

数字图像处理第一次作业

赵子瑞

自动化钱61班

2019年3月5日

摘要

本文是数字图像与视频处理的第一次作业，通过理论学习和代码实践，完成了以下任务：

首先介绍了bmp图像格式，从存储算法，文件格式和像素存储等方面进行介绍，并且按照给定的图像为例进行说明，通过解析图片和存储结构，来得到例图的详细信息。

其次，本文将对lena图片进行一系列操作，对lena的照片进行灰度递减显示，计算均值方差，利用多种方法进行插值，同时对lena和elain的图像进行切割、旋转和插值。从而加深我们对图像处理基本知识的掌握

关键词： bmp图像格式 图像处理 灰度降级 图像均值方差计算 图像插值 图像旋转 图像切割

目录

1	项目任务	3
2	bmp图像简介	3
2.1	文件格式	4
2.1.1	像素数组	5
2.1.2	压缩	5
2.1.3	像素存储	5
2.2	bmp图像举例	6
3	bmp图像处理	7
3.1	<i>lena</i> 图像的灰度递减	7
3.2	<i>lena</i> 图像的均值方差	7
3.3	<i>lena</i> 图像的插值	9
3.3.1	最邻近插值	10
3.3.2	双线性插值	10
3.3.3	双三次插值	10
3.4	<i>lena</i> 图像和 <i>elain</i> 图像的仿射变换和插值处理	10
3.4.1	水平偏移	11
3.4.2	旋转	12
4	小结	12

1 项目任务

本次数字图像与食品处理主要是对课堂上介绍的算法进行实践，进而加深对图像处理的基本知识的掌握情况。

- (1) BMP图像格式简介，以7.bmp为例说明；
- (2) 把lena 512×512 的图像灰度从1到8逐级递减显示；
- (3) 计算lena图像的均值方差
- (4) 把lena图像用近邻、双线性和双三次插值法放大到 2048×2048 ；
- (5) 把lena和elain图像分别进行水平剪切（参数可设置为1.5，或者自行选择）和旋转30度，并采用用近邻、双线性和双三次插值法放大到 2048×2048 ；

2 bmp图像简介

BMP取自位图**BitMaP**的缩写，也称为DIB（与设备无关的位图），是一种与显示器无关的位图数字图像文件格式。常见于微软视窗和OS/2操作系统，Windows GDI API内部使用的DIB数据结构与 BMP 文件格式几乎相同。

图像通常保存的颜色深度有2（1位）、16（4位）、256（8位）、65536（16位）和1670万（24位）种颜色（其中位是表示每点所用的数据位）。8位图像可以是索引彩色图像外，也可以是灰阶图像。表示透明的alpha通道也可以保存在一个类似于灰阶图像的独立文件中。带有集成的alpha通道的32位版本已经随着Windows XP出现，它在视窗的登录和主题系统中都有使用^[1]。

从开发人员的角度来看，位图由一组指定或包含以下元素的结构组成：

- (1) 一个标题，描述创建像素矩形的设备的分辨率，矩形的尺寸，位数组的大小等。
- (2) 逻辑调色板。
- (3) 一个位数组，用于定义图像中的像素与逻辑调色板中的条目之间的关系。

位图大小与其包含的图像类型有关。位图图像可以是单色或彩色。在图像中，每个像素对应于位图中的一个或多个位。单色图像具有每像素1比特（bpp）的比率。彩色成像更复杂。位图可以显示的颜色数等于两个每像素位数。因此，256色位图需要8bpp（ $2^8 = 256$ ）。

控制面板应用程序是使用位图的应用程序的示例。为桌面选择背景（或壁纸）时，实际上选择了一个位图，系统使用该位图来绘制桌面背景。系统通过在桌面上重复绘制32

x 32像素的图案来创建所选的背景图案。

图1显示了开发人员对*Redbrick.bmp*文件中位图的透视图。它显示了一个调色板数组，一个32 x 32像素的矩形，以及将调色板中的颜色映射到矩形中的像素的索引数组。

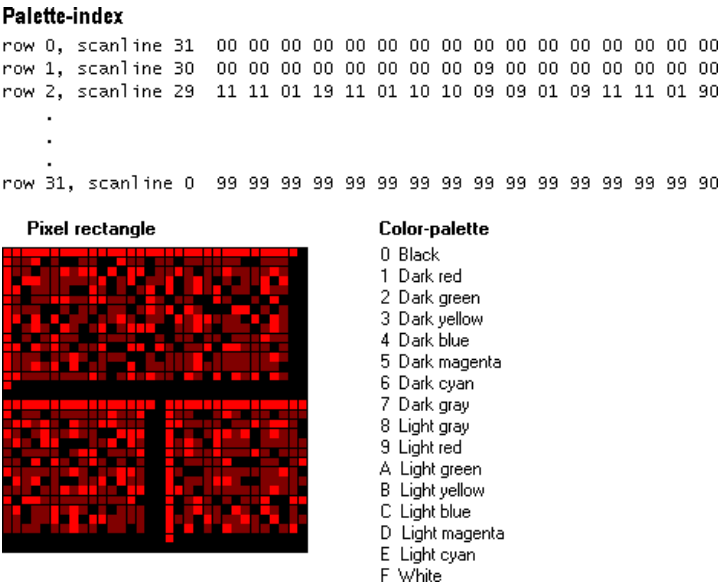


图 1: Redbrick.bmp文件

在前面的示例中，使用16种颜色的调色板在VGA显示设备上创建像素矩形。16色调色板需要4位索引; 因此，将调色板颜色映射到像素颜色的数组也由4位索引组成。

2.1 文件格式

位图图像文件由若干大小固定（文件头）和大小可变的结构体按一定的顺序构成。典型的BMP图像文件由四部分组成：

- （1）位图头文件数据结构，它包含BMP图像文件的类型、显示内容等信息；
- （2）位图信息数据结构，它包含有BMP图像的宽、高、压缩方法，以及定义颜色等信息；
- （3）调色板，这个部分是可选的，有些位图需要调色板，有些位图，比如真彩色图（24位的BMP）就不需要调色板；
- （4）位图数据，这部分的内容根据BMP位图使用的位数不同而不同，在24位图中直接使用RGB，而其他的小于24位的使用调色板中颜色索引值。

表示位图中像素的比特是以行为单位对齐存储的，每一行的大小都向上取整为4字节（32位DWORD）的倍数。如果图像的高度大于1，多个经过填充实现对齐的行就形成了

像素数组。

完整存储的一行像素所需的字节数可以通过公式1计算：

$$RowSize = \left\lceil \frac{BitsPerPixel \cdot ImageWidth + 31}{32} \right\rceil \cdot 4 \quad (1)$$

其中 $ImageWidth$ 以像素为单位

2.1.1 像素数组

这部分逐个像素表示图像。每个像素使用一个或者多个字节表示。通常，像素是从下到上、从左到右保存的。但如果使用的不是BITMAPCOREHEADER，那么未压缩的Windows位图还可以从上到下存储，此时图像高度为负值。

每一行的末尾通过填充若干字节的数据（并不一定为0）使该行的长度为4字节的倍数。像素数组读入内存后，每一行的起始地址必须为4的倍数。这个限制仅针对内存中的像素数组，针对存储时，仅要求每一行的大小为4字节的倍数，对文件的偏移没有限制。例如：对于24位色的位图，如果它的宽度为1像素，那么除了每一行的数据（蓝、绿、红）需要占3字节外，还会填充1字节；而如果宽为2像素，则需要2字节的填充；宽为3像素时，需要3字节填充；宽为4像素时则不需要填充。

图像相同的条件下，位图图像文件通常比使用其它压缩算法的图像文件大很多。

2.1.2 压缩

索引色图像可以使用4位或8位RLE或霍夫曼1D算法压缩。OS/2 BITMAPCOREHEADER2 24位色图像则可以使用24位RLE算法压缩。16位色与32位色图像始终为未压缩数据。如果需要，任何色深的图像都可以以未压缩形式存储。

2.1.3 像素存储

无论是磁盘上的位图文件还是内存中的位图图像，像素都由一组位（英语：bit）表示。

(1) 每像素占1位（色深为1位，1bpp）的格式支持2种不同颜色。像素值直接对应一个位的值，最左像素对应第一个字节的最高位。使用该位的值用来对色表的索引：为0表示色表中的第一项，为1表示色表中的第二项（即最后一项）。

(2) 每像素占2位（色深为2位，2bpp）的格式支持4种不同颜色。每个字节对应4个像素，最左像素为最高的两位（仅在Windows CE中有效）。需要使用像素值来对一张含有4个颜色值的色表进行索引。

(3) 每像素占4位（色深为4位，4bpp）的格式支持16种不同的颜色。每个字节对应2个像素，最左像素为最高的四位。需要使用像素值来对一张含有16个颜色值的色表进行索引。

(4) 每像素占8位（色深为8位，8bpp）的格式支持256种不同的颜色。每个字节对应1个像素。需要使用像素值来对一张含有256个颜色值的色表进行索引。

(5) 每像素占16位（色深为16位，16bpp）的格式支持65536种不同的颜色，每2个字节（byte）对应一个像素。该像素的不透明度（英语：alpha）、红、绿、蓝采样值即存储在该2个字节中。

(6) 每像素占24位（色深为24位，24bpp）的格式支持16777216种不同的颜色，每3个字节对应一个像素。

(7) 每像素占32位（色深为32位，32bpp）的格式支持4294967296种不同的颜色，每4个字节对应一个像素。

为了区分一个颜色值中的哪些位表示哪种采样值，DIB头给出了一套默认规则，同时也允许使用BITFIELDS将某组位指定为像素中的某个通道。

2.2 bmp图像举例

利用Python和OpenCV将图片进行解析，得到了一个 $7 \times 7 \times 3$ 的一个张量，这反映了这个图像是 7×7 的图像，拥有三个颜色（R,G,B）的通道，其中三个通道的数据完全相同，所以我们可以用下面的矩阵表示。

$$\begin{pmatrix} 82 & 82 & 73 & 59 & 55 & 80 & 90 \\ 97 & 89 & 90 & 95 & 71 & 40 & 69 \\ 104 & 71 & 63 & 105 & 93 & 76 & 42 \\ 88 & 75 & 85 & 101 & 90 & 91 & 70 \\ 97 & 92 & 91 & 99 & 72 & 71 & 82 \\ 98 & 101 & 102 & 86 & 69 & 71 & 95 \\ 103 & 99 & 100 & 84 & 86 & 98 & 98 \end{pmatrix}$$

同时直接读取7.bmp的图像结构信息，可以得到以下信息：

- (1) 1,2位，为“BM”，表示是Windows位图。
- (2) 一个4字节整数，换算成十进制为1134，表示位图大小；
- (3) 一个4字节整数，保留位，始终为0；
- (4) 一个4字节整数，换算为十进制是1078，表示实际图像的偏移量；
- (5) 一个4字节整数，换算为十进制是40，表示Header的字节数；
- (6) 一个4字节整数，十进制为7，表示图像宽度；
- (7) 一个4字节整数，十进制为7，表示图像高度；
- (8) 一个2字节整数，始终为1；
- (9) 一个2字节整数，十进制为8，表示颜色位数。

由此，可以得到这个BMP图片的基本信息。

3 bmp图像处理

本节将阐述我对图像lena.bmp和elain.bmp进行相应的处理的过程和算法。这里为了未来的作业的处理方便，我创建了一个用来处理数字图像的工具包“CVpythontoolbox”，这里包含了所有作业中布置的任务，包括了图像的读取，图像信息的分析，以及其他各种计算。同时也调用了部分opencv的库，本着学习和实践的目的，工具包中的核心算法和处理过程均为自行实现。这里采用了Python作为主体语言，主要是考虑到为了方便未来可能会采用深度学习的相关工具包。

3.1 lena图像的灰度递减

首先我将对lena.bmp图像进行灰度递减的操作。这里将lena.bmp的灰度值的bit位进行逐级递减的操作，从8位逐级降到1位，经过灰度降级操作，灰度的层级丰富程度也由256种降到了2种。我们的处理结果可以如下图2所示。

在这里我们运用了比较简单的降维方式，采用了手写函数实现的方式，这里对算法1进行简要介绍。

3.2 lena图像的均值方差

图像的均值方差计算很容易，通过numpy的矩阵运算方法，可以得到lena.bmp和elain.bmp的



图 2: 图像灰度递减结果

```

Initialization: Read image information;
Get Paramter: Greyscale_Reduce_Index;
while  $i \leq \text{Width\_of\_image}$  do
    while  $j \leq \text{Height\_of\_image}$  do
        while  $k \leq \text{Color\_Tunnel}$  do
             $\text{Img\_Pixel}_{i,j,k} = \left\lfloor \frac{\text{Img\_Pixel}_{i,j,k}}{\text{Greyscale\_Reduce\_Index}} \right\rfloor$  //将像素值变为灰度降维后的位数值
             $\text{Img\_Pixel}_{i,j,k} = \text{Img\_Pixel}_{i,j,k} \cdot 255 / \left\lfloor \frac{255}{\text{Greyscale\_Reduce\_Index}} \right\rfloor$  //将降维后的像
            //素位数值映射到0-255之间
        end
    end
end
end
end

```

Algorithm 1: 灰度降维

图像的均值和方差，计算结果如表1所示。

<i>image</i>	<i>mean</i>	<i>variance</i>
<i>lena</i>	99.0512	2796.0318
<i>elain</i>	135.3779	2120.6471

表 1: lena和elain图片的均值与方差

3.3 *lena*图像的插值

我们分别对图像进行最邻近插值，双线性插值和双三次插值，分别得到了三幅不同的图像，我将从图像和算法两个方面对这三个结果进行展示，插值的结果如图所示3。



(a) 最邻近插值的图像



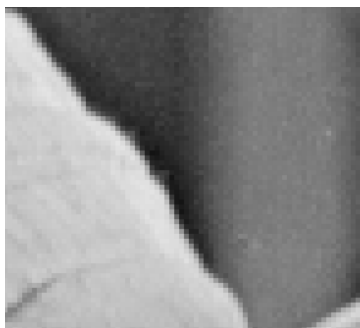
(b) 双线性插值的图像



(c) 双三次插值的图像

图 3: 插值结果

我们可以对图片进行放大，观察其细节，如图4所示。



(a) 最邻近插值



(b) 双线性插值



(c) 双三次插值

图 4: 插值结果放大

从图像细节中可以看出，最邻近插值的方法表现出的效果较差，区域模糊化明显，同时双线性内插的效果较最邻近内插有重大改进，而双三次内插的结果较双线性内插的结果稍微清晰一些，这我们的设想一致。

3.3.1 最邻近插值

最邻近内插的方法是将原图像中最临近的灰度赋值给每个新的位置，这种方法简单，但是会有很多缺陷，如很多边缘的失真，由于这些缺陷，实际上这样的插值方法并不常用。

3.3.2 双线性插值

双线性内插的效果较最邻近内插有了明显的改观，其原理如下。

我们用 (x, y) 来表示我们想要赋以灰度值的位置，并令 $v(x, y)$ 表示灰度，则对于双线性内插来说，赋值公式2为：

$$v(x, y) = ax + by + cxy + d \quad (2)$$

式中，4个系数可用由点 (x, y) 的四个最邻近的点写出的未知方程确定。不过，这也增加了插值的计算量^[3]。

3.3.3 双三次插值

双三次内插的效果较双线性内插有了更加清晰的效果，其赋值公式3为：

$$v(x, y) = \sum_{i=0}^3 \sum_{j=0}^3 a_{i,j} x^i y^j \quad (3)$$

式中，16个系数可用由点 (x, y) 的16个最邻近的点写出的未知方程确定。双三次插值在保持细节方面比双线性插值做的要更好^[3]。

3.4 lena图像和elain图像的仿射变换和插值处理

下面我们将介绍利用图像的仿射变换，对图像进行水平偏移错切和旋转的操作。最常用的空间坐标变化是仿射变换，其一般形式如公式4所示^[3]。

$$\begin{pmatrix} x & y & 1 \end{pmatrix} = \begin{pmatrix} v & w & 1 \end{pmatrix} T = \begin{pmatrix} v & w & 1 \end{pmatrix} \begin{pmatrix} t_{1,1} & t_{1,2} & 0 \\ t_{2,1} & t_{2,2} & 0 \\ t_{3,1} & t_{3,2} & 1 \end{pmatrix} \quad (4)$$

式中, (v, w) 是原图像中像素的坐标, (x, y) 是变换后图像的坐标。通过仿射变换我们可以实现图像的水平偏移和旋转等操作。

3.4.1 水平偏移

水平偏移是通过图片的仿射变换来完成的, 如公式5所示^[3]。

$$\begin{pmatrix} x & y & 1 \end{pmatrix} = \begin{pmatrix} v & w & 1 \end{pmatrix} \begin{pmatrix} 1 & S_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5)$$

将 S_h 的值设为1.5, 可以得到图5和图6。

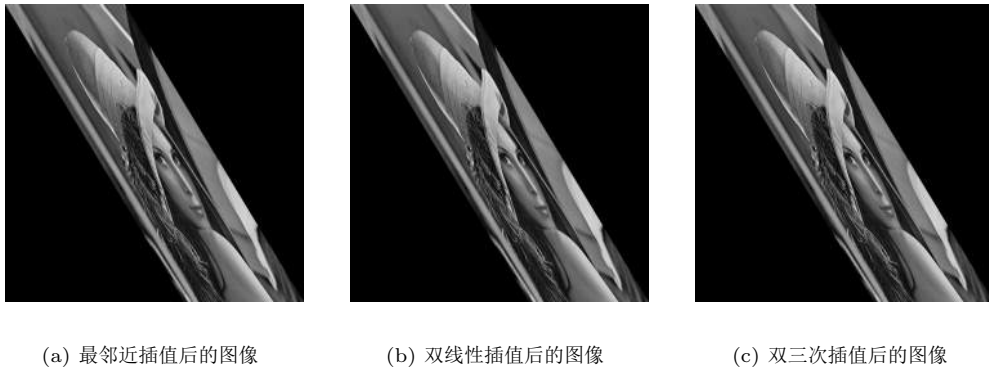


图 5: lena图像水平偏移变换并插值的结果

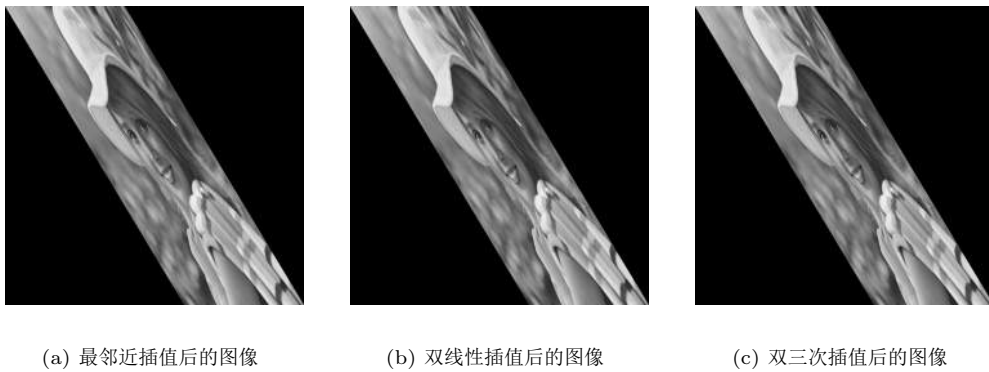


图 6: elain图像水平偏移变换并插值的结果

3.4.2 旋转

旋转变换也是通过图片的仿射变换来完成的，如公式6所示^[3]。

$$\begin{pmatrix} x & y & 1 \end{pmatrix} = \begin{pmatrix} v & w & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (6)$$

将 θ 的值设为30度，可以得到图7和图8。



图 7: lena图像旋转30度变换并插值的结果



图 8: elain图像旋转30度变换并插值的结果

4 小结

本次作业，通过实践，我深入了解了图像处理的相关基础知识，同时也提高了动手能力，加深了理解和掌握程度，也让我拥有了解决实际问题的基本功。同时，代码能力

也得到了有效锻炼，我的python能力得到了有效锻炼，同时也学习了opencv的很多基本功能。我自行组织的cvtoolbox在我的github上维护，欢迎访问

<https://github.com/1989Ryan/Digital-Image-Processing-Project>进行了解。

参考文献

- [1] Microsoft Windows Dev Center, Windows GDI, “Bitmap”, [online] available:
[https : //docs.microsoft.com/en – us/windows/desktop/gdi/bitmaps](https://docs.microsoft.com/en-us/windows/desktop/gdi/bitmaps) (February, 26, 2019).
- [2] Wikipedia, BMP file format, [online] available:
[https : //en.wikipedia.org/wiki/BMP_file_format](https://en.wikipedia.org/wiki/BMP_file_format) (February, 26, 2019).
- [3] Rafael C. Gonzalez, Richard E. Woods, 数字图像处理. 北京: 电子工业出版社, 2017.5.