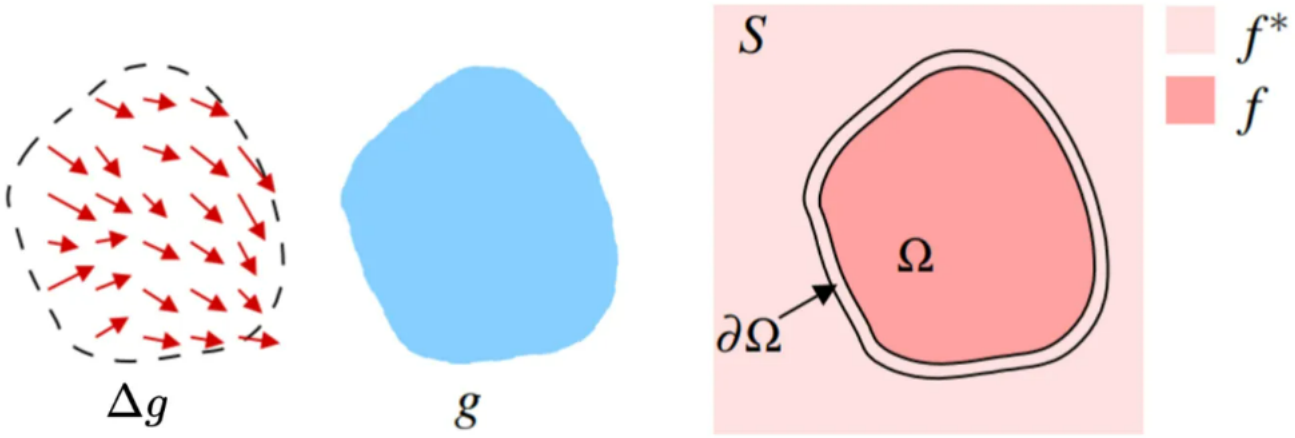
# 第三次作业报告

PB21010362 汪兆辰

2023 年 3 月 26 日

## 1 算法介绍




Poisson 融合算法要求参考图与目标图的边值相等，而内部的梯度相等，即

$$\begin{cases} f = f^*|_{\partial\Omega} \\ \bigtriangledown f = \bigtriangledown g|_{\Omega} \end{cases} \tag{1}$$

这也就是求下式的最小值：

$$\min \iint_{\Omega} |\bigtriangledown f - \bigtriangledown g|^2 \tag{2}$$

也即求解带 Dirichlet 边值的 Poisson 方程：

$$\triangle f = \triangle g = div(\bigtriangledown g), s.t. f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{3}$$

而在图像中，需要将上述条件离散化. 由于 g 已知，其 Laplace 可以直接求得：

$$\triangle g = g(x+1,y) + g(x-1,y) + g(x,y+1) + g(x,y-1) - 4g(x,y) \tag{4}$$

对应的 f 就是：

$$\triangle f = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{5}$$

且满足 $\triangle g = \triangle f$.

而考虑到上式中并非所有点均存在，对于在 f 的边界上的点需要代入边界条件求解. 假设对于 (x,y) 点，其邻点 (x+1,y) 在边界上，则可列方程为：

$$\triangle g - f^*(x+1,y) = f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{6}$$

对于其他情况同理.

为了解出 f 每一点处的取值，需要对每一点建立上述方程并解出方程组. 令 A 为系数矩阵，x 为 f 内部所有点组成的列向量，为待解的未知数，V 为 g 的 Laplace 减去边值. 上述问题就变成了求解矩阵方程：

$$Ax = V \tag{7}$$

# 2 算法实现

算法的实现过程分为三步，第一步判断区域的内点并用索引矩阵存储其序号；第二步遍历 f 的所有内点从而建立方程组；第三步求解方程组并对背景图像赋值.

## 2.1 判断内点

为了保证所有内点都被遍历，需要遍历足够多的点，我们考虑以点列 roi 最外部的四个点生成包含所有内点的矩形，从而减少不必要的遍历. 被找到的内点被存储在索引数组 Map 中并按扫描顺序标号，从而方便后续建立矩阵 A 和 V 时快速找到对应点的位置.

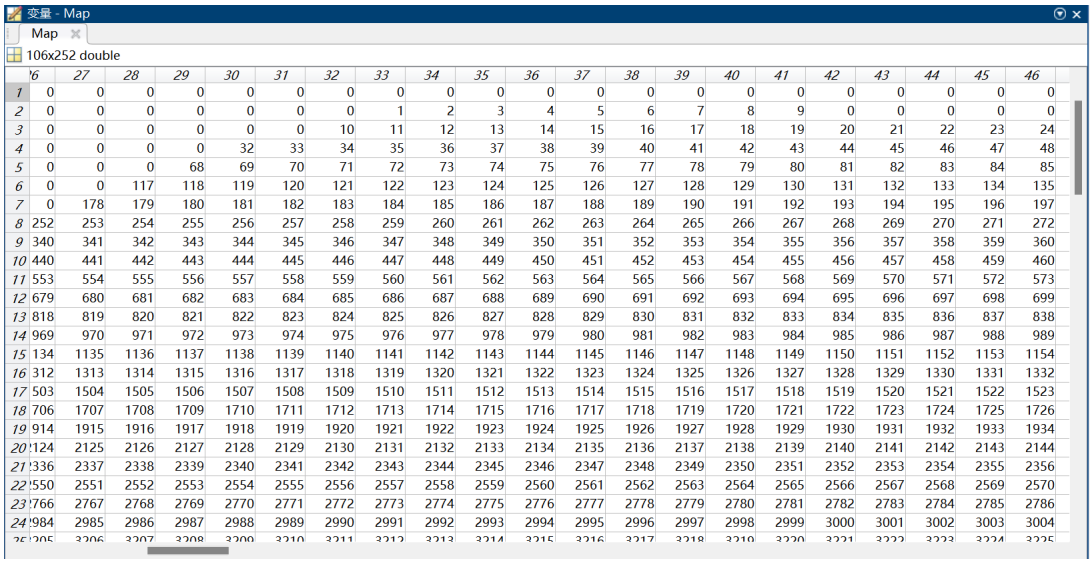



图 1: 如图所示，外部的点都被标记为 0，而内部的点按照顺序被依次编号

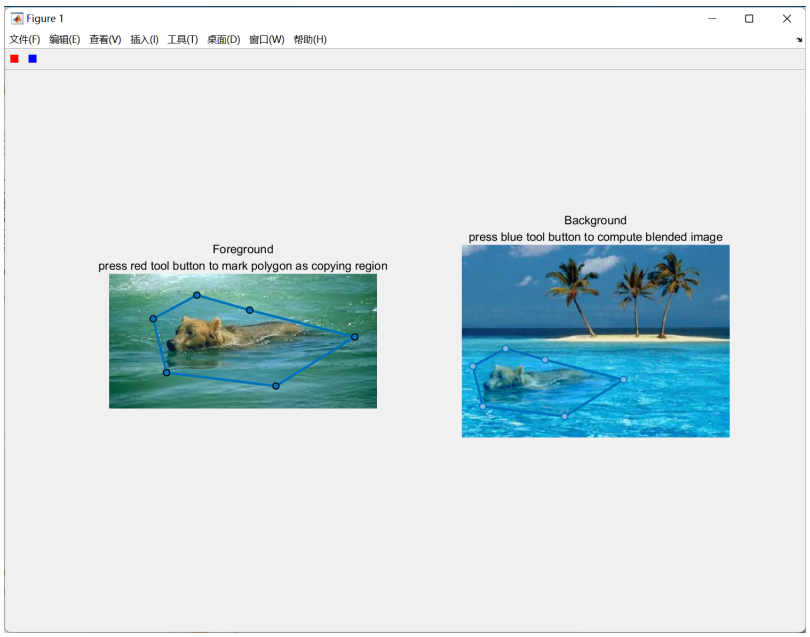
这里用到了内置函数 inpolygon，用来判断给定点的坐标是否在给定区域内.

## 2.2 建立方程组

由于方程中有 Laplace 算子，在程序中使用了两次内置的函数 imgradientxy，对二维的图像求梯度. 求两次梯度后将 x,y 方向上的二阶偏导相加就得到了 Laplace. 然后经过一次遍历建立系数矩阵 A 与 laplace 向量 V:

```
V=zeros(index,1,"double");
for i=1:m
    for j=1:n
        neighbors=[i,j+1;i,j-1;i-1,j;i+1,j];
        if Map(i,j)~=0
            V(Map(i,j))=lap(i,j);
            if i==1||i==m||j==1||j==n
                V(Map(i,j))=V(Map(i,j))-bg(i,j);
            else
                for h=1:4
                    if Map(neighbors(h,1),neighbors(h,2))==0
                        V(Map(i,j))=V(Map(i,j))-bg(neighbors(h,1),neighbors(h,2));
                    else
                        A(Map(i,j),Map(neighbors(h,1),neighbors(h,2)))=1;
                    end
                end
            end
        end
    end
end
```

## 2.3 实现效果



# 3 Bonus: 实时返回结果

由于在目标点 targetPosition 移动的过程中，围成的多边形区域不变，前面计算的系数矩阵 A 可以直接使用，并通过调用 decomposition 命令对稀疏矩阵预分解，加快计算速度.

为了实现实时结果，我们添加回调函数 RealtimeCallback，并对选区的移动事件 MovingROI 注册回调函数. 同时将回调函数需要的变量上传至工作区:

```
assignin("base","decA",decA);
assignin("base","Map",Map);
assignin("base","im2copy",im2copy);
assignin("base","index",index);
hpolys=evalin("base","hpolys");
hCallback=addlistener(hpolys(2),"MovingROI",@RealtimeCallback);
assignin("base","hCallback",hCallback);
```

在回调函数中，算法与前述函数中相同，其中部分变量直接使用工作区中上传的变量，无需再次赋值，这样我们就可以拖动选区得到实时的计算结果. 计算结果是动态的，此处不再展示.

# 参考文献

[1] Patrick Perez, Michel Gangnet and Andrew Blake. Poisson Image Editing. SIGGRAPH2003.