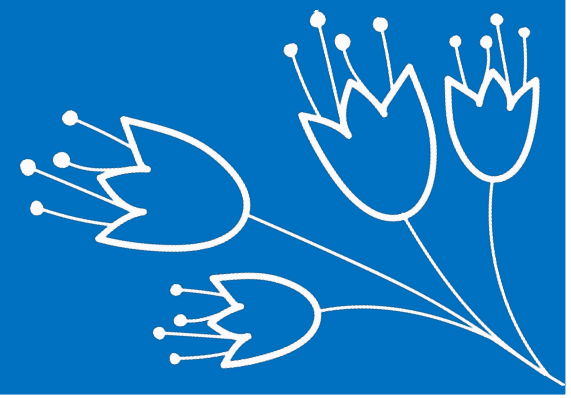

Mandelbrot Set

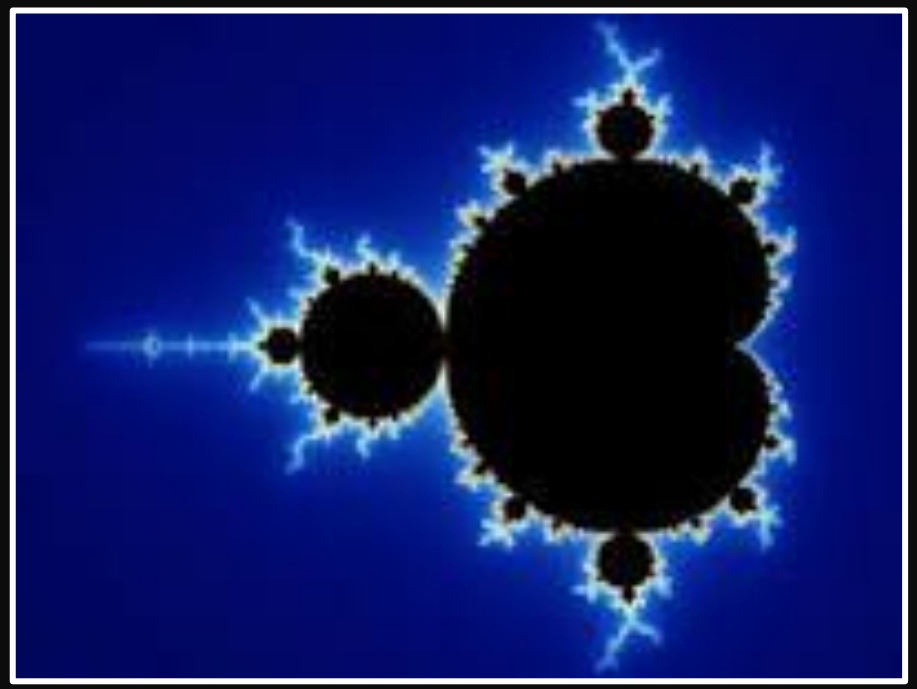
Presentation Main Points

- ✔ Introduction
- ✔ Math
- ✔ Drawing
- ✔ Interaction
- ✔ Color

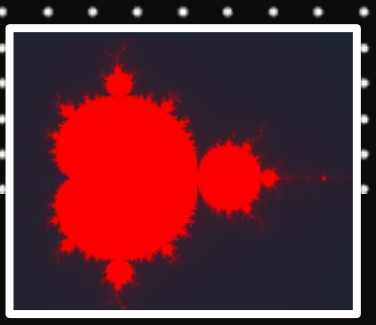


Introduction

The mandelbrot set is a set of complex numbers that when put into the function $z_{n+1} = {z_n}^2 + c$ recursively does not head towards infinity.



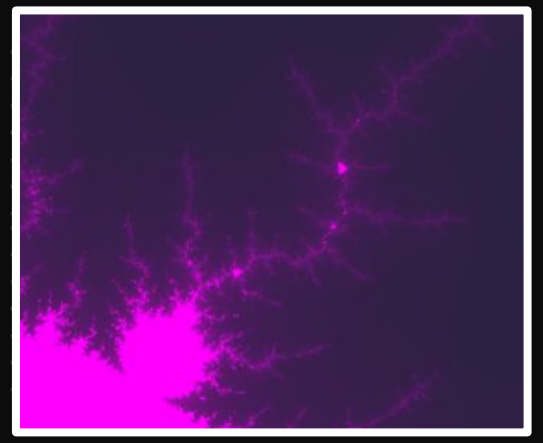
Math



- The Mandelbrot Set is drawn on a complex graph, in which x corresponds to the real number of a complex number and y corresponds to the coefficient of the imaginary number (x + y*i*)
- For each x and y value we use the complex number (c) in the function $z_{n+1} = {z_n}^2 + c$ with z starting as zero and increasing. If we find that as z increases the solution gets closer and closer to infinity, then that point on the graph is not part of the mandelbrot set and is not drawn.

```
function mandelbrot() {
  for (let i = 0; i < h; i++) {
    for (let j = 0; j < w; j++) {
      let x0 = scale(j, w, 0, xMin, xMax);
      let y0 = scale(i, h, 0, yMin, yMax);
      let x = 0;
      let y = 0;
      let iteration = 0;
      let xTemp = 0;

      while (x * x + y * y <= 2 * 2 && iteration < maxIter) {
        xTemp = x * x - y * y + x0;
        y = 2 * x * y + y0;
        x = xTemp;
        iteration++;
      }
```



Drawing

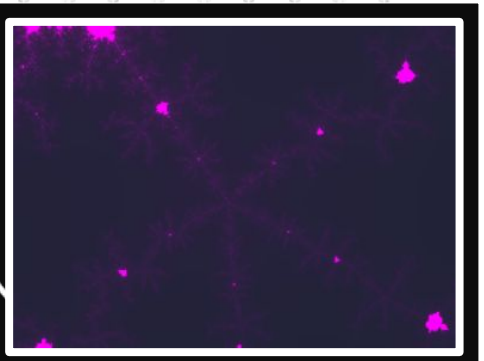
- When drawing the mandelbrot set we need to keep in mind the scale
- By having a value that corresponds to the upper and lower limits of x and y in our “graph”, we can enable movement within the graph by changing these individual values without having to mess with the object that contains the actual image we are drawing.
- For improved speed I am using the image data object, which is an array that contains the individual r,g, and b values for each and every pixel

```
let xMin = (-5 * w) / 100; //-2;
let yMin = (-5 * h) / 100; //-1.12;
let xMax = w / 100; //4.7;
let yMax = h / 100; //1.12;
let imageData = ctx.createImageData(w, h);
```

```
for (let i = 0; i < imageData.data.length; i += 4) {
  // Modify pixel data
  imageData.data[i + 0] = 255; // R value
  imageData.data[i + 1] = 255; // G value
  imageData.data[i + 2] = 255; // B value
  imageData.data[i + 3] = 255; // A value
}

function changePixel(x, y, r, g, b) {
  // Modify pixel data
  imageData.data[y * (imageData.width * 4) + x * 4] = r; // R
value
  imageData.data[y * (imageData.width * 4) + x * 4 + 1] = g; // G
value
  imageData.data[y * (imageData.width * 4) + x * 4 + 2] = b; // B
value
  imageData.data[y * (imageData.width * 4) + x * 4 + 3] = 255; //
A value
}
```

Interaction



- Compared to everything else, movement is fairly simple, I just need individual key presses to change the corresponding upper and lower bounds of x and y to “move” around the graph and zoom in and out
- By changing the number of max iterations I can also increase and decrease the transition between two colors

```
switch (event.code) {
  //increase range
  case "Minus":
    xMin -= xRatio;
    xMax += xRatio;
    yMin -= yRatio;
    yMax += yRatio;
    mandelbrot();
    break;
  //decrease range
  case "Equal":
    xMin += xRatio;
    xMax -= xRatio;
    yMin += yRatio;
    yMax -= yRatio;
    mandelbrot();
    break;
```

```
//decrease clarity
case "BracketLeft":
  maxIter--;
  mandelbrot();
  break;
//increase clarity
case "BracketRight":
  maxIter++;
  mandelbrot();
  break;
```

```
//move right
case "KeyA":
  xMin += xRatio;
  xMax += xRatio;
  mandelbrot();
  break;
//move left
case "KeyD":
  xMin -= xRatio;
  xMax -= xRatio;
  mandelbrot();
  break;
//move up
case "KeyS":
  yMin -= yRatio;
  yMax -= yRatio;
  mandelbrot();
  break;
//move down
case "KeyW":
  yMin += yRatio;
  yMax += yRatio;
  mandelbrot();
  break;
```

Color

- By using the lerp function which interpolates between two numbers I can have the mandelbrot drawing transition between two different colors, with one color representing a point that reaches infinity instantly, and the other representing a point that never reaches infinity

```
changePixel(
  j,
  i,
  lerp(28, 255, iteration / maxIter),
  lerp(35, 0, iteration / maxIter),
  lerp(51, 0, iteration / maxIter)
);
```

```
function lerp(a, b, x) {
  return a + (b - a) * x;
}
```