作业一实验报告及测试文档

数据科学与计算机学院 梁育诚 16340133

一、实验内容

输入图像：

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。

要求 ：

1. 对于上面的第三 、四、五步，先不用 CImg 的函数调用，绘制出相应图形。然后再调用 CImg 相关函数绘制出相应的图形。并在实验文档中对比两者的差异。
2. 把代码写成类的形式，把上面的第三、四、五步操作封装为类的操作。

二、实验过程

首先是前两个要求，需要显示图片和替换指定颜色。使用 CImg 库导入和现实图片的代码在老师给出的样例代码中已经有，直接使用就可以，为了封装的需要，我将这一步的代码写在了构造函数中。替换指定颜色的原理很简单，使用 cimg_forXY 遍历图片中的每一个 pixel，对其颜色进行判定，若符合要求则将对应 pixel 的颜色替换为题目要求的颜色即可。

然后是画图部分。该部分分为两小部分，第一部分是使用 CImg 库函数画圆、画直线，第二部分是不使用 CImg 库，通过像素点的方式进行绘画。

1. 使用 CImg 库。这一操作比较简单，无论是画圆还是画直线，只需要给出相应的参数就可以了。需要注意，画图的函数一般需要传入一个颜色，是 unsigned char 数组，由于整个代码中多个地方用到了颜色，因此我将颜色定义为全局的 const 变量，减少程序出错的可能。特别需要注意的是，画直线的时候需要给出直线的终点，我们需要通过长度 100 和角度 35° 这两个信息计算出顶点的位置。

2. 不使用 CImg 库。不直接使用 CImg 库的话，就需要遍历每一个 pixel 来画图了。原理很简单，我们可以检查每个点是否满足圆或直线的方程，如果满足，则填充相应的颜色。圆的方程是$(x-x_0)^2+(y-y_0)^2=r^2$，直线方程是$y=kx+b$。

使用直线方程的方法画直线得出的效果一般不是很理想，因此需要使用更加有效的方式来绘制直线，其中比较出名的方法是基于数值微分思想的 DDA 算法。DDA 算法使用了步长的思想，每次推进一小步，逐个像素点画出直线。而我使用了 DDA 的改进版 Bresenham 算法，相比起 DDA，它避免了在步进时的浮点数运算，提高了运算效率，而且在步进过程中以加法替代了乘法。但在这个实验中，因为线段的长度较短，而且在使用直线方程的过程中我也考虑到了精度问题，因此两个方法画出来的直线效果都比较不错。

三、测试

自己封装的类为 MyImage，放在了 hw1.hpp 中

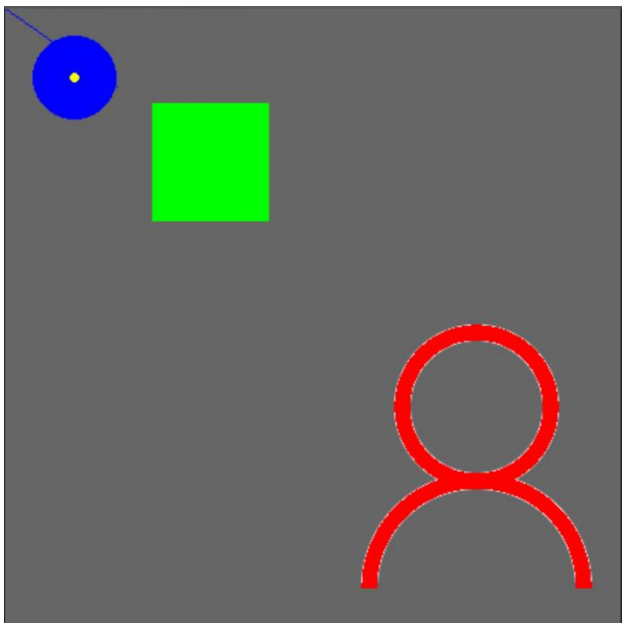
测试代码如下 hw1.cpp：

```
#include <iostream>
#include "hw1.hpp"
using namespace std;

int main() {
   // Prompt user to choose
   int cmd;
   cout << "Please input your choice:" << endl;
   cout << "[0]: use CImg functions" << endl;
   cout << "[1]: without using CImg functions" << endl;
   cout << "Your Choice: " << endl;
   cin >> cmd;

   MyImage m;
   if (cmd == 0) {
      m.changeColor();
      m.drawBlueCircle();
      m.drawYellowCircle();
      m.drawBlueLine();
   }
   else if (cmd == 1) {
      m.changeColor();
      m.drawBlueCircleByMe();
      m.drawYellowCircleByMe();
      m. Bresenham ();
   }
   cout << "Please check 2.bmp in your file, Thank you!" << endl;
   return 0;
}
```

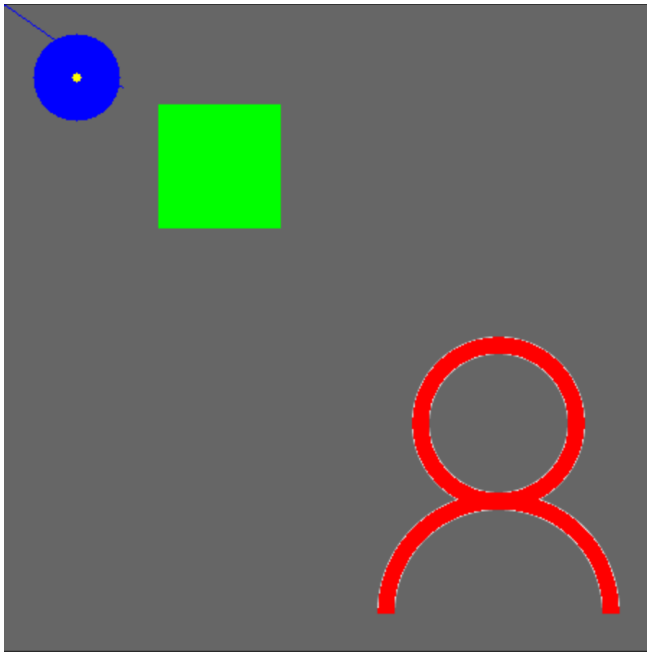
用户可以选择[0]使用 CImg 库的方法进行绘制，也可以选择[1]不使用 CImg 库，结果保存为 2.bmp。每次运行首先会 display 原图 1.bmp，最后 display 结果 2.bmp。

四、结果分析

使用 CImg 库：



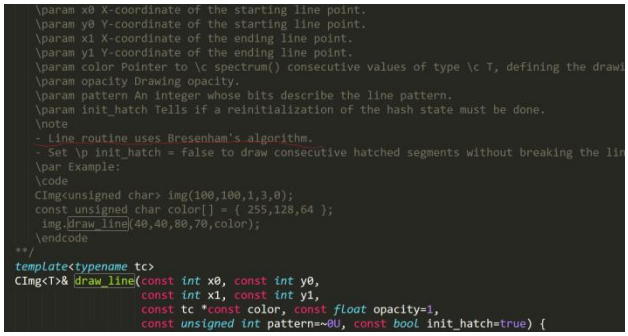
不使用 CImg 库：



分析：先对比圆形。使用 CImg 库的 draw_circle()方法画出来的圆更加光滑，尤其体现在黄色的小圆上。由于使用圆方程计算的时候会有精度丢失的问题，因此手动画的圆会比较粗糙。

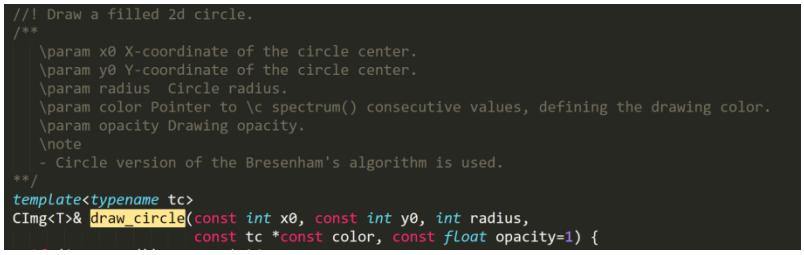
再对比直线。直线的光滑程度大致一样。但在终点处就有所不同了。使用 CImg 库 draw_line()方法画的直线在终点处比较光滑，而且很清晰，而手动实现的直线在终点处比较模糊，有很多重叠的点，显得不够清晰。

然后来看看源码。在 draw_line 函数的介绍中，源码是这样写的：



说明 CImg 库中画直线使用的是精准度较高的 Bresenham 算法，这是对 DDA 算法的改进，所以效果会更好。

而在 draw_circle 函数中，我们同样看到：



这说明 CImg 库画圆也是基于 Bresenham 算法的。

五、难点及总结

首先回答课后的思考题：为什么第四步绘制的圆形区域形状效果不好？原因是：第四步的圆半径比较小，所占的像素点太少，以至于边缘会有不光滑的情况出现。半径增大后效果会比较好。

这一次实验主要熟悉了 CImg 库的一些基本操作，也学会了如何对于一个 bmp 图像进行像素操作。总体而言没有特别难的地方，但是在研究如何把直线画的更光滑上花费了一定的时间，也初步了解了 DDA 算法和 Bresenham 算法，收获不少。希望在接下来的课程能学到更多与图像处理有关的知识。