Tsinghua University, Department of Computer Science and Technology

Efficient Gradient-Domain Compositing Using Quadtrees

刘泓尊　2018011446　计 84

2022 年 2 月 23 日

1　算法介绍

该算法 [1] 提供一种高效的图像融合手段，利用四叉树划分图像区间，使得四叉树的一个节点的解可以近似代替该区域所有像素的解，从而将算法时间复杂度从 $O(n)$ 降低到 $O(p)$, 其中 n 是融合区域的像素个数，p 是融合边界的长度，使得算法复杂度接近线性。

该算法没有直接去解图像融合的泊松方程 $Ax = b$, 而是考虑到最终解和被嵌入区域的差值 x_δ, 即 $x = x_0 + x_\delta$. 那么考虑到 $Ax = b$ 的最小二乘解，可以转换为求解：

$$A^T A x_\delta = A^T (b - A x_0)$$

因为图像有局部性连续的特征，可以认为 $(b - Ax_0)$ 在离融合边界远的区域均接近 0，所以可以利用四叉树缩点来降低求解方程的规模。

假设四叉树缩点之后图像有 y 个关键点，那么我们希望全部像素 x 能够是 y 的线性组合，即存在矩阵 S 使得 $x = Sy$。关于 S 的构造，可以采用双线性插值的方法，四叉树节点所覆盖区域中的像素就是其区域四个顶点像素值的线性插值。从而上述方程可以转换为：

$$S^T A^T A S y_\delta = S^T A^T (b - A S y_0)$$

在构建四叉树之后，等式右侧的 $S^T A^T (b - AS y_0)$ 遍历一遍四叉树即可求得。而等式左边的 $S^T A^T AS$ 也可以在遍历过程中沿着四叉树节点的边缘对每对相邻像素之间的梯度加权求和。所以算法开销就是遍历四叉树与求解线性方程组的总和。

算法流程如下：

1. 根据融合区域边界和图像边界构建四叉树
2. 遍历四叉树得到关键点，并建立矩阵 S 和矩阵 AS, 以及等式右边的 $(b - AS y_0)$
3. 利用最小二乘法求解 $AS y_\delta = (b - AS y_0)$ 的最优解
4. 根据 y_δ 还原每个像素值，融合

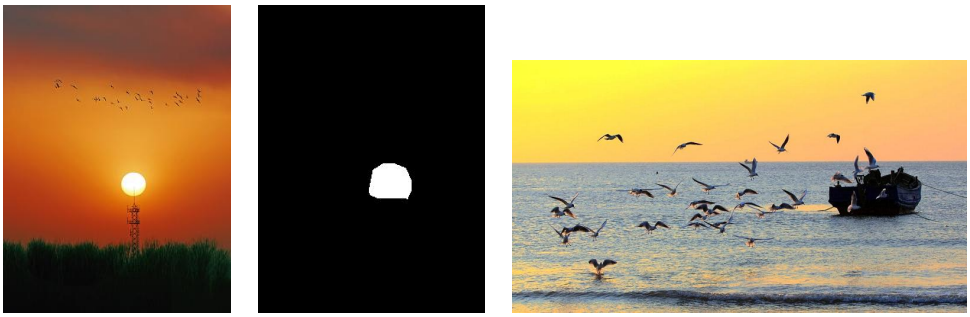
2 实例

我选择了春季学期“数字图像处理”课程提供的若干测例，每个测例包括被融合图、融合图以及 mask 图用于标注融合边界。

表1展示了这两个测例需要的时间和大致存储空间（用存储线性方程组所占用空间近似计算）的对比，以及优化后相对于优化前的精确解的均方误差 RMS(average per-pixel root-mean-square error) 以及 R 通道上的最大误差 (max error on R channel)。

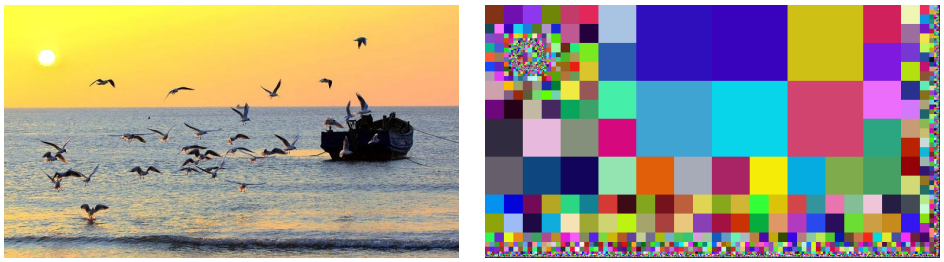
表 1: 两个例子的性能对比与误差统计

	优化前用时 (s)	优化后用时 (s)	优化前空间 (MB)	优化后空间 (MB)	均方误差	最大误差 (R 通道)	融合区域像素
Example1	67.32	**0.87**	2.508	**0.036**	0.002	1.000	360*266
Example2	15.49	**0.50**	1.135	**0.034**	0.094	1.000	120*180



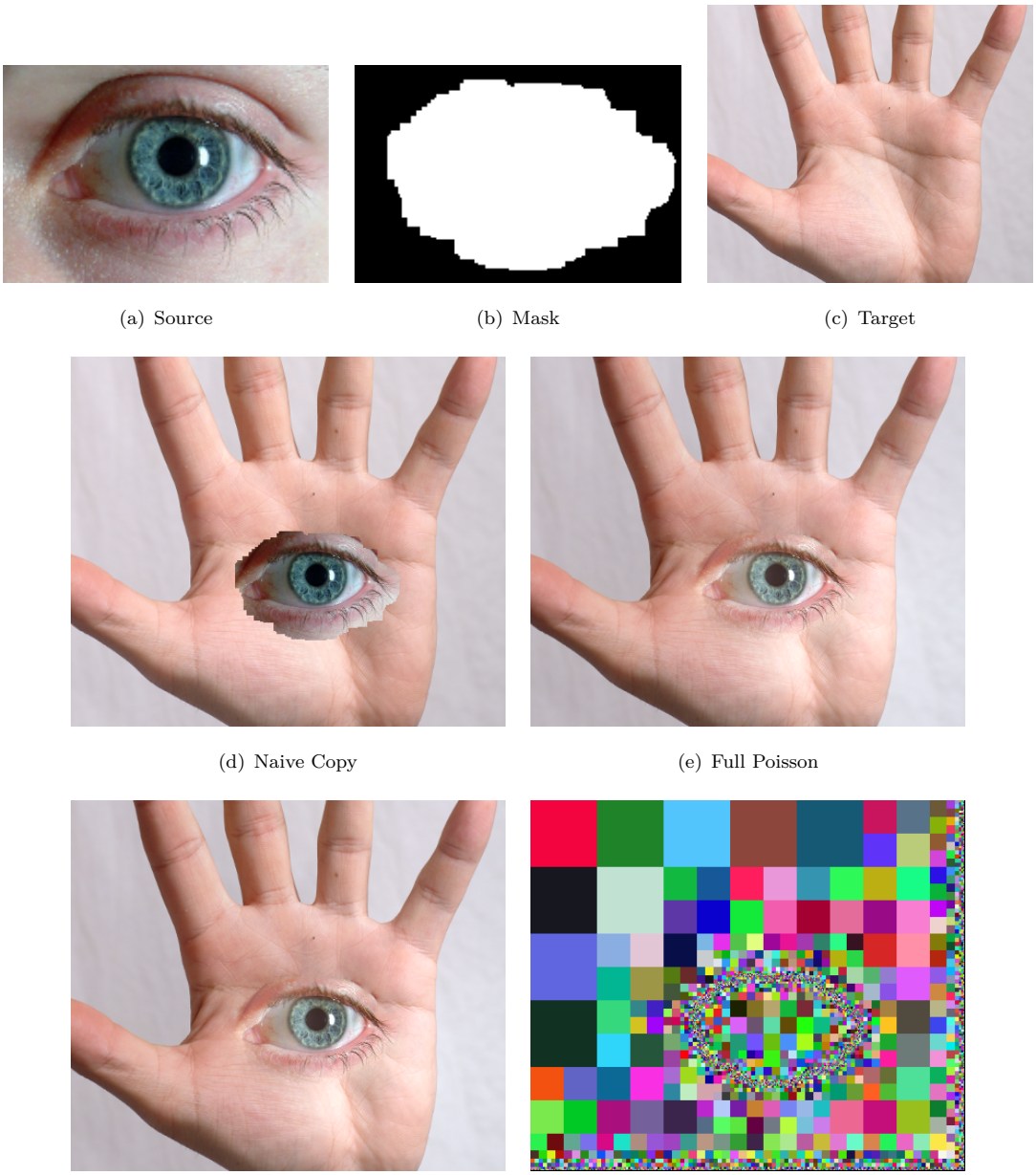
(a) Source (b) Mask (c) Target

(d) Naive Copy (e) Full Poisson



(f) Quadtree Poisson (g) Quadtree

图 1: Image Fusion Example 1 (“Full Poisson”即使用优化前的求解全像素泊松方程的结果；“Quadtree Poisson”是使用四叉树优化之后的算法结果；“tree”是构建的四叉树可视化示意图)



(a) Source (b) Mask (c) Target

(d) Naive Copy (e) Full Poisson

(f) Quadtree Poisson (g) Quadtree

图 2: Image Fusion Example 2

可以看到使用四叉树优化之后的融合结果与优化前的结果在肉眼上没有明显区别，融合区域平滑自然。对于构造的四叉树特征，可以看到在离融合边界较远的区域，四叉树节点较大；而融合边界和图像边界都使得四叉树节点逐渐缩小到一个像素范围，和前面的分析是一致的。

从表1可以看到，使用四叉树优化之后的算法复杂度大幅缩减，在两个测例上的时间和空间开销都有了 100 倍的提升。相当于精确解，使用四叉树优化的近似解误差很小，任一像素的误差都不超过 1.0。

参考文献

[1] Aseem Agarwala. Efficient gradient-domain compositing using quadtrees. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 94–es, New York, NY, USA, 2007. Association for Computing Machinery.

A 运行说明

`src/`文件夹下是本项目源码，同时提供了 `Makefile` 方便编译。

1. 本项目使用了 `eigen3` 库用以加速方程求解，运行前需要自行配置
2. 测试环境为 Ubuntu 20.04 on x86 CPU, 使用 GNU g++ 编译

程序使用命令行参数解析，运行示例如下：

运行示例

```
$ cd src/
$ make

$ ./main -?
usage: ./main --source=string --target=string --mask=string --xoffset=int --yoffset=int [options] ...
options:
    -s, --source    source image to be embedded into target image. (string)
    -t, --target    target image. (string)
    -m, --mask      mask image. (1 channel) (string)
    -x, --xoffset   x-dim offset to composite. (int)
    -y, --yoffset   y-dim offset to composite. (int)
    -r, --resultdir directory to save results. (string [=results/])
    -?, --help      print this message

$ make example1
./main -s ../pic/test1_src.png -t ../pic/test1_target.png -m ../pic/test1_mask.png -r example1 -x -120 -y -80
args:
        source image: ../pic/test1_src.png
        target image: ../pic/test1_target.png
        mask  image: ../pic/test1_mask.png
        x offset: -120
        y offset: -80
        directory to save results: example1/

load image from ../pic/test1_src.png (H, W, C)=360 266 3
load image from ../pic/test1_target.png (H, W, C)=427 770 3
load image from ../pic/test1_mask.png (H, W, C)=360 266 3
final image width: 770 height: 427

simple copy...
build full equation...
full problem scale: 328790
full memory used: 2.50847 MB
```

```
time elasped (full): 75.68s

building QuadTree...
find seam points: 296
building QuadTree Equation...
quadtree problem scale: 4813
quadtree memory used: 0.0367203 MB
time elasped (quadtree): 0.90s

RMS error over all pixels on R channel: 0.002
max 1-norm error over all pixels on R channel: 1.000

$ ./main -s ../pic/test2_src.png -t ../pic/test2_target.png -m ../pic/
    test2_mask.png -r example2 -x 160 -y 140
...(Omitted for brevity)
```