**如何将Fbx格式转换成VVO格式**

1. **Fbx文件格式简介**

**1.1 KFbxSdkManage和KFbxScene**

Fbx文件是Autodesk开发的文件格式，其开发目的就是为了实现Autodesk旗下软件之间的数据交换。Fbx文件格式本身是不公开的，而是通过FBX SDK实现对Fbx文件的读取以及写入。

使用FBX SDK时，最先遇到的两个对象就是KFbxSdkManage和KFbxScene。

KFbxSdkManage是sdk中的中心类，负责了整个sdk内部状态的管理，很多其他对象创建也依赖于KFbxSdkManage，程序中只需要有一个KFbxSdkManage类的实例即可。

KFbxScene如其名所示，代表了一个场景，而这里的场景就是fbx文件中包含的所有信息，fbx文件导入以后，在程序中就是一个KFbxScene对象,所以一个fbx文件只需要一个KFbxScene类的实例。

**1.2 Fbx的数据组织方式**

Fbx的数据组织方式是scene tree，即场景树。由KFbxScene所声明的对象可以得到该场景树的根节点，根节点包含了一系列子节点KFbxNode，每个KFbxNode又有其自己的子节点，以此往下类推。这样通过递归循环就可以遍历到每一个节点，然后获取该节点的信息。

RootNode是该Fbx文件所对应的根节点，由以下语句得到：

KFbxNode\* pNode = pScene->GetRootNode();//获得根节点

图1是一个圆柱体的例子。该圆柱体总共有四个节点，Patch、SkeletonRoot、

SkeletonLimbNode1、SkeletonLimbNode2是这四个节点的名称。其中Patch、SkeletonRoot是RootNode的孩子，SkeletonLimbNode1是SkeletonRoot的孩子，SkeletonLimbNode2是SkeletonLimbNode1的孩子。

 图 1



**1.3 Fbx中节点KFbxNode说明**

**1.3.1 节点坐标**

场景树中每个节点都是KFbxNode，KFbxNode类本身包含了坐标变换信息，例如可以用函数EvaluateGlobalTransform(KTime pTime)获得在pTime时刻该节点的全球变换矩阵。获得全球变换矩阵后，我们可以得到节点在该时刻的世界坐标系下的平移、旋转和缩放。同理由EvaluateLocalTransform(KTime pTime)获得在pTime时刻该节点的本地变换矩阵，进而获得该时刻本地坐标系下节点的平移、旋转和缩放。

**1.3.2 节点类型**

一个节点KFbxNode包含其他数据作为KFbxNodeAttribute对象，包含在其内部，这里的其他数据是指mesh，Nurbs，skeleton，camara，light等定义在KFbxNodeAttribute：：EAttributeType中的枚举类型。获得一个节点的类型可用以下函数语句来实现:

KFbxNodeAttribute::EAttributeType lAttributeType;

lAttributeType = pNode->GetNodeAttribute()->GetAttributeType();

**1.3.3 节点中层次Layer**

层次Layer：法线、纹理坐标等是存储在层次Layer中的，每个节点可以有多个层次，然后在每个层次中包含一套纹理，法线等。但是，通常我们只会用到一个层次，很多建模软件也只支持一个层次。比如在一个节点类型为eMESH的结点中层次概念如下所示：

mesh -------layer 0{ KFbxLayerElementNormal,KFbxLayerElementUV……}

      |

      |------layer 1{ KFbxLayerElementNormal,KFbxLayerElementUV………}

      |

      |-- ………………..

      |

      |------layer n{ KFbxLayerElementNormal,KFbxLayerElementUV………}

关于层次的常用函数：

//求pMesh中包含UV的层次数

pMesh->GetElementUVCount();

//获得第i层中的UV对象

KFbxGeometryElementUV\* leUV = pMesh->GetElementUV(i);

每种保存在Layer中的元素（如上面提到的UV）都继承于KFbxLayerElement，比如KFbxLayerElementNormal对应normal数据，KFbxLayerElementUV对应的UV数据，可以通过KFbxLayer中定义的各种Get函数得到，如GetElementNormal()和GetElementUV，返回需要的KFbxLayerElement，如果为空，则说明当前layer中没有这种元素。下面是关于KFbxLayerElement的类的大概的继承图。



KFbxLayerElement还中包含了两个非常重要的属性EMappingMode和EReferenceMode。

typedef enum

{

eNONE,

eBY\_CONTROL\_POINT, //对于每一个Control Point有一个贴图坐标

eBY\_POLYGON\_VERTEX, //对于polygon中每一个顶点有一个贴图坐标

eBY\_POLYGON, //对于一个polygon有一个贴图坐标

eBY\_EDGE,

eALL\_SAME

} EMappingMode;

typedef enum

{

eDIRECT,

eINDEX,

eINDEX\_TO\_DIRECT

} EReferenceMode;

* MappingMode定义了当前类型的元素如何映射到mesh上。举例来说，

对于KFbxLayerElementNormal，MappingMode=eBY\_POLYGON\_VERTEX表示如果一个顶点被n个多边形共享，那么这个顶点就有n条法线与之相对应；eBY\_CONTROL\_POINT则表示每个顶点无论被几个多边形共享，都只有一条normal；eBY\_POLYGON则表示构成多边形的n个顶点只对应着一条normal。

* ReferenceMode定义了如何访问相关的数据。同样举例来说，每个

KFbxLayerElement内部通常可能包含两个数组，分别称为DirectArray和IndexArray。如果reference mode为eDIRECT，则第i个顶点相对的element元素就在DirectArray的第i位置(第i个顶点的normal在KFbxLayerElementNormal.DirectArray[i]中) ，此时IndexArray为空。eINDEX\_TO\_DIRECT通常和eBY\_POLYGON\_VERTEX一起使用，因为一个控点可能对应多个值，所以这时必须用多边形顶点索引来获得某个多边形顶点所对应的值，例如

int id = pNormal->GetIndexArray().GetAt(vertexId);

KFbxVector4\* pTemp = pNormal->GetDirectArray().GetAt(id));

vertexId 是polygon顶点的索引。

**1.4 两个重要的节点结构：Mesh、Skeleton**

**1.4.1 mesh节点**

网格（Mesh）存储了模型结构的重要数据，包括顶点坐标，颜色，UV（纹理坐标），法线（Normal）等。这些都是我们在新平台中所要用到的。

Fbx文件中包含的Mesh节点，我们一般称为蒙皮节点。该节点由多边形组成，多边形至少是三角形，除此之外还可能是四边形，五边形等等。如果要将该Fbx中的mesh节点三角面片化，可用下面方法实现：

KFbxGeometryConverter converter(sdkManager);

mesh = converter.TriangulateMesh(mesh);

* 顶点坐标

Mesh中有两个概念控点（Control point）和顶点（Polygon vertex）。控点只包含位置信息，顶点就是多边形中的顶点，通过索引可以得到该多边形顶点位置，法线，纹理坐标等信息。

控点和顶点的关系：要看mesh中所有layer(通常只用第一层)中所有元素的MappingMode：如果是eBY\_CONTROL\_POINT，则控点的数量和顶点的数量是一一对应的，如果是eBY\_POLYGON\_VERTEX，则需要分裂控点，一个控点可以出现在不同的三角面中作为顶点，这时候一个控点对应多个顶点。一般来说当mesh由多个Polygon组成时，MappingMode的第二种eBY\_POLYGON\_VERTEX 更常用。

不论是控点还是顶点，在求位置坐标时用的是控点中的位置信息。对于多边形polygon中顶点的位置，是通过求该顶点所对应的控点在控点数组中的索引，最后求得的。

下面列几个求坐标时常用函数：

//求mesh中控点的数目

int lControlPointsCount = pMesh->GetControlPointsCount();

//求mesh中控点数组

KFbxVector4\* lControlPoints = pMesh->GetControlPoints();

若是要求在多边形polygon中顶点坐标，则需求该顶点对应的控点的索引

//如果这个mesh已经三角面片化，得到三角形的个数；如果没有三角面片化，则获得多边形的个数

int numPoly = mesh->GetPolygonCount();

for(int i=0;i<numPoly;i++)

{

int vn = mesh->GetPolygonSize(i); //mesh第i个poly的顶点个数

for(int j=0;j<vn;j++) //获得每个顶点的位置坐标

{

//获得第i个多边形中第j个顶点相对应的索引

int pControlPointsIndex = mesh->GetPolygonVertex(i,j);

KFbxVector4\* pTemp=lControlPoints[pControlPointsIndex];

}

}

* 纹理坐标

对于纹理坐标的求取，根据前面所写的MappingMode和ReferenceMode类型，总共分三种情况：

KFbxGeometryElementUV\* leUV = pMesh->GetElementUV(0);

1. MappingMode是eBY\_CONTROL\_POINT，而ReferenceMode

是eDIRECT时，

KFbxVector2\* pTemp;

pTemp = leUV->GetDirectArray().GetAt(lControlPointIndex)；

1. MappingMode是eBY\_CONTROL\_POINT，而ReferenceMode

是eINDEX\_TO\_DIRECT时，

KFbxVector2\* pTemp;

int id = leUV->GetIndexArray().GetAt(lControlPointIndex);

pTemp = leUV->GetDirectArray().GetAt(id);

1. MappingMode是eBY\_POLYGON\_VERTEX,无论ReferenceMode

是eDIRECT，还是eINDEX\_TO\_DIRECT，都用下面方法读取：

int lTextureUVIndex = pMesh->GetTextureUVIndex(i, j);

KFbxVector2\* pTemp;

pTemp = leUV->GetDirectArray().GetAt(lTextureUVIndex);

另外读取法向Normal、顶点颜色时基本类似，这里不再赘述。

**1.4.2 skeleton节点**

Fbx骨骼动画中的骨骼节点，一般用skeleton节点，即KFbxNode类型为eSKELETON的节点来表示。对于骨骼节点，我们要读取的信息包括：节点的关键帧个数、每个关键帧的时间、在关键帧时刻节点的本地变换矩阵。

要获得以上信息，首先要了解Fbx中动画组织方式。

对于一个含有动画信息的fbx模型，它可以包含一个或多个动画栈（KFbxAnimStack），每个动画栈存储一套动作。比如一个人的模型，他有两套动作，一套是由走到跑的动作，另一套是中枪倒下。那么每一套动作就可以用一个动画栈来表示。

一个动画栈中可以包含一个或多个动画层(KFbxAnimLayer)，每一个动画层存储一个动作，上例中由跑到走这套动作中总共有两个动作：跑、走，那么每个动作就可以用一个动画层来表示，这样在这个动画栈中就有两个动画层。

关于动画相关类的继承图如下：



在Fbx中关键帧是打在每个属性上的，动画层组织这些关键帧动画，动画栈再组织这些动画层。比如动画师建一个动画模型时，总的动画有十个关键帧。他在第2、8、10帧给平移属性Translation打了关键帧，而在第2、3、6、9帧给旋转属性Rotation打了关键帧，因此这两个属性的关键帧数目是不同的，而且有的关键帧的时间点也是不同的。不但Translation和Rotation上的关键帧数不同，也可能在同一属性如Translation的不同值上的关键帧数也是不同的，比如在TranslationX和TranslationY上的关键帧数可以不同。这就是Fbx中动画的大概组织方式。

下面用一个简单的例子，即Fbx中只有一个动画栈，动画栈中只有一个动画层，来演示一下如何获取TranslationX上的关键帧个数，以及关键帧时间。

//获取fbx的动画栈pAnimStack, 0代表第一个栈

KFbxAnimStack\* pAnimStack = KFbxCast<KFbxAnimStack>

(pScene->GetSrcObject(FBX\_TYPE(KFbxAnimStack), 0));

//获取动画栈中的动画层pAnimLayer, 0代表第一层。

KFbxAnimLayer\* pAnimLayer;

pAnimLayer= pAnimStack->GetMember(FBX\_TYPE(KFbxAnimLayer), 0);

//获取动画层中的动画曲线Curve，相当于属性的值。

KFbxAnimCurve\* pAnimCurveTX;

pAnimCurveTX=pNode->LclTranslation.GetCurve<KFbxAnimCurve>(pAnimLayer, KFCURVENODE\_T\_X);

//获取关于属性Translation的X的关键帧个数以及时间。

if (pAnimCurveTX)

{

KFCurve\* pCurve = pAnimCurveTX->GetKFCurve();

int pTXKeyCount = pCurve->KeyGetCount();

for (int k=0; k<pTXKeyCount; k++)

{

KTime temp = pCurve->KeyGetTime(k);

}

}

用同样的方法可以获得其他的属性的各个值的关键帧个数和时间。

求出关键帧时间后再求节点的本地变换矩阵可用函数

KFbxMatrix pMatrix;

pMatrix=pNode->EvaluateLocalTransform(KTime pTime);

KFbxVector4 pTranslation = pMatrix.GetT(); //平移

KFbxVector4 pRotation = pMatrix.GetR(); //旋转

KFbxVector4 pScale = pMatrix.GetS(); //缩放

**1.5 读取蒙皮信息**

**1.5.1 关于蒙皮信息**

前面我们已经可以读取关节动画的信息，下面读取附着在骨骼上的蒙皮信息。

在骨骼动画中，不是把Mesh直接放到世界坐标系中，Mesh只是作为Skin使用的，是依附于骨骼的，真正决定模型在世界坐标系中的位置和朝向的是骨骼。

Mesh节点是作为皮肤使用，蒙在骨骼之上。为了让普通的Mesh具有蒙皮的功能，必须添加蒙皮信息，即Skin info。我们知道Mesh是由顶点构成的，建模时这些顶点（就是我们前面说过的控点）是定义在模型自身坐标系的，即相对于Mesh原点的，而骨骼动画中决定模型顶点最终世界坐标的是骨骼，所以要让骨骼决定顶点的世界坐标，这就要将顶点和骨骼联系起来，Skin info正是起了这个作用。

顶点的Skin info包含影响该顶点的骨骼数目，这些骨骼作用于该顶点的权重(Skin weight)。

* + 1. **Fbx中的蒙皮结构**

在Fbx中蒙皮是由mesh节点来表示的，与普通mesh不同，这里的mesh加上了蒙皮信息即Skin info，我们要读取Skin info，首先应了解Fbx中的蒙皮结构。

Mesh |------skin 0 |-----Cluster 0{ Vertex 0, Vertex 1…………Vertex n1 }

      |   |-----Cluster 1{ Vertex 0, Vertex 1…………Vertex n2 }

      | |--- ………

      | |-----Cluster n{ Vertex 0, Vertex 1…………Vertex n3 }

      |------skin 1{ }

      |

      |-- ………………..

      |

      |------skin n{ }



从以上结构可以看出，Fbx的mesh中可以包含多重皮肤Skin，但绝大多数情况下，我们只用到第一层皮肤Skin 0。在每层皮肤下一层是骨骼，在这里叫群聚Cluster，即和这层皮肤相关的骨骼有哪些。在每个骨骼下有顶点Vertex，即每一个骨骼影响的顶点，以及对这些顶点的权重都可以在这里得到。



下面是获得蒙皮信息的常用代码。

//获得pMesh中皮肤的个数，皮肤个数通常为1

int lSkinCount=pMesh->GetDeformerCount(KFbxDeformer::eSKIN);

//获得pMesh的第0层skin，若是多层皮肤，可用循环将0改为皮肤的索引i，就得到第i层皮肤

KFbxSkin\* lSkin=(KFbxSkin\*)pMesh->GetDeformer(0,

KFbxDeformer::eSKIN);

//获得lSkin所包含的骨骼数

int lClusterCount = lSkin->GetClusterCount();

for(int i = 0;i < lClusterCount; ++i) //对皮肤中的骨骼进行循环

{

int lVertexIndexCount = lCluster->GetControlPointIndicesCount();

for (int k = 0; k < lVertexIndexCount; ++k)

{

//求出lCluster第k个顶点的索引（在mesh中所有顶点中的索引）

int lIndex = lCluster->GetControlPointIndices()[k];

if (lIndex >= lVertexCount)

continue;

double lWeight = lCluster->GetControlPointWeights()[k];

if (lWeight == 0.0)

{

continue;

}

}

}