郭隆邦 2023-08-23

6. 视图矩阵、投影矩阵

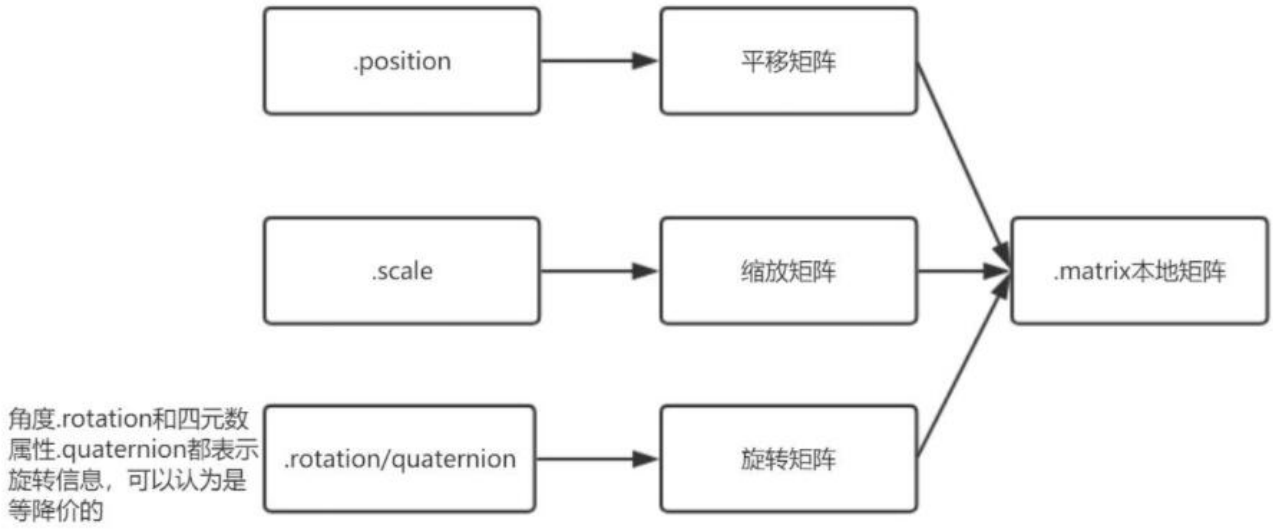
这节课给大家介绍Three.js相机对象`Camera`的两个属性**视图矩阵**`.matrixWorldInverse`和**投影矩阵**`.projectionMatrix`。

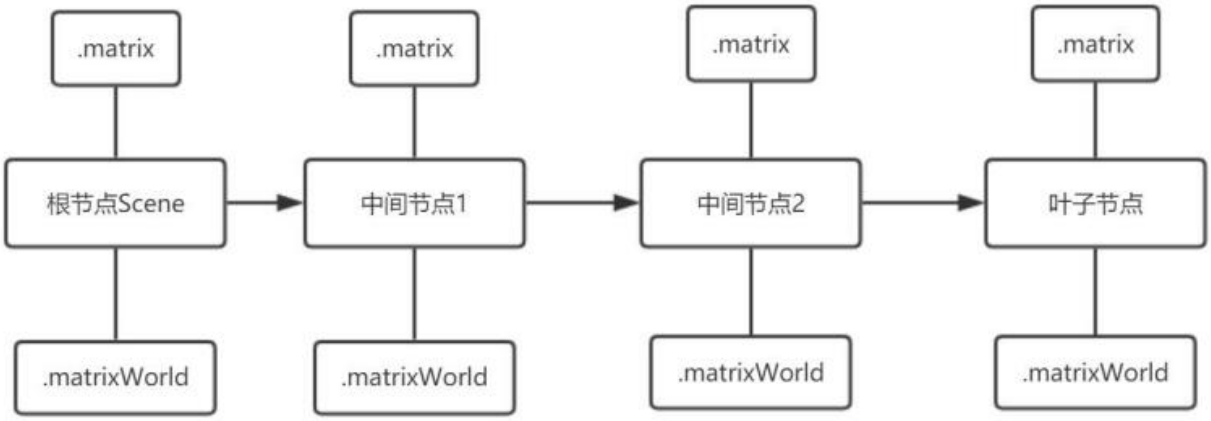
如果你有图形学基础，我提到视图矩阵或投影矩阵，你基本上都有概念，那么本节课，你可以快进学习，如果你没有相关的基础，就跟着视频通过具体theeejs代码，来认识相机矩阵相关的抽象概念。

模型矩阵知识点回顾

上节课给大家讲解过，Three.js内部渲染的时候，会把置`.position`、缩放`.scale`或角度`.rotation`(`.quaternion`)属性的值转为自己模型矩阵(本地矩阵`.matrix`、世界矩阵`.matrixWorld`)。

Three.js内部会通过模型的矩阵`.matrixWorld`旋转、缩放、平移模型自身。





根节点世界矩阵.matrixWorld = 根节点本地矩阵.matrix

中间节点1.matrixWorld = scene.matrixWorld X 中间节点1.matrix

中间节点2.matrixWorld= 中间节点1.matrixWorld X 中间节点2.matrix

叶子节点.matrixWorld= 中间节点2.matrixWorld X 叶子结点.matrix

对象世界矩阵=父对象世界矩阵 X 自身本地矩阵

相机知识点回顾

学习本节课内容之前，你可以先把以前学习的相机知识点，回顾一遍。

1.5 透视投影相机

```js
// 透视投影相机
PerspectiveCamera( fov, aspect, near, far )
```

10.1正投影相机

```js
// 正投影相机
OrthographicCamera( left, right, top, bottom, near, far )
```

相机动画(.position和.lookAt())

不同方向的投影视图

旋转渲染结果(.up相机上方向)

```js
camera.up.set(0,-1,0);
camera.position.set(292, 223, 185);
```

```
camera.lookAt(0, 0, 0);
```

两种旋转三维场景方式对比测试

改变模型或者场景自身的角度属性，旋转三维场景。

```js
// 渲染循环
function render() {
    // model.rotation.y+=0.01;
    scene.rotation.y+=0.01;
    renderer.render(scene, camera);
    requestAnimationFrame(render);
}
render()
```

改变相机位置属性`.position`让相机绕场景中心旋转，和上面代码效果相似，都是旋转整个三维场景。

你把相机的位置改变，绕着目标观察点做圆周运动，你会发现threejs场景中的模型进行了旋转，其实在threejs内部渲染过程中，threejs会获取相机参数，生成相关矩阵，对场景模型进行了旋转变换。

```js
// 渲染循环
let angle = 0; //用于圆周运动计算的角度值
const R = 260; //相机圆周运动的半径
function render() {
    angle += 0.01;
    // 相机y坐标不变，在XOZ平面上做圆周运动
    camera.position.x = R * Math.cos(angle);
    camera.position.z = R * Math.sin(angle);
    // 相机圆周运动过程，如果希望视线始终指向圆心，位置改变后必须重新执行lookAt指向圆心
    camera.lookAt(0,0,0);
    renderer.render(scene, camera);
    requestAnimationFrame(render);
}
render();
```

两种缩放三维场景方式对比测试

你想缩放整个三维场景，可以直接通过模型`.scale`属性控制

```js
// 放大工厂模型(换句话说，能观察的范围更小了，工厂周边东西不能看到那么多了)
model.scale.set(2,2,2);
```

另一方面，以透视投影相机为例，你如果改变相机的位置距离目标观察点更近，你会发现能够看到目标观察点周围的范围更小，其实本质上相当于threejs渲染时候，内部通过相机参数，生成对应矩阵，对场景进行了缩放。

```js
// 放大工厂模型(换句话说，能观察的范围更小了，工厂周边东西不能看到那么多了)
// model.scale.set(2,2,2);
//相机
const width = window.innerWidth;
const height = window.innerHeight;
const camera = new THREE.PerspectiveCamera(30, width / height, 1, 3000);
// camera.position.set(202, 123, 125);
// 相机距离目标观察点更近(能观察到范围变小，在画布上工厂放大了)
camera.position.set(202*0.5, 123*0.5, 125*0.5);
camera.lookAt(0, 0, 0);
```

测试总结

通过改变Three.js相机的参数对三维场景进行旋转、缩放或平移变换，threejs内部会获取相机参数，生成相关矩阵，对场景物体进行旋转缩放平移变换，就像**模型矩阵**对模型的旋转缩放平移变换。

咱们上面的测试，目的就是为了让大家通过具体代码测试，知道threejs相机参数的变化，本质上就是通过相机参数生成的矩阵，对场景模型进行旋转、缩放、平移。至于具体影响，下面会给大家说明。

相机视图矩阵 `.matrixWorldInverse`

在three.js内部，threejs会把相机的位置`.position`、`lookAt`指向**目标观察点**、上方向`.up`，生成一个视图矩阵`.matrixWorldInverse`,在threejs渲染的时候，生成的视图矩阵会被用来对模型顶点进行几何变换。

相机投影矩阵 `.projectionMatrix`

影响透视投影相机投影矩阵属性的参数`( fov, aspect, near, far )`

```js
// 透视投影相机
PerspectiveCamera( fov, aspect, near, far )
```

影响正投影相机投影矩阵属性的参数`( left, right, top, bottom, near, far )`

```js
// 正投影相机
OrthographicCamera( left, right, top, bottom, near, far )
```

更新透视投影矩阵`.updateProjectionMatrix()`

在Three.js内部，渲染期间，透视投影矩阵threejs并不会始终读取相机的参数，计算，这样太浪费CPU计算资源了，为了性能考虑，threejs默认就是计算一次生成投影矩阵的值，所以如果你因为某种需要，改变了相机的相关参数，就要执行`.updateProjectionMatrix()`告诉threejs重新合成透视投影矩阵的值`.projectionMatrix`。

```js
// onresize 事件会在窗口被调整大小时发生
window.onresize = function () {
    // 重置渲染器输出画布canvas尺寸
    renderer.setSize(window.innerWidth, window.innerHeight);
    // 全屏情况下：设置观察范围长宽比aspect为窗口宽高比
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
};
```

扩展：矩阵对顶点变换

大家都知道模型本质上都是由几何体的顶点构成的，threejs渲染的时候，内部会读取模型和相机的矩阵属性，对顶点进行几何变换，具体应用案例咱们在后面shader课程中给大家讲解。

```js
投影矩阵 * 视图矩阵 * 模型矩阵 * 模型顶点坐标
```

← 5. 模型本地矩阵、世界矩阵

1. 射线Ray(复习前面内容)→