GPU Programming for Video Games

Introduction to Surface Shaders

Prof. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Georgia Institute of Technology

Unity 5's rendering paths

- Deferred Shading: `Deferred` pass
- Forward Rendering: `ForwardBase` and `ForwardAdd` passes
- Legacy Deferred Lighting: `PrepassBase` and `PrepassFinal` passes
- Legacy Vertex Lit
- "In any of the above, to render Shadows or a depth texture, `ShadowCaster` pass is used."
  - No more `ShadowCollector` pass

From http://docs.unity3d.com/Manual/SL-RenderPipeline.html

Forward rendering

- "In Forward Rendering, some number of brightest lights that affect each object are rendered in fully per-pixel lit mode.
- Then, up to 4 point lights are calculated per-vertex.
- The other lights are computed as Spherical Harmonics (SH), which is much faster but is only an approximation."

From http://docs.unity3d.com/Manual/RenderTech-ForwardRendering.html

Which lights get what treatment?

- "Lights that have their Render Mode set to **Not Important** are always per-vertex or SH.
- Brightest directional light is always per-pixel.
- Lights that have their Render Mode set to **Important** are always per-pixel.
- If the above results in less lights than current Pixel Light Count Quality Setting, then more lights are rendered per-pixel, in order of decreasing brightness."

From http://docs.unity3d.com/Manual/RenderTech-ForwardRendering.html

Surface shaders – the Why

- "Writing shaders that interact with lighting is complex. There are
  - different light types
  - different shadow options
  - different rendering paths (forward and deferred rendering)
- and the shader should somehow handle all that complexity."

From http://docs.unity3d.com/Manual/SL-SurfaceShaders.html

Surface shaders – the How

- "Surface Shaders in Unity is a code generation approach that makes it much easier to write lit shaders than using low level vertex/pixel shader programs.
- ...there are no custom languages, magic or ninjas involved in Surface Shaders; it just generates all the repetitive code that would have to be written by hand.
- You still write shader code in Cg / HLSL."

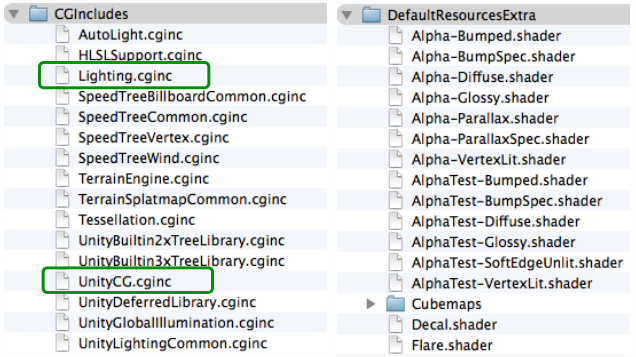
From http://docs.unity3d.com/Manual/SL-SurfaceShaders.html

Unity's built-in shaders

- Source available separately:
  http://unity3d.com/unity/download/archive

Normal-Bumped.shader	Reflect-Bumped.shader
Normal-BumpSpec.shader	Reflect-BumpNolight.shader
Normal-Diffuse.shader	Reflect-BumpSpec.shader
Normal-DiffuseDetail.shader	Reflect-BumpVertexLit.shader
Normal-DiffuseFast.shader	Reflect-Diffuse.shader
Normal-Glossy.shader	Reflect-Glossy.shader
Normal-Parallax.shader	Reflect-Parallax.shader
Normal-ParallaxSpec.shader	Reflect-ParallaxSpec.shader
Normal-VertexLit.shader	Reflect-VertexLit.shader
Particle Add.shader	Skybox-Cubed.shader
Particle AddMultiply.shader	Skybox-Procedural.shader

Dig through the files



Numeric data types

- "float: high precision floating point. Generally 32 bits, just like float type in regular programming languages.
- half: medium precision floating point. Generally 16 bits, with a range of –60000 to +60000 and 3.3 decimal digits of precision.
- fixed: low precision fixed point. Generally 11 bits, with a range of –2.0 to +2.0 and 1/256th precision."

From http://docs.unity3d.com/Manual/SL-ShaderPerformance.html

Standard appdata structs (1)

```
struct appdata_base {
  float4 vertex : POSITION;
  float3 normal : NORMAL;
  float4 texcoord : TEXCOORD0;
};

struct appdata_tan {
  float4 vertex : POSITION;
  float4 tangent : TANGENT;
  float3 normal : NORMAL;
  float4 texcoord : TEXCOORD0;
};
```

In UnityCG.cginc

Standard appdata structs (2)

```
struct appdata_full {
  float4 vertex : POSITION;
  float4 tangent : TANGENT;
  float3 normal : NORMAL;
  float4 texcoord : TEXCOORD0;
  float4 texcoord1 : TEXCOORD1;
  float4 texcoord2 : TEXCOORD2;
  float4 texcoord3 : TEXCOORD3;
#if defined(SHADER_API_XBOX360)
  half4 texcoord4 : TEXCOORD4;
  half4 texcoord5 : TEXCOORD5;
#endif
  fixed4 color : COLOR;
};
```

In UnityCG.cginc

SurfaceOutput structure

- Your job is to fill this in, as necessary:

```
struct SurfaceOutput {
    fixed3 Albedo; // diffuse color
    fixed3 Normal; // tangent space normal,
                   // if written
    fixed3 Emission;
    half Specular; // specular power in
                   // 0..1 range
    fixed Gloss;   // specular intensity
    fixed Alpha;   // alpha for
                   // transparencies
};
```

From http://docs.unity3d.com/Manual/SL-SurfaceShaders.html

Lambert lighting (Unity 4)

- In Lighting.cginc (Unity 4):

```
inline fixed4 LightingLambert (SurfaceOutput s,
                               fixed3 lightDir,
                               fixed atten) {
  fixed diff = max (0, dot (s.Normal, lightDir));

  fixed4 c;
  c.rgb = s.Albedo * _LightColor0.rgb *
                               (diff * atten * 2);
  c.a = s.Alpha;
  return c;
}
```

The multiplication by 2 is a weird Unity 4 thing; not present in Unity 5

Diffuse surface shader (1)

```
Shader "Legacy Shaders/Diffuse" {
Properties {
  _Color ("Main Color", Color) = (1,1,1,1)
  _MainTex ("Base (RGB)", 2D) = "white" {}
}
SubShader {
  Tags { "RenderType"="Opaque" }
  LOD 200

CGPROGRAM
#pragma surface surf Lambert

sampler2D _MainTex;
fixed4 _Color;
```

From Normal-Diffuse.shader

Diffuse surface shader (2)

```
struct Input {
  float2 uv_MainTex;
};

void surf (Input IN, inout SurfaceOutput o) {
  fixed4 c =
      tex2D(_MainTex, IN.uv_MainTex) * _Color;
  o.Albedo = c.rgb;
  o.Alpha = c.a;
}
ENDCG
}

Fallback "Legacy Shaders/VertexLit"
}
```

From Normal-Diffuse.shader

Shade4PointLights (1)

```
float3 Shade4PointLights (
        float4 lightPosX, float4 lightPosY, float4 lightPosZ,
        float3 lightColor0, float3 lightColor1,
        float3 lightColor2, float3 lightColor3,
        float4 lightAttenSq, float3 pos, float3 normal) {
  // to light vectors
  float4 toLightX = lightPosX - pos.x;
  float4 toLightY = lightPosY - pos.y;
  float4 toLightZ = lightPosZ - pos.z;
  // squared lengths
  float4 lengthSq = 0;
  lengthSq += toLightX * toLightX;
  lengthSq += toLightY * toLightY;
  lengthSq += toLightZ * toLightZ;
  // NdotL
  float4 ndotl = 0;
  ndotl += toLightX * normal.x;
  ndotl += toLightY * normal.y;
  ndotl += toLightZ * normal.z;
```

In UnityCG.cginc

Shade4PointLights (2)

```
// correct NdotL
float4 corr = rsqrt(lengthSq);
ndotl = max (float4(0,0,0,0), ndotl * corr);
// attenuation
float4 atten =
     1.0 / (1.0 + lengthSq * lightAttenSq);
float4 diff = ndotl * atten;
// final color
float3 col = 0;
col += lightColor0 * diff.x;
col += lightColor1 * diff.y;
col += lightColor2 * diff.z;
col += lightColor3 * diff.w;
return col;
}
```

In UnityCG.cginc

Blinn-Phong lighting (Unity 4)

```
inline fixed4 LightingBlinnPhong (SurfaceOutput s,
          fixed3 lightDir, half3 viewDir, fixed atten) {
  half3 h = normalize (lightDir + viewDir);

  fixed diff = max (0, dot (s.Normal, lightDir));

  float nh = max (0, dot (s.Normal, h));
  float spec = pow (nh, s.Specular*128.0) * s.Gloss;

  fixed4 c;
  c.rgb = (s.Albedo * _LightColor0.rgb * diff
           + _LightColor0.rgb * _SpecColor.rgb * spec)
          * (atten * 2);
  c.a = s.Alpha +
           _LightColor0.a * _SpecColor.a * spec * atten;
  return c;
}
```

The multiplication by 2 is a weird Unity 4 thing, as is the addition to s.Alpha; not present in Unity 5

In Lighting.cginc

Specular surface shader (1)

```
Shader "Legacy Shaders/Specular" {
Properties {
  _Color ("Main Color", Color) = (1,1,1,1)
  _SpecColor ("Specular Color", Color) = (0.5, 0.5, 0.5, 1)
  _Shininess ("Shininess", Range (0.01, 1)) = 0.078125
  _MainTex ("Base (RGB) Gloss (A)", 2D) = "white" {}
}

SubShader {
  Tags { "RenderType"="Opaque" }
  LOD 300

CGPROGRAM
#pragma surface surf BlinnPhong

sampler2D _MainTex;
fixed4 _Color;
half _Shininess;
```

From Normal-Glossy.shader

Specular surface shader (2)

```
struct Input {
  float2 uv_MainTex;
};

void surf (Input IN, inout SurfaceOutput o) {
  fixed4 tex = tex2D(_MainTex, IN.uv_MainTex);
  o.Albedo = tex.rgb * _Color.rgb;
  o.Gloss = tex.a;
  o.Alpha = tex.a * _Color.a;
  o.Specular = _Shininess;
}
ENDCG
}

Fallback "Legacy Shaders/VertexLit"
}
```

etaRatio=

From Normal-Glossy.shader

Bumped specular surface shader (1)

```
Shader "Legacy Shaders/Bumped Specular" {
Properties {
  _Color ("Main Color", Color) = (1,1,1,1)
  _SpecColor ("Specular Color", Color) = (0.5, 0.5, 0.5, 1)
  _Shininess ("Shininess", Range (0.03, 1)) = 0.078125
  _MainTex ("Base (RGB) Gloss (A)", 2D) = "white" {}
  _BumpMap ("Normalmap", 2D) = "bump" {}
}
```

From Normal-BumpSpec.shader

Bumped specular surface shader (2)

```
CGINCLUDE
sampler2D _MainTex;
sampler2D _BumpMap;
fixed4 _Color;
half _Shininess;

struct Input {
  float2 uv_MainTex;
  float2 uv_BumpMap;
};

void surf (Input IN, inout SurfaceOutput o) {
  fixed4 tex = tex2D(_MainTex, IN.uv_MainTex);
  o.Albedo = tex.rgb * _Color.rgb;
  o.Gloss = tex.a;
  o.Alpha = tex.a * _Color.a;
  o.Specular = _Shininess;
  o.Normal = UnpackNormal(tex2D(_BumpMap, IN.uv_BumpMap));
}
ENDCG
```

From Normal-BumpSpec.shader

Bumped specular surface shader (3)

```
SubShader {
  Tags { "RenderType"="Opaque" }
  LOD 400

  CGPROGRAM
  #pragma surface surf BlinnPhong
  #pragma target 3.0
  ENDCG
}

SubShader {
  Tags { "RenderType"="Opaque" }
  LOD 400

  CGPROGRAM
  #pragma surface surf BlinnPhong nodynlightmap
  ENDCG
}

FallBack "Legacy Shaders/Specular"
}
```

From Normal-BumpSpec.shader

UnpackNormal

```
inline fixed3 UnpackNormalDXT5nm (fixed4 packednormal) {
  fixed3 normal;
  normal.xy = packednormal.wy * 2 - 1;
  normal.z = sqrt(1 - saturate(dot(normal.xy, normal.xy)));
  return normal;
}

inline fixed3 UnpackNormal(fixed4 packednormal) {
#if defined(UNITY_NO_DXT5nm)
  return packednormal.xyz * 2 - 1;
#else
  return UnpackNormalDXT5nm(packednormal);
#endif
}
```

In UnityCG.cginc

TANGENT_SPACE_ROTATION

- In UnityCG.cginc:

```
// Declares 3x3 matrix 'rotation' filled with tangent space basis
#define TANGENT_SPACE_ROTATION \
  float3 binormal = cross( v.normal, v.tangent.xyz ) * v.tangent.w; \
  float3x3 rotation = float3x3( v.tangent.xyz, binormal, v.normal )
```

Surface shader input structure (1)

"The input structure Input generally has any texture coordinates needed by the shader. Texture coordinates must be named “uv” followed by texture name (or start it with “uv2” to use second texture coordinate set).

Additional values that can be put into Input structure:

- float3 viewDir - will contain view direction, for computing Parallax effects, rim lighting etc.
- float4 with COLOR semantic - will contain interpolated per-vertex color.
- float4 screenPos - will contain screen space position for reflection effects. Used by WetStreet shader in Dark Unity for example.
- float3 worldPos - will contain world space position."

From http://docs.unity3d.com/Manual/SL-SurfaceShaders.html

Surface shader input structure (2)

- "float3 worldRefl - will contain world reflection vector *if surface shader does not write to o.Normal*. See Reflect-Diffuse shader for example.
- float3 worldNormal - will contain world normal vector *if surface shader does not write to o.Normal*.
- float3 worldRefl; INTERNAL_DATA - will contain world reflection vector *if surface shader writes to o.Normal*. To get the reflection vector based on per-pixel normal map, use WorldReflectionVector (IN, o.Normal). See Reflect-Bumped shader for example.
- float3 worldNormal; INTERNAL_DATA - will contain world normal vector *if surface shader writes to o.Normal*. To get the normal vector based on per-pixel normal map, use WorldNormalVector (IN, o.Normal)."

From http://docs.unity3d.com/Manual/SL-SurfaceShaders.html

Optional surface shader parameters

- Use #**pragma** surface surfaceFunction lightModel [optionalparams]
- vertex:VertexFunction - Custom vertex modification function. This function is invoked at start of generated vertex shader, and can modify or compute per-vertex data.
- finalcolor:ColorFunction - Custom final color modification function.
  addshadow - Add a shadow caster pass. Commonly used with custom vertex modification, so that shadow casting also gets any procedural vertex animation.
- decal:add - Additive decal shader (e.g. terrain AddPass).
- noambient - Do not apply any ambient lighting or light probes."

From http://docs.unity3d.com/Manual/SL-SurfaceShaders.html