Assignment 6

/*叶俊利 3130100672*/

Bilateral Filtering

First, according to the equation:

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|) I_{\mathbf{q}}$$

We need to set the two parameters of Gaussian function.

According to the PPT:

Depends on the application. For instance:

- space parameter: proportional to image size
  - e.g., 2% of image diagonal
- intensity parameter: proportional to edge amplitude
  - e.g., mean or median of image gradients
- independent of resolution and exposure

I set the first parameter to be 2% of the diagonal:

```
double dSigma1 = 0.02*pow((nWidth*nWidth + nHeight*nHeight),0.5);
```

Then, set the second parameter to be mean of the image gradients:

```
double dSigmaR = 0, dSigmaG = 0, dSigmaB = 0;
for (int i = 0; i < nHeight-1; i++)
{
        for (int j = 0; j < nWidth-1; j++)
        {
                pixel = imgBackup.GetPixel(j, i);
                double r = GetRValue(pixel);
                double g = GetGValue(pixel);
                double b = GetBValue(pixel);
                pixel = imgBackup.GetPixel(j + 1, i);
                double r1 = GetRValue(pixel);
```

```
                double g1 = GetGValue(pixel);
                double b1 = GetBValue(pixel);
                pixel = imgBackup.GetPixel(j, i + 1);
                double r2 = GetRValue(pixel);
                double g2 = GetGValue(pixel);
                double b2 = GetBValue(pixel);
                dSigmaR += abs(r - r1) + abs(r - r2);
                dSigmaG += abs(g - g1) + abs(g - g2);
                dSigmaB += abs(b - b1) + abs(b - b2);
            }
        }
        dSigmaR /= (nHeight - 1)*(nWidth - 1);
        dSigmaG /= (nHeight - 1)*(nWidth - 1);
        dSigmaB /= (nHeight - 1)*(nWidth - 1);
```

Also, we need to set the size of the windows. In normal distribution, area [-3σ, 3σ] cover more than 99% of the possibility. Therefore, I set the radius of the windows to be 3σ. I.e. The size of windows is (6σ +1)*(6σ+1). The code is shown:

```
int nWindows = (int)(1 + 2 * ceil(3 * dSigma1));
```

Now, in order to speed up the program, we can pre-calculate the Gaussian matrix.

```
D       ouble* dDistance = new double[nWindows*nWindows]; //计算距离中间点的几何距离
        for (int i = 0; i < nWindows*nWindows; i++)
        {
            int nNumX = i / nWindows;
            int nNumY = i%nWindows;
            dDistance[i] = ((nWindows - 1) / 2 - nNumX)*((nWindows - 1) / 2 - nNumX) +
            ((nWindows - 1) / 2 - nNumY)*((nWindows - 1) / 2 - nNumY);
            dDistance[i] = exp(-0.5*dDistance[i] / dSigma1 / dSigma1);   //距离参数
        }
```

OK, we can begin to process the image now:

```
    for (int i = 0; i < nHeight; i++)
```

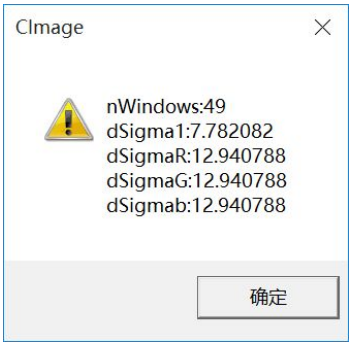
```
{
        for (int j = 0; j < nWidth; j++)
        {
                pixel = imgBackup.GetPixel(j, i);
                double rP = GetRValue(pixel);
                double gP = GetGValue(pixel);
                double bP = GetBValue(pixel);
                double dDataR = 0.0;
                double dTotalR = 0.0;                        //用于进行归一化
                double dDataG = 0.0;
                double dTotalG = 0.0;                        //用于进行归一化
                double dDataB = 0.0;
                double dTotalB = 0.0;                        //用于进行归一化
                for (int m = 0; m < nWindows*nWindows; m++)
                {
                        int nNumX = m / nWindows;                //行索引
                        int nNumY = m%nWindows;                //列索引
                        int nX = i - (nWindows - 1) / 2 + nNumX;
                        int nY = j - (nWindows - 1) / 2 + nNumY;
                        if ((nY >= 0) && (nY < nWidth) && (nX >= 0) && (nX < nHeight))
                        {
                                pixel = imgBackup.GetPixel(nY, nX);
                                double r = GetRValue(pixel);
                                double g = GetGValue(pixel);
                                double b = GetBValue(pixel);
                                double dRDiff = fabs(rP - r);
                                double dGDiff = fabs(gP - g);
                                double dBDiff = fabs(bP - b);
                                dRDiff = exp(-0.5*dRDiff * dRDiff / dSigmaR / dSigmaR);
                                //色参数
```

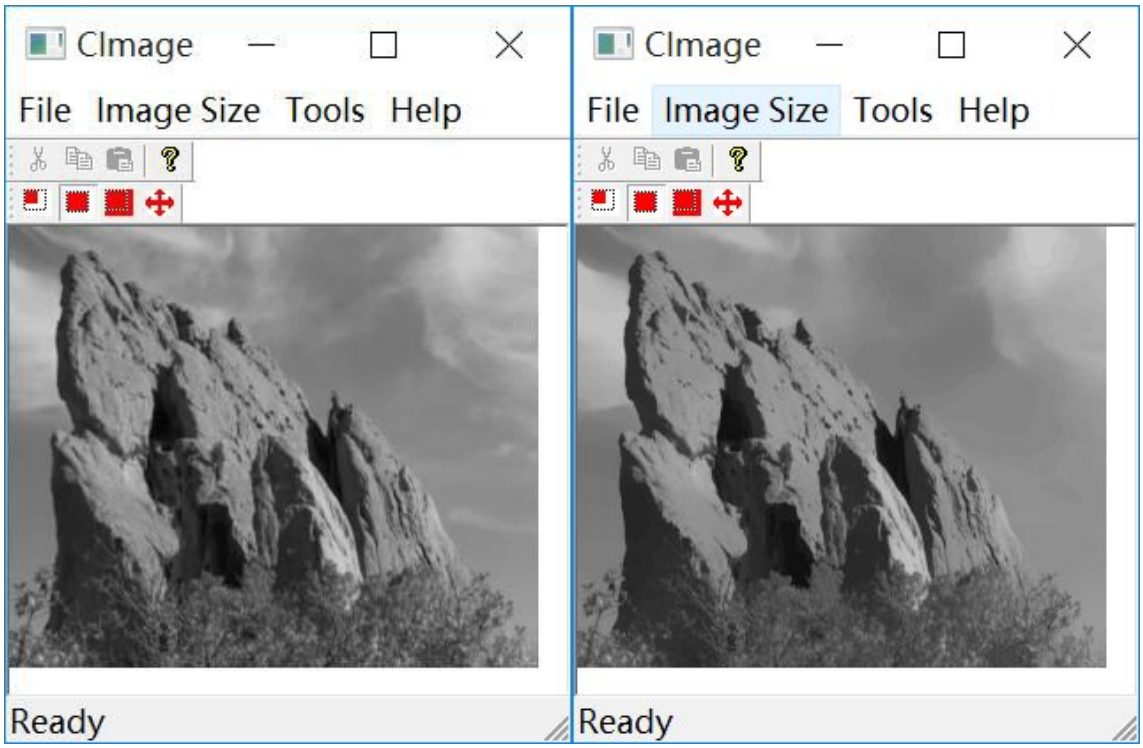
```
                        dGDiff = exp(-0.5*dGDiff * dGDiff / dSigmaG / dSigmaG);
                        //色参数
                        dBDiff = exp(-0.5*dBDiff * dBDiff / dSigmaB / dSigmaB);
                        //色参数
                        if (m != 4)
                        {
                                dDataR += r * dRDiff * dDistance[m];
                                //距离和色参数综合
                                dTotalR += dRDiff * dDistance[m];
                                //加权系数求和，进行归一化
                                dDataG += g * dGDiff * dDistance[m];
                                dTotalG += dGDiff * dDistance[m];
                                dDataB += b * dBDiff * dDistance[m];
                                dTotalB += dBDiff * dDistance[m];
                        }
                }
        }
        dDataR /= dTotalR;
        dDataG /= dTotalG;
        dDataB /= dTotalB;
        imgOriginal.SetPixelRGB(j, i, (byte)dDataR, (byte)dDataG, (byte)dDataB);
        }
        Invalidate();
        UpdateWindow();
}
```

Results

First I test a image of 299 * 249.

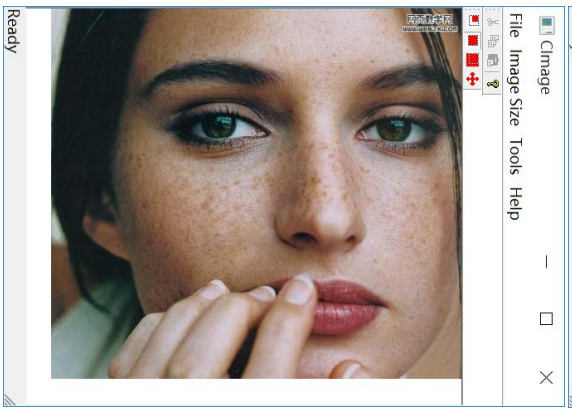
The generated parameters are:



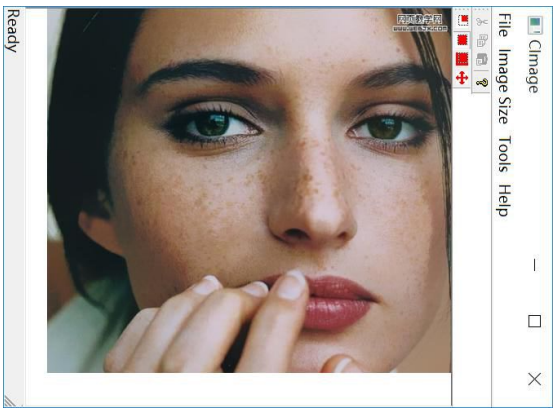


Original Filtered

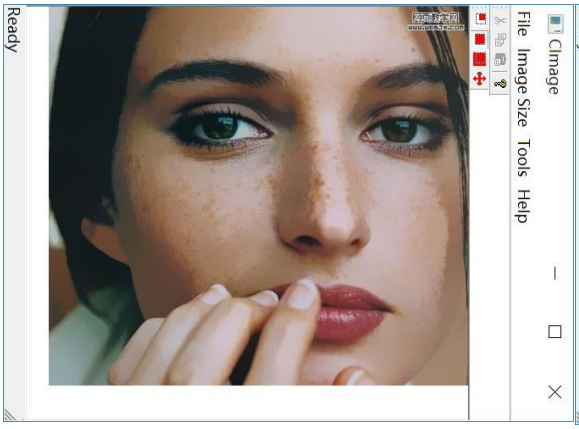
Here is another test:



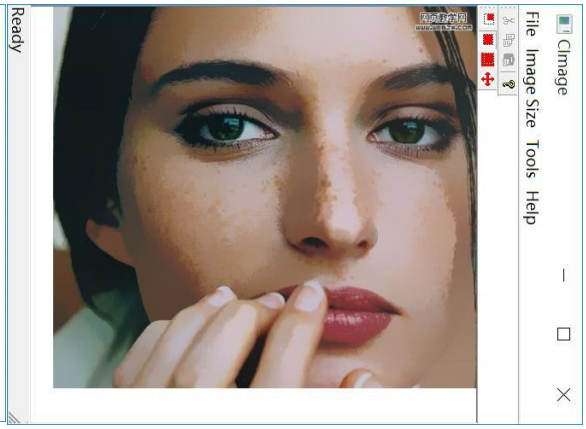
Original



Filtered Once



Filtered Twice



Filtered Three Times