The Mandelbrot Set Code

When I saw the complex fractal appear on the screen upon loading up the document, I had a moment of semi-recognition. On the one hand, the fractal seemed eerily familiar, like something one would have seen before - in a fever dream or on a dose of psilocybin. Yet, I did not entirely recognise it. So as a first step, I decided to investigate what the fractal meant and visualised.

As I understand it, when most integers are squared, and then heir square squared, and then the square of the square squared and so on, in an expression given by *f(z) = z^2*, they tend forward on the number line towards infinity with each subsequent gap on the number line increasing exponentially. On the other hand, decimal values between 1 and (-1) tend to 0. If instead, each time we add a fixed number to the product upon squaring z, the resultant graph for all values between 1 and (-1) start showing some very interesting behaviour. The code is an attempt to visualise this behaviour given by the expression $z_{n+1} = z_n^2 + c$ where c is a number in the complex plane such that $z_0 = c$. The visualisation maps out the set of numbers C for whom z_n does not tend to infinity.

The Code

Setup and Variables

The code starts out simple. A fixed position canvas with the top-left corner at (0,0) is created in HTML. The first bit of the JavaScript sets up the canvas much like some of the previous exercises.

The Mandelbrot set is generally given by the expression $z = z^2 + c$. Since the programme maps out the Mandelbrot set (in black) in terms of the colour of each pixel, it runs a test based on every pixel's location.

Global variables z (absolute value of z), zx and zy (x and y co-ordinates for z) and cx and cy (x and y co-ordinates for c) are declared as vars and initialised to 0. Two values each are required for z and c as they are numbers in the complex plane (two-dimensional numbers) and correspond to pixel locations mapped out on a 2D cartesian space.

The var maxIterations is declared and initialised to 50. This gives the total number of iterations performed on each pixel. The rule is that if the value from the initial number z_n goes outside the Mandelbrot set range from (-2) to (2) after 50 iterations, the pixel is visualised in white as opposed to black. Changing this value changes the fidelity of the result.

The var res is set to 2 and is used as the value of the increment in the for-loops. The line context.translate(width*0.5,height*0.5); sets the origin (0,0) to the middle of the canvas.

The Test/Main Calculation Loop

The main loop starts out like other 2D convolution techniques using a for-loop to go through each pixel in a row nested within a for-loop to go through each column of the canvas. The integers i and j are initialised to index the 2D grid with each of them going from the negative half-width to positive half-width in increments of 2 (res) per run.

The X and Y co-ordinates of c are set first by calling the expressions cx = i / (width * 4) and cy = j / (width/4). This is because the value of complex number c needs to be between (-2) and 2 in order to correspond to the Mandelbrot set.

Next, a for-loop is called to calculate the value of z. The loop is run 50 times (or maxIterations times). In every run, the absolute value of z is calculated by calculating the x and y co-ordinates for z by adding the X and Y co-ordinates of c to the previous value of z. (The value of z is calculated using the values of zx and zy (initially 0). These values are added to cx and cy and saved as the new values of zx and zy.)
z is used to determine if anything is to be done to the pixel. The pixel is kept white in the next run if the value of z exceeds 2 or (-2) and coloured in greyscale if it remains within the range.

First, an if statement is called to see if the absolute value of z has exceeded 2. If it has, nothing else in the for-loop is executed meaning the pixel on the canvas remains white as it was when the canvas was initialised.

If, however, the value of z is still under 2, the code for calculating the value of z for the next iteration is run and the code to colour the pixel on the greyscale is executed. This is done by initialising new vars x and y given by $x = zx^2 - zy^2$ and $y = 2(zx.zy)$. These values give the x and y co-ordinates of the square of z (z^2). The z for the next run of the expression $z = z^2 + c$ is then calculated by adding the x and y co-ordinates of c to these vars x and y in the lines zx=x+cx; and zy=y+cy;
The absolute value of z is calculated by taking the square root of the addition of the squares of each x and y co-ordinate using the line z=Math.sqrt((zx*zx)+(zy*zy)); from zx and zy. These are saved for the next run through the for-loop and get updated again and again till the loop runs through itself 50 times. If, instead, z exceeds 2.0, the values of z, zx and zy are reset to zero at the end of the for-loop to 'reset' them before performing the test on the next pixel.

The colour of the pixel is then calculated based on the iteration number the run is in. The value of col is used alike in the R, G, and B arguments to set the value on the greyscale between (0,0,0) and (255,255,255) and the pixel is coloured by setting its fillStyle.

Colouring in the Canvas

Finally, the lines context.beginPath();, context.rect(i,j,res,res);, and context.fill(); fill in each pixel on the canvas using the fillStyle set for it to create the complex visualisation of the Mandelbrot set.

Notes
All the lines of code directly quoted from the programme are typed in green.
All the variables that are directly quoted from the programme are typed in violet.
All the mathematical expressions used to explain the programme are typed in Italics.