Charles Hancock

HW3

PROBLEM 1

Added _BaseTex code back in to access a texture that had an alpha and stuff, removed Fresnel, etc., and returned a lerp.

```
// Aaron Lanterman, June 21, 2014
// Cobbled together from numerous sources, particularly The Cg Turtorial
Shader "GPUXX/EnvMapPerPixel" {
    Properties {
        _Cube ("Reflection Cubemap", CUBE) = "white" {}
        _BaseTex ("Texture Material", 2D) = "white" {}
        _etaRatio ("Eta Ratio", Range(0.01,3)) = 1.5
        _crossfade ("Crossfade", Range(0,1)) = 0
        _fresnelBias ("Fresnel Bias", Range(0,1)) = 0.5
        _fresnelScale ("Fresnel Scale", Range(0,1)) = 0.5
        _fresnelPower ("Fresnel Power", Range(0,10)) = 0.5
    }

    SubShader {
        Pass {
            CGPROGRAM
            #include "UnityCG.cginc"
            // includes #define TRANSFORM_TEX(tex,name) (tex.xy * name##_ST.xy + name##_ST.zw)

            #pragma vertex vert_envmapperpixel
            #pragma fragment frag_envmapperpixel

            uniform samplerCUBE _Cube;
            uniform sampler2D _BaseTex;
            uniform float4 _BaseTex_ST;
            uniform float _etaRatio;
            uniform float _crossfade;
            uniform float _fresnelBias;
            uniform float _fresnelScale;
            uniform float _fresnelPower;

            struct a2v {                    // application to vertex
                float4 v: POSITION;
                float3 n: NORMAL;
                float2 tc: TEXCOORD0;
            };


            struct v2f {                // vertex to fragment
                float4 sv: SV_POSITION;
                float3 nWorld: TEXCOORD2;
                float3 vWorldPos: TEXCOORD1;
                float2 tc: TEXCOORD0;

            };
```

```
46
47              v2f vert_envmapperpixel(a2v input) {
48                  v2f output;
49                  output.sv = mul(UNITY_MATRIX_MVP, input.v);
50                   // To transform normals, we want to use the inverse transpose of upper left 3x3
51                  // Putting input.n in first argument is like doing trans((float3x3)_World2Object) * input.n;
52                  output.vWorldPos = mul(_Object2World, input.v).xyz;
53                  output.nWorld = normalize(mul(input.n, (float3x3) _World2Object));
54                  output.tc = TRANSFORM_TEX(input.tc, _BaseTex);
55                  return output;
56              }
57
58              float4 frag_envmapperpixel(v2f input) : COLOR {
59                  // incident is opposite the direction of eyeDir in our other programs
60                  float3 incidentWorld = normalize(input.vWorldPos - _WorldSpaceCameraPos.xyz);
61                  float3 reflectWorld = reflect(incidentWorld,input.nWorld);
62                  //float3 refractWorld = refract(incidentWorld,input.nWorld,_etaRatio);
63
64                  float4 reflectColor = texCUBE(_Cube, reflectWorld);
65                  //float4 refractColor = texCUBE(_Cube, refractWorld);
66                  //float reflectFactor = saturate(_fresnelBias +
67                    // _fresnelScale * pow(1 + dot(incidentWorld, input.nWorld), _fresnelPower));
68
69                  // Diagnostic: uncomment next lines to code refraction as yellow & reflection as blue
70                  // refractColor = float4(1,1,0,1);
71                   //reflectColor = float4(0,0,1,1);
72
73                  // Comment out one of the following two lines and leave the other as desired
74                  // return(lerp(reflectColor, refractColor, _crossfade));
75                  float4 base = tex2D(_BaseTex, input.tc);
76                  return(lerp(base, reflectColor, base.a));
77              }
78
79              ENDCG
80          }
81      }
82  }
```

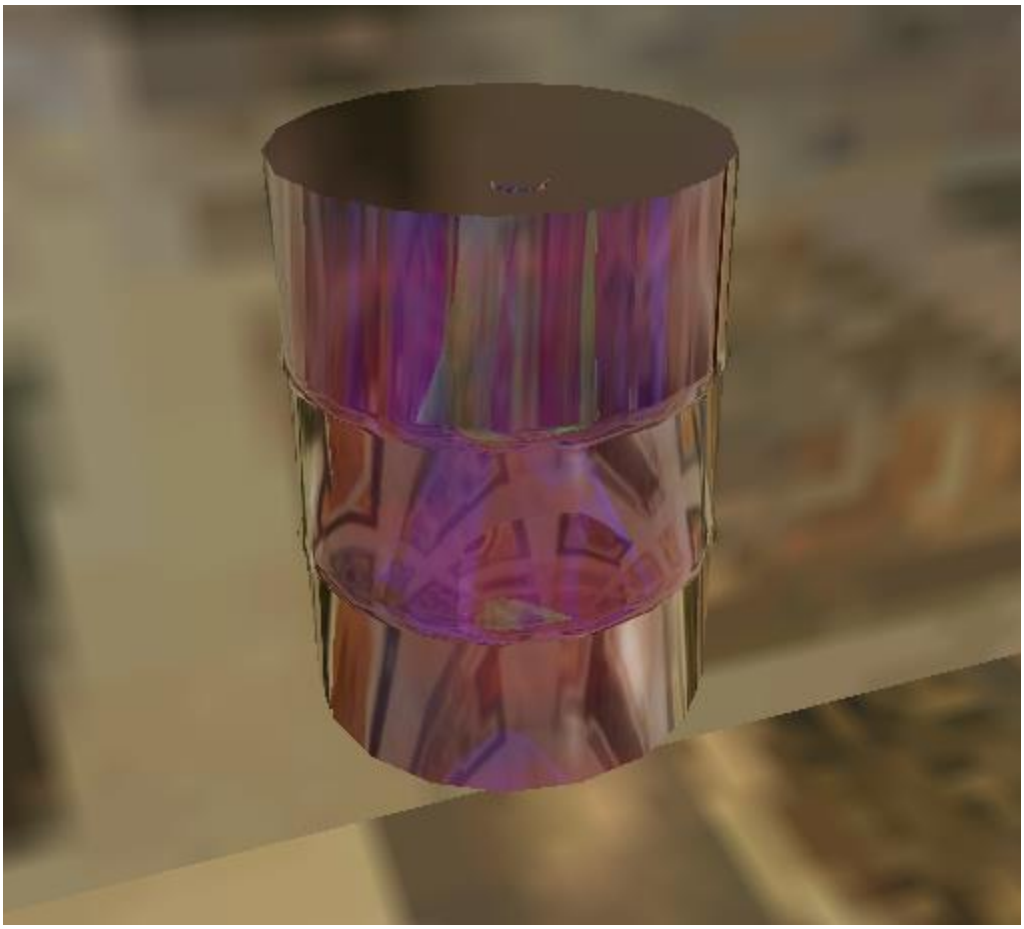
Note the reflections in the windows.

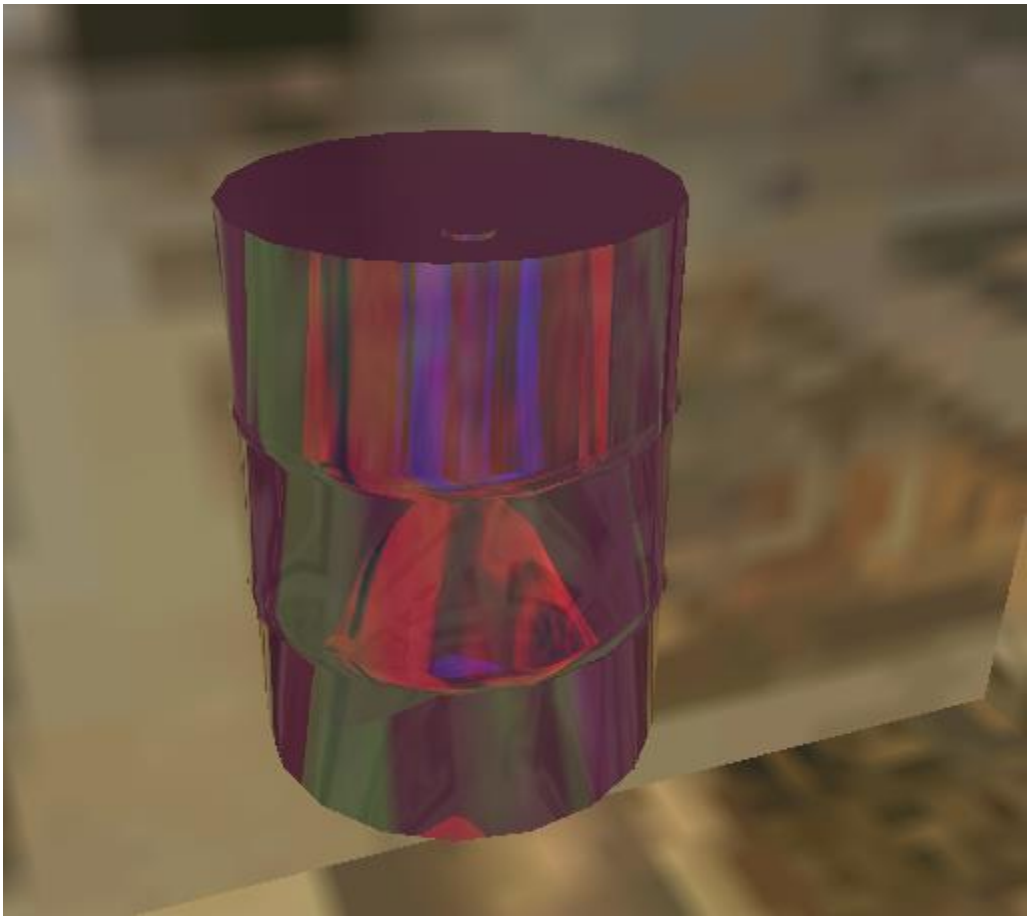
PROBLEM 2

Split into three different color channels with three sliders

```
// Aaron Lanterman, June 21, 2014
// Cobbled together from numerous sources, particularly The Cg Turtorial
Shader "GPUXX/EnvMapPerPixel" {
    Properties {
        _Cube ("Reflection Cubemap", CUBE) = "white" {}
        _etaRatioRed ("Eta Ratio Red", Range(0.01,3)) = 1.5
        _etaRatioGreen ("Eta Ratio Green", Range(0.01,3)) = 1.5
        _etaRatioBlue ("Eta Ratio Blue", Range(0.01,3)) = 1.5
        _crossfade ("Crossfade", Range(0,1)) = 0
        _fresnelBias ("Fresnel Bias", Range(0,1)) = 0.5
        _fresnelScale ("Fresnel Scale", Range(0,1)) = 0.5
        _fresnelPower ("Fresnel Power", Range(0,10)) = 0.5
    }

    SubShader {
        Pass {
            CGPROGRAM
            #include "UnityCG.cginc"
            // includes #define TRANSFORM_TEX(tex,name) (tex.xy * name##_ST.xy + name##_ST.zw)

            #pragma vertex vert_envmapperpixel
            #pragma fragment frag_envmapperpixel

            uniform samplerCUBE _Cube;
            uniform float _etaRatioRed;
            uniform float _etaRatioGreen;
            uniform float _etaRatioBlue;
            uniform float _crossfade;
            uniform float _fresnelBias;
            uniform float _fresnelScale;
            uniform float _fresnelPower;

            struct a2v {                    // application to vertex
                float4 v: POSITION;
                float3 n: NORMAL;
            };


            struct v2f {                // vertex to fragment
                float4 sv: SV_POSITION;
                float3 nWorld: TEXCOORD0;
                float3 vWorldPos: TEXCOORD1;

            };
```

```
45
46          v2f vert_envmapperpixel(a2v input) {
47              v2f output;
48              output.sv = mul(UNITY_MATRIX_MVP, input.v);
49               // To transform normals, we want to use the inverse transpose of upper left 3x3
50              // Putting input.n in first argument is like doing trans((float3x3)_World2Object) * input.n;
51              output.vWorldPos = mul(_Object2World, input.v).xyz;
52              output.nWorld = normalize(mul(input.n, (float3x3) _World2Object));
53              return output;
54          }
55
56
57          float4 frag_envmapperpixel(v2f input) : COLOR {
58              // incident is opposite the direction of eyeDir in our other programs
59              float3 incidentWorld = normalize(input.vWorldPos - _WorldSpaceCameraPos.xyz);
60              float3 reflectWorld = reflect(incidentWorld,input.nWorld);
61              //float3 refractWorld = refract(incidentWorld,input.nWorld,_etaRatioRed);
62
63
64
65              float4 reflectColor = texCUBE(_Cube, reflectWorld);
66              //float4 refractColor = texCUBE(_Cube, refractWorld);
67
68
69              float3 refractWorld = refract(incidentWorld,input.nWorld,_etaRatioRed);
70              float refractedRed = texCUBE(_Cube, refractWorld).r;
71
72              refractWorld = refract(incidentWorld,input.nWorld,_etaRatioGreen);
73              float refractedGreen = texCUBE(_Cube, refractWorld).g;
74
75              refractWorld = refract(incidentWorld,input.nWorld,_etaRatioBlue);
76              float refractedBlue = texCUBE(_Cube, refractWorld).b;
77
78              float4 refractColor = float4(refractedRed, refractedBlue, refractedGreen, 1);
79
80
81              float reflectFactor = saturate(_fresnelBias +
82                 _fresnelScale * pow(1 + dot(incidentWorld, input.nWorld), _fresnelPower));
83
84              // Diagnostic: uncomment next lines to code refraction as yellow & reflection as blue
85              // refractColor = float4(1,1,0,1);
86              // reflectColor = float4(0,0,1,1);
87
88              // Comment out one of the following two lines and leave the other as desired
89              // return(lerp(reflectColor, refractColor, _crossfade));
90              return(lerp(refractColor, reflectColor, reflectFactor));
91          }
92
93          ENDCG
```





PROBLEM 3

All specular and normal map code successfully ported from the GPUXXSpecNormMap.shader file.

```
Shader "GPUXX/ProjectTexture" {
    Properties {
        _BaseTex ("Base (RGB) Gloss (A)", 2D) = "white" {}
        _NormalMap ("Normal Map", 2D) = "bump" {}
        _ProjTex ("Base (RGB)", 2D) = "white" {}
        _SpotPower ("Spotlightiness", Range(0.01,1)) = 0.7
        _Shininess ("Shininess", Range(0.01,1)) = 0.7
        _SpecColor ("Spec Color", Color) = (1,1,1,1)
    }

    SubShader {
        // Directional light colors aren't exposed in "ForwardBase" mode, so we try "Vertex" mode,
        // which really should be called "Simple" mode, as we can still do custom per-pixel lighting
        Tags { "LightMode" = "Vertex" }

        Pass {
            CGPROGRAM
            #include "UnityCG.cginc"

            #pragma vertex vert_projecttexture
            #pragma fragment frag_projecttexture

            uniform sampler2D _BaseTex;
            uniform float4 _BaseTex_ST;
            uniform sampler2D _ProjTex;
            uniform float _SpotPower;
            uniform sampler2D _NormalMap;
            uniform float4 _NormalMap_ST;

            uniform float4x4 _myProjectorMatrix;
            uniform float3 _spotlightDir;
            uniform float4 _SpecColor;
            uniform float _Shininess;

            struct a2v {
                float4 v: POSITION;
                float3 n: NORMAL;
                float2 tc: TEXCOORD0;
                float4 t: TANGENT;
            };

            struct v2f {
                float4 sv: SV_POSITION;
                float2 tc: TEXCOORD0;
                float3 vWorldPos: TEXCOORD1;
                float3 nWorld: TEXCOORD2;
                float4 vProj: TEXCOORD3;
                float3 tWorld: TEXCOORD4;
                float3 btWorld: TEXCOORD5;
                float2 ntc: TEXCOORD6;
            };
```

```
v2f vert_projecttexture(a2v input)  {
    v2f output;
    output.sv = mul(UNITY_MATRIX_MVP, input.v);
    output.vProj = mul(_myProjectorMatrix, input.v);
    // To transform normals, we want to use the inverse transpose of upper left 3x3
    // Putting input.n in first argument is like doing trans((float3x3)_World2Object) * input.n;
    output.vWorldPos = mul(_Object2World, input.v).xyz;
    output.nWorld = normalize(mul(input.n, (float3x3) _World2Object));
    output.tWorld = normalize(mul((float3x3) _Object2World, input.t.xyz));
    output.btWorld = normalize(cross(output.nWorld, output.tWorld)
            * input.t.w);
    output.tc = TRANSFORM_TEX(input.tc, _BaseTex);
    output.ntc = TRANSFORM_TEX(input.tc, _NormalMap);
    return output;
}

float4 frag_projecttexture(v2f input) : COLOR {

    float2 nMapXY = 2 * tex2D(_NormalMap, input.ntc).ag - 1;
    float nMapRecreatedZ = sqrt(1 - saturate(dot(nMapXY, nMapXY)));

    float3 nW = normalize(input.nWorld);
    float3 tW = normalize(input.tWorld);
    float3 btW = normalize(input.btWorld);

    float3 newNormal = tW * nMapXY.x + btW * nMapXY.y + nW * nMapRecreatedZ;
    newNormal = normalize(newNormal);

    // Only use this shader with a point light
    float3 lightDir = normalize(_WorldSpaceLightPos0.xyz - input.vWorldPos * _WorldSpaceLightPos0.w);
    float3 eyeDir = normalize(_WorldSpaceCameraPos.xyz - input.vWorldPos);
    // Renormalizing because the GPU's interpolator doesn't know this is a unit vector
    float3 n = normalize(input.nWorld);
    float3 diff_almost = 2*unity_LightColor0.rgb * max(0, dot(n, lightDir));
    float spotlightEffect = pow(dot(normalize(_spotlightDir), -lightDir),_SpotPower * 128.0);
    diff_almost *= spotlightEffect;
    diff_almost *= tex2Dproj(_ProjTex,input.vProj);

    float3 h = normalize(lightDir + eyeDir);
    float ndoth = max(0, dot(newNormal, h));
    float3 spec_almost = 2*unity_LightColor0.rgb * _SpecColor.rgb * pow(ndoth, _Shininess*128.0);









    float4 base = tex2D(_BaseTex, input.tc);
    float3 output = (diff_almost + 2*UNITY_LIGHTMODEL_AMBIENT.rgb) * base.rgb +
                spec_almost.rgb * base.a;
    return(float4(output,1));
}

ENDCG
```

I put the revolver in the example scene to demonstrate that buzz still lights up on it. Below is a picture that better shows the weirdo bump mapping. (Look at the barrel.)