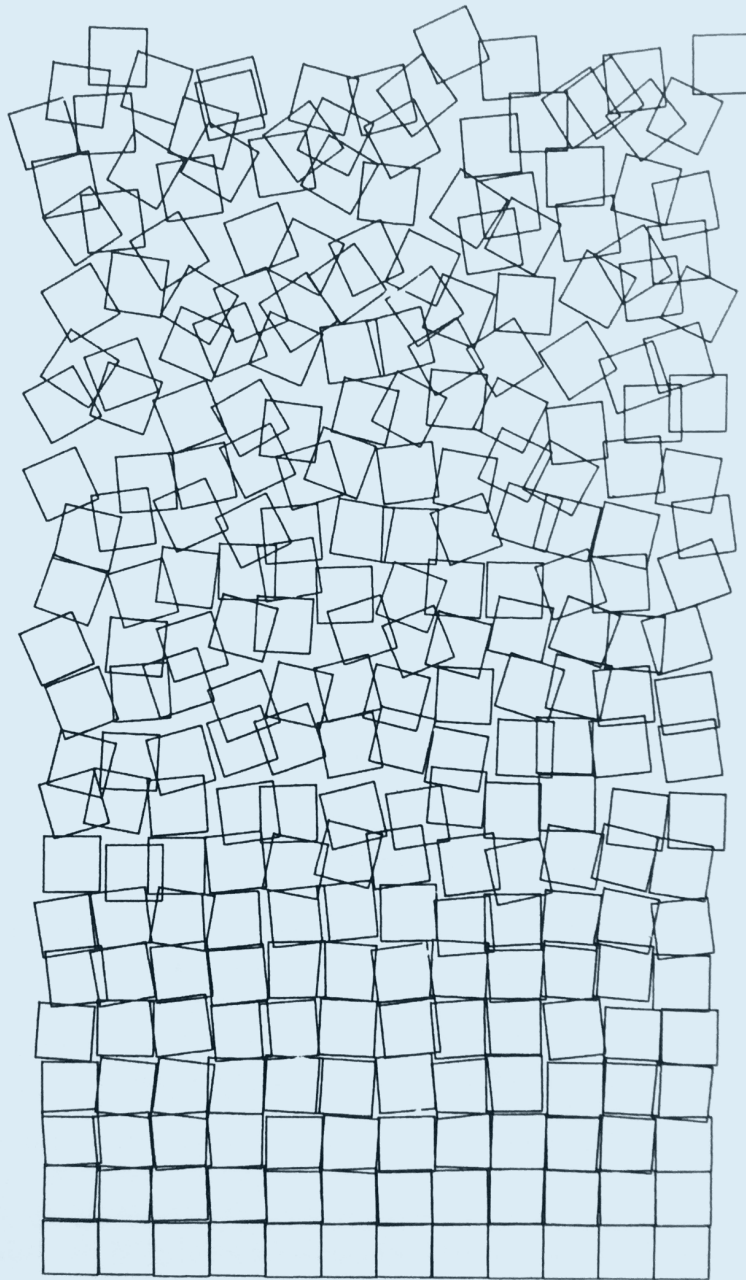


Gravel

60 Years of Computer Art



Joachim Wedekind

JOACHIM WEDEKIND

Gravel

60 Years of Computer Art

Tübingen 2025

Joachim Wedekind:
Gravel - 60 Years of Computer Art

Author:
Dr. Joachim Wedekind
Eschenweg 26
72076 Tübingen

Cover image: Schotter (Georg Nees, 1965-1969)

2025 Tübingen

© Copyright for the complete work lies with the author

Graphic design and typesetting: Joachim Wedekind

Imprint: <http://joachim-wedekind.de/impressum/>

This work by Joachim Wedekind is under a Creative Commons License 4.0: Attribution, Non-commercial, No Derivative Works. Permissions beyond this license can be requested from the author.



Table of contents

Preface.....	1
Schotter: The picture	3
Georg Nees: The pioneer	5
Schotter: The pictures	11
Recoding Schotter	14
Remixing Schotter	26
Schotter: interactive & animated	47
Think the Image, Don't Make It!.....	55
Literature	56
Image Credits.....	57
My Schotter-Programs	58
About the person	59

Preface

This short essay is primarily dedicated to a single image, the plotter drawing *Schotter* (1965-1969; English: *Gravel*) by Georg Nees, one of the pioneers of Computer Art. This image has become an icon of Algorithmic and Generative Art. It has often been the subject of *Recoding*, it is a popular example in courses on Digital Art and it was and it is a stimulus for *Remixing* with many pictorial variations.

Over the years, I have found many examples of the *Recoding* and *Remixing* of *Schotter* that have confirmed to me what a great influence the preliminary work of Georg Nees and the other pioneers of early Computer Art had and still have today on the development of Algorithmic and Generative art and thus ultimately on the current forms of Digital Art.

It is now sixty years since the world's first exhibition of Computer Art with works by Georg Nees took place in the rooms of the Studiengalerie of the TH Stuttgart. I have not found any references in the established art world, let alone events commemorating this significant event. After all, Computer Art is the recognized forerunner of today's Digital Art.

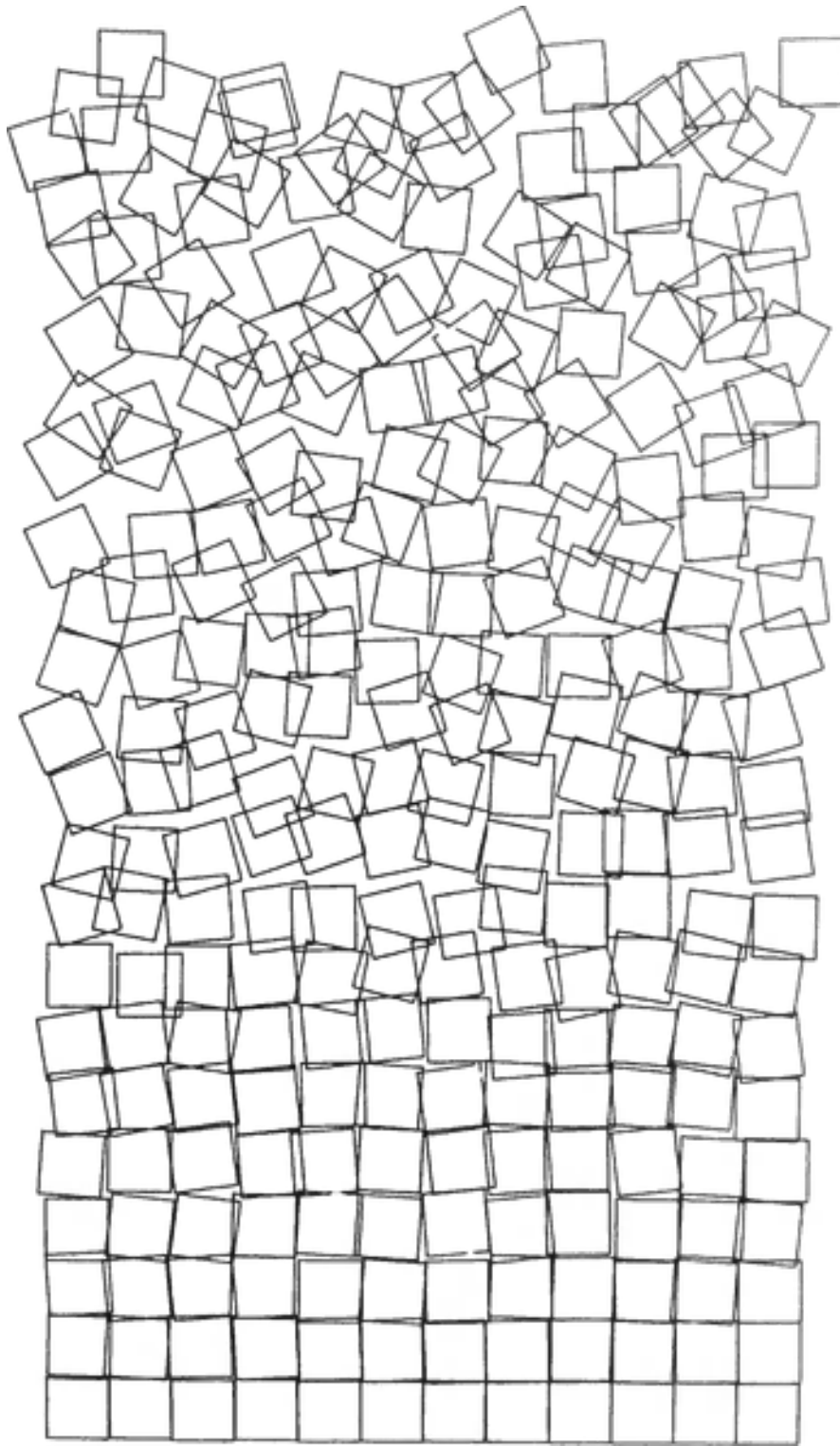
This publication is a summary of my repeated preoccupation with the image *Schotter*. It appears 60 years after the public perception of early Computer Art in the Stuttgart exhibition and is intended as a tribute to Georg Nees and his great contribution to early Computer Art.

Tübingen, April 2025

Joachim Wedekind

Note on the english version of this essay: I apologize for the poor linguistic quality of this booklet. It is my own translation of the German original. In most sections I used the help of [DeepL](#). Corrections and improvements are welcome.

Schotter: The picture



Georg Nees: Schotter (1965-1968, Plotter drawing, ink on paper, 29.7 x 21 cm). Kunsthalle Bremen - Kupferstichkabinett; from Groos & Froitzheim, 2016.

Georg Nees: The pioneer

Georg Nees (1926 - 2016) was a German engineer and mathematician. Born in Nuremberg in 1926, he studied mathematics, physics and philosophy in Erlangen and Stuttgart.



*Georg Nees, 1986
(Photo: A. Kempkens)*

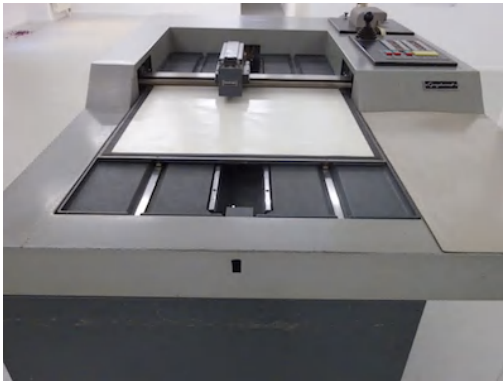
Nees began programming in 1959. From 1965, he worked at Siemens in Erlangen, where he began programming in the ALGOL language. This also involved computer graphics for architectural projects, for example.

While working at Siemens, Nees began to focus more intensively on programming graphics. Around 1964, he began to write his first algorithms in the ALGOL language, which he used to print his graphics on paper with the Zuse Graphomat Z64 plotter. He created patterns from randomly generated numbers that were output on paper using the plotter.

In order to create these types of graphics, he first had to develop his own graphics libraries, which added random functions and plotter control functions to the ALGOL language. At the time, these libraries were a necessary prerequisite for his artistic work. The same applied to the required random number generators.

The development of an image like *Schotter* was a real pioneering achievement for the time. As there were no monitors on which his projects could have been visualized, Nees had to rely on his imagination to see what result could be achieved with his algorithms.

Nees programmed his early graphics on Siemens systems, such as the System 4004, a mainframe computer that Siemens produced itself from 1968. Nees used plotters to print the graphics, which were mainly used in technical and scientific fields at the time; specifically the *Graphomat Z 64* from Zuse:



*ZUSE Graphomat Z 64, drawing table (left); control electronics cabinet (bottom).
Photos: IW, 2015*



At the Hannover Fair in 1961, computer pioneer Konrad Zuse presented the Graphomat Z64 drawing table, a flatbed plotter that was controlled by punched cards. It was characterized by high drawing accuracy and could draw dots, curves and symbols with up to four colors. Thanks to high-precision planetary gears, the device could be controlled at 16 different speeds. The series devices were available for the output formats 550 x 600mm and 1200 x 1400mm. They weighed approx. 1400 kg; the pictures above can only give a rough impression of the size and mass of the Graphomat Z64.

Basically, the plotters were used to create images in an almost conventional way: The plotters move a pen across the surface of a sheet of paper. The results are vector graphics, not raster graphics as we know them from our modern screens. Plotters do not work line by line like printers, but trace the lines of vector graphics. Vector graphics can be scaled as required without any loss of quality and therefore avoid the technical limitations of raster graphics in terms of image resolution and image size. They are therefore particularly suitable for large and complex graphics.

With the pen plotters, a pen holder is moved on two axes. The pen can also be raised and lowered. A single colored pen or several pens of different colors attached to a holder are moved over the paper by motors and draw the lines. Filled areas can only be achieved by narrow hatching. A different pen must be used for each color and for each line thickness. Due to the mechanical movement of the pens, the creation of an image is very slow. For a long time, however, this was the only way to create vector-based images, even in large formats.

Felt-tip and ink pens as well as ballpoint pens, which also allow different line widths, are suitable for drawing. In the beginning, only monochrome drawings were possible, as changing colors meant interrupting a running program and only then changing the pen. Later, automatic color changes were also possible using color carousels.

Today, plotters have largely become superfluous in their original areas of application. Their tasks are taken over by laser or inkjet printers of the appropriate size. Nevertheless, plotters are currently experiencing something of a renaissance because they now often offer extended areas of application and device combinations. There are affordable devices that can even produce image sizes up to A1 format.

For all their precision, there is something non-machine-like about plotters. As a rule, plotter images are not 1:1 images of the results of a program shown on the screen. The movement of the carriage with the drawing pens, its acceleration, deceleration, raising and lowering of the pens and the properties of the paper and ink often result in the appearance of artifacts. Basically, therefore, each picture is unique and has a similarity to conventional manual work with pen and paper.

So much for the technical aspects of Georg Nees' work, which form an important background for his role in Computer Art. In 1969, Nees completed his doctorate in philosophy on *generative computer graphics* (Nees: *Generative Computergraphik*, 1969). His doctoral supervisor was Max Bense (1910-1990), Professor of Philosophy of Technology, Philosophy of Science and Mathematical Logic at the Technical University of Stuttgart. Bense coined the term *Information Aesthetics*. Since the early 1960s, a group of doctoral students had formed in his environment, who deepened the aesthetics of information. Among them were Rul Gunzenhäuser, Frieder Nake and Helmar Frank.

In his dissertation, Nees attempts to concretize the fundamental concepts and philosophical aspects of Bense's aesthetics. He therefore deals with the subject of computer graphics from both a computer-scientific and a philosophical-aesthetic perspective. He introduces this way of thinking with concrete examples and illustrates the interplay of procedural abstraction and (controlled) chance by developing exemplary programs. His field of application is *Computer Art*¹, a typical result is the image *Schotter*².

As a student of Max Bense, Nees was given the opportunity to organize the first exhibition of his Computer Art in Stuttgart. In February 1965, the world's first exhibition of Computer Art took place at the University of Stuttgart - with pictures that Nees himself called *Aesthetic Objects*. There was a vernissage at which Bense and Nees spoke and there were reactions from the press. Some established artists were also present, such as Heinz Trokes, Anton Stankowski and Herbert W. Kapitzki. One of the artists present asked: "*But can you get your computer to paint like me?*" And Georg Nees is said to have replied: "*Of course. If you tell me how you do it.*"

The mathematician Nake, who was working on the production of artistic computer graphics at the same time as Georg Nees in Stuttgart³, finds this dialog "*brilliant!*" because it is so typical of the beginnings of digital aesthetics.

¹ H.W. Franke defines Computer Art as follows (Franke, 1971, p. 7 f.): "*Computer Art is to be understood as any aesthetic entity created on the basis of logical or numerical conversions of given data with the aid of electronic automata.*" (Translation JW) Accordingly, it is always created with the help of digitally controlled devices, i.e. with programs to generate the data to be displayed, the computers on which the programs run and the output devices with which the data is displayed as perceptible artefacts.

² In the following I will always use the German term *Schotter* when referring to the original title of a corresponding work. When it comes to the concept of creating images based on it, I will use the English term *Gravel*.

³ Both exhibited together at the Galerie Niedlich (Stuttgart) in November 1965.

According to Nake, it is wrong to attribute artistic ability to the computer. But for the first time it was possible to use random generators to implement ideas in a way that was unforeseeable for the creator of the program. For Nake, this changed the concept of the image in art - even if the works were rejected in art circles as scribbles.

Georg Nees occupies a very special place in early Computer Art. Not only was he one of the first to create and exhibit such works, he was also aware of the innovative nature of his graphics from the very beginning: *"When I saw how one figure after another jumped out from under the pen, it sent shivers down my spine. I thought: here is something that will not disappear again."* (Nees, 2005; Translation JW).

Georg Nees considered working with computers and plotters as an *aesthetic laboratory* (Nees, 1995, p. 7). The computer provides him with an experimental workbench that allows him to experiment graphically *"until we succeed in creating aesthetic objects in a targeted manner - but often enough also as the result of a happy coincidence"* (Translation JW). Nees (op. cit., p. 183) once characterized the process as *Gestalt trapping* (*Gestaltfallenstellen*): *"At some point you are rewarded with an extraordinary catch"* (Translation JW).

Nees consciously programmed his works from an aesthetic point of view and saw them as models of the artistic production process. He was therefore a pioneer of Computer Art who not only found influential graphics, but also systematically investigated the redundancies in randomly controlled images. Georg Nees also designed the first computer-generated sculpture in 1968.

Well-known Conceptual Artists such as Sol LeWitt worked with rule-based art, while Op Art artists dealt with forms that were ambiguous in terms of perceptual psychology and also created mathematically motivated works of art. For their part, the early Computer Artists wanted to unite these two approaches and express them using new technology. In this sense, Georg Nees tried to show with his works that science, mathematical calculations, algorithms and technology can be combined to create a work of art.

Conceptual Art and Computer Art in particular are often described as being closely related. However, Nake characterizes this relationship in a somewhat more differentiated way as follows (Nake, 2021, p. 64):

"In 'The idea becomes a machine that makes the art', Sol LeWitt wrote in 1967: 'The idea becomes a machine that makes the art'. Argue belatedly, but nevertheless, I would like to reply: 'Exactly this, dear Sol LeWitt, has happened some years before you wrote it."

For since around 1962, a few computer experts in the USA and West Germany had been experimenting with turning their ideas into computer programs that would put drawings on the walls of galleries, in other words making nothing other than art. [...] The machine, which was a thought for the conceptualizers, was a tangible reality for the computationalists." (Translation Nake's part JW)

If you want to understand Georg Nees' thoughts in detail, you should definitely read and study his dissertation. It was also published as the book *Generative Computergraphik* (1969)⁴. The deliberate juxtaposition of central aspects of Bense's information aesthetics with the programming of his projects and their visual results is still a worthwhile and stimulating read today.

The experience that Nees gained in programming his aesthetic objects is also summarized in a systematic introduction to the world of Computer Art. In *Formel, Farbe, Form. Computerästhetik für Medien und Design* (Nees, 1995), he uses the latest software at the time - the BASIC programming language - and a wide range of examples to provide practical access to the independent creation of interesting and appealing graphics.

His book shows "*... how anyone, professional or hobbyist, can use simple programming skills to uncover rich treasures of images, colors, designs and textures [...]. The first steps are so easy: you start with images that catch your eye. Then curiosity, closer study of the text and perhaps even a computer at hand will gradually pave the way into the enticing world of forms.*" (Nees, 1995, p. VII; Translation JW)

In fact, my personal path into Computer Art is also very accurately described by Nees. Without his suggestions, my own contributions to Computer Art (exhibitions, lectures and writings) would not exist in their present form.

⁴ A reprint with introductory texts by Herrmann and Nees was published in 2006 and is also available as an e-book.

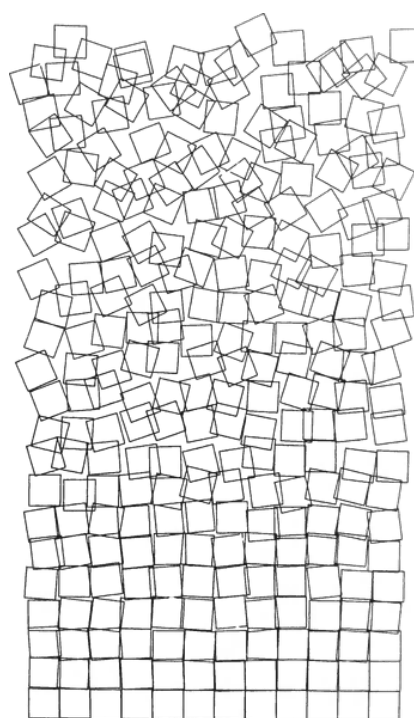
Schotter: The pictures

I first came across the image *Schotter* (as I said, referred to as *Gravel* in English) when I was looking for programming exercises for an introduction to programming with the turtle graphic⁵. I had discovered it - called *Schotterhaufen* (*Gravel Pile*) there - in an introduction to the Logo programming language, initially without knowing that it was a work by Nees⁶. In any case, it was my first independent implementation of this image.

It was only decades later, when I had begun to engage intensively and actively with Computer Art, that I found the original of the picture and the information about its creator, the engineer and mathematician Georg Nees (Wedekind, 2018, p. 149)

The starting point for the depiction in this context is the original painting of *Schotter*, as found in the dissertation by Georg Nees (Nees, 1969, p. 242). This version is also part of the collection of the Kupferstichkabinett of the Kunsthalle Bremen and was shown in 2016 in the exhibition *[un]erwartet* at the Kunstmuseum Stuttgart (Groos & Froitzheim, 2016).

However, there are probably other variants of the painting *Schotter*, each with slightly different dimensions. For example, the Victoria and Albert Museum (Kensington, UK)⁷ owns a version dated 1968-1970, while The Anne and Michael Spalter Digital Art Collection owns another version dated 1968⁸.



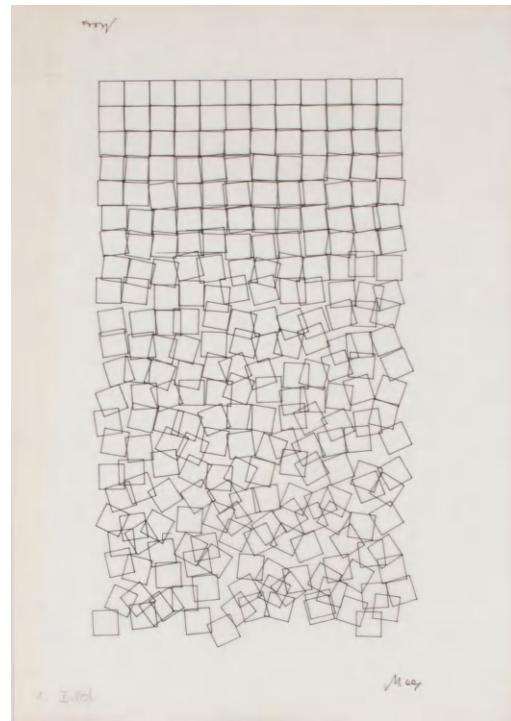
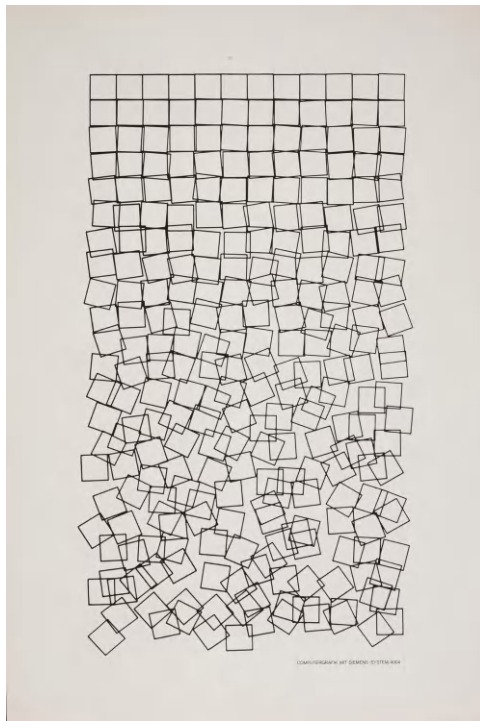
*Georg Nees: Schotter (from
Nees, 1969, image 38, p. 242)*

⁵ The *Turtle Graphic* is the graphic component of the programming language Logo. We developed and distributed a corresponding version as part of a distance learning project on *Teaching and Learning with the Computer* (Rauch & Wedekind, 1989)

⁶ Although Nees' book appears in the bibliography of this book (Stein, 1984) without mentioning the publisher or year, the origin of the 70-page reprogramming of several Nees examples is not mentioned at all in the text.

⁷ <https://collections.vam.ac.uk/item/O221321/schotter-print-nees-georg/>

⁸ <https://spalterdigital.com/artworks/shotter-gravel-stones/>



Georg Nees: *Schotter*: V&A Museum (left); *Spalter Collection* (right)

The black and white picture is made up of squares with a fixed edge length. The picture structure is basically very simple: *Schotter* consists of 22 rows of 12 squares each. In the first row, the squares lie next to each other without being rotated. From row to row, the position and orientation (rotation) of the squares are changed slightly. The degree of randomness in the rotation and the horizontal and vertical positioning of the squares gradually increases with each row. This increasingly disturbs the order of the squares. All squares are transparent, which is why the resulting overlaps always remain visible.

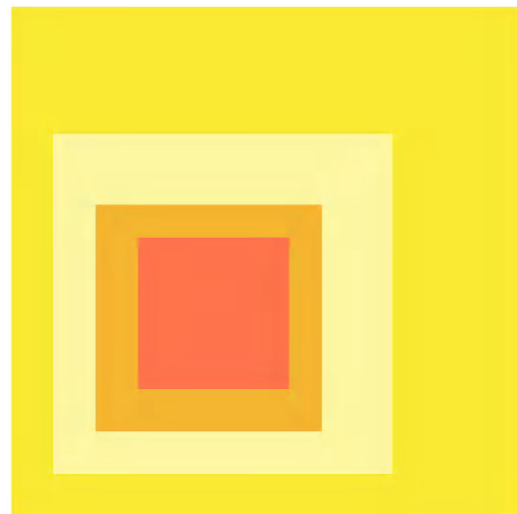
In Nees' dissertation, the disorder of the squares increases from the bottom to the top; in the collection pieces mentioned above, it is exactly the opposite. A contemporary of Nees told the anecdote according to which Georg Nees, when asked about the correct orientation of the artwork, replied that he didn't care whether the work was depicted with cascading or ascending squares. I myself prefer the cascading version, because I imagine gravel stones being tipped over by a lorry.

The picture *Schotter* is a prime example of the interplay of order and disorder through (controlled) chance. It directly visualises their interaction. Another central element is repetition, which manifests itself in the recurring sequence of similar pictorial elements - such as the squares here. Such repetitive techniques are not specific to computer work.

Frieder Nake discusses the cultural significance of such basic possibilities of computer utilisation as a tool and medium (Nake, 1995). Using concrete examples, he relates Computer Art to Pop Art, Concrete Art and Constructivism. A typical example of this is the series *Homage to the Square* by Josef Albers, which he created from 1949 onwards in countless variations, each of which represents an original. For Nake, it is perhaps the simplest example in art history that can be completely replaced by a combinatorial program. His ironic conclusion (which in my opinion probably applies to many works of modern art) is that

'[...] the art of Albers' squares does not lie in the scheme. It must lie in his particular choice of colours. Otherwise, however, this pedagogical exercise is only 'art' because art history hangs these pictures in museums and includes them in catalogues. [...] It is the social process of reception that makes a work a work of art, not the artist's creative process.' (Translation JW)

Nake's suggestion of a combinatorial program for generating nested squares ('*a scheme for initial computer science lessons*'; Nake, op. cit.) is actually extremely easy to implement⁹.



Remixing of Josef Albers: Homage to the Square, JW 2015

⁹ You can find my Remixing of the Square series by Josef Albers here: [Homage to the Square: \[Program 1\]](#)

Recoding Schotter

Georg Nees was one of the very few to document his algorithm for the image *Schotter* (Nees, 1969, p. 241 and 186). This serves as a template for *Recoding & Remixing* the image. Different procedures are possible, which I would like to discuss here. So let's start with his original program:

```

1 'BEGIN' 'COMMENT' SCHOTTER.,
2 'REAL' R, PIHALB, PI4T.,
3 'INTEGER' I.,
4 'PROCEDURE' QUAD.,
5 'BEGIN'
6 'REAL' P1, Q1, PSI., 'INTEGER' S.,
7 JE1.=5*I/264., JA1.= -JE1.,
8 JE2.=PI4T*(1+I/264.),
9 JA2.=PI4T*(1-I/264.),
9 P1.=P+5+J1., Q1.=Q+5+J1.,
10 PSI.=J2.,
10 LEER(P1+R*COS(PSI),
11 Q1+R*SIN(PSI)),
11 'FOR' S.=1 'STEP' 1 'UNTIL' 4 'DO'
12 'BEGIN' PSI.=PSI+PIHALB.,
13 LINE(P1+R*COS(PSI),Q1+R*SIN(PSI))
14 'END', I.=I+1
15 'END' QUAD.,
16 R.=5*1.4142.,
17 PIHALB.=3.14159*.5., PI4T.=PIHALB*.5.,
18 I.=0.
19 'SERIE'(10.0,10.0,22,12,QUAD)
20 'END' SCHOTTER.,

1 'REAL' P, Q, P1, Q1
2 ,XM, YM, HOR, VER
3 ,JLI, JRE, JUN, JOB
4 ,
5 'INTEGER' I, M, N, T
6 ,
7 'PROCEDURE' SERIE
8 (QUER,HOCH,XMAL,YMAL,FIGUR),
9 'VALUE' QUER, HOCH, XMAL, YMAL.,
10 'REAL' QUER, HOCH.,
11 'INTEGER' XMAL, YMAL.,
12 'PROCEDURE' FIGUR.,
13 'BEGIN'
14 'REAL' YANF.,
15 'INTEGER' COUNTX, COUNTY.,
16 P.= -QUER*XMAL*.5.,
17 Q.=YANF = -HOCH*YMAL*.5.,
18 'FOR' COUNTX.=1 'STEP' 1 'UNTIL' XMAL
19 'DO'
20 'BEGIN' Q.=YANF.,
21 'FOR' COUNTY.=1 'STEP' 1 'UNTIL' YMAL 'DO'
22 'BEGIN' FIGUR., Q.=Q+HOCH
23 'END', P.=P+QUER
24 'END',
25 LEER(-148.0,-105.0),CLOSE.,
26 SONK(11),
27 OPEN(X,Y)
28 'END' SERIE.,

```

Nees comments on the code as follows (Nees, 1965, p. 241):

'[Schotter] is generated by calling the SERIES procedure [...]. The parameterless procedure QUAD is used to generate the elementary figure, which is multiplied in the composition process controlled by SERIES. QUAD is defined in lines 4 to 15 of the generator; this procedure draws squares of constant side length but random position and angle. You can recognise from the double lines 9 and 10 that the position of the single square is influenced by the random generator J1, the angular position by J2. The successively widened spread of the relative position coordinates P and Q and the position angle PSI of the individual square is controlled by a counter I, which is incremented each time QUAD is called (see line 14).' (Translation JW)

Without going into details at this point, it can be seen in the SCHOTTER program (left, marked in blue) that a QUAD procedure (marked in red) is defined there, which is passed when the SERIES procedure (right, marked in green) is called. In the SERIES procedure, you can see that the squares generated by the QUAD

procedure are displayed row by row with random position and angle in two nested loops (marked in grey).

The QUAD procedure creates the elementary elements of the image, the squares. Each square has a fixed edge length, a specific position and orientation. The exact values for the position and orientation of the next square to be drawn are determined using counters, which are increased by the value one each time the loops are run. As the counters increase, the deviations also increase.

With his code for the image *Schotter*, Nees has provided the template for Recoding his image. The attempt to recode works of early Computer Art already existed before the end of the first heyday of Computer Art in the mid-1980s. One reason for this was and is the interest in learning from existing works and gaining inspiration - an approach that has a long tradition in the training of classical painting with the copying of known works for practice purposes. In Computer Art, the possibility of learning concerns both the technical aspects and the composition of the picture.

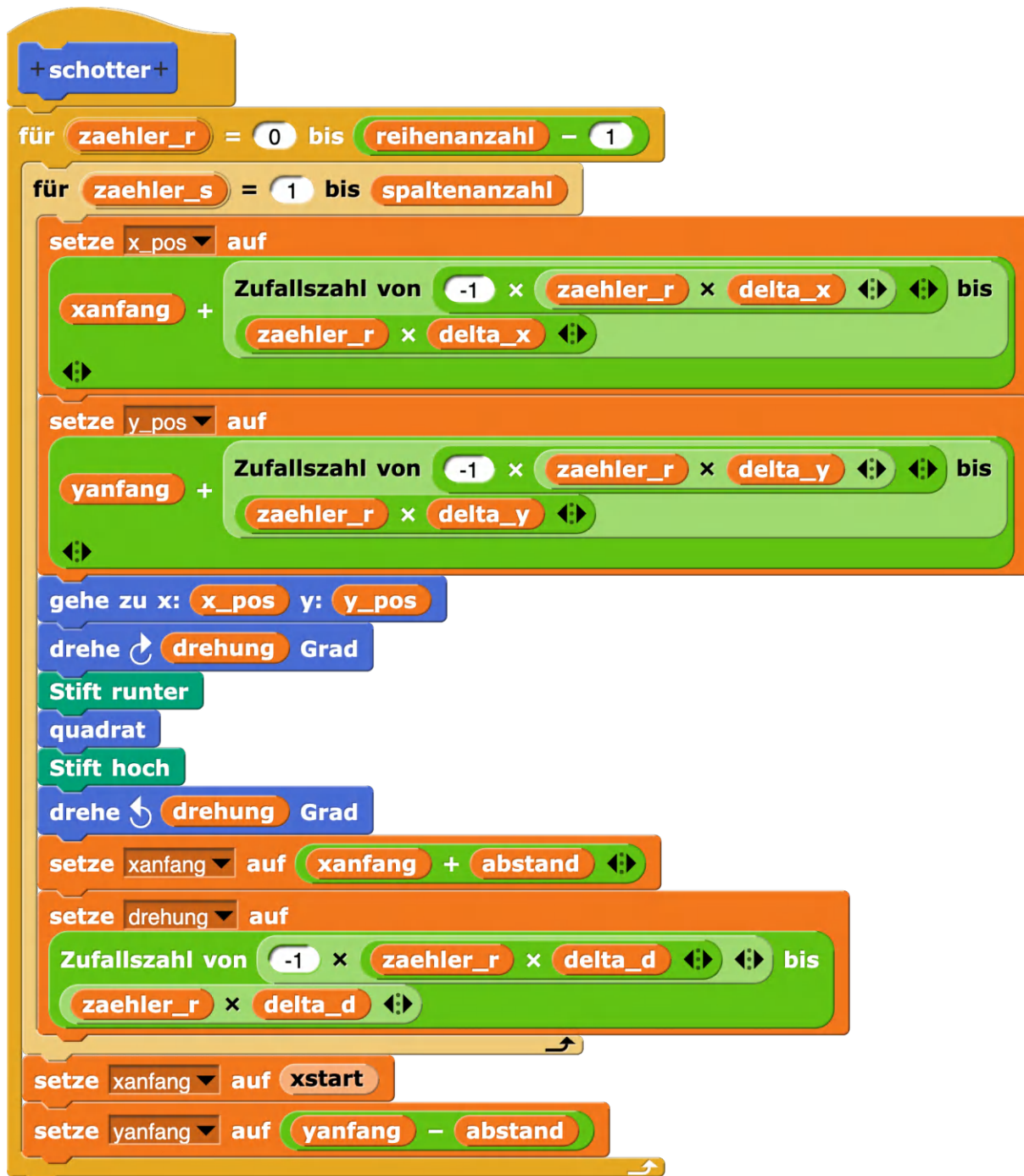
My first own attempt at Recoding *Schotter* was in 1988 with the turtle graphic, which I took up again and repeated in 2014 with ACSLogo¹⁰. Since I started working independently on Computer Art, I have been using the visual program-ming environment Snap!¹¹, which has proven to be the ideal tool for me (for more details, see my textbook *Codierte Kunst*, Wedekind, 2018).

If we closely follow Nees' code and implement this in Snap!, we get the program **schotter** shown on the following page. Differences can be found in the labelling of the characteristics:

- **xstart, ystart**: Starting point of the drawing process
- **xanfang, yanfang**: Starting point of a row of square
- **xpos, ypos**: Current position of a single square
- **spaltenanzahl, reihenanzahl**: Number of columns and rows
- **laenge**: Side length of the squares
- **abstand**: Horizontal and vertical distance between the squares
- **drehung**: Angle of rotation of a square

¹⁰ This Logo version - introduced by Alan Smith in 2002 - is still available for download today as ACSLogo version 1.6.0.2 (for MacOS X only) (<https://www.alancsmith.co.uk/logo/>).

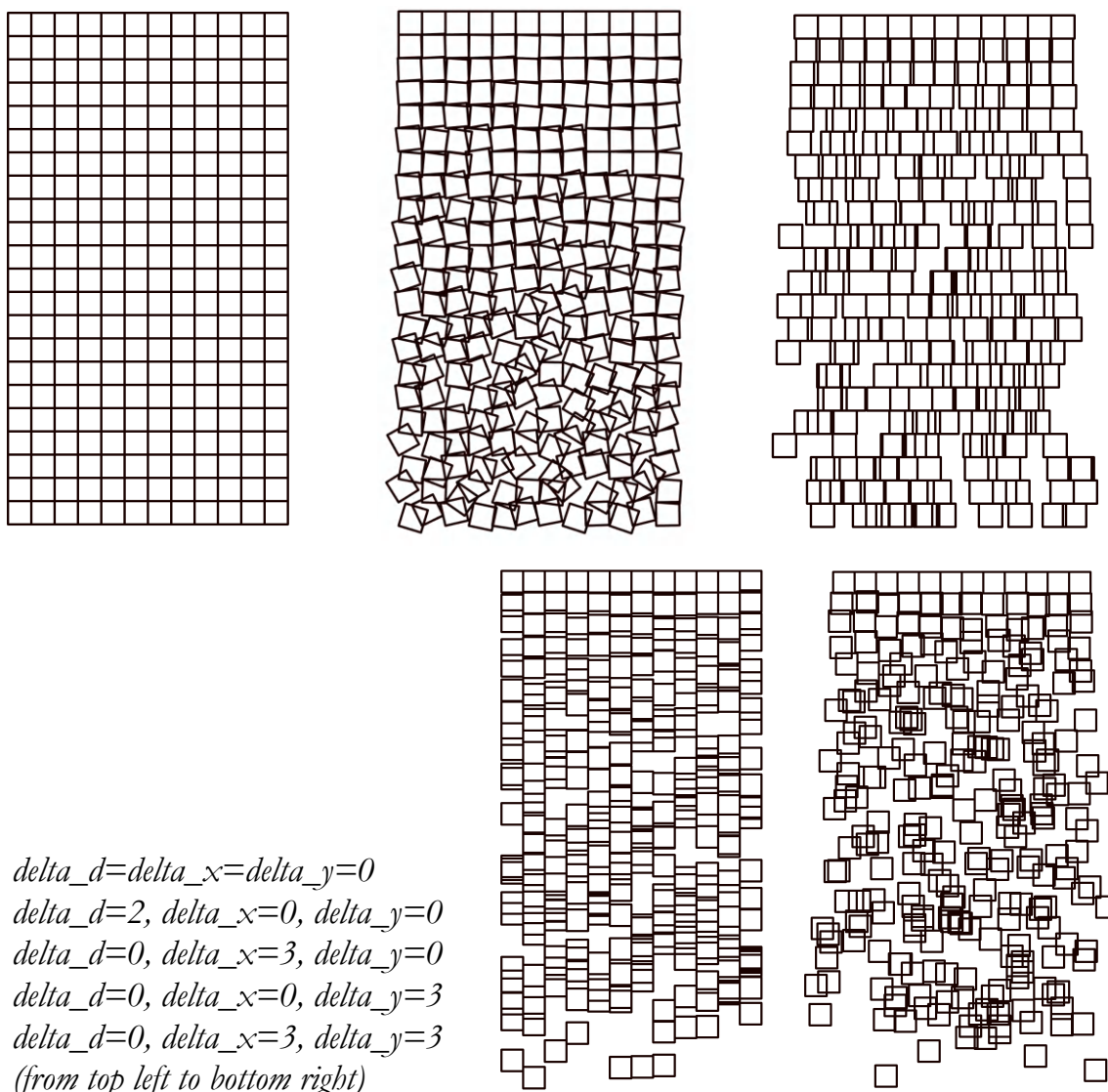
¹¹ Snap! (<https://snap.berkeley.edu/>) is developed by Brian Harvey and Jens Mönig and is freely available on the Internet. The system runs in the browser, therefore it requires no installation and runs on all common platforms.



- **delta_d**: Degree of change in the angle of rotation from row to row
- **delta_x, delta_y**: Degree of change in the x and y position of the squares from row to row

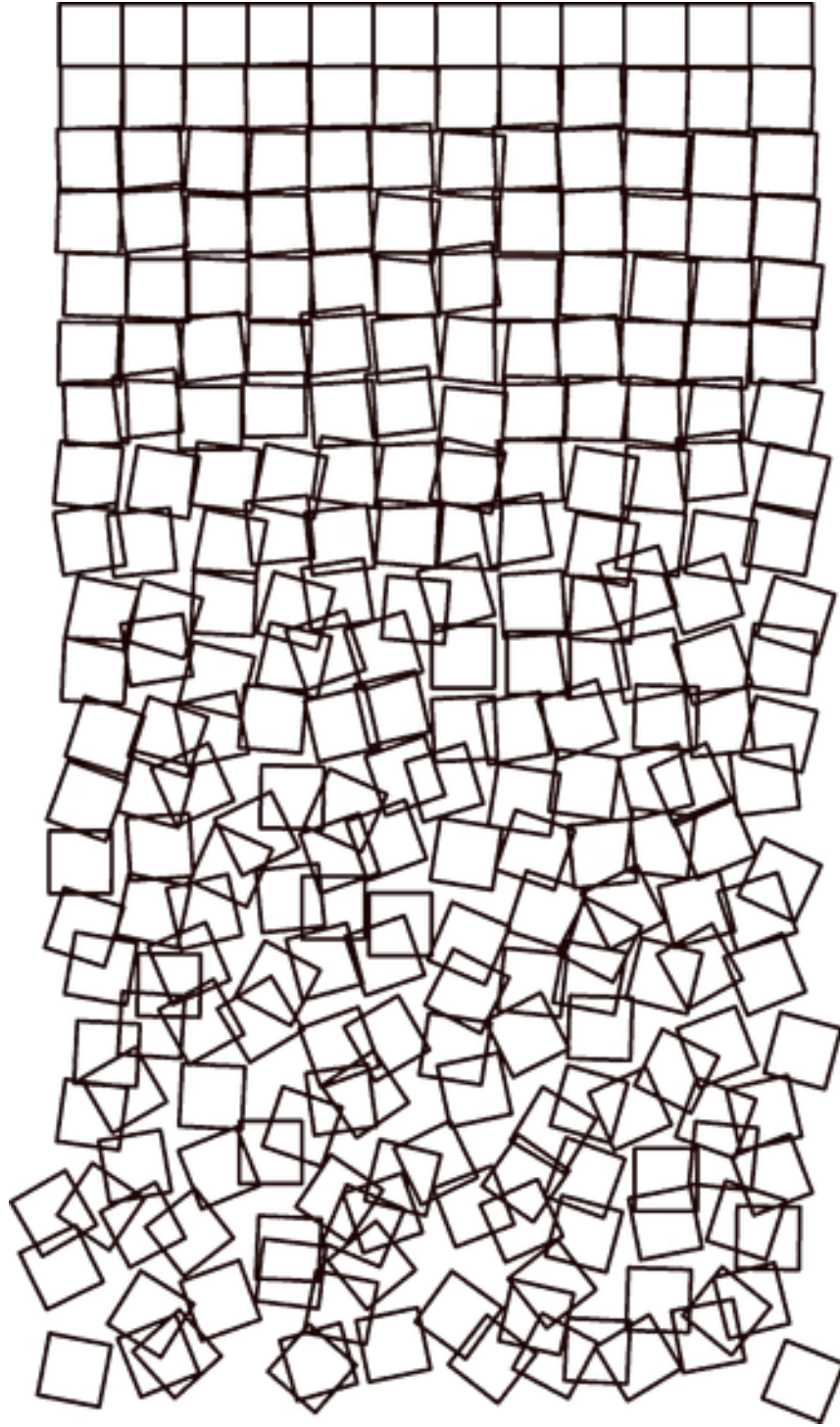
After starting the program, the initial values of this key data are defined and then the block **schotter** is executed. Despite the deviations in detail, this can result in an image that is very similar to the original.

The simple change of the key data **delta_d**, **delta_x** and **delta_y** allows the targeted testing of random influences on the image structure.

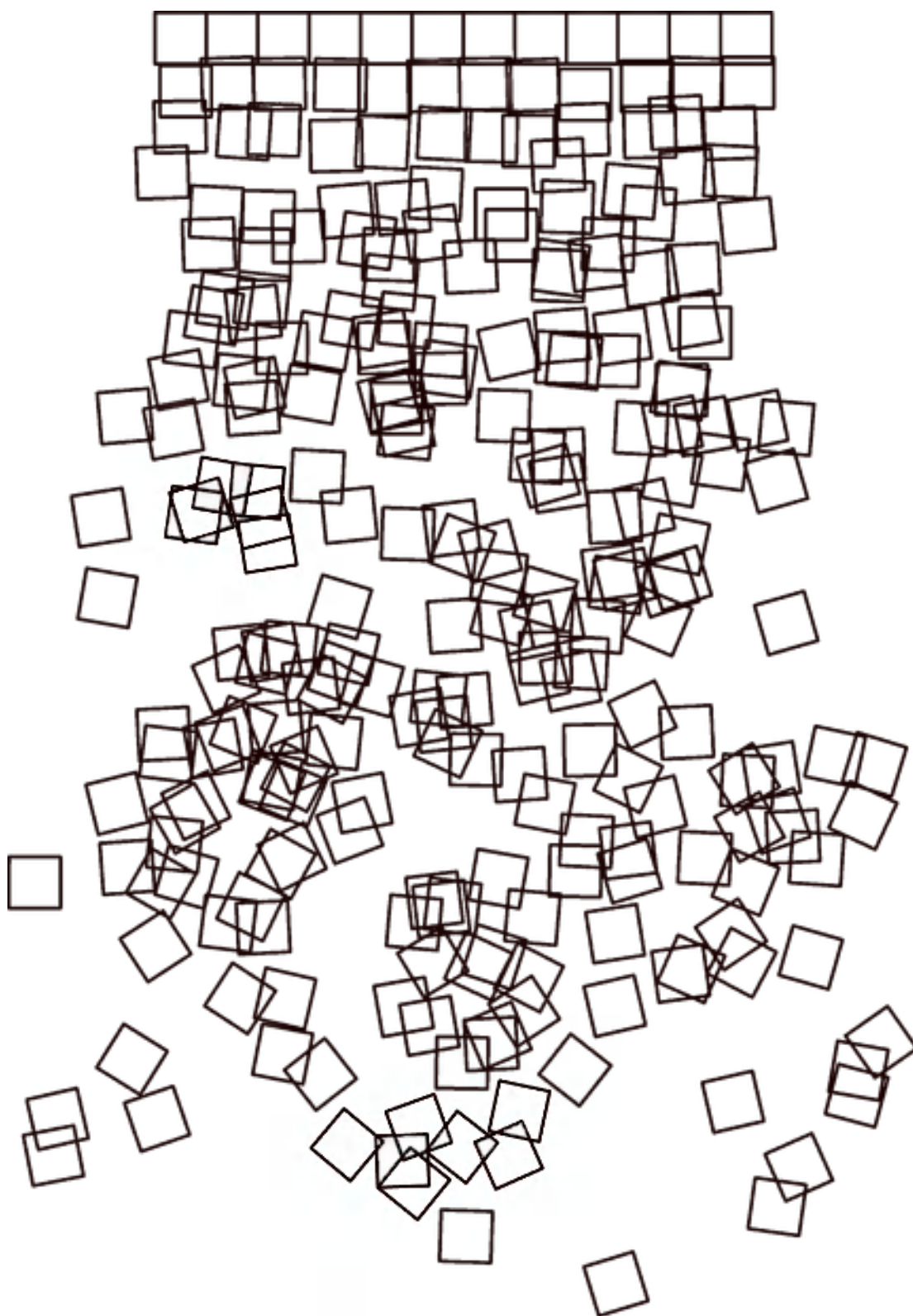


The five images show the result when restricting to individual parameters: all = 0, with rotation, with horizontal and vertical shifting of the squares, as well as the combination of these two parameters. The following two images are examples of my Recoding, in which values other than 0 are taken into account for all parameters at the same time, whereby the first image is very close to the original¹².

¹² You can test this Recoding yourself: Schotter Nees original [\[Program 2\]](#)



Schotter Recoded, JW 2014



Schotter Recoded, JW 2025

The picture *Schotter* provides the background for important statements of principle on Computer Art, whereby I can essentially refer to Frieder Nake. He speaks of the *doubling of things* in digitally generated images, i.e. that every digital artwork has a visible and a calculable element. He refers to the visible digital artwork as the *analogue surface* and the algorithm or program as the *digital subface* (Nake, 2016).

As viewers, we generally have no access to the subface developed by the artist. In the case of *Schotter* and many of his other pictures, however, Georg Nees actually allows us to see both: the subface with the algorithms and programs in his books and the surface in the form of the pictures that have found their way into many exhibitions.

Nake likes to say that we should *Think the Image, Don't Make It!* (Nake & Grabowski, 2017), because in Computer Art, the making has been handed over to the machines. Thinking the images implies that the individual image is always only a representative, an instance of *an entire class of possible images* that can theoretically be created by the algorithms developed by the creators. The corresponding computer program thus becomes the operative description of an infinite number of images.

By using the above-mentioned characteristics, which can basically be varied at will, the *Schotter* program is a typical example of such a class of images as described by Nake. In the spirit of Georg Nees, it is possible and necessary to select and document the most striking findings from this variety of theoretically possible results of our aesthetic laboratory.

As already mentioned, I am neither the first nor the only one to recode *Schotter*. The generative artist with the pseudonym *hex6c* describes his own approach to Recoding (and Remixing) *Schotter* and other role models in the same way as I did (hex6c, 2020) as follows:

- Start from the original template (for *Schotter* this is the original code);
- Reverse engineering of the code (therefore not necessary for *Schotter*);
- Compare the results of the code with the original template, optimise and adapt the code accordingly to obtain image versions as close as possible to the original (for *Schotter* by varying the parameters).

However, if the original code is not available (as is unfortunately usually the case), the process becomes somewhat more complex. For me, the following procedure has proven itself, which also applies to the development and realisation of my own image ideas:

- The first step is always a precise *analysis of an image template*: Which basic elements are included (which shapes, colours and recurring patterns and structures)? Which operations are applied to them (e.g. geometric transformations, repetitions, variations)? What role does chance play (random changes in frequencies or shapes of the picture elements)?
- As soon as we can deduce exactly what we want to draw, we need to describe how the image elements and the image structure are to be generated. To do this, we need an *algorithmic description*. This includes the definition of parameters such as the number and positioning of the graphic elements, as well as their size, alignment and colour.
- *Programming* is the final step in the creation of Computer Art. In this phase, we develop the digital surface. The algorithms must be implemented using our chosen programming language so that they can be executed by the machine.

My analysis of works of early Computer Art has shown that just a few programming concepts are sufficient for the first steps in Computer Art. These include the *repetition* of a few basic pictorial elements (squares are sufficient for Schotter), their *variation* and *systematic or random changes to central parameters*. In this way, the class of all images that can be generated by *Schotter* can be described algorithmically.

Information technologist and software engineer Jim Plaxco is a digital artist and creative programmer as well. He has also reprogrammed the *Schotter* picture. Plaxco used *Processing*¹³, an object-oriented programming language with an associated integrated development environment, which is based on the Java programming language and specialises in the areas of graphics, simulation and animation. He has described his approach in a short tutorial (Plaxco, 2006).

¹³ You can download his program and test it in the Processing environment:
https://www.artsnova.com/Nees_Schotter.pde

```

// Georg Nees Schotter Reproduction by Jim Plaxco,
www.artsnova.com
int columns = 12;           // number of columns of squares
int rows = 22;              // number of rows of squares
int sqrsz=30;               // size of each square
float rndStep=.22;          // Rotation Increment in degrees
float randsum=0;            // Cumulative rotation value
int padding=2*sqrsz;        // margin area
float randval;              // random value for translation and
rotation
float dampen=0.45;          // soften random effect for position

void setup() {
  size((columns+4)*sqrsz,(rows+4)*sqrsz);
  background(255);          // set background color to white
  stroke(0);                // set pen color to black
  smooth();                 // use line smoothing
  noFill();                 // do not fill the squares with color
  rectMode(CENTER);         // use x,y value as the center of the
square
  noLoop();                 // execute draw() just one time
} // end of setup()

void draw() {
  for (int y=1; y <=rows; y++){
    randsum+=(y*rndStep);    // Add to the random value
    for (int x=1; x<=columns; x++) {
      pushMatrix();
      randval=random(-randsum,randsum);
      translate( padding + (x * sqrsz) - (.5*sqrsz) +
(randval*dampen),
padding + (y * sqrsz) - (.5*sqrsz) +
(randval*dampen));
      rotate(radians(randval));
      rect(0,0,sqrsz,sqrsz);
      popMatrix();
    } // end of x loop
  } // end of y loop
} // end of draw()

```

With the code shown from Plaxco, you get a *Schotter* image that is very similar to the original. He also suggests testing some variations by changing individual lines of the program¹⁴.

Zellyn Hunter (2024-1) has undertaken a special form of Recoding. He implemented his *Schotter* version using the Python programming language. He orientated himself very closely to the original code with variable and procedure names.

¹⁴ Jörg Kantel (the *Schockwellenreiter*) has created a version of this program with Processingpy - the Python version of Processing - and has also made the code available: <http://blog.schockwellenreiter.de/2019/04/2019042501.html>

```

# SCHOTTER
R = 5 * 1.4142
HALFPI = 3.14159 * 0.5
PI4T = HALFPI * 0.5
I = 0
P = Q = PSI = 0.0

def QUAD():
    global I, JE1, JA1, JE2, JA2

    JE1 = 5*I/264
    JA1 = -JE1
    JE2 = PI4T * (1+I/264)
    JA2 = PI4T * (1-I/264)
    P1 = P+5+J1()
    Q1 = Q+5+J1()
    PSI = J2()

    LEER(P1+R*COS(PSI), Q1+R*SIN(PSI))
    for S in (1,2,3,4):
        PSI = PSI+HALFPI
        LINE (P1+R*COS(PSI), Q1+R*SIN(PSI))
    I += 1

def SERIE(QUER, HOCH, XMAL, YMAL, FIGUR):
    global P, Q

    P = -QUER * XMAL * 0.5
    Q = YANF = -HOCH * YMAL * 0.5

    for COUNTX in range(1, XMAL+1):
        Q = YANF
        for COUNTY in range(1, YMAL+1):
            FIGUR()
            Q = Q + HOCH
            P = P + QUER
        LEER(-148.0, -105.0)
        CLOSE()

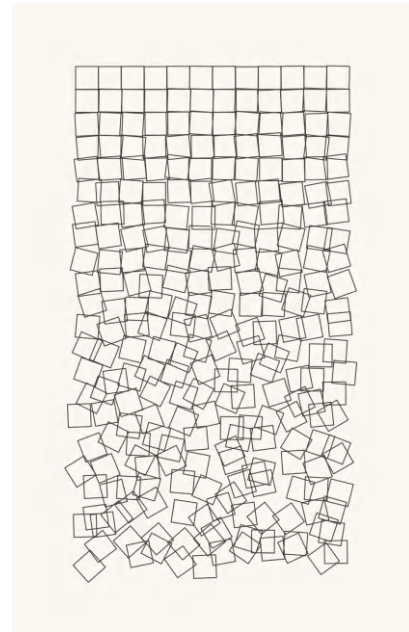
SERIE(10.0, 10.0, 22, 12, QUAD)
# /SCHOTTER

```

However, Zellyn Hunter is not satisfied with simply recoding the *Gravel* principle and creating images that are as similar as possible to the original. He wants to reproduce the original down to the last detail, i.e. all the positions and rotations of the squares in the 22 rows.

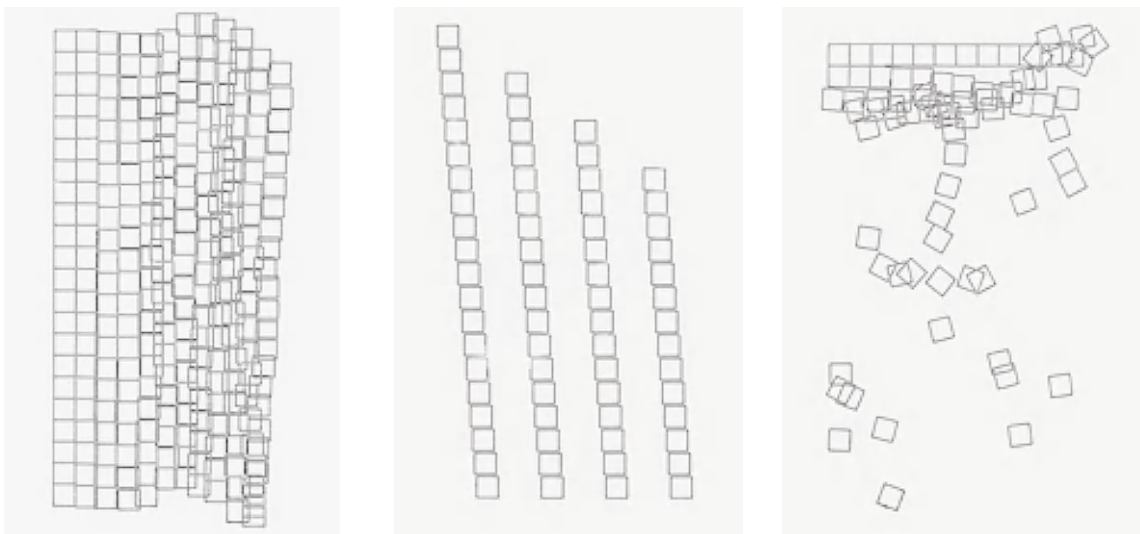
Consequently, this means, that the use of the same random number generator that Nees used is required. Hunter found what he was looking for in Nees' dissertation (Nees, 1969, p. 113 ff.).

Using this generator and finding the starting values for the x- and y-displacement as well as the rotation, he succeeded in reproducing one of the originals 1:1 (Hunter, 2024-2; he refers to the version in the collection of the Victoria and Albert Museum, see p. 12).



*Hunter: 1:1 Recoding of Schotter
(Version V&A Museum)*

Media scientist Martin Zeilinger deals with questions of authorship, creativity, originality and ownership in new technological contexts. In his *Pattern Recognition* project, he combines a critical approach with an artistic approach (Zeilinger, 2018). Using the example of *Schotter*, he demonstrates the underlying aspects of artistic collaboration between humans and computers in Generative and Algorithmic Art by analysing the original work in detail, rewriting the code and reproducing it using technical devices comparable to the originals.



*Zeilinger: Recoding Schotter Iteration_7488 (left); Iteration_0041 (centre);
Iteration_7780 (right), 2017*

By linking this with socio-legal issues, Zeilinger addresses a fundamental problem with Recoding and Remixing, which is currently becoming more apparent in Generative AI. Does this not merely result in imitations, replicas, i.e. repetitions of artistic works? Basically, the issue of plagiarism raises the same question that early Computer Artists were confronted with: what is art in their work?

Using the example of the *ReCode Project*¹⁵, Schneeberger (2013) already discussed the boundaries between the conservation efforts to preserve Digital Art or make it accessible and plagiarism. His short essay on this topic should be required reading if the programming exercises in the quiet chamber are to become products that find their way into the public space!

¹⁵ The *ReCode project* is a community-driven project to preserve Computer Art by translating it into a modern programming language (Processing): <http://recodeproject.com/>

Remixing Schotter

Recoding is often simple - even without using the original code - if the image structures are relatively easy to understand. *Recoding* is now followed by *Remixing*! This refers to the reworking and combining of creative artefacts, usually in the form of music, video or other interactive media, here analogous to my use of visual templates. Recoding aims to achieve a largely similar reproduction of the respective original (in extreme cases even with the claim of identical reproduction as with Zellyn Hunter). In Remixing, on the other hand, the original serves as a source of ideas for minor or even major visual changes.

In our context, Remixing means that the images and source elements created during Recoding can be recombined, modified and expanded to create new works. This can range from careful changes close to the original to the free combination of known elements in new constellations. These can already be achieved through small additions or changes to the original code. However, if the visual structure is changed, this may require reprogramming or even new programming.

In Computer Art, the artefacts used can therefore originate from two different levels:

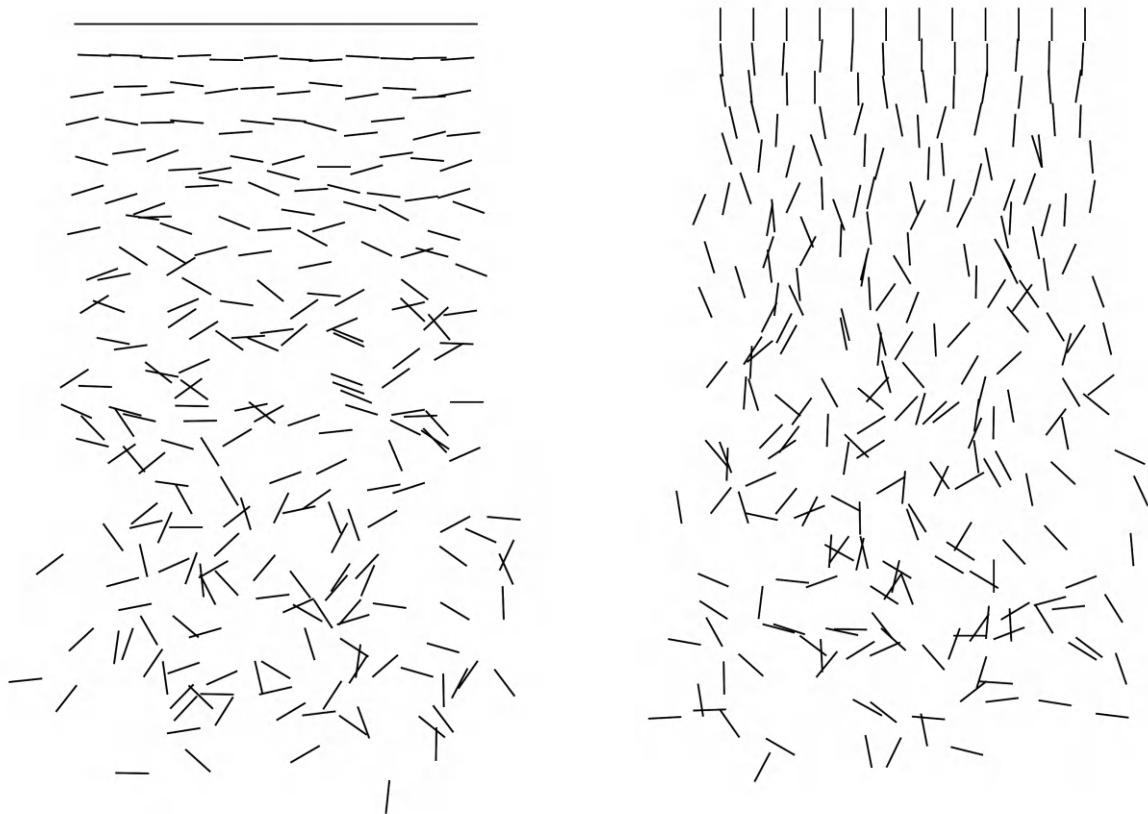
- On the one hand, these can be the graphic results themselves or individual elements and structures taken from them, which can appear in new contexts and combinations.
- On the other hand, it can be the algorithms or programs with which the aesthetic objects were created, from which individual procedures or code snippets are then extracted.

A basic attitude in Remixing should always be that the sources remain recognisable and named, and that these sources are handled transparently (Stalder, 2009).

In the following, I would like to present several examples of Remixing understood in this way. They come from myself and others who have continued to work on the example of *Schotter* in this sense. Some of the results are the result of slight changes to the original, others are new versions inspired by it.

In 2015, I showed several examples of Remixing alongside the Recoding of *Schotter* in an exhibition to mark 50 years of Computer Art.

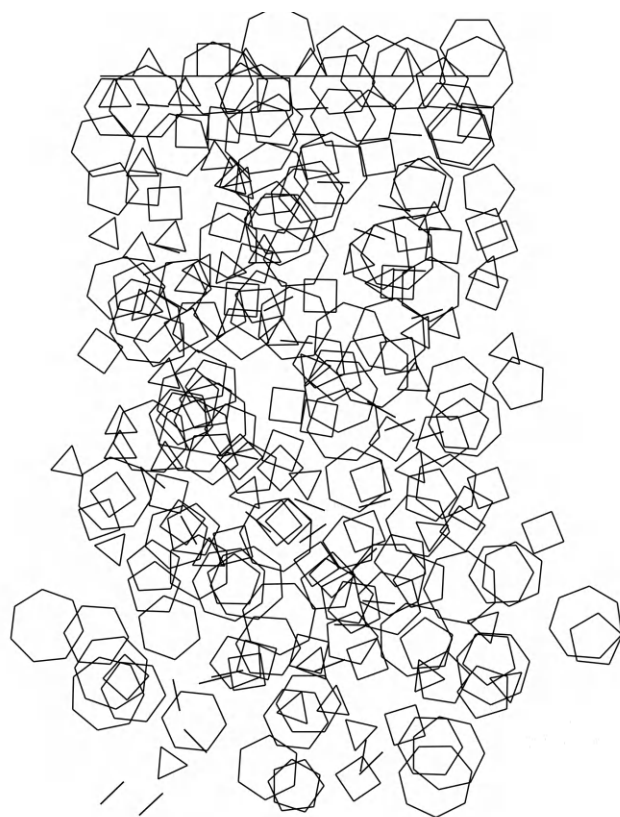
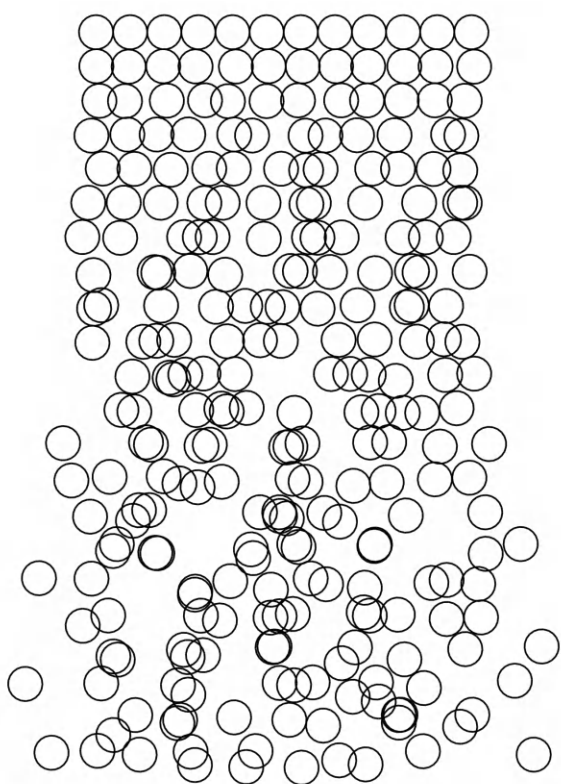
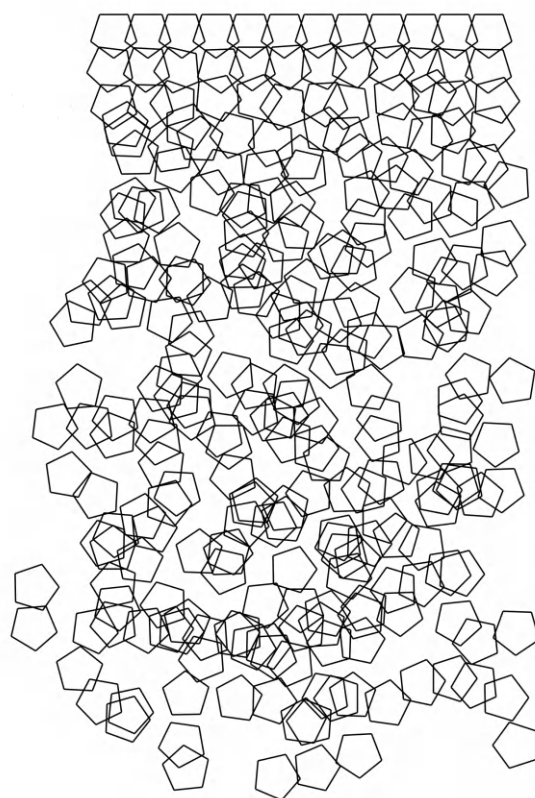
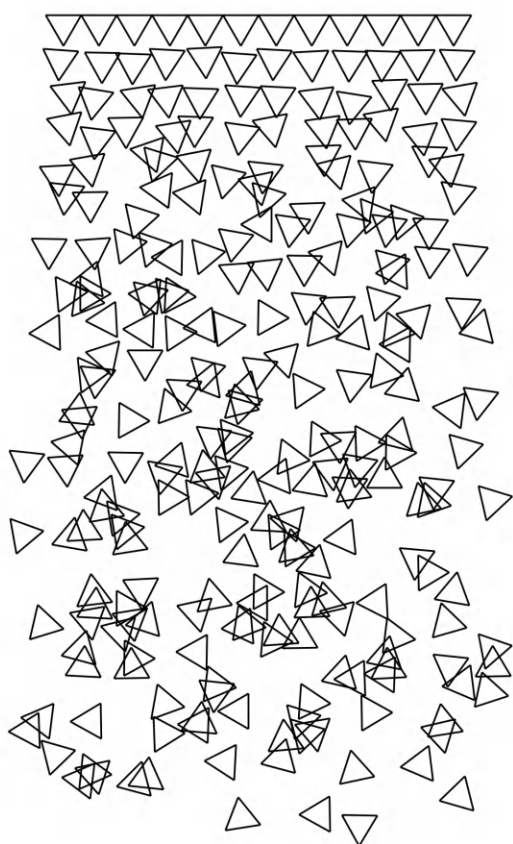
A first series showed polygons with any number of corner points - deviating from the squares as the basic element with four corner points. With two corner points, this results in lines that can be orientated horizontally or vertically in the starting position.



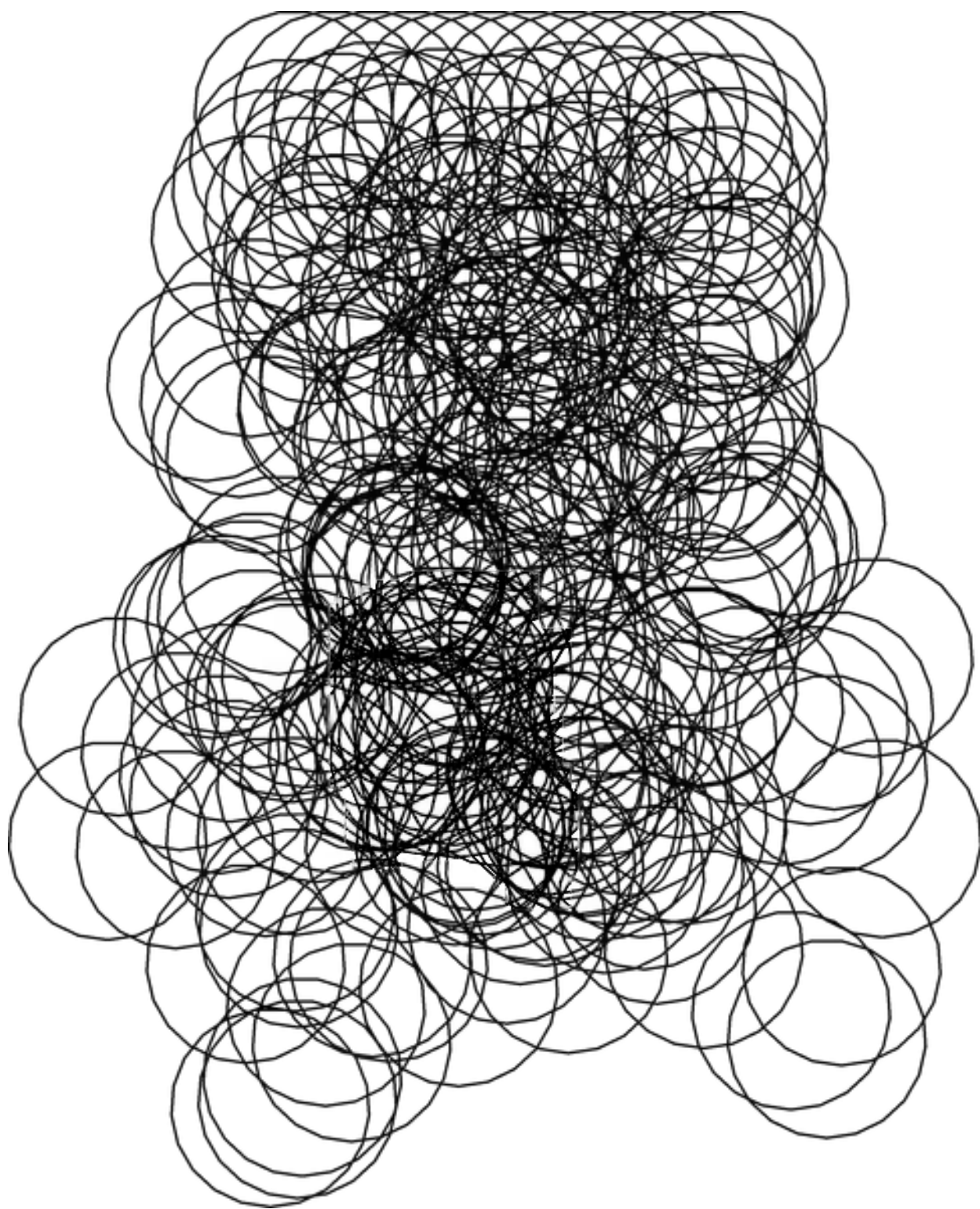
Schotter Remixed, Two corners, horizontal (left) and vertical (right) aligned, JW 2015

All other polygons with three or more corner points behave in principle like the squares in *Schotter*. With very many (>20) corner points, they are circles for which the rotation factor no longer plays a role.

Such variants are very easy to implement in terms of programming. In my Snap! procedure **Schotter** shown above, simply replace the call of the procedure **quadrat** with a procedure **polygon**, to which the desired number of corner points is to be passed. The original *Schotter* version is thus automatically included when the **polygon** procedure is called with four corner points (this is also the case in my *Schotter total interaktiv* version, see p. 48).

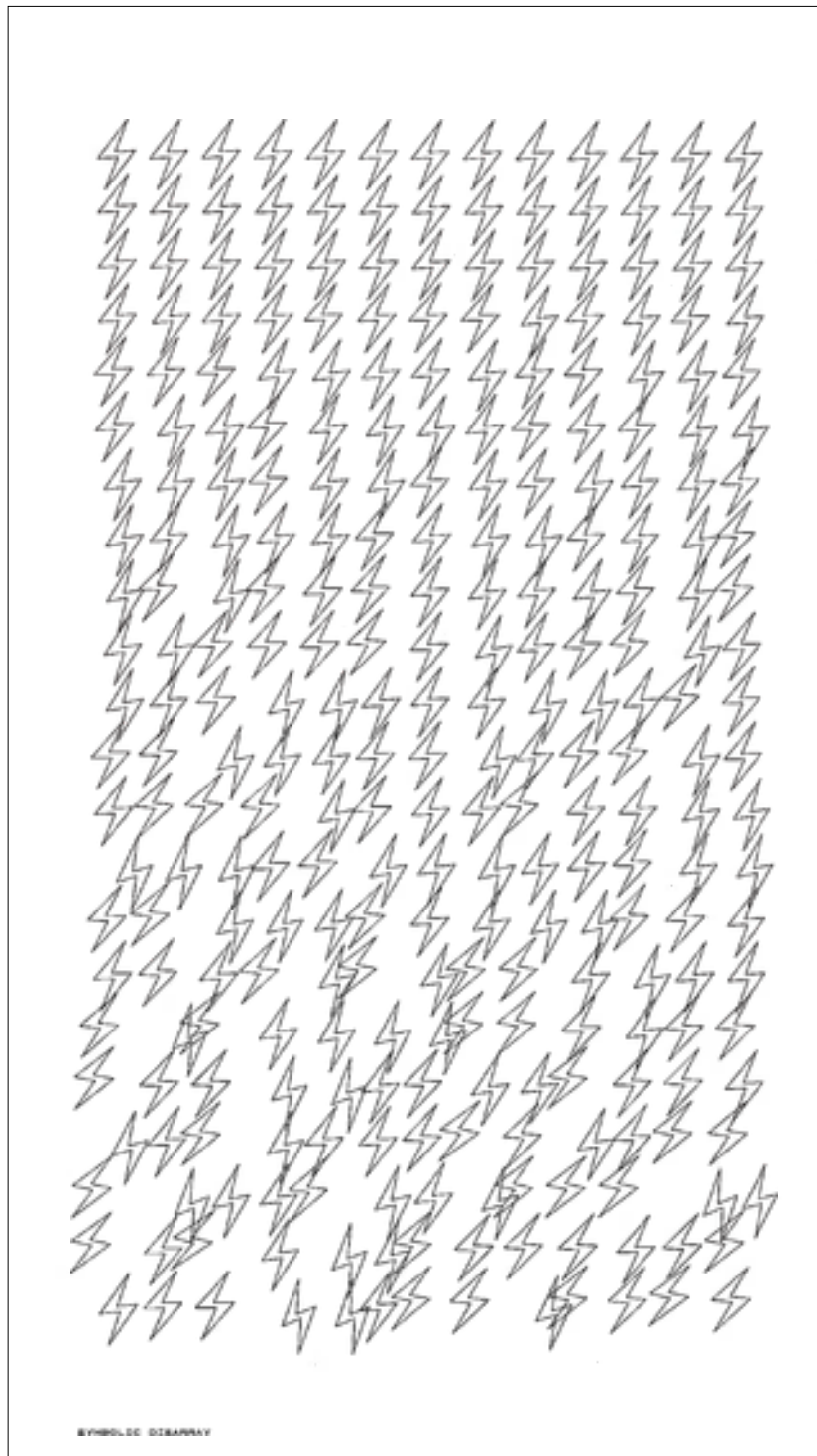


*Schotter Remixed, triangles (top left), pentagons (top right), 20-corners (bottom left)
random number of corners (bottom right), JW 2015 - 2025*



Schotter Remixed, 20-corners, JW 2015

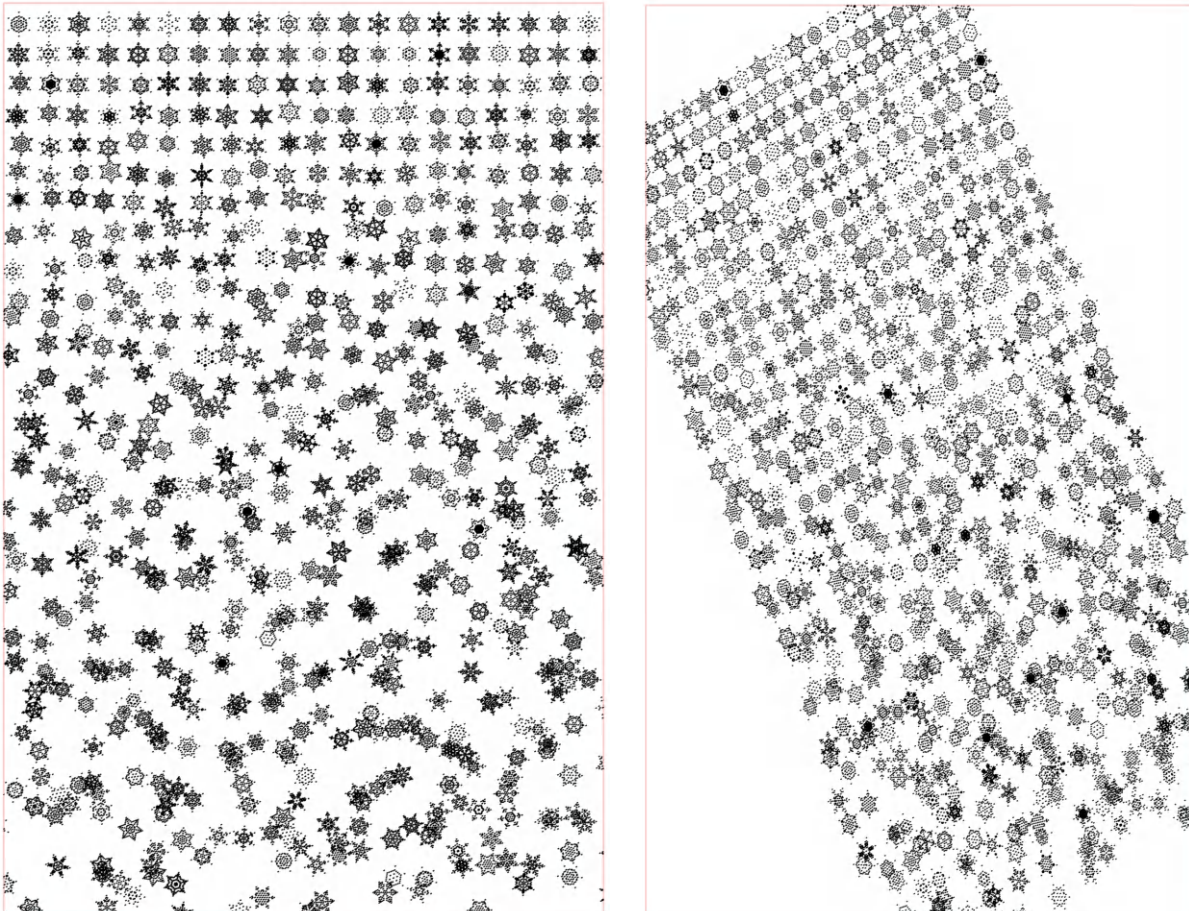
This approach can be further generalised by the fact that the pictorial elements depicted take on completely new forms. In Tobias Toft's work, for example, these are stylised flashes of lightning (Toft, 2014)¹⁶.



Tobias Toft: Symbolic Disarray, 2014

¹⁶ You can download and test his program (in this case, however, some Processing knowledge is required): <https://github.com/tobiastoft/SymbolicDisarray>

The graphic designer Daniel Catt deals with important aspects of Generative Art. In his Remixing of *Schotter*, the squares are replaced by snowflakes. However, his *Snowflakes* program is more than just a Remixing of *Schotter*. It is a working environment in which you can easily set a variety of parameters and follow the changes to the image live¹⁷. It's worth experimenting!



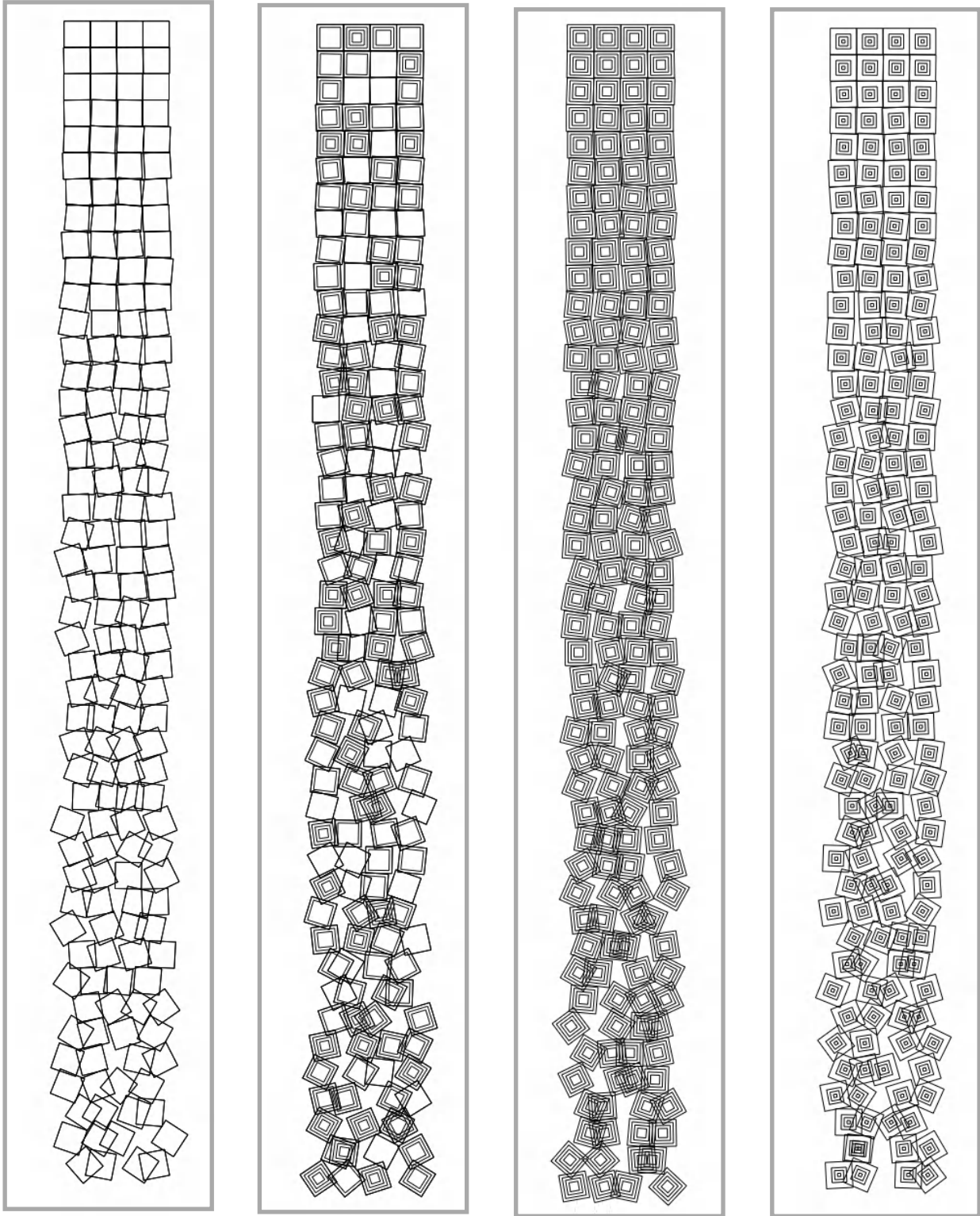
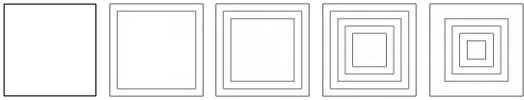
Daniel Catt: *Schotter Snowflakes* (2021), Default setting (left), x - y - z -Rotation (right), JW 2025

For a *Malen mit Maschinen* exhibition¹⁸, I modified the basic idea of *Schotter* by shaking nested squares in a long track. The result were flags with polygons that reached from the floor to the ceiling of the exhibition space.

¹⁷ Its environment runs directly in the web browser without installation: <https://revdancatt.com/penplotter/036-Snowflakes/>

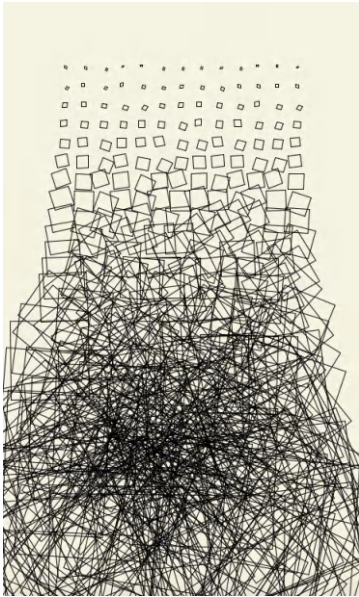
¹⁸ A documentation of the exhibition can be found here: <http://digitalart.joachim-wedekind.de/malen-mit-maschinen-2023/>

I used five basic graphic elements, each of which is drawn in a 4x4 matrix with controlled random influences.

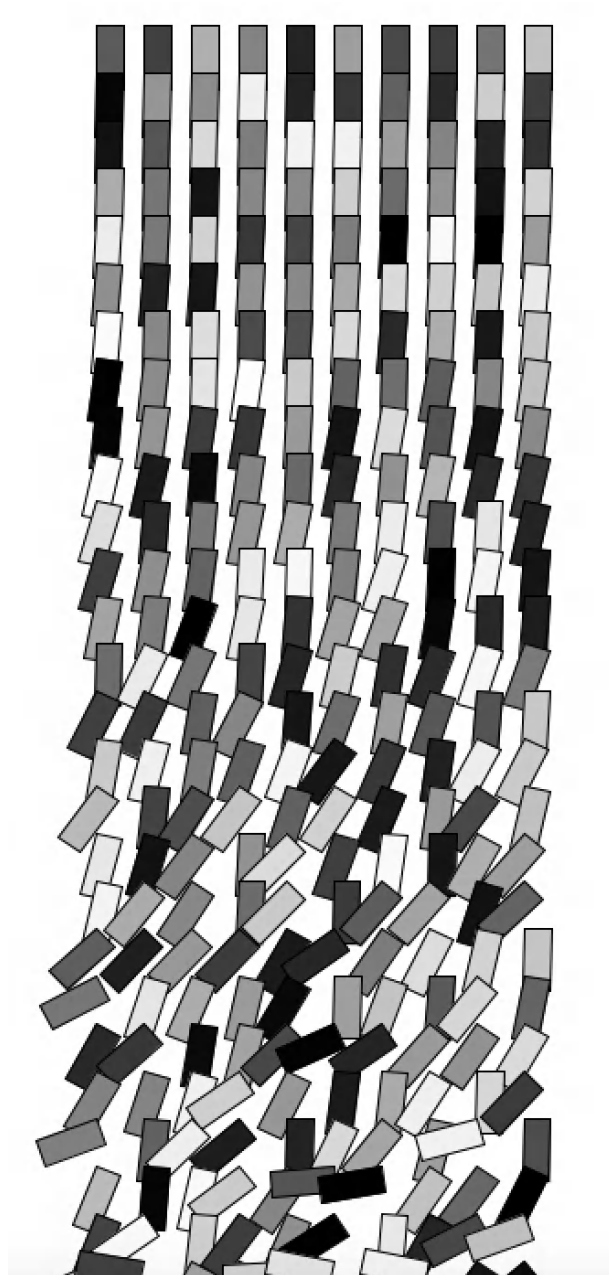
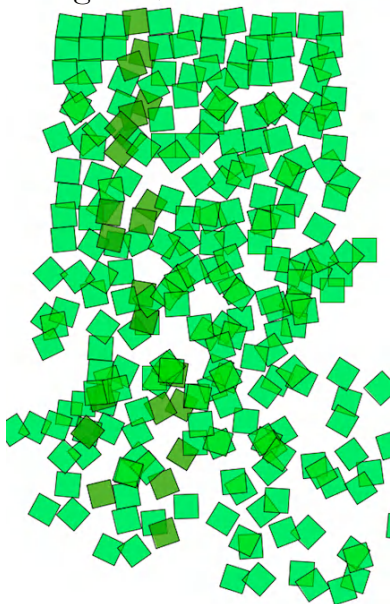


Polygonfelder 4x4 I-IV, JW 2023

Angela Ferraiolo has repeatedly offered the *Art from Code* course at Sarah Lawrence College¹⁹. A key element of this is the Recoding & Remixing of works of early Computer Art. So it is not surprising that the 2018 course also included *Schotter* by Georg Nees with the assignment *A response to Georg Nees*. The original was probably inspiring, as the very different results show. New shapes were used in the Remixing and colours were often used..

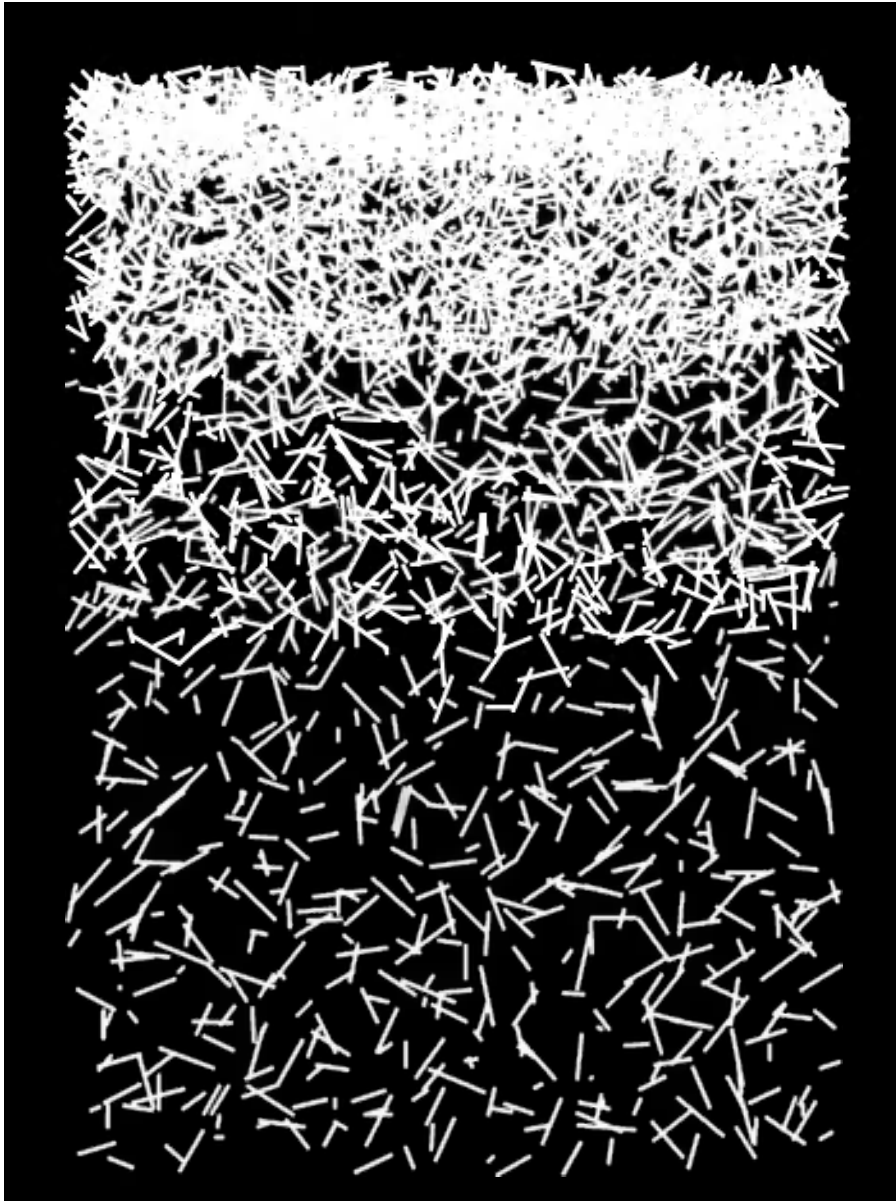


Emma Hoffmann: A response to Georg Nees, 2018



Vivian Zhou (left); Yaman Wang (right): A response to Georg Nees, 2018

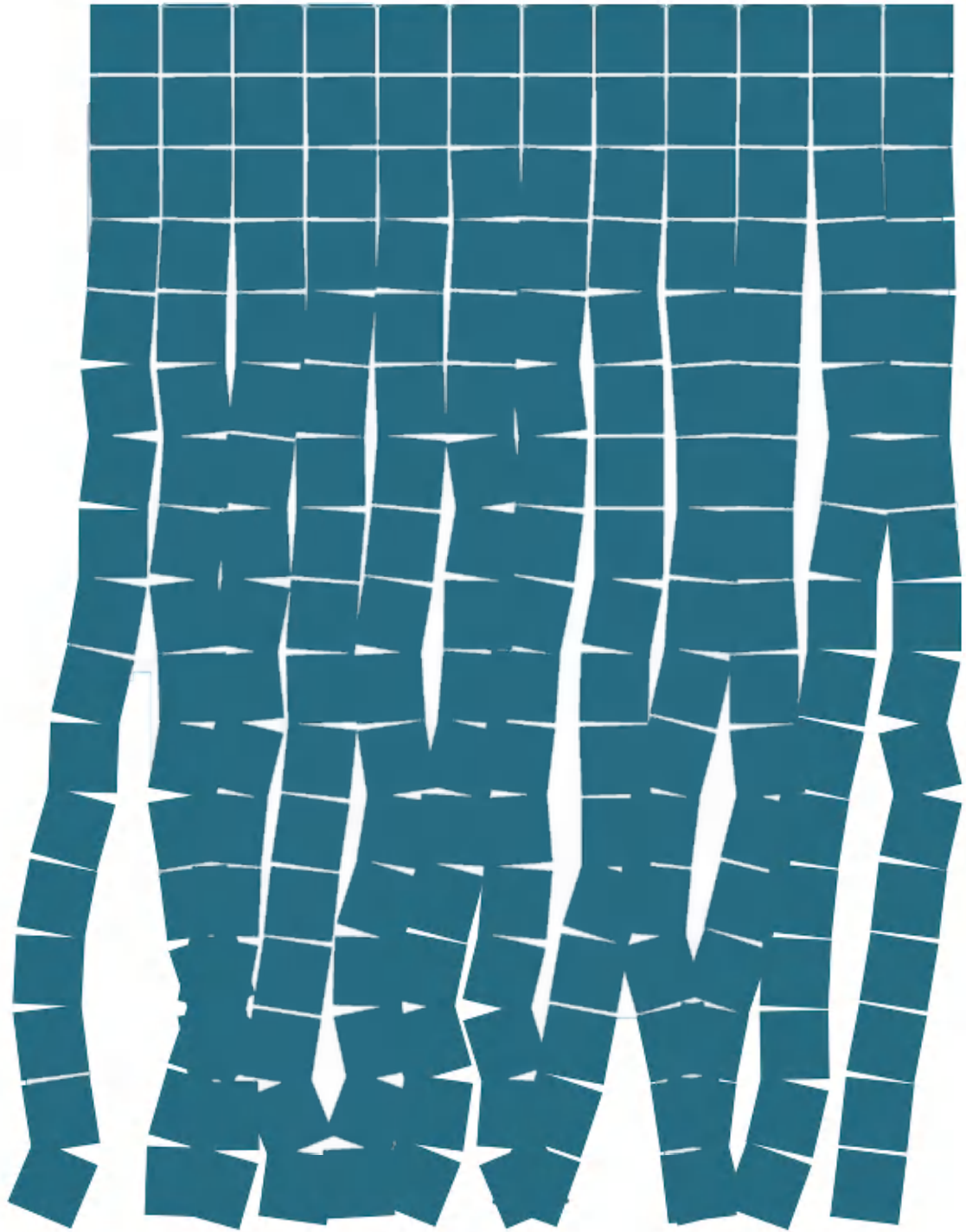
¹⁹ A documentation of the course results can be found at <https://astoryisnotatree.net/?cat=225>



Jordan Hiedeman: A response to Georg Nees, 2018

Hiedeman's work is similar to my remix of *Schotter* with polygons with two corner points, here with variable alignment and compression towards the edge of the picture.

The use of colour alone (as with Zhou or Hiedemann) already results in a clear change in the character of the picture, even with an otherwise comparable picture structure. This can also be seen in my own colour versions: The *Vorhang* version uses uniformly coloured squares. In *Loses Mauerwerk* the colour becomes increasingly transparent with the vertical position.

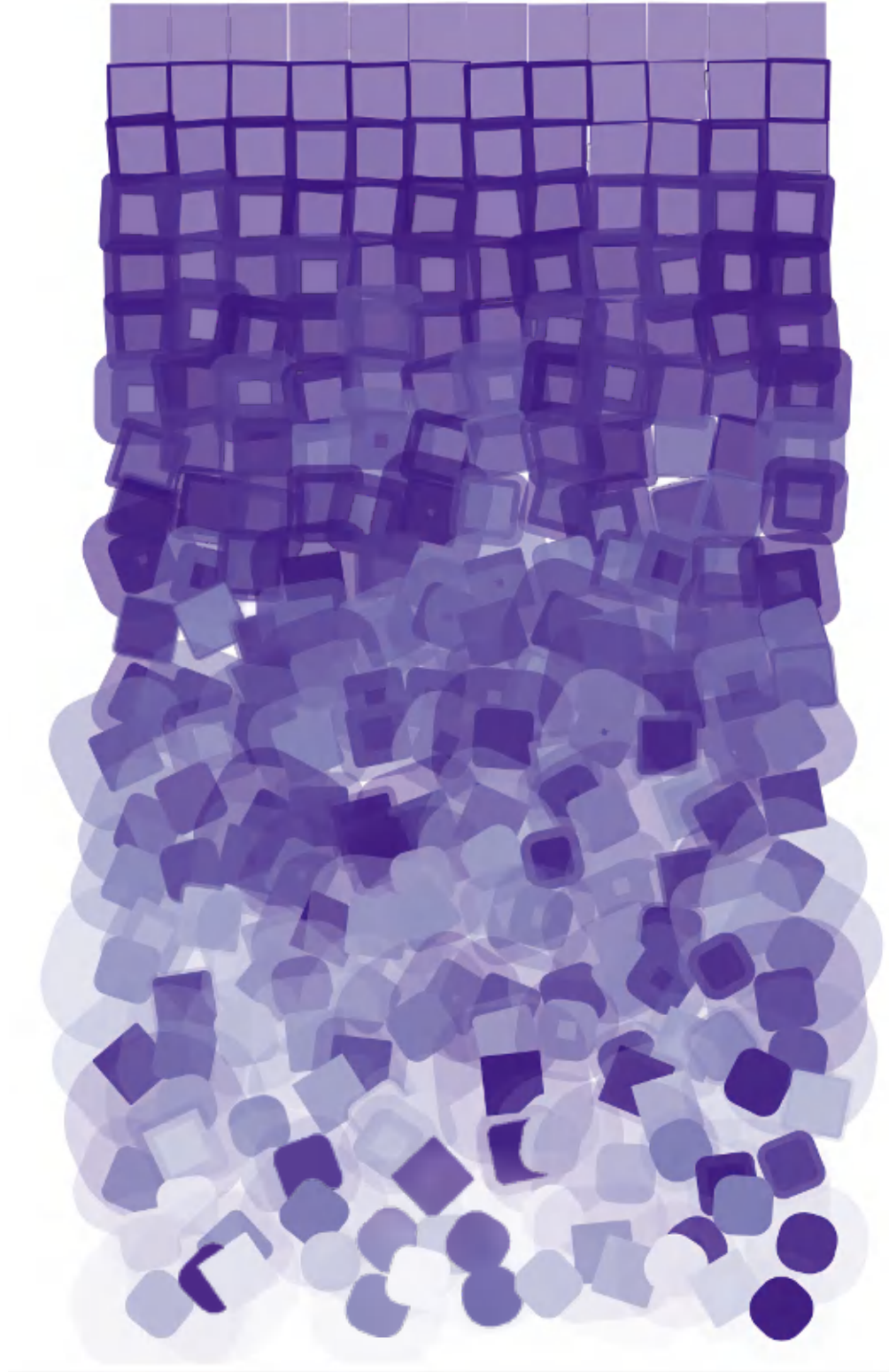


Remixing Schotter: Vorhang, JW 2019



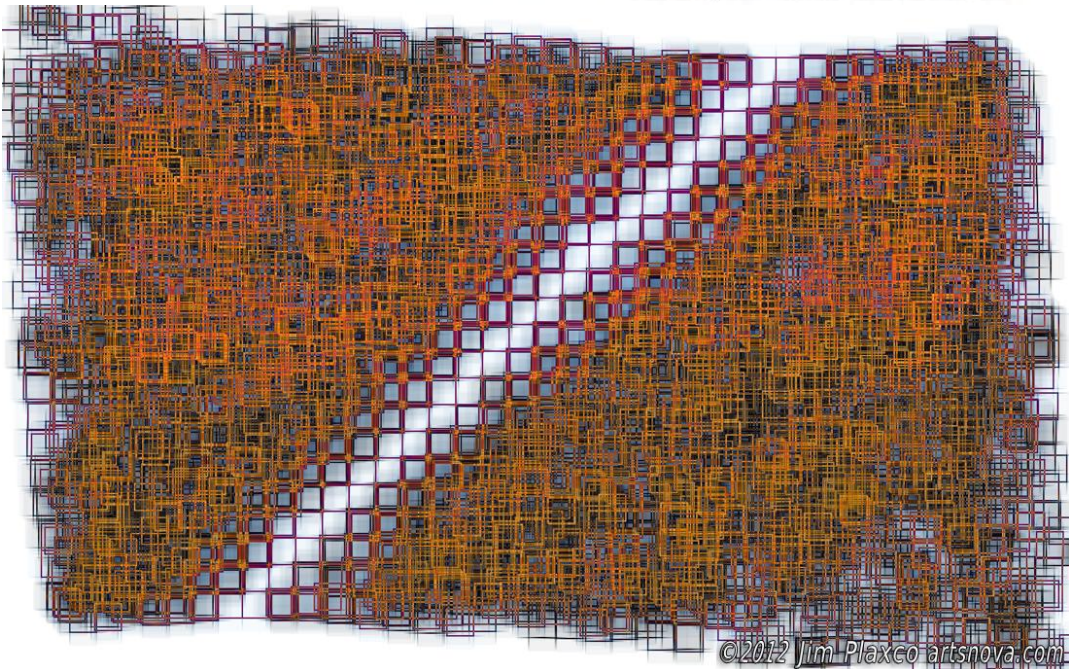
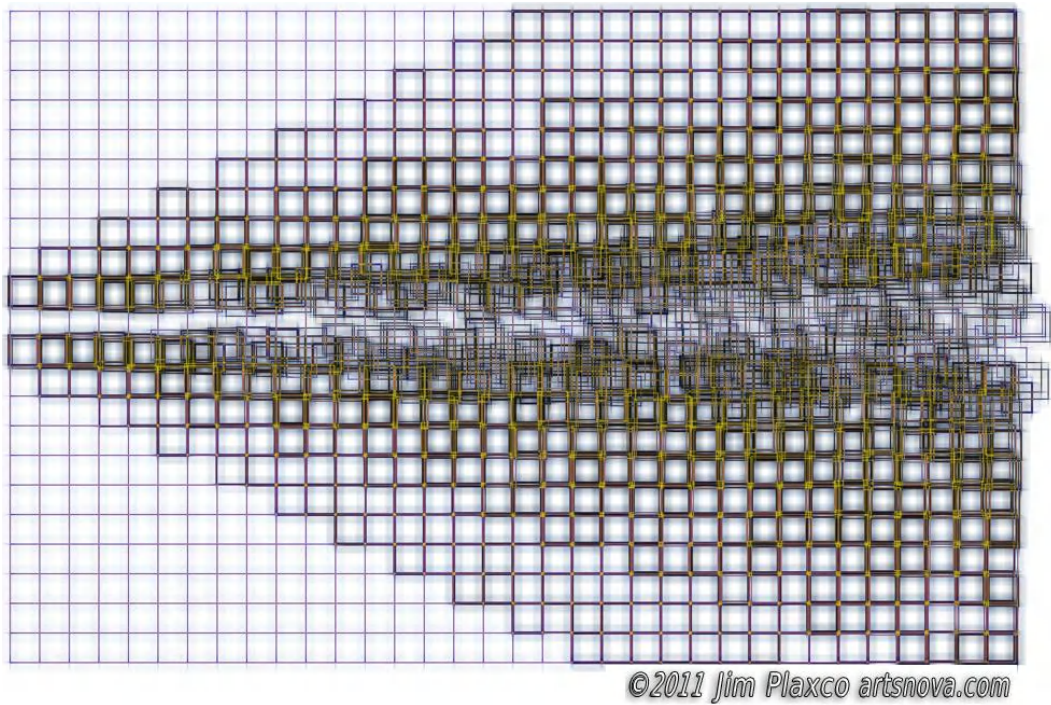
Remixing Schotter: Loses Manerwerk, JW 2022

Data visualiser Ilyá Belsky also works with colour. In his work, squares and the overlapping polygons are coloured with increasing transparency:



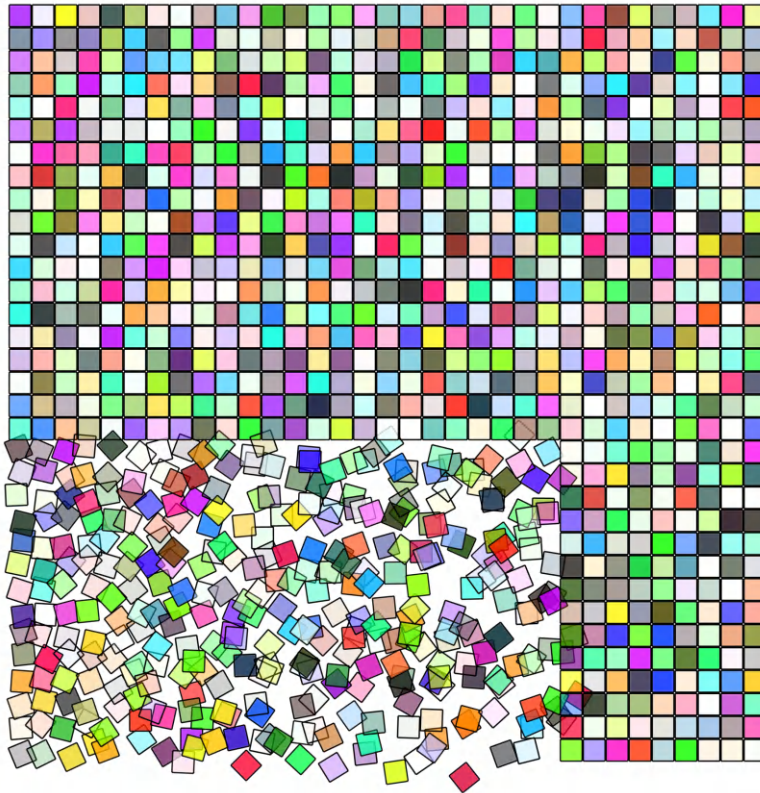
Ilyá Belsky: Remixing Schotter, 2019

The aforementioned Jim Plaxco has developed an entire series *Cubic Disarray*, which he was inspired to create by *Schotter*. His pictures are rare examples of Remixing, in which the usual vertical image alignment is replaced by a horizontal alignment. The reference to *Schotter* is no longer directly recognisable due to the colour and the arrangement with overlays..

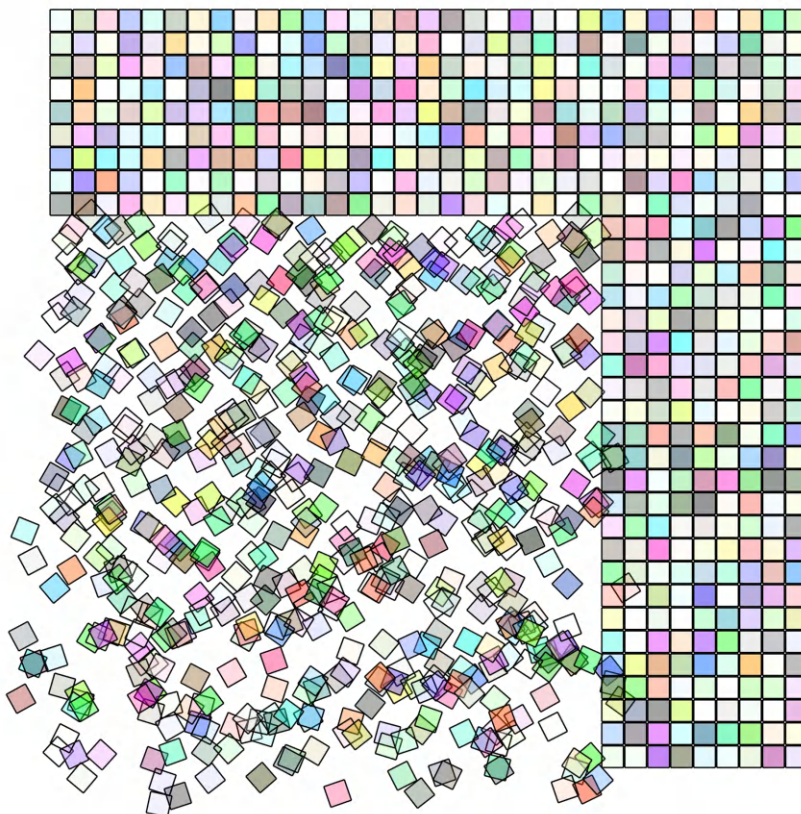


Jim Plaxco: Cubic Disarray - Turbulence (top), 2011; Bisection (bottom), 2012

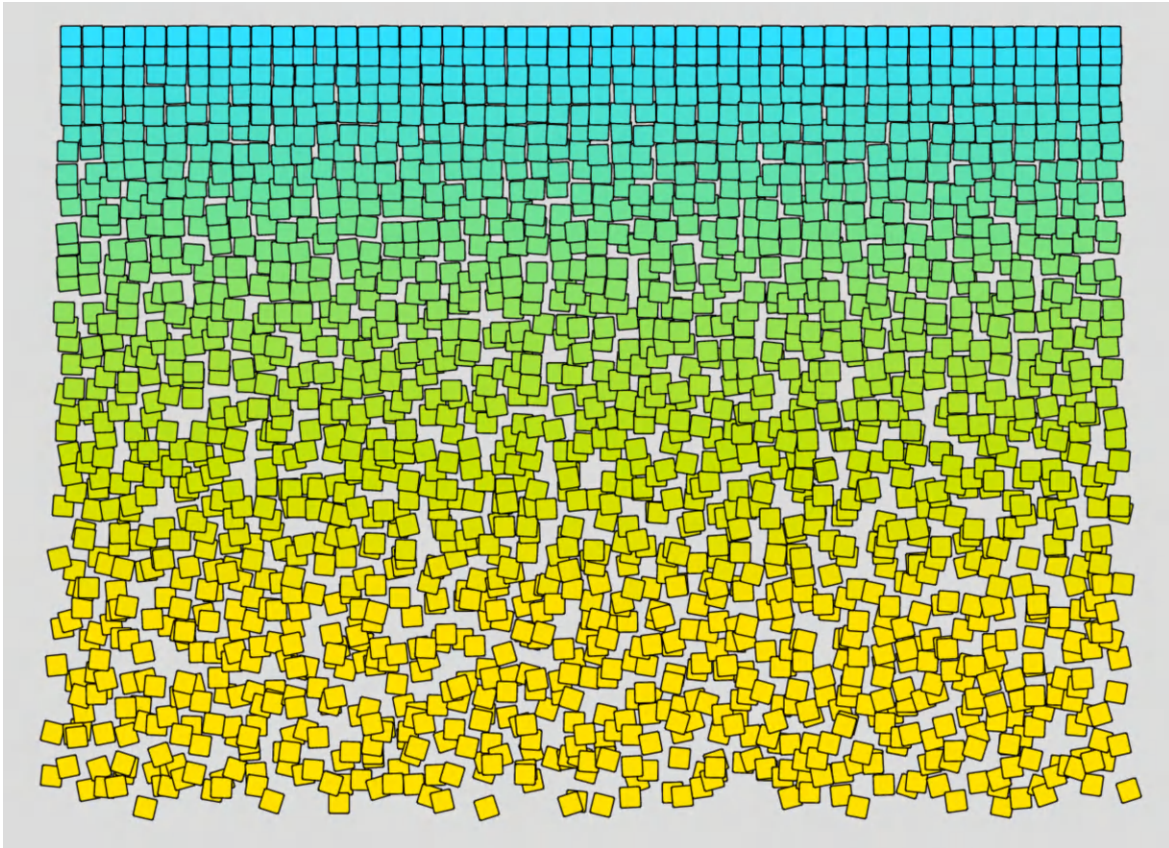
In the following Remixing *Mauerfall I&II* I have - in addition to the random colouring of the squares - limited the random changes to a partial area of the 'wall' in each case. The random change also affects the transparency of the coloured squares.



*Remixing Schotter: Mauerfall I,
JW 2017*



*Remixing Schotter: Mauerfall II,
JW 2017*

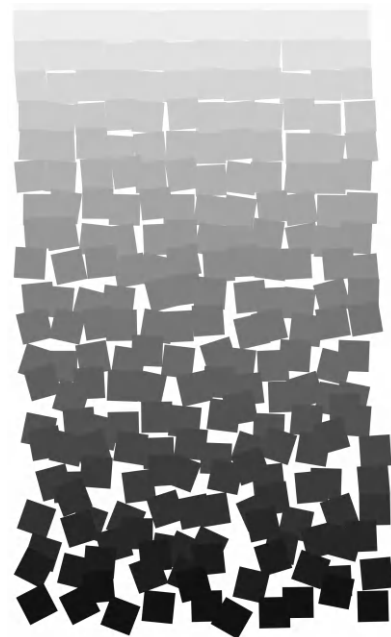


Remixing Schotter: Breitwandformat, JW 2017

In the case of my picture *Schotter Breitwandformat*, the remixing concerns the picture format and the use of a colour gradient.

The computer science professor and Generative Artist Massimo Franceschet (pseudonym hex6c) has carried out a *Generative Art Recoded* (GARP) project (hex6c, 2020), with which he aims to preserve selected works of early Computer Art by reprogramming them.

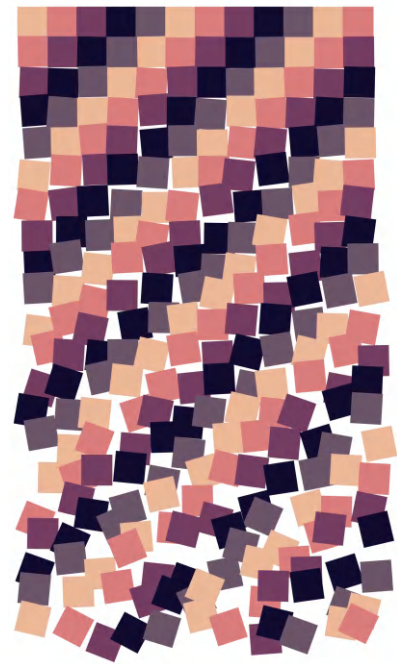
In addition to *Schotter*, he also applies his typical approach (see the Recoding chapter) to *Computer Composition With Lines* by Michael Noll, *Hommage à Paul Klee* by Frieder Nake and *(Dés)Ordres* by Vera Molnar. In *Take I*, he uses coloured (black) squares and a transparency that decreases from top to bottom (very similar to Remixing *Loses Mauerwerk* by



hex6c: Schotter - Take I, 2020

me and *Remixing Schotter* by Ilyá Belsky. In *Take II* he uses five colours without changing the transparency.

In the images *Take III* and *Take IV*, hex6c introduces sequences of letters and numbers to replace the squares (or polygons). This replacement of the picture elements - as in the pictures by Tobias Toft or Daniel Catt shown above - is still orientated towards the *Schotter* model in the picture structure, but allows completely new picture compositions.



hex6c: Schotter - Take II, 2020



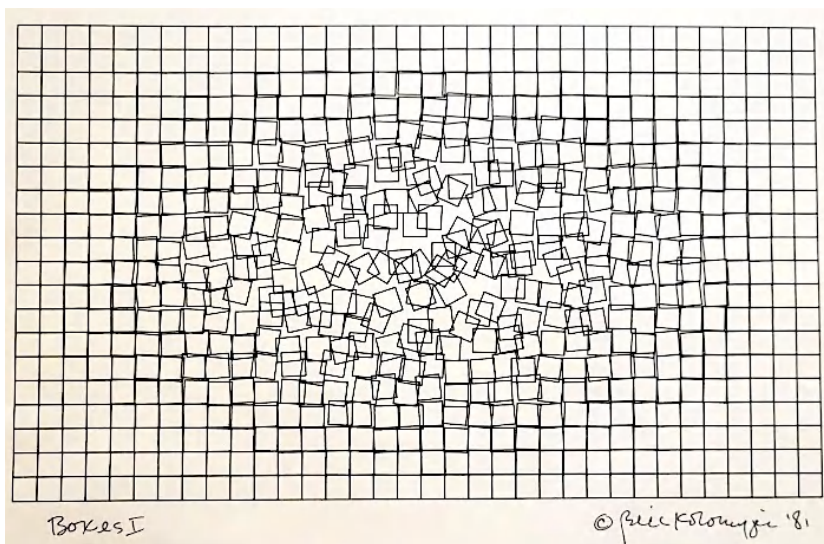
hex6c: Schotter - Take III, 2020



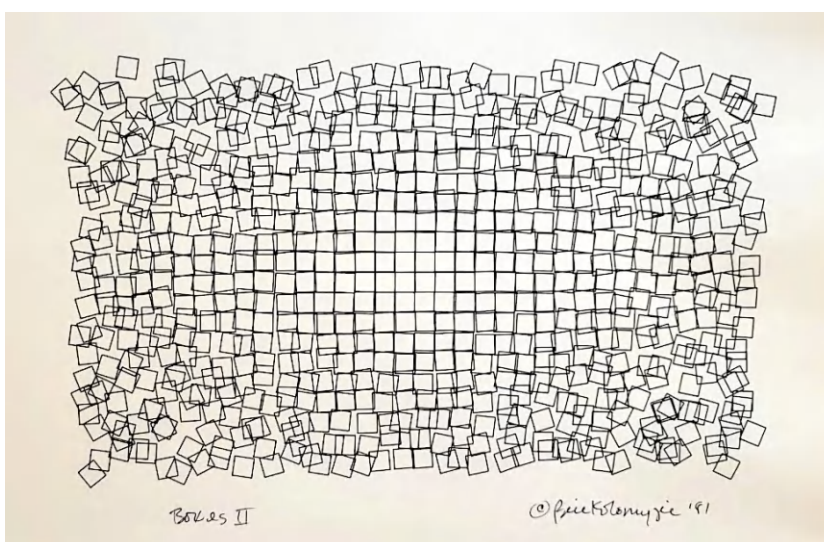
hex6c: Schotter - Take IV, 2020

An early and still very active representative of Generative Art is William J. Kolomyjec. He was inspired by *Schotter* to create his picture series *Boxes I & II*. These pictures were created as early as 1973 and can be regarded as the first examples of a Remixing of early computer art. Bill Kolomyjec is probably one of the first to explicitly refer to these models. He explains the reference as follows (Kolomyjec, 1975):

„It was not uncommon for these early computer image makers to use the computer to duplicate a basic theme by a traditional artist. A classic study is by A. Michael Noll, Computer Compositions with Lines 1965, after Piet Mondrian's Composition with Lines. Boxes in turn is my interpretation of a work by George Nees entitled Gravel Stones (Schotter). I feel that it is appropriate to study works of other artists, especially traditional artists and those who established Computer Art in the fine art realm.“



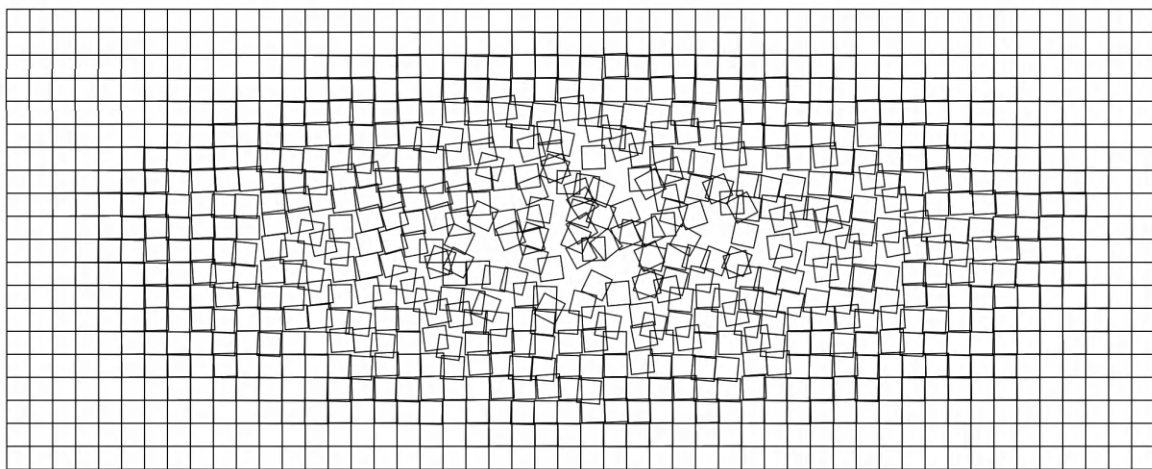
Kolomyjec: Boxes I. 1981



Kolomyjec: Boxes II. 1981

The basic structure of *Boxes* is a regular grid of square tiles. This is close to the initial situation with *Schotter*. In the case of *Boxes I*, however, the random rotations and displacements increase the closer the individual squares are to the centre of the grid. In the case of *Boxes II*, on the other hand, they increase the further away the individual squares are from the centre of the grid²⁰.

Interestingly, the Recoding of this Remixing already exists. As part of the *ReCode Project*, Hendrik Kleper has modelled his work on various classics of Computer Art (John Whitney, Vera Molnar, Roger Coqart and others). His Recoding of *Boxes I* allows the reproduction of images close to the original²¹. The more recent pictures in square format can also be easily created in this way.



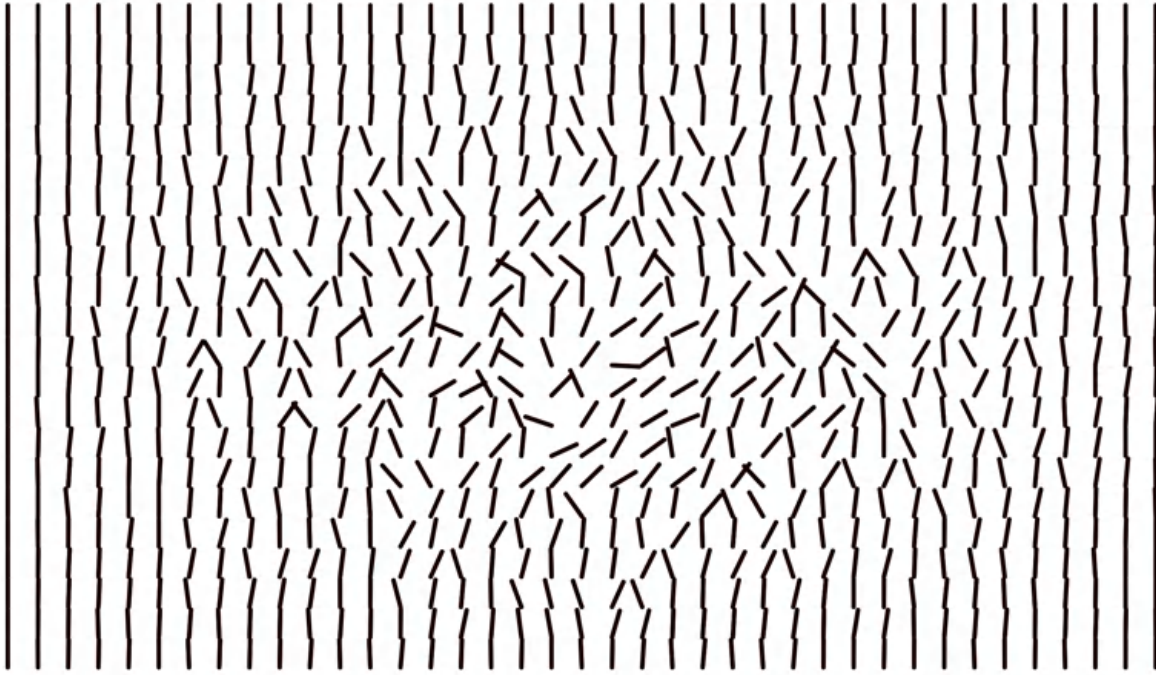
Kleper: Recoding Boxes I, 2012. JW, 2025

My own involvement with *Boxes* has led to the recoding of the *Boxes I* and *Boxes II* versions²². This makes it possible to create images close to Kolomyjec's originals. I have applied various forms of Remixing to these versions, the Remixing of Remixing, so to speak. If, for example, polygons with any number of corner points are used - as already shown with *Schotter* - different image impressions can again be created with *Boxes*.

²⁰ Bill Kolomyjec offers current versions of *Boxes I* & *II* (now in square picture format): : <https://objkt.com/collections/fxhash/projects/18483>

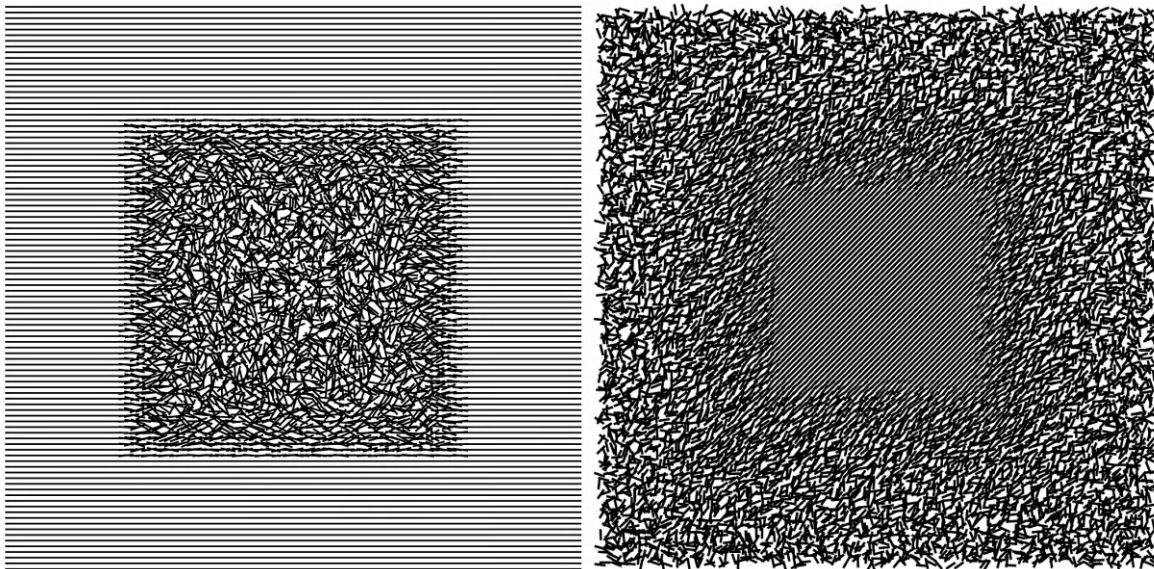
²¹ Its program runs directly in the browser. The program code can be viewed and characteristic values and the code itself can be changed: https://editor.p5js.org/hendrikleper/sketches/OD_JZNjqw

²² You can try out my program for yourself: Boxes I [[Program 3](#)]



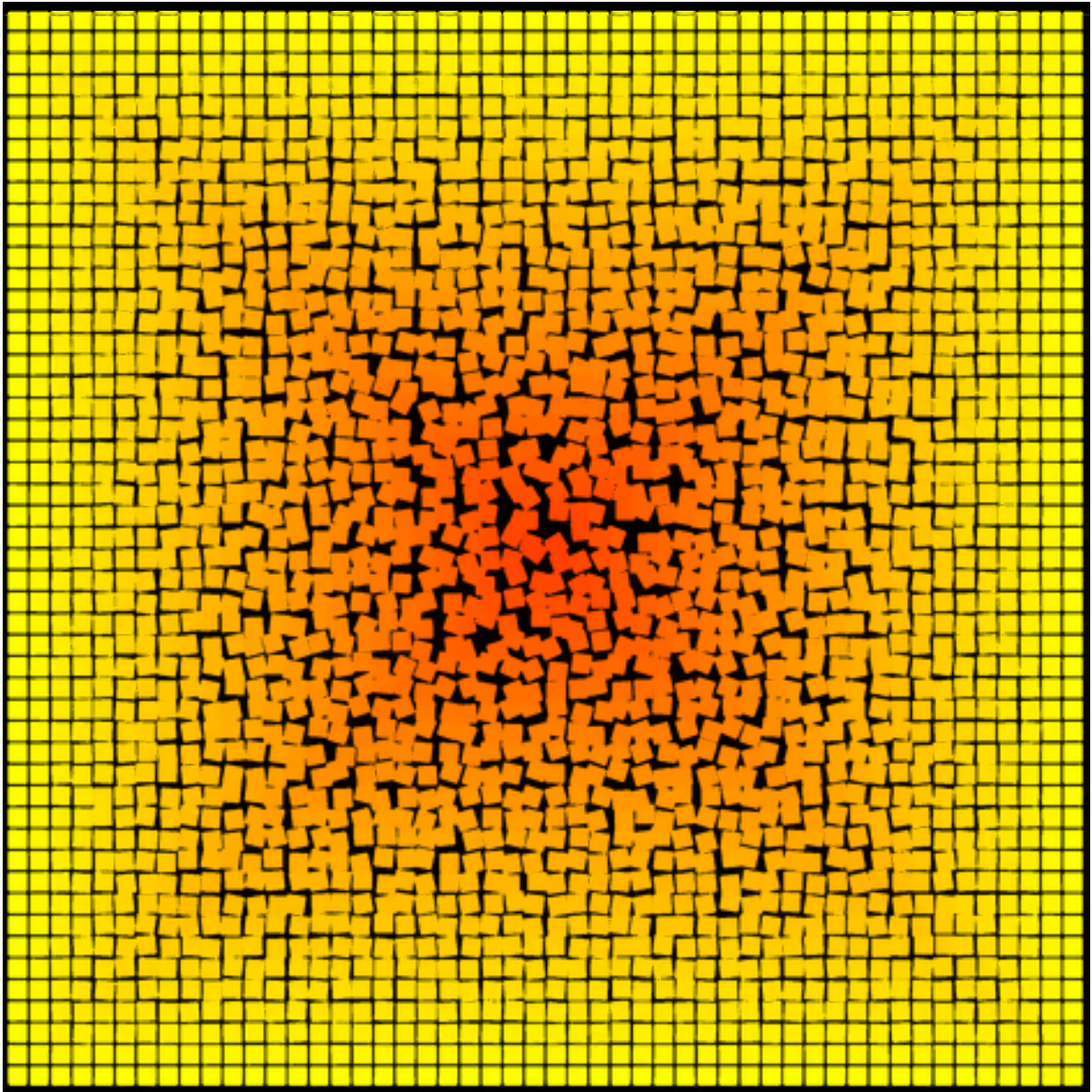
Remixing Boxes I, 2 corner points. JW 2017

If the random deviations calculated from the inside are suppressed from a certain position in the polygons, *Boxes I* produces a flat ‘edge plateau’. In the opposite case, i.e. if the random deviations are suppressed from a certain position calculated from the outside, a ‘central plateau’ is created for *Boxes II*.



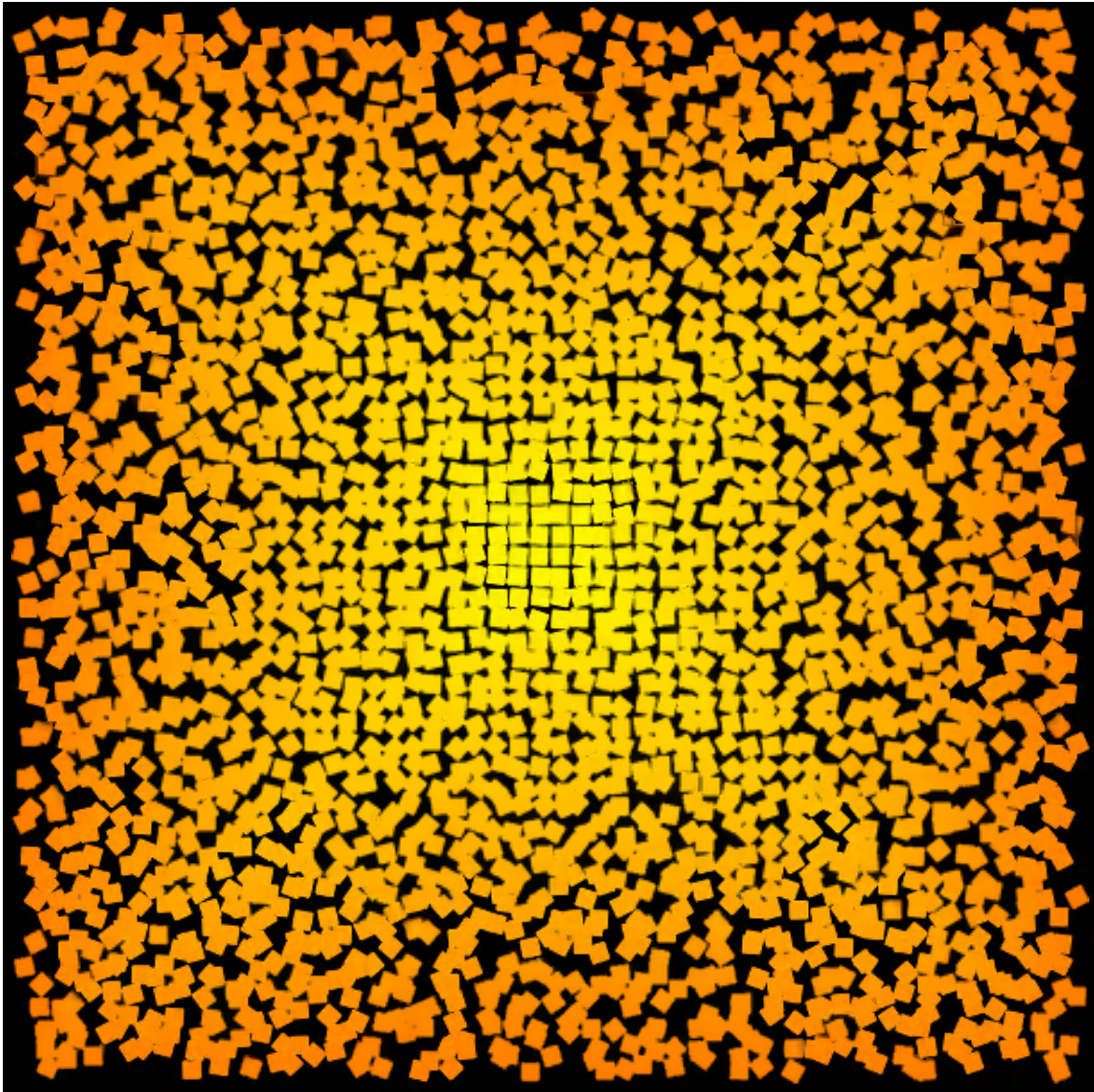
Remixing Boxes I & II: Plateau on the edge (left) or in the centre (right), JW 2017

If the polygons are coloured, an almost three-dimensional impression can be created. This can be enhanced by a systematic colour shift, coupled to the position of the squares.²³



Remixing Boxes I: Plateau on the edge with colour shift, JW 2017

²³ You can test my program yourself: Boxes I farbig [[Program 4](#)]



Remixing Boxes II: Centre platform with colour shift, JW 2017

Schotter: interactive & animated ...

Media Art refers to all artistic work that makes use of media. As we are particularly interested in the use of digital media and thus the influence of digital technologies on the production and reception of art when Recoding and Remixing *Schotter*, it makes sense to speak of *Art of the Digital Age*, as Bruce Wands (2006) did.

Compared to the 1960s, the new developments associated with this cannot be separated from the drastic improvements in the tools and output media now available. As we have already seen from a number of examples, it is now quite natural to use colour representations in high resolution. This makes it possible to realise significant variants and extensions of early Computer Art as well as new applications.

The output of program results is no longer limited to paper. Large monitors allow high-resolution, static and animated displays, even in colour²⁴.

In particular, the interactive modification of parameters by viewers and the evaluation of user data using sensors have also become possible. The direct reaction of the programs to such data generated by the viewers increasingly blurs the distinction between the technical processing of algorithms and the individual creation of a work by the artists themselves and/or the viewers.

Frieder Nake - one of the aforementioned pioneers of Computer Art - also sees the future of Algorithmic Art as being characterised by this (Nake, 2014):

In any case, the image must be in motion: whether as a dynamic installation or through interaction with the environment, it doesn't matter. The static image is not up to the standard of Algorithmic Art. [...] I believe that the future lies in the further exploration of the fundamental problem - form and colour, which can only occur together.' (Translation JW)

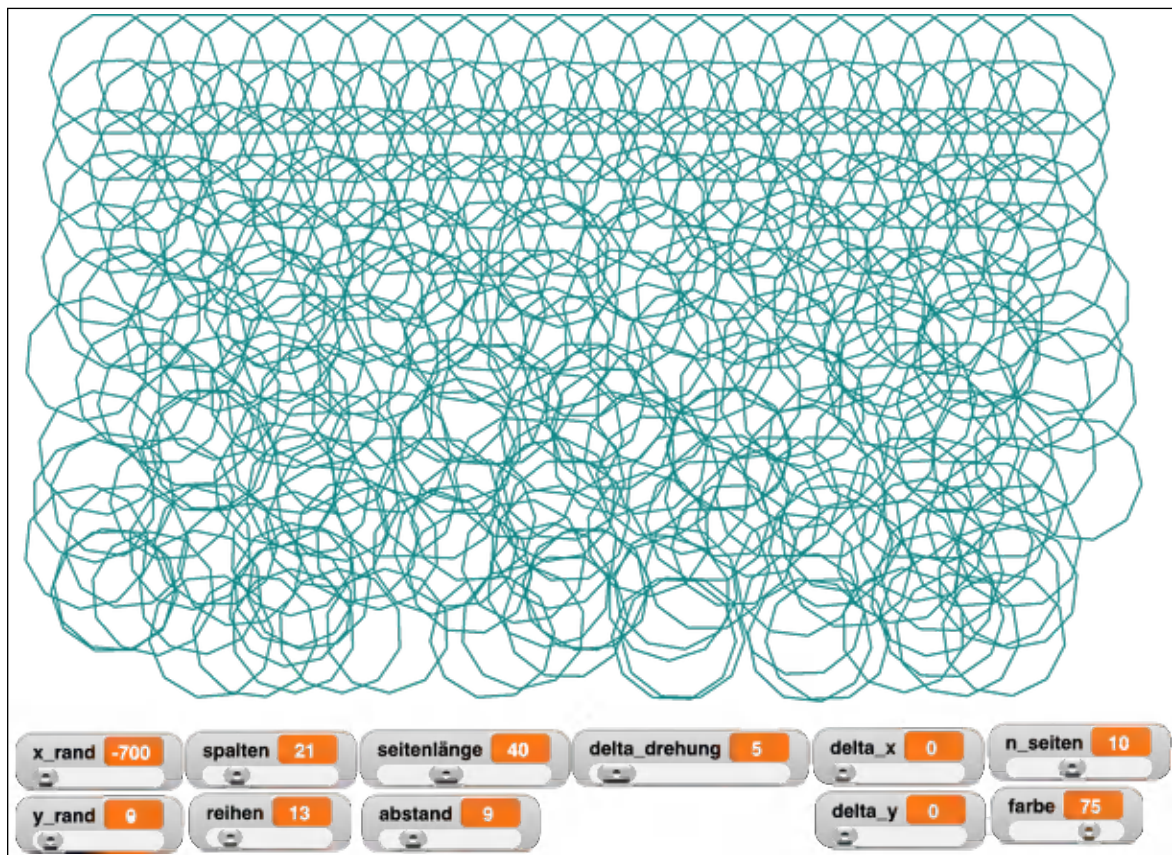
This expanded perspective can also be demonstrated by current forms of Remixing *Schotter*.

While working on an interactive version of *Schotter*, I became aware of a version by the Swiss engineer Antoine Beyeler. His replica of *Schotter* allows the following parameters to be changed: number of columns and rows, distance between squares, side length of squares, random rotation angles and horizontal/vertical

²⁴ Entire house fronts can even serve as an output medium, whether as projection surfaces or with program-controlled colour windows, as the *Arbolet frontage* project impressively demonstrated. You can find a short video documentation here: <https://vimeo.com/205139207>

displacements as well as line thickness. All changes are implemented immediately, which is why continuous changes with the sliders create quasi-animated changes²⁵.

My version of *Schotter total interaktiv* offers these and some additional options. With my project, I would also like to illustrate how the principle of total parameterisation can be used to create a very flexible program that offers users a wide range of experimentation options²⁶. The same parameters as in Beyeler's version can be changed using sliders. This also applies to the horizontal and vertical starting point, the number of corner points (i.e. the type of polygons) and the colour of the polygons.



As an alternative to the sliders, there are a number of individual switches that can be used to increase or decrease the parameters precisely by the value 1 or -1 respectively by clicking on them. There are also individual switches for other options relating to the colours of the polygons:

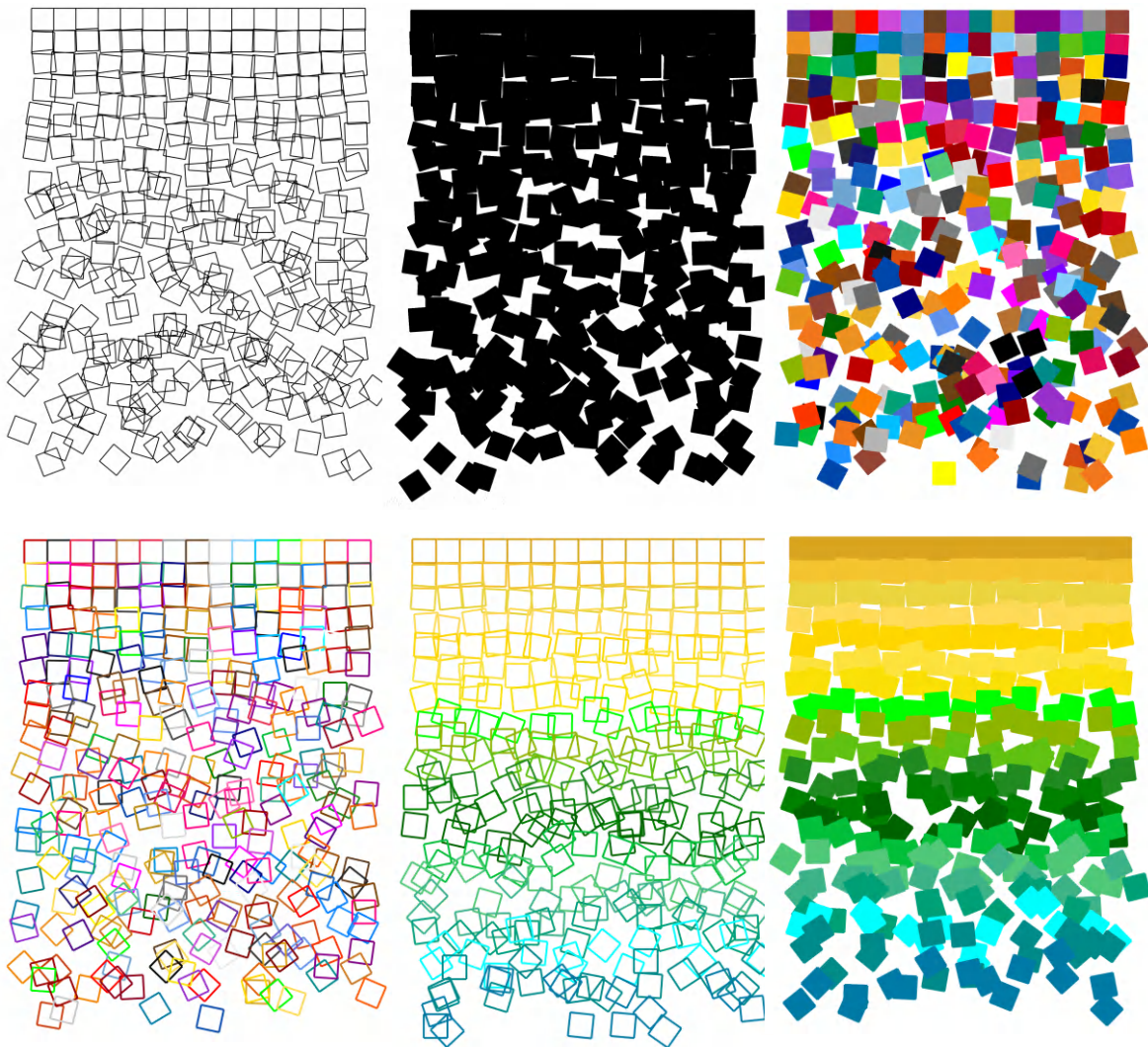
²⁵ You can test Beyeler's interactive version directly in your browser: <https://whisk.rs/>

²⁶ You can also work with this version directly in the browser: Schotter total interaktiv [[Program 5](#)]

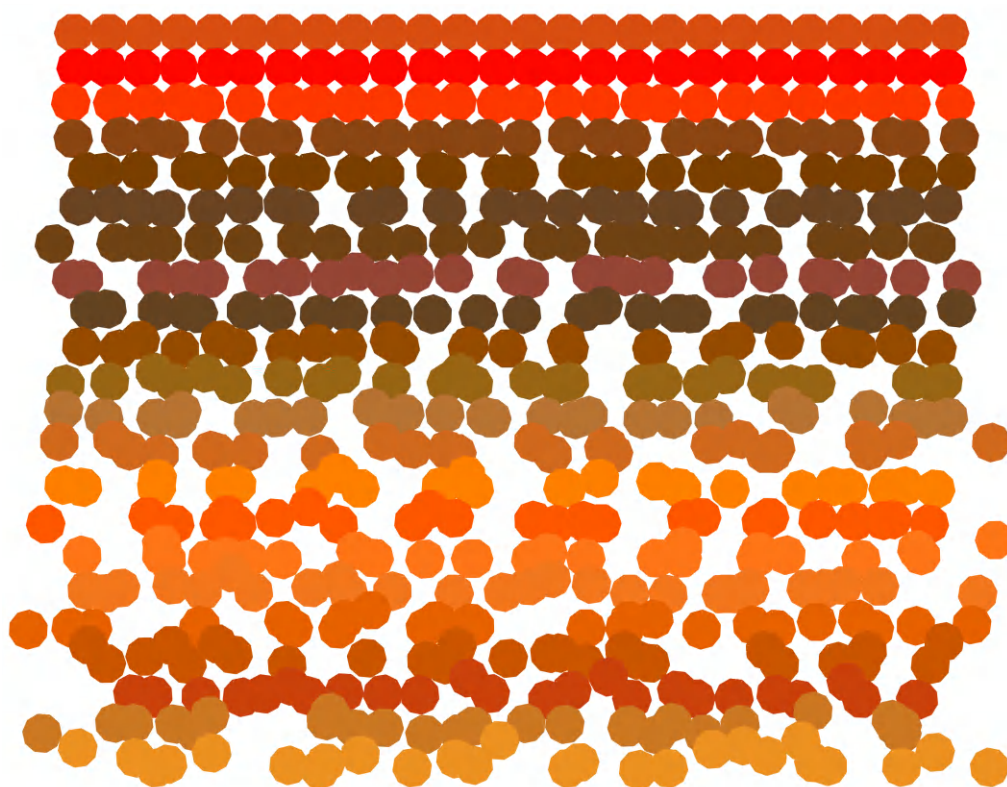
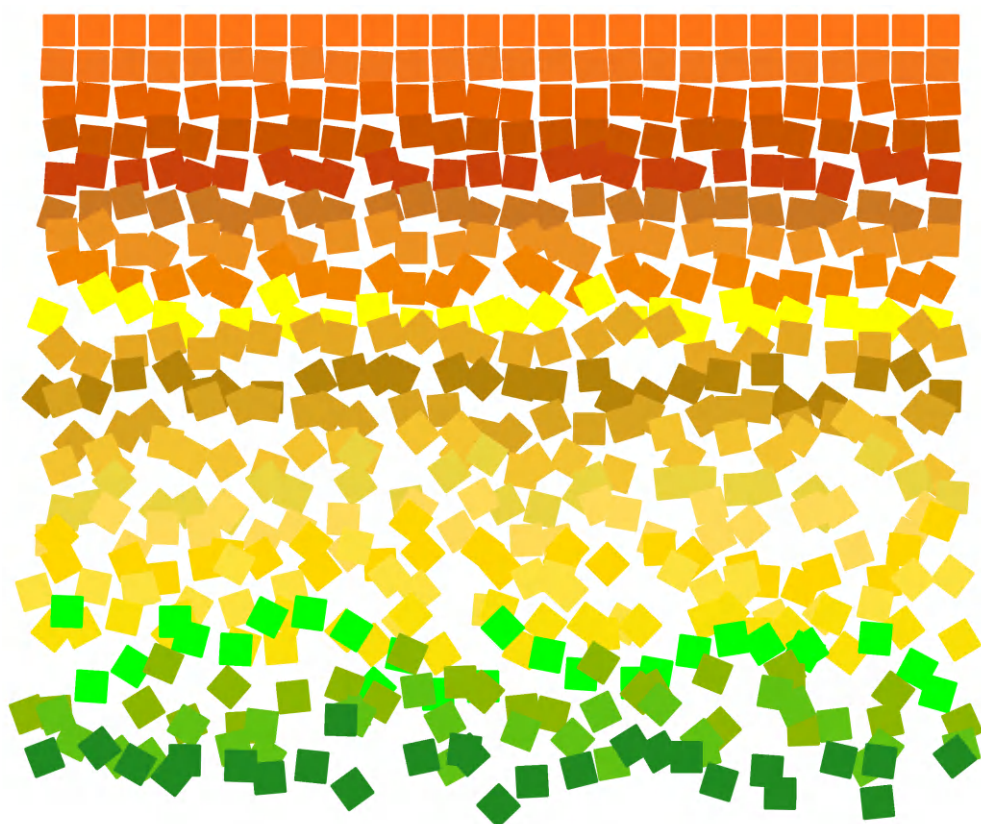
- Change pen thickness by the value 1/-1
- Set polygons to filled/not filled
- Set colouring to multicoloured/single-coloured
- Set colour to same colour per row
- Change colour value by the value 1/-1
- Set pen colour randomly to a value between 0 and 99



The picture series shows some examples of the use of colours:



Examples from top left to bottom right: Original black unfilled, black filled, multicolour filled, multicolour not filled, colour per row not filled, colour per row filled



Schotter total interaktiv: 26 × 22, colour filled per row, four corner points (top), ten corner points (bottom), JW 2025

The interactive versions of *Schotter* shown already have a rudimentary animation, made possible by the almost instantaneous reaction to the viewer's input. Real animation, on the other hand, is based on the viewing experience we know from films. Films consist of individual frames, of which professional films typically show 24 frames per second in order to create the impression of realistic movement. Computer animations are films or corresponding image sequences that run in real time and are generated and displayed entirely on the computer. In Computer Art, too, such computer films have been part of the repertoire of realised forms from the very beginning.

Charles Csuri, who helped shape Computer Art and Computer Animation from both an artistic and technical perspective, is worthy of mention. It is remarkable that for Csuri, even back then, the computer film was not intended to be a finished, presentable film, but one that was created in interaction with the users: *'Viewers can also experience themselves as artists to a certain extent when they sit down at the 'steering wheel' of the computer. Even on the basis of a given context, the possibilities are almost infinite.'* (Csuri, 1978, p.62, Translation JW).

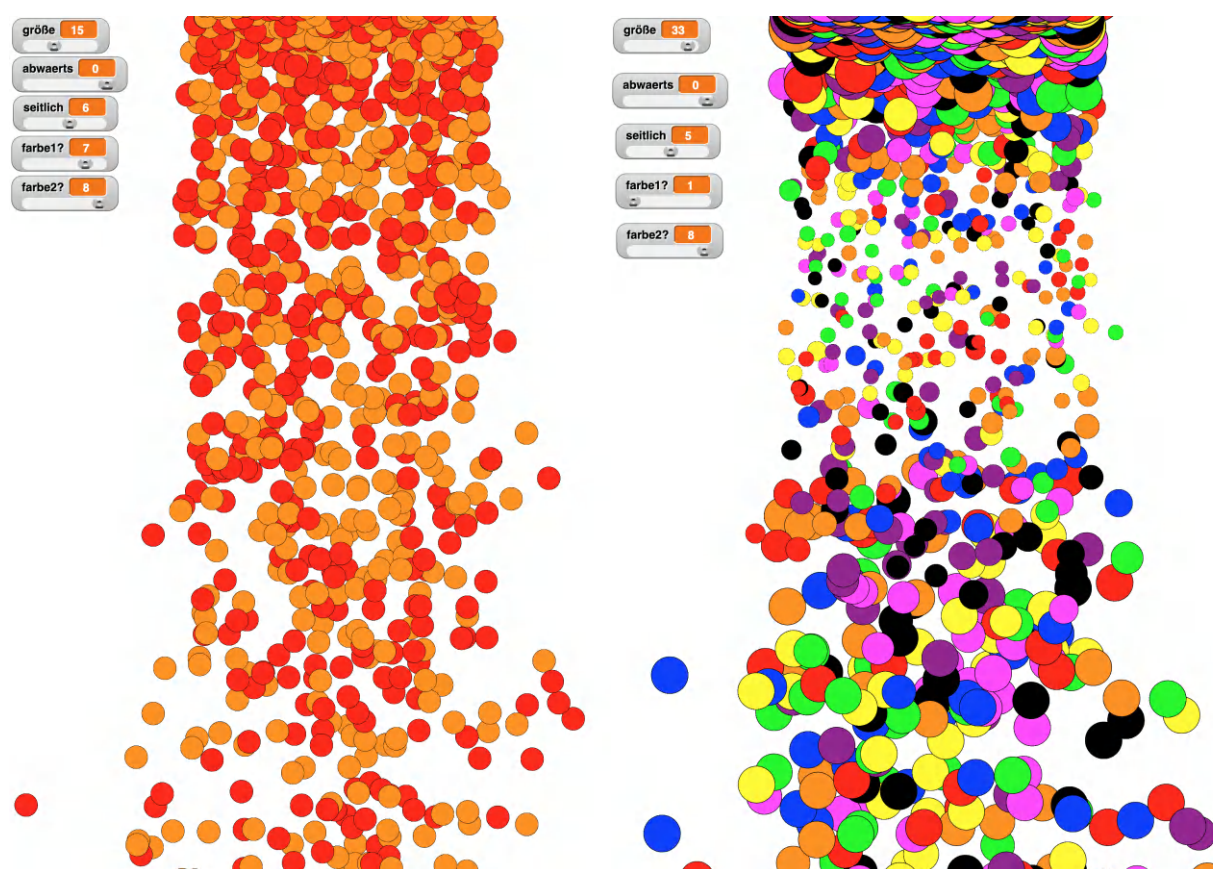
In my first recoding of *Schotter*, I had already turned the original image direction upside down (cascading squares) because I wanted to create the impression of gravel being poured from a lorry. It therefore makes sense to reinforce this impression with animation, based on the original version with squares²⁷. The image below illustrates this principle (naturally only as a static sequence of four intermediate stages). Stones lying next to each other move from top to bottom. They disappear at the bottom of the screen, but new stones appear from above so that the animation runs without interruption.



²⁷ You can test this version directly in the browser: Schotter Animation [[Program 6](#)]

The impression of falling stones is further enhanced if coloured polygons are used, as shown in the previous example. The animation is enhanced by the element of interactivity, as the viewer has the option of influencing the course of the animation directly during the sequence. In the example shown below, this is made possible by the **abwärts**, **seitlich** and **drehung** sliders. The program constantly accesses the updated values and therefore reacts immediately to the viewer's input²⁸.

The final version of the program (intended for the presentation with an interactive station) uses prefabricated coloured discs²⁹. Their **größe** can be changed using sliders, and the colour can be changed continuously using **farbe1?** and **farbe2?** (for single-colour discs, the same value must be selected for **farbe1?** and **farbe2?**).



*Schotter-Scheiben interaktiv: two colours (left), eight colours, **größe** changed several times (right)*

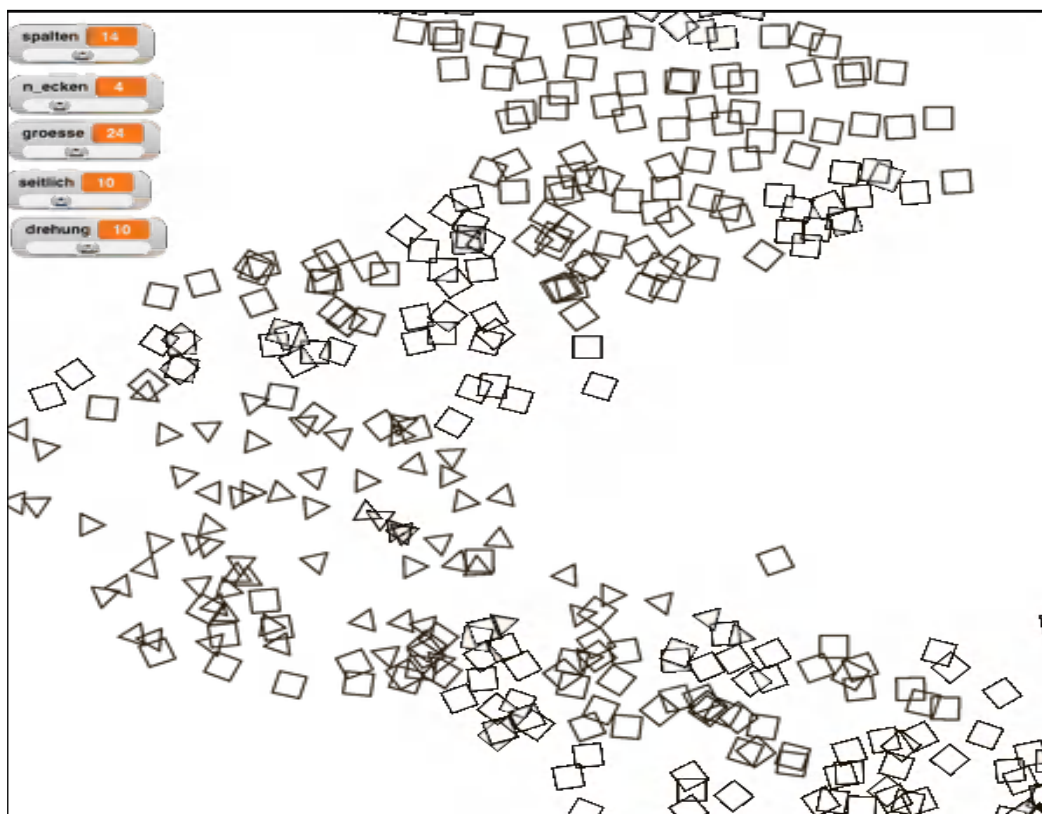
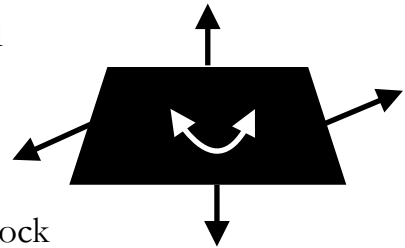
²⁸ We recommend enlarging the display of the variables so that viewers can safely hit and move the sliders with a mouse or finger on a touchscreen or tablet.

²⁹ You can test this version directly in the browser: Schotter Animation Farbscheiben [[Program 7](#)]

Here too, the effect of the animated display can only be guessed at in the static image.

An even more intuitive form of interaction is possible with smartphones or tablets if their inclination sensors are analysed. The following control is possible in such an extended form of the interactive version of *Schotter*³⁰:

- Tilting to the right or left changes the direction of fall towards the right or left edge of the screen.
- Tilting downwards accelerates the fall speed, tilting upwards slows it down or reverses the initial direction (it is best to activate the tablet's orientation lock for this).
- Turning the tablet like a steering wheel changes the number of corners of the polygons.

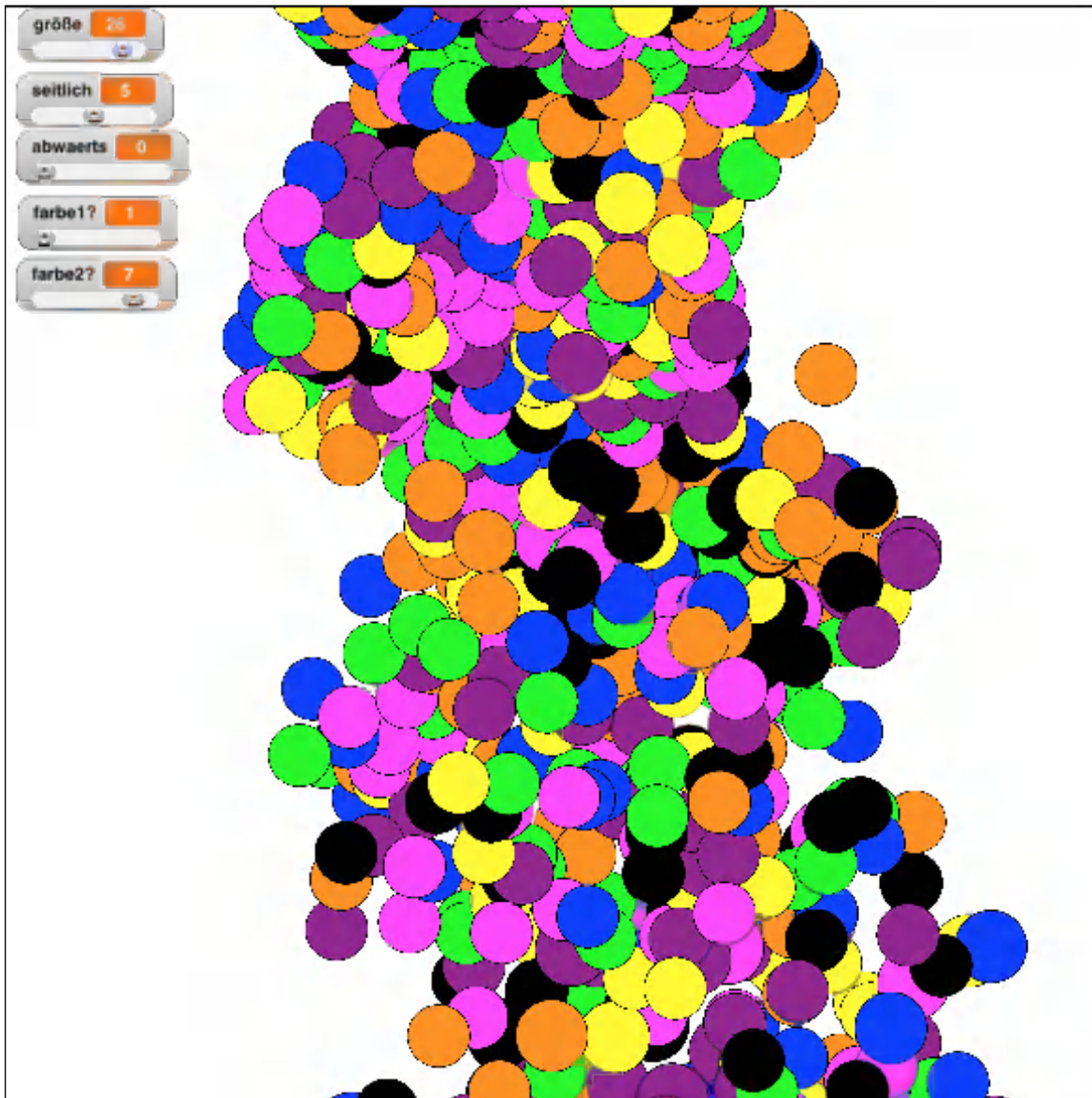


Schotter with sensors: effects through tilting and turning

³⁰ You can also test this version directly in the browser (only with tablet or smartphone!): Schotter Polygone Sensoren [[Program 8](#)]

It is still possible to change the other characteristic values using sliders.

A corresponding extension of the interactive *Schotter-Scheiben* version works according to the same principle. However, the parameter **seitlich** here is reduced and enlarged laterally by rotating the device³¹.



Schotter-discs with sensors: effects through tilting and turning

³¹ You can also test this version directly in the browser (only with tablet or smartphone!): Schotter Farbscheiben Sensoren [\[Program 9\]](#)

Think the Image, Don't Make It!

With this short retrospective of 60 years of Computer Art using the example of *Schotter*, I hope I have been able to show how important it is in all forms of Algorithmic and Generative Art that we can *think images, not have to make them*. To this end, we should understand what role machines play in the process of creating images and whether and how the role of the artist changes as a result.

In the example of *Schotter*, we have seen how the analysis of what at first glance appears to be a fairly simple image structure can prove to be extremely stimulating. The Recoding and Remixing of early Computer Art could be continued in a comparable way in numerous works from this phase. The low-threshold access to tools that can be used for this purpose, as they are available today, perhaps invites you to participate.

At an exhibition of my own works of Computer Art, I was asked, among other things, what I had done myself with the exhibits. For me, my role is clear. I didn't actually pick up a pencil or paintbrush. But that doesn't change the fact that I myself, and not the machine, am still responsible for the characteristics of the images created by my algorithms - as representatives of the respective class of all conceivably possible images. In this respect, this work differs fundamentally from *Generative AI*, which is currently attracting a great deal of attention both inside and outside the art world. But that is another topic ...

Joachim Wedekind

Literature

- Csuri, C. (1978). Real Time Picture Animation. In: IBM Deutschland GmbH (Hrsg.): Computerkunst. p. 62 f. Stuttgart: IBM Deutschland GmbH.
- Franke, H. W. (1971). Computergraphik Computerkunst. München: Bruckmann.
- Groos, U. & Froitzheim, E.-M. (Hrsg.) (2016). [un]erwartet. Die Kunst des Zufalls. Köln: Wienand Verlag.
- hex6c (2020). Generative Art Recoded. Medium 13. April. <https://hex6c.medium.com/generative-art-recoded-675c3eabb2f2>
- Hunter, Z. (2024-1). Schotter - Georg Nees - Part 1: Introduction. <https://zellyn.com/2024/06/schotter-1/>
- Hunter, Z. (2024-2). Schotter - Georg Nees - Part 2: Investigation. <https://zellyn.com/2024/06/schotter-2/>
- Kolomyjec, W.J. (1975). The Appeal of Computer Graphics. In: R. Leavitt (Ed.): Artist and Computer. pp. 45–51, New York: Harmony Books. Download unter <http://www.atariarchives.org/artist/sec15.php>
- Nake, F. (1995). Künstliche Kunst im Strom der Zeit. Vortrag Berliner Sommer-Uni '95. Download unter <http://radicalart.info/AlgorithmicArt/Nake95.html>
- Nake, F. (2014). „No Message Whatsoever“ (Interview). HPImgzn Ausgabe 15, pp. 11-13. Potsdam: HPI. Download unter: https://hpi.de/fileadmin/user_upload/hpi/dokumente/hpi_mgzn/HPImgzn_Ausgabe15.pdf
- Nake, F. (2016). The Disappearing Masterpiece Digital Image & Algorithmic Revolution. In: M. Verdicchio et al. (Ed.): xCoAx 2016 Proceedings, pp. 12-27. Bergamo. <http://2016.xcoax.org/xcoax2016.pdf>
- Nake, F. (2021). $I(\chi, \chi') = g(\chi, \chi') [e(\chi, \chi') + \int p(\chi, \chi', \chi'') I(\chi', \chi'') dx]$. In Diestelmeyer, Ehrmanntraut & Müller (Hg.): Algorithmen & Zeichen. Beiträge von Frieder Nake zur Gegenwart des Computers. Berlin: Kulturverlag Kadmos.
- Nake, F. & Grabowski, S. (2017). Think the Image, Don't Make It! On Algorithmic Thinking, Art Education, and Re-Coding. Journal of Science and Technology of the Arts, Volume 9, No. 3. <https://revistas.ucp.pt/index.php/jsta/article/view/7292/7292>
- Nees, G. (1969). Generative Computergraphik. A reprint with introductory texts by Herrmann and Nees (2006) is available as eText: <http://www.diaphanes.net/titel/generative-computergraphik-1948>
- Nees, G. (1995). Formel, Farbe, Form. Computerästhetik für Medien und Design. Heidelberg: Springer.
- Nees, G. (2005). Künstliche Kunst. Die Anfänge. Ausstellungsbroschüre (unpaginiert). Bremen: Kunsthalle.
- Plaxco, J. (2006). Georg Nees, Processing, and a Schotter Tutorial. https://www.artsnova.com/Nees_Schotter_Tutorial.html
- Rauch, H. & Wedekind, J. (1989). Schildkrötengrafik. Einfaches Programmieren von Grafiken. Lehren und Lernen mit dem Computer 13. Tübingen: DIFP.
- Schneeberger, R. (2013). Programmierte Fälschungen Digitaler Kunst als Museumsprojekt? Telepolis (Online-Journal). Haar: Heise Medien.

- Sol Le Witt (1967). Paragraphs on conceptual art. Artforum 5, 10 (June 1967), pp. 79–83.
- Stalder, F. (2009). Neun Thesen zur Remix-Kultur. https://irights.info/wp-content/uploads/fileadmin/texte/material/Stalder_Remixing.pdf
- Stein, H. (1984). LOGO - Grafik, Sprache, Mathematik. Haar Bei München: Markt & Technik Verlag.
- Toft, T. (2014). An intro to Pen Plotters. Medium <https://medium.com/quarterstudio/an-intro-to-pen-plotters-29b6bd4327ba>
- Wands, B. (2006). Art of the Digital Age. London: Thames & Hudson.
- Wedekind, J. (2018). Codierte Kunst. Kunst Programmieren mit Snap! Tübingen: Selbstverlag. <http://digitalart.joachim-wedekind.de/my-books/>
- Zeilinger, M. (2018). Plotting Critical Research-Practice in Digital Art. In: Golding (Ed.): Parsing Digital - Conversations in digital art by practitioners and curators, pp. 22-37. London: Austrian Cultural Forum. https://www.academia.edu/39607548/Plotting_Critical_Research_Practice_in_Digital_Art

Image Credits

- p. 3: Schotter: Nees (1969), Bild 38, S. 242
- p. 5: Georg Nees: Foto Alexander Kempkens, 1986; https://commons.wikimedia.org/wiki/File:Georg_Nees-1- BID_1986.jpg
- p. 6: ZUSE Graphomat Z 64, Zeichentisch & Steuerelektronikschrank; Leihgabe des Deutschen Technikmuseums Berlin an DAM Galerie Berlin, Ausstellung Aesthetica, 2015; Fotos: Inge Wedekind
- p. 12: Schotter: Victoria and Albert Museum (Kensington, GB); <https://collections.vam.ac.uk/item/O221321/schotter-print-nees-georg/>
- p. 12: Schotter: The Anne and Michael Spalter Digital Art Collection; <https://spalterdigital.com/artworks/schotter-gravel-stones/>
- p. 37: Schotter Ilyá Belsky, <https://observablehq.com/@oluckyman/georg-nees-schotter?collection=@oluckyman/generative-art>
- p. 38: Jim Plaxco: Cubic Disarray: Turbulence, <https://www.artsnova.com/art-Cubic-Disarray-Turbulence.html>
- p. 38: Jim Plaxco: Cubic Disarray: Bisection, <https://www.artsnova.com/art-Cubic-Disarray-Bisection.html>
- p. 40: hex6c: Schotter - Take I, <https://superrare.com/artwork/eth/0xb932a70a57673d89f4acffbe830e8ed7f75fb9e0/9453>
- p. 41: hex6c: Schotter - Take II, <https://superrare.com/artwork/eth/0xb932a70a57673d89f4acffbe830e8ed7f75fb9e0/9454>
- p. 41: hex6c: Schotter - Take III, <https://superrare.com/artwork/eth/0xb932a70a57673d89f4acffbe830e8ed7f75fb9e0/9487>
- p. 41: hex6c: Schotter - Take IV, <https://superrare.com/artwork/eth/0xb932a70a57673d89f4acffbe830e8ed7f75fb9e0/9488>
- p. 42: Boxes I & II: <https://www.drbillkolomyjec.com/hard-copy/plotter-prints>

My *Schotter*-Programs

p. 13 [Program 1] Hommage to the Square: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Hommage%20To%20The%20Square&editMode&noRun>

p. 17 [Program 2] Schotter Nees Original: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Schotter%20Nees%20Original%20recoded>

p. 43 [Program 3] Boxes I: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Hommage%20Kolomyjec%20Boxes%201>

p. 45 [Program 4] Boxes I farbig: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Hommage%20Kolomyjec%20Boxes%201%20mit%20Randebene%20farbig>

p. 48 [Program 5] Schotter total interaktiv: https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=schotter_total_interaktiv

p. 51 [Program 6] Schotter Animation: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Schotter%20Animation>

p. 52 [Program 7] Schotter Animation Farbscheiben: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Schotter%20Animation%20Farbscheiben>

p. 53 [Program 8] Schotter Polygone Sensoren: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Schotter%20Polygone%20Sensoren>

p. 54 [Program 9] Schotter Farbscheiben Sensoren: <https://snap.berkeley.edu/snap/snap.html#present:Username=jowede&ProjectName=Schotter%20Farbscheiben%20Sensoren>

About the person

Joachim Wedekind was a educational technologist and media didactician. As a latecomer and career changer, he now explores the role of painting machines with close reference to early Computer Art. This was the focus of several exhibitions, which are also documented online:

<http://digitalart.joachim-wedekind.de/exhibitions/>

He has summarized his experiences in programming Generative Art in the book *Codierte Kunst Kunst - Programmieren mit Snap!* The result is a mixture of illustrated book and programming guide:

<http://digitalart.joachim-wedekind.de/ueber-das-buch/>

