泊松图像编辑

以源图像内梯度场为指导，将融合边界上目标场景和源图像的差异平滑地扩散到融合图像块 I 中。

一、 文章简介

1. 关于 paper

seamless editing of image regions
1. first set: seamless importation of both opaque and transparent source image regions.
2. second set: modify the appearance:(texture, illumination, color)

2. Intro

image editing: global changes || local changes
本文的 focus：**local changes**

- classic tools 的问题：visible seams
- 本文的 **math tool**：Poisson partial differential equation with Dirichlet boundary conditions.
- motivation:
  1. 感知上明显：图像中，拉普拉斯算子提取的二阶变化量在感知上最明显
  2. 唯一解：有界域上标量方程的解由边界上的值和内部的拉普拉斯算子**唯一**决定

二、 理论部分

1. 微分和卷积

∇表示位置x一阶微分计算（一阶中心导）：$\$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h}$

$\triangle$表示位置x二阶微分计算（二阶中心导）：$\frac{d^2 f(x)}{d^2 x} = \frac{f(x+h) - 2f(x) + f(x-h)}{2h^2}$

当 $h \to 0$ 时，上式的微分算式的结果会逐渐逼近真实的微分值。对于图像，图像的每个像素都是离散非连续的，因此这里的 h 放在实际的图像处理当中可看作为像素的间距，可视为1。将1代入上面的二阶微分计算式，我们可以将二阶微分的计算结果看作是一个 1×3 的卷积核 $[1, -2, 1]$ 在一维度数组上进行卷积计算的结果。

当数组维度变为二维数组时，也就是图像处理的拉普拉斯算子： $\triangle = \frac{\partial}{\partial x^2} + \frac{\partial}{\partial y^2}$

此时卷积核尺寸应该是 3×3 ,即

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$ ， 称为拉普拉斯卷积核。

2. 泊松方程的求解

已知图像每点的二阶微分值（即散度 div），求解各个图像点的像素值。

举个例子，假设有一张 4×4 的图像

$$X: \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix}$$

x_i 表示各个位置上的图像像素值，共16个未知参数需要被求解。

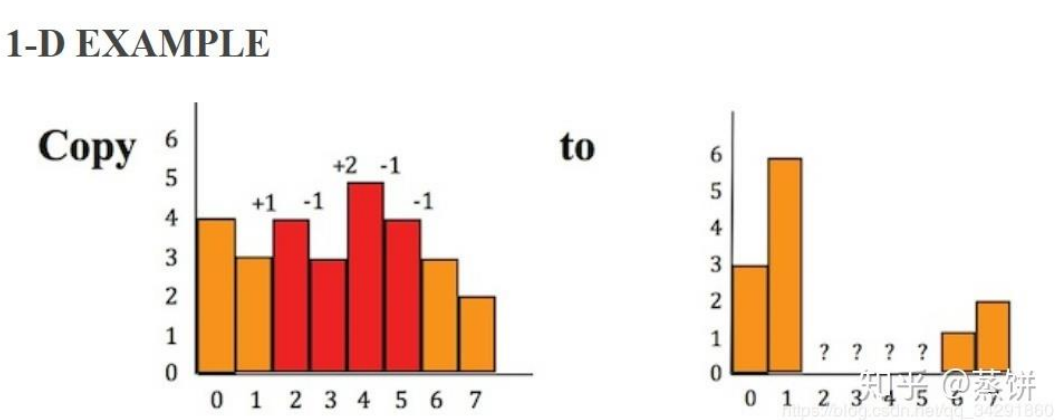
应用拉普拉斯卷积核后，得到4个方程式：

$$\begin{cases} x_2 + x_5 + x_7 + x_{10} - 4x_6 = \operatorname{div} x_6 \\ x_3 + x_6 + x_8 + x_{11} - 4x_7 = \operatorname{div} x_7 \\ x_6 + x_9 + x_{11} + x_{14} - 4x_{10} = \operatorname{div} x_{10} \\ x_7 + x_{10} + x_{12} + x_{15} - 4x_{11} = \operatorname{div} x_{11} \end{cases}$$

但是通过4个方程求解16个未知量是不可行的。但是如果边界的元素是已知的呢，即如果我们认为 $x_1, x_2, x_3, x_4, x_8, x_{12}, x_{16}, x_{15}, x_{14}, x_{13}, x_9, x_5$ 这几个边界元素是已知的，那么就剩下4个未知量，就可以求解了。

矩阵化该方程，的此式 $Ax = b$ 。

3. 一维的融合



目标：将红色的方块插入右边的“？？？？”。同时希望衔接自然。

fi 代表右图 i 方块上的高值。

f1=6, f6=1

$$\begin{cases} Min & ((f_2 - f_1) - 1)^2 \\ Min & ((f_3 - f_2) - (-1))^2 \\ Min & ((f_4 - f_3) - 2)^2 \\ Min & ((f_5 - f_4) - (-1))^2 \end{cases}$$

可以转化为下式：

$$\begin{aligned} Q = Min(&f_2^2 + 49 - 14f_2 \\ &+ f_3^2 + f_2^2 + 1 - 2f_3f_2 + 2f_3 - 2f_2 \\ &+ f_4^2 + f_3^2 + 4 - 2f_3f_4 - 4f_4 + 4f_3 \\ &+ f_5^2 + f_4^2 + 1 - 2f_5f_4 + 2f_5 - 2f_4 \\ &+ f_5^2 + 4 - 4f_5) \end{aligned}$$

然后我们可以得到下面这个式子：

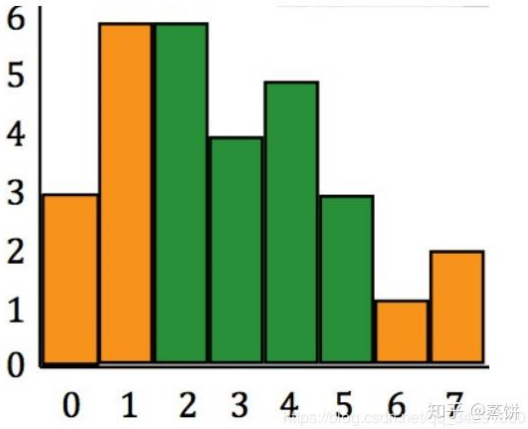
$$\begin{aligned} \frac{dQ}{df_2} &= 2f_2 + 2f_2 - 2f_3 - 16 \\ \frac{dQ}{df_3} &= 2f_3 - 2f_2 + 2 + 2f_3 - 2f_4 + 4 \\ \frac{dQ}{df_4} &= 2f_4 - 2f_3 - 4 + 2f_4 - 2f_5 - 2 \\ \frac{dQ}{df_5} &= 2f_5 - 2f_4 + 2 + 2f_5 - 4 \end{aligned}$$

转化为矩阵，即 $Ax = b$ 的形式表示为：

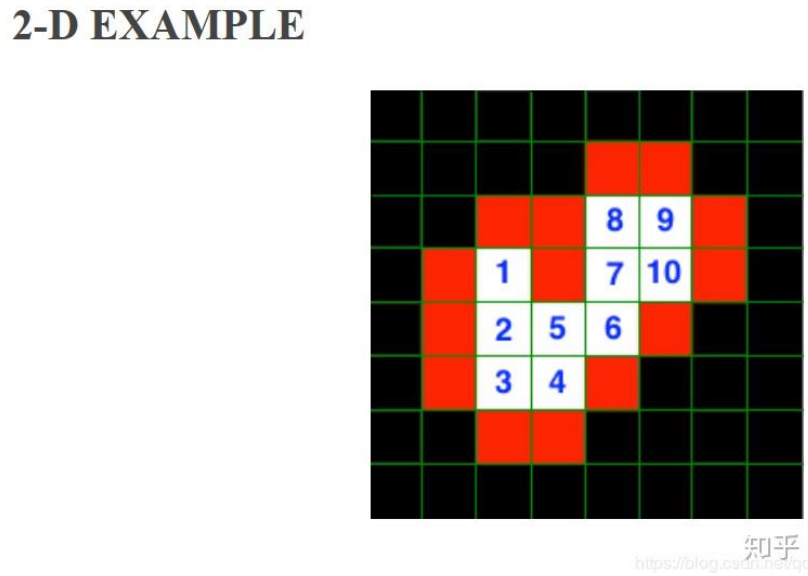
$$\begin{pmatrix} 4 & -2 & 0 & 0 \\ -2 & 4 & -2 & 0 \\ 0 & -2 & 4 & -2 \\ 0 & 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} f_2 \\ f_3 \\ f_4 \\ f_5 \end{pmatrix} = \begin{pmatrix} 16 \\ -6 \\ 6 \\ 2 \end{pmatrix}$$

解得 $f_2 = 6, f_3 = 4, f_4 = 5, f_5 = 3$

插入进去的效果图如下：



4. 二维的融合



有标号的像素为图像融合要插入的内容像素，红色的像素代表内容的边界。
以标号为 1 的像素为例

$$像素1_{up}*1+像素1_{left}*1+像素1_{right}*1+像素1_{down}*1-4*像素1=div(像素1)$$

像素 1 的相邻元素中，left、right、up 已知。像素 1 与像素 1down 为未知量。

$$像素1_{down}*1-4*像素1=div(像素1)-像素1_{up}*1-像素1_{left}*1-像素1_{right}*1$$

根据 Ax=b，我们已知 A 与 b，求解 x。x=[像素 1，像素 2，…,像素 10]
矩阵 A 的创建方法：

```
for i=1:row number
    for j=1:col number
        if(i==j)
            matrix(i, j)=-4
        elif(adjacent(pixel(j), pixel(i)))
            matrix(i, j)=1
        else
            matrix(i, j)=0
        end
    end
end
```


该伪代码的含义是，对于（i,j），若 i,j 相邻，则值为 1，若 i 与 j 是同一像素，值为-4，其他情况为 0。
该例子中 A 的矩阵如下:

	1	2	3	4	5	6	7	8	9	10
1	-4	1	0	0	0	0	0	0	0	0
2	1	-4	1	0	1	0	0	0	0	0
3	0	1	-4	1	0	0	0	0	0	0
4	0	0	1	-4	1	0	0	0	0	0
5	0	1	0	1	-4	1	0	0	0	0
6	0	0	0	0	1	-4	1	0	0	0
7	0	0	0	0	0	1	-4	1	0	1
8	0	0	0	0	0	0	1	-4	1	0
9	0	0	0	0	0	0	0	1	-4	1
10	0	0	0	0	0	0	1	0	1	-4

向量 b 的创建方法：

b[i]=div(像素 i)-(像素 i 相邻已知像素值的和)

求解 x：

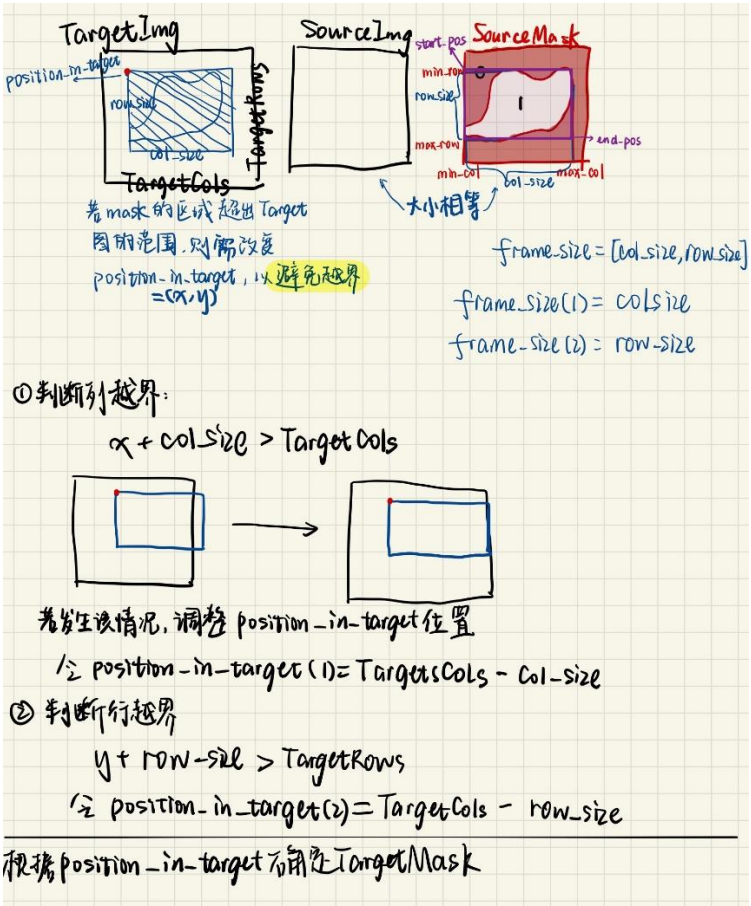
$$x = A^{-1} * b$$

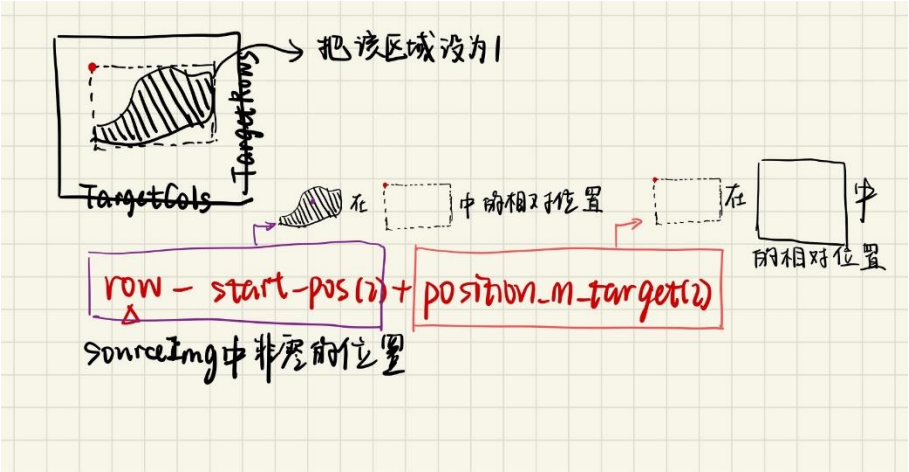
div(G(Source(x,y))) = -4f(x,y) + f(x-1,y) + f(x,y-1) + f(x+1,y) + f(x,y+1)

三、 具体实现方法（MATLAB）

1. 根据源图像的 mask，确定在目标图像上 clone 部分的 mask

注意需避免越界





2. Seamless cloning

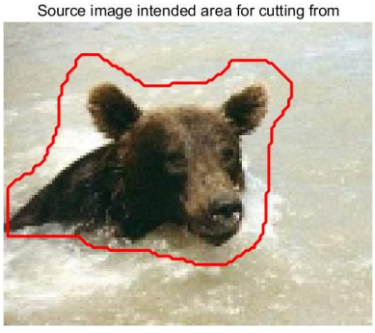
a. 对源图像执行拉普拉斯算子；
b. 计算 b 时，把 b 分为两部分，b1 是散度，b2 是已知邻接点的值的相反数；
c. 计算邻接矩阵 A（对角线元素 i = j 值为-4；若 i，j 为相邻点，值为 1）和 b2；
d. 合并拉普拉斯算子卷积结果 b1 和 b2，得到 b；
e. 解方程 Ax=b，求得 x；
f. 给目标图像上 mask 区域内的像素点重新赋值。

四、 效果展示

1. 示例 1

图片：

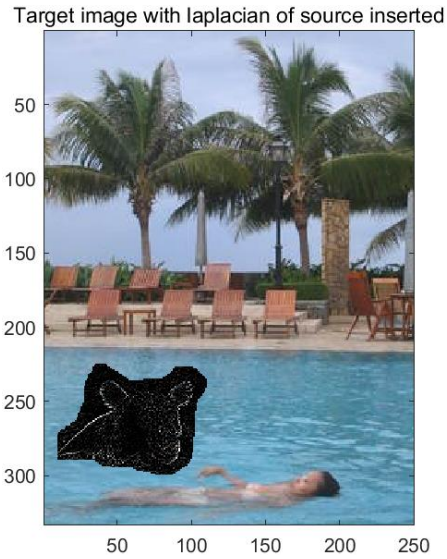
a. 原图上需要截取的区域



b. 目标图像上 paste 的区域

Target Image with intended place for pasting Source

c. 在 target 图像上 mask 区域上使用拉普拉斯卷积核

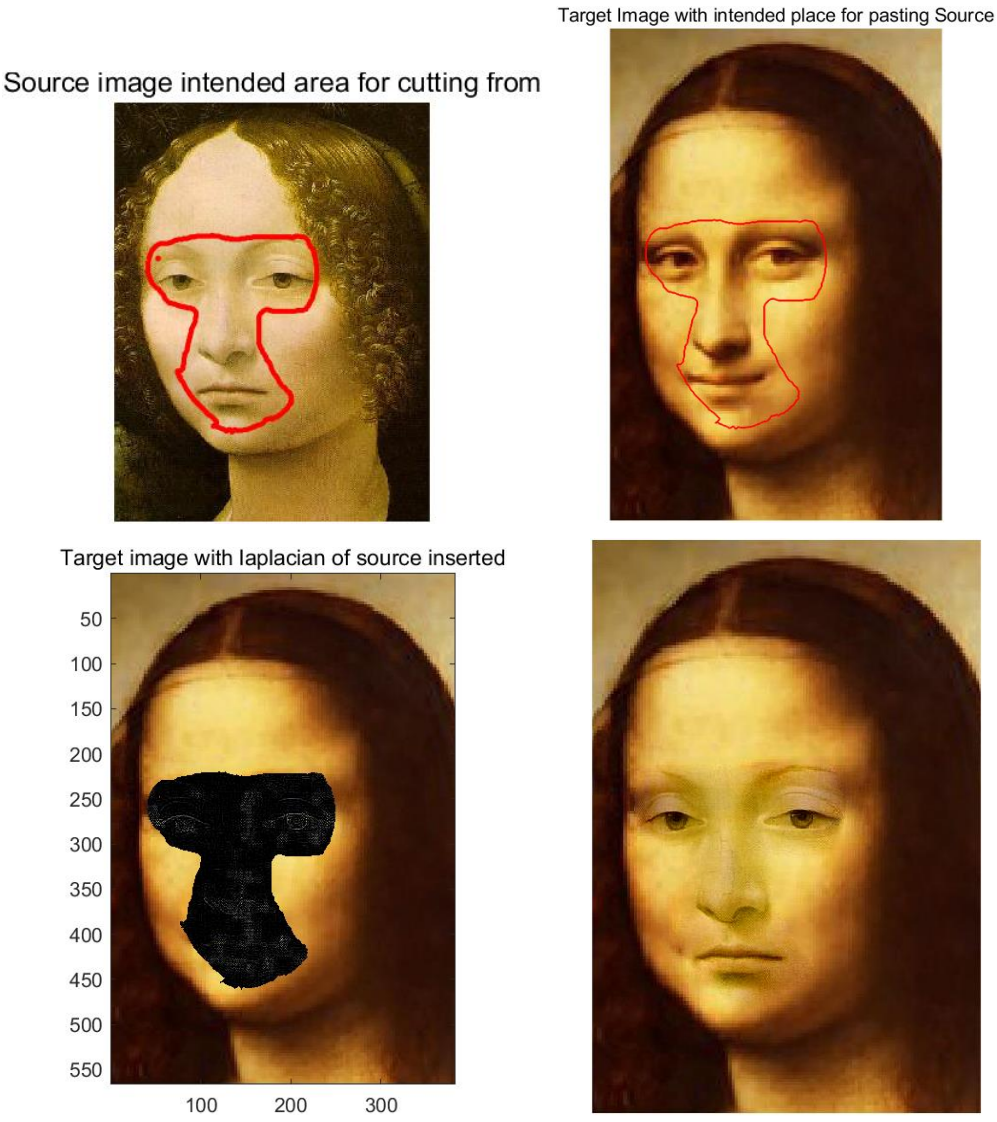


d. 最终效果

命令行

```
命令行窗口                                                    工作区
  cgs 在解的 迭代 68 处收敛，并且相对残差为 9.8e-05。
  cgs 在解的 迭代 70 处收敛，并且相对残差为 9.7e-05。
  cgs 在解的 迭代 71 处收敛，并且相对残差为 9.7e-05。
fx >>
```

2. 示例 2



五、 代码

1. 主程序 poisson_image_editting

```
% 清理环境
```

```
close all;
clear;
clc;

%添加图像路径
addpath img

% 用户设定
% 设定源图、目标图像、mask 的路径，设定目标图像 paste 区域的起始坐标点
% TargetImgPath = 'poolTarget.jpg';
% SourceImgPath = 'bear.jpg';
% SourceMaskPath = 'bearMask.jpg';
TargetImgPath = 'femaleTarget.png';
SourceImgPath = 'femaleSource.png';
SourceMaskPath = 'femaleMask1.png';

% 设定要将 source 中轮廓内的图像粘贴到 target 图中具体哪个位置
% position_in_target = [10, 225];%狗熊
position_in_target = [42, 220];%蒙娜丽莎

% 读入三张图片
TargetImg = imread(TargetImgPath);
SourceImg = imread(SourceImgPath);
SourceMask = im2bw(imread(SourceMaskPath));

% 获取 mask 的二值图的对象轮廓
[SrcBoundary,L] = bwboundaries(SourceMask, 8);

% 绘制裁剪的轮廓
figure, imshow(SourceImg), axis image
hold on
for k = 1:length(SrcBoundary)
   boundary = SrcBoundary{k};
   plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 2)
end
title('Source image intended area for cutting from');


% 获取目标图像大小
[TargetRows, TargetCols, ~] = size(TargetImg);
% SourceMask 中非零的部分
[row, col] = find(SourceMask);

% mask 框在 source 图中的大小
```

```
start_pos = [min(col), min(row)];
end_pos   = [max(col), max(row)];
frame_size  = end_pos - start_pos;

% 判断是否出现越界，若出现越界，则调整 position_in_target
if (frame_size(1) + position_in_target(1) > TargetCols)
    position_in_target(1) = TargetCols - frame_size(1);
end

if (frame_size(2) + position_in_target(2) > TargetRows)
    position_in_target(2) = TargetRows - frame_size(2);
end

% 构建一个与 Target 图相等的新的 mask
MaskTarget = zeros(TargetRows, TargetCols);

% ind = sub2ind(sz,row,col) 针对大小为 sz 的矩阵返回由 row 和 col 指定的行列下标的对应线性索引 ind。
% 此处，sz 是包含两个元素的向量，其中 sz(1) 指定行数，sz(2) 指定列数。
% 利用线性索引，把 TargetImg 中对应位置的值赋值为 1。
MaskTarget( sub2ind( [TargetRows, TargetCols], row - start_pos(2) + position_in_target(2), ...
col - start_pos(1) + position_in_target(1) ) ) = 1;

% 获取二值图像中对象的轮廓，把这个轮廓在 TargetImg 上绘制出来
TargBoundry = bwboundaries(MaskTarget, 8);
figure, imshow(TargetImg), axis image
hold on
for k = 1:length(TargBoundry)
    boundary = TargBoundry{k};
    plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 1)
end
title('Target Image with intended place for pasting Source');

templt = [0 1 0; 1 -4 1; 0 1 0];  % 拉普拉斯算子
LaplacianSource = imfilter(double(SourceImg), templt, 'replicate');  % 对源图像执行拉普拉斯算子
VR = LaplacianSource(:, :, 1);
VG = LaplacianSource(:, :, 2);
VB = LaplacianSource(:, :, 3);

% 取出目标图像的 R、G、B
TargetImgR = double(TargetImg(:, :, 1));
TargetImgG = double(TargetImg(:, :, 2));
```

```
TargetImgB = double(TargetImg(:, :, 3));

% 给 mask 区域赋值，R,G,B 需要分开赋值
% 注意 MaskTarget 是 double 类型，SourceMask 已经是 logical 类型
TargetImgR(logical(MaskTarget(:))) = VR(SourceMask(:));
TargetImgG(logical(MaskTarget(:))) = VG(SourceMask(:));
TargetImgB(logical(MaskTarget(:))) = VB(SourceMask(:));

% 合并 3 通道，形成新的图像
TargetImgNew = cat(3, TargetImgR, TargetImgG, TargetImgB);
% 绘制新的图像
figure, imagesc(uint8(TargetImgNew)), axis image, title('Target image with lap
lacian of source inserted');

% 计算邻接矩阵 A 和 b2
[A,b] = calcAdjancency( MaskTarget, TargetImgR, TargetImgG, TargetImgB );

% 计算 b
b1 = b(:,1) + VR(SourceMask(:));
b2 = b(:,2) + VG(SourceMask(:));
b3 = b(:,3) + VB(SourceMask(:));

% 求解 Ax = b 中的 x
RX = cgs(A,b1,1e-4,100);
GX = cgs(A,b2,1e-4,100);
BX = cgs(A,b3,1e-4,100);
% RX = A\b1;
% GX = A\b2;
% BX = A\b3;

% 取出目标图像的 R、G、B
FinalImgR = double(TargetImg(:, :, 1));
FinalImgG = double(TargetImg(:, :, 2));
FinalImgB = double(TargetImg(:, :, 3));

% 给 mask 区域赋值，R,G,B 需要分开赋值
% 注意 MaskTarget 是 double 类型，SourceMask 已经是 logical 类型
FinalImgR(logical(MaskTarget(:))) = RX;
FinalImgG(logical(MaskTarget(:))) = GX;
FinalImgB(logical(MaskTarget(:))) = BX;

% 合并 RGB 三分量
ResultImg = cat(3, FinalImgR, FinalImgG, FinalImgB);
```

```
% 显示最终融合效果
figure;
imshow(uint8(ResultImg));
```

2. 函数 calcAdjancency

```
function [neighbors,b] = calcAdjancency( Mask, TargetImgR, TargetImgG, TargetI
mgB )
% 计算稀疏的邻接矩阵 A & b 的一部分(b2)
%b 分为两部分，第一部分是 div(定义为 b1)，第二部分是已知邻接点的相反数(定义为 b2)
[height, width]      = size(Mask);
[row_mask, col_mask] = find(Mask);

% length(row_mask)代表有多少个待解像素点
% neighbors = zeros(length(row_mask), length(row_mask));
neighbors = sparse(length(row_mask), length(row_mask), 0);
b = zeros(length(row_mask),3);

%下标转线性索引
roi_idxs = sub2ind([height, width], row_mask, col_mask);

%求 A
for k = 1:size(row_mask, 1)
    neighbors(k, k) = -4;
    %4 邻接点
    connected_4 = [row_mask(k), col_mask(k)-1;%left
                   row_mask(k), col_mask(k)+1;%right
                   row_mask(k)-1, col_mask(k);%top
                   row_mask(k)+1, col_mask(k)];%bottom

    ind_neighbors = sub2ind([height, width], connected_4(:, 1), connected_4(:,
 2));

    for neighbor_idx = 1: 4 %number of neighbors,
        adjacent_pixel_idx =  ismembc2(ind_neighbors(neighbor_idx), roi_idxs);
  %判断临接点是否是待解的未知点
        % 注：ismembc2 是 matlab 的二分查找函数，i = ismembc2(t, X)，
        % 注：返回 t 在 X 中的位置，其中 X 必须为递增的的数值向量
        if (adjacent_pixel_idx ~= 0)  % 该临接点是待解的未知点
            neighbors(k, adjacent_pixel_idx) = 1;  %若待解的点两者相邻，则赋值为 1
        else % 该临接点不是待解的未知点
```

```
            %首先判断这个邻接点在 Mask 中的线性索引位置
            %再求出它在 TargetImg 中的线性索引位置
            b(k, 1) = b(k, 1) - TargetImgR(connected_4(neighbor_idx, 1),connecte
d_4(neighbor_idx, 2));
            b(k, 2) = b(k, 2) - TargetImgG(connected_4(neighbor_idx, 1),connecte
d_4(neighbor_idx, 2));
            b(k, 3) = b(k, 3) - TargetImgB(connected_4(neighbor_idx, 1),connecte
d_4(neighbor_idx, 2));
        end
    end
end


end
```