Lab 4

Use two images for each operation to do the following operations.

1. Sharpen images with Laplacian and Sobel operators respectively through masking process
2. Gamma correction using gamma value 0.1, 0.4, 0.7, 1 and compute the variances of the resulted images
3. Write histogram enhancement function to perform global and local image enhancement

Algorithm:

1. Laplacian operator

Here I use the following operator. $\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$

Input: an unsigned char array that stores the source image data

Output: an unsigned char array that stores the output image data

```
for (i = 1; i < inimage.Width - 1; i++) {
    for (j = 1; j < inimage.Height - 1; j++) {
        sum = -4 * inimage[i][j]
            + inimage [i – 1][j]
            + inimage [i + 1][j]
            + inimage [i][j - 1]
            + inimage [i][j + 1];
        sum = (sum < 0) ? 0: ((sum > 255) ? 255 : sum);//If the calculate results
                                                        //are smaller than 0 or larger
                                                        //than 255, just set them as 0 or
                                                        //255 respectively
        outimage [i] [j] = sum;
    }
}
return outimage
```

2. Sobel operator

Here I use vertical and horizontal operators $\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$, $\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$

and then calculate $\sqrt{x^2 + y^2}$ as the final result.

Input: an unsigned char array that stores the source image data

Output: an unsigned char array that stores the output image data

```
for (i = 1; i < inimage.Width - 1; i++) {
    for (j = 1; j < inimage.Height - 1; j++) {
        x = (-1) * inimage [i - 1] [j - 1]
            + inimage [i - 1][ j + 1)]
            - 2 * inimage [i][j - 1]
            + 2 * inimage [i][j + 1]
            - 1 * inimage[i – 1][j + 1]
            + inimage[i + 1] [j + 1];
        y = (-1) * inimage[i – 1][j - 1]
            - 2 * inimage[i – 1][ j]
```

```
                    - inimage[i – 1][j + 1]
                    + inimage[i + 1][j - 1]
                    + 2 * inimage[i + 1][ j]
                    + inimage[i + 1][j + 1];
                temp = round(sqrt(x * x + y * y));
                temp =((temp > 255) ? 255 : temp);//If the calculate results are larger than
                                                   255, just set them 255
                outimage [i] [j] = temp;
            }
        }
        return outimage
```

3. Gamma correction

   The function is $s(x,y) = 255 \times (\frac{f(x,y)}{255})^{\gamma}$

   Input: an unsigned char array that stores the source image data, the value of gamma
   Output: an unsigned char array that stores the output image data

```
        for (i = 0; i < inimage.Width; i++) {
            for (j = 0; j < inimage.Height; j++) {
                outimage[i] [j] = round(255 * pow(inimage[i][j] / 255, gamma));
             sum += outimage[i][j];//One factor of calculating the variance of the picture
            }
        }
             avg =sum / (image->Width * image->Height); //One factor of calculating
                                                         // the variance of the picture
        for (i = 0; i < inimage.Width; i++) {
            for (j = 0; j < inimage.Height; j++) {
                temp=temp+pow( (outimage[i][ j] - avg),2);
            }
        }
```

   var = temp / (outImage.Width * outImage.Height); $//var = \frac{\Sigma(f(x,y)-\mu)^2}{MN}$

   return outimage

4. Histogram Enhancement
   1) Global

```
 //Calculate the accumulating number of each pixel value
          for (i = 0; i <inimage->Width; i++) {
              for (j = 0; j < inimage.Height; j++) {
                  hist[inimage[i] [j]] += 1;
              }
          }
          //Transfer to the probability
          for (i = 0; i < 256; i++) {// The grey image has a larger value of 255
              prob[i]= hist[i]/(inimage.Height*inimage.Width);
          }
          for (i = 0; i < 256; i++) {
              accProb += prob[i];
              s[i] = 255 * accProb;
```

```
        }
        //Assign the result to the outimage
        for (i = 0; i < image.Width; i++) {
            for (j = 0; j < image.Height; j++) {
                outimage[i][j] = s[inimage[i][j]];
            }
        }
        return outImage;
```

2) Local

The algorithm of local histogram enhancement is similar with the global one. Here I use 4×4 pixels as one local unit.

```
    for (i = 0; i < image->Height; i =i+ 4) {
        for (j = 0; j < image->Width; j =j + 4) {
            //Reset every temporary variable as zero
            accProb = 0;
            memset(hist, 0, sizeof(hist));
            memset(prob, 0, sizeof(prob));
            memset(s, 0, sizeof(s));
            //Calculate the accumulating number of each pixel value in every 4×4 units
            for (p = 0; p < 4; p++) {
                for (q = 0; q < 4; q++) {
                    hist[inimage[i+p][j+q]] += 1;
                }
            }
            for (r = 0; r < 256; r++) {
                prob[r] = hist[r] /16;
            }
            for (r = 0; r < 256; r++) {
                accProb += prob[r];
                s[r] = 255 * accProb;
            }
            for (p = 0; p < 4; p++) {
                for (q = 0; q < 4; q++) {
                    outimage[i+p][j+q] = s[inimage[i + p] [j + q]];
                }
            }
        }
    }
    return outimage
```

Results (compare the results with the original image):

1. Sharpen operator

Source

Laplacian

Sobel

Source

Laplacian

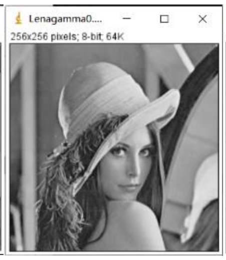
Sobel

2. Gamma correction

Source

$\gamma = 0.1$

$\gamma = 0.4$



$\gamma = 0.7$

$\gamma = 1$

Source

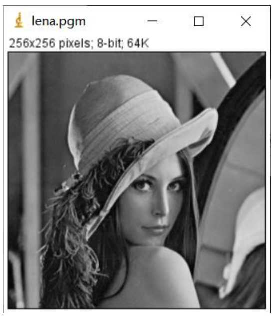
$\gamma = 0.1$

$\gamma = 0.4$

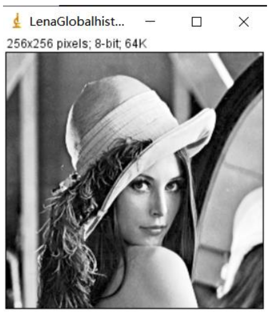
$\gamma = 0.7$

$\gamma = 1$

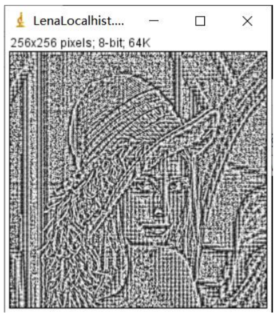
3. Histogram Enhancement



Source



Global



Local

Source

Global

Local

Discussion:

1. When conducting the sharpen operation, sobel operator seems more efficient than the Laplacian operator comparing with their performance of boundary extraction. Also, when the original image has a more clear edge feature, which means that there is a not small difference between the edge and the background, the performance of sharpen operator is better.
2. When conducting the gamma correction, the bigger gamma value, the clearer picture. When gamma equal to 1, the output image is the same as the input image.
3. After conducting histogram enhancement, the original image histogram can be transformed into a uniform distribution (equilibrium) form, so as to increase the dynamic range of gray value difference between pixels, so as to enhance the overall contrast effect of the image. The local histogram enhancement has better local details but possible blocky artifact.

Codes:

1. Laplacian Operator

```
Image* Laplacian(Image* image) {
    Image* outImage;
    int i, j, p, q, temp,count;
    unsigned char* tempin, * tempout;

    outImage = CreateNewImage(image, "#Laplacian", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    for (i = 1; i < image->Width - 1; i++) {
        for (j = 1; j < image->Height - 1; j++) {
            temp = -4 * tempin[i * image->Width + j]
                + tempin[(i - 1) * image->Width + j]
                + tempin[(i + 1) * image->Width + j]
                + tempin[i * image->Width + j - 1]
                + tempin[i * image->Width + j + 1];
            temp = (temp < 0) ? 0: ((temp > 255) ? 255 : temp);
            tempout[i * outImage->Width + j] = temp;
        }
    }
    return outImage;
}
```

2. Sobel Operator

```
Image* Sobel(Image* image) {
    Image* outImage;
    int i, j, p, q, temp, count, x, y;
    unsigned char* tempin, * tempout;

    outImage = CreateNewImage(image, "#Sobel", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    for (i = 1; i < image->Width - 1; i++) {
        for (j = 1; j < image->Height - 1; j++) {
            x = (-1) * tempin[(i - 1) * image->Width + (j - 1)]
                + tempin[(i - 1) * image->Width + (j + 1)]
                - 2 * tempin[(i)*image->Width + (j - 1)]
                + 2 * tempin[(i)*image->Width + (j + 1)]
                - 1 * tempin[(i - 1) * image->Width + (j + 1)]
                + tempin[(i + 1) * image->Width + (j + 1)];
            y = (-1) * tempin[(i - 1) * image->Width + (j - 1)]
                - 2 * tempin[(i - 1) * image->Width + j]
                - tempin[(i - 1) * image->Width + (j + 1)]
                + tempin[(i + 1) * image->Width + (j - 1)]
                + 2 * tempin[(i + 1) * image->Width + j]
                + tempin[(i + 1) * image->Width + (j + 1)];
            temp = round(sqrt(x * x + y * y));
            temp = (temp > 255) ? 255 : temp;
            tempout[i * outImage->Width + j] = temp;
        }
    }
    return outImage;
}
```

3. Gamma Correction

```
Image* GammaCorrection(Image* image, float gamma) {
    Image* outImage;
    int i, j, p, q, temp = 0, sum = 0;
    double var,avg;
    unsigned char* tempin, * tempout;

    outImage = CreateNewImage(image, "#GammaCorrection", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    for (i = 0; i < image->Width; i++) {
        for (j = 0; j < image->Height; j++) {
            tempout[i * outImage->Width + j] = round((float)255 * pow((float)tempin[i * image->Width + j] / 255, gamma));
            sum += tempout[i * outImage->Width + j];
        }
    }
    avg = (float)sum / (image->Width * image->Height);
    for (i = 0; i < image->Width; i++) {
        for (j = 0; j < image->Height; j++) {
            temp=temp+pow((float)(tempout[i * outImage->Width + j] - avg),2);
        }
    }
    var = temp / (outImage->Width * outImage->Height);
    printf("%f ", var);
    return outImage;

}
```

4. Global histogram enhancement

```c
Image* GlobalHistogramEnhancement(Image* image) {
    Image* outImage;
    int i, j;
    float accProb=0;
    unsigned char* tempin, * tempout;
    int hist[257] = { 0 };
    float prob[257] = { 0 };
    int s[257] = { 0 };
    outImage = CreateNewImage(image, "#HistogramEnhancement", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    //Calculate the accumulating number of each pixel value
    for (i = 0; i < image->Width; i++) {
        for (j = 0; j < image->Height; j++) {
            hist[tempin[i * image->Width + j]] += 1;
        }
    }
    //Transfer to the probability
    for (i = 0; i < 256; i++) {
        prob[i]=(float)hist[i]/(image->Height*image->Width);
    }
    for (i = 0; i < 256; i++) {
        accProb += prob[i];
        s[i] = 255 * accProb;
    }

    for (i = 0; i < image->Width; i++) {
        for (j = 0; j < image->Height; j++) {
            tempout[i * outImage->Width + j] = s[tempin[i * image->Width + j]];
        }
    }
    return outImage;
}
```

5. Local histogram enhancement

```c
Image* LocalHistogramEnhancement(Image* image) {
    Image* outImage;
    int i, j, p, q,r, temp = 0,sum = 0,count=0;
    float accProb;
    unsigned char* tempin, * tempout;
    int hist[257] = { 0 };
    float prob[257] = { 0 };
    int s[257] = { 0 };
    outImage = CreateNewImage(image, "#HistogramEnhancement", image->Width, image->Height);
    tempout = outImage->data;
    tempin = image->data;

    for (i = 0; i < image->Height; i =i+ 4) {
        for (j = 0; j < image->Width; j =j + 4) {

            accProb = 0;
            memset(hist, 0, sizeof(hist));
            memset(prob, 0, sizeof(prob));
            memset(s, 0, sizeof(s));
            //Calculate the accumulating number of each pixel value
            for (p = 0; p < 4; p++) {
                for (q = 0; q < 4; q++) {
                    hist[tempin[(i+p) * image->Width + j+q]] += 1;
                }
            }
            for (r = 0; r < 256; r++) {
                prob[r] = (float)hist[r] /16;
            }
            for (r = 0; r < 256; r++) {
                accProb += prob[r];
                s[r] = 255 * accProb;
            }
            for (p = 0; p < 4; p++) {
                for (q = 0; q < 4; q++) {
                    tempout[(i+p) * outImage->Width + j+q] = s[tempin[(i + p) * image->Width + j + q]];
                }
            }
        }
    }
    return outImage;
}
```