1. 使用直方圖拓寬(histogram Stretching)影像對比增強。

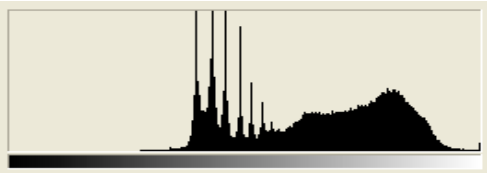
$$\text{Stretch}(I(r,c)) = \left[\frac{I(r,c) - I(r,c)_{\text{MIN}}}{I(r,c)_{\text{MAX}} - I(r,c)_{\text{MIN}}}\right][\text{MAX} - \text{MIN}] + \text{MIN}$$

$I(r,c)_{\text{MAX}}$ is the largest gray-level value in the image $I(r,c)$

$I(r,c)_{\text{MIN}}$ is the smallest gray-level value in $I(r,c)$

MAX and MIN correspond to the maximum and minimum gray-level values possible (for an 8-bit image these are 0 and 255)

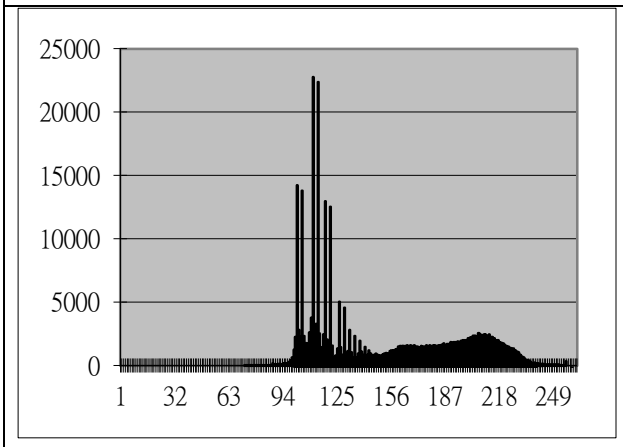
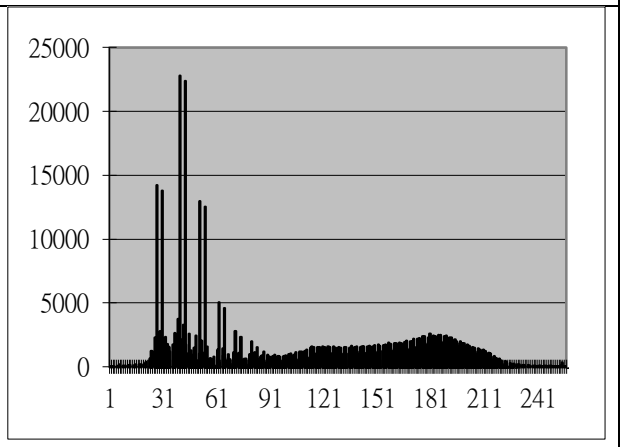
如下圖將 kaoshiung512x512.raw 的灰階分布拉寬至[0,255]。



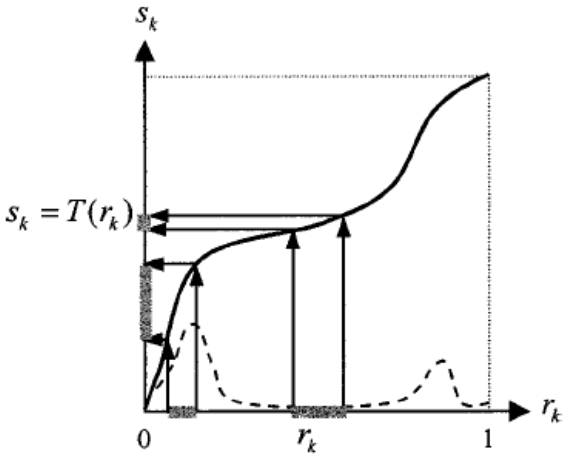
Source code :

```
#include <fstream.h>
#include "array.h"
void main()
{
      ifstream in("kaoshiung512x512.raw",ios::binary);
      ofstream out("histogram Stretching.raw",ios::binary);
      ofstream out1("histogram.txt");

      uc2D ima;
      ima.Initialize(512,512);
      char c;
      for(int i=0;i<ima.nr;i++)for(int j=0;j<ima.nc;j++)
      {
         in.get(c);ima.m[i][j]=c;
      }
      int max=0,min=255;
      for(int i=0;i<ima.nr;i++)for(int j=0;j<ima.nc;j++)
      {
         if(ima.m[i][j]>max)max=ima.m[i][j];
         if(ima.m[i][j]<min)min=ima.m[i][j];
      }

      for(int i=0;i<ima.nr;i++)for(int j=0;j<ima.nc;j++)
         ima.m[i][j]=(float(ima.m[i][j]-min)/(max-min))*255;

      //histogram
      int histo[256];
      for(int i=0;i<256;i++)histo[i]=0;
      for(int i=0;i<ima.nr;i++)for(int j=0;j<ima.nc;j++)
         histo[ima.m[i][j]]++;
      for(int i=0;i<256;i++)
         out1<<i<<"\t"<<histo[i]<<endl;

      for(int i=0;i<ima.nr;i++)for(int j=0;j<ima.nc;j++)
         out<<ima.m[i][j];
}
```

程式執行結果：

原始影像	拓寬後的影像
	
	

2. 使用 Histogram Equalization(HE)增強影像對比



Transformation function for histogram equalization.

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j), \quad = \sum_{j=0}^{k} \frac{n_j}{n},$$
$$0 \le r_k \le 1 \quad \text{and} \quad k = 0, 1, \ldots, L-1$$

where

$p_r(r_j) = n_j/n$	probability density function (pdf) of the input image level j;
n	total number of pixels in the input image;
n_j	input pixel number of level j.

演算法：

Step 1. 計算影像灰階統計直方圖(histogram)Pr

Step 2. 從灰階統計直方圖計算累增直方圖(cumulative histogram) Sk

Step 3. 從累增直方圖計算等化分布直方圖(equalized histogram)f(x)，使灰階頻率平均分布在 [X0, XL-1]: f(x)=X0+(XL-1-X0)Sk

X0 是期望的最小灰階值(例如 0)，XL-1 是期望的最大灰階值(例如 255)

Step 4. 以此等化分布直方圖 f(x)當作映射函數，重新指定影像每一 pixel 的灰階值。

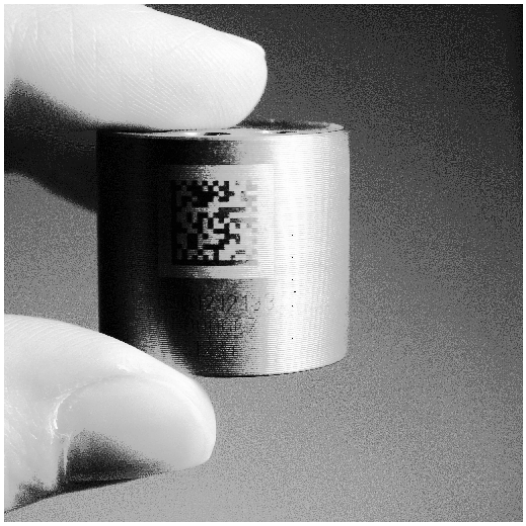
Source code :

```
#include <iostream>
#include "stdlib.h"
#include "bmp.h"
void HistogramEqualization(unsigned char **ima, unsigned char **bima, int nr,int nc);

using namespace std;

int main(int argc, char** argv) {

    unsigned char **ima, **bima;
    int nr,nc; //image height and width
    char filename[128],temp;
    bool isfilefine = false;
```

```
            //read bmp image from file
        cout << "Enter input filename:";
        cin >> filename;
        isfilefine = Read_BMP(filename, ima, nr, nc);
        if (!isfilefine)      return 0;
        bima=UC2D(nr,nc);
        Write_BMP_8bits("ima.bmp", ima, nr, nc);
        HistogramEqualization(ima,bima,nr,nc);
        Write_BMP_8bits("ch2_2.bmp", bima, nr, nc);
        cout << "\nProgram done.\n";
        return 1;
}

void HistogramEqualization(unsigned char **ima, unsigned char **bima, int nr,int nc)
{
        long ImaSize=nr*nc;
       int histo[256];    //histogram
       float accpbhisto[256]; // cumulative istogram
       int table[256];// Look-up table for mapping fuction of histogram equalization
  // Initialize
       for(int i=0;i<256;i++)
       {
          histo[i]=0;
          table[i]=0;
          accpbhisto[i]=0.0;
       }
  // Compute histogram
       for(int i=0;i<nr;i++)for(int j=0;j<nc;j++)histo[ima[i][j]]++;

  // Compute cumulative histogram
       accpbhisto[0]=float(histo[0])/float(ImaSize);
       for(int i=1;i<256;i++)
          accpbhisto[i]=accpbhisto[i-1]+float(histo[i])/float(ImaSize);

// compute mapping function
        for(int i=0;i<256;i++)table[i]=char(accpbhisto[i]*256.);

// Enhancement
```

```
        for(int i=0;i<nr;i++)for(int j=0;j<nc;j++)
                bima[i][j]=table[ima[i][j]];
}
```

程式執行結果：

原始影像 Hard3.bmp	HE 對比增強後的影像 histogram_Stretching.bmp
	

3. Local HE 影像增強方法

每一個 pixel 與鄰近 pixel 的灰階值比較，決定其排序。再依此一排序的正比關係指定一個新的灰階值給這個 pixel。Local HE 影像增強方法是根據區域性(而非整張影像)的資訊來增強對比。

```
for each (x,y) in image do
{
    rank = 0
    for each (i,j) in contextual region of (x,y) do
    {
        if image[x,y] > image[i,j] then
            rank = rank + 1
    }
    output[x,y] = rank * max_intensity / (# of pixels in contextual region)
}
```

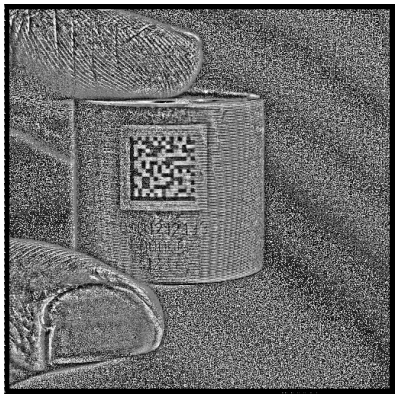
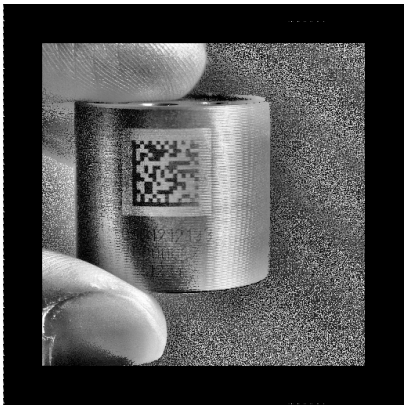
Source code :

```
#include <iostream>
#include "stdlib.h"
#include "bmp.h"
void HistogramEqualization(unsigned char **ima, unsigned char **bima, int nr,int nc);


using namespace std;
```

```
int main(int argc, char** argv) {

        unsigned char **ima, **bima;
        int nr,nc; //image height and width
        char filename[128],temp;
        bool isfilefine = false;

       //read bmp image from file
       cout << "Enter input filename:";
        cin >> filename;
        isfilefine = Read_BMP(filename, ima, nr, nc);
        if (!isfilefine)      return 0;
        bima=UC2D(nr,nc);
        Write_BMP_8bits("ima.bmp", ima, nr, nc);
        HistogramEqualization(ima,bima,nr,nc);
        Write_BMP_8bits("ch2_2.bmp", bima, nr, nc);
        cout << "\nProgram done.\n";
        return 1;
}

void HistogramEqualization(unsigned char **ima, unsigned char **bima, int nr,int nc)
{
        long ImaSize=nr*nc;
       int histo[256];    //histogram
       float accpbhisto[256]; // cumulative istogram
       int table[256];// Look-up table for mapping fuction of histogram equalization
  // Initialize
       for(int i=0;i<256;i++)
       {
          histo[i]=0;
          table[i]=0;
          accpbhisto[i]=0.0;
       }
  // Compute histogram
       for(int i=0;i<nr;i++)for(int j=0;j<nc;j++)histo[ima[i][j]]++;

  // Compute cumulative histogram
       accpbhisto[0]=float(histo[0])/float(ImaSize);
```

```
        for(int i=1;i<256;i++)
            accpbhisto[i]=accpbhisto[i-1]+float(histo[i])/float(ImaSize);

// compute mapping function
        for(int i=0;i<256;i++)table[i]=char(accpbhisto[i]*256.);

// Enhancement
        for(int i=0;i<nr;i++)for(int j=0;j<nc;j++)
                bima[i][j]=table[ima[i][j]];
}
```

程式執行結果(LocalHE.bmp) :

區域大小視窗 15X15	區域大小視窗 40X40	區域大小視窗 100X100
		

4. 參數可調整的 HE 影像增強方法—AHE(Adaptive Histogram Equalization)

$$\text{ACE} = k_1 \left[\frac{m_{I(r,c)}}{\sigma_I(r,c)} \right] \left[I(r,c) - m_I(r,c) \right] + k_2 \, m_I(r,c)$$

where $m_{I(r,c)}$ = is the mean for the entire image $I(r,c)$

σ_l = local standard deviation (in the window under consideration)

m_l = local mean (average in the window under consideration)

k_1, k_2 = constants, vary between 0 and 1

Source code :

```
#include <iostream>
#include <math.h>
#include "stdlib.h"
#include "bmp.h"
void mean_stddev(unsigned char **ima, float &mean, float &std_dev, int nr, int nc);

int main(int argc, char** argv) {

        unsigned char **ima, **ahima, **window;
        int nr,nc; //image height and width
        char filename[128], c;
        bool isfilefine = false;

       //read bmp image from file
       cout << "Enter input filename:";
        cin >> filename;
        isfilefine = Read_BMP(filename, ima, nr, nc);
        if (!isfilefine)      return 0;
        ahima=UC2D(nr,nc);
        Write_BMP_8bits("ima.bmp", ima, nr, nc);

        for(int i=0;i<nr;i++)
                for(int j=0;j<nc;j++)
                        ahima[i][j]=ima[i][j];

        int winsize=21, hsize=winsize/2;
        window=UC2D(winsize,winsize);
        float globalmean=0, mean=0.0;
```

```
	float std_dev=0.0;
	float k1=0.0, k2=0.0;
	cout<<"input k1= ";
	cin>>k1;
	cout<<"input k2= ";
	cin>>k2;
	globalmean=0;
	for(int i=0;i<nr;i++)
		for(int j=0;j<nc;j++)
			globalmean+=ima[i][j];
	globalmean=globalmean/(nr*nr);
	int t;
	for(int i=hsize;i<(nr-hsize);i++)
		for(int j=hsize;j<(nc-hsize);j++)
		{
			for(int ii=-hsize;ii<=hsize;ii++)
				for(int jj=-hsize;jj<=hsize;jj++)
					window[ii+hsize][jj+hsize]=ima[i+ii][j+jj];
			mean_stddev(window, mean, std_dev, winsize, winsize);
			t=(k1*(globalmean/std_dev)*(ima[i][j]-mean))+(k2*mean);
			if(t>255)
				ahima[i][j]=255;
			else
				ahima[i][j]=t;
		}
	Write_BMP_8bits("AHE.bmp", ahima, nr, nc);

	cout << "\nProgram done.\n";
	return 1;
}

void mean_stddev(unsigned char **ima, float &mean, float &std_dev, int nr, int nc)
{
	long N, sum=0;
	N=(long)nr*(long)nc;
	for(int i=0;i<nr;i++)
		for(int j=0;j<nc;j++)
			sum+=ima[i][j];
```

```
        mean=(float)sum/(float)N; //Calculating the mean

        float sumdev=0.0;
        float d=0.0;
        for(int i=0;i<nr;i++)
                for(int j=0;j<nc;j++)
                {
                        d=ima[i][j]-mean;
                        sumdev=sumdev+d*d;
                }
        std_dev=sqrt(sumdev/N); //Calculating the standard deviance
}
```

程式執行結果(AHE.bmp)：

k1 = 0.1, k2 = 0.1	k1 = 0.5, k2 = 0.5	k1 = 0.9, k2 = 0.9
		
k1 = 0.1, k2 = 0.9	k1 = 0.3, k2 = 0.7	k1 = 0.9, k2 = 0.1