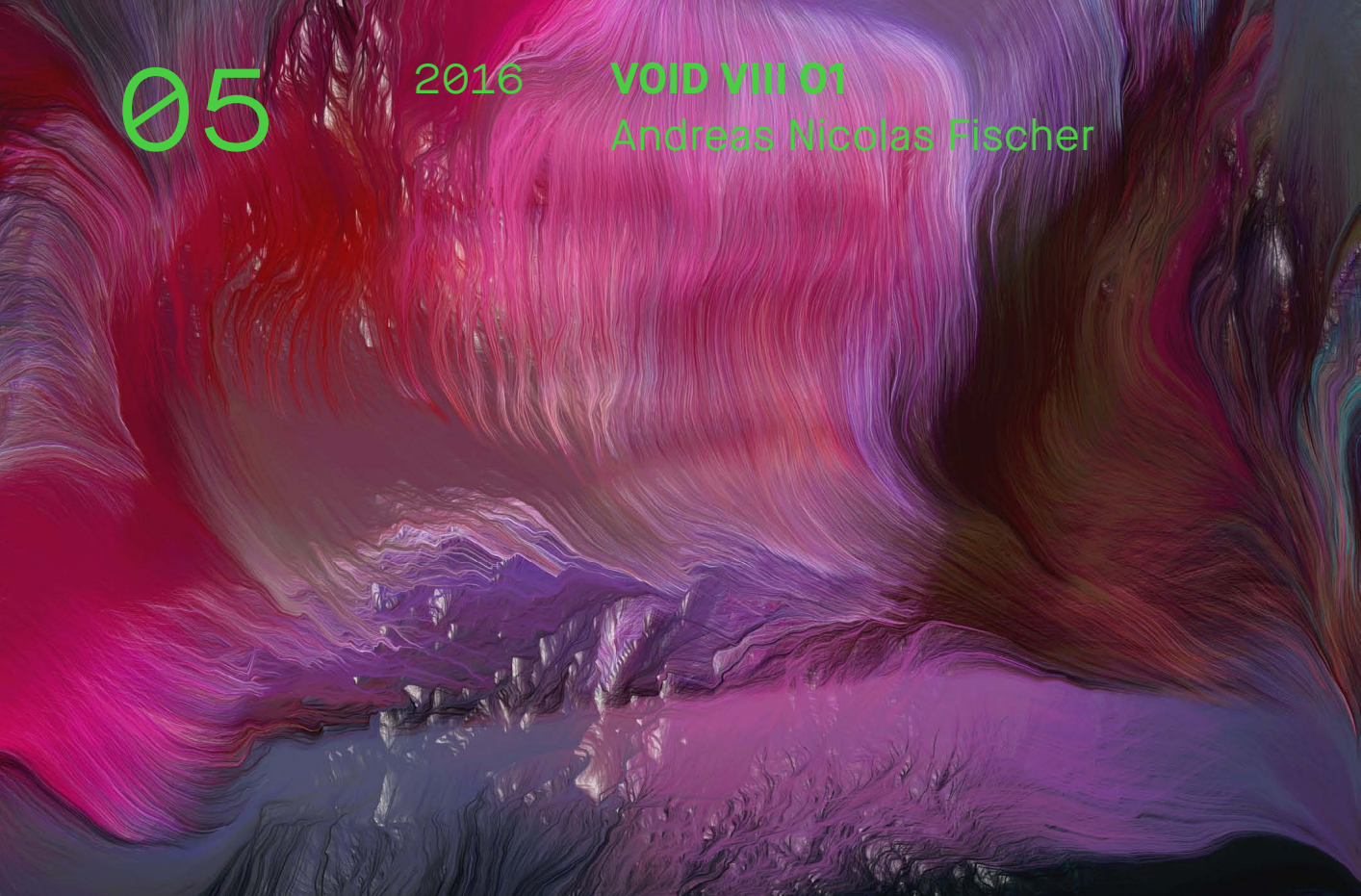


05

2016

VOID VIII 01

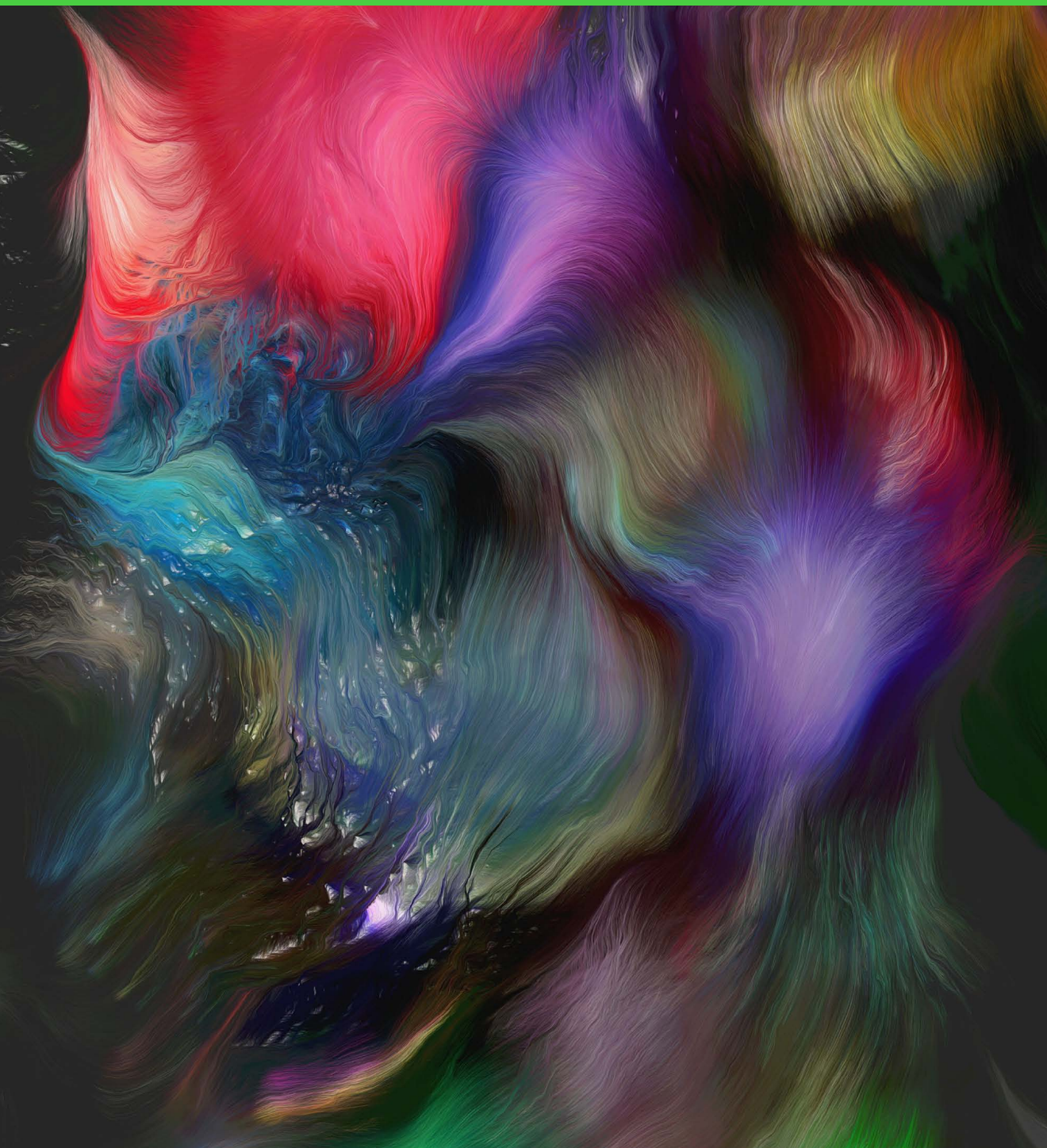
Andreas Nicolas Fischer



Primary Programming
Andreas Nicolas Fischer
and Benjamin Maus

Additional Programming
Abraham Pazos Solatie

VOID is a series of images created with custom generative software. A swarm of particles governed by an algorithm moves over a surface, leaving behind colorful traces. Over time, this results in an abstract composition that develops in unpredictable ways.



Commissioned by and in
collaboration with Monotype

Monotype: Type Reinvented is a series of three typographic installations that reflect how type can become “smart, dynamic, and emotional” through new digital approaches using interactivity, generative design, and data input. Popular Monotype typefaces are recontextualized and reimagined in new spaces and materials.



06

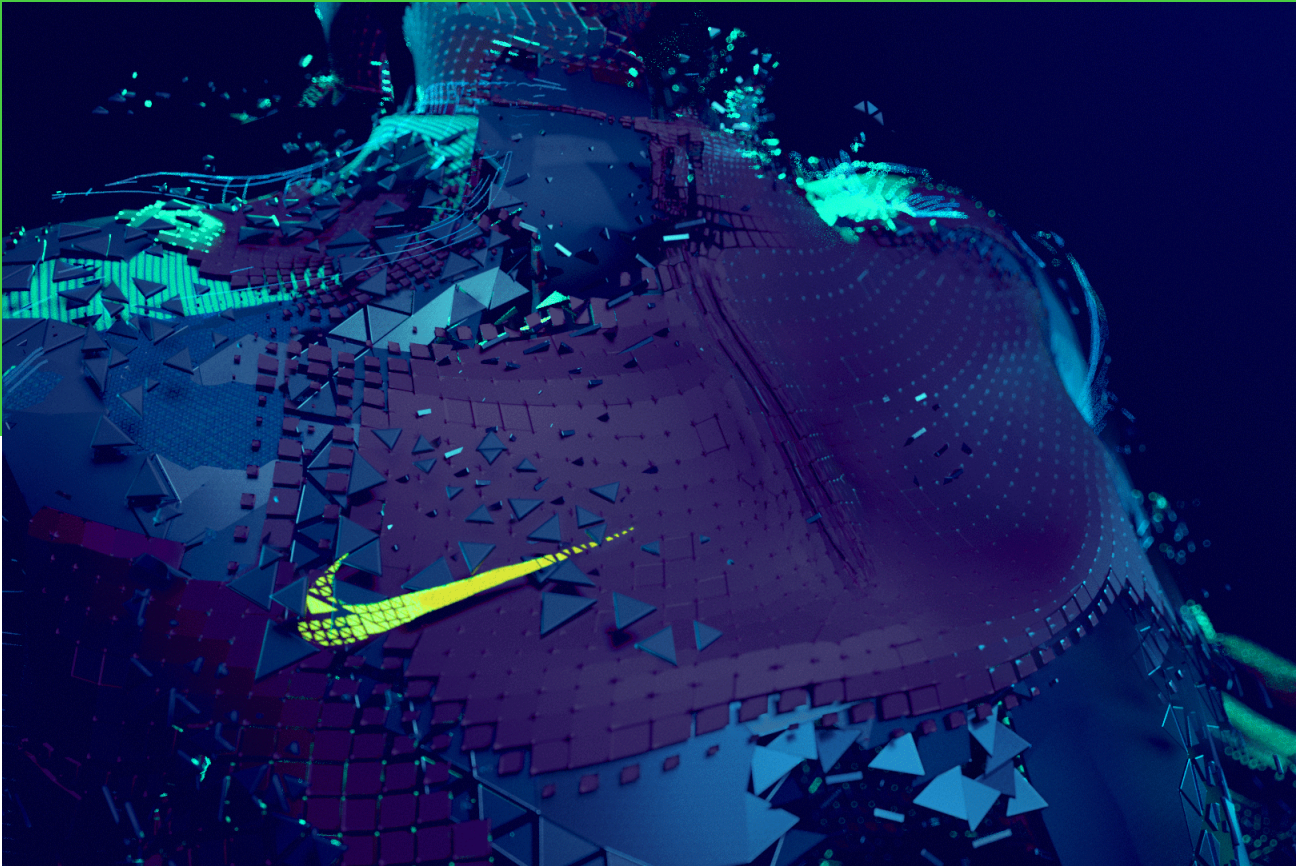
2017

Monotype:
Type Reinvented
Field



Client
Nike Global Football

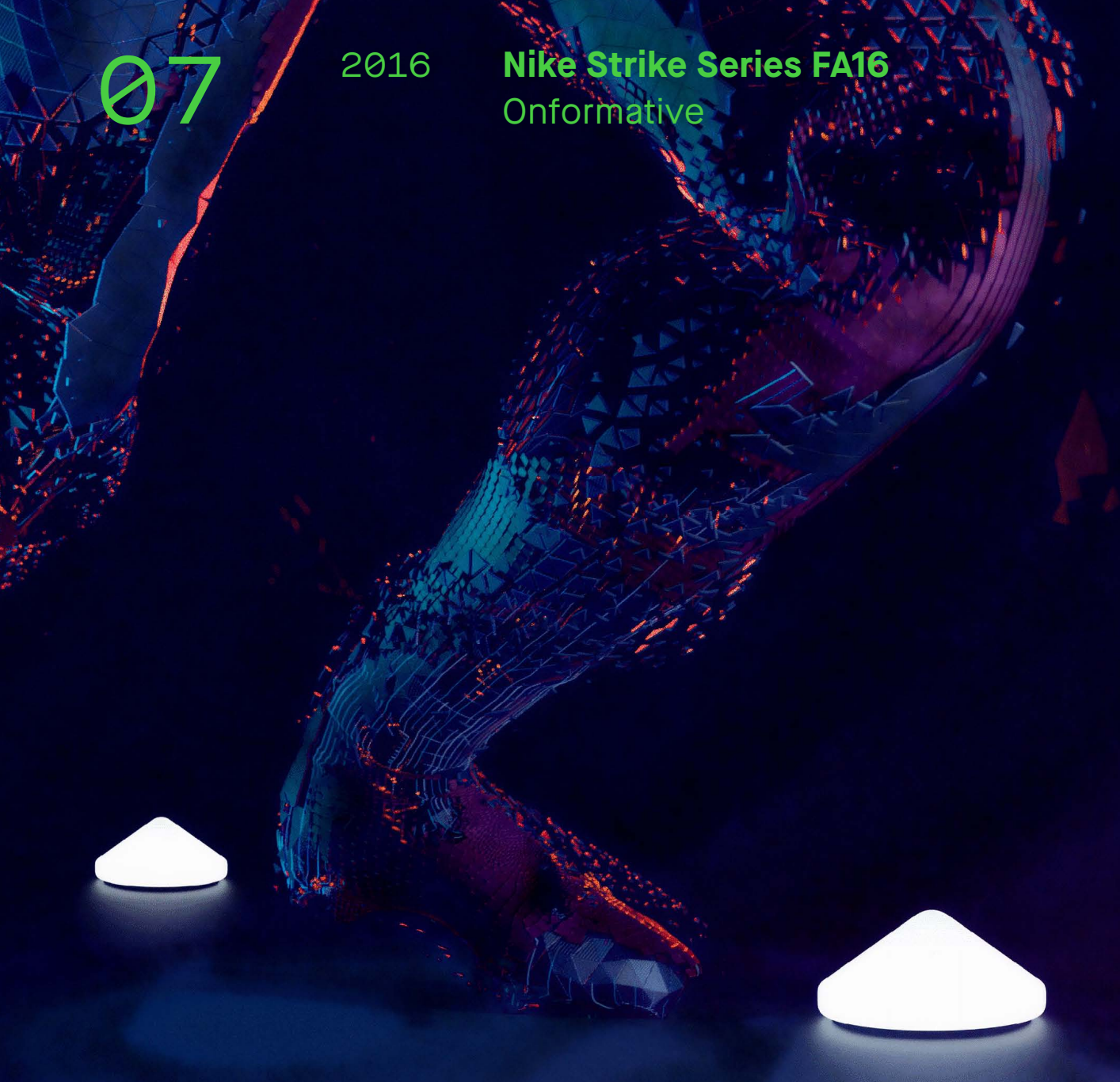
Nike Strike Series FA16 is a series of short films and still imagery composed of inspiring 3D renderings that exemplify the power and precision of world-class athletes. Full-body scans were taken of professional football players to create specialized 3D models that manifest the true essence of these athletes. The data of their distinctive characteristics, such as speed, velocity, and physical skills, defined the visual execution of the players.



07

2016

Nike Strike Series FA16
Onformative



UCL Design Computational Lab

Design

Manuel Jimenez García and
Gilles Retsin

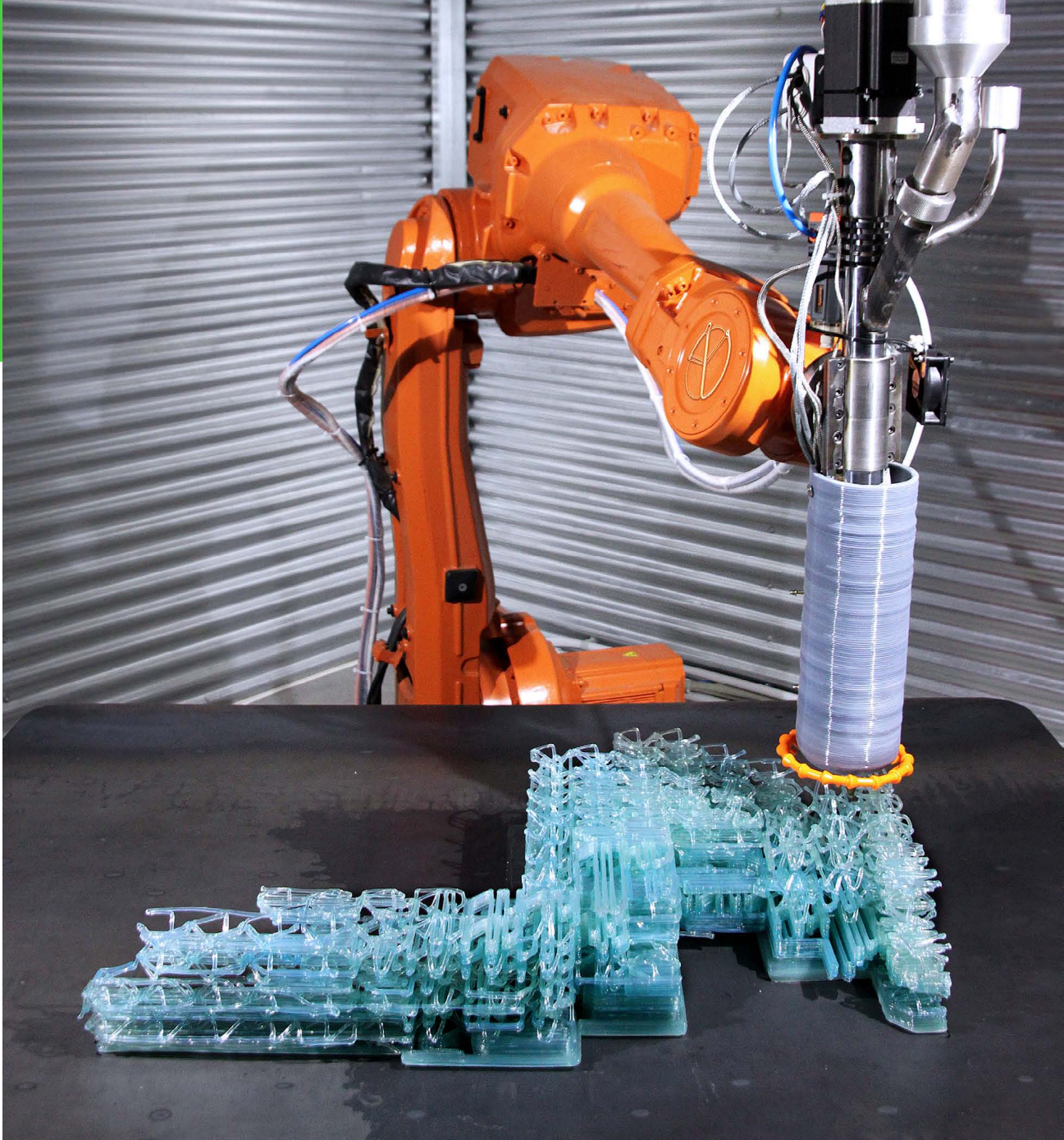
Fabrication Support

Nagami Design, Vicente Soler

Team

Manuel Jimenez García,
Miguel Angel Jimenez García,
Ignacio Viguera Ochoa,
Gilles Retsin, Vicente Soler

VoxelChair is the first prototype resulting from new and ongoing software innovation and development for robotic 3D printing. The software behind the VoxelChair draws on methodologies ranging from computational architecture to medical imaging to create new opportunities for designing and printing 3D compositions and structures. The creators suggest that "instead of designing the form of the chair, designers should design the behaviors and properties of the material directly."



08

2016

VoxelChair v.1.0

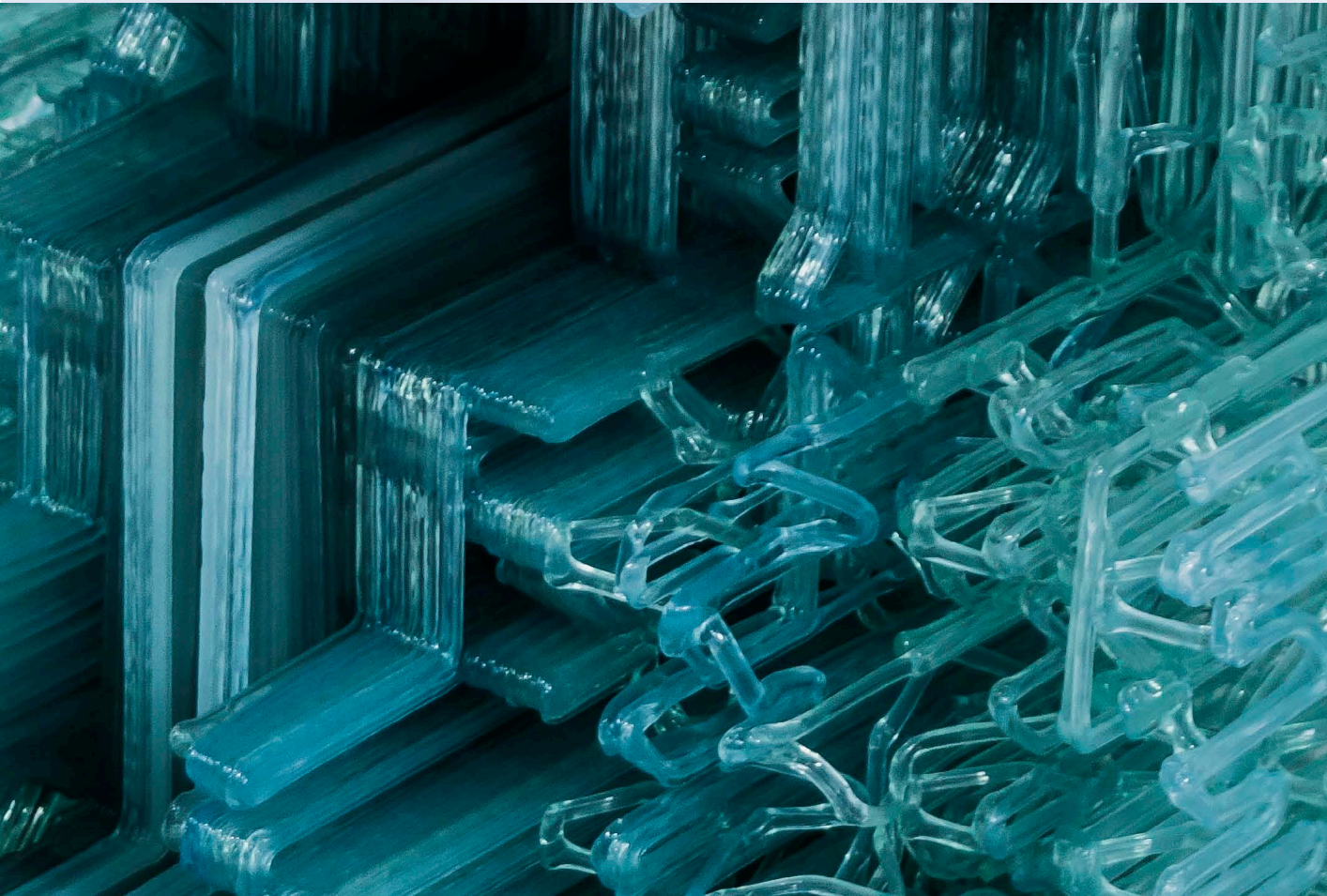
Manuel Jimenez García

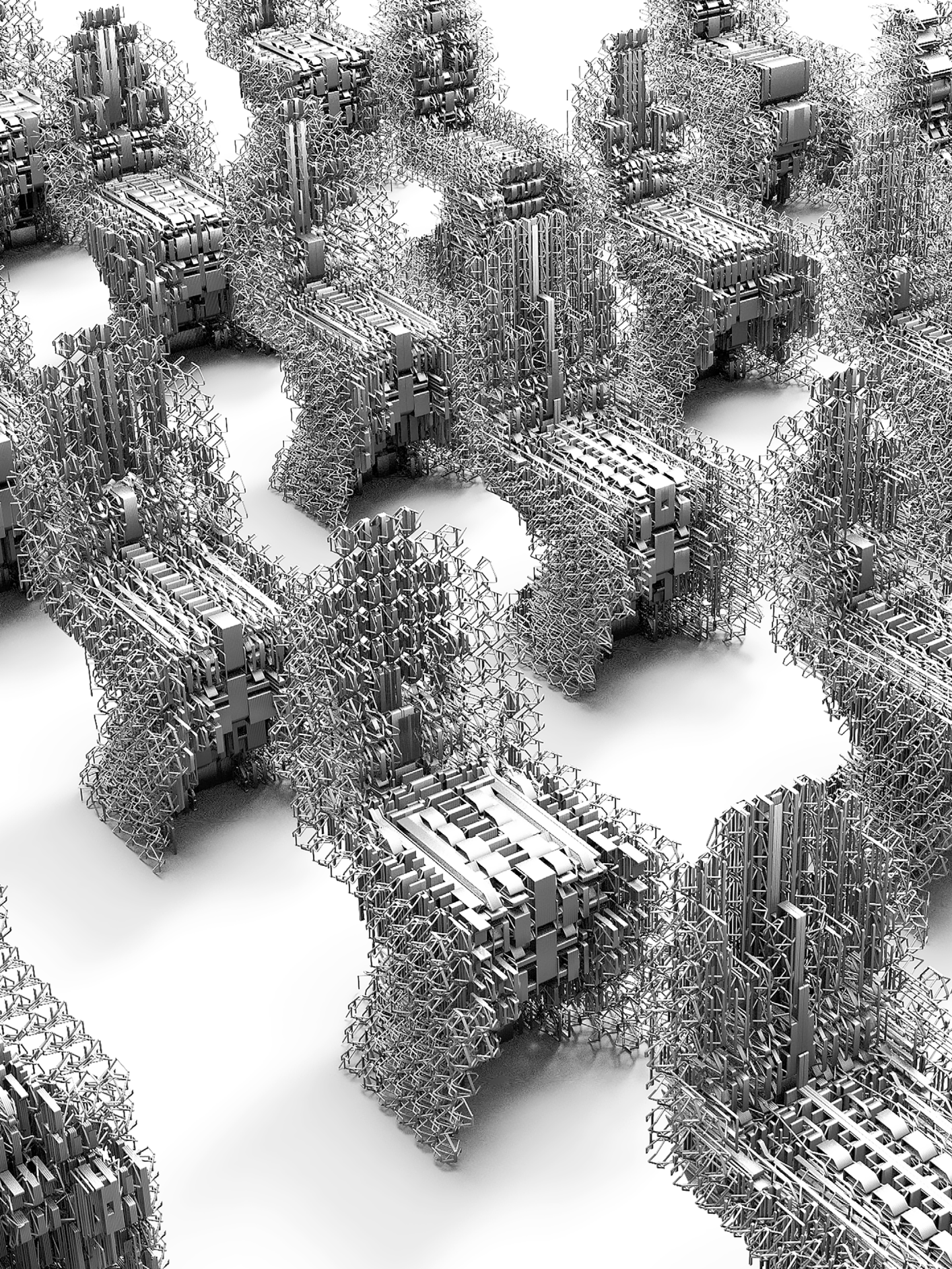
Gilles Retsin

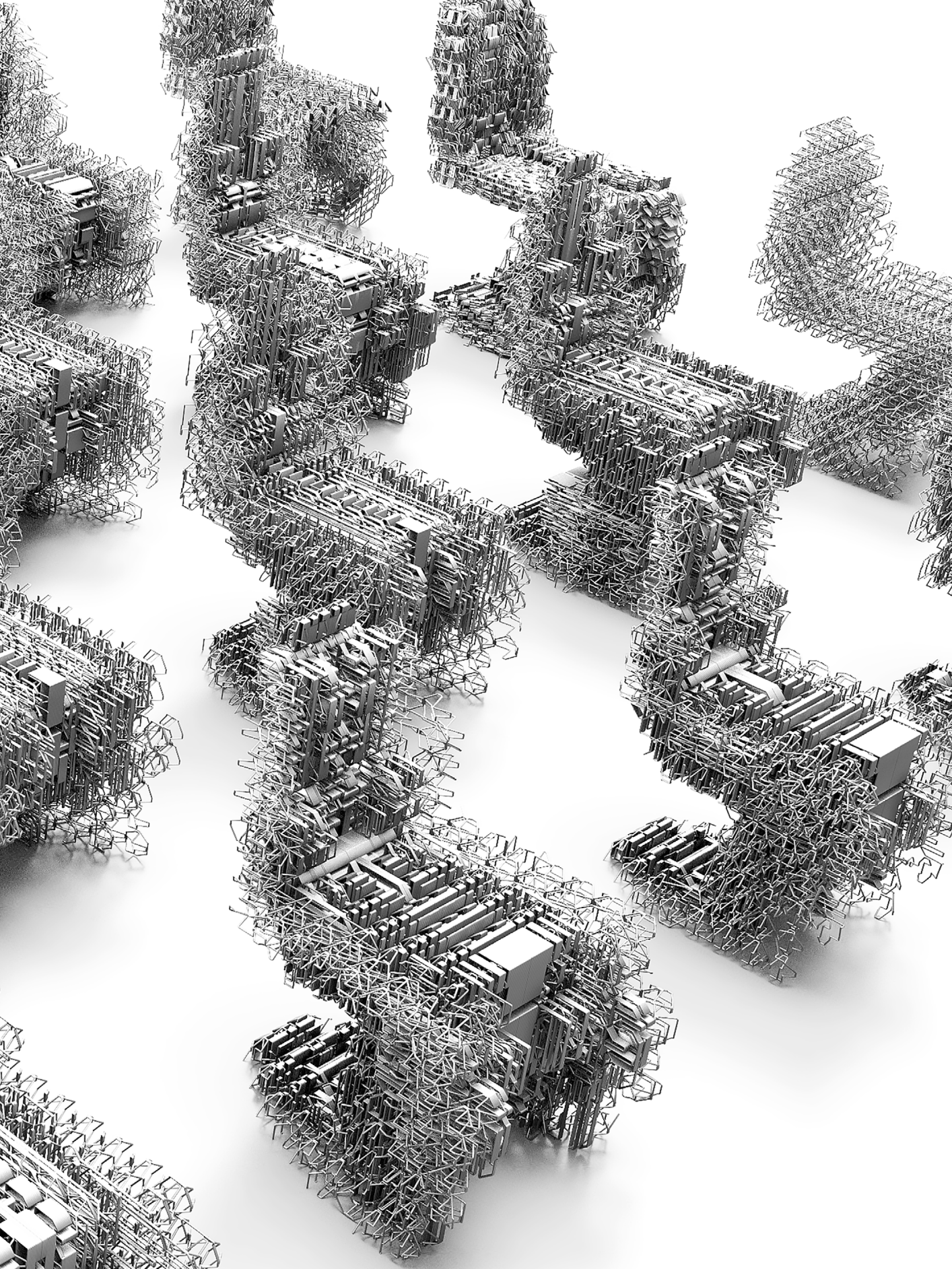
Nagami Design

Vicente Soler

UCL Design Computational Lab

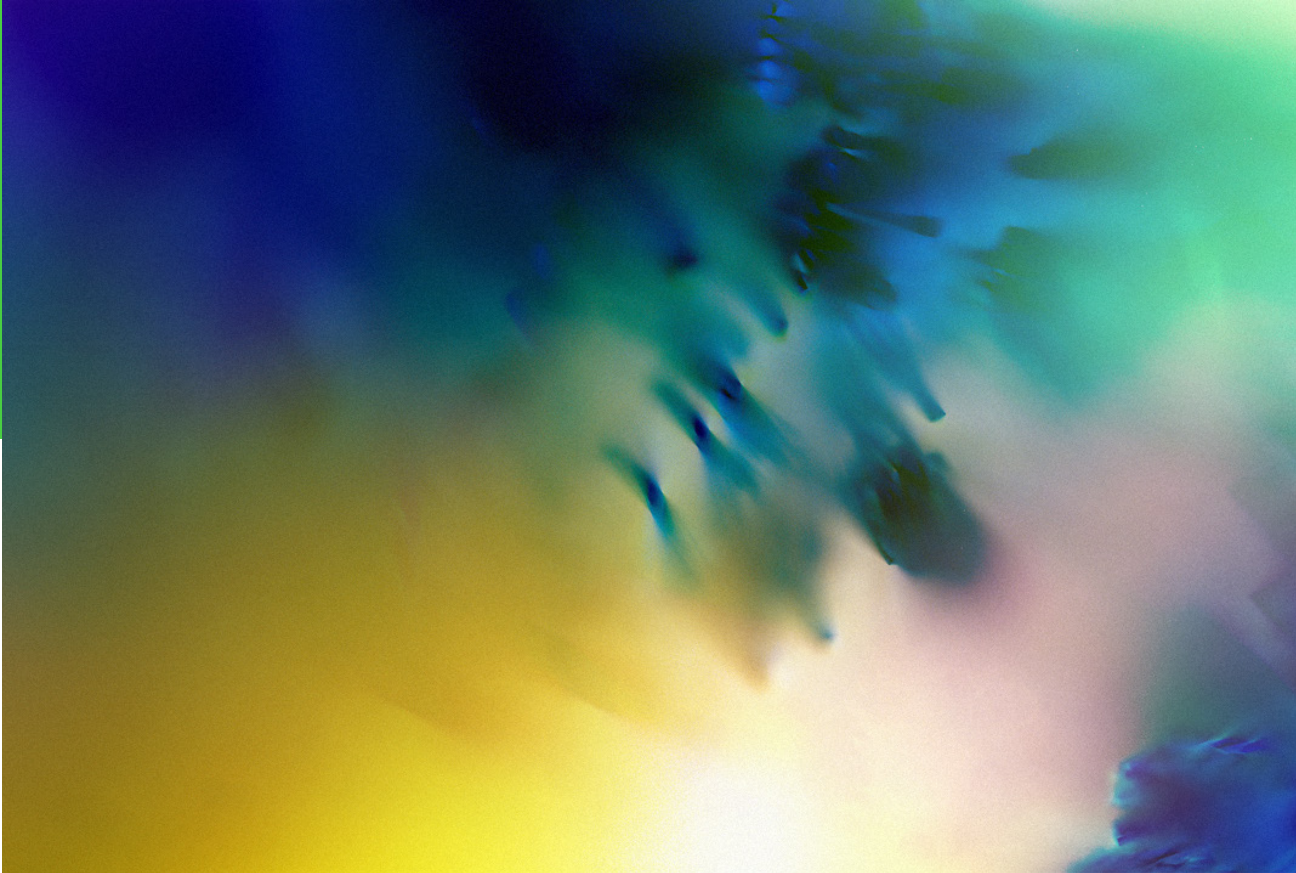






Client
Dolby Laboratories

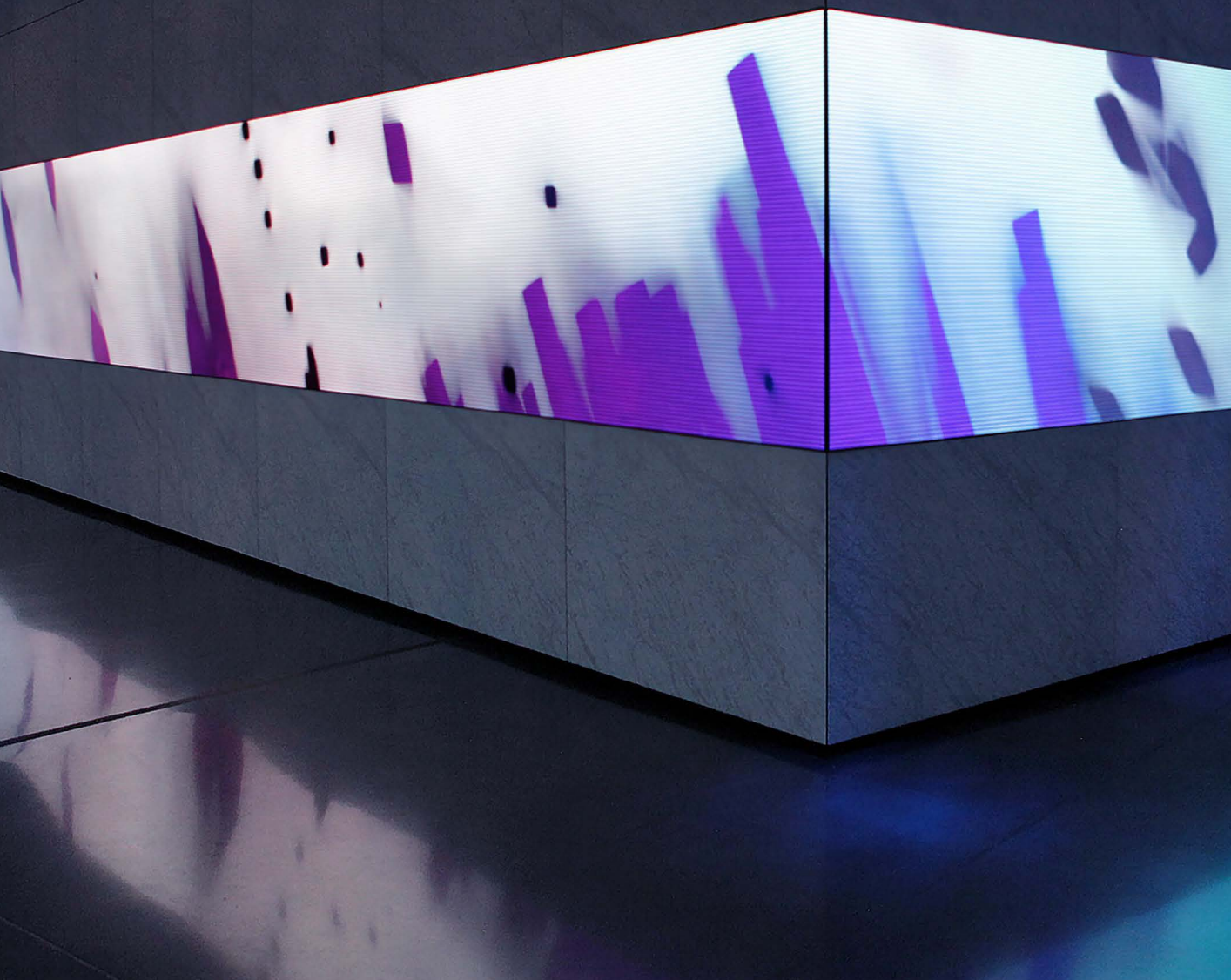
Collide is an art installation that transforms recorded motion data into abstract graphics and sound. Surreal imagery and an engaging soundscape create an immersive space that captures the essence of motion, color, and sound to represent the experience of “letting go.” Inspired by the phenomenon of synesthesia, the union of the senses, Collide creates a new language by combining original chamber music and painterly visuals.



09

2016

Collide:
synaesthetic art installation
Onformative



10

2017

Block Bills

Matthias Dörfelt



1 4 8 9 1 4 3 2 0 6

Archival digital print on paper, 3.3 x 5.9 in.

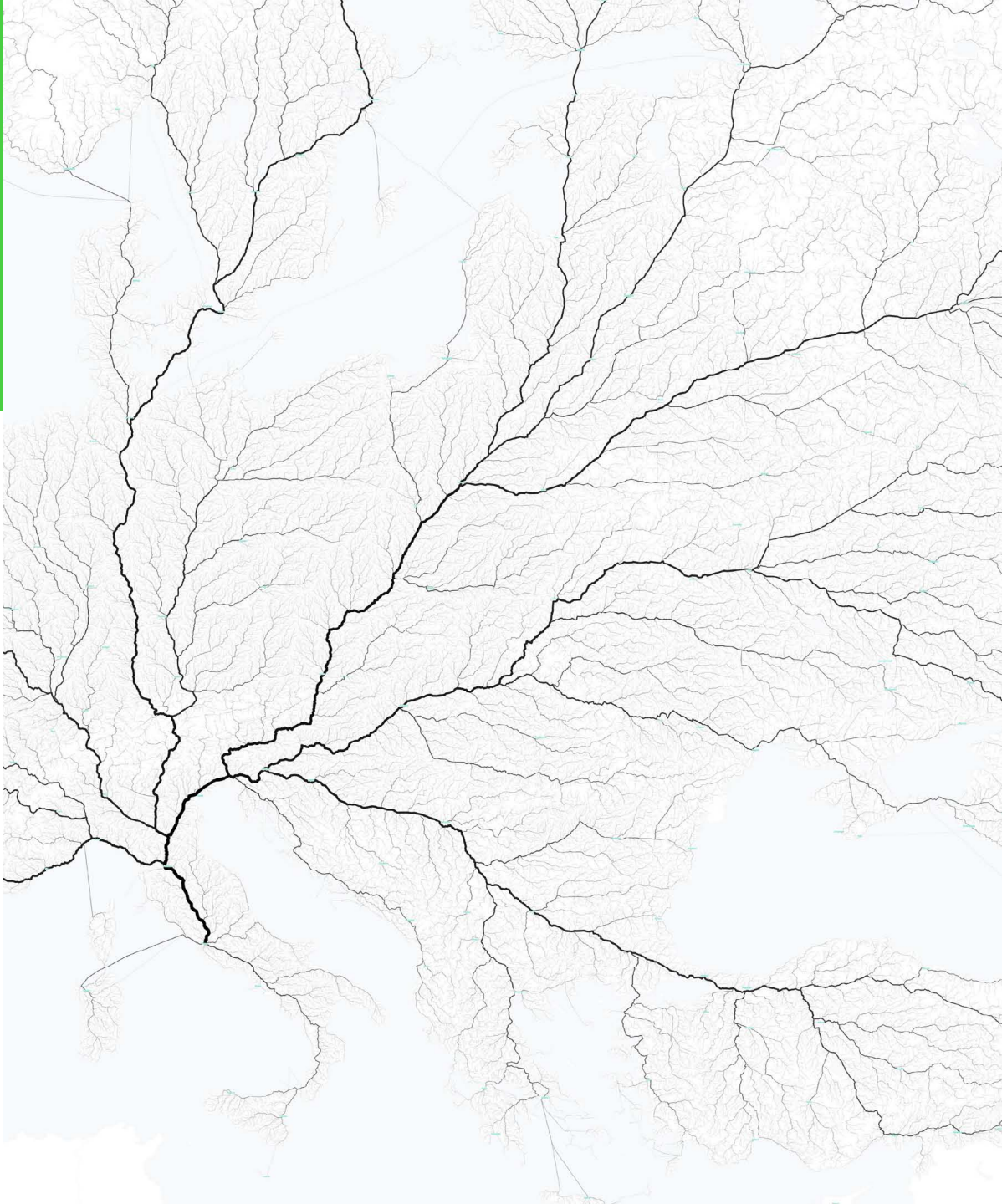
Block Bills is a series of sixty-four generated banknotes representing blocks in a Bitcoin blockchain. A visual system defined by the artist Matthias Dörfelt uses both randomness and the unique aspects of each block in the selected blockchain to affect every aspect of each bill in the series. The bills are allusions to Dörfelt's ambivalence toward Bitcoin and cryptocurrencies; they bring these otherwise intangible and invisible constructions into a visually and emotionally tangible context.



Map and Road Network Data
OpenStreetMap

Routing Engine
GraphHopper

Roads to Rome is a data visualization project that explores the idiom "All roads lead to Rome." The outcome is both information visualization and data art and unveils mobility patterns at a very large scale. The visualizations were created using routing algorithms on existing street infrastructure (OpenStreetMap) from the city to continent scale. In addition to their aesthetic quality, the resulting images bring insights into the ways in which road infrastructures reflect regional, political, and geographical situations.



11

2015

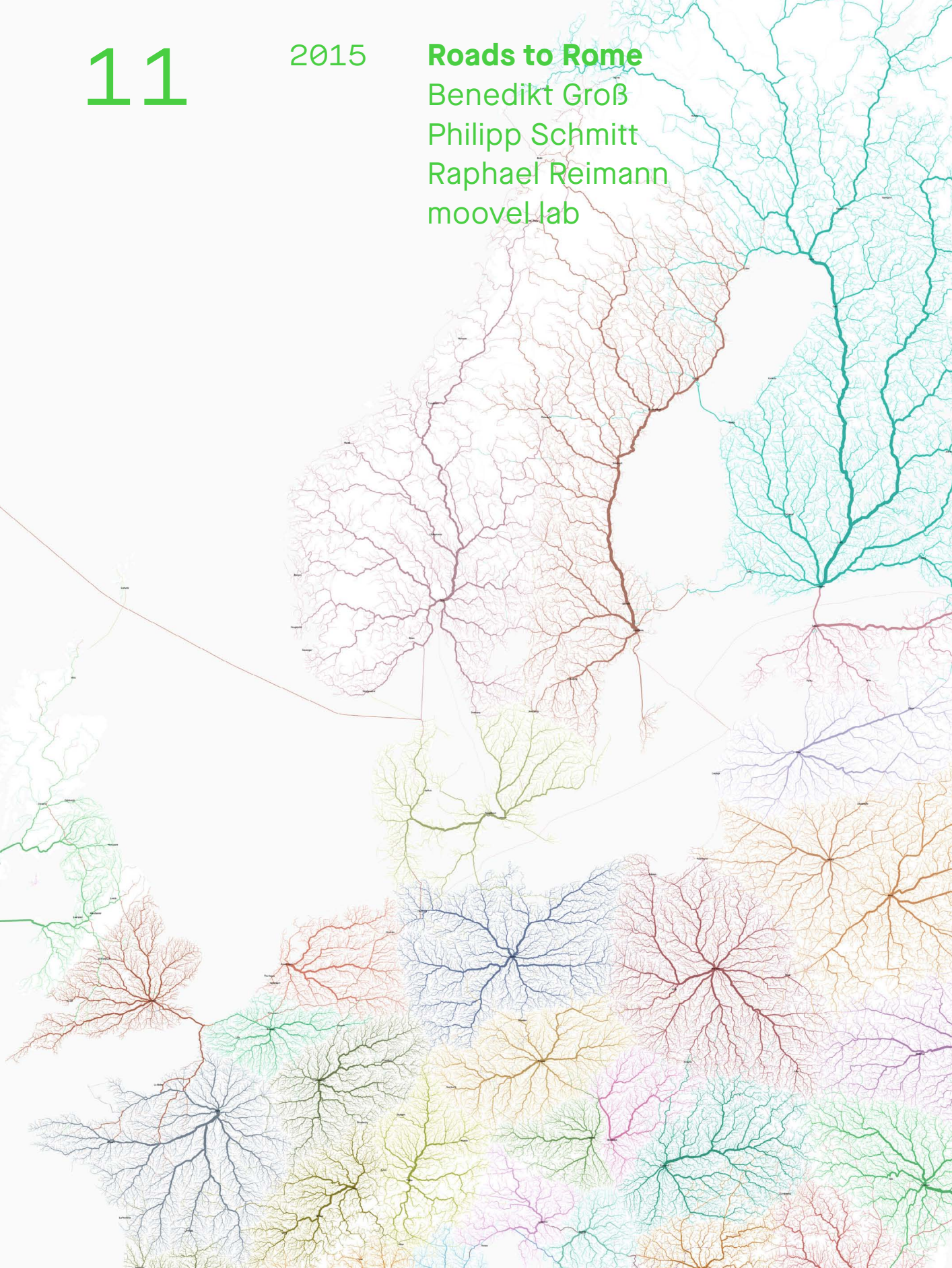
Roads to Rome

Benedikt Groß

Philipp Schmitt

Raphael Reimann

moovel lab



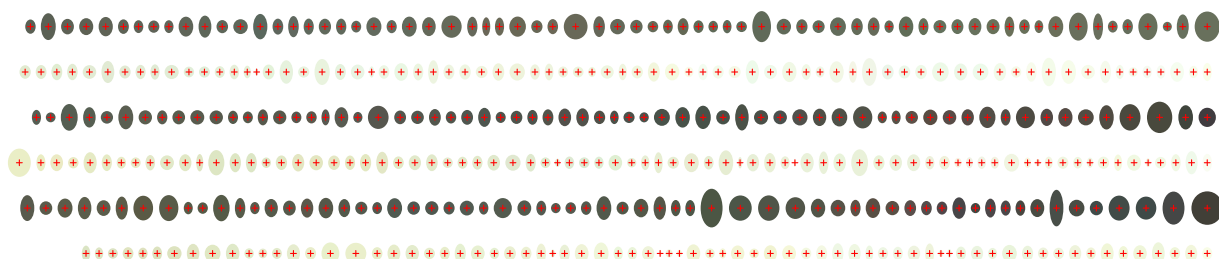
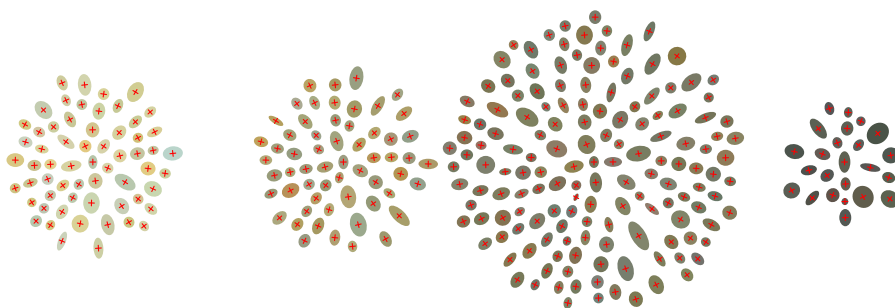
12

2015

Jller

Benjamin Maus

Prokop Bartoníček



**Additional Mechanics
and Electronics**
Tomislav Arnaudov

Developed at
pebe/lab (Prague)
and FELD (Berlin)

Jller is part of an ongoing research project in the fields of industrial automation and historical geology. This apparatus uses computer vision to sort pebbles from a specific river by their geologic age. The installation displays natural geological history in an automated sorting process that constitutes a performance in its own right.









Font Design
Marco Berends

Machine Learning MSc Thesis
Ankita Agrawal, Institute
for Artificial Intelligence,
HS Ravensburg-Weingarten

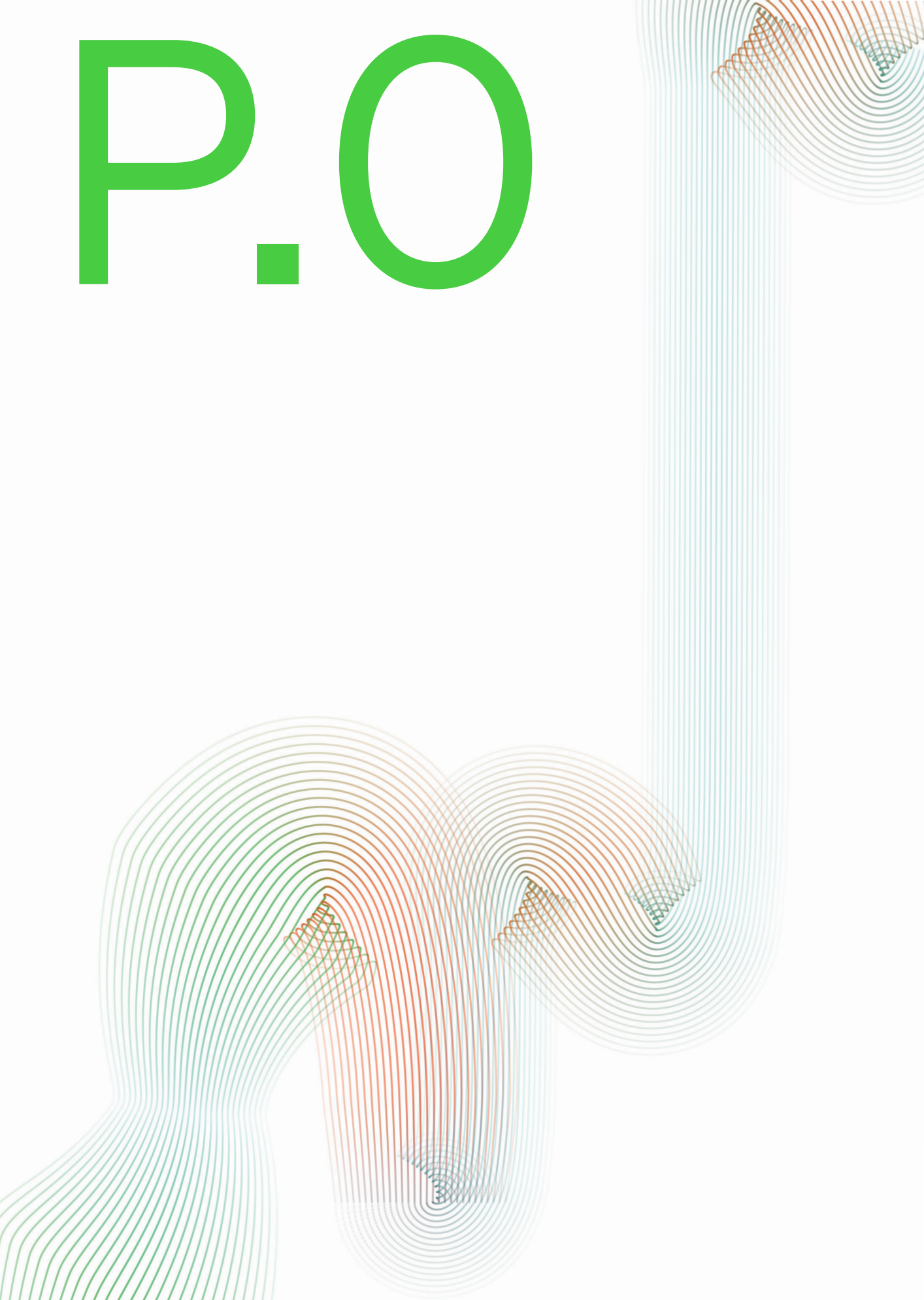
Funding and Support
Kickstarter Backers

Aerial Imagery
USGS (United States
Geological Survey)

Contrary to popular belief, much of the world has yet to be fully mapped. Every day, satellites orbit the earth, taking countless pictures, yet there is very limited knowledge about what exactly was captured among the unique things in these pictures and how they can be found and classified. **Aerial Bold**—a planet-wide letter search mission—questions this and draws attention to this potential.



P.O



Introduction to p5.js

P.0	Introduction to p5.js	42
P.0.0	p5.js, JavaScript, and Processing	44
P.0.1	The development environment	46
P.0.2	Language elements	48
P.0.3	Programming beautifully	56

P.0.0 **p5.js, JavaScript, and Processing**

The JavaScript library p5.js is used in this book to introduce the concepts of generative design.

Presented on the following pages is an overview of the basic elements and functions of JavaScript and a brief outline of the history of Processing and p5.js.

The project p5.js was started in August 2013 by Lauren McCarthy, an artist and assistant professor in UCLA's Design Media Arts program. To explain the idea behind p5.js, it is useful to start by referencing the much older Processing project.

The programming language Processing was initiated in the spring of 2001 by Ben Fry and Casey Reas with a small group of assistants. The main aim of Processing was and still is to give visually oriented people easy access to programming. Processing is a tool or development environment for the quick creation of digital, programmed sketches (a Processing program is called a “sketch”). **1** There is a large, vibrant community surrounding Processing that greatly simplifies collaborative learning. There are many examples, videos, tutorials, et cetera, available online. Processing has become standard for programming in design, infographics, architecture, and art.

→ processingfoundation.org

1 A very good and detailed introduction to programming with Processing can be found in the book *Processing: A Programming Handbook for Visual Designers and Artists*.

→ [Casey Reas and Ben Fry](#)

p5.js pursues the same goal: making it as easy as possible to program graphics. The individual commands of p5.js and Processing are extremely similar. The main difference is that unlike Processing, p5.js is based not on Java programming but on JavaScript. This is technically significant as well as important for users because JavaScript is the programming language that runs in web browsers and makes websites dynamic and interactive. The programs created with the help of p5.js can be used directly on the web.

p5.js is an open-source project; it can be downloaded for free and used for independent projects. Since p5.js is a JavaScript library and JavaScript is the most widely used programming language of the internet age, it is easy to translate the code examples to most other development environments. JavaScript is cross-platform, so the same source code can be used on all operating systems for which a modern browser or a corresponding runtime environment exists—not only on computers but also on smartphones and tablets.

Additionally, there is an enormous online community surrounding p5.js and, to an even greater extent, JavaScript. There is an online answer to almost any JavaScript-related question or problem. The internet is full of other JavaScript libraries, which can usually be combined easily with p5.js.

For the programs in this book, we use our own program library, which can be found on the book's website; we also have provided a list of external libraries. For example, you can use the generative-design-library to work on your graphics tablet or create color palettes in Adobe ASE format.



generative-gestaltung.de



[generative-design-library](#)



[Palette exporting P.1.2](#)

Color palettes

Writing Programs p5.js uses the JavaScript programming language, but it is simpler than pure JavaScript. The most important commands, especially those for graphical output (such as drawing a circle), are very easy to use. The beginner is not burdened with additional constructions that would otherwise be necessary to display graphic elements in the browser window.

On the following pages we will demonstrate how to set up a work space—the development environment—and introduce the most important language elements and programming concepts of p5.js. You can start immediately by typing the commands presented and seeing what those commands generate.

P.0.1 The development environment

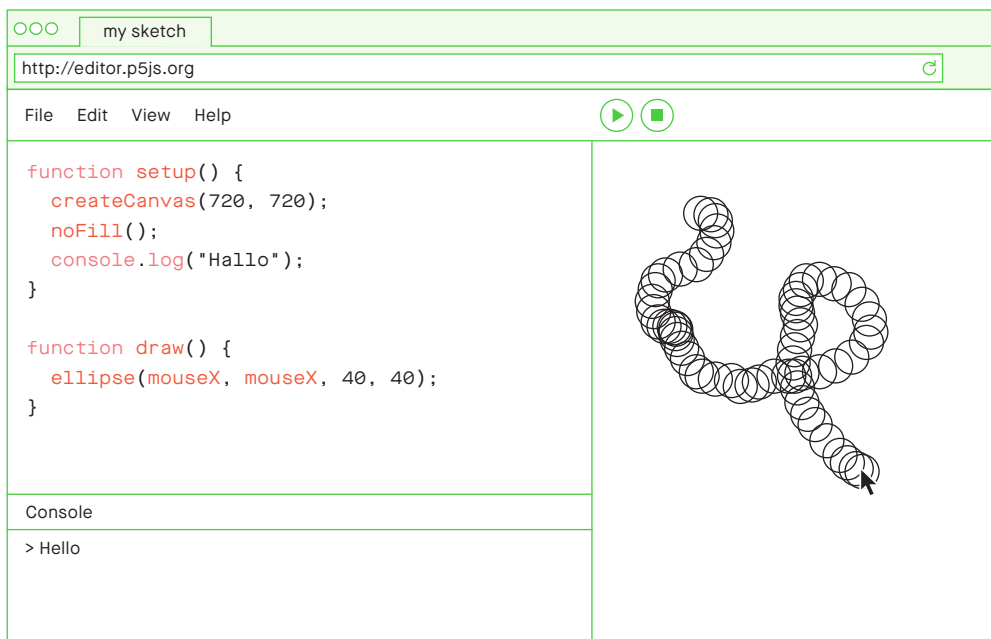
Itching to get started? Good! Here's how to try out our programs, expand on them, or create completely new ones. Just download our code package with all the programs of the book. The link can be found at generative-gestaltung.de.

Variation 1: The p5.js Web Editor This is the easiest way to start. All you need is a browser and access to the internet.

- 1 Open the online editor of p5.js. This provides a comfortable environment in which to try something quickly. [→ editor.p5js.org](http://editor.p5js.org)

In the browser window, there is an area to edit the code on the left side. There are play and stop buttons to start and stop the program; a canvas on the right where your sketch will be rendered; and, at the bottom, the console, which will display any error messages or other text output from the program.

- 2 Upload the book's programs to the online editor, copy and paste the text, or copy one of the sketches from the book's collection.



Code editor and output are united in the browser window. Several other online code editors, such as Codepen.io or jsfiddle.net, are similar to the p5.js web editor, but they are more generic solutions and are not optimized for p5.js.

Variation 2: Code Editor and Browser The web editor is a great way to initially experiment with p5.js; in the long term, however, we recommend that you set up a local working environment, as described below.

- 1 Above all, a good code editor, such as Sublime Text, Atom, Brackets, or Coda, is essential.
- 2 Download our code package. The link can be found on the book's website.
- 3 You can now run the program by opening the index.html from one of the sketch folders (e.g., P_1_O_01) in the browser.
- 4 To modify a program, open the sketch.js file in a text editor, make some edits, and open the corresponding index.html in the browser. Any changes made in sketch.js must be saved before the browser content is updated. Reload the browser to view the changes.

A more detailed guide to working with p5.js can be found on the p5.js website.

→ generative-gestaltung.de

→ p5js.org/get-started

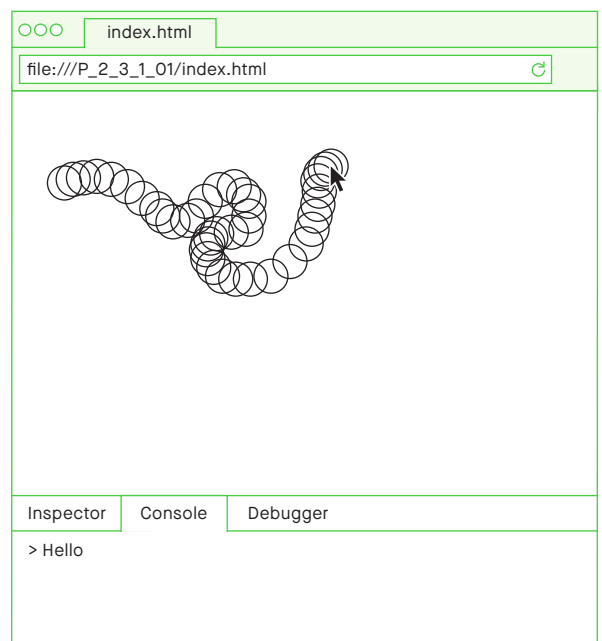
- ! Some of our code samples need to be run on a server to work. These sketches use external resources such as a webcam or external files and must be executed by a web server (the URL of the sketch must start with "http" or the browser will prohibit the use of these resources). Detailed explanations of how to use them can be found on the book's website.



```
function setup() {
  createCanvas(720, 720);
  noFill();
  console.log("Hello");
}

function draw() {
  background(255);
  ellipse(mouseX, mouseY, 40, 40);
}
```

A code editor, for example Sublime Text. The editor is a separate program and is not integrated in the browser.



The file index.html, and thus the sketch, is opened directly in the browser from the hard drive. The URL therefore also starts with "file."

P.0.2 Language elements

To be able to give instructions to a computer, you will need to speak its language. In the following section, the most important functions and control structures of JavaScript and p5.js will be introduced.

Hello, Ellipse A first program. Open the sketch.js file (located in the empty-example folder), enter the following line within the draw() function, and open the associated index.html in a browser.

```
1 function draw() {  
    ellipse(50, 50, 80, 80);  
}
```

There are commands that draw, such as `ellipse`, `rect`, `line`, et cetera, and commands that specify how the graphic that follows should be drawn, such as `stroke`, `strokeWeight`, `noStroke`, `fill`, `noFill`, et cetera. Once a drawing mode has been configured, it will apply for all additional drawing commands until the drawing mode is reconfigured. Most drawing commands require one or more parameters that indicate where something should be drawn and at what size. The unit of measure for this is the pixel. The origin of the coordinate system is located in the upper left corner of the drawing canvas.

```
2 point(60, 50);
```

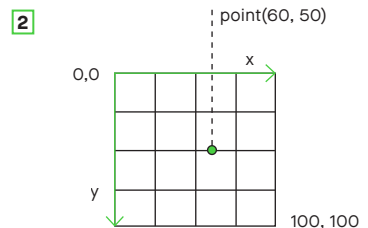
The pixel generated by the command `point(60, 50)` is thus drawn sixty pixels from the left edge and fifty pixels from the upper edge.

Of course, it is possible to have more than one line of code. The individual lines are then processed sequentially from top to bottom. The following command lines are thus read by p5.js:

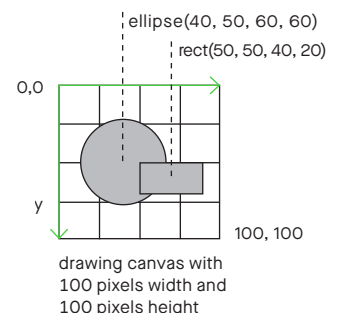
```
3 fill(128);  
strokeWeight(1);  
ellipse(40, 50, 60, 60);  
rect(50, 50, 40, 20);
```

With the commands, it is important to pay attention to the exact spelling, including uppercase and lowercase letters. For example, the `strokeWeight()` command would not be recognized if it were written as `strokeweight()` or `StrokeWeight()`.

1 A circle is drawn on the canvas. A **function** is created by declaring the word “function” followed by the function name and a set of parentheses `()`. What the function should do—for example, draw a circle—is defined between the curly brackets `{...}`.



3 Set the fill color to a medium gray. Set the line width to 1 pixel. Draw an ellipse at the point (40, 50) with width and height 60 pixels. Draw a rectangle to the coordinates (50, 50) with width 40 and height 20.



Setup, Draw, and Preload p5.js provides various functions that are called automatically. The two most important are `setup()` and `draw()`.

The `draw()` function is called in every drawing step and each command line is executed each time.

4 `function draw() { }`

5 `function draw() {
 console.log(frameCount);
}`

The `draw()` function is displayed with a preset frequency that specifies how many images are shown per second. The standard number is set at sixty images per second, but this can be reset using the command `frameRate()`.

6 `frameRate(30);`

However, if the computational effort per frame becomes so high that the browser is unable to execute it within the preset time, the set frame rate decreases automatically.

Some actions should only be performed once when the program is started and not repeatedly in each frame. The `setup()` function is used for this.

7 `function setup() {
 frameRate(30);
}`

The `preload()` function ensures that additional data are fully loaded when the program starts.

8 `function preload(){
 img = loadImage("data/pic1.jpg");
}`

4 Although this `draw()` function contains no commands, it keeps the program running.

5 A `draw()` function with one command. The command `console.log()` writes text to the console. In this case, it displays the continually increasing number of the actual frame.

6 This drawing speed is set at 30 images per second.

7 These commands, which should be executed just once when the program is started, appear in the `setup()` function.

8 In `preload()` the instructions for loading data are inserted; in this example an image should be loaded.

Drawing Canvas and Renderer A drawing canvas is created within the browser window. This is the display window for the program's visual output and can be scaled to any size.

9 `createCanvas(640, 480);`

In addition to the parameters' width and height, using the command `createCanvas()` makes it possible to enter which renderer should be used for displaying the image. The renderer is responsible for how the different graphic commands are actually translated into pixels. The following rendering options are available:

10 `createCanvas(640, 480, P2D);`

11 `createCanvas(640, 480, WEBGL);`

9 The drawing canvas is now 640 pixels wide and 480 pixels high.

10 The standard renderer: this is used if nothing else has been specified.

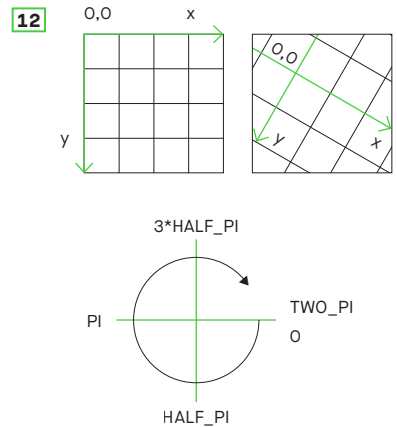
11 Renderer for hardware-accelerated 3D graphics in the web browser.

Transformations One of the strengths of p5.js is that it can move, rotate, and scale the coordinate system. All graphic commands thereafter refer to this altered coordinate system.

```
12 translate(40, 20);  
    rotate(0.5);  
    scale(1.5);
```

In this example, the coordinate system is moved 40 pixels to the right and 20 pixels downward, then rotated 0.5 radians (about 30°), and finally scaled by a factor of 1.5.

In Processing, angles are generally shown in radians, in which 180° equals the number pi (≈ 3.14) and the rotation is in a clockwise direction.



Variables and Data Types In a program, information is stored in variables so that it is available to other parts of the program. These variables can have any name other than the keywords for JavaScript keywords. Keywords can be recognized by their special color in the editor.

```
13 var myVariable;  
    myVariable = 5;
```

Variables can contain different data types. Unlike most other programming languages, JavaScript does not have to take this into account when creating a variable. Nevertheless, it is important to keep track of which variables store which type of value. For example, data types can be:

```
14 var myBooleanvalue = true;
```

```
15 var myInteger = 7;
```

```
16 var myFloatingPointNumber = -3.219;
```

```
17 var myCharacter = "A";
```

```
18 var myString = "This is a text";
```

13 The variable `myVariable` is created. The value 5 is then saved there.

14 Logic values (Boolean values): `true` or `false`

15 Integers: e.g., 50, -532.

16 Floating point value: e.g., 0.02, -73.1.

17 A single character: e.g., "a", "A", "9", "8".

18 Character string/text: e.g., "Hello, world".

Arrays When working with a large number of values, it is inconvenient to have to create a variable for each value. An array allows a list of values to be managed.

```
19 var planets = ["Mercury", "Venus", "Earth", "Mars",  
                "Jupiter", "Saturn", "Uranus", "Neptune"];
```

19 Arrays are defined by their square brackets. Within these, any number of values (here the eight planet names) can be listed, separated by commas.

20 `console.log(planet[0]):`

If values are not immediately assigned to an array, the array can be created first and filled later:

21 `var planetDiameter = [];
planetDiameter[0] = 4879;
planetDiameter[1] = 12104;
planetDiameter[2] = 12756;
planetDiameter[3] = 6794;
...`

Arrays have many other functions for processing elements. In this book we mainly use `<yourarray>.push()` to add new elements to the end of an array.

22 `planet.push ("Pluto");`

20 Values can be accessed by entering an index number in the square brackets following the variable name. The index 0 points to the first entry in the array.

21 The square brackets initialize an empty array. Values are then assigned to the array. This could also happen later while the program is running.

22 Although Pluto is no longer considered a planet, its name can be added to the array `planet` in this way.

Objects In addition to arrays, many pieces of information can be stored in objects at one time. The difference is that individual values are not accessed via a numeric index but via a key.

23 `var planet = {name: "Saturn", mass: 5.685e26,
 temperature: 134};
planet.diameter = 120536;

console.log("mass in kg: " + planet.mass);`

Alternatively, a value can be accessed as follows. This is necessary if the keys are not fixed from the beginning but are dynamically generated or read out.

24 `var k = "mass";
console.log("mass in kg: " + planet[k]);`

23 JavaScript objects are created by enclosing a set of key/value pairs within a set of curly brackets {}. Here an object with three key/value pairs is generated. Another key/value pair is added to the existing object, and one of the values is accessed and logged to the console.

24 If the name of the key is stored in a variable, then the value can be accessed using that variable as a reference to that key.

Operators and Mathematics Naturally, calculations can also be performed in Processing. This is possible with simple numbers...

25 `var a = (4 + 2.3) / 7;`

with strings...

26 `var s = "circumference of Jupiter: " + (142984*PI) + "
km";`

and with variables:

27 `var i = myVariable * 50;`

25 After execution the value 0.9 is saved in `a`.

26 The variable `s` then contains the string "circumference of Jupiter: 449197.5 km".

27 The value in `myVariable` is multiplied by 50 and the result saved in `i`.

The following computational operators are available: `+`, `-`, `*`, `/`, `%`. A whole series of mathematical functions are also available. Here are a few:

- 28** `var converted aValue = map(aValue, 10, 20, 0, 1);`
- 29** `var roundedValue = round(2.67);`
- 30** `var randomValue = random(-5, 5);`
- 31** `var cosineValue = cos(angle);`

- 28** The value in `aValue` is converted from a number between 10 and 20 to a number between 0 and 1.
- 29** Round: `roundedValue` is 3.
- 30** A random number between -5 and 5.
- 31** Calculates the cosine of the angle.

Mouse and Keyboard There are several ways to access information using the mouse and keyboard as input devices. One option is to query the variables available in `p5.js`.

- 32** `function draw() {
 console.log("mouseposition:" + mouseX + "," + mouseY);
 console.log("mousekey pressed?" + mouseIsPressed);
 console.log("key pressed?" + keyIsPressed);
 console.log("last pressed key:" + key);
}`

Another possibility is to implement one of the event handlers. An event handler is called when the corresponding event occurs—i.e., when a mouse button or key on the keyboard is pressed.

- 33** `function mouseReleased() {
 console.log("The mouse key was released");
}`

Additional event handlers include `mousePressed()`, `mouseMoved()`, `keyPressed()`, and `keyReleased()`.

- 32** The Processing variables `mouseX` and `mouseY` always contain the actual position of the mouse; `mouseIsPressed` is `true` if one of the mouse buttons is pressed in that moment. For the keyboard, `keyIsPressed` indicates if a key is pressed; the key last pressed appears in `key`.

- 33** The function `mouseReleased()` is called when the left mouse button is released.

Conditions It is often necessary to execute lines of code only when certain conditions are met. The `if` statement is used to do this.

- 34** `if (aNumber == 3) {
 fill(255, 0, 0);
 ellipse(50, 50, 80, 80);
}`
- 35** `if (aNumber == 3) {
 fill(255, 0, 0);
} else {
 fill(0, 255, 0);
}`
- 36** `if (aNumber == 3) fill(255, 0, 0);
else fill(0, 255, 0);`

- 34** The two lines of code inside the curly brackets are executed only when the condition inside the `if` statement is met—i.e., when the value of the variable `aNumber` is 3.

- 35** With `else`, the `if` condition is expanded by one code snippet that is executed when the specified `if` condition is not met.

- 36** If a code snippet consists of only one line, as in the previous example, the curly brackets can be eliminated.

Differentiating between multiple values of a variable can usually be achieved by the `switch` command.

```
37 switch (aNumber) {  
    case 1:  
        rect(20, 20, 80, 80);  
        break;  
    case 2:  
        ellipse(50, 50, 80, 80);  
        break;  
    default:  
        line(20, 20, 80, 80);  
}
```

37 The `switch` command tests if the value of the variable `aNumber` corresponds to one of the values listed in the `case` lines, jumps there if it does, and continues code execution until the following `break` statement.

If no corresponding value exists, program execution continues at `default`.

Functions There are often parts of a program that appear in a similar way in different places. In many such cases it is wise to encapsulate these parts in a function. For example:

```
38 function draw() {  
    translate(40, 15);  
    line(0, -10, 0, 10);  
    line(-8, -5, 8, 5);  
    line(-8, 5, 8, -5);  
    translate(20, 50);  
    line(0, -10, 0, 10);  
    line(-8, -5, 8, 5);  
    line(-8, 5, 8, -5);  
}
```

38 The coordinate system is moved to another location, where a star is made of three lines. The coordinate system is then moved again, and a star is drawn using the same commands at the new location.

To change the star's appearance in this program, the drawing commands have to be changed in two places. It is therefore better to store the corresponding lines in a function.

```
39 function draw() {  
    translate(40, 15);  
    drawStar();  
    translate(20, 50);  
    drawStar();  
}  
  
function drawStar() {  
    line(0, -10, 0, 10);  
    line(-8, -5, 8, 5);  
    line(-8, 5, 8, -5);  
}
```

39 The function `drawStar()` now contains the drawing commands. This function can be called from any other point in the program to execute the code contained there.

Values can be passed in functions, which can also return a value as a result.

```
40 function setup() {
    console.log("The faculty of 5 is 1*2*3*4*5 = "
        + faculty(5));
}

function faculty(theValue) {
    var result = 1;
    for (var i = 1; i <= theValue; i++) {
        result = result * i;
    }
    return result;
}
```

In JavaScript, it is particularly common for functions to accept as one of its arguments another function called a "callback function." Callback functions allow actions to occur when triggered by an event.

```
41 loadJSON("myData.json", callback);

function callback(data) {
    console.log(data);
}
```

40 A function `faculty()` is defined here, in which a value is passed, `theValue`.

When the function is called, the parameter, 5 in this case, is passed in the variable `theValue`.

In the function, various calculations are executed with this value, and the result is returned using the command `return`.

41 When the p5.js function `loadJSON()` is used in this way, the program flow is not blocked by the loading process but runs in the background (asynchronous loading). When the file is fully loaded, the `callback()` function is called.

Loops Loops are used to execute a particular command several times within a program. You can program loops in several ways:

The `for` loop is used to loop a code snippet for a specific number of repetitions.

```
42 for (var i = 0; i <= 5; i++) {
    line(0, 0, i * 20, 100);
    line(100, 0, i * 20, 100);
}
```

A `while` loop will continue as long as a certain condition is fulfilled.

```
43 var myvalue = 0;
while (myValue < 100) {
    myValue = myValue + random(5);
    console.log("The value of myValue is " + myValue);
}
```

42 The two lines of code inside the curly brackets are executed exactly six times. First the variable `i` is set to the value 0, then increased by 1 (`i++`) after each cycle, as long as the value is 5 or less.

43 This `while` loop will continue as long as the value of the variable `myValue` is less than 100. A random value between 0 and 5 is added in each new loop cycle.

There are several ways to go through all the values in an array. Some of the usual methods are shown here:

```
var planets = ["Mercury", "Venus", "Earth", "Mars",  
              "Jupiter", "Saturn", "Uranus", "Neptune"];
```

44

```
for (var i = 0; i < planets.length; i++) {  
    console.log(planets[i]);  
}
```

45

```
planet.forEach(function(planet) {  
    console.log(planet);  
});
```

In order to traverse all key/value pairs of an object, the following variation of the for loop is suitable:

```
var planet = {name: "Saturn", mass: 5.685e26,  
             temperature: 134, diameter: 120536};
```

46

```
for (k in planet) {  
    console.log("Key: " + k + ", Value: " + planet[k]);  
}
```

44 Variation 1: The for loop runs if the variable `i` is smaller than the length of the array.

45 Variation 2: The array function `forEach()` is used here. This requires a callback function, which is called in every iteration of the loop. The variable `planet` contains the individual values of the array successively.

46 The variable `k` contains each key of the object in succession. This can be used to access the associated values.

P.0.3 Programming beautifully

In programming there are usually many ways to attain a desired result. The fact that a program works, however, is just one part of the solution, albeit a very important one; there are other aspects you should consider.

Comments The trickier and more complex a program is, the harder it is for others—and for you, too, after a short time—to understand it. A program that is not accessible cannot be modified or updated. Comments in the program help to clarify the code.

1

```
// calculate distance between actual and previous mouse
// position, which represents the speed of the mouse
var speed = dist(mouseX, mouseY, pmouseX, pmouseY);
```

1 Text that follows two forward slashes is ignored and can be used for comments.

The p5.js community is spread across the entire world. It is therefore advisable to write comments and variable names in English. This makes it easier for you and others to look for and find suggestions and answers to problems on internet forums.

Useful Names and Clear Structures In addition to the use of comments, sensibly selected variable and function names help users to maintain an overview and to understand what a specific program does.

2

```
function mixer(apples, oranges) {
  var juice = (apples + oranges) / 2;
  return juice;
}
```

2 In this example it is difficult to recognize just what the function calculates. The fact that the average of two numbers is calculated can only be guessed from the formula.

To this end, it is necessary to structure the program into smaller, logical sections with the help of functions and classes. This encapsulation of functionality also means that the program can be quickly modified and extended.

Performance Even though computers are becoming faster and faster, it still takes time to execute a command. Even short periods of time add up noticeably when a command is executed often. Try not to burden the program with unnecessary work.

3

```
for (var i = 0; i < 10000000; i++) {  
    var speed = dist(mouseX, mouseY, pmouseX, pmouseY);  
    doSomethingWithMouseSpeed(speed);  
}
```

4

```
var speed = dist(mouseX, mouseY, pmouseX, pmouseY);  
for (var i = 0; i < 10000000; i++) {  
    doSomethingWithMouseSpeed(speed);  
}
```

3 Here mouseSpeed is calculated in all ten million loops even though the mouse position cannot be changed during this process.

4 Performance is therefore improved enormously when the mouse speed is calculated outside the loop.

With the programs in this book, we have tried to uphold the aforementioned principles. Sometimes, however, clarity and performance contradict each other, meaning that program code optimized for performance can be more difficult to understand. In such cases we have opted for clearer structures.

P.1



Color

In contrast to this book, the colors of which we perceive through reflected and filtered light, computers are true quantum catapults. When we look at a screen, light is sent directly into our eyes in various wavelengths and can be manipulated in real time. The following examples provide users with the most important technical features for working with color and introduce several ways to use color when designing on the screen.

P.1 Color

58

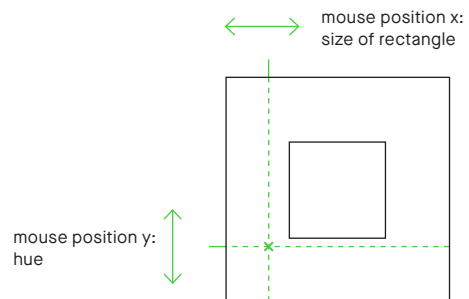
P.1.0	Hello, color	60
P.1.1	Color spectrum	62
P.1.1.1	Color spectrum in a grid	62
P.1.1.2	Color spectrum in a circle	64
P.1.2	Color palettes	66
P.1.2.1	Color palettes through interpolation	66
P.1.2.2	Color palettes from images	68
P.1.2.3	Color palettes from rules	72

P.1.0 Hello, color

The ability to directly influence 16,777,216 colors gives users an amazing freedom. Simultaneous contrast—without which it would be impossible to perceive colors—is illustrated here by juxtaposing a number of color combinations. Our perception of a color is affected by its neighboring color and the shifting proportions of that color to its background.

→ P_1_0_01

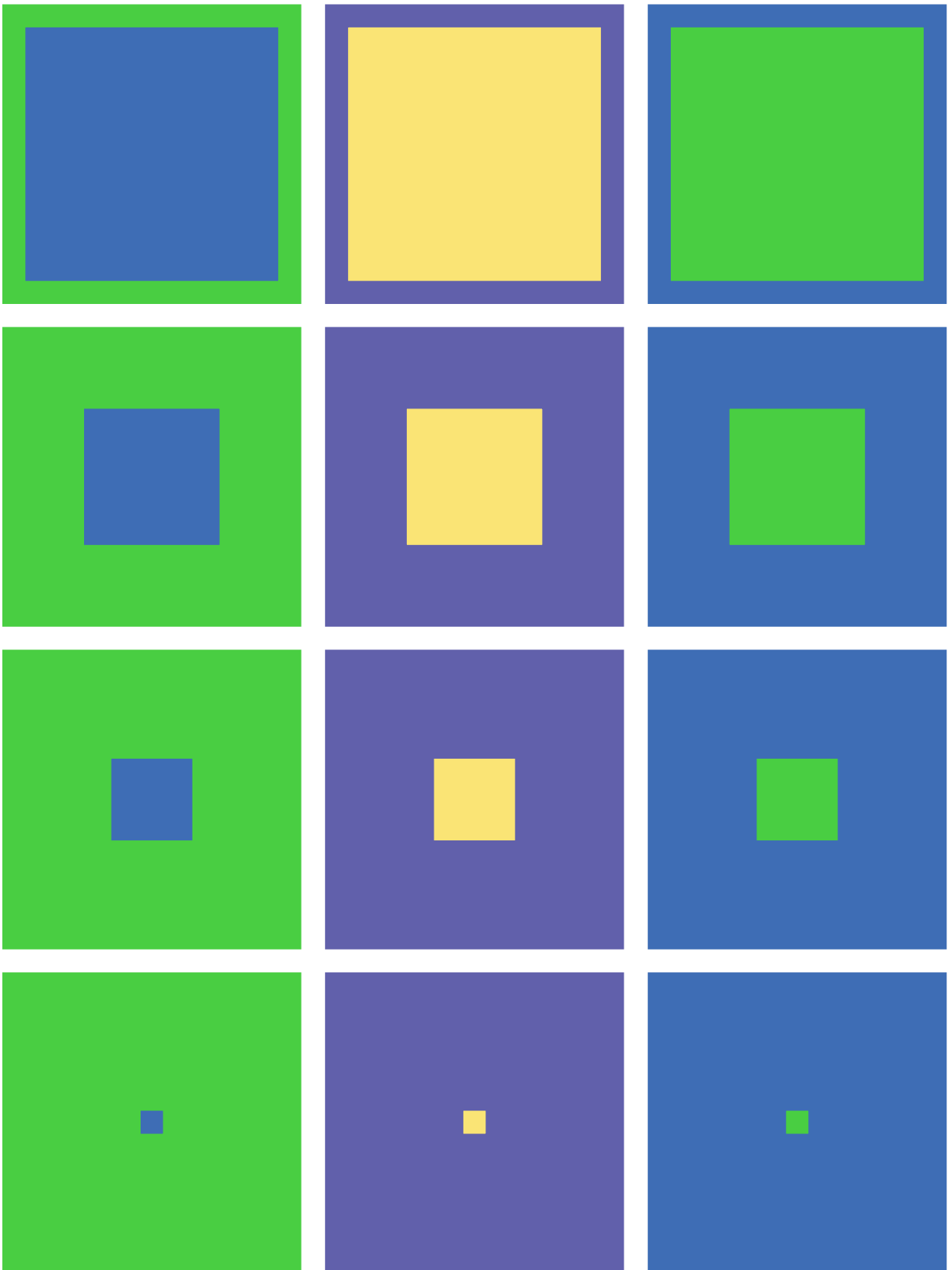
The horizontal position of the mouse controls the size of the color field. Starting in the center, the colored area is depicted with a height and width of 1 to 720 pixels. The vertical mouse position controls the hue. The background passes through the color spectrum from 0 to 360, while the color field passes through the spectrum in the opposite direction, from 360 to 0.



```
1 function setup() {  
  createCanvas(720, 720);  
  noCursor();  
  
2  colorMode(HSB, 360, 100, 100);  
  rectMode(CENTER);  
  noStroke();  
}  
  
function draw() {  
3  background(mouseY / 2, 100, 100);  
  
4  fill(360 - mouseY / 2, 100, 100);  
5  rect(360, 360, mouseX + 1, mouseX + 1);  
}
```

Mouse: Position x: Size of rectangle
Position y: Hue
Keys: S: Save image

- 1 The `setup()` function sets the size of the display window and makes the cursor invisible.
- 2 The colors should pass through the hue spectrum in this program. For this reason, `colorMode()` allows users to change the way color value is interpreted. `HSB` specifies the color model, and the three values following it specify the respective range. Hue, for example, can only be specified by values between 0 and 360.
- 3 The y-value of the mouse position is divided by 2 to get values from 0 to 360 on the color wheel.
- 4 The halved y-value of the mouse position is subtracted from 360, creating values from 360 to 0.
- 5 The size of the color field changes relative to the x-value of the mouse position, with a side length between 1 and 720 pixels.



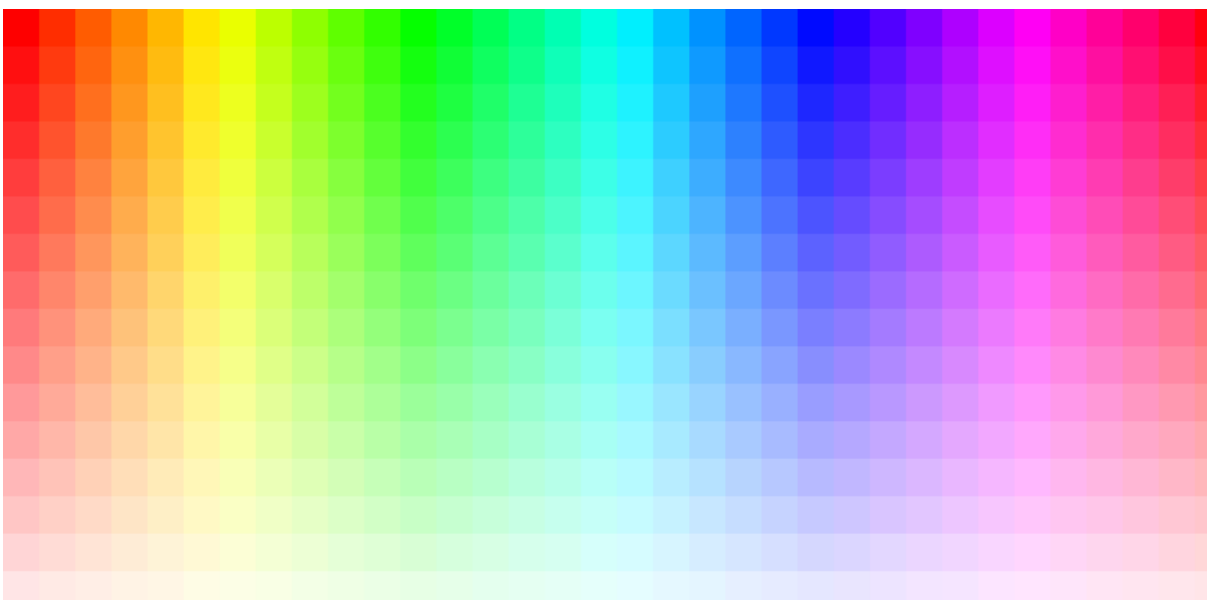
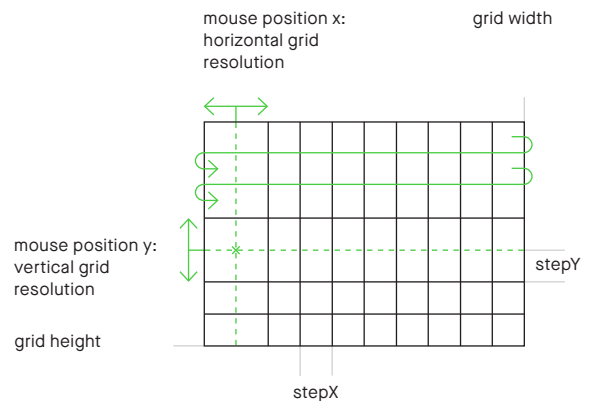
→ P_1_0_01 The x-value of the mouse position defines the size of the inside color field, the y-value the hue.

P.1.1.1 Color spectrum in a grid

This color spectrum is composed of colored rectangles. Each tile is assigned a hue on the horizontal axis and a saturation value on the vertical. The color resolution can be reduced by enlarging the rectangles so that the primary colors in the spectrum become clearer.

→ P_1_1_1_01

The grid is created by two nested for loops. In the outer loop, the y-position is increased, step by step. The inner loop then draws a line by increasing the value for the rectangle's x-position, step by step, until the entire width is processed. The step size is set by the value of the mouse position and is located in the variables stepX and stepY. It also determines the length and width of the rectangles.



→ P_1_1_1_01 Although the distance of the color values is the same between all the color tiles, the contrasts are perceived as greater in some positions than in others.

```

var stepX;
var stepY;

function setup() {
  createCanvas(800, 400);
  noStroke();
  colorMode(HSB, width, height, 100);
}

```

1

2

```

function draw() {
  stepX = mouseX + 2;
  stepY = mouseY + 2;

  for (var gridY = 0; gridY < height; gridY += stepY) {
    for (var gridX = 0; gridX < width; gridX += stepX) {
      fill(gridX, height - gridY, 100);
      rect(gridX, gridY, stepX, stepY);
    }
  }
}

```

3

4

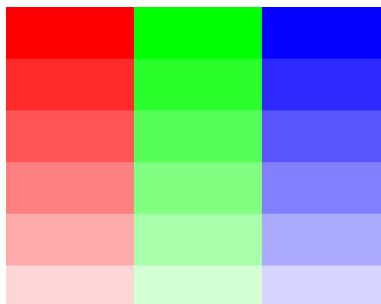
5

Mouse: Position x/y: Grid resolution
 Keys: S: Save image

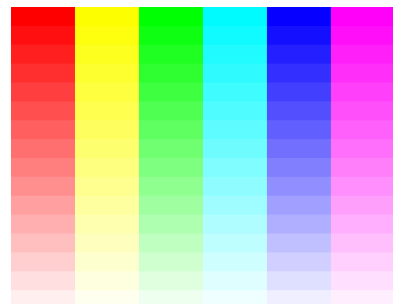
- 1 The display size is set using `createCanvas()`. The values defined here can be retrieved at any time using the system variables `width` and `height`.
- 2 The value range for hue and saturation is set at 800 or 400 using the command `colorMode()`. Hue is no longer defined as a number between 0 and 360 but rather as one between 0 and 800. The same is true of the saturation value.
- 3 The addition of 2 prevents `stepX` or `stepY` from being too small, which would lead to longer display times.
- 4 With the help of two nested loops, all the positions in the grid will now be processed. The y-position of the rectangle to be drawn is defined by `gridY` in the outer loop. The value increases only when the inner loop has been processed (i.e., once a complete row of rectangles has been drawn).
- 5 The variables `gridX` and `gridY` are used not only to position the tile but also to define the fill color. The hue is determined by `gridX`. The saturation value decreases proportionally to increases in the value `gridY`.



→ P_1_1_1_01 A soft rainbow effect occurs at full resolution.



→ P_1_1_1_01 The primary colors of a computer monitor—red, green, and blue—in various gradations.



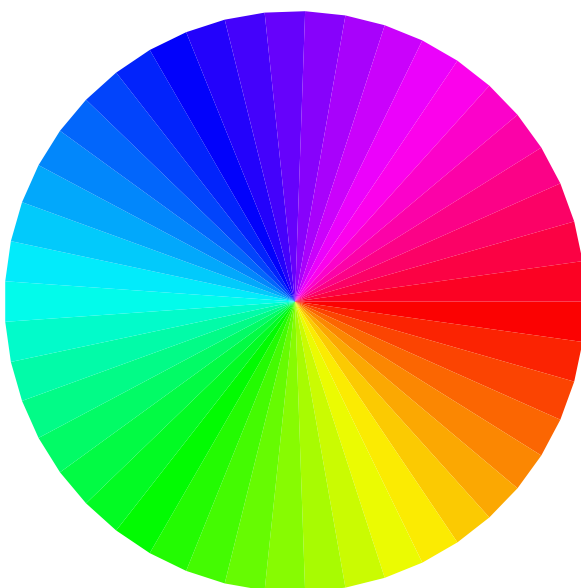
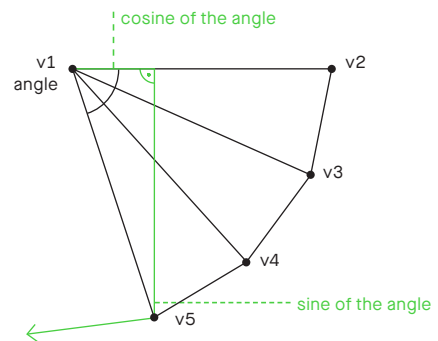
→ P_1_1_1_01 The secondary colors are also visible at this resolution.

P.1.1.2 Color spectrum in a circle

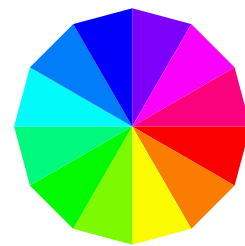
There are numerous models for organizing colors. The color spectrum arranged in a circle (a color wheel) is a popular model for comparing harmonies, contrasts, and color tones. You can control the number of circle segments, as well as their brightness and saturation values, allowing you to better understand the color arrangement in HSB mode.

→ P_1_1_2_01

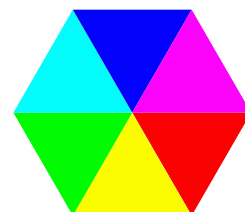
The color-wheel segments are arranged in the shape of a fan. The individual vertices are computed from the cosine and sine values of the corresponding angle. A Processing method mode can be used that makes it especially easy to create the wheel segments. The individual points have to be set in the following order: first the middle point, and then the outer ones sequentially.



→ P_1_1_2_01 Segment count 45, key 2.



→ P_1_1_2_01 Segment count 12, key 4.



→ P_1_1_2_01 Segment count 6, key 5.

```

1 function draw() {
    colorMode(HSB, 360, width, height);
    background(360, 0, height);

2     var angleStep = 360 / segmentCount;

    beginShape(TRIANGLE_FAN);
3     vertex(width / 2, height / 2);

    for (var angle = 0; angle <= 360; angle += angleStep) {
4         var vx = width / 2 + cos(radians(angle)) * radius;
        var vy = height / 2 + sin(radians(angle)) * radius;
        vertex(vx, vy);
5         fill(angle, mouseX, mouseY);
    }

6     endShape();
}

```

```

function keyPressed() {
    ...
7     switch (key) {
        case '1':
            segmentCount = 360;
            break;
        case '2':
            segmentCount = 45;
            break;
        case '3':
8         segmentCount = 24;
            break;
        case '4':
            segmentCount = 12;
            break;
        case '5':
            segmentCount = 6;
            break;
    }
}

```

Mouse: Position x: Saturation
 Position y: Brightness
Keys: 1-5: Segment count
 S: Save image

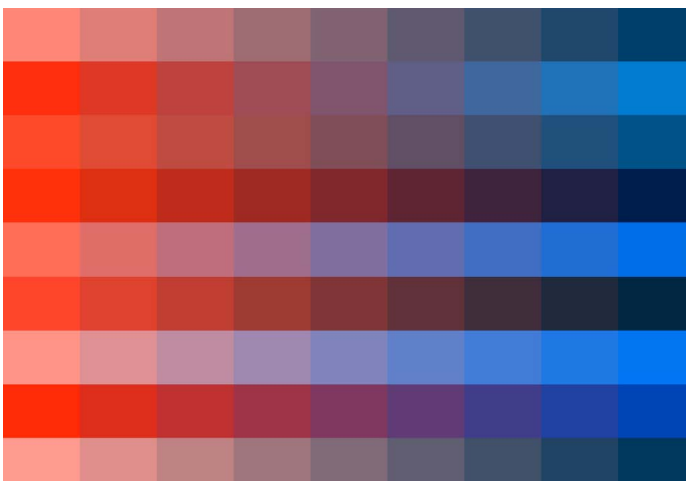
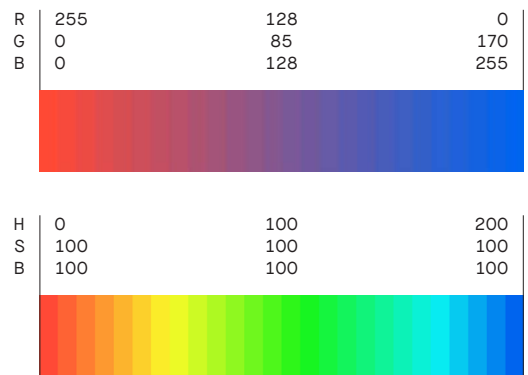
- 1 The ranges of values for saturation and brightness are adjusted in such a way that mouse coordinates can be taken as their values.
- 2 The angle increment `angleStep` depends on how many segments are to be drawn (`segmentCount`).
- 3 The first vertex point is in the middle of the drawing canvas.
- 4 For the other vertices, `angle` has to be converted from degrees (0–360) to radians (0–2π) because the functions `sin()` and `cos()` require the angle be input this way. This conversion is done by using `radians()`.
- 5 The fill color for the next segment is defined: the value of `angle` as hue, `mouseX` as saturation, and `mouseY` as brightness.
- 6 The construction of the color segment is ended with `endShape()`.
- 7 The `switch()` command checks the last `key` pressed, which enables easy switching between different cases.
- 8 If the key 3 was pressed, e.g., then `segmentCount` is set to the value 24.

P.1.2.1 Color palettes through interpolation

In each color model, colors have their clearly defined place. The direct path from one color to another always has precisely definable gradations, which will vary depending on the specific model. Using this interpolation you can create color groups in every gradation, as well as locate individual intermediate nuances.

→ P_1_2_1_01

Because a color is not defined by a single number but by several values, it is necessary to interpolate between these values. Depending on the chosen color model, RGB or HSB, the same color is defined by different values, thereby causing the path from one to another to lead past different colors. In the HSB color model, for instance, a detour is made past the color wheel. This difference is due to the characteristics of the color models, both of which can be very useful, depending on the situation. It is therefore important to choose the appropriate color model to solve a specific problem.



→ P_1_2_1_01 Interpolation in the RGB color space. In each row, two colors are interpolated in nine steps.


```

function draw() {
  tileCountX = int(map(mouseX, 0, width, 2, 100));
  tileCountY = int(map(mouseY, 0, height, 2, 10));
  var tileWidth = width / tileCountX;
  var tileHeight = height / tileCountY;
  var interCol;
  ...
  for (var gridY = 0; gridY < tileCountY; gridY++) {
    var col1 = colorsLeft[gridY];
    var col2 = colorsRight[gridY];

    for (var gridX = 0; gridX < tileCountX; gridX++) {
      var amount = map(gridX, 0, tileCountX - 1, 0, 1);
      ...
      interCol = lerpColor(col1, col2, amount);
      ...
      fill(interCol);

      var posX = tileWidth * gridX;
      var posY = tileHeight * gridY;
      rect(posX, posY, tileWidth, tileHeight);
      ...
    }
  }
}

```

Mouse: Left click: New random color set

Position x: Resolution

Position y: Number of rows

Keys: 1-2: Interpolation style

S: Save image

C: Save ASE palette

- 1** The number of color gradations `tileCountX` and the number of rows `tileCountY` are determined by the position of the mouse.
 - 2** Drawing the grid row by row.
 - 3** The colors for the left and right columns are set in the arrays `colorsLeft` and `colorsRight`.
 - 4** The intermediary colors are calculated with `lerpColor()`. This function performs the interpolation between the individual color values. The variable `amount`, a value between 0 and 1, specifies the position between the start and end color.
- !** In all programs that are described in the color palette chapters, a color palette in ASE format for Adobe applications can be saved using the C key.



→ **P.1_2_1_01** Interpolation in the HSB color space. Although the colors at the beginning and end of the rows are the same as those in the image opposite, completely different intermediary color steps are generated.

P.1.2.2 Color palettes from images

We constantly are surrounded by color palettes; we need only to record and evaluate them. The colors obtained from the photograph of one individual's wardrobe—a very personal color palette—are selected and sorted using the following program. You can export the resulting color collection and use it as an inspirational color palette.

→ P_1_2_2_01

The pixels of a loaded image are scanned using the mouse position in a specific grid spacing, one by one and row for row, in order to define the respective color value. These values are stored in an array and can be sorted by hue, saturation, brightness, or gray value.

mouse position x:
horizontal grid
resolution

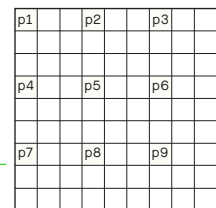
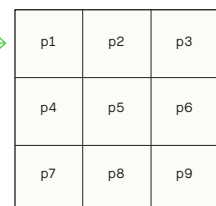


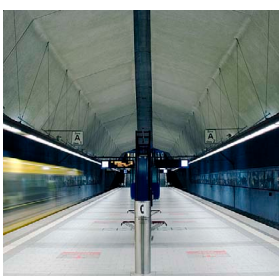
image is scanned in a specific
grid spacing

[p1, p2, p3, p4, ...]

pixel values are saved
in an array and
rearranged



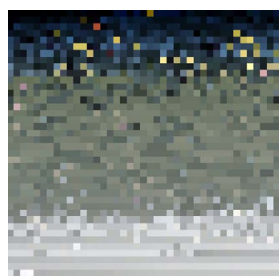
palette with color fields



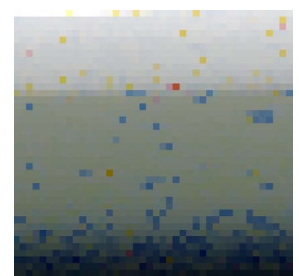
→ Fotografie: Stefan Eigner
Original image: Subway tunnel.



→ P_1_2_2_01 Pixels arranged
according to hue.



Pixels arranged according to
saturation.



Pixels arranged according to
brightness.

```

var img;
var colors = [];
var sortMode = null;

```

```

function preload(){
  img = loadImage("data/pic1.jpg");
}

```

```

function draw() {
  var tileCount = floor(width / max(mouseX, 5));
  var rectSize = width / tileCount;

```

```

  img.loadPixels();
  colors = [];

```

```

  for (var gridY = 0; gridY < tileCount; gridY++) {
    for (var gridX = 0; gridX < tileCount; gridX++) {
      var px = int(gridX * rectSize);
      var py = int(gridY * rectSize);
      var i = (py * img.width + px) * 4;
      var c = color(img.pixels[i], img.pixels[i+1],
                    img.pixels[i+2], img.pixels[i+3]);
      colors.push(c);
    }
  }

```

```

  gd.sortColors(colors, sortMode);

```

```

  var i = 0;
  for (var gridY = 0; gridY < tileCount; gridY++) {
    for (var gridX = 0; gridX < tileCount; gridX++) {
      fill(colors[i]);
      rect(gridX*rectSize, gridY*rectSize,
           rectSize, rectSize);
      i++;
    }
  }
}

```

```

function keyReleased(){
  if (key == 'c' || key == 'C') writeFile(
    [gd.ase.encode(colors)],
    gd.timestamp(), 'ase');
  ...
  if (key == '5') sortMode = null;
  if (key == '6') sortMode = gd.HUE;
  if (key == '7') sortMode = gd.SATURATION;
  if (key == '8') sortMode = gd.BRIGHTNESS;
  if (key == '9') sortMode = gd.GRAYSCALE;
}

```

Mouse: Position x: Resolution

Keys: 1-4: Change example image

5-9: Change sort mode

S: Save image

C: Save ASE palette

1 The currently selected sorting mode is always stored in the variable `sortMode`. The default is to not sort, and the value is therefore set at `null` (undefined).

2 Before the program starts to run, the image is loaded and stored into the variable called `img`.

3 The number of rows and columns in the grid `tileCount` depends on the x-value of the mouse. The function `max()` selects the larger of the two given values.

4 The grid resolution just calculated is now used to define the size of the tiles, `rectSize`.

5 Only after calling `loadPixels()` can the individual pixels of the image be accessed.

6 Now the picture is scanned line by line in the previously calculated grid spacing, `rectSize`. The pixels are stored in the `pixels[]` array as a long list of values. Therefore, from `px` and `py`, the corresponding index `i` must be calculated.

7 The colors are sorted using the `sortColors()` function. This function must pass the array `colors` and sort mode `sortMode`.

8 In order to draw the palette, the grid is processed again. The fill colors for the tiles are taken, value by value, from the array `colors`.

9 The function `ase.encode()` allows an array of `colors` to be saved as an Adobe Swatch Exchange (ASE) file. The palette can then be loaded as a color swatch library, e.g., in Adobe Illustrator.

10 The keys 5 to 9 control what sorting function is applied to colors. For this the `sortMode` is set at `null` (no sorting) or at one of the constants, `HUE`, `SATURATION`, `BRIGHTNESS`, or `GRAYSCALE`, from the Generative Design library.

→ P_1_2_2_01 "pic4.jpg" with sorting by gray value.



→ Photograph: **Steffen Knöll**.



→ [P.1.2.2_01](#) "pic4.jpg" with sorting by hue.

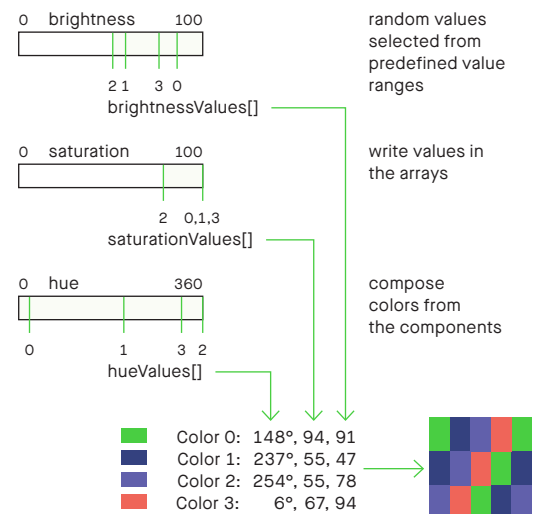
P.1.2.3 Color palettes from rules

All colors are made up of three components: hue, saturation, and brightness. The values for these color components are defined using a set of rules. By using controlled random functions, you can quickly create different palettes in specific color nuances.

→ P_1_2_3_01

Values for hue, saturation, and brightness are randomly selected from predefined ranges of values. This combination of rule sets—the definition of value ranges—and random functions means that new palettes are continually created and that they always produce specific color nuances.

Because the perception of color depends on context, the produced colors are drawn in an interactive grid. Even the color nuances emerge more distinctly.



```
1 var hueValues = [];  
  var saturationValues = [];  
  var brightnessValues = [];  
  
function draw() {  
  ...  
2  var index = counter % currentTileCountX;  
  
  fill(hueValues[index],  
       saturationValues[index],  
       brightnessValues[index]);  
  rect(posX, posY, tileWidth, tileHeight);  
  counter++;  
  ...  
}
```

- 1 An individual array is used to save hue, saturation, and brightness. Depending on what key is pressed (0–9), the arrays are filled according to different rules.
- 2 When the grid is drawn, the colors are selected from the arrays, one by one. The continually incrementing variable `counter` starts to cycle through the same values because of the modulo operator, `%`. When `currentTileCountX` is 3, e.g., `index` will consecutively hold the values 0, 1, 2, 0, 1, 2, ... This means only the first colors in the arrays are used in the grid.


```

3  if (key == '1') {
    for (var i = 0; i < tileCountX; i++) {
      hueValues[i] = int(random(0, 360));
      saturationValues[i] = int(random(0, 100));
      brightnessValues[i] = int(random(0, 100));
    }
  }

```

3 When key 1 is pressed, the three arrays are filled with random values from the complete ranges of values. This means any color can appear in the palette.

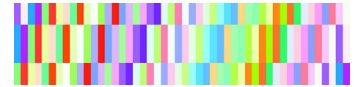


```

4  if (key == '2') {
    for (var i = 0; i < tileCountX; i++) {
      hueValues[i] = int(random(0, 360));
      saturationValues[i] = int(random(0, 100));
      brightnessValues[i] = 100;
    }
  }

```

4 Here brightness is always set at the value 100. The result is a palette dominated by bright colors.



```

5  if (key == '3') {
    for (var i = 0; i < tileCountX; i++) {
      hueValues[i] = int(random(0, 360));
      saturationValues[i] = 100;
      brightnessValues[i] = int(random(0, 100));
    }
  }

```

5 When the saturation value is set at 100, no pastel tones are created.



```

6  if (key == '7') {
    for (var i = 0; i < tileCountX; i++) {
      hueValues[i] = int(random(0, 180));
      saturationValues[i] = int(random(80, 100));
      brightnessValues[i] = int(random(50, 90));
    }
  }

```

6 Here a restriction occurs in all color components; e.g., the hues are only selected from the first half of the color wheel, creating warmer colors.



```

7  if (key == '9') {
    for (var i = 0; i < tileCountX; i++) {
      if (i % 2 == 0) {
        hueValues[i] = int(random(0, 360));
        saturationValues[i] = 100;
        brightnessValues[i] = int(random(0, 100));
      } else {
        hueValues[i] = 195;
        saturationValues[i] = int(random(0, 100));
        brightnessValues[i] = 100;
      }
    }
  }

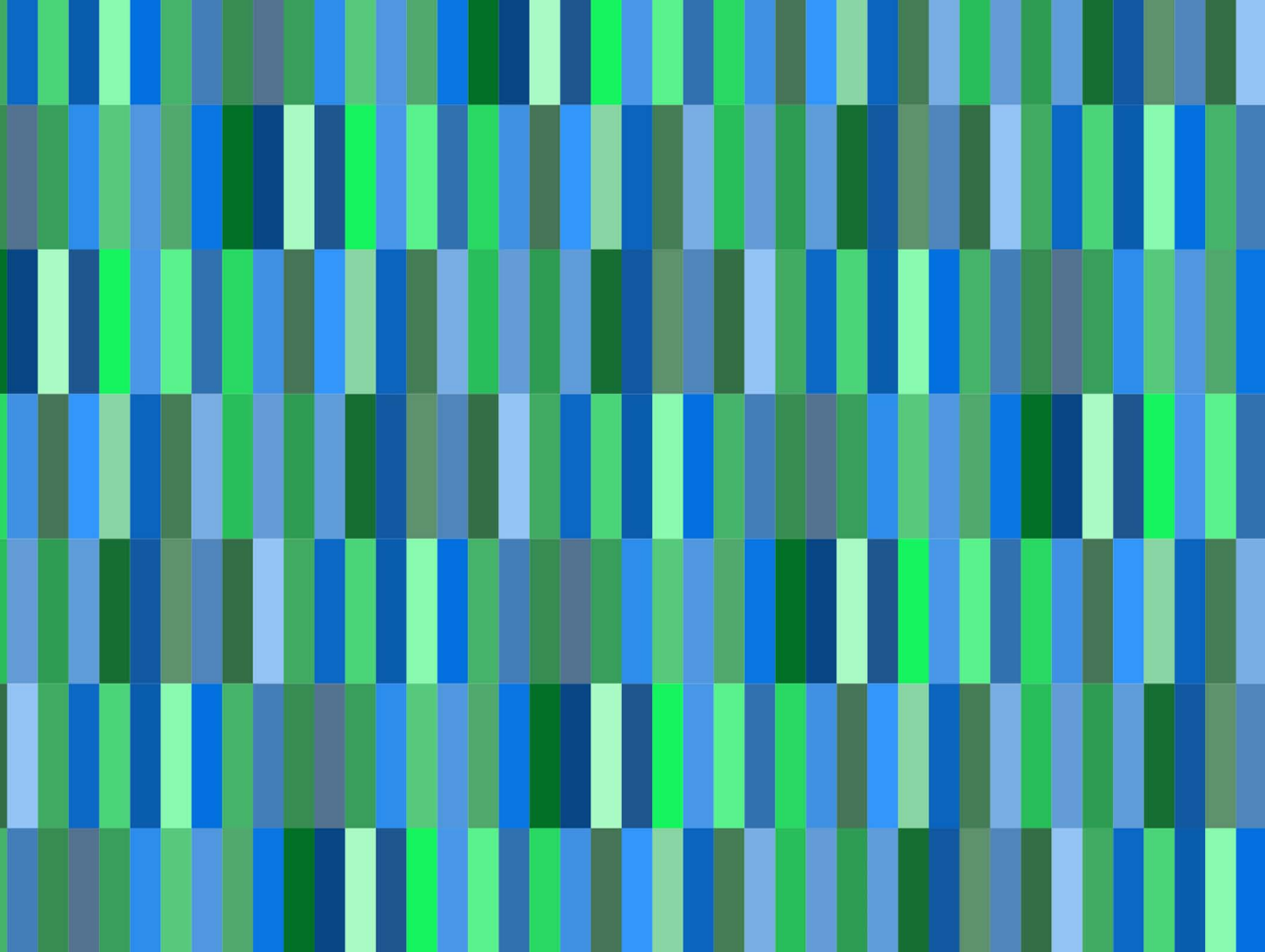
```

7 It is also possible to mix two color palettes. The expression `i % 2` produces alternately the numbers 0 and 1. When the result is 0, a darker, more saturated color is saved in the array.

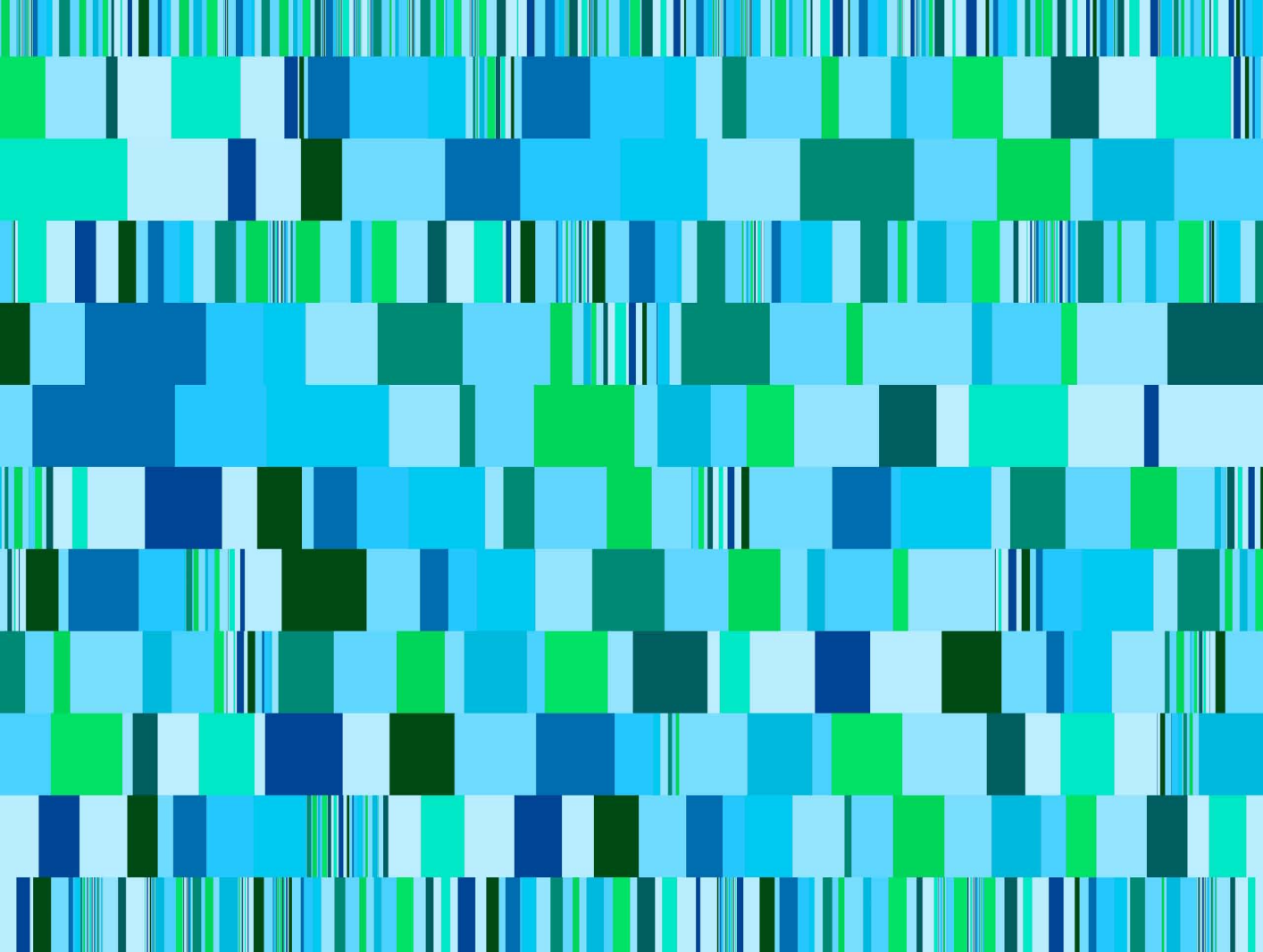
8 Otherwise the second rule is applied, and hue and brightness are set to fixed values. These values produce bright blue tones.



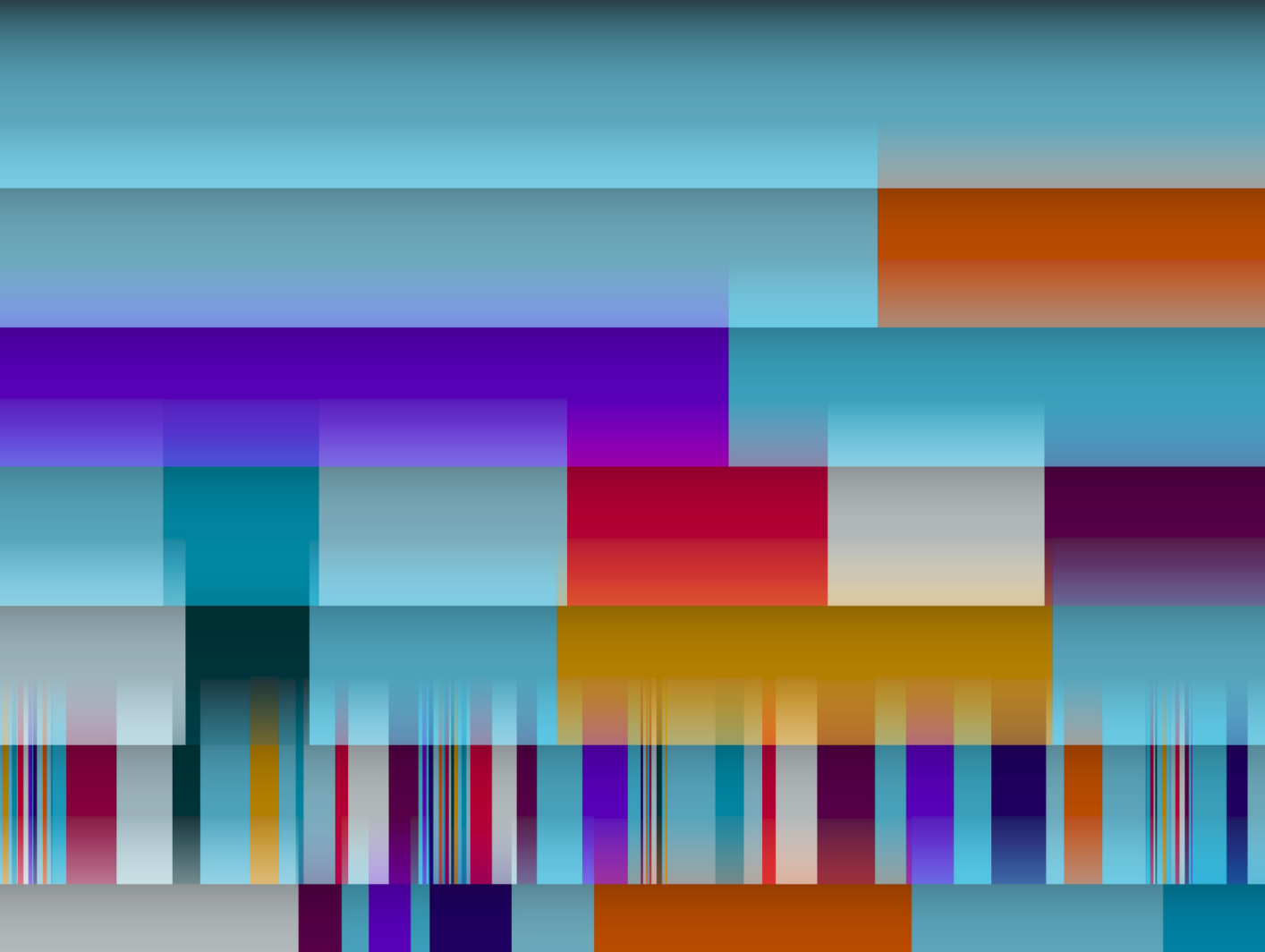
Mouse: Position x/y: Grid resolution
Keys: O-9: Change color palette
 S: Save image
 C: Save ASE palette



→ **P_1_2_3_01** The O key creates a color palette in which two preset hues (cyan and violet) alternate. Saturation and brightness are varied randomly.



→ [P_1_2_3_02](#) Randomness plays a larger role here. The rows are divided into tiles of different widths. It is randomly decided which tiles are divided again.



→ **P_1_2_3_03** If the tiles are slightly transparent and overlapping, another series of colors is created in addition to those in the color palette.



→ P_1_2_3_05 The main difference between this image and the previous one is that here about half of the tiles are simply not drawn.

P.2



Shape

The previous chapter was primarily devoted to color, with shape playing a lesser role. Since it is possible to control every element of an image, we are able to access, modularize, and automate an enormous variety of not only colors but also shapes. A dialog with form emerges.

P.2 Shape	78
P.2.0 Hello, shape	80
P.2.1 Grid	82
P.2.1.1 Alignment in a grid	82
P.2.1.2 Movement in a grid	86
P.2.1.3 Complex modules in a grid	90
P.2.1.4 Checkboxes in a grid	94
P.2.1.5 From grid to moiré	98
P.2.2 Agents	102
P.2.2.1 Dumb agents	102
P.2.2.2 Intelligent agents	104
P.2.2.3 Shapes from agents	108
P.2.2.4 Growth structure from agents	112
P.2.2.5 Structural density from agents	116
P.2.2.6 Agents on a pendulum	120
P.2.3 Drawing	126
P.2.3.1 Drawing with animated brushes	126
P.2.3.2 Relation and distance in drawing	130
P.2.3.3 Drawing with type	132
P.2.3.4 Drawing with dynamic brushes	134
P.2.3.5 Drawing with the pen tablet	138
P.2.3.6 Drawing with complex modules	142
P.2.3.7 Drawing with multiple brushes	146

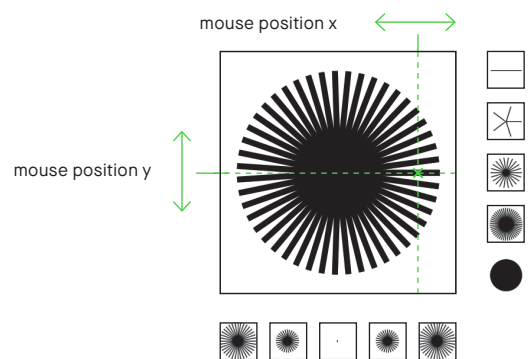
P.2.0 Hello, shape

Are point, line, and plane still the holy trinity of every form? Wassily Kandinsky's determination of these basic elements is more relevant than ever when viewed in the context of generative design. In keeping with this approach, the pixel is the origin of the small black circle in the image below. Lines are created by a series of pixels while planes are created by a collection of connected lines.

→ P_2_0_01

The shape is reduced to one pixel when the cursor is located in the middle of the upper border of the display window.

The x-value of the mouse position sets the length of the lines. The y-value sets the value and number of the lines.

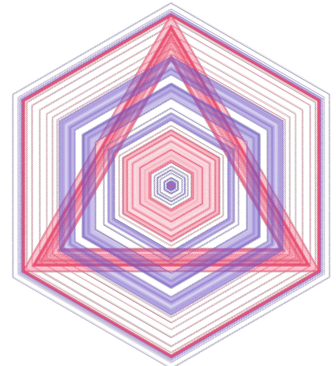
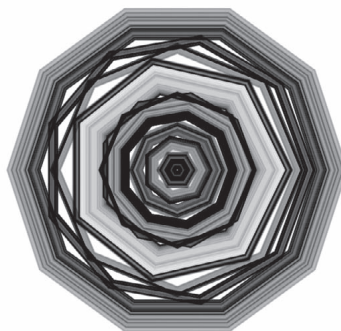
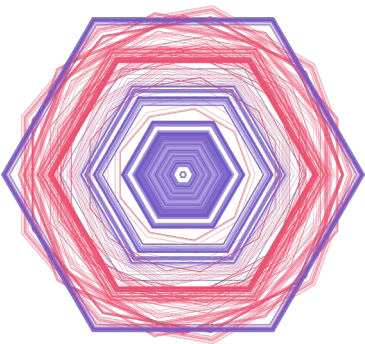
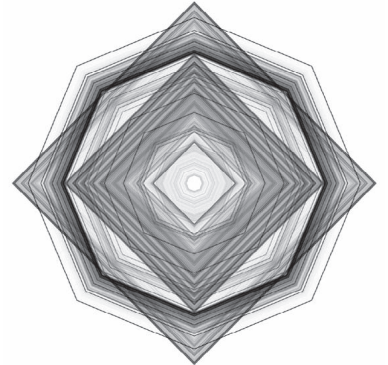
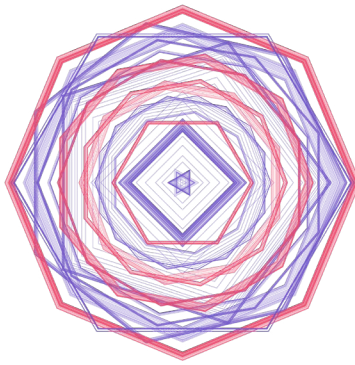
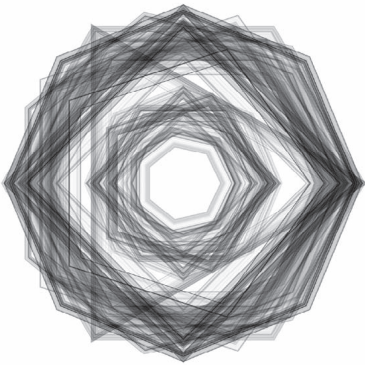
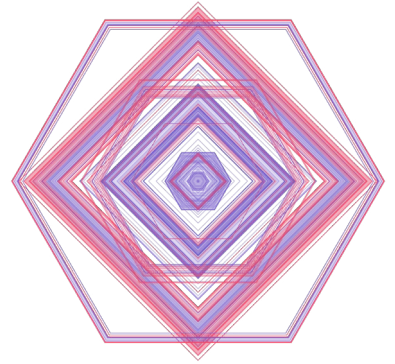
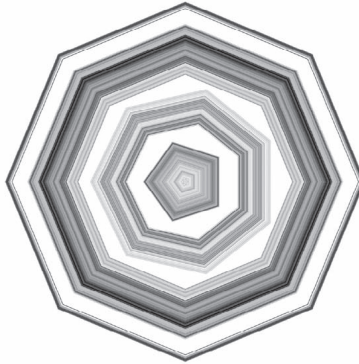
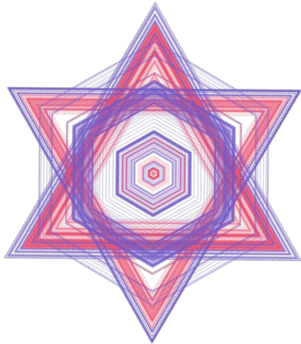
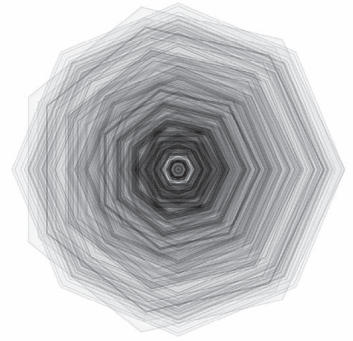
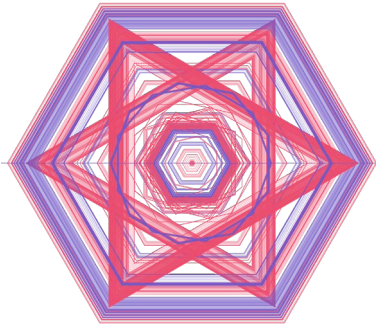
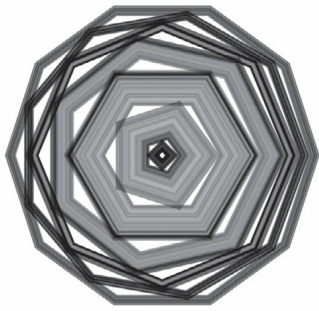


```
function draw() {  
  background(255);  
  translate(width / 2, height / 2);  
  
  var circleResolution = map(mouseY, 0, height, 2, 80);  
  var radius = mouseX - width / 2 + 0.5;  
  var angle = TWO_PI / circleResolution;  
  
  strokeWeight(mouseY / 20);  
  
  beginShape();  
  for (var i = 0; i <= circleResolution; i++) {  
    var x = cos(angle * i) * radius;  
    var y = sin(angle * i) * radius;  
    line(0, 0, x, y);  
    // vertex(x, y);  
  }  
  endShape(CLOSE);  
}
```

Mouse: Position x: Line length
Position y: Value and number of lines

Keys: S: Save image

- 1 The origin of the coordinate system is moved to the center of the drawing canvas.
- 2 The function `map()` converts the y-value of the mouse position (a number between 0 and height) to a value between 2 and 80.
- 3 By subtracting half of the display's width from the x-value of the mouse position, the radius becomes increasingly smaller the more the mouse is moved toward the center. Adding 0.5 to the x-value ensures the diameter of the circle is at least 1.
- 4 The increment `angle` is calculated as the full circle, `TWO_PI`, divided by the number of lines, `circleResolution`.
- 5 The end points of the line can be connected as a closed vertex by deleting the comment characters `//`.



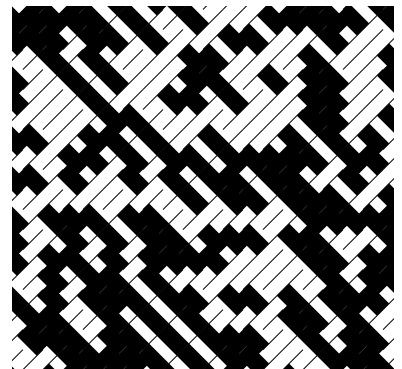
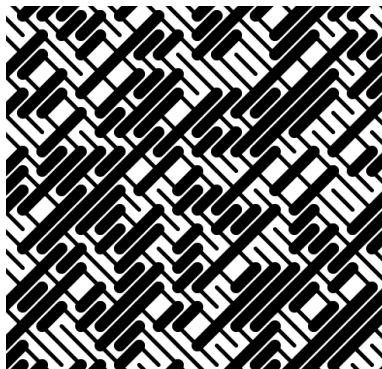
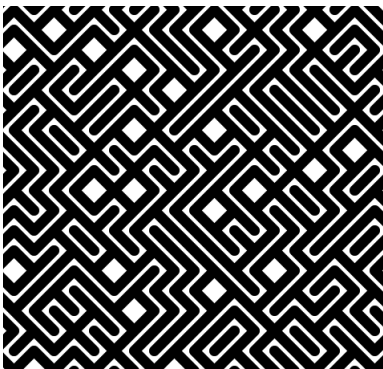
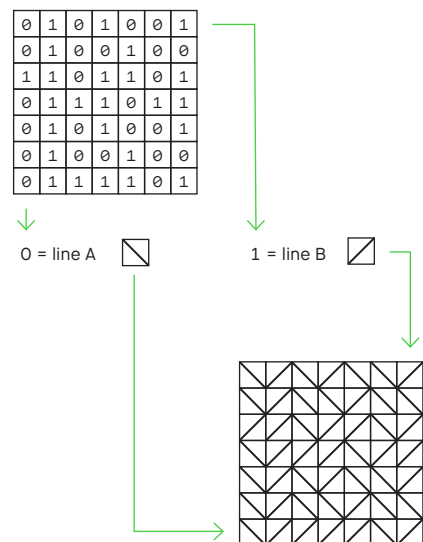
→ P_2_0_02 and → P_2_0_03 In this variation, the end points of the star forms are connected to the closed polygons. In addition, when drawing with the mouse, the transformation tracks are retained because the background is not given a new color.

P.2.1.1 Alignment in a grid

How can a diagonal with only two possible directions use the strict order of a grid to make complex structures? The direction of the diagonals in every grid is decided randomly. Overlapping, resulting from changes in line width, creates new forms, connections, and interstices.

→ P_2_1_1_01

In a grid, either a line A is drawn from the upper-left to the lower-right corner or a line B is drawn from the lower-left to the upper-right corner. The direction of the diagonals is selected randomly.



→ P_2_1_1_01 The line values of the diagonals are linked to the mouse position, and the types of line endings are switched using keys 1 to 3.

```

1  var tileCount = 20;

function draw() {
  ...
  strokeCap(actStrokeCap);
  ...
  for (var gridY = 0; gridY < tileCount; gridY++) {
    for (var gridX = 0; gridX < tileCount; gridX++) {
      var posX = width / tileCount * gridX;
      var posY = height / tileCount * gridY;
      var toggle = int(random(0, 2));
      if (toggle == 0) {
        strokeWeight(mouseX / 20);
        line(posX, posY, posX + width / tileCount,
              posY + height / tileCount);
      }
      if (toggle == 1) {
        strokeWeight(mouseY / 20);
        line(posX, posY + width / tileCount,
              posX + height / tileCount, posY);
      }
    }
  }
  ...
}

```

```

6  function keyReleased() {
  ...
  if (key == '1') actStrokeCap = ROUND;
  if (key == '2') actStrokeCap = SQUARE;
  if (key == '3') actStrokeCap = PROJECT;
}

```

Mouse: Position x: Line value of left diagonals
 Position y: Line value of right diagonals
 Left click: New random value

Keys: 1-3: Change line endings
 S: Save image

1 The value of the variable `tileCount` defines the grid resolution.

2 The command `random(0,2)` creates a random number between 0.000 and 1.999. When converting this into an integer using `int`, it is rounded off and the variable `toggle` becomes either 0 or 1.

3 In this `if` query, the value of the variable `toggle` is compared to 0. When this statement is true, the next two lines of code are executed, drawing line A.

4 The value of the current mouse position on the x-axis defines the stroke value of line A. It is divided by 20 so the stroke value does not become too large.

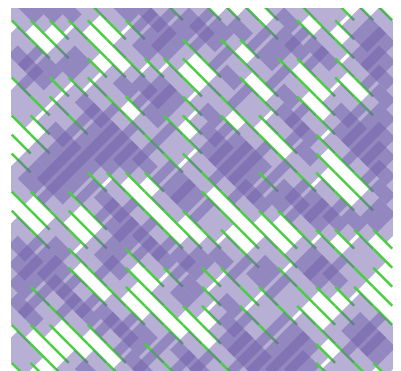
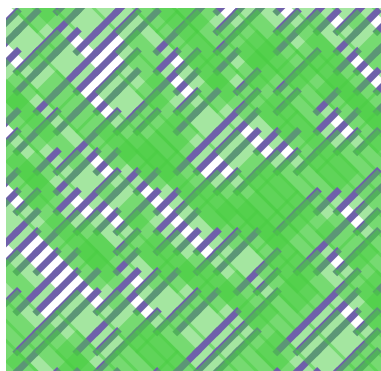
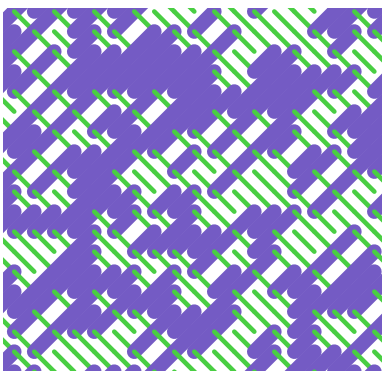
5 The same applies to line B.

6 Pressing one of the keys 1 to 3 sets the variable `actStrokeCap` to one of the Processing constants: `ROUND`, `SQUARE`, or `PROJECT`. This parameter can then be used to set how the ends of the lines are drawn using the function `strokeCap()`.

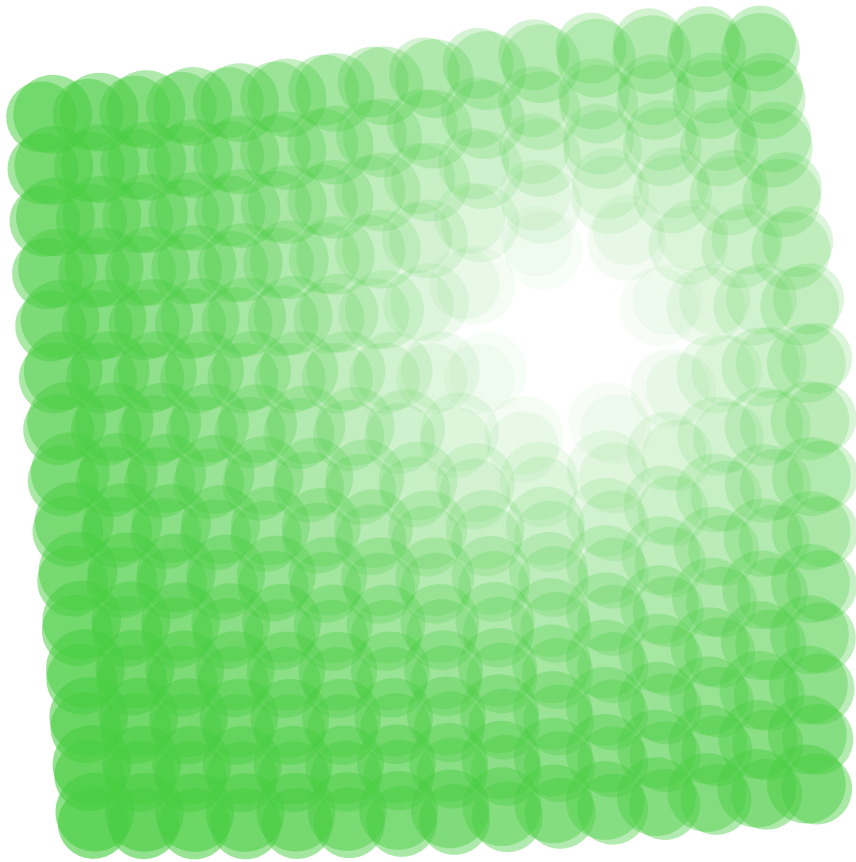
7 `strokeCap(ROUND)`

8 `strokeCap(SQUARE)`

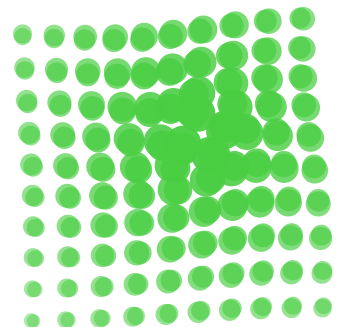
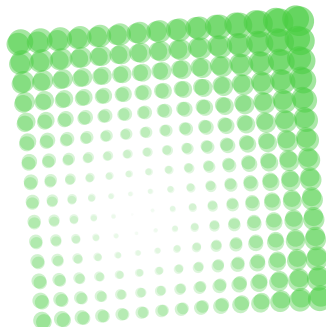
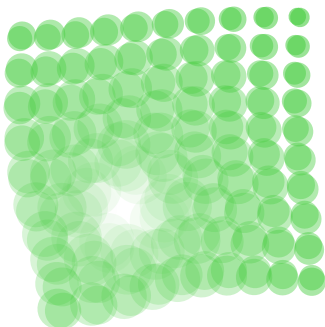
9 `strokeCap(PROJECT)`



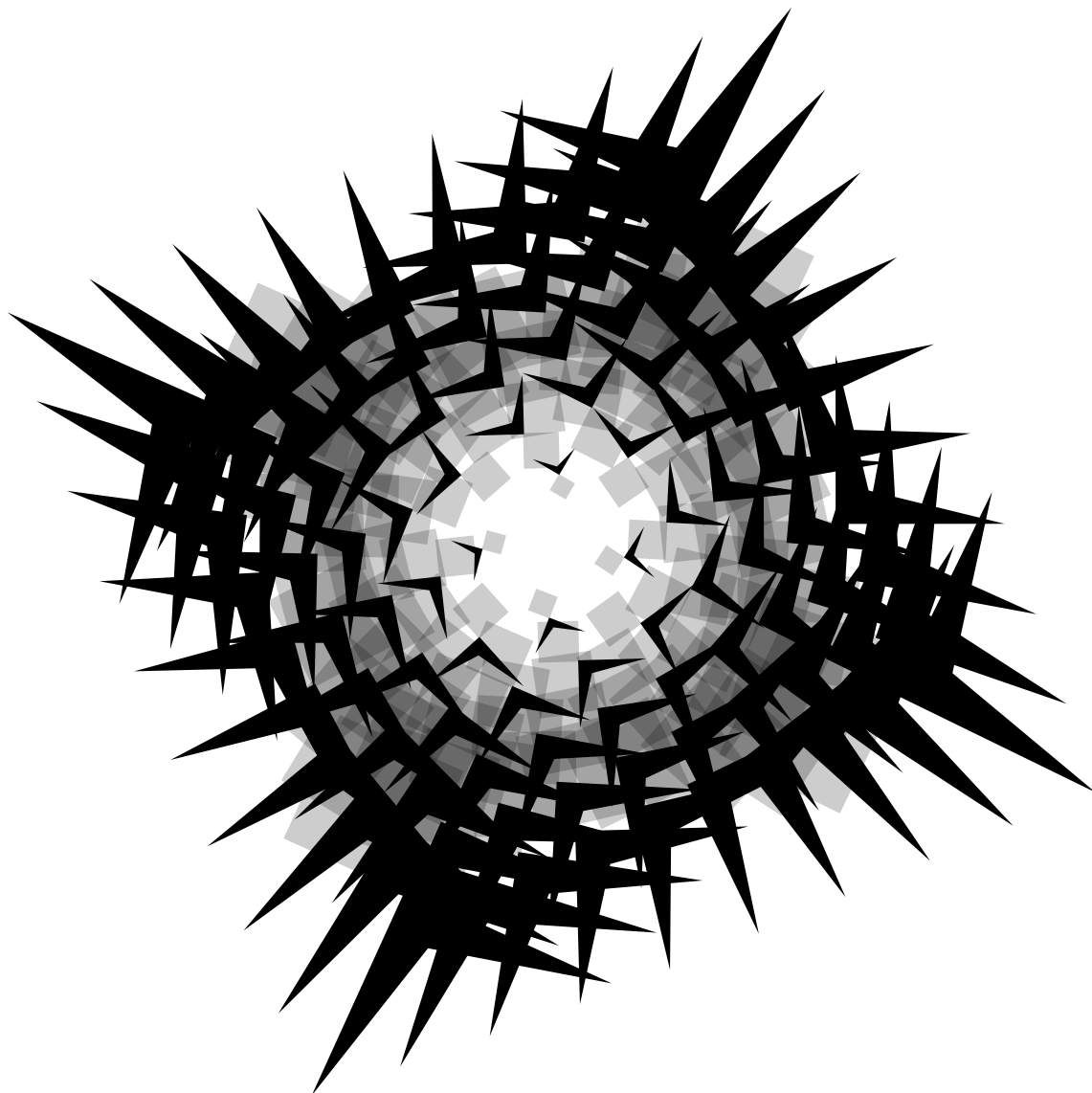
→ P_2_1_1_02 In this variation, additional colors and transparencies are used for both types of diagonals.



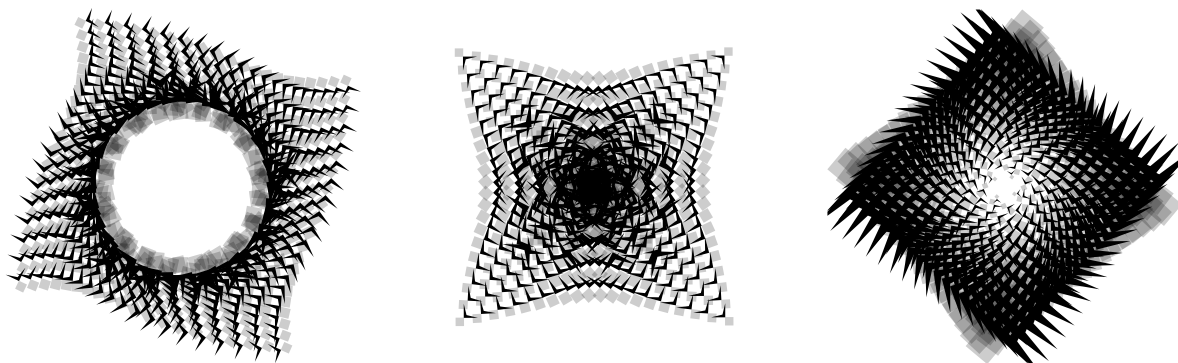
→ **P_2_1_1_04** Here the elements in the grid are loaded out of the data file SVG graphics, which are always turned toward the mouse position. Using the key C, it is possible to switch between different color modes. The transparency of an element depends on its distance from the mouse position.



→ **P_2_1_1_04** Using key D, it is possible to select whether the elements become smaller or larger as they near the mouse.



→ **P_2_1_1_04** Additional SVG modules can be selected with the keys 1 to 7. When the SVG graphics are decentralized in position, the grid appears to dissolve, although the modules are drawn strictly within the grid.



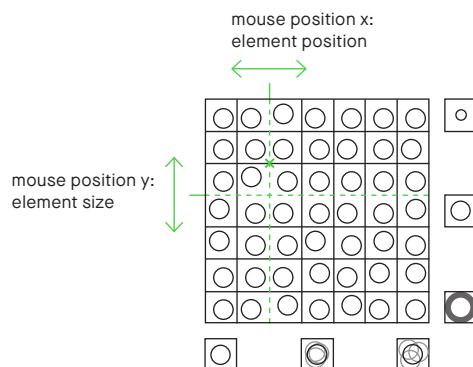
→ **P_2_1_1_04** The elements can also be rotated (left- and right-arrow keys).

P.2.1.2 Movement in a grid

Tension is highest when order borders on chaos. Individual forms abandon their strict arrangement in the dynamic grid and submit to random configurations. Elements inclined to the grid and those adverse to it fight for visual supremacy. It is the moments of transition that are important.

→ P_2_1_2_01

A fixed number of circles are drawn to the display one by one, line by line. Random values are added to the grid position of a circle, forcing it to move along the x- and y-axes. The farther the mouse is moved to the right, the greater the circle's movement.



```
function draw() {
  translate(width / tileCount / 2, height / tileCount / 2);
  ...
  strokeWeight(mouseY / 60);

  for (var gridY = 0; gridY < tileCount; gridY++) {
    for (var gridX = 0; gridX < tileCount; gridX++) {

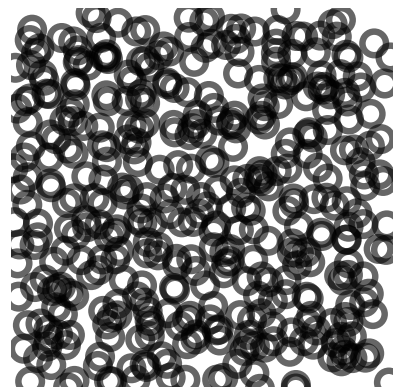
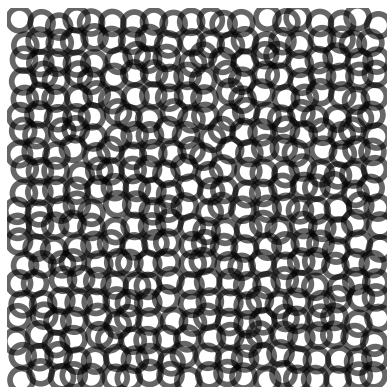
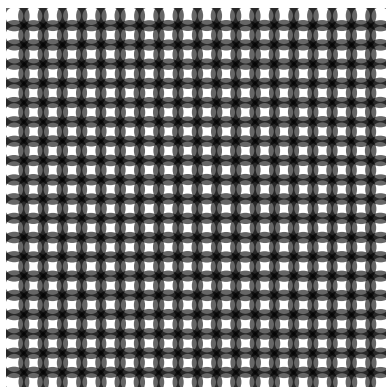
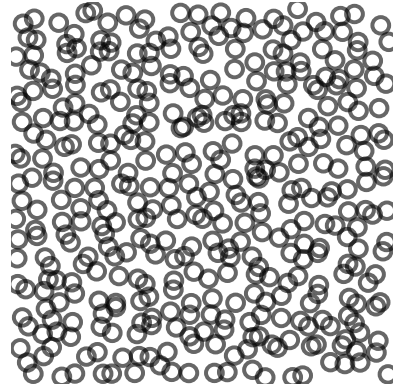
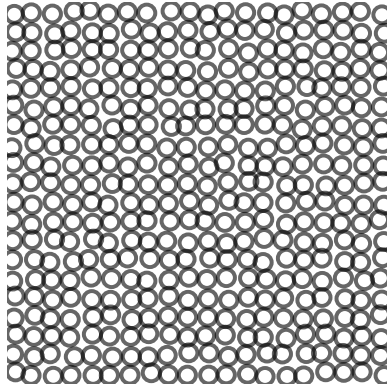
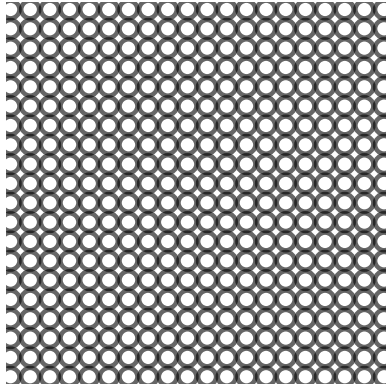
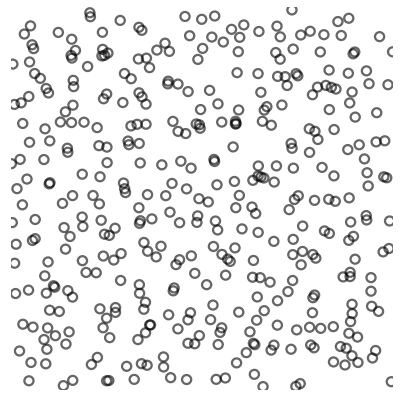
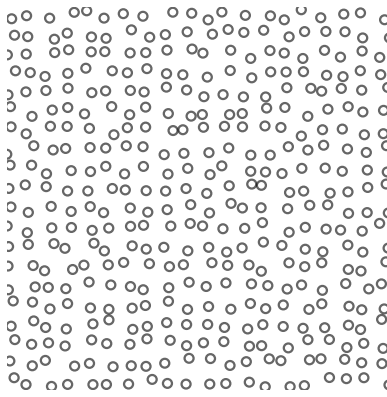
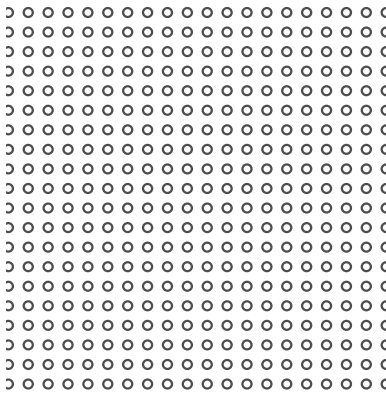
      var posX = width / tileCount * gridX;
      var posY = height / tileCount * gridY;

      var shiftX = random(-mouseX, mouseX) / 20;
      var shiftY = random(-mouseY, mouseY) / 20;

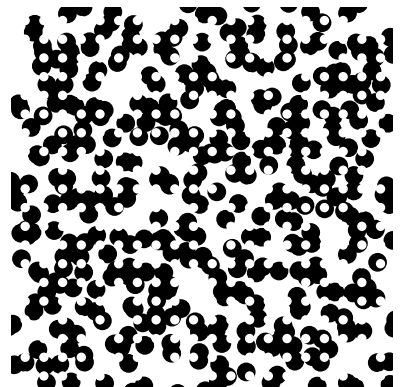
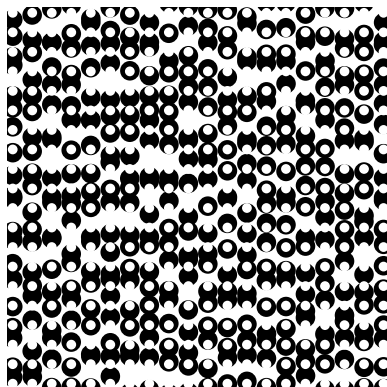
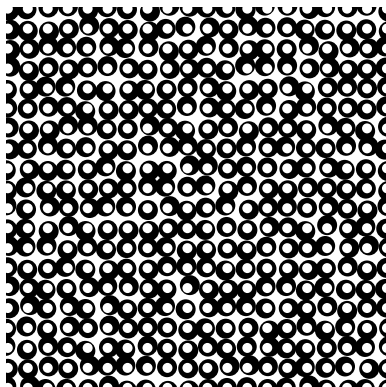
      ellipse(posX + shiftX, posY + shiftY,
              mouseY / 15, mouseY / 15);
    }
  }
}
```

Mouse: Position x: Circle position
 Position y: Circle size
 Left click: New random values of circle position
Keys: S: Save image

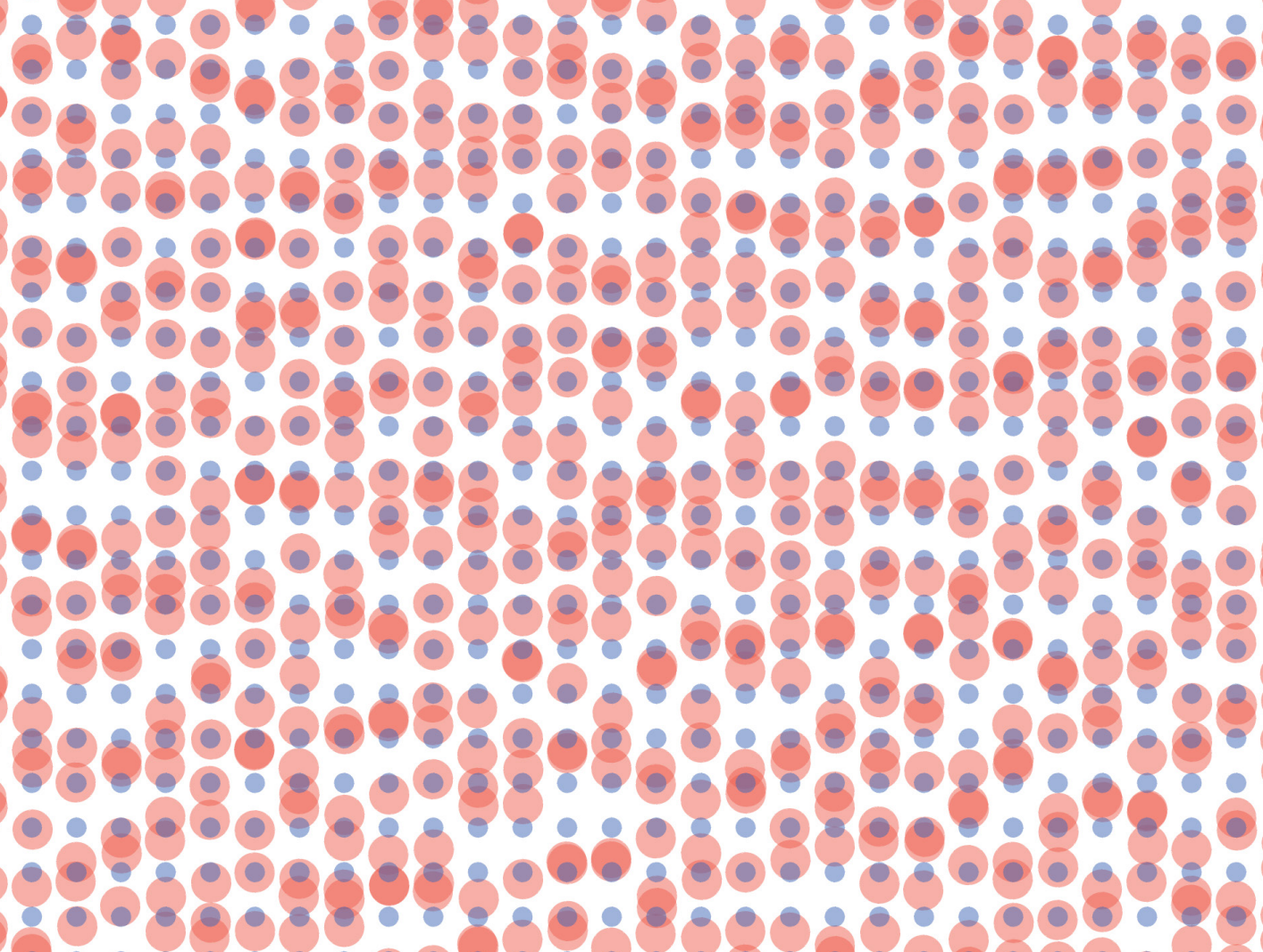
- 1 The origin of the coordinate system is shifted by half a tile width and height to the right and down so the circles are located in the center of their tiles.
- 2 The higher the x value of the mouse position `mouseX`, the greater the range of values for the random numbers.
- 3 Adding the values `shiftX` and `shiftY` to the grid coordinates `posX` and `posY` results in the final shifted positions of the circles.



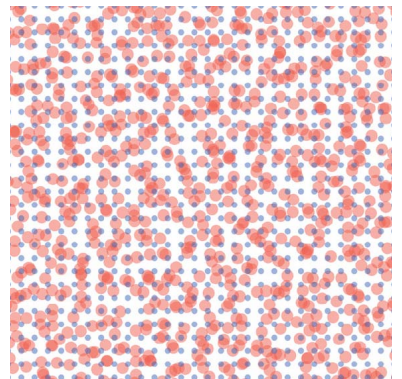
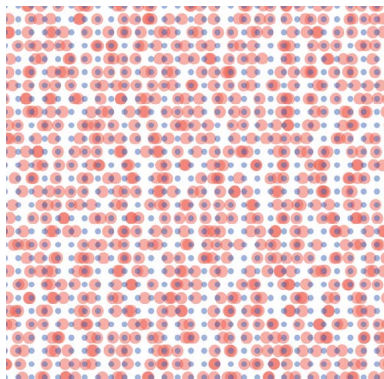
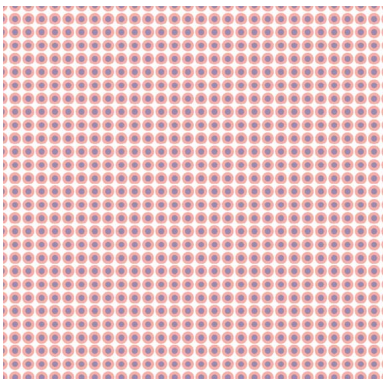
→ P_2_1_2_01 Transparent circles move in the horizontal grid according to the mouse position. The vertical mouse position determines the size of the circles.



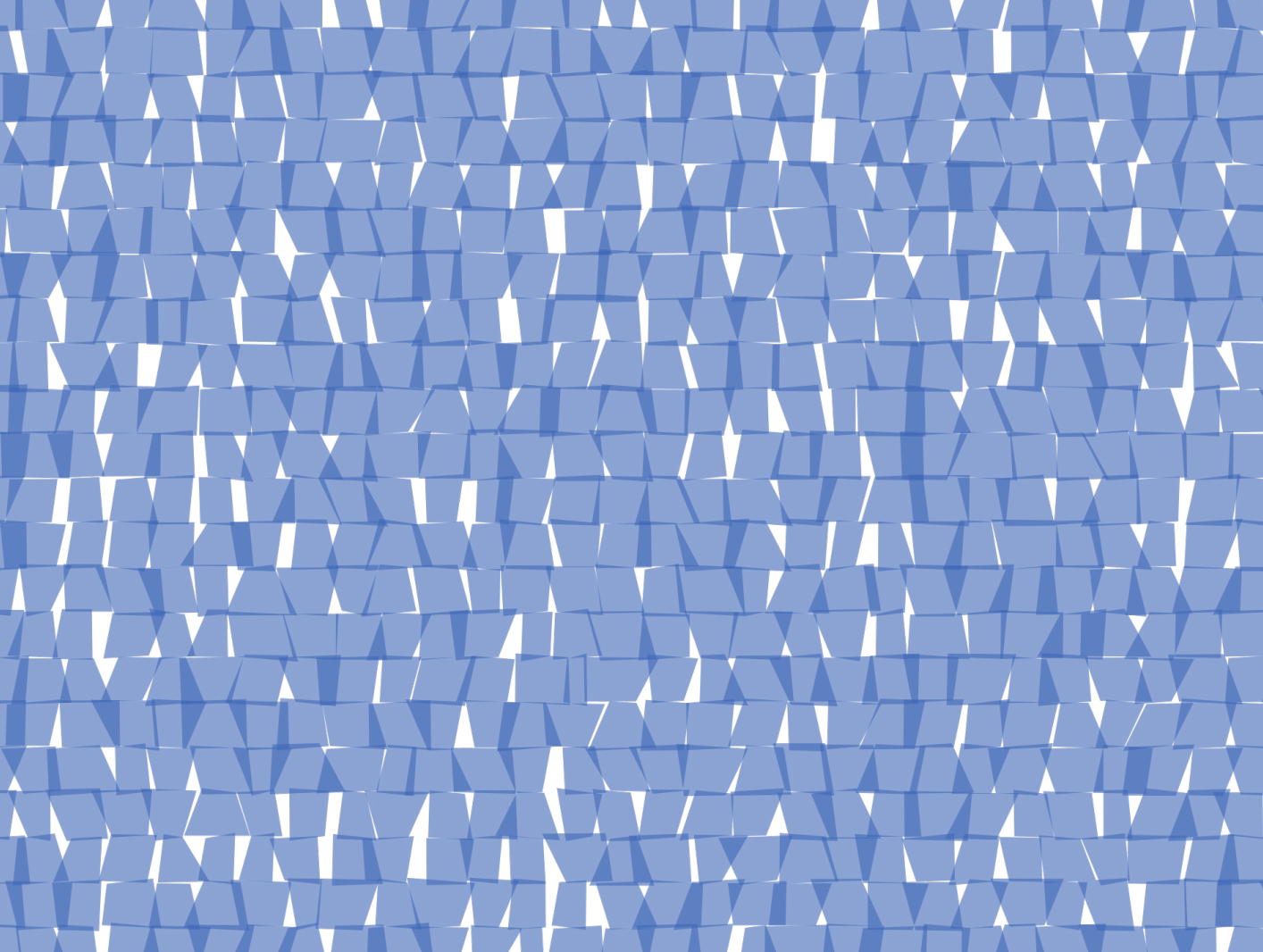
→ P_2_1_2_02 The shifting black circles are situated behind a dense grid of white circles.



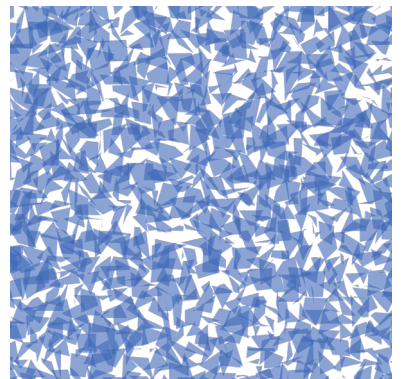
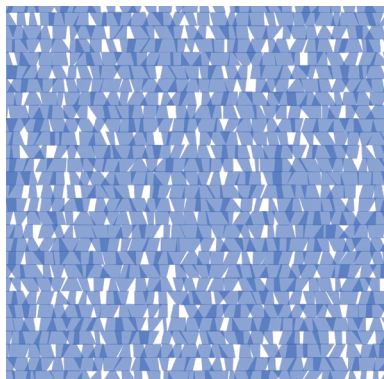
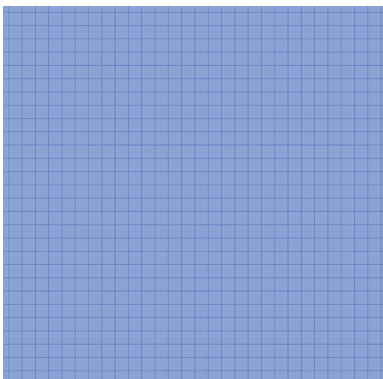
→ P_2_1_2_02 A color variation with transparent circles.



→ P_2_1_2_02 In this row, only the violet circles are offset, while the pink ones remain fixed to the grid.



→ P_2_1_2_04 Only the corners of the elements are shifted here, not the elements themselves.



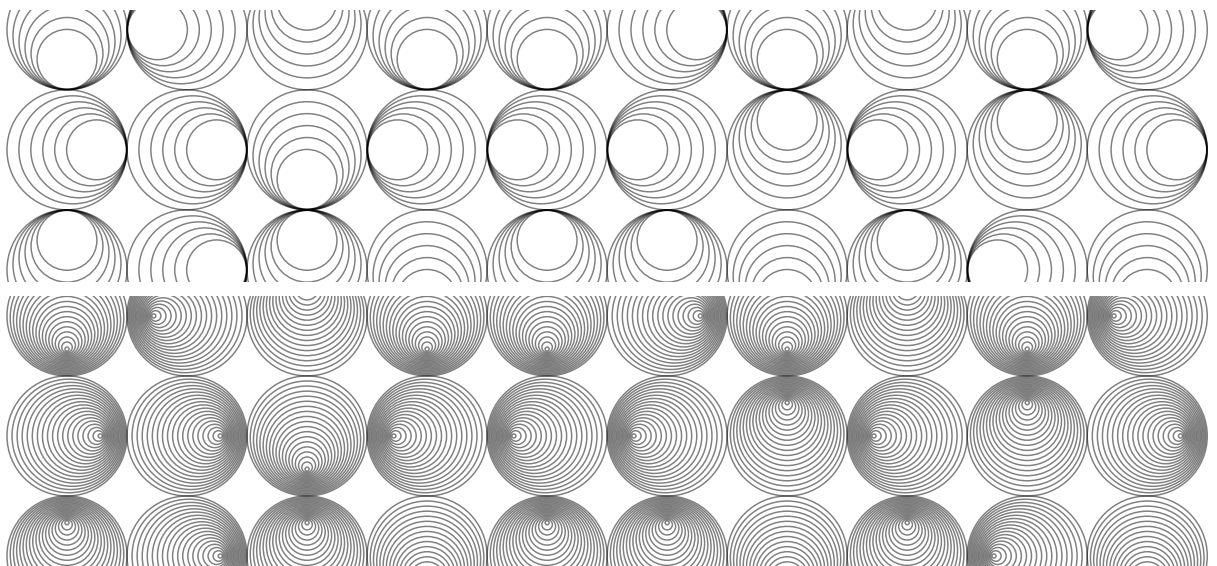
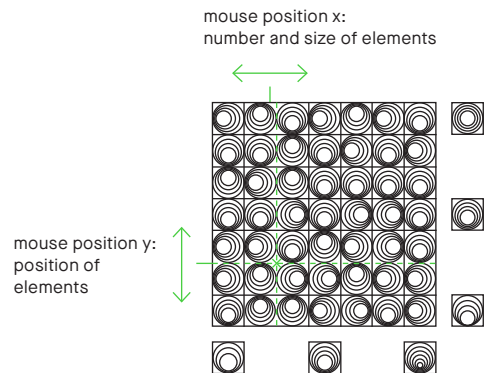
→ P_2_1_2_04 The overlaps and gaps here reveal the grid before it gradually becomes unrecognizable.

P.2.1.3 Complex modules in a grid

Even more interesting is the nesting of several forms into a complex module. In this illustration, four different clusters of ellipses increase in size in opposite directions and are positioned alternately on the grid. The varied diameters and transparencies of the individual ellipses create an illusion of depth.

→ P_2_1_3_01

The module filling the grid consists of a stack of circles that become smaller and smaller and move in one direction (top, bottom, right, or left) that is decided randomly. The number of circles corresponds to mouse position x, the movement to mouse position y.



→ P_2_1_3_01 Each module in the grid consists of several circles. The mouse position defines their number, size, and position.


```

function draw() {
  ...
  circleCount = mouseX / 30 + 1;
  endSize = map(mouseX, 0, max(width, mouseX),
                tileWidth / 2, 0);
  endOffset = map(mouseY, 0, max(height, mouseY),
                  0, (tileWidth - endSize) / 2);

  for (var gridY = 0; gridY <= tileCountY; gridY++) {
    for (var gridX = 0; gridX <= tileCountX; gridX++) {
      push();
      translate(tileWidth * gridX, tileHeight * gridY);
      scale(1, tileHeight / tileWidth);

      var toggle = int(random(0, 4));
      if (toggle == 0) rotate(-HALF_PI);
      if (toggle == 1) rotate(0);
      if (toggle == 2) rotate(HALF_PI);
      if (toggle == 3) rotate(PI);

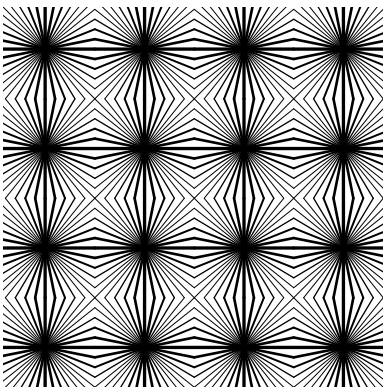
      // draw module
      for (var i = 0; i < circleCount; i++) {
        var diameter = map(i, 0, circleCount,
                          tileWidth, endSize);
        var offset = map(i, 0, circleCount, 0, endOffset);
        ellipse(offset, 0, diameter, diameter);
      }
      pop();
    }
  }
}

```

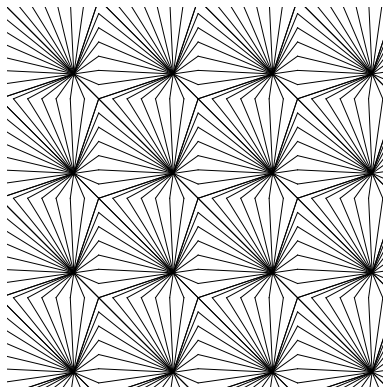
Mouse: Position x: Number and size of circles
 Position y: Position of circles
 Left click: New random value for position

Keys: S: Save image

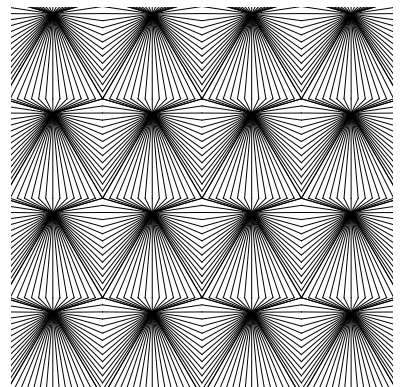
- 1 The mouse position defines `circleCount` in a module, the size of the circles, and the offset of the last circle.
- 2 The varying rotation of the modules is easier to manage when, before drawing a module, the origin of the coordinate system is temporarily moved to the position where it will be drawn. The command `push()` ensures that the current state of the coordinate system is saved before the origin is moved using `translate()`.
- 3 The random number `toggle` decides between the four rotation directions. `random(0, 4)` generates a number between 0 and 3.999; so, rounded off, the values 0, 1, 2, and 3. The radian angle `HALF_PI` is a rotation of 90°.
- 4 The module is constructed by drawing the circles in succession. The value of `diameter` ranges between `tileWidth` and the previously calculated `endSize`. The value `offset` comprises the shift from the center. The circles become smaller and smaller, thereby shifting farther and farther to the right.
- 5 Finally, the previously saved state of the coordinate system is restored using `pop()`.



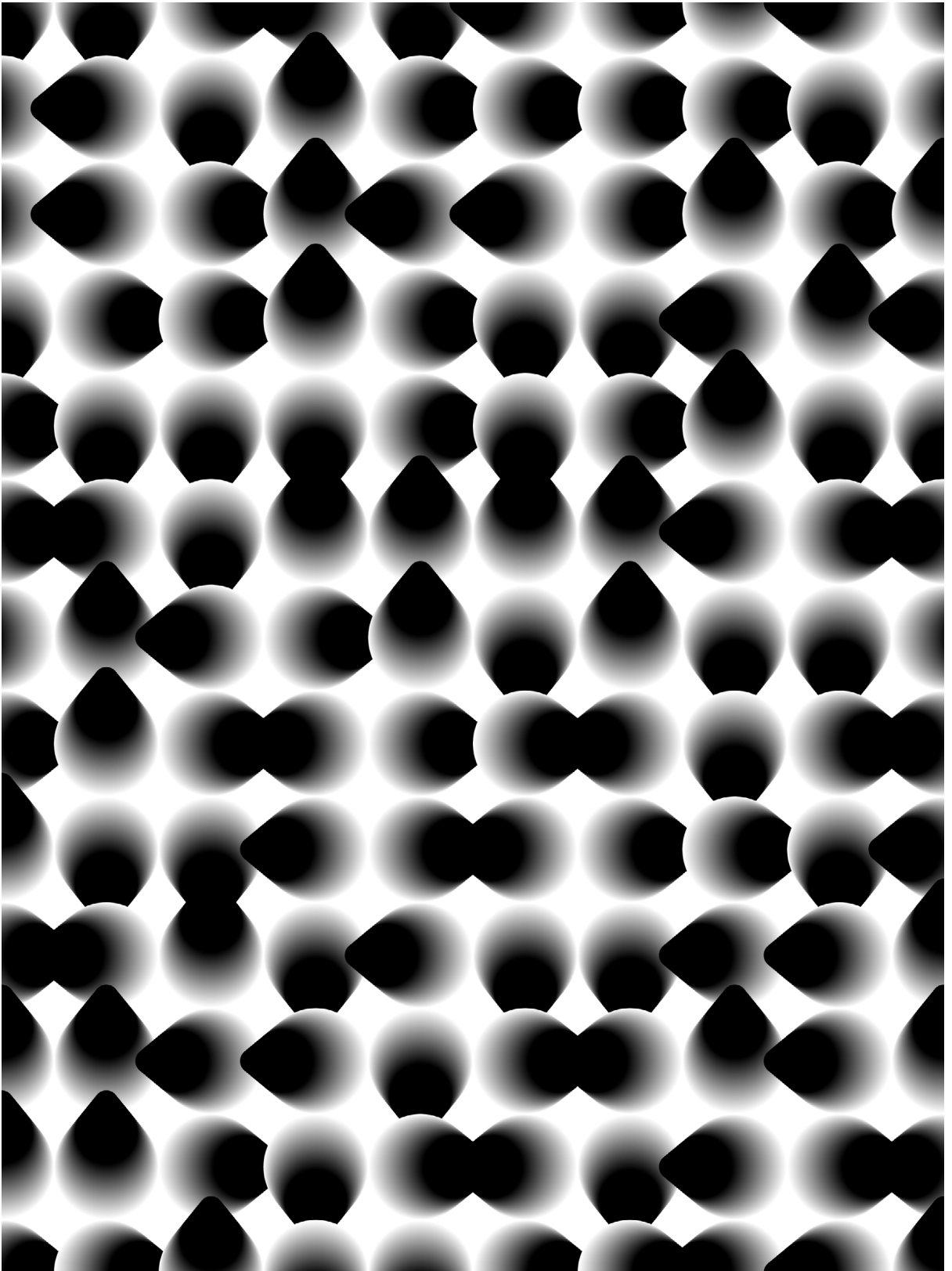
→ P_2_1_3_02 Complex modules made of straight lines. The consolidation of the lines generates new forms.



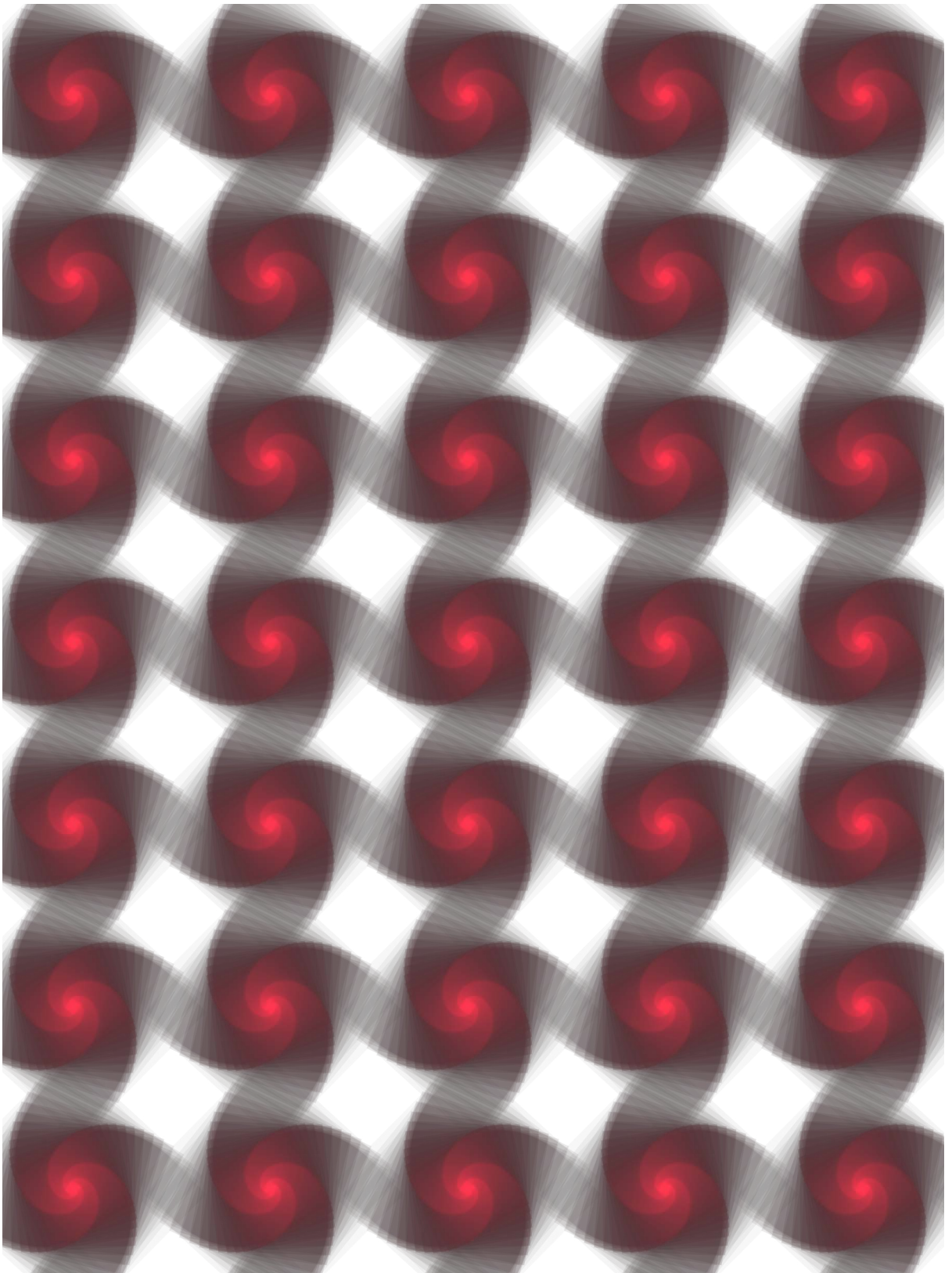
→ P_2_1_3_02 The point toward which the lines converge can be shifted diagonally with the mouse.



→ P_2_1_3_02 Consolidation easily creates a 3D effect.



→ P_2_1_3_05 This module consists of increasingly smaller and darker circles.



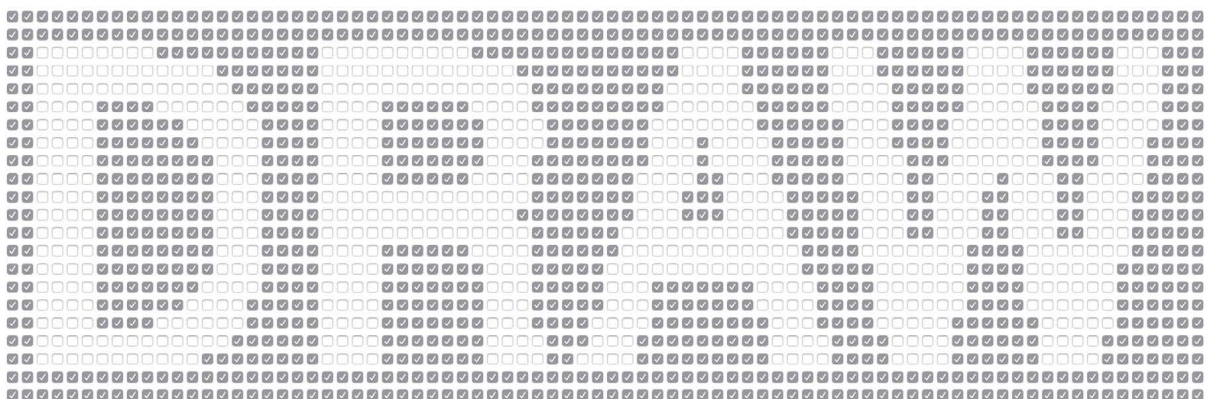
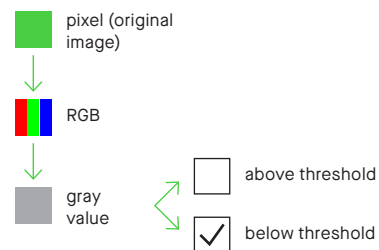
→ **P_2_1_3_04** This module displays a similar principle to the one opposite but with rotating squares. The squares are hardly recognizable as colored and transparent.

P.2.1.4 Checkboxes in a grid

This is where the controls' wildest dreams come true: they finally become visual design elements. Buttons, checkboxes, and sliders are used as raster elements in the browser, serving as pure visualization elements that become intriguing when grouped and arranged together.

→ P_2_1_4_01

Each pixel of an input image is analyzed and displayed as a checkbox. The brightness of the pixel determines whether it is checked off or not. If the brightness is above a specified threshold, the checkmark will not be set. Those checkboxes therefore will appear brighter.



→ P_2_1_4_01 The appearance of the checkboxes varies from browser to browser but can be adjusted via style sheets.

```

<html>
...
<body>
  <div id="container"></div>
  ...
</body>
</html>

```

1

```

function setup() {
  noCanvas();
  ...
  for (var y = 0; y < rows; y++) {
    for (var x = 0; x < cols; x++) {
      var box = createCheckbox();
      box.style('display', 'inline');
      box.parent('container');
      boxes.push(box);
    }
    var linebreak = createSpan('<br/>');
    linebreak.parent('mirror');
  }

  slider = createSlider(0, 255, 0);
}

```

2

3

4

5

```

function draw() {
  for (var y = 0; y < img.height; y++) {
    for (var x = 0; x < img.width; x++) {
      var c = color(img.get(x, y));
      var bright = (red(c) + green(c) + blue(c)) / 3;

      var threshold = slider.value();

      var checkIndex = x + y * cols;

      if (bright > threshold) {
        boxes[checkIndex].checked(false);
      } else {
        boxes[checkIndex].checked(true);
      }
    }
  }
}

```

6

7

8

9

Keys: 1-3: Change image
Slider: Set grayscale threshold

1 The index.html document provides a `<div>` element with the `id container`, to which the checkboxes will be added later.

2 Since the elements land directly in the HTML document, the standard drawing canvas can be removed.

3 Create checkbox with `createCheckbox()`. These can be positioned with `style()` style properties, and `parent()` inserts it into the HTML element `container`.

4 A line break is added at the end of each line.

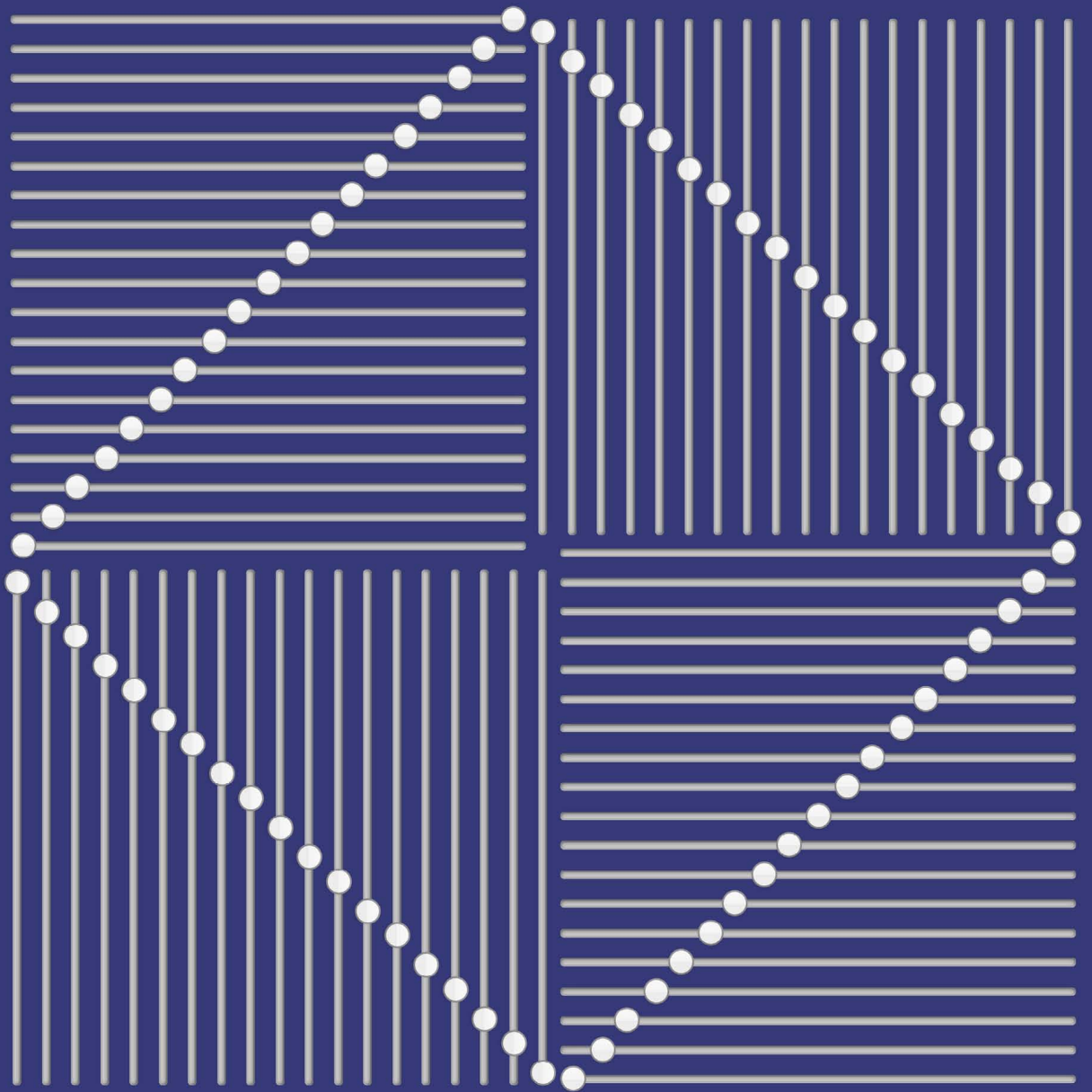
5 The slider can later be used to control the brightness level threshold. A range of values from 0 to 255 is therefore useful. The starting value is 0.

6 When converting the image into checkboxes, the individual pixels of the image must be converted into a value, in this case, `bright`. For grayscale images, a simple mean value calculation is sufficient. For color images, a slightly more complex calculation should be used.
→ P.4.3.1 Graphics from pixel values

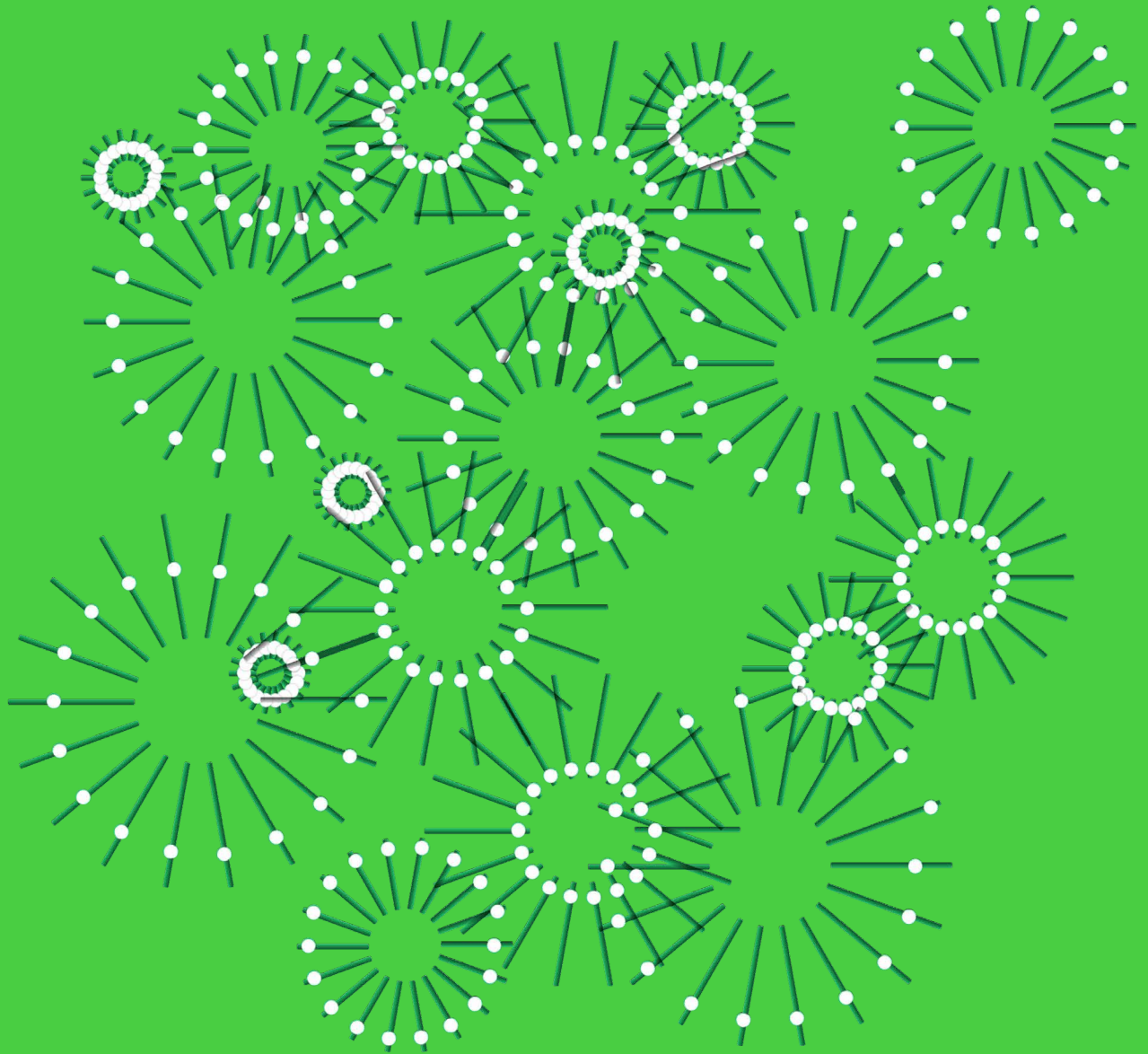
7 The `value()` function retrieves the value currently set by the slider.

8 The index of the checkbox associated with this pixel is calculated from `x`, `y`, and the number of columns, `cols`.

9 If the brightness of the pixel is above the set `threshold`, the check mark is removed; otherwise it is set.



→ P_2_1_4_03 Many sliders arranged geometrically.



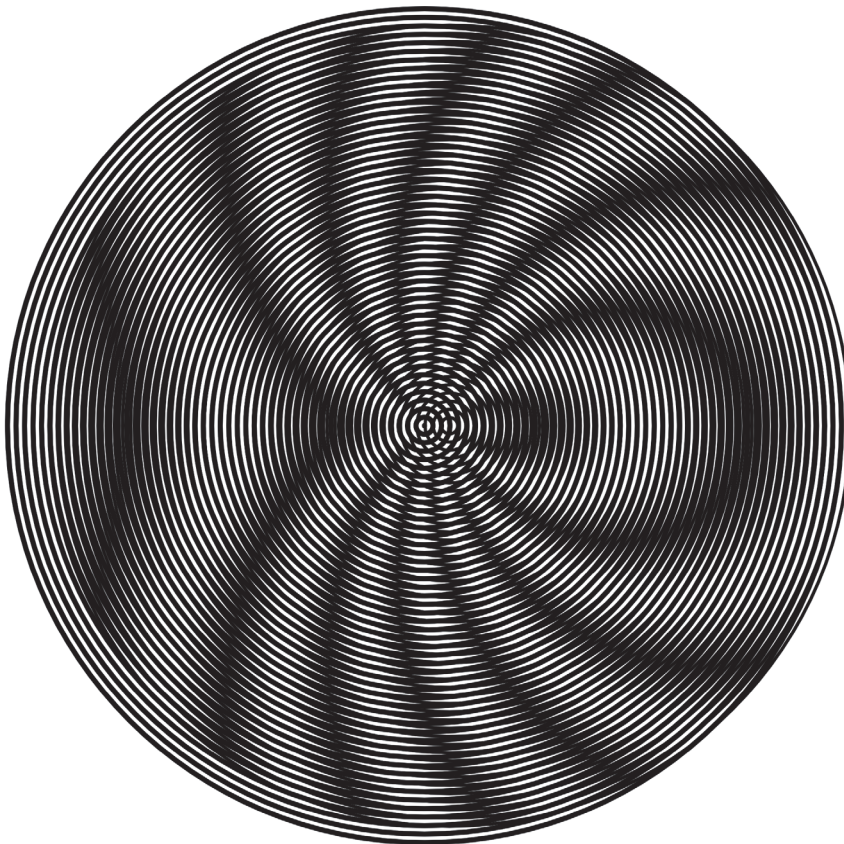
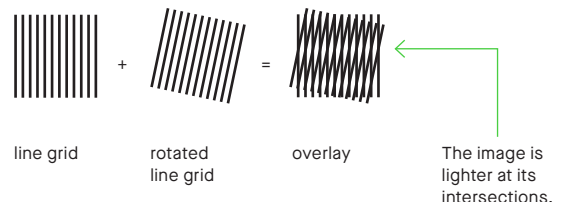
→ P_2_1_4_04 These sliders automatically stay in motion.

P.2.1.5 From grid to moiré

A moiré, which is usually considered a mistake in printing technology, is desirable here. By laying one graphic grid over another identical grid and moving it, you can generate unexpected optical illusions that you can change in real time.

→ P_2_1_5_01

The moiré effect occurs when two or more fine grid structures overlap to form a coarser pattern. This happens because the overall picture shines brighter at all crossing points, because more of the white background can be seen next to these intersections.



→ P_2_1_5_01 Curved raster elements usually create particularly interesting overlays.

```
function draw() {
  ...
  strokeWeight(3);
  overlay();

  var x = map(mouseX, 0, width, -50, 50);
  var a = map(mouseX, 0, width, -0.5, 0.5);
  var s = map(mouseY, 0, height, 0.7, 1);

  if (drawMode == 2) translate(x, 0);
  if (drawMode == 1) rotate(a);
  scale(s);

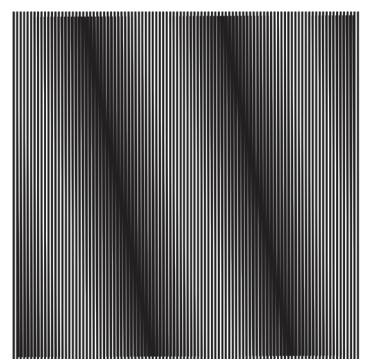
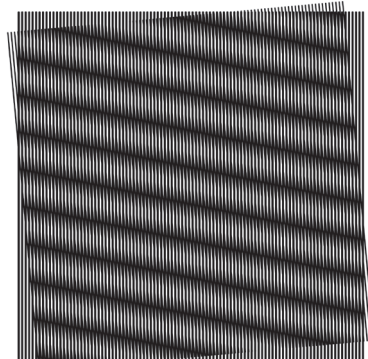
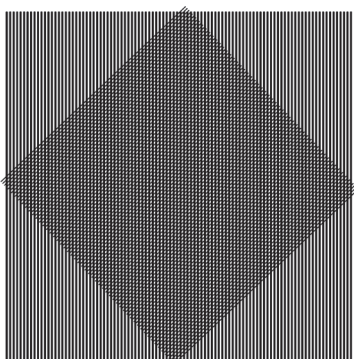
  strokeWeight(2);
  overlay();
}
```

```
function overlay() {
  var w = width - 100;
  var h = height - 100;

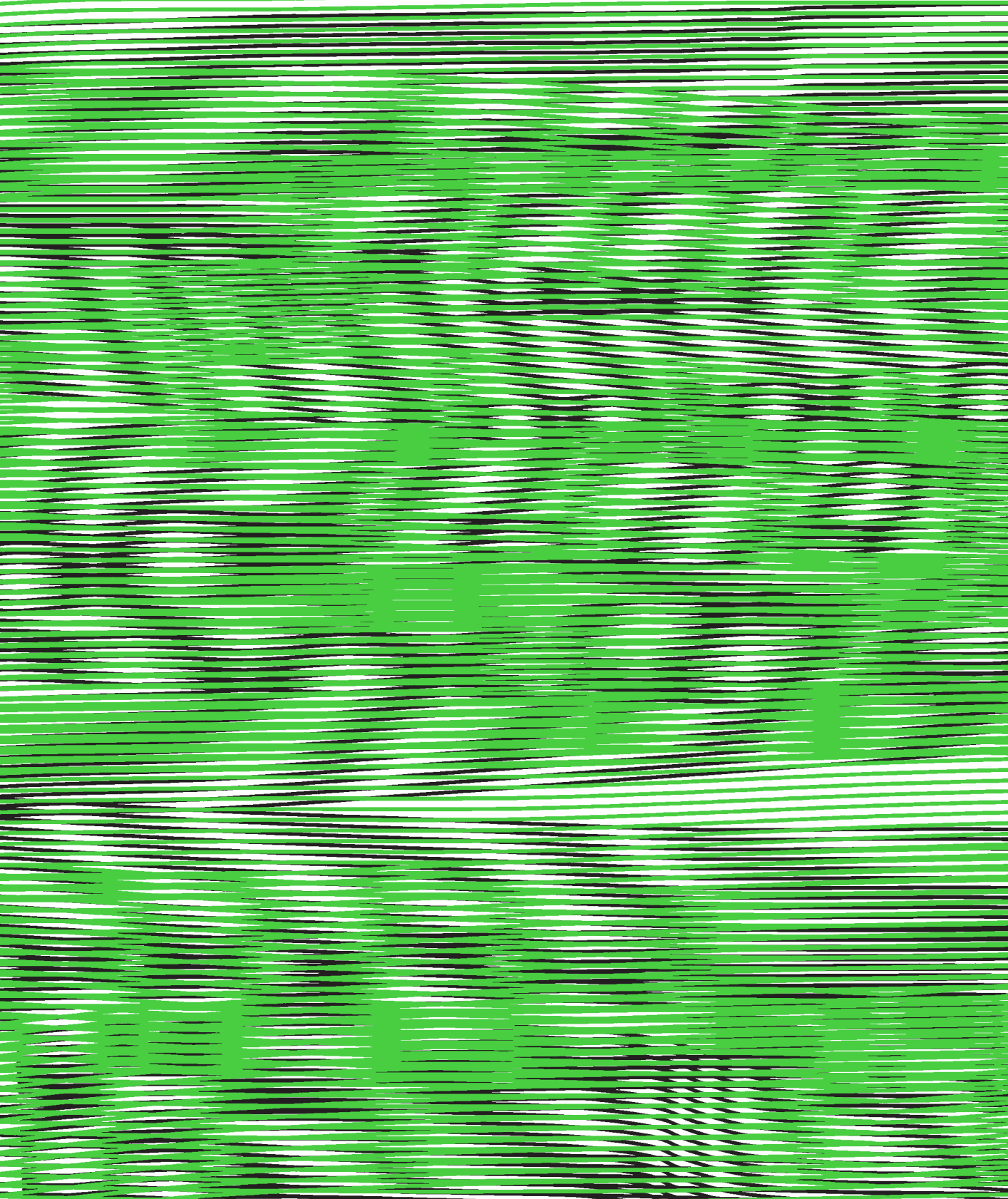
  if (drawMode == 1) {
    for (var i = -w / 2; i < w / 2; i += 5) {
      line(i, -h / 2, i, h / 2);
    }
  }
  if (drawMode == 2) {
    for (var i = 0; i < w; i += 10) {
      ellipse(0, 0, i);
    }
  }
}
```

Mouse: Position x: Rotate or move overlay
 Position y: Scale overlay
Keys: 1-2: Change drawMode
 S: Save image

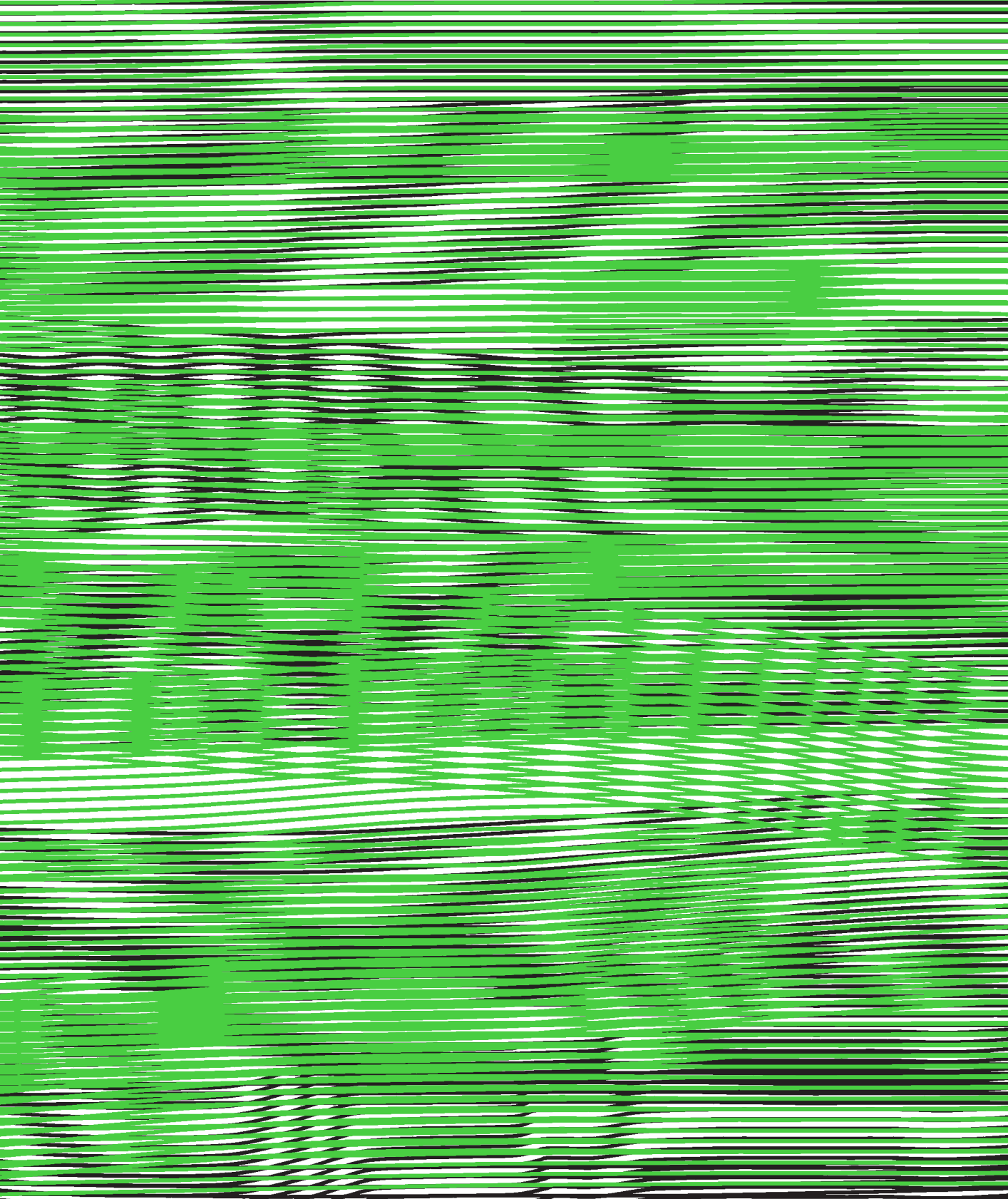
- 1 The background graphic is drawn. Since this graphic must be generated a second time, this process has been swapped out to the `overlay()` function.
- 2 The mouse position is used to calculate the corresponding values for the translation `x`, the rotation `a`, and the scaling `s`.
- 3 Depending on the `drawMode`, the coordinate system is either moved horizontally or rotated. Scaling is done in both cases.
- 4 The overlay graphic is drawn with a slightly different stroke width.
- 5 The graphic is created in the function `overlay()`.
- 6 In drawMode 1, a series of lines.
- 7 In drawMode 2, gradually increasing circles.



→ **P_2_1_5_01** The moiré effect becomes stronger as the two grids get closer in terms of rotation and scaling.



→ P_2_1_5_04 Freely drawn lines, drawn numerous times, parallel to each other and overlaid.

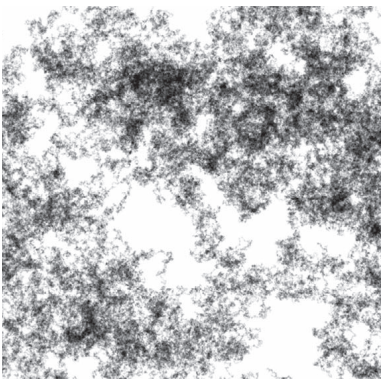
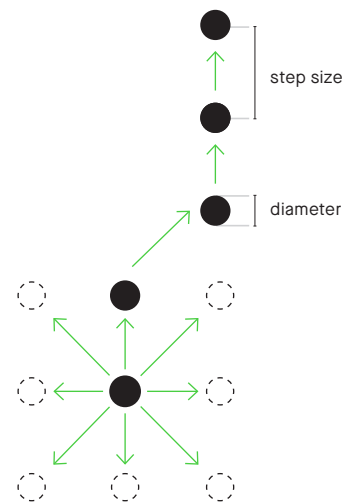
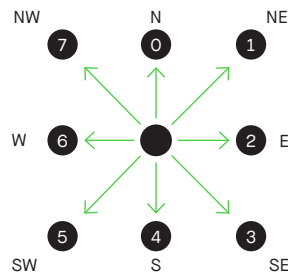


P.2.2.1 Dumb agents

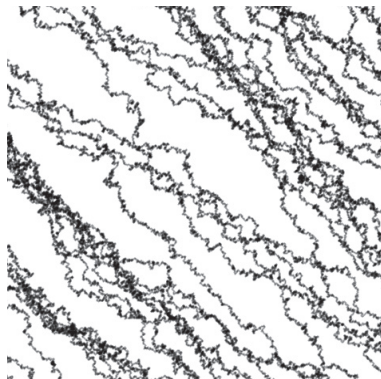
Instead of being rigidly embedded in a grid, the pixel now becomes an agent and can move freely based on different behavioral patterns. With each step the agent advances according to one of eight directions, leaving a trail behind it. It pursues its mission and never gives up.

→ P_2_2_1_01

During each drawing operation, one of the eight possible directions is randomly selected for the next step. Steps are made by adding or subtracting a predetermined value (step size) to the current position's coordinates. The circle is finally drawn at the new position.



→ P_2_2_1_01 The longer the agents migrate, the denser the cloud structure becomes.



→ P_2_2_1_02 The agents' direction of movement is restricted here.



→ P_2_2_1_02 In this variation, the diameters of the circles are larger and sometimes colored blue.


```

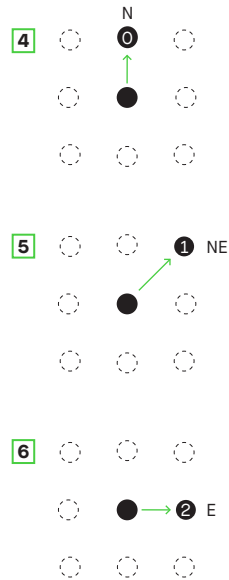
1  var NORTH = 0;
    var NORTHEAST = 1;
    var EAST = 2;
    var SOUTHEAST = 3;
    var SOUTH = 4;
    var SOUTHWEST = 5;
    var WEST = 6;
    var NORTHWEST = 7;
    ...
2  var stepSize = 1;
    var diameter = 1;

function draw() {
3    for (var i = 0; i <= mouseX; i++) {
        direction = int(random(0, 8));
4
5        if (direction == NORTH) {
            posY -= stepSize;
6        } else if (direction == NORTHEAST) {
            posX += stepSize;
            posY -= stepSize;
7        } else if (direction == EAST) {
            posX += stepSize;
8        } else if (direction == SOUTHEAST) {
            posX += stepSize;
            posY += stepSize;
        } else if (direction == SOUTH) {
            posY += stepSize;
        } else if (direction == SOUTHWEST) {
            posX -= stepSize;
            posY += stepSize;
        } else if (direction == WEST) {
            posX -= stepSize;
        } else if (direction == NORTHWEST) {
            posX -= stepSize;
            posY -= stepSize;
        }
7
        if (posX > width) posX = 0;
        if (posX < 0) posX = width;
        if (posY < 0) posY = height;
        if (posY > height) posY = 0;
8
        ellipse(posX + stepSize / 2, posY + stepSize / 2,
                diameter, diameter);
    }
}

```

Mouse: Position x: Speed of the image construction
Keys: DEL: Clear drawing canvas
 S: Save image

- 1 Eight constants, each with a different numerical value, are defined.
- 2 The step size and diameter of the agents can be set by changing the values of `stepSize` and `diameter`.
- 3 The command `random(0,8)` creates a random number between 0.000 and 7.999. When rounded off, this results in a value between 0 and 7. This, in turn, is stored in `direction` and determines the next step.



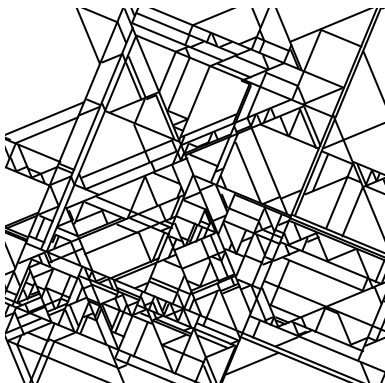
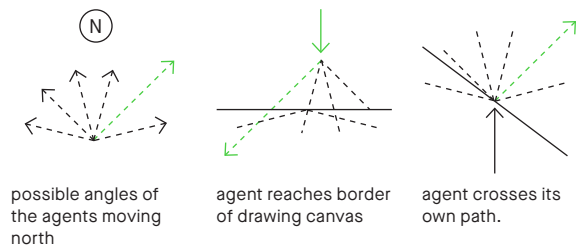
- 4
- 5
- 6
- 7 When the agent's current position extends beyond the right border of the display window, `posX` is set at 0. This causes the agent to continue its path on the opposite side.
- 8 A transparent circle is drawn in the new position. Half the increment size, `stepSize/2`, is added to this so the circle is not cut off at the border of the display window.

P.2.2.2 Intelligent agents

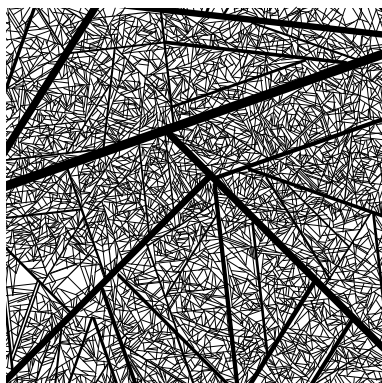
Behavioral patterns are now far more complex and subject to precise conditions. The agent easily goes astray when it crosses its own path. When it reaches the border of the display window, it changes direction. The straight lines, which are drawn between two points of intersection, change their color and stroke value depending on the traveled distance.

→ P_2_2_2_01

The agent always moves in one of the cardinal directions: north, east, south, or west. It can, however, choose from several possible angles, whereby right angles are not possible. When the agent reaches the border of the drawing canvas, it turns around and randomly selects one of the possible angles. When it crosses its own path, it maintains its general direction but selects a new angle.



→ P_2_2_2_01 By pressing key R, lines are drawn without tracks in a PDF. The process is ended by pressing key E.



→ P_2_2_2_02 A line's value depends on its distance.



→ P_2_2_2_02 The lines here are colored according to their length (key 2).

```

function draw() {
  var speed = int(map(mouseX, 0, width, 0, 20));
  for (var i=0; i<=speed; i++) {

    strokeWeight(1);
    stroke(180, 0, 0);
    point(posX, posY);

    posX += cos(radians(angle)) * stepSize;
    posY += sin(radians(angle)) * stepSize;

    reachedBorder = false;

    if (posY <= 5) {
      direction = SOUTH;
      reachedBorder = true;
    } else if (posX >= width - 5) {
      direction = WEST;
      reachedBorder = true;
    }
    ...

    loadPixels();
    var currentPixel = get(floor(posX), floor(posY));
    if ((currentPixel[0]!=255 && currentPixel[1]!=255 &&
        currentPixel[2]!=255) || reachedBorder) {

      angle = getRandomAngle(direction);

      var distance = dist(posX, posY,
                          posXcross, posYcross);
      if (distance >= minLength) {
        strokeWeight(3);
        stroke(0, 0, 0);
        line(posX, posY, posXcross, posYcross);
      }

      posXcross = posX;
      posYcross = posY;
    }
  }
}

```

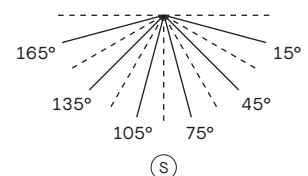
```

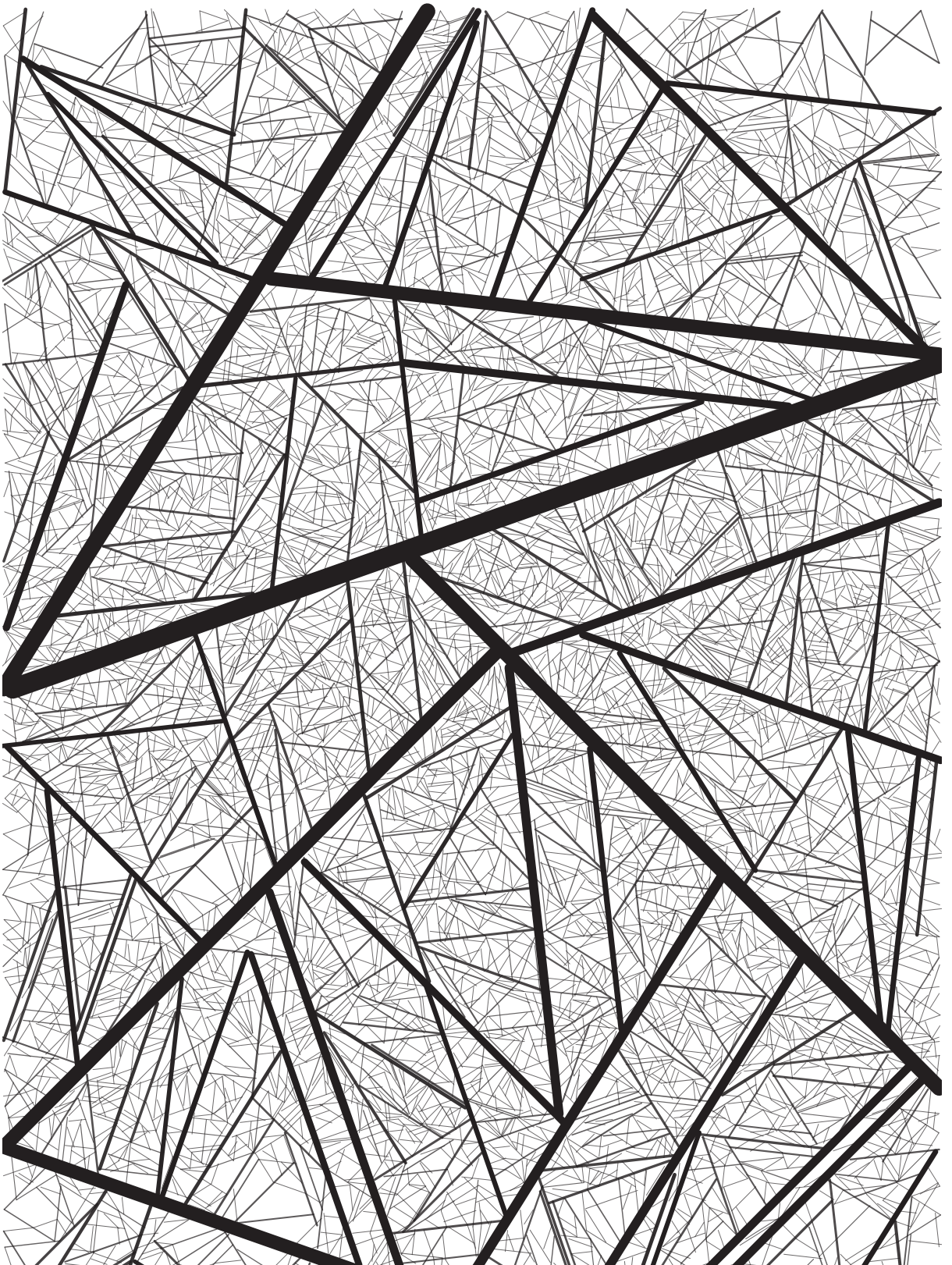
function getRandomAngle(currentDirection) {
  var a = (floor(random(-angleCount, angleCount)) +
    0.5) * 90 / angleCount;
  if (currentDirection == NORTH) return a - 90;
  if (currentDirection == EAST) return a;
  if (currentDirection == SOUTH) return a + 90;
  if (currentDirection == WEST) return a + 180;
  return 0;
}

```

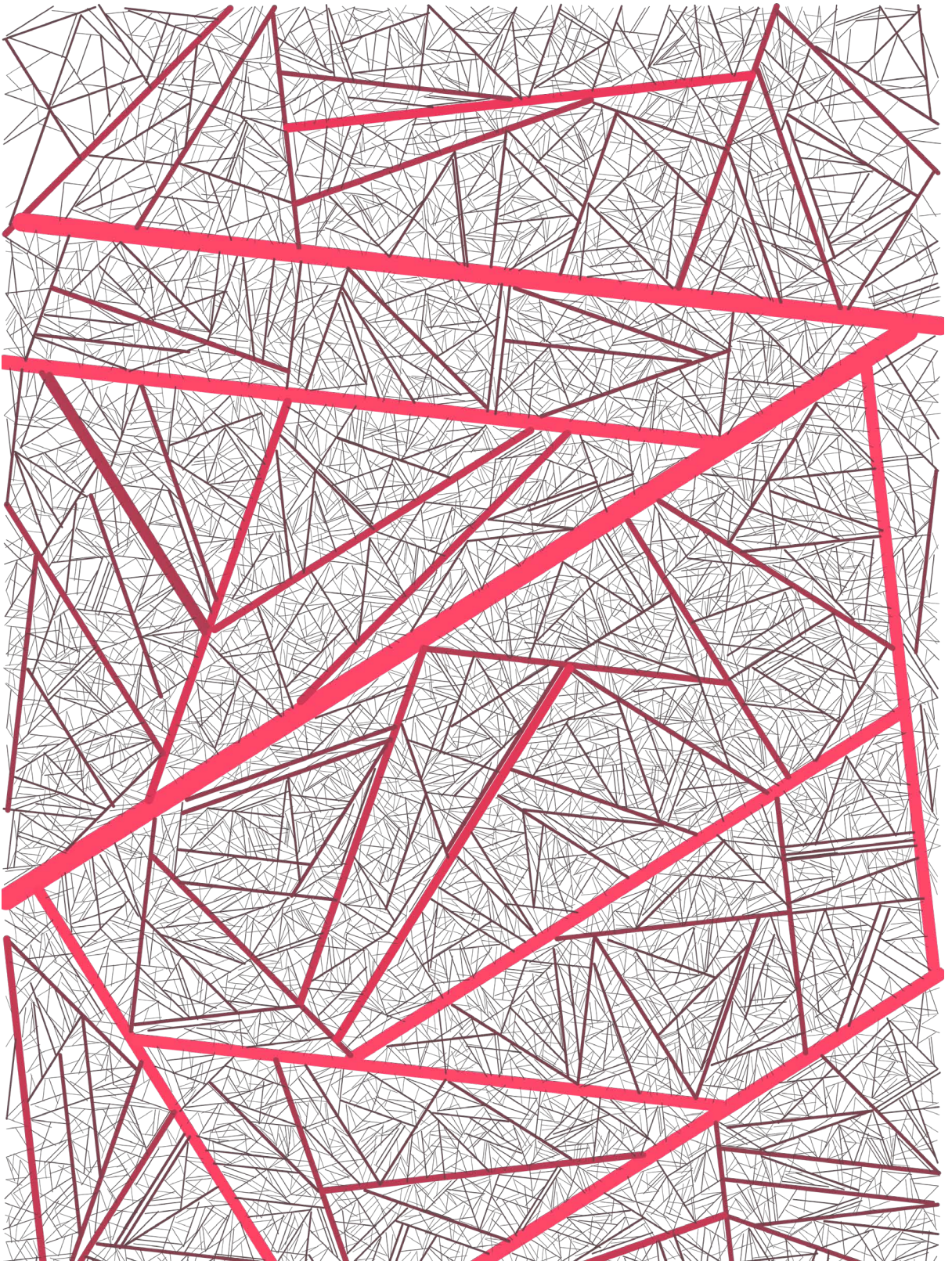
Mouse: Position x: Speed of the image construction
Keys: DEL: Clear canvas
 S: Save image

- 1 A point is drawn on the agent's current position (`posX, posY`). The point can become almost (or completely) invisible when its color matches the background color.
- 2 The agent takes a step. In doing so, its position is updated: `angle` defines the direction and `stepSize` the length of the step.
- 3 Now the agent is checked to determine if it has reached a border of the display window. If, e.g., it is only five pixels or fewer away from the upper border, the `direction` is changed to `SOUTH` and the variable `reachedBorder` set to `true`.
- 4 The function `get()` checks every time the agent moves to determine whether it is on a pixel that is not white. If this is the case or if the variable `reachedBorder` is `true`, then a new random angle step increment in the main direction is selected using the function `getRandomAngle()`.
- 5 When changing direction, a line is only drawn when the position of the last direction change (`posXcross, posYcross`) is at least as far away as the distance defined by `minLength`.
- 6 Finally, the current position is saved in the variables `posXcross` and `posYcross`.
- 7 The function `getRandomAngle()` randomly selects and returns one of the possible angles, which is dependent on the passed main direction, `currentDirection`. When `angleCount`, e.g., is 3 (i.e., three directions per quadrant) and `currentDirection` is `SOUTH`, then one of these angles is returned.





→ P_2_2_2_02 Longer lines are opaque in appearance, shorter ones transparent.



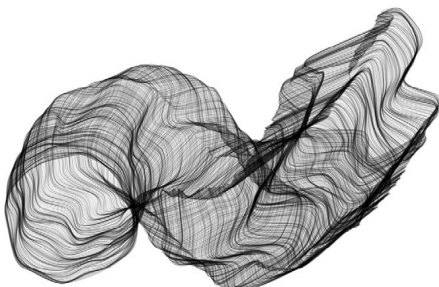
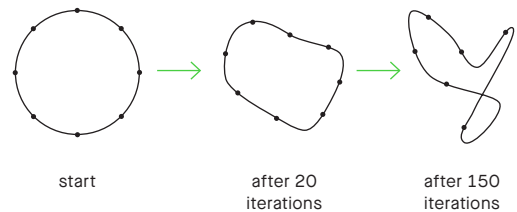
→ P_2_2_2_02 Fixed values are set for hue and saturation; brightness depends on the length of the lines.

P.2.2.3 Shapes from agents

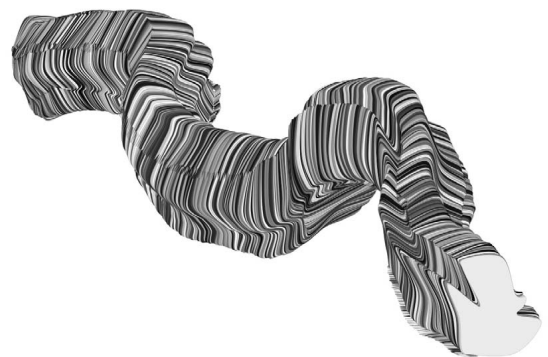
It is only through cooperative efforts that dumb agents become strong. A circle is the source shape in this example. The permutation rule is that every point on the circle is represented by a dumb agent and that movement of the points will cause the circle to gradually change its shape. This produces a surprisingly large repertoire of forms.

→ P_2_2_3_01

The calculation of the points on the circle produces the starting positions of the agents. A flexible curve firmly connects each agent with both its neighbors. The farther away the dumb agents move from their original positions, the more the circular organization dissolves.



→ P_2_2_3_01 The constantly changing shape always moves toward the mouse and can thereby be controlled by it.



→ P_2_2_3_01 The fill mode can be set using key 2 so that the form is colored with a random and changing gray value.


```
function setup() {
  ...
  centerX = width / 2;
  centerY = height / 2;
  var angle = radians(360 / formResolution);
  for (var i = 0; i < formResolution; i++) {
    x.push(cos(angle * i) * initRadius);
    y.push(sin(angle * i) * initRadius);
  }
  ...
}
```

```
function draw() {
  centerX += (mouseX - centerX) * 0.01;
  centerY += (mouseY - centerY) * 0.01;

  for (var i = 0; i < formResolution; i++) {
    x[i] += random(-stepSize, stepSize);
    y[i] += random(-stepSize, stepSize);
    // ellipse(x[i] + centerX, y[i] + centerY, 5, 5);
  }
  ...
  beginShape();
  curveVertex(x[formResolution - 1] + centerX,
             y[formResolution - 1] + centerY);

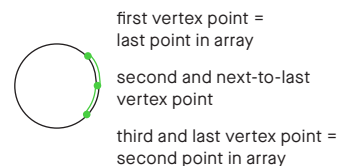
  for (var i = 0; i < formResolution; i++) {
    curveVertex(x[i] + centerX, y[i] + centerY);
  }
  curveVertex(x[0] + centerX, y[0] + centerY);

  curveVertex(x[1] + centerX, y[1] + centerY);
  endShape();
}
```

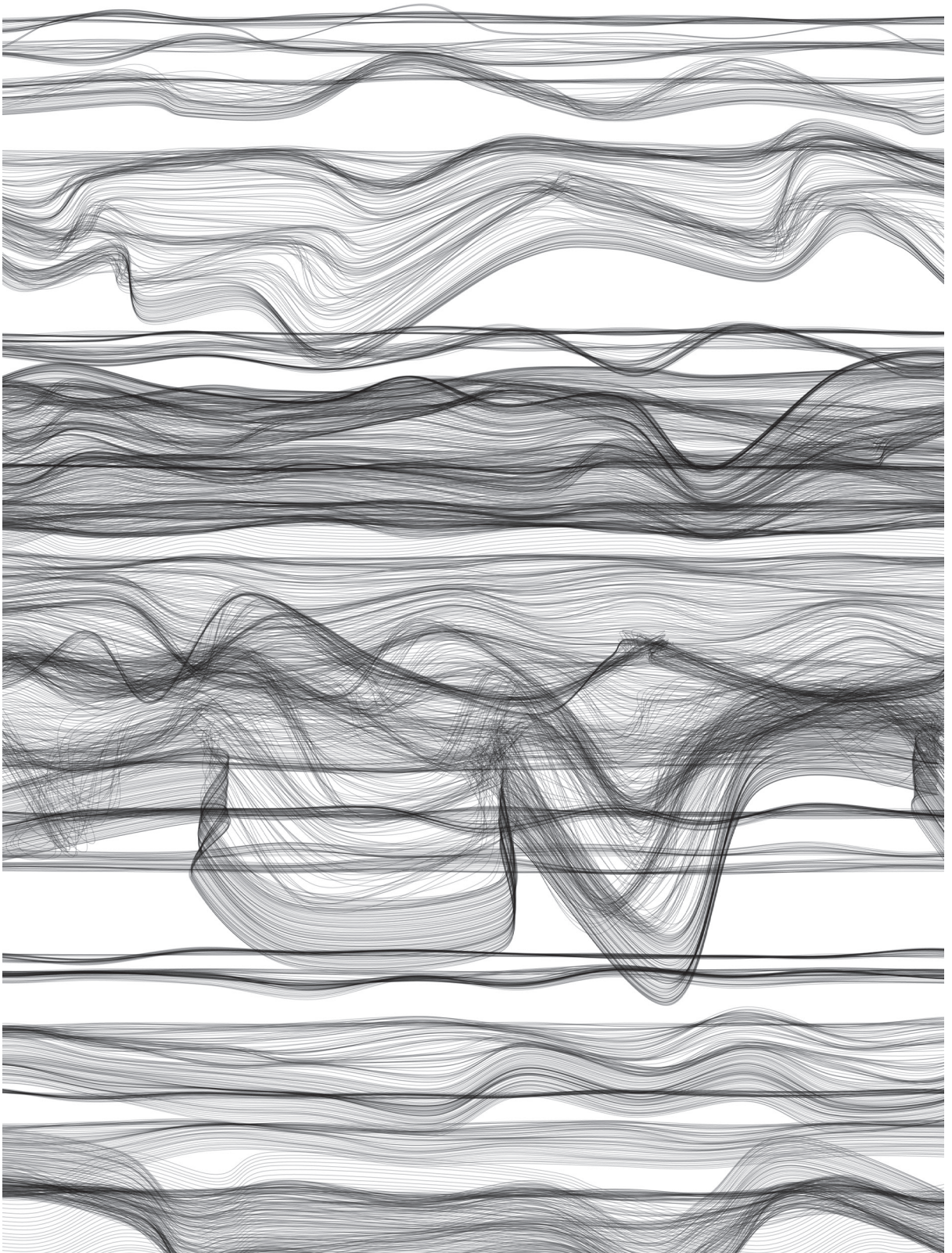
Mouse: Left click: New circle
 Position x/y: Movement direction

Keys: 1-2: Fill mode
 S: Save image

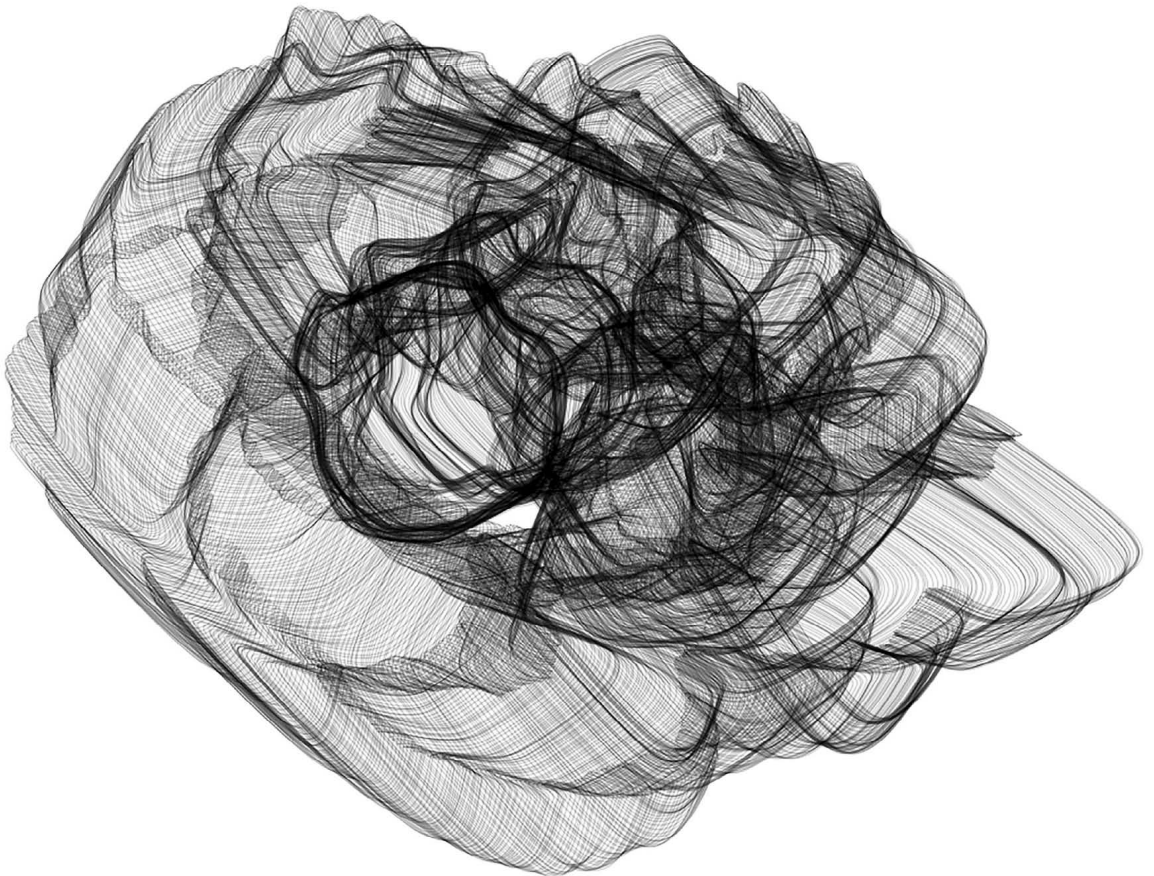
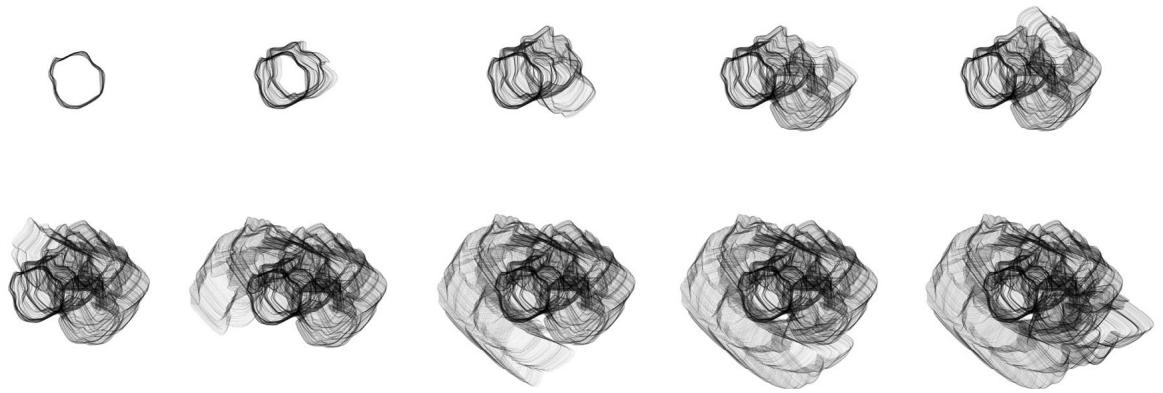
- 1 The agents are drawn later around the position (`centerX`, `centerY`). The initial starting point is the center of the display.
- 2 The agents' starting positions are calculated as points on the circle and saved in the arrays `x` and `y` by the array function `push()`.
 → P.1.1.2 Color spectrum in a circle
- 3 The mouse follows the position (`centerX`, `centerY`). With every frame the difference between the agent position and the mouse position is calculated, multiplied by a small value, and added again to this position.
- 4 The addition of random values between `-stepSize` and `stepSize` to the agents' current positions results in an up and down movement.
- 5 The agents' positions can also be visualized by adding the `ellipse` command.
- 6 Note that when drawing the form, the first and last points set with the `curveVertex()` serve as control points and are not drawn. These two control points ensure the circle is completed without any bends so the final circle is smooth.



- 7 If, e.g., `curveVertex` is replaced with `vertex`, straight connections are produced. Experiments with the fill color `fill` also result in interesting variations.



→ **P_2_2_3_02** In the second version of the program, a drawMode can be selected that initially creates a straight line, which then gradually deforms.



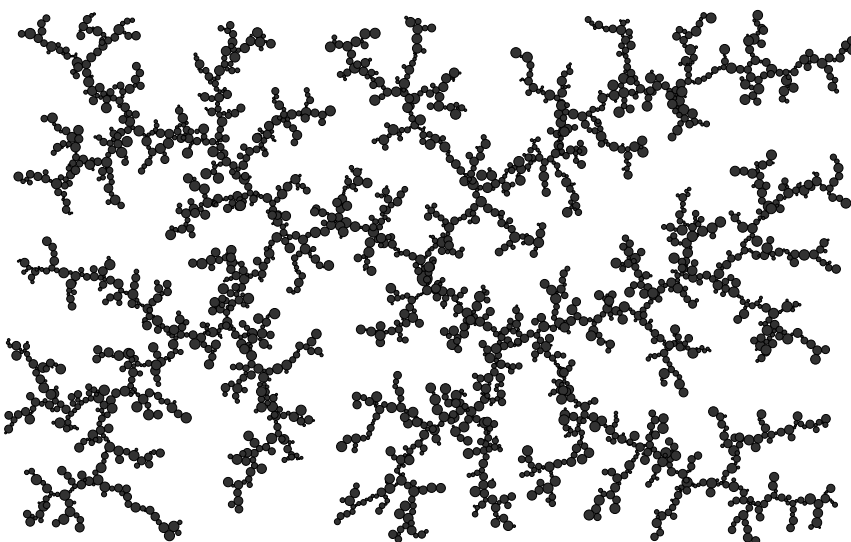
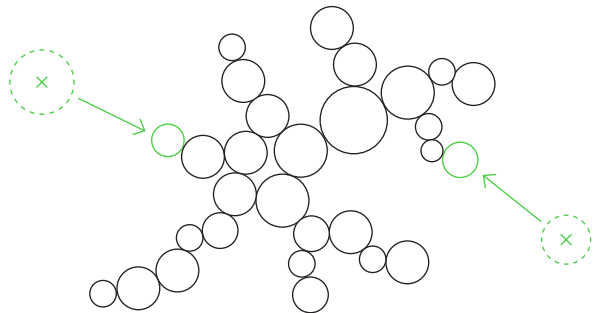
→ **P_2_2_3_02** Drawing with a changing form. When you click, a slightly deformed circular geometry appears. As you move the mouse, the form follows the position of the cursor and the distortion becomes increasingly apparent.

P.2.2.4 Growth structure from agents

A stable structure results from the convergence of multiple agents based on simple rules. Complex shapes are created out of simple propagation patterns such as: draw a new circle and position it as close as possible to its nearest neighbor. These types of algorithms also describe growth processes in plants and minerals.

→ P_2_2_4_01

In each frame, a new circle is generated at a random position and with a random radius (dashed circles). It is then determined which of the existing circles lies nearest to the new one. In the final step, the new circle docks with its closest neighbor via the shortest path.



→ P_2_2_4_01 Circles increasingly fill the area and an organic structure evolves.

```

function draw() {
  background(255);

1  var newR = random(1, 7);
    var newX = random(newR, width - newR);
    var newY = random(newR, height - newR);

    var closestDist = Number.MAX_VALUE;
    var closestIndex = 0;
2  for (var i = 0; i < currentCount; i++) {
      var newDist = dist(newX, newY, x[i], y[i]);
      if (newDist < closestDist) {
        closestDist = newDist;
        closestIndex = i;
      }
    }

3  // fill(230);
    // ellipse(newX, newY, newR * 2, newR * 2);
    // line(newX, newY, x[closestIndex], y[closestIndex]);

4  var angle = atan2(newY - y[closestIndex],
                    newX - x[closestIndex]);

    x[currentCount] = x[closestIndex] + cos(angle) *
                      (r[closestIndex] + newR);
    y[currentCount] = y[closestIndex] + sin(angle) *
                      (r[closestIndex] + newR);
    r[currentCount] = newR;
    currentCount++;

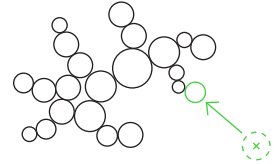
    for (var i = 0; i < currentCount; i++) {
      fill(50);
5      ellipse(x[i], y[i], r[i] * 2, r[i] * 2);
    }

6  if (currentCount >= maxCount) noLoop();
}

```

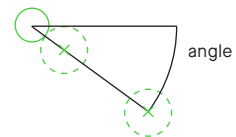
Keys: S: Save image

- 1 The radius `newR` and the position (`newX`, `newY`) for the circle are defined randomly.
- 2 The closest neighbor is searched in the `for` loop. All circles are processed one by one and their distance to the new circle is calculated. If this distance is smaller than all previous distances, a reference to this circle is saved in the variable `closestIndex`.

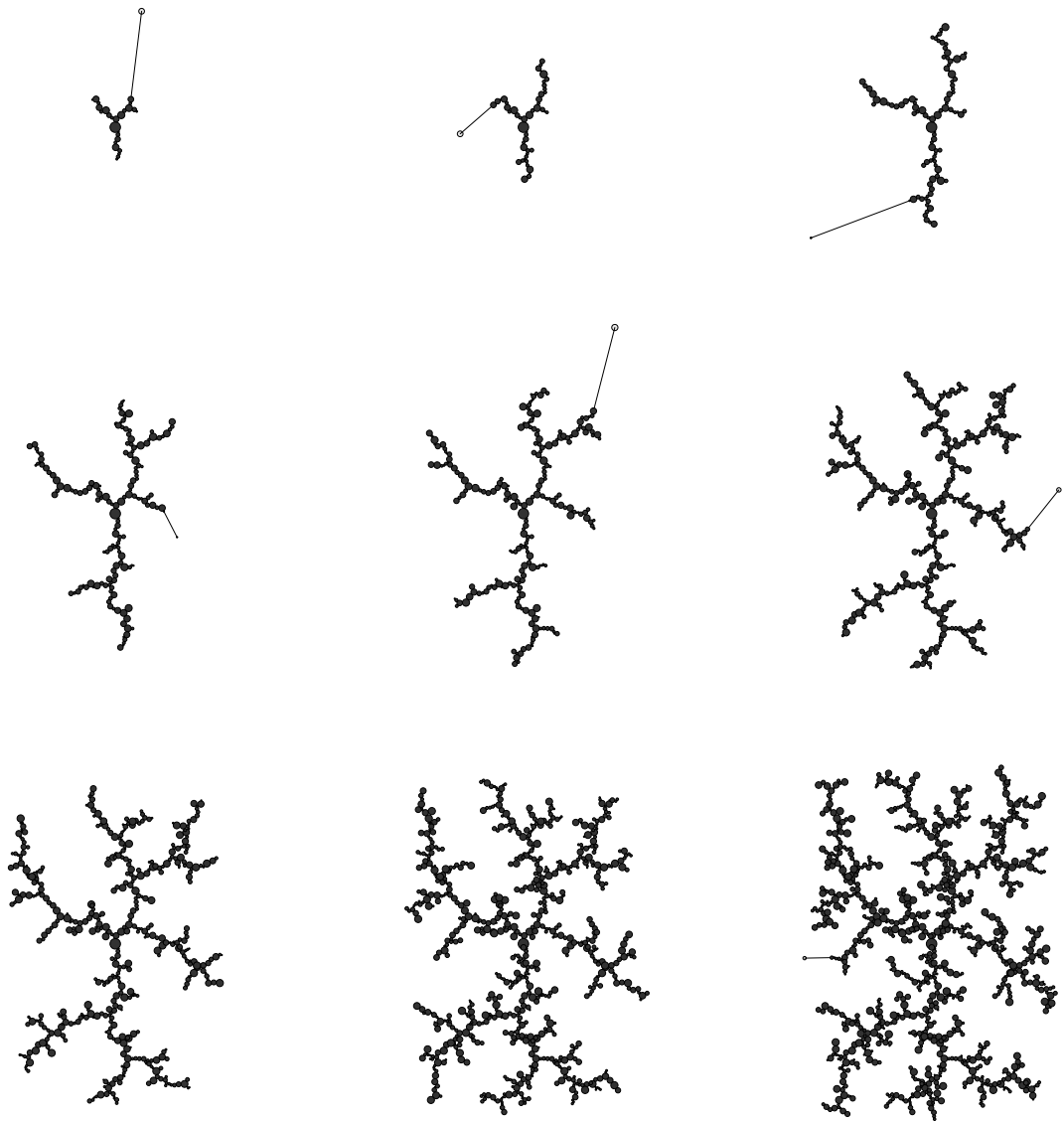


- 3 By drawing the starting position of the new circle and the connecting line to the circle closest to it, the process just described can be visualized using these three lines of code.

- 4 By calculating the `angle` to the closest neighbor, the new circle can be positioned so the two circles touch.
→ [Kap.P.1.1.2 Color spectrum in a circle](#)

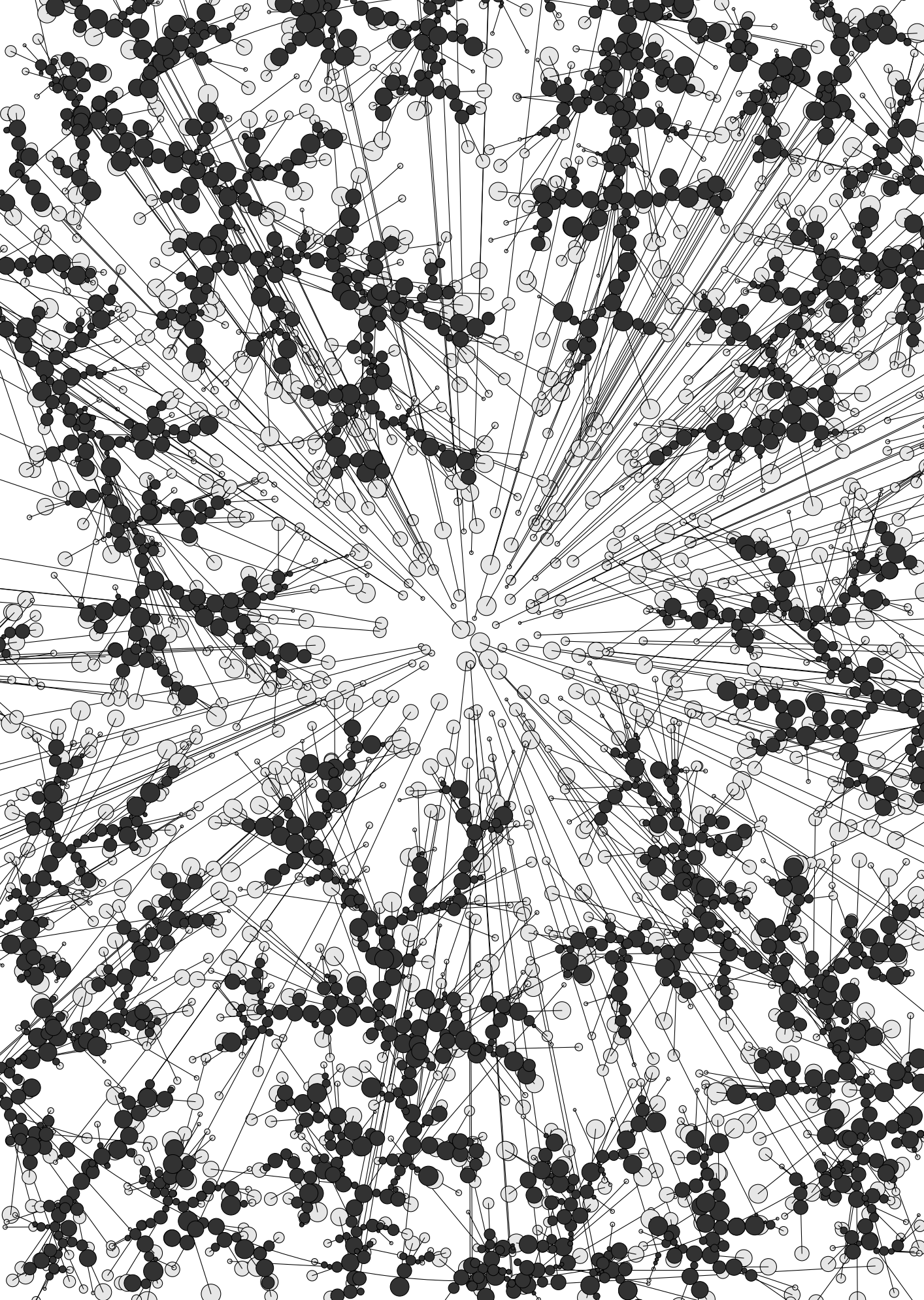


- 5 The circles are drawn.
- 6 When `currentCount` reaches the defined upper limit, the program is stopped using the function `noLoop()`.



→ **P_2_2_4_01** This sequence demonstrates how the structure grows gradually but continually.

→ **P_2_2_4_02** When the source circle is particularly large, the structure grows from the outside in. In addition, the initial positions of the circles and connecting lines are drawn here to their new position (Key 1).



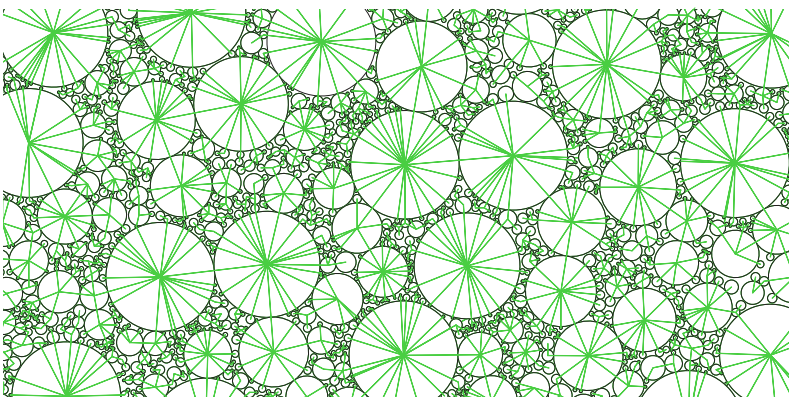
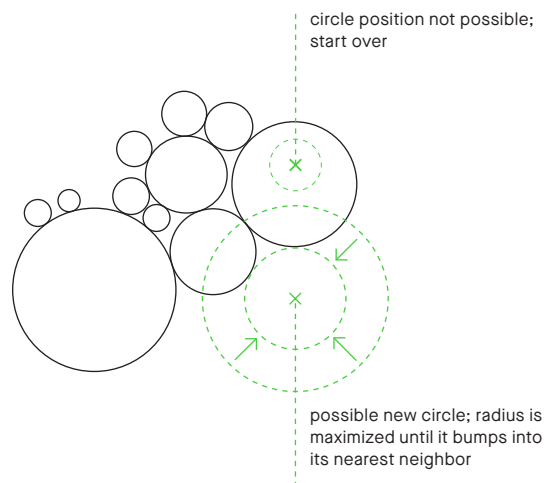
P.2.2.5 Structural density from agents

In this example an iterative process again serves as a shape-giver: Generate a new circle. If this circle does not intersect with any other circle in the display, make it as large as possible; if it intersects with another circle, start over. The aim of this algorithm is to pack the circles so densely that eventually even the smallest gaps are closed.

→ P_2_2_5_01

Here, too, a new circle (shown by a dashed yellow outline) with a random position and size is generated in each frame. When this intersects with a preexisting circle, the algorithm starts over.

Otherwise, it locates the closest circle. The distance to this circle and its radius now defines how large the new circle has to be drawn so that it touches its neighbor and the circles can be packed densely.



→ P_2_2_5_01 The algorithm fills the area with circles that become smaller and smaller. The green lines show on which circles new growths have docked.

```

function draw() {
  background(255);

1  var newX = random(maxRadius, width - maxRadius);
   var newY = random(maxRadius, height - maxRadius);
2  if (mouseIsPressed && mouseButton == LEFT) {
    newX = random(mouseX - mouseRect, mouseX + mouseRect);
    newY = random(mouseY - mouseRect, mouseY + mouseRect);
  }

  var intersection = false;
3  for (var newR = maxRadius; newR >= minRadius; newR--) {
    for (var i = 0; i < circles.length; i++) {
      var d = dist(newX, newY, circles[i].x, circles[i].y);
4      intersection = d < circles[i].r + newR;
      if (intersection) {
        break;
      }
    }
5    if (!intersection) {
      circles.push(new Circle(newX, newY, newR));
      break;
    }
  }

  for (var i = 0; i < circles.length; i++) {
6    if (showLine) {
      var closestCircle;
      for (var j = 0; j < circles.length; j++) {
        var d = dist(circles[i].x, circles[i].y,
                     circles[j].x, circles[j].y);
        if (d <= circles[i].r + circles[j].r + 1) {
          closestCircle = circles[j];
          break;
        }
      }
      if (closestCircle) {
        stroke(100, 230, 100);
        strokeWeight(0.75);
        line(circles[i].x, circles[i].y,
             closestCircle.x, closestCircle.y);
      }
    }
  }

7  if (showCircle) circles[i].draw();
}

...
}

```

Mouse: Drag: Targeted placement of the circles

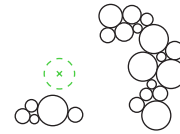
Keys: 1: Show/hide circles

2: Show/hide lines

Arrow ↓/↑: Change area to be drawn

S: Save image

- 1 Create a position and radius for a new circle.

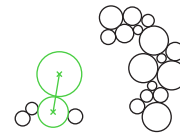


- 2 By holding down the mouse button, the range for random values is limited, allowing new circles to be positioned selectively and generating an interactive drawing tool.

- 3 The radius of the new circle must now be determined. For this, `newR` is set to the maximum radius, `maxRadius`, and counted down.

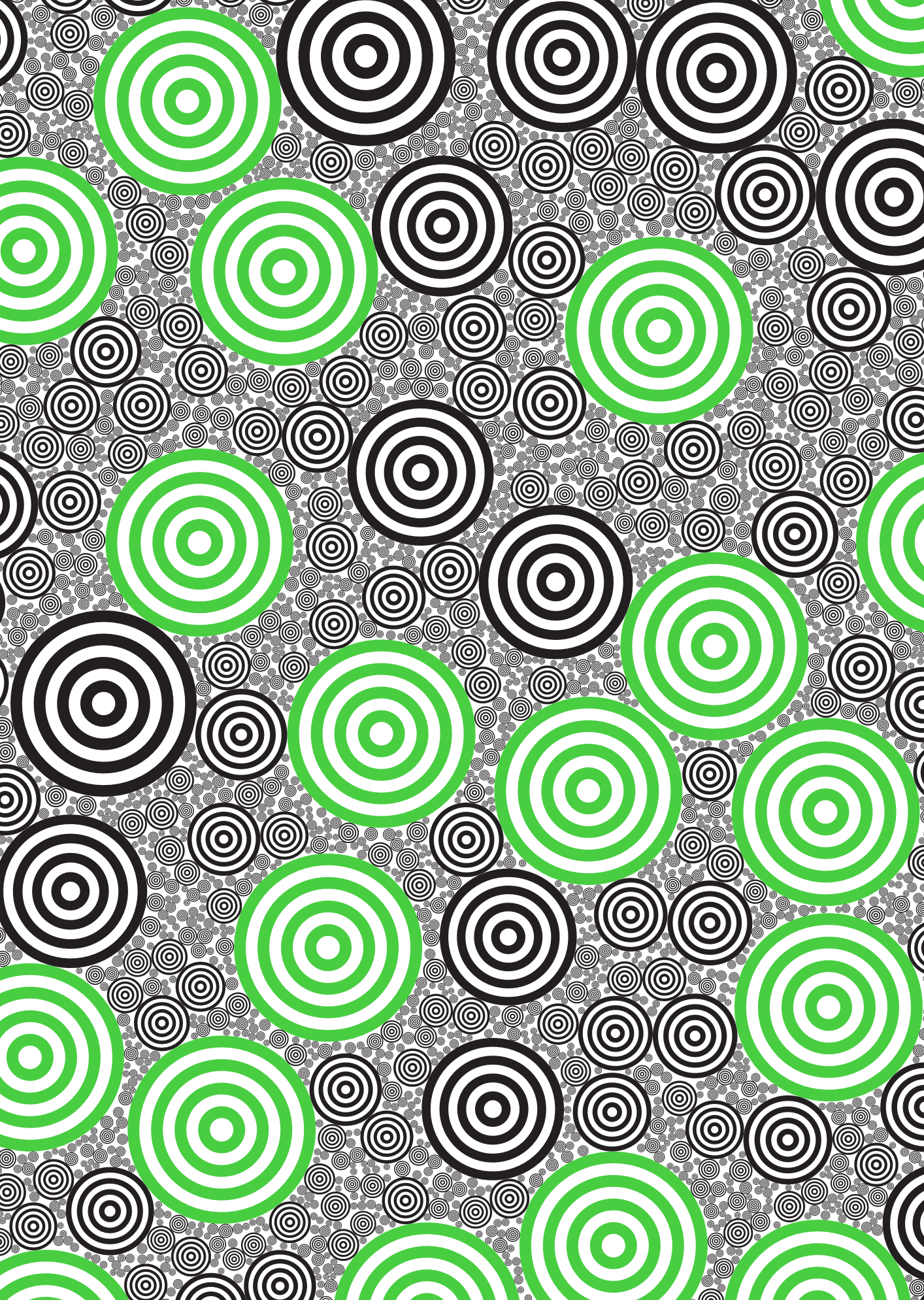
- 4 All existing circles are compared to the new one. When an intersection exists (e.g., when the distance is smaller than the sum of both radii), then the variable `intersection` is set to `true`.

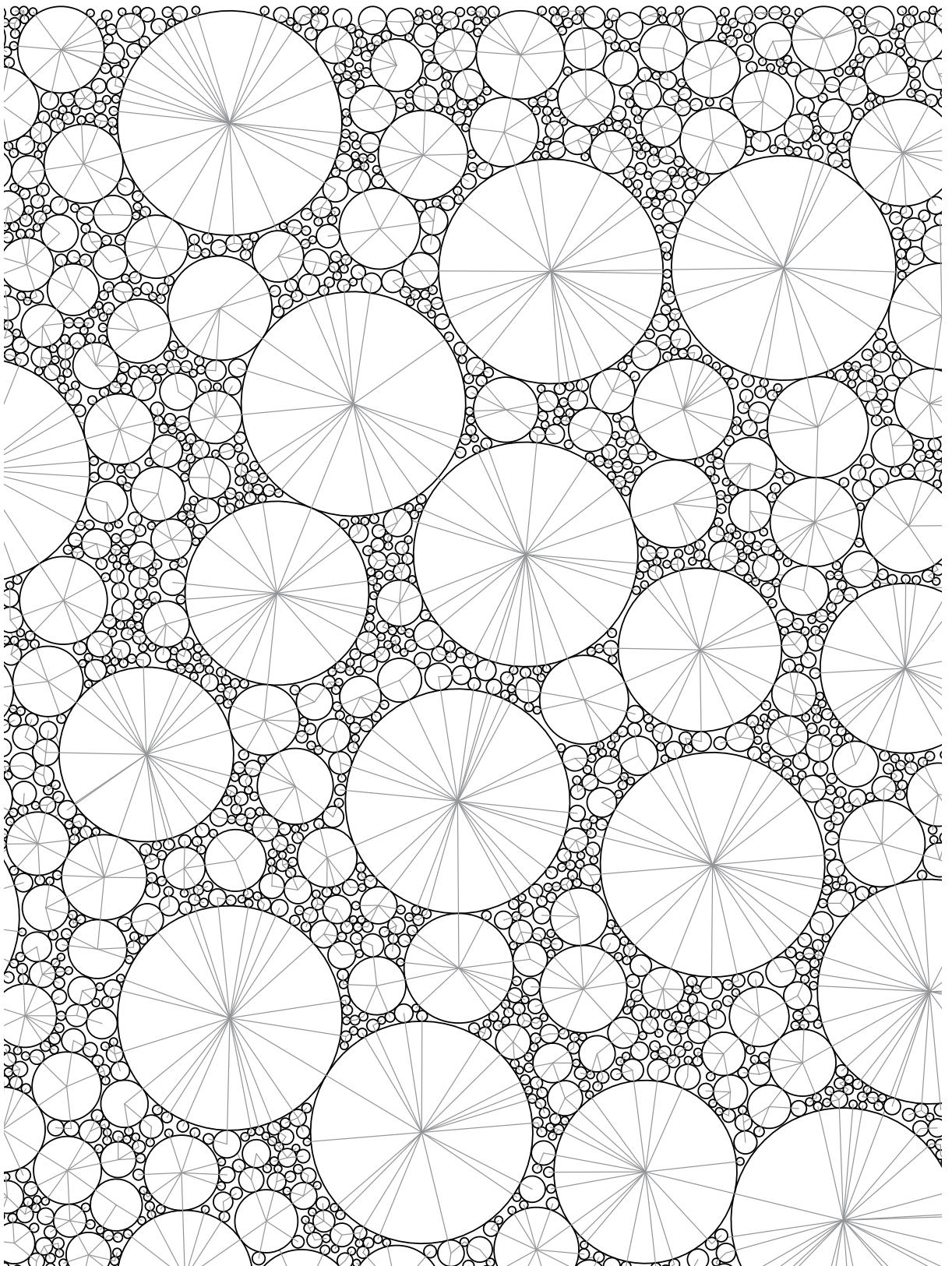
- 5 If there is no overlap, a new instance of the `Circles` class is created and saved.



- 6 If the `showLine` option is selected, each circle is compared to all others to find an adjoining circle, `closestCircle`.

- 7 If `showCircle` is `true`, the circle is also drawn.





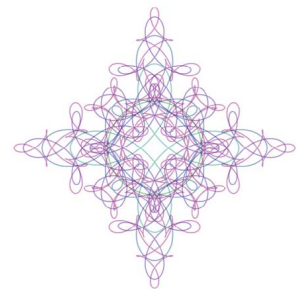
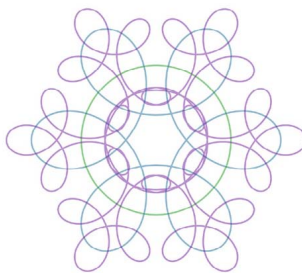
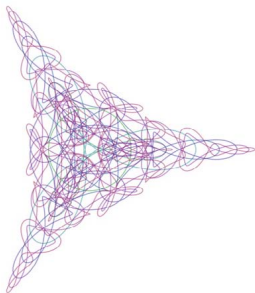
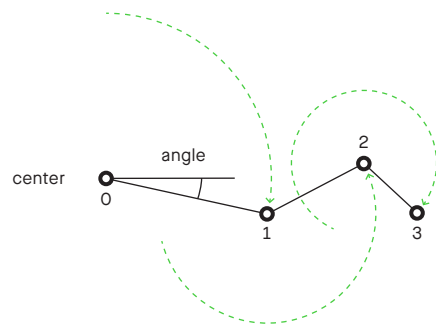
→ **P_2_2_5_02** By loading SVG modules, very different images are created. Using keys 1 to 3, it is possible to specify which elements are to be visible: the SVGs, the connecting lines, or the circles.

P.2.2.6 Agents on a pendulum

These double agents are changeable and chained together. The agents move along sets of interrelated pendulums, leaving behind complex traces that are reminiscent of the drawings of a Spirograph. Over time, their secret mission becomes visible.

→ P_2_2_6_01

The agents form a chain of pendulums. To determine the individual positions of the agents, we begin at the center. The angle of rotation and the length of the pendulum determine the position of the first agent. This is also the rotation center for the next pendulum. The farther out in the chain, the shorter the pendulum becomes and the faster it turns. Every other pendulum turns in the opposite direction.



→ P_2_2_6_01 Shapes with, e.g., three, six, or four symmetry axes are produced with different settings for the ratio of the rotational speeds.


```

function draw() {
  background(0, 0, 100);

1  angle += speed;

2  if (angle <= maxAngle + speed) {
    var pos = center.copy();

3    for (var i = 0; i < joints; i++) {
      var a = angle * pow(speedRelation, i);
      if (i % 2 == 1) a = -a;
4      var nextPos = p5.Vector.fromAngle(radians(a));
      nextPos.setMag((joints - i) / joints * lineLength);
      nextPos.add(pos);

5      if (showPendulum) {
        noStroke();
        fill(0, 10);
        ellipse(pos.x, pos.y, 4, 4);
        noFill();
        stroke(0, 10);
        line(pos.x, pos.y, nextPos.x, nextPos.y);
      }

6      pendulumPath[i].push(nextPos);
      pos = nextPos;
    }
  }

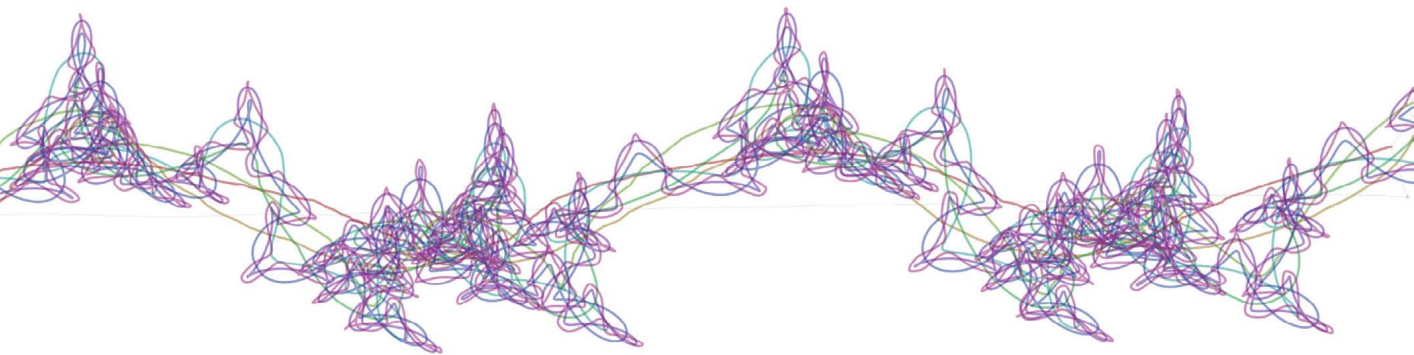
  if (showPendulumPath) {
    strokeWeight(1.6);
    for (var i = 0; i < pendulumPath.length; i++) {
      var path = pendulumPath[i];

7      beginShape();
      var hue = map(i, 0, joints, 120, 360);
      stroke(hue, 80, 60, 50);
      for (var j = 0; j < path.length; j++) {
        vertex(path[j].x, path[j].y);
      }
      endShape();
    }
  }
}

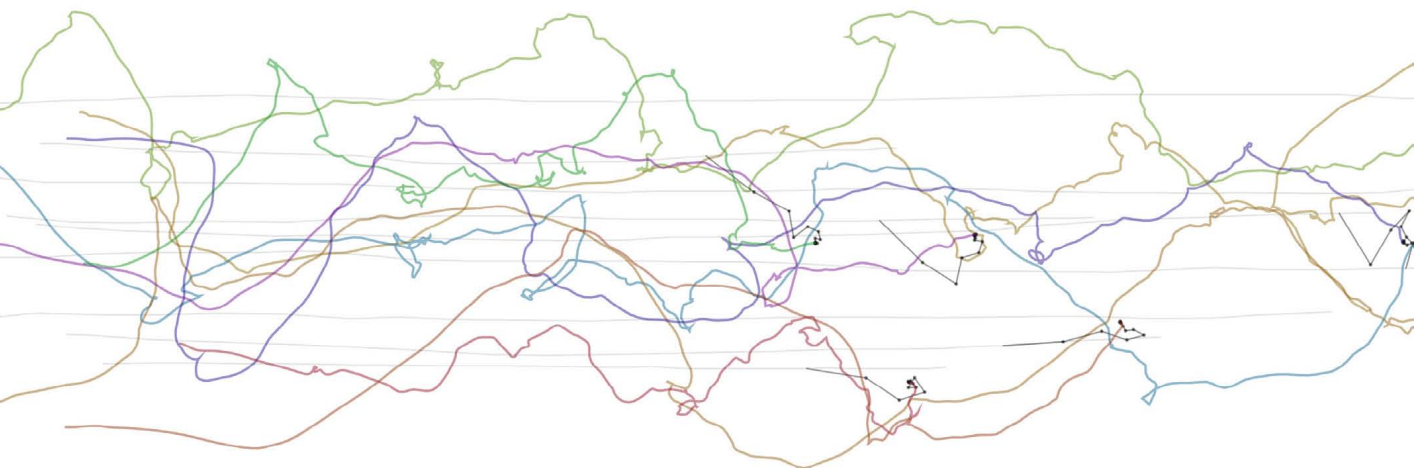
```

Keys: 1: Toggle display of pendulum off/on
 2: Display toggle path off/on
 DEL: Clear canvas
 -/+ : Jog velocity ratio -/+
 Arrow↓/↑: Decrease/increase line length -/+
 Arrow ←/→: Jog rotational velocity -/+
 S: Save image

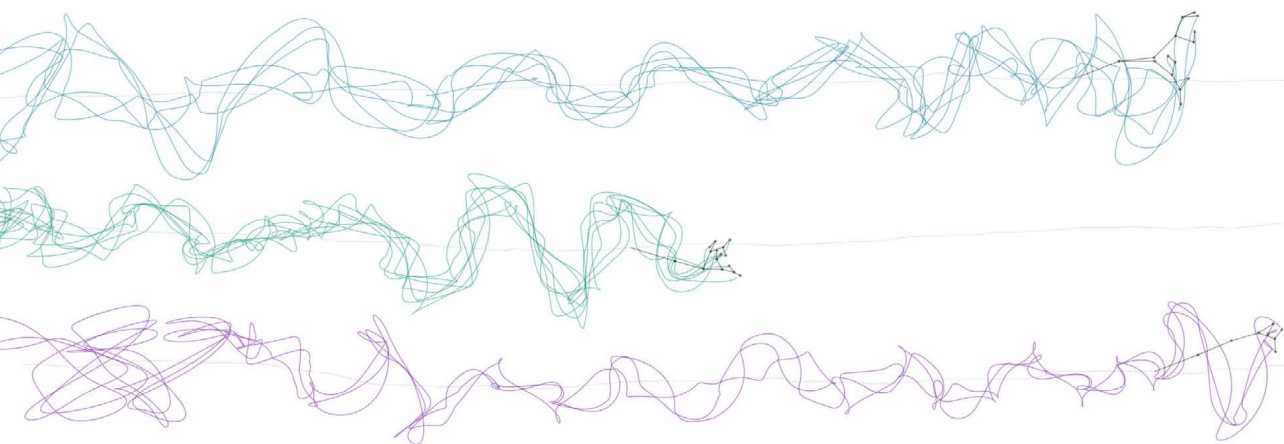
- 1 The variable `angle` contains the angle for the first pendulum. In each frame, the angle is incremented slightly.
- 2 The variable `center` is copied to `pos`. It contains a value from `p5.Vector`. Thus, only one variable is needed to store the x and y coordinates of a point. In addition, the `p5.Vector` object offers much more practical computing functionalities.
- 3 The angle `a` for a pendulum is calculated from the base angle and a factor, which increases in size the farther the pendulum is from the center; e.g., for the value 2 for `speedRelation`, the factor for the pendulum with index 3 is, in this case, 8 (2 to the power of 3). For every other angle, the direction of rotation is reversed.
- 4 To reach the position of the pendulum end, use the `fromAngle()` function to generate a unit vector (a vector of length 1) from the angle just calculated. `setMag()` extends it and adds the current position `pos`.
- 5 If the pendulums are to be displayed, a circle and a line are drawn.
- 6 The new calculated position `nextPos` is added to the path of the pendulum and is used for the next iteration as start position `pos`.
- 7 The paths are allotted individual colors. Depending on their index `i`, a hue between 120° (green) and 360° (red) is determined.



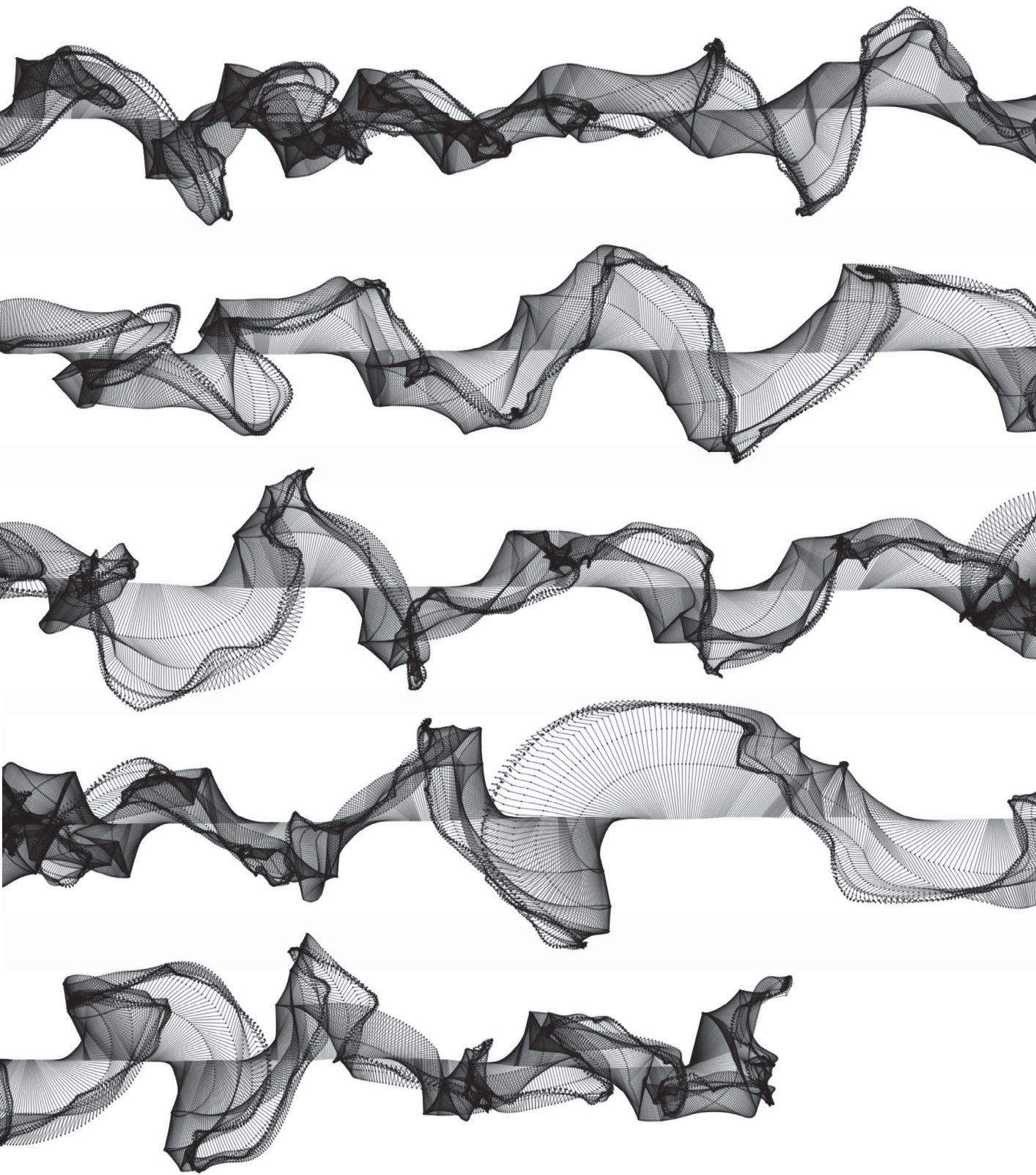
→ **P_2_2_6_02** In other variations of the program, pendulums move along a path drawn with the mouse.



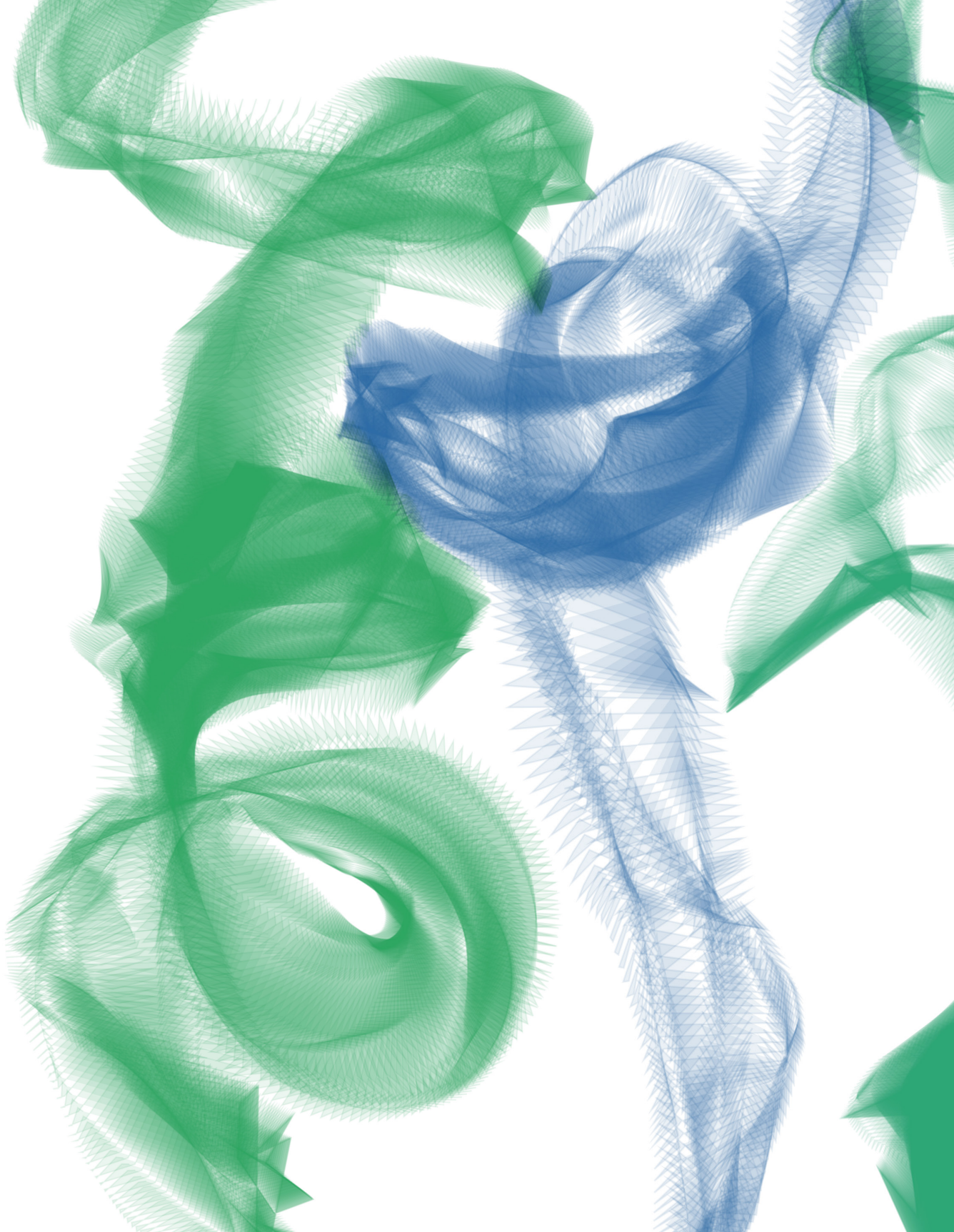
→ **P_2_2_6_03** Marks like these occur when a pendulum no longer completely orbits its center but—as its name suggests—swings back and forth.



→ **P_2_2_6_04** If the pendulums form a branched tree structure, then there will be multiple endpoints.



→ **P_2_2_6_03** In this drawing mode, the background is not deleted. The animation of the pendulum is continuous and thus becomes visible.



→ P_2_2_6_04 Here the end points of the tree structure are connected to form a transparent polygon.

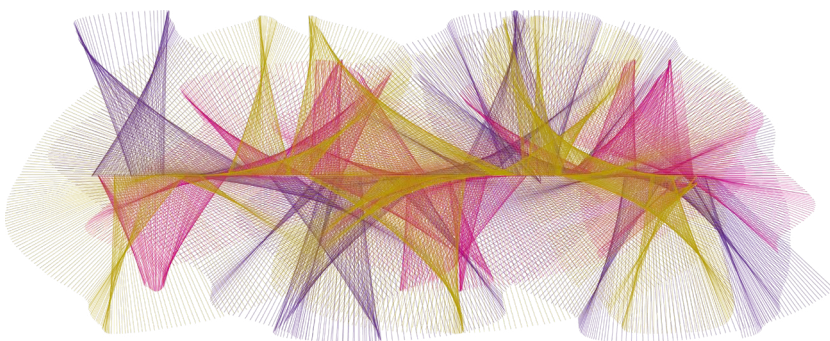
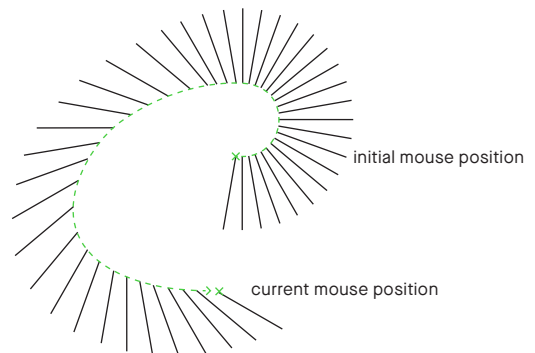


P.2.3.1 Drawing with animated brushes

In previous chapters, agents moved autonomously according to predefined rules. In this example, users will interact with an agent, creating an experimental drawing tool that follows its own set of rules. What makes animated brushes so unusual is their ability to be inspired by their own behavior while drawing. This process offers a much greater range of expression, as the act of drawing becomes like a ballroom dance with a partner.

→ P_2_3_1_01

The first drawing tool is an excellent demonstration of how much visual potential can be contained in very simple principles. A line rotates around the mouse position. The lines consolidate in different ways depending on the speed and direction of the mouse's movements. The line changes its color and length with each mouse click. The rotation speed can be set with the right or left arrow keys.



→ P_2_3_1_01 The mouse was only moved back and forth along the middle of the horizontal line. The animated brush thereby generated different levels of density.


```
function draw() {
  1  if (mouseIsPressed && mouseButton == LEFT) {
    2    push();
    3    translate(mouseX, mouseY);
    rotate(radians(angle));
    stroke(c);
    line(0, 0, lineLength, 0);
    pop();

    4    angle += angleSpeed;
  }
}
```

```
5  function mousePressed() {
    lineLength = random(70, 200);
  }
```

Mouse: Drag: Draw

Keys: 1-4: Change color settings

Space: New random color

DEL: Clear canvas

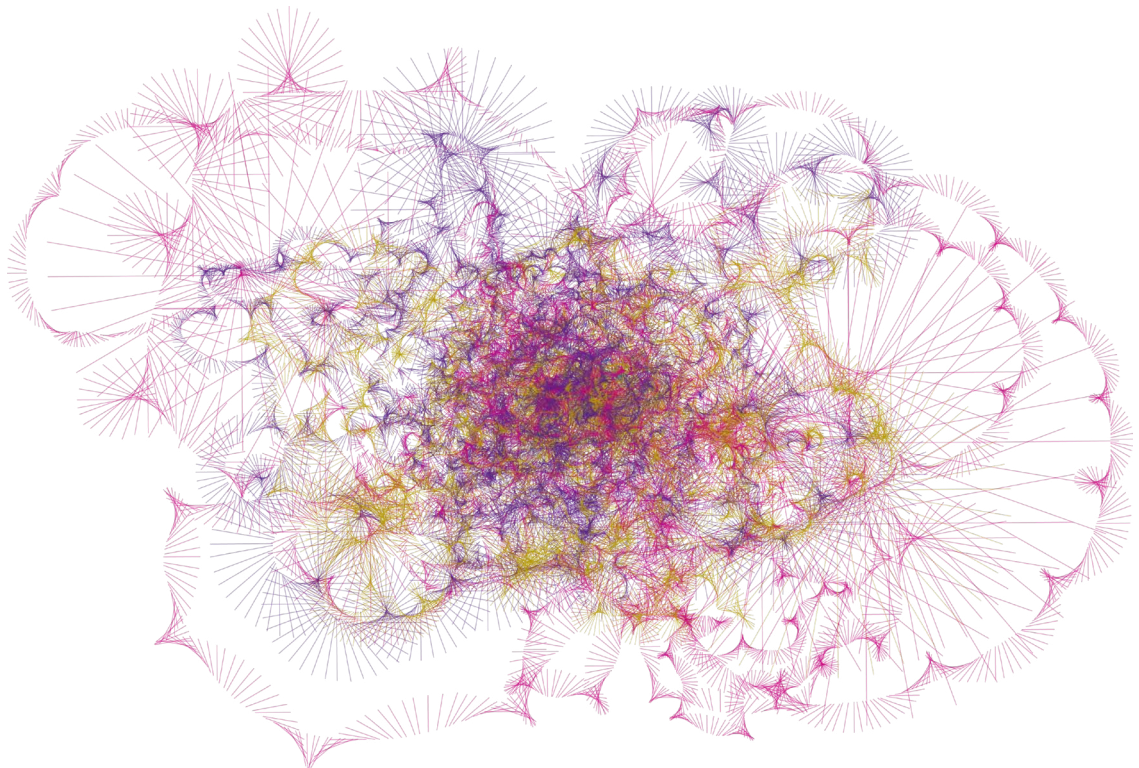
D: Change direction and mirror angle

Arrow ↓/↑: Line length -/+

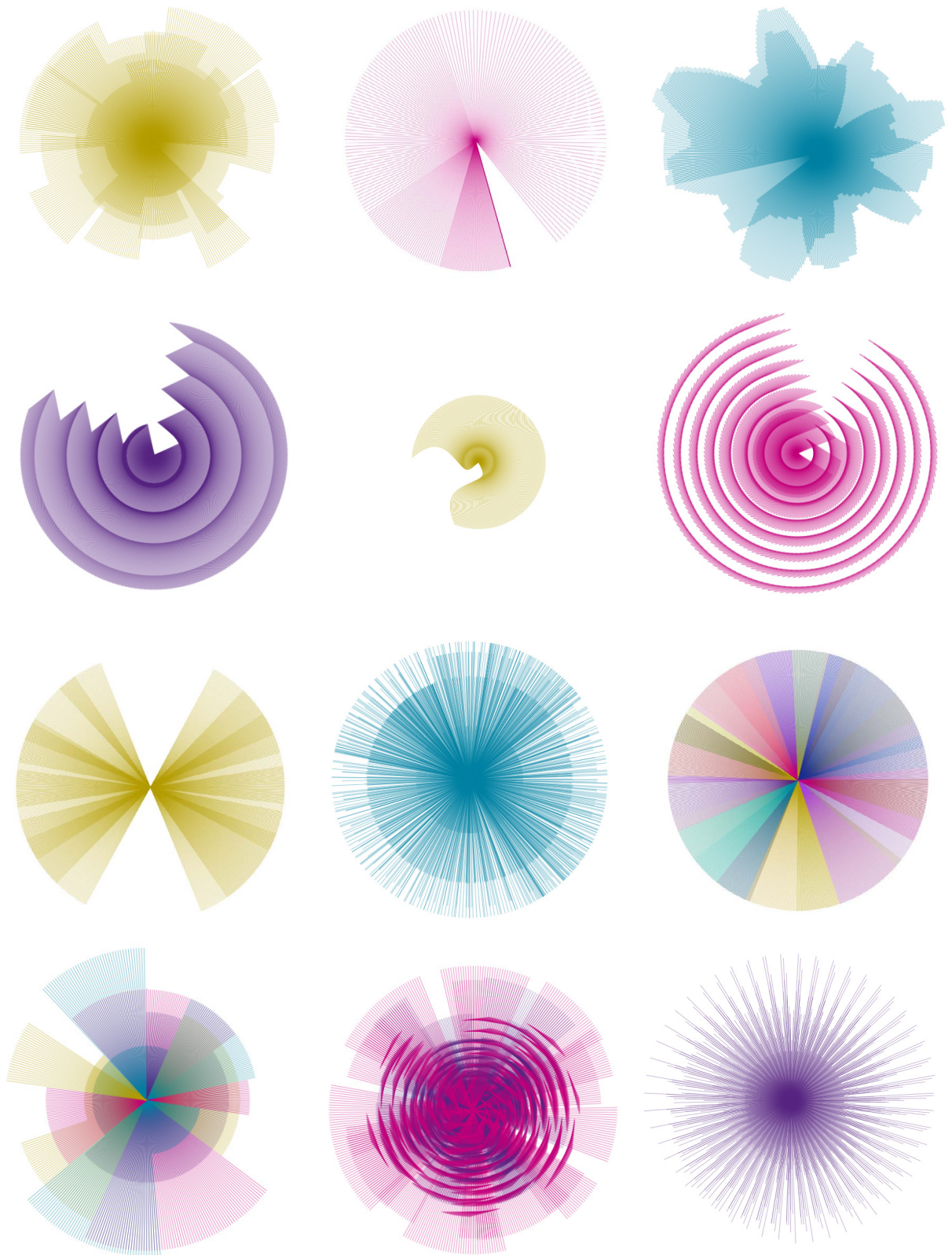
Arrow ←/→: Rotation speed -/+

S: Save image

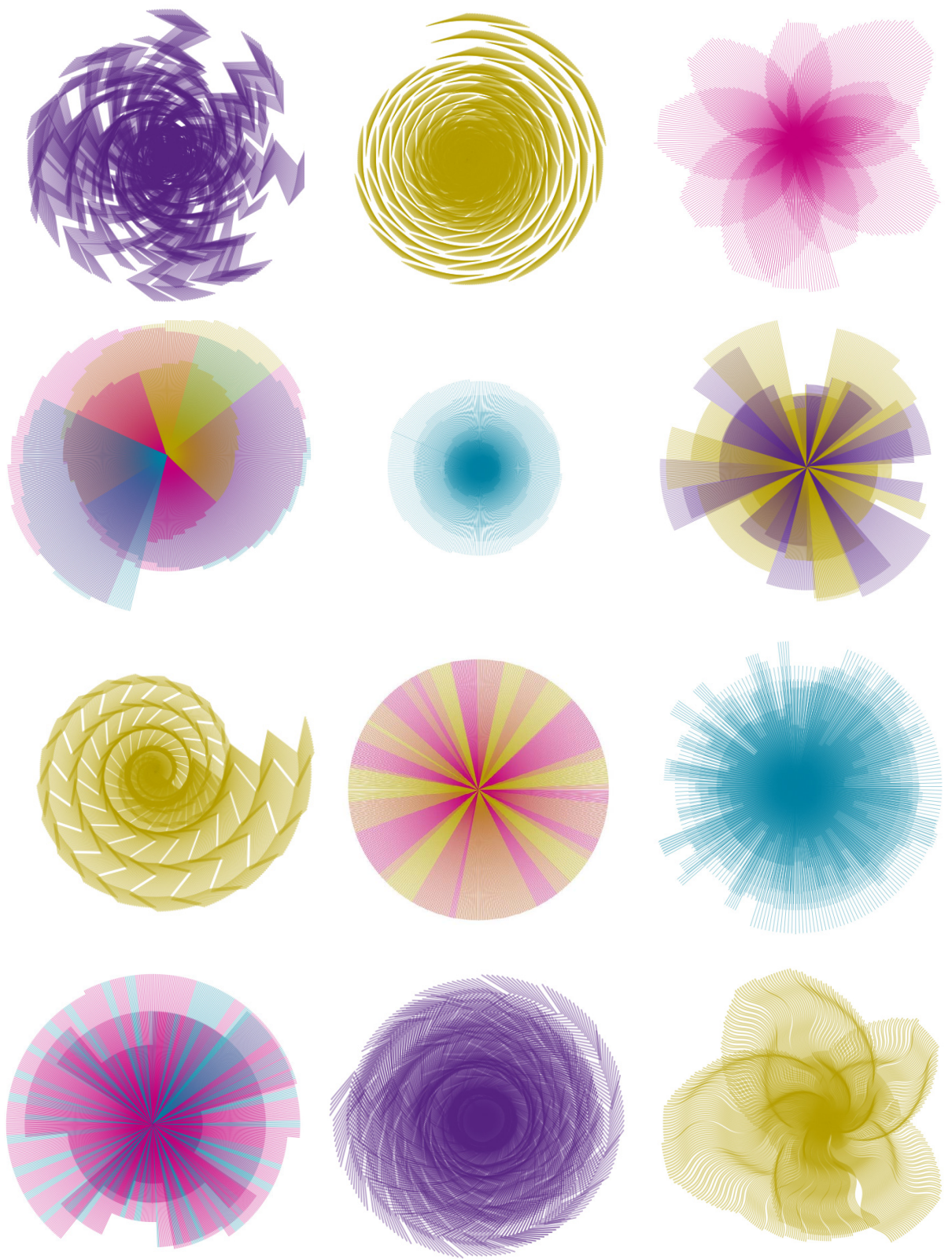
- 1 Only when the left mouse button is pressed will something be drawn.
- 2 The line is supposed to rotate around the mouse position. Therefore, the origin of the coordinate system must first be moved to the mouse position using the `translate()` function. The coordinate system is then rotated with the `rotate()` function.
- 3 The horizontal line drawn now becomes a rotating brush.
- 4 The rotation angle is increased by a value for the rotation speed.
- 5 The length of the line changes with each click.



→ P_2_3_1_02 In addition to the mouse movement and the different colors, the step size of the rotating lines is also changed.



→ **P_2_3_1_02** Lines of random colors rotate around the mouse position when the mouse button is held down. Even without moving the mouse, a wealth of parameters such as line's length, color, line shape, and rotation speed can be influenced with the keys, thereby generating a variety of images.



P.2.3.2 Relation and distance in drawing

Since the relation of individual elements to one another is crucial to an overall image, it is important for users to be able to control individual parameters such as distance and angle; this program provides the necessary knowledge.

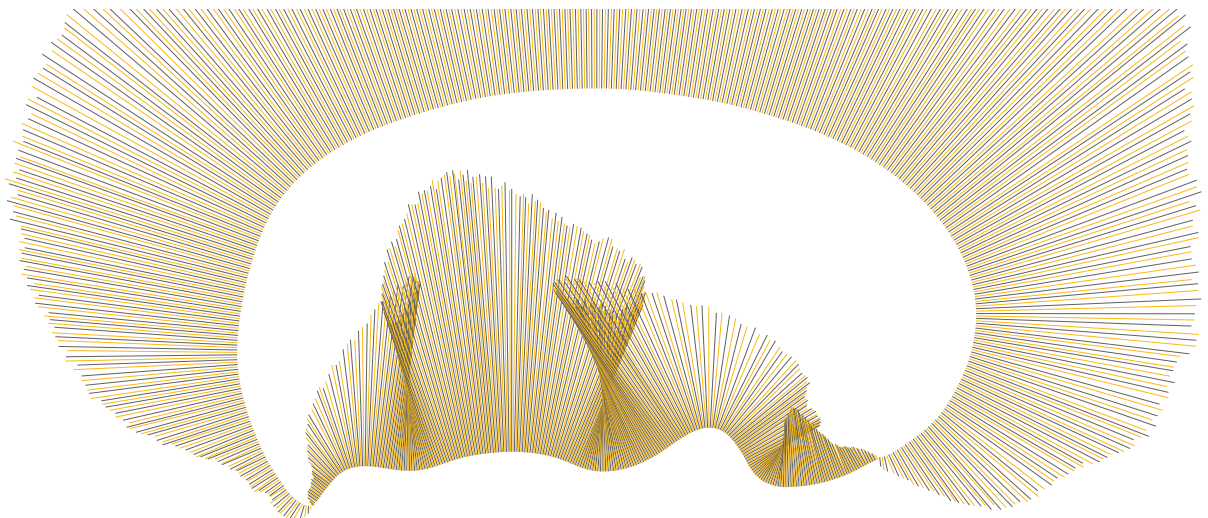
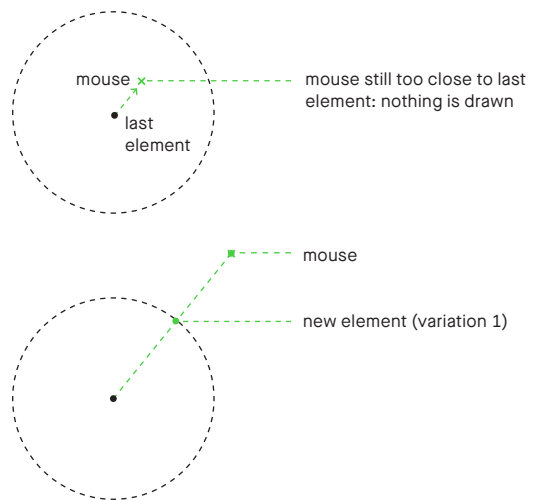
→ P_2_3_2_01

In the previous example, a new element was drawn on the drawing canvas in each frame when the mouse button was held down. This function is now restricted: a new element is positioned only when it stays a minimum distance from the previously drawn element. There are several ways to do this:

Variation 1: The new element is not placed directly at the mouse position but rather at the exact specified minimum distance from the last element.

Variation 2: The new element is placed at the mouse position, and the minimum distance only serves as a threshold value.

minimum distance radius



→ P_2_3_2_01 The more quickly the mouse is moved during the drawing process, the longer the lines become.

```

function draw() {
1   if (mouseIsPressed && mouseButton == LEFT) {
       var d = dist(x, y, mouseX, mouseY);

2       if (d > stepSize) {
           var angle = atan2(mouseY - y, mouseX - x);

           push();
           translate(x, y);
           rotate(angle);
           stroke(col);
3       if (frameCount % 2 == 0) stroke(150);
4       line(0, 0, 0, lineLength * random(0.95, 1.0) * d/10);
           pop();

5       if (drawMode == 1) {
           x = x + cos(angle) * stepSize;
           y = y + sin(angle) * stepSize;
       }
       else {
           x = mouseX;
           y = mouseY;
       }
   }
}

```

Mouse: Drag: Draw

Keys: 1-2: Drawing mode

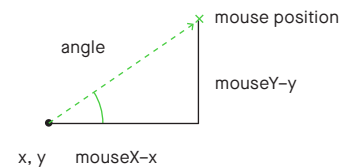
DEL: Clear canvas

Arrow ↓/↑: Line length - / +

S: Save image

1 Holding down the mouse button (i.e., when you want to draw) calculates the distance from the last drawing position (x, y) to the current mouse position.

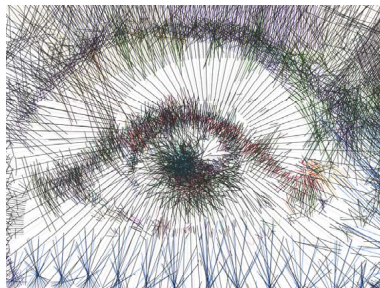
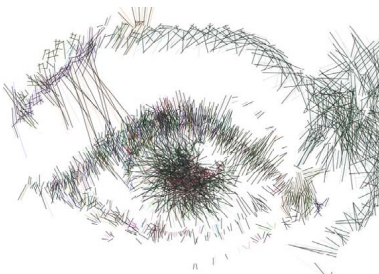
2 If this distance is greater than stepSize, a new point is drawn. To do this, the angle to the previous drawing position has to be calculated. This is easily accomplished with the function atan2(), which requires two parameters: the vertical distance between the two points mouseY - y and the horizontal distance between the two points mouseX - x.



3 The lines are drawn alternately in different colors: the randomly selected color (col) or a medium gray.

4 A vertical line is drawn. Since the coordinate system was previously rotated by angle, the line is now perpendicular to the drawing path. The line's length results from the basic length lineLength, a random factor that varies the length slightly, by the factor d/10. This means that the greater the distance between the old and new points, the longer the line is drawn, thereby reflecting the speed of the mouse.

5 In version 1 (drawMode==1), the new point is placed at the distance stepSize from the old position. In version 2, the mouse determines the new position.



→ P_2_3_2_01 → Illustration: Víctor Juárez Hernández The multiple superimposition of the lines can be used to create fine shading.

P.2.3.3 Drawing with type

Who doesn't like to switch brushes while painting?
In this application, the position and size of characters are constantly transformed according to the position and speed of the brush. The user can paint random series of letters or even whole novels.

→ P_2_3_3_01

Along the mouse's drawing line, a text appears that is defined in the program and is drawn larger or smaller depending on mouse speed.

drawn with a previously entered text
mouse movement

```
function draw() {  
  if (mouseIsPressed && mouseButton == LEFT) {  
    1 var d = dist(x, y, mouseX, mouseY);  
    textSize(fontSizeMin + d / 2);  
    2 var newLetter = letters.charAt(counter);  
    stepSize = textWidth(newLetter);  
  
    3 if (d > stepSize) {  
      var angle = atan2(mouseY - y, mouseX - x);  
  
      push();  
      translate(x, y);  
      rotate(angle + random(angleDistortion));  
      text(newLetter, 0, 0);  
      pop();  
  
      4 counter++;  
      if (counter >= letters.length) counter = 0;  
  
      x = x + cos(angle) * stepSize;  
      y = y + sin(angle) * stepSize;  
    }  
  }  
}
```

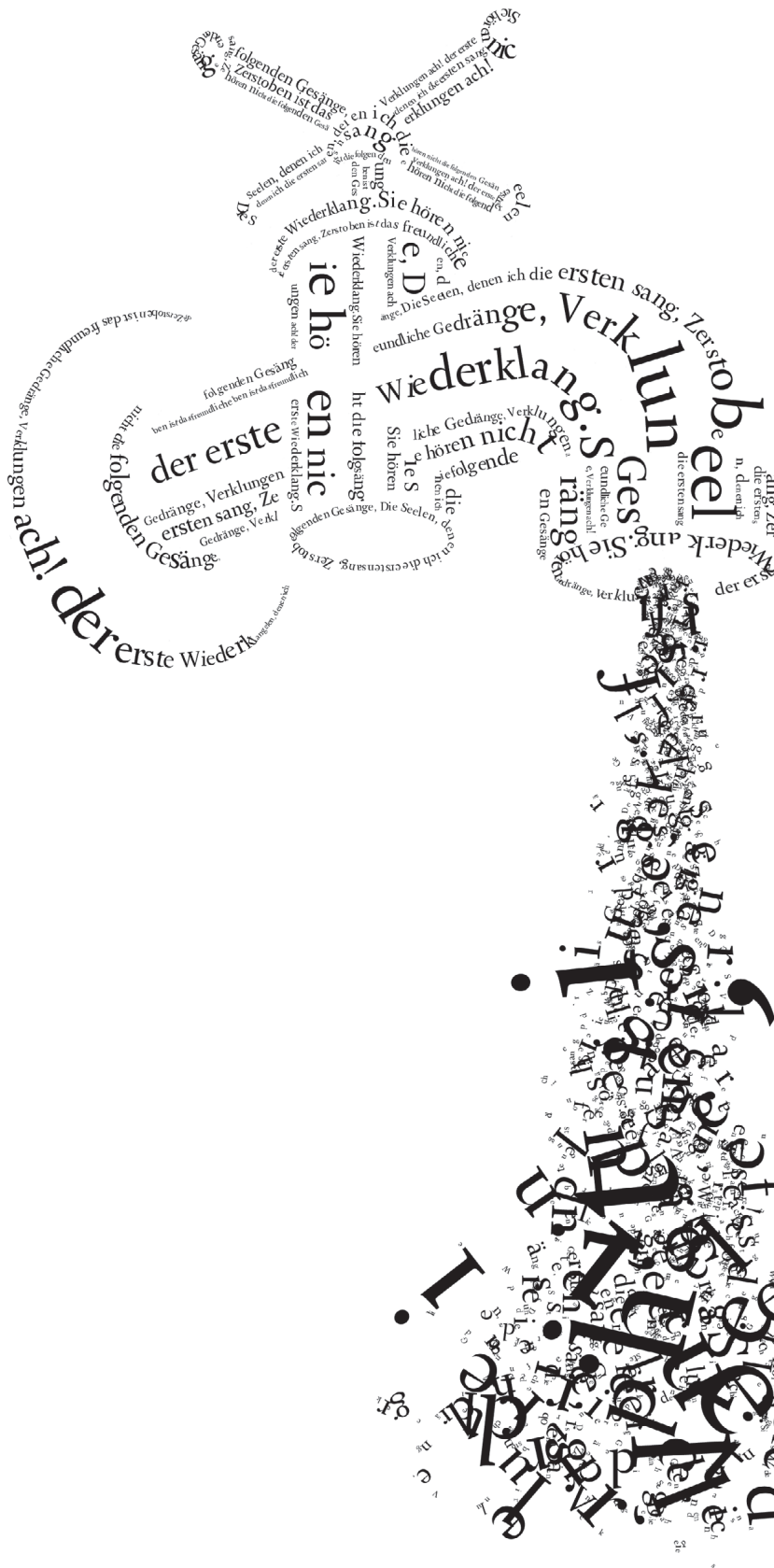
Mouse: Drag: Draw text

Keys: DEL: Clear canvas

Arrow ↓/↑: Distortion angle -/+

S: Save image

- 1 The distance between the mouse and the current writing position (x,y) is calculated. This, in turn, determines the font size for the next character. The value in `fontSizeMin` ensures that the font will not be smaller than a specified size.
- 2 To check whether a new letter can be written, the next character is selected from the string `letters` and the variable `stepSize` set to the character's width.
- 3 When there is enough space between the mouse and the current writing position, the new letter is written.
- 4 The variable `counter` counts how many letters have already been drawn. This value is used to read the letters in succession from the specified text `letters`. When `counter` is greater than the number of letters in the original text, it is reset to 0.



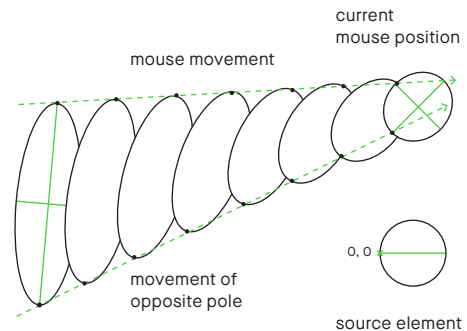
→ P_2_3_3_01_TABLET → Illustration: Pau Domingo In the tablet version, the pressure on the pen modulates the size of the text. An image can be placed as a template in the background and can be shown or hidden while drawing.

P.2.3.4 Drawing with dynamic brushes

A virtual rubber band becomes a dynamic paint-brush as arbitrary basic elements are strung like pearls on a string between brushstrokes and a lazy agent. The tension between the two poles defines the size and position of the elements when drawing.

→ P_2_3_4_01

A graphic element is dragged on one end by the mouse. The opposite end moves sluggishly in the direction of the mouse. Depending on the drawing speed and set inertia, the original elements are depicted as stretched to their limits or virtually unchanged. The width of the element remains the same.



```
function draw() {  
  1 if (mouseIsPressed && mouseButton == LEFT) {  
    var d = dist(x, y, mouseX, mouseY);  
  
    2 if (d > stepSize) {  
      var angle = atan2(mouseY - y, mouseX - x);  
  
      push();  
      3 translate(mouseX, mouseY);  
      rotate(angle + PI);  
      image(lineModule, 0, 0, d, moduleSize);  
      pop();  
  
      x = x + cos(angle) * stepSize;  
      y = y + sin(angle) * stepSize;  
    }  
  }  
}
```

Mouse: Drag: Draw

Keys: 1-9: Change module

DEL: Clear canvas

Arrow ↓/↑: Module size +/-

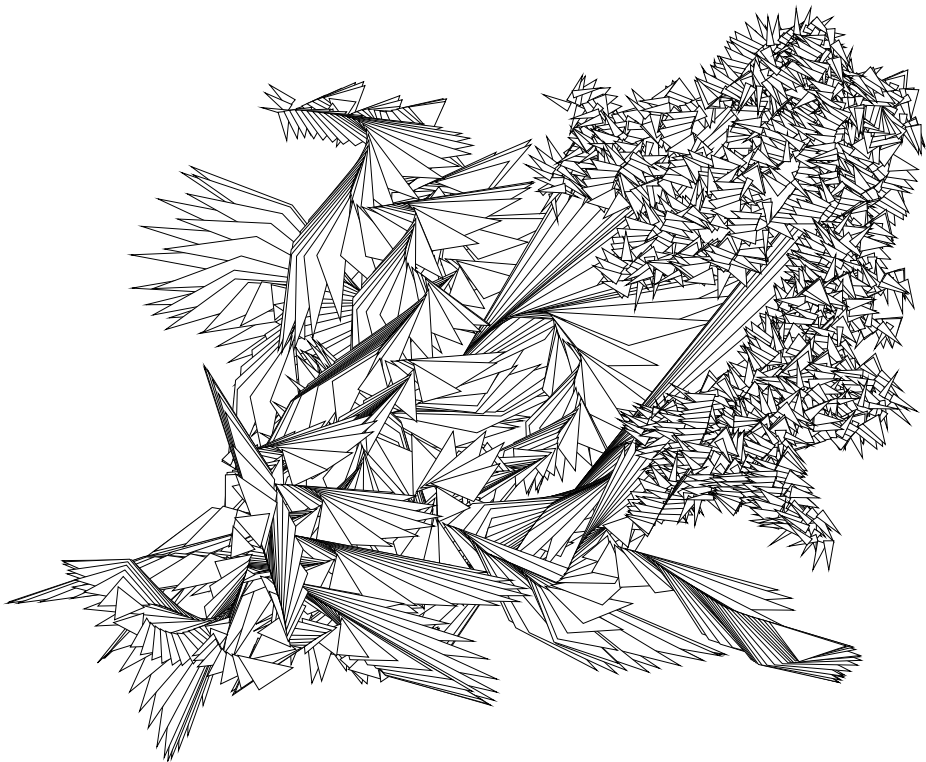
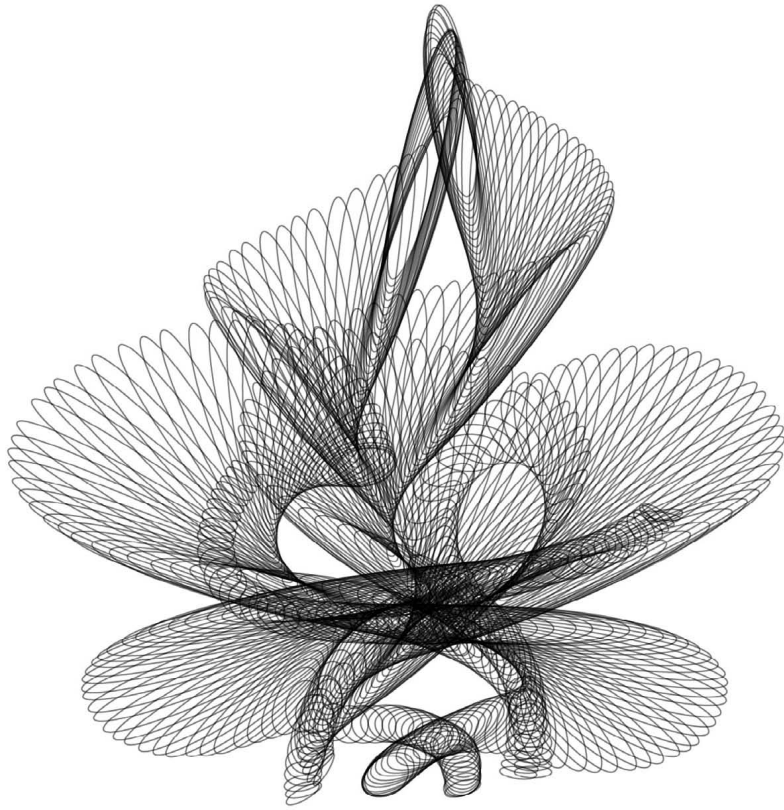
Arrow ←/→: Step size +/-

S: Save image

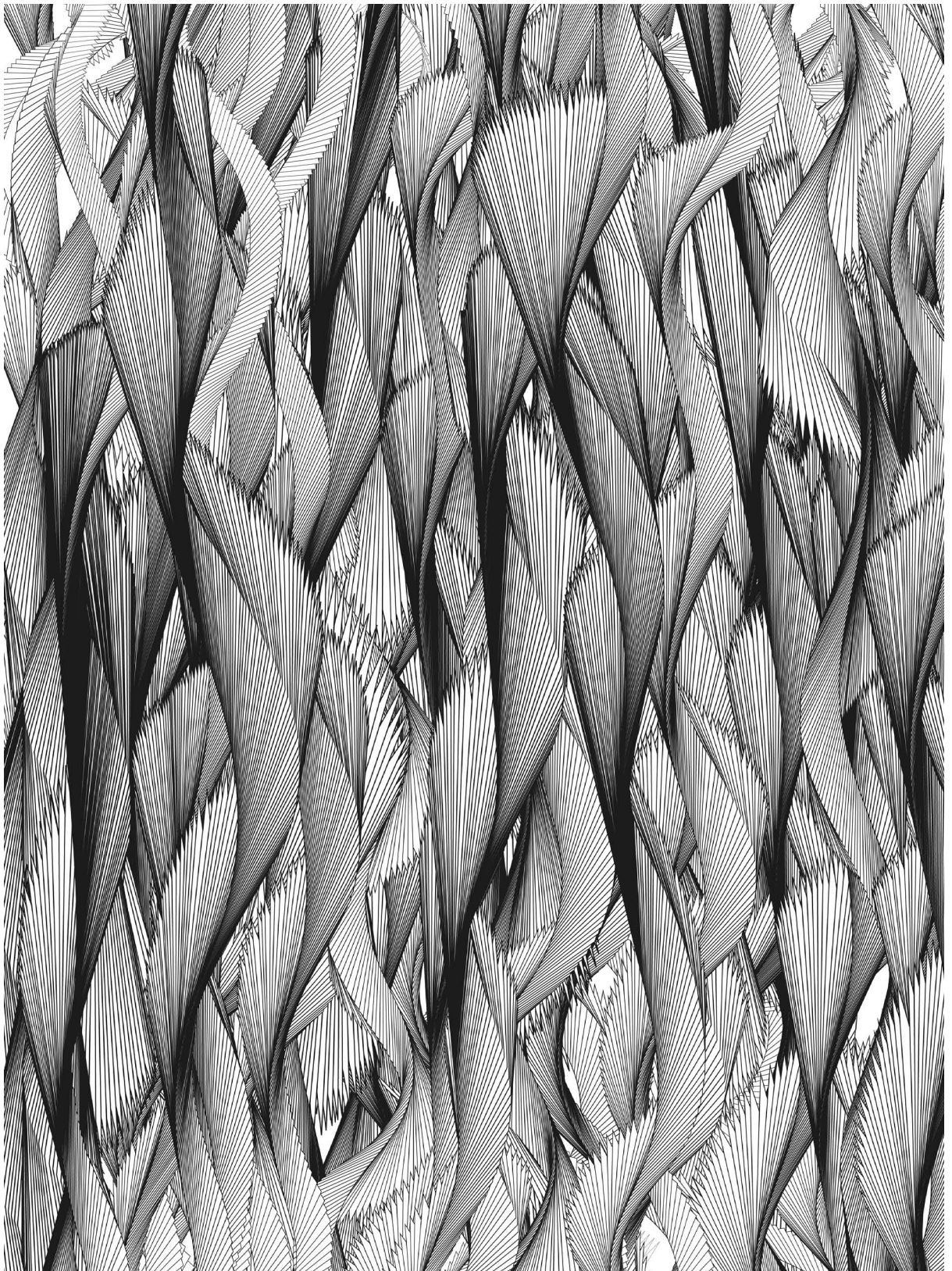
1 In this program, the variables `x` and `y` are used for the position of the opposite pole. The distance between these variables and the mouse position determines how much the character element is stretched.

2 The coordinate system is moved to the mouse position with `translate()` and turned so it faces the opposite pole; the element is then drawn. The additional angle, `PI` here, can vary depending on the SVG module. Its length is set at `d` (the distance to the mouse), the width to `moduleSize`.

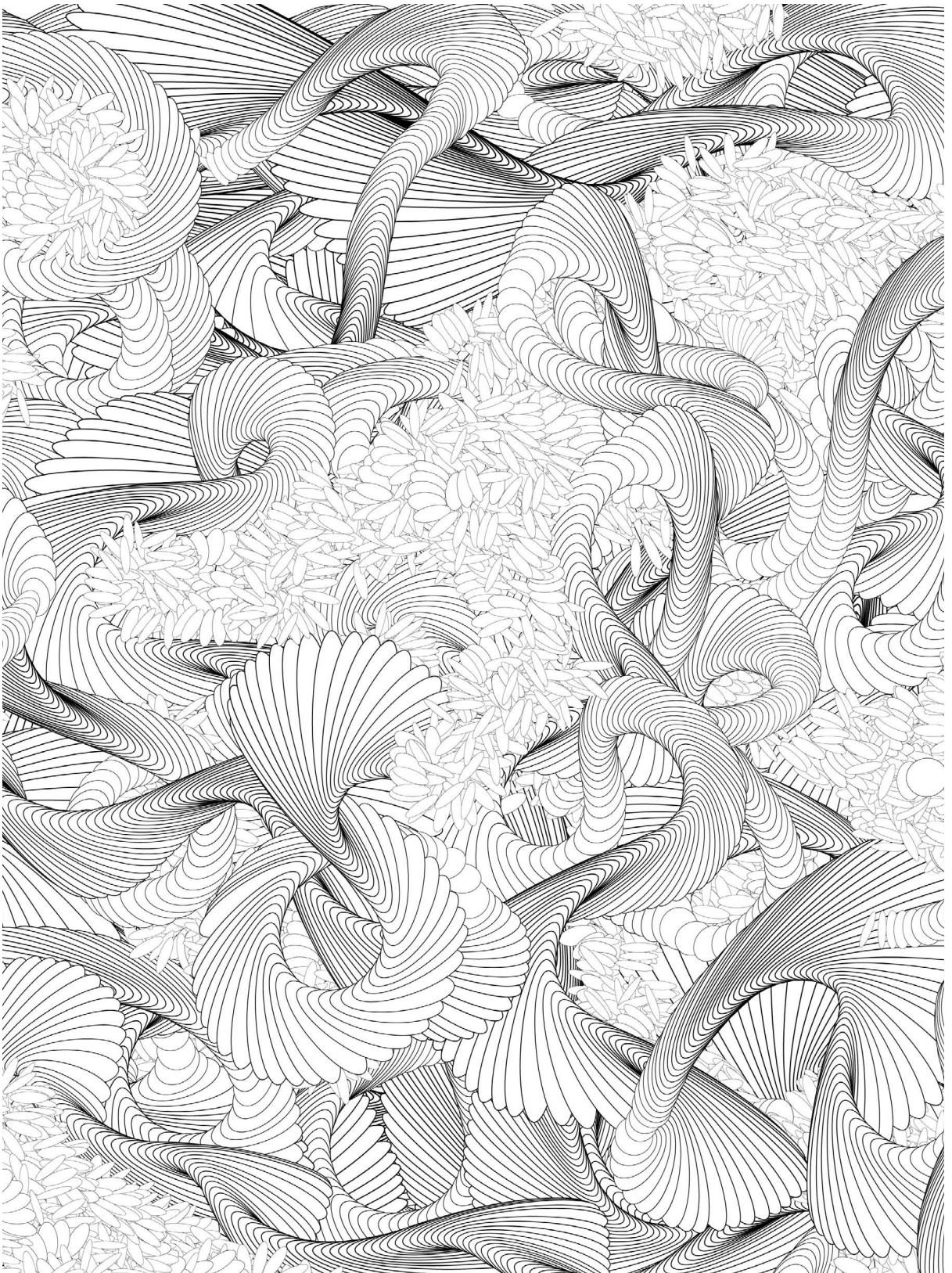
3 The position of the opposite pole is moved by the value `stepSize` at every step. Since this movement is usually much slower than that of the mouse, a rubber-band effect is produced.



→ P_2_3_4_01 → Illustrationen: Pau Domingo A very subdued opponent to the fast-moving mouse.



→ P_2_3_4_01 → **Illustration: Pau Domingo** Quick, regular movements with the mouse generate flowing forms.



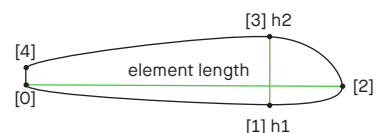
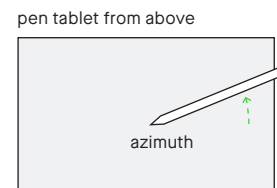
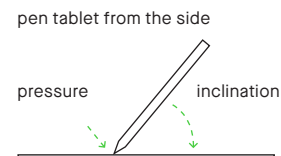
→ P_2_3_4_01 → Illustration: Pau Domingo Different movement speeds of the mouse also generate different structures.

P.2.3.5 Drawing with the pen tablet

Compared to a conventional mouse, a pen tablet puts several more parameters at a user's disposal, including pressure and position. When drawing, the pen movement can be better recorded and interpreted so that the movement of your hand can be reproduced more accurately. When you use a pen tablet you are closer to the generative process.

→ P_2_3_5_01_TABLET

The additional parameters provided by a pen tablet are transferred to a basic element (in this case a pen). The azimuth, pressure, and inclination define the rotation, saturation, and length, respectively, of the element. The parameters are read using the tablet classes of the book's Generative Design library. The form is not loaded from an SVG file but rather drawn as a curve, enabling the individual curve points to be manipulated more easily.



→ P_2_3_5_01_TABLET → Illustration: Jana-Lina Berkenbusch The new elements are always superimposed on the old ones, thereby making it necessary to work from the background to the foreground.


```

function draw() {
  var tabletValues = tablet.values();

1  var pressure = gamma(tabletValues.pressure, 2.5);
   var angle = tabletValues.azimuth;
   var penLength = cos(tabletValues.inclination);

2  if (pressure > 0 && penLength > 0) {
    push();
    translate(mouseX, mouseY);
    rotate(angle);

3    var elementLength = penLength * 250;
    var h1 = random(10) * (1.2 + penLength);
    var h2 = (-10 + random(10)) * (1.2 + penLength);

    ...

4    pointsX = [];
    pointsY = [];

    pointsX[0] = 0;
    pointsY[0] = 0;
    pointsX[1] = elementLength * 0.77;
    pointsY[1] = h1;
    pointsX[2] = elementLength;
    pointsY[2] = 0;
    pointsX[3] = elementLength * 0.77;
    pointsY[3] = h2;
    pointsX[4] = 0;
    pointsY[4] = -5;

5    beginShape();
    curveVertex(pointsX[3], pointsY[3]);
    for (var i = 0; i < pointsX.length; i++) {
      curveVertex(pointsX[i], pointsY[i]);
    }
    curveVertex(pointsX[1], pointsY[1]);
    endShape(CLOSE);
    pop();
  }
}

```

Mouse: Drag: Draw

Tablet: Pressure: Saturation
Azimuth: Rotation
Inclination: Length

Keys: 1-3: Drawing mode
6-0: Colors
DEL: Clear canvas
S: Save image

1 The tablet captures many characteristics of the pen. These three queries get pressure and angle information.

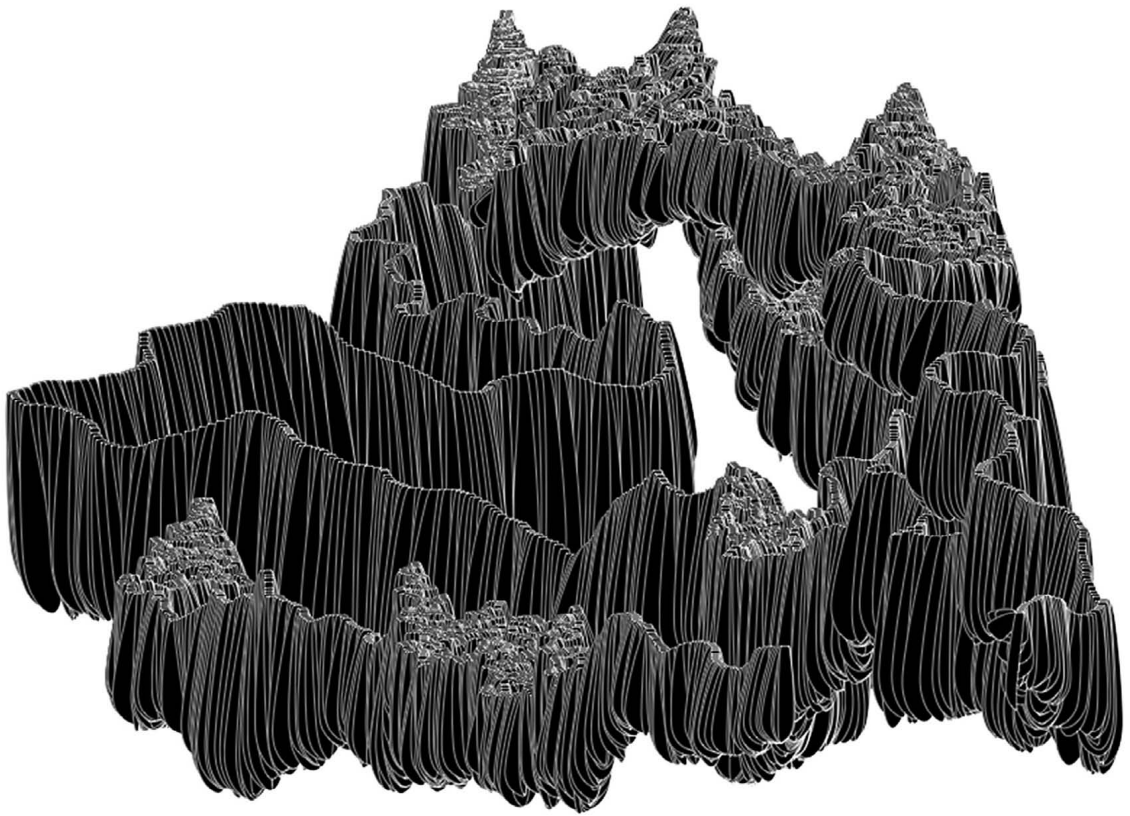
2 The elements should only be drawn if the pen is not completely perpendicular to the tablet and the pressure is greater than zero.

3 Declaration of variables that are needed to describe the shape. Using the `random()` function, the shapes are varied slightly in their length and width when calculated.

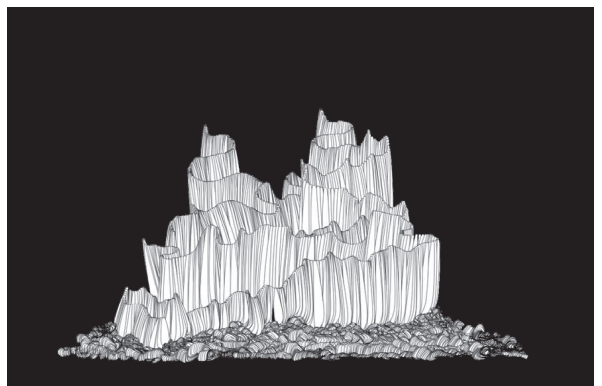
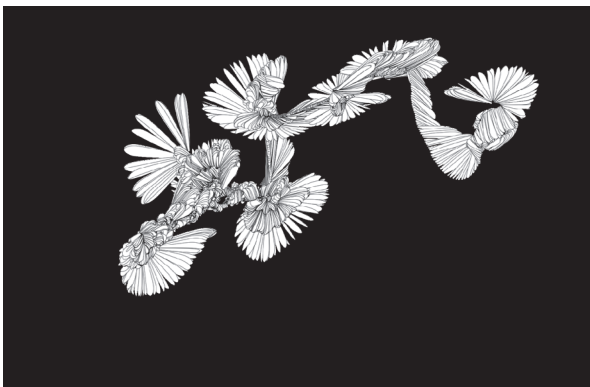
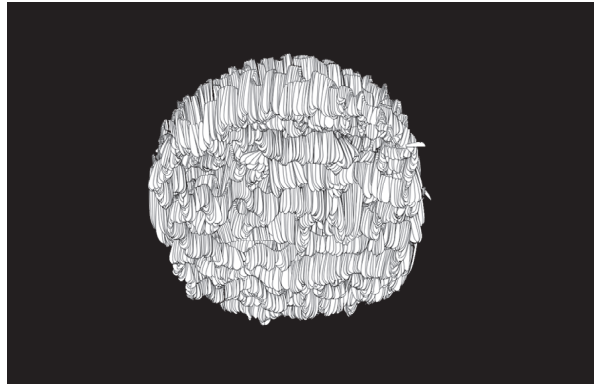
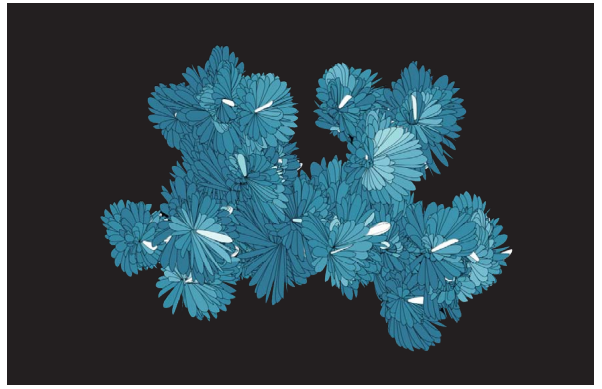
4 Two arrays, `pointsX` and `pointsY`, are created for the shape's contour points and then filled with values.

5 The shape is drawn.

! For drawing the closed curve, see:
[→P.2.2.3 Shapes from agents](#)



→ P_2_3_5_01_TABLET → Illustration: Pau Domingo The basic forms, which are superimposed on each other in rapid succession, lead to the creation of complex organic forms.



→ P_2_3_5_01_TABLET → Illustrations: **Pau Domingo, Franz Stämmele, and Jana-Lina Berkenbusch**
From birds to mountain ranges to abstract organic forms, the pen tablet inspires the creation of unique illustrations.

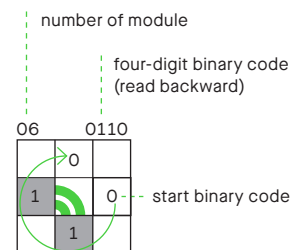
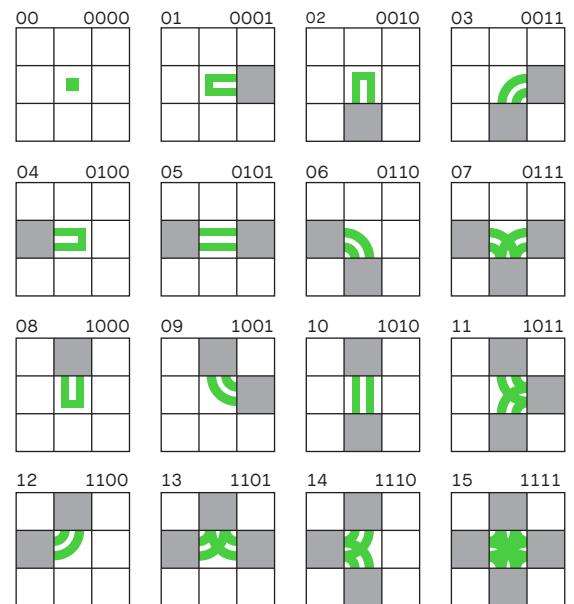
P.2.3.6 Drawing with complex modules

Together they are strong: simple modules become supercharacters. Using combinatorics and complex modules as paintbrushes, each module is defined in relation to its four neighbors, generating supercharacters. Depending on the module repertoire and constellation, many different character groups can be generated.

→ P_2_3_6_01

With the mouse, the user draws a grid and positions various SVG modules in its fields. The only information that exists for each field is whether it is empty or filled. When the grid is displayed, the filled or empty state of each field's neighbors becomes apparent; according to this, a specific graphic module is selected.

The sixteen possible states of the four neighbors can be easily summarized in a four-digit binary code. The correct SVG module can be determined simply by converting the binary code into a decimal number, without complicated combinatorics.



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

  if (mouseIsPressed) {
    if (mouseButton == LEFT) setTile();
    if (mouseButton == RIGHT) unsetTile();
  }

  if (doDrawGrid) drawGrid();
  drawModules();
}
```

```
function setTile() {
  var gridX = floor(mouseX / tileSize) + 1;
  gridX = constrain(gridX, 1, gridResolutionX - 2);
  var gridY = floor(mouseY / tileSize) + 1;
  gridY = constrain(gridY, 1, gridResolutionY - 2);
  tiles[gridX][gridY] = 1;
}
```

```
function drawModules() {
  for (var gridX=0; gridX<gridResolutionX-1; gridX++) {
    for (var gridY=0; gridY<gridResolutionY-1; gridY++) {
      if (tiles[gridX][gridY] == 1) {
        var NORTH = str(tiles[gridX][gridY - 1]);
        var WEST = str(tiles[gridX - 1][gridY]);
        var SOUTH = str(tiles[gridX][gridY + 1]);
        var EAST = str(tiles[gridX + 1][gridY]);

        var binaryResult = NORTH + WEST + SOUTH + EAST;
        var decimalResult = parseInt(binaryResult, 2);

        var posX = tileSize * gridX - tileSize / 2;
        var posY = tileSize * gridY - tileSize / 2;

        image(modules[decimalResult], posX, posY,
              tileSize, tileSize);
        ...
      }
    }
  }
}
```

Mouse: Drag: Draw modules

Drag right mouse button: Delete modules

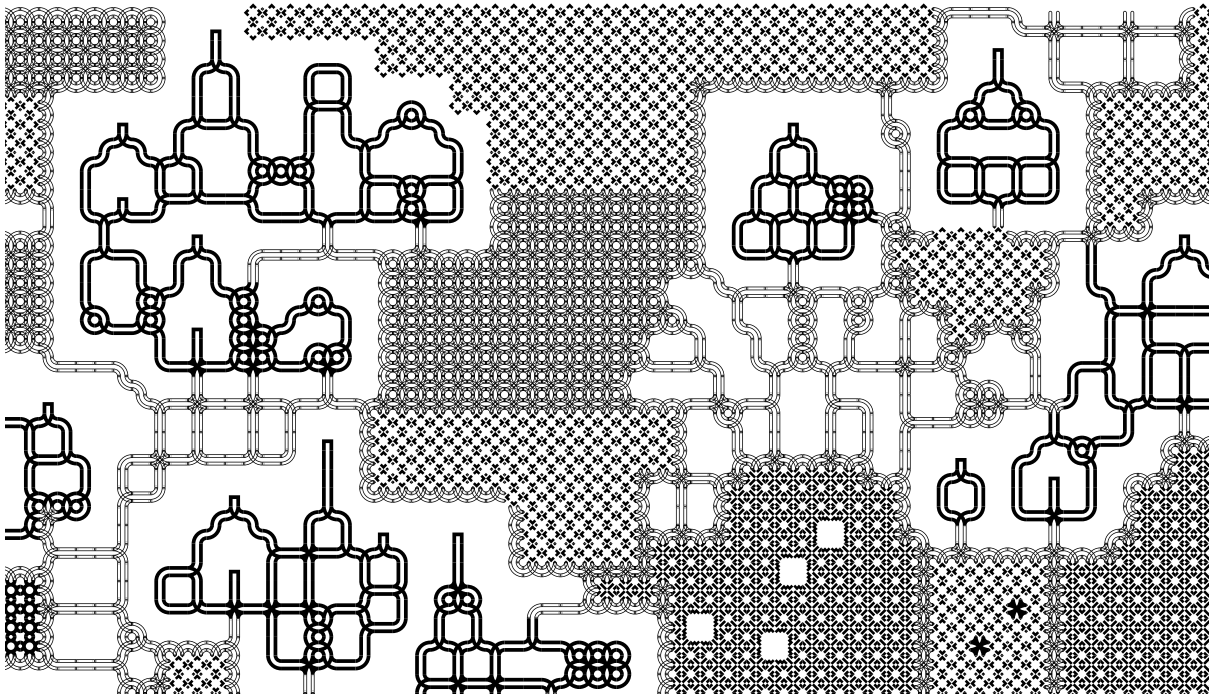
Keys: DEL: Clear canvas

G: Show grid on/off

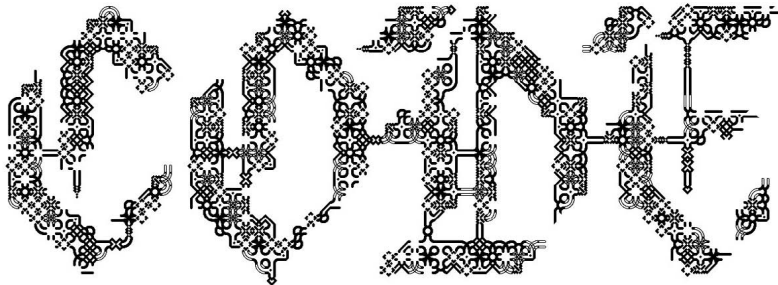
D: Show module values on/off

S: Save image

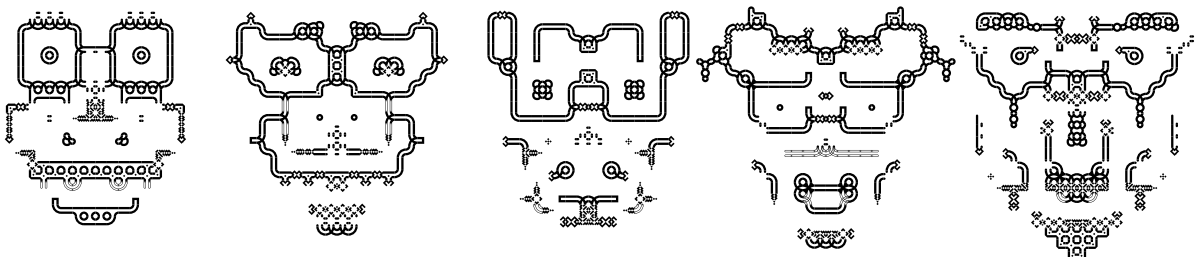
- 1 Using the functions `setTile()` and `unsetTile()`, the state of a field in the grid is set to 1 or 0.
- 2 Using the function `drawModules()`, all SVG modules are displayed. The underlying grid can be drawn with the function `drawGrid()`.
- 3 The mouse position is transferred to the corresponding grid field square in the grid (`gridX`, `gridY`).
- 4 The state of all the fields in the grid is saved in the two-dimensional `tiles` array. The clicked-on field is set to the value 1.
- 5 All tiles are processed. Only fields whose state is 1 (i.e., filled ones) are taken into account.
- 6 The state of the four neighbors is queried, converted into strings, and concatenated. Consequently, `binaryResult` contains a sequence of four zeros or ones.
- 7 The coded states of the binary representations are converted into a decimal number using the function `parseInt()`.
- 8 The corresponding SVG module is selected with `decimalResult` and drawn on the drawing canvas.



→ P_2_3_6_02 → Illustration: Pau Domingo Patterns are created by filling in adjacent grids.



→ P_2_3_6_02 → Illustration: Cedric Kiefer The tiles here are used as ornamental structures.



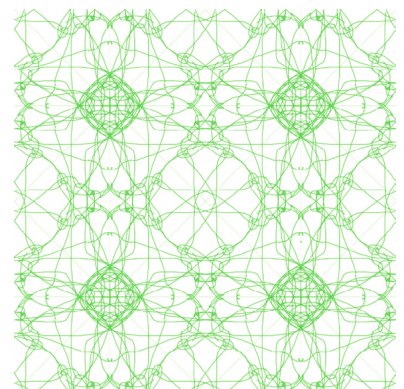
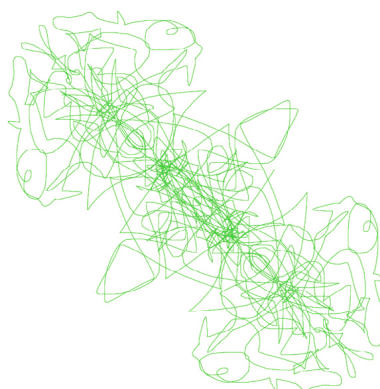
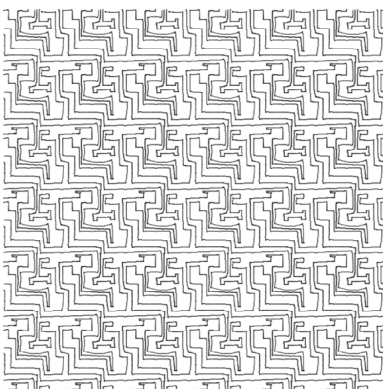
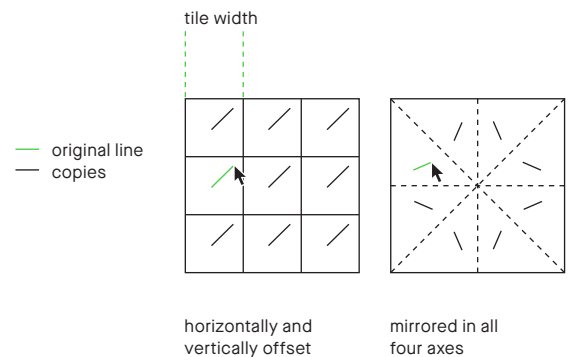
→ P_2_3_6_02 → Illustration: Cedric Kiefer Symmetrical masklike forms.

P.2.3.7 Drawing with multiple brushes

The arrangement of the different brushes, all of which simultaneously react to the mouse position, gives the impression of drawing in a hall of mirrors. The horizontal and vertical mirroring of each brush movement generates kaleidoscopic results.

→ P_2_3_7_01

Creating a paint program is very simple: in each frame, a line is drawn between the current mouse position and the one in the preceding frame. This small line can easily be drawn several times, either offset as exact copies horizontally and vertically or as reflections: horizontal, vertical, or diagonal—or both offset and reflected at the same time.



→ P_2_3_7_01 Three different ways to use mirror axes and repetitions.

```

function setup() {
  canvasElement = createCanvas(800, 800);
  ...
  img = createGraphics(width, height);
  ...
}

function draw() {
  background(255);
  image(img, 0, 0);
  ...
  if (mouseIsPressed && mouseButton == LEFT) {
    var w = width / penCount;
    var h = height / penCount;
    var x = mouseX % w;
    var y = mouseY % h;
    var px = x - (mouseX - pmouseX);
    var py = y - (mouseY - pmouseY);

    for (var i = 0; i < penCount; i++) {
      for (var j = 0; j < penCount; j++) {
        var ox = i * w;
        var oy = j * h;

        img.line(x+ox, y+oy, px+ox, py+oy);
        if (mh || md2 && md1 && mv)
          img.line(w-x+ox, y+oy, w-px+ox, py+oy);
        if (mv || md2 && md1 && mh)
          img.line(x+ox, h-y+oy, px+ox, h-py+oy);
        if (mv && mh || md2 && md1)
          img.line(w-x+ox, h-y+oy, w-px+ox, h-py+oy);

        if (md1 || md2 && mv && mh)
          img.line(y+ox, x+ox, py+ox, px+oy);
        if (md1 && mh || md2 && mv)
          img.line(y+ox, w-x+oy, py+ox, w-px+oy);
        if (md1 && mv || md2 && mh)
          img.line(h-y+ox, x+oy, h-py+ox, px+oy);
        if (md1 && mv && mh || md2)
          img.line(h-y+ox, w-x+oy, h-py+ox, w-px+oy);
      }
    }
    ...
  }
}

```

Mouse: Drag: Draw

Keys: 1-4: Various reflections on/off

5-0: Colors

Arrow ↓/↑: Line width -/+

Arrow ←/→: Number of brushes -/+

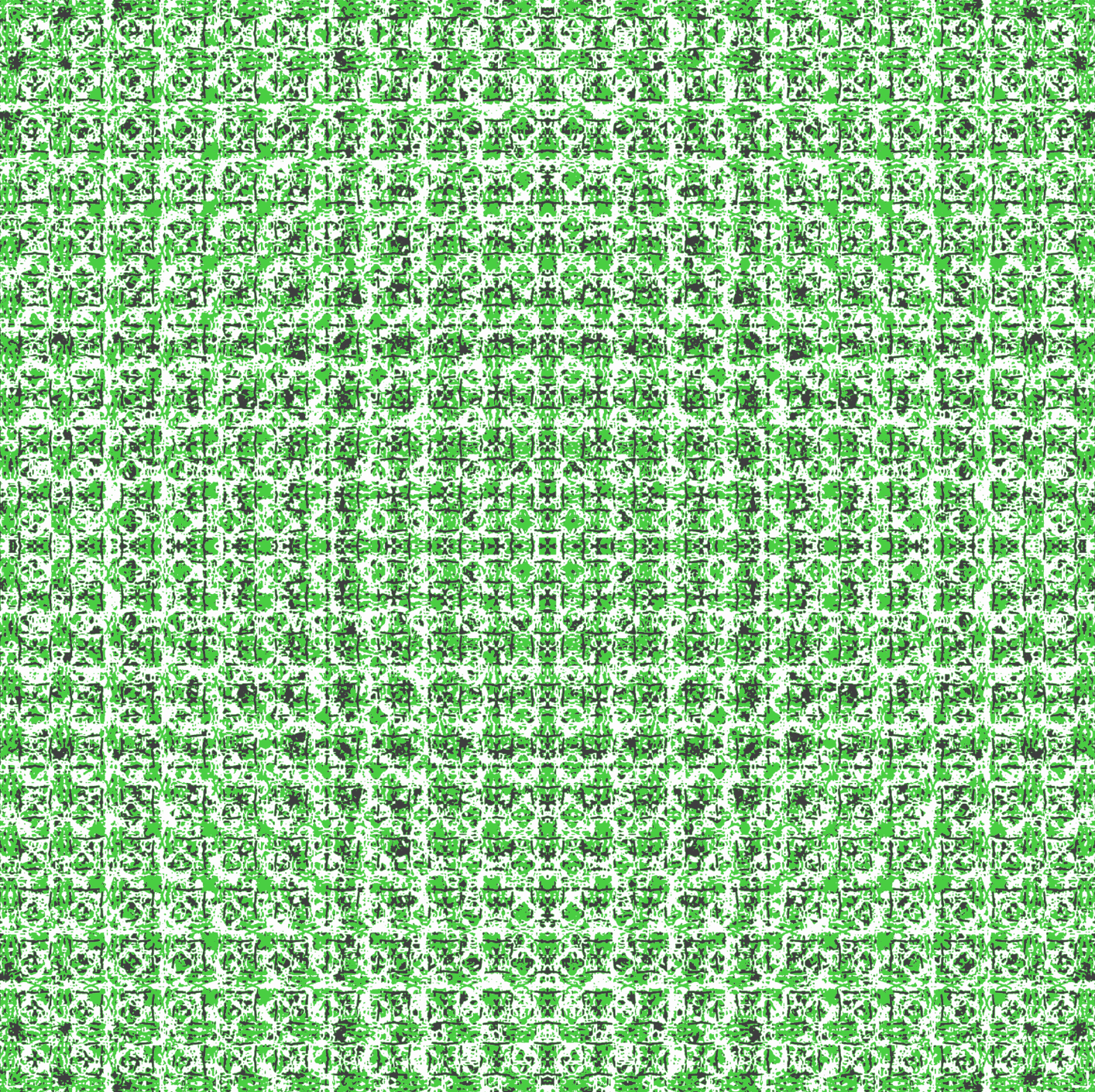
DEL: Clear canvas

D: Display mirror axes on/off

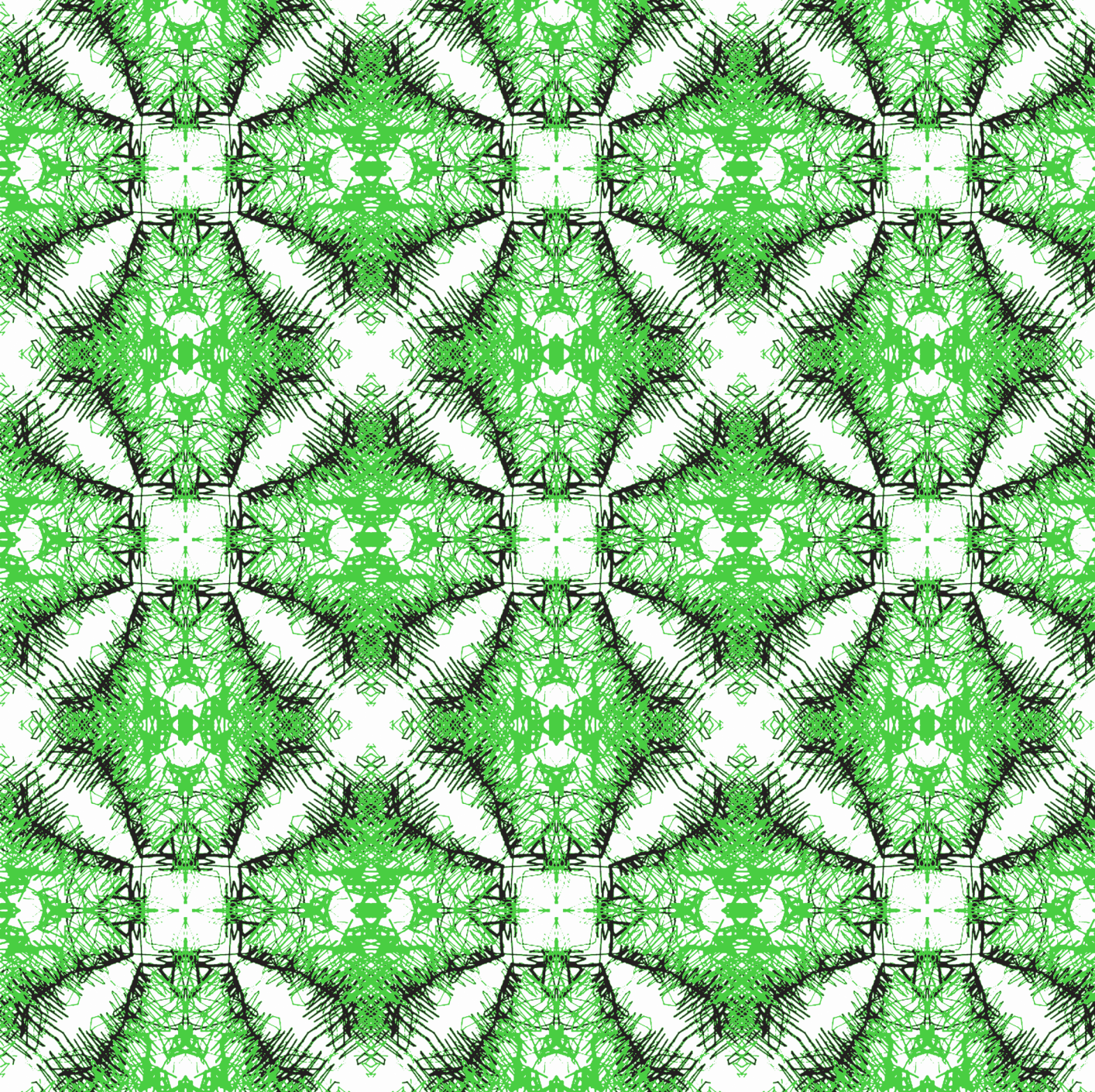
G: GIF recording start/stop

S: Save image

- 1 This sketch should be used only on a square drawing canvas. Otherwise, the diagonal reflections will cause problems.
- 2 The function `createGraphics()` creates an image that will later be drawn. This allows the mirror axes to be displayed and hidden again.
- 3 In each frame, the drawn picture is first copied to the drawing canvas.
- 4 The `penCount` variable determines how many virtual brushes are generated in the `x` and `y` directions. A value of three produces nine brushes. The variables `w` and `h` are the width and height of a tile in which a brush moves.
- 5 The coordinate `x, y` is the position of the brush in the upper left tile.
- 6 The variables `pmouseX` and `pmouseY` are provided by p5.js and always contain the mouse position in the previous frame.
- 7 The values `ox` and `oy` represent the `x` and `y` offset for a brush.
- 8 In all cases, a line is drawn between the current and previous mouse positions.
- 9 If `mh` is true in value (it should be mirrored horizontally), the `x` position is subtracted from the tile width. Incidentally, a horizontal reflection also appears when mirroring vertically and around both diagonal axes.
- 10 One-diagonal reflection is most easily generated by swapping `x` and `y`.



→ **P_2_3_7_02_TABLET** The number of tiles was constantly changed for this picture, and the graphics were often painted over with black, green, and white.



→ **P_2_3_7_02_TABLET** In the version for the pen tablet, the line weight responds to the pen's pressure.

P.3



Type

In the Shape chapter, we demonstrated how shapes can be generated using the principles of repetition (grid), iteration (agents), and interaction (drawing). This chapter is devoted to a special kind of form that is also extremely important in design: typography. Using various methods—from the visual analysis of a text to the outlines of a character—typography will be viewed in the following examples in the context of generative design.

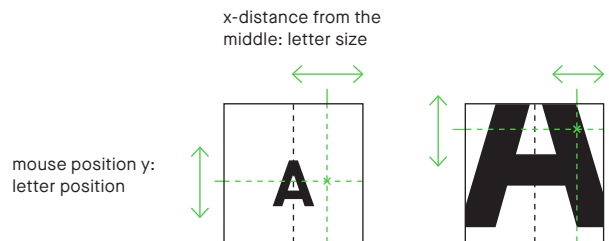
P.3	Type	150
P.3.0	Hello, type	152
P.3.1	Text	154
P.3.1.1	Writing time-based text	154
P.3.1.2	Text as blueprint	156
P.3.1.3	Text image	160
P.3.1.4	Text diagram	166
P.3.2	Font outline	170
P.3.2.1	Dissolving the font outline	170
P.3.2.2	Varying the font outline	174
P.3.2.3	Font outline from agents	178
P.3.2.4	Parallel font outlines	180
P.3.2.5	Kinetic font	184

P.3.0 Hello, type

Letters become spaces. In generating a vector-based font, you can directly influence numerous parameters and design with letters in time and space. Traces of the emergence of the character and the interactive manipulation of its size and position can be made visible.

→ P_3_0_01

The size of the letter is controlled with the horizontal movement of the mouse, and its vertical movement moves the letters up and down. The letter leaves a trail when the mouse button is held down.



→ P_3_0_01 The letter leaves the tracks of its changes, then becomes unrecognizable and generates new forms.

```
1 var font = "sans-serif";
   var letter = "A";
```

```
function setup() {
  createCanvas(windowWidth, windowHeight);
  background(255);
  fill(0);
```

```
2   textFont(font);
   textAlign(CENTER, CENTER);
}
```

```
function mouseMoved() {
  clear();
  textSize((mouseX - width / 2) * 5 + 1);
  text(letter, width / 2, mouseY);
}
```

```
4 function mouseDragged() {
  textSize((mouseX - width / 2) * 5 + 1);
  text(letter, width / 2, mouseY);
}
```

Mouse: Position x: Size
 Position y: Position
 Drag: Draw

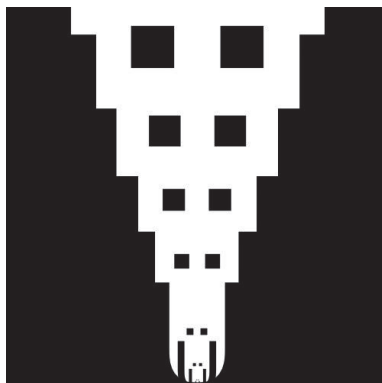
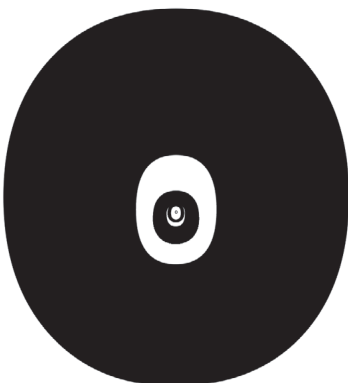
Keys: A-Z: Letter selection
 CTRL: Save image

1 The name of the font to use is saved in the variable `font`.

2 The function `textFont()` makes it the current font. The horizontal and vertical alignment can be specified with `textAlign()`.

3 When the mouse is moved, the letter size changes according to the value of the horizontal mouse position. The `letter` is positioned horizontally in the middle of the display window `width / 2`, vertically in the position `mouseY`, and displayed using the `text()` command.

4 This also occurs when the mouse is moved with the mouse button held down, but now the background does not get a new color and the letter leaves a trail.



P.3.1.1 Writing time-based text

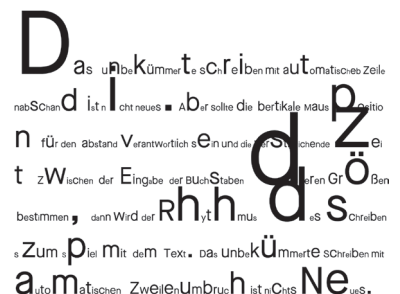
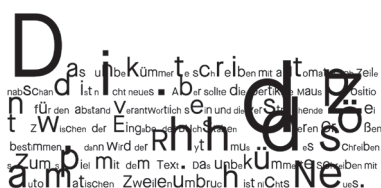
Composing text with automatic line breaks is nothing new. But when the vertical mouse position is responsible for the leading (the space between lines), and the elapsed time before entering each letter determines its size, then the rhythm of writing begins to interact with the text.

→ P_3_1_1_01

When typing, the virtual “pen tip” moves from left to right over the drawing canvas. When it reaches the right border, it starts again from the beginning and skips a line. Leading is defined by the vertical mouse position. The time between the individual keystrokes is measured. The greater the time interval, the larger the entered letter.

mouse position y:
line spacing

The more time that passes
before a letter is typed,
the larger it becomes.



→P_3_1_1_01 The vertical mouse position defines the leading (line spacing).
Different levels of legibility result.

```

function draw() {
  ...
  spacing = map(mouseY, 0, height, 0, 120);
  translate(0, 200 + spacing);

  var x = 0;
  var y = 0;
  var fontSize = 20;

  for (var i = 0; i < textTyped.length; i++) {
    fontSize = fontSizes[i];
    textFont(font, fontSize);
    var letter = textTyped.charAt(i);
    var letterWidth = textWidth(letter) + tracking;

    if (x + letterWidth > width) {
      x = 0;
      y += spacing;
    }

    text(letter, x, y);
    x += letterWidth;
  }

  var timeDelta = millis() - pMillis;
  newFontSize = map(timeDelta, 0, maxTimeDelta,
    minFontSize, maxFontSize);
  newFontSize = min(newFontSize, maxFontSize);

  fill(200, 30, 40);
  if (int(frameCount / 10) % 2 == 0) fill(255);
  rect(x, y, newFontSize / 2, newFontSize / 20);
}

```

```

function keyTyped(){
  if(keyCode >= 32){
    textTyped += key;
    fontSizes.push(newFontSize);
  }
}

```

```

function keyPressed() {
  ...
  pMillis = millis();
}

```

Mouse: Position x: Size
 Position y: Position
Keys: A–Z: Letter selection
 CTRL: Save image

- 1 The variable `textTyped` contains the entered text. This is now processed letter by letter.
- 2 The `fontSize` is taken from the array `fontSizes[]` from index `i`, and the font is set to this size.
- 3 Now the letter at index `i` is selected from the string `textTyped` and saved in `letter`. In addition, the letter width `textWidth(letter)` is increased by the value `tracking`.
- 4 When the current position plus the character width exceeds the width of the drawing canvas, a line break occurs. Thus `x` is reset to 0 and the vertical position `y` is increased by `spacing`.
- 5 The letter is drawn at the position `x, y`.
- 6 After all letters have been drawn, a blinking cursor is displayed. Over time, this is supposed to grow, so `timeDelta`—the time that has elapsed since the last typing operation—must be determined. The current time is obtained in milliseconds with the `millis()` function. The value `pMillis`, which was saved with the last keystroke, is subtracted from this. This time difference can now be converted to the range `minFontSize` to `maxFontSize`.
- 7 This value is used to draw a rectangle at the current drawing position.
- 8 By pressing a key, the typed letter of the character string `textTyped` is attached and the array `fontSizes[]` is grown by appending the new letter size in `newFontSize`.
- 9 The current time is saved in `pMillis` so that the time of the last keystroke is always available.

P.3.1.2 Text as blueprint

In this example, time no longer determines letter size. Rather, certain letters modify, for instance, the text orientation. In this program, every character is translated using a visual rule. The source text thus becomes the blueprint for the composition.

→ P_3_1_2_01

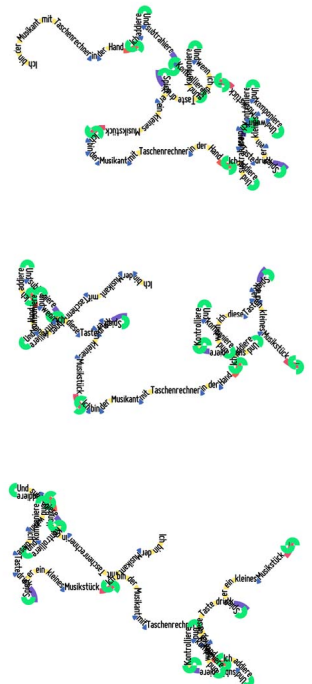
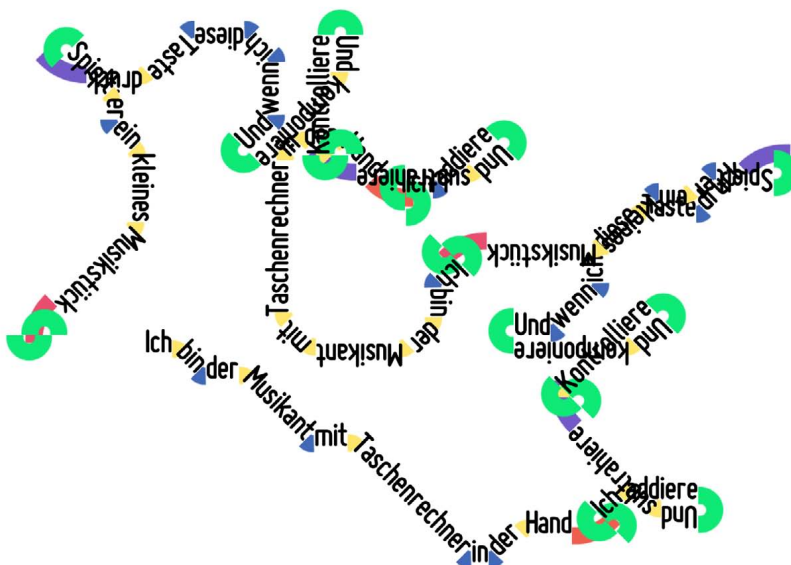
The user can freely enter the text using the keyboard. Every character is translated by a fixed rule in the program that specifies what will be drawn and how position and size are to be altered.

Every entry can be undone using the back-space or delete key.

In this example, some of the characters are replaced by loaded SVG modules.



Key	Translation
A	→ Enter A → Shift writing position
B	→ Enter B → Shift writing position
Space	→ Draw graphic image → Turn 45° → Shift writing position
C	→ Enter C → Shift writing position
Comma	→ Draw graphic image → Turn 45° → Shift writing position



→ **P_3-1_2_01** The program interprets a text (here the lyrics of the song *Taschenrechner* by Kraftwerk) as a floor plan. Using the ALT key, different RandomSeeds can be generated that constantly give the text a new appearance, since there are two possible random directions for the space key.


```

function draw() {
  ...
  1 translate(centerX, centerY);
  scale(zoom);

  2 for (var i = 0; i < textTyped.length; i++) {
    var letter = textTyped.charAt(i);
    3 var letterWidth = textWidth(letter);

    4 switch (letter) {
    5 case ' ':
      var dir = floor(random(0, 2));
      if (dir == 0) {
        image(shapeSpace, 1, -15);
        translate(4, 1);
        rotate(QUARTER_PI);
      }
      if (dir == 1) {
        image(shapeSpace2, 1, -15);
        translate(14, -5);
        rotate(-QUARTER_PI);
      }
      break;

    6 case ',':
      image(shapeComma, 1, -15);
      translate(35, 15);
      rotate(QUARTER_PI);
      break;
      ...
    7 default:
      fill(0);
      text(letter, 0, 0);
      translate(letterWidth, 0);
    }
  }

  // blinking cursor after text
  fill(0);
  8 if (int(frameCount / 6) % 2 == 0) rect(0, 0, 15, 2);
}

```

Mouse: Drag: Move drawing canvas

Keys: A–Z: Input text

Punctuation mark: Curves

Space bar: Curve with random position

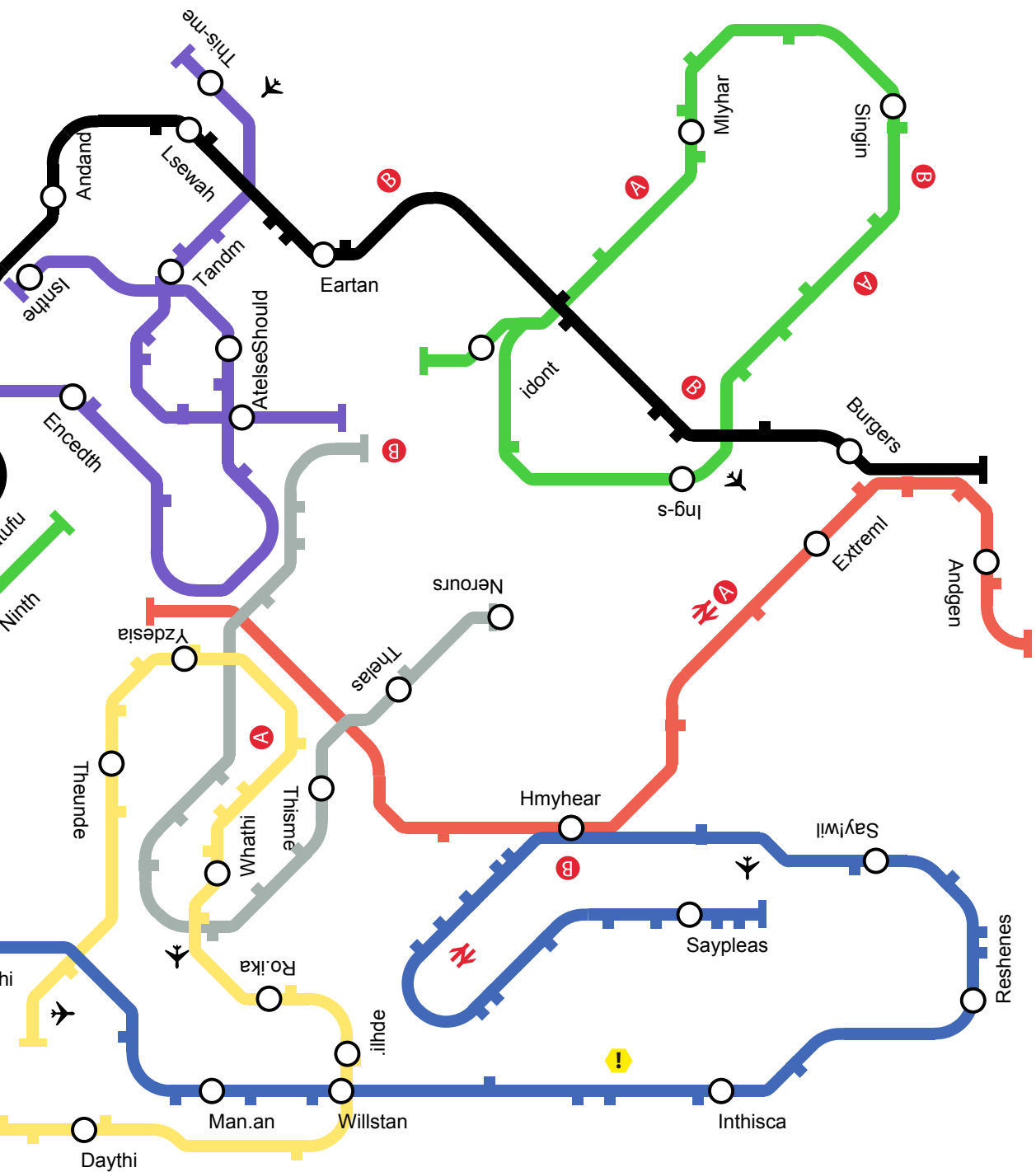
DEL: Delete letters

Arrow ↓/↑: Zoom into the drawing canvas

ALT: New random layout

CTRL: Save image

- 1 The origin of the coordinate system is moved to the point (centerX, centerY) before the text is displayed. This flexible definition allows the point to be moved via mouse interaction.
- 2 All typed letters are processed sequentially.
- 3 The character width of each letter is calculated so that the writing position can later be offset to the correct distance.
- 4 The heart of the program is the set of rules that specifies how the individual characters impact the image and writing behavior. For this purpose, the current letter is distinguished using the switch() command.
- 5 A space is translated as follows: depending on the random value dir, one of the two loaded SVG modules shapeSpace or shapeSpace2 is drawn; the writing position is adjusted using translate(); and the writing direction is rotated 45° to either the left or right using rotate().
- 6 For other special characters (in this case, the comma), SVGs are also loaded in variables. When one of these characters appears, the corresponding module is drawn, and the writing position and direction are adjusted.
- 7 With all other characters the letter is drawn and the writing position shifted by the letterWidth.
- 8 Display of the blinking cursor: this construction uses the p5.js variable frameCount, which automatically increments in each frame, and the modulo operator % to produce alternately the values 0 and 1. This is used to fade the cursor in and out.



- all keys without special features
- space
- comma, question mark, and exclamation point
- period

- letters "a" and "u"
- letter "o"
Intisca (station name, the last seven typed letters)
- hyphen

- colon
- letter "x"
- letter "z"
- plus sign

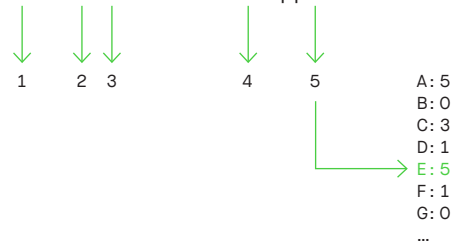
P.3.1.3 Text image

Which characters appear, and how often? The properties of an analyzed text can generate images. The number of appearances in a text is calculated for each character and determines its appearance. The color of the letters, for instance, corresponds to how frequently they appear. Ultimately we do not even need the letters anymore.

→ P_3_1_3_01

A text is processed character by character, and each character's corresponding counter is advanced. That results in a list of counters representing the frequency of each character. These values can now be used as parameters for displaying the text.

How often does each character appear?



```
1 ...  
  var alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜß.,;!? ";  
  var counters = [];  
  ...
```

```
2 function preload() {  
  joinedText = loadStrings("data/faust_kurz.txt");  
}
```

```
3 function setup() {  
  createCanvas(620, windowHeight);  
  ...  
  joinedText = joinedText.join(" ");  
  for (var i = 0; i < alphabet.length; i++) {  
    counters[i] = 0;  
  }  
4  countCharacters();  
}
```

1 The character string `alphabet` determines which characters are to be counted. The `counters` array is initialized with the string's length and thereby provides each character with a counter.

2 The text to be analyzed is loaded with the `loadStrings()` function.

3 The individual texts are now located in an array of strings. Since it is more practical to work with a continuous text, the lines are joined together using the `join()` function.

4 The `countCharacters()` function is called to determine the frequency.

```
function countCharacters() {
  for (var i = 0; i < joinedText.length; i++) {
    var c = joinedText.charAt(i);
    var upperCaseChar = c.toUpperCase();
    var index = alphabet.indexOf(upperCaseChar);
    if (index >= 0) counters[index]++;
  }
}
```

```
function draw() {
  ...
  posX = 20;
  posY = 40;

  for (var i = 0; i < joinedText.length; i++) {
    var upperCaseChar = joinedText.charAt(i).toUpperCase();
    var index = alphabet.indexOf(upperCaseChar);
    if (index < 0) continue;

    if (drawAlpha) {
      fill(87, 35, 129, counters[index] * 3);
    } else {
      fill(87, 35, 129);
    }

    var sortY = index * 20 + 40;
    var m = map(mouseX, 50, width - 50, 0, 1);
    m = constrain(m, 0, 1);
    var interY = lerp(posY, sortY, m);

    text(joinedText.charAt(i), posX, interY);

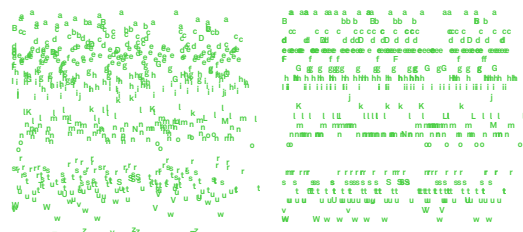
    posX += textWidth(joinedText.charAt(i));
    if (posX >= width - 200 && upperCaseChar == " ") {
      posY += 30;
      posX = 20;
    }
  }
}
```

Mouse: Position x: Survivors from the normal and sorted text

Keys: 1: Switch alpha mode

S: Save image

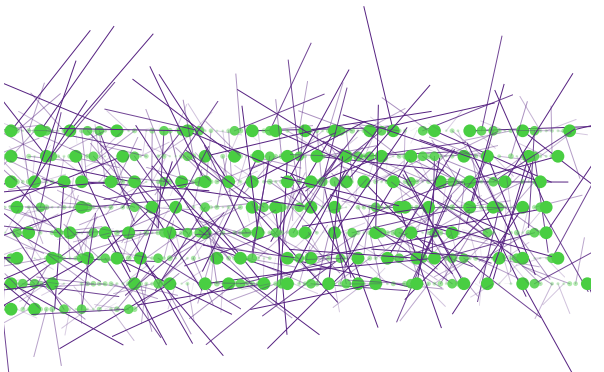
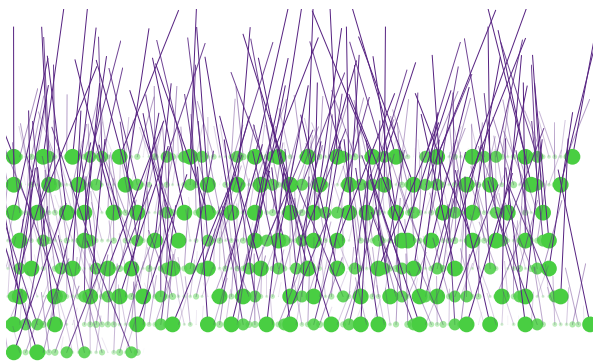
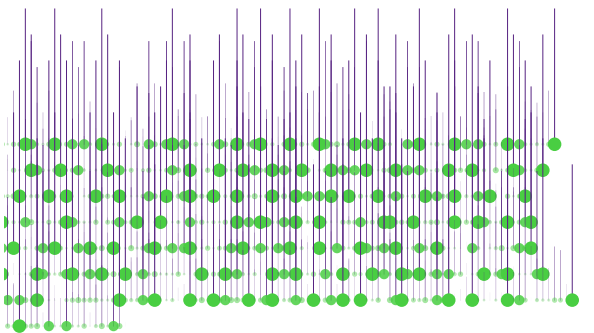
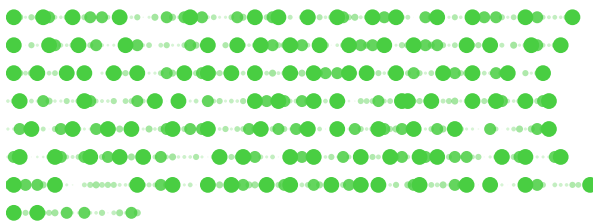
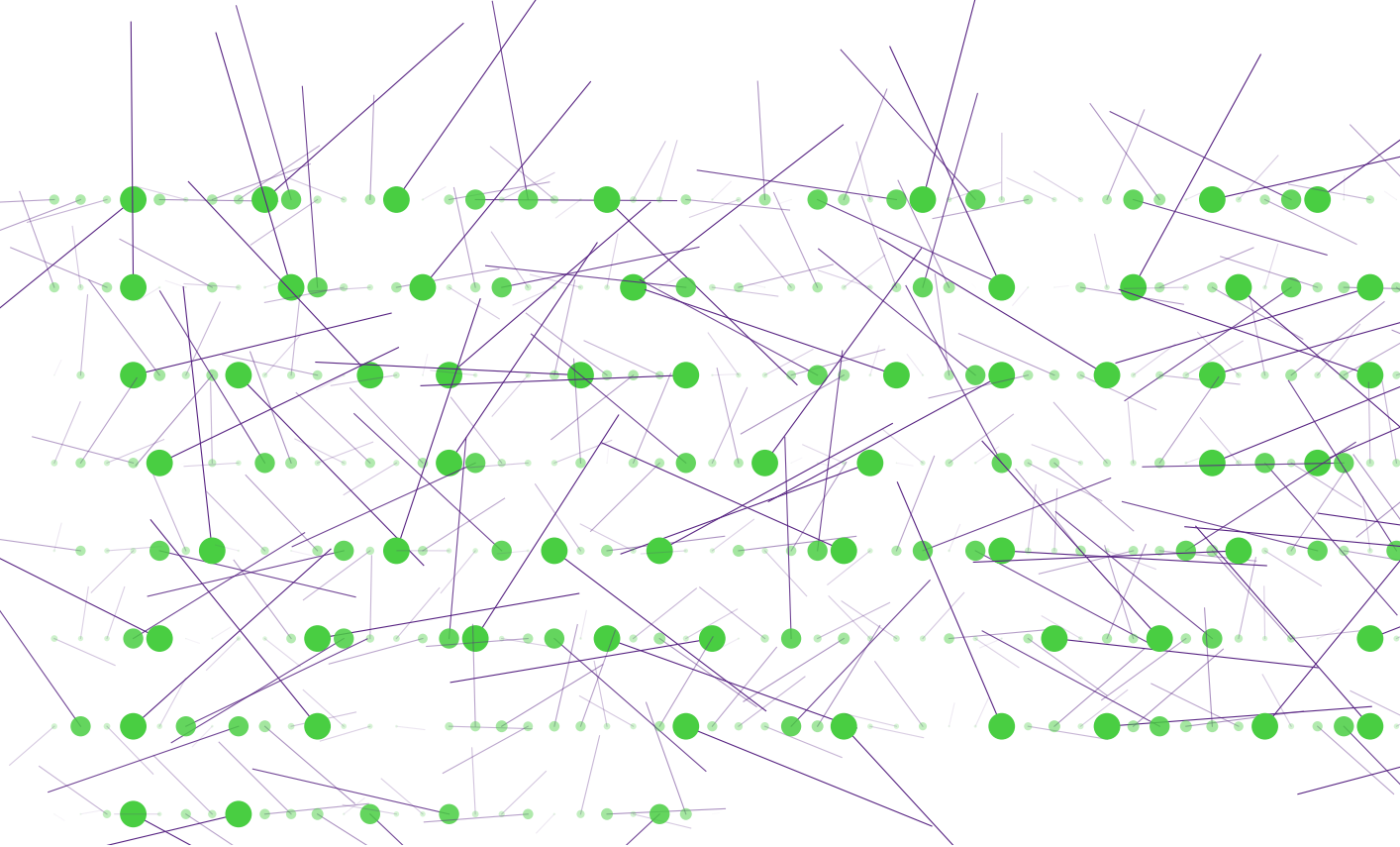
Ihr naht euch wieder, schwankende Gestalten, Die früh
sich einst dem trüben Blick gezeigt. Versuch ich wohl,
euch diesmal festzuhalten? Fühl ich mein Herz noch jenem
Wahn geneigt? Ihr drängt euch zu! nun gut, so mögt ihr
walten, Wie ihr aus Dunst und Nebel um mich steigt; Mein
Busen fühlt sich jugendlich erschüttert Vom Zauberhauch,
der euren Zug umwirrt. Ihr bringt mit euch die Bilder
froher Tage, Und manche liebe Schatten steigen auf; Gleich
einer alten, halbverklungenen Sage Kommt erste Lieb- und
Freundschaft mit herauf; Der Schmerz wird neu, es wiederholt
die Klage Des Lebens labyrinthisch irren Lauf, Und nennt
die Guten, die, um schöne Stunden Vom Glück getauscht,
vor mir hinweggeschwunden.



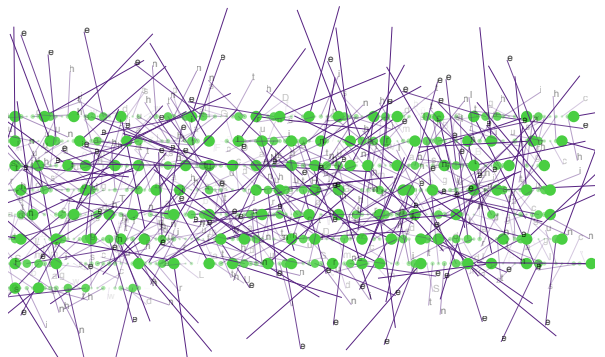
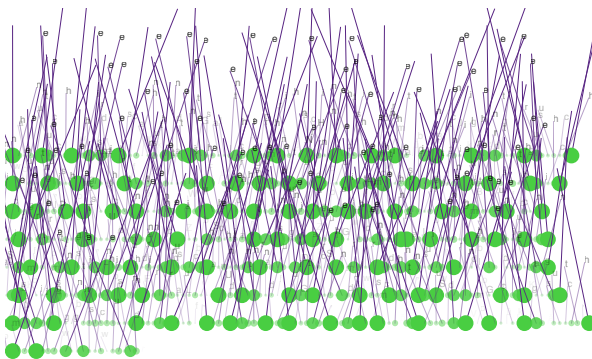
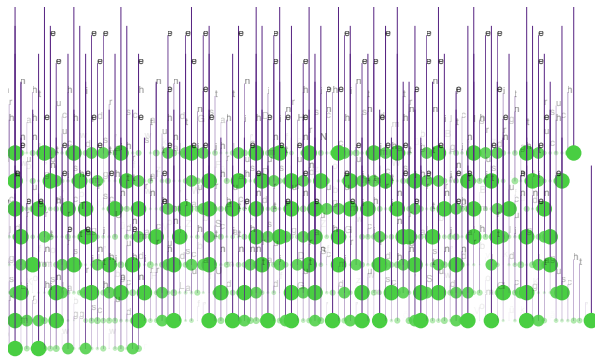
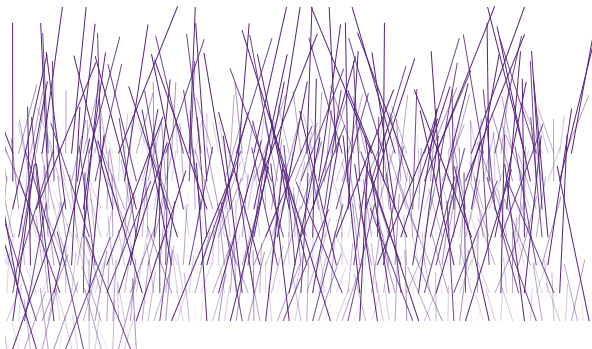
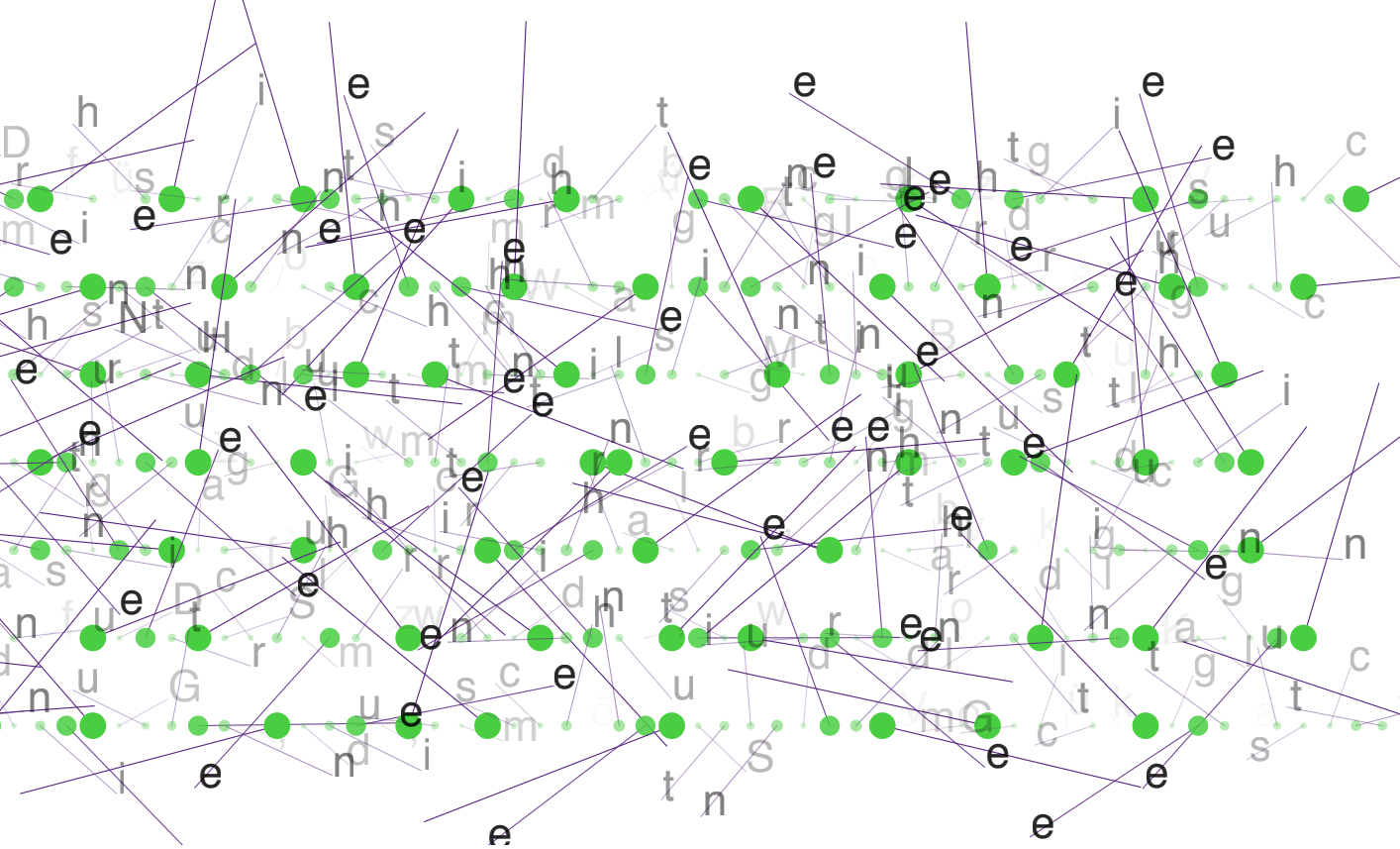
→ **P_3.1.3_01** The horizontal mouse position controls whether the letters are displayed in the normal text position or sorted.

- 5** The text is processed: a character is taken from the text using `charAt()` and converted into an uppercase letter using the `toUpperCase()` function.
- 6** The function `indexOf()` is used to find where each letter from the given character string is positioned in the `alphabet`. If the letter is found, the `index` for the corresponding letter is incremented.
- 7** The variables `posX` and `posY` are initialized with the values at the start.
- 8** When drawing, the text is processed from the beginning each time, and, just as in the `countCharacters()` function, the index of the current character is determined for the counter array. If the character is not found—`index < 0`—the drawing process for this character is canceled using the `continue` command.
- 9** When the drawing mode `drawAlpha` is engaged, the transparency of the fill color is dependent on the frequency of the character and set with `counters[index]`.
- 10** The variable `sortY` is introduced to sort the characters. This represents the y-coordinate of the line to which the character should go or migrate. For A, it is 40. For B, the value is 60, etc.
- 11** The mouse position is converted into a number `m` between 0 and 1 that will serve as an interpolation variable. Using the `lerp()` function, an interpolation between `posY` and `sortY` is carried out, and the calculated value `interY` is used to position the character.

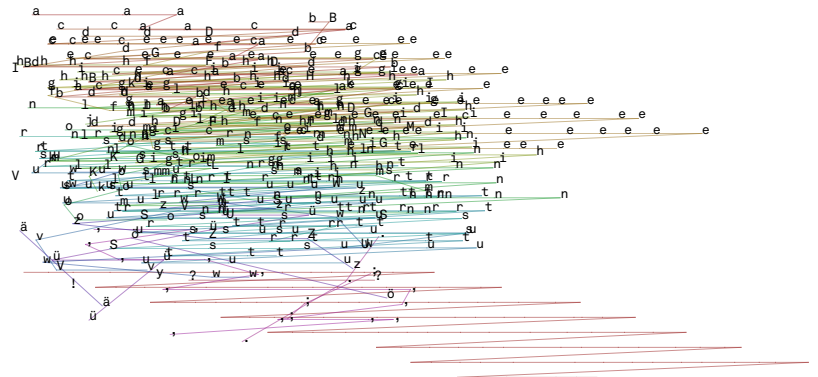
- 12** Now only the writing position has to be updated. The value `posX` is increased by the character width; if this value approximates the right border of the display and the current character is a space, then the line break takes place. The value `posY` is increased by the line spacing and `posX` set back to the left.



→ **P.3_1_3.03** The frequency of the letters in a text are coded here several times and in various combinations: by the size and transparency of the green circles, by the length of the purple lines, and by the opacity of the letters themselves.



Ihr naht euch wieder, schwankende Gestalten, Die früh sich einst dem trüben Blick gezeigt.
 Versuch ich wohl, euch diesmal festzuhalten? Fühl ich mein Herz noch jenem Wahn geneigt? Ihr
 drängt euch zu! nun gut, so mögt ihr walten, Wie ihr aus Dunst und Nebel um mich steigt; Mein
 Busen fühlt sich jugendlich erschüttert Vom Zauberhauch, der euren Zug umwittert. Ihr bringt
 mit euch die Bilder froher Tage, Und manche liebe Schatten steigen auf; Gleich einer alten,
 halbverklungenen Sage Kommt erste Lieb und Freundschaft mit herauf; Der Schmerz wird neu, es
 wiederholt die Klage Des Lebens labyrinthisch irren Lauf, Und nennt die Guten, die, um schöne
 Stunden Vom Glück getauscht, vor mir hinweggeschwunden.



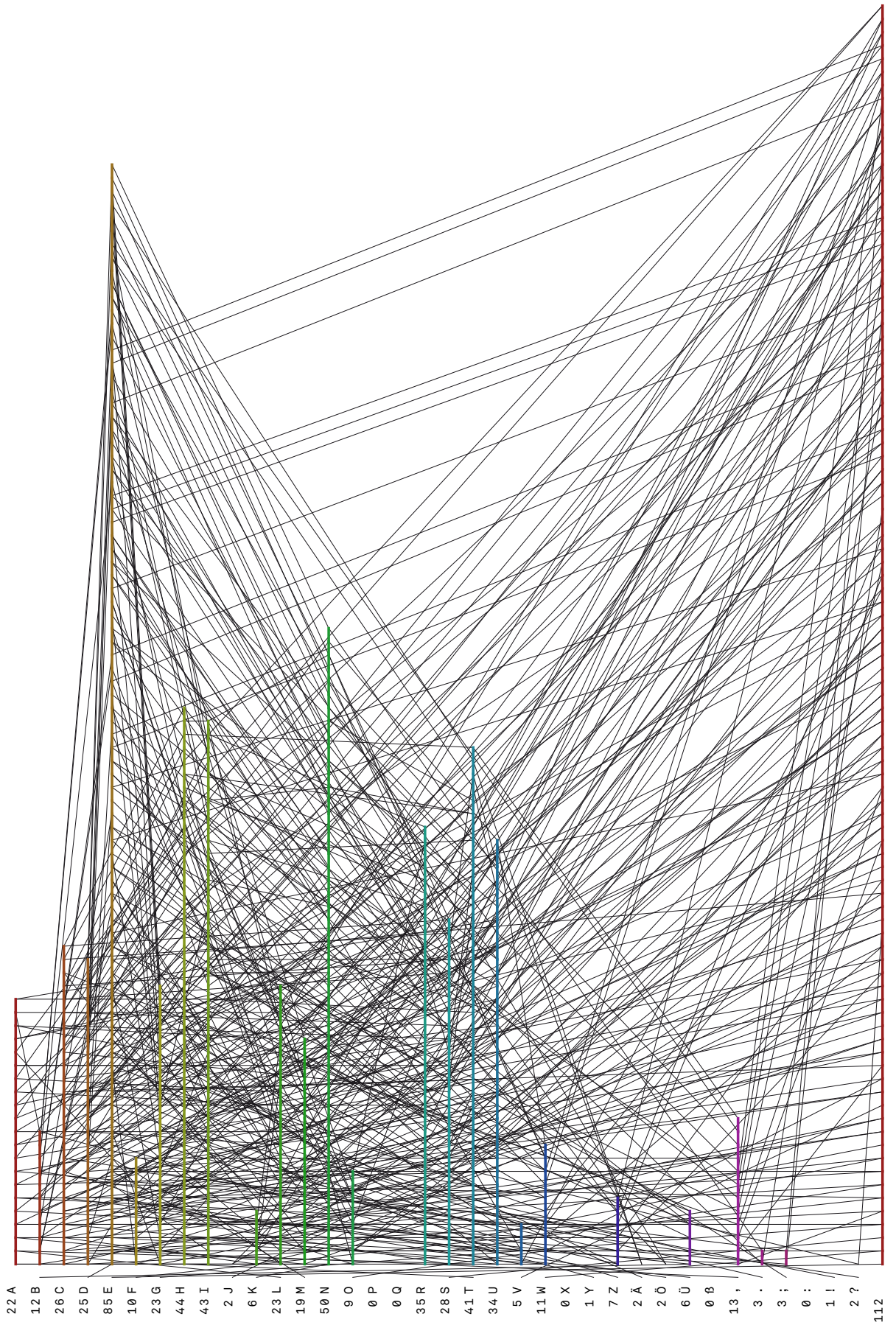
```

22A aaaaaaaaaaaaaaaaaaaaaa
12B bBbBbbBbbbbb
26C ccccccccccccccccccccc
25D ddDdddDdddddDdddDddd
85E eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
10F fFffFffFff
23G GggggggggggggGgggGggg
44H hhhhhhhhhhhHhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
43I IiiiiiiiiiiiIiiiiiiiiIiiiiiiiiiiiiiiiiiii
2J jj
6K kkkKKk
23L llllllllllllllllllllll
19M mmmmmmmMmmmmmmmmmmmm
50N nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
90 oOooooooo
0P
0Q
35R rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
28S sssssssssssssSsSssSssssSss
41T tttttttttttttttttttttttttttttttttttttttttttttttttttttttt
34U uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu
5V VVvVv
11W wwwWwWwWwWwWw
0X
1Y y
7Z zzzzZZz
2Ä ää
2Ö öö
6Ü üüüüüü
0ß
13, ,,,,,,
3. .,.
3; ;+;
0:
1!
2? ??
112

```

→ **P_3_1_3_04** Identical letters are connected by colored lines; the colors of the lines pass through the color wheel once in alphabetical order. The letters are sorted by their frequency when the mouse is moved to the right.

Any kind of letter can be individually switched on and off, allowing, e.g., the frequency of the vowels to be observed separately. The gray line (see illustration on the right), which can be turned on and off using key 2, connects each letter with the letter directly following it. Although hardly visible in the normal text, when arranged by letter, these gray lines create a striking network structure.

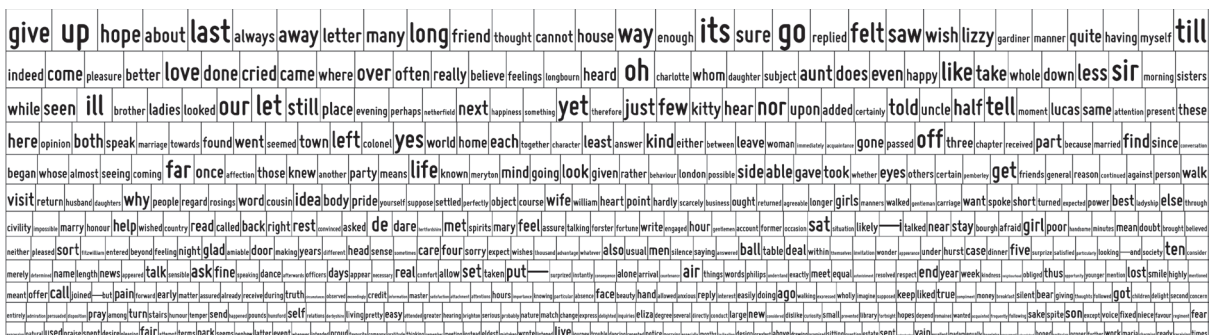
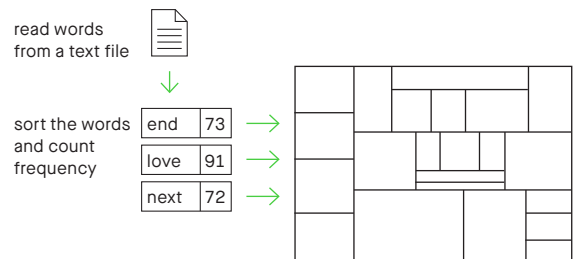


P.3.1.4 Text diagram

What were Jane Austen's favorite words? The possibility of mechanically reading and processing large amounts of text provides considerable room for experimentation. All the words in *Pride and Prejudice*, for instance, can be counted and their frequency represented by elements (in this case, rectangles) of varying sizes to create diagrams that function as static literary criticism.

→ P_3_1_4_01

The aim of a so-called treemap is to divide a rectangle into smaller rectangles according to the frequency with which each word is used in a text. The complete text of Jane Austen's *Pride and Prejudice* is read from a text file and transferred to the Treemap class for visualization. Individual parameters for the TreeMap algorithm can be keyed in.



→ **P_3_1_4_01** Section of a treemap arranged according to frequency, in which only horizontal elements are permitted.

```

1  var mapData = {};
    ...
2  var doSort = true;
    var rowDirection = "both";

    function setup() {
        ...
        joinedText = joinedText.join(" ");
3  var words = joinedText.match(/\w+/g);

4  treemap = new gd.Treemap(1, 1, width - 3, height - 3,
                           {sort:doSort, direction:rowDirection});

        for (var i = 0; i < words.length; i++) {
5  var w = words[i].toLowerCase();
            treemap.addData(w);
        }

        treemap.calculate();
    }

```

```

    function draw() {
        background(255);
        textAlign(CENTER, BASELINE);

6  for (var i = 0; i < treemap.items.length; i++) {
            var item = treemap.items[i];

            fill(255);
            stroke(0);
            strokeWeight(1);
7  rect(item.x, item.y, item.w, item.h);

            var word = item.data;
            textFont(font, 100);
8  var textW = textWidth(word);
            var fontSize = 100 * (item.w * 0.9) / textW;
9  fontSize = min(fontSize, (item.h * 0.9));
            textFont(font, fontSize);

            fill(0);
            noStroke();
10 text(word, item.x + item.w/2, item.y + item.h*0.8);
        }
        ...
    }

```

Keys: R: Random on/off
H: Horizontal arrangement
V: Vertical arrangement
B: Arrangement in both directions
S: Save image

1 The data container `mapData` will later be filled with the loaded text's words and their numbers.

2 The `doSort` and `rowDirection` variables control the various layouts of the Treemap.

3 The complete text is split into individual words using `match()`. The parameter for this function is a regular expression. With its help, very complex text searches can be performed. The one used here is relatively simple: one or more consecutive (+) word characters (\w) are to be searched. The g at the end means "global," i.e., the search should not stop after the first occurrence.

4 The `Treemap` is generated. The first four parameters specify position (x, y), width, and height. In addition, an object is swapped for the display options.

5 The array with the words is processed. Each word is initially converted into lowercase letters so that different capitalizations of the same word (e.g., when it is used as the first word of a sentence) are still recognized. Then, `addData()` adds the word to the treemap and starts the tally with `calculate()`.

6 The Treemap class was used to calculate a treemap element for each word, all of which are now available as `items` in the array.

7 Position, width, and height of the element are automatically available in the variables `x`, `y` and `w` and `h` and can be used to draw the element's frame.

8 Then the font size is set to 100 to determine the width of the word. Using a rule of thirds, the font size can be calculated so that the word fits within the width of the `item.w` rectangle.

9 The word, however, may not be taller than the rectangle.

10 The word is written in the rectangle.

→ **P_3_1_4_01** The frequency of all the words in Jane Austen's *Pride and Prejudice* as a treemap. The algorithm tries to keep the rectangles as square as possible in this arrangement.

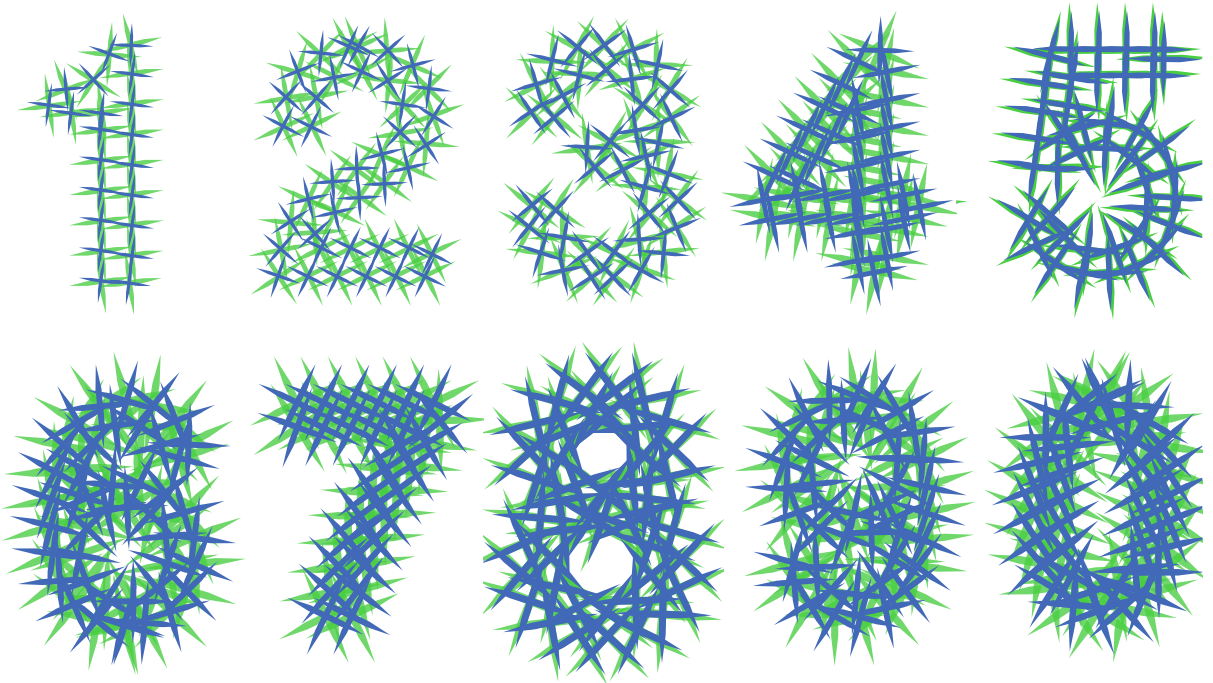
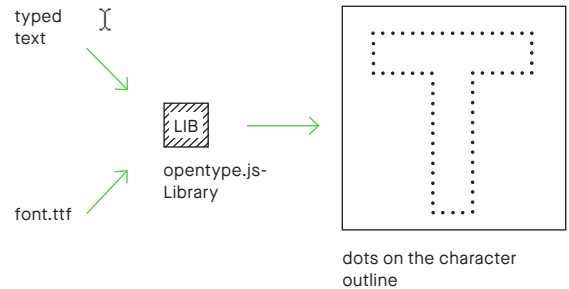
that										which could would									
with										their darcy there every									
have										never think other after might									
they										again being lydia shall thing first									
were										great young about house lizzy quite cried									
been										where often heard happy whole while place still									
from										added kitty uncle these lucas found world speak least									
very										woman leave three whose since began those party going means									
what										known given visit pride heart point girls ought short spoke marry									
your										power right asked write doubt night years table sense sorry usual under									
this										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
them										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
will										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
said										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
such										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
when										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
much										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
more										evil good match two good good good good good good good good good good good good good good good good good good good									
jane										quest good child tale good good good good good good good good good good good good good good good good good good good									
miss										any one good									
than										know given visit pride heart point girls ought short spoke marry									
know										power right asked write doubt night years table sense sorry usual under									
well										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
soon										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
only										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
some										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
time										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
good										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
lady										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
most										evil good match two good good good good good good good good good good good good good good good good good good good									
then										quest good child tale good good good good good good good good good good good good good good good good good good good									
make										any one good									
dear										know given visit pride heart point girls ought short spoke marry									
room										power right asked write doubt night years table sense sorry usual under									
into										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
ever										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
made										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
give										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
hope										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
last										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
away										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
many										evil good match two good good good good good good good good good good good good good good good good good good good									
long										quest good child tale good good good good good good good good good good good good good good good good good good good									
sure										any one good									
felt										know given visit pride heart point girls ought short spoke marry									
wish										power right asked write doubt night years table sense sorry usual under									
till										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
come										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
done										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
love										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
came										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
over										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
does										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
whom										evil good match two good good good good good good good good good good good good good good good good good good good									
aunt										quest good child tale good good good good good good good good good good good good good good good good good good good									
even										any one good									
like										know given visit pride heart point girls ought short spoke marry									
take										power right asked write doubt night years table sense sorry usual under									
down										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
less										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
seen										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
just										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
next										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
upon										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
hear										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
both										evil good match two good good good good good good good good good good good good good good good good good good good									
half										quest good child tale good good good good good good good good good good good good good good good good good good good									
told										any one good									
tell										know given visit pride heart point girls ought short spoke marry									
here										power right asked write doubt night years table sense sorry usual under									
same										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
went										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
home										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
town										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
left										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
each										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
kind										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
gone										evil good match two good good good good good good good good good good good good good good good good good good good									
part										quest good child tale good good good good good good good good good good good good good good good good good good good									
find										any one good									
knew										know given visit pride heart point girls ought short spoke marry									
once										power right asked write doubt night years table sense sorry usual under									
life										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
mind										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
look										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
able										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
side										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
gave										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
took										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
eyes										evil good match two good good good good good good good good good good good good good good good good good good good									
walk										quest good child tale good good good good good good good good good good good good good good good good good good good									
word										any one good									
body										know given visit pride heart point girls ought short spoke marry									
idea										power right asked write doubt night years table sense sorry usual under									
wife										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
want										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
best										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
else										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
read										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
back										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
help										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
dare										evil good match two good good good good good good good good good good good good good good good good good good good									
rest										quest good child tale good good good good good good good good good good good good good good good good good good good									
mary										any one good									
feel										know given visit pride heart point girls ought short spoke marry									
hour										power right asked write doubt night years table sense sorry usual under									
girl										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
poor										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
near										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
stay										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
sort										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
mean										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
glad										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
door										evil good match two good good good good good good good good good good good good good good good good good good good									
care										quest good child tale good good good good good good good good good good good good good good good good good good good									
four										any one good									
head										know given visit pride heart point girls ought short spoke marry									
ball										power right asked write doubt night years table sense sorry usual under									
also										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
deal										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
five										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
case										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
name										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
fine										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
talk										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
news										evil good match two good good good good good good good good good good good good good good good good good good good									
days										quest good child tale good good good good good good good good good good good good good good good good good good good									
real										any one good									
meet										know given visit pride heart point girls ought short spoke marry									
thus										power right asked write doubt night years table sense sorry usual under									
pain										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
week										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
lost										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
year										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
call										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
face										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
hand										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
turn										evil good match two good good good good good good good good good good good good good good good good good good good									
pray										quest good child tale good good good good good good good good good good good good good good good good good good good									
true										any one good									
keep										know given visit pride heart point girls ought short spoke marry									
bear										power right asked write doubt night years table sense sorry usual under									
send										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
easy										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
self										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
sake										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
fear										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
used										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
fair										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
park										evil good match two good good good good good good good good good good good good good good good good good good good									
live										quest good child tale good good good good good good good good good good good good good good good good good good good									
work										any one good									
vain										know given visit pride heart point girls ought short spoke marry									
sent										power right asked write doubt night years table sense sorry usual under									
drew										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
paid										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
none										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
late										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
join										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
need										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
full										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
high										evil good match two good good good good good good good good good good good good good good good good good good good									
open										quest good child tale good good good good good good good good good good good good good good good good good good good									
play										any one good									
cold										know given visit pride heart point girls ought short spoke marry									
book										power right asked write doubt night years table sense sorry usual under									
kept										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
view										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
wait										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
ease										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
loss										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
tone										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
seem										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
dine										evil good match two good good good good good good good good good good good good good good good good good good good									
pass										quest good child tale good good good good good good good good good good good good good good good good good good good									
step										any one good									
fond										know given visit pride heart point girls ought short spoke marry									
mine										power right asked write doubt night years table sense sorry usual under									
says										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
duty										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
shook										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
broke										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
down										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
with										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
luck										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
first										evil good match two good good good good good good good good good good good good good good good good good good good									
time										quest good child tale good good good good good good good good good good good good good good good good good good good									
fall										any one good									
low										know given visit pride heart point girls ought short spoke marry									
wind										power right asked write doubt night years table sense sorry usual under									
lose										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
hill										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
lane										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
quit										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
past										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
card										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
worry										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
fall										evil good match two good good good good good good good good good good good good good good good good good good good									
bore										quest good child tale good good good good good good good good good good good good good good good good good good good									
well										any one good									
till										know given visit pride heart point girls ought short spoke marry									
died										power right asked write doubt night years table sense sorry usual under									
at										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
last										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
year										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
ago										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
was										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
not										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
but										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
for										evil good match two good good good good good good good good good good good good good good good good good good good									
the										quest good child tale good good good good good good good good good good good good good good good good good good good									
first										any one good									
time										know given visit pride heart point girls ought short spoke marry									
ago										power right asked write doubt night years table sense sorry usual under									
was										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
not										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
but										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
for										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
the										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
first										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
time										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
ago										evil good match two good good good good good good good good good good good good good good good good good good good									
was										quest good child tale good good good good good good good good good good good good good good good good good good good									
not										any one good									
but										know given visit pride heart point girls ought short spoke marry									
for										power right asked write doubt night years table sense sorry usual under									
the										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
first										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
time										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
ago										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
was										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
not										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
but										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
for										evil good match two good good good good good good good good good good good good good good good good good good good									
the										quest good child tale good good good good good good good good good good good good good good good good good good good									
first										any one good									
time										know given visit pride heart point girls ought short spoke marry									
ago										power right asked write doubt night years table sense sorry usual under									
was										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
not										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
but										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
for										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
the										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
first										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
time										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
ago										evil good match two good good good good good good good good good good good good good good good good good good good									
was										quest good child tale good good good good good good good good good good good good good good good good good good good									
not										any one good									
but										know given visit pride heart point girls ought short spoke marry									
for										power right asked write doubt night years table sense sorry usual under									
the										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
first										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
time										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
ago										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
was										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
not										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
but										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
for										evil good match two good good good good good good good good good good good good good good good good good good good									
the										quest good child tale good good good good good good good good good good good good good good good good good good good									
first										any one good									
time										know given visit pride heart point girls ought short spoke marry									
ago										power right asked write doubt night years table sense sorry usual under									
was										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
not										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
but										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
for										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
the										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
first										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
time										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
ago										evil good match two good good good good good good good good good good good good good good good good good good good									
was										quest good child tale good good good good good good good good good good good good good good good good good good good									
not										any one good									
but										know given visit pride heart point girls ought short spoke marry									
for										power right asked write doubt night years table sense sorry usual under									
the										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
first										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
time										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
ago										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
was										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
not										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
but										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
for										evil good match two good good good good good good good good good good good good good good good good good good good									
the										quest good child tale good good good good good good good good good good good good good good good good good good good									
first										any one good									
time										know given visit pride heart point girls ought short spoke marry									
ago										power right asked write doubt night years table sense sorry usual under									
was										hurst dance taken allow equal words alone smile offer meant early truth hours liked									
not										doing money reply among match hopes eliza spite large small fixed terms proud event seems spent									
but										voice niece above wrote madam hoped maria times ready thank cause state women fancy weeks civil pause									
for										bring laugh avoid sight round light angry blame miles comes chief scene tried chose yours worth trust wrong books									
the										taste share child alarm lived aware stood chose twice longer looks were up send dured begin dead eager shrewdly false music									
first										enter judge deny order shook went boast noble hurry drove folly plain wrong view called owed each number near necessity table									
time										com stand tears name clear likes know laugh rebeccah babe force give lines back youth story how prove tested even good countable									
ago										evil good match two good good good good good good good good good good good good good good good good good good good									
was																			

P.3.2.1 Dissolving the font outline

A text is made up of characters. A character, in turn, is shaped by its outlines. In the following chapters, this outline dissipates into a multitude of points and will establish the basis for generative font manipulation. Individual points are replaced by other elements, thereby disguising the original font.

→ P_3_2_1_01

The starting point is a text and a font file. The opentype.js Library, designed by Frederik De Bleser, generates a multitude of points onto the font outline. This information can then be used to give the characters new visual identities.



→ P_3_2_1_02 SVGs are loaded and placed on the character outline in this version of the program. Rotation angles and scaling can be controlled with the mouse.

```

1 function setup() {
  ...
  opentype.load('data/FreeSans.otf', function(err, f) {
    font = f;
    loop();
  });
}

```

```

function draw() {
  ...
  if (textTyped.length > 0) {
2   var fontPath = font.getPath(textTyped, 0, 0, 200);
3   var path = new g.Path(fontPath.commands);
4   path = g.resampleByLength(path, 11);

  stroke(181, 157, 0);
  strokeWeight(1.0);
  var l = 5;
5   for (var i = 0; i < path.commands.length; i++) {
    var pnt = path.commands[i];
    line(pnt.x - l, pnt.y - l, pnt.x + l, pnt.y + l);
  }

  fill(0);
  noStroke();
  var diameter = 7;
  for (var i = 0; i < path.commands.length; i++) {
    var pnt = path.commands[i];
6    if (i % 2 == 0) {
      ellipse(pnt.x, pnt.y, diameter, diameter);
    }
  }
  }
  ...
}

```

Keys: Keyboard: Text input
 DEL: Delete letters
 CTRL: Save image

- 1 The opentype.js library loads the font file and stores it in the `font` variable.
- 2 There are three steps to extract the points: first, the `getPath()` function of the opentype.js library converts the characters to path.
- 3 This is then converted to a `g.Path` object (g.js is another library).
- 4 The command `resampleByLength()` divides this path into sections of equal length (here eleven pixels).
- 5 The dots are processed. Short, angled lines are first drawn on their positions.

- 6 Next, black circles are processed. In this run, only every second position is used.

→ P_3_2_1_01 Graphic elements are placed on character outlines.

When you

Only look at

the

Once you have

what will you say?

→ P_3_2_1_02 The appearance of the generated character forms can be regulated by the scaling and rotation of the elements on the character outlines.



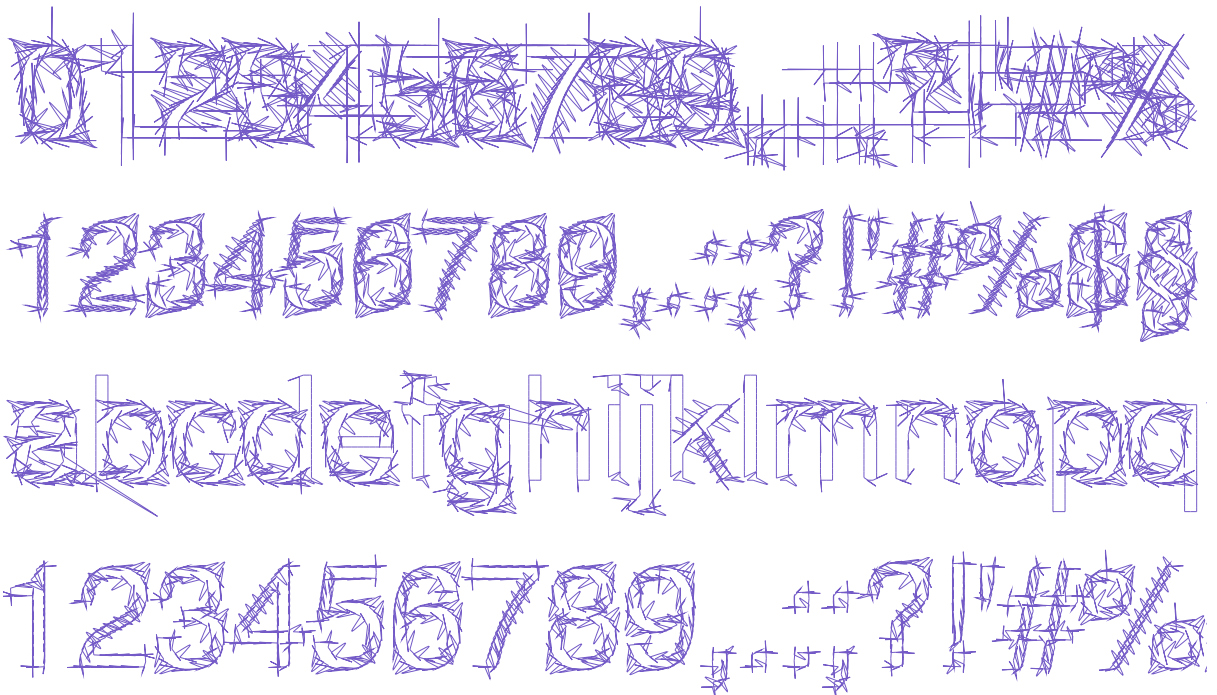
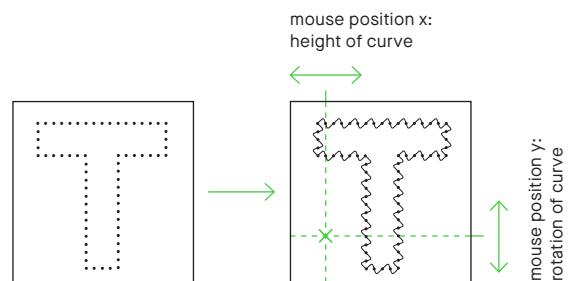
Jonathan Harris

P.3.2.2 Varying the font outline

If the font outline is not made up of straight lines or curves but controllable elements instead, then we increasingly free ourselves from the underlying font. All existing base points are joined together with specially formed Bézier curves. This is just one of the myriad ways of quickly creating dozens of new fonts.

→ P_3_2_2_01

The dots on the text outline are connected with Bézier curves. The shape of the curves can be controlled interactively with the mouse.



→ P_3_2_2_01 Different settings for the height and rotation of the Bézier curves.


```

function draw() {
  ...
  if (textTyped.length() > 0) {
    ...
    var addToAngle = map(mouseX, 0, width, -PI, +PI);
    var curveHeight = map(mouseY, 0, height, 0.1, 2);

    for (var i = 0; i < path.commands.length-1; i++) {
      var pnt0 = path.commands[i];
      var pnt1 = path.commands[i+1];
      var d = dist(pnt0.x, pnt0.y, pnt1.x, pnt1.y);

      if (d > 20) continue;

      var stepper = map(i%2, 0, 1, -1, 1);
      var angle = atan2(pnt1.y-pnt0.y, pnt1.x-pnt0.x);
      angle = angle + addToAngle;

      var cx = pnt0.x+cos(angle*stepper)*d*4*curveHeight;
      var cy = pnt0.y+sin(angle*stepper)*d*3*curveHeight;

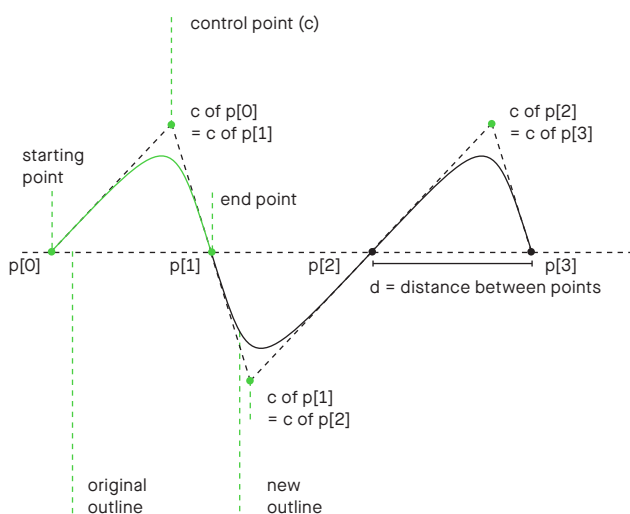
      bezier(pnt0.x,pnt0.y, cx,cy, cx,cy, pnt1.x,pnt1.y);
    }
  }
}

```

Mouse: Position x: Rotation of curve
Position y: Height of curve

Keys: Keyboard: Input text
DEL: Delete letters
ALT: Change filling mode
CTRL: Save image

- 1 The variables `addToAngle` and `curveHeight` result from the x- and y-coordinates of the mouse position and control the rotation and height of the Bézier curves.
- 2 The dots are processed from the first to the next-to-last. The distance from the current dot to the next one is calculated each time.
- 3 When the distance is greater than fifty, this loop is aborted and no line is drawn. Since the opentype.js library provides the dots for the entire text as a chain of dots, it ensures that the individual letters are not connected.
- 4 For the variable `stepper`, the values `-1` and `1` are alternately produced. These are used to calculate the control points for the Bézier curve `cx, cy` in order to move the curve up and down.
- 5 Four points have to be defined when drawing the Bézier curve: the beginning point, the end point, and two control points. The control point just calculated is used twice here.



A new outline is produced and moves up and down around the original outline; it consists of individual Bézier curves that are defined by the beginning and end points and the two control points, which in this case have the same value: $c \text{ of } p[0] = c \text{ of } p[1]$.

When you'

your own

even the

the limit

→ **P_3_2_2_01** The mouse position determines the form of the Bézier curves.
It is possible to switch to filled curves by using the ALT key.

no creating

still men

sky ain't

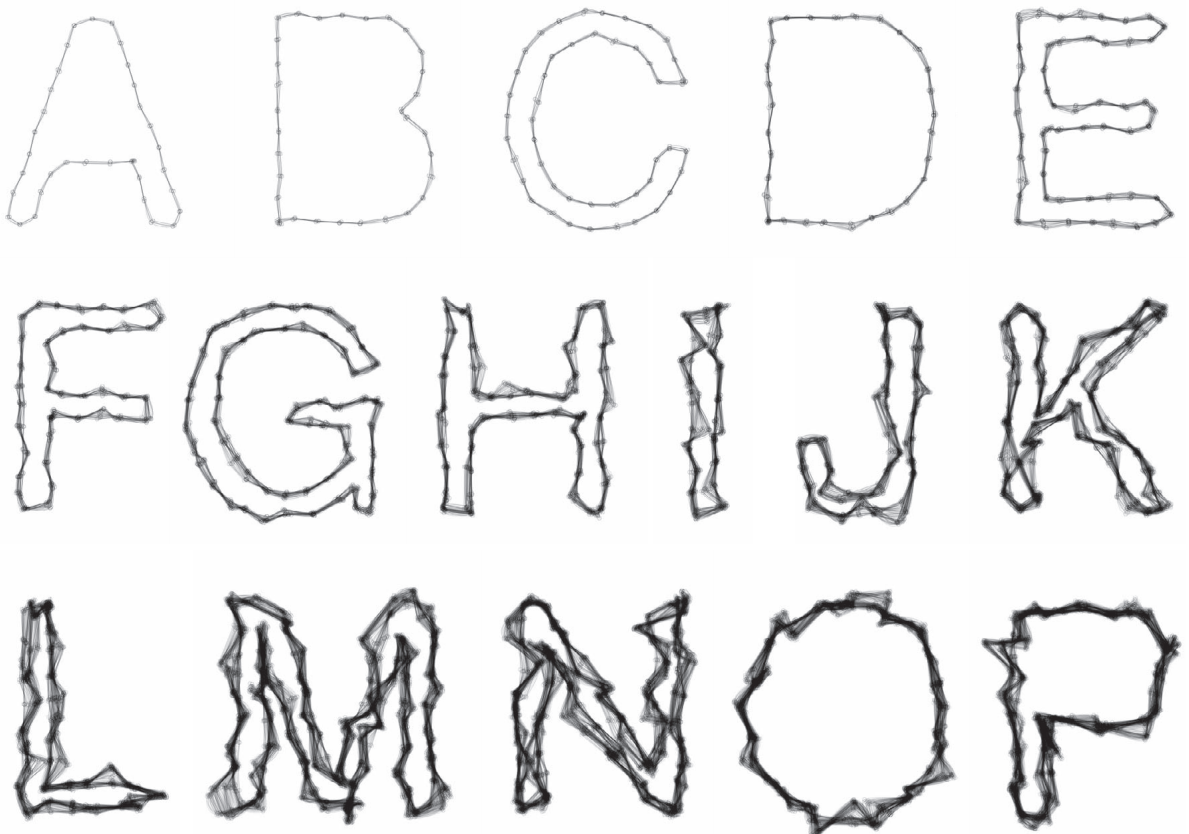
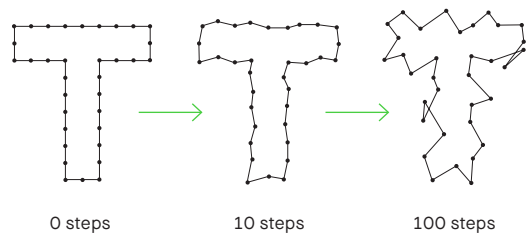
Charles Mingus

P.3.2.3 Font outline from agents

How long is a letter recognizable as such? In this example, the outlines of a letter serve as the source shape. Each individual nodal point moves like a dumb agent. Over time, the letter becomes illegible and is transformed into something new.

→ P_3_2_3_01

Points are again generated from a font outline. Each point becomes an independent dumb agent but remains connected to its neighbor.



→ P_3_2_3_01 The more time that passes without a key being pressed, the more a character becomes deformed.

```

function draw() {
  ...
1  translate(letterX, letterY);

  danceFactor = 1;
2  if (mouseIsPressed && mouseButton == LEFT)
    danceFactor = map(mouseX, 0, width, 0, 3);

  if (pnts.length > 0) {
    for (var i = 0; i < pnts.length; i++) {
3     pnts[i].x += random(-stepSize, stepSize)
      * danceFactor;
      pnts[i].y += random(-stepSize, stepSize)
      * danceFactor;
    }

    strokeWeight(0.1);
    stroke(0);
    beginShape();
    for (var i = 0; i < pnts.length; i++) {
4     vertex(pnts[i].x, pnts[i].y);
      ellipse(pnts[i].x, pnts[i].y, 7, 7);
    }
5    vertex(pnts[0].x, pnts[0].y);
    endShape();
  }

  pop();
}

```

Mouse: Left click + position x: Deformation speed

Keys: Keyboard: Input text
 SHIFT: Movement start/stop
 DEL: Clear canvas
 CTRL: Save image

- 1 The origin of the coordinate system is moved to the current writing position before a letter is written.
- 2 By keeping the mouse button pressed down, the variable `danceFactor` is set to a value, which increases proportionally to the value of the mouse's x-coordinate.
- 3 Random values are added to a point's position in every iteration. The value `danceFactor` increases the speed of the movement.
- 4 Lines connect the dots.
- 5 Finally, another line is drawn to the first point, closing the outline.

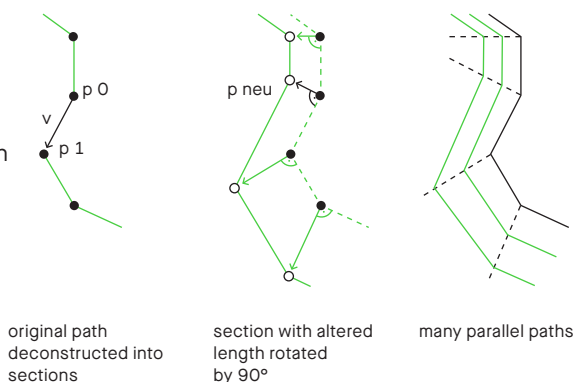


P.3.2.4 Parallel font outlines

Using the moiré effect of overlapping grid structures, you can create optical illusions that affect font outlines and change the impression of font volumes. Eventually forms emerge that detach themselves from the font and lead a life of their own.

→ P_3_2_4_01

The starting point is the font contour of a letter. For each of the many short sections that make up the font's outline, the same calculation procedure is used: the section is rotated 90° and set to the correct length. The result is a path that runs parallel to the original path. The grid structure arises when this is repeated several times with ever-increasing distances.



→ P_3_2_4_01 The lowercase letter "a" shown in three variations. The font outline here, however, has been increasingly simplified.


```

1 function createLetters() {
2   letters = [];
3   var chars = textTyped.split('');

   var x = 0;
   for (var i = 0; i < chars.length; i++) {
     if (i > 0) {
4       var charsBefore = textTyped.substr(0, i);
       x = font.textBounds(charsBefore, 0, 0, fontSize).w;
     }
5     var newLetter = new Letter(chars[i], x, 0);
6     letters.push(newLetter);
   }
}

```

```

function Letter(char, x, y) {
  this.char = char;
  this.x = x;
  this.y = y;

5  Letter.prototype.draw = function() {
6    var path = font.textToPoints(
      this.char, this.x, this.y, fontSize,
      {sampleFactor: pathSampleFactor});
    stroke(shapeColor);

7    for (var d = 0; d < ribbonWidth; d += density) {
      beginShape();

      for (var i = 0; i < path.length; i++) {
        var pos = path[i];
8        var nextPos = path[i + 1];

9        if (nextPos) {
          var p0 = createVector(pos.x, pos.y);
          var p1 = createVector(nextPos.x, nextPos.y);
10         var v = p5.Vector.sub(p1, p0);
11         v.normalize();
          v.rotate(HALF_PI);
          v.mult(d);
12         var pneu = p5.Vector.add(p0, v);
          curveVertex(pneu.x, pneu.y);
        }
      }

      endShape(CLOSE);
    }
  }
}

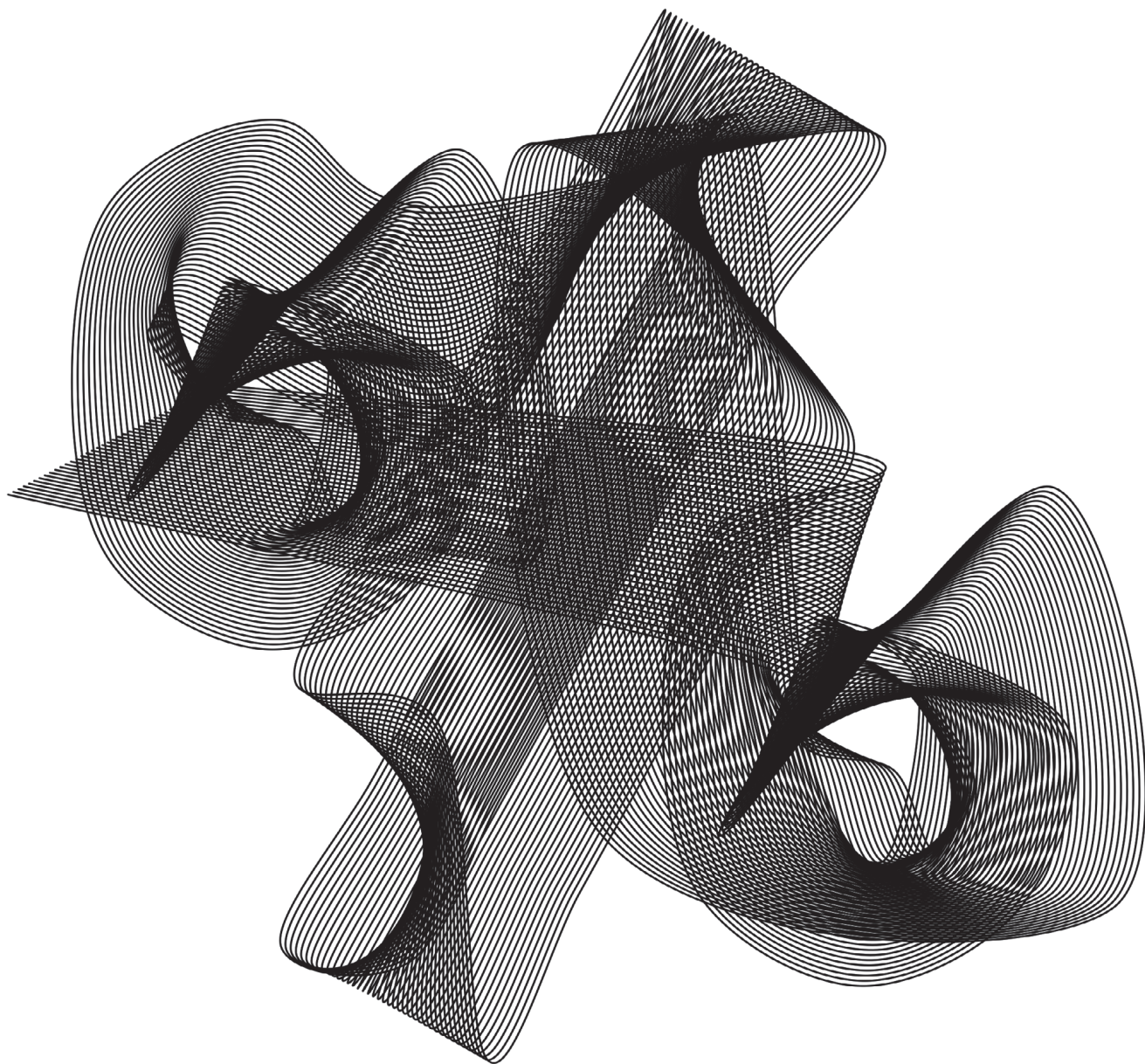
```

Mouse: Position x: Simplification of font outline
 Position y: Width of ribbon outline

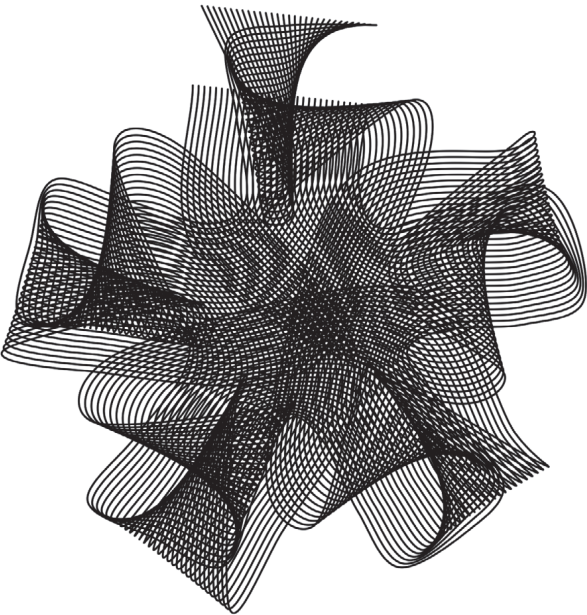
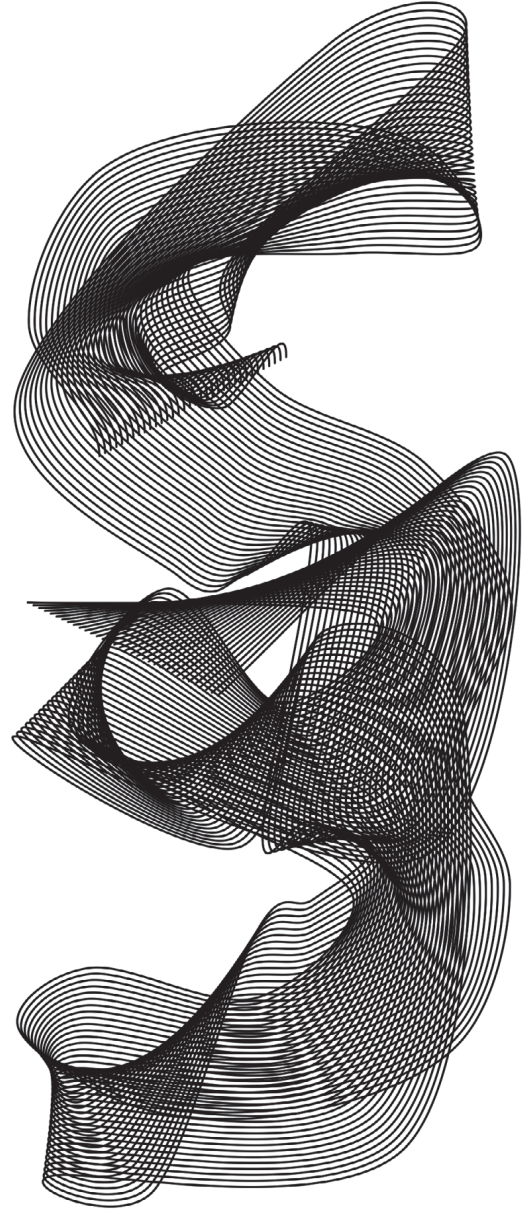
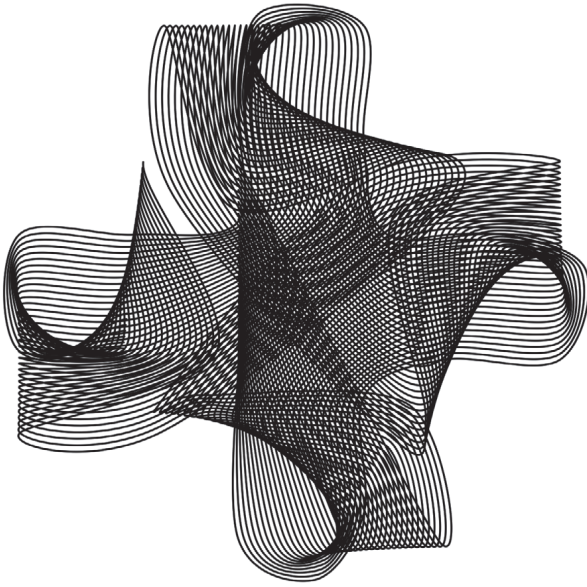
Keys: Arrow ←/→: Change line density
 Arrow ↓/↑: Change font size

- 1 When the program starts, or whenever the entered text changes, the `createLetters()` function is called.
- 2 There, the input text with `split()` generates an array of single letters, `chars`.

To determine the `x`-position of a letter, use `substr()` to remove the substring up to the current character and use the `textBounds()` function to calculate its width, `w`.
- 3 For each letter a new instance of the `Letter` class is created and added to the array `letters`.
- 5 The letter class has a `draw()` function called by the main program in each frame. There, the font outline is moved farther and farther inward.
- 6 The `textToPoints()` function turns the `char` character into an array of points.
- 7 This loop draws the individual paths. In each loop, the variable `d` contains the distance of the path to be drawn from the original path.
- 8 Two consecutive positions are taken from the array `path`.
- 9 If `nextPos` is not empty (i.e., the end of the path has not yet been reached), the two positions are converted to values of type `p5.Vector` with `createVector()`.
- 10 `sub()` calculates the difference between the two points and stores them in `v`.
- 11 The vector `v` is moved to length 1 with `normalize()`, rotated 90° with `rotate()`, and then multiplied by `d`.
- 12 The position on the offset path is determined by adding `v` to `p0`.



→ **P_3_2_4_01** Four characters: percent, plus, star, and paragraph. The outline was greatly simplified. This results in ornate figures.

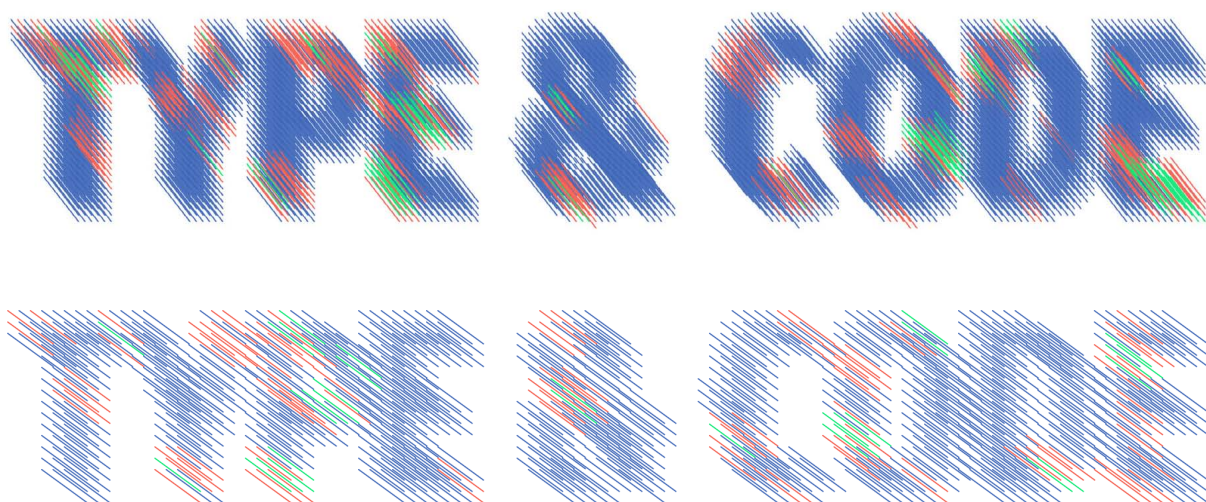
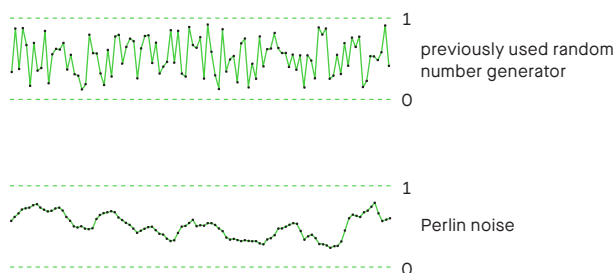


P.3.2.5 Kinetic font

Here the font outline may do as it wishes. Ignoring legibility, it transforms into patterns and leaves no formal gimmick untried. In constant motion, these metamorphoses remain alive and make us wonder: When does writing become a form of its own?

→ P_3_2_5_01

Normally, it is good if every newly generated random number is truly random. When creating animations, however, this usually leads to the image flickering. The use of Perlin noise prevents this. This method of calculating random numbers generates values where the difference from one to the next is never very large.



→ P_3_2_5_01 Rotating lines—sometimes densely arranged, sometimes with a greater distance between them.

```

1 function setupText() {
  textImg = createGraphics(width, height);
  textImg.pixelDensity(1);
  textImg.background(255);
  textImg.textFont(font);
  textImg.textSize(fontSize)
2 textImg.text(textTyped, 100, fontSize + 50);
3 textImg.loadPixels();
}

```

```

function draw() {
  background(255);

  nOff++;

  for (var x = 0; x < textImg.width; x+=pointDensity) {
    for (var y = 0; y < textImg.height; y+=pointDensity)
4 {
5     var index = (x + y * textImg.width) * 4;
    var r = textImg.pixels[index];

    if (r < 128) {

      if(drawMode == 1){
        strokeWeight(1);

        var noiseFac = map(mouseX, 0,width, 0,1);
        var lengthFac = map(mouseY, 0,height, 0.01,1);

6       var num = noise((x+nOff) * noiseFac,
7                 y * noiseFac);
        if (num < 0.6) {
          stroke(colors[0]);
        } else if (num < 0.7) {
          stroke(colors[1]);
        } else {
          stroke(colors[2]);
        }

        push();
        translate(x, y);
        rotate(radians(frameCount));
8       line(0, 0, fontSize * lengthFactor, 0);
        pop();
      }
      ...
    }
  }
}

```

Mouse: Position x/y: Different parameters
(depending on drawing mode)

Keys: Keyboard: Text input
Arrow ←/→: Change drawing mode
Arrow ↓/↑: Change point density
DEL: Clear canvas
CTRL: Save image

1 Each time the text is changed, the `setupText()` function is called. This creates a so-called off-screen graphic using `createGraphics()`. This is an image that is not visible but exists only in memory.

2 The entered text, `textTyped`, is written in this image in the previously set font and size.

3 Call `loadPixels()` to be able to read the individual pixel values later.

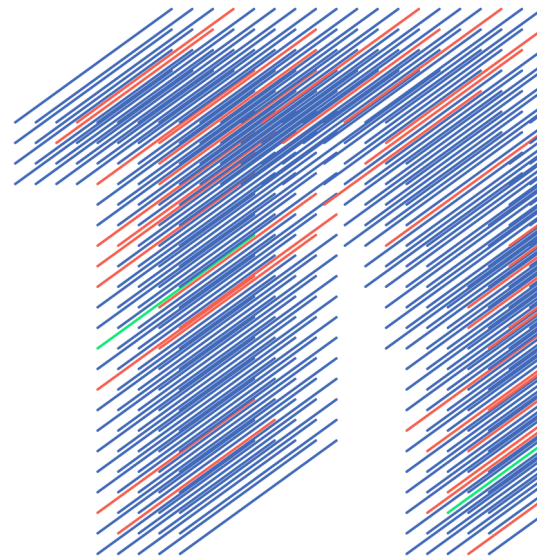
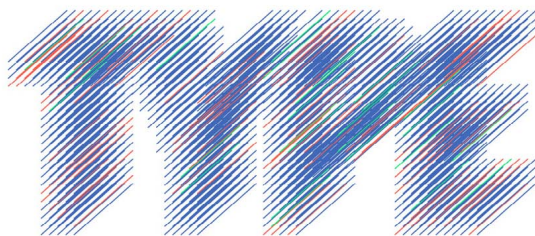
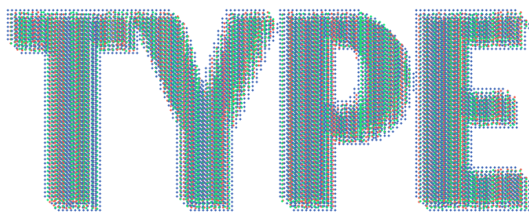
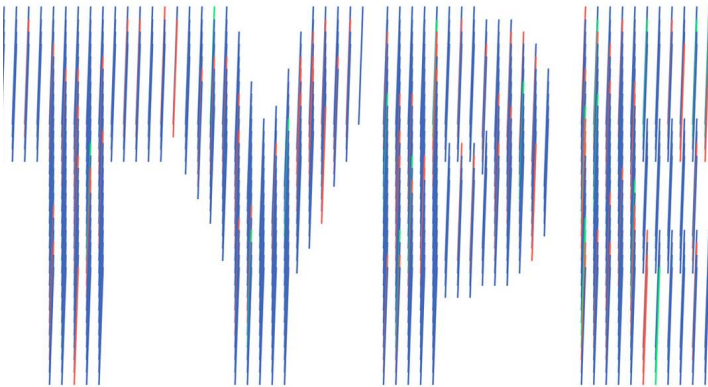
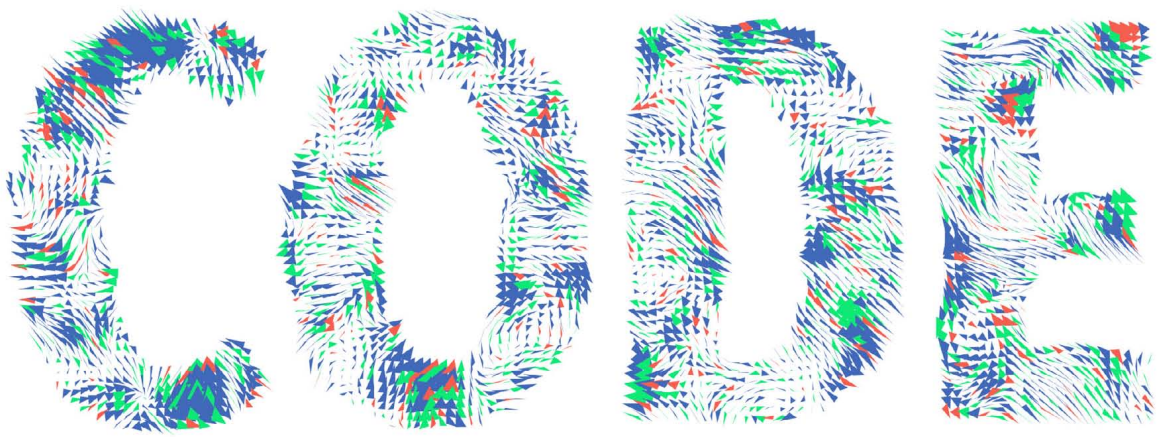
4 The color values of the image are stored as a long string of values. Therefore, to get the color value of a pixel, an index must be calculated from `x` and `y`. The factor 4 is necessary because one pixel consists of four separately stored values (one each for red, green, blue, and transparency).

5 The image with the text consists only of black, white, and a few gray pixels. Therefore, it is sufficient to check only if the red value `r` is below a certain limit, in which case it is a dark pixel.

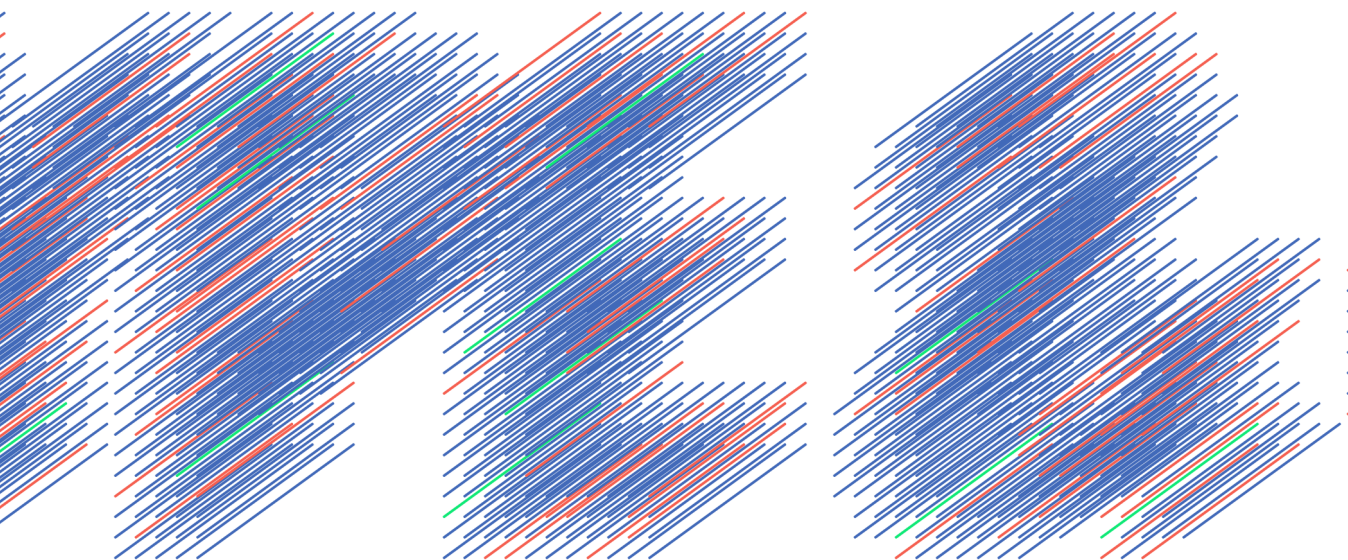
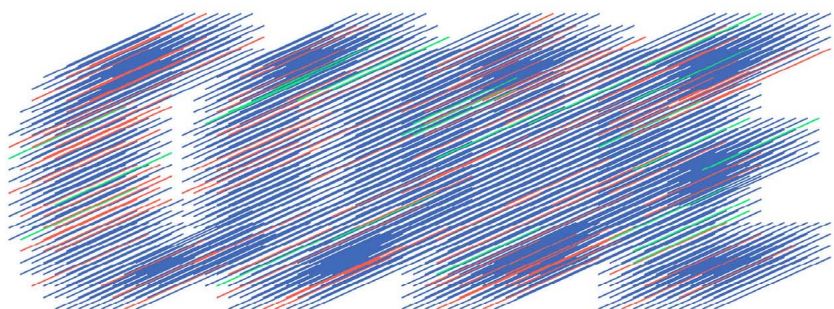
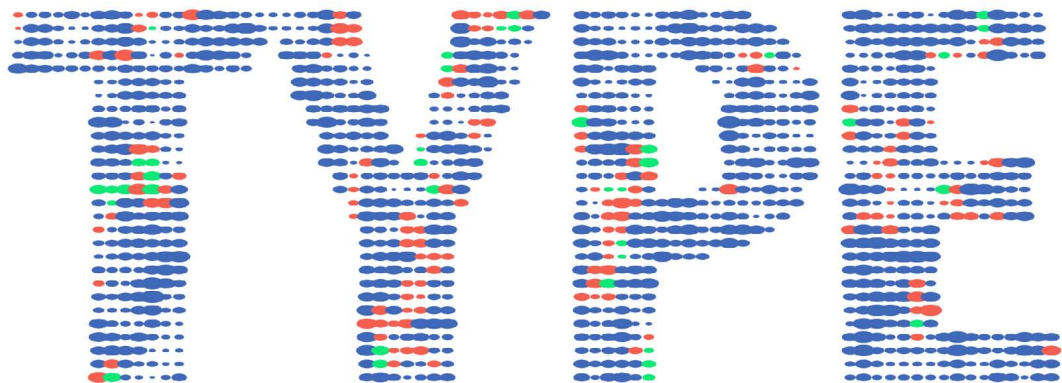
6 A random value is required to color the lines. To avoid flickering, `noise()` is preferable to the `random()` function. This produces random numbers, similar to a mountainous landscape. The function `noise()` is called with two parameters here, the first dependent on `x`, the second on `y`. The variable `nOff` is incremented continuously, thus ensuring an animation of the random numbers.

7 Depending on `num`, one of the three predefined colors will be selected.

8 A horizontal line is drawn in a previously shifted and rotated coordinate system.



→ P_3_2_5_01 to → P_3_2_5_03 Different parameter settings and results from all three versions of the program. The letter shapes are generated in different ways: from the pixels (version 01), entirely programmed (version 02), or from the font contours (version 03).



P.4

Image

In the last chapter, we saw how text can be dissolved and how the resulting elements—words, letters, and even dots on contours—can be used for experimentation. Similarly, images can be manipulated: details can be copied, collages can be produced, and pixels—the digital image’s smallest units of information—can become the basis of a new visual world.

P.4 Image

188

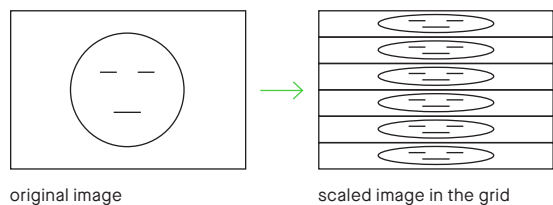
P.4.0	Hello, image	190
P.4.1	Image cutouts	192
P.4.1.1	Image cutouts in a grid	192
P.4.1.2	Feedback of image cutouts	196
P.4.2	Image collection	198
P.4.2.1	Collage from image collection	198
P.4.2.2	Time-based image collection	202
P.4.3	Pixel values	204
P.4.3.1	Graphics from pixel values	204
P.4.3.2	Type from pixel values	210
P.4.3.3	Real-time pixel values	214
P.4.3.4	Emojis from pixel values	220

P.4.0 Hello, image

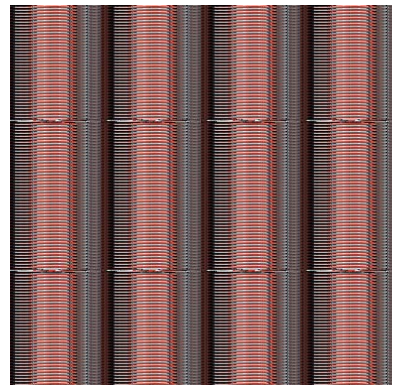
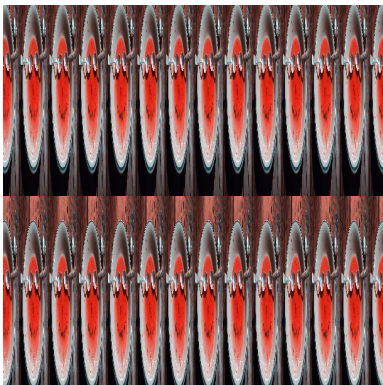
A digital image is a mosaic of small color tiles. Dynamic access to these tiny elements allows for the generation of new compositions. It is possible to create your own collection of image tools with the following programs.

→ P_4_0_01

An image is loaded and displayed in a grid defined by the mouse. Each tile in the grid is filled with a scaled copy of the source image.



original image



→ P_4_0_01 Abstract images are created through the repeated copying and extreme scaling of the source image.

```

1  var img;

    function preload() {
      img = loadImage('data/image.jpg');
    }

```

```

2  function draw() {
    var tileCountX = mouseX / 3 + 1;
    var tileCountY = mouseY / 3 + 1;
    var stepX = width / tileCountX;
    var stepY = height / tileCountY;
    for (var gridY = 0; gridY < height; gridY += stepY) {
      for (var gridX = 0; gridX < width; gridX += stepX) {
3    image(img, gridX, gridY, stepX, stepY);
      }
    }
  }

```

Mouse: Position x: Number of horizontal tiles

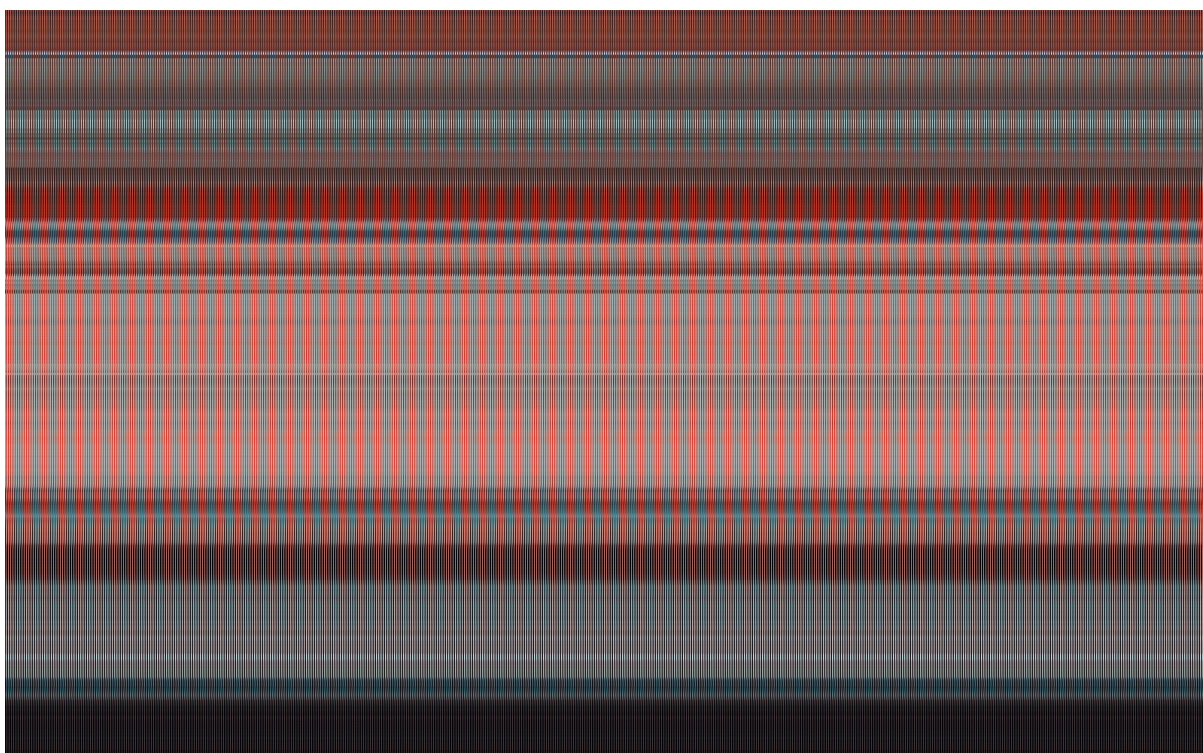
Position y: Number of vertical tiles

Keys: S: Save image

1 The image is loaded in the `preload()` function. This ensures that the loading process is completed before calling the `setup()` and `draw()` functions.

2 The mouse position determines `tileCountX` and `tileCountY` and, thereby, their width `stepX` and height `stepY`.

3 The image is drawn using the function `image()`. The upper-left corner of the image is located in the grid (`gridX`, `gridY`); width and height are determined by tile width `stepX` and tile height `stepY`.

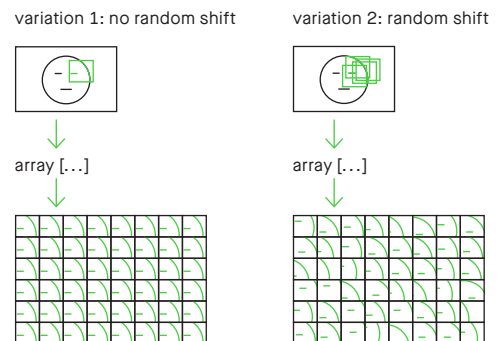


P.4.1.1 Image cutouts in a grid

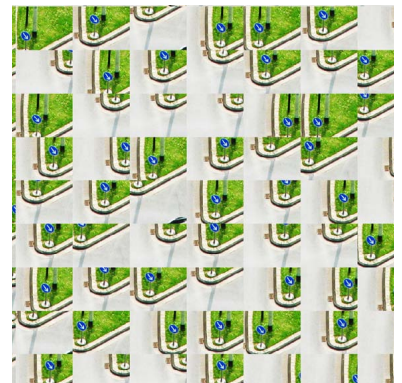
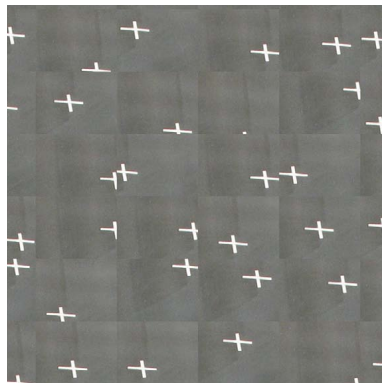
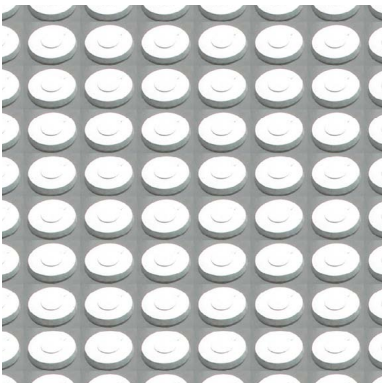
The principle illustrated below is almost the same as the one in the previous example, and yet a whole new world of images emerges. An image's details and fine structures become pattern generators when only a portion of it is selected and configured into tiles. The results are even more interesting if these sections are randomly selected.

→ P_4_1_1_01

Using the mouse, a section of the image is selected in the display window. After releasing the mouse button, several copies of this section are stored in an array and organized in a grid. The program offers two variations. In variation one, all copies are made from the exact same section. In variation two, the section is shifted slightly at random each time.



original image



→ P_4_1_1_01 By repeatedly copying and moving image sections, abstract images are created.


```

1 function cropTiles() {
    tileWidth = width / tileCountY;
    tileHeight = height / tileCountX;
2 imgTiles = [];

    for (var gridY = 0; gridY < tileCountY; gridY++) {
        for (var gridX = 0; gridX < tileCountX; gridX++) {
3             if (randomMode) {
                cropX = int(random(mouseX - tileWidth / 2,
                                   mouseX + tileWidth / 2));
                cropY = int(random(mouseY - tileHeight / 2,
                                   mouseY + tileHeight / 2));
            }
4             cropX = constrain(cropX, 0, width - tileWidth);
            cropY = constrain(cropY, 0, height - tileHeight);
5             imgTiles.push(img.get(cropX, cropY,
                                   tileWidth, tileHeight));
        }
    }
}

```

Mouse: Position x/y: Detail positioning

Left click: Copy detail

Keys: 1-3: Change detail size

R: Random on/off

S: Save image

1 The core of the program is the `cropTiles()` function. Here the image is fragmented and the copies of the sections are stored in an array.

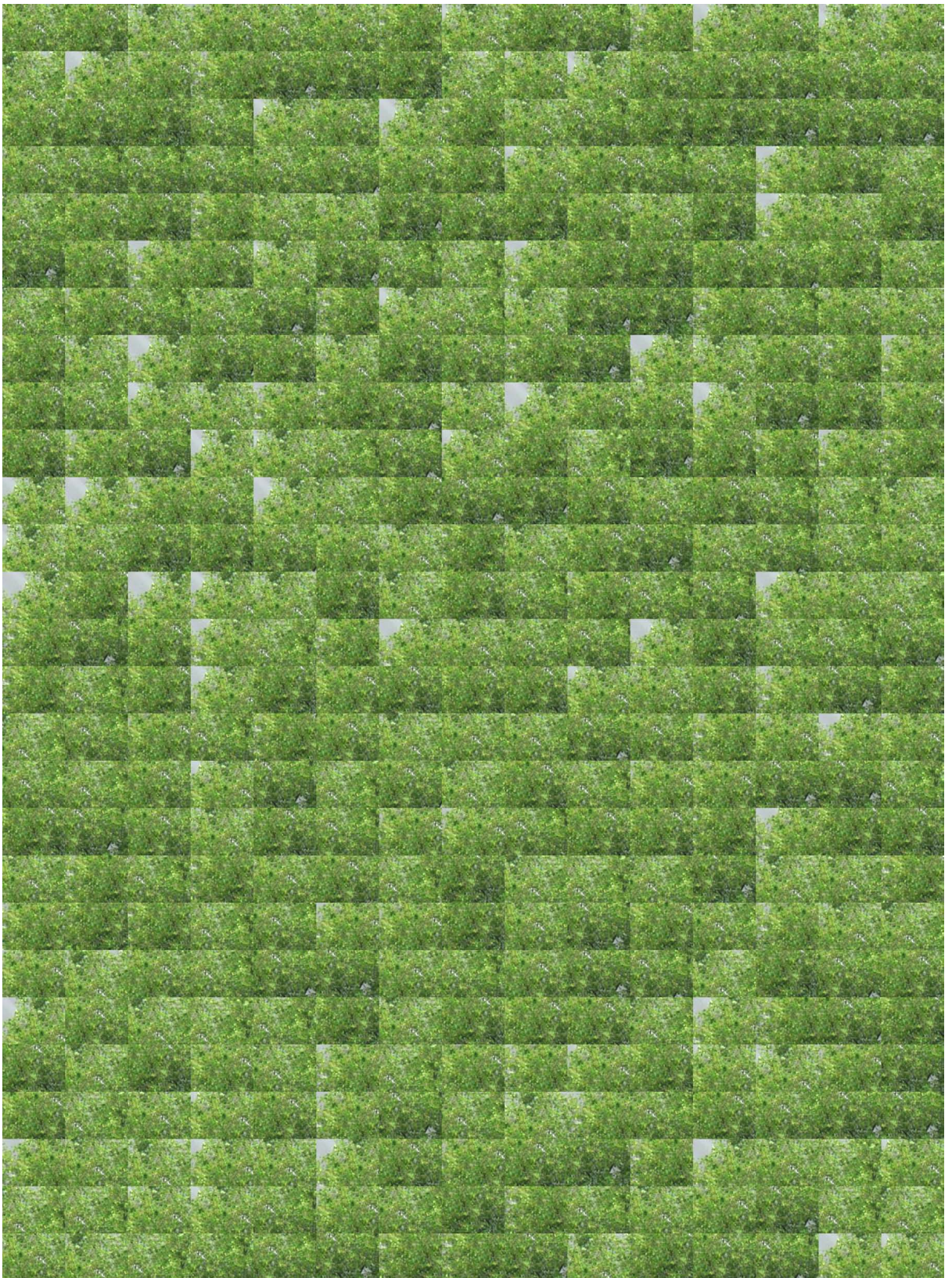
2 The image framing array `imgTiles` is cleared.

3 When version two comes into play (`randomMode` is true), all the values for `cropX` and `cropY` are randomly selected from a value range around the mouse position.

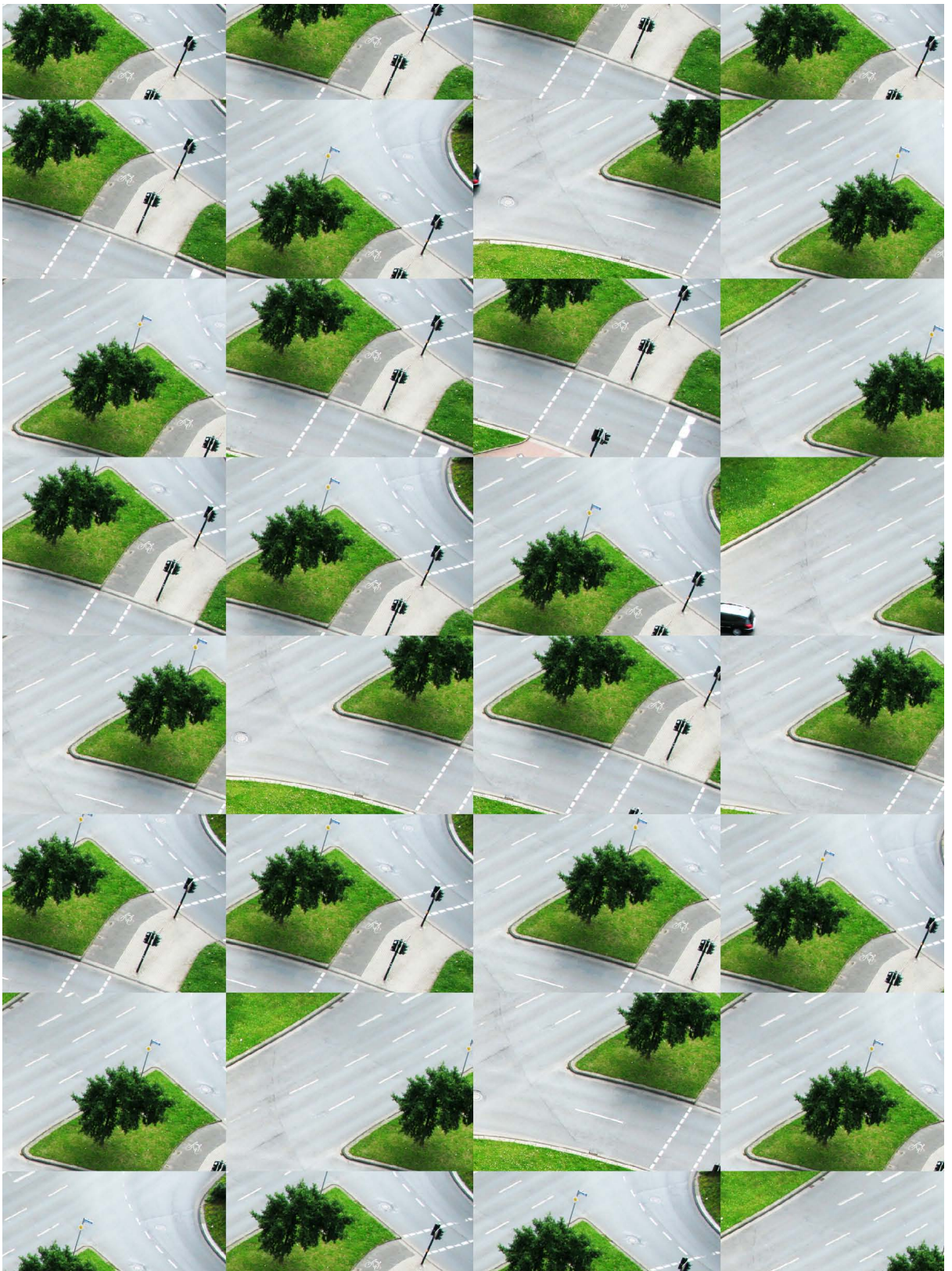
4 The `constrain()` function ensures that the cutout section does not extend beyond the image boundaries.

5 Finally, the section is copied from the image `img` using `get()` and stored in the array.





→ P_4_1_1_01 The multiplication of small image sections creates rhythmic structures that are only recognizable as image sections at a second glance.



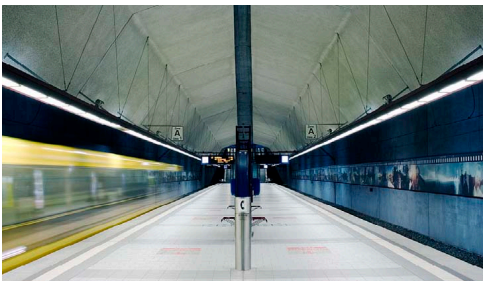
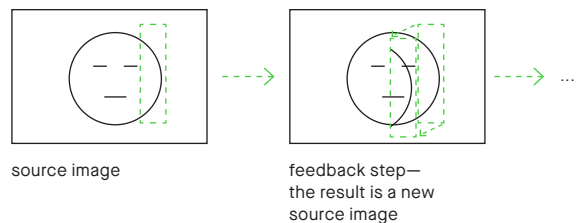
→ **P_4_1_1_01** Using keys 1 to 3, selections can be made among different portions of the image sections. The motifs are still recognizable in these large, detail-rich excerpts but now have an unsettling perspective.

P.4.1.2 Feedback of image cutouts

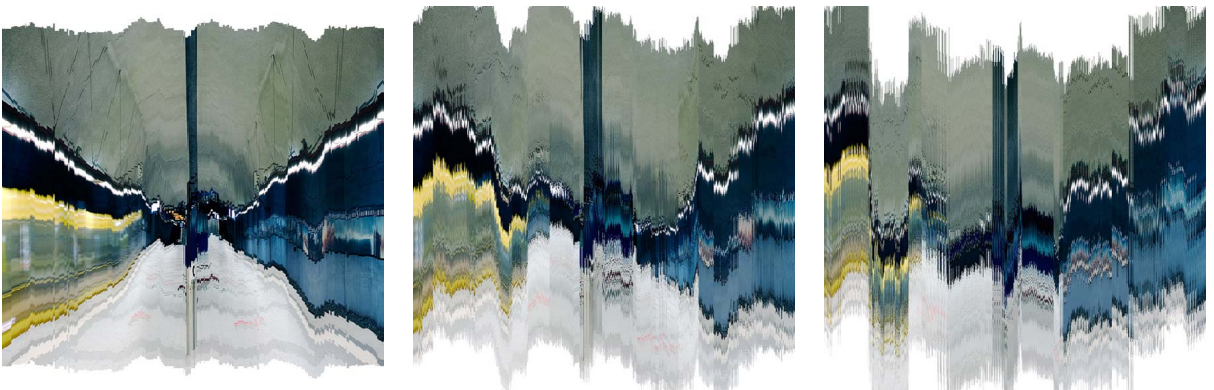
A familiar example of feedback: a video camera is directed at a television screen that displays the image taken by the camera. After a short time, the television screen depicts an ever-recurring and distorted image. When this phenomenon is simulated, an image's level of complexity is increased. This repeated overlaying leads to a fragmentary composition.

→ P_4_1_2_01

First, the image is loaded and shown in the display. A section of the image is copied to a new randomly selected position with each iteration step. The resulting image now serves as the basis for the next step—the principle of each and every feedback.



→ Fotografie: Stefan Eigner original image: subway tunnel



→ P_4_1_2_01 Right after the program starts, the motif is easily recognizable. It then dissolves more and more through the overlapping of copied image strips.

```

1 function setup() {
  createCanvas(1024, 780);
  image(img, 0, 100);
}

2 function draw() {
  var x1 = floor(random(width));
  var y1 = 0;

  var x2 = round(x1 + random(-7, 7));
  var y2 = round(random(-5, 5));

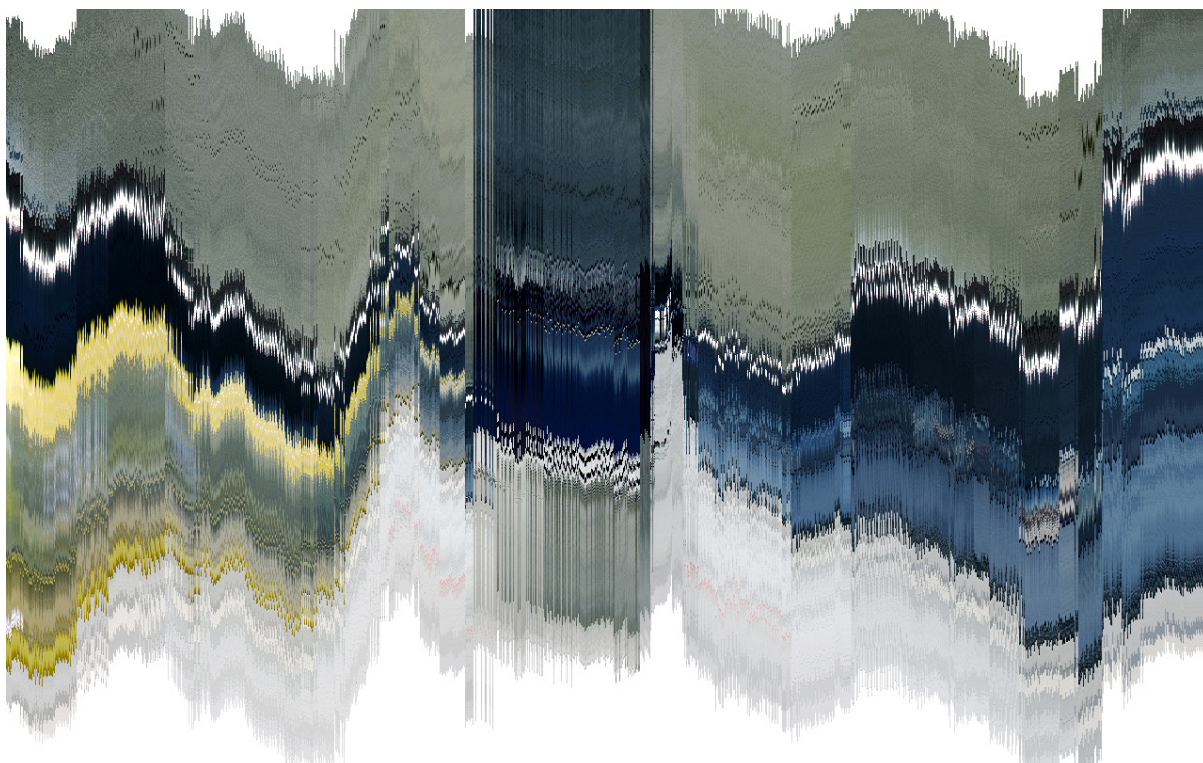
  var w = floor(random(10, 40));
  var h = height;

3  set(x2, y2, get(x1, y1, w, h));
}

```

Keys: DEL: Delete display
S: Save image

- 1 When the program is started, the loaded image is offset one hundred pixels downward using the `image()` command and positioned on the drawing canvas.
- 2 The x-position of the detail to be copied (`x1`), the target position (`x2`, `y2`), and its width (`w`) are all determined randomly.
- 3 Using the function `get()`, some of the canvas content is copied and then pasted into the new position (`x2`, `y2`) using `set()`.

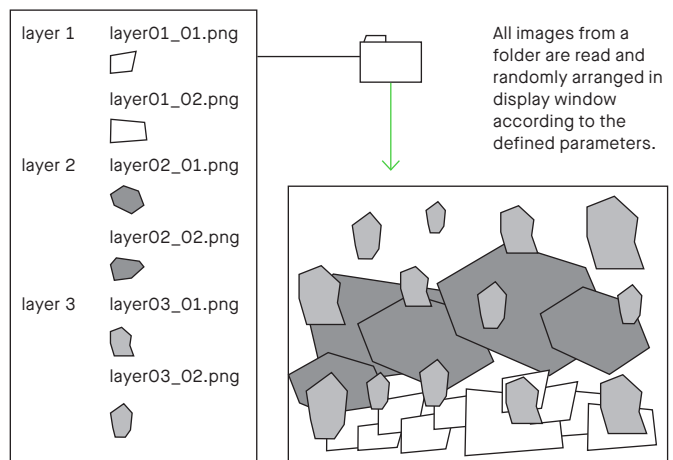


P.4.2.1 Collage from image collection

Your archive of photographs now becomes artistic material. This program assembles a collage from a folder of images. The cropping, cutting, and sorting of the source images are especially important, since only picture fragments are recombined in the collage.

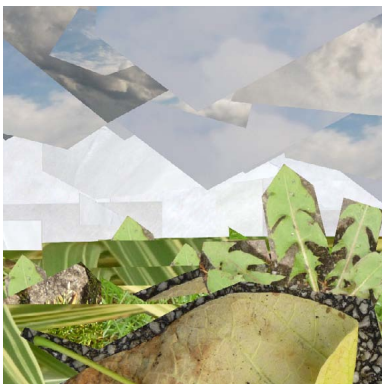
→ P_4_2_1_01

All the pictures in a folder are read dynamically and assigned to one of several layers. This allows the semantic groups to be treated differently. The individual layers also have room for experimentation with rotation, position, and size when constructing the collage. Note the layer order; the first level is drawn first and is thus in the background.



Images are assigned to layers according to the filenames (e.g., "layer01_01.png").

Here, only a few large elements are created from the images on layer 2, while many small ones are created from those on layer 3.



→ P_4_2_1_01 Illustration: **Andrea von Danwitz** The image consists of three levels: scraps of paper are on layer 1, cutouts of the sky on layer 2, and plants and street elements on layer 3.

A new composition is immediately created when the images on a level are switched or the parameters are changed.


```

1  var layer1Images = [];
    var layer2Images = [];
    var layer3Images = [];

    var layer1Items = [];
    var layer2Items = [];
    var layer3Items = [];

```

```

function setup() {
    ...
2  layer1Items = generateCollageItems(
    layer1Images, 100, width / 2, height / 2,
    width, height, 0.1, 0.5, 0, 0);
    layer2Items = generateCollageItems(
    layer2Images, 150, width / 2, height / 2,
    width, height, 0.1, 0.3, -HALF_PI, HALF_PI);
    layer3Items = generateCollageItems(
    layer3Images, 110, width / 2, height / 2,
    width, height, 0.1, 0.4, 0, 0);

3  drawCollageItems(layer1Items);
    drawCollageItems(layer2Items);
    drawCollageItems(layer3Items);
}

```

```

4  function CollageItem(image) {
    this.image = image;
    this.x = 0;
    this.y = 0;
    this.rotation = 0;
    this.scaling = 1;
}

```

Keys: 1-3: New random arrangement for one of the three levels
S: Save image

1 Several arrays are needed to load the images and arrange the layout of the collage pieces. In `layer1Images`, e.g., the loaded images are saved for the first layer in order to be used later to create the collage items.

2 The function `generateCollageItems()` fills the array layers (`layer1Items,...`) with collage items. The parameters determine which loaded images are to be used and how many items are to be created, and they specify value ranges for positions, scattering, scaling, and rotation. In this example, images in the array `layer1Items` are used. All instances are placed in the position (`width/2, height/2`) with a scattering of `width` and `height`. The scaling varies from `0.1` to `0.5` and no rotation is used.

3 Each time the `drawCollageItems()` function is run, one of the layers is drawn. The order of the layers' invocation determines the construction of the final composition. The images from `layer1Items` are located in the background, those from `layer3Items` in the foreground.

4 All features of a collage item are summarized in the class `CollageItem`.



→ P_4_2_1_02 → Illustration: **Andrea von Danwitz** A second version of the program allows the image details to be arranged radially around a particular center. The angle at which the images gather and their distance from the center can be specified for each layer.

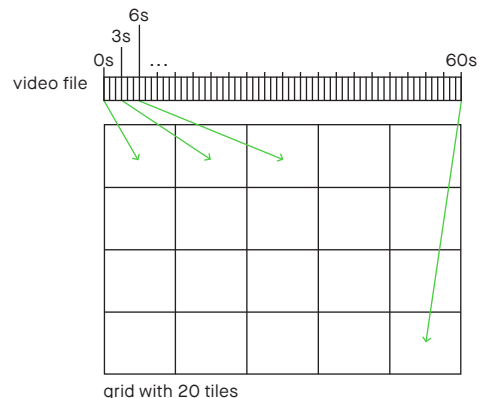


P.4.2.2 Time-based image collection

In this example, the inner structures of moving images are visualized. After extracting individual images from a video file, this program arranges the images in defined and regular time intervals in a grid. This grid depicts a compacted version of the entire video file and represents the rhythm of its cuts and frames.

→ P_4_2_2_01

To fill the grid, individual still images are extracted at regular intervals from the entire length of a video. Accordingly, a sixty-second video and a grid with twenty tiles results in three-second intervals.



```
1 function draw() {
  if(movie.elt.readyState == 4) {
    var posX = tileWidth * gridX;
    var posY = tileHeight * gridY;

    image(movie, posX, posY, tileWidth, tileHeight);

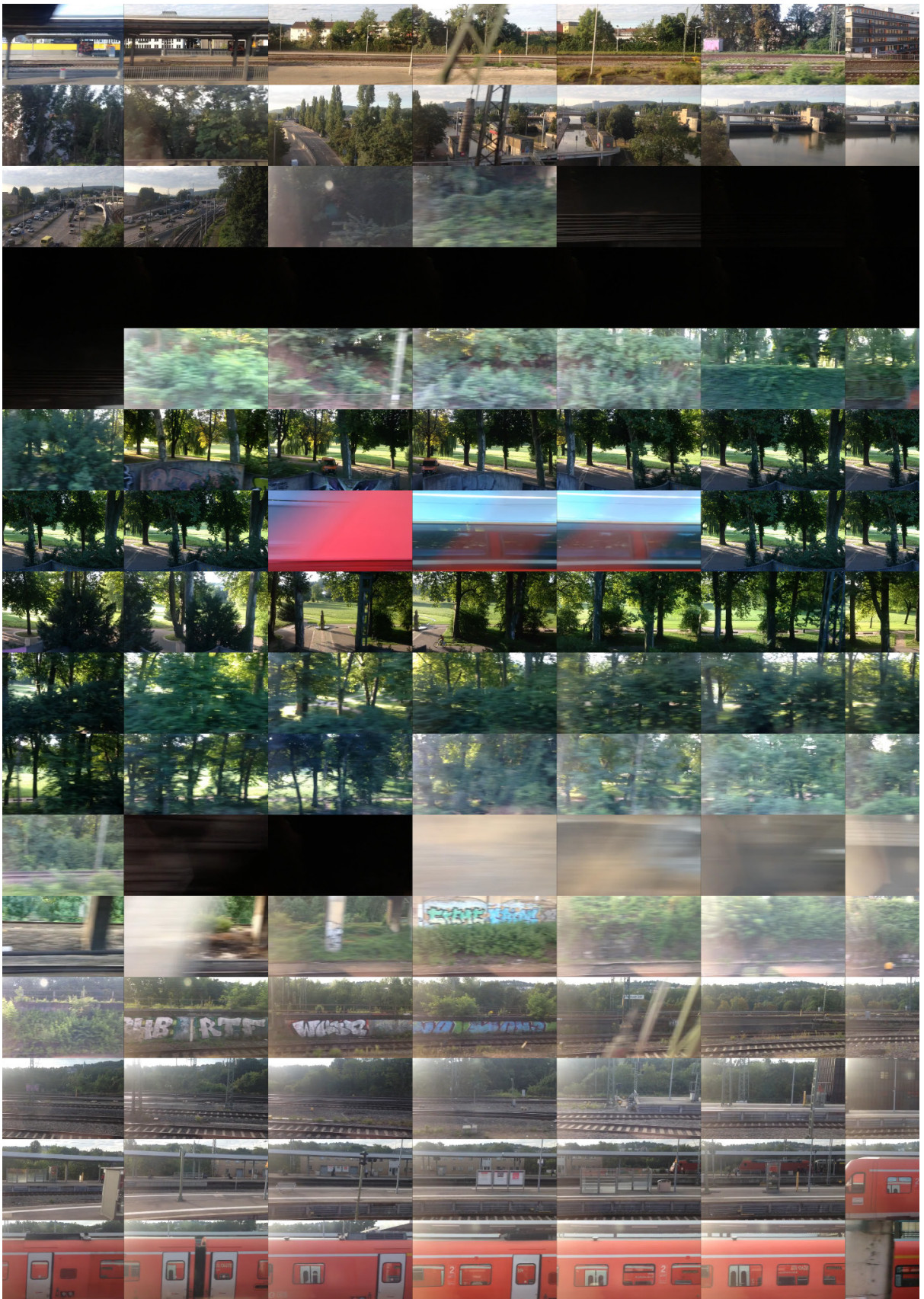
    currentImage++;
2   var nextTime = map(currentImage, 0, imageCount,
3                       0, movie.duration());
    movie.time(nextTime);

4   gridX++;
    if (gridX >= tileCountX) {
      gridX = 0;
      gridY++;
    }

5   if (currentImage >= imageCount) noLoop();
  }
}
```

Keys: S: Save image

- 1 Every time the `draw()` function is run, an image is selected from the video and depicted in the grid. The first image at time 0 can be placed immediately.
- 2 The next time in the video (`nextTime`) is calculated. The variable `currentImage` (a number between 0 and `imageCount`) is converted to a second value between 0 and the entire playing time of the video.
- 3 Using the `time()` function, the program jumps to the newly calculated time.
- 4 To define the next tile, `gridX` is increased by 1. If the end of the line has been reached, the program jumps to the first image of the next line by setting `gridX` to 0 and increasing `gridY` incrementally.
- 5 The end of the program is reached when all the tiles are filled with images.



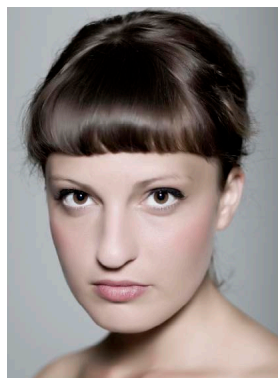
→ P_4_2_2_01 Fifty-five images from a two-and-a-half-minute video clip, filmed on the way to the main train station in Stuttgart, Germany.

P.4.3.1 Graphics from pixel values

Pixels, the smallest elements of an image, can serve as the starting point for the composition of portraits. In this example, each pixel is reduced to its color value. These values modulate design parameters such as rotation, width, height, and area. The pixel is completely replaced by a new graphic representation, and the portrait becomes somewhat abstract.

→ P_4_3_1_01

The pixels of an image are analyzed sequentially and transformed into other graphic elements. The key to this is the conversion of the color values of pixels (RGB) into the corresponding gray values, because—in contrast to the pure RGB values—these can be practically applied to design aspects such as line width. It is advisable to reduce the resolution of the source image first.



→ P_4_3_1_01 → Photograph: Tom Ziora
The gray value of each pixel defines the size of its diameter; the pixels' original color values are kept.


```

1 for (var gridX=0; gridX<img.width; gridX++) {
  for (var gridY=0; gridY<img.height; gridY++) {
    var tileWidth = width / img.width;
    var tileHeight = height / img.height;
    var posX = tileWidth * gridX;
    var posY = tileHeight * gridY;

    img.loadPixels();
2    var c = color(img.get(gridX, gridY));
3    var grayscale = round(red(c) * 0.222 +
      green(c) * 0.707 + blue(c) * 0.071);

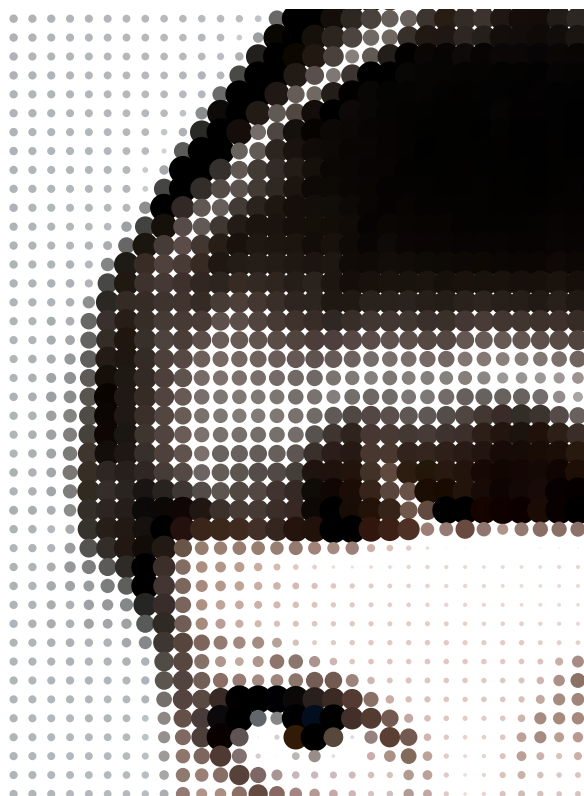
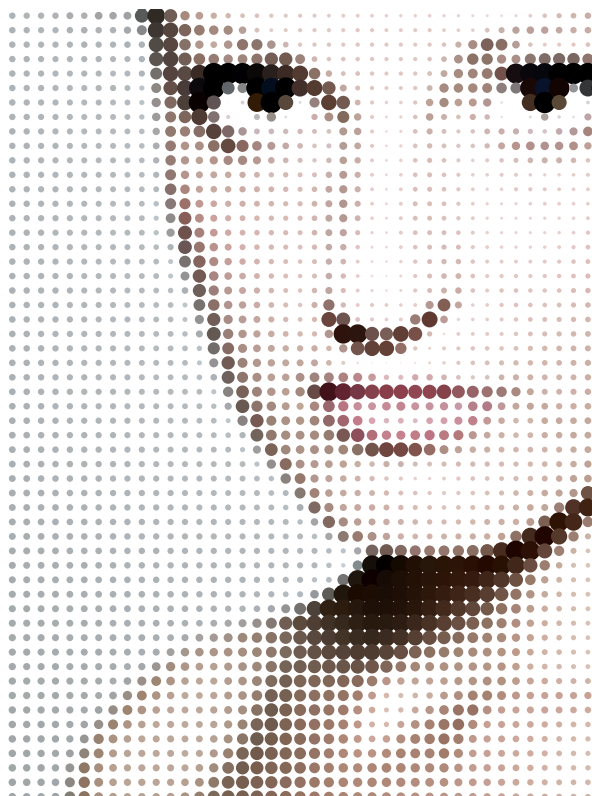
4    switch (drawMode) {
      case 1:
        var w1 = map(grayscale, 0, 255, 15, 0.1);
        stroke(0);
        strokeWeight(w1 * mouseXFactor);
        line(posX, posY, posX + 5, posY + 5);
        break;
      case 2:
        ...
    }
  }
}

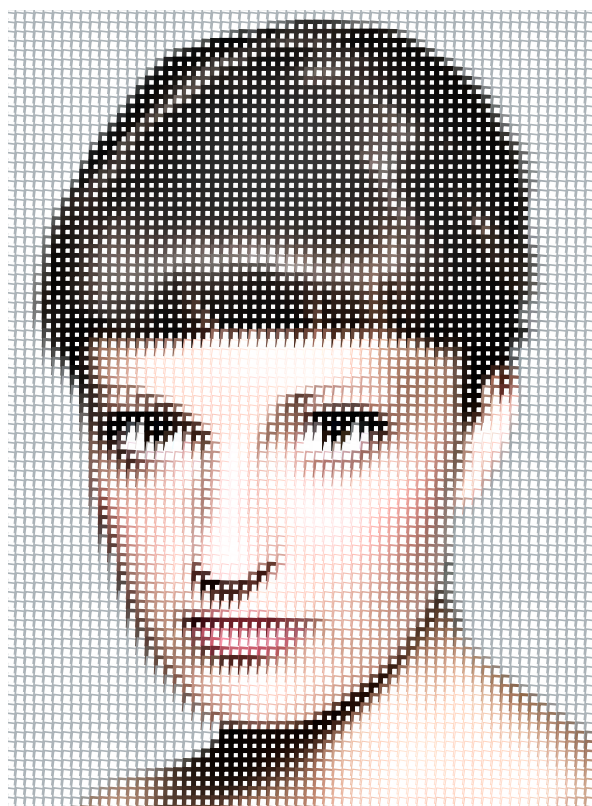
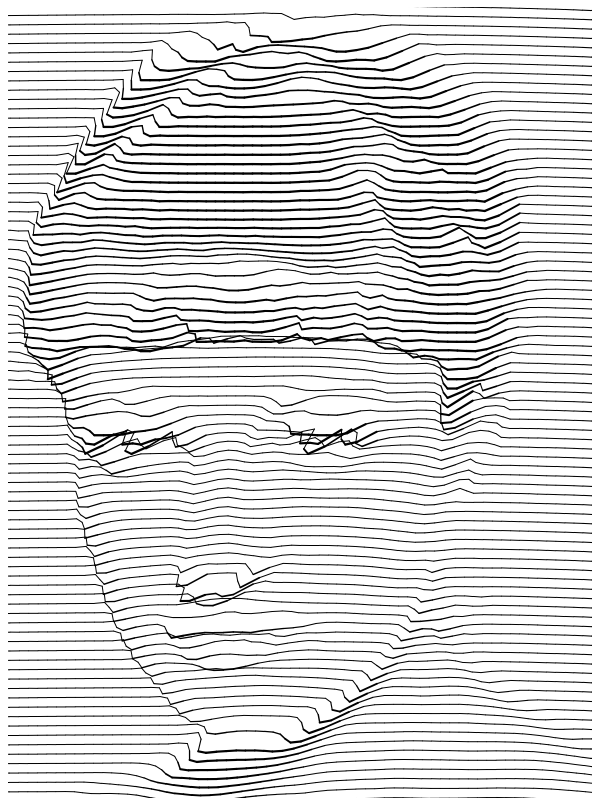
```

Mouse: Position x/y: Different parameters
(dependent on drawing mode)

Keys: S: Save image

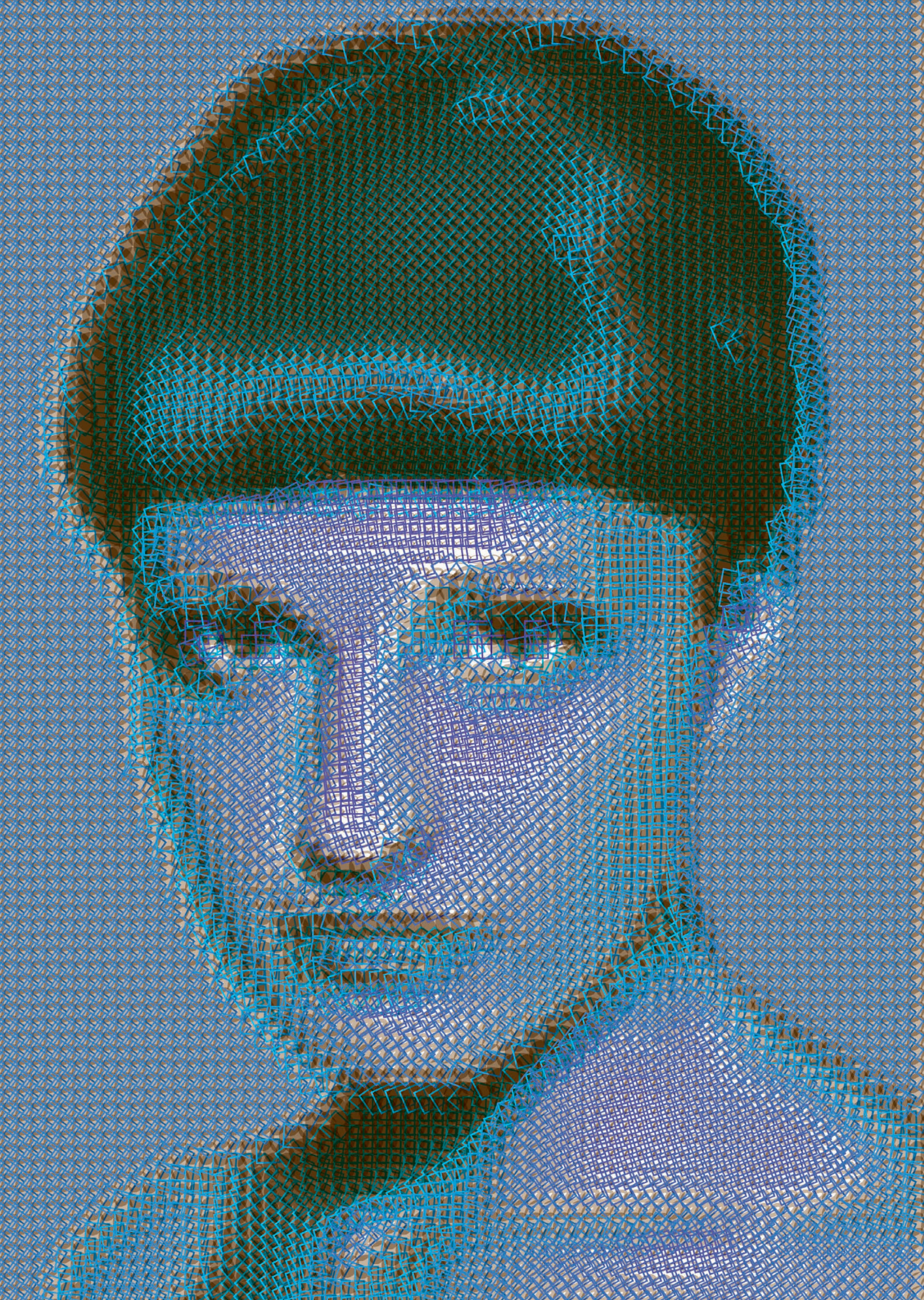
- 1 The width and height of the original image determines the resolution of the grid.
- 2 The color of the pixels at the current grid position (and thus the image) is defined.
- 3 In calculating the gray value, the values for red, green, and blue are weighted differently, whereby there are no absolutely correct weights since colors are both displayed and perceived differently. This gray value is used later to control individual parameters.
- 4 The program provides several drawing modes, `drawMode`, which can also be influenced by the horizontal mouse position. These were previously converted into a value between 0.05 and 1 and are available in the variable `mouseXFactor`.

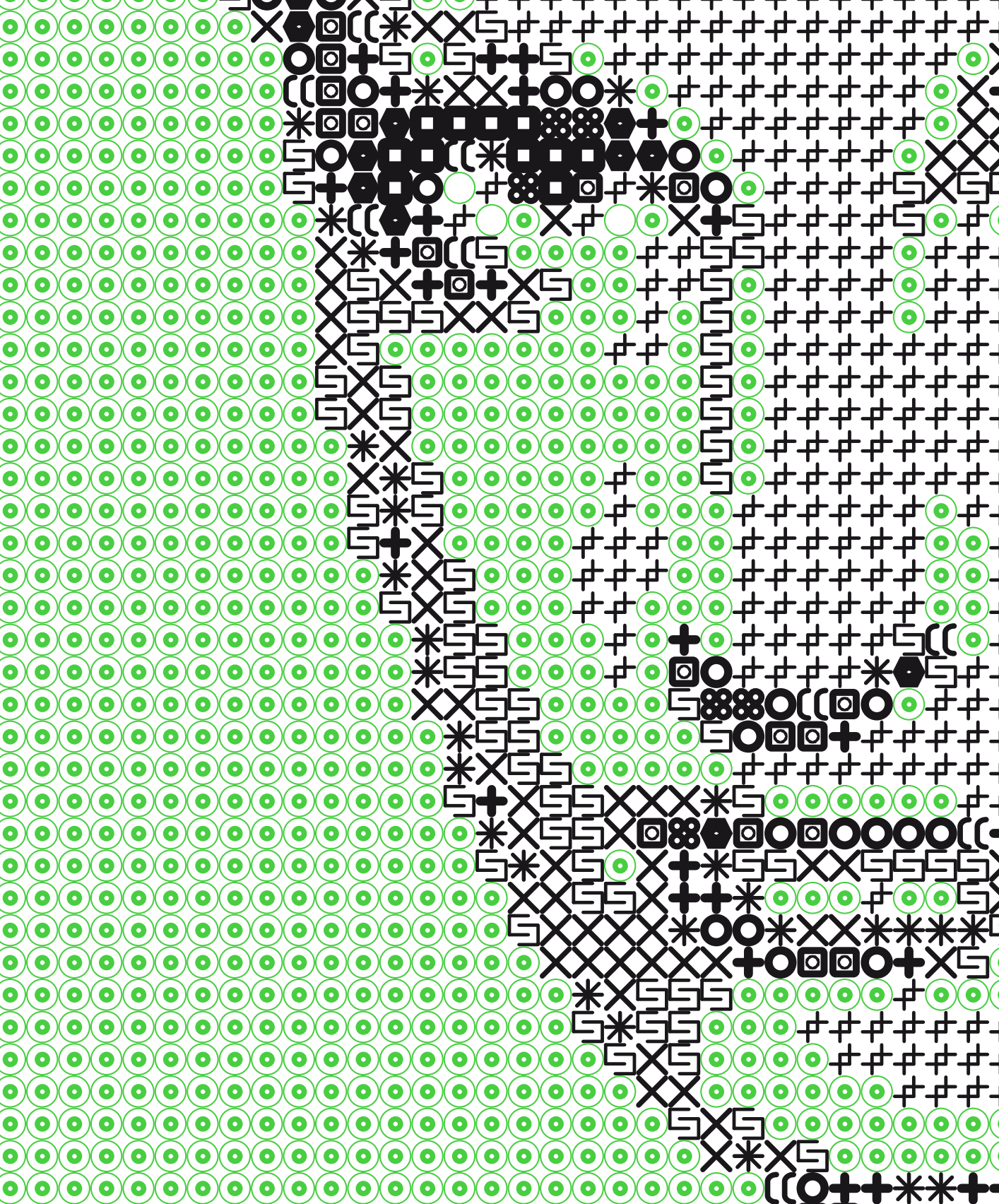




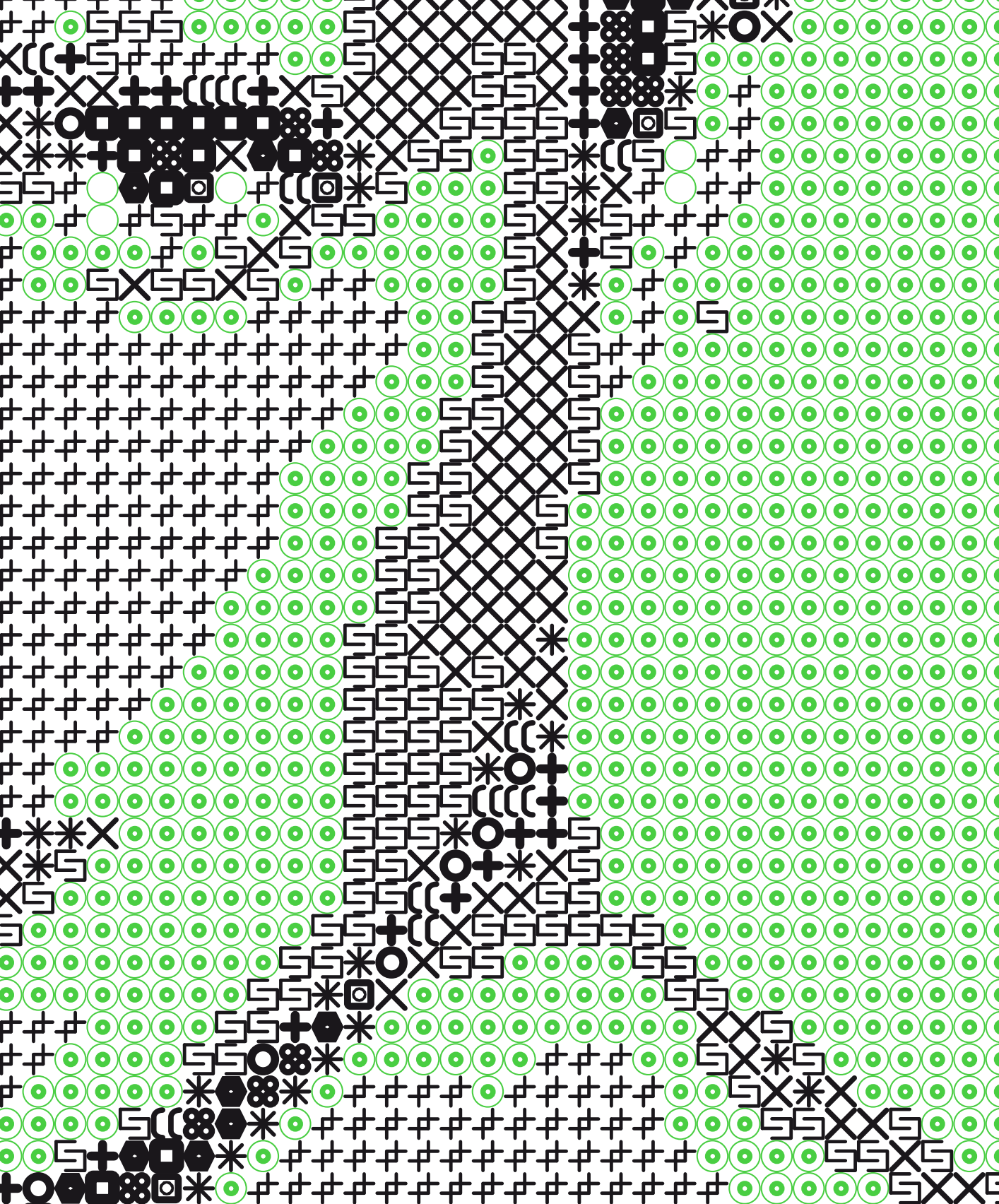
→ **P_4_3_1_01** The gray value defines the size, stroke value, rotation, and position of the elements.

→ **P_4_3_1_01** In this drawing mode (key 9), each pixel is represented by several color-distorting elements.





→ P_4_3_1_02 Pixels of various brightness are replaced here by SVG units. The SVG files have been sorted according to brightness using a supplementary program. Note that the files have been renamed (the brightness value forms the beginning of the file name).

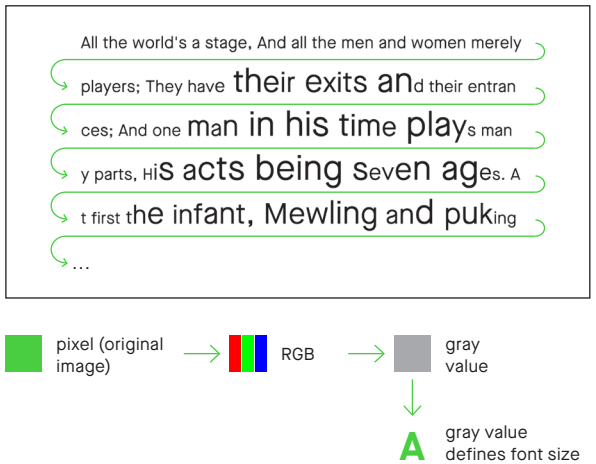


P.4.3.2 Type from pixel values

The following text image is ambiguous. It can be read for its meaning, or viewed at a distance and perceived as a picture. The pixels from the image control the configuration of the letters. The size of each letter depends on the gray values of the pixels in the original image and thereby creates an additional message.

→ P_4_3_2_01

A character string is processed letter by letter → P.3.1.1/P.3.1.2 and constructed row by row in the normal writing direction. Before a character is drawn, its position in display coordinates is matched to the corresponding position in the original image in pixel coordinates. Only a subset of the original pixels is used—merely those for which a corresponding character position exists. The color of the selected pixel can now be converted into its gray value and the gray value used to modulate the font size, for example.

[illegible]

infant, My mind and
 arms. Then the winning
 is satchel And shining morn
 like snail unwilling to sc
 green in the morning
 erful ballad Made to this mistress
 a wall of a
 erow
 and soldier Full of strange ca
 earded like the paria vealoping
 bubble reputation Even in the cannon s
 And then the justice in fair found b
 do of pama cut Fifth of wise severe
 a more than
 six instances, And so he plays his p
 e sixth age shifts into the lean and si
 mental. With spectacles of no ide
 a wo
 lo
 Last sc
 ount, I histo
 and here
 e event
 a sista
 players
 ances
 a man
 end
 like snail
 e love
 eard made
 like the paria
 a quick in d
 tation Ever
 pon in d
 nial cut
 nces, And
 of wise saws
 the plays his

I've been looking for the meaning of the word "jealousy" for a long time. I've found it in the dictionary, but I don't think it's the same as the word I'm looking for. I've found it in the Bible, but I don't think it's the same as the word I'm looking for. I've found it in the dictionary, but I don't think it's the same as the word I'm looking for. I've found it in the Bible, but I don't think it's the same as the word I'm looking for.

→ **P 4 3 2 01** The color of a pixel can define the size or color of the letters, or both.


```

function draw() {
  ...
  var x = 0;
  var y = 10;
  var counter = 0;

1  while (y < height) {
    img.loadPixels();

2    var imgX = round(map(x, 0, width, 0, img.width))
    var imgY = round(map(y, 0, height, 0, img.height))
    var c = color(img.get(imgX, imgY));
    var grayscale = round(red(c) * 0.222 +
                          green(c) * 0.707 +
                          blue(c) * 0.071);

    push();
    translate(x, y);

3    if (fontSizeStatic) {
      textSize(fontSizeMax);
      if (blackAndWhite) fill(grayscale);
      else fill(c);
    } else {
      var fontSize = map(grayscale, 0, 255,
                        fontSizeMax, fontSizeMin);

4      fontSize = max(fontSize, 1);
      textSize(fontSize);
      if (blackAndWhite) fill(0);
      else fill(c);
    }

    var letter = inputText.charAt(counter);
    text(letter, 0, 0);
    var letterWidth = textWidth(letter) + kerning;

5    x += letterWidth;

    pop();

6    if (x + letterWidth >= width) {
      x = 0;
      y += spacing;
    }

    counter++;
    if (counter >= inputText.length) {
      counter = 0;
    }
  }
  noLoop();
}

```

Keys: 1: Switch character size mode
 2: Switch character color mode
 Arrow ↓/↑: Maximum character size -/+
 Arrow ←/→: Minimum character size -/+
 S: Save image

- 1 The writing process continues as long as the y-coordinate of the current writing position `y` is still less than the height of the display.
- 2 Using the `map()` function, the display coordinates are converted into image coordinates; e.g., the x-coordinate `x` is proportionally converted from a value between 0 and the display `width` to a corresponding value between 0 and the width of the image `img.width`.
- 3 Depending on the selected mode `fontSizeStatic` (key 1 or 2), the font size is set to a fixed value `fontSizeMax` or is varied by the gray value.
- 4 The value `fontSize` cannot be zero or negative, as this would cause problems. Therefore, the function `max()` ensures this value is at least 1.
- 5 The value of the variable `x` is increased by the character width.
- 6 If `x` is greater than or equal to the width of the drawing canvas, the line is wrapped. The y value then increases by the line spacing and `x` restarts from 0 on the far left of the drawing canvas.

→ **P_4_3_2_01** The size and color of the characters are defined by an underlying image.

→ P 4 3 2 01 Here the gray value of the pixels determines font size

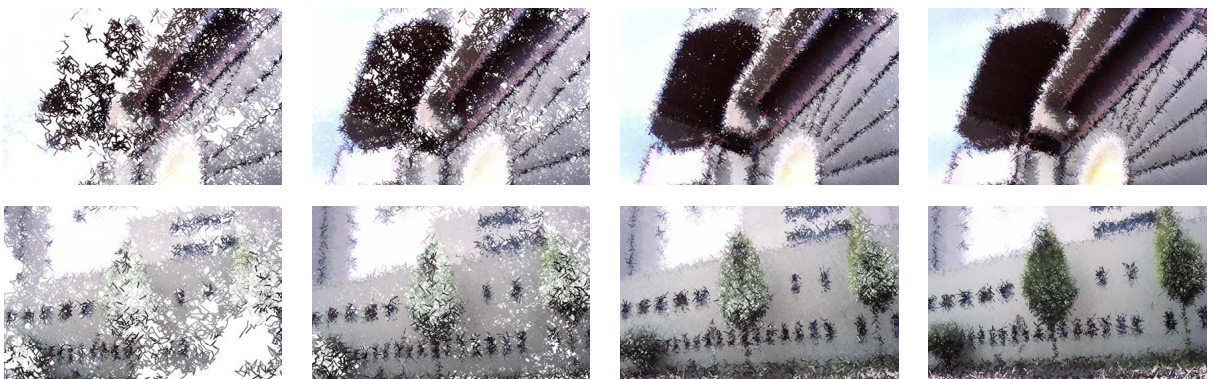
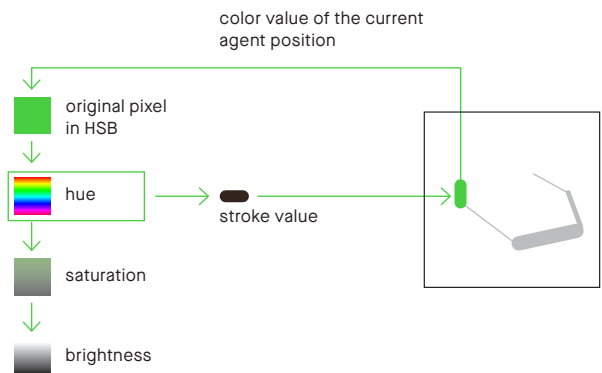
P.4.3.3 Real-time pixel values

The color values of pixels can again be translated into graphic elements but with two important differences: first, the pixels are constantly changing because the images come from a video camera, and second, pixels are translated sequentially by dumb agents that are constantly in motion rather than all at once. The motion captured by the camera and the migration of the agents thus can paint a picture right before our eyes.

→ P_4_3_3_01

A dumb agent moves over the drawing canvas. The color value of the current real-time video image is analyzed at each position and serves as a parameter for each color and stroke value. The mouse position defines the stroke length and the speed of the agent.

→P.2.2.1 Dumb agents



→ P_4_3_3_01 The agent's path gradually creates an image.

```

function setup() {
  ...
  video = createCapture(VIDEO, function(){
    streamReady = true
  });
  video.size(width, height);
  video.hide();
  ...
}

function draw() {
  if(streamReady) {
    for (var j = 0; j <= mouseX / 50; j++) {
      video.loadPixels();

      var pixelIndex = (((video.width - 1 - x)
        + y * video.width) * 4);
      var c = color(video.pixels[pixelIndex],
        video.pixels[pixelIndex + 1],
        video.pixels[pixelIndex + 2]);

      var cHSV = chroma(red(c), green(c), blue(c));
      strokeWeight(cHSV.get('hsv.h') / 50);
      stroke(c);

      diffusion = map(mouseY, 0, height, 5, 100);

      beginShape();
      curveVertex(x, y);
      curveVertex(x, y);

      for (var i = 0; i < pointCount; i++) {
        var rx = int( random(-diffusion, diffusion));
        curvePointX = constrain(x + rx, 0, width - 1);
        var ry = int( random(-diffusion, diffusion));
        curvePointY = constrain(y + ry, 0, height - 1);
        curveVertex(curvePointX, curvePointY);
      }

      curveVertex(curvePointX, curvePointY);
      endShape();

      x = curvePointX;
      y = curvePointY;
    }
  }
}

```

Mouse: Position x: Drawing speed

Position y: Direction

Keys: Arrow ↓/↑: Number of curve points -/+

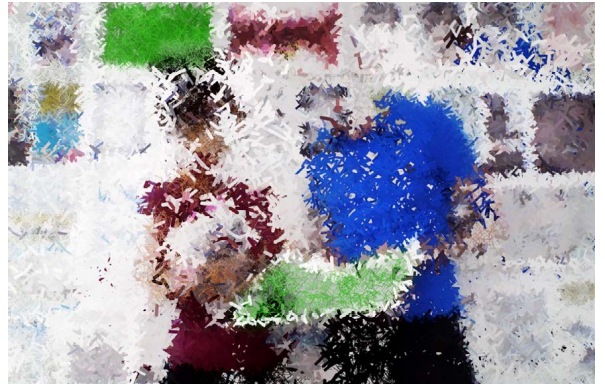
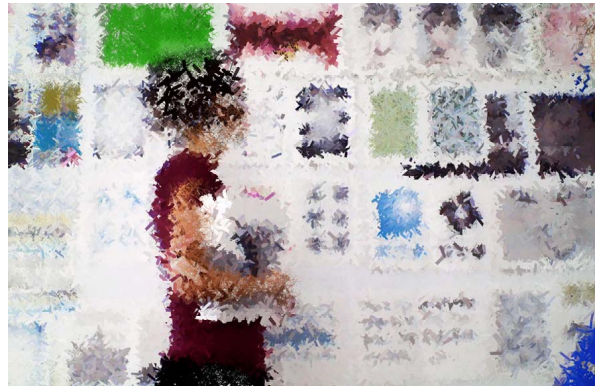
Q: Stop drawing

W: Continue drawing

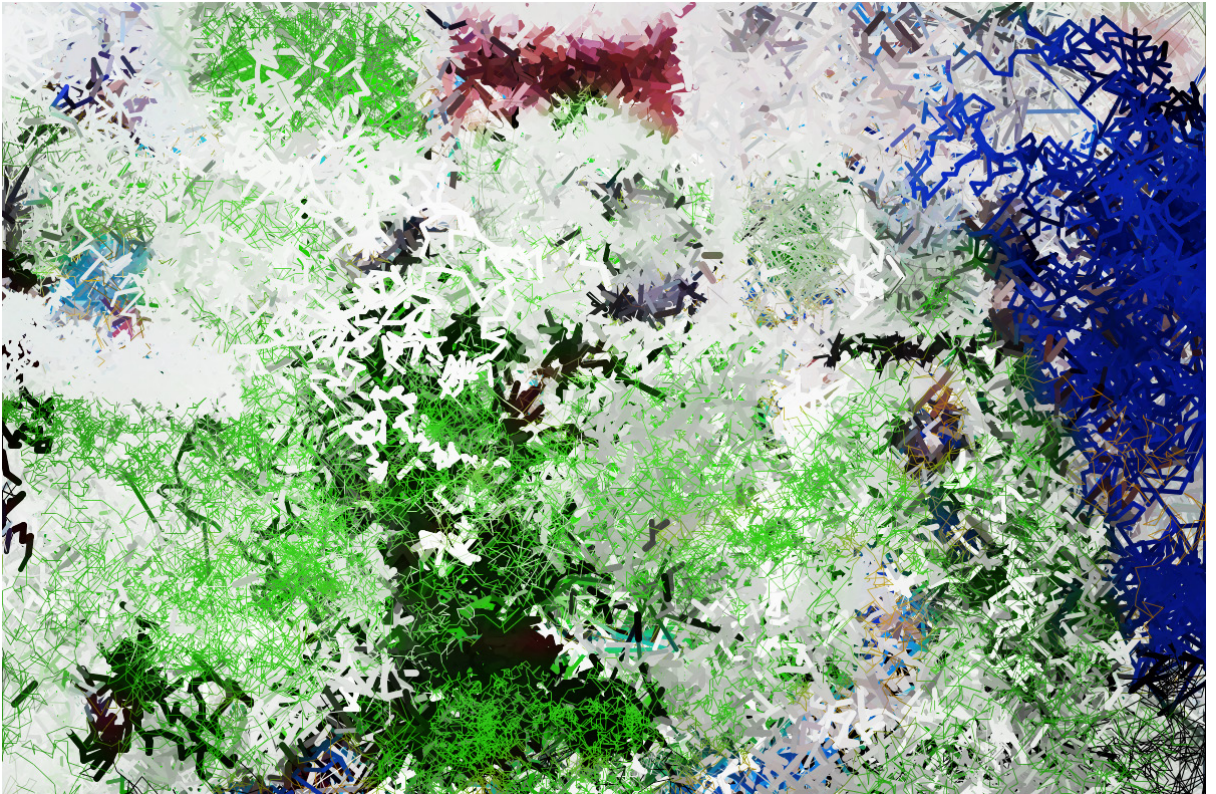
Arrow ←/→: Minimum font size -/+

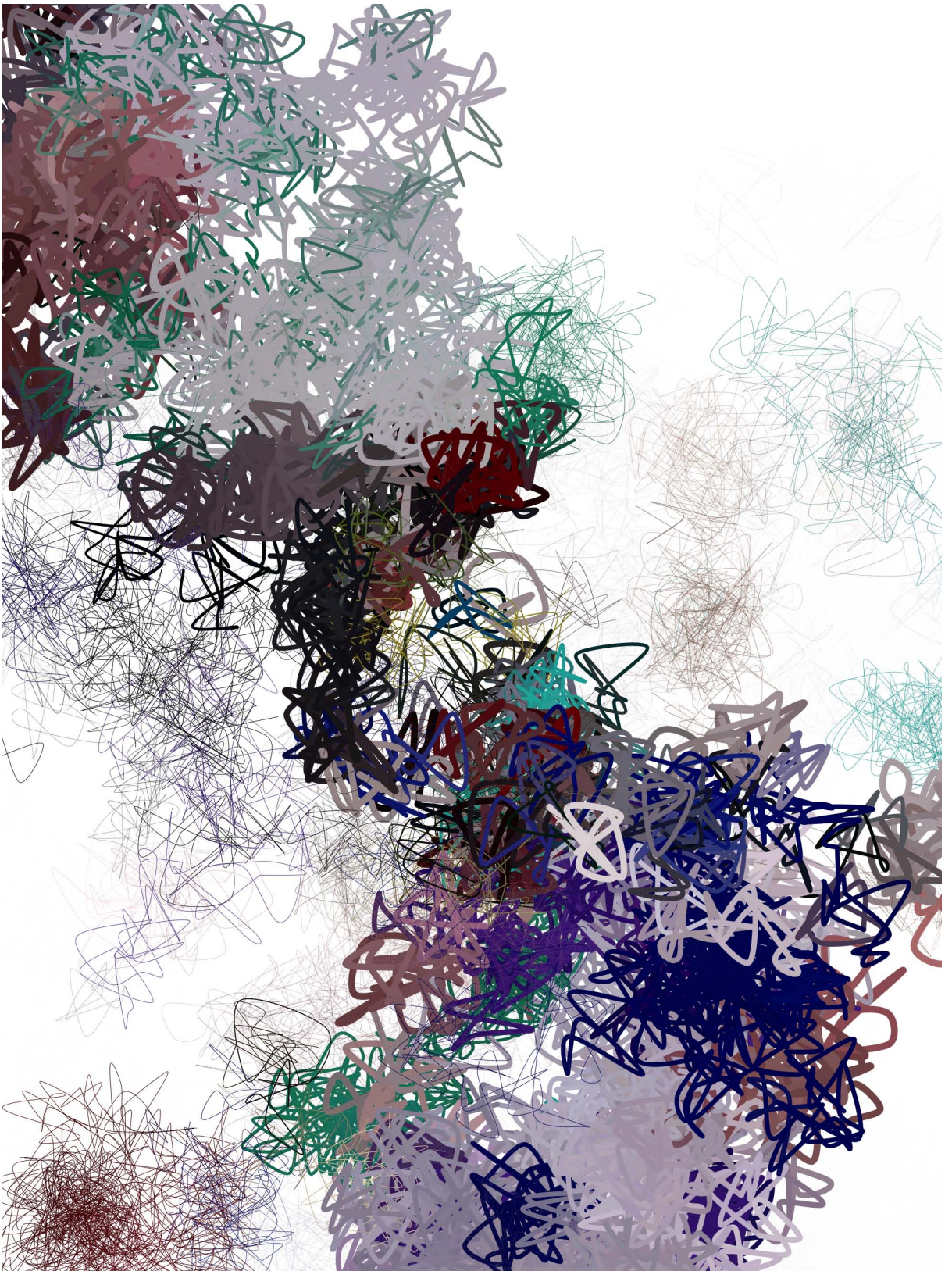
S: Save image

- 1 The function `createCapture()` creates a video element to access images on the webcam.
- 2 The live video images from the connected video camera are reduced to the size of the drawing canvas.
- 3 The `hide()` command prevents the video image from being automatically displayed on the drawing canvas.
- 4 If the video signal is available, the pixels of the current video image are loaded.
- 5 As in a static pixel image, the pixels in a video image are also numbered row by row. Therefore, the pixel index has to be calculated from the current writing position (x, y). When using webcams directed at the user, it is useful to mirror the video image horizontally using the calculation `video.width-1-x`.
- 6 The stroke value is set so it is defined by the hue of the pixel. The chroma.js library helps to convert RGB to HSV values.
- 7 The line element can now be drawn. The first curve point is placed on the current drawing position. This is done twice because the first and last points are not drawn when drawing lines with `curveVertex()`.
- 8 The variable `pointCount` now specifies how many curve points are to be drawn. The default value is 1, so only one line is drawn. The curve points are placed in random positions around the drawing position. The value `diffusion` specifies how large this area is.
- 9 The last curve point is specified as the new drawing position.

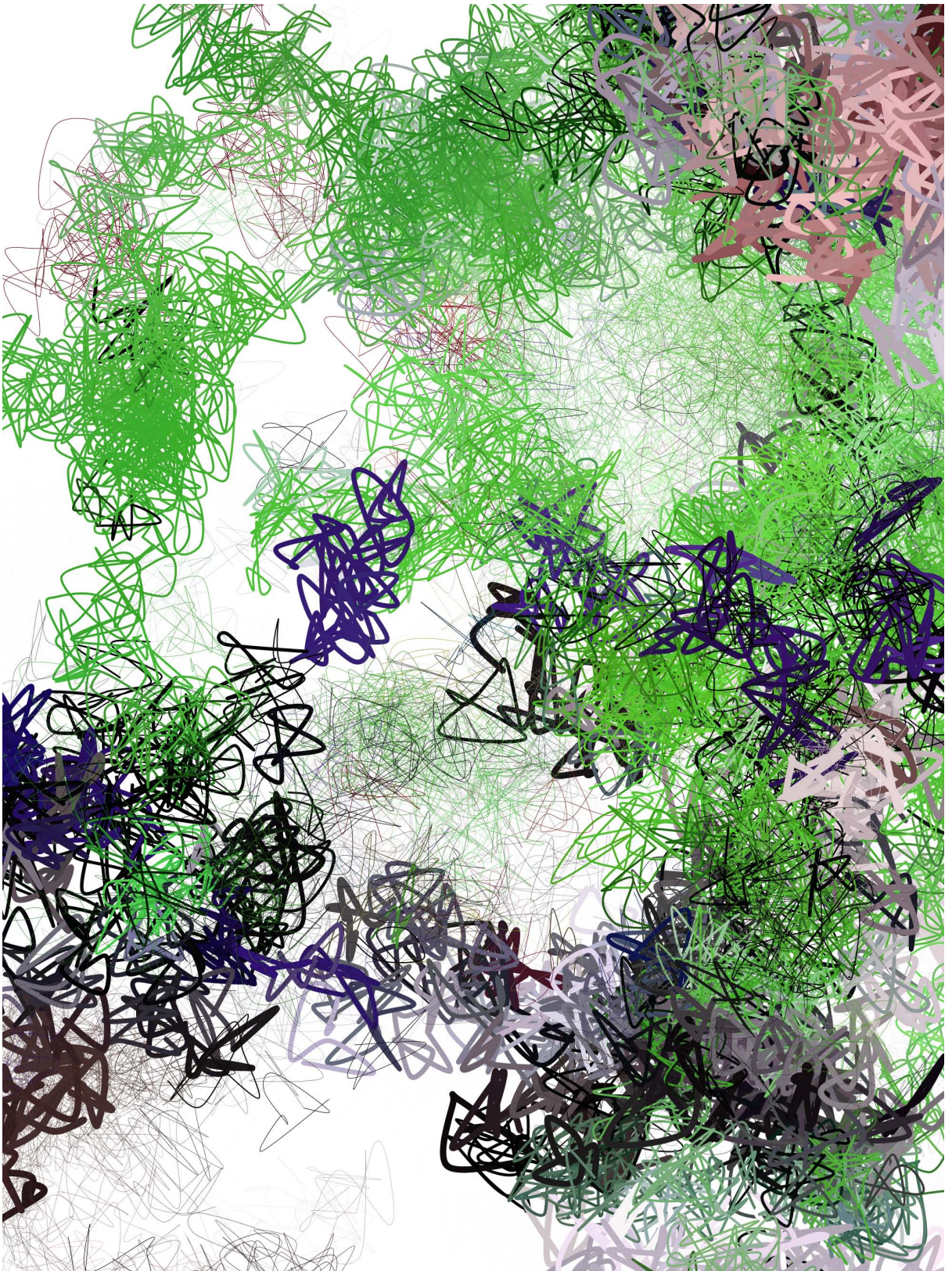


→ **P_4_3_3_01** The people leave tracks in the image with their movements. The length of the lines varies throughout the drawing process, whereby the image is sometimes more detailed and sometimes more abstract.





→ **P_4_3_02** Three agents move around the display in this version of the program. The first agent's stroke value is defined by the pixel's hue; the second's by the pixel's saturation; and the third's by the pixel's brightness.



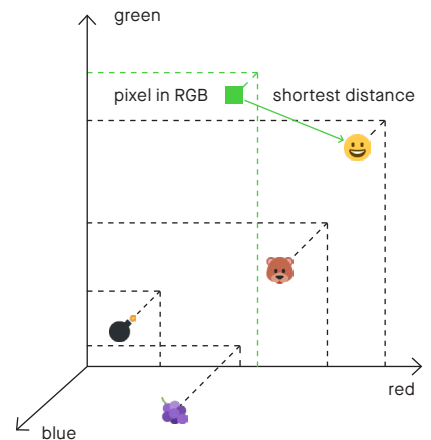
→ P_4_3_3_02 When a subject moves in front of the camera, a collection of seemingly random scribbles come together to represent the subject's form.

P.4.3.4 Emojis from pixel values

An emotional transformation from a raster element to a symbol: here the things that visualize feelings in an SMS become a small part of a greater whole. Any collection of thumbnails in this program can be the source material, and the pixel values determine which can join.

→ P_4_3_4_01

Color values can be interpreted as points in 3D space. For this program, it is necessary to find the shortest distance from a color value to a set of other color values, here the average colors of the individual emojis. There are various mathematical methods for this search. Particularly fast and comparatively easy to use is the search in a so-called k-dimensional tree. The functionality for this is provided by the library `kdTree.js`.




```
1 <script src="../../libraries/kd-tree/kdTree.js"
  type="text/javascript"></script>
2 <script src="data/emoji-average-colors.js"
  type="text/javascript"></script>


var emojis = {
3   "1f4a3": {"averageColor": {"r":57, "g":57, "b":52}},
4   "1f43b": {"averageColor": {"r":187, "g":111, "b":88}},
5   "1f600": {"averageColor": {"r":227, "g":181, "b":70}},
  ...
}


function preload() {
  img = loadImage("data/pic.png");
  icons = {};
  for (var name in emojis) {
    icons[name] = loadImage(emojisPath + "36x36/" +
                           name + ".png");
  }
}
```

1 Two additional scripts are required for this program. These are loaded in `index.html`.

2 Information about the average colors of emoji files and their respective file names is contained in `emoji-average-colors.js`. The calculation of the average colors takes place in an additional program: [→ P_4_3_4_emoji_color_analyser](#).

3 file `1f4a3.png`: 

4 file `1f43b.png`: 

5 file `1f600.png`: 

6 In the variable `icons`, all images of the emojis are loaded and can be called later using their names (`name`).

```
function setup(){
  ...
7  var colors = [];
  for (var name in emojis) {
    var col = emojis[name].averageColor;
    col.name = name;
    colors.push(col);
  }

8  var distance = function(a, b){
    return pow(a.r - b.r, 2) +
      pow(a.g - b.g, 2) +
      pow(a.b - b.b, 2);
  }

9  tree = new kdTree(colors, distance, ["r", "g", "b"]);
}
```

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

  for (var gridX = 0; gridX < img.width; gridX++) {
    for(var gridY = 0; gridY < img.height; gridY++) {
      var posX = tileWidth * gridX;
      var posY = tileHeight * gridY;

10  var c = color(img.get(gridX, gridY));

11  var nearest = tree.nearest(
    {r:red(c), g:green(c), b:blue(c)},
    1);

12  var name = nearest[0][0].name;
    image(icons[name], posX, posY,
      tileWidth, tileHeight);
  }
}
noLoop();
}
```

Keys: S: Save image

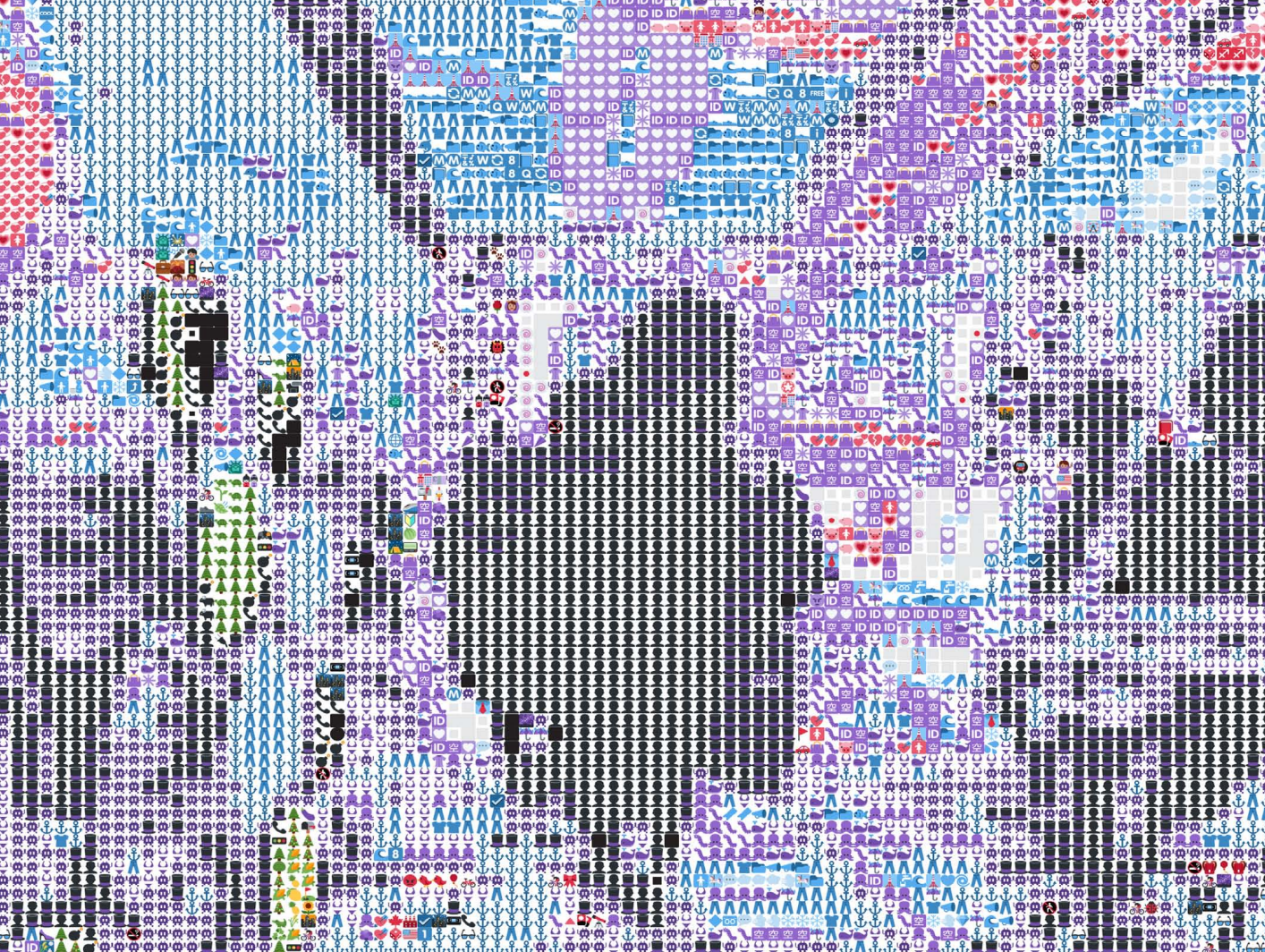
- 7 The search for the closest color begins here. Two things must be created: first, an array of points and their corresponding **r**, **g**, and **b** colors. Each entry in the **colors** array is also assigned the name **name**, in order to display the appropriate image files later.
- 8 Second, a function is defined that specifies how the distance between two points should be calculated. Typically, this is the length of the diagonal from color **a** to color **b** in the RGB color space.
- 9 With the help of the **kdTree** library, a so-called k-dimensional tree is created. For this, the newly created point list **colors** and the distance function **distance** are passed as parameters. In addition, a list of dimensions must be passed.
- 10 The image, **img**, to be displayed is scanned and the pixel color is stored in **c**.
- 11 The **nearest()** function in the **kdTree** library looks for the closest points to the one just passed. The last parameter indicates how many results should be returned; only one is needed here.
- 12 The search result is an array with the closest points. Each entry in it is itself an array with two elements: the point itself and its distance to the point passed in **nearest()**. In both arrays the first entry is required: **[0] [0]**. The image of the emoji can now be placed on the drawing canvas via the name.



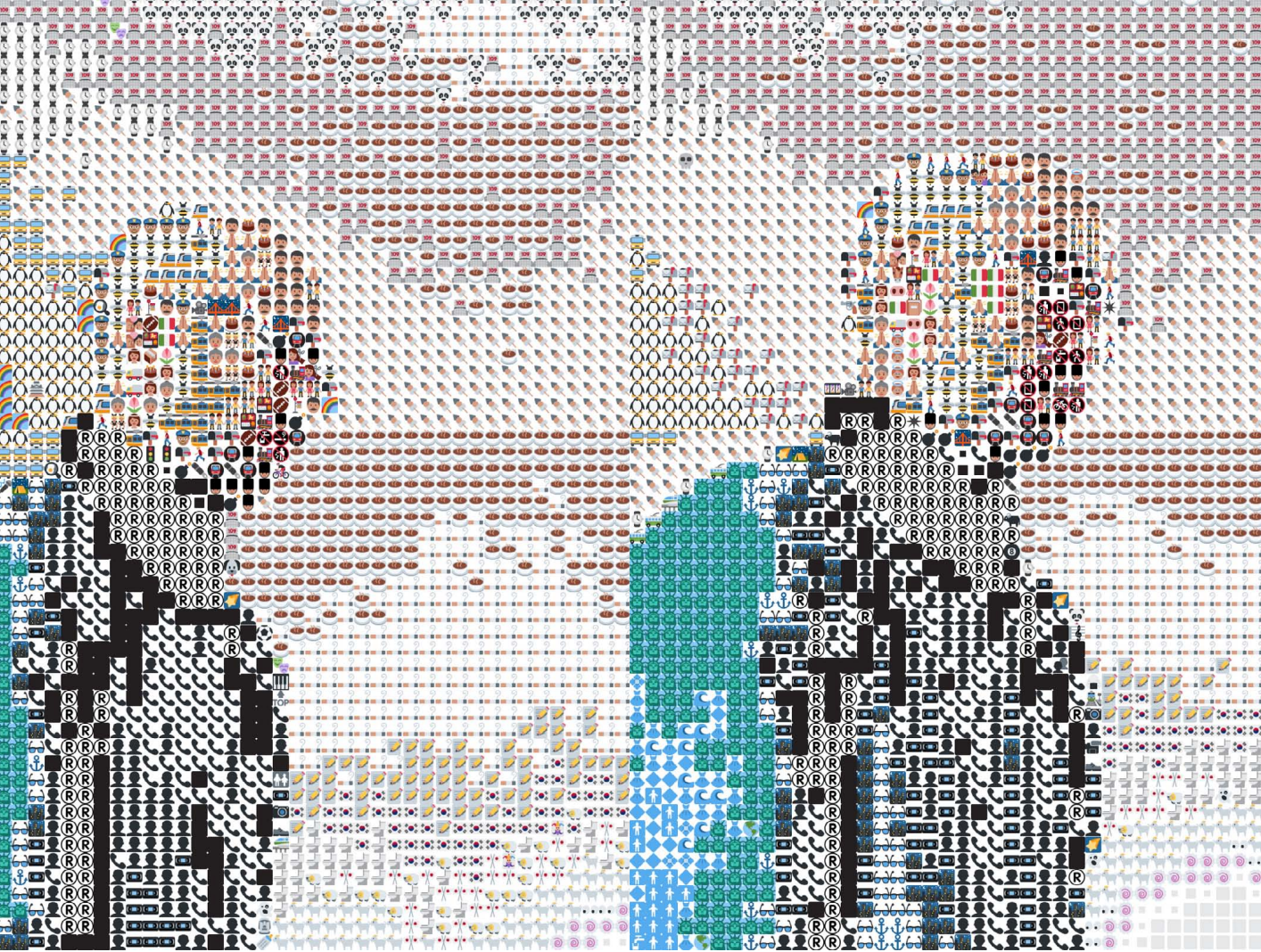
→ P_4_3_4_01 Every pixel becomes an emoji. Any image can be used, as long as it is large enough to include many color values.



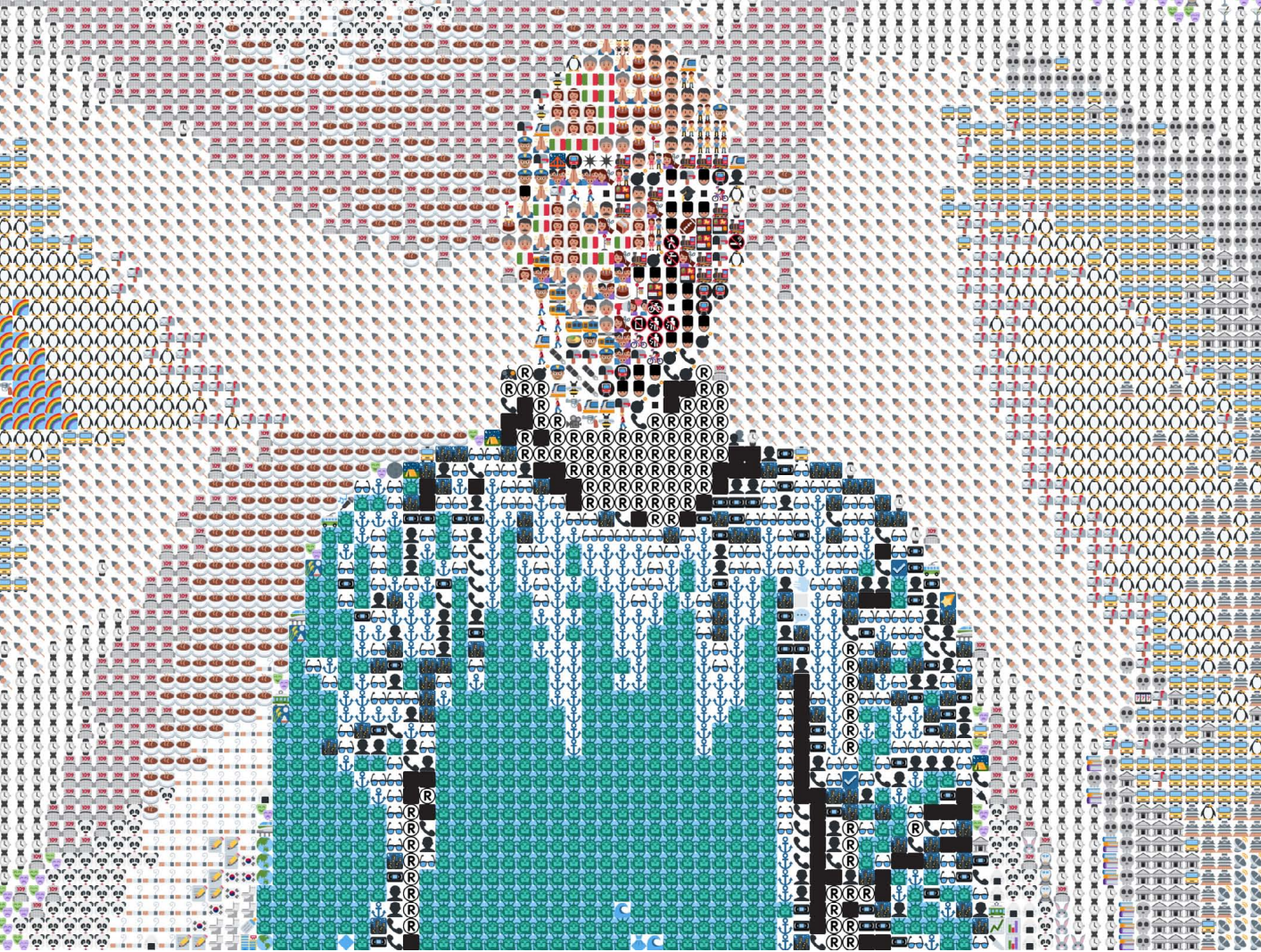
original photo



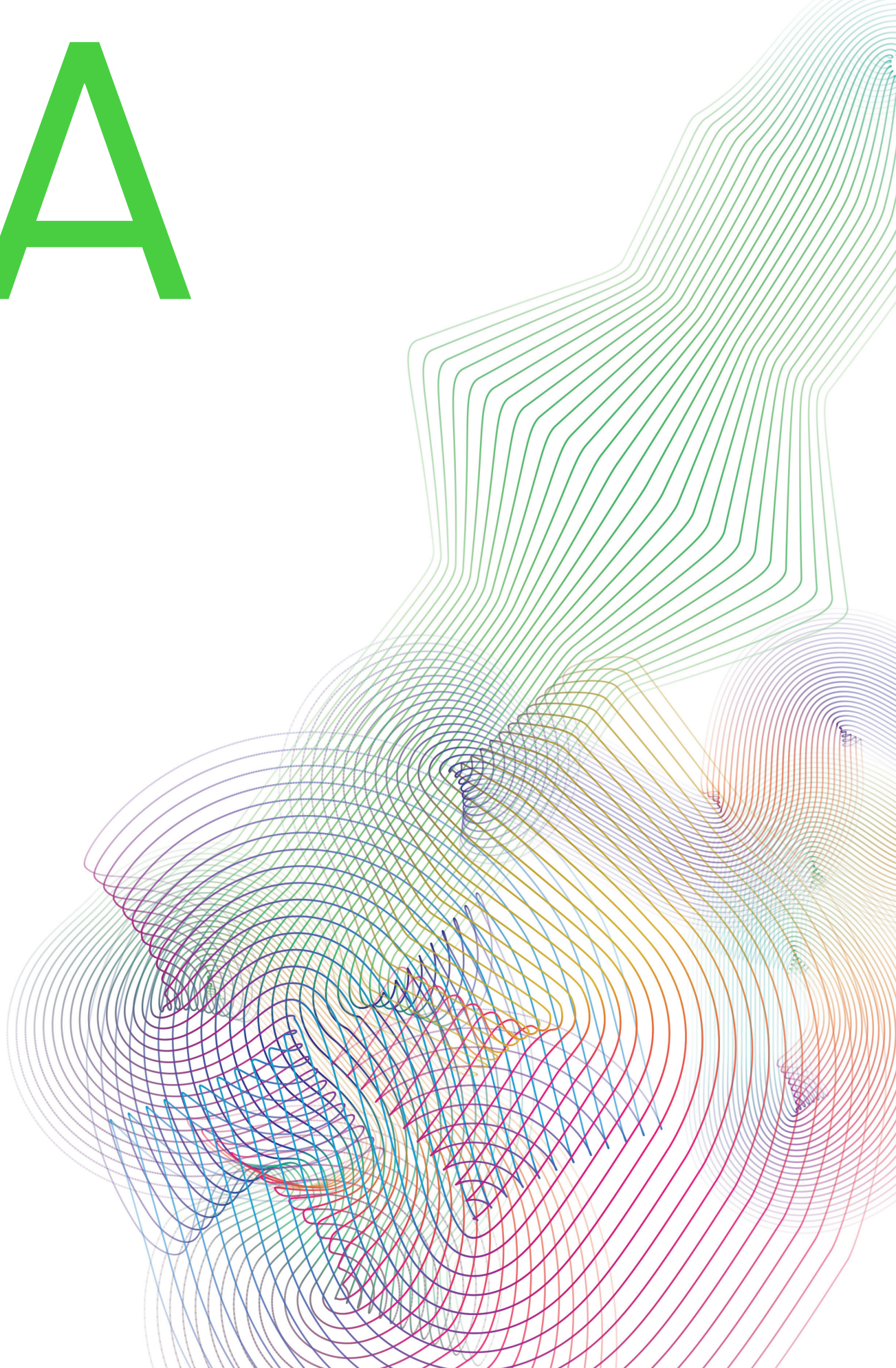
original photo



→ P_4_3_4_02 An image from a webcam could also be the basis of a rasterization in emojis.



A

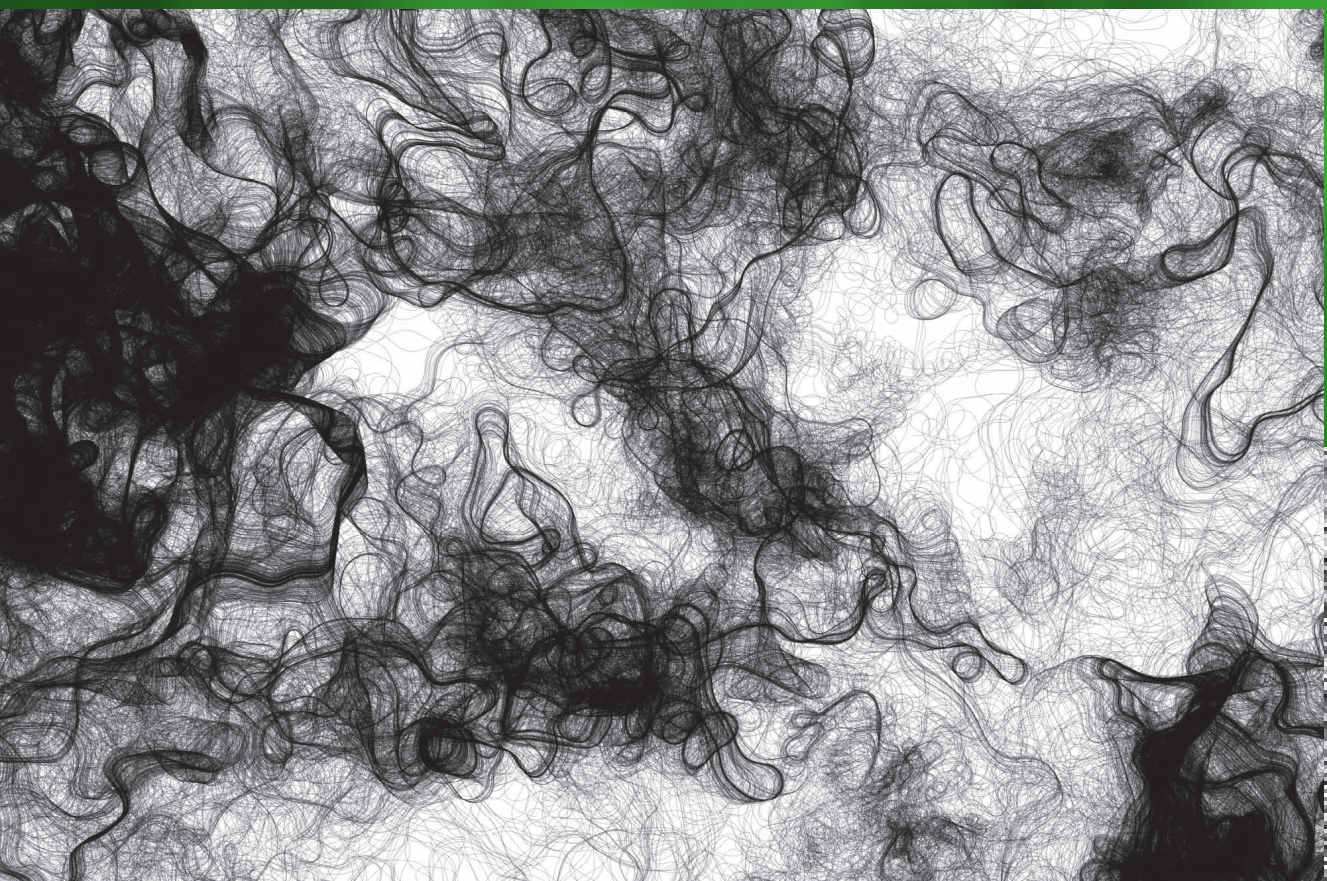


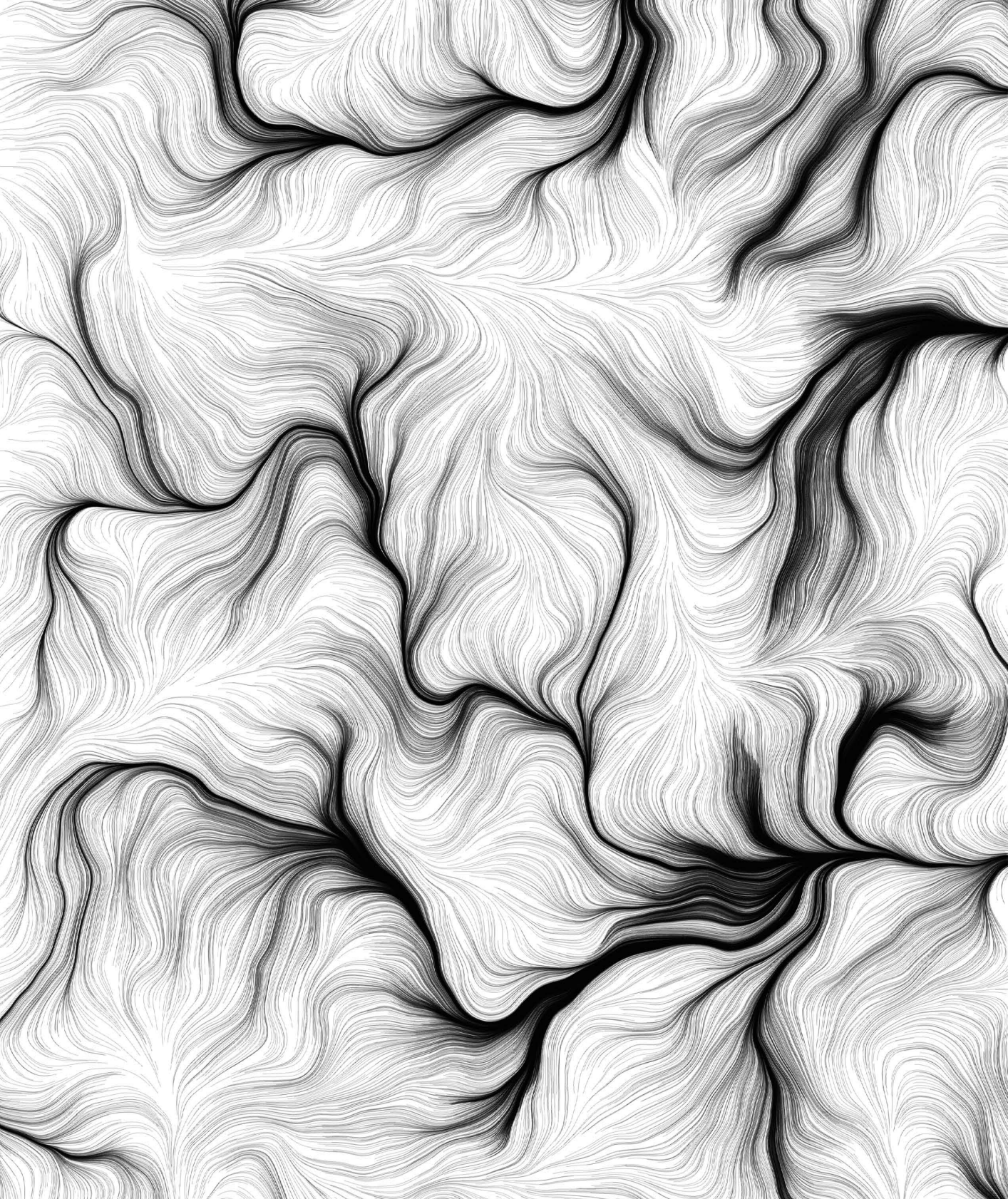
Appendix

A	Appendix	226
A.1	Looking ahead	228
A.2	Reflection	244
A.3	Bibliography	250
A.4	The authors	252
A.5	We thank	253
A.6	Copyright	254
A.7	Farewell	256

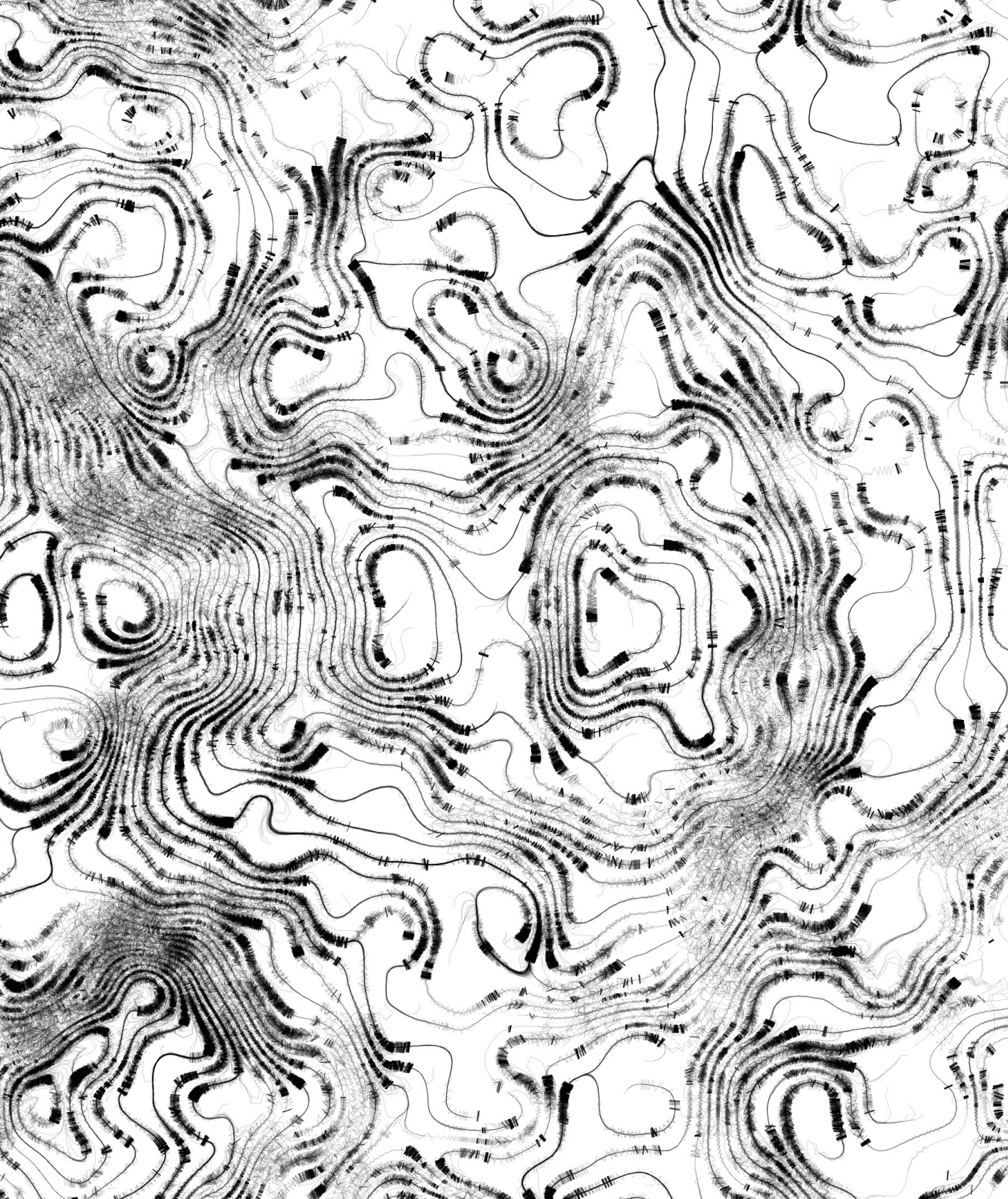
A.1 **Looking ahead**

Would you like to dive deeper into generative design? We hope so, because there is still much to discover! This chapter gives an overview on complex methods that are the basis for data visualization, 3D shape generation, wild particle animation, et cetera. You can find the code in the additional sketches on our website and our GitHub channel (links available on generative-gestaltung.de). After all the practical making of the previous chapters, this chapter is an invitation to reflect on what you've learned and to explore theoretical and conceptual connections in the context of generative design.



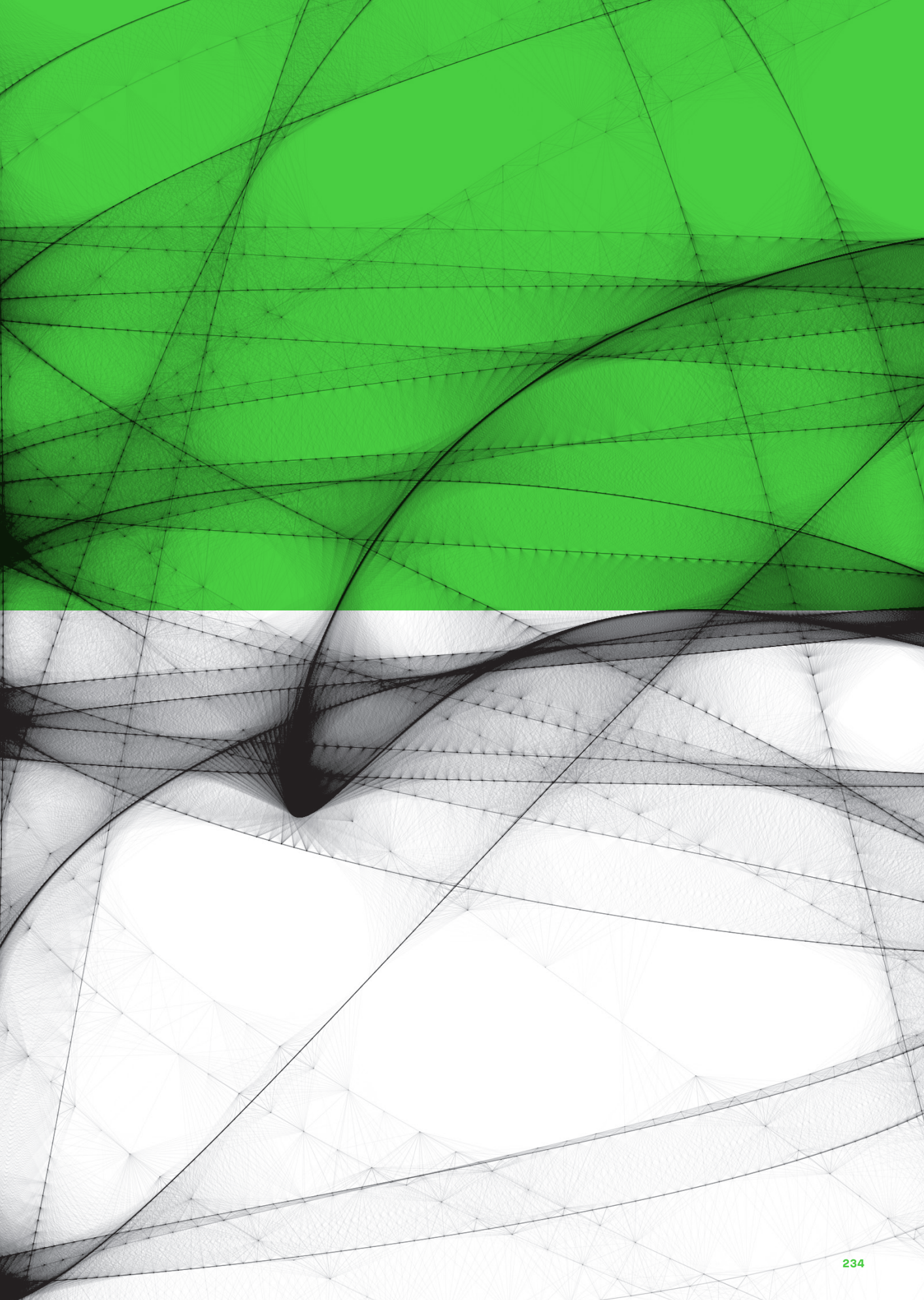


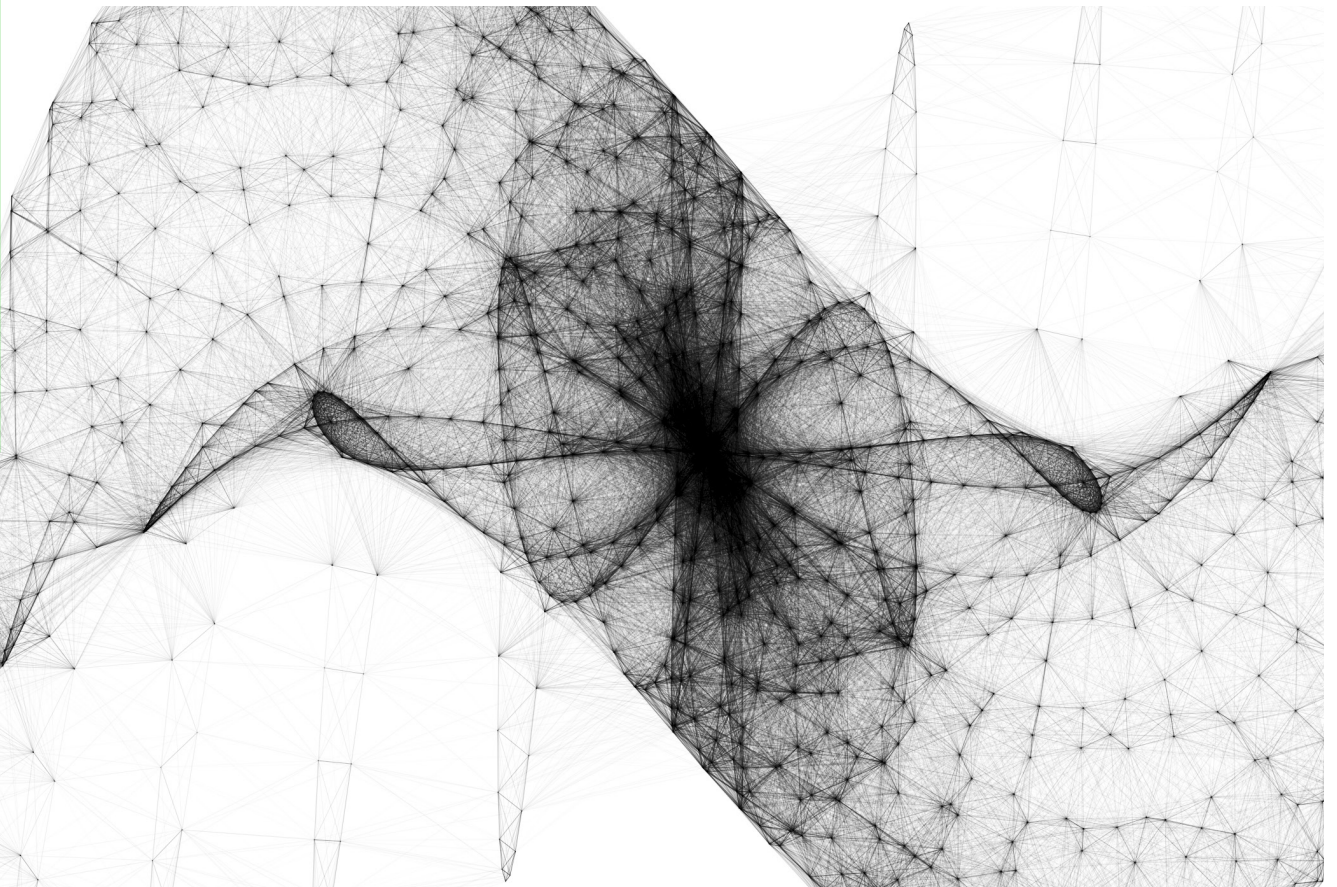
→ **M_1_5_02** A typical image that arises when noise-generated values control the directional movements of a swarm of agents. The agents are represented as dots and leave a trail as they are drawn continuously over the old image.



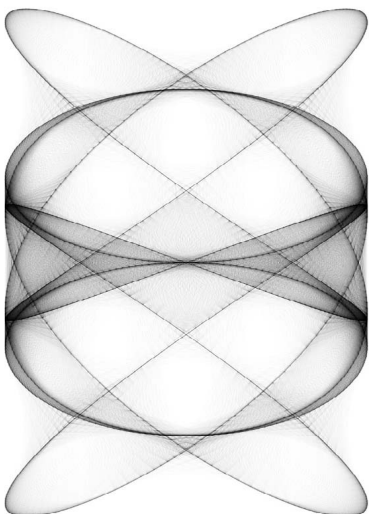
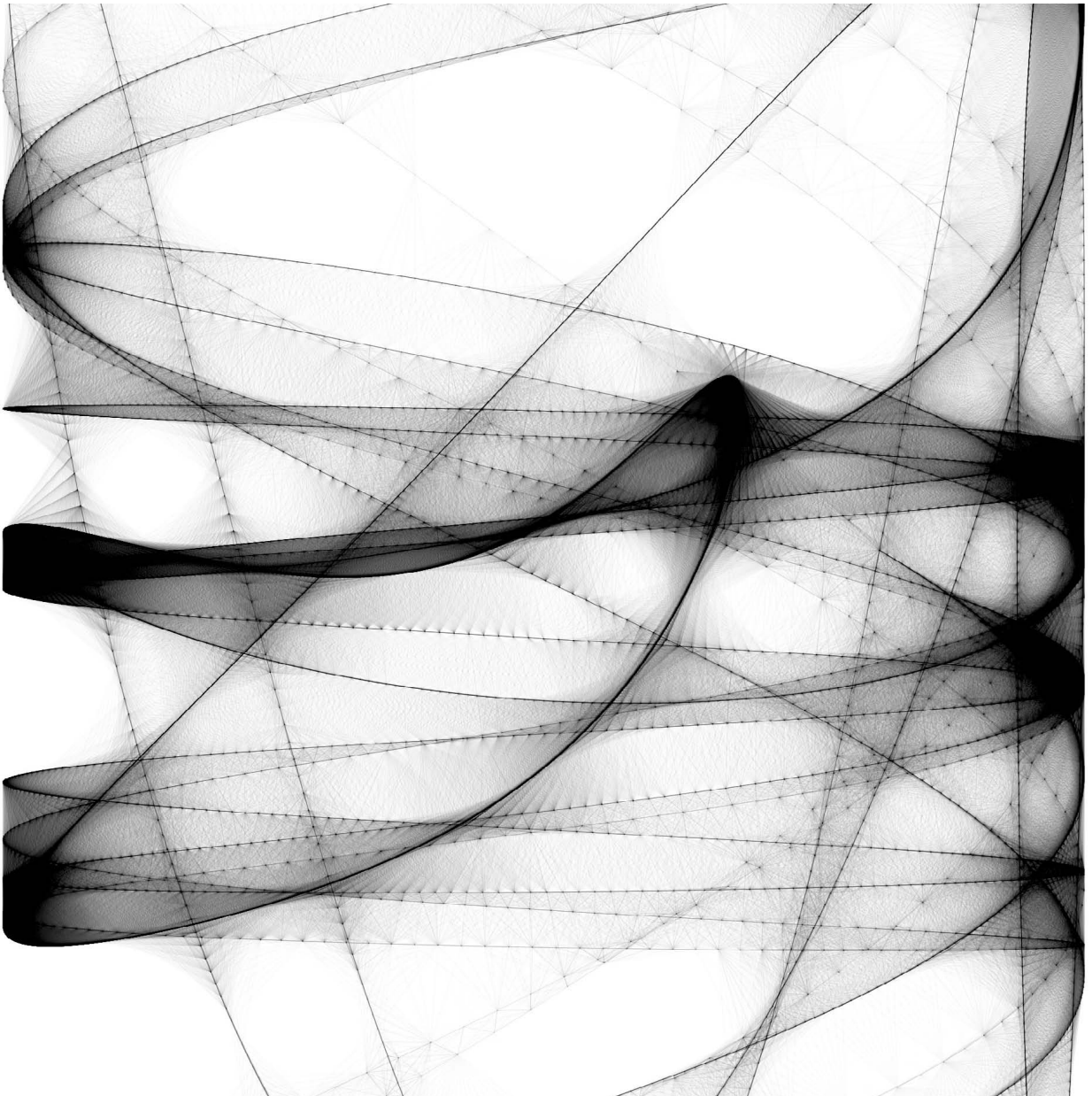


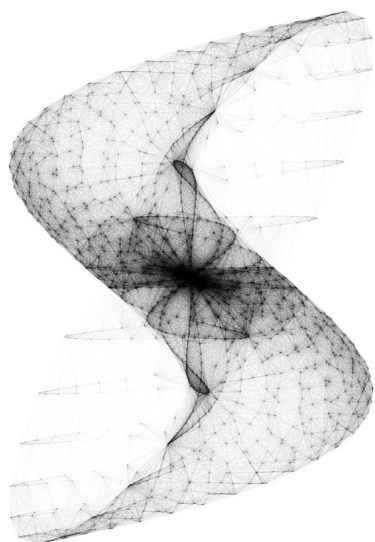
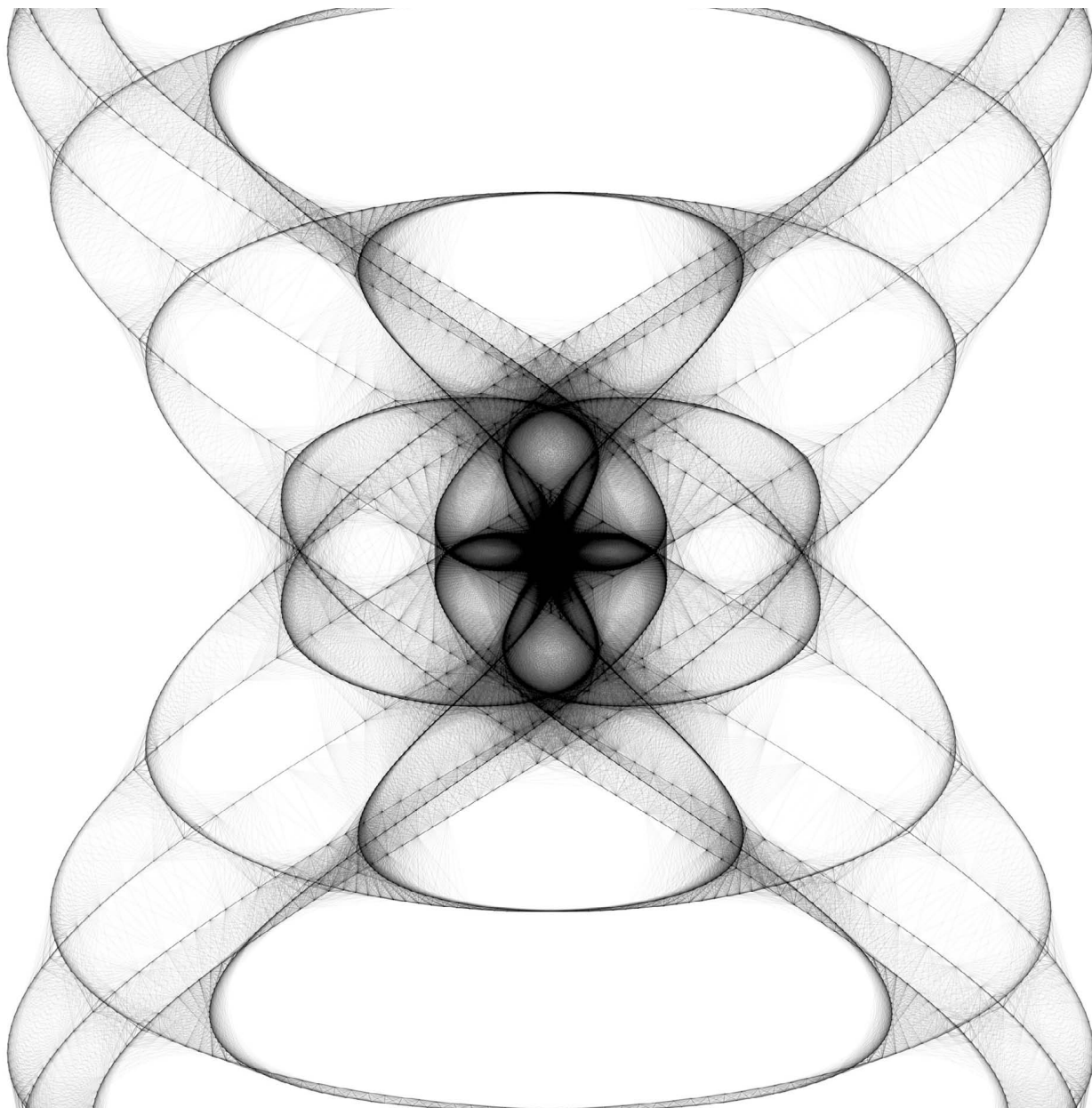


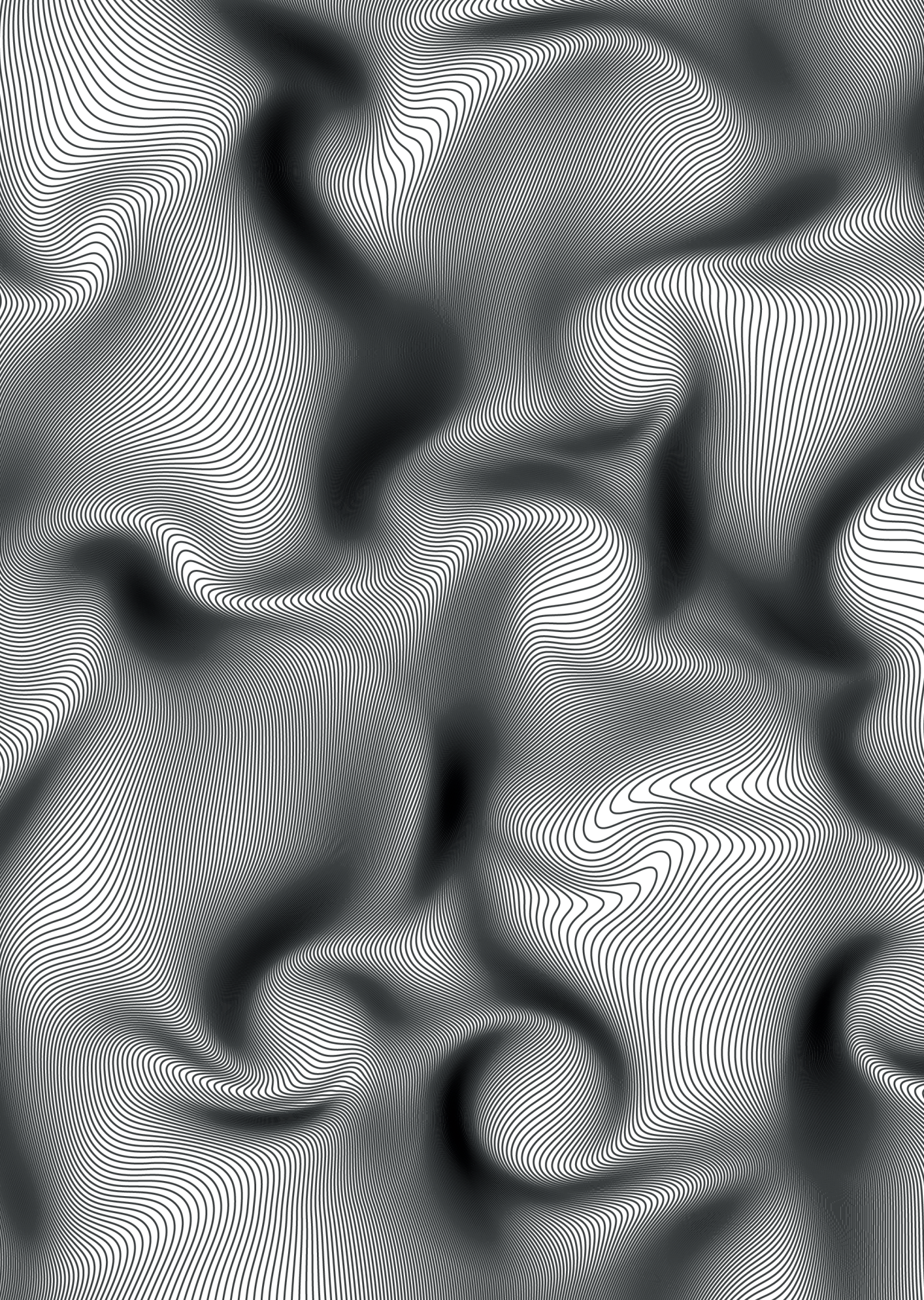


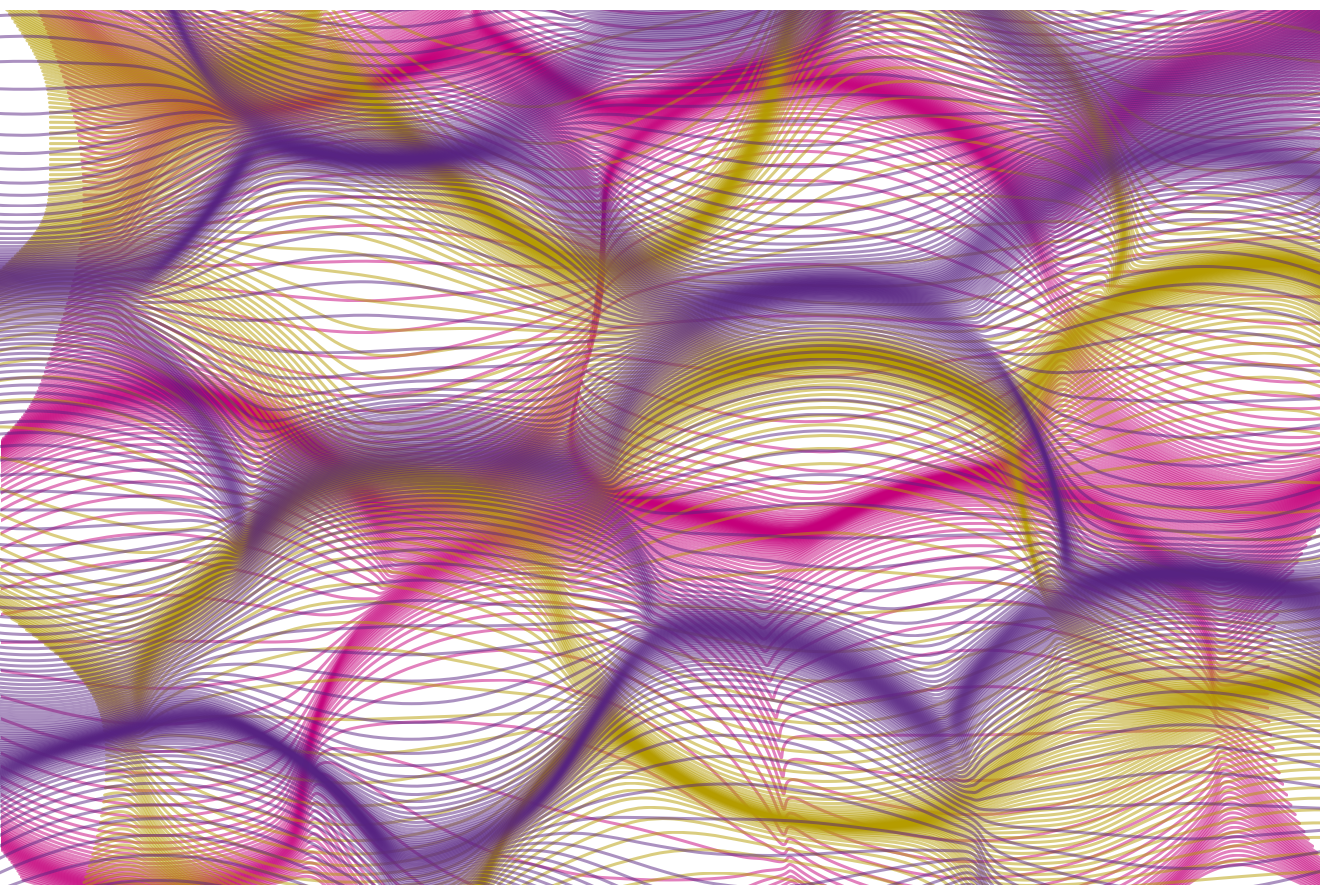


→ **M_2_5_02** The superimposition of different sinusoids results in Lissajous figures, named after the person who researched them, Jules Antoine Lissajous. If you connect the generated points not in sequence but rather each point with all the others, exciting netlike structures arise.

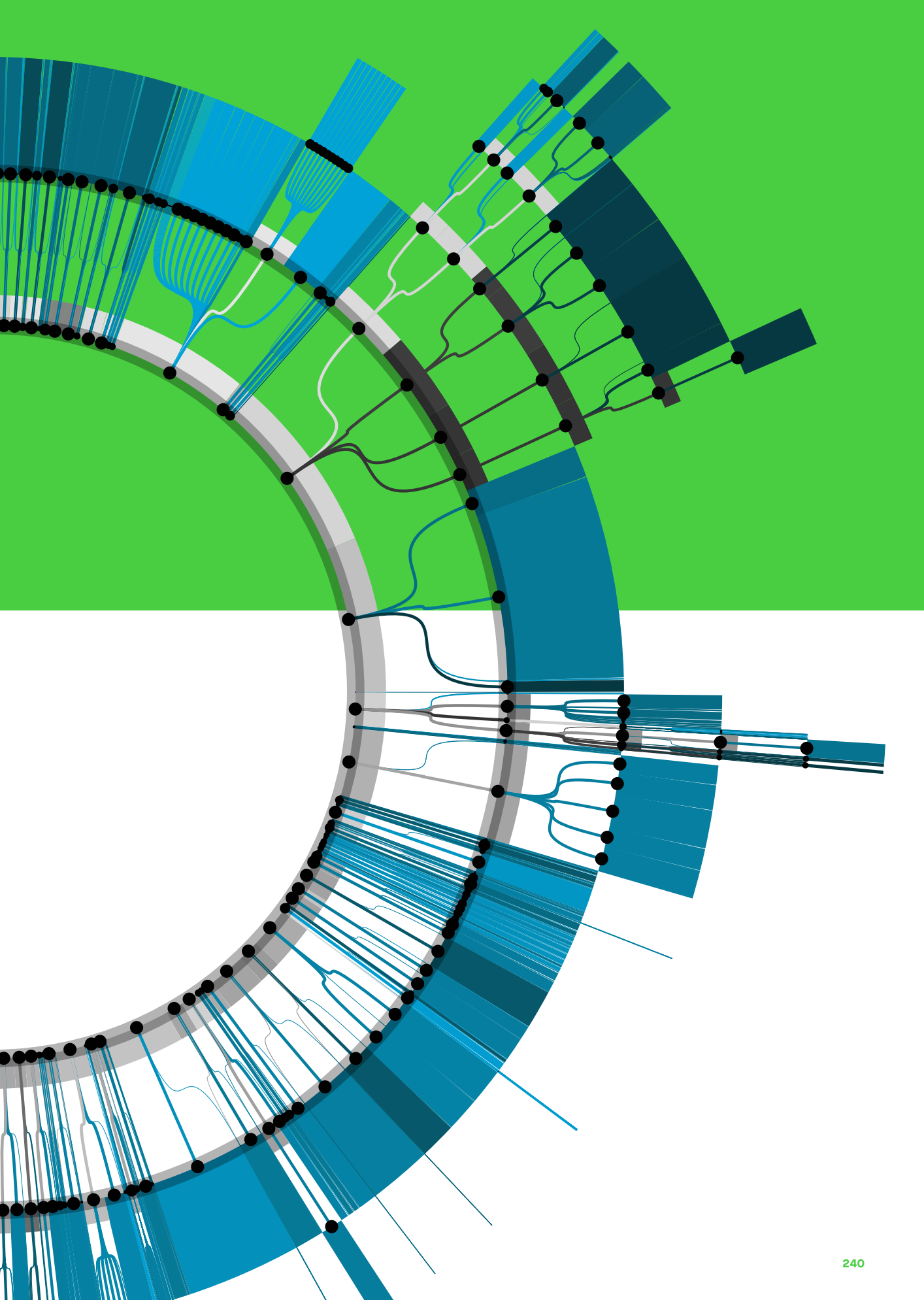


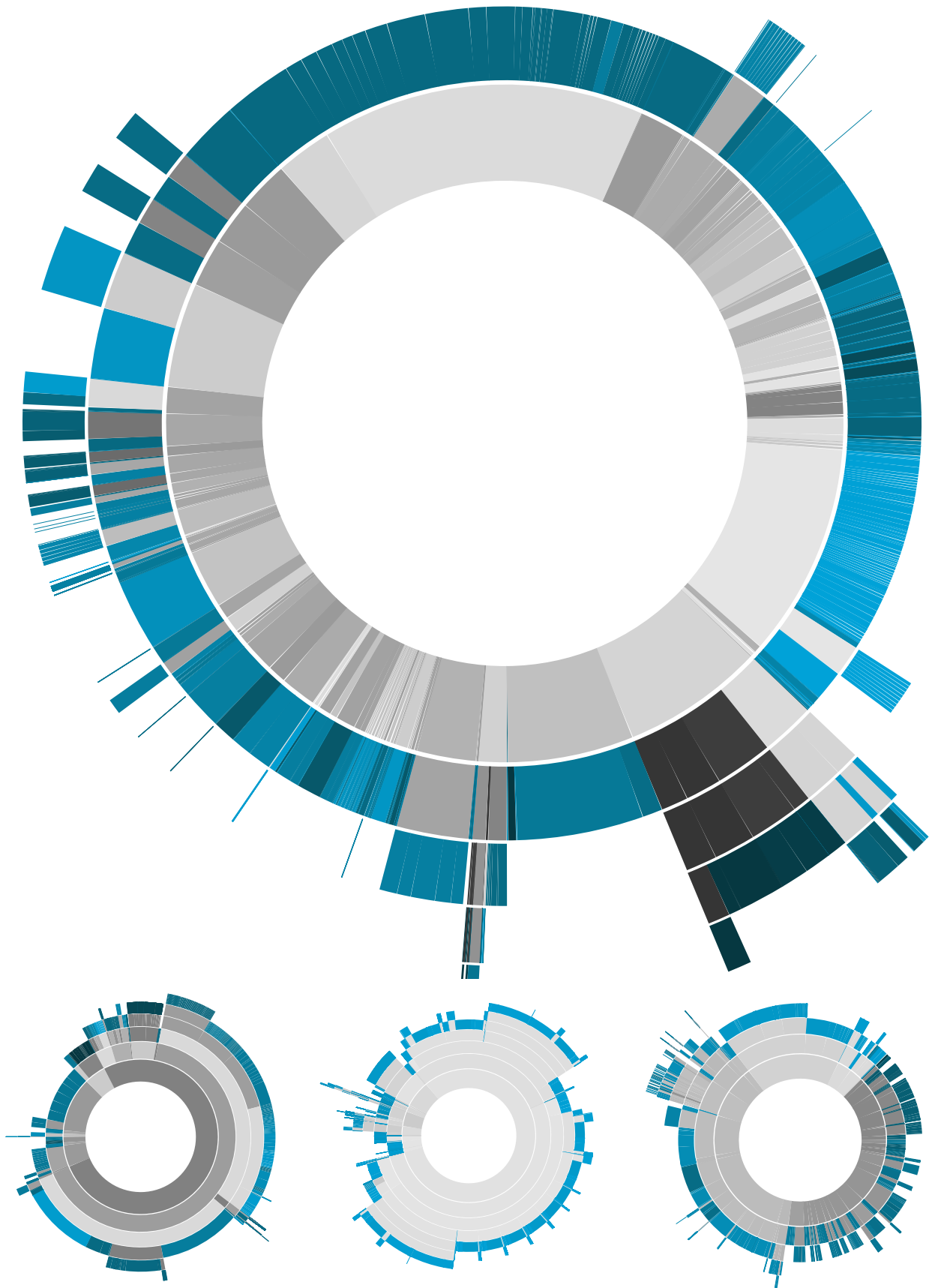




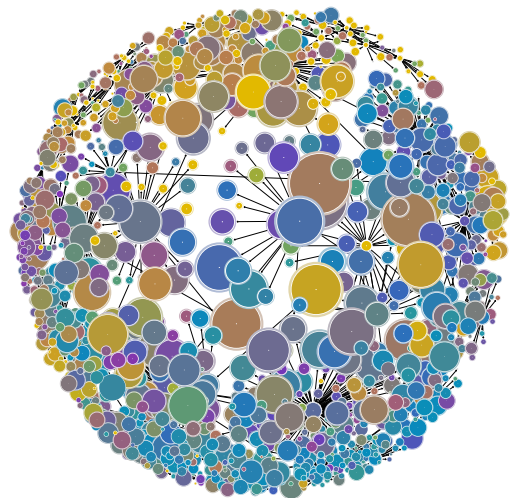
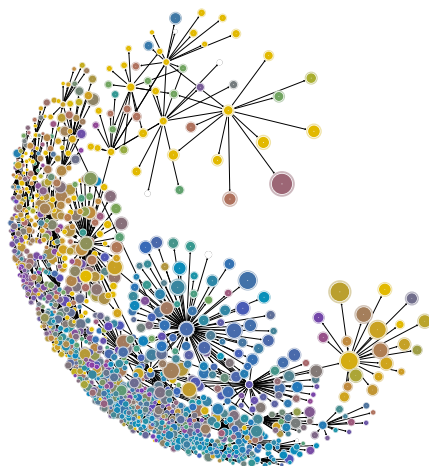
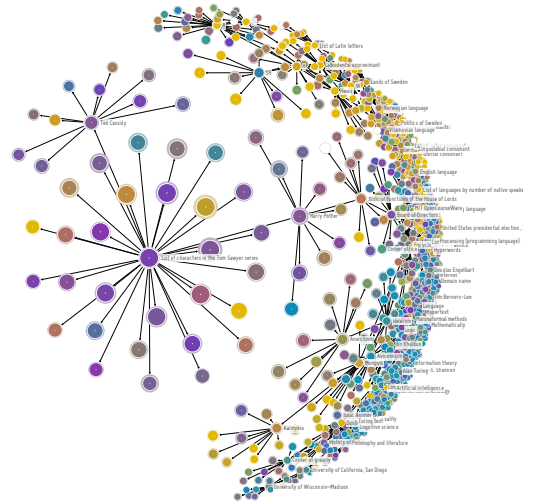
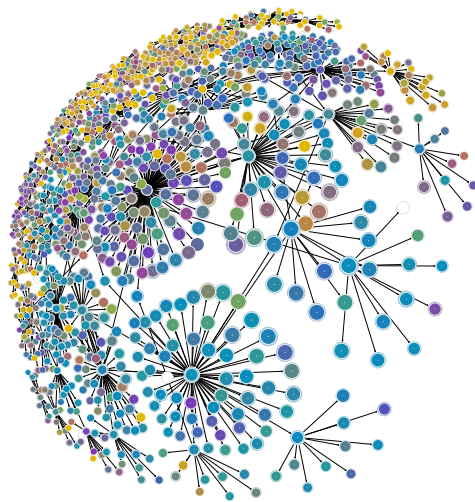
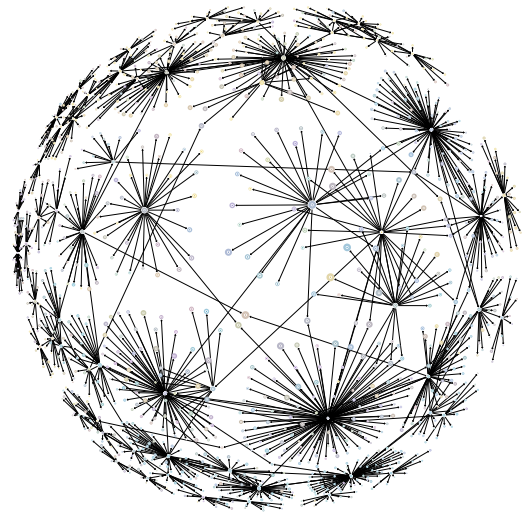
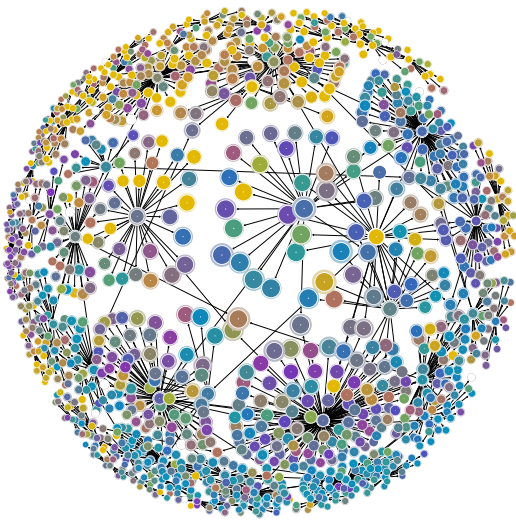


→ M_4_3_01 The points of regularly extending lines were gradually deformed by attractors, programmed virtual magnets that attract and repel.

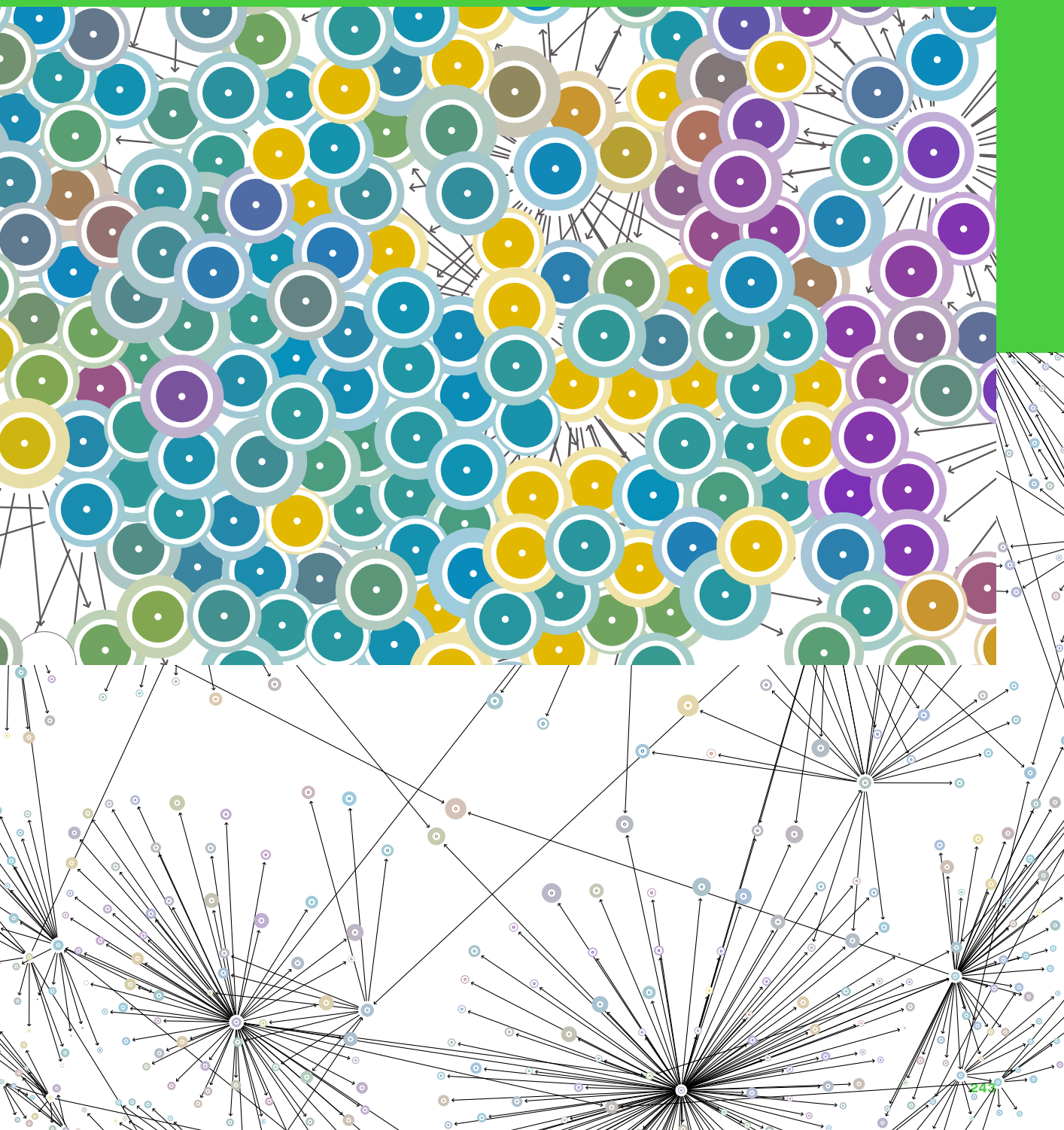




→ M_4_3_01 A data visualization of folders and files presented in the form of a sunburst diagram. The darker the circle segments, the longer the files in the folders have remained unchanged. Any folder and file structures can be loaded and visualized from a hard disk.



→ **M_5_4_01** Data visualization of linking structures of Wikipedia articles. The size of each circle represents the length of an article, while the color signifies its theme. By experimenting with the sample programs, it is possible to gain a sense of what generative design is all about. On the following pages, we reflect on the ideas behind the sample programs and put what has been learned into context. This part also has a referential character—it identifies related topics and connections.



A.2 Reflection

This book shows how imagery can be generated through code. By experimenting with the sample programs, you can gain a sense of what generative design is all about. On the following pages, we reflect on the ideas behind the sample programs and put what we have learned into context. This section also identifies related topics and connections.

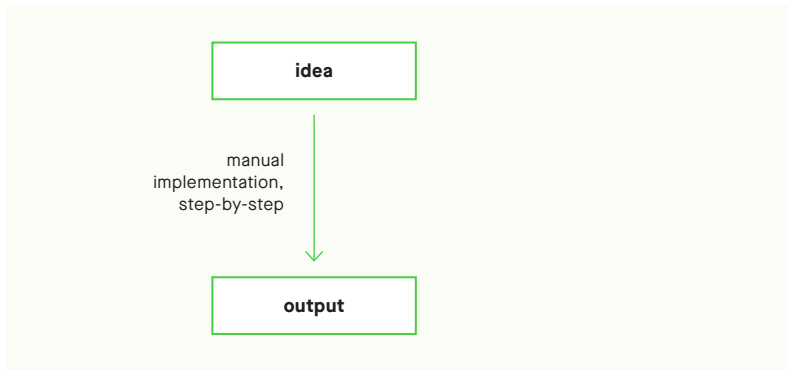
Status Quo Digitization has become an integral part of the discipline of design, and all of us use computers for our daily work. The best-known tools in the design sector, however, such as the Adobe Creative Cloud products (Photoshop, Illustrator, et cetera) have only an illustrative and imaging function. Existing tools, such as brushes, scissors, or photo labs, can be made virtual and more efficient, allowing us to get results faster and work more comfortably. Yet the design process has not evolved with these tools. ¹ Whether an image has been produced with a mouse or a paintbrush, the concept of creating a line, for instance, remains the same. Generative design is different from conventional methods; the design process is unique and by its nature results in new possibilities.

¹ “Painting with a mouse on the computer screen has a high entertainment value, but [...] drawing a stroke with a pen is no different from drawing a stroke with a mouse. The real challenge is to discover the intrinsic properties of the new medium and to find out how the stroke you draw via computation is one you could never draw, or even imagine, without computation.”

→ **John Maeda**
Design by Numbers

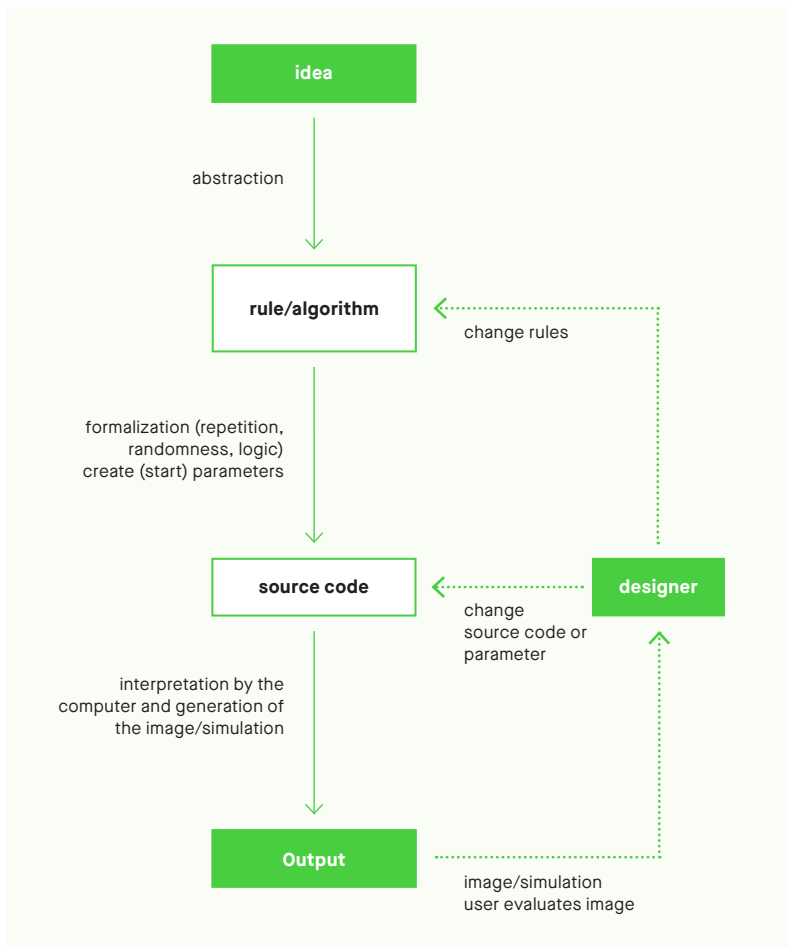
The New Design Process The main change in the design process achieved by using generative design is that traditional craftsmanship recedes into the background, and abstraction and information become the new principal elements. The relevant question is no longer “How do I draw?” but rather “How do I abstract?” This is because the process that leads from the idea to the final image can only take place using an algorithm—a sequence of rules—that the computer interprets and processes. Before appearing in the display, each generated image must first be described completely using a set of rules. This poses two challenges for the designer: how to abstract a vague idea, and how to enter an idea into the computer in a formalized way. No set rules exist for how to abstract an idea; for a complex idea to be implemented, the problem must be broken down into smaller chunks. This problem-solving strategy is also known as

Design process: analog and digital



The idea for an output is manually implemented, step by step, in a “visual flight rule” fashion. Designers can evaluate and intervene directly in each step as they draw on paper, create keyframes for animation, et cetera. If the idea is implemented digitally, tools like brushes, scissors, or photo labs become virtual and therefore more efficient. Although this makes everything faster and easier, the design process remains the same.

Design process: generative



Example:

Structural density from agents

Idea:

Generate a density structure of circles that are similar in structure to foam bubbles.

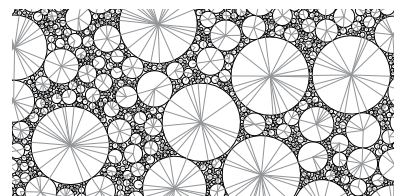
Establish abstracted rules:

1. Generate a new circle.
2. If this does not intersect with any other circle in the display, make it as large as possible.
3. If an intersection exists, start over.

Formalize in code syntax:

```

for(var i=0; i < currentCount; i++) {
  var d = dist(newX,newY,x[i],y[i]);
  if (newRadius > d-r[i]) {
    newRadius = d-r[i];
    closestIndex[currentCount] = i;
  }
}
  
```



P.2.2.5

Structural density from agents.

“divide and conquer.” [2](#) For instance, a surface is to be filled with as many circles with random diameters as possible without any overlaps. The first step is to convert the vague idea into a concrete and simple “recipe.” Draw a new circle. If this circle does not intersect with any other on the display, make this circle as large as possible. If the circle intersects another, start over. [→ P.2.2.5](#) Only through this decomposition is it possible to formulate the individual steps in a programming language, thereby making it executable for the computer. A programming language offers the elementary building blocks with which to do this, such as *repetition*, *randomness*, and *logic*, which will be explained more precisely in the following pages.

Repetition allows us to let the computer work on a task until it is solved and gives us the ability to manipulate a huge number of objects. The importance of repetition is reflected by how often the for-loop is used in the programs in this book or by the fact that the draw-loop is the central function of p5.js.

Randomness is used to create variations to break up the strict regularity of the computer. True randomness rarely produces compositionally interesting results. These are created instead when randomness is limited and applied in measured doses. In p5.js randomness is represented by the keywords “random” and “noise.” [→ P.3.2.5](#)

We use **logic** as a kind of control structure to steer the generative process. Here we can set up conditions for directing the program flow into different branches. For example, the chapter “Text as a Blueprint” [→ P.3.1.2](#) contains another switch structure we can use to execute different parts of the program depending on the entered text: each letter is written out normally, but every time a punctuation mark appears, the writing direction changes and the punctuation mark is replaced by a curved element. The most common keywords for control structures are “if,” “else,” “switch,” and “case.”

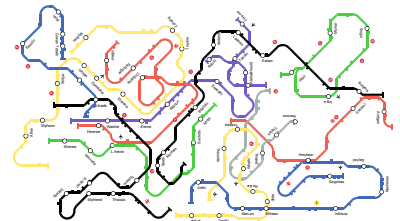
Once a design idea has been translated into code that can be interpreted by the computer, it is possible to generate many image possibilities, without the hand ever drawing a line. However, the first results are almost never 100 percent satisfactory. These results must be evaluated; this evaluation serves as the basis for improving the next iteration. In contrast to the conventional approach, we do not directly put our hands on the image but instead change the

[2](#) “In addition to ‘divide and conquer,’ there are many other problem-solving strategies, such as top-down and bottom-up, et cetera. In this context, the ‘pattern’ idea (analyzing problems in recurring patterns and solving these systematically) can be very helpful.”

[→](#) **Christopher Alexander**
A Pattern Language

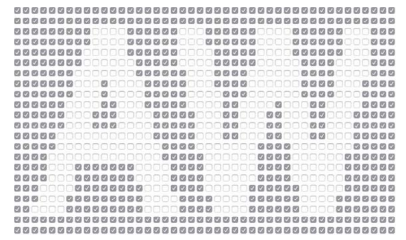


[→ P_3_2_5_01 bis → P_3_2_5_0](#)
Text in motion



[→ P.3.1.2](#)
Text as a blueprint

underlying abstraction or individual parameters in the program and continue to refine each iteration until the desired end result is achieved. Interaction helps accelerate this reciprocal effect. A generative system in and of itself is not necessarily interactive, and this book does not deal explicitly with interaction. The extra effort involved in installing control elements (buttons, sliders, et cetera) is worth it, however. These allow us to track and control every parameter change in real time. Since all the programs in this book run in the web browser, it is very easy to enhance them with standard HTML elements such as buttons, sliders, et cetera. [→ P.2.1.4](#)

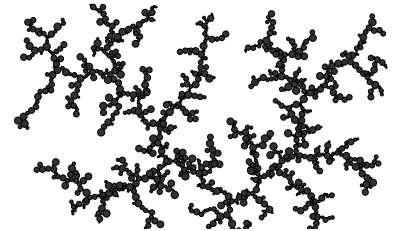


[→ P.2.1.4](#)

Checkboxes in a grid

New Possibilities in Design The integration of programming into the design process greatly increases the possibilities for designers. Conceptual competence still lies with the designer; the computer assumes only the role of the tireless helper. It is a given in today's automated and digital world that designs can be executed quickly and easily and that it is possible to generate a composition with thousands of elements. But it is more interesting to look at the new possibilities of generative design in terms of *emergence*, *simulation*, and *tools*.

In the context of generative design, processes are emergent when their results are not predetermined and when the interaction of all the elements leads to more than is obvious from their individual properties. A common example of *emergence* is the flocking behavior of birds: simple rules result in highly complex, unpredictable behavior. In the section "Growth structure from agents," [→ P.2.2.4](#) a very simple algorithm that consists of only two steps results in completely unexpected organic structures. (Boids and automata are other examples.)



[→ P.2.2.4](#)

Growth structure from agents

The **simulation** of natural processes is another important method in generative design. For example, in the chapter "Agents on a pendulum," [→ P.2.2.6](#) a structure of several connected pendulums is created. The movement of a single pendulum is simple: it circles its point of articulation. However, if several of them are linked to form a chain, then the outermost ones move in a complex manner. And, as is so often the case, the transfer of knowledge between fields leads to amazing results; certainly there are many more models in nature that could potentially be transferred to a generative system.

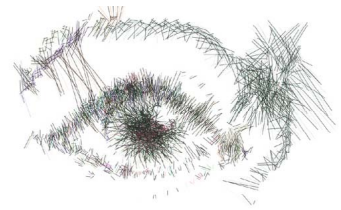


[→ P.2.2.6](#)

Agents on the pendulum

Perhaps the most important aspect of this increase in possibilities is that the designer is now the creator of individualized tools, since each generative program is also a customized software tool. This allows the designer to explore new paths that would not have been available using existing software, leading to a wider range of visual design mediums. It is astonishing how much designers can accomplish with tools they develop themselves, as is evident in “Drawing.”

→ P.2.3 Even these simple tools greatly increase a designer’s options, and, because such tools can be refined constantly, they become perfectly tailored work instruments. In addition, it is usually not a huge leap from a generative program to developing one’s own app.



→ P.2.3

Drawing

Looking ahead The world of generative design has significantly evolved since the first edition of this book. Generative design—also referred to as creative coding—has established itself in the design world. Working with generative design has become common practice, as the basis of data visualizations, artwork, media installations, architectural models, video clips, fonts, dynamic appearances—all the way to customized mass production. A look into the future clearly shows that the use of generative design will continue to increase. This is indicated by the following favorable factors:

The amount of information with which we are confronted daily is rapidly increasing thanks to ubiquitously accessible computer networks. Helping society deal with this flood of information through data visualization is an important task for designers. 3 Generative design will be indispensable in coping with this challenge.

The expansion of technological possibilities is a constant source of stimulation for generative design. Just a few years ago, for example, it was nearly impossible to generate complex, 3D worlds or to make them tangible with virtual reality and augmented reality—something that is now possible even on a smartphone. This technical potential will continue to be a driving force in generative design.

Equally important is the community of collaborators, which in almost no other design field is so vital. It is astounding how much the internet-based communities that have arisen around Processing, p5.js, vvvv, openFrameworks, NodeBox, or Basil.js have contributed to libraries, tutorials, sample programs, articles, forum contributions, wikis, et cetera. This is certainly in part because creating code—generative design’s underlying design medium—is well suited to teamwork, as software development has demonstrated. One advantage of code is that it can easily be exchanged and spread, in contrast to other media such as video.

3 “The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that’s going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids. Because now we really do have essentially free and ubiquitous data. So the complementary scarce factor is the ability to understand that data and extract value from it.”



Hal Varian

Professor at the University of California, Berkeley

Nevertheless, a designer who has programming skills is still the exception rather than the rule today. This has historical and cultural reasons: generally designers have been forced to decide to be either artists or technicians. To be both in perfect union is rarely an option, for example, in university curricula, although this distinction has begun to blur in recent years. In addition, learning this new design process—using mathematical-analytical source code to transform an idea into a visual result—feels to many like an insurmountable obstacle. Overcoming this obstacle is one of this book's main objectives.

An understanding of generative design will lead to a new matter-of-factness in taking advantage of the computer's potential. Programming, and thus generative design, is rapidly becoming a universal cultural asset and technical tool, just as photography and film were in the last century. The possibilities are already there; we just need to use them.



**Georg Trogemann,
Jochen Viehoff**
CodeArt, foreword

General

- **Armstrong, Helen.** *Digital Design Theory: Readings from the Field*. New York: Princeton Architectural Press, 2016.
Collection of new and old essays on generative design and related fields.
- **Flake, Gary William.** *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. Cambridge, MA: MIT Press, 1998.
Generative design looked at from a (very) mathematical viewpoint.
- **Reas, Casey, Chandler McWilliams, and Jeroen Barendse.** *Form+Code in Design, Art, and Architecture*. New York: Princeton Architectural Press, 2010.

p5.js and Processing

- **McCarthy, Lauren, Ben Fry, and Casey Reas.** *Make: Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. San Francisco: Maker Media, 2016.
- **Reas, Casey, and Ben Fry.** *Processing: A Programming Handbook for Visual Designers and Artists*, second edition. Cambridge, MA: MIT Press, 2014.
- **Shiffman, Daniel.** TheCodingTrain.com.
Great video tutorials on p5.js, processing, and generative design.
- **Shiffman, Daniel.** *The Nature of Code: Simulating Natural Systems with Processing*. Self-published, 2012.

Data graphics

- **Cairo, Alberto.** *The Truthful Art: Data, Charts, and Maps for Communication*. San Francisco: New Riders, 2016.
- **Fry, Ben.** *Visualizing Data: Exploring and Explaining Data with the Processing Environment*. Sebastopol, CA: O'Reilly Media, 2008.
- **Klanten, Robert, Sven Ehmman, Thibaud Tissot, and Nicolas Bourquin.** *Data Flow 2: Visualizing Information in Graphic Design*. Berlin: Gestalten, 2010.
Illustrated book covering advanced data graphic programming with Processing and presenting a wide range of data graphics projects
- **Lupi, Giorgia, and Stefanie Posavec.** *Dear Data*. New York: Princeton Architectural Press, 2016.
- **Munzner, Tamara.** *Visualization Analysis and Design*. Boca Raton, FL: CRC Press, 2014.
- **Tufte, Edward R.** *Envisioning Information*. Cheshire, CT: Graphics Press, 1990.
- **Tufte, Edward R.** *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press, 2001.

Aesthetics

- **Gerstner, Karl.** *Designing Programmes*. Zurich: Lars Müller, 2007. Originally published in 1964; its title is somewhat misleading in our computer age—if it were published for the first time today, it would be called something like *Inventing Design Rules*.
- **Kandinsky, Wassily.** *Point and Line to Plane*. Edited by Hilla Rebay. Translated by Howard Dearstyne and Hilla Rebay. New York: Dover Publications, 1979. Originally published in 1925; theoretical musings on abstract painting by the Bauhaus master.
- **Leborg, Christian.** *Visual Grammar*. Translated by Diane Oatley. New York: Princeton Architectural Press, 2006. Visual syntax abstract in overview; very clear and well illustrated Studio Moniker. *Conditional Design Workbook*. Amsterdam: Valiz, 2013.

History and context

- **Brown, Paul, Charlie Gere, Nicholas Lambert, and Catherine Mason, eds.** *White Heat Cold Logic: British Computer Art, 1960–1980*. Cambridge, MA: MIT Press, 2008.
- **Burnham, Jack.** *Software: Information Technology; Its New Meaning for Art*. New York: Jewish Museum, 1970.
- **Leavitt, Ruth, ed.** *Artist and Computer*. New York: Harmony Books, 1976.
- **Lee, Pamela M.** *Chronophobia: On Time in the Art of the 1960s*. Cambridge, MA: MIT Press, 2004.
- **Maeda, John.** *Design by Numbers*. Cambridge, MA: MIT Press, 2001. First popular book that showed the design world how it was done, using the programming language DBN.
- **Maeda, John.** *Maeda@Media*. New York: Rizzoli, 2000. Autobiography and retrospective of John Maeda's major contribution to generative design.
- **McCollough, Malcolm.** *Abstracting Craft: The Practiced Digital Hand*. Cambridge, MA: MIT Press, 1998.
- **Paul, Christiane.** *Digital Art*. New York: Thames & Hudson, 2003.
- **Reichardt, Jasia.** *Cybernetic Serendipity: The Computer and the Arts*. New York: Frederick A. Praeger, 1968. Catalog of the first major exhibition on computer art in London.
- **Rosen, Margit, ed.** *A Little-Known Story about a Movement, a Magazine, and the Computer's Arrival in Art: New Tendencies and Bit International, 1961–1973*. Cambridge, MA: MIT Press, 2011.
- **Shanken, Edward A.** *Art and Electronic Media*. New York: Phaidon Press, 2009.
- **Wardrip-Fruin, Noah, and Nick Montfort, eds.** *The New Media Reader*. Cambridge, MA: MIT Press, 2003.
- **Whitelaw, Mitchell.** *Metacreation: Art and Artificial Life*. Cambridge, MA: MIT Press, 2004.
- **Wilson, Mark.** *Drawing with Computers: The Artist's Guide to Computer Graphics*. New York: Perigee Books, 1985.
- **Wood, Debora.** *Imaging by Numbers: A Historical View of the Computer Print*. Chicago: Northwestern University Press, 2008.

- **Benedikt Groß** Born in 1980 in Baden-Württemberg. 2002: studied geography and computer science. Discontinued those studies and became interested in design. 2007: completed master's degree in generative systems with Julia Laub at the HfG Schwäbisch Gmünd. Thereafter, employed in higher education and in the field of design. 2011–13: master's student in design interactions at the Royal College of Art London under Anthony Dunne and Fiona Raby; at the same time, data visualization specialist at the MIT Senseable City Lab in Cambridge, Massachusetts / Singapore. Since 2013: freelance work in speculative and computational design; various international exhibitions, awards, and publications. Since 2017: professor of interaction design at the HfG Schwäbisch Gmünd.
- **Hartmut Bohnacker** Born in 1972 in Baden-Württemberg. Beginning in 1992: studied mathematics (discontinued) and earned a degree in economics in 1997. 1998: studied communication design at the HfG Schwäbisch Gmünd. Since completing studies in 2002: independent designer in Stuttgart, specializing in conception, design, and prototypical implementation of projects in the field of interface and interaction development; teacher of digital media. Since 2009: professor of interaction design at the HfG Schwäbisch Gmünd.
- **Julia Laub** Born in 1980 in Bavaria. 2003: studied communication design at the HfG Schwäbisch Gmünd. Studied abroad at the HGK Basel. 2007: completed master's thesis, "Generative Systems," with Benedikt Groß. Since 2008: independent graphic designer specializing in book design, corporate design, and generative design. 2010: cofounder of the design studio onformative (studio for digital art and design) in Berlin with Cedric Kiefer. Teacher and leader of workshops at various universities.
- **Claudius Lazzeroni** Born in 1965 in Bavaria. 1984: trained to become a photographer with Raoul Manuel Schnell. 1987: tutor at Massachusetts College of Art in Boston. 1992: earned degree in media design at the BILDO Academy in Berlin. Until 1996: creative director at Pixelpark. Until 2001: founder, director, and creative director of the design agency IM STALL in Berlin. Since 1999: professor of interface design at the Folkwang Kunsthochschule in Essen. Since 2005: exploration, development, and construction of solographs. Since 2007: expansion of academic department to include physical computing.
- **Niels Poldervaart** Born in 1994 in the Netherlands. 2016: earned bachelor's degree in communication and multimedia design at the Avans University of Applied Sciences's-Hertogenbosch. Since 2016: web developer and designer, with a focus on data visualization, e-learning, gaming, and graphic design.
- **Joey Lee** Born in 1990 in California. 2012: earned bachelor's degree in geography from the University of California, Los Angeles, and master's degree in geography from the University of British Columbia, Vancouver. International activities in research and teaching at institutions such as MIT Senseable City Lab in 2013 and at Mozilla Science Lab in 2015. Various international exhibitions, prizes, and publications in the fields of new spatial and digital media, climate research in urban areas, and open source.

- Lauren McCarthy, the initiator of p5.js.
- All those involved at Gold & Wirtschaftswunder: Julia Kühne, Christian Schiller, Steffen Knöll, Christian Nicolaus, Andreas Lörinc.
- Everyone at the Hermann Schmidt Mainz publishing house (especially Brigitte Raab).
- Sabrina Groß, for her text feedback and advice.
- All the contributors who made small code contributions and all the developers of the JavaScript libraries we used in the book (see readme file in the code package).
- The team of twemoji (github.com/twitter/twemoji).
- Frederik De Bleser, for his prompt answers to unusual questions about Opentype.js and g.js.

Published by
Princeton Architectural Press
A McEvoy Group company
202 Warren Street
Hudson, New York 12534
www.papress.com

Princeton Architectural Press is a leading publisher in architecture, design, photography, landscape, and visual culture. We create fine books and stationery of unsurpassed quality and production values. With more than one thousand titles published, we find design everywhere and in the most unlikely places.

Originally published by Verlag Hermann Schmidt
© 2018 Verlag Hermann Schmidt and the authors
All rights reserved
English translation © 2018 Princeton Architectural Press

No part of this book may be used or reproduced in any manner without written permission from the publisher, except in the context of reviews.

Every reasonable attempt has been made to identify owners of copyright. Errors or omissions will be corrected in subsequent editions.

Special thanks to: Paula Bayer, Janet Behning, Abby Bussel, Benjamin English, Jan Cigliano Hartman, Susan Hershberg, Kristen Hewitt, Lia Hunt, Valerie Kamen, Jennifer Lippert, Sara McKay, Parker Menzimer, Eliana Miller, Nina Pick, Wes Seeley, Rob Shaeffer, Marisa Tesoro, Paul Wagner, and Joseph Weston of Princeton Architectural Press
—Kevin C. Lippert, publisher

Library of Congress Cataloging-in-Publication Data

Names: Bohnacker, Hartmut, 1972– author. | Gross, Benedikt, 1980– author. | Laub, Julia, 1980– author. | Lazzeroni, Claudius, author. | Frohling, Marie, translator.

Title: Generative design : visualize, program, and create with JavaScript in p5.js / Benedikt Gross, Hartmut Bohnacker, Julia Laub, Claudius Lazzeroni ; with contributions by Joey Lee and Niels Poldervaart ; translated by Marie Frohling.

Other titles: Generative Gestaltung. English Description: New York : Princeton Architectural Press, [2018] | Translation of Generative Gestaltung by Hartmut Bohnacker, Benedikt Gross, and Julia Laub and edited by Claudius Lazzeroni | Includes bibliographical references.

Identifiers: LCCN 2018007216 | ISBN 9781616897581 (pbk. : alk. paper) | ISBN 978-1-61689-784-0 (epub, mobi)

Subjects: LCSH: Computer art. | Computer drawing. | Graphic arts—Data processing. | Generative programming (Computer science) | Computer-aided design. | JavaScript (Computer program language)

Classification: LCC N7433.8 .B6313 2018 | DDC 776—dc23

LC record available at
<https://lccn.loc.gov/2018007216>

Concept

Benedikt Groß, Hartmut Bohnacker, Julia Laub, Claudius Lazzeroni

Text

Hartmut Bohnacker, Benedikt Groß, Claudius Lazzeroni

Programming

Benedikt Groß, Hartmut Bohnacker, Niels Poldervaart, Joey Lee

All code in the book has been created using p5.js, version 0.5.11, and is published under the Apache license.

Illustrations and photography

Hartmut Bohnacker, Benedikt Groß,
Julia Laub, Claudius Lazzeroni, Jana-Lina
Berkenbusch, Andrea von Danwitz, Pau
Domingo, Stefan Eigner, Victor Juarez
Hernandez, Cedric Kiefer, Steffen Knöll,
Andreas Lörinc, Franz Stämmele, Tom Ziora

Book design

Gold & Wirtschaftswunder Stuttgart,
www.gww-design.de
Julia Kühne & Christian Schiller (art direction),
Steffen Knöll, Christian Nicolaus

Layout and typesetting

Julia Kühne, Steffen Knöll, Christian Nicolaus,
Hartmut Bohnacker, Flore de Crombrugghe,
Maximilian Semmler

For Princeton Architectural Press

Translator: Marie Frohling
Project editor: Sara Stemen
Typesetting: Tina Henderson

Typefaces

Maison Neue Book, Maison Neue Bold,
Maison Neue Mono
For programming code: FreeSans, Fira Sans,
Miso, Noto Sans

Website

www.generative-gestaltung.de
Concept and design: Gold &
Wirtschaftswunder, Benedikt Groß
Programming and implementation: Niels
Poldervaart, Joey Lee, Benedikt Groß

Go on coding!

www.generative-gestaltung.de

Bring ideas to life through creative coding!

Designers, artists, and makers increasingly use generative design as their go-to technique for producing artwork, models, animations, illustrations, and films. Simple programming languages such as JavaScript in p5.js jump-start the creative process for everything from interactive typography and textiles to 3D-printed furniture to complex and elegant infographics.

Generative Design presents a gallery of visual inspiration from international creators, followed by a handbook of easy-to-follow, step-by-step tutorials demonstrating how to work with color, form, typography, and images. Instructions are cross-referenced to a companion website where users can download ready-to-adapt programs and share code, tips, and designs. *Generative Design* is a stunning showcase and accessible reference guide to this dynamic visual approach.

Praise for *Generative Design: Visualize, Program, and Create with Processing*

“Accessible, well written, easy to use, beautiful to look at, and guaranteed to jump start the mind of anyone who spends time (or would like to spend time) writing code to create art and design.”

— **Jonathon Russell,**
The Designer's Review of Books

“Generative design is helping us find new ways to express information with novel metaphors. This book, equal parts art and textbook, is a valuable tool for both learning what exists and triggering new ideas.”

— **Steven Heller,**
The New York Times

“At the same time *Generative Design* introduces readers to the jaw-dropping output created by some of the most adventurous practitioners... it is also a thorough step-by-step guidebook to learning the language, complete with code snippets and full-color samples.... *Generative Design* provides a powerful new design vocabulary.”

— **Sam McMillan,**
Communication Arts