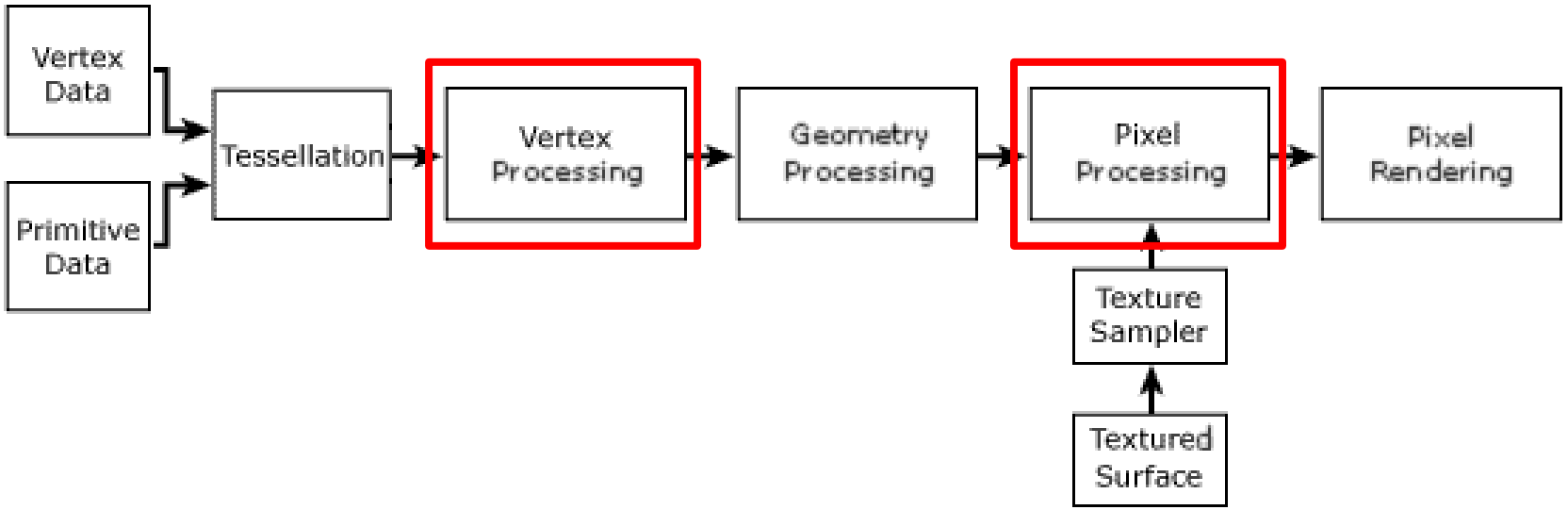
DirectX Programming #4

Kang, Seongtae

Computer Graphics, 2009 Spring

Programmable Shader

- For recent hardwares, vertex and pixel processing stage of the graphics pipeline is programmable
  - Programmable Vertex Shader
  - Programmable Pixel Shader



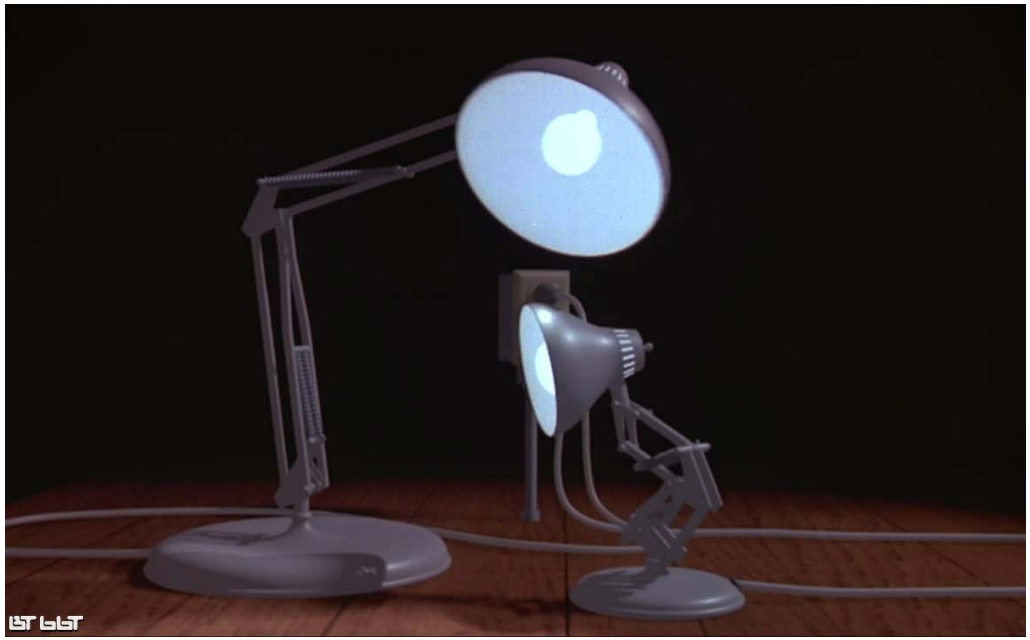
- Geometry processing also programmable since D3D10

Programming for Shader

- In early stage, hardware-dependant mnemonic language is used to program for shader
  - Shader Assembler
- Currently, C-like shading language is widely used
  - Cg, HLSL, GLSL
- General programming platform to utilize GPU's computing power for general purpose is being introduced lately
  - NVIDIA CUDA
  - OpenCL

Brief History on Programmable Shader

- Pixar RenderMan Interface Specification (1988)
  - Concept of Shader Language(SL) is introduced
  - Required special-purpose hardwares and used for studio works



Luxo Jr. (SIGGRAPH, 1986)

Brief History on Programmable Shader

- Programmable shader for consumer hardware
  - Microsoft DirectX 8 (Shader Model 1.1)
  - Shader assembler is introduced for vertex and pixel shader programmability

```
dcl_input v0.xyzw
dcl_input v1.xyz
dcl_input v2.xy
dcl_output_siv  o0.xyzw , position
dcl_output o1.xyzw
dcl_output o2.xy
dcl_constantbuffer  cb0[19], immediateIndexed
dcl_temps 2
dp3 r0.x, v1.xyzx, cb0[11].xyzx
dp3 r0.y, v1.xyzx, cb0[12].xyzx
dp3 r0.z, v1.xyzx, cb0[13].xyzx
dp3 r0.w, r0.xyzx, r0.xyzx
rsq r0.w, r0.w
mul r0.xyz, r0.xyzx, r0.wwww
```

Brief History on Programmable Shader

- NVIDIA Cg (1998)
  - Introduced to program NVIDIA's programmable GPU
  - C-like grammar with reserved keywords

```
// input vertex
struct VertIn {
    float4 pos   : POSITION;
    float4 color : COLOR0;
};

// output vertex
struct VertOut {
    float4 pos   : POSITION;
    float4 color : COLOR0;
};

// vertex shader main entry
VertOut main(VertIn IN, uniform float4x4 modelViewProj) {
    VertOut OUT;
    OUT.pos     = mul(modelViewProj, IN.pos); // calculate output coords
    OUT.color   = IN.color; // copy input color to output
    OUT.color.z = 1.0f; // blue component of color = 1.0f
    return OUT;
}
```

Brief History on Programmable Shader

- Microsoft High-Level Shading Language (HLSL)
  - Introduced with Microsoft DirectX 9 (Shader Model 2)
  - Developed alongside NVIDIA Cg
    - Very similar to Cg

```
struct VS_OUTPUT0 {
     float4 Pos          : POSITION;          // position
     float4 ObjSliceTex : TEXCOORD0;        // Scaled Front slice Texture Coordinate
     float4 OutInterTex      : TEXCOORD2;
};

struct PS_OUTPUT0 {
    float4 Color : COLOR0;
    float Depth  : DEPTH;
};

VS_OUTPUT0 EmptySkipVShader( float4 inPos : POSITION, float4 inSliceTex   : TEXCOORD0) {
    VS_OUTPUT0 Out = (VS_OUTPUT0) 0;
    float4 ttt = inPos + sign(isFrontFace)*faceNormal;
    Out.Pos = mul( ttt, objectTransMatrix );  // transform vertices by objectTransMatrix matrix
    Out.ObjSliceTex = inSliceTex;
    Out.OutInterTex = Out.Pos;
    return Out;
}
```

Shader Model

- Programmability of the shader strongly rely on hardware implementations
- Distinguished by some *Generations*
- No de jure standards for shader model
  - Microsoft's shader model, which inherited NVIDIA model, is de facto standard

Microsoft Shader Model

- Shader Model 1.1
  - Introduced in DirectX 8
  - Shader Assembler
- Shader Model 2
  - DirectX 9.0
  - HLSL introduced
  - No flow control

Microsoft Shader Model

- Shader Model 3
  - DirectX 9.0c
  - identical to SM2 HLSL grammar
  - Dynamic flow control (conditional branch, loop ...)
- Shader Model 4
  - Direct3D 10
  - Renewal of shader language : HLSL4
  - object and template based grammar
  - Programmable geometry processing stage
  - Full flow control and general-purpose features

HLSL on Direct3D 9

- Pass
  - A unit for custom-tailored pipeline stage
  - Only one pass can be executed for one GPU
  - Consists of entry functions for VS/PS, and render states
- Technique
  - Made up of one or more passes

HLSL on Direct3D 9

- Effect
  - Contains one or more techniques
  - Global variables and type definitions
  - Format
    - Text source
    - Binary object
      - You can compile an effect source file to the binary object on the command line using fx compiler tool (fxc.exe)
  - Location
    - External file (usually .fx for source, .obj for binary)
    - Included resource

Structure of a HLSL Effect File

- Variable Declaration
- Type Declaration
- Functions
- Technique and pass description

HLSL : Variable Declaration

- Global variables for the effect
- Constants
- These variables can be assigned from CPU-side

```
float4x4 g_mWorld;
float4x4 g_mView;
float4x4 g_mProj;

float4 g_vLight      = float4( 0.0f, 0.0f, -10.0f, 1.0f );
float4 g_vLightColor = float4( 1.0f, 1.0f, 1.0f, 1.0f );

texture  g_txScene;

float4   Diffuse;
```

HLSL : Type Declaration

- Custom structures
- Render state description set
- Texture sampler

```
struct VS_OUTPUT0
{
    float4 Pos : POSITION;
    float4 Diffuse : COLOR0;
    float2 Tex0 : TEXCOORD0;
};

sampler2D g_samScene = sampler_state
{
    Texture = <g_txScene>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = Linear;
};
```

HLSL : Type Declaration

- Custom structures
- Render state description set
- Texture sampler

```
struct VS_OUTPUT0
{
    float4 Pos : POSITION;
    float4 Diffuse : COLOR0;
    float2 Tex0 : TEXCOORD0;
};

sampler2D g_samScene = sampler_state
{
    Texture = <g_txScene>;
    MinFilter = Linear;
    MagFilter = Linear;
    MipFilter = Linear;
};
```

HLSL : Functions

- Limitations for VS and PS function are not identical!
  - Until SM 3

```
VS_OUTPUT0 VertScene( float4 vPos : POSITION, float3 vNormal : NORMAL, float2 vTex0 : TEXCOORD0)
{
    VS_OUTPUT out;
    out.Pos = mul( vPos, g_mWorld );
    out.Pos = mul( out.Pos, g_mView );
    out.Pos = mul( out.Pos, g_mProj );

    float4 wPos = mul( vPos, g_mWorld );
    wPos = mul( wPos, g_mView );

    float3 N =  mul( vNormal, (float3x3)g_mWorld );
    N = normalize( mul( N, (float3x3)g_mView ) );

    float3 InvL = g_vLight - wPos;
    InvL = normalize( InvL );

    oout.Diffuse = saturate( dot( N, InvL ) ) * Diffuse * g_vLightColor;
    out.Tex0 = vTex0;

    return out;
}

float4 PixScene( float4 Diffuse : COLOR0, float2 Tex0 : TEXCOORD0 ) : COLOR0
{
    return tex2D( g_samScene, Tex0 ) * Diffuse;
}
```

HLSL : Technique and Pass Description

- NULL for VertexShader/PixelShader means fixed pipeline behavior

```
technique MyRenderTech
{
    pass P0
    {
        VertexShader = compile vs_2_0 VertScene();
        PixelShader  = compile ps_2_0 PixScene();
        CullMode = NONE;
        AlphaBlendEnable = FALSE;
    }
}
```

Creating an Effect

- From external file
  - Either a source text or compiled binary can be used

```
HRESULT D3DXCreateEffectFromFile(LPDIRECT3DDEVICE9 pDevice,  LPCTSTR pSrcFile,
                                 CONST D3DXMACRO * pDefines,  LPD3DXINCLUDE pInclude,
                                 DWORD Flags, LPD3DXEFFECTPOOL pPool,
                                 LPD3DXEFFECT *ppEffect,  LPD3DXBUFFER *ppCompilationErrors);
```

- pSrcFile : effect file name
- Flags : several flags such as compilation options
- ppEffect : a pointer to a buffer containing the compiled effect

```
ID3DXEffect* m_pEffect;
D3DXCreateEffectFromFile( pd3dDevice, str, NULL, NULL, NULL, NULL, &m_pEffect, NULL );
```

Binding Variables

- Set<Type> method series of effect class
- Primitive types are passed by value
- Struct types are passed by pointer
- Textures are bound by texture object pointer

```
D3DXMATRIX matWorld, matView, matView;
IDirect3DTexture9 *myTex;
                              :
                              :
m_pEffect->SetMatrix("g_mWorld", &matWorld);  // struct type
m_pEffect->SetMatrix("g_mView", &matView);
m_pEffect->SetFloat("Diffuse", 0.6f);                  // primitive type
m_pEffect->SetTexture("g_txScene", myTex);       // texture
```

Using Effects on Rendering

- Begin an effect
  - Set a technique
  - Begin a pass
    - Render
  - End a pass
- End an effect

```
m_pEffect->Begin(&nPasses, 0 ); // # of passes for this technique will be returned to nPasses
int nTechnique=pTerrainPainter->GetTechniqueByName(”MyRenderTech”);
pTerrainPainter->SetTechnique(nTechnique);       // using ”MyRenderTech” technique
m_pEffect->BeginPass(0);           // using pass 0
m_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLESTRIP, 0, lineCnt); // rendering code
m_pEffect->EndPass();
m_pEffect->End();                   // Begin’s 2nd param 0 triggers restoration
                                    // of previous render state when an effect ends
```

Next Class

- @ 301-314
- Practice on programming DirectX and HLSL application using Visual C++ on Win32 platform
- Basic sample code will be supplied
  - Direct3D initialization
  - Vertex buffer creation
  - Drawing some polygon
- We will code in next class
  - Effect creation and binding
  - Effect file coding in HLSL
    - Basic fixed pipeline
    - Phong shading
- DirectX homework will be assigned

Practice Assignments

- Prepare for HLSL coding
  - See HLSL contents on the DirectX document
- Analyse the sample code
  - Will be uploaded on homepage this week