主要知识点：

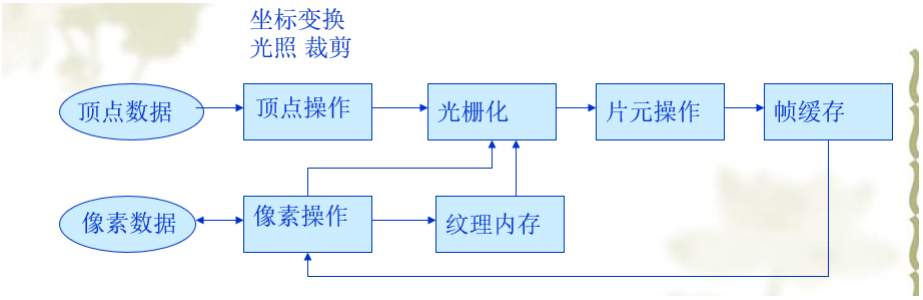
1， 数字图像处理、 计算机图形学、模式识别三门课之间的关系

图像处理 是利用计算机对图像进行分析处理，继而得到图形信息，再现图像
模式识别 计算机对图形信息进行识别和分析描述，是从图形到描述表达的过程
计算机图形学 则是研究图形的计算机生成和基本的图形操作，是从描述到图形生成的过程

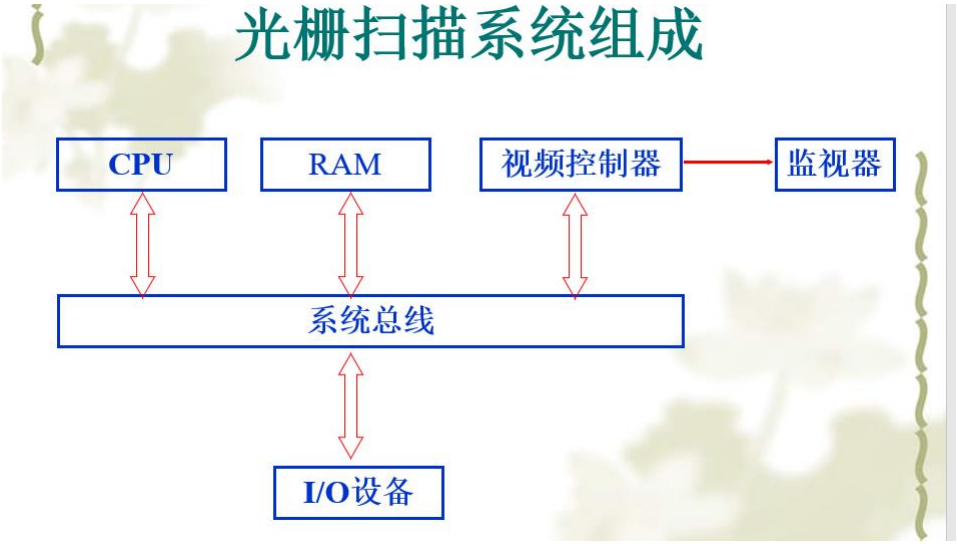
2， 就能够根据图进行分析。图形管线的各个阶段工作。

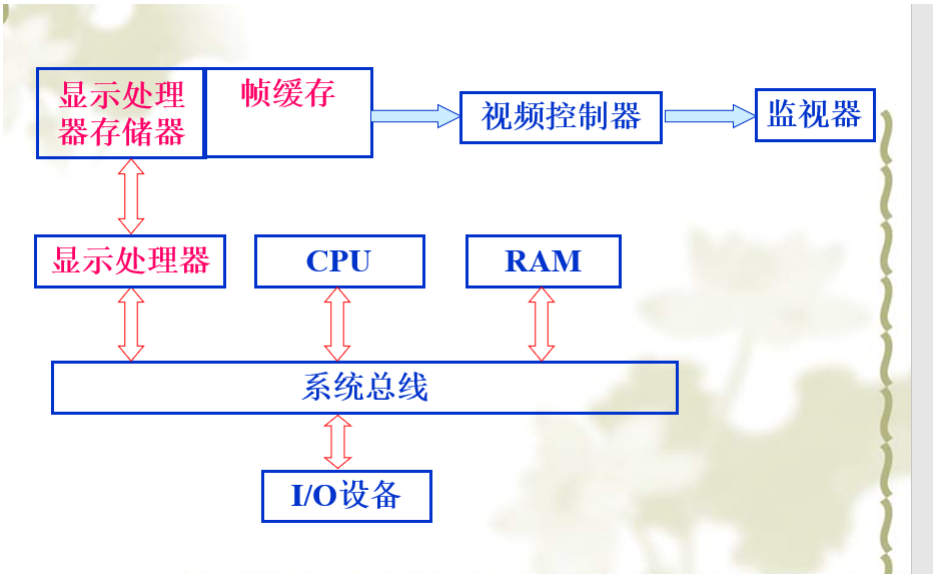
渲染管线的功能是通过给定虚拟相机、3D 场景物体以及光源等场景要素来产生或者渲染一副 2D 的图像，是从 3D 到 2D 的过程。

输入顶点数据、进行顶点操作（包括顶点着色器着色和图元装配）【从世界坐标系到屏幕坐标系】、进行光栅化得到片元，进行片元操作、帧缓存为像素、进行像素操作。



3， 计算机图形处理架构（有 GPU 模式和无 GPU 模式）





4， 简述光栅式扫描显示系统

由 显示处理器、帧缓冲存储器、视频控制器构成。

显示处理器将应用程序给出的图形定义数字化为一组离散的像素强度值并将其存放在帧缓冲存储器之中，最后由视频控制器进行刷新操作，将帧缓冲器之中的像素对应位置的值取出，用来设置 CRT 电子束的强度值

1， 屏幕分辨率三种描述，扫描频率，带宽计算

屏幕分辨率：

- 光点直径，**荧光屏上两个相邻的相同颜色磷光点之间的最短距离**
- 水平方向上的光点数×垂直方向上的光点数 r(x×y)
- 显示器精度 dpi

扫描频率：**也叫刷新率，分为行频、场频**

行频（水平扫描频率，h 来表示）：电子枪每秒在屏幕上扫描过的水平线数

场频（v 表示）：就可以理解为一个场，是每秒钟重复绘制显示画面的次数

带宽计算：

理论带宽计算：B = r(x) ✕ r(y) ✕ v

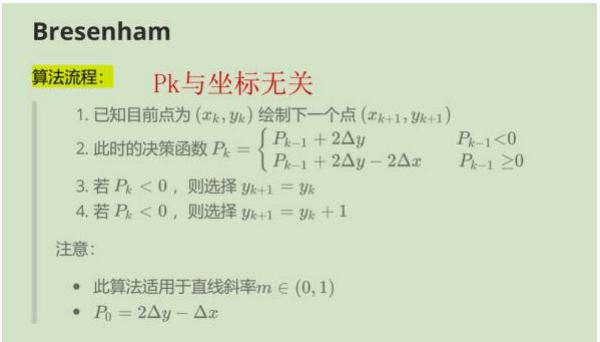
r（x）为水平扫描的点数 r(y) * v = h

r（y）为每帧扫描的线数

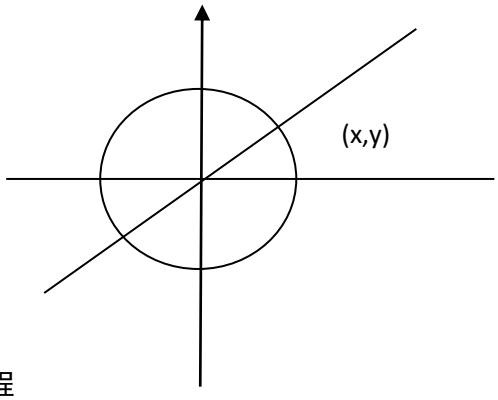
v 为场频

5， 用 bresenham 算法光栅化线段 $P_1(30, 20)$，$P_2(40, 27)$

可以先算出 Pk 的表，再列 Xk+1, Yk+1 的表



6， 画图描述圆对称性坐标计算，



4. 描述扫描线多边形填充算法流程

对于一条扫描线填充过程可以分为四个步骤：

（1）求交

（2）排序

（3）配对

（4）填色

1. 输入多边形顶点数及顶点坐标，建立有序边表；

2. 扫描线自底向上进行扫描，根据当前扫描值建立活化边表【排序和配对】;

3. 用要求的颜色显示这些区间的象素，即完成填充工作

4. 更新活化边表并重新排序；

5. 进入下一条扫描线，重复上述过程直至扫描线值为最高顶点的 y 坐标值

算法描述

1) 输入多边形顶点数及顶点坐标

2) 建立有序边表

3) 根据当前扫描值建立活化边表

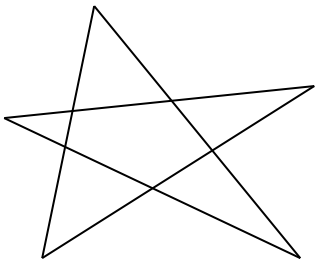
4) 填充

5) 更新活化边表并重新排序

6) 进入下一条扫描线，重复步骤3，直至扫描线值为最高顶点的y坐标值

6， 分别用奇偶规则和非零环绕规则判断下列内外区域

奇内，偶外 ；非零为内



6， 构造大矩阵实现，视点（0, 0, 0）从 z 轴负方向，到 y 轴上一点（0, 5, 0）向（0, 0, 0）观察变换。

8， 在二维平面上，构造大矩阵实现，绕 y=x 轴旋转 45 度角的旋转矩阵。

绕着哪个轴转，哪个轴的分量不变。确定角度时，忽视这个分量！另外俩分量的平方和相等，斜边永远是向量长度。注意是 z x y

9， 描述观察变换的坐标变换关系

世界坐标系中部分场景映射到设备坐标系的过程称为观察变换，也叫视像变换，或称为从窗口到视口的变换。先建立坐标系，再列出变换矩阵。

二维：

在世界坐标系中选择某个位置作为观察参考坐标系的原点 $P_0=(x_0, y_0)$

将一个世界坐标系的矢量 V 作为观察坐标系 y_v 轴方向，矢量 V 为观察向量

给定 V，就可分别计算在观察坐标轴 y_v 和 x_v 上的单位向量 $v=(v_x, v_y)$ 和 $u=(u_x, u_y)$

变换矩阵：R * T

三维：

步骤 1：观察参考点：P0（x0，y0，z0）

步骤 2：观察平面法向量 N：观察 Zv 轴的正方向和观察平面方向

步骤 3：向量 v，观察向上向量：建立 Yv 轴（v 下标代表观察坐标 w 代表世界坐标）的正方向

步骤 4：利用右手原则得到 Xv 轴

变换矩阵：Rz *Ry* Rx * T

10，已知 w1=10， w2=20， w3=40， w4=80，

v1=-10， v2=20， v3=10， v4=120，

窗口中一点 P(15，60)，求视口中的映射点 P'

$$(X_w-W_1)/(W_2-W_1) = (X_v-V_1)/(V_2-V_1)$$

$$(Y_w-W_3)/(W_4-W_3) = (Y_v-V_3)/(V_4-V_3)$$

11，已知线段的两个端点

P1(-3/2， 1/6)，P2(1/2， 3/2)

窗口边界 x=-1， x=1， y=-1， y=1

用 CS 算法对线段进行剪裁

1. 计算直线端点编码，c_1和c_2
2. 判断:
   - c_1和c_2**均为0000**，保留直线
   - c_1&c_2**不为零**，同在某一边界外，删除该直线
   - c_1&c_2**为零**，需要进一步求解交点 区域码从右到左
3. **先起点后终点，以L,R,B,T为序**，找出端点区域码中第一位为1的位，将x=w1、w2或y=w3、w4代入直线方程，计算直线与窗口边界的交点，将交点和另一端点重复上述过程，直至线段保留或删除

12， 利用 LB 算法实现上述例子

$$P_k \times u \le Q_k \qquad \text{k=1,2,3,4}$$

$$\begin{cases} P_1 = -\Delta x & Q_1 = x_1 - w_1 \\ P_2 = \Delta x & Q_2 = w_2 - x_1 \\ P_3 = -\Delta y & Q_3 = y_1 - w_3 \\ P_4 = \Delta y & Q_4 = w_4 - y_1 \end{cases}$$

算法描述：

1. 计算P_k和Q_k
2. 若存在k，使$P_k = 0$且$Q_k < 0$，则舍去直线
3. 对其他情况通过计算Q_k / P_k来确定交点所对应的u值
4. $\begin{cases} u_1 = \max(\{\frac{Q_k}{P_k} | P_k < 0\}, 0) \\ u_2 = \min(\{\frac{Q_k}{P_k} | P_k > 0\}, 1) \end{cases}$
5. 如果$u_1 > u_2$，则直线在窗口外，否则计算交点坐标$\begin{cases} x(u) = x_1 + \Delta x \cdot u \\ y(u) = y_1 + \Delta y \cdot u \end{cases}$

书写模板：

线段参数方程x(u)= -2+3u

y(u) = -1+2.5u

$P_1 = -3$ $Q_1 = -1$ $R_1 = 1/3$

$P_2 = 3$ $Q_2 = 3$ $R_2 = 1$

$P_3 = -2.5$ $Q_3 = 0$ $R_3 = 0$

$P_4 = 2.5$ $Q_4 = 2$ $R_4 = 4/5$

对于$P < 0$，$u_1 = \max\{0, 1/3, 0\} = 1/3$

对于$P > 0$，$u_2 = \min\{1, 1, 4/5\} = 4/5$

则$u_1 < u_2$，则可见线段的端点坐标：

$x = x_1 + 3u_1 = -1$，　$x = x_1 + 3u_2 = 2/5$，

$y = y_1 + 2.5u_1 = -1/6$　$y = y_1 + 2.5u_2 = 1$

即(-1, -1/6)　即(2/5, 1)

LB 效率高于 CS， 因为减少了交点计算次数

LB 和 CS 都易于拓展成三维裁剪算法

13， 简述平行投影，透视投影概念

投影：**将三维物体投射到二维观察平面上**

平行投影：将物体表面上的点**坐标沿平行线变换到观察平面上**

分为正投影（投影向量垂直观察屏幕）、斜投影

透视投影：将物体位置沿收敛于某点的直线变换到观察平面

14， 投影的三要素

- **投影中心(也叫投影参考点)COP**
- **投影平面(也叫观察平面)**

- **投影线（也叫视线）**

15， 主灭点，一点透视，两点透视，三点透视

一组平行线投影后收敛于一点称之为灭点。

物体中平行于某一坐标轴的平行线的灭点

透视投影按照主灭点数目分类 一点透视 二点透视 三点透视

16， 观察体调整

观察体：利用投影窗口边界来设置

观察体作用：**对三维物体进行裁剪**

分为无限型：棱锥、无穷平行管道 有限型：棱台、平行六面体

调整目的：**计算、处理方便快捷、完整渲染图形**，节省计算资源。

斜平行投影和透视投影的观察体均需要调整，

比如错切摆正，利于计算，放缩使得观察体外的点落进里面

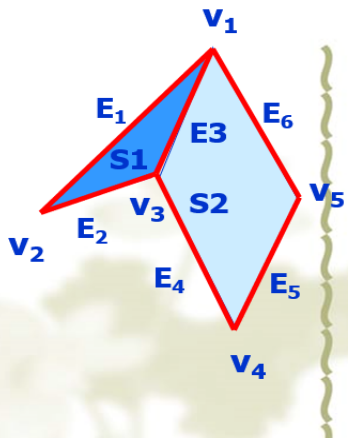
17， 概念，多边形网格模型表示中，基本数据表形式，顶点表，边表，面表

- **几何表：顶点坐标和用来标识多边形表面空间方向的参数**
- **属性表：指明物体透明度及表面反射度的参数和纹理特征**

顶点表	
序号	点坐标
1	x1, y1, z1
2	x2, y2, z2
3	x3, y3, z3
4	x4, y4, z4
5	x5, y5, z5

边表	
序号	顶点号
1	v1, v2
2	v2, v3
3	v3, v1
4	v3, v4
5	v4, v5
6	v5, v1

多边形面表	
序号	边序号
1	E1, E2, E3
2	E3, E4, E5, E6



18，概念，用函数描述的二次曲面模型，如球体表面，如何进行绘制

一旦给定函数，图形包将指定曲线方程投影到显示平面上，且沿着投影函数路径绘制像素位置，最终绘制成曲面。

19，概念， 样条曲线，样条曲面

- **样条：通过一组指定点集而生成平滑曲线的柔性带。**
- **样条曲线在计算机图形学中的含义**
  - **由多项式曲线段连接而成的曲线**
  - **在每段的边界处满足特定的连续性条件**
- **样条曲面**
  - **使用两组正交样条曲线进行描述**

20，概念，样条曲线的两种类型

- 插值样条曲线：选取的多项式使得曲线通过每个控制点

- 逼近样条曲线：选取的多项式不一定使曲线通过每个控制点

21，概念， 凸壳的概念

定义：包含一组控制点的凸多边形边界

作用：

提供了曲线或曲面与包围控制点的区域之间的偏差的测量

以凸壳为界的样条保证了多项式沿控制点的平滑前进

22，概念，分段连续中连续的定义

参数连续性条件：

两个相邻曲线段在相交处的参数导数相等

零阶连续(C0 连续)：简单地表示曲线连接，两个曲线必在相交点处有相同的坐标

一阶连续(C1 连续)：说明代表两个相邻曲线的方程在相交点处有相同的一阶导数（切线）

二阶连续(C2 连续)：两个曲线段在交点处有相同的一阶和二阶导数，交点处的切向量变化率相等

几何连续性条件：

两个相邻曲线段在相交处的参数导数成比例

零阶连续（G0 连续）：与 0 阶参数连续性相同，即两个曲线必在公共点处有相同的坐标

一阶连续（G1 连续）：表示一阶导数在两个相邻曲线的交点处成比例

二阶连续（G2 连续）：表示两个曲线段在相交处的一阶和二阶导数均成比例

23， 简述 Bezier 的几个特点

- **Bezier 曲线总是通过第一个和最后一个控制点**
- **Bezier 曲线在第一个控制点 P_0 处与直线 P_0P_1 相切，在最后一个控制点 P_n 处与直线 $P_{n-1}P_n$ 相切。**
- **Bezier 曲线总是落在控制点的凸壳内，保证了曲线沿控制点的平稳前进**
- **第一和最后一个控制点重合则生成封闭 Bezier 曲线**
- **多个控制点位于同一位置会对该位置加以更多的权**

24， threejs 的浏览器中如何调试的方法，文件相对路径关系（src=“/images/ddd.jpg”与 “../images/ddd.jpg” 与 “./images/ddd.jpg”与 “images/ddd.jpg”的区别）

绝对路径，父文件夹，当前文件夹，默认当前文件夹。

25， VBO,PBO,FBO 的概念

顶点缓冲对象（Vertex Buffer Objects，VBO）存放顶点数组数据

像素缓冲对象(Pixel Buffer Object) 用于保存像素数据

帧缓存对象（Frame Buffer Object）逻辑缓存。

26， 如何创建一个 vbo，并基于 vbo 进行绘制（写出相应的伪代码过程）

使用 glGenBuffers()生成新缓存对象。

使用 glBindBuffer()绑定缓存对象。

使用 glBufferData()将顶点数据拷贝到缓存对象中。

Generate 生成 bind 绑定 data 拷贝

27， 深度缓存的意义，基于深度检测的方法
可用于 深度缓冲器算法，**进行可见面的计算**
利用深度缓存器对各像素的深度进行缓存，保存可见面的深度值，逐个扫描多边形面表的各个表面，每次扫描一行，计算深度值，最后存储深度最小的值。
而算法通过对投影平面上每个像素所对应的表面深度进行比较，可进行可见面的计算

28， 描述固定管线中的光照模型
表面照明效果分为：
Lambert 模型(漫反射)、Phong 模型(镜面反射)、Rendering Equation(全局光照模型、环境光)

环境光：该场景各个表面的反射光生成的综合光照效果。整体一致的亮度
$I_{ambdiff} = K_a I_a$
K_a - ambient-reflection coefficient
I_a -- ambient light intensity

漫反射：粗糙表面会将反射光向个方向发散出去，从任何角度来看表面亮度均相同
$I_{Ldiff} = K_d I_l \cos\theta = K_d I_l (N.L)$
K_d - diffuse-reflection coefficient
I_l -- intensity of point light source

镜面反射：在平滑材质的表面，反射光会集中成醒目的或明亮的一个点
$I_{spec} = w(\theta) I_l \cos^{ns}\Phi$
$=K_s I_l (V.R)^{ns}$
W() 镜面反射系数
Φ 为观察方向 V 与镜面反射方向 R 的夹角
ns 代表镜面反射参数

Phong Model ： $I = I_{ambdiff} + I_{ldiff} + I_{spec}$

$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$

最后有实验题，而且都是大题，一般共占 40 分左右

1， threejs 的程序主框架（初始化和主循环,）
2， 利用 threejs 实现某个场景的绘制，写出设计和具体代码
3， 利用 threejs 实现对象的运动控制的设计，如太阳系运动，以及汽车与轮子运动等。
4， 利用 threejs 实现天空盒的搭建
6. 利用 threejs 设计实现一个函数曲线的绘制
7， 利用 threejs 设计实现一个曲面的绘制

Threejs 题目

1.主框架

```
//初始化
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 );

//主循环
function update () {
 renderer.render(scene, camera);
 requestAnimationFrame(update);
}

//完整示例
<html>
<head>
```

```
    <title>My first three.js app</title>
    <style>
        body {
            margin: 0;
        }
        canvas {
            width: 100%;
            height: 100%
        }
    </style>
</head>

<body>
    <script src="js/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(1, 1, 1);
        var material = new THREE.MeshBasicMaterial({
            color: 0x00ff00
        });
        var cube = new THREE.Mesh(geometry, material);
        scene.add(cube);
        camera.position.z = 5;
        function animate() {
            requestAnimationFrame(animate);
            cube.rotation.x += 0.1;
            cube.rotation.y += 0.1;
            renderer.render(scene, camera);
        };
        animate();
    </script>
</body>

</html>
```

2.场景绘制(我猜这是要自己设置几个几何体组合起来、得记住常见的几何体)

```
var group = new THREE.Group();
group.add(mesh1).remove(mesh1);
group.translateY(100);
```

```
group.scale.set(4,4,4);
group.rotateY(Math.PI/6)

//光照
var point = new THREE.PointLight(0xffffff);
   point.position.set(400, 200, 300);
   scene.add(point);
   var ambient = new THREE.AmbientLight(0x444444);
   scene.add(ambient);
var directionalLight = new THREE.DirectionalLight(0xffffff,
1);
   directionalLight.position.set(80, 100, 50);
   directionalLight.target = mesh2;
   scene.add(directionalLight);
var spotLight = new THREE.SpotLight(0xffffff);
   spotLight.position.set(200, 200, 200);
   spotLight.target = mesh2;
   spotLight.angle = Math.PI / 6
   scene.add(spotLight);

//阴影
    光源、投影物体 .castShadow 设置成 true 接收的物
```

```
体.recieveShadow 设为 true
```

```
    // 球体 参数：半径 60 经纬度细分数 40,40
```

```
    var geometry = new THREE.SphereGeometry(60, 40, 40); //
球体
```

```
  //长方体 参数：长，宽，高
```

```
    var geometry = new THREE.BoxGeometry(100, 100, 100); //
立方体
```

```
  // 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 );
```

```
  var planeGeometry = new THREE.PlaneGeometry(300, 200);
```

```
CircleGeometry
```

```

```

```
 var material = new THREE.MeshLambertMaterial、PointsMaterial、
LineBasicMaterial({
```

```
    color: 0x0000ff,
```

```
    transparent:true,
```

```
    opacity:xx,
```

```
    wireframe:true,
```

```
 });
```

```
 var mesh = new THREE.Mesh(geometry, material);
```

```
 scene.add(mesh);
```

3.利用 threejs 实现对象的运动控制的设计，如太阳系运动，以及汽车与轮子运动等。

```
    var geometry = new THREE.BoxGeometry(100, 100, 100);
    var material = new THREE.MeshLambertMaterial({
      color: 0x0000ff
   });
    var mesh = new THREE.Mesh(geometry, material);
    scene.add(mesh);

function render() {
   mesh.position.x = mesh.position.x + 1
        renderer.render(scene,camera);//执行渲染操作
        requestAnimationFrame(render);//请求再次执行渲染函数
render
   }
render();
```

- 方法
  - 平移
    - translateX(distance)
    - translateOnAxis(axis,distance)
    - 改变 position 属性
  - 旋转
    - rotateX(angle)
    - rotateOnAxis(axis,angle)
    - 改变 rotation，quaternion 属性
- 属性 = 属性值
  - 缩放 .scale = Vector3
  - 位置 .position = Vector3
  - 角度 .rotation = Euler
  - 四元数 .quaternion = Quaternion

平移

Vector3 对象具有属性.x、.y、.z，Vector3 对象还具有方法.set()，.set 方法有三个表示 xyz 方向缩放比例的参数。

```
//网格模型 xyz 方向分别缩放 0.5,1.5,2 倍
mesh.scale.set(0.5, 1.5, 2)
//x 轴方向放大 2 倍
mesh.scale.x = 2.0;
//mesh.position 是 Vector3，而 Vector3 有方法 set
mesh.position.set(80,2,10);
```

执行.translateX()、.translateY()、.translateOnAxis()等方法本质上改变的都是模型的位置属性.position。

4.天空盒

```
scene.background = new THREE.CubeTextureLoader()
        // .setPath( 'images/' ) z
            .load( [
                'px.jpg',
                'nx.jpg',
                'py.jpg',
                'ny.jpg',
                'pz.jpg',
                'nz.jpg'
            ] );
```

5 射线

```
        //(一）鼠标控制旋转
        var rotateStart=new THREE.Vector2();
        var pivot = new THREE.Object3D(); //创建一个 obj 对象

        /*
            鼠标移动控制模型旋转思想：
            当按下鼠标时及时当前鼠标的水平坐标 clientX1，在鼠标移动
的过程中不断触发 onMouseMove 事件，
            不停的记录鼠标的当前坐标 clientX2，由当前坐标减去记录的
上一个水平坐标，
            并且将当前的坐标付给上一个坐标 clientX1，计算两个坐标的
之间的差 clientX2-clientX1，
            将得到的差值除以一个常量(这个常量可以根据自己的需要调整)，
得到旋转的角度
        */
 document.addEventListener('mousedown', onMouseDown,
false);
        var mouseDown = false;
        function onMouseDown(event) {
            event.preventDefault();
            mouseDown = true;
            mouseX = event.clientX; //出发事件时的鼠标指针的水
```

```
平坐标
            rotateStart.set(event.clientX, event.clientY);
            document.addEventListener('mousemove',
onMouseMove2, false);
        }
 document.addEventListener('mouseup', onMouseUp, false);
        function onMouseup(event) {
            mouseDown = false;
            document.removeEventListener("mousemove",
onMouseMove2);
        }

        function onMouseMove2(event) {
            if (!mouseDown) {
                return;
            }
            var deltaX = event.clientX - mouseX;
            mouseX = event.clientX;
            pivot.rotation.y += -deltaX / 279;
            render();
        }


```

```
var left = false;.......//先设置全局变量

        //键盘控制移动，使用 jquery；同时实现空格键发射子弹
        $(window).keydown(function (event) {
            switch (event.keyCode) {
                case 65: // a
                    left = true;
                    break;
                case 68: // d
                    right = true;
                    break;
                case 83: // s
                    back = true;
                    break;
                case 87: // w
                    front = true;
                    break;
                case 32: //空格
                    shutbin = true;
                    break;
```

```
            }
        });

        $(window).keyup(function (event) {
            switch (event.keyCode) {
                case 65: // a
                    left = false;
///

        function animate() { //实时渲染
            requestAnimationFrame(animate);
            renderer.render(scene, camera);
            if (front) {
                // cube.translateZ(-1)
                cube.position.z -= 1;
            }
            if (back) {
                // cube.translateZ(1);
                cube.position.z += 1;
            }
            if (left) {
                // cube.translateX(-1);
```

```
                cube.position.x -= 1;
            }
            if (right) {
                // cube.translateX(1);
                cube.position.x += 1;
            }
            if (shutbin) {
                game(); //将 zidan 的模型体导入，同时导入 cube 模型是为了获取他的位置坐标
            }
        }

        var v = 100;//速度
        function game() {
            //先创建一个子弹 scene.add(zidan)
            zidan.position = cube.position;
            zidan.rotation = cube.rotation;
            if (!check()) {
                zidan.position.X += Math.cos(zidan.rotation) * v;
                zidan.position.Y += Math.sin(zidan.rotation) * v;
            } else {
```

```
                scene.remove(zidan);
            }

            // Raycaster 来检测碰撞的原理很简单，我们需要以物体的中心为起点，向各个顶点（vertices）发出射线，然后检查射线是否与其它的物体相交。如果出现了相交的情况，检查最近的一个交点与射线起点间的距离，如果这个距离比射线起点至物体顶点间的距离要小，则说明发生了碰撞。
            check() {
                //target 是目标物体。
                var origin = zidan.position.clone();
                var crash = false;
                for (var i = 0 ; i < target.geometry.vertices.length; i++) {//遍历顶点

                    var one = target.geometry.vertices[i].clone();
                    var two = one.applyMatrix4(target.matrix);
                    var vector = two.sub(origin.position);

                    var ray = new THREE.Raycaster(origin, vector.clone().normalize());
                    var collision = ray.intersectObjects(o);
```

```
                    if (collision.length>0 &&
collision[0].distance < vector.length()) {
                        crash = true; // crash 是一个标记变量
                    }
                    return crash;
                }
```

6.利用 threejs 设计实现一个函数曲线的绘制

```
var geometry = new THREE.Geometry();
//三维样条曲线
var curve = new THREE.CatmullRomCurve3([
    new THREE.Vector3(0, 0, 90),
    new THREE.Vector3(-10, 40, 40),
    new THREE.Vector3(0, 0, 0),
    new THREE.Vector3(60, -60, 0),
    new THREE.Vector3(70, 0, 80)
]);
var yarray = [];
for (i =0,i<n;i++){
    yarray.push[Math.sin(i/n)]
}
var curve = new Three.SplineCurve([
    new THREE.Vector3(0, yarray[0]),
```

```
    new THREE.Vector3(1/n, yarray[1]),
    ....
])
//二维样条
var points = curve.getPoints(100);
//圆弧曲线
var arc = new THREE.ArcCurve(0, 0, 100, 0, 2 * Math.PI);
var points = arc.getPoints(50);//分段数 50，返回 51 个顶点

//取点
geometry.setFromPoints(points);
var material = new THREE.LineBasicMaterial({
    color: 0xff00f0
});
var line = new THREE.Line(geometry, material);
scene.add(line);
```

7.利用 threejs 设计实现一个曲面的绘制

```
//Shape 对象
var shape = new THREE.Shape(); //Shape 对象
shape.moveTo(0,0);//起点
shape.lineTo(0,100);//第 2 点
shape.lineTo(100,100);//第 3 点
```

```
shape.lineTo(100,0);//第 4 点
shape.lineTo(0,0);//第 5 点
shape.arc(0, 0, 100, 0, 2 * Math.PI);//或调用 arc 方法直接画圆

//也可以通过顶点定义轮廓
var points = [
  new THREE.Vector2(-50, -50),
  new THREE.Vector2(-60, 0),
  new THREE.Vector2(0, 50),
  new THREE.Vector2(60, 0),
  new THREE.Vector2(50, -50),
  new THREE.Vector2(-50, -50),
]
var shape = new THREE.Shape(points);
var geometry = new THREE.ShapeGeometry(shape, 50);
var material = new THREE.MeshBasicMaterial({
    color: 0xff0000,
});
var mesh = new THREE.Mesh(geometry, material);
mesh.position.z = 100;
scene.add(mesh);
```