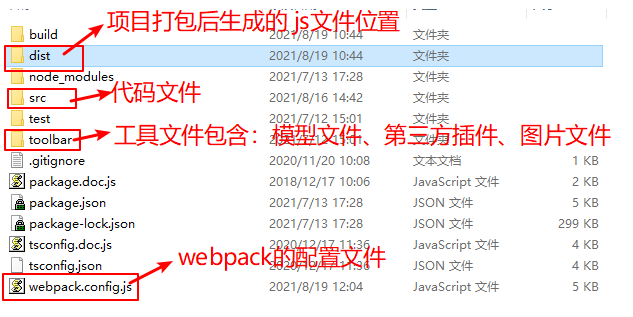
# 1、项目中相关的文件作用

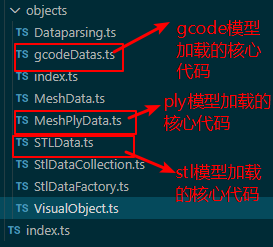
## 1.1项目文件



## 1.2代码文件



## 1.3主要功能代码

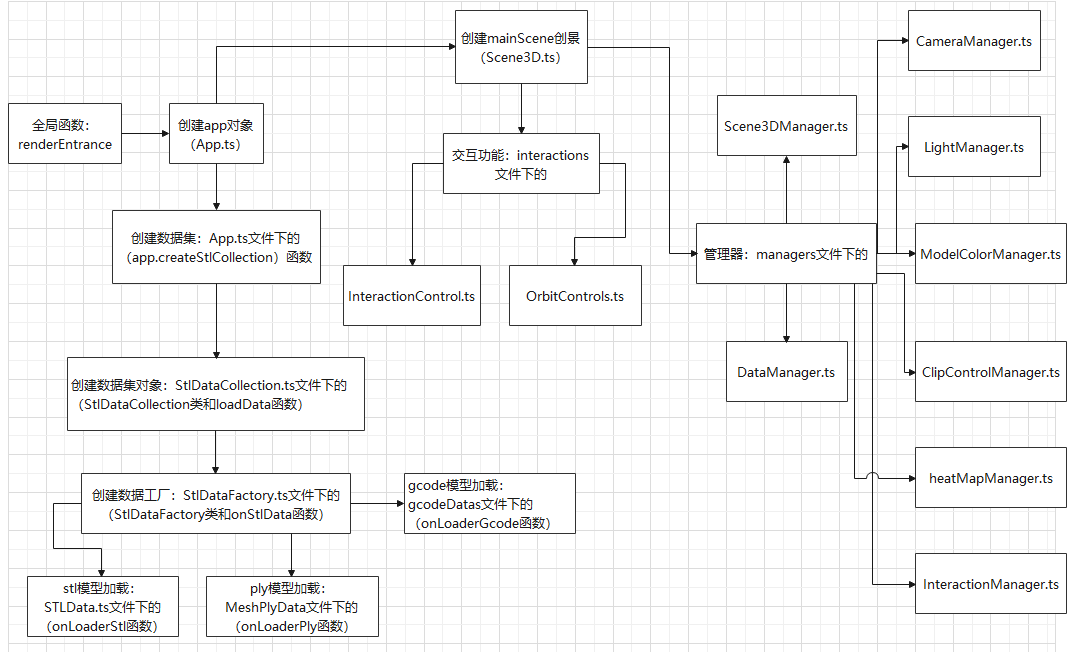
  

## 1.4相关命令

项目启动：npm run start

打包项目： npm run build

# 2、代码流程图



# 3、源码解析

## 3.1代码入口

let app = TTS.App.getInstance();//获取App静态对象

let mainScene = app.createScene('main\_scene', 'model\_view',{

        clearColor: 0x000000,

        sceneName: 'main\_scene',

 });//创建场景

var modelUrlArray1 = "https://file-cdn.creality.com/model/ply/9CD192037DA9EE438006AD6DEF8F150D.ply";

    dataConfigArray = [

        {

            id: '2',

            dataType: 'ply',(此处模型类型有：ply/stl/gcode)

            isActive: true,

            url: modelUrlArray1,

            length: 1,

            chunkSize: 500000,

            modelSize: 14

        },

    ];

var objectColss = app.createStlCollection('objCollection', dataConfigArray);//创建数据集

mainScene .bindMaxDataCollection(objectColss);//绑定数据集

## 3.2全局函数globeRenderEntrance在renderEntrance.ts文件下：此函数保证3D场景中的模型能够得到实时的渲染

import { App } from "../core/App";

function globeRenderEntrance() {

    App.getInstance().render();

let globeRenderId = requestAnimationFrame(globeRenderEntrance);

}

globeRenderEntrance();

## 3.3核心代码

### 3.3.1 App.ts代码文件：

**app就是一个window页面上的3D实例，只能被创建一次。它管理绘制场景的工具Scene3D和绘制的模型Object数据**

*/\*\**

*\* 一个window页面中对应的唯一app*

*\*/*

    private static \_appInstance: App | undefined = undefined;

    private \_config: any = undefined;

    private \_temScene3D: any = undefined;

*/\*\**

*\* 使用一个静态函数确保一个页面只能有一个app*

*\*/*

    public static getInstance(): App {

        if (this.\_appInstance == undefined) {

            this.\_appInstance = new App();

        }

        return this.\_appInstance;

    }

*/\*\**

*\* 场景管理器*

*\*/*

    private \_sceneManager: Scene3DManager | undefined | null;

*/\*\**

*\* 数据管理器*

*\*/*

    private \_dataManager: DataManager | undefined | null;

*/\*\**

*\* 将构造函数设置为private避免在外部实例化*

*\*/*

    constructor() {

        this.\_sceneManager = new Scene3DManager();

        this.\_dataManager = new DataManager();

    }

*/\*\**

*\* 得到数据集管理器(内存)*

*\*/*

    public getDataManager(): DataManager | undefined | null{

        return this.\_dataManager;

    }

*/\*\**

*\* 得到场景管理器*

*\*/*

    public getSceneManager(): Scene3DManager | undefined| null{

        return this.\_sceneManager;

    }

### 3.3.2 Scene3D.ts代码文件：

**此文件功能主要管理3D场景的绘制和模型数据的实时渲染，同时管理各个功能的管理器，控制场景中的灯光和相机，监听事件的变化**

#### 3.3.2.1场景中实时渲染的函数：render

    public render() {

        if (!this.enable) {

            return;

        }

        if (!this.\_tempConfig?.heatMapFlag) {

            if (!this.\_enable) {

                if (this.\_windowScale) {

                    this.\_enable = true;

                    this.\_windowScale = false;

                } else {

                    return

                }

            }

        }

*//设置跳帧*

        if(this.skip !== 0) {

            this.skip = ++this.skip % this.\_tempSkip;

            return;

        } else {

            this.skip = ++this.skip % this.\_tempSkip;

        }

        if (this.\_cameraManager == undefined) return;

        let activeCamera = this.\_cameraManager.getActiveCamera();

        if (activeCamera == undefined) {

            return;

        }

        this.\_cameraManager.onWindowResize();

        let flagContainer = document.getElementById(this.\_containerId);

        if (flagContainer == undefined) {

            EventControl.dispatchEvent('MSG\_Ajax\_Event\_ABORT', null, 'all');

            this.dispose();

        }

        if (this.\_interactionManager != undefined) {

            this.\_interactionManager.update();

        };

*// 实时设置模型的分辨率*

        if (this.\_meshGroup != undefined ) {

            if (this.\_meshGroup.children.length != 0) {

                let mainContainer = this.getMainContainer();

                if (this.\_meshGroup.children[0].userData.type == 'gcode') {

*//@ts-ignore*

                    this.\_meshGroup.children[0].material.resolution.set(mainContainer.clientWidth, mainContainer.clientHeight)

                }

            }

        };

        if (this.\_renderer == undefined) return;

        this.\_renderer.setRenderTarget( null );

        if (this.\_scene == undefined) return;

        this.\_renderer.render(this.\_scene, activeCamera);

*//判断模型是否已经渲染出来*

        if (this.\_renderer.info.memory.geometries > 0) {

            let scope = this;

            if (!this.\_tempConfig?.heatMapFlag) {

                if (scope.\_windowResize) {

                    this.\_time = setTimeout(function() {

                        scope.\_enable = false;

                        scope.\_windowResize = false;

                        clearTimeout(scope.\_time);

                    }, 100)

                } else {

                    scope.\_enable = false;

                }

            }

        }

        this.\_renderer.info.reset();

    }

#### 3.3.2.2绑定相关数据集的函数：bindMaxDataCollection

    public bindMaxDataCollection(objectCollection: StlDataCollection) {

        this.\_stlDataCollection = objectCollection;

        if (this.\_modelData.modelSize <= 10) {

            this.\_tempSkip = 4;

        } else if (10 < this.\_modelData.modelSize && this.\_modelData.modelSize <= 20) {

            this.\_tempSkip = 10;

        } else if (this.\_modelData.modelSize > 20) {

            this.\_tempSkip = 12;

        }

        let scope = this;

        objectCollection.onLoadMaxComplete = function() {

            let data = objectCollection.getStlData();

            if (data == undefined) return;

            if (scope.\_tempConfig?.pro3GridHelper) {

                if (data[0].userData.type == 'gcode') {

                    if (scope.\_meshGroup == undefined) return;

                    scope.\_meshGroup.translateZ(2900);

                }

            }

            if (data != undefined) {

                for (let i=0; i<data.length; ++i) {

                    if (scope.\_meshGroup == undefined) return;

                    scope.\_meshGroup.add(data[i]);

                    scope.\_enable = true;

                }

            }

        }

    }

#### 3.3.2.3 初始化各个管理器的函数：initManager

    private initManager(config: SceneConfig) {

*// 相机和光源管理器*

        this.\_cameraManager = new CameraManager(this, undefined);

*// 再交互控件管理器和普通控件管理器*

        this.\_interactionManager = new InteractionManager(this, config);

        this.\_clipControlManager = new ClipControlManager(this, config);

*//构件着色*

        this.\_modelColorManager = new ModelColorManager(this);

*//创建热力图对象*

        if (config.heatMapFlag) {

            this.\_heatMapManager = new heatMapManager(this);

        }

*//创建光源管理器*

        this.\_lightManager = new LightManager(this);

    }

## 3.4管理器代码

### 3.4.1 InteractionManager.ts代码文件：

**此文件主要管理相关的交互功能，主要涉及到相机的视角变换、模型的矩阵变换、模型的角度变换、事件的监听功能等，由于代码太多下面简单罗列一部分函数功能：**

#### 3.4.1.1初始化相机视角的功能函数：changeFov

    public changeFov() {

        let cameraManager = this.\_scene3D.getCameraManager();

        if (cameraManager == undefined) return;

        let activeCamera = cameraManager.getActiveCamera();

        if (activeCamera != undefined ) {

            let inter = this.\_scene3D.getInteractionManager();

            if (inter == undefined) return;

*//判断轨道控制器是否已经添加*

            if (inter.orbitControl != undefined) {

                if (this.\_config.pro3GridHelper) {

                    activeCamera.position.set(272.87, 235.51, 234.36);

                } else {

                  activeCamera.position.set(14.09, 175.97, 860.65);

                }

                if (this.\_scene3D.\_cameralView == null) return;

                this.\_scene3D.\_cameralView.set(activeCamera.position.x, activeCamera.position.y, activeCamera.position.z);

*// 设置焦点*

                this.orbitControl.target.set(0,0,0);

                return;

            }

*// 以后我想将第二个参数去掉，在轨道控制器中实现获取激活相机*

            this.orbitControl = new OrbitControls(this.\_scene3D, activeCamera);

            this.orbitControl.minDistance = 1.0;

            this.orbitControl.rotateSpeed = 1.5;

            if (!this.\_config.pro3GridHelper) {

                this.orbitControl.enablePan = false;

            }

*// activeCamera.position.set(14.09, 175.97, 860.65);*

            activeCamera.position.set(272.87, 235.51, 234.36);

            if (this.\_scene3D.\_cameralView == null) return;

            this.\_scene3D.\_cameralView.set(activeCamera.position.x,activeCamera.position.y, activeCamera.position.z);

*// 设置焦点*

            this.orbitControl.target.set(0,0,0);

            this.addInteraction(this.orbitControl);

        }

        EventControl.removeEvent( 'MSG\_CHANGE\_MODEL\_FOV', ()=> {this.changeFov()}, 'all');

    }

#### 3.4.1.2 模型旋转功能函数：onModelRotate

#### 3.4.1.3 模型缩放功能函数：onModelScale

#### 3.4.1.4 获取模型变换矩阵：getModelMatrixWorld

### 3.4.2 ClipControlManager.ts代码文件：

**此文件主要管理模型外部边框的构建，如3D模型在注水时的外部边框和进入创想云接口是模型预览中的外部边框；同时管理计算模型大小的函数接口等。**

#### 3.4.2.1 创建模型外边框的函数：onCreateModelBox

 private onCreateModelBox(boxSize: any) {

        if(this.\_scene3D.\_meshGroup == undefined) return;

        let Xsize = boxSize.Xsize;

        let Ysize = boxSize.Zsize;

        let Zsize = boxSize.Ysize;

*//创建模型底部的网格线框*

        if(this.\_scene3D.\_outBoxGroup == undefined) return;

        let size = Zsize, divisions = 10;

        let color1 = new THREE.Color( 0x607d8b );

        let color2 = new THREE.Color( 0x607d8b );

        let center = divisions / 2;

        let step = size / divisions;

        let vertices = [], colors: any=[];

*//X轴方向的线框数据*

        for ( let i = 0, j = 0, k = -Zsize/2; i <= divisions; i ++, k += step ) {

            vertices.push( Xsize/2, 0, k, -Xsize/2, 0, k );

            let color = i === center ? color1 : color2;

            color.toArray( colors, j ); j += 3;

            color.toArray( colors, j ); j += 3;

            color.toArray( colors, j ); j += 3;

            color.toArray( colors, j ); j += 3;

        }

        let size1 = Xsize, divisions1 = 10;

        let step1 = size1 / divisions1;

*//Y轴方向的线框数据*

        for (let i = 0, j = 0, k = -Xsize/2; i <= divisions1; i ++, k += step1) {

            vertices.push( k, 0, Zsize/2, k, 0, -Zsize/2 );

            let color = i === center ? color1 : color2;

            color.toArray( colors, j ); j += 3;

            color.toArray( colors, j ); j += 3;

            color.toArray( colors, j ); j += 3;

            color.toArray( colors, j ); j += 3;

        }

*//创建线框几何体*

        let geometryd = new THREE.BufferGeometry();

        geometryd.setAttribute( 'position', new THREE.Float32BufferAttribute( vertices, 3 ) );

        geometryd.setAttribute( 'color', new THREE.Float32BufferAttribute( colors, 3 ) );

        let material = new THREE.LineBasicMaterial( { vertexColors: true, toneMapped: false, linewidth: 1 } );

        let gridHelper = new THREE.LineSegments( geometryd, material );

        gridHelper.renderOrder = 3;

        gridHelper.position.y = -Ysize/2;

        this.\_scene3D.\_outBoxGroup.add( gridHelper );

*//调整模型的位置*

        EventControl.dispatchEvent('MSG\_MODEL\_POSITION\_COUNT', 1, 'all');

    }

#### 3.4.2.2 计算模型大小函数：onCalculationModelSize

public onCalculationModelSize(boxSize: any) {

        if (boxSize == undefined) return;

*//判断是否显示包围盒*

        if (this.\_config.boxHelper) {

*//改变机型设置模型外框*

            this.onChangeModelBoxSize();

            if(this.\_scene3D.\_outBoxGroup == undefined) return;

            let Xsize = boxSize.Xsize;

            let Ysize = boxSize.Zsize;

            let Zsize = boxSize.Ysize;

            let colors = [];

            for (let i=0; i<24; ++i) {

                if (i>= 12 && i<=13) {

*//X轴坐标颜色*

                    colors.push(0.89, 0.03, 0.83);

                } else if (i>= 4 && i<=5) {

*//Y轴坐标颜色*

                    colors.push(0.05, 0.89, 0.91);

                } else if (i>= 8 && i<=9) {

*//Z轴坐标颜色*

                    colors.push(0, 0, 1);

                }else {

                    colors.push(.9, .9, .1);

                }

            }

            let color = new Float32Array(colors);

            let edgesGeometry = new THREE.BoxGeometry( Xsize, Ysize, Zsize );

            let edges = new THREE.EdgesGeometry(edgesGeometry);

            edges.setAttribute('color', new THREE.BufferAttribute(color, 3));

            this.\_lineBox = new THREE.LineSegments( edges, new THREE.LineBasicMaterial({ vertexColors: true, linewidth: 3}));

            this.\_lineBox.name = "lineBox";

            this.\_lineBox.visible = true;

            this.\_lineBox.castShadow = true;

            this.\_lineBox.renderOrder = 99;

            this.\_scene3D.\_outBoxGroup.add( this.\_lineBox );

            if(this.\_scene3D.\_meshGroup == undefined) return;

            let meshS = new THREE.Box3().setFromObject(this.\_scene3D.\_meshGroup);

            let meshSize = meshS.getSize(new THREE.Vector3());

            let meshSizeC = {'x': parseFloat(meshSize.x.toFixed(2)), 'y': parseFloat(meshSize.z.toFixed(2)), 'z': parseFloat(meshSize.y.toFixed(2))}

*//计算模型尺寸*

            let rangeX = (1 / meshSize.x) \* 100;

            let xMin = rangeX;

            let rangeY = (1 / meshSize.y) \* 100;

            let yMin = rangeY;

            let rangeZ = (1 / meshSize.z) \* 100;

            let zMin = rangeZ;

            let rMin = Math.max(xMin, yMin, zMin);

*//计算模型的缩放范围*

            let Xrange = ( Xsize / meshSize.x ) \* 100;

            let Yrange = ( Ysize / meshSize.y ) \* 100;

            let Zrange = ( Zsize / meshSize.z ) \* 100;

            let minRange = Math.min(Xrange, Yrange, Zrange);

            let rangeNum = parseFloat(minRange.toFixed(2));

            let xRangeNum = parseFloat(Xrange.toFixed(2));

            let yRangeNum = parseFloat(Yrange.toFixed(2));

            let zRangeNum = parseFloat(Zrange.toFixed(2));

            let tempData = {size: meshSizeC, scaleRange: {min: parseFloat(rMin.toFixed(2)) , max: parseFloat(rangeNum.toFixed(2)) },

            boxSize: {x: Xsize, y: Ysize, z: Zsize}, scaleRangeX: {min: parseFloat(xMin.toFixed(2)), max: parseFloat(xRangeNum.toFixed(2)) },

            scaleRangeY: {min: parseFloat(zMin.toFixed(2)), max: parseFloat(zRangeNum.toFixed(2)) },

            scaleRangeZ: {min: parseFloat(yMin.toFixed(2)) , max: parseFloat(yRangeNum.toFixed(2)) }, modeSize: this.\_modelSize};

            this.\_tempData = tempData;

*//云切片功能中返回模型尺寸事件*

            EventControl.dispatchEvent('MSG\_GET\_MODEL\_SIZE', this.\_tempData, '3D');

            if(this.\_scene3D.\_meshGroup == undefined) return;

            if (this.\_scene3D.\_meshGroup.children.length == 0) return;

            let box = new THREE.Box3().expandByObject(this.\_scene3D.\_meshGroup.children[0]);

            let centerLen = new THREE.Vector3().subVectors(box.max, box.min).multiplyScalar(0.5);

*//设置模型组的位置坐标*

            this.\_scene3D.\_meshGroup.position.y = -(Ysize/2 -centerLen.y);

*//根据不同的长宽高创建模型的打印台*

            this.onCreateModelBox(boxSize);

        }

    }

### 3.4.3 ModelColorManager.ts：代码文件

**此文件功能主要管理模型的效果设置包括：模型的颜色设置、透明值设置和在打印状态下的注水效果的设置。**

#### 3.4.3.1 模型打印状态下的注水函数：setIndexModelAlph

public setIndexModelAlph(zData: any) {

        if (zData == undefined || zData == "") return;

        let key = parseFloat(zData);

        let modelData = this.\_scene3D.getStlDataModel();

        if (modelData != undefined) {

            if (modelData[0] == undefined) return;

*//获取模型的类型*

            let type = modelData[0].userData.type;

            if (type == 'stl') {

                if (this.\_scene3D.\_temGroup == undefined) return;

                let box = new THREE.Box3().expandByObject(this.\_scene3D.\_temGroup);

                let center = new THREE.Vector3().subVectors(box.max, box.min);

                let len = this.\_scene3D.\_temGroup.children.length;

                for (let i=0; i<len; ++i) {

                    let child = this.\_scene3D.\_temGroup.children[i];

                    if (child.name == 'print\_stl') {

                        child.visible = true;

                        child.position.y = (-center.y/2 + 0.1 + Math.abs(key));

                    }

                    if (child.name == 'outMesh') {

                        child.visible = true;

*//@ts-ignore*

                        let vertices = child.geometry.vertices;

                        if (zData == "finished") {

                             EventControl.removeEvent('MSG\_RENDER\_MODEL\_LAYER\_PRINTER', ()=> {}, '3D');

                             let outMaterial = new THREE.MeshStandardMaterial( {

                                 color: 0x00CFEE,

                                 roughness: 0.5,

                                 emissive: 0x000000,

                                 metalness: 0.5,

                                 transparent: true,

                                 opacity: 0.8,

                             } );

*//@ts-ignore*

                             child.material = outMaterial;

                             return;

                        }

                        for (let j=0; j<vertices.length; ++j) {

                            if (vertices[j].y < 0) {

                                vertices[j].y = -Math.abs(key)/2

                            } else {

                                vertices[j].y = Math.abs(key)/2;

                            }

                        }

*// @ts-ignore*

                        child.geometry.verticesNeedUpdate = true;

                        child.position.y = (-center.y/2 + 0.1 + Math.abs(key)/2);

                    }

                }

            } else if (type == 'gcode') {

                if (zData == "finished") {

                    this.clearPrinterLayerColor();

                    EventControl.removeEvent('MSG\_RENDER\_MODEL\_LAYER\_PRINTER', ()=> {}, '3D');

                    if (this.\_scene3D.\_meshGroup == undefined) return;

                    let children = this.\_scene3D.\_meshGroup.children;

                    if (children.length != 0) {

*//@ts-ignore*

                        children[0].material.depthWrite = true;

                    }

                    return;

                }

                let indexData = this.\_scene3D.getIndexGcodeData();

                for (let i=0; i<modelData.length; ++i) {

                    if (indexData[key] == undefined) {

                        return;

                    };

*//清除打印时的高亮颜色*

                    this.clearPrinterLayerColor();

                    let cStart = indexData[key].start;

                    let cEend = indexData[key].end;

                    this.\_printerColorData[key]={

                        "start": cStart,

                        "end": cEend

                    }

*//计算透明值的开始索引和结束索引*

                    let startAlphData = (cStart \* 3 / 6);

                    let endAlphData = (cEend \* 3 / 6);

*//@ts-ignore*

                    let transparentAttribute = modelData[i].geometry.getAttribute('instanceAlphStart');

                    for (let i = startAlphData; i < endAlphData; ++i) {

                        transparentAttribute.data.array[i] = 1.0;

                    }

*// @ts-ignore*

                    transparentAttribute.needsUpdate = true;

*// @ts-ignore*

                    let colorsAttribute = modelData[i].geometry.getAttribute('instanceColorStart');

*//@ts-ignore*

                    let colorsAttributeEnd = modelData[i].geometry.getAttribute('instanceColorEnd');

                    for (let j = cStart; j < cEend; ++j) {

*//修改前端的颜色值*

                        colorsAttribute.data.array[3\*j] = 0;

                        colorsAttribute.data.array[3\*j + 1] = .87;

                        colorsAttribute.data.array[3\*j + 2] = 1;

*//修改末端的颜色值*

                        colorsAttributeEnd.data.array[3\*j] = 0;

                        colorsAttributeEnd.data.array[3\*j + 1] = .87;

                        colorsAttributeEnd.data.array[3\*j + 2] = 1;

                    }

*//@ts-ignore*

                    colorsAttribute.needsUpdate = true;

*//@ts-ignore*

                    colorsAttributeEnd.needsUpdate = true;

                }

            }

        }

    }

#### 3.4.3.2 设置模型颜色值的函数：setStlModelColor

public setStlModelColor(colorData: any){

        let modelData = this.\_scene3D.getStlDataModel();

        if (modelData != undefined) {

            for (let i=0; i<modelData.length; ++i) {

                let type = modelData[i].userData.type;

                if (type === 'stl') {

                    if (colorData == undefined) return;

                    let r = parseFloat((colorData[0]/255).toFixed(1));

                    let g = parseFloat((colorData[1]/255).toFixed(1));

                    let b = parseFloat((colorData[2]/255).toFixed(1));

*//@ts-ignore*

                    modelData[i].material.color = new THREE.Color(r, g, b);

                }

            }

        }

    }

#### 3.4.3.3 设置模型透明值的函数：setModelAlph

此函数代码过长在这里就不展示了，可以自行进入代码文件查看。

### 3.4.4 heatMapManager.ts：代码文件

**此文件主要管理模型二三维等高线热力图的展示功能。**

#### 3.4.4.1 创建等高线热力图模型函数：createHeatMap

 private createHeatMap(dataObj: any) {

        if (dataObj == undefined) return;

        this.\_left = dataObj.elementLeft;

        this.\_top = dataObj.elementTop;

        this.\_type = dataObj.type;

        let data = dataObj.threeData;

        if (data.length == 0) return;

*//判断采点数据*

        let len = Math.sqrt(data.length);

        this.\_tempLen = len;

        this.\_mapData = new Array(data.length)

        let newData = [];

        let index = 0;

        switch(len) {

            case 3:

            for(var i=0;i<data.length;i+=3){

                let tData = [data[i], data[i+1], data[i+2]];

*// if (index == 1) tData.reverse();*

                for (let n=0; n<tData.length; ++n) {

                    newData.push(tData[n]);

                }

                index++;

            }

            break;

            case 4:

            for(var i=0;i<data.length;i+=4){

                let tData = [data[i], data[i+1], data[i+2],data[i+3]];

*// if (index == 0 || index == 2) tData.reverse();*

                for (let n=0; n<tData.length; ++n) {

                    newData.push(tData[n]);

                }

                index++;

            }

            break;

            case 5:

                for(var i=0;i<data.length;i+=5){

                    let tData = [data[i], data[i+1], data[i+2], data[i+3], data[i+4]];

*// if (index == 1) tData.reverse();*

                    for (let n=0; n<tData.length; ++n) {

                        newData.push(tData[n]);

                    }

                    index++;

                }

            break;

            case 6:

                for(var i=0;i<data.length;i+=6){

                    let tData = [data[i], data[i+1], data[i+2], data[i+3], data[i+4], data[i+5]];

*// if (index == 1) tData.reverse();*

                    for (let n=0; n<tData.length; ++n) {

                        newData.push(tData[n]);

                    }

                    index++;

                }

                break;

            default:

            break;

        }

        this.\_tnewData = newData;

*//等高线面板*

        this.\_pgeometry = new THREE.PlaneBufferGeometry(len, len, len-1, len-1);

        let posotionAttribute = this.\_pgeometry.getAttribute('position');

*//修改模型顶点坐标*

        this.\_distance = parseFloat(((len/2)/5).toFixed(2));

        for (let j = 0; j < posotionAttribute.count; ++j) {

            posotionAttribute.array[j\*3+2] = -newData[j] \* this.\_distance;

        }

        posotionAttribute.needsUpdate = true;

*//原点网格*

        this.createGridHelper(len);

        this.\_heatMap = undefined;

*//创建热力图贴图*

        this.\_heatMap = document.createElement('canvas');

        this.\_heatMap.width = 60;

        this.\_heatMap.height = this.\_type == 'portrait'? 440: 376;

        this.\_heatMap.id = 'heatMap';

        this.\_heatMap.style.right = '5px';

        this.\_heatMap.style.position = 'absolute';

        this.\_heatMap.style.top = '0px';

        this.\_heatMap.style.zIndex = '1';

*// this.\_heatMap.borderRadius = '10px';*

        this.\_heatMap.style.border = '#1px solid yellow';

        let main = document.getElementById('3DCanvas');

        if (main == null) return;

        main.appendChild(this.\_heatMap);

        let c = this.\_heatMap;

        this.\_tempCtx = c.getContext('2d');

        if (this.\_tempCtx == null) return;

        let gradient = this.\_tempCtx.createLinearGradient(0, c.height, 0, 0);

        gradient.addColorStop(0, '#6dbde1');

        gradient.addColorStop(.25, '#07ffe8');

        gradient.addColorStop(.5, '#66ff07');

        gradient.addColorStop(.75, '#ffc107');

        gradient.addColorStop(1, '#e78d71');

        this.\_tempCtx.fillStyle = gradient;

        this.\_tempCtx.rect(0, 0, 30, c.height);

        this.\_tempCtx.fill();

        this.\_tempCtx.closePath();

*//设置模型的热力图*

        let heatMap = document.createElement('canvas');

        heatMap.width = 60;

        heatMap.height = this.\_type == 'portrait'? 440: 376;

        let b = heatMap;

        let tempCtx: any;

        tempCtx = b.getContext('2d');

        let gradientCavas = tempCtx.createLinearGradient(0, b.height, 0, 0);

        let tData: any =[];

        for (let k=0; k<newData.length; ++k) {

            let ttData = newData[k].toFixed(1);

            if (ttData == 0 || ttData == -0) ttData = '0.0';

            tData.push(ttData);

        }

        let colorData = [];

        for (let t=0; t<tData.length; ++t) {

            let index = tData[t];

            if (parseFloat(index)  > 2.0) {

                index = '2.0';

            } else if (parseFloat(index) < -2.0){

                index = '-2.0';

            }

            let color = this.\_colorData[index]

            colorData.push(color.r/255, color.g/255, color.b/255);

        }

        this.\_pgeometry.addAttribute('color', new THREE.Float32BufferAttribute(colorData, 3));

        var materialc = new THREE.MeshPhongMaterial({

            side: THREE.DoubleSide,

*//@ts-ignore*

            vertexColors: THREE.VertexColors,

*// wireframe: true*

        });

*// let mesh = new THREE.Mesh(this.\_pgeometry, this.\_shader);*

        let mesh = new THREE.Mesh(this.\_pgeometry, materialc);

        mesh.rotateX(Math.PI \* 0.5);

        mesh.name = "heatMesh";

        if (this.\_temGroup == undefined) return;

        this.\_temGroup.add(mesh);

*//对数据进行排序*

        data.sort().reverse();

        let t = 1;

        for (let n = 0; n<data.length; ++n) {

            let len = data.length;

            let tem = data[n];

            if (tem >= 0) {

               this.\_mapData[n] = tem;

            } else {

                let tem1 = data[len-t];

                if (tem1 > tem) {

                    this.\_mapData[n] = tem1;

                    this.\_mapData[len-t] = tem;

                }

                ++t;

            }

        }

*//设置热力图视角*

        this.\_scene3D.setHeatMapFov();

*//绘制颜色值刻度*

        this.drawText(this.\_distance, len/2);

*//绘制坐标轴*

        let geometry = new THREE.CylinderGeometry( .03, .03, 1, 8 );

        let cylinder = new THREE.Mesh( geometry, new THREE.MeshBasicMaterial( {color: 0x0000ff} ) );

        let box1 = new THREE.Box3().setFromObject(cylinder);

        let center1 = new THREE.Vector3().subVectors(box1.max, box1.min).multiplyScalar(0.5);

        cylinder.name = 'Helper';

        cylinder.position.set(-len/2, -len/2 + center1.y, len/2);

        this.\_temGroup.add( cylinder );

        let cylinder2 = new THREE.Mesh( geometry.clone(), new THREE.MeshBasicMaterial( {color: 0x00ff00} ) );

        let box2 = new THREE.Box3().setFromObject(cylinder2);

        let center2 = new THREE.Vector3().subVectors(box2.max, box2.min).multiplyScalar(0.5);

        cylinder2.name = 'Helper';

        cylinder2.position.set(-len/2, -len/2, len/2 - center2.y);

        cylinder2.rotateX(-Math.PI \* 0.5);

        this.\_temGroup.add( cylinder2 );

        let cylinder3 = new THREE.Mesh( geometry.clone(), new THREE.MeshBasicMaterial( {color: 0xff0000} ) );

        let box3 = new THREE.Box3().setFromObject(cylinder2);

        let center3 = new THREE.Vector3().subVectors(box3.max, box3.min).multiplyScalar(0.5);

        cylinder3.name = 'Helper';

        cylinder3.position.set(-len/2 + center3.z, -len/2, len/2);

        cylinder3.rotateZ(-Math.PI \* 0.5);

        this.\_temGroup.add( cylinder3 );

*//注销事件*

        EventControl.removeEvent('MSG\_CREATE\_HEATMAP\_MODEL', (data: any)=> {this.createHeatMap(data)}, '3D');

    }

#### 3.4.4.2 绘制热力图网格间距函数：createGridHelper

private createGridHelper(len: any) {

        let gC = 0xffffff;

        let gcp = new THREE.GridHelper(len, len, gC, gC);

        gcp.name = 'Helper';

*//@ts-ignore*

        let gcpAttribute = gcp.geometry.attributes.position;

        let gcount = gcpAttribute.count;

        for (let n=0; n<gcount; ++n) {

*//@ts-ignore*

            gcpAttribute.array[n\*3+1] = -len/2

        }

        gcpAttribute.needsUpdate = true;

        if (this.\_temGroup == undefined) return;

        this.\_temGroup.add(gcp);

        let gcp1 = gcp.clone();

        gcp1.name = 'Helper';

        gcp1.rotateX(Math.PI \* 0.5);

        this.\_temGroup.add(gcp1);

        let gcp2 = gcp.clone();

        gcp2.name = 'Helper';

        gcp2.rotateZ(-Math.PI \* 0.5);

        this.\_temGroup.add(gcp2);

    }

#### 3.2.6.3 绘制颜色值刻度尺函数: drawText

#### 3.4.4.4 二三维热力图显示或隐藏函数：onHideAnyModel

### 3.4.5 DataManager.ts：代码文件

**此文件主要管理不同的数据类型的集合。**

#### 3.4.5.1 添加数据集函数：addMaxDataCollection

public addMaxDataCollection(dataCollection: StlDataCollection) {

        let findIndex = -1;

        this.getStlDataCollection(dataCollection.name, findIndex);

*// 没有重复名称的对象，直接添加*

        if (findIndex == -1) {

            this.\_stlDataColArray.push(dataCollection);

        }

*// 出现相同名称的数据集时是直接将原来的数据替换呢？*

*// 还是说出现了同名的数据集时直接抛一个异常，用户必须自己先删除同名数据集，再添加新的数据集？*

        else {

            let removeArray = this.\_stlDataColArray.splice(findIndex, 1);

            let removeItem = removeArray[0];

            this.\_stlDataColArray.push(dataCollection);

        }

    }

#### 3.4.5.2 获取数据集函数：getStlDataCollection

 public getStlDataCollection(name: string, findIndex?: number): StlDataCollection | undefined {

*//清空stl数据集*

        this.clearStl();

        findIndex = -1;

        for (let i = 0; i < this.\_stlDataColArray.length; ++i) {

            let item = this.\_stlDataColArray[i];

            if (item.name == name) {

                findIndex = i;

                break;

            }

        }

        if (findIndex == -1) {

            return undefined;

        }

        let item = this.\_stlDataColArray[findIndex];

        return item;

    }

### 3.4.6 Scene3DManager.ts：代码文件

**此文件主要管理不同的3D场景，通过场景名称来区分场景的类型，但是目前的项目中只用到一个场景，所以不必纠结有几个场景。**

#### 3.4.6.1 获取当前3D场景的函数：getScene3D

public getScene3D(name: string, findIndex?: number): Scene3D | undefined {

        if (findIndex == undefined) {

            findIndex = -1;

        }

        for (let i = 0; i < this.\_scene3DArray.length; ++i) {

            if (this.\_scene3DArray[i].name == name) {

                findIndex = i;

                break;

            }

        }

        if (findIndex == -1) {

            return undefined;

        }

        return this.\_scene3DArray[findIndex];

    }

#### 3.4.6.2 删除当前的3D场景函数：removeScene3D

public removeScene3D(name: string): Scene3D | undefined {

        let findIndex = -1;

        this.getScene3D(name, findIndex);

        if (findIndex == -1) {

            return undefined;

        }

        let removeItemArray = this.\_scene3DArray.splice(1, findIndex);

        let removeItem = removeItemArray[0];

*// 释放事件*

*// EventControl.dispatchEvent('MSG\_SCENE3D\_REMOVE', removeItem, name);*

        return removeItem;

    }

## 3.5 模型对象代码

### 3.5.1 gcodeDatas.ts 文件代码：

**此文件主要管理gcode模型数据的加载功能。**

**3.5.1.1 加载gcode模型文件函数：onLoaderGcode**

public onLoaderGcode(modelUrl: string, objClass: any) {

        let scope = this;

        scope.\_objClass = objClass;

*//@ts-ignore*

        scope.\_loader = new THREE.GCodeLoader();

        if (this.\_config.chunkSize != undefined) this.\_chunkSize = this.\_config.chunkSize;

        if (this.\_tConfig.particlesLoaded) {

*//分片加载*

            fileModule.fileFragmentDownload(modelUrl, this.\_chunkSize, this.\_config.dataType, () => {} ).then((fileConten: any) =>{

                if (fileConten == "") return;

                let modelSize = fileConten.totalSize/1024/1024;

                this.\_temScene3D.\_worker = new Worker('./worker.js');

                this.\_temScene3D.\_worker.onmessage = function(e: any){

                    const message = e.data;

                    let obj = message.dataObj;

                    let data = message.data;

                    scope.\_objClass.\_objectCollection.\_indexGcodeData = data;

*//根据数据绘制模型*

                    scope.onRenderGcode(obj);

                    message.dataObj = null;

                    message.data = null;

                };

                let downloadText = fileConten.downloadText;

                let type = 'particlesLoaded';

                this.\_temScene3D.\_worker.postMessage({downloadText, modelSize, type});

            }).catch((err: any) => {

                console.log("分片加载异常");

*//终止请求*

*// this.onLoadStop();*

                EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', 'gcode', '3D');

            });

        } else {

*//非分片加载源码*

            let onProgress = function (xhr: any) {

                scope.\_ajax = xhr;

            };

            let onError = function () {

*//触发事件*

                EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', 'gcode', '3D');

*//删除事件*

                EventControl.removeEvent('MSG\_MODEL\_URL\_ERROR', () => { }, '3D');

            };

            if (this.\_config.portType == 'qt') {

                if (this.\_config.data != "" || this.\_config.data != undefined) {

                    scope.onRenderGcode(this.\_config.data);

                    scope.\_objClass.\_objectCollection.\_indexGcodeData = this.\_config.data.indexGcodeData;

                    this.\_config.data = null;

                } else {

                    EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', 'gcode', '3D');

*//删除事件*

                    EventControl.removeEvent('MSG\_MODEL\_URL\_ERROR', () => { }, '3D');

                }

            } else {

                let modelSize = this.\_config.modelSize;

                this.\_temScene3D.\_worker = new Worker('./worker.js');

                this.\_temScene3D.\_worker.onmessage = function(e: any){

                    const message = e.data;

                    let obj = message.dataObj;

                    let data = message.data;

                    scope.\_objClass.\_objectCollection.\_indexGcodeData = data;

*//根据数据绘制模型*

                    scope.onRenderGcode(obj);

                    message.dataObj = null;

                    message.data = null;

                };

                let type = 'noParticlesLoaded';

                this.\_temScene3D.\_worker.postMessage({modelUrl, modelSize, type});

            }

        }

    }

### 3.5.2 STLData.ts代码文件：

**此文件主要管理stl模型的加载功能。**

**3.5.2.1 加载stl模型文件的函数：onLoaderStl**

 public onLoaderStl(modelUrl: string, objClass: any) {

        let scope = this;

        if (this.\_tConfig.particlesLoaded) {

*//分片加载*

*//@ts-ignore*

            let loader = new THREE.STLLoader();

            let shaderMaterial = new THREE.MeshPhongMaterial( {

                color: 0x444242,

                shininess: 15,

                emissive: 0x000000,

                specular: 0xb4b6bc,

            });

            if (this.\_config.chunkSize != undefined) this.\_chunkSize = this.\_config.chunkSize;

            fileModule.fileFragmentDownload(modelUrl, this.\_chunkSize, this.\_config.dataType, () => {} ).then((fileConten: any) =>{

                if (fileConten == "" ) return;

                let modelSize = this.\_config.modelSize;

                this.\_temScene3D.\_worker = new Worker('./stlWorker.js');

                this.\_temScene3D.\_worker.onmessage = function(e: any){

                   scope.onCreateMesh(e.data, objClass)

                };

                let type = 'particlesLoaded';

                let downloadText = fileConten.downloadText;

                this.\_temScene3D.\_worker.postMessage({downloadText, modelSize, type});

            }).catch((err: any) => {

                console.log("分片加载异常");

*//终止请求*

*// this.onLoadStop();*

*//触发事件*

                EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', "stl", '3D');

            });

        } else {

*//非分片加载源码*

            let onProgress = function(xhr:any ) {

                if (scope.\_ajax == undefined) {

                    scope.\_ajax = xhr;

                }

            };

            let onError = function(e: Error) {

                console.log(e);

*//触发事件*

                EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', "stl", '3D');

*// 删除事件*

                EventControl.removeEvent('MSG\_MODEL\_URL\_ERROR', ()=>{}, 'UI');

            };

*//进入qt端*

            if (this.\_config.portType == 'qt') {

                if (this.\_config.data != "" || this.\_config.data != undefined) {

*// 创建模型解析对象*

                    let dataparsing = new Dataparsing(objClass);

*//传路径加载模型数据*

                    dataparsing.onParseStl(scope.\_config.data, scope.\_tConfig);

                }

            } else {

                let modelSize = this.\_config.modelSize;

                this.\_temScene3D.\_worker = new Worker('./worker.js');

                this.\_temScene3D.\_worker.onmessage = function(e: any){

                   console.log(e.data);

                   scope.onCreateMesh(e.data, objClass)

                };

                let type = 'noParticlesLoaded';

                this.\_temScene3D.\_worker.postMessage({modelUrl, modelSize, type});

            }

        }

    }

### 3.5.3 MeshPlyData.ts 文件代码：

**此文件主要管理ply模型的加载功能。**

**3.5.3.1 加载ply模型文件的函数：onLoaderPly**

public onLoaderPly(modelUrl: string, objClass: any) {

        let scope = this;

        if (this.\_config.chunkSize != undefined) this.\_chunkSize = this.\_config.chunkSize;

*//@ts-ignore*

        let loader = new THREE.PLYLoader();

        let shaderMaterial = new THREE.MeshPhongMaterial( {

            color: 0x444242,

            shininess: 15,

            emissive: 0x000000,

            specular: 0xb4b6bc,

        });

        let onProgress = function(xhr:any ) {

            if (scope.\_ajax == undefined) {

                scope.\_ajax = xhr;

            }

        };

        let onError = function(e: Error) {

            console.log(e);

*//触发事件*

            EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', "ply", '3D');

*// 删除事件*

            EventControl.removeEvent('MSG\_MODEL\_URL\_ERROR', ()=>{}, 'UI');

        };

        loader.load(modelUrl, (geometry: any) => {

            geometry.computeVertexNormals();

            let mesh = new THREE.Mesh( geometry, shaderMaterial );

            let box1 = new THREE.Box3();

            box1.expandByObject(mesh);

            let mmaxLen = box1.max.distanceTo(box1.min);

            if (mmaxLen <=1) {

                mesh.scale.set(1000,1000,1000);

            }

            let radius = geometry.boundingSphere.radius;

            if (isNaN(radius)) {

*//触发事件*

                EventControl.dispatchEvent('MSG\_MODEL\_URL\_ERROR', "ply", '3D');

*//删除事件*

                EventControl.removeEvent('MSG\_MODEL\_URL\_ERROR', ()=>{}, 'UI');

                return;

            }

*//注意模型反转后y轴与z轴要注意*

            mesh.rotation.set( -Math.PI / 2, 0, 0 );

            mesh.name = "modelStl";

            mesh.userData = {type: 'ply', id: '001'};

            mesh.renderOrder = 6;

            mesh.receiveShadow = true;

            mesh.castShadow = true;

            let box = new THREE.Box3();

            box.expandByObject(mesh);

            let center = box.getCenter(new THREE.Vector3());

            mesh.position.set(0-center.x, 0-center.y, 0-center.z);

            if (scope.\_tConfig.pro3GridHelper) {

                mesh.translateY(-2900);

            }

*//添加模型进入场景*

            objClass.\_objectCollection.addStlData(mesh);

*//触发模型加载完成事件*

            objClass.\_objectCollection.finishLoadTask();

        }, onProgress, onError );

    }