中山大學
SUN YAT-SEN UNIVERSITY

《计算机视觉》

实验文档

（实验一）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	16 软件工程（数字媒体）
学 生 姓 名 ：	黎汛言
学　　号 ：	16340109
时　　间 ：	2018　年　9　月　14　日

实验一 ：图像读取和显示以及像素操作

一. 实验目的

1. 学习CImg库的基本使用方法，进行简单的图像操作；

二. 实验内容

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，直线的颜色为蓝色；
6. 把上面的操作结果保存为2.bmp

三. 实验环境

Windows 10 64位

四. 实验代码

1. 读入1.bmp文件，并用CImg.display()显示；

   编写MyImg类。在构造函数中，根据路径，使用CImg.load_bmp()读入图片文件，并在函数display()中使用CImg.display()来显示读入的图像。

   MyImg类中的CImg对象、构造函数、display()函数如下：

```
class MyImg {
    private:
        // CImg对象
        CImg<unsigned char> img;
```

```
public:
    // 构造函数
    MyImg(const char* path) {
        img.load_bmp(path);
    }

    // 显示图像
    void display() {
        img.display();
    }
```

2. 把1.bmp文件的白色区域变成红色，黑色区域变成绿色；

编写MyImg类的change_color()函数。为了让红色到灰色的过渡更自然，对于三个颜色分量均大于或等于112的灰色像素，绿色和蓝色分量都改为0：

```
// 改变颜色
void change_color() {
    cimg_forXY(img, x, y) {
        if (img(x, y, 0) >= 112 && img(x, y, 1) >= 112 && img(x, y, 2) >= 112) {
            // 白色区域变红
            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) {
            // 黑色区域变绿
            img(x, y, 1) = 255;
        }
    }
}
```

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；

编写MyImg类的draw_blue_circle_1()函数，不使用CImg函数调用，利用勾股定理算法画圆（半径增加0.5以优化边缘效果）：

```
// 不调用CImg函数画蓝色圆
void draw_blue_circle_1() {
    cimg_forXY(img, x, y) {
        // 当前像素与(50, 50)的距离
        double distance = sqrt(pow(x - 50, 2) + pow(y - 50, 2));

        if (distance <= 30.5) {
            img(x, y, 0) = 0;
            img(x, y, 1) = 0;
            img(x, y, 2) = 255;
        }
    }
}
```

编写MyImg类的draw_blue_circle_2()函数，使用CImg.draw_circle函数画圆：

```
// 调用CImg函数画蓝色圆
void draw_blue_circle_2() {
    unsigned char blue[] = {0, 0, 255};
    img.draw_circle(50, 50, 30, blue);
}
```

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；

编写MyImg类的draw_yellow_circle_1()函数，不使用CImg函数调用，利用勾股定理算法画圆（半径增加0.5以优化边缘效果）：

```
// 不调用CImg函数画黄色圆
void draw_yellow_circle_1() {
    cimg_forXY(img, x, y) {
        // 当前像素与(50, 50)的距离
        double distance = sqrt(pow(x - 50, 2) + pow(y - 50, 2));

        if (distance <= 3.5) {
            img(x, y, 0) = 255;
            img(x, y, 1) = 255;
            img(x, y, 2) = 0;
        }
    }
}
```

编写MyImg类的draw_yellow_circle_2()函数，使用CImg.draw_circle函数画圆：

```
// 调用CImg函数画黄色圆
void draw_yellow_circle_2() {
    unsigned char yellow[] = {255, 255, 0};
    img.draw_circle(50, 50, 3, yellow);
}
```

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，直线的颜色为蓝色；

编写MyImg类的draw_line_1()函数，不使用CImg函数调用，利用Bresenham算法逐个像素操作：

```
// 不调用CImg函数画蓝色线（Bresenham算法）
void draw_line_1() {
    // 直线终点的x坐标
    double x_end = 100 * cos(35 * M_PI / 180);

    cimg_forXY(img, x, y) {
        if (x <= x_end) {
            // x坐标对应的理论y坐标
            double y_1 = tan(35 * M_PI / 180) * x;

            // 若y_1坐标位于[y - 0.5, y + 0.5)区间内，则填充蓝色
            if (fabs(y - y_1) < 0.5 || y - y_1 == 0.5) {
                img(x, y, 0) = 0;
                img(x, y, 1) = 0;
                img(x, y, 2) = 255;
            }
        }
    }
}
```

编写MyImg类的draw_line_2()函数，使用CImg.draw_line函数画直线：

```
// 调用CImg函数画蓝色线
void draw_line_2() {
    unsigned char blue[] = {0, 0, 255};
    img.draw_line(0, 0,
        100 * cos(35 * M_PI / 180), 100 * sin(35 * M_PI / 180), blue);
}
```

6. 把上面的操作结果保存为2.bmp；

编写MyImg类的save方法，保存到指定的路径：

```
// 保存图像
void save(const char* path) {
    img.save(path);
}
```

五. 测试代码及结果

1. 读入1.bmp文件，并用CImg.display()显示；

```
// step1: 读入1.bmp文件，并用CImg.display()显示
MyImg img = MyImg("1.bmp");
img.display();
```

2. 第2~6步；

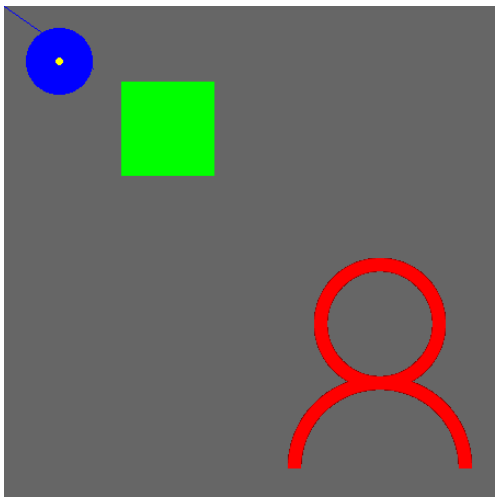
不使用CImg函数调用：

```
// step2: 把1.bmp文件的白色区域变成红色，黑色区域变成绿色
img.change_color();

// step3: 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
img.draw_blue_circle_1();

// step4: 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
img.draw_yellow_circle_1();

// step5: 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，直线的颜色为蓝色
img.draw_line_1();

// step6: 把上面的操作结果保存为2.bmp
img.save("2(1).bmp");
```

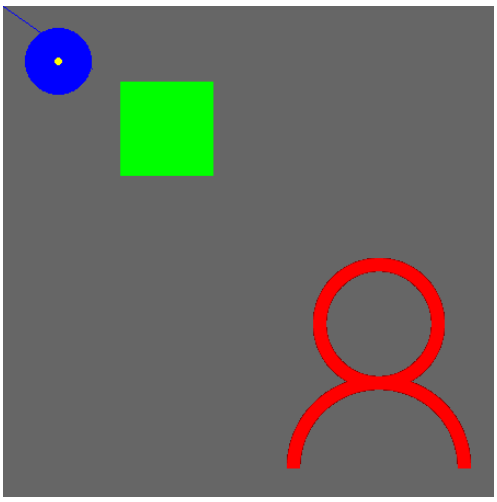
使用CImg函数调用：

```
// step2: 把1.bmp文件的白色区域变成红色，黑色区域变成绿色
img.change_color();

// step3: 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
img.draw_blue_circle_2();

// step4: 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
img.draw_yellow_circle_2();

// step5: 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，直线的颜色为蓝色
img.draw_line_2();

// step6: 把上面的操作结果保存为2.bmp
img.save("2(2).bmp");
```

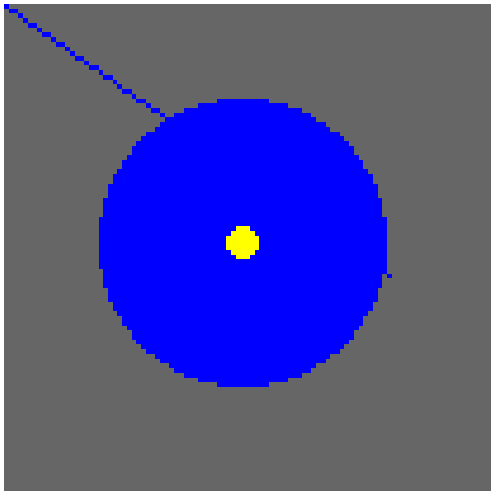
效果对比：



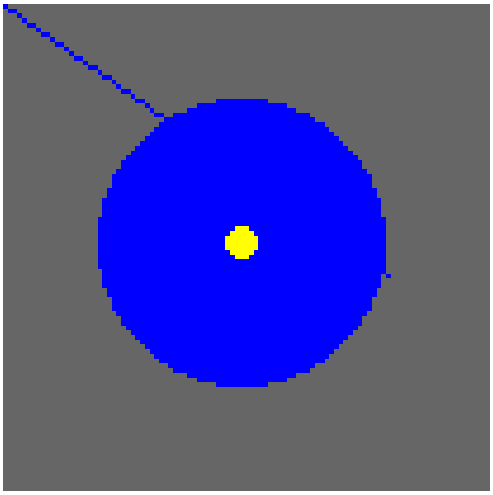
不使用CImg函数调用（优化后）



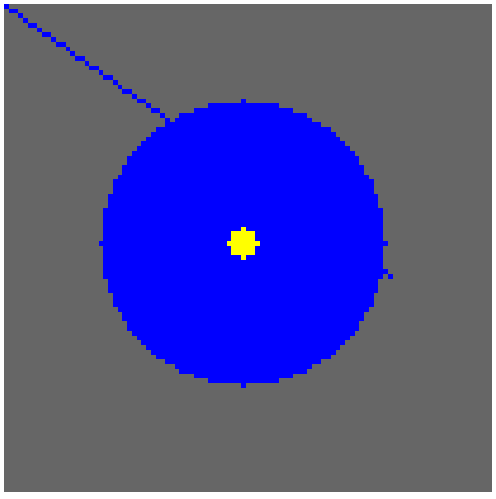
使用CImg函数调用



不使用CImg函数调用（优化后）



使用CImg函数调用



不使用CImg函数调用（优化前）

六. 结果分析

从上面的图像可以看出，使用优化后的勾股定理算法画圆，使用Bresenham算法画直线，结果的精度和平滑度都可以和使用CImg函数调用得到的结果相媲美。不管采用哪种方法，第四步绘制的黄色圆形都没有太好的效果，原因是半径为3的圆形包含的像素点太少。我们很难用太少的离散像素点去近似地表示出平滑的圆周。

为什么优化后的圆形效果比优化前的更好？如果将圆的半径固定为3或30，由于像素点的坐标都是整数，在使用勾股定理计算后，圆周经过的许多像素区域会被判定为在圆外。只有像素中心点在圆周以内的像素区域会被颜色填充。这就导致了圆周的不光滑，会有突起的部分。如果允许半径有0.5的误差，就能适当补充圆周上的像素，使圆周更加平滑。