

Interface ID3DXMesh from D3DX library

Hello, I am proud to present you another tutorial about DirectX.

Today we will „play” with Direct3D Meshes. It's not a description about base application, device initializing, lights etc. so I will focus only on meshes. This tutorial is rather old (using DirectX8) because I wrote it few years ago in polish language and it's only the translation. If you want to use it in DirectX9 environment it is not a problem. The idea is the same – only function parameters are little bit different in some places. Look to the documentation.

After this tutorial you will be able to render something better than simple cube :) You will render your first 3D model. To make it more interesting it will be textured. Ofcourse, you cannot compare .x files it to own model format but for the begining it's more than enough.

I can see faces of many programmers saying that this is waste of time because in today's games it is not used. OK, easy – for simple games DX Meshes are quite good. If we don't need animations, it is a good choice. Except own model format, ofcourse ;) If you want you can read more in tutorials about Milkshape and 3DStudio tutorials.

Let's back to the Mesh. We need some pointers:

```
LPD3DXMESH g_pMesh;
D3DMATERIAL8 *g_pMeshMaterials;
LPDIRECT3DTEXTURE8 *g_pMeshTextures;
DWORD g_dwNumMaterials = 0L;
LPD3DXBUFFER pD3DXMtrlBuffer;
```

It is quite much but everything is simple and easy to understand.

- g_pMesh is nothing more but our Mesh object.
- g_pMeshMaterials is a pointer on material. Material describes how the light is being reflected from model surface.
- g_pMeshTextures is a pointer on textures. I hope you know what is a texture ;)
- g_dwNumMaterials is a number of materials used in the mesh because each mesh can have more than one material.
- pD3DXMtrlBuffer is data buffer. You can store anything you want there and we will use it for materials.

OK, now we must answer a simple question – what we are going to render today ?

In fact, answer is not important – the solution is always the same. You can create a creature, for example in Milkshape or any other tool and export it to the format DirectX can recognize. I talk about „.X” files. Cool files that have .x extension in the name :)

To not waste the time, I've taken a tiger from DirectX SDK. There is a texture too so we have got „zero” work with modelling and texturing. We are programmers – not graphicicians :)

Let's put model and texture file in our program directory and load them. We need only one function but it not as simple as it seems. This is the function:

```
D3DXLoadMeshFromX( "Tiger.x", D3DXMESH_SYSTEMMEM,
                  g_pd3dDevice, NULL,
                  &pD3DXMtrlBuffer, &g_dwNumMaterials,
                  &g_pMesh )
```

Many parameters but let's try to understand them. First is a path to our .x file. We have got it in application directory so it's only the file name. Second is a flag describing creation process. In our case it's D3DXMESH_SYSTEMMEM, which means that DirectX should put index and vertex buffers in system memory. You can read more about it in SDK. Third parameter is our device, fourth is adjacency buffer but we don't need this data so put NULL there. Next parameter is filled

with D3DXMATERIAL structures.

This structure looks like below:

```
struct D3DXMATERIAL {
    D3DMATERIAL8  MatD3D; //material
    LPSTR        pTextureFilename; //texture file
};
```

I hope it's not necessary to explain it.

Sixth param is a pointer on material number The last one is a pointer on a mesh object. The most important pointer in our application..

Now we must extract everything what is in material buffer:

```
D3DXMATERIAL *d3dxMaterials = (D3DXMATERIAL*)pD3DXMtrlBuffer->GetBufferPointer();
g_pMeshMaterials = new D3DMATERIAL8[g_dwNumMaterials];
g_pMeshTextures  = new LPDIRECT3DTEXTURE8[g_dwNumMaterials];
```

d3dxMaterials is a pointer on structure what you've seen above. We are initializing it by the beginig of material buffer. Ofcourse in the buffer can be anything so it's necessary to make proper cast. g_pMeshMaterials and g_pMeshTextures are known pointers. We must create as many materials and textures as g_dwNumMaterials.

Write some data to the material array::

```
for( DWORD i=0; i < g_dwNumMaterials; i++ )
{
    // we copy the material
    g_pMeshMaterials[i] = d3dxMaterials[i].MatD3D;

    // additional properties
    g_pMeshMaterials[i].Ambient = g_pMeshMaterials[i].Diffuse;

    // creating texture
    if( FAILED( D3DXCreateTextureFromFile( g_pd3dDevice,
d3dxMaterials[i].pTextureFilename, &g_pMeshTextures[i] ) ) )
    {
        g_pMeshTextures[i] = NULL;
    }
}
```

We make the loop and go through it for g_dwNumMaterials times. Copy materials from buffer to an array. Let's stop for a while here and explain one thing:

```
g_pMeshMaterials[i].Ambient = g_pMeshMaterials[i].Diffuse;
```

I must describe it because somebody can say „what the hell are materials and their properties ?”

We have many kinds of lights:

Ambient – it's the light which surrounds us. We cannot say where exactly is a source of this light. It comes from everywhere.

Diffuse – it's rays are coming from one direction and reflect from the surface to all directions. Intensity of this light depends on the reflection angle.

Specular – rays of this light also comes from one point but reflect only in one direction. Something like car reflector.

Material, like the light is described by these three factors (four in fact because material can produce it's own light – „emissive” factor). These factors describe the ability of light reflecting.

Properties of D3DMATERIAL8 – our Ambient, Diffuse, Specular, Emissive are variables of D3DCOLORVALUE type. In other words r - red, g - green, b - blue, a – alpha channel

Making it like below:

```
Material.Diffuse.r=1.0f
```

we tell DirectX that our material will be reflecting all diffuse light (in this case - for red colour). If we put there 0.0f, the material will reflect nothing. The same is with green and blue components and also with different lights.

I hope you understood but if not „play” with it as long as you need.

Ok, another operation – creating of texture.

D3DXCreateTextureFromFile(...)

First parameter is our device. Second is a path to the texture and third is a pointer on a texture object. If operation fails this pointer gets NULL.

Material buffer is not longer needed so release it:

```
pD3DXMtrlBuffer->Release();
```

Well, we've got everything now. The rendering is very simple:

```
for( DWORD i=0; i<dwNumMaterials; i++);
{
    g_pd3dDevice->SetMaterial( &g_pMeshMaterials[i] );
    g_pd3dDevice->SetTexture( 0, g_pMeshTextures[i] );

    // drawing
    g_pMesh->DrawSubset( i );
}
```

This piece of code must be placed between BeginScene() and EndScene() methods of device interface. SetMaterial method sets material for the group of polygons. SetTexture sets texture and it has two parameters. First is a texture slot (we can have up to 8 slots from 0 to 7) . Second is a pointer on texture object.

Next is only drawing. But you may ask why we need to do it this way. Let's see:

Our model can have one or more subsets. If there is more than one material we can be sure that our mesh is divided into subsets. One subset for one material. For example a motocyclist. It is obvious that his boots will not shine like his helmet so we need two groups of polygons with different materials. These are subsets and we need to set one material then render first subset. Next pass of the loop will change the material and render next group of polygons. As a result we will have properly rendered model. That's why we need a loop.

As usual we need to „clean” our workplace:

```
if( g_pMeshMaterials != NULL )
    delete[] g_pMeshMaterials;

    if( g_pMeshTextures )
    {
        for( DWORD i = 0; i < g_dwNumMaterials; i++ )
        {
            if( g_pMeshTextures[i] )
                g_pMeshTextures[i]->Release();
        }
        delete[] g_pMeshTextures;
    }
    if( g_pMesh != NULL )
        g_pMesh->Release();
```

As I said before – this tutorial only describes meshes. You must put it into 3D space by yourself. You can add lights, effects... whatever.

If you have any questions or suggestions feel free to email me.

Have a nice coding.

SirMike