三、种子填充算法原理和实践

1、原理

1）区域填充概述

区域是指光栅网格上的一组像素，而区域填充就是把某确定的像素值填充到区域内部的所有像素中。
区域填充方法分为两大类：
第一类方法中，区域由多边形围成，区域由多边形的顶点序列来定义。
第二类方法是通过像素的值来定义区域的内部，相应的技术是以像素为基础的，本文所讲的种子填充算法就是采用的这种方法。

2）定义区域

以像素为基础的区域填充主要依据区域的连通性进行填充，可以使用两种方式来定义区域：
① 内定义区域：定义方法是指出区域内部原有的像素值 **oldvalue**，此时光栅网格上连通的且像素值为 oldvalue 的一组像素称为一个区域。
② 边界定义区域：定义方法是指出区域边界所具有的像素值 **boundaryvalue**，区域的边界应该是封闭的，并且应该指明区域的内部。

3）四连通和八连通

四连通：从区域中的一个象素出发，经连续地向上下左右四个相邻象素的移动，就可以到达区域内的任意另一个象素。
八连通：如果除了要经上下左右的移动，还要经左上、右上、左下和右下的移动，才能由一个象素走到区域中另外任意一个象素。

4）定义种子

利用区域的连通性进行区域填充，除了需要区域应该明确定义外，还需要事先给定一个区域内部象素，这个象素称为种子。
做区域填充时，要进行对光栅网格的遍历，找出由种子出发能达到而又不穿过边界的所有象素。如果使用递归进行遍历的话，只要区域大一点就会出现栈溢出的错误，因此应该采用迭代的方式进行遍历。

5）优缺点分析

优点：不受区域不规则性的影响。
缺点：①需要事先知道一个内部象素种子；②填充区域必须是连通的；③如果有多个区域要进行填充时区域之间不能重叠；④因为要进行遍历，而且每次遍历都需要进行判断，所以效率很慢。

2、实践

1）题目要求

因为该算法有较大缺陷，所以此次实践只是简单验证种子填充算法即可，更为详细的区域填充实践在“四、多边形的扫描转换算法原理和实践”中进行。将一个 100 * 100 的矩形区域填充为红色，要求矩形边界使用绿色。

2）分析

首先使用 DDA 直线扫描转换算法绘制一个矩形区域，然后使用边界定义区域的种子填充算法进行填充（使用区域内原像素定义区域的方法和边界定义区域方法的实现十分类似）。

3）代码实现

（1）种子填充算法实现（边界定义区域）：

```
//(x, y)表示种子，boundaryvalue 表示边界颜色，color 表示填充颜色
void CTestView::Boundaryfill(CDC * pDC, int x, int y, COLORREF boundaryvalue, COLORREF color)
{
    /*
    //递归调用会造成栈溢出
    COLORREF oldColor = pDC->GetPixel(p);
    if (oldColor != boundaryvalue && oldColor != color) //  未达边界且未访问过
    {
        pDC->SetPixel(x, y, color);

        Boundaryfill(pDC, x, y - 1, boundaryvalue, color);   //向四个方向扩散。
        Boundaryfill(pDC, x, y + 1, boundaryvalue, color);
        Boundaryfill(pDC, x - 1, y, boundaryvalue, color);
        Boundaryfill(pDC, x + 1, y, boundaryvalue, color);
    }*/

    //将递归改为迭代
```

```
    CPoint p;
    vector<CPoint> stack;
    stack.push_back(CPoint(x, y)); //种子入栈
    while (!stack.empty())
    {
        p = stack.back(); //栈顶象素出栈
        stack.pop_back();

        COLORREF oldColor = pDC->GetPixel(p);
        if (oldColor != boundaryvalue && oldColor != color) //  未达边界且未访问过
        {
            pDC->SetPixel(p.x, p.y, color);

            stack.push_back(CPoint(p.x, p.y + 1));    //四连通
            stack.push_back(CPoint(p.x, p.y - 1));
            stack.push_back(CPoint(p.x + 1, p.y));
            stack.push_back(CPoint(p.x - 1, p.y));
        }
    }
}
```

（2）实现 OnDraw 函数：

```
void CTestView::OnDraw(CDC* pDC)
{
    CTestDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    //绘制矩形
    DDALine(pDC, 10, 10, 110, 10, RGB(0, 255, 0));
    DDALine(pDC, 10, 10, 10, 110, RGB(0, 255, 0));
    DDALine(pDC, 110, 110, 110, 10, RGB(0, 255, 0));
    DDALine(pDC, 110, 110, 10, 110, RGB(0, 255, 0));
    //区域填充
    Boundaryfill(pDC, 20, 20, RGB(0, 255, 0), RGB(255, 0, 0));
}
```

4）效果