Poisson Matting实验报告

计62 金镇书 2016080036

实验步骤

1. 初始化(F-B)，因为F和B未知，分别近似为在 ΩF 、ΩB 中最邻近的像素。并对(F-B)做高斯滤波
2. 计算α，求解$\Delta\alpha$ = div(∇I / (F-B))
3. F、B优化，把 Ω 中的F、B更新为到 Ω ∪ Ω+ 、 Ω ∪ Ω+中与之最近的像素颜色

实验过程

首先需要求出F-B和散度，可以用opencv中图像修补的发哪个是求出F和B

```
fGray = np.zeros((height, width), dtype=np.uint8)
bGray = np.zeros((height, width), dtype=np.uint8)
unknown = np.zeros((height, width), dtype=np.uint8)
for i in range(height):
    for j in range(width):
        if trimap[i,j] == 0:
            bGray[i,j] = 1
        elif trimap[i,j] == 255:
            fGray[i,j] = 1
        else:
            unknown[i,j] = 1
fImg = fGray * gray
bImg = bGray * gray
fInpaint = cv2.inpaint(fImg, (unknown + bGray), 3, cv2.INPAINT_TELEA)
bInpaint = cv2.inpaint(bImg, (unknown + fGray), 3, cv2.INPAINT_TELEA)

cv2.imshow('fInpaint', fInpaint)
cv2.imshow('bInpaint', bInpaint)
cv2.waitKey(0)

fInpaint = fInpaint * np.logical_not(bGray)
bInpaint = bInpaint * np.logical_not(fGray)
diff = scipy.ndimage.filters.gaussian_filter(fInpaint - bInpaint, 0.2

# 高斯滤波
```

同时需要求出div：

```
yGrad, xGrad = np.gradient(gray)
yyGrad, none = np.gradient(yGrad / diff)
none, xxGrad = np.gradient(xGrad / diff)
laplace = xxGrad + yyGrad
```

初定alpha = trimap * 0.5 + 前景（灰度）

然后通过高斯-赛德尔迭代法进行迭代计算：

其中：

拉普拉斯算子是一个二阶微分算子，定义为梯度∇f和散度$\nabla f'$。**对图像求二阶导数,因为图像是二维的，所以不用分开求横向和纵向的导数，然后相加。**

$$\mathrm{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

离散形式，在一个二维函数 $f(x, y)$ 中，x, y 两个方向的二阶差分分别为：

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$
$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

所以拉普拉斯算子的差分形式为：

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

高斯-赛德尔迭代法：

$$A = \left[-\frac{4}{\Delta x^2} \quad \frac{1}{\Delta x^2} \quad \frac{1}{\Delta x^2} \quad \frac{1}{\Delta x^2} \right]$$

$$\alpha = \begin{bmatrix} u_{ij} \\ u_{i+1,j} \\ u_{i-1,j} \\ u_{i,j+1} \\ u_{i,j-1} \end{bmatrix},$$ 由 laplacian 有，$A\alpha = g$，现求 α. 用 Gauss-Seidel 求.

我们只求 α 中的 u_{ij} 项.

记 $x_i^{(k+1)}$ 就是第k次迭代求得的 u_{ij}.

即 $$u_{ij}^{(k+1)} = -\frac{\Delta x^2}{4}\left(g_{ij} - \Delta x^2 u_{i-1,j}^{(k+1)} - \Delta x^2 u_{i,j-1}^{(k+1)} - \Delta x^2 u_{i,j+1}^{(k)} - \Delta x^2 u_{i+1,j}^{(k)}\right]$$

其中 我猜 $\Delta x^2 = 1$. 另外 $g_{ij} = b_{ij}$

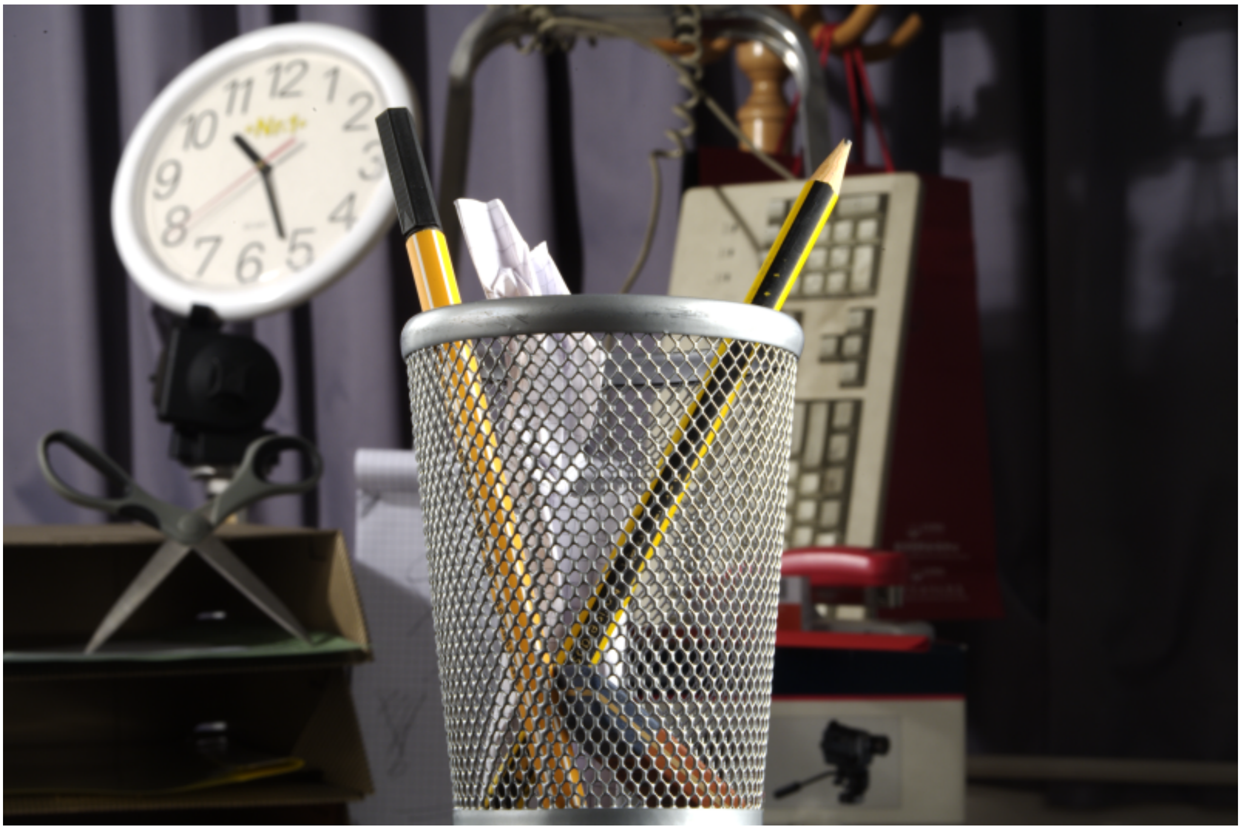
所以得到代码中的步骤.

```python
def getAlpha(laplace, alpha, unknown):
    height = unknown.shape[0]
    width = unknown.shape[1]
    alphaNew = alpha.copy()
    alphaOld = np.zeros((height, width), dtype=np.uint8)
    n = 1
    while (n < 100 and np.sum(np.abs(alphaNew - alphaOld)) > th):
        alphaOld = alphaNew.copy()
        # print(alphaOld)
        for i in range(1, height-1):
            for j in range(1, width-1):
                if(unknown[i,j]):
                    alphaNew[i,j] = 1/4 * (alphaNew[i-1 ,j] +
alphaNew[i,j-1] + alphaOld[i, j+1] + alphaOld[i+1,j] - laplace[i,j])
        n += 1
    alpha = alphaNew
    return alphaNew
```

实验结果

以笔筒为例：

原图



trimap

poisson