ADVANCED CREATIVE CODING (SEMESTER 1)
WEEK-BY-WEEK ASSIGNMENTS SUBMISSION- ADVAIT UKIDVE

WEEK 1

Link: https://mimicproject.com/code/091bfd4a-9870-618c-3d5d-91dfa0899137
About My Process: Having never learned JavaScript before and only limited understanding of HTML-CSS, I was a little late to the game in the first week. After creating a few soundscapes that sounded eerily familiar to narrator in the 1984 audio book, I created this project. It contains 4-5 different iterations of the same basic combinations of sound with each, in order, getting more and more complex.

WEEK 2

Link: https://mimicproject.com/code/c4195187-1e96-bd33-949c-aadbc8088055
About My Process: Having spent all of my first week and most of my second trying to get week 1 right, I was in an odd mental space. So, for Week 2, I decided to represent this chaos by making a noise piece of sound with ugly sounds adding together randomly in a visualisation of the grey noise within. I used sounds such as guns firing and cocking, bells ringing loudly, and drawers shutting to create a frustrating soundscape using the same techniques I could've put to making something sound great (possibly). So as much as I'd like to wish you enjoy, the idea is DON'T hehe.

WEEK 3

Link: https://mimicproject.com/code/b620004d-5d41-dd52-7316-939fdf16be7c
About My Process: Week 3 was one of, if not the absolute, favourite week I had. Having been raised by two architects, it was fun to return to the Flores Geometrici with the computer making studying its form so much more fun. For the week's homework, after experimenting a little with the algorithm, I concluded no one form I make could justify the vast possibilities that two simple lines of code hold. So, I decided to make a geometry randomiser. Along the way I picked up how to make HTML sliders and checkboxes and have them pass over data to the JavaScript code.

WEEK 4

Link: https://drive.google.com/file/d/1IFZ0-snb_BJqqLCWDStNQLmLnKJ_FX3I/view?usp=sharing
About My Process: For Week 4, I decided to do the writing assignment out of the two as I found the fractal patterns very intriguing. I wrote 1000 words and have attached a link to the PDF hosted on my personal drive and the raw text at the bottom of this document. What I found most intriguing about the document is how it uses a filter to assign values on the grayscale over several iterations. It's not a technique I had encountered before although it made sense instantly. The week also led me to a healthy dose of research on numbers in a complex space and wrapping my head around some of the beautiful visualisations that can come out of it!

WEEK 6

Link: https://mimicproject.com/code/4164147d-f1e4-7cd2-7e24-b1a27033fc87
About My Process: For the introductory week to 3D, I decided to make a quick fork of the document used in the lecture and experiment with all that it can do once I tweak some values. Then I used it to create a simple continuously rotating form that changes its curvature, orientation, and rate of spinning based on the mouse position. One of the more interesting details of this code is using the Math.random() function strategically while assigning the line width to create a sense of a flicker leading to a visual, I found reminiscent of the Manchester United home kit from the Premier League season of 2020-21. I also experimented with linear gradients for the first time, and it was really satisfying.

WEEK 7

Link: https://mimicproject.com/code/3df8986c-3090-4519-468a-656a4482272f
About My Process: Week 7 and the Introduction of threejs was my favourite. Having done industrial design and CAD for my undergraduate degree, I just instantly 'got' threejs and what it stood for although, happily, it took a lot longer to explore what all it could do (Leading me to use it in my final project). I decided to make a 3D scene representing my experience working in my room over the last couple weeks due to running a slight fever and how it had gotten increasingly mundane and one-dimensional. The scene is an absurdist viewpoint of what it felt like with – an endless loop of time as represented by the spinning top – alongside trying to get work done (making notes) while being constantly distracted by a laptop brimming with possibilities of entertainment (cat videos) while still stuck in a matrix-like prison.

WEEK 8

Homework 1

****** TRIGGER WARNING: Homework 1 and Homework 2 contain flashing lights. Please do not view if you are prone to photosensitive epilepsy. I have set it to not 'play on open'. *****

Link: https://mimicproject.com/code/82090c45-cfcf-a2b4-ec5c-edca296ae484
About My Process: For the first homework of the week, I decided to use the final example provided on Moodle with sinewaves originating from the centre and explore some of the other operators used in GLSL. I was particularly interested in fract() and used it to calculate the petals and to create a pulsating effect by modifying the green value in the FragColor. I also used the floats of time and mouse X and Y position to change the colour of the result as the mouse moves and time passes. What I find the most fascinating about the result is the way the shapes melt into themselves in pulses as they change colour.

Homework 2

Link: https://mimicproject.com/code/a98f397a-acaa-add7-80fb-caa87c1ec6b8
About My Process: For the second homework, I decided to take the same ideas applied in Homework 1 further to create three asynchronous flashing movements in what turned out to look like an acid dose representation of techno music I listened to in Goa. I'm not too big on techno so don't hold me to it. The first uses the position of the mouse and the time it has passed since the program begun passed through a sin curve to create rotation that builds up speed in the upper two

quadrants and slows down in the bottom while changing its direction of rotation based on whether the mouse is in the left two quadrants or the right two. The second uses the pulsating effects of the squares. The third is rooted in using functions such as sin, cos, tan, exp and fract while assigning colours to all the shapes on the canvas.

WEEK 4 ESSAY TEXT

The Mandelbrot Set Code
-Advait Ukidve

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.