> 本补充材料可以帮助你理解有效边表算法，以及可能涉及到的部分js语法知识。数据结构部分使用了大家比较熟悉的C++语法，后面给出了本算法的js伪代码，最后是一些js编程的提示，大家各取所需。

1 有效边表多边形填充算法

- 基本思想
  - 要对多边形进行填充，简单直接的一种思路是X-扫描线算法，循环求交、排序、交点配对和区域填充的过程；但每次扫描都需要和所有边求交，而显然不是每一条边都和当前扫描线相交，有效边表的优化主要在求交。
- 数据结构
  - 点

```
struct Point{
    int x, y; // 分别存储x和y的坐标
};
```

  - 边
    - 为方便理解，y值小的点我称为“矮”，反之称为“高”

```
// 链表的方式
struct Edge{
    float x; // 边的起始x值，指较“矮”的点的x值，用float是因为需要加浮点数m，方便存储
    int ymax; // 边的结束y值，指较“高”的点的y值
    float m; // 斜率的倒数，方便计算，扫描线每提高一格，意味着y值增加1，对应的x值则增加m
    Edge* next; // 链表结构，指向下一条边
    // 本应该存有一个ymin值，但ymin被作为数组索引了，无需额外存储
};
Edge* ET[200];

Edge* edge1 = new Edge(); // 新建一条边，假设这条边的起始y是1
edge1->x = ...;
edge1->ymax = ...;
edge1->m = ...;
ET[1] = edge1; // 起始y为1，第一条边赋值为edge1

Edge* edge2 = new Edge(); // 新建另一条边，假设这条边的起始y也是1
edge2->x = ...;
edge2->ymax = ...;
edge2->m = ...;
ET[1]->next = edge2; // 起始y为1，第二条边赋值为edge2
```

    > 注意，本算法里我们不用考虑浮点数的优化

  - 活动边
    - 和边的数据结构相同
- 算法思路
  - 算法的核心围绕着活动边表展开

1. 初始化边表
   - 在开始填充之前，我们需要准备好一份边表
   - 这个边表包含了多边形所有边的信息
2. 使用活动边表，进行求交、排序、交点配对和区域填充
   - 首先找到扫描线起始y值YMIN和结束y值YMAX
   - 初始化AET表为空
   - 从y=YMIN开始，循环以下步骤，直到y==YMAX
     - 遍历AET表，如果边的ymax==y，将这条边移除（删除无用边，这条边已经不会在和后面的扫描线有交点了）。对于其他边，进行求交，即将x的值加上m（y值增加了1，x值增加m）。如果ET表中有ymin==y的边，将这些边加入AET表
     - 排序。将交点（AET表每个边就对应了一个交点，边的x值是交点的x坐标，扫描线y值就是交点的y坐标）按x从小到大排序，如果x值相同，则按m从小到大排序
     - 交点配对和区域填充。遍历AET表的每个x值，两两组成一个交点区间，对每个区间内的像素进行填充。
- 基于js的伪代码

```
function drawPolygon(points){
    // 建立边表
    var ET = [];
    for(points中两两相邻的点){
        var p1 = 两点中较矮的点;
        var p2 = 两点中较高的点;
        var edge = {
            x: p1的x值,
            ymax: p2的y值,
            m: Δx/Δy
        };
        var ymin = p1的y值;
        添加edge到ET[ymin]中（注意ET[ymin]ss数组的初始化，关于嵌套数组，后面的js提示有讲）
    }
    minY = 点中最小y值;
    maxY = 点中最大y值;

    // 活动边表的求交、排序、配对和填充
    var AET = [];
    for(y从minY到maxY){
        // 删除与求交
        for (AET中每个边){
            if (当前边ymax>y){
                删除当前边
            }
            else{
                当前边x的值加上m
            }
        }
        // 添加新边
        if (ET表中有ymin=y的，即ET[ymin]不为空){
            添加这些边到AET表
        }
        // 排序
        对AET表进行排序，x从小到大，x相同则m从小到大
        // 配对和填充
        for(AET中两两一对的边){
            var x1 = 点1的x值;
            var x2 = 点2的x值;
            填充(x1, y)到(x2, y)
        }
    }
}
```

2 js语法提示

- 关于结构体
  - js是面向对象语言，且类的概念被淡化了，你不需要使用结构体来构造数据结构，直接使用对象构造方法即可，如

```
struct Edge{
    float x;
    int ymax;
    float m;
};
struct Edge edge1 = {1, 10, 0.5};
```

在js中可以直接

```
var edge1 = {
    x: 1,
    ymax: 10,
    float: 0.5
}
```

- 关于链表
  - js没有指针的概念，链表结构在js中可以用数组替代，如

```
struct Node{
    int val;
    Node* next;
}
Node* root = new Node();
root->val = 1;
Node* leaf1 = new Node();
leaf->val = 2;
Node* leaf2 = new Node();
leaf->val = 3;

root->next = leaf1; // 指定下一个元素为leaf1
root->next = nullptr; // 修改下一个元素为空
root->next = leaf2; // 修改下一个元素为leaf2
```

可以使用js动态数组的特性实现

```
var list = [];
var root = 1;
var leaf1 = 2;
var leaf2 = 3;

list.push(root); // [1]
list.push(leaf1); // [1, 2]
list.pop(); // [1]
list.push(leaf2); // [1, 3]
```

- 关于多维数组
  - js没有多维数组，但你可以使用数组的嵌套来实现，如

```
int array[10][10];
array[0][0] = 0;
array[9][9] = 18;
```

js可以这么实现

```
var array = []; // 新建一个空数组
array[0][0]; // 错误！ array数组为空，访问其array[0]返回undefined，接着访问undefined的[0]会报错
array[0] = []; // 给array[0]赋值，值是一个空数组
array[0][0]; // undefined，array[0]数组为空，访问其[0]返回undefined
array[0][0] = 1; // 赋值
array[0][0]; // 1
```

- 关于vec2
  - 如上面所说，js类的概念很淡，vec2也不是类名，vec2本质是一个函数，返回一个数组，这个数组包含两个元素，你简单可以把源码理解成

```
function vec2(x, y){
    return [x, y];
}
// test
var x = vec2(1, 2); // [1, 2]
```

使用vec2函数是为了让改变量的用处更加清晰，vec2返回值的使用和一般数组没有区别

- 关于js调试
  - 写代码一定是需要调试的，调试能大大提升debug效率，各浏览器也提供了好用的debug工具
  - 一般来说，按下F12可以打开调试台，其中console栏用以查看程序报错、输出记录等，source栏用于添加断点、观察变量、观察函数调用情况等。但各浏览器调试台存在差异，具体用法可以按照自己的浏览器百度一下~

完

2020.11.1