# 数值分析大作业：人脸变形

自 64　赵文亮　2016011452

2018 年 11 月 15 日

## 目录

# 1 引言

本次数值分析大作业的目标是编写程序对人脸图像进行扭曲变形。程序的输入为待修改人脸和目标人脸两张图片，输出为一张按照以目标人脸为模板修改后的人脸。本文中的人脸变形算法主要分为三个部分：人脸关键点检测、变形函数、插值，第 2 节至第 4 节将会分别介绍。第 5 节对本文算法的整体流程进行了详细的描述；第 6 节展示了程序使用方法和变形结果；最后对算法流程中产生的误差进行分析。

# 2 变形函数

本文实现了两种变形函数，分别为 TPS 变形和 B 样条变形，分别介绍如下：

## 2.1 TPS 变形

TPS（Thin plate spline，薄板样条插值）是一种有效的插值模型，它可以产生无穷阶可导的光滑曲面。给定 $n$ 个控制点 $P_1(x_1, y_1), \dots, P_n(x_n, y_n)$，以及函数在这些点上的取值 $f(P_i), 1 \le i \le n$，TPS 的目标是寻找一个插值函数 $f_{\text{tps}}$，使得 $f_{\text{tps}}$ 通过所有的控制点，并且能量函数 $I_f$ 达到最小。

$$I_f = \iint_{R^2} \left( \left( \frac{\partial^2 f}{\partial x^2} \right)^2 + 2 \left( \frac{\partial^2 f}{\partial x \partial y} \right)^2 \left( \frac{\partial^2 f}{\partial y^2} \right)^2 \right) \mathrm{d}x \, \mathrm{d}y \tag{2.1}$$

本文中由于需要将一组二维点映射到另一组二维点，故实际上相当于进行了两次 TPS 插值。设目标点为 $P'_1(x'_1, y'_1), \dots, P'_n(x'_n, y'_n)$，插值基函数

$$U(r) = \begin{cases} r^2 \log(r^2), & r \ne 0 \\ 0, & r = 0 \end{cases} \tag{2.2}$$

令

$$K = \begin{bmatrix} 1 & U(r_{12}) & \cdots & U(r_{1n}) \\ U(r_{21}) & 0 & \cdots & U(r_{2n}) \\ \vdots & \vdots & \ddots & \vdots \\ U(r_{n1}) & U(r_{n2}) & \cdots & 0 \end{bmatrix} \quad P = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & y_n \end{bmatrix} \quad L = \begin{bmatrix} K & P \\ P^T & O \end{bmatrix} \tag{2.3}$$

其中 $r_{ij} = \|P_i - P_j\|$ 为两个控制点之间的欧式距离。记

$$V = \begin{bmatrix} x'_1 & x_2 & \cdots & x'_n \\ y'_1 & y_2 & \cdots & y'_n \end{bmatrix} \quad Y = \left[ \begin{array}{c|ccc} V & 0 & 0 & 0 \\ & 0 & 0 & 0 \end{array} \right]^{\mathrm{T}} \tag{2.4}$$

构造并求解方程

$$L[\boldsymbol{w}_1, \dots, \boldsymbol{w}_n, \boldsymbol{a}_1, \boldsymbol{a}_x, \boldsymbol{a}_y]^{\mathrm{T}} = Y \tag{2.5}$$

得到 $\boldsymbol{a}, \boldsymbol{a}_x, \boldsymbol{a}_y, \boldsymbol{w}$，则有

$$\boldsymbol{f}_{\text{tps}}(x, y) = [f_{\text{tps},x}(x, y), f_{\text{tps},y}(x, y)]^{\mathrm{T}} = \boldsymbol{a}_1 + \boldsymbol{a}_x x + \boldsymbol{a}_y y + \sum_{i=1}^{n} \boldsymbol{w}_i U(|P_i - (x, y)|) \tag{2.6}$$

式 2.6 即为 TPS 变形函数的求解结果。

### 2.2 B 样条变形

B 样条曲线是一种由 B 样条基函数线性组合得到的曲线。三次 B 样条的基函数为:

$$\begin{cases} B_0(t) = \frac{1}{6}(-t^3 + 3t^2 - 3t + 1) \\ B_1(t) = \frac{1}{6}(3t^3 - 6t^2 + 4) \\ B_2(t) = \frac{1}{6}(-3t^3 + 3t^2 + 3t + 1) \\ B_3(t) = \frac{1}{6}t^3 \end{cases} \quad t \in [0, 1] \tag{2.7}$$

本文中采用了 [1] 中的基于 B 样条的 FFD 算法（Free-Form Deformation）。设图片宽度为 $m$, 高度为 $n$。构造点集 $\boldsymbol{\Phi}$ 为 $((m-1)/h+3) \times ((n-1)/h+3))$ 的网格，$\phi(i,j)$ 表示网格上第 $ij$ 个控制点。变形函数 $\boldsymbol{w}$ 可以表示为:

$$\boldsymbol{w}(u, v) = \sum_{k=0}^{3} \sum_{l=0}^{3} B_k(s) B_l(t) \phi_{(i+k)(j+l)} \tag{2.8}$$

其中

$$\begin{cases} i = \left\lfloor \frac{u}{h} \right\rfloor - 1 \\ j = \left\lfloor \frac{v}{h} \right\rfloor - 1 \\ s = \frac{u}{h} - \left\lfloor \frac{u}{h} \right\rfloor \\ t = \frac{v}{h} - \left\lfloor \frac{v}{h} \right\rfloor \end{cases} \tag{2.9}$$

初始状态下未经过变形的图像中的点可以表示为

$$\boldsymbol{w}^0(u, v) = (u, v) = \sum_{k=0}^{3} \sum_{l=0}^{3} B_k(s) B_l(t) \phi^0_{(i+k)(j+l)} \tag{2.10}$$

设 $\Delta\phi_{ij} = \phi_{ij} - \phi^0_{ij}$，初始状态下图形中的某一点 $p$ 变换后为 $q = \boldsymbol{w}(p)$，二者差值为 $\Delta q = \boldsymbol{w}(p) - \boldsymbol{w}^0(p) = q - p$，则有

$$\Delta q = \sum_{k=0}^{3} \sum_{l=0}^{3} w_{kl} \phi_{(i+k)(j+l)} \tag{2.11}$$

其中 $w_{kl} = B_k(s)B_l(t)$。$\Delta\phi_{kl}$ 由下式选取:

$$\Delta\phi_{kl} = \frac{w_{kl}\Delta q}{\sum_{a=0}^{3}\sum_{b=0}^{3} w_{ab}^2} \tag{2.12}$$

由于需要将图像中的一个点集通过变形函数映射到另一个点集，设源点集为 $P$，目的点集为 $Q$，采用算法 1 即可计算出来对应的栅格控制点的偏移量。其中对 $\Delta\phi_{ij}$ 进行截断使得 $|\Delta\phi_{ij}|_\infty \leq 0.48$ 是为了保证映射的一一对应。计算出控制点后，再从式 (2.8) 即可计算得到映射后点的坐标。

# 3 插值函数

插值总体上分为正向插值和反向插值两种思路。正向插值是指将原图中的所有像素点经过变形函数映射到目标图片，得到许多散点，再对没有被赋值的点进行插值；反向插值是指将目标图片中的坐标经过变形函数

**算法 1:** Free-Form Deformation

```
Input   : points sets P and Q
Output: control point displacements ΔΦ = {Δφ_ij}
for all i, j do
    let δ_ij = 0 and ω_ij = 0;
end
for each point p = (u, v) in P do
    let i = ⌊u/h⌋ − 1 and j = ⌊v/h⌋ − 1;
    let s = u/h − ⌊u/h⌋ and t = v/h − ⌊v/h⌋;
    for k, l = 0, 1, 2, 3 do
        compute Δφ_kl with (2.12);
        add w_kl^2 Δφ_kl to δ_(i+k)(j+l);
        add w_kl^2 to ω_(i+k)(j+l)
    end
end
for all i, j do
    if ω_ij ≠ 0 then
        compute Δφ_ij = δ_ij / ω_ij;
        truncate Δφ_ij so thad |Δφ_ij|_∞ ≤ 0.48;
    else
        let Δφ_ij = 0;
    end
end
```

反向映射回原始图像的坐标，再对这个点进行插值。事实证明，正向插值算法复杂度较高，而反向插值不仅复杂度低，而且可以方便地实现多种插值算法，故本文最终采用了反向插值。本节将介绍基于反向插值的三种插值方法，并在最后简述正向插值的思路。

## 3.1 反向插值

设从目标图像 $I'$ 中的点 $(x', y')$ 经变形函数反向映射后得到原始图像 $I$ 中的坐标 $(x, y)$，下面使用三种插值方法求解 $I'(x', y')$。

### 3.1.1 最近邻

直接采用 $I$ 中距离 $(x, y)$ 最近的像素点的值，即

$$I'(x', y') = I(\lfloor x + 0.5 \rfloor, \lfloor y + 0.5 \rfloor) \tag{3.1}$$

### 3.1.2 双线性

设 $(x, y)$ 落在由点 $p_{11}(x_1, y_1), p_{12}(x_1, y_2), p_{21}(x_2, y_1), p_{22}(x_2, y_2)$ 组成的方格中（包括边界），则有

$$I'(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} I(x_1, y_1) & I(x_1, y_2) \\ I(x_2, y_1) & I(x_2, y_2) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix} \tag{3.2}$$

双线性插值可以看成分别在 $x$ 和 $y$ 方向上做一次线性插值。

### 3.1.3 双三次

双三次插值考虑 $(x, y)$ 所落在的 $4 \times 4$ 网格中。令

$$\begin{aligned} \alpha &= x - \lfloor x \rfloor \\ \beta &= y - \lfloor y \rfloor \end{aligned} \tag{3.3}$$

设

$$\begin{aligned} A &= \begin{bmatrix} S(\alpha + 1) & S(\alpha) & S(\alpha - 1) & S(\alpha - 2) \end{bmatrix}^T \\ C &= \begin{bmatrix} S(\beta + 1) & S(\beta) & S(\beta - 1) & S(\beta - 2) \end{bmatrix}^T \end{aligned} \tag{3.4}$$

其中

$$S(x) = \begin{cases} 1 - 2|x|^2 + |x|^3 & |x| \le 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3 & 1 < |x| < 2 \\ 0 & otherwise \end{cases} \tag{3.5}$$

由 $4 \times 4$ 网格点取值构造矩阵

$$B = \begin{bmatrix} I(\lfloor x \rfloor - 1, \lfloor y \rfloor - 1) & I(\lfloor x \rfloor - 1, \lfloor y \rfloor) & I(\lfloor x \rfloor - 1, \lfloor y \rfloor + 1) & I(\lfloor x \rfloor - 1, \lfloor y \rfloor + 2) \\ I(\lfloor x \rfloor, \lfloor y \rfloor - 1) & I(\lfloor x \rfloor, \lfloor y \rfloor) & I(\lfloor x \rfloor, \lfloor y \rfloor + 1) & I(\lfloor x \rfloor, \lfloor y \rfloor + 2) \\ I(\lfloor x \rfloor + 1, \lfloor y \rfloor - 1) & I(\lfloor x \rfloor + 1, \lfloor y \rfloor) & I(\lfloor x \rfloor + 1, \lfloor y \rfloor + 1) & I(\lfloor x \rfloor + 1, \lfloor y \rfloor + 2) \\ I(\lfloor x \rfloor + 2, \lfloor y \rfloor - 1) & I(\lfloor x \rfloor + 2, \lfloor y \rfloor) & I(\lfloor x \rfloor + 2, \lfloor y \rfloor + 1) & I(\lfloor x \rfloor + 2, \lfloor y \rfloor + 2) \end{bmatrix} \tag{3.6}$$

则插值结果

$$I'(x', y') = ABC^T \tag{3.7}$$

## 3.2 正向插值简述

由于正向变换后，目标图像上只有一些不规则的散点处有像素值，给插值带来了不便。本节将简述正向插值中最近邻插值实现方法[1]。

### 3.2.1 最近邻插值

正向变换后得到的是一系列散点，所以不能直接通过简单的四舍五入找到最近邻。基于 k-d 树可以实现复杂度较低的最近邻查找。如果 $n$ 为所有已知点的数目（即为源图像的像素数目），则查找最近邻的复杂度为 $\mathcal{O}(n\log n)$。事实上，基于 k-d 树和优先级队列可以实现 knn（k-nearest neighbors）算法，可以在 $\mathcal{O}(n\log n)$ 的时间内搜索与待插值点最近的 $k$ 个已知点。

# 4 人脸关键点检测

本文使用 OpenCV 的中的扩展模块 face[2] 中的相关 API 来识别输入图片中的人脸关键点。该算法通过HELEN 数据集训练出一个模型，在运行时调用预训练模型并加载级联分类器，即可完成人脸和关键点的检测。

# 5 算法流程

前文介绍了算法中用到的变形函数和插值方法，本节将对算法的流程进行详细的描述。

## 5.1 关键点检测与读取

程序运行时，会首先读取与输入图片对应的人脸关键点数据。如果找不到关键点数据，则调用人脸关键点检测函数，生成数据并保存。设待修改人脸和目标人脸的关键点分别为 $P_S$ 和 $P_D$。

## 5.2 变形与插值

本节将分别介绍使用 TPS 变形和 B 样条变形的算法流程。

### 5.2.1 TPS 变形

由于 TPS 变形对图像整体都有影响，所以我采用了先求取目标图像坐标范围，再反过来求解目标图像像素值的方式。具体流程如下：

1. 求取从 $P_S$ 到 $P_D$ 的 TPS 变形函数，记作 $TPS_1$
2. 使用 $TPS_1$ 对待修改人脸的四个角点进行变换，得到输出图像的坐标范围
3. 求取从 $P_D$ 到 $P_S$ 的 TPS 变形函数，记作 $TPS_2$
4. 对输出图像坐标范围内的每一个坐标 $(x',y')$ 使用 $TPS_2$ 变换，得到其对应的坐标 $(x,y)$
5. 调用插值函数获得 $I'(x',y')=I(x,y)$

[1] 我最初实现的版本即为正向插值，后来由于复杂度过高将其改进为反向插值。

[2] `https://github.com/opencv/opencv_contrib`

### 5.2.2 B 样条变形

B 样条变形只对图像的局部有影响，故可以认为目标图像的范围和原图像的范围相同。具体流程如下：

1. 求解源关键点和目标关键点的 MBR[3]，分别记作 $MBR_S$ 和 $MBR_D$
2. 对 $MBR_D$ 进行平移和缩放，使得 $MBR_D$ 的长度与 $MBR_S$ 的长度相等，且 $MBR_D$ 的 $MBR_S$ 的中心点重合。由此可以得到变换后的目标关键点，记作 $P'_D$
3. 求解从 $P'_D$ 到 $P_S$ 的 B 样条变形函数 $W$
4. 对输出图像中的每一个点 $(x', y')$，使用变形函数 $W$ 求出其对应的 $(x, y)$
5. 调用插值函数获得 $I'(x', y') = I(x, y)$

# 6 实验结果

## 6.1 程序界面

程序的初始界面如图 1 所示。用户可以选择变形函数或插值方法，在 B 样条中，用户可以指定控制网格的间隔。中间的三个区域分别为待修改人脸、目标人脸和修改后人脸。

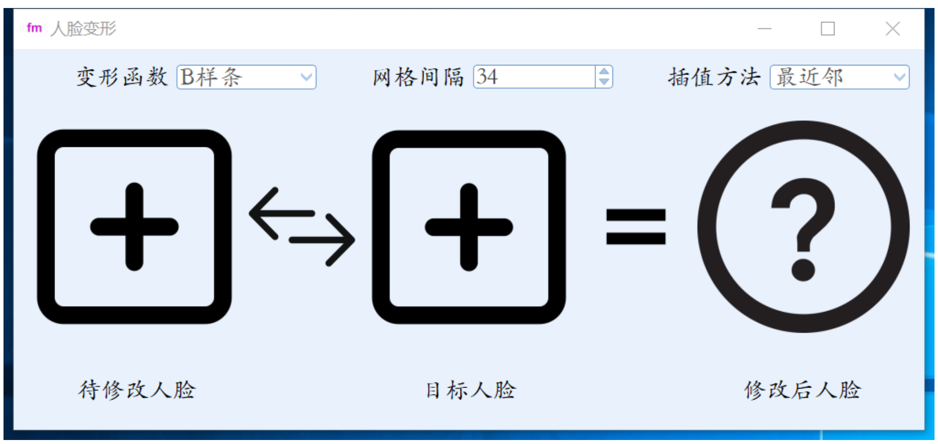


图 1: 程序初始界面

点击程序中几种符号的作用如下：

- 加号：载入待修改人脸或目标人脸图片。载入成功后可以点击对应图片再次修改；
- 互换符号：交换待修改人脸和目标人脸图片；
- 等号：开始变形；
- 问号：指定输出图片的保存路径。如果未指定，则会根据输入图片的路径自动生成路径。

[3] Minimum Bounding Rectangle，最小外接矩形

### 6.2 人脸变形结果

图 2 展示了从数据集中的 6.jpg 到 9.jpg 的变形结果，分别使用了 TPS 和 B 样条变形。对比可见，TPS 变形函数作用于全局，导致图像的边界也发生了变化；B 样条变形函数作用于局部。另一方面，由于 B 样条变形函数改变的是控制点的位置，并不能保证变形后人脸关键点完全重合。


(a) TPS 变形


(b) B 样条变形

图 2: 两种变形函数比较（6.jpg → 9.jpg 变形结果）

图 3 展示了使用 B 样条变形从数据集中的 8.jpg 到 7.jpg 的变形结果，分别使用了最近邻、双线性、双三次插值方法。由于图片中像素较多，三种插值方法看起来效果都较好。实验中发现，三种算法消耗时间为最近邻 < 双线性 < 双三次。

## 7 误差分析

误差的来源分为模型误差、观测误差、舍入误差和方法误差。在数值分析中，我们不讨论模型误差和观测误差，只研究用数值方法求解问题过程中的误差 [2]。本节将着重分析舍入误差和方法误差。

(a) 最近邻 (b) 双线性 (c) 双三次

图 3: 三种插值方式对比（8.jpg → 7.jpg 变形结果）

### 7.1 舍入误差

- 数字图像处理的过程中将图片的像素值采样后存储在计算机中，其 RGB 值存在一个离散化的过程，假设每个通道像素值取值范围均为 $0 \sim 255$，则由于采样间隔为 1，带来的舍入误差 $R \le \frac{1}{2}$。
- 由于计算机字长有限，在计算过程中由于四舍五入会产生误差。在程序中关于坐标的计算均采用 double 类型变量，故单次计算中的误差量级为 $10^{-16}$，这个数值乘以计算的次数即可用来估计累积的误差，求得累计误差的量级约为 $10^{-15}$。
- 最终求取像素值时，由于将 double 类型转换为 $0 \sim 255$ 之间的整数，存在舍入误差 $R \le \frac{1}{2}$.

### 7.2 方法误差

方法误差主要存在于插值的过程中，三种方法的插值公式在第 3 节中已经给出，下面进行误差分析。

#### 7.2.1 最近邻插值

设点 $(x, y)$ 处的像素值 $A = I(x, y)$,最近邻误差在 $x$ 和 $y$ 方向都进行了四舍五入,所以 $|\Delta x| \le \frac{1}{2}, |\Delta y| \le \frac{1}{2}$，则误差为

$$\begin{aligned} R = |\Delta A| &\le \max\left|\frac{\partial I}{\partial x}\right| \cdot |\Delta x| + \max\left|\frac{\partial I}{\partial y}\right| \cdot |\Delta y| \\ &= \frac{1}{2}\left(\max\left|\frac{\partial I}{\partial x}\right| + \max\left|\frac{\partial I}{\partial y}\right|\right) \end{aligned} \tag{7.1}$$

#### 7.2.2 双线性插值

由于双线性插值可以看成对 $x$ 方向和 $y$ 方向先后进行线性插值，所以首先求解一维的插值误差。设 $u = x - \lfloor x \rfloor, v = y - \lfloor y \rfloor$，则有

$$\begin{cases} R_x \leq \dfrac{1}{2} \max \left| \dfrac{\partial^2 I}{\partial x^2} \right| \cdot |u(u-1)| \leq \dfrac{1}{8} \max \left| \dfrac{\partial^2 I}{\partial x^2} \right| \\ R_y \leq \dfrac{1}{2} \max \left| \dfrac{\partial^2 I}{\partial y^2} \right| \cdot |v(v-1)| \leq \dfrac{1}{8} \max \left| \dfrac{\partial^2 I}{\partial y^2} \right| \end{cases} \tag{7.2}$$

则总误差

$$R \leq |R_x + R_y| \leq \frac{1}{8} \left( \max \left| \frac{\partial^2 I}{\partial x^2} \right| + \max \left| \frac{\partial^2 I}{\partial y^2} \right| \right) \tag{7.3}$$

#### 7.2.3 双三次插值

双三次插值可以看成对 $x$ 方向和 $y$ 方向先后进行三次样条插值。一维时，有以下公式[4]

$$\begin{cases} R_x \leq \dfrac{5}{384} \max \left| \dfrac{\partial^4 I}{\partial x^4} \right| \\ R_y \leq \dfrac{5}{384} \max \left| \dfrac{\partial^4 I}{\partial y^4} \right| \end{cases} \tag{7.4}$$

则总误差为

$$R \leq |R_x + R_y| \leq \frac{5}{384} \left( \max \left| \frac{\partial^4 I}{\partial x^4} \right| + \max \left| \frac{\partial^4 I}{\partial y^4} \right| \right) \tag{7.5}$$

# 8 总结

通过完成本次大作业，我收获到了很多知识和技能。下面按照时间顺序简要总结：

- 首先是配置 OpenCV 环境。OpenCV 环境按照网上的教程配置下来并不困难，但是由于我一开始就想要完成附加题中的人脸关键点检测，所以我需要同时配置 OpenCV-Contrib 的环境。而 OpenCV-Contrib 的 Github 中只有源码，需要 CMake 从源码编译。于是我采用这个方式成功地配置了环境，学会了使用 CMake 进行源码编译的方法。

- 程序中关于矩阵的运算，我使用自己编写的 Mat 类完成。我新建了一个 namesapce，名为 mycv，之后便可以使用 mycv::Mat 进行矩阵操作。接口设计上，我力求于 OpenCV 中的 cv::Mat 类的接口一致。这个过程中，我复习了 C++ 中关于动态内存分配、运算符重载、复制构造函数等内容，这些在矩阵类中都是必不可少的。

- 程序的界面我使用 Qt 完成。本次大作业中，我第一次在 VS 中使用 Qt。我继承了 QLabel 类编写了 ClickLabel 类，重写鼠标点击事件，得到了一个可以点击的 Label。这样就可以既在上面显示图片，又可以点击触发相应操作。

- 算法设计的过程中，我学会了 TPS、B 样条变形函数，更加熟悉了三种插值算法的特点。在之前刚刚结束的数图大作业（全景图像拼接）中，我学以致用，将 TPS 用到了一个插值问题中，得到了很好的效果。B 样条变形函数的编写是比较困难的，我查阅了大量资料，最终找到了一个适合于本次作业要求的算法，这一过程也锻炼了我查阅文献、复现算法的能力。

[4] 见教材 46 页

- 在最后的误差分析的过程中，我重温了拉格朗日插值和埃尔米特插值的相关内容，从而能够更加深入理解双线性插值和双三次插值。

# 参考文献

[1] S. Lee, G. Wolberg, K.-Y. Chwa, and S. Y. Shin, “Image metamorphosis with scattered feature constraints,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 4, pp. 337–354, 1996.

[2] 李庆扬, 王能超, 易大义, 数值分析. 清华大学出版社, 2008.