3D 数据可视化

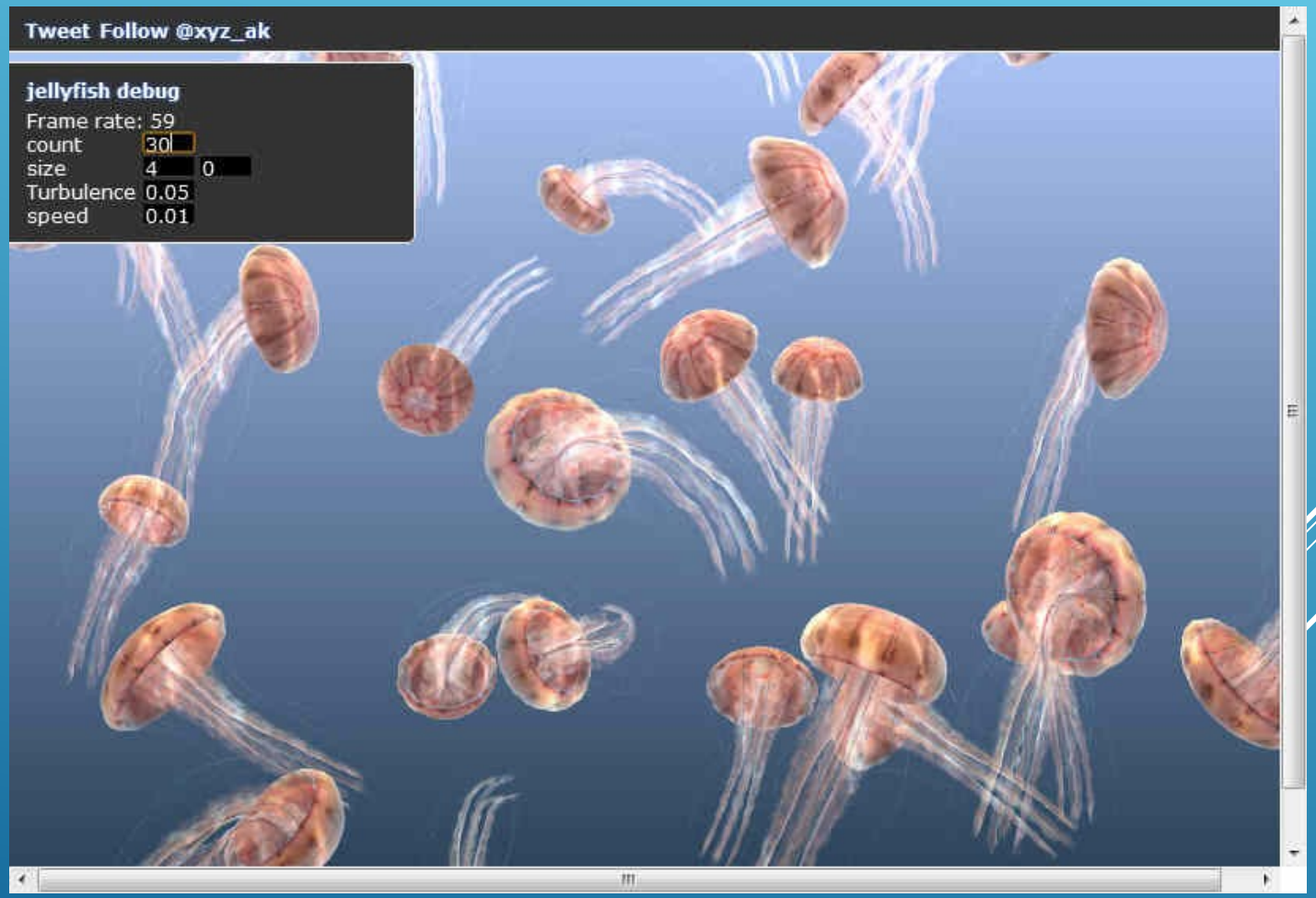
李春芳

2019.6.17.6.24

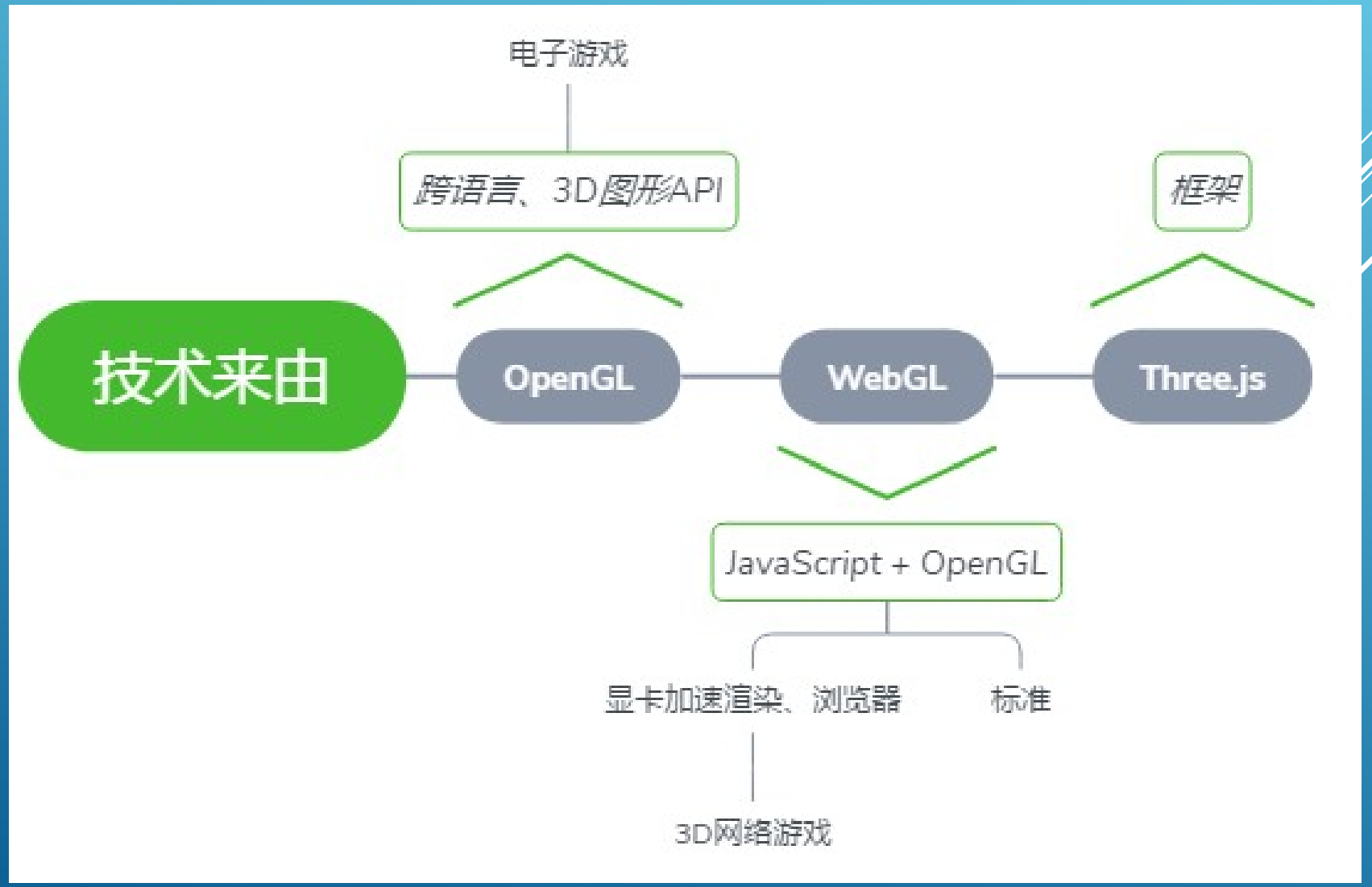
WEBGL 简介

（ http://aleksandarrodic.com/p/jellyfish/ ）

WebGL 是在浏览器中实现三维效果的一套规范。



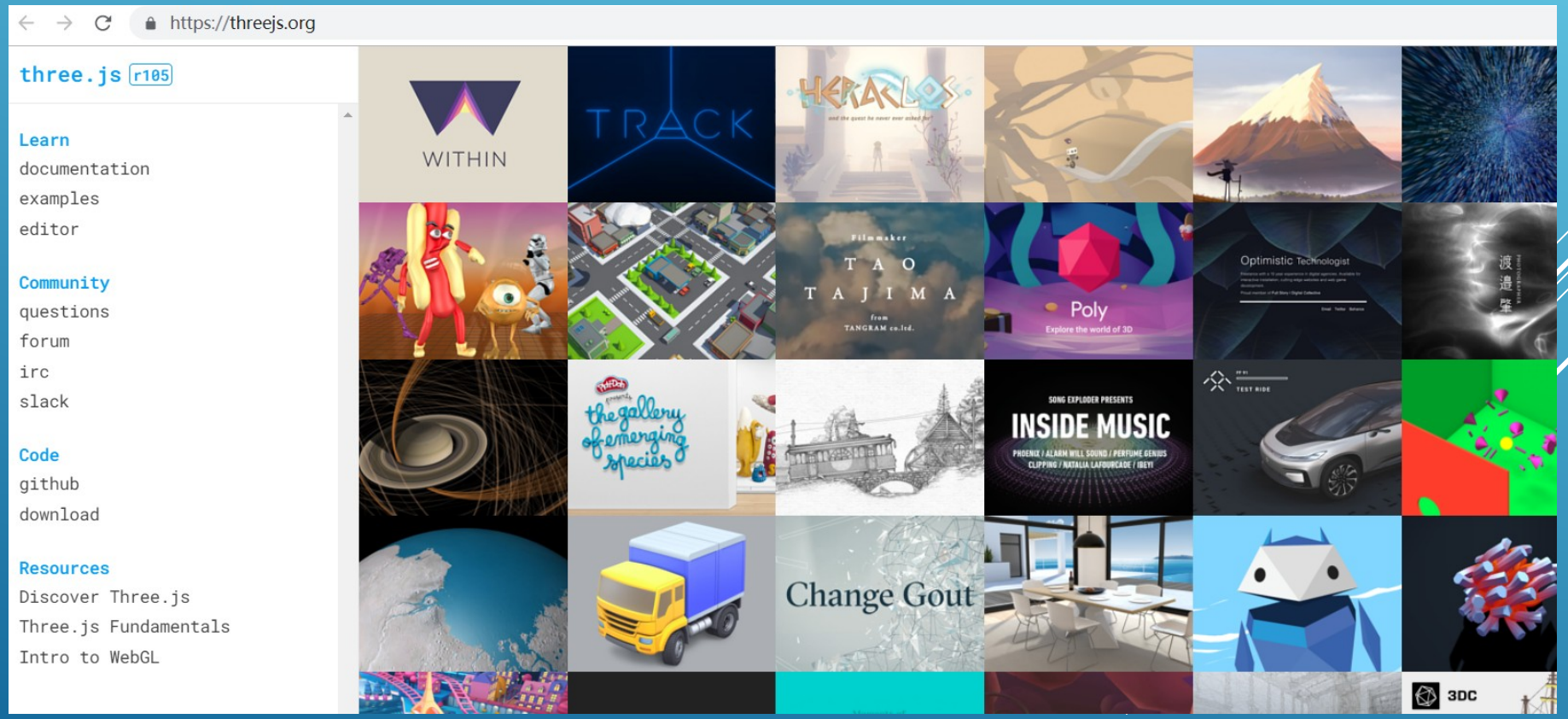
实现方法


电子游戏
跨语言、3D图形API
框架
技术来由
OpenGL
WebGL
Three.js
JavaScript + OpenGL
显卡加速渲染、浏览器
标准
3D网络游戏

可视化

- 写 3D 程序，最好是用 c++ ，保证效率
- javascript 的计算能力因为 google 的 V8 引 擎得到了迅猛的增强
- https://github.com/mrdoob/three.js

THREE.JS



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <style>canvas { width: 100%; height: 100% }</style>
    <script src="three.js"></script>
</head>
<body>
    <script>
        var scene = new THREE.Scene();

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

        var renderer = new THREE.WebGLRenderer();

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

        document.body.appendChild(renderer.domElement);
        var geometry = new THREE.CubeGeometry(1,1,1);
        var material = new THREE.MeshBasicMaterial({color: 0xFF0000});
        var cube = new THREE.Mesh(geometry, material);
        scene.add(cube);
        camera.position.z = 5;
        function render() {
            requestAnimationFrame(render);
            cube.rotation.x += 0.1;
            cube.rotation.y += 0.1;
            renderer.render(scene, camera);
        }
        render();
    </script>
</body>
</html>
```

HELLO3D.HTM

不安全 | jsbin.com/?html,output

New bin

JS Bin features »

Getting started

Keyboard Shortcuts

Exporting/importing gist

Textarea editor mode

Pro features »

Private bins

Dropbox backup

Vanity URLs

Upgrade to pro now

Blog »

The Return and The Refactor

Help »

Vanity URLs

About JS Bin

Donate to JS Bin ♥ »

Support JS Bin to keep the project open source & MIT for all

Follow @js_bin on twitter

By using JS Bin you agree to our legal terms

"Everyone should learn how to program a computer because it teaches you how to think" — Steve Jobs

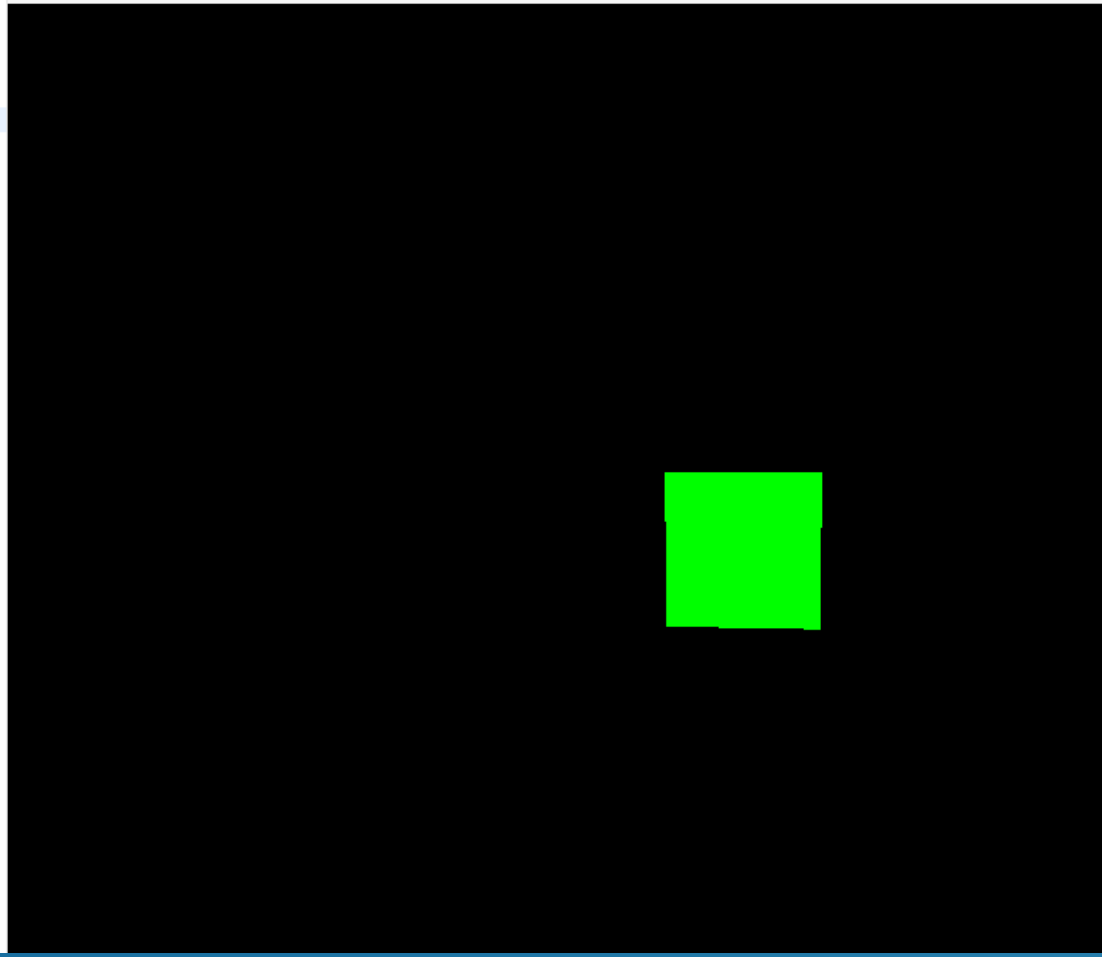
File ▾ Add library

HTML | CSS | JavaScript | Console | Output

HTML ▾

```
<html>
  <head>
    <title>My first three.js app</title>
    <style>
      body { margin: 0; }
      canvas { display: block; }
    </style>
  </head>
  <body>
    <script src="https://johnson2heng.github.io/three.js-demo/lib/three.js"></script>
    <script>
      var scene = new THREE.Scene();
      var camera = new THREE.PerspectiveCamera( 75, window.innerWidth / window.innerHeight, 0.1, 1000 );

      var renderer = new THREE.WebGLRenderer();
      renderer.setSize( window.innerWidth, window.innerHeight );
      document.body.appendChild( renderer.domElement );

      var geometry = new THREE.BoxGeometry();
      var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
      var cube = new THREE.Mesh( geometry, material );
      scene.add( cube );

      camera.position.z = 5;

      var animate = function () {
        requestAnimationFrame( animate );

        cube.rotation.x += 0.01;
        cube.rotation.y += 0.01;

        renderer.render( scene, camera );
      };

      animate();
    </script>
  </body>
</html>
```

Output

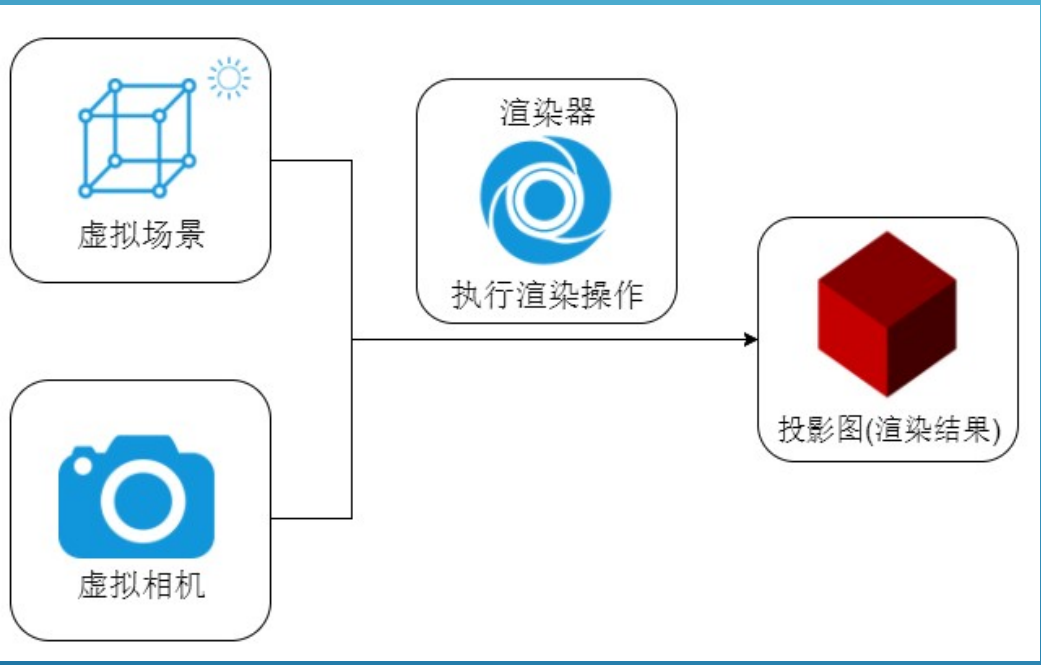


3D 图形

- 1- 需要 3 个组件
  - 场景（ scene ）
  - 相机（ camera ）：人的眼睛
  - 渲染器（ renderer ）

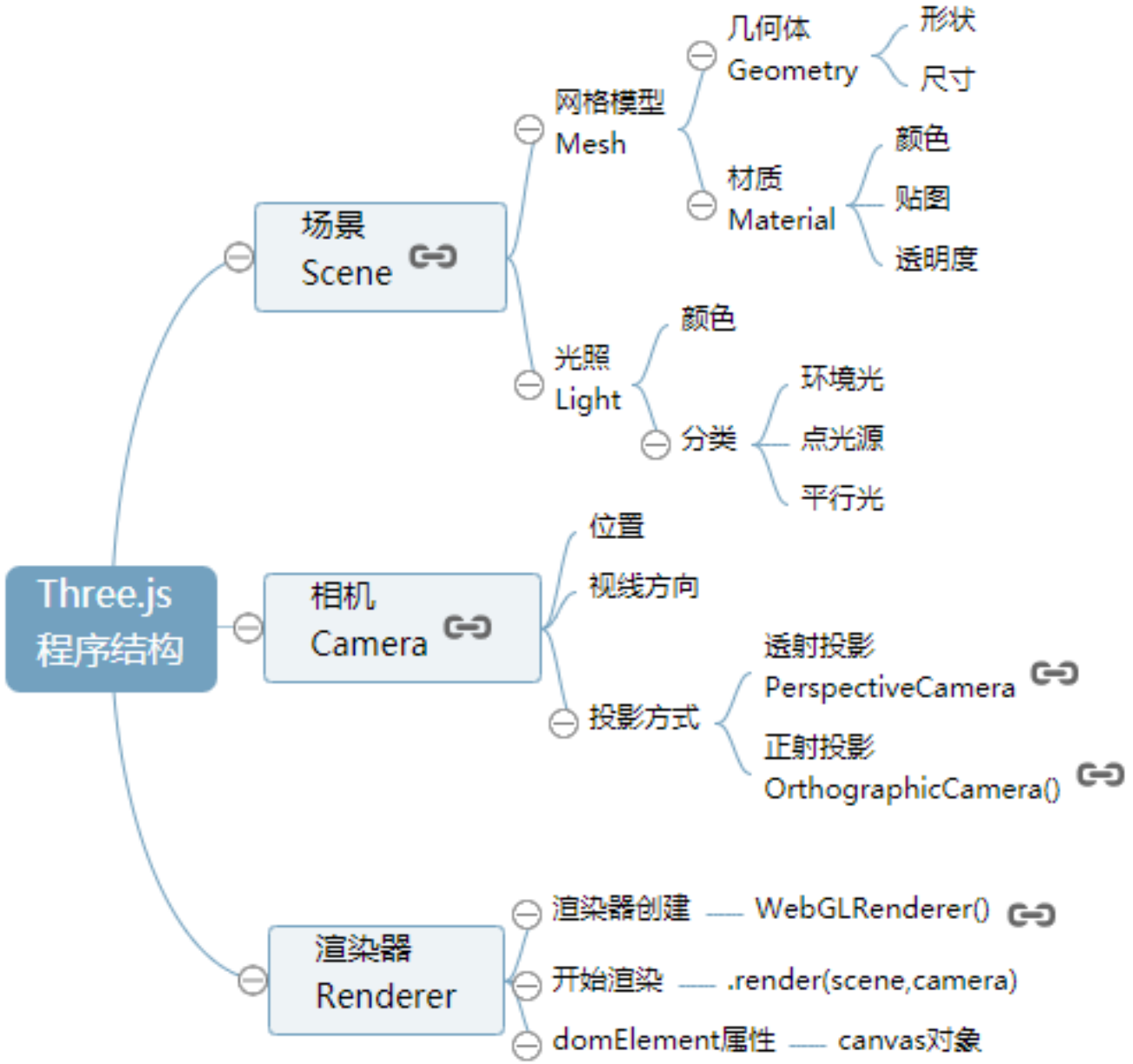
```
function render() {
    cube.rotation.x += 0.1;
    cube.rotation.y += 0.1;
    renderer.render(scene, camera);
    requestAnimationFrame(render);
}
```

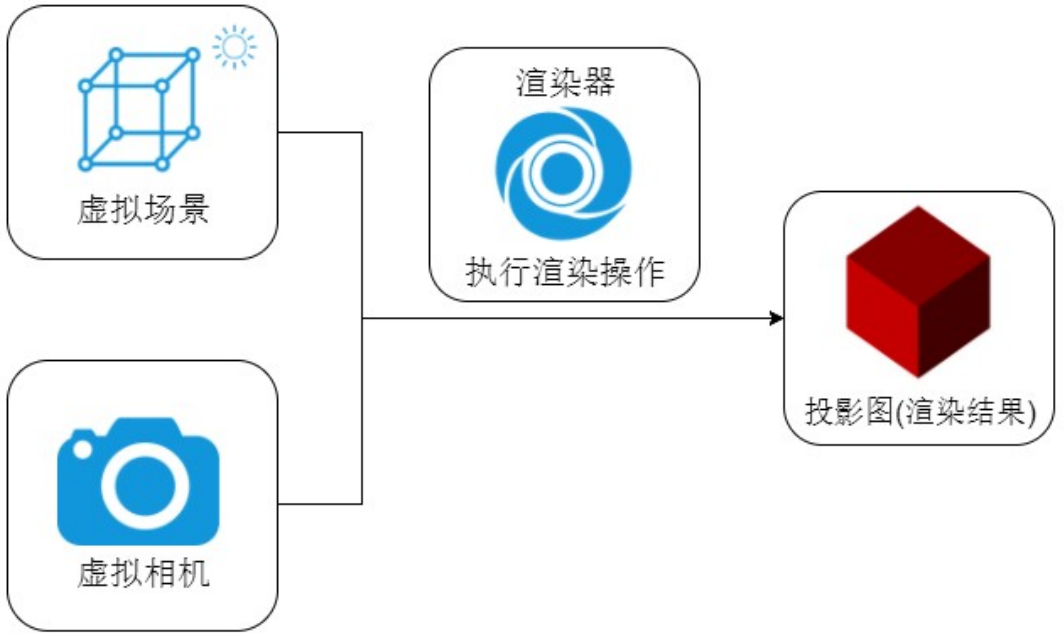
```
1  var scene = new THREE.Scene();  // 场景
2  var camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight, 
3  var renderer = new THREE.WebGLRenderer();   // 渲染器
4  renderer.setSize(window.innerWidth, window.innerHeight);    // 设置渲染器的大小为窗口的
5  document.body.appendChild(renderer.domElement);
```

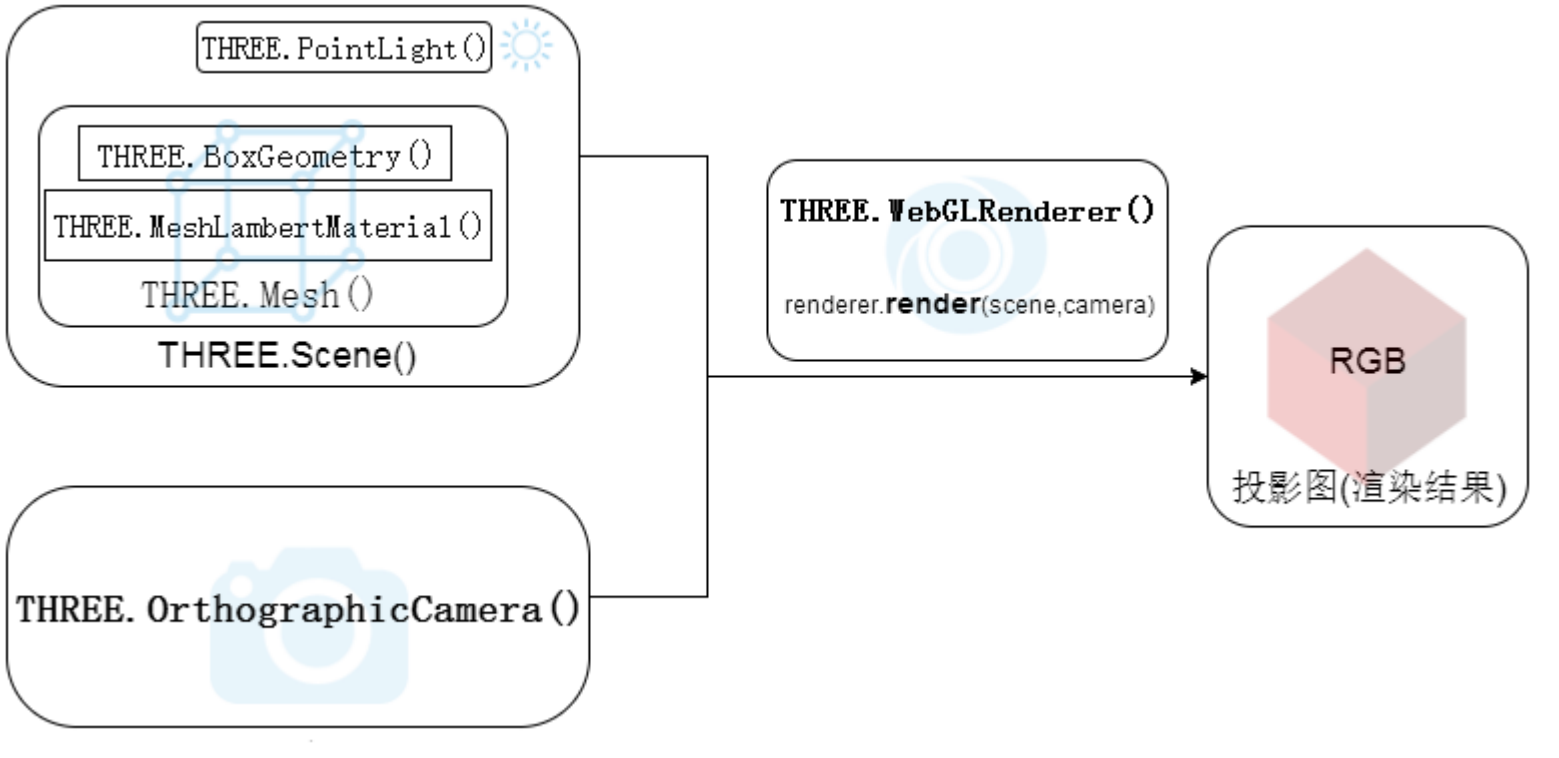


HTTP://WWW.WEBGL3D.CN/THREE.JS/

整个程序的结构

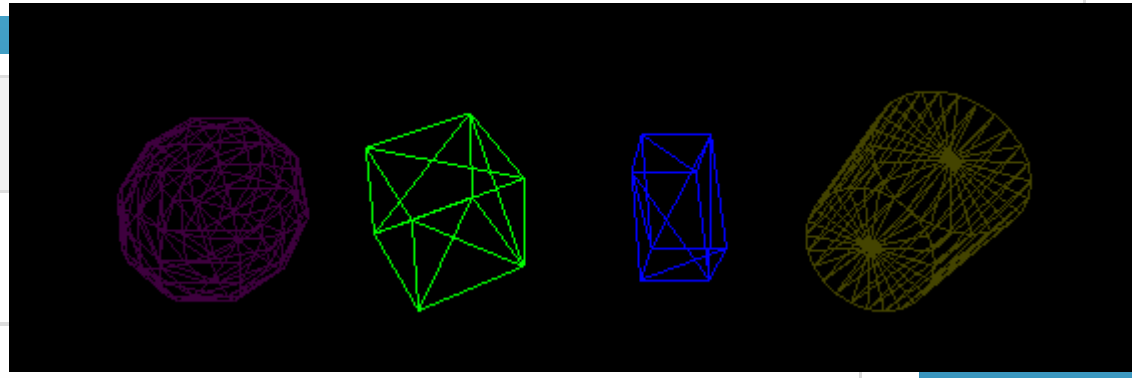



虚拟场景
渲染器
执行渲染操作
投影图(渲染结果)
虚拟相机


THREE.PointLight()
THREE.BoxGeometry()
THREE.MeshLambertMaterial()
THREE.Mesh()
THREE.Scene()
THREE.OrthographicCamera()
THREE.WebGLRenderer()
renderer.render(scene,camera)
RGB
投影图(渲染结果)

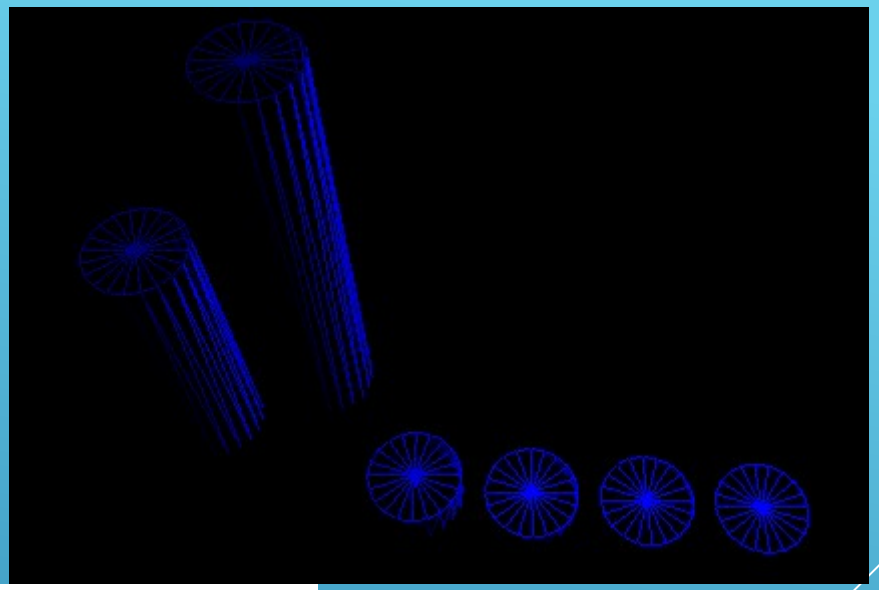
- var box=new THREE.SphereGeometry(60,40,40);// 创建一个球体几何对象
- // 长方体 参数：长，宽，高
- var geometry = new THREE.BoxGeometry(100, 100, 100);
- // 球体 参数：半径 60 经纬度细分数 40,40
- var geometry = new THREE.SphereGeometry(60, 40, 40);
- // 圆柱 参数：圆柱面顶部、底部直径 50,50 高度 100 圆周分段数
- var geometry = new THREE.CylinderGeometry(50, 50, 100, 25);
- // 正八面体
- var geometry = new THREE.OctahedronGeometry(50);
- // 正十二面体
- var geometry = new THREE.DodecahedronGeometry(50);
- // 正二十面体
- var geometry = new THREE.IcosahedronGeometry(50);

材质类型	功能
MeshBasicMaterial	基础网格材质，不受光照影响的材质
MeshLambertMaterial	Lambert网格材质，与光照有反应，漫反射
MeshPhongMaterial	高光Phong材质,与光照有反应
MeshStandardMaterial	PBR物理材质，相比较高光Phong材质可以更好的模拟金属、玻璃等效果

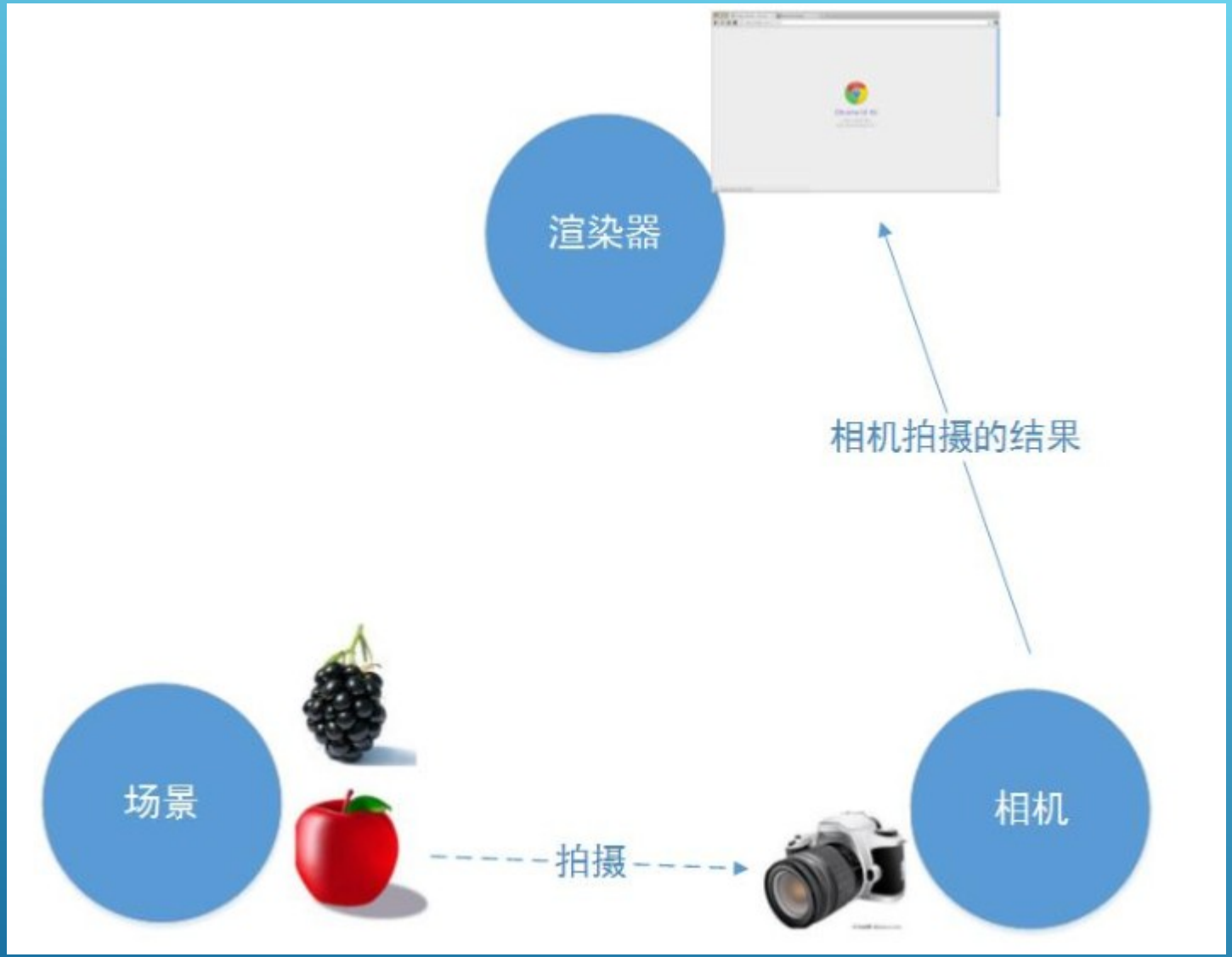


光源	简介
AmbientLight	环境光
PointLight	点光源
DirectionalLight	平行光，比如太阳光
SpotLight	聚光源

```
var data=[{date:'6 月 15',totle:673,add:28},
          {date:'6 月 14',totle:645,add:44},
          {date:'6 月 13',totle:601,add:4},
          {date:'6 月 12',totle:597,add:2},
          {date:'6 月 11',totle:595,add:1},
          {date:'6 月 11',totle:594,add:0}];
```



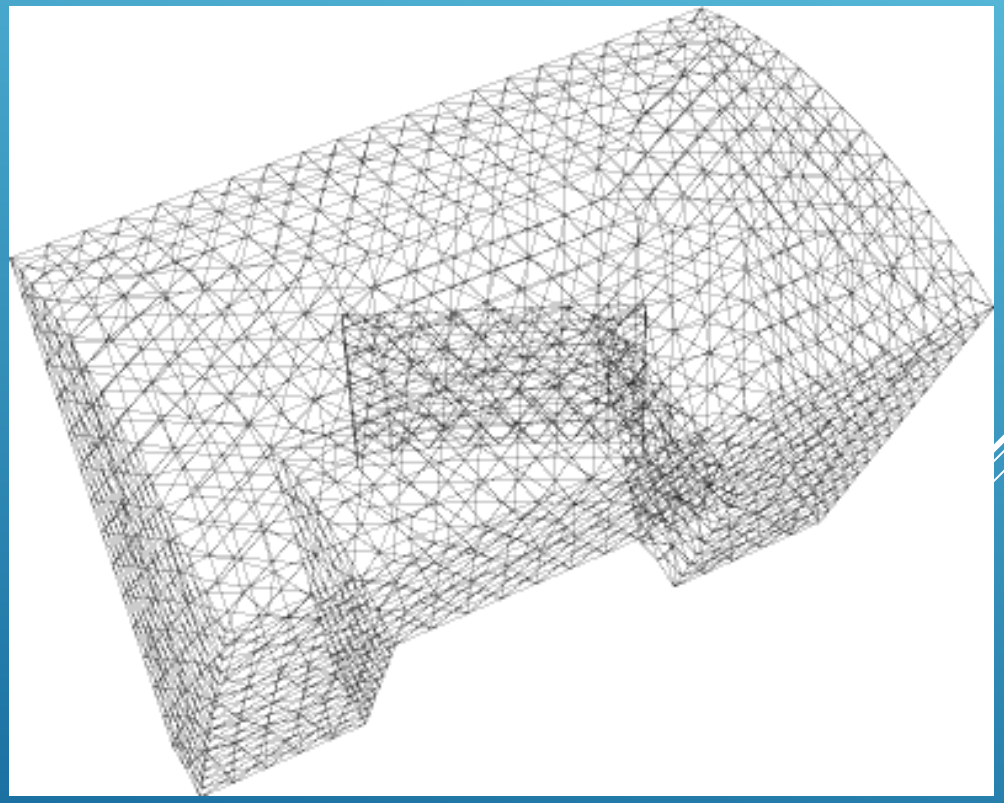
```
var cylinder=new Array(6);

// 圆柱网格模型
for(var i=0;i<6;i++){
    var geometry3 = new THREE.CylinderGeometry(20, 20, data[i].add*5, 20);
    var material3 = new THREE.MeshPhongMaterial({
      wireframe:true,
      color: 'rgb('+(255-data[i].add)+',255,255)',
      specular:0x4488ee,
      shininess:12
    });
    cylinder[i] = new THREE.Mesh(geometry3, material3); //网格模型对象Mesh

    cylinder[i].position.set(-100+i*50,data[i].add*5/2,0);//设置mesh3模型对象的xyz坐标
    scene.add(cylinder[i]); //
}
```



场景，相机，渲染器之间的关系

- 3D 世界是由点组成，两个点能够组成一条直线，三个不在一条直线上的点就能够组成一个三角形面，无数三角形面就能够组成各种形状的物体
- 网格模型叫做 Mesh 模型
- 给物体贴上皮肤，或者专业点就叫做纹理

3D 世界的组成



- 中间使用了一个“||”（或）运算符，就是当 x=null 或者 undefine 时， this.x 的值应该取 0 。

```
THREE.Vector3 = function ( x, y, z ) {

this.x = x || 0;
this.y = y || 0;
this.z = z || 0;

};
```

在 THREEJS 中定义一个点

```
var point1 = new THREE.Vecotr3(4,8,9);
```

你也可以使用set方法，代码如下：

```
var point1 = new THREE.Vector3();
point1.set(4,8,9);
```

定义一个点

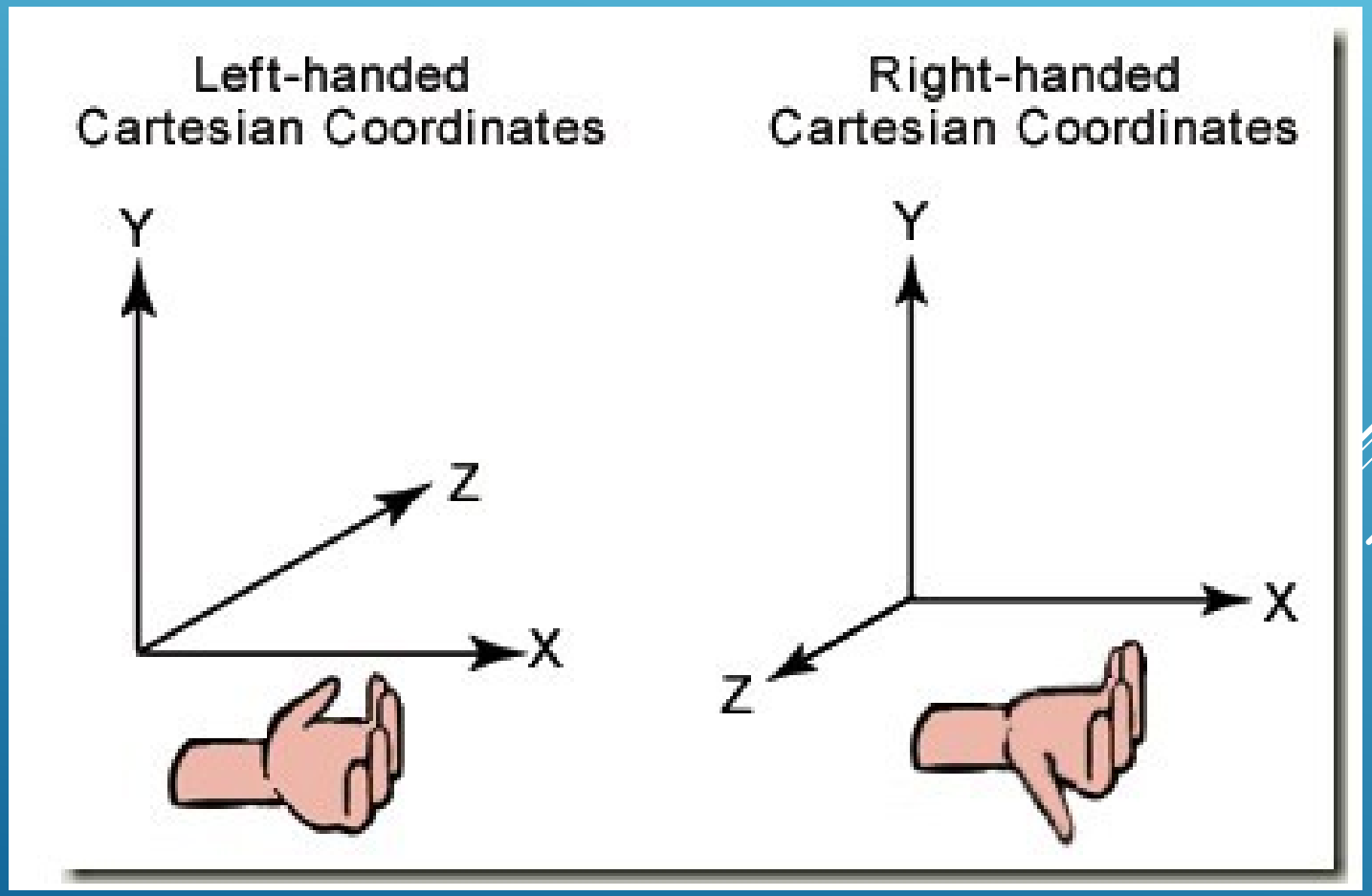
NoColors: 0,

FaceColors: 1,

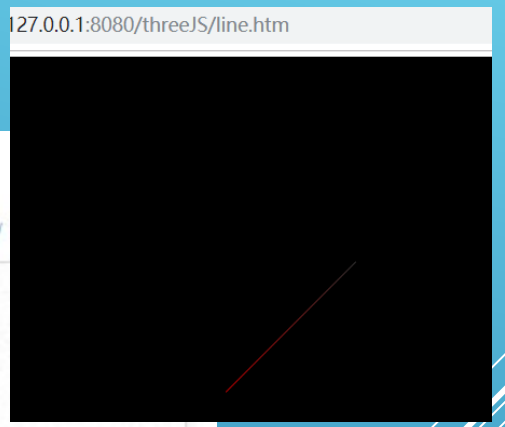
VertexColors: 2,

- x 轴正方向向右， y 轴正方向向上， z 轴由屏幕从里向外。

坐标系



- 在 Threejs 中，一条线由点，材质和颜色组成。
- 点由 THREE.Vector3 表示， Threejs 中没有提供单独画点的函数，它必须被放到一个 THREE.Geometry 形状中，这个结构中包含一个数组 vertices ，这个 vertices 就是存放无数的点（ THREE.Vector3 ）的数组。
- 可以使用专为线准备的材质， THREE.LineBasicMaterial 。



为了绘制一条直线，首先我们需要定义两个点，如下代码所示：

View

```
1  var p1 = new THREE.Vector3( -100, 0, 100 );
2
3  var p2 = new THREE.Vector3(  100, 0, -100 );
```

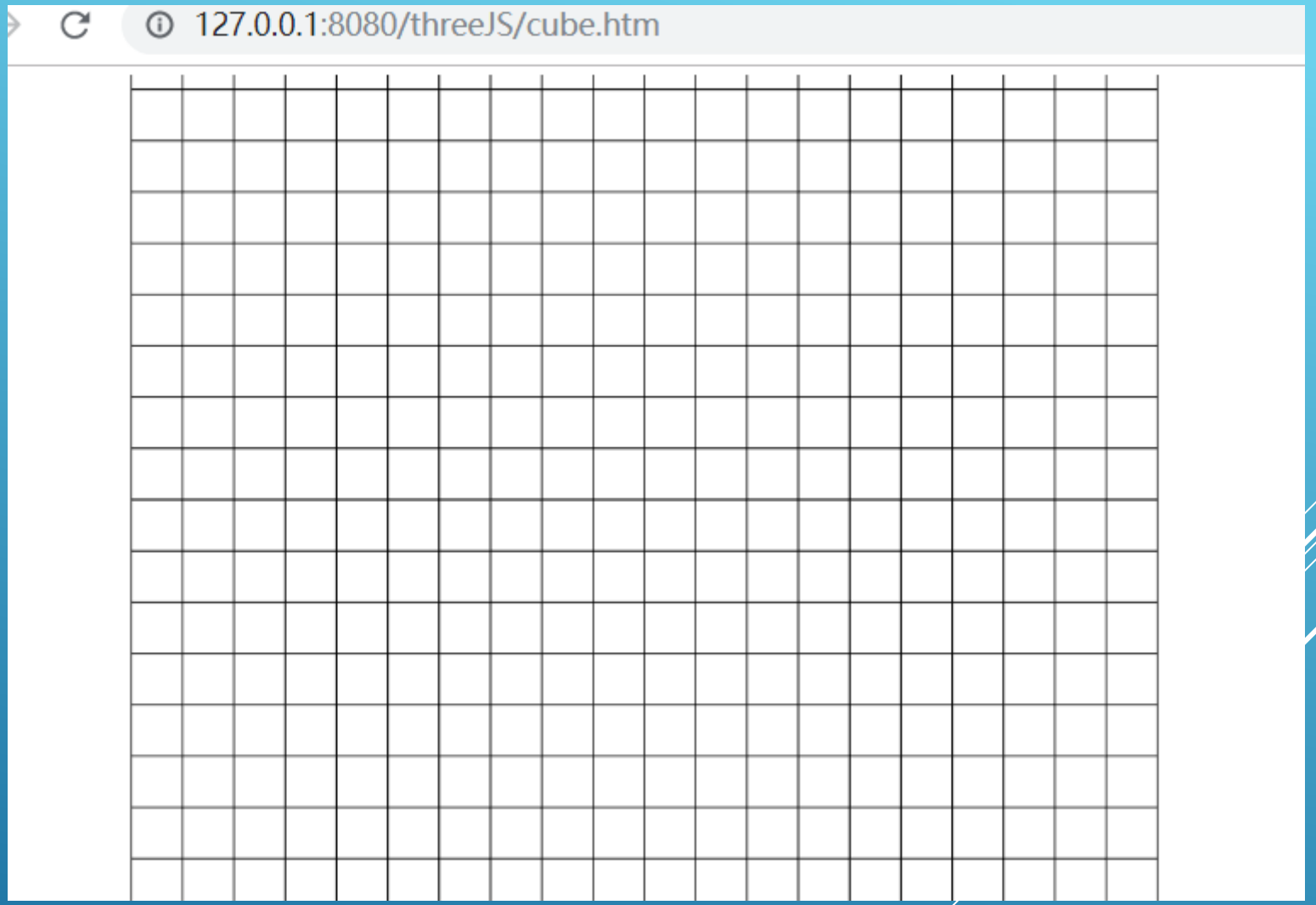
请大家思考一下，这两个点在坐标系的什么位置，然后我们声明一个THREE.Geometry，并把点加进入，代码如下所示：

View Raw Code ?

```
1  var geometry = new THREE.Geometry();
2
3  geometry.vertices.push(p1);
4
5  geometry.vertices.push(p2);
```

geometry.vertices的能够使用push方法，是因为geometry.vertices是一个数组。这样geometry 中就有了2个点了。

画线修改为网格线



动态的场景

- 第一种方法是让物体在坐标系里面移动，摄像机不动。
- 第二种方法是让摄像机在坐标系里面移动，物体不动。

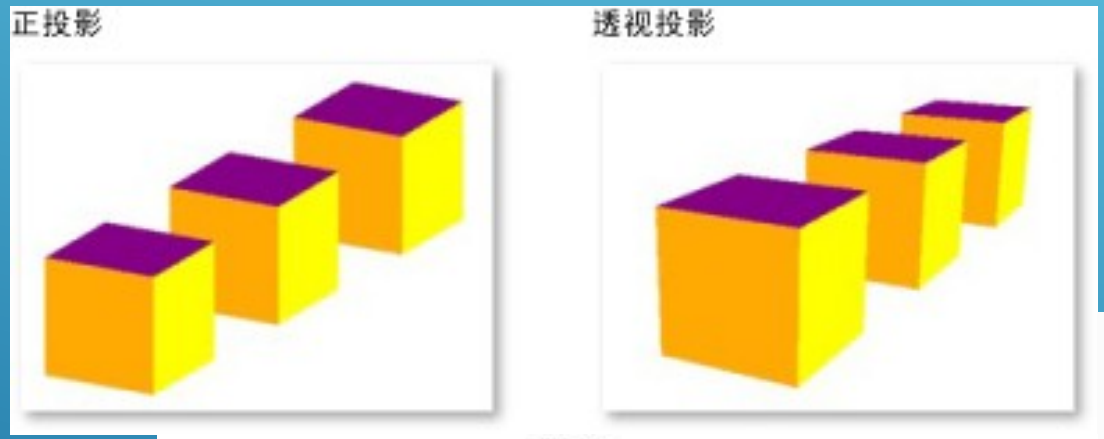
渲染循环

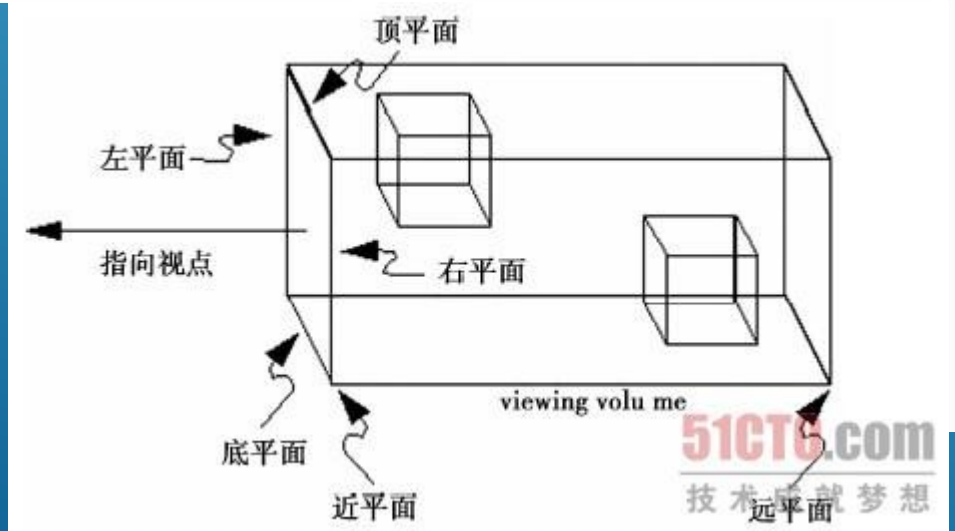
- 物体运动还有一个关键点，就是要渲染物体运动的每一个过程，让它显示给观众。渲染的时候，我们调用的是渲染器的 render() 函数。代码如下：
- renderer.render(scene, camera);
- 如果我们改变了物体的位置或者颜色之类的属性，就必须重新调用 render() 函数，才能够将新的场景绘制到浏览器中去。不然浏览器是不会自动刷新场景的。
- 需要 javascript 的一个特殊函数，这个函数是 requestAnimationFrame 。

```
function animate() {

    render();

    requestAnimationFrame( animate );

}
```

- OrthographicCamera(left, right, top, bottom, near, far)
- var camera = new THREE.OrthographicCamera(width / - 2, width / 2, height / 2, height / - 2, 1, 1000);
- scene.add(camera);

相机与投影

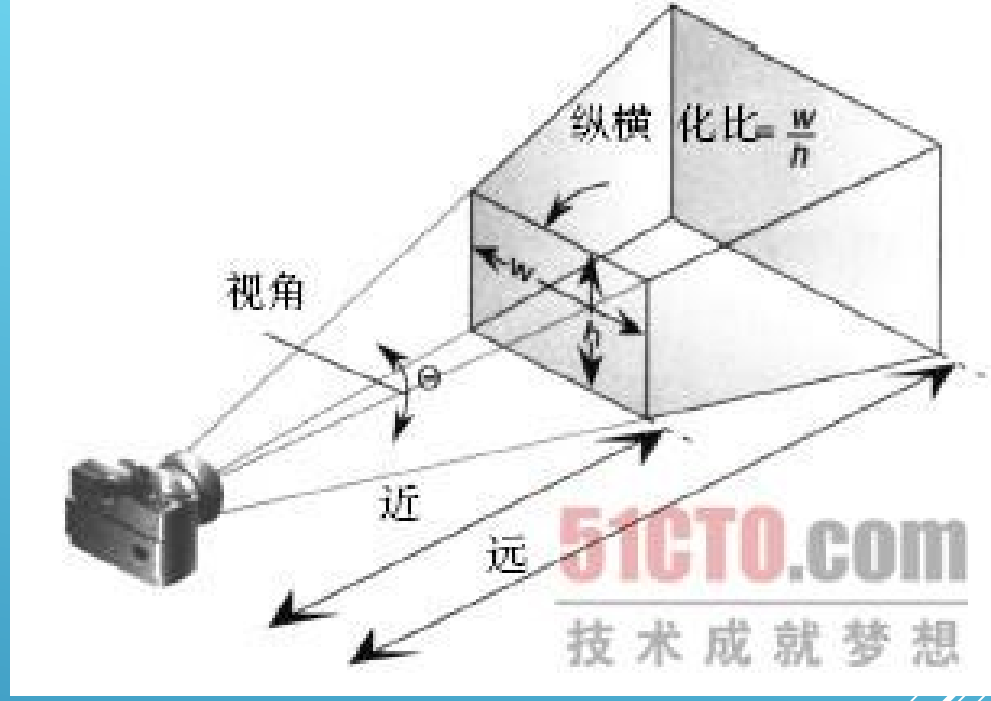




投影相机

- 透视投影相机的构造函数如下所示：
- PerspectiveCamera(fov, aspect, near, far)
- 视角 fov
- 纵横比 aspect
- 近平面 near
- 远平面 far
- var camera = new THREE.PerspectiveCamera(45, width / height, 1, 1000);

- scene.add(camera);

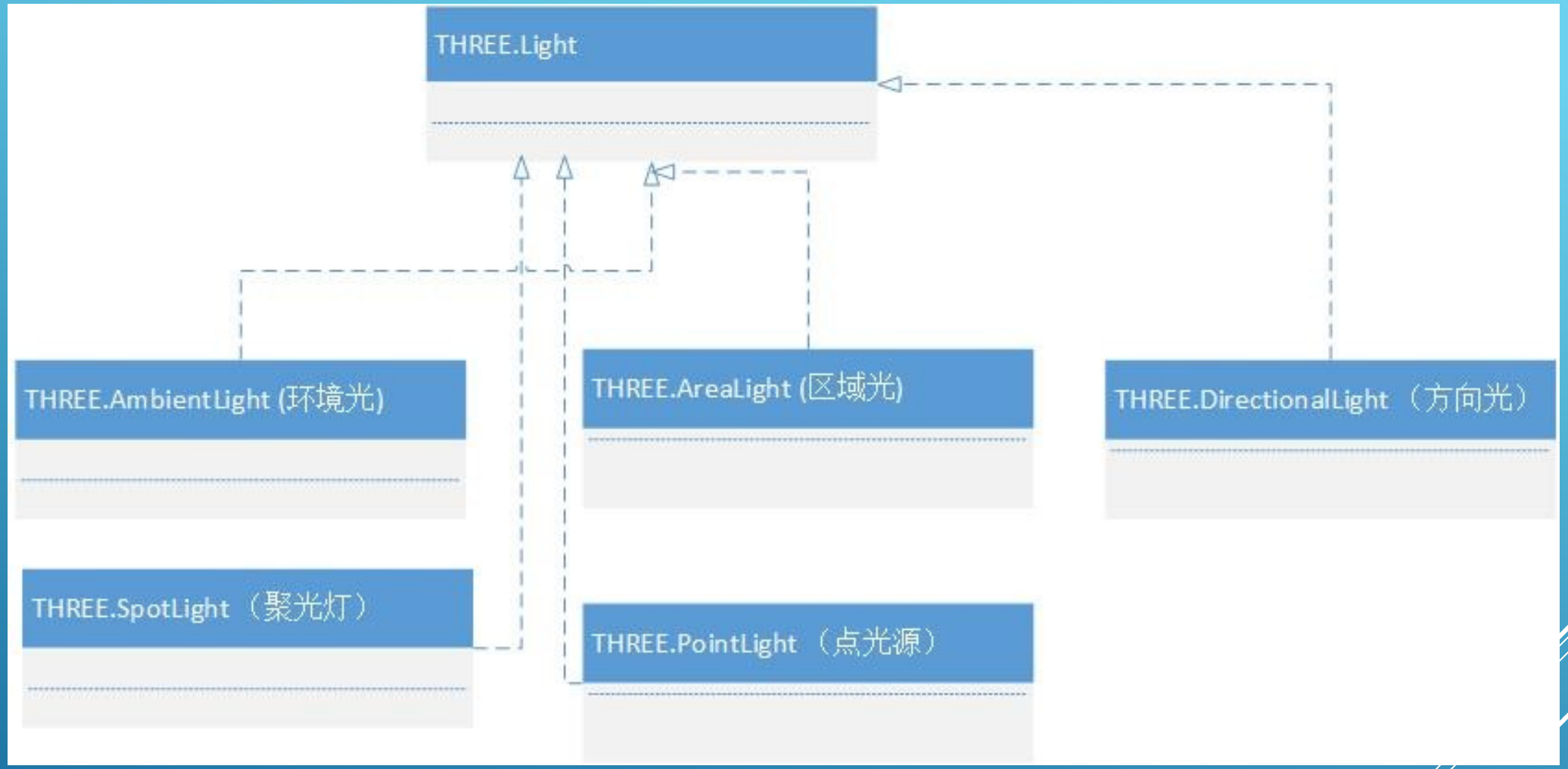


120度视角效果图如下:

160度视角效果图如下:

100度视角效果图如下:

人类的正常视角是 120 度左右
集中注意力看清楚，在 30-40 度



世界有了光，就不在黑暗
THREE.LIGHT 只是其他所有光源的基类

环境光

- 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。
- var light = new THREE.AmbientLight(0xff0000);
- scene.add(light);

点光源

- 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 。

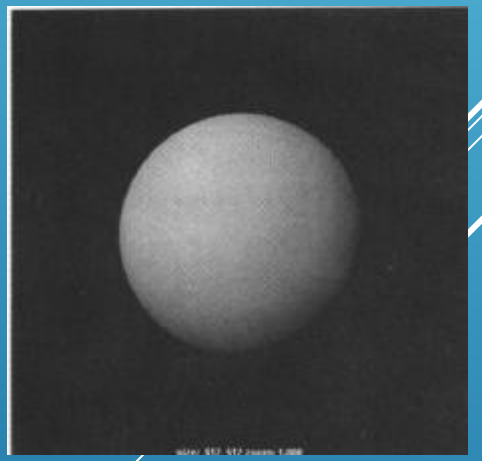
聚光灯

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

- 不带任何光源的物体
- 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
- 当没有任何光源的时候，最终的颜色将是材质的颜色。
- **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。**

不带任何光源的物体

- 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。
- 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。



LAMBERT 材质：均匀散射