计算机视觉 - Homework1

姓名：陈明亮

学号：16340023

一、实验需求

1. 学习如何使用C++库文件CImg.h进行图像的简单处理，包括读取图像，以及基本的像素操作。

2. 结合CImg类的各项操作，对绘制圆和曲线的函数，要求自己通过逻辑编写C++代码，比较与CImg类原生操作的差异。

3. 思考为什么作业要求中的第四步绘制的圆形区域效果不好

二、实验过程

1. 读入1.bmp文件，并用CImg.display()显示

方法：直接使用CImg类实例化一个对象，读入1.bmp图像，并且使用成员函数display()显示

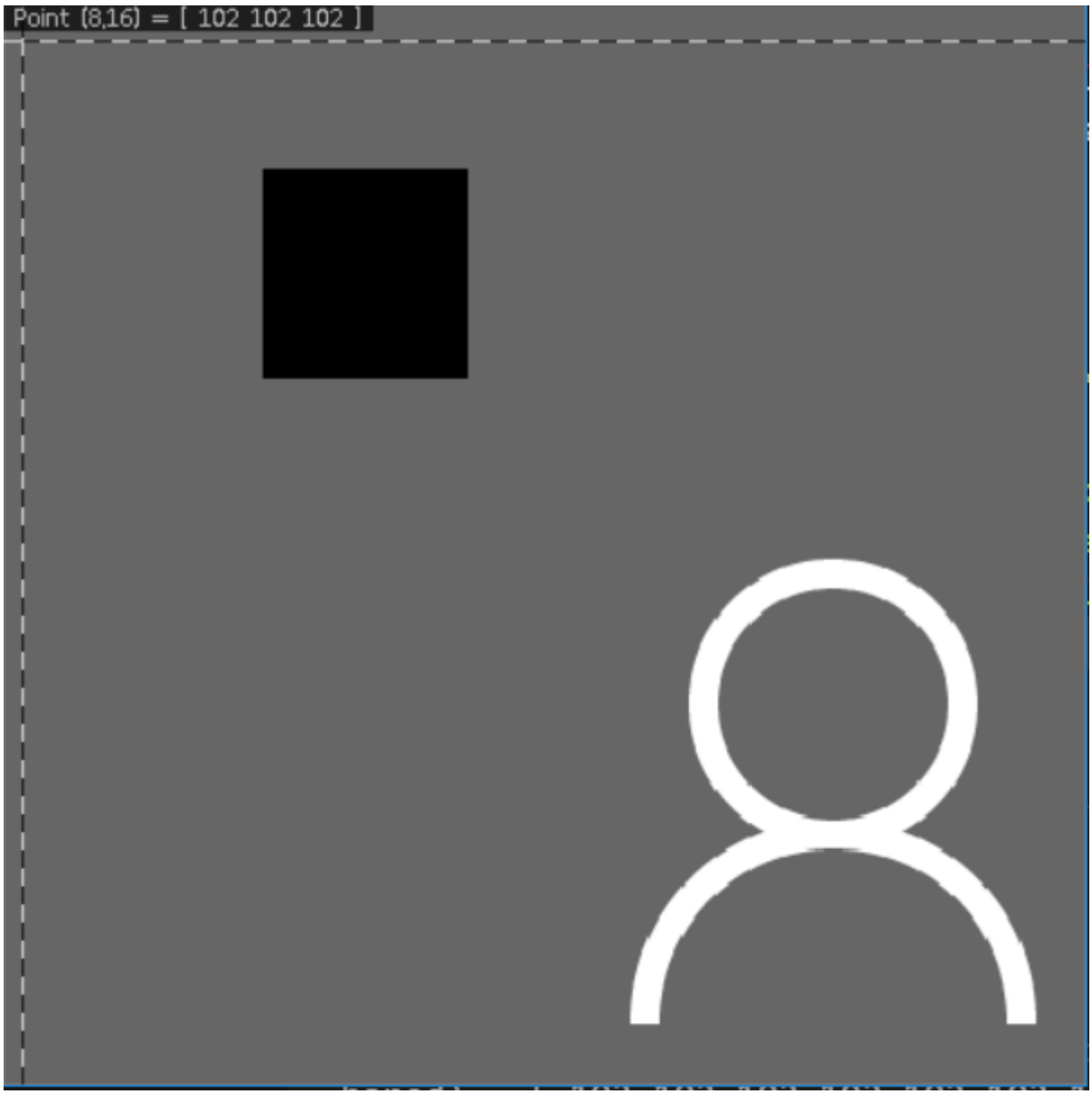
代码：

```
#include "CImg.h"

using namespace cimg_library;

int main(){
    // Step 1
    CImg<unsigned char> img("1.bmp");
    img.display();
    return 0;
}
```

运行结果：



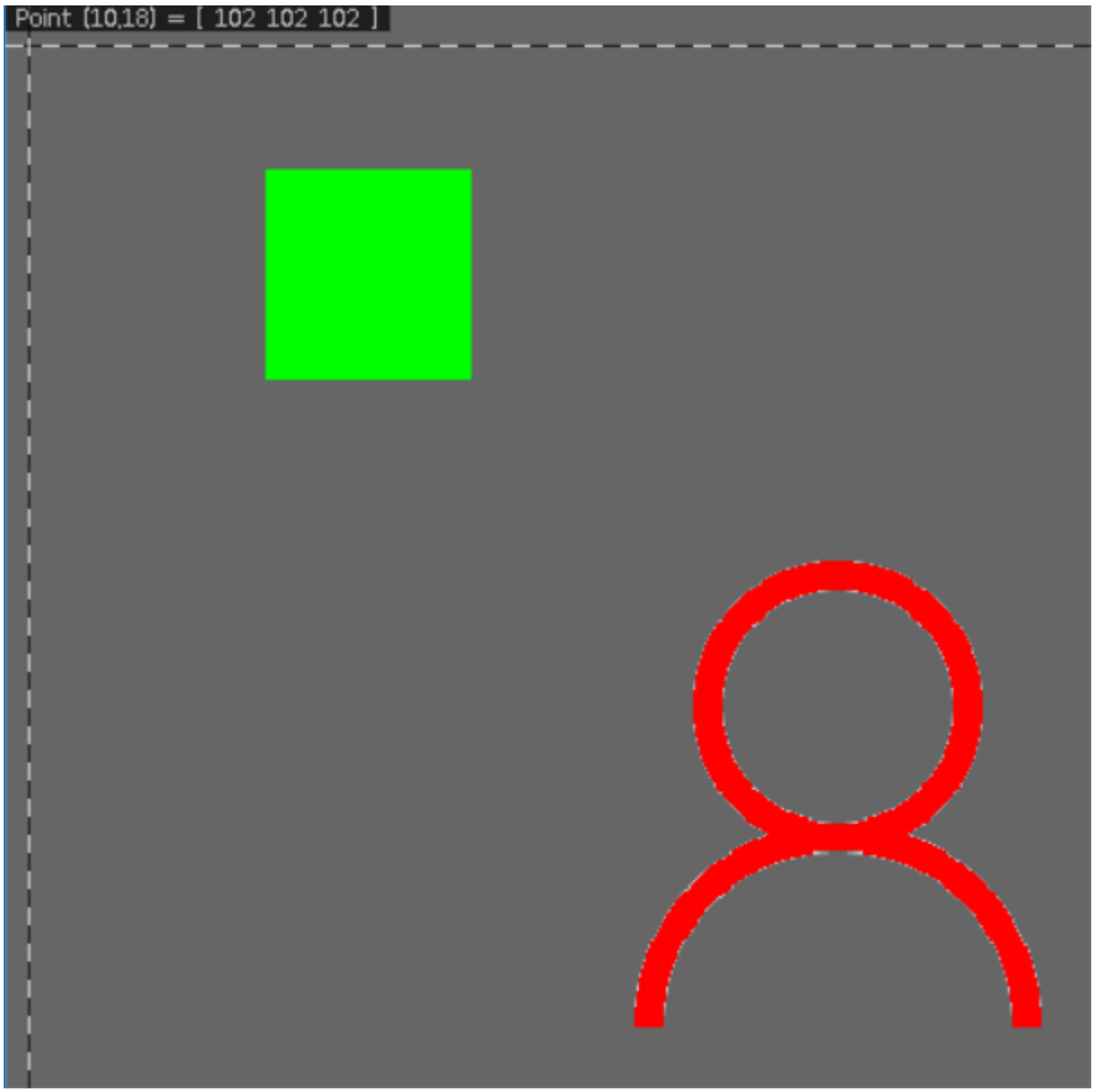
2. 把读入的BMP文件的白色区域变为红色，黑色区域变为绿色

方法：采用CImg内的宏定义`cimg_forXY`遍历图像的每个像素点，并判断像素点是否为黑色(*RGB={0,0,0}*)，或者白色(*RGB={255,255,255}*)，符合条件则把该像素点的RGB值对应改成红色(RGB={255,0,0})，或绿色(RGB={0,255,0})

代码：

```
// Step 2
    cimg_forXY(img, x, y){
        // White color's RGB is (255, 255, 255), Red's RGB is(255, 0, 0)
        if(img(x, y, 0) == 255 && img(x, y, 1) == 255 && img(x, y, 2) == 255){
            img(x, y, 1) = 0;
            img(x, y, 2) = 0;
        }
        // Black color's RGB is (0, 0, 0), Green's RGB is(0, 255, 0)
        if(!img(x, y, 0) && !img(x, y, 1) && !img(x, y, 2)){
            img(x, y, 1) = 255;
        }
    }
```

运行结果：



3. 先不使用CImg提供的类方法接口，编写C++代码实现绘制圆心为(50,50)，半径为30的蓝色圆形区域，然后再调用CImg的方法，比较两者的差异

方法：自己使用`cimg_forXY`遍历像素点，然后根据两点之间的距离公式，判断某点是否距离圆心小于等于半径，符合条件则改变像素点为蓝色

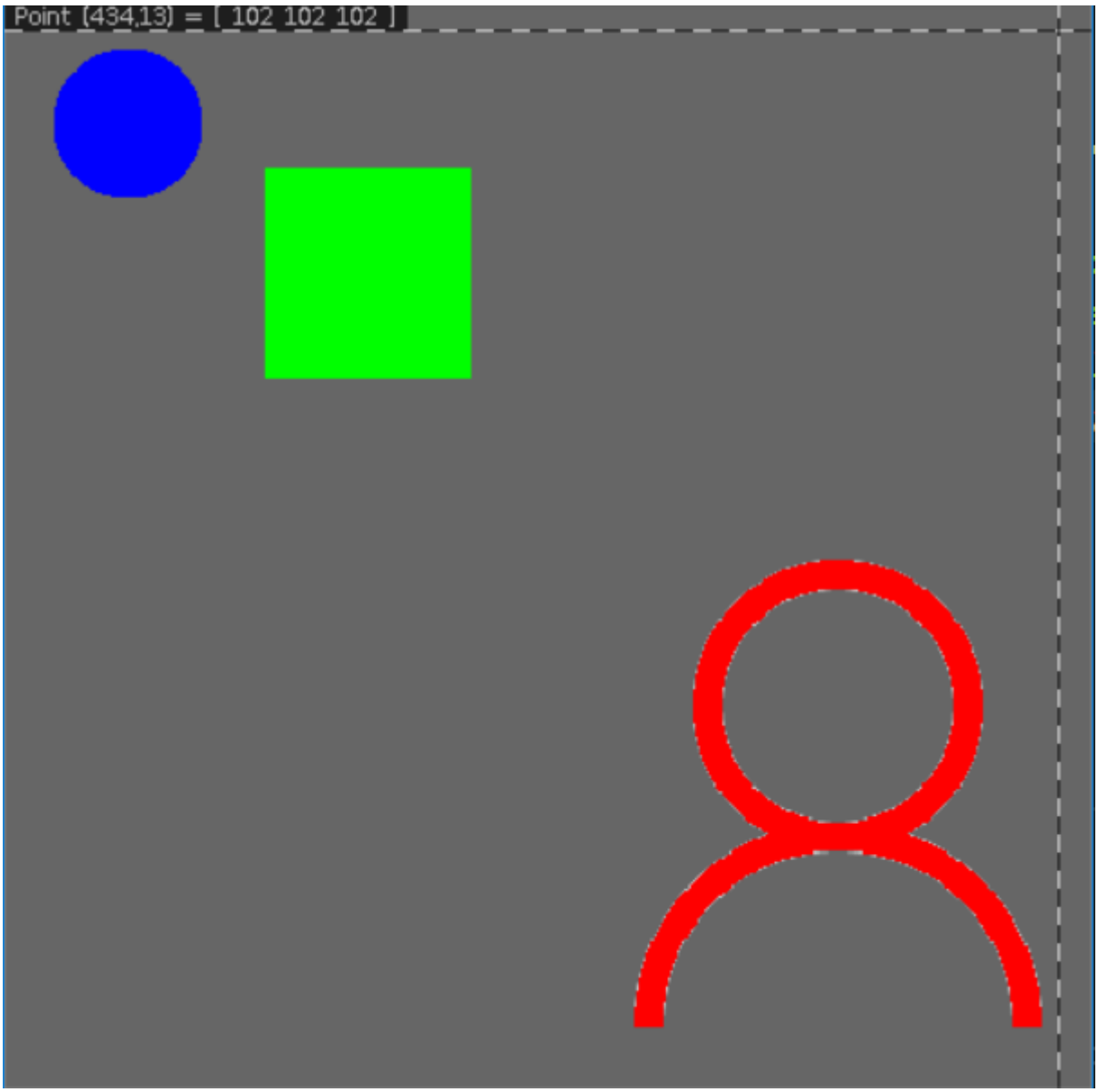
不使用接口的代码：

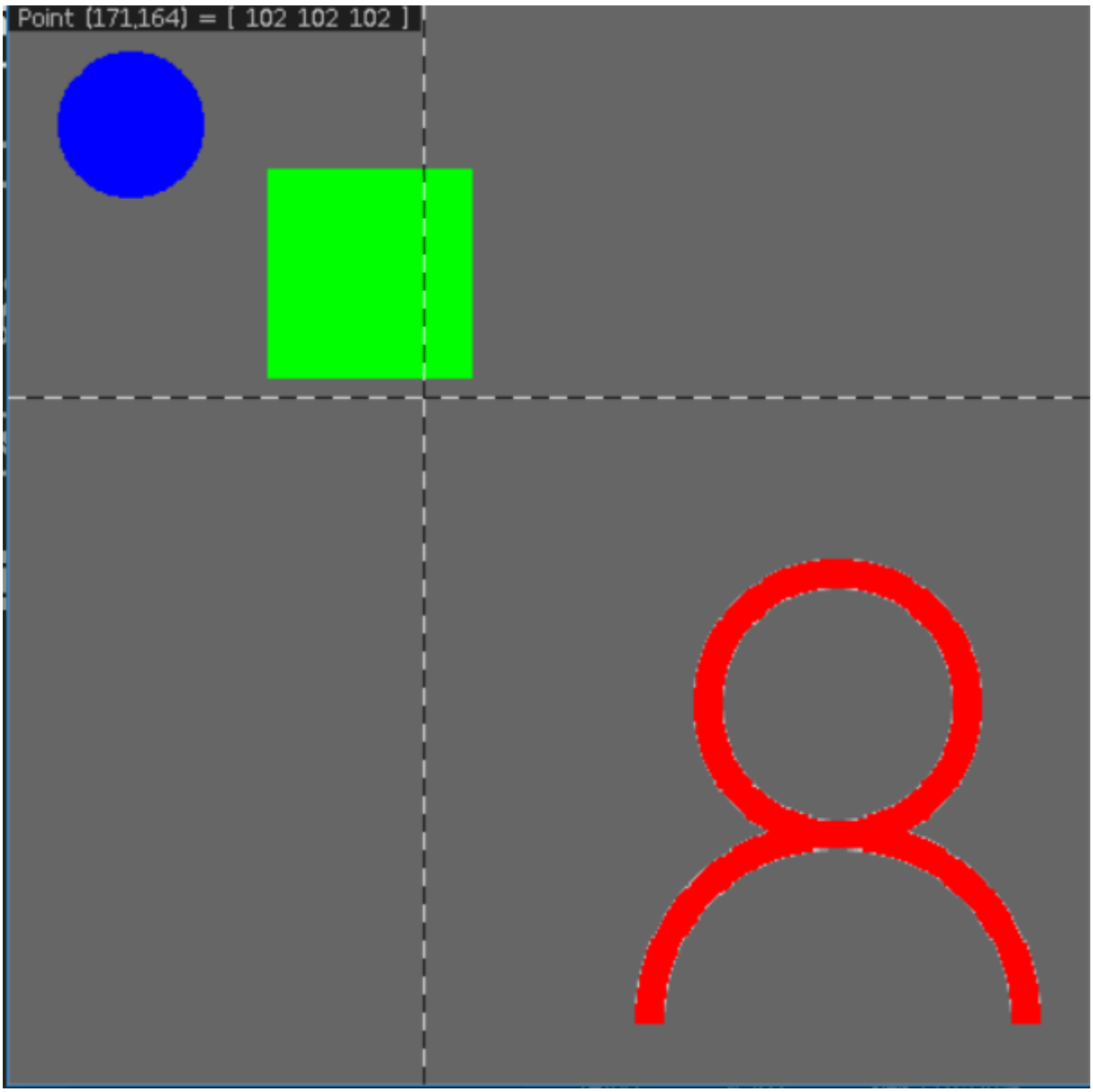
```
void drawBlueCircle(pair<int, int> center, int radius){
        cimg_forXY(img, x, y){
            if(getDistance(center, make_pair(x, y)) <= radius){
                img(x, y, 0) = img(x, y, 1) = 0;
                img(x, y, 2) = 255;
            }
        }
}
```

使用接口的代码：

```
unsigned char blue[] = {0, 0, 255};
img.draw_circle(50, 50, 30, blue);
```

两种情况分别的运行结果：(不使用 | 使用)


Point (434,13) = [102 102 102]



分析两者差异：

不使用CImg的draw_circle()接口画出的圆相对于使用了的图像来说，边缘锯齿数量多了一些，但是总体效果上差不多，因为CImg本身使用的方法就是直接判断点与圆心之间的距离来染色的。

4. 与上一步骤相同，采取两种方法绘制圆心为(50, 50)，半径为3的黄色圆形区域，比较差异

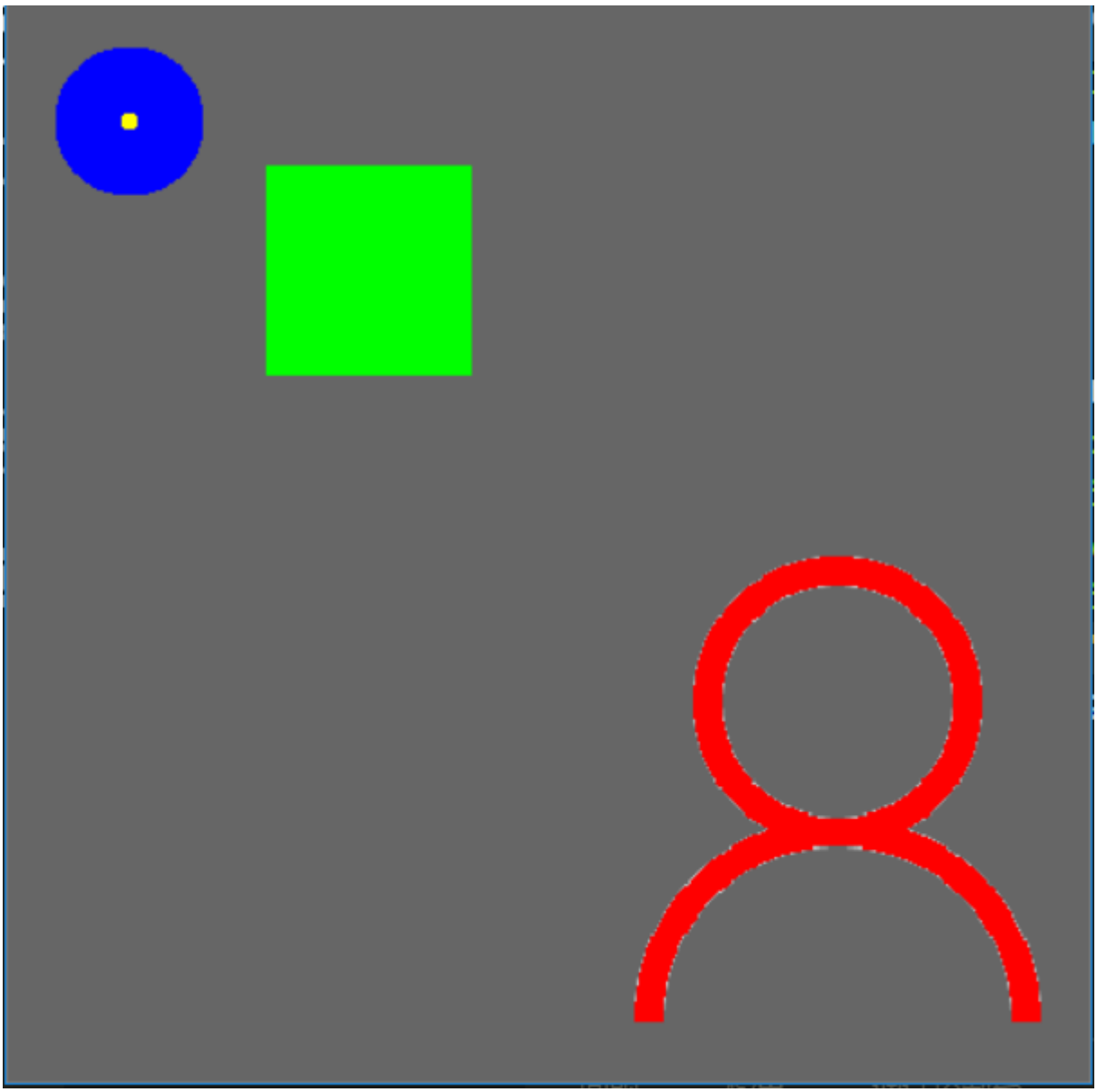
不使用接口的代码：

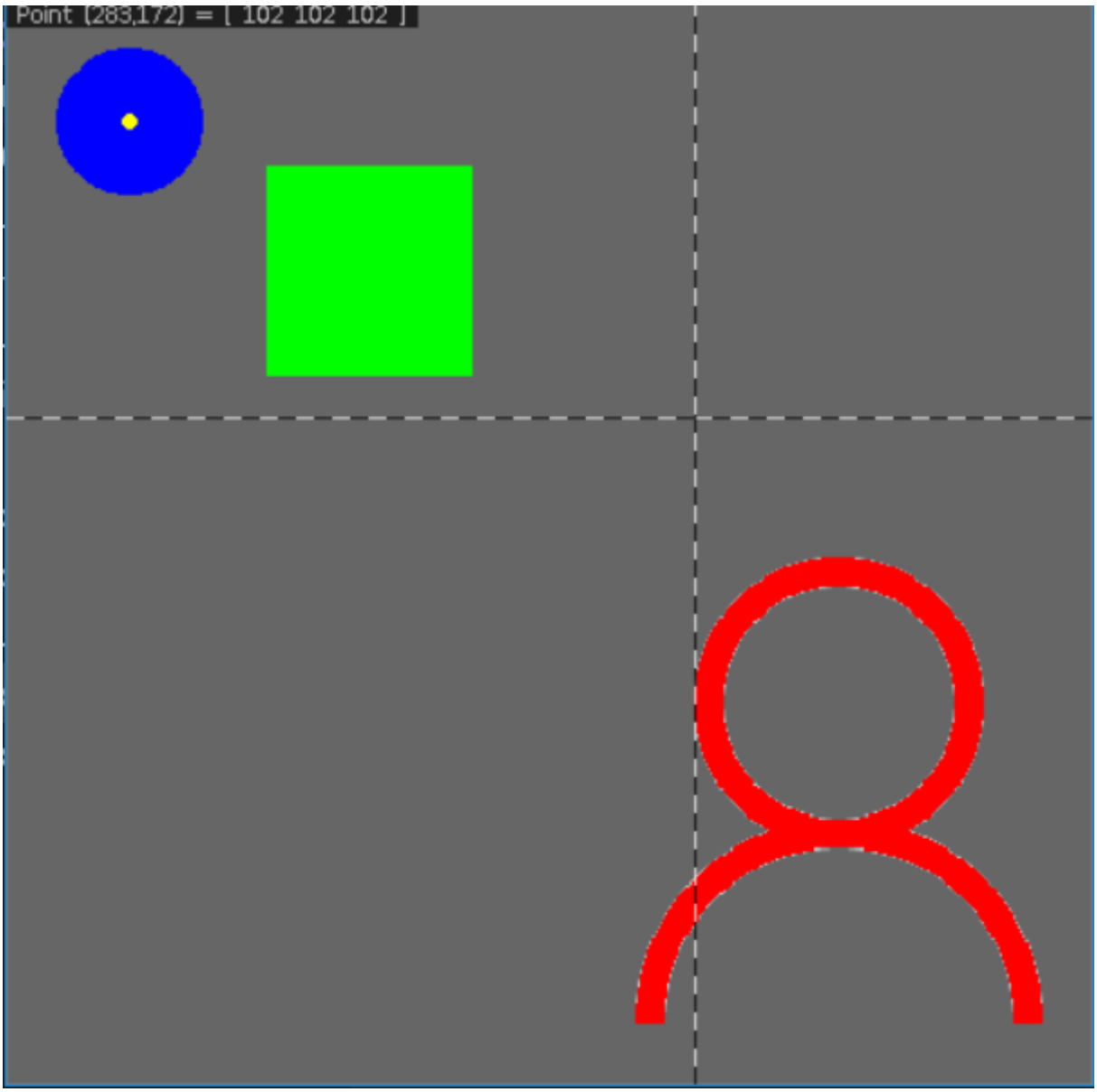
```
void drawYellowCircle(pair<int, int> center, int radius){
        cimg_forXY(img, x, y){
            if(getDistance(center, make_pair(x, y)) <= radius){
                img(x, y, 0) = img(x, y, 1) = 255;
                img(x, y, 2) = 0;
            }
        }
}
```

使用接口的代码：

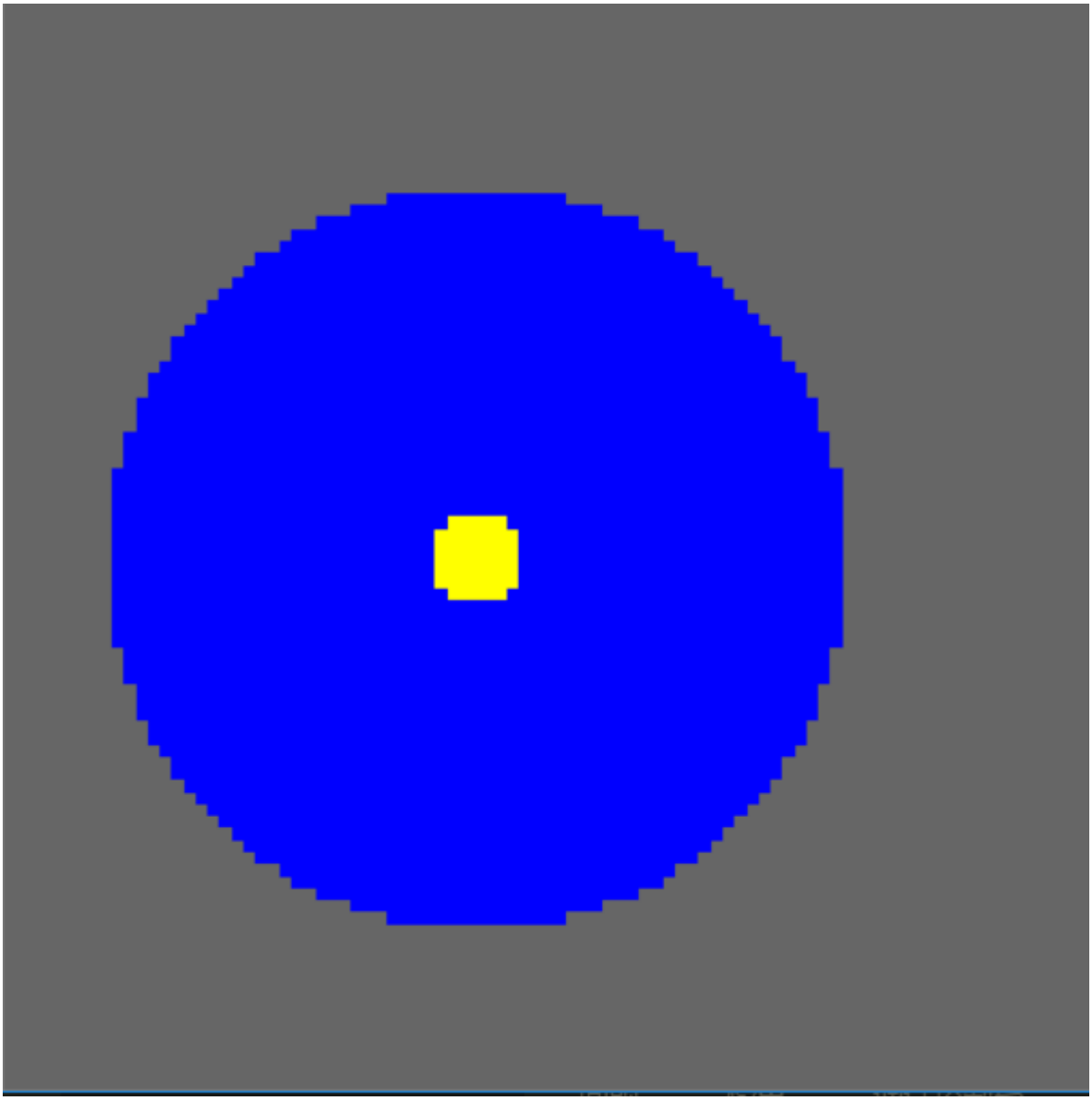
```
unsigned char yellow[] = {255, 255, 0};
img.draw_circle(50, 50, 3, yellow);
```

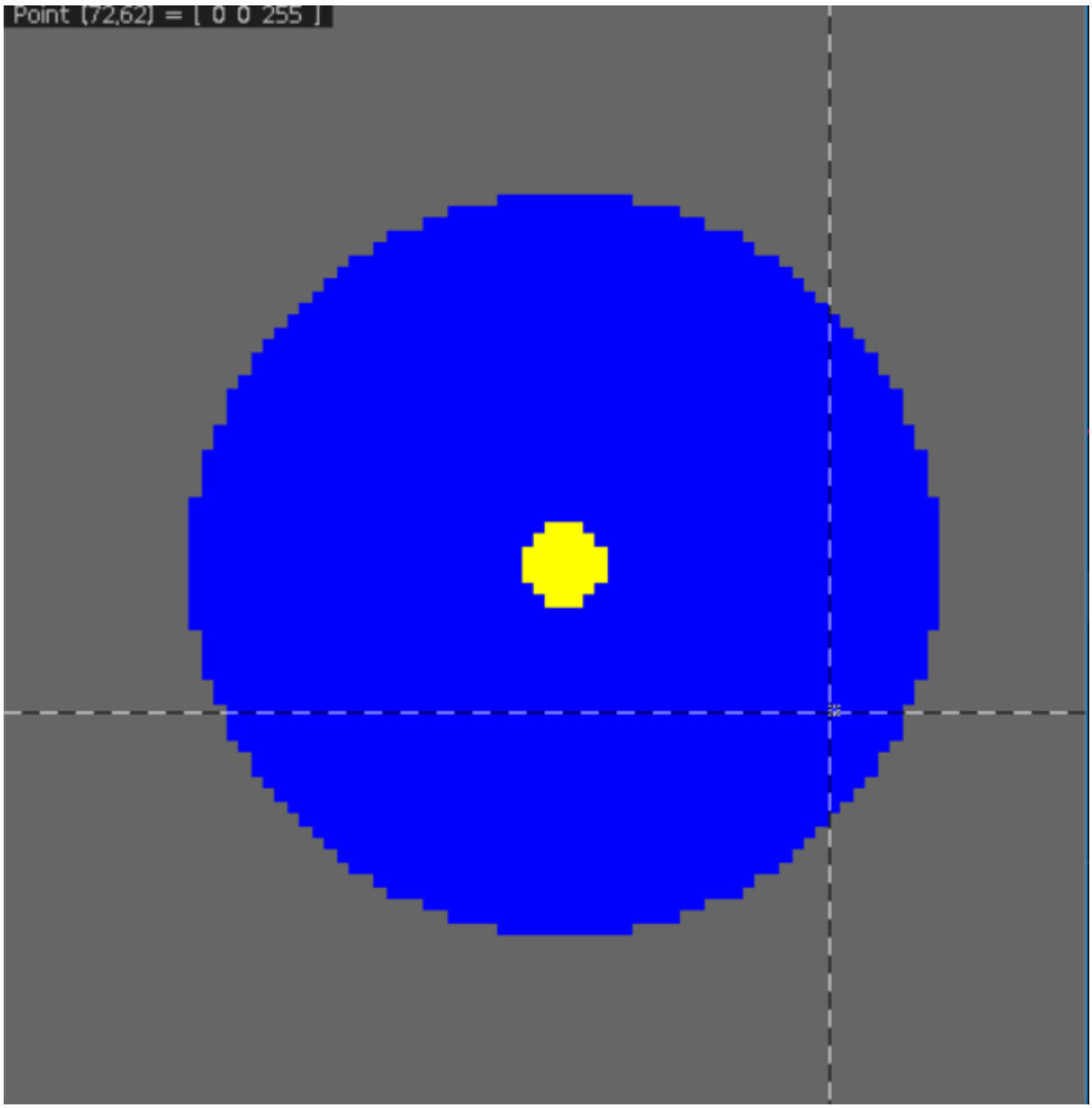
分别的运行结果：(不使用 | 使用)





放大黄色圆形之后的图像：





分析两者差异：

使用CImg类方法接口绘制的黄色圆的锯齿很多，并且由于半径过小，导致显示结果不是很好，丧失了圆形的特征；采用距离判断法绘制的圆则相对来说比较饱和，虽然也并非显示为圆形。

5. 与上一步骤相同，采用两种方法绘起点为(0,0)，角度为35度，长度为100的蓝色直线

方法：使用循环建立100个点，存储当前长度，每个点的横坐标都是该点当前长度乘以cos35度，纵坐标则是当前长度乘以sin35度，由于结果都是浮点数，所以此处采取四舍五入方法取整

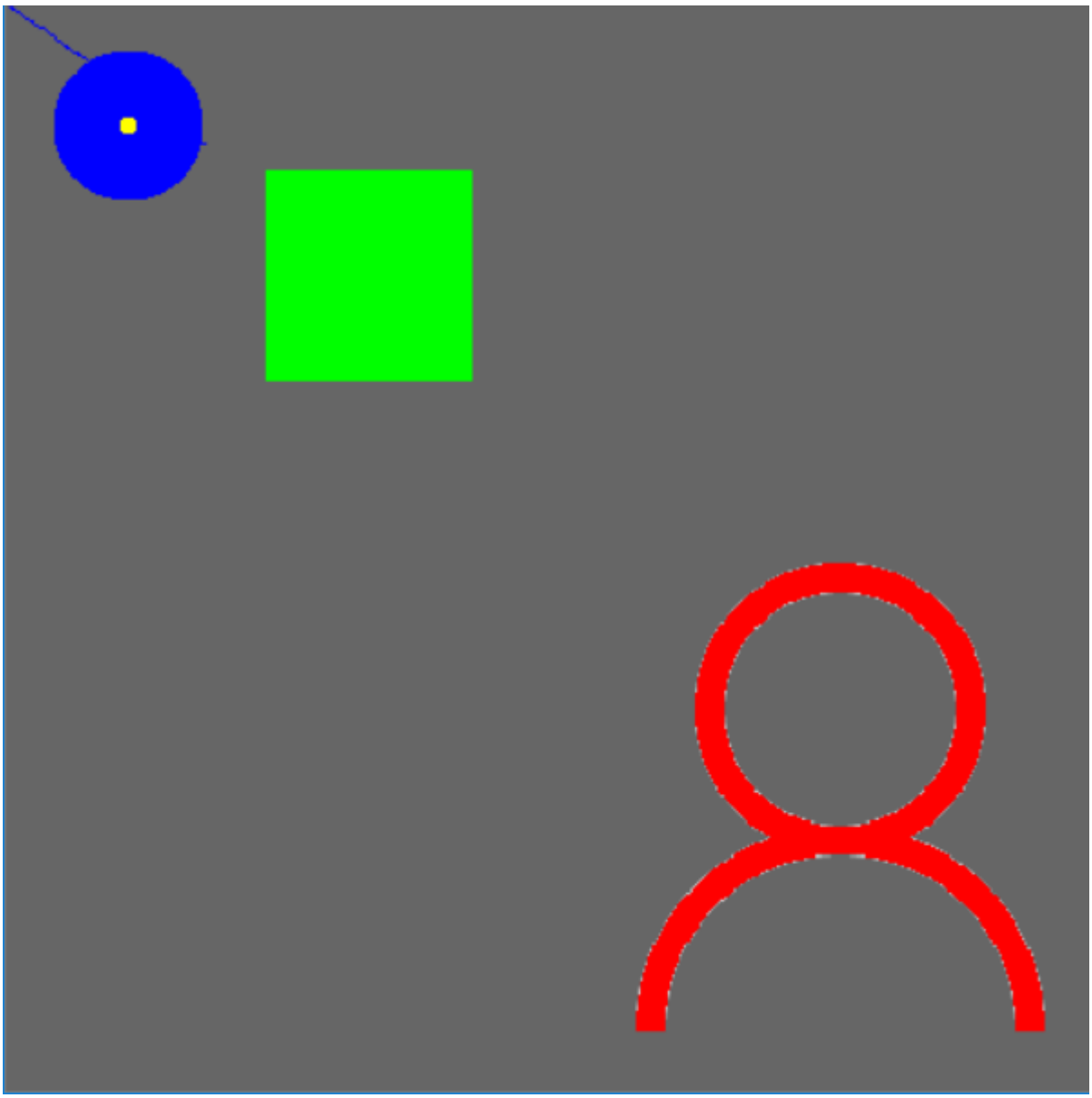
不使用接口代码：

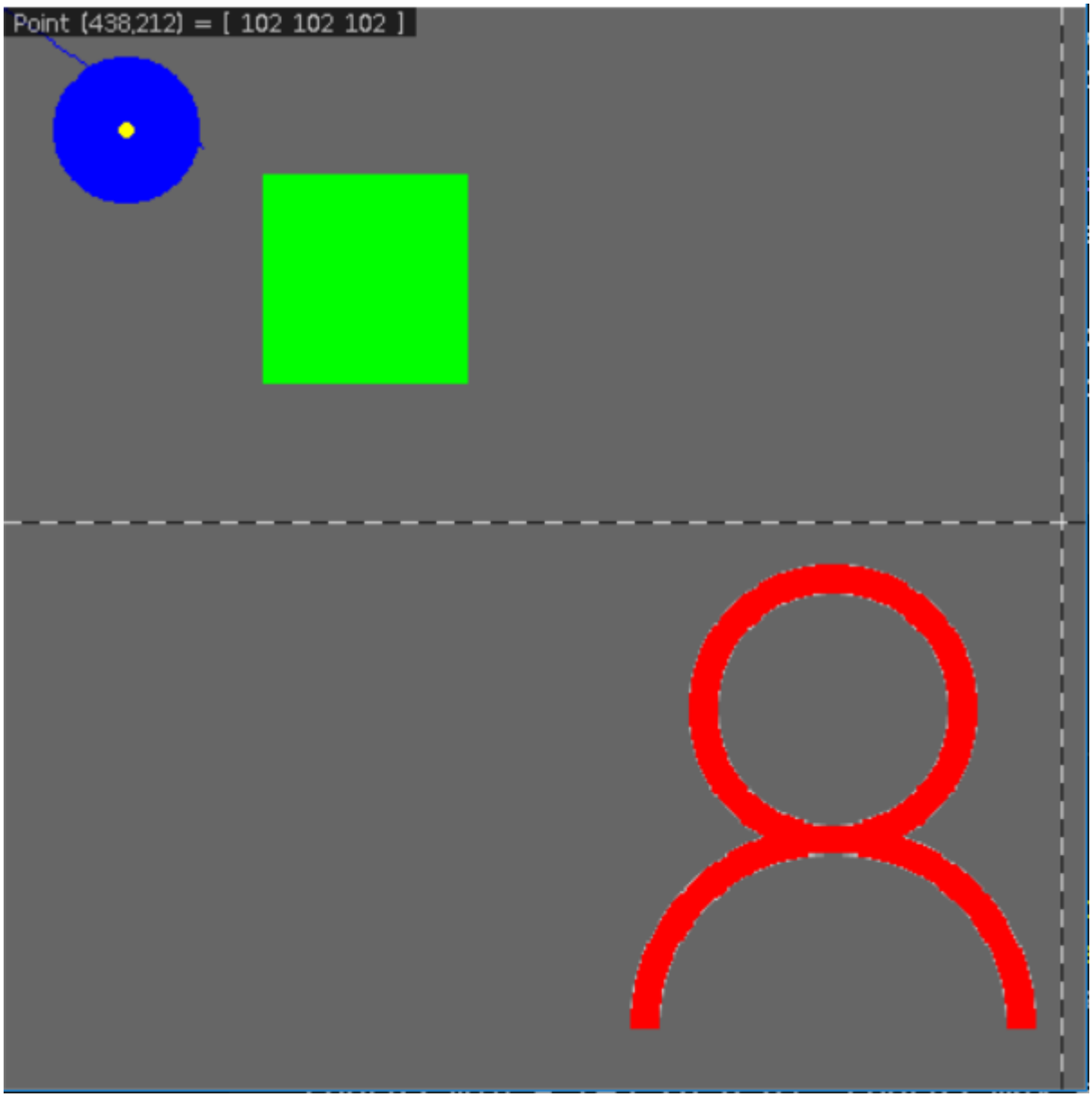
```
void drawBlueLine(pair<int, int> center, double angle, int length){
        for(int i=1; i<=100; i++){
            double x_ = center.first + i * cos(angle/180 * cimg::PI),
                   y_ = center.second + i * sin(angle/180 * cimg::PI);
            // Tansform double into int, with round-up(四舍五入)
            int x = x_ + 0.5, y = y_ + 0.5;
            img(x, y, 0) = img(x, y, 1) = 0;
            img(x, y, 2) = 255;
        }
}
```

使用接口代码：

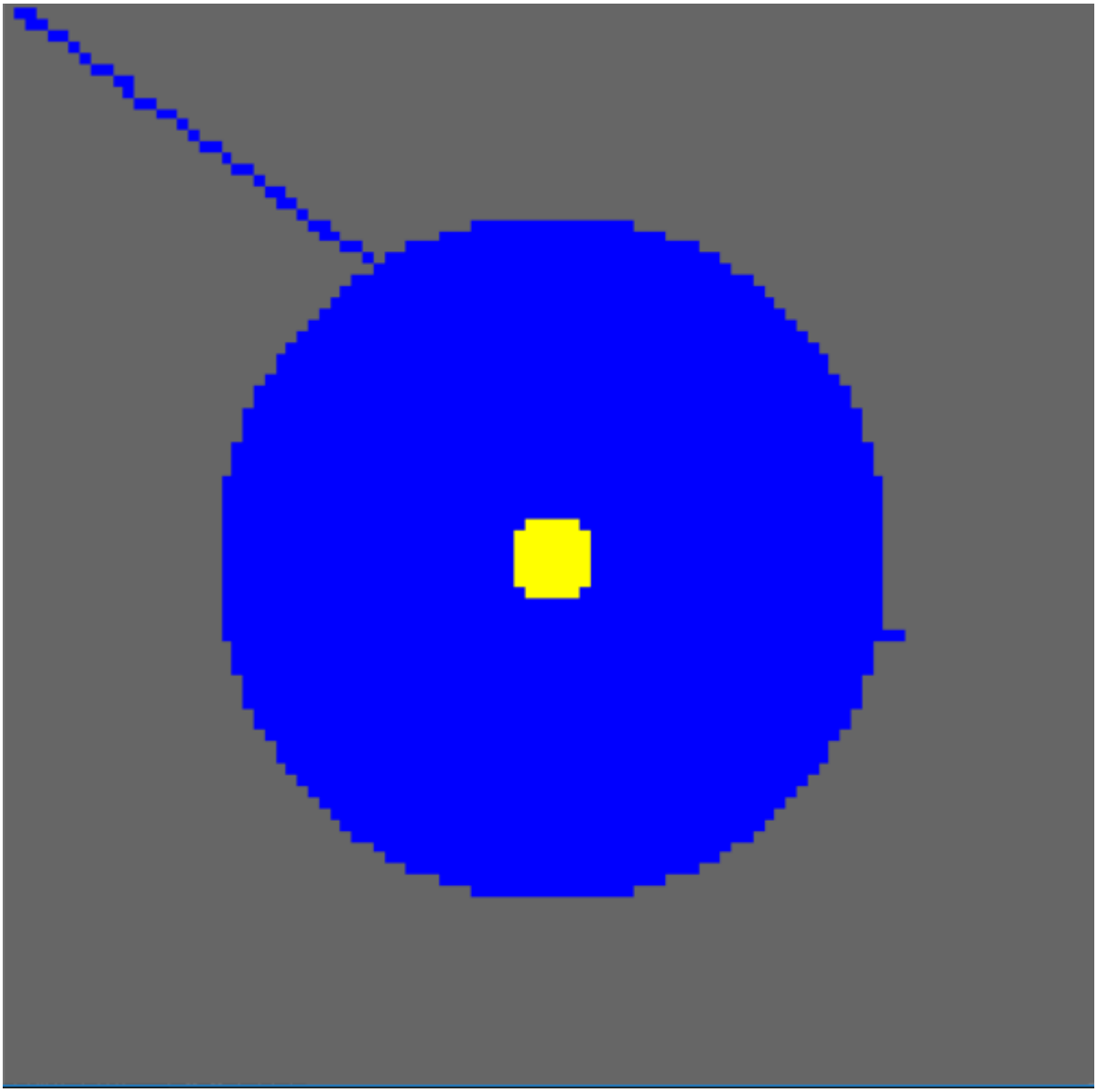
```
int x_ = cos((double)7/36 * cimg::PI)*100 + 0.5, y_ = sin((double)7/36 * cimg::PI)*100 + 0.5;
img.draw_line(0, 0, x_, y_, blue);
```

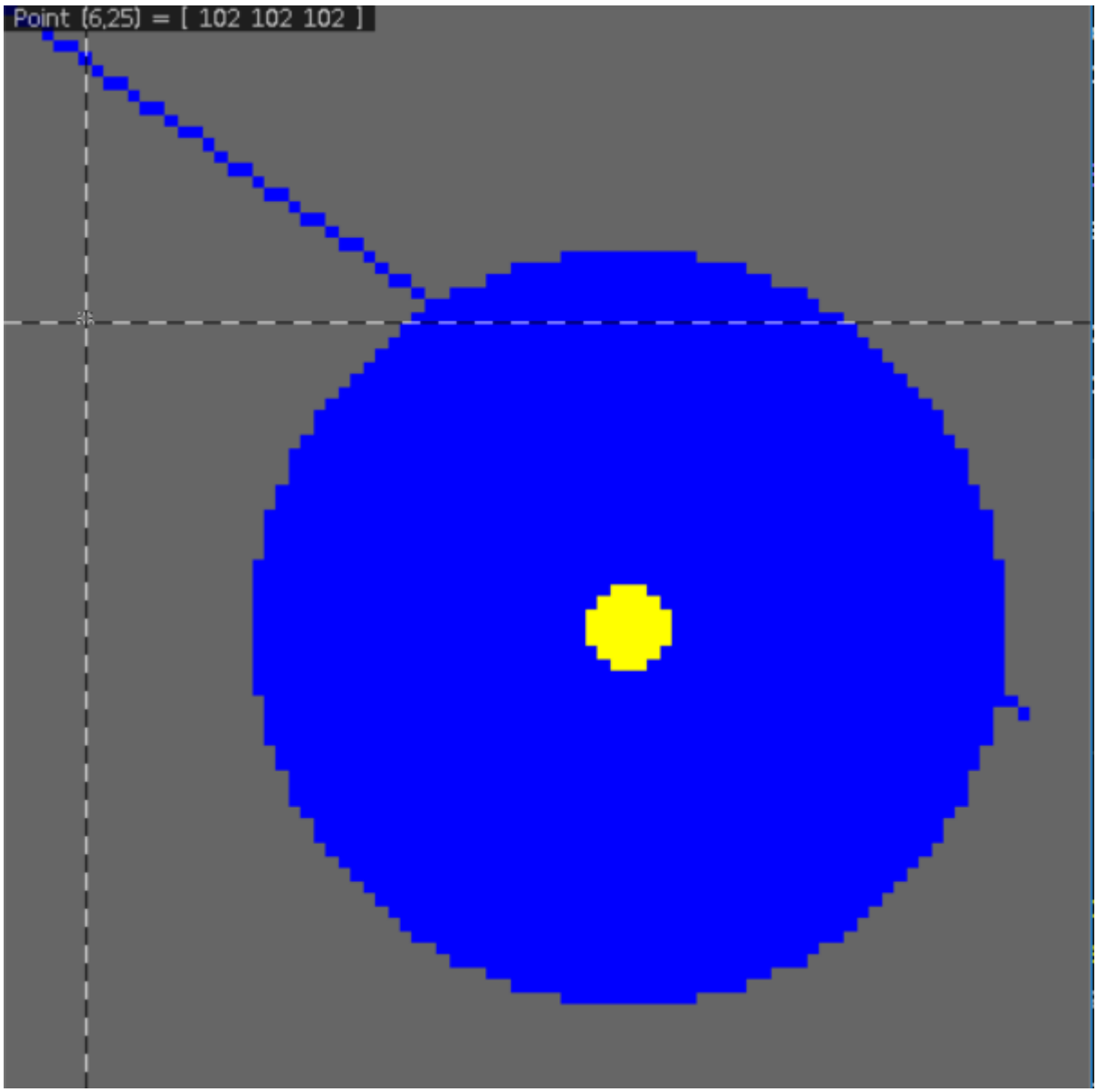
运行结果：





放大蓝色直线之后的图像：





分析结果差异：

CImg原生接口绘制直线采取的是Bresenham算法，所以比起直接循环取sin，cos定坐标的方法要好地多，实际作出的直线也比较平滑。

6. 保存处理之后的图片

代码：

```
// Step 6
img.save("2.bmp");
```

三、实验思考题

- 由于步骤四中的半径过小，用到的像素太少，所以分辨率不高，因此方块感明显，边缘锯齿明显，不圆润，不连续，形状不好，看起来有点像十字形。一般的圆形的外缘有更多的像素点组成，但同样地在方块像素图

里面就需要半径足够大，看起来锯齿才会没那么明显，方块感也减少，因此方块越密集，给人的视觉效果就越平滑，越连续。