实验三 Poisson Image Editing

ID: 58 陈文博

March 8, 2020

1 实验要求

* 实现 Poisson Image Editing 算法
* 实现多边形光栅化的 (扫描线转换算法)
* 学习使用 Eigen 库求解大型稀疏方程组
* 学习使用 OpenCV
* 实时拖动区域显示结果（Optional）
  - 矩阵预分解

2 开发环境

IDE：Microsoft Visual Studio 2019 community

CMake：3.16.3

Qt：5.14.1

Eigen：3.3.7

OpenCV：4.2.0

3 算法原理

3.1 问题描述

Figure 3.1: girl

Figure 3.2: sea

如上两幅图，现我们需要将 Figure 3.1 中的女孩搬到 Figure 3.2 的海水中，为使得复制粘贴更加逼真自然，我们需要设计算法来满足我们两幅图像融合的需要

3.2 Poisson Image Editing 算法 [1]

Poisson Image Editing 算法的基本思想是在尽可能保持原图像内部梯度的前提下，让粘贴后图像的边界值与新的背景图相同，以实现无缝粘贴的效果。从数学上讲，对于原图像 $f(x,y)$，新背景 $f^*(x,y)$ 和嵌入新背景后的新图像 $v(x,y)$，等价于解最优化问题：

$$\min_f \iint_\Omega |\nabla f - \nabla \boldsymbol{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{3.1}$$

利用变分法可转化为具有 Dirichlet 边界条件的 Poisson 方程：

$$\Delta f = \Delta \boldsymbol{v} \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{3.2}$$

以 Figure 3.1 和 Figure 3.2 为例，将 Figure 3.1 中需要复制的区域设为 S，定义 N_p 为 S 中的每一个像素 p 四个方向连接邻域，令 $< p,q >$ 为

满足 $q \in N_p$ 的像素对。边界 Ω 定义为 $\partial\Omega = \{p \in S \setminus \Omega : N_p \cap \Omega \neq \emptyset\}$，设 f_p 为 p 处的像素值 f，目标即求解像素值集 $f|_\Omega = \{f_p, p \in \Omega\}$
利用 Poisson Image Editing 算法的基本原理，上述问题转化为求解最优化问题：

$$\min_{f|_\Omega} \sum_{<p,q>\cap\Omega\neq\emptyset} (f_p - f_q - v_{pq})^2, \text{with } f_p = f_p^*, \text{for all} p \in \partial\Omega \tag{3.3}$$

化为求解线性方程组：

$$\text{for all } p \in \Omega, |N_p| f_p - \sum_{q\in N_p\cap\Omega} f_q = \sum_{q\in N_p\cap\partial\Omega} f_p^* + \sum_{q\in N_p} v_{pq} \tag{3.4}$$

对于梯度场 $\boldsymbol{v}(\boldsymbol{x})$ 的选择，文献 [1] 给出两种方法，一种是完全使用前景图像的内部梯度，即：

$$\text{for all } < p, q >, v_{pq} = g_p - g_q \tag{3.5}$$

另一种是使用混合梯度：

$$\text{for all } \boldsymbol{x} \in \Omega, \ \boldsymbol{v}(\boldsymbol{x}) = \begin{cases} \nabla f^*(\boldsymbol{x}) & \text{if } |\nabla f^*(\boldsymbol{x}) > |\nabla g(\boldsymbol{x})|, \\ \nabla g(\boldsymbol{x}) & \text{otherwise} \end{cases} \tag{3.6}$$

3.3 扫描线算法

为实现多边形和自由绘制闭合图形区域的 Poisson Image Editing 算法，需通过扫描线算法获取多边形内部掩膜。这里从网上资料了解到一种有序边表法，其基本思想是定义边表 ET 和活动边表 AET，ET 记录当前扫描线与边的交点坐标、从当前扫描线到下一条扫描线间 x 的增量、该边所交的最高扫描线，AET 记录只与当前扫描线相交的边的链表，通过迭代得到当前扫描线与待求多边形各边的交点，再利用奇偶检测法判断该点是否在多边形内部进行填充。
参考博客:https://blog.csdn.net/xiaowei_cqu/article/details/7712451

4 程序架构

4.1 文件结构

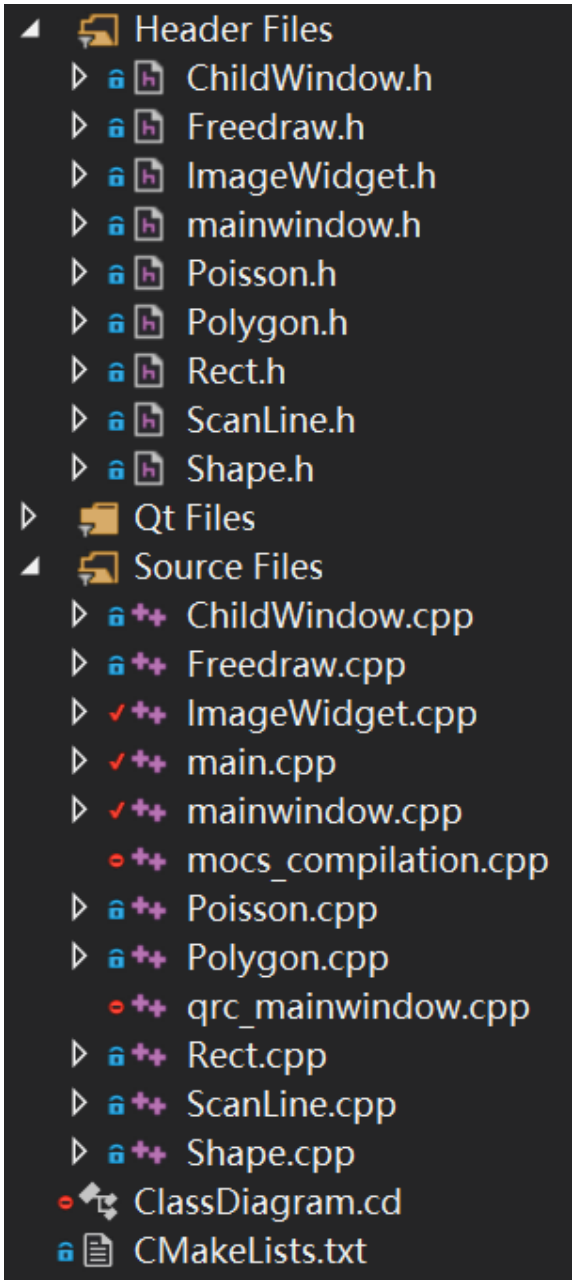

Figure 4.1: 文件结构

4.2 面向对象设计

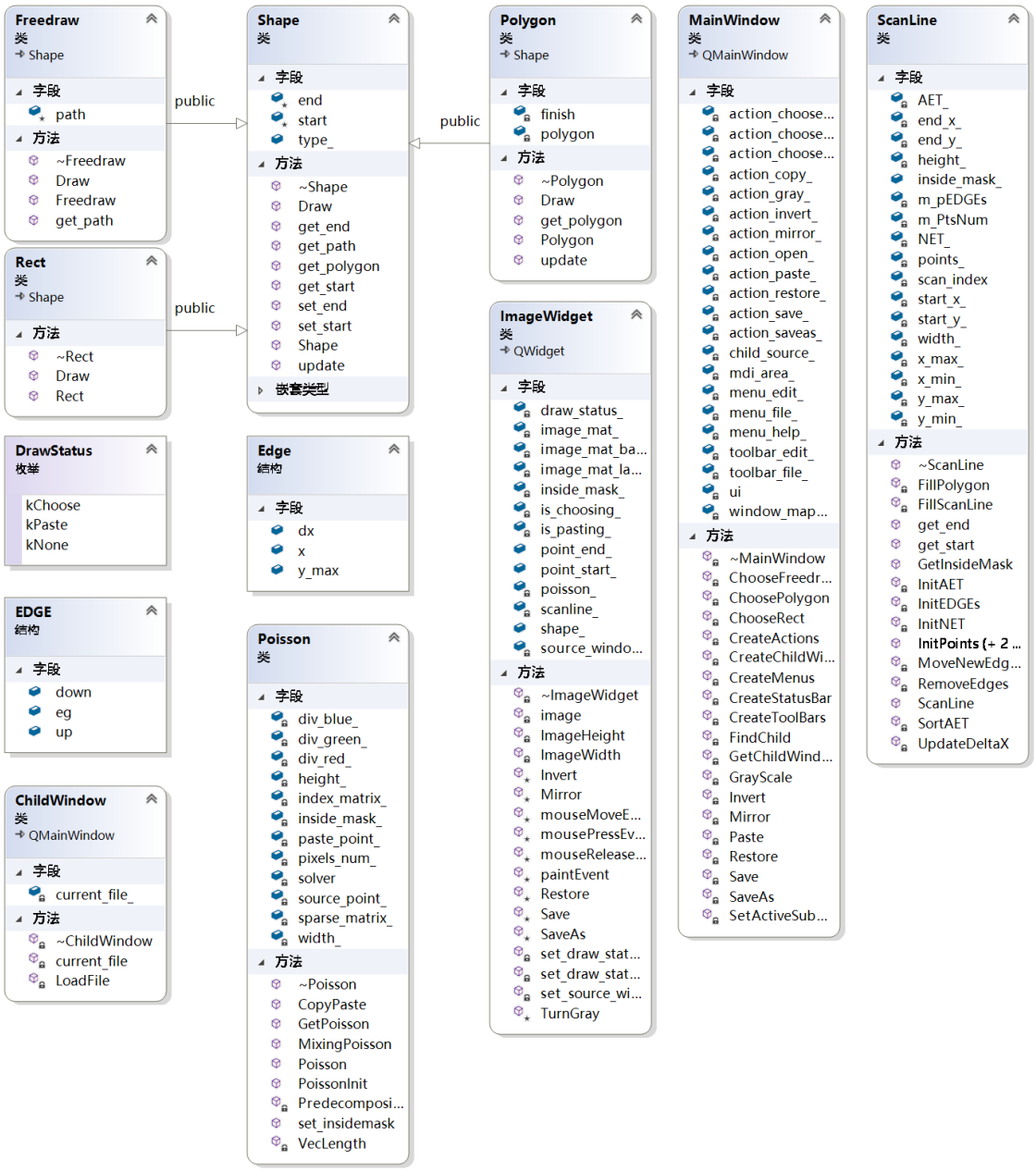

Figure 4.2: 类图

直接使用实验一 MiniDraw 中的 Shape 类进行修改实现 Rectangle、Freedraw 和 Polygon 形状的绘制，使用 Poisson 类实现 Poisson Image Editing，ScanLine 类实现多边形内部填充算法

5 设计难点与解决

5.1 OpenCV 框架的移植

使用 OpenCV 进行图像处理会比直接使用 QImage 进行像素操作方便很多，在移植过程中要注意各个涉及 QImage 的环节都要更换为 Mat 类的等价表示，在显示图像的最后一步将 Mat 类型转为 QImage 类型实现 Qt 上的显示

5.2 图像处理的实时显示

由于图像在新背景中拖动的时候只改变边界值，即矩阵表示的线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 中的 $\boldsymbol{b}$，利用该特性可以采用矩阵预分解减小计算量，经比较采用 Eigen 的 SimplicialLLT 求解器，在 release 模式下可达到实时显示的效果。

5.3 关于扫描线算法

除了利用有序边表的方法实现多边形内部的填充，也可以利用 OpenCV 的 fillPoly 同样能够进行多边形内部的填充，经测试可以实现同样的效果。

6 实验效果

6.1 标准图像测试

原图像：

Figure 6.1: bear

Figure 6.2: girl

新背景图像：

Figure 6.3: sea

6.1.1 *Rectangle*

Figure 6.4: bear

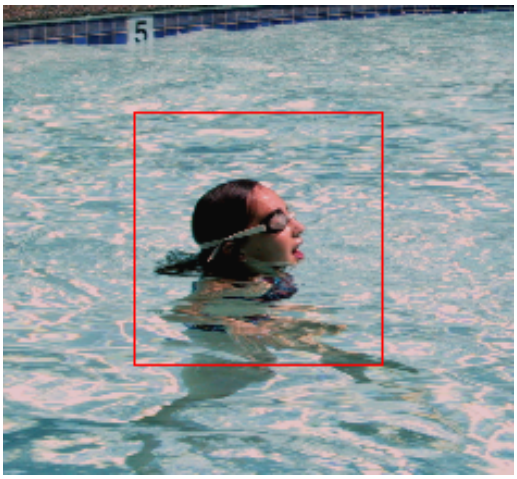

Figure 6.5: girl

Figure 6.6: 处理效果

6.1.2 Polygon

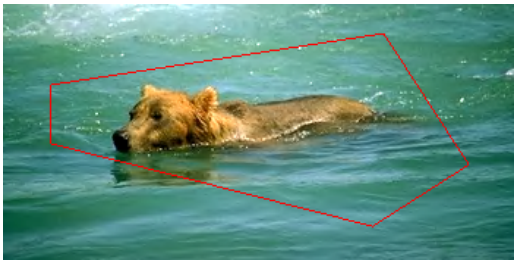

Figure 6.7: bear

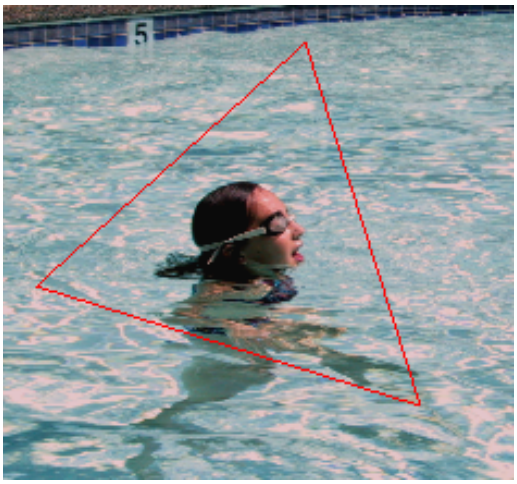

Figure 6.8: girl

Figure 6.9: 处理效果

6.1.3 *Freedraw*

Figure 6.10: bear

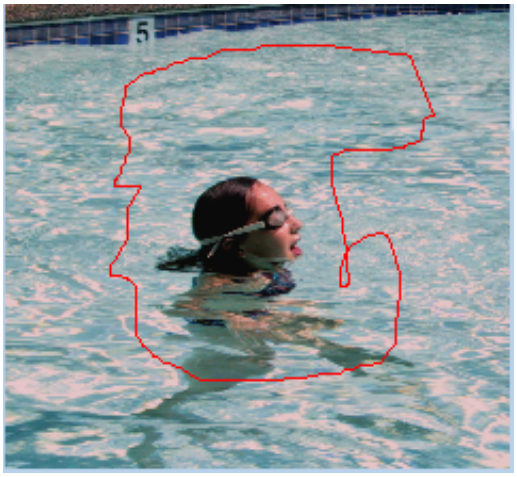

Figure 6.11: girl

Figure 6.12: 处理效果

6.1.4 Poisson & Mix Poisson

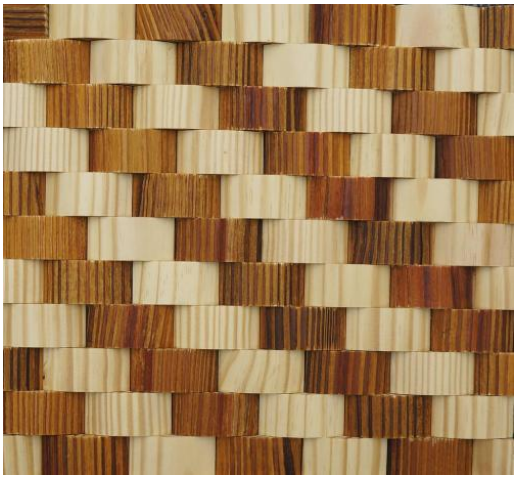

Figure 6.13: background

Figure 6.14: sample

Figure 6.15: 处理效果

如图，左上为直接复制粘贴，保留前景全部颜色梯度信息；左下为普通 Poisson 编辑，保留前景全部梯度信息，前景像素颜色与背景作融合；右上为应用混合梯度的 Poisson 编辑，前景梯度部分保留，效果上比普通 Poisson 编辑更加“透明”，适合用在水印等场景。

6.2 其他的应用

6.2.1 遮盖不必要的信息（如去皱纹）

原图像：

Figure 6.16: 抬头纹

处理效果：

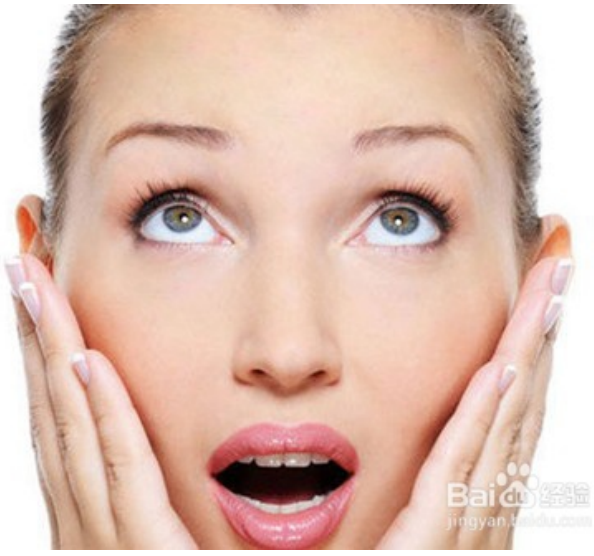

Figure 6.17: 利用脸部其他部位的纹理祛皱

6.2.2 恐怖片特效

原图：

Figure 6.18: 电影《鬼三惊》剧照

镜中人物掩盖：

Figure 6.19: 处理效果

恐怖角色原图：

Figure 6.20: 《招魂》系列中鬼修女形象

处理效果：

Figure 6.21: 处理效果

6.2.3 生成表情包

原图：

Figure 6.22: Richard Milos 高清图

处理效果：

Figure 6.23: Richard Milos 熊猫头表情包

7 总结

本例中忽视了原图像选取范围和图像边界重合即 $|N_p| < 4$ 的情况，还可进一步优化。计算上还是采用遍历像素操作的方式进行处理，或许可以对计算方式类似的区域使用矩阵方块操作提高速率

REFERENCES

[1] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pages 313–318. 2003.