# 摘要

图像旋转是图像处理中最常见的操作之一，本文主要介绍图片旋转的原理，包括前向映射算法、常见的两种后向映射算法，以及用C++编程实现图像旋转。

实验的平台、框架以及语言：VS Code、OpenCV（仅用于读写图片）、C++

本文关键词：**图片旋转算法、前向映射算法、后向映射算法（最邻近法、双线性插值法）**

# 计算旋转后图像的宽与高

## 公式推导

计算旋转后图像的宽与高，通过画草图的方式来模拟旋转的过程，可以很容易算出旋转后的图像的宽与高。旋转模拟见图2-1，公式推导过程如下文所示。

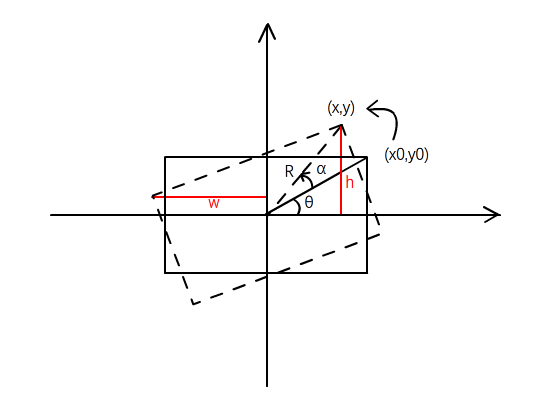


图2-1

假设原图的宽为W，高为H，旋转后图片的宽为W’，高为H’，并根据上图可知：

同理可得：

## 关键代码



图2-2

如图2-2红框所示，计算出旋转后图片的宽与高。需要注意的是，用户可以输入任意的旋转角度，因此要保证三角函数的计算结果是正值。

# 像素点的旋转与坐标系变换

## 像素点的旋转公式推导

图示

描述已自动生成

图3-1

与计算图片旋转后的宽与高类似，计算坐标点的旋转也可以通过画草图模拟，公式推导过程如下。

假设点旋转θ度后到达点处，并假设该点到原点的距离为R，易得有以下公式：

化简，可得：

使用矩阵表示：

## 数学坐标系与图像坐标系的转换

我们可知，图像的旋转是以图像的中心点为旋转点，对应着数学坐标系的原点，而图像坐标系是以图像左上角为原点，因此我们需要在图像旋转时实现数字坐标与图像坐标的相互转换。在旋转图像时，先把坐标系从图像坐标系转换到数学坐标系，旋转后再转换回图像坐标系。

假设原图像的宽高为W，H，旋转后的图像的外接矩形宽高大小为W’，H’，则图像坐标系的点与数学坐标系的点的相互转换，有以下转换公式：

结合像素点的旋转矩阵，我们得到最终的像素点旋转变换公式：

实际上，式①即为前向映射变换，式②为后向映射变换。在编程时，我们可以利用矩阵乘法函数把多个矩阵相乘，得到最终变换后的坐标，也可以对上述公式进一步化简，以便计算结果。化简后，可得式③与式④：

，③

，④

## 关键代码

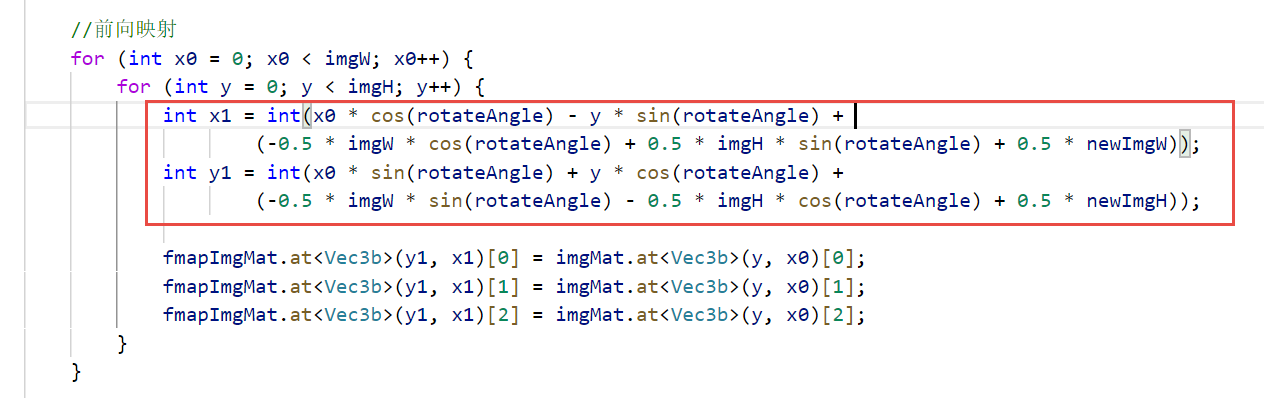


图3-2

在本实验中，没有采用矩阵的直接运算，而是通过将变化矩阵化简，再计算得到坐标变换后的值。

# 前向映射法与后向映射法

通俗的讲，前向映射法就是遍历原图的点，将其映射到旋转后的图像上。后向映射法，就是遍历旋转后的图像上的点，然后在原图上找到所需要的像素点，将其填充到旋转后的图像中。

前向映射法一般不会被采用，因为经过计算后得到的坐标值是浮点型，而现实中图像的坐标值只能是整型，可能会导致旋转后图像上的某些点是缺失的。

后向映射法则无上述问题，因为后向映射的原理是在原图上找对应的像素点，可以保证旋转后的图像每个像素点都可以从原图中得到或者生成。**后向映射主要要解决的问题，是该如何在原图中寻找对应的像素点，其本质是一个插值问题。**

在实践中，后向映射算法一般有最近邻法(Nearest Interpolation)、双线性插值(Bilinear Interpolation)、双三次插值(Bicubic interpolation)等算法。

## 最邻近法及关键代码

最邻近算法的原理是通过坐标映射直接找到原图像的某个对应的点，将其像素值赋值给新图像对应的位置的像素点中。由于最邻近法直接寻找原图的像素点，不涉及其他额外的计算，因此计算效率最高，但由于这种“暴力”的处理，往往会破坏图像中原有的渐变关系，在图像的某些细节可能显示效果比较差。

图4-1为最邻近法的关键代码，注意红框代码中末端的”+0.5”再整体取整，以便取到离当前的坐标点最近的像素点。

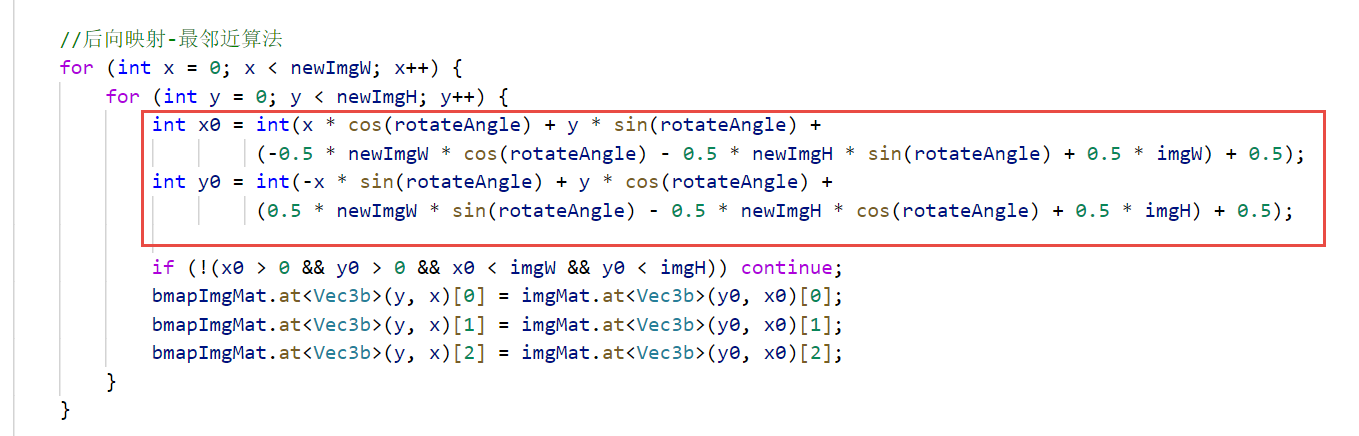


图4-1

## 前向映射法与最邻近法的实验效果对比

图4-2 图4-3

采用前向映射算法得到的图像如图4-2所示，明显肉眼可见图像中存在着有规律的像素缺失。图4-3则采用后向映射算法中的最邻近算法，则无像素缺失的情况出现。

## 双线性插值法及关键代码

在上一节中，我们采用最邻近法实现了无像素点缺失的图像旋转，但由于最邻近法往往会破坏图像原有的渐变效果，在某些细节上显得过渡特别生硬。因此，我们可以采用另一种算法，即**双线性插值法(Bilinear Interpolation)**。

双线性插值的坐标变换公式和最近邻法一样，区别在于后者是找到离当前坐标点最近的1个像素点，而前者是通过找到最近的4个像素点，以此计算生成1个像素点。因此，双线性插值法往往有着比较出色的显示效果。

同样地，我们可以通过画草图的方式来理解这个算法的原理。

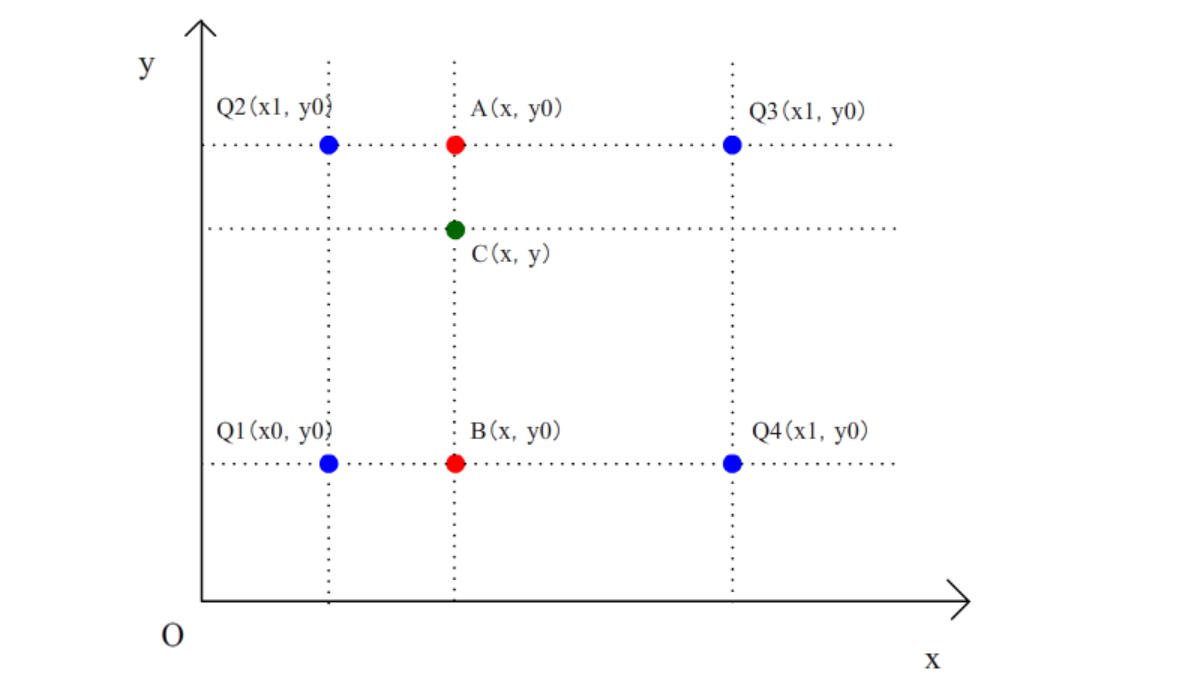


图4-4

如上图所示，假设点C(x,y)是通过坐标变换后得到的像素点坐标，那它的像素值会根据与之相邻的四个像素点计算得出。具体的计算步骤是先分别沿X轴方向，根据点Q1与点Q4、点Q2与点Q3的像素值计算了两次单线性插值得到点A与点B的像素值，再根据点A与点B的像素值与坐标，再沿Y轴方向上计算一次单线性插值，进而得到点C的像素值。

若使用函数f(X)表示该点X的像素值，则有以下公式：

因为4个像素点是相邻的，所以有关系式：，并将式①，式②代入式③中并化简，可得点C的像素值计算如下：

关键代码如下图4-5所示：



图4-5

## 最邻近法与双线性插值法的实验效果对比

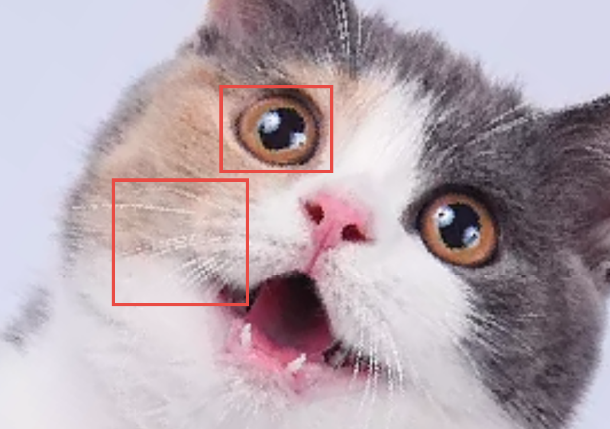
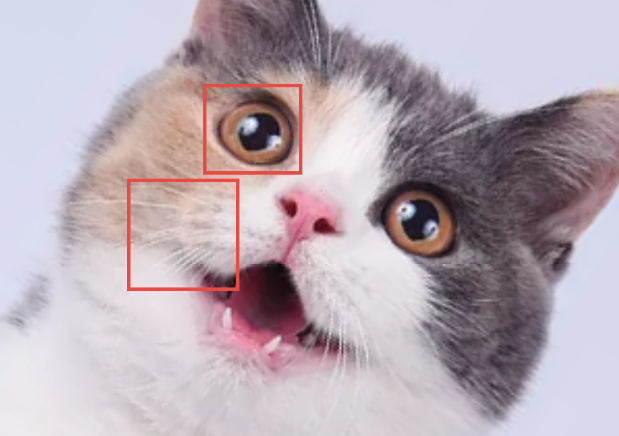
 

图4-6 图4-7

图4-6是采用最邻近法生成的图像，可见猫的眼睛以及胡子处渐变效果较差，过渡生硬。反之，采用双线性插值生成的图像4-7在同样的位置则有比较好的过渡与渐变效果。

# 总结

* 前向映射法会导致旋转后的图像出现某部分像素的缺失，而后向映射法则无此问题。
* 后向映射法包括最邻近法、双线性插值法、双三次插值插值等算法。
* 最近邻法(Nearest Interpolation)：计算速度最快，但显示效果最差。
* 双线性插值(Bilinear Interpolation)：双线性插值是用原图像中4个点计算出新图像中1个点，在性能与效果之间取得比较好的平衡，也是很多图像处理框架中属于默认算法。