# 在我们以前的程序中，着色器源码都是以字符串的形式编写，很麻烦，我们其实可以使用vscode的webGL glsl插件来帮助我们编写并且导入着色器程序

# 项目结构

|  |
| --- |
|  |

## 1.安装插件，打开vscode，点击扩展按钮，在搜索框中输入webgl glsl，就会找下面的插件

|  |
| --- |
|  |

## 2.点击安装，就可以安装了

## 3.需要注意的是我们可以利用这个插件很方便的编辑着色器源码，但是我们不能在js中直接导入glsl文件，此时如果你使用vite作为构建工具，我们需要一个插件：

### 1.安装vite-plugin-glsl，可以使用npm来安装：npm i vite-plugin-glsl --save-dev

## 2.编辑vite.config.js文件，添加如下配置

|  |
| --- |
|  |

## 3.然后我们在src文件夹里面添加一个a\_vertex.glsl文件，内容如下

|  |
| --- |
|  |

## 4.在src文件夹里面新建2个fragment着色器文件，因为我们想测试2种情况

### a\_frag.glsl

|  |
| --- |
|  |

### a\_frag2.flsl

|  |
| --- |
|  |

## 5.然后就可以在App.jsx中导入我们写的shader程序

|  |
| --- |
|  |

## 6、导入后就可以把它传递给ShaderMaterial的构造函数，由于我们上一节有很多shader源码，我们需要很多给main函数来测试，我们的main0的源码如下

|  |
| --- |
| function main0() {    //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: vertexShaderSource,      fragmentShader: fragShaderSource1,    });    const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);    threeApp.scene.add(boxMesh);  } |

### 效果：波浪形的盒子

|  |
| --- |
|  |

# 项目重组

# 为了更好的学习，我们把上一节的项目重组一下，不要在App.jsx里面写非常多的代码，我们采用分文件，然后导入的方式，我们插件一个shadertests文件夹，把glsl文件和函数文件写在这里

## 1.在src文件夹里面新建一个shadertests，然后在里面新建一个test1文件夹，然后把这两个glsl文件剪切进来，再新建两个js文件，副本使用这两个fragment shader 文件

|  |
| --- |
|  |

### shadertest1\_a.js

|  |
| --- |
| import \* as THREE from 'three'  import ThreeApp from '../../lib/threeapp'  import vertexShaderSource from './a\_vertex.glsl'  import fragShaderSource1 from './a\_frag.glsl'  export function shaderTest1a(){     //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: vertexShaderSource,       fragmentShader: fragShaderSource1,     });     const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);     threeApp.scene.add(boxMesh);  } |

### shadertest1\_b.js

|  |
| --- |
| import \* as THREE from 'three'  import ThreeApp from '../../lib/threeapp'  import vertexShaderSource from './a\_vertex.glsl'  import fragShaderSource2 from './a\_frag2.glsl'  export function shaderTest1b(){     //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: vertexShaderSource,       fragmentShader: fragShaderSource2,     });     const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);     threeApp.scene.add(boxMesh);  } |

## 2.然后在App.jsx里面导入这些文件，我们先测试shadertest1\_a.js

|  |
| --- |
| import { useEffect } from 'react'  import './App.css'  import { shaderTest1a } from './shadertests/test1/shadertest1\_a'  function App() {    useEffect(() => {      shaderTest1a()    }, [])    return (      <>        <div>          <canvas id="myThreeJSCanvas"></canvas>        </div>      </>    )  }  export default App |

### 效果和上面是一样的

|  |  |
| --- | --- |
| 波浪形的盒子  |  | | --- | |  | |

## 3.然后我们来测试shadertest1\_b.js

|  |
| --- |
|  |

### 效果：图形是一样的，但是它红色和绿色部分会交替消失和出现

## 4.在shadertests文件夹里面新建一个test2文件夹，添加下面的文件

### t2\_frag.glsl

|  |
| --- |
| 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);      } else {          gl\_FragColor = vec4(0.0, 1.0, 0.0, 1.0);      }  } |

### t2\_frag2.glsl

|  |
| --- |
| varying vec3 pos;  uniform float u\_time;  void main() {      if(pos.x >= 0.0) {          gl\_FragColor = vec4(abs(sin(u\_time)), 0.0, 0.0, 1.0);      } else {          gl\_FragColor = vec4(0.0, abs(cos(u\_time)), 0.0, 1.0);      }  } |

### t2\_vertex.glsl

|  |
| --- |
| varying vec3 pos;  uniform float u\_time;  void main() {      vec4 result;      pos = position;      result = vec4(position.x, sin(position.z), position.z, 1.0);      result = vec4(position.x, sin(position.z + u\_time), position.z, 1.0);      //位置=投影矩阵\*视图模型矩阵\*result      gl\_Position = projectionMatrix \* modelViewMatrix \* result;  } |

### shadertest2\_a.js

|  |
| --- |
| import \* as THREE from 'three'  import ThreeApp from '../../lib/threeapp'  import vertexShaderSource from './t2\_vertex.glsl'  import fragShaderSource1 from './t2\_frag.glsl'  export function shaderTest2a(){     //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: vertexShaderSource,       fragmentShader: fragShaderSource1,     });     const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);     threeApp.scene.add(boxMesh);  } |

### 效果：红绿波浪平面

### shadertest2\_b.js

|  |
| --- |
| import \* as THREE from 'three'  import ThreeApp from '../../lib/threeapp'  import vertexShaderSource from './t2\_vertex.glsl'  import fragShaderSource2 from './t2\_frag2.glsl'  export function shaderTest2b(){     //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: vertexShaderSource,       fragmentShader: fragShaderSource2,     });     const boxMesh = new THREE.Mesh(boxGeometry, boxMaterial);     threeApp.scene.add(boxMesh);  } |

### 效果：红绿波浪平面两个颜色的波浪会交替消失和出现

# 总结，需要安装一个vscode webgl glsl插件为了可以比较方便的编写着色器源程序，然后安装一个vite-plugin-glsl是构建工具的插件，为了可以将我们些的glsl文件导入我们的程序中，没有这个插件是不能导入glsl文件的。