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's rendering paths

- Deferred Lighting: PrepassBase and PrepassFinal passes (Unity Pro only!)
- Forward Rendering: ForwardBase and ForwardAdd passes
- Vertex Lit (won't go into here)
- "In any of the above, to render Shadows, ShadowCaster and ShadowCollector passes are used."

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

What happens on each pass?

- "Base pass renders object with one per-pixel directional light and all SH lights. This pass also adds any lightmaps, ambient and emissive lighting from the shader. Directional light rendered in this pass can have Shadows. Note that Lightmapped objects do not get illumination from SH lights.

- Additional passes are rendered for each additional per-pixel light that affect this object. Lights in these passes can't have shadows (so in result, Forward Rendering supports one directional light with shadows)."

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

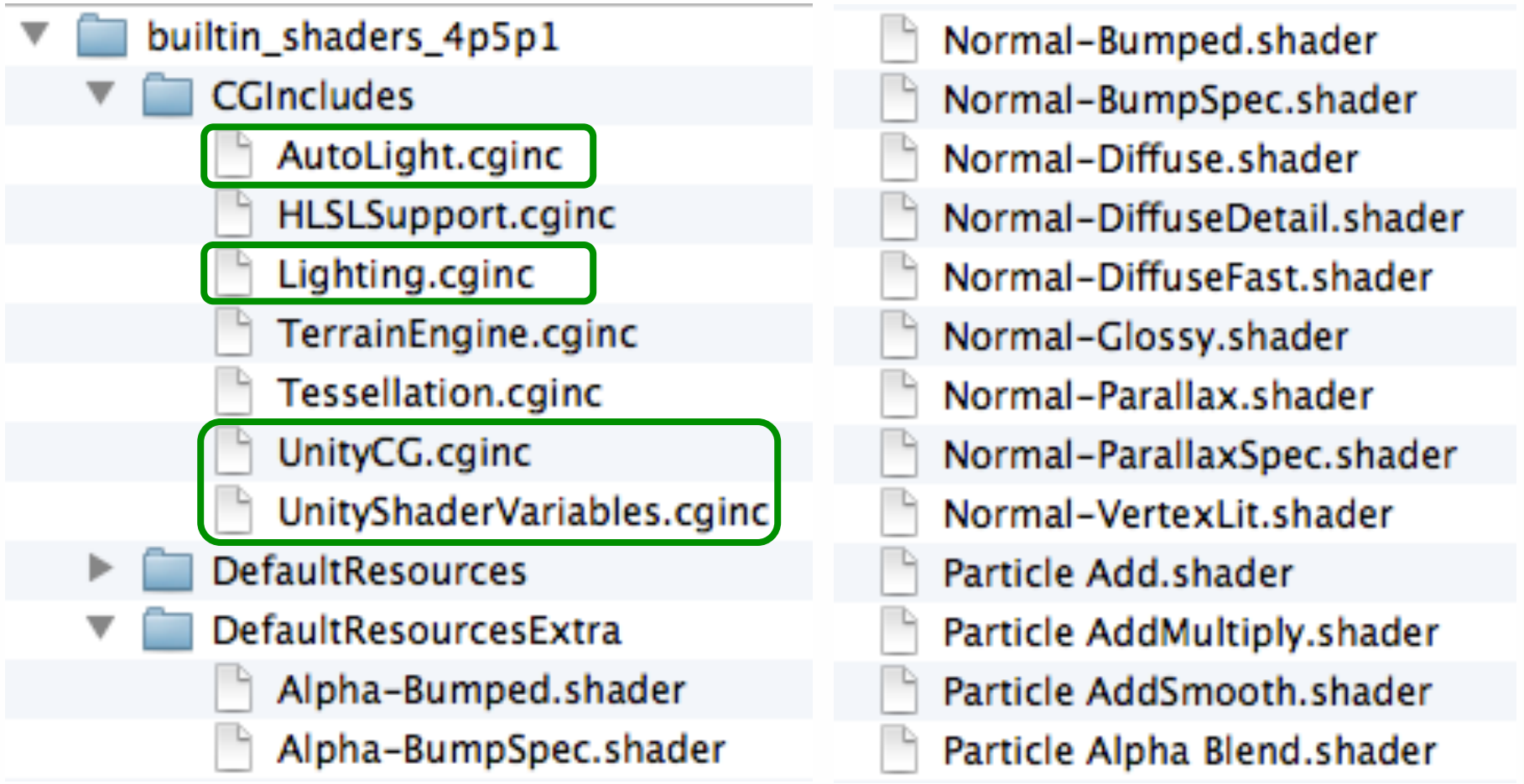
Unity's built-in shaders

- Source available separately:

  http://unity3d.com/unity/download/archive

Version	Installer	Release notes	Built-in shaders
Unity 4.5.1	Win / Mac	View	Download
Unity 4.5.0	Win / Mac	View	Download
Unity 4.3.4	Win / Mac	View	Download
Unity 4.3.3	Win / Mac	View	Download

Dig through the includes



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;
    fixed4 color : COLOR;
#if defined(SHADER_API_XBOX360)
    half4 texcoord2 : TEXCOORD2;
    half4 texcoord3 : TEXCOORD3;
    half4 texcoord4 : TEXCOORD4;
    half4 texcoord5 : TEXCOORD5;
#endif
};
```

In UnityCG.cginc

SurfaceOutput structure

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

```
struct SurfaceOutput {
    fixed3 Albedo;
    fixed3 Normal;
    fixed3 Emission;
    half Specular;
    fixed Gloss;
    fixed Alpha;
};
```

Lambert lighting

- In Lighting.cginc:

```
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;
}
```

Diffuse surface shader (1)

```
Shader "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 "VertexLit"
}
```

From Normal-Diffuse.shader

Base pass setup

```
Pass {
  Name "FORWARD"
  Tags { "LightMode" = "ForwardBase" }

CGPROGRAM
// compile directives
#pragma vertex vert_surf
#pragma fragment frag_surf
#pragma multi_compile_fwdbase
#include "HLSLSupport.cginc"
#include "UnityShaderVariables.cginc"
#define UNITY_PASS_FORWARDBASE
#include "UnityCG.cginc"
#include "Lighting.cginc"
#include "AutoLight.cginc"

#define INTERNAL_DATA
#define WorldReflectionVector(data,normal) data.worldRefl
#define WorldNormalVector(data,normal) normal
```

Base pass structures

```
// vertex-to-fragment interpolation data
#ifdef LIGHTMAP_OFF
struct v2f_surf {
  float4 pos : SV_POSITION;
  float2 pack0 : TEXCOORD0;
  fixed3 normal : TEXCOORD1;
  fixed3 vlight : TEXCOORD2;
  LIGHTING_COORDS(3,4)
};
#endif
#ifndef LIGHTMAP_OFF
struct v2f_surf {
  float4 pos : SV_POSITION;
  float2 pack0 : TEXCOORD0;
  float2 lmap : TEXCOORD1;
  LIGHTING_COORDS(2,3)
};
#endif
#ifndef LIGHTMAP_OFF
float4 unity_LightmapST;
#endif
float4 _MainTex_ST;
```

Base pass vertex program (1)

```
// vertex shader
v2f_surf vert_surf (appdata_full v) {
  v2f_surf o;
  o.pos = mul (UNITY_MATRIX_MVP, v.vertex);
  o.pack0.xy = TRANSFORM_TEX(v.texcoord, _MainTex);
  #ifndef LIGHTMAP_OFF
  o.lmap.xy = v.texcoord1.xy * unity_LightmapST.xy
                                + unity_LightmapST.zw;
  #endif
  float3 worldN =
            mul((float3x3)_Object2World, SCALED_NORMAL);
  #ifdef LIGHTMAP_OFF
  o.normal = worldN;
  #endif// vertex-to-fragment interpolation data
```

Base pass vertex program (2)

```
  // SH/ambient and vertex lights
  #ifdef LIGHTMAP_OFF
  float3 shlight = ShadeSH9 (float4(worldN,1.0));
  o.vlight = shlight;
  #ifdef VERTEXLIGHT_ON
  float3 worldPos = mul(_Object2World, v.vertex).xyz;
  o.vlight += Shade4PointLights (
    unity_4LightPosX0, unity_4LightPosY0, unity_4LightPosZ0,
    unity_LightColor[0].rgb, unity_LightColor[1].rgb,
    unity_LightColor[2].rgb, unity_LightColor[3].rgb,
    unity_4LightAtten0, worldPos, worldN );
  #endif // VERTEXLIGHT_ON
  #endif // LIGHTMAP_OFF

 // pass lighting information to pixel shader
  TRANSFER_VERTEX_TO_FRAGMENT(o);
  return o;
}
```

Shade4PointLights??? (1)

```
float3 Shade4PointLights (
          float4 lightPosX, float4 lightPosY, float4 lightPosZ,
          float3 lightColor0, float3 lightColor1,
          float3 lghtColor2, 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

Base pass lightmap variables

```
#ifndef LIGHTMAP_OFF
sampler2D unity_Lightmap;
#ifndef DIRLIGHTMAP_OFF
sampler2D unity_LightmapInd;
#endif
#endif
```

Base pass fragment program (1)

```
// fragment shader
fixed4 frag_surf (v2f_surf IN) : SV_Target {
  // prepare and unpack data
  #ifdef UNITY_COMPILER_HLSL
  Input surfIN = (Input)0;
  #else
  Input surfIN;
  #endif
  surfIN.uv_MainTex = IN.pack0.xy;
  #ifdef UNITY_COMPILER_HLSL
  SurfaceOutput o = (SurfaceOutput)0;
  #else
  SurfaceOutput o;
  #endif
  o.Albedo = 0.0;
  o.Emission = 0.0;
  o.Specular = 0.0;
  o.Alpha = 0.0;
  o.Gloss = 0.0;
  #ifdef LIGHTMAP_OFF
  o.Normal = IN.normal;
  #endif
```

Base pass fragment program (2)

```
// call surface function
surf (surfIN, o);

// compute lighting & shadowing factor
fixed atten = LIGHT_ATTENUATION(IN);
fixed4 c = 0;

// realtime lighting: call lighting function
#ifdef LIGHTMAP_OFF
c =
 LightingLambert (o, _WorldSpaceLightPos0.xyz, atten);
#endif // LIGHTMAP_OFF || DIRLIGHTMAP_OFF
#ifdef LIGHTMAP_OFF
c.rgb += o.Albedo * IN.vlight;
#endif // LIGHTMAP_OFF
```

Base pass fragment program (3)

```
// lightmaps:
 #ifndef LIGHTMAP_OFF
   #ifndef DIRLIGHTMAP_OFF
     // directional lightmaps
     fixed4 lmtex = tex2D(unity_Lightmap, IN.lmap.xy);
     fixed4 lmIndTex =
                  tex2D(unity_LightmapInd, IN.lmap.xy);
     half3 lm = LightingLambert_DirLightmap(o, lmtex,
                                            lmIndTex, 0).rgb;
   #else // !DIRLIGHTMAP_OFF
     // single lightmap
     fixed4 lmtex = tex2D(unity_Lightmap, IN.lmap.xy);
     fixed3 lm = DecodeLightmap (lmtex);
   #endif // !DIRLIGHTMAP_OFF
```

Base pass fragment program (4)

```
    // combine lightmaps with realtime shadows
    #ifdef SHADOWS_SCREEN
      #if (defined(SHADER_API_GLES) ||
           defined(SHADER_API_GLES3)) &&
                 defined(SHADER_API_MOBILE)
      c.rgb += o.Albedo * min(lm, atten*2);
      #else
      c.rgb += o.Albedo *
             max(min(lm,(atten*2)*lmtex.rgb), lm*atten);
      #endif
    #else // SHADOWS_SCREEN
      c.rgb += o.Albedo * lm;
    #endif // SHADOWS_SCREEN
  c.a = o.Alpha;
  #endif // LIGHTMAP_OFF

  return c;
}
```

Add pass setup

```
Pass {
  Name "FORWARD"
  Tags { "LightMode" = "ForwardAdd" }
  ZWrite Off Blend One One Fog { Color (0,0,0,0) }

CGPROGRAM
// compile directives
#pragma vertex vert_surf
#pragma fragment frag_surf
#pragma multi_compile_fwdadd
#include "HLSLSupport.cginc"
#include "UnityShaderVariables.cginc"
#define UNITY_PASS_FORWARDADD
#include "UnityCG.cginc"
#include "Lighting.cginc"
#include "AutoLight.cginc"

#define INTERNAL_DATA
#define WorldReflectionVector(data,normal) data.worldRefl
#define WorldNormalVector(data,normal) normal
```

Add pass structures

```
// vertex-to-fragment interpolation data
struct v2f_surf {
  float4 pos : SV_POSITION;
  float2 pack0 : TEXCOORD0;
  fixed3 normal : TEXCOORD1;
  half3 lightDir : TEXCOORD2;
  LIGHTING_COORDS(3,4)
};
float4 _MainTex_ST;
```

Add pass vertex program

```
// vertex shader
v2f_surf vert_surf (appdata_full v) {
  v2f_surf o;
  o.pos = mul (UNITY_MATRIX_MVP, v.vertex);
  o.pack0.xy = TRANSFORM_TEX(v.texcoord, _MainTex);
  o.normal =
            mul((float3x3)_Object2World, SCALED_NORMAL);
  float3 lightDir = WorldSpaceLightDir( v.vertex );
  o.lightDir = lightDir;

  // pass lighting information to pixel shader
  TRANSFER_VERTEX_TO_FRAGMENT(o);
  return o;
}
```

Add pass fragment program (1)

```
// fragment shader
fixed4 frag_surf (v2f_surf IN) : SV_Target {
  // prepare and unpack data
  #ifdef UNITY_COMPILER_HLSL
  Input surfIN = (Input)0;
  #else
  Input surfIN;
  #endif
  surfIN.uv_MainTex = IN.pack0.xy;
  #ifdef UNITY_COMPILER_HLSL
  SurfaceOutput o = (SurfaceOutput)0;
  #else
  SurfaceOutput o;
  #endif
  o.Albedo = 0.0;
  o.Emission = 0.0;
  o.Specular = 0.0;
  o.Alpha = 0.0;
  o.Gloss = 0.0;
  o.Normal = IN.normal;
```

Add pass fragment program (2)

```
 // call surface function
   surf (surfIN, o);
   #ifndef USING_DIRECTIONAL_LIGHT
   fixed3 lightDir = normalize(IN.lightDir);
   #else
   fixed3 lightDir = IN.lightDir;
   #endif
   fixed4 c = LightingLambert (o, lightDir,
                               LIGHT_ATTENUATION(IN));
   c.a = 0.0;
   return c;
}
```

Blinn-Phong lighting

```
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;
}
```

In Lighting.cginc

In Lighting.cginc

Specular surface shader (1)

```
Shader "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-Gloss.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 "VertexLit"
}
```

etaRatio=

From Normal-Gloss.shader

Bumped specular surface shader (1)

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

CGPROGRAM
#pragma surface surf BlinnPhong

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

From Normal-BumpSpec.shader

Bumped specular surface shader (2)

```
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
}

FallBack "Specular"
}
```

From Normal-BumpSpec.shader

UnpackNormal???

```
inline fixed3 UnpackNormalDXT5nm (fixed4 packednormal) {
  fixed3 normal;
  normal.xy = packednormal.wy * 2 - 1;
#if defined(SHADER_API_FLASH)
  // Flash does not have efficient saturate(),
  // and dot() seems to require an extra register.
  normal.z = sqrt(1 - normal.x*normal.x - normal.y * normal.y);
#else
  normal.z = sqrt(1 - saturate(dot(normal.xy, normal.xy)));
#endif
  return normal;
}

inline fixed3 UnpackNormal(fixed4 packednormal) {
#if (defined(SHADER_API_GLES) || defined(SHADER_API_GLES3)) &&
                                    defined(SHADER_API_MOBILE)
  return packednormal.xyz * 2 - 1;
#else
  return UnpackNormalDXT5nm(packednormal);
#endif
}
```

In UnityCG.cginc

Bumped specular base pass vertex shader

```
// vertex shader
v2f_surf vert_surf (appdata_full v) {
  v2f_surf o;
  o.pos = mul (UNITY_MATRIX_MVP, v.vertex);
  o.pack0.xy = TRANSFORM_TEX(v.texcoord, _MainTex);
  o.pack0.zw = TRANSFORM_TEX(v.texcoord, _BumpMap);
  #ifndef LIGHTMAP_OFF
  o.lmap.xy =
           v.texcoord1.xy * unity_LightmapST.xy + unity_LightmapST.zw;
  #endif
  float3 worldN = mul((float3x3)_Object2World, SCALED_NORMAL);
  TANGENT_SPACE_ROTATION;
  float3 lightDir = mul (rotation, ObjSpaceLightDir(v.vertex));
  #ifdef LIGHTMAP_OFF
  o.lightDir = lightDir;
  #endif
  #if defined (LIGHTMAP_OFF) || !defined (DIRLIGHTMAP_OFF)
  float3 viewDirForLight = mul (rotation, ObjSpaceViewDir(v.vertex));
  o.viewDir = viewDirForLight;
  #endif
```

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]

- "alpha - Alpha blending mode. Use this for semitransparent shaders.
- vertex:VertexFunction - Custom vertex modification function. See Tree Bark shader for example.
- finalcolor:ColorFunction - Custom final color modification function. See Surface Shader Examples.
- addshadow - Add shadow caster & collector passes. 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 spherical harmonics lights."

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