目录

[1. threejs的三大组件 2](#_Toc524470149)

[1.1 加个物体到场景中去，看看效果 3](#_Toc524470150)

[1.2 加性能监视器Stats 4](#_Toc524470151)

[1.3 加入gui控制器 4](#_Toc524470152)

[1.4 ascii效果 4](#_Toc524470153)

[2. 场景 5](#_Toc524470154)

[3.相机 6](#_Toc524470155)

[3.1透视相机 PerspectiveCamera 6](#_Toc524470156)

[3.2正交相机 OrthographicCamera 8](#_Toc524470157)

[5.光源 10](#_Toc524470158)

[5.1 环境光——AmbientLight光源的颜色会影响到整个场景， 10](#_Toc524470159)

[5.2 点光源——PointLight 11](#_Toc524470160)

[5.3 聚光灯光源——SpotLight 11](#_Toc524470161)

[5.4 方向光——DirectionalLight也称无限光（类似于太阳光的一种光源） 13](#_Toc524470162)

[5.5 半球光源----HemisphereLight 13](#_Toc524470163)

[5.6 平面光光源------AreaLight平面光光源可以定义为一个发光的矩形 14](#_Toc524470164)

[5.7 镜头眩光----LensFlare 14](#_Toc524470165)

[总结 14](#_Toc524470166)

[6.几何体 15](#_Toc524470167)

[7.材质 15](#_Toc524470168)

[8.相机移动 15](#_Toc524470169)

[9.加载和使用纹理 15](#_Toc524470170)

[10.物理效果 15](#_Toc524470171)

[11.粒子效果 15](#_Toc524470172)

[12.高级几何体 15](#_Toc524470173)

# 1. threejs的三大组件

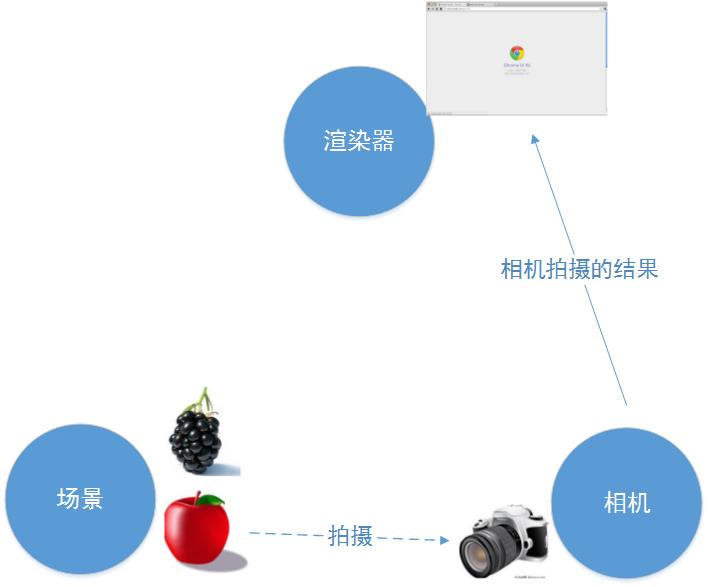
场景（scene）、相机（camera）和渲染器（renderer）

有这三样东西，我们才能够使用相机将场景渲染到网页上去

场景是一个物体的容器，开发者可以将需要的角色放入场景中，例如苹果，葡萄。同时，角色自身也管理着其在场景中的位置。

相机的作用就是面对场景，在场景中取一个合适的景，把它拍下来。

渲染器的作用就是将相机拍摄下来的图片，放到浏览器中去显示



看demo：basic.html

var scene = new THREE.Scene(); // 1.场景

var camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight, 0.1, 1000);// 2.透视相机

var renderer = new THREE.WebGLRenderer(); // 3.渲染器

renderer.setSize(window.innerWidth, window.innerHeight); // 设置渲染器的大小为窗口的内宽度，也就是内容区的宽度

document.body.appendChild(renderer.domElement);

## 1.1 加个物体到场景中去，看看效果

demo：basic-add-cube.html

## 1.2 加性能监视器Stats

stats插件：https://github.com/mrdoob/stats.js

demo:basic-stats.html

FPS表示：上一秒的帧数，这个值越大越好，一般都为60左右。

MS表示渲染一帧需要的毫秒数，这个数字是越小越好。

begin，在你要测试的代码前面调用begin函数，在你代码执行完后调用end()函数，这样就能够统计出这段代码执行的平均帧数了。

## 1.3 加入gui控制器

demo:basic-gui.html

var controls = new function () {

this.rotationSpeed = 0.02;

};

var gui = new dat.GUI();

gui.add(controls, 'rotationSpeed', 0, 0.5);

cube.rotation.x += controls.rotationSpeed;

几种用法：http://www.hangge.com/blog/cache/detail\_1785.html

## 1.4 ascii效果

demo:basic-ascii.html

1.将document.getElementById('WebGL-output').appendChild(renderer.domElement);换成：

effect = new THREE.AsciiEffect(renderer);

effect.setSize(window.innerWidth, window.innerHeight);

document.getElementById('WebGL-output').appendChild(effect.domElement)

2.将renderer.render(scene, camera);换成effect.render(scene, camera);

# 2. 场景

场景有三种组件：

相机：决定哪些东西将要显示在屏幕上渲染   
光源：它们对材质如何显示，以及生产阴影是材质如何产生影响   
物体（Mesh对象）：它们是在相机视图里主要的渲染对象：方块、球体等

变量

scene.children.length; //场景中物体的个数

函数

scene.add(obj); //向场景中添加物体

scene.remove(obj); //删除场景中的物体

scene.children(); //获取场景中的所有子对象

scene.getObjectByName(name); //通过名称获取场景中的物体对象

scene.traverse(function (e){}); //遍历场景中的每个物体，对物体进行操作

demo: scene-function.html

场景的两个属性 -- fog 雾化 和 overrideMaterial 材质覆盖

//雾化效果

//1-线性雾，密度随着距离的增加呈线性增长。参数为：雾的颜色，开始的地方，浓度的加深程度。

scene.fog = new THREE.Fog(0xffffff, 0.015, 100);

//2-指数雾，密度随距离呈指数级增长。参数为：雾的颜色，浓度

scene.fog = new THREE. FogExp2 (0xffffff, 0.015);

//材质覆盖：所有物体设置为同样的材质

scene.overrideMaterial = new THREE.MeshLambertMaterial({color: 0xffffff});

demo:scene-effect.html

# 3.相机

相机决定了场景中哪个角度的景色会被渲染出来。Threejs提供的相机有正交相机 OrthographicCamera、透视相机 PerspectiveCamera、全景相机 CubeCamera 和 3D相机 StereoCamera。主要介绍正交相机和透视相机。

## ****3.1透视相机 PerspectiveCamera****

透视相机更接近人眼的观看效果，有“近大远小”的效果。

var camera = new THREE.PerspectiveCamera(fov, aspect, near, far);

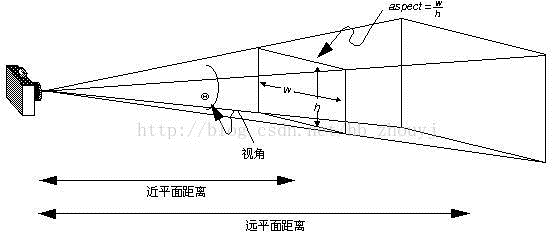
参数：

fov 视角 — 人差不多有180度的视角，但计算机显示器一般会选择一块儿较小的区域。对于游戏来说，大多数情况下会用60-90度左右的视角。推荐默认值：45。

aspect 长宽比 — 渲染区域的长宽比。一般会使用整个窗口作为输出界面。推荐默认值：window.innerWidth/window.innerHeight。

near 近面 — 相机开始渲染场景的地方，通常会设一个很小的值，从而可以看到所有物体。推荐默认值：0.1。

far 远面 — 相机结束渲染场景的地方，如果值太低，场景中的一部分可能不会被渲染；如果值太高，某些情况下会影响渲染效率。推荐默认值：1000。



属性：

.fov — 相机视椎体垂直视角，从下到上的观察角度。

.zoom — 获取或设置相机缩放因子。

.near, .far — 相机视椎体近裁剪面，远裁剪面。

.focus — 焦点，用于立体视觉和场深度效果的目标距离。

.aspect — 相机视椎体宽高比。

.view — 视椎体窗口规格或null。

.filmGauge — 用于较大坐标轴的胶片尺寸。默认为35毫米。

.filmOffset — 水平偏离中心偏移量。和 .filmGauge 同单位。

**方法**：

.getEffectiveFOV( )  — 返回考虑了缩放系数 .zoom 的当前视角，单位为角度。

.getFocalLength( ) — 返回当前和 .filmGauge 有关的 .fov 的焦距。

.getFilmWidth( ) — 返回胶片上图像的宽度。如果 .aspect >= 1(景观格式 landscape format)，结果等于 .filmGauge。

.getFilmHeight( ) — 返回胶片上图像的高度。如果 .aspect <=1(肖像格式 portrait format)，结果等于 .filmGauge。

.setFocalLength( focalLength ) — 设置当前和 .filmGauge 有关的 .fov 的焦距。默认为35mm。

.setViewOffset( fullWidth, fullHeight, x, y, width, height ) — 同 OrthographicCamera。

.clearViewOffset( ), .updateProjectionMatrix( ), .clone( ), .toJSON — 同 OrthographicCamera。

**demo:camera-perspectiveCamera.html**

## ****3.2正交相机 OrthographicCamera****

正交相机重在表现物体的实际尺寸，没有近大远小的效果；一般是用在制图、建模上面。

var camera = new THREE.OrthographicCamera(left, right, top, bottom, near, far);

参数：

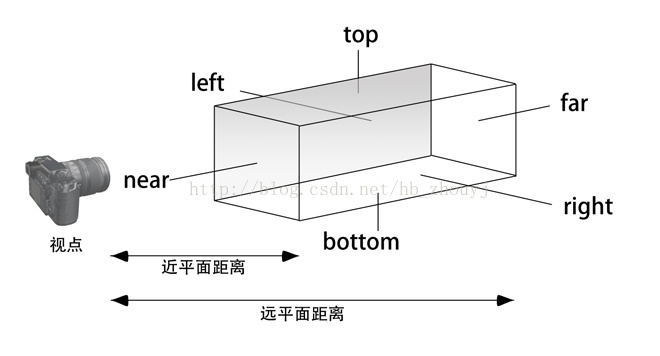
left：左平面距离相机中心点的垂直距离。从图中可以看出，左平面是屏幕里面的那个平面。

right：右平面距离相机中心点的垂直距离。从图中可以看出，右平面是屏幕稍微外面一点的那个平面。

top：顶平面距离相机中心点的垂直距离。上图中的顶平面，是长方体头朝天的平面。

bottom：底平面距离相机中心点的垂直距离。底平面是头朝地的平面。

near：近平面距离相机中心点的垂直距离。近平面是左边竖着的那个平面。

far：远平面距离相机中心点的垂直距离。远平面是右边竖着的那个平面。

**属性**：

.zoom — 获取和设置相机缩放因子。

.left, .right, .top, .bottom, .near, .far — 相机视椎体左面,右面,上面,下面,前面,后面。

**方法**：

.setViewOffset( fullWidth, fullHeight, x, y, width, height )

     fullWidth — 多视图设置的全宽

     fullHeight — 多视图设置的全高

     x — 副摄像头的水平偏移

     y — 副摄像头的垂直偏移

     width — 副摄像头的宽度

     height — 副摄像头的高度

 该方法用于在一个较大的视椎体中设置视图偏移。这对于多窗口或多监视器/多机设置是有用的。

 注意，没有任何理由在一个网格中的显示屏必须具备同样的尺寸。

.clearViewOffset( ) — 清除视图偏移。

.updateProjectionMatrix( ) — 更新相机投影矩阵，必须在参数发生变化后调用。

.clone( ) — 返回一个 OrthographicCamera 对象的克隆。

.toJSON( ) — 把相机数据转换成JSON格式。

**demo:camera-orthographicCamera.html**

4.相机移动

另外相机还有常用的函数

camera.position //位置

camera.lookAt//看向

4.1移动相机——第一人称控件

4.2移动相机——飞行控件

4.3移动相机——翻滚控件

4.4移动相机——轨迹球控件

4.5移动相机——轨道控件

4.6移动相机——路径控件

# 5.光源

1. 环境光（AmbientLight它的颜色会添加到整个场景和所有对象的当前颜色上），
2. 点光源（PointLight空间中的一点，朝所有的方向发射光线），
3. 聚光灯光源（SpotLight这种光源有聚光的效果，类似于台灯，吊灯，手电筒），
4. 方向光（DirectionalLight也称无限光，从这种光源发出的光线可以看作是平行的，例如太阳光）
5. 半球光光源（HemisphereLight，可以为室内场景创建更加自然的光照效果，模拟反光面和光线微弱的天气）
6. 面光源（AreaLight使用这种光源可以指定散发光线的平面，而不是空间的一个点）
7. 镜头炫光（LensFlare这不是一种光源，但是通过该炫光可以为场景中的光源添加炫目的效果）

5.1 环境光——AmbientLight光源的颜色会影响到整个场景，

环境光没有特定的光源，是模拟漫反射的一种光源，因此不需要指定位置

它能将灯光均匀地照射在场景中每个物体上面，一般情况下用来弱化阴影或者添加一些颜色到环境中，因此不能将AmbientLight作为场景中的唯一光源。

注意一下AmbientLight会将颜色应用于整个场景，因此使用AmbientLight的时候，用色应该尽量的保守，如果颜色过于明亮，画面颜色就会饱和

var ambiColor = "#0c0c0c";

var ambientLight = new THREE.AmbientLight(ambiColor);

scene.add(ambientLight);

//环境光不需要设置光源的位置等信息，只需要一个颜色即可，这个颜色就是漫反射到所有对象上的颜色

还应该将物体材质换为对光照有反应的材质

* **MeshBasicMaterial**：对光照无感，给几何体一种简单的颜色或显示线框
* **MeshLambertMaterial**：这种材质对光照有反应，用于创建暗淡的不发光的物体
* **MeshPhongMaterial**：这种材质对光照也有反应，用于创建金属类明亮的物体

demo: AmbientLight.html

5.2 点光源——PointLight

点光源是一种单点发光，照射全部方向的光源，例如生活中的照明弹就属于点光源，由于点光源是向所有方向发射光线，因此不会产生阴影，

它有颜色color，强度intensity，距离distance，位置position，是否可见visible等几个属性，

PointLight光源的distance属性决定的是光线可以照射多远，值为0时，表示光线的亮度不会随着距离的增加而递减

var pointColor = "#ccffcc";

var pointLight = new THREE.PointLight(pointColor);

pointLight.distance = 100;//距离，决定了光线可以照射多远

pointLight.intensity = 1;//强度

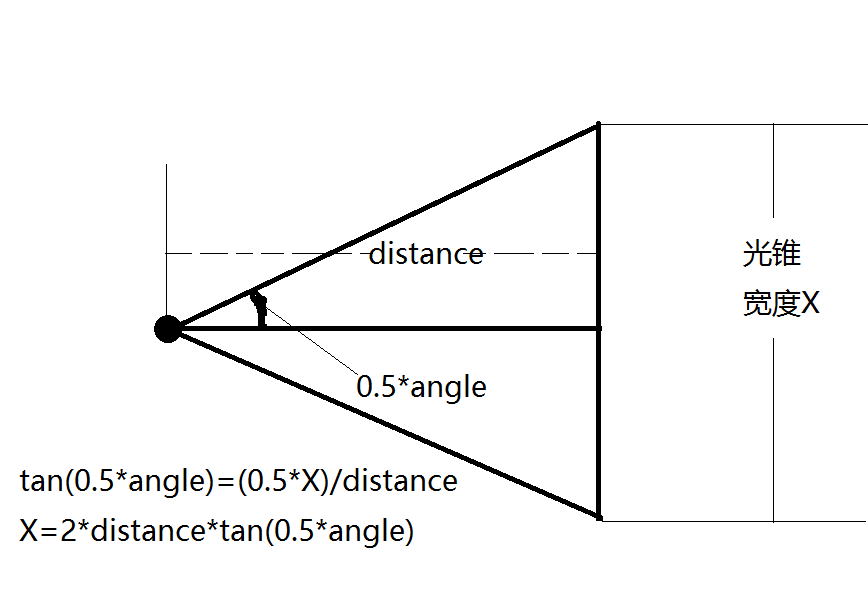
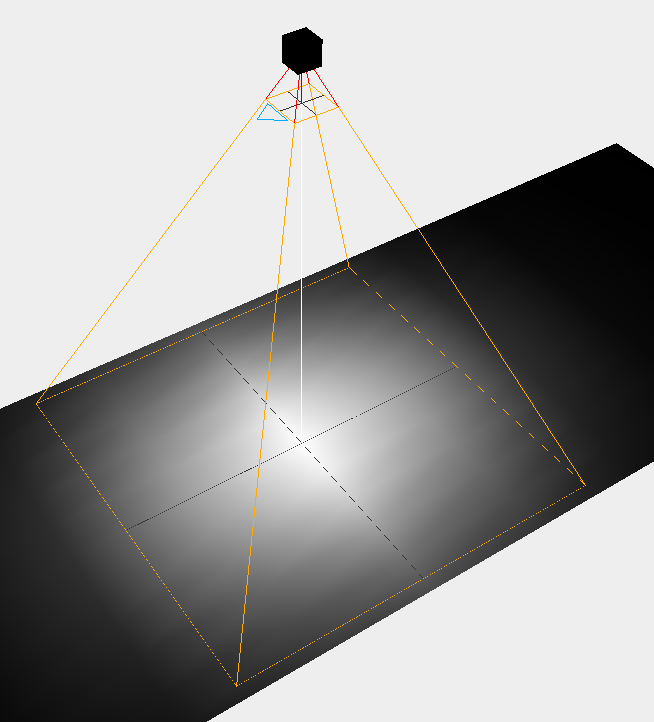
scene.add(pointLight);

demo: PointLight.html

5.3 聚光灯光源——SpotLight

聚光灯光源有一种锥形的效果，例如日常生活中的手电筒，灯笼等,该光源具有下面的属性

|  |  |
| --- | --- |
| 属性 | 描述 |
| castShadow（投影） | 设置为true，则该光源可以产生阴影 |
| shadowCameraNear | 从距离光源的哪一点开始生成阴影 |
| shadowCameraFar | 从距离光源哪一点开始停止生成阴影 |
| shadowCameraFov | 形成阴影的视场 |
| target | 决定了光源的方向，如果target属性的值出现移动的情况，那么光源也会跟着该target进行移动，该值必须为Object3D类型，因此可以是geom，也可以是具体的某一个点，这个代码中会有体现 |
| angle | 光源照射出来的光柱有多宽 |
| exponent | 光强衰减指数，距离光源越远，光照强度越弱，但是减弱的速度是由该值控制 |
| shadowCameraVisible | 通过设置此值，可以看到光源在哪里，以及如何产生阴影，即该值主要用于调试 |
| shadowDarkness | 阴影强度，即阴影有多黑，场景渲染后该值不可以再修改了 |
| shadowMapWidth | 多少像素用来生成阴影，如果阴影的边缘看起来不够光滑参差不齐，那么可以考虑加大该值，该值同shadowDarkness，场景渲染后不可修改 |
| distance | 距离target的距离 |

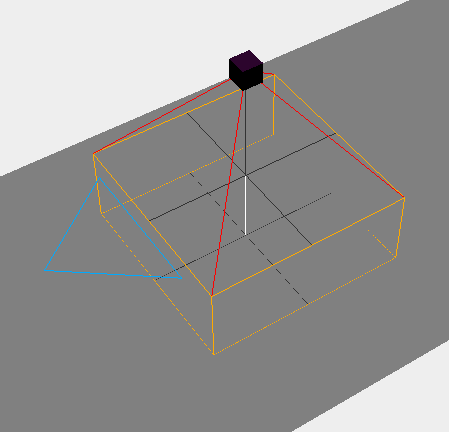
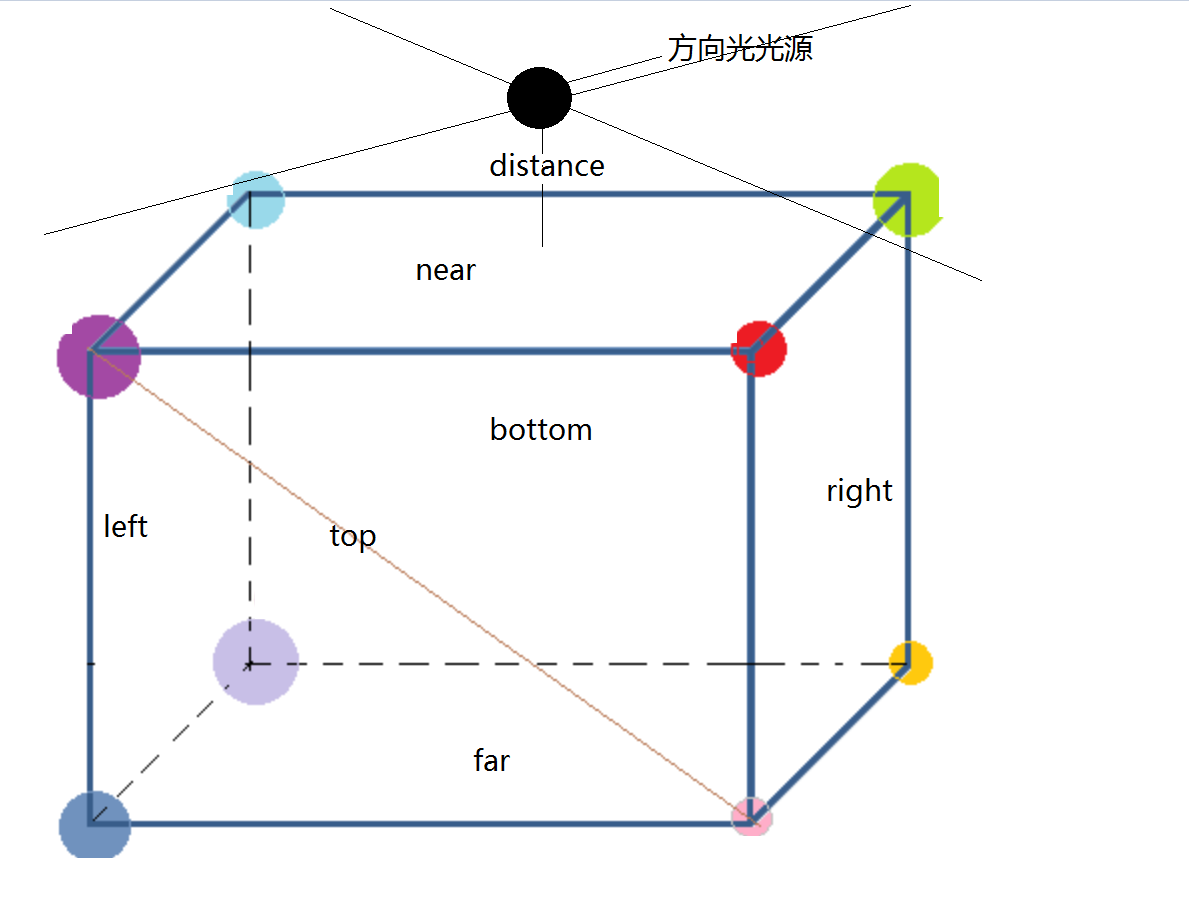
 

demo: SpotLight.html

## 5.4 方向光——DirectionalLight也称无限光（类似于太阳光的一种光源）

该光源可以看做是距离很远很远的光源，以至于该光源所发出的所有光线都是相互平行的，方向光光源的一个范例就是太阳，方向光光源不像聚焦光那样离目标越远越暗淡，被方向光光源照亮的整个区域接收到的光强是一样的

方向光光源的shadowCameraNear，far，left，right,top，bottom六个属性构成了一个方块的范围, 在这个方块的范围内的所有对象都可以投影或者接收投影，包围对象的方块范围定义的越紧密，投影的效果越好



demo: DirectionalLight.html

5.5 半球光源----HemisphereLight

这种光源可以为室外场景创建更加自然的光照效果

我们模拟室外光照的时候，可以使用方向光源来模拟太阳，再添加一个环境光源，为场景添加基础色, 但是这样看起来不太自然，因为室外的光并不都是来自于上方，很多是来自于空气的散射和地面的反射，以及其他物体的反射,所以在使用方向光源来模拟太阳的情况下再添加一个半球光就自然多了.

var hemiLight = new THREE.HemisphereLight(天空的反光颜色,地面的反光颜色,光的强度);

demo: HemisphereLight.html

5.6 平面光光源------AreaLight平面光光源可以定义为一个发光的矩形

1. 由于该光源是THREE.js的扩展，需要引入扩展库
2. 平面光源是一种很复杂的光源，因此不能再使用WebGLRenderer对象了，该对象会造成严重的性能损失
3. 因此在处理复杂光源或者多个光源的时候，使用WebGL的延迟渲染WebGLDeferredRenderer

demo: AreaLight.html

5.7 镜头眩光----LensFlare

当我们直接朝着太阳拍照时就会出现镜头眩光，对于游戏或者三维图像来说，镜头眩光会使得场景看起来更真实

demo: LensFlare.html

总结

1. 环境光源没有位置概念，会将颜色应用到场景的每一个物体上，主要作用是弱化阴影，给场景添加颜色
2. 点光源类似于照明弹，朝所有的方向发光，因此不产生阴影
3. 聚光灯光源类似于手电筒，形成锥形的光束，随着距离的增加而变弱，可以设定生成阴影
4. 方向光光源类似于太阳，从很远的地方发出的平行光束，距离越远，衰减的越多
5. 想要一个自然的室外效果，除了添加环境光弱化阴影，添加聚光灯为场景增加光线，还需要使用半球光光源将天空和空气以及地面的散射计算进去，使得更自然，更真实
6. 平面光光源定义了一个发光的发光体，需要使用webgl的延迟渲染机制
7. 眩光效果，在有太阳的时候使用眩光光源，会使得场景更真实

# 6.几何体

简单几何体

new THREE.CubeGeometry(长, 宽, 高, 长的分割, 宽的分割, 高的分割); // 立方体

new THREE.PlanGeometry(长,宽, 长的分割, 宽的分割); // 平面

new THREE.SphereGeometry(半径, 经度切片, 纬度分割, 经度分割, 经度跨过, 纬度开始, 纬度跨过); // 球体

new THREE.CircleGeometry(半径, 切片数, 开始, 跨过角度); // 圆形

new THREE.CylinderGeometry(顶部面积, 底部面积, 高, 圆分割, 高分割, 是否没有顶面和底面); // 圆台

new THREE.TetrahedronGeometry(半径, 细节); // 正四边形

new THREE.OctahedronGeometry(半径, 细节); // 正八边形

new THREE.IconsahedronGeometry(半径, 细节);  // 正十二边形

new THREE.TorusGeometry(半径, 管道半径, 纬度分割, 经度分割, 圆环面的弧度);  // 圆环面

# 7.材质

# 8.相机移动

# 9.加载和使用纹理

# 10.物理效果

# 11.粒子效果

# 12.高级几何体