Shader

Le but de ce TD est de vous familiariser avec l'utilisation des vertex shader dans une application DirectX.

Instructions :

1) Suivez le tutoriel de FXComposer 2.5 qui est un éditeur de shader:
   http://developer.download.nvidia.com/tools/FX_Composer/2.5/FX_Composer_Quick_Tutorial.pdf

2) Dans la library de shader de FXComposer 2.5 se trouve un vertex shader appellé "Balloon", une version modifiée de ce shader vous est fournie (modifiée pour pouvoir être compilée par Visual C++).
   La solution fournie permet de charger un mesh (classe MeshLoader) et de l'afficher.
   L'objectif est d'appliquer le shader "Balloon" au mesh.

   - Complétez la classe Shader.

   - Appliquez le shader au mesh

A ce stade le mesh n'est pas visible car le shader ne connait pas les matrices de l'objet et de la camera.

- Dans la fonction Render() de Game, passez les variables non-ajustables au shader

Pour cela vous devrez faire des manipulations de matrices (Inverse, transposé, multiplication).
A ce stade, le mesh est visible mais semble éclairé par l'intérieur. C'est à cause de la lumière par défaut du shader qui est omnidirectionnelle centrée à l'origine.

- Passer au shader les variables ajustables de lumière de la scène

A ce stade, vous devriez voir le mesh éclairé comme au début. Mais l'effet du shader n'est pas visible car il déforme peu le mesh.

- Faites varier un facteur fInflate de 0.0f à 2.0f avant de le passer au shader

Vous devez obtenir un mesh qui gonfle comme un ballon. Toutefois le résultat obtenu est gris, les couleurs originales des matériaux du mesh ont été remplacées par la couleur par défaut du pixel shader.

- Passer au shader la variable SurfaceColor en allant chercher dans le mesh chaque couleur de matériau

Vous devez obtenir le mesh avec ses couleurs originales qui gonfle comme un ballon.

Informations :

Pour fonctionner correctement, un shader a besoin d'un certain nombre d'informations données par l'application:

- Position et orientation de l'objet (World)
- Position et orientation de la camera (View)
- Matrice de projection (Projection)
- Autres données spécifiques qu shader

Généralement on donne à un shader une matrice WorldViewProjection (qui est le résultat de la multiplication de la matrice de World, de la matrice de View et de la matrice de Projection).

Pour savoir quelles informations fournir à un shader, il faut étudier son code (fichier texte, ouvrir avec le bloc-note par exemple). Dans Balloon.fx on trouve :

- Une entête
- Des defines pour adapter le code aux applications
- Les variables non-ajustables (untweakables) qui sont donc à fournir obligatoirement à chaque frame.

```
// transform object vertices to world-space:
float4x4 gWorldXf : World < string UIWidget="None"; >;
// transform object normals, tangents, & binormals to world-space:
float4x4 gWorldITXf : WorldInverseTranspose < string UIWidget="None"; >;
// transform object vertices to view space and project them in perspective:
float4x4 gWvpXf : WorldViewProjection < string UIWidget="None"; >;
// provide tranform from "view" or "eye" coords back to world-space:
float4x4 gViewIXf : ViewInverse < string UIWidget="None"; >;
```

- Les variables ajustables (tweakables) qui ont des valeurs par défaut.

```
float4 gLamp0DirPos : POSITION < // or direction, if W==0
    string Object = "Light0";
    string UIName =  "Lamp 0 Position/Direction";
    string Space = (LIGHT_COORDS);
> = {-0.5f,2.0f,1.25f,1.0};
```

- Les structures de données passées aux shaders

```
/* data from application vertex buffer */
struct appdata {
    float3 Position     : POSITION;
    float4 UV           : TEXCOORD0;
    float4 Normal : NORMAL;
    float4 Tangent      : TANGENT0;
    float4 Binormal     : BINORMAL0;
};
/* data passed from vertex shader to pixel shader */
struct vertexOutput {
    float4 HPosition    : POSITION;
    float2 UV           : TEXCOORD0;
    float3 LightVec     : TEXCOORD1;
    float3 WorldNormal  : TEXCOORD2;
    float3 WorldTangent : TEXCOORD3;
    float3 WorldBinormal : TEXCOORD4;
    float3 WorldView    : TEXCOORD5;
};
```

- Le vertex shader

```
vertexOutput balloon_VS(appdata IN,
      uniform float4x4 WorldITXf, // our four standard "untweakable" xforms
      uniform float4x4 WorldXf,
      uniform float4x4 ViewIXf,
      uniform float4x4 WvpXf,
      uniform float Inflate,
      uniform float4 LampDirPos
) {...}
```

- Le pixel shader

```
float4 balloonPS(vertexOutput IN,
                uniform float3 SurfaceColor,
                uniform float Kd,
                uniform float Ks,
                uniform float SpecExpon,
                uniform float3 LampColor,
                uniform float3 AmbiColor
) { ... }
```

- Les techniques et les passes

```
technique Main <
      string Script = "Pass=p0;";
> {
    pass p0 <
      string Script = "Draw=geometry;";
    > {
            VertexShader = compile vs_2_0 balloon_VS(gWorldITXf, gWorldXf,
                        gViewIXf, gWvpXf, gInflate, gLamp0DirPos);
            ZEnable = true;
            ZWriteEnable = true;
            ZFunc = LessEqual;
            AlphaBlendEnable = false;
            CullMode = None;
            PixelShader = compile ps_2_0 balloonPS(gSurfaceColor, gKd, gKs,
                        gSpecExpon, gLamp0Color, gAmbiColor);
    }
}
```

Pour appliquer un shader sur un modèle, un algorithme doit être respecté à chaque frame :

```
m_D3DDevice->BeginScene()
      // Donner au shader les variables nécessaires. Ex:
      Shader -> SetMatrix ("gWorldXf", &mWorld) ;

      Shader -> Begin() // Recupère le nombre de passes de la technique
      Pour i allant de 0 au nombre de passes faire
            Shader -> BeginPass(i)
                  Afficher le mesh
            Shader -> EndPass()
      Shader -> End()
m_D3DDevice->EndScene()
```

Documentation : à chercher sur MSDN (http://msdn.microsoft.com/en-us/library)

D3DXCreateEffectFromFile Function

ID3DXEffect Interface

ID3DXBaseEffect Interface