PA0报告

吴佳启 2022010869

代码逻辑

线段绘制

- 这里我们采用Bresenham算法进行线段绘制，算法的基本思想是通过与中点的位置关系每次判断下一个像素点的位置。
- 根据k和0,1的关系，我们将k分为四种情况进行讨论，这里从k为正且k<1的情况开始。
- 构造一个函数e，这里e指的是，当前坐标为(x,y)下一个像素点的纵坐标是否达到了$y+0.5$，因此x每增加1，e就会增加k，y每增加1，e减少1，e一旦大于0，那么下一个像素点就为$(x+1,y+1)$，否则为$(x+1,y)$。
- 为了避免小数的计算，同时考虑到我们只需要在意e的正负，因此可以每次给e增加$2dy$和减少$2dx$来刻画，同时把e的初始值设为$-dx$即可。

```
for (int i = 0; i <= dx; i++) {
    img.SetPixel(x, y, color);
    x++;
    e += 2 * dy;
    if (e > 0) {
        y++;
        e -= 2 * dx;
```

```
    }
}
```

圆绘制

- 考虑到圆的对称性，我们只需要绘制圆的八分之一即可，然后将各个对称点计算出来。

```
void circlePoints(int x, int y, Image& img) {
    img.SetPixel(x, y, color);
    img.SetPixel(2 * cx - x, y, color);
    img.SetPixel(x, 2 * cy - y, color);
    img.SetPixel(2 * cx - x, 2 * cy - y, color);
    img.SetPixel(y - cy + cx, x - cx + cy, color);
    img.SetPixel(2 * cx - (y - cy + cx), x - cx + cy, color);
    img.SetPixel(y - cy + cx, 2 * cy - (x - cx + cy), color);
    img.SetPixel(2 * cx - (y - cy + cx), 2 * cy - (x - cx + cy), color);
}
```

- 至于一个八分之一圆周，我们从$(cx, cy + radius)$开始，这里可以类比k大于-1小于0的情况执行Bresenham算法，每次对x加一，再通过中点的关系考虑y是否加一即可。
- 这里有$F(x,y) = (x - cx)^2 + (y - cy)^2 - radius^2$，我们要做的则是代入$(x + 1, y - 0.5)$进入表达式计算，根据d和0的关系判断下一次的判断。

```
    int x = cx, y = cy + radius;
    float d = 1.25 - radius;
    circlePoints(x, y, img);
    while ((x - cx) <= (y - cy)) {
        if (d < 0) {
            d += 2 * (x - cx) + 3;
        }
        else {
            d += 2 * (x - cx) - 2 * (y - cy) + 5;
            y--;
        }
        x++;
        circlePoints(x, y, img);
    }
```

填充算法

- 考虑到目的为填充一个连通区域，最自然的想法是从种子节点出发向各个方向进行试探拓展，用dfs的方式进行即可，但考虑到很深的递归空间，我们考虑进行优化。
- 优化的主要逻辑是，每次填充“条形”区域，获得种子节点后，我们先往左拓展，再往右拓展，把这个区域给涂上颜色后，分别向上和向下搜索可能的情况，即我们从左边界出发往右边界扫描，将类似的“条形”区域全部压入栈。
- 正确性与效率：这样做保证了，对每一个条形区域，都可以把与他连接的条形区域都压入栈，从而保证了对整个连通区域的渲染，再加上实现了自己的栈，同时不是基于像素的搜索，大大提高了效率。

代码参考

在实现过程中只读取了代码和课本内容，并未与其他人交流。

在实现过程中遇到的问题

线段绘制

我在执行Line的draw方法时，首先很自然想到了对k和1的大小关系进行分类，但是我在接下来绘制出的图像中发现只获得了一半的线段，发现是在$k < 0$时未讨论，加上后则完成了绘制任务。

圆绘制

这里主要的坑点是在八个对称点的计算过程中，需要注意的是，这里的对称中心不再是我们熟悉的$(0, 0)$，而是圆心(cx, cy),因此我们在计算点(x, y)对折45度的时候，首先要注意不是直接将x和y坐标进行对换，其次是不是对于$y = x$这条直线进行对称，而是圆心，因而对称后点的坐标应为$(y - cy + cx, x - cx + cy)$，接下来其余点则分别关于$x = cx$和$y = cy$进行对称即可得到对应的八个点。

填充算法

- 首先是往左往右扫描的过程中注意不要越界，我在第一次虽然注意到了这个问题，但是在实现时将边界条件的判断放在了后面，仍然触发了越界的错误，更改后解决问题。

```
    while (x < img.Width() && img.GetPixel(x, y) == oldColor) { // 向右填充
        img.SetPixel(x, y, color);
        x++;
    }
```

- 其次是扫描情况的判断，我们的依据是如果扫描得到的颜色不为原来的颜色，则进行扩展，我在实现时却使用的是不为填充的颜色就进行扩展的算法，导致最后的图像与要求相差很大，因为在这种情况下，比如要用红色涂掉蓝色的区域，可能也会覆盖掉一些绿色、黄色等，因此正确的方法是先从种子点获取原来的颜色后再进行填充算法。