使用Three.js建立一个基本的3D动画场景

原创 千瞳 于 2018-08-30 23:08:29 发布 8626 收藏 39 版权

分类专栏： 前端 文章标签： javascript three.js 3D

> 本文主要内容翻译自：https://tympanus.net/codrops/2016/04/26/the-aviator-animating-basic-3d-scene-threejs/

今天我们准备创建一个简单的飞机飞行的3D场景，使用的工具是Three.js。这是一个3D javascript，通过这个库我们可以更简单的通过WebGL编写3D程序。因为WebGL的复杂性和GLSL语言（OpenGL着色语言）的语法问题，导致很多人望而却步。但是通过Three.js我们可以很容易的在浏览器中实现3D效果。

在这一个教程中我们将会创建一个简单的3D场景，在两个主要场景中有一些简单的交互。在第一个部分我们会介绍Three.js的基本内容并且介绍如何搭建一个简单的场景。在第二个部分我们会介绍如何优化某些形状，如何为场景的不同元素添加合适的氛围和更好的移动效果。

现在开始吧！

The HTML & CSS

> Three.js的官网地址：https://threejs.org/

下载好three.js或者Three.min.js脚本文件之后，就可以准备编写程序了。

第一件事是把这个脚本引入到你的HTML文件的header中去：

```
<script type="text/javascript" src="js/three.min.js"></script>
```

接下来你需要在HTML文件的body中创建一个容器元素来放置渲染场景:

```
<div id="world"></div>
```

你可以简单的设置一下这个容器的格式，让它填充满整个页面：

```
* { margin: 0; }
#world {
    position: absolute;
    width: 100%;
    height: 100%;
    overflow: hidden;
    background: linear-gradient(#e4e0ba, #f7d9aa);
}
```

OK，这样写完的话，现在的页面应该是这样的：

The Javascript

如果你之前有写过Javascript的话，那么对你来说three.js还是很容易上手的。来看一下我们将要实现的代码的各个部分。

色块

在开始动手写代码编写场景之前，我发现提前定义好你所需要使用的色块对后面写程序会有很大的帮助，在这个项目中我们会使用到以下颜色：

```
var Colors = {
    red: 0xf25346,
    white: 0xd8d0d1,
    brown: 0x59332e,
    pink: 0xf5986e,
    brownDark: 0x23190f,
    blue: 0x68c3c0
};
```

代码结构

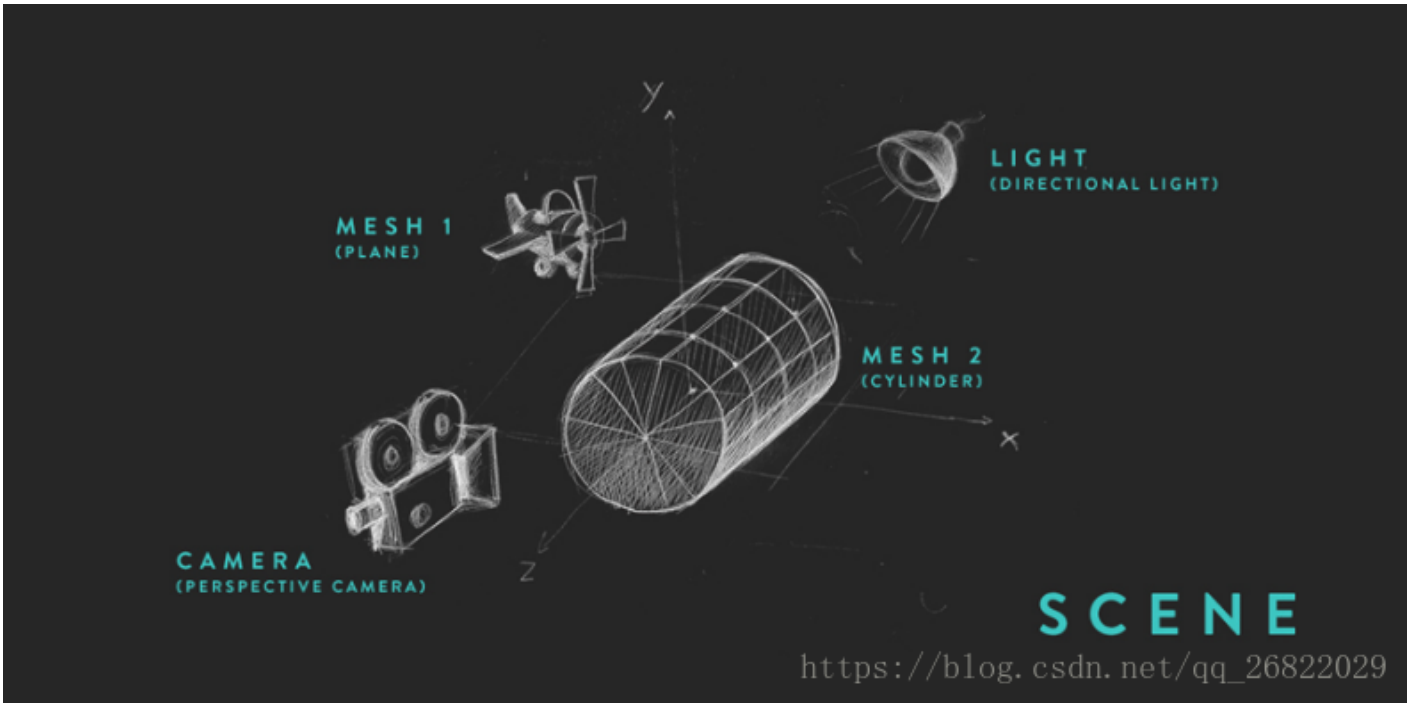
尽管Javascript的代码相当冗余，但它的结构却是相当简单的。所有主要的我们需要创建的函数都被放到了init()函数中。

```
window.addEventListener('load', init, false);

function init() {
    // 创建场景，相机和渲染器
    createScene();
    // 添加光源
    createLights();
    // 添加对象
    createPlane();
    createSea();
    createSky();


    // 调用循环函数，在每帧更新对象的位置和渲染场景
    loop();
}
```

设置场景

想要创建一个three.js场景，我们至少需要以下内容：

1. **一个场景**：把这个看做一个舞台，然后将所有需要的对象添加上去。
2. **一个相机**：在这个案例中我们创建一个透视摄像机，但它也可能是投影相机。
3. **一个渲染器**：渲染器将会使用WebGL渲染场景中的所有的物体。
4. **一个或多个物体**：在这个案例中，我们会创建一架飞机，一片海和天空（少量云）。
5. **一个或多个光源**：可以使用不同样式的光源。在这个案例中我们主要使用营造氛围的半球光和制造阴影的直射光。



我们在createScene()中创建场景，相机和渲染器。

```
var scene, camera, fieldOfView, aspectRatio, nearPlane, farPlane, HEIGHT, WIDTH, renderer
    function createScene() {
    // Get the width and the height of the screen,
    // use them to set up the aspect ratio of the camera
    // and the size of the renderer.
    HEIGHT = window.innerHeight;
    WIDTH = window.innerWidth;

    // Create the scene
    scene = new THREE.Scene();

    // Add a fog effect to the scene; same color as the
    // background color used in the style sheet
    scene.fog = new THREE.Fog(0xf7d9aa, 100, 950);

    // Create the camera
    aspectRatio = WIDTH / HEIGHT;
    fieldOfView = 60;
    nearPlane = 1;
    farPlane = 10000;

        /**
        * PerspectiveCamera 透视相机
        * @param fieldOfView 视角
        * @param aspectRatio 纵横比
        * @param nearPlane 近平面
        * @param farPlane 远平面
        */
```

```
        */
                camera = new THREE.PerspectiveCamera(
        fieldOfView,
        aspectRatio,
        nearPlane,
        farPlane
        );

    // Set the position of the camera
    camera.position.x = 0;
    camera.position.z = 200;
    camera.position.y = 100;

    // Create the renderer
    renderer = new THREE.WebGLRenderer({
        // Allow transparency to show the gradient background
        // we defined in the CSS
        alpha: true,

        // Activate the anti-aliasing; this is less performant,
        // but, as our project is low-poly based, it should be fine :)
        antialias: true
    });

    // Define the size of the renderer; in this case,
    // it will fill the entire screen
    renderer.setSize(WIDTH, HEIGHT);

    // Enable shadow rendering
    renderer.shadowMap.enabled = true;

    // Add the DOM element of the renderer to the
    // container we created in the HTML
    container = document.getElementById('world');
    container.appendChild(renderer.domElement);

    // Listen to the screen: if the user resizes it
    // we have to update the camera and the renderer size
    window.addEventListener('resize', handleWindowResize, false);
}
```

当屏幕的大小改变之后，我们需要更新渲染器的大小并且更新相机的纵横比：

```
function handleWindowResize() {
    // update height and width of the renderer and the camera
    HEIGHT = window.innerHeight;
```

```
    WIDTH = window.innerWidth;
                                        renderer.setSize(WIDTH, HEIGHT);
    camera.aspect = WIDTH / HEIGHT;
    camera.updateProjectionMatrix();
}
```

灯光

在创建一个场景时，灯光应该是最棘手的部分。灯光可以奠定整个场景的基调，因此要仔细设计这一部分。在这一部分，我们将只设置灯光使得场景中的所有物体可见即可。

```
var hemisphereLight, shadowLight;

function createLights() {
    // A hemisphere light is a gradient colored light;

 // the first parameter is the sky color, the second parameter is the ground color,
    // the third parameter is the intensity of the light
 hemisphereLight = new THREE.HemisphereLight(0xaaaaaa,0x000000, .9)
    // A directional light shines from a specific direction.
    // It acts like the sun, that means that all the rays produced are parallel.
    shadowLight = new THREE.DirectionalLight(0xffffff, .9);

    // Set the direction of the light
    shadowLight.position.set(150, 350, 350);

    // Allow shadow casting
    shadowLight.castShadow = true;

    // define the visible area of the projected shadow
    shadowLight.shadow.camera.left = -400;
    shadowLight.shadow.camera.right = 400;
    shadowLight.shadow.camera.top = 400;
    shadowLight.shadow.camera.bottom = -400;
    shadowLight.shadow.camera.near = 1;
    shadowLight.shadow.camera.far = 1000;

    // define the resolution of the shadow; the higher the better,
    // but also the more expensive and less performant
    shadowLight.shadow.mapSize.width = 2048;
    shadowLight.shadow.mapSize.height = 2048;

    // to activate the lights, just add them to the scene
    scene.add(hemisphereLight);
    scene.add(shadowLight);
}
```

如你所见，创建灯光需要好多的参数。不要犹豫，大胆尝试用不同的颜色，强度的光源。你发现不同的光源在场景中能够营造有趣的氛围和环境。而且你会找到感觉：如何按照你的需求优化它们。

使用Tree.js创建一个对象

> 如果你会使用3D建模软件的话，你可以自己3D模型然后导入到Three.js中，不过在这个教程中我不会涉及到这一部分。不过为了让大家了解如何创建对象，我们会使用Three.js中提供的几何体来创建物体。

Three.js内置了许多内置的对象原型，比如立方体、球、圆形面、圆柱体和飞机。

在我们的项目中，所有对象都由这些基本的原型组合而成。对于一个低多边形风格的场景来说这再合适不过了，我们也不需要使用3D建模软件来构建模型了，开心。

用圆柱体来表示大海

首先来创建大海，因为这应该是我们的场景里最简单的对象了。简单起见，我们用一个放置在屏幕下方的圆柱体来简单的表示大海。之后再深入研究如何改变大海的外观。

接下来我们让大海和浪花看起来更逼真一点。

```
// First let's define a Sea object :
Sea = function(){

    // create the geometry (shape) of the cylinder;
    // the parameters are:

 // radius top, radius bottom, height, number of segments on the radius, number of se
    var geom = new THREE.CylinderGeometry(600,600,800,40,10);
    // rotate the geometry on the x axis
    geom.applyMatrix(new THREE.Matrix4().makeRotationX(-Math.PI/2));

    // create the material
    var mat = new THREE.MeshPhongMaterial({
        color:Colors.blue,
        transparent:true,
        opacity:.6,
        shading:THREE.FlatShading,
    });
```

```
    // To create an object in Three.js, we have to create a mesh
    // which is a combination of a geometry and some material
    this.mesh = new THREE.Mesh(geom, mat);
    // Allow the sea to receive shadows
    this.mesh.receiveShadow = true;
}

// Instantiate the sea and add it to the scene:

var sea;

function createSea(){
    sea = new Sea();

    // push it a little bit at the bottom of the scene
    sea.mesh.position.y = -600;

    // add the mesh of the sea to the scene
    scene.add(sea.mesh);
}
```

总结一下创建一个物体需要些什么：

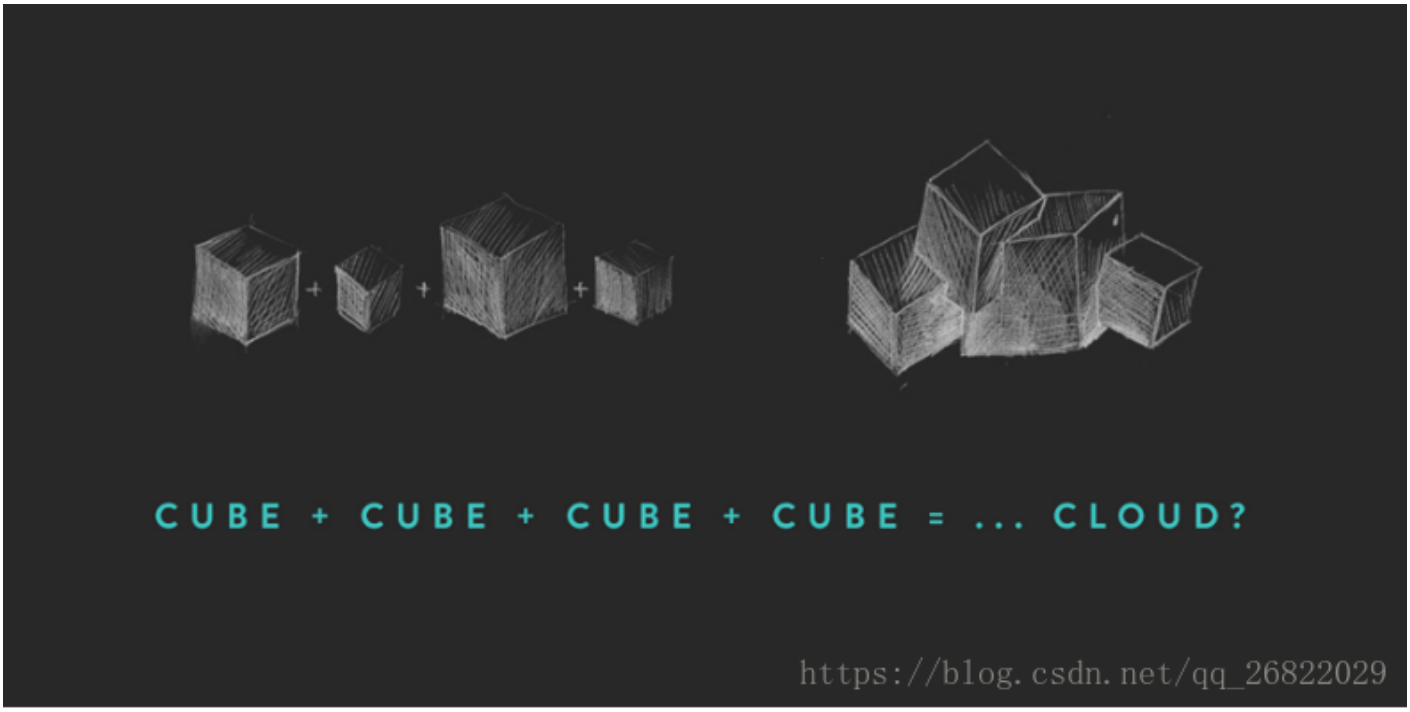
1. 创建几何体
2. 创建材质
3. 把他们进行匹配
4. 匹配后加入场景

通过这些步骤，我们可以创建许多不同种类的几何体。现在，如果我们把它们组合起来，就可以创建更多复杂的形状。

接下来我们将会学习如何创建更加精致的几何形状。

简单的立方体组合成复杂的几何形状

云朵的形状比较复杂，因为云朵是有许多个立方体随机的组合形成的几何形状。




```
Cloud = function(){
    // Create an empty container that will hold the different parts of the cloud
    this.mesh = new THREE.Object3D();

    // create a cube geometry;
    // this shape will be duplicated to create the cloud
    var geom = new THREE.BoxGeometry(20,20,20);

    // create a material; a simple white material will do the trick
    var mat = new THREE.MeshPhongMaterial({
        color:Colors.white,
    });

    // duplicate the geometry a random number of times
    var nBlocs = 3+Math.floor(Math.random()*3);
    for (var i=0; i<nBlocs; i++ ){

        // create the mesh by cloning the geometry
        var m = new THREE.Mesh(geom, mat);

        // set the position and the rotation of each cube randomly
        m.position.x = i*15;
        m.position.y = Math.random()*10;
        m.position.z = Math.random()*10;
        m.rotation.z = Math.random()*Math.PI*2;
        m.rotation.y = Math.random()*Math.PI*2;

        // set the size of the cube randomly
        var s = .1 + Math.random()*.9;
```

```
        m.scale.set(s,s,s);
        // allow each cube to cast and to receive shadows
        m.castShadow = true;
        m.receiveShadow = true;

        // add the cube to the container we first created
        this.mesh.add(m);
    }
}
```

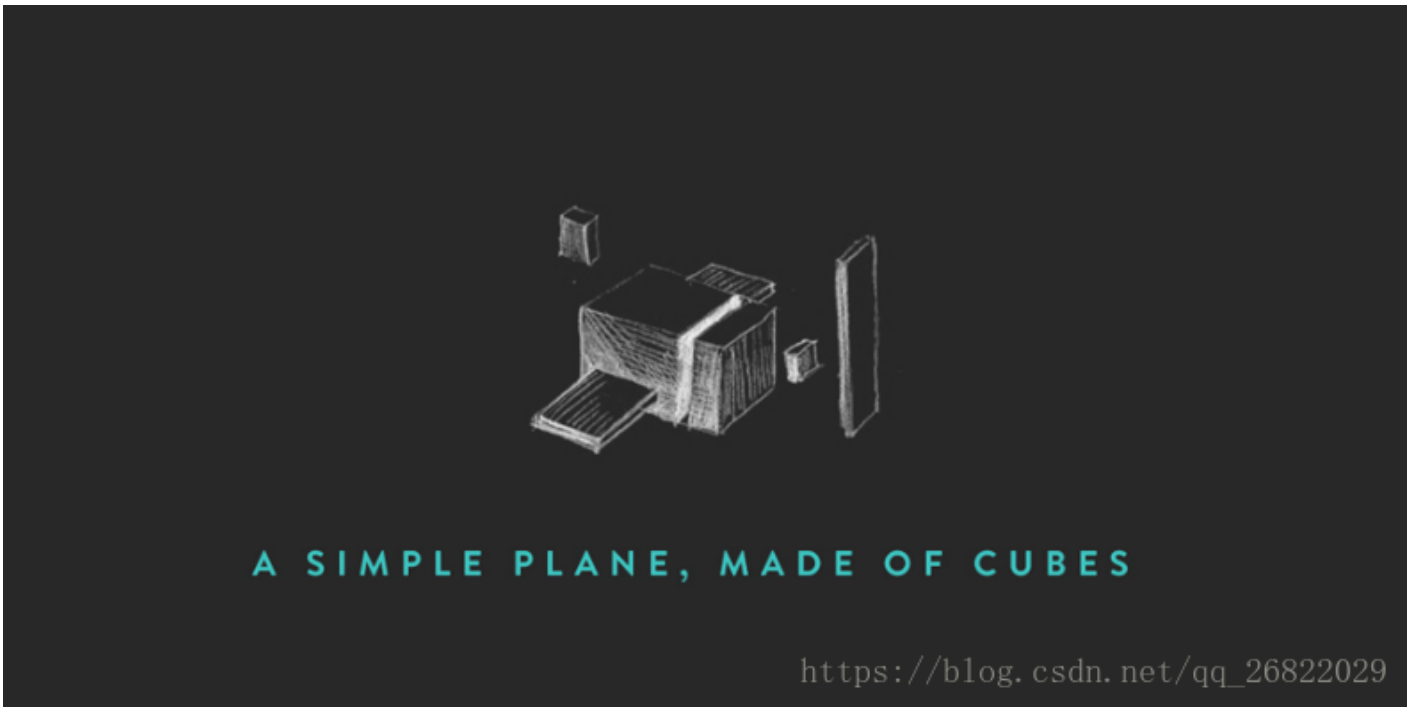
通过以上代码我们定义好了云朵这个对象，下面通过复制该对象并将其放置在z轴的随机位置来实现天空的效果 。

```
// Define a Sky Object
Sky = function(){
    // Create an empty container
    this.mesh = new THREE.Object3D();

    // choose a number of clouds to be scattered in the sky
    this.nClouds = 20;

    // To distribute the clouds consistently,
    // we need to place them according to a uniform angle
    var stepAngle = Math.PI*2 / this.nClouds;

    // create the clouds
    for(var i=0; i<this.nClouds; i++){
        var c = new Cloud();

        // set the rotation and the position of each cloud;
        // for that we use a bit of trigonometry
        var a = stepAngle*i; // this is the final angle of the cloud
        var h = 750 + Math.random()*200; // this is the distance between the center

        // Trigonometry!!! I hope you remember what you've learned in Math :)
        // in case you don't:
        // we are simply converting polar coordinates (angle, distance) into Cartesi
        c.mesh.position.y = Math.sin(a)*h;
        c.mesh.position.x = Math.cos(a)*h;

        // rotate the cloud according to its position
        c.mesh.rotation.z = a + Math.PI/2;

        // for a better result, we position the clouds
        // at random depths inside of the scene
        c.mesh.position.z = -400-Math.random()*400;
```

```
                // we also set a random scale for each cloud
            var s = 1+Math.random()*2;
            c.mesh.scale.set(s,s,s);

            // do not forget to add the mesh of each cloud in the scene
            this.mesh.add(c.mesh);
        }
    }

    // Now we instantiate the sky and push its center a bit
    // towards the bottom of the screen

    var sky;

    function createSky(){
        sky = new Sky();
        sky.mesh.position.y = -600;
        scene.add(sky.mesh);
    }
```

再复杂一点：创建一架飞机

有一个好消息一个坏消息，坏消息是创建飞机对象的代码将会有点长而且有点复杂。好消息是不管代码多么长多么复杂，使用到的方法我们都已经学过了。这里所有的都是关于组合和封装的代码。



```
var AirPlane = function() {

    this.mesh = new THREE.Object3D();

    // Create the cabin
    var geomCockpit = new THREE.BoxGeometry(60,50,50,1,1,1);

var matCockpit = new THREE.MeshPhongMaterial({color:Colors.red, shading:THREE.FlatSl
    var cockpit = new THREE.Mesh(geomCockpit, matCockpit);
cockpit.castShadow = true;      cockpit.receiveShadow = true;
    this.mesh.add(cockpit);

    // Create the engine
    var geomEngine = new THREE.BoxGeometry(20,50,50,1,1,1);

var matEngine = new THREE.MeshPhongMaterial({color:Colors.white, shading:THREE.Flat
    var engine = new THREE.Mesh(geomEngine, matEngine);
engine.position.x = 40;      engine.castShadow = true;
    engine.receiveShadow = true;
    this.mesh.add(engine);

    // Create the tail
    var geomTailPlane = new THREE.BoxGeometry(15,20,5,1,1,1);

var matTailPlane = new THREE.MeshPhongMaterial({color:Colors.red, shading:THREE.Fla
    var tailPlane = new THREE.Mesh(geomTailPlane, matTailPlane);
tailPlane.position.set(-35,25,0);      tailPlane.castShadow = true;
    tailPlane.receiveShadow = true;
    this.mesh.add(tailPlane);

    // Create the wing
    var geomSideWing = new THREE.BoxGeometry(40,8,150,1,1,1);

var matSideWing = new THREE.MeshPhongMaterial({color:Colors.red, shading:THREE.Flat
    var sideWing = new THREE.Mesh(geomSideWing, matSideWing);
sideWing.castShadow = true;      sideWing.receiveShadow = true;
    this.mesh.add(sideWing);

    // propeller
    var geomPropeller = new THREE.BoxGeometry(20,10,10,1,1,1);

var matPropeller = new THREE.MeshPhongMaterial({color:Colors.brown, shading:THREE.F
    this.propeller = new THREE.Mesh(geomPropeller, matPropeller);
this.propeller.castShadow = true;      this.propeller.receiveShadow = true;

    // blades
    var geomBlade = new THREE.BoxGeometry(1,100,20,1,1,1);
```

```
    var matBlade = new THREE.MeshPhongMaterial({color:Colors.brownDark, shading:THREE.F
        var blade = new THREE.Mesh(geomBlade, matBlade);
    blade.position.set(8,0,0);
    blade.castShadow = true;
    blade.receiveShadow = true;
    this.propeller.add(blade);
    this.propeller.position.set(50,0,0);
    this.mesh.add(this.propeller);
};
```

现在我们可以实例化我们的飞机，将它加入到场景中去：

```
var airplane;

function createPlane(){
    airplane = new AirPlane();
    airplane.mesh.scale.set(.25,.25,.25);
    airplane.mesh.position.y = 100;
    scene.add(airplane.mesh);
}
```

渲染

到目前为止，我们已经创建了一组模型并且把它们加入到场景中。但是如果你现在运行这个程序，你能看到东西才怪！这是因为我们还没有渲染这个场景。加入下面一行渲染代码：

```
renderer.render(scene, camera);
```

动画

我们需要加点动画使得整个场景看起来更有生机，所以下面我们让飞机上的螺旋桨转起来，让天空中的云动起来。

为了实现动画我们需要添加一个无限循环：

```
function loop(){
    // Rotate the propeller, the sea and the sky
    airplane.propeller.rotation.x += 0.3;
    sea.mesh.rotation.z += .005;
```

```
    sky.mesh.rotation.z += .01;

    // render the scene
    renderer.render(scene, camera);

    // call the loop function again
    requestAnimationFrame(loop);
}
```

如你所见，现在我们将渲染器的 render() 函数移动到 loop() 函数中。因为每次修改物体的位置或颜色之类的属性就需要重新调用一次 render() 函数。

鼠标响应：添加交互操作

现在如果运行起程序来看，你会发现飞机一直在屏幕的中央。我们接下来想实现的，是让飞机跟随鼠标移动。

一旦这个html文件加载完成，我们需要添加一个监听器监听鼠标是否移动了。

为了实现这个功能，我们在init()函数中做如下修改：

```
function init(event){
    createScene();
    createLights();
    createPlane();
    createSea();
    createSky();

    //add the listener
    document.addEventListener('mousemove', handleMouseMove, false);

    loop();
}
```

另外，我们需要新创建一个函数handleMouseMove()来响应鼠标移动事件：

```
var mousePos={x:0, y:0};

// now handle the mousemove event

function handleMouseMove(event) {

    // here we are converting the mouse position value received
```

```
    // to a normalized value varying between -1 and 1;

 // this is the formula for the horizontal axis:
    var tx = -1 + (event.clientX / WIDTH)*2;

    // for the vertical axis, we need to inverse the formula
    // because the 2D y-axis goes the opposite direction of the 3D y-axis

    var ty = 1 - (event.clientY / HEIGHT)*2;
    mousePos = {x:tx, y:ty};

}
```

现在我们能获取鼠标的x和y值，接下来就需要正确的改变飞机的位置。

我们需要修改loop()函数，并且添加新的函数来更新飞机的位置。

```
function loop(){
    sea.mesh.rotation.z += .005;
    sky.mesh.rotation.z += .01;

    // update the plane on each frame
    updatePlane();

    renderer.render(scene, camera);
    requestAnimationFrame(loop);
}

function updatePlane(){

    // let's move the airplane between -100 and 100 on the horizontal axis,
    // and between 25 and 175 on the vertical axis,
    // depending on the mouse position which ranges between -1 and 1 on both axes;
    // to achieve that we use a normalize function (see below)

    var targetX = normalize(mousePos.x, -1, 1, -100, 100);
    var targetY = normalize(mousePos.y, -1, 1, 25, 175);

    // update the airplane's position
    airplane.mesh.position.y = targetY;
    airplane.mesh.position.x = targetX;
    airplane.propeller.rotation.x += 0.3;
}

function normalize(v,vmin,vmax,tmin, tmax){
```

```
    var nv = Math.max(Math.min(v,vmax), vmin);
    var dv = vmax-vmin;
    var pc = (nv-vmin)/dv;
    var dt = tmax-tmin;
    var tv = tmin + (pc*dt);
    return tv;

}
```

恭喜！到目前为止，你已经实现了一个能够跟随你鼠标移动的飞机了！看看目前实现的效果如何：

https://tympanus.net/Tutorials/TheAviator/part1.html