A *light*...

Introduction to Real-Time Physically Based Rendering

Agenda

- Intro and a bit of history...
- Physics of light
- Shading model
- Material model, glTF, Implementation
- References

History

Phong/Blinn-Phone → lack of expressiveness (“plastic look”), parameters not intuitive for artists.

Physically-based Shading → more expressive and realistic, parameters based on physical properties.

SIGGRAPH 2012 → *“Practical Physically Based Shading in Film and Game Production”*
SIGGRAPH 2013+ → *“Physically Based Rendering in Theory and Practice”* + *“Real Shading in Unreal Engine 4”*

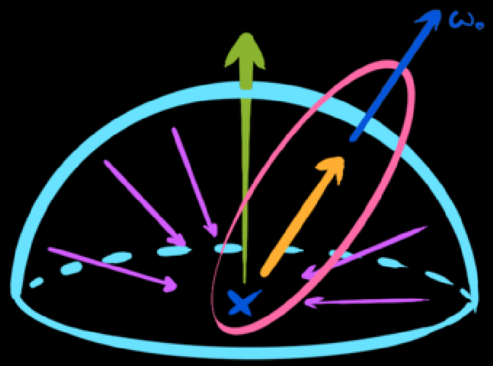
- 2014, 2015, 2016, 2017, 2020, ...

PBR Sample Models

Context: The Reflectance Equation

$$L_o(p,\omega_o) = \int_\Omega f(p,\omega_o,\omega_i)\, L_i(p,\omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i$$

Today's topic



Drawing: Chuck LePlant

Physics of Light

Light Through Media

Media → refractive index:

$$\bar{n} = n + ik$$

Phase velocity

Absorption

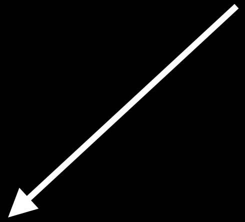
Mostly concerned with this part.

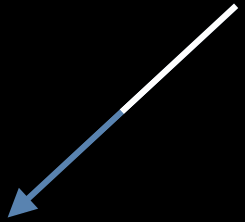
Homogeneous Media

<u>Homogeneous media</u>
- does not change light direction
- may absorb light

Absorption at different wavelengths
→ change of colour.

Longer distance → greater absorption.





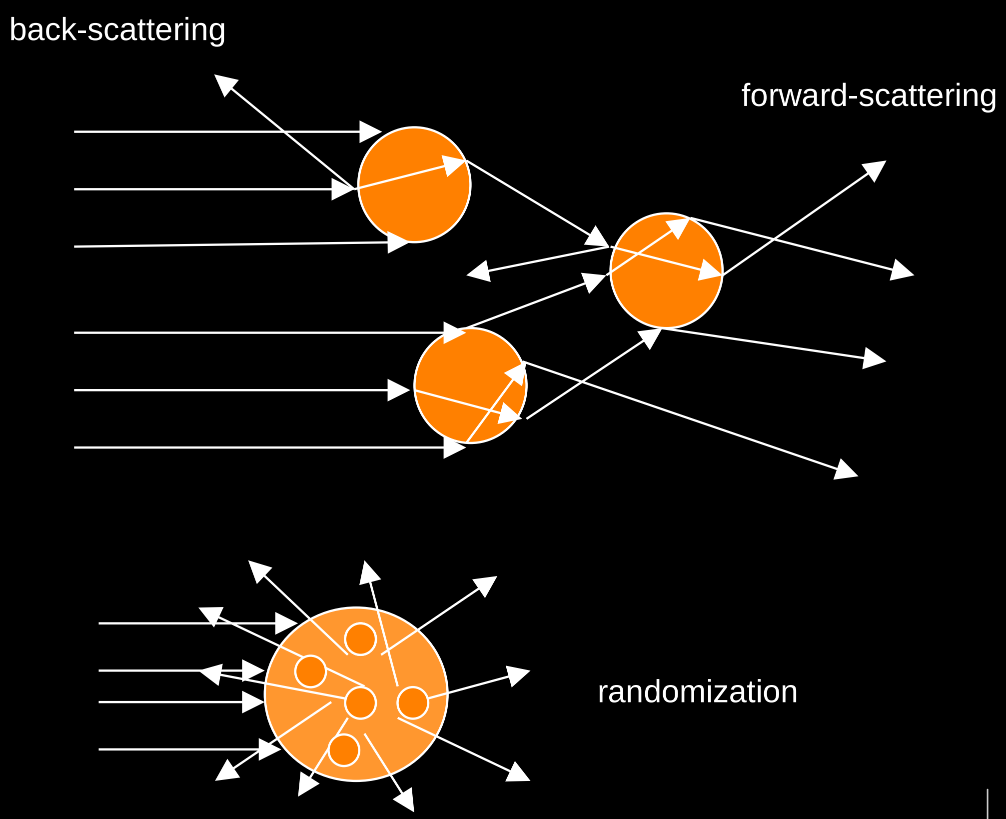
Heterogeneous Media

<u>Heterogeneous media</u> presents variations in the refractive index

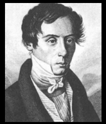
→ changes light direction & may also absorb

Abrupt changes in refractive index (over distances < light wavelength)

→ scattering



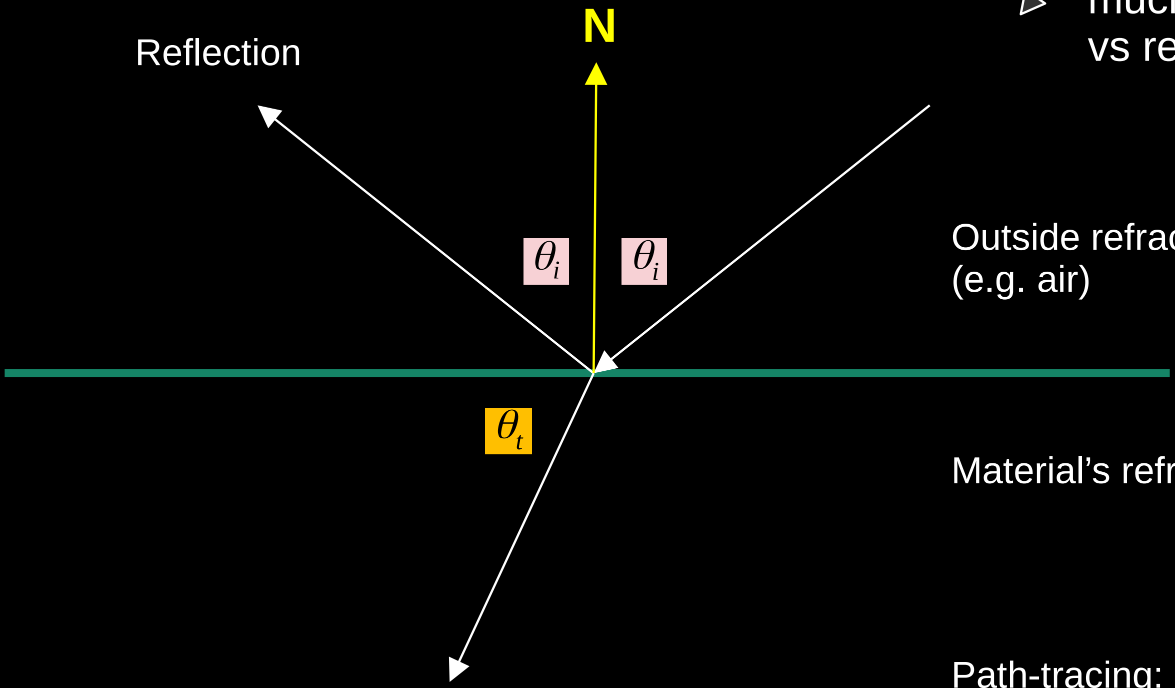
Planar Surfaces



Light splits in exactly two directions.

Angle reflection = Angle incoming

Angle of refraction is generally different (Snell's Law).



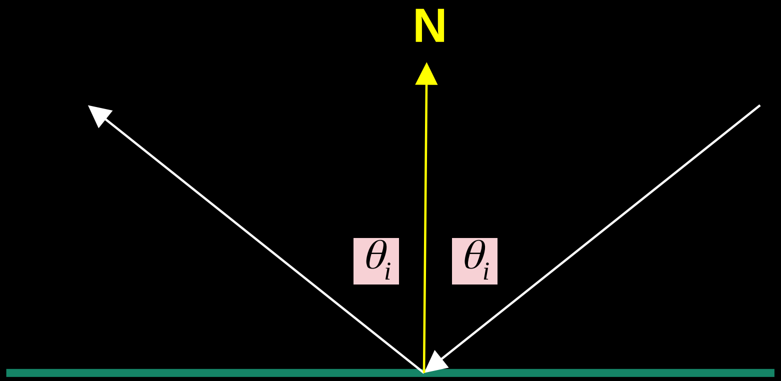
Fresnel tells us how much light is reflected vs refracted.

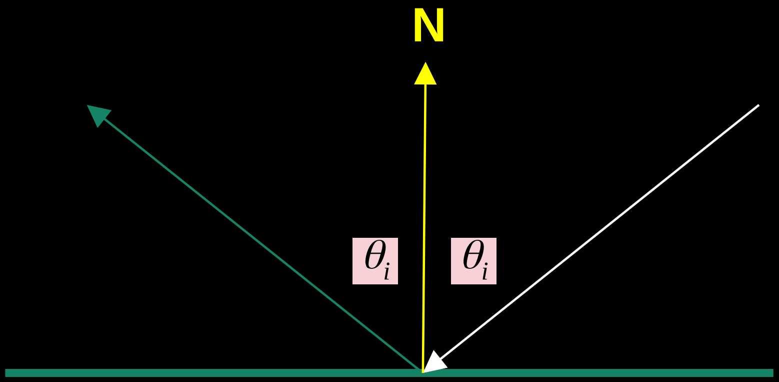
Outside refractive index (e.g. air)

Material's refractive index

Path-tracing: probability
Shading: weight

Reflection

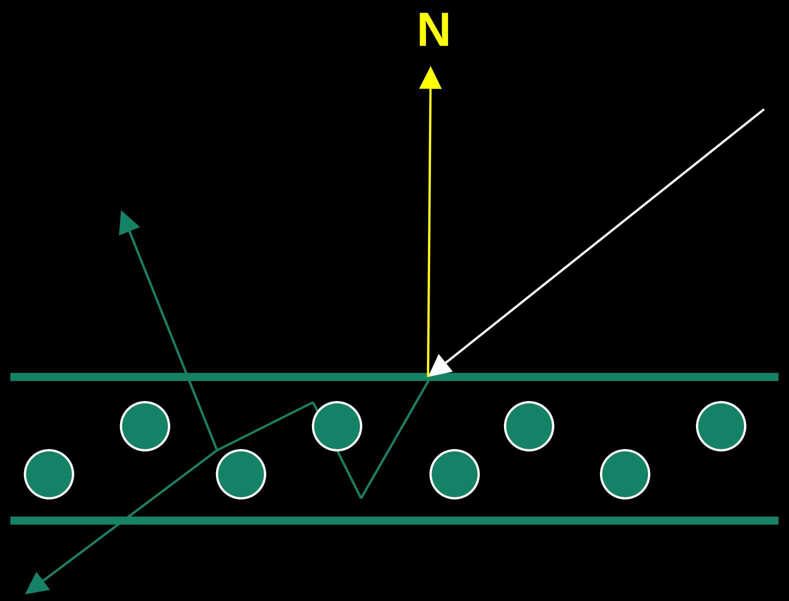


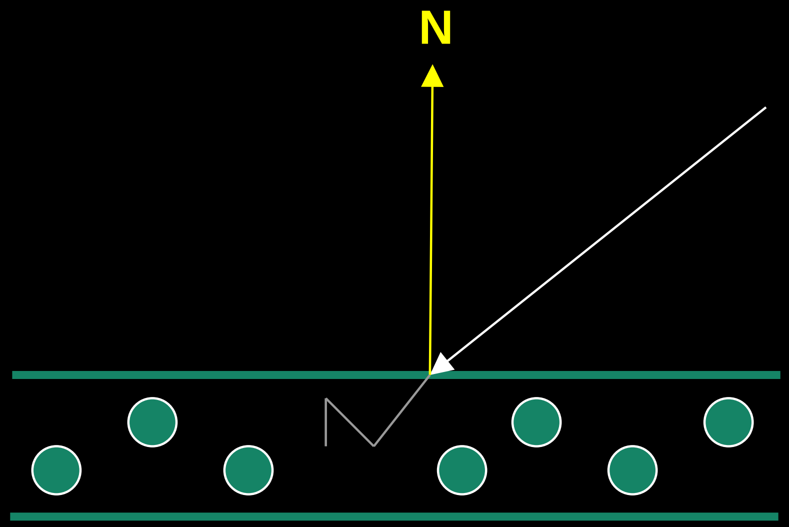


Dielectrics → reflection is NOT tinted
(light has not penetrated the surface)

Metals (conductors) → reflection is tinted
(e.g., gold → yellow)

Refraction

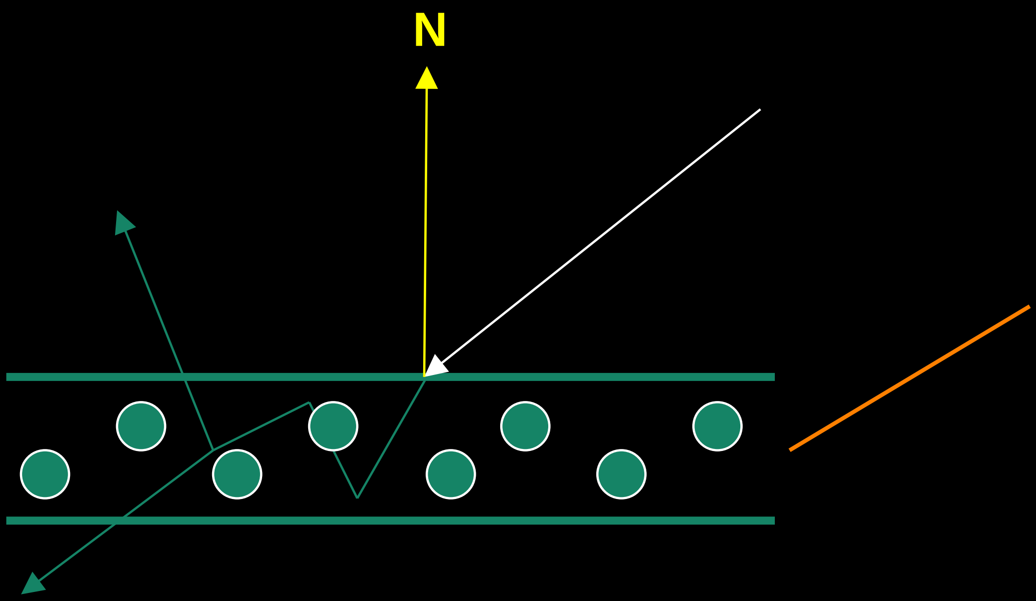




Dielectrics → refracted light is tinted, scatters back (diffuse) or forward (transmission)

Metals (conductors) → refracted light is absorbed (by free electrons)

Refraction



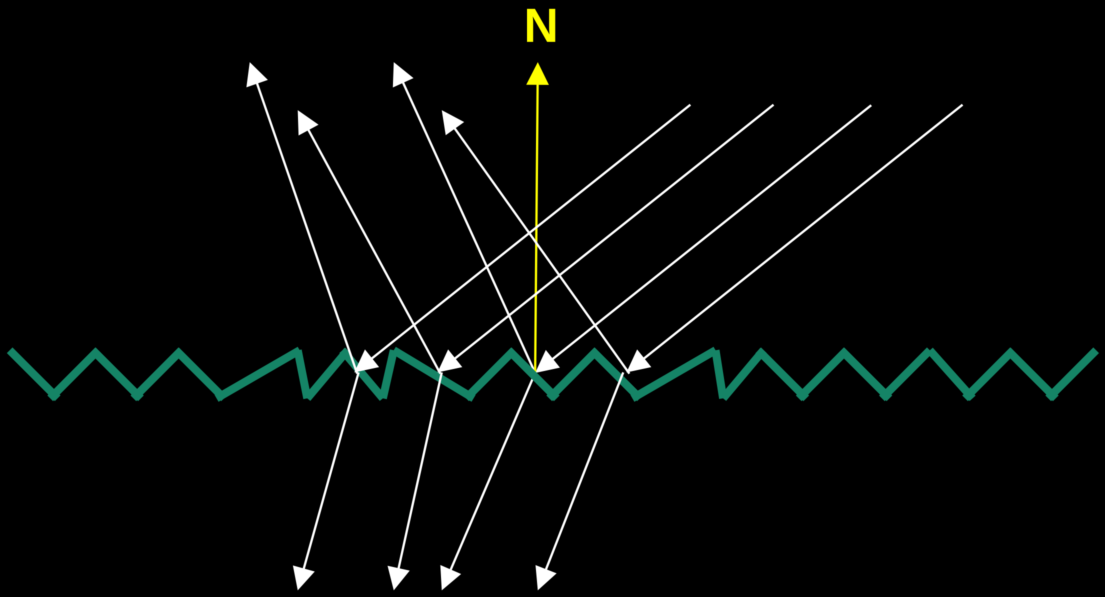
Inside a dielectric, light interacts with molecules as we have seen previously:

- Forward/back scattering.
- Absorption, changing colour if different across wavelengths.

Dielectrics → refracted light is tinted, scatters back (diffuse) or forward (transmission)

General Surfaces

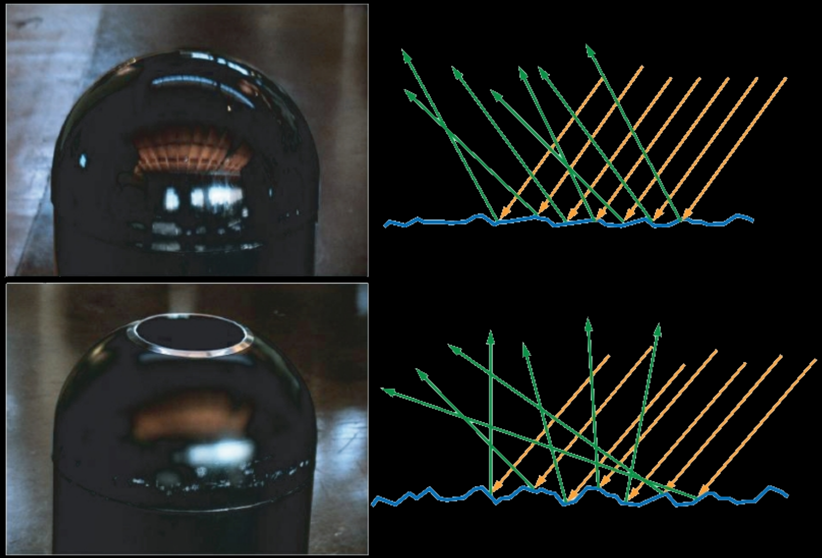
We model general surfaces as a set of ***microfacets***, each optically flat.



Microfacet size < pixel, but large enough to alter appearance.

General Surfaces

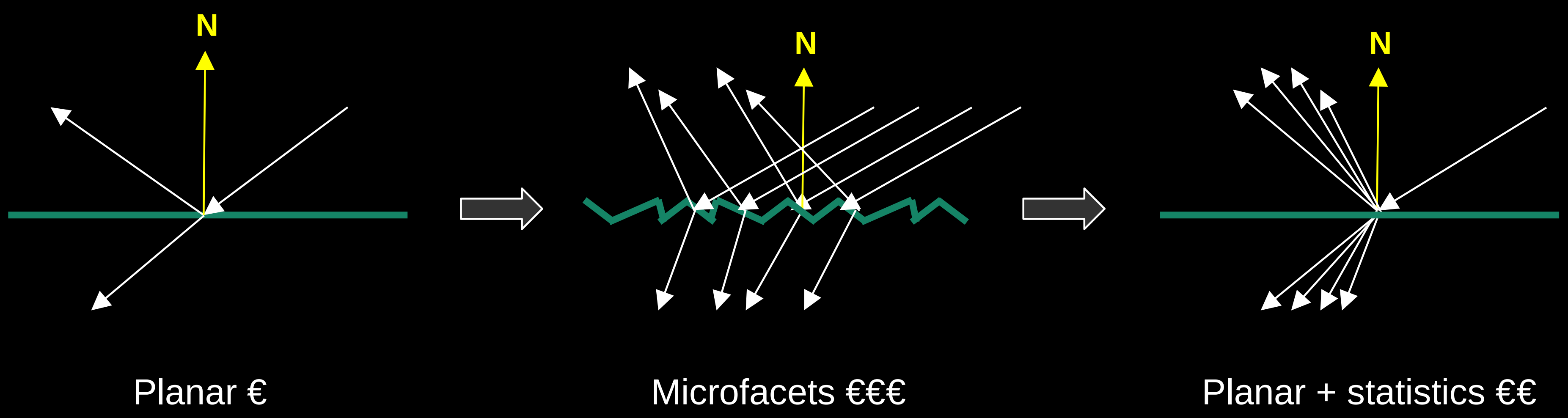
Rougher surface
→ more chaotic reflections



Source: Real-Time Rendering, 3rd edition

General Surfaces

Use statistics to make computationally feasible:

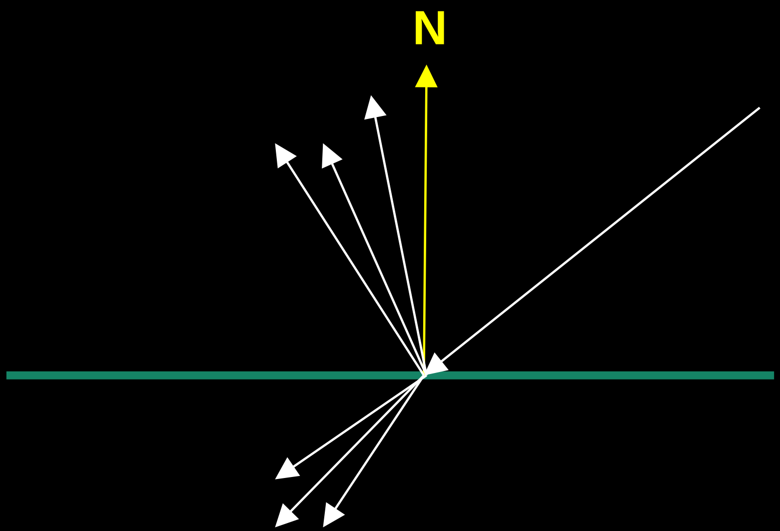


Statistical View

At the macroscopic level, we can think of every light ray reflecting/refracting into one of many possible directions (density function):

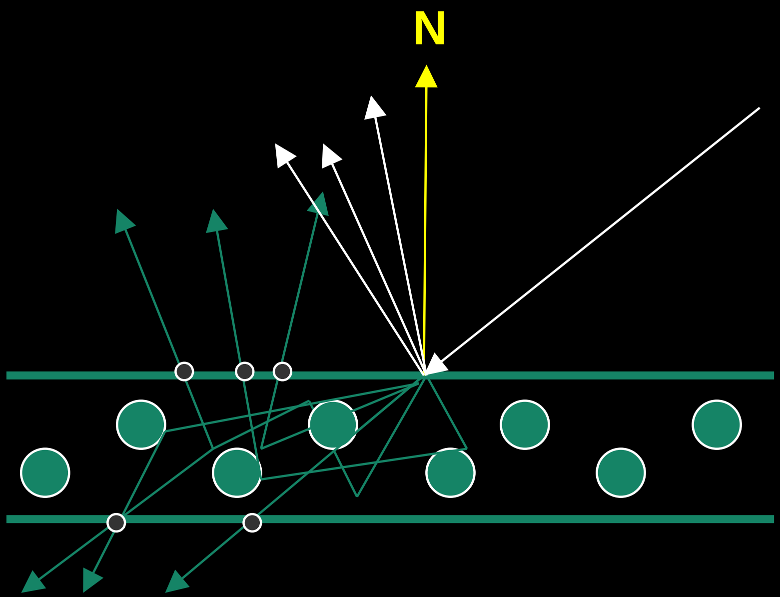
Ray tracing: sample & shoot

Shading: % contribution



Subsurface Scattering

For dielectrics, refracted light bounces and forward/back-scatters through several exit points:

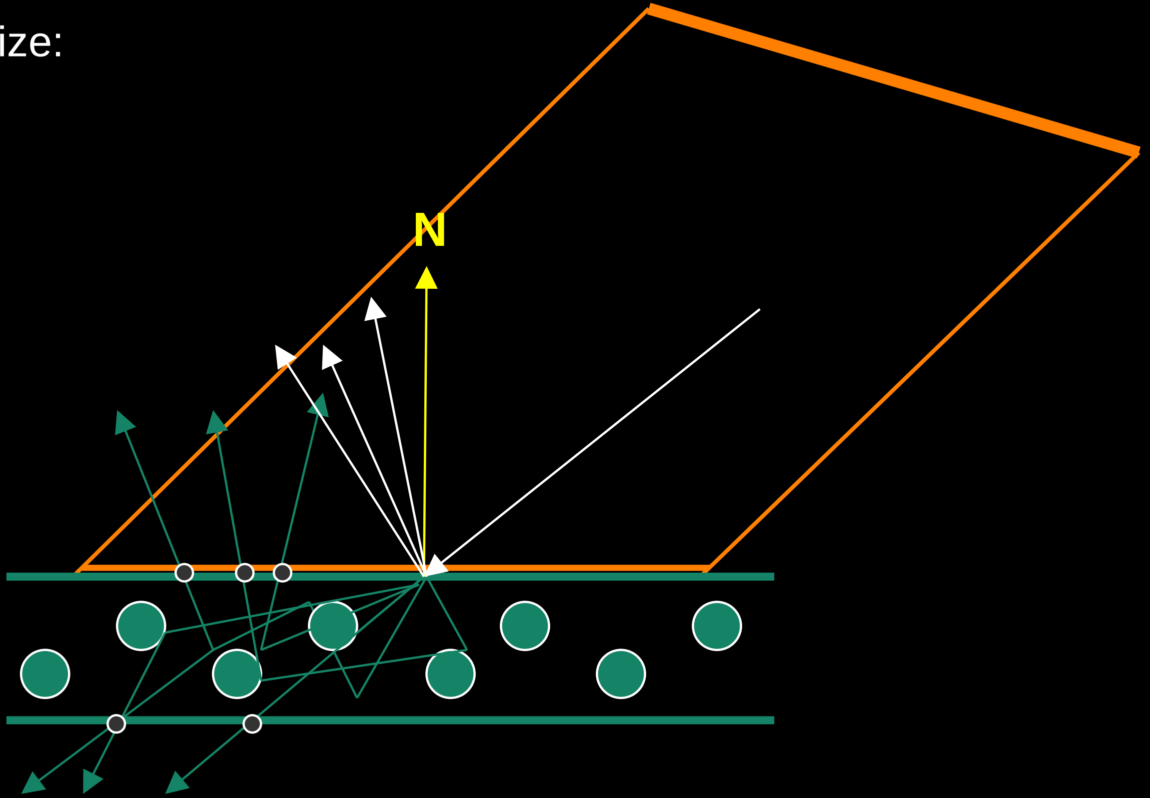


Subsurface Scattering

Assume scatter area < pixel size:

Makes shading local to the point.

Materials like skin and cloth do exhibit subsurface scattering when viewed up close and need to be handled separately.

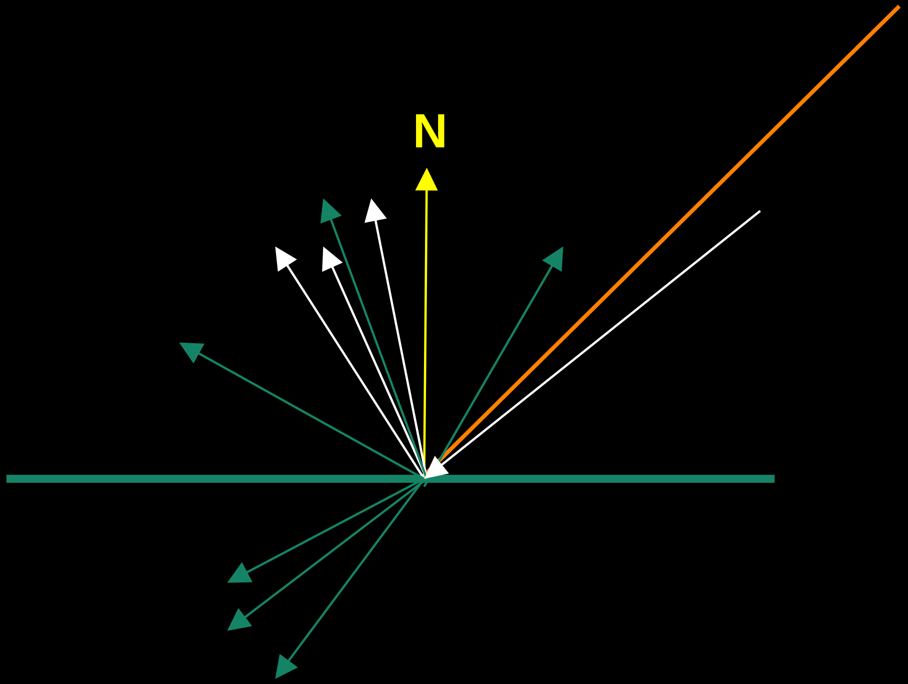


Subsurface Scattering

Assume scatter area < pixel size:

Makes shading local to the point.

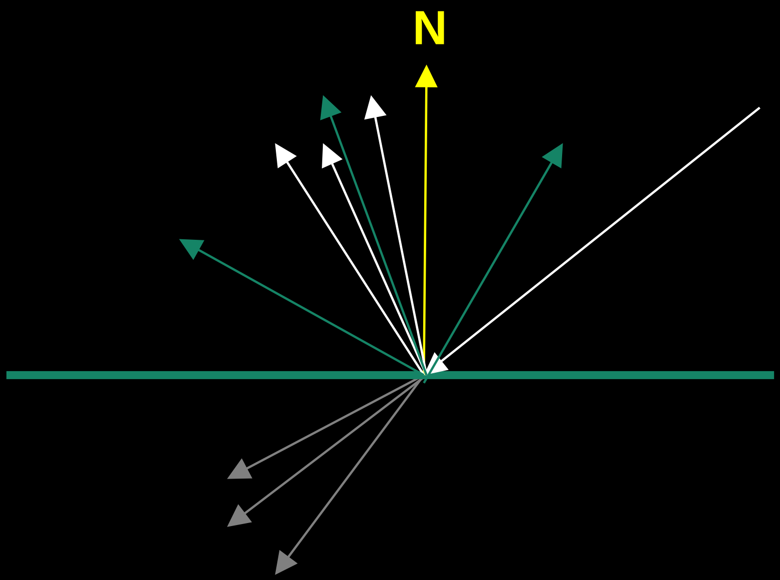
Materials like skin and cloth do exhibit subsurface scattering when viewed up close and need to be handled separately.



No Transmission

Assume no transmission (handle separately):

(We will focus on a BRDF-based model. Transmission is handled by a BTDF or BSDF.)



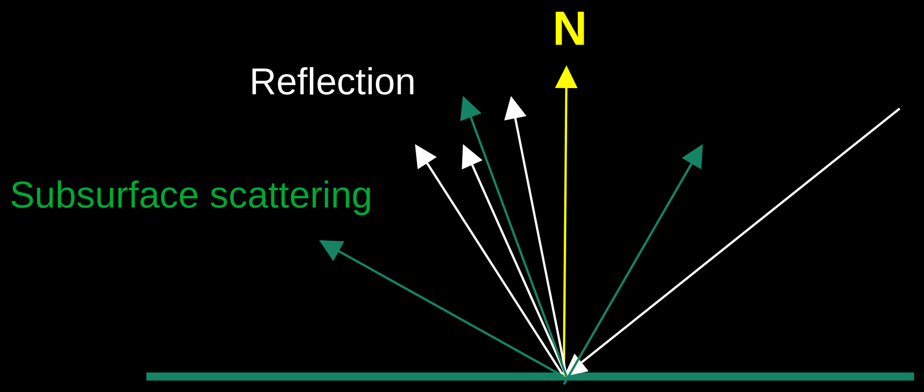
Final Model

Back to the good life:

100% local

Distribution for reflection.

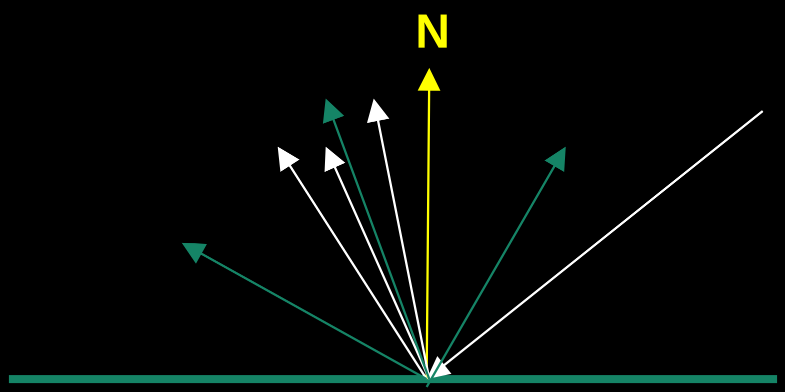
Distribution for scattering.

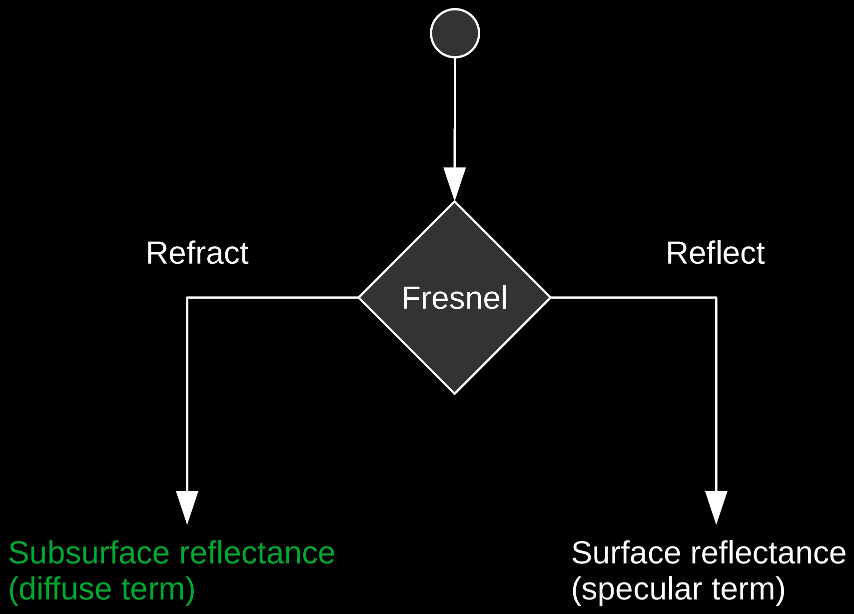


Shading Model (math)

Shading Model

BRDF = Diffuse + Specular, each modulated (shading) or conditioned (path tracing) by Fresnel.





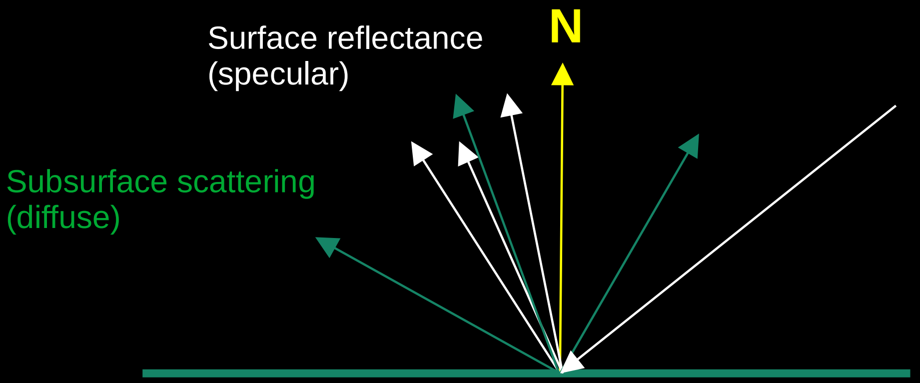
Cook-Torrance BRDF

$$f_r = k_d F_{Lambert} + k_s F_{Cook-Torrance}$$

We will look at the widely-used Cook-Torrance BRDF.

Many alternatives exist resulting from picking different diffuse and specular terms.

Quality-performance trade-off.



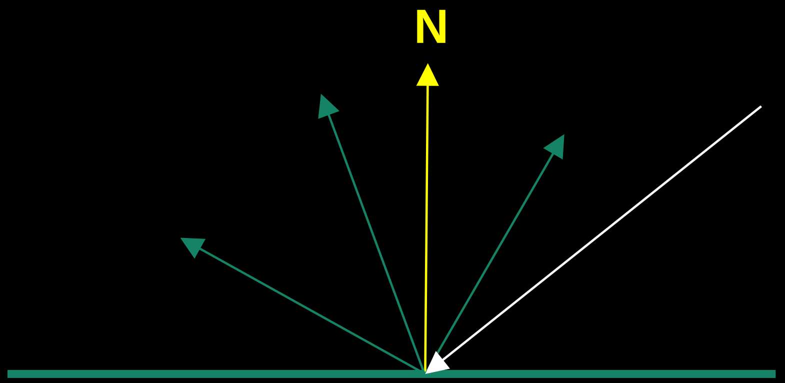
Diffuse Term

Use a Lambertian BRDF. More complex alternatives often make only a very subtle quality difference.

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

Albedo (surface colour)
RGB in [0,1]

Normalization term
(energy conservation)



Remember that here we are simulating light that penetrates the surface and bounces back (tinted; dielectrics only).

Specular Term

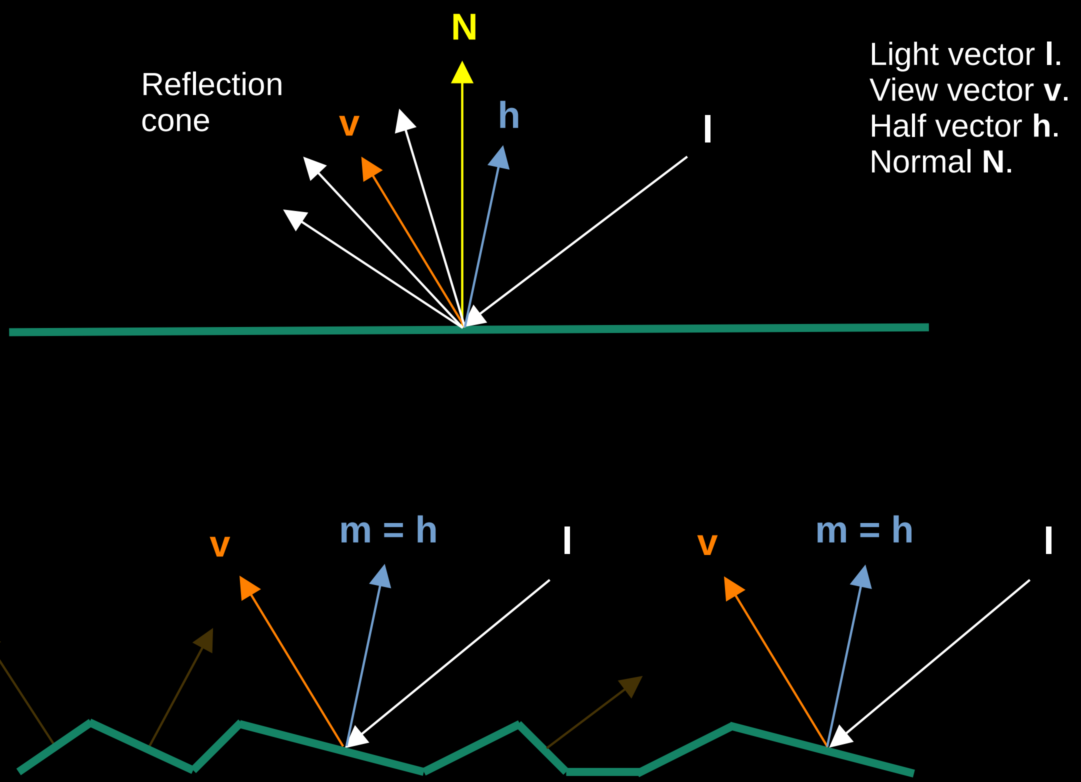
Based on microfacet theory.

Surface is modeled as a set of microfacets.

Each microfacet is optically flat.

In shading, **l**,**v** are given; **v** is the reflected ray of interest.

Only microfacets with microgeometry normal **m = h** contribute to reflection.



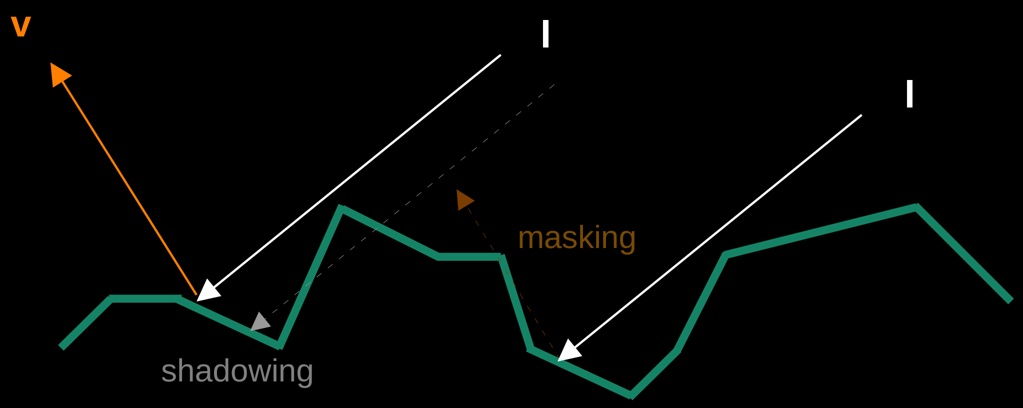
Specular Term

Based on microfacet theory.

Not all microfacets with microgeometry normal **m = h** contribute to reflection.

Light may be blocked from the direction of **l** (**shadowing**) or **v** (**masking**).

Inter-reflections not accounted for in microfacet theory.



Specular Term

Cook-Torrance specular term:

<u>NDF</u>: surface area of microfacets with microgeometry normal **m = h**.

<u>G</u>: % of microfacets with **m = h** that are <u>neither</u> shadowed nor masked.

<u>F</u>: Fresnel → % of reflected vs refracted light.

Fresnel

Geometry function

$$f(l,v) = \frac{D(h)F(v,h)G(l,v,h)}{4(n \cdot l)(n \cdot v)}$$

Normal distribution function (NDF)

Normalization term for Geometry function

Normal Distribution Function

The NDF gives us a statistical distribution of surface point orientations.

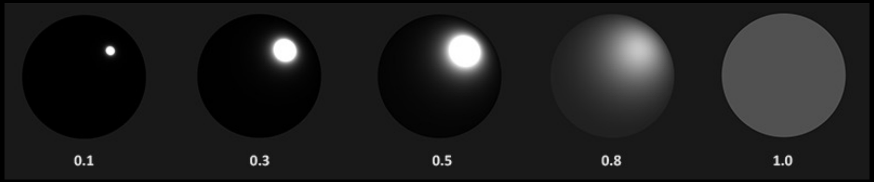
In microfacet theory, the NDF gives us the relative surface area of microfacets with microgeometry normal **m = h** given **h** and surface roughness.

Trowbridge-Reitz GGX is a popular choice:

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

Surface roughness

Source: learnopengl.com



Surface roughness

Geometry Function

The geometry function gives us the probability that microfacets with microgeometry normal **m** are visible from both **l** and **v** given surface roughness.

Smith's method with a Schlick GGX is a popular choice:

Source: learnopengl.com

Surface roughness

Masking

Shadowing

$$G(n,v,l,k) = G_{SchlickGGX}(n,v,k)\,G_{SchlickGGX}(n,l,k)$$

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

*Does not depend on **h**, but **n**.*
See references.

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

Fresnel

Fresnel determines % of light that is reflected / refracted.

Implementations commonly use the Fresnel-Schlick approximation:

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

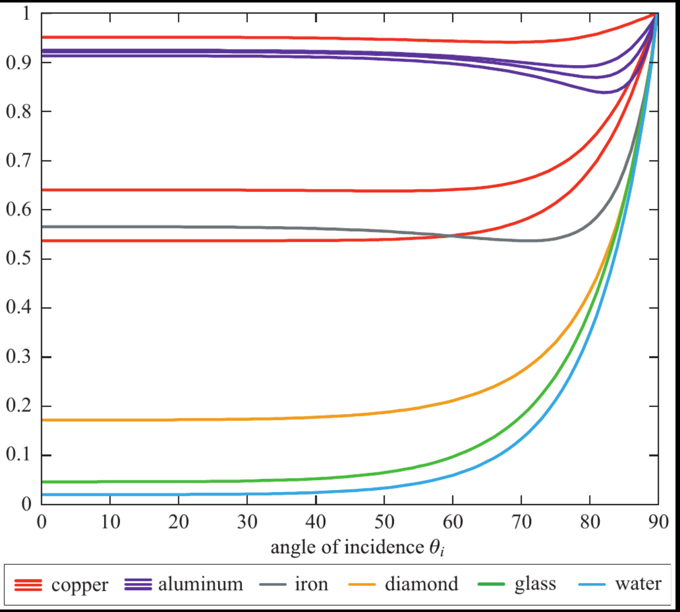
Approximation based on specular color instead of index of refraction.

Fresnel reflectance at normal incidence, or specular color, RGB in [0,1]

Equivalently, **h · l**, angle of incidence

Fresnel

F0 is deemed a sufficiently-good approximation of Fresnel reflectance at any angle and for a variety of materials. It is also referred to as the specular color of the surface.

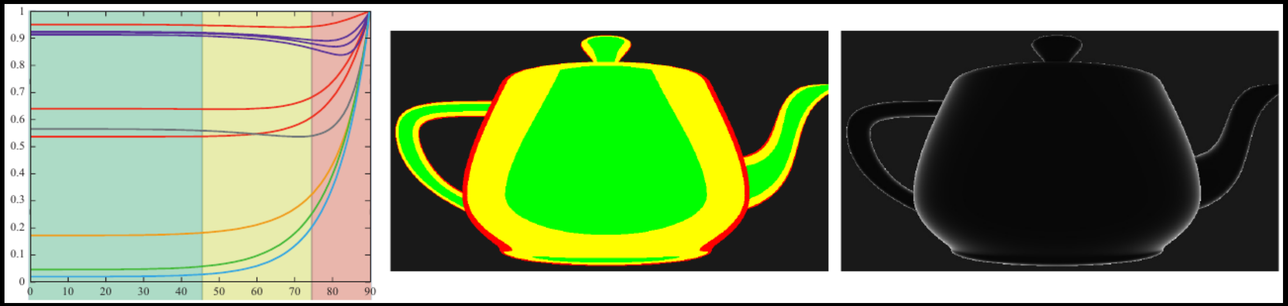


Material	F_0 (Linear)	F_0 (sRGB)	Color
Water	0.02,0.02,0.02	0.15,0.15,0.15	
Plastic / Glass (Low)	0.03,0.03,0.03	0.21,0.21,0.21	
Plastic High	0.05,0.05,0.05	0.24,0.24,0.24	
Glass (High) / Ruby	0.08,0.08,0.08	0.31,0.31,0.31	
Diamond	0.17,0.17,0.17	0.45,0.45,0.45	
Iron	0.56,0.57,0.58	0.77,0.78,0.78	
Copper	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	
Aluminum	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	

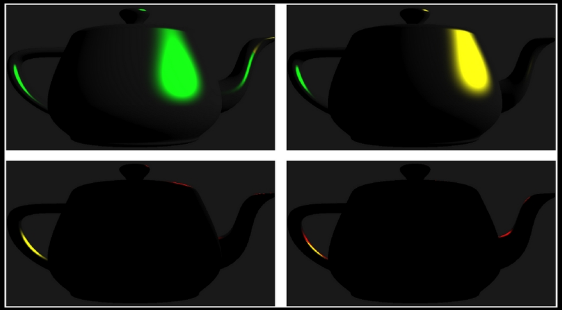
Source: Real-Time Rendering, 3rd edition

Fresnel

Fresnel reflectance changes mostly at angles beyond 75°, but these are a minority of the pixels:



Smooth metallic surface, angle n,v.
Source: PBS course.



Rough metallic surface, angle h,v.
Fresnel combined with other BRDF terms, NDF and geometry function.
Source: PBS course.

Image-Based Lighting (IBL)

Everything we have seen so far is *for a given light direction.*

N analytical lights → evaluate equations N times.

How should we handle environment lights?

→ In principle, sample and evaluate equations N times. (e.g., path tracing)
→ Too expensive for real-time rendering.

$$L_o(p, \omega_o) = \int_\Omega (k_d \frac{c}{\pi} + k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}) L_i(p, \omega_i) n \cdot \omega_i d\omega_i$$

N times

Image-Based Lighting (IBL)

Take the diffuse and specular components apart and handle separately:

$$L_o(p,\omega_o) = \int_\Omega (k_d \frac{c}{\pi}) L_i(p,\omega_i) n \cdot \omega_i d\omega_i + \int_\Omega (k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)}) L_i(p,\omega_i) n \cdot \omega_i d\omega_i$$

Diffuse IBL

Specular IBL

Diffuse IBL

Diffuse BRDF is a constant; does not depend on light or view directions:

$$L_o(p, \omega_o) = k_d \frac{c}{\pi} \int_{\Omega} L_i(p, \omega_i) n \cdot \omega_i d\omega_i$$

Comes out

Irradiance map

Diffuse IBL

Apply convolution to pre-compute irradiance map:

Input environment map (radiance) | Output irradiance map

$$L_o(p,\phi_o,\theta_o) = k_d \frac{c\pi}{n1n2} \sum_{\phi=0}^{n1} \sum_{\theta=0}^{n2} L_i(p,\phi_i,\theta_i)\cos(\theta)\sin(\theta)d\phi d\theta$$

Source: learnopengl.com

Specular IBL: Split-Sum Approx.

Specular BRDF depends on both light and view directions; cannot really pre-convolute:

$$L_o(p,\omega_o) = \int_\Omega \left(k_s \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} L_i(p,\omega_i) n \cdot \omega_i d\omega_i = \int_\Omega f_r(p,\omega_i,\omega_o) L_i(p,\omega_i) n \cdot \omega_i d\omega_i\right.$$

But do it anyway. Apply Epic Games' split-sum approximation:

$$L_o(p,\omega_o) = \int_\Omega L_i(p,\omega_i) d\omega_i * \int_\Omega f_r(p,\omega_i,\omega_o) n \cdot \omega_i d\omega_i$$

Pre-filtered environment map — BRDF integration map

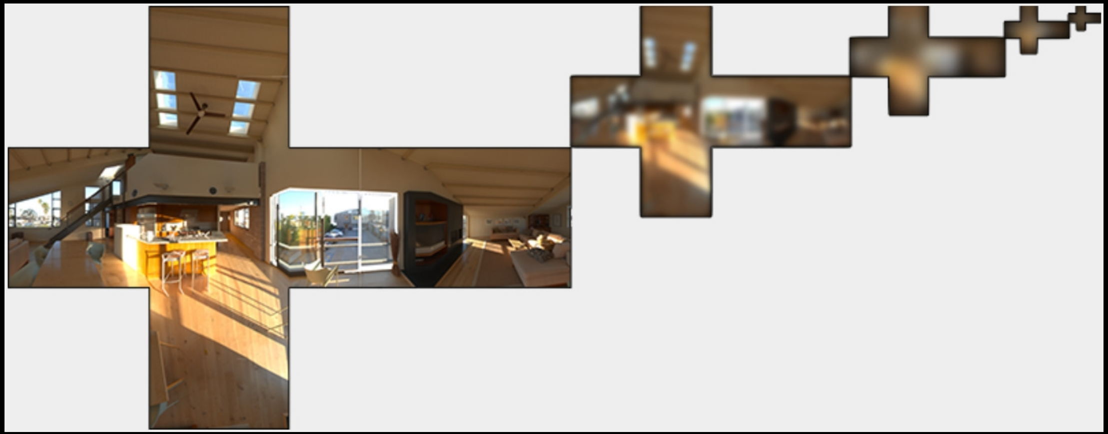
Specular IBL: Pre-filtered Env Map

The pre-filtered environment map is similar to the irradiance map, but takes roughness into account.

Each mip level encodes the sum of incoming light for cones of a given angle based on surface roughness.

Samples the NDF to generate light directions.

Assumes $w_o = w_i \equiv v = l$ to make the computation feasible given that **v** is unknown beforehand.



Source: learnopengl.com

Higher mip → rougher surface → larger angle.

Specular IBL: Pre-filtered Env Map

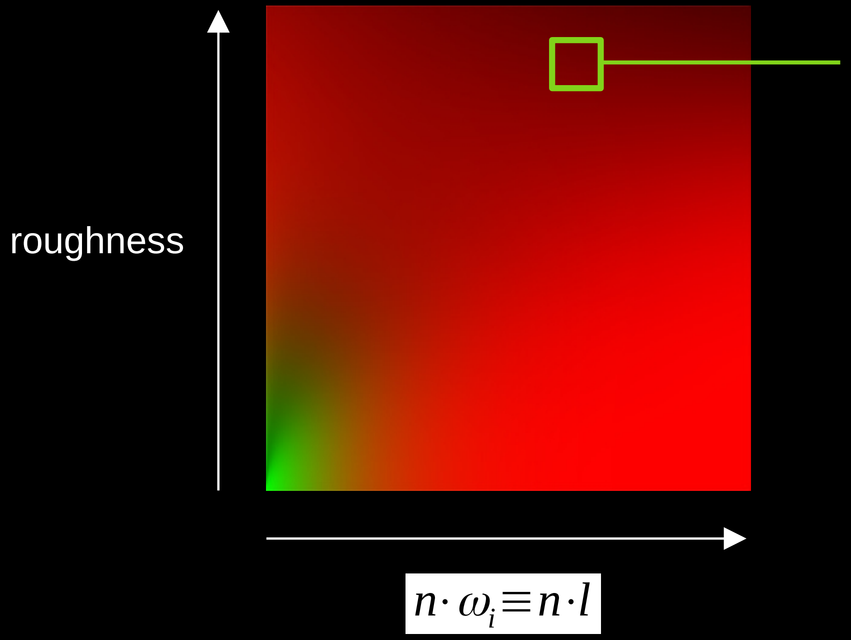
The assumption $w_o = w_i \equiv v = l$ means we don't get sharp specular reflections at grazing angles.

A relatively small price to pay for computational feasibility.

Source: learnopengl.com, "Moving Frostbite to PBR"

Specular IBL: BRDF Integration Map

BRDF integration map:



BRDF response given surface roughness and input light direction (light-normal angle).

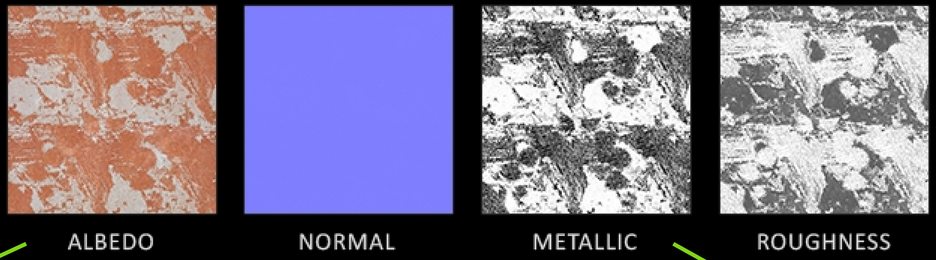
Red: scale
Green: bias

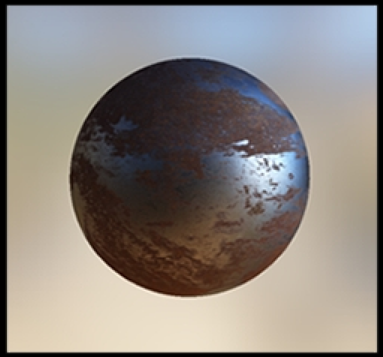
Scale & bias transform the Fresnel response. Full derivation in *“Real Shading in Unreal Engine 4”* (SIGGRAPH 2013).

Implementation

Material Model

The metal-roughness workflow is almost a direct representation of the theory described so far:





Dielectrics: albedo
Metals: specular color (F0)

Binary in principle.
Values in [0,1] in practice to simulate mixtures.

Source: learnopengl.com

Fresnel

For non-metals, a specular color (F0) of 0.04 is often used as an approximate average of F0 values across various materials (scalar value; specular not tinted for non-metals.)

non-metals

0.04 won't work well for diamond, semiconductors, and other exotic materials. Hopefully your application doesn't have too many of those.

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	
Aluminum	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	

Source: Real-Time Rendering, 3rd edition

glTF

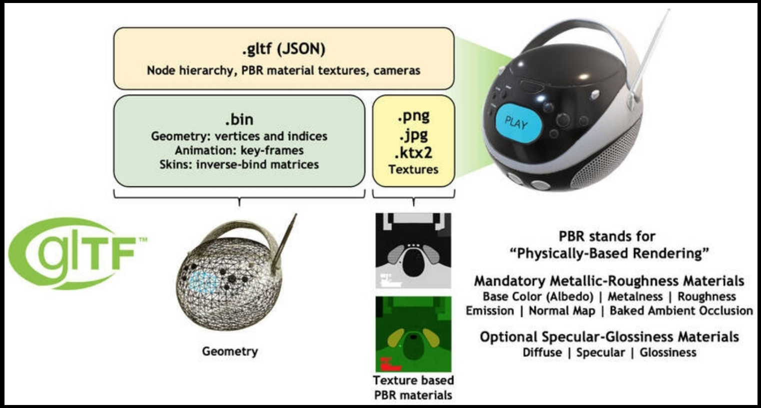
A standard asset format from Khronos.

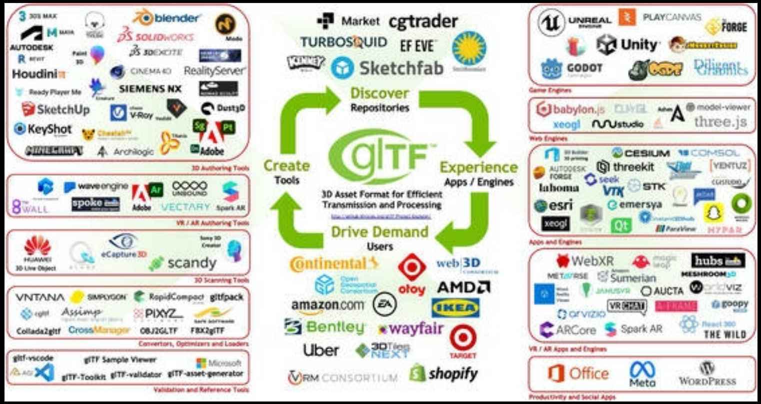
First-class support for physically-based rendering (PBR).

Widespread support across tools.

Sample models: https://github.com/KhronosGroup/glTF-Sample-Models

Reference renderer: https://github.com/KhronosGroup/glTF-Sample-Viewer





Analytical Lights

For point and directional lights, we get a relatively straightforward implementation of everything we have seen so far:

```
float trowbridge_reitz_GGX(float roughness, float NdotH) {
  float a = roughness * roughness;
  float a2 = a * a;
  float d = NdotH * NdotH * (a2 - 1.0) + 1.0;
  return a2 / (PI * d * d);
}

float geometry_schlick_GGX(float k, float NdotV) {
  return NdotV / (NdotV * (1.0 - k) + k);
}

float geometry_smith(float roughness, float NdotL, float NdotV) {
  float a = roughness + 1;
  float k = a*a / 8.0; // Analytical light.
  return geometry_schlick_GGX(k, NdotV) * geometry_schlick_GGX(k, NdotL);
}

vec3 fresnel_schlick(vec3 F0, float HdotV) {
  return F0 + (1.0 - F0) * pow(clamp(1.0 - HdotV, 0.0, 1.0), 5.0);
}
```

Different term for IBL

Analytical Lights

For point and directional lights, we get a relatively straightforward implementation of everything we have seen so far:

```
const float nonMetalF0 = 0.04;

// Cook-Torrance BRDF for a single light direction.
vec3 cook_torrance(
    vec3 albedo, float metallic, float roughness, vec3 emissive,
    float NdotL, float NdotV, float NdotH, float HdotV) {
  vec3 F0 = mix(vec3(nonMetalF0), albedo, metallic);
  float D = trowbridge_reitz_GGX(roughness, NdotH);
  vec3  F = fresnel_schlick(F0, HdotV);
  float G = geometry_smith(roughness, NdotL, NdotV);
  vec3 Kd = mix(vec3(1.0) - F, vec3(0.0), metallic);
  vec3 diffuse = Kd*albedo*INV_PI;
  // Take a max to prevent division by 0 when either dot product is 0.
  vec3 specular = (D*F*G) / max(4.0 * NdotV * NdotL, 0.0001);
  return emissive + diffuse + specular;
}
```

“albedo” is actually the specular color, or F0, for metals

Metallic surfaces do not have diffuse reflection.

Image-Based Lighting (IBL)

For IBL, we use the different versions of Geometry and Fresnel functions:

```
float geometry_smith(float roughness, float NdotL, float NdotV) {
  float k = roughness * roughness / 2.0; // IBL
  return geometry_schlick_GGX(k, NdotV) * geometry_schlick_GGX(k, NdotL);
}

vec3 fresnel_schlick_roughness(vec3 F0, float NdotV, float roughness) {
  return F0
    + (max(vec3(1.0 - roughness), F0) - F0)
    * pow(clamp(1.0 - NdotV, 0.0, 1.0), 5.0);
}
```

Different term for IBL

Corrects for the fact that, in IBL, we do not have a single half vector. Empirically attenuates Fresnel for better results.

Image-Based Lighting(IBL)

Cook-Torrance using IBL:

```
// Cook-Torrance BRDF for IBL.
vec3 cook_torrance_IBL(
    vec3 albedo, float metallic, float roughness, vec3 emissive,
    float NdotV,
    vec3 irradiance, vec3 prefiltered_env, vec2 BRDF_env, vec3 ambient) {
  vec3 F0 = mix(vec3(nonMetalF0), albedo, metallic);
  vec3  F = fresnel_schlick_roughness(F0, NdotV, roughness);
  vec3 Kd = mix(vec3(1.0) - F, vec3(0.0), metallic);
  vec3 diffuse = Kd * albedo * irradiance;
  vec3 specular = prefiltered_env * (F * BRDF_env.x + BRDF_env.y);
  return emissive + ambient + diffuse + specular;
}
```

Diffuse IBL

Irradiance map:

```
vec3 N = normalize(Ray);
vec3 B = (abs(N.x) - 1.0 <= EPS) ? vec3(0.0, 0.0, 1.0) : vec3(1.0, 0.0, 0.0);
vec3 T = cross(B, N);
B = cross(N, T);

int num_samples = 0;
vec3 irradiance = vec3(0.0);
for (float theta = 0.0; theta < MAX_AZIMUTH; theta += AZIMUTH_DELTA) {
  for (float phi = 0.0; phi < MAX_ZENITH; phi += ZENITH_DELTA) {
    // Spherical to Cartesian.
    vec3 sample_tangent_space = vec3(
      sin(phi) * cos(theta),
      sin(phi) * sin(theta),
      cos(phi));
    // Tangent space to world space.
    vec3 sample_world_space =
      sample_tangent_space.x * B +
      sample_tangent_space.y * T +
      sample_tangent_space.z * N;

    irradiance += texture(Sky, sample_world_space).rgb * sin(phi) * cos(phi);
    num_samples += 1;
  }
}
irradiance = PI * irradiance / float(num_samples);

Color = vec4(irradiance, 1.0);
```

Specular IBL

Pre-filtered environment map:

```
vec3 irradiance = vec3(0.0);
float total_weight = 0.0;
for (uint i = 0; i < NUM_SAMPLES; ++i) {
  vec2 sample_box = hammersley(i, NUM_SAMPLES);
  vec3 H = importance_sample_GGX(sample_box, N, Roughness);
  vec3 L = reflect(-V,H);

  float NdotL = max(dot(N,L), 0.0);
  if (NdotL > 0.0) {
    irradiance += texture(Sky, H).rgb * NdotL;
    total_weight += NdotL;
  }
}
irradiance /= total_weight;

Color = vec4(irradiance, 1.0);
```

2D low-discrepancy sampling.

Sampling the NDF. Takes the 2D sample and maps it to the hemisphere, accounting for specular lobe size.

See learnopengl.com for implementation details.

Specular IBL

BRDF integration map:

```
vec2 integrate_brdf(float NdotV, float roughness)
{
  vec3 V = vec3(sqrt(1.0 - NdotV * NdotV), 0.0, NdotV);
  vec3 N = vec3(0.0, 0.0, 1.0);

  float scale = 0.0;
  float bias = 0.0;
  for (int i = 0; i < NUM_SAMPLES; ++i) {
    vec2 sample_box = hammersley(i, NUM_SAMPLES);
    vec3 H = importance_sample_GGX(sample_box, N, roughness);
    vec3 L = reflect(-V,H);
    float NdotL = max(0.0, L.z);

    if (NdotL > 0.0) {
      float NdotH = max(0.0, H.z);
      float VdotH = max(0.0, dot(V,H));
      float G = geometry_smith(roughness, NdotL, NdotV);
      float G_vis = (G * VdotH) / (NdotH * NdotV);
      float Fc = pow(1.0 - VdotH, 5.0);
      scale += (1.0 - Fc) * G_vis;
      bias  += Fc * G_vis;
    }
  }
  scale /= float(NUM_SAMPLES);
  bias  /= float(NUM_SAMPLES);
  return vec2(scale, bias);
}
```

Same sampling technique here.

See learnopengl.com for the full derivation.

See learnopengl.com for implementation details.

References

https://blog.selfshadow.com/publications/s2013-shading-course/
https://blog.selfshadow.com/publications/s2012-shading-course/

https://learnopengl.com/PBR/Theory
https://learnopengl.com/PBR/Lighting

https://seblagarde.wordpress.com/2011/08/17/hello-world/

http://holger.dammertz.org/stuff/notes_HammersleyOnHemisphere.html

https://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.pdf
https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc2017/Presentations/Hammon_Earl_PBR_Diffuse_Lighting.pdf

https://www.kinematicsoup.com/news/2016/6/15/gamma-and-linear-space-what-they-are-how-they-differ