



# 图像信息处理实验报告

**Digital Image Processing (Experiments)**

|  |  |
| --- | --- |
| 姓 名 |  |
| 指导老师 |  |
| 学 号 |  |
| 专业班级 |  |

**二〇一九年**

**秋冬学期**

## 实验三

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称：** | 作业3 | **指导老师：** |  | **成绩：** |  |

### 一、实验目的和要求（必填）

1、可见性增强的图像对数运算（Image logarithmic operation for visibility enhancement）

2、直方图均衡（Histogram equalization）

### 二、实验内容和原理（必填）

#### 1、图像对数运算

由于对数曲线在像素值较低的区域斜率大，在像素值较高的区域斜率较小，所以图像经过对数变换后，较暗区域的对比度将有所提升，所以就可以增强图像的暗部细节。

对数变换可以将图像的低灰度值部分扩展，显示出低灰度部分更多的细节，将其高灰度值部分压缩，减少高灰度值部分的细节，从而达到强调图像低灰度部分的目的。变换方法：

#### 2、直方图均衡

直方图均衡化(Histogram Equalization) 又称直方图平坦化,实质上是对图像进行非线性拉伸,重新分配图像象元值,使一定灰度范围内象元值的数量大致相等。这样,原来直方图中间的峰顶部分对比度得到增强,而两侧的谷底部分对比度降低,输出图像的直方图是一个较平的分段直方图:如果输出数据分段值较小的话,会产生粗略分类的视觉效果。

图1所示就是直方图均衡化, 即将随机分布的图像直方图修改成均匀分布的直方图。基本思想是对原始图像的像素灰度做某种映射变换, 使变换后图像灰度的概率密度呈均匀分布。这就意味着图像灰度的动态范围得到了增加, 提高了图像的对比度。

设变量r 代表图像中像素灰度级。对灰度级进行归一化处理, 则0≤r≤1, 其中r= 0表示黑, r= 1表示白。对于一幅给定的图像来说, 每个像素值在[ 0,1] 的灰度级是随机的。用概率密度函数[clip_image003](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183441580-807001235.png)来表示图像灰度级的分布。

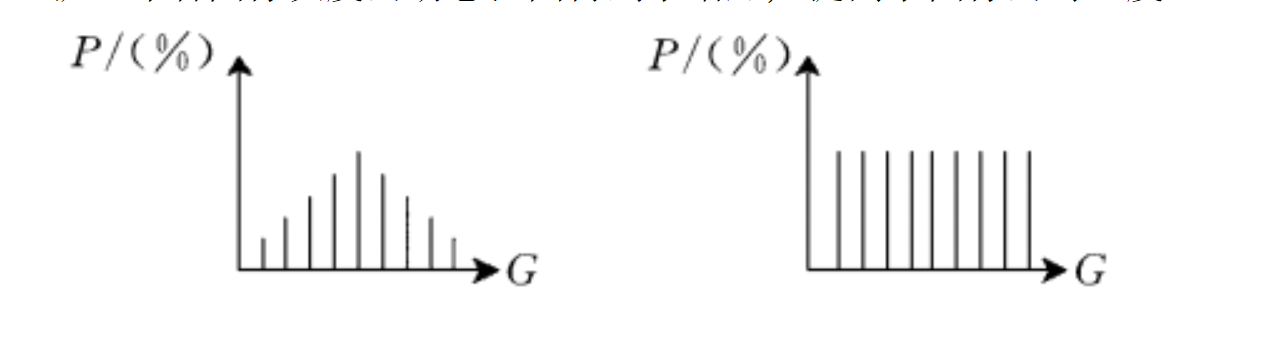
为了有利于数字图像处理, 引入离散形式。在离散形式下, 用[clip_image005](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183442705-602304938.png) 代表离散灰度级, 用[clip_image007](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183443393-1942824170.png) 代表[clip_image003[1]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183444159-152628690.png) , 并且下式成立:[clip_image009[12]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183444799-209130683.png)

其中, 0≤[clip_image005[1]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183445534-815614771.png)≤1, k=0, 1, 2, …, n-1。式中[clip_image011[12]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183446252-832684663.png) 为图像中出现[clip_image005[2]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183448002-539344618.png)这种灰度的像素数, n是图像中的像素总数, 而[clip_image013[12]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183448955-119550248.png)就是概率论中的频数。图像进行直方图均衡化的函数表达式为:

[clip_image015[12]](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183449768-424574320.png)

式中, k为灰度级数。相应的反变换为:

[clip_image017](http://images2015.cnblogs.com/blog/904258/201603/904258-20160305183451674-196081498.png)



### 三、源代码与分析

#### 1、图像对数运算主函数

int main(int argc, char\*argv[]){

    unsigned char\* file\_temp;

    FILE\* fpbmp;

    FILE\* fpout;

    fpbmp = openFile("input.bmp");          //读入图片

    fpout = writeFile("enhancement.bmp");   //建立输出文件

    readBMP(fpbmp);             //读取数据，判断文件是否为bmp图像

    addHeader(fpbmp, fpout);    //在输出文件中先写入标准文件头

    RGBtoYUV();         //将bmp文件中的rgb数据转换为yuv格式

    Enhancement();      //对数均衡化函数

    YUVtoRGB();         //将yuv数组转换为rgb

    outBMP(fpout);      //将更改后的rgb输出

    fclose(fpbmp);

    fclose(fpout);

}

#### 2、直方图均衡化主函数

int main(int argc, char\*argv[]){

    unsigned char\* file\_temp;

    FILE\* fpbmp;

    FILE\* fpout;

    fpbmp = openFile("input.bmp");          //读入图片

    fpout = writeFile("enhancement.bmp");   //建立输出文件

    readBMP(fpbmp);             //读取数据，判断文件是否为bmp图像

    addHeader(fpbmp, fpout);    //在输出文件中先写入标准文件头

    Equalization\_r();           //对rgb三个通道分别进行直方图均衡化

    Equalization\_g();

    Equalization\_b();

    outBMP(fpout);      //将更改后的rgb输出

    fclose(fpbmp);

    fclose(fpout);

}

#### 3、图像处理基本函数

（1）打开文件FILE\* openFile(char name[])

（2）新建文件FILE\* writeFile(char name[])

FILE\* openFile(char name[]) {

    FILE\* fpbmp;

    fpbmp = fopen(name, "rb");

    if (fpbmp == NULL) {

        printf("open failed(；′⌒`)\n");

        exit(1);

    }

    //test if it is a bmp

    unsigned short bfType = 0;

    fseek(fpbmp, 0L, SEEK\_SET);

    fread(&bfType, sizeof(char), 2, fpbmp);

    if (BM!=bfType) {

        printf("it is not a bmp(；′⌒`)\n");

        exit(1);

    }

    //get the header length

    fseek(fpbmp, 10L, SEEK\_SET);

    fread(&OffSet, sizeof(char), 4, fpbmp);

    //get width and height

    int size;

    fseek(fpbmp, 18L, SEEK\_SET);

    fread(&width, sizeof(char), 4, fpbmp);

    fseek(fpbmp, 2L, SEEK\_SET);

    fread(&size, sizeof(char), 4, fpbmp);

    fseek(fpbmp, 22L, SEEK\_SET);

    fread(&height, sizeof(char), 4, fpbmp);

    fseek(fpbmp, 0L, SEEK\_SET);

    return fpbmp;

}

FILE\* writeFile(char name[]) {

    FILE\* fpout;

    fpout = fopen(name, "wb+");

    if (fpout == NULL) {

        printf("open output file failed(；′⌒`)\n");

        exit(1);

    }

    fseek(fpout, 0L, SEEK\_SET);

    return fpout;

}

（3）bmp文件中数据储存读入void readBMP(FILE\* fpbmp)

（4）向bmp文件中写入数据void outBMP(FILE\* fpout)

void readBMP(FILE\* fpbmp) {

    int i = 0;

    int j = 0;

    unsigned char\* p = NULL;

    fseek(fpbmp, OffSet, SEEK\_SET);

    int s = ((24 \* width + 31) / 8) / 4 \* 4;

    p = malloc(s);

    for (i = 0; i < height; i++) {

        fread(p, 1, s, fpbmp);

        for (j = 0; j < width;j++) {

            R[height - 1 - i][j] = p[j \* 3 + 2];//一次读入rgb数据

            G[height - 1 - i][j] = p[j \* 3 + 1];//bmp文件中rgb值逆序储存

            B[height - 1 - i][j] = p[j \* 3];

            out\_R[height - 1 - i][j] = 255; //初始化三个通道均为255

            out\_G[height - 1 - i][j] = 255;

            out\_B[height - 1 - i][j] = 255;

        }

    }

}

void outBMP(FILE\* fpout) {

    int i = 0;

    int j = 0;

    unsigned char\* p\_out = NULL;

    fseek(fpout, OffSet, SEEK\_SET);

    int s = ((24 \* width + 31) / 8) / 4 \* 4;

    p\_out = malloc(s);

    for (i = 0; i < height; i++) {

        for (j = 0; j < width; j++) {

            p\_out[j \* 3 + 2] = out\_R[height - 1 - i][j];

            p\_out[j \* 3 + 1] = out\_G[height - 1 - i][j];

            p\_out[j \* 3] = out\_B[height - 1 - i][j];

        }

        fwrite(p\_out, 1, s, fpout);

    }

}

（4）向输出文件中写入bmp文件头

void addHeader(FILE\* fpbmp,FILE\* fpout) {

    unsigned char\* fp\_temp;

    fseek(fpbmp, 0L, SEEK\_SET);

    fseek(fpout, 0L, SEEK\_SET);

    fp\_temp = malloc(OffSet);

    fread(fp\_temp, 1, OffSet, fpbmp);

    fp\_temp[2] = (int)(height\*width \* 3 +OffSet);

    fp\_temp[18] = (int)width;

    fp\_temp[22] = (int)height;

    fp\_temp[34] = (int)(height\*((24 \* width / 8 + 3) / 4 \* 4));

    fwrite(fp\_temp, 1, OffSet, fpout);

}

（5）yuv与rgb各个通道间的线性变化

void RGBtoYUV() {

    int i, j;

    for (i = 0; i < height; i++) {

        for (j = 0; j < width; j++) {

            Y[i][j] = 0.299\*R[i][j] + 0.587\*G[i][j] + 0.114\*B[i][j];

            U[i][j] =-0.147\*R[i][j] - 0.289\*G[i][j] + 0.436\*B[i][j];

            V[i][j] = 0.615\*R[i][j] - 0.515\*G[i][j] - 0.100\*B[i][j];

        }

    }

}

void YUVtoRGB() {

    int i, j;

    for (i = 0; i < height; i++) {

        for (j = 0; j < width; j++) {

            out\_R[i][j] = out\_Y[i][j] + 1.140\*out\_V[i][j];

            out\_G[i][j] = out\_Y[i][j] - 0.394\*out\_U[i][j] -0.581\*out\_V[i][j];

            out\_B[i][j] = out\_Y[i][j] + 2.032\*out\_U[i][j];

        }

    }

}

#### 4、可见性增强的图像对数运算（Image logarithmic operation for visibility enhancement）处理函数

void Enhancement() {

    int max\_L = 0;

    int i, j;

    for (i = 0; i < height; i++) {

        for (j = 0; j < width; j++) {

            if (Y[i][j] > max\_L)max\_L = Y[i][j];

        }

    }

    for (i = 0; i < height; i++) {

        for (j = 0; j < width; j++) {

            out\_Y[i][j] = (unsigned char)255 \* (log10(Y[i][j] + 1)) / (log10(max\_L+1));

            out\_U[i][j] = U[i][j];

            out\_V[i][j] = V[i][j];

        }

    }

}

#### 5、直方图均衡（Histogram equalization）处理函数

（以r通道为例）

void Equalization\_r() {

    long n0[256] = { 0.0 };

    int i, j;

    for (i = 0;i < height; i++) {

        for (j = 0; j < width; j++) {

            n0[R[i][j]]++;

        }

    }

    double p0[256] = { 0.0 };

    for (i = 0; i < 256; i++)

        p0[i] = n0[i] / (1.0\*height\*width);

    double s0[256] = { 0.0 };

    s0[0] = p0[0];

    for (i = 1; i < 256; i++)

        s0[i] = s0[i - 1] + p0[i];

    double minmize = 1.0;

    int min\_number[256] = { 0 };

    double diff = 0.0;

    for (int j = 0; j < 256; j++)

    {

        //printf("!! == %f\n",s0[j]);

        diff = 0.0; minmize = 1.0;

        for (int i = 0; i < 256; i++)

        {

            double pix0 = 1.0\*i / 256;

            diff = fabs(s0[j] - pix0);

            //printf("%d == %f\n",i,diff);

            if (diff < minmize) {

                minmize = diff;

                min\_number[j] = i;//min\_number[j] --> i ;

            }

        }

        //printf("%d == %d\n",j,min\_number[j]);

    //  break;

    }

    for (int j = 0; j < height; j++)

    {

        for (int i = 0; i < width; i++)

        {

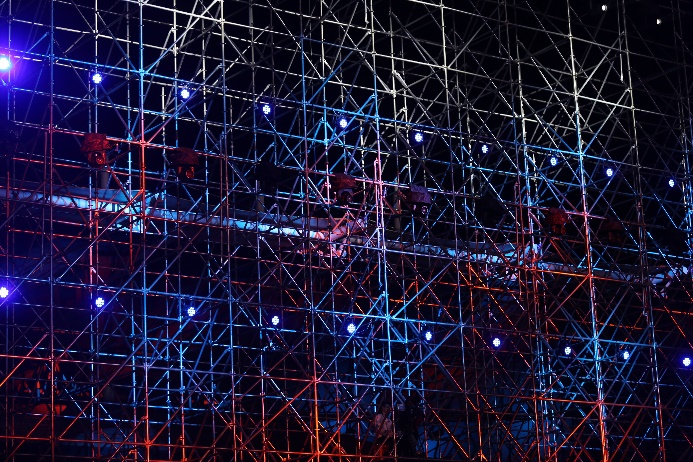
            out\_R[j][i] = min\_number[R[j][i]];

        }

    }

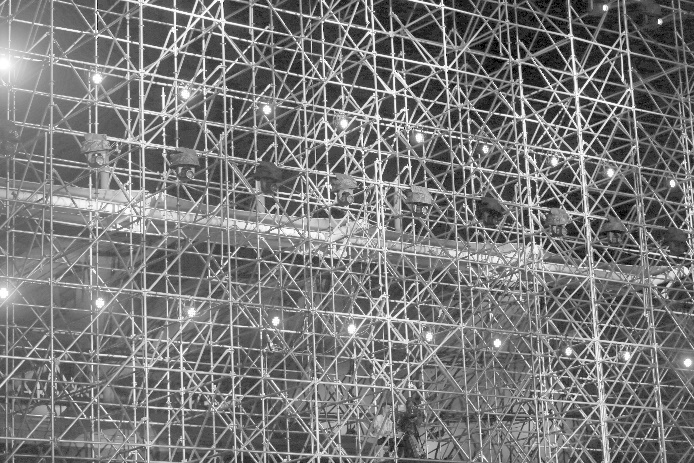
}

#### 6、结果分析

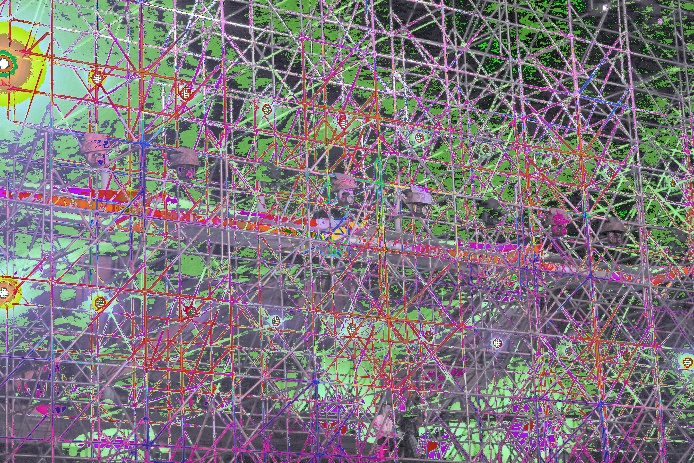
【原图】

【灰度通道对数处理结果】

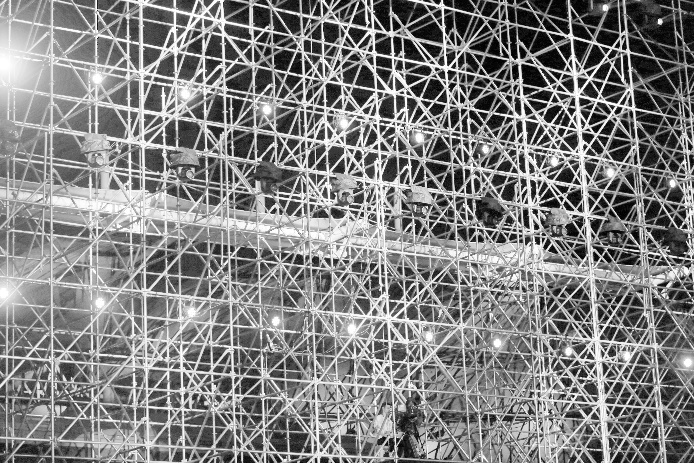
对灰度图进行对数处理后，暗处细节变得较为清晰

【灰度通道对数处理后输出】

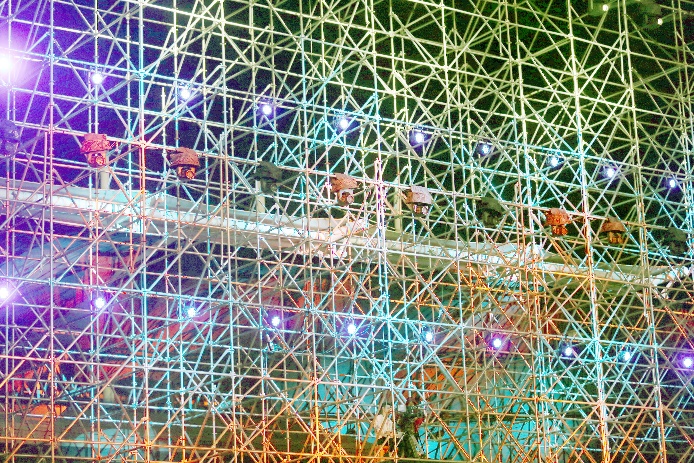
可以看出结构得到了保存，但图片颜色改变，可以看出yuv三组数值是线性相关的，不能随意改变其中一组

【灰度图直方均衡化】

直方图均衡化后的灰度图明显比对数处理后的灰度图更加清晰，对比度更高

【彩色图直方均衡化】

彩色图直方均衡化后展现出更多暗处色彩，将原图中隐藏的一些颜色暴露出来



### 四、心得体悟

1、不同的函数操作最好要选择合适的图像进行处理，才能得到较为明显的结果，本次实验中由于我曾选错照片导致输出图像变化不大，难以看出是否进行了处理，最好选择一些拍摄失败的照片，可能会得到意想不到的结果

2、对数处理和直方图均衡化都能较好地突出暗处细节，但容易导致明处一些细节地过曝，直方图均衡化能较好地避免这一情况的出现