图像处理第一次作业

王兆盟 3017218072

2019 年 11 月 14 日

摘要

第一次作业的两道题的思路解析，由于本人对于python语言比较熟悉，因此两道作业题均用python语言完成。

1 第一题：绘制三条彩色曲线

这道题中我们需要用红、绿、蓝三种颜色的线分别绘制正弦、余弦以及$y = x^2$函数在$[0, \pi]$区间内的图像。我们不能直接采用现有的函数如pyplot.plot直接绘制，否则就失去了作业练习的意义。因为我采用双重循环一个个分别绘制单个的像素点的方式来画图。具体思路是：首先明确一张$m * n$的图片本质上可以看成是一个$m * n$的二维数组，其中每一个元素是四元组，用(R,G,B,A)来表示。元组中R、G、B分别代表了RGB颜色体系中的颜色值，取值范围为0-255，A为透明度值，取值范围为0-1。

我将这张图片的背景设置为乳白色，根据输入的图片长和宽，使用一个双重循环一列列将图片矩阵的每一个元素设置为一个乳白色的像素点，绘制一张乳白色的背景。

```
bg_pixTuple = (255, 251, 240, 1)
 for i in range(imgW):
        for j in range(imgH):
            img.putpixel((i, j), bg_pixTuple)
```

然后定义不同颜色的像素点绘制不同颜色的函数图像

```
    line_pixTuple_r = (255, 0, 0, 1)  #red
    line_pixTuple_g = (0, 255, 0, 1)  #green
    line_pixTuple_b = (0, 0, 255, 1)  #blue
```

函数图像绘制的思路是根据函数解析式，使得满足函数解析式的点成为不同颜色像素点。例如在图像矩阵中a_{ij}为红色像素点当且仅当i，j满足：

$$j = A\sin(\omega i + \varphi) + B$$

为了使函数图像占据整个画面，且由于图像大小不确定（由输入决定），因此需计算上述函数的具体参数值。根据

$$\omega = \frac{2\pi}{\lambda}$$

得到

$$\omega = \frac{2\pi}{imgW}$$

其中imgW为图片的长度。同时为了是曲线完整地显示在图片的适当位置，同时大小合适，因此我们需要调整A和B的值，我将其设置为

$$A = B = \frac{imgH}{6}$$

其中imgH为图片的高度。

```
w = (2 * math.pi) / imgW
A = imgH/6

for i in range(imgW-10):
    y = int(A * math.sin(w * i) + A)
    for j in range(10):
        for k in range(10):
            img.putpixel((i+j, y+k), line_pixTuple_r)
```

上述代码中的

```
    for j in range(10):
        for k in range(10):
            img.putpixel((i+j, y+k), line_pixTuple_r)
```

这双重循环的目的是：由于单个像素点实在太小，以单个像素点连成的曲线无法看清，因此我将以该像素点为原点周围10∗10范围的内一个正方形块区域作为一个“像素块”，链接这些“像素块”使之形成一条粗细为10像素的曲线。

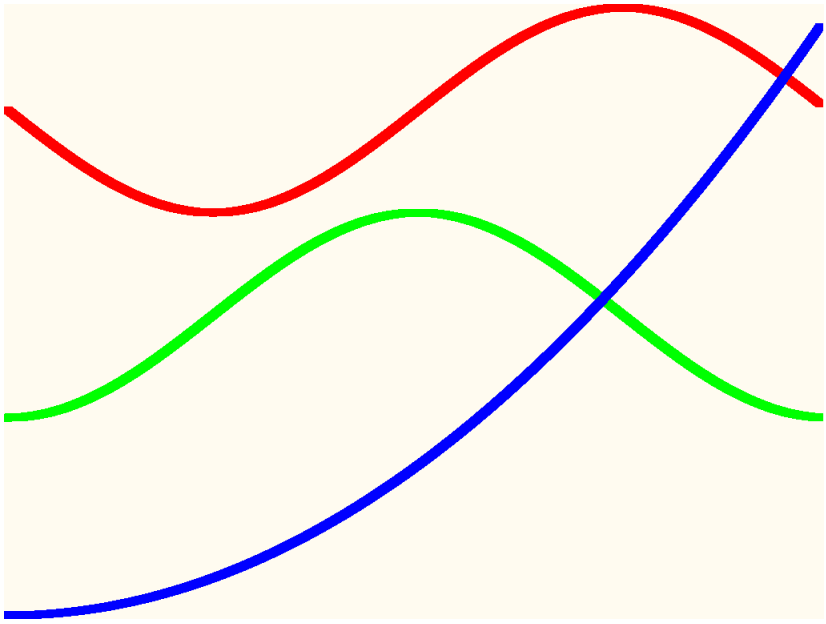
剩余两条曲线绘制代码类似：

```
#y = cos(x)
for i in range(imgW-10):
```

```
        y = int(A * math.cos(w * i) + 3*A)
        for j in range(10):
            for k in range(10):
                img.putpixel((i+j, y+k), line_pixTuple_g)

    #y = x^2
    a = int(imgH / math.pow(w * (imgW-10), 2))
    for i in range(imgW - 10):
        y = imgH - 10 - int(a * math.pow((w * i), 2))
        for j in range(10):
            for k in range(10):
                img.putpixel((i + j, y + k), line_pixTuple_b)
```

结果如图所示：



2 第二题：实现双线性内插法（Bilinear Interpolation）调整图片大小

这道题由于我水平有限能力一般且周五有考试复习时间紧张因此未能实现不使用for循环的版本，下周再改进。

假设源图像大小为$m*n$，目标图像为$a*b$。那么两幅图像的边长比分别为：$\frac{m}{a}$和$\frac{n}{b}$。通常这个比例不是整数，编程存储的时候要用浮点型。目标图像的第(i,j)个像素点（i行j列）可以通过边长比对应回源图像。其对应坐标为$(\frac{i*m}{a},\frac{j*n}{b})$。显然，这个对应坐标一般来说不是整数，而非整数的坐标是无法在图像这种离散数据上使用的。双线性插值通过寻找距离这个对应坐标最近的四个像素点，来计算该点的值（灰度值或者RGB值）。

如图，已知$Q_{12}, Q_{22}, Q_{11}, Q_{21}$但是要插值的点为P点，这就要用双线性插值了，首先在x轴方向上，对R_1和R_2两个点进行插值，这个很简单，然后根据R_1和R_2对P点进行插值，这就是所谓的双线性插值。

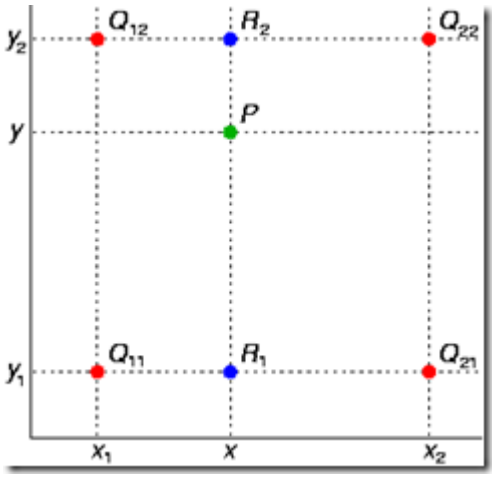


图 1: 双线性插值示意图

假如我们想得到未知函数 f 在点 P (x, y) 的值，假设我们已知函数 f 在 $Q_{11} = (x_1, y_1), Q_{12} = (x_1, y_2), Q_{21} = (x_2, y_1)$, 及 $Q_{22} = (x_2, y_2)$四个点的值。则未知点P(x, y) 的值用矩阵运算表示为：

$$f(x,y) \approx \begin{bmatrix} x & 1-x \end{bmatrix} \begin{bmatrix} f(x_1,y_1) & f(x_1,y_2) \\ f(x_1,y_2) & f(x_2,y_1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

然而由于坐标系的选择问题，或者说源图像和目标图像之间的对应问题。在计算对应坐标的时候改为以下公式

```
corr_x = (i+0.5)/th*pic.shape[0]-0.5
corr_y = (j+0.5)/tw*pic.shape[1]-0.5
```

利用上述公式，就能得到正确的双线性插值结果，代码实现如下：

```
def bi_linear(src, dst, target_size):
    pic = cv2.imread(src)
    th, tw = target_size[0], target_size[1]
    emptyImage = np.zeros(target_size, np.uint8)
    for k in range(3):
        for i in range(th):
            for j in range(tw):
                corr_x = (i+0.5)/th*pic.shape[0]-0.5
                corr_y = (j+0.5)/tw*pic.shape[1]-0.5
                point1 = (math.floor(corr_x), math.floor(corr_y))
                point2 = (point1[0], point1[1]+1)
                point3 = (point1[0]+1, point1[1])
                point4 = (point1[0]+1, point1[1]+1)
                fr1 = (point2[1]-corr_y)*pic[point1[0], point1[1], k] +
                                                    (corr_y-point1[1])
                                                   *pic[point2[0],
                                                   point2[1], k]
                fr2 = (point2[1]-corr_y)*pic[point3[0], point3[1], k] +
                                                    (corr_y-point1[1])
                                                   *pic[point4[0],
                                                   point4[1], k]
                emptyImage[i, j, k] = (point3[0]-corr_x)*fr1 + (corr_x-
                                                   point1[0])*fr2

    cv2.imwrite(dst, emptyImage)
    new_img = cv2.resize(pic, (200, 300))
    cv2.imwrite('pic/1_cv_img.png', new_img)
```

将第一题绘制的图像（原尺寸1024*768）缩放到300*200，结果如图所示：