Wish3D Earth

二次开发手册

V1.0

苏州中科图新网络科技有限公司

2017 年 10 月 11 日

1.概述

Wish3D Earth 是基于 WebGL 技术实现的浏览器端高效三维渲染 SDK。实现了使用纯 javascript 在浏览器端对三维模型进行渲染显示，并具备良好的用户交互、添加标注、量算分析、场景漫游等功能。其支持的模型格式目前包括 osgb、geojson 和 lrp 等。其进行了完善的接口封装，支持二次开发。通过将高门槛的三维引擎功能打包成 API 接口，大幅降低开发者对三维地图技术的使用门槛。

1.1 硬件环境

各种操作系统下支持 HTML5 的浏览器：Chrome、FireFox，Safari、IE11、最新腾讯浏览器等；

显卡：NVIDIA 8 以上

1.2 功能特性

1. 高效

本三维引擎使用多线程、多并发完成在线数据的浏览器端三维渲染。数据加载速度快、渲染效率高。渲染帧率能否达到 45 帧每秒以上。

2. 海量数据支持

本三维引擎在倾斜摄影数据的支持度上达到了海量数据的渲染支持。对数据组织结构进行优化，实现海量数据的请求和渲染。

3. 多格式模型支持

本三维引擎可以支持影像、地形、倾斜数据和传统三维数据的叠加展示。

4. 鼠标操作

包括场景的放大、缩小、旋转、沿线飞行等。

5. 分析功能

本三维引擎目前实现了在浏览器端对模型的分析功能，包括测量距离、测量面积、可视域分析、方量分析等。

6. 标记功能

本三维引擎目前支持在浏览器端实现对模型的标注，包括添加点、线、面以及属性编辑等功能。对模型进行标示，分享、展示的时候有据可依。

7. 完善的 API 接口

本三维引擎进行了接口封装，形成了完善的 SDK 产品，包含二次开发 api，接口介绍，功能介绍，示例代码等。

2 接口介绍

2.1 引入 JS 库

引用的资源均来自 SDK 文件夹下 Wish3D Earth 文件夹下，在<head>标签中添加引用 js 文件的代码，详细代码如下：

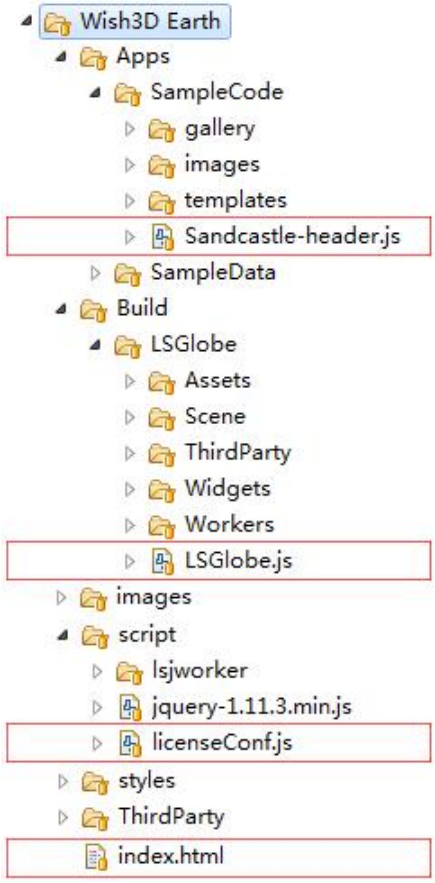
```
<script type="text/javascript" src="Apps/SampleCode/Sandcastle-header.js"></script>

<script src="Build/LSGlobe/LSGlobe.js"></script>

<script src="script/licenseConf.js"></script>
```

2.2 初始化场景

在 script/licenseConf.js 中修改 licenseCode 和 licenseUrl，licenseCode 为 Wish3DEarth 所在服务器的许可码，licenseUrl 为 Wish3DEarth 服务访问地址，如下：

```
/**

 * 许可码和许可访问地址

 */

var licenseCode = "-194002815";

var licenseUrl = "http://localhost:8080/Wish3DEarth";
```

在 Wish3D Eart 根目录下新建一个 html 页面（可命名 index.html），在页面的加载完成事件或者<body>标签中添加初始化场景的函数，文件路径结构如下图：



详细代码如下：

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>初始化-Wish3D Earth</title>
</head>
<script type="text/javascript" src="Apps/SampleCode/Sandcastle-header.js"></script>
<script src="Build/LSGlobe/LSGlobe.js"></script>
<script src="script/licenseConf.js"></script>
<body>

<div id="Wish3DEarth" class="fullSize"></div>
<div id="loadingOverlay"><h1>Loading...</h1></div>
```

```
<script type="text/javascript" >

//定义一个全局变量（暴露给其他方法块使用）
var viewer;

function startup(LSGlobe) {
    'use strict';
    viewer = new LSGlobe.Viewer('Wish3DEarth', {
        baseLayerPicker: false,
        sceneModePicker: false,
        fullscreenButton:false,
        guid:licenseCode,//许可码

       licenseUrl :licenseUrl//许可访问路径（Wish3D Earth 服务地址）
    });
    Sandcastle.finishedLoading();
}
if (typeof LSGlobe !== "undefined") {
    startup(LSGlobe);
} else if (typeof require === "function") {
    require(["LSGlobe"], startup);
}
</script>
</body>
</html>
```

然后把 Wish3D Earth 文件夹用 web 服务发布出来，本处以 tomcat 为例，放在 tomcat 安装路径下 webapps 文件夹下，然后启动 tomcat 服务，访问该页面，访问路径为 http://localhost:8080/Wish3D Earth/index.html ;（注：确保 8080 端口号没有被占用），即可初始化成功。

效果图

2.3 加载数据

2.3.1 加载倾斜摄影数据

加载倾斜数据，可以在页面加载尾部加入以下代码，或者写入方法块，页面加载完成后调用。

```
var    tileset;

tileset = new LSGlobe.PageLOD({

        url:    licenseUrl + "/data/" + dataGuid + "/data/model.json",//dataGuid 为 server 中发布的倾斜数据服务的 GUID

          shadows:LSGlobe.ShadowMode.ENABLED
```

```
});

viewer.scene.primitives.add(tileset);

tileset.readyPromise.then(function(pagelod) {

        viewer.camera.flyTo({

              destination:
LSGlobe.Cartesian3.fromDegrees(pagelod.origin.x,pagelod.origin.y,pagelod.origin.z)//括号内是返回的数据的经度纬度高度

           });

}).otherwise(function(error){

             });
```

2.3.2 加载 GEOJSON 数据

加载 geojson 数据，可以在页面加载尾部加入以下代码，或者写入方法块，页面加载完成后调用。

```
var attachPolygonDataSource = undefined;

  var promise = LSGlobe.GeoJsonDataSource.load(licenseUrl+"/data/"+dataGuid+"/data.geojson",{clampToGround:true,attachPolygon:true});

//dataGuid 为 server 中发布的倾斜数据服务的 GUID

            promise.then(function(dataSource) {

                  attachPolygonDataSource = dataSource;

                  viewer.dataSources.add(dataSource);

                  var entities = dataSource.entities.values;

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

                        var entity = entities[i];
```

```
                entity.polygon.material = LSGlobe.Color.fromRandom({ alpha : 0.01 });

            }

            viewer.zoomTo(dataSource);

        }).otherwise(function(error){

        });
```

2.3.3 加载 LMZ 模型

加载 lmz 数据，可以在页面加载尾部加入以下代码，或者写入方法块，页面加载完成后调用。

```
var model = new LSGlobe.LSModelLOD({

            Id:”可以自定义一个 id”,

            url:    licenseUrl+"/data/"+dataGuid+"/slf.lmz",//dataGuid 为 server 中发布的倾斜数据服务的 GUID

shadows:LSGlobe.ShadowMode.ENABLED,position:new LSGlobe.Cartesian3(-122.22, 46.12,0)});

        model.setRotate(0,0,30);//沿 x、y、z 轴旋转

        model.setScale(10,10,10);//沿 x、y、z 轴缩放

        viewer.scene.primitives.add(model);

        viewer.camera.flyTo({

            destination：LSGlobe.Cartesian3.fromDegrees(-122.22, 46.12, 10.0),//括号内填经度纬度经度

            orientation : {

                heading : LSGlobe.Math.toRadians(20.0),

                pitch : LSGlobe.Math.toRadians(-35.0),

                roll : 0.0
```

```
            },
            duration: 2//飞行时长
        });
```

2.3.4 加载 LRP 数据（需要配合服务端）

```
var imageryLayers = viewer.imageryLayers;
var imageLayer = new LSGlobe.LRPImageryProvider({
            "url" : licenseUrl,
            "layer" : "lrpdata/"+dataGuid+"/data.lrp"
})
//dataGuid 为 server 中发布的倾斜数据服务的 GUID
var alreadyLayer=imageryLayers.addImageryProvider(imageLayer);
```

2.4 飞行

2.4.1 飞到目标点

```
viewer.camera.flyTo({
        destination : LSGlobe.Cartesian3.fromDegrees(120.111,39.555,2000),//括号内填经度纬度高度
        duration: 1.5//飞行时长
    });
```

2.4.2 沿线飞行

```
//构建相机飞行对象
    var cameraTrackControls = new LSGlobe.LSJCameraTrackControls(viewer.scene);
    var timeDelta = 0;
```

```
    //增加关键帧
    function AddKeys () {
        cameraTrackControls.getAllKeys().push({
            destination : viewer.scene.camera.position.clone(),
            orientation : {
                direction: viewer.scene.camera.direction.clone(),
                up:viewer.scene.camera.up.clone()
            },
            time: timeDelta
        });
        timeDelta += 3000;
    };
 //飞行'
function Fly() {
        cameraTrackControls.play();
    };
//暂停',
function pause() {
        cameraTrackControls.pause();
        timeDelta = 0;
    };
//停止',
function Stop() {
        cameraTrackControls.stop();
```

```
        timeDelta = 0;

    };
```

2.5 量算

2.5.1 距离量算

可以把一下代码放入开始量测事件内

```
//距离

var distance = new LSGlobe.LSDistanceMeasure(viewer);

distance.enable = true;
```

2.5.2 高度量算

可以把一下代码放入开始量测事件内

```
//高度

var heightMeasure = new LSGlobe.LSHeightMeasure(viewer);

heightMeasure .enable = true;
```

2.5.3 面积量算

可以把一下代码放入开始量测事件内

```
//面积

var areaMeasure = new LSGlobe.LSAreaMeasure(viewer);

areaMeasure .enable = true;
```

2.5.4 清除量算

可以把一下代码放入开始量测事件内

```
//清除测量

heightMeasure.clear();//高度
```

```
distance.clear();//距离

areaMeasure.clear();//面积
```

2.6 分析

2.6.1 可视域分析

```
//可视域分析'

 function ViewAnalysis() {

        var viewshed3d = new LSGlobe.LSViewshed3D(viewer);

        //点击左键事件下，设置视点,同时 apply 开启分析

        viewshed3d.viewerPositon = new LSGlobe.Cartesian3(0,0,0);//括号内传入点的 xyz 坐标

        viewshed3d.apply();

        //鼠标移动时设置，目标点

        viewshed3d.targetPoint = new LSGlobe.Cartesian3(0,0,0);;//括号内传入点的 xyz 坐标

        //再次点击左键，结束分析

        viewshed3d.finish();

    };
```

2.6.2 方量分析

```
//方量计算

function volume() {

        //接口传入一个面对象

        var cutfill = new LSGlobe.LSCutFill(viewer);

        cutfill.vertice = tileset.NodeMesh;//tileset 是倾斜摄影数据返回对象

        cutfill.matLocal = tileset.m_matLocal;

LSGlobe.Matrix4.inverseTransformation(tileset.m_matLocal,cutfill.matLoc
```

```
alInvert);

        //分析的面高度

        cutfill.zFactor = 100;

        //分析的面对象

        cutfill.polygon = MyPolygon;//MyPolygon 是方量分析的面对象

        //采样间隔

        cutfill.sample = 1;

        //执行分析

        cutfill.apply();

        //分析结果

        //填方体积

        var fillvolume = cutfill.fillVolume;

        //挖方体积

        var cutvolume = cutfill.cutVolume;

        cutfill.clear();

    };
```

2.7 事件

2.7.1 鼠标左击事件

```
var handler = new LSGlobe.ScreenSpaceEventHandler(viewer.scene.canvas);

handler.setInputAction(function (movement) {

    alert("鼠标左击");

     }, LSGlobe.ScreenSpaceEventType.LEFT_CLICK);
```

2.7.2 鼠标右击事件

```
var handler = new LSGlobe.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    alert(“鼠标右击”);
     }, LSGlobe.ScreenSpaceEventType.RIGHT_CLICK);
```

2.7.3 鼠标移动事件

```
var handler = new LSGlobe.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    alert(“鼠标移动”);
     }, LSGlobe.ScreenSpaceEventType.MOUSE_MOVE);
```

2.7.4 鼠标滚轮事件

```
var handler = new LSGlobe.ScreenSpaceEventHandler(viewer.scene.canvas);
handler.setInputAction(function (movement) {
    alert(“鼠标移动”);
     }, LSGlobe.ScreenSpaceEventType.WHEEL
);
```

2.8 标绘

2.8.1 点

```
var promise;
var drawDataSource;
      promise=viewer.dataSources.add(new LSGlobe.GeoJsonDataSource("drawDataSource"));
            promise.then(function(dataSource) {
                  drawDataSource = dataSource;
```

```
        }).otherwise(function(error){});
drawDataSource.entities.add({
        name:"未命名点",
        clampToGround : true,
        position: {
           x:-2760855.7705809874,
           y:4702570.061589066,
           z:3297120.595417835},//x,y,z 为该点的空间坐标
        point: {
              pixelSize: 3,
              color: LSGlobe.Color.YELLOW,
              disableDepthTestDistance : 1000000000
               }
        });
```

2.8.2 线

```
drawDataSource.entities.add({
        name: 'line on the surface',
        clampToGround : true,
        parent: polylines,
        polyline: {
                positions:
LSGlobe.Cartesian3.fromDegreesArrayHeights([120.41361697964531,
31.31779626235843,-0.01213042757899609,
120.41444388457514, 31.322422773003876,-0.009140443145189945]),//数组里面是构成线
的所有点坐标
```

```
                width: 2,

                material: LSGlobe.Color.BLUE

            }

});
```

2.8.3 面

```
var PolygonPointArray_fill;

PolygonPointArray_fill=[120.42411947307119, 31.337953991332057, -0.012817742144694305, 120.40815393585946, 31.338137247773552, -0.007886288461311773, 120.40804984077438, 31.32042451784116, -0.0047008157416940045, 120.42401046530267, 31.319714381781658, -0.006021071626987037, 120.42459792475103, 31.32326582252122, -0.0048488072024749085];//数组里面是构成面边框的所有点集合

drawDataSource.entities.add({

            name: '未命名面',

            id:'可以自定定义 id',

          clampToGround : true,

          polygon: {

                hierarchy: {

                    positions : LSGlobe.Cartesian3.fromDegreesArrayHeights(PolygonPointArray_fill)

                },

              material: LSGlobe.Color.YELLOW,
```

```
            fill: true,    //不显示填充

            outline: false,

            perPositionHeight : false,

            outlineColor: LSGlobe.Color.YELLOW

          }

});
```

2.9 模型压平并添加模型

2.9.1 模型压平

```
//面对象

var myPolygon = new LSGlobe.PolygonGeometry({

polygonHierarchy:new LSGlobe.PolygonHierarchy(LSGlobe.Cartesian3.fromDegreesArrayHeights([120.41615823760809, 31.328724867187994, 9.36784088170369, 120.41706583494121, 31.32878103064065, 9.936111366778974, 120.41710268038933, 31.32803340317643, 16.360085974532677, 120.41633787473079, 31.327985837729308, 9.324475948807756, 120.41619115657267, 31.32809337212252, 9.29595510619817, 120.41616482751077, 31.328740813501224, 9.349549090950706])),////数组里面是构成面边框的所有点集合

perPositionHeight : true

});

tileset._flattenPolygon.push(myPolygon);

tileset._needUpateFlatten = true;
```

2.9.2 添加模型

此处添加模型同 2.3.3 加载在对应压平的面上

2.10 双屏对比

双屏对比原理：通过 iframe 标签引用模型渲染页，实时操作双屏同步，操作一个框架同时把相机参数传到另一个框架中。

```
//视角转换方法
function jumpToViewPoint(position,direction,up){
//position,direction,up 是相机对象属性
    viewer.camera.flyTo({
        destination :new LSGlobe.Cartesian3(position.x,position.y,position.z),
        orientation : {
            direction : new LSGlobe.Cartesian3(direction.x,direction.y,direction.z),
            up : new LSGlobe.Cartesian3(up.x,up.y,up.z)
        },
        duration:0
    });
}
//框架 1，同步框架 2 视角
parent.document.getElementById("proWin2").contentWindow.jumpToViewPoint(camera.position,camera.direction,camera.up);
```

2.11 单体化

```
var attachPolygonDataSource = undefined;
    Sandcastle.addToolbarButton('加载单体化面对象', function() {
        //加载单休化对象
     var promise = LSGlobe.GeoJsonDataSource.load('../../SampleData/Data/test.geojson',{clampToGround:true,attachPolygon:true});//test.geojson 是保存后提交给服务转换的文件
```

```
        promise.then(function(dataSource) {
            attachPolygonDataSource = dataSource;
            viewer.dataSources.add(dataSource);
            var entities = dataSource.entities.values;
            for (var i = 0; i < entities.length; i++) {
                var entity = entities[i];
                entity.polygon.material = LSGlobe.Color.fromRandom({ alpha : 0.01 });
            }
            viewer.zoomTo(dataSource);
        }).otherwise(function(error){
        });
 });


var _actionFlatPolygon = false;
var _actionCutFill = false;


var _selectedObject = undefined;
var _selectedMaterial = undefined;
function selectEntityChanged(value) {
        if(_selectedObject == undefined){
            _selectedObject = value;
        }
        //单体化风格设置
        if(_selectedObject !== value && _selectedObject.attachPolygon) {
```

```
            //不选中时，设为透明
            if (_selectedObject !== undefined && _selectedObject.polygon !== undefined){
                _selectedObject.polygon.material = LSGlobe.Color.fromRandom({
                    alpha : 0.01
                });
            }
            _selectedObject = value;
        }

        //面选中时的风格
        if (value != undefined){
            if (value.polygon != undefined){
                value.polygon.material = new LSGlobe.Color(0.5,1.0,1.0,0.7);
            }
        }
    }
    viewer.selectedEntityChanged.addEventListener(selectEntityChanged);
```

2.12 日照分析

```
function computes(now){
            var month = now.getUTCMonth() + 1;
            var day = now.getUTCDate();
            var year = now.getUTCFullYear()
            var y = eval(year);
            var m = eval(month);
```

```
                var d = eval(day);
                var extra = 100.0*y + m - 190002.5;
                var rjd = 367.0*y;
                rjd -= Math.floor(7.0*(y+Math.floor((m+9.0)/12.0))/4.0);
                rjd += Math.floor(275.0*m/9.0);
                rjd += d;
                rjd += 1721013.5;
                rjd -= 0.5*extra/Math.abs(extra);
                rjd += 0.5;
                return rjd;
            }
 //如果没有选择年月日，就是现在的年月日
var JulianDate =computes(new Date());
//也可以自己设置一个 Date 对象 然后传
var JulianDate =computes(Date 对象);
julianDay = JulianDate;
var hour=当前 Date 对象的小时;
var minute=当前 Date 对象的分钟;
var secound=当前 Date 对象的秒;
julianSecound=hour*60*60+minute*60+secound;
if(julianSecound>=28800){
viewer.clock.currentTime= new LSGlobe.JulianDate(julianDay,julianSecound-28800);
}else{
    viewer.clock.currentTime= new LSGlobe.JulianDate(julianDay-1,julianSecound+57600);
```

```
}
```

2.13 图层管理

2.13.1 添加场景数据

标绘数据和矢量数据加载方式同手册的 2.3.2 条，加载模型数据同手册 2.3.3 条，加载 Lrp 数据同手册 2.3.4 条

2.13.2 场景数据的显示隐藏

1.标绘数据:如 2.3.2 条，加载完成后会返回一个 datasource，然后遍历每个 entity 可以获取每个 entity 的 id,可以通过 dataSource.entities.getById('entity 的 id').show=false;//false：隐藏， true:显示；

2.矢量数据的显示隐藏:如 2.3.2 条，加载完成后会返回一个 datasource，可以通过 dataSource.show=false;//false：隐藏， true:显示；

3.模型数据的显示隐藏：如 2.3.3 条，隐藏可以通过 viewer.scene.primitives.remove(model);，显示可以重新添加该模型；

4.LRP 数据的显示隐藏：如 2.3.4 条，图源添加完成后返回一个 alreadyLayer，可以通过 alreadyLayer.show = false;//false：隐藏， true:显示；