图像处理算法

一、锐化  
  
　　锐化的算法很简单，就是比较相邻的几个像素，把当前像素加上和周围的像素的差就可以了。这里我给出一个示例：

  
　　假设有一个图片，4\*4，共16个像素，分别用A－－L来代表。我们先观察这个图片，只有中间的F,G,J,K这四个像素的“邻居”是全的。  
  
　　为了简便起见，我们只处理这4个像素，因为在实际的图片中由于图片的大小都很多像素组成，所以周围的一圈像素不做处理不会影响到最终的效果。  
  
　　先计算差值:

|  |
| --- |
| Delta= F - (A+B+C+E+G+I+J+K) / 8 (A+B+C+E+G+I+J+K) / 8就是F周围的像素的平均值， |

　　将这个平均值乘以一个系数再加到F上，就得到了一个新的F值：

|  |
| --- |
| F=F + Delta \* Alpha |

　　这个系数Alpha就是锐化度，改变这个系数就能得到不同的锐化效果。不过一般都是取得比较小的，如：0.3  
  
　　于是，我们只要使用两个循环来遍历整个图片的像素值（去除边界）就能得到一个锐化的效果了。  
  
　　但是大家或许会发现在处理后面几个点的时候，前面的点的值已经不是原来的值了，比如处理G的时候，需要用到F的值，而F则已经被改变，并且F的改变又和G的值有关系，这样就会变成一种循环引用。为了避免整个问题，这里给出一个改良的方法：

  
　　我们从A点开始做，将差值计算方法改成:

|  |
| --- |
| Delta= A - (B+E+F) / 3 F=F + Delta \* Alpha |

　　按照从左到右，从上到下的顺序来扫描所有像素，这时在计算中就不会遇到已经被处理过的像素了，并且因为减少了参与运算的像素，整个处理过程也得以加快。

二、柔化  
 柔化算法的效果是让图片的每一个点与周围点的颜色更平滑，算法原理很简单，就是针对每一个像素，将其颜色值置为周围8个点加上自身的RGB的平均值。不过这样处理后的效果不是很明显，可以采用高斯模糊算法，能获取更好的效果

　　柔化的算法和锐化相近似，不过作用正好相反，就是把当前点用周围几个点的平均值来代替。

  
　　计算方法：

|  |
| --- |
| F=(A+B+C+E+F+G+I+J+K) / 9 G=(B+C+D+F+G+H+J+K+L) / 9 ... ... |

三、扩散  
  
　　产生一种类似水彩画的效果。  
  
　　算法很简单，就是将当前点用周围的随即的点来代替。

  
　　F点可以从它周围的A,B,C,E,G,I,J,K中任意选一点代替。  
　　G点可以从它周围的B,C,D,F,H,J,K,L中任意选一点代替。  
　　J点可以从它周围的E,F,G,I,K,M,N,O中任意选一点代替。  
　　K点可以从它周围的F,G,H,J,L,N,O,P中任意选一点代替。  
  
　　至于选哪一点，可以用一个随即数来选定。

四、雕刻

浮雕的算法相对复杂一些，用当前点的RGB值减去相邻点的RGB值并加上128作为新的RGB值。由于图片中相邻点的颜色值是比较接近的，因此这样的算法处理之后，只有颜色的边沿区域，也就是相邻颜色差异较大的部分的结果才会比较明显，而其他平滑区域则值都接近128左右，也就是灰色，这样就具有了浮雕效果。

在实际的效果中，这样处理后，有些区域可能还是会有”彩色”的一些点或者条状痕迹，所以最好再对新的RGB值做一个灰度处理

　　将相邻的两个像素相减，得到的差加上127作为新的值

  
　　如果我们按照从左向右的方向来“雕刻”

|  |
| --- |
| A=B-A+127 B=C-B+127 C=D-C+127 ... |

　　如果我们按照从上向下的方向来“雕刻”

|  |
| --- |
| A=E-A+127 B=F-B+127 C=G-C+127 ... |

　　当然我们还可以从更多的方向来“雕刻”比如：向左下、右上、左上、右下...等等，一共8个可以选择的方向。  
  
　　另外这个127，就是“雕刻”效果后的亮度。我们可以把雕刻方向和亮度都作为参数写到过程中

## 五、灰度

灰度公式：

Gray = R\*0.299 + G\*0.587 + B\*0.114

Gray = (R\*19595 + G\*38469 + B\*7472) >> 16

## ****六、黑白图片****

这里的黑白图片，跟上面的灰度不一样。灰度有256种颜色，而黑白则是只保留黑和白这两种颜色，看了后面的对比处理图片就能明白了。

黑白图片的处理算法更简单：  
求RGB平均值Avg ＝ (R + G + B) / 3，如果Avg >= 100，则新的颜色值为R＝G＝B＝255；如果Avg < 100，则新的颜色值为R＝G＝B＝0；255就是白色，0就是黑色；至于为什么用100作比较，这是一个经验值吧，设置为128也可以，可以根据效果来调整。

## ****七 底片效果****

算法原理：将当前像素点的RGB值分别与255之差后的值作为当前点的RGB值，即  
R = 255 – R；G = 255 – G；B = 255 – B；

## 八、光照

有时需要在照片中增加一个光源这样的效果，如何实现呢？

首先我们设定一个光源中心坐标和光照强度，然后以此光源和图片边缘的最短距离为半径，依次为每个点的RGB增加一个同样的值，当然，图片上的点距离光源中心越远，则增加的颜色值越小，通过这样的方法，就能够实现光照的效果了。

## 九 放大镜

前面介绍过通过坐标变换矩阵来缩放、旋转图片的方法，这里通过直接操作图像数据来实现图像的局部放大功能。

假如我们定义放大镜的坐标为(CenterX，CenterY)，半径为Radius，而放大倍数为M = 2，那么其实就是将原图中的坐标为(CenterX，CenterY)、半径为Radius/M的区域的图像放大到放大镜覆盖的区域即可，算法其实很简单，对图片上的每一个点(X,Y)，求其与(CenterX，CenterY)的距离Distance，若Distance < Radius，则取原图中坐标为(X/M,Y/M)的像素的颜色值作为新的颜色值。