GROUP 17

QUESTION 1

The mandelbrot set is an example of fractal geometry, which is a branch of mathematics that studies irregular, infinitely complex, and self-similar structures found in nature, mathematics, and computer-generated imagery. The mandelbrot set defines a set of all complex numbers 'C' for which the sequence $f_c(z) = z^2 + C$ does not escape to infinity. I.e it remains bounded.

The iterative process to define the set is by:
i) We start with a complex number C.
ii) We iterate it through the function starting with 0.
iii) For each point c in the complex plane, the algorithm checks how many iterations it takes for to exceed a threshold (usually 2)
iv) If it does not escape to infinity, *c* is considered part of the Mandelbrot set else it is not and the number of iterations determines the pixel's color, creating intricate fractal patterns.

Some fractal properties of it include:
Self- similarity as zooming in to the edges reveal smaller copies of the larger structure
Infinite complexity as new details emerge when zoomed
Boundary complexity where the boundary is infinitely detailed and follows fractal dimensions.

A use of it in computer graphics would include creating visual images using colour gradients based on iteration counts, it is used to simulate natural phenomena (e.g., clouds, terrains) and also creating psychedelic animations.

QUESTION 2
Code in OpenGL

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <cmath>
#include <complex>
#include <iostream>
#include <vector>

// Image dimensions
const int IMAGE_WIDTH = 1024;
const int IMAGE_HEIGHT = IMAGE_WIDTH / 2;
const int MAX_ITER = 1000;

// Global pixel buffer (RGB, 8 bits per channel)
std::vector<unsigned char> pixels(IMAGE_WIDTH * IMAGE_HEIGHT * 3);

// Convert HSV to RGB (values in [0,1])
void hsvToRgb(float h, float s, float v, float &r, float &g, float &b) {
```

```
    int i = int(h * 6);
    float f = h * 6 - i;
    float p = v * (1 - s);
    float q = v * (1 - f * s);
    float t = v * (1 - (1 - f) * s);
    i %= 6;
    if (i == 0) { r = v; g = t; b = p; }
    else if (i == 1) { r = q; g = v; b = p; }
    else if (i == 2) { r = p; g = v; b = t; }
    else if (i == 3) { r = p; g = q; b = v; }
    else if (i == 4) { r = t; g = p; b = v; }
    else if (i == 5) { r = v; g = p; b = q; }
}

// Get RGB color for a given iteration count.
void getRGBColor(int iter, unsigned char &R, unsigned char &G, unsigned char &B) {
    const float baseHue = 0.6f;
    float hue = fmod(baseHue + iter / 255.0f, 1.0f);
    float r, g, b;
    hsvToRgb(hue, 1.0f, 0.5f, r, g, b);
    R = static_cast<unsigned char>(r * 255);
    G = static_cast<unsigned char>(g * 255);
    B = static_cast<unsigned char>(b * 255);
}

// Mandelbrot function
void mandelbrotColor(double real, double imag, unsigned char &R, unsigned char &G, unsigned
char &B) {
    std::complex<double> c(real, imag);
    std::complex<double> z(0, 0);
    int i;
    for (i = 1; i < MAX_ITER; i++) {
        if (std::abs(z) > 2.0) {
            getRGBColor(i, R, G, B);
            return;
        }
        z = z * z + c;
    }
    R = G = B = 0;
}

// Compute the Mandelbrot fractal and fill the pixel buffer.
void computeFractal() {
    for (int x = 0; x < IMAGE_WIDTH; x++) {
```

```
        std::cout << "\r" << (x * 100 / IMAGE_WIDTH) << "% complete" << std::flush;
        for (int y = 0; y < IMAGE_HEIGHT; y++) {
            double real_part = (x - 0.75 * IMAGE_WIDTH) / (IMAGE_WIDTH / 4.0);
            double imag_part = (y - (IMAGE_WIDTH / 4.0)) / (IMAGE_WIDTH / 4.0);
            unsigned char R, G, B;
            mandelbrotColor(real_part, imag_part, R, G, B);
            int index = 3 * (y * IMAGE_WIDTH + x);
            pixels[index] = R;
            pixels[index + 1] = G;
            pixels[index + 2] = B;
        }
    }
    std::cout << "\nFractal computation complete." << std::endl;
}

void drawFractal() {
    glClear(GL_COLOR_BUFFER_BIT);
    glRasterPos2i(-1, -1); // OpenGL coordinates range from -1 to 1
    glDrawPixels(IMAGE_WIDTH, IMAGE_HEIGHT, GL_RGB, GL_UNSIGNED_BYTE,
pixels.data());
}

int main() {
    computeFractal();

    // Initialize GLFW
    if (!glfwInit()) {
        std::cerr << "Failed to initialize GLFW\n";
        return -1;
    }

    // Request OpenGL 2.1 context
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);

    GLFWwindow* window = glfwCreateWindow(IMAGE_WIDTH, IMAGE_HEIGHT, "Mandelbrot
Fractal", nullptr, nullptr);
    if (!window) {
        std::cerr << "Failed to create GLFW window\n";
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent(window);
```

```
    // Initialize GLEW
    glewExperimental = GL_TRUE;
    GLenum glewStatus = glewInit();
    if (glewStatus != GLEW_OK) {
        std::cerr << "GLEW Error: " << glewGetErrorString(glewStatus) << std::endl;
        return -1;
    }

    // Set up OpenGL
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glViewport(0, 0, IMAGE_WIDTH, IMAGE_HEIGHT);

    // Main loop
    while (!glfwWindowShouldClose(window)) {
        drawFractal();
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    glfwDestroyWindow(window);
    glfwTerminate();
    return 0;
}
```

Code in Python

```
from PIL import Image



def mandelbrot(width, height, max_iter):

    # Define the region of the complex plane to visualize

    xmin, xmax = -2.5, 1.5

    ymin, ymax = -2.0, 2.0



    # Create a new grayscale image ("L" mode)

    image = Image.new("L", (width, height))
```

```
    pixels = image.load()

    for x in range(width):
        for y in range(height):
            # Map pixel position to a point in the complex plane
            a = xmin + (x / width) * (xmax - xmin)
            b = ymin + (y / height) * (ymax - ymin)
            c = complex(a, b)
            z = 0
            iter_count = 0

            # Mandelbrot iteration
            while abs(z) <= 2 and iter_count < max_iter:
                z = z * z + c
                iter_count += 1

            # Map the iteration count to a grayscale value (0-255)
            # Points inside the Mandelbrot set will be black if iter_count == max_iter
            color = int(255 * iter_count / max_iter)
            pixels[x, y] = color

    return image

if __name__ == "__main__":
```

```
    # Define image dimensions and maximum iterations
    width = 800
    height = 800
    max_iter = 100


    # Generate the Mandelbrot image and save it
    mandelbrot_image = mandelbrot(width, height, max_iter)
    mandelbrot_image.save("mandelbrot.png")
    mandelbrot_image.show()
```

QUESTION 3

Youtube Link:
https://youtu.be/2480Mt5azME