**Project4-Carving**

一、运行环境

操作系统：Windows11

Python版本：3.8

第三方库： numpy 1.24.2、opencv4.7.0

二、过程分析

①计算能量矩阵：以x方向梯度绝对值与y方向梯度绝对值之和作为像素点的能量值。

②计算值函数矩阵与路径矩阵：逐行迭代，更新值函数与路径矩阵。

值函数更新公式：

路径函数更新公式：

③删除seam：将最小seam右侧的像素点左移，并删除图像的最后一列像素点。每次只删除一条seam，并重新计算各矩阵以进行下一次的删除。

④插入seam：在图像最后增加一列像素点，并将最小seam右侧的像素点右移，空出来的位置像素值为最小seam像素值与左侧像素值的平均。如果每次仅插入一条seam，这样下次寻找到的最小seam可能还是在前一次最小seam的附近。为了减少信息冗余，先一次性找到k条最小seam，在进行插入操作。

（此外，对于水平方向上seam的寻找、删除、插入等操作，可以通过将图像翻转，然后转化为竖直方向上的相关操作）

⑤目标擦除与保护：通过将目标所在区域的能量矩阵值设置为较小的负值以实现目标擦除，同理通过设置较大的正值以实现目标保护。为了在多次的删除循环中锁定目标，可以设置一个mask矩阵，mask矩阵中目标所在区域的值为负或正（取决于对目标的操作类型），其余区域均为0，将mask矩阵与计算出的能量矩阵相加以得到最终的能量矩阵，然后对原始图像进行seam的删除时，也对mask矩阵进行同样的操作，从而保证了在删除的多次循环中目标的一致性。

三、运行结果

①原始图像



兴庆校区北门腾飞塔

②缩小图像



水平方向删除100行



竖直方向删除150列



水平方向删除100行，竖直方向删除150列

总结：水平方向删除的效果较好，但由于图像中的道路边缘线沿着竖直斜向倾斜，因而在竖直方向进行删除时道路边缘线发生了弯折。

③放大图像



水平方向插入50行



竖直方向插入30列



水平方向插入50行，竖直方向插入30列

总结：水平方向插入的效果较好，但在竖直方向进行插入时，图像中的部分区域出现了明显的“残影”。

④目标删除



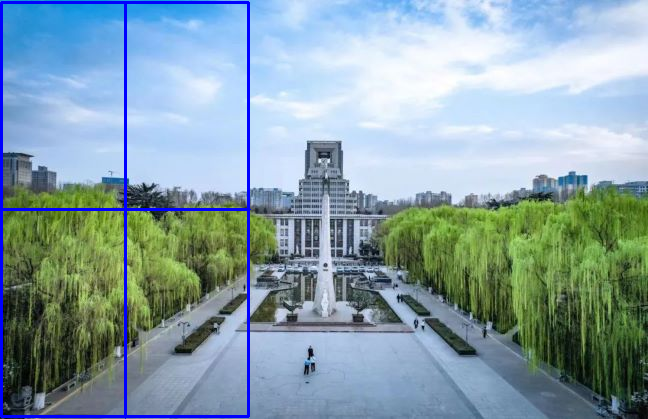
图中蓝色方框为待删除目标



目标删除后的图像

总结：目标删除后，图像的其余区域仍得到了较好的保留。

⑤目标保护



图中蓝色方框为想保护的目标



目标保护后的图像

总结：在之前的竖直方向删除中，图像左右两侧的区域都被删除了一部分，但当选择将左侧的目标区域进行保护后，进行同样的删除操作只删除了其余区域，而目标区域仍保持完整。

四、源代码

|  |
| --- |
| **import** cv2 **import** numpy **as** np   *# 求解能量矩阵* **def** genEngMap(Img):  **if** len(Img.shape) == 3:  *# BGR彩色图像转灰度图像* Ig = np.float64(cv2.cvtColor(Img, cv2.COLOR\_BGR2GRAY))  **else**:  Ig = np.float64(Img)   *# 以梯度的L1范数作为能量值* gx, gy = np.gradient(Ig)  **return** np.abs(gx) + np.abs(gy)   *# 求解竖直方向上的值函数矩阵与路径矩阵* **def** cumMinEngVer(E):  row, col = E.shape  Mx = np.zeros((row, col)) *# 值函数矩阵* Tbx = np.zeros((row, col)) *# 路径矩阵* Mx[0, :] = E[0, :]   **for** i **in** range(1, row):  **for** j **in** range(col):  **if** j == 0:  Mx[i, j] = E[i, j] + min([Mx[i-1, j], Mx[i-1, j+1]])  Tbx[i, j] = 0 **if** Mx[i-1, j] < Mx[i-1, j+1] **else** 1  **elif** j == col-1:  Mx[i, j] = E[i, j] + min([Mx[i-1, j], Mx[i-1, j-1]])  Tbx[i, j] = 0 **if** Mx[i-1, j] < Mx[i-1, j-1] **else** -1  **else**:  Mx[i, j] = E[i, j] + min([Mx[i-1, j-1], Mx[i-1, j], Mx[i-1, j+1]])  Tbx[i, j] = -1 **if** Mx[i-1, j-1] < min([Mx[i-1, j], Mx[i-1, j+1]]) **else** 0 **if** Mx[i-1, j] < Mx[i-1, j+1] **else** 1   **return** Mx, Tbx   *# 求解水平方向上的值函数矩阵与路径矩阵* **def** cumMinEngHor(E):  My, Tby = cumMinEngVer(np.transpose(E))  **return** np.transpose(My), np.transpose(Tby)   *# 删除竖直seam* **def** rmVerSeam(Img, Mx, Tbx):  row, col = Mx.shape  Mx\_min = np.finfo(np.float64).max  rm\_x = 0  *# 计算seam起点* **for** i **in** range(col):  **if** Mx[row - 1, i] < Mx\_min:  Mx\_min = Mx[row - 1, i]  rm\_x = i   img\_array = np.array(Img)  *# 将竖直seam右侧的像素点左移* **for** i **in** range(row - 1, -1, -1):  **for** j **in** range(rm\_x, col - 1):  img\_array[i, j] = img\_array[i, j + 1]  rm\_x += int(Tbx[i, rm\_x])   *# 删除最后一列像素点* **return** np.delete(img\_array, -1, axis=1)   *# 删除水平seam* **def** rmHorSeam(Img, My, Tby):  My = np.transpose(My)  Tby = np.transpose(Tby)  Img = cv2.transpose(Img)   **return** cv2.transpose(rmVerSeam(Img, My, Tby))   *# 增加k个竖直seam* **def** addVerSeam(Img, Mx, Tbx, k=1):  row, col = Mx.shape  add\_x = [0] \* k  *# 计算k个最小seam起点* **for** m **in** range(k):  Mx\_min = np.finfo(np.float64).max  **for** i **in** range(col):  **if** Mx[row - 1, i] < Mx\_min:  Mx\_min = Mx[row - 1, i]  add\_x[m] = i  Mx[row - 1, int(add\_x[m])] = np.finfo(np.float64).max  *# 避免seam过于集中* **for** i **in** range(5):  **if** int(add\_x[m]) - i - 1 > 0:  Mx[row - 1, int(add\_x[m]) - i - 1] = np.finfo(np.float64).max  **if** int(add\_x[m]) + i + 1 < col:  Mx[row - 1, int(add\_x[m]) + i + 1] = np.finfo(np.float64).max  add\_x.sort(reverse=**True**)   img\_array = np.array(Img)  **for** m **in** range(k):  *# 插入最后一列像素点* img\_array = np.insert(img\_array, -1, img\_array[:, -1], axis=1)  **for** i **in** range(row - 1, -1, -1):  *# 将竖直seam右侧的像素点右移* **for** j **in** range(col + m, int(add\_x[m]), -1):  img\_array[i, j] = img\_array[i, j - 1]  *# 将seam位置的像素值与左侧像素值取平均* t = int(add\_x[m])  **if** t > 0:  img\_array[i, t] = np.mean([img\_array[i, t-1], img\_array[i, t]], axis=0)  *# 更新seam位置* add\_x[m] += Tbx[i, int(add\_x[m])]   **return** img\_array   *# 增加k个水平seam* **def** addHorSeam(Img, My, Tby, k=1):  My = np.transpose(My)  Tby = np.transpose(Tby)  Img = cv2.transpose(Img)   **return** cv2.transpose(addVerSeam(Img, My, Tby, k))   *# 将图片缩小或者放大* **def** carving(Img, nr, nc, opType=**'rm'**):  **if** opType == **'rm'**:  *# 缩小图片* **for** i **in** range(nr):  e = genEngMap(Img)  A, B = cumMinEngHor(e)  Img = rmHorSeam(Img, A, B)   **for** j **in** range(nc):  e = genEngMap(Img)  A, B = cumMinEngVer(e)  Img = rmVerSeam(Img, A, B)   **elif** opType == **'add'**:  *# 放大图片* e = genEngMap(Img)  A, B = cumMinEngHor(e)  Img = addHorSeam(Img, A, B, nr)   e = genEngMap(Img)  A, B = cumMinEngVer(e)  Img = addVerSeam(Img, A, B, nc)   **elif** opType == **'rm\_obj'**:  *# 目标擦除* rect = (0, 0, 0, 0)  **while** rect == (0, 0, 0, 0):  rect = cv2.selectROI(**''**, Img)  cv2.destroyAllWindows()   H = Img.shape[0]  W = Img.shape[1]  mask = np.zeros((H, W))  **for** i **in** range(rect[0], rect[0] + rect[2]):  **for** j **in** range(rect[1], rect[1] + rect[3]):  mask[j, i] = -100000   **while not** np.all(mask == 0):  e = genEngMap(Img) + mask  **if** rect[1] > rect[3]:  A, B = cumMinEngVer(e)  Img = rmVerSeam(Img, A, B)  mask = rmVerSeam(mask, A, B)  **else**:  A, B = cumMinEngHor(e)  Img = rmHorSeam(Img, A, B)  mask = rmHorSeam(mask, A, B)   **elif** opType == **'rm\_save'**:  *# 目标保留* rect = (0, 0, 0, 0)  **while** rect == (0, 0, 0, 0):  rect = cv2.selectROI(**''**, Img)  cv2.destroyAllWindows()   H = Img.shape[0]  W = Img.shape[1]  mask = np.zeros((H, W))  **for** i **in** range(rect[0], rect[0] + rect[2]):  **for** j **in** range(rect[1], rect[1] + rect[3]):  mask[j, i] = 100000   **for** i **in** range(nr):  e = genEngMap(Img) + mask  A, B = cumMinEngHor(e)  Img = rmHorSeam(Img, A, B)  mask = rmHorSeam(mask, A, B)   **for** j **in** range(nc):  e = genEngMap(Img) + mask  A, B = cumMinEngVer(e)  Img = rmVerSeam(Img, A, B)  mask = rmVerSeam(mask, A, B)   **return** Img   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  img = cv2.imread(**'img0.jpg'**)  new\_img = carving(img, 100, 0, opType=**'rm'**)  *# new\_img = carving(img, 0, 150, opType='rm')  # new\_img = carving(img, 100, 150, opType='rm')  # new\_img = carving(img, 50, 0, opType='add')  # new\_img = carving(img, 0, 30, opType='add')  # new\_img = carving(img, 50, 30, opType='add')  # new\_img = carving(img, 0, 0, opType='rm\_obj')  # new\_img = carving(img, 0, 150, opType='rm\_save')* cv2.imshow(**'original'**, img)  cv2.imshow(**'carving'**, new\_img)  cv2.waitKey(0) |