

# Efficient Gradient-Domain Compositing Using Quadtrees Experiment Report

2014011355

辛杭高

## Abstract

本次实验中，我实现了论文 Efficient Gradient-Domain Compositing Using Quadtrees 中的算法，能够完成图象融合工作。报告中我介绍了关于本算法的细节，在文末，我附上了一些本次实验的结果，并将实验的结果与未使用 Quadtree 的泊松融合算法进行了比较。

## 1. 泊松融合的思想及局限

图片融合的任务中，如果直接将原图的部分拷贝到目标图片中，会产生比较明显的融合边界。因为在融合边界处，图片的梯度是由人为因素造成的。为了增强融合后图片的真实性，该方法将融合区域的梯度修正为目标梯度，从而达到消除融合边界的效果。目标梯度的选取是多样的，可以选择原图像相应区域的梯度，也可以选择目标图像相应区域的梯度，还可以是两者的混合。

以修改梯度为目标的思想可以转化为方程：

$$Ax = b \quad (1)$$

其中  $x$  是融合区域各个像素组成的向量， $b$  是目标梯度。为方便表述，我们用  $n$  来表示待融合区域像素的数目。 $A$  是用来求像素梯度的稀疏矩阵，规模为  $n^2$ ，每一行仅有两个元素非零。

这是一个过约束线性系统，因此在欧式距离的意义上减小  $Ax$  与  $b$  的差异，可转化为线性系统

$$Ax A^T = A^T b \quad (2)$$

在方程 (2) 里，Laplacian 变化的每行最多有 5 个元素非零。在解方程的时候会有比较高的复杂度，

这给泊松融合的时间和空间都带来很大的压力。利用 Quadtree 可以极大地减小矩阵  $A$  的规模，通常可以将  $n * n$  降低到  $\sqrt{n} * \sqrt{n}$ 。

## 2. 优化的核心思想

对 (1) 中的  $x$  重新进行表述：

$$x = x_0 + x_\delta \quad (3)$$

则 (2) 中的方程可以改写为：

$$A^T A x_\delta = A^T (b - Ax_0) \quad (4)$$

此时我们选择原图像的梯度作为  $b$ 。显然，如果  $x$  的 Laplacian 变化不需要涉及到融合区域的边界，那么 (4) 式方程的右边为 0。

因此，对于融合区域的所有内部元素方程  $A^T A x_\delta = 0$  成立。此式子成立为双线性插值提供了支撑，保证了双线性插值的合理性。

综上，我们只需要参照某种依据将待融合区域划分成 Quadtree 的子节点，在 Quadtree 的叶节点上进行双线性插值，即可解出待融合区域各像素点的 RGB 值。

## 3. 双线性插值

双线性插值是有两个变量的插值函数的线性插值扩展，其核心思想是在两个方向分别进行一次线性插值。红色的数据点与待插值得到的绿色点。假如我们想得到未知函数  $f$  在点  $P = (x, y)$  的值。假设我们已知函数  $f$  在  $Q_{11} = (x_1, y_1)$ ,  $Q_{12} = (x_1, y_2)$ ,  $Q_{21} = (x_2, y_1)$ ,  $Q_{22} = (x_2, y_2)$  四个点的值。

首先在  $x$  方向进行线性插值，得到

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad R_1 = (x, y_1), \quad (5)$$

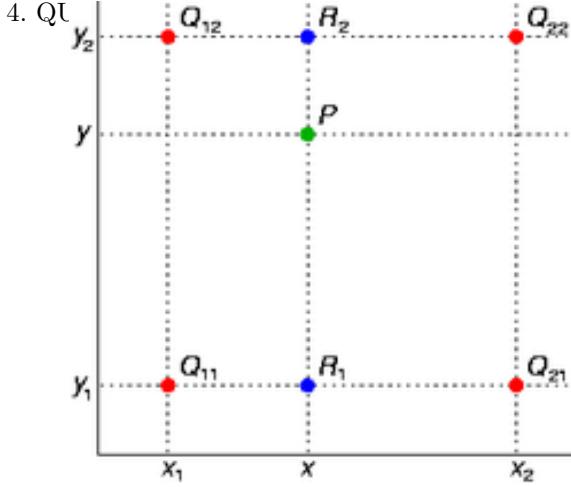


图 1. 红色的数据点与待插值得到的绿色点。

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad R_1 = (x, y_2) \quad (6)$$

然后在 y 方向进行线性插值，得到

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2). \quad (7)$$

用矩阵运算表示为

$$f(x, y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0, 0) & f(0, 1) \\ f(1, 0) & f(1, 1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix} \quad (8)$$

### 3.1. T-Junction

由上述定义可知，为了完成双线性插值的工作，除了以左上角的节点为基点之外，还需要引入右侧相邻叶节点的像素值和下方相邻叶节点的像素值。

由于 Quadtree 的划分过程本身对子节点之间的关系并没有要求，因此对于某个叶节点来说，与其相邻的节点规模难以确定，致使我们无法求出右侧相邻节点和下方相邻节点的像素值。

为了能够精确求出某叶节点右侧和正下方叶节点的像素值，我们规定某个节点与其相邻节点规模的差异不能超过两倍，否则需要进一步对 Quadtree 的划分进行细化。

此时分为三种情况，若某叶节点的相邻节点与其规模相同，则显然不形成 T-Junction。若相邻节点的规模是本节点的一半，那也不形成 T-Junction，因为此时右

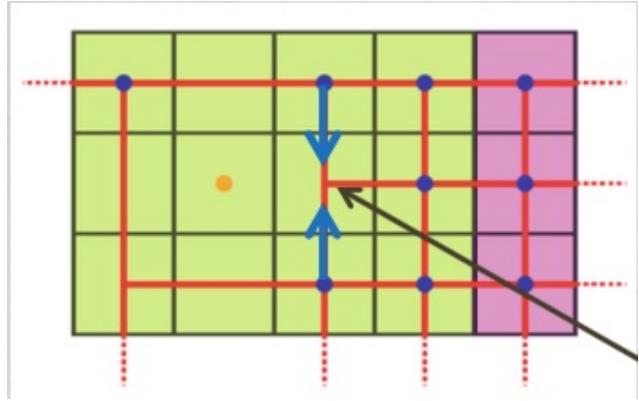


图 2. 箭头所指位置即为 T-Junction，其像素值需要通过插值得到。

侧节点和正下方节点的像素值可以直接获取。若相邻节点的规模是本节点的两倍，则此时形成 T-Junction。

对于 T-Junction，其像素值需要通过相邻节点的插值得到，如此也可以保证插值的连续性。

### 4. Quadtree 划分原则

双线性插值是以 Quadtree 的叶节点为基本单位的，因此我们划分 Quadtree 的首要依据是保证双线性插值的合理性。双线性插值只有在 (4) 式右侧为 0 时才合理，即必须使得需要被插值的叶节点内部不包含边界点。如果某个叶节点的规模为 1，该节点就不需要进行插值，当然可以包含边界点。

综上，我们对 Quadtree 是否还有必要进行划分的依据是：

(1) 该节点若包含边界点，则应该继续划分下去，直至不包含边界点或该节点的规模变为 1。(2) 节点划分的过程中，必须满足节点与相邻节点的规模差异不超过 2 倍，否则将形成除 T-Junction 之外的其他插值情况。

另外，我们在插值过程中，会寻找每个节点的右侧节点与正下方节点，为保证这两者都存在，我们需要人为的规定，最右侧与最下方的一排像素点为边界点，以便使得会有某个叶节点单独包含它们。

下图为某次实验中我们得到的 Quadtree 划分结果，其中每个正方形方块代表一个叶节点，每个叶节点的灰度值随机产生。从图中可以看出，在边界位置处，分布着密集的叶节点，在非边界位置，叶节点的数目明

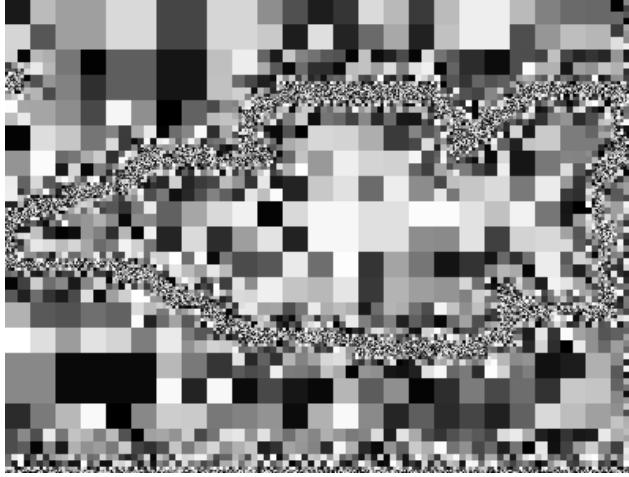


图 3. 箭头所指位置即为 T-Junction，其像素值需要通过插值得到。

显减少。但是为了保证相邻节点之间的规模之差不会超过两倍，也不会产生过大的叶节点。

如前所述，图象最右侧和最下方的一排像素都被人为设置成了边界点，以此来保证每个点在进行插值运算的时候可以找到需要的插值点，当然，如此会加大部分时间上的复杂度。

#### 4.1. 边界点的寻找

在本任务中，我们会根据一张用来标记原图像的 mask 图象来寻找融合区域的边界位置。在 mask 图象中，0 表示非融合区域，1 表示融合区域。我们需要找的边界点由由待融合区域的最外层组成，该层像素点的相邻像素中至少有一个是非融合区域的像素。为了找到这些边界点，我们对 mask 图象进行一次卷积操作，卷积核为：

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (9)$$

卷积后，若某点的值小于 9 则说明该点在边界范围内。

### 5. 降维后的泊松方程

在泊松方程的求解过程中，方程的维度由融合区域像素个数来决定。对于降维后的泊松方程，方程的维度由 Quadtree 的叶节点个数来决定。



(a) 用于融合的前景与背景

方程右侧的  $b$  向量，在未降维度的时候是由融合区域像素之间的差异来决定的，在降维后的情况下，向量由融合区域内叶节点之间像素的差异来决定。

也正是由于此处降低了解方程的维度，最终解方程的时候才会大大缩短时间，从根本上讲缩短了进行图象融合的时间。

设叶节点的数目为  $m$ ，则时间和空间的复杂度会降低到  $O(m)$ ，通常情况下， $m = O(\sqrt{n})$ 。

解出方程之后，即可得到各个节点的像素值。再根据各个点的像素值，来通过双线性插值来解出最后图象中的像素值。

降维后的泊松方程求解即为法方程的求解：

$$S^T A^T A S S_\delta = S^T A^T (b - ASy_0) \quad (10)$$

双线性插值的过程也可以表述为：

$$x = Sy \quad (11)$$

以上图为例，在降为前泊松融合的维度为 44792，降为后仅为 4732。



(b) 原图像



(c) 目标图象

图 4. 原图象和目标图象

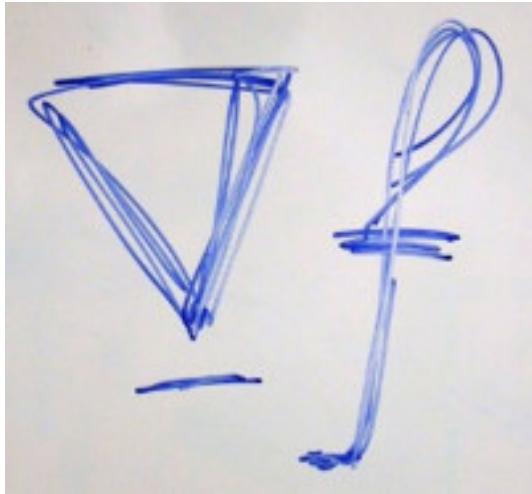


(a) 未降维泊松融合的结果



(b) 降维后泊松融合的结果

图 5. 降维和未降维后得到的结果



(a) 原图像



(b) 目标图象

图 6. 原图象和目标图象



(a) 未降维泊松融合的结果



(b) 降维后泊松融合的结果

图 7. 降维和未降维后得到的结果



(a) 原图像



(b) 目标图象

图 8. 原图象和目标图象



(a) 未降维泊松融合的结果



(b) 降维后泊松融合的结果

图 9. 降维和未降维后得到的结果

## 5. 降维后的泊松方程

7



图 10. 原图象和目标图象

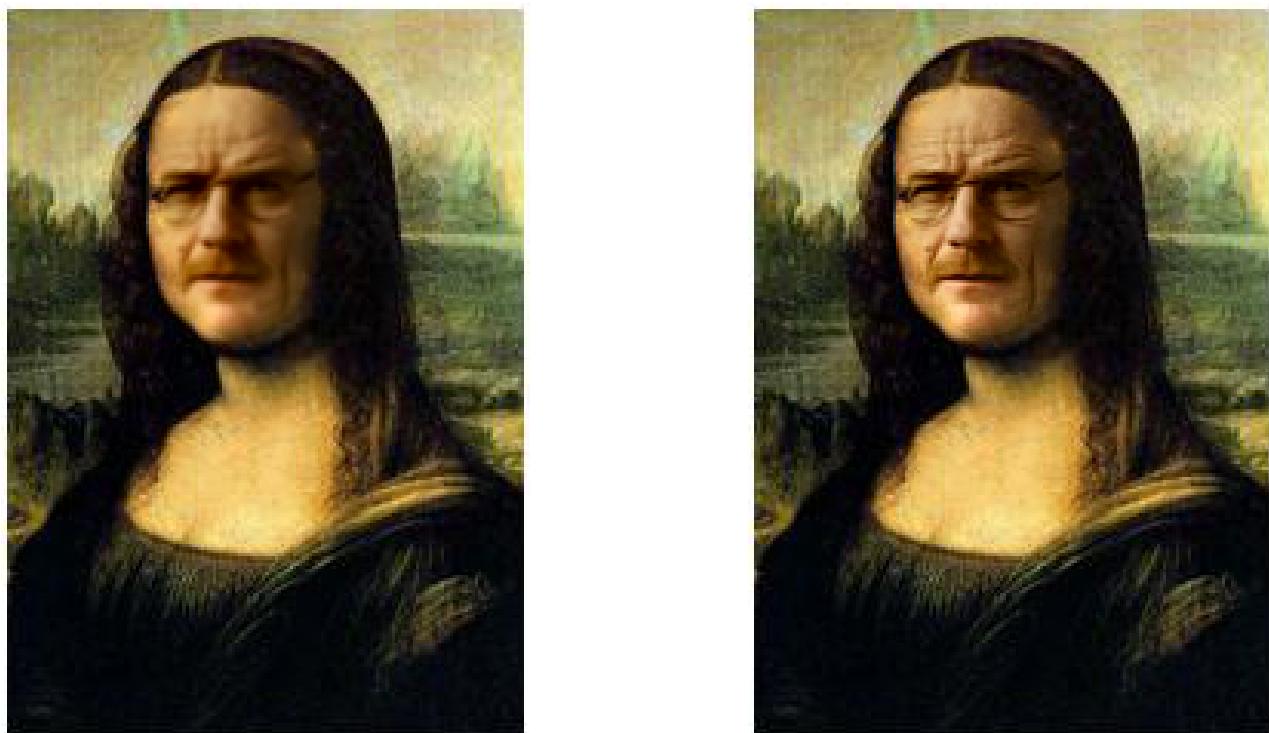


图 11. 降维和未降维后得到的结果