数值分析与算法大作业一
图像变形

自 72　高子靖　2017010917

2019 年 11 月 7 日

1　引言

本次数值分析大作业实现了图像变形的功能，具体实现了图像的扭曲变换和图像畸变，以及基于 TPS 薄板样条插值的人脸变形，插值函数分别使用了最近邻，双线性和双三次插值，本报告中将会对设计方法以及插值方法和误差分析进行详细的阐述。

2　需求分析

在必做任务中需要实现两部分内容，分别是图像的扭曲变换和畸变校正，对于每一种图像变换的方法，需要实现三种不同的函数进行插值，分别是最近邻插值，双线性插值，双三次插值方法。在选做任务中，需要对给出的九张图片利用 TPS 变形网络进行人脸变形，插值方法和必做任务中相同。除此之外，需要进行 UI 的设计来使用户可以使用上述方法完成的功能，具体来说需要实现的功能如下：

- 实现扭曲变换函数，畸变校正函数和 TPS 插值函数。
- 实现最近邻插值，双线性插值，双三次插值对像素进行插值。
- UI 中可读入图片并保存变换的结果图片。
- 对于旋转扭曲可供用户选择：旋转方向，旋转中心，最大旋转半径，旋转角度这些参数。
- 对于畸变校正可以试用户选择：畸变半径，畸变方式 (凹/凸)。
- 对于 UI 的容错设计。

其中的必做任务和选做任务的需求具体如下所示：

2.1 旋转扭曲和畸变校正

每一张数字图像都可以看做一张“网格”，而旋转扭曲和畸变校正则是对此网格进行坐标的映射，使得该网格发生扭曲变形，而格点上的像素值仍为之前的像素值，只是此时的格点已经并非处在旋转后的图像的网格格点上，因此需要利用已知的“非整数格点”的像素值来近似整数格点的像素值，实现如下所示的效果：

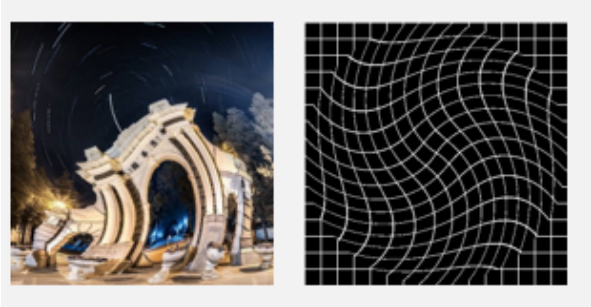

图 1: 旋转扭曲

畸变校正和上述的扭曲旋转同样是对于网格的变换，仅需要更改变形函数，不再赘述，需要实现以下的效果 (展示桶形畸变)：

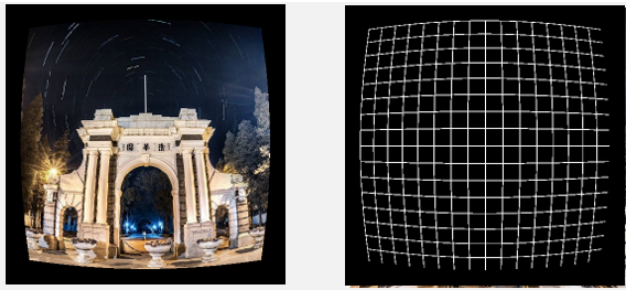

图 2: 畸变校正

2.2 TPS 人脸变形

TPS 人脸变形需要设计一个关键点匹配的方案，寻找到一个能够通过所有控制点 (目标人脸) 的方案，使得原图像在这些关键点上能够呈现出相同的” 形状”，以达到人脸变形的目的。而任务中的关键点均给出，只需要按照 TPS 函数系数的求解方法进行求解得到坐标的映射关系后进行插值即可。此外由于人脸坐标的相对位置不重合以及尺度不匹配，需要额外进行一些关键点的配准工作，最终实现以下图 [3] 的变形效果：

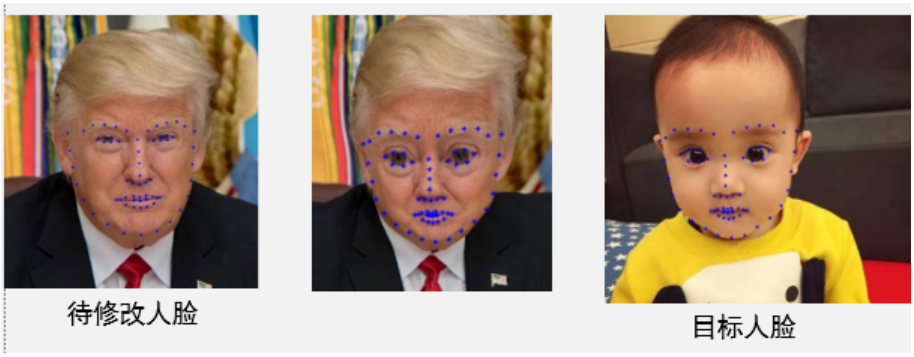

图 3: 人脸变形

3 方案设计及基本原理

3.1 整体方案

我对于本次大作业将任务主要分解为三部分，也是下面我将详细叙述的：**变形函数**，**插值函数**和 **UI 设计**。整体上本次大作业中所有的图像变换都是通过**后向插值**来完成的:

变形函数是独立于其他两部分的，主要实现了坐标的变换关系，这是由需要实现的具体变形而决定的。变形函数是插值的前提，因此需要按照上述的需求来完成并测试插值函数，通过后期修改系数等来达到图片变换的最理想效果。

插值函数是用于变换后的非整数型坐标对整数型坐标的像素进行插值的函数，主要是由于大多数左边在经过变形函数的变换后并非整数格点，而变换前后的图片的大小是相等的，这意味着许多格点是无法直接获取像素值的，需要通过其临近的已知整数格点来估计这一点的像素值，不同的插值函数会导致图像的效果有微小的区别，因此按照三种插值方法我分别设计了三个函数，在每一个变形函数后使用。

UI 设计我采用了 C# WPF 进行设计，在完成作业的过程中，我将必做和选做任务分开完成的，因此我通过一个分页设计 *tabcontrol* 控件将两个任务的 UI 分开进行设计，这也是因为两任务所需要展示的图片数量不同使然以及所需要用户选取的参数不同，接下来我将具体介绍各部分的原理和设计。

此外，我设计了两个类库，将一些操作进行封装，分别是 My_cv 类和 $Matrix$ 类，前者实现了对于矩阵的网格变形以及 TPS 函数的坐标映射，后者是由于 C# 中没有对于矩阵操作的类库，因此我实现了一些矩阵的基本操作，如：矩阵转置，矩阵求逆，矩阵合并等 TPS 求解过程中所需要用到的函数。以此在主窗口中进行调用来完成整体程序和界面的设计。

3.2 变形函数

3.2.1 旋转变形

旋转变形首先需要多个参数，分别是中心点，最大旋转角和最大旋转半径，以及旋转方向，现考虑顺时针旋转，则对于每一对坐标 (x,y)，其旋转后的坐标 (x',y') 满足下式：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

几何直观上来看为将坐标轴逆时针旋转了 θ 角：

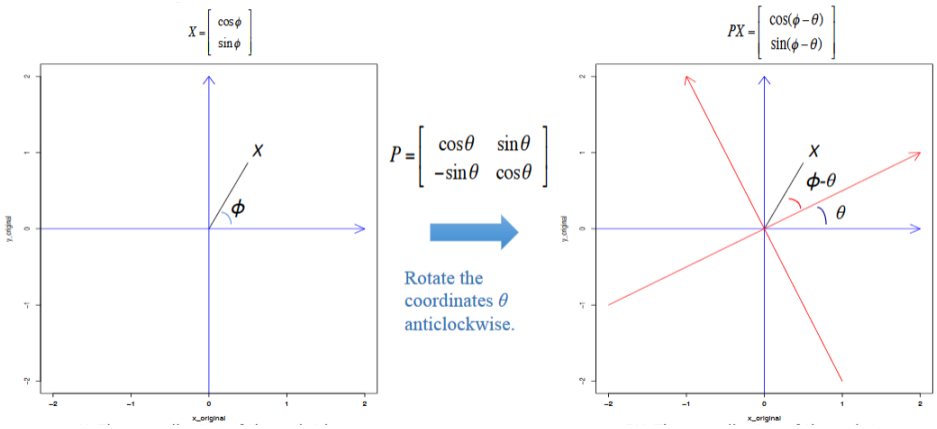

图 4: 坐标旋转矩阵

对于图片的旋转，我们规定了最大旋转角度和最大旋转半径，而对于每个像素点，其旋转角度与其旋转半径有关，具体按照如下的公式确定旋转角：

$$a = a_{max} \times \frac{Radius - Distance}{Radius}$$

而针对于反向插值的方法，倘若使目标图片呈现出顺时针旋转的效果，需要对目标图片的像素点进行逆时针旋转以求其在原图像中的位置，得到的为浮点型坐标，而此浮点坐标需要继续通过插值来获得其像素值。

3.2.2 畸变校正

畸变校正的变形函数在几何意义上理解为：将图片铺在一个半径为 Radius 的球面上方，从竖直方向向下看此图片可以得到此图片的桶形畸变效果，也即“凸图像”的效果，运用简单的立体几何知识进行推导，可以得到原图像 (x,y) 和畸变图像 (x',y') 的坐标映射关系：

$$\begin{cases} x &= [\frac{2Raidus}{\pi Distance} \cdot arcsin(\frac{Distance}{Radius})]x' \\ y &= [\frac{2Raidus}{\pi Distance} \cdot arcsin(\frac{Distance}{Radius})]y' \end{cases}$$

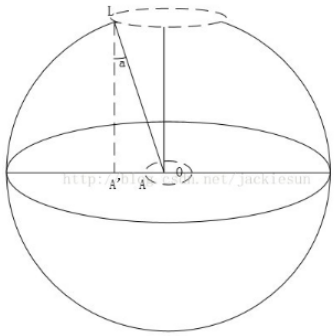

图 5: 畸变校正示意图

以上公式对坐标进行了 $\pi/2$ 的放缩，表现了图片平铺在球面 [5] 上方的情形，而畸变校正本质上应实现上述公式的逆映射，而这种映射是可以方便地通过倒数来完成。

以上这种畸变属于径向畸变，而径向畸变分为桶形畸变和枕形畸变，对于枕形畸变的公式，为了实现凹的效果我直接将上述公式系数取倒数。

3.2.3 TPS 变形函数

薄板样条是一种常见的插值模型，旨在找到通过所有控制点的光滑曲面，并且使能量函数最小，能量函数如下：

$$I_f = \iint_{R^2} ((\frac{\partial^2 f}{\partial x^2})^2 + 2(\frac{\partial^2 f}{\partial x \partial y})^2 + (\frac{\partial^2 f}{\partial y^2})^2) \mathrm{d}x\mathrm{d}y$$

解析解的求法如下，设控制点 (目标人脸) 矩阵为 P，记录为：

$$K = \begin{pmatrix} 0 & U(r_{12}) & \cdots & U(r_{1n}) \\ U(r_{21}) & 0 & \cdots & U(r_{2n}) \\ \cdots & \cdots & \cdots & \cdots \\ U(r_{n1}) & U(r_{n2}) & \cdots & 0 \end{pmatrix}, P = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \cdots & \cdots & \cdots \\ 1 & x_n & yn \end{pmatrix}, L = \begin{pmatrix} K & P \\ P^T & 0 \end{pmatrix}$$

K 矩阵中的元素 U 的计算方法为 U(r) 径向基函数：

$$U(r) = \begin{cases} r^2 log(r) & , r \neq 0 \\ 0 & , r = 0 \end{cases}$$

目标 (被变形图片) 关键点坐标记录为 V 矩阵，记录矩阵如下：

$$V = \begin{pmatrix} x'_1 & x'_2 & \cdots & x'_n \\ y'_1 & y'_2 & \cdots & y'_n \end{pmatrix}, Y = \begin{pmatrix} x'_1 & x'_2 & \cdots & x'_n & 0 & 0 & 0 \\ y'_1 & y'_2 & \cdots & y'_n & 0 & 0 & 0 \end{pmatrix}$$

求解 TPS，按照如下的就矩阵运算方法：

$$\begin{pmatrix} \omega_1 & \cdots & \omega_n & a_1 & a_x & a_y \end{pmatrix}^T = L^{-1}Y$$

那么坐标映射关系函数为

$$f(x,y)=[f_x(x,y),f_y(x,y)]^T=a_1+a_xx+a_yy+\sum_{i=1}^{n}\omega_iU(|P_i-(x,y|)$$

进而按照上述的坐标映射关系进行后向插值即可。

此外，由于每两张图像上的关键点所在位置并不相同，并且尺度也不相同，倘若直接使用上述关键点，可能导致结果图像大小不合适或边缘拉伸严重，因此我进行了简单的关键点配准。

假设目标关键点为 Targrt, 控制关键点为 Source，那么需要将 Source 面向 Target 进行配准，我使用 $mean(Target)$ 和 $mean(Source)$ 代表两组关键点的中心点，那么平移 Source 的中心至 Target 的中心即可。关于尺度配准，分别计算两组关键点在 x,y 方向上的标准差

$$std(Source)=\sqrt{\frac{\sum_{i=1}^{n}((x_i,y_i)-(\bar{x},\bar{y}))^2}{n-1}}$$

同上将 Source 中的点关于其中心点进行标准差放缩。可以改善最终的变形效果。

3.3 插值函数

3.3.1 最近邻插值 Nearest

最近邻插值对于浮点型坐标 (x',y') 上的像素值，只需要按照四舍五入找到距离 (x',y') 最近的整数型坐标 (x,y)，使用此整数型坐标的像素作为浮点型坐标的像素。也即：

$$Pixel(x',y')=Pixel(round(x),round(y))$$

3.3.2 双线性插值 Bilinear

双线性插值意为在二维的矩阵上，通过最近邻的四个点在 x,y 方向上进行两次线性插值即可，按照下图进行推导：

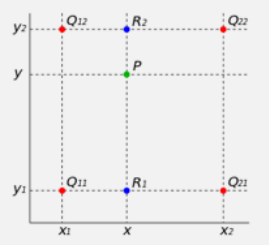

图 6: 双线性插值示意图

上图中因为取浮点型坐标的最近邻的四个坐标，因此 $(Q_{22}-Q_{12})_x=(Q_{12}-Q_{11})_y=1$, 若使用 f(x,y) 表示在 (x,y) 处的像素值，利用两次线性插值, 首先在 x 方向上插值得到 R_1 和 R_2 的像素值：

$$f(R_2)=\frac{x_2-x}{x_2-x_1}f(Q_{22})+\frac{x-x_1}{x_2-x_1}f(Q_{12})$$

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{21}) + \frac{x - x_1}{x_2 - x_1} f(Q_{11})$$

进而在 y 方向上利用 R_1 和 R_2 插值得到 P 的像素值：

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_2) + \frac{y - y_1}{y_2 - y_1} f(R_1)$$

在上式中 $y_2 - y_1 = x_2 - x_1 = 1$, 将最后的结果写成矩阵形式为：

$$f(x,y) = \begin{pmatrix} x_2 - x & x - x_1 \end{pmatrix} \begin{pmatrix} f(x_1,y_1) & f(x_1,y_2) \\ f(x_2,y_1) & f(x_2,y_2) \end{pmatrix} \begin{pmatrix} y_2 - y \\ y - y_1 \end{pmatrix}$$

从最终推得的公式也可以验证，如果先从 y 方向上进行插值，再在 x 方向上进行插值，那么最终得到的插值结果仍是相同的。

3.3.3 双三次插值 Bicubic

双三次插值旨在利用更多的临近点的像素信息来给出浮点坐标 (x',y') 像素值一个更好的估计，若假设 $(floor(x', y') = (x, y))$, 那么其使用邻近的十六个整数型点为 (x-1:x+2,y-1:y+2), 共十六个点去进行插值的估计，根据到此点 xy 方向距离不同，这 16 个点的权重也不同，权重函数如下：

$$S(x) = \begin{cases} 1 - 2|x|^2 + |x|^3 & , |x| \le 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3 & , 1 < |x| < 2 \\ 0 & , otherwise \end{cases}$$

不妨使用 (x',y') 表示浮点型坐标，$(floor(x', y') = (x, y))$，并且其小数部分表示为 $u = x' - x, v = y' - y$, 那么 (x',y') 处的像素值表示为：

$$f(x', y') = \begin{pmatrix} S(u+1) & S(u) & S(u-1) & S(u-2) \end{pmatrix} \times f(x-1:x+2, y-1:y+2) \times \begin{pmatrix} S(v+1) \\ S(v) \\ S(v-1) \\ S(v-2) \end{pmatrix}$$

则可以利用更多的信息得到此浮点型坐标的像素值，往往能够使图像看起来更为“真实”，此外，由于取临近点，因此需要在程序中考虑边界情况。

3.4 UI 设计

本次 UI 设计使用 WPF 进行设计，力求简约美观的风格，共分为两个页面，由于两必做任务和选做任务所需的参数不同，因此参数区的选择分开选取，整体上根据对需求的分析设计界面，按照核心算法部分的分块进行相应的 UI 的分开设计，整体展示如下 (另一 page 详见于结果分析5中)：

具体的各个控件的操作以及 UI 的具体细节，不在此处进行展示，详见于 README[1]文档。

[1]README.pdf 关于本程序的使用说明和技术分析

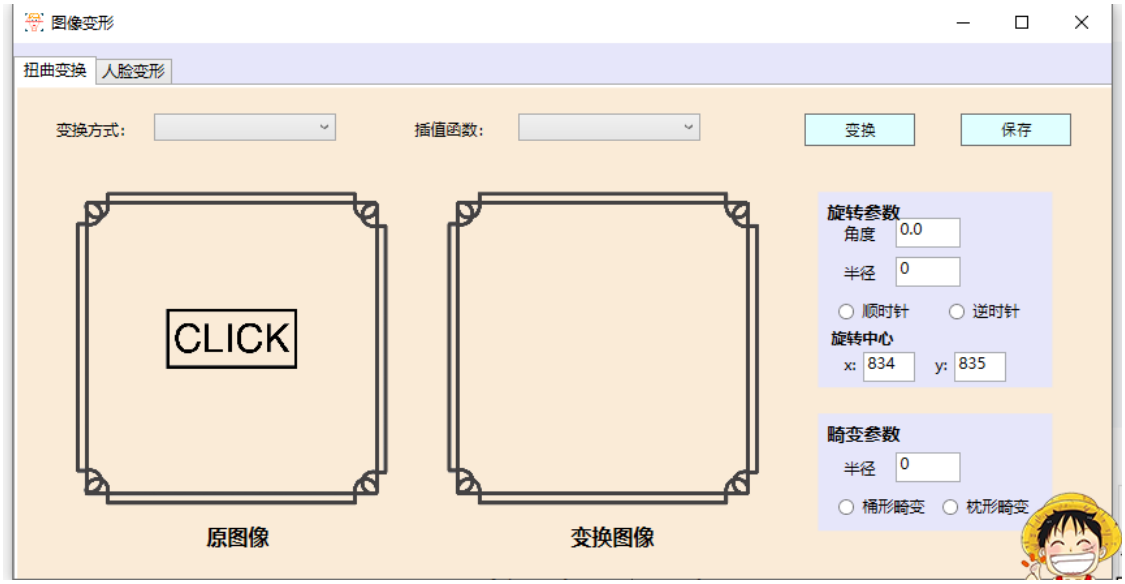

图 7: UI 初始界面

4 误差分析

在这一节中，我将按照误差分析的来源，按照课上的分析方法分别进行误差分析。

4.1 观测误差

在本次作业的计算过程中，并没有出现由于观测者主观所引起的误差，仅在最后观测结果图片时具有主观行为，但这时的观测并没有进行任何数值分析，因此不再赘述观测误差。

4.2 模型误差

模型误差反映的是由实际对象抽象到数学模型所带来的误差，在本次任务中不存在建立数学模型的过程，因为所使用的图片其本身存储已经为 RGB 三通道存储像素值的格式，实际上本程序是在已经建立的数学模型上进行操作，而如果考虑从真实模拟的图片转换为.jpg 格式的图片的模型误差，那么像素值在 0-255 之间的整数，因此 $R \leq \frac{1}{2}$。

4.3 舍入误差

舍入误差主要来自于两个方面，分别是**计算误差**的累积和**存储误差**, 其中每步的计算误差实际上是由于存储误差所带来的，因此下面分别分析这两方面的误差。

存储误差。本程序中存储共用了两种形式，分别为 double 型和 int 型。double 可以保证十进制科学计数法小数点后 15 位有效精度和第 16 位的部分精度，因此我们取其精度为小数点后 15 位，那么每一步 double 型的计算误差约为 10^{-16}, 而对于 int 类型的存储，只发生在最终转换成整数作为像素值的时刻，中间步骤的所有计算均使用 double，因此这部分的误差为 $\frac{1}{2}$。

计算误差。这部分主要分析累积的计算误差，实际上 double 类型的单步计算误差量级十分小，对于整体问题而言需要观察其累积误差是否会超过 int 型的存储误差，否则整体误差量级仍由 int 型存储而决定。

对于 double 类型的累积运算，主要出现在程序的**关键点配准**对于关键点方差的计算过程和**求解逆矩阵**的高斯消元过程及后续的**矩阵运算**过程。前者的运算复杂度为 $O(n^2)$, 高斯消元过程的复杂度量级在 $O(n^2)$，对于每个元素其操作复杂度为 2n, 在两者中 n 都为 68，因此若对其取一个上界，不妨取 10^4, 而在矩阵乘法运算的过程中，每个元素的计算需要 (n+3) 次乘法和 (n+2) 次加法，数量级仍为 n 不超过 $O(n^2)$, 那么 double 类型累积的舍入误差取上限为 10^{-12}，仍远小于 $\frac{1}{2}$, 因此这部分误差取 $R \leq \frac{1}{2}$

4.4 方法误差

4.4.1 最近邻插值

不妨设像素值在坐标 (x,y) 处满足函数 $f(x,y)$, 仅考虑方法误差时则可以认为原图像本是模拟的，也即图像像素值函数本应满足连续性，那么对于二元函数的误差估计，有如下的结果：

$$|\Delta A| \leq max|\frac{\partial f}{\partial x}| \cdot |\Delta x| + max|\frac{\partial f}{\partial y}| \cdot |\Delta y|$$

而在 x,y 方向上的截断误差均为 $|\Delta x| \leq \frac{1}{2}, |\Delta y| \leq \frac{1}{2}$, 因此有如下的放缩：

$$|\Delta A| \leq \frac{max|\frac{\partial f}{\partial x}| + max|\frac{\partial f}{\partial y}|}{2}$$

因此对于最近邻插值可以取误差上限为 $\frac{1}{2}(max|\frac{\partial f}{\partial x}| + max|\frac{\partial f}{\partial y}|)$。

4.4.2 双线性插值

对于双线性插值，在原理3.3.2中已经分析，可以看做 x 方向和 y 方向按照先后顺序进行插值，根据拉格朗日插值 [1], 在 x 方向上进行两次一次的插值, 下面对于某一个一次插值进行分析，根据拉格朗日插值余项：

$$|R_n(x)| \leq \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|$$

那么，对于 x 而言，为方便表示我首先规定 $u = x - floor(x), 1 - u = ceiling(x) - x$, 那么上式为：

$$\begin{aligned} |R_1(x)| &\leq \frac{M_2}{2!}|\omega_2(x)| \\ &= \frac{1}{2}|\frac{\partial^2 f}{\partial x^2}| \cdot |u(1-u)| \\ &\leq \frac{1}{2}|\frac{\partial^2 f}{\partial x^2}| \cdot \frac{1}{4} \\ &\leq \frac{1}{8}max|\frac{\partial^2 f}{\partial x^2}| \end{aligned}$$

而对于图 [6] 中，进而对 x 方向上的两次插值结果进行运算，但此次插值与 x 方向无关，相当于按 y 方向分配权重，但在两个不同的 y 值上的两次 x 方向插值的误差上界都如上式4.4.2所示，因此最终 x 方向上的误差上界为 $\frac{1}{8}max|\frac{\partial^2 f}{\partial x^2}|$。在 y 方向上，同样有：

$$|R_1(y)| \leq \frac{1}{8}max|\frac{\partial^2 f}{\partial y^2}|$$

则对于方法整体上，误差上界为：

$$|R(x,y)| \leq |R_x(x,y)| + |R_y(x,y)| \leq \frac{1}{8}(max|\frac{\partial^2 f}{\partial x^2}| + max|\frac{\partial^2 f}{\partial y^2}|)$$

4.4.3 双三次插值

双三次插值类似于上述的双线性插值，不过是把一维的三次样条插值推广到二维情形, 对于其插值函数同时满足，

$$\begin{cases} S_i(x_i) & = y_i \\ S_i(x_{i+1}) & = y_{i+1} \end{cases}$$

对于任意一组 x 方向上的三次样条插值，使用书上关于三次样条函数的误差界的定理 [1]:

$$\max_{a\leq x\leq b} |f^{(k)}(x) - S^{(k)}(x)| \leq C_k \max_{a\leq x\leq b} |f^{(4)}(x)|h^{4-k}, k = 0, 1, 2$$

其中，$C_0 = \frac{5}{384}, C_1 = \frac{1}{24}, C_2 = \frac{3}{8}$, 而在这里我使用的插值为 k=0 的情形，并且步长 h 为 1，对于二维情况，需要将导数符号修改为偏导，那么在 x 方向上的三次样条插值的误差上界为：

$$R_x(x,y) \leq \frac{5}{384}|\frac{\partial^4 f}{\partial x^4}|$$

对于 y 方向上，同理，

$$R_y(x,y) \leq \frac{5}{384}|\frac{\partial^4 f}{\partial y^4}|$$

同理，整体的误差界可如下放缩，

$$|R(x,y)| \leq |R_x(x,y)| + |R_y(x,y)| \leq \frac{5}{384}(max|\frac{\partial^4 f}{\partial x^4}| + max|\frac{\partial^4 f}{\partial y^4}|)$$

因此双三次插值的方法误差上界为 $384(max|\frac{\partial^4 f}{\partial x^4}| + max|\frac{\partial^4 f}{\partial y^4}|)$, 整理比较如下：

方法	误差上界
最近邻插值	$\frac{1}{2}(max\|\frac{\partial f}{\partial x}\| + max\|\frac{\partial f}{\partial y}\|)$
双线性插值	$\frac{1}{8}(max\|\frac{\partial^2 f}{\partial x^2}\| + max\|\frac{\partial^2 f}{\partial y^2}\|)$
双三次插值	$384(max\|\frac{\partial^4 f}{\partial x^4}\| + max\|\frac{\partial^4 f}{\partial y^4}\|)$

表 1: 误差上限表格

5 实验结果

实验结果在这小节内我将展示需求分析中期望达到的目标，以及对每一种图像的变形进行展示。

旋转扭曲 我采用二校门的图片进行演示：

图 8: 旋转扭曲结果

上面图为我在如图所示的参数情况下对二校门进行旋转扭曲，可以看到已经取得了理想的效果，这时的插值函数的选择为“最近邻插值”。参数的调整通过右侧的两个面板即可，同时图片的上传通过点击原图像即可，保存图片可以通过点击结果图片或保存按钮。

畸形矫正 同样采用二校门的图片，将变形方式变为畸变校正，这里我采用桶形畸变并且插值函数为双二次插值。

图 9: 桶形畸变结果

由于畸变校正是将图片平铺在球面上，为使这一效果更明显，我取消了系数的放缩，因此若球体半径较小时，图片将会包围成一个圆，呈现出一个圆形的图片，因此为取得良好的效果，往往需要畸变半径较大。

TPS 人脸变形 在这里测试我选用的图片为 8 变换至 6，选用双三次插值函数，至此将三种插值方法都已经展示，结果如下图所示：

图 10: 人脸变形结果

可见，经过我的关键点配准后，结果的图片尺寸和图片位置已经能够较好供用户去观察，也并未有大幅度的拉伸和失真。在 UI 界面上通过点击原图和目标人脸下的图片可以实现上传新图片，关键点显示按钮，按下时可以根据当前状态对两张变换前图像进行关键点的标记或者消除，右侧展现了核心插值函数的用时，最近邻，双线性，双三次的复杂度依次增高，其运行时间也依次增加。

对于上方三幅图片未能展示出的情况，用户可以通过运行可执行程序手动进行测试，查看结果，本程序已经进行了一定的容错设计，使得用户操作不会导致程序崩溃。

6 总结

6.1 出现的问题及解决方法

本次作业的完成过程中，整体上较为顺利，但过程中也出现了一些问题，下面我将叙述一些出现的典型的问题。

变换数组越界问题 这部分问题实际上属于细节问题，由于变形函数并不能保证将当前坐标进行映射后，一定处在原图内，因此会出现访问越界的问题，对于这些情况，应当单独用 (0,0,0) 的像素对这些点进行幅值。其次，由于映射后的坐标不能保证不处在边缘，因此对于双线性插值

和双三次插值也会出现越界的情况，导致程序会异常退出，这里针对不同的插值函数进行不同的边界处理即可。

TPS 插值问题 对于 TPS 插值实际上主要过程为写出矩阵运算的方法，因此需要自己去设计一个矩阵类，提供给 TPS 部分进行使用，而由于我在刚开始写的时候对于 TPS 的理解并不透彻，导致我将控制点和目标点混淆，导致并未出现人脸变换的效果，这在调试时也很难去排查。于是我从网上查阅资料，并重新去分析大作业课件上给出的公式，最后发现控制点为目标人脸而目标点为待修改人脸。修改后能够出现类似于课件上的状态。但例如 8 至 6 的变换，变换后的图片较小且集中在左上角，我进一步去理解 TPS 函数发现 TPS 函数实际上使得被修改人脸变形使其特征和目标人脸的关键点相匹配，那么如果平移和尺度变换不会改变关键点特征，因此我使用控制点向目标点进行了中心对齐和标准差对齐，最终才能呈现较良好的效果。

6.2 心得体会

本次数值大作业进一步锻炼了我的编程能力和 UI 设计的能力，这一次大作业我使用了 WPF 进行界面的设计，相比与我刚刚完成的人智大作业的 winform，WPF 具有更强的可定制性，其界面也更易做的简约美观。此外，这次作业使我进一步对插值函数加深理解，也了解到其在图像处理上的应用，特别是双三次插值，在课上对此的学习和理解都较为浅显，通过实际应用它能够加深对其的理解。通过误差分析部分，我对于误差分析的方法以及一次实际的项目中的误差来源都更加了解，总而言之，本次大作业加强了我对于课内插值知识的应用拓展，也使我对理论知识有更加深刻的掌握。

参考文献

[1] 王能超易大义, 李庆扬. 数值分析. 清华大学出版社, 2008.