Using Effect files (.FX) in DirectX 9

by Mauro Gentile

Introduction

This is a simple article on how to load effect files and how to manage them and create a simle class to use them in an easy way. I am using DirectX 9 (with the D3DX utility library) and NVIDIA FX files. This code is for those programmers who might be interested in loading NVIDIA effects (or any other DXSAS effect file).

DXSAS is the DirectX Standard Annotations and Semantics notation for .FX files.

Why to use .FX effect files and why is DXSAS so important? Well, first of all, .FX effect files are so comfortable that using Cg or other shaders languages will be so difficult that you'll discard that in a moment. Effect files, in add, allow, for example, the programmer to specify different techniques dependant on the capabilities of various platform. DXSAS is so easy to understand and to implement and to load that it'll be so useful. It let to the programmer to get a lot of information directly form .FX file without any hard work. Rendering techniques could be a "scripting" separated part of the the graphics engine. I'll write down a sample to load semantics from .FX files and applying a generic effect on a mesh.

Basics

An effect file is a simple text file. Its only difference is its .FX extension. It is split into three main parts:

1. Variable declarations - these are values that can be set before or during rendering [textures, matrces, lights, ...]
2. Techniques & Passes- defines how something is to be rendered. [sampler or texture stages information and vertex and shader declaration]
3. Functions - the shader code [HLSL coded]

Parameters are defined in the effect file. You specify a type and a name and then you could use them easily in your engine.

Techniques are simply containers for Direct3D stages and every Technique could have one ore more Pass in which to specify them.

Shader code could be written in HLSL or in Assembly. It is compiled at loading time and then used easily.

Effect files could also been included directly in .X files. But this is another story!

The File-Dependent Code

Using D3DX to load and manage an effect is very easy. I will implement NVDIA Flame.FX effect file on a simple quad mesh (saved in an .X file). The following code is the necessary to start:

```
D3DXMESHCONTAINER_EX * g_Mesh;      // This contain quad .X file
IDirect3DVertexDeclaration9 *g_VertDecl = NULL;         //Vertex declaration
ID3DXEffect* g_pEffect;             // This contain Effect File information
LPD3DXBUFFER pBufferErrors;         // This is a buffer containing errors during effect creation
D3DXEFFECT_DESC pEffectDesc;        // This variable will be used to get out effect description
DWORD dwShaderFlags;                // These are shader flags at creation time
DWORD iPass;
UINT cPasses;                       // These are two variables we'll use at render time
```

To manage parameters you need to declare some D3DXHANDLE that will contain parameters information. If you want to load a parameter and you know its name you can simply load it in this way:

```
g_pEffect->SetParameterType( 'parameterName', ... );   // 'parameterName' is the name of the parameter
```

Suppose you want to set a float parameter named 'noiseFreq' with '0.3f' value, a vector parameter named 'noiseAnim' with { 0.2, -0.4, 0.2 } and a texture. You could do it in this simple way:

```
g_pEffect->SetFloat( "noiseFreq", 0.3f );   // 'noiseFreq' is the name of the parameter
g_pEffect->SetValue( "noiseAnim", D3DXVECTOR3(0.2, -0.4, 0.2), sizeof(D3DXVECTOR3) );     // 'noiseAnim' is the name of
the parameter

D3DXCreateTextureFromFile(g_pD3DDevice, PathFlameTexture , &g_FlameTexture );    //Loading texture file
g_pEffect->SetTexture( "noiseTexture", g_FlameTexture );                         //Setting 'noiseTexture' parameter
with created texture
```

Now is important to set the Technique you want to use:

```
g_pEffect->SetTechnique("TechniqueName");      // 'TechniqueName' is the name of the technique to use
```

Every effect need some vertex information. You can get it out directly from mesh's vertices. Generally vertex declaration is a this following one:

```
  // Vertex shader declaration
D3DVERTEXELEMENT9 vertexElement[] =                                                 // ARRAY of vertex information
{
{ 0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0 },     //Position information
{ 0, 12, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL, 0 },      //Normal information
{ 0, 24, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0 },    //Texture coordinates information
{ 0, 36, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TANGENT, 0 },     //Tangent information
{ 0, 48, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_BINORMAL, 0 },    //Binormal information
{ 0, 60, D3DDECLTYPE_D3DCOLOR, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_COLOR, 0 },     //Color information
D3DDECL_END()
};

g_pD3DDevice->CreateVertexDeclaration( vertexElement, &g_VertDecl );    //We are creating here vertex declaration
g_pD3DDevice->SetVertexDeclaration(g_VertDecl);      // 'g_VertDecl' is the vertex declaration needed by the effect
```

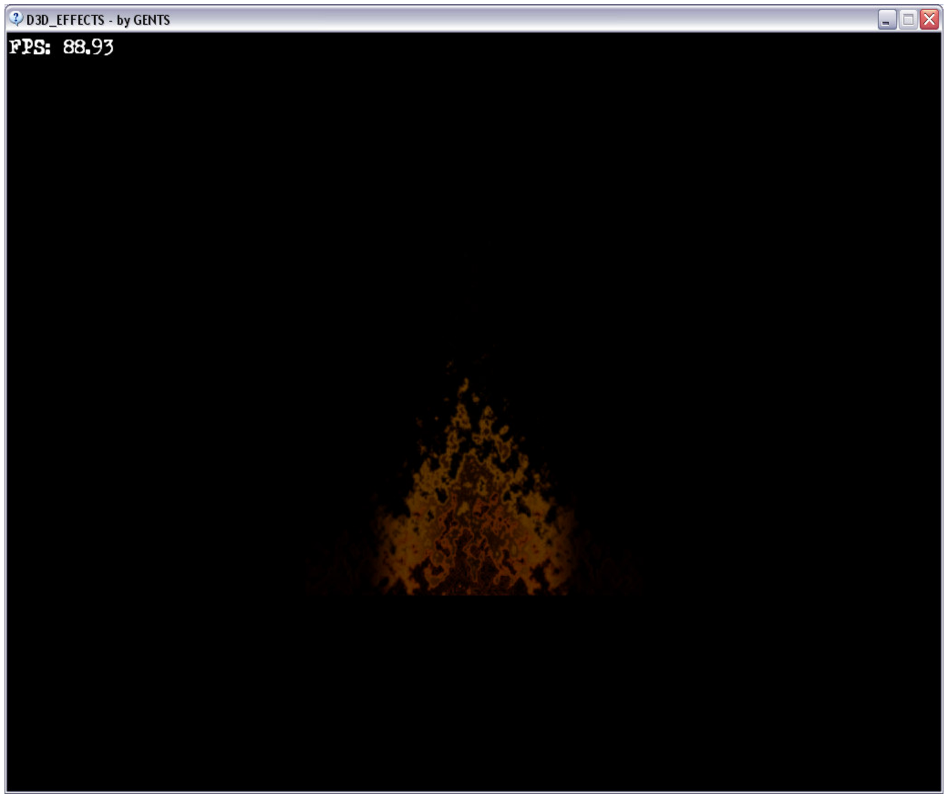
You have to notice that there are some parameters that need to be setted only at creation time and other parameters that change every frame and you've to set them at render time. You can understand that easily. Parameters as matrix, time or position need to change during rendering.

At this time you can set parametes that are changed, render effect file and then mesh with effect applied on it:

```
  // Begin 3D scene
  g_pD3DDevice->BeginScene();

        //set a time parameter that needs ticks count
        g_pEffect->SetFloat( "ticks", (float)GetTickCount()/1000.0f );
        //set a matrix parameter that need World*View*Projection as value
        g_pEffect->SetMatrix( "wvp", &mWorldViewProjection );
        //set a parameter that need world matrix
        g_pEffect->SetMatrix( "world", &mWorld );

        //STARTING TO RENDER EFFECT
        if( SUCCEEDED( g_pEffect->Begin(&cPasses, 0) ) ){
               //Iterate for each passof Technique chosen
               for (iPass = 0; iPass < cPasses; iPass++)
               {
                   g_pEffect->BeginPass(iPass);
                   // Render simply the mesh
                   g_Mesh->MeshData.pMesh->DrawSubset(0);
```

```
                    g_pEffect->EndPass();
                }
                g_pEffect->End();
        }

  // End 3D scene
  g_pD3DDevice->EndScene();
```

You've now finished to rendering and you could see you effect on the screen.



The File-Independent Code

The following code is the necessary to start:

```
D3DXMESHCONTAINER_EX * g_Mesh;     // This contain quad .X file
ID3DXEffect* g_pEffect;            // This contain Effect File information
LPD3DXBUFFER pBufferErrors;        // This is a buffer containing errors during effect creation
D3DXEFFECT_DESC pEffectDesc;       // This variable will be used to get out effect description
DWORD dwShaderFlags;               // These are shader flags at creation time
DWORD iPass;
UINT cPasses;                      // These are two variables we'll use at render time
```

To manage DXSAS information is useful to declare some D3DXHANDLE that will contain parameters information. In this moment is important to notice that there are a lot of parameter in the Flame.FX file that doesn't use DXSAS notation. These parameter will be loaded with .FX files values and their values could be modified simply changing .FX file values manually or declaring a D3DXHANDLE to manage them.

If you want to implement something that is not File-dependent you have to think that you don't know parameters name, so reading DXSAS Microsoft Reference you understand that you've to declare D3DXHANDLE for every parameter that use DXSAS notation doing as it:

```
//DXSAS handles
D3DXHANDLE ambient, attenuation, bbmax, bbmin, bbsize, bcenter, bssize, bsmin, bsmax, diffuse,
                       elapsed, emissive, envNormal, height, joint, jointW, jointWI, jointWIT, jointWV,
                       jointWVI, jointWVIT, jointWVP, jointWVPI, jointWVPIT, last, normal, opacity,
                       position, proj, projI, projIT, random, refraction, renderCT, renderDST, renderTC,
                       renderTD, specular, specularP, standarGlob, TextureMat, time, UnitsScale, view,
                       viewI, viewIT, viewP, viewPI, viewPIT, world, worldI, worldIT, worldV, worldVI,
                       worldVIT, worldVP, worldVPI, worldVPIT;
```

For every handle you need to create a variable to save information:

```
//DXSAS variables
D3DXVECTOR4 amb4, att4, diff, emis, join, nor2, opa4, posit, ref4, rtc, spec, specP4;
D3DXVECTOR3 att3, bbMax, bbMin, bbSiz, bCen, nor1, opa3, ref3, specP3;
D3DXVECTOR2 opa2, ref2, rtd, specP2;
float bsSiz, bsMin, bsMax, elapTime, heigtMap1, lasTime, opa1, ran, ref1, specP1, tim, unit;
LPDIRECT3DTEXTURE9 envNorm, heightMapT, nor3, opa5, ref5, rct, rdst, stdG;
```

```
D3DXMATRIX      jWor, jWI, jWIT, jWV, jWVI, jWVIT, jWVP, jWVPI, jWVPIT, pro, proI, proIT, texM,
                       vie, vieI, vieIT, vieP, viePI, viePIT, wor, worI, worIT, worV, worVI, worVIT, worVP, worVPI,
worVPIT;
```

Now you can proceed in this way :

```
void Effect::LoadEffectParams()
{

 //GET PARAMETERS HANDLE
  ambient = g_pEffect->GetParameterBySemantic( NULL, "Ambient" );
  attenuation = g_pEffect->GetParameterBySemantic( NULL, "Attenuation" );
  bbmax = g_pEffect->GetParameterBySemantic( NULL, "BoundingBoxMax" );
  bbmin = g_pEffect->GetParameterBySemantic( NULL, "BoundingBoxMin" );
  bbsize = g_pEffect->GetParameterBySemantic( NULL, "BoundingBoxSize" );
  bcenter = g_pEffect->GetParameterBySemantic( NULL, "BoundingCenter" );
  bssize = g_pEffect->GetParameterBySemantic( NULL, "BoundingSphereSize" );
  bsmin = g_pEffect->GetParameterBySemantic( NULL, "BoundingSphereMin" );
  bsmax = g_pEffect->GetParameterBySemantic( NULL, "BoundingSphereMax" );
  diffuse = g_pEffect->GetParameterBySemantic( NULL, "Diffuse" );
  elapsed = g_pEffect->GetParameterBySemantic( NULL, "ElapsedTime" );
  emissive = g_pEffect->GetParameterBySemantic( NULL, "Emissive" );
  envNormal = g_pEffect->GetParameterBySemantic( NULL, "EnviromentNormal" );
  height = g_pEffect->GetParameterBySemantic( NULL, "Height" );
  joint = g_pEffect->GetParameterBySemantic( NULL, "Joint" );
  jointW = g_pEffect->GetParameterBySemantic( NULL, "JointWorld" );
  jointWI = g_pEffect->GetParameterBySemantic( NULL, "JointWorldInverse" );
  jointWIT = g_pEffect->GetParameterBySemantic( NULL, "JointWorldInverseTranspose" );
  jointWV = g_pEffect->GetParameterBySemantic( NULL, "JointWorldView" );
  jointWVI = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewInverse" );
```

```
jointWVIT = g_pEffect->GetParameterBySemantic( NULL, "JointWolrdViewInverseTranspose" );
jointWVP = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewProjection" );
jointWVPI = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewProjectionInverse" );
jointWVPIT = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewProjectionTranspose" );
last = g_pEffect->GetParameterBySemantic( NULL, "LastTime" );
normal = g_pEffect->GetParameterBySemantic( NULL, "Normal" );
opacity = g_pEffect->GetParameterBySemantic( NULL, "Opacity" );
position = g_pEffect->GetParameterBySemantic( NULL, "Position" );
proj = g_pEffect->GetParameterBySemantic( NULL, "Projection" );
projI = g_pEffect->GetParameterBySemantic( NULL, "ProjectionInverse" );
projIT = g_pEffect->GetParameterBySemantic( NULL, "ProjectionInverseTranspose" );
random = g_pEffect->GetParameterBySemantic( NULL, "Random" );
refraction = g_pEffect->GetParameterBySemantic( NULL, "Refraction" );
renderCT = g_pEffect->GetParameterBySemantic( NULL, "RenderColorTarget" );
renderDST = g_pEffect->GetParameterBySemantic( NULL, "RenderDepthStencilTarget" );
renderTC = g_pEffect->GetParameterBySemantic( NULL, "RenderTargetClipping" );
renderTD = g_pEffect->GetParameterBySemantic( NULL, "RenderTargetDimension" );
specular = g_pEffect->GetParameterBySemantic( NULL, "Specular" );
specularP = g_pEffect->GetParameterBySemantic( NULL, "SpecularPower" );
standarGlob = g_pEffect->GetParameterBySemantic( NULL, "StandardGlobal" );
TextureMat = g_pEffect->GetParameterBySemantic( NULL, "TextureMatrix" );
time = g_pEffect->GetParameterBySemantic( NULL, "Time" );
UnitsScale = g_pEffect->GetParameterBySemantic( NULL, "UnitsScale" );
view = g_pEffect->GetParameterBySemantic( NULL, "View" );
viewI = g_pEffect->GetParameterBySemantic( NULL, "ViewInverse" );
viewIT = g_pEffect->GetParameterBySemantic( NULL, "ViewInverseTranspose" );
viewP = g_pEffect->GetParameterBySemantic( NULL, "ViewProjection" );
viewPI = g_pEffect->GetParameterBySemantic( NULL, "ViewProjectionInverse" );
```

```
viewPIT = g_pEffect->GetParameterBySemantic( NULL, "ViewProjectionInverseTranspose" );
world = g_pEffect->GetParameterBySemantic( NULL, "World" );
worldI = g_pEffect->GetParameterBySemantic( NULL, "WorldInverse" );
worldIT = g_pEffect->GetParameterBySemantic( NULL, "WorldInverseTranspose" );
worldV = g_pEffect->GetParameterBySemantic( NULL, "WorldView" );
worldVI = g_pEffect->GetParameterBySemantic( NULL, "WorldViewInverse" );
worldVIT = g_pEffect->GetParameterBySemantic( NULL, "WorldViewInverseTranspose" );
worldVP = g_pEffect->GetParameterBySemantic( NULL, "WorldViewProjection" );
worldVPI = g_pEffect->GetParameterBySemantic( NULL, "WorldViewProjectionInverse" );
worldVPIT = g_pEffect->GetParameterBySemantic( NULL, "WorldViewProjectionInverseTranspose" );

//get effect description
g_pEffect->GetDesc( &pEffectDesc );


//check if there is some texture to load iterating through all parameters and checking if there is some
//parameter that has a "texture description"
for( int index=0; index<pEffectDesc.Parameters; index++ )
{
    D3DXHANDLE hParam = g_pEffect->GetParameter( NULL, index );
    D3DXPARAMETER_DESC pDesc;

    // get parameter description
    g_pEffect->GetParameterDesc( hParam, &pDesc );

    // check if it is a Volume Texture and then load and set it
    if((pDesc.Type == D3DXPT_TEXTURE3D)){
        LPDIRECT3DVOLUMETEXTURE9 value;
        D3DXHANDLE hAnnot = g_pEffect->GetAnnotationByName( hParam, "ResourceName" );
        const char* Path;
        std::ostringstream oss, debug;
        g_pEffect->GetString( hAnnot, &Path );
```

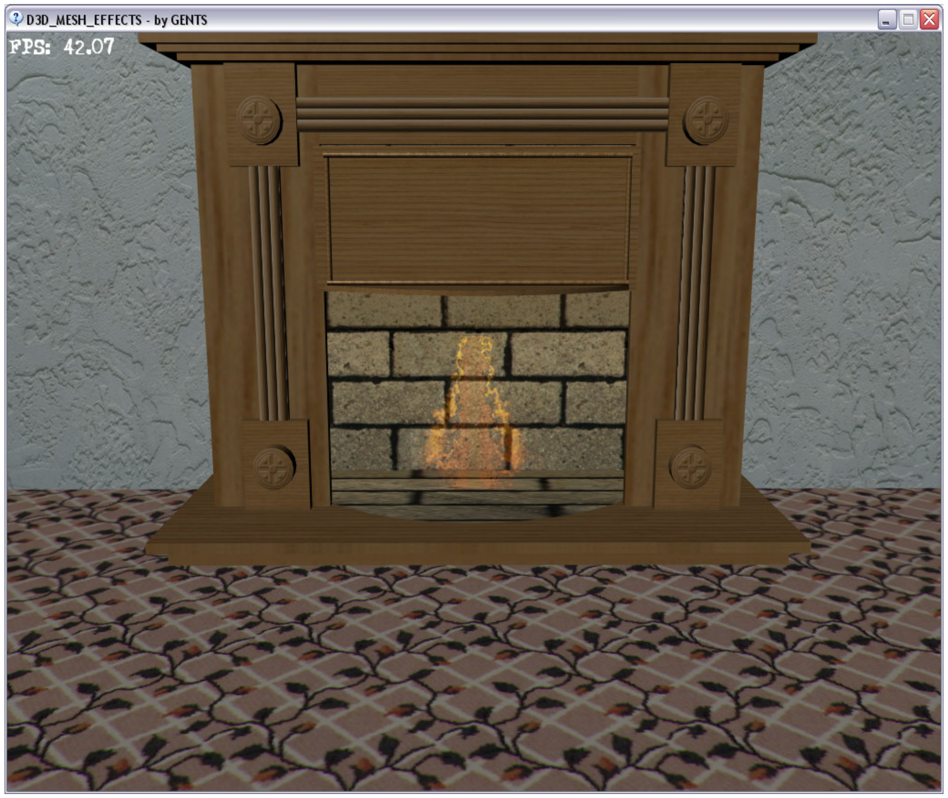
```
          oss << "..\\Data\\Textures\\" << Path;
          debug << "VolumeTexture PATH: " << oss;
          OutputDebugString( debug.str().c_str() );
          OutputDebugString( "\n" );
          if( FAILED( D3DXCreateVolumeTextureFromFileEx(g_pD3DDevice, oss.str().c_str(), D3DX_DEFAULT, D3DX_DEFAULT,
D3DX_DEFAULT, D3DX_DEFAULT,  NULL, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_DEFAULT, D3DX_DEFAULT, 0xff000000, NULL, NULL,
&value) ) )
                  SetError("Impossibile caricare la VolumeTexture");
          g_pEffect->SetTexture( pDesc.Name, value );
      }

      // check if it is a simple Texture and then load and set it
      else if((pDesc.Type == D3DXPT_TEXTURE)||(pDesc.Type == D3DXPT_TEXTURE1D)||(pDesc.Type == D3DXPT_TEXTURE2D)){
          LPDIRECT3DTEXTURE9 value;
          D3DXHANDLE hAnnot = g_pEffect->GetAnnotationByName( hParam, "ResourceName" );
          const char* Path;
          std::ostringstream oss, debug;
          g_pEffect->GetString( hAnnot, &Path );
          oss << "..\\Data\\Textures\\" << Path;
          debug << "Texture PATH: " << oss;
          OutputDebugString( debug.str().c_str() );
          OutputDebugString( "\n" );
          if( FAILED( D3DXCreateTextureFromFileEx(g_pD3DDevice, oss.str().c_str(), D3DX_DEFAULT, D3DX_DEFAULT,
D3DX_DEFAULT, NULL, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_DEFAULT, D3DX_DEFAULT, 0xff000000, NULL, NULL, &value) ) )
                  SetError("Impossibile caricare la texture");
          g_pEffect->SetTexture( pDesc.Name, value );
      }
   }



   // Apply the technique contained in the effect
   D3DXHANDLE hTech;
   g_pEffect->FindNextValidTechnique( NULL, &hTech );
   g_pEffect->SetTechnique( hTech );
 }
```

Now you are quite ready to render, but you have to set also changing parameters as in this sample code (I've not implemented all variables update here):

```
void Effect::SetEffectParams()
{
        //GET PARAMETERS VALUES
        g_pD3DDevice->GetTransform( D3DTS_WORLD, &wor );
   g_pD3DDevice->GetTransform( D3DTS_PROJECTION, &pro );
        g_pD3DDevice->GetTransform( D3DTS_VIEW, &vie );

        D3DXMatrixInverse( &proI, NULL, &pro );
        D3DXMatrixTranspose( &proIT, &proI );
        D3DXMatrixInverse( &vieI, NULL, &vie );
        D3DXMatrixTranspose( &vieIT, &vieI );
        vieP = vie * pro;
        D3DXMatrixInverse( &viePI, NULL, &vieP );
        D3DXMatrixTranspose( &viePIT, &viePI );
        D3DXMatrixInverse( &worI, NULL, &wor );
        D3DXMatrixTranspose( &worIT, &worI );
        worV = wor * vie;
        D3DXMatrixInverse( &worVI, NULL, &worV );
        D3DXMatrixTranspose( &worVIT, &worVI );
        worVP= wor * vie * pro;
        D3DXMatrixInverse( &worVPI, NULL, &worVP );
        D3DXMatrixTranspose( &worVPIT, &worVPI );
   tim = (float)GetTickCount()/1000.0f;

  //AND THEN SET PARAMETERS VALUES TO THE EFFECT

  g_pEffect->SetFloat( time, tim );

  g_pEffect->SetMatrix( proj, &pro );

  g_pEffect->SetMatrix( projI, &proI );

  g_pEffect->SetMatrix( projIT, &proIT );

  g_pEffect->SetMatrix( view, &vie );

  g_pEffect->SetMatrix( viewI, &vieI );

  g_pEffect->SetMatrix( viewIT, &vieIT );
  g_pEffect->SetMatrix( viewP, &vieP );

  g_pEffect->SetMatrix( viewPI, &viePI );


  g_pEffect->SetMatrix( viewPIT, &viePIT );


  g_pEffect->SetMatrix( world, &wor );
  g_pEffect->SetMatrix( worldI, &worI );

  g_pEffect->SetMatrix( worldIT, &worIT );

  g_pEffect->SetMatrix( worldV, &worV );

  g_pEffect->SetMatrix( worldVI, &worVI );

  g_pEffect->SetMatrix( worldVIT, &worVIT );
  g_pEffect->SetMatrix( worldVP, &worVP );

  g_pEffect->SetMatrix( worldVPI, &worVPI );

  g_pEffect->SetMatrix( worldVPIT, &worVPIT );

}
```

You could now proceed to render as the usual way :

```
  // Render the mesh with the applied technique
  if( FAILED( g_pD3DDevice->SetVertexDeclaration(g_Mesh->g_VertDecl) ) )
              SetError("Error in SetVertexDeclaration()");

                     SetEffectParams();

                     cPasses = 0;

                     if( SUCCEEDED( g_pEffect->Begin(&cPasses, 0) ) ){
                            for (iPass = 0; iPass < cPasses; iPass++)
                            {
                                g_pEffect->BeginPass(iPass);

                                   g_MeshOcean->MeshData.pMesh->DrawSubset(0);

                                g_pEffect->EndPass();
                            }
                            g_pEffect->End();
                     }

  }
```



Effect class

I've implemented a class that would help a lot with all this code:

```
class Effect{

public:

ID3DXEffect* g_pEffect;
```

```
LPD3DXBUFFER pBufferErrors;
D3DXEFFECT_DESC pEffectDesc;

DWORD dwShaderFlags;
DWORD iPass;
UINT cPasses;

CObjCollection g_Object;
D3DXMESHCONTAINER_EX * g_MeshObject;

protected:

//DXSAS handles
D3DXHANDLE ambient, attenuation, bbmax, bbmin, bbsize, bcenter, bssize, bsmin, bsmax, diffuse,
                       elapsed, emissive, envNormal, height, joint, jointW, jointWI, jointWIT, jointWV,
                       jointWVI, jointWVIT, jointWVP, jointWVPI, jointWVPIT, last, normal, opacity,
                       position, proj, projI, projIT, random, refraction, renderCT, renderDST, renderTC,
                       renderTD, specular, specularP, standarGlob, TextureMat, time, UnitsScale, view,
                       viewI, viewIT, viewP, viewPI, viewPIT, world, worldI, worldIT, worldV, worldVI,
                       worldVIT, worldVP, worldVPI, worldVPIT;
//DXSAS variables
D3DXVECTOR4 amb4, att4, diff, emis, join, nor2, opa4, posit, ref4, rtc, spec, specP4;
D3DXVECTOR3 att3, bbMax, bbMin, bbSiz, bCen, nor1, opa3, ref3, specP3;
D3DXVECTOR2 opa2, ref2, rtd, specP2;
float bsSiz, bsMin, bsMax, elapTime, heigtMap1, lasTime, opa1, ran, ref1, specP1, tim, unit;
LPDIRECT3DTEXTURE9 envNorm, heightMapT, nor3, opa5, ref5, rct, rdst, stdG;
D3DXMATRIX     jWor, jWI, jWIT, jWV, jWVI, jWVIT, jWVP, jWVPI, jWVPIT, pro, proI, proIT, texM,
                       vie, vieI, vieIT, vieP, viePI, viePIT, wor, worI, worIT, worV, worVI, worVIT, worVP, worVPI,
worVPIT;

public:
        Effect(){
                g_pEffect = NULL;
                pBufferErrors = NULL;
                dwShaderFlags = 0;
                iPass = 0;
                cPasses = 0;
        }
        ~Effect(){
                g_pEffect = NULL;
                pBufferErrors = NULL;
                dwShaderFlags = 0;
                iPass = 0;
                cPasses = 0;
        }

        void LoadEffectParams();
        void SetEffectParams();
        void Render();

};







void Effect::LoadEffectParams()
{

        //GET PARAMETERS HANDLE
        ambient = g_pEffect->GetParameterBySemantic( NULL, "Ambient" );
        attenuation = g_pEffect->GetParameterBySemantic( NULL, "Attenuation" );
        bbmax = g_pEffect->GetParameterBySemantic( NULL, "BoundingBoxMax" );
        bbmin = g_pEffect->GetParameterBySemantic( NULL, "BoundingBoxMin" );
        bbsize = g_pEffect->GetParameterBySemantic( NULL, "BoundingBoxSize" );
        bcenter = g_pEffect->GetParameterBySemantic( NULL, "BoundingCenter" );
        bssize = g_pEffect->GetParameterBySemantic( NULL, "BoundingSphereSize" );
        bsmin = g_pEffect->GetParameterBySemantic( NULL, "BoundingSphereMin" );
        bsmax = g_pEffect->GetParameterBySemantic( NULL, "BoundingSphereMax" );
        diffuse = g_pEffect->GetParameterBySemantic( NULL, "Diffuse" );
        elapsed = g_pEffect->GetParameterBySemantic( NULL, "ElapsedTime" );
        emissive = g_pEffect->GetParameterBySemantic( NULL, "Emissive" );
        envNormal = g_pEffect->GetParameterBySemantic( NULL, "EnviromentNormal" );
        height = g_pEffect->GetParameterBySemantic( NULL, "Height" );
        joint = g_pEffect->GetParameterBySemantic( NULL, "Joint" );
        jointW = g_pEffect->GetParameterBySemantic( NULL, "JointWorld" );
        jointWI = g_pEffect->GetParameterBySemantic( NULL, "JointWorldInverse" );
        jointWIT = g_pEffect->GetParameterBySemantic( NULL, "JointWorldInverseTranspose" );
        jointWV = g_pEffect->GetParameterBySemantic( NULL, "JointWorldView" );
        jointWVI = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewInverse" );
        jointWVIT = g_pEffect->GetParameterBySemantic( NULL, "JointWolrdViewInverseTranspose" );
        jointWVP = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewProjection" );
```

```
jointWVPI = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewProjectionInverse" );
jointWVPIT = g_pEffect->GetParameterBySemantic( NULL, "JointWorldViewProjectionTranspose" );
last = g_pEffect->GetParameterBySemantic( NULL, "LastTime" );
normal = g_pEffect->GetParameterBySemantic( NULL, "Normal" );
opacity = g_pEffect->GetParameterBySemantic( NULL, "Opacity" );
position = g_pEffect->GetParameterBySemantic( NULL, "Position" );
proj = g_pEffect->GetParameterBySemantic( NULL, "Projection" );
projI = g_pEffect->GetParameterBySemantic( NULL, "ProjectionInverse" );
projIT = g_pEffect->GetParameterBySemantic( NULL, "ProjectionInverseTranspose" );
random = g_pEffect->GetParameterBySemantic( NULL, "Random" );
refraction = g_pEffect->GetParameterBySemantic( NULL, "Refraction" );
renderCT = g_pEffect->GetParameterBySemantic( NULL, "RenderColorTarget" );
renderDST = g_pEffect->GetParameterBySemantic( NULL, "RenderDepthStencilTarget" );
renderTC = g_pEffect->GetParameterBySemantic( NULL, "RenderTargetClipping" );
renderTD = g_pEffect->GetParameterBySemantic( NULL, "RenderTargetDimension" );
specular = g_pEffect->GetParameterBySemantic( NULL, "Specular" );
specularP = g_pEffect->GetParameterBySemantic( NULL, "SpecularPower" );
standarGlob = g_pEffect->GetParameterBySemantic( NULL, "StandardGlobal" );
TextureMat = g_pEffect->GetParameterBySemantic( NULL, "TextureMatrix" );
time = g_pEffect->GetParameterBySemantic( NULL, "Time" );
UnitsScale = g_pEffect->GetParameterBySemantic( NULL, "UnitsScale" );
view = g_pEffect->GetParameterBySemantic( NULL, "View" );
viewI = g_pEffect->GetParameterBySemantic( NULL, "ViewInverse" );
viewIT = g_pEffect->GetParameterBySemantic( NULL, "ViewInverseTranspose" );
viewP = g_pEffect->GetParameterBySemantic( NULL, "ViewProjection" );
viewPI = g_pEffect->GetParameterBySemantic( NULL, "ViewProjectionInverse" );
viewPIT = g_pEffect->GetParameterBySemantic( NULL, "ViewProjectionInverseTranspose" );
world = g_pEffect->GetParameterBySemantic( NULL, "World" );
worldI = g_pEffect->GetParameterBySemantic( NULL, "WorldInverse" );
worldIT = g_pEffect->GetParameterBySemantic( NULL, "WorldInverseTranspose" );
worldV = g_pEffect->GetParameterBySemantic( NULL, "WorldView" );
worldVI = g_pEffect->GetParameterBySemantic( NULL, "WorldViewInverse" );
worldVIT = g_pEffect->GetParameterBySemantic( NULL, "WorldViewInverseTranspose" );
worldVP = g_pEffect->GetParameterBySemantic( NULL, "WorldViewProjection" );
worldVPI = g_pEffect->GetParameterBySemantic( NULL, "WorldViewProjectionInverse" );
worldVPIT = g_pEffect->GetParameterBySemantic( NULL, "WorldViewProjectionInverseTranspose" );

        g_pEffect->GetDesc( &pEffectDesc );

        for( int index=0; index< pEffectDesc.Parameters; index++ )
        {
                D3DXHANDLE hParam = g_pEffect->GetParameter( NULL, index );
```

```
                        D3DXPARAMETER_DESC pDesc;
                        g_pEffect->GetParameterDesc( hParam, &pDesc );

                        if((pDesc.Type == D3DXPT_TEXTURE3D)){
                                        LPDIRECT3DVOLUMETEXTURE9 value;
                                        D3DXHANDLE hAnnot = g_pEffect->GetAnnotationByName( hParam, "ResourceName" );
                                        const char* Path;
                                        std::ostringstream oss, debug;
                                        g_pEffect->GetString( hAnnot, &Path );
                                        oss << "..\\Data\\Textures\\" << Path;
                                        debug << "VolumeTexture PATH: " << oss;
                                        OutputDebugString( debug.str().c_str() );
                                        OutputDebugString( "\n" );
                                        if( FAILED( D3DXCreateVolumeTextureFromFileEx(g_pD3DDevice, oss.str().c_str(),
D3DX_DEFAULT, D3DX_DEFAULT, D3DX_DEFAULT, D3DX_DEFAULT, NULL, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_DEFAULT,
D3DX_DEFAULT, 0xff000000, NULL, NULL, &value) ) )
                                                        SetError("Impossibile caricare la VolumeTexture");
                                        g_pEffect->SetTexture( pDesc.Name, value );
                        }
                        else if((pDesc.Type == D3DXPT_TEXTURE)||(pDesc.Type == D3DXPT_TEXTURE1D)||(pDesc.Type ==
D3DXPT_TEXTURE2D)){
                                        LPDIRECT3DTEXTURE9 value;
                                        D3DXHANDLE hAnnot = g_pEffect->GetAnnotationByName( hParam, "ResourceName" );
                                        const char* Path;
                                        std::ostringstream oss, debug;
                                        g_pEffect->GetString( hAnnot, &Path );
                                        oss << "..\\Data\\Textures\\" << Path;
                                        debug << "Texture PATH: " << oss;
                                        OutputDebugString( debug.str().c_str() );
                                        OutputDebugString( "\n" );
                                        if( FAILED( D3DXCreateTextureFromFileEx(g_pD3DDevice, oss.str().c_str(),
D3DX_DEFAULT, D3DX_DEFAULT, D3DX_DEFAULT, NULL, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_DEFAULT, D3DX_DEFAULT,
0xff000000, NULL, NULL, &value) ) )
                                                        SetError("Impossibile caricare la texture");
                                        g_pEffect->SetTexture( pDesc.Name, value );
                        }
                }

                // Apply the technique contained in the effect
  D3DXHANDLE hTech;
  g_pEffect->FindNextValidTechnique( NULL, &hTech );
  g_pEffect->SetTechnique( hTech );
```

```
}



void Effect::SetEffectParams()
{
        //GET PARAMETERS VALUES
        g_pD3DDevice->GetTransform( D3DTS_WORLD, &wor );
    g_pD3DDevice->GetTransform( D3DTS_PROJECTION, &pro );
        g_pD3DDevice->GetTransform( D3DTS_VIEW, &vie );

        D3DXMatrixInverse( &proI, NULL, &pro );
        D3DXMatrixTranspose( &proIT, &proI );
        D3DXMatrixInverse( &vieI, NULL, &vie );
        D3DXMatrixTranspose( &vieIT, &vieI );
        vieP = vie * pro;
        D3DXMatrixInverse( &viePI, NULL, &vieP );
        D3DXMatrixTranspose( &viePIT, &viePI );
        D3DXMatrixInverse( &worI, NULL, &wor );
        D3DXMatrixTranspose( &worIT, &worI );
        worV = wor * vie;
        D3DXMatrixInverse( &worVI, NULL, &worV );
        D3DXMatrixTranspose( &worVIT, &worVI );
        worVP= wor * vie * pro;
        D3DXMatrixInverse( &worVPI, NULL, &worVP );
        D3DXMatrixTranspose( &worVPIT, &worVPI );
        tim = (float)GetTickCount()/1000.0f;

        //SET PARAMETERS VALUES
        g_pEffect->SetFloat( time, tim );
        g_pEffect->SetMatrix( proj, &pro );
        g_pEffect->SetMatrix( projI, &proI );
        g_pEffect->SetMatrix( projIT, &proIT );
        g_pEffect->SetMatrix( view, &vie );
        g_pEffect->SetMatrix( viewI, &vieI );
        g_pEffect->SetMatrix( viewIT, &vieIT );
        g_pEffect->SetMatrix( viewP, &vieP );
        g_pEffect->SetMatrix( viewPI, &viePI );
        g_pEffect->SetMatrix( viewPIT, &viePIT );
        g_pEffect->SetMatrix( world, &wor );
        g_pEffect->SetMatrix( worldI, &worI );
        g_pEffect->SetMatrix( worldIT, &worIT );
        g_pEffect->SetMatrix( worldV, &worV );
        g_pEffect->SetMatrix( worldVI, &worVI );
        g_pEffect->SetMatrix( worldVIT, &worVIT );
        g_pEffect->SetMatrix( worldVP, &worVP );
        g_pEffect->SetMatrix( worldVPI, &worVPI );
        g_pEffect->SetMatrix( worldVPIT, &worVPIT );

}


void Render()
{
  // Render the mesh with the applied technique
  if( FAILED( g_pD3DDevice->SetVertexDeclaration(g_MeshObject->g_VertDecl) ) )
                SetError("Errore nella funzione SetVertexDeclaration");

                        SetEffectParams();

                        cPasses = 0;

                        if( SUCCEEDED( g_pEffect->Begin(&cPasses, 0) ) ){
                                for (iPass = 0; iPass < cPasses; iPass++)
                                {
                                    g_pEffect->BeginPass(iPass);

                                        //I'm assuming you've only one mesh and only one material
                                        g_MeshObject->MeshData.pMesh->DrawSubset(0);

                                    g_pEffect->EndPass();
                                }
                                g_pEffect->End();
                        }

  }
}
```

To use this class you have simply to do that:

```
Effect* fire;
```

Then you've to load your mesh and create the effect file in a similar way:

```
LoadFXMesh( &fire->g_MeshObject, g_pD3DDevice, PathCamino );

fire = new Effect();

 fire->dwShaderFlags |= D3DXSHADER_NO_PRESHADER;

 if( FAILED( D3DXCreateEffectFromFile( g_pD3DDevice, PathFlameEffect,

  NULL, // CONST D3DXMACRO* pDefines,

  NULL, // LPD3DXINCLUDE pInclude,

  fire->dwShaderFlags, NULL, &fire->g_pEffect, &fire->pBufferErrors ))){

  OutputDebugString( "Errore nel caricamento del file Flame.fx\n" );

  LPVOID pCompilErrors = fire->pBufferErrors->GetBufferPointer();

  SetError("FX Creation Error");

  MessageBox(NULL, (const char*)pCompilErrors, "Fx Creation Error", MB_OK|MB_ICONEXCLAMATION);
  }

 fire->LoadEffectParams();
```

And finally call your:

```
fire->Render();
```

Notes & Improvements

Effect files are so easy to use that you could not complete your engine without using them. Probably this code is not correctly optimized and its not perfect, but its a simple demo for understanding how to manage .FX files, HLSL, including various handles and variables to use them and how to render effects. I hope that will help a lot of you to understand better these type of files.

The included MSVC++ code [http://www.gents.it/lavoro/Effects.h] demonstrate how to use these effects. This demo [http://www.gents.it/lavoro/D3D_MESH_EFFECTS_FRAMEWORK.zip] is a sample of effects created and rendered with this code.

Take also a look to MSDN Library [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/EffectFileReference/EffectFileFormat/EffectFileFormat.asp], NVDIA Developer Site [http://www.developer.nvidia.com/page/tools.html] and its SDK [http://developer.nvidia.com/object/sdk_home.html] and FX Composer [http://developer.nvidia.com/object/fx_composer_home.html]. Nothing too fancy.

Conclusion

Thanks for reading and I hope somebody found this article useful. Take a look to my website [http://www.gents.it/lavoro/].