Usare gli Effect files (.FX) nelle DirectX 9.0

di Mauro Gentile

Introduzione

Questo articolo è una semplice spiegazione di come caricare ed utilizzare gli effect file con le DirectX 9.0. Spiegherò anche come creare una classe per gestire al meglio le varie componenti e variabili dei file con estensione .FX. Tutto il codice si basa su semplici istruzioni della libreria D3DX e ed è testato su shaders NVDIA. Questo articolo è rivolto a quei programmatori o grafici che sono interessati ad utilizzare gli effect files (NVDIA o comunque compatibili alla DXSAS).

DXSAS significa: DirectX Standard Annotations and Semantics per i files .FX.

Perchè usare questi .FX files e perchè la notazione DXSAS è così importante?! Beh, prima di tutto utilizzare gli effetti sarà così semplice che si farà presto a dimenticarsi delle nottate passate sul codice Assembly o as impazzire dietro a Cg o altri linguaggi shaders. Inoltre gli effect files permettono al programmatore di implementare dirette tecniche dipendenti dalle capacità delle varie piattaforme senza dover modificare l'engine di utilizzo. La notazione DXSAS è inoltre così facile da comprendere e così utile che non utilizzarla ci complicherebbe solo la vita. Tale notazione ci permette di ottenere molte informazioni direttamente salvate all'interno del file .FX in pochi e semplici passi. Le tecniche di rendering diventano inoltre una sorta di "script" separato dall'engine.

Nel codice che segue mostrerò come applicare un semplice effetto ed applicarlo su una generica mesh.

Le basi

Un effect file non è nient'altro che un file di testo. La sola differenza consiste nell'estenzione .FX. Il file è diviso in tre sezioni:

1. Dichiarazione delle variabili – sono valori che devono essere impostati prima e/o durante il rendering [come texture, matrici, luci…]
2. Tecniche e passi – definiscono come una cosa deve essere "renderizzata" [come sampler e texture stage o la vertex declaration]
3. Funzioni - il codice vero e proprio dello shaker [scritto in HLSL]

I parametri sono definiti nell'effect file con tipo e nome e sono facilmente importabili nell'engine.

Le tecniche sono una sorta di “scatole nere” in cui sono inseriti gli stage Direct3D e ogni tecnica consiste di uno o più passi.

Il codice è scritto in HighLevelShadingLanguage o, per i vecchietti ancora inseparabili, in Assembly. Questo viene compilato a runtime.

Gli effect file possono anche essere inclusi nei file .X. Ma questa è tutta un'altra storia!

Codice dipendente dal file

Usare D3DX per caricare ed utilizzare un effect file è veramente facile. In questo caso importerò un semplice quadrato (salvato in un .X file) e vi applicherò il Flame.FX della NVDIA.

Il codice seguente è quanto serve per cominciare:

```
D3DXMESHCONTAINER_EX * g_Mesh;     // La mesh del quadrato dell' .X file
IDirect3DVertexDeclaration9 *g_VertDecl = NULL;          //Vertex declaration
ID3DXEffect* g_pEffect;            // Informazioni dell' Effect File
LPD3DXBUFFER pBufferErrors;        // Un  buffer che conterrà eventuali errori ottenuti alla creazione dell'effetto
D3DXEFFECT_DESC pEffectDesc;       // Una variabile usata per contenere la descrizione dell'effetto
DWORD dwShaderFlags;               // Le eventauli opzioni per la creazione dell'effetto
DWORD iPass;
UINT cPasses;                      // Due semplici variabili che useremo al momento del rendering
```

Per gestire i parametri bisogna dichiarare alcune variabili D3DXHANDLEche conterranno le informazioni dei parametri. Se si vuole caricare un parametro, di cui si conosce il nome e impostargli un valore, basterà fare così:

```
g_pEffect->SetParameterType( 'parameterName', ... );   // 'parameterName' è il nome effettivo del parametro
```

Ora supponiamo di voler impostare un parametro chiamato ‘noiseFreq’ di tipo float con il valore ‘0.3f’, un vettore chiamato ‘noiseAnim’ con I valori { 0.2, -0.4, 0.2 } e una texture. Lo si può fare in questo semplice modo:

```
g_pEffect->SetFloat( "noiseFreq", 0.3f );   // 'noiseFreq' è il nome del parametro
g_pEffect->SetValue( "noiseAnim", D3DXVECTOR3(0.2, -0.4, 0.2), sizeof(D3DXVECTOR3) );     // 'noiseAnim' è il nome del
parametro
D3DXCreateTextureFromFile(g_pD3DDevice, PathFlameTexture , &g_FlameTexture );    //Caricamento di una texture da file
```

```
g_pEffect->SetTexture( "noiseTexture", g_FlameTexture );                          //Impost oil parametro 'noiseTexture'
con la texture appena caricata
```

A questo punto è importante settare la tecnica che si vuole adoperare:

```
g_pEffect->SetTechnique("TechniqueName");      // 'TechniqueName' è il nome della tecnica da utilizzare
```

Ogni effetto richiede le informazioni sui vertici. Queste si possono facilmente estrarre dalla mesh. In genere una dichiarazione delle informazioni dei vertici è come quella che segue:

```
  // Vertex shader declaration
D3DVERTEXELEMENT9 vertexElement[] =                                               // ARRAY di informazioni sui vertici
{
{ 0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0 },     //Posizione
{ 0, 12, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL, 0 },      //Normale
{ 0, 24, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0 },    //Coordinate della Texture
{ 0, 36, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TANGENT, 0 },     //Tangente
{ 0, 48, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_BINORMAL, 0 },    //Bi-normale
{ 0, 60, D3DDECLTYPE_D3DCOLOR, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_COLOR, 0 },     //Colore
D3DDECL_END()
};

g_pD3DDevice->CreateVertexDeclaration( vertexElement, &g_VertDecl );    //stiamo creando la dichiarazione dei vertici
g_pD3DDevice->SetVertexDeclaration(g_VertDecl);      // 'g_VertDecl' è la "vertex declaration" dei vertici
```

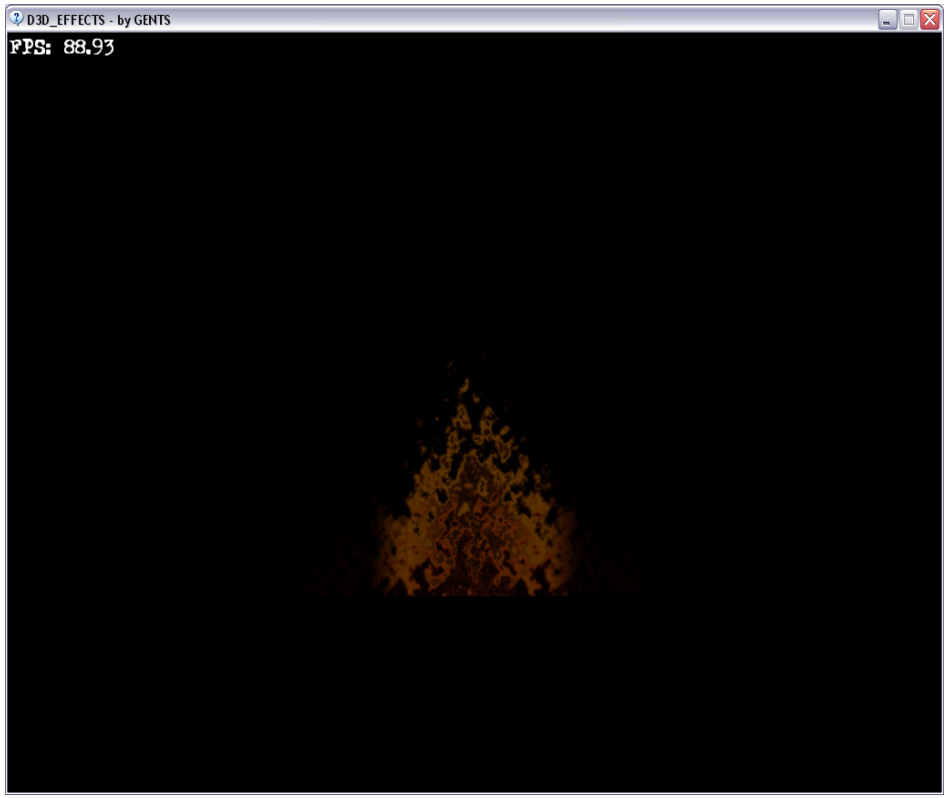
Bisogna notare che ci sono dei parametric che bisogna impostare solo al momento del caricamento dell'effect file ed altri che vanno aggiornati costantemente ogni frame. Questo lo si può dedurre e comprendere facilmente. Parametri come matrici, tempo e/o posizione, cambiano continuamente e quindi bisogna re-impostarli ogni frame.

A questo punto, dopo aver aggiornato I dovuti parametric, possiamo passare alla fase di rendering:

```
  // Inizio della scena 3D
  g_pD3DDevice->BeginScene();

        //Setto un parametro temporale
        g_pEffect->SetFloat( "ticks", (float)GetTickCount()/1000.0f );
        //Setto una matrice che richiede come valorele il risultato di World*View*Projection
        g_pEffect->SetMatrix( "wvp", &mWorldViewProjection );
```

```
        //Setto un parametro che necessita della matrice del mondo
        g_pEffect->SetMatrix( "world", &mWorld );

        //INIZIO AD EFFETTUARE IL RENDERING DELL'EFFETTO
        if( SUCCEEDED( g_pEffect->Begin(&cPasses, 0) ) ){
                //Itero per ogni passo dela tecnica scelta
                for (iPass = 0; iPass < cPasses; iPass++)
                {
                    g_pEffect->BeginPass(iPass);
                    // renderizzo la mesh
                    g_Mesh->MeshData.pMesh->DrawSubset(0);
                    g_pEffect->EndPass();
                }
                g_pEffect->End();
        }

    // Fine della scena 3D
    g_pD3DDevice->EndScene();
```

Ora l'operazione è completate e si può osservare il risultato sullo schermo.



Codice indipendente dal file

Il codice seguente è ciò che necessita per cominciare:

```
D3DXMESHCONTAINER_EX * g_Mesh;      // La mesh del quadrato dell' .X file
ID3DXEffect* g_pEffect;             // Informazioni dell' Effect File
LPD3DXBUFFER pBufferErrors;         // Un  buffer che conterrà eventuali errori ottenuti alla creazione dell'effetto
```

```
D3DXEFFECT_DESC pEffectDesc;       // Una variabile usata per contenere la descrizione dell'effetto
DWORD dwShaderFlags;               // Le eventauli opzioni per la creazione dell'effetto
DWORD iPass;
UINT cPasses;                      // Due semplici variabili che useremo al momento del rendering
```

Per gestire le informazioni della notazione DXSAS bisogna nuovamente dichiarare dei D3DXHANDLE per contenere le informazioni dei vari parametri. Non tutti i parametri necessitano di usare o utilizzano la notazione DXSAS. Tutti questi parametri vengono caricati al momento della creazione dell'effetto e sono impostati con il vaolore salvato all'interno del file .FX. Tali valori potranno essere cambiati o modificando manualmente il file .FX oppure dichiarando opportuni D3DXHANDLE per modificarli.

Se si vuole implementare qualcosa che sia indipendente dal file bisogna però pensare che non si conoscono i nomi dei parametri. E così leggendo la DXSAS Microsoft Reference si capisce che bisogna dichiarare un D3DXHANDLE per ogni parametro che utilizza tale notazione, in un modo simile a questo:

```
//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;
```

Chiaramente per ogni handler bisogna chreare una variabile dove salvare le informazioni:

```
//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;
```

E si può poi procedure in questo modo :

```
void Effect::LoadEffectParams()
{

 //OTTENGO I 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" );

//ottengo la descrizione del file
g_pEffect->GetDesc( &pEffectDesc );

//controllo se sto caricando tra I parametric qualcuno che indica una texture e che ha quindi come
//descrizione Il tipo “texture”
for( int index=0; index<pEffectDesc.Parameters; index++ )
{
    D3DXHANDLE hParam = g_pEffect->GetParameter( NULL, index );
    D3DXPARAMETER_DESC pDesc;

    // ottengo la descrizione del parametro stavolta
    g_pEffect->GetParameterDesc( hParam, &pDesc );

    // controllo se ho una VolumeTexture e nel caso, la carico e la imposto
    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 );
        }

        // controllo se ho una Texture e nel caso, la carico e la imposto

        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 );
        }
    }



    // Applico la prima tecnica contenuta nell'effect file
    D3DXHANDLE hTech;
    g_pEffect->FindNextValidTechnique( NULL, &hTech );
    g_pEffect->SetTechnique( hTech );
  }
```

Ora siamo pronti per passare al rendering, però dobbiamo ricordarci di aggiornare le variabili necessarie (qui di seguito mostro come aggiornarne alcune, come le matrici ad esempio):

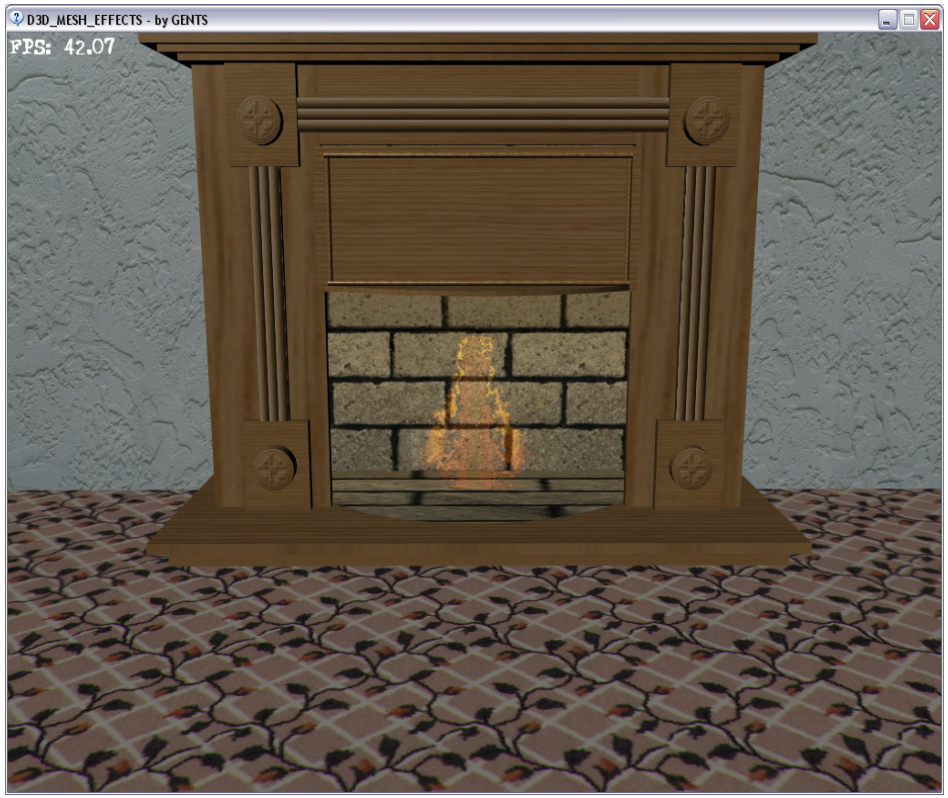
```
void Effect::SetEffectParams()
{
        //OTTENGO I PARAMETERS VALUES
        g_pD3DDevice->GetTransform( D3DTS_WORLD, &wor );
    g_pD3DDevice->GetTransform( D3DTS_PROJECTION, &pro );
        g_pD3DDevice->GetTransform( D3DTS_VIEW, &vie );
```

```
      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;

//E POI SETTO I PARAMETERS VALUES NELL'EFFETTO

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 );

}
```

Ecco che possiamo procedure al rendering nel modo usuale :

```
  // Renderizzo la mesh con la techica scelta applicata su essa
  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();
                       }

  }
```



La classe ‘Effect’

Ho implementato inoltre una classe che aiuta in modo chiaro con tutto questo codice:

```
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, 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();
                       }

  }
}
```

Per usare qusta classe basta fare in modo simile a questo:

```
Effect* fire;
```

Qui carico la mesh da un file .X e creo l'effetto:

```
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();
```

e finalmente chiamo la mia funzione:

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

Note & Miglioramenti

Gli effect files sono così facili da usare che oramai non si può completare un engine che non li supporti. Probabilmente questo codice contiene qualche errore o non è ottimizzato al meglio, ma serve solo come dimostrazione di come usare gli .FX file, l'HLSL, includendo vari handlers e variabili e quindi renderizzarli in modo rapido. Spero davvero che questo articolo/tutorial possa aiutare qualcuno a comprendere di più questi misteriosi file dall'estensione .FX.

Il codice incluso in MSVC++ [http://www.gents.it/lavoro/Effects.h] dimostra come usare questi. Questa demo [http://www.gents.it/lavoro/D3D_MESH_EFFECTS_FRAMEWORK.zip] è invece un esempio di effect create e renderizzati con il codice qui presentato.

Mi raccomando, consiglio di dare un'occhiata anche a questi links: 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] e i suoi SDK [http://developer.nvidia.com/object/sdk_home.html] e FX Composer [http://developer.nvidia.com/object/fx_composer_home.html].

Alla prossima.

Conclusione

Grazie della lettura e spero che qualcuno trovi utile questo articolo. Date anche uno sguardo al mio website [http://www.gents.it/lavoro/].