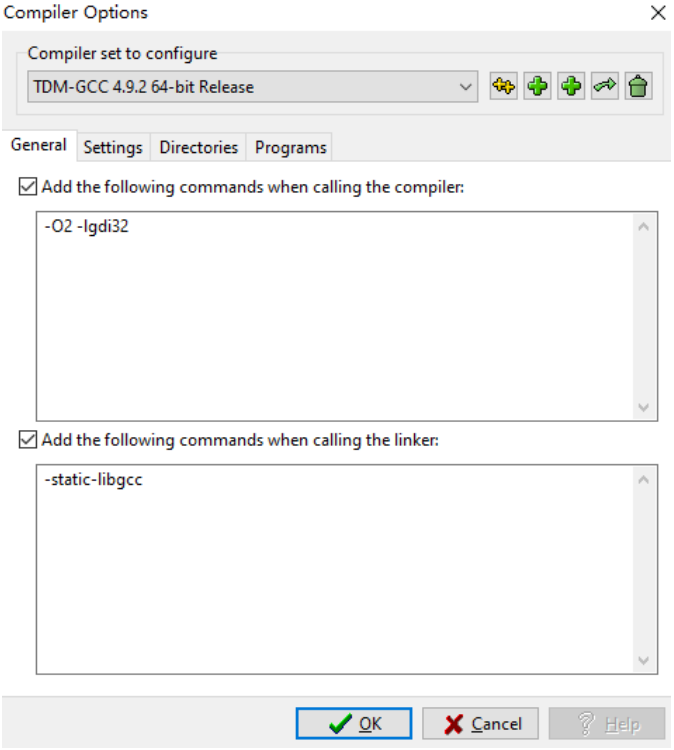
Ex1:图像读取和显示以及像素操作

学号：17343124
姓名：伍斌
完成时间：2020.9.21

【实验准备】下载 CImg 库，解压放入相应位置，并尝试调用 CImg 库调试。
遇到问题：error: ld returned 1 exit status
解决方法：参考博客：https://blog.csdn.net/sinat_31790817/article/details/79521714
命令行下：在编译选项后面加上-O2 -lgdi32 即可解决：

```
g++ -o hello_word.exe hello_word.cpp -O2 -lgdi32
```

Dev-C++下：在导航栏找到工具->编译选项->编译时加入以下命令-O2 -lgdi32，如下图：



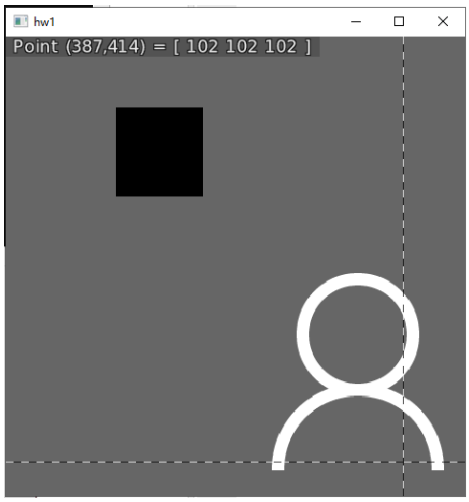
【实验过程】

一、读入 1.bmp 文件，并用 CImg.display()显示。

代码如下：

```
#include "CImg.h"
using namespace cimg_library;
int main() {
	CImg <unsigned char> SrcImg;
	SrcImg.load_bmp("1.bmp");
	SrcImg.display("hw1");
	return 0;
}
```

显示结果如下：

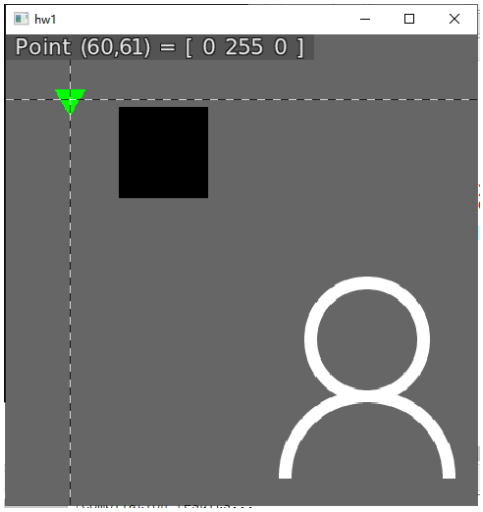


二、在图上绘制一个等边三角形区域，其中心坐标(60, 60)，边长为 30，填充颜色为绿色。

代码如下:

```
void DrawTriangle_green() {
	cimg_forXY(SrcImg, x, y) {
		if (y > 60-15*sqrt(3)/3 && y < 60+30*sqrt(3)/3){
			if (x > (y + 50 * sqrt(3) - 60) / sqrt(3) && x < (y - 70 * sqrt(3) - 60) / (-1 * sqrt(3))) {
				SrcImg(x, y, 0) = 0;
				SrcImg(x, y, 1) = 255;
				SrcImg(x, y, 2) = 0;
			}
		}
	}
}
```

显示结果如下:



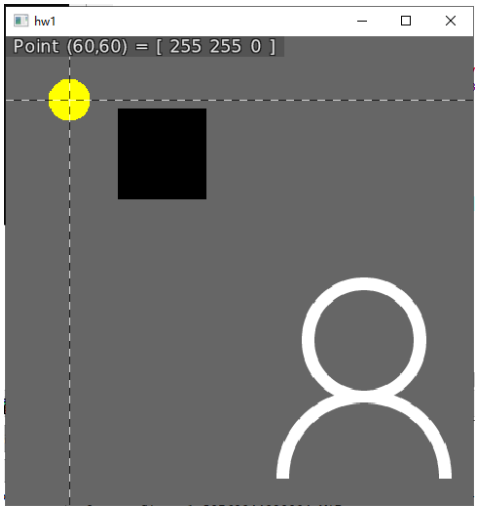
三、在图上绘制一个圆形区域，圆心坐标（60,60），半径为 20，填充颜色为黄色。

1. 不用 CImg 的接口函数时:

代码如下:

```
void DrawCircle_yellow1(CImg <unsigned char> TempImg) {
    CImg<unsigned char> SrcImg = TempImg;
    cimg_forXY(SrcImg, x, y) {
        if (pow(pow(x-60,2)+pow(y-60,2),0.5) < 20) {
            SrcImg(x,y,0) = 255;
            SrcImg(x,y,1) = 255;
            SrcImg(x,y,2) = 0;
        }
    }
}
```

显示结果如下：

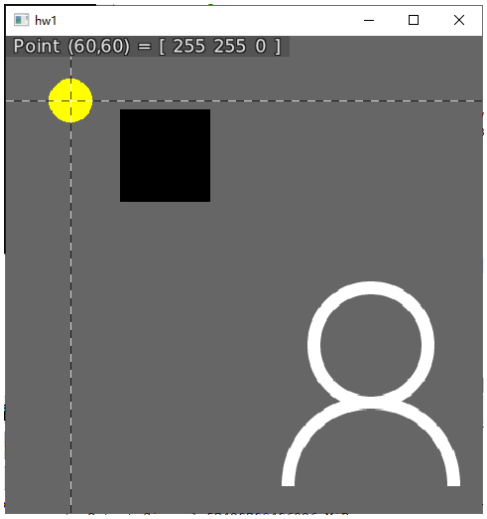


2. 使用 CImg 的接口函数时：

代码如下：

```
void DrawCircle_yellow2() {
    unsigned char yellow[] = { 255, 255, 0 };
    SrcImg.draw_circle(60, 60, 20, yellow);
}
```

显示结果如下：



对比：感觉实现效果差不多，因为我们的实验要求画的圆大小是半径为 20。查阅师兄的博客发现，当半径较小时，因为遍历图片的像素点是对图片进行采样，所以点坐标为整数值，圆的半径越小意味着所取的像素点范围越少，点越少，所以图片的效果会较差，甚至接近正方形。

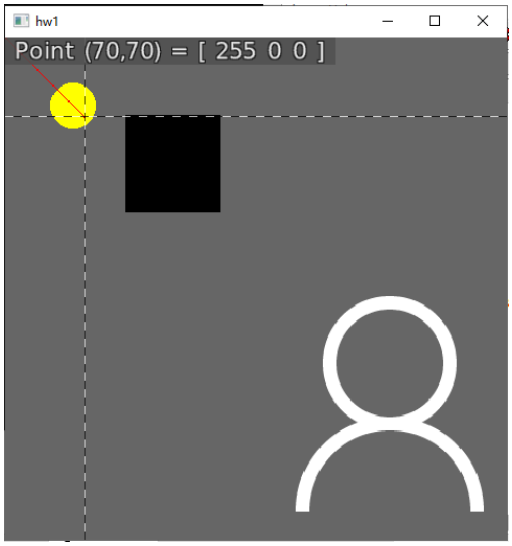
四、在图上绘制一条长为 100 的直线段，起点坐标为(0, 0)，方向角为 45 度，直线的颜色为红色。

1. 不用 CImg 的接口函数时：

代码如下：

```
void DrawLine1(CImg <unsigned char> TempImg) {
	CImg<unsigned char> SrcImg = TempImg;
	double x0 = 100*cos(45*pi/180);
	double y0 = 100*sin(45*pi/180);
	cimg_forXY(SrcImg, x, y) {
		if (x == 0) {
			if (y == 0) {
				SrcImg(x,y,0) = 255;
				SrcImg(x,y,1) = 0;
				SrcImg(x,y,2) = 0;
			}
		}
		else {
			if (cmp((double)y, (double)x*tan(45*pi/180)) && (double)x <= x0 && (double)y
<= y0) {
				SrcImg(x,y,0) = 255;
				SrcImg(x,y,1) = 0;
				SrcImg(x,y,2) = 0;
			}
		}
	}
}
bool cmp(double x , double y) { //compare x and y，如果差值小于一定范围则近似相等
	if (abs(x - y) <= 0.5)
		return 1;
	return 0;
}
```

显示结果如下：

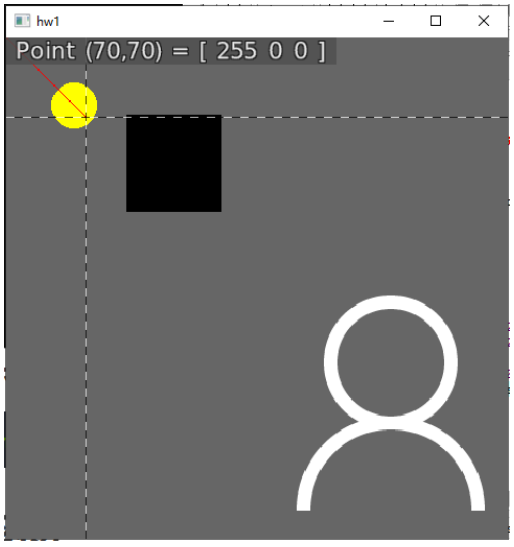


2. 使用 CImg 的接口函数时:

代码如下:

```
void DrawLine2() {
	unsigned char red[] = {255,0,0};
	TempImg.draw_line(0,0,100*cos(45*pi/180),100*sin(45*pi/180),red);
}
```

显示结果如下:



对比:（有参考师兄博客）这个比较效果感觉肉眼很难看出差距，但是因为遍历图片的像素点是对图片进行采样所以点坐标为整数值，所以不可能存在彻底等于所得的浮点数 tan 角要求的坐标，只能通过设置误差尽可能小，也就是直线的粗细，通过设置 cmp 函数调节几个值（0.1, 0.3, 0.5, 1.0）发现 0.5 的效果是最好的。

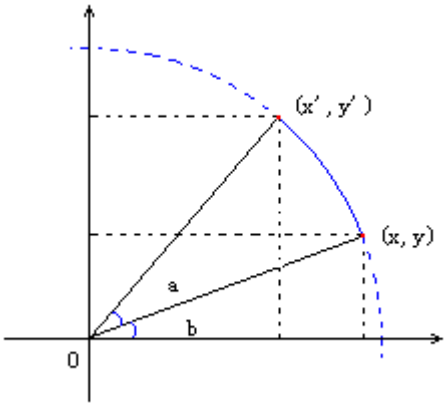
五、用 C++的写图像旋转函数（不调用其他的图像处理函数），把第四步的结果图像顺时针旋转 15 度和 45 度。

1. 不用 CImg 的接口函数时:

花了三天时间也没能写出实际实现代码，但是查阅博客有了思路，旋转图像的算法设计如下:

（参考博客：https://blog.csdn.net/wonengguwozai/article/details/52049092 ）

首先推导旋转公式:



旋转示意图

有：　tg(b)=y/x　　----(1)

　　tg(a+b)=y'/x'　　----(2)

　　x*x + y*y = x'*x' + y'*y'　　----(3)

有公式：tg(a+b) = (tg(a)+tg(b)) / (1-tg(a)*tg(b))　----(4)

把(1)代入(4)从而消除参数 b;

　tg(a)+y/x = y'/x'*(1-tg(a)*y/x)　　----(5)

　由(5)可以得 x'=y'*(x-y*tg(a))/(x*tg(a)+y)　　----(6)

把(6)代入(3)从而消除参数 x',化简后求得:

　　y'=x*sin(a)+y*cos(a);　　----(7)

把(7)代入(6),有:

　　x'=x*cos(a)-y*sin(a);　　----(8)

在图片旋转中的应用:

　　假设对图片上任意点(x,y), 绕一个坐标点(rx0,ry0)逆时针旋转 RotaryAngle 角度后的新的坐标设为(x', y'), 有公式:

　(x 平移 rx0,y 平移 ry0,角度 a 对应-RotaryAngle ， 带入方程(7)、(8)后有：)

　x'= (x - rx0)*cos(RotaryAngle) + (y - ry0)*sin(RotaryAngle) + rx0 ;

　y'=-(x - rx0)*sin(RotaryAngle) + (y - ry0)*cos(RotaryAngle) + ry0 ;

那么, 根据新的坐标点求源坐标点的公式为:

　x=(x'- rx0)*cos(RotaryAngle) - (y'- ry0)*sin(RotaryAngle) + rx0 ;

　y=(x'- rx0)*sin(RotaryAngle) + (y'- ry0)*cos(RotaryAngle) + ry0 ;

旋转的时候还可以顺便加入 x 轴和 y 轴的缩放和平移, 而不影响速度, 那么完整的公式为:

　x=(x'-move_x-rx0)/ZoomX*cos(RotaryAngle) - (y'- move_y-ry0)/ZoomY*sin(RotaryAngle) + rx0 ;

　y=(x'-move_x-rx0)/ZoomX*sin(RotaryAngle)+ (y'- move_y-ry0)/ZoomY*cos(RotaryAngle) + ry0 ;

　其中:

　　RotaryAngle 为逆时针旋转的角度;

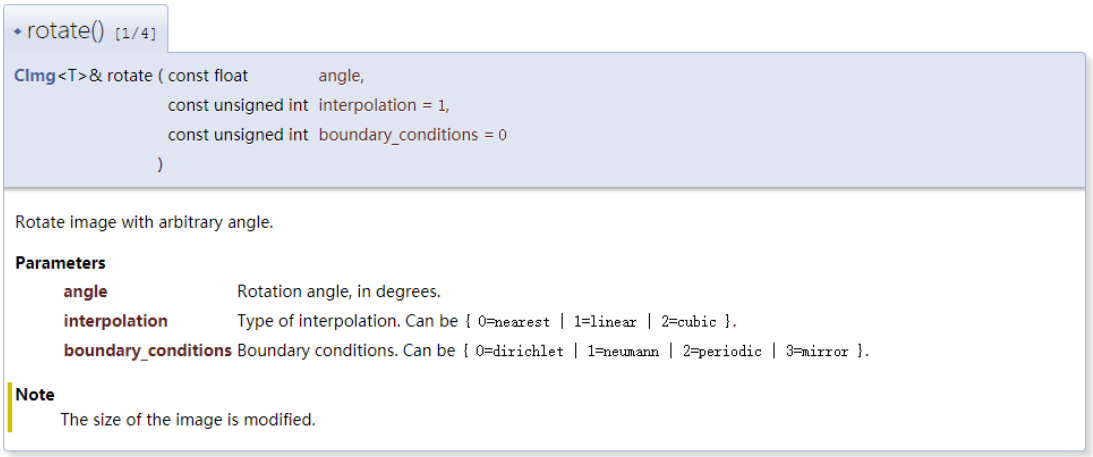
　　ZoomX,ZoomY 为 x 轴 y 轴的缩放系数(支持负的系数,相当于图像翻转);

　　move_x,move_y 为 x 轴 y 轴的平移量;

实际遇到的困难:

不清楚如何重写整个图像，包括怎么算需要新开的空间大小和描述新的映射关系。在 csdn 和博客园上都没有更详细的有关 CImg 的库的源码分析，阅读库的源代码尚未有收获。

2. 使用 CImg 的接口函数时：
查阅 CImg 手册，得到如下接口函数：



• rotate() [1/4]

```
CImg<T>& rotate ( const float          angle,
                  const unsigned int   interpolation = 1,
                  const unsigned int   boundary_conditions = 0
                )
```

Rotate image with arbitrary angle.

Parameters

angle	Rotation angle, in degrees.
interpolation	Type of interpolation. Can be { 0=nearest | 1=linear | 2=cubic }.
boundary_conditions	Boundary conditions. Can be { 0=dirichlet | 1=neumann | 2=periodic | 3=mirror }.

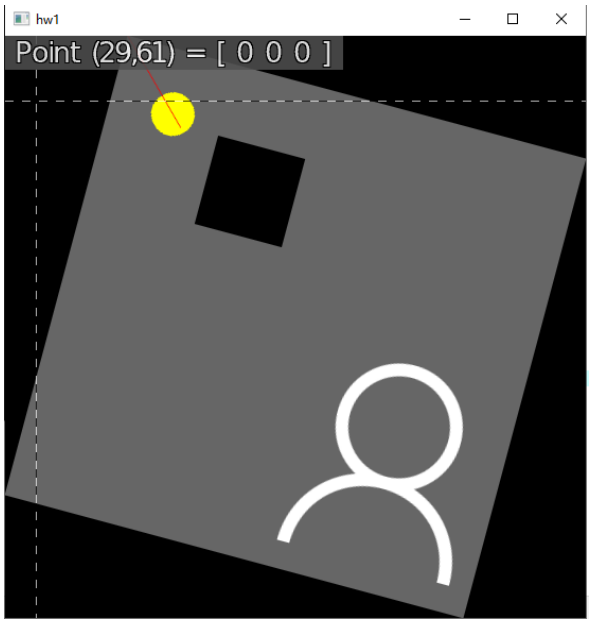
Note

The size of the image is modified.

旋转 15 度：
代码如下：

```
void Rotate1() {
	SrcImg.rotate(15);
}
```

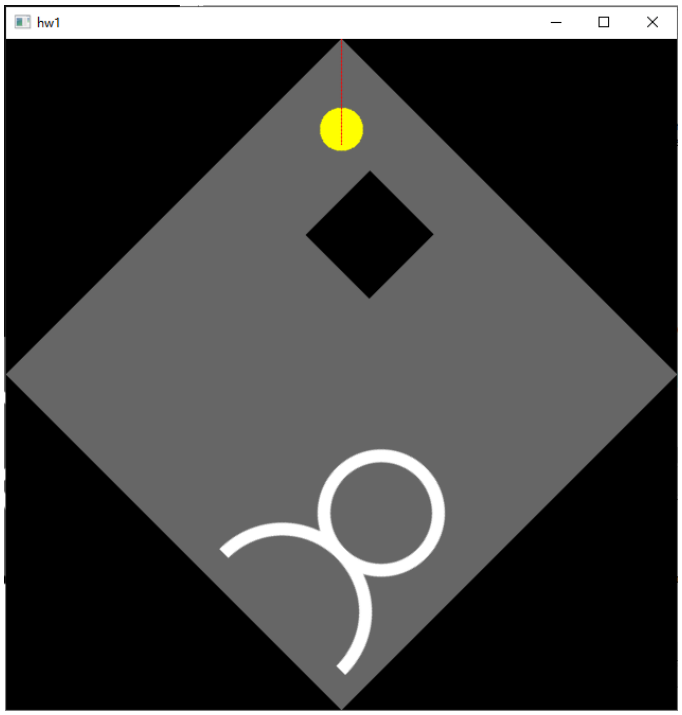
显示结果如下：



旋转 45 度：
代码如下：

```
void Rotate2() {
	SrcImg.rotate(45);
}
```

显示结果如下：



六、把上面的操作结果保存为 2.bmp。

代码如下：

```
SrcImg.save("2.bmp");
```

特殊要求：把上述功能代码携程类的形式，把三四五步封装为该类的操作：

main.cpp

```
#include "ex.hpp"
using namespace std;
int main(int argc, char const *argv[])
{
	Test pic;
	pic.DrawTriangle_green();
	pic.DrawCircle_yellow2();
	pic.DrawLine2();
	pic.Rotate1();
	pic.Rotate2();
	pic.Todisplay();
	CImg<unsigned char> temp = pic.getSrcImg();
	temp.save("2.bmp");
	return 0;
}
```

ex.hpp

```
#ifndef _EX_HPP_
```

```
#define _EX_HPP_
#include "CImg.h"
#include <cmath>
#include <string>
using namespace std;
using namespace cimg_library;
const double pi(3.14159265);
class Test
{
	public:
		Test();
		~Test();
		void Todisplay();
	//不调用 CImg 接口函数
		void DrawTriangle_green();
		void DrawCircle_yellow1();
		void DrawLine1();

	//调用 CImg 接口函数
		void DrawCircle_yellow2();
		void DrawLine2();
		void Rotate1();
		void Rotate2();

		CImg<unsigned char> getSrcImg();
	private:
		CImg<unsigned char> SrcImg;
};
#endif
```

ex.cpp

```
#include "ex.hpp"
using namespace std;
bool cmp(double x , double y);

Test::Test() {
	SrcImg.load_bmp("1.bmp");
}

Test::~Test() {}

void Test::Todisplay() {
	SrcImg.display("hw1");
}
```

```
CImg<unsigned char> Test::getSrcImg() {
	return SrcImg;
}
void Test::DrawTriangle_green() {
	cimg_forXY(SrcImg, x, y) {
		if (y > 60-15*sqrt(3)/3 && y < 60+30*sqrt(3)/3){
			if (x > (y + 50 * sqrt(3) - 60) / sqrt(3) && x < (y - 70 * sqrt(3) - 60) / (-1 *
sqrt(3))) {
				SrcImg(x, y, 0) = 0;
				SrcImg(x, y, 1) = 255;
				SrcImg(x, y, 2) = 0;
			}
		}
	}
}

void Test::DrawCircle_yellow1() {
	cimg_forXY(SrcImg, x, y) {
		if (pow(pow(x-60,2)+pow(y-60,2),0.5) < 20) {
			SrcImg(x,y,0) = 255;
			SrcImg(x,y,1) = 255;
			SrcImg(x,y,2) = 0;
		}
	}
}

void Test::DrawLine1() {
	double x0 = 100*cos(45*pi/180);
	double y0 = 100*sin(45*pi/180);
	cimg_forXY(SrcImg, x, y) {
		if (x == 0) {
			if (y == 0) {
				SrcImg(x,y,0) = 255;
				SrcImg(x,y,1) = 0;
				SrcImg(x,y,2) = 0;
			}
		}
		else {
			if (cmp((double)y, (double)x*tan(45*pi/180)) && (double)x <= x0 && (double)y
<= y0) {
				SrcImg(x,y,0) = 255;
				SrcImg(x,y,1) = 0;
				SrcImg(x,y,2) = 0;
```

```
                }
            }
        }
}

void Test::DrawCircle_yellow2() {
        unsigned char yellow[] = { 255, 255, 0 };
        SrcImg.draw_circle(60, 60, 20, yellow);
}

void Test::DrawLine2() {
        unsigned char red[] = {255,0,0};
        SrcImg.draw_line(0,0,100*cos(45*pi/180),100*sin(45*pi/180),red);
}

void Test::Rotate1(){
        SrcImg.rotate(15);
}

void Test::Rotate2(){
        SrcImg.rotate(30);
}

bool cmp(double x , double y) { //compare x and y，如果差值小于一定范围则近似相等
        if (abs(x - y) <= 0.5)
                return 1;
        return 0;
}
```

【分析】

1. 为什么第三步绘制的圆形区域形状效果不好。

图像处理方式的问题。因为是用采样处理的方式，在遍历像素点时去掉了整数值。后面的图像范围较小，像素点采集的个数会不足，通过人眼所观察到的则更加不同。误差增大。

2. 第五步如果效果不好，问题出在哪？为什么？

未能实现不用 Cimg 接口函数的代码，而使用 Cimg 接口函数的代码没有出现问题。在群里看到其他同学的效果不好的实验结果，主要问题是图像在旋转后分辨率变大，如果只是在原图上映射，会切掉四个三角形区域，两次旋转之后图像变成了一个十二边形。

【自评分数】

自我评价：90

理由：未能完成实现不用 CImg 接口的旋转操作。