实验名称：图像读取和显示以及像素操作

实验准备：

1. 下载Cimg运行库
2. 学习Cimg库的相关接口操作

实验步骤：

1. 读入 1.bmp 文件，并用 CImg.display() 显示。
2. 把 1.bmp 文件的白色区域变成红色，黑色区域变成绿色。
3. 在图上绘制一个圆形区域，圆心坐标(50,50)，半径为 30，填充颜色为蓝色。
4. 在图上绘制一个圆形区域，圆心坐标(50,50)，半径为 3，填充颜色为黄色。
5. 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 35 度，直线 的颜色为蓝色。
6. 把上面的操作结果保存为 2.bmp。

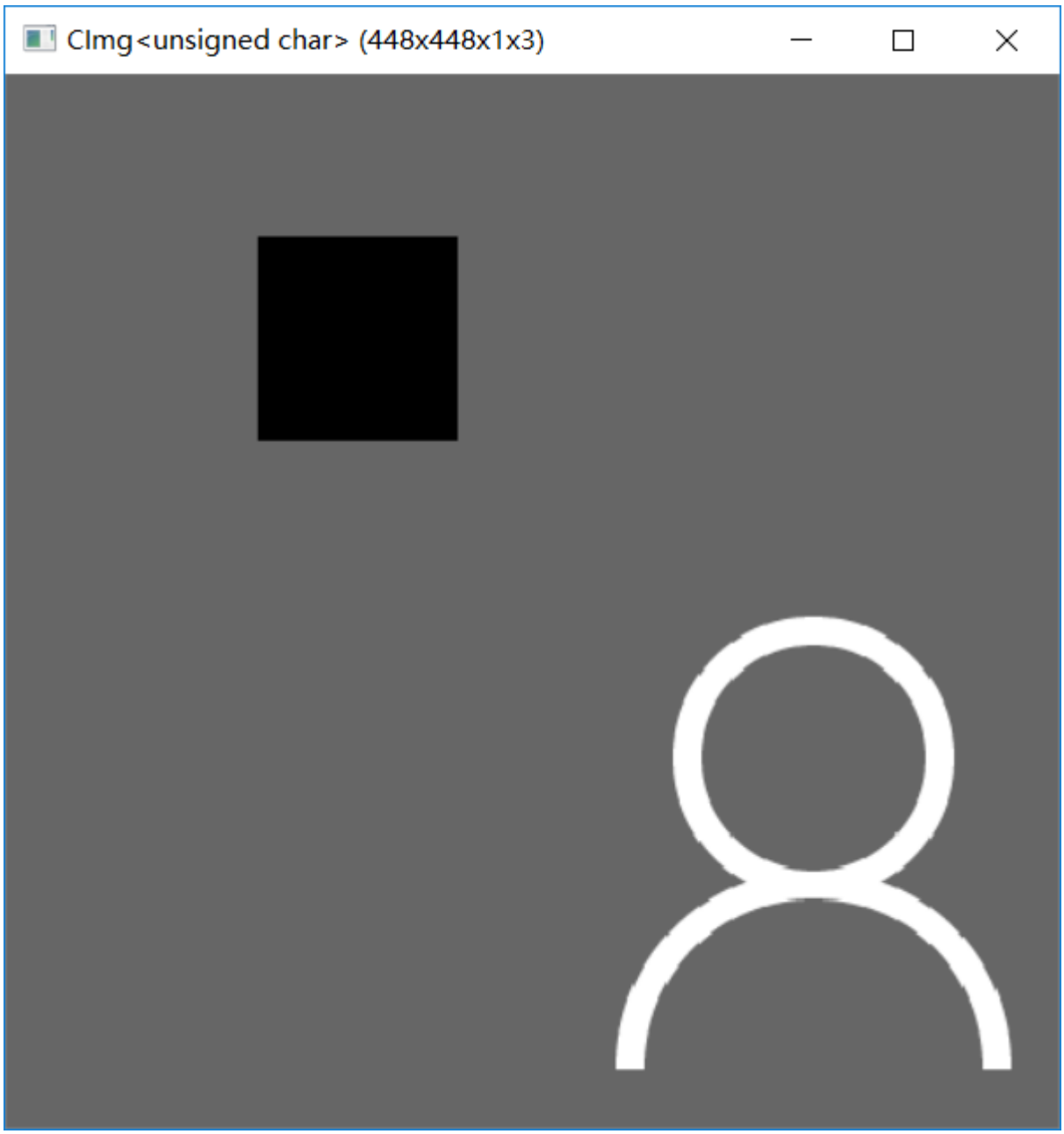
实验过程

一、读入 1.bmp 文件，并用 CImg.display() 显示。

这是最简单的一步，只需注意用对Cimg的api：load_bmp以及display，当然，要先创建一个Cimg类型的实体。

```
CImg<unsigned char> img;
img.load_bmp("1.bmp");
img.display();
```

display结果：

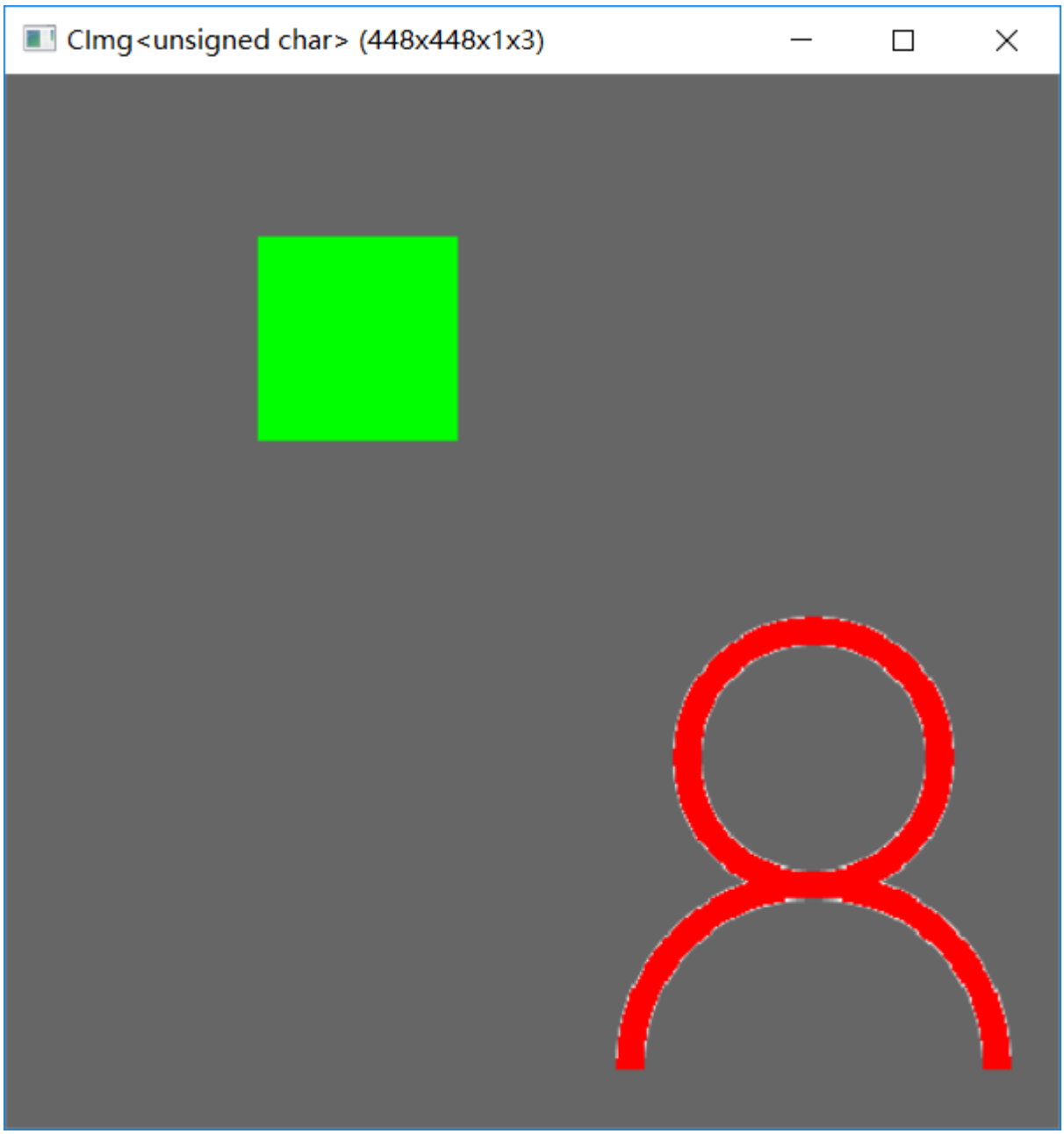


二、把 1.bmp 文件的白色区域变成红色，黑色区域变成绿色。

这一步需要用到cimg_forXY这个宏定义，它的作用是遍历图片的每个像素点，这样就可以修改每个像素点的属性值了。我们要做的就是判断像素点的颜色，并修改对应颜色像素点的颜色。

```
cimg_forXY(img, x, y){
    if(img(x, y, 0) == 255 && img(x, y, 1) == 255 && img(x, y, 2) == 255){
        img(x, y, 0) = 255;
        img(x, y, 1) = 0;
        img(x, y, 2) = 0;
    }
    else if(img(x, y, 0) == 0 && img(x, y, 1) == 0 && img(x, y, 2) == 0){
        img(x, y, 0) = 0;
        img(x, y, 1) = 255;
        img(x, y, 2) = 0;
    }
}
```

转变结果：

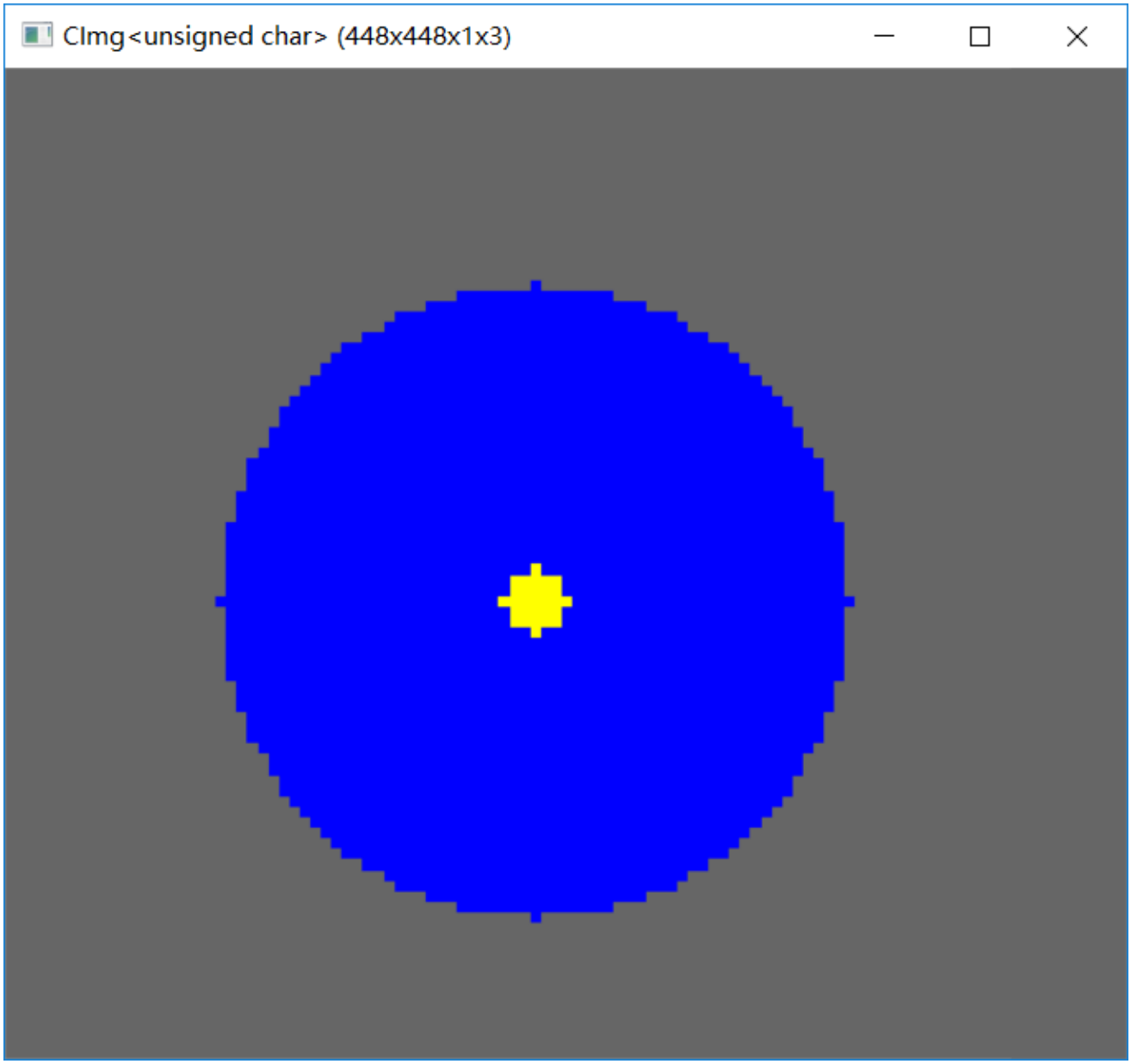


三+四、在图上绘制一个圆形区域，圆心坐标(50,50)，半径为 30，填充颜色为蓝色；在图上绘制一个圆形区域，圆心坐标(50,50)，半径为 3，填充颜色为黄色。

这两步都为画圆，首先是不使用CImg库中函数的实现方法。我把实现方法封装在了一个类中：

```
class Editimg{
    private:
        CImg<unsigned char> img;
    public:
        void drawscircle(){
            cimg_forXY(img, x, y){
                if((x - 50) * (x - 50) + (y - 50) * (y - 50) <= 9){
                    img(x, y, 0) = 255;
                    img(x, y, 1) = 255;
                    img(x, y, 2) = 0;
                }
            }
        }
        void drawbcircle(){
            cimg_forXY(img, x, y){
                if((x - 50) * (x - 50) + (y - 50) * (y - 50) <= 900){
                    img(x, y, 0) = 0;
                    img(x, y, 1) = 0;
                    img(x, y, 2) = 255;
                }
            }
        }
```

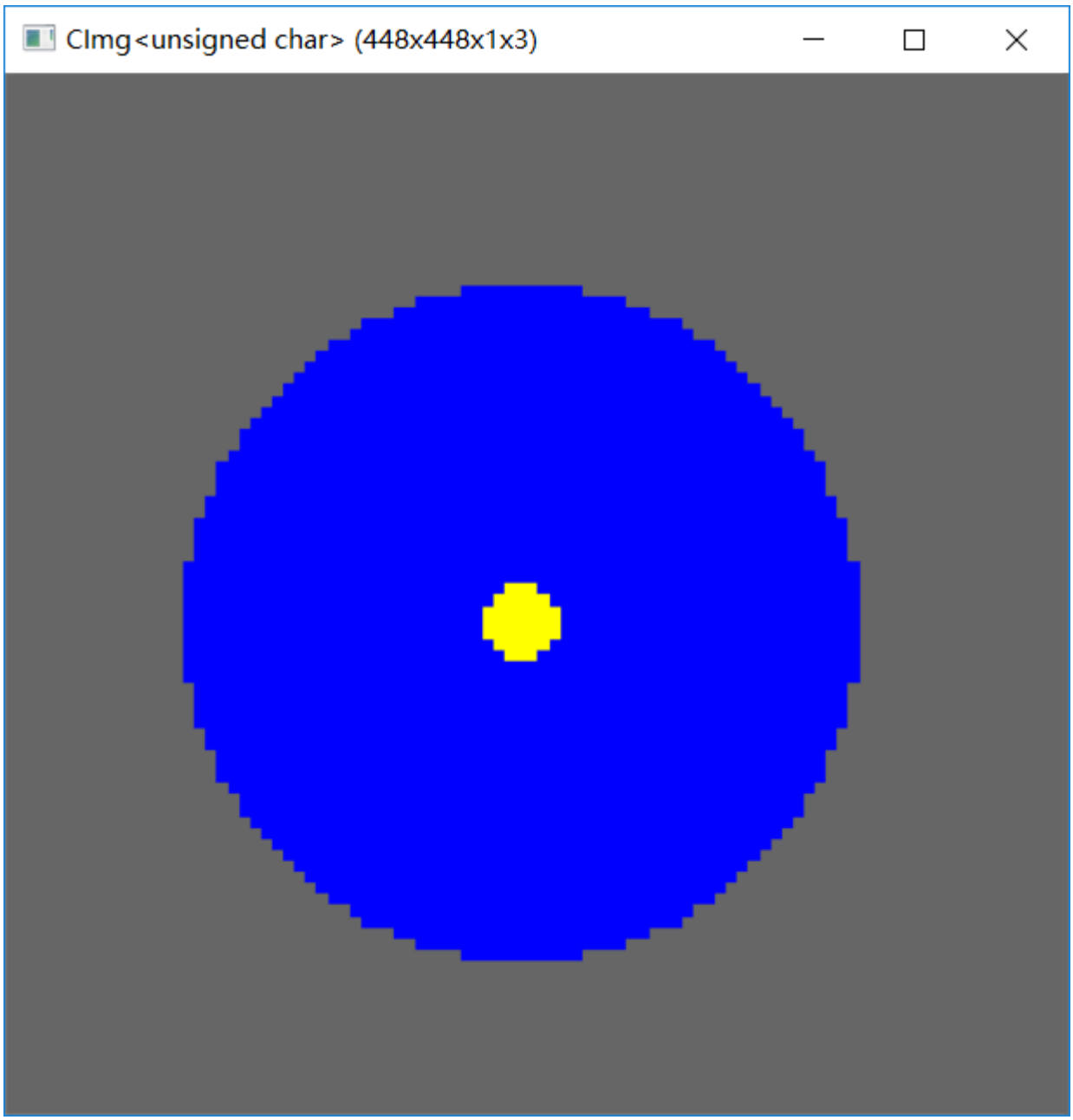
其中，drawscircle画的是半径为3的小圆，drawbcircle画的是半径为30的大圆，可以看到，我使用的仍是遍历像素点赋值的方法，根据像素点与圆心的距离来判断要不要修改它的颜色。这样实现的效果是：



而如果使用CImg提供的画圆函数：

```
img.draw_circle(50, 50, 30, blue); //unsigned char blue[] = {0, 0, 255};
img.draw_circle(50, 50, 3, yellow); //unsigned char yellow[] = {255, 255, 0};
```

实现效果是这样的：



对比发现，我的实现和CImg库函数的实现还是有些不同，可以说CImg库函数实现画出的圆比我的实现画出的圆要更平滑一些，为什么会有这样的差别呢，回顾一下，我实现的方法是比较像素点与圆心的距离，因为像素点始终是一小块区域，而并非一个点，这就可能在计算时造成误差（同时还有舍入误差等）。

那么CImg库的实现方法是什么呢？为了弄清这个问题我查看了CImg库的源代码，发现在它的draw_circle函数中并不是靠计算距离来给像素点赋值的，它是从要画的圆的一条水平直径开始，向上下递推并调用cimg_draw_scanline这个函数来“画线”——也就是给像素点赋值，在递推过程中只是做了坐标的加减运算，这就避免了计算距离时的乘法运算。

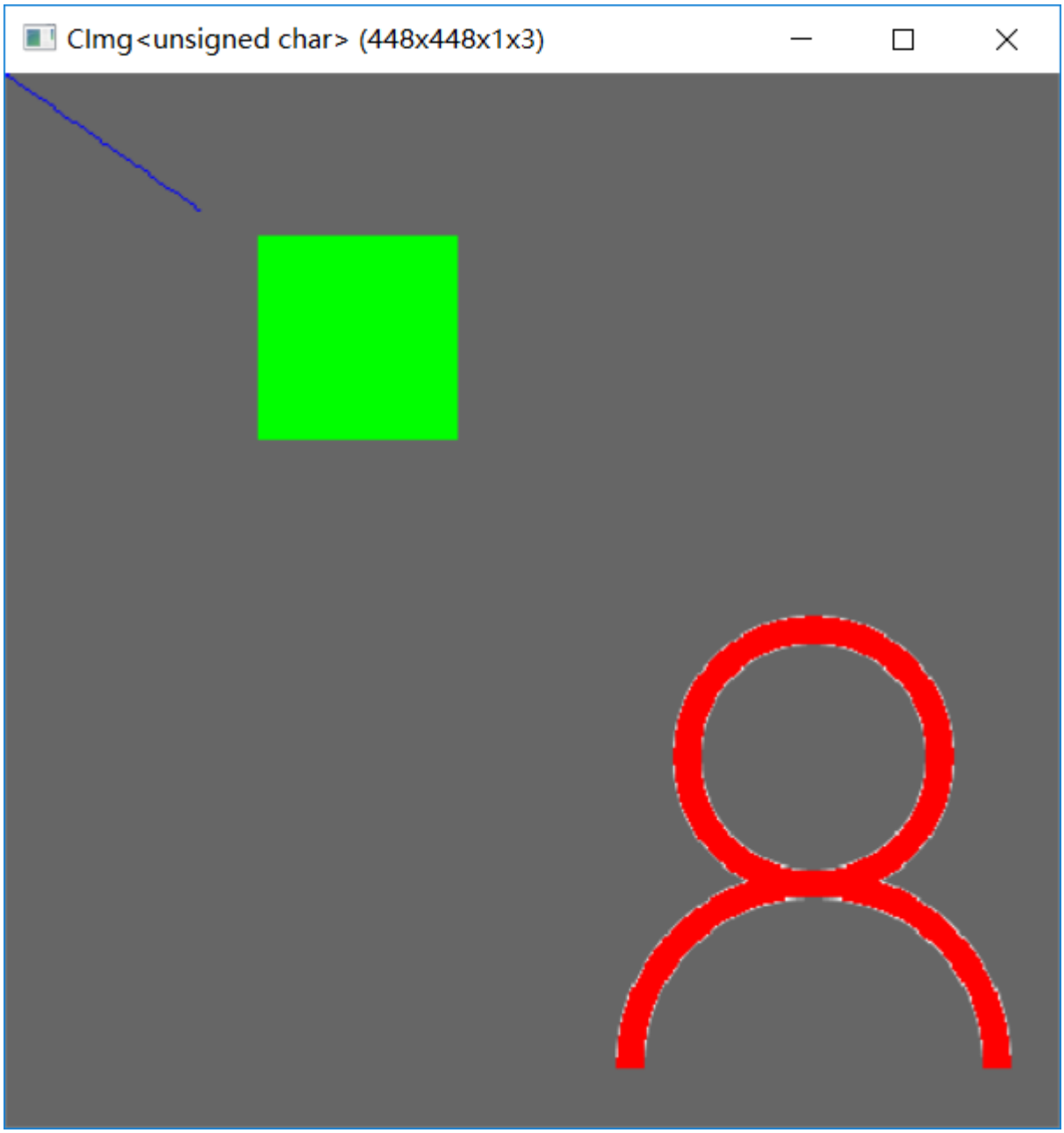
五、在图上绘制一条长为 100 的直线段，起点坐标为(0, 0)，方向角为 35 度，直线 的颜色为蓝色。

同样，首先是不使用CImg库函数的实现方法，一样是封装在了类中：

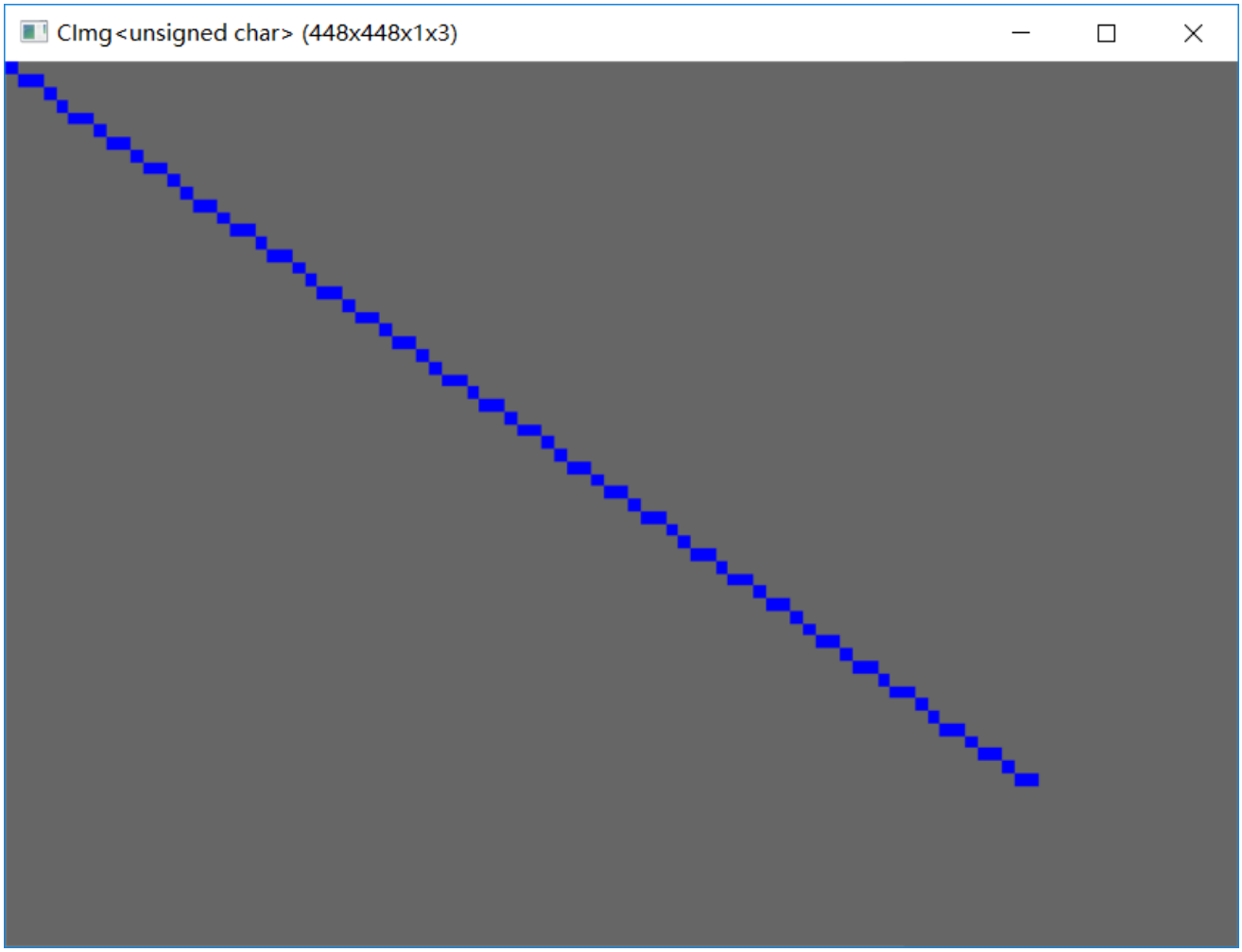
```
void drawline(){
    cimg_forXY(img, x, y){
        if(x*x + y*y <= 10000 && y - x * tan(35.0 * PI / 180.0) < 0.5 && y - x *
tan(35.0 * PI / 180.0) > -0.5){
            img(x, y, 0) = 0;
            img(x, y, 1) = 0;
            img(x, y, 2) = 255;
        }
    }
}
```

我的做法是首先借助勾股定理来确保线段长为100。而在判断角度为35度时，我用到了tan值来做，经过多次调整后，我最终将精度设置为1.0（正负0.5判断），这样能获得一个比较好的结果。

结果（正常大小看）：



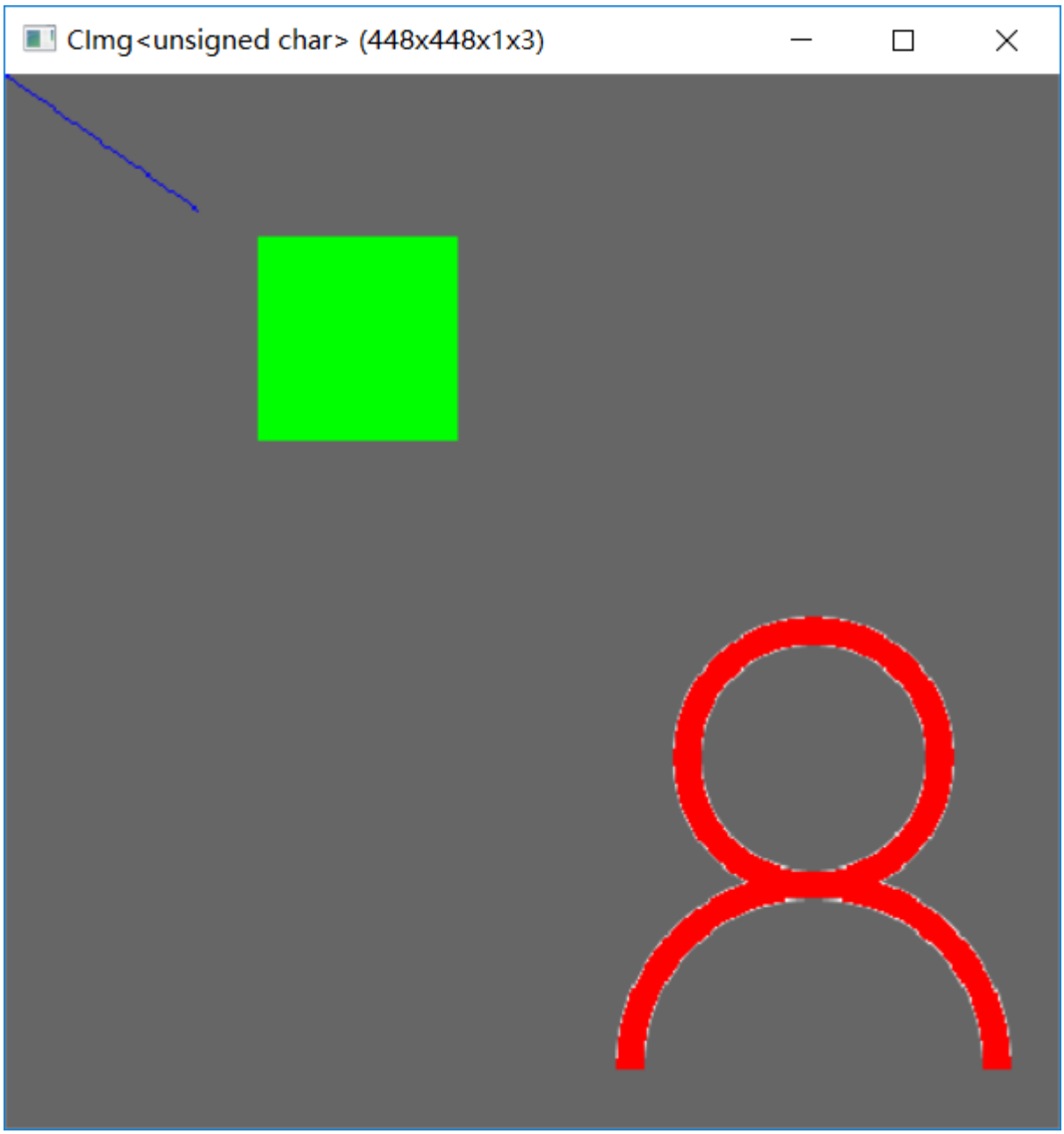
结果（放大看）：



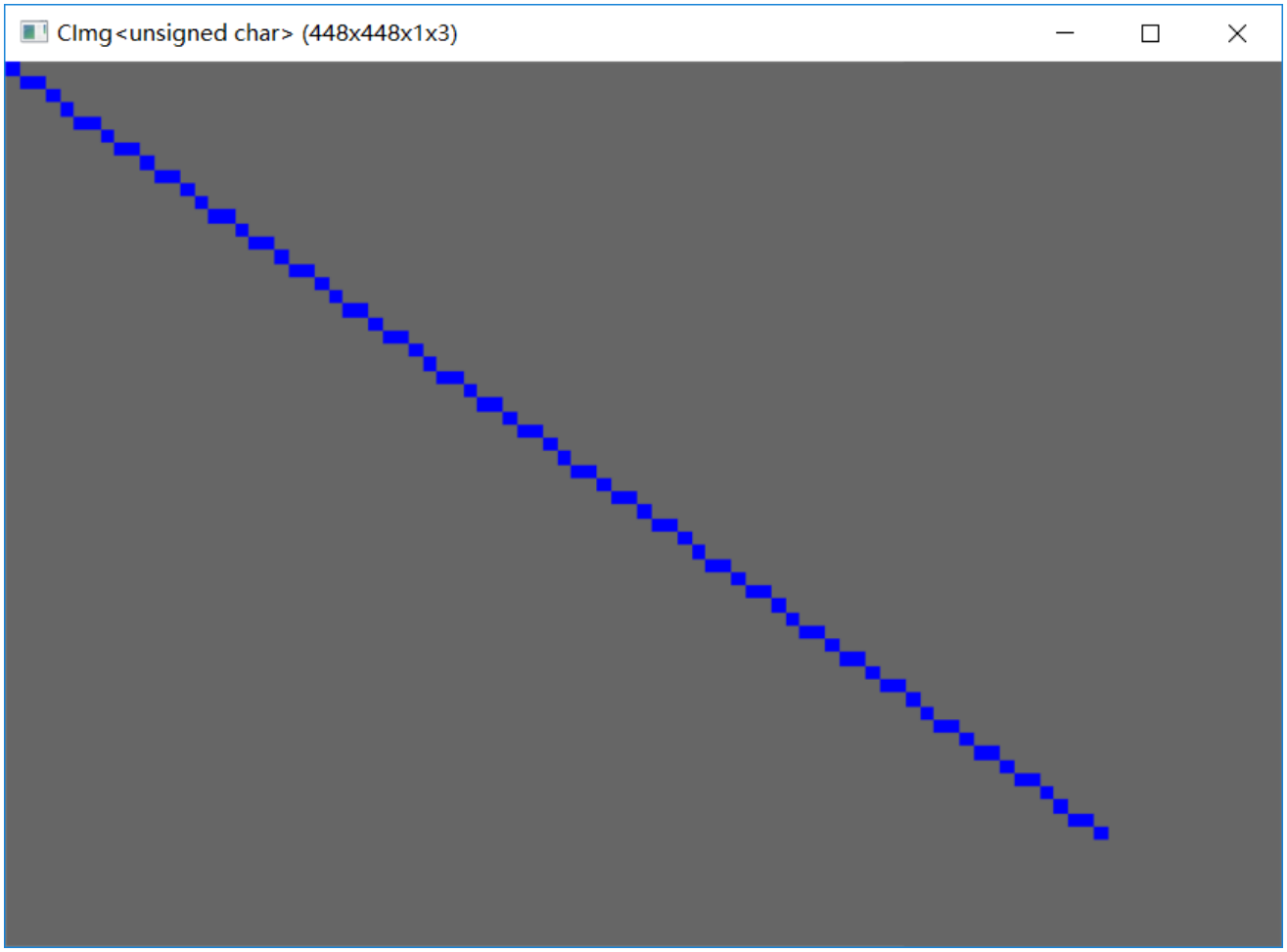
接下来是使用CImg库函数的实现：

```
img.draw_line(0, 0, cos(35.0 * PI / 180) * 100, sin(35.0 * PI / 180) * 100, blue);  //
参数为起点坐标，终点坐标，颜色
```

效果为（正常大小看）：



放大看:



可以看到，在画直线上，我的实现和CImg库函数的实现效果基本相同。

六、把上面的操作结果保存为 2.bmp。

这一步只需要调用CImg库提供的save函数就行了。

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

实验感想

这次的实验主要是通过CImg提供的遍历等方法，熟悉对像素的操作。在实验过程中也遇到了一些问题，比如一开始不知道CImg提供的遍历为什么可以这样来写，后来查看官方文档才发现是宏定义。在画圆那一步中，通过查看源代码也发现了画圆的新方法，再看回自己的实现方法还是比较简陋的。同时，也有一些问题遗留，比如再画直线的实验中，虽然我自己的实现和CImg库函数的实现效果差不多，但是对于库函数的实现机制并没有搞懂。总的来说，这次初步接触了图像处理，也感受到了难度所在。