

🔹 8. 管道漫游案例

通过一个轨迹线生成一个管道几何体，然后相机沿着该轨迹线移动，注意相机的方向要沿着轨迹线的切线方向，这样会形成一个管道漫游的效果。

- 管道几何体 `TubeGeometry` 、纹理贴图
- 相机对象 `Camera` 的 `.position` 属性和 `.lookAt()` 方法



管道模型

课件源码“演示”文件中提前给大家提供了一个管道模型，大家可以在此基础上写代码，生成相机在管道内漫游移动的动画。

你可以尝试自己利用前面学习过的知识，创建这样一个管道模型，再学习下面相机动画的讲解。

```
// 三维样条曲线
const path = new THREE.CatmullRomCurve3([
```

```

    new THREE.Vector3(-50, 20, 90),
    new THREE.Vector3(-10, 40, 40),
    new THREE.Vector3(0, 0, 0),
    new THREE.Vector3(60, -60, 0),
    new THREE.Vector3(90, -40, 60),
    new THREE.Vector3(120, 30, 30),
  ]);
// 样条曲线path作为TubeGeometry参数生成管道
const geometry = new THREE.TubeGeometry(path, 200, 5, 30);
const texLoader = new THREE.TextureLoader();
//纹理贴图
const texture = texLoader.load('./diffuse.jpg');
//UV坐标U方向阵列模式
texture.wrapS = THREE.RepeatWrapping;
//纹理沿着管道方向阵列(UV坐标U方向)
texture.repeat.x = 10;
const material = new THREE.MeshLambertMaterial({
  map: texture,
  side: THREE.DoubleSide, //双面显示看到管道内壁
});
const mesh = new THREE.Mesh(geometry, material);

```

相机选择

为了模拟人眼观察世界的规律，管道漫游选择透视投影相机，而不是正投影相机。

获得运动轨迹上的顶点

通过曲线的 `.getSpacedPoints()` 方法可以从轨迹线上均匀的获得一系列顶点坐标数据，然后你可以用这些轨迹线上顶点坐标设置相机位置。

```

// 从曲线上等间距获取一定数量点坐标
const pointsArr = path.getSpacedPoints(500);

```

相机放在管道内轨迹线上

相机放在管道内轨迹线上任意一个位置,并控制相机视线和曲线切线重合。

- 曲线当前点 `pointsArr[i]`
- 曲线下一个点 `pointsArr[i + 1]`

曲线上当前点 `pointsArr[i]` 和下一个点 `pointsArr[i+1]` 近似模拟当前点曲线切线，两点间距越小，模拟精度越高。

`.lookAt()` 设置相机观察点为当前点 `pointsArr[i]` 的下一个点 `pointsArr[i + 1]`，使相机视线和曲线上当前点切线重合。

```
// 从曲线上等间距获取一定数量点坐标
const pointsArr = path.getSpacedPoints(500);
const i = 100;
// 相机位置：曲线上当前点pointsArr[i]
camera.position.copy(pointsArr[i]);
// 相机观察目标：当前点的下一个点pointsArr[i + 1]
camera.lookAt(pointsArr[i + 1]);
```

改变视场角度 `fov` 调节渲染效果

你可以比较相机视锥体不同视场角度 `fov` 对应的视觉效果。

```
// fov:90度
const camera = new THREE.PerspectiveCamera(90, width / height, 1, 3000);
```

```
// fov:30度
const camera = new THREE.PerspectiveCamera(30, width / height, 1, 3000);
```

相机控件 `.target` 和 `.lookAt()` 参数一致

相机控件 `.target` 和 `.lookAt()` 参数同步，这样你可以旋转相机观察管道内部。

```
const controls = new OrbitControls(camera, renderer.domElement);
controls.target.copy(pointsArr[i+1]);
controls.update();
```

相机动画完整代码

```
// 从曲线上等间距获取一定数量点坐标
const pointsArr = path.getSpacedPoints(500);
```

```
// 渲染循环
let i = 0; //在渲染循环中累加变化
function render() {
    if (i < pointsArr.length - 1) {
        // 相机位置设置在当前点位置
        camera.position.copy(pointsArr[i]);
        // 曲线上当前点pointsArr[i]和下一个点pointsArr[i+1]近似模拟当前点曲线切线
        // 设置相机观察点为当前点的下一个点，相机视线和当前点曲线切线重合
        camera.lookAt(pointsArr[i + 1]);
        i += 1; //调节速度
    } else {
        i = 0
    }
    renderer.render(scene, camera);
    requestAnimationFrame(render);
}
render();
```

渲染方式

可以使用点模型 `Points` 渲染，也可以使用网格模型 `Mesh` 渲染。使用网格的话可以实现一个相机在管道内部漫游的效果，使用点模式渲染，可以看到一个虫洞特效，当然你也可以从管道几何体获得顶点数据来设置精灵的位置，多少个顶点，多少个精灵模型对象，也可以实现一个虫洞效果。

← 7. 旋转渲染结果(.up相机上方向)

9. OrbitControls旋转缩放限制→