数值分析大作业一

孙大伟 2014010782
https://github.com/sundw2014/numericalAnalysis1

需求分析

所需功能：输入一张图片，对其进行变换，最后输出变换后的图像
插值算法：可以选择最近邻、双线性、双三次插值方法中的一种
变换：可以选择扭曲、畸变、TPS变形中的一种

方案设计

符号说明

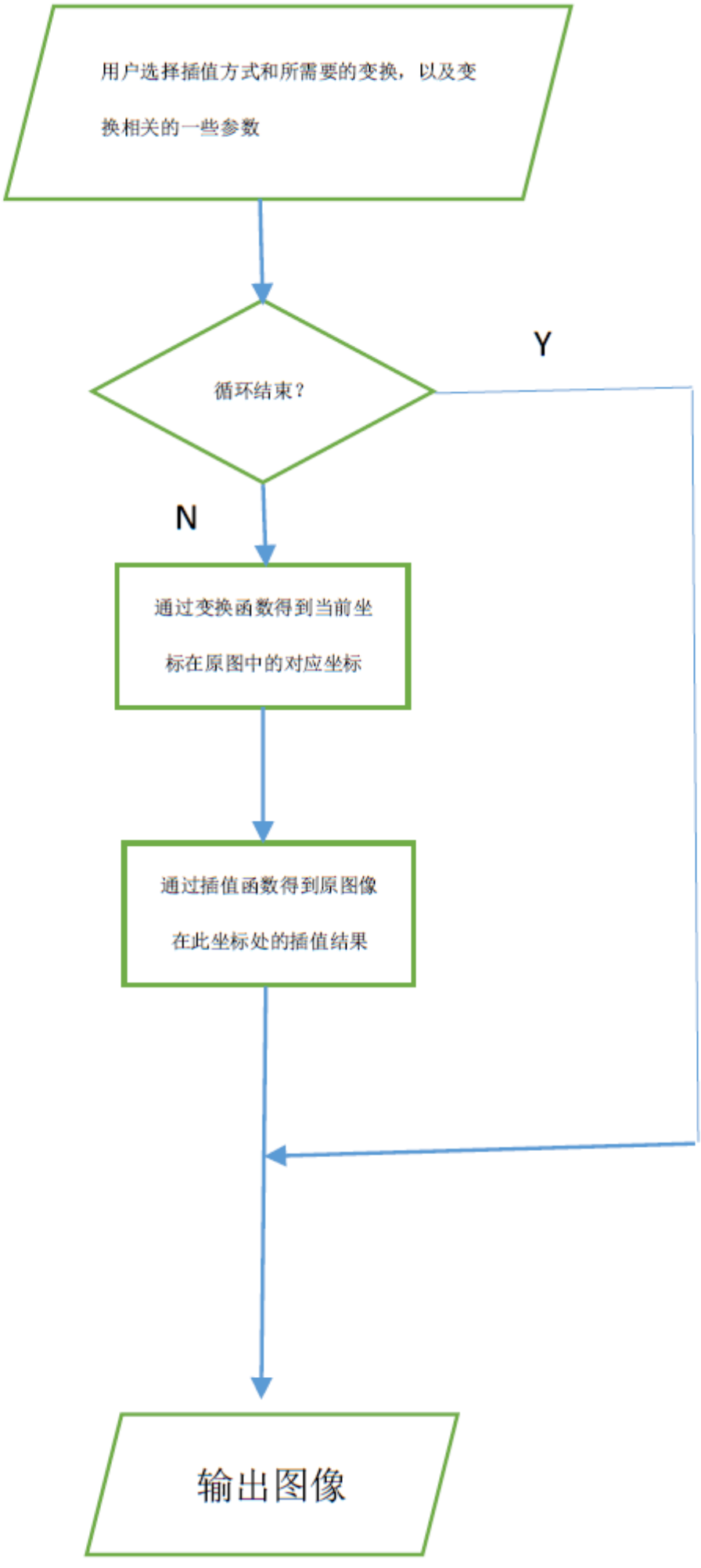
I_i表示输入图像，I_O表示输出图像，$f(x,y)$表示变换所对应的映射，用于将输出图像中的坐标映射到输入图像中。

基本思路

算法的目的是求输出图像中各像素点的值$I_O(x,y)$，其中x,y是两个整数值，即像素的索引。
按照如下方法计算$I_O(x,y)$：

1. 得到I_I中对应的坐标$(x_I,y_I)=f(x,y)$
2. 通过插值函数和原图像得到插值结果$\tilde{I}_I(x_I,y_I)$
3. 令$I_O(x,y)=\tilde{I}_I(x_I,y_I)$

流程图


用户选择插值方式和所需要的变换，以及变换相关的一些参数
循环结束？
Y
N
通过变换函数得到当前坐标在原图中的对应坐标
通过插值函数得到原图像在此坐标处的插值结果
输出图像

方案具体原理

插值算法

符号说明

插值算法的输入是输入图像I_I中的坐标(x_I, y_I)和此坐标周围的一些像素点的值，输出是插值结果$\tilde{I}_I(x_I, y_I)$

最近邻插值

对于坐标(x_I, y_I)，与其最为接近的整数点为$(round(x_I), round(y_I))$,其中$round(x)$是对x求近似（四舍五入），所以得到

$$\tilde{I}_I(x_I, y_I) = I_I(round(x_I), round(y_I))$$

双线性插值

双线性插值算法可以用矩阵运算来描述

$$\tilde{I}_I(i+u, j+v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} I_I(i,j) & I_I(i,j+1) \\ I_I(i+1,j) & I_I(i+1,j+1) \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}$$

双三次插值

双三次插值算法可以用矩阵运算来描述

$$\tilde{I}_I(i+u, j+v) = ABC^T$$

其中

$$A = \begin{bmatrix} S(u+1) & S(u) & S(u-1) & S(u-2) \end{bmatrix}$$
$$C = \begin{bmatrix} S(v+1) & S(v) & S(v-1) & S(v-2) \end{bmatrix}$$
$$B = I_I(i-1:i+2, j-1:j+2)$$

$S(x)$为三次插值的核函数，可以按照如下公式近似

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

其中，根据经验a取$-0.5 \sim -0.7$，本次试验中取-0.6

变换算法

符号说明

(x_O, y_O)是I_O中的直角坐标，(x_I, y_I)是I_I中的直角坐标；(r_O, α_O)是I_O中的极坐标，(r_I, α_I)是I_I中的极坐标；

扭曲

扭曲变换的公式如下：

$$x_O = r_I * cos\left(\alpha_I + \theta * \frac{row - r_I}{row}\right), y_O = r_I * sin\left(\alpha_I + \theta * \frac{row - r_I}{row}\right)$$

其中row为最大旋转半径，θ为最大旋转角度。
从上述公式可知$\alpha_O = \alpha_I + \theta * \frac{row - r_I}{row}$，$r_O = r_I$。从而易得反变换的公式为

$$\alpha_I = \alpha_O - \theta * \frac{row - r_O}{row}\ ，r_I = r_O$$
$$x_I = r_I * cos\alpha_I, y_I = r_I * sin\alpha_I$$

畸变

在本次试验中畸变的模型选为三个参数的径向畸变，描述如下

$$r_I = r_O(1 + k_1 r_O^2 + k_2 r_O^4 + k_3 r_O^6), \alpha_i = \alpha_O$$

$$x_I = r_I cos(\alpha_I), y_I = r_I sin(\alpha_I)$$

TPS

用户通过点击鼠标输入两个点集：

$\left\{(x_O^1, y_O^1)(x_O^2, y_O^2)(x_O^3, y_O^3)\ldots(x_O^N, y_O^N)\right\}$ 为输出图像中的坐标点

$\left\{(x_I^1, y_I^1), (x_I^2, y_I^2), (x_I^3, y_I^3)\ldots(x_I^N, y_I^N)\right\}$ 为输入图像中的坐标点

想要寻找一个函数，在以上两个集合的对应点之间建立映射关系，即将输出图像中的坐标(x_O^k, y_O^k)，$k = 1, 2, 3...N$映射到输入图像中的坐标(x_I^k, y_I^k)，为了得到这样的函数，我们分别考虑x和y坐标。
先考虑x坐标，我们要找到一个函数$f(x, y)$，满足:

$$f(x_O^i, y_O^i) = x_I^i - x_O^i\ ，i = 1, 2, 3...N$$

为了使映射光滑，假设函数$f(x, y)$具有以下形式：

$$f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{N} \omega_i U(\left\|(x_O^i, y_O^i) - (x, y)\right\|)$$

其中$U(x) = x^2 log(x)$，为了使映射形成的曲面能量最小，再添加一些限制条件，最终得到：

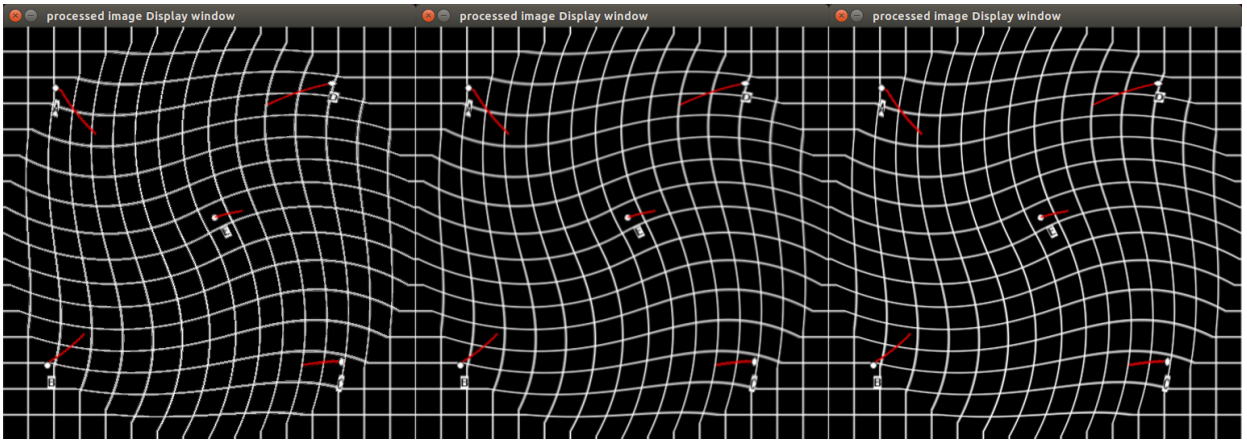
$$\begin{cases} f(x_O^i, y_O^i) = x_I^i - x_O^i \ , \ i = 1, 2, 3...N \\ \sum_{i=0}^{N} \omega_i = 0 \\ \sum_{i=0}^{N} \omega_i x_i = 0 \\ \sum_{i=0}^{N} \omega_i y_i = 0 \end{cases}$$

共$N+3$个方程，$N+3$个未知数，可以解出$f(x,y)$，假设得到函数$f_X(x,y)$，同理也可以通过相同方法得到$f_Y(x,y)$。
从而可以根据以下关系得到输出图像中的任意点在输入图像中的对应点：

$$x_I = f_X(x_I, y_I) + x_O \ , \ y_I = f_Y(x_O, y_O) + y_O$$

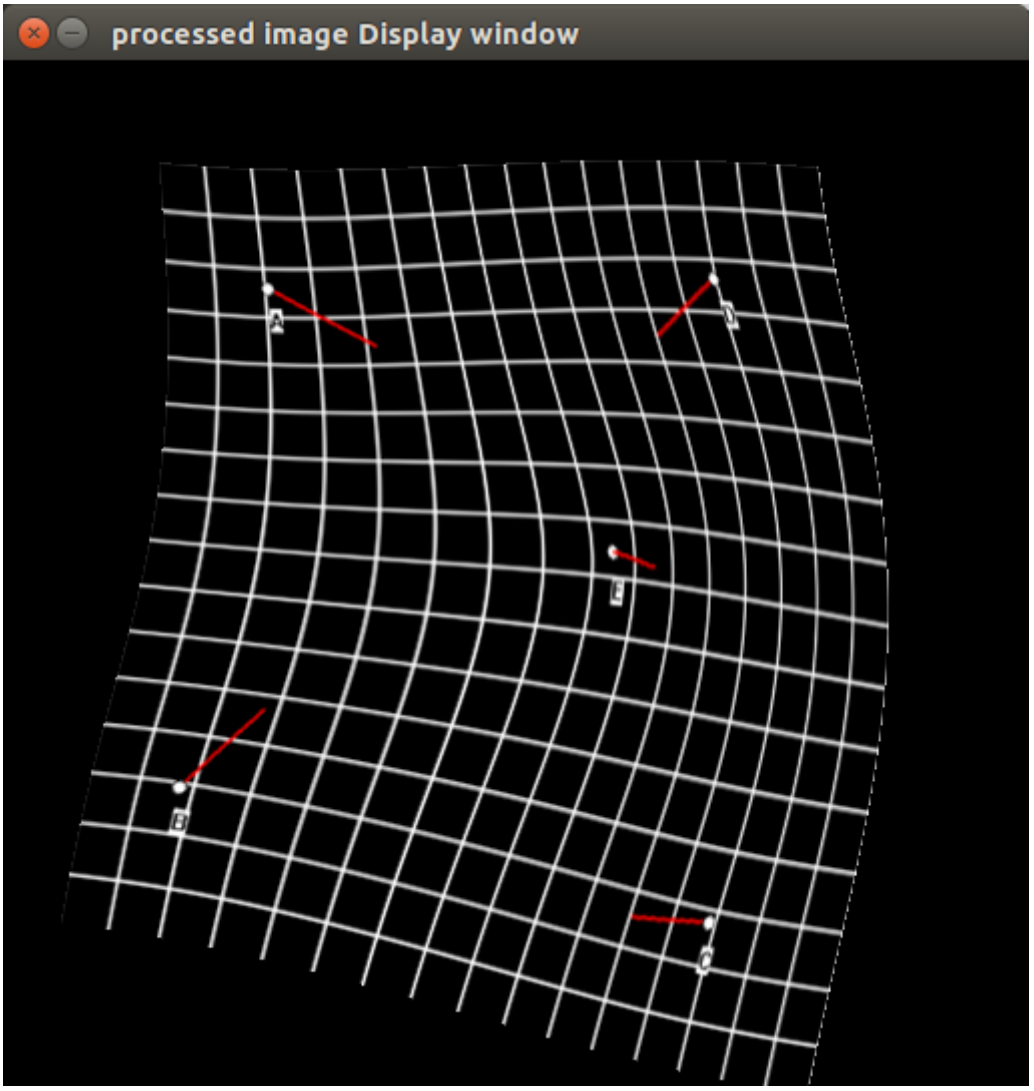
结果分析

不同插值算法的结果对比



从左至右分别为最近邻、双线性、双三次插值算法对30度扭曲畸变插值的结果，可以看出最近邻插值中有很多锯齿，双线性和双三次插值有效地减少了锯齿，且双三次插值的效果更好一些。

TPS插值结果



上图是作业要求中的图做TPS变换后的效果，与作业要求中的图接近。

性能表现

因为所有数学函数都是自己实现的，没有考虑任何加速，所以计算效率比较低。在TPS+双三次插值时每张图片的处理都需要一秒钟左右。

误差分析

边缘处的插值

在边缘处，如果使用最近邻插值，则插值的结果有相同的可能性取到边缘左侧的值和右侧的值，当左右相差较大时，就会造成毛刺现象，比如上图对网格插值的结果；使用双线性和双三次插值时，会用到边缘两侧的点的加权平均，插值出来的结果方差比较下，毛刺就被消除了。所以使用多个差值节点插值可以起到平滑边缘的效果。

矩阵运算

观察到TPS算法中产生的矩阵中的元素最大和最小值相差很大，在求逆等运算中，会因为大数加小数而造成精度损失。

Reference

> Gianluca Donato and Serge Belongie http://cseweb.ucsd.edu/~sjb/pami_tps.pdf
> WikiPedia Gussian elimination
> https://en.wikipedia.org/wiki/Gaussian_elimination#CITEREFGolubVan_Loan1996
> Golub, Gene H. and Van Loan and Charles F. (1996), Matrix Computations (3rd ed.), Johns Hopkins, ISBN 978-0-8018-5414-9.