OptimizedMesh Sample

Collapse All

This OptimizedMesh Sample sample demonstrates the different types of meshes D3DX can load and optimize, as well as the different types of underlying primitives it can render. An optimized mesh has its vertices and faces reordered so that rendering performance can be improved.

Path

Source	*SDK root* \Samples\C++\Direct3D\Meshes\OptimizedMesh
Executable	*SDK root* \Samples\C++\Direct3D\Bin\ *x86 or x64* \OptimizedMesh.exe

How the Sample Works

The sample can be divided into two areas: loading and rendering.

Loading

The loading code first initializes the mesh from data in the mesh file and calls OptimizeMeshData three times, each with a different optimization type. The optimization types used are D3DXMESHOPT_ATTRSORT, D3DXMESHOPT_VERTEXCACHE, and D3DXMESHOPT_STRIPREORDER. The optimization type is passed to **ID3DX10Mesh::Optimize** which reorders the mesh's vertices and faces accordingly.

After the meshes are set up, OptimizeMeshData creates a number of index buffers to hold the face information in the form of triangle strips. First, it calls **D3DXConvertMeshSubsetToSingleStrip** once for each material in the mesh. This returns an index buffer that represents the mesh triangles of that material with a single triangle strip. This index buffer can then be used to render the mesh. Next, OptimizeMeshData calls **D3DXConvertMeshSubsetToStrips** for each material in the mesh, and returns an index buffer representing a series of triangle strips. To return a single triangle strip, call **D3DXConvertMeshSubsetToSingleStrip** instead.

After the loading is complete, the sample has three sets of mesh data. Each set of mesh data includes an **ID3DXMesh** optimized with a different flag, one index buffer per material that represents the triangle subset as a single strip, and one index buffer per material that represents the subset as a series of strips.

Rendering

The rendering code is simpler compared to the loading code, because the mesh data sets have already been set up during the load time and are ready for use directly with rendering calls. To render, the sample calls DrawMeshData. This function takes a mesh data set and performs the rendering with it. Based on the current optimization setting, the appropriate mesh data set is passed to DrawMeshData.

Inside DrawMeshData, the mesh is rendered according to the primitive type selected by the user. For triangle lists, the mesh object's **ID3DXBaseMesh::DrawSubset** is called once per material, because the mesh stores its geometry as triangle lists internally. For strips, DrawMeshData sets up primitive sources by calling **IDirect3DDevice9::SetStreamSource** and SetIndices(). The index buffer that is passed to **IDirect3DDevice9::SetIndices** is one of the index buffers created in OptimizeMeshData. If the mesh is to be rendered with a single triangle strip, the index buffer returned by **D3DXConvertMeshSubsetToSingleStrip** is used. If multiple triangle strips are desired, the buffer returned by **D3DXConvertMeshSubsetToStrips** is used instead. Once the vertex and index data are properly initialized, DrawMeshData renders the mesh by calling **IDirect3DDevice9::DrawIndexedPrimitive**.