计算机图形学

第二次实验报告

ImageWrapping

PB20000264

韩昊羽

一、实验要求

- 阅读论文，实现 IDW 和 RBF 算法
- 巩固面向对象编程方法，继承与多态
- 使用 QImage 类
- 学会使用 eigen 库解线性方程组
- 拓展：去白缝
- 拓展：RBF 的其他改进
- 对各个算法结果进行比较

二、操作环境

2.1 QT 图形化编程

IDE：Microsoft Visual Studio 2019 community

QT: 5.12.12

2.2 Eigen 库

Eigen ： 3.4.0

三、 功能介绍

3.1 导入图片

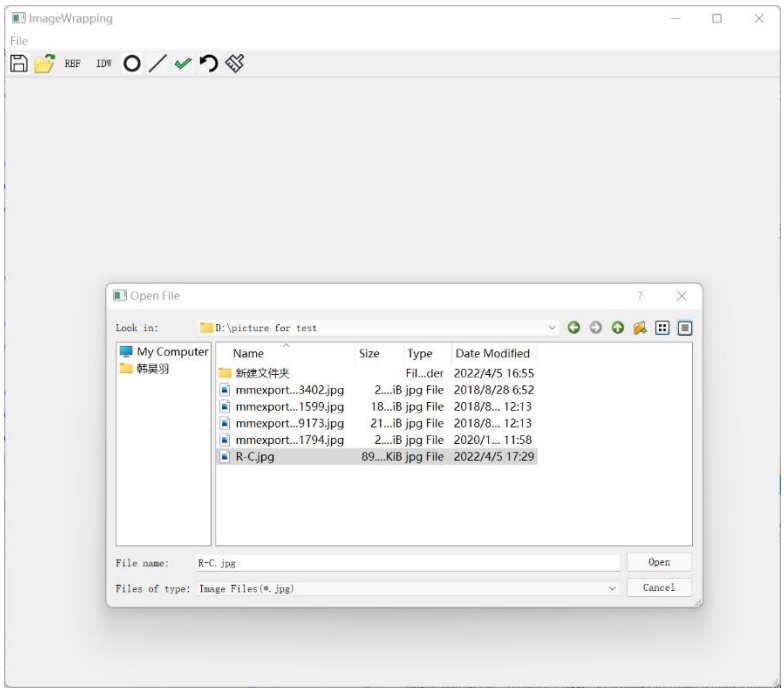


Figure.1 导入图片

主要通过 filedialog 实现。

3.2 保存图片

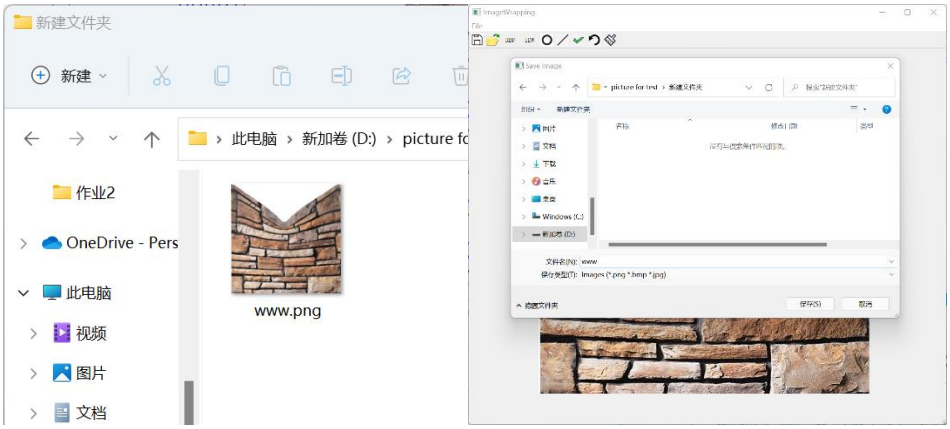


Figure.2 保存图片

3.3 绘制操作点（交互）

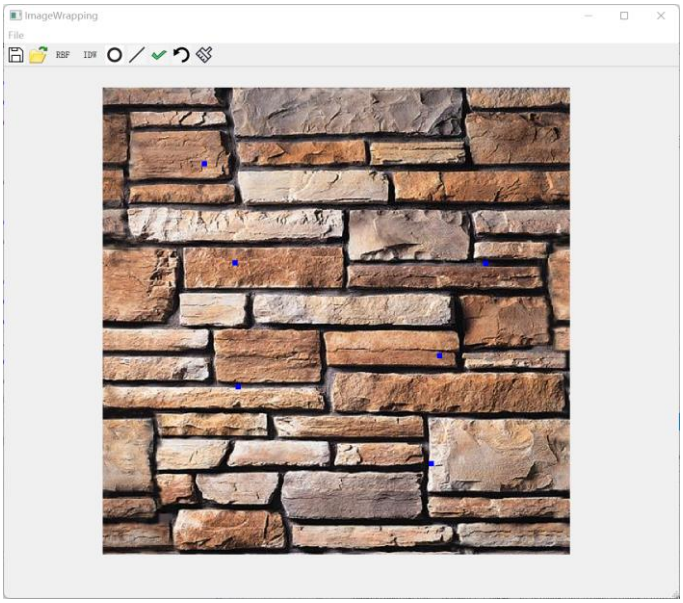


Figure.3 绘制操作点

操作点表示点在变换过程中不发生改变，是不动点，通过蓝点表示。

3.4 绘制操作线（交互）

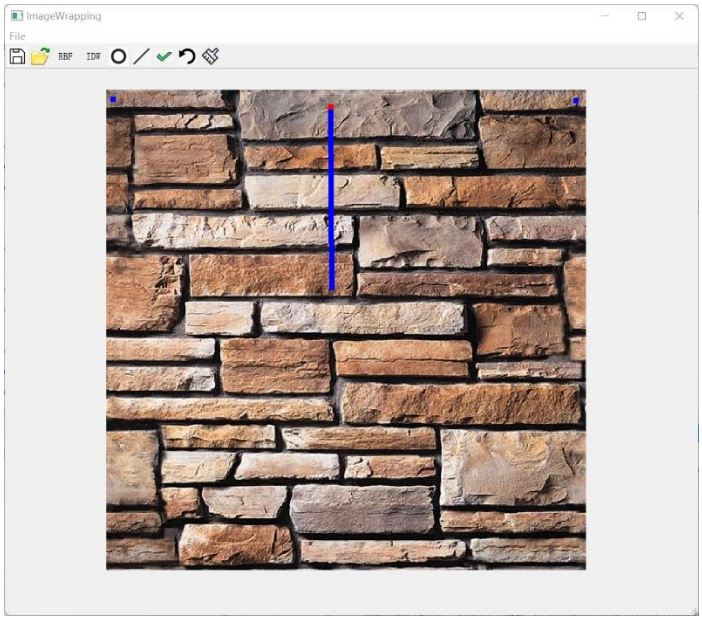


Figure.4 绘制操作线

操作线的绘制和 MiniDraw 中直线的绘制一样，都是通过鼠标拖拽确定位置，其中红点位置表示初始位置，而另一端的蓝点

位置表示目标位置。

3.5 撤销

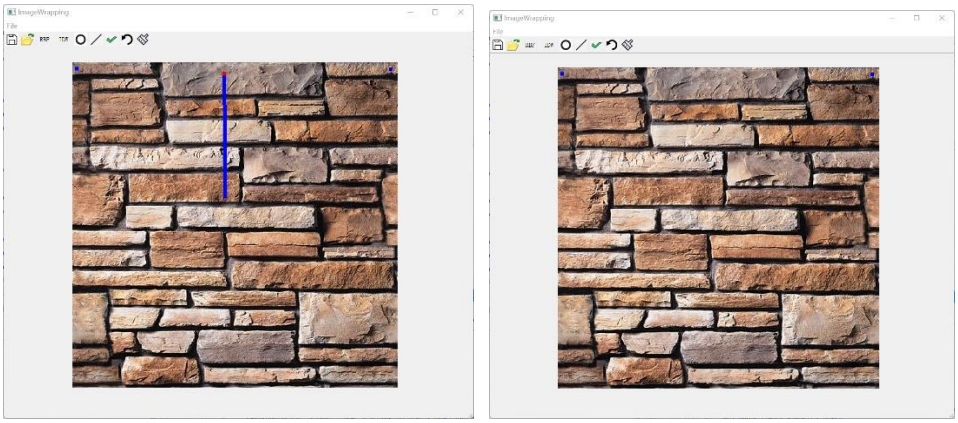


Figure.5 撤销

撤销操作可以去除一条操作点线。

3.6 恢复初始情形（工具栏最后一个）

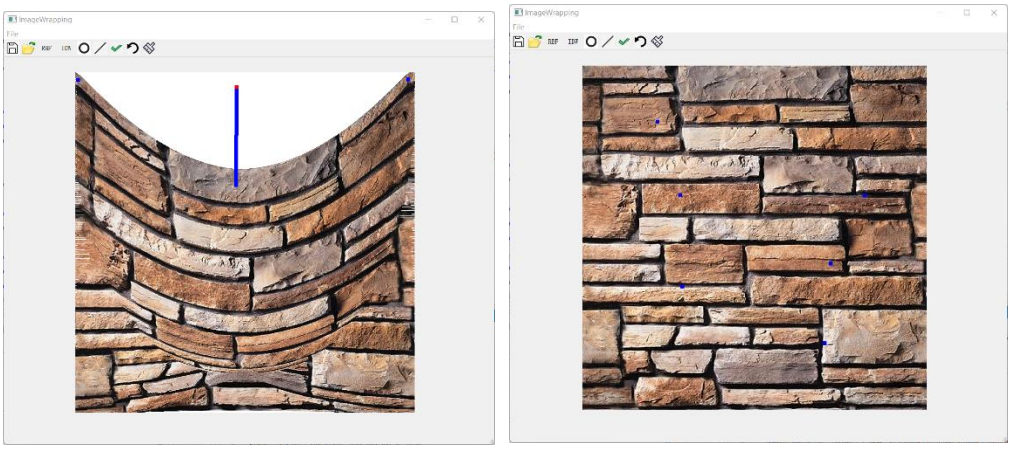


Figure.6 恢复

可以将已经变换过的图片恢复初始位置。

3.7 IDW 算法

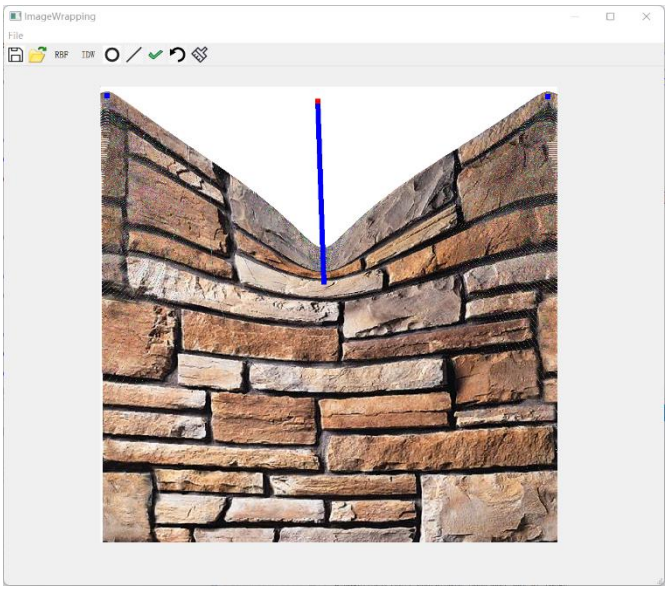


Figure.7IDW

通过 IDW 算法对图像进行变换。

3.8 RBF 算法

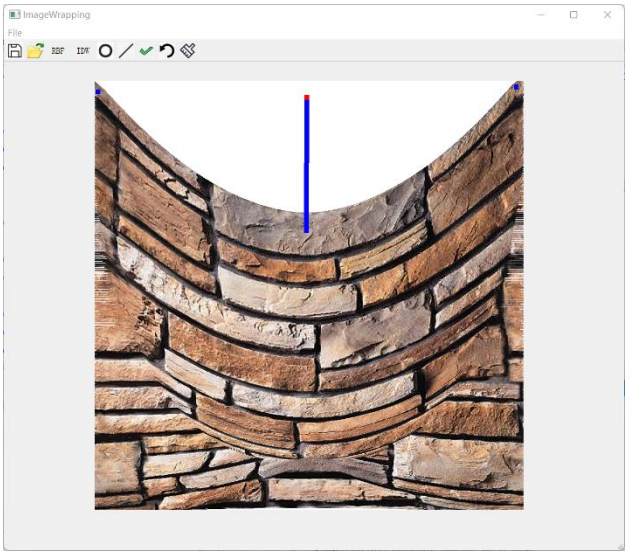


Figure.8 RBF

通过 RBF 算法对图像进行变换。

3.9 去除白缝

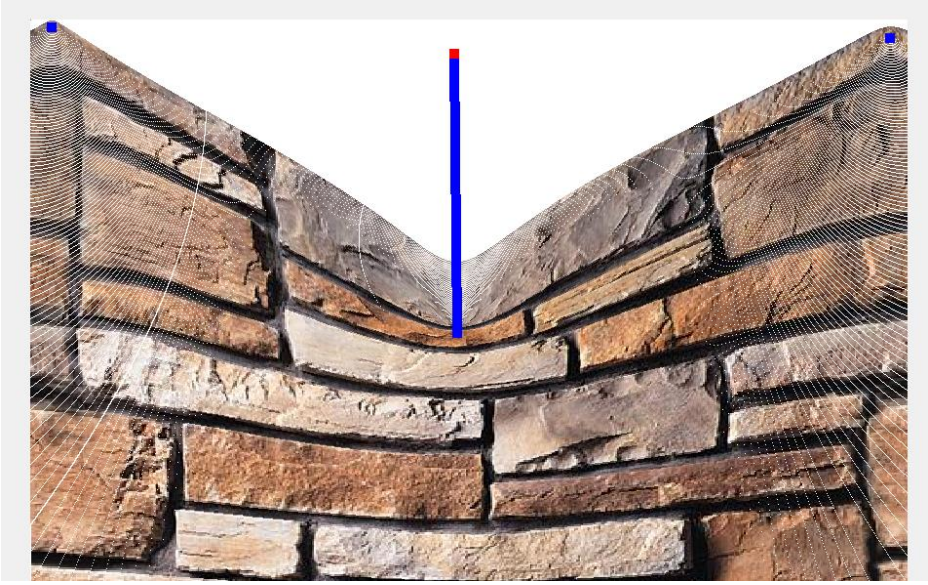


Figure.9 去缝（图片为去缝前）

去除因为映射不是满射而产生的白缝。

四、架构设计

4.1 文件结构

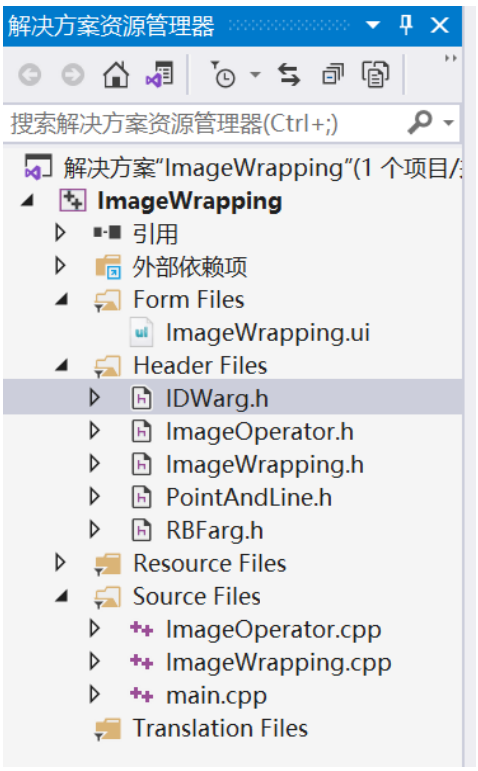


Figure.10 文件结构

头文件主要包括 imageoperatator 是对图像矩阵进行操作的类，IDW 和 RBF 是继承自 imageoperator 的；两个类，主要负责两个算法的具体实现。Pointandline 主要涉及到操作点和操作线的属性存储，imagewrapping 是主窗体中相关变量的存储。

Cpp 文件主要是 imagewrapping.cpp，涉及到打开关闭文件管理器、绘制图像和操作点线、更新、交互、调用 IDW 和 RBF 类等功能。

4.2 类图

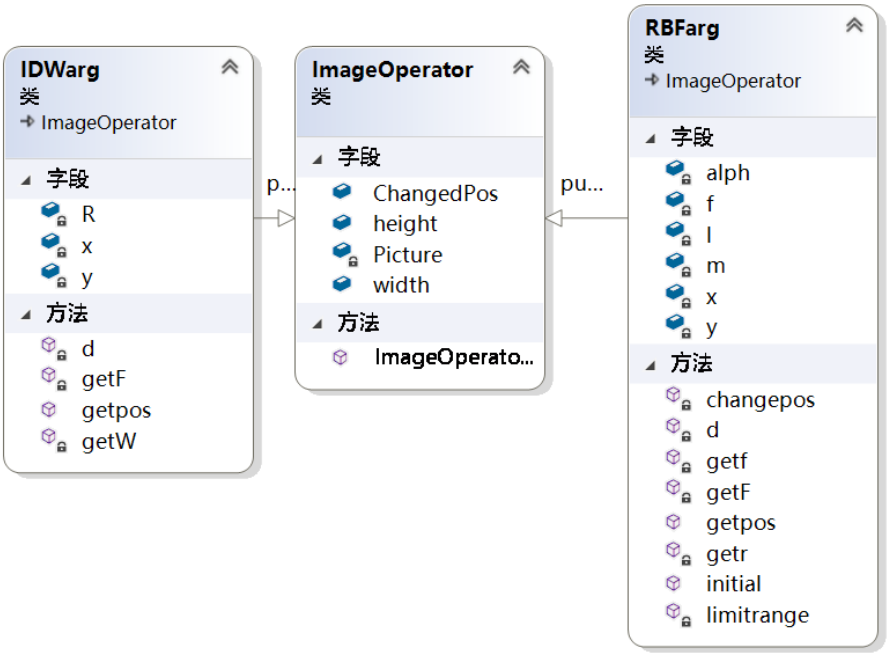


Figure.11 类图 1

这张类图主要呈现了与算法相关的类，基类 ImageOperator 涉及到一个存储操作点线的 vector 向量 ChangedPos，图片的高度和宽度。

具体到两个算法，IDW 中 R 是参数，X,Y 是中间变量。Getpos 是接口函数，传入一个点的坐标后返回变换后的坐标。GetF 和 GetW 都是为了简便计算所定义的函数。RBF 的变量要多一些，m 是矩阵，f 和 l 是 AX=B 中的 B 和 X，alpha 是经过转化后系数的向量，x，y 是中间变量。特别的，RBF 多了一个 initial 函数主要负责矩阵的初始化和求解，eigen 库的调用在这里进行。Limitrange 函数则是获得函数的正函数（大于零取原值，小于零取零）。

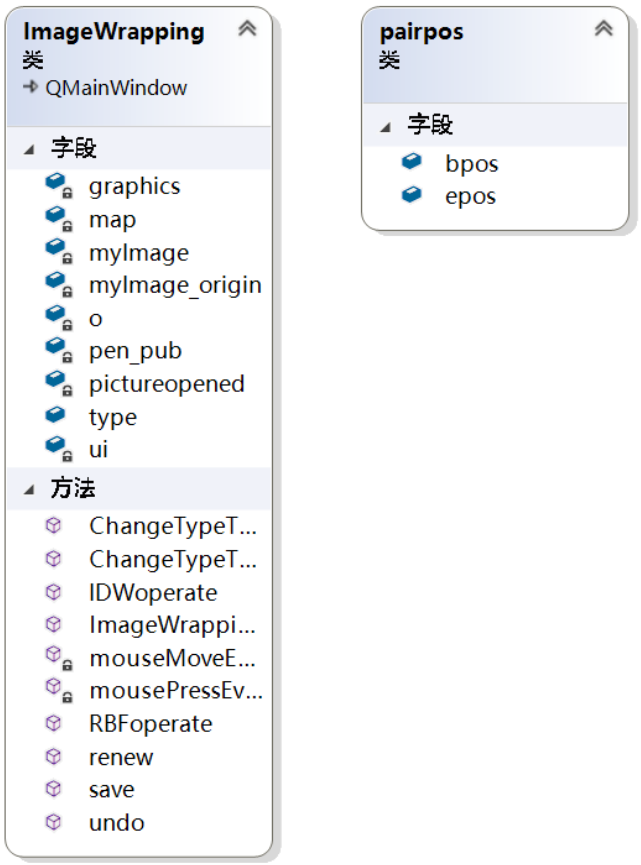


Figure.12 类图 2

Pairpos 里就是一对操作点（起点和终点）。ImageWrapping 里存的是主窗体的相关字段，map，myimage 和 myimage_origin 都时关于图像显示储存的，graphics 是操作点在画布上的呈现，

penpub，type 是关于追踪鼠标并添加操作点线的。Save，renew，undo 和两个 changetype 函数都是 ui 的槽函数。IDW/RBFoperate 是进行 RBF 和 IDW 算法的计算。、

五、功能实现

5.1 打开保存功能

主要通过 QT 中的 filedialog 类，可以自定义打开文件选择器，其中有打开模式和保存模式两种，可以在 filedialog 关闭时获取文件路径，用 QImage 类打开图片，同样也要让图片大小和屏幕大小适配。

5.2 绘制功能

与 MiniDraw 的绘制功能类似，都是通过追踪鼠标事件实现，不同的是在绘制操作线时需要绘制一条线和一个点，在细节处理上有些不同。同时还要检测鼠标是否在图片内部。

5.3 撤销功能与恢复功能

通过弹出 imageoperator 中 changedpos 和主窗体中 graphics 来进行撤销。恢复就是弹出所有并且把图像恢复。

5.4 算法实现

找到插值函数f使得$f(p_i) = q_i$。

5.5 IDW

通过阅读文献，根据控制点对对不同位置的影响，计算反距离权重，实现每个点的位移。公式如下：

$$f(p) = \sum_{1}^{n} w_i(p) f_i(p)$$

其中 w 满足

$$w_i(p_i) = 1, w_i(p) \geq 0 \ , \ \sum_{1}^{n} w_i(p) = 1$$

权重 w 定义为

$$w_i = \frac{\sigma_i(p)}{\sum_1^n \sigma_j(p)}$$

其中σ可以取

$$\sigma_i(p) = \frac{1}{d(p,p_i)^u}$$

u 可以取 1，2，3 等等

或者取

$$\sigma_i(p) = \left[\frac{R - d(p,p_i)}{Rd(p,p_i)}\right]^u$$

locally bounded weight 函数

并且

$$f_i(p) = q_i + p - p_i$$

满足 $f_i(p_i) = q_i$

5.6 RBF

RBF 算法通过待定一个线性方程组的系数在径向基函数下进行分解。

公式为

$$\boldsymbol{f(p)} = \sum_{\boldsymbol{1}}^{\boldsymbol{n}} \boldsymbol{\alpha_i R_i}(||\boldsymbol{p} - \boldsymbol{p_i}||) + \boldsymbol{p}$$

其中 R 取

$$\boldsymbol{R_i(d)} = (\boldsymbol{d^2 + r_i^2})^{\frac{\boldsymbol{1}}{\boldsymbol{2}}}$$

其中

$$\boldsymbol{r_i = min\{||pi - pj||\},\ i \neq j}$$

而系数$\boldsymbol{\alpha}_{\mathrm{i}}$可以由$f(p_i) = \mathrm{q_i}$解得

进一步，如果想控制点的作用范围可以改进 R

$$R_i^* = \boldsymbol{R_i(d)} * (\boldsymbol{1} - (\frac{\boldsymbol{d}}{\boldsymbol{L_i}})^{\boldsymbol{2}})^{+}$$

其中$(\ \ )^{+}$表示正函数(小于零时取 0)

六、难点难题

6.1 图片实时更新问题

因为要在图片上面显示控制点对，需要在图片上画图，但图片的 QImage 本身需要呈现在 label 画布上，所以如果还是在主窗口画图的话轨迹就会呈现在 image 的下方，我采用了笨办法解决，即先将控制点对画在一个 pixmap 对象上，pixmap 本身存的对象是 image，再

将这个 pixmap 投射到 label 上。

6.2 绘制问题

因为控制线需要画一条蓝线和一个红点，所以在鼠标创建的时候要创建两个对象，而且红点要显示在蓝线的上方，所以不能直接选择 graphics 中最后一个图形操作，需要检测是不是控制线，再决定是选倒数第一个还是倒数第二个。

6.3 全空白问题

有时会出现操作后整个画布变成空白，调试后发现是个别参数取得值太大或太小导致。

6.4 获取坐标问题

最初我处理绘制点对的时候，不知道为什么最后绘制的位置总是比鼠标的位置高一截，我尝试了各种坐标但还没有解决，最后发现是因为 toolbar 的存在使坐标发生了偏移，最后需要添加一个偏移项。

6.5 白缝问题

在坐标变换结束后，因为映射不是满射，会导致一些点没有像素，在图片上呈现出一条条的“白缝”。白缝问题主要有下面三种解决方案：

- 通过 f 的表达式求解出一个近似的逆函数，然后把对应点代入求解
- 寻找白缝附近最近的点取其像素值

- 把白缝附近的一些点的像素值加起来取平均

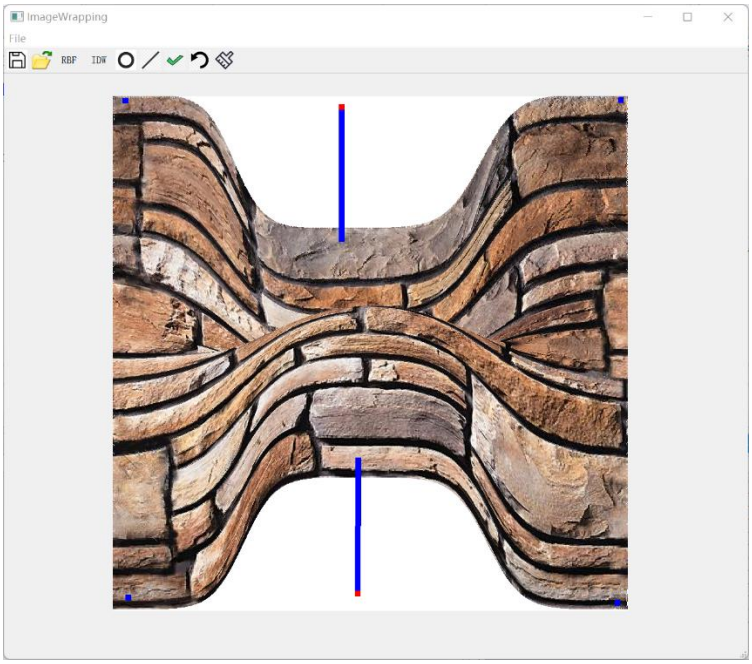
我主要实现了后两种算法，在寻找白缝附近点的过程中我发现如果直接便利寻找最近点的话效率比较低，且扭曲感比较强，所以采用了只在四个方向上寻找：左上左下，右上右下，通过 while 循环直到找到一个涂色的最近点（是原始涂色，不是在去白缝的过程中涂的色）。缺点是还会有一些点没有被上色，且会轻微扭曲。

另一种方法可能在求平均时不均匀从而导致有一些“彩线”出现。

七、实验结果

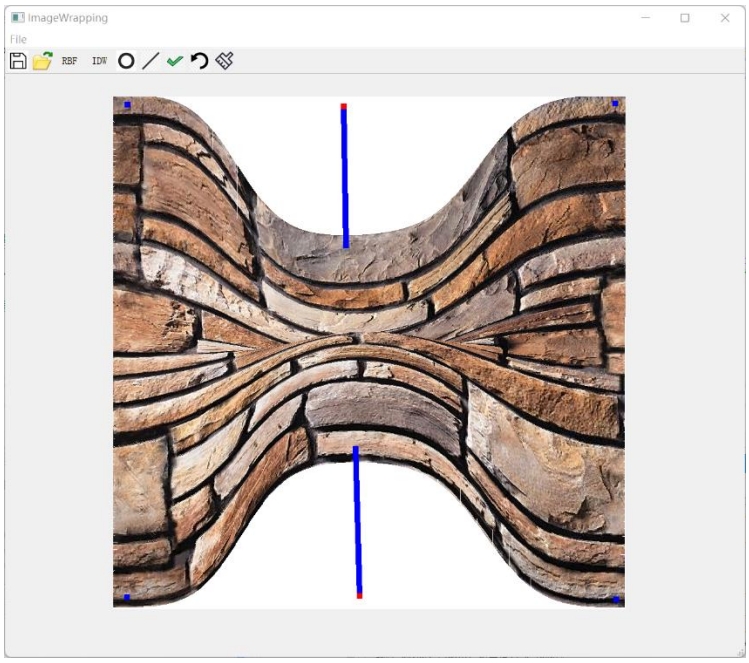
各种算法比较：测试图片的大小为 900*900

- IDW 算法采用 locally bounded weight 函数，u=3，R=1000



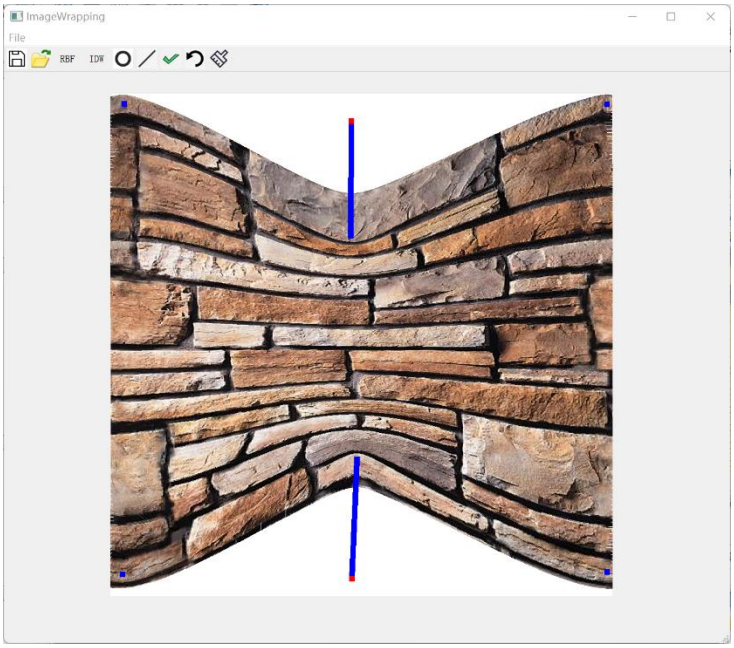
Time：9944ms

- IDW 算法采用 locally bounded weight 函数，u=2，R=1000



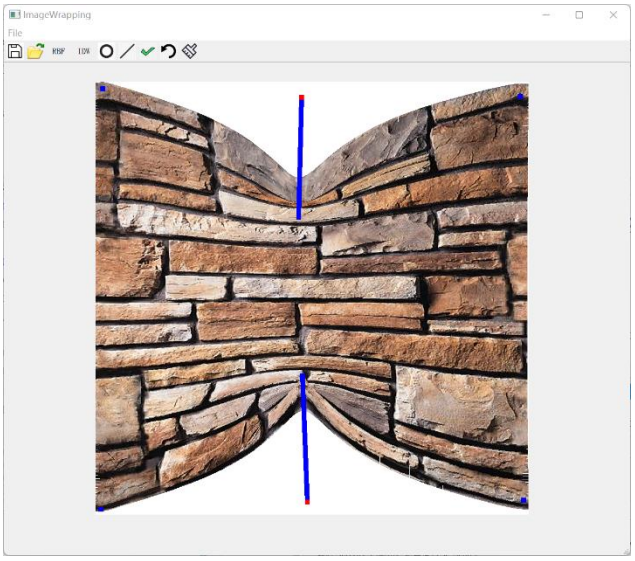
Time：9204ms

- IDW 算法采用 locally bounded weight 函数，u=1，R=1000



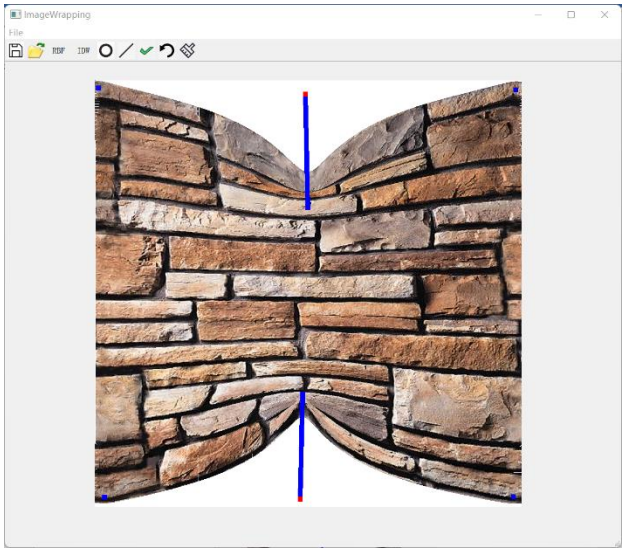
Time=7298ms

- IDW 算法采用 locally bounded weight 函数，u=1，R=5000



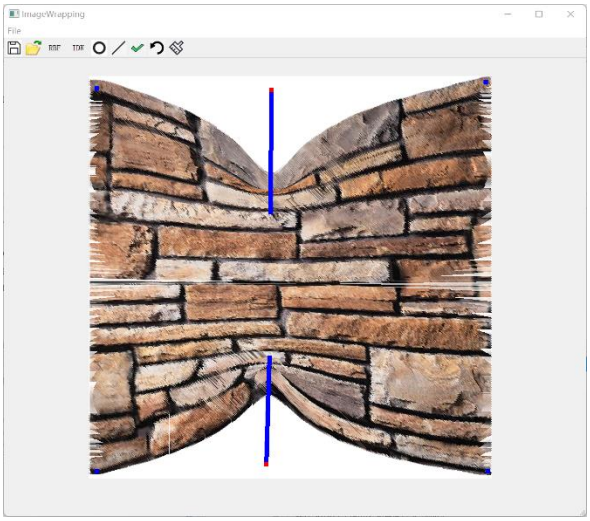
Time：7091ms

- IDW 算法采用普通距离函数，u=1



Time：5074ms

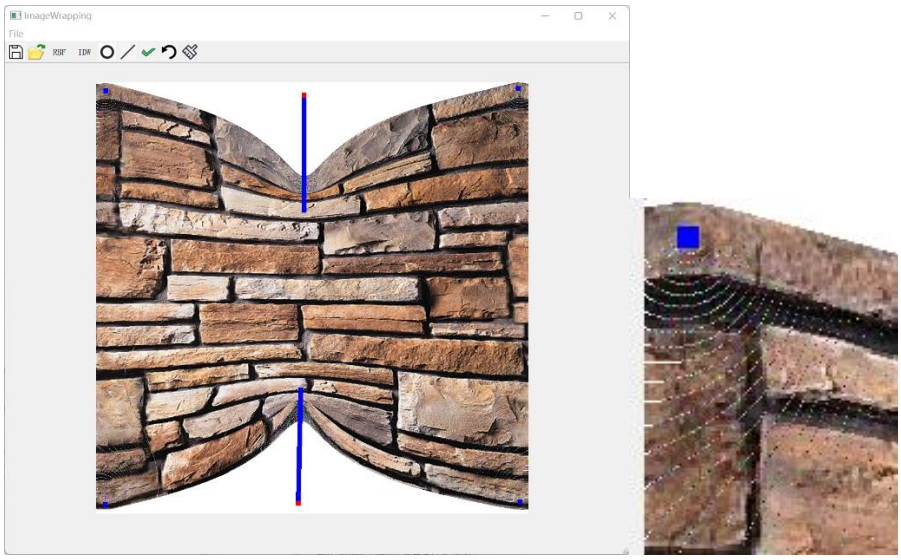
- IDW 算法采用普通距离函数,并且采用取样的方法隔三个取一个，u=1



Time：1507ms

以上的算法处理白缝使用的都是第二种找最近点方法。

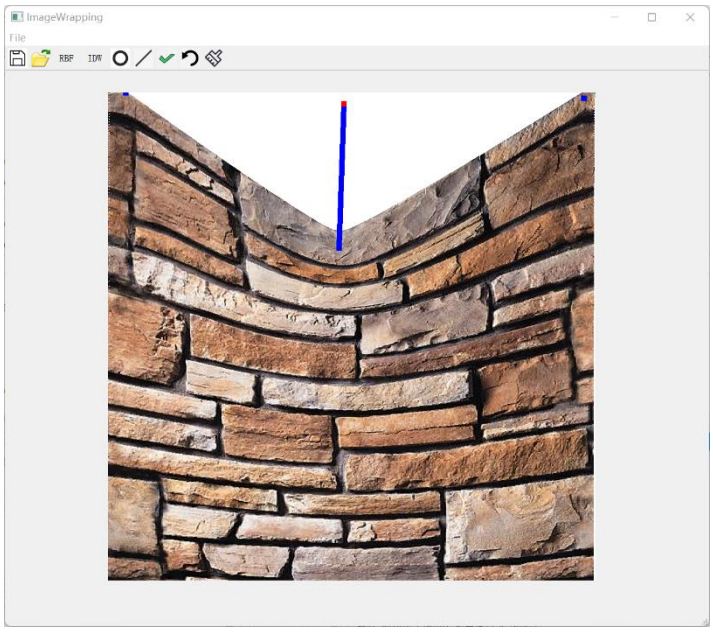
- IDW 算法采用普通距离函数，白缝处理采用平均值方法，u=1



Time：4338ms

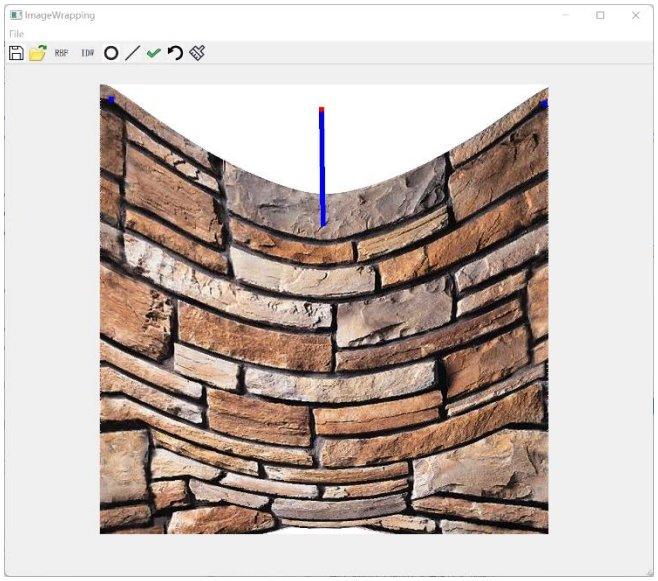
- RBF 算法采用改进距离函数（约束作用范围），L=2000，u=1

（修正因子对应的指数，主因子对应指数都取 0.5）



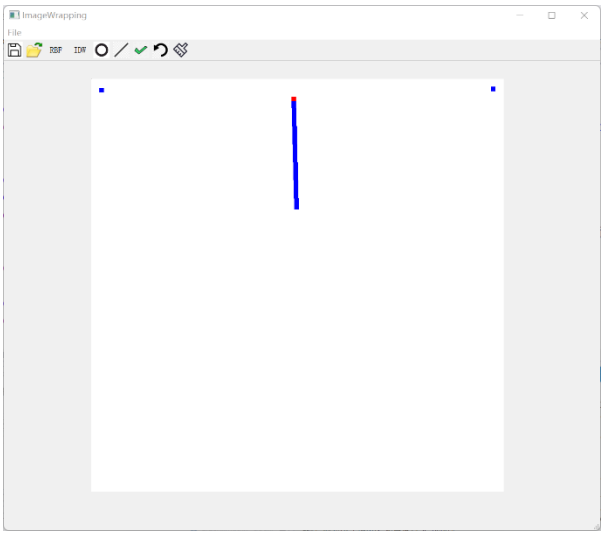
Time：6490ms

- RBF 算法采用改进距离函数（约束作用范围），L=1000，u=1



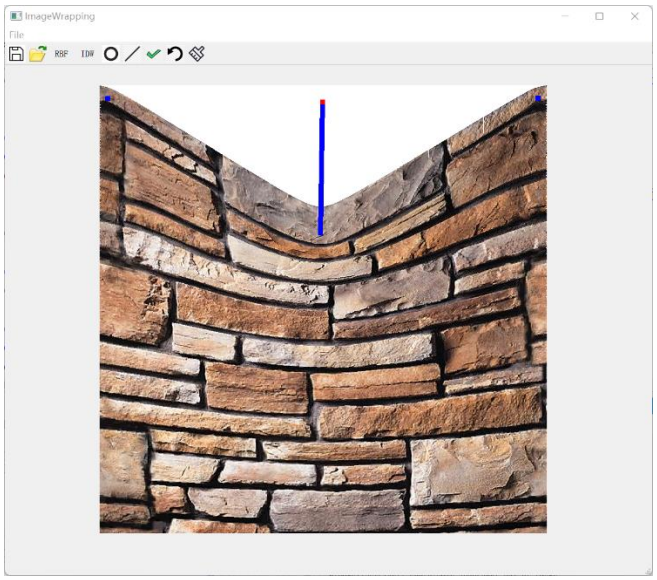
Time=3575ms

- RBF 算法采用改进距离函数（约束作用范围），L=500，u=1



Time = 2624ms，出现全白

- RBF 算法采用普通距离函数



Time = 5420ms

对各类算法进行比较，我们可以直到 RBF 算法整体表现比 IDW 算法好，时间更快且不会出现明显扭曲，作用范围恰当，IDW 算法很多时候需要固定四个角才能工作。具体到 IDW 算法内部，locally bounded weight 函数比普通距离函数要平滑一些，但是花费的时间也更长，两个白缝算法各有优点，取平均值会出现明显的彩条，整体上取最近点更胜一筹。RBF 算法内部，改进算法的表现并不是很

好（我自己的问题），在 L 小的时候会变成全白，且图像不够平滑，也是剩余的问题，我会在以后改进。

八、问题与展望

8.1 遇到的问题

- RBF 改进算法
- RBF 差分问题使能量最小
- 点的绘制坐标偏差
- 运行速度（最大的问题）
- IDW 图像局部扭曲
- 白缝处理不完全

8.2 future work

- 对各自改进算法的研究
- 加快时间（三角形网格）
- 更好的交互界面
- 实时计算
- 白缝算法的改进，逆函数方法
- 面向对象思想的更多应用与理解