# 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