Physically Based Rendering (PBR)

BRDFs for direct light

A targeted overview

- This is an overview of PBR
- Focusing on one “half”: direct light
  - Light coming directly from light sources
  - Directional/point/spot lights
- Why only half?
  - We can implement it with what we already know
  - And it’s the less complicated half

Physically Based Rendering (PBR)

- Modeling a more physically accurate interaction between light and surfaces
- Sometimes called “Physically Based *Shading*” (PBS)
- Not one specific implementation
  - Engines often handle PBR slightly differently
  - All have the same goals

PBR goals

- More “realistic looking”
- Ensure materials look consistent
  - Under different lighting conditions
  - Across multiple environments
- Ease creation of realistic materials for artists
  - More consistency / predictability
  - Less guess work
  - Fewer “knobs to tweak”

What PBR is (and isn't)

- Light-surface interaction for specific types of surfaces
  - Isotropic – appear the same from all directions
  - Conductors & dielectrics – metals & non-metals
- Can be expanded upon / altered to handle:
  - Anisotropic surfaces
  - Translucency
  - Cloth
- Other effects can be used with PBR:
  - Normal mapping
  - Post-process effects like AO, bloom, depth of field

Rendering and light

- We’ve looked at several BRDFs for light
  - Bidirectional Reflectance Distribution Functions
- BRDFs approximate just one aspect of light
  - How light is reflected at a surface
- What about the rest of “lighting”?
- For that we need The Rendering Equation

“The Rendering Equation”

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) \ + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}}$$

- Introduced in 1986
  - Literally called “The Rendering Equation”
  - Original papers here and here

- Calculates the radiance leaving a point as the sum of emitted and reflected radiance

Deciphering The Rendering Equation

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\omega_{\mathrm{i}}$$

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t)$$

Outgoing light
(spectral radiance)

Position in space

Direction of outgoing light

Light wavelength

Time

Deciphering: Emitted light

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}}$$

$$L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t)$$

Emitted light
(spectral radiance)

Position in space

Direction of outgoing light

Light wavelength

Time

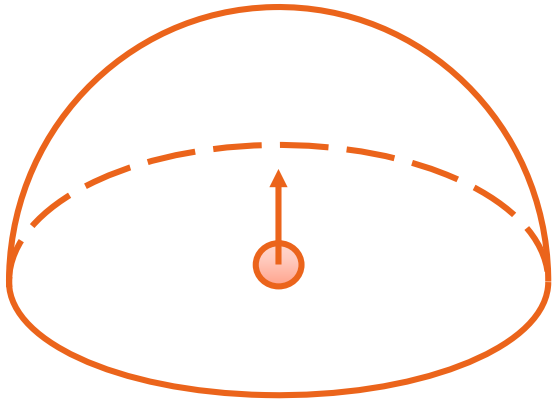
Deciphering: All incoming light

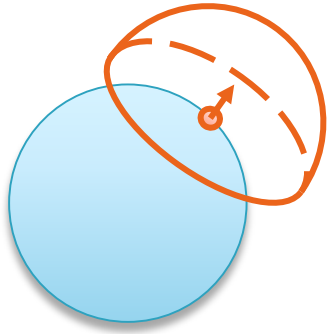
$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}}$$

$$\int_{\Omega}$$

Integral over Ω

Ω is a **hemisphere** encompassing *directions of all incoming light* (ω_i) that can strike this position in space





Deciphering: BRDF

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\omega_{\mathrm{i}}$$

$$f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t)$$

BRDF

Position in space

Negative direction*
of incoming light
(direction TO the light)

* *All* directions over Ω

Direction of outgoing light

Light wavelength

Time

Deciphering: Incoming light

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\omega_{\mathrm{i}}$$

$$L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)$$

Incoming light
(spectral radiance)

Position in space

Negative direction*
of incoming light
(direction TO the light)

Time

Light wavelength

* *All* directions over Ω

Deciphering: The rest

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n})\, \mathrm{d}\,\omega_i$$

$$(\omega_i \cdot \mathbf{n})\, \mathrm{d}\,\omega_i$$

Negative direction*
of incoming light
(direction TO the light)

* *All* directions over Ω

normal

Dot product
(Hey look, it's N dot L!)

Differential of negative
direction of incoming light
(solid angle)

The Rendering Equation for us

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda, t) \; + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda, t) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda, t)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}}$$

- We’re obviously not using the whole thing
- Where are we simplifying?

The Rendering Equation for us

$$L_o(\mathbf{x}, \omega_o, \cancel{\lambda}, \cancel{t}) = \cancel{L_e(\mathbf{x}, \omega_o, \lambda, t)} + \cancel{\int_\Omega} f_r(\mathbf{x}, \omega_i, \omega_o, \cancel{\lambda}, \cancel{t}) L_i(\mathbf{x}, \omega_i, \cancel{\lambda}, \cancel{t})(\omega_i \cdot \mathbf{n}) \, \cancel{d}\omega_i$$

- **Wavelength?** We assume RGB are similar
- **Time?** Single point in time
- **Emitted light?** Nope (but could be emissive texture)
- **Integral** over a hemisphere?
  - WAY too expensive for a shader!
  - We only handle light coming directly from a light source
  - And only a single direction per light
- Don't need that **solid angle** anymore, either

The Rendering Equation – What's left?

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) \, d\omega_i$$

- BRDF
- Incoming light
- N dot L

Taking a step back

- What kinds of light have we been simulating up to this point?
  - “Old School” rendering calculations

- **Diffuse**
- **Specular**
- **Ambient**
- **Reflections**

Old school – Direct light

- Light coming directly from light sources

- **Diffuse**
  - Light that hits surface and reflects in all directions
  - A “diffusion” of light

- **Specular**
  - Light that hits surface and reflects at the viewer
  - The “reflection” of the light source

Old school – Indirect light

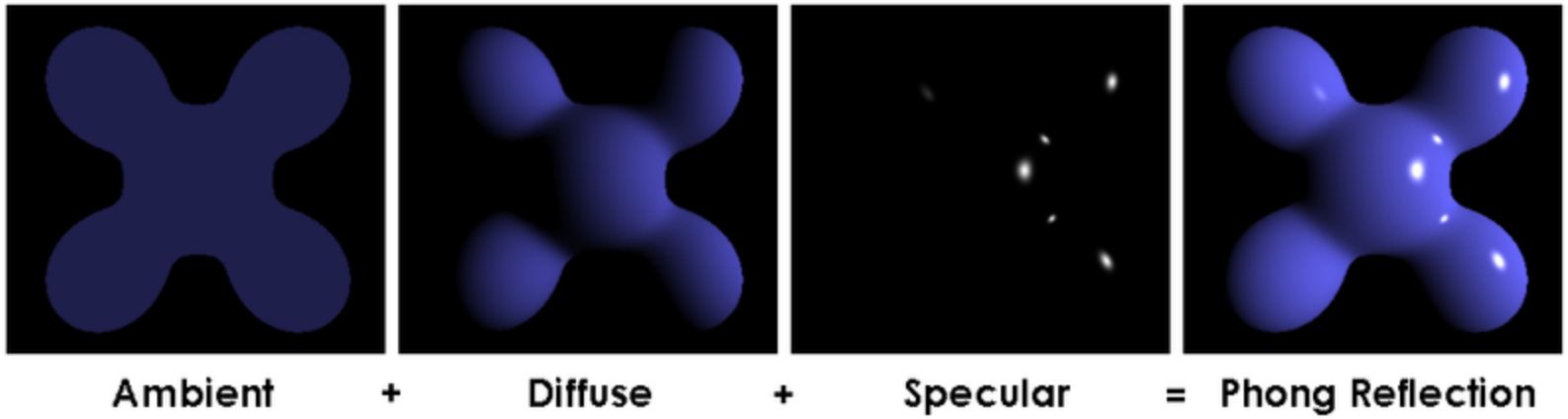
- Light coming in from the environment

- **Ambient**
  - Omnipresent, omnidirectional light
  - Fakes light “bouncing around”
  - A poor man’s *global illumination*

- **Reflections**
  - A mirror-like reflection of the world around us
  - Shiny objects usually reflect the world too

Old school – Light equations

- **Ambient**: Color (usually dim) added to all surfaces
- **Diffuse**: Lambert ($N \cdot L$) – Tints surface color
- **Specular**: Phong or Blinn-Phong – Add to diffuse



- **Reflection**: Environment mapping (cubemaps)

Chart for “old school” BRDFs

	Direct Light (from light sources)	Indirect Light (from “environment”)
Diffuse	Lambert $(N \cdot L)$	Ambient
Specular	Phong	Cubemap Reflections

Old school – Light sources

- Uses punctual light sources
  - “Pertaining to or of the nature of a point”
  - No perfect real-world analogs, but the math is fast!
- Directional Light
  - Simulates an infinitely big “wall of light”
  - Decent for simulating main light source (often sun)
- Point Light – Similar to a light bulb
- Spot Light – Like a flash light

Old school – Surface materials

- Surface (diffuse) color
  - Color of the surface under 100% white light
  - Often other effects baked in, like ambient occlusion
- Specular (gloss) map
  - How “reflective” the surface is, per pixel
  - Potentially RGB texture for colored reflections
- Normal map

Old school issues – Energy

- Objects are unnaturally bright
  - Ambient + Diffuse + Specular?
  - Too much light!

- A surface can't reflect more light (energy) than it receives
  - Law of conservation of energy
  - Technically reflects less – some turns into heat
  - (But that's mostly irrelevant to us)

Old school issues – Asset creation

- Artists have many “knobs” to tweak
  - Ambient & diffuse color, reflectivity, specularity, etc
  - Can be time consuming to get it all correct
  - Lots of guesswork
- Often tweaked for a certain environment
  - Great “daytime” material != great “night” material
- Breaking the laws of physics
  - Final lighting usually physically inaccurate
  - Unless artists are doing math

PBR to the rescue!

- PBR reduces the impact of these issues:
  - More accurate lighting on objects
  - More realistic materials
  - Predictable workflow for artists

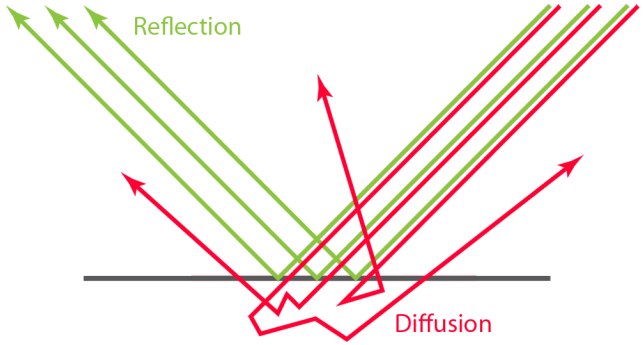
- Still uses punctual light sources
  - Because the math is fast
  - And they mostly do what we need
  - Some engines support real-time area lights

What makes up PBR?

- Physically-Based BRDFs
  - Microfacet model
  - Fresnel term
- Real-world material parameters
- Energy conservation
- Optionally: Indirect lighting calculations
  - Image-Based Lighting
  - Covered in advanced course

Diffusion & reflection

- Diffusion – Light scattering
- Reflection – Light bouncing

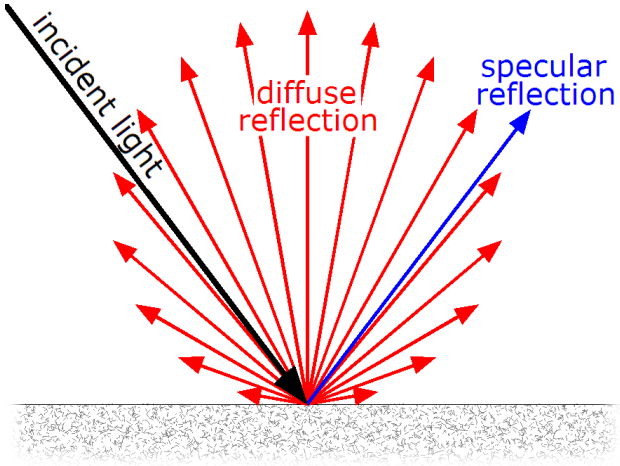


- Need functions to calculate:
  - How light diffuses
  - How light reflects

- Also need to define surface color (albedo)
- And incoming light color

BRDF

- Bidirectional Reflectance Distribution Functions
  - Defines how light is reflected at a surface
- Many different BRDF options exist
  - Each is used for slightly different purposes
- PBR uses them to compute *diffuse* and *specular* terms

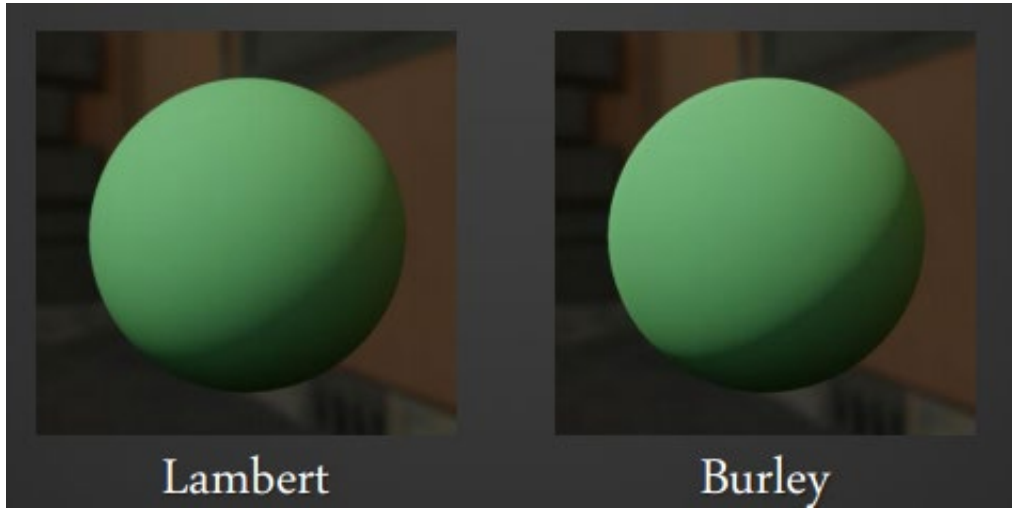


Diffuse BRDF – Lambert

- Good old $N \cdot L$ is back!

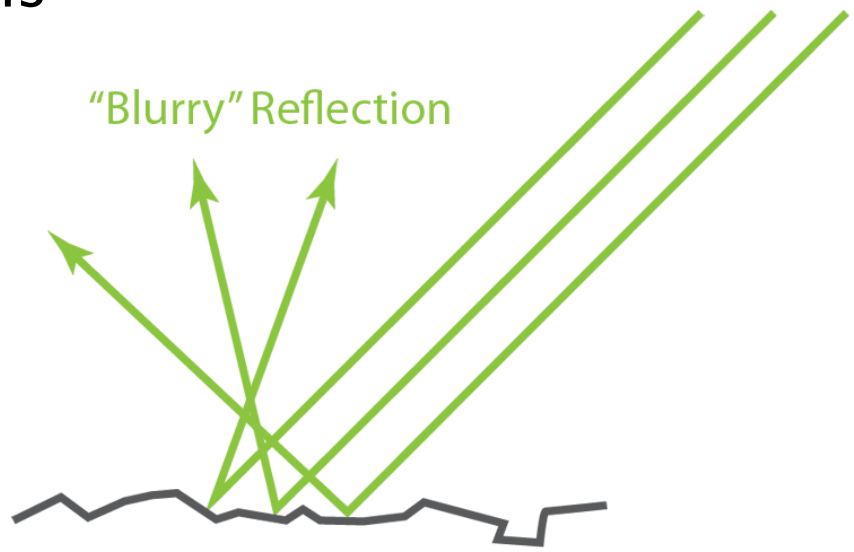
- Isn't that just an approximation?
  - Yup, but it's a pretty good one – and it's fast
  - Good enough for PBR engines like Unreal & Unity

- BRDF Examples:
  - From Epic's slides
  - More sophisticated BRDF wasn't worth computation cost



Specular BRDF – Microfacets

- Every surface must account for reflections
  - But we don't always want *perfect* reflections
- Some surfaces are rough
  - Often what we think of as "less shiny"
  - Rough surfaces actually have "blurry" reflections
- In general, everything reflects light
  - That's how you can see it
  - It's *how* the light is reflected that matters



Side Note: Everything is Shiny

- Can capture real-world diffuse & specular
  - With a camera and a light polarizer

- Example: An obviously shiny plate

Regular picture

Diffuse Light

Specular Reflection

And everything has Fresnel

Everything is Shiny – More Stuff

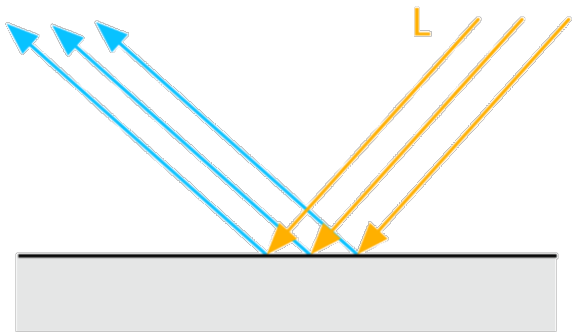
- Cardboard is shiny



- Cloth is shiny



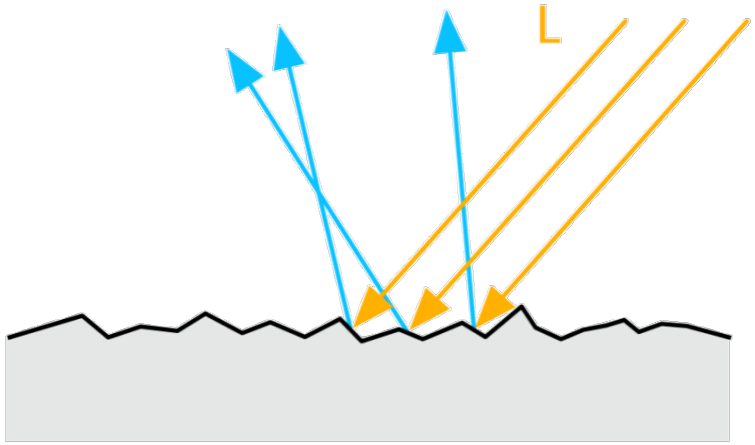
Microfacet model



- Most real surfaces aren't perfectly smooth
  - Not *optically flat*, as many simple equations assume

- Microfacet model
  - Models surfaces as collections of microscopic, smooth surfaces
  - Microfacets *are* optically flat and too small to be seen

- Microfacet equations calculate percentage of facets that reflect light towards us



Microfacet examples

- Can define roughness as a normalized gradient
  - 0: Perfectly smooth (microfacets aligned)
  - 1: Perfectly rough (microfacets unaligned)

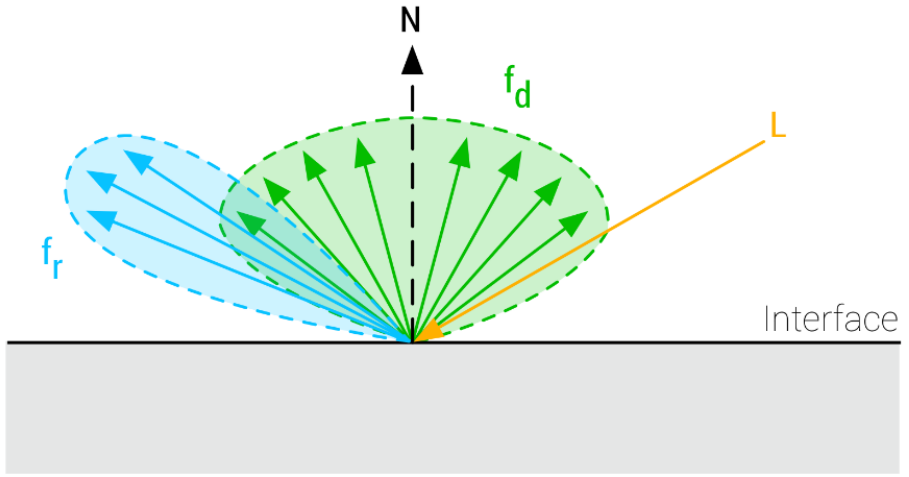
1.0

0.0

Reflectivity

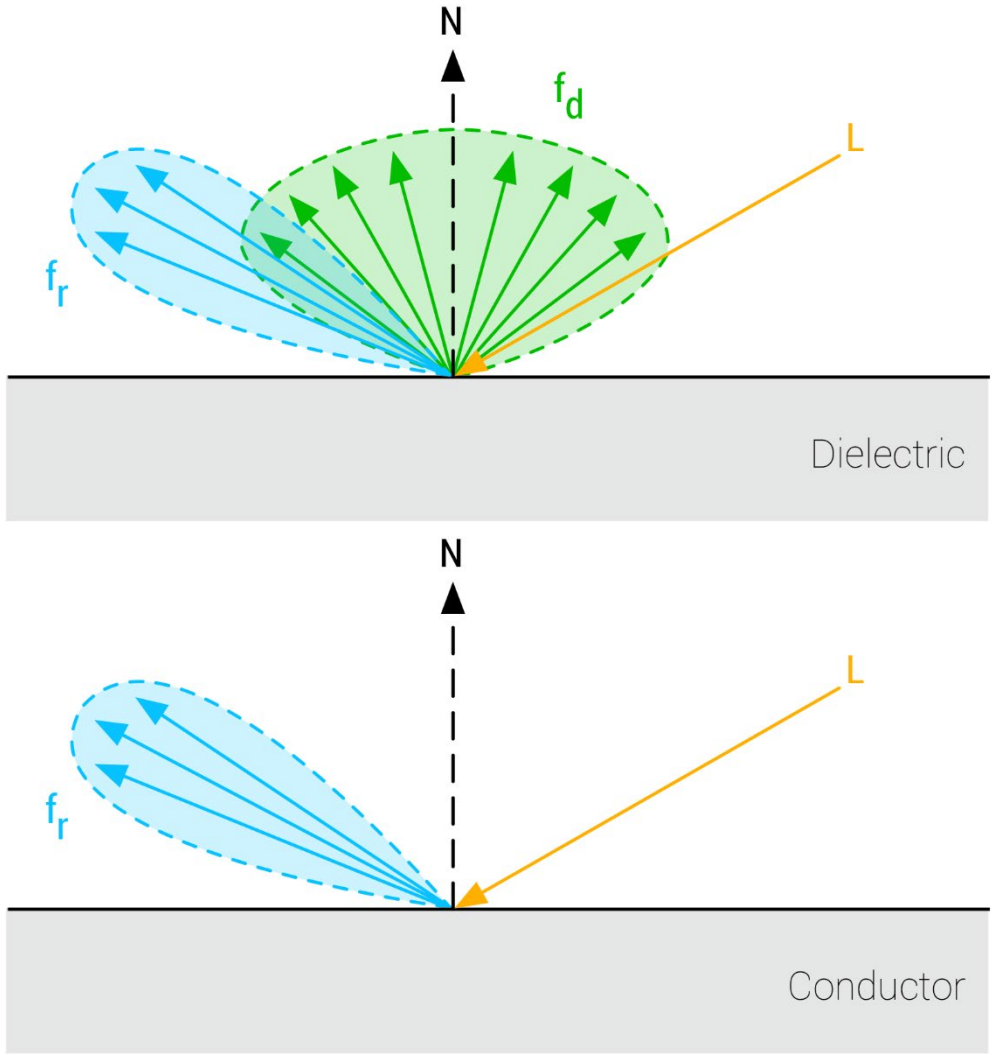
- Roughness determines how *blurry* reflections end up being
  - Very artist-driven – how smooth is the surface?

- What about surface *reflectivity?*
  - Inherent ratio of diffuse vs. specular
  - Very physics-dependent
  - Is the surface metal or non-metal?
  - Controlled through “metalness” parameter

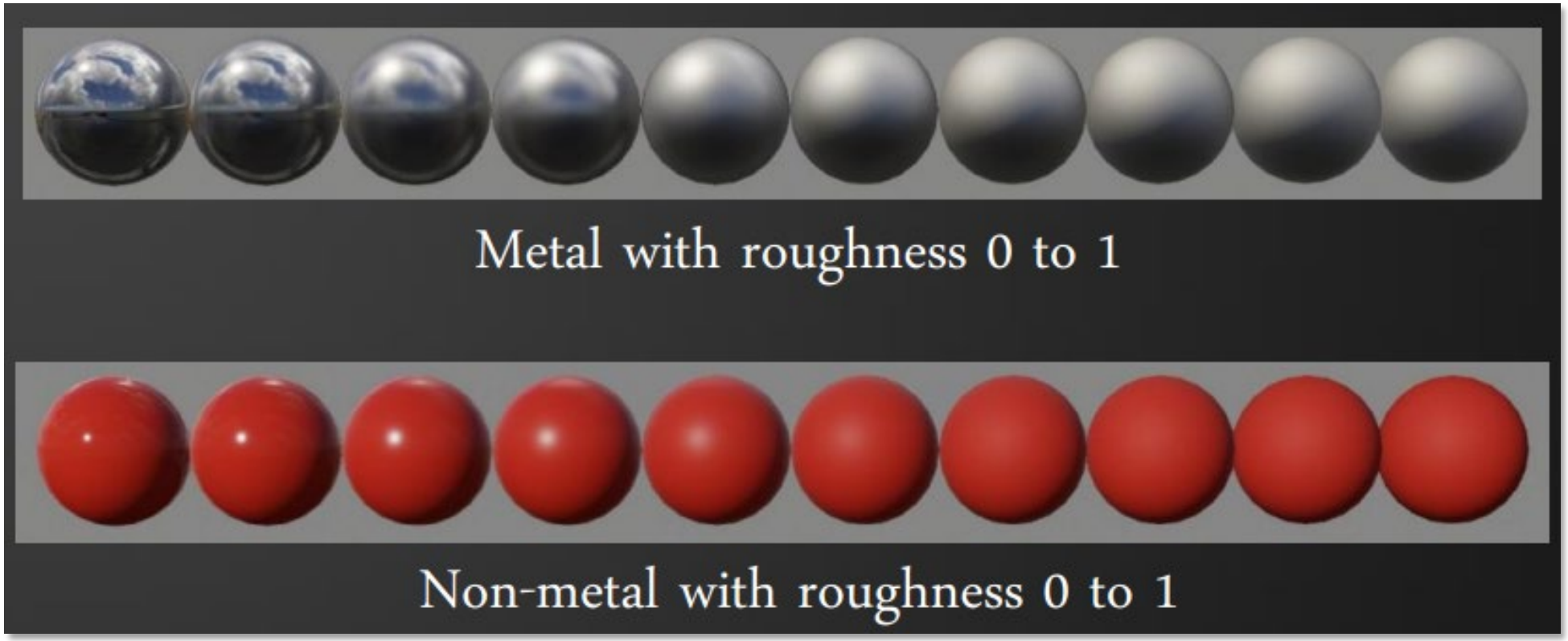


Metalness, huh?

- Real-world objects are either
- *Dielectrics*
  - Plastic, rubber, etc.
  - Both diffusion and reflection
- *Conductors*
  - Metals
  - Reflection only
  - No diffusion



Metalness example



Determining real-world material properties

- Lots of work into sampling real-world materials, BRDFs, etc.
- Quixel Megascans
- MERL 100
  - Mitsubishi Electric Research Laboratories
  - Captured 100 different material samples
  - BRDF data available freely online
- Disney's 2012 paper on Physically Based Shading

Microfacet BRDF: Cook-Torrance

- Widely used specular BRDF:

$$specular(v,l) = \frac{D(n,h,\alpha)F(v,h,f_0)G(n,v,l,\alpha)}{4(n\cdot v)(n\cdot l)}$$

- Where:
  - **L** is the vector to the light
  - **V** is the vector to the camera
  - **H** is the half-angle vector between view and light: (V + L) / 2
  - **N** is the surface normal
  - α is roughness
  - f_0 is reflectivity at normal incidence

Old school vs. PBR – BRDF charts

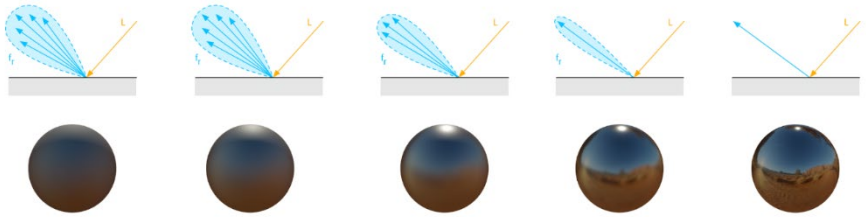
(Old school)	Direct Light	Indirect Light
Diffuse	Lambert $(N \cdot L)$	Ambient
Specular	Phong	Cubemap Reflections

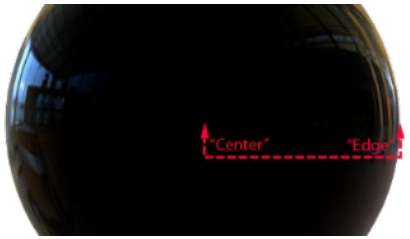
(PBR)	Direct Light	Indirect Light (not covered today)
Diffuse	Lambert $(N \cdot L)$	Image-Based Lighting (IBL) w/ irradiance map
Specular	Cook-Torrance Microfacet	IBL w/ convolved environ. map

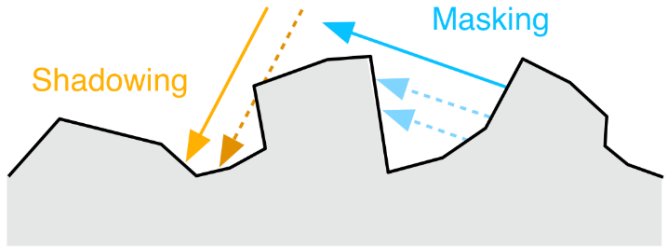
Microfacet BRDF: Cook-Torrance

$$specular(v,l) = \frac{D(n,h,\alpha)F(v,h,f_0)G(n,v,l,\alpha)}{4(n\cdot v)(n\cdot l)}$$

- $D(\,)$ is Normal Distribution
- $F(\,)$ is Fresnel
- $G(\,)$ is Geometric Shadowing





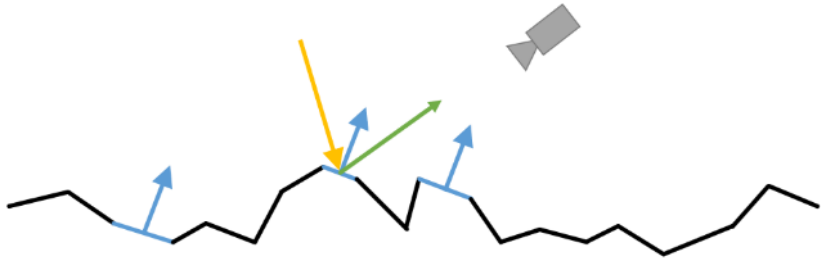


Common Cook-Torrance Functions (UE4)

- $D()$ Normal Distribution Function
  - Often Trowbridge-Reitz (GGX)
  - Better falloff than Blinn-Phong
- $F()$ Fresnel
  - Schlick approximation – faster, almost identical
- $G()$ Geometric Shadowing
  - Schlick-GGX – Schlick's version, remapped to match GGX D( )
- Many other options can be found here

Normal Distribution Function: D()

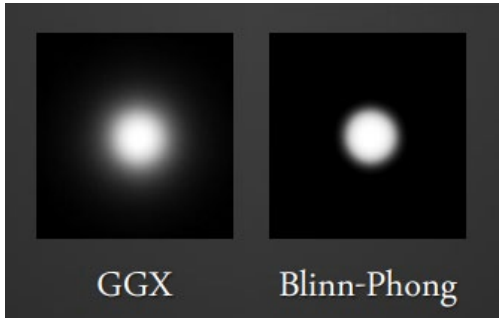
- Percent of microfacets that *could* reflect light towards camera



- Trowbridge-Reitz (GGX)

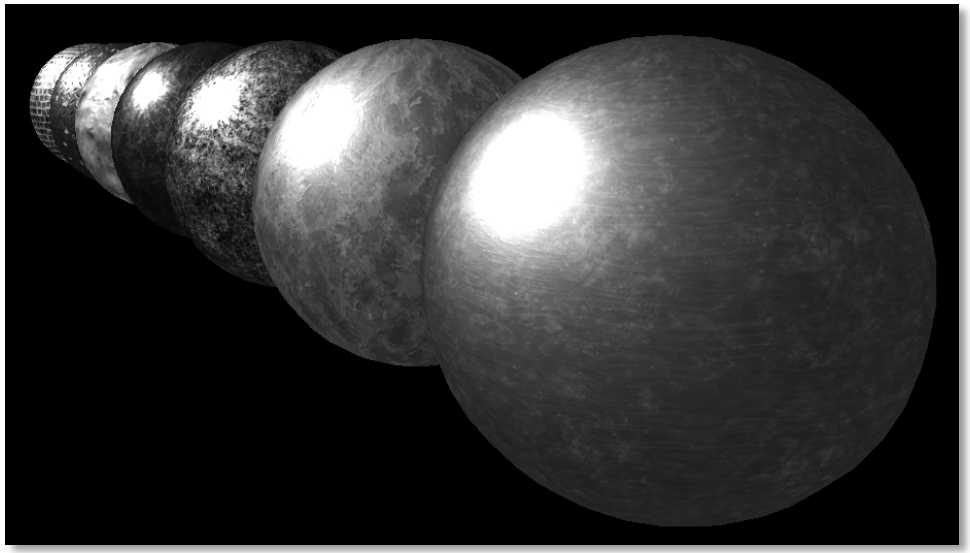
$$D_{GGX}(\mathbf{m}) = \frac{\alpha^2}{\pi((\mathbf{n}\cdot\mathbf{m})^2(\alpha^2-1)+1)^2}$$

  - n is the normal
  - m is the half vector in our case
  - α is roughness (often remapped as roughness^2, as per Unreal & Disney)

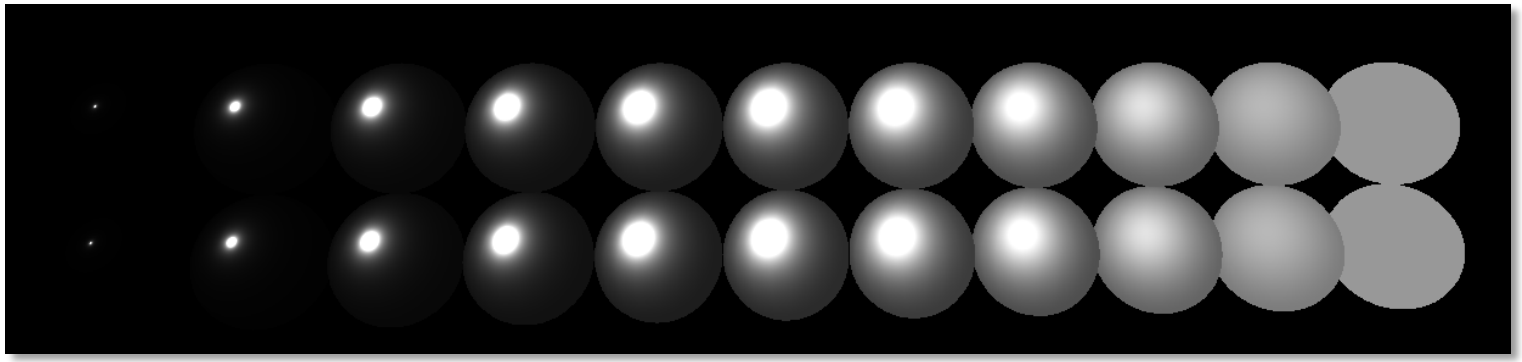


More accurate falloff than Phong variations

Normal Distribution Function – Results



D(h) with normal and roughness maps



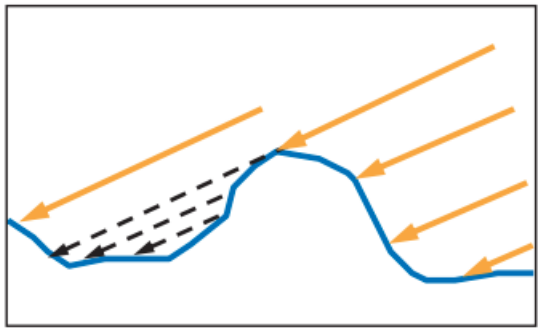
D(h) with roughness ranging zero to one

Rougher → reflection is more spread out (blurry)

Normal Distribution Function – Code

```
float D_GGX(float3 n, float3 h, float roughness)
{
  // Pre-calculations
  float NdotH = saturate(dot(n, h));
  float NdotH2 = NdotH * NdotH;
  float a = roughness * roughness; // Remap roughness (Unreal & Disney)
  float a2 = max(a * a, MIN_ROUGHNESS); // MIN_ROUGHNESS is 0.0000001

  float denomToSquare = NdotH2 * (a2 - 1) + 1;

  // Final value
  return a2 / (PI * denomToSquare * denomToSquare);
}
```

Geometric Shadowing – G()



- Shadow-masking function
  - % of microfacets not blocked by other microfacets
- Smith implementation
  - Breaks G() into two parts:
  - Light (shadowing) & View (masking)
- G_1 is Schlick approximated for GGX
- *k* depends on roughness
  - Roughness first remapped
  - (see Unreal paper linked later)

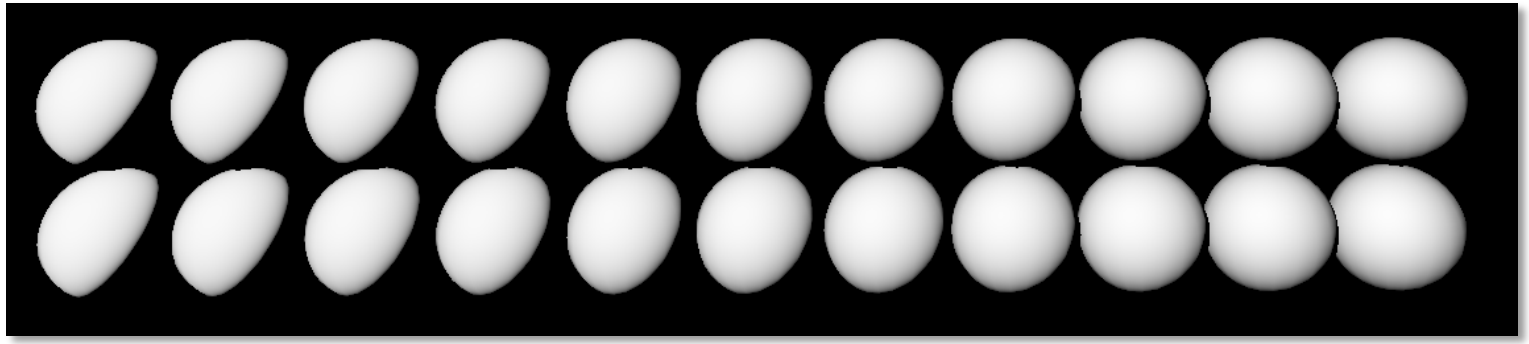
$$G(\mathbf{l}, \mathbf{v}, \mathbf{h}) = G_1(\mathbf{l})G_1(\mathbf{v})$$

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

$$k = \frac{\alpha}{2}$$

Geometric Shadowing – Results

G() with roughness maps



G() with roughness ranging zero to one

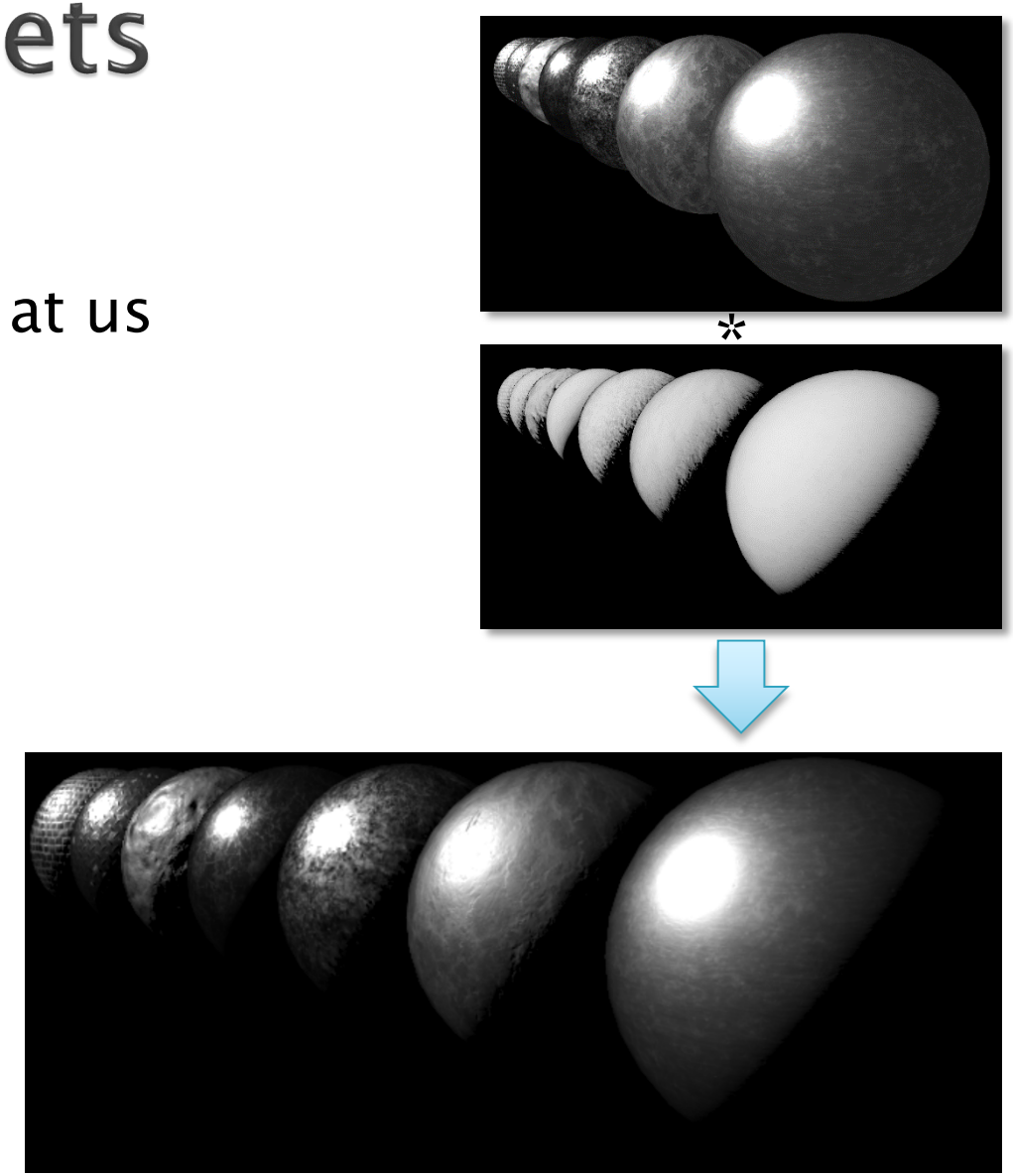
Geometric Shadowing – Code

```
// Roughness remapped to (r+1)/2 before squaring, then k remapped to a/2
float G_SchlickGGX(float3 n, float3 v, float roughness)
{
  // End result of remapping:
  float k = pow(roughness + 1, 2) / 8.0f;
  float NdotV = saturate(dot(n, v));

  // Final value
  return NdotV / (NdotV * (1 - k) + k);
}

// Note: This is called twice and combined: G(n,v,r) * G(n,l,r)
```

D() * G(): Active microfacets

- Summary:
  - D() is % of facets that *could* reflect light at us
  - G() is % of facets not shadowed

- D() * G()
  - *Concentration of active microfacets*
  - % of facets that reflect light at us
  - At the current pixel



Fresnel: F()

- Calculates amount of light reflected, based on…
  - Viewing angle vs. light
  - Physical material

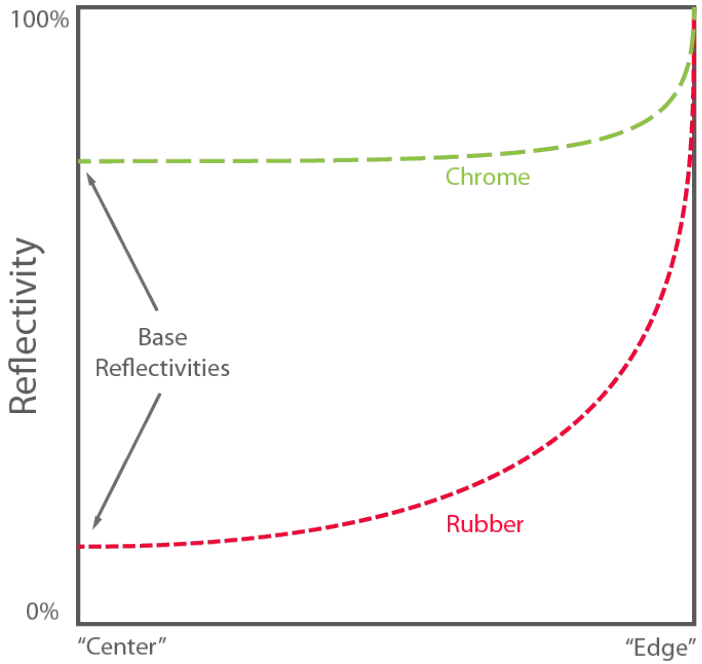
- Schlick approximation
  - Faster to compute but almost identical curve

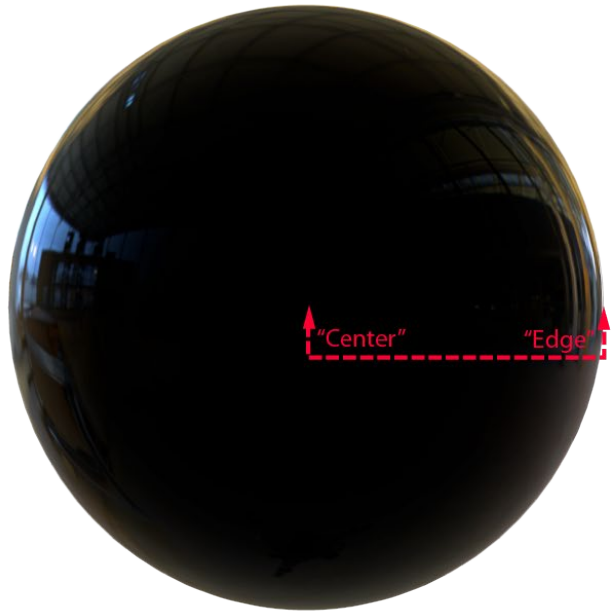
$$F_{Schlick}(\mathbf{v}, \mathbf{h}) = F_0 + (1 - F_0)(1 - (\mathbf{v} \cdot \mathbf{h}))^5$$

  - v is view vector (to camera)
  - h is half vector: (view + toLight) / 2
  - F_0 is reflectance at normal incidence (0°) for the material

Simple Fresnel example

- Everything is reflective, if only at the edge

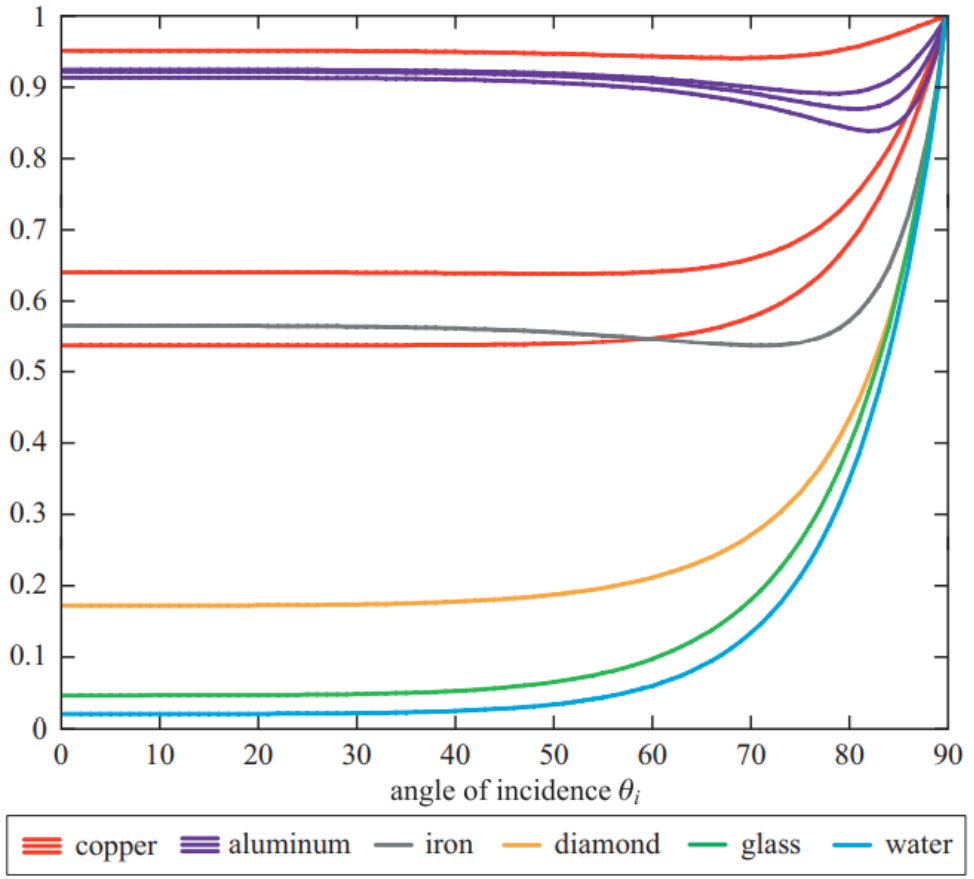




Fresnel example in pictures

Fresnel – Real-world measurements

- Metals
  - High reflectance overall
  - Tint their reflections
  - Wavelengths reflect differently!
- Dielectrics have low reflectance at normal incidence
- All materials become highly reflective at glancing angles



Fresnel and F_0

- F_0 = reflectance at normal incidence
  - Assumes light is always hitting head-on (0 degrees)

- Dielectrics (non-metals):
  - Glass & plastic are approx. 0.04
  - Liquids range from 0.02 – 0.04
  - Gemstones range from 0.05 – 0.17

- Conductors (metals):
  - Have larger ranges that vary per color channel
  - Often called “specular color” in PBR discussion

F_0 charts

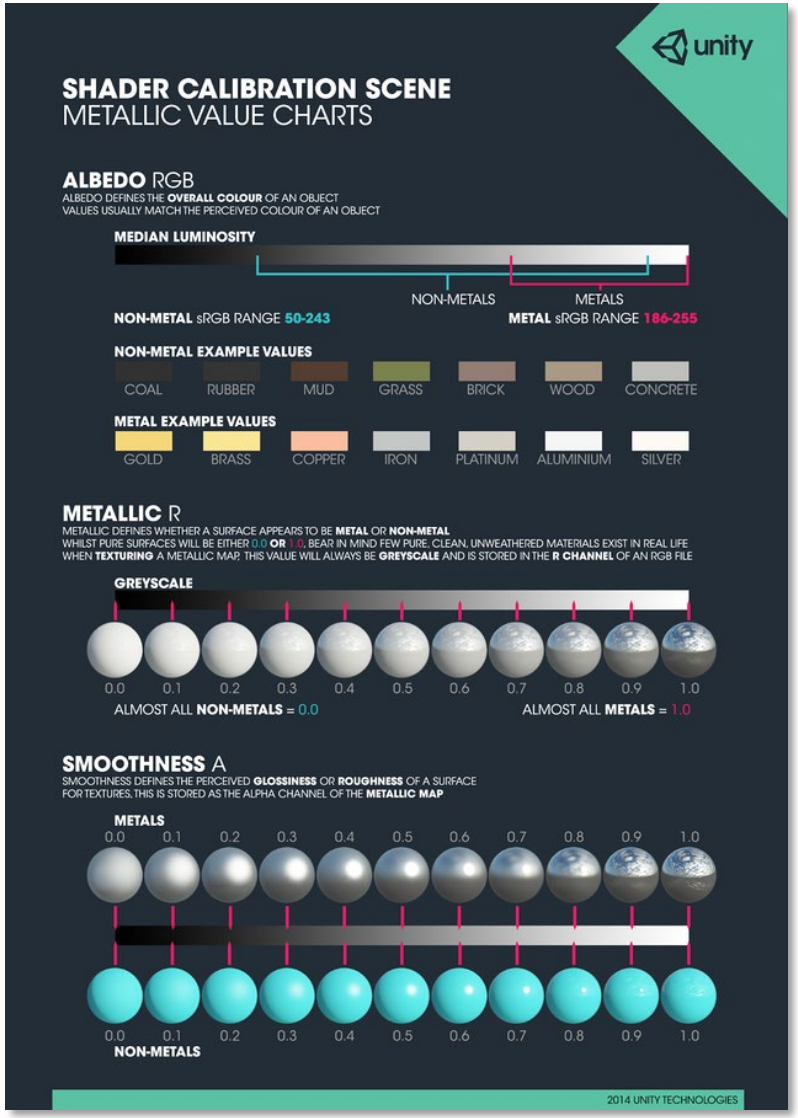
- Companies often make color charts for artists

	R	G	B
Silver	0.971519	0.959915	0.915324
Aluminium	0.913183	0.921494	0.924524
Gold	1	0.765557	0.336057
Copper	0.955008	0.637427	0.538163
Chromium	0.549585	0.556114	0.554256
Nickel	0.659777	0.608679	0.525649
Titanium	0.541931	0.496791	0.449419
Cobalt	0.662124	0.654864	0.633732
Platinum	0.672411	0.637331	0.585456

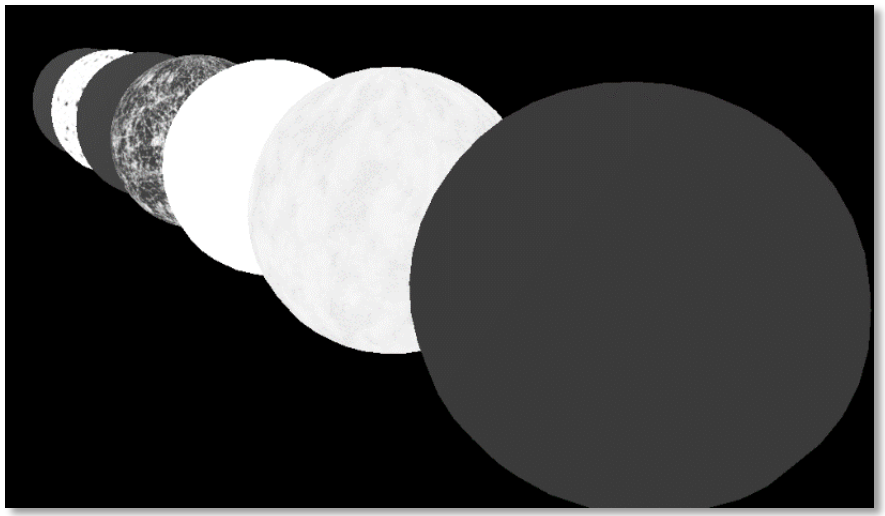
Quartz	0.045593921
ice	0.017908907
Water	0.020373188
Alcohol	0.01995505
Glass	0.04

Milk	0.022181983
Ruby	0.077271957
Crystal	0.111111111
Diamond	0.171968833
Skin	0.028

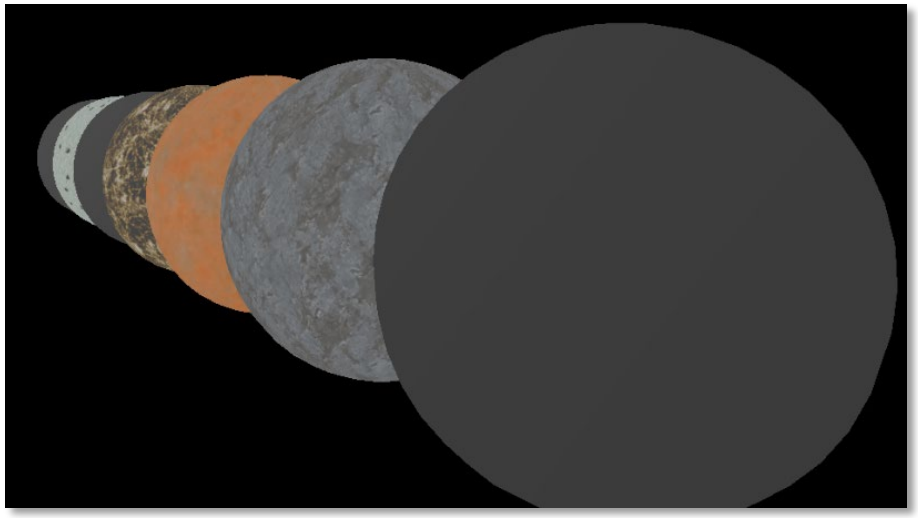
From here



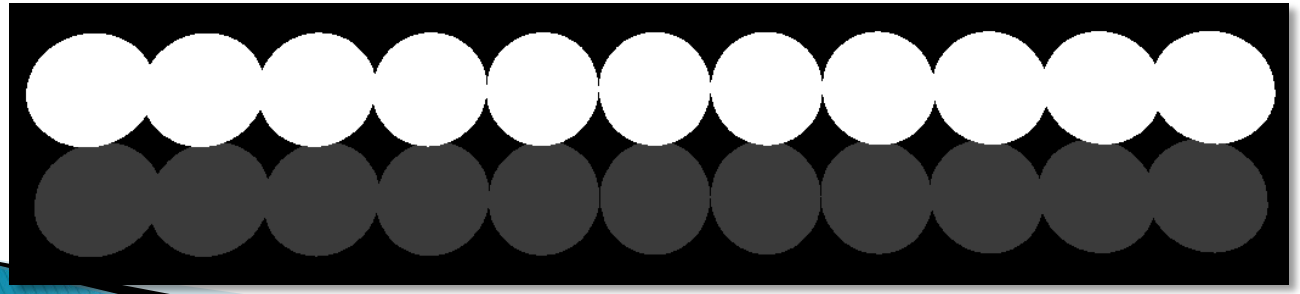
Fresnel – Results



Just metalness map
Notice metals are more reflective



Metalness & spec color
Metals have separate R/G/B reflectivity

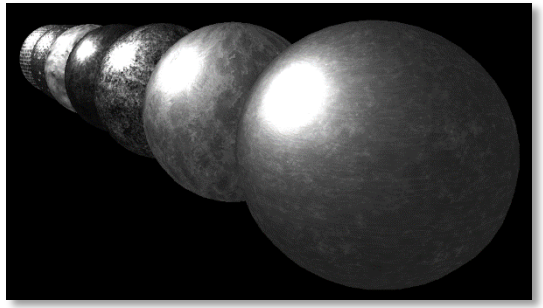


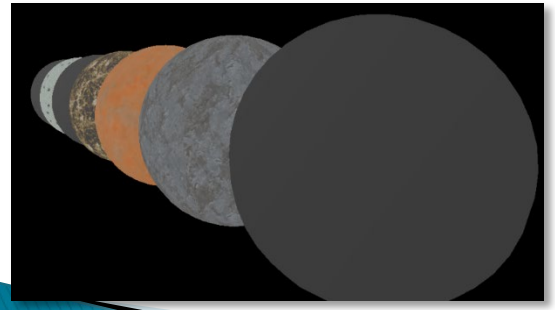
- "Silver" $F_0 = (1,1,1)$
- "Plastic" $F_0 = (0.04, 0.04, 0.04)$

Fresnel – Code

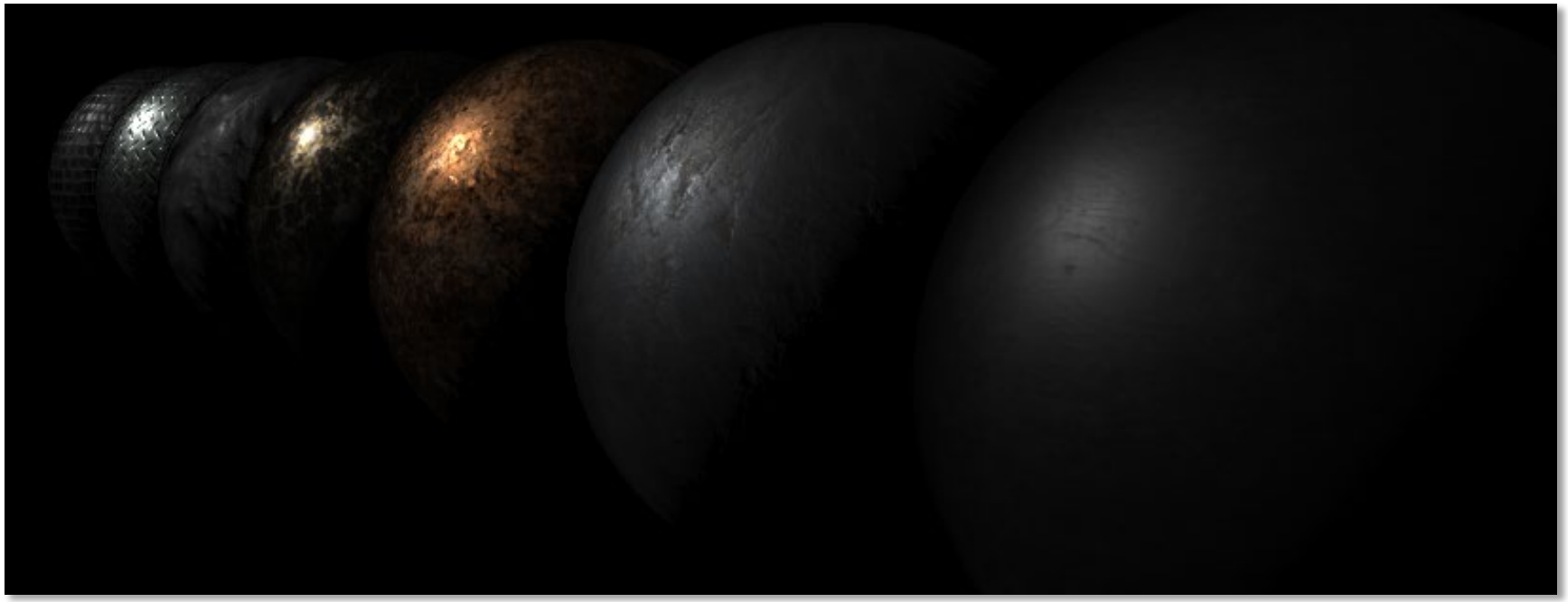
```
// f0 ranges from 0.04 for non-metals to
// a specific specular color for metals
float3 F_Schlick(float3 v, float3 h, float3 f0)
{
  // Pre-calculations
  float VdotH = saturate(dot(v, h));

  // Final value – Schlick's approximation
  return f0 + (1 - f0) * pow(1 - VdotH, 5);
}
```

Microfacet BRDF: Final specular term





$$specular(v,l) = \frac{D(n,h,\alpha)F(v,h,f_0)G(n,v,l,\alpha)}{4(n\cdot v)(n\cdot l)}$$



Microfacet – Overall code

```
// Grab various functions
float  D = D_GGX(n, h, roughness);
float3 F = F_Schlick(v, h, specColor);
float  G =
   G_SchlickGGX(n, v, roughness) *
   G_SchlickGGX(n, l, roughness);

// Final formula
// Note: NdotV or NdotL may go to zero!
// Some implementations use max(NdotV, NdotL)
return (D * F * G) / (4 * max(dot(n,v), dot(n,l)));
```

Now what?

- Combine diffuse and specular terms
- So just…
  - diffuse = LambertBRDF()
  - spec = MicrofacetBRDF()
  - final = diffuse * surfaceColor + spec
- Nope! That's too much light!
  - If light is reflecting, it's not diffusing
  - We need to *conserve energy*

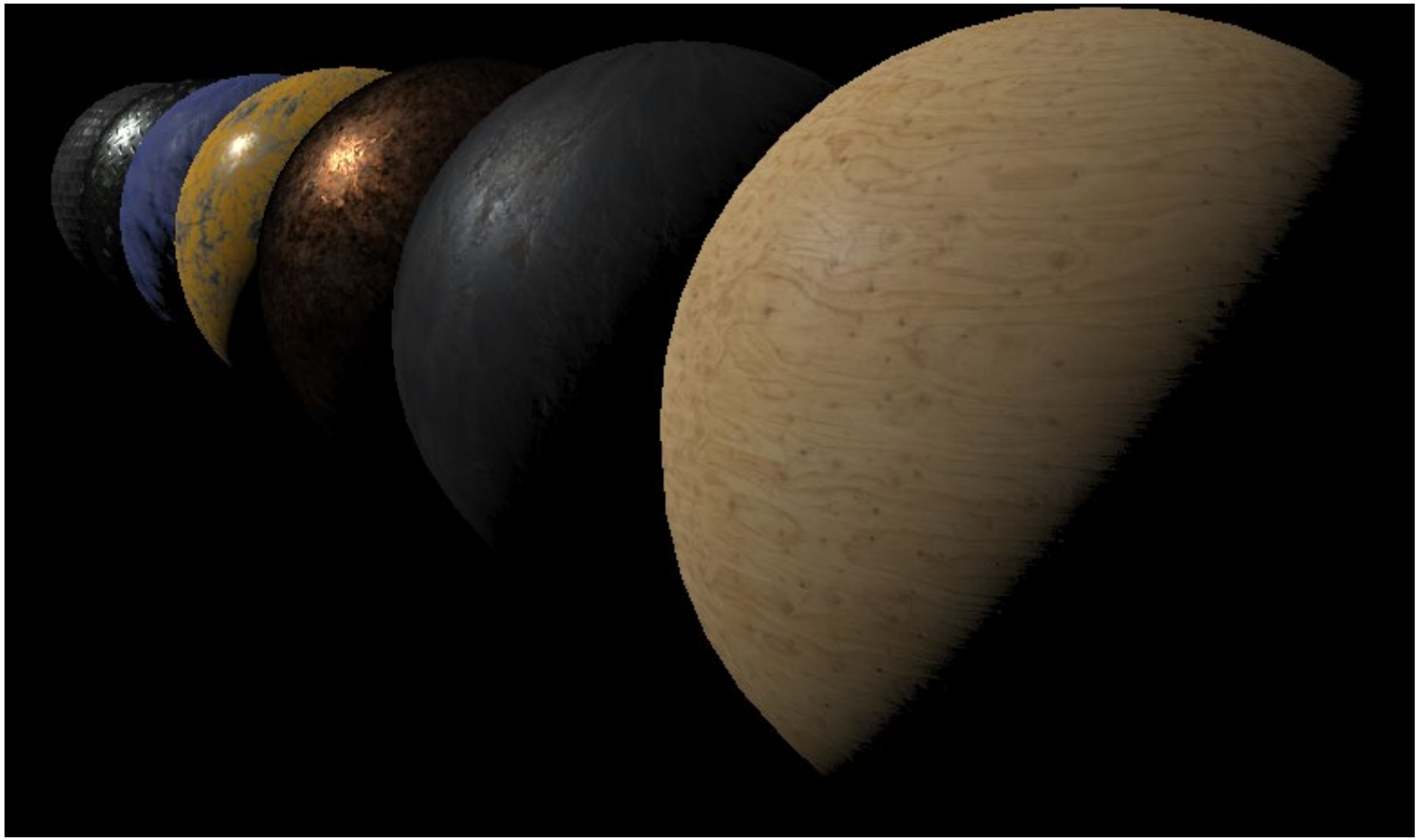
Energy conservation

- Combining diffuse and specular needs to conserve energy
  - Surface cannot reflect more light than it receives!
  - More reflection = less diffusion
  - Article on Energy Conservation in Games

Conserving energy – Code

```
// Calculates diffuse color based on energy conservation
// Note: metals should have an albedo of approx.
//  (0,0,0), so take metalness into account here!
float3 DiffuseEnergyConserve(
   float3 diffuse,
   float3 F,  // Should be result of just F()
   float metalness)
{
  return diffuse * (1 - F) * (1 - metalness);
}
```

PBR: Diffuse + Specular (balanced)



What about indirect light?

- A big part of PBR is indirect light
  - Reflecting the “environment”
  - Or an approximation (image-based lighting)

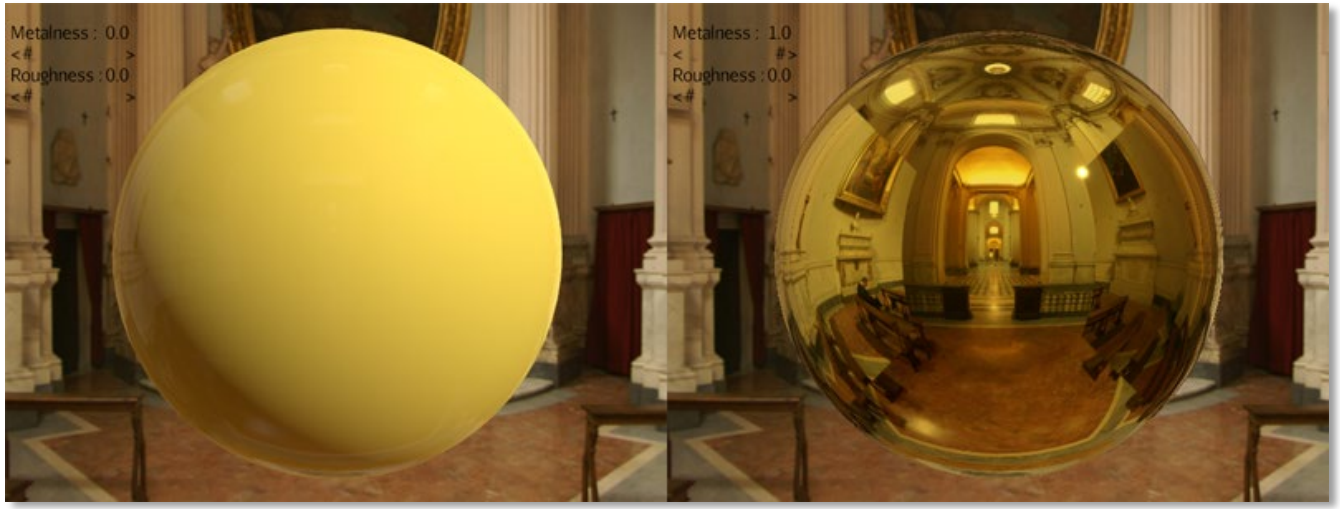


Image based lighting: Example

- Environment (cube map) is used to light the object in addition to actual lights
  - Looks good (and correct) in any environment

What are we reflecting?

- Direct Light
  - Light from individual light sources
  - (This is what Cook-Torrance helped us with)

- Indirect Light
  - Light bouncing around the scene
  - A simple “global illumination” approximation

- How do we determine the indirect light?
- And how do we handle these blurry reflections?
- Topics for the next course!

Other useful resources

- SIGGRAPH 2013 course on PBS
- PBR Theory and Practice from Marmoset
  - Focused on artists
- Slides and Paper about Unreal's PBS system
- Articles from DICE about adopting and feeding a PBS system
- PBR in Filament