

# 数值分析大作业 1 实验报告

2015011506 金帆 自 56 班

## 目录

- 1 需求分析 ..... 2
  - 1-1 符号约定..... 2
  - 1-2 概述..... 2
  - 1-3 旋转扭曲变换 ..... 2
  - 1-4 水波纹扭曲变换..... 3
  - 1-5 B 样条变换..... 4
    - 1-5-1 一次 B 样条变换..... 4
    - 1-5-2 三次 B 样条变换..... 5
  - 1-6 最近邻插值 ..... 5
  - 1-7 双线性插值 ..... 5
  - 1-8 双三次插值 ..... 6
- 2 方案设计 ..... 7
  - 2-1 编程语言及环境..... 7
  - 2-2 系统模块图、类图 ..... 8
  - 2-3 必做任务的反变换求解（解析法） ..... 9
  - 2-4 选做任务的反变换求解（迭代法） ..... 9
    - 2-4-1 B 样条正变换的性质 ..... 9
    - 2-4-2 B 样条逆变换的存在性..... 10
    - 2-4-3 迭代法求解逆变换 ..... 10
  - 2-5 插值..... 11
- 3 误差分析 ..... 11
  - 3-1 观测误差..... 11
  - 3-2 舍入误差..... 11
  - 3-3 截断误差（迭代法） ..... 11
  - 3-4 方法误差（插值） ..... 12
- 4 结果展示与参数讨论 ..... 16
  - 4-1 旋转扭曲变换 ..... 16
  - 4-2 水波纹扭曲变换..... 19
  - 4-3 B 样条变换..... 21
- 参考文献..... 24

# 1 需求分析

## 1-1 符号约定

首先约定，以下对于像素点坐标的描述 $(x, y)$ ，其中 $x$ 表示像素点所在的行数， $y$ 表示像素点所在的列数。假设图像有 $X$ 横行、 $Y$ 纵列，则图像左上角、右上角、左下角、右下角的坐标分别为 $(0, 0)$ 、 $(0, Y - 1)$ 、 $(X - 1, 0)$ 、 $(X - 1, Y - 1)$ 。

## 1-2 概述

总体目标是，编写图像扭曲变形程序，可以对图像进行扭曲变形。

图像扭曲的本质是一个坐标到坐标的变换，即一个 $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ 的映射 $f$ 。给定原图，在求取扭曲后的图像时，我们的目标是，对于扭曲后图像的每一个像素点 $(i, j)$ ，寻找原图中对应点的坐标 $(x, y)$ ，使得 $f(x, y) = (i, j)$ ，即求解映射 $f$ 的逆映射。

一般地，通过解析法或者迭代法求得的 $(x, y)$ 不是整数点，而原图是离散的，只有整数点处才有值，因而必须对原始图像进行插值，用插值结果作为原图 $(x, y)$ 处的像素值，也就是扭曲后图像 $(i, j)$ 处的值。本次作业，我们尝试使用三种插值方式：最近邻、双线性、双三次。

## 1-3 旋转扭曲变换

对于行数与列数分别为 $X$ 与 $Y$ 的图像，其图像中心点的坐标是 $(\frac{X-1}{2}, \frac{Y-1}{2})$ ，旋转半径 $R = \min(\frac{X-1}{2}, \frac{Y-1}{2})$ 。点 $(x, y)$ 相对于图像中心的极坐标表示记为 $(r, \alpha)$ ，则

$$x = r \cdot \cos(\alpha) + \frac{X - 1}{2}$$

$$y = r \cdot \sin(\alpha) + \frac{Y - 1}{2}$$

假设变换后的极坐标表示为 $(r^*, \alpha^*)$ ，则变换方程为：

$$\begin{cases} r^* = r \\ \alpha^* = \alpha + \frac{\theta(R-r)}{R} \cdot I(r < R) \end{cases}$$

其中 $I(\cdot)$ 是示性函数， $\theta$ 是参数，控制旋转的方向和角度。

其反变换为

$$\begin{cases} r = r^* \\ \alpha = \alpha^* - \frac{\theta(R-r^*)}{R} \cdot I(r^* < R) \end{cases}$$

设变换后的点的直角坐标是 $(i, j)$ ，则有

$$\begin{aligned} r^* &= \sqrt{\left(i - \frac{X-1}{2}\right)^2 + \left(j - \frac{Y-1}{2}\right)^2} \\ \alpha^* &= \text{atan2}\left(j - \frac{Y-1}{2}, i - \frac{X-1}{2}\right) \end{aligned}$$

这样给定 $(i, j)$ ，先计算 $(r^*, \alpha^*)$ ，再计算 $(r, \alpha)$ ，得到变换前的直角坐标 $(x, y)$ 。这样得到的 $(x, y)$ 不是整数点，因此需要使用插值方法，将插值结果作为变换后图像的 $(i, j)$ 处的值。

这一过程对 RGB 三个通道遍历，并对新图像的每一个像素点 $(i, j)$ 遍历。

## 1-4 水波纹扭曲变换

直角坐标和极坐标转换的部分与 1-3 节的旋转扭曲相同。只是，我们将正向变换公式变形为

$$\begin{cases} r^* = r \\ \alpha^* = \alpha + \theta \sin\left(\frac{r}{R}\rho + \phi\right) \end{cases}$$

其中 $\theta$ 、 $\rho$ 、 $\phi$ 是参数， $\theta > 0$ 控制水波纹的幅度， $\rho > 0$ 控制波长， $\phi$ 控制相位。

反向变换公式相应变为

$$\begin{cases} r = r^* \\ \alpha = \alpha^* - \theta \sin(\frac{r^*}{R} \rho + \phi) \end{cases}$$

其余步骤与 1-3 节旋转扭曲相同，此处不再赘述。需要注意的是，由于此处不再有  $I(r < R)$  的示性函数，因此变换后的  $(x, y)$  可能超出了图像范围。这时我们的处理是，令  $x' = \max(0, \min(X - 1, x))$ ， $y' = \max(0, \min(Y - 1, y))$ 。

## 1-5 B 样条变换

在原图等距选取一些控制点，组成阵列。在  $x$  方向相邻控制点间隔  $N_x$  像素，在  $y$  方向相邻控制点间隔  $N_y$  像素。我们假设图像边长是上述间隔的整数倍加 1，这样保证图像的四个角上的像素一定是控制点。

每个控制点的位置都可以在图像范围内拖动。记  $\Delta P_x(i, j)$  表示  $x$  方向第  $i$  个、 $y$  方向第  $j$  个控制点在  $x$  方向的位移， $\Delta P_y(i, j)$  表示  $x$  方向第  $i$  个、 $y$  方向第  $j$  个控制点在  $y$  方向的位移。该控制点在原图中的原始坐标应为  $(N_x i, N_y j)$ ，拖动后的坐标变为

$$(N_x i + \Delta P_x(i, j), N_y j + \Delta P_y(i, j))$$

### 1-5-1 一次 B 样条变换

对于原图坐标为  $(x, y)$  的像素点，一次 B 样条变换后，其位移如下：

$$\begin{aligned} v_x(x, y) &= \sum_{l=0}^1 \sum_{m=0}^1 G_{l,1}(u) \cdot G_{m,1}(v) \cdot \Delta P_x(i + l, j + m) \\ v_y(x, y) &= \sum_{l=0}^1 \sum_{m=0}^1 G_{l,1}(u) \cdot G_{m,1}(v) \cdot \Delta P_y(i + l, j + m) \end{aligned}$$

其中，基函数定义由作业要求给出，

$$u = \frac{x}{N_x} - \left\lfloor \frac{x}{N_x} \right\rfloor, \quad v = \frac{y}{N_y} - \left\lfloor \frac{y}{N_y} \right\rfloor, \quad i = \left\lfloor \frac{x}{N_x} \right\rfloor, \quad j = \left\lfloor \frac{y}{N_y} \right\rfloor$$

可以看到，一次 B 样条使用了该点“周围”最近邻的 4 个控制点的信息，是这 4 个控制点的位移的线性组合。这里的“周围”是在控制点位置没有移动时定义的。

### 1-5-2 三次 B 样条变换

对于原图坐标为  $(x, y)$  的像素点，三次 B 样条变换后，其位移如下：

$$v_x(x, y) = \sum_{l=0}^3 \sum_{m=0}^3 G_{l,3}(u) \cdot G_{m,3}(v) \cdot \Delta P_x(i + l, j + m)$$

$$v_y(x, y) = \sum_{l=0}^3 \sum_{m=0}^3 G_{l,3}(u) \cdot G_{m,3}(v) \cdot \Delta P_y(i + l, j + m)$$

其中，基函数定义由作业要求给出，

$$u = \frac{x}{N_x} - \left\lfloor \frac{x}{N_x} \right\rfloor, \quad v = \frac{y}{N_y} - \left\lfloor \frac{y}{N_y} \right\rfloor, \quad i = \left\lfloor \frac{x}{N_x} \right\rfloor - 1, \quad j = \left\lfloor \frac{y}{N_y} \right\rfloor - 1$$

可以看到，三次 B 样条使用了该点“周围”最近邻的 16 个控制点的信息，是这 16 个控制点的位移的线性组合。

由于使用了取整运算，B 样条变换的逆变换没有解析解。之后我们采用迭代法，以求取其逆变换的数值解。

### 1-6 最近邻插值

对于非整数坐标  $(x, y)$ ，使用与它距离最近的整点处的值作为插值结果。

### 1-7 双线性插值

对于非整数坐标  $(x, y)$ ，定义

$$u = x - [x], \quad v = y - [y], \quad i = [x], \quad j = [y]$$

则插值结果

$$f(x, y) = f(u + i, v + j) = \begin{bmatrix} 1 - u & u \end{bmatrix} \begin{bmatrix} f(i, j) & f(i, j + 1) \\ f(i + 1, j) & f(i + 1, j + 1) \end{bmatrix} \begin{bmatrix} 1 - v \\ v \end{bmatrix}$$

直观来说，就是先在 $x$ 方向做两次线性插值，再对所得结果在 $y$ 方向做线性插值。

## 1-8 双三次插值

对于非整数坐标 $(x, y)$ ，定义

$$u = x - [x], \quad v = y - [y], \quad i = [x], \quad j = [y]$$

则插值结果

$$f(x, y) = f(u + i, v + j) = \sum_{l=0}^3 \sum_{m=0}^3 a_{lm} \cdot u^l \cdot v^m$$

其中系数 $a_{lm}$ 通过以下矩阵运算得到：

$$\begin{bmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{bmatrix} = A^{-1} \begin{bmatrix} f(i, j) \\ f(i + 1, j) \\ f(i, j + 1) \\ f(i + 1, j + 1) \\ f_x(i, j) \\ f_x(i + 1, j) \\ f_x(i, j + 1) \\ f_x(i + 1, j + 1) \\ f_y(i, j) \\ f_y(i + 1, j) \\ f_y(i, j + 1) \\ f_y(i + 1, j + 1) \\ f_{xy}(i, j) \\ f_{xy}(i + 1, j) \\ f_{xy}(i, j + 1) \\ f_{xy}(i + 1, j + 1) \end{bmatrix}$$

其中系数矩阵为（图片来自 [https://en.wikipedia.org/wiki/Bicubic\\_interpolation](https://en.wikipedia.org/wiki/Bicubic_interpolation) ）

$$A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{bmatrix}$$

微分算子的定义为：（误差分析参见 3-4-3 节）

$$f_x(x, y) \approx \frac{1}{2}[f(x+1, y) - f(x-1, y)]$$

$$f_y(x, y) \approx \frac{1}{2}[f(x, y+1) - f(x, y-1)]$$

$$f_{xy}(x, y) \approx \frac{1}{2}[f_x(x, y+1) - f_x(x, y-1)]$$

## 2 方案设计

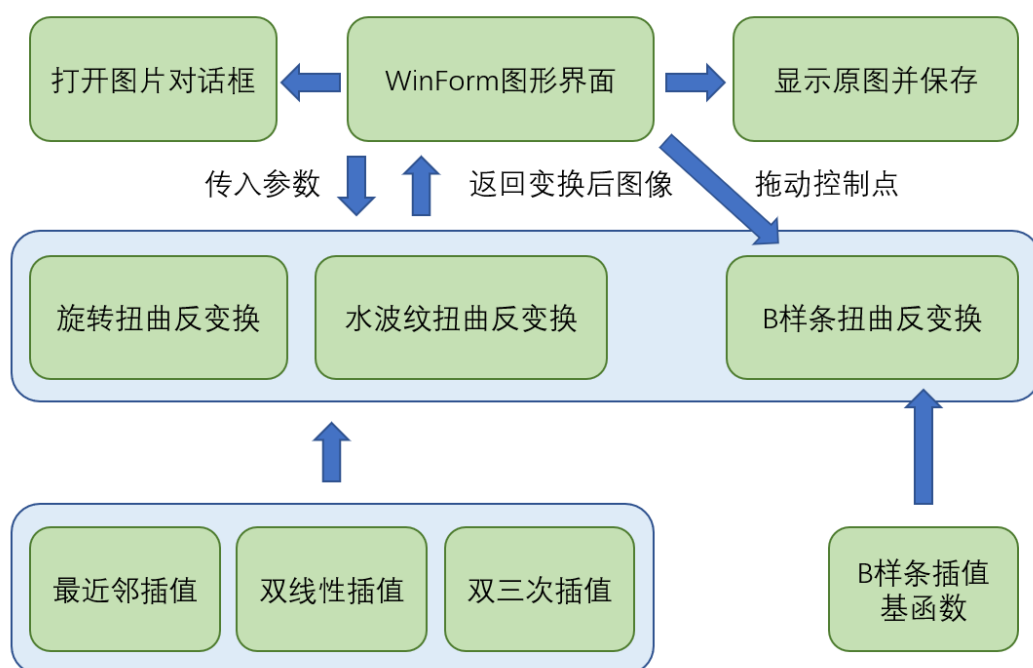
### 2-1 编程语言及环境

采用 C#（.NET Framework 4.5.2），在 Visual Studio 2015 下开发，可执行文件可在 Windows 10 环境下直接运行，无需安装第三方库。

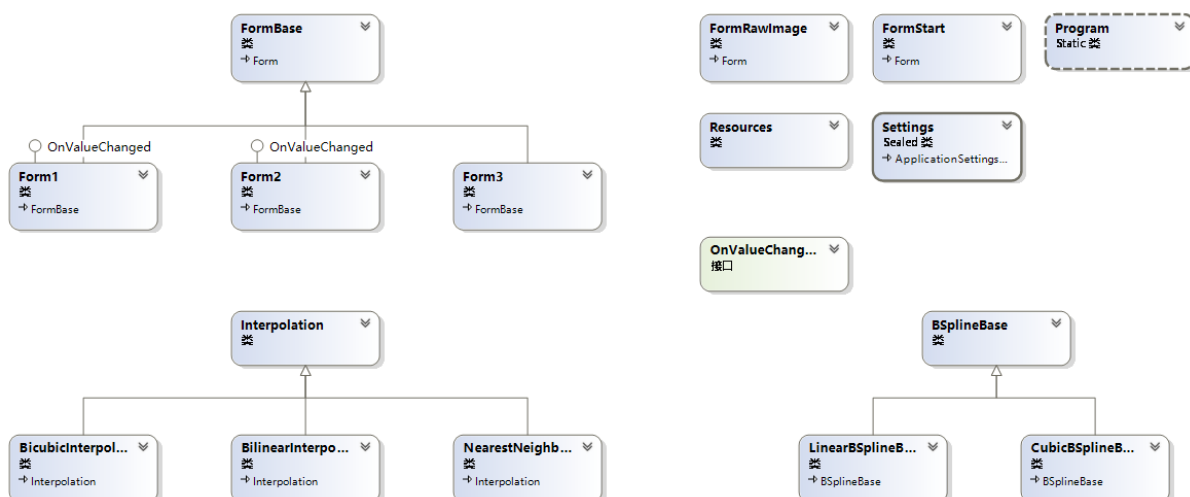
图像读写使用了 .NET 框架内置的 `System.Drawing.Bitmap` 类，为提高性能，绕过了 `Bitmap` 类提供的 `SetPixel` 和 `GetPixel` 方法，使用不安全的指针直接操作内存。已在编译选项中开启了“允许不安全代码”，以支持 `unsafe` 语句。

程序自带一个 WinForm 界面，不提供命令行接口。在 Visual Studio 中已对工程的输出做了重定向，生成的 exe 文件就在根目录下的 exe 文件夹内。

## 2-2 系统模块图、类图



我们使用面向对象（OOP）的思想编写程序。3 个任务分别对应 3 个窗体，由于有较多的共有部分，共有部分写为 **FormBase** 类中，其被 3 个任务的窗体分别继承，并加入各自的参数控件，并重写反变换算法。插值被单独做成一个类，三种插值分别是其一个子类，接口统一，这样上层就无需关心具体是哪种插值的实现。同理，一阶和三阶的 B 样条变换同样也采用父类-子类的继承关系。





## 2-3 必做任务的反变换求解（解析法）

在 1-3 节和 1-4 节，我们已经给出了必做任务的反变换的解析公式。程序从参数控件上获取到当前的参数，代入解析公式，对每个 $(i, j)$ 像素点，反变换得出对应的坐标 $(x, y)$ 。由于这里的坐标 $(x, y)$ 不是整数点，需要调用插值模块，将插值结果作为变换后图像的 $(i, j)$ 位置的值。对于 RGB 图像，每个通道都如此操作。

## 2-4 选做任务的反变换求解（迭代法）

B 样条正变换的公式（参见 1-5 节） $f(x, y) = (i, j)$ 比较复杂，由于取整运算，其逆变换 $f^{-1}(i, j) = (x, y)$ 的解析公式不易求得。

### 2-4-1 B 样条正变换的性质

显然，B 样条正变换公式 $f(x, y)$ 在 $x$ 和 $y$ 均是非整数时是连续的。

注意到一阶 B 样条基函数具有以下性质：

$$\begin{aligned} G_{l,m}(1) &= G_{l+1,m}(0) \\ G_{0,m}(0) &= G_{m,m}(1) = 0 \end{aligned}$$

因此，展开和式后，发现 $f(x, y) = (i, j)$ 在当 $x$ 或 $y$ 跨越整数时，仍然连续的。

综上，一阶 B 样条正变换是一个 $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ 的连续映射。对于三阶 B 样条变换，基函数的一阶、二阶导数也有类似性质，使得三阶 B 样条变换是二次连续可微的。

又由于基函数是有界的，假设其绝对值的界是 $T$ ，则对 1-5 节的公式进行放缩：

$$\begin{aligned} |v_x(x, y)| &\leq (L + 1)(M + 1)T^2 \cdot \max(|\Delta P_x(i + l, j + m)|) \\ |v_y(x, y)| &\leq (L + 1)(M + 1)T^2 \cdot \max(|\Delta P_y(i + l, j + m)|) \end{aligned}$$

也是有界的，因此 B 样条正变换满足 $\|f(x, y) - (x, y)\|$ 有界。

## 2-4-2 B 样条逆变换的存在性

此部分不做要求，此处略去，以下假定逆变换存在且唯一。

## 2-4-3 迭代法求解逆变换

在本小节中，以符号 $P$ 表示一个点的坐标的二元数组 $(x, y)$ ，经过 B 样条正变换后坐标变为 $f(P)$ 。逆变换的本质是，给定 $P_0$ ，求取 $P_\infty$ 使得 $f(P_\infty) = P_0$ 。算法如下：

输入：  $P_0$ （二元数组），映射 $f$

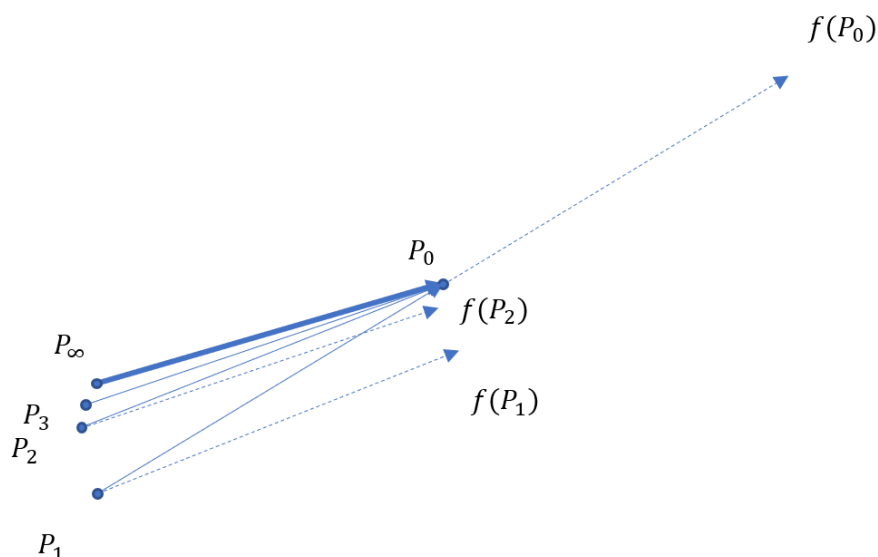
输出：  $P_\infty$ 使得 $f(P_\infty) \approx P_0$

步骤：

for  $i = 1, 2, \dots, \text{MaxIteration}$

    求解满足如下条件的 $P_i$ ： 向量 $P_0 - P_i = f(P_{i-1}) - P_{i-1}$

$P_\infty := P_{\text{MaxIteration}}$



假设得到的一系列点 $P_i$ 收敛至 $P_\infty$ 。注意到

$$|P_0 - f(P_{i-1})| = |P_i - P_{i-1}|$$

因此“二维点列 $\{P_i\}$ 收敛到 $P_\infty$ ”与“二维点列 $\{f(P_i)\}$ 收敛到 $P_0$ ”等价。

## 2-5 插值

插值算法见 1-6、1-7、1-8 节。

在插值时，图像的边缘处理比较麻烦。我们采取的办法是，累加的时候，对于下标超出了原图范围的项一律置零，相当于补上 0。

为了加快双三次插值的速度，在代码中部分运算被展开，使其差分的意义不容易看出，这是以牺牲可读性为代价的。

## 3 误差分析

### 3-1 观测误差

不属于本文讨论范围。

### 3-2 舍入误差

C#中的 `double` 类型，精度有 15-16 位有效数字，因此舍入误差可以忽略。在计算的最后一步，将 `double` 类型的数转成 0~255 范围的 `uint8`，舍入误差上界为 0.5。

### 3-3 截断误差（迭代法）

在 2-4-3 节中，迭代算法在给定的最大迭代次数后停止。为了便于控制精度，同时也为加快运算速度，我们在每次迭代后，比较当前的  $f(P_i)$  与目标点  $P_0$  的距离（以无穷范数度量，即  $\|f(P_i) - P_0\| := \max(|f_x(P_i) - P_{0x}|, |f_y(P_i) - P_{0y}|)$ ）；如果距离小于给定的误差界  $\epsilon$ ，则提前停止迭代。

经过试验，在每一个像素点上，我们都提前停止了迭代。因此，我们下式满足：

$$\|f(P) - P_0\| < \epsilon$$

其中 $P$ 是我们算法的输出结果，进而，其与 $P_\infty$ 的误差满足：

$$\|P - P_\infty\| = \|P - f^{-1}(P_0)\| < M_0 \cdot \epsilon$$

其中 $M_0 = \max(\max(\left\|\frac{d}{dx}f^{-1}(P_j)\right\|), \max(\left\|\frac{d}{dy}f^{-1}(P_j)\right\|))$ ， $P_j$ 在 $P_\infty$ 的某个邻域内。

这样在无穷范数意义下，截断误差的上界是 $M_1\epsilon$ 。

### 3-4 方法误差（插值）

在本节中，约定符号 $z(x, y)$ 表示一个 $\mathbb{R}^2 \rightarrow \mathbb{R}$ 的映射。当 $x$ 与 $y$ 是整数时， $z(x, y)$ 表示原图在该位置的某个通道的取值（0~255）。 $z(x, y)$ 就是未知的被插值函数。

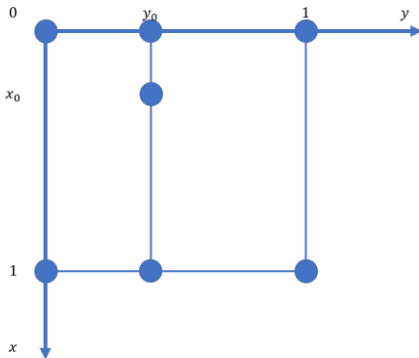
#### 3-4-1 最近邻插值

最近邻插值的 $x$ 方向与 $y$ 方向是解耦的。我们假定， $z(x, y)$ 在 $x$ 方向和 $y$ 方向上的一阶导数存在且有界 $M_1$ 。除去物体边缘等梯度大的区域外， $M_1$ 都很小；在图像中一些梯度大的地方（例如物体轮廓线）， $M_1$ 较大。

$x$ 方向与 $y$ 方向的误差上界各为 $\frac{1}{2}M_1$ ，总的方法误差上界为 $M_1$ 。

#### 3-4-2 双线性插值

不失一般性，假设求值点 $(x_0, y_0)$ 位于单位正方形内，4个顶点是插值点。



假设 $z(x, y)$ 在 $x$ 方向和 $y$ 方向的二阶导数存在且有界 $M_2$ 。

首先考察 $(0, y_0)$ 与 $(1, y_0)$ 处的插值结果的误差界。这相当于在直线 $x = 0$ 和 $x = 1$ 上做两次 $y$ 方向的线性插值，因此误差界是

$$|z(0, y_0) - z^*(0, y_0)| \leq \frac{1}{2} M_2 \cdot \max(x(1-x)) = \frac{1}{8} M_2$$

$$|z(1, y_0) - z^*(1, y_0)| \leq \frac{1}{2} M_2 \cdot \max(x(1-x)) = \frac{1}{8} M_2$$

然后给定 $z(0, y_0)$ 与 $z(1, y_0)$ ，考察 $(x_0, y_0)$ 处的插值结果的误差界。这相当于在直线 $y = y_0$ 做一次 $x$ 方向的线性插值，因此相对于给定的 $z(0, y_0)$ 与 $z(1, y_0)$ ，误差界是

$$|z(x_0, y_0) - z^*(x_0, y_0)|_{z(0, y_0), z(1, y_0)} \leq \frac{1}{2} M_2 \cdot \max(x(1-x)) = \frac{1}{8} M_2$$

再加上 $z(0, y_0)$ 与 $z(1, y_0)$ 的误差界，最坏情形下两种误差叠加，总误差为

$$|z(x_0, y_0) - z^*(x_0, y_0)| \leq \frac{1}{8} M_2 + \frac{1}{8} M_2 = \frac{1}{4} M_2$$

### 3-4-3 双三次插值

在使用双三次插值时，误差来自两方面：一是使用差分近似代替导数的误差，二是双三次插值本身的误差。在最坏情形下，两者叠加，

首先，假设插值函数 $z(x, y)$ 的二阶导数存在且有界，即

$$\left| \frac{\partial^2}{\partial x^2} z(x, y) \right| \leq M_2$$

$$\left| \frac{\partial^2}{\partial y^2} z(x, y) \right| \leq M_2$$

$$\left| \frac{\partial^2}{\partial x \partial y} z(x, y) \right| \leq M_2$$

同理假设 $z(x, y)$ 的三、四阶导数存在，且有界 $M_3$ 、 $M_4$ 。

首先分析差分代替导数的误差界。对于导数的估计，公式是（参加 1-8 节）：

$$f_x(x, y) \approx \frac{1}{2}[z(x+1, y) - z(x-1, y)]$$

$$f_y(x, y) \approx \frac{1}{2}[z(x, y+1) - z(x, y-1)]$$

$$f_{xy}(x, y) \approx \frac{1}{2}[z_x(x, y+1) - z_x(x, y-1)]$$

构造辅助函数 $z^*(x) = \frac{1}{2}[z(x+1) - z(x-1)]$ 。

由微分中值定理，存在 $x^* \in [x-1, x+1]$ 使得

$$z^*(x) = z'(x^*)$$

故有

$$|z^*(x) - z'(x)| = |z'(x^*) - z'(x)| \leq M_2 \cdot |x^* - x| \leq M_2$$

将这个结论应用到 1-8 节的 3 个公式，得到误差界

$$\left| f_x(x, y) - \frac{1}{2}[z(x+1, y) - z(x-1, y)] \right| \leq M_2$$

$$\left| f_y(x, y) - \frac{1}{2}[z(x, y+1) - z(x, y-1)] \right| \leq M_2$$

同理，根据辅助函数

$$z^*(x, y) = \frac{1}{4}[z(x+1, y+1) + z(x-1, y-1) - z(x+1, y-1) - z(x-1, y+1)]$$

可以得到误差界

$$\left| f_{xy}(x, y) - \frac{1}{2}[z_x(x, y+1) - z_x(x, y-1)] \right| \leq M_2 + M_3$$

第二部分的误差来自双三次插值本身。根据 Wikipedia 提供的公式，双三次插值

的结果可以使用以下的二次型表示：

$$p(x, y) = [1 \quad x \quad x^2 \quad x^3] P^T \cdot F \cdot P \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}$$

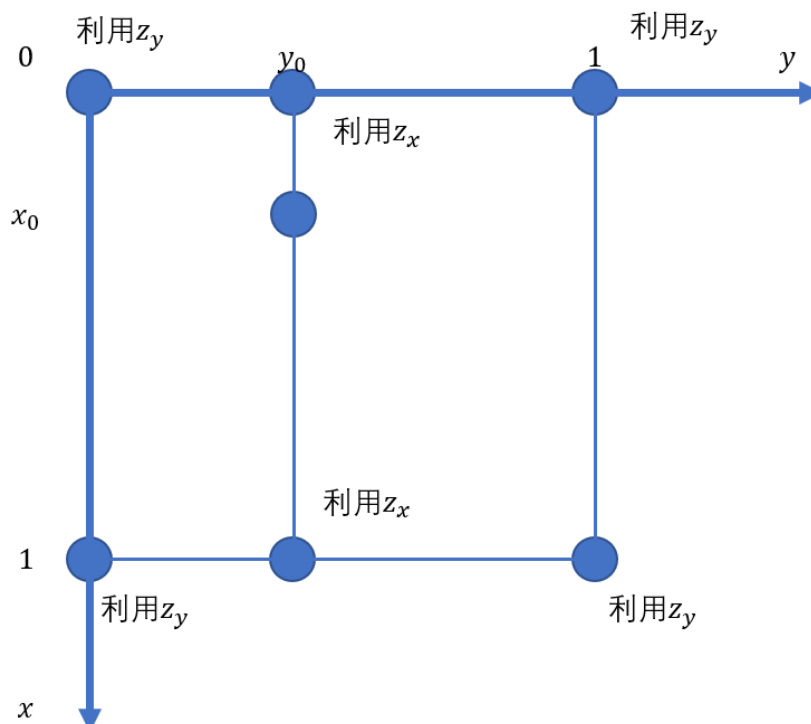
其中矩阵

$$P = \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

$$F = \begin{bmatrix} z(0, 0) & z(0, 1) & z_y(0, 0) & z_y(0, 1) \\ z(1, 0) & z(1, 1) & z_y(1, 0) & z_y(1, 1) \\ z_x(0, 0) & z_x(0, 1) & z_{xy}(0, 0) & z_{xy}(0, 1) \\ z_x(1, 0) & z_x(1, 1) & z_{xy}(1, 0) & z_{xy}(1, 1) \end{bmatrix}$$

如果记  $p_x = P \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix}$ ,  $p_y = P \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}$ , 则  $p(x, y) = p_x^T \cdot F \cdot p_y = p_x^T \cdot (F \cdot p_y)$ 。

由于矩阵乘法的结合律，可以将双三次插值视为先在  $y$  方向利用  $z$  和  $z_y$  信息进行 2 次三次 Hermite 插值，然后在  $x$  方向利用  $z_x$  和  $z_{xy}$  对结果做 1 次三次 Hermite 插值。



与 3-4-2 节同理，最坏情形下，插值的误差界是两方向的误差界之和，即

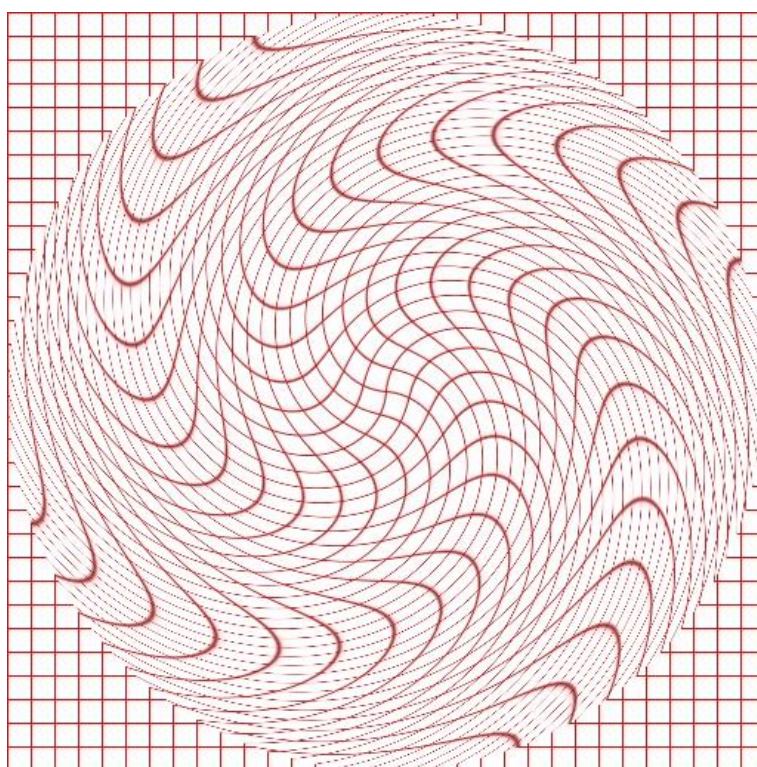
$$\max(|z(x,y) - f(x,y)|) \leq \frac{1}{384}M_4 + \frac{1}{384}M_4$$

综上，最坏情形下，双三次插值方法（包含导数的差分估计）误差界是两方面误差之和，即

$$\frac{1}{192}M_4 + M_2 + M_3$$

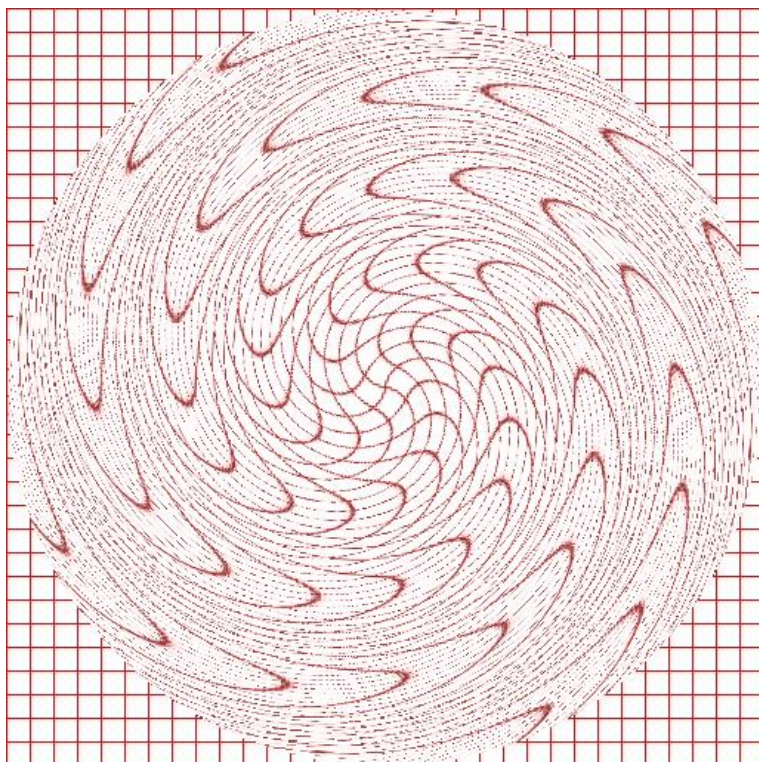
## 4 结果展示与参数讨论

### 4-1 旋转扭曲变换

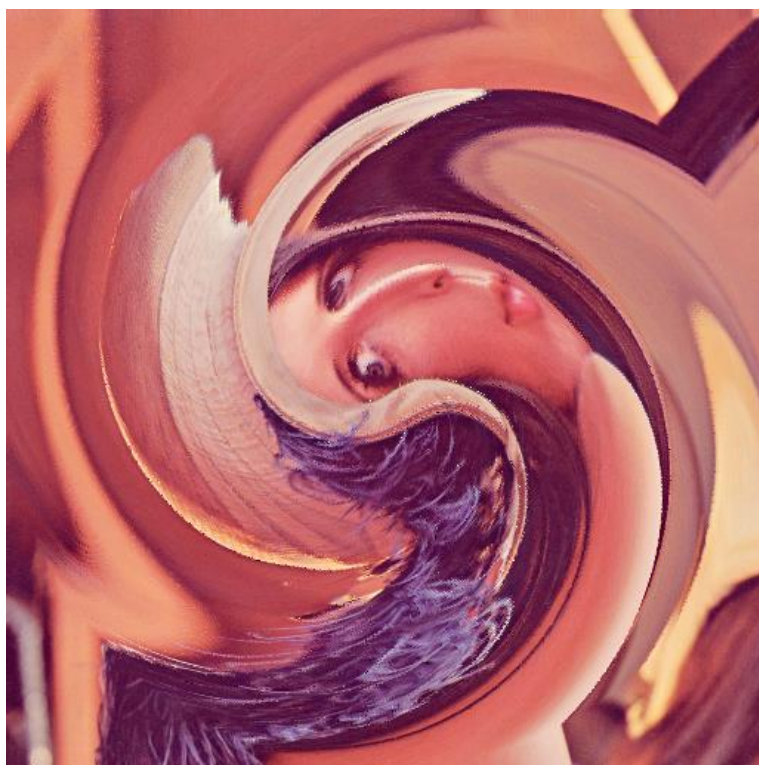


（双线性）



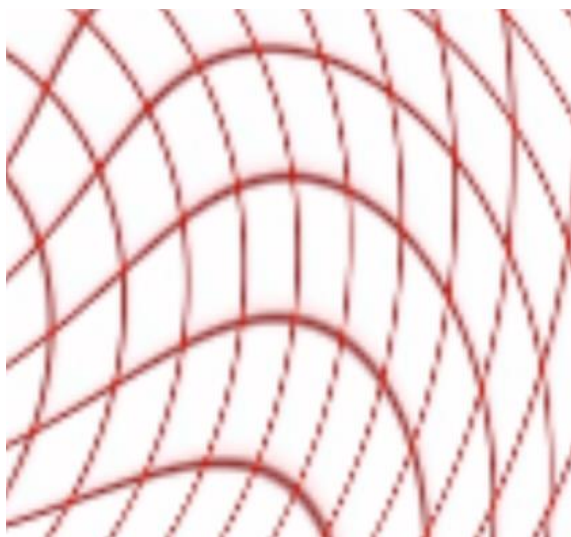


(最近邻)

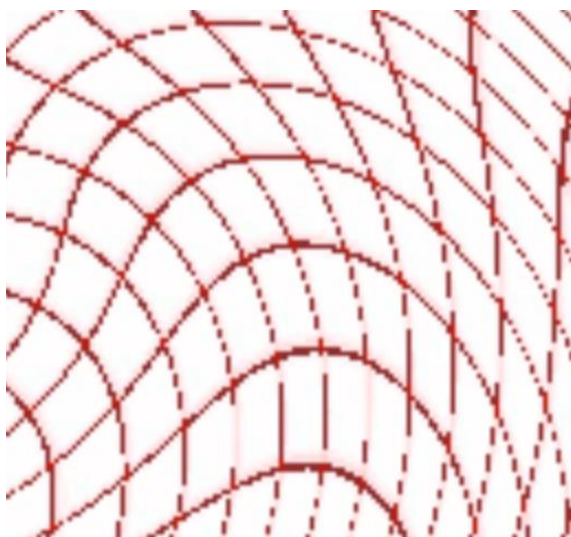


(双三次)

考察一些扭曲较大的局部，观察双线性插值相比最近邻插值的改进：



(双线性)



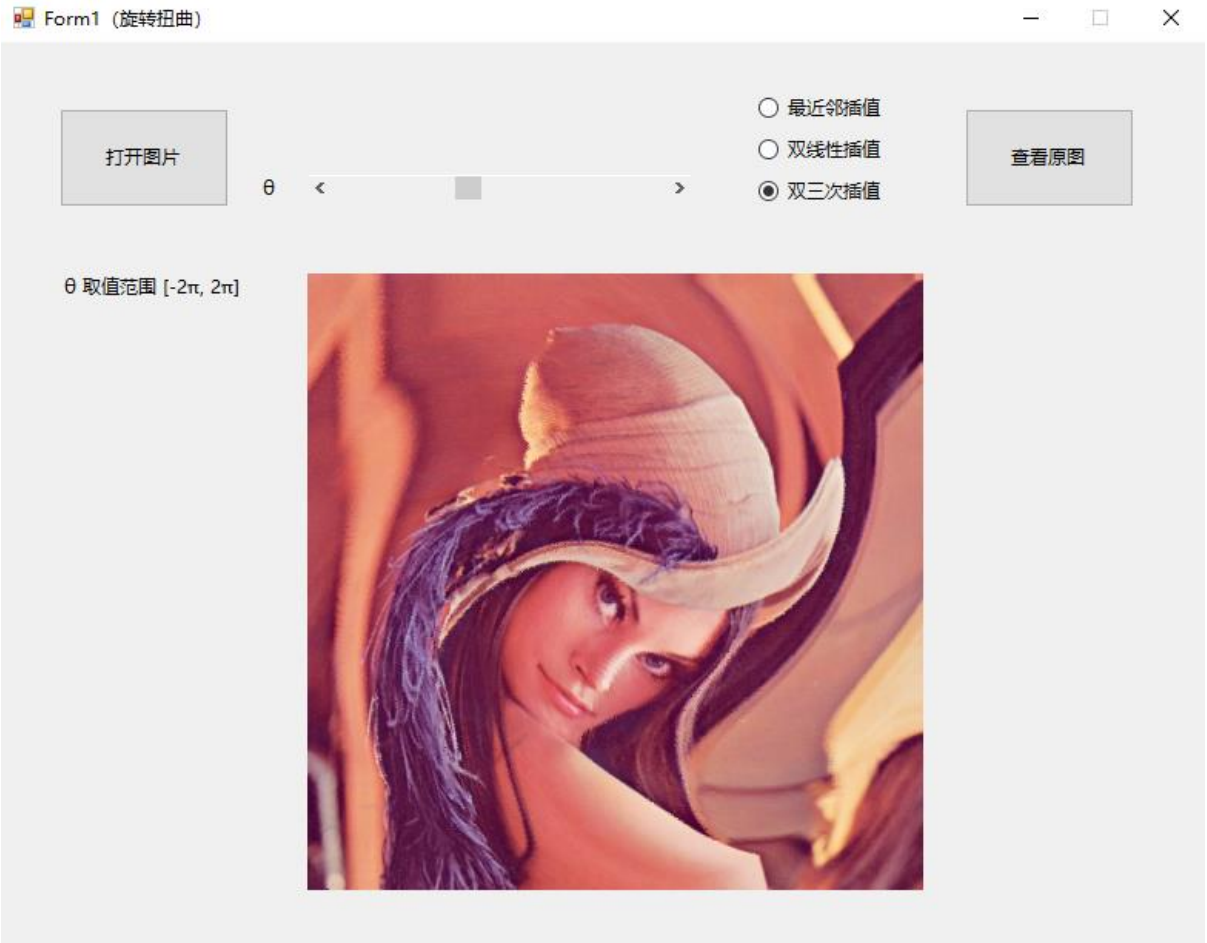
(最近邻)

最近邻插值的结果存在断续，而双线性的插值结果更平滑，在红色的线条和空白之间存在一些浅红色的渐变。这符合预期。

双三次的结果则更加平滑：



利用图形界面，可以打开更多的图片，并尝试改变旋转角度 $\theta$ ：



点击“查看原图”可以保存图片，并避免操作系统自带的 DPI 缩放机制。

## 4-2 水波纹扭曲变换

改变参数取值，有如下发现：

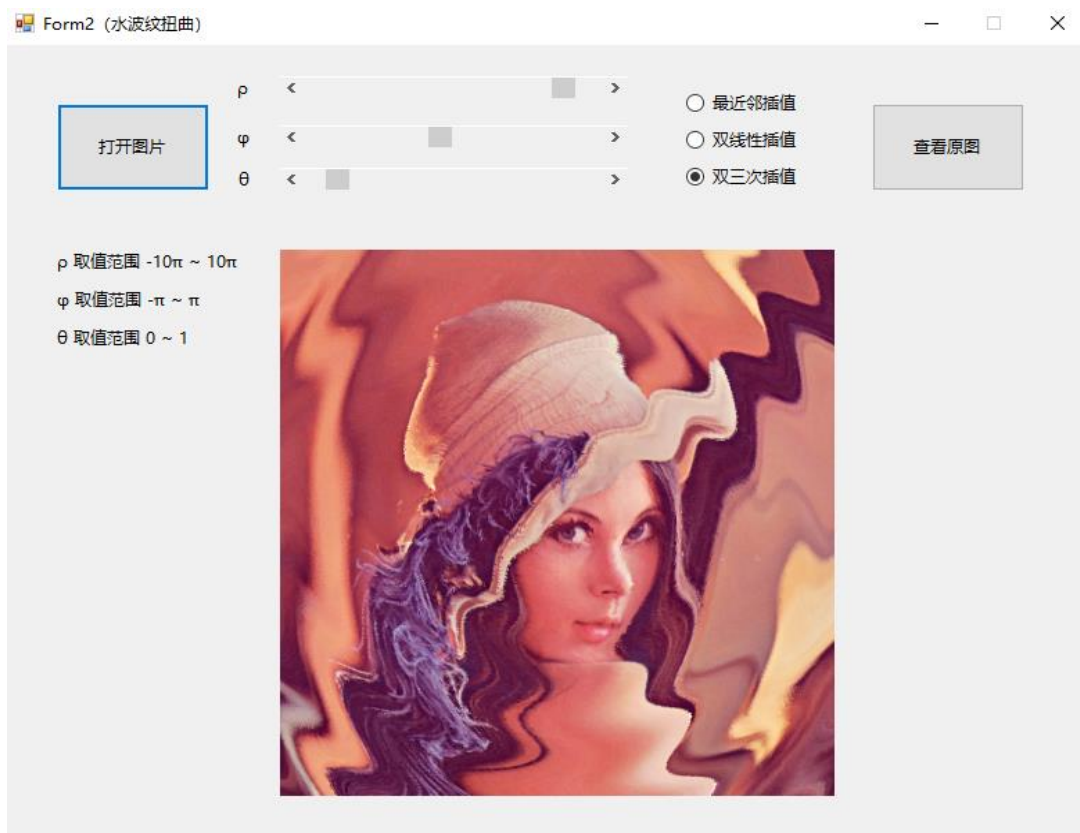
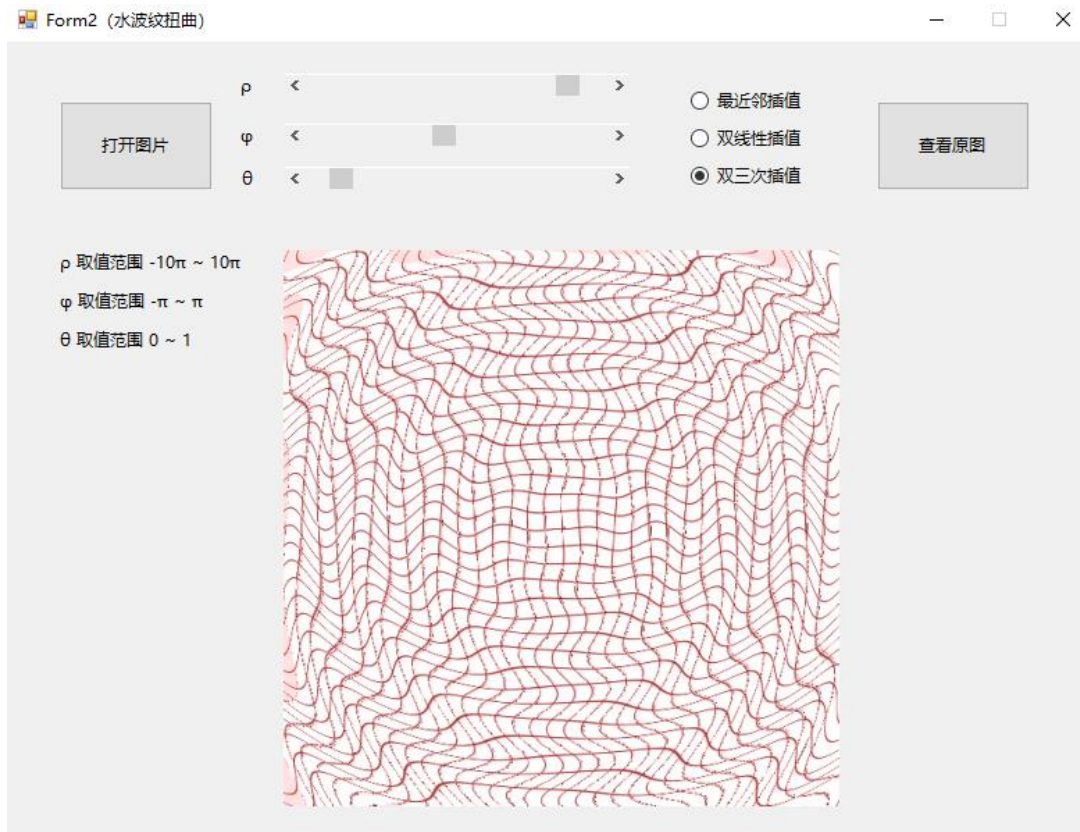
参数 $\rho$ 控制水波纹的波长，为了使波长较短，将 $\rho$ 调大，取值在 $5\pi$ 到 $10\pi$ 之间。

参数 $\phi$ 控制水波纹的相位，改变 $\phi$ 的时候，水波纹产生传播的效果。

参数 $\theta$ 控制水波纹的幅值，当 $\theta$ 较小（小于0.2）时，水波纹效果最清晰。

此组参数取值下，变换效果如下：

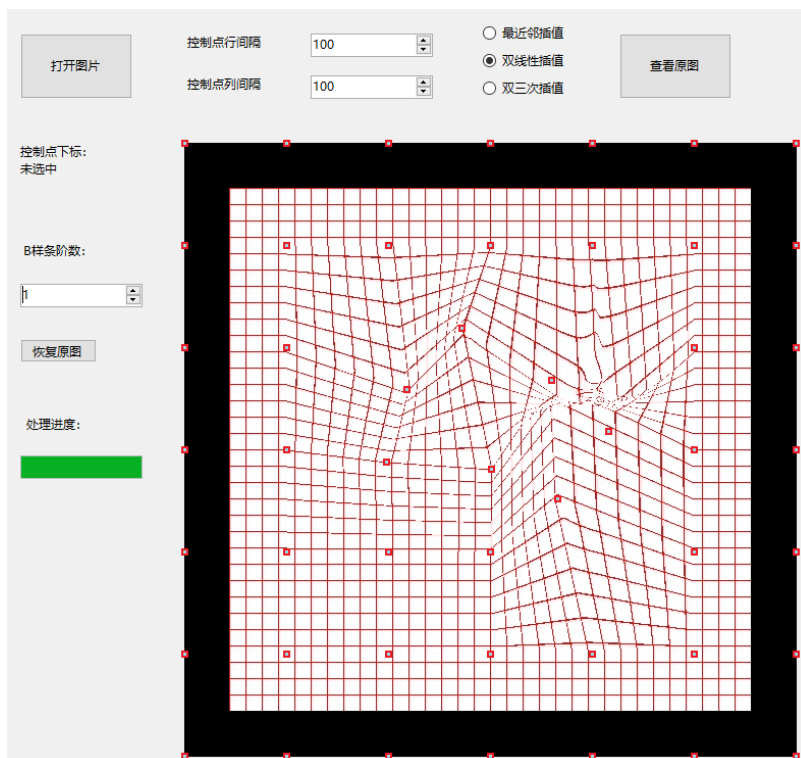
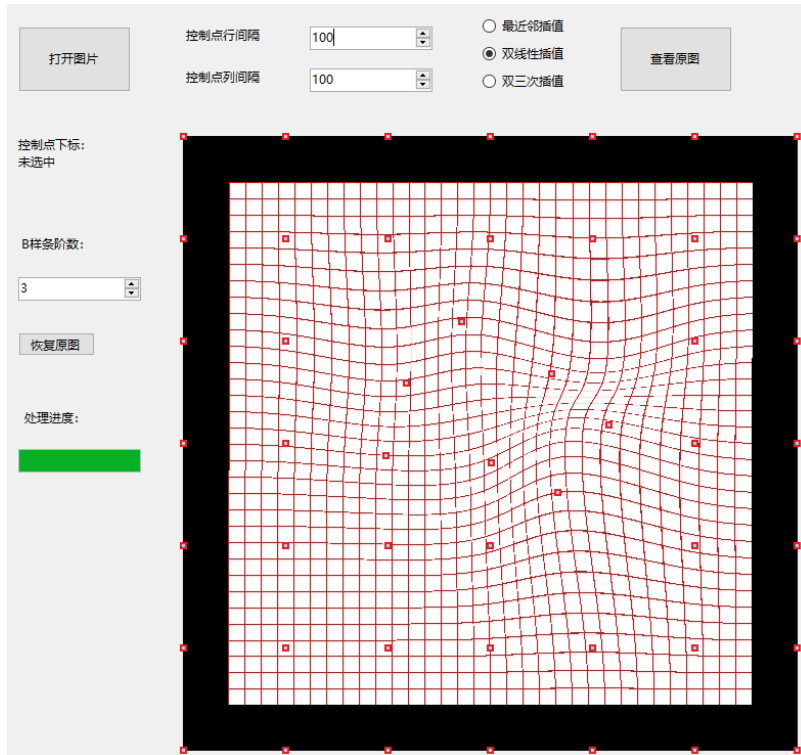


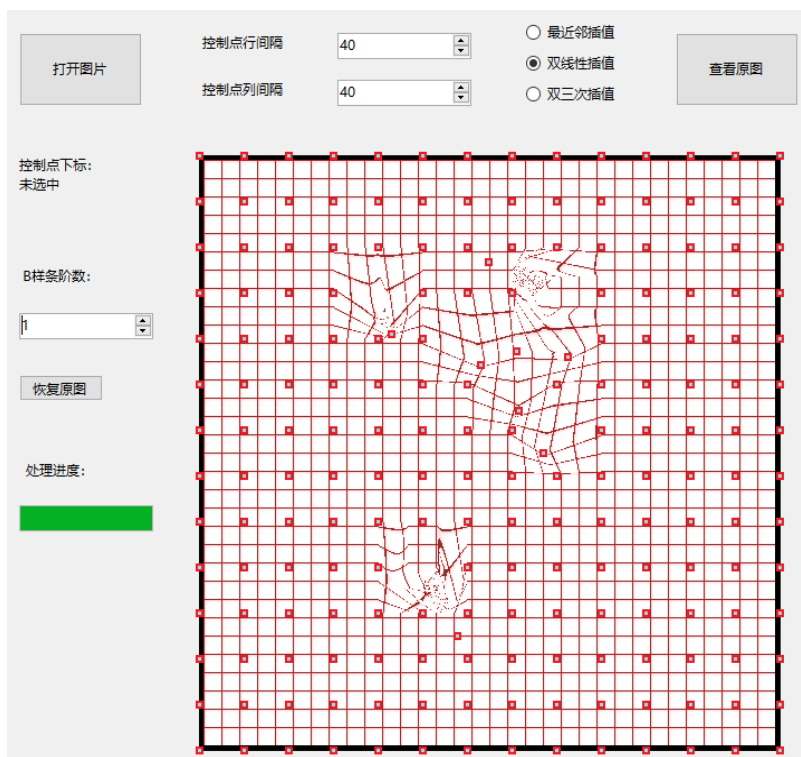
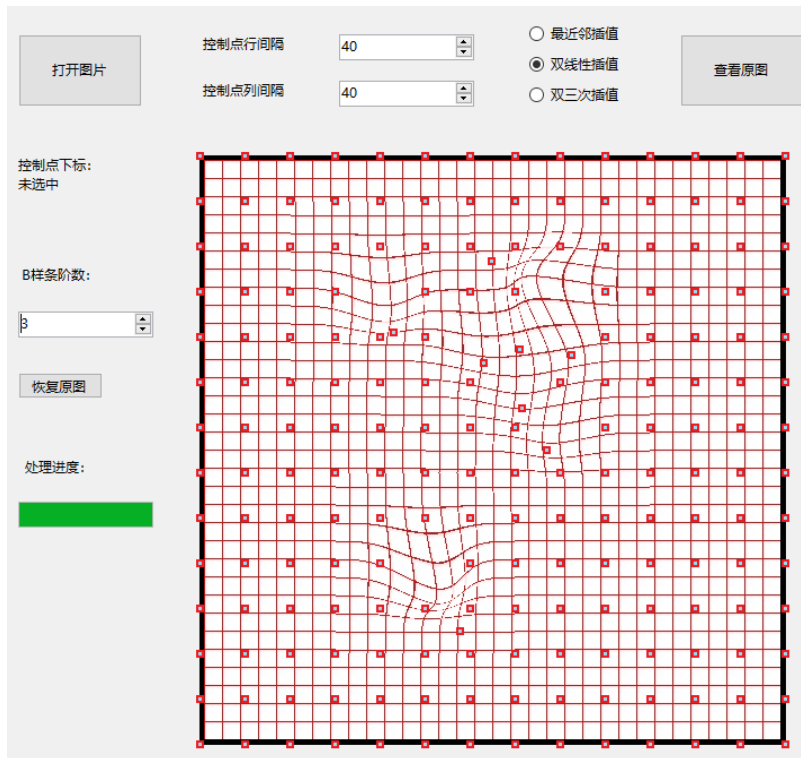


也可以尝试其他的参数，图像可能会被扭曲得难以辨认。

### 4-3 B 样条变换

B 样条的间隔 $N_x$ 和 $N_y$ 是可调整的。阶数也可调。我们尝试不同的值，观察效果：

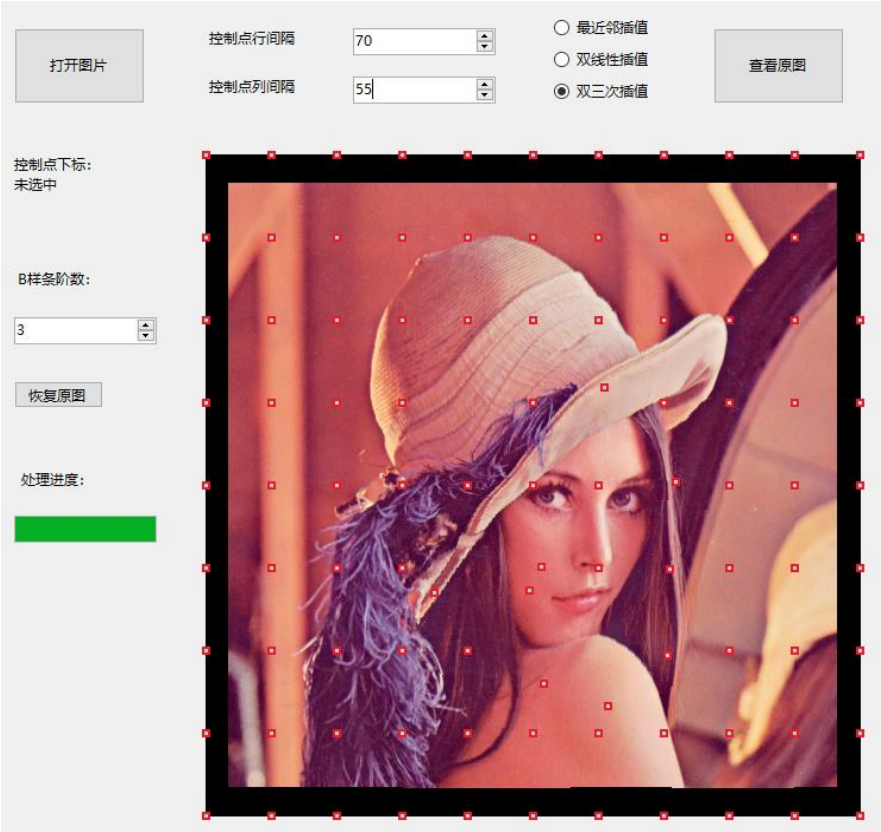




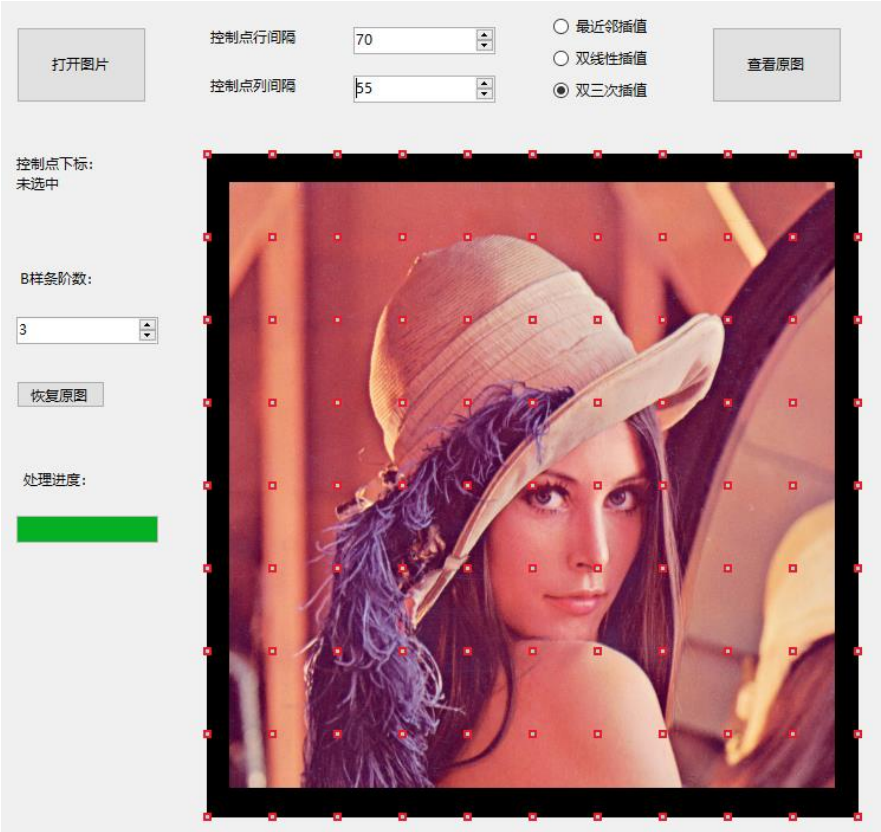
控制点间隔越大，某个控制点位移的影响范围也越大，但影响也更分散。

1 阶 B 样条的结果棱角分明，直线在变换后仍是直线；3 阶就平滑得多。

再拿 LENA 图作为测试：



(实验结果)



(原图)



## 5 遇到的问题及解决

**C#调用 Bitmap 类的接口太慢:**在 Stackoverflow 上查到了使用 C#的 unsafe 指针, 直接操作内存, 绕过那些影响效率的接口。(尽管这很不面向对象)

**水波纹扭曲效果与样例相差甚远:**经过研究, 我发现是我的参数选择不恰当。参数 $\rho$ 应当很大, 以将波长减小到屏幕能显示多个波峰。我还自己增加了一个参数 $\theta$ 以调节水波纹的振幅。经过这番操作后, 4-2 节效果好多了。

**B 样条的控制点不好拖动:**自己写了控件, 继承自 PictureBox, 添加鼠标事件, 来模拟用户的拖动。在程序运行时动态生成。

**双三次插值的误差分析:**经助教允许, 改用二次型形式的近似公式, 这样双三次插值的误差界, 可以转化为 2 个单三次插值的误差界。这样的误差估计比较粗糙, 不是很精确, 存在一些精度浪费, 但是能保证将误差控制住。

## 参考文献

李庆阳、王能超、易大义:《数值分析》第 5 版

[https://en.wikipedia.org/wiki/Bicubic\\_interpolation](https://en.wikipedia.org/wiki/Bicubic_interpolation)

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/double>

<https://msdn.microsoft.com/en-us/library/ms165366.aspx>

[https://msdn.microsoft.com/en-us/library/system.drawing.bitmap\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.drawing.bitmap(v=vs.110).aspx)

<https://stackoverflow.com/questions/13511661/create-bitmap-from-double-two-dimensional-array>