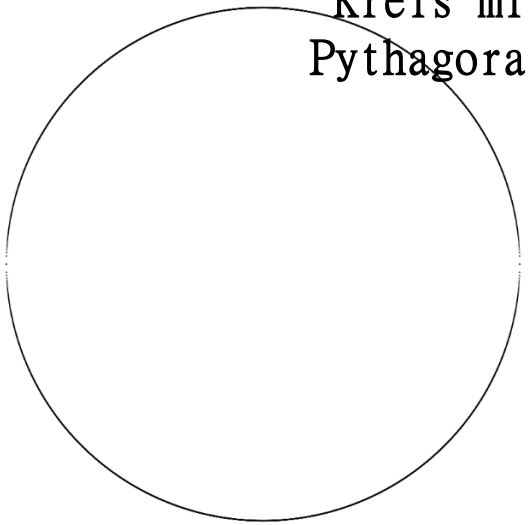
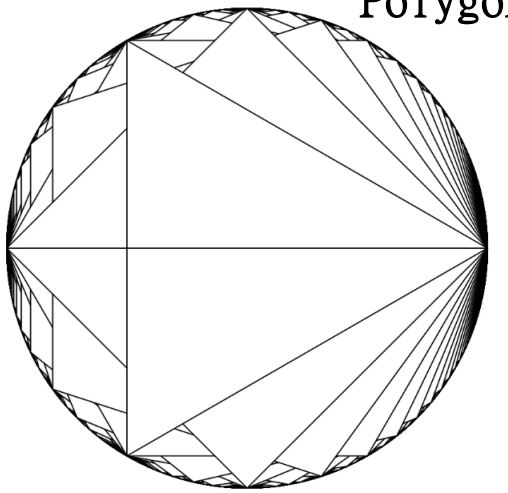
```
function setup() {
    let r = 200;
    let xpos = windowWidth / 2;
    let ypos = windowHeight / 2;
    createCanvas(windowWidth - 40, windowHeight - 40);

    drawCircle(xpos, ypos, r);
    //drawSpiral(xpos, ypos, 0.5, 5);
}

function drawCircle(xpos, ypos, r) {
    for (let x = 0; x <= r; x += 0.1) {
        let y = sqrt(r * r - x * x);

        point(x + xpos, y + ypos);
        point(x + xpos, -y + ypos);
        point(-x + xpos, y + ypos);
        point(-x + xpos, -y + ypos);
    }
}

function drawSpiral(xpos, ypos, growth, size) {
    let r = 0;
    for (let i = 0; i < size; i++) {
        for (let x = r; x >= -r; x--) {
            let y = sqrt(r * r - x * x);

            point(x + xpos, y + ypos);
            r += growth;
        }
        for (let x = r; x >= -r; x--) {
            let y = sqrt(r * r - x * x);

            point(-x + xpos, -y + ypos);
            r+=growth;
        }
    }
}
```

Kreis mit Pythagoras

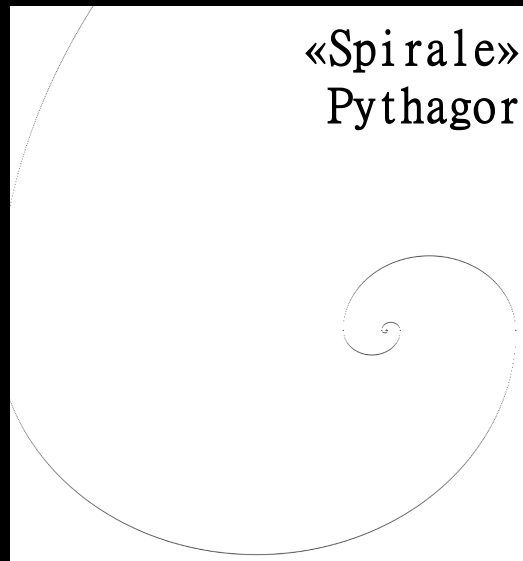


Polygone



«Spirale» mit Pythagoras 1

«Spirale» mit Pythagoras 2

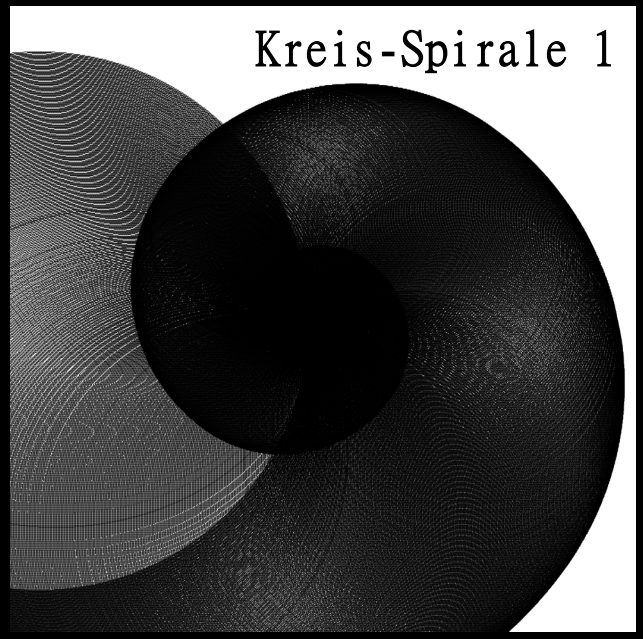


```
function setup() {
    let r = 200;
    let xpos = windowWidth * 0.8 / 2;
    let ypos = windowHeight * 0.8 / 2;
    createCanvas(windowWidth * 0.8, windowHeight * 0.8);

    drawCircle(xpos, ypos, r);
    //drawSpiral(xpos, ypos, 20, 5);
}

function drawCircle(xpos, ypos, r) {
    let x = -r;
    let y = 0;
    point(x + xpos, y + ypos);
    //1/8
    while (y > x) {
        let r1 = sqrt(x * x + (y - 1) * (y - 1));
        let r2 = sqrt((x + 1) * (x + 1) + (y - 1) * (y - 1));
        if (Math.abs(r1 - r) > Math.abs(r2 - r)) {
            x++;
            y--;
        } else {
            y--;
        }
        point(x + xpos, y + ypos);
    }
    //2/8
    while (x < 0) {
        let r1 = sqrt((x + 1) * (x + 1) + (y - 1) * (y - 1));
        let r2 = sqrt((x + 1) * (x + 1) + y * y);
        if (Math.abs(r1 - r) > Math.abs(r2 - r)) {
            x++;
        } else {
            x++;
            y--;
        }
        point(x + xpos, y + ypos);
    }
    //3/8
    // (usw. für alle 8 Kreisabschnitte...)
}
```

Kreis mit Bresenham

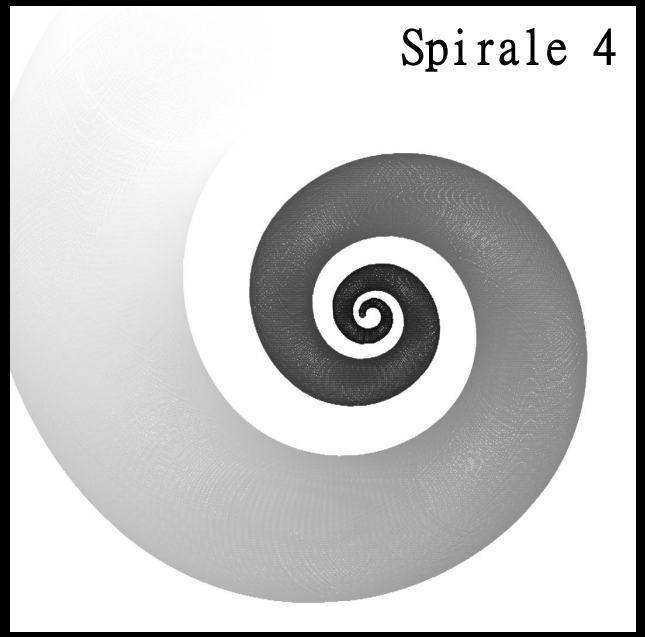
Spirale mit Bresenham

```
function setup() {
    let r = 200;
    let xpos = windowWidth * 0.8 / 2;
    let ypos = windowHeight * 0.8 / 2;
    createCanvas(windowWidth * 0.8, windowHeight * 0.8);

    //drawSpiral(xpos, ypos, 12, 6);
    drawSpiral(xpos, ypos, 20, 3);
}

function drawSpiral(xpos, ypos, growth, size) {
    growth *= 0.01;
    let r = 10;
    let x = -r;
    let y = 0;

    drawCircle(x + xpos, y + ypos, r / 3);
    r += growth;
    for (let i = 0; i <= size; i++) {
        console.log(x + " " + y);
        //1/8
        while (y >= x) {
            let r0 = sqrt((x - 1) * (x - 1) + (y - 1) * (y - 1));
            let r1 = sqrt(x * x + (y - 1) * (y - 1));
            let r2 = sqrt((x + 1) * (x + 1) + (y - 1) * (y - 1));
            if (Math.abs(r1 - r) > Math.abs(r2 - r)) {
                x++;
                y--;
            } else if (Math.abs(r0 - r) > Math.abs(r1 - r)) {
                y--;
            } else {
                x--;
                y--;
            }
            drawCircle(x + xpos, y + ypos, r / 3);
            r += growth;
        }
        //2/8
        // (usw. für alle 8 Spiralenabschnitte...)
    }
    function drawCircle(xpos, ypos, r) {
        // (...)
    }
}
```



Spiralen aus grösser werdenden Kreisen mit Bresenham

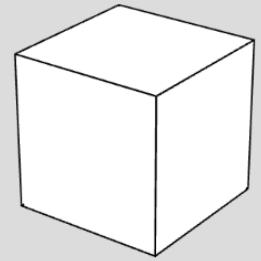
```
//(...)

let red = 255;
let green = 255;
let blue = 255;

//(...)

//Spirale 3
if (green > 0) {
    green -= 0.2;
} else if (blue > 0) {
    blue -= 0.2;
} else if (red > 0) {
    red -= 0.2;
}

//Spirale 1
red = random(255) * 0.5 + 127.5;
green = random(255);
blue = random(255) * 0.5 + 127.5;

//Spirale 4
red = r * 2;
green = r * 2;
blue = r * 2;

//(...)
```



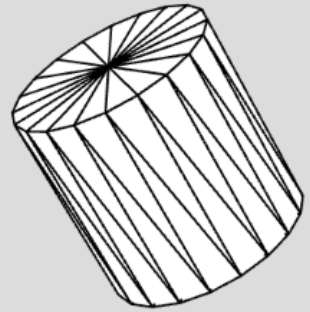
Experimente mit Farbe

```
//(...)

function drawCylinder(radius, height) {

    let angle = TWO_PI / numSides;

    beginShape(TRIANGLE_STRIP);
    for (let i = 0; i <= numSides; i++) {
        let x = radius * cos(angle * i);
        let y = radius * sin(angle * i);
        let z = height / 2;
        vertex(x, y, z);
        vertex(x, y, -z);
    }
    endShape(CLOSE);

    beginShape(TRIANGLE_FAN);
    vertex(0, 0, height / 2);
    for (let i = 0; i <= numSides; i++) {
        let x = radius * cos(angle * i);
        let y = radius * sin(angle * i);
        vertex(x, y, height / 2);
    }
    endShape(CLOSE);

    beginShape(TRIANGLE_FAN);
    vertex(0, 0, -height / 2);
    for (let i = 0; i <= numSides; i++) {
        let x = radius * cos(angle * i);
        let y = radius * sin(angle * i);
        vertex(x, y, -height / 2);
    }
    endShape(CLOSE);
}

//(...)
```

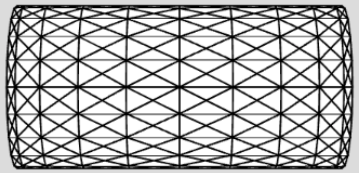
Würfel



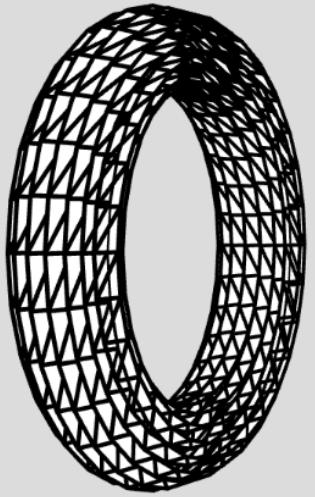
Zylinder



«Zylinder» / «Torus»

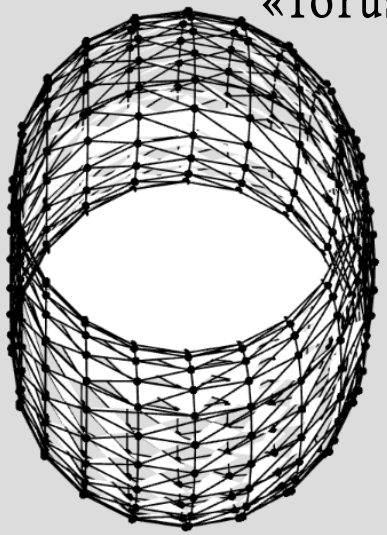


«Torus»

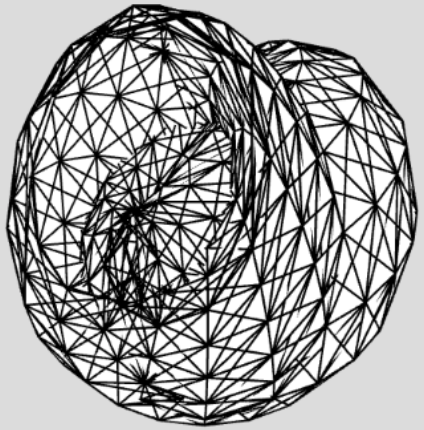


```
//(...)

//Torus 4
function drawTorus(radius, tubeRadius) {
    beginShape(TRIANGLES);
    fill(255);
    stroke(0);
    strokeWeight(1);

    let numSides = 20;
    let vertices = [];
    let angle = TWO_PI / numSides;
    for (let i = 0; i < numSides; i++) {
        let x = radius * cos(angle * i);
        let y = radius * sin(angle * i);
        for (let j = 0; j < numSides; j++) {
            let y = tubeRadius * sin(angle * j);
            let z = tubeRadius * cos(angle * j);
            vertices.push([x, y, z]);
        }
    }

    for (let i = 0; i < numSides; i++) {
        let x = radius * cos(angle * i);
        let y = radius * sin(angle * i);
        for (let j = 0; j < numSides; j++) {
            let z = tubeRadius * cos(angle * j);
            let a = (i + 1) % numSides;
            let b = (j + 1) % numSides;
            let c = (i + 1) % numSides;
            let d = (j + 1) % numSides;
            vertex(vertices[i][0], vertices[i][1], vertices[i][2]);
            vertex(vertices[a][0], vertices[a][1], vertices[a][2]);
            vertex(vertices[b][0], vertices[b][1], vertices[b][2]);
            vertex(vertices[i][0], vertices[i][1], vertices[i][2]);
            vertex(vertices[c][0], vertices[c][1], vertices[c][2]);
            vertex(vertices[d][0], vertices[d][1], vertices[d][2]);
        }
    }

    endShape(CLOSE);
}

//(...)
```

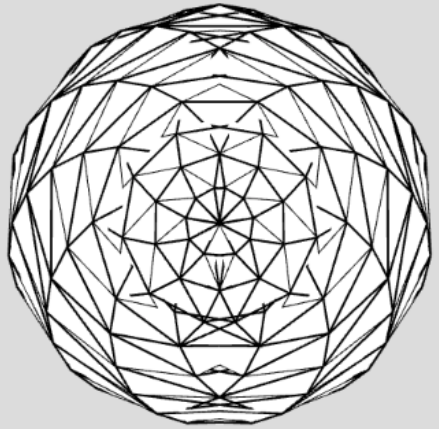
«Torus» 1



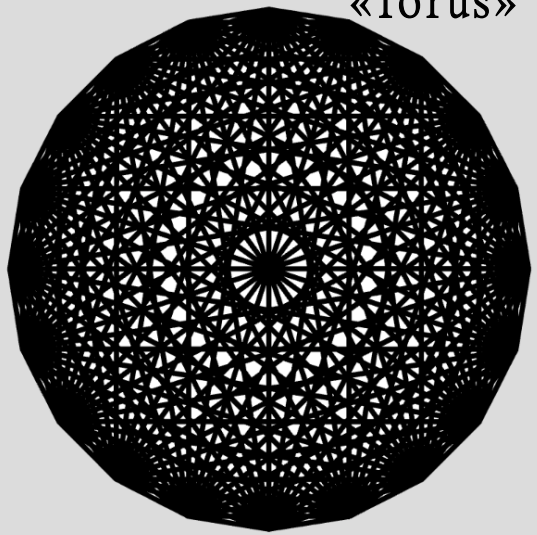
«Torus» 2



«Torus» 3



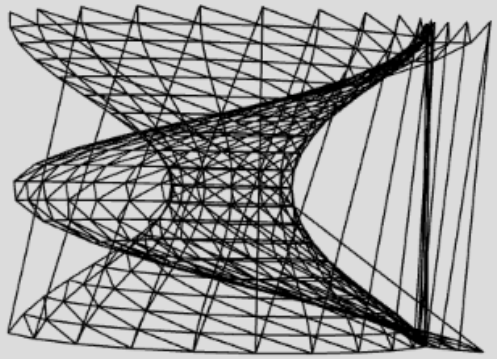
«Torus» 4



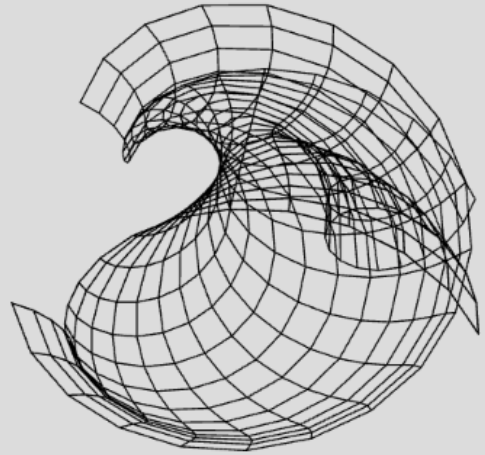
Der Weg zum Torus

```
//(...)

function drawSnail(radius, tubeRadius) {
    let angle = TWO_PI / numSides;
    for (let i = 0; i < numSides; i++) {
        let vertex = [];
        let xpos = radius * cos(angle * i);
        let ypos = radius * sin(angle * i);
        for (let j = 0; j < numSides; j++) {
            let x = tubeRadius * cos(angle * j);
            let y = tubeRadius * sin(angle * j);
            let z = j * 10;

            vertex.push([x + xpos, y + ypos, z]);
        }
        vertex.push(vertex[vertex.length - 1]);
        vertices.push(vertex);

        tubeRadius -= 10; //Schnecke 4
        radius -= 5; //Schnecke 3
    }
    vertices.push(vertices[vertices.length - 1]);

    for (let i = 0; i < numSides; i++) {
        beginShape(TRIANGLE_STRIP);
        for (let j = 0; j < numSides; j++) {
            let v1 = vertices[i + 1][j];
            let v2 = vertices[i + 1][j + 1];
            let v3 = vertices[i][j + 1];
            let v4 = vertices[i][j];

            vertex(v1[0], v1[1], v1[2]);
            vertex(v2[0], v2[1], v2[2]);
            vertex(v3[0], v3[1], v3[2]);

            vertex(v1[0], v1[1], v1[2]);
            vertex(v3[0], v3[1], v3[2]);
            vertex(v4[0], v4[1], v4[2]);
        }
        endShape();
    }
}

//(...)
```

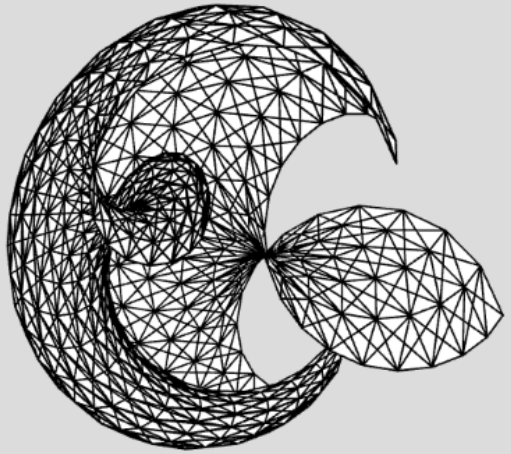
«Schnecke» 1



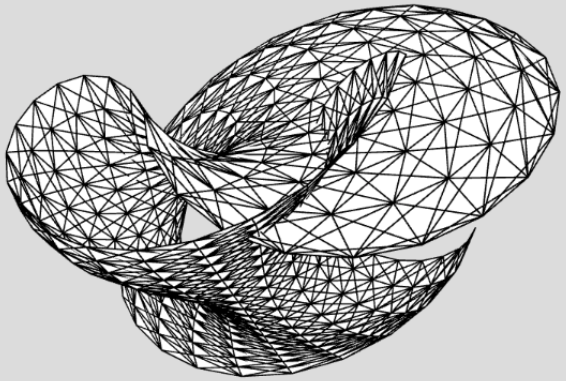
«Schnecke» 2



«Schnecke» 3



«Schnecke» 4



Der Weg zur «Schnecke»

```
//(...)

function feedText(text) {
    let words = text.split(" ");

    for (let i = 0; i < words.length - 1; i++) {
        let currentWord = words[i];
        let nextWord = words[i + 1];

        if (markovChain[currentWord]) {
            markovChain[currentWord].push(nextWord);
        } else {
            markovChain[currentWord] = [nextWord];
        }
    }
}

function generateText(startWord, length) {
    let result = startWord + " ";
    let currentWord = startWord;

    for (let i = 0; i < length; i++) {
        if (markovChain[currentWord]) {
            let nextWord = getRandomElement(markovChain[currentWord]);
            result += nextWord + " ";
            currentWord = nextWord;
        } else {
            break;
        }
    }

    return result;
}

function setup() {

    let inputText = `Birdo ist […] Figur im Videospiel-Franchise.`;
    feedText(inputText);

    generatedText = generateText("Birdo", 100);
    createP(generatedText);
}

//(...)
```

Deutscher Text

Birdo beispielsweise dafür bekannt, Eier als Doki Panic veröffentlicht, war Birdo beispielsweise dafür bekannt, Eier als männlich beschrieben, der Charaktere mit mächtigen Schlägen bekannt ist. Birdos Status als einer der ikonischen Figuren im Laufe der ersten bekannten transgender-Charaktere in einem Mainstream-Videospiel, hat Birdo respektiert oder Missverständnissen basiert. Nintendo hat in Japan als Doki Panic veröffentlicht, war das Spiel war das Nintendo Entertainment System eingeführt, das Nintendo Entertainment System eingeführt, das Nintendo die Darstellung oft weggelassen oder Hintergrund von Birdo ein Spin-off von der ersten bekannten transgender-Charaktere in der englischen Anleitung von Super Mario Tennis und einem hervorstehenden Loch dargestellt, aus den Hintergrund zu vielen Diskussionen weiter

Japanischer Text in Romaji

Watashi wa Nihongo ga arimasu. Haha wa ōkina kaisha de iru toki ni, yoku kazoku de hataraki o mimasu. Watashi wa chichi wa gakkō no tanjōbi wa gorufu desu. Kaisha no tomodachi ni wa gakkō ni noru koto ga arimasu. Watashi no chichi no ie wa inu mo irasshaimasu. Hajimemashite, watashi wa Nihon de shigoto o sugosu koto ga daisuki desu. Chichi wa chichi no ie ni wa totemo muzukashii desu. Shigoto no bunka wa yoku benkyō o tsukurimasu. Watashi no ato ni iku yō ni wa neko ga arimasu. Watashi no kanojo wa totemo kawaii desu. Watashi no haha wa neko ga

In Zusammenarbeit mit Lisa

```
//(...)

function generateSentences(startWord, num) {
    let sentences = [];
    let result = startWord + " ";
    let currentWord = startWord;

    for (let i = 0; i < num; i++) {
        do {
            //(...)
        } while (!currentWord.includes("."));
        console.log(result);
        result = toHiragana(result);
        sentences.push(result);
        result = "";
    }
    return sentences;
}

function toHiragana(text) {

    text = text.toLowerCase();
    //(...)
    text = text.replace(/a/g, "あ");
    text = text.replace(/i/g, "い");
    text = text.replace(/u/g, "う");
    text = text.replace(/e/g, "え");
    text = text.replace(/o/g, "お");
    text = text.replace(/n/g, "ん");
    text = text.replace(/\./g, "。");
    text = text.replace(/,/g, "、");
    text = text.replace(/-/g, "");
    text = text.replace(/ /g, "");
    //(...)
    return text;
}

//(...)
```

Japanischer Text in Hiragana

わたしはよくでぱあとにはねこがあります。わたしのいえにぷれぜんとおはじめたから、にほんごがいます。 ねこはしゅうまつです。せんぱいはぎんこうのゆうがたにぷれぜんとおあげます。ちちはちちのせんせいです。 わたしのがくせいです。わたしはこうばんがいます。 ねこはよくねます。わたしのtあんです。ともだちはとてもうれしいです。 どいつたいしかんはにほんじんです。わたしはねこがじょうずです。ともだちはごるふです。せんぱいはがくせいもいます。わたしのせんぱいはどいつたいしかんはむずかしいけど、たのしいほびはごるふです。あしたはとてもかわいいです。わたしのなまえ

Japanische Sätze in Hiragana

わたしのびょういんでしごとはしゅうまつです。

にほんにはおおきながっこうのちちにのることです。

いぬがあります。

ははのいえにいくようにすんでいます。

わたしのしごとしています。

わたしのかのじょはいぬもいます。

せんぱいはおおきながっこうのがくせいもいます。

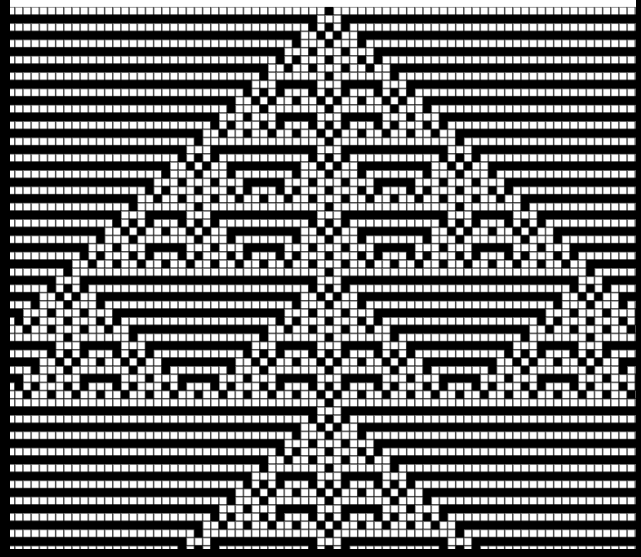
```
//(...)

function makeLine(text) {
    let characters = text.split('');
    let p = createP();
    let index = 0;

    p.elt.style.left = Math.floor(Math.random() * (window.innerWidth
    - p.elt.offsetWidth))-window.innerWidth/20 + 'px';

    let interval = setInterval(() => {
        if (index < characters.length) {
            let span = document.createElement('span');
            span.textContent = characters[index];
            span.classList.add('initial');
            if (characters[index] == "。" ||
                characters[index] == "、") {
                span.classList.add('dot');
            }
            p.elt.appendChild(span);
            setTimeout(() => {
                span.classList.add('transition');
                setTimeout(() => {
                    span.style.color = 'black';
                }, 2000);
            }, 100);
            index++;
        } else {
            clearInterval(interval);
        }
    }, 100);
}

//(...)
```

Experimente mit Farbe, Schriftart & Sprache

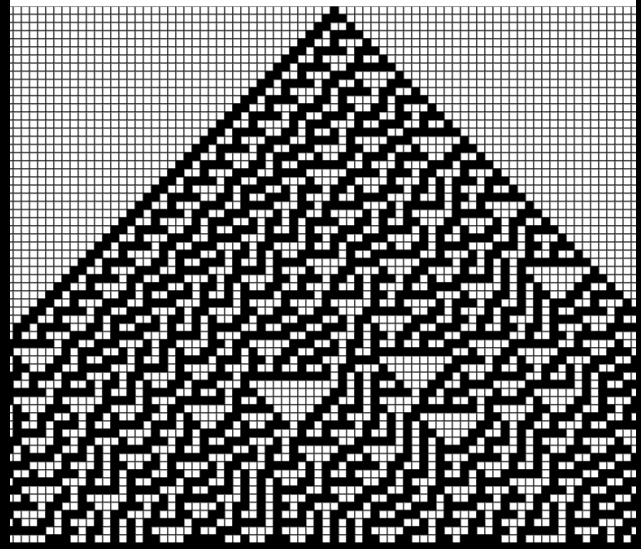
```
//(...)

function setup() {

    let inputText = `Doitsu […] ureshii desu.`;
    feedText(inputText);

    generatedSentences = generateSentences("Watashi", 70);
    generatedSentences.forEach(sentence => {
        setTimeout(() => {
            makeLine(sentence);
        }, Math.random() * 10000);
    });
}

//(...)
```

```
//(...)

function myRule(left, middle, right) {
    let binaryString = '' + left + middle + right;
    switch (binaryString) {
        case '111': return 0;
        case '110': return 1;
        case '101': return 1;
        case '100': return 0;
        case '011': return 1;
        case '010': return 0;
        case '001': return 0;
        case '000': return 1;
    }
}

function generateNextGen() {
    for (let i = 0; i < cols; i++) {
        let left = currentGen[(i - 1 + cols) % cols];
        let middle = currentGen[i];
        let right = currentGen[(i + 1) % cols];
        nextGen[i] = myRule(left, middle, right);
    }
    currentGen = nextGen.slice();
}

function drawGeneration(gen, row) {
    for (let i = 0; i < gen.length; i++) {
        let x = i * cellSize;
        let y = row * cellSize;
        if (gen[i] === 1) {
            fill(0);
        } else {
            fill(255);
        }
        stroke(0);
        rect(x, y, cellSize, cellSize);
    }
}

//(...)
```

My Rule

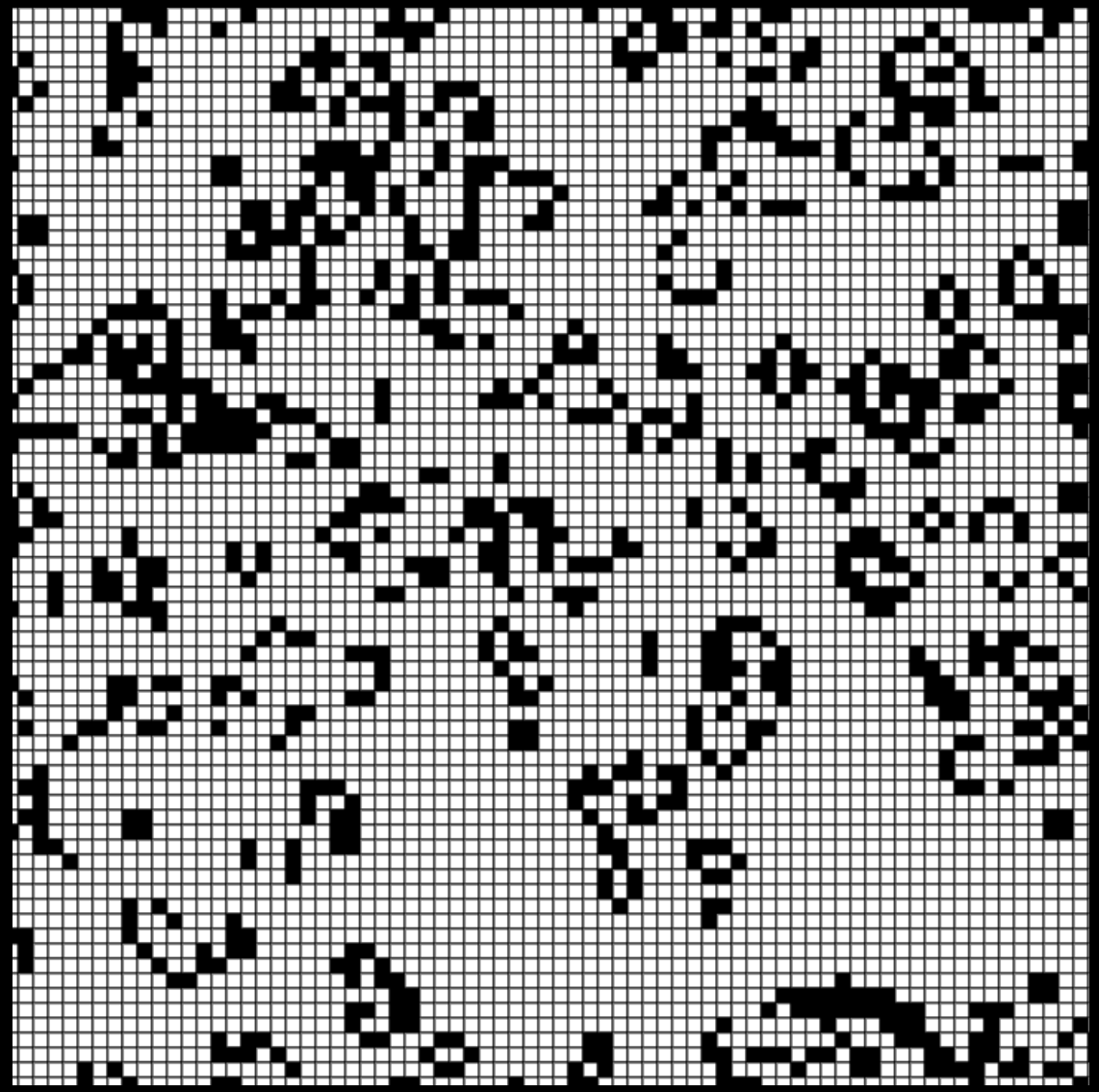


Rule 30



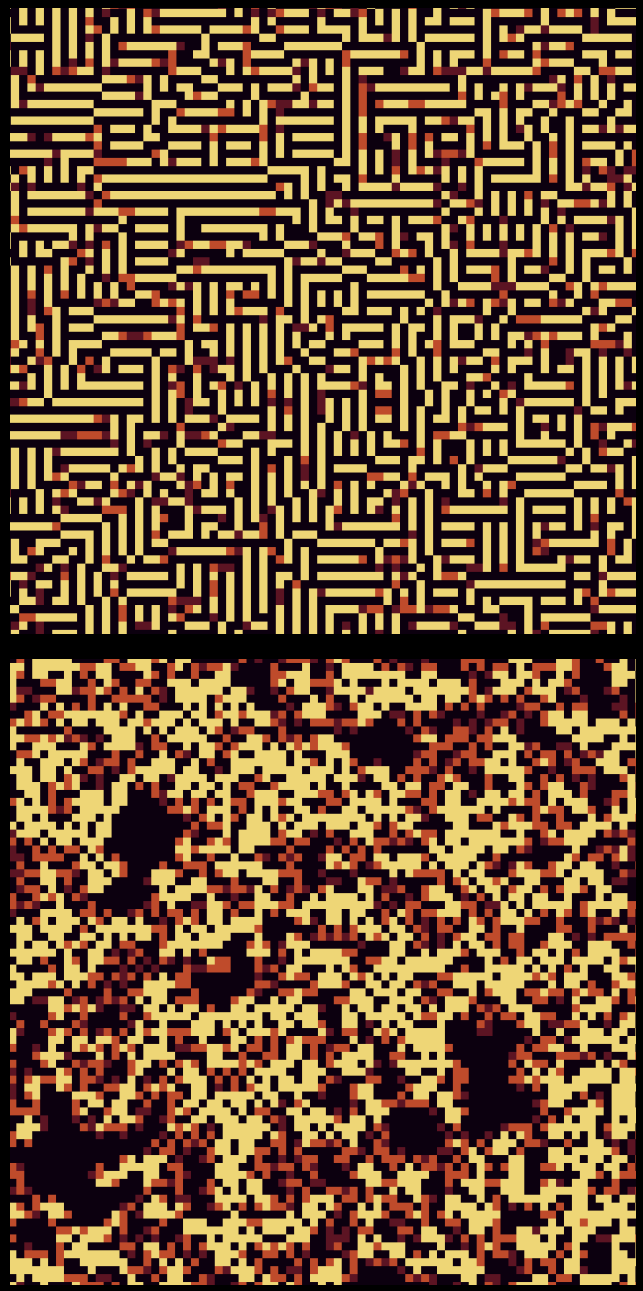
Zelluläre Automaten in 1D

```
//(...)

function draw() {
    background(255);

    for (let i = 0; i < cols; i++) {
        for (let j = 0; j < rows; j++) {
            let x = i * cellSize;
            let y = j * cellSize;
            if (grid[i][j] === 1) {
                fill(0);
            } else {
                fill(255);
            }
            stroke(255);
            rect(x, y, cellSize, cellSize);
        }
    }
    generateNextGen();
}

function generateNextGen() {
    for (let i = 0; i < cols; i++) {
        for (let j = 0; j < rows; j++) {
            let state = grid[i][j];
            let neighbors = countNeighbors(grid, i, j);

            if (state === 0 && neighbors === 3) {
                nextGrid[i][j] = 1; // Birth
            } else if (state === 1 && (neighbors < 2 || neighbors > 3)) {
                nextGrid[i][j] = 0; // Death
            } else {
                nextGrid[i][j] = state; // Stays the same
            }
        }
    }
    let temp = grid;
    grid = nextGrid;
    nextGrid = temp;
}

//(...)
```



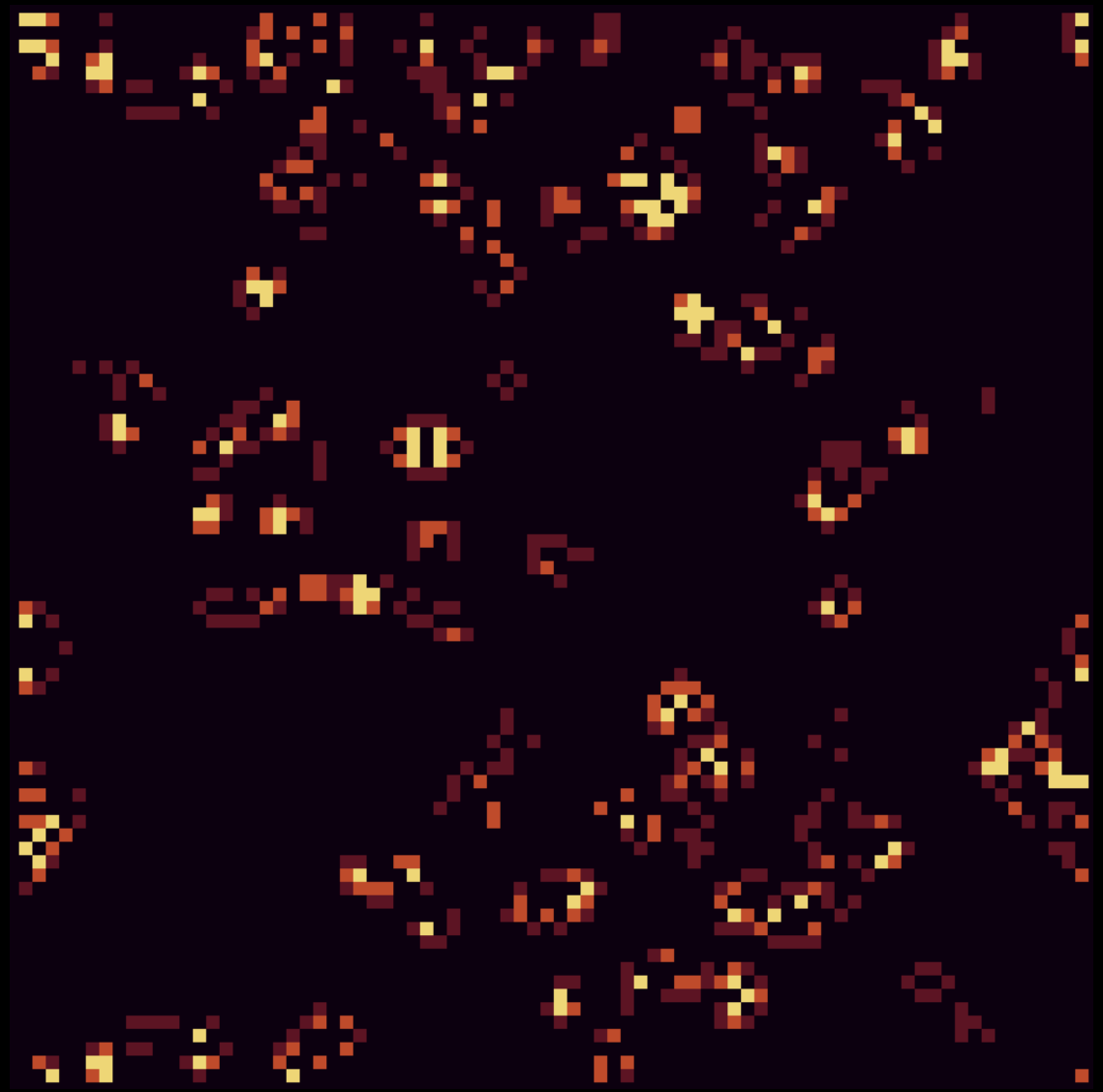
Zellulärer Automat in 2D «Game of Life»

```
//(...)

for (let i = 0; i < cols; i++) {
        for (let j = 0; j < rows; j++) {
            let state = grid[i][j];
            let neighbors = countNeighbors(grid, i, j);

            if (neighbors > 10 && neighbors < 14) {
                if (state === 3) {
                    nextGrid[i][j] = state; // Stay the same
                } else {
                    nextGrid[i][j] = state + 1; // Upgrade
                }
            } else if (neighbors < 8 || neighbors > 16) {
                if (state === 0) {
                    nextGrid[i][j] = state; // Stay the same
                } else {
                    nextGrid[i][j] = state - 1; // Downgrade
                }
            } else {
                nextGrid[i][j] = state; // Stay the same
            }
        }
}

//(...)

function countNeighbors(x, y) {
    let sum = 0;
    for (let i = -1; i <= 1; i++) {
        for (let j = -1; j <= 1; j++) {
            let col = (x + i + cols) % cols;
            let row = (y + j + rows) % rows;
            sum += grid[col][row];
        }
    }
    sum -= grid[x][y];
    return sum;
}

//(...)
```



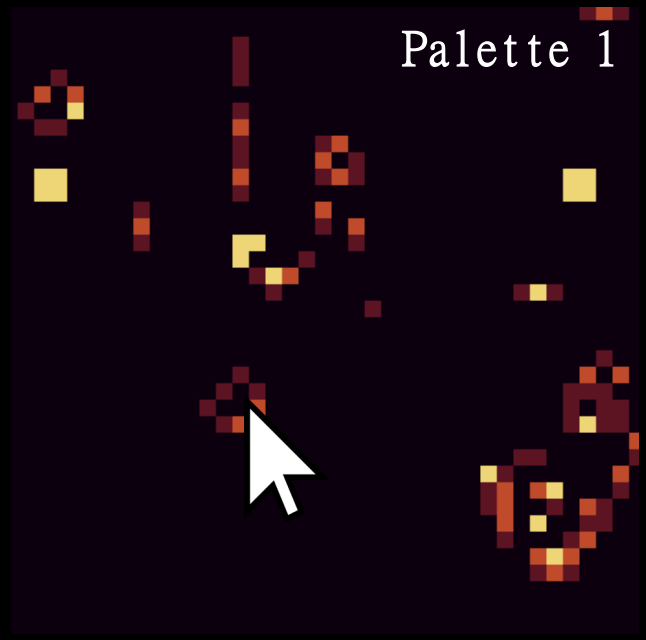
Erste Versuche mit mehr Zuständen

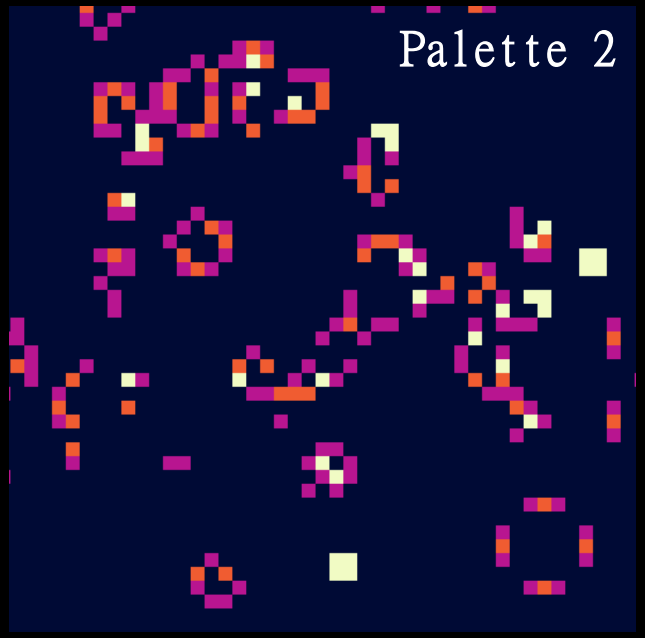
```
//(...)

for (let i = 0; i < cols; i++) {
        for (let j = 0; j < rows; j++) {
            let state = grid[i][j];
            let neighbors = countNeighbors(i, j);
            if (neighbors === 3) {
                if (state === 3) {
                    nextGrid[i][j] = state; // Stay the same
                } else {
                    nextGrid[i][j] = state + 1; // Upgrade
                }
            } else if (neighbors < 2 || neighbors > 3) {
                nextGrid[i][j] = 0; // Death
            } else {
                nextGrid[i][j] = state; // Stay the same
            }
        }
}

function countNeighbors(x, y) {
    let sum = 0;
    for (let i = -1; i <= 1; i++) {
        for (let j = -1; j <= 1; j++) {
            let col = (x + i + cols) % cols;
            let row = (y + j + rows) % rows;
            if (grid[col][row] !== 0) {
                sum++;
            }
        }
    }
    if (grid[x][y] !== 0) {
        sum--;
    }
    return sum;
}

//(...)
```

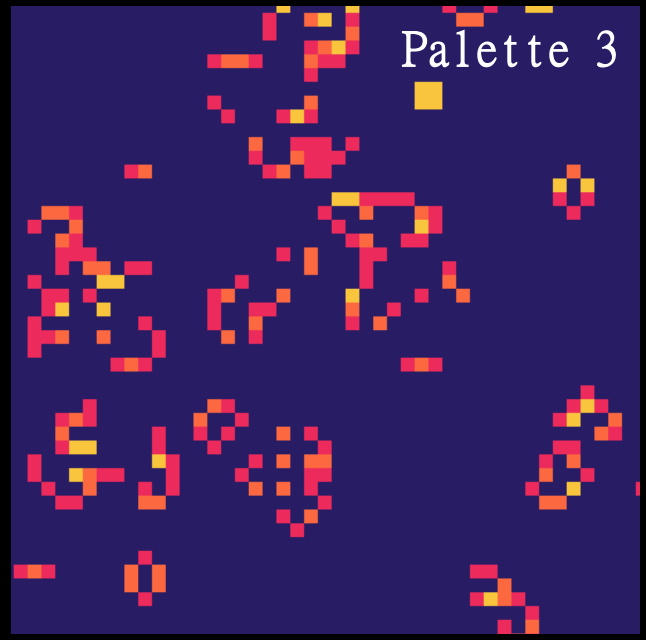


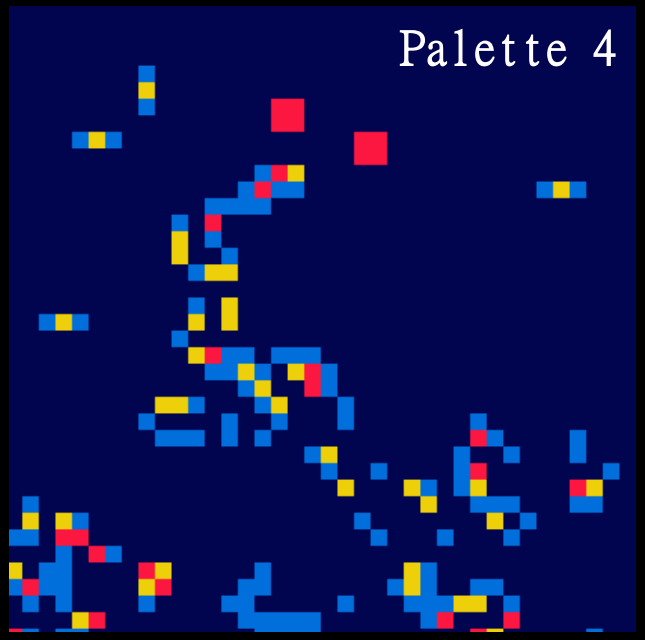
«Game of Life» mit mehr Zuständen

```
//(...)

//Palette 1
let color0 = [12, 0, 15]; // Dark purple
let color1 = [92, 20, 35]; // Dark red
let color2 = [191, 75, 43]; // Orange
let color3 = [238, 214, 118]; // Yellow
/*
//Palette 2
let color0 = [40, 28, 100]; // Dark purple
let color1 = [236, 43, 92]; // Pink
let color2 = [252, 104, 64]; // Orange
let color3 = [248, 197, 61]; // Yellow

//Palette 3
let color0 = [0, 10, 53]; // Dark purple
let color1 = [184, 21, 144]; // Pink
let color2 = [240, 92, 49]; // Orange
let color3 = [241, 251, 196]; // Yellow

//Palette 4
let color0 = [1, 4, 79]; // Dark blue
let color1 = [0, 111, 219]; // Blue
let color2 = [237, 208, 9]; // Yellow
let color3 = [252, 23, 65]; // Red
*/

function checkInput() {
    if (mouseIsPressed) {
        let x = Math.floor(mouseX / cellSize);
        let y = Math.floor(mouseY / cellSize);
        if (x >= 0 && x < cols && y >= 0 && y < rows) {
            for (let i = -1; i <= 1; i++) {
                for (let j = -1; j <= 1; j++) {
                    let col = (x + i + cols) % cols;
                    let row = (y + j + rows) % rows;
                    grid[col][row] = Math.floor(random(4));
                }
            }
        }
    }
}

//(...)
```



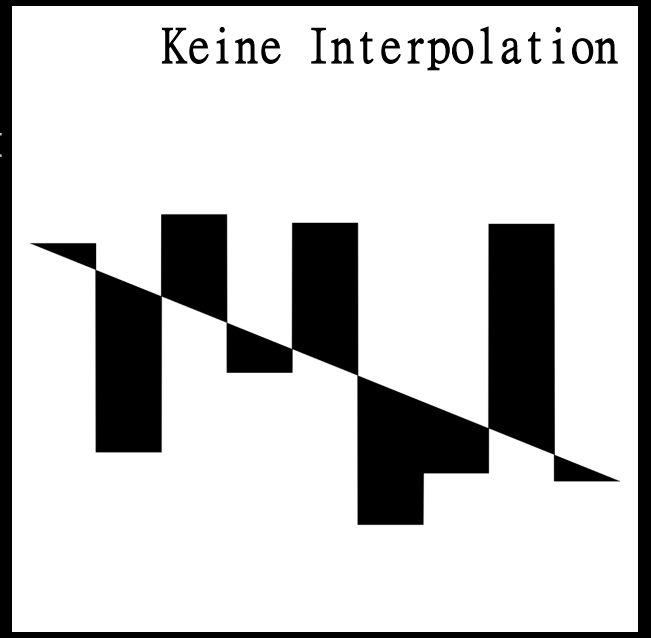


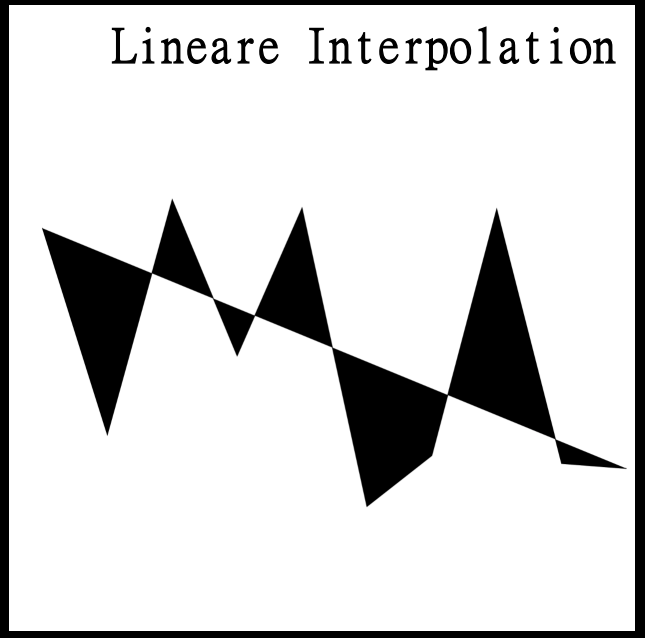


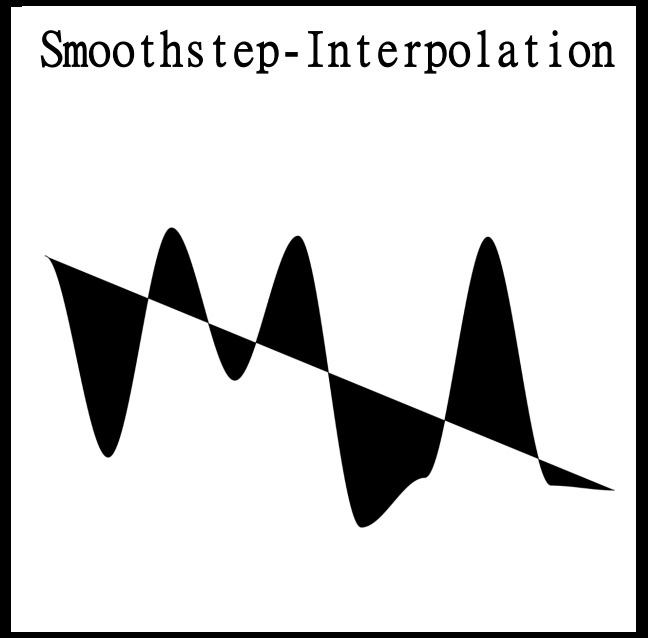


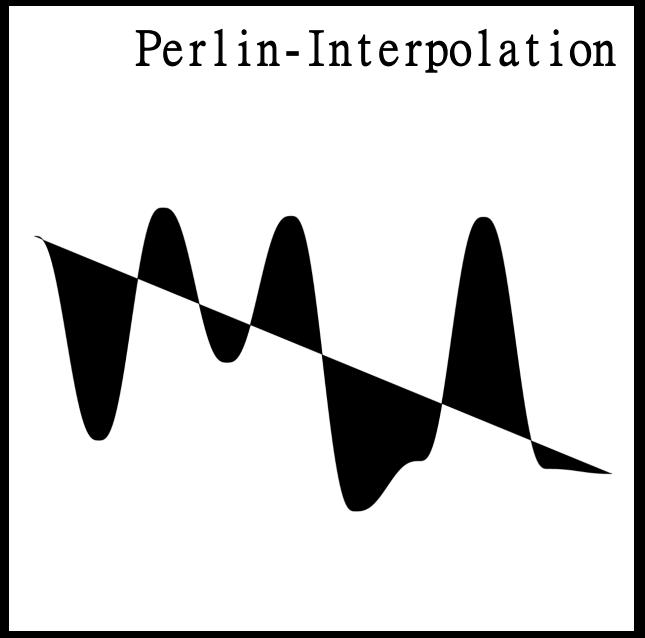
Experimente mit Farben & Interaktivität

```
//(...)

for (let i = 0; i < numPoints; i++) {
    values.push(random(height / 2));
}
drawInterpolatedPoints(values, 800, modes[mode]);

function drawInterpolatedPoints(values, yOffset, interpolationFunc) {
    beginShape();
    for (let i = 0; i < values.length - 1; i++) {
        for (let t = 0; t < resolution; t++) {
            let x = (width / numPoints) * (i + t / resolution)
                        + width / numPoints / 2;
            let y = yOffset - interpolationFunc(values[i],
                        values[i + 1], t / resolution);
            vertex(x, y);
        }
    }
    if (interpolationFunc == noInterpolation) {
        vertex((width / numPoints) * (values.length - 1) + width /
numPoints / 2, yOffset - values[values.length - 1]);
    }
    endShape();
}

function linearInterpolation(a, b, t) {
    return a * (1 - t) + b * t;
}
function cosineInterpolation(a, b, t) {
    let ft = t * PI;
    let f = (1 - cos(ft)) * 0.5;
    return a * (1 - f) + b * f;
}
function perlinInterpolation(a, b, t) {
    return a * (1 - (6 * Math.pow(t, 5) - 15 * Math.pow(t, 4) +
            10 * Math.pow(t, 3))) + b * (6 * Math.pow(t, 5) -
            15 * Math.pow(t, 4) + 10 * Math.pow(t, 3));
}
function smoothstepInterpolation(a, b, t) {
    t = t * t * (3 - 2 * t);
    return a * (1 - t) + b * t;
}

//(...)
```

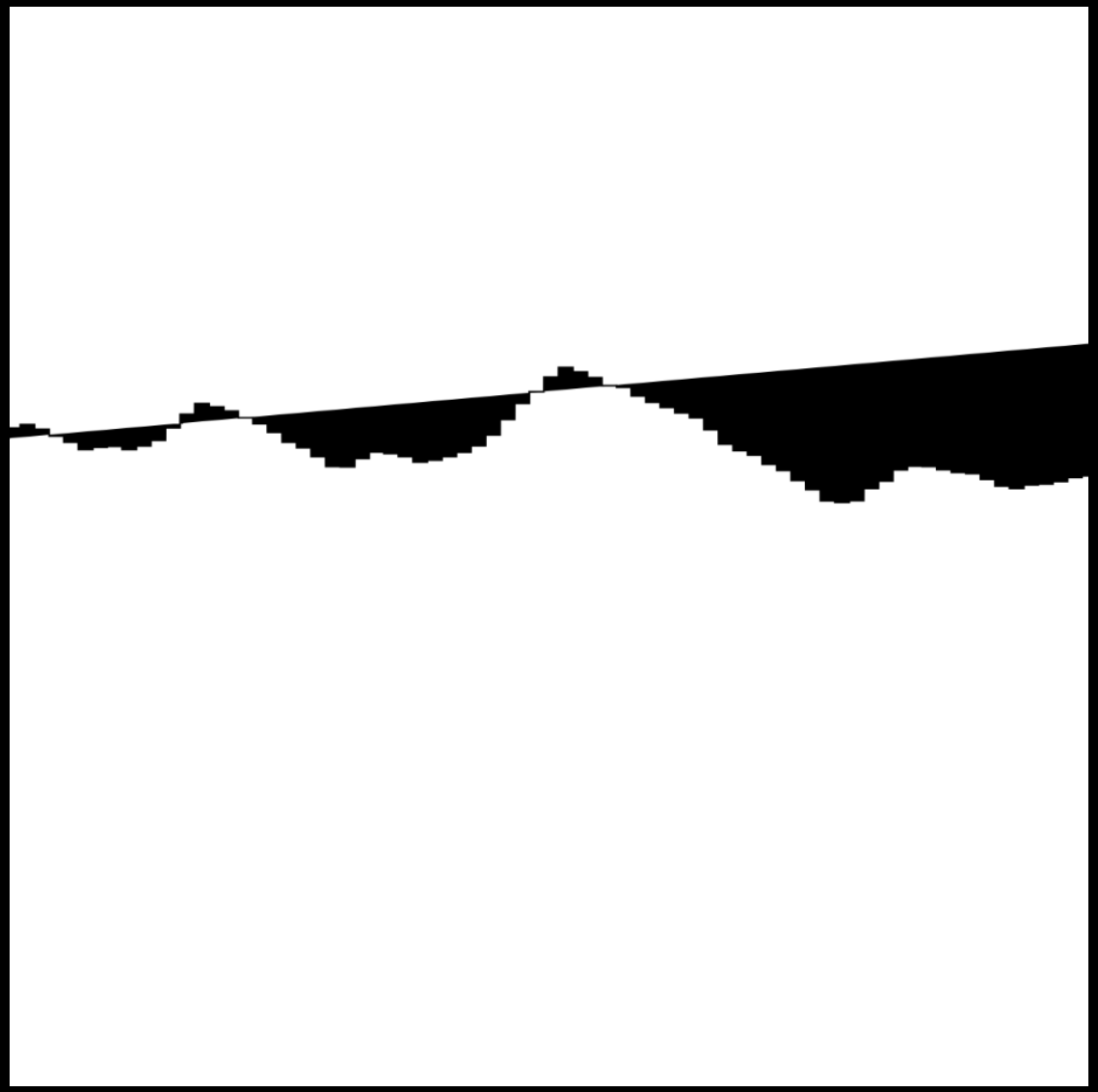








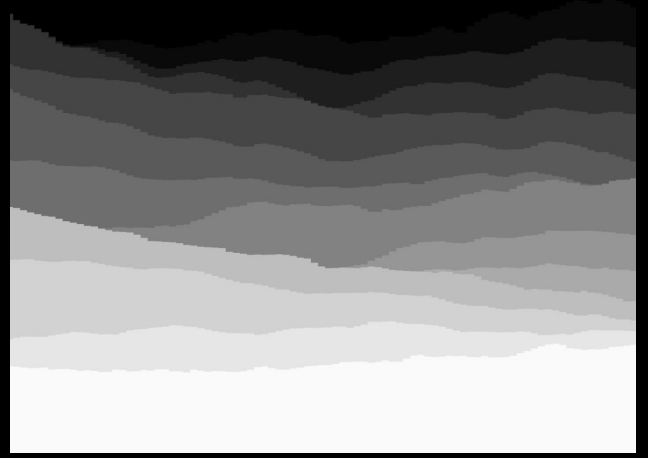
```
//(...)

let values = [];
let numPoints = 100;
let persistence = 0.5;

function setup() {
    let canvas = createCanvas(windowWidth, windowHeight);
    canvas.position(0, 0);
    stroke(0);
    fill(0);

    generateFractalNoiseValues();
}

function generateFractalNoiseValues() {
    values = [];
    for (let i = 0; i < numPoints; i++) {
        values.push(fractalNoise(i / numPoints) * height / 2);
    }
}

function fractalNoise(x) {
    let total = 0;
    let frequency = 1;
    let amplitude = 1;
    let maxAmplitude = 0;

    for (let i = 0; i < 8; i++) { // Number of octaves
        total += noise(x * frequency) * amplitude;
        frequency *= 2;
        maxAmplitude += amplitude;
        amplitude *= persistence;
    }

    return total / maxAmplitude;
}

//(...)
```



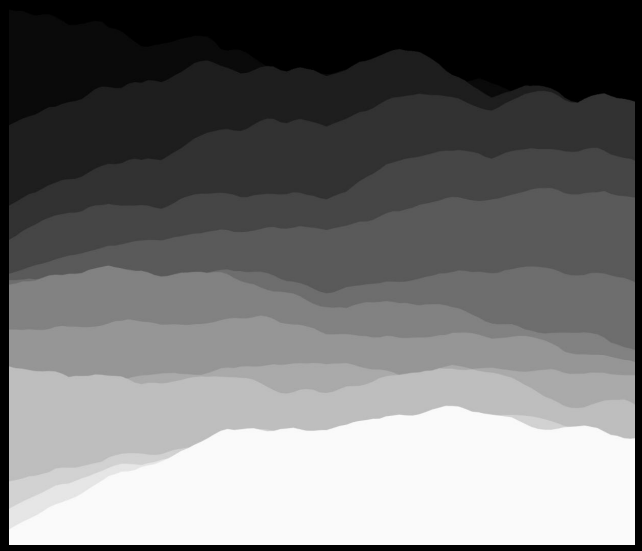
Fractal Noise mit Persistenz 1/2

```
//(...)

let tints = [
    [0.6, 1.2, 1.6], // Blue
    [1.2, 1, 0.5], // Yellow
    [1, 1, 1], // Gray
    [1.5, 0.5, 1], // Pink
    [1, 1, 1.5], // Purple
    [1, 1.5, 1], // Green
    [1.5, 1, 1], // Red
    [1.5, 1.5, 1], // Orange
    [1, 1.5, 1.5], // Teal
];
let tint = tints[0];

let colors = [
    [10, 10, 10],
    [20, 20, 20],
    //(...)
];

function setup() {
    let canvas = createCanvas(windowWidth * 1.1, windowHeight);
    canvas.position(-windowWidth * 0.05, 0);
    noStroke();

    background(0)

    generateFractalNoiseValues();
}

function draw() {
    for (let i = 0; i < colors.length; i++) {
        fill(colors[i]);
        // Draw interpolated points
        drawInterpolatedPoints(values[i],
            (i / colors.length + 1) * 500, modes[mode]);
    }
}

//(...)
```

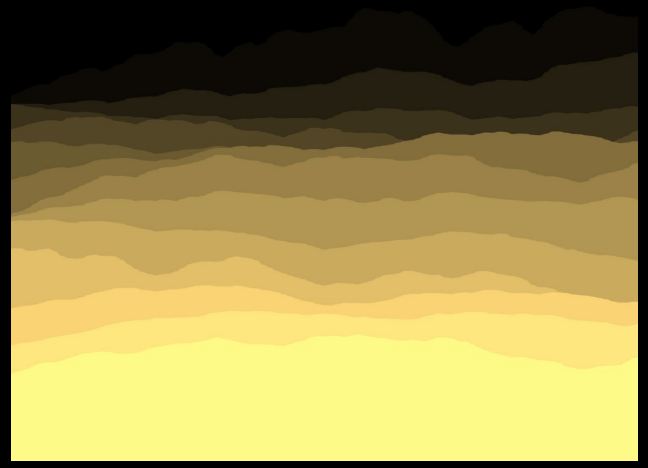
Keine Interpolation

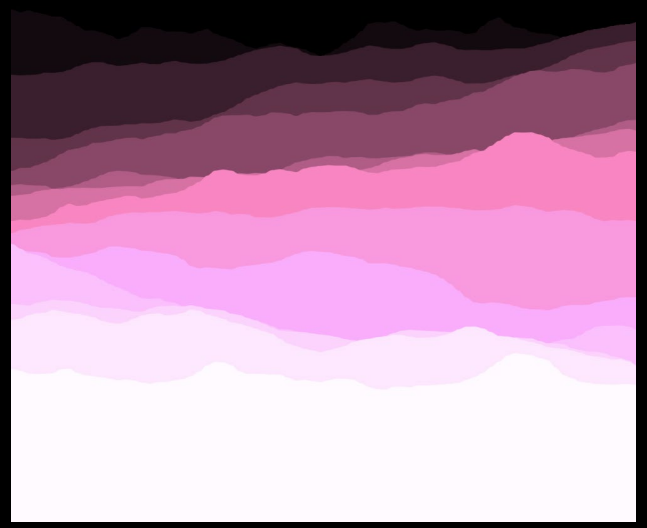


Lineare Interpolation



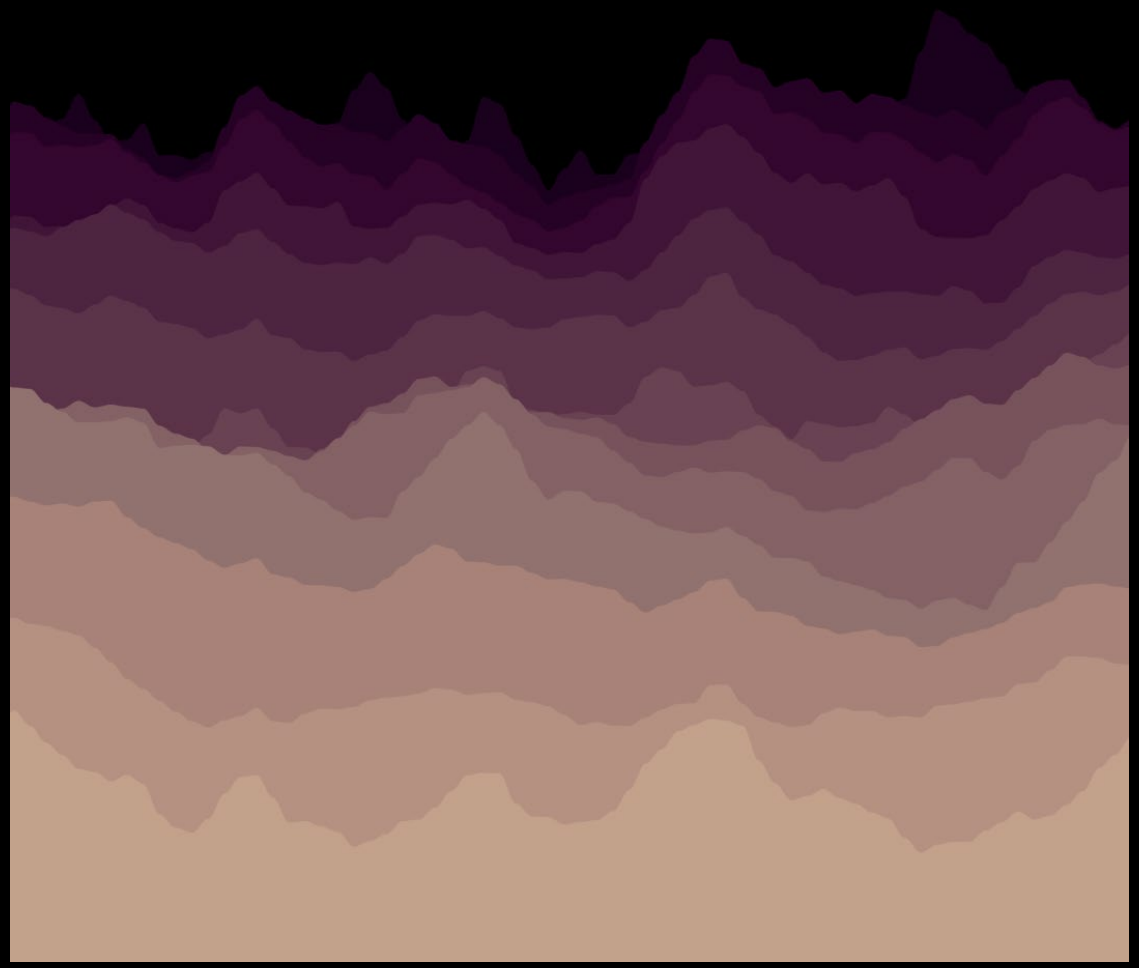
Varianten mit Smoothstep-Interpolation





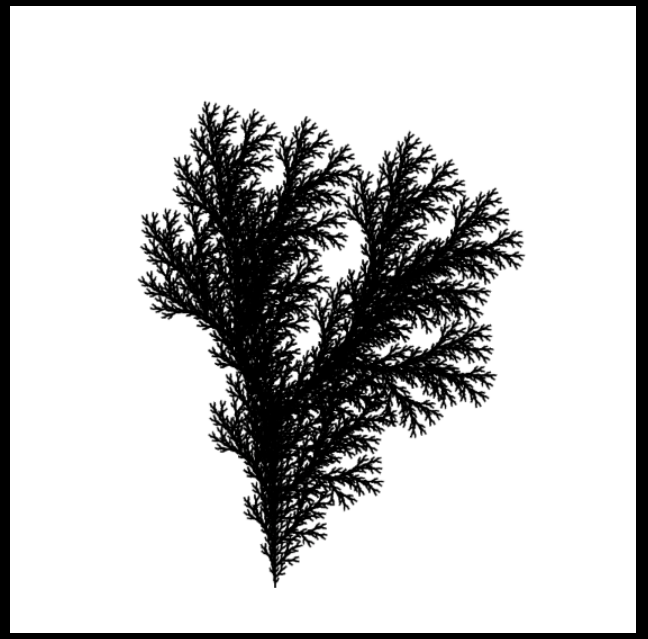
Layers durch 2D-Noise

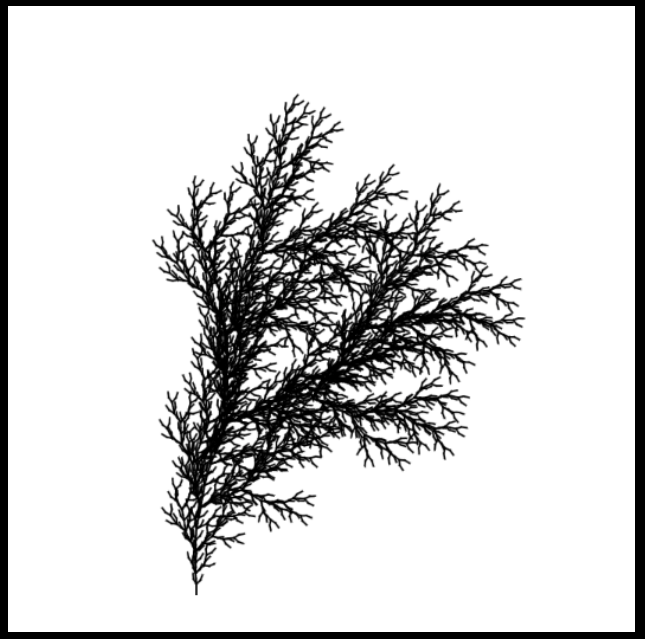
```
//(...)

function draw() {
    for (let i = 0; i < colors.length; i++) {
        colors[i] = [colors[i][0] * 0.97,
            colors[i][1] * 0.95, colors[i][2] * 0.96];
    }

    for (let i = 0; i < colors.length; i++) {
        fill(colors[i]);
        // Draw interpolated points
        drawInterpolatedPoints(values[i],
            (i / colors.length + 1) * 500, modes[mode]);
    }
}

function mouseClicked() {
    tint = tints[Math.floor(random(tints.length))];

    colors = [
        [30, 0, 50],
        [45, 0, 65],
        [60, 5, 80],
        [75, 30, 95],
        [90, 55, 110],
        [105, 80, 125],
        [120, 105, 140],
        [135, 130, 155],
        [150, 155, 170],
        [165, 180, 185],
        [180, 205, 200],
        [195, 230, 215],
        [210, 255, 230]
    ];

    for (let i = 0; i < colors.length; i++) {
        colors[i] = [colors[i][0] * tint[0],
            colors[i][1] * tint[1], colors[i][2] * tint[2]];
    }
}

//(...)
```

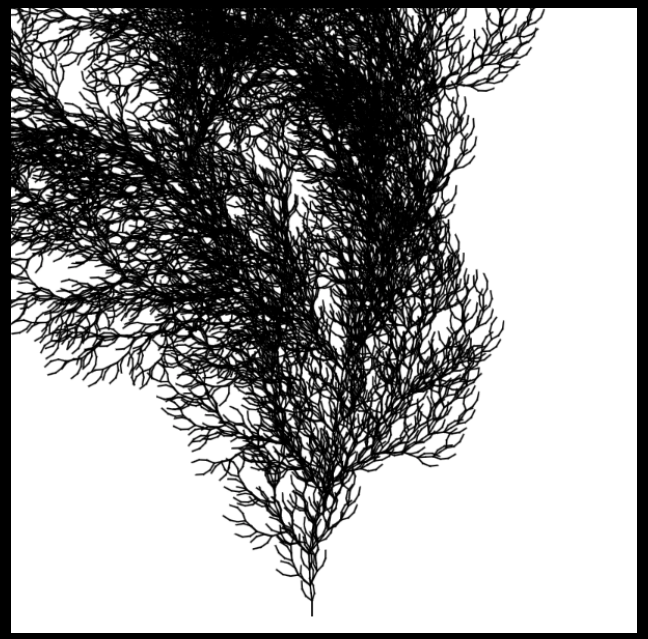


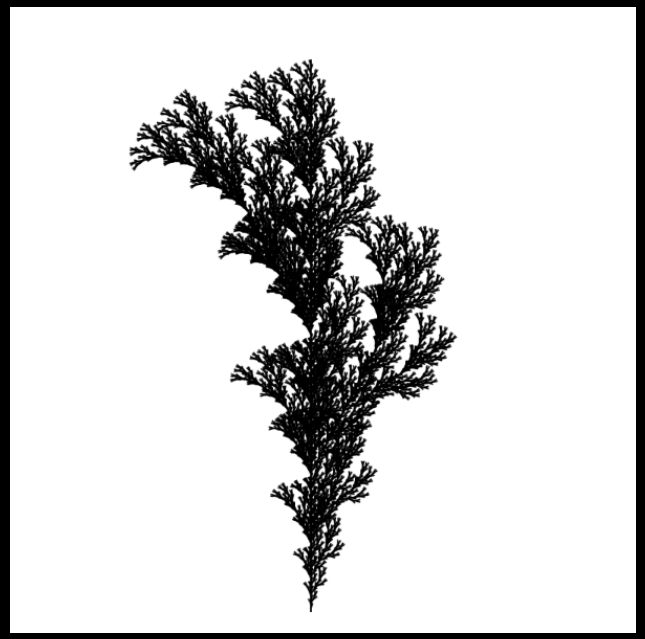
Farbverläufe & Fade-Out

```
//(...)

function generate() {
    let nextSentence = "";
    for (let i = 0; i < sentence.length; i++) {
        let current = sentence.charAt(i);
        let found = false;
        for (let j = 0; j < rules.length; j++) {
            if (current == rules[j].predecessor) {
                found = true;
                nextSentence += rules[j].successor;           }
        }
        if (!found) {
            nextSentence += current;
        }
    }
    sentence = nextSentence;
    len *= 0.55;
    turtle();
}

function turtle() {
    //(...)
    for (let i = 0; i < sentence.length; i++) {
        angle = radians(random(20, 30));
        let current = sentence.charAt(i);
        if (current == "F") {
            line(0, 0, 0, -len);
            translate(0, -len);
        } else if (current == "G") {
            translate(0, -len);
        } else if (current == "+") {
            rotate(angle);
        } else if (current == "-") {
            rotate(-angle);
        } else if (current == "[") {
            push();
        } else if (current == "]") {
            pop();
        }
    }
}

//(...)
```



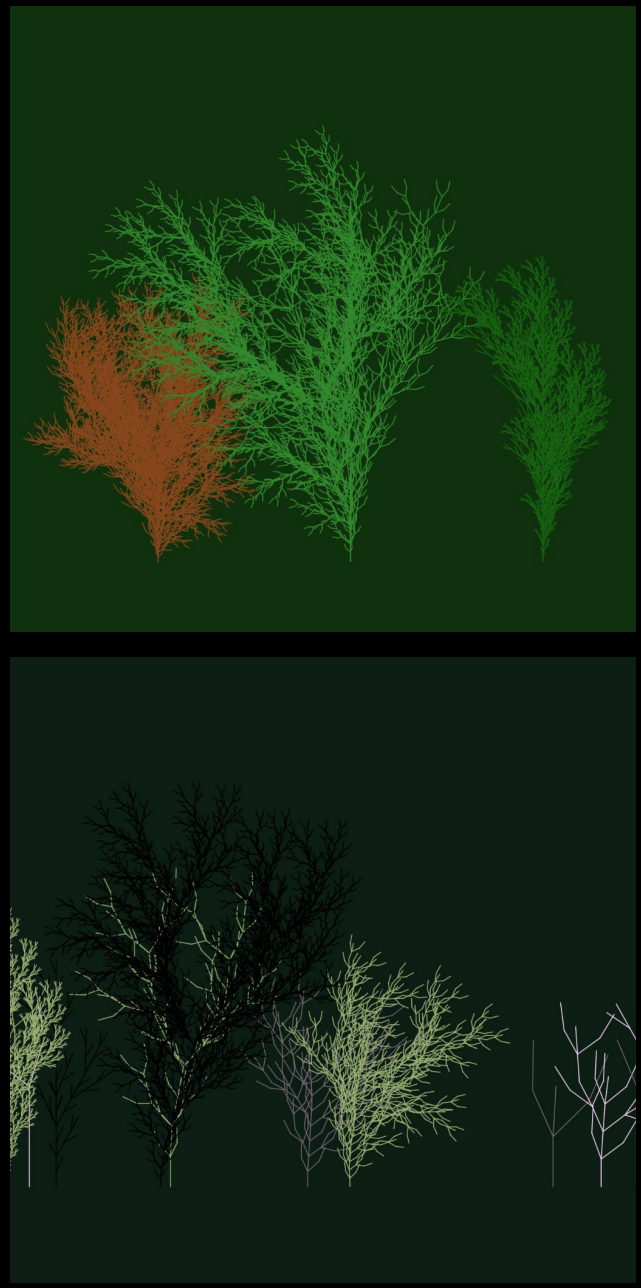






Baum mit L-System

```
//(...)

function setup() {
    let canvas = createCanvas(windowWidth * 1.1, windowHeight * 1.1);
    canvas.position(-windowWidth * 0.05, 0);
    angle = radians(12);
    background(10, 30, 20);
    nTrees = Math.floor(width / 20);
    let colors = [
        '#98B06F',
        '#776472',
        'black'
    ];
    for (let i = 0; i < nTrees; i++) {
        sentence = generateSentence(Math.floor(random(0, ruleSets.length)),
            Math.floor(random(3, 6)));
        len = 100 * Math.pow(0.55, 5);
        drawTree(sentence, random(-width / 2, width / 2),
            colors[Math.floor(random(0, colors.length))]);
    }
}

//(...)
```



Mehrere Bäume mit unterschiedlichen Farben & Rulesets

```
//(...)

let ruleSets = [
    [{ predecessor: "F", successor: "F+[+F-F]-[-F+F]F" }],
    [{ predecessor: "F", successor: "FF-[-F+F]+[+F-F]" }],
    [{ predecessor: "F", successor: "F[+F]F[-F][F]" }],
    [{ predecessor: "F", successor: "F[+F]F[-F]F" }],
    [{ predecessor: "F", successor: "F[+F[-F[+F]]]" }]
];

function setup() {
    let canvas = createCanvas(windowWidth * 1.1, windowHeight * 1.1);
    canvas.position(-windowWidth * 0.05, 0);
    angle = radians(12);
    background('#051f20');
    let colors = [
        '#0b2b26',
        '#163832',
        '#235347',
        '#8eb69b',
        '#daf1de'
    ];
    nTrees = 4;
    let initialStrokeWeight = 2.5;

    for (let layer = 0; layer < colors.length; layer++) {
        let strokeWeightValue = initialStrokeWeight - layer * 0.5;
        if (strokeWeightValue < 0.5) strokeWeightValue = 0.5;
        for (let i = 0; i < nTrees; i++) {
            sentence = generateSentence(Math.floor(random(0,
                        ruleSets.length)), 5);
            len = 100 * Math.pow(0.6, 5);
            drawTree(sentence, random(-width / 2, width / 2),
                        colors[layer], strokeWeightValue);
        }
    }
}

//(...)
```

Layers von Bäumen