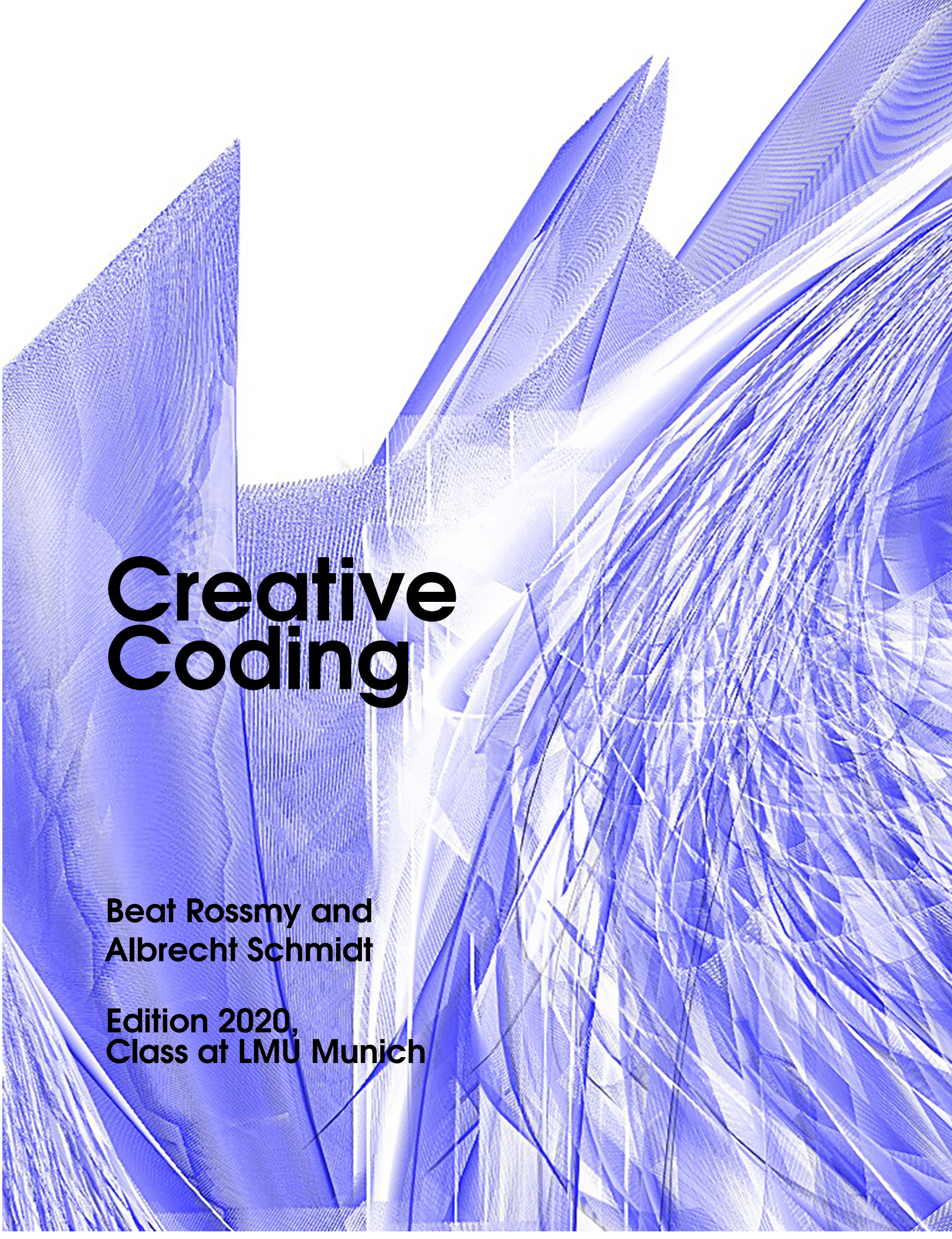

# Creative Coding

Beat Rossmy and
Albrecht Schmidt

Edition 2020,
Class at LMU Munich

# Creative Coding

## Implementing an ∞-Exhibition

BEAT ROSSMY AND ALBRECHT SCHMIDT

**EDITION 2020**

**THIS BOOK IS WORK IN PROGRESS AND BASED ON A CLASS AT LMU MUNICH**

**Cover Design by Maya Czupor Copyright © 2019**

# Table of Contents

# 1. Programming and Creative Coding

## 1.1 Concept of this Book

The concept of the book is to get people excited for programming artistic pieces. In this book we teach basic concepts that make up building blocks for interactive and potentially artistic programs. The focus is to help the reader to create creative software and to learn how to program such software. The artistic part is up to the reader, but we hope to give good starting points, where readers can bring in their creativity.

In parts we suggest critical reflection on creative coding and programming art pieces. In the lecture at LMU this is done as discussions, that are not captured in the book. We recommend readers to show their ideas and programs to other and discuss and defend it. This aids reflection and learning, similar to what we do in the course.

Besides the programming concepts and the implementation techniques, we discuss different topics, including:

- How aesthetics can be programmed?
- Creativity vs. determined by algorithms?
- Creative information visualization vs. art?
- Evolutionary systems and creativity?
- ...

## 1.2 The Goal: an ∞-Exhibition

In the course we create different interactive software programs. With the selection of the topics we try to cover many different programming concepts and implementation techniques.

All the programs are designed to create changing new outputs, either after some time or based on user interaction. All programs together can then be set-up as an exhibition that is ever changing, which we call an ∞-Exhibition. If the screens with the software are set-up in a circle, where screens face outwards, a visitor of the exhibition can walk around an will always see new exhibits.

## 1.3 Target Group and Expected Skills

The course and book is designed for readers studying at the intersection of art, media, and technology. We expect that readers have an interest in interactive art and that they want to improve their abilities to create interactive installations.

We expect that readers have basic programming skills in processing. If readers have no programming skills at all, we suggest to do an introductory tutorial (e.g. `https://processing.org/tutorials/` ). For readers with programming experience in another languages, we expect that they can follow the course and read up on processing details as they go along.

# 2. Aesthetics and Information

The oeuvre of many famous modern painters includes very abstract paintings reduced to simple geometric forms, primary colours and clear pictorial structures. When you look at such paintings, many of you will think that they are easy to copy. At first sight you can grasp the colour palette, the organisation of the forms or simply the whole picture pattern. This allows us to visually separate a Mondrian from a Malevich. But can our understanding of these images be easily formalized and captured as an algorithm?

## 2.1 Programming a Random Mondrian

Following this idea, the first task of the book is to use Processing to recreate a mondrianesque image. Implement a program that creates a drawing that is inspired by Mondrian.

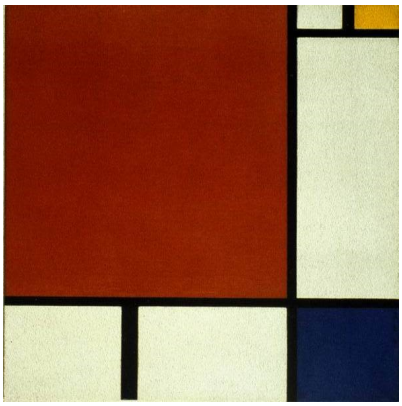

Figure 2.1: Example of a Mondrian Painting.

The program should create a new picture every 120 seconds or when the user presses a button on the keyboard.

### Useful Commands

```
void keyPressed();
random();
randomSeed();
second();
minute();
hour();
```

The following listing is a starting point for your project.

```
size(400,400); // draving canvas, coordinate systems
background(255); // background is white

fill(255, 0, 0); // color is red
rect(50, 100, 250, 100);

fill(0, 255, 0); // color is green
rect(250, 250, 100, 100);

fill(0, 0, 255); // color is blue
rect(50, 320, 150, 10);

fill(255, 255, 0); // color is yellow
rect(50, 340, 150, 6);

fill(255, 255, 255); // color is white
rect(50, 360, 150, 4);

fill(0, 0, 0); // color is black
rect(50, 380, 150, 3);
```

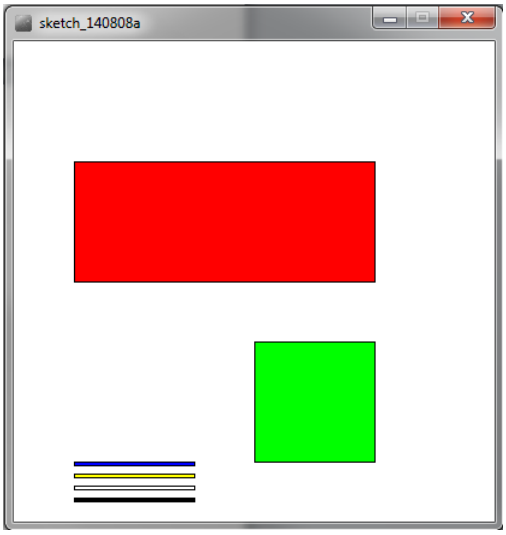


Figure 2.2: Resulting Image from the example code.

See `https://processing.org/reference/random_.html` for examples how to use the random function.

## 2.2 Random Creation and Aesthetics

Experiment with the parameters in the random creation. Explore the impact of parameters such as the size, dimension, line width, colors, and number of element on the aesthetics of the resulting picture.

Why are some pictures are aesthetically more pleasing than others? How can you change your algorithm to always create aesthetic drawings, while still having a great variety?

## 2.3 Encoding Information in Art

People have discussed the idea of informative art, see [Fer07; HS03; RSH00]. Can this utilitarian approach to creating visual displays based on know artworks be considered art?

Create a display that shows the time. Take a well know artwork as inspiration and encode the time in your visualization (a 15 minutes resolution is sufficient). To a casual observer it should not be apparent that the visualization shows the time.

What are the difficulties in designing such a visual display?

The following listing shows a basic clock. You can use this as a starting point for your own project.

```
1  // Variabeln definieren - ausserhalb von setup() und draw()
2  // da sie in beiden Funktionen verwendet werden
3  int stunde, minute, sekunde;
4
5  void setup() {
6  // Fenstergroesse festlegen
7     size(260, 100);
8
9     // Variabeln mit 0 initialisieren
10    stunde = 0;
11    minute = 0;
12    sekunde = 0;
13
14    // die Funktion draw() wird 2x pro Sekunde ausgefuehrt
15    frameRate(2);
16 }
17
18 void draw() {
19    // loesche den Hintergrund: setze ihn schwarz20background(0);
20    // setze Zeichenfarbe fuer Text und Recheck: weiss
21    fill(255, 255, 255);
22
23    // speichere in den Variablen die aktuelle Zeit
24    stunde = hour();
25    minute = minute();
26    sekunde = second();
27
28    // Kontrollausgabe der Zeit in der Console - kann spaeter
         geloeschtwerden
29    // in processing kann man mit "+" einen String aus Buchstaben und
         Zahlenbauen
30    println(stunde + ":" + minute + ":" + sekunde);
31
32    // zeige die Zeit im Fenster an
33    textSize(32);
34    text(stunde + ":" + minute + ":" + sekunde, 10, 30);
35
36    // zeichne einen Sekundenbalken
```

```
37     // Sekunde hat einen wert von 0 bis 59
38     // Sekunde multipliziert mit 4 hat einen Wert zwischen 0 und 236
39     rect(10, 40, sekunde*4, 20);
40 }
```

Ther is another example of a analog clock that may be helpful: `https://processing.org/examples/clock.html`

## 2.4 Tasks - 1st lecture

In the 1st session we try to mimic paintings of the famous dutch painter Piet Mondrian. He is famous for his abstract paintings that are reduced to a minimal palette of colors and geometric structures. His systematic approach invites to be recreated with algorithms.

### Preparation

If Processing isn't already installed on your computer, go to `https://processing.org/download/` download the program and install it on your machine. Open some example sketches and familiarize yourself with the Processing IDE. If you feel comfortable, proceed with the following task.

### Breakout Session

During the following task we try to approach the generation of random Mondrianesque images in a systematic way. Step by step we go from the analysis of real and fake Mondrians, over theoretical concepts for the description of Mondrianesque structures, to working implementations of the algorithm.

- Search on the internet for pictures of the painter Mondrian (see Figure 2.3). What are the connecting commonalities? Can you spot artworks not painted by Mondrian? Try to figure out the "recipe" that makes a distinguishable Mondrian.
- Let us discuss your approach to generate Mondrianesque images with the help of an algorithm. Try to sketch such structures and formalize them first.
- Try to implement one of the approaches we discussed in the plenum. Does it lead to the expected result? Does it work as simply as expected or have there been unexpected complications?
- If your algorithm is working, try to extend it to add your own style to the generated "Mondrians". Tweak parameters, change the color palette or even introduce more randomness.

### Discussion

We just created algorithms that mimic the style of a famous artist. Let us discuss your impressions and thoughts along the way. As a starting point think about the following questions:

- Can you call the product of these algorithms "art"?
- Can art in general be scripted? (Pollock, Warhol, ...)
- Does the used technology influence such a differentiation? (Style transfer performed with neural networks vs classic "conditional" algorithms)

### Homework

1. **Reading:** Read the following papers [Fer07; HS03; RSH00] and write a short (300-500 words) essay on Informative Art. Your essay should contain an explanation what it is, why people would want to do it, pros and cons.

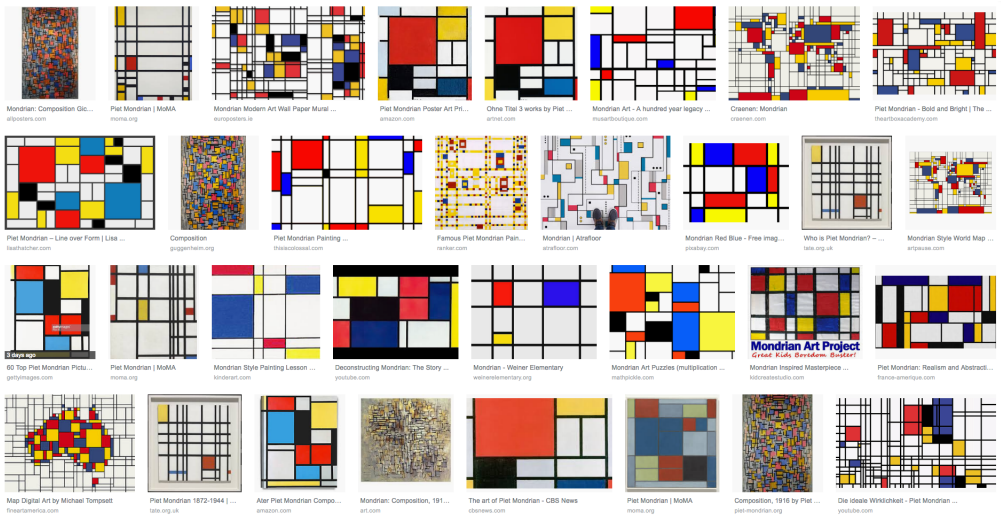


Figure 2.3: Artworks painted by Mondrian and Mondrian-inspired artworks.

2. **Implementation:** Create a display that shows the time. Take Mondrian algorithm and encode the time in your visualization (a 15 minutes resolution is sufficient).
3. ~~Collect feedback on your implementation. Record 3 videos from people seeing your implementation (15 seconds each).~~

# 3. Pixel based Art

## 3.1 Programming a "Pixel Converter"

Implement a program that takes an image as an input, analyzes the content and transfers it to a new output design. As an example look at pictures from Pointilism, ASCII art, etc..

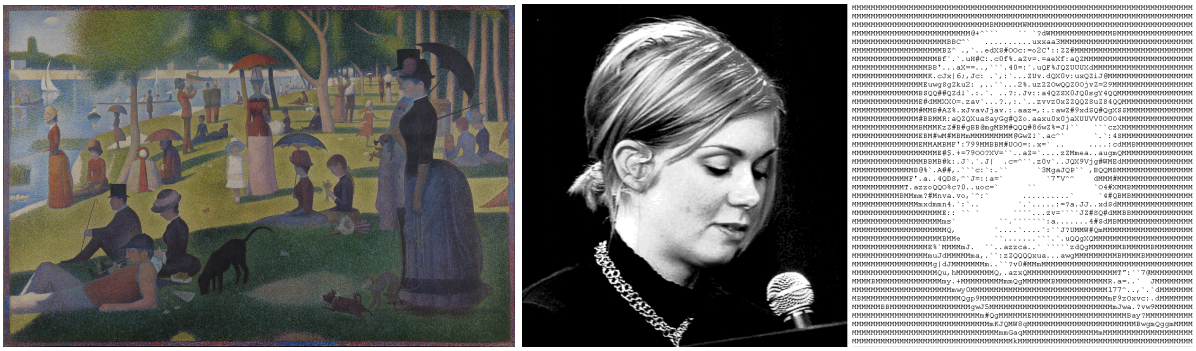

Figure 3.1: The shown examples demonstrate the translation of a color information to a interpreted output.

The program should take a predefined image from your file system or a snapshot over the integrated webcam of your computer.

**Useful Commands**

```
PVector vec;
loadImage("testfile.jpg");
IntList list;
color c;
red(c);
green(c);
blue(c;)
```

The following code is a starting point for your project.

```
PImage source;
int area = 25;

void setup () {

  size(900,600);
  source = loadImage("test.jpg");
  noStroke();

  for (int y=0; y<source.height; y+=area) {
    for (int x=0; x<source.width; x+=area){
      fill(getAverageColor(x,y,area,area,source));
      rect(x,y,area,area);
    }
  }
}

color getAverageColor ( int x, int y, int w, int h, PImage img) {
  float r = 0;
  float g = 0;
  float b = 0;

  for (int i=0; i<w*h; i++) {
    int index = x+(i%w)+y*img.width+(img.width*(i/w));
    r += red(img.pixels[index]);
    g += green(img.pixels[index]);
    b += blue(img.pixels[index]);
  }

  return color(r/(w*h),g/(w*h),b/(w*h));
}
```

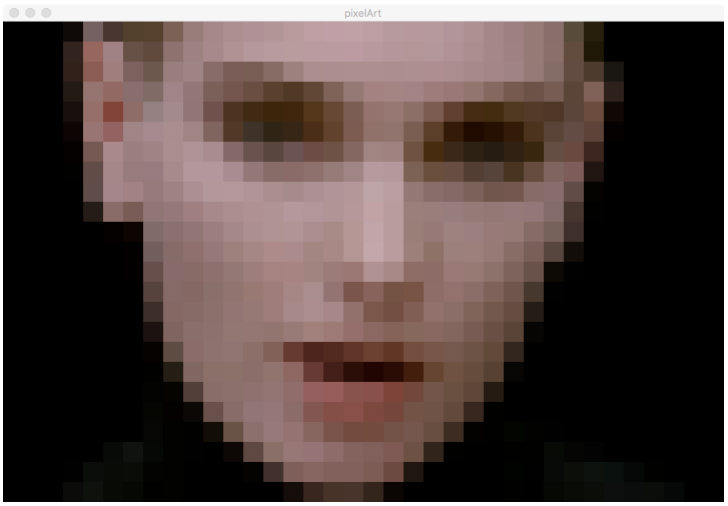


Figure 3.2: Resulting Image from the example code.

## 3.2 Convert to: Pointillism

Try to extend the given code to approximate the style of a pointillistic artwork:

- Draw many circles instead of one big square.
- Randomize the position of these circles.
- Randomize color values.

What effects can you observe that interfere with the pointillisitc style? Try to make your algorithm generate more vivid outcomes. If you look closely at the pointillistic image shown in the beginning of the chapter, one can observe that the painter uses small colors dots of primary colors that are mixed in our brain to multiple shades of the color palette. Can we imitate this effect?

## 3.3 Convert to other styles

In the first example code a function was given that translates a given area of an image to the average color of this area. What other informations are contained in such areas?

- color spread
- color palette
- ...

Try to define functions that take an image, coordinates an dimensions of the subarea to be observed and extract these informations.

```
color getAverageColor ( int x, int y, int w, int h, PImage img) {
  float r = 0;
  float g = 0;
  float b = 0;

  for (int i=0; i<w*h; i++) {
    int index = x+(i%w)+y*img.width+(img.width*(i/w));
    r += red(img.pixels[index]);
    g += green(img.pixels[index]);
    b += blue(img.pixels[index]);
  }

  return color(r/(w*h),g/(w*h),b/(w*h));
}

ArrayList<Integer> getColors ( int x, int y, int w, int h, PImage img) {
  ArrayList<Integer> colors = new ArrayList<Integer>();

  for (int i=0; i<w*h; i++) {
    int index = x+(i%w)+y*img.width+(img.width*(i/w));
    color c = img.pixels[index];
    if (!colors.contains(c)) colors.add(c);
  }

  return colors;
}

...
```

## 3.4 Tasks - 2nd lecture

In the 2nd session we try to process image information and translate these into new "styles". As a source of inspiration we discuss the genre of pointillism as well as pictures from the ASCII art

scene. Further, we try to go beyond these two styles an approach new ideas and concepts.

#### Breakout Session

During the following task we try to think of our algorithms as transfer functions. With an image given we take the contained information (colors of pixels) and transfer it into: colored shapes, ASCII characters, ...

In this model the image is the input, which is then analyzed on a subarea level. The contained information is transformed by a small instruction set and thereby applied to the whole image.

- Search on the internet for pointillisitc images and for examples of ASCII art.
- Discuss the different concepts how colors, shapes and more are abstracted.
- Start to implement your analyzed concept based on the previously given example code.
- Where are the weak points of the algorithm? Can you spot its algorithmic nature? How can theses problems be addressed and solved?

#### Discussion

What are the most interesting aspects of this approach? Let us discuss your impressions and thoughts along the way. As a starting point think about the following questions:

- Can you call the product of these algorithms "art"?
- For now, we have reduced information (subareas), can we also extend to a higher richness of detail?
- Could you imagine to use such a technique in your own work?

#### Homework

1. **Reading:** Read the following papers `http://www.cse.cuhk.edu.hk/~ttwong/papers/asciiart/asciiart.pdf` to get an idea about the challanges and approaches to ASCII art.
2. **Implementation:** Look into the following webpage `http://paulbourke.net/dataformats/asciiart/` and adopt these rules for ASCII art to our algorithm. Implement a "Convert to: ASCII" sketch.
3. Think of the following: For the moment our ASCII sketch produces random character sequences based on the color value of the source pixels. Can we use words instead of random sequences? What do you have to do to take as an additional input something like Shakespeare's collected work and search in there for matching "color/character" sequences? Try an implementation.

# 5. Lines

## 5.1 Frieder Nake

In this section we will look into the algorithmic art of Frieder Nake. Nake is famous for his first explorations of the "Graphomat" (today we call such machines plotters). He decided to experiment with this new type of machines in an artistic way during his task to write software for this new hardware which was targeted at architects and engineers to speed up and ease the drawing process. Instead of typical calibration tests he wrote algorithms that "created" artistic and randomly generated drawings.

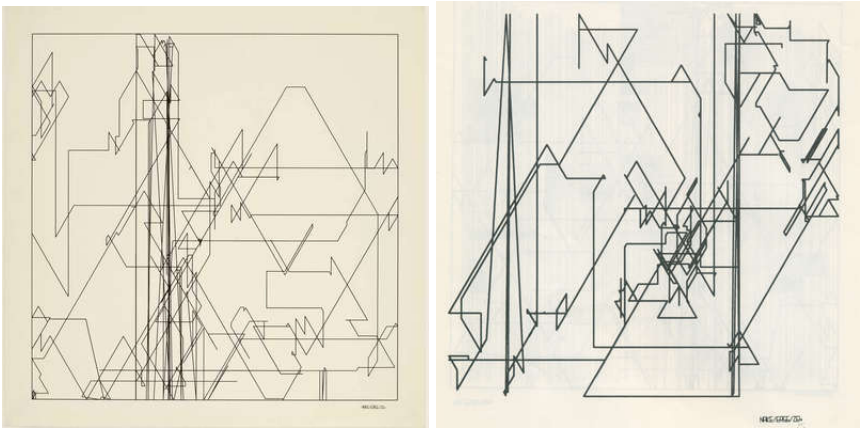

Figure 5.1: Rule sets can specify angles minimum and maximum distances, ... .

We will try to implement to implement an algorithm that generates similar aesthetics. Further, it would be interesting to "watch" the algorithm draw. How can we follow along with the "plotter" during the "creation" process?

### Useful Commands

```
PVector vec;
PVector.random2D();
PVector.add(vec1,vec2);
PVector.sub(vec1,vec2);
PVector.mult(vec1,f);
random();
constrain();
```

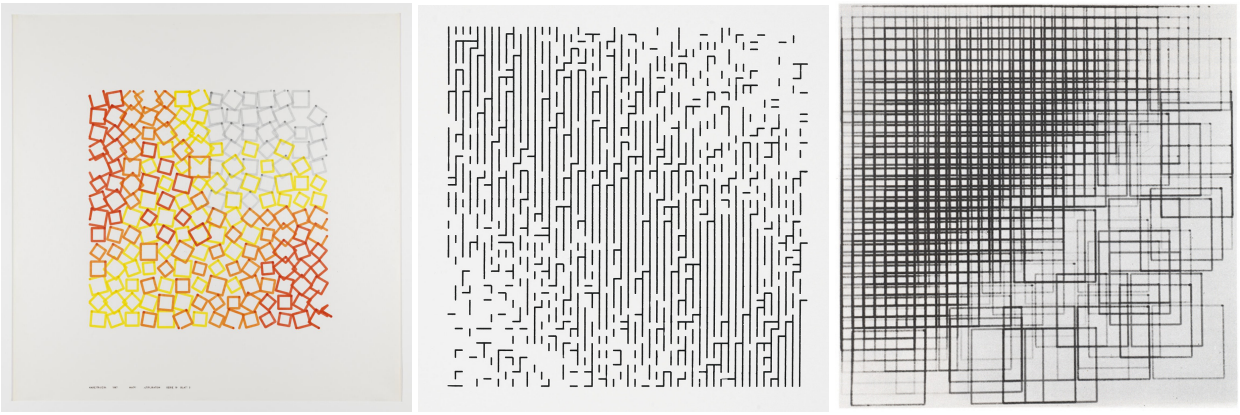

Figure 5.2: Nake's work is manifold and nevertheless beautiful in its simplicity.

The following code is a starting point for your project.

```
PVector plotterHead,last;
PVector direction;

void setup () {
  size(800,800);
  smooth();
  background(255,250,240);
  stroke(0,100);
  strokeWeight(4);

  plotterHead = new PVector(random(800),random(800));
  last = plotterHead.copy();
  direction = PVector.random2D();
}

void draw () {
  plotterHead = PVector.add(plotterHead,direction);
  line(plotterHead.x,plotterHead.y,last.x,last.y);

  last = plotterHead.copy();
  if (random(100)<=1 || (plotterHead.x>=width || plotterHead.x<=0 ||
      plotterHead.y>=height || plotterHead.y<=0)) {
    direction = PVector.random2D();
  }
}
```

## 5.2 Implement: Specify Angles

Try to give the algorithm a "rule set" that specifies the angles to be used. Should there be just 90 degree angles or random angles or ...?

- Just pick specific angles instead of random ones.
- Think of using an array that contains the valid angles?

How can we apply more complex rules such as: 90 degree lines are longer then 60 degree ones.

## 5.3 What else to implement?

If we compare our outcome to the found drawings of Nake's algorithms where are the differences?

- Try to find and point out as many differences as occurre to you.
- Do you think every drawing of the algorithm is an artwork?
- Did the artist picked the best ones?

Discuss the given tasks and questions in your group.

## 5.4 Tasks - 3rd lecture

In the 4rd session we want to discuss how we can implement simple abstract and line-based algorithmic art.

### Breakout Session

During the following task we take a closer look at different kinds of brushes and strokes.

- Search on the internet for examples of Nake's work.
- Discuss the different parameters he used over the course of his different cycles.
- what role do randomness and order play in his work?
- Is the beauty just in the aesthetic of the drawings or as well in the simplicity of the logic behind?
- Expand the example code with some of your ideas. Try to find the most elegant solution for your specific problem.

### Homework

1. **Reading:** Read through the following web document `http://dreher.netzliteratur.net/4_Medienkunst_Kybernetik.html` and this paper written by Frieder Nake [Nak05] `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.470.2597&rep=rep1&type=pdf`
2. **Implementation:** Implement two of Nake's other algorithms (as shown in Figure 5.2 or found online).

```
19   in.start();
20
21   // patch the AudioIn
22   fft.input(in);
23 }
24
25 void draw() {
26   background(255);
27   fft.analyze(spectrum);
28
29   for(int i = 0; i < bands; i++){
30   // The result of the FFT is normalized
31   // draw the line for frequency band i scaling it up by 5 to get more
         amplitude.
32   line( i, height, i, height - spectrum[i]*height*5 );
33   }
34 }
```

## 7.4 Tasks - 7th lecture

Work on your final projects, try to sketch the rough structure of your processing sketch. What are the major challenges of your idea? What do you need as additional input?

### Homework

As the homework prepare and document one your favorite sketches you implemented over the course of the semester. We will use these sketches as illustrative material for the next year's course. Please do the following:

1. Pick your favorite sketch you implemented so far,
2. review your code and try to simplify it as far as possible (use clear variable names, use methods to avoid redundancy, ...),
3. go through your code and add comments to make the sketch comprehensible, add a description of your sketch in the beginning of the sketch,
4. make a high quality rendering of the outcome (.png, .jpg, ...).

Collect all the materials and hand them in as a .zip file.

# Bibliographie


[Fer07] Alois Ferscha. “Informative art display metaphors”. In: International Conference on Universal Access in Human-Computer Interaction. Springer. 2007, pages 82–92 (cited on pages 11, 12).

[HS03] Lars Erik Holmquist and Tobias Skog. “Informative art: information visualization in everyday environments”. In: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia a ACM. 2003, pages 229–235 (cited on pages 11, 12).

[Nak05] Frieder Nake. “Computer art: a personal recollection”. In: Proceedings of the 5th conference on Creativity & cognition. ACM. 2005, pages 54–62 (cited on pages 23, 26).

[RSH00] Johan Redström, Tobias Skog, and Lars Hallnäs. “Informative art: using amplified artworks as information displays”. In: Proceedings of DARE 2000 on Designing augmented reality environments. ACM. 2000, pages 103–114 (cited on pages 11, 12).