浙江大学实验报告

课程名称：图象信息处理　　指导老师：宋明黎　　成绩：_____

实验名称：Assignment-3　visibility enhancement and Histogram equalization

专业：求是科学班（计算机）

姓名：蒋仕彪

学号：3170102587

日期：2018/11/18

一、实验目的和要求

1.学习和掌握图像的对数处理方法。

2.学习和掌握图像的直方图均衡化处理方法。

二、实验内容和原理

1. 图像的对数处理方法。

对数处理的公式为：

- In order to enhance the image's visibility, adjust the pixel value by a logarithmic operator.
- $L_d = \frac{\log(L_w+1)}{\log(L_{max}+1)}$
- L_d is display luminance, L_w is the real luminance, L_{max} is the maximal luminance value in the image.

当前 pixel 的亮度 Lw 可能会是 0（全黑），导致 log 里的结果出现 0 而 error。解决办法是，在 Lmax 和 Lw 后面都加上 1。

2. 图像的直方图均衡化（Histogram Equalization）

比如考虑 YUV 模式下的维度 Y。

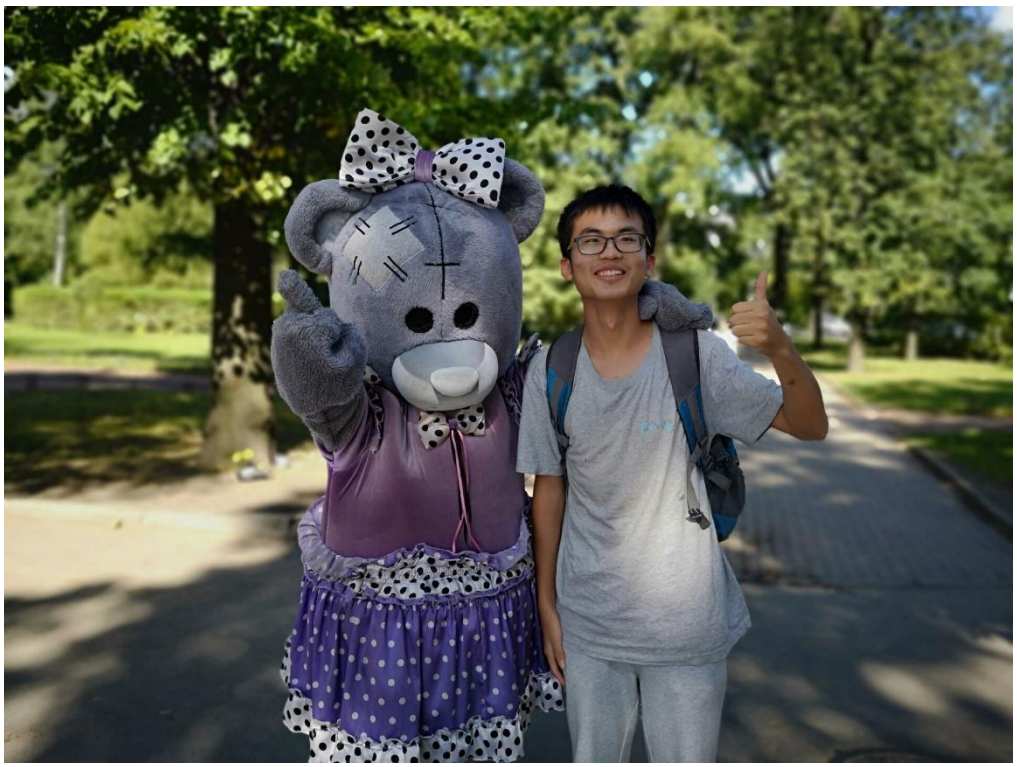
在离散的情况下（设第 i 行第 j 列的像素是 a[i][j]）：

定义 $$h_k = \sum_{i=1}^{h}\sum_{j=1}^{w}[a_{i,j} = k]$$

$$s_k = \frac{1}{wh}\sum_{i=0}^{k} h_i$$

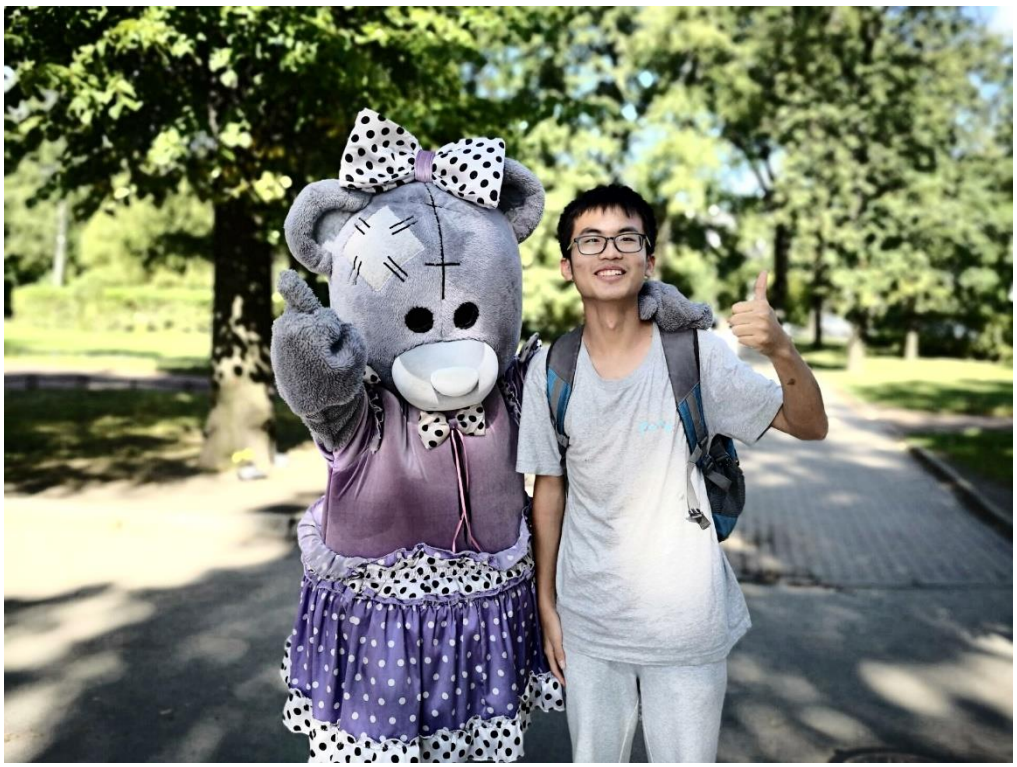
则 $$a'_{i,j} = s_{a_{i,j}} \times 255$$

三、成果展示



原彩色图像

对数处理后的图像



对 Y 做直方图均衡化后的图像

对 RGB 分别做直方图均衡化的图像

四、源代码与分析

分析：

对原图直接做对数处理后，感觉图片整体都变亮变白了好多。这也是在意料之中，因为 log 函数在后来会趋于平缓，很多 Y 较大的色块都会趋于白色。

对 Y 做直方图均衡化后感觉还行，图像稍微变白了一些。

如果对 RGB 分别均衡化，颜色间的对比度提高了不少。和原图做比较，感觉颜色略奇怪。

总的来说，对数处理感觉结果更加柔和，对比度降低；而直方图均衡化会保留光影效果，对比度高。

源代码：

```
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <stdlib.h>

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef int LONG;

typedef struct  tagBITMAPFILEHEADER{
    WORD type;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
}head1;
//定义第一个头

typedef struct tagBITMAPINFOHEADER{
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD  biCompression;
    DWORD  biSizeImage;
    LONG  biXPelsPerMeter;
    LONG  biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
}head2;
//定义第二个头
```

```
typedef struct _RGB{
    BYTE R;
    BYTE G;
    BYTE B;
}RGB;

typedef struct _YUV{
    short Y;
    short U;
    short V;
}YUV;
//YUV 格式可能会有负数，就直接用 short 存了

typedef struct _HSV{
    short H;
    short S;
    short V;
}HSV;

YUV RGB_To_YUV(RGB cur){
    YUV ret;
    ret.Y = round(0.299 * cur.R + 0.587 * cur.G + 0.114 * cur.B);
    ret.U = round(-0.147 * cur.R - 0.289 * cur.G + 0.435 * cur.B);
    ret.V = round(0.615 * cur.R - 0.515 * cur.G - 0.100 * cur.B);
    return ret;
}

BYTE In(short cur){
    if (cur > 255) cur = 255;
    if (cur < 0)   cur = 0;
    return (BYTE)cur;
}
//担心 YUV 转 RGB 时导致 RGB 范围出错，写一个框定范围的函数

RGB YUV_To_RGB(YUV cur){
    RGB ret;
    ret.R = In(round(cur.Y + 1.14 * cur.V));
    ret.G = In(round(cur.Y - 0.395 * cur.U - 0.581 * cur.V));
    ret.B = In(round(cur.Y + 2.033 * cur.U));
    return ret;
}

int line_byte, extra_byte, S;
```

```
head1 bmfh;
head2 bmih;

void IntoStream(RGB *cur, BYTE *p){
    for (int i = 0; i < S; i++){
        *p++ = cur->R;
        *p++ = cur->G;
        *p++ = cur->B;
        if ((i + 1) % bmih.biWidth == 0)
            for (int k = 0; k < extra_byte; k++)
                *p++ = 0;
        cur++;
    }
}
//将 RBG 数组导入到最后的输出流里

YUV Logarithmic(YUV *p, YUV *q, int S){
    short Lmax = 0;
    for (int i = 0; i < S; i++)
        if (p[i].Y > Lmax) Lmax = p[i].Y;
    printf("%d\n", Lmax);
    for (int i = 0; i < S; i++){
        q[i] = p[i];
        q[i].Y = round(log(p[i].Y + 1) / log(Lmax + 1) * 255);
        //除掉最大值的对数后，将会被限制在[0,1]；所以乘上 255 来确定具体的值
    }
}
//将图片进行对数处理

void EqualizationY(YUV *p, YUV *q, int S){
    int num[256] = {0};
    for (int i = 0; i < S; i++)
        num[p[i].Y]++;
    for (int i = 1; i <= 255; i++)
        num[i] += num[i - 1];
    for (int i = 0; i < S; i++){
        q[i] = p[i];
        q[i].Y = round(num[p[i].Y] * 255.0 / S);
    }
}
//将 YUV 的 Y 进行直方图均衡化

void EqualizationRGB(RGB *p, RGB *q, int S){
    int numR[256] = {0}, numG[256] = {0}, numB[256] = {0};
```

```
    for (int i = 0 ; i < S; i++){
        numR[p[i].R]++;
        numG[p[i].G]++;
        numB[p[i].B]++;
    }
    for (int i = 1; i <= 255; i++){
        numR[i] += numR[i - 1];
        numG[i] += numG[i - 1];
        numB[i] += numB[i - 1];
    }
    for (int i = 0; i < S; i++){
        q[i].R = round(numR[p[i].R] * 255.0 / S);
        q[i].G = round(numG[p[i].G] * 255.0 / S);
        q[i].B = round(numB[p[i].B] * 255.0 / S);
    }
}
//将 RGB 色彩中的每一维分别进行直方图均衡化

int main(){
    FILE* fin = fopen("Origin.bmp", "rb"), * fout;   //读入文件

    fread(&bmfh, 14, 1, fin);
    fread(&bmih, sizeof(head2), 1, fin);

    line_byte = (bmih.biWidth * 3 + 3) / 4 * 4; //计算实际存储时每行的字节数
    extra_byte = line_byte - bmih.biWidth * 3;  //计算每行结尾空的字节数
    S = bmih.biWidth * bmih.biHeight;           //计算像素总个数
    int all =  line_byte * bmih.biHeight;       //计算像素矩阵总的字节数

    BYTE *Stream = (BYTE *) malloc(all);
    fread(Stream, 1, all, fin);

    RGB *Origin = (RGB*) malloc(S * sizeof(RGB));

    BYTE *p = Stream;
    RGB *cur = Origin;
    for (int i = 0; i < S; i++){
        cur->R = *p++;
        cur->G = *p++;
        cur->B = *p++;
        if ((i + 1) % bmih.biWidth == 0)
            p = p + extra_byte;
        cur++;
    }
```

```
    //从读入流里获取实际的像素矩阵

    YUV *now = (YUV *) malloc(S * 6);           //原图的 YUV 格式
    YUV *New = (YUV *) malloc(S * 6);           //新图的 YUV 格式
    RGB *Print = (RGB *) malloc(S * 3);         //新图的 RGB 格式（为了输出）
    for (int i = 0; i < S; i++)
        now[i] = RGB_To_YUV(Origin[i]);

    Logarithmic(now, New, S);
    for (int i = 0; i < S; i++)
        Print[i] = YUV_To_RGB(New[i]);
    IntoStream(Print, Stream);
    fout = fopen("Logarithmic.bmp", "wb");
    //输出对数处理后的结果
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&bmih, sizeof(head2), 1, fout);
    fwrite(Stream, 1, all, fout);

    EqualizationY(now, New, S);
    for (int i = 0; i < S; i++)
        Print[i] = YUV_To_RGB(New[i]);
    IntoStream(Print, Stream);
    fout = fopen("Equalization_for_Y.bmp", "wb");
    //输出对 Y 直方图均衡化的结果
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&bmih, sizeof(head2), 1, fout);
    fwrite(Stream, 1, all, fout);

    EqualizationRGB(Origin, Print, S);
    IntoStream(Print, Stream);
    fout = fopen("Equalization_for_RGB.bmp", "wb");
    //输出对 RGB 分别直方图均衡化后的结果
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&bmih, sizeof(head2), 1, fout);
    fwrite(Stream, 1, all, fout);

    return 0;
}
```