Sessió 5
Shaders

Durant les dues properes sessions aprendrem a incloure shaders dins del nostre projecte.

QUÈ SÓN ELS SHADERS?

Un shader és un conjunt de línies de codi que s'executen directament dins de la GPU (la CPU incorporada dins de la tarja de vídeo), la forma de programar aquestes línies de codi normalment ho farem mitjançant HLSL o directament codi ASM. També existeixen altres llenguatges de shaders com GLSL (d'OpenGL) o CG (d'nvidia).

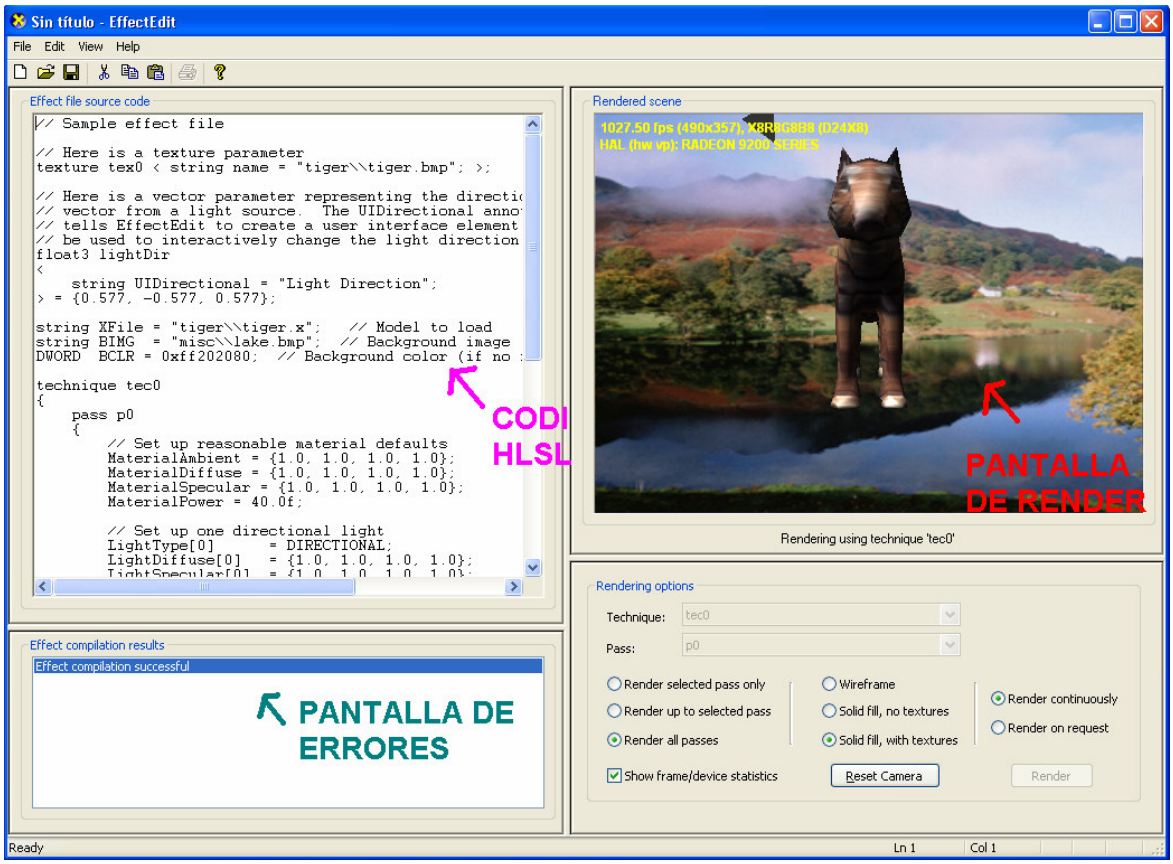
Dins del terme de shader, diferenciem dos tipus de shader:

- ***vertex shader,* és un codi que s'executa per cadascun dels vèrtexs de la nostra malla i modificarà a nivell de vèrtex. S'utilitza per exemple per fer efectes d'ones, skinning, ...**
- ***pixel shader,* és un codi que s'executa a nivell de pixel i modificarà els pixels de renderitçat de l'objecte. Molt utilitzat per efectes de post-processat com *glow, shadowmap, zblur, ...***

EffectEdit

El SDK de Microsoft DirectX porta una eina (segons versió) per desenvolupar de forma ràpida i senzilla shaders, aquesta aplicació s'anomena *EffectEdit.exe*.

Quan obrim l'aplicació ens trobem amb una finestra com la següent.



En aquesta aplicació veiem quatre separacions de la finestra:

- ***Codi HLSL,* en aquesta finestra podem modificar el codi HLSL en temps real i així poder veure el resultat dins de la finestra de render**
- ***Finestra d'errors,* si escribim algun error dins del nostre codi de shader el resultat de l'error sortirà en aquesta finestra**
- ***Finestra de Render,* en aquesta finestra veurem el resultat de l'execució del *SHADER*.**
- ***Finestra de control de Render,* en aquesta pantalla situada a la dreta abaix podem controlar la forma de renderitzat, si volem renderitzar una passada en específic o renderitzar continuament o només "*on demand*".**

Utilitzant aquesta eina implementarem diferents shaders durant el transcurs de la classe d'avui.

- ***Textured.fx***

  Amb aquest primer shader farem un render standard d'una malla 3D amb una textura de difús. Definim un vèrtex shader i un pixel shader que rebran un vèrtex amb estructura VNormalVertex i PNormalVertex respectivament. El codi serà el següent.

```
texture tex0 < string name = "tiger\\tiger.bmp"; >;

string XFile = "tiger\\tiger.x";   // Model to load

DWORD  BCLR = 0xff202080;  // Background color (if no image)

float4x4 wvpMatrix : WORLDVIEWPROJECTION;

sampler DiffuseTextureSampler  = sampler_state
{
    Texture = (tex0);
    MipFilter = LINEAR;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
};


struct VNormalVertex
{
    float3 pos : POSITION;
    float2 uv : TEXCOORD0;
};


struct PNormalVertex
{
    float4 pos : POSITION;
    float2 uv : TEXCOORD0;
};

PNormalVertex RenderNormalsVS(
    VNormalVertex IN,
    uniform float4x4 WorldViewProj)
{
    PNormalVertex OUT = (PNormalVertex )0;
    OUT.pos=mul(float4(IN.pos,1),WorldViewProj);
    OUT.uv=IN.uv;

    return OUT;
}

float4 RenderNormalsPS(PNormalVertex IN) : COLOR
{
    return tex2D(DiffuseTextureSampler,IN.uv);
}

technique tec0
{
    pass p0
    {
        VertexShader =compile vs_1_1 RenderNormalsVS(wvpMatrix);
        PixelShader = compile ps_1_1 RenderNormalsPS();
    }
}
```

Ens trobem el codi que defineix el següent:

- `texture tex0 < string name = "tiger\\tiger.bmp"; >`**, aquesta funcionalitat d'HLSL ens defineix una textura que utilitzarem dins del nostre codi. Carregant el fitxer tiger.bmp com a textura tex0.**
- `string XFile = "tiger\\tiger.x"`**, aquesta línia defineix en l'aplicació EffectEdit la malla de tipus .x (fitxer específic de DirectX) que utilitzarem pel nostre shader.**
- `float4x4 wvpMatrix : WORLDVIEWPROJECTION`**, ara ens trobem amb el codi necessari per definir la matriu de tipus WORLDVIEWPROJECTION amb nom wvpMatrix**
- `sampler DiffuseTextureSampler = sampler_state`**, aquest codi defineix el sampler de textura que utilitzarem per accedir-hi a la textura tiger.bmp. Li passem diferents estats amb els següents atributs**
  - `Texture = (tex0)`**, aquest codi defineix que volem utilitzar en aquest sampler la textura tex0 definida previament (tiger.bmp)**
  - `MinFilter = LINEAR`**, aquest codi defineix el filtre de MinMapping, en aquest cas Lineal**
  - `MagFilter = LINEAR`**, aquest codi defineix el filtre de MagMapping, en aquest cas Lineal**
  - `MipFilter = LINEAR`**, aquest codi defineix el filtre de MipMapping, en aquest cas Lineal**
- **La següent part de codi defineix el tipus de vèrtex que utilitzarem en el vèrtex shader. Definint una estructura amb una posició formada per 3 components (x.y,z) i un parell de coordenades de textura formada per 2 componets (u,v). La part de codi escrita després del : s'anomena semàntica i defineix el tipus de paràmetre del vèrtex.**

```
struct VNormalVertex
{
    float3 pos : POSITION;
    float2 uv : TEXCOORD0;
};
```

- **La següent part de codi defineix el tipus de vèrtex que utilitzarem en el pixel shader. Definint una estructura amb una posició formada per 3 components (x.y,z) i un parell de coordenades de textura formada per 2 componets (u,v)**

```
struct VNormalVertex
{
    float3 pos : POSITION;
    float2 uv : TEXCOORD0;
};
```

- **Ara ens trobem per fi amb el codi del vèrtex shader, on declara la funció `RenderNormalsVS` rebent com a paràmetres un vèrtex `VNormalVertex IN` i un paràmetre de tipus uniform `uniform float4x4 WorldViewProj`.**

  *** Els paràmetres de tipus uniform són paràmetres que es passen per cadascun dels vèrtexs de la nostra malla, el paràmetre serà el mateix per cadascun dels vèrtexs i es pasarà al declarar la *technique*.**

```
PNormalVertex RenderNormalsVS(
    VNormalVertex IN,
    uniform float4x4 WorldViewProj)
{
    PNormalVertex OUT = (PNormalVertex )0;
    OUT.pos=mul(float4(IN.pos,1),WorldViewProj);
    OUT.uv=IN.uv;

    return OUT;
}
```

- **El que fa el shader en aquest fragment de codi és multiplicar el vèrtex d'entrada per la matriu de WorldViewProj donant un valors de vèrtex transformat a la càmara que tinguem i passar-li a les coordenades uv les coordenades de textura d'entrada pel pixel shader.**
- **La següent funció que en strobem dins del *shader* és la funció referent al pixel shader.**

```
float4 RenderNormalsPS(PNormalVertex IN) : COLOR
{
    return tex2D(DiffuseTextureSampler,IN.uv);
}
```

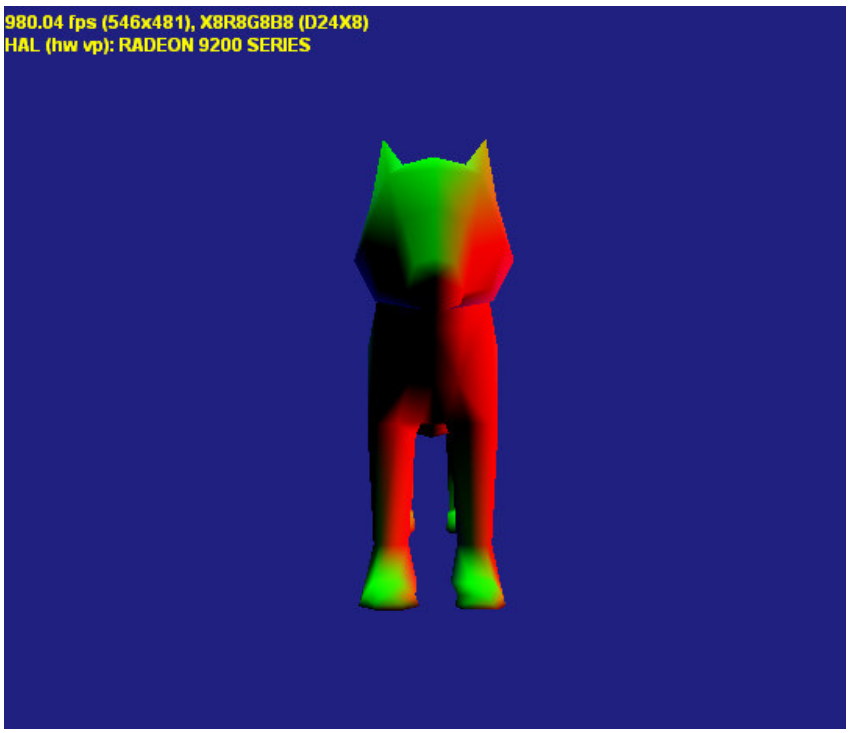
- **Aquest codi de shader retornara el valor del pixel (float4) que retorna la funció tex2D passant-li com a paràmetres el sampler de la textura i les coordenades de textura.**
- **Per últim ens trobem amb la declaració de la technique estarà formada pel vertex i el pixel shader.**

```
technique tec0
{
    pass p0
    {
        VertexShader =compile vs_1_1 RenderNormalsVS(wvpMatrix);
        PixelShader = compile ps_1_1 RenderNormalsPS();
    }
}
```

Amb aquest primer shader ens hem apropat amb el món dels shaders amb l'eina de DirectX *EffectEdit.exe*. Durant la classe d'avui dessenvoluparem els següents shaders que ens seran útils per fer un rendr més vistòs dins del nostre videojoc.

- *Normal.fx*

Amb aquest shader podrem visualitzar les normals de la malla de forma visual. El resultat serà com la següent imatge.



- *LightTextured.fx*

Amb aquest shader podrem visualitzar la iluminació a nivell de vèrtex.

- *LightTexturedPerPixel.fx*

Amb aquest shader podrem visualitzar la iluminació a nivell de pixel.

- *LightTexturedAmbient.fx*

Amb aquest shader podrem visualitzar la iluminació a nivell de pixel amb una constant d'ambient.

- *Specular.fx*

Amb aquest shader podrem visualitzar la iluminació a nivell de pixel amb specular.

- *LightTexturedPerPixelCellShading.fx*

Amb aquest shader podrem visualitzar la iluminació amb efecte cell shading.

- *SpecularNightVision.fx*

Amb aquest shader simularem una visió nocturna.

Normals en fitxers .ASE

Per realitzar totes aquestes tècniques dins del nostre videojoc FPS necessitem tenir les normals de cada vèrtex de la nostra malla.

Per coneguir les normals de la malla haurem de repasar el codi que llegia un fitxer .ASE i llegir-les. Per fer això buscarem el tag *MESH_NORMALS i llegirem cadascuna de les normals dels vèrtexs que trobem amb el tag *MESH_VERTEXNORMAL.

Un exemple de fitxer .ase serà la següent:

```
*MESH_NORMALS {
	*MESH_FACENORMAL 0  0.0000 0.0000 -1.0000
		*MESH_VERTEXNORMAL 0	0.5000 0.8000 -1.0000
		*MESH_VERTEXNORMAL 2	0.0000 0.0000 -1.0000
		*MESH_VERTEXNORMAL 3	0.0000 0.0000 -1.0000
	*MESH_FACENORMAL 1  0.0000 0.0000 -1.0000
		*MESH_VERTEXNORMAL 3	0.0000 0.0000 -1.0000
		*MESH_VERTEXNORMAL 1	0.0000 0.0000 -1.0000
		*MESH_VERTEXNORMAL 0	0.0000 0.0000 -1.0000
	*MESH_FACENORMAL 2  0.0000 0.0000 1.0000
		*MESH_VERTEXNORMAL 4	0.0000 0.0000 1.0000
		*MESH_VERTEXNORMAL 5	0.0000 0.0000 1.0000
		*MESH_VERTEXNORMAL 7	0.0000 0.0000 1.0000
}
```

Per rebre la normal invertirem el valor de x i de z i hem de tenir en compte l'ordre de les components de la normal serà x, z, y és a dir que si volem obtenir la normal del primer vèrtex del primer triangle ens trobem amb una normal (-0.5, -1.0, -0.8).

El codi ha sigut modificat dins de la classe CASEObject en el nostre projecte, per qualsevol consulta mireu el codi.

EffectManager

Amb aquest nou mànager s'ens permetrà carregar arxius de tipus fx (shaders) d'HLSL dins del nostre projecte. El primer que ens trobem és amb una classe de tipus singleton que deriva de la classe CXMLParser que ens permetrà llegir fitxers XML.

En el fitxer xml "./data/xml/effects.xml" ens trobarem definits el fitxers .x que volem carregar dins de la nostra aplicació, el contingut del fitxer és el següent:

```
<effects>
       <effect name="normal" filename="data/models/effects/normal.fx"/>
       <effect name="light_textured_per_pixel_ambient"
          filename="data/models/effects/LightTexturedPerPixelAmbient.fx"/>
</effects>
```

En el fitxer ens trobem definits els effects amb el nom que utilitzarem dins del nostre joc per identificar el shader que utilizarem. Llavors aquesta classe carregarà els fitxers .x i els ficara dins d'un mapa que tindrà com a clau el nom del shader i com valor l'efecte (el shader).

Dins d'aquesta classe ens trobem amb els següents mètodes:

- `LoadEffects`**, mètode privat que recorrerà el mapa i carregarà el fitxer .fx amb el shader.**
- `UnloadEffects`**, aquest mètode s'encarregarà de descarregar els shaders de memòria.**
- `CleanUp`**, com sempre utilitzarem aquest mètode per eliminar de memòria les instàncies dels shaders del nostre joc.**
- `Reload`**, amb aquest mètode podrem recarregar els shaders en temps d'execució.**
- `Load`**, aquest mètode s'encarregarà de carregar el fitxer xml i tots els efectes.**
- `GetEffect`**, mitjançant aquest mètode podrem conseguir l'efecte del mapa segons el nom del efecte per poder ser utilitzat dins del nostre joc.**

RenderScenary

Dins d'aquesta sessió renderitzarem l'escenary utilitzant l'efecte `"light_textured_per_pixel_ambient"` **que ens permetrà renderitzar les malles de l'escenari amb una il·luminació de tipus direcció i una il·luminació ambiental.**

Per realitzar aquesta funció modificarem la trucada al render de l'ASEObject per un nou mètode que es dirà `RenderWithTechnique` **que farà lo següent:**

- **Agafar l'efecte** `"light_textured_per_pixel_ambient"` **del manager d'efectes**
- **Establir la matriu** `"g_WorldViewProjMatrix"` **segons els valors de les matrius actuals de tipus World, View i Projection.**
- **Establir la *technique*** `"LightTexturedPerPixelAmbientTechnique"` **de l'efecte que hem agafat en el primer pas.**
- **Establim el *Vèrtex Declaration* que utilitzem dins d'aquest shader (utilitzem un mètode static per crear un vèrtex declaration comú per tots els ASEObjects).**
- **Establim la matriu** `"g_WorldMatrix"` **que s'utilitzarà dins del shader.**
- **I per últim fem el begin de l'efecte, el render segons el número de passades que té el shader i l'end de l'efecte per terminar.**
- **I per terminar establim el *Vertex Declaration* a NULL.**