哈爾濱工業大學

数字媒体技术

实验报告

题　　目	声音、图像数据显示与变换
学　　院	计算机科学与技术
专　　业	软件工程
学　　号	1173710204
学　　生	陈东鑫
任 课 教 师	刘绍辉

哈尔滨工业大学计算机科学与技术学院

2020.3

实验 1:PCM 音频数据、位图数据读取及显示

注意：请按照大家阅读文献的格式进行撰写，确保文档格式的规范性！

一、 实验内容或者文献情况介绍

熟悉编程环境

熟悉 BMP 图像的结构(不要调用接口)，编程实现 BMP 图像的阅读和显示

- 能获取图像任意一点的像素值
- 能将图像分成任意块大小，例如 4*4,8*8,16*16,32*32,64*64，并置乱块的位置并显示（类似马赛克效果）；能指定区域内的图像分块并置乱块的顺序再显示（本条可以调用软件或库的读图接口）

能阅读 wav 音频文件，并将原始的 PCM 音频数据显示出来，并画出其大小示意图（画出波形图）

- 读取 wav 音频文件，获取其数据的规律，并调用任何显示接口显示这种一维数据
- 统计 1 秒钟 wav 音频的数量跟哪些因素有关？

二、 算法简介及其实现细节

编程环境：

Windows 10

Python 3.7.0

库版本：

numpy:1.17.4

opencv2:4.2.0

matplotlib:3.0.2

图像部分：

对 bmp 图像的读取，定义 Bitmap 类，类中有两个内部类 BITMAPFILEHEADER 和 BITMAPINFOHEADER 分别完成对文件头和信息头的读取工作，两个内部类的具体代码如下：

BITMAPFILEHEADER 类：

```
1.  class BITMAPFILEHEADER:
2.      def __init__(self, filePath):
3.          with open(filePath, 'rb') as bitmap:
4.              #         bfType:2 字节，说明文件类型，一般为 19778，其转化为十六进制为 0x4d42，对应的字符串为 BM
5.              #         bfSize:4 字节，文件大小，以字节为单位
6.              #         bfReserved1:2 字节，保留，为 0
7.              #         bfReserved1:2 字节，保留，为 0
8.              #         bfOffBits:4 字节，从文件开始处到像素数据的偏移，也就是这两个结构体大小之和
9.              self.bfType = unpack('<h', bitmap.read(2))[0]
10.             self.bfSize = unpack('<i', bitmap.read(4))[0]
11.             self.bfReserved1 = unpack('<h', bitmap.read(2))[0]
12.             self.bfReserved2 = unpack('<h', bitmap.read(2))[0]
```

```
            self.bfOffBits = unpack('<i', bitmap.read(4))[0]
```

BITMAPINFOHEADER 类：

```
class BITMAPINFOHEADER:
    def __init__(self, filePath):
        with open(filePath, 'rb') as bitmap:
            bitmap.read(14)
            #         bisize:4 字节，此信息头大小
            #         biWidth:4 字节，图像的宽
            #         biHeight:4 字节，图像的高，正数代表位图为倒向，复数代表位图为正向，通常为正数
            #         biPlanes:2 字节，图像的帧数，一般为 1
            #         biBitcount:2 字节，一像素所占的位数，一般为 24
            #         biCompression:4 字节，说明图像数据压缩类型，一般为 0(不压缩)
            #         biSizeImage:4 字节，像素数据所占大小，说明图像的大小，以字节为单位。当压缩类型为 0 时，总设置为 0
            #         biXPelsPerMeter:4 字节，水平分辨率，用像素/米表示，有符号整数
            #         biYPelsPerMeter:4 字节，水平分辨率，用像素/米表示，有符号整数
            #         biClrUsed:4 字节，说明位图实际使用的彩色表中的颜色索引数，若设为 0 则说明使用所有调色板项
            #         biClrImportant:4 字节，说明对图像显示有重要影响的颜色索引的数目。若为 0，表示都重要
            self.biSize = unpack('<i', bitmap.read(4))[0]
            self.biWidth = unpack('<i', bitmap.read(4))[0]
            self.biHeight = unpack('<i', bitmap.read(4))[0]
            self.biPlanes = unpack('<h', bitmap.read(2))[0]
            self.biBitcount = unpack('<h', bitmap.read(2))[0]
            self.biCompression = unpack('<i', bitmap.read(4))[0]
            self.biSizeImage = unpack('<i', bitmap.read(4))[0]
            self.biXPelsPerMeter = unpack('<i', bitmap.read(4))[0]
            self.biYPelsPerMeter = unpack('<i', bitmap.read(4))[0]
            self.biClrUsed = unpack('<i', bitmap.read(4))[0]
            self.biClrImportant = unpack('<i', bitmap.read(4))[0]
```

在 Bitmap 类的__init__方法中调用之，获得文件的头部信息，共 54 字节，并将其打印，相关部分代码如下：

```
class Bitmap:
    def __init__(self, filePath):
        self.fileHeader = self.BITMAPFILEHEADER(filePath)
```

```
        self.infoHeader = self.BITMAPINFOHEADER(filePath)

    def printHeaderInfo(self):
        print('BITMAPFILEHEADER')
        print('bfType:', self.fileHeader.bfType)
        print('bfSize:', self.fileHeader.bfSize)
        print('bfReserved1:', self.fileHeader.bfReserved1)
        print('bfReserved2:', self.fileHeader.bfReserved2)
        print('bfOffBits:', self.fileHeader.bfOffBits)
        print('\nBITMAPINFOHEADER')
        print('biSize:', self.infoHeader.biSize)
        print('biWidth:', self.infoHeader.biWidth)
        print('biHeight:', self.infoHeader.biHeight)
        print('biPlanes:', self.infoHeader.biPlanes)
        print('biBitcount:', self.infoHeader.biBitcount)
        print('biCompression:', self.infoHeader.biCompression)
        print('biSizeImage:', self.infoHeader.biSizeImage)
        print('biXPelsPerMeter:', self.infoHeader.biXPelsPerMeter)
        print('biYPelsPerMeter:', self.infoHeader.biYPelsPerMeter)
        print('biClrUsed:', self.infoHeader.biClrUsed)
        print('biClrImportant:', self.infoHeader.biClrImportant)
```

至此，完成了文件头部信息的读取，开始读入 bmp 的图像数据。由于 4 字节对齐和倒图的原因，在读入图像数据的时候要做特殊的处理。

通过

```
if (self.width * 3 % 4) != 0:
            self.placeholder = 4 - (self.width * 3 % 4)
        else:
            self.placeholder = 0
```

获得每一行末尾的占位符 0x00（placeholder）的字节数.

在每一行的图像数据读完之后，再读入 placeholder 个字节，将占位符读走。同时，由于是小端保存的，所以在使用 struct.unpack 读取的时候，需要使用参数’<B’其中’<’代表小端序，B 代表一个字节的 bytes 数据，同时查阅资料可知，bmp 中每个像素的像素值是按 BGR 排列的，所以按序读入 B，G，R 通道中。由于图像是倒图保存的，而后续使用的图像显示库要求传入一个正图，故保存图像数据为 img 时倒序保存。如下：

```
        with open(filePath, 'rb') as bitmap:
            temp = bitmap.read(bfOffBits)
            for x in range(row):
                for y in range(column):
                    b = unpack('<B', bitmap.read(1))[0]
                    g = unpack('<B', bitmap.read(1))[0]
```

```
                    r = unpack('<B', bitmap.read(1))[0]
                    self.img[row - 1 - x][y] = (b, g, r)
                if (self.placeholder != 0):
                    temp = bitmap.read(self.placeholder)
```

完整的__init__方法如下所示：

```
class Bitmap:
    def __init__(self, filePath):
        self.fileHeader = self.BITMAPFILEHEADER(filePath)
        self.infoHeader = self.BITMAPINFOHEADER(filePath)

        self.height = self.infoHeader.biHeight
        self.width = self.infoHeader.biWidth
        bfOffBits = self.fileHeader.bfOffBits

        row = self.height
        column = self.width
        if (self.width * 3 % 4) != 0:
            self.placeholder = 4 - (self.width * 3 % 4)
        else:
            self.placeholder = 0
        self.img = np.ndarray((row, column), dtype=tuple)

        with open(filePath, 'rb') as bitmap:
            temp = bitmap.read(bfOffBits)
            for x in range(row):
                for y in range(column):
                    b = unpack('<B', bitmap.read(1))[0]
                    g = unpack('<B', bitmap.read(1))[0]
                    r = unpack('<B', bitmap.read(1))[0]
                    self.img[row - 1 - x][y] = (b, g, r)
                if (self.placeholder != 0):
                    temp = bitmap.read(self.placeholder)
        self.B, self.G, self.R = self.departBGR(self.img, row, column)
```

图像显示方法为定义在 Bitmap 类中的 showImg，调用该方法时会弹出一个弹窗，其中显示本次读入的图像，按下任意键可是弹窗消失，程序继续运行

```
def showImg(self):
    b = self.B
    g = self.G
    r = self.R
    merged = cv2.merge([b, g, r])
    cv2.imshow('any key to continue', merged)
```

```
        cv2.waitKey()
        cv2.destroyAllWindows()
```

获取图像任意点的像素值，使用者输入以左上角为原点，向右为 x 正方向，向下为 y 正方向的坐标，程序输出该点像素值（R，G，B）并打印在控制台。相应的方法定义在 Bitmap 中，名为 getPixel，如下所示

```
def getPixel(self):
    print('以左上角为原点，获得某像素点的 RGB 值:(R,G,B)')
    print('x 为 0 至%d 的整数，y 为 0 至%d 的整数
' % (self.width - 1, self.height - 1))
    while True:
        x = int(input('请输入 x:'))
        if x < 0 or x >= self.width:
            print('非法的 x')
            continue
        y = int(input('请输入 y:'))
        if y < 0 or y >= self.height:
            print('非法的 y')
            continue
        print('像素点(%d,%d)的 RGB 值为(%d,%d,%d)' %
              (x, y, self.R[x][y], self.G[x][y], self.B[x][y]))
```

将读入的图像写入到 output 文件夹中并保存为 fav.bmp。代码如下：

```
def writeImg(self):
    b = self.B
    g = self.G
    r = self.R
    merged = cv2.merge([b, g, r])
    cv2.imwrite("output/fav.bmp", merged)
```

这一过程通过如下代码运行：

```
# 读 bmp 图像
filePath = 'bmp/fav.bmp'
bitmap = Bitmap(filePath)
# 打印文件头的信息头的信息
bitmap.printHeaderInfo()
# 展示该图像
bitmap.showImg()
# 将读到的 bmp 图像重新写入到 output 文件夹中
bitmap.writeImg()
# 获取某点的 RGB 值
bitmap.getPixel()
```

对图像进行分块处理，读入图像并获取其长宽通道信息，接收用户输入，确定分块大小及范围

```
def slicing(filePath):
    img = cv2.imread(filePath)
    (y, x, alpha) = img.shape

    while True:
        chooseX = input('请输入选区宽度，不输入为全图%d 像素:'%(x))
        if chooseX != '' and (int(chooseX) > x or int(chooseX) <= 0):
            print('输入非法！')
            continue
        chooseY = input('请输入选取高度，不输入为全图%d 像素:'%(y))
        if chooseY != '' and (int(chooseY) > y or int(chooseY) <= 0):
            print('输入非法！')
            continue
        chooseM = int(input('请输入图块宽度:'))
        if chooseM > x or chooseM <= 0:
            print('输入非法！')
            continue
        chooseN = int(input('请输入图块高度:'))
        if chooseN > y or chooseN <= 0:
            print('输入非法！')
            continue
        break
    if chooseX != '':
        x = int(chooseX)
    if chooseY != '':
        y = int(chooseY)

    m = chooseM
    n = chooseN
```

由于 cv2 读入的图像保存为 np.ndarray 格式，可以直接通过切片的方式将图像进行分块，具体的代码如下：

```
matrix = np.zeros((slicingY, slicingX), dtype=list)
for i in range(slicingY):
    for j in range(slicingX):
        matrix[i][j] = img[nowY:nextY, nowX:nextX].copy()
        nowX = nowX + atomX
        nextX = min(nowX + atomX, x)
    nowY = nowY + atomY
    nextY = min(nextY + atomY, y)
    nowX = 0
```

```
10.     nextX = min(nowX + atomX, x)
```

将分块之后的图像信息保存在 matrix 中，matrix 为一个矩阵，矩阵中的每个元素为每个以数组形式保存的图像块。

定义一个元素个数为图像所分块个数的一维数组，使用 np.random.permutation 方法将其打乱，然后使用 reshape 方法将其变成与 matrix 具有相同形状的矩阵 sort。将分块之后的每一块按照 sort 中的顺序进行重新排列，获得置乱之后的图像块。考虑到可能会有不够分块大小的块，若打乱之可能会使图像变形，故而这些块将不参加打乱过程，保持其原来的位置不变。整个过程的代码如下：

```
1.  preShuffleMatrix = matrix[0:slicingY, 0:slicingX].copy()
2.  shuffleMatrix = preShuffleMatrix.copy()
3.  sort = np.random.permutation(np.array(range(slicingY * slicingX)))
4.  sort = sort.reshape((slicingY, slicingX))
5.  for i in range(slicingY):
6.      for j in range(slicingX):
7.          index = sort[i][j]
8.          shuffleMatrix[i][j] = preShuffleMatrix[int(
9.              index / slicingX)][index % slicingX]
```

再将其组合成跟原来图像大小一致的图像，并显示，分块并打乱的方法的完整定义如下：

```
1.  def slicing(filePath):
2.      img = cv2.imread(filePath)
3.      (y, x, alpha) = img.shape
4.
5.      while True:
6.          chooseX = input('请输入选区宽度，不输入为全图%d 像素:'%(x))
7.          if chooseX != '' and (int(chooseX) > x or int(chooseX) <= 0):
8.              print('输入非法！')
9.              continue
10.         chooseY = input('请输入选取高度，不输入为全图%d 像素:'%(y))
11.         if chooseY != '' and (int(chooseY) > y or int(chooseY) <= 0):
12.             print('输入非法！')
13.             continue
14.         chooseM = int(input('请输入图块宽度:'))
15.         if chooseM > x or chooseM <= 0:
16.             print('输入非法！')
17.             continue
18.         chooseN = int(input('请输入图块高度:'))
19.         if chooseN > y or chooseN <= 0:
20.             print('输入非法！')
21.             continue
```

```
        break
    if chooseX != '':
        x = int(chooseX)
    if chooseY != '':
        y = int(chooseY)

    m = chooseM
    n = chooseN

    atomX = m
    atomY = n
    slicingY = int(y / atomY)
    slicingX = int(x / atomX)
    #     shuffleY = slicingY
    #     shuffleX = slicingX
    #     if y % atomY != 0:
    #         slicingY += 1
    #     if x % atomX != 0:
    #         slicingX += 1
    nowX = 0
    nowY = 0
    nextX = min(nowX + atomX, x)
    nextY = min(nowY + atomY, y)
    matrix = np.zeros((slicingY, slicingX), dtype=list)
    for i in range(slicingY):
        for j in range(slicingX):
            matrix[i][j] = img[nowY:nextY, nowX:nextX].copy()
            nowX = nowX + atomX
            nextX = min(nowX + atomX, x)
        nowY = nowY + atomY
        nextY = min(nextY + atomY, y)
        nowX = 0
        nextX = min(nowX + atomX, x)


#     cv2.imshow('slicing',matrix[6][9])
#     cv2.moveWindow('slicing',400,400)
#     cv2.waitKey()
#     cv2.destroyAllWindows()

#     shuffleMatrix = np.random.permutation(matrix[0:shuffleY, 0:shuffleX])

    preShuffleMatrix = matrix[0:slicingY, 0:slicingX].copy()
    shuffleMatrix = preShuffleMatrix.copy()
```

```
    sort = np.random.permutation(np.array(range(slicingY * slicingX)))
    sort = sort.reshape((slicingY, slicingX))
    for i in range(slicingY):
        for j in range(slicingX):
            index = sort[i][j]
            shuffleMatrix[i][j] = preShuffleMatrix[int(
                index / slicingX)][index % slicingX]

    shuffle = img.copy()
    for i in range(slicingY * atomY):
        for j in range(slicingX * atomX):
            for k in range(alpha):
                shuffle[i][j][k] = shuffleMatrix[int(i / atomY)][int(
                    j / atomX)][i % atomY][j % atomX][k]
    cv2.imwrite('output/shuffle.bmp', shuffle)

    cv2.imshow('any key to continue',shuffle)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

图像部分的完整代码如下：

```
from struct import unpack
import numpy as np
import cv2


class Bitmap:
    def __init__(self, filePath):
        self.fileHeader = self.BITMAPFILEHEADER(filePath)
        self.infoHeader = self.BITMAPINFOHEADER(filePath)

        self.height = self.infoHeader.biHeight
        self.width = self.infoHeader.biWidth
        bfOffBits = self.fileHeader.bfOffBits

        row = self.height
        column = self.width
        if (self.width * 3 % 4) != 0:
            self.placeholder = 4 - (self.width * 3 % 4)
        else:
            self.placeholder = 0
        self.img = np.ndarray((row, column), dtype=tuple)

        with open(filePath, 'rb') as bitmap:
```

```
            temp = bitmap.read(bfOffBits)
            for x in range(row):
                for y in range(column):
                    b = unpack('<B', bitmap.read(1))[0]
                    g = unpack('<B', bitmap.read(1))[0]
                    r = unpack('<B', bitmap.read(1))[0]
                    self.img[row - 1 - x][y] = (b, g, r)
                if (self.placeholder != 0):
                    temp = bitmap.read(self.placeholder)
        self.B, self.G, self.R = self.departBGR(self.img, row, column)

    def departBGR(self, img, row, column):
        B = np.zeros_like(img, dtype=np.uint8)
        G = np.zeros_like(img, dtype=np.uint8)
        R = np.zeros_like(img, dtype=np.uint8)
        for x in range(row):
            for y in range(column):
                BGR = img[x][y]
                B[x][y] = BGR[0]
                G[x][y] = BGR[1]
                R[x][y] = BGR[2]
        return B, G, R

    def getPixel(self):
        print('以左上角为原点，获得某像素点的 RGB 值:(R,G,B)')
        print('x 为 0 至%d 的整数，y 为 0 至%d 的整数
    ' % (self.width - 1, self.height - 1))
        while True:
            x = int(input('请输入 x:'))
            if x < 0 or x >= self.width:
                print('非法的 x')
                continue
            y = int(input('请输入 y:'))
            if y < 0 or y >= self.height:
                print('非法的 y')
                continue
            print('像素点(%d,%d)的 RGB 值为(%d,%d,%d)' %
                  (x, y, self.R[x][y], self.G[x][y], self.B[x][y]))
            break

    def writeImg(self):
        b = self.B
        g = self.G
        r = self.R
```

```
67.         merged = cv2.merge([b, g, r])
68.         cv2.imwrite("output/fav.bmp", merged)
69.
70.     def showImg(self):
71.         b = self.B
72.         g = self.G
73.         r = self.R
74.         merged = cv2.merge([b, g, r])
75.         cv2.imshow('any key to continue', merged)
76.         cv2.waitKey()
77.         cv2.destroyAllWindows()
78.
79.     def printHeaderInfo(self):
80.         print('BITMAPFILEHEADER')
81.         print('bfType:', self.fileHeader.bfType)
82.         print('bfSize:', self.fileHeader.bfSize)
83.         print('bfReserved1:', self.fileHeader.bfReserved1)
84.         print('bfReserved2:', self.fileHeader.bfReserved2)
85.         print('bfOffBits:', self.fileHeader.bfOffBits)
86.         print('\nBITMAPINFOHEADER')
87.         print('biSize:', self.infoHeader.biSize)
88.         print('biWidth:', self.infoHeader.biWidth)
89.         print('biHeight:', self.infoHeader.biHeight)
90.         print('biPlanes:', self.infoHeader.biPlanes)
91.         print('biBitcount:', self.infoHeader.biBitcount)
92.         print('biCompression:', self.infoHeader.biCompression)
93.         print('biSizeImage:', self.infoHeader.biSizeImage)
94.         print('biXPelsPerMeter:', self.infoHeader.biXPelsPerMeter)
95.         print('biYPelsPerMeter:', self.infoHeader.biYPelsPerMeter)
96.         print('biClrUsed:', self.infoHeader.biClrUsed)
97.         print('biClrImportant:', self.infoHeader.biClrImportant)
98.
99.     class BITMAPFILEHEADER:
100.         def __init__(self, filePath):
101.             with open(filePath, 'rb') as bitmap:
102.                 #         bfType:2 字节，说明文件类型，一般为 19778，其转化为十六
    进制为 0x4d42，对应的字符串为 BM
103.                 #         bfSize:4 字节，文件大小，以字节为单位
104.                 #         bfReserved1:2 字节，保留，为 0
105.                 #         bfReserved1:2 字节，保留，为 0
106.                 #         bfOffBits:4 字节，从文件开始处到像素数据的偏移，也就是
    这两个结构体大小之和
107.                 self.bfType = unpack('<h', bitmap.read(2))[0]
108.                 self.bfSize = unpack('<i', bitmap.read(4))[0]
```

```
109.                 self.bfReserved1 = unpack('<h', bitmap.read(2))[0]
110.                 self.bfReserved2 = unpack('<h', bitmap.read(2))[0]
111.                 self.bfOffBits = unpack('<i', bitmap.read(4))[0]
112.
113.     class BITMAPINFOHEADER:
114.         def __init__(self, filePath):
115.             with open(filePath, 'rb') as bitmap:
116.                 bitmap.read(14)
117.                 #         bisize:4 字节，此信息头大小
118.                 #         biWidth:4 字节，图像的宽
119.                 #         biHeight:4 字节，图像的高，正数代表位图为倒向，复数代表位图为正向，通常为正数
120.                 #         biPlanes:2 字节，图像的帧数，一般为 1
121.                 #         biBitcount:2 字节，一像素所占的位数，一般为 24
122.                 #         biCompression:4 字节，说明图像数据压缩类型，一般为 0(不压缩)
123.                 #         biSizeImage:4 字节，像素数据所占大小，说明图像的大小，以字节为单位。当压缩类型为 0 时，总设置为 0
124.                 #         biXPelsPerMeter:4 字节，水平分辨率，用像素/米表示，有符号整数
125.                 #         biYPelsPerMeter:4 字节，水平分辨率，用像素/米表示，有符号整数
126.                 #         biClrUsed:4 字节，说明位图实际使用的彩色表中的颜色索引数，若设为 0 则说明使用所有调色板项
127.                 #         biClrImportant:4 字节，说明对图像显示有重要影响的颜色索引的数目。若为 0，表示都重要
128.                 self.biSize = unpack('<i', bitmap.read(4))[0]
129.                 self.biWidth = unpack('<i', bitmap.read(4))[0]
130.                 self.biHeight = unpack('<i', bitmap.read(4))[0]
131.                 self.biPlanes = unpack('<h', bitmap.read(2))[0]
132.                 self.biBitcount = unpack('<h', bitmap.read(2))[0]
133.                 self.biCompression = unpack('<i', bitmap.read(4))[0]
134.                 self.biSizeImage = unpack('<i', bitmap.read(4))[0]
135.                 self.biXPelsPerMeter = unpack('<i', bitmap.read(4))[0]
136.                 self.biYPelsPerMeter = unpack('<i', bitmap.read(4))[0]
137.                 self.biClrUsed = unpack('<i', bitmap.read(4))[0]
138.                 self.biClrImportant = unpack('<i', bitmap.read(4))[0]
139.
140.
141. # 读 bmp 图像
142. filePath = 'bmp/fav.bmp'
143. bitmap = Bitmap(filePath)
144. # 打印文件头的信息头的信息
145. bitmap.printHeaderInfo()
```

```
# 展示该图像
bitmap.showImg()
# 将读到的 bmp 图像重新写入到 output 文件夹中
bitmap.writeImg()
# 获取某点的 RGB 值
bitmap.getPixel()


filePath = 'bmp/fav.bmp'

def slicing(filePath):
    img = cv2.imread(filePath)
    (y, x, alpha) = img.shape

    while True:
        chooseX = input('请输入选区宽度，不输入为全图%d 像素:'%(x))
        if chooseX != '' and (int(chooseX) > x or int(chooseX) <= 0):
            print('输入非法！')
            continue
        chooseY = input('请输入选取高度，不输入为全图%d 像素:'%(y))
        if chooseY != '' and (int(chooseY) > y or int(chooseY) <= 0):
            print('输入非法！')
            continue
        chooseM = int(input('请输入图块宽度:'))
        if chooseM > x or chooseM <= 0:
            print('输入非法！')
            continue
        chooseN = int(input('请输入图块高度:'))
        if chooseN > y or chooseN <= 0:
            print('输入非法！')
            continue
        break
    if chooseX != '':
        x = int(chooseX)
    if chooseY != '':
        y = int(chooseY)

    m = chooseM
    n = chooseN

    atomX = m
    atomY = n
    slicingY = int(y / atomY)
    slicingX = int(x / atomX)
```

```
    #     shuffleY = slicingY
    #     shuffleX = slicingX
    #     if y % atomY != 0:
    #         slicingY += 1
    #     if x % atomX != 0:
    #         slicingX += 1
    nowX = 0
    nowY = 0
    nextX = min(nowX + atomX, x)
    nextY = min(nowY + atomY, y)
    matrix = np.zeros((slicingY, slicingX), dtype=list)
    for i in range(slicingY):
        for j in range(slicingX):
            matrix[i][j] = img[nowY:nextY, nowX:nextX].copy()
            nowX = nowX + atomX
            nextX = min(nowX + atomX, x)
        nowY = nowY + atomY
        nextY = min(nextY + atomY, y)
        nowX = 0
        nextX = min(nowX + atomX, x)


#     cv2.imshow('slicing',matrix[6][9])
#     cv2.moveWindow('slicing',400,400)
#     cv2.waitKey()
#     cv2.destroyAllWindows()

#     shuffleMatrix = np.random.permutation(matrix[0:shuffleY, 0:shuffleX])

    preShuffleMatrix = matrix[0:slicingY, 0:slicingX].copy()
    shuffleMatrix = preShuffleMatrix.copy()
    sort = np.random.permutation(np.array(range(slicingY * slicingX)))
    sort = sort.reshape((slicingY, slicingX))
    for i in range(slicingY):
        for j in range(slicingX):
            index = sort[i][j]
            shuffleMatrix[i][j] = preShuffleMatrix[int(
                index / slicingX)][index % slicingX]

    shuffle = img.copy()
    for i in range(slicingY * atomY):
        for j in range(slicingX * atomX):
            for k in range(alpha):
                shuffle[i][j][k] = shuffleMatrix[int(i / atomY)][int(
```

```
                    j / atomX)][i % atomY][j % atomX][k]
    cv2.imwrite('output/shuffle.bmp', shuffle)

    cv2.imshow('any key to continue',shuffle)
    cv2.waitKey()
    cv2.destroyAllWindows()

slicing(filePath)
```

音频部分：

定义 Wave 类，使用该类读取音频文件及其各项信息并打印之，相关代码如下：

```
class Wave:
    def __init__(self, filePath):
        self.audio = we.open(filePath, 'rb')
        nchannels = self.audio.getnchannels()
        sampwidth = self.audio.getsampwidth()
        self.framerate = self.audio.getframerate()
        self.nframes = self.audio.getnframes()
        comptype = self.audio.getcomptype()
        compname = self.audio.getcompname()
        self.params = self.audio.getparams()
        self.dataWav = self.audio.readframes(self.nframes)
        self.secs = self.nframes / self.framerate
        self.audio.close()

    def getInfo(self):
        print(self.params)
```

逐帧读取音频，将获得的音频信息转化为 short 类型保存在矩阵中，由于使用的音频为双通道音频，故将获得的数组使用 reshape 函数变形为 2 行，n 列的矩阵。为每一帧计算其所在的时刻，记录为 time，调用 matplotlib.pyplot 库将其显示输出，由于所使用的音频文件较长，故选取了中间的 10000 帧进行画图，便于观察效果，该部分代码如下：

```
def drawWav(self):
    dataUse = np.fromstring(self.dataWav, dtype=np.short)
    dataUse = dataUse.reshape((-1, 2))
    dataUse = dataUse.T
    time = np.arange(0, self.nframes) * (1.0 / self.framerate)
    plt.subplot(211)
```

```
        plt.plot(time[8450000:8460000],
                 dataUse[0][8450000:8460000],
                 color='green')
        plt.subplot(212)
        plt.plot(time[8450000:8460000], dataUse[1][8450000:8460000])
        plt.show()
```

研究一秒钟音频的数量跟什么因素有关，通过信息可知，该音频总共约有 382 秒，采样率为 44.1kHz，总帧数为 16882607 粗略计算可得

nframes / second
= 16882607 / 382
= 44100
= framerate

故每秒音频的数量与采样率有关，验证的代码如下：

```
def confirm(self):
    print('  nframes / second\n= %d / %d\n= %d\n= framerate' %
          (self.nframes, self.secs, self.framerate))
    print('由于 nframes / framerate = seconds, 故 1 秒钟 wav 音频的数量与采样率有关')
```

整个音频部分的完整代码如下：

```
import wave as we
import matplotlib.pyplot as plt
import numpy as np


class Wave:
    def __init__(self, filePath):
        self.audio = we.open(filePath, 'rb')
        nchannels = self.audio.getnchannels()
        sampwidth = self.audio.getsampwidth()
        self.framerate = self.audio.getframerate()
        self.nframes = self.audio.getnframes()
        comptype = self.audio.getcomptype()
        compname = self.audio.getcompname()
        self.params = self.audio.getparams()
        self.dataWav = self.audio.readframes(self.nframes)
        self.secs = self.nframes / self.framerate
        self.audio.close()

    def drawWav(self):
        dataUse = np.fromstring(self.dataWav, dtype=np.short)
        dataUse = dataUse.reshape((-1, 2))
```

```
        dataUse = dataUse.T
        time = np.arange(0, self.nframes) * (1.0 / self.framerate)
        plt.subplot(211)
        plt.plot(time[8450000:8460000],
                 dataUse[0][8450000:8460000],
                 color='green')
        plt.subplot(212)
        plt.plot(time[8450000:8460000], dataUse[1][8450000:8460000])
        plt.show()

    def getInfo(self):
        print(self.params)

    def confirm(self):
        print('  nframes / second\n= %d / %d\n= %d\n= framerate' %
              (self.nframes, self.secs, self.framerate))
        print('由于 nframes / framerate = seconds, 故 1 秒钟 wav 音频的数量与采样率有关')


filePath = 'wav/fav.wav'
wav = Wave(filePath)
wav.getInfo()
wav.drawWav()
wav.confirm()
```

三、 实验设置及结果分析（包括实验数据集）

图像部分：

使用样本图片为 bmp/fav.bmp

bmp/fav.bmp

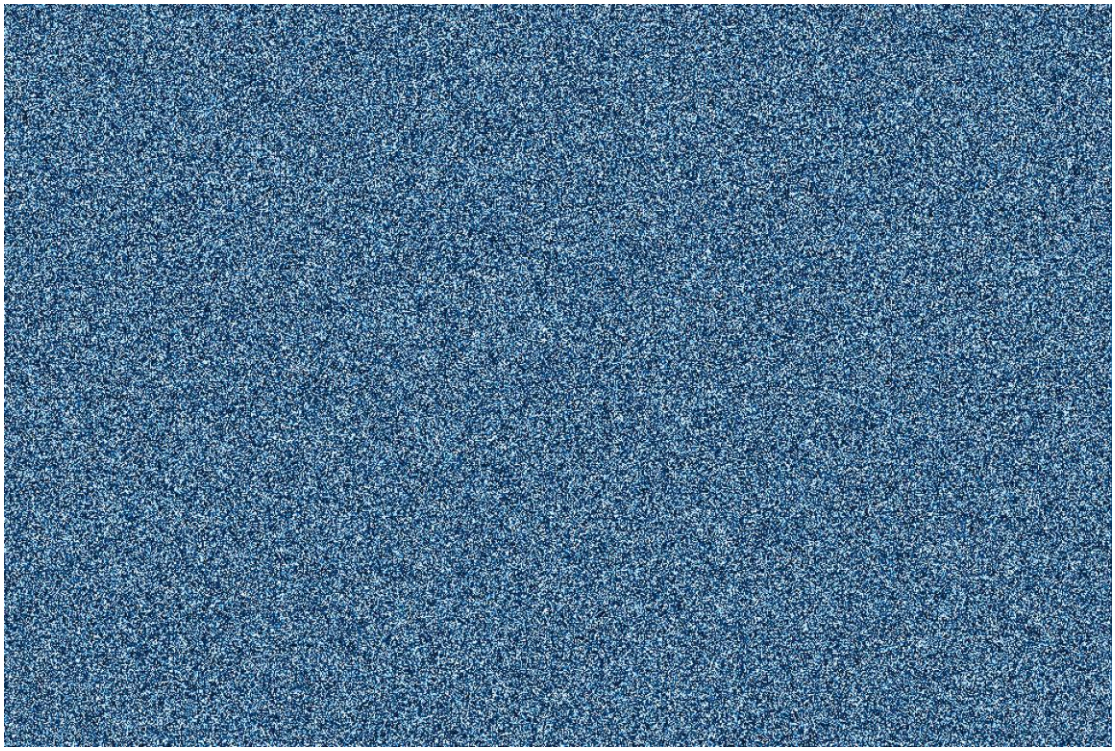
读得其各项信息为
BITMAPFILEHEADER
bfType: 19778
bfSize: 7478150
bfReserved1: 0
bfReserved2: 0
bfOffBits: 54

BITMAPINFOHEADER
biSize: 40
biWidth: 1929
biHeight: 1292
biPlanes: 1
biBitcount: 24
biCompression: 0
biSizeImage: 0
biXPelsPerMeter: 2835
biYPelsPerMeter: 2835
biClrUsed: 0
biClrImportant: 0
与实际相符

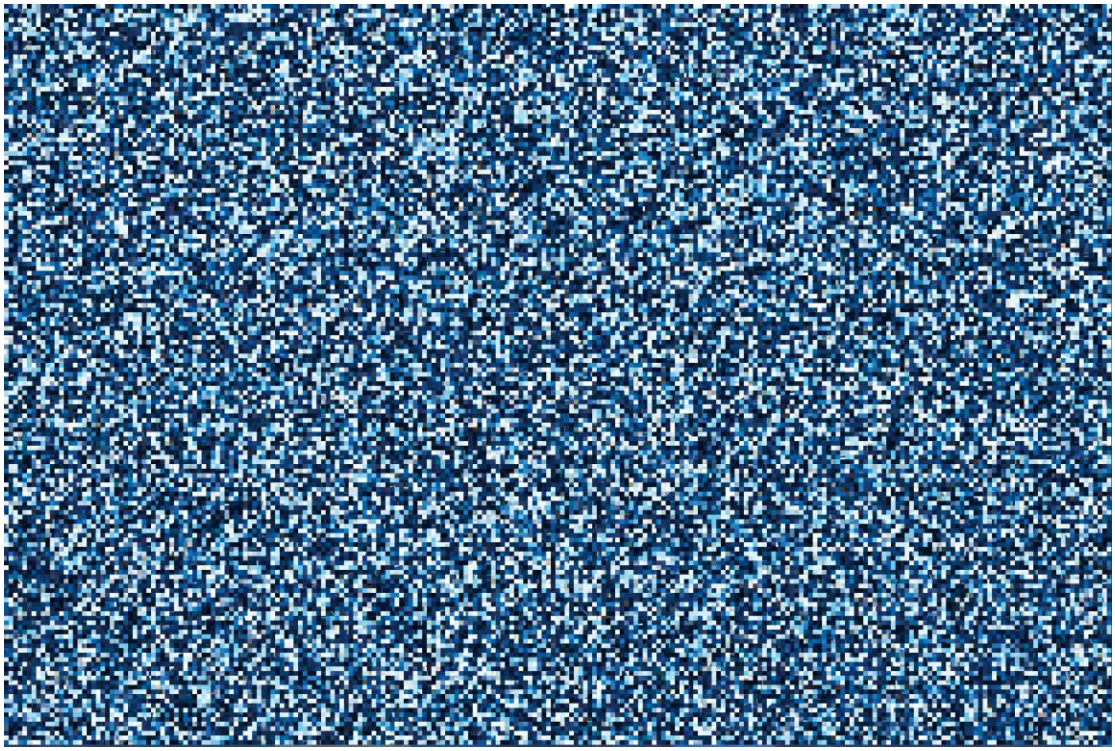
实验中获取了点(33, 33)的像素值，控制台输出如下：
以左上角为原点，获得某像素点的 RGB 值:(R,G,B)
x 为 0 至 1928 的整数，y 为 0 至 1291 的整数

请输入 x:33
请输入 y:33
像素点(33,33)的 RGB 值为(180,225,248)

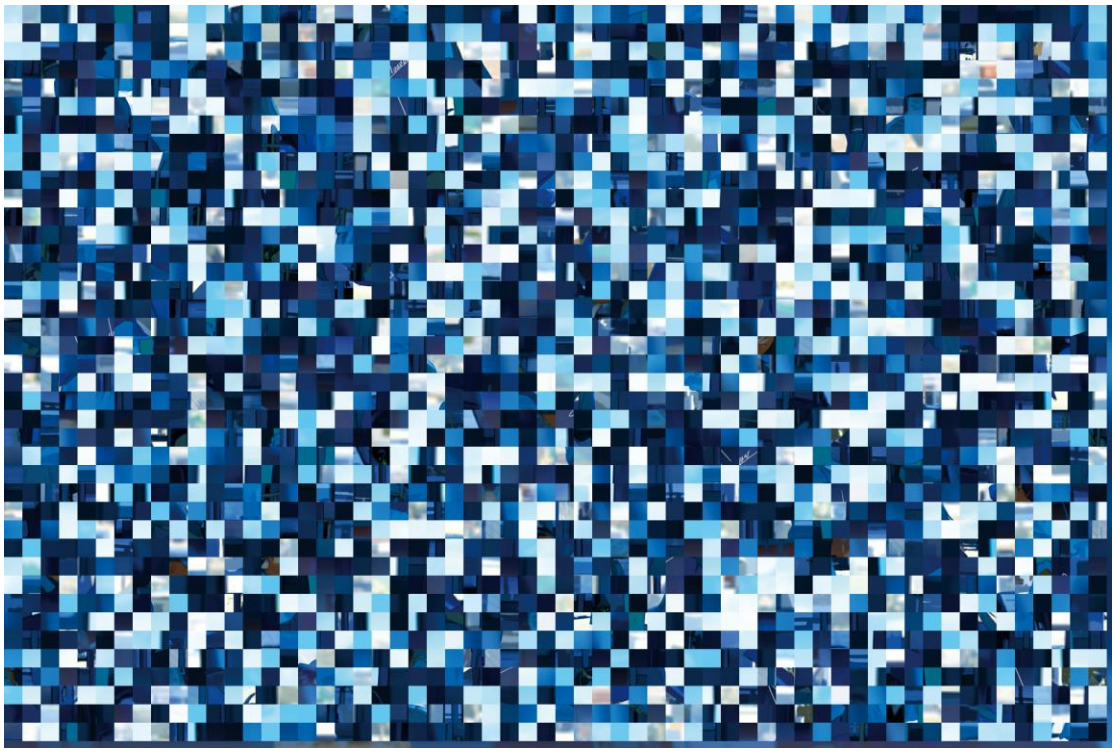
将图像分割成任意块大小并打乱再输出：



2x2



8x8



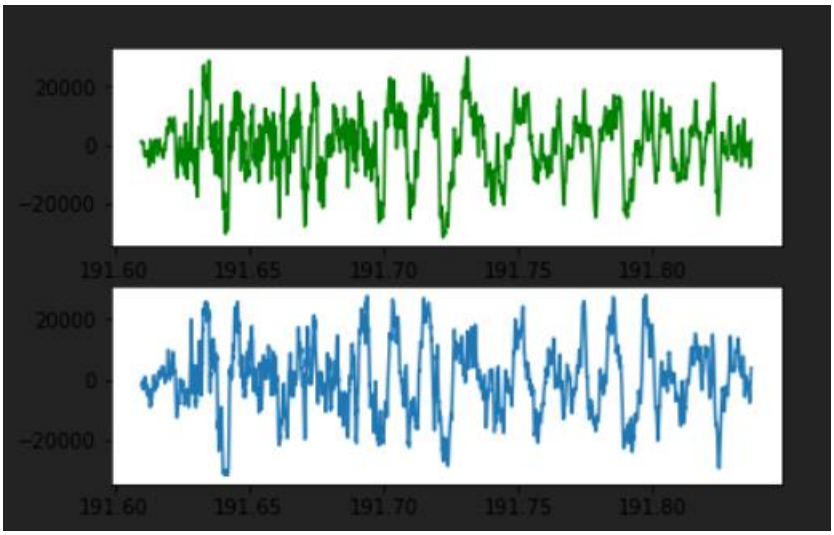
32x32

256x256

选区分块打乱，选取左上角 1000x1000 大小的图像，将其分成 256x256 大小的块，并打乱，效果如下：

选区 1000x1000, 分块大小 256x256

音频部分
读取的音频的基本信息如下：
_wave_params(nchannels=2, sampwidth=2, framerate=44100, nframes=16882607, comptype='NONE', compname='not compressed')
音频双声道选取的[8450000:8460000]部分帧的显示效果如下：



音频频谱

每秒音频数与采样率有关的证明过程如下：
nframes / second
= 16882607 / 382
= 44100

= framerate

由于 nframes / framerate = seconds, 故 1 秒钟 wav 音频的数量与采样率有关

四、 结论

bmp 格式图片是一种无压缩的图片格式，对于后续对各种其他图片格式的学习有着至关重要的作用，可以说，无法读懂 bmp 文件，就无法读懂其他格式的图片。bmp 格式的文件一般由三部分组成，分别是文件头（14 字节），信息头（40 字节）和图片数据，对于 8 位的 bmp 图像还有调色板。文件头包含 bmp 作为一个文件最基本的信息，大小，头部大小，类型等，信息头包含 bmp 作为一个图片的基本信息，如宽高，颜色深度，帧数，分辨率等。bmp 格式文件中图像数据以小端保存并且是倒图（上下翻转）的形式。同时由于读取文件的特殊性，以 4 字节为单位读取速度更快，内存更小，故图像数据中还有对齐机制，当图像一行的字节数不是 4 的整数倍时，需要在每行末尾填充若干为 0 的字节使每行的字节数为 4 的整数倍，在读取文件时需要注意这点，否则图像会倾斜。bmp 格式图片的图像数据是按一个个像素点的像素值来保存的，且像素值以(B,G,R)的顺序存储。

wav 格式音频是一种无压缩的音频格式，在文件头中包含有帧数采样率通道数等信息。采样率为 44.1kHz 意味着每秒对原信号采用 44.1 个点，来组成数字信号，决定了音频每秒钟的帧数大小。

五、 参考文献

bmp 格式文件结构

wav 格式文件结构

opencv2 文档

numpy 文档