

MicroStation 三维模型导出 ArcGIS SLPK 格式探讨

金波

(遵义水利水电勘测设计研究院, 遵义 563000)

摘要: Bentley 公司基于 MicroStation 平台的系列软件能进行多专业、多行业三维建模, 但其模型不能直接的导出到 ArcGIS 平台中使用。SceneLayer Package (SLPK) 是 ArcGIS 支持的基于 OGC I3S 规范的国际三维标准格式, 通过研究 SLPK 格式和 MicroStation 三维模型, 基于 MicroStation 进行 MDL 二次开发, 实现了在 MicroStation 中三维模型直接导出成 SLPK 格式, 以便在 GIS 平台中更好的应用和展示 BIM 模型。

关键词: MicroStation; SLPK; ArcGIS Pro; MDL 二次开发; BIM+GIS

1 引言

MicroStation 是一款工程领域常用的二维、三维一体化计算机辅助设计软件, Bentley 公司基于 MicroStation 研发了一系列软件用于土木、工厂、建筑和地理信息行业, 可用于创建各个专业和行业的三维模型, 并附加属性信息, 提供了一套广泛应用的 BIM 解决方案。另外 MicroStation 提供了 SDK, 用于用户二次开发以实现更多的功能。

为了更好地将 BIM 三维模型与 GIS 结合起来, ESRI 公司发起了 I3S 规范; 该规范被开放地理空间联盟 (OGC) 采纳为新的国际三维标准。基于 I3S 规范 ESRI 提出了三维数据格式 SLPK (场景图层包)。SLPK 格式能够在 ESRI 公司的 ArcGIS Pro 以及 ArcGIS online 中使用, 发布为服务后可用 ArcGIS API for JavaScript 调用应用于 Web。

MicroStation 三维模型目前不支持直接导出 SLPK 格式供 ArcGIS 使用, 通过研究 SLPK 格式和 MicroStation 模型, 基于 MicroStation MDL 二次开发, 实现了在

MicroStation 中将模型直接导出为 SLPK 格式; 同时为了方便地在 GIS 系统中使用 SLPK, 在导出时添加了坐标系转换的功能, 可以将模型导出为经纬度坐标或者 Web 墨卡托坐标等。其转换流程如图 1 所示。

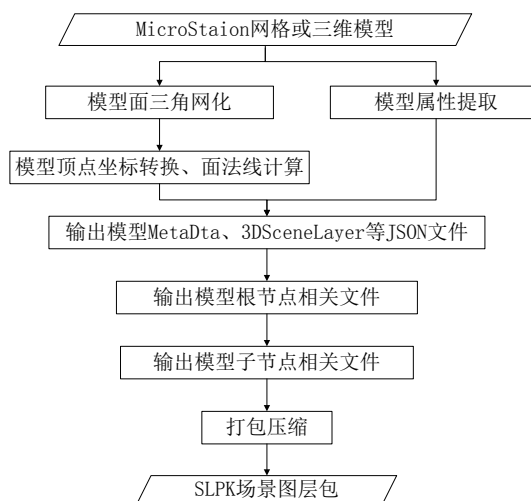


图 1 MicroStation 中模型转换流程

2 MicroStation三维模型简介

在 MicroStation 中, 元素种类很多, 主要分为线、面和体几类。三维模型通常属于体, 极少数属于网格面。三维模型常见的元素种类见表 1。

表 1 三维模型常见的元素种类

元素种类	描述	示例
网格	由多个三角形或多边形组成的一个复杂表面或体, 不包含曲面	道路模型、开挖体、地质模型
实体	单个由基本实体功能如拉伸、扫描、加厚创建的体	立方体构件、圆柱体构件
智能实体、单元	由复杂实体功能创建或简单实体运算之后形成的体, 以及专业功能放置的单元库中的简单构件	板、梁、柱、屋顶和异形构件
复合单元	多个实体或单元复合而成的单元, 由多个部分组成一个整体	门、窗

针对不同类型的三维模型, 需要进行不同类型的处理。在程序中, 首先需要判断三维模型的类型, 主要用到的函数包括 `mdlElement_getType()`、`mdlTFElmdscr_getApplicationType()` 和 `mdlKISolid_isSolidElement()` 等。

`mdlElement_getType()` 可用于判断三维模型是否是网格或实体。

`mdlKISolid_isSolidElement()` 可用于判断三维模型是否是实体或智能实体。

`mdlTFElmdscr_getApplicationType()` 可用于判断三维模型是否属于建筑模型单元或者复合单元。

3 模型面三角形化

三维模型几何信息由若干个面组成, MicroStation 中三维模型可能包含曲面、含圆弧的面或者多边形面, 但是在 SLPK 格式中, 三维模型目前只支持三角形面, 因此需要将 MicroStation 中三维模型进行三角形化, 这样就能满足 SLPK 的要求。针对不同类型的三维模型需要进行不同的处理。

(1) 网格三角形化: 网格可能全部由三角形面组成, 也可能包含四边形面或多边形面, 可以使用 `mdlMesh_copyTriangulated()` 函数将网格复制并三角形化, 得到的新网格中只包含三角形面。接着使用 `mdlMesh_getPolyfaceArrays()` 函数即可获得网格各个三角形面的顶点坐标。图 2 中左边为原始网格, 右边为三角形化的网格。

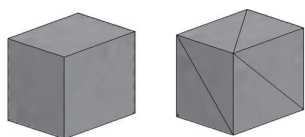


图 2 网格三角形化

(2) 单元、实体和智能实体三角形化: 单元、实体和智能实体都只包含单个的体, 首先使用 `mdlKISolid_elementToBody2()` 函数将元素转换成内部 `KISolid`, 再使用 `mdlKISolid_facetBody()` 函数进行三角形化。将含曲线和圆弧的面转换成三角形面会有精度丢失, 主要依据弦公差和角度公差两个参数来保证精度。图 3 中展示的是转换前后的变化。

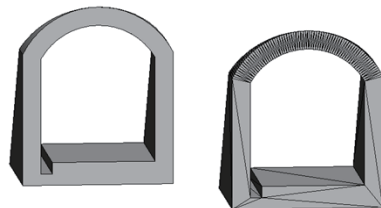


图 3 转换前后的变化

(3) 复合单元: 复合单元由多个单元或智能实体组成, 应使用 `mdlKISolid_elementToBodyList()` 来将元素转换成多个 `KISolid`, 对于每个 `KISolid` 使用单元、实体和智能实体的方法来进行三角形化, 最后把三角形化的面合并起来存储即可。图 4 中左边的窗户包含 6 个部分, 右边是三角形化之后的效果。

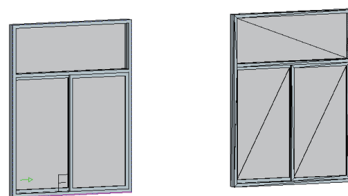


图 4 复合单元

在完成三角形化的同时, 也获得了模型在当前坐标系的顶点坐标。

4 坐标转换、法线计算

SLPK 的主要使用场合是 Web, WebGIS 系统常用的坐标系是网络墨卡托坐标系 (EPSG: 3 857) 和 WGS84 经纬度坐标系 (EPSG: 4 326), 而 MicroStation 建模结合地形模型总装后的坐标系通常为 Beijing54、Xian80 或 CGCS2000 的三度分带投影坐标系, 因此在模型输出成 SLPK 格式时提供坐标系转换功能很有必要。

基于 GDAL 和 Proj.4 开源库, 根据源坐标系和目标坐标系的 EPSG 代码, 实现对坐标进行批量转换。

当不涉及到网络墨卡托坐标系时, 主要代码如下:

```
bool coordTranform(int count,int srcEpsg,int
destEpsg,double* px,double* py)
```

```

{
    OGRSpatialReference oSrcSRS,oDestSRS;
    OGRCoordinateTransformation* pTanfrom;
    OGRErr error1=oSrcSRS.ImportFromEPSG(srcEpsg);
    OGRErr error2=oDestSRS.ImportFromEPSG(destEpsg);
    if(error1==OGRErr::OGRERR_UNSUPPORTED_SRS || error2==OGRErr::OGRERR_UNSUPPORTED_SRS)
    {return false;}
    pTanfrom=OGRCreateCoordinateTransformation(&oSrcSRS,&oDestSRS);
    if (pTanfrom==NULL || !pTanfrom->Transform(count,px,py)){return false;}
    return true;
}

```

当输入或者输出坐标系为网络墨卡托坐标系时,所使用的 Proj.4 版本尚不支持,需要先将坐标转换成 WGS84 经纬度再进行转换。其转换代码如下:

```

#define GC_PI 3.141592653589793
2384626433832795e0//PI
#define GC_R2D 57.295779513082320876798
154814105e0//弧度转度
#define GC_D2R 0.017453292519943295769236
907684886e0//度转弧度
void xy2bl_WebMercator(const double& x,const double&
y,double* lat,double* lon)//web墨卡托转经纬度
{
    double &_lon=*lon;
    double &_lat=*lat;
    _lon=x/20037508.34*180.0;
    _lat=y/20037508.34*180.0;
    _lat=GC_R2D*(2*atan(exp(_lat*GC_D2R))-GC_PI*0.5);
    return;
}
void bl2xy_WebMercator(const double& lat,const double&
lon,double *x,double *y)//经纬度转web墨卡托
{
    double &_x=*x;
    double &_y=*y;
    _x=lon*180.0/20037508.34;
    _y=log(tan((90+lat)*GC_D2R*0.5))*GC_R2D;
    _y*=20037508.34/180.0f;
}

```

当输入坐标系和输出坐标系椭球不一样时,通常坐标转换需要七参数或四参数等才能精确转换,否则转换结果平面误差可达数十米甚至上百米。但在实际试验中发现,proj.4 和 GDAL 中已内置了相关转换参数,不同椭球的坐标转换后的结果精度较高,与网络地图套和较好,故可直接使用。假如出现了偏差较大的情况,只需对模型进行相应的移动即可。

除了顶点坐标外,三维模型的显示和渲染还需要每个面的法线向量,法线的计算方式为向量叉乘。如一个三角形三个顶点依次分别为 P1、P2 和 P3,则这个面的法线为,一般还需要进行归一化使法线向量的模为 1。在计算法线时需要注意,经纬度坐标不能直接拿来对法线的计算,如果输出坐标系是 WGS84 经纬度,则应先计算法线,再进行坐标转换。

5 模型属性提取

BIM 模型除了几何信息之外属性信息也非常重要,基于 MicroStation 的建筑建模相关软件的 SDK 中提供了一些提取属性信息的方法。

首先设计了相应数据结构来存储属性,如下:

```

enum I3SFieldType{
    FieldTypeFloat=0,
    FieldTypeDouble,
    FieldTypeInteger,
    FieldTypeString,
    FieldTypeOID
};
union I3SFieldValue
{
    int nValue;
    char sValue[256];
    float fValue;
    double dValue;
};
struct I3SField{
    string fieldName;
    I3SFieldType fieldType;
    string fieldAlias;
    I3SFieldValue value;
    I3SField()

```



```
{
    fieldType=FieldTypeString;
    strcpy(value.sValue, " ");
}
};
```

下面列出了一些常见的属性。

(1) 面积：模型的表面积，类型为双精度浮点数，直接使用 `mdlMeasure_AreaProperties()` 函数获得。

(2) 体积：模型的体积，类型为双精度浮点数，直接使用 `mdlMeasure_volumeProperties()` 函数获得。

(3) 位置：模型的中心坐标，因为有三个浮点数 (X , Y , Z)，简化使用字符串来存储，从 `MicroStation` 元素的元素头中 `range` 变量中能获取模型坐标值的最大值和最小值，取其平均数即为近似的模型中心坐标。

(4) 图层：模型所在的图层名称，使用 `mdlElement_getProperties()` 函数可获得图层 ID，再使用 `mdlLevel_getName()` 函数即可获得图层名称，类型为字符串。

(5) 颜色：模型顶点颜色，类型为字符串，在 `MicroStation` 中一个元素只能有一种颜色，使用 `mdlElement_getSymbology()` 函数获取颜色索引号，如果获得的索引号是“按层”，则需要根据图层 ID 获得颜色索引号，即使用 `mdlLevel_getElementColor()` 函数。得到颜色索引号后，通过调色板来查询索引号对应的颜色 RGB 值，用到的函数是 `mdlColorPal_getColorDescr()` 和 `mdlColorDescr_getRgb()`，最后将 RGB 值转成字符串。

(6) 类别和样式：类别和样式是 `MicroStation` 建筑模型最重要的属性之一，类型为字符串，涵盖模型材料、图样、工程量、渲染特性、切图和结构分析等一系列属性的定义，知道了样式名称，即可在库中查询到该样式的这些属性，如砖外墙的类别为“`Wall_Ext`”，样式为“`Brick wall`”。使用 `mdlITFPartRef_getPart()` 函数获得样式，再使用 `mdlITFPartRef_getPartName()` 函数和 `mdlITFPartRef_getPartFamilyName()` 函数即可分别获得样式和类别。

下面是系统文件中对砖墙的样式定义：

```
<Part name=" Brick Wall" >
  <Description>砖墙</Description>
  <Model thickness=" 200.0" height=" 3000.0" >
    <Level>A-WALL-EXTL</Level>
    <Symbology color=" BYLEVEL "
style=" BYLEVEL" weight=" BYLEVEL" />
```

```
</Model>
  <CutView activate=" true "
cutInCompoundCells=" false" >
    <Level>A-WALL-BRIC</Level>
    <Symbology color=" BYLEVEL "
style=" BYLEVEL" weight=" BYLEVEL" />
    <Unifier activate=" true" >Unifier Brick</
Unifier>
  </CutView>
  <FrontAndRearView activate=" true" >
    <Level>A-WALL-BRIC</Level>
    <Symbology color=" BYLEVEL" style=" BYLEVEL "
weight=" 0" />
    <Unifier activate=" true" >Unifier Brick</Unifier>
  </FrontAndRearView>
  <CutPatterns activate=" false" >
    <FilledColor activate=" true" color=" 26" />
    <Pattern mode=" PATTERN" >
      <LinearHatching angle1=" 45.000000"
spacing1=" 3.0" >
        <Symbology color=" 14" style=" 0"
weight=" 0" />
      </LinearHatching>
      <CrossHatching angle1=" 45.000000"
spacing1=" 3.0" angle2=" -45.000000"
spacing2=" 3.0" >
        <Symbology color=" 14" style=" 0"
weight=" 0" />
      </CrossHatching>
      <Patterning rowSpacing=" 0.0"
columnSpacing=" 2.0" >
        <Cell angle=" 45.000000"
scale=" 1.000000" >ANSI32</Cell></Patterning>
      </Pattern>
    </CutPatterns>
    <Centerline activate=" false" >
      <Level>A-WALL-CNTR</Level>
      <Symbology color=" 0" style=" 0" weight=" 0" />
      <CustomLine scaleToForm=" false "
scale=" 1.000000" />
    </Centerline>
    <Rendering activate=" true" >
      <Palette>$( _DGNLIB )|Dataset_Materials.dgnlib|ANZ_
Blocks&Bricks.pal</Palette>
```

```
<Material>brick Hayden's red</Material>
</Rendering>
```

(7) 材质: 建筑模型的材质定义了模型的纹理和贴图信息, 从样式定义中能获取到材质信息的定义, 如“Concrete_3”表示贴图为混凝土。

除了这些属性, 还能够根据不同专业和行业来获取所需的属性, 但是具体需要配置各专业的工作环境和约定相应的标准, 目前还未深入研究。

6 输出SLPK

完成了上述处理后, 即可开始创建 SLPK 格式的场景图层包了。

(1) 格式简介: SLPK 文件是由一系列 JSON 文件和二进制文件按照严格的文件组织再打包成 ZIP 格式的压缩包, 其主要文件夹结构如图 5 所示。

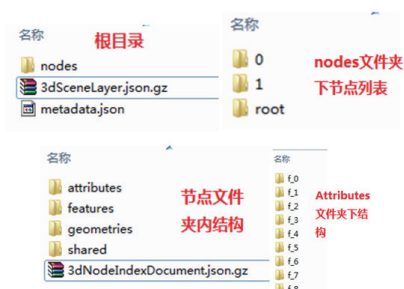


图 5 主要文件夹结构

根目录下有 metadata.json、3DSceneLayer.json 两个文件和 nodes 文件夹; nodes 文件夹中存储有各个节点文件夹; root 文件夹下存储的是根节点信息, 只有一个文件 3dNodeIndexDocument.json; 其他每个节点文件夹下可存储模型的一个部分, 每部分的属性可以不同, 包括 3dNodeIndexDocument.json 文件和若干个子文件夹 (attributes、features、geometries 和 shared 文件夹); attributes 文件夹下存储的是各个属性的值; features 文件夹下有 0.json 文件; geometries 文件夹下存储的是 0.bin 文件, 以二进制格式存储顶点坐标、法线坐标、纹理坐标和顶点颜色值; shared 文件夹中存储的是 sharedResource.json, 主要存储纹理贴图信息; 如果存在纹理贴图, 还会有 textures 文件夹, 里面存储的是 dds 或者 jpeg 格式的纹理图片。每个节点的 3dNodeIndexDocument.json 文件中会存有父节点的信息和子节点的信息, 所有节点之间的关系是一种树状关

系, 这对于大数据量的模型有利于实现 LOD。

(2) 创建 JSON 文件: JSON 文件是 JavaScript Object Notation 的缩写, 是一种用来存储和表示数据的文本格式, 使用 cJSON 库能够很方便地进行 JSON 文件的读写。对于 SLPK 包内的所有 json 格式的文件, 都可以用 cJSON 库进行创建, 各个文件具体内容参见 OGC SLPK 官方文档。

metadata.json 文件的创建代码如下:

```
bool saveMetaData(const string& modelFolder,const int&
nodeCount)
{
    //write metaData.json file
    cJSON* cJSONObject cJSONObj;
    cJSONObj.Add( "folderPattern", "BASIC" );
    cJSONObj.Add( "archiveCompressionType", "STORE" );
    cJSONObj.Add( "resourceCompressionType", "GZIP" );
    cJSONObj.Add( "I3SVersion", "1.6" );
    cJSONObj.Add( "nodeCount", nodeCount);
    ofstream metaDataFile;
    string metaDataFileName=modelFolder+"\\metadata.
json" ;
    metaDataFile.open(metaDataFileName.data());
    metaDataFile<<cJSONObj.ToFormattedString();
    cJSONObj.Clear();
    metaDataFile.close();
    return true;
}
```

创建出来的内容如下:

```
{
    "folderPattern": "BASIC",
    "archiveCompressionType": "STORE",
    "resourceCompressionType": "GZIP",
    "I3SVersion": "1.6",
    "nodeCount": 2
}
```

(3) 创建几何和属性文件: 几何信息 0.bin 详细格式参见官方文档, 其中包含主要信息见表 2。

表 2 几何和属性文件主要信息

数据	描述	每个顶点相应数据所需的存储空间
顶点	Vertex	3*float
法线	Normal	3*float
纹理坐标	UV	2*float
颜色	RGBA	4*byte

在 0.bin 中依次存储的是顶点数组、法线数组、纹理坐标数组和颜色数组，写入文件可使用 ofstream，ofstream::write() 函数能够快速批量将数组写入到文件中。

属性值也可存储在名为 0.bin 的二进制文件中，对于不同属性类型的属性值，写入方法有所差别，其代码如下：

```
I3SField field; string attrFileName;
.....//此处对field和attrFileName赋值
ofstream attrFile;
attrFile.open(attrFileName.data(),ios_base::binary);
attrFile.write((char*)&featureCount,sizeof(UINT32));
switch(field.fieldType)
{
case FieldTypeDouble:
    int nTemp;nTemp=0;
    attrFile.write((char*)&nTemp,4);//偏移必须是double的
    倍数，double长度为字节减去int的字节
    attrFile.write((char*)&field.value.dValue,sizeof(double));
    break;
case FieldTypeInteger:
case FieldTypeOID:
    attrFile.write((char*)&field.value.nValue,sizeof(int));
    break;
case FieldTypeFloat:
    attrFile.write((char*)&field.value.fValue,sizeof(float));
    break;
default://string,字符串末尾要加\0
    string strValue=string_To_UTF8(field.value.sValue);
    UInt32 attributeByteCounts=strValue.size()+1;
    attrFile.write((char*)&attributeByteCounts,sizeof(UI
nt32));
    attrFile.write((char*)&attributeByteCounts,sizeof(UI
nt32));
    attrFile.write(strValue.data(),attributeByteCounts);
}
attrFile.close();
```

要注意将字符串转成 UTF-8 编码，否则汉字显示为乱码。

(4) 文件压缩：完成创建所有文件后，需要将除了 metaData.json 外的所有文件先压缩成 gz 格式，在程序中使用 ShellExecute() 函数调用开源的压缩软件 7z.exe，给定参

数“a < 文件名 >.gz < 文件名 >”即可完成压缩，压缩的同时删除掉原文件；接着使用 ShellExecute() 调用 7z.exe，给定参数“a < 模型文件夹 >< 模型名 >.zip -ir!< 模型文件夹 >*. * -mx0”，即可将所有文件打包成 zip 格式；最后将 zip 文件名称中后缀名“.zip”修改成“.slpk”即完成的场景图层包的创建。

7 运行结果

在 MicroStation 中运行 MDL 程序，界面如图 6 所示。



图 6 MicroStation 中运行 MDL 程序

选中要导出的一个或者多个模型（多个模型时，每个模型会导出成场景图层包的一个节点），设置相应路径、名称和坐标系等参数，在视图中单击左键即开始导出。导出完成后，在 ArcGIS online 中，直接即可将 SLPK 文件上传并发布成 Scene Layer。图 7、图 8 分别是大坝原始三维模型和在网页中展示的效果。图 9、图 10 是房屋原始三维模型和在网页中展示的效果。

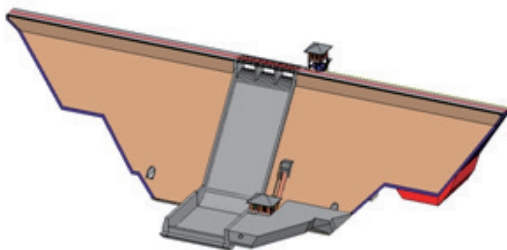


图 7 大坝原始三维模型

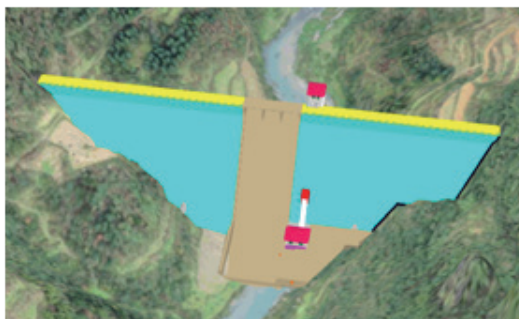


图 8 大坝在网页中展示的效果



图9 房屋原始三维模型



图10 房屋在网页中展示的效果

图11 为在网页中可以查询模型每一个部分的属性。



图11 在网页中查询模型属性

使用 ArcGIS API for JS 调用 SLPK 场景图层的示例代码如下:

```
<script>
    require([ "esri/Map", "esri/views/SceneView"
    , "esri/layers/SceneLayer" , "dojo/domReady!" ],
```

```
function(Map,SceneView, SceneLayer){
    var myMap=new Map();
    var view=new SceneView({container:viewDiv,map:my
Map});
    var sLayerdb=new SceneLayer({url:" https:// xxx.
arcgis.com/server/rest/services/Hosted/%E5%
BE%B7%E9%9A%86%E6%B0%B4%E5%BA%93%E5%A4
%A7%E5%9D%9D/SceneServer" });
    var sLayerglf=new SceneLayer({url:" https://xxx.
arcgis.com/server/rest/services/Hosted/%E5%
BE%B7%E9%9A%86%E6%B0%B4%E5%BA%93%E5%BC
%80%E6%8C%96/SceneServer" });
    .....
    myMap.add(sLayerdb);myMap.add(sLayerkw);
    .....
</script>
```

8 结语

通过研发和实践,解决了 MicroStation 三维建筑模型直接导出成 SLPK 场景图层包的问题,为 BIM 模型和 GIS 系统的应用结合打下了一些基础。

参考文献:

- [1]OGC Indexd 3d Scene Layer(I3S) and Scene Layer Package Format Specification.
- [2] 彭殿军. 从 MicroStation 三维模型到二维图纸的生成流程[J]. 有色冶金设计与研究, 2016 (05) : 106-107, 110.
- [3] 张燕, 唐相桢. MicroStation 图形数据到 ArcGIS 数据转换方法的研究 [J]. 城市勘测, 2015 (1) : 26-29.
- [4] 乔新, 王海银, 胡振彪. MicroStation 调用 WebGIS 地图服务的开发与应用 [J]. 现代测绘, 2011 (2) : 29-31.