Class Lab 4

Use two images for each operation to do the following operations and write down their advantages and disadvantages and explain your results:

1. Laplacian operation (lena, bridge):

Algorithm:

It can be shown (Rosenfeld and Kak [1982]) that the simplest isotropic derivative operator (kernel) is the *Laplacian*, which, for a function (image) $f(x, y)$ of two variables, is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3-50}$$

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (3-49), keeping in mind that we now have a second variable. In the x-direction, we have

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{3-51}$$

and, similarly, in the y-direction, we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \tag{3-52}$$

It follows from the preceding three equations that the discrete Laplacian of two variables is

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{3-53}$$

In this case, we use the filter:

$$\begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Then we can get the final image:

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)]$$

Where $c = 0.5$.

Results (including pictures):

Result of processing "Lena.pgm":

Source Image:

Result after Laplacian operation:

Result of processing "Bridge.pgm":

Source Image:

Result after Laplacian operation

Discussion:

The detail in this image is unmistakably clearer and sharper than in the original image. Adding the Laplacian to the original image restored the overall intensity variations in the image. Adding the Laplacian increased the contrast at the locations of intensity discontinuities. The net result is an image in which small details were enhanced and the background tonality was reasonably preserved.

Codes:

```
Image *LaplacianImage(Image *image) {
    unsigned char *tempin, *tempout;
    Image *outimage;
    outimage = CreateNewImage(image, image->Height, image->Width, "#Laplacian Sharpen Filter");
    tempin = image->data;
    tempout = outimage->data;

    int matrixWidth = image->Width + 2;
    int matrixHeight = image->Height + 2;
    int filter[3][3] = {
        {0, -1, 0},
        {-1, 4, -1},
        {0, -1, 0}};

    int matrix[matrixWidth][matrixHeight];
    memset(matrix, 0, sizeof(matrix));
    for (int i = 1; i < matrixWidth - 1; i++) {
        for (int j = 1; j < matrixHeight - 1; j++, tempin++) {
            matrix[i][j] = *tempin;
        }
    }

    for (int i = 1; i < matrixWidth - 1; i++) {
        for (int j = 1; j < matrixHeight - 1; j++, tempout++) {
            int res = 0;
            for (int m = 0; m < 3; m++) {
                for (int n = 0; n < 3; n++) {
                    res += matrix[i - 1 + m][j - 1 + n] * filter[m][n];
                }
            }
            res = matrix[i][j] + (float)res * 0.5;
            if (res < 0) {
                res = 0;
            } else if (res > 255) {
                res = 255;
            }
            *tempout = res;
        }
    }

    return (outimage);
}
```

2. Sobel operation (lena, bridge):

Algorithm:

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Sobel vertical and horizontal operators

Results (including pictures):

Result of processing “Lena.pgm”:

Source Image:

Result after Sobel operation:

Result of processing “Bridge.pgm”:

Source Image:

Result after Sobel operation

Discussion:

The advantage of Sobel operator is that the method is simple, the processing speed is fast, and the edge obtained is smooth and continuous. The disadvantage is that the edges are thicker.

Codes:

```
Image *SobelImage(Image *image) {
    unsigned char *tempin, *tempout;
    Image *outimage;
    outimage = CreateNewImage(image, image->Height, image->Width, "#Sobel Sharpen Filter");
    tempin = image->data;
    tempout = outimage->data;

    int matrixWidth = image->Width + 2;
    int matrixHeight = image->Height + 2;
    int filter_x[3][3] = {
        {1, 0, -1},
        {2, 0, -2},
        {1, 0, -1}};
    int filter_y[3][3] = {
        {-1, -2, -1},
        {0, 0, 0},
        {1, 2, 1}};

    int matrix[matrixWidth][matrixHeight];
    memset(matrix, 0, sizeof(matrix));
    for (int i = 1; i < matrixWidth - 1; i++) {
        for (int j = 1; j < matrixHeight - 1; j++, tempin++) {
            matrix[i][j] = *tempin;
        }
    }

    for (int i = 1; i < matrixWidth - 1; i++) {
        for (int j = 1; j < matrixHeight - 1; j++, tempout++) {
            int res_x = 0;
            int res_y = 0;
            for (int m = 0; m < 3; m++) {
                for (int n = 0; n < 3; n++) {
                    res_x += matrix[i - 1 + m][j - 1 + n] * filter_x[m][n];
                    res_y += matrix[i - 1 + m][j - 1 + n] * filter_y[m][n];
                }
            }
            int res = sqrt(pow(res_x, 2) + pow(res_y, 2));
            if (res < 0) {
                res = 0;
            } else if (res > 255) {
                res = 255;
            }
            *tempout = res;
        }
    }

    return (outimage);
}
```

3. Gamma correction (lena, bridge):

Algorithm:

$$s = cr^{\gamma}$$

In this case, we use $c = L - 1$ where L is the domain of pixel. And $r = \frac{p}{L-1}$ where p is the value of pixel.

$$\gamma = 0.1, 0.4, 0.7, 1$$

Results (including pictures):

Result of processing "Lena.pgm":

Source Image:

Result after Gamma correction($\gamma = 0.1$):

Result of processing "Bridge.pgm":

Source Image:

Result after Gamma correction

Source Image:

Result after Gamma correction($\gamma = 0.4$):

Result of processing “Bridge.pgm”:

Source Image:

Result after Gamma correction

Source Image:

Result after Gamma correction($\gamma = 0.7$):

Result of processing “Bridge.pgm”:

Source Image:

Result after Gamma correction

Source Image:

Result after Gamma correction($\gamma = 1$):

Result of processing “Bridge.pgm”:

Source Image:

Result after Gamma correction

```
Loading ..\images\lena.pgm .....Image Type PGM

Saving Image ..\output\lena_Laplacian.pgm

Saving Image ..\output\lena_Sobel.pgm

The variance of Gamma Correction(0.10) is 232.99.
Saving Image ..\output\lena_GammaCorrect_0.1.pgm

The variance of Gamma Correction(0.40) is 1676.72.
Saving Image ..\output\lena_GammaCorrect_0.4.pgm

The variance of Gamma Correction(0.70) is 2516.13.
Saving Image ..\output\lena_GammaCorrect_0.7.pgm

The variance of Gamma Correction(1.00) is 2734.38.
Saving Image ..\output\lena_GammaCorrect_1.pgm

Saving Image ..\output\lena_HistogramEnhancementGlobal.pgm

Saving Image ..\output\lena_HistogramEnhancementLocal.pgm

Loading ..\images\bridge.pgm .....Image Type PGM

Saving Image ..\output\bridge_Laplacian.pgm

Saving Image ..\output\bridge_Sobel.pgm

The variance of Gamma Correction(0.10) is 391.75.
Saving Image ..\output\bridge_GammaCorrect_0.1.pgm

The variance of Gamma Correction(0.40) is 1378.47.
Saving Image ..\output\bridge_GammaCorrect_0.4.pgm

The variance of Gamma Correction(0.70) is 2282.79.
Saving Image ..\output\bridge_GammaCorrect_0.7.pgm

The variance of Gamma Correction(1.00) is 2819.52.
Saving Image ..\output\bridge_GammaCorrect_1.pgm

Saving Image ..\output\bridge_HistogramEnhancementGlobal.pgm

Saving Image ..\output\bridge_HistogramEnhancementLocal.pgm
```

Discussion:

Observe that as gamma decreased from 1 to 0.4, more detail became visible. A further decrease of gamma to 0.1 enhanced a little more detail in the background, but began to reduce contrast to the point where the image started to have a very slight washed-out appearance, especially in the background. The best enhancement in terms of contrast and discernible detail was obtained with g = 0.4. A value of g = 0.1 is an approximate limit below which contrast in this particular image would be reduced to an unacceptable level.

Codes:

```
Image *GammaCorrectionImage(Image *image, float gamma) {
    unsigned char *tempin, *tempout;
    Image *outimage;
    outimage = CreateNewImage(image, image->Height, image->Width, "#Gamma Correction");
    tempin = image->data;
    tempout = outimage->data;

    int matrixWidth = image->Width;
    int matrixHeight = image->Height;

    int matrix[matrixWidth][matrixHeight];
    memset(matrix, 0, sizeof(matrix));
    for (int i = 0; i < matrixWidth; i++) {
        for (int j = 0; j < matrixHeight; j++, tempin++) {
            matrix[i][j] = *tempin;
        }
    }
    for (int i = 0; i < matrixWidth; i++) {
        for (int j = 0; j < matrixHeight; j++, tempout++) {
            int res = (matrixWidth - 1) * pow((float)matrix[i][j] / matrixWidth, gamma);
            *tempout = res;
        }
    }

    // compute the variances of the resulted images

    float sum = 0.0;
    tempout = outimage->data;
    for (int i = 0; i < matrixWidth * matrixHeight; i++, tempout++) {
        sum += *tempout;
    }
    float mean = sum / (matrixWidth * matrixHeight);
    sum = 0.0;
    tempout = outimage->data;
    for (int i = 0; i < matrixWidth * matrixHeight; i++, tempout++) {
        sum += pow(*tempout - mean, 2);
    }
    float var = sum / (matrixWidth * matrixHeight);

    printf("The variance of Gamma Correction(%0.2f) is %0.2f.\n", gamma, var);
    return (outimage);
}
```

4. Histogram enhancement Global (lena, bridge):

Algorithm:

1. Count the number of values for each pixel;
2. Calculate the PDF for each gray value;
3. By using function $s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), k = 0,1,2,\dots,L-1$. Get the value of histogram equalization transformation function s_k;
4. Round the value of s_k, and then you could get the final result.

Results (including pictures):

Result of processing "Lena.pgm":

Source Image:

Result after Histogram enhancement Global:

Result of processing "Bridge.pgm":

Source Image:

Result after Histogram enhancement Global

Discussion:

By changing the histogram of the image to change the gray of each pixel in the image, it is mainly used to enhance the contrast of the image with small dynamic range. The original image is not

clear because its gray distribution may be concentrated in a narrow range. Image contrast will be improved and details will be sharper

Codes:

```
Image *HistogramEnhancementGlobalImage(Image *image) {
    unsigned char *tempin, *tempout;
    Image *outimage;
    outimage = CreateNewImage(image, image->Height, image->Width, "#Histogram Enhancement Global");
    tempin = image->data;
    tempout = outimage->data;
    int matrixWidth = image->Width;
    int matrixHeight = image->Height;

    int matrix[matrixWidth][matrixHeight];
    memset(matrix, 0, sizeof(matrix));
    for (int i = 0; i < matrixWidth; i++) {
        for (int j = 0; j < matrixHeight; j++, tempin++) {
            matrix[i][j] = *tempin;
        }
    }

    float array[256];
    memset(array, 0, sizeof(array));

    for (int i = 0; i < matrixWidth; i++) {
        for (int j = 0; j < matrixHeight; j++) {
            array[matrix[i][j]]++;
        }
    }

    for (int i = 0; i < 256; i++) {
        array[i] /= matrixWidth * matrixHeight;
    }

    int res[256];
    memset(res, 0, sizeof(res));

    for (int i = 0; i < 256; i++) {
        float sum = 0.0;
        for (int j = 0; j <= i; j++) {
            sum += array[j];
        }
        res[i] = round(sum * 255);
    }

    for (int i = 0; i < matrixWidth; i++) {
        for (int j = 0; j < matrixHeight; j++, tempout++) {
            *tempout = res[matrix[i][j]];
        }
    }

    return (outimage);
}
```

5. Histogram enhancement local (lena, bridge):

Algorithm:

Determine the kernel and size of the local processing area.

Count the number of values for each pixel;

Calculate the PDF for each gray value;

By using function $s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), k = 0,1,2,\dots,L-1$. Get the value of histogram equalization transformation function s_k;

Round the value of s_k, and then you could get the final result.

Results (including pictures):

Result of processing "Lena.pgm":

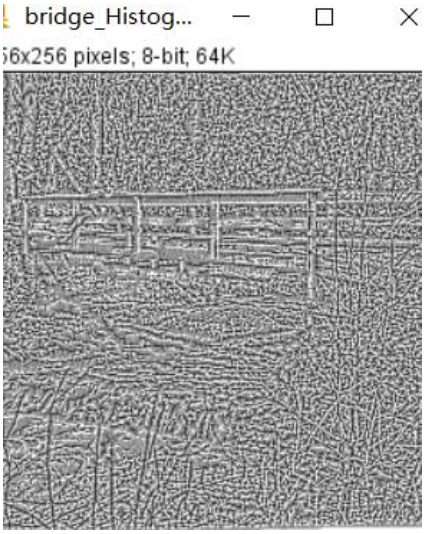
Source Image:

Result after Histogram enhancement Global:

Result of processing "Bridge.pgm":

Source Image:

Result after Histogram enhancement Global



Discussion:

The local enhancement method enhances the edge details of the image. The influence of the number of pixels in the cell domain on the global transformation is not ignored.

Codes:

```
    int matrix[matrixWidth][matrixHeight];
    memset(matrix, 0, sizeof(matrix));
    for (int i = zero1 / 2; i < matrixWidth - zero1 / 2; i++) {
        for (int j = zero2 / 2; j < matrixHeight - zero2 / 2; j++, tempin++) {
            matrix[i][j] = *tempin;
        }
    }

    for (int i = zero1 / 2; i < matrixWidth - zero1 / 2; i++) {
        for (int j = zero2 / 2; j < matrixHeight - zero2 / 2; j++, tempout++) {
            float array[256];
            memset(array, 0, sizeof(array));
            for (int m = 0; m < number1; m++) {
                for (int n = 0; n < number2; n++) {
                    array[matrix[i - zero1 / 2 + m][j - zero2 / 2 + n]]++;
                }
            }
            for (int i = 0; i < 256; i++) {
                array[i] /= number1 * number2;
            }
            int res[256];
            memset(res, 0, sizeof(res));
            for (int i = 0; i < 256; i++) {
                float sum = 0.0;
                for (int j = 0; j <= i; j++) {
                    sum += array[j];
                }
                res[i] = round(sum * 255);
            }
            *tempout = res[matrix[i][j]];
        }
    }

    return (outimage);
}
```