Introduction to Shader development

h_da WS2020/21

Paul Nasdalack

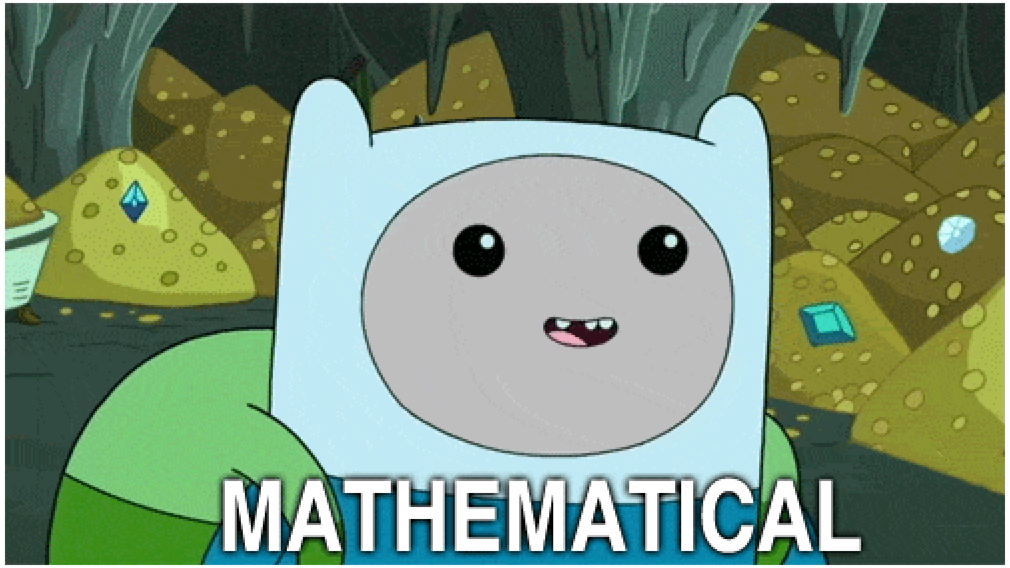
info@paul-nasdalack.com

@littleBugHunter

Heavily based on: https://learnopengl.com/PBR/Theory
All formulas taken from there

Physically Based Rendering

$$L_o(p,\omega_o) = \int\limits_{\Omega} f_r(p,\omega_i,\omega_o) L_i(p,\omega_i) n \cdot \omega_i d\omega_i$$



What is PBR

- Based on Physical light interaction
- Must be energy conserving
- Looks “correct” in almost all lighting scenarios

What is not PBR (aka the crimes we‘ve committed so far)

```
fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse + specular;
```

- Lambert is an ok abstraction of physically based diffuse light
- BlinnPhong is not physically based at all
- Simply adding diffuse and specular light components is a big no no!

Why can‘t we add diffuse and specular

```
fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse + specular;
```

- Diffuse and specular functions calculate light interactions of the same ray
- By adding them together we double the energy of that ray
- We need to split the ray into two parts with separate energies
  - (kS and kD)

Why can‘t we add diffuse and specular

```
float3 kS = SpecularPart;
float3 kD = 1-kS;

fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse * kD + specular * kS;
```

Why can‘t we add diffuse and specular

float3?!

```
float3 kS = SpecularPart;
float3 kD = 1-kS;

fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse * kD + specular * kS;
```

Why can‘t we add diffuse and specular

float3?!

```
float3 kS = SpecularPart;
float3 kD = 1-kS;

fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse * kD + specular * kS;
```

- The Split is not always constant over the whole color spectrum
  - Metals usually have a colored specular component
  - Non metals are usually non Colored

Why can‘t we add diffuse and specular

Specular Part?!

```
float3 kS = SpecularPart;
float3 kD = 1-kS;

fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse * kD + specular * kS;
```

- The Split is not always constant over the whole color spectrum
  - Metals usually have a colored specular component
  - Non metals are usually non Colored

The „specular part“

aka specular component

aka Fresnel term

(speak: “Freh-nel”)

(the “s” is silent)

(because it’s cool)

(and french)

Fresnel

- Fresnel is what light does at low viewing angles
- The flatter you look at a surface, the more light will be reflected
- You can try this at home with your desk
- In theory at a 90° viewing angle 100% of the light is reflected and none is absorbed

Fresnel Function

$$F_{Schlick}(h, v, F_0) = F_0 + (1 - F_0)(1 - (h \cdot v))^5$$

cosTheta?!

```
float3 fresnelSchlick(float cosTheta, float3 F0)
{
	return F0 + (1.0 - F0) * pow(1.0 - cosTheta, 5.0);
}
```

- Just a fancy word for ViewDir dot HalfViewDir (halfway point between Light and View Direction)

Fresnel Function

$$F_{Schlick}(h, v, F_0) = F_0 + (1 - F_0)(1 - (h \cdot v))^5$$

```
float3 fresnelSchlick(float cosTheta, float3 F0)
{
        return F0 + (1.0 - F0) * pow(1.0 - cosTheta, 5.0);
}
```

F0?!

- F0 describes the specular component at the worst case Scenario of viewing the surface from the top
- At 90° it's always white
  - Note how this is just a fancy lerp between F0 and white with some power curve applied to cosTheta

F0

- F0 can be measured and there are tables online for it
- Non metals usually stay between 0.02 and 0.17
- We can simply set it to 0.04 and it will look ok

Material	F_0 (Linear)	F_0 (sRGB)	Color
Water	(0.02, 0.02, 0.02)	(0.15, 0.15, 0.15)	
Plastic / Glass (Low)	(0.03, 0.03, 0.03)	(0.21, 0.21, 0.21)	
Plastic High	(0.05, 0.05, 0.05)	(0.24, 0.24, 0.24)	
Glass (high) / Ruby	(0.08, 0.08, 0.08)	(0.31, 0.31, 0.31)	
Diamond	(0.17, 0.17, 0.17)	(0.45, 0.45, 0.45)	
Iron	(0.56, 0.57, 0.58)	(0.77, 0.78, 0.78)	
Copper	(0.95, 0.64, 0.54)	(0.98, 0.82, 0.76)	
Gold	(1.00, 0.71, 0.29)	(1.00, 0.86, 0.57)	
Aluminium	(0.91, 0.92, 0.92)	(0.96, 0.96, 0.97)	
Silver	(0.95, 0.93, 0.88)	(0.98, 0.97, 0.95)	

F0

- F0 can be measured and there are tables online for it
- Metals have a much bigger range and are colored
- As metals don't have an albedo, we can use the albedo color here

Material	F_0 (Linear)	F_0 (sRGB)	Color
Water	(0.02, 0.02, 0.02)	(0.15, 0.15, 0.15)	
Plastic / Glass (Low)	(0.03, 0.03, 0.03)	(0.21, 0.21, 0.21)	
Plastic High	(0.05, 0.05, 0.05)	(0.24, 0.24, 0.24)	
Glass (high) / Ruby	(0.08, 0.08, 0.08)	(0.31, 0.31, 0.31)	
Diamond	(0.17, 0.17, 0.17)	(0.45, 0.45, 0.45)	
Iron	(0.56, 0.57, 0.58)	(0.77, 0.78, 0.78)	
Copper	(0.95, 0.64, 0.54)	(0.98, 0.82, 0.76)	
Gold	(1.00, 0.71, 0.29)	(1.00, 0.86, 0.57)	
Aluminium	(0.91, 0.92, 0.92)	(0.96, 0.96, 0.97)	
Silver	(0.95, 0.93, 0.88)	(0.98, 0.97, 0.95)	

Why can‘t we add diffuse and specular

```
float3 kS = fresnelSchlick(NdotV, F0);
float3 kD = 1-kS;

fixed3 diffuse = lambert();
fixed3 specular = blinnPhong();
return diffuse * kD + specular * kS;
```

- This whole code part is called BRDF (bidirectional reflectance distribution function)

Why can‘t we add diffuse and specular

```
fixed3 brdf()
{
    float3 kS = fresnelSchlick(NdotV, F0);
    float3 kD = 1-kS;

    fixed3 diffuse = lambert();
    fixed3 specular = blinnPhong();
    return diffuse * kD + specular * kS;
}
```

- This whole code part is called BRDF (bidirectional reflectance distribution function)

Why can‘t we add diffuse and specular

```
fixed3 brdf()
{
    float3 kS = fresnelSchlick(NdotV, F0);
    float3 kD = 1-kS;

    fixed3 diffuse = lambert();
    fixed3 specular = blinnPhong();
    return diffuse * kD + specular * kS;
}
```

- This whole code part is called BRDF (bidirectional reflectance distribution function)
- Lambert and BlinnPhong are pretty bad, we can do better

Cook-Torrance

Cook-Torrance

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}$$

Lambert is back!

$$f_{lambert} = \frac{c}{\pi}$$

We divide by PI to counteract some later calculations

Cook-Torrance

(The specular part)

Cook-Torrance (The specular part)

$$f_{cook-torrance} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

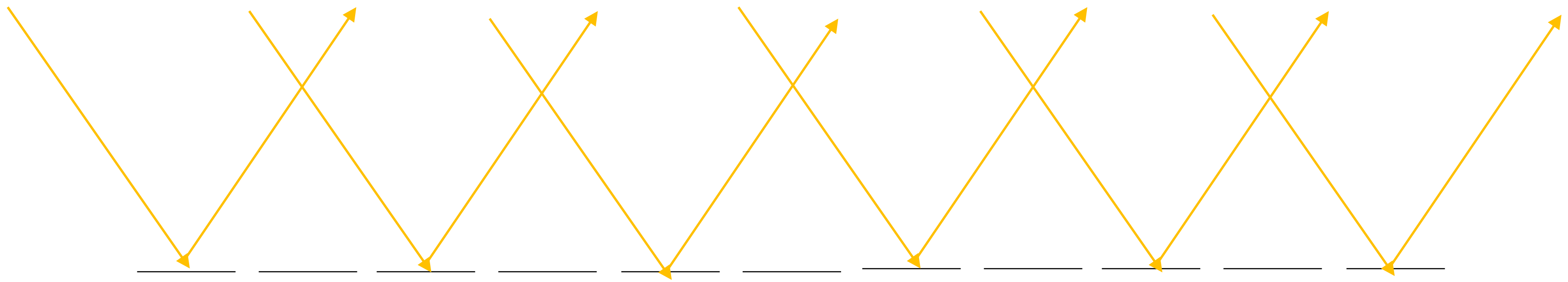
- DFG stands for:
  - normal **D**istribution function
  - **F**resnel
  - **G**eometry
- We already know Fresnel
- The other two are functions are dependent on roughness

INTRODUCING: MICROFACETS

Image source: https://commons.wikimedia.org/wiki/File:Disco_ball4.jpg

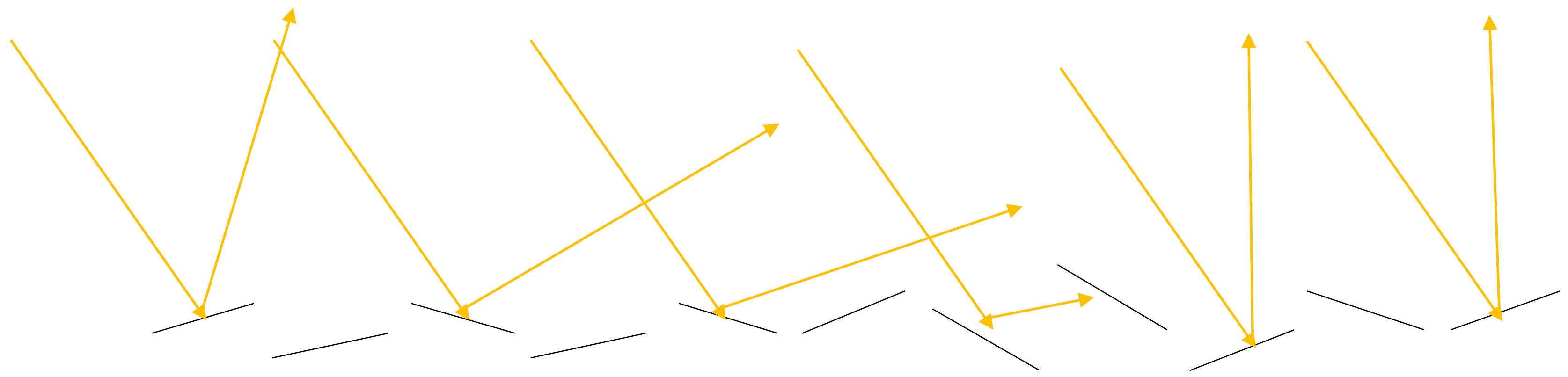
Microfacets

- Material roughness can be approximated with many tiny little mirrors



Microfacets

- Material roughness can be approximated with many tiny little mirrors

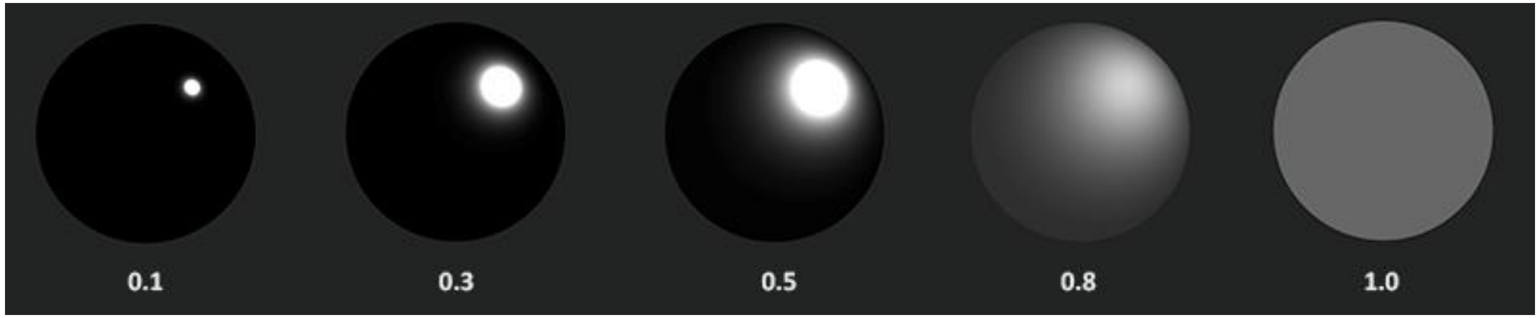


- Two things are happening:

1. The reflected light gets more spread out (Normal distribution)
2. Some rays are obstructed by microbumps (geometry obstruction)

Normal distribution(The reflected light gets more spread out)

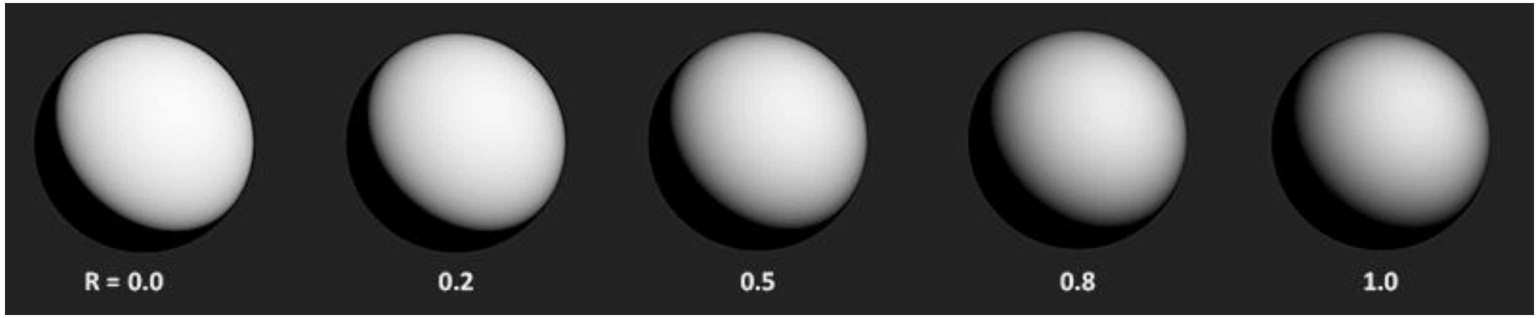
$$NDF_{GGXTR}(n, h, \alpha) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2}$$



- a is our roughness
- n is the surface normal
- h is the half normal (viewDir + lightDir)
- Just like in blinnPhong comparing the angle between the halfNormal and the surface normal gives the distance to the highlight center

Geometry Obstruction(The reflected light sometimes hits microbumps)

$$G_{SchlickGGX}(n, v, k) = \frac{n \cdot v}{(n \cdot v)(1 - k) + k}$$



- n is the surface normal
- v is the view Direction
- k is a function that is switched depending on direct or indirect lighting

$$k_{direct} = \frac{(\alpha + 1)^2}{8}$$

$$k_{IBL} = \frac{\alpha^2}{2}$$

Cook-Torrance (The specular part)

$$f_{cook-torrance} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

- DFG stands for:
    - normal **D**istribution function
    - **F**resnel
    - **G**eometry

Cook-Torrance

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}$$

Lambert is back!

$$f_{lambert} = \frac{c}{\pi}$$

We divide by PI to counteract some later calculations

$$f_{cook-torrance} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}$$

Fresnel and microfacet interaction describe how specular light is reflected

The big one:

Lo stands for Light out

The last part is quite complex for area lights, but for point lights collapses nicely to:
lightCol * falloff * NdotL

$$L_o(p, \omega_o) = \int\limits_{\Omega} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) n \cdot \omega_i \, d\omega_i$$

fr is our BRDF
(Lambert+Cook-Torrance)

!

Integral over OMEGA?!

(it’s just math telling us to do this for every light)

The big one

$$L_o(p,\omega_o) = \int\limits_{\Omega} f_r(p,\omega_i,\omega_o) L_i(p,\omega_i) n \cdot \omega_i \, d\omega_i$$

```
float3 light = float3(0,0,0);
for (uint i = 0; i < MAX_LIGHTS; ++i){
        float3 F = fresnelSchlick(…);
        kS = F;
        kD = (1-kS);
        float3 diffuse = (albedo * kD) / PI;
        float3 specular = cookTorrance(…);
        float NdotL = dot(N, L);
        float3 radidance = lightColor[i] * falloff(…);
        light += (diffuse + specular) * radiance * NdotL;
}
```