# 这一节在上一节的基础上解析扩展，可以做一些比较酷炫的动画效果

## 示例1. 主要代码是App.jsx中的main1函数，代码如下

|  |
| --- |
| 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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `        void main() {          vec4 result;          result = vec4(position.x, position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        // varying vec3 pos;        void main() {          gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：

|  |
| --- |
|  |

## 示例2.App.jsx的main2函数，代码如下：

|  |
| --- |
| 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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 1.re-write boiler plate code with shader          result = vec4(position.x, position.y, position.z, 1.0);          result = vec4(position.x, position.y + sin(u\_time), position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：会上下移动的长方体线框盒子

|  |
| --- |
|  |

## 示例3.main3函数

|  |
| --- |
| 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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;           //3. convert box into a 2D sine wave plane            result = vec4(position.x, sin(position.z), position.z, 1.0);            result = vec4(position.x, sin(position.z + u\_time), position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：会运动的波浪平面

|  |
| --- |
|  |

## 示例4.main4函数

|  |
| --- |
| 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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 4.change the 2D sine wave plane into a wavy box          result = vec4(position.x, sin(position.z) + position.y, position.z, 1.0);          result = vec4(position.x, sin(position.z + u\_time) + position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：运动的波浪盒子

|  |
| --- |
|  |

## 示例5.main5函数

|  |
| --- |
| 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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 5.change how wavy the box is by updating frequency          result = vec4(position.x, sin(position.z/4.0) + position.y, position.z, 1.0);          result = vec4(position.x, sin((position.z)/4.0 + u\_time) + position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：扭曲运动盒子

|  |
| --- |
|  |

## 示例6.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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 6change the amplitude of the box's waves          result = vec4(position.x, 4.0\*sin(position.z/4.0) + position.y, position.z, 1.0);          // result = vec4(position.x, 4.0\*sin(position.z/4.0 + u\_time) + position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：扭曲的盒子，会出现和消失

|  |
| --- |
|  |

## 示例7，，主要代码是App.jsx中的main7函数，代码如下

|  |
| --- |
| 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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // varying variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        varying vec3 pos;        uniform float u\_time;        void main() {          vec4 result;          pos = position;          result = vec4(            position.x,            4.0\*sin(position.z/4.0 + u\_time) + position.y,            position.z,            1.0          );          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        varying vec3 pos;        uniform float u\_time;        void main() {          if (pos.x >= 0.0) {            // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);            gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);          } else {            // gl\_FragColor = vec4(0.0, 1.0, 0.0, 1.0);            gl\_FragColor = vec4(0.0, abs(cos(u\_time)), 0.0, 1.0);          }        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：红绿波浪效果线框盒子

|  |
| --- |
|  |
|  |

# 效果还蛮多的，需要慢慢理解

# 完整源码

## src/lib/addonutil.js

|  |
| --- |
| 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/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/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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      vertexShader: `        void main() {          vec4 result;          result = vec4(position.x, position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        // varying vec3 pos;        void main() {          gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }  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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 1.re-write boiler plate code with shader          result = vec4(position.x, position.y, position.z, 1.0);          result = vec4(position.x, position.y + sin(u\_time), position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }  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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;           //3. convert box into a 2D sine wave plane            result = vec4(position.x, sin(position.z), position.z, 1.0);            result = vec4(position.x, sin(position.z + u\_time), position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }  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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 4.change the 2D sine wave plane into a wavy box          result = vec4(position.x, sin(position.z) + position.y, position.z, 1.0);          result = vec4(position.x, sin(position.z + u\_time) + position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }    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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 5.change how wavy the box is by updating frequency          result = vec4(position.x, sin(position.z/4.0) + position.y, position.z, 1.0);          result = vec4(position.x, sin((position.z)/4.0 + u\_time) + position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }  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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // glsl shader with uniform variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        uniform float u\_time;        void main() {          vec4 result;          // 6change the amplitude of the box's waves          result = vec4(position.x, 4.0\*sin(position.z/4.0) + position.y, position.z, 1.0);          // result = vec4(position.x, 4.0\*sin(position.z/4.0 + u\_time) + position.y, position.z, 1.0);          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        uniform float u\_time;        void main() {          // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);          // gl\_FragColor = vec4(sin(u\_time), 0.0, 0.0, 1.0);          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }  function main7() {    //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()    //axeshelper    const axesHelper = new THREE.AxesHelper(16);    threeApp.scene.add(axesHelper);    // define uniform data    const uniformData = {      u\_time: {        type: 'f',        value: threeApp.clock.getElapsedTime(),      },    };    const render = () => {      uniformData.u\_time.value = threeApp.clock.getElapsedTime();      window.requestAnimationFrame(render);    };    render();    // varying variables    const boxGeometry = new THREE.BoxGeometry(24, 4, 24, 24, 4, 24);    const boxMaterial = new THREE.ShaderMaterial({      wireframe: true,      uniforms: uniformData,      vertexShader: `        varying vec3 pos;        uniform float u\_time;        void main() {          vec4 result;          pos = position;          result = vec4(            position.x,            4.0\*sin(position.z/4.0 + u\_time) + position.y,            position.z,            1.0          );          gl\_Position = projectionMatrix            \* modelViewMatrix            \* result;        }        `,      fragmentShader: `        varying vec3 pos;        uniform float u\_time;        void main() {          if (pos.x >= 0.0) {            // gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);            gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);          } else {            // gl\_FragColor = vec4(0.0, 1.0, 0.0, 1.0);            gl\_FragColor = vec4(0.0, abs(cos(u\_time)), 0.0, 1.0);          }        }        `,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  }  function App() {    useEffect(() => {      main1()      // main2()      // main3()      // main4()      // main5()      // main6()      // main7()    }, [])    return (      <>        <div>          <canvas id="myThreeJSCanvas"></canvas>        </div>      </>    )  }  export default App |