数字图像处理

大作业二

图像去遮挡

学号　　2017011589

姓名　　吾尔开西

班级　　自 76

目录

一、总述

本次大作业要求将图片前景的黑色栏杆去除，恢复遮挡部分。要求修改的图片有两张，第一张图片的前景栏杆较窄，第二张图片的前景栏杆大部分较窄，有一根栏杆较粗。

（原图片一和图片二）

为了去除遮挡部分，首先需要找到遮挡部分的位置，即 mask，这一步使用一系列图像处理的方法来完成。

针对第一张图片，由于遮挡物较窄，我使用了 Fast Marching Method 算法来实现去遮挡，该算法运算较快，对窄遮挡物去除效果较好，但对宽遮挡物（宽度大于 15 像素）进行修复时会出现模糊现象(Telea, 2004)。

因此，针对第二张图片的宽栏杆部分，我使用了 criminisi 算法来进行修复，该算法运算较慢，但对宽遮挡物修复效果极佳(Criminisi, Perez, & Toyama, 2003)。

此外，我对 criminisi 算法做了一点改动，大大提高了它的运算速度，同时保证修复质量。

为了便于可执行文件的使用，我制作简单的 UI 界面。

去遮挡算法难以用矩阵运算实现，只能使用 for 循环，因此运算速度整体较慢。为了减少运算时间，我缩小了图片尺寸。

上述两篇论文中的计算思路清晰，甚至有伪代码，易于复现。本次作业的代码均由本人自己编写，但代码思路参考了论文以及如下网站代码。测试图片部分来自下面的网站。

https://github.com/cantarinigiorgio/Image-Inpainting

https://github.com/eodos/inpainting-telea-matlab

二、提取遮挡物

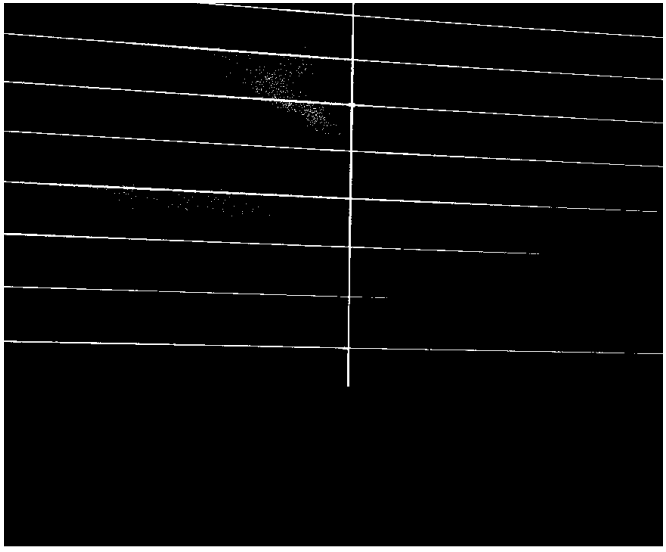
为了去遮挡，首先需要得到遮挡物的位置，即 mask。为此我进行了一系列图像处理的操作。这一部分的算法可拓展性较差，只适用于这次作业的两张图片，且参数之间的依赖性较强。

两张图片中的栏杆都是以直线的形式呈现，所以这一部分算法的基本思路是利用直线检测提取位置。

1、图片一中栏杆位置的提取

对应代码：get_mask/get_mask_1.m

首先将图片二值化并去除图片下方的黑色部分，避免对直线检测结果造成干扰。



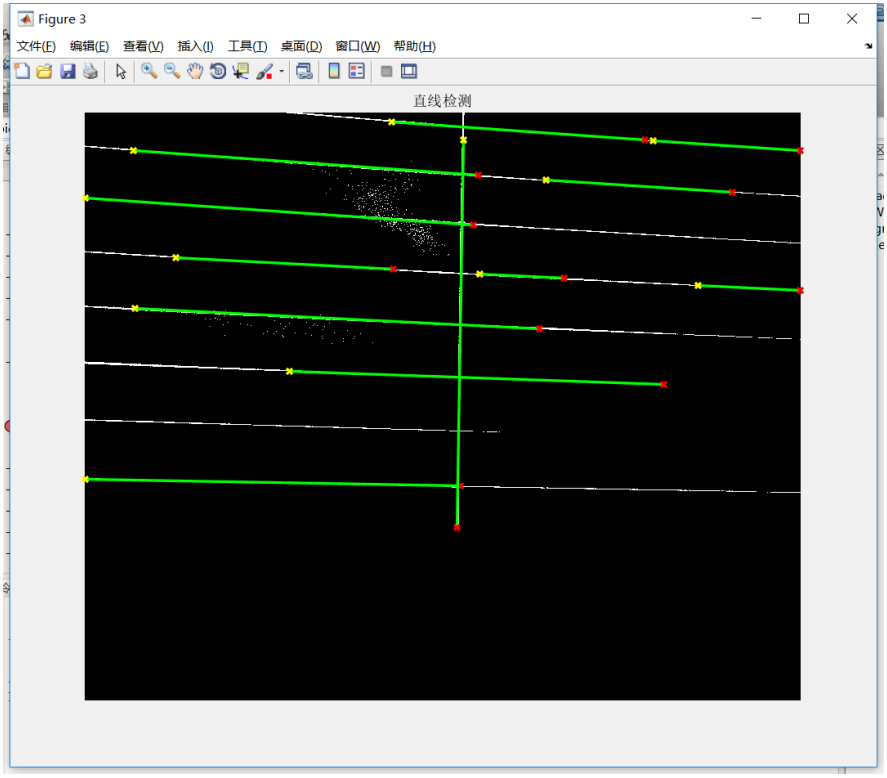
（图片一二值化）

接下来，对二值化后的图片使用 Hough 变换，Hough 变换是把图像空域中的点转化到$\rho - \theta$空间，使用如下公式:

$$\rho = x\cos\theta + y\sin\theta$$

其中ρ和θ表示极坐标中直线到原点的距离以及距离垂线的角度。因此，图像中的一个点对应 Hough 空间中的一条正弦曲线，而图像中的一条直线对应 Hough 空间中多条正弦曲线的交点。

使用 Hough 变换检测直线的另一个好处是可以知道直线的起止点，这对本例是十分有用的。Hough 变换检测到的直线如下图所示。

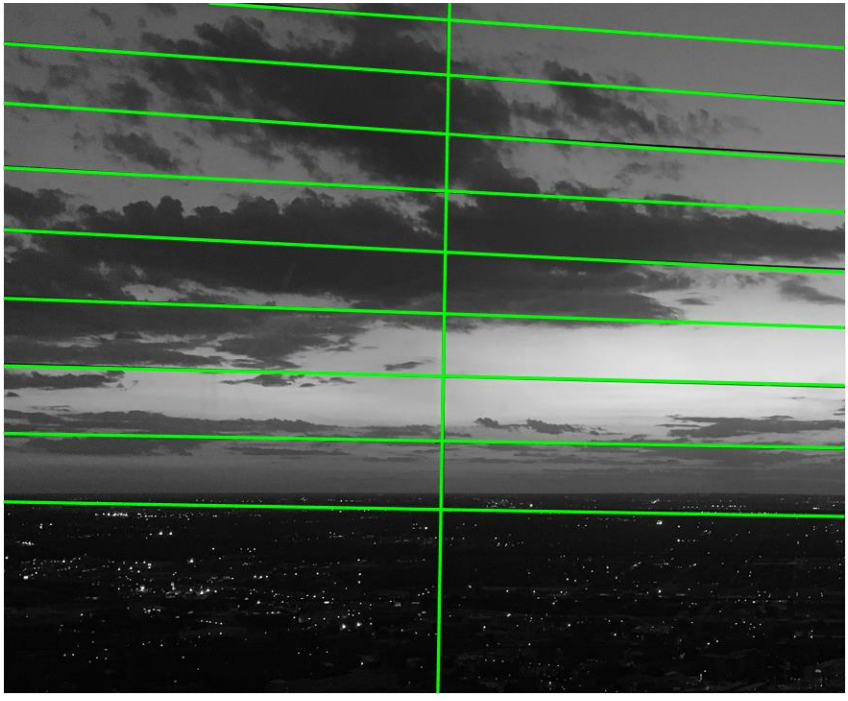


（Hough 变换检测直线）

Hough 变换得到的曲线中，有一些冗余的检测结果，也有一条直线没有检测到，因此我们需要做一些进一步的处理。

在 get_mask/lines_integrate.m 函数中，我首先将$\rho-\theta$直线转换成$k-b$斜率截距型，方便后面的计算。再将重复的直线去掉，判断标准是两直线角度差以及距离。

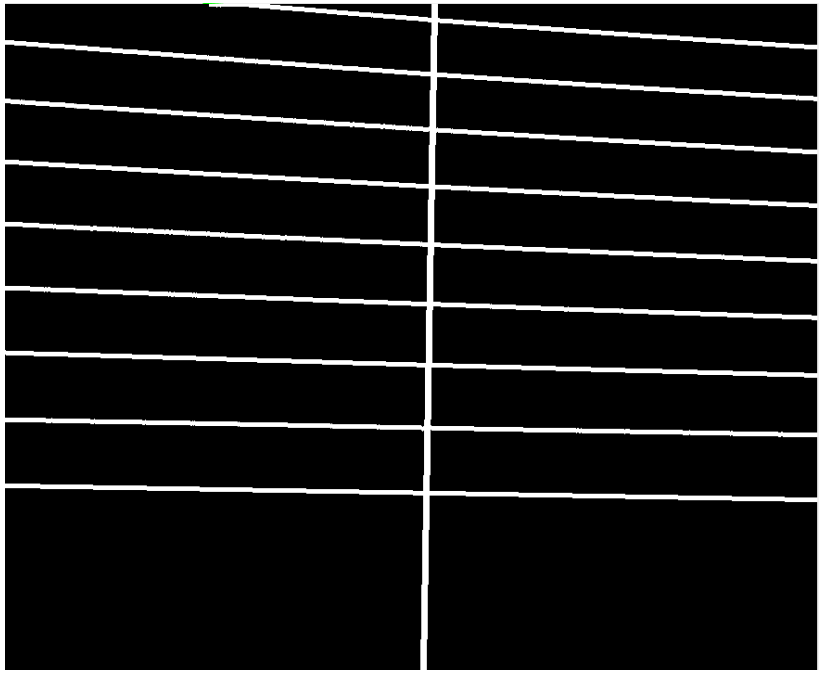
处理之后，将直线按从上到下的顺序排列，并利用上下两条直线间的距离找到遗漏直线的位置，补上遗漏的直线。下面是补上遗漏直线后的处理结果：



（去掉重复直线并补上遗漏直线）

找到所有直线的位置后，下面就需要用这些直线填补出栏杆的 mask，这个 mask 是和原图一样大小的矩阵，在有栏杆的地方矩阵值为 1（或 255），在没有栏杆的地方矩阵值为 0.

get_mask/maskFromlines.m 函数是这一算法的实现，基本思路是遍历直线上每一列的像素点（竖线是遍历每一行），寻找这一点附近灰度值的局部最小值点，将这一最小值点附近一部分点（具体多少点由超参决定，其实就是栏杆的宽度）的 mask 设为 1。下面是得到的图片一中栏杆的 mask。



（图片一中栏杆的 mask）

2、图片二中栏杆位置的提取

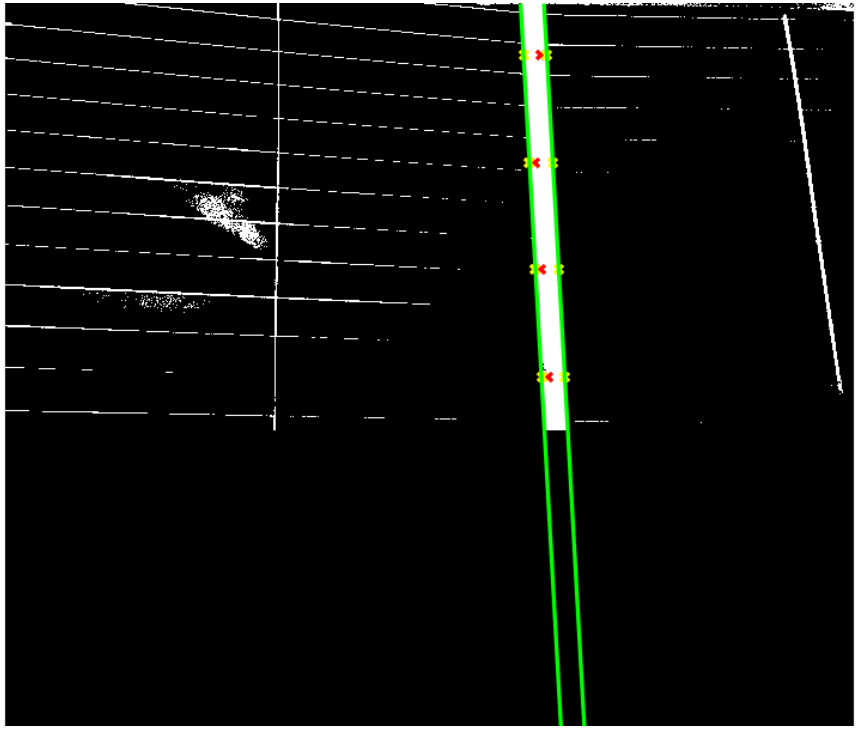
对应代码：`get_mask_2.m`

图片二的处理比图片一的处理更复杂，但大体思路相同。

图片二中间有一条栏杆宽度很大，需要先将这根栏杆的位置找到并除去，方便后面直线检测的顺利进行。

首先依然是将图片二值化，去除下方的黑色干扰部分。

寻找宽栏杆时要利用宽栏杆中点附近黑点更多的特点，找到宽栏杆中若干点。具体操作时，我在图片中间隔相等的四行进行扫描，对每一个点，计算它附近一定大小的领域中黑点的数量，每一行中领域黑点数量最多的那个点应该就在宽栏杆内部。找到的宽栏杆内部四个点纵向均匀分布，从这个四个点分别向左向右出发，找到宽栏杆共八个边界点。找到的四个内部点（红点）和八个边界点（黄点）如下图所示。



（宽栏杆边界点、内部点和边界线）

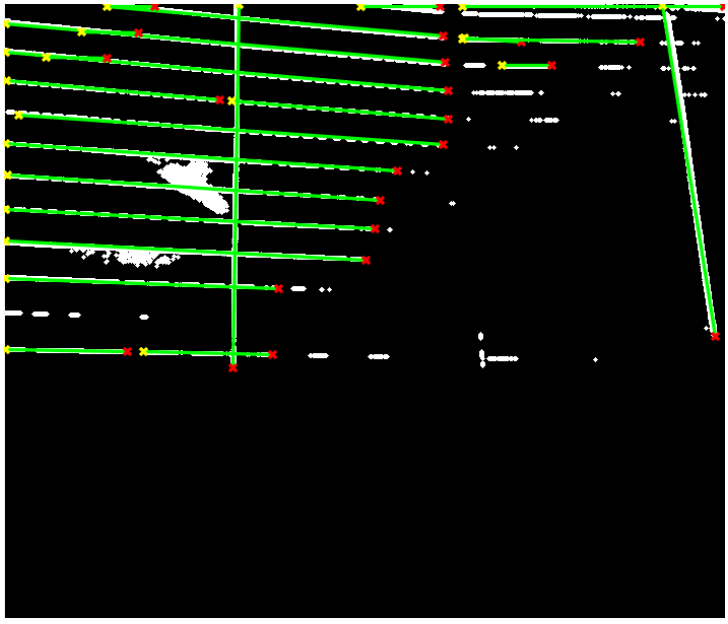
宽栏杆八个边界点中左右各有四个边界点，利用这些边界点可以拟合出宽栏杆的边界线，如上图绿线所示。

得到宽栏杆左右边界线后，就可以得到其左边区域和右边区域，并去除宽栏杆。

图片二中的直线有很多断裂部分，如果直接使用直线检测效果不佳，因此我先对其进行膨胀处理。如下图，直线的连接度高了许多。

（图片二的二值化、去除宽栏杆、膨胀处理结果）

之后就能使用图片一中类似的步骤来检测直线了，但是，由于图片二中宽栏杆左右部分直线的斜率不同，位置不同，所以需要分别进行处理。直线检测结果如下图所示。



（图片二中扁栏杆位置检测）

（图片二中的扁栏杆 mask 在原灰度图上的位置）

（图片二中的宽栏杆 mask 在原灰度图上的位置）

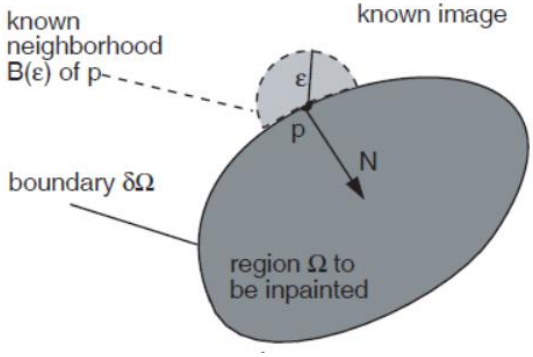
三、FFM 算法修复图片一

代码：FFM_inpaint 文件夹中 inpaint_FMM.m, nearest_interp.m, myMinheap.m, solve_T.m, compute_outside.m

Fast Marching Method 算法其实是一种插值方法，其特殊之处在于插值权重的确定以及最佳修复点的快速查找。

考虑下面图中的情况：区域Ω是图片需要修复的区域，$\delta\Omega$是Ω的边缘. 现在需要修复 p 点的像素值，我们用已知像素区域中 p 的一小片圆形领域$B(\varepsilon)$中的像素做插值：

$$I(p)=\frac{\sum_{q\in B(\varepsilon)} w(p,q)I(q)}{\sum_{q\in B(\varepsilon)} w(p,q)}$$



下面我们来介绍怎样计算权重$w(p,q)$，怎样快速找到最佳修复点以及算法流程。

算法实现中用到了$F(i,j)$和$T(i,j)$两个关键的矩阵，$T(i,j)$表示点(i,j)到待修补区域边缘的距离。$F(i,j)$表示点(i,j)的性质，待修补区域中的点值为 2（内部点，未知点），待修补区域边界点值为 1，已知像素区域的点值为 0。这两个矩阵都会在代码运行过程中动态更新。

1、权重的计算

权重的计算综合考虑了各个方面，具体地，$w(p,q)=|dir(p,q)*dst(p,q)*lev(p,q)|$。p 是需要修复的点，q 是其已知像素领域$B(\varepsilon)$中的一个点。

其中，$dir(p,q)$保证沿着 N(p)向量方向的点有更大的权重，$N(p)=\nabla T(p_i,p_j)$，即 T 矩阵在 p 处的梯度，实际表示$\delta\Omega$在 p 点处指向Ω内部的垂线。p-q 表示 q 指向 p 的向量。

$dst(p,q)$保证距离 p 近的点有更大的权重，$\left|\left|p-q\right|\right|^2$表示 p 到 q 距离的平方，$d_0$一般取 1。

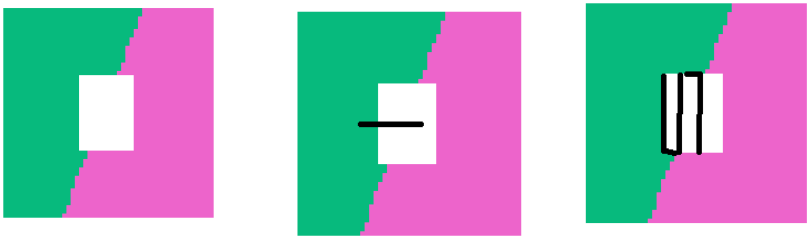
$lev(p,q)$保证距离待修复区域边缘（$\delta\Omega$）近的点有更大的权重

$$
\begin{aligned}
dir(p,q) &= \frac{p-q}{||p-q||}\cdot N(p) \\
dst(p,q) &= \frac{d_0^2}{||p-q||^2} \\
lev(p,q) &= \frac{T_0}{1+|T(p)-T(q)|}.
\end{aligned}
$$

2、快速查找最佳修复点

FFM 算法很重要的一个特点是快速查找最佳修复点，因为在修复图片的过程中，我们可以随意挑选一个边缘点，然后对它旁边的未知点进

行修复，但这样的修复效果很差。例如修复下方左图中的中间白色区域，按中间图黑线所示的顺序进行修复效果差，因为内层的点原本是未知区域，其像素值是计算出来的，可信度差。但如果按右边图黑线所示顺序修复效果相对较好。



上述两种修复方式的区别在于，边缘点的选择方式，如果我们优先选择 T 矩阵值更小的边缘点，则能保证选择的边缘点可信度更高。

为了快速找到 T 矩阵值最小的边缘点，我们可以维护一个最小堆，提高运算速度。

T 矩阵的计算其实是一个方程的解，我们下面会提到。

3、算法流程

算法的输入是一张 rgb 图片、mask 矩阵（需要修复的区域值大于 0），以及修复半径，即圆形领域$B(\varepsilon)$的半径。

（1）首先求出初始 F 矩阵，具体做法是将 mask 矩阵用$\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$进行膨胀得到 F’，将 F’大于 0 的部分置为 1, F_temp=F’-F 表示边缘点，则 F=2F’-F_temp，这样得到的 F 满足：待修补区域中的点值为 2（内部点，未知点），待修补区域边界点值为 1，已知像素区域的点值为 0。

（2）然后初始化 T 矩阵，具体做法是先将内部点的 T 值置为很大的一个值（1e6），边缘点值为 0，然后计算部分外部点的 T 值。只计算部分外部点的 T 值的原因是修复过程中只用到待修复区域周围的一部分外部点。计算外部点的 T 值其实是将 F 中的 0 和 2 替换（即内部点和外部点替换），再用我们下面要介绍的流程进行。

（3）T 矩阵与 F 矩阵初始化成功后，我们先找到边缘点坐标以及它们的 F 矩阵值，放进最小堆 heap 里，就可以开始修复循环了。

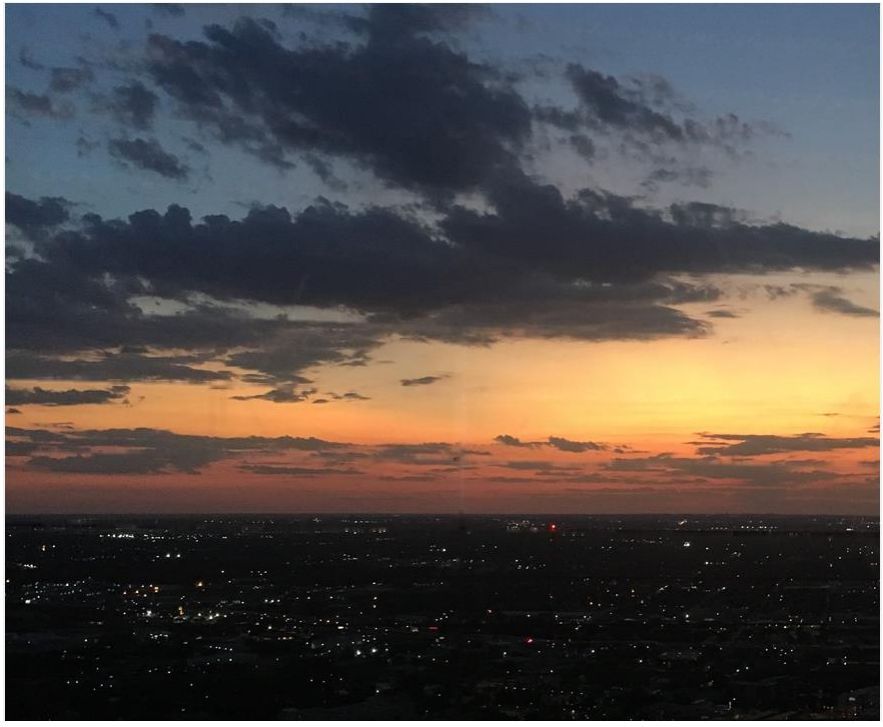
下面是修复循环在论文中的伪代码，其中，NarrowBand 指待修复区域边缘点，简单来说就是每一轮从最小堆 heap 里提取出 T 值最小的边缘点 p，找到 p 上下左右的点中待修复的点，插值计算这个点的像素值，T 矩阵值，在 F 矩阵中将它设为边缘点，并把它放入最小堆 heap 中，循环直到没有边缘点为止。

```
while (NarrowBand not empty)
{
  extract P(i,j) = head(NarrowBand);      /* STEP 1 */
  f(i,j) = KNOWN;
  for (k,l) in (i1,j),(i,j1),(i+1,j),(i,j+1)
  if (f(k,l)!=KNOWN)
  {
    if (f(k,l)==INSIDE)
    {
      f(k,l)=BAND;                        /* STEP 2 */
      inpaint(k,l);                       /* STEP 3 */
    }
    T (k,l) = min(solve(k1,l,k,l1),       /* STEP 4 */
                  solve(k+1,l,k,l1),
                  solve(k1,l,k,l+1),
                  solve(k+1,l,k,l+1));
    insert(k,l) in NarrowBand;            /* STEP 5 */
  }
}
```

（FFM 算法循环部分的伪代码）

4、修复结果

下面是图片一的修复结果，可以看出结果基本看不到栏杆的痕迹，且图片比较自然，看不出修复痕迹，而且图片其他部分不会受影响。



（用 FFM 算法修复图片一）

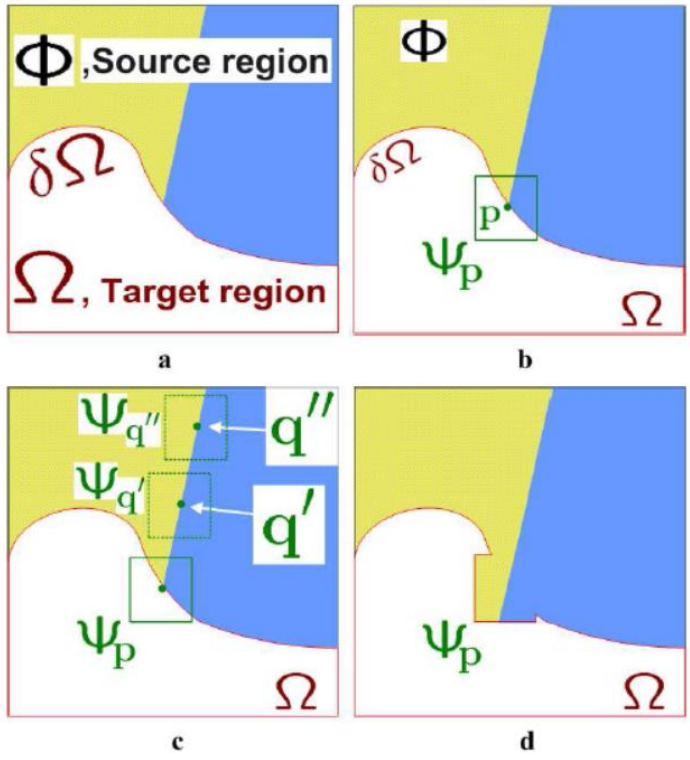
下面是使用FFM算法修复图片二的结果，可以看到中间的宽栏杆有一定程度的模糊，其他部分效果较好。

（用FFM算法修复图片二）

四、criminisi 算法修复图片二

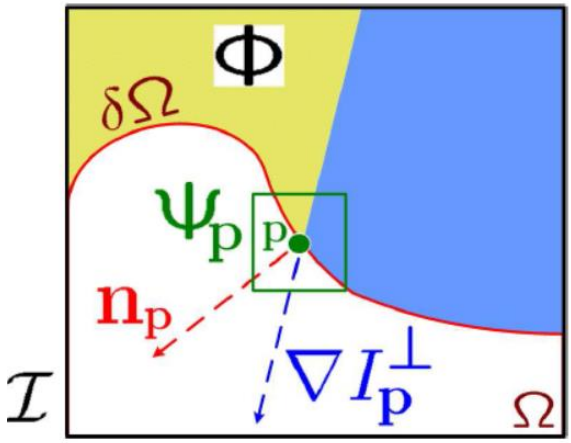
criminisi 算法的本质是查找已知像素区域中的小方块补到未知部分，如下图所示，Ω是待修复区域，ϕ是已知像素区域。修补 p 附件小块Ψ_p

时，找到与之匹配度最高的小块$\Psi_{q'}$，并将它补到Ψ_p的位置。算法的关键是找到最高优先级的 p，再找到与之领域正方形小块Ψ_p匹配度最高的$\Psi_{q'}$。



1、修复优先级

我们在三.2 中已经阐述了修复顺序的重要性，criminisi 算法中也是如此，考虑下图中的情况



定义边缘点 p 的优先级 P(p)，

$$P(p) = C(p) * D(p)$$

$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\Psi_{\mathbf{p}}\cap(\mathcal{I}-\Omega)} C(\mathbf{q})}{|\Psi_{\mathbf{p}}|}, \qquad D(\mathbf{p}) = \frac{|\nabla I_{\mathbf{p}}^{\perp}\cdot \mathbf{n}_{\mathbf{p}}|}{\alpha}$$

C(p)表示 p 点附近已知像素区域部分的比重，D(p)表示 p 点图片像素值梯度的垂线$\nabla I_p^{\perp}$与待修复区域边缘线单位垂线n_p的内积大小。α一般取 255。

计算出所有边缘点的优先级后，挑出优先级最高的边缘点进行修复。

2、寻找最佳匹配

挑出优先级最高的边缘点 p 后，要对 p 的正方形领域Ψ_p进行修复，需要在全图范围内找到与之匹配度最高的小块$\Psi_{q'}$，具体方法是求出图片中每个正方形块Ψ_q与Ψ_p的已知部分的像素值差的平方和，二者相差最小的块即为$\Psi_{q'}$。

注意这里的像素值差是指在 LAP 空间中的像素值差，因为 LAP 空间更能体现图片的纹理结构。因此，需要在初始化时将 rgb 图片转换得到 LAP 图片。

3、算法流程

算法的输入是一张 rgb 图片、mask 矩阵（需要修复的区域值大于 0），以及修复块半径，即正方形修补块的边长。

算法的具体流程也很清楚：

（1）先求出图片像素值梯度的垂线$\nabla I_p^{\perp}$（这里的像素值指在 rgb 空间中的值），做一些初始化。

（2）进入算法循环，先根据当前的 mask 求出边缘点，再循环求出每个边缘点的 C 值，求出所有边缘点的 D 值，用 C 和 D 数组求出优先级 P，挑出优先级最高的边缘点 p。找到边缘点 p 领域Ψ_p的最佳匹配$\Psi_{q'}$，复制

$\Psi_{q'}$的 rgb 值、lap 值；更新 C（使Ψ_p中未知像素点的 C 值都等于 p 点 C 值）；更新 mask（Ψ_p中点的 mask 都设为 0，即已知点）。

（3）循环（2）中流程，直到没有未知点为止。

4、结果

下图是用 criminisi 算法修复图片二的结果，为了减少运算时间，图片缩的比较小，可以看出宽栏杆部分的修复效果比用 FFM 算法修复的效果好一些，没有出现模糊，但还是有待改进。

（用 criminisi 算法修复图片二）

具体运行算法过程中，其实是先用 FFM 算法进行修复，再用 criminisi 算修复宽栏杆部分。

五、criminisi 算法优化

Criminisi 算法每一轮循环都只修复一个正方形块大小的区域，但 C 值和 D 值需要在所有边缘点上进行计算，于是我尝试了每一轮循环中局部更新 C 值和 D 值，但发现这样做遗漏了一些特殊情况，运算结果会变差，甚至进入死循环。

我转而从最佳匹配部分下手，原算法每次寻找边缘边缘点 p 领域Ψ_p的最佳匹配$\Psi_{q'}$时都要遍历整张图片，十分耗时间，但实际上最佳匹配块一般离待修复区域不远，所以我将查找范围限定在 p 点附近一定区域，结果运算速度大大提升，修复质量又没有太大影响。

以下面的图片为例，需要将图片中第二个骑行者去除。

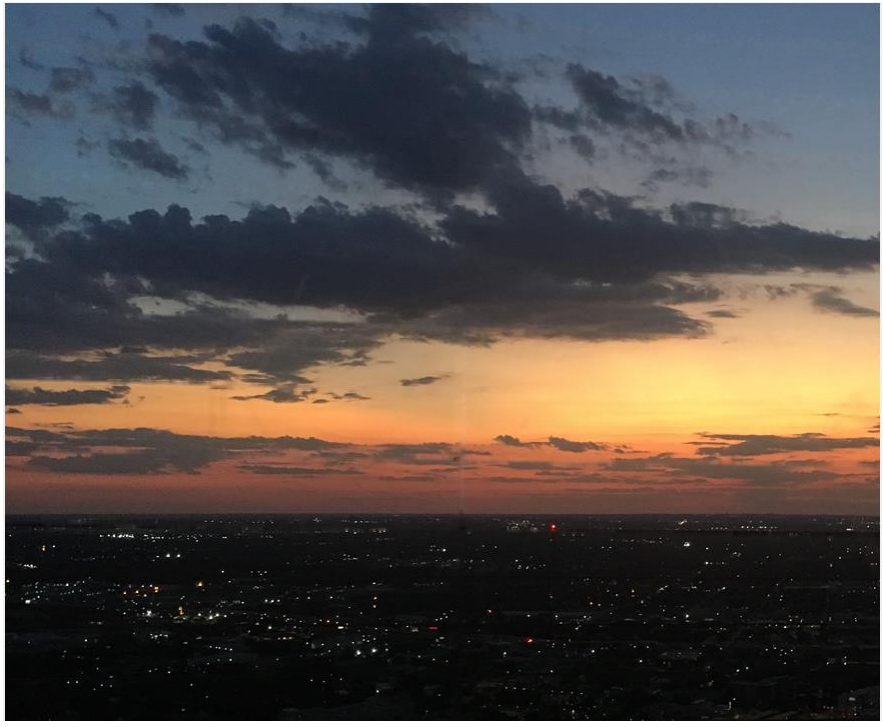
使用原算法，用时 40.74 秒:

快速算法，用时 4.18 秒

可以看出两种算法效果差不多，但后一种算法快了 10 倍。

六、结果

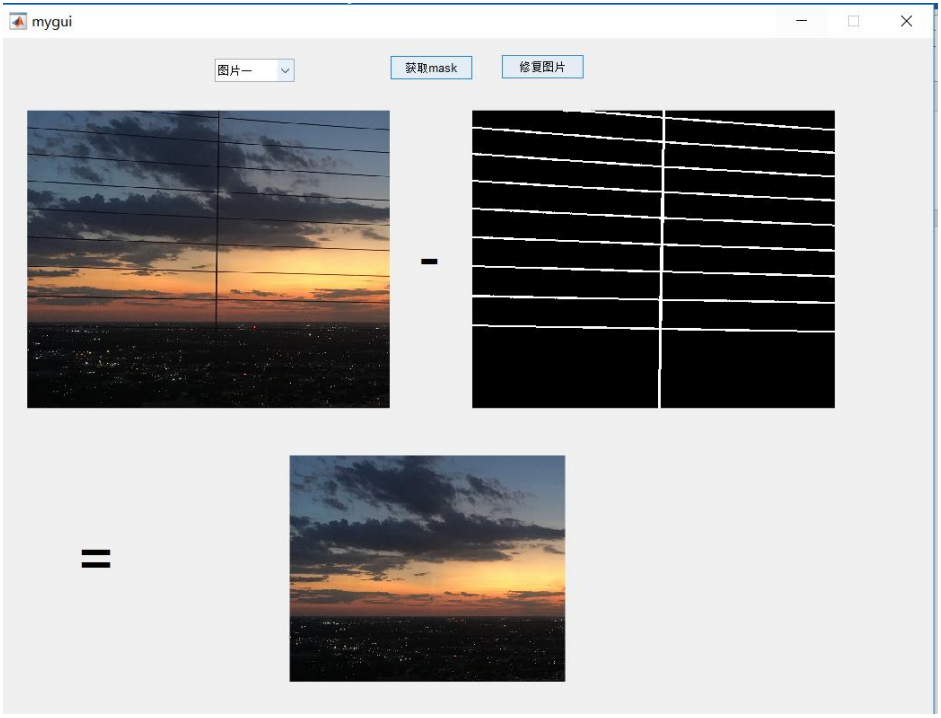
1、FFM 算法修复图片一



2、FFM 算法修复图片二

3、criminisi 算法和 FFM 算法修复图片二

4、UI 界面



七、总结

本次大作业有了第一次的经验，后期进行得比较顺利，但第一周还是经历了一段焦虑的时期，因为第一周做文献调研时，读了一些 90 年代的文献，它们的算法思路很好，但难以用计算机语言实现，比如等光线法(Masnou & Morel, 1998)，将待修复区域附近的等光线连起来，用等光

线上的像素值作为待修复区域像素值，得到的图片能有较好的平滑性。但这个算法很难用计算机实现。

后来又看了本世纪初的一些算法，发现 FFM 算法和 criminisi 算法应用广泛，论文中算法流程说的也比较清楚，同时符合本次大作业第一张图片先用简单方法，第二张图片再用复杂方法的要求，于是决定用这两个算法，接下来就比较顺利了。

这次大作业的遗憾是提取图片中栏杆位置信息时用的方法缺乏普遍性，且参数之间互相关联，难以拓展。

总之，经过本次大作业的推进，我对图像修复领域有了一个大致的了解，当然，该领域最前沿的研究与深度学习有关，也许将来有机会能接触到。

八、参考资料

https://github.com/cantarinigiorgio/Image-Inpainting

https://github.com/eodos/inpainting-telea-matlab

Criminisi, A., Perez, P., & Toyama, K. (2003, 18-20 June 2003). *Object removal by exemplar-based inpainting.* Paper presented at the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.

Masnou, S., & Morel, J. (1998, 7-7 Oct. 1998). *Level lines based disocclusion.* Paper presented at the Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269).

Telea, A. (2004). An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics Tools, 9*(1), 23-34. doi:10.1080/10867651.2004.10487596