UNIVERSITY OF CALCUTTA

DIGITAL IMAGE PROCESSING

Msc. Computer and Information Science

Semester- II

Roll-No :- 91/CIS/091007

Registration No :- 0000439 OF 2006-2007

Paper :- CISM-205(P)

CONTENT

ZOOMING AND SHRINKING DIGITAL IMAGES

Theory:-

Zooming and Shrinking of digital images is related to Image Sampling and Quantization.Zooming may be viewed as oversampling and shrinking may be viewed as undersampling.

ZOOMING:

Zooming requires two steps:

- The creation of new pixel locations
- The assignment of gray levels to those new locations.

Example:

Suppose that we have an image of size 500*500 pixels and we want to enlarge it 1.5 times to 750*750 pixels.

Conceptually, one of the easiest ways to visualize zooming is laying an imaginary 750*750 grid over the original image. Obviously, the spacing in the grid would be less than one pixel because

We are fitting it over a smaller image. In order to perform gray-level assignment for any point in the overlay, we look for the closest pixel in the original image and assign its gray level to the new pixel in the grid. When we are done with all points in the overlay grid, we simply expand it to the original specified size to obtain the zoomed image.

This method of gray-level assignment is called *nearest neighbor interpolation*.

SHRINKING:

Image shrinking is done in a similar manner as just described for zooming

Example

To shrink an image by one-half, we delete every other row and column.

We can use the zooming grid analogy to visualize the concept of shrinking by a non integer factor :

- we now *expand* the grid to fit over the original image
- do gray-level nearest neighbor or bilinear interpolation, and then shrink the grid back to its original specified size.
- To reduce possible aliasing effects, it is a good idea to blur an image slightly before shrinking it.

Algorithm:-

Step 1:- Take a sample JPEG Image file ‘ImageOriginal’ from the user.

Step 2:- Create a ‘ImageBuffer’ in the memory and place the ‘ImageOriginal’ in the image buffer.

Step 3:- Find the Height and Width of the image file ‘ImageOriginal’ and assign then to variables height and width respectively.

Step 4:- Take the ‘zoom_factor’ or ‘shrink_factor’ from the user as an Input.

Step5:- Perform nearest neighbor interpolation for each pixel position of the ‘ImageOriginal’ depending on the ‘zoom_factor/shrink_factor’.

ZOOMING:

Step 5.1:- Find out the RGB value for each pixel(i/zoom_factor,j/zoom_factor)

Step 5.2:- Assign the RGB value for each pixel(i, j) for the ‘New_Zoomed_Image’.

Step 5.3:- Create New_Buffer_Space for the ‘New_Zoomed_Image’ having dimensions

New_Height= height * zoom_factor

New_Width= width * zoom_factor

SHRINKING:

Step 5.1:- Find out the RGB value for each pixel(i*Shrink_factor, j*Shrink_factor)

Step 5.2:- Assign the RGB value for each pixel(i, j) for the ‘New_Shrinked_Image’.

Step 5.3:- Create New_Buffer_Space for the ‘New_Shrinked_Image’ having dimensions

New_Height= height/Shrink_factor

New_Width= width/Shrink_factor

Step 6:- Draw the New_ Zoomed/Shrinked Image.

Step 7:- End

Code:-

```
/*Assignment 1:-Zooming and Shrinking of a Digital Image*/
package ImageProcessingAssignments;

/*Import JAVA and JAVAX packages*/
import java.awt.*;
import java.io.*;
import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

/*Declaring the main class that extends JPanel and implements KeyListner*/
public class ZoomShrink extends JPanel implements KeyListener
{
	private static final long serialVersionUID = 1L;

		/*Declaring BufferImage variables which will store the original image and the
	modified image*/
	//Stores the original image
	BufferedImage imageOld;
	//Stores the new image
	BufferedImage imageNew;

	/*Declaring variables which will be used for creating the new image*/
	//Stores the width of the image
	public static int width = 0;
	//Stores the height of the image
	public static int height = 0;
	//Stores the pixel intensity of the image
	public static int val = 0;
	//Zoom factor in the x direction
	public static int zoomx = 1;
	//Zoom factor in the y direction
	public static int zoomy = 1;
	//Shrink factor in the x direction
	public static int shrinkx = 1;
	//Shrink factor in the y direction
	public static int shrinky = 1;
	//Flag to modify the factors for zooming
	public static int flags = 1;
	//Flag to modify the factors for shrinking
```

```
public static int flagz = 1;

/*Constructor to initialize the variables of the class*/
public ZoomShrink(BufferedImage imageOld)
{
	//Initialize the original image
	this.imageOld = imageOld;
	//Initialize the new image
	this.imageNew = imageOld;

	//Initialize the width of the image
	width=imageOld.getWidth();
	//Initialize the height of the image
	height=imageOld.getHeight();
 }

 /*Function to reload the image for repainting the screen*/
 public void LoadBuffer()
{
	//Create the buffer space for the new image to be drawn on the screen for zooming
	if(flagz == 1)
	{
		try
		{
			this.imageNew = new
			BufferedImage(imageOld.getWidth()*zoomx,imageOld.getHeight(
			)*zoomy,BufferedImage.TYPE_INT_RGB);
		}
		catch(Exception e)
		{
		}
	}

	/*Create the buffer space for the new image to be drawn on the screen for
	shrinking */
	if(flags == 1)
	{
		try
		{
			this.imageNew = new
			BufferedImage(imageOld.getWidth()/shrinkx,imageOld.getHeight(
			)/shrinky,BufferedImage.TYPE_INT_RGB);
		}
		catch(Exception e)
		{
		}
```

```
        }
}

/*Function which is called to paint the screen to display the images at regular intervals*/
protected void paintComponent(Graphics g)
{
        /*Passing the graphics display screen components to the superclass function to be
        displayed on the screen area*/
         super.paintComponent(g);

         /*Calling the function that creates the new image depending on the user's key
        input*/
        makeImage();

        //Calling the function that draws the images on the screen
        drawImages(g);
}

/*Function to create the new image*/
private void makeImage()
{
        //Declaring local variables
        int i = 0,j = 0;

        //For zooming the image
        if(flagz == 1)
        {
                for(i = 0;i < width*zoomx;i++)
                {
                        for(j = 0;j < height*zoomy;j++)
                        {
                                /*Reading the value at particular pixel position in the
                                original image which is close to the pixel position in the
                                new image*/
                                val = imageOld.getRGB(i/zoomx,j/zoomy);
                                //Mapping the value obtained to the new image
                                imageNew.setRGB(i,j,val);
                        }
                }
        }

        //For shrinking the image
        if(flags == 1)
        {
                for(i = 0;i < width/shrinkx;i++)
                {
```

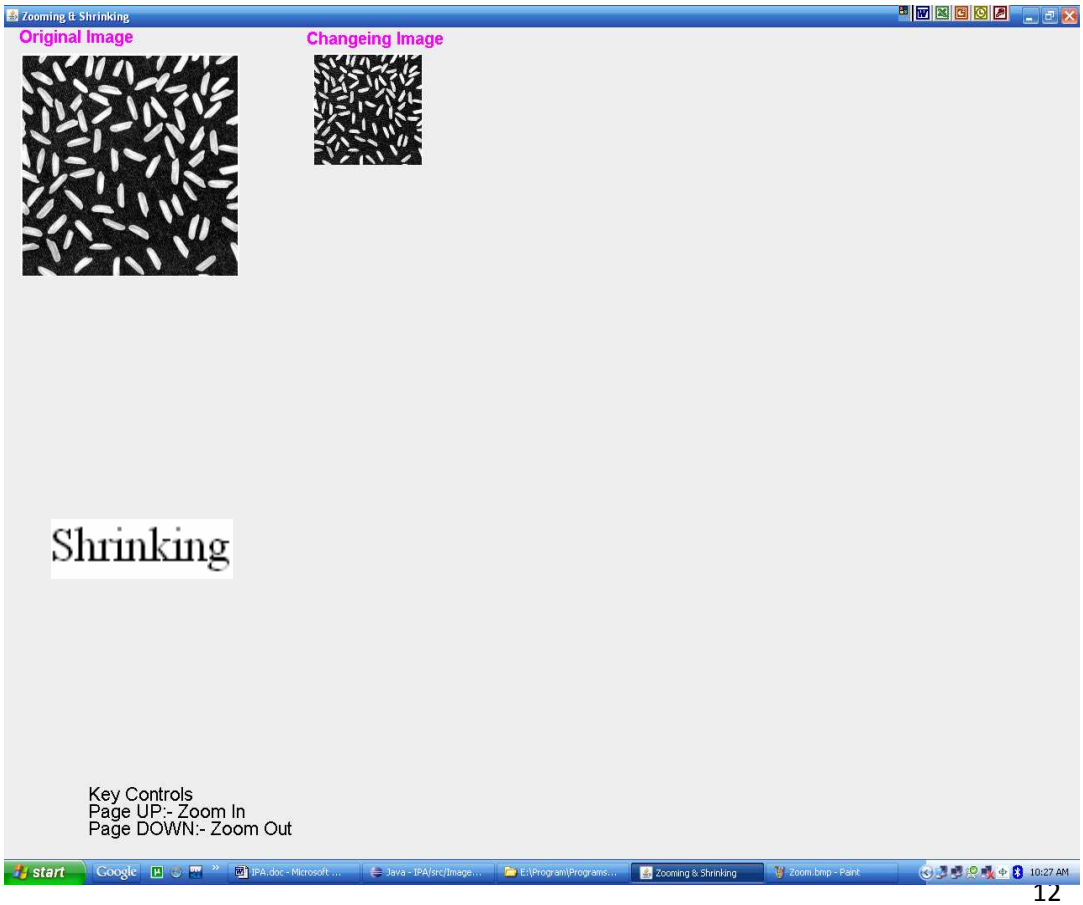
```
                        for(j = 0;j < height/shrinky;j++)
                        {
                                /*Reading the value at particular pixel position in the
                                original image which is close to the pixel position in the
                                new image*/
                                val = imageOld.getRGB(i*shrinkx,j*shrinky);
                                //Mapping the value obtained to the new image
                                imageNew.setRGB(i,j,val);
                        }
                }
        }
}

/*Function to draw the images on the screen*/
private void drawImages(Graphics g)
{
        /*Setting the color,font,font type and font size for the text to be displayed on the
        screen  */
        g.setColor(Color.magenta);
        g.setFont(new Font("Arial",Font.BOLD,20));

        //Displaying the original image at a particular screen location
        g.drawString("Original Image",18,18);
        g.drawImage(imageOld,20,30,this);

        //Displaying the new image at a particular screen location
        g.drawString("Changeing Image",100+width,20);
        g.drawImage(imageNew,108+width,30,this);

        /*Setting the color,font,font type and font size for the text to be displayed on the
        screen  */
        g.setColor(Color.black);
        g.setFont(new Font("Arial",Font.PLAIN,22));

        //Displaying the key controls to help the user
        g.drawString("Key Controls",100,900);
        g.drawString("Page UP:- Zoom In",100,920);
        g.drawString("Page DOWN:- Zoom Out",100,940);
}

/*Function that detects the key pressed*/
@Override
public void keyPressed(KeyEvent Key)
{
        /*If PAGE UP key is pressed then for zooming the factors are changed
        accordingly*/
```

```
if(Key.getKeyCode() == KeyEvent.VK_PAGE_UP)
{
	/*Checking if the original image has been shrinked then modify on the
	new image else on the old image*/
	if(flags == 1)
	{
		if(shrinkx > 1)
		{
			shrinkx-=1;
			shrinky-=1;
			flags = 1;
			flagz = 0;
		}
		else
		{
			zoomx+=1;
			zoomy+=1;
			flagz = 1;
			flags = 0;
		}
	}

	else
	{
		zoomx+=1;
		zoomy+=1;
		flagz = 1;
		flags = 0;
	}
	LoadBuffer();
	repaint();
}

/*If PAGE DOWN key is pressed then for shrinking the factors are changed
accordingly*/
if(Key.getKeyCode() == KeyEvent.VK_PAGE_DOWN)
{
	/*Checking if the original image has been zoomed then modify on the new
	image else on the old image*/
	if(flagz == 1)
	{
		if(zoomx > 1)
		{
			zoomx-=1;
			zoomy-=1;
			flagz = 1;
```

```
                        flags = 0;
                }
                else
                {
                        shrinkx+=1;
                        shrinky+=1;
                        flags = 1;
                        flagz = 0;
                }
        }
        else
        {
                shrinkx+=1;
                shrinky+=1;
                flags = 1;
                flagz = 0;
        }
        LoadBuffer();
        repaint();
    }
}

@Override
public void keyReleased(KeyEvent arg0)
{
}

@Override
public void keyTyped(KeyEvent Key)
{
}

/*Main function*/
public static void main(String[] args)
{
    /*Loading the image path on which the actios of zooming and shrinking will be
    done*/
    String path = "C:\\images\\02.png";
    //Creating a local buffer image in which the image will be loaded
    BufferedImage image = null;

    //Loading the image onto the buffer
    try
    {
        image = ImageIO.read(new File(path));
    }
```

```
            catch(IOException e)
            {
                  System.out.println("Cannot Read File at location :-"+path);
            }

            //Initializing the class image variable
            ZoomShrink obj = new ZoomShrink(image);
            //Creating the frame on which the image will be displayed
            JFrame frame = new JFrame("Zooming & Shrinking");
            //Setting the default close option
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //Adding the class object to display the image
            frame.add(obj);
            //Maximizing the frame by default
            frame.setExtendedState(Frame.MAXIMIZED_BOTH);
            //Adding the key listener to the frame
            frame.addKeyListener(obj);
            //Setting the visibility of the frame
            frame.setVisible(true);
      }
}
```

Input/Output:-



Discussion:-

In the “Zooming and Shrinking” assignment we are using buffer images to store the images(original image and the new modified image). The buffer space is initialized with height and width of the images and they can be modified by the user. After the original image has been buffered it is used to find the new image which is either a result of zoom or a result of shrinking. The buffered space for this assignment are initialized as “TYPE_INT_RGB”. In the function “makeImage()” the new images(zoomed or shrinked) are created depending on key press. The function “drawImages()” draws the images on the screen.

The function “keyPressed()” detects the key pressed on and modifies the zoom or shrink factors accordinly. Also the function checks if the image has been zoomed or shrinked previously then it works on the latest modified image.

The main function loads the image from the specified path, initializes the image to be shown on the screen and creates the frame on which the image will be drawn in the screen. It also sets the visibility of the frame. It also adds the key listener to the frame to respond to the key events.

HISTOGRAM EQUALIZATION OF DIGITAL IMAGES

Theory:-

The histogram equalization is an approach to enhance a given image. The approach is to design a transformation *T(.)* such that the gray values in the output is uniformly distributed

STEPS TO PERFORM HISTOGRAM EQUALIZATION:

- Let us assume for the moment that the input image to be enhanced has continuous gray values, with $r = 0$ representing black and $r = 1$ representing white.
- We need to design a gray value transformation $s = T(r)$, based on the histogram of the input image, which will enhance the image.
- we assume that:
  (1) $T(r)$ is a monotonically increasing function for $0 <= r <= 1$ (preserves order from black to white).
  (2) $T(r)$ maps [0,1] into [0,1] (preserves the range of allowed Gray values).
- Let us denote the inverse transformation by $r = T^{-1}(s)$. We assume that the inverse transformation also satisfies the above two conditions.
- We consider the gray values in the input image and output image as random variables in the interval [0, 1].
- Let $p_{in}(r)$ and $p_{out}(s)$ denote the probability density of the Gray values in the input and output images.
- If $p_{in}(r)$ and T(r) are known, and $r = T^{-1}(s)$ satisfies condition 1, we can write (result from probability theory):

$$p_{out}(s) = \left[p_{in}(r) \frac{dr}{ds} \right]_{r=T^{-1}(s)}$$

- One way to enhance the image is to design a transformation*T(.)* such that the gray values in the output is uniformly distributed in [0, 1], i.e. $p_{out}(s) = 1, \quad 0 <= s <= 1$
- This technique is called **histogram equalizatio**n.

Algorithm:-

Step 1:- Take a sample JPEG Image file ‘ImageOriginal’ from the user.

Step 2:- Create a ‘ImageBuffer’ in the memory and place the ‘ImageOriginal’ in the image buffer.

Step 3:- Find the Height and Width of the image file ‘ImageOriginal’ and assign then to variables height and width respectively.

Step 4:- Convert the ‘ImageOriginal’ to a GrayScale format image and store it in another ImageBuffer ‘ImageGrayScale’.

Step 5 :- Show Both ‘ImageOriginal’ and ‘ImageGrayScale’ .

Step 6:- Calculate r_k and n_k where r_k is the k^{th} intensity value and n_k is the number of pixels in the image ‘ImageGrayScale’ having the intensity value r_k and normalise histogram components by dividing it by the total number of pixels in the image i.e. height * width of ‘ImageGrayScale’. We store the values of r_k , n_k and $P(r_k) = n_k/($ height * width) in a table $T(r_k)$ in the memory.

Step 7:- Plot histogram of the ‘ImageGrayScale’ and display the histogram

Step 8:- Find the histogram equilization transformation for the interval [0, L-1] (where L is the maximum intensity if the image) as follows :

Step 8.1:- For k = 0 to L-1 do

Step 8.2:- For j = 0 to k do

Step 8.3:- $S(k) = S(k) + P(r_j)$ End do

Step 8.4:- $S(k) = (L-1)*S(k)$ End do

Where S(k) is a table in the memory for storing histogram equilization transformation data.

Step 9:- Create a ImageBuffer ‘ImageHistEqualized’ and obtain a processed output image by mapping each pixel in the input image with intensity r_k into a corresponding pixel with level s_k in the output image .

Step 10:- Show the histogram equalized image ‘ImageHistEqualized’‘.

Step 11:- Plot the histogram of ‘ImageHistEqualized’ and display it.

Step 12:- End.

Code:-

```
/*Assignment 2:- Histogram Equalization of a Digital Image*/
package ImageProcessingAssignments;

/*Import JAVA and JAVAX packages*/
import java.awt.*;
import java.io.*;
import javax.swing.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

/*Declaring the main class that extends JPanel*/
public class Histogram extends JPanel
{
   private static final long serialVersionUID = 1L;

        /*Declaring BufferImage variables which will store the original image and the modified
        images*/
        //Stores the original image
        BufferedImage imageOld;
        //Stores the gray style image
        BufferedImage imageNew;
        //Stores the new image equalized image
        BufferedImage imageEq;

        /*Declaring variables which will be used for creating the new image*/
        //Stores the width of the image
        public static int width = 0;
        //Stores the height of the image
        public static int height = 0;
        //Stores the pixel intensity of the image
        public static int val = 0;
        //Used to calculate the histogram only once
        public int flag = 1;
        /*Stores the number of pixels of the original image with the intensities equal to the index
        of the array */
        public int arr1[] = new int[256];
        /*Stores the number of pixels of the equalized image with the intensities equal to the
        index of the array*/
        public int arr2[] = new int[256];
        //Stores the original histogram values
        public float pr[] = new float[256];
        //Stores the equalized histogram values
```

```
public float ps[] = new float[256];

/*Constructor to initialize the variables of the class*/
public Histogram(BufferedImage imageOld)
{
	//Initialize the original image
	this.imageOld = imageOld;

	//Initialize the width of the image
	width=imageOld.getWidth();
	//Initialize the height of the image
	height=imageOld.getHeight();

	/*Create the buffer space for the new image to be drawn on the screen for in gray
	style*/
	this.imageNew = new
	BufferedImage(imageOld.getWidth(),imageOld.getHeight(),BufferedImage.TYP
	E_INT_RGB);
	//Create the buffer space for the equalized image to be drawn on the screen
	this.imageEq = new
	BufferedImage(imageOld.getWidth(),imageOld.getHeight(),BufferedImage.TYP
	E_INT_RGB);

	/*Initializes the array which stores the intensity values of the original gray style
	*/image
	for(int i = 0;i < 256;i++)
	{
		arr1[i] = 0;
	}

	/*Initializes the array which stores the intensity values of the equalized gray style
	image*/
	for(int i = 0;i < 256;i++)
	{
		arr2[i] = 0;
	}
}

/*Function which is called to paint the screen to display the images at regular intervals*/
protected void paintComponent(Graphics g)
{
	/*Passing the graphics display screen components to the superclass function to be
	displayed on the screen area*/
	super.paintComponent(g);
```

```
        /*Calling the function that creates the gray style image from the original image
        and then equalizes the image and creates the original and the equalized
        histogram*/
        makeHistogram();

        //Calling the function that draws the images on the screen
        drawImage(g);

        //Calling the function that draws the histograms on the screen
        drawHistogram(g);
}

/*Function to create the create the gray image and then calculate the histograms*/
private void makeHistogram()
{
        //Declaring local variables
        int i = 0,j = 0;
        int v = 0;
        int valu = 0;
        int re = 0,ge = 0,bl = 0;
        float s[] = new float[256];
        int org_img_arr[][] = new int[width][height];

        //Initializing array to be used to calculate the histogram
        for(i = 0;i < 256;i++)
        {
                s[i] = 0;
        }

        //Calculating the histograms only once and also creating the images only once
        if(flag == 1)
        {
                //Creating the gray scale image
                for(i = 0;i < width;i++)
                {
                        for(j = 0;j < height;j++)
                        {
                                //Extracting the pixel intensity at a particular point
                                val = imageOld.getRGB(i,j);
                                re = (int)(val >> 16) & 0xFF;
                                ge = (int)(val >> 8) & 0xFF;
                                bl = (int)(val >>0) & 0xFF;
                                v = (re+ge+bl)/3;
                                valu = (v<<16)|(v<<8)|(v);
                                //Creating the image
                                imageNew.setRGB(i,j,valu);
```

```
        }
}

//Creating the intensity map and storing the pixel intensities
for(i = 0;i < width;i++)
{
        for(j = 0;j < height;j++)
        {
                //Extracting the pixel intensity at a particular point
                val = imageNew.getRGB(i,j);
                re = (val >> 16) & 0xFF;
                ge = (val >> 8) & 0xFF;
                bl = (val >> 0) & 0xFF;
                //Calculating the n for each k of the original image
                arr1[(re+ge+bl)/3]++;
                //Creates the intensity map array
                org_img_arr[i][j] = (re+ge+bl)/3;
        }
}

flag = 0;

//Calculating the original histogram of the original image
for(i = 0;i < 256;i++)
{
        try
        {
                pr[i] = (float)arr1[i]*255/(float)(width*height);
        }
        catch(Exception e)
        {
        }
}

//Calculating temporary values for the equalized image
for (i = 0;i < 256;i++)
{
        for (j = 0;j <= i;j++)
        {
                s[i] = s[i] + pr[j];
        }
}

//Creating the equalized image
for(i = 0;i < width;i++)
{
```

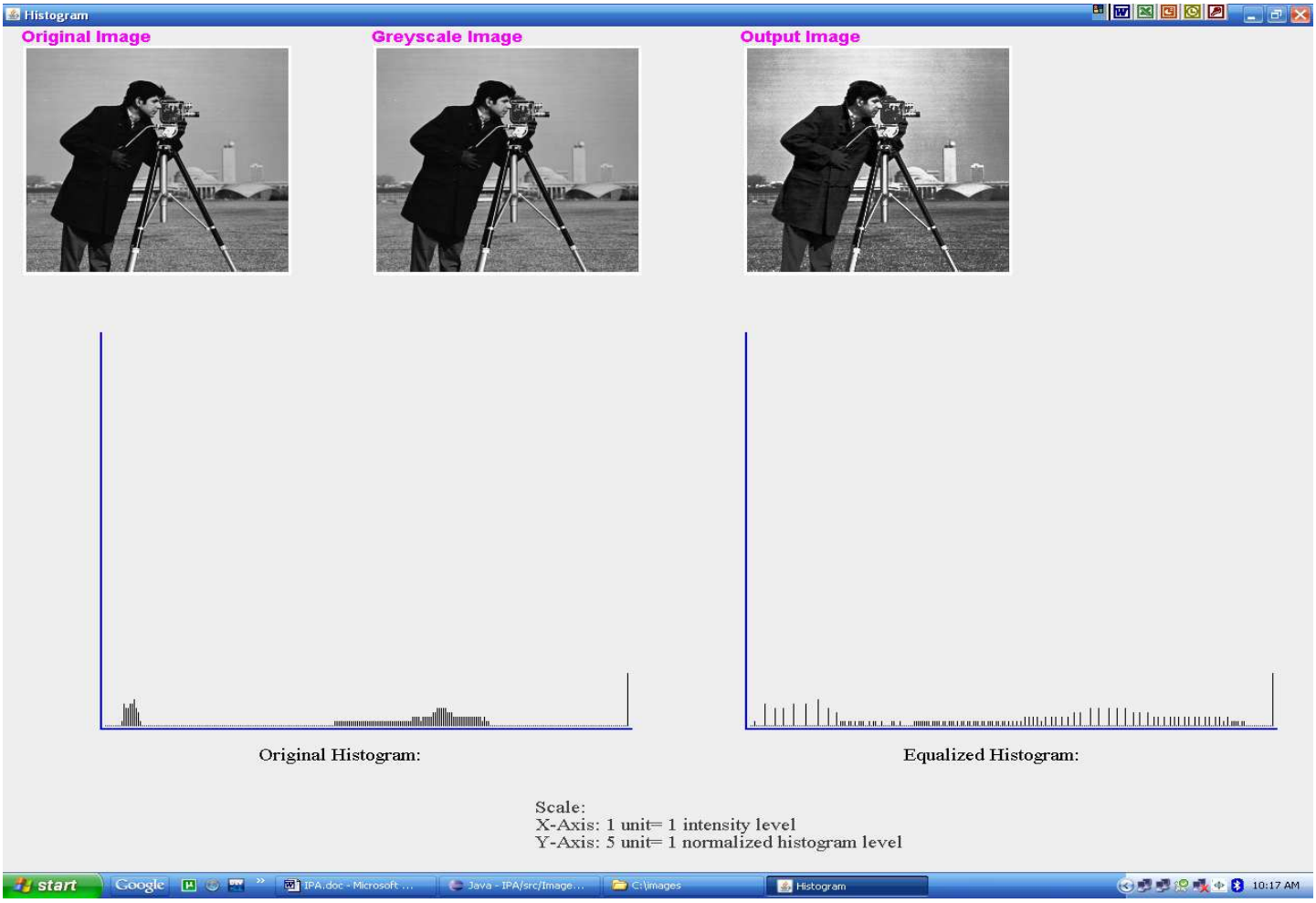
```
            for(j = 0;j < height;j++)
            {
                  //Extracting the equalized pixel of the equalized image
                  v = Math.round(s[org_img_arr[i][j]]);
                  re = (v<<16);
                  ge = v<<8;
                  bl = v;
                  valu = re|ge|bl;

                  try
                  {
                        imageEq.setRGB(i,j,valu);
                  }
                  catch(Exception e)
                  {
                  }
            }
      }

      //Storing the pixel intensities
      for(i = 0;i < width;i++)
      {
            for(j = 0;j < height;j++)
            {
                  //Extracting the pixel intensity at a particular point
                  val = imageEq.getRGB(i,j);
                  re = (val >> 16) & 0xFF;
                  ge = (val >> 8) & 0xFF;
                  bl = (val >> 0) & 0xFF;
                  //Calculating the n for each k of the equalized image
                  arr2[(re+ge+bl)/3]++;
            }
      }

      //Calculating the equalized histogram of the equalized image
      for(i = 0;i < 256;i++)
      {
            try
            {
                  ps[i] = (float)arr2[i]*255/(float)(width*height);
            }
            catch(Exception e)
            {
            }
      }
}
```

```
}

/*Function to draw the images on the screen*/
private void drawImage(Graphics g)
{
	/*Setting the color,font,font type and font size for the text to be displayed on the
	screen*/
	g.setColor(Color.magenta);
	g.setFont(new Font("Arial",Font.BOLD, 18));

	//Displaying the original image at a particular screen location
	g.drawString("Original Image",18,18);
	g.drawImage(imageOld,20,20,this);

	//Displaying the new gray style image at a particular screen location
	g.drawString("Greyscale Image",98+width, 18);
	g.drawImage(imageNew,100+width,20,this);

	//Displaying the equalized image at a particular screen location
	g.drawString("Output Image",196+2*width, 18);
	g.drawImage(imageEq,200+2*width,20,this);
}

/*Function to plot the histograms on the screen*/
private void drawHistogram(Graphics g)
{
	//Declaring local variables
	int i = 0,j = 0;

	/*Setting the color,font,font type and font size for the text to be displayed on the
	screen*/
	g.setFont(new Font("Times New Roman",Font.PLAIN,20));

	/*Setting the color and drawing the x and y axis for the original histogram on the
	screen*/
	g.setColor(Color.BLUE);
	g.fillRect(95,803,520,2);
	g.fillRect(95,350,2,454);

	//Labeling
	g.setColor(Color.BLACK);
	g.drawString("Original Histogram:",250,840);

	//Drawing the original histogram
	for(i = 0,j = 100;i <= 255;i++,j+=2)
	{
```

```
                g.drawLine(j,800-(int)pr[i]*5,j,800);
        }

        /*Setting the color and drawing the x and y axis for the equalized histogram on
        the screen*/
        g.setColor(Color.BLUE);
        g.fillRect(725,803,520,2);
        g.fillRect(725,350,2,454);

        //Labeling
        g.setColor(Color.BLACK);
        g.drawString("Equalized Histogram:",880,840);

        //Drawing the equalized histogram
        for(i = 0,j = 730;i <= 255;i++,j+=2)
        {
                g.drawLine(j,800-(int)ps[i]*5,j,800);
        }

        //Displaying the scale used
        g.setColor(Color.DARK_GRAY);
        g.drawString("Scale:",520,900);
        g.drawString("X-Axis: 1 unit= 1 intensity level",520,920);
        g.drawString("Y-Axis: 5 unit= 1 normalized histogram level",520,940);
}

/*Main function*/
public static void main(String[] args)
{
        /*Loading the image path on which the actios of zooming and shrinking will be
        done*/
        String path = "C:\\images\\01.png";
        //Creating a local buffer image in which the image will be loaded
        BufferedImage imageOld = null;

        //Loading the image onto the buffer
        try
        {
                imageOld = ImageIO.read(new File(path));
        }
        catch(IOException e)
        {
                System.out.println("Cannot Read File at location :-"+path);
        }

        //Initializing the class image variable
```

```
                Histogram obj = new Histogram(imageOld);
                //Creating the frame on which the image will be displayed
                JFrame frame = new JFrame("Histogram");
                //Setting the default close option
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                //Adding the class object to display the image
                frame.add(obj);
                //Maximizing the frame by default
                frame.setExtendedState(Frame.MAXIMIZED_BOTH);
                //Setting the visibility of the frame
                frame.setVisible(true);
        }
}
```

Input/Output:-



Output of the Histogram Equilization Program :

- Given as input is a color Low Contrast Image .
- The Image is converted to Gray Scale format.
- The histogram of the original image matches the charactaristics of a Low Contrast image as a narrow histogram ia produces and the histogram components are in the middle of the intensity scale.
- Histogram Equalized image is on the far right side which shows a significant improvement in contrast.
- The Equalized Histogram shows that the pixels now occoupy the entire range of the intensity level and are evenly distributed

Discussion:-

In the “Histogram Equalization” assignment we are using buffer images to store the images(original image, gray scale image and the equalized image). The buffer space is initialized with height and width of the images and they can be modified by the user. After the original image has been buffered it is used to find the gray scale image which whose buffered space is denoted as “TYPE_INT_RGB”. Also, the equalized image buffer space is of the type “TYPE_INT_RGB” .Within the ‘paintComponent()’ function the histogram in created and stored in a 1D array. The function draws the histograms(original and equalized) on the screen. The function is also used to draw the image on the screen.

The main function loads the image from the specified path, initializes the image to be shown on the screen and creates the frame on which the image will be drawn in the screen. It also sets the visibility of the frame.