数值分析大作业一

图像扭曲变形

项目报告

晏筱雯

自 42 班

2014011459

2016.11

目录

一、实验任务

1.目标

编写图像扭曲变形程序，可以对图像进行扭曲变形。

（1）扭曲变形方式：

必做：旋转扭曲；图像畸变。

选做：TPS 网格变形。

（2）可采用的插值方法

最近邻插值；双线性插值（bilinear）； 双三次插值（bicubic）。

2.作业要求

提交程序源代码、执行码以及实验报告（包括：变形函数的选取与设计，使用插值方法的介绍与分析，程序框图，实验结果及分析等）。

3.注意

（1）建议程序用 VC 编写，详见大作业要求；
（2）可参考本书讲授方法，也可采用其它方法；
（3）自行编写全部算法，图像读写函数可使用现成的；
（4）鼓励创新，严禁抄袭。

二、编程环境

操作系统：Windows10

开发环境：Visual Studio 2012

编程语言：C#

三、UI 设计说明

程序界面如下图：

1.界面主要分为两部分，左边用来显示原图和变换后的图片，右边是功能选项菜单。

2.“变换类型”下拉菜单中可以选择“旋转扭曲”、“图像畸变”或者“TPS 网格变形”，“插值方式”下拉菜单中可以选择“最近邻插值”、“双线性插值”或者“双三次插值”。

3.“导入图片”和“保存图片”可以分别以指定的路径导入或者保存图片。

4.如果“变换类型”选择了“旋转扭曲”，需要设置旋转方向、旋转角度、区域大小等参数，旋转方向有顺时针和逆时针两个方向；如果“变换类型”选择了“图像畸变”，需要设置畸变类型和畸变程度，畸变类型有桶型和枕型两个选项；如果“变换类型”选择了“TPS 网格变形”，需要手动输入控制点对数。

5.设置好参数后，点击“开始”选项即可开始变换，如果有参数未设置，会弹出消息框提示；变换完成后，点击“恢复原图”就可以恢复变换前的图片。

四、需求分析

计算上的数字图像是一个大的二维数组，该数组的元素称为像素，其值称为灰度值。索引图像的文件结构比较复杂，除了存放图像的二维矩阵外，还包括一个称之为颜色索引矩阵 MAP 的二维数组。MAP 中每一行的三个元素分别指定该行对应颜色的红、绿、蓝单色值，MAP 中每一行对应图像矩阵像素的一个灰度值。

一种特定的图像变换对应着特定的变换函数，通过合适的数学模型我们可以求出变换函数，通过**坐标变换**完成图像变换。由于函数的计算结果可能是一个小数，故需要通过合适的**插值方式**来求出整数坐标从而完成像素值填充。

1.坐标变换

（1）旋转扭曲

以图片中心为旋转中心，选择一定的旋转程度和旋转区域大小，对原图进行旋转。

（2）图像畸变

以图片中心为畸变中心，选择畸变的类型（桶型或枕型）和畸变程度，对原图进行畸变。

（3）TPS 网格变形

设置控制点的对数并在原图上标注，根据所选控制点对的关系，进行 TPS 网格变形。

2.插值方式

（1）最近邻插值

以距离某点最近的像素点的值代替未知点的值。

（2）双线性插值

对某点向其附近 4 个点像素值做加权平均作为该点的像素值。

（3）双三次插值

对某点向其附近 16 个点像素值做加权平均作为该点的像素值。

3.展示与可靠性

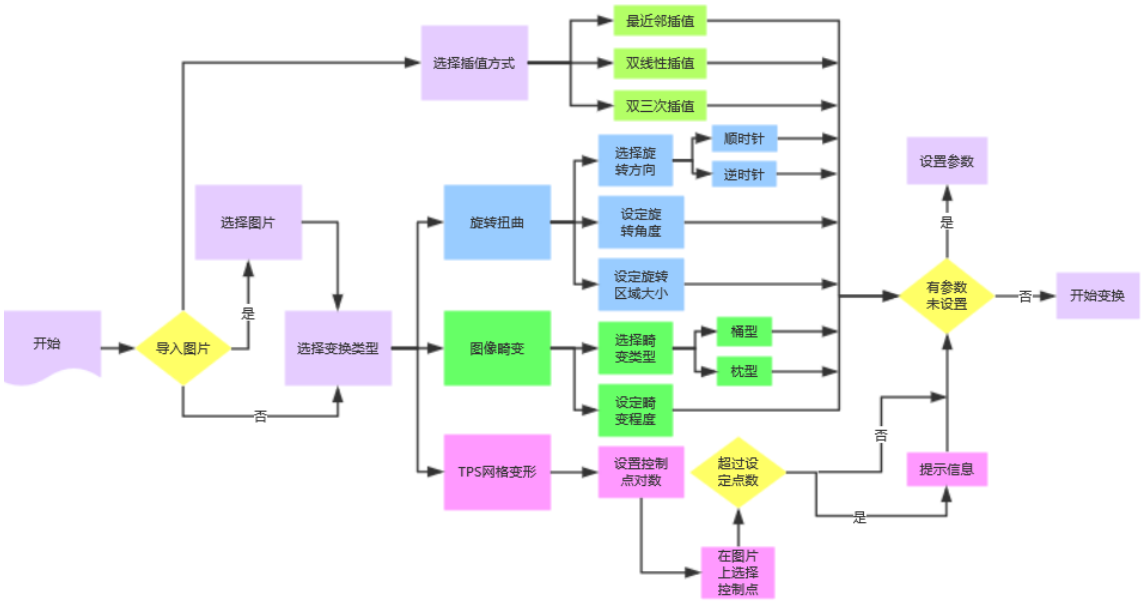
实验需要有一个可展示的界面来对功能进行展示，并且程序在运行中具有基本的鲁棒性，保证程序避免崩溃、停止响应等情况。

五、方案原理及设计

1.整体思路

对于变换后的图，其上的每一点都可以通过坐标变换函数$f(x,y)$的逆变换在原图上找到对应的点；由于某些逆变换后的点的坐标不是整数，因此需要进行插值。把插值之后得到的像素值赋给变换后的坐标，即可得到变换后的图片。

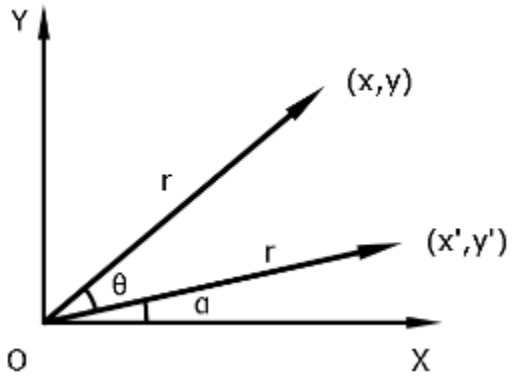
2.程序框图



程序总流程图[1]

3.变换函数

（1）旋转扭曲



旋转[2]变换

对于平面直角坐标系中任意一个点(x,y)，求它绕旋转中心(x_0,y_0)逆时针旋转θ之后的坐标(x',y')，根据数学关系我们可以得到如下公式：

[1] 流程图地址：https://www.processon.com/view/link/582d1d6be4b0645c0ea6979c

[2] 图片来源：http://www.cnblogs.com/cv-pr/p/4850465.html

$$x' = (x - x_0)\cos\theta - (y - y_0)sin\theta + x_0$$

$$y' = (x - x_0)sin\theta + (y - y_0)\cos\theta + y_0$$

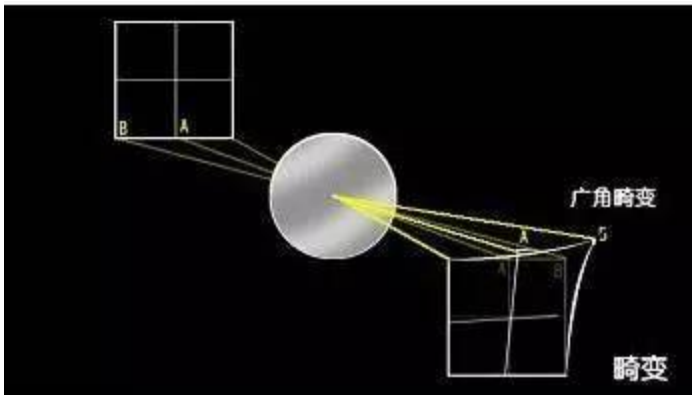
当旋转方向为顺时针时，θ取负值。

对于本次旋转扭曲而言，设置图像中心点为旋转原点，偏转角度θ从原点处向外衰减，并在距离原点无限远处衰减为 0，从而把从原点出发的一条射线变换成一条从原点螺旋向外的曲线。在这次大作业中，衰减量由用户选择的旋转区域大小和旋转程度来决定。

在算法实现中，顺时针旋转和逆时针旋转互为逆变换，通过(x', y')可以求出其反向旋转对应的原图上的点(x, y)。

（2）图像畸变

物体按照理想针孔模型成像，所得的图像为理想图，由于实际的光学镜头与理想针孔模型不完全一直，因此所成的图像有畸变，成为畸变图（如下图所示）：



畸变[3]

假定理想图中的点(x, y)在畸变图中对应坐标为(x', y')，(x, y)与(x', y')的映射关系为

$$x' = s(x, y)$$

$$y' = t(x, y)$$

工程中一般使用多项式来近似这两个函数。假定多项式的次数为k，则

$$x' = \hat{s}(x, y) = \sum_{i=0}^{k} \sum_{j=0}^{k-i} u_{ij} x^i y^j$$

[3] 图片来源：http://www.360doc.com/content/16/0229/08/248984_538176303.shtml

$$y' = \hat{t}(x,y) = \sum_{i=0}^{k} \sum_{j=0}^{k-i} v_{ij} x^i y^j$$

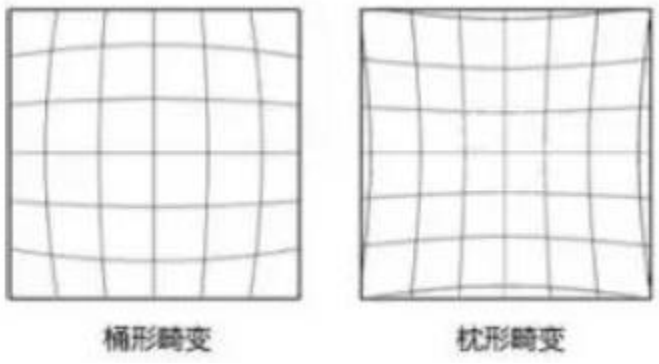
若图像只发生径向畸变，所用多项式次数必须在 3 次或者 3 次以上，需要执行大量运算，在实时图像处理系统中难以承受，因此可以采用分片近似算法的思想，即把图像分成很多矩形区域，在每个区域用矩形的 4 个顶点做月书店，辨识出适合这个区域的一次模型，运算量大大降低。而在径向畸变这样的非线性畸变中，忽略高阶项可以得到如下关系：

$$x' = x(1 + kr^2)$$

$$y' = y(1 + kr^2)$$

这是对于畸变中心为（0,0）点而言的，r为(x,y)到畸变中心的距离。

对于公式中的k称为畸变因子，如果为桶形畸变，$k < 0$；如果为枕形畸变，$k > 0$。



桶形畸变和枕形畸变[4]

在实际的算法设计中，以图片中心(x_0, y_0)为畸变中心，有：

$$x' - x_0 = (x - x_0) \times (1 + kr^2)$$

$$y' - y_0 = (y - y_0) \times (1 + kr^2)$$

而对应的逆变换为：

$$x = \frac{x' - x_0}{(1 + kr^2)} + x_0$$

$$y = \frac{y' - y_0}{(1 + kr^2)} + y_0$$

从而得到原图上的点的坐标。

[4] 图片来源：http://www.360doc.com/content/16/0229/08/248984_538176303.shtml

（3）TPS 网格变形

对于 TPS 网格变形的坐标变换函数的设计，课件中讲的很清楚，此处不再赘述。

给定n个控制点 $P_1=(x_1,y_1),\mathrm{L}\ ,P_n=(x_n,y_n)$ ，记

$$K=\begin{bmatrix} 0 & U(r_{12}) & \mathrm{L} & U(r_{1n}) \\ U(r_{21}) & 0 & \mathrm{L} & U(r_{2n}) \\ \mathrm{L} & \mathrm{L} & \mathrm{L} & \mathrm{L} \\ U(r_{n1}) & U(r_{n2}) & \mathrm{L} & 0 \end{bmatrix} \quad P=\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \mathrm{L} & \mathrm{L} & \mathrm{L} \\ 1 & x_n & y_n \end{bmatrix} \quad L=\begin{bmatrix} K & P \\ P^T & \boldsymbol{0} \end{bmatrix}$$

假设目标点为 $P_1'=(x_1',y_1'),\mathrm{L}\ ,P_n'=(x_n',y_n')$，记

$$V=\begin{bmatrix} x_1' & x_2' & \mathrm{L} & x_n' \\ y_1' & y_2' & \mathrm{L} & y_n' \end{bmatrix} \quad Y=\begin{pmatrix} V \mid 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}^T$$

则 $f(x,y)=\left[f_x(x,y),f_y(x,y)\right]^T=\boldsymbol{a}_1+\boldsymbol{a}_x x+\boldsymbol{a}_y y+\sum_{i=1}^{n}\boldsymbol{w}_i U\left(\left|P_i-(x,y)\right|\right)$

其中 $\boldsymbol{a}_1,\boldsymbol{a}_x,\boldsymbol{a}_y,\boldsymbol{w}$ 为线性方程组 $L\left[\boldsymbol{w}_1,\mathrm{L}\ ,\boldsymbol{w}_n,\boldsymbol{a}_1,\boldsymbol{a}_x,\boldsymbol{a}_y\right]^T=Y$ 的解。

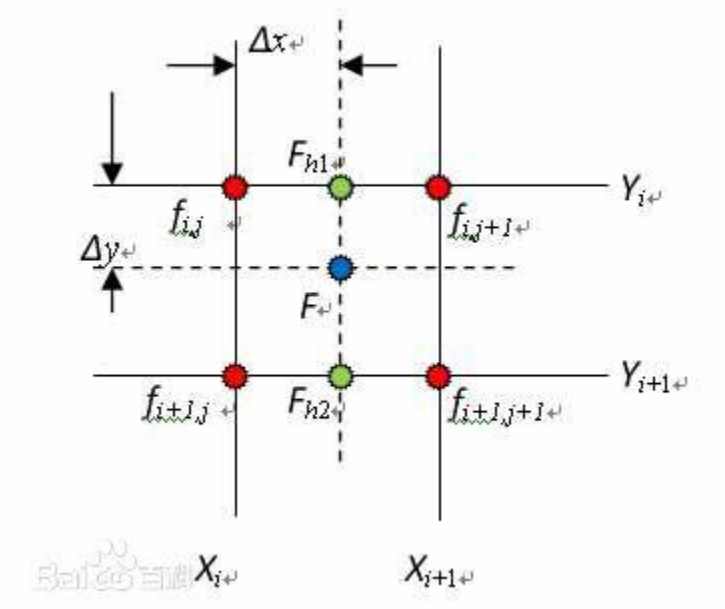
$$U(r)=\begin{cases} r^2\log\left(r^2\right), r\neq 0 \\ 0, \quad r=0 \end{cases}$$

如前所述，算法实现中，依然是通过求坐标变换函数的逆变换来得到目标点在原图中的对应点。在 TPS 网格变形中，把目标点当做原图上的点带入上述公式即可。

3.插值函数的设计

（1）最近邻插值

最近邻插值即把离目标点最近的整数坐标点的像素值赋给目标点，如下图：



插值[5]

如上图，F是通过坐标变换求得的原图中对应的点，其横纵坐标为小数；离它最近的四个整数坐标点分别为$f_{i,j}$、$f_{i,j+1}$、$f_{i+1,j}$、$f_{i+1,j+1}$，把这四个点中离F最近的点的像素值赋给F，即：

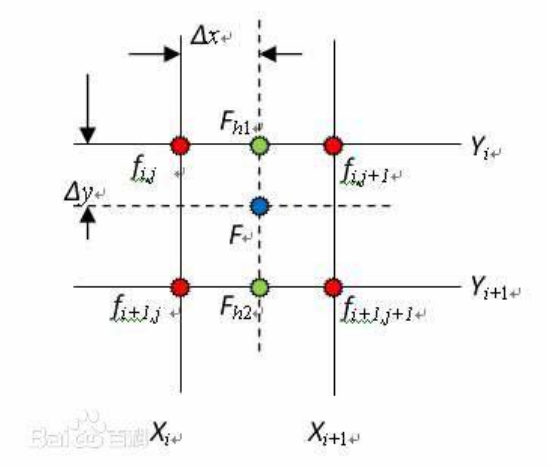
如果$\Delta x < 0.5 \& \Delta y < 0.5, F(x,y) = f_{i,j}(x,y);\ \Delta x > 0.5 \& \Delta y < 0.5, F(x,y) = f_{i,j+1}(x,y);$

$\Delta x > 0.5 \& \Delta y > 0.5, F(x,y) = f_{i+1,j+1}(x,y);\ \Delta x < 0.5 \& \Delta y > 0.5, F(x,y) = f_{i+1,j}(x,y);$

亦即：$F(x,y) = F([x+0.5],[y+0.5])$.

由于插值时只用到了原图上一个点的像素值，故插值后的图片上可能出现锯齿状。

（2）双线性插值



双线性插值又叫双线性内插，其核心思想就是在x和y两个方向上分别进行一次线性插值。如上图，由坐标变换求得的原图上的F的像素值为其以相对于上下左右四个点的距离加权的各点像素值之和，即：

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

这里对于待插值点的像素值用到了周围四个点的像素值进行插值，因此光滑性比只用了一个点的最近邻插值好，但是计算显然更复杂。

[5] 图片来源：http://baike.baidu.com/view/3038019.htm

（3）双三次插值

在双三次插值中，带插值点的像素值由 16 个点的像素值加权平均得到。先对横坐标方向每行的四个点的像素值进行拉格朗日插值，得到纵坐标方向上的 4 个点，之后对此 4 点的像素值做拉格朗日插值，最后得到目标点(x, y)的像素值。具体公式如下：

$$f(i+u, j+v) = ABC^T$$

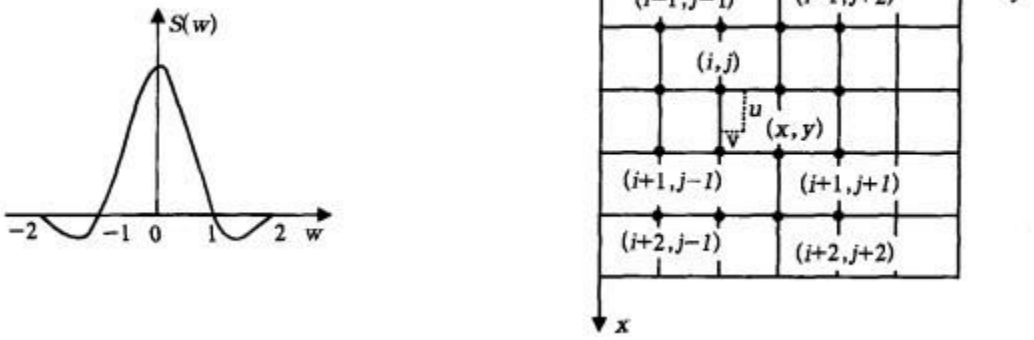
$$A = \begin{bmatrix} S(u+1) & S(u) & S(u-1) & S(u-2) \end{bmatrix}$$

$$C = \begin{bmatrix} S(v+1) & S(v) & S(v-1) & S(v-2) \end{bmatrix}$$

$$B = f(i-1:i+2, j-1:j+2)$$

其中$S(x)$为三次插值核函数，可由如下式子近似：

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



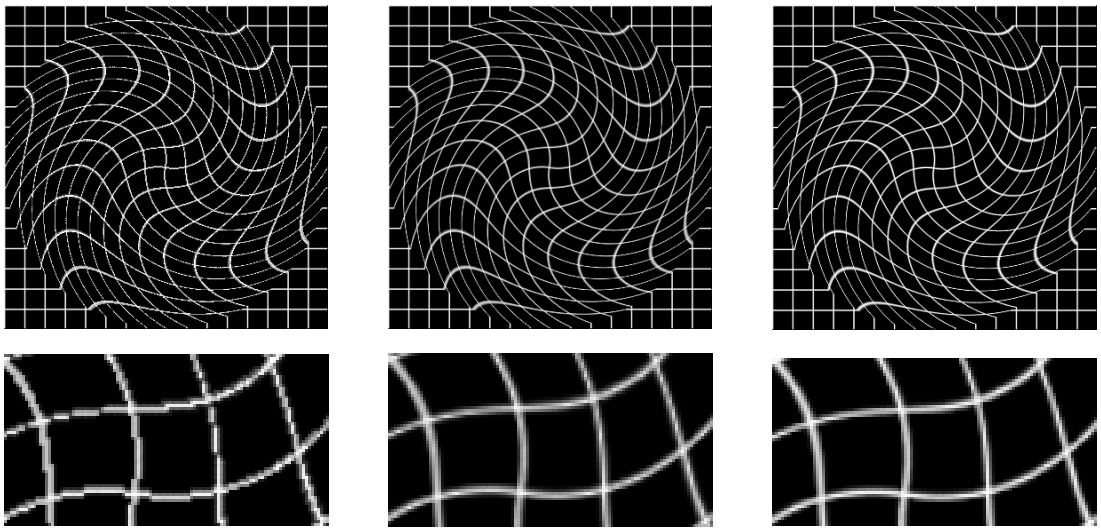
双三次插值[6]

双三次插值与前面两种插值方式相比，用到了更多的原图上的点的信息，计算量更大，但是效果也更好。

[6] 图片来源：http://www.cnblogs.com/wjgaas/p/3597095.html

六、结果比较

1.旋转扭曲

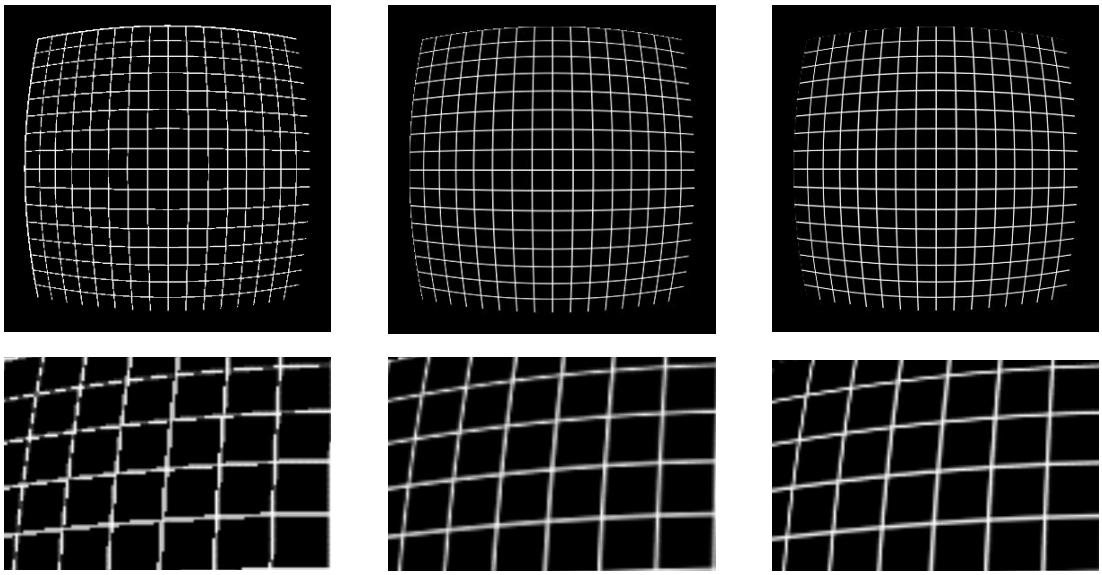


最近邻插值　　双线性插值　　双三次插值

如图，最近邻插值由于只使用了周围一个点的像素值，锯齿状比较明显，计算量小，响应很快；双线性插值用了周围 4 个点的像素值，已经基本没有锯齿状，响应也比较快；双三次插值使用了更多的点进行插值，平滑性最好，但是计算量大，响应时间很长。

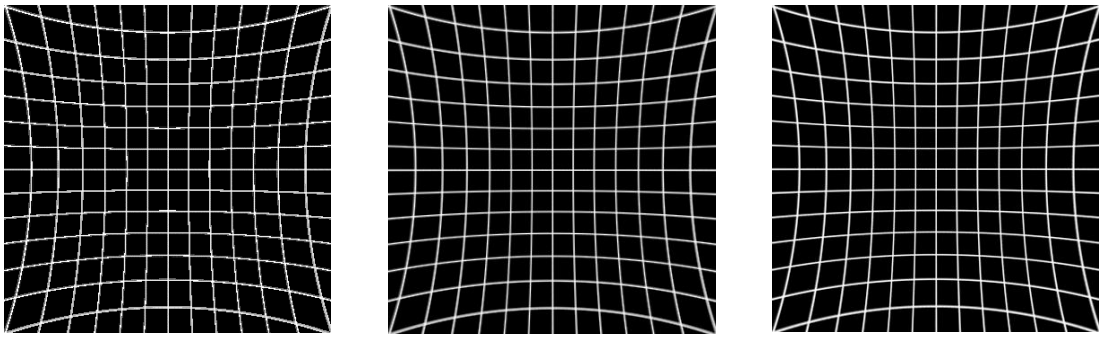
2.图像畸变

桶型：



最近邻插值　　双线性插值　　双三次插值

枕型：

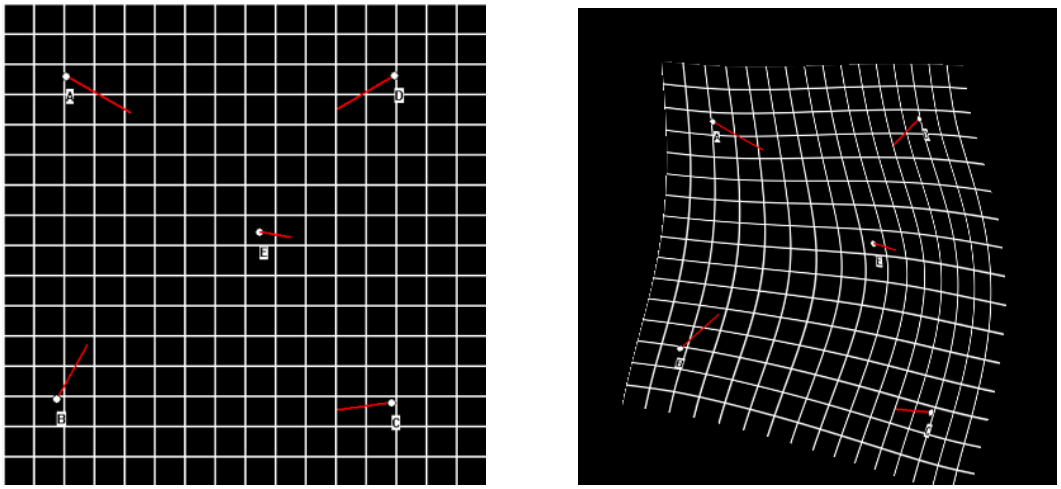


最近邻插值　　双线性插值　　双三次插值

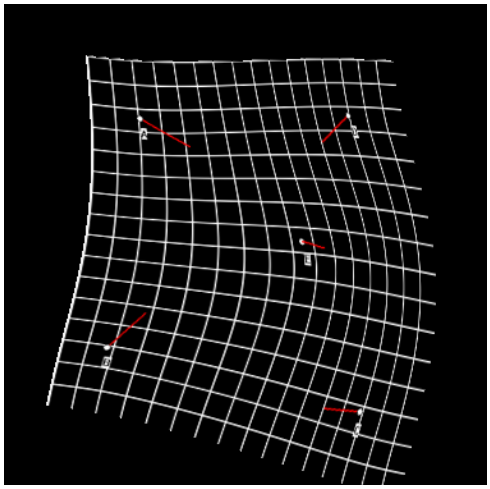
畸变效果依然是最近邻插值<双线性插值<双三次插值。

但是从桶型畸变我们可以看出，畸变后图像的完整性最近邻插值>双线性插值>双三次插值.原因是最近邻插值只用到了一个点，所以变换范围就是整张图片；而双线性插值用到了前后左右的两个点，故变换范围是$[1, width-1] \times [1, height-1]$，图像边界的信息缺失了；双三次插值用到了前面两点和后面两点的像素值，变换范围是$[2, width-2] \times [2, height-2]$，边界的信息缺失的更多了。

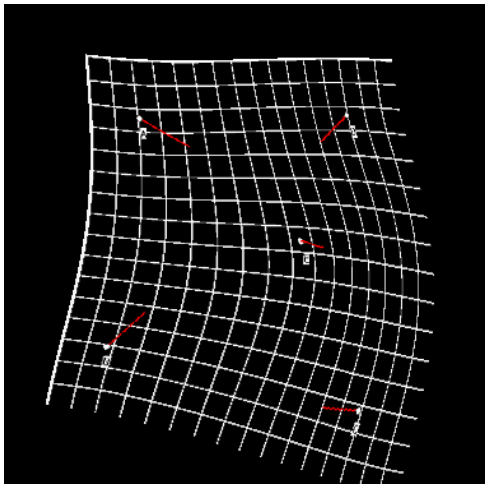
3.TPS 网格变形



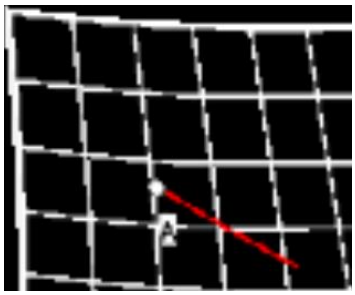
原图及变换点　　双三次插值



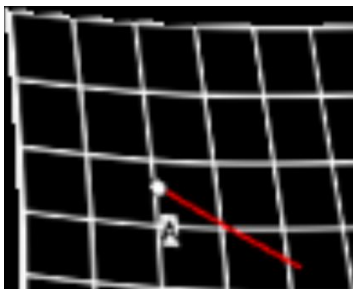
双线性插值



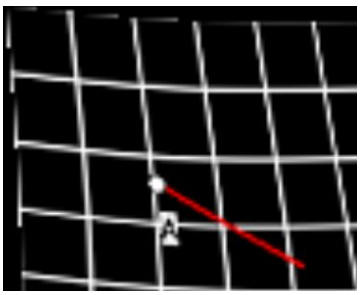
最近邻插值



最近邻插值



双线性插值



双三次插值

对各插值方法的分析同上。

七、误差分析

由于图像由计算机处理，故不存在观测误差；三种变换所涉及的模型也都可认为是合理的。下面分析方法误差和舍入误差。

1.方法误差

插值是用整数坐标点的像素值近似小数坐标点的像素值，本身存在误差。

假设插值函数各阶导数连续，对其在插值点上进行 Taylor 展开。

（1）最近邻插值

在最近邻插值中，使用距离待插值点(x',y')最近的整数(x,y)的像素值替代(x',y')的像素值，则有$R(x,y)=f(x',y')-f(x,y)=O(u)$.

（2）双线性插值

如前所述，双线性插值是先对x方向插值，再对y方向插值；二者是叠加的关系。以x方向的插值为例，待插值点(x',y)的像素值由(x,y)的像素值和$(x+1,y)$的像素值插值得到，将插值函数在(x,y)上展开，有$f(x',y)=f(x,y)+u\times f'(x,y)+O(u^2)$，故

$$\begin{aligned}R(x,y)&=f(x',y)-(1-u)f(x,y)-uf(x+1,y)\\&=f(x',y)-f(x,y)-u\big(f(x+1,y)-f(x,y)\big)\\&=f(x',y)-f(x,y)-\ u\times f'(x,y)=O(u^2).\end{aligned}$$

（3）双三次插值

计算方法同上，对插值函数三阶 Taylor 展开可得$R(x,y)=O(u^3)$。

2.舍入误差

摄入误差主要由计算机存储位数带来。

（1）C#中的 double 类型是 64bit，只有 16 位，$\Delta\le 0.5\times 10^{-16}$。

（2）在 C#中，主要使用了 Bitmap 和 Bitmap Data 类，对于图片像素的 RGB 三个通道的存储都是 8 位二进制，从 0 到 255，故对于每个像素点的每个通道，$\Delta\le 0.5$。

①最近邻插值：用距离待插值点(x',y')最近的整数(x,y)的像素值替代(x',y')的像素值，故$\Delta\le 0.5$。

②双线性插值：$\Delta\le(1-u)(1-v)\times 0.5+(1-u)v\times 0.5+(1-v)u\times 0.5+uv\times 0.5\le 4\times 0.5^3=0.5.$

③双三次插值：$\Delta\le\big(S(1+u)+S(u)+S(u-1)+S(u-2)\big)\times 0.5\times\big(S(1+v)+S(v)+S(v-1)+S(v-2)\big)\le 1\times 0.5\times 1=0.5.$

（3）C#中 Math.PI=3.1415926535897931,故$\Delta\le 0.5\times 10^{-16}$。

参考文献：光学图象几何畸变的快速校正算法

http://wenku.baidu.com/link?url=mvtE_6Noxmo158w8a_8TmPc_PQ7cW8g1-ux2s2f3sQsCyFtbgFY7AlyrcChk613I-LmT4wyQVhMXPpytG9OB6my4yokqApfhR4foZFai-qC