Physically Based Shading

Dušan Drevický

March 2, 2018

1 Introduction

This work is concerned with implementing a physically based shading model running in real-time on the GPU. The main attributes of such a model is that it takes into greater account the way light actually behaves as opposed to approximating what we intuitively think it should do [Eng17]. The model will be implemented in OpenGL and GLSL and include analytical light sources (point lights) and also image-based lighting (the environment will be represented by an HDR environment map).

2 Theory

The theory of physically based shading is described thoroughly in [AMHH08] and [Hof13].

2.1 Bidirectional Reflectance Distribution Function

A physically based shading model deals with the radiometric quantity called *radiance* (denoted L), used to quantify the magnitude of light along a single ray. The model we use works with a common assumption that shading can be handled locally at all points. The response of a given surface to light then depends only on the incoming light direction $\mathbf{l}$ and the view direction $\mathbf{v}$ (both are unit vectors). The response is characterized by the **Bidirectional Reflectance Distribution Function** (BRDF) denoted $f(\mathbf{l}, \mathbf{v})$. Given a ray of light incoming from a certain direction, the function returns the relative contribution of reflected and scattered light over all outgoing directions above the surface. It is a spectral quantity but the wavelengths do not interact between each other which means that the wavelengths do not need to be

inputs to the function. We can instead perform an element-wise product[1] between an RGB-valued[2] BRDF and RGB-valued light color. The *reflectance equation* can then be defined as

$$L_o(\mathbf{v}) = \int_\Omega f(\mathbf{l}, \mathbf{v}) \odot L_i(\mathbf{l})\mathbf{n} \cdot \mathbf{l} \, \mathrm{d}\mathbf{l} \tag{1}$$

where L_o denotes the outgoing radiance from a surface, L_i the incoming radiance, $\mathbf{n}$ is the surface normal and Ω is the hemisphere oriented around $\mathbf{n}$.

The BRDF models two distinct physical phenomena: surface reflection and subsurface scattering. It is useful to model these as separate terms (specular and diffuse) since their behavior is different. The equation for BRDF is

$$f(\mathbf{l}, \mathbf{v}) = k_d f_{Lambert}(\mathbf{l}, \mathbf{v}) + k_s f_{Cook-Torrance}(\mathbf{l}, \mathbf{v}) \tag{2}$$

where k_s is the amount of light that is specularly reflected (often determined by the F term), k_s is the amount of light undergoing subsurface scattering and $k_s + k_d = 1$.

2.2 Surface Reflectance (Specular Term)

The specular term is most often based on *microfacet theory*. It assumes the presence of surface variation *microgeometry* at a scale smaller than is observable but greater than light wavelength. Each microgeometry facet is treated as optically flat surface (it splits the light into exactly two directions: reflection and refraction). The specular term has the form

$$f_{Cook-Torrance}(\mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h})G(\mathbf{l}, \mathbf{v}, \mathbf{h})D(\mathbf{h})}{4(\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{v})} \tag{3}$$

Each microsurface reflects light based on its *microgeometry normal* $\mathbf{m}$. When evaluating the BRDF for a specific $\mathbf{l}$ and $\mathbf{v}$, only those microsurfaces for which $\mathbf{m}$ corresponds to the half-way vector $\mathbf{h}$ between $\mathbf{v}$ and $\mathbf{l}$ can contribute to the outgoing irradiance (see Figure 1 left). This is modeled by the **Normal Distribution Function** $D(\mathbf{h})$. It returns the concentration of surface points oriented in such a way that they could reflect light from $\mathbf{l}$ to $\mathbf{v}$. The version used in this work is the GGX/Trowbridge-Reitz[3] where $\alpha = Roughness^2$[4] as

[1] Denoted as $\odot$ in this work

[2] We use an RGB triplet as an approximation of the visible light spectrum

[3] For different F, G, D functions see [Kar13b]

[4] In the commonly used *roughness/metallic* model creation work-flow (see [McD17]), roughness is a value in range 0 to 1 defining whether the specular highlight is concentrated but small (1) or weak but broad (0).

described in [Kar13a]

$$D(\mathbf{h}) = \frac{\alpha^2}{\pi((\mathbf{n}\cdot\mathbf{h})^2(\alpha^2-1)+1)^2} \tag{4}$$

$G(\mathbf{l},\mathbf{v},\mathbf{h})$ is the **Geometry Function** and it gives the percentage of surface points with $\mathbf{m}=\mathbf{h}$ that are not *shadowed* (occluded from the light $\mathbf{l}$ by surrounding microgeometry) or *masked* (occluded from $\mathbf{v}$ by the surrounding microgeometry). Masked and shadowed points cannot contribute to the reflection (see Figure 1 middle and right respectively). Note that the G term depends on the distribution function D [WMLT07].

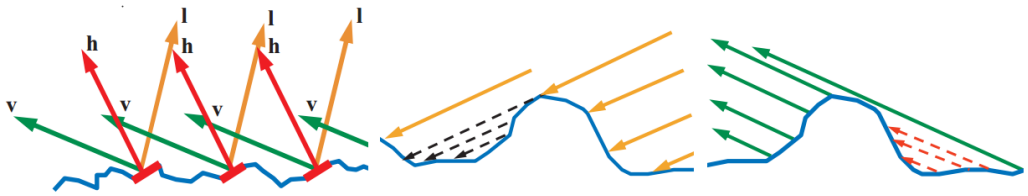

Figure 1: On the left, surface points with $\mathbf{m}=\mathbf{h}$ reflect $\mathbf{l}$ to $\mathbf{v}$. In the middle, occluded points are shadowed. On the right some points are masked and cannot reach the $\mathbf{v}$ direction. (Figure from [AMHH08])

We use the Smith G approximation as described in [Kar13a] and [WMLT07] which approximates the bidirectional shadowing-masking function G as the separable product of two monodirectional shadowing terms G_1 (where G_1 is derived from the D distribution) as follows

$$\begin{aligned} k &= \frac{(Roughness+1)^2}{8} \\ G_1(\mathbf{v}) &= \frac{\mathbf{n}\cdot\mathbf{v}}{(\mathbf{n}\cdot\mathbf{v})(1-k)+k} \\ G(\mathbf{l},\mathbf{v},\mathbf{h}) &= G_1(\mathbf{l})G_1(\mathbf{v}) \end{aligned} \tag{5}$$

The term $F(\mathbf{l},\mathbf{h})$ is the **Fresnel Reflectance Function** computing the ratio of reflected to refracted light of a surface. It depends on the *incident angle* of light and the refractive index of the material. It is a spectral quantity (RGB triplet) in range 0 to 1. We use the popular Schlick's approximation [Sch94] to the Fresnel equations

$$F(\mathbf{v},\mathbf{h}) = F_0 + (1-F_0)(1-(\mathbf{h}\cdot\mathbf{v}))^5 \tag{6}$$

where F_0 is the material's surface reflectance at incident angle equal to 0.

2.3 Subsurface Reflectance (Diffuse Term)

The most commonly used model is the *Lambertian* model which is a constant value

$$f_{Lambert}(\mathbf{l}, \mathbf{v}) = \frac{\mathbf{c}_{diff}}{\pi} \tag{7}$$

where $\mathbf{c}_{diff}$ is the diffusely reflected fraction of light (RGB-triplet) often called the *diffuse color*.

2.4 Direct Light Sources

Since we know the position of all direct light sources (point lights in this work) we do not have to integrate over the entire hemisphere Ω above the shaded surface point. It is enough to calculate the direction vectors $\mathbf{l}$ for each light in the scene and compute the Reflectance Equation as a simple sum over all of the lights in the scene (see 3.1 for implementation details).

2.5 Image Based Lighting

The situation is more complicated for indirect (ambient) light which can come from any direction in the environment. Expanding the Reflectance Equation (1) we have

$$L_o(\mathbf{v}) = \int_\Omega (\frac{k_d c}{\pi} + k_s \frac{F(\mathbf{v}, \mathbf{h}) G(\mathbf{l}, \mathbf{v}, \mathbf{h}) D(\mathbf{h})}{4(\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{v})}) L_i(\mathbf{l}) \mathbf{n} \cdot \mathbf{l} \, \mathrm{d}\mathbf{l} \tag{8}$$

2.5.1 Diffuse part

We can split the integral in (8) and deal with the diffuse part first. After factoring out the constant term we have

$$L_{oDiffuse}(\mathbf{v}) = \frac{k_d c}{\pi} \int_\Omega L_i(\mathbf{l}) \mathbf{n} \cdot \mathbf{l} \, \mathrm{d}\mathbf{l} \tag{9}$$

The integral now depends only on $\mathbf{l}$. Using the environment cubemap as input, we create a new *irradiance cubemap* which stores the integral result for the given outgoing direction in the corresponding point of this cubemap (for implementation details see 3.2.1).

2.5.2 Specular part

The specular part which is left after factoring (9) from (8) is

$$L_{oSpecular}(\mathbf{v}) = \int_{\Omega} f_{Cook-Torrance}(\mathbf{l}, \mathbf{v}) L_i(\mathbf{l}) \mathbf{n} \cdot \mathbf{l} \, \mathrm{d}\mathbf{l} \tag{10}$$

and is in practice computed using numerical integration

$$L_{oSpecular}(\mathbf{v}) \approx \frac{1}{N} \sum_{k=1}^{N} \frac{f_{Cook-Torrance}(\mathbf{l}_k, \mathbf{v}) L_i(\mathbf{l}_k) \mathbf{n} \cdot \mathbf{l}}{p(\mathbf{l}_k, \mathbf{v})} \tag{11}$$

Note that in practice $k_s = F(\mathbf{l}, \mathbf{h})$ so we can omit the k_s term as it is already included in the expression for $f_{Cook-Torrance}$. It would be inefficient to sample the entire hemisphere as was done when computing (9) considering that most of the light undergoing specular reflection ends up being in a specular lobe focused around the reflection vector $\mathbf{r}$ ($\mathbf{l}$ reflected about microgeometry normal $\mathbf{h}$). We therefore proceed by importance sampling the half-vectors $\mathbf{h}$ from the NDF using the roughness parameter and pseudorandom sample from the 2D Hammersley point set (see [Dam12] for generating this low-discrepancy sequence) as inputs. Actual incoming light samples are then taken from the $\mathbf{l}$ direction ($\mathbf{v}$ reflected about the sampled half-vector $\mathbf{h}$).

Even this is too slow for real-time use however, so we use the *split sum approximation* of (11) suggested by [Kar13a] as follows

$$L_{oSpecular}(\mathbf{v}) \approx \left(\frac{1}{N} \sum_{k=1}^{N} L_i(\mathbf{l}_k) \right) \left(\frac{1}{N} \sum_{k=1}^{N} \frac{f_{Cook-Torrance}(\mathbf{l}_k, \mathbf{v}) \mathbf{n} \cdot \mathbf{l}}{p(\mathbf{l}_k, \mathbf{v})} \right) \tag{12}$$

Each of the two sums can now be precomputed separately and both cases are described in part 3.2.2. Some further explanation is appropriate for the second sum however. Comparing equation (10) and the second sum from above (rewritten in the form of an integral again) in (13) we can see that the computing the latter amounts to integrating the specular BRDF with a solid-white environment, that is $L_i(\mathbf{l}_k) = 1$. We can substitute Schlick's Fresnel approximation $F(\mathbf{v}, \mathbf{h}) = F_0 + (1 - F_0)(1 - \mathbf{v} \cdot \mathbf{h})^5$ into the $f_{Cook-Torrance}$ term and after some algebraic manipulation find that F_0 can be factored out of the integral as shown in (14). Note that f stands for $f_{Cook-Torrance}$ in both of these equations.

$$\int_{\Omega} f(\mathbf{l}, \mathbf{v})\mathbf{n} \cdot \mathbf{l} \, \mathrm{dl} = \tag{13}$$

$$F_0 \int_{\Omega} \frac{f(\mathbf{l}, \mathbf{v})}{F(\mathbf{v}, \mathbf{h})}(1 - (1 - \mathbf{v} \cdot \mathbf{h})^5)\mathbf{n} \cdot \mathbf{l} \, \mathrm{dl} + \int_{\Omega} \frac{f(\mathbf{l}, \mathbf{v})}{F(\mathbf{v}, \mathbf{h})}(1 - \mathbf{v} \cdot \mathbf{h})^5 \mathbf{n} \cdot \mathbf{l} \, \mathrm{dl} \tag{14}$$

By factoring F_0 out of the integral we have left it dependent only on two inputs (*Roughness* and $\mathbf{n} \cdot \mathbf{l}$). The computation now only has two outputs which are a scale and bias to F_0 that can now be precalculated as described in section 3.2.3.

3 Implementation

The application is cross-platform (tested on Ubuntu 16.04 and Windows 10) and uses SDL2 [sdl17] for window and OpenGL context creation. Additional libraries used are glad [Her17], glm [glm17] and imgui [img17]. The shading model implementation is based primarily on [Kar13a].

3.1 Direct Light Sources

Implementing the Reflectance Equation and BRDF for point light sources is relatively straightforward. Firstly, we use the roughness/metallic workflow to parametrize the object materials and these attributes are received either as uniforms or sampled from textures. With these values given, we cycle through all of the lights in the scene and accumulate the outgoing radiance from the currently evaluated surface point. The computation is done in the fragment shader (see Listing 1). The integral in equation (1) is reduced to a sum for analytical lights. This is implemented in `PBR.frag` file.

```
// Reflectance Equation (1)
vec3 Lo = vec3(0.0);
for (int i = 0; i < NUM_POINT_LIGHTS; ++i) {
  vec3 Li = lightColor;
  vec3 L = normalize(uPointLights[i].position - p);
  vec3 H = normalize(L + V);
  // BRDF (2)
  vec3 fLambert = albedo / PI;
  float D = N_GGXTR(N, H, roughness);
  vec3 F = F_Schlick(V, H, F0);
  float G = G_Smith(N, V, L, roughness);
  float LdotN = max(dot(N, L), 0);
  vec3 fCookTorrance = (D * F * G) / (4 * VdotN * LdotN + 0.001);
  vec3 kD = 1.0 - F;      // F approximates the kS term from (2)
  Lo += (kD * fLambert + fCookTorrance) * Li * LdotN;}
```

Listing 1: The Reflectance Equation (1) and BRDF (2) implemented in the fragment shader for point light sources (from `PBR.frag`)

3.2 Image Based Lighting

An HDR environment cubemap is used to represent the environment which is the source of ambient (indirect) lighting.

3.2.1 Diffuse part

To create the irradiance diffuse cubemap described in 2.5.1 we render the inside of a cube from six different view directions each perpendicular to a different cube side. Sampling the environment cubemap in a given direction $\mathbf{l}$ provides us with the irradiance $L_i(\mathbf{l})$. In the fragment shader, the integral for each view direction is approximated using a fixed amount of uniformly spaced steps (and an equal amount of sampled vectors $\mathbf{l}$). We generate the sample vectors in the hemisphere by iterating over two angles `polar` ranging from 0 to $\pi/2$ and `azimuth` ranging from 0 to 2π. For each combination of these two angles we create a vector pointing in the corresponding direction in the hemisphere and then project it to the world space coordinates of the direction $\mathbf{v}$ for which we are computing the integral. The direction $\mathbf{v}$ is just the normalized position of the shaded cube fragment in world space. See the `EnvToIrradiance.frag` file for more details.

3.2.2 Specular part

The result of precalculating the first sum in (12) is a *pre-filtered environment cubemap*. It is also created by rendering a cube from six different view directions and storing the six resulting color buffers as textures of a new cubemap. The cube is however rendered for different levels of *Roughness* and the results are stored in different mip-map levels of the created cubemap texture (as the roughness increases so does the mip-map level). The $\mathbf{l}_k$ vectors are generated using importance sampling the NDF distribution as described in section 2.5.2. Since the NDF depends on the viewing direction $\mathbf{v}$, an approximation is made when creating this cubemap by setting $\mathbf{v} = \mathbf{n} = \mathbf{r}$. The resulting code is in `EnvToPrefilteredEnv.frag` file.

For each sample we generate a pseudorandom vector from the Hammersley sequence and use it along with the *Roughness* value for which we are currently computing the cubemap texture (*Roughness* is passed into the shader as a uniform) to generate a half-vector `H` and then the incoming light direction `L`. We sample the environment map at this direction and add that to the resuling color. The `NdotL` weighing factor is included because the generated map looks better in practice.

```
vec3 N = normalize(vsLocalPosition);
vec3 R = N;
vec3 V = R;
vec3 prefilteredColor = vec3(0);
float totalWeight = 0.0;
for(int i = 0; i < numSamples; i++) {
  vec2 Xi = Hammersley(i, numSamples);
  vec3 H = ImportanceSampleGGX(Xi, uRoughness, N);
  vec3 L = normalize(2 * dot(V, H) * H - V);
  float NdotL = clamp(dot(N, L), 0, 1);
  if(NdotL > 0) {
    prefilteredColor += texture(uHDREnvironmentCubemap, L).rgb * NdotL;
    totalWeight += NdotL;
  }
}
vec3 irradiance = prefilteredColor / totalWeight;
fragColor = vec4(irradiance, 1.0);
```

Listing 2: Creating the prefiltered environment map (from `EnvToPrefilteredEnv.frag`)

We can precalculate the result of the second sum in (12) in a 2D LUT created by rendering a screen quad whose 2D texture coordinates serve as inputs to the approximated integral (recall that the inputs are *Roughness* and $\mathbf{n} \cdot \mathbf{l}$ and that both of these are in the range $[0, 1]$ as are the texture coordinates). The outputs of the computation (scale and bias to F_0) are stored in a `GL_RG16F` texture.

```
float NdotV = vsTexCoords.x;
float roughness = vsTexCoords.y;
vec3 N = vec3(0, 0, 1);
vec3 V;
V.x = sqrt(1 - NdotV * NdotV);
V.y = 0.0;
V.z = NdotV;

float F0_scale = 0;
float F0_bias = 0;
for (uint i = 0u; i < numSamples; ++i) {
  vec2 Xi = Hammersley(i, numSamples);
  vec3 H = ImportanceSampleGGX(Xi, roughness, N);
  vec3 L = normalize(2 * dot(V, H) * H - V);
  float NdotL = saturate(L.z);
  float NdotH = saturate(H.z);
  float VdotH = saturate(dot(V, H));
  if (NdotL > 0) {
    float G = G_Smith(N, V, L, roughness);
    float G_Vis = G * VdotH / (NdotH * NdotV);
    float Fc = pow(1 - VdotH, 5);
    F0_scale += G_Vis * (1 - Fc);
    F0_bias += G_Vis * Fc;
  }
}
fragColor = vec2(F0_scale, F0_bias) / float(numSamples);
```

Listing 3: Creating the integrated BRDF 2D LUT (from `EnvToIntegratedBRDF.frag`)

3.2.3 Indirect lighting

The only thing that remains is to sample the created textures in the `PBR.frag` shader when calculating the color of a surface point. Notice that we call `textureLod` with mip level based on the *Roughness* of the current material when sampling the prefiltered environment map. For this to work properly it is necessary to set `GL_LINEAR_MIPMAP_LINEAR` filtering when creating the cubemap and also to enable `GL_TEXTURE_CUBE_MAP_SEAMLESS` for seamless interpolation between cubemap texture faces.

```
vec3 F = F_Schlick(V, N, F0);
vec3 kD = 1.0 - F;
kD *= 1.0 - metalness;
// Indirect diffuse
vec3 diffuseIrradiance = texture(uCubeIrradiance, N).rgb;
vec3 indirectDiffuse = diffuseIrradiance * albedo;
// Indirect specular
vec3 R = 2 * dot(V, N) * N - V;
float mip  = roughness * MAX_MIP_LEVEL;
vec3 prefilteredColor = textureLod(uCubePrefilteredEnv, R, mip).rgb;
vec2 envBRDF = texture(uTexIntegratedBRDF, vec2(VdotN, roughness)).rg;
vec3 indirectSpecular = prefilteredColor * (F*envBRDF.x + envBRDF.y);

vec3 indirectLight = kD * indirectDiffuse + indirectSpecular;}
vec3 fragmentColor = directLight + indirectLight;
```

Listing 4: Computing indirect lighting (from `PBR.frag`)

4 Evaluation

The final results seem to hold quite well visually when compared with a reference in [Kar13a]. The scene rendered in Figure 2 with 4 multisamples takes on average[5] 3.05 $ms/frame$ or 327.91 FPS to compute but it is important to note that the shaders were not especially optimized with performance in mind and there are a plenty of obvious and not so obvious optimizations possible.

5 Conclusion

We implemented a physically based shading model with image based lighting performing reasonably well in real-time. As mentioned above, the shader code is not optimized for speed but rather for clarity. It is also worth noting that for a larger and more complicated scene we would not be able to imitate ambient lighting with a single environment map but would have to use a more

[5]Measured on GeForce GTX 860M, Intel Core i5-4210H, 8GB RAM system.

Figure 2: Final results. Image on the left is illuminated only with direct lighting, image on the right also uses image-based lighting. Metalness property increases from bottom to top and roughness increases from left to right.

advanced technique such as implementing several light probes placed in the scene with the environment light for a scene point given by interpolation among these probes (see [Jag15] for more details on this technique).

References

[AMHH08] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering*. CRC Press, 3rd edition, 2008.

[Dam12] Holger Dammertz. Hammersley points on the hemisphere, 2012. [Online; accessed 29-September-2017].

[Eng17] Unreal Engine. Physically based materials, 2017. [Online; accessed 29-September-2017].

[glm17] Opengl mathematics, 2017. [Online; accessed 29-September-2017].

[Her17] David Herberth. Simple directmedia layer, 2017. [Online; accessed 29-September-2017].

[Hof13] Naty Hoffman. Siggraph 2013 course - background: Physics and math of shading, 2013. [Online; accessed 29-September-2017].

[img17] Immediate mode gui for c++, 2017. [Online; accessed 29-September-2017].

[Jag15] Chetan Jags. Image based lighting, 2015. [Online; accessed 30-September-2017].

[Kar13a] Brian Karis. Siggraph 2013 course - real shading in unreal engine 4, 2013. [Online; accessed 29-September-2017].

[Kar13b] Brian Karis. Specular brdf reference, 2013. [Online; accessed 29-September-2017].

[McD17] Wes McDermott. The comprehensive pbr guide by allegorithmic - vol. 2, 2017. [Online; accessed 29-September-2017].

[Sch94] Christophe Schlick. *An Inexpensive BDRF Model for Physically based Rendering. Computer Graphics Forum*, 1994. [Online; accessed 29-September-2017].

[sdl17] Simple directmedia layer, 2017. [Online; accessed 29-September-2017].

[WMLT07] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. *Microfacet Models for Refraction through Rough Surfaces. Eurographics Symposium on Rendering*, 2007. [Online; accessed 29-September-2017].