

# 图像旋转缩放

数值分析与算法 第一次大作业

刘柏

自 36 | 2013011548

# 目录

1.	实验内容.....	3
2.	需求分析.....	3
2.1.	图像处理.....	3
2.2.	用户交互.....	3
3.	方案设计.....	3
3.1.	模块设计.....	4
3.1.1.	图片读取与存储.....	4
3.1.2.	操作界面.....	4
3.1.3.	插值计算.....	4
3.2.	程序框图.....	4
4.	基本原理与误差分析.....	5
4.1.	最近邻插值.....	5
4.1.1.	原理.....	5
4.1.2.	误差分析.....	5
4.2.	双线性插值.....	6
4.2.1.	原理.....	6
4.2.2.	误差分析.....	6
4.3.	双三次插值.....	7
4.3.1.	原理.....	7
4.3.2.	误差分析.....	8
5.	实现方法.....	9
5.1.	图像缩放实现.....	9
5.2.	图像旋转实现.....	9
5.3.	插值方法实现.....	10
6.	实验结果.....	10
6.1.	程序实现情况.....	10
6.1.1.	基本情况.....	10
6.1.2.	基本功能.....	10
6.1.3.	附加功能.....	11
6.2.	插值效果对比.....	14
7.	问题与解决方法.....	15

7.1.	边界溢出问题.....	16
7.2.	旋转方向问题.....	16
7.3.	双三次插值噪声问题.....	16
8.	实验体会.....	17

## 1. 实验内容

- (1) 目标：编写程序实现对图像的旋转和缩放；
- (2) 要求：图像旋转输入为任意角度，缩放尺度为任意倍数，横纵坐标缩放尺度可以不同。

## 2. 需求分析

### 2.1. 图像处理

在模拟图像（即可任意放大而不失真）中，不论是旋转还是缩放，原图像和目标图像的点是——对应的。此时对于目标图像中的任一点 $(r, c)$ ，只要 $(r, c)$ 根据映射关系求得原图像中对应的坐标 $(r_0, c_0)$ ，令 $f(r, c) = f(r_0, c_0)$ 即可。

但在数字图像中，像素点是离散的，坐标是整数。可能得到的 $(r_0, c_0)$ 不是整数，在原图像中没有直接对应像素点， $f(r_0, c_0)$ 需要通过利用周围的整点进行插值得到。故需要编写相应的插值计算模块。

### 2.2. 用户交互

一般的图像处理工具都有 GUI 界面，编写相应的操作界面能够大大方便日常所需的图像处理任务，故还需进行操作界面的 GUI 编写。

## 3. 方案设计

### 3.1. 模块设计

#### 3.1.1. 图片读取与存储

由于核心算法是用 C++ 语言开发的，相应的图形界面一般采用 MFC 或 Qt 框架，本项目采用 Qt 框架。由于 Qt 配置 OpenCV 相当麻烦，且 Qt 提供的 QImage 类也能方便地实现图片的读取与存储，故在图片读取与存储部分使用 Qt 的库。

#### 3.1.2. 操作界面

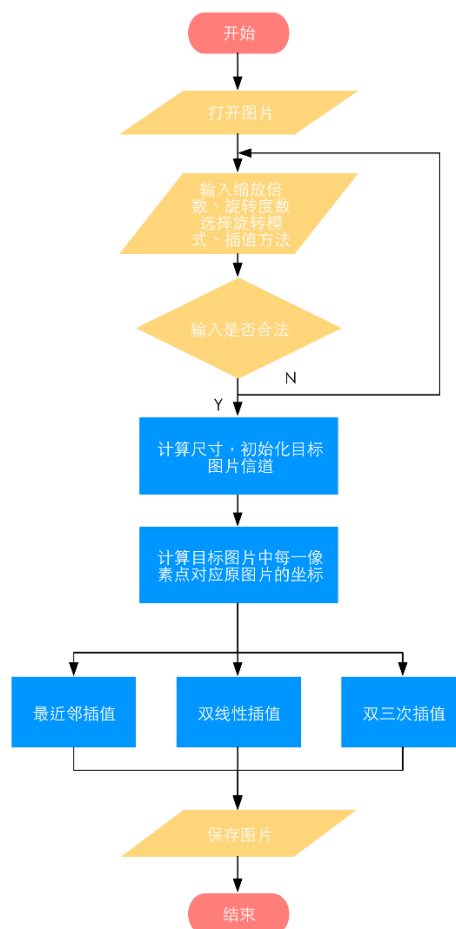
由于 Qt 具有简洁易用、跨平台的优势，该部分界面采用 Qt 框架进行编写。

#### 3.1.3. 插值计算

该部分主要功能为通过插值计算获取信道信息，进而生成目标图片。该部分采用 C++ 实现。

### 3.2. 程序框图

绘制程序框图如下。



## 4. 基本原理与误差分析

约定如下：某点坐标为 $(r, c)$ ，颜色为 $\mathbf{f}(r, c) = \begin{bmatrix} R(r, c) \\ G(r, c) \\ B(r, c) \end{bmatrix}$ ，对 $(r, c)$ 向下取整得到整点

坐标 $(i, j)$ ，定义 $\begin{cases} u = r - i \\ v = c - j \end{cases}$

### 4.1. 最近邻插值

#### 4.1.1. 原理

将离 $(r, c)$ 距离最近的像素点的颜色直接近似认为是 $(r, c)$ 的颜色。

在实现上，直接将离 $(r, c)$ 最近的整点的颜色赋给 $\mathbf{f}(r, c)$ 即可，即：

$$\mathbf{f}(r, c) = \mathbf{f}(r', c')$$

$$\text{其中 } r' = \begin{cases} i, & 0 \leq u < 0.5 \\ i + 1, & 0.5 \leq u < 1 \end{cases}, \quad c' = \begin{cases} j, & 0 \leq v < 0.5 \\ j + 1, & 0.5 \leq v < 1 \end{cases}$$

#### 4.1.2. 误差分析

以点 $(r_1, c_1)$ 为例进行分析。

##### (1) 观测误差

在对旋转操作进行插值时，由于取 $\pi = 3.1415926$ ，并非精确的 $\pi$ 值，故 $\pi$ 值处带来观测误差 $|\Delta x| \leq 0.5 \times 10^{-7}$ 。

##### (2) 模型误差

假如实际中的模拟图像颜色无畸变（即 $\mathbf{f}(r, c)$ 几乎处处连续），那么采样点足够多时，插值函数会无限趋近于 $\mathbf{f}(r, c)$ ，模型误差可忽略不计。

假如实际中的模拟图像颜色有畸变（即 $\mathbf{f}(r, c)$ 存在间断点或跳跃点），那么会存在模型误差。

##### (3) 方法误差

以 R 通道的插值为例。

可将最近邻插值视为二元的 0 阶拉格朗日插值，先在 r 方向上找到最近邻的整数，

固定  $c$ ，使  $R(r, c)$  一元化，变为关于  $r$  的函数。方法误差为：

$$|R_0(r)| \leq \max_{r \in [i, i+1]} \left| \frac{\partial R(r, c)|_{c=c_1}}{\partial r} \right| |r_1 - i| \leq \max_{r \in [i, i+1]} \left| \frac{\partial R(r, c)|_{c=c_1}}{\partial r} \right|.$$

再在  $c$  方向上找到最近邻的整数，固定  $r$ ，使  $R(r, c)$  一元化，变为关于  $c$  的函数。方法误差为：

$$|R_0(c)| \leq \max_{c \in [j, j+1]} \left| \frac{\partial R(r, c)|_{r=r_1}}{\partial c} \right| |c_1 - j| \leq \max_{c \in [j, j+1]} \left| \frac{\partial R(r, c)|_{r=r_1}}{\partial c} \right|.$$

$$\text{故总方法误差 } |R_0(r, c)| \leq \max_{r \in [i, i+1]} \left| \frac{\partial R(r, c)|_{c=c_1}}{\partial r} \right| + \max_{c \in [j, j+1]} \left| \frac{\partial R(r, c)|_{r=r_1}}{\partial c} \right|.$$

G 通道、B 通道同理。

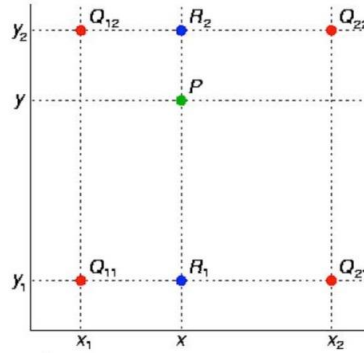
#### (4) 舍入误差

- 处理后获得的 RGB 通道值会被取为整数，故对于一个像素点而言，它的每条通道在被赋值时会有舍入误差  $|\Delta x| \leq 0.5$ ；
- 程序中使用 `float` 作为浮点数的类型，由于 `float` 有效数字为 6 位，故在使用浮点数时舍入误差  $|\Delta x| \leq 0.5 \times 10^{-5}$ 。

## 4.2. 双线性插值

### 4.2.1. 原理

双线性插值的核心思想就是在两个方向上依次进行插值。



如上图所示，先在  $x$  方向上对  $Q_{11}$ 、 $Q_{21}$  进行线性插值，得到  $R_1$ ，对  $Q_{12}$ 、 $Q_{22}$  进行线性插值，得到  $R_2$ ；再在  $y$  方向上对  $R_1$ 、 $R_2$  进行线性插值，得到  $P$ 。

具体到本例中，以 R 通道为例，有：

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

同理可求得  $G(r, c)$ 、 $B(r, c)$ ，相应的就得到了  $f(r, c)$ 。

### 4.2.2. 误差分析

以点 $(r_1, c_1)$ 为例进行分析。

(1) 观测误差

同 4.1.2 (1)

(2) 模型误差

同 4.1.2 (2)

(3) 方法误差

以 R 通道的插值为例。

双线性插值可视为分别在 r、c 方向上进行插值。

首先分别在直线  $c=j$ ,  $c=j+1$  上进行线性插值。固定 c, 使  $R(r, c)$  一元化, 变为关于 r 的函数。方法误差分别为:

$$|R_1(r)|_1 \leq \frac{\max_{r \in [i, i+1]} \left| \frac{\partial^2 R(r, c)|_{c=j}}{\partial r^2} \right|}{2!} |(r_1 - i)(r_1 - i - 1)| \leq \frac{\max_{r \in [i, i+1]} \left| \frac{\partial^2 R(r, c)|_{c=j}}{\partial r^2} \right|}{8},$$

$$|R_1(r)|_2 \leq \frac{\max_{r \in [i, i+1]} \left| \frac{\partial^2 R(r, c)|_{c=j+1}}{\partial r^2} \right|}{2!} |(r_1 - i)(r_1 - i - 1)| \leq \frac{\max_{r \in [i, i+1]} \left| \frac{\partial^2 R(r, c)|_{c=j+1}}{\partial r^2} \right|}{8};$$

接着再在直线  $r = r_1$  上进行线性插值, 固定 r, 使  $R(r, c)$  一元化, 变为关于 c 的函数。

方法误差为:

$$|R_1(c)| \leq \frac{\max_{c \in [j, j+1]} \left| \frac{\partial^2 R(r, c)|_{r=r_1}}{\partial c^2} \right|}{2!} |(c_1 - j)(c_1 - j - 1)| \leq \frac{\max_{c \in [j, j+1]} \left| \frac{\partial^2 R(r, c)|_{r=r_1}}{\partial c^2} \right|}{8}.$$

$$\text{故总方法误差 } |R_1(r, c)| \leq \frac{\max_{r \in [i, i+1]} \left| \frac{\partial^2 R(r, c)|_{c=j}}{\partial r^2} \right|}{8} + \frac{\max_{r \in [i, i+1]} \left| \frac{\partial^2 R(r, c)|_{c=j+1}}{\partial r^2} \right|}{8} + \frac{\max_{c \in [j, j+1]} \left| \frac{\partial^2 R(r, c)|_{r=r_1}}{\partial c^2} \right|}{8}.$$

G 通道、B 通道同理。

(4) 舍入误差

同 4.1.2 (4)

### 4.3. 双三次插值

#### 4.3.1. 原理

利用  $(r, c)$  附近的  $4 \times 4$  个像素点计算得到  $(r, c)$  处的颜色。

具体到本例中, 以 R 通道为例, 有:

$$R(r, c) = R(i + u, j + v) = ABC^T$$

其中  $A = [S(u+1) \ S(u) \ S(u-1) \ S(u-2)]$ ,

$$B = \begin{bmatrix} R(i-1, j-1) & R(i-1, j) & R(i-1, j+1) & R(i-1, j+2) \\ R(i, j-1) & R(i, j) & R(i, j+1) & R(i, j+2) \\ R(i+1, j-1) & R(i+1, j) & R(i+1, j+1) & R(i+1, j+2) \\ R(i+2, j-1) & R(i+2, j) & R(i+2, j+1) & R(i+2, j+2) \end{bmatrix},$$



$$C = [S(v+1) \quad S(v) \quad S(v-1) \quad S(v-2)],$$

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

同理可求得 $G(r, c)$ 、 $B(r, c)$ ，相应的就得到了 $f(r, c)$ 。

#### 4.3.2. 误差分析

##### (1) 观测误差

同 4.1.2 (1)

##### (2) 模型误差

同 4.1.2 (2)

##### (3) 方法误差

以点 $(r_1, c_1)$ 为例进行分析。

双三次插值可视为在  $r$  方向与  $c$  方向上分别进行三次插值。

首先分别在直线  $c=j-1$ ,  $c=j$ ,  $c=j+1$ ,  $c=j+2$  上进行三次插值。固定  $c$ , 使 $R(r, c)$

一元化, 变为关于  $r$  的函数。方法误差分别为:

$$\begin{aligned} |R(r)|_1 &\leq \frac{\max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j-1}}{\partial r^4} \right|}{4!} |(r_1 - i + 1)(r_1 - i)(r_1 - i - 1)(r_1 - i - 2)| \leq \\ &\frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j-1}}{\partial r^4} \right|; \\ |R(r)|_2 &\leq \frac{\max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j}}{\partial r^4} \right|}{4!} |(r_1 - i + 1)(r_1 - i)(r_1 - i - 1)(r_1 - i - 2)| \leq \\ &\frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j}}{\partial r^4} \right|; \\ |R(r)|_3 &\leq \frac{\max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j+1}}{\partial r^4} \right|}{4!} |(r_1 - i + 1)(r_1 - i)(r_1 - i - 1)(r_1 - i - 2)| \leq \\ &\frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j+1}}{\partial r^4} \right|; \\ |R(r)|_4 &\leq \frac{\max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j+2}}{\partial r^4} \right|}{4!} |(r_1 - i + 1)(r_1 - i)(r_1 - i - 1)(r_1 - i - 2)| \leq \\ &\frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j+2}}{\partial r^4} \right|. \end{aligned}$$

接着再在直线 $r = r_1$  上进行三次插值, 固定  $r$ , 使 $R(r, c)$ 一元化, 变为关于  $c$  的函数。

方法误差为:

$$\begin{aligned} |R(c)| &\leq \frac{\max_{c \in [j-1, j+2]} \left| \frac{\partial^4 R(r, c)|_{r=r_1}}{\partial c^4} \right|}{4!} |(c_1 - j + 1)(c_1 - j)(c_1 - j - 1)(c_1 - j - 2)| = \\ &\frac{9}{384} \max_{c \in [j-1, j+2]} \left| \frac{\partial^4 R(r, c)|_{r=r_1}}{\partial c^4} \right|. \end{aligned}$$

故总方法误差为：

$$\begin{aligned}
 |R_0(r, c)| &\leq \max_r \left| \frac{\partial R(r, c)}{\partial r} \right| + \max_c \left| \frac{\partial R(r, c)}{\partial c} \right| \\
 &= \frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j-1}}{\partial r^4} \right| + \frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j}}{\partial r^4} \right| + \\
 &\quad \frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j+1}}{\partial r^4} \right| + \frac{9}{384} \max_{r \in [i-1, i+2]} \left| \frac{\partial^4 R(r, c)|_{c=j+2}}{\partial r^4} \right| + \frac{9}{384} \max_{r \in [j-1, j+2]} \left| \frac{\partial^4 R(r, c)|_{r=r_1}}{\partial r^4} \right|.
 \end{aligned}$$

(4) 舍入误差

同 4.1.2 (4)

## 5. 实现方法

分析可知，在模拟图像（即可任意放大而不失真）中，不论是旋转还是缩放，原图像和目标图像的点是——对应的。此时对于目标图像中的任一点 $(r, c)$ ，只要 $(r, c)$ 根据映射关系求得原图像中对应的坐标 $(r_0, c_0)$ ，令 $f(r, c) = f(r_0, c_0)$ 即可。

但在数字图像中，像素点是离散的，坐标是整数。可能得到的 $(r_0, c_0)$ 不是整数，在原图像中没有直接对应像素点。因此， $f(r_0, c_0)$ 需要通过利用周围的整点进行插值得到。

### 5.1. 图像缩放实现

设纵向缩放倍数为 $\alpha$ ，横向缩放倍数为 $\beta$ ，则可得到映射关系：

$$\begin{cases} r_0 = \frac{r}{\alpha} \\ c_0 = \frac{c}{\beta} \end{cases}$$

对于整张图片而言，扫描目标图像每一像素点，将该点的坐标 $(r, c)$ 通过映射关系求出 $(r_0, c_0)$ 。利用插值手段求出 $(r_0, c_0)$ 的颜色 $f(r_0, c_0)$ 后，令 $f(r, c) = f(r_0, c_0)$ ，根据 $f(r, c)$ 生成目标图像即可。

### 5.2. 图像旋转实现

设顺时针旋转角度为 $\theta$ ：

$$\begin{cases} r_0 = r \cos \theta - c \sin \theta \\ c_0 = r \sin \theta + c \cos \theta \end{cases}$$

对于整张图片而言，扫描目标图像每一像素点，将该点的坐标 $(r, c)$ 通过映射关系求出 $(r_0, c_0)$ 。利用插值手段求出 $(r_0, c_0)$ 的颜色 $f(r_0, c_0)$ 后，令 $f(r, c) = f(r_0, c_0)$ ，根据 $f(r, c)$ 生成目标图像即可。

### 5.3. 插值方法实现

将 2.1、2.2、2.3 的公式编程实现即可。

需要注意的是，当 $(r_0, c_0)$ 位于边界时，在进行插值时可能会有数组下标越界的情况发生，为此需要进行判断。我自己的解决方案是：当数组下标超出范围时，将该点对应的值赋为最邻近边界点的值。

## 6. 实验结果

### 6.1. 程序实现情况

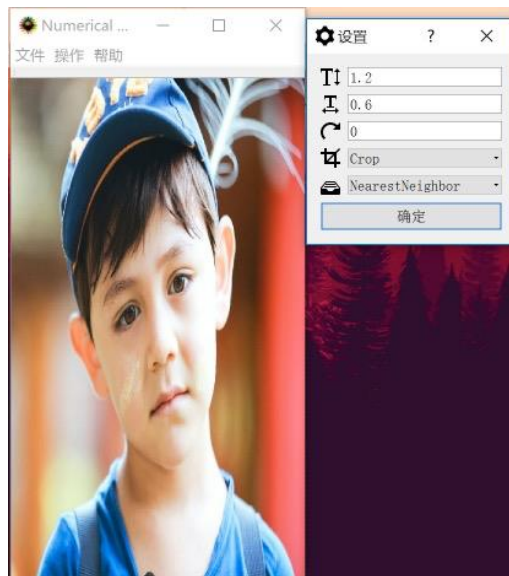
#### 6.1.1. 基本情况

- (1) 语言：C++;
- (2) 框架：Qt 5.5.1 (仅利用了其图像读取、存储库，GUI 控件库)
- (3) IDE: Qt Creator (Community)

#### 6.1.2. 基本功能

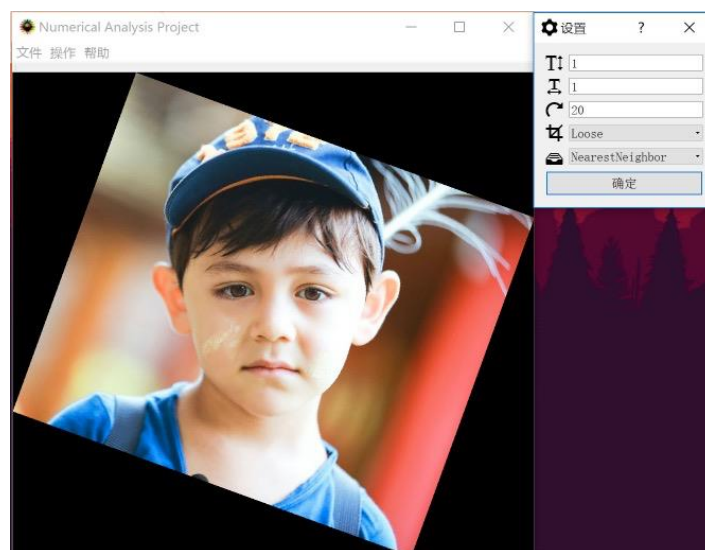
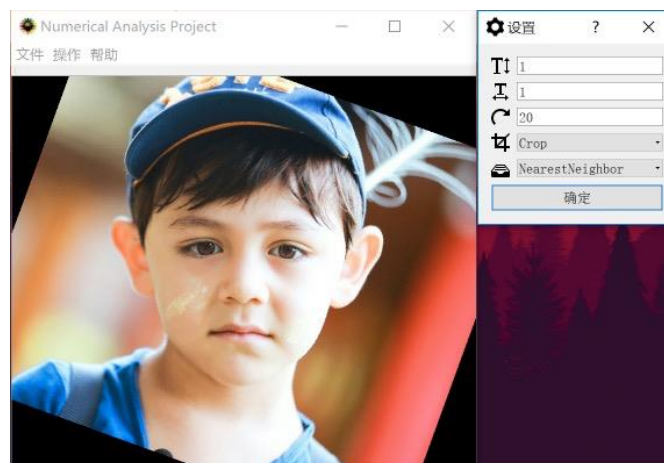
- (1) 图像缩放

如下图所示，能实现正实数倍的缩放。



## (2) 图像旋转

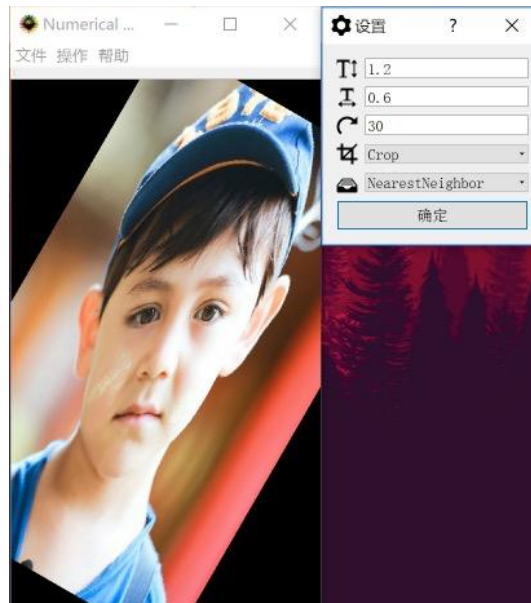
如下图所示，能实现两种模式下实数角度的旋转（注：本程序选取顺时针为正方向）。



## 6.1.3. 附加功能

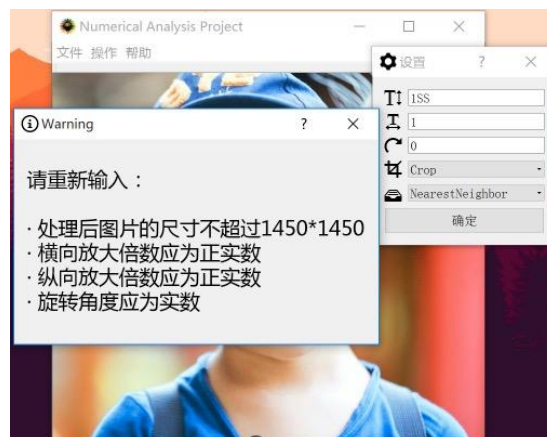
### (1) 缩放、旋转相结合

如下图所示，能实现缩放、旋转的一步到位。



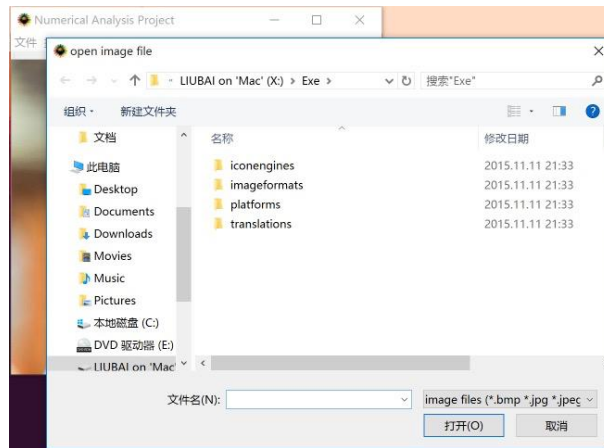
### (2) 非法输入警告

当想得到的图片尺寸过大（会导致程序崩溃）或输入含非法字符时，程序不运行并会弹出警告。



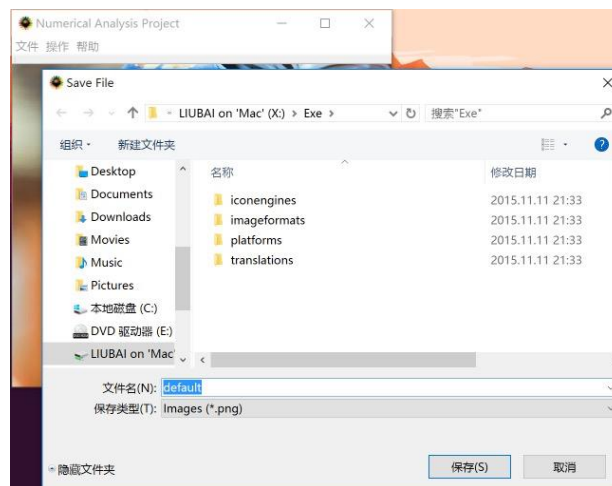
### (3) 打开图片

如下图所示。



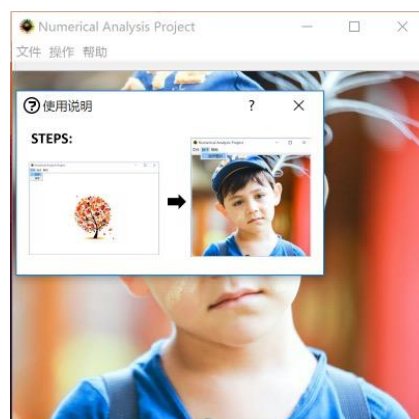
#### (4) 保存图片

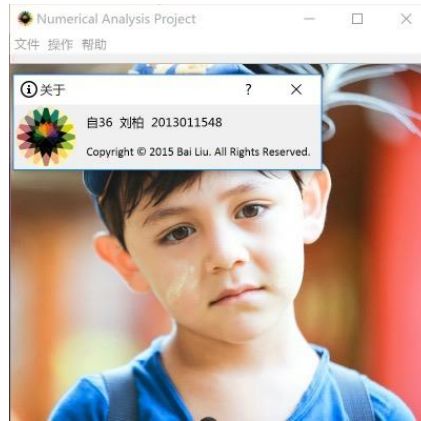
如下图所示。



#### (5) 提示信息

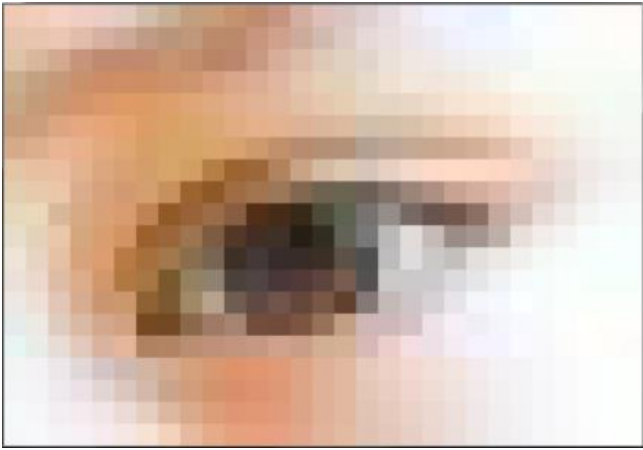
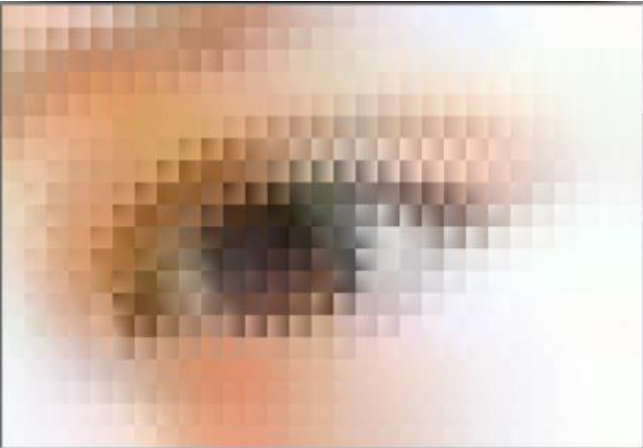
如下图所示，能帮助用户更好地操作。

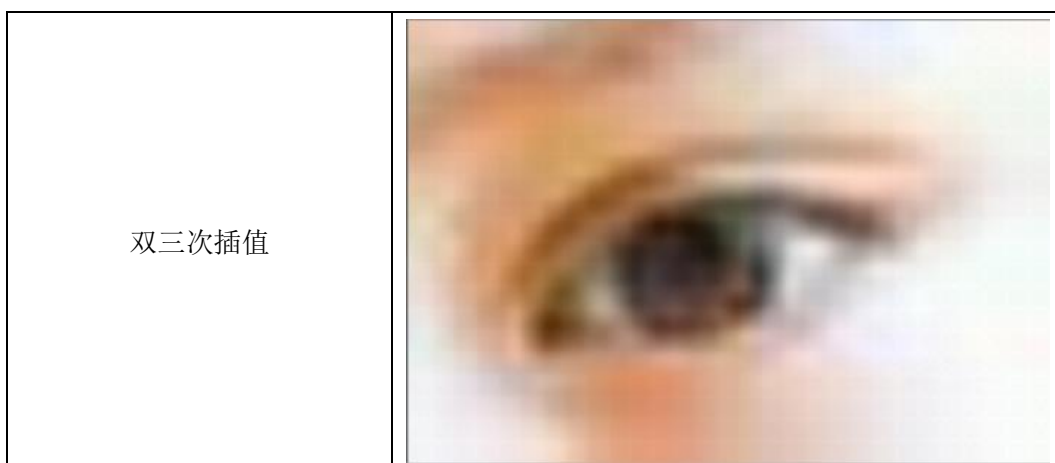




## 6.2. 插值效果对比

我们选取一张眼部的小图像为原图像，在纵向、横向各拉伸 15 倍，得到插值效果如下。

插值方法	插值效果
最近邻插值	
双线性插值	



分析：

(1) 最近邻插值

仅仅利用了几何上距离最近的像素点，未考虑像素点之间的联系，从而计算得到的通道值不连续性比较明显。得到的图像比较容易产生马赛克和锯齿，图像质量较低。但该方法计算用时较短，适合对图片质量要求不高而对速度要求较高的场合；

(2) 双线性插值

双线性插值方法在最近邻插值方法上有所改善，考虑了相邻的点之间的联系，图像质量有所改善。但是该方法未考虑到相邻点之间通道值变化率的相互影响，图像中的高频成分会被滤掉，质量仍会受损。该方法耗时比最近邻插值方法长，但图像质量有所改进，适合图像处理速度与质量的折中。

(3) 双三次插值

双三次差值方法克服了前两种方法的缺点，考虑到了相邻点的通道值与通道值变化率，图像处理的质量相对而言最高，但运算量大、运算时间长，适用于对图片处理质量要求高但对运算时间要求不高的情况。

## 7. 问题与解决方法

本次大作业理论与实现相并重，在完成的过程中遇到了不少问题，现将问题描述与解决方案介绍如下。



### 7.1. 边界溢出问题

在进行算法实现的过程中，一开始的最近邻插值我很快就完成了。但是在编写双线性插值的算法时，发现在 Debug 时总会有下标越界的情况使程序停止运行。

经过分析，发现原因在于没有考虑边界点：双线性插值要求用四个像素点确定  $f(r_0, c_0)$ ，如果  $(r_0, c_0)$  与边界的距离小于 1 像素，那么四个像素点的下标可能会超出范围。

为了解决这一问题，在代码中添加语句，当判断出下标越界时，将该点对应的值赋为最邻近边界点的值。

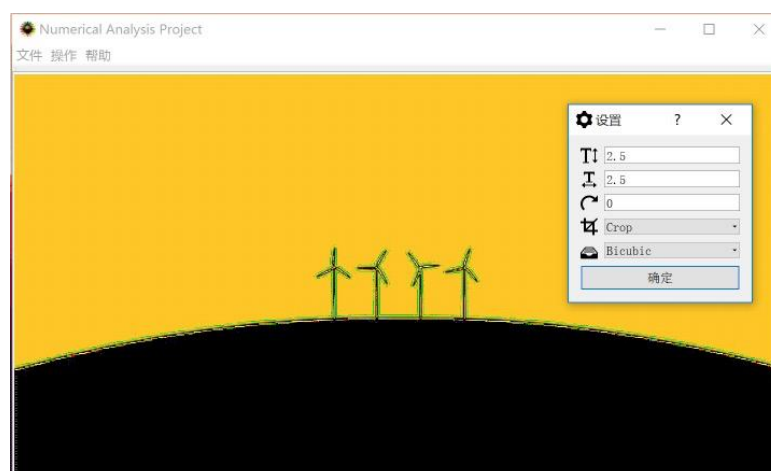
### 7.2. 旋转方向问题

一开始，图像的旋转问题困扰了我很久，操作时图像总是会与自己预期的相反方向旋转，其他方面则一切正常。思考了很久之后，才意识到，原因在于我在设定坐标系时， $r$  轴正方向向下的， $c$  轴正方向向右，与一般的  $x-y$  坐标系镜像对称。两个坐标系下，按照相同的旋转矩阵进行运算后得到的旋转方向刚好相反而我的旋转矩阵是按照传统的  $x-y$  坐标系选取的。

解决方法也比较简单，将旋转矩阵取逆即可。

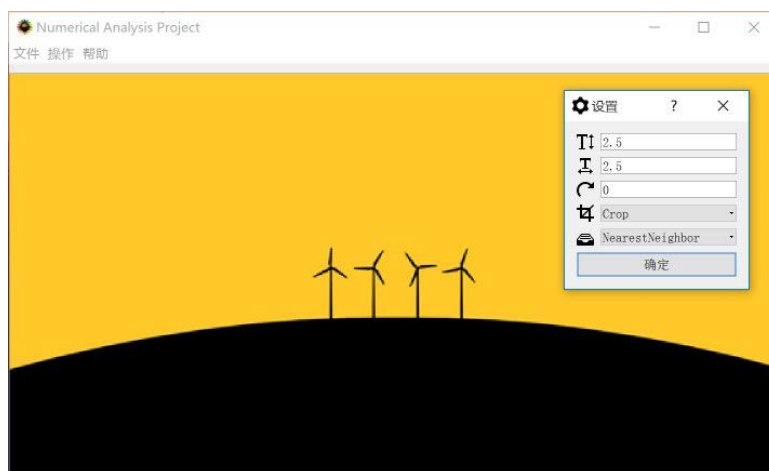
### 7.3. 双三次插值噪声问题

写好双三次插值的算法后，大多数情况下能得到比较理想的图像处理结果，但当图像对比度较高时，可能会产生比较明显的噪声，如下图所示。



经过分析，发现原因在于，在图像对比度较高的地方，双三次差值后得到的通道值不一定在  $[0, 255]$  区间内，如果溢出，则会导致颜色显示不正常。

解决方法也比较简单，加上条件判断语句，使通道值小于 0 时限制为 0，大于 255 时限制为 255，改正后效果如下图。



如上图所示，噪声得到消除。

## 8. 实验体会

本次大作业花了我整整 4 天左右的时间，其间的经历也是一波三折。

一开始我打算使用 OpenCV 进行图像的读取与存储操作，在配置了比较久的环境变量后终于配置成功，但由于是第一次操作 OpenCV，刚开始遇到了很多困难，比如有一次调试图片无法被正常显示的问题调试了很久都没成功，百度了之后才了解到原因在于漏加了一条语句。

在算法实现的过程中，尽管有现成的公式，但程序的架构、边界条件的考虑都需要我们自己思考实现，我在旋转问题上纠结了差不多半天，最后终于解决，解决方法参见 7.2.。

而编写图形界面则远比我想象的耗时间。首先是工具选取上，一开始我打算使用 Qt，尝试了一下感觉比较难，于是想换成 MFC，试了一下又觉得 MFC 太麻烦，最终还是使用了 Qt。在编写图形界面的过程中，我得以复习了一年多以前 GUI 编程的技能，但整个过程充满了坎坷。比如说我一开始怎么尝试都没办法利用.qrc 文件，调试了一个晚上后被逼无奈重新建立工程后问题才得到解决；再比如在界面的设计上我也是纠结许久，期望能够设计出比较符合

使用习惯的界面。

总的来说，这次大作业让我第一次接触了图像处理的编程实现，并进一步锻炼了我的算法实现能力和软件开发能力，虽然遇到了许多坎坷，但最终完成后还是有很大的成就感。