User DirectX Shaders

3DSimED3 has support for user-defined DirectX shaders.

This support is through a combination of GFX effect files and HLSL vertex & pixel shaders.
The GFX files contain a description of the Shader including references to vertex shader & pixel shader source files. These vertex & pixel shader source files contain the HLSL code required to render the Shader.

The GFX files, vertex shaders source (.VSH), and pixel shaders source (.PSH) are almost identical to the formats used in the rFactor sims, so use the .GFX, .VSH & .PSH supplied with rFactor as examples (all archived in `coreshaders.mas`).

- All files must be placed in the `HardwareShaders` sub-folder of 3DSimED3.
- If the GFX file contains a Shader definition which is defined in 3DSimED3 this definition will replace the internal one.
- Should the GFX file define a new Shader this will be added to the available DirectX Shaders which the user can set for a Material see Material Edit Pane
- 3DSimED3 will compile the vertex & pixel shader HLSL text source to binary format. These binary shaders, which have the .OSH extension, are much faster for 3DSimED3 to load than the text source format.
- 3DSimED3 first looks for .OSH binary shaders, then for the text HLSL source, and then for an internally defined binary shader. This allows the internal 3DSimED3 vertex & pixel shaders to be overridden by user vertex & pixel shaders.
- If the Shader description in the GFX refers to vertex & pixel shaders which are already defined by 3DSimED3 there is no need to include the vertex & pixel shader source.
- If the text source for a vertex or pixel shader is modified the corresponding .OSH binary should be deleted as the binary .OSH takes precedence over the text source.

See How To Create User DirectX Shaders

How To Create User DirectX Shaders

For more information on how 3DSimED3 uses shaders please see User DirectX Shaders.

To create DirectX shaders for use in 3DSimED3 the shader needs a descriptor placed in a .GFX file and may also require vertex & pixel shader source.
An example of how to add a Shader is given below. This example creates a Shader that is similar to a simple diffuse shader but it only displays the red channel; the green and blue channels are filtered out.

Create Description

- The description of the shader needs to be written to a .GFX file.
- The GFX files must be placed in the HardwareShaders sub-folder. 3DSimED3 reads all .GFX files residing in this sub-folder when it starts.
- Multiple descriptions can be written to .GFX files.
- The descriptor entries have the same format as that found in rFactor and other sims based on gMotor2 engine.

Note, that the description below is very similar to the simple diffuse description used in rFactor except for setting the pixel shader function to be ps_DiffuseRedOnly to be found in the source file ps_DiffuseRedOnly.psh

Example .GFX

```
ShaderName=L2DiffuseRedOnly
{
  ShaderDesc="T1, red only"
  ShaderLongDesc="Diffuse lighting, tex1 only, only red channel DX9."
  ShaderLevel=(2) // DX9
  {
    Pass=(0) // normal
    {
      VertexShader=vs30_blinnDiffuseT0
      {
        File=vs30_blinnDiffuse.vsh
        Language=HLSL
        Define=(NUMTEX, 1)
        ShaderConstants=(Default, OmniLight)
      }
      PixelShader=ps_DiffuseRedOnly
      {
        File=ps_DiffuseRedOnly.psh
        Language=HLSL
```

```
        ShaderConstants=Lighting
        StageState=(0, DiffuseMap, Modulate)
        SamplerState=(0, Wrap, Wrap, Wrap)
      }
    }
  }
}
```

Creating Vertex and Pixel Shaders Source

Writing vertex and pixel shaders requires some understanding of HLSL, the language used to write these shaders. The example below would also work in rFactor which itself uses HLSL shaders.
In the .GFX above only a new pixel shader was defined and this must have a source file named `ps_DiffuseRedOnly.psh`

Example Pixel Shader

`ps_DiffuseRedOnly.psh`

```
float4 ambientColor  : register (c0);
float4 diffuseColor  : register (c1);
float4 worldLight    : register (c2);
float4 specularColor : register (c3); //power in c3.a
float4 blendPct      : register (c4); //base/cube blend pct in c4.a

struct PS_INPUT
{
   float4 Color  : COLOR0;
   float4 Omni   : COLOR1;
   float  Fog    : FOG;
   float2 Tex    : TEXCOORD0; // base
   float3 Normal : TEXCOORD2;
   bool3  Blend  : TEXCOORD3; // not used yet
};

struct PS_OUTPUT
{
   float4 Color : COLOR0;
};

sampler2D sTex0 : register (s0);

PS_OUTPUT ps_DiffuseRedOnly(PS_INPUT Input)
{
  PS_OUTPUT Out;

  // sample the textures
  float4 tex0 = tex2D (sTex0, Input.Tex.xy);  // sample tex0


  // compute diffuse color
  float3 normal = normalize (Input.Normal);
  half4 diffuse = saturate ((saturate (dot (worldLight, normal)) * diffuseColor) + ambientColor + Input.Omni);
  Out.Color = (diffuse * saturate (tex0)) * Input.Color;
  Out.Color.gb = 0 ;

  return (Out);
}
```