Integrating DirectX Shaders in 3ds Max A Developer's Perspective

Neil Hazzard
Software Engineer
Autodesk Media & Entertainment Division

Overview

- Real time Shaders in 3ds max
- Support for DirectX Effect files
- DirectX Standard Material
- SDK Support for DX Shaders

Perspective
© CODEMASTERS SOFTWARE COMPANY Ltd. CENTRAL TECHNOLOGY GROUP

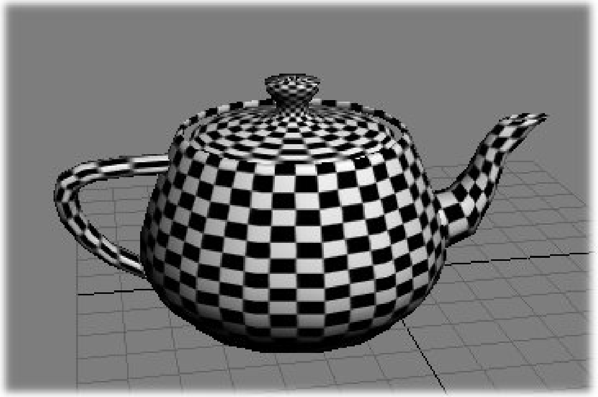
Real time Shaders in 3ds max

3ds max 2.0

- First DirectX Release
- DirectX 5.0
- T & L in software

3ds max 3.0

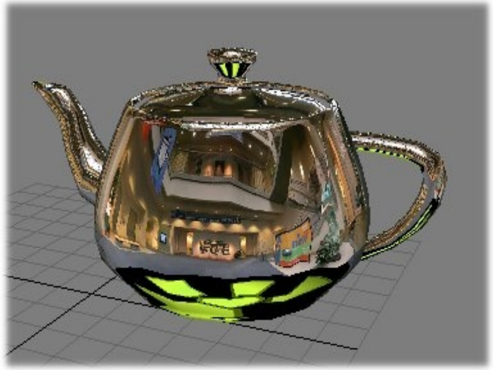
- DirectX 6.0
- Texture Improvements
- Very Similar to max 2.0



Real time Shaders in 3ds max

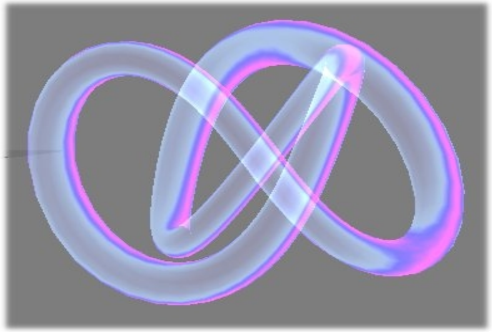
3ds max 4.2

- First DCC tool to support DX8 and HW shaders
- 1122 lines of max code, 41 lines of DX
- Cube Map sample



3ds max 5.1

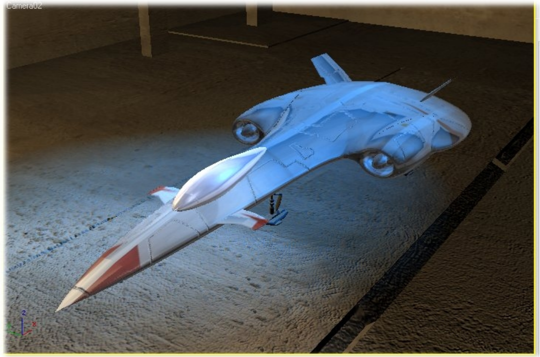
- First DirectX 9 Support
- MetalBump Shader
- DirectX Manager
- 650 line of max code, 41 line of DX



Real time Shaders in 3ds max

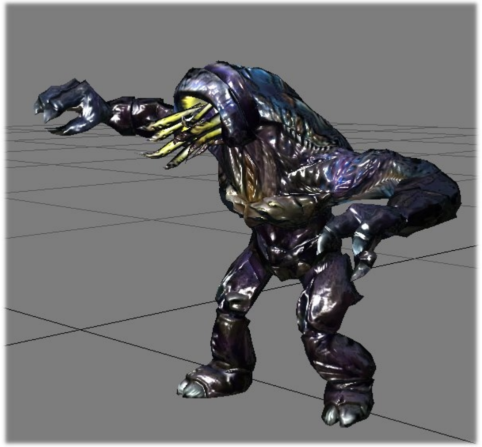
3ds max 6.0

- DirectX Effect file support
- 0 lines of max code, 41 lines of DX



3ds max 7.0

- DirectX Standard Material
- First to support DXSAS
- 0 lines of max code, 0 lines of DX



Support for DirectX Effect Files

- **The DirectX 9 Shader Material supports loading of effect files.**
- **Uses custom Semantics and Annotations to support dynamic UI and custom data binding**
- **Automatically reloads files that change on disk, so can be used as part of the shader development cycle**
- **Full support for 3ds max's mapping channels including Vertex Colour, Alpha and Illumination**
- **Each texture used can have unique mapping coordinate which can be set from the material's UI**
- **Supports a subset of Microsoft's DxSAS version 0.8**
    - No full screen effects
- **Supports custom effect formats via parser DLLs**

Example Effect File Usage

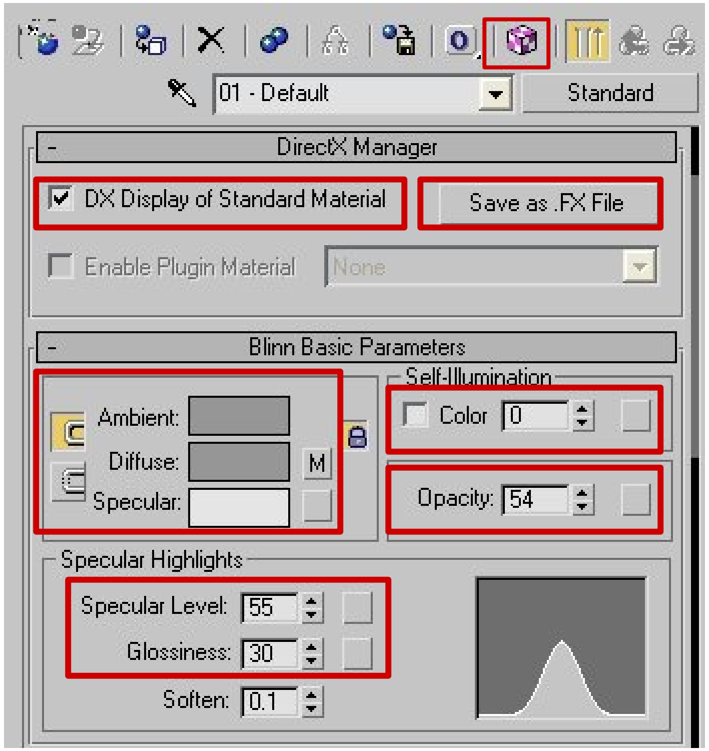
```
float bumpHeight
<
   string UIType = "FloatSpinner";
   string UIName = "Bump Height";
   float UIMin = 0.0f;
   float UIMax = 2.0f;
> = { 1.5};

texture TangentMap : NormalMap
<
   string name = “Default.dds";
   string UIName = "Tangent Space";
   int Texcoord = 0;
   int MapChannel = 1;
>;
```

Key : *Semantic Annotation*

DirectX Standard Material - Overview

- Activate
- Save Effect File
- Dynamic Parameters
- Texture Toggle
- Blinn & Oren-Nayar Blinn lighting models



01 - Default
Standard

☑ DX Display of Standard Material

□ Enable Plugin Material: None

Ambient:
Diffuse:
Specular:

□ Color: 0

Opacity: 54

Specular Level: 55

Glossiness: 30

Soften: 0.1

DirectX Standard Material - Details

Uses ATI's Ashli Technology

- A GPU compiler
- Can convert between HLSL, GLSL and RenderMan
- Compiles to the target platform and language supported
    - In 3ds max's case compiles to PS2.0 and PS2.X
    - Queries the card to find the instruction and texture counts supported
    - Will use Render To Texture when needed.

Builds an HLSL representation of the Standard Material

- Supports dynamic bindings for the most common Params
- Recompile needed for anything else – e.g Changing the shader model

HLSL is compiled by ASHLI into an DX effect

- Effect is then parsed by the default max parser
- Displayed using similar techniques as the DirectX Material

Code is available in the maxsdk

SDK Support for DX Shaders

Extending the DirectX Standard Material

Any texture can "inject" HLSL code
This allows custom code to be used to represent non standard or procedural texture
An example from the SDK is marble.

Texture Maps need to support a new interface called *IHLSLTexmap*

- *bool DoesSupportHLSL();*
- *void GetTexmapHLSLFunction(TCHAR * code, TCHAR * entryPoint);*
- *void GetTextureData(TexDataList * list);*

Extending the DirectX Standard Material

Basic Implementation of GetTexmapHLSLFunction() – full code in marble.cpp

```
if(mapOn[0] && subTex[0])
{
        _tcscat(HLSL, _T("uniform sampler2D MarbleCol1;\n"));
}
//… lots of HLSL
// then copy it into the buffer
_tcscpy(code,HLSL);
_tcscpy(entryPoint,_T("marblefunc")); // used by compiler
```

Note the use of “uniform”. This is required by the compiler for global variables.

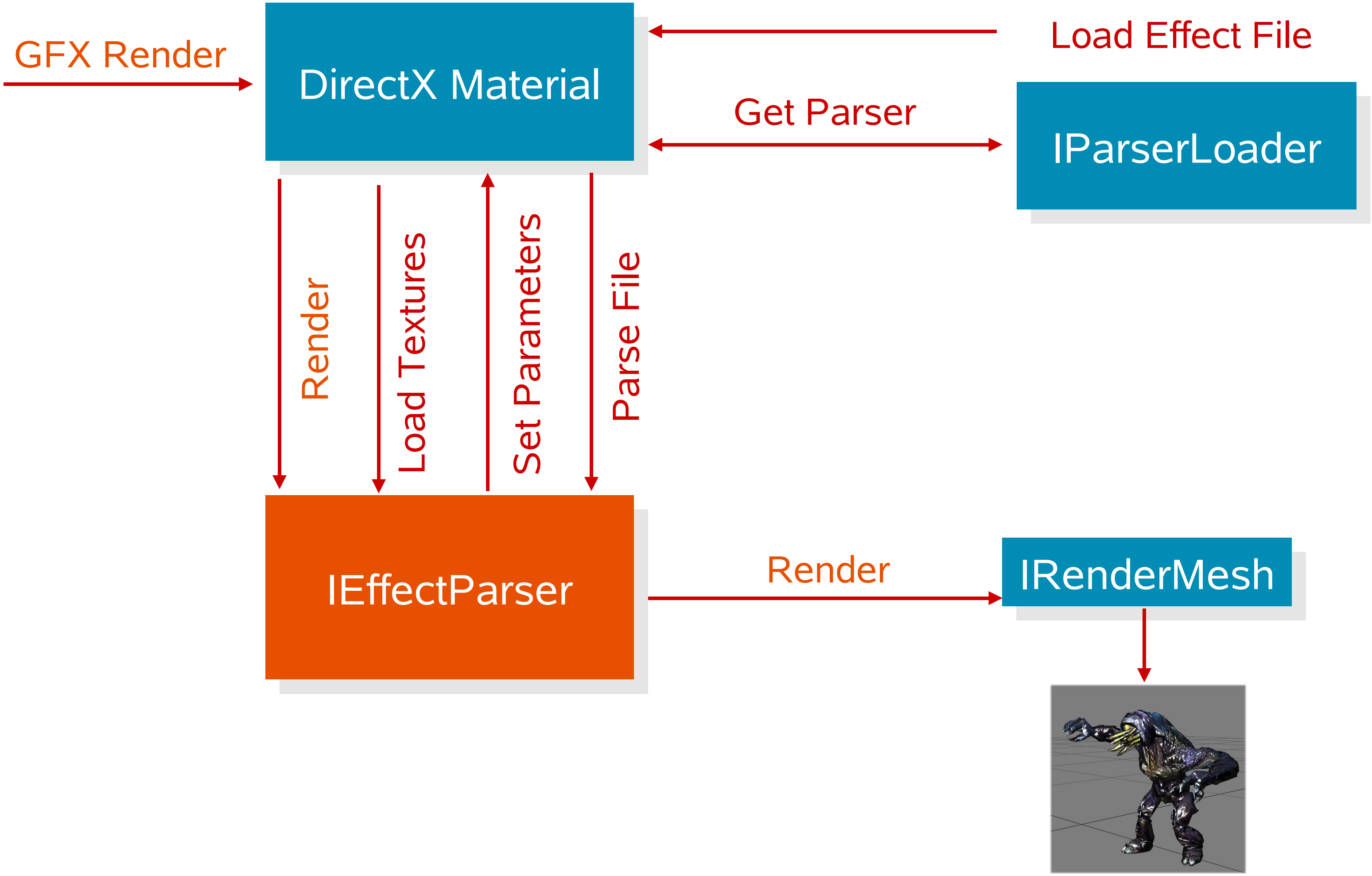
Extending the DirectX Standard Material

Basic Implementation of GetTextureData () – full code in marble.cpp

```
if(mapOn[0]&& subTex[0])
{
   TextureData sample1;
   _tcscpy(sample1.UIName, _T("MarbleCol1"));
   _tcscpy(sample1.SamplerName, _T("MarbleCol1"));
   sample1.SubMapNum = 0;
   list->Append(1, &sample1);
}
```

This information is used to create the UI for the effect file

DirectX Effect System Overview



GFX Render

DirectX Material

Load Effect File

Get Parser

IParserLoader

Render

Load Textures

Set Parameters

Parse File

IEffectParser

Render

IRenderMesh

Extending the DirectX Effect Support

New Effect Parsers can be written by Developers.

New Interfaces IEffectParser and EffectDescriptor are available for use.

- EffectDescriptor:: CreateParser()
- EffectDescriptor::GetParserID()
- IEffectParser::ParseEffectFile()
- IEffectParser::PreRender()
- IEffectParser::Render()

Each parser is a DLL, max ships with 2

- The default max parser and DXSAS 0.8

Effects contains info on what parser to load

- string ParamID = "0x0001"; // DXSAS

Extending the DirectX Effect Support

IEffectManager

- Used to store parameters that need to be set from 3ds max
  - UI Parameters – int, float, Boolean, point4, texture and colour
  - Transform data
  - Provides access to the parameters via Trackview
  - Used to integrate the effect into 3ds max
    - Render to texture
    - Vertex Paint Modifier

- Access to lighting data
  - Position and Direction
  - Colour
  - Hotspot and Falloff

Writing your own DirectX shaders

When you need more than max can deliver, you will need to write your own code.

Two choices, a Material/TextureMap or a DirectX Manager plugin

Cubemap, LightMap and MetalBump are examples of a DirectX Manager plugin
DXStdMtl2 is an example of a full material

There is little between them, so its really personal choice. A DirectX Manager plugin requires the least amount of actual 3ds max code

Writing a DirectX Manager Plugin

Create a basic plugin derived from class ReferenceTarget

IDX9DataBridge & IDX9VertexShader Interfaces

- Provides the hooks into 3ds max
- No need to use IDX9PixelShader anymore
  - Only needed if 3ds max is drawing the object, which limits the shader you can use.
- Use IRenderMesh & IRenderMeshCache helper classes
  - Creates and renders a D3D compliant mesh
  - Creates Normals, UVs andTangent Vectors
- IDX9VertexShader::DrawWireMesh() simply, return true or false
  - Tell 3ds max whether you have drawn the mesh or not.
- IDX9DataBridge::GetDXVersion() – return 9.0

Look at MetalBump and Membrane samples found at Maxsdk\samples\hardwareshaders

Writing a DirectX Hardware Material

Use all the same interfaces as the DirectX Manager plugin, but simply derive from class Mtl instead

- SetMtlFlag(MTL_HW_MAT_ENABLED)
- IMtlRender_Compatibility_MtlBase
  - Allows you to supply a nice colour icon for material browser!
  - Specifies what Software Renderer is supported.

Tips and Tricks

Use IDirect3DStateBlock9 cautiously

- Max can ask an object to be drawn various times, and this can be a resource hog.

Do the actual drawing in the IDX9VerterShader::Initialize() methods

- Simplifies the multi material support

Keep DrawMeshStrips simple – just return True/False

Only support Mesh objects – keeps code simple.

- Use MNMesh::OutToTri()

Use GetCOREInterface()->GetTime(), not TimeValue t = 0

- Prevents unwanted evaluations of the modifier stack

Manage your resources, and RenderStates !!

- 3ds max is easy to upset – typically you loose the viewport names or transparency settings.
    - Typically DESTBLEND and SRCBLEND

3ds max resources

Sparks Support Program

- http://sparks.discreet.com
- Debug builds of max
- Advanced beta exposure
- Knowledge base

Me!!

- neil.hazzard@autodesk.com

Questions

?