《基于泊松方程的图像融合》

实验报告

计 23 鲁逸沁 2012011314

【实验内容】

使用泊松编辑（Poisson Editing）的方法，将源图像的部分加入到目标图像中，使得在目标图像中，源图像的那部分能很好的融合其中，没有明显的不和谐现象。

泊松编辑的关键在于利用泊松方程的形式描述嵌入的图像块，以及目标背景，在该图像块边界出的梯度，使得图像块在边界处保持梯度的连续，形成融合的效果。

下面的图就是一个图像融合的例子，将其他图片的人和狗放入目标图的水中。直接放入会导致图像的失真（如左图），经过图像融合后的效果就更接近真实（如右图）。

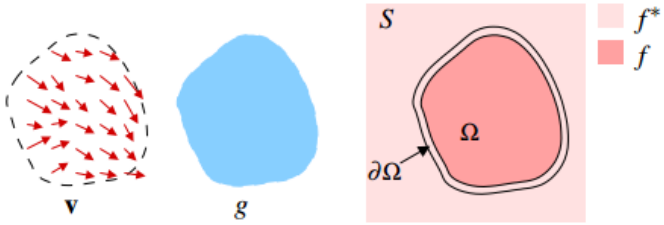
【实验配置】

- 实验语言：C++
- 实验平台：Visual Studio 11
- 实验使用工具：OpenCV-2.4.6.0
- 实验数据：网络图片

【算法描述】

✧ 泊松方程

假设目标图像是S，源图像是区域Ω，它们都是$\mathbb{R}^2$中的闭区域。假设区域Ω在S中的的边界为$\partial\Omega$，如下图所示。



假设f^*是S的像素函数减去原区域Ω内部像素函数的结果，f为Ω区域内目前未知的像素方程。$\vec{\mathrm{v}}$为Ω上的向量场。

要做到融合的效果，就需要区域内外在边界处梯度一致，以保证不存在跳变的情况。则这个问题的解，就是下面这个最优化问题的解

$$\begin{cases}\min\limits_{f}\iint_{\Omega}|\nabla f-\vec{\mathrm{v}}|^2\\ f|_{\partial\Omega}=f^*|_{\partial\Omega}\end{cases}\tag{1}$$

其中∇.是梯度算子。这个问题的最优解泊松方程在狄利克雷边界条件下的解

$$\begin{cases}\Delta f=\operatorname{div}\vec{\mathrm{v}}\\ f|_{\partial\Omega}=f^*|_{\partial\Omega}\end{cases}\tag{2}$$

其中$\Delta.=\frac{\partial^2.}{\partial x^2}+\frac{\partial^2.}{\partial y^2}$是拉普拉斯算子，$\operatorname{div}\vec{\mathrm{v}}=\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}$是$\vec{\mathrm{v}}=(u,v)$的散度。

在离散的情况下，假设N_p是像素p的四连通相邻像素，$\langle p,q\rangle$是两个相邻像素对，令f在p处的函数值为f_p，则 (1) (2) 式可改写成

$$\begin{cases}\min\limits_{f|_{\Omega}}\sum_{\langle p,q\rangle\cap\Omega\neq\emptyset}\left(f_p-f_q-v_{pq}\right)^2\\ f_p=f_p^* \qquad\qquad p\in\partial\Omega\end{cases}\tag{3}$$

其中$v_{pq}=\vec{\mathrm{v}}\left(\frac{p+q}{2}\right)\overrightarrow{pq}$，可以理解为原像素函数之差。而 (3) 式中问题满足下面的线性方程组

$$|N_p|f_p-\sum_{q\in N_p\cap\Omega}f_q=\sum_{q\in N_p\cap\partial\Omega}f_q^*+\sum_{q\in N_p}v_{pq}\qquad\forall p\in\Omega$$

这个方程组直接求解将是 $O(N^3)$左右的复杂度，太高不能接受。但观察发现其系数矩阵是一个稀疏矩阵，每行非零系数最多为 5 个，又是对角优势矩阵，因此可以通过高斯-赛德尔迭代法来快速求得其近似解。

✧ 高斯-赛德尔迭代法

假设有方程组$\mathrm{Ax}=\mathrm{b}$，其中$\mathrm{A}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$是对角优势矩阵，即对于$\forall i$有$a_{ii} > \sum_{\substack{1\le j\le n \\ j\neq i}} a_{ij}$。$\mathrm{b}=[b_1 \quad b_2 \quad \cdots \quad b_n]^{\mathrm{T}}$是方程的右部。则该方程的解x可用下面的方式迭代求解：

$$\mathrm{x}^{(0)}=\left[x_1^{(0)} \quad x_2^{(0)} \quad \cdots \quad x_n^{(0)}\right]^{\mathrm{T}}$$

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{\substack{1\le j\le n \\ j\neq i}} a_{ij}\cdot x_j^{(k)}\right)$$

其中$x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$为任意的初始值。当$k$足够大时，$\mathrm{x}^{(k)}$就会足够接近最优解了。

这个算法的原理就是利用之前得到的解作为条件，不断求出新的解，新的解一般比原来的解更接近正确解，则经过多轮迭代后，解会无限逼近正确解。

当然，可以用下面的方式迭代，在求解的过程中直接用了新解中先求出的答案，收敛的速度更快：

$$\mathrm{x}^{(0)}=\left[x_1^{(0)} \quad x_2^{(0)} \quad \cdots \quad x_n^{(0)}\right]^{\mathrm{T}}$$

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j<i} a_{ij}\cdot x_j^{(k+1)}-\sum_{j>i} a_{ij}\cdot x_j^{(k)}\right)$$

另外，从我实验的结果来看，由于不能保证我们求得的方程组具有唯一解，因此不同的初始值会导致最终的解不同，从而影响最后的融合效果。

✧ 比例融合方式

将源图和目标图按比例融合，有两种方式：

- 融合块中的梯度只考虑源图的梯度，求解出最优解后，再将最优解和目标图的对应位置像素值按比例融合；
- 融合块中的梯度按比例考虑源图的梯度和目标图的梯度，求出的最优解直接成为对应位置的像素值。

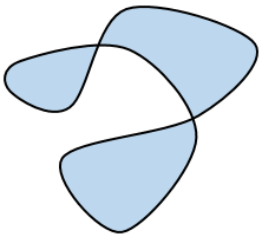
两种方式各有各的特点，在结果分析处会详细讨论。

✧ 融合块选择算法

在 Demo 中，程序可以要求用户画出随意的图形，将画出的轨迹围起来的部分作为待融合的块。

第一个问题，程序可以检测用户的鼠标操作，但是这样的检测并不是很高效，在鼠

标移动足够快的时候，程序中检测鼠标移动并返回其位置的函数中，鼠标相邻两次移动的坐标并不会相邻，甚至还能相差很远。如果仅仅用鼠标移动返回的位置，是构造不了一个封闭图形的。包括鼠标移动的起始点和终止点之间也不是相邻的。这就需要我们根据两个点的信息，画出一条封闭的直线，将鼠标的轨迹模拟出来。这里我使用的方法是图形学课中学习的 DDA 算法。这样就能通过鼠标移动，构造出一条封闭的曲线，来描述用户选择的区域。



第二个问题，如何根据画出的轨迹确定用户围出的区域。如上图所示，用户会画出奇奇怪怪的封闭曲线，这是仅仅使用线型扫描的方法进行区域识别是无法做到的。我使用的方法是对每个像素点进行 Flood Fill 算法进行扩展，如果某个像素点能找到一条不和用户轨迹相交的，且能通到图片外部的路径，则认定这个像素点在所选区域外，否则就在所选区域内。同时如果再给每个点加上已 Flood Fill 过的标记，就能让整体复杂度和图片像素点个数同阶。

【Demo 说明】

开启程序以后，程序会弹出文件选择框，提示用户分别导入源图和目标图。

在源图的窗口上，用户可以使用左键拖拽进行选择要融合的范围。程序会自动地连接鼠标轨迹的首尾以保证轨迹封闭，如下图所示。

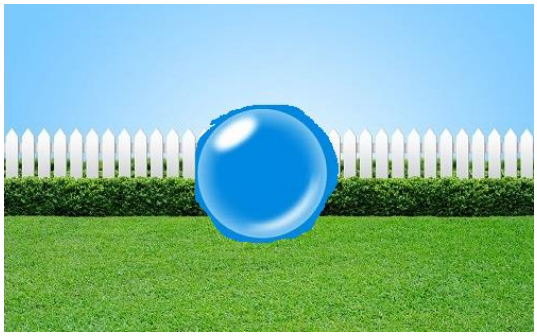
可以在源图上使用右键清空轨迹，进行轨迹重绘。

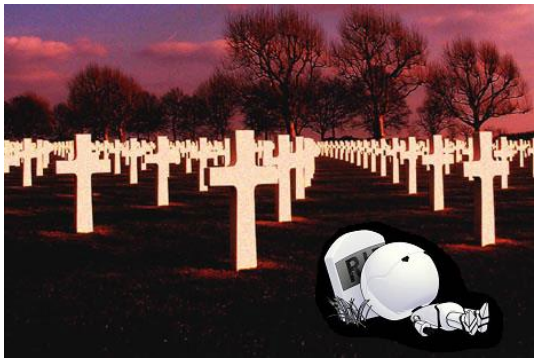
选定好需要融合的区域后，可以在目标图上点击左键选择融合的位置。这个位置将源图中选择区域的“中心”重合上去。这里的“中心”是指最小外接长方形的中心。

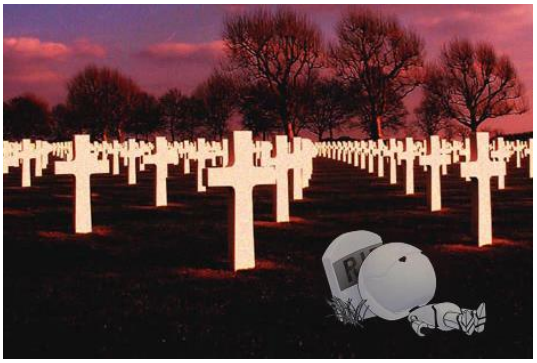
如果区域重合上去后超过目标图的边界，则程序会提示越界，并要求你重新选择融合的位置。

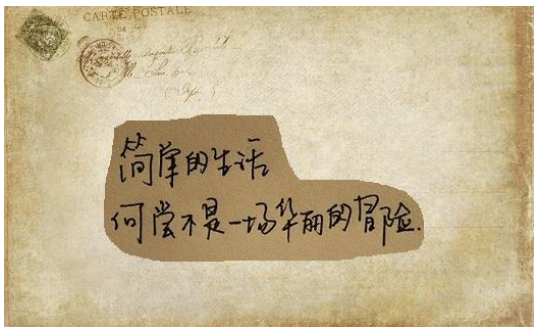
如果选择合法，则等待程序运算完毕后，源图所选取区域会变色，同时在目标图的相应位置会显示出融合后的结果。

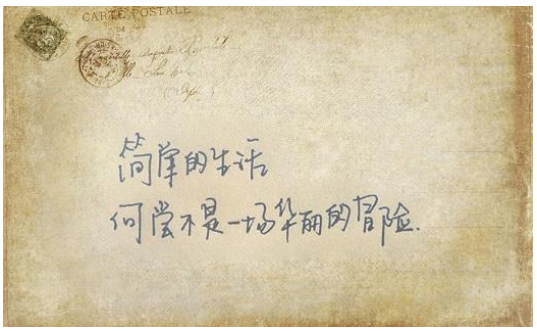
【实验结果】








简单的生活
何尝不是一场华丽的冒险.


简单的生活
何尝不是一场华丽的冒险.

【结果分析】

✧ 比例融合方式讨论

之前提到有两种比例融合方式：

- 融合块中的梯度只考虑源图的梯度，合并时按比例
- 融合块中的梯度按比例考虑源图的梯度和目标图的梯度

从直观上看，似乎第一种方式的源图和目标图在融合块中的部分的信息只是简单的透明度式融合，而第二种方式用更复杂的方式进行了融合。

下面是融合比例为 1:1 的两种方式的融合效果：

第一种方式

第二种方式

从这张图的效果上看，两种方式并没有太大的差异。但是第二种方式在一些边界和纹理的细节上处理的比第一种方式好。原因可能是第一种方式的融合太过简单，仅仅修改了透明度进行叠加而已，对于一些纹理性较强的图，前后纹理不一致，就会造成局部混淆的现象。而第二种方式，是对于融合部分的整体考虑，每个像素的梯度都考虑的全局的因素，相对来讲后期混淆的程度就会轻一些。

✧ 不同融合比例效果对比

下面是使用考虑目标图梯度的融合方式，针对同一幅图的几种不同融合比例进行对比的实验。融合系数$= x$表示源图和目标图融合的比例为$x : 1 - x$：

融合系数=0.1

融合系数=0.2

融合系数=0.3

融合系数=0.4

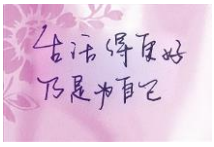
融合系数=0.5

融合系数=0.6

融合系数=0.7

融合系数=0.8

融合系数=0.9



融合系数=1.0

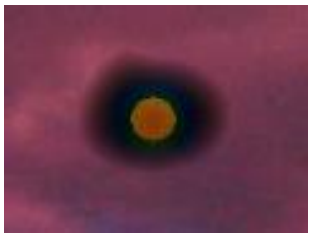
实验结果符合预期。

✧ 迭代不同初始值效果对比

高斯-赛德尔迭代法中，由于方程组并不保证存在唯一解，因此不同的初始值会收敛成不同的解，从而影响不同的效果。

下面是三种不同初始值的效果对比：

初始值为随机值



初始值为 0

初始值为 255

可以明显的发现，初始值的不同会导致结果产生巨大差异。初始值为 0 时，整个融合块的色调会接近黑色；初始值为 255 时，整个融合块的颜色会接近白色。初始值为随机值时，整个融合块的色调才会趋近正常。不过不同的初始随机值之间也会产生微小的差异。

【实验总结】

通过本次实验，我了解了泊松编辑解决图像融合问题的方法，不仅了解了理论上泊松方程的求解、稀疏矩阵的求解，也动手尝试了稀疏矩阵的求解。

本次实验中，我实现了利用泊松编辑的理论进行列方程，然后使用高斯-赛德尔迭代法进行求解。在这个基础上，我尝试了使用不同的比例融合算法，并尝试使用源图和目标图不同的比例，以寻求视觉最佳方案。同时我探究了不同融合比例的效果的对比，以及不同迭代初始值的效果的对比，寻找到了其中的一些规律和法则，尝试很多不同的图片，将很多图片两两组合观看效果，并调节一些参数使得视觉效果最优化。

另外，制作 demo 中用鼠标圈区域的算法是我自己设计的，遇到了诸如鼠标轨迹点不连续、轨迹不封闭、封闭轨迹不合法等问题，都使用相应的方法进行了解决，最终形成了可以用鼠标很方便选取任意形状区域的 demo。

同时通过本次实验，我也掌握了基本的 OpenCV 用法，并写了近 18K 的代码，锻炼了自己的代码能力，受益匪浅！

【参考文献】

[1] Pérez P, Gangnet M, Blake A. Poisson image editing[C]//ACM Transactions on Graphics (TOG). ACM, 2003, 22(3): 313-318.

[2] Gauss-Seidel Iteration Algorithm, Wikipedia.