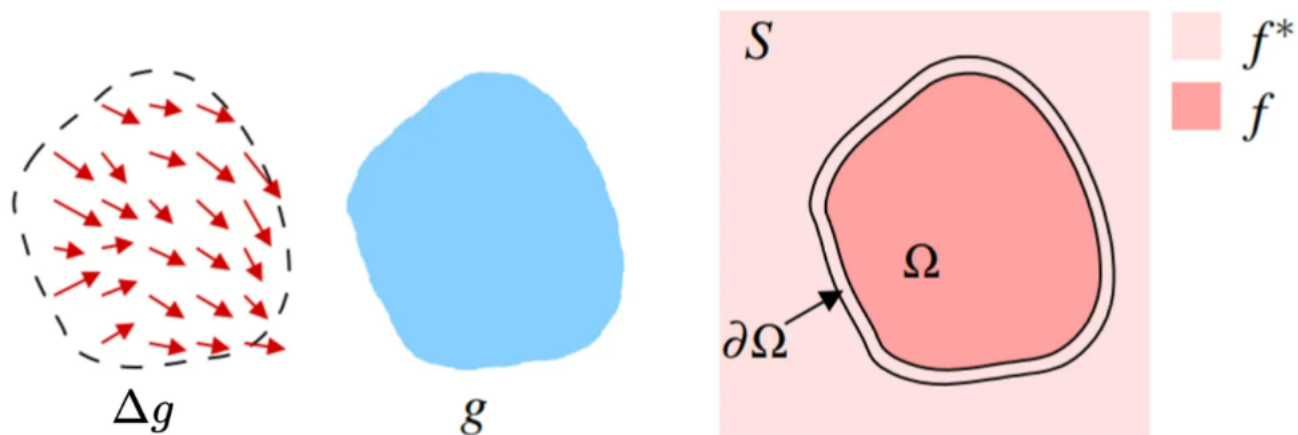


# 第三次作业报告

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

$$\begin{cases} f = f^*|_{\partial\Omega} \\ \nabla f = \nabla g|_{\Omega} \end{cases} \quad (1)$$

这也就是求下式的最小值：

$$\min \iint_{\Omega} |\nabla f - \nabla g|^2 \quad (2)$$

也即求解带 Dirichlet 边值的 Poisson 方程：

$$\Delta f = \Delta g = \operatorname{div}(\nabla g), s.t. f|_{\partial\Omega} = f^*|_{\partial\Omega} \quad (3)$$

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：

$$\Delta g = g(x+1, y) + g(x-1, y) + g(x, y+1) + g(x, y-1) - 4g(x, y) \quad (4)$$

对应的  $f$  就是：

$$\Delta f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (5)$$

且满足  $\Delta g = \Delta f$ .

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：

$$\Delta g - f^*(x+1, y) = f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (6)$$

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：

$$Ax = V \quad (7)$$

## 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  $Map$  中并按扫描顺序编号，从而方便后续建立矩阵  $A$  和  $V$  时快速找到对应点的位置。

	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	9	0	0	0	0
3	0	0	0	0	0	0	10	11	12	13	14	15	16	17	18	19	20	21	22	23
4	0	0	0	0	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
5	0	0	0	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
6	0	0	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134
7	0	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196
8	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
9	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359
10	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459
11	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572
12	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698
13	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837
14	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988
15	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153
16	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327	1328	1329	1330	1331
17	1503	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519	1520	1521	1522
18	1706	1707	1708	1709	1710	1711	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725
19	1914	1915	1916	1917	1918	1919	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933
20	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
21	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355
22	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569
23	2766	2767	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783	2784	2785
24	2984	2985	2986	2987	2988	2989	2990	2991	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003
25	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215	3216	3217	3218	3219	3220	3221	3222	3223	3224

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

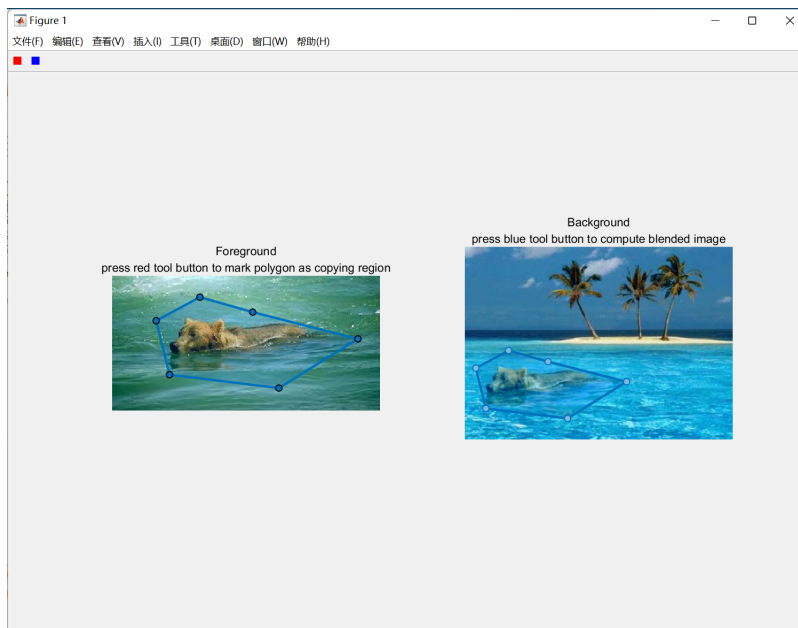
这里用到了内置函数 `inpolygon`，用来判断给定点的坐标是否在给定区域内。

## 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.