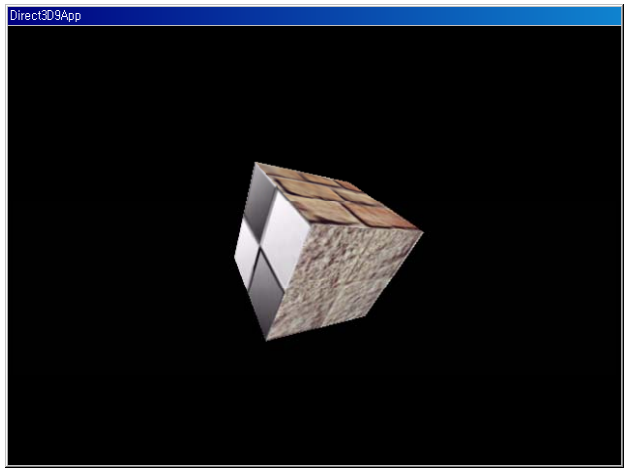
Mesh

305890
Spring 2010
5/28/2010
Kyoung Shin Park

Overview

- To gain an understanding of the internal data organization of an ID3DXMesh object
- To find out how to create, optimize, and render an ID3DXMesh object
- To learn how to load the data of an .X file into an ID3DXMesh object
- To become familiar with several D3DX mesh-related utility functions

Mesh Part I



Mesh Part I

- Geometry Information
- Subsets and the Attribute buffer
- Drawing
- Adjacency Information
- Optimizing
- The Attribute Table
- Cloning a Mesh
- Creating a Mesh (D3DXCreateMesh)
- .X Files
- Bounding Volumes

Geometry Information

- **ID3DXMesh** inherits the majority of its functionality from its parent, **ID3DXBaseMesh**.

Geometry Information

- The **ID3DXBaseMesh** interface contains a **vertex buffer** that stores the vertices of the mesh, and an **index buffer** that defines how these vertices are put together to form the triangles of the mesh.

```
HRESULT ID3DXMesh::GetVertexBuffer(LPDIRECT3DVERTEXBUFFER9*
    ppVB);
HRESULT ID3DXMesh::GetIndexBuffer(LPDIRECT3DINDEXBUFFER9* ppIB);

IDirect3DVertexBuffer9* vb = 0;
Mesh->GetVertexBuffer( &vb );
IDirect3DIndexBuffer9* ib = 0;
Mesh->GetIndexBuffer( &ib );
```

Geometry Information

- Lock/Unlock the buffers to read or write

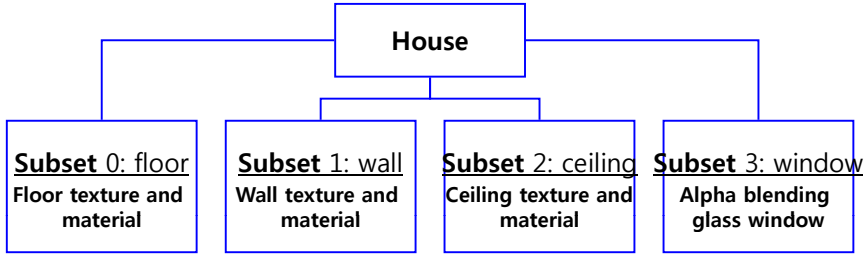
```
HRESULT ID3DXMesh::LockVertexBuffer(DWORD Flags,BYTE** ppData);
HRESULT ID3DXMesh::LockIndexBuffer(DWORD Flags, BYTE** ppData);
HRESULT ID3DXMesh::UnlockVertexBuffer( );
HRESULT ID3DXMesh::UnlockIndexBuffer( );
```

- Additional ID3DXMesh methods to obtain various info

```
GetDeclaration(D3DVERTEXELEMENT9, Declaration[MAX_FVF_DECL_SIZE]);
GetNumVertices( ); // num of vertices
GetNumBytesPerVertex( ); // bytes per vertex
GetNumFaces( ); // num of faces
GetOptions(); // returns whose bits are bit-flags that describe various
    options about the mesh such as what memory pool it is stored in, the
    format of indices, and whether it is static or dynamic
```

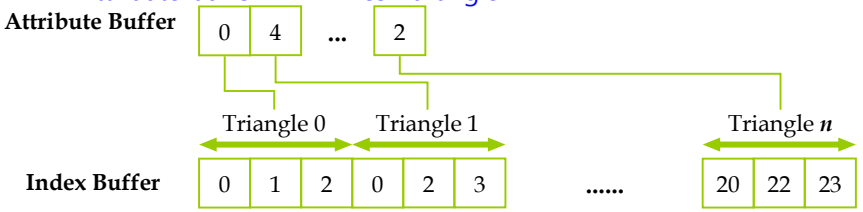
Subset and Attribute Buffer

- Subset
  - A subset is a group of triangles in a mesh that can all be rendered using the same attribute.
  - A mesh (e.g., house) may be divided into several attributes (e.g., floor, wall, ceiling, window)
- Attribute
  - Material, texture, and render states



Subset and Attribute Buffer

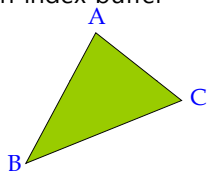
- Attribute ID
  - We label each subset by specifying a **unique positive integer** value for that subset
  - Each triangle in a mesh is given an attribute ID that specifies the subset in which the triangle lives
- Attribute buffer
  - Attribute IDs for triangles are stored in a mesh's attribute buffer (DWORD array)
  - Attribute buffer # == Mesh triangle #



Subset and Attribute Buffer

- 1:1 Correspondence
  - Entry i in the attribute buffer corresponds with triangle i in the index buffer
  - Triangle ***i*** is defined by the following 3 indices in index buffer

$$A = i \cdot 3$$
$$B = i \cdot 3 + 1$$
$$C = i \cdot 3 + 2$$



- Access the attribute buffer by locking it

```
DWORD *buffer = 0;
Mesh->LockAttributeBuffer(lockingFlags, &buffer);

// read or write to attribute buffer...
Mesh->UnlockAttributeBuffer( );
```

Mesh Drawing

- DrawSubset
  - Draw triangles of a particular subset specified by **Attrib ID**

```
HRESULT ID3DXMesh::DrawSubset(DWORD AttribId);
```

  - E.g., draw all the triangles that live in **subset 0**

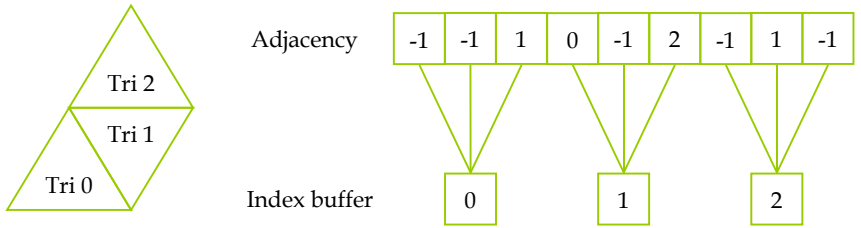
```
Mesh->DrawSubset(0);
```

  - E.g., draw all the subsets of the mesh to draw an entire mesh

```
HR(mFX->BeginPass(0));
for (int i=0; i<mMtrl.size(); ++i) {
   HR(mFX->SetValue(mhMtrl, &mMtrl[i], sizeof(Mtrl)));
   if (mTex[i] != 0) HR(mFX->SetTexture(mhTex, mTex[i]));
   else HR(mFX->SetTexture(mhTex, mWhiteTex));
   HR(mFX->CommitChanges());
   HR(mMesh->DrawSubset(i));
}
HR(mFX->EndPass());
```

Adjacency Array

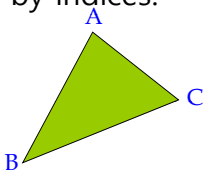
- Adjacency array
  - Optimizing operations needs to know the triangles that are adjacent to a given triangle
  - The adjacency array is a DWORD array, where each entry contains an **index** identifying a triangle in the mesh



Adjacency Array

- An entry *i* refers to the *i*th triangle formed by indices:

$$A = i \cdot 3$$
$$B = i \cdot 3 + 1$$
$$C = i \cdot 3 + 2$$



- ULONG_MAX indicating that the particular edge does not have an adjacent triangle
  - We can also use -1 to denote this because assigning -1 to DWORD (unsigned 32-bit integer) results in ULONG_MAX
  - ULONG_MAX == 4,294,967,295 == –1
- Since each triangle has 3 edges, it can have up to 3 adjacent triangles
  - # of elements = ID3DXBASEMESH::GetNumFaces( ) * 3

Adjacency Array

- Generating adjacency array
  - Many of the D3DX mesh creation functions can output the adjacency information
  - Use GenerateAdjacency()

```
HRESULT ID3DXMesh::GenerateAdjacency (
        FLOAT fEpsilon,
        DWORD* pAdjacency);
```

  - fEpsilon – value to be considered equal
  - pAdjacency – a pointer to an array of DWORD that is to be filled with the adjacency information, (in bytes 3 * ID3DXMesh::GetNumFaces*sizeof(DWORD))
- E.g.,

```
vector<DWORD> adjacencyInfo(Mesh->GetNumFaces()*3);
Mesh->GenerateAdjacency( 0.001f, &adjacencyInfo[0] );
```

Mesh Optimizing

- Optimizing
  - Vertices and indices of a mesh can be reorganized to render the mesh more efficiently

```
HRESULT ID3DXMesh::OptimizeInplace (
    DWORD Flags,
    CONST DWORD* pAdjacencyIn,
    DWORD* pAdjacencyOut,
    DWORD* pFaceRemap,
    LPD3DXBUFFER* ppVertexRemap);
```

  - Flags – kinds of optimizations to perform
  - pAdjacencyIn – pointer to an array containing adjacency info
  - pAdjacencyOut – pointer to an array containing optimized adjacency info
  - pFaceRemap – pointer to a DWORD array to filled with the face remap info. The array should be of size D3DXMesh :: GetNumFaces().
  - ppVertexRemap – pointer to an ID3DXBuffer that will be filled with the vertex remap info. This buffer should contain ID3DXMesh :: GetNumVertices() many vertices.

Mesh Optimizing

- Flags – D3DXMESHOPT flags
  - D3DXMESHOPT_COMPACT – removes unused geometry that the mesh may contain.
  - D3DXMESHOPT_ATTRSORT – sorts the geometry by attribute and generates an attribute table. This allows DrawSubset to be more efficient.
  - D3DXMESHOPT_VERTEXCACHE – reorganizes the geometry of the mesh to take better advantage of the vertex cache. (recommended)
  - D3DXMESHOPT_STRIPREORDER – reorganizes the geometry so that triangle strips can be as long as possible.
  - D3DXMESHOPT_IGNOREVERTES – optimizes index info only; ignores vertices.

Mesh Optimizing

- E.g.

```
// Get the adjacency info of the non-optimized mesh
DWORD* adjacencyInfo = new DWORD[Mesh->GetNumFaces()*3];
Mesh->GenerateAdjacency( 0.0f, adjacencyInfo );

// Array to hold optimized adjacency info
vector<DWORD> optimizedAdjacencyInfo(Mesh->GetNumFaces()*3);
Mesh->OptimizeInplace(
    D3DXMESH_MANAGED | D3DXMESHOPT_COMPACT |
    D3DXMESHOPT_ATTRSORT | D3DXMESHOPT_VERTEXCACHE,
    adjacencyInfo,
    optimizedAdjacencyInfo,
    0,
    0 );
```

Mesh Optimizing

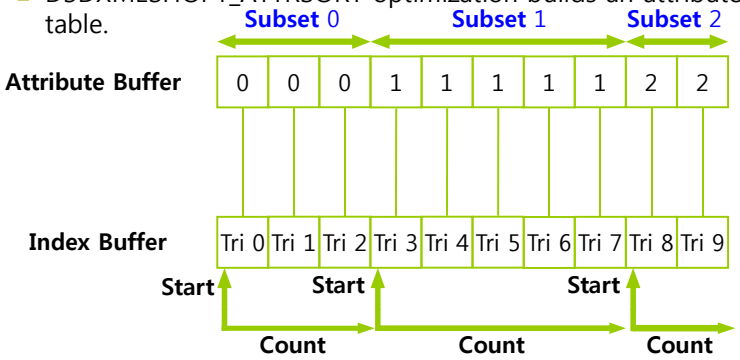
- Optimize()
  - Similar to ID3DXBaseMesh::CloneMesh

```
HRESULT ID3DXMesh::Optimize (
    DWORD Flags,
    CONST DWORD* pAdjacencyIn,
    DWORD* pAdjacencyOut,
    DWORD* pFaceRemap,
    LPD3DXBUFFER* ppVertexRemap,
    LPD3DXMESH* ppOptMesh);
```

  - ppOptMesh – outputting an optimized mesh

Attribute Table

- Optimize with D3DXMESHOPT_ATTRSORT flag
  - The geometry of the mesh is sorted by its attribute so that the geometry of a particular subset exists as a contiguous block in the vertex/index buffers
  - D3DXMESHOPT_ATTRSORT optimization builds an attribute table.



Attribute Table

- D3DXATTRIBUTERANGE structure
  - Each entry in the attribute table corresponds to a subset of the mesh and specifies the block of memory in the vertex/index buffers where geometry for the subset resides.

```
typedef struct _D3DXATTRIBUTERANGE {
    DWORD AttribId;
    DWORD FaceStart;
    DWORD FaceCount;
    DWORD VertexStart;
    DWORD VertexCount;
} D3DXATTRIBUTERANGE
```

  - AttribId – subset ID
  - FaceStart, FaceCount – an offset into the index buffer identifying the start of the triangles & the number of faces
  - VertexStart, VertexCount – an offset into the vertex buffer identifying the start of the vertices & the number of vertices

Attribute Table

- To access the attribute table of a mesh:

```
HRESULT ID3DXMesh::GetAttributeTable (
        D3DXATTRIBUTERANGE* pAttribTable,
        DWORD* pAttribTableSize);
```

  - Returns the number of attributes in the attribute table
  - Fill an array of D3DXATTRIBUTERANGE struct with attribute data
- E.g.,

```
// To get the number of elements of attribute table, we pass in 0
DWORD numSubsets = 0;
Mesh->GetAttributeTable( 0, &numSubsets );
// Then, fill a D3DXATTRIBUTERANGE array with attribute table
D3DXATTRIBUTERANGE table =
                        new D3DXATTRIBUTERANGE[numSubsets];
Mesh->GetAttributeTable( table, &numSubsets);
```

Attribute Table

- To set the attribute table

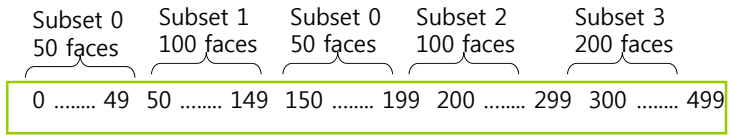
```
HRESULT ID3DXMesh:: SetAttributeTable(
    CONST D3DXATTRIBUTERANGE* pAttribTable,
    DWORD cAttribTableSize);
```

- E.g.,

```
// directly set the attribute table
D3DXATTRIBUTERANGE attributeTable[12];
// manually define and fill the attributeTable array with data ...
Mesh->SetAttributeTable( attributeTable, 12 );
```

Subset Example

- Example of a mesh containing several subsets




```
// the last element 0 in faceCount indicates the last element
DWORD faceCount[] = {50, 100, 50, 100, 200, 0};
DWORD subsetNum[] = {0, 1, 0, 2, 3};
if(FAILED(SetSubsets(pMesh, faceCount, subsetNum)) {
    // handle error
}
```

Subset Example

```
DWORD SetSubsets(ID3DXMesh* pMesh, DWORD *pFaceCount,
                                DWORD *pSubsetNum) {
    // get the maximum size of attribute buffer
    DWORD numFaces = pMesh->GetFaceCount(); // get face count
    DWORD *attribBuf;
    HRESULT hr;
    if (SUCCEEDED (hr=pMesh->LockAttributeBuffer (D3DLOCK_DISCARD,
    &attribBuf))) {
        DWORD faceNum = 0; // initialize face counter
        for (int i=0; pFaceCount[i]; i++) { // loop through the subsets
            // make sure there are enough faces for this subset
            if (faceNum + pFaceCount[i] >= numFaces) { // not enough faces
                pMesh->UnlockAttributeBuffer(); // unlock attribute buffer
                return E_INVALIDARG; // return err
            }
            for (int j=0; j<pFaceCount[i]; j++) {
                attribBuf[faceNum] = pSubsetNum[i]; // set subset number of each
    face
                faceNum++; // increase face counter
            }
        }
```

Subset Example

```
        pMesh->UnlockAttributeBuffer(); // unlock attribute buffer
        // allocate storage and generate adjacency data
        // doesn't need to create a new adj buffer if there is already
        DWORD *pAdj = new DWORD[numFaces*3];
        if (!pAdj) return E_OUTOFMEMORY;
        if (FAILED(hr = pMesh->GenerateAdjaceny(0.0f, pAdj))) {
            delete pAdj; return hr;
        }
        // optimize the mesh with attribute D3DXMESHOPT_ATTRSORT
        if (FAILED(hr = pMesh->OptimizeInplace(
                    D3DXMESHOPT_VERTEXCACHE, pAdj, NULL, NULL, NULL))) {
            delete pAdj; return hr;
        }
        delete pAdj; // de-allocate adjacency data storage
    }
    else
        return hr;
    return S_OK; // return success
}
```

Cloning

- Cloning a mesh, to add space for normals, texture coordinates, colors, etc that are not originally present.

```
HRESULT ID3DXMesh::CloneMeshFVF (DWORD Options,
                                 DWORD FVF,
                                 LPDIRECT3DDEVICE9 pDevice,
                                 LPD3DXMESH* ppCloneMesh);
```

  - Options – mesh cloning option flag
  - FVF – can have different FVF
  - pDevice – a device
  - ppCloneMesh – a cloned mesh
- E.g,

```
ID3DMesh* clone = 0;
Mesh->CloneMeshFVF(Mesh->GetOptions(), // same as original
                   D3DFVF_XYZ | D3DFVF_NORMAL, // cloned mesh FVF
                   Device, &clone);
```

Cloning

- Cloning a mesh

```
HRESULT ID3DXMesh::CloneMesh(DWORD Options,
                             const D3DXVERTEXELEMENT9 *pDeclaration,
                             LPDIRECT3DDEVICE9 pDevice,
                             LPD3DXMESH* ppCloneMesh);
```

  - Options – mesh cloning option flag
  - pDeclaration – an array of D3DXVERTEXELEMENT9 elements, specify the vertex format for the vertices in the output mesh
  - pDevice – a device
  - ppCloneMesh – a cloned mesh
- E.g,

```
ID3DMesh* clone = 0;
Mesh->CloneMesh(D3DXMESH_SYSTEMMEM, elements,
                Device, &clone);
```

Creating a Mesh

- Create a ID3DXMesh object
  - Shape creation – create a primitive shape mesh
    - D3DXCreateBox, D3DXCreateTeapot, ..
  - Basic mesh creation – create a mesh with specifying format
    - D3DXCreateMesh, D3DXCreateMeshFVF
  - Mesh file – load a mesh from X file
    - D3DXLoadMeshFromX
  - Mesh operations – create a new mesh from a mesh
    - OptimizeInplace, Optimize, CloneMeshFVF

Creating Mesh – Shape Creation

- Shape Creation

D3DXCreateBox(LPDIRECT3DDEVICE9 pDevice, FLOAT Width, FLOAT Height, FLOAT Depth, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency);

D3DXCreateCylinder(LPDIRECT3DDEVICE9 pDevice, FLOAT Radius1, FLOAT Radius2, FLOAT Length, UINT Slices, UINT Stacks, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency);

D3DXCreatePolygon(LPDIRECT3DDEVICE9 pDevice, FLOAT Length, UINT Sides, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency);

D3DXCreateSphere(LPDIRECT3DDEVICE9 pDevice, FLOAT Radius, UINT Slices, UINT Stacks, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency);

D3DXCreateTeapot(LPDIRECT3DDEVICE9 pDevice, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency);

D3DXCreateText(LPDIRECT3DDEVICE9 pDevice, HDC hDC, LPCTSTR pText, FLOAT Deviation, FLOAT Extrusion, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency, LPGLYPHMETRICSFLOAT pGlyphMetrics);

D3DXCreateTorus(LPDIRECT3DDEVICE9 pDevice, FLOAT InnerRadius, FLOAT OuterRadius, UINT Sides, UINT Rings, LPD3DXMESH **ppMesh, LPD3DXBUFFER **ppAdjacency);

Creating Mesh – Basic Mesh Creation

- Basic Mesh Creation
  1. Determine the number of faces and vertices for a mesh
  2. Allocate vertex/index/attribute buffer for D3DXCreateMeshFVF
  3. Fill the mesh data for each buffer
- Create a mesh with given faces and vertices

```
D3DXCreateMeshFVF(DWORD NumFaces, // index buffer size
                  DWORD NumVertices, // vertex buffer size
                  DWORD Options, // D3DXMESH flag
                  DWORD FVF, // FVF flag
                  LPDIRECT3DDEVICE9 pD3DDevice, // IDirect3DDevice9
                  LPD3DXMESH *ppMesh); // ID3DXMesh
```

Creating Mesh – Basic Mesh Creation

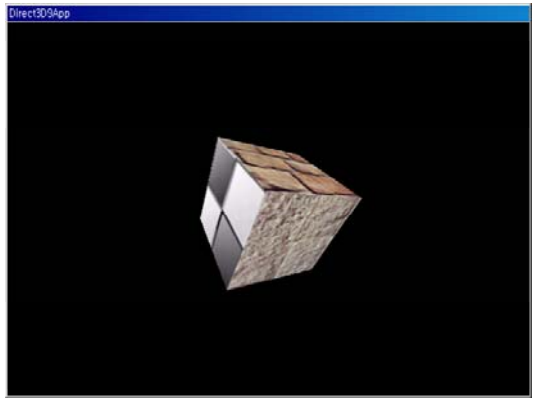
- D3DXCreateMesh
  - Use **D3DVERTEXELEMENT9** structure instead of FVF

```
D3DXCreateMesh(DWORD NumFaces, DWORD NumVertices,
               DWORD Options,
               const LPD3DVERTEXELEMENT9 *pDeclaration,
               LPDIRECT3DDEVICE9 pD3DDevice,
               LPD3DXMESH **ppMesh);
```

  - pDeclaration
    - a mesh vertex format using D3DVERTEXELEMENT9 structure
- To get pDeclaration using D3DXDeclarationFromFVF

```
HRESULT D3DXDeclaratorFromFVF (DWORD FVF,
                D3DVERTEXELEMENT9 Declaration[MAX_FVF_DECL_SIZE]);
typedef enum { MAX_FVF_DECL_SIZE = 18 } MAX_FVF_DECL_SIZE;
```

Example: D3DXCreateMeshFVF



1. Create a mesh
   - D3DXMesh
2. Fill geometry
   - Vertex, Index buffer
3. Set subsets for mesh
   - Attribute buffer
4. Create an adjacency info for a mesh
   - Adjacency buffer
5. Optimize
   - Optimize
6. Mesh drawing
   - Draw subsets

Example: D3DXCreateMeshFVF

```
#include "d3dUtility.h"
#include <vector>
ID3DXMesh* Mesh = 0;
const DWORD NumSubsets = 3;
IDirect3DTexture9* Textures[3] = {0, 0, 0};    // texture for each subset
std::ofstream OutFile; // for mesh data dump
struct Vertex {
    Vertex() {}
    Vertex(float x, float y, float z, float nx, float ny, float nz, float u, float v) {
        _x=x; _y=y; _z=z; _nx=nx; _ny=ny; _nz=nz; _u=u; _v=v; }
    float _x, _y, _z, _nx, _ny, _nz, _u, _v;
    static const DWORD FVF;
};
const DWORD Vertex::FVF = D3DFVF_XYZ|D3DFVF_NORMAL|D3DFVF_TEX1;
void Cleanup() {
    d3d::Release<ID3DMesh*>(Mesh);
    for (i=0; i<3; i++) d3d::Release<IDirect3DTexture9*>(Textures[i]);
}
```

Example: D3DXCreateMeshFVF

```
bool Setup() {
    HRESULT hr = 0;
    hr = D3DXCreateMeshFVF(12, 24, D3DXMESH_MANAGED, Vertex::FVF, Device,
    &Mesh); // 12 triangles & 24 vertices
    if (FAILE(hr)) {
        ::MessageBox(0, "D3DXCreateMeshFVF() – FAILED", 0, 0); return false; }
    Vertex* v  = 0;
    Mesh->LockVertexBuffer(0, (void**) &v);
    v[0] = Vertex(-1.0f, -1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f); // front
    v[1] = Vertex(-1.0f,  1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f);
    v[2] = Vertex( 1.0f,  1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f);
    v[3] = Vertex( 1.0f, -1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f);
    v[4] = Vertex(-1.0f, -1.0f,  1.0f, 0.0f, 0.0f,  1.0f, 0.0f, 0.0f); // back
    .....
    v[20] = Vertex( 1.0f, -1.0f, -1.0f, 1.0f, 0.0f,  0.0f, 0.0f, 0.0f); // right
    v[21] = Vertex( 1.0f,  1.0f, -1.0f, 1.0f, 0.0f,  0.0f, 0.0f, 1.0f);
    v[22] = Vertex( 1.0f,  1.0f,  1.0f, 1.0f, 0.0f,  0.0f, 1.0f, 1.0f);
    v[23] = Vertex( 1.0f, -1.0f,  1.0f, 1.0f, 0.0f,  0.0f, 1.0f, 0.0f);
    Mesh->UnlockVertexBuffer();
```

Example: D3DXCreateMeshFVF

```
    WORD* i = 0;
    Mesh->LockIndexBuffer(0, (void**) &i); // index data
    i[0] = 0;   i[1] = 1;     i[2] = 2;     i[3] = 0;    i[4] = 2;     i[5] = 3; // front
    i[6] = 4;   i[7] = 5;     i[8] = 6;     i[9] = 4;    i[10] = 6;    i[11] = 7; // back
    i[12] = 8; i[13] = 9;    i[14] = 10;  i[15] = 8;  i[16] = 10;  i[17] = 11; // top
    ....
    i[30] = 20; i[31] = 21; i[32] = 22; i[33] = 20; i[34] = 22; i[35] = 23; // right
    Mesh->UnlockIndexBuffer();
    DWORD* attributeBuffer = 0; // specify the subset
    Mesh->LockAttributeBuffer(0, &attributeBuffer);
    for (int a=0; a<4; a++) attributeBuffer[a] = 0; // first two faces – subset0
    for (int b=4; b<8; b++) attributeBuffer[b] = 1; // next two faces – subset1
    for (int c=8; c<12; c++) attributeBuffer[c] = 2; // last two faces – subset2
    Mesh->UnlockAttributeBuffer();
```

Example: D3DXCreateMeshFVF

```
    // optimize the mesh to generate an attribute table
    std::vector<DWORD> adjacencyBuffer(Mesh->GetNumFaces() * 3);
    Mesh->GenerateAdjacency(0.0f, &adjacencyBuffer[0]);
    hr = Mesh->OptimizeInplace(D3DXMESHOPT_ATTRSORT |
            D3DXMESHOPT_COMPACT | D3DXMESHOPT_VERTEXCACHE,
            &adjacencyBuffer[0], 0, 0, 0);
    // dump the mesh data to file
    OutFile.open("MeshDump.txt");
    dumpVertices(OutFile, Mesh);
    dumpIndices(OutFile, Mesh);
    dumpAttributeTable(OutFile, Mesh);
    dumpAttributeBuffer(OutFile, Mesh);
    dumpAdjacencyBuffer(OutFile, Mesh);
    OutFile.close();
    // load textures
    D3DXCreateTextureFromFile(Device, "brick0.jpg", &Textures[0]);
    D3DXCreateTextureFromFile(Device, "brick1.jpg", &Textures[1]);
    D3DXCreateTextureFromFile(Device, "checker.jpg", &Textures[2]);
    Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
    Device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_POINT);
```

Example: D3DXCreateMeshFVF

```
    // disable lighting
    Device->SetRenderState(D3DRS_LIGHTING, false);
    // set camera
    D3DXVECTOR3 pos(0.0f, 0.0f, -4.0f);
    D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
    D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
    D3DXMATRIX V;
    D3DXMatrixLookAtLH(&V, &pos, &target, &up);
    // set projection matrix
    D3DXMatrix proj;
    D3DXMatrixPerspectiveFovLH(&proj, D3DX_PI * 0.5f,
                                        (float)Width / (float) Height, 1.0f,
    1000.0f);
    Device->SetTransform(D3DTS_PROJECTION, &proj);
    return true;
}
```

Example: D3DXCreateMeshFVF

```
bool Display(float timeDelta) {
    if (Device) {
        D3DXMATRIX xRot, yRot, World;
        static float y = 0.0f;
        D3DXMatrixRotationX(&xRot, D3DX_PI * 0.2f);
        D3DXMatrixRotationY(&yRot, y);
        y += timeDelta;
        if (y >= 6.28f) y = 0.0f;
        World = xRot * yRot;
        Device->SetTransform(D3DTS_WORLD, &World);
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                              0x00000000, 1.0, 0);
        Device->BeginScene();
        for (int i=0; i < NumSubsets; i++) {
            Device->SetTexture(0,  Textures[i]);
            Mesh->DrawSubset(i);
        }
        Device->EndScene();
        Device->Present(0, 0, 0, 0);
    } return true; }
```

ID3DXBuffer

- ID3DXBuffer
  - Used as a data buffer, storing vertex, adjacency, and material information during mesh optimization and loading operations

```
LPVOID GetBufferPointer(VOID); // retrieve a pointer to the data
DWORD GetBufferSize(VOID); // retrieve the total size of the data
```

  - Must manage the type of data

```
DWORD* info = (DWORD*)adjacencyInfo->GetBufferPointer( );
D3DXMATERIAL *mtrls =
      (D3DXMATERIAL*)mtrlBuffer->GetBufferPointer( );
```

  - Release object after uses, to prevent memory leak

```
adjacencyInfo->Release( );
mtrlBuffer->Release( );
```

ID3DXBuffer

- Create a ID3DXBuffer

```
HRESULT D3DXCreateBuffer (
                DWORD NumBytes, // buffer size (in bytes)
                LPD3DXBUFFER *ppBuffer); // ID3DXBuffer
```

  - E.g.,

```
// create a buffer to store 4 integers
ID3DXBuffer* buffer = 0;
D3DXCreateBuffer( 4*sizeof(int), &buffer );
```

XFiles

- The 3D modelers allow the use to build complex and realistic meshes in a visual and interactive environment
  - 3DS Max (www.discreet.com)
  - LightWave3D (www.newtek.com)
  - Maya (www.aliaswavefront.com)
  - Multigen Creator (www.multigen.com)
  - Soft Image (www.softimage.com)
- 3D Model Exporter
  - Okino Polytrans (www.okino.com)
  - Pandasoft
    - Plug-in tool to export a 3D Max object into .x file
    - http://www.andytather.co.uk/Panda/directxmax_downloads.aspx

Converting 3DS MAX to X File

- Converter
  - Run 'conv3ds.exe' in a command prompt to create a .x file
    - ~>conv3ds ExFile.3ds
  - Conv3ds options
    - http://telnet.or.kr/sec_directx/index.html?init_mode=api_contents_read&api_no=86
  - Refer to http://dis.dankook.ac.kr/lectures/game10/entry/XFiles

Load Xfiles

- Load the .X file data into an ID3DXMesh object using D3DXLoadMeshFromX

```
HRESULT D3DXLoadMeshFromX (
    LPCSTR pFilename,
    DWORD Options,
    LPDIRECT3DDEVICE9 pDevice,
    LPD3DXBUFFER* ppAdjacency,
    LPD3DXBUFFER* ppMaterials,
    LPD3DXBUFFER* ppEffectInstances,
    PDWORD pNumMaterials,
    LPD3DXMESH* ppMesh);
```

  - pFilename – the .X filename
  - Options – creation flags (D3DXMESH enum type)
  - ppAdjacency, ppMaterials, ppEffectInstaces, pNumMaterials, ppMesh – return parameters

Load XFiles

- E.g.,

```
HRESULT hr = 0;

ID3DXBuffer* adjBuffer  = 0;
ID3DXBuffer* mtrlBuffer = 0;
DWORD        numMtrls   = 0;

hr = D3DXLoadMeshFromX( "bigship1.x",
                        D3DXMESH_MANAGED,
                        Device,
                        &adjBuffer,
                        &mtrlBuffer,
                        0,
                        &numMtrls,  // # of D3DXMATERIAL structures
                                    // in mtrlBuffer
                        &Mesh );    // loaded ID3DXMesh
```

XFile Materials

- D3DXMATERIAL structure containing the material data

```
typedef struct _D3DMATERIAL9 {
    D3DCOLORVALUE Diffuse, Ambient, Specular, Emissive,
    float Power;
} D3DMATERIAL9;
typedef struct D3DXMATERIAL {
    D3DMATERIAL9 MatD3D;
    LPSTR pTextureFilename;
} D3DXMATERIAL, *LPD3DXMATERIAL;
```

  - D3DXMATERIAL contains the basic D3DMATERIAL9 structure and a pointer to a null terminating string that specifies the associative texture filename.
  - D3DXLoadMeshFromX loads the .X file data so that the ith entry in the returned D3DXMATERIAL array corresponds with the ith subset.

Example: XFile Demo

```
ID3DXMesh* Mesh = 0;
ID3DXBuffer* adjBuffer = 0;
ID3DXBuffer* mtrlBuffer = 0;
DWORD numMtrls = 0;
// 1. load the .x file from file into a system memory
HR(D3DXLoadMeshFromX("bigship1.x", D3DXMESH_SYSTEMMEM, Device, &adjBuffer,
    &mtrlBuffer, 0, &numMtrls, &Mesh));

//2. find out if the mesh already has normal info
D3DVERTEXELEMENT9 elems[MAX_FVF_DECL_SIZE];
HR(Mesh->GetDeclaration(elems));
bool hasNormals = false;
for (int i = 0; I < MAX_FVF_DECL_SIZE; ++i) {
    // did we reach D3DDECL_END
    if (elems[i].Stream == 0xff) break;
    if (elems[i].Type == D3DDECLTYPE_FLOAT3 &&
        elems[i].Usage == D3DDECLUSAGE_NORMAL &&
        elems[i].UsageIndex == 0) {
        hasNormals = true;
        break;
    }
}
```

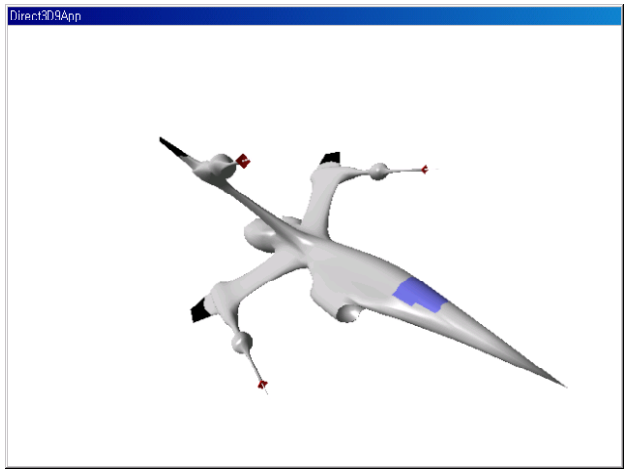
Example: XFile Demo

```
// 3. change vertex format to VertexPNT
D3DVERTEXELEMENT9 elements[64];
UINT numElements = 0;
VertexPNT::Decl->GetDeclaration(elements, &numElements);
ID3DXMesh* temp = 0;
HR(Mesh->CloneMesh(D3DXMESH_SYSTEMMEM, elements, Device, &temp));
ReleaseCom(Mesh);
Mesh = temp;

//4. generate normals
if  (hasNormals == false) HR(D3DXComputeNormals(Mesh, 0));

//5. Optimize the mesh
HR(Mesh->Optimize(D3DXMESH_MANAGED | D3DXMESHOPT_COMPACT |
                  D3DXMESHOPT_ATTRSORT | D3DXMESHOPT_VERTEXCACHE,
                  (DWORD*)adjBuffer->GetBufferPointer(), 0, 0, 0, MeshOut));
ReleaseCOM(Mesh);
ReleaseCOM(adjBuffer);
```

Example: XFile Demo

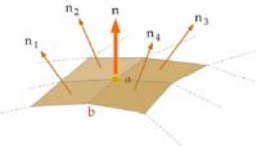
```
// 6. Extract the material and load the textures
if (mtrlBuffer != 0 && numMtrls != 0) {
    D3DXMATERIAL* d3dxmtrls = (D3DXMATERIAL*) mtrlBuffer->GetBufferPointer();
    for(DWORD i = 0; i < numMtrls; ++i) {
        // Save the ith material. Note that the MatD3D property
        // does not have an ambient value set when it's loaded, so
        // just set it to the diffuse value.
        Mtrl m;
        m.ambient = d3dxmtrls[i].MatD3D.Diffuse;
        m.diffuse = d3dxmtrls[i].MatD3D.Diffuse;
        m.spec = d3dxmtrls[i].MatD3D.Specular;
        m.specPower = d3dxmtrls[i].MatD3D.Power;
        mtrls.push_back( m );
        // Check if the ith material has an associative texture
        if( d3dxmtrls[i].pTextureFilename != 0 ) { // Yes, load the texture for the ith subset
            IDirect3DTexture9* tex = 0;
            char* texFN = d3dxmtrls[i].pTextureFilename;
            HR(D3DXCreateTextureFromFile(Device, texFN, &tex));
            // Save the loaded texture
            texs.push_back( tex );
        } else { // No texture for the ith subset texs.push_back( 0 ); } } }
    ReleaseCOM(mtrlBuffer); // done w/ buffer
```

Example: XFile



Creating Vertex Normals

- ID3DXBaseMesh vertex normal compute function



```
HRESULT D3DXComputeNormals (
    LPD3DXBASEMESH pMesh,
    CONST DWORD *pAdjacency);
```

  - pMesh – a pointer to the mesh containing NORMAL vertex format
  - pAdjacency – a pointer to the adjacency array (NULL if not used)

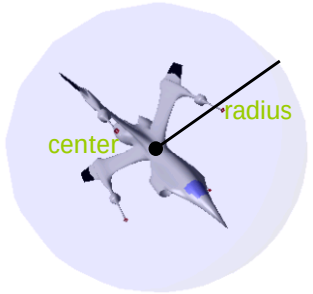
Creating Vertex Normals

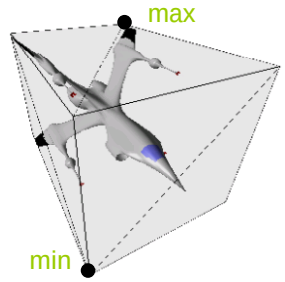
- Create vertex normals using D3DXComputeNormals
  - If Xfile doesn't have vertex normal, the corresponding ID3DXMesh's FVF wouldn't have D3DFVF_NORMAL flag.

```
if( !(pMesh->GetFVF() & D3DFVF_NORMAL) ) {
    ID3DXMesh* pTempMesh = 0;
    pMesh->CloneMeshFVF (
      D3DXMESH_MANAGED,
      pMesh->GetFVF() | D3DFVF_NORMAL,
      Device,
      &pTempMesh );
    D3DXComputeNormals( pTempMesh, 0 );
    pMesh->Release();
    pMesh = pTempMesh;
}
```

Bounding Volumes

- Two common example of bounding volumes:
  - Bounding Sphere: center, radius
  - Bounding Box: min, max
- Others – cylinder, ellipsoid, lozenge, capsule
- Uses – visibility test, intersection, collision test





Bounding Volumes

- D3DX library provides functions to calculate the bounding sphere of a mesh and the AABB of a mesh.

```
HRESULT D3DXComputeBoundingSphere (
    LPD3DXVECTOR3 pFirstPosition, // position element in first vertex
    DWORD NumVertices,
    DWORD dwStride, // vertex size (in bytes)
    D3DXVECTOR3* pCenter, // center of bounding sphere
    FLOAT* pRadius); // radius of bounding sphere

HRESULT D3DXComputeBoundingBox (
    LPD3DXVECTOR3 pFirstPosition,
    DWORD NumVertices,
    DWORD dwStride, // vertex size (in bytes)
    D3DXVECTOR3* pMin, // lower-left corner of bounding box
    D3DXVECTOR3* pMax); // upper-right corner of bounding box
```

Bounding Volumes – d3dUtility

- Some new special constants:

```
const float INFINITY = FLT_MAX; // max float
const float EPSILON = 0.001f; // number smaller than it equal to 0
```

- BoundingSphere/Box types

```
struct AABB{
    AABB() : minPt(INFINITY, INFINITY, INFINITY),
             maxPt(-INFINITY, -INFINITY, -INFINITY) {}
    D3DXVECTOR3 center() { return 0.5f * (minPt + maxPt); }
    D3DXVECTOR3 minPt;
    D3DXVECTOR3 maxPt;
};
struct BoundingSphere {
    BoundingSphere() : pos(0.0f, 0.0f, 0.0f), radius(0.0f) {}
    D3DXVECTOR3 pos;
    float radius;
};
```

Example: Bounding Volumes

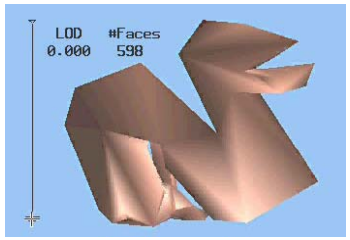
```
LoadXFile("skullocc.x", &mMesh, mMtrl, mTex);
D3DXMatrixIdentity(&mWorld);

// Compute the bounding box.
VertexPNT* v = 0;
HR(mMesh->LockVertexBuffer(0, (void**)&v));
HR(D3DXComputeBoundingBox(&v[0].pos, mMesh->GetNumVertices(), sizeof(VertexPNT),
    &mBoundingBox.minPt, &mBoundingBox.maxPt));
HR(mMesh->UnlockVertexBuffer());

float width = mBoundingBox.maxPt.x - mBoundingBox.minPt.x;
float height = mBoundingBox.maxPt.y - mBoundingBox.minPt.y;
float depth = mBoundingBox.maxPt.z - mBoundingBox.minPt.z;

// Build a box mesh so that we can render the bounding box visually.
HR(D3DXCreateBox(gd3dDevice, width, height, depth, &mBox, 0));

D3DXVECTOR3 center = mBoundingBox.center();
D3DXMatrixTranslation(&mBoundingBoxOffset, center.x, center.y, center.z);
// Define the box material--make semi-transparent.
mBoxMtrl.ambient = D3DXCOLOR(0.0f, 0.0f, 1.0f, 1.0f);
mBoxMtrl.diffuse = D3DXCOLOR(0.0f, 0.0f, 1.0f, 0.5f);
mBoxMtrl.spec = D3DXCOLOR(0.5f, 0.5f, 0.5f, 1.0f); mBoxMtrl.specPower = 8.0f;
```

Example: Bounding Volumes

```
// Draw the bounding box with alpha blending.
HR(gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, true));
HR(gd3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA));
HR(gd3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA));
HR(mFX->SetMatrix(mhWVP, &(mBoundingBoxOffset*mView*mProj)));

D3DXMatrixInverse(&worldInvTrans, 0, &mBoundingBoxOffset);
D3DXMatrixTranspose(&worldInvTrans, &worldInvTrans);

HR(mFX->SetMatrix(mhWorldInvTrans, &worldInvTrans));
HR(mFX->SetMatrix(mhWorld, &mBoundingBoxOffset));
HR(mFX->SetValue(mhMtrl, &mBoxMtrl, sizeof(Mtrl)));
HR(mFX->SetTexture(mhTex, mWhiteTex));
HR(mFX->CommitChanges());
HR(mBox->DrawSubset(0));
HR(gd3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, false));
```

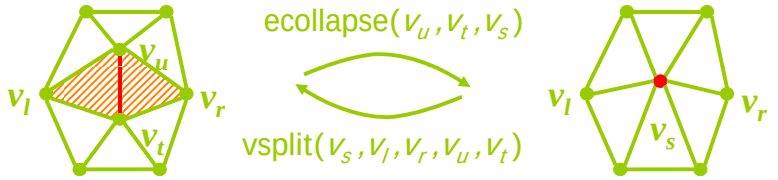
Progressive Mesh

- **ID3DXPMesh** interface
  - Simplify a mesh through ECT (Edge Collapse Transformation)
    - 1 ECT removes 1 vertex and 1~2 faces
    - Each ECT is reversible through vertex split

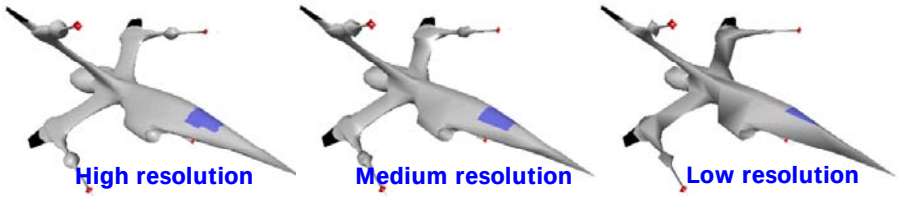


Progressive Mesh

- ECT vs. Vertex split



  - Go back to original mesh through ECT sequence



Creating Progressive Mesh

- Creating a progressive mesh object

```
HRESULT D3DXGeneratePMesh (
    LPD3DXMESH pMesh,
    CONST DWORD *pAdjacency,
    CONST LPD3DXATTRIBUTEWEIGHTS pVertexAttributeWeights,
    CONST FLOAT *pVertexWeights,
    DWORD MinValues,
    DWORD Options,
    LPD3DXPMESH *ppPMesh);
```

  - pVertexAttributeWeights – a pointer of D3DXATTRIBUTEWEIGHTS array (pMesh->GetNumVertices() size)
  - pVertexWeights – pMesh->GetNumVertices() float array
  - MinValue – minimal vertex/face #
  - Options – D3DXMESHSIMP_VERTEX, D3DXMESHSIMP_FACE, etc

Vertex Attribute Weights

- Vertex attribute weights structure

```
typedef struct _D3DXATTRIBUTEWEIGHTS {
    FLOAT Position;
    FLOAT Boundary;// blend weight
    FLOAT Normal;   // normal
    FLOAT Diffuse;   // diffuse light value
    FLOAT Specular;  // specular light value
    FLOAT Texcoord[8]; // texture coordinates
    FLOAT Tangent;
    FLOAT Binormal;
} D3DXATTRIBUTEWEIGHTS
```

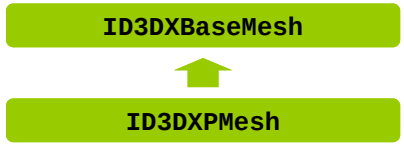
Vertex Attribute Weights

- Default vertex attribute weights:

```
D3DXATTRIBUTEWEIGHTS AttributeWeights;
AttributeWeights.Position     = 1.0;
AttributeWeights.Boundary     = 1.0;
AttributeWeights.Normal       = 1.0;
AttributeWeights.Diffuse      = 0.0;
AttributeWeights.Specular     = 0.0;
AttributeWeights.Texcoord[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };
```

ID3DXPMesh Methods

- ID3DXPMesh interface is also derived from ID3DXBaseMesh interface.



ID3DXPMesh Methods

- ID3DXPMesh methods:
  - DWORD ID3DXPMesh::GetMaxFaces(VOID);
  - DWORD ID3DXPMesh::GetMaxVertices(VOID);
  - DWORD ID3DXPMesh::GetMinFaces(VOID);
  - DWORD ID3DXPMesh::GetMinVertices(VOID);
  - HRESULT ID3DXPMesh::SetNumFaces(DWORD Faces);
    - Allows to adjust LOD by setting a new face count [min - max range]
  - HRESULT ID3DXPMesh::SetNumVertices(DWORD Vertices);
    - Allows to adjust LOD by setting a new vertex count [min - max range]

ID3DXPMesh Methods

- ID3DXBaseMesh로부터 상속받은 것 외에 추가된 method
  - HRESULT ID3DXPMesh::TrimByFaces(DWORD NewFacesMin, DWORD NewFacesMax, DWORD *rgiFaceRemap, DWORD *rgiVertRemap)
    - Specify new min/max faces [GetMinFaces() - GetMaxFaces() range]
  - HRESULT ID3DXPMesh::TrimByVertices(DWORD NewVerticesMin, DWORD NewVerticesMax, DWORD *rgiFaceRemap, DWORD *rgiVertRemap)
    - Specify new min/max vertices [GetMinVertices() - GetMaxVertices() range]