

图像扭曲变形

——数值分析 课程大作业 1

贾成君 自 46 学号：2014011552

需求分析

任务要求分为三个方面：实现图像的旋转扭曲、图像畸变和 TPS 网格变形。因为该程序要实现的功能并不复杂，因此不使用 UI 界面，而使用控制台直接完成。原来考虑是否增加手动调用函数（手动指定各个图像变形的参数），因为程序这样封装实现后和基本代码并没有什么区别，没有实现什么简洁性，因此放弃了这种功能的增加。

因此，最后程序的需求定义为：使用控制台选择变形方式、源图片目标、使用的插值方式，利用鼠标进行参数的调整和指定。

方案设计

为了提高代码的复用性，将主要的函数封装成一个类：PicChange；类主要成员为原图 `orgIm(Mat 类型)` 和图像的相关性质(行、高、通道数¹)。

计算变换后的图像的思路为：计算新的图像上的每一点对应的原图的坐标（为了记录映射关系，建立一种数据类型，存储的是离散的点到点的映射关系——见下左图）；然后根据坐标对应关系和原来图像的像素值，使用合适的插值方法，计算出新图像上的点对应的像素。则封装的类的成员变量和成员函数如下右图。

PointToPoint
-原来的点
-新的点

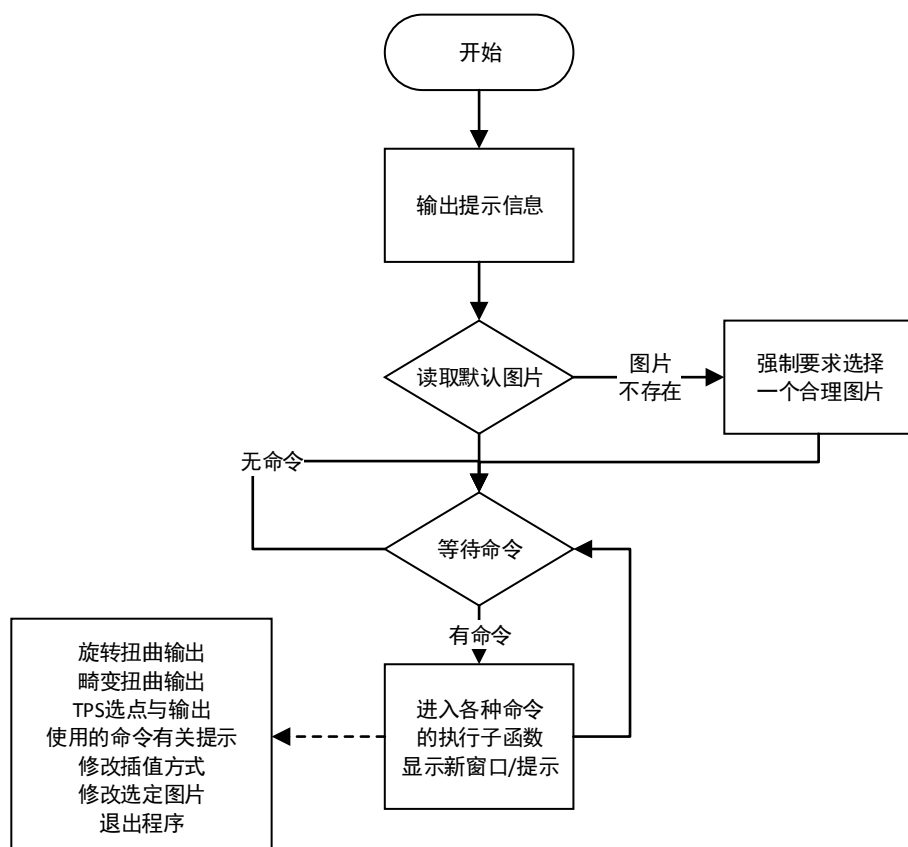
映射表

$$\begin{array}{ccccccc} (x_{11}, y_{11}) \rightarrow (x'_{11}, y'_{11}) & (x_{12}, y_{12}) \rightarrow (x'_{12}, y'_{12}) & \cdots & (x_{1n}, y_{1n}) \rightarrow (x'_{1n}, y'_{1n}) \\ (x_{21}, y_{21}) \rightarrow (x'_{21}, y'_{21}) & (x_{22}, y_{22}) \rightarrow (x'_{22}, y'_{22}) & \cdots & (x_{2n}, y_{2n}) \rightarrow (x'_{2n}, y'_{2n}) \\ \cdots & \cdots & \cdots & \cdots \\ (x_{m1}, y_{m1}) \rightarrow (x'_{m1}, y'_{m1}) & \cdots & \cdots & (x_{m02}, y_{m02}) \rightarrow (x'_{m02}, y'_{m02}) \end{array}$$

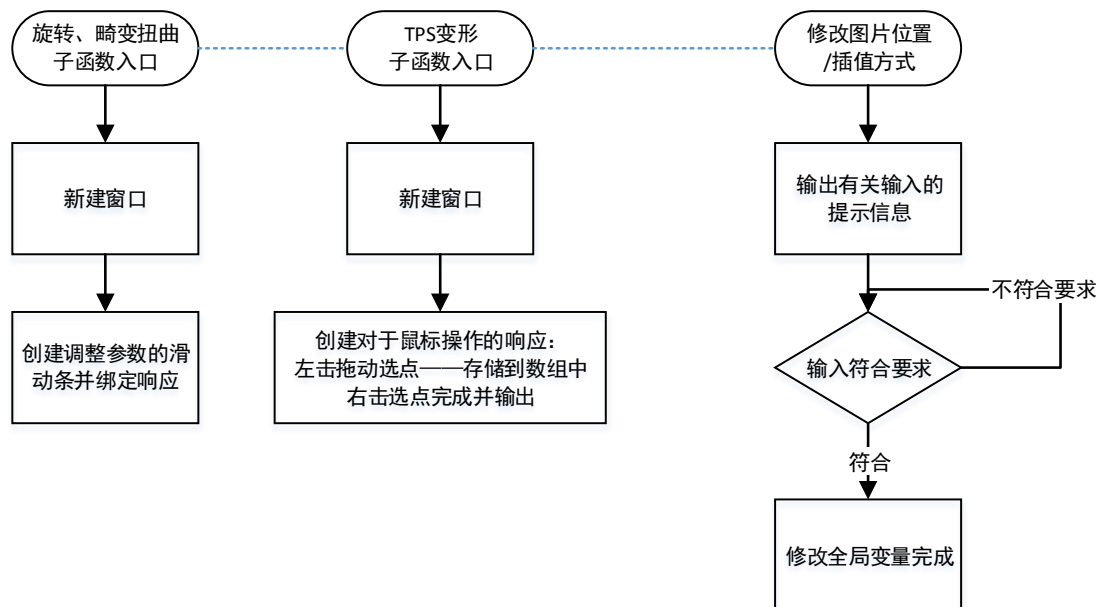
PicChange//图像变换
私有变量：
-原始图像
-原始图像性质
-使用的一些中间变量
函数：
-读取图像
-计算映射关系
-插值计算新图像(使用映射关系)

¹ 为了便于后续计算，本程序把图像统一处理成 3 维图像(RGB 三色图)。

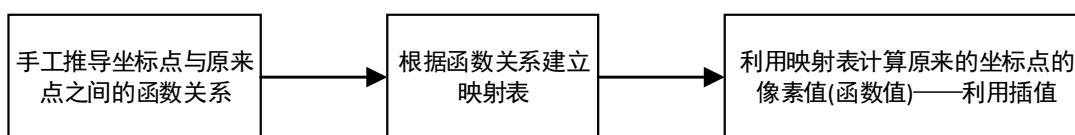
整理软件的工作流程如下：



而各种命令调用的子函数的工作流程为：



计算旋转、扭曲、TPS 变形求变换后的图像使用的计算思路统一为：



方案基本原理

这里着重说明映射表的计算和插值的计算方法；对于不同的变形方案，使用不同的函数来计算。整理成下表：

变形方案	核心思想	典型方案设计
旋转扭曲	把图像上的点绕着中心点旋转	<p>把中心点作为坐标系的原点（如果不是，则进行坐标变换）</p> <p>极坐标系：$(\rho', \theta') = (\rho, \theta + \alpha)$，其中 $\alpha = \frac{row - r}{row} \times \theta^*$，$row, \theta^*$ 提前给定，$r = \rho$</p> <p>正交坐标系：利用矩阵变换 $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & 1 - \cos(\alpha) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$</p>
畸变扭曲	把图像上的点沿着中心点连接的射线外移或内移	<p>把中心点作为坐标系的原点（如果不是，则进行坐标变换）</p> <p>极坐标系：$(\rho', \theta') = (D\rho, \theta)$，其中 D 提前给定或者写成 ρ 的函数；比较典型的方法是写成 $D = \frac{\theta - \tan(\theta)}{\tan(\theta)}$。其中 $\frac{\rho}{h} = \tan(\theta)$，而 $h = \frac{\rho_{\max}}{\theta_{\max}}$ (θ_{\max} 给定)</p> <p>正交坐标系：利用矩阵变换 $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} D & 0 \\ 0 & D \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$</p>
TPS 变形	根据选定的映射点和最小功率的约束计算出变形函数	<p>利用老师给定的说明的插值函数：</p> $f(x, y) = [f_x(x, y), f_y(x, y)]^T = \mathbf{a}_1 + \mathbf{a}_x x + \mathbf{a}_y y + \sum_{i=1}^n \mathbf{w}_i U(P_i - (x, y))$ <p>其中 $\mathbf{a}_1, \mathbf{a}_x, \mathbf{a}_y, \mathbf{w}$ 是 $L[\mathbf{w}_1, \dots, \mathbf{w}_n, \mathbf{a}_1, \mathbf{a}_x, \mathbf{a}_y]^T = Y$ 的解；而 $L = \begin{bmatrix} K & P \\ P^T & \mathbf{0} \end{bmatrix}$</p> $K = \begin{bmatrix} 0 & U(r_{12}) & \cdots & U(r_{1n}) \\ U(r_{21}) & 0 & \cdots & U(r_{2n}) \\ \cdots & \cdots & \cdots & \cdots \\ U(r_{n1}) & U(r_{n2}) & \cdots & 0 \end{bmatrix} \quad P = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \cdots & \cdots & \cdots \\ 1 & x_n & y_n \end{bmatrix}$ $U(r) = \begin{cases} r^2 \log(r^2), & r \neq 0 \\ 0, & r = 0 \end{cases} \quad Y = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ V & \mathbf{0} & \mathbf{0} \end{pmatrix}^T \quad V = \begin{bmatrix} x'_1 & x'_2 & \cdots & x'_n \\ y'_1 & y'_2 & \cdots & y'_n \end{bmatrix}$

而使用的插值方案整理如下：

插值方案	核心理念	典型方案设计
最近邻插值	使用距离插值点最近的点对应的函数值作为插值结果	<p>如果我们已有的函数点都是“整点”的话：</p> $f(i+u, j+v) = \begin{cases} f(i, j) & i \leq 0.5, j \leq 0.5 \\ f(i+1, j) & i > 0.5, j \leq 0.5 \\ f(i, j+1) & i \leq 0.5, j > 0.5 \\ f(i+1, j+1) & i > 0.5, j > 0.5 \end{cases}$ <p>其中 i, j 是整数，u, v 是 $[0, 1]$ 上的小数(下同)</p>
双线性插值	使用插值点最近的两个点(x, y 方向上各取两个)进行多项式插值计算	$f(i+u, j+v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} f(i, j) & f(i, j+1) \\ f(i+1, j) & f(i+1, j+1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}$
双三次插值	使用插值点最近的三个点(x, y 方向上各取三个)进行多项式插值计算	$f(i+u, j+v) = ABC^T$ $A = \begin{bmatrix} S(u+1) & S(u) & S(u-1) & S(u-2) \end{bmatrix}$ $C = \begin{bmatrix} S(v+1) & S(v) & S(v-1) & S(v-2) \end{bmatrix}$ $B = f(i-1:i+2, j-1:j+2)$ <p>其中插值核函数是：</p> $S(x) = \begin{cases} 1-2 x ^2+ x ^3 & x \leq 1 \\ 4-8 x +5 x ^2- x ^3 & 1 < x < 2 \\ 0 & otherwise \end{cases}$

理论误差分析

进行图像变换的主要误差来源分为两部分：一个是计算对应的原来的点时产生的坐标误差；一个是使用插值计算时产生的误差。

首先，我们估计第一个坐标误差。根据新的图像的坐标点计算原来图像的坐标点的过程是利用模型模拟求值的过程，我们把这个过程记为 $\begin{bmatrix} x \\ y \end{bmatrix} = f\left(\begin{bmatrix} x' \\ y' \end{bmatrix}\right)$ ；其中 (x,y) 为新的坐标点

(x',y') 对应的原来图像上的坐标。由于，新的坐标必须是整数点，属于强制要求，不存在观测误差；而三种图像变形的的方法，都是直接给定的坐标变换关系，因此函数 f 不存在模型误差；舍入误差是由于“存储数据为 `double` 类型”，其有效位数为 15 位；而计算出来的坐标的绝对值一般在 1000 以下(受到图像大小的影响)，带来舍入误差小于 10^{-12} ；计算过程中的方法误差和最后的误差；则整理如下表





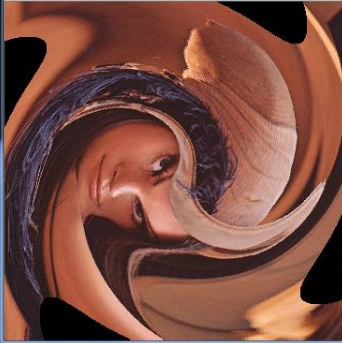



变 形 方法	方法误差分析	总误差
旋 转 扭曲	<p>由于最后的坐标是在直角坐标系中，使用变化方案为直角坐标系中的方案 $x = x' * \cos(\alpha) + y' * \sin(\alpha)$ ($\alpha = \frac{row - r}{row} * \theta$)，其中 <code>row</code>、$\theta$ 指定，不存在误差；x',y' 为整数，不存在误差，<code>r</code> 使用 <code>cmath</code> 包中的开根函数，只记录舍入误差 10^{-12}，由此带来 α 的误差约为 $\frac{\theta \Delta r}{row}$，其中给定的 $\theta \in [0, 2\pi] \subseteq [0, 10]$，$row \in [1, 1000]$，由此 α 的误差最大为 10^{-8}（此时舍入误差为 10^{-12}，可以忽略不计；但是一般 <code>row</code> 和 θ 一般会取得较大，使得方法误差比舍入误差还小）；带入 x 的表达式中，得到 x 的方法误差</p> $< (-x' \sin \alpha + y' \cos \alpha) \Delta \alpha < (x' + y') \Delta \alpha < 2 * 10^3 * 10^{-8} = 2 * 10^{-5}$ <p>但是一般会在 $2 * 10^{-9}$</p>	最大不超过 $2 * 10^{-5}$ ，一般为 $2 * 10^{-9}$ 左右(受到指定的旋转半径大小的影响)
畸 变 扭曲	<p>各种基本分析同上，$h = \frac{\rho_{\max}}{\theta_{\max}}$ (θ_{\max} 给定)，则 h 的误差约为 10^{-12}（误差由舍入误差和 ρ_{\max} 的计算带来的误差；一般 θ_{\max} 会比 1 大使得畸变效果较为明显；极端情况下，本程序允许指定为 <code>pi/180=0.017</code>，此时误差会达到 $5.7 * 10^{-11}$）；$\frac{\rho}{h} = \tan(\theta)$ 带来 θ 的误差约为 $0.1 \Delta h$（使用求导，计算过程同上求解 x 的误差），再加上舍入误差，θ 的误差在 10^{-11} 左右；下一步 $D = \frac{\theta - \tan(\theta)}{\tan(\theta)}$ 中的 D 的误差，$\theta \in [0.1, \frac{1}{2} \pi]$ ($\theta = \arctan(\frac{\rho}{\rho_{\max}} \theta_{\max})$，约在 10^{-2} 量级)，求导计算可</p>	不会超过 10^{-6}

	<p>得，D 的误差约为$100\Delta\theta$，误差在 10^{-9}，带入$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} D & 0 \\ 0 & D \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix}$，可知误差为 10^{-6} 左右</p>	
TPS 变形	<p>TPS 的误差较难求解，由于求解方程组可以直接通过求逆矩阵求得；根据求解出来的方程组的解的大小情况，估计一系列参数的误差为 10^{-9}（误差与选择点的大小关系相关，很难计算出上界；因此根据最极端的情况，所有点都映射到边界点上，带来的根最大为 10^3 量级，对应的存储误差为 10^{-9}）。</p> <p>使用 TPS 对应的函数计算结果，误差大约为 $6 \cdot 10^6(\text{U(r)带来的误差}) \cdot 10(\text{选择的点的个数一般不多于 } 10 \text{ 个}) \cdot 10^{-9} = 6 \cdot 10^{-2}$</p>	<p>最坏不会超过 $6 \cdot 10^{-2}$，但是使用的控制点比较平均，误差会大大缩小(会小 3 个数量级左右[估计值])</p>

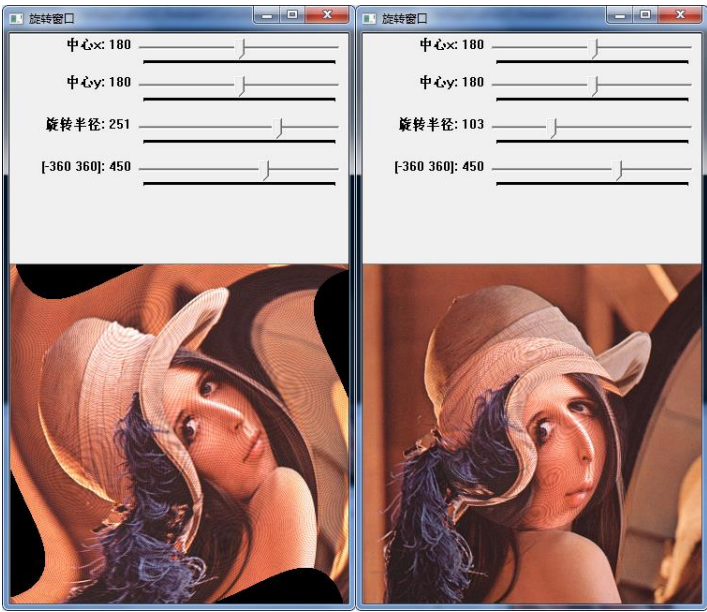
根据上述计算的误差，我们知道一般情况下，最后的点的误差不会多于 10^{-3} ，这样我们使用插值所选择的区间是比较合理的，可以认为没有模型误差，根据插值公式估算插值带来的误差。如果我们认为图像较为平滑连续（事实上，图像的像素值一般不会突变，比较平滑；如果不假设图像具有这个性质，那么我们插值最后带来的误差可能会达到 255[最大误差]）。双线性的插值余项估计值约为 $2 \cdot 1/2 \cdot 1/4 \cdot f''(m)$ ；图像像素值突变值不大于 10 时，加上舍入误差(舍入误差为 1),计算出的像素值的误差大约为 3；双三次插值的误差理论会更小一些。

程序输出结果与讨论

使用典型的图像进行输入，然后调整各种参数进行测试，得到不同情况下的输出结果，显示部分输出结果如下：

	旋转扭曲	畸变扭曲
双线性	<div><div><div>旋转窗口</div><div>中心x: 180 中心y: 180 旋转半径: 180 [-360 360]: 450</div></div><div><div>旋转窗口</div><div>中心x: 208 中心y: 205 旋转半径: 225 [-360 360]: 529</div></div></div>	<div><div><div>畸变窗口</div><div>中心x: 180 中心y: 180 畸变参数: 490 畸变方向: 0</div></div><div><div>畸变窗口</div><div>中心x: 36 中心y: 36 畸变参数: 274 畸变方向: 0</div></div></div>
最近邻	<div><div><div>旋转窗口</div><div>中心x: 180 中心y: 180 旋转半径: 227 [-360 360]: 201</div></div><div><div>旋转窗口</div><div>中心x: 180 中心y: 180 旋转半径: 239 [-360 360]: 720</div></div></div>	<div><div><div>畸变窗口</div><div>中心x: 180 中心y: 180 畸变参数: 441 畸变方向: 1</div></div><div><div>畸变窗口</div><div>中心x: 180 中心y: 180 畸变参数: 501 畸变方向: 1</div></div></div>

双三次



TPS 变形选择控制点和最后输出图像的关系举例如下：

控制台输出情况

```

E:\Visual Studio 2012\Projects\ConsoleApplication1\Release\ConsoleApplication1.exe
畸变窗口输出
t
输入命令：开始选择TPS匹配点...
选定点之间的映射：[63, 501]->[229, 781]
选定点之间的映射：[55, 1431]->[229, 1791]
选定点之间的映射：[62, 2371]->[241, 2641]
选点结束
选择了3对点
TPS窗口输出
c
选择插值方案(0/1/2):
0-双线性;
1-最近邻;
2-双三次
0
插值方法切换完成
输入命令：t
输入命令：开始选择TPS匹配点...
选定点之间的映射：[90, 621]->[240, 1011]
选定点之间的映射：[68, 1521]->[259, 1851]
选定点之间的映射：[61, 2551]->[250, 2891]
选点结束
选择了3对点
TPS窗口输出

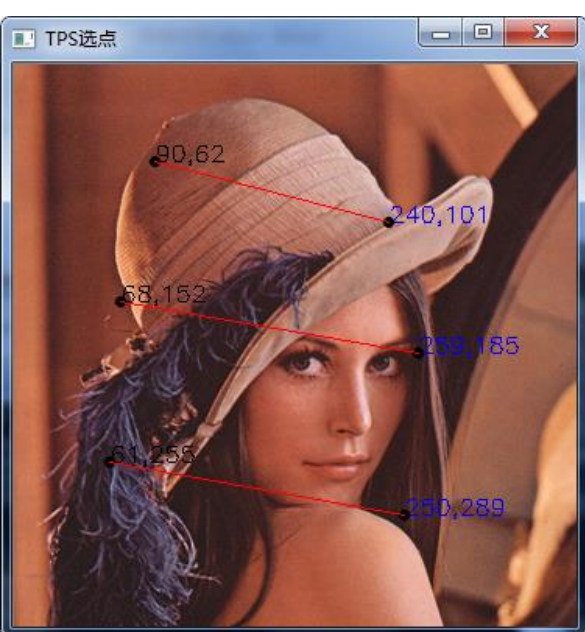
```

```

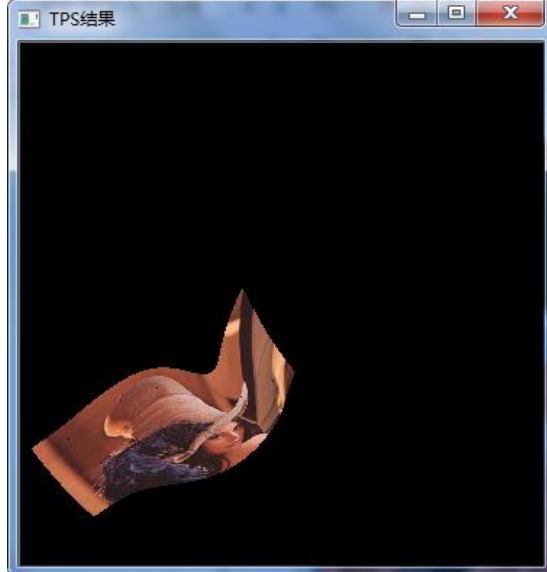
E:\Visual Studio 2012\Projects\ConsoleApplication1\Release\ConsoleApplication1.exe
0-双线性;
1-最近邻;
2-双三次
插值方法切换完成
输入命令：t
输入命令：开始选择TPS匹配点...
选定点之间的映射：[90, 621]->[240, 1011]
选定点之间的映射：[68, 1521]->[259, 1851]
选定点之间的映射：[61, 2551]->[250, 2891]
选点结束
选择了3对点
TPS窗口输出
开始选择TPS匹配点...
选定点之间的映射：[50, 371]->[273, 331]
选定点之间的映射：[56, 1391]->[238, 941]
选定点之间的映射：[75, 2921]->[244, 1521]
选定点之间的映射：[261, 3321]->[233, 1741]
选定点之间的映射：[321, 2661]->[269, 1661]
选定点之间的映射：[352, 521]->[308, 801]
选定点之间的映射：[348, 1091]->[298, 1291]
选点结束
选择了7对点
TPS窗口输出

```

选择点



输出结果



关于该程序还有部分可以扩展的地方；当然，基于主要函数的功能，用户交互方面可以做出改进，但是如需求分析中所说，意义不大。这里提供的是另外一种计算映射点的思路。

前文采用的都是根据新的坐标点，直接计算出其对应的原来坐标点；然后**在原来图像上做插值**从而推导出现在的图像结果；这样做的好处是：便于计算，使用的插值节点均匀、误差可以控制的比较好，甚至可以对原图进行一些高清修复从而使得最后计算结果更加准确。但是，这种控制方法缺点是 对于不清楚映射函数关系的变化没有办法进行。所以，我们可以使用另外一种思路：根据变换，计算出原来图像的坐标到新图像的坐标，然后**在新图像上进行插值**计算（这里的插值为了寻找最近的点，需要把数据存储为 桶，或者使用 kd-tree 来查找使用的插值节点；会增加计算量）；这样可以在只知道某些原图到新图的映射点而不知道映射关系的情况下求解新图像；当然，我们也可以使用函数拟合这些映射点，得到映射函数，进而使用原来的方法计算。