MeshFromOBJ10 Sample

Collapse All

This sample shows how an **ID3DX10Mesh** object can be created from mesh data stored in a Wavefront Object file (.obj). This sample is an adaption of the Direct3D 9 sample MeshFromOBJ, which demonstrates best practices when porting to Direct3D 10 from Direct3D 9. The sample code has been constructed to facilitate side-by-side comparisons between versions to assist with learning. For additional details on how to load and parse the .obj file, see the documentation for MeshFromOBJ, the Direct3D 9 sample.

Path

Source	*SDK root* \Samples\C++\Direct3D10\MeshFromOBJ10
Executable	*SDK root* \Samples\C++\Direct3D10\Bin\ *x86 or x64* \MeshFromOBJ10.exe

Sample Overview

ID3DX10Mesh can be considered a DirectX 10 version of **ID3DXMesh**. However, it provides more functionality, such as the ability to test whether the entire mesh, or a particular subset, intersects with a ray.

There is a minor difference in the way that mesh subsets are rendered between the Direct3D 10 version and the Direct3D 9 version of the sample. In both versions of the sample, mesh data is pre-sorted by calling the optimization methods with the D3DXMESHOPT_ATTRSORT flag (described in **D3DXMESHOPT**). The code to render the mesh in the Direct3D 9 sample looks like the following:

```
for ( UINT iAttr = 0; iAttr < nMaterials; ++iAttr )
{
    // apply effect parameters for material iAttr
    // ...

    pD3DXMesh->DrawSubset( iAttr );
}
```

This code iterates through all the materials and passes the material ID to the **ID3DXMesh::DrawSubset** method. **ID3DXMesh::DrawSubset** does nothing if the mesh doesn't use that material (that is, the material ID doesn't appear in the per-face attribute ID array of the mesh).

Meanwhile, the code to do the same thing in Direct3D 10 sample using **ID3DX10Mesh::DrawSubset** looks like:

```
// get the attribute table somewhere before rendering the mesh
pD3DX10Mesh->GetAttributeTable( NULL, &nAttribTableEntries );
D3DX10_ATTRIBUTE_RANGE* pAttribTable = new D3DX10_ATTRIBUTE_RANGE[nAttribTableEntries];
pD3DX10Mesh->GetAttributeTable( pAttribTable, &nAttribTableEntries );

// ...

for ( UINT iAttr = 0; iAttr < nAttribTableEntries; ++iAttr )
{
    // apply effect parameters for material pAttribTable[iAttr].AttribId
    // ...

    pD3DX10Mesh->DrawSubset( iAttr );
}
```

This version of the code iterates through all the attribute table entries of the mesh to find and apply corresponding material by querying the *AttribId* field of the attribute table. Then it passes the attribute table ID to **ID3DX10Mesh::DrawSubset** to render the subset.

In many cases, the material list may contain more materials than a single mesh actually uses. This is particularly true if you construct your code in such a way that multiple mesh objects share the same material list. Therefore, the latter code segment performs more efficiently than the former, because looping through the material list for rendering each single mesh object may introduce redundant iterations.