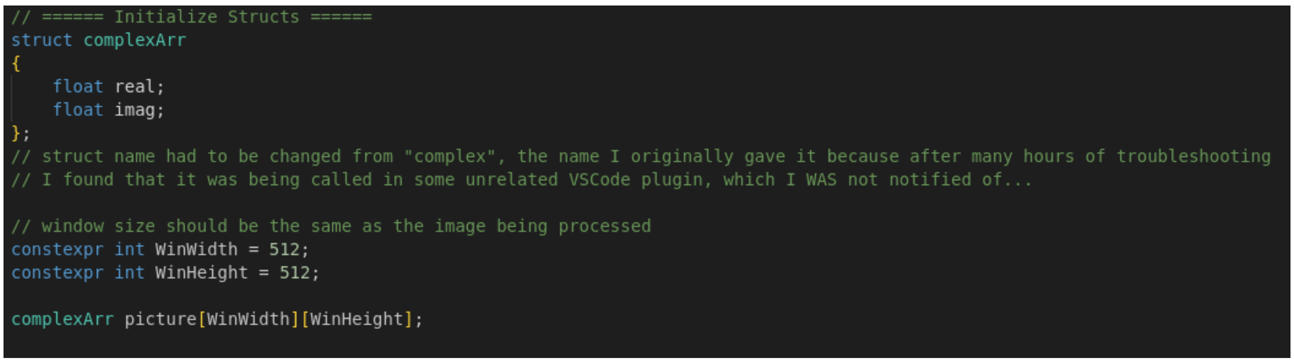
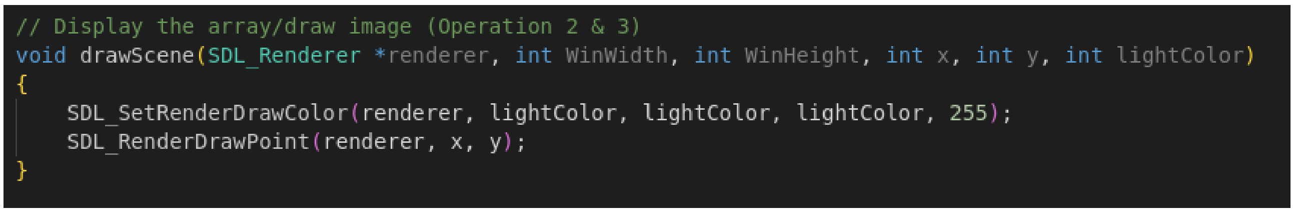
Fourier Transforms Image Manipulation Code Documentation

I made this code in C++



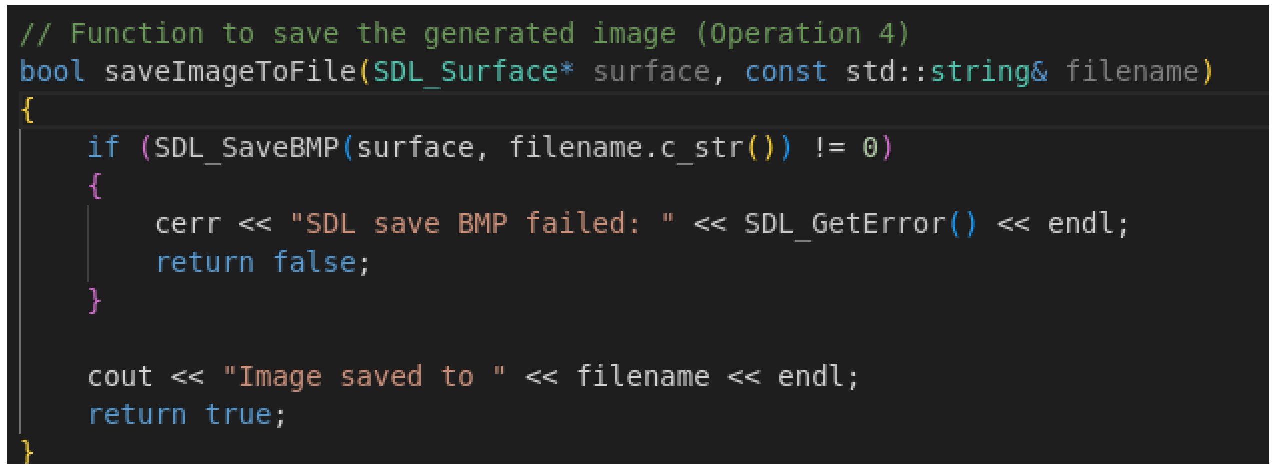
```cpp
// ====== Initialize Structs ======
struct complexArr
{
    float real;
    float imag;
};
// struct name had to be changed from "complex", the name I originally gave it because after many hours of troubleshooting
// I found that it was being called in some unrelated VSCode plugin, which I WAS not notified of...

// window size should be the same as the image being processed
constexpr int WinWidth = 512;
constexpr int WinHeight = 512;

complexArr picture[WinWidth][WinHeight];
```

Before the code is run the “WinWidth” & “WinHeight” should be changed to whatever the image size is.
This code initialises a structure “complexArr” representing complex numbers with real and imaginary parts. It also declares a 2D array “picture” with dimensions 512x512 to store instances of “complexArr”.



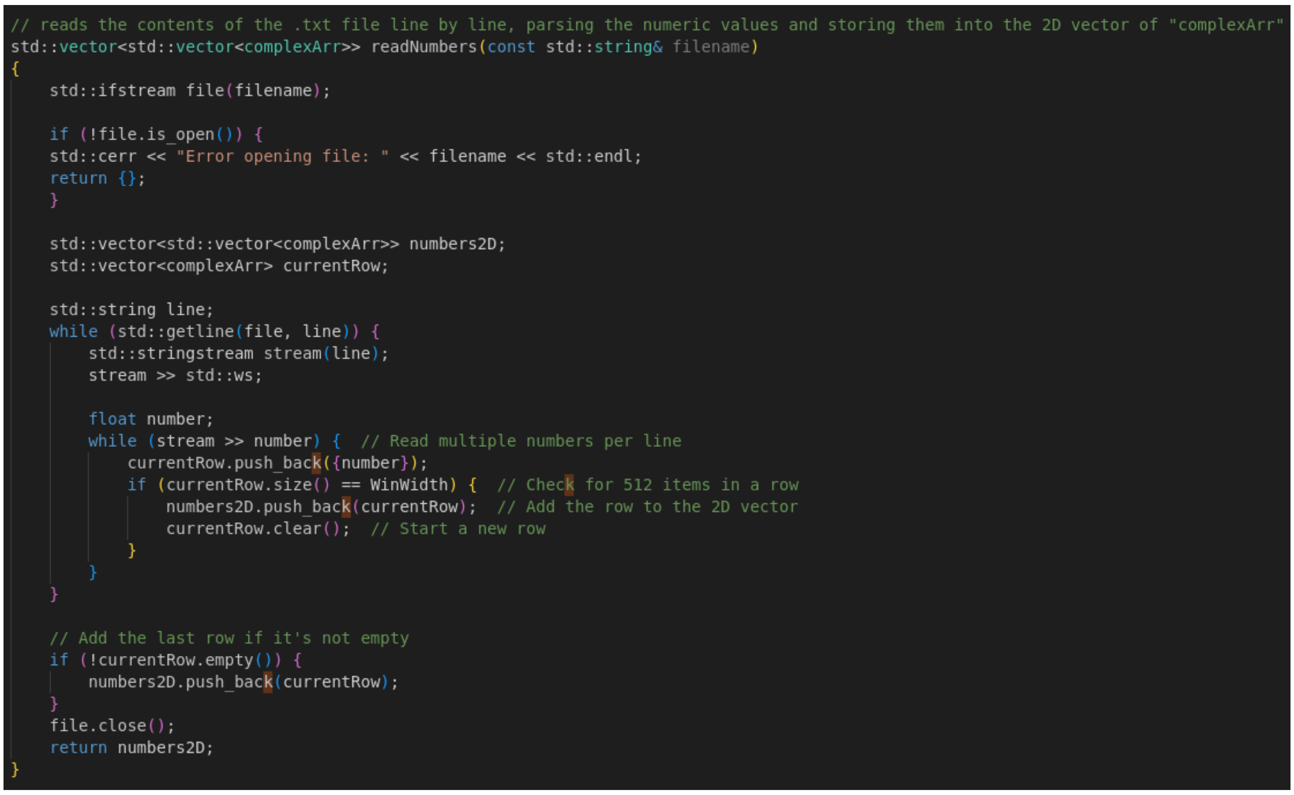
```cpp
// Display the array/draw image (Operation 2 & 3)
void drawScene(SDL_Renderer *renderer, int WinWidth, int WinHeight, int x, int y, int lightColor)
{
    SDL_SetRenderDrawColor(renderer, lightColor, lightColor, lightColor, 255);
    SDL_RenderDrawPoint(renderer, x, y);
}
```

This function draws a point (pixel) at the specified coordinates (x, y). The colour of the point is determined by the render draw colour.



```cpp
// Function to save the generated image (Operation 4)
bool saveImageToFile(SDL_Surface* surface, const std::string& filename)
{
    if (SDL_SaveBMP(surface, filename.c_str()) != 0)
    {
        cerr << "SDL save BMP failed: " << SDL_GetError() << endl;
        return false;
    }

    cout << "Image saved to " << filename << endl;
    return true;
}
```

This function saves the contents of an SDL surface to a BMP image file. It error checks by returning true if the saving process succeeds and false otherwise.



```cpp
// reads the contents of the .txt file line by line, parsing the numeric values and storing them into the 2D vector of "complexArr"
std::vector<std::vector<complexArr>> readNumbers(const std::string& filename)
{
    std::ifstream file(filename);

    if (!file.is_open()) {
    std::cerr << "Error opening file: " << filename << std::endl;
    return {};
    }

    std::vector<std::vector<complexArr>> numbers2D;
    std::vector<complexArr> currentRow;

    std::string line;
    while (std::getline(file, line)) {
        std::stringstream stream(line);
        stream >> std::ws;

        float number;
        while (stream >> number) {  // Read multiple numbers per line
            currentRow.push_back({number});
            if (currentRow.size() == WinWidth) {  // Check for 512 items in a row
                numbers2D.push_back(currentRow);  // Add the row to the 2D vector
                currentRow.clear();  // Start a new row
            }
        }
    }

    // Add the last row if it's not empty
    if (!currentRow.empty()) {
        numbers2D.push_back(currentRow);
    }
    file.close();
    return numbers2D;
}
```

This code reads numeric values from a text file and organises them into a 2D vector structure.

The functions order of operation:

Open the file:
It attempts to open the specified file using std::ifstream.
If the file fails to open, it prints an error message and returns an empty vector.

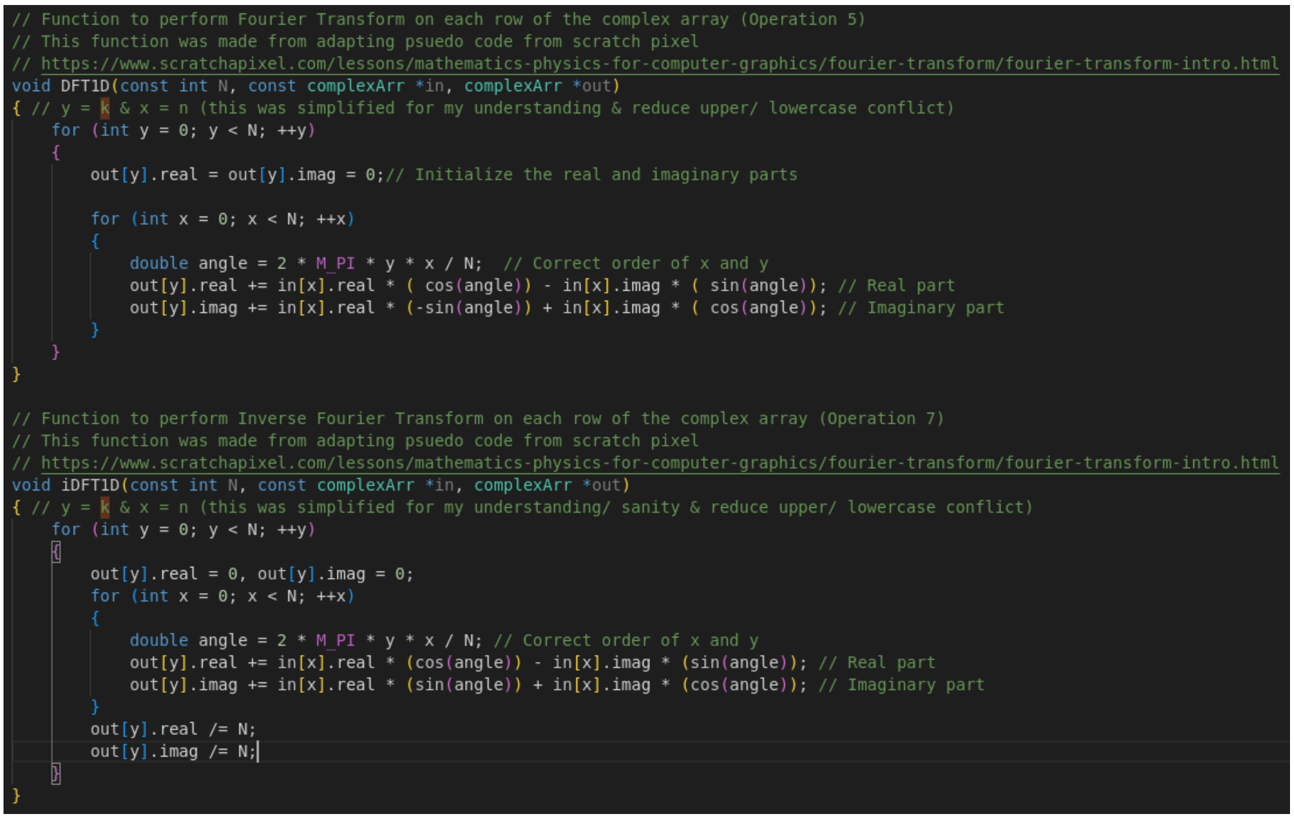
Initialise variables:
It creates two empty vectors:
numbers2D: holds the 2D structure of the parsed numbers.
currentRow: This temporarily stores numbers from each line before adding them to “numbers2D”.

Read lines from the file:
It enters a loop using std::getline.

Parse numbers from each line:
For each line, it creates a stringstream to extract individual numbers.
It reads numbers from the stream using stream >> number and stores them in currentRow.
When currentRow reaches a specific capacity (based on “WinWidth” default: 512):
It clears currentRow to start a new row.

After processing all lines, it checks if currentRow is not empty and adds it to “numbers2D” to ensure the last row is included.

Close the file and return “numbers2D”:

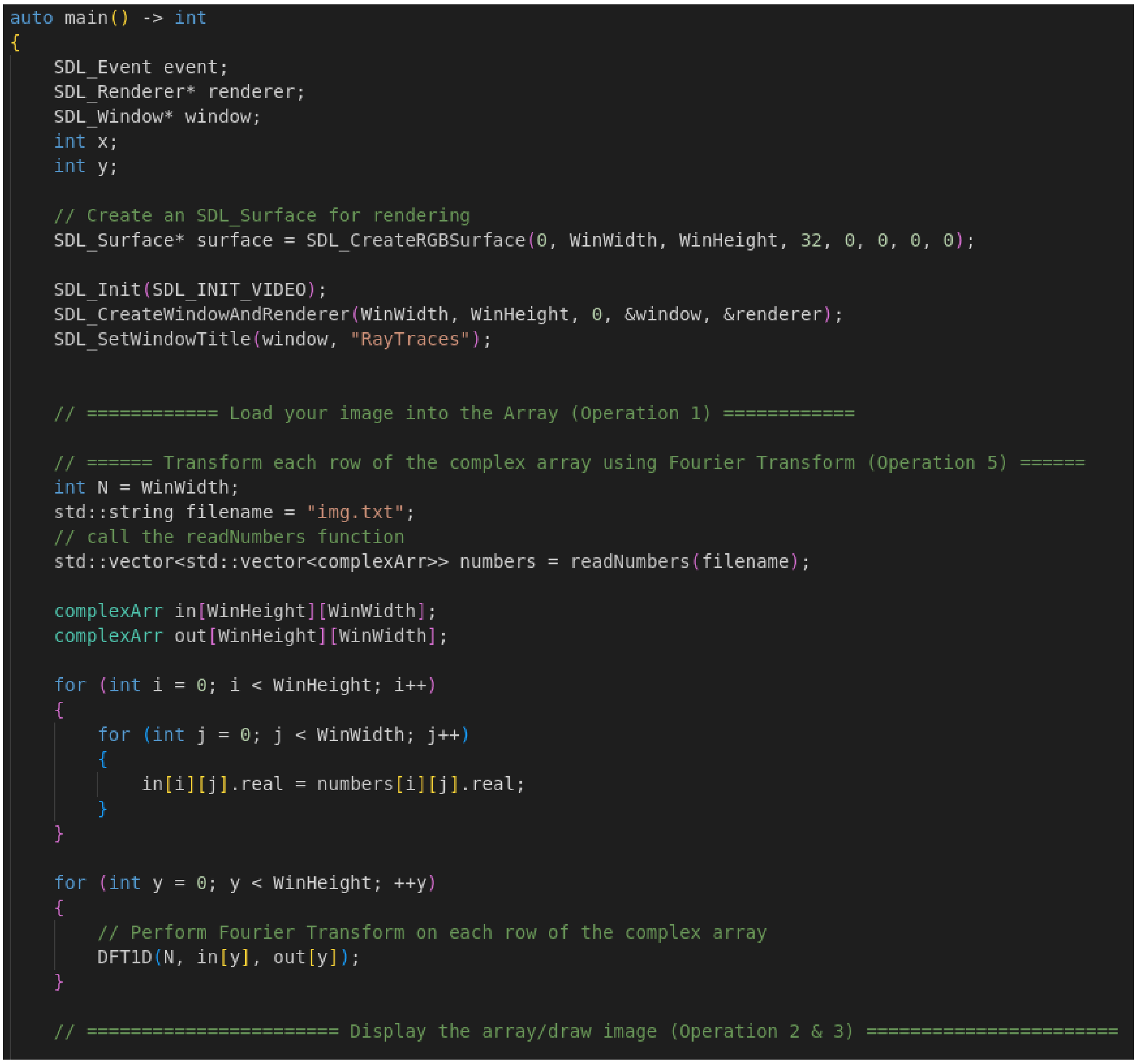


```
// Function to perform Fourier Transform on each row of the complex array (Operation 5)
// This function was made from adapting psuedo code from scratch pixel
// https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/fourier-transform/fourier-transform-intro.html
void DFT1D(const int N, const complexArr *in, complexArr *out)
{ // y = k & x = n (this was simplified for my understanding & reduce upper/ lowercase conflict)
    for (int y = 0; y < N; ++y)
    {
        out[y].real = out[y].imag = 0;// Initialize the real and imaginary parts

        for (int x = 0; x < N; ++x)
        {
            double angle = 2 * M_PI * y * x / N;  // Correct order of x and y
            out[y].real += in[x].real * ( cos(angle)) - in[x].imag * ( sin(angle)); // Real part
            out[y].imag += in[x].real * (-sin(angle)) + in[x].imag * ( cos(angle)); // Imaginary part
        }
    }
}

// Function to perform Inverse Fourier Transform on each row of the complex array (Operation 7)
// This function was made from adapting psuedo code from scratch pixel
// https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/fourier-transform/fourier-transform-intro.html
void iDFT1D(const int N, const complexArr *in, complexArr *out)
{ // y = k & x = n (this was simplified for my understanding/ sanity & reduce upper/ lowercase conflict)
    for (int y = 0; y < N; ++y)
    {
        out[y].real = 0, out[y].imag = 0;
        for (int x = 0; x < N; ++x)
        {
            double angle = 2 * M_PI * y * x / N; // Correct order of x and y
            out[y].real += in[x].real * (cos(angle)) - in[x].imag * (sin(angle)); // Real part
            out[y].imag += in[x].real * (sin(angle)) + in[x].imag * (cos(angle)); // Imaginary part
        }
        out[y].real /= N;
        out[y].imag /= N;
    }
}
```

These are the functions for the Discrete Fourier Transform (DFT) & its inverse (IDFT).

This function was made from adapting psuedo code from scratch pixel
https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/fourier-transform/fourier-transform-intro.html

I substituted k & n for x & y as it made it more readable for myself, in the scratch pixel pseudo code version this “2 * M_PI * y * x / N” was used repeatedly in the function so I shortened it to a variable “angle”



```
auto main() -> int
{
    SDL_Event event;
    SDL_Renderer* renderer;
    SDL_Window* window;
    int x;
    int y;

    // Create an SDL_Surface for rendering
    SDL_Surface* surface = SDL_CreateRGBSurface(0, WinWidth, WinHeight, 32, 0, 0, 0, 0);

    SDL_Init(SDL_INIT_VIDEO);
    SDL_CreateWindowAndRenderer(WinWidth, WinHeight, 0, &window, &renderer);
    SDL_SetWindowTitle(window, "RayTraces");


    // ============ Load your image into the Array (Operation 1) ============

    // ====== Transform each row of the complex array using Fourier Transform (Operation 5) ======
    int N = WinWidth;
    std::string filename = "img.txt";
    // call the readNumbers function
    std::vector<std::vector<complexArr>> numbers = readNumbers(filename);

    complexArr in[WinHeight][WinWidth];
    complexArr out[WinHeight][WinWidth];

    for (int i = 0; i < WinHeight; i++)
    {
        for (int j = 0; j < WinWidth; j++)
        {
            in[i][j].real = numbers[i][j].real;
        }
    }

    for (int y = 0; y < WinHeight; ++y)
    {
        // Perform Fourier Transform on each row of the complex array
        DFT1D(N, in[y], out[y]);
    }

    // ====================== Display the array/draw image (Operation 2 & 3) ======================
```

The main function creates the window, the scene and calls the other functions:

1) Call image to complex array function/ code
2) Call the Fourier Transform functions/ code
3) Render the results using for loops (x,y) calling the drawscene function
4) Display to user & save using SDL_SaveBMP
5) Repeat 2-4 until code is done
6) Clean up and quit

(The current version doesn't save the inverse transform that I attempted)