西安交通大學

XI'AN JIAOTONG UNIVERSITY

数字图像处理实验报告

项目名称：BMP 图像的基本操作

姓名：XXX

班级：XXXXXX

学号：XXXXXXXXXX

摘　要

该实验报告以 BMP 文件的处理为主线，对 Windows 操作系统中的标准图像文件格式——BMP 格式文件进行了基本的介绍，理解了 BMP 图像文件的文件结构和存储特征。

本文主要以经典的 Lena 图片为例，对 BMP 文件进行了灰度逐级递减操作、求均值方差、放大、剪切和旋转等基本变换，通过这些操作实践，在理论的基础上深刻认识了数字图像处理中图像基本操作的特点和本质，为将来的学习打下良好基础。

本实验基于 opencv 这一开源的计算机视觉库，使用 Python 进行程序的编写。

一 题目 1

1.1 题目

对 BMP 格式文件进行简介。

1.2 解答

BMP 取自位图 Bitmap 的缩写，也称为 DIB（device independent bitmap，与设备无关的位图），是一种独立于显示器的位图数字图像文件格式。常见于 Windows 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。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。

BMP 文件通常是不压缩的，所以它们通常比同一幅图像的压缩图像文件格式要大很多。因此它们通常不适合在因特网或者其他低速或者有容量限制的介质上进行传输。

根据颜色深度的不同，图像上的一个像素可以用一个或者多个字节表示，它由 n/8 所确定（n 是位深度，1 字节包含 8 个数据位）。图片浏览器等基于字节的 ASCII 值计算像素的颜色，然后从调色板中读出相应的值。

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。由于该文件格式几经演进，这些结构体的版本也很多。

位图文件由以下结构体依次构成：

表1　BMP 文件结构体

结构体名称	可选	大小	用途	备注
位图文件头	否	14 字节	存储位图文件通用信息	仅在读取文件时有用
DIB 头	否	固定（存在 7 种不同版本）	存储位图详细信息及像素格式	紧接在位图文件头后
附加位掩码	是	3 或 4 DWORD（12 或 16 字节）	定义像素格式	仅在 DIB 头是 BITMAPINFOHEADER 时存在

调色板	半可选	可变	定义图像数据（像素数组）所用颜色	色深≤ 8 时不能省略
填充区 A	是	可变	结构体对齐	位图文件头中像素数组偏移量的产物
像素数组	否	可变	定义实际的像素数值	像素数据在 DIB 头和附加位掩码中定义。像素数组中每行均以 4 字节对齐
填充区 B	是	可变	结构体对齐	DIB 头中 ICC 色彩特性数据偏移量的产物
ICC 色彩特性数据	是	可变	定义色彩特性	可以包含外部文件路径，由该文件来定义色彩特性

位图文件头数据块位于文件开头，用于进行文件的识别。典型的应用程序会首先普通读取这部分数据以确保的确是位图文件并且没有损坏。所有的整数值都以小端序存放（即最低有效位前置）。

DIP 头数据告诉应用程序图像的详细信息，在屏幕上显示图像将会使用这些信息，它从文件的第 15 个字节开始。它对应了 Windows 中的内部使用的头结构以及其它一些版本的变体。但所有版本均以一个 DWORD 位（32 位）开始，用以说明该数据区块的大小，使得应用程序能够根据这个大小来区分该图像实际使用了哪种版本的 DIB 头结构。存在多种版本的头结构的原因是微软对 DIB 格式进行过多次扩展。

调色板部分定义了图像中所用的颜色。如上所述，位图图像一个像素接着一个像素储存，每个像素使用一个或者多个字节的值表示，所以调色板的目的就是要告诉应用程序这些值所对应的实际颜色。典型的位图文件使用 RGB 彩色模型。在这种模型中，每种颜色都是由不同强度（从 0 到最大强度）的红色（R）、绿色（G）和蓝色（B）组成的，也就是说，每种颜色都可以使用红色、绿色和蓝色的值所定义。在位图文件的实现中，调色板可以包含很多条目，条目个数就是图像中所使用的颜色的个数。

```
typedef struct tagRGBQUAD {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
}   RGBQUAD;
```

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这个公式计算：

$$\text{displaystyleRowSize} = \left\lfloor \frac{\text{BitsPerPixel} \cdot \text{ImageWidth} + 31}{32} \right\rfloor \cdot 4$$

二 题目 2

2.1 题目

把 lena 512*512 图像灰度级逐级递减 8-1 显示。

2.2 解答

如果想让灰度图像的灰度更少，有如下两个选择：一种简单的方法是将图像除以因子 N（整数除法），然后乘以因子N。或者，可以使用一些聚类算法（例如 K-means 或中值切割算法）将图像划分为 k 种颜色。

在这里我们使用第一种做法，即将图像的灰度值整体除以N，这里可以采用截断取整或四舍五入取整的方法，之后再乘以N。经试验，采用截断取整（floor）比四舍五入取整（around）的效果更好，更能体现灰度级逐级递减 8-1 的区别，因此这里采用截断取整的方式。源代码见附录 6.2。

原图有 128 阶（$=2^7$）灰度值，通过如下公式（$N=2^n$）

$$I_{new} = \left\lfloor \frac{I_{old}}{2^n} \right\rfloor \cdot 2^n,\ n = 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7$$

可分别得到灰度级递减从 8 到 1 的图，其中第一张（$n=0$）为原图。

如下即为所得灰度级递减结果，均采用截断取整（为适应 word 文档大小，已缩小图片）：

图1　灰度级逐级递减的图像（截断取整）

注：若采用四舍五入取整，结果不如截断取整，如下图所示：

图2　灰度级逐级递减的图像（四舍五入取整）

三 题目 3

3.1 题目

计算 lena 图像的均值方差。

3.2 解答

可以通过 opencv 的 meanStdDev()函数计算图像的均值和方差，源代码见附录 6.3，该函数分别返回图像的均值和标准差，其计算公式为

$$N = \sum_{I,\ mask(I)\neq 0} 1$$

$$mean_c = \frac{\sum_{I,\ mask(I)\neq 0} src(I)_c}{N}$$

$$stddev_c = \sqrt{\frac{\sum_{I,\ mask(I)\neq 0} \left(src(I)_c - mean_c\right)^2}{N}}$$

计算得到均值为 99.0512，标准差为 52.8775，方差为标准差的平方，即为 2796.0318。

四 题目 4

4.1 题目

把 lena 图像用近邻、双线性和双三次插值法更改大小（zoom）到2048×2048。

4.2 解答

原图为512×512的图像，要经三种方式更改为分辨率更高的图片，我们使用 opencv 的 resize 函数进行操作，该函数提供了多种插值方法（interpolation method），如最近邻插值（nearest neighbor interpolation）、双线性插值（bilinear interpolation）、双三次插值（bicubic interpolation）等方式，通过对该函数指定不同的参数来分配不同的插值方式。

注意到2048×2048图片再 opencv 中的输出结果会铺满整个屏幕，不方便查看，我们在此使用 imwrite 函数将输出的图片存为文件后再进行查看和输出，结果如下所示（为

适应 word 文档大小，已缩小图片）。源代码见附录 6.4。

图3　近邻插值结果

图4　双线性插值结果

图5　双三次插值结果

经过仔细观察，该 lena 图片扩大插值后的效果（清晰度）为双三次优于双线性，双线性优于近邻插值。

五 题目 5

5.1 题目

把 lena 和 elain 图像分别进行水平 shear（参数可设置为 1.5，或者自行选择）和旋转$30°$，并采用用近邻、双线性和双三次插值法 zoom 到2048×2048。

5.2 解答

对于图像剪切（Image Shearing），分为水平剪切和垂直剪切，其变换矩阵有如下广义形式：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

在这里我们采用水平剪切，只需将sh_y设为 0 即可，并将sh_x设为 1.5。在代码中，我们使用 opencv 的 warpPerspective()函数进行变换，该函数能够接收变换矩阵对图形完成剪切。图片剪切后，将图像通过近邻、双线性和双三次这三种插值方式进行放大至2048×2048得到所求结果。以下图片显示了图片剪切和放大后的结果。

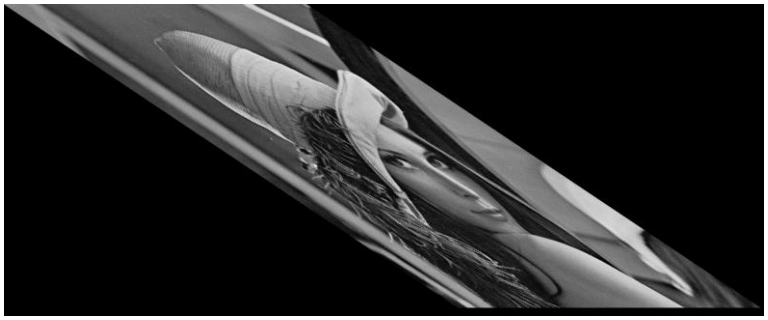

图6　Lena 图片剪切、最近邻插值

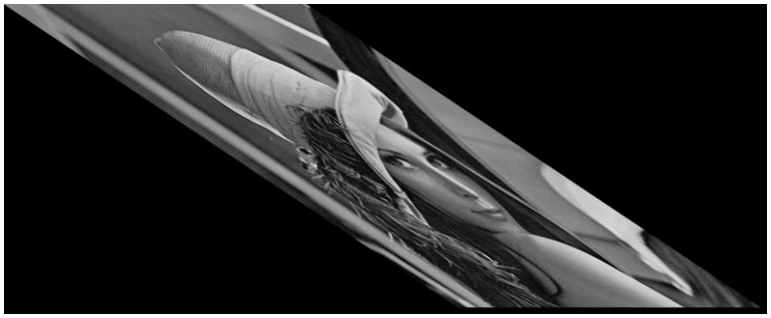

图7　Lena 图片剪切、双线性插值

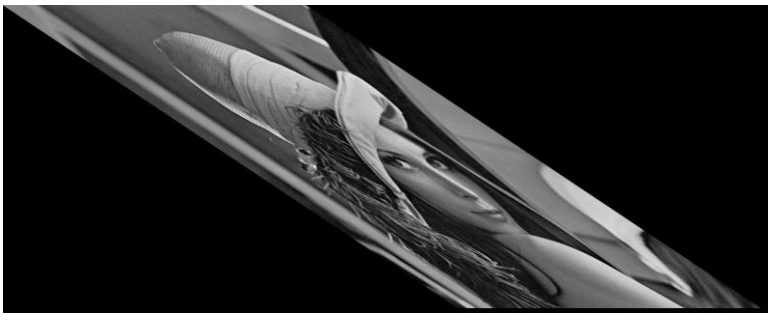

图8　Lena 图片剪切、双三次插值

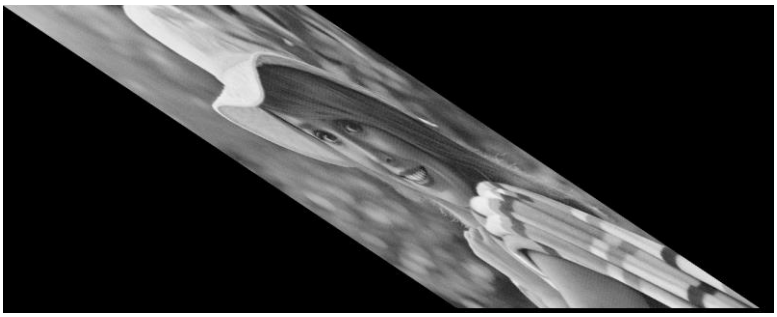

图9　Elain 图片剪切、最近邻插值

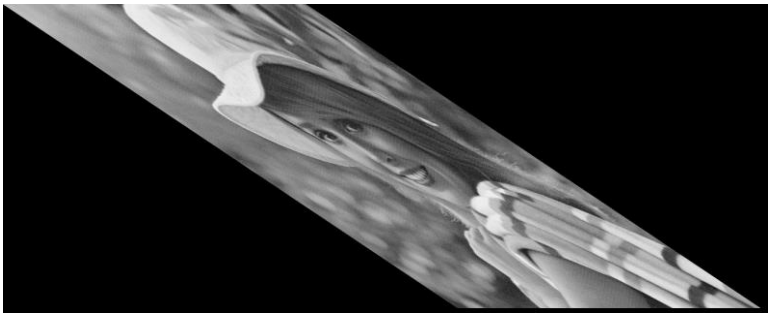

图10　Elain 图片剪切、双线性插值

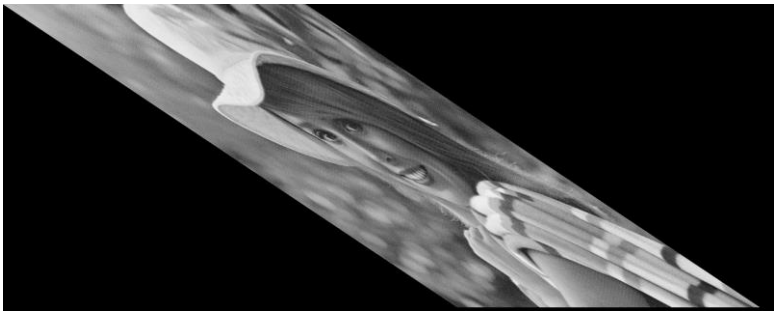

图11　Elain 图片剪切、双三次插值

而对于图像旋转，其关于原点的旋转矩阵为$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$，坐标变换公式为

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

这里为了保证图片旋转后居中，且考虑到题中所给两张图片均为正方形，我们将图片中心点设为旋转中心，而非坐标原点。在代码中，我们使用 warpAffine()函数进行旋转变换，该函数可接收旋转矩阵得出变换结果，而旋转矩阵可通过 getRotationMatrix2D()函数求得，这一函数接收旋转角度、旋转角和旋转原点坐标，计算得到旋转矩阵，注意

该函数输入角度为正时表示默认的逆时针旋转，若要顺时针旋转则需输入负角度值。

旋转后通过类似的方式将图片放大即可，源代码见附录 6.5，如下为图片逆时针旋转30°后放大的输出结果。

图12　Lena 图片旋转、最近邻插值

图13　Lena 图片旋转、双线性插值

图14　Lena 图片旋转、双三次插值

图15　Elain 图片旋转、最近邻插值

图16　Elain 图片旋转、双线性插值

图17　Elain 图片旋转、双三次插值

六 附录：所有代码

6.1 导入库文件

```
import cv2
import numpy as np
```

6.2 题目 2

```
# 把 lena 512*512 图像灰度级逐级递减 8-1 显示
img = cv2.imread("Resource/lena.bmp", flags=cv2.IMREAD_GRAYSCALE)


# 可以改变图片大小的堆叠函数，参考 https://github.com/murtazahassan/Learn-OpenCV-in-3-hours
def stackImages(scale, imgArray):
    rows = len(imgArray)
    cols = len(imgArray[0])
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    if rowsAvailable:
        for x in range ( 0, rows):
            for y in range(0, cols):
                if imgArray[x][y].shape[:2] == imgArray[0][0].shape [:2]:
                    imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)
                else:
                    imgArray[x][y] = cv2.resize(imgArray[x][y], (imgArray[0][0].shape[1], imgArray[0][0].shape[0]), None, scale, scale)
                if len(imgArray[x][y].shape) == 2: imgArray[x][y]= cv2.cvtColor( imgArray[x][y], cv2.COLOR_GRAY2BGR)
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
        ver = np.vstack(hor)
```

```
    else:
        for x in range(0, rows):
            if imgArray[x].shape[:2] == imgArray[0].shape[:2]:
                imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)
            else:
                imgArray[x] = cv2.resize(imgArray[x], (imgArray[0].shape[1],
imgArray[0].shape[0]), None,scale, scale)
            if len(imgArray[x].shape) == 2: imgArray[x] =
cv2.cvtColor(imgArray[x], cv2.COLOR_GRAY2BGR)
        hor= np.hstack(imgArray)
        ver = hor
    return ver


# 四舍五入取整
# img7 = np.array(np.around(img / 2) * 2, dtype=np.uint8)
# img6 = np.array(np.around(img / 4) * 4, dtype=np.uint8)
# img5 = np.array(np.around(img / 8) * 8, dtype=np.uint8)
# img4 = np.array(np.around(img / 16) * 16, dtype=np.uint8)
# img3 = np.array(np.around(img / 32) * 32, dtype=np.uint8)
# img2 = np.array(np.around(img / 64) * 64, dtype=np.uint8)
# img1 = np.array(np.around(img / 128) * 128, dtype=np.uint8)

# 截断取整
img7 = np.array(np.floor(img / 2) * 2, dtype=np.uint8)
img6 = np.array(np.floor(img / 4) * 4, dtype=np.uint8)
img5 = np.array(np.floor(img / 8) * 8, dtype=np.uint8)
img4 = np.array(np.floor(img / 16) * 16, dtype=np.uint8)
img3 = np.array(np.floor(img / 32) * 32, dtype=np.uint8)
img2 = np.array(np.floor(img / 64) * 64, dtype=np.uint8)
img1 = np.array(np.floor(img / 128) * 128, dtype=np.uint8)

imgStack = stackImages(0.5, ([img, img7, img6, img5], [img4, img3, img2, img1]))
cv2.imshow("ImgStack", imgStack)
cv2.waitKey(0)
```

6.3 题目 3

```
# 计算图像均值和方差

imgOriginal = cv2.imread("Resource/lena.bmp", flags=cv2.IMREAD_UNCHANGED)
```

```
mean, stddev = cv2.meanStdDev(imgOriginal)
print(mean)
print(stddev)
```

6.4 题目 4

```
# zoom(resize) 图像
img = cv2.imread("Resource/lena.bmp")
imgNear = cv2.resize(imgOriginal, (2048, 2048), interpolation=cv2.INTER_NEAREST)
# 近邻插值
imgBilinear = cv2.resize(imgOriginal, (2048, 2048),
interpolation=cv2.INTER_LINEAR)  # 双线性插值
imgBicubic = cv2.resize(imgOriginal, (2048, 2048),
interpolation=cv2.INTER_CUBIC)  # 双三次插值

cv2.imwrite("Output/imgNear.bmp", imgNear)
cv2.imwrite("Output/imgBilinear.bmp", imgBilinear)
cv2.imwrite("Output/imgBicubic.bmp", imgBicubic)
```

6.5 题目 5

```
# 图像 shearing 后 resize 至 2048*2048
lena = cv2.imread("Resource/lena.bmp", cv2.IMREAD_UNCHANGED)
elain = cv2.imread("Resource/elain1.bmp", cv2.IMREAD_UNCHANGED)

shearMatrix = np.float32([[1, 1.5, 0], [0, 1, 0], [0, 0, 1]])
shearedLenaNear = cv2.warpPerspective(lena, shearMatrix, dsize=(2048, 2048),
flags=cv2.INTER_NEAREST)
shearedElainNear = cv2.warpPerspective(elain, shearMatrix, dsize=(2048, 2048),
flags=cv2.INTER_NEAREST)
cv2.imwrite("Output/shearedLenaNear.bmp", shearedLenaNear)
cv2.imwrite("Output/shearedElainNear.bmp", shearedElainNear)

shearedLenaLinear = cv2.warpPerspective(lena, shearMatrix, dsize=(2048, 2048),
flags=cv2.INTER_LINEAR)
```

```
shearedElainLinear = cv2.warpPerspective(elain, shearMatrix, dsize=(2048, 2048),
flags=cv2.INTER_LINEAR)
cv2.imwrite("Output/shearedLenaLinear.bmp", shearedLenaLinear)
cv2.imwrite("Output/shearedElainLinear.bmp", shearedElainLinear)

shearedLenaCubic = cv2.warpPerspective(lena, shearMatrix, dsize=(2048, 2048),
flags=cv2.INTER_CUBIC)
shearedElainCubic = cv2.warpPerspective(elain, shearMatrix, dsize=(2048, 2048),
flags=cv2.INTER_CUBIC)
cv2.imwrite("Output/shearedLenaCubic.bmp", shearedLenaCubic)
cv2.imwrite("Output/shearedElainCubic.bmp", shearedElainCubic)
```

```
# 图像旋转
def rotate(image, angle, rotPoint=None):
    (height, width) = image.shape[:2]

    if rotPoint is None:
        rotPoint = (width//2, height//2)

    rotMat = cv2.getRotationMatrix2D(rotPoint, angle, scale=1.0)
    dimensions = [width, height]
    # dimensions = [2048, 2048]

    return cv2.warpAffine(image, rotMat, dimensions)


rotatedLena = rotate(lena, 30)
rotatedLenaNear = cv2.resize(rotatedLena, (2048, 2048),
interpolation=cv2.INTER_NEAREST)
rotatedLenaLinear = cv2.resize(rotatedLena, (2048, 2048),
interpolation=cv2.INTER_LINEAR)
rotatedLenaCubic = cv2.resize(rotatedLena, (2048, 2048),
interpolation=cv2.INTER_CUBIC)
cv2.imwrite("Output/rotatedLenaNear.bmp", rotatedLenaNear)
cv2.imwrite("Output/rotatedLenaLinear.bmp", rotatedLenaLinear)
cv2.imwrite("Output/rotatedLenaCubic.bmp", rotatedLenaCubic)

rotatedElain = rotate(elain, 30)
rotatedElainNear = cv2.resize(rotatedElain, (2048, 2048),
interpolation=cv2.INTER_NEAREST)
```

```
rotatedElainLinear = cv2.resize(rotatedElain, (2048, 2048),
interpolation=cv2.INTER_LINEAR)
rotatedElainCubic = cv2.resize(rotatedElain, (2048, 2048),
interpolation=cv2.INTER_CUBIC)
cv2.imwrite("Output/rotatedElainNear.bmp", rotatedElainNear)
cv2.imwrite("Output/rotatedElainLinear.bmp", rotatedElainLinear)
cv2.imwrite("Output/rotatedElainCubic.bmp", rotatedElainCubic)
```