基于三维展厅的 Three.js 应用

最近在实现基于三维展厅的 web3D 全景图，技术上选用了 Three.js 作为基础，主要介绍 Three.js 的开发基础和基本原理，以及如何实现 3D 全景图和热点编辑。想在 web 端实现 3D 全景图的效果。目前国外 3D 全景图比较好的是 KrPano 和 Pano2VR，国内很多 3D 全景服务是在使用 KrPano 的工具。老师想让我仿照 Pano2VR 软件实现一个通过 web 进行在线编辑全景中的热点，并生成相应参数的 xml 字符串与后台交互实现全景热点的 web 编辑和预览。

WebGL 是一套跨平台的针对 3D 图形 API 的 web 标准，借助 canvas 元素和 DOM 接口实现 3D 图形绘制，其基于 OpenGL ES 2.0 标准，和 openGL ES 2.0 规范非常接近。目前，主流浏览器普遍实现了对 WebGL 的支持。

而我所选用的技术 Three.js，是基于简化 WebGL 开发复杂度和降低入门难度的目的，在 WebGL 标准基础上封装了一个轻量级的 JS 3D 库。并具有如下特点：

1. 完备——具备 3D 开发所需完整功能，基本上使用 WebGL 能实现的效果，用 Three.js 都能更简单地实现
2. 易用——架构设计比较清晰和合理，易于理解，扩展性较好，且开发效率高于 WebGL
3. 开源——项目开源，且有一批活跃的贡献者，持续维护升级中

Three.js 的主要组件包括：Scene（场景）；Material & Texture（材质 & 纹理）；Geometry（几何）；Object（物体）；Light （光线）；Camera （相机）；Renderer & Shader (渲染器 & 着色器)；Loader （加载器）。

全景图，又称 Panorama，根据 WikiPedia 的定义，是一种宽角度的物理场景视图或展现形式，包括绘画、摄影、电影、震动映像或 3D 模式等。Web 3D 全景图即是在 Web 页面内，以 3D 的展现形式，将通过某种介质记录（图片&视频）的宽角度（一般是 360°）现实场景进行展示和播放。简单说，Web3D 全景的核心就是做一个全景图播放器。在不远的过去，webGL 还没有被提出和支持的时候，要想实现 3D 全景还是要依赖 Flash 的。随着浏览器，尤其是 HTML5 对 webGL 的支持，使用 JS 实现 3D 全景图具备了可行性。

我通过 Three.js 官网查找相关全景的例子(包括视频全景，图片全景等)，一般情况下全景图的实现有两种策略：

- 将全景图片切图为 6 张子图贴到六面体的六个面上
- 将全景图片作为纹理整体贴合到一个球体上

最后选用了将全景图片作为纹理整体贴合在球体上实现 web 全景图。具体代码如下：

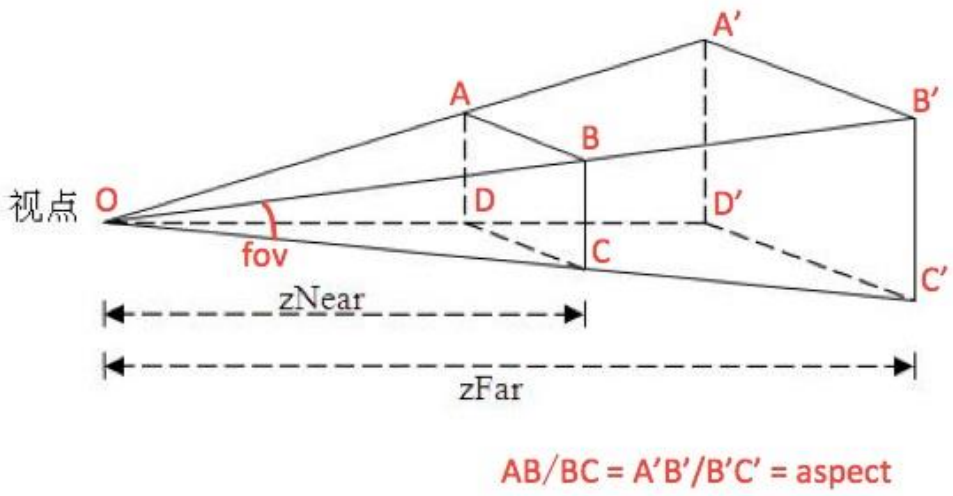
```
const geometry = new THREE.SphereGeometry(500, 60, 40);
geometry.scale(-1, 1, 1);

//防止跨域用canvas作为纹理
let canvas = document.createElement("canvas");
canvas.style.backgroundColor = "rgba(255,255,255,0)";
let context = canvas.getContext("2d");
let img = new Image();
img.src = imgurl[0];

//绘制全景图
img.onload = function () {
    canvas.width = this.width;
    canvas.height = this.height;
    context.drawImage(img, 0, 0, this.width, this.height);
    let texture = new THREE.Texture();
    texture.image = canvas;
    texture.needsUpdate = true;//开启纹理更新
    texture.minFilter = THREE.LinearFilter;//minFilter属性：指定纹理如何缩小
    let material = new THREE.MeshBasicMaterial({
        map: texture,
        transparent: false
    });
    mesh = new THREE.Mesh(geometry, material);
    scene.add(mesh);
};
```

如上代码，通过创建 canvas 元素，把图片绘制在 canvas 元素上，并把纹理的 image 属性设置为 canvas 元素，并把使用 Basic 材质的球形实体添加到场景中即可实现。

我们能够让全景图能够有动态的变化，有两种方法：

- 位置变换:通过控制观察者位置的变换来实现场景整体的运动效果
- 摄像机属性变换:和人眼不同，摄像机作为观察者更加精确，同时增加了视角等属性，通过改变这些属性可以获得不同的观察效果

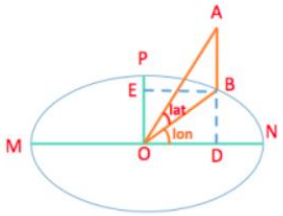
所以我们采用摄像机的属性变换，Three.js 提供的 Camera(摄像机)主要有两种，一种是正交投影相机(Orthographic Camera)，一种是透视投影相机(Perspective Camera)。两种相机的区别在于投影方式：正交投影等同平行投影的效果；透视投影则是一个物体的位置距离相机越远，则视觉体积越小，透视投影和人类的视觉感知是一致的，所以选用透视投影的相机。下面是透视投影相机的 4 个主要属性：



- near: 近视距，从视点 O 到近视面 ABCD 的垂直距离
- far: 远视距，从视点 O 到远视面 A'B'C'D'的垂直距离
- fov: 视线纵向张开角度，任意过视点 O 的竖切面的顶角，例如角 BOC 和角 A'OD'
- aspect: 代表视线横向展开幅度，等于任意横截面的宽高比，例如 AB/BC

这些属性共同确定了摄像机的可视区域，称为视域，渲染器只会渲染摄像机视域内的内容，也就是最终呈现的内容。改动摄像机的 position 或其他四个属性，视域就会随之改变。通过控制摄像机持续改变位置或其他属性，就可以实现全场景的运动效果。并通过 requestAnimationFrame 实现将发生改变的世界场景实时的渲染到 web 页面上

在球体模拟的 3D 全景场景下，可以使用经纬度坐标和半径来绝对定位空间中的一个点，同时鼠标控制下的摄像机镜头旋转用经纬度变化表示比较方便。但是最终我们需要将经纬度转换为真实的空间坐标，转换关系如下：



以点A为例：

R = OA；lat为维度，lon为经度；

Z_A = AB = OA * Sin(lat) = R * Sin(lat);

X_A = OD = OB * Cos(lon) = OA * Cos(lat) * Cos(lon)
= R * Cos(lat) * Cos(lon);

Y_A = OE = BD = OB * Sin(lon) = OA * Cos(lat) *
Sin(lon) = R * Cos(lat) * Sin(lon);

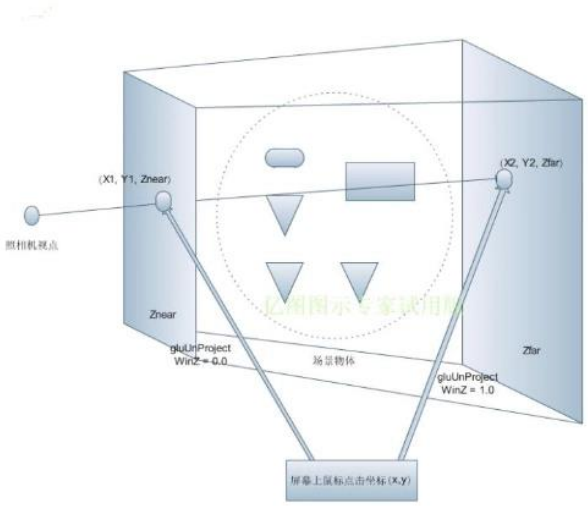
A 的坐标为(X_A, Y_A, Z_A)

具体代码如下：

```
onDocumentMouseMove(event) {
    if (isUserInteracting === true) {
      lon = (onPointerDownPointerX - event.clientX) * 0.1 + onPointerDownLon;
      lat = (event.clientY - onPointerDownPointerY) * 0.1 + onPointerDownLat;
    }

  },
  onDocumentMouseUp() {
    isUserInteracting = false;
    // camera.fov += event.deltaY * 0.05;
  },
  onDocumentMouseWheel(event) {
    camera.fov += event.deltaY * 0.05;
    if (camera.fov >= 10 && camera.fov <= 110) {
      camera.updateProjectionMatrix();//projectionMatrix – 投影变换矩阵
    }
  },
  animate() {
    if (this.imgUrl.length > 1) {
      return;
    }
    this.update();
    requestAnimationFrame(this.animate);
  },
  update() {
    //控制自动旋转速度
    if (isUserInteracting === false) {
      lon += 0;
    }
    lat = Math.max(-85, Math.min(85, lat));
    phi = THREE.Math.degToRad(90 - lat);
    theta = THREE.Math.degToRad(lon);//degToRad()方法返回与参数degrees所表示的角度相等的弧度值
    camera.target.x = 500 * Math.sin(phi) * Math.cos(theta);
  camera.target.y = 500 * Math.cos(phi);
  camera.target.z = 500 * Math.sin(phi) * Math.sin(theta);
  camera.lookAt(camera.target);
  renderer.render(scene, camera);
  },
```

通过 isUserInteracting 变量判断用户是否按住鼠标的左右键，若用户按住鼠标键进行移动(即对全景图进行拖拽)的话，则改变相机的 lon 和 lat 实现全景图的移动。并添加滚轮事件，通过滚轮的移动则让 camera 的 fov(视场)增加，超出一定范围则更新投影变换矩阵从而让鼠标滚轮实现全景的放大和缩小。

由于浏览器是一个 2d 视口，而在里面显示 three.js 的内容是 3d 场景，所以，现在有一个问题就是如何将 2d 视口的 x 和 y 坐标转换成 three.js 场景中的 3d 坐标。Three.js 是通过 THREE.Raycaster 射线用于鼠标拾取（计算出鼠标移过的三维空间中的对象）。如下图所示：



鼠标在屏幕上点击的时候，得到二维坐标 p(x, y),再加上深度坐标的范围(0, 1)，就可以形成两个三位坐标 A(x1, y1, 0), B(x2, y, 1)，由于它们的 Z 轴坐标是 0 和 1，则转变到投影坐标系的话，一定分别是前剪切平面上的点和后剪切平面上的点，也就是说，在投影坐标系中，A 点一定在能看见的所有模型的最前面，B 点一定在能看见的所有的模型的最后边，将 AB 点连成线，AB 线穿过的物体就是被点击的物体。而 Three.js 提供一个射线类 Raycasting 来拾取场景里面的物体。更方便的使用鼠标来操作 3D 场景。

可以通过给 raycaster 相交的物体添加删除热点绑定事件从而实现了鼠标点击之后删除热点的功能。

为了实现场景对浏览器大小改变的自适应，我们给浏览器注册监听事件，只要大小发生改变，就触发更新照相机的 aspect 属性(即屏幕的长宽比)，并且用 renderer 进行重新渲染即可。

老师需要能在网页上实现跳转热点 ,图像热点以及视频热点的编辑 ,通过操作 Pano2VR 软件，我发现图片热点随着视角的移动会发生显著的形变，然而 Three.js 中的基础二维几何体(PlaneGeometry)无法实现，原因是二维几何体无法贴附在球面上，虽然图像在我们绘制时是四边形，但实际上是球面上的一个“曲面”。所以我选择通过用户在场景中点击四个点，通过四个点在球面上绘制两个三角面片从而实现相应的随视角拖动而发生形变的效果。

但是 Three.js 中的 Face3()方法(即三角面片的构造函数)必须指定三个点和对应的法向量(默认是三角形两条边的标准化叉积)，由于用户在点击过程中点的位置是任意的，第四个点和已生成的三角面片中的哪两个点生成三角面片是未知的 ,如果在用户点击完四个点后绘制三角面片很有可能导致构造函数传参有问题，而无法实现四边形面片。所以每个三角面片的绘制都实时呈现出来，类似于画图直线的橡皮筋效果，这样用户每次都能够判断是哪几个点生成的三角面片从而解决了这个问题。

跳转热点，我则是通过 sprite(精灵图)来实现的，因为精灵图本身很小，随着视角的拖动在 Pano2VR 中很难观察到形变，用精灵图作为图片纹理很适合绘制跳转热点。在用户绘制时获取热点的位置参数，把热点位置乘以一定的比例系数，放置在球内部，则随着鼠标拖动跳转热点就不会发生明显的形变。

视频热点目前还没有完全完成 ,想法是通过视频作为纹理贴图来实现场景中视频热点的播放。

在 Pano2VR 中为了能让视频热点能够放置到合适的位置，则需要实现三角面片能够在场景中的拖拽和缩放，本质上就是把鼠标的拖动距离转化为三角面片位置或角度变化量。在 Three.js 中，代表物体的基类是 Object3D，该组件提供了对物体变化的基本支持：

1.Object3D.scale 放缩

2.Object3D.rotate 旋转

3.Object3D.transform 位移

Three.js 也提供了拖拽控件 DragControls.js，但是这是基于三维坐标轴对三维模型的拖拽，而实际上我们需要能够在“球面”上平移和缩放。如果想实现球面上”曲面”的移动，显然不是那么简单。虽然 Pano2VR 是通过 pan（上下摇摄）和 tilt（左右倾斜）来实现热点在各个方面的移动，而 Three.js 绘制三角形面片需要的是三维坐标，所以在水平移动后，难以保证所有的点都是在球面上。

为了解决这个问题，我有查阅了相关二维屏幕坐标与三维实体交点的转换关系，由于视频存在分辨率问题，如 720×480 等格式，其实 720 像素实际上就对应着二维屏幕坐标上两个顶点间相隔 720 像素的距离。

所以我把视频热点设计成用户点击一个点之后，我们再把用户的这个点的二维坐标加上相应的像素长度最终得到对应的四个点的屏幕二维坐标，在根据浏览器中页面的比例转换为 NOC 标准设备坐标，在通过 Three.js 中 Vector3 的 project 方法把标准设备坐标转换为世界坐标，然后用 Raycaster(射线)的 set 方法用四个点的世界坐标和相机原点连接形成射线获得三维实体上的相交点，最后用球上对应的四个相交点绘制出对应的四边形面片。

视频热点的旋转，平移，放缩都设计为基于屏幕二维坐标的变换，这个操作很简单，也符合 Pano2VR 的实际操作效果，但是又产生了新的问题！

通过屏幕二维坐标进行绘制的话，鼠标拖拽或者滚轮缩放都会导致屏幕二维坐标的改变，而相应的旋转，放缩，平移还是基于拖拽前的屏幕二维坐标，所以会产生问题。

为了修复这个问题，我发现屏幕二维坐标转换为三维世界坐标是固定不变的，不会由于相机的移动而发生改变，所以每次进行旋转，平移，放缩变换前要保存一个含有四个顶点对应的三维世界坐标的数组，在绘制时通过 unproject()方法又投影到二维屏幕坐标上，随着用户的操作不断更新数组中的值就可以实现连续的操作。

心得体会

由于项目刚接手，Three.js 的学习也是很零散，都是需要解决什么问题，才学习与问题相关的一部分知识，并没有系统的学习，希望项目做完之后整体系统地学习 Three.js 和 WebGL 相关内容。

编程技巧上也有很多问题，由于组件之间的数据传递，需要命名很多变量，变量的命名不合乎规范，存在大量重复的代码，有一些封装成了函数，但是还有很多没有进行封装，希望之后编写代码时能够注意这些问题。