计算机图形学研究进展
期末程序实验报告

高思远

数学科学学院

3120103782

rickygao213@gmail.com

『实验背景』

在图像融合领域，基于梯度场的对于图像的无缝融合是一种非常有效且易于实现的融合方法，在 SIGGRAPH 上也有不少文章是基于梯度场进行图像合成的研究。其中**《Poisson Image Editing》**这篇文章是其中最为经典的文章之一，被引用次数也达到了 1000+，它的思路基于了这样的事实：人眼对于图像二阶梯度的大小，即一阶梯度的变化幅度最为敏感，所以如果可以最小化图像的二阶梯度且在 source image 和 destination image 的边界上像素值达到一致，就可以实现图像的无缝融合，而完成这个任务我们需要解决的就是求解一个 Poisson Equation。

而在这篇文章在 SIGGRAPH 2003 上发表之后，有了许多基于梯度场的图像克隆研究，而在这其中，SIGGRAPH 2008 上的一篇**《Coordinates for Instant Image Cloning》**引起了我的兴趣，因为虽然它与《Poisson Image Editing》实现基于的高层原理相同，即在边界条件下最小化二阶梯度，但是它并没有使用求解 Poisson Equation 这个方法来实现目的，因为求解方程不仅费时而且实现起来费力。它基于《Mean-Value Coordinates (MVC)》 [Floater 2003]这篇文章提出的中值坐标(Mean Value Coordinate)系统，对两幅图像的内部像素值偏差基于边界像素值偏差进行插值，从而实现更加方便的插值，因为实际上求解 Poisson Equation 实现的也是一个 Membrane Interpolation，所以两者最后效果并无太大差别，但是后者却以其更小的计算量、更小的实现代价以及更加方便的并行实现体现了自己独特的优势。

这次课程项目，我基于了《Poisson Image Editing》和《Coordinates for Instant Image Cloning》这两篇文章，实现了其中图像克隆、局部亮度调整、纹理平坦化的算法，并进行了一些自己的尝试比如加入根据人脸五官调整图片大小的功能和一些并行处理，并对这两种方法进行了比较。

『运行环境』

操作系统:

Windows 8.1 Professional

软件版本：

Microsoft Visual Studio 2013

OpenCV 2.4.10

CGAL 4.6

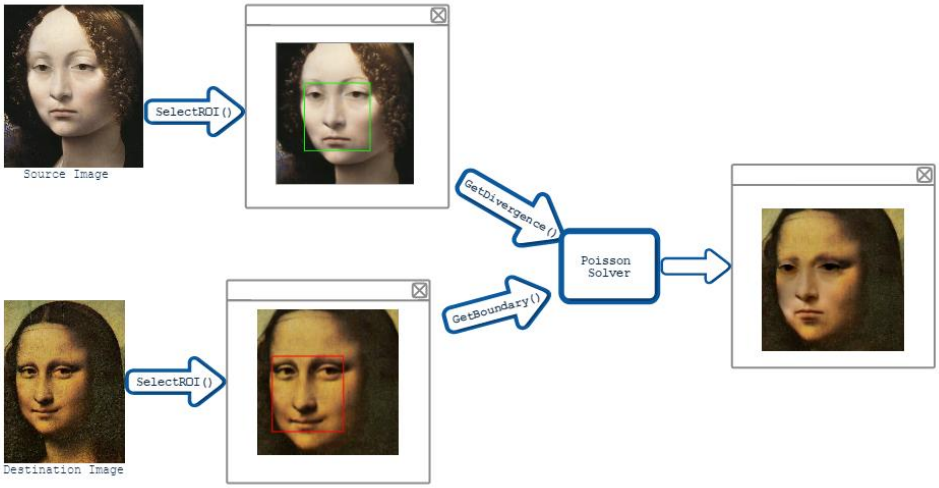
OpenMP

开发语言：

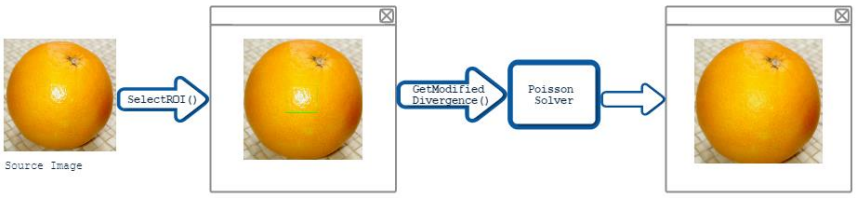
C++

『系统整体流程』

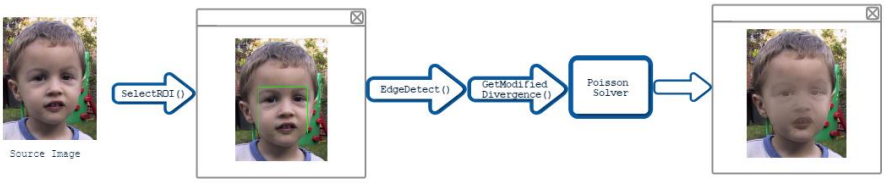
1. Poisson Image Clone



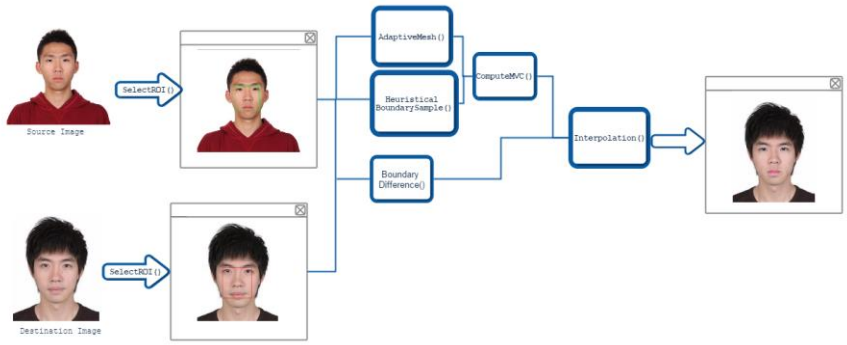
2. Poisson Local Illumination Change



3. Poisson Texture Flattening



4. Mean Value Coordinate Clone with Optimization



『具体实现』

1. Poisson Image Editing

计算图像每个像素的二阶梯度只需要将相邻的四个像素进行加减。而这里的主要任务是解相应的 Discrete Poisson Equation，这也是决定实现的程序的速度的决定性因素。在这里，我参考了 UC Berkeley 的 CS267 课程使用 Fast Fourier Transformation 方法解这个方程。

a) Fast Fourier Transformation

在这里先复习一下 Discrete Fourier Transformation 的定义。对于一个 $\mathrm{m}\times 1$ 的向量 v 的 Discrete Fourier Transformation，就是指一个矩阵向量乘法 $F\times v$ ，这里

$$F(j,k)=\Omega^{j\times k} \quad , \quad \Omega=e^{2\times\pi\times i/m}=\cos(2\times\pi/m)+i\times\sin(2\times\pi/m)$$

b) 利用 FFT 求解 Discrete Poisson Equation

在这里，我们的 Poisson Equation 可以表达为

$T\times U+U\times T=B$，其中 U 为图像矩阵， B 为边界条件，而

$$\mathrm{T}=\begin{vmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{vmatrix}$$

而经过对 T 的分解和变形，我们可以得到方程的等价形式

$\lambda\times\tilde{\mathrm{U}}+\tilde{\mathrm{U}}\times\lambda=\tilde{B}$ ，其中 λ 为对角线矩阵，而 $\tilde{\mathrm{U}}$ 和 $\tilde{\mathrm{B}}$ 则是 U和B 经过变换后的矩阵，而且可以在求解出前者后，可以比较轻松地经过变换得到后者。而求解这个等价方程组是比较简单的因为这里 λ 为对角线矩阵。所以，主要的运算量就到了如何通过 $\tilde{\mathrm{U}}$ 和 $\tilde{\mathrm{B}}$ 通过矩阵乘法变换出 U和B。而事实上，通过之前对于 T 的分解，我们只需要计算

$U = Q \times \tilde{U} \times Q, B = Q \times \tilde{B} \times Q$ ，其中 Q 由 T 的特征向量组成。

而经过计算，恰巧可得

$Q(j,k) = sin\left((j+1)\times(k+1)\times\pi/(n+1)\right)\times\sqrt{\left(2/(n+1)\right)} = Q(k,j)$,则

$Q \times v = image((F \times vv)(1:n)),$ 其中 $vv = [0;\ v;\ zeros(n,1)]$ ，故 $Q \times v$ 可由对于 $\tilde{U}$ 和 $\tilde{B}$ 的 FFT 的结果得到。而由于 FFT 的时间复杂度为 $O(mlogm)$，这样就大幅度削减了求解这个 Discrete Poisson Equation 的开销。

c) Mixed Gradient

在计算 source image 的梯度场时，我们没有考虑 destination image，这样会使得在合成时图像边缘出现 source image 的背景杂色，所以为了解决这个问题，我们把 destination image 的信息也考虑进去，改变要求解图像的梯度为

$$\text{for all } \mathbf{x} \in \Omega,\ \mathbf{v}(\mathbf{x}) = \begin{cases} \nabla f^*(\mathbf{x}) & \text{if } |\nabla f^*(\mathbf{x})| > |\nabla g(\mathbf{x})|, \\ \nabla g(\mathbf{x}) & \text{otherwise.} \end{cases}$$

这样的效果就是，如果 destination image 在某个区域的梯度大于 source image，也就是说这里的图像变化更加明显，包含了更多信息，那么就取 destination image 的梯度，这样在合成一些带孔图片时，就可以实现更加无缝的拼接，具体的区别可以看之后的结果对比部分。

d) Poisson Local Illumination Change

有了"梯度场引导、边界条件限制"这个思路，我们还可以实现许多其它的功能。比如通过 Fattal et al. transformation [Fattal et al. 2002]对于图像梯度场梯度大小的调整，我们可以实现对图片的局部光亮调整，具体的变换公式如下：

$$\mathbf{v} = \alpha^{\beta} |\nabla f^*|^{-\beta} \nabla f^*$$

可以看出，通过这个非线性变换，梯度场的梯度大小有了一个基于 β 和 α 的变换，而 β 越大这个梯度缩小效果也越明显，具体区别也可以在后面的结果对比部分体现出来。

e) Poisson Texture Flattening

另一个利用梯度场思路对图像进行调整的应用就是只取边界处的图像梯度，从而主要保留边界部分的图像变化信息，这样可以舍弃一些不在边界上的图像变化，从而使整个图像变得更加平坦，起到一个很独特的效果，具体的变换公式如下：

$$v_{pq} = \begin{cases} f_p - f_q & \text{if an edge lies between } p \text{ and } q, \\ 0 & \text{otherwise,} \end{cases}$$

2. Mean Value Coordinate Clone

观察求解 Poisson Equation 得到的图像像素偏差柱状图图 1，直观理解可以认为解这个方程是为了将两个图像在边界部分的偏差在图像内部均匀扩散开来，就像一张薄膜一样，所

以这也叫 Membrane Interpolation(薄膜插值)，而这个插值的特点就是内部的二阶导数为 0。因此 Zeev Farbman 等人就看到了实现无缝合成的另一种思路，一种不同于求解方程的思路。参考《Mean-Value Coordinates (MVC)》这篇文章提出的 Mean-Value Coordinates(中值坐标系统)，它们提出了通过求解图像内部每个点关于图像边缘的中值坐标，从而直接对内部进行加权平均的思路。

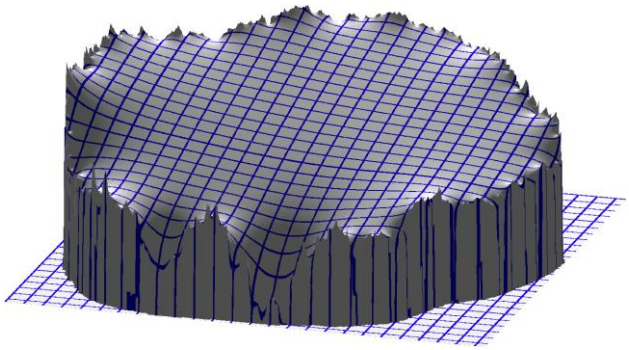


图 1

a) Mean Value Coordinates

先介绍一下中值坐标的具体数学表达式。以 Mean-Value Theorem 为基础，对于一个多边形内部的每个点，定义

$$\lambda_i(\mathbf{x}) = \frac{w_i}{\sum_{j=0}^{m-1} w_j}, \quad i = 0, \ldots, m-1,$$

,其中

$$w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|\mathbf{p}_i - \mathbf{x}\|},$$

对于每一个固定的 $\mathbf{x}$，$\lambda_i(x)$ 就是每个边界点对于它的中值坐标，也就是在插值时的归一化权重。而根据中值定理，这个坐标可以保证插值时的均匀性。

b) Clone based on MVC

因为中值坐标的特点就是实现一个均匀的插值，所以这个算法的效果非常不错，基本可以达到和求解 Poisson Equation 实现的图像克隆一样的效果，具体可见后面的结果对比。而这个算法的实现也比较简单，大体框架如下

Algorithm 1 MVC Seamless Cloning

{Preprocessing stage}
for each pixel $\mathbf{x} \in P_s$ **do**
  {Compute the mean-value coordinates of $\mathbf{x}$ w.r.t. ∂P_s}
  $\lambda_0(\mathbf{x}), \ldots, \lambda_{m-1}(\mathbf{x}) = MVC(x, y, \partial P_s)$
end for
for each new P_t **do**
  {Compute the differences along the boundary}
  for each vertex $\mathbf{p}_i$ of ∂P_t **do**
    $diff_i = f^*(\mathbf{p_i}) - g(\mathbf{p}_i)$
  end for
  for each pixel $\mathbf{x} \in P_t$ **do**
    {Evaluate the mean-value interpolant at $\mathbf{x}$}
    $r(\mathbf{x}) = \sum_{i=0}^{m-1} \lambda_i(\mathbf{x}) \cdot diff_i$
    $f(\mathbf{x}) = g(\mathbf{x}) + r(\mathbf{x})$
  end for
end for

虽然这个算法的效果非常好，而且实现简单，但是还是这样也基本意味着一个很大的时间开销，虽然不用求解方程，但是由于对于图像内部的 n 个点每个都要计算 m 次乘法，其中 m 为图像边界像素数目，这个时间复杂度随着图像大小变大会轻易变得无法忍受，更不用说实现工业级的用户交互速度。于是我们要对这个算法进行必要的优化，使得即使不用在 GPU 上进行运算也可以实现实时用户交互体验。

c) Optimization

i. Hierarchical Boundary Sampling.

这个优化是基于这样的观察:对于一个图像，在中值坐标系统下对于它影响比较大的边界点基本都分布在这个点的周围，离它比较远的点对它的影响虽然有，但是寥寥无几。所以我们可以在计算中值坐标时直接舍弃掉一些对它影响比较小的点，让取点更加稀疏并接近一个常数(大约为 16-32)，这样通过必要的预计算，就可以大幅度削减在实际图像克隆时的开支，而实际实现后，我发现只要通过这一个优化就基本可以超越 Poisson Clone 的克隆速度，而且不用并行也可以基本达到用户实时交互的要求。

实现. 这个算法的实现也并不太复杂，首先以一定固定间隔对边界点取大约 16 个点作为初始等级的边界，然后对每个点不断进行迭代，进行 *boundary refinement*，迭代时的条件如下，对每个第 k 等级的边界，假设 $\mathrm{p}_{\mathrm{i-s}}^{k}, \mathrm{p}_{\mathrm{i}}^{k}, \mathrm{p}_{\mathrm{i+s}}^{k}$ 是三个连续的边界点，如果满足下列条件

$$\begin{aligned} \|\mathbf{x} - \mathbf{p}_i^k\| &> \varepsilon_{dist} \\ \sphericalangle \mathbf{p}_{i-s}^k, \mathbf{x}, \mathbf{p}_i^k &< \varepsilon_{ang} \\ \sphericalangle \mathbf{p}_i^k, \mathbf{x}, \mathbf{p}_{i+s}^k &< \varepsilon_{ang} \end{aligned}$$

其中，

$$\varepsilon_{dist} = \frac{\#\text{ boundary pixels}}{16 \cdot 2.5^k} \quad \text{and} \quad \varepsilon_{ang} = 0.75 \cdot 0.8^k,$$

那么就不用再对这三个边界点进行 *boundary refinement*，否则的话，就在这三个点中间再添加两个边界点，从而让这里有更多边界点参与插值，说明这个边界点附近有更多的图像差值信息。当所有边界点都满足这个条件之后，我们的 *boundary refinement* 就完成了。通常，对于一副 600*400 的图片，只需要 30+个边界点就可以实现一个非常不错的图像融合。

ii. Adaptive Mesh

这个优化的思路是基于这样的观察：对于两幅图像内部的像素偏差，在比较靠近中间的部分其实是很均匀的，只是在靠近边界的部分变化比较剧烈。所以我们可以对图像内部首先进行一个非均匀的撒点，在靠近中间部分撒点数目少些，靠近边界部分撒点数目多一点，然后进行一个 *Delaunay Triangulation*，得到一个 *Triangulation Mesh*，然后只计算这个 Mesh 里每个顶点的像素偏差插值，而对于三角形内部的像素点，查找它所在的三角形，然后只通过这个三角形的三个顶点的像素差值进行一个双线性插值即可。这样，我们就省去了之前对图像内部每个顶点都计算中值坐标以及插值的时间开销。

优化后的图片如图 2 所示，边界上的红色点代表 Hierarchical Boundary，内部的三角形网格代表 Adaptive Mesh。可以看出，经过这两个优化，整个算法的时间复杂度由原来的 $O(n^3)$ 变为现在的 $O(n^2)$，其中 n 为图像边长，而且时间增长速度会显著变慢。

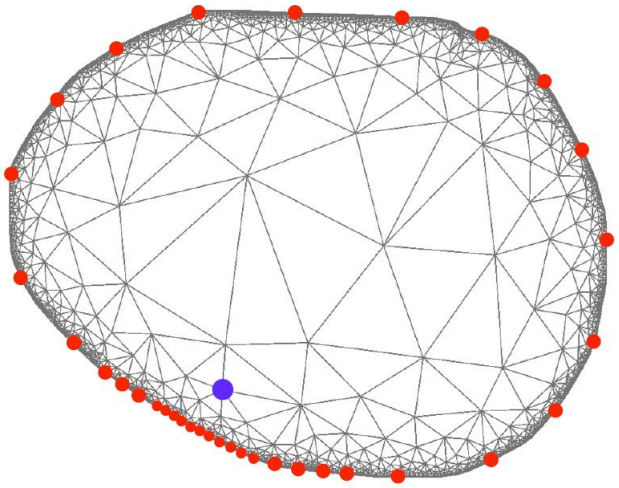


图 2

『结果对比』

1. 两种方法的对比

首先我对于两种不同的图像合成方法进行了对比。

a) 5500 个点

当要把一个大约有 5500 个点的鸟图 3 合成到一个天空上时，我对使用的方法进行了合成效果和时间开销的对比。

图 3

图 4





图 5 第一排从左到右依次是 Poisson Clone, Mixed Poisson Clone, Naïve MVC 的结果，第二排从左到右分别是 Hierarchic MVC, Hierarchic MVC with Triangulation 的结果

Method	Poisson Clone	Mixed Poisson Clone	Naïve MVC	Hierarchic MVC	Hierarchic MVC with Triangulation
Preparation time	\	\	1s	2.2s	4.2s
Execution time	0.13s	0.14s	1.2s->0.5s	0.16s	0.03s

表格 1

b) 38760 个点

当 source image 图像的内部点增加到 38760 个时

图 6

图 7

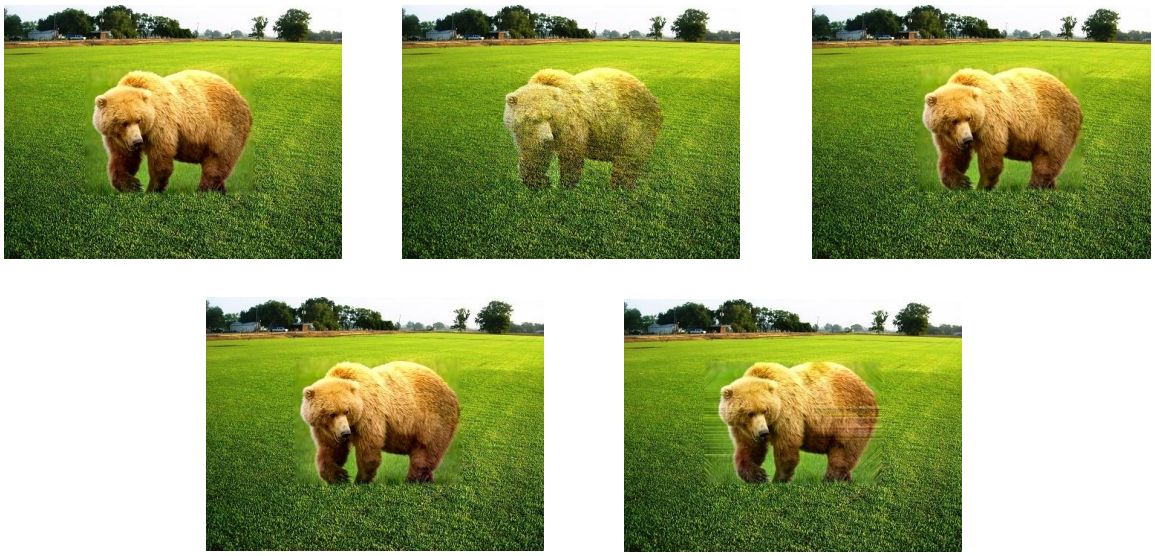


图 8 第一排从左到右依次是 Poisson Clone, Mixed Poisson Clone, Naïve MVC 的结果，第二排从左到右分别是 Hierarchic MVC, Hierarchic MVC with Triangulation 的结果

Method	Poisson Clone	Mixed Poisson Clone	Naïve MVC	Hierarchic MVC	Hierarchic MVC with Triangulation
Preparation time	\	\	19s	16s	55s
Execution time	1.1-1.2s	1.1-1.2s	21s->9s	1-1.1s->0.5s	0.13s
Time complexity	$O(\sqrt{n}\log n)$	$O(\sqrt{n}\log n)$	$O(nm)$	$O(n)$	$O(n)\downarrow$

表格 2 →代表并行后的结果

c) 总结

效果. 通过这两组实验结果，先从效果来讲，我们可以看出 Poisson Clone、Naïve MVC、Hierarchic MVC 的克隆效果是十分相似的，但是由于被合成的图像带有自己的背景，所

以他们都或多或少在边缘区域有一些模糊，这个问题一方面可以通过更加精确的选择图像边缘，比如结合使用 grabcut 进行选择，另一方面在本次实验中，Mixed Poisson Clone 的效果比较好地解决了这个问题，使得边界部分的色块模糊问题有了比较大的改善。当然，这也有问题比如物体本身有可能“融入”背景中，比如图 8 中的熊就被“融入”到了草坪中，这也是这个算法本身的问题，如果要改进的话，可以在实际操作中设置阈值，使得当 source image 梯度场梯度大于某个值的话，梯度不会被 destination image 影响。但由于这个阈值要随不同场合分别确定，会增加用户的软件学习成本，所以这里我并没有将这个功能融入进来。而最后，要提下使用了 Adaptive Mesh 方法的结果，我们可以看出在图 8 的熊的边缘出现了一些棱状杂色，这也是因为在优化时的取点不够合理导致，但是在请教助教以及侯启明老师后，我还是没能找到一个能够对于所有的图像都适用的取点方法，所以我并没有把这个优化加入到我最终的程序中，而是作为了一个单独的程序列了出来。

速度. 既然这两种方法在效果上类似，我们就从速度上对它们进行一下比较。首先是 Poisson Clone。通过 FFT 来求解 Discrete Poisson Equation, 虽然我们求解了一个较大规模的稀疏系数矩阵，但是速度还是比较可观的，对于一个 600*600 的图像，1s 左右的合成时间也是可以接受的。而对于 MVC Clone，通过使用 Hierarchic Boundary Sampling 和并行处理， 我们的合成速度只需要 0.5s 左右。而使用了两种优化方法后，我们的合成速度只需要 0.13s，并行处理后可以接近 0.05s，这样的速度也可以让它将来应用到视频合成中。当然，速度不止包括运行速度，也包括程序实现所需的时间，就这个方面来讲，MVC Clone 也因为省去了求解 Poisson Equation 的步骤所以在实现速度上略胜一筹，当然，进行 Delaunay Triangulation 时我使用的是现成的 CGAL 库中的函数，所以这也让我的实现简单了些。所以，在速度方面来说，MVC Clone 方法因为它易于并行和易于实现以及更直接的思路，要比 Poisson Clone 表现更好。

可扩展性. 因为 Poisson Clone 考虑了整个克隆区域的梯度场，而 MVC Clone 主要考虑的是边界像素差值，对于内部的梯度场实际上并没有进行求解，所以 Poisson Clone 可以被扩展出更多的应用比如局部光亮调整和纹理平坦化，所以从这个方面来讲，MVC Clone 要略逊一些，虽然通过一些调整，它也可以实现这个功能，但是它本身的特色就不能被很好体现出来。

2. 更多结果

由于两种方法在效果上及其类似，这里我就以 Poisson Clone 的结果为主来进行更多的结果展示以及与原论文的效果对比。

首先是与原论文合成效果对比，由于原论文并没有具体的合成所需时间，所以这里对于合成速度就不进行比较了。

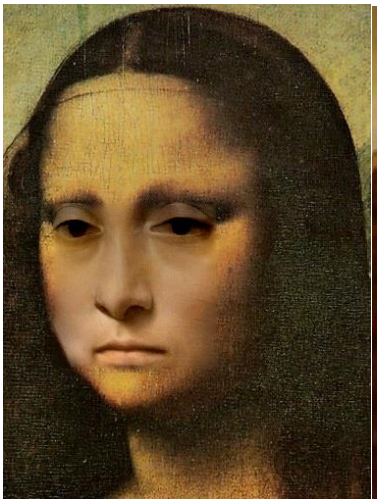


图 9


图 10

图 9 和图 10 分别是我的实现效果和论文的实现效果，由于论文中采用的是多边形边界选择克隆区域，而我采用的是方框选择，所以我无法仅仅选出五官，必然会有些部位影响克隆，所以肤色的融合并不太完美，但是通过后面的结果可是看出对于一般的正面照片，这些照片仅仅使用方框就可以选出五官，我的合成效果是很和谐的。


图 11


图 12

图 11 和图 12 分别是我和论文对于 Mixed Gradient Clone 的实现效果，可以看出基本是没有区别的。

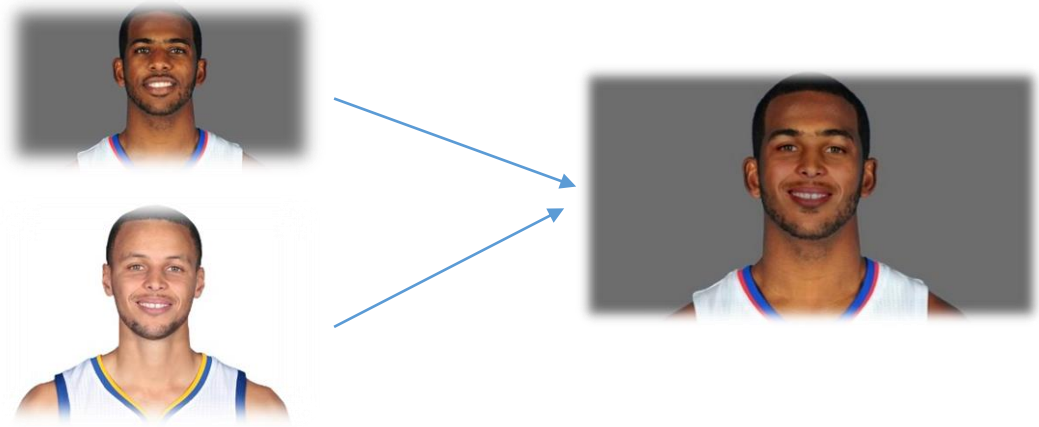
除了与论文的对比，合成到底和不和谐其实很多时候也是比较主观的。于是我做了一个比较有意思的“第三方测试”。我用自己（图 13）和明星俞灏明（图 14）的证件照进行了合成，并放在了微信朋友圈（图 15）上，在总共 40 多个回复中，我的朋友们并没有看出来这是一张处理的照片，所以可以通过这个小测试说明合成效果还是不错的。

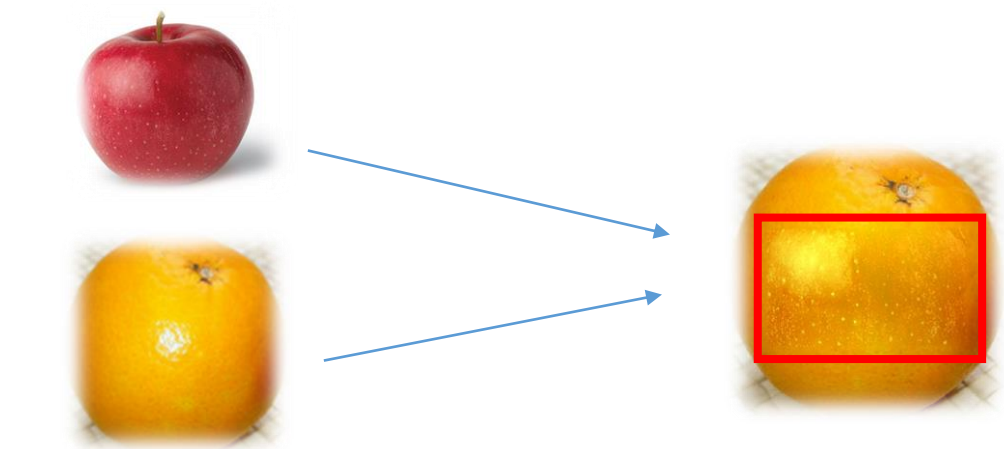
图 13

图 14

图 15

以下是其它合成结果：





3. 局部光亮调整与纹理平坦化

除了进行图像克隆，这里我再对 Poisson Image Editing 的其他应用结果进行比较。

a) Poisson Local Illumination Change

图 16 从左到右依次是原图、我的结果、论文中的结果

从图 16 可以看出我的实验结果和原文中结果基本相同，对于局部光照的调整非常和谐，值得一提的是，对于 β，我的实践中 β=0.4 要比文章中 β=0.2 效果更好。

b) Poisson Texture Flattening

图 17 从左到右依次是原图、我的结果、论文中的结果

图 17 可以看出我的结果和论文中的结果有些差距，这主要还是因为我使用的是方框选择区域而原文使用的是多边形选择，可以更好地选择出小孩的整个面部，而我选择的主要是小孩眉毛下方到下嘴唇上方的区域，所以结果不够和谐。

『未来改进』

由于时间有限且做的内容也比较多，所以我的项目还有一些不足之处可以在后续改进。

- 边界选择改为多边形区域选择
- 进行 Adaptive Mesh 时图像内部的撒点算法进行改进
- 进行更多的并行处理，包括 FFT 求解 Poisson Equation
- GPU 上的实现
- 在预处理过程也进行并行处理
- 自己写 Delaunay Triangulation，因为 CGAL 库中在查找像素点所在三角形时效率很低

『感想』

作为一个数学系的学生，这次的大作业无疑对我来说是一个很大的考验，甚至可以说是大学以来经历的最大考验。从一个从未接触过图形学和 OpenCV 的“小白”到可以独自实现两篇论文，这其中我收获了太多太多。

不同于计算机系或者数媒的同学，无论对于编程还是数学，我都缺少一些把这些知识应用到实践上的意识，很多时候，学了一个知识，却不知道怎么用它，这是一个很大的问题，尤其对于天天接触的是比较枯燥的数学知识的数学系学生。但是，通过这次课程，起码，我看到了学科之间是如何交叉应用的，看到了我在平时学到的知识可以怎样地被应用到看似不相关的事物当中。

我学过的编程课也并不太多，C++、C#之前也只是能够实现一个矩阵计算器这样的水平，但是，通过这次大程，我第一次写了超过千行的代码，让我更加体会到之前学习的编程知识以及良好的编程习惯对于开发大项目的意义所在。

总之，这次的大程应该是我在大学迄今为止做的最不“水”的一个项目，也是最有收获的一个项目。

『参考文献』

BerkeleyCS267. (1996). Solving the Discrete Poisson Equation using Jacobi, SOR, Conjugate Gradients, and the FFT.

FarbmanZeev. (2008). Coordinates for Instant Image Cloning.

P´erezPatrick. (2003). Poisson Image Editing.