**void FBXExporter::ProcessMesh(FbxNode\* inNode)  
{  
 FbxMesh\* currMesh = inNode->GetMesh();  
  
 mTriangleCount = currMesh->GetPolygonCount();  
 int vertexCounter = 0;  
 mTriangles.reserve(mTriangleCount);  
  
 for (unsigned int i = 0; i < mTriangleCount; ++i)  
 {  
 XMFLOAT3 normal[3];  
 XMFLOAT3 tangent[3];  
 XMFLOAT3 binormal[3];  
 XMFLOAT2 UV[3][2];  
 Triangle currTriangle;  
 mTriangles.push\_back(currTriangle);  
  
 for (unsigned int j = 0; j < 3; ++j)  
 {  
 int ctrlPointIndex = currMesh->GetPolygonVertex(i, j);  
 CtrlPoint\* currCtrlPoint = mControlPoints[ctrlPointIndex];  
  
  
 ReadNormal(currMesh, ctrlPointIndex, vertexCounter, normal[j]);  
 // We only have diffuse texture  
 for (int k = 0; k < 1; ++k)  
 {  
 ReadUV(currMesh, ctrlPointIndex, currMesh->GetTextureUVIndex(i, j), k, UV[j][k]);  
 }  
  
  
 PNTIWVertex temp;  
 temp.mPosition = currCtrlPoint->mPosition;  
 temp.mNormal = normal[j];  
 temp.mUV = UV[j][0];  
 // Copy the blending info from each control point  
 for(unsigned int i = 0; i < currCtrlPoint->mBlendingInfo.size(); ++i)  
 {  
 VertexBlendingInfo currBlendingInfo;  
 currBlendingInfo.mBlendingIndex = currCtrlPoint->mBlendingInfo[i].mBlendingIndex;  
 currBlendingInfo.mBlendingWeight = currCtrlPoint->mBlendingInfo[i].mBlendingWeight;  
 temp.mVertexBlendingInfos.push\_back(currBlendingInfo);  
 }  
 // Sort the blending info so that later we can remove  
 // duplicated vertices  
 temp.SortBlendingInfoByWeight();  
  
 mVertices.push\_back(temp);  
 mTriangles.back().mIndices.push\_back(vertexCounter);  
 ++vertexCounter;  
 }  
 }  
  
 // Now mControlPoints has served its purpose  
 // We can free its memory  
 for(auto itr = mControlPoints.begin(); itr != mControlPoints.end(); ++itr)  
 {  
 delete itr->second;  
 }  
 mControlPoints.clear();  
}**

**FBX 컨트롤포인트 즉 FBXSDK버텍스가 필요하다.**

**따라서 메시의 모든 버텍스 위치를 얻어와야 한다.**

**씬을 생성한 후에 이를 통해 정점의 위치를 잡아줘야 하며,**

**필요한 버텍스의 노말을 인덱스로 접근해야함.**