**ENDSEMESTER REPORT**

**on**

**TRANSFORMATION OF IMAGES USING IMAGE PROCESSING ALGORITHM**

**Submitted by**

SHISHIR GOYAL (Enroll No. R110216146)

SHIVANSH THAPLIYAL (Enroll. No. R110216150)

SIDDHARTH SHARMA (Enroll. No. R110216154)

SOURADEEP BANERJEE (Enroll. No. R110216159)

**Under the guidance of**

Dr. Jagdish Chandra Patni

Assistant Professor (SG), Department of Virtualization



**SCHOOL OF COMPUTER SCIENCE**

UNIVERSITY OF PETROLEUM & ENERGY STUDIES

Bidholi Campus, Energy Acres, Dehradun – 248007.

**December - 2018**

**CONTENTS**

[ABSTRACT 1](#_Toc532800965)

[INTRODUCTION 2](#_Toc532800966)

[PROBLEM STATEMENT 3](#_Toc532800967)

[LITERATURE REVIEW 3](#_Toc532800968)

[OBJECTIVES 4](#_Toc532800969)

[METHODOLOGY 4](#_Toc532800970)

[SYSTEM REQUIREMENTS 7](#_Toc532800971)

[Hardware Interface: 7](#_Toc532800972)

[Software Interface 7](#_Toc532800973)

[SCHEDULE 8](#_Toc532800974)

[DESIGN 9](#_Toc532800975)

[ALGORITHMS USED 10](#_Toc532800976)

[Brightness Modification Algorithm: 11](#_Toc532800977)

[Histogram Equalization 11](#_Toc532800978)

[Color to Greyscale Algorithm 12](#_Toc532800979)

[Blurring of Greyscale image 12](#_Toc532800980)

[Halftoning Algorithm 13](#_Toc532800981)

[Scaling of Image Algorithm 13](#_Toc532800982)

[Rotation of Image by any angle 14](#_Toc532800983)

[Sepia Filter Algorithm 15](#_Toc532800984)

[Negative Filter Algorithm 16](#_Toc532800985)

[IMPLEMENTATION AND RESULTS 16](#_Toc532800986)

[Brightness Modification 17](#_Toc532800987)

[Histogram Equalization 18](#_Toc532800988)

[Color to Greyscale 24](#_Toc532800989)

[Blurring of Greyscale Image 26](#_Toc532800990)

[Halftoning 28](#_Toc532800991)

[Scaling of Image 31](#_Toc532800992)

[Rotation by an Angle 35](#_Toc532800993)

[Sepia Filter 39](#_Toc532800994)

[Negative Filter 41](#_Toc532800995)

[Client-Server Architecture 43](#_Toc532800996)

[CONCLUSION 49](#_Toc532800997)

[FUTURESCOPE 49](#_Toc532800998)

[REFERENCES 49](#_Toc532800999)



**School of Computer Science**

**University of Petroleum & Energy Studies, Dehradun**

****I

**Minor**

**PROJECT TITLE:** TRANSFORMATION OF IMAGES USING IMAGE PROCESSING ALGORITHM

# ABSTRACT

Pixel by pixel image transformation of bitmap images to enhance it by making filters and applying transformations and showing results with actual photographs. Deploying this image processing module on a client-server architecture, we will model a real-world application where the user may upload his image to the server and the server would apply transformations according to the user’s specific needs. All this is implemented in C using image processing algorithms without using any predefined external library.

**Keywords:** bitmap images, histogram-equalization, Bitdepth.

# INTRODUCTION

Image processing is the process of converting an image into digital form, storing the image data in an array and then manipulating those values using different mathematical operations to do different types of editing and modifications. Various algorithms can be designed to do various types of operations after the image data is digitalized. We are using Bitmap which is a digital image composed of matrix of dots and each dot corresponds to an individual pixel on a display. Each pixel in bitmap image represent different color and together these pixels can be used to represent any type of rectangular picture.

Bitmap image (BMP) format is capable of storing 2-D digital images both monochrome and color.

Bitmap image header is of 54 bytes in which 18th bit stores image width, 22nd bit stores image height

and 28th bit stores Bitdepth. If value of Bitdepth is less than 8, then there is a color table of 1024

bytes. Color table is a block of bytes listing the colors used by image.

The project is basically divided in three modules - a client, a server and the program application running on the server with the image processing algorithms. Images will be read using file input output (I/O) routines and filters will be applied on the images to produce enhanced or edited images (according to user’s choice). Filtering would be done by understanding and modifying the bitmap image headers. Halftoning technique would be used to transform a grayscale image to a black and white image. Histogram equalization technique would be used to correct the poor contrast of the user’s images. Altering geometry of images would also be possible by displacement, scaling, rotations and cross product. Operations such as brightness, contrast, saturation would be performed. All these image processing will be executed on server side after the successful image transfer from client to server where server can handle multiple clients together using threads. Server will then transfer the new image back to the client. This server to client file transfer and vice versa will be achieved using TCP sockets by creating a connection between the two. For any read/write on server side there will be a corresponding write/read on client side and vice versa.

# PROBLEM STATEMENT

Raw images that are being captured in day to day life are not always of the quality that we want. Sometimes, we need to enhance the quality of the image, sharpen the image, increase the brightness or contrast, get the black and white copy of a image, etc. Information of images in the form of digital data is helpful for further processing. This is where our project will become handy. It will help the users to perform such operations deployed on a client server architecture.

# LITERATURE REVIEW

Here are the conclusions drawn of some of the reference papers that we will review to understand the importance of our project and give a light on how it can be extended using future technologies.

The client-server architecture is a way to distribute a service from a central source or server to number of clients [1]. In it, there is a single server that provides a service, and multiple clients that communicate with the server to consume its service. In a cloud computing architecture, there is a style of computing in which IT-related capabilities are provided “as a service”, allowing users to access technology-enabled services from the cloud system using internet connection. SaaS (Software as a Service) is one class of service provided by cloud wherein a software application is made available to clients over the network, which is more likely a server-client architecture. Thus, a server providing application services to a client over a network forms the basis of SaaS.

The prime objective of enhancement is to process an image so that the result is more suitable than the original image for a specific application [2]. Filters can be suitably used as robust technique for any image processing application like biometric devices, fingerprint scanner, X-ray where image enhancement is a pre-processing step before where human viewing is required for further processing. The process of image Enhancement is divided into two domains i.e. frequency domain and spatial domain. In frequency domain, techniques operate on frequency transform of the image whereas in spatial domain, techniques operate directly on the pixels of the image [3].

Histogram equalization is a nonlinear technique for adjusting the contrast of an image using its

histogram [4]. It still has limitations. Therefore, these limitations lead to the development of several histogram equalization (HE) methods. These methods generally fall into three main families of HE, namely Mean Brightness Preserving HE (MBPHE), Bin Modified HE (BMHE), and Local HE (LHE) [5].

Nowadays Bitmap (BMP) images are preferred in image processing compared to another formats because BMP image contain all the image information in a simple format. Therefore, in order to investigate the corruption point in JPEG, the file is required to be converted into BMP format [6].

# OBJECTIVES

The main purpose of the system is to enable the clients to connect to a particular server, send their images to the server and get the desired output by choosing the proper filter or edits. Using this the client does not require to download the particular code for that on their system and rather can directly access the server for the same.

**4.1 Sub-Objectives**:

* We are using Client Server architecture to achieve the file transfer from client to server and vice-versa.
* We are using the byte header format of the image in order to retrieve the pixel data of the image and manipulate the same to do the editing.

# METHODOLOGY

Development of a particular system involves a particular life cycle model to be followed in the whole process. The life cycle model that we are following is the Iterative model. Using this model, we design a simple version of the model and then enhance it for further functionalities.

Client-Server module/architecture is based on the principle of distributing the work-load among various programs called servers and different invoking programs called clients. The client requests some data from the server and the server responds the same using the programs written in the back end. Using this architecture, we can lower the load on client side and do the processing part on a centralized, high capacity server. The client server architecture that we are using in this project is based on the following principle:

**5.1 Server:** The server is used to connect with multiple clients and will be able to handle multiple clients using concept of threads. The steps followed by the server are:

* Step 1: Server waits for the client to connect using the IP.
* Step 2: The server listens for connection from client side and accepts if there is any
* Step 3: The server receives the original image from the client.
* Step 4: Server gets the option chosen by client and does the processing according to it on the image.
* Step 5: Server then sends back the updated image back to the Client.

**5.2 Client:** The client connects to the server using the IP and follows the following steps:

* Step 1: Client sends request for the connection.
* Step 2: After the connection is established the client sends the bitmap image to be processed by the server-side program.
* Step 3: The client chooses one of the choices from the options given to him.
* Step 4: The option is then sent to the server for the corresponding processing,
* Step 5: The client then receives the edited image for the server.

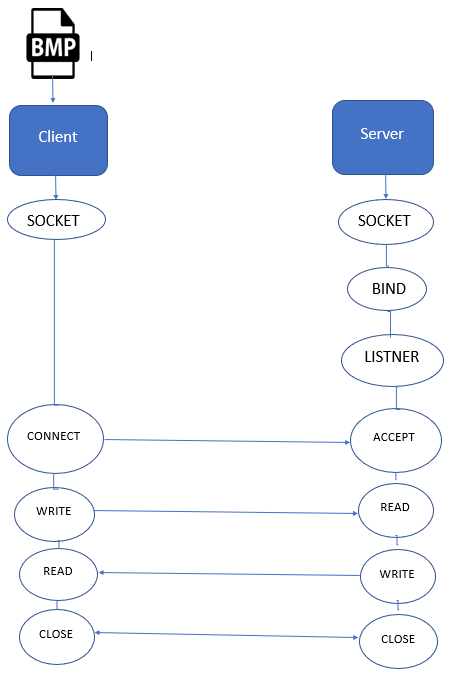


Figure No 1: Client Server Architecture for Image Transfer

Processing of the image is done by using the header format of the bmp image file. After receiving the binary data from the client side via TCP sockets, the server creates a file in binary append mode and appends the incoming binary data from the connection to the created bmp file. The image data is then extracted from the newly created bmp file (stored at server) by the server-side program, which has a 54 Byte header. From this header file we can get the height, width and bit-depth of the bmp image. The image size is then calculated by multiplying the height and width of the image. The image data is being read into a buffer. We do the required calculations on this buffer in order to modify the image data and then copy this data to the byte header of the output image bmp file (also stored at server). The output image file is then sent to the appropriate client by the server again via TCP sockets.

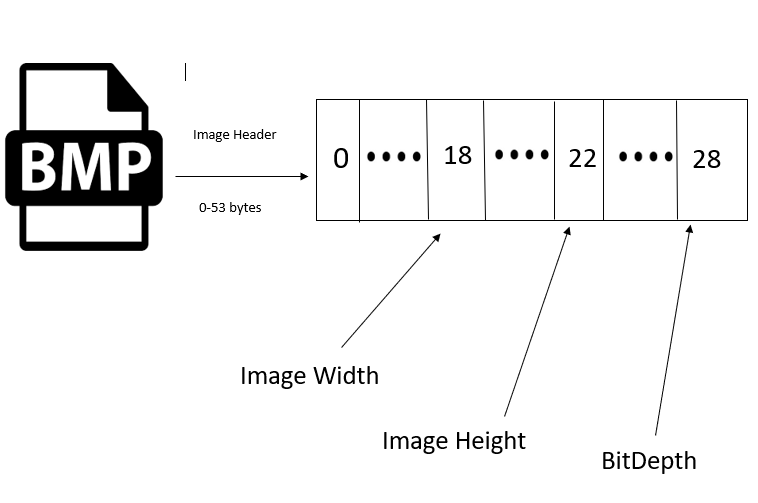


Figure No 2: .bmp Header Format

# SYSTEM REQUIREMENTS

## Hardware Interface:

* 64bit processor architecture supported by Linux.
* Minimum ram requirement for proper execution is 4GB.
* Storage Capacity of 100GB is sufficient.

Software Interface:

* OS Supported: Linux distribution
* The system is developed in C programming language.
* Compiler Required: GCC
* Image format supported: .bmp

# SCHEDULE

The schedule to be followed in this project is given in the form of the following pert chart:

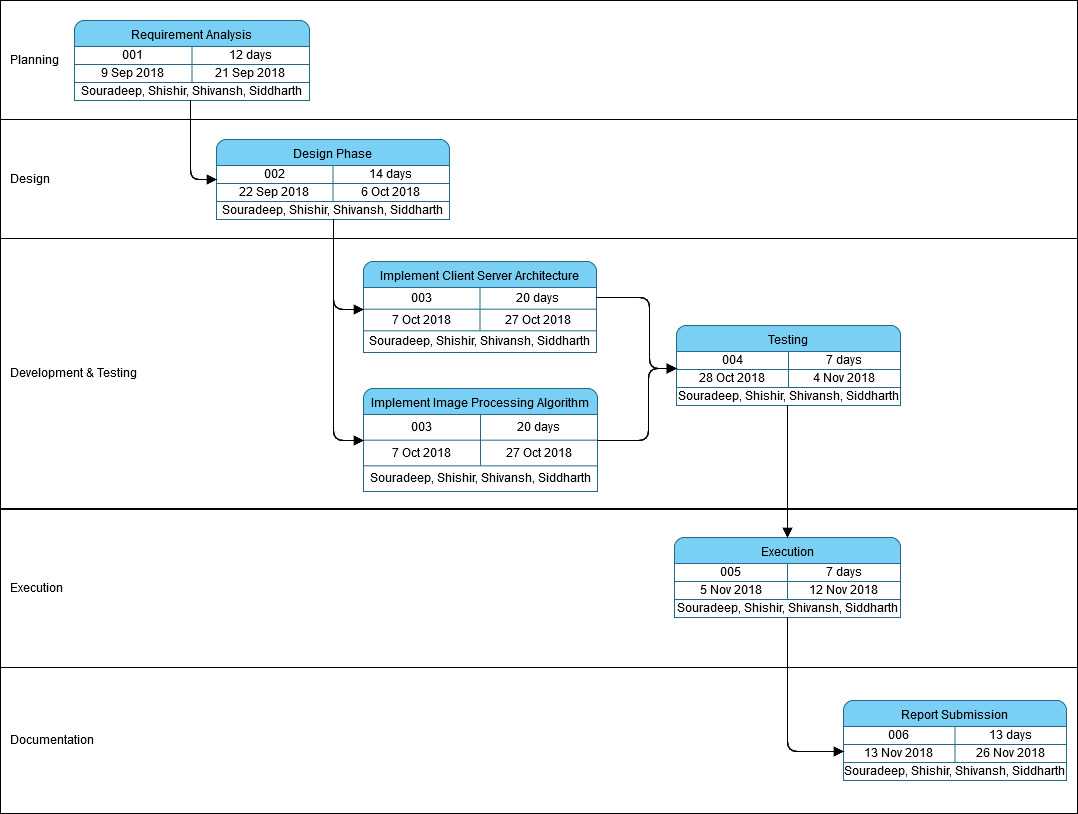


Figure No 3: Pert Chart

# DESIGN

Our program is being designed in two parts- one part involves designing the client server architecture and the other part involves the proper functioning of the image processing algorithms. The data flow diagram of the same is enclosed below:

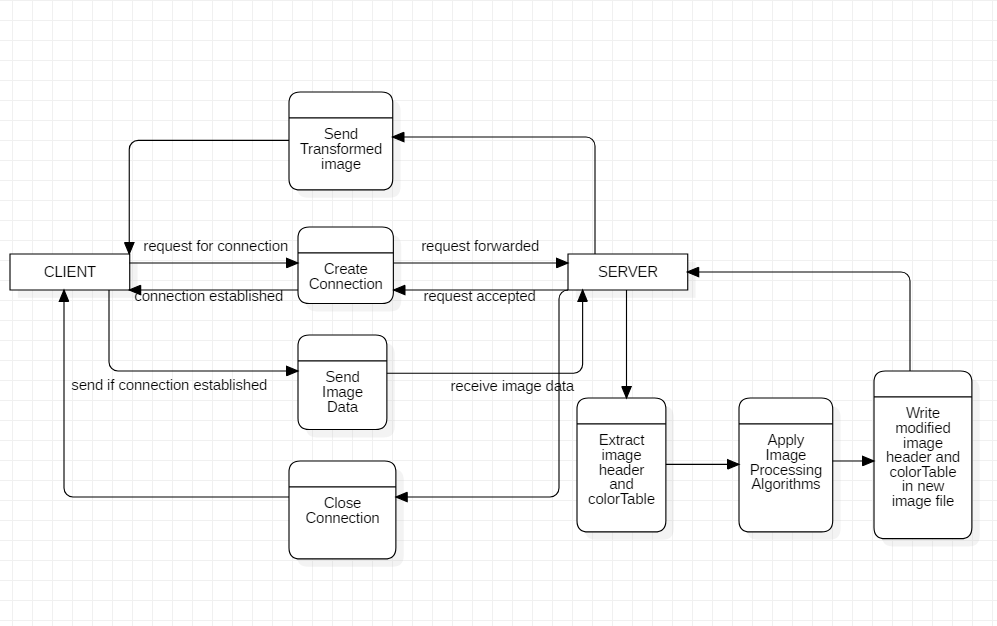


Figure No 4: Data Flow Diagram

The flow of the program operations is being given in the following diagram. It shows how the program interacts with the user and produces output. The client only sends the name of the file to be uploaded and all the processing work is being done on the server side.

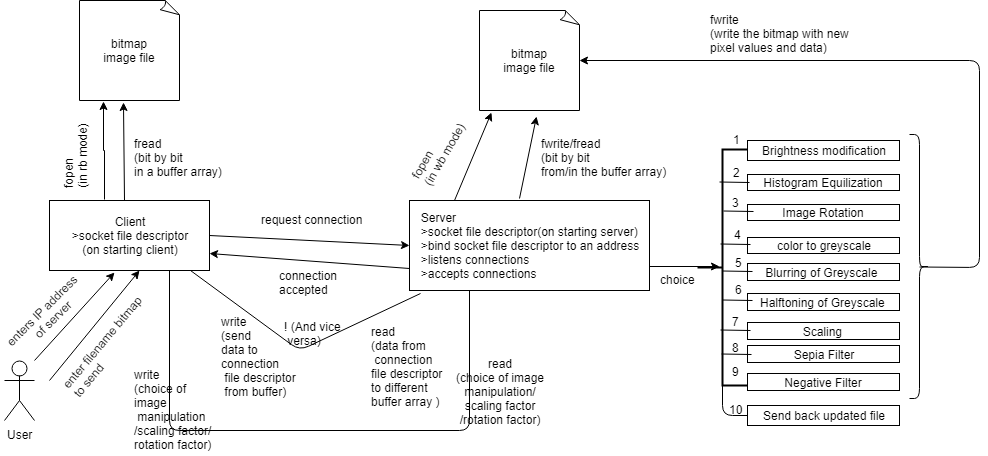


Figure No 5: Flow of the program

# ALGORITHMS USED

The various functionalities being used in our project to apply image processing on a given bitmap image are as follows:

* Brightness Modification
* Histogram Equalization
* Rotation of the image
* Color image to greyscale
* Blurring of grey scale image
* Halftoning of greyscale image
* Scaling of an image
* Rotation of image by any angle.
* Applying Sepia Filter
* Applying Negative Filter

## Brightness Modification Algorithm:

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer (first 54 Bytes)

STEP 4: Extract width (18th byte), height (22nd byte) and bitDepth (28th byte) from image header.

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer (first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: Calculate the size of the image by multiplying height and width.

STEP 9: Extract the image data and store it in an array of size of size of the image calculated earlier.

STEP 10: Increase the array values by the brightness increasing factor and store the updated value if it is less than 255 otherwise store 255.

STEP 11: The updated image data is written to the output image file using fwrite.

STEP 12: STOP

Histogram Equalization:

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer(first 54 Bytes)

STEP 4: Extract width(18th byte), height(22nd byte) and bitDepth(28th byte) from image header

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: Create a 2D input buffer to store greyscale values for each pixel for i=0 to height and j=0 to width:

input\_buffer[i][j]=getc(input\_image\_file)

input\_buffer[i][j]=getc(input\_image\_file)

input\_buffer[i][j]=getc(input\_\_image\_file)

STEP 9: Calculate pixel\_density in overall input image:

pixel\_density=1/(width\*height)

STEP 10: Create a cumulative\_pixel\_count array off 256 size to hold the frequency of pixels for each intensity value ranging from 0 t0 255 in the input greyscale image and initialize it to 0

cumulative\_pixel\_count[256]={0}

STEP 11: Calculate cumulative pixel\_count for i=0 to height and for j=0 to width

cumulative\_pixel\_count[input\_buffer[i][j]]+=pixel\_density

STEP 12: A desired histogram is set such that for each intensity values(i) ranging from 0 to 256

12.1) if intensity < 129

it has same number of pixel as the intensity value i.e desired\_pixel\_count[i]=i

else

it has 128-intensity\_value as number of pixel i.e desired\_pixel\_count[i]=128-i

12.2) Calculate total pixels in this desired histogram

desired\_total\_pixels+=desired\_pixel\_count[i]

STEP 13: Calculate cumulative desired\_pixel\_count for i=0 to 256

fraction+=desired\_pixel\_count[i]\*(1/desired\_total\_pixels)

cumulative\_desired\_pixel\_count=fraction

STEP 14: Now create another array for storing new image intensity values as a replacement for each of the 256 greyscale values in the input image

new\_intensity[256]={0}

STEP 15: Initialize J as 0 and for i=0 to 256

15.1) if cumulative\_pixel\_count[i]<desired\_cumulative\_pixel\_count[j]

new\_intensity[i]=j; // if i is old\_intensity to be replaced

15.2) else

10.2.1) while cumulative\_pixel\_count[i]>desired\_cumulative\_pixel\_count[j]

j++;

10.2.2) if desired\_cumulative\_pixel\_count[j]-cumulative\_pixel\_count[i] > cumulative\_pixel\_count[i]-desired\_cumulative\_pixel\_count[j-1]

new\_intensity[i]=j-1;

else

new\_intensity[i]=j;

STEP 16: Store the new intensity values present in output\_buffer for i=0 to height and j=0 to width and simultaneously store them to output image file

output\_buffer[i][j]=new\_intensity[input\_buffer[i][j]]

// Store replaced intensities for only those intesity values present in image

putc(output\_buffer[i][j],output\_image\_file)

STEP 17: STOP

Color to Greyscale Algorithm**:**

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer (first 54 Bytes)

STEP 4: Extract width (18th byte), height (22nd byte) and bitDepth (28th byte) from image header.

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: Create a 3D buffer to store RGB values for each pixel

STEP 9: Iterate through each pixel from Left to Right and Top to Bottom and store RGB values in 3D buffer

STEP 10: Using Luminosity method change value of each pixel using formula:(most efficient grey scale method)

g=0.21\*R+0.72\*G+0.07\*B

/\*Other methods to convert pixel from RBG to grey scale are:

Lightness method: (max(R,G,B)+min(R,G,B))/2

Average method: (R+G+B)/3 \*/

STEP 11: Store the modified pixel values back to the colorTable

STEP 12: STOP

Blurring of Greyscale image**:**

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer (first 54 Bytes)

STEP 4: Extract width (18th byte), height (22nd byte) and bitDepth (28th byte) from image header.

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: Now using mean filter method, create a blurring kernal masking matrix(3x3 with each element value 1/9)

/\* Rules for creating masking matrix:

-Must be of odd order

-Sum of elements should be one

-All elements should be same \*/

STEP 9: Now iterate through each pixel and calculate the kernal sum for each pixel

STEP 10: To obtain the kernal sum, multiply each adjacent pixel with kernal masking value and sum together

STEP 11: Now assign the kernal sum to the pixel and store it in new bitmap file

STEP 12: STOP

Halftoning Algorithm**:**

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer (first 54 Bytes)

STEP 4: Extract width (18th byte), height (22nd byte) and bitDepth (28th byte) from image header.

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: Calculate the size of the image by multiplying height and width.

STEP 9: Extract the image data and store it in an array of size of size of the image calculated earlier.

STEP 10: Compare the array values with threshold value, if the array value is greater than threshold value then we update it with 255, otherwise we update it with 0.

STEP 11: Updated image data is being written to the output image file.

STEP 12: STOP

Scaling of Image Algorithm**:**

STEP 1: START

STEP 2: Load an image in file pointer f

STEP 3: Extract the image header from file pointer(first 54 Bytes)

STEP 4: Extract width(18th byte), height(22nd byte) and bitDepth(28th byte) from image header

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Create a 3D input buffer to store RGB values for each pixel for i=0 to height and j=0 to width:

input\_buffer[2][i][j]=getc(input\_image\_file)

input\_buffer[1][i][j]=getc(input\_image\_file)

input\_buffer[0][i][j]=getc(input\_\_image\_file)

STEP 8: Input vertical resize factor and horizontal resize factorfrom user

STEP 9: Calculate:

new\_height= height \* vertical resize factor

new\_width= width \* horizontal resize factor

STEP 10: Write the imageHeader with new\_height & new\_width value and colorTable to another empty bitmap output file

STEP 11: Create a 3D output buffer to store RGB values for each pixel of output image

STEP 12: For i=0 to new\_height and j=0 to new\_width:

12.1)Calculate vertical as well as horizontal position of (i,j)th pixel of new image w.r.t original image:

vertical\_pos= i \* (height/new\_height) //Float value

horizontal\_pos= j \* (width/new\_width) //Float value

12.2) Calculate dimensions:

top= floor(vertical\_pos)

bottom= top+1

left= floor(horizontal\_pos)

right= left+1

12.3) Bilinealy interpolate the pixel according to above dimensions on input buffer for each of k in (R,G,B) values:

12.3.1) Determine the pixel value of the top-left, top-right, bottom-left, bottom-right pixel associated with vertical\_pos and horizontal\_pos

top\_left=input\_buffer[top][left]

top\_right=input\_buffer[top][right]

bottom\_left=input\_buffer[bottom][left]

bottom\_right=input\_buffer[bottom][right]

12.3.2) Calculate:

horizontal\_progress = horizontal\_position - left

vertical\_progress = vertical\_position - top

12.3.3) Calculate:

top\_block= top\_left + horizontal\_progress \* (top\_right - top\_left)

bottom\_block= bottom\_left + horizontal\_progress(bottom\_right - bottom\_left)

12.3.4) Calculate:

resulted\_pix\_value=top\_block + vertical\_progress \* (bottom\_block - top\_block);

12.3.5) Store it in output buffer:

output\_buffer[k][i][j]= resulted\_pix\_value

STEP 13: Store the new pixel values present in output\_buffer to output file for i=0 to new\_height and j=0 to new\_width:

putc(output\_buffer[2][i][j],output\_image\_file)

putc(output\_buffer[1][i][j],output\_image\_file)

putc(output\_buffer[0][i][j],output\_image\_file)

STEP 14: STOP

Rotation of Image by any angle:

STEP 1: START

STEP 2: Load an image in file pointer f

STEP 3: Extract the image header from file pointer(first 54 Bytes)

STEP 4: Extract width(18th byte), height(22nd byte) and bitDepth(28th byte) from image header

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Write the image Header and colorTable to another empty bitmap output file

STEP 8: Create a 3D input buffer to store RGB values for each pixel for i=0 to height and j=0 to width:

input\_buffer[2][i][j]=getc(input\_image\_file)

input\_buffer[1][i][j]=getc(input\_image\_file)

input\_buffer[0][i][j]=getc(input\_\_image\_file)

STEP 9: Input no. of degrees image to be rotated or rotation factor from user

STEP 10: Calculate:

vertical\_center= floor(height/2)

horizontal\_center= floor(width/2)

STEP 11: Create a 3D output buffer to store RGB values for each pixel of output image

STEP 12: For i=0 to height and j=0 to width:

11.2) Calculate:

angle= rotation\_factor \* pi/180

vertical\_position = cos(angle) \*(i - vertical\_center) + sin(angle) \* (j - horizontal\_center) + vertical\_center

horizontal\_position = (-sin(angle) \*(i - vertical\_center) + cos(angle) \* (j - horizontal\_center) + horizontal\_center)

12.2) Calculate dimensions:

top= floor(vertical\_pos)

bottom= top+1

left= floor(horizontal\_pos)

right= left+1

12.3) Bilinearly interpolate the pixel according to above dimensions on input buffer for each of k in (R,G,B) values:

11.3.1) Determine the pixel value of the top-left, top-right, bottom-left, bottom-right pixel associated with vertical\_pos and horizontal\_pos

top\_left=input\_buffer[top][left]

top\_right=input\_buffer[top][right]

bottom\_left=input\_buffer[bottom][left]

bottom\_right=input\_buffer[bottom][right]

12.3.2) Calculate:

horizontal\_progress = horizontal\_position - left

vertical\_progress = vertical\_position - top

12.3.3) Calculate:

top\_block= top\_left + horizontal\_progress \* (top\_right - top\_left)

bottom\_block= bottom\_left + horizontal\_progress(bottom\_right - bottom\_left)

12.3.4) Calculate:

resulted\_pix\_value=top\_block + vertical\_progress \* (bottom\_block - top\_block);

12.3.5) Store it in output buffer:

output\_buffer[k][i][j]= resulted\_pix\_value

STEP 13: Store the new pixel values present in output\_buffer to output file for i=0 to new\_height and j=0 to new\_width:

putc(output\_buffer[2][i][j],output\_image\_file)

putc(output\_buffer[1][i][j],output\_image\_file)

putc(output\_buffer[0][i][j],output\_image\_file)

STEP 14: STOP

Sepia Filter Algorithm:

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer(first 54 Bytes)

STEP 4: Extract width(18th byte), height(22nd byte) and bitDepth(28th byte) from image header

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: For i=0 to height and j=0 to width

8.1)Create a 3D buffer to store RGB values for each pixel

buffer[2][i][j] = getc(input\_file);

buffer[1][i][j]= getc(input\_file);

buffer[0][i][j]= getc(input\_file);

8.2)Calculate:

r = (buffer[0][i][j]\*0.393) + (buffer[1][i][j]\*0.769) + (buffer[2][i][j]\*0.189)

g = (buffer[0][i][j]\*0.349) + (buffer[1][i][j]\*0.686) + (buffer[2][i][j]\*0.168)

b = (buffer[0][i][j]\*0.272) + (buffer[1][i][j]\*0.534) + (buffer[2][i][j]\*0.131)

8.3)Store the modified pixel values back to the output file

putc(b,output\_file)

putc(g,output\_file)

putc(r,output\_file)

STEP 9: Close the output and input file

STEP 10: STOP

Negative Filter Algorithm:

STEP 1: START

STEP 2: Load an image in file pointer

STEP 3: Extract the image header from file pointer(first 54 Bytes)

STEP 4: Extract width(18th byte), height(22nd byte) and bitDepth(28th byte) from image header

STEP 5: Check the value of bitDepth, if less than or equal to 8 then color table is present

STEP 6: Extract the colorTable from file pointer(first 1024 Bytes)

STEP 7: Now write the imageHeader and colorTable to another empty file

STEP 8: For i=0 to height and j=0 to width

8.1) Create a 3D buffer to store RGB values for each pixel

buffer[2][i][j] = getc(input\_file);

buffer[1][i][j]= getc(input\_file);

buffer[0][i][j]= getc(input\_file);

8.2) Calculate:

r = 255-buffer[2][i][j];

g = 255-buffer[1][i][j];

b = 255-buffer[0][i][j];

8.3) Store the modified pixel values back to the output file

putc(b,output\_file)

putc(g,output\_file)

putc(r,output\_file)

STEP 9: Close the output and input file

STEP 10: STOP

# IMPLEMENTATION AND RESULTS

## Brightness Modification

#define A1 255

#define B1 25

void bright(FILE \*f1)

{

int r,g,b,r1,g1,b1;

FILE \*f2=fopen("final.bmp","w+");

unsigned char byte[54];

int i;

unsigned char ct[1024];

if(f1==NULL)

{

printf("File Not Found!!!");

}

for(i=0;i<54;i++)

{

byte[i]=getc(f1);

}

fwrite(byte,sizeof(unsigned char),54,f2);

int h=\*(int\*)&byte[22];

int w=\*(int\*)&byte[18];

int bd=\*(int\*)&byte[28];

int s=h\*w;

if(bd<=8)

{

fread(ct,sizeof(unsigned char),1024,f1);

fwrite(ct,sizeof(unsigned char),1024,f2);

}

unsigned char buff[3][h][w];

for (i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

r = 0;

g = 0;

b = 0;

r1 = buff[0][i][j] = getc(f1); //red

g1 = buff[1][i][j] = getc(f1); //green

b1 = buff[2][i][j] = getc(f1); //blue

r=r1+B1;

b=b1+B1;

g=g1+B1;

if(r>A){r=A1;}

if(g>A){g=A1;}

if(b>A){b=A1;}

putc(r, f2);

putc(g, f2);

putc(b, f2);

}

}

fclose(f1);

fclose(f2);

return 0;

}



Original Image Modified Image

## Histogram Equalization

#include <stdlib.h>

#include <stdio.h>

#define MAX\_VALUE 255

double\* alloc1dd (int n)

{

int i;

double\* array;

if ((array = (double\*) malloc(n \* sizeof(double))) == NULL) {

printf("Unable to allocate memory for 1D double array...\n");

exit(0);

}

for (i = 0; i < n; i++) {

array[i] = 0.0;

}

return array;

}

int\* alloc1di (int n)

{

int i, \*array;

if ((array = (int\*) malloc(n \* sizeof(int))) == NULL) {

printf("Unable to allocate memory for 1D int array...\n");

exit(0);

}

for (i = 0; i < n; i++) {

array[i] = 0.0;

}

return array;

}

float\* alloc1df (int n)

{

int i;

float\* array;

if ((array = (float\*) malloc(n \* sizeof(float))) == NULL) {

printf("Unable to allocate memory for 1D float array...\n");

exit(0);

}

for (i = 0; i < n; i++) {

array[i] = 0.0;

}

return array;

}

float\*\* alloc2df (int m, int n)

{

int i;

float\*\* array;

if ((array = (float \*\*) malloc(m \* sizeof(float\*))) == NULL) {

printf("Unable to allocate memory for 2D float array...\n");

exit(0);

}

for (i = 0; i < m; i++) {

array[i] = alloc1df(n);

}

return array;

}

float\*\*\* alloc3df (int l, int m, int n)

{

int i;

float\*\*\* array;

if ((array = (float \*\*\*) malloc(l \* sizeof(float\*\*))) == NULL) {

printf("Unable to allocate memory for 3D float array...\n");

exit(0);

}

for (i = 0; i < l; i++) {

array[i] = alloc2df(m,n);

}

return array;

}

void histo(FILE \*fIn)

{

//FILE \*fIn = fopen("Ref.bmp","r");

FILE \*f2 = fopen("histo.bmp","w+");

int i,j,k,r,g,b,r1,g1,b1;

unsigned char header[54];

if(fIn==NULL)

{

printf("File does not exist.\n");

}

for(i=0;i<54;i++)

{

header[i] = getc(fIn);

}

fwrite(header,sizeof(unsigned char),54,f2);

//fwrite(header,sizeof(unsigned char),17,f2);

int width = \*(int\*)&header[18];

int height = \*(int\*)&header[22];

int bitDepth = \*(int\*)&header[28];

int M\_in=height;

int N\_in=width;

//FILE \*f2 = fopen("histo.bmp","w+");

// Declare output.

float\*\*\* buffer=alloc3df(3, height, width);

float\*\*\* output = alloc3df(3, height, width);

for(i=0;i<height;i++){

for(j=0;j<width;j++){

buffer[2][i][j]= getc(fIn); //blue

buffer[1][i][j]= getc(fIn); //green

buffer[0][i][j]= getc(fIn); //red

if(r > MAX\_VALUE){ //if value exceeds

r = MAX\_VALUE;

}

if(g > MAX\_VALUE){

g = MAX\_VALUE;

}

if(b > MAX\_VALUE){

b = MAX\_VALUE;

}

//putc(b,f2);

//putc(g,f2);

//putc(r,f2);

}}

// Create the desired density histogram.

// int i, j, k;

int\* desired\_histogram = alloc1di(256);

double\* desired\_histogram\_count = alloc1dd(256);

int total\_pixels = 0;

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

if (i < 32) {

desired\_histogram[i] = i;

} else if(i>=32 && i<64) {

desired\_histogram[i] = 32 - i;

}

else if(i>=64 && i<96) {

desired\_histogram[i] = 64 + i;

}

else if(i>=96 && i<128) {

desired\_histogram[i] = 96 - i;

}

else if(i>=128 && i<160) {

desired\_histogram[i] = 128 + i;

}

else if(i>=160 && i<192) {

desired\_histogram[i] = 160 - i;

}

else if(i>=192 && i<224) {

desired\_histogram[i] = 192 + i;

}

else {

desired\_histogram[i] = 224 - i;

}

total\_pixels += desired\_histogram[i];

}

// Create the desired cumulative distribution for the above histogram.

double fraction = 0.0;

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

fraction += (double) desired\_histogram[i] \* (1 / (double)

total\_pixels);

desired\_histogram\_count[i] = fraction;

}

// Prepare to calculate the cumulative histogram.

double\* pixel\_count;

int\* new\_histogram;

float pixel\_density = 1 / (float) (M\_in \* N\_in);

// Equalize the histogram on all three channels.

for (k = 0; k < 3; k++) {

pixel\_count = alloc1dd(256);

new\_histogram = alloc1di(256);

// Construct the current cumulative histogram.

for (i = 0; i < M\_in; i++) {

for (j = 0; j < N\_in; j++) {

pixel\_count[(int) buffer[k][i][j]] += pixel\_density;

}

}

// Create the lookup table for the new values of each grayscale color.

j = 0;

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

if (pixel\_count[i] <= desired\_histogram\_count[j]) {

new\_histogram[i] = j;

} else {

while (pixel\_count[i] > desired\_histogram\_count[j]) {

j++;

}

if (desired\_histogram\_count[j] - pixel\_count[i] >

pixel\_count[i] - desired\_histogram\_count[j - 1]) {

new\_histogram[i] = j - 1;

} else {

new\_histogram[i] = j;

}

}

}

// Create the new image.

for (i = 0; i < M\_in; i++) {

for (j = 0; j < N\_in; j++) {

output[k][i][j] = new\_histogram[(int) buffer[k][i][j]];

}

}

free(pixel\_count);

free(new\_histogram);

}

free(desired\_histogram);

free(desired\_histogram);

for(i=0;i<M\_in;i++){

for(j=0;j<N\_in;j++){

putc(output[2][i][j],f2);

putc(output[1][i][j],f2);

putc(output[0][i][j],f2);

}

}

fclose(f2);

}



Original Image Modified Image

## Color to Greyscale

#define A 128

#define B 0

#define C 255

#define MAX\_VALUE 255

#define MIN\_VALUE 0

void greyscale(FILE \*f1)

{

FILE \*f2 = fopen("grey.bmp","w+");

int i, j, r, g, b, r1, g1, b1,y;

unsigned char header[54];

if (f1 == NULL)

{

printf("File does not exist.\n");

exit(0);

}

for (i = 0; i < 54; i++)

{

header[i] = getc(f1);

}

fwrite(header, sizeof(unsigned char), 54, f2);

int height = \* (int \* ) & header[22];

int width = \* (int \* ) & header[18];

printf("width: %d\n", width);

printf("height: %d\n", height);

unsigned char buffer[3][height][width];

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

r = 0;

g = 0;

b = 0;

r1 = buffer[0][i][j] = getc(f1);

g1 = buffer[1][i][j] = getc(f1);

b1 = buffer[2][i][j] = getc(f1);

y=(buffer[0][i][j]\*0.21) + (buffer[1][i][j]\*0.72) + (buffer[2][i][j]\*0.07);

putc(y, f2);

putc(y, f2);

putc(y, f2);

}

}

fclose(f2);

}



Original Image Modified Image

## Blurring of Greyscale Image

#define A 128

#define B 0

#define C 255

void blur(FILE \*f1)

{

//FILE \*fIn = fopen("GreyScale10.bmp","r");

FILE \*f2 = fopen("blur2.bmp","w+");

int i,j,x,y;

unsigned char byte[54];

if(f1==NULL)

{

printf("File does not exist.\n");

}

for(i=0;i<54;i++)

{

byte[i] = getc(f1);

}

fwrite(byte,sizeof(unsigned char),54,f2);

int height = \*(int\*)&byte[18];

int width = \*(int\*)&byte[22];

int bitDepth = \*(int\*)&byte[28];

printf("width: %d\n",width);

printf("height: %d\n",height );

int size = height\*width;

unsigned char buffer[size][3];

unsigned char out[size][3];

for(i=0;i<size;i++)

{

buffer[i][2]=getc(f1);

buffer[i][1]=getc(f1);

buffer[i][0]=getc(f1);

}

float v=1.0 / 25.0;

float kernel[5][5]={{v,v,v,v,v},{v,v,v,v,v},{v,v,v,v,v},{v,v,v,v,v},{v,v,v,v,v}};

for(x=1;x<height-1;x++)

{

for(y=1;y<width-1;y++)

{

float sum0= 0.0;

float sum1= 0.0;

float sum2= 0.0;

for(i=-1;i<=1;++i)

{

for(j=-2;j<=2;++j)

{

sum0=sum0+(float)kernel[i+1][j+1]\*buffer[(x+i)\*width+(y+j)][0];

sum1=sum1+(float)kernel[i+1][j+1]\*buffer[(x+i)\*width+(y+j)][1];

sum2=sum2+(float)kernel[i+1][j+1]\*buffer[(x+i)\*width+(y+j)][2];

}

}

out[(x)\*width+(y)][0]=sum0;

out[(x)\*width+(y)][1]=sum1;

out[(x)\*width+(y)][2]=sum2;

}

}

for(i=0;i<size;i++)

{

putc(out[i][2],f2);

putc(out[i][1],f2);

putc(out[i][0],f2);

}

fclose(f2);

return 0;

}



Original Image Modified Image

## Halftoning

#define A 128

#define B 0

#define C 255

void half(FILE \*f1)

{

int r,g,b,r1,g1,b1;

FILE \*f2=fopen("half.bmp","w+");

unsigned char byte[54];

int i;

unsigned char ct[1024];

if(f1==NULL)

{

printf("File Not Found!!!");

}

for(i=0;i<54;i++)

{

byte[i]=getc(f1);

}

fwrite(byte,sizeof(unsigned char),54,f2);

int h=\*(int\*)&byte[22];

int w=\*(int\*)&byte[18];

int bd=\*(int\*)&byte[28];

int s=h\*w;

if(bd<=8)

{

fread(ct,sizeof(unsigned char),1024,f1);

fwrite(ct,sizeof(unsigned char),1024,f2);

}

unsigned char buff[3][h][w];

for (i = 0; i < h; i++) {

for (int j = 0; j < w; j++) {

r = 0;

g = 0;

b = 0;

r1 = buff[0][i][j] = getc(f1); //red

g1 = buff[1][i][j] = getc(f1); //green

b1 = buff[2][i][j] = getc(f1); //blue

r = (buff[0][i][j]\*0.21) + (buff[1][i][j]\*0.72) + (buff[2][i][j]\*0.07);

g = (buff[0][i][j]\*0.21) + (buff[1][i][j]\*0.72) + (buff[2][i][j]\*0.07);

b = (buff[0][i][j]\*0.21) + (buff[1][i][j]\*0.72) + (buff[2][i][j]\*0.07);

if (r < A)

{

r = B;

}

else{r=C;}

if (g < A)

{

g = B;

}

else{g=C;}

if (b < A)

{

b = B;

}

else{b=C;}

putc(r, f2);

putc(g, f2);

putc(b, f2);

}

}

//for gray scale

/\*fread(buff,sizeof(unsigned char),s,f1);

for(i=0;i<s;i++)

{

if(buff[i]>A)

{buff[i]=C;}

else{buff[i]=B;}

}\*/

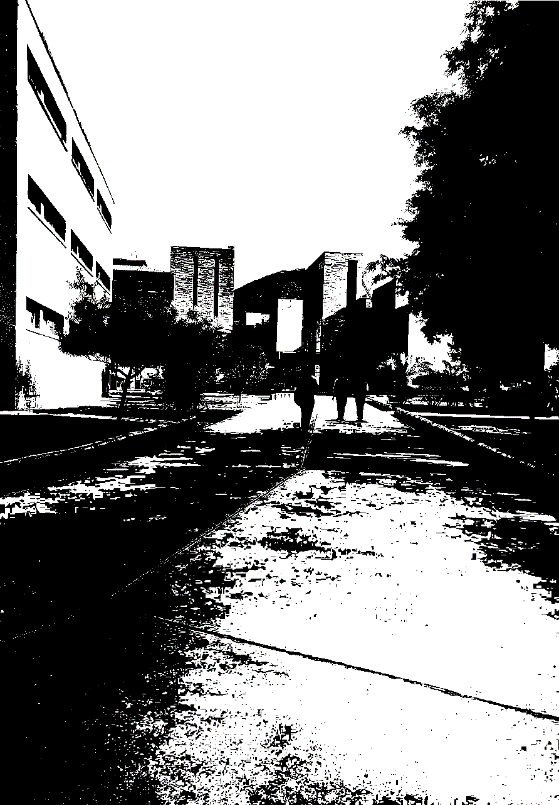
//fwrite(buff,sizeof(unsigned char),s,f2);

fclose(f1);

fclose(f2);

return 0;

}



Original Image Modified Image

## Scaling of Image

#include <stdio.h>

#include <time.h>

#include <math.h>

#include<stdlib.h>

#define MAX\_VALUE 255

float\* alloc1df (int n)

{

int i;

float\* array;

if ((array = (float\*) malloc(n \* sizeof(float))) == NULL) {

printf("Unable to allocate memory for 1D float array...\n");

exit(0);

}

for (i = 0; i < n; i++) {

array[i] = 0.0;

}

return array;

}

float\*\* alloc2df (int m, int n)

{

int i;

float\*\* array;

if ((array = (float \*\*) malloc(m \* sizeof(float\*))) == NULL) {

printf("Unable to allocate memory for 2D float array...\n");

exit(0);

}

for (i = 0; i < m; i++) {

array[i] = alloc1df(n);

}

return array;

}

float\*\*\* alloc3df (int l, int m, int n)

{

int i;

float\*\*\* array;

if ((array = (float \*\*\*) malloc(l \* sizeof(float\*\*))) == NULL) {

printf("Unable to allocate memory for 3D float array...\n");

exit(0);

}

for (i = 0; i < l; i++) {

array[i] = alloc2df(m,n);

}

return array;

}

float bilinearly\_interpolate (int top, int bottom, int left, int right,float horizontal\_position, float vertical\_position, float \*\*input)

{

float top\_left = input[top][left];

float top\_right = input[top][right];

float bottom\_left = input[bottom][left];

float bottom\_right = input[bottom][right];

float horizontal\_progress = horizontal\_position - (float) left;

float vertical\_progress = vertical\_position - (float) top;

float top\_block = top\_left + horizontal\_progress\* (top\_right - top\_left);

float bottom\_block = bottom\_left + horizontal\_progress \* (bottom\_right - bottom\_left);

return top\_block + vertical\_progress \* (bottom\_block - top\_block);

}

void Scaling(FILE \*f1)

{

FILE \*f2 = fopen("Scaled.bmp","w+");

int i,j,k,r,g,b,r1,g1,b1;

unsigned char header[54];

if(f1==NULL)

{

printf("File does not exist.\n");

}

for(i=0;i<54;i++)

{

header[i] = getc(f1);

}

int width = \*(int\*)&header[18];

int height = \*(int\*)&header[22];

int bitDepth = \*(int\*)&header[28];

int \*wt=(int\*)&header[18];

int \*ht=(int\*)&header[22];

int size = height\*width;

float verti\_resize\_factor, horiz\_resize\_factor;

printf("By what factor do you want to scale your image horizontally?\n");

scanf("%f", &horiz\_resize\_factor);

printf("By what factor do you want to scale your image vertically?\n");

scanf("%f", &verti\_resize\_factor);

printf("Resizing...\n");

int new\_height = height \* verti\_resize\_factor;

int new\_width = width \* horiz\_resize\_factor;

\*ht=new\_height;

\*wt=new\_width;

fwrite(header,sizeof(unsigned char),54,f2);

float \*\*\* output = alloc3df(3, \*ht, \*wt);

float\*\*\* buffer = alloc3df(3, height, width);

for(i=0;i<height;i++){

for(j=0;j<width;j++){

buffer[2][i][j] = getc(f1);

buffer[1][i][j]= getc(f1);

buffer[0][i][j]= getc(f1);

if(r > MAX\_VALUE){

r = MAX\_VALUE;

}

if(g > MAX\_VALUE){

g = MAX\_VALUE;

}

if(b > MAX\_VALUE){

b = MAX\_VALUE;

}

}}

for (i = 0; i < new\_height; ++i) {

for (j = 0; j < new\_width; ++j) {

float vertical\_position = i \* ( (float) height / new\_height);

float horizontal\_position = j \* ( (float) width / new\_width);

int top = (vertical\_position);

int bottom = top + 1;

int left = (horizontal\_position);

int right = left + 1;

if (bottom >= height) {

bottom = top;

}

if (right >= width) {

right = left;

}

for (k = 0; k < 3; k++) {

float interpolated = bilinearly\_interpolate(top, bottom, left,right, horizontal\_position, vertical\_position, buffer[k]);

output[k][i][j] = interpolated;

}

}

}

for(i=0;i<new\_height;i++){

for(j=0;j<new\_width;j++){

putc(output[2][i][j],f2);

putc(output[1][i][j],f2);

putc(output[0][i][j],f2);

}

}

fclose(f2);

printf("Resizing Done\n");

}



Original Image Modified Image

## Rotation by an Angle

#include <stdio.h>

#include <math.h>

#include<stdlib.h>

#define Pi 3.1414

#define MAX\_VALUE 255

float\* alloc1df (int n)

{

int i;

float\* array;

if ((array = (float\*) malloc(n \* sizeof(float))) == NULL) {

printf("Unable to allocate memory for 1D float array...\n");

exit(0);

}

for (i = 0; i < n; i++) {

array[i] = 0.0;

}

return array;

}

float\*\* alloc2df (int m, int n)

{

int i;

float\*\* array;

if ((array = (float \*\*) malloc(m \* sizeof(float\*))) == NULL) {

printf("Unable to allocate memory for 2D float array...\n");

exit(0);

}

for (i = 0; i < m; i++) {

array[i] = alloc1df(n);

}

return array;

}

float\*\*\* alloc3df (int l, int m, int n)

{

int i;

float\*\*\* array;

if ((array = (float \*\*\*) malloc(l \* sizeof(float\*\*))) == NULL) {

printf("Unable to allocate memory for 3D float array...\n");

exit(0);

}

for (i = 0; i < l; i++) {

array[i] = alloc2df(m,n);

}

return array;

}

float bilinearly\_interpolate (int top, int bottom, int left, int right,float horizontal\_position, float vertical\_position, float \*\*input)

{

float top\_left = input[top][left];

float top\_right = input[top][right];

float bottom\_left = input[bottom][left];

float bottom\_right = input[bottom][right];

float horizontal\_progress = horizontal\_position - (float) left;

float vertical\_progress = vertical\_position - (float) top;

float top\_block = top\_left + horizontal\_progress\* (top\_right - top\_left);

float bottom\_block = bottom\_left + horizontal\_progress \* (bottom\_right - bottom\_left);

return top\_block + vertical\_progress \* (bottom\_block - top\_block);

}

void main()

{

FILE \*f1 = fopen("Ref.bmp","r");

FILE \*f2 = fopen("Rotated.bmp","w+");

int i,j,k,r,g,b,r1,g1,b1;

unsigned char header[54];

if(f1==NULL)

{

printf("File does not exist.\n");

}

for(i=0;i<54;i++)

{

header[i] = getc(f1);

}

int width = \*(int\*)&header[18];

int height = \*(int\*)&header[22];

int bitDepth = \*(int\*)&header[28];

int size = height\*width;

float rot\_factor;

printf("By what factor do you want to rotate your image ?\n");

scanf("%f", &rot\_factor);

printf("Rotating...\n");

double angle = rot\_factor \* (double) Pi / 180;

fwrite(header,sizeof(unsigned char),54,f2);

float \*\*\* output = alloc3df(3, height, width);

float\*\*\* buffer = alloc3df(3, height, width);

for(i=0;i<height;i++){

for(j=0;j<width;j++){

buffer[2][i][j] = getc(f1); //blue

buffer[1][i][j]= getc(f1); //green

buffer[0][i][j]= getc(f1); //red

if(r > MAX\_VALUE){ //if value exceeds

r = MAX\_VALUE;

}

if(g > MAX\_VALUE){

g = MAX\_VALUE;

}

if(b > MAX\_VALUE){

b = MAX\_VALUE;

}

}}

int vertical\_center = floor(height / 2);

int horizontal\_center = floor(width / 2);

for (i = 0; i < height; ++i) {

for (j = 0; j < width; ++j) {

float vertical\_position = (float) cos(angle) \*(i - vertical\_center) + sin(angle) \* (j - horizontal\_center)+ vertical\_center;

float horizontal\_position = (float) -sin(angle) \*(i - vertical\_center) + cos(angle) \* (j - horizontal\_center)+ horizontal\_center;

int top = (vertical\_position);

int bottom = top + 1;

int left = (horizontal\_position);

int right = left + 1;

if (top >= 0 && bottom < height && left >= 0 && right < width ) {

for (k = 0; k < 3; k++) {

float interpolated = bilinearly\_interpolate(top, bottom, left,right, horizontal\_position, vertical\_position, buffer[k]);

output[k][i][j] = interpolated;

}

}

}

}

for(i=0;i<height;i++){

for(j=0;j<width;j++){

putc(output[2][i][j],f2);

putc(output[1][i][j],f2);

putc(output[0][i][j],f2);

}

}

fclose(f2);

fclose(f1);

printf("Rotation Done\n");

}



Original Image Modified Image

## Sepia Filter

#define MAX\_VALUE 255

#define MIN\_VALUE 0

void sepia(FILE \*f1) {

FILE \* f2 = fopen("Sepia.bmp", "w+");

int i, j, r, g, b, r1, g1, b1;

unsigned char header[54];

if (f1 == NULL)

{

printf("File does not exist.\n");

exit(0);

}

for (i = 0; i < 54; i++)

{

header[i] = getc(f1);

}

fwrite(header, sizeof(unsigned char), 54, f2);

int height = \* (int \* ) & header[22];

int width = \* (int \* ) & header[18];

printf("width: %d\n", width);

printf("height: %d\n", height);

unsigned char buffer[3][height][width];

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

r = 0;

g = 0;

b = 0;

r1 = buffer[0][i][j] = getc(f1);

g1 = buffer[1][i][j] = getc(f1);

b1 = buffer[2][i][j] = getc(f1);

b = (buffer[0][i][j]\*0.393)+(buffer[0][i][j]\*0.769)+(buffer[0][i][j]\*0.189);

g = (buffer[0][i][j]\*0.349)+(buffer[0][i][j]\*0.686)+(buffer[0][i][j]\*0.168);

r = (buffer[0][i][j]\*0.272)+(buffer[0][i][j]\*0.534)+(buffer[0][i][j]\*0.131);

if(r > MAX\_VALUE){

r = MAX\_VALUE;

}

if(g > MAX\_VALUE){

g = MAX\_VALUE;

}

if(b > MAX\_VALUE){

b = MAX\_VALUE;

}

putc(r, f2);

putc(g, f2);

putc(b, f2);

}

}

fclose(f2);

}



Original Image Modified Image

## Negative Filter

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VALUE 255

#define MIN\_VALUE 0

void negative(FILE\* f1) {

FILE \* f2 = fopen("Negative.bmp", "w+"); //Output File name

int i, j, r, g, b, r1, g1, b1;

unsigned char header[54];

if (f1 == NULL) // check if the input file present

{

printf("File does not exist.\n");

exit(0);

}

for (i = 0; i < 54; i++) //read the 54 byte header of input file

{

header[i] = getc(f1);

}

//writing the header back as it is

fwrite(header, sizeof(unsigned char), 54, f2);

// extract image height, width from Header

int height = \* (int \* ) & header[22];

int width = \* (int \* ) & header[18];

printf("width: %d\n", width);

printf("height: %d\n", height);

unsigned char buffer[3][height][width]; //to store the image data

for (i = 0; i < height; i++) {

for (j = 0; j < width; j++) {

r = 0;

g = 0;

b = 0;

r1 = buffer[0][i][j] = getc(f1); //red

g1 = buffer[1][i][j] = getc(f1); //green

b1 = buffer[2][i][j] = getc(f1); //blue

//conversion formula of rgb to negative

r = (255 - r1);

g = (255 - g1);

b = (255 - b1);

//if value in less than MIN\_VALUE then the pixel should not be lighted

if (r < MIN\_VALUE)

{

r = MIN\_VALUE;

}

if (g < MIN\_VALUE)

{

g = MIN\_VALUE;

}

if (b < MIN\_VALUE)

{

b = MIN\_VALUE;

}

putc(r, f2);

putc(g, f2);

putc(b, f2);

}

}

fclose(f2);

return 0;

}



Original Image Modified Image

## Client-Server Architecture

**Server-Side Code**:

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

FILE \*fn;

#include"half.h"

#include"bright.h"

#include"blur.h"

#include"sepia.h"

#include"greyscale.h"

#include"Negative.h"

#include"rot.h"

#include”rot1.h”

#include"Scaling.h"

#inlcude"hist.h"

struct sockaddr\_in c\_addr;

char fname[100];

FILE \*fp;

int receiveFile(int connfd)

{

int bytesReceived = 0;

char recvBuff[1024];

memset(recvBuff, '0', sizeof(recvBuff));

read(connfd, fname, 256);

strcat(fname,"1");

printf("File Name: %s\n",fname);

printf("Receiving file...");

strcat(fname,"1");

fp = fopen(fname, "ab");

if(NULL == fp)

{

printf("Error opening file");

return 1;

}

long double sz=1;

/\* Receive data in chunks of 256 bytes \*/

while((bytesReceived = read(connfd, recvBuff, 1024)) > 0)

{

sz++;

// gotoxy(0,4);

//printf("Received: %llf Mb",(sz/1024));

fflush(stdout);

// recvBuff[n] = 0;

fwrite(recvBuff, 1,bytesReceived,fp);

// printf("%s \n", recvBuff);

}

if(bytesReceived < 0)

{

printf("\n Read Error \n");

}

printf("\nFile OK....Completed\n");

return 0;

}

void\* Handle(int \*arg)

{

int connfd=(int)\*arg;

int choice;

printf("Connection accepted and id: %d\n",connfd);

printf("Connected to Clent: %s:%d\n",inet\_ntoa(c\_addr.sin\_addr),ntohs(c\_addr.sin\_port));

receiveFile(connfd);

printf("======Welcome to the Image Editor======")

printf("\tMenu\n");

printf("1\tHalftoning\n");

printf("2\tBrightness\n");

printf("3\tBlurring\n");

printf("4\tColor to GreyScale\n");

printf("5\tNegative\n");

printf("6\tSepia\n");

printf("7\tRotate\n");

printf("8\tScale an Image\n");

printf("9\tHistogram Equalization\n");

printf(“10\tRotate by an Angle\n”);

printf("Enter your choice:\t");

scanf("%d",&choice);

FILE \*f1=fopen(fname,"r");

switch(choice)

{

case 1:half(f1);break;

case 2:bright(f1);break;

case 3:blur(f1);break;

case 4:greyscale(f1);break;

case 5:negative(f1);break;

case 6:sepia(f1);break;

case 7:rot(f1);break;

case 8:Scaling(f1);break;

case 9:hist(f1);break;

case 10:rot1(f1);break;

default:printf("Wrong");break;

}

printf("Closing Connection for id: %d\n",connfd);

close(connfd);

shutdown(connfd,SHUT\_WR);

sleep(2);

}

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

{

int connfd = 0,err;

pthread\_t tid;

struct sockaddr\_in serv\_addr;

int listenfd = 0,ret;

char sendBuff[1025];

int numrv;

size\_t clen=0;

listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

if(listenfd<0)

{

printf("Error in socket creation\n");

exit(2);

}

printf("Socket retrieve success\n");

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

serv\_addr.sin\_port = htons(5000);

ret=bind(listenfd, (struct sockaddr\*)&serv\_addr,sizeof(serv\_addr));

if(ret<0)

{

printf("Error in bind\n");

exit(2);

}

if(listen(listenfd, 10) == -1)

{

printf("Failed to listen\n");

return -1;

}

/\*if (argc < 2)

{

printf("Enter file name to send: ");

gets(fname);

}

else

strcpy(fname,argv[1]);

\*/

while(1)

{

clen=sizeof(c\_addr);

printf("Waiting...\n");

connfd = accept(listenfd, (struct sockaddr\*)&c\_addr,&clen);

if(connfd<0)

{

printf("Error in accept\n");

continue;

}

err = pthread\_create(&tid, NULL, &Handle, &connfd);

if (err != 0)

printf("\ncan't create thread :[%s]", strerror(err));

}

close(connfd);

return 0;

}

**Client-Side Code**:

#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <netdb.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <arpa/inet.h>

char fname[100];

FILE \*fp;

int sendFile(int sockfd)

{

write(sockfd, fname,256);

fp = fopen(fname,"rb");

if(fp==NULL)

{

printf("File opern error");

return 1;

}

/\* Read data from file and send it \*/

while(1){

/\* First read file in chunks of 256 bytes \*/

unsigned char buff[1024]={0};

int no\_of\_read = fread(buff,1,1024,fp);

//printf("Bytes read %d \n", no\_of\_read);

/\* If read was success, send data. \*/

if(no\_of\_read > 0)

{

//printf("Sending \n");

write(sockfd, buff, no\_of\_read);

}

if (no\_of\_read < 1024)

{

if (feof(fp))

{

printf("End of file\n");

printf("File transfer completed for id: %d\n",sockfd);

}

if (ferror(fp))

printf("Error reading\n");

break;

}

//printf("Closing Connection for id: %d\n",sockfd);

//close(sockfd);

//shutdown(sockfd,SHUT\_WR);

}

return 0;

}

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

{

system("clear");

int sockfd = 0;

struct sockaddr\_in serv\_addr;

/\* Create a socket first \*/

if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0))< 0)

{

printf("\n Error : Could not create socket \n");

return 1;

}

/\* Initialize sockaddr\_in data structure \*/

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(5000); // port

char ip[50];

if (argc < 2)

{

printf("Enter IP address to connect: ");

scanf("%s",ip);

}

else

strcpy(ip,argv[1]);

serv\_addr.sin\_addr.s\_addr = inet\_addr(ip);

/\* Attempt a connection \*/

if(connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr))<0)

{

printf("\n Error : Connect Failed \n");

return 1;

}

printf("Connected to ip: %s : %d\n",inet\_ntoa(serv\_addr.sin\_addr),ntohs(serv\_addr.sin\_port));

/\* Create file where data will be stored \*/

FILE \*fp;

printf("Enter file name to send: ");

scanf("%s",fname);

sendFile(sockfd);

return 0;

}

# CONCLUSION

In Conclusion, Processed images that we get after applying image processing algorithms will enhance it features. Images can be smoothened using Blurring algorithm, errors can be rectified using Histogram Equalization, brightness can be increased using Brightness algorithm, Negative images can be created by applying Negative filter that will be helpful in analyzing the image. Physical dimensions of the image can be changed by using rotation and scaling algorithm.

Also, server will process images on client-server architecture so that load on client side will be reduced. Client will only send images to server and server will do all the processing and return client a enhanced image.

# FUTURESCOPE

In Future, we can compress and encrypt the images so that it can be safely exchanged over the client-server architecture. Edge Detection algorithm can also be used to detect objects of various shapes from the images. Further, we can deploy the whole project as a service on the cloud instead of client-server architecture.

Also, other developers can use the results of enhanced images for further image processing operations.

# REFERENCES

1. Prabhjot Kaur & Reena Rani, “Distributed and Cloud Computing Architecture”, Imperial Journal of Interdisciplinary Research (IJIR) Vol-3, Issue-2, 2017.
2. Ganesh Ram Sinha, “Design and Implementation of Image Enhancement Techniques in Frequency Domain”, September 2009
3. Gonzalez, R.C., Woods, “Digital Image Processing”, second ed. Prentice Hall, R.E., (2002)
4. Kanika Kapoor and Shaveta Arora, “Colour Image Enhancement based on Histogram Equalization”, International Journal (ECIJ) Volume 4, Number 3
5. Nicholas Sia Pik Kong, Haidi Ibrahim, and Seng Chun Hoo, “A Literature Review on Histogram Equalization and Its Variations for Digital Image Enhancement”, International Journal of Innovation, Management and Technology, Vol. 4, No. 4, August 2013.
6. Tengku Norsuhaila T Azmi, Nurul Azma Abdullah, “Image Size Variation Influence on Corrupted and Nonviewable BMP Image”, Information Security Interest Group (ISIG), University Tun Hussein Onn Malaysia
7. Dr T. Meyyappan, SM.Thamarai and N.M.Jeya Nachiaban, “Lossless digital image compression method for bitmap images”
8. Mamta Sharma, “Compression Using Huffman Coding”, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010

**Midsemester Report verified by**

**Project Guide HOD**

**(Name & Sign) (Dept. of Virtualization)**