## 1.在threejs中也可以使用着色器程序，如果要用着色器程序，物体的材质需要使用：ShaderMaterial，然后资源在创建材质的时候把顶点着色器程序和片元着色器程序的源码以字符串的形式提供

|  |
| --- |
|  |

### 其他和普通的物体的创建方式类似，也是需要几何体和材质来创建物体

## 2.代码示例1，App.jsx中的main1函数

## src/lib/threeapp.js

|  |
| --- |
| import \* as THREE from 'three'  import { OrbitControls } from './addonutil'  import Stats from 'three/examples/jsm/libs/stats.module';  //注意：这是第二课的class，内容有点不一样所以我们把class名称改为ThreeApp2  //帧率使用聚光灯而不是平行光  export default class ThreeApp{      constructor(canvasId){          //1.定义场景，相机，渲染器          this.scene = undefined          this.camera = undefined          this.renderer = undefined          //2.定义相机参数,但是保存在本类中          this.fov = 45          this.nearPlane = 1          this.farPlane = 1000          this.canvasId = canvasId          //3.定义额外组件          this.clock = undefined          this.controls = undefined          this.stats = undefined          //4.定义环境光和聚光灯          this.ambientLight = undefined          this.directionalLight = undefined      }      initApp(){          //创建场景对象并且赋值给成员变量          this.scene = new THREE.Scene()          //创建相机对象并且用相机成员变量接收          this.camera = new THREE.PerspectiveCamera(              this.fov,              window.innerWidth/window.innerHeight,              this.nearPlane,              this.farPlane          )          //调整相机位置          this.camera.position.z = 48 //cameraz轴值越大，图像越小          //根据传入的id获取画布对象          let canvas = document.getElementById(this.canvasId)          //创建渲染器          this.renderer = new THREE.WebGLRenderer({               canvas,               antialias:true           })            //设置渲染器的渲染尺寸          this.renderer.setSize(window.innerWidth,window.innerHeight)          //添加到body中          document.body.appendChild(this.renderer.domElement)          //创建时钟，轨道控制器，检测帧数(FPS)的工具          this.clock = new THREE.Clock()          this.controls = new OrbitControls(this.camera,this.renderer.domElement)          this.stats = Stats()          //将检测帧数(FPS)的工具添加到body中          document.body.appendChild(this.stats.domElement)          //创建环境光          this.ambientLight = new THREE.AmbientLight(0xffffff,0.5)          this.ambientLight.castShadow = true          //把环境光添加到场景中          this.scene.add(this.ambientLight)          //创建方向光          this.directionalLight = new THREE.SpotLight(0xffffff,1)          //设置方向光的位置          this.directionalLight.position.set(0,32,64) //位置也不一样            //添加方向光到场景中          this.scene.add(this.directionalLight)          //给window对象添加事件监听，用来实现窗口的响应式功能          window.addEventListener('resize',()=>this.onWindowResize(),false)      }        //定义animate成员函数，这个函数在外部调用      animate(){          window.requestAnimationFrame(this.animate.bind(this))          this.render()          this.stats.update()          this.controls.update()      }      render(){          this.renderer.render(this.scene,this.camera)      }      onWindowResize(){          //重新计算相机的宽高比          this.camera.aspect = window.innerWidth/window.innerHeight          //更新相机的投影矩阵          this.camera.updateProjectionMatrix()          //重新设置渲染器的渲染大小          this.renderer.setSize(window.innerWidth,window.innerHeight)      }  } |

## src/lib/addonutil.js,用来方便我们从examples里面导入资源，方便编程，有需要的话还会扩充

|  |
| --- |
| import {GUI} from 'three/examples/jsm/libs/lil-gui.module.min.js'  import {OrbitControls} from 'three/examples/jsm/controls/OrbitControls.js'  import {GLTFLoader} from 'three/examples/jsm/loaders/GLTFLoader.js'  import {FBXLoader} from 'three/examples/jsm/loaders/FBXLoader.js'  import {VOXLoader} from 'three/examples/jsm/loaders/VOXLoader.js'  import {TTFLoader} from 'three/examples/jsm/loaders/TTFLoader.js'  import { FontLoader } from 'three/examples/jsm/loaders/FontLoader';  import { TextGeometry } from 'three/examples/jsm/geometries/TextGeometry';  //很简单就是把这些类导入进来又导出，做的编写一次，使用多次的目的  export {    GUI,OrbitControls,GLTFLoader,FBXLoader,VOXLoader,TTFLoader,FontLoader,TextGeometry  } |

## src/App.jsx

|  |
| --- |
| import { useEffect } from 'react'  import './App.css'  import \* as THREE from 'three'  import ThreeApp from './lib/threeapp'  function main1() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            position.x,position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)  }  function main2() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            position.x,sin(position.z),position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)  }  function main3() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            position.x,sin(position.z)+position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)  }  function main4() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(position.x, 4.0\*sin(position.z/4.0) + position.y, position.z, 1.0);         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)  }  function main5() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            sin(position.x),position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)    }  function main6() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            sin(position.z)+position.x,position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)  }  function App() {    useEffect(() => {      main1()      // main2()      // main3()      // main4()      // main5()      // main6()    }, [])    return (      <>        <div>          <canvas id="myThreeJSCanvas"></canvas>        </div>      </>    )  }  export default App |

### 效果：

|  |
| --- |
|  |

## 3.代码示例2.App.jsx中的main2函数

|  |
| --- |
| function main2() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            position.x,sin(position.z),position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)    //添加鼠标移动事件监听    window.addEventListener('mousemove', onMouseMove)    // function animate() {    //   if (loadedModel) {    //     loadedModel.scene.rotation.x += 0.01    //     loadedModel.scene.rotation.y += 0.01    //     loadedModel.scene.rotation.z += 0.01    //   }    //   requestAnimationFrame(animate);    // }    // animate()  } |

### 效果：

|  |
| --- |
|  |

## 4.示例3.代码只有shader源码不一样，所以我这里只记录main3函数的代码

|  |
| --- |
| function main3() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            position.x,sin(position.z)+position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)    //添加鼠标移动事件监听    window.addEventListener('mousemove', onMouseMove)    } |

### 效果：

|  |
| --- |
|  |

## 5.示例4.main4函数的代码如下

|  |
| --- |
| function main4() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            position.x,4\*sin(position.z/4.0)+position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)    //添加鼠标移动事件监听    window.addEventListener('mousemove', onMouseMove)  } |

### 效果

|  |
| --- |
|  |

## 6.示例5.main5函数的内容如下

|  |
| --- |
| function main5() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            sin(position.x),position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)    //添加鼠标移动事件监听    window.addEventListener('mousemove', onMouseMove)  } |

### 效果

|  |
| --- |
|  |

## 7.示例7.main6函数

|  |
| --- |
| function main6() {    //创建three应用程序对象    let threeApp = new ThreeApp("myThreeJSCanvas")    //初始化场景    threeApp.initApp()    //调用动画渲染效果    threeApp.animate()    let boxGeo = new THREE.BoxGeometry(16, 16, 16, 16, 16, 16)    let boxMat = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `         void main(){          gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(            sin(position.z)+position.x,position.y,position.z,1.0          );         }      `,      fragmentShader: `        void main(){          gl\_FragColor = vec4(1.0,0.0,0.0,1.0);        }      `    })    let box = new THREE.Mesh(boxGeo, boxMat)    threeApp.scene.add(box)  } |

### 效果：

|  |
| --- |
|  |