AMRITA VISHWA VIDYAPEETHAM

Department of Artificial Intelligence

22AIE101: Problem Solving & C Programming, Fall 2023

Project Report

IMAGE PROCESSING:
HISTOGRAM GENERATION OF AN IMAGE

Team Members:

C Vishnuvardhan Chowdary: CB.SC.U4AIE23219
Abhishek Pandey: CB.SC.U4AIE23205
G Harshavardhan: CB.SC.U4AIE226
Aditya Santosh: CB.SC.U4AIE23207
Adith Krishna: CB.SC.U4AIE23202

Supervised By:

Dr. Vidhya Kamakshi V
Assistant Professor
Department of Artificial Intelligence
Amrita Vishwa Vidyapeetham

Date of Submission: 30/12/2023

Signature of the Project Supervisor:

Amrita Vishwa Vidyapeetham

Coimbatore

CERTIFICATE

This is to certify that we, the student of Amrita School of Artificial Intelligence, has completed the "***HISTOGRAM GENERATION OF AN IMAGE"*** as part of their Image processing in C Programming.

The project work was carried out under the guidance and supervision of Dr. Vidhya Kamakshi V, Assistant Professor, Amrita School of Artificial Intelligence, Coimbatore. To the best of our knowledge this work has not formed the basis for the aware of any degree/diploma/ associate ship/fellowship/ or a similar award to any candidate in any University.

Date:

Dr. Vidhya Kamakshi V

School of Artificial Intelligence

Amrita Vishwa Vidyapeetham

Coimbatore

Acknowledgement

We would like to make this an opportunity to transfuse our appreciation to everyone who was behind the successful completion of this design. First and foremost, we would like to thank "Department of Artificial Intelligence" for accepting our project.

Kindest regard goes to **Dr. Soman KP**, Head of the Department of Artificial Intelligence for allowing us to accomplish the design project.

We would like to convey appreciation to our supervisor**. Dr. Vidhya Kamakshi V**, for his encouragement and motivation to expand our vision regarding proper process designing. We are very thankful for his help and supervision. This task would have been little success without his proper guidance and support.

Last but not the least, we are thankful to our group members and friends of our department for their friendly support given to accomplish this project successfully and our family members for always providing the best guidance possible.

A big thanks to you all.

Abstract

In many applications, including computer vision, medical imaging, and multimedia systems, digital image processing is essential. The creation of histograms, which offer a statistical depiction of the distribution of pixel intensities within an image, is a crucial procedure in image processing. This abstract describes how to use the C programming language to implement the creation of a histogram for a picture. The suggested workaround entails writing a C program that can accept image data, compute the histogram of pixel intensities, and output the results in an understandable manner. For basic pixel manipulation and picture input/output operations, the application makes use of either bespoke functions or conventional image processing libraries.

The application has the ability to read a BMP image file and extract the values of pixel intensity. It should also be able to save the altered image or the created histogram. The algorithm used to determine the pixel intensity histogram is a crucial component. This entails going over each pixel in the picture, calculating its intensity, and then updating the histogram bin that corresponds to it. To make user interaction easier, a straightforward and intuitive user interface is used. Users can input an image file, start the histogram's creation, and examine or save the outcomes via the interface. Efficient data structures are used for storing and updating the histogram data in order to maximize program performance. The resulting program serves as a valuable tool for visualizing and analysing the distribution of pixel intensities in an image. This information is essential for various image processing tasks, such as contrast enhancement, image segmentation, and equalization.

1. Introduction

1.1 General

Images in modern computers are digital representations of visual information. Digital images are made up of discrete pieces called pixels, with each pixel representing a tiny square or dot in the image. The visual content of the image is created by the attributes of individual pixels, such as colour and intensity. Images are frequently portrayed as a grid of pixels. Each pixel has a unique location and a unique colour value. An image's resolution is defined as the number of pixels per unit of length.

In digital imaging, the RGB colour model is frequently used. Each pixel is represented by a colour intensity combination of red, green, and blue. Images can also be grayscale, with each pixel representing a different shade of Gray ranging from black to white. Digital photos are stored in several file formats. JPEG is commonly used for photographs, whereas PNG is used for compression without loss of data and GIF is used for simple graphics and animations.

Digital images are used in a variety of applications. Advanced computer vision algorithms examine digital images for object recognition, face detection, and scene comprehension. Machine learning models are trained on massive datasets of images for tasks such as image classification and separation.

A histogram is a graphical depiction of the intensity distribution of pixels in an image. A histogram is useful in image processing because it shows the distribution of colors or intensities throughout an image. This data is required for a variety of image analysis and enhancement tasks.

1.2Scope of our project

1.2.1 Image Processing and Analysis:

- Feature Extraction: Histograms are fundamental for analysing and extracting features from images. They provide insights into the distribution of pixel intensities, aiding in tasks such as edge detection, texture analysis, and object recognition.
- Image Enhancement: Histogram equalization and stretching techniques, based on the histogram analysis, can be employed to enhance the contrast and overall quality of images.

1.2.2 Medical Imaging:

- Diagnostic Imaging: Histogram analysis is valuable in medical imaging.

1.2.3 Digital Photography:

- Image Editing: Histograms are often used in photo editing software to adjust brightness, contrast, and colour balance, providing users with intuitive controls for image manipulation.

1.2.4 Education and Research:

- Learning Image Processing: The implementation of histogram generation in C can serve as an educational tool for students learning image processing algorithms and techniques.
- Research and Development: Researchers can use histogram analysis as a foundational step for developing more advanced image processing algorithms.

2. Overview

This project focuses on creating a histogram from an image provided as input. The purpose is to receive a histogram of the given image so that we may analyse it and make the required changes to improve the image. We employed the following approaches in this project: RGB Image scaling, identifying pixel intensity, and histogram generation.

Understanding RGB images and pixel intensity are fundamental topics. RGB images comprise of three channels (Red, Green, Blue), with pixel intensities ranging from No colour to most intense colour. Pixels are the smallest component of an image, and their intensity represents the image's colour and brightness.

Histograms are important in image analysis because they provide a visual depiction of an image's pixel intensity distribution.

Histograms represent the frequency of occurrence of each pixel or their intensity values in an image, which can provide information about the image's colour, brightness, and general quality. By utilizing and altering these factors through image processing techniques, we may improve image visibility, highlight specific features of the image, and overall image quality.

Histogram, grayscale, and pixel intensity determination form the groundwork for image processing and are critical strategies for making image processing possible.

3. Literature Review

3.1 1970s-1980s:

Early Digital Imaging and Basic Histograms.

3.1.1 Context:

As digital imaging emerged, early developments in image processing started to take place.

3.1.2 Development:

Basic histogram calculations were implemented in early programming languages, including C, to analyse pixel intensity distributions in digital images.

3.2 1980s-1990s:

Integration into Image Processing Libraries

3.2.1 Context:

Image processing libraries and frameworks started to incorporate histogram functionalities.

3.2.2 Development:

C programming played a role in the integration of histogram-related functions into popular image processing libraries. This period saw the emergence of C libraries that provided tools for basic histogram analysis.

3.3 1990s-2000s:

Optimization and Efficiency Improvements

3.3.1 Context:

With the increasing size and complexity of digital images, there was a need for more efficient histogram computation.

3.3.2 Development:

C programming techniques for optimizing histogram calculations were developed. This involved algorithmic improvements and low-level optimizations to enhance the speed of histogram processing.

3.4 2000s-2010s:

Parallelization and Multithreading

3.4.1 Context:

As multicore processors became common, efforts were made to parallelize image processing tasks, including histogram computation.

3.4.2 Development:

C programmers explored techniques for parallelizing histogram calculations using multithreading and SIMD (Single Instruction, Multiple Data) instructions. This led to improved performance on modern hardware.

3.5 2010s-Present:

Integration with GPU Programming

3.5.1 Context:

The rise of GPU computing opened up new possibilities for accelerating image processing tasks.

3.5.2 Development:

C programming was extended to utilize GPUs for histogram computation. This involved the use of CUDA (Compute Unified Device Architecture) or OpenCL (Open Computing Language) to harness the parallel processing power of graphics cards for faster histogram analysis.

3.6 Recent Years:

Machine Learning Integration

3.6.1 Context:

With the growth of machine learning, there has been an increased interest in using image histograms as features for training models.

3.6.2 Development:

C programming has been involved in developing interfaces between traditional image processing tasks, including histogram analysis, and machine learning frameworks. This integration allows for the incorporation of histogram-based features in machine learning algorithms.

3.7 Future Directions:

Real-time and Edge Computing

3.7.1 Context:

The demand for real-time image processing, particularly in applications like autonomous vehicles and edge devices, has driven ongoing developments.

3.7.2 Development:

C programming is likely to play a role in optimizing histogram computations for real-time applications. Techniques such as edge computing may involve more efficient ways of processing histograms directly on resource-constrained devices.

4. Methodology

4.1 Image Representation and About Histogram

4.1.1) Pixels: In digital images they are compressed to tiny pictures elements called pixels.

4.1.2) Grid Structures: This pixel forms a grid each pixel contains colour information.

4.1.3) Resolution: The number of pixels in an image talks about its resolution.

4.1.4) About colour channels: Each pixel has its colour information. It often represents as RGB values. (Red, Green, Blue)

4.1.5) Bit Depth: Its bit depth will talk about the number of colours the pixel can represent.

4.1.6) File Formats: Here we are taking "BMP" Image to encode data and compression.

4.1.7) Meta Data: Images may include metadata, like data, and geolocation.

4.1.8) Digital Processing: In digital processing the software can modify images through produces like resizing, cropping and filtering.

4.1.9) Data Ranges: First divide the data into intervals next divide bins, based on range of values.

4.1.10) Count Observation: Determine the number of data points that falls on each bin.

4.1.11) Bin width: We have to select a suitable bin width and balance the need for detail with desire for a clear representation.

4.2 Image Histogram Generation Methods or Ways:

4.3.1) Data Distribution: We have to Understand the nature of dataset to choose a suitable histogram generation technique based on its features.

4.3.2) Bin Width: We have to Decide whether the set is fixed bin width is considering the detail needed for a precise representation.

4.3.3) Cumulative Frequency Histogram: First, we have to Construct a cumulative histogram after that, we have to show the cumulative frequency of observations up to a certain point, aiding in understanding the overall distribution

4.3.4) Overlay Multiple Histograms: Then Compare datasets by overlaying multiple histograms on the same axis, using different colours patterns for clarity in difference between distributions.

4.3.5) Data Transformation: we have to Consider applied data transformation, such as logarithmic scaling to skewed distributions and enhance the visibility of patterns in the histogram.

4.3 Header Files

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#pragma pack(2)
```

- #include <stdio.h>:

  This line includes the standard input/output library (‘stdio.h’). It provides functions for input and output operations, such as ‘printf’ and ‘scanf’.

- #include <stdlib.h> :

  This line includes the standard library (‘stdlib.h’). It contains functions involving memory allocation (‘malloc’ , ‘free’) and other general utility functions.

- #include <math.h> :

  This line includes the math library (‘math.h’). It provides mathematical functions, such as ‘sqrt’ and ‘sin’.

- #pragma pack(2)

  This line is a compiler directive that specifies the alignment of structure members in memory. In this case, it sets alignment to 2 bytes. This is often used when dealing with file formats like BMP, where the file structure’s alignment is crucial.

4.4 BMP Header and DIB Header

```c
typedef struct {
    unsigned short signature;
    unsigned int fileSize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int dataOffset;
} BMPHeader;

typedef struct {
    unsigned int headerSize;
    int width;
    int height;
    unsigned short planes;
    unsigned short bitDepth;
    unsigned int compression;
    unsigned int imageSize;
    int xPixelsPerMeter;
    int yPixelsPerMeter;
    unsigned int colorsUsed;
    unsigned int importantColors;
} DIBHeader;
```

- BMP Header Structure

  This is a comment indicating that the following code defines a structure representing the BMP (Bitmap) header.
- unsigned short signature

  This field represents the signature of the BMP file, which is typically "BM" (0x424D in hexadecimal). It is a 16-bit unsigned integer.
- unsigned int fileSize

  This field represents the total size of the BMP file in bytes. It is a 32-bit unsigned integer (unsigned int).
- unsigned short reserved1

  This field is reserved for future use and should be set to 0. It is a 16-bit unsigned integer.
- unsigned short reserved2

  Another reserved field, set to 0. It is a 16-bit unsigned integer.
- unsigned int dataOffset

  This field represents the offset from the beginning of the file to the start of the image data. It is a 32-bit unsigned integer.
- DIB Header Structure:

  This is a comment indicating that the following code defines a structure representing the DIB (Device-Independent Bitmap) header.
- unsigned int headerSize

  This field represents the size of the DIB header in bytes. It is a 32-bit unsigned integer.
- int width;

  This field represents the width of the image in pixels. It is a 32-bit signed integer (int).
- int height;

  This field represents the height of the image in pixels. It is a 32-bit signed integer (int).
- unsigned short planes;

  This field represents the number of colour planes in the image. For BMP files, it must be 1. It is a 16-bit unsigned integer (unsigned short).
- unsigned short bit Depth;

  This field represents the number of bits used to represent each pixel. For RGB images, it is typically 24 bits. It is a 16-bit unsigned integer (unsigned short).

- unsigned int compression;

  This field indicates the compression method used for the image data. For uncompressed BMP files, this should be 0. It is a 32-bit unsigned integer (unsigned int).
- unsigned int imageSize;

  This field represents the size of the image data in bytes. It is a 32-bit unsigned integer (unsigned int).
- int xPixelsPerMeter;

  This field represents the horizontal resolution of the image in pixels per meter. It is a 32-bit signed integer (int).
- int yPixelsPerMeter;

  This field represents the vertical resolution of the image in pixels per meter. It is a 32-bit signed integer (int).
- unsigned int colorsUsed;

  This field indicates the number of colors used in the colour palette. For full-colour images, this should be set to 0. It is a 32-bit unsigned integer (unsigned int).
- unsigned int importantColors;

  This field indicates the number of important colors. For most BMP files, all colors are considered important, so this is typically set to 0. It is a 32-bit unsigned integer (unsigned int).

4.5 Read BMP Image

```
void readBMP(const char* filename, BMPHeader* bmpHeader, DIBHeader* dibHeader, unsigned char** imageData) {
    FILE* file = fopen(filename, "rb");
    if (!file) {
        perror("Error opening file");
        return;
    }

    fread(bmpHeader, sizeof(BMPHeader), 1, file);

    if (bmpHeader->signature != 0x4D42) {
        fprintf(stderr, "Not a valid BMP file\n");
        fclose(file);
        return;
    }

    fread(dibHeader, sizeof(DIBHeader), 1, file);

    if (dibHeader->bitDepth != 24 || dibHeader->compression != 0) {
        fprintf(stderr, "Unsupported BMP format\n");
        fclose(file);
        return;
    }

    int padding = (4 - (dibHeader->width * 3) % 4) % 4;

    *imageData = (unsigned char*)malloc(dibHeader->width * dibHeader->height * 3);

    fseek(file, bmpHeader->dataOffset, SEEK_SET);
    fread(*imageData, 1, dibHeader->width * dibHeader->height * 3, file);

    fclose(file);
}
```

- void readBMP(const char* filename, BMPHeader* bmpHeader, DIBHeader* dibHeader, unsigned char** imageData) {

  This line declares a function named readBMP that takes four parameters: filename (the path to the BMP file), bmpHeader (a pointer to the BMPHeader structure), dibHeader (a pointer to the DIBHeader structure), and imageData (a pointer to a pointer to unsigned characters, which will store the pixel data).

- FILE* file = fopen(filename, "rb");

  This line opens the BMP file specified by the filename in binary read mode ("rb"), and it assigns the file pointer to the variable file.

- if (!file) {

  perror("Error opening file");

  return;

  }

  This checks if the file opening operation was successful. If not, it prints an error message using perror and exits the function.

- if (bmpHeader->signature != 0x4D42) {

```
fprintf(stderr, "Not a valid BMP file\n");
fclose(file);
return;
}
```

This checks if the BMP signature is equal to the hexadecimal value 0x4D42 (ASCII characters 'B' and 'M'). If not, it prints an error message, closes the file, and exits the function.

- fread(dibHeader, sizeof(DIBHeader), 1, file);
  This line reads the DIB header from the file into the memory location pointed to by dibHeader. It reads sizeof(DIBHeader) bytes and repeats this operation once (1 item).
- if (dibHeader->bitDepth != 24 || dibHeader->compression != 0) {
  fprintf(stderr, "Unsupported BMP format\n");
  fclose(file);
  return;
  }
  This checks if the image is in the 24-bit true colour format (bits per pixel equal to 24) and if it is uncompressed (compression equals 0). If not, it prints an error message, closes the file, and exits the function.
- int padding = (4 - (dibHeader->width * 3) % 4) % 4;
  This line calculates the number of padding bytes needed for each row of the image data. The padding ensures that each row's size is a multiple of 4 bytes.
- imageData = (unsigned char*)malloc(dibHeader->width * dibHeader->height * 3)
  This line attempts to allocate memory to store the pixel data. fseek(file, bmpHeader->dataOffset, SEEK_SET);
  This line moves the file pointer to the beginning of the image data based on the offset stored in bmpHeader->dataOffset.
- fread(*imageData, 1, dibHeader->width * dibHeader->height * 3, file);
  This reads the pixel data from the file into the memory location pointed to by imageData. It reads `dibHeader->width * dibHeader

- fclose(file);
  This line closes the file stream using fclose. Closing the file is essential to release system resources and ensure that the file is properly handled.

4.6 Histogram Calculation

```
63  void histogram(unsigned char* imageData, int width, int height, int* redFrequency, int* greenFrequency, int* blueFrequency) {
64      for (int i = 0; i < 256; i++) {
65
66          redFrequency[i] = 0;
67          greenFrequency[i] = 0;
68          blueFrequency[i] = 0;
69      }
70
71      for (int i = 0; i < height; i++) {
72          for (int j = 0; j < width; j++) {
73
74              blueFrequency[imageData[(i * width + j) * 3]]++;
75              greenFrequency[imageData[(i * width + j) * 3 + 1]]++;
76              redFrequency[imageData[(i * width + j) * 3 + 2]]++;
77          }
78      }
```

- for (int i = 0; i < 256; i++) {

redFrequency[i] = 0;

greenFrequency[i] = 0;

blueFrequency[i] = 0;

}

This loop initializes three arrays (redFrequency, greenFrequency, and blueFrequency) with zeros. These arrays are used to store the frequency of each pixel value for the red, green, and blue colour channels, respectively. The loop runs 256 times, corresponding to the possible pixel values (0 to 255).

- for (int i = 0; i < height; i++) {
  for (int j = 0; j < width; j++) {
  This nested loop iterates through each pixel in the image. i represents the row (height), and j represents the column (width).
- **blueFrequency[imageData[(i * width + j) * 3]]++;:**
  This line increments the count of the blue colour channel for the pixel at position (i, j) in the image. The pixel data is stored in BGR (Blue, Green, Red) order, and imageData is assumed to be a 1D array containing the pixel values.
- greenFrequency[imageData[(i * width + j) * 3 + 1]] ++;

Similarly, this line increments the count of the green color channel for the pixel at position (i, j) in the image.

- redFrequency[imageData[(i * width + j) * 3 + 2]]++;

  This line increments the count of the red color channel for the pixel at position (i, j) in the image.

4.7 Display Histogram

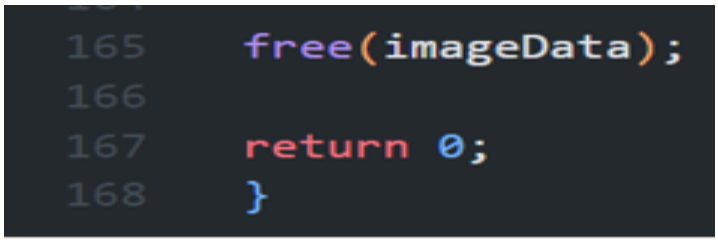
```
92  void displayHistogram(int* redFrequency, int* greenFrequency, int* blueFrequency, int size) {
93      int max = 0;
94      for (int i = 0; i < size; i++) {
95          if (redFrequency[i] > max) {
96              max = redFrequency[i];
97          }
98          if (greenFrequency[i] > max) {
99              max = greenFrequency[i];
100         }
101         if (blueFrequency[i] > max) {
102             max = blueFrequency[i];
103         }
104     }
105
106     double scale = 80.0 / max;
107
108     printf("Red Histogram\n");
109     for (int i = 0; i < size; i++) {
110         printf("%3d: ", i);
111
112         printf("\033[31m");
113         int barLength = (int)(redFrequency[i] * scale);
114         for (int j = 0; j < barLength; j++) {
115             printf("*");
116         }
117         printf("\033[0m");
118         printf("\n");
119     }
120
```

```
    printf("Green Histogram\n");
    for (int i = 0; i < size; i++) {
        printf("%3d: ", i);

        printf("\033[32m");
        int barLength = (int)(greenFrequency[i] * scale);
        for (int j = 0; j < barLength; j++) {
            printf("*");
        }
        printf("\033[0m");
        printf("\n");
    }

    printf("Blue Histogram\n");
    for (int i = 0; i < size; i++) {
        printf("%3d: ", i);

        printf("\033[34m");
        int barLength = (int)(blueFrequency[i] * scale);
        for (int j = 0; j < barLength; j++) {
            printf("*");
        }
        printf("\033[0m");
        printf("\n");
    }
}
```

- loop iterates through each element of the frequency arrays (redFrequency, greenFrequency, blueFrequency) to find the maximum frequency among them. The maximum frequency is used later for scaling the bar lengths.The scale variable is calculated to determine how much each unit of frequency corresponds to in terms of the console's column width. The goal is to scale the histograms to fit within 80 columns.For each color channel, the code prints the pixel value, followed by a bar chart representing the frequency. The bars are printed as asterisks ('*') and are color-coded using ANSI escape codes.

The color-coded bars are printed using ANSI escape codes. For instance, \033[31m sets the text color to red, and \033[0m resets the color to the default.The same process is repeated for the green and blue channels, changing the color codes accordingly.The histograms are displayed in the console with color-coded bars. Red bars are printed in red, green bars in green, and blue bars in blue.

4.8 Memory Cleanup



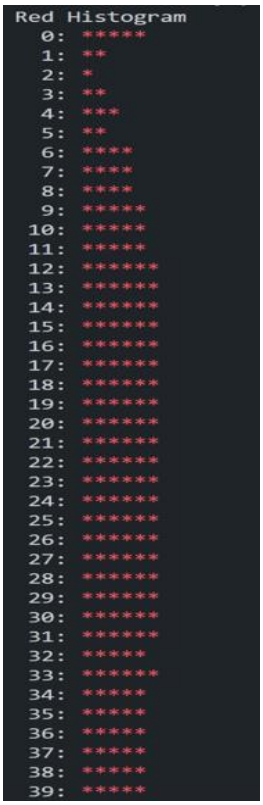
- free(imageData);
  This line is responsible for releasing the memory allocated for the imageData array. The imageData array was dynamically allocated in the readBMP function to store the pixel data of the BMP image.
  The free function is a part of the C standard library, and it is used to deallocate the memory previously allocated by functions like malloc, calloc, or realloc. This step is crucial to prevent memory leaks, where allocated memory is not properly released, leading to inefficient use of system resources.
  After this line executes, the memory occupied by the imageData array is returned to the system for reuse.
- return 0;
  This line signals the end of the main function and the termination of the program. In C, a return value of 0 typically indicates successful execution.
  In this case, return 0 suggests that the program has executed successfully, and the memory cleanup process has been completed.

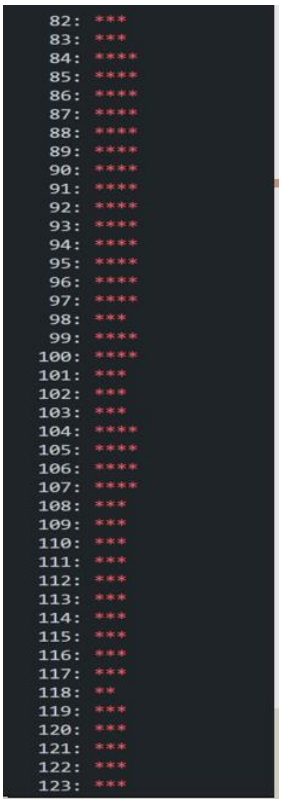
5. Result and Observation

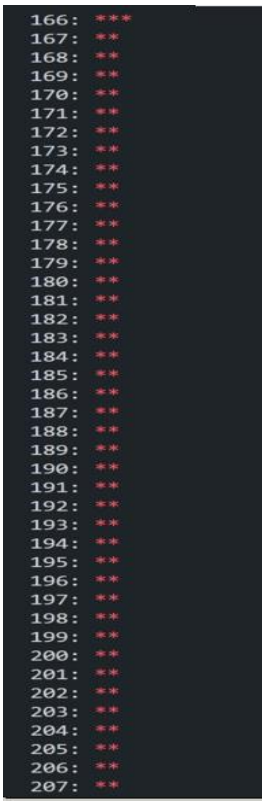
5.1 Input BMP image

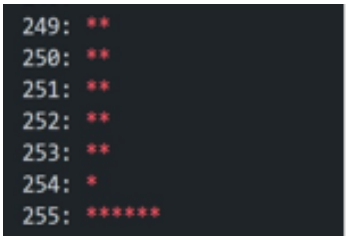
Here, the Image is in the format of BMP which is given by the user. This RGB Image has 3 colour channels. For each colour channel it will calculate the frequency of each colour pixel (i.e., Red Frequency, Green Frequency and Blue Frequency) and it will store in each colour arrays. Then it will display the histogram of each pixel value lie between 0 to 255 for each colour channel. The output of the image is shown below: -

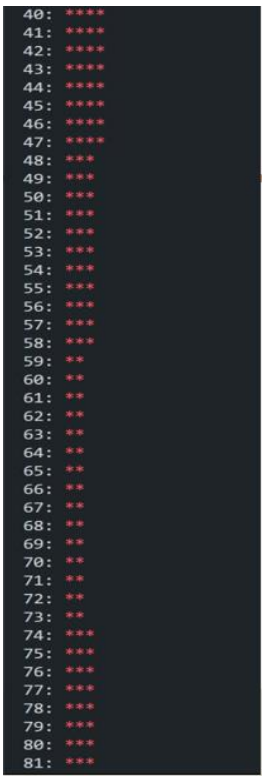
5.2 Output for Red Channels

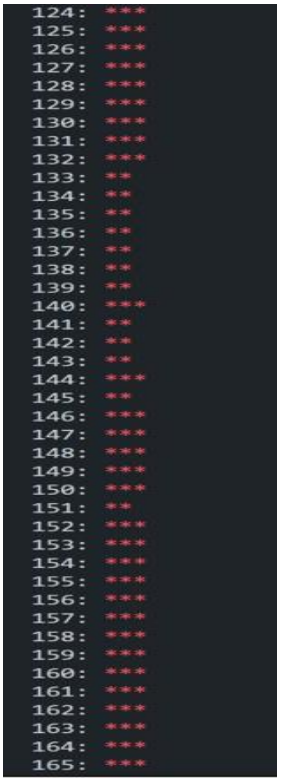


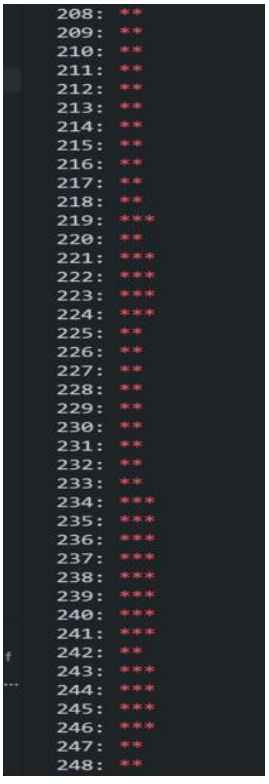




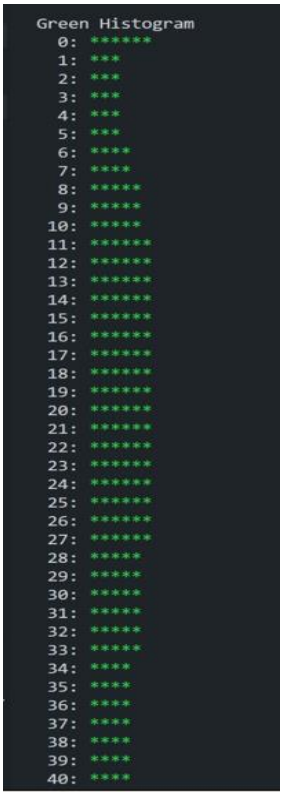


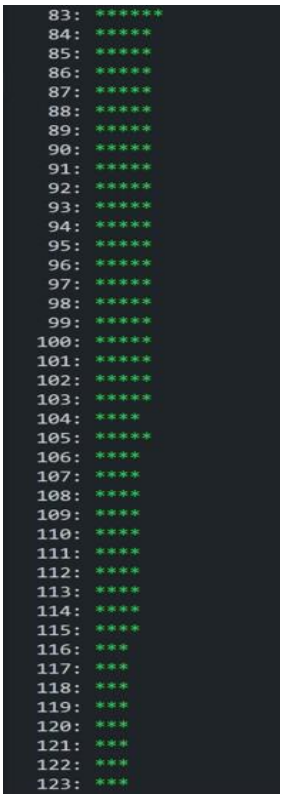


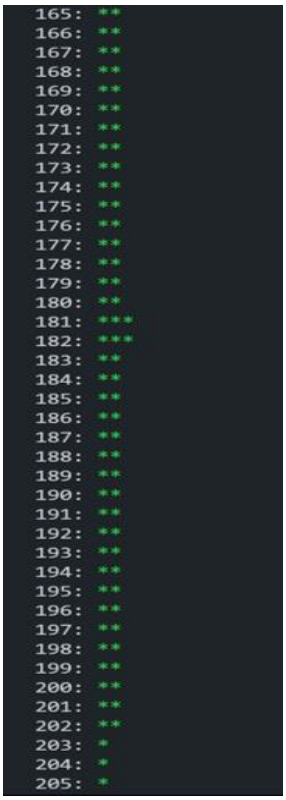


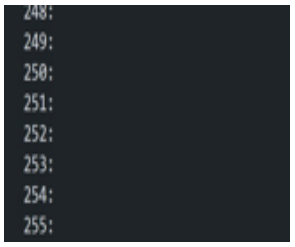


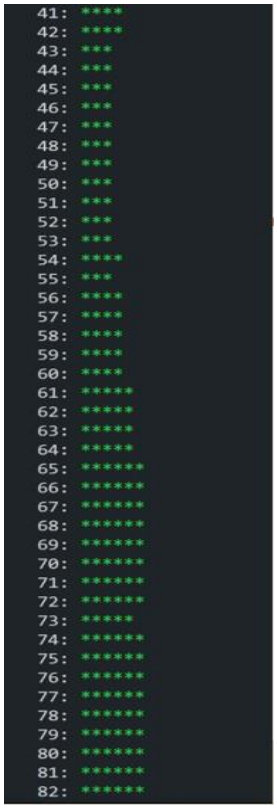
5.3 Output Green Channel

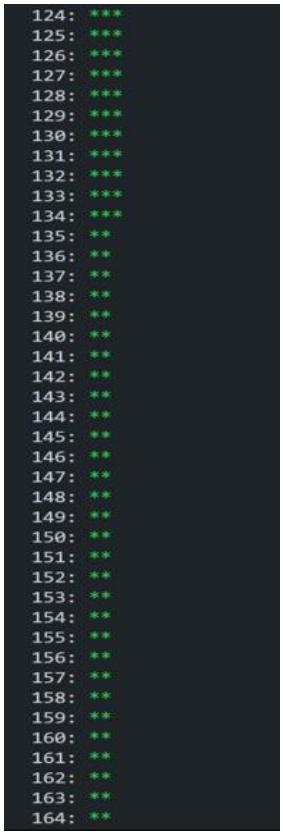


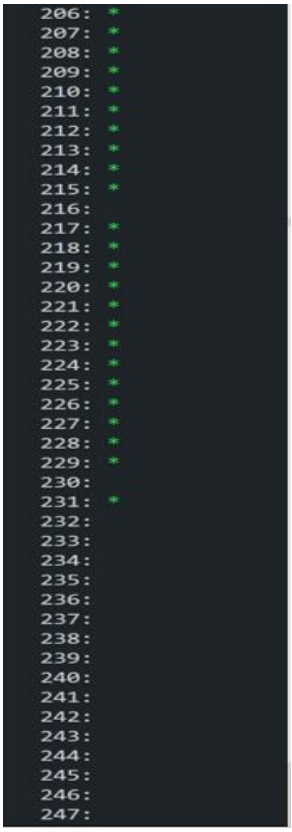




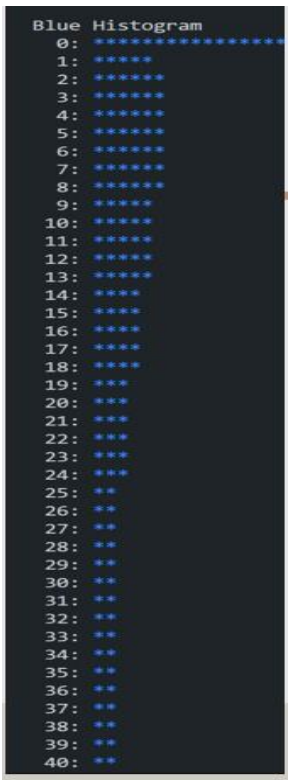


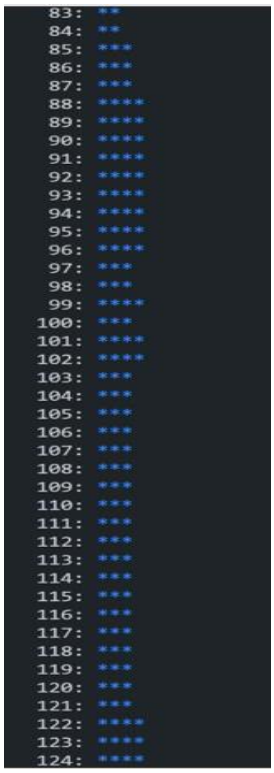


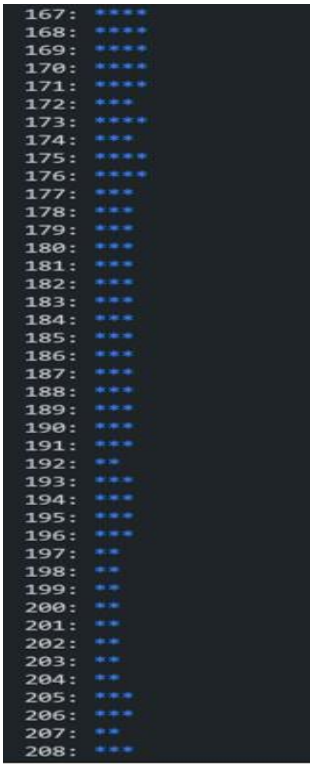


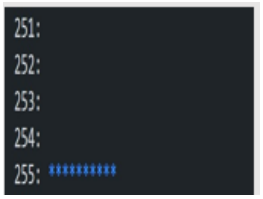


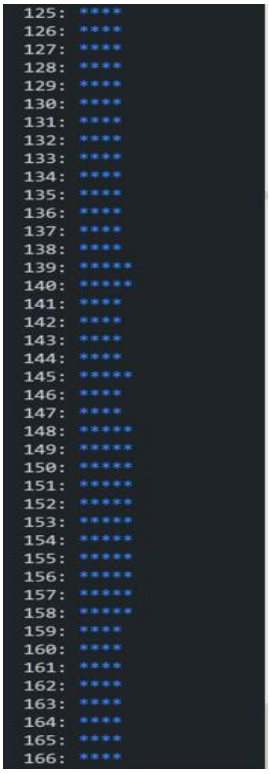
5.4 Blue Output Channel

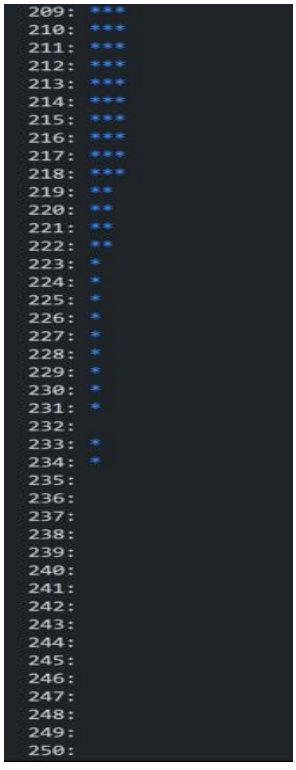












6. Future Scope

The future scope of histogram generation for an RGB image in C programming holds promising possibilities in several domains. Here are some potential future directions for this topic:

- ✓ **Machine Learning Integration:**
  Integration with machine learning algorithms for automated feature extraction and image classification. The RGB histogram data can serve as input features for training models, enhancing the capability of systems to recognize and categorize images.
- ✓ **Real-time Image Processing:**
  Expansion into real-time image processing applications, particularly in the context of embedded systems, robotics, and augmented reality. Optimizing the algorithm and implementation for faster processing can enable real-time analysis of live video streams.

7. Conclusion

In conclusion, the implementation of histogram generation for an RGB image in C programming stands as a versatile and potent tool with broad applicability in computer science and image processing. The program's capacity to analyse the distribution of pixel intensities across the red, green, and blue colour channels enhances its utility, making it suitable for diverse applications. Beyond its foundational role in education, offering a practical example for learning pixel manipulation, file input/output operations, and histogram analysis, the program proves valuable in real-world scenarios. It provides insights into colour composition, facilitating tasks like colour correction, channel manipulation, and colour-based feature extraction. Additionally, its potential applications span medical imaging, computer vision, and multimedia systems, contributing to tasks ranging from medical diagnosis to object recognition. The program's efficiency, ensured by the use of the C programming language, coupled with well-designed algorithms and data structures, establishes it as an essential building block for more advanced image processing pipelines. Ultimately, this implementation not only addresses fundamental aspects of image processing but also serves as a foundational component, laying the groundwork for further exploration and development in the dynamic fields of computer science and digital image analysis.

8. References

[1] Generating Histogram in C – Stack Overflow. (Overview for Basic 1D and 2D Arrays)

Link to Address

[2] Program To Print Histogram In C | C Graphics | Programmerbay.

Link to Address

[3] Histograms: A Useful Data Analysis Visualization**,** Regina L. Nuzzo PhD, online library

First published: 13 February 2019

Link to Address

[4] Research on Histogram Generation Algorithm Optimization Based on OpenCL

https://r.search.yahoo.com/_ylt=AwrKB1Ok8o5lomYeixe7HAx.;_....

[5] C Programming 19 – Draw a Histogram using Arrays of 1D and 2D – YouTube

https://r.search.yahoo.com/_ylt=Awrx.0oH9I5l7KYbjzC7HAx.;_.

[6] Fast multi-dimensional generalized histogram with convenient interface for C++14

https://r.search.yahoo.com/_ylt=Awrx.0oH9I5l7KYbmjC7HAx....

[7] Image – Processing Fundamentals | The C/C++ based Image Processing Learner's Toolkit.

https://r.search.yahoo.com/_ylt=AwrKADXy9I5lTc8abkC7.....