目录

引言----------2
设计概述----------2
任务和目标----------2
运行环境----------3
开发环境----------3
系统详细设计----------4
系统流程图----------4
系统模块----------4
模块详细设计----------5
用户首页模块: ----------5
场景渲染模块----------5
场景控制模块----------6
场景切换模块----------6
场景动画模块----------8
纹理混合模块----------9

引言

系统课题——VR 看房。

系统类型——基于 Three.js 的实现不同 3D 场景间的渲染、漫游、切换。

系统项目组——校企合作 VR 看房项目组：谢岳、宋沆群、韩晶、李心丽、张磊。

VR（Virtual Reality）——虚拟现实，虚拟现实技术是一种能够创建和体验虚拟世界的计算机仿真技术，它利用计算机生成一种交互式的三维动态视景，其实体行为的仿真系统能够使用户沉浸到该环境中，给用户提供包括视觉、听觉、嗅觉、触觉等感官上模拟真实的感受。

简单来说，VR 看房是通过 VR 虚拟现实技术与 3D 全景展示技术让用户在线全方位了解房屋的户型和细节，并利用场景贴图、射线拾取，以及着色器对贴图纹理混合，实现不同场景之间切换的自然过渡。

设计概述

任务和目标

1) 用户主页
2) 3D 场景渲染
3) 3D 场景控制和事件
4) 3D 场景切换和自然过渡

运行环境

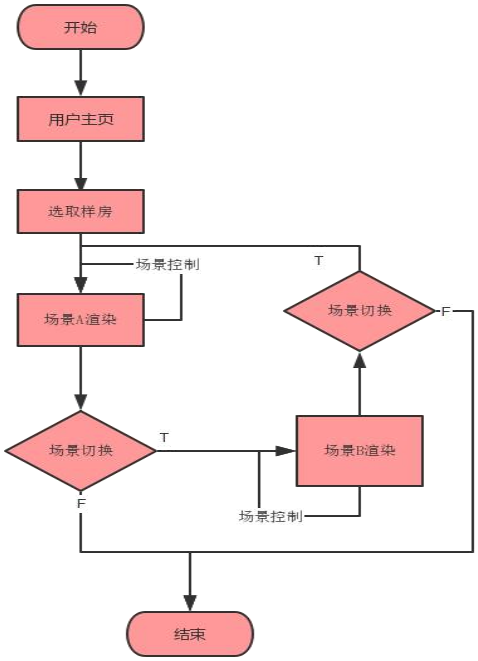
环境	名称	版本	说明
PC	Chrome	51 版起	可支持 97%的 ES6 新特性
PC	Firefox	53 版起	可支持 97%的 ES6 新特性
PC	Safari	10 版起	可支持 99%的 ES6 新特性
PC	Edge	15	可支持 96%的 ES6 新特性
PC	IE	7~11	不支持
移动端	IOS	10.0 版起	可支持 99%的 ES6 新特性
移动端	Android	5.1 仅支持 25%	基本不支持 ES6 新特性
服务器	Node.js	6.5 版起	可支持 97%的 ES6 新特性

开发环境

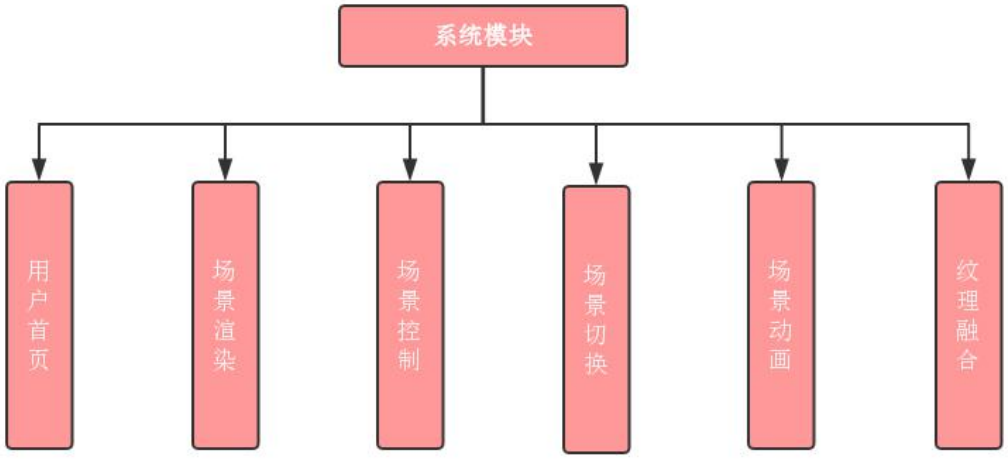
软件	名称	版本	说明
开发工具	IntelliJ IDEA	2020.3	
	VS Code	1.56	
服务器端环境	Nodejs	12.9.0	
前端语言	ECMAScript	6	
	HTML5		
前端开发框架	Vue	2.6.10	
	element	2.13.2	PC 端 UI 组件库
包管理工具	npm	6.10.2	
代码管理	git	2.24.1	

系统详细设计

系统流程图



系统模块



模块详细设计

用户首页模块：

功能：

展示所有房产信息

场景渲染模块

功能：

渲染不同场景

实现：

使用 Three.js 渲染一个场景，至少需要以下几个步骤：

1) 创建一个容纳三维空间的场景 — Sence
2) 将需要绘制的元素加入到场景中，对元素的形状、材料、阴影等进行设置
3) 给定一个观察场景的位置，以及观察角度，我们用相机对象（ Camera）来控制
4) 将绘制好的元素使用渲染器（ Renderer）进行渲染，最终呈现在浏览器上

本系统中，准备好一个立方体盒子，然后拍好六张全景图的图片作为材质贴在立方体的内壁上。

相机（Camera）设定一个观察点，在立方体的中心，通过场景控制模块进行 720° 观察。

场景控制模块

功能：

切换视角，对 Threejs 的三维场景进行缩放、平移、旋转操作。

实现：

基于 Three.js 的轨道控制器 OrbitControls 实现，本质上改变的并不是场景，而是相机的参数，相机的位置角度不同，同一个场景的渲染效果是不一样，比如一个相机绕着一个场景旋转，就像场景旋转一样。

```
//用户交互插件  鼠标左键按住旋转，右键按住平移，滚轮缩放
    var controls = new THREE.OrbitControls( camera, renderer.domElement );
        // 如果使用 animate 方法时，将此函数删除
        //controls.addEventListener( 'change', render );
        // 使动画循环使用时阻尼或自转 意思是否有惯性
        controls.enableDamping = true;
        //动态阻尼系数 就是鼠标拖拽旋转灵敏度
        //controls.dampingFactor = 0.25;
        //是否可以缩放
        controls.enableZoom = true;
        //是否自动旋转
        controls.autoRotate = true;
        //设置相机距离原点的最远距离
        controls.minDistance    = 1;
        //设置相机距离原点的最远距离
        controls.maxDistance    = 200;
        //是否开启右键拖拽
        controls.enablePan = true;
```

场景切换模块

功能：

添加切换事件、实现场景切换。

实现：

点击事件逻辑：

我们无法给一个三维物体添加事件绑定，这里我们需要借助一个新的概念就是碰撞检测，由我们的摄像机的角度出发，发射一条射线，也就是鼠标，当我们的鼠标控制的光线碰

撞到这个物体时，会返回一个数组，我们可以对返回的数组做一个判空操作，如果不为空则说明我们点击了这个物体，然后就触发事件。

three.js 提供了一个类 THREE.Raycaster 可以用来解决这个问题。

THREE.Raycaster 对象从屏幕上的点击位置向场景中发射一束光线。

Raycaster(origin, direction, near, far)

origin — 射线的起点向量。
direction — 射线的方向向量，应该归一标准化。
near — 所有返回的结果应该比 near 远。Near 不能为负，默认值为 0。
far — 所有返回的结果应该比 far 近。Far 不能小于 near，默认值为无穷大。

setFromCamera(coords : Vector2, camera : Camera) : null
coords - 鼠标的二维坐标，在归一化的设备坐标(NDC)中，也就是 X 和 Y 分量应该介于 -1 和 1 之间。
camera - 射线起点处的相机，即把射线起点设置在该相机位置处。

intersectObject (object, recursive : Boolean, optionalTarget : Array) : Array

object - 检测与射线相交的物体
recursive- 若为 true 则检查后代对象，默认值为 false
optionalTarget - （可选参数）用来设置方法返回的设置结果。若不设置则返回一个实例化的数组。如果设置，必须在每次调用之前清除这个数组（例如，array.length= 0;)

返回值 Array
[{ distance, point, face, faceIndex, object }, …]
distance - 射线的起点到相交点的距离
point - 在世界坐标中的交叉点
face -相交的面
faceIndex - 相交的面的索引
object - 相交的对象
uv - 交点的二维坐标

场景切换逻辑：

在最初，尝试用两个紧挨着的盒子，分别加载好贴图，当点击的时候，让摄像机进行移动，移动到第二个盒子的中间，但是造成的结果是，在穿过盒子的时候会使得贴图变形并且给用户的感受十分不好，因此又尝试移动到第一个盒子边缘时进行跳变直接使得摄像机调到相应的位置，但是给人感觉都不是很好。

本系统中在同一个盒子中，切换 2 套纹理，每套纹理共 6 个面，通过后续的场景动画补

间和纹理混合，借助自定义着色器中的 mix 函数，通过动态的修改其中的混合比例，实现纹理的渐变。

场景动画模块

功能：

控制两种纹理的混合比例，实现纹理的补间动画

实现：

tween.js 允许你以平滑的方式修改元素的属性值。你只需要告诉 tween 你想修改什么值，以及动画结束时它的最终值是什么，动画花费多少时间等信息，tween 引擎就可以计算从开始动画点到结束动画点之间值，来产生平滑的动画效果。

本系统中，设场景 A 的纹理贴图比例为 Pa，设场景 B 的纹理贴图比例为 Pb，任意时刻 t 恒有 Pa+Pb = 1。

渲染场景 A：Pa=1，Pb=0。

渲染场景 A：Pa=0，Pb=1。

利用 tween.js 在时间段 T 内按照某种函数规则，实现任意时刻 t 的比例变化，再传给纹理混合模块进行进一步处理。

Linear 线性匀速运动效果
Quadratic 二次方的缓动（t^2）
Cubic 三次方的缓动（）
Quartic 四次方的缓动（）
Quintic 五次方的缓动
Sinusoidal 正弦曲线的缓动（）
Exponential 指数曲线的缓动（）
Circular 圆形曲线的缓动（）
Elastic 指数衰减的正弦曲线缓动（）
Back超过范围的三次方的缓动
Bounce 指数衰减的反弹缓动

In 加速，先慢后快
Out 减速，先快后慢
InOut 前半段加速，后半段减速

纹理混合模块

功能：

接收 2 套纹理的比例，编写自定义着色器，对点进行纹理处理。

实现：

重写顶点着色器和片元着色器。

着色器是，材质的一种，和其他材质不同之处在于，其它材质的属性和方法都是封装好的，而自定义着色器我们可以自定以方法以及属性，但需要一些 GLSL 的基础。

```
<!-- 顶点着色器 -->
<script id="vertexShader" type="x-shader/x-vertex">
        // attribute vec3 position;
        // attribute vec3 color;
        // 系统自动声明顶点纹理坐标变量 uv
        // attribute vec2 uv;
        // varying 关键字声明一个变量表示顶点纹理坐标插值后的结果
        varying vec2 vUv;
        void main(){
            // 顶点纹理坐标 uv 数据进行插值计算
            vUv = uv;
            // 投影矩阵 projectionMatrix、视图矩阵 viewMatrix、模型矩阵 modelMatrix，这些矩阵控制着模型的旋转，放大缩小，位置变换等等，都是通过矩阵运算出来的
            gl_Position = projectionMatrix*viewMatrix*modelMatrix*vec4( position, 1.0 );
        }
    </script>
<!-- 片元着色器 -->
<script id="fragmentShader" type="x-shader/x-fragment">
        // 声明一个纹理对象变量
        uniform sampler2D texture1;
        uniform sampler2D texture2;
        uniform float time;//纹理的比例
        // 顶点片元化后有多少个片元就有多少个纹理坐标数据 vUv
        varying vec2 vUv;
        void main() {
            //内置函数 texture2D 通过纹理坐标 vUv 获得贴图 texture 的像素值
            gl_FragColor = mix(texture2D( texture1, vUv ),texture2D( texture2, vUv ), time);
         }
    </script>
```