

浙江大学实验报告

专业：计算机科学与技术

姓名：沈一芑

学号：322010827

日期：2023/11/16

课程名称：图像信息处理 指导老师：宋明黎 成绩：

实验名称：图像对数增强与直方图均衡化

1. 实验目的和要求

1.1. 实验目的

- 对图像进行增强处理

1.2. 实验要求

- 对图像进行对数增强
- 对图像进行直方图均衡化处理

2. 实验内容和原理

2.1. 实验原理

2.1.1. 对数处理

对图像进行对数化处理，以达到增强图像的效果。其中， L_w 为实际亮度， L_{max} 为图像最高亮度， L_d 为输出亮度：

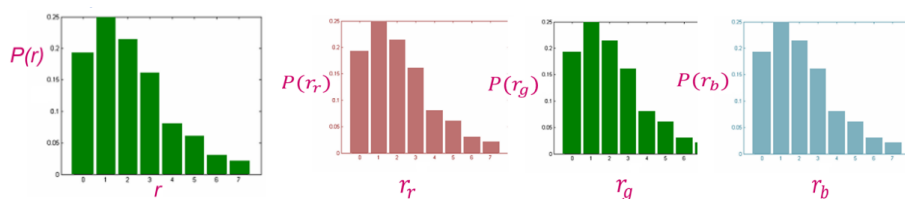
$$L_d = \frac{\log(L_w + 1)}{\log(L_{max} + 1)}$$

2.2. 直方图均衡化

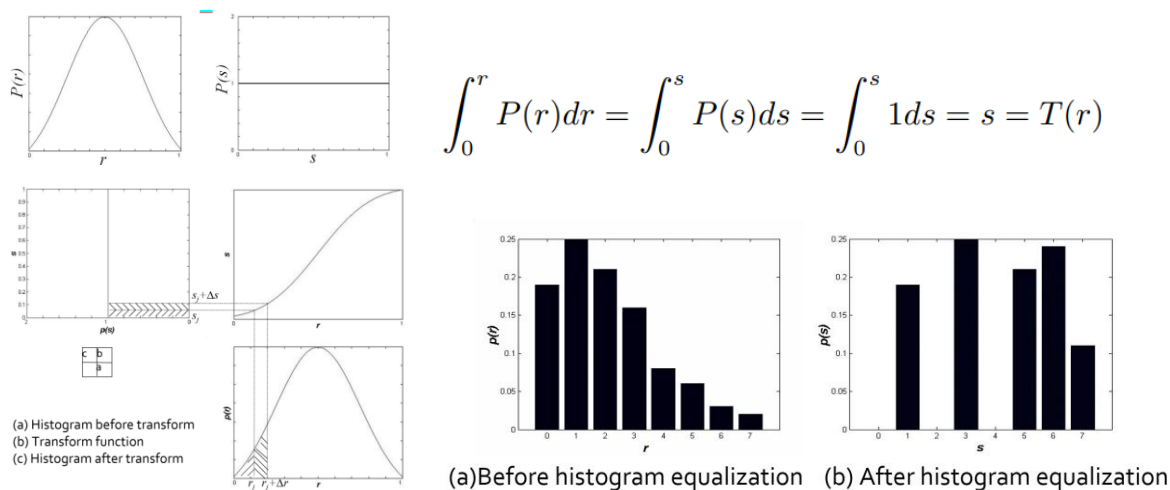
- 直方图：图像的灰度/R/G/B 值的统计图，横轴表示灰度/色彩等级，纵轴表示该灰度/色彩等级在整张图像中的占比。
- 设 r_k 为第 k 个灰度级， n_k 为灰度级 r_k 所占有的像素数，则使用概率密度函数对直方图进行归一化：

$$P(r_k) = \frac{n_k}{n} \quad \sum_{k=0}^{L-1} P(r_k) = 1$$

- 如下图所示分别为灰度直方图与 R/G/B 彩色直方图：



- 直方图均衡化则是通过改变图像的直方图来改变图像中各像素的等级，通过把原始图像的直方图变换为均匀分布（均衡）的形式，增加了像素之间灰度/色彩等级差别的动态范围，从而达到增强图像整体对比度的效果。
- 设 s 为均衡化后的灰度级。则需构建 r 与 s 的映射关系 $s=T(r)$ ：



- 由于 s_k 不可能直接等于八级灰度值中的某一级，因此需要就近归入某一个灰度级中。这样，相邻的多个 s_k 就可能落入同一个灰度级，需要将处于同一个灰度级的像素个数累加。因此，离散灰度直方图均衡化操作以后，每个灰度级处的概率密度（或像素个数）并不完全一样。

2.3. 实验内容

- 使用 lab1 中方法，读入 BMP 图片，并将 RGB 色彩空间转为 YUV，进而生成灰度图。
- 使用对数处理公式，将图像对数化增强。
- 仿照上述步骤，对图像进行直方图均衡化处理。

3. 实验步骤与分析

3.1. 准备工作

- 定义结构体并声明变量如下；每个变量对应一张 BMP 图片：

```
• typedef struct{
•     BMFH bmfh;
•     BMIH bmiH;
•     RGBQUAD rgbquad[256];
•     BYTE *data;
•     int LineBytes, WidthBMP, HeightBMP, SizeImageBMP;
•     //LineBytes is to compliment the width given that bytes mod 4 == 0
•     //size of image part, instead of whole bmp
• }IMAGE;
•
• IMAGE input, gray, log_gray, log_color, equalize_gray, equalize_color;
```

3.2. 对数处理

- 将 gray 拷贝至 log_gray。
- 遍历整张灰度图，更新最大灰度值 l_max。
- 根据公式，计算对数处理后的灰度值，并写入图像。

```
1. void do_log_gray(IMAGE *input, IMAGE *output){
2.
3.     int l_max=0;
4.
5.     copyBMP(input, output);
6.
7.     for(int i=0;i<output->HeightBMP;i++){
8.         for(int j=0;j<output->WidthBMP;j++){
9.             if(output->data[output->LineBytes*i+j]>l_max)
10.                 l_max=output->data[output->LineBytes*i+j];
11.         }
12.     }
13.     for(int i=0;i<output->HeightBMP;i++){
14.         for(int j=0;j<output->WidthBMP;j++){
15.             double x=log(output->data[output->LineBytes*i+j]+1)/log(l_max+1);
16.             output->data[output->LineBytes*i+j]=(int)(x*255);
17.         }
18.     }
19.
20. }
```

- 对 log_color, 则用对数处理后的灰度值 Y 与 U, V 计算 B, G, R, 重新写入图像。

```
1. for(int i=0;i<output->HeightBMP;i++){
2.     for(int j=0;j<output->WidthBMP;j++){
3.         double R, G, B, Y, U, V;
4.         B=rgb->data[rgb->LineBytes*i+j*3];
5.         G=rgb->data[rgb->LineBytes*i+j*3+1];
6.         R=rgb->data[rgb->LineBytes*i+j*3+2];
7.         Y=0.299*R+0.587*G+0.114*B;           //calculate Y,U,V
8.         U=-0.147*R-0.289*G+0.435*B;
9.         V=0.615*R-0.515*G-0.100*B;
10.
11.         Y=(int)(log(Y+1)/log(l_max+1)*255);
12.         //change luminance
13.         R=Y+1.4075*V;                         //calculate R,G,B
14.         G=Y-0.3455*U-0.7169*V;
15.         B=Y+1.779*U;
16.
17.         if(R>255)R=255;
18.         if(R<0)R=0;
19.         if(G>255)G=255;
20.         if(G<0)G=0;
21.         if(B>255)B=255;
22.         if(B<0)B=0;
23.
24.         output->data[output->LineBytes*i+j*3]=B; //write in data
25.         output->data[output->LineBytes*i+j*3+1]=G;
26.         output->data[output->LineBytes*i+j*3+2]=R;
27.     }
28. }
```

3.3. 图像均衡化

3.3.1. 灰度图均衡化

- 将 gray 拷贝至 equalization_gray。
- 设定灰度等级 degree。
- 统计各灰度等级中像素点的数量，记录于 prob[]。
- 将 prob[] 从离散密度函数更新为分布函数 (prob[i]+=prob[i-1])。
- 遍历灰度图，将各像素点灰度值依 prob[] 分布函数归类，按照相应灰度等级赋值。

```
1. void do_equalize_gray(IMAGE *input, IMAGE *output){
2.     copyBMP(input, output);
3.     int degree=16;
4.     int divide=256/degree;
5.     double prob[256];
6.     for(int i=0;i<degree;i++)    prob[i]=0;
7.     for(int i=0;i<output->HeightBMP;i++)
8.         for(int j=0;j<output->WidthBMP;j++)
9.             prob[output->data[output->LineBytes*i+j]/divide]++;
10.    for(int
        i=0;i<degree;i++)    prob[i]/=(output->HeightBMP*output->WidthBMP);
11.    for(int i=1;i<degree;i++)    prob[i]+=prob[i-1];
12.    for(int i=0;i<output->HeightBMP;i++)
13.        for(int j=0;j<output->WidthBMP;j++)
14.            output->data[output->LineBytes*i+j]=prob[output->data[output->LineBytes*i+j]/divide]*256;
15.}
```

3.3.2. 彩色图均衡化

- 将 input 拷贝至 equalization_color。
- 设定 B, G, R 色彩等级分别为 degree[0, 1, 2]。
- 统计各色彩等级中像素点的数量，记录于 prob[0, 1, 2]。
- 将 prob[0, 1, 2] 从离散密度函数更新为分布函数 (prob[][i]+=prob[][i-1])。
- 遍历彩色图，将各像素点 B, G, R 值依 prob[0, 1, 2] 分布函数归类，按照相应的 B, G, R 色彩等级赋值。

```

1. void do_equalize_color(IMAGE *input, IMAGE *output){
2.
3.     memcpy(&(output->bmfh), &(input->bmfh), sizeof(BMFH));
4.     memcpy(&(output->bmih), &(input->bmih), sizeof(BMIH));
5.     output->WidthBMP=input->WidthBMP;
6.     output->HeightBMP=input->HeightBMP;
7.     output->LineBytes=input->LineBytes;
8.     output->SizeImageBMP=output->bmih.biSizeImage;
9.     output->data=new BYTE[output->bmih.biSizeImage];
10.    int degree[3];                                //in the order of B G R
11.    degree[0]=64;
12.    degree[1]=64;
13.    degree[2]=64;
14.    int divide[3];
15.    divide[0]=256/degree[0], divide[1]=256/degree[1],
        divide[2]=256/degree[2];
16.    double prob[3][256];
17.    for(int i=0;i<3;i++)
18.        for(int j=0;j<degree[i];j++)
19.            prob[i][j]=0;
20.    for(int i=0;i<input->HeightBMP;i++)
21.        for(int j=0;j<input->WidthBMP;j++)
22.            prob[0][input->data[input->LineBytes*i+j*3]/divide[0]]++,
23.            prob[1][input->data[input->LineBytes*i+j*3+1]/divide[1]]++,
24.            prob[2][input->data[input->LineBytes*i+j*3+2]/divide[2]]++;
25.    for(int i=0;i<3;i++)
26.        for(int j=0;j<degree[i];j++)
27.            prob[i][j]/=(output->HeightBMP*output->WidthBMP);
28.    for(int i=0;i<3;i++)
29.        for(int j=1;j<degree[i];j++)
30.            prob[i][j]+=prob[i][j-1];
31.    for(int i=0;i<output->HeightBMP;i++)
32.        for(int j=0;j<output->WidthBMP;j++)
33.            output->data[output->LineBytes*i+j*3]=prob[0][input->data[output->
                LineBytes*i+j*3]/divide[0]]*256,
34.            output->data[output->LineBytes*i+j*3+1]=prob[1][input->data[output->
                LineBytes*i+j*3+1]/divide[1]]*256,
35.            output->data[output->LineBytes*i+j*3+2]=prob[2][input->data[output->
                LineBytes*i+j*3+2]/divide[2]]*256;
36.    return;
37.}







```

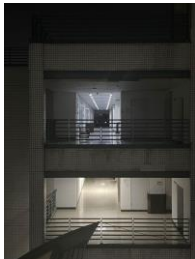






4. 实验结果展示

4.1. 对数化操作

- 展示图像对数化操作与 256 灰度等级/色彩等级的直方图均衡化操作效果：

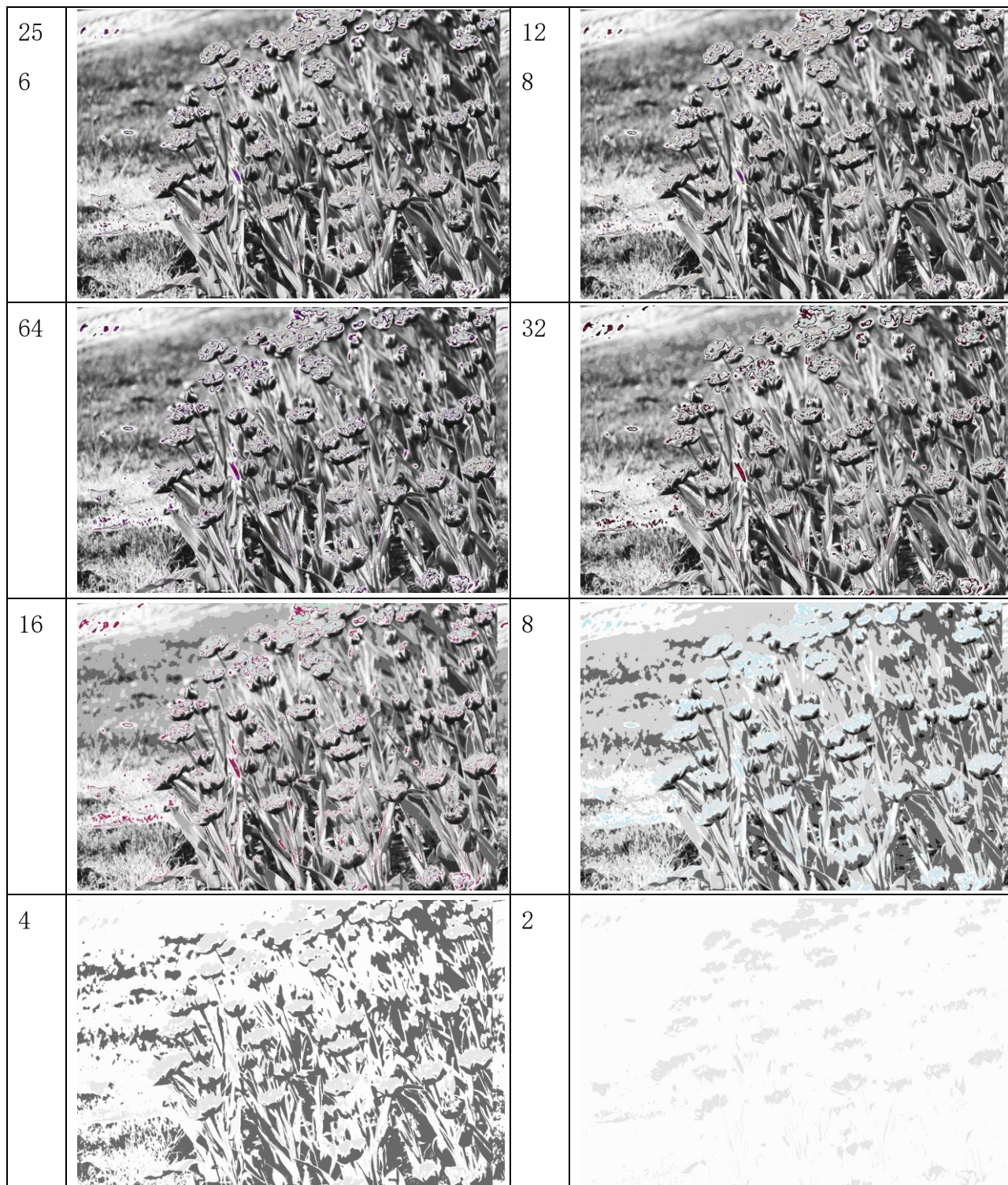
Origin	Log_color	equal_color	gray	Log_gray	Equal_gray
					

origin		Gray	
Log_color		Log_gray	
Equal_color		Equal_gray	

Origin	Log_color	equal_color	gray	Log_gray	Equal_gray
					





4.2. 灰度图直方图均衡化

- 对比不同灰度等级的直方图均衡化：



4.3. 彩色图直方图均衡化：




- 首先将 B, G, R 三个通道的色彩等级捆绑，一同更改。

12 8		6 4	
32		2 0	

再降低色彩等级数量后，图片无法输出（低于 16 位后）。

- 其次将 B, G, R 三个通道的色彩等级分开，分别更改。

In the order of B/G/R

20 256 256		256 20 256		256 256 12	
20 256 256		256 20 256		20 20 256	

5. 心得体会

通过这次实验我了解了图像视觉效果增强的相关方法：对数化操作与直方图均衡化，并通过 c++ 实现了对数化操作和灰度图/彩色图直方图均衡化。

通过比对实验结果，可以得出不同操作的基本处理增强特点：对数化操作整体而言会使图像向亮度更高的方向整体平滑处理，直观效果为图像好像增雾、虚化。而直方图均衡化则是使图像整体更加平均，高亮图会适当降低亮度，暗图则会一定程度拉高曝光。与此同时，均衡化由于均衡色彩等级，也会起到增加对比度的效果。

此外，对于直方图均衡化，通过改变灰度/色彩等级个数，可以实现不同效果。选取 256 灰度/色彩等级进行均衡会十分显著地提高对比度。对于灰度图，随着灰度等级数量的下降，对比度不断增强，而图像的复杂程度也不断下降，图像的主体更容易被抓住。而当灰度等级降低至 2 时，若选取 0 和 256 作为两个等级各自的灰度取值，则生成图片效果和以 128 为阈值的灰度图二值化效果近似。对于彩色图，随着色彩等级数量的下降，图片背景虚化效果加重，噪点一定程度上先聚集成片、更明显，再融成团块，消失不见。或许直方图均衡化降低色彩等级数量可以以模糊为代价换取降噪效果。

同时，分别改变 B\G\R 三个色彩通道各自的色彩等级数，可以改变画面的色调色温。直观来看，拉高 R 降低 BG 使图像偏向冷色温，拉高 BG 降低 R 使图像偏向暖色温。

对于本次实验，存在两个问题。其一：对于亮度/暗度过大的图片，在 \log 对数处理后会呈现部分区域彩色，怀疑是溢出的缘故。但程序中写了溢出保护，故而不清楚原因为何。其二：灰度图的色彩等级数量可以无限下降至 1（全白/黑），但彩色图的色彩等级下降至 16 后便无法继续生成可读 BMP。并不是很理解原因。最后，希望能获得一些渠道，了解色温色调和色彩等级数量的理论关系。

综上，本次实验收获满满，希望继续探索对图像的处理。