CG-PJ2—说明文档

开发运行环境

- **硬件**：带有支持OpenGL ES2.0显卡的计算机
- **软件**：支持HTML5, JavaScript和WebGL的浏览器，建议选择最新版本的Google Chrome浏览器，也可以选择Internet Explorer 11+或者最新版本的Firefox，Safari等。

运行使用方法

参考project2文档

代码结构

- line.js 定义了一个LineShader类，用于三角形的边框绘制
- triangle.js 定义了一个TriangleShader类，用于三角形的涂色
- transfer.js 定义了一个转换类，提供了canvas坐标向webgl坐标的转换方法
- main.js 主脚本，用于显示与交互

line.js

```
//该类用于三角形边框的渲染
class LineShader {
    //绑定绘制工具gl
    constructor(gl)
    //定义VSHADER_SOURCE与FSHDER_SOURCE，初始化shader
    initial()
    //根据vertex初始化buffer，利用currentAngle设置旋转角，绘制图像
    draw(vertex, currentAngle)
    //清理gl
    clear()
}
```

triangle.js

```
//该类用于三角形内部渲染
class LineShader {
    //绑定绘制工具gl
    constructor(gl)
    //定义VSHADER_SOURCE与FSHDER_SOURCE，初始化shader
    initial()
    //根据vertex初始化buffer，利用currentAngle设置旋转角，绘制图像
    draw(vertex, currentAngle)
    //清理gl
    clear()
}
```

transfer.js

```
//该类提供了canvas到webgl的转换方法，包括颜色转换与坐标转换
class Transfer {
    //颜色转换，如[255,0,0]->[1,0,0]
    static color_transfer(color)
    //坐标转换，如[350,100,0]->[(350*2/width-width)*scale, (100*2/height-
height)*scale]
    static vertex_transfer(vertex, width, height, scale)
    //涂色三角形时，转换坐标（需要转换颜色）
    static triangle_transfer(vertex_list, color_list, width, height, scale)
    //描绘三角形边框时，转换坐标（不需要转换颜色）
    static triangle_line_transfer(vertex_list, width, height, scale)
}
```

main.js

```
/* 三个特殊的状态变量 */
//用于记录最近操作过的三角形序号，决定叠层的次序
let LRUCache = [0, 1, 2, 3, 4, 5, 6, 7];
//上一帧旋转的时间，用于计算旋转角度，要注意旋转刚刚开始时不能直接直接使用该值
let g_last = Date.now();
//renderInAnimation记录是否是旋转动画的第一帧
let renderInAnimation = false;


/* 绑定事件 */
//绑定鼠标事件与键盘事件
function initialEvent()


/* 三角形初始化 */
//将config.polygon中的4个矩形转化为8个三角形
function initialTriangleVertice()
```

```
/* 涂色与绘制 */
//涂色单个三角形
function paintSingleTriangle(triangleVertex, currentAngle, triangleShader)
//涂色所有8个三角形
function paintAllTriangles(triangleVertices, currentAngle, LRUCache)
//绘制单个三角形边框
function drawSingleTriangleLine(triangleVertex, currentAngle, lineShader)
//绘制所有三角形边框
function drawAllTriangleLines(triangleVertice, currentAngle, LRUCache)


/* 渲染与动画 */
//渲染
function render()
//帧动画(响应帧动画切换事件的函数)
function animation()
//根据时间动画时间差进行图像旋转
function rotate(angle)


/* canvas的清除 */
//清除整个canvas
function clearCanvas()


/* 工具函数 */
//工具函数，返回离x,y在tolerance范围内的顶点序号
function getSelectedVertexIndex(x, y, triangleVertice, tolerance)
//工具函数，判断value是否在array里面
function isInArray(value, array)
//工具函数，将arr中值为val的第一个元素去除
function removeByValue(arr, val)
```

额外的功能

使用LRU机制决定层叠的次序。最近一次拖动的顶点所关联的三角形在最后层叠，能覆盖之前层叠的三角形。

遇到的问题

第一次写的时候，是把requestAnimationFrame直接写在渲染函数render里的，如下

```
//渲染
function render() {
    clearCanvas();
    ...//与绘制相关的函数

    if (animationStatus) { //处于动画状态
        requestAnimationFrame(render);
    }
}
```

每次需要重新渲染的时候 调用render，,这个函数将会通过你的当前animationStatus的状态判断是否进行动画。一眼望去没有什么不妥，但是会出现一个问题。当你在动画进行过程中，按下B键进行边框显示状态的切换时，不可避免会调用render重新绘制边框，但是当前你的animationStatus状态为true，这时就相当于又调用了能进行动画的render，此时页面会出现明显的卡顿，帧数大大降低。解决办法就是将渲染与动画分离，动画仅仅应对动画事件的触发，而渲染应对其他事件（如鼠标拖动编辑，边框显示与隐藏等），而动画函数调用渲染函数即可。如下所示

```
//渲染
function render() {
    clearCanvas();
    ...//与绘制相关的函数
}
//帧动画(响应帧动画切换事件的函数)
function animation() {
    if (animationStatus) {
        if (!renderInAnimation) {
            g_last = Date.now();
            renderInAnimation = true;
        }
        currentAngle = rotate(currentAngle);
        render();
        requestAnimationFrame(animation);
    }
}
```

看sample中的源代码发现他的动画是用一个叫tick的函数实现的，看着tick函数的实现方法我写了一个animation函数，用于帧动画的实现。

建议

project2知识性比较强，虽然比较烦但还是挺有趣的，特别是当遇到一些很奇怪的bug比如上述的例子，深究下去会学到很多东西，针对project的建议的话，我觉得可以将3个这样的pj减少为1个或2个，增加一个大的PJ。