

图像可视性对数增强和直方图均衡化

实 验 报 告

姓名：欧翌昕

专业：软件工程

学号：3190104783

课程名称：图像信息处理

指导老师：宋明黎

2020~2021秋冬学期 2020 年 11 月 18 日

目录

[1 实验目的和要求 2](#_Toc56976963)

[2 实验内容和原理 2](#_Toc56976964)

[2.1 可视性对数增强 2](#_Toc56976965)

[2.2 直方图 2](#_Toc56976966)

[2.3 直方图均衡化 3](#_Toc56976967)

[3 实验步骤与分析 4](#_Toc56976968)

[4 实验结果 8](#_Toc56976969)

[4.1 原始图 8](#_Toc56976970)

[4.2 可视性对数增强结果图 9](#_Toc56976971)

[4.3 灰度图 10](#_Toc56976972)

[4.4在Y通道上进行直方图均衡化灰度图结果 11](#_Toc56976973)

[4.5 在Y通道上进行直方图均衡化彩色图结果 12](#_Toc56976974)

[4.6 在RGB通道上进行直方图均衡化彩色图结果 13](#_Toc56976975)

[5 心得体会 13](#_Toc56976976)

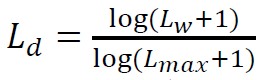
# 1 实验目的和要求

1. 对BMP灰度图像进行可视性对数增强
2. 对BMP灰度图像进行直方图均衡化

# 2 实验内容和原理

## 2.1 可视性对数增强

为了增强图像的可视信息，对图像中的像素进行基于对数的操作。



Ld是显示亮度（我们所要求的值），Lw是真实世界的亮度（图片的当前值），Lmax是场景中的最亮值（图片的最大值）。

这个映射能够确保不管场景的动态范围是怎么样的，其最大值都能映射到1（白），其他的值能够比较平滑地变化。

## 2.2 直方图

灰度直方图是一类统计图形，它表示一幅图像中各个灰度等级的像素个数在像素总数中所占的比重。

彩色直方图是一类统计图形，它表示一幅图像中R、G、B通道上各个灰度等级的像素个数在像素总数中所占的比重。

直方图是空间域处理技术的基础，它反映图像灰度的分布规律，但不能体现图像中的细节变化情况。对于一幅给定的图像，其直方图是唯一的。不同的图像可以对应相同的直方图。

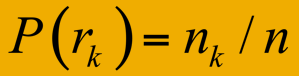
对直方图进行操作能有效地用于图像增强、压缩和分割, 他们是图像处理的一个实用手段。

## 2.3 直方图均衡化

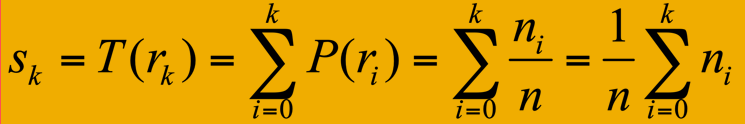
将原图像的非均匀分布的直方图通过变换函数T修正为均匀分布的直方图，然后按均衡直方图修正原图像。

图像均衡化处理后，图像的直方图是平直的，即各灰度级具有相同的出现频数。

设一幅图像的像素总数为n，分L个灰度级，nk为第k个灰度级出现的像素数，则第k个灰度级出现的概率为：



离散灰度直方图均衡化的转换公式为：



对于原直方图中的任意一个灰度级*rk*，只需将灰度级为[0,*rk*]以内的所有像素个数的和除以图像的像素总数，就可以得到转换之后的对应灰度级*sk*。

# 3 实验步骤与分析

遍历所有像素寻找最亮值：

    int maxY = 0; //最亮值

    int i, j;

    for (i = 0; i < h; i++)

    {

        for (j = 0; j < w\*3-skip; j += 3)

        {

            int t = i\*bytesPerLine + j;

            int B = data[t];

            int G = data[t+1];

            int R = data[t+2];

            int Y = 0.299\*R + 0.587\*G + 0.114\*B;

            if (Y < 0 ) Y = 0;

            if (Y > 255) Y = 255;

            if (Y > maxY) maxY = Y;

        }

    }

通过公式将真实亮度转换为显示亮度：

    for (i = 0; i < h; i++)

    {

        for (j = 0; j < w\*3-skip; j += 3)

        {

            int t = i\*bytesPerLine + j;

            int B = data[t];

            int G = data[t+1];

            int R = data[t+2];

            int Y = 0.299\*R + 0.587\*G + 0.114\*B;  //真实亮度

            if (Y < 0 ) Y = 0;

            if (Y > 255) Y = 255;

            int U = -0.147\*R - 0.289\*G + 0.435\*B;

            int V = 0.615\*R - 0.515\*G - 0.100\*B;

            int d = 255\*log10(Y+1)/log10(maxY+1);  //显示亮度

            B = d + 2.032\*U;

            if (B < 0) B = 0;

            if (B > 255) B = 255;

            G = d -0.394\*U - 0.581\*V;

            if (G < 0) G = 0;

            if (G > 255) G = 255;

            R = d + 1.140\*V;

            if (R < 0) R = 0;

            if (R > 255) R = 255;

            data[t] = (BYTE)B;

            data[t+1] = (BYTE)G;

            data[t+2] = (BYTE)R;

        }

    }

遍历所有像素寻找最亮值及最暗值，并统计各灰度值的个数：

    int maxY = 0;

    int minY = 255;

    int num[256];  //各灰度值的个数

    memset(num, 0, sizeof(int)\*256);

    int i, j;

    for (i = 0; i < h; i++)

    {

        for (j = 0; j < w\*3-skip; j += 3)

        {

            int t = i\*bytesPerLine + j;

            int B = data[t];

            int G = data[t+1];

            int R = data[t+2];

            int Y = 0.299\*R + 0.587\*G + 0.114\*B;

            if (Y < 0 ) Y = 0;

            if (Y > 255) Y = 255;

            if (Y > maxY) maxY = Y;

            if (Y < minY) minY = Y;

            num[Y]++;

        }

    }

通过公式在Y通道上进行直方图均衡化：

    double histogram[256];

    memset(histogram, 0, sizeof(double)\*256);

    for (i = 0; i < 256; i++)

    {

        for (j = 0; j <= i; j++)

        {

            histogram[i] += 1.0\*num[j]/(w\*h);

        }

    }

    for (i = 0; i < h; i++)

    {

        for (j = 0; j < w\*3-skip; j += 3)

        {

            int t = i\*bytesPerLine + j;

            int B = data[t];

            int G = data[t+1];

            int R = data[t+2];

            int Y = 0.299\*R + 0.587\*G + 0.114\*B;

            if (Y < 0 ) Y = 0;

            if (Y > 255) Y = 255;

            Y = minY + (maxY-minY)\*histogram[Y];

            data[t] = data[t+1] = data[t+2] = (BYTE)Y;

        }

    }

同理在R、G、B通道上分别进行直方图均衡化：

int i, j;

    for (i = 0; i < h; i++)

    {

        for (j = 0; j < w\*3-skip; j += 3)

        {

            int t = i\*bytesPerLine + j;

            int B = data[t];

            int G = data[t+1];

            int R = data[t+2];

            if (B > maxB) maxB = B;

            if (B < minB) minB = B;

            if (G > maxG) maxG = G;

            if (G < minG) minG = G;

            if (R > maxR) maxR = R;

            if (R < minR) minR = R;

            numB[B]++;

            numG[G]++;

            numR[R]++;

        }

    }

    double histogramB[256];

    memset(histogramB, 0, sizeof(double)\*256);

    double histogramG[256];

    memset(histogramG, 0, sizeof(double)\*256);

    double histogramR[256];

    memset(histogramR, 0, sizeof(double)\*256);

    for (i = 0; i < 256; i++)

    {

        for (j = 0; j <= i; j++)

        {

            histogramB[i] += 1.0\*numB[j]/(w\*h);

            histogramG[i] += 1.0\*numG[j]/(w\*h);

            histogramR[i] += 1.0\*numR[j]/(w\*h);

        }

    }

    for (i = 0; i < h; i++)

    {

        for (j = 0; j < w\*3-skip; j += 3)

        {

            int t = i\*bytesPerLine + j;

            int B = data[t];

            int G = data[t+1];

            int R = data[t+2];

            B = minB + (maxB-minB)\*histogramB[B];

            G = minG + (maxG-minG)\*histogramG[G];

            R = minR + (maxR-minR)\*histogramR[R];

            data[t] = (BYTE)B;

            data[t+1] = (BYTE)G;

            data[t+2] = (BYTE)R;

        }

    }

# 4 实验结果

## 4.1 原始图



## 4.2 可视性对数增强结果图



## 4.3 灰度图



## 4.4在Y通道上进行直方图均衡化灰度图结果



## 4.5 在Y通道上进行直方图均衡化彩色图结果



## 4.6 在RGB通道上进行直方图均衡化彩色图结果



# 5 心得体会

由实验结果可以看出，对RGB通道独立进行直方图均衡化操作确实能得到效果，但图像被破坏的程度高。若将图像先转换为YUV通道，只对Y通道进行直方图均衡化操作，得到的效果有明显改善。但老师曾讲，YUV通道实际上并不是相互独立的，在LAB通道上进行操作也许会得到更好的效果。

通过本次实验，我对BMP文件结构有了更深了解的同时还巩固了老师上课时讲到的一些操作，同时对本次实验的深入思考也让我收获到很多知识，可视性对数增强的用途，以及YUV通道实际上不独立等等，总的来说此次实验基本达到预期效果。