* **julia\_openmp.c (backend)**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define WIDTH 800

#define HEIGHT 800

#define MAX\_ITER 300

void save\_ppm(const char\* filename, int\* data) {

    FILE\* fp = fopen(filename, "wb");

    fprintf(fp, "P6\n%d %d\n255\n", WIDTH, HEIGHT);

    for (int i = 0; i < WIDTH \* HEIGHT; i++) {

        unsigned char color = (unsigned char)data[i];

        fwrite(&color, 1, 1, fp);  // R

        fwrite(&color, 1, 1, fp);  // G

        fwrite(&color, 1, 1, fp);  // B

    }

    fclose(fp);

}

int main(int argc, char\* argv[]) {

    if (argc < 3) {

        printf("Usage: %s <real> <imaginary>\n", argv[0]);

        return 1;

    }

    double creal = atof(argv[1]);

    double cimag = atof(argv[2]);

    int\* result = (int\*)malloc(WIDTH \* HEIGHT \* sizeof(int));

    #pragma omp parallel for schedule(dynamic)

    for (int y = 0; y < HEIGHT; y++) {

        for (int x = 0; x < WIDTH; x++) {

            double zx = 1.5 \* (x - WIDTH / 2) / (0.5 \* WIDTH);

            double zy = (y - HEIGHT / 2) / (0.5 \* HEIGHT);

            int i = MAX\_ITER;

            while (zx \* zx + zy \* zy < 4.0 && i > 0) {

                double tmp = zx \* zx - zy \* zy + creal;

                zy = 2.0 \* zx \* zy + cimag;

                zx = tmp;

                i--;

            }

            result[y \* WIDTH + x] = (int)(255.0 \* i / MAX\_ITER);

        }

    }

    save\_ppm("julia\_output.ppm", result);

    free(result);

    return 0;

}

* **julia\_openmp.py (GUI)**

import tkinter as tk

from tkinter import messagebox

from PIL import Image

import subprocess

import os

def generate\_julia():

    try:

        real = entry\_real.get()

        imag = entry\_imag.get()

        # Run the compiled C program

        subprocess.run(["julia\_openmp.exe", real, imag], check=True)

        # Convert PPM to PNG

        img = Image.open("julia\_output.ppm")

        img.save("julia\_output.png")

        # Display image in GUI

        img = img.resize((400, 400))

        img\_tk = tk.PhotoImage(file="julia\_output.png")

        image\_label.config(image=img\_tk)

        image\_label.image = img\_tk

        messagebox.showinfo("Success", "Julia set generated and saved as 'julia\_output.png'")

    except Exception as e:

        messagebox.showerror("Error", str(e))

# GUI Setup

root = tk.Tk()

root.title("Julia Set Generator")

root.geometry("450x550")

root.configure(bg="orange")

tk.Label(root, text="Real Part:", bg="orange", fg="black", font=("Arial", 12)).pack(pady=5)

entry\_real = tk.Entry(root, font=("Arial", 12))

entry\_real.pack(pady=5)

entry\_real.insert(0, "-0.7")

tk.Label(root, text="Imaginary Part:", bg="orange", fg="black", font=("Arial", 12)).pack(pady=5)

entry\_imag = tk.Entry(root, font=("Arial", 12))

entry\_imag.pack(pady=5)

entry\_imag.insert(0, "0.27015")

tk.Button(root, text="Generate Julia Set", font=("Arial", 14), command=generate\_julia).pack(pady=20)

image\_label = tk.Label(root)

image\_label.pack(pady=10)

root.mainloop()