

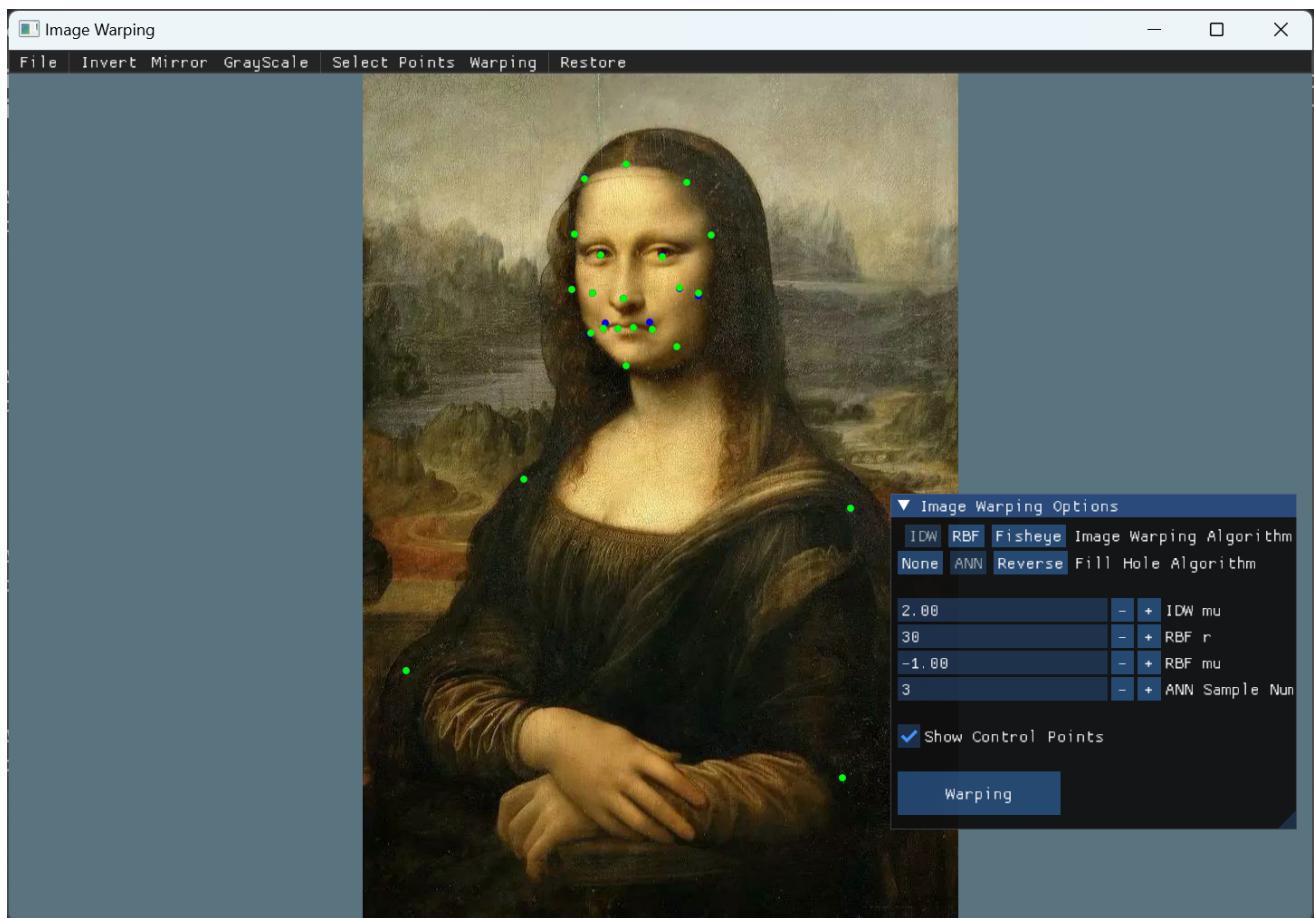
Homework 2 作业报告 by 76-朱雨田

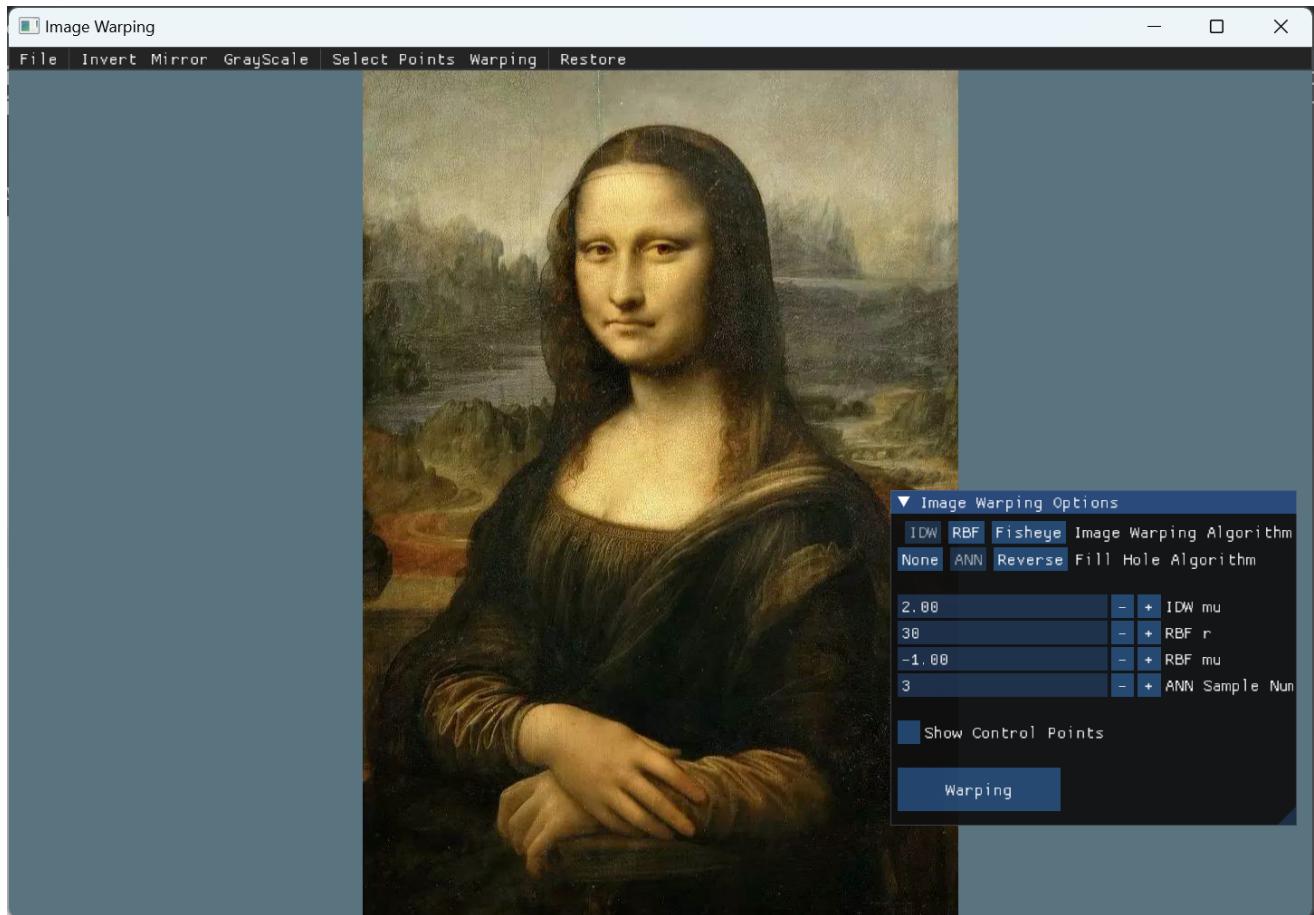
本次作业要求根据文档及论文实现 IDW、RBF 两种图像变形算法，初步理解数学插值的思想，并学习 Eigen 库实现线性方程组求解，ANN 库以实现填缝的功能，对比不同参数对图像变形结果造成的影响。

我在实现以上算法的基础上设计了一个参数控制窗口（下图 Image Warping Options），可以在 GUI 内修改各个算法的参数，从而能更容易地比较不同算法、参数对 Image Warping 造成的影响。控制窗口中还设置了是否显示控制点的选项，以便更清晰地查看图像变形的结果。

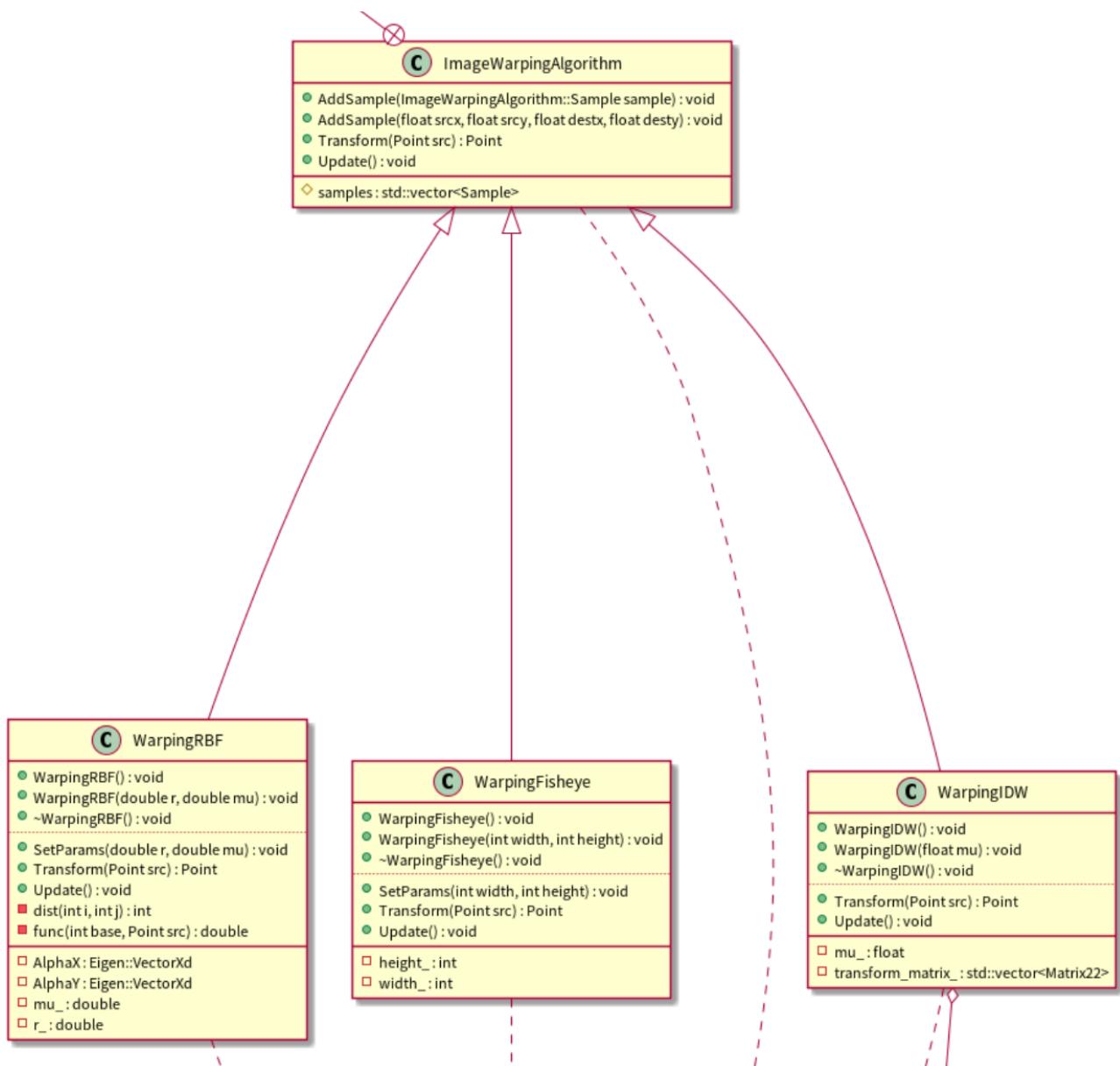
每次执行 Warping 算法时，都会从控制台输出当前算法的参数及耗时。

以下是效果展示：

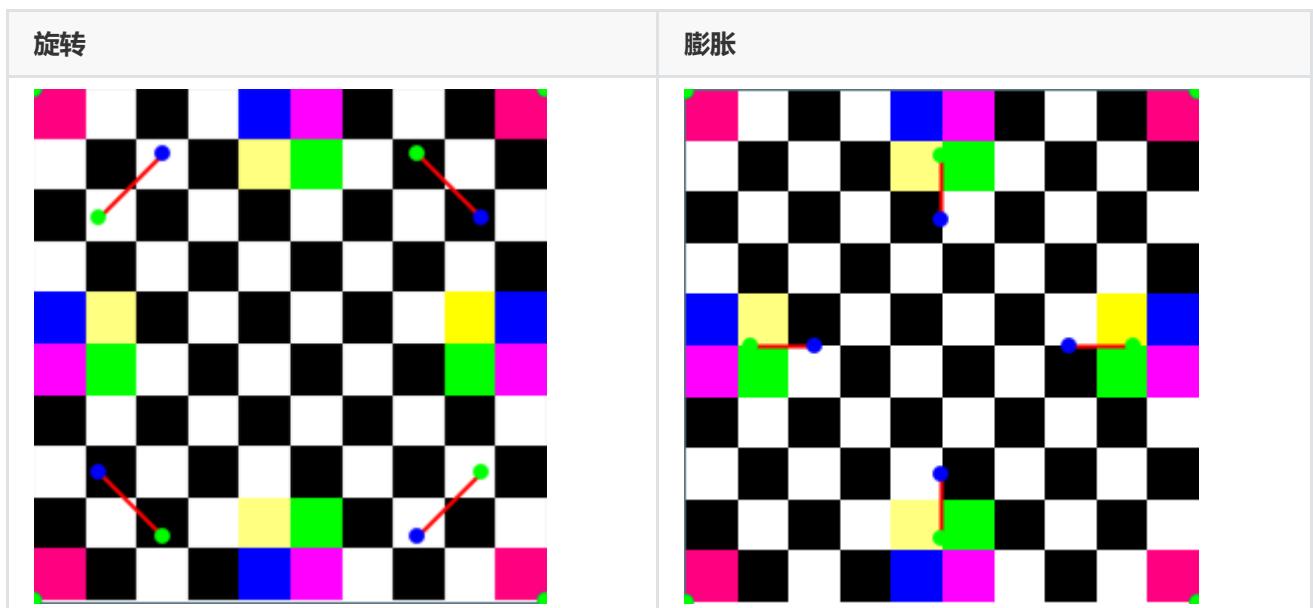




以下是本次作业的部分类图：



本次报告的图片对比采用以下两个控制点预设：



1. IDW

IDW，即距离倒数权重。因为我们的目的是提高控制点附近的图像变形幅度，并尽量减小对离控制点较远的图像的影响，尽量少地造成不必要的变形，因此以距离的倒数作为权重是很自然的想法。每个像素都会按照其与每个控制点目标点的距离倒数权重被映射到新的位置上，以达到平滑地将一些位置变形到指定的位置上的效果。

论文给出了以下插值函数：

$$\mathbf{f}_i(\mathbf{p}) = \mathbf{q}_i + \mathbf{D}_i(\mathbf{p} - \mathbf{q}_i)$$

其中 $\mathbf{D}_i : \mathbb{R}^2 \rightarrow \mathbb{R}^2$, 满足 $\mathbf{D}_i(\mathbf{0}) = \mathbf{0}$

$$\mathbf{f}(\mathbf{x}) = \sum_{i=1}^n w_i(\mathbf{x}) \mathbf{f}_i(\mathbf{x})$$

其中 $w_i : \mathbb{R}^2 \rightarrow \mathbb{R}$, 为归一化的以下权重：

$$\sigma_i(\mathbf{x}) = \frac{1}{\|\mathbf{x} - \mathbf{x}_i\|^\mu}$$

其中 $\mu > 1$ 。

从 $\mathbf{f}(\mathbf{x})$ 的计算公式中可以看出它是连续且平滑的，从而能取得令人满意的图像变形效果。 μ 值越大，权重函数在 $\mathbf{x} \rightarrow \mathbf{x}_i$ 处就越陡峭。

文档还给出了计算 \mathbf{D}_i 的方式，即定义能量为

$$\begin{aligned} E_i(\mathbf{D}_i) &= \sum_{j=1, j \neq i}^n w_{ij} \left\| \mathbf{q}_i + \begin{pmatrix} d_{i,11} & d_{i,12} \\ d_{i,21} & d_{i,22} \end{pmatrix} (\mathbf{p}_j - \mathbf{p}_i) - \mathbf{q}_j \right\|^2 \\ &= \sum_{j=1, j \neq i}^n w_{ij} ((d_{i,11}(p_{j,1} - p_{i,1}) + d_{i,12}(p_{j,2} - p_{i,2}) + q_{i,1} - q_{j,1})^2 + \\ &\quad (d_{i,21}(p_{j,1} - p_{i,1}) + d_{i,22}(p_{j,2} - p_{i,2}) + q_{i,2} - q_{j,2})^2) \end{aligned}$$

并最小化该能量。

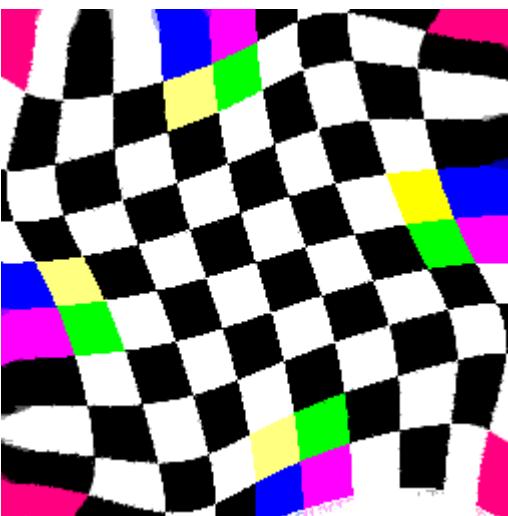
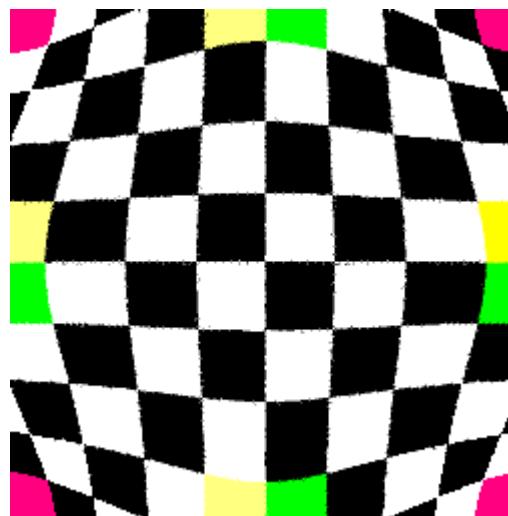
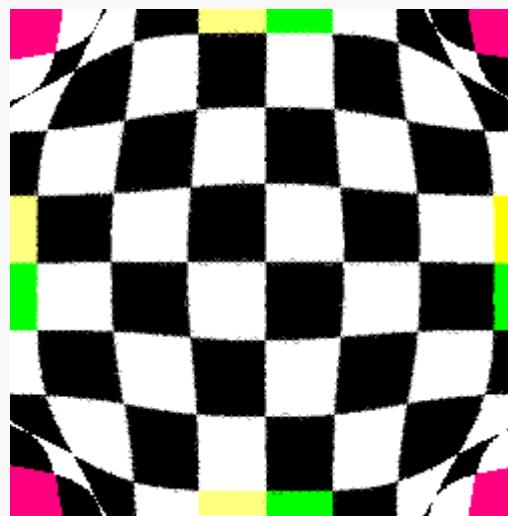
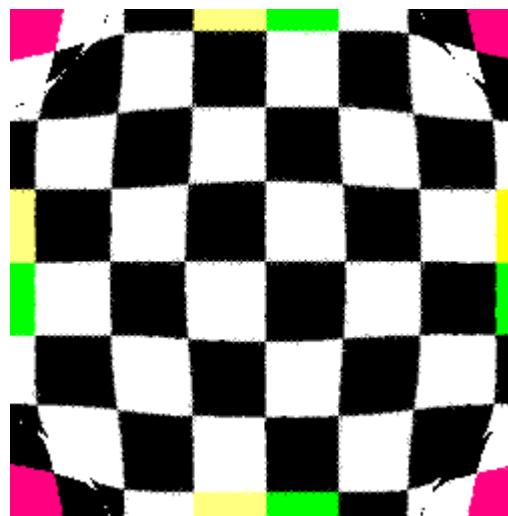
在做 USTC CG 2020 作业时我 skip 了这一部分（因为没学过最优化相关方法），不过在完成 USTC CG 2020 Homework5 (ARAP) 后我似乎看懂这里的最优化方法了。即对 $d_{i,j}$ 分别求偏导，用拉格朗日乘数法求出条件极值。这一步可以化为解下方程组：

$$\begin{aligned} \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,1} - p_{i,1}) (p_{j,1} - p_{i,1}) \right) d_{i,11} + \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,2} - p_{i,2}) (p_{j,1} - p_{i,1}) \right) d_{i,12} &= \sum_{j \neq i} w_{i,j} \cdot (q_{j,1} - q_{i,1}) (p_{j,1} - p_{i,1}) \\ \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,1} - p_{i,1}) (p_{j,2} - p_{i,2}) \right) d_{i,11} + \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,2} - p_{i,2}) (p_{j,2} - p_{i,2}) \right) d_{i,12} &= \sum_{j \neq i} w_{i,j} \cdot (q_{j,1} - q_{i,1}) (p_{j,2} - p_{i,2}) \\ \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,1} - p_{i,1}) (p_{j,1} - p_{i,1}) \right) d_{i,21} + \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,2} - p_{i,2}) (p_{j,1} - p_{i,1}) \right) d_{i,22} &= \sum_{j \neq i} w_{i,j} \cdot (q_{j,2} - q_{i,2}) (p_{j,1} - p_{i,1}) \\ \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,1} - p_{i,1}) (p_{j,2} - p_{i,2}) \right) d_{i,21} + \left(\sum_{j \neq i} w_{i,j} \cdot (p_{j,2} - p_{i,2}) (p_{j,2} - p_{i,2}) \right) d_{i,22} &= \sum_{j \neq i} w_{i,j} \cdot (q_{j,2} - q_{i,2}) (p_{j,2} - p_{i,2}) \end{aligned}$$

这个方程组可以写出其解析解。不过我这里仍然使用了 Eigen 进行求解。

当 $n = 1$ 时，因为系数全部为 0，所以该方程有无穷组解（Eigen 会导出错误的结果），求解这一矩阵会使得图像消失。因此程序中特判了 $n = 1$ 的情况，直接令 \mathbf{D}_1 为单位阵。

不同 μ 值下的效果如下（使用 ANN 补洞）：

μ	旋转	膨胀
2.00		
4.00		
6.00		

2. RBF

RBF 的基本思想是定义一系列径向基函数，并将它们线性组合，以对控制点进行平滑插值。径向基函数可以采取 $R(d) = (d^2 + r^2)^{\mu/2}$ 等形式。

n 个径向基函数线性组合后拟合到 n 组控制点上，这样就构造出了一个线性方程组，可以分别解出 x, y 方向的系数数组。为了减少图像本身的扭曲，我们在插值函数中引入仿射项 $\mathbf{A}\mathbf{p} + \mathbf{b}$ ，如下：

$$f(\mathbf{p}) = \sum_{i=1}^n \alpha_i R(\|\mathbf{p} - \mathbf{p}_i\|) + \mathbf{A}\mathbf{p} + \mathbf{b}$$

这使得系数矩阵变成了 $n \times (n + 3)$ 的非方阵，方程存在多解。为避免这种问题，论文提出了两种方法。

一是引入补充约束。为了保持系数矩阵对称阵的良好性质（目前也只能想到这个原因），我们引入三行补充约束，如下：

$$\begin{pmatrix} \mathbf{p}_1 & \cdots & \mathbf{p}_n \\ 1 & \cdots & 1 \end{pmatrix}_{3 \times n} \begin{pmatrix} \boldsymbol{\alpha}_1^\top \\ \vdots \\ \boldsymbol{\alpha}_n^\top \end{pmatrix}_{n \times 2} = \mathbf{0}_{3 \times 2}.$$

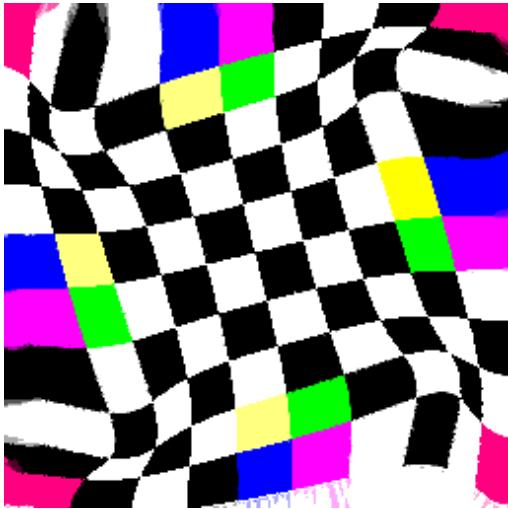
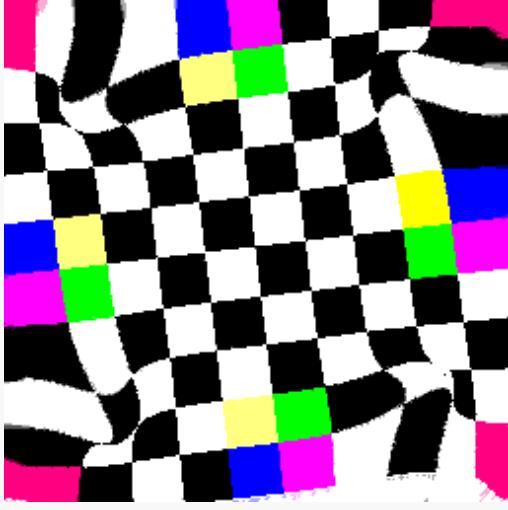
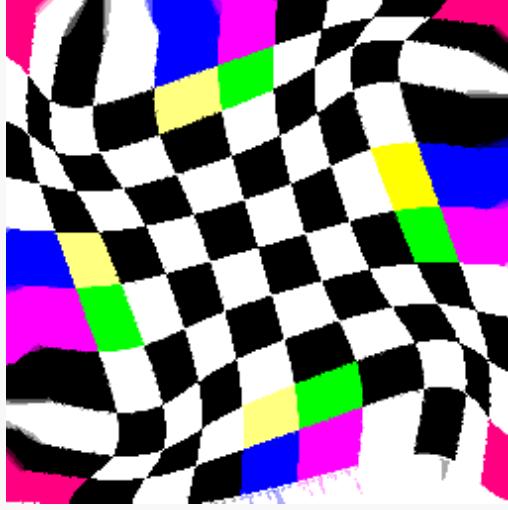
这样就能使系数矩阵变为 $(n + 3) \times (n + 3)$ 的，从而得到相当不错的结果。

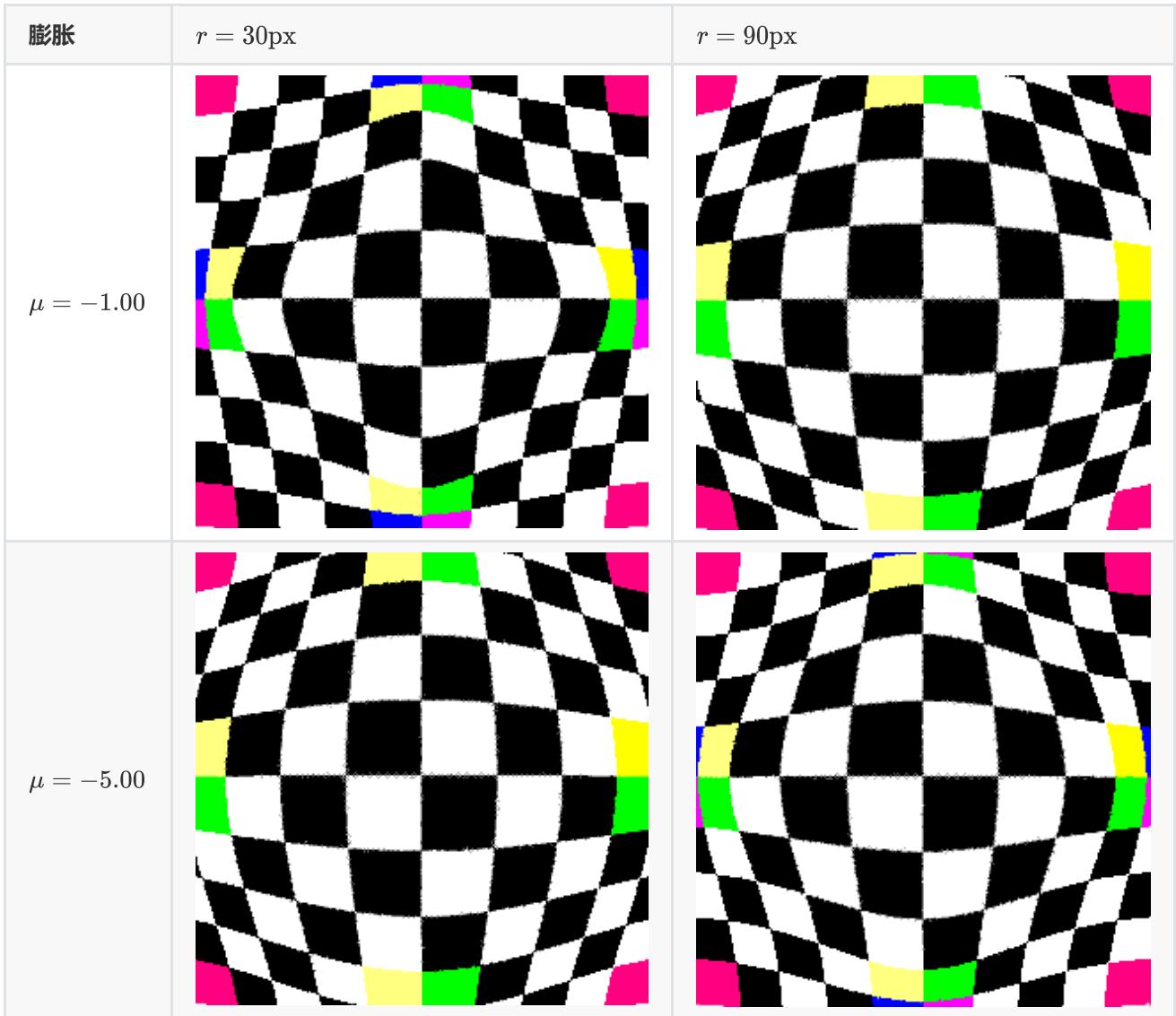
另一种思路是采用最小二乘法确定 \mathbf{A} 和 \mathbf{b} ，将问题转化为解额外的方程组。

我这里只采用了前者的方法。（最小二乘法的公式没推出来）

在 $n \leq 3$ 时，与 IDW 中的情况类似，方程会因为低秩而导致有无穷组解。这里采用了文档描述的方法，即手动计算仿射变换，并将其应用于图像（而不是用 RBF 去计算）。

以下展示了不同 r 与 μ 的效果（使用 ANN 补洞）：

旋转	$r = 30\text{px}$	$r = 90\text{px}$
$\mu = -1.00$		
$\mu = -5.00$		



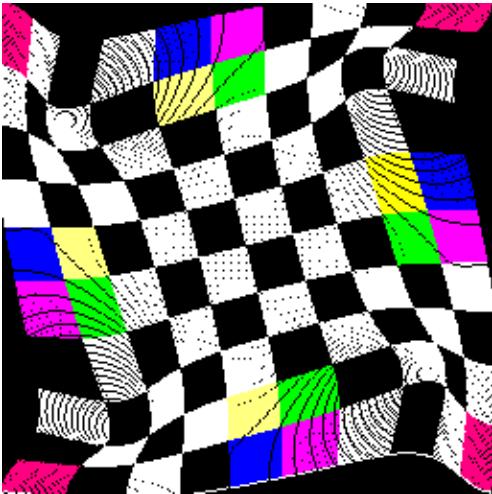
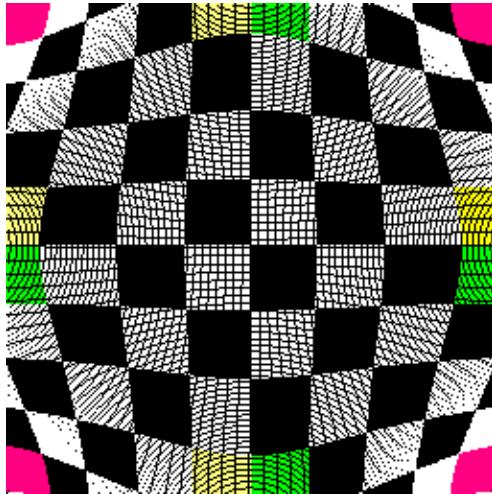
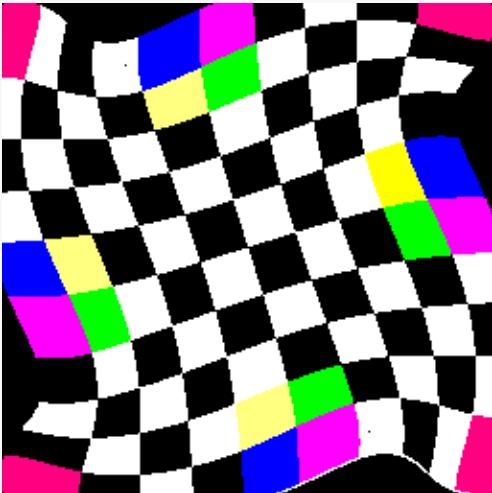
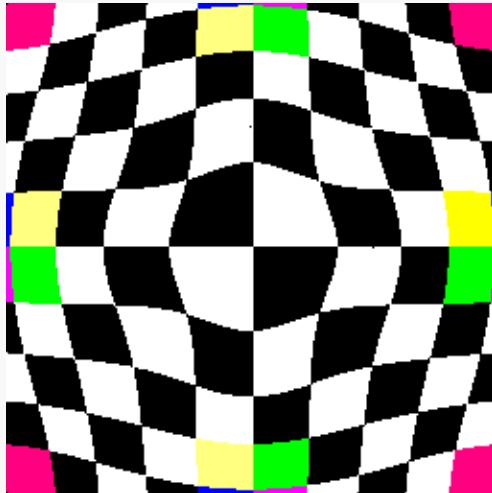
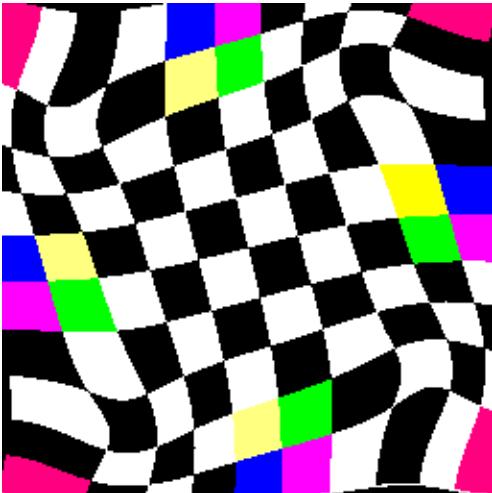
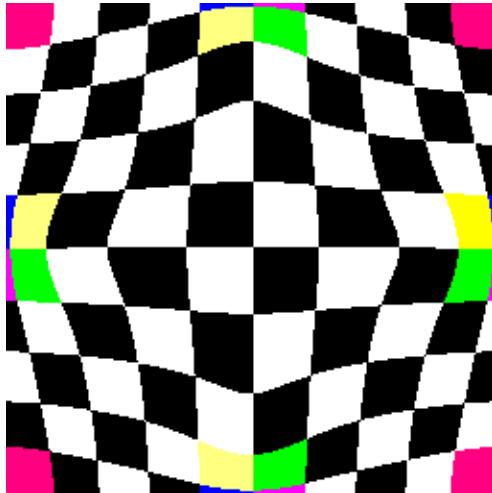
3. 补洞

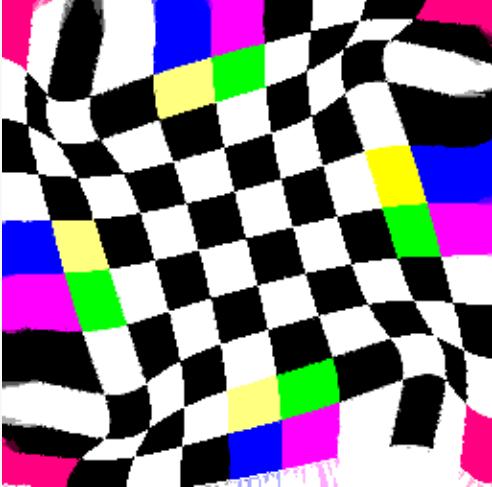
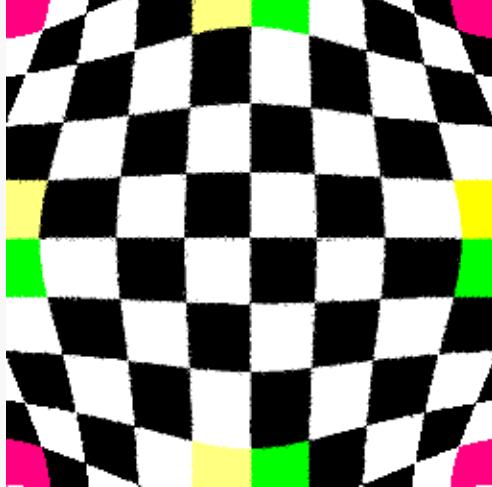
该算法的流程是对于每一个像素，寻找其映射后的新位置。因为新位置有可能重复或超出边界，所以映射后的图像中一定会存在未被映射的区域，如下图第一行所示。

助教的注释中描述的一种比较简单的方式是：互换控制点中映射与被映射的点来计算映射，以每个像素变形后映射后得到的坐标为采样坐标。这样就保证了图像变形后每个像素点都能找到其颜色值，且依然保证了要求的插值条件 $f(\mathbf{p}_i) = \mathbf{q}_i$ 。Optional 窗口中的 Reverse 即这种方法，如下图第二、三行所示。

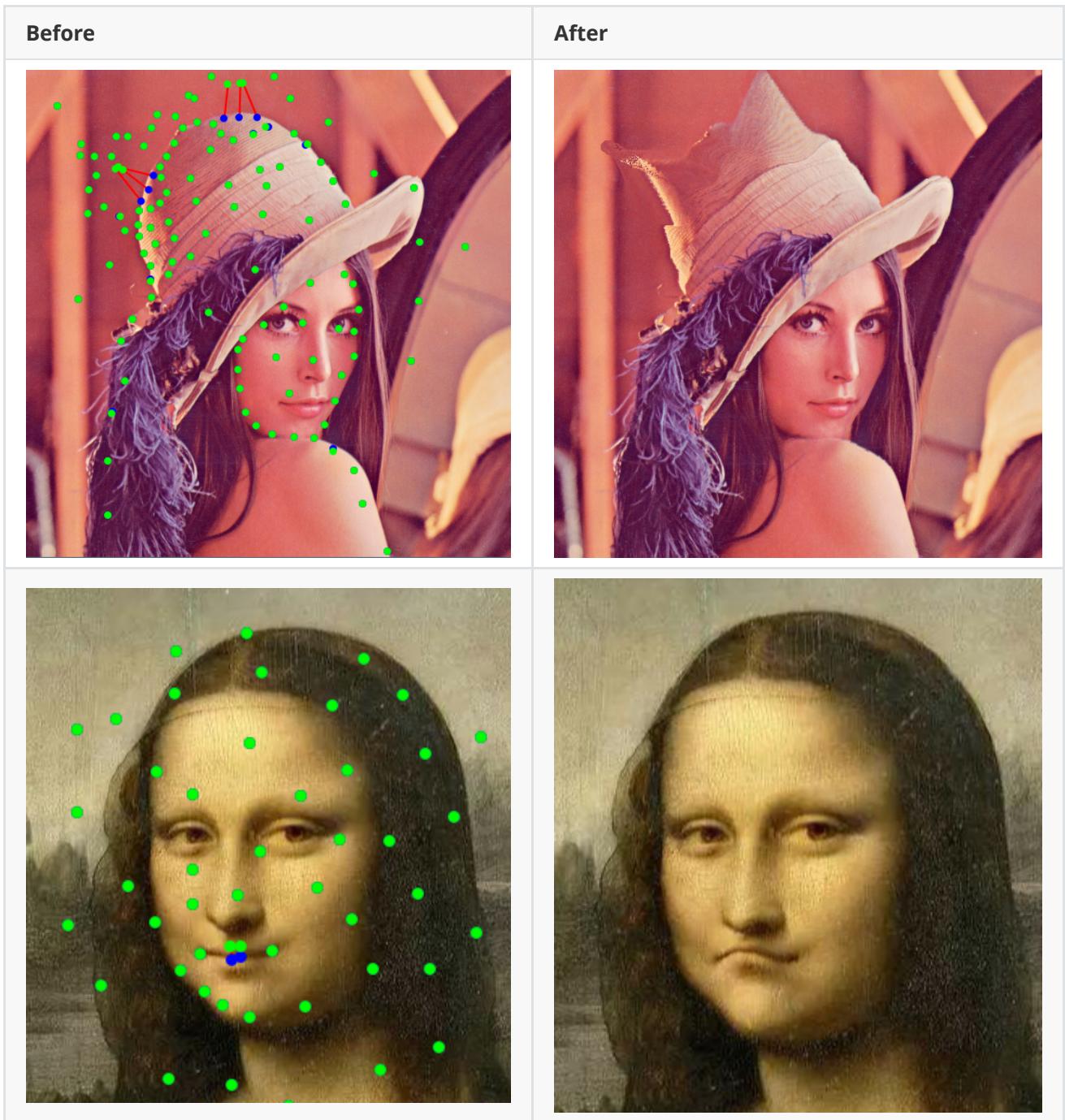
可以注意到其形变与补洞前的形变是有一定不同的，但控制点处都得到了正确的形变。

另一种方法是对于未被映射的点，寻找其最近的被映射的点，以其颜色值（或以一系列最近点颜色值的平均值）为该点的颜色值。如果手动枚举最近的被映射的点，该算法的时间复杂度将达到于图像处理不可接受的 $O(m^2n^2)$ ，因此我们需要一些空间划分数据结构来实现该目的。所幸的是，ANNoy 库为我们实现了该算法，只需安装并调用该库即可顺利完成该算法。Optional 窗口中的 ANN 即这种方法，效果如下图第四行所示（三个采样点平均）。有趣的是，ANNoy 库本身是 Spotify（音乐软件）为用户推送歌曲而设计的最近点查找库。在看到上传者是 Spotify，以及了解该用途的后感觉到很不可思议。

算法	旋转	膨胀
None+IDW		
Reverse+IDW		
Reverse+RBF		

算法	旋转	膨胀
ANN+IDW		

4. 其他效果展示



心得体会

这次同样是二周目。一周目的时候没有实现 IDW 的能量最低（当时没看懂），在理解 RBF 算法上也磕磕绊绊了很久。直到做完了后面更难的部分，回头才发现这些其实已经没那么难了。

当时的算法参数也是在代码里写死的，每次查看效果都要重新编译一遍源码，很不方便。因此这次设计了 Options 窗口，以在程序中方便地调整参数。

当时还没有实现 ANN 的补洞。这次为了全收集所有 Optional 所以也一起完成了。其实这点倒远没有理解两种算法难了。

这次作业我还修改了助教的框架代码，设置了控制点的预设（写死在代码里了），从而能方便地进行测试。

部分向量运算我在 geometry.h 中手动封装了几个计算几何类 (Point, Vector, Matrix22) , 可以有效简化向量运算的代码。

不足之处在于这次二周目作业我依然没有认真去读两个算法的论文，只是简单地根据文档实现了算法。阅读英文论文对我来说仍然是相当恐怖的事情，以后也要慢慢学会适应了。以及，图像变形的参数全部存在了 Image 里，这使得每次重新打开图像都会重置一遍参数，在操作上略微不友好。

Visual Studio 里的类设计器不知道为什么检测不出来图像变形算法类。Windows 上又配不好 clang-uml，转到了 Linux 上才用上 clang-uml 生成了这次的类图。

这是本课程作业中“数学”的开始。一周目前曾经还怀疑过自己“能不能把这门课的数学啃下来”，做完 Homework 2 之后我逐渐开始相信自己了。遇到不会的数学就去学，带着问题去看，数学的学习也会有趣得多。同时从这里开始我也对计算机图形学有了更清晰的认知——不止是计算机，更是数学。