## 1 世界有了光，就不在黑暗

宇宙间的物体有的是发光的，有的是不发光的，我们把发光的物体叫做光源。太阳、电灯、燃烧着的蜡烛等都是光源。

在Threejs的世界里，有了光，就不会在黑暗。

## 2 Threejs中的各种光源

作为3D技术的发展趋势，浏览器端3D技术越来越被一些技术公司重视。由此，Threejs非常注重3D渲染效果的真实性，对渲染真实性来说，使用光源是比不可少的技巧。Threejs，在光源方面提供了多种光源供选择。

### **1、 光源基类**

在Threejs中，光源用Light表示，它是所有光源的基类。它的构造函数是：

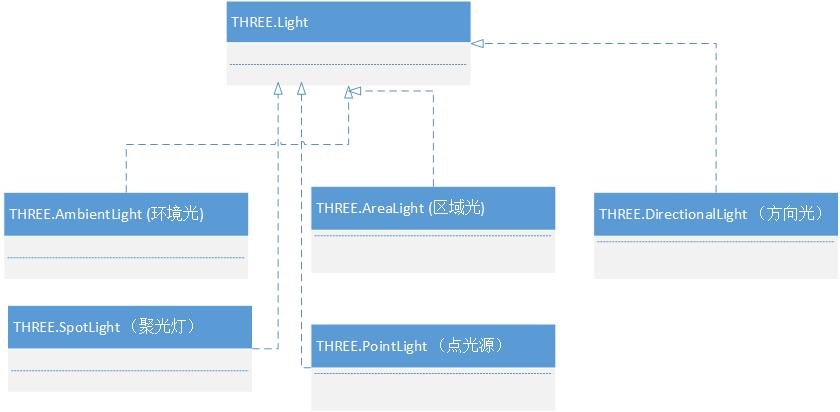
THREE.Light ( hex )

它有一个参数hex，接受一个16进制的颜色值。例如要定义一种红色的光源，我们可以这样来定义：

Var redLight = new THREE.Light(0xFF0000);

### **2、 由基类派生出来的其他种类光源**

THREE.Light只是其他所有光源的基类，要让光源除了具有颜色的特性之外，我们需要其他光源。看看，下面的类图，是目前光源的继承结构。



可以看出，所有的具体光源都继承与THREE.Light类。下面我们来具体看一下，其他光源。

## 3 环境光

环境光是经过多次反射而来的光称为环境光，无法确定其最初的方向。环境光是一种无处不在的光。环境光源放出的光线被认为来自任何方向。因此，当你仅为场景指定环境光时，所有的物体无论法向量如何，都将表现为同样的明暗程度。 （这是因为，反射光可以从各个方向进入您的眼睛）

环境光用THREE.AmbientLight来表示，它的构造函数如下所示：

THREE.AmbientLight( hex )

它仍然接受一个16进制的颜色值，作为光源的颜色。环境光将照射场景中的所有物体，让物体显示出某种颜色。环境光的使用例子如下所示：

var light = new THREE.AmbientLight( 0xff0000 );

scene.add( light );

只需要将光源加入场景，场景就能够通过光源渲染出好的效果来了。

## 4 点光源

点光源：由这种光源放出的光线来自同一点，且方向辐射自四面八方。例如蜡烛放出的光，萤火虫放出的光。

点光源用PointLight来表示，它的构造函数如下所示：

PointLight( color, intensity, distance )

这个类的参数稍微复杂一些，我们花点时间来解释一下：

Color：光的颜色

Intensity：光的强度，默认是1.0,就是说是100%强度的灯光，

distance：光的距离，从光源所在的位置，经过distance这段距离之后，光的强度将从Intensity衰减为0。 默认情况下，这个值为0.0，表示光源强度不衰减。

## 5 聚光灯

聚光灯：这种光源的光线从一个锥体中射出，在被照射的物体上产生聚光的效果。使用这种光源需要指定光的射出方向以及锥体的顶角α。聚光灯示例如图所示：



聚光灯的构造函数是：

THREE.SpotLight( hex, intensity, distance, angle, exponent )

函数的参数如下所示：

Hex：聚光灯发出的颜色，如0xFFFFFF

Intensity：光源的强度，默认是1.0，如果为0.5，则强度是一半，意思是颜色会淡一些。和上面点光源一样。

Distance：光线的强度，从最大值衰减到0，需要的距离。 默认为0，表示光不衰减，如果非0，则表示从光源的位置到Distance的距离，光都在线性衰减。到离光源距离Distance时，光源强度为0.

Angle：聚光灯着色的角度，用弧度作为单位，这个角度是和光源的方向形成的角度。

exponent：光源模型中，衰减的一个参数，越大衰减约快。

## 6 材质与光源的关系

材质与光源有什么关系，这是一个容易傻傻分不清的问题。在没有深入讲解前，我们只能说它们是相互联系，相互依托的关系。

我们会在后面的章节专门来解释什么是材质，不过这里也需要简单的给你介绍一下。

### **1、 材质的真相**

材质是啥子（四川话），材质就是物体的质地。我们可以用撤分文字的方法来理解。材质就是材料和质感的完美结合。

如果你还不理解，那么看看下面我引用的这段话：

在渲染程序中，它是表面各可视属性的结合，这些可视属性是指表面的色彩、纹理、光滑度、透明度、反射率、折射率、发光度等。正是有了这些属性，才能让我们识别三维中的模型是什么做成的，也正是有了这些属性，我们计算机三维的虚拟世界才会和真实世界一样缤纷多彩。

这就是材质的真相吗？答案是否定的。不要奇怪，我们必须仔细分析产生不同材质的原因，才能让我们更好的把握质感。那么，材质的真相到底是什么呢？仍然是光，离开光材质是无法体现的。举例来说，借助夜晚微弱的天空光，我们往往很难分辨物体的材质，因为他们很多都表现出黑色，我们难以区分是铝合金，还是塑料的。而在正常的照明条件下，则很容易分辨。另外，在彩色光源的照射下，我们也很难分辨物体表面的颜色，在白色光源的照射下则很容易。这种情况表明了物体的材质与光的微妙关系。下面，我们将具体分析两者间的相互作用。

首先，我们来看一些例子。这些例子是一系类的，掌握一个，我们就印下了一个脚

印。

## 7 脚印一：不带任何光源的物体

我们首先在屏幕上画一个物体，不带任何的光源，定义物体的颜色为黑色，其值为0x000000，定义材质如下：

var material = new THREE.MeshLambertMaterial( { color:0x000000} ); // 这是兰伯特材质，材质中的一种

先看看最终的运行截图，如下所示：



由这幅图得出结论，当没有任何光源的时候，最终的颜色将是材质的颜色。但是这个结论目前来说，并没有依据。

代码如下，你可以在cube0.html中发现。

|  |
| --- |
| <-!DOCTYPE html> |
| <-html> |
| <-head> |
| <-meta charset="UTF-8"> |
| <-title>Three框架<-/title> |
| <-script src="js/Three.js" data-ke-src="js/Three.js"><-/script> |
| <-style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| <-/style> |
| <-script> |
| var renderer; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColorHex(0xFFFFFF, 1.0); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 600; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
| // A begin |
| var light; |
| function initLight() { |
| } |
| // A end |
| var cube; |
| function initObject() { |
| var geometry = new THREE.CubeGeometry( 200, 100, 50,4,4); |
| // B begin |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| // B end |
| var mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| renderer.clear(); |
| renderer.render(scene, camera); |
| } |
|  |
| <-/script> |
| <-/head> |
|  |
| <-body data-ke-onload="threeStart();"> |
| <-div id="canvas-frame"><-/div> |
| <-/body> |
| <-/html> |

现在我们来解析一下：

1、 在A处，关于灯光的代码，什么也没有做。也就是Threejs中没有添加任何灯光。

2、 在B处，我们使用了兰伯特材质，并将这种材质赋予了黑色，所以，你才会发现最后的效果是黑色。如果，我们把材质颜色设置为红色，那么物体是不是就会显示红色呢？

答案是否定的，这是因为，在场景中没有任何光源的情况下，物体不能反射光源到人的眼里，所以物体应该是黑色的。这与物体的材质颜色几乎没有关系。打个比方，在月高风黑夜，伸手不见五指的夜晚，一群穿着彩衣的美女在你面前跳舞，你能分辨出他们是穿的彩色衣服吗？不能。

结论：当没有任何光源的时候，最终的颜色将是黑色，无论材质是什么颜色。

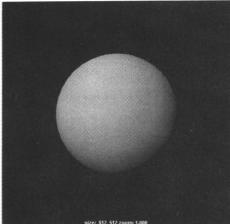
## 8 脚印二：兰伯特材质与光源

最常见的材质之一就是Lambert材质，这是在灰暗的或不光滑的表面产生均匀散射而形成的材质类型。比如一张纸就是Lambert表面。 首先它粗糙不均匀，不会产生镜面效果。我们在阅读书籍的时候，没有发现书上一点亮，一点不亮吧，它非常均匀，这就是兰伯特材质。

有的朋友觉得纸不粗糙啊，你怎么说它粗糙吗？人的肉眼是不好分辨出来，它粗不粗糙的。

Lambert材质表面会在所有方向上均匀地散射灯光，这就会使颜色看上去比较均匀。想想一张纸，无论什么颜色，是不是纸的各个部分颜色都比较均匀呢。

Lambert材质的图例如下所示：



Lambert材质会受环境光的影响，呈现环境光的颜色，与材质本身颜色关系不大。

我们现在来做一个例子

例子：红色环境光照射下的长方体，它用的是黑色的兰伯特材质。效果如下图：



我们来看看代码，你可以在cube.html中找到它，这里不存在环保问题，所以，我把所有代码都列出来了。

[View Raw Code](http://hewebgl.com/article/getarticle/60)[?](http://www.oriontransfer.co.nz/software/jquery-syntax" \t "http://hewebgl.com/article/getarticle/oriontransfer)

|  |
| --- |
| <-!DOCTYPE html> |
| <-html> |
| <-head> |
| <-meta charset="UTF-8"> |
| <-title>Three框架<-/title> |
| <-script src="js/Three.js" data-ke-src="js/Three.js"><-/script> |
| <-style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| <-/style> |
| <-script> |
| var renderer; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColorHex(0xFFFFFF, 1.0); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 600; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
| // A begin |
| var light; |
| function initLight() { |
| light = new THREE.AmbientLight(0xFF0000); |
| light.position.set(100, 100, 200); |
| scene.add(light); |
|  |
| } |
| // A end |
| var cube; |
| function initObject() { |
| var geometry = new THREE.CubeGeometry( 200, 100, 50,4,4); |
| // B begin |
| var material = new THREE.MeshLambertMaterial( { color:0x000000} ); |
| // B end |
| var mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| renderer.clear(); |
| renderer.render(scene, camera); |
| } |
|  |
| <-/script> |
| <-/head> |
|  |
| <-body data-ke-onload="threeStart();"> |
| <-div id="canvas-frame"><-/div> |
| <-/body> |
| <-/html> |

好了，我们来分析一下这段代码。

1、 在A处，我们设置了一个红色的环境光，并把它放在了一个位置上。

2、 在B处，我们使用了黑色的兰伯特材质。

最后整个效果中，长方体呈现的是红色。我们要说的是，长方体呈现什么颜色目前只与光源有关，无论兰伯特材质使用的是什么颜色。不信，你可以试一试，把材质换成其他颜色。不过，一定注意，这条理论只对兰伯特材质有效果。

我们现在一直在使用环境光，从环境光的构造函数来看，它只有颜色，其位置对场景中的物体并没有影响，因为他是均匀的反射到物体的表面的。

## 9 脚印三：环境光对物体的影响

环境光就是在场景中无处不在的光，它对物体的影响是均匀的，也就是无论你从物体的那个角度观察，物体的颜色都是一样的，这就是伟大的环境光。

你可以把环境光放在任何一个位置，它的光线是不会衰减的，是永恒的某个强度的一种光源。

## 5 脚印四：方向光（平行光）

平行光又称为方向光（Directional Light），是一组没有衰减的平行的光线，类似太阳光的效果。

方向光的模型如图：



方向光的构造函数如下所示：

THREE.DirectionalLight = function ( hex, intensity )

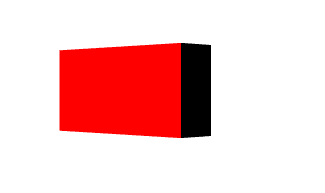
其参数如下：

Hex：关系的颜色，用16进制表示

Intensity：光线的强度，默认为1。因为RGB的三个值均在0~255之间，不能反映出光照的强度变化，光照越强，物体表面就更明亮。它的取值范围是0到1。如果为0，表示光线基本没什么作用，那么物体就会显示为黑色。呆会你可以尝试来更改这个参数，看看实际的效果

我们来看一个方向光的例子：

一个红色的方向光，把它放在（0，0,1）的位置，密度为1，照射在一个长方体中。效果如下图所示：



你可以再cube1.html中，看到它的完整代码：

[View Raw Code](http://hewebgl.com/article/getarticle/61)[?](http://www.oriontransfer.co.nz/software/jquery-syntax" \t "http://hewebgl.com/article/getarticle/oriontransfer)

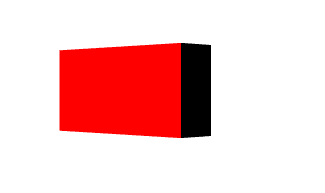
|  |
| --- |
| <-!DOCTYPE html> |
| <-html> |
| <-head> |
| <-meta charset="UTF-8"> |
| <-title>Three框架<-/title> |
| <-script src="js/Three.js" data-ke-src="js/Three.js"><-/script> |
| <-style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| <-/style> |
| <-script> |
| var renderer; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColorHex(0xFFFFFF, 1.0); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 600; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| // A begin |
| light = new THREE.DirectionalLight(0xFF0000,1); |
| light.position.set(0, 0,1); |
| scene.add(light); |
| // A end |
| } |
|  |
| var cube; |
| function initObject() { |
| var geometry = new THREE.CubeGeometry( 200, 100, 50,4,4); |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| var mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
| } |
|  |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| renderer.clear(); |
| renderer.render(scene, camera); |
| } |
|  |
| <-/script> |
| <-/head> |
|  |
| <-body data-ke-onload="threeStart();"> |
| <-div id="canvas-frame"><-/div> |
| <-/body> |
| <-/html> |

我们来分析一下上面的代码：

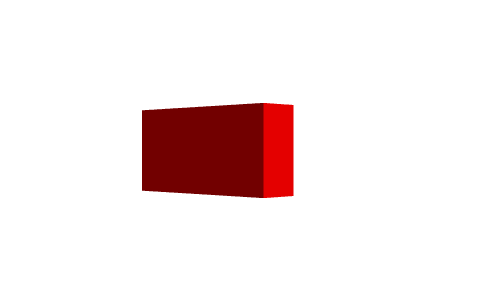
1、在A处，我们定义了一个红色的强度为1的方向光，它的位置为与（0,0,1）。现在你可以将强度值进行更改一下，例如把它分别改为0.2,0.4,0.6,0.8和1.0，请看看渲染的场景的变换。我敢保证，因为光线强度越来越大，所以红色从黑色、暗红、一直到鲜艳的红色了。

2、平行光有一个方向，它的方向是如何决定的呢？

方向由位置和原点（0,0,0）来决定，方向光只与方向有关，与离物体的远近无关。分别将平行光放到（0,0,100），（0,0,50），（0,0,25），（0,0,1），渲染的结果还是红色和黑色，见下图，颜色的深浅不与离物体的距离相关。



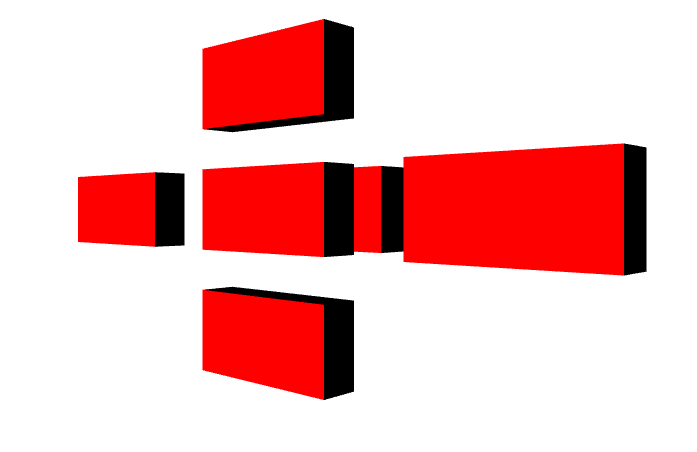
但是它与方向有关，如果，我们灯光的位置改为（1,0,0,5），那么效果如图所示：



请仔细领会这幅图的意思。

## 6 脚印五：增加几个物体，从宏观上看一下光源对物体的影响

现在，我们在场景中增加几个物体，来看看，光源对物体的影响。如图是添加了几个物体的截图。仍然是使用方向光。



代码如下，你可以再cube4.html中找到：

|  |
| --- |
| <-!DOCTYPE html> |
| <-html> |
| <-head> |
| <-meta charset="UTF-8"> |
| <-title>Three框架<-/title> |
| <-script src="js/Three.js" data-ke-src="js/Three.js"><-/script> |
| <-style type="text/css"> |
| div#canvas-frame { |
| border: none; |
| cursor: pointer; |
| width: 100%; |
| height: 600px; |
| background-color: #EEEEEE; |
| } |
|  |
| <-/style> |
| <-script> |
| var renderer; |
| function initThree() { |
| width = document.getElementById('canvas-frame').clientWidth; |
| height = document.getElementById('canvas-frame').clientHeight; |
| renderer = new THREE.WebGLRenderer({ |
| antialias : true |
| }); |
| renderer.setSize(width, height); |
| document.getElementById('canvas-frame').appendChild(renderer.domElement); |
| renderer.setClearColorHex(0xFFFFFF, 1.0); |
| } |
|  |
| var camera; |
| function initCamera() { |
| camera = new THREE.PerspectiveCamera(45, width / height, 1, 10000); |
| camera.position.x = 600; |
| camera.position.y = 0; |
| camera.position.z = 600; |
| camera.up.x = 0; |
| camera.up.y = 1; |
| camera.up.z = 0; |
| camera.lookAt({ |
| x : 0, |
| y : 0, |
| z : 0 |
| }); |
| } |
|  |
| var scene; |
| function initScene() { |
| scene = new THREE.Scene(); |
| } |
|  |
| var light; |
| function initLight() { |
| light = new THREE.DirectionalLight(0xFF0000); |
| light.position.set(0, 0,1); |
| scene.add(light); |
| } |
| // A begin |
| var cube; |
| function initObject() { |
| var geometry = new THREE.CubeGeometry( 200, 100, 50,4,4); |
| var material = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| var mesh = new THREE.Mesh( geometry,material); |
| mesh.position = new THREE.Vector3(0,0,0); |
| scene.add(mesh); |
|  |
| var geometry2 = new THREE.CubeGeometry( 200, 100, 50,4,4); |
| var material2 = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| var mesh2 = new THREE.Mesh( geometry2,material2); |
| mesh2.position = new THREE.Vector3(-300,0,0); |
| scene.add(mesh2); |
|  |
| var geometry3 = new THREE.CubeGeometry( 200, 100, 50,4,4); |
| var material3 = new THREE.MeshLambertMaterial( { color:0xFFFFFF} ); |
| var mesh3 = new THREE.Mesh( geometry3,material3); |
| mesh3.position = new THREE.Vector3(0,-150,0); |
| scene.add(mesh3); |
|  |
| var mesh4 = new THREE.Mesh( geometry3,material3); |
| mesh4.position = new THREE.Vector3(0,150,0); |
| scene.add(mesh4); |
|  |
| var mesh5 = new THREE.Mesh( geometry3,material3); |
| mesh5.position = new THREE.Vector3(300,0,0); |
| scene.add(mesh5); |
|  |
| var mesh6 = new THREE.Mesh( geometry3,material3); |
| mesh6.position = new THREE.Vector3(0,0,-100); |
| scene.add(mesh6); |
| } |
| // A end |
| function threeStart() { |
| initThree(); |
| initCamera(); |
| initScene(); |
| initLight(); |
| initObject(); |
| renderer.clear(); |
| renderer.render(scene, camera); |
| } |
|  |
| <-/script> |
| <-/head> |
|  |
| <-body data-ke-onload="threeStart();"> |
| <-div id="canvas-frame"><-/div> |
| <-/body> |
| <-/html> |

在A处，我们一共new了6个Mesh，并将每一个mesh放到了不同的位置，这样就生了上图的模样。这里并没有太多的技术含量，童鞋们只需要如法炮制就ok了。

## 6、环境光和方向光

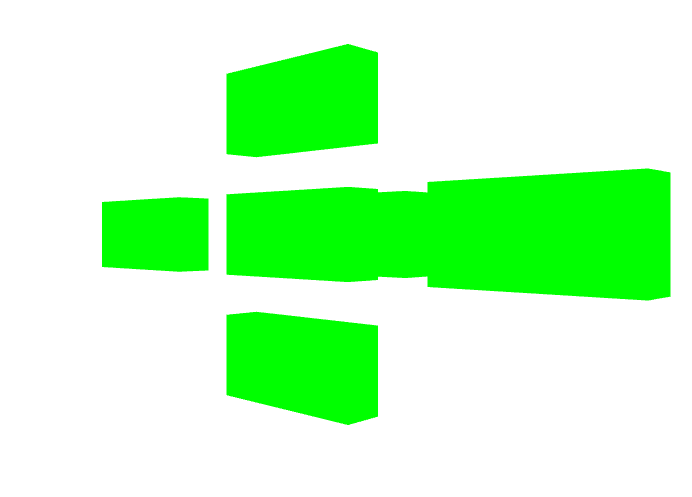
接下来，我们来看看多种光源同时存在于场景之中，对物体颜色的影响。

当环境光和方向光同时存在的时候，会出现怎么样的情况呢？可以把这种情况想成两种光源同时作用于物体，它产生的情况，和每种光源分别作用于物体，然后将两者的结果相加，是一样的效果。首先看看下面的代码：

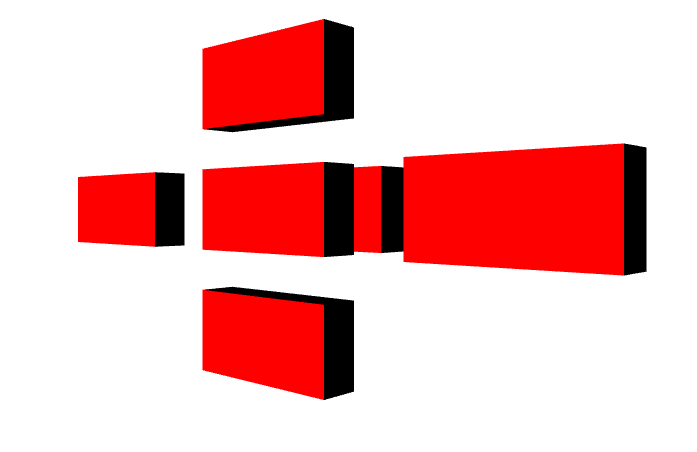
|  |
| --- |
| function initLight() { |
|  |
| light = new THREE.AmbientLight(0x00FF00); |
|  |
| light.position.set(100, 100, 200); |
|  |
| scene.add(light); |
|  |
| *// 方向光* |
|  |
| light = new THREE.DirectionalLight(0xFF0000); |
|  |
| light.position.set(0, 0,1); |
|  |
| scene.add(light); |
|  |
| } |
|  |

从代码上可以看出，环境光是绿色0x00FF00，方向光是红色0xFF0000，

我们来看看只有环境光，把方向光去掉的时候，渲染的结果是怎么样的：

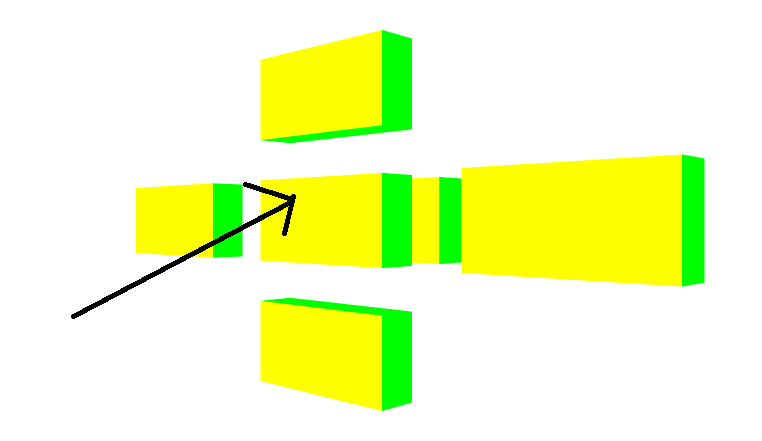


反过来，只有方向光的情况，没有环境光的时候，渲染的结果又会是怎么样呢？看看下图：



是的，总结一下，当方向光照射过来的时候，被照射的表面呈现光的颜色，而由于是方向光，没有照射到的表面，就呈现暗色，一般是黑色，表示没有任何光源照到该表面。

ok，好了，现在我们将环境光和方向光都加上，看看会出现什么效果，也会你已经猜到了效果，不过我还是不厌其烦的给你演示一次。



好了，我们马上来总结一下：

1.首先方向光，是如图箭头的方向着色到物体的。而环境光由于与位置没有关系，方向又是任何方向都可以照射的，所以我们不管光的方向。

2、图中绿色的部分，是由环境光造成的。由于方向光根本照射不到绿色的部分，所以，这部分只有环境光对其影响。

3、图中黄色的部分是由环境光和方向光共同作用而成的，其实是两种光源颜色的简单相加，

0x00FF00 + 0xFF0000 = 0xFFFF00，oxFFFF00 就是黄色。

## 7、点光源

点光源是理想化为质点的向四面八方发出光线的光源。点光源是抽象化了的物理概念，为了把物理问题的研究简单化。就像平时说的光滑平面，质点，无空气阻力一样，点光源在现实中也是不存在的，指的是从一个点向周围空间均匀发光的光源。

点光源的特点是发光部分为一个小圆面，近似一个点

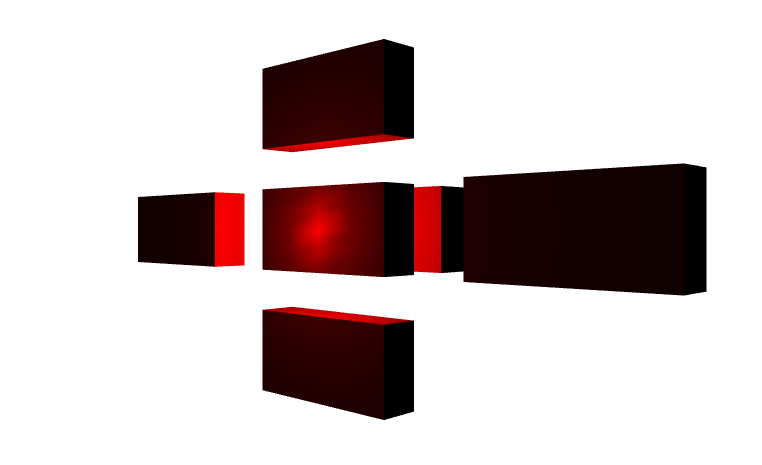
下面的例子介绍了怎么使用点光源：

light = new THREE.PointLight(0xFF0000);

light.position.set(0, 0,50);

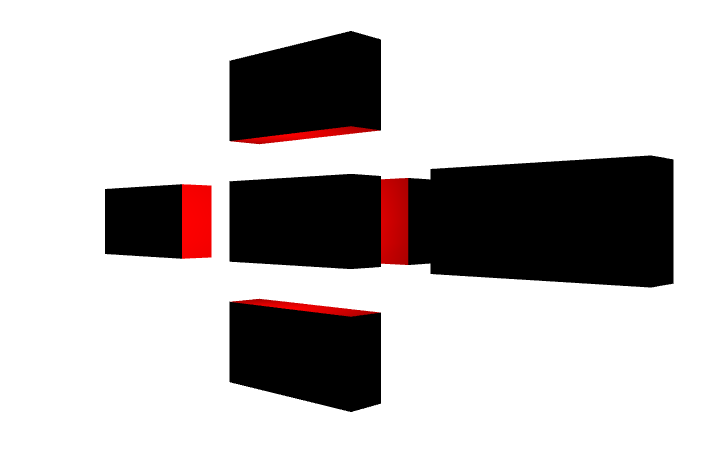
scene.add(light);

效果如下图：



点光源就是在一个点向周围发出的光，所以，你会看到照在物体上的光，有点像球的形状。改变点光源的位置，那么得到的效果图又会有一些区别。

将光源的位置改在(0, 0,25)，则刚好在一个长方体的边上，效果图如下所示：



比较上面两幅图，你会发现，第二幅图和第一幅被照射的位置是不一样的。第二幅图，由于刚好在中间的一个长方形的边上，所以被边挡住，只有长方体内部受到光源，而外部面没有受到光源的，所以呈现黑色。

从这里也反应出了，一个面分前后两个面的，只有被光源照射的那个面才能够被看到。

## 8、混合光源

将方向光和点光源混合使用。

效果如图所示：

