# 数字图像处理第一次作业

自动化84 刘佳帅 2160506014

## 摘要

本文结合课程学习内容，以对Bmp图像格式的理解为基础，手动读取图像后对图像内容进行了分析和处理。

在实验一中重点解析了Bmp图像格式，简介了其四部分内容：bmp文件信息头、位图信息头、调色板、位图数据的相关内容。

实验二中，结合对灰度等级的理解，使用Python对图像的每个像素实现了不同灰度等级的显示。实验三中则使用8位（0~255）的灰度值求得均值方差，并对其物理意义简要分析。

实验四五中主要完成对图像的处理。实验四中采用三种常用方案：近邻法、双线性法、双三次插值法，扩大了图像的分辨率。分析了实现过程，并对比了实验结果。实验五则重点讲解了图像的仿射变换，分析了变换矩阵获得方式，实现了图像的旋转和错切。

2021年3月15日

## 实验一

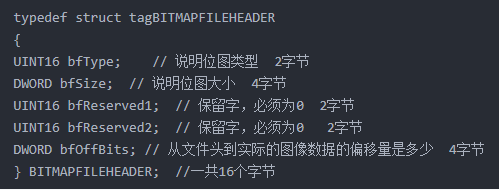
题目：Bmp图像格式简介，以7.bmp为例说明

Bmp图像格式简介：

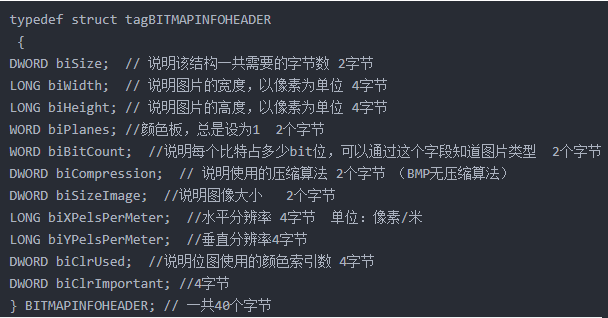
BMP文件格式，又称为Bitmap（位图）或是DIB(Device-Independent Device，设备无关位图)，是Windows系统中广泛使用的图像文件格式。由于它可以不作任何变换地保存图像像素域的数据，因此成为我们取得RAW数据的重要来源。Windows的图形用户界面（graphical user interfaces）也在它的内建图像子系统GDI中对BMP格式提供了支持。

BMP文件的数据按照从文件头开始的先后顺序分为四个部分[1]：

1）bmp文件头(bmp file header)：提供文件的格式、大小等信息



2）位图信息头(bitmap information)：提供图像数据的尺寸、位平面数、压缩方式、颜色索引等信息

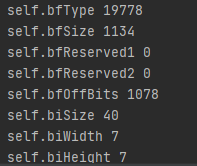
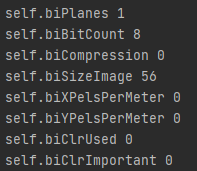


3）调色板(color palette)：可选，如使用索引来表示图像，调色板就是索引与其对应的颜色的映射表。

4）位图数据(bitmap data)：就是图像数据。

结果说明：

以图像7.bmp为例，输出上述信息的结果：



需要注意的是，上述信息按位一次存储在bmp格式的图像中。而整张图片的数据信息是从左下角到右上角排列的。例子中的图像每个像素为8位，因此一次读一个字节将数据依次读出后需要进行反转才能得到原图顺序的矩阵。

代码中实现了图像各个信息的读取，见附录。

## 实验二

题目：把lena 512\*512图像灰度级逐级递减8-1显示

实验原理[2]：

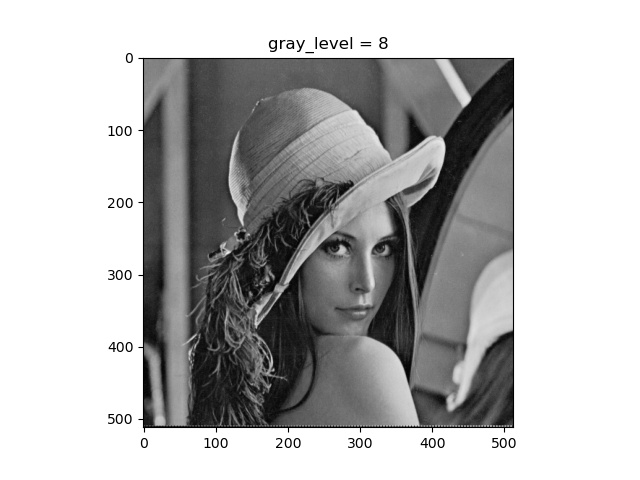
给定的图像lena.bmp是一张8位灰度图像，有256个灰度色，即灰度级为8。灰度级为K的灰度图像中某个像素的灰度值k(x,y) 与原图同像素的灰度值v(x,y)(即256色的图像)的关系为：



floor为向下取整。按照这样的方法，先读取每个图像的矩阵信息之后将图像的处理后的图像展示出来。

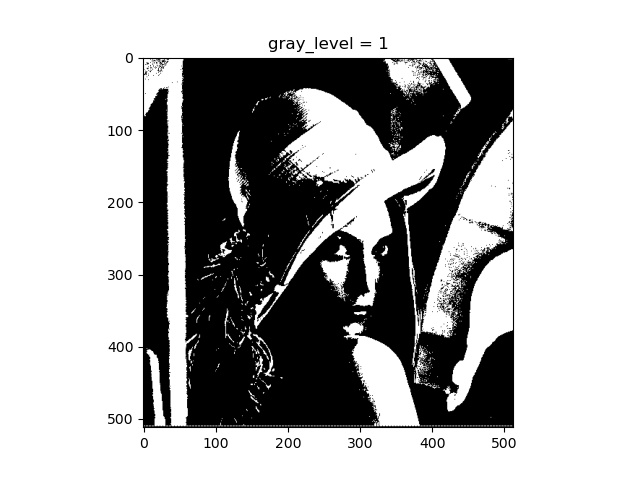
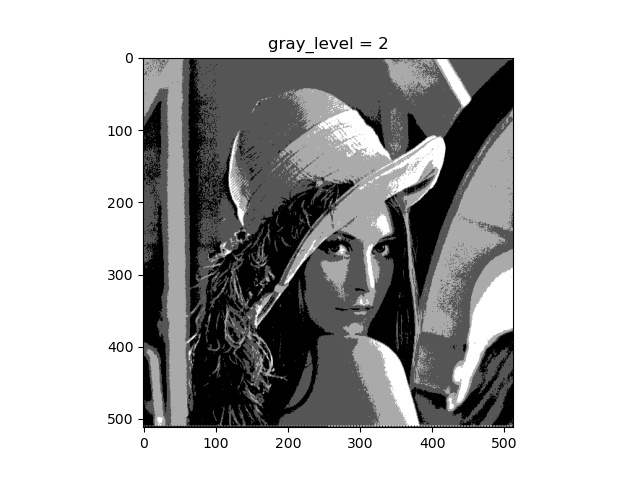
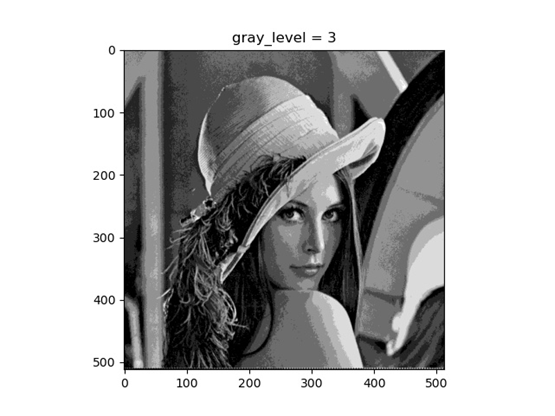
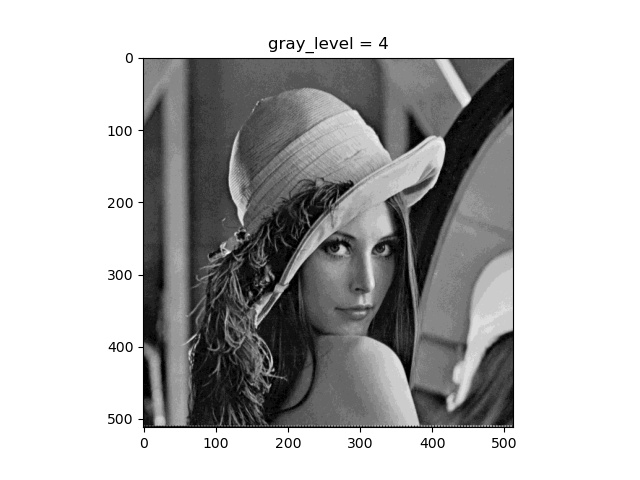
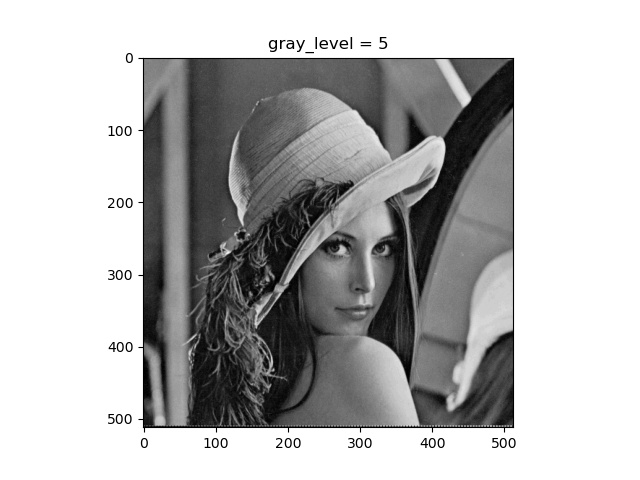
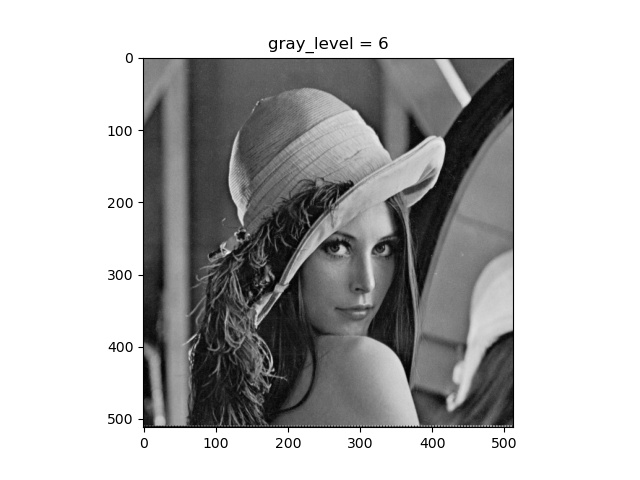
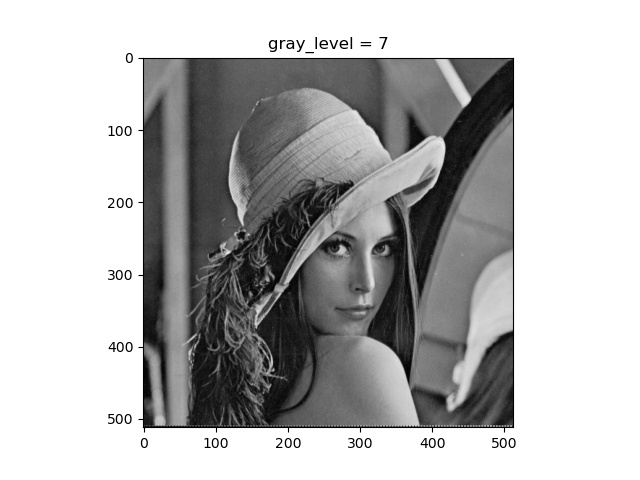
实验结果：

原图也就是灰度级为8的图像：



灰度级8

按照上述的方法依次得到下面的结果：



从上述实验结果中能够明显看出来，随着灰度级的下降，图像的细节有明显的变化，会觉得图像像素之间并不那么连续。这样的现象在灰度级大于等于5之前并不明显，跟人眼的分辨能力有关。最后图像退化为只有黑白色的图像。我们知道，图像采样的灰度阶数越高，细节越丰富，更接近真实的情况。

## 实验三

题目：计算lena图像的均值方差

实验原理：

对于分辨率为M\*N的灰度图像，其均值和方差分别为：





实验结果：

图像的均值为 99.00114440917969

图像的方差为 2820.8324266688433

图像的标准差为 53.111509361614296

通过课上的学习我们了解到，图像的均值表征了图像整体的亮度。而图像的标准差（或者方差）表征了图像的对比度。

## 实验四

题目：把lena图像用近邻、双线性和双三次插值法zoom到2048\*2048

实验原理：

当给定分辨率的图像希望放大时，在图像像素上就会出现没有值的点，因此需要使用插值算法根据该点附近的点来产生更高分辨率的图像。典型方法：最近邻、双线性、双三次等。

近邻法：

最近邻方法就是将待插值的像素点取值为与其相对于原图坐标距离最近的点的灰度值。

双线性插值：

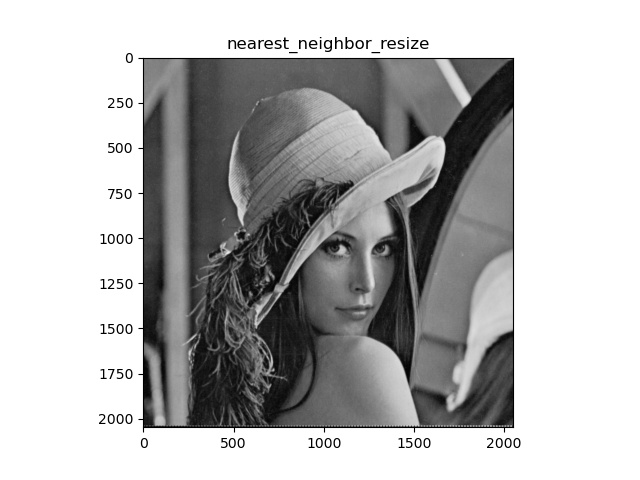
双线性插值法是将待插值点像素的灰度值取为与其最近的4个像素点的加权平均。得到的结果相对近邻法能更好地显示图像的边界。

双三次插值[3]：

该方法是将待插值点

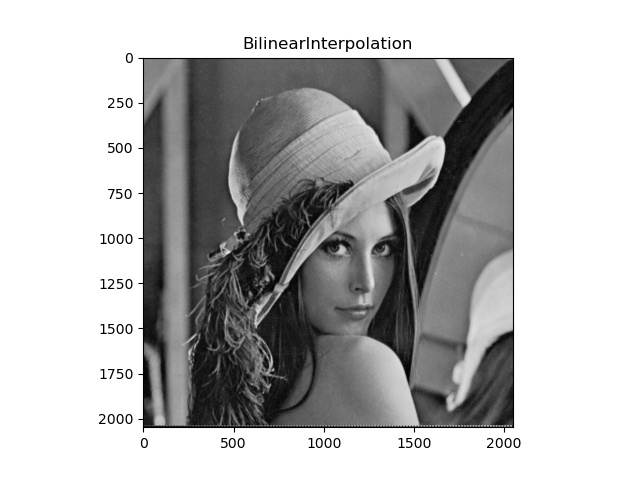
将待插值像素点取值为周围最近的4\*4像素方格内像素点的加权求和。实现时可以使用卷积算法或者使用两次三次样条插值。

实验结果：



最近邻法

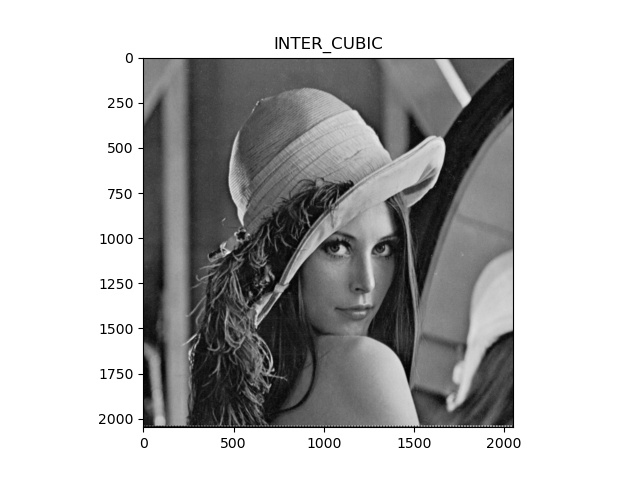
通过坐标我们能够看出来整个图像已经扩大。直接通过Python的OpenCV实现算法并且不对计算过程进行优化也并不需要很大的时间开销。接下来是线性插值：



双线性法

该算法在实现的过程中没有进行算法优化，与最近邻法相似采用遍历新图像每个像素点并进行计算的方案，计算的时间消耗很大，整张图片大约花费5min运行得到结果。放大图片能够发现边缘的处理更加柔和。

最后是三次插值，预计运行时间过长。由于没有掌握算法优化的方法，因此采用了python自带的插值函数实现：



双三次插值

由于内置函数的算法都经过优化，比如使用卷积运算就能够得到相似的结果而仅需要不大的开销，因此运行时间与前两种直接实现的没有什么可比性。结果较前两者有一定的改进，效果在人眼看来并不明显。

## 实验五

实验题目：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度，并采用用近邻、双线性和双三次插值法zoom到2048\*2048；

实验原理[4]：

无论是旋转还是水平错切(shear)，本质上都是对图像进行仿射变换，即在保持原有图像位置关系的基础上，将图像的坐标转换到新的坐标上。没有平移或者平移量为0的所有仿射变换可以用如下变换矩阵描述：



不同变换对应的a,b,c,d约束不同，排除了平移变换的所有仿射变换为线性变换（linear transformation），其特点是原点位置不变，多次线性变换的结果仍是线性变换。

为了涵盖平移，引入齐次坐标，在原有2维坐标的基础上，增广1个维度，如下所示：对于旋转而言，其仿射变换可以表示为：



因此仿射变换就用这样的矩阵表示。对于旋转变换，其坐标变换方程为：



因此相应的仿射变换的矩阵为：



将每个像素的坐标乘上变换矩阵就得到了变换后的坐标，对于某些像素点处没有值的情况，则采用之前实验四采用的插值方法解决。

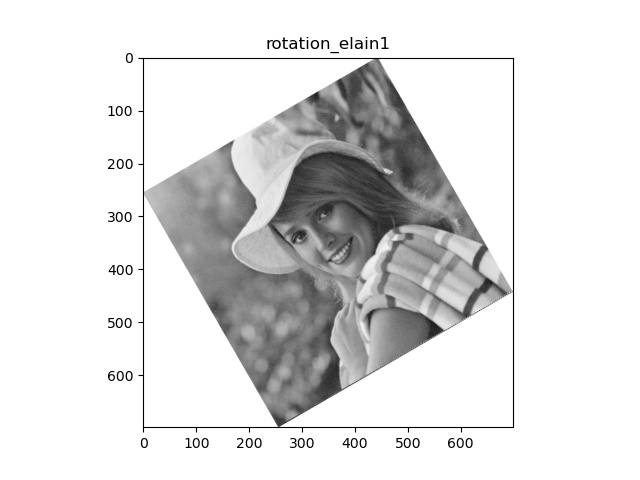
对于错切来说，相当于横坐标加上了纵坐标的倍数，而原纵坐标不变，能够产生一种错位投影的效果。其变换矩阵为：



其中是对应的系数，也就是横坐标相应加上了多少纵坐标的值。

实验结果：

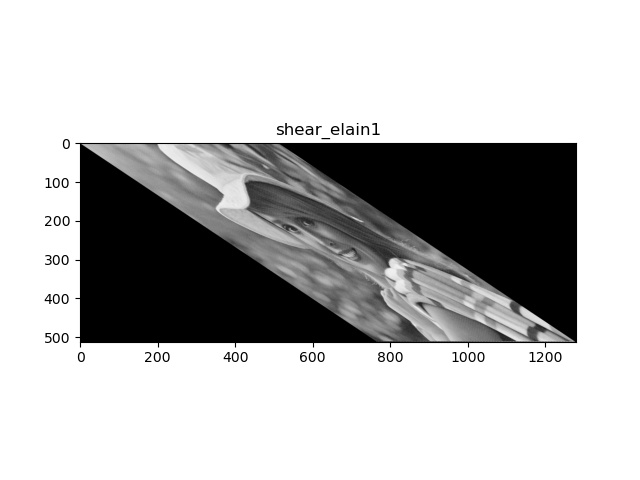
对于图像elain1.bmp，实现旋转：



elain1错切

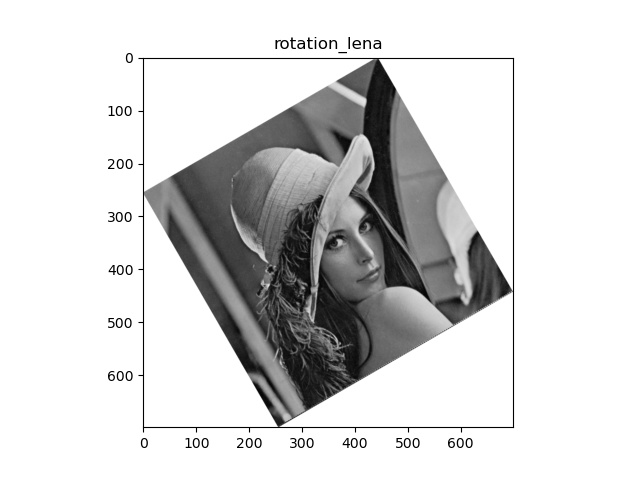
实现过程中直接使用了Python的OpenCV仿射变换函数，插值方法使用双三次插值解决没有完全覆盖的点。

实现错切：

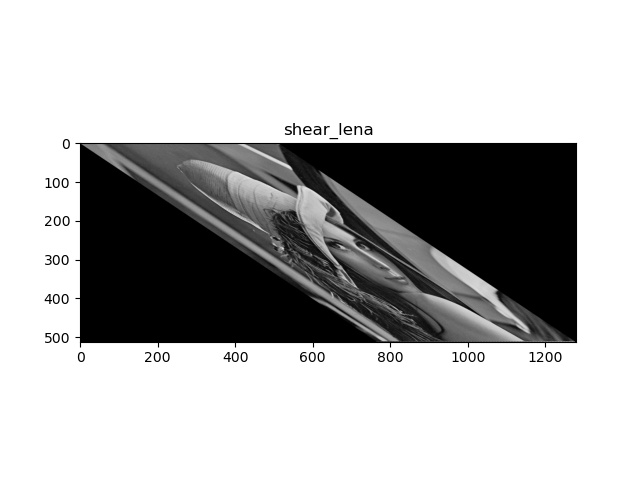


elain1错切

同理实现了lena.bmp的旋转与错切。



lena旋转



lena错切

参考文献：

[1]<https://blog.csdn.net/qq_43409114/article/details/104538619>

[2]<https://wenku.baidu.com/view/1357523d00f69e3143323968011ca300a6c3f6cc.html>

[3]<https://zhuanlan.zhihu.com/p/322393329#:~:text=%E5%8F%8C%E4%B8%89%E6%AC%A1%E6%8F%92%E5%80%BC%E6%98%AF%E4%BD%BF%E7%94%A8,%E5%92%8C%E6%94%BE%E5%A4%A7%E6%95%B0%E5%AD%97%E5%9B%BE%E5%83%8F%E3%80%82>

[4] <https://www.cnblogs.com/shine-lee/p/10950963.html>

附录：

ReadBMPFile.py:

# -\*- coding: UTF-8 -\*-

# 实现读取bmp文件及其相关内容

from struct import unpack

# 读取并存储 bmp 文件

class ReadBMPFile:

def \_\_init\_\_(self, filePath):

file = open(filePath, "rb")

# 读取 bmp 文件的文件头 14 字节

# 其中<表示Bmp是小端，h表示用16进制表示

self.bfType = unpack("<h", file.read(2))[0] # 0x4d42 对应BM 表示这是Windows支持的位图格式

self.bfSize = unpack("<i", file.read(4))[0] # 位图文件大小

self.bfReserved1 = unpack("<h", file.read(2))[0] # 保留字段 必须设为 0

self.bfReserved2 = unpack("<h", file.read(2))[0] # 保留字段 必须设为 0

self.bfOffBits = unpack("<i", file.read(4))[0] # 偏移量 从文件头到位图数据需偏移多少字节（位图信息头、调色板长度等不是固定的，这时就需要这个参数了）

# 读取 bmp 文件的位图信息头 40 字节

self.biSize = unpack("<i", file.read(4))[0] # 所需要的字节数

self.biWidth = unpack("<i", file.read(4))[0] # 图像的宽度 单位 像素

self.biHeight = unpack("<i", file.read(4))[0] # 图像的高度 单位 像素

self.biPlanes = unpack("<h", file.read(2))[0] # 说明颜色平面数 总设为 1

self.biBitCount = unpack("<h", file.read(2))[0] # 说明比特数

self.biCompression = unpack("<i", file.read(4))[0] # 图像压缩的数据类型

self.biSizeImage = unpack("<i", file.read(4))[0] # 图像大小

self.biXPelsPerMeter = unpack("<i", file.read(4))[0] # 水平分辨率

self.biYPelsPerMeter = unpack("<i", file.read(4))[0] # 垂直分辨率

self.biClrUsed = unpack("<i", file.read(4))[0] # 实际使用的彩色表中的颜色索引数

self.biClrImportant = unpack("<i", file.read(4))[0] # 对图像显示有重要影响的颜色索引的数目

self.bmp\_data = []

# # 对前面得到的结果进行输出

# print("self.bfType", self.bfType)

# print("self.bfSize", self.bfSize)

# print("self.bfReserved1", self.bfReserved1)

# print("self.bfReserved2", self.bfReserved2)

# print("self.bfOffBits", self.bfOffBits)

#

# print("self.biSize", self.biSize)

# print("self.biWidth", self.biWidth)

# print("self.biHeight", self.biHeight)

# print("self.biPlanes", self.biPlanes)

# print("self.biBitCount", self.biBitCount)

# print("self.biCompression", self.biCompression)

# print("self.biSizeImage", self.biSizeImage)

# print("self.biXPelsPerMeter", self.biXPelsPerMeter)

# print("self.biYPelsPerMeter", self.biYPelsPerMeter)

# print("self.biClrUsed", self.biClrUsed)

# print("self.biClrImportant", self.biClrImportant)

for height in range(self.biHeight):

bmp\_data\_row = []

# 四字节填充位检测

count = 0

for width in range(self.biWidth):

bmp\_data\_row.append(unpack("<B", file.read(1))[0])

count = count + 3

# bmp 四字节对齐原则

while count % 4 != 0:

file.read(1)

count = count + 1

self.bmp\_data.append(bmp\_data\_row)

self.bmp\_data.reverse()

file.close()

# print("self.bmp\_data", self.bmp\_data)

Picture\_01.py

"""

图像比特数是8，是黑白图像，不是RGB

读取7.bmp中的信息

"""

import numpy as np

import sys

from ReadBMPFile import ReadBMPFile

import cv2

# 传入的文件路径

filePath1 = r'D:\A\_learn\大三下数字图像处理\第一次作业\7.bmp'

filePath2 = r'D:\A\_learn\大三下数字图像处理\第一次作业\elain1.bmp'

filePath3 = r'D:\A\_learn\大三下数字图像处理\第一次作业\lena.bmp'

# 读取 第一张 BMP 文件

bmpFile = ReadBMPFile(filePath1)

data = bmpFile.bmp\_data

Data = np.array(data, dtype=np.uint8)

# Data = Data[0:256, :]

cv2.imshow("7", Data)

cv2.waitKey(0) # 解决窗口不显示图像的问题

cv2.destroyAllWindows()

picture\_02.py

"""

把lena 512\*512图像灰度级逐级递减8-1显示

"""

import numpy as np

from ReadBMPFile import ReadBMPFile

import matplotlib.pyplot as plt

# 命令行传入的文件路径

filePath1 = r'D:\A\_learn\大三下数字图像处理\第一次作业\7.bmp'

filePath2 = r'D:\A\_learn\大三下数字图像处理\第一次作业\elain1.bmp'

filePath3 = r'D:\A\_learn\大三下数字图像处理\第一次作业\lena.bmp'

# 读取 第一张 BMP 文件

bmpFile = ReadBMPFile(filePath3)

data = bmpFile.bmp\_data

Data = np.array(data, dtype=np.uint8)

plt.imshow(Data, cmap="gray")

plt.title("gray\_level = 8")

plt.savefig('./gray\_level8.jpg')

# img7 = np.floor(Data/2) \* 2 # 灰度等级为7

# plt.imshow(img7, cmap="gray")

# plt.title("gray\_level = 7")

# plt.savefig('./gray\_level7.jpg')

#

# img6 = np.floor(Data/4) \* 4 # 灰度等级为6

# plt.imshow(img6, cmap="gray")

# plt.title("gray\_level = 6")

# plt.savefig('./gray\_level6.jpg')

#

# img5 = np.floor(Data/8) \* 8 # 灰度等级为5

# plt.imshow(img5, cmap="gray")

# plt.title("gray\_level = 5")

# plt.savefig('./gray\_level5.jpg')

#

#

# img4 = np.floor(Data/16) \* 16 # 灰度等级为4

# plt.imshow(img4, cmap="gray")

# plt.title("gray\_level = 4")

# plt.savefig('./gray\_level4.jpg')

#

#

# img3 = np.floor(Data/32) \* 32 # 灰度等级为3

# plt.imshow(img3, cmap="gray")

# plt.title("gray\_level = 3")

# plt.savefig('./gray\_level3.jpg')

#

# img2 = np.floor(Data/64) \* 64 # 灰度等级为2

# plt.imshow(img2, cmap="gray")

# plt.title("gray\_level = 2")

# plt.savefig('./gray\_level2.jpg')

#

# img1 = np.floor(Data/128) \* 128 # 灰度等级为1

# plt.imshow(img1, cmap="gray")

# plt.title("gray\_level = 1")

# plt.savefig('./gray\_level1.jpg')

picture\_03.py

"""

计算图像lena的均值方差

"""

import numpy as np

from ReadBMPFile import ReadBMPFile

# 传入的文件路径

filePath1 = r'D:\A\_learn\大三下数字图像处理\第一次作业\7.bmp'

filePath2 = r'D:\A\_learn\大三下数字图像处理\第一次作业\elain1.bmp'

filePath3 = r'D:\A\_learn\大三下数字图像处理\第一次作业\lena.bmp'

# 读取 第一张 BMP 文件

bmpFile = ReadBMPFile(filePath3)

data = bmpFile.bmp\_data

Data = np.array(data, dtype=np.uint8)

pic\_mean = np.mean(Data)

print("图像的均值为", pic\_mean)

pic\_var = np.var(Data)

print("图像的方差为", pic\_var)

pic\_std = np.std(Data)

print("图像的标准差为", pic\_std)

picture\_04.py

"""

实现最近邻法，线性插值

"""

import numpy as np

import matplotlib.pyplot as plt

from ReadBMPFile import ReadBMPFile

def nearest\_neighbor\_resize(img, new\_w, new\_h):

# 定义最近邻

h, w = img.shape[0], img.shape[1]

# new image with rgb channel

ret\_img = np.zeros(shape=(new\_h, new\_w, 3), dtype='uint8')

# scale factor

s\_h, s\_c = (h \* 1.0) / new\_h, (w \* 1.0) / new\_w

# insert pixel to the new img

for i in range(new\_h):

for j in range(new\_w):

p\_x = int(j \* s\_c)

p\_y = int(i \* s\_h)

ret\_img[i, j] = img[p\_y, p\_x]

return ret\_img

def saturate(num):

# 定义越界函数

if num >= 512:

return 511

if num <= 0:

return 0

return num

def BilinearInterpolation(img, new\_h, new\_w):

# 实现双线性

height, width = img.shape

outimg = np.zeros(shape=(new\_h, new\_w))

for i in range(new\_h):

sy = i\*height/new\_h

i\_sy = int(sy)

j\_sy = sy-i\_sy

for j in range(new\_w):

sx = j\*width/new\_w

i\_sx = int(sx)

j\_sx = sx-i\_sx

c1 = (1-j\_sy)\*(1-j\_sx)

c2 = j\_sy\*(1-j\_sx)

c3 = j\_sy \* j\_sx

c4 = (1-j\_sy) \* j\_sx

# Sji\_x = saturate(int(np.round((j+0.5)\*width/shape[1]-0.5)), width-1)

# Sji\_y = saturate(int(np.round((i+0.5)\*height/shape[0]-0.5)), height-1)

# Sji1\_x = saturate(int(np.round((j+0.5)\*width/shape[1]-0.5)), width-1)

# Sji1\_y = saturate(int(np.round((i+1.5)\*height/shape[0]-0.5)), height-1)

# Sj1i1\_x = saturate(int(np.round((j+1.5)\*width/shape[1]-0.5)), width-1)

# Sj1i1\_y = saturate(int(np.round((i+1.5)\*height/shape[0]-0.5)), height-1)

# Sj1i\_x = saturate(int(np.round((j+1.5)\*width/shape[1]-0.5)), width-1)

# Sj1i\_y = saturate(int(np.round((i+0.5)\*height/shape[0]-0.5)), height-1)

Sji\_x = saturate(int(np.round((j+0.5)\*width/new\_w-0.5)))

Sji\_y = saturate(int(np.round((i+0.5)\*height/new\_h-0.5)))

Sji1\_x = saturate(int(np.round((j+0.5)\*width/new\_w-0.5)))

Sji1\_y = saturate(int(np.round((i+1.5)\*height/new\_h-0.5)))

Sj1i1\_x = saturate(int(np.round((j+1.5)\*width/new\_w-0.5)))

Sj1i1\_y = saturate(int(np.round((i+1.5)\*height/new\_h-0.5)))

Sj1i\_x = saturate(int(np.round((j+1.5)\*width/new\_w-0.5)))

Sj1i\_y = saturate(int(np.round((i+0.5)\*height/new\_h-0.5)))

outimg[i, j] = int(img[Sji\_y, Sji\_x]\*c1 + img[Sji1\_y, Sji1\_x]\*c2 + img[Sj1i1\_y, Sj1i1\_x]\*c3 + img[Sj1i\_y, Sj1i\_x]\*c4)

return np.array(outimg, dtype=np.uint8)

if \_\_name\_\_ == "\_\_main\_\_":

# 传入的文件路径

filePath1 = r'D:\A\_learn\大三下数字图像处理\第一次作业\7.bmp'

filePath2 = r'D:\A\_learn\大三下数字图像处理\第一次作业\elain1.bmp'

filePath3 = r'D:\A\_learn\大三下数字图像处理\第一次作业\lena.bmp'

# 读取 第一张 BMP 文件

bmpFile = ReadBMPFile(filePath3)

data = bmpFile.bmp\_data

Data = np.array(data, dtype=np.uint8)

new\_w, new\_h = 2048, 2048 # 定义新的长宽

# 使用最近邻绘制图像

zoom\_nn\_pic = nearest\_neighbor\_resize(Data, new\_w, new\_h)

plt.imshow(zoom\_nn\_pic, cmap="gray")

plt.title("nearest\_neighbor\_resize")

plt.savefig('./nearest\_neighbor\_resize.jpg')

# # 使用双线性插值

# zoom\_bi\_pic = BilinearInterpolation(Data, new\_h, new\_w)

# plt.imshow(zoom\_bi\_pic, cmap="gray")

# plt.title("BilinearInterpolation")

# plt.savefig('./BilinearInterpolation.jpg')

picture\_05.py

"""

使用python 内置函数实现三次插值

"""

import cv2

import numpy as np

from ReadBMPFile import ReadBMPFile

import matplotlib.pyplot as plt

if \_\_name\_\_ == "\_\_main\_\_":

# 传入的文件路径

filePath1 = r'D:\A\_learn\大三下数字图像处理\第一次作业\7.bmp'

filePath2 = r'D:\A\_learn\大三下数字图像处理\第一次作业\elain1.bmp'

filePath3 = r'D:\A\_learn\大三下数字图像处理\第一次作业\lena.bmp'

# 读取 第一张 BMP 文件

bmpFile = ReadBMPFile(filePath3)

data = bmpFile.bmp\_data

Data = np.array(data, dtype=np.uint8)

shape = (2048, 2048) # 定义放大的大小

pic\_cubic = cv2.resize(Data, shape, interpolation=cv2.INTER\_CUBIC)

plt.imshow(pic\_cubic, cmap="gray")

plt.title("INTER\_CUBIC")

plt.savefig('./INTER\_CUBIC.jpg')

picture\_06.py

"""

通过仿射变换实现旋转和错切

"""

import cv2

import numpy as np

from ReadBMPFile import ReadBMPFile

import matplotlib.pyplot as plt

from math import sin, cos, fabs, radians

if \_\_name\_\_ == "\_\_main\_\_":

# 传入的文件路径

filePath1 = r'D:\A\_learn\大三下数字图像处理\第一次作业\7.bmp'

filePath2 = r'D:\A\_learn\大三下数字图像处理\第一次作业\elain1.bmp'

filePath3 = r'D:\A\_learn\大三下数字图像处理\第一次作业\lena.bmp'

# 读取 第一张 BMP 文件

bmpFile = ReadBMPFile(filePath3)

data = bmpFile.bmp\_data

Data = np.array(data, dtype=np.uint8)

h, w = (512, 512)

# size = (2048, 2048)

size = (int(h + 1.5\*h), w)

# 实现错切

Mat\_rat = np.array([[1, 1.5, 0], [0, 1, 0]])

# pic\_shear = cv2.warpAffine(Data, Mat\_rat, size, flags=cv2.INTER\_CUBIC)

# plt.imshow(pic\_shear, cmap="gray")

# plt.title("shear\_elain1")

# plt.savefig('./shear\_elain1.jpg')

pic\_shear2 = cv2.warpAffine(Data, Mat\_rat, size, flags=cv2.INTER\_CUBIC)

plt.imshow(pic\_shear2, cmap="gray")

plt.title("shear\_lena")

plt.savefig('./shear\_lena.jpg')

# 实现旋转

angle = 30 # 定义旋转角度

M = cv2.getRotationMatrix2D((w / 2, h / 2), angle, 1)

newW = int(h \* fabs(sin(radians(angle))) + w \* fabs(cos(radians(angle))))

newH = int(w \* fabs(sin(radians(angle))) + h \* fabs(cos(radians(angle))))

M[0, 2] += (newW - w) / 2

M[1, 2] += (newH - h) / 2

# pic\_rot = cv2.warpAffine(Data, M, (newW, newH), borderValue=(255, 255, 255))

# plt.imshow(pic\_rot, cmap="gray")

# plt.title("rotation\_elain1")

# plt.savefig('./rotation\_elain1.jpg')

pic\_rot2 = cv2.warpAffine(Data, M, (newW, newH), borderValue=(255, 255, 255))

plt.imshow(pic\_rot2, cmap="gray")

plt.title("rotation\_lena")

plt.savefig('./rotation\_lena.jpg')