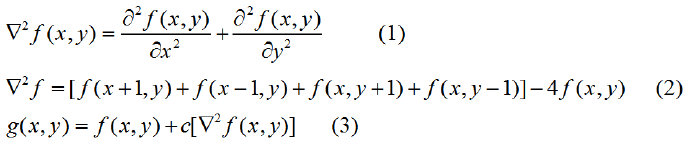
**Lab 4**

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

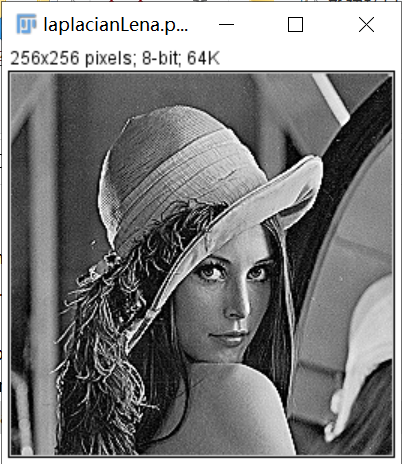
1. **Laplacian operator (lena, image from the textbook):**

**Algorithm:**



**Results (including pictures):**

Source: result (c=-1):

Source:



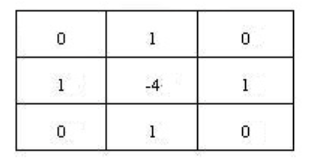
result(c=-1):



**Discussion:**

In order to better program, Laplacian has different operation templates. In equation (3), different c values can achieve different effects, and different templates can also achieve different effects.

I use this template,

and c equals -1.

**Codes:**

Image \*Laplacian\_operator(Image \*image)

{

unsigned char \*tempin, \*tempout;

int i, j;

int ia1,ja1,im1,jm1,temp;

Image \*outimage;

int width = image->Width;

int height = image->Height;

outimage = CreateNewImage(image, "#testing Swap", width, height);

tempin = image->data;

tempout = outimage->data;

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

{

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

{

temp = tempin[i \* width + j] - (tempin[(i-1) \* width + j] + tempin[(i+1) \* width + j] - 4 \* tempin[i \* width + j] + tempin[i \* width + j-1] + tempin[i \* width + j + 1]) ;

if (temp < 0) {

temp = 0;

}

else if (temp > 255) {

temp = 255;

}

tempout[i\*width + j] = (unsigned char)temp;

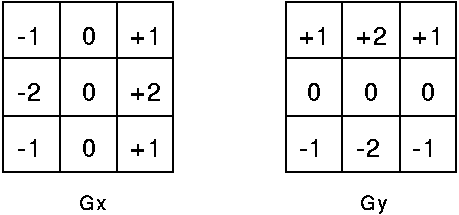
}

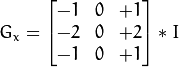
}return (outimage);

}

1. **Sobel operator (lena, image from the textbook):**

**Algorithm:**

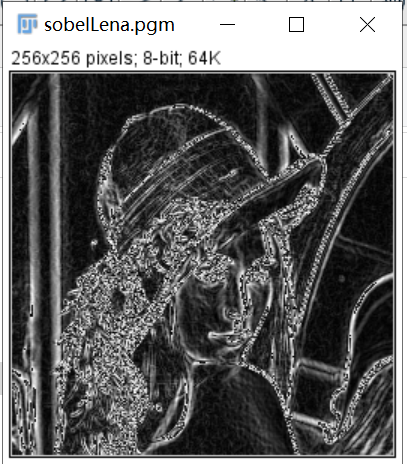






**Results (including pictures):**

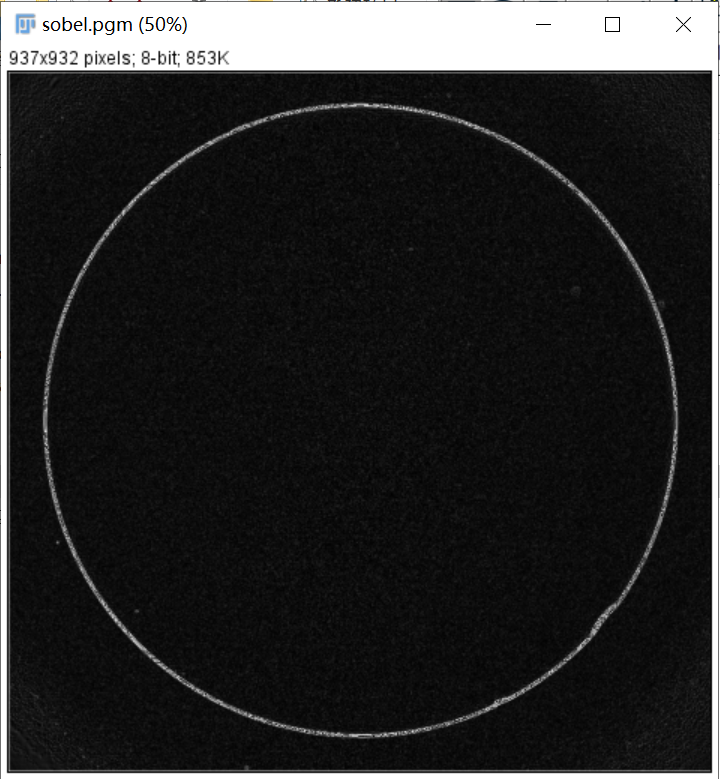
Source: result:

Source:



result:



**Discussion:**

After processing the image, the edge defects are clearly visible, and there is an additional point, that is, the pattern shadow with constant gray or slow change is removed, which simplifies the calculation task required by automatic detection.

**Codes:**

Image \*Sobel\_operator(Image \*image)

{

unsigned char \*tempin, \*tempout;

int i, j;

int gx = 0, gy = 0;

Image \*outimage;

int width = image->Width;

int height = image->Height;

outimage = CreateNewImage(image, "#testing Swap", width, height);

tempin = image->data;

tempout = outimage->data;

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

{

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

{

tempout[i\*width + j] = abs(tempin[(i - 1)\*width + j + 1] + 2 \* tempin[i\*width + j + 1] + tempin[(i + 1)\*width + j + 1] - (tempin[(i - 1)\*width + j - 1] + 2 \* tempin[i\*width + j - 1] + tempin[(i + 1)\*width + j - 1]))

+ abs(tempin[(i + 1)\*width + j - 1] + 2 \* tempin[(i + 1)\*width + j] + tempin[(i + 1)\*width + j + 1] - (tempin[(i - 1)\*width + j - 1] + 2 \* tempin[(i - 1)\*width + j] + tempin[(i - 1)\*width + j + 1]));

}

}

return (outimage);

}

1. **Gamma correction (lena, image from the textbook):**

**Algorithm:**

s = c(c = 1)

tempout[i\*width + j] = pow((double)tempin[i\*width + j] / 256,0.1) \* 256;

**Results (including pictures):**

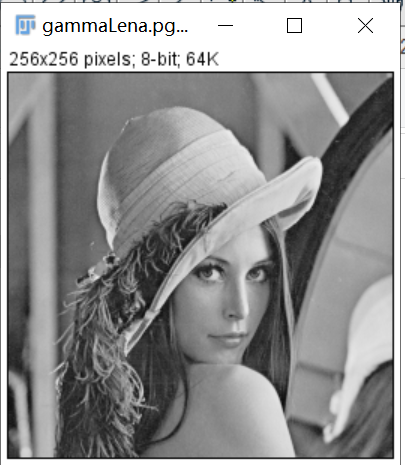
Source: result (0.1):

Source: result (0.4):

Source: result (0.7):

Source: result (1):

Source:



result(0.1):



result(0.4):



result(0.7):



result(1):



**Discussion:**

As the gamma value decreases, more detail becomes visible, but the contrast decreases. For different images, the gamma value will have an approximate limit point, below which the contrast will drop to an unacceptable level.

**Codes:**

Image \*Gamma\_correction(Image \*image)

{

unsigned char \*tempin, \*tempout;

int i, j;

Image \*outimage;

int width = image->Width;

int height = image->Height;

outimage = CreateNewImage(image, "#testing Swap", width, height);

tempin = image->data;

tempout = outimage->data;

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

{

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

{

tempout[i\*width + j] = pow((double)tempin[i\*width + j] / 256,0.6) \* 256;

}

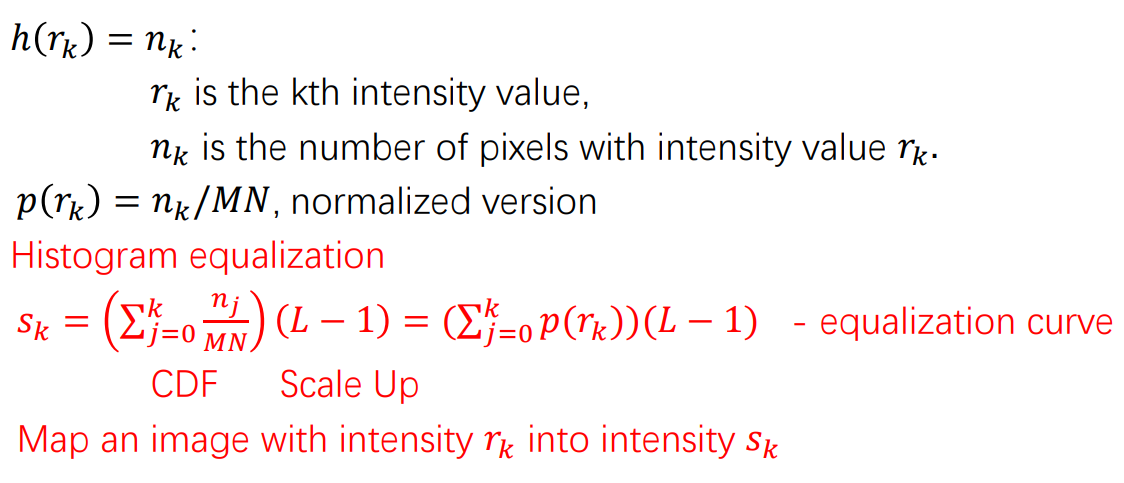
}

return (outimage);

}

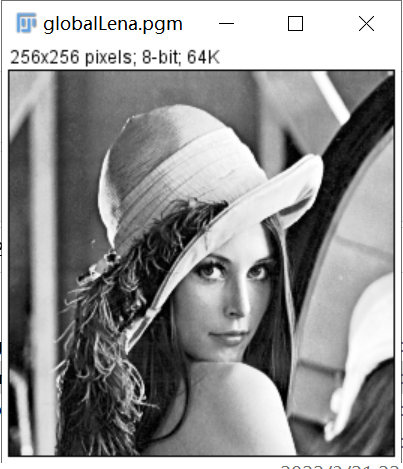
1. **Global enhancement (lena, image from the textbook):**

**Algorithm:**

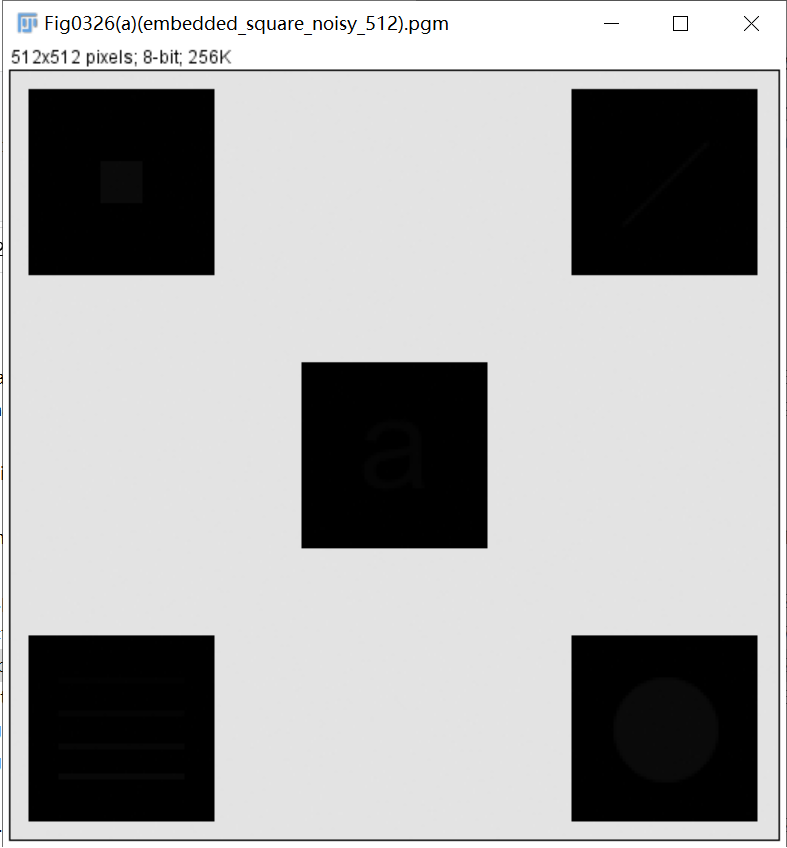


**Results (including pictures):**

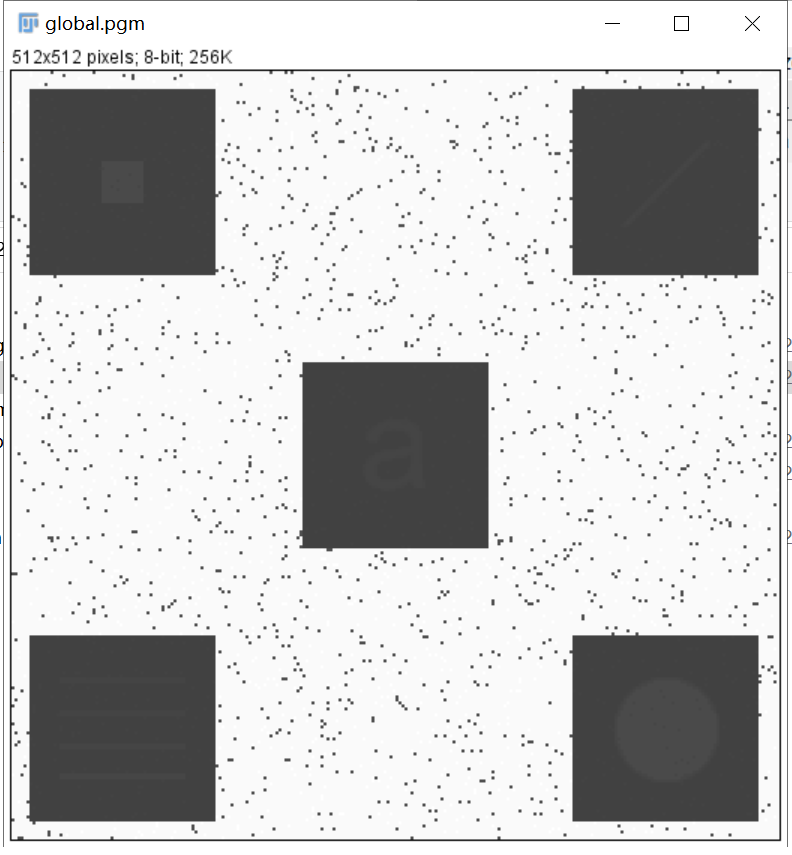
Source: result:

Source:



result:



**Discussion:**

After enhancement, the gray level of the image will become less and some details will disappear. When the histogram of the input image has a very dense part, the contrast of the enhanced image will be over enhanced.

**Codes:**

Image \*Global\_enhancement(Image \*image)

{

unsigned char \*tempin, \*tempout;

int m, n, p, i, j, k, r, temp = 0;

float a;

Image \*outimage;

int width = image->Width;

int height = image->Height;

outimage = CreateNewImage(image, "#testing Swap", width, height);

tempin = image->data;

tempout = outimage->data;

int R[256];

r = 0;

for (m = 0; m < 256; m++)

{

for (n = 0; n < width; n++)

{

for (p = 0; p < height; p++)

{

if (tempin[n\*width + p] == m)

{

r++;

}

}

}

R[m] = r;

}

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

{

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

{

tempout[i\*width + j] = floor(R[tempin[i\*width + j]]\*255 / (width \* height) + 0.5);

}

}return (outimage);

}

1. **Local enhancement (lena, image from the textbook):**

**Algorithm:**

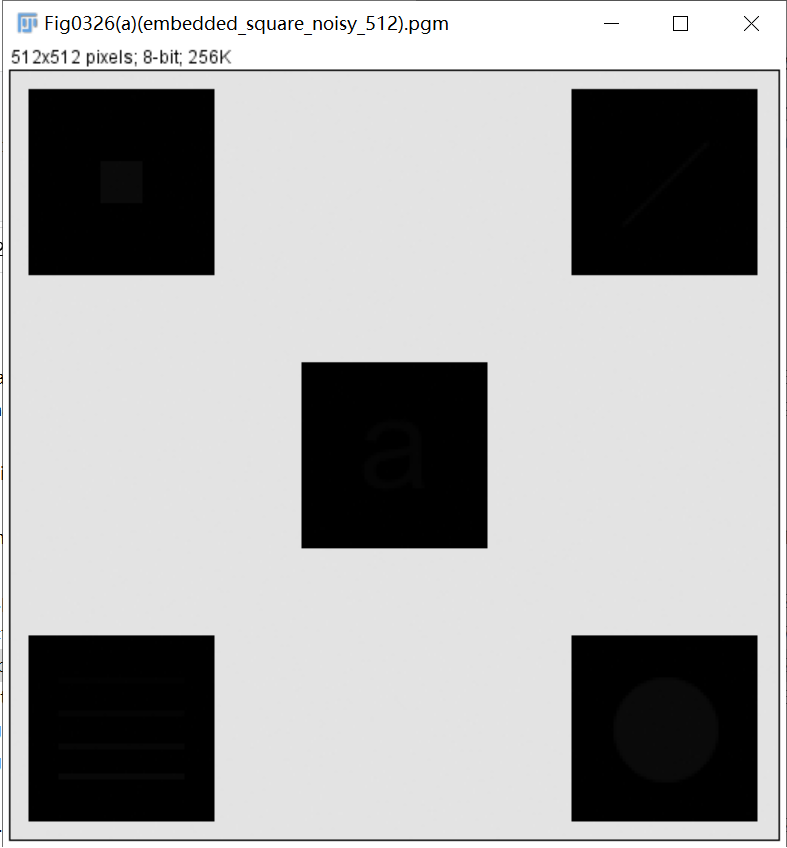
The algorithm is just like global enhancement. The only difference is that the local enhancement divides the picture into several parts and each part uses the global enhancement.

**Results (including pictures):**

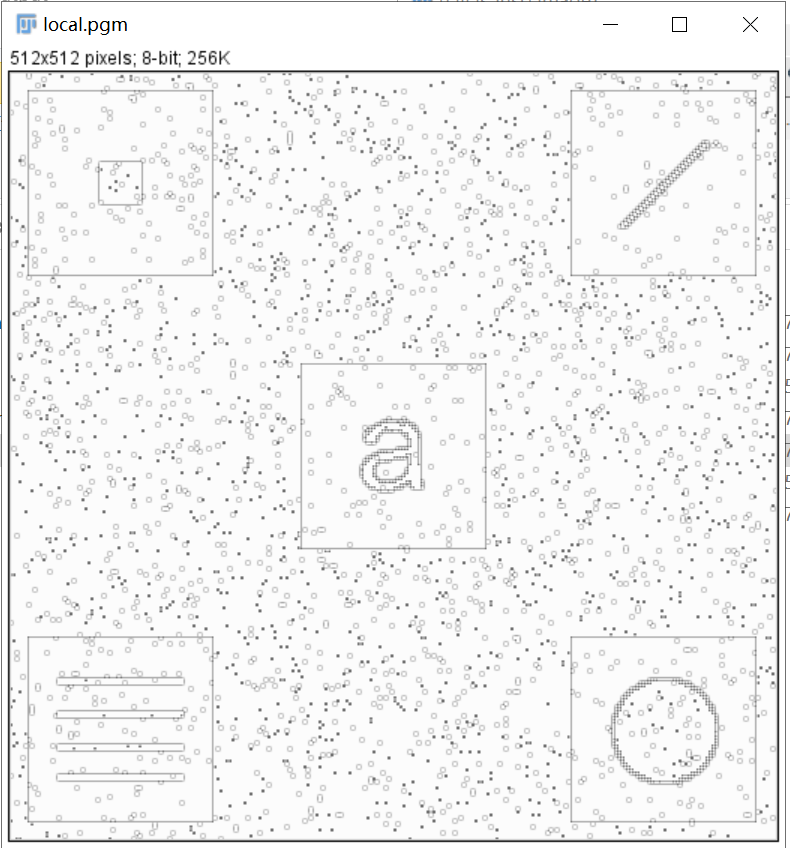
Source: result(3×3):

Source:



result(3×3):



**Discussion:**

The key of local enhancement is to divide the image into several parts and each parts use the global enhancement. Use a local window such as 3x3 to perform histogram equalization and then map the pixel value – better local details but possible blocky artifact.

**Codes:**

Image \*Local\_enhancement(Image \*image)

{

int Width, Height, size,i,j,k,t;

Width = image->Width;

Height = image->Height;

size = Width \* Height;

Image\* outimage;

unsigned char\* tempin, \* tempout,local[9];

outimage = CreateNewImage(image, "#testing GlobalHE",Width,Height);

tempin = image->data;

tempout = outimage->data;

double temp;

int nums[9];

int row, column;

for(i = 1; i < image->Height - 1; ++i){

for(j = 1; j < image->Width - 1; ++j){

int pos = 0;

for(k = -1; k < 2; ++k)

for(t = -1; t < 2; ++t)

local[pos++] = tempin[image->Width \* (i + k) + (j + t)];

temp = 0;

for (int i = 0;i < 9;i++) {

if (local[i] <= local[4]) {

temp++;

}

}

tempout[i\*Width + j] = floor(255 / 9 \* temp + 0.5);

}

}return(outimage);

}