

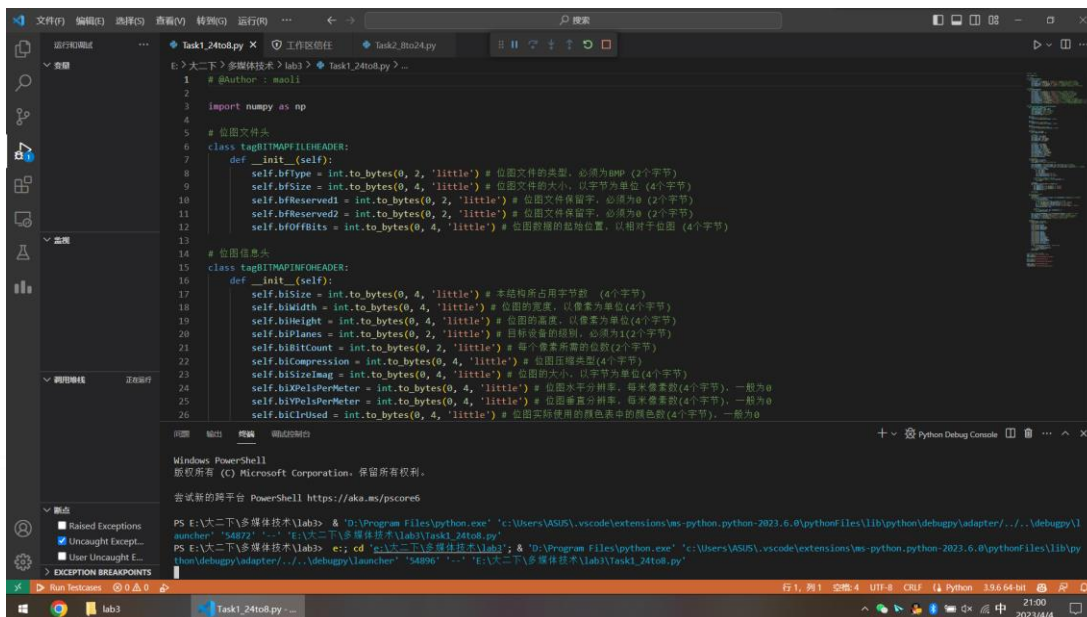
《多媒体技术》实验报告 3

黄勛 22920212204392

1. 运行程序截图和简要说明

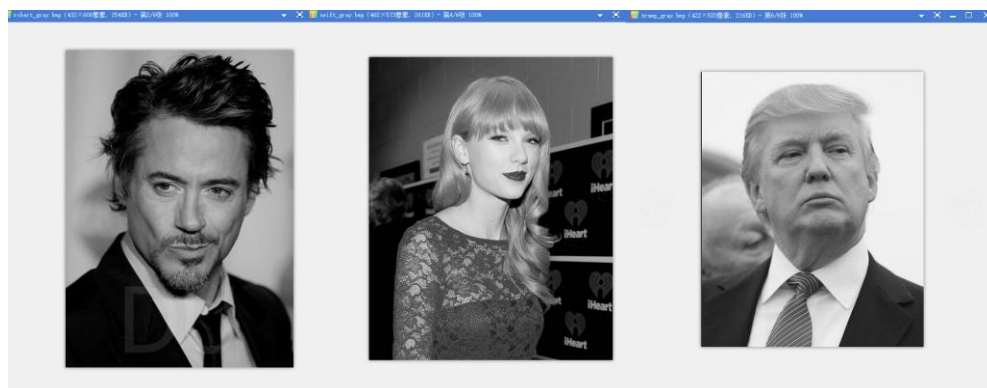
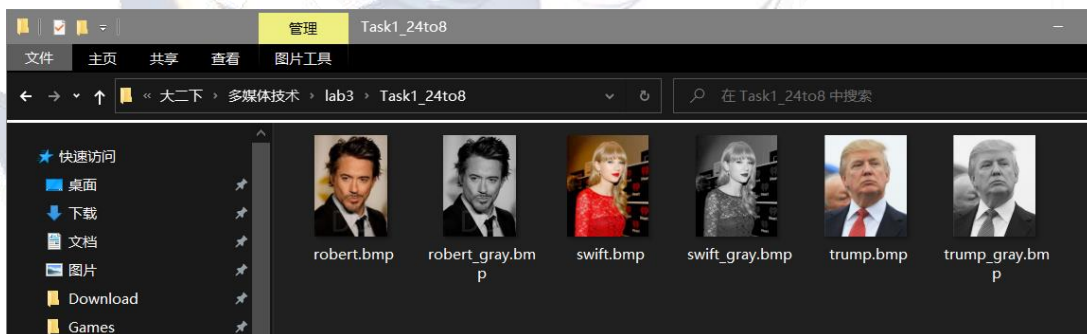
1) 读取 24 位真彩色 bmp 位图 (3 张) 转化位 8 位伪彩色 bmp 灰度位图

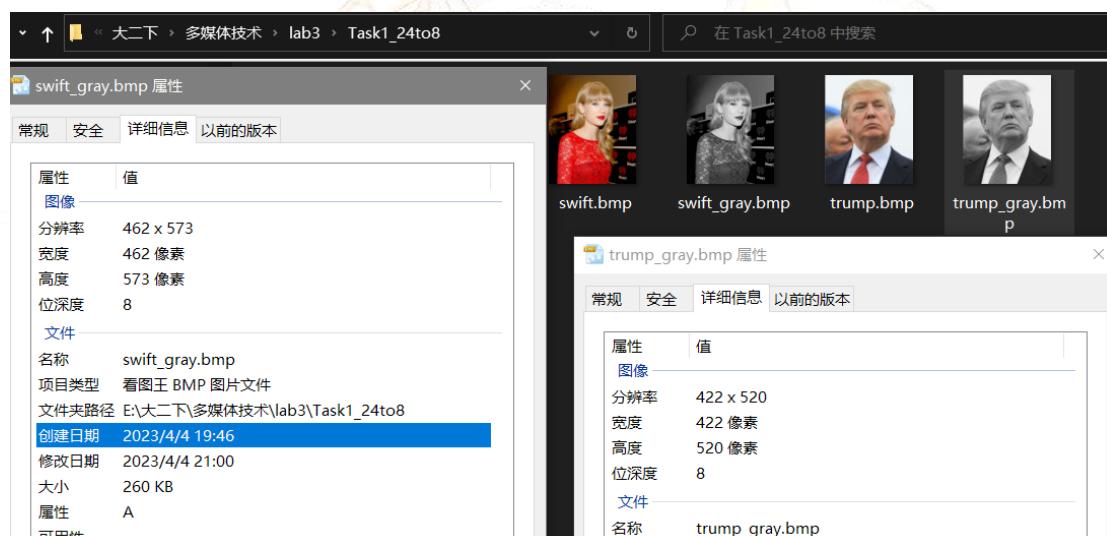
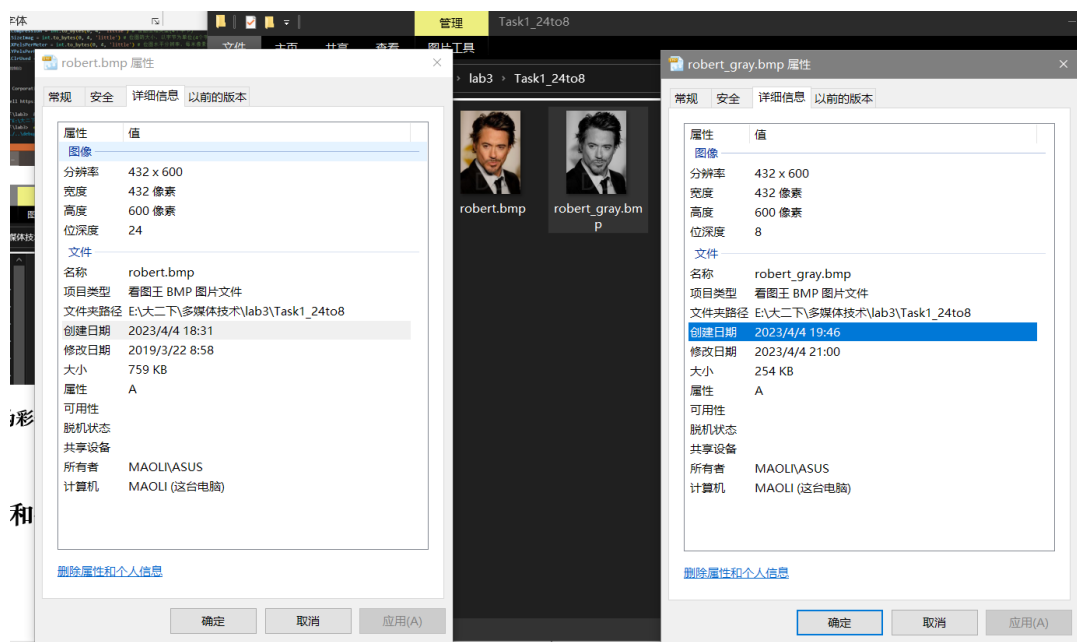
运行:



```
1 # @Author : masli
2
3 import numpy as np
4
5 # 位图文件头
6 class tagBITMAPFILEHEADER:
7     def __init__(self):
8         self.bfType = int.to_bytes(0, 2, 'little') # 位图文件的类型, 必须为BMP (2个字节)
9         self.bfSize = int.to_bytes(0, 4, 'little') # 位图文件的大小, 以字节为单位 (4个字节)
10        self.bfReserved1 = int.to_bytes(0, 2, 'little') # 位图文件保留字, 必须为0 (2个字节)
11        self.bfReserved2 = int.to_bytes(0, 2, 'little') # 位图文件保留字, 必须为0 (2个字节)
12        self.bfOffBits = int.to_bytes(0, 4, 'little') # 位图数据的起始位置, 以相对于位图 (4个字节)
13
14 # 位图信息头
15 class tagBITMAPINFOHEADER:
16     def __init__(self):
17         self.biSize = int.to_bytes(0, 4, 'little') # 本结构所占字节数 (4个字节)
18         self.biWidth = int.to_bytes(0, 4, 'little') # 位图的宽度, 以像素为单位 (4个字节)
19         self.biHeight = int.to_bytes(0, 4, 'little') # 位图的高度, 以像素为单位 (4个字节)
20         self.biPlanes = int.to_bytes(0, 2, 'little') # 目标设备的平面, 必须为1 (2个字节)
21         self.biBitCount = int.to_bytes(0, 2, 'little') # 每个像素所需的位数 (2个字节)
22         self.biCompression = int.to_bytes(0, 4, 'little') # 位图压缩类型 (4个字节)
23         self.biSizeImg = int.to_bytes(0, 4, 'little') # 位图的大小, 以字节为单位 (4个字节)
24         self.biXpelsPerMeter = int.to_bytes(0, 4, 'little') # 位图水平分辨率, 每米像素数 (4个字节), 一般为0
25         self.biYpelsPerMeter = int.to_bytes(0, 4, 'little') # 位图垂直分辨率, 每米像素数 (4个字节), 一般为0
26         self.biClrUsed = int.to_bytes(0, 4, 'little') # 位图实际使用的颜色表中的颜色数 (4个字节), 一般为0
```

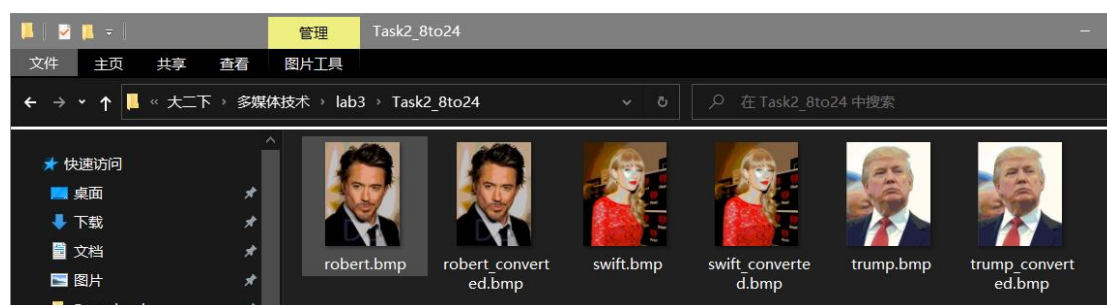
生成结果:

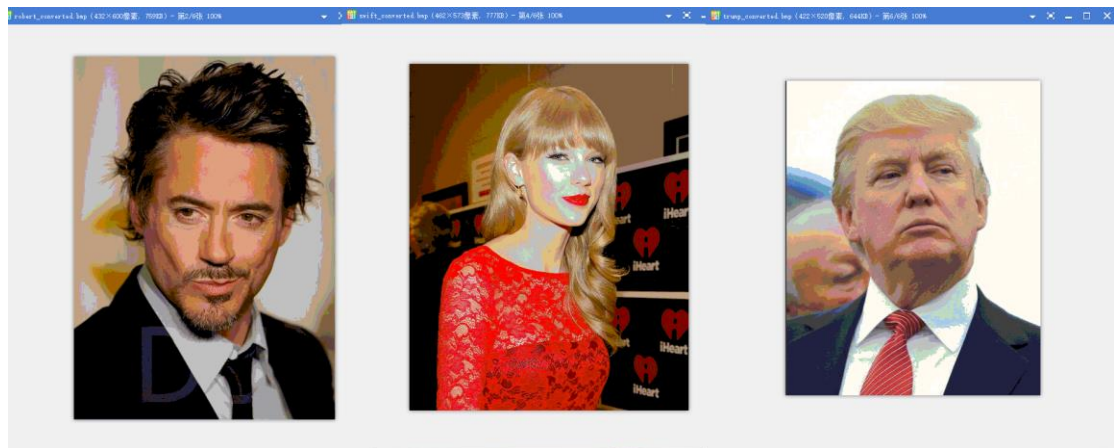




2) 读取 8 位伪彩色 bmp 位图 (3 张) 转化为 24 位真彩色 bmp 位图

生成结果:





2.主要代码展示和分析

1) 面向对象方法设计

位图文件头:

```
Task1_24to8.py • Task2_8to24.py •
E:\大二下\多媒体技术\lab3> Task1_24to8.py Bmp > Parse
1 # @Author : maoli
2
3 import numpy as np
4
5 # 位图文件头
6 class tagBITMAPFILEHEADER:
7     def __init__(self):
8         self.bfType = int.to_bytes(0, 2, 'little') # 位图文件的类型, 必须为BMP (2个字节)
9         self.bfSize = int.to_bytes(0, 4, 'little') # 位图文件的大小, 以字节为单位 (4个字节)
10        self.bfReserved1 = int.to_bytes(0, 2, 'little') # 位图文件保留字, 必须为0 (2个字节)
11        self.bfReserved2 = int.to_bytes(0, 2, 'little') # 位图文件保留字, 必须为0 (2个字节)
12        self.bfOffBits = int.to_bytes(0, 4, 'little') # 位图数据的起始位置, 以相对于位图 (4个字节)
13
```

位图信息头:

```
15 class tagBITMAPINFOHEADER:
16     def __init__(self):
17         self.biSize = int.to_bytes(0, 4, 'little') # 本结构所占字节数 (4个字节)
18         self.biWidth = int.to_bytes(0, 4, 'little') # 位图的宽度, 以像素为单位(4个字节)
19         self.biHeight = int.to_bytes(0, 4, 'little') # 位图的高度, 以像素为单位(4个字节)
20         self.biPlanes = int.to_bytes(0, 2, 'little') # 目标设备的级别, 必须为1(2个字节)
21         self.biBitCount = int.to_bytes(0, 2, 'little') # 每个像素所需的位数(2个字节)
22         self.biCompression = int.to_bytes(0, 4, 'little') # 位图压缩类型(4个字节)
23         self.biSizeImag = int.to_bytes(0, 4, 'little') # 位图的大小, 以字节为单位(4个字节)
24         self.biXPelsPerMeter = int.to_bytes(0, 4, 'little') # 位图水平分辨率, 每米像素数(4个字节),
25         self.biYPelsPerMeter = int.to_bytes(0, 4, 'little') # 位图垂直分辨率, 每米像素数(4个字节),
26         self.biClrUsed = int.to_bytes(0, 4, 'little') # 位图实际使用的颜色表中的颜色数(4个字节), 一
27         self.biClrImportant = int.to_bytes(0, 4, 'little') # 位图显示过程中重要的颜色数(4个字节), 一
```

主要实现: 位图类

基本数据处理:

```
30 class Bmp(tagBITMAPFILEHEADER, tagBITMAPINFOHEADER):
31     def __init__(self, file_name):
32         tagBITMAPFILEHEADER.__init__(self)
33         tagBITMAPINFOHEADER.__init__(self)
34         self.originalPixels = [] # 原始像素阵列
35         self.newImageWidth = 0 # 新图片宽度
36         self.originalWidth = 0 # 原始图片宽度
37         self.Parse(file_name) # 解析图片
38
39     @property
40     def width(self): # 获取图片宽度
41         return int.from_bytes(self.biWidth, 'little')
42
43     @property
44     def height(self): # 获取图片高度
45         return int.from_bytes(self.biHeight, 'little')
46
47     @property
48     def bit_count(self): # 获取图片位数
49         return int.from_bytes(self.biBitCount, 'little')
```

解析图片：读文件头、信息头

```
52  def Parse(self, file_name):
53      file = open(file_name, 'rb')
54
55      # 读取文件头
56      self.bfType = file.read(2)
57      self.bfSize = file.read(4)
58      self.bfReserved1 = file.read(2)
59      self.bfReserved2 = file.read(2)
60      self.bfOffBits = file.read(4)
61
62      # 读取信息头
63      self.biSize = file.read(4)
64      self.biWidth = file.read(4)
65      self.biHeight = file.read(4)
66      self.biPlanes = file.read(2)
67      self.biBitCount = file.read(2)
68      self.biCompression = file.read(4)
69      self.biSizeImage = file.read(4)
70      self.biXPelsPerMeter = file.read(4)
71      self.biYPelsPerMeter = file.read(4)
72      self.biClrUsed = file.read(4)
73      self.biClrImportant = file.read(4)
```

获取图片像素阵列信息：Width 利用公式计算。利用 numpy 先填充 0，再在 self.originalPixels 中存储每个像素的 r, g, b 分量。此时由于行对齐的原因，需要每次内循环跳过填充好的 0。

```
79  def GetPixel(self, file_name):
80      file = open(file_name, 'rb')
81      file.seek(54) # 跳过文件头和信息头(54个字节)
82
83      self.newImageWidth = int((((self.width * 8) + 31) & ~31) / 8) # 新图片宽度
84      self.originalWidth = int((((self.width * 24) + 31) & ~31) / 8) # 原始图片宽度
85      self.originalPixels = np.zeros((self.height, self.originalWidth), dtype=bytes) # 原始像
86      for i in range(self.height):
87          for j in range(self.width):
88              b = file.read(1)
89              g = file.read(1)
90              r = file.read(1)
91              self.originalPixels[i][j * 3] = b
92              self.originalPixels[i][j * 3 + 1] = g
93              self.originalPixels[i][j * 3 + 2] = r
94
95      # 读填充字节(每行字节数必须是4的倍数)
96      for k in range(self.width * 3, self.originalWidth, 1):
97          file.read(1)
```

生成调色板：此处展示的是灰度图的示例，由于是灰度图，调色板需要将红、绿、蓝分量都改为同样的数值；这样，对于 0~255 之间的数，调色板能返回不同明度的灰。同时，调色板的保留字段要设置为 0。

```
self.RGBQuad = np.zeros((256, 4), dtype=bytes)
for i in range(256):
    self.RGBQuad[i][0] = int.to_bytes(i, 1, 'little')
    self.RGBQuad[i][1] = int.to_bytes(i, 1, 'little')
    self.RGBQuad[i][2] = int.to_bytes(i, 1, 'little')
    self.RGBQuad[i][3] = 0
```


Convert 方法，用于实现具体要求的任务：

```
110 def Convert(self):
```

生成图片写入文件：

```
133 def generate(self, file_name):
```

2) 任务 1: 24To8

生成灰度图像，填充行：

灰度公式： $\text{Gray} = R * 0.299 + G * 0.587 + B * 0.114$

```
112 self.newPixels = np.zeros((self.height, self.newImageWidth), dtype=bytes) # 新像素阵列
113 for i in range(self.height):
114     for j in range(self.width):
115         # 灰度公式: Gray = R * 0.299 + G * 0.587 + B * 0.114
116         r = int.from_bytes(self.originalPixels[i][j * 3 + 2], 'little')
117         g = int.from_bytes(self.originalPixels[i][j * 3 + 1], 'little')
118         b = int.from_bytes(self.originalPixels[i][j * 3], 'little')
119         gray = int.to_bytes(int((r * 299 + g * 587 + b * 114) / 1000), 1, 'little')
120         self.newPixels[i][j] = gray
121
122     # 填充字节
123     for k in range(self.width, self.newImageWidth, 1):
124         self.newPixels[i][k] = 0
```

其他数据：

```
126 bsI = int(self.newImageWidth * int.from_bytes(self.biHeight, 'little')) # 图像大小
127 self.biSizeImage = int.to_bytes(bsI, 4, 'little') # 信息头中的图像大小
128 self.bfSize = int.to_bytes(bsI + 54 + 256 * 4, 4, 'little') # 文件头中的文件大小
129 self.bfOffBits = int.to_bytes(54 + 256 * 4, 4, 'little') # 文件头中的偏移量
130 self.biBitCount = int(8).to_bytes(2, 'little') # 信息头中的位数
```

写入具体信息，保存文件：

```
133 def generate(self, file_name):
134     file = open(file_name, 'wb+')
135     # 文件头
136     file.write(self.bfType)
137     file.write(self.bfSize)
138     file.write(self.bfReserved1)
139     file.write(self.bfReserved2)
140     file.write(self.bfOffBits)
141     # 信息头
142     file.write(self.biSize)
143     file.write(self.biWidth)
144     file.write(self.biHeight)
145     file.write(self.biPlanes)
146     file.write(self.biBitCount)
147     file.write(self.biCompression)
148     file.write(self.biSizeImage)
149     file.write(self.biXPelsPerMeter)
150     file.write(self.biYPelsPerMeter)
151     file.write(self.biClrUsed)
152     file.write(self.biClrImportant)
153     # 调色板
154     if int.from_bytes(self.biBitCount, 'little') == 8:
155         for i in range(256):
156             file.write(self.RGBQuad[i])
157     file.write(self.newPixels)
158     file.close()
```

3) 任务 2: 8To24

修改获取像素阵列：由于 8 位伪彩色的图片有颜色表，而 24 位真彩色没有颜色表字段（偏移量不同），因此这一任务要对此处代码进行修改，进而完成不同的操作。

```

79     def GetPixel(self, file_name):
80         file=open(file_name,'rb')
81         file.seek(1078) # 54 + 1024(调色板)
82
83         self.newImageScanWidth = int((((self.width * 24) + 31) & ~31) / 8) # 新图片每行字节数
84         self.originalWidth = int((((self.width * 8) + 31) & ~31) / 8) # 原始图片每行字节数
85         self.originalPixels = np.zeros((self.height, self.width), dtype=int) # 原始像素阵列
86         for i in range(self.height):
87             for j in range(self.width):
88                 self.originalPixels[i][j] = int.from_bytes(file.read(1), 'little')
89
90             # 原始图片每行字节数不是4的倍数时，需要补齐
91             for k in range(self.width, self.originalWidth, 1):
92                 file.read(1)
93
94         file.seek(54)
95         self.RGBQuad = np.zeros((256, 4), dtype=bytes) # 调色板
96         for i in range(256):
97             self.RGBQuad[i][0] = file.read(1)
98             self.RGBQuad[i][1] = file.read(1)
99             self.RGBQuad[i][2] = file.read(1)
100             self.RGBQuad[i][3] = file.read(1)
101
102         file.close()

```

注意，此时 originalPixels 中存储的是每个像素在颜色表中对应的索引值，转变为 24 位真彩色还需要对应调色板。

生成 24 位真彩色图像（方法形似上一个任务）：先设定图像大小、像素数据偏移量、每个像素所占用的字节数等，然后按照已存储的颜色表，进行真彩色信息的改写，补齐 0 以符合扫描行标准，注意不要超出像素范围。

```

106         bsI = int(self.newImageScanWidth * int.from_bytes(self.biHeight, 'little'))
107         self.biSizeImage = int.to_bytes(bsI, 4, 'little') # 位图大小
108         self.bfSize = int.to_bytes(bsI + 54, 4, 'little') # 文件大小
109         self.bfOffBits = int.to_bytes(54, 4, 'little') # 数据偏移量
110         self.biBitCount = int.to_bytes(24, 2, 'little') # 位数
111
112         self.newPixels = np.zeros((self.height, self.newImageScanWidth), dtype=bytes) # 新像素阵列
113         for i in range(self.height): # 逐行处理
114             for j in range(self.width): # 逐列处理
115                 b = self.RGBQuad[self.originalPixels[i][j]][0]
116                 g = self.RGBQuad[self.originalPixels[i][j]][1]
117                 r = self.RGBQuad[self.originalPixels[i][j]][2]
118                 self.newPixels[i][j * 3] = b
119                 self.newPixels[i][j * 3 + 1] = g
120                 self.newPixels[i][j * 3 + 2] = r
121
122             for k in range(self.width * 3, self.newImageScanWidth, 3): # 补齐每行字节数
123                 self.newPixels[i][k] = 0
124                 if k + 1 < self.newImageScanWidth:
125                     self.newPixels[i][k + 1] = 0
126                 if k + 2 < self.newImageScanWidth:
127                     self.newPixels[i][k + 2] = 0

```

写入具体信息，保存文件：

```
131     def generate(self, file_name):
132         file = open(file_name, 'wb+')
133         # 文件头
134         file.write(self.bfType)
135         file.write(self.bfSize)
136         file.write(self.bfReserved1)
137         file.write(self.bfReserved2)
138         file.write(self.bfOffBits)
139         # 信息头
140         file.write(self.biSize)
141         file.write(self.biWidth)
142         file.write(self.biHeight)
143         file.write(self.biPlanes)
144         file.write(self.biBitCount)
145         file.write(self.biCompression)
146         file.write(self.biSizeImage)
147         file.write(self.biXPelsPerMeter)
148         file.write(self.biYPelsPerMeter)
149         file.write(self.biClrUsed)
150         file.write(self.biClrImportant)
151         file.write(self.newPixels)
152         file.close()
```

3.其他

本次实验主要涉及的方面是图像处理，颇有挑战性（不能使用自带的图像库和方法），在实验的过程中我对位图文件的存储模式有了深入的了解，包括伪彩色图片的具体数据格式、真彩色图片的具体数据格式。

在具体完成的过程中，主要出现的问题是扫描行对齐及补 0 的问题，这会导致生成的图像有误，需要仔细地检查以及调试。

这些任务的完成不仅需要良好的编程能力，还需要对文件数据的处理方法有深刻的理解。通过本次实验，我们可以更深入地理解计算机对 BMP 图像数据的处理方式，以及编写代码的具体指示，有很大的帮助。