**threejs實現簡單全景看房demo😜**

[threejs實現簡單全景看房demo😜 - 掘金 (juejin.cn)](https://juejin.cn/post/7047709128600322056)

各位大家好😊，最近一直在學習 **threejs** ，在學習過程中不斷進步，在將來我會不斷完善我的 **threejs** 案例庫，希望能在學習路上幫到大家 🌹

接下來為各位介紹的是一個全景看房的demo，我們先上位址：

* 源碼位址：[github.com/ljnMeow/360...](https://link.juejin.cn/?target=https%3A%2F%2Fgithub.com%2FljnMeow%2F360-house-viewing.git)
* 預覽位址：[ljnmeow.github.io/360-house-v...](https://link.juejin.cn/?target=https%3A%2F%2Fljnmeow.github.io%2F360-house-viewing%2Fdist" \o "https://ljnmeow.github.io/360-house-viewing/dist" \t "_blank)
* 全景圖切割工具：[matheowis.github.io/HDRI-to-Cub...](https://link.juejin.cn/?target=https%3A%2F%2Fmatheowis.github.io%2FHDRI-to-CubeMap%2F" \o "https://matheowis.github.io/HDRI-to-CubeMap/" \t "_blank)

**前言**

**threejs** 是 **javascript** 編寫的一個 **WebGL** 第三方庫。 在 **threejs** 中，我們通過 **Scene（場景）** 、 **Camera（相機）** 和 **Renderer（渲染器）** 來實現一個 3d 的場景，然後往裡面添加各種光源、物體等等，形成一個 3d 世界。

**threejs支援程度**

因為 **threejs** 是基於 **webgl** 寫的，所以我們主要看設備是否支援 **webgl** 。 據我所知目前主流最新版本的設備、瀏覽器都支援**webgl**，但如果客服要在一些早期的設備上跑，例如某些超大屏大屏監控系統，跑這個的設備可能是早期的 **IE** 並且無法升級。 我們還是需要權衡下早期 **IE** 和 安卓、蘋果設備是否支援 **webgl**， 詳情可見下圖：

**基本介紹**

要實現3d圖形的展示，大致是以下思路：

* 建立三維場景（Scene）
* 選擇視覺點並確認視覺位置角度（Camera）
* 添加物體供觀察
* 渲染場景（Renderer）

**現場**

場景是我們所有物體的容器，通俗來講就相當於我們的世界

**照相機**

相機是我們這個世界的觀察者，使用右手座標系定位

**threejs** 常用的相機有兩種，分別是 **正交投影相機（OrthographicCamera）** 和 **透視投影相機（PerspectiveCamera）** ，下面是我對這兩種方式的理解，錯了請各位大佬指點我一下 🤞

**正交投影相機（OrthographicCamera）**

正交投影是一種相對簡單的投影模式，我們可以把該模式看作看作一個長方形矩形，總共有6個面。 該模式的特點是無論出於矩形中的哪個位置，其投影的大小都是一樣的，通俗理解就是該模式跟常生活中的視覺效果不一樣。 我們平時看物體是遠小近大，而該模式下展示的物體無論物體距離相機距離遠或者近，在最終渲染的圖片中物體的大小都保持不變。 通過下面圖片能清楚看到，相機視覺中的物體投放到進平面，然後近平面中的物體渲染到遠平面也就是螢幕中，並且無論投放在哪一面，物體的大小都不變。

**透视投影相机(PerspectiveCamera)**

透视投影的视觉效果呈现一个锥形，该模式就是我们现实世界中的视觉体现。该模式的特点是物体远小近大。

相機效果體驗位址在官方demo中有：[threejs.org/examples/#w...](https://link.juejin.cn/?target=https%3A%2F%2Fthreejs.org%2Fexamples%2F%23webgl_camera" \o "https://threejs.org/examples/#webgl_camera" \t "_blank)

**渲染**

渲染器渲染我們精心製作的場景

**進入正題**

根據我的瞭解目前常用於實現全景看房效果的有兩種，分別是 **天空盒（skyBox）** 和 **全景圖片貼圖** ，我這個demo使用的是後者。

**實現方式**

**天空盒**

**skyBox** 方法是最容易理解的，在我們身處的場景內，無非就是6個面，上下、前後、左右。 將這6個面的視覺處理成圖片就得到6張不同方向視覺的圖片。

我們將6個視覺的圖片貼到立方體的6個面，可以得到一個房間。

initContent() {

let picList = ["left", "right", "top", "bottom", "front", "back"];

let boxGeometry = new THREE.BoxGeometry(10, 10, 10);

let boxMaterials = [];

picList.forEach((item) => {

let texture = new THREE.TextureLoader().load(

require(`@/assets/image/${item}.png`)

);

boxMaterials.push(new THREE.MeshBasicMaterial({ map: texture }));

});

this.box = new THREE.Mesh(boxGeometry, boxMaterials);

this.scene.add(this.box);

},

將視覺移到立方體中心，並讓貼圖內翻轉一下，就能實現全景看房

this.box.geometry.scale(10, 10, -10);

**全景圖貼圖**

全景圖貼圖這種方式我認為是簡單而且效果最好的一種。 寫之前需要一張全景圖片，這個用單反的全景模式就能拍一張

**threejs** 添加一個球體。 並把全景圖作為貼圖貼到球體上，得到的效果如下：

initContent() {

let sphereGeometry = new THREE.SphereGeometry(16, 50, 50);

let texture = new THREE.TextureLoader().load(require("@/assets/image/livingRoom.jpg"));

let sphereMaterial = new THREE.MeshBasicMaterial({ map: texture });

this.sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);

this.scene.add(this.sphere);

},

同样，把视觉放球内，贴图反转。

sphereGeometry.scale(16, 16, -16);

**添加标签**

我们现在已经实现全景看房了，接下来给一些物体添加说明标签，我这边的标签分两种，一种点击能够进入下一个房间，一种是弹出信息框。先看效果：

下面看看我的数据结构

dataList: [

{

image: require("@/assets/image/livingRoom.jpg"), // 场景贴图

tipsList: [ // 标签数据

{

position: { x: -200, y: -4, z: -147 }, // 标签位置

content: { // 标签内容

title: "进入厨房", // 标题

text: "", // 文本内容

image: 1, // 场景贴图的下标，对应dataList下标

showTip: false, // 是否展示弹出框

showTitle: true, // 是否展示提示标题

},

},

{

position: { x: -100, y: 0, z: -231 },

content: {

title: "信息点2",

text: "77989",

showTip: true,

showTitle: false,

},

},

{

position: { x: 150, y: -50, z: -198 },

content: {

title: "信息点3",

text: "qwdcz",

showTip: true,

showTitle: false,

},

},

{

position: { x: 210, y: 11, z: -140 },

content: {

title: "信息点4",

text: "大豆食心虫侦察十大大苏打大大大大大大大",

showTip: true,

showTitle: false,

},

},

{

position: { x: 208, y: -12, z: 140 },

content: {

title: "信息点5",

text: "eq",

showTip: true,

showTitle: false,

},

},

{

position: { x: 86, y: -9, z: 236 },

content: {

title: "进入房间",

text: "",

showTip: false,

showTitle: true,

},

},

],

},

{

image: require("@/assets/image/kitchen.jpg"),

tipsList: [

{

position: { x: -199, y: -24, z: 145 },

content: {

title: "进入大厅",

text: "",

image: 0,

showTip: false,

showTitle: true,

},

},

],

},

],

往場景中添加標籤，得到如下效果。

addTipsSprite(index = 0) {

let tipTexture = new THREE.TextureLoader().load(

require("@/assets/image/tip.png")

);

let material = new THREE.SpriteMaterial({ map: tipTexture });

this.tipsSpriteList = [];

this.dataList[index].tipsList.forEach((item) => {

let sprite = new THREE.Sprite(material);

sprite.scale.set(10, 10, 10);

sprite.position.set(item.position.x, item.position.y, item.position.z); // 设置标签位置

sprite.content = item.content; // 设置标签内容

this.tipsSpriteList.push(sprite); // 储存标签

this.scene.add(sprite); // 添加到场景中

});

},

接下來實現滑鼠移入，內容彈出效果

// html

<div class="tooltip-box" :style="tooltipPosition" ref="tooltipBox">

<div class="container">

<div class="title">标题：{{ tooltopContent.title }}</div>

<div class="explain">说明：{{ tooltopContent.text }}</div>

</div>

</div>

<p class="title-text" ref="titleBox" :style="titlePosition">

{{ tooltopContent.title }}

</p>

复制代码

// data

tooltipPosition: { // 初始化位置全部在屏幕之外

top: "-100%",

left: "-100%",

},

titlePosition: {

top: "-100%",

left: "-100%",

},

tooltopContent: {}, // 展示的内容

// method

onMousemove(e) {

e.preventDefault();

let element = this.$refs.threeDBox;

let raycaster = new THREE.Raycaster();

let mouse = new THREE.Vector2();

// 通过鼠标点击的位置计算出raycaster所需要的点的位置，以屏幕中心为原点，值的范围为-1到1.

mouse.x = (e.clientX / element.clientWidth) \* 2 - 1;

mouse.y = -(e.clientY / element.clientHeight) \* 2 + 1;

raycaster.setFromCamera(mouse, this.camera);

// 将标签精灵数据放进来做视线交互

let intersects = raycaster.intersectObjects(this.tipsSpriteList, true);

// 视线穿过集合选择最前面的一个

if (intersects.length > 0) {

// 将标签的空间坐标转屏幕坐标，通过计算赋给元素的top、left

let elementWidth = element.clientWidth / 2;

let elementHeight = element.clientHeight / 2;

let worldVector = new THREE.Vector3(

intersects[0].object.position.x,

intersects[0].object.position.y,

intersects[0].object.position.z

);

let position = worldVector.project(this.camera);

this.tooltopContent = intersects[0].object.content;

if (intersects[0].object.content.showTip) {

let left = Math.round(

elementWidth \* position.x +

elementWidth -

this.$refs.tooltipBox.clientWidth / 2

);

let top = Math.round(

-elementHeight \* position.y +

elementHeight -

this.$refs.tooltipBox.clientHeight / 2

);

this.tooltipPosition = {

left: `${left}px`,

top: `${top}px`,

};

} else if (intersects[0].object.content.showTitle) {

let left = Math.round(

elementWidth \* position.x +

elementWidth -

this.$refs.titleBox.clientWidth / 2

);

let top = Math.round(-elementHeight \* position.y + elementHeight);

this.titlePosition = {

left: `${left}px`,

top: `${top}px`,

};

}

} else {

// 鼠标移出去隐藏所有

this.handleTooltipHide(e);

}

},

handleTooltipHide(e) {

e.preventDefault();

this.tooltipPosition = {

top: "-100%",

left: "-100%",

};

this.titlePosition = {

top: "-100%",

left: "-100%",

};

this.tooltopContent = {};

},

复制代码

// mounted

this.renderer.domElement.addEventListener(

"mousemove",

this.onMousemove,

false

);

this.$refs.tooltipBox.addEventListener(

"mouseleave",

this.handleTooltipHide,

false

);

最終得到效果：

**場景切換**

全景看房總不能一直看一個房間，所以我們還需要實現點擊切換場景功能。 直接codeing...

changeContentAndtips(index) {

// 清除场景数据内所有的精灵标签

this.scene.children = this.scene.children.filter(

(item) => String(item.type) !== "Sprite"

);

// 储存数组置空

this.tipsSpriteList = [];

// 重新加载贴图，这边应用gasp做一个简单的过渡动画，将透明度从0 ~ 1

let texture = new THREE.TextureLoader().load(this.dataList[index].image);

let sphereMaterial = new THREE.MeshBasicMaterial({

map: texture,

transparent: true,

opacity: 0,

});

this.sphere.material = sphereMaterial;

gsap.to(sphereMaterial, { transparent: true, opacity: 1, duration: 2 });

// 手动更新投影矩阵

this.camera.updateProjectionMatrix();

// 添加当前场景标签

this.addTipsSprite(index);

},

onMouseClick(e) {

e.preventDefault();

let element = this.$refs.threeDBox;

let raycaster = new THREE.Raycaster();

let mouse = new THREE.Vector2();

mouse.x = (e.clientX / element.clientWidth) \* 2 - 1;

mouse.y = -(e.clientY / element.clientHeight) \* 2 + 1;

raycaster.setFromCamera(mouse, this.camera);

let intersects = raycaster.intersectObjects(this.tipsSpriteList, true);

if (intersects.length > 0 && intersects[0].object.content.showTitle) {

this.changeContentAndtips(intersects[0].object.content.image);

this.handleTooltipHide(e);

}

},

最終得到完整效果如下圖：

**總結**

在這個demo中有一些細節問題我處理的不是很好，例如載入時因為貼圖載入慢黑屏（這是因為貼圖太大，優化時可加入等待動畫），還有就是載入順序沒處理好，場景貼圖還沒載入完標籤已經進來了。