演示地址: http://creativejs.com/uploads/tutorials/three/Part1_particles/ThreeParticles.html

three.js 是 JavaScript 编写的 WebGL 第三方库。提供了非常多的 3D 显示功能。Three.js 是一款运行在浏览器中的 3D 引擎，你可以用它创建各种三维场景，包括了摄影机、光影、材质等各种对象。

下载地址: http://threejs.org/

使用 three.js 引擎实现空间粒子效果

首先创建一个 HTML 文件,引入 three.js 引擎包.

```
<!DOCTYPE HTML>
<html>
	<head>
		<meta charset="utf-8">
		<title>Three.js 实现 3D 空间粒子效果</title>
		<style type="text/css">
			body{
				background-color:#000000;
				margin:0px;
				overflow:hidden;
			}
		</style>
		<script src="scripts/three.js"></script>
	</head>
```

```
    <body >
    </body>
</html>
```

声明全局变量

```
    //定义应用所需的组件:相机,场景,渲染器
    var camera, scene, renderer;
    //跟踪鼠标的位置
    var mouseX = 0, mouseY = 0;
    //定义存储粒子的数组
    var particles = [];
```

相机:

OpenGL（WebGL）中、三维空间中的物体投影到二维空间的方式中，存在透视投影和正投影两种相机。

透视投影就是、从视点开始越近的物体越大、远处的物体绘制的较小的一种方式、和日常生活中我们看物体的方式是一致的。

正投影就是不管物体和视点距离，都按照统一的大小进行绘制、在建筑和设计等领域需要从各个角度来绘制物体，因此这种投影被广泛应用。

在 Three.js 也能够指定透视投影和正投影两种方式的相机。

场景:

场景就是一个三维空间。 用 [Scene] 类声明一个叫 [scene] 的对象。

渲染器:

三维空间里的物体映射到二维平面的过程被称为三维渲染。 一般来说我们都把进行渲染的操作叫做渲染器。

数据初始化

```
    //数据初始化
    function init(){
        //相机参数：
```

```
        //四个参数值分别代表:视野角：fov   纵横比：aspect  相机离视体最近的距离：near  相机离视体最远的距离：far
        camera = new THREE.PerspectiveCamera(80, window.innerWidth / window.innerHeight, 1, 4000 );
        //设置相机位置,默认位置为:0,0,0.
        camera.position.z = 1000;

        //声明场景
        scene = new THREE.Scene();
        //将相机装加载到场景
        scene.add(camera);

         //生成渲染器的对象
         renderer = new THREE.CanvasRenderer();
        //设置渲染器的大小
        renderer.setSize( window.innerWidth, window.innerHeight );
        //追加元素
        document.body.appendChild(renderer.domElement);
        //调用自定义的生成粒子的方法
        makeParticles();
        //添加鼠标移动监听
        document.addEventListener('mousemove',onMouseMove,false);
        //设置间隔调用 update 函数,间隔次数为每秒 30 次
        setInterval(update,1000/30);
    }
```

相机初始化说明:

实例中使用的是透视投影. var camera = new THREE.PerspectiveCamera(fov , aspect , near , far);
透视投影中，会把称为视体积领域中的物体作成投影图。 视体积是通过以下 4 个参数来指定。
视野角：fov
纵横比：aspect
相机离视体积最近的距离：near
相机离视体积最远的距离：far

设置相机的位置:
相机的位置坐标和视野的中心坐标，按照

```
//设置相机的位置坐标
camera.position.x = 100;
camera.position.y = 20;
camera.position.z = 50;
```

方式进行设置。 和该方式一样，下面这样的方法也可以。

```
camera.position.set(100,20,50);
```

此外还可以设置相机的上方向,视野中心等,
设置相机的上方向为正方向:

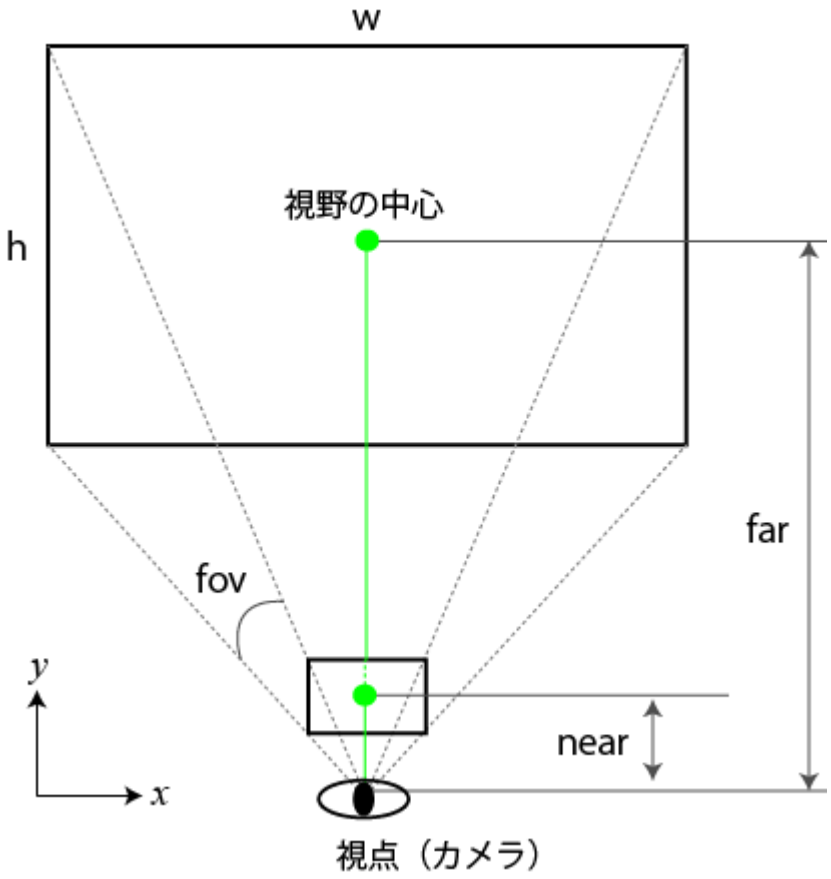
```
camera.up.x = 0;
camera.up.y = 0;
camera.up.z = 1;
```

设置相机的视野中心
利用[lookAt]方法来设置相机的视野中心。 「
lookAt()」的参数是一个属性包含中心坐标「x」「y」「z」的对象。
「lookAt()」方法不仅是用来设置视点的中心坐标、 在此之前设置的相机属性要发生实际作用，也需要调用 [lookAt] 方法。

其他投影方式

在 Three.js 中、有各种各样的类，用来来实现透视投影、正投影或者复合投影（透视投影和正投影）这样的相机。

```
var camera = THREE.OrthographicCamera = function ( left, right, top, bottom, near, far ) //正投影
var camera = THREE.CombinedCamera = function ( width, height, fov, near, far, orthonear, orthofar ) //複合投影
```



渲染器

创建 CanvasRenderer 对象.这是一个普通的 2D 画布对象,实例中我们添加到 body 标签中. 否则我们就不会看到它。我们想让它充满整个浏览器窗口，所以我们设置其大小为 window.innerwidth 和 window.innerheight。

鼠标监听

使用自定义函数 makeParticles()创建粒子,并为其添加 mousemove 侦听器来跟踪鼠标的位置,最后我们建立一个间隔调用 update 函数一秒 30 次。

update 函数中的定义如下:

```
function update() {
    updateParticles();
    renderer.render( scene, camera );
}
```

产生粒子的函数

```
//定义粒子生成的方法
function makeParticles(){

    var particle,material;
    //粒子从 Z 轴产生区间在-1000 到 1000
    for(var zpos=-1000;zpos<1000;zpos+=20){
        //we make a particle material and pass through the colour and custom particle render function we defined.
        material = new THREE.ParticleCanvasMaterial( { color: 0xffffff, program: particleRender } );
        //生成粒子
        particle = new THREE.Particle(material);
        //随即产生 x 轴,y 轴,区间值为-500-500
        particle.position.x = Math.random()*1000-500;
        particle.position.y = Math.random()*1000-500;
        //设置 z 轴
```

```
                particle.position.z = zpos;
                //scale it up a bit
                particle.scale.x = particle.scale.y = 10;
                //将产生的粒子添加到场景,否则我们将不会看到它
                scene.add(particle);
                //将粒子位置的值保存到数组
                particles.push(particle);
            }
        }
```

math . random()返回一个浮点数在 0 和 1 之间,我们乘以 1000,给了我们一个 0 到 1000 之间的数字。然后我们减去 500,这给了我们一个号码在-500 和 500 之间.我们也可以这样定义一个生成范围区间内随机值的函数

```
function randomRange(min, max) {
    return Math.random()*(max-min) + min;
}
```

绘制粒子形状的函数

```
    //定义粒子绘制函数
    function particleRender( context ) {
        //获取 canvas 上下文的引用
        context.beginPath();
        // and we just have to draw our shape at 0,0 - in this
        // case an arc from 0 to 2Pi radians or 360°  - a full circle!
        context.arc( 0, 0, 1, 0,    Math.PI * 2, true );
        //设置原型填充
        context.fill();
    }
```

定义粒子移动的函数,这里设置成移动速度随着鼠标距离 Y 轴 0 点的值越大,粒子移动越快,

```
    //移动粒子的函数
    function updateParticles(){

        //遍历每个粒子
        for(var i=0; i<particles.length; i++){
            particle = particles[i];
            //设置粒子向前移动的速度依赖于鼠标在平面 Y 轴上的距离
            particle.position.z +=  mouseY * 0.1;
            //如果粒子 Z 轴位置到 1000,将 z 轴位置设置到-1000,即移动到原点,这样就会出现无穷尽的星域效果.
            if(particle.position.z>1000){
                particle.position.z-=2000;
            }
        }
    }
```

鼠标移动时函数监听

```
    //鼠标移动时调用
    function onMouseMove(event){
        mouseX = event.clientX;
        mouseY = event.clientY;
    }
```

至此,空间粒子简单效果学习完毕.完整代码如下:

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="utf-8">
        <title>Three.js 实现 3D 空间粒子效果</title>
```

```
<style type="text/css">
    body{
        background-color:#000000;
        margin:0px;
        overflow:hidden;
    }
</style>
<script src="scripts/three.js"></script>
<script>
    //定义应用所需的组件:相机,场景,渲染器
    var camera, scene, renderer;
    //跟踪鼠标的位置
    var mouseX = 0, mouseY = 0;
    //定义存储粒子的数组
    var particles = [];

    //数据初始化
    function init(){
        //相机参数：
        //四个参数值分别代表:视野角：fov  纵横比：aspect 相机离视体最近的距离：near 相机离视体最远的距离：far
        camera = new THREE.PerspectiveCamera(80, window.innerWidth / window.innerHeight, 1, 4000 );
        //设置相机位置,默认位置为:0,0,0.
        camera.position.z = 1000;

        //声明场景
        scene = new THREE.Scene();
```

```
        //将相机装加载到场景
        scene.add(camera);

        //生成渲染器的对象
        renderer = new THREE.CanvasRenderer();
        //设置渲染器的大小
        renderer.setSize( window.innerWidth, window.innerHeight );
        //追加元素
        document.body.appendChild(renderer.domElement);
        //调用自定义的生成粒子的方法
        makeParticles();
        //添加鼠标移动监听
        document.addEventListener('mousemove',onMouseMove,false);
        //设置间隔调用 update 函数,间隔次数为每秒 30 次
        setInterval(update,1000/30);
    }

    function update() {
        //调用移动粒子的函数
        updateParticles();
        //重新渲染
        renderer.render( scene, camera );
    }

    //定义粒子生成的方法
    function makeParticles(){
```

```
        var particle,material;
        //粒子从 Z 轴产生区间在-1000 到 1000
        for(var zpos=-1000;zpos<1000;zpos+=20){
            //we make a particle material and pass through the colour and custom particle render function we defined.
            material = new THREE.ParticleCanvasMaterial( { color: 0xffffff, program: particleRender } );
            //生成粒子
            particle = new THREE.Particle(material);
            //随即产生 x 轴,y 轴,区间值为-500-500
            particle.position.x = Math.random()*1000-500; //math . random()返回一个浮点数在 0 和 1 之间
            particle.position.y = Math.random()*1000-500;
            //设置 z 轴
            particle.position.z = zpos;
            //scale it up a bit
            particle.scale.x = particle.scale.y = 10;
            //将产生的粒子添加到场景
            scene.add(particle);
            //将粒子位置的值保存到数组
            particles.push(particle);
        }
}

//定义粒子渲染器
function particleRender( context ) {
        //获取 canvas 上下文的引用
        context.beginPath();
```

```
        // and we just have to draw our shape at 0,0 - in this
        // case an arc from 0 to 2Pi radians or 360º - a full circle!
        context.arc( 0, 0, 1, 0,  Math.PI * 2, true );
        //设置原型填充
        context.fill();
}


//移动粒子的函数
function updateParticles(){

        //遍历每个粒子
        for(var i=0; i<particles.length; i++){
                particle = particles[i];
                //设置粒子向前移动的速度依赖于鼠标在平面 Y 轴上的距离
                particle.position.z +=  mouseY * 0.1;
                //如果粒子 Z 轴位置到 1000,将 z 轴位置设置到-1000
                if(particle.position.z>1000){
                        particle.position.z-=2000;
                }
        }
}

//鼠标移动时调用
function onMouseMove(event){
        mouseX = event.clientX;
```

```
                        mouseY = event.clientY;
                    }
                </script>
        </head>
        <body onload="init()">
        </body>
</html>
```

参考资料: http://creativejs.com/tutorials/three-js-part-1-make-a-star-field/

文章地址: http://www.cnblogs.com/dennisit/archive/2013/04/20/3032837.html