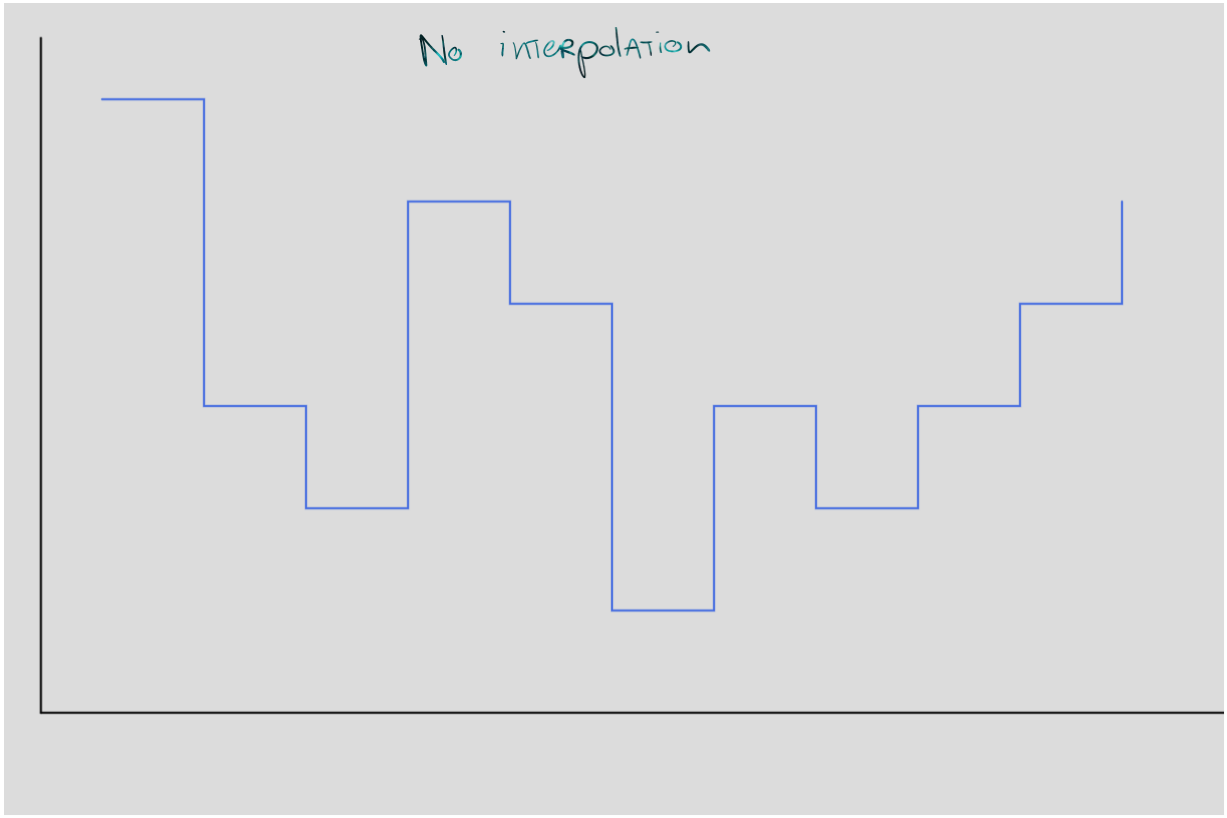
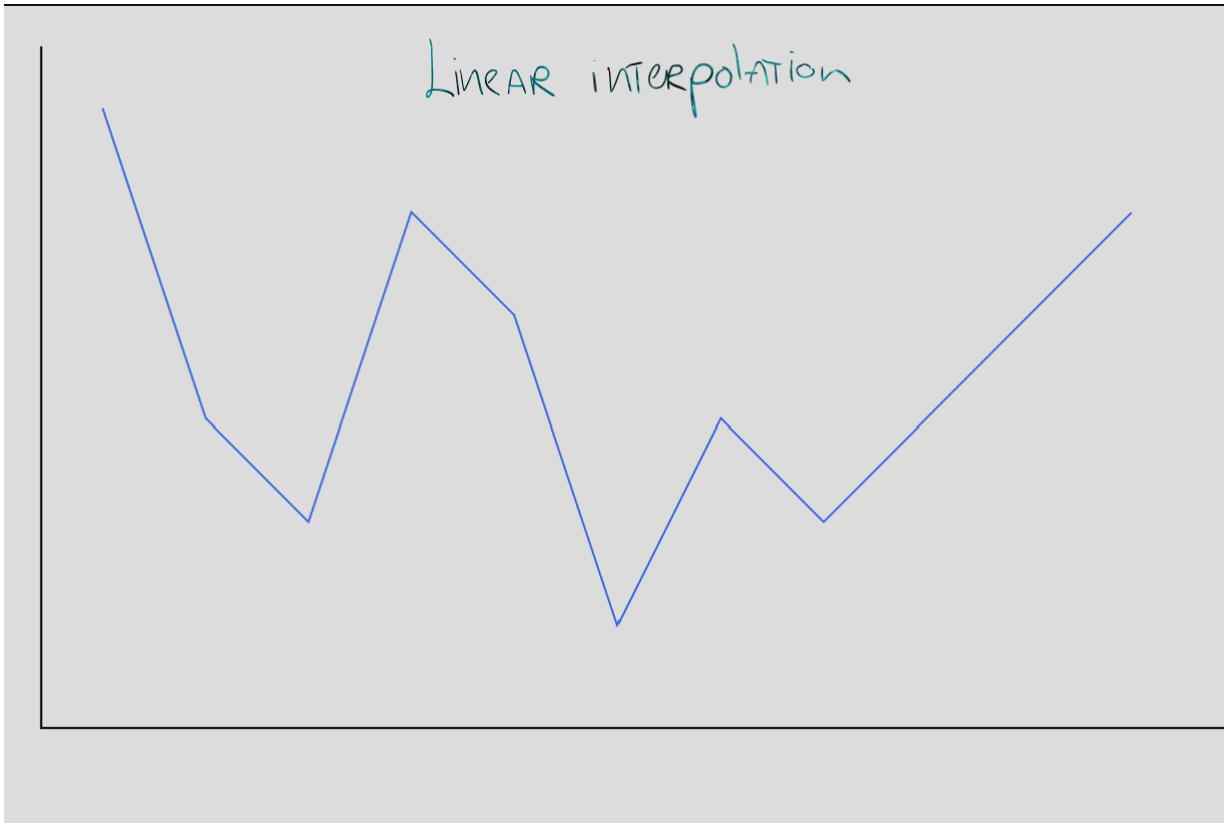
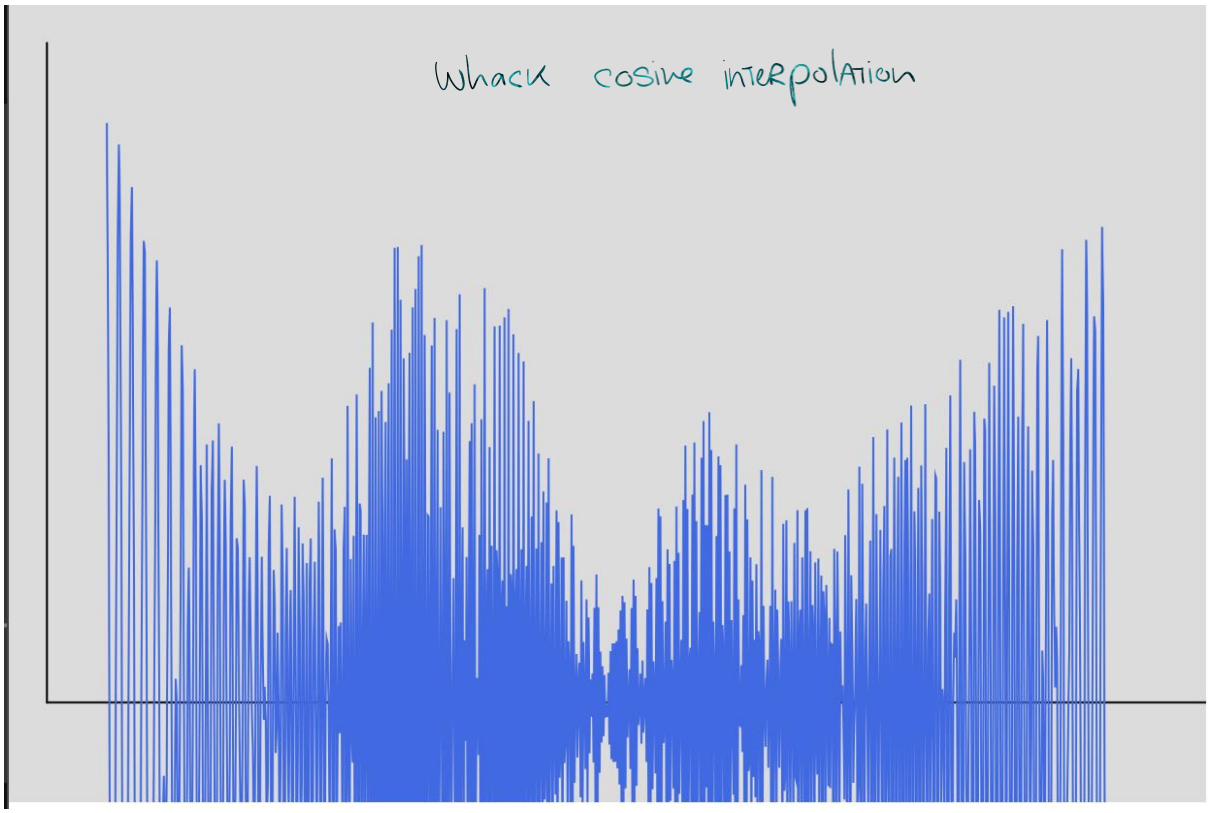
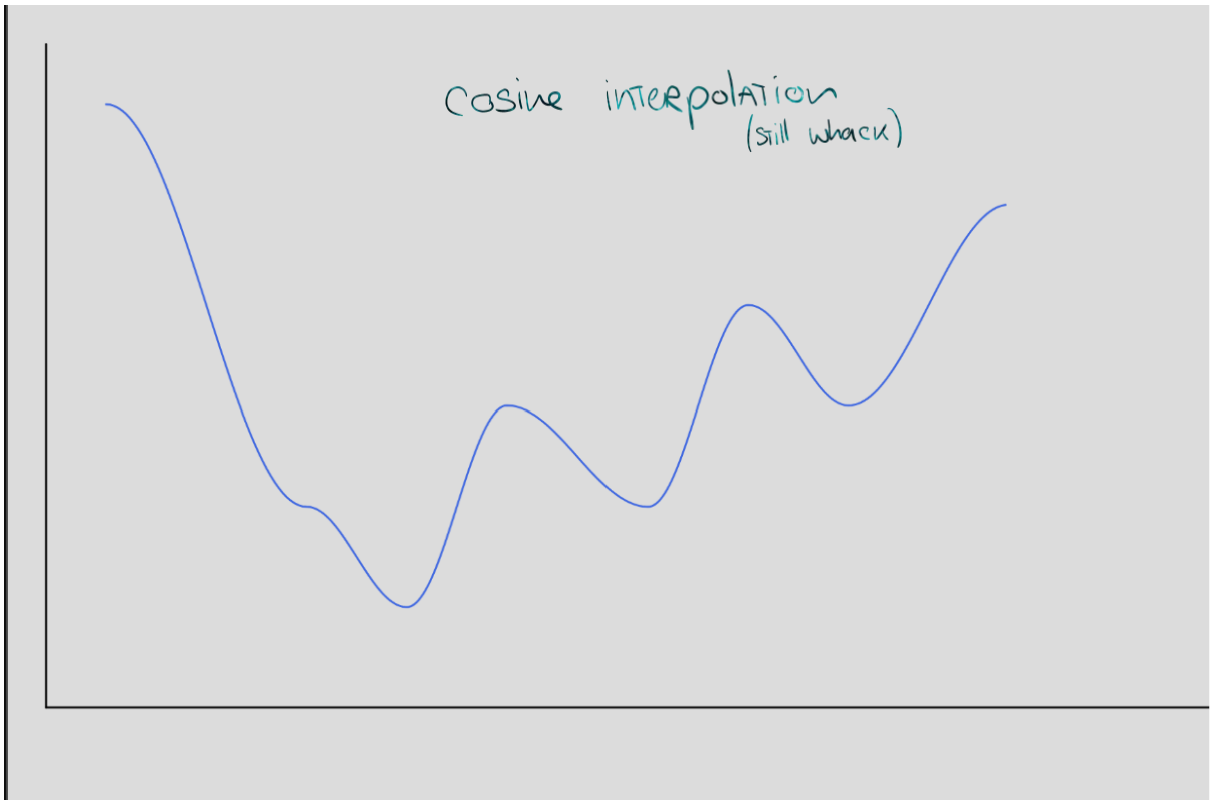
MATBF – Bergkette

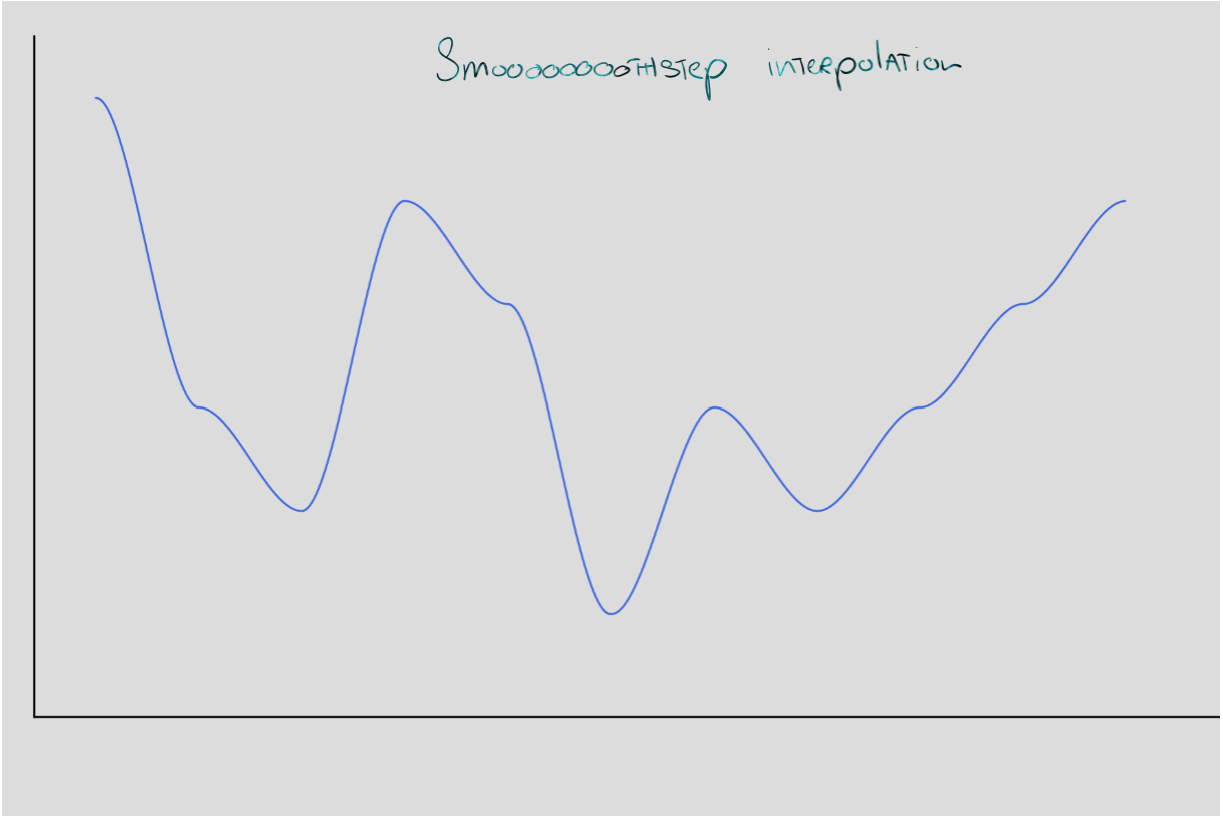
1) Interpolation






Whack cosine interpolation


Cosine interpolation
(still whack)



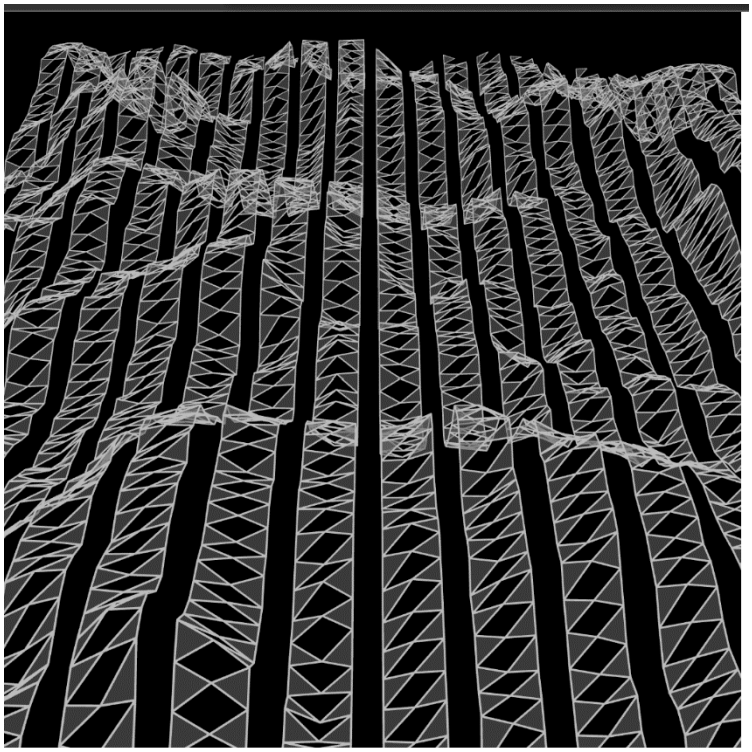
2) The weird case of cosine interpolation

sketch.js | Simple Browser

Cosine Interpolation > sketch.js > draw

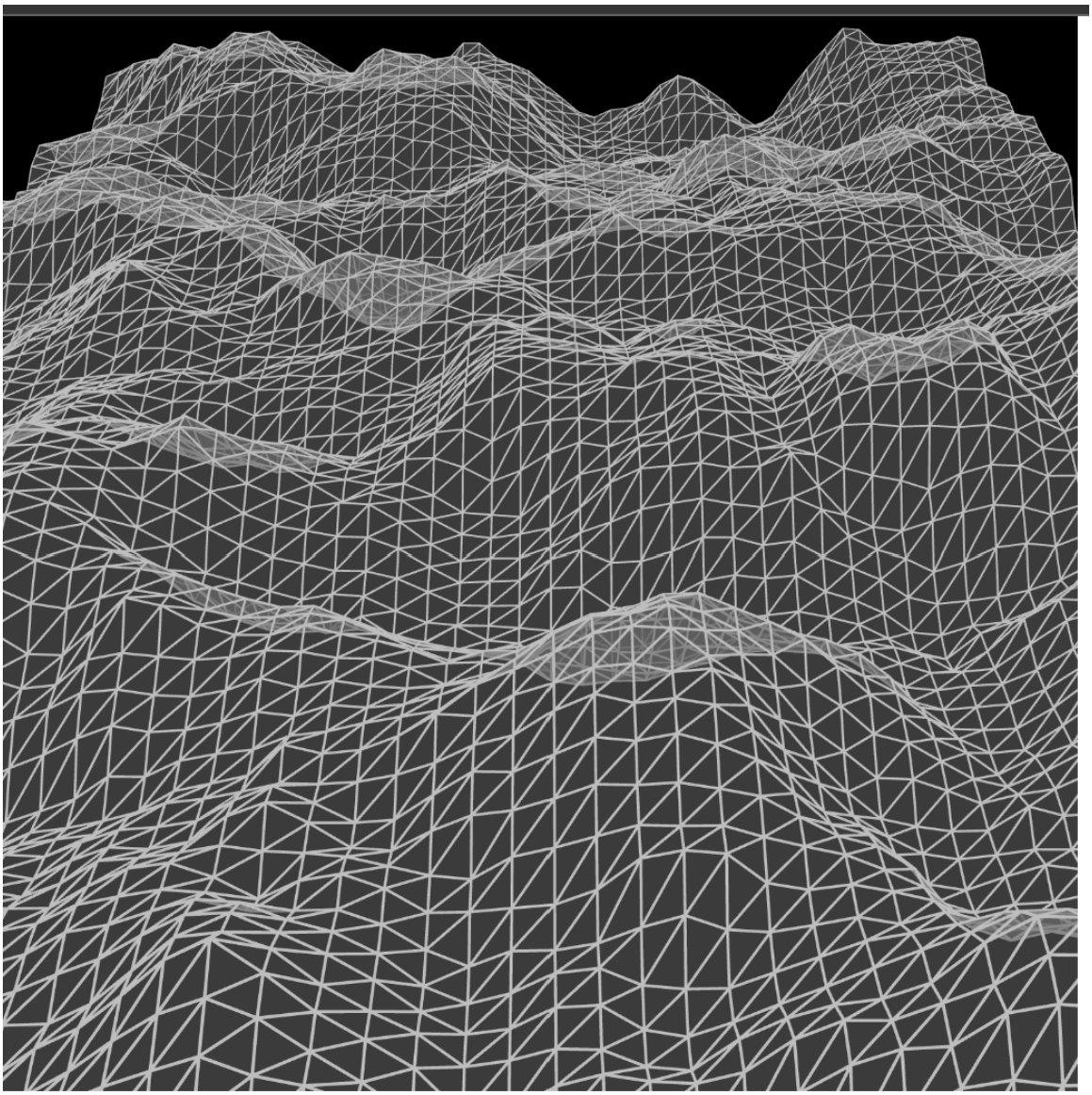
```
function draw() {
  let points = [
    {x: 50, y: 300},
    {x: 150, y: 100},
    {x: 200, y: 50},
    {x: 250, y: 150},
    {x: 320, y: 100},
    {x: 370, y: 200},
    {x: 420, y: 150},
    {x: 500, y: 250}
  ];

  // Achsen
  stroke(0);
  line(20, 350, 650, 350); // x-Achse
  line(20, 350, 20, 20);   // y-Achse

  // Zeichne den interpolierten Graphen
  stroke('royalblue');
  noFill();
  beginShape();
  for (let i = 0; i < points.length - 1; i++) {
    let p0 = points[i];
    let p1 = points[i + 1];
    for (let t = 0; t <= 1; t += 0.02) { // Ich habe die Schrittgröße auf 0.02 reduziert, um die Linien zu glätten
      let x = p0.x * (1 - t) + p1.x * t;
      let y = cosineInterpolation(p0.y, p1.y, t); // Kosinus-Interpolation für y-Werte
      vertex(x, 350 - y);
    }
  }
  endShape();
}

// By Copilot
function cosineInterpolation(y0, y1, t) {
  let t2 = (1 - Math.cos(t * Math.PI)) / 2; // Apply cosine interpolation formula
  return y0 * (1 - t2) + y1 * t2;
}
```

3) Bergkette



(TRIANGLES)

(TRIANGLE_STRIP)



JS sketch.js Smoothstep Interpolation | JS sketch.js Value Noise | JS sketch.js Bergkette | Simple Browser

Bergkette > JS sketch.js > ...

```
// Mit Hilfe von Chat GPT 4.0 & Perlin Noise Wikipedia: https://en.wikipedia.org/wiki/Perlin_noise

let cols, rows;
let fragments = 15;  // Skalierung der Dreiecke
let w = 800;  // Breite der Fläche
let h = 1800;  // Höhe der Berge

let terrain = []; //speichert die Höhenwerte des Terrains

function setup() {
  createCanvas(500, 500, WEBGL);
  cols = w / fragments; // Fläche in Fragmente unterteilen
  rows = h / fragments; // ''

  // Initialisiere das Terrain Array
  for (let x = 0; x < cols; x++) {
    terrain[x] = [];
    for (let y = 0; y < rows; y++) {
      terrain[x][y] = 0; // Alle Werte auf 0 setzen
    }
  }

  let yoff = 0;
  for (let y = 0; y < rows; y++) {
    let xoff = 0;
    for (let x = 0; x < cols; x++) { // nächste Zeile: generiert glatte, pseudo-zufällige Werte (Perlin-Noise)
      terrain[x][y] = map(noise(xoff, yoff), 0, 1, -100, 100); // Perlin Noise Werte in das Terrain Array schreiben
      xoff += 0.1;
    }
    yoff += 0.1;
  }
}

function draw() {
  background('black');
  stroke(190); // Kantenfarben
  fill(100, 100, 100, 150); // Bergfarben
  rotateX(PI / 3); // dreht die Landschaft um die x-Achse umd 3D-Perspektive zu erzeugen
  translate(-w / 2, -h / 2); // Zentriert die Landschaft

  for (let y = 0; y < rows - 1; y++) {
    beginShape(TRIANGLE_STRIP);
    for (let x = 0; x < cols; x++) {
      vertex(x * fragments, y * fragments, terrain[x][y]);
      vertex(x * fragments, (y + 1) * fragments, terrain[x][y + 1]);
    }
    endShape();
  }
}

```