Physically Based Shading

Diana Algma

29.03.17

I will talk about

- What is Physically Based Shading?
  - Why is it good?
- Light physics
  - Diffusion and reflection (Why does snow appear white?)
  - Fresnel effect
  - Microsurface
  - Conductors vs insulators
  - Energy conservation
- Simple rendering example
- Examples of PBS

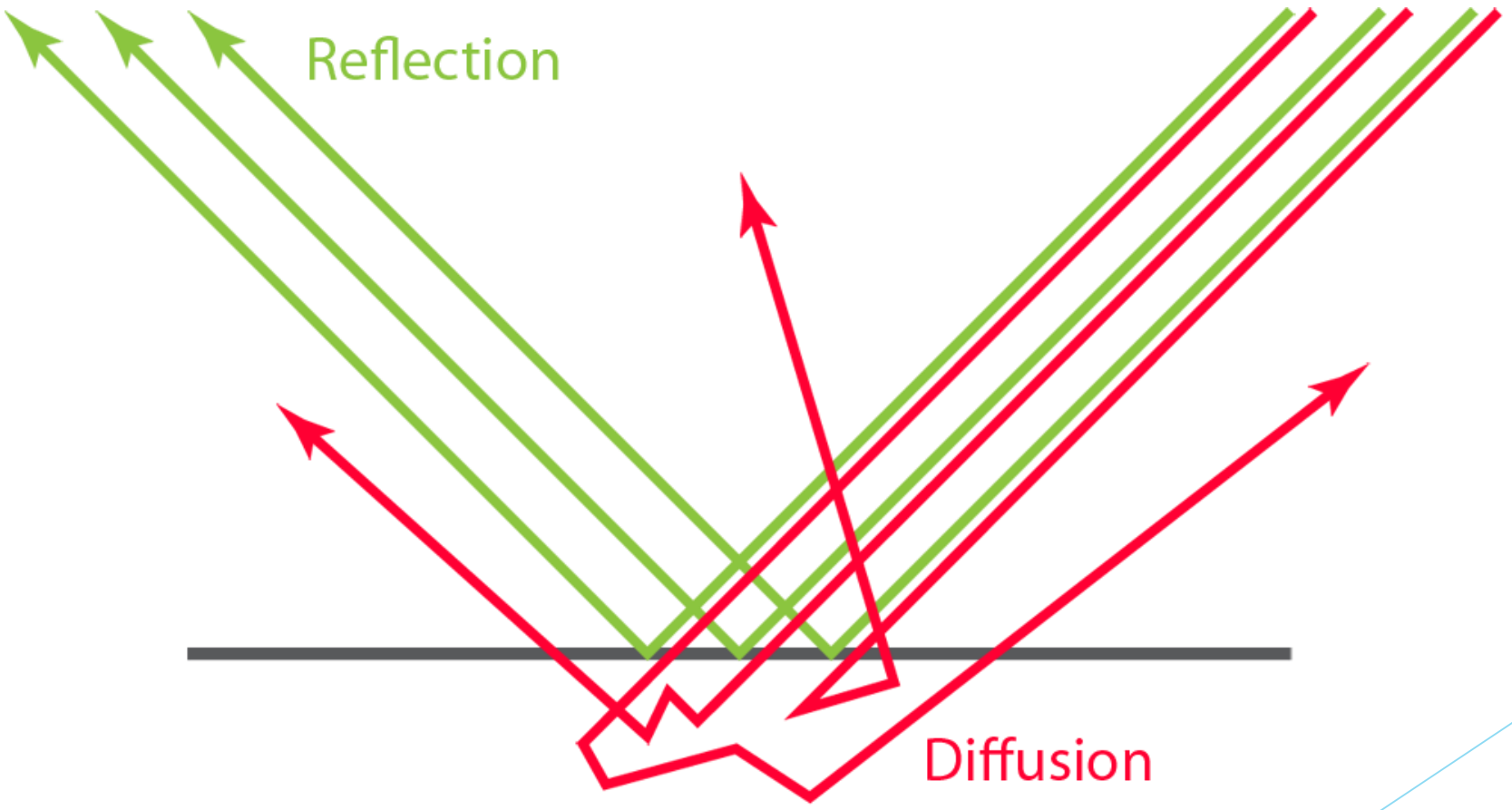
What is PBS, why is it good?

- Shading that models how light behaves in reality
- Follows principles of physics:
  - Energy conservation
  - Fresnel reflections
  - Much more

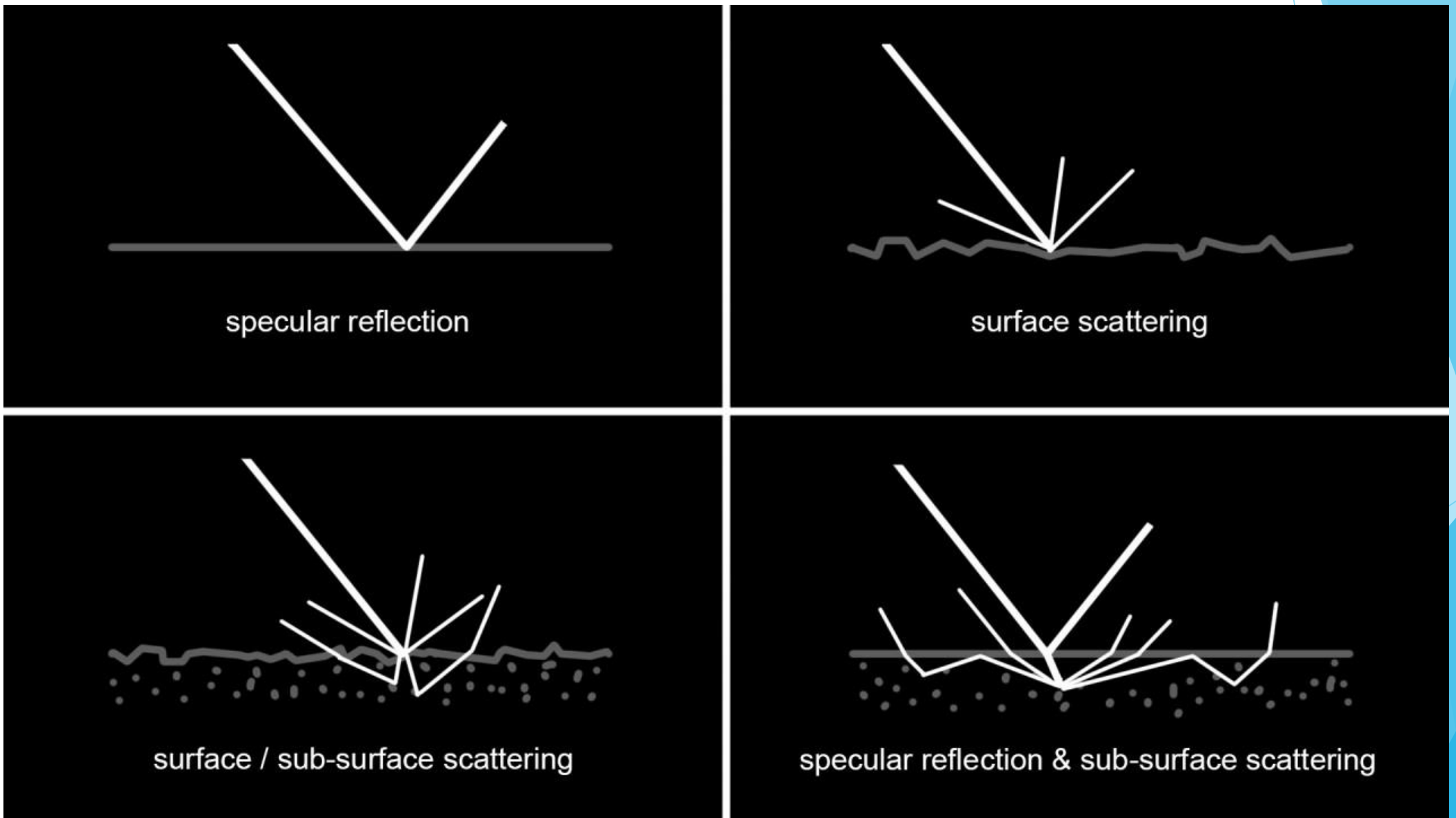
- One shader for many different lighting conditions

Light

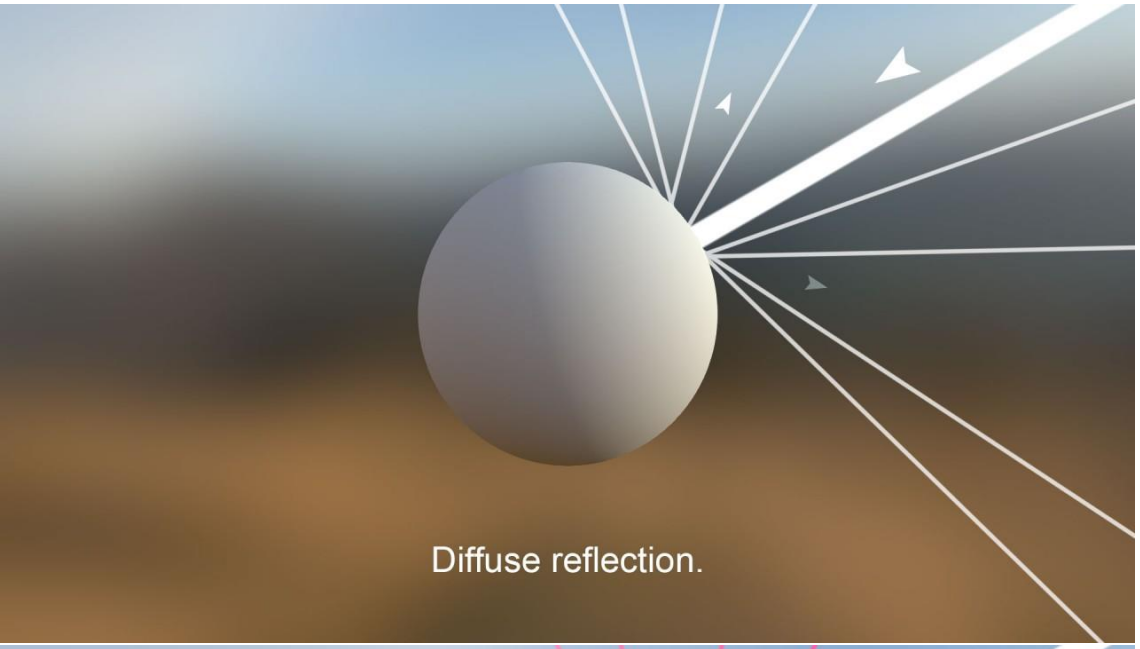
Reflection vs Diffusion



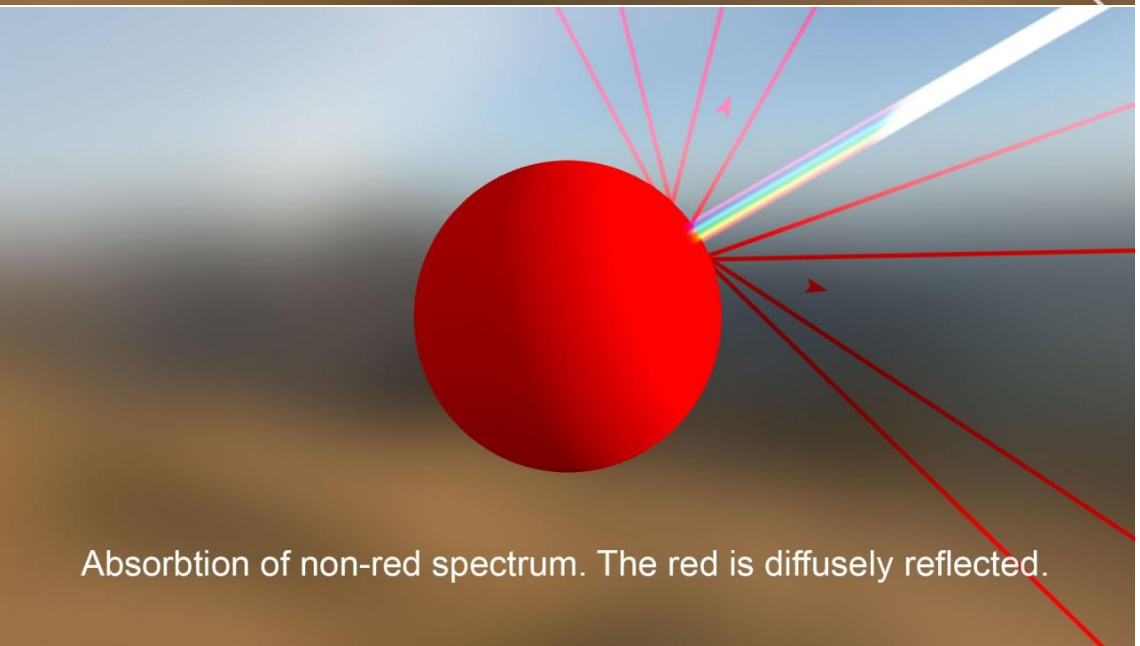
Reflection vs Diffusion



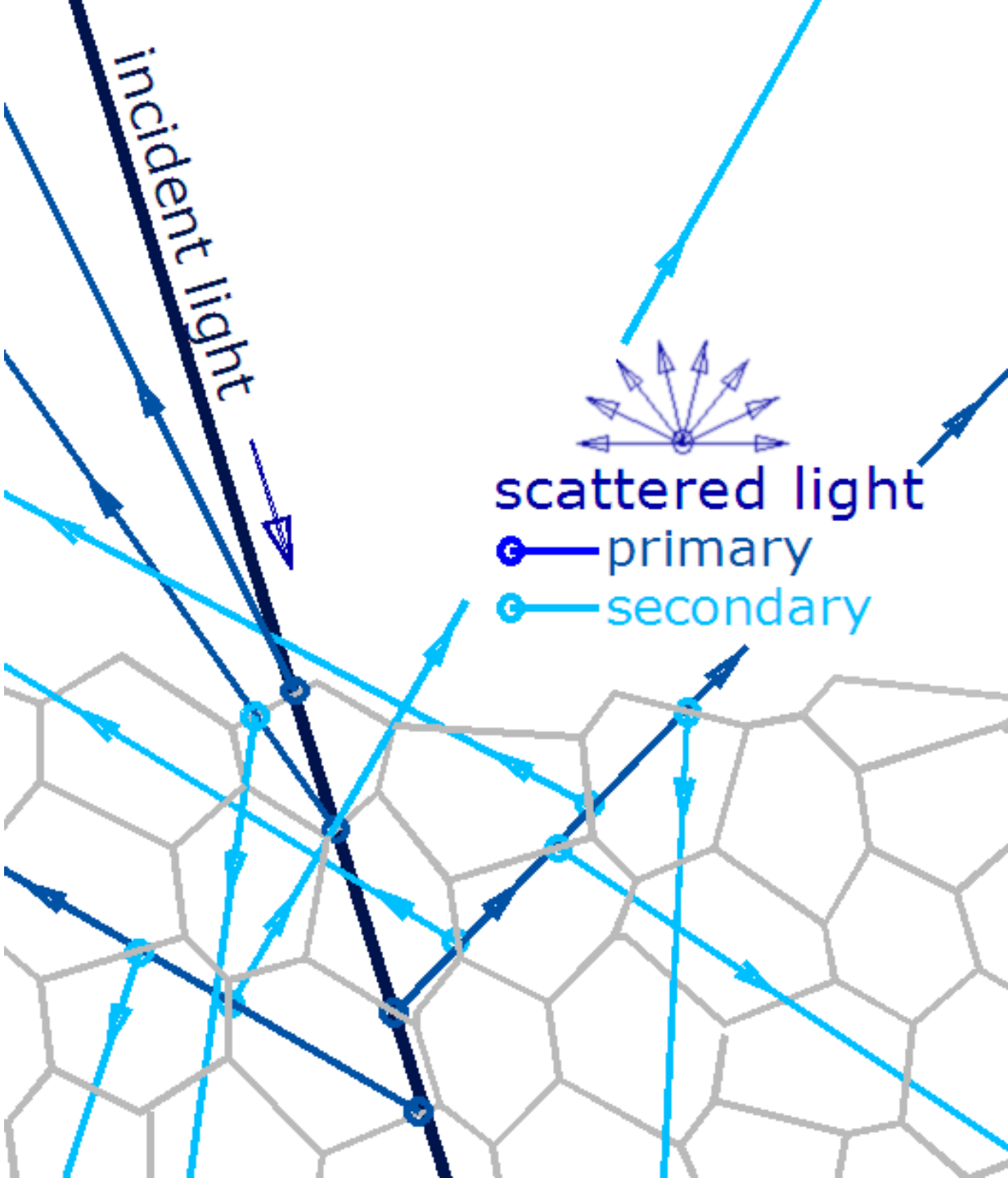
Diffusion



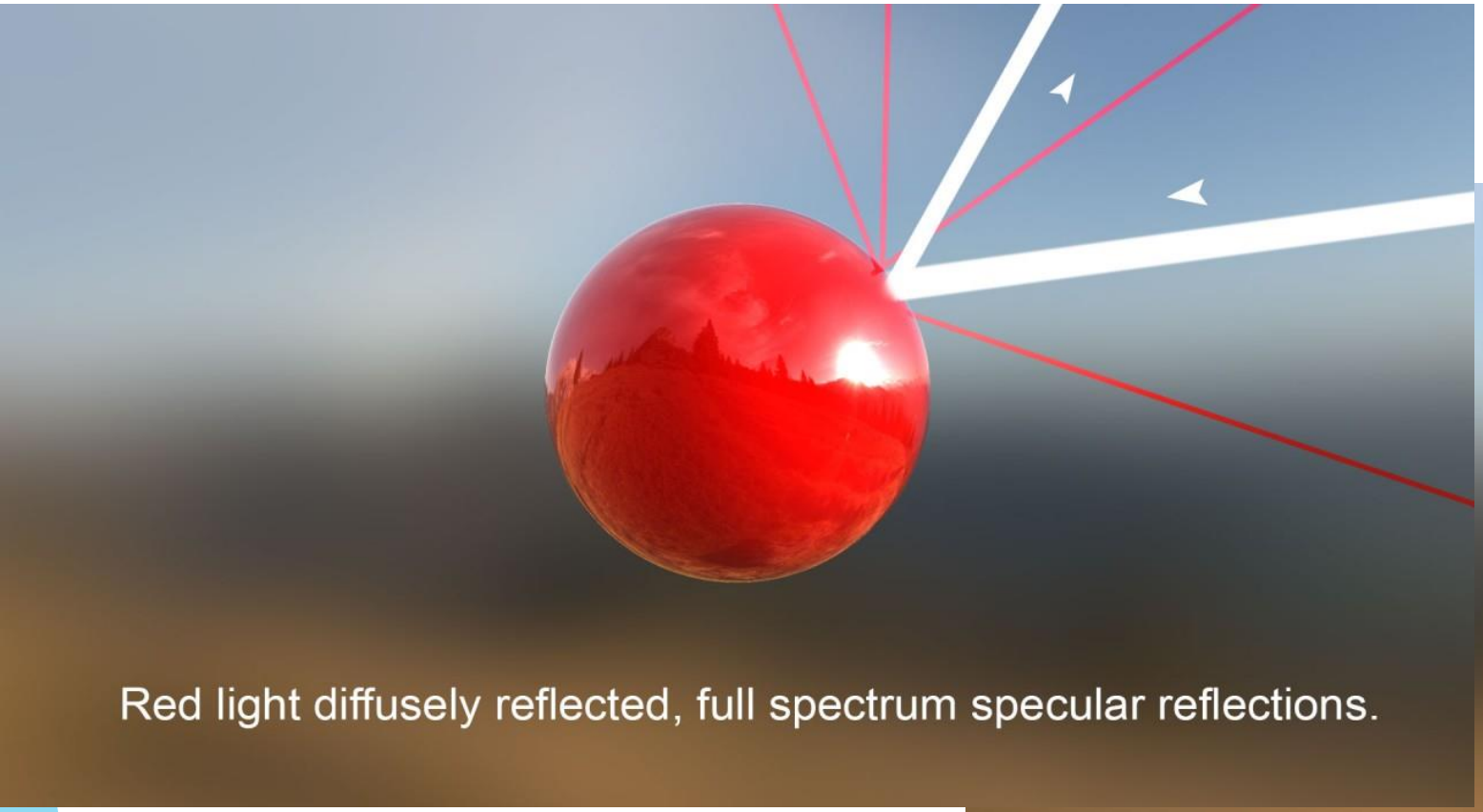
Diffuse reflection.

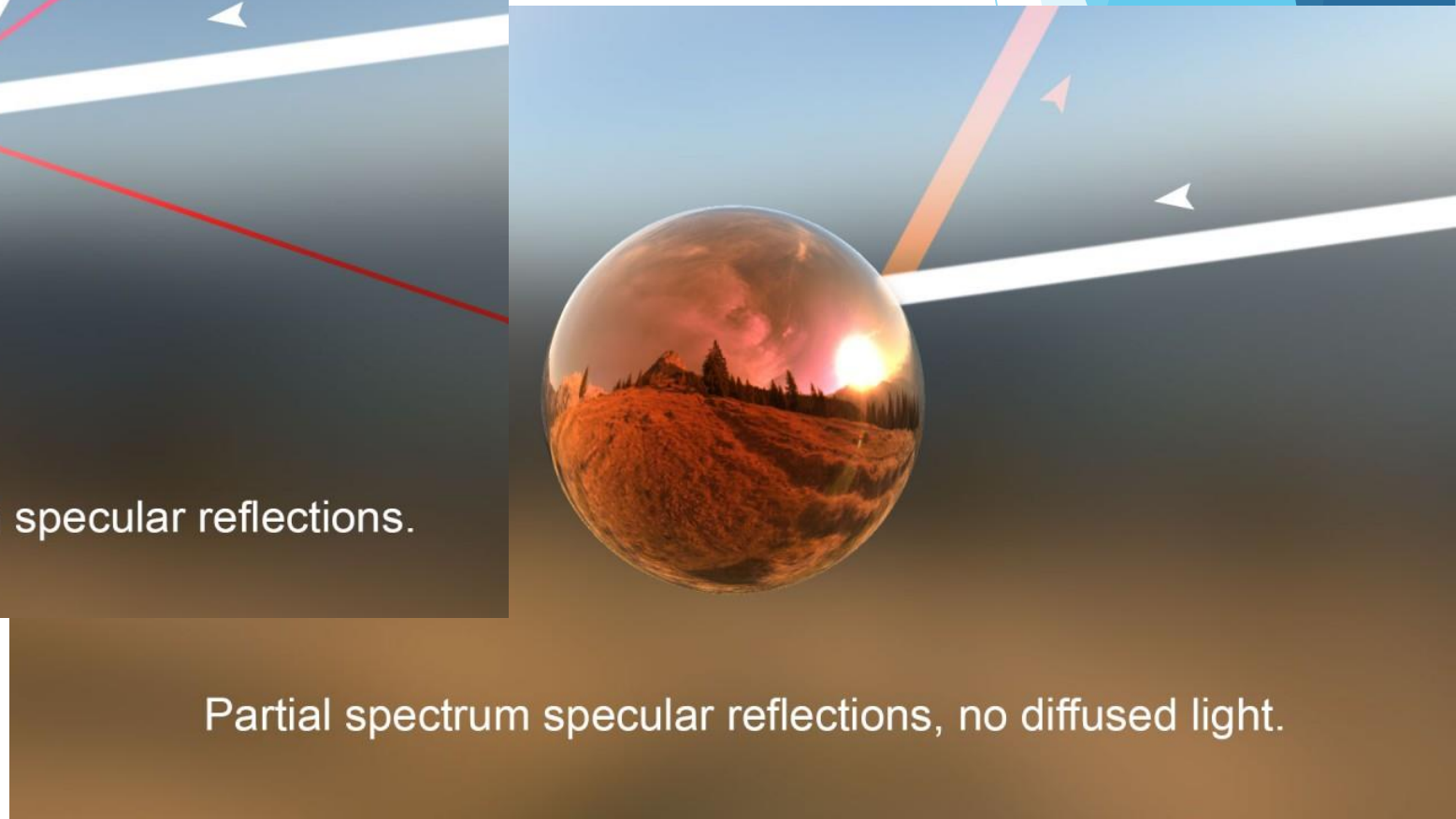


Absorbtion of non-red spectrum. The red is diffusely reflected.

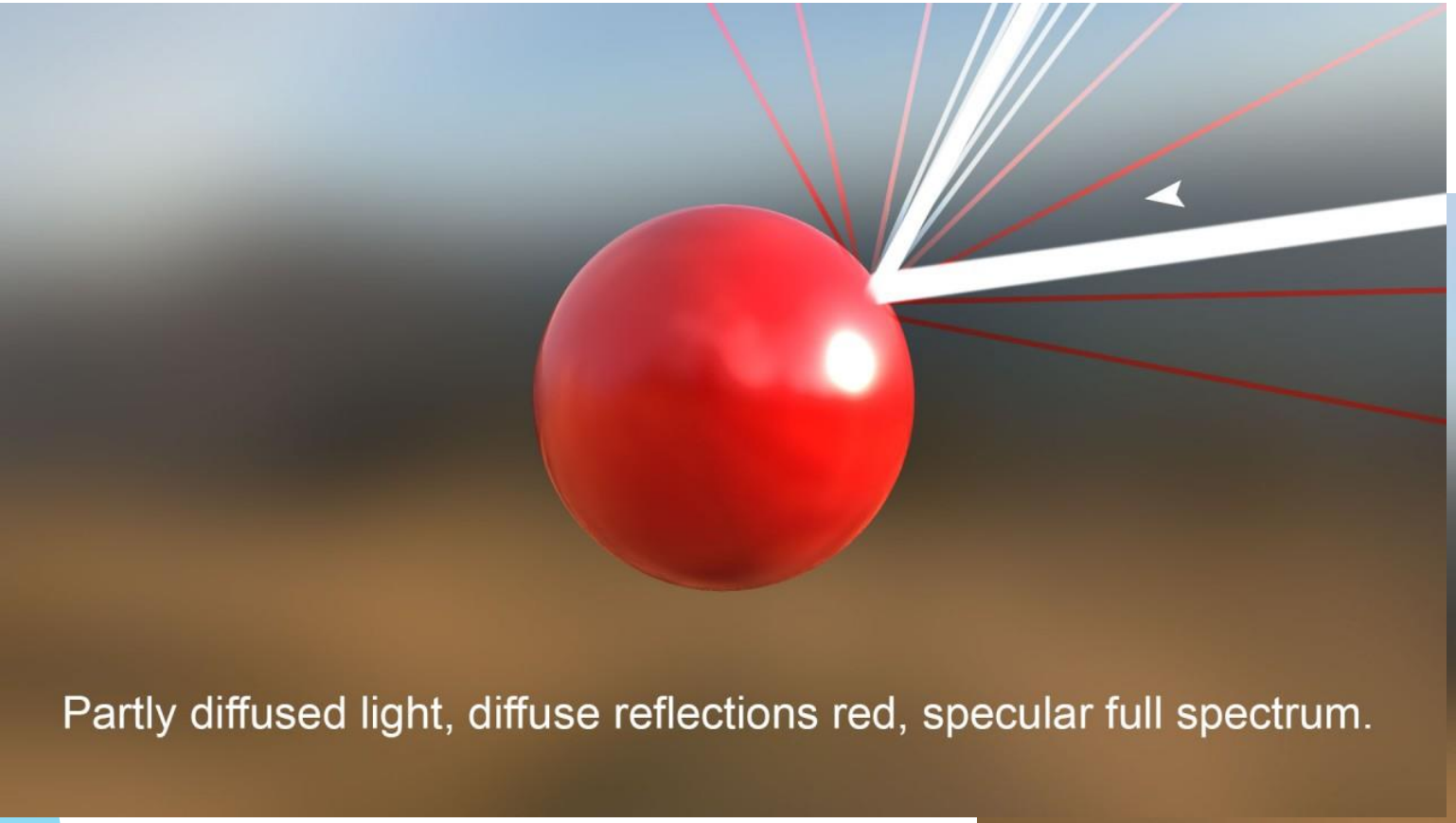


Reflection

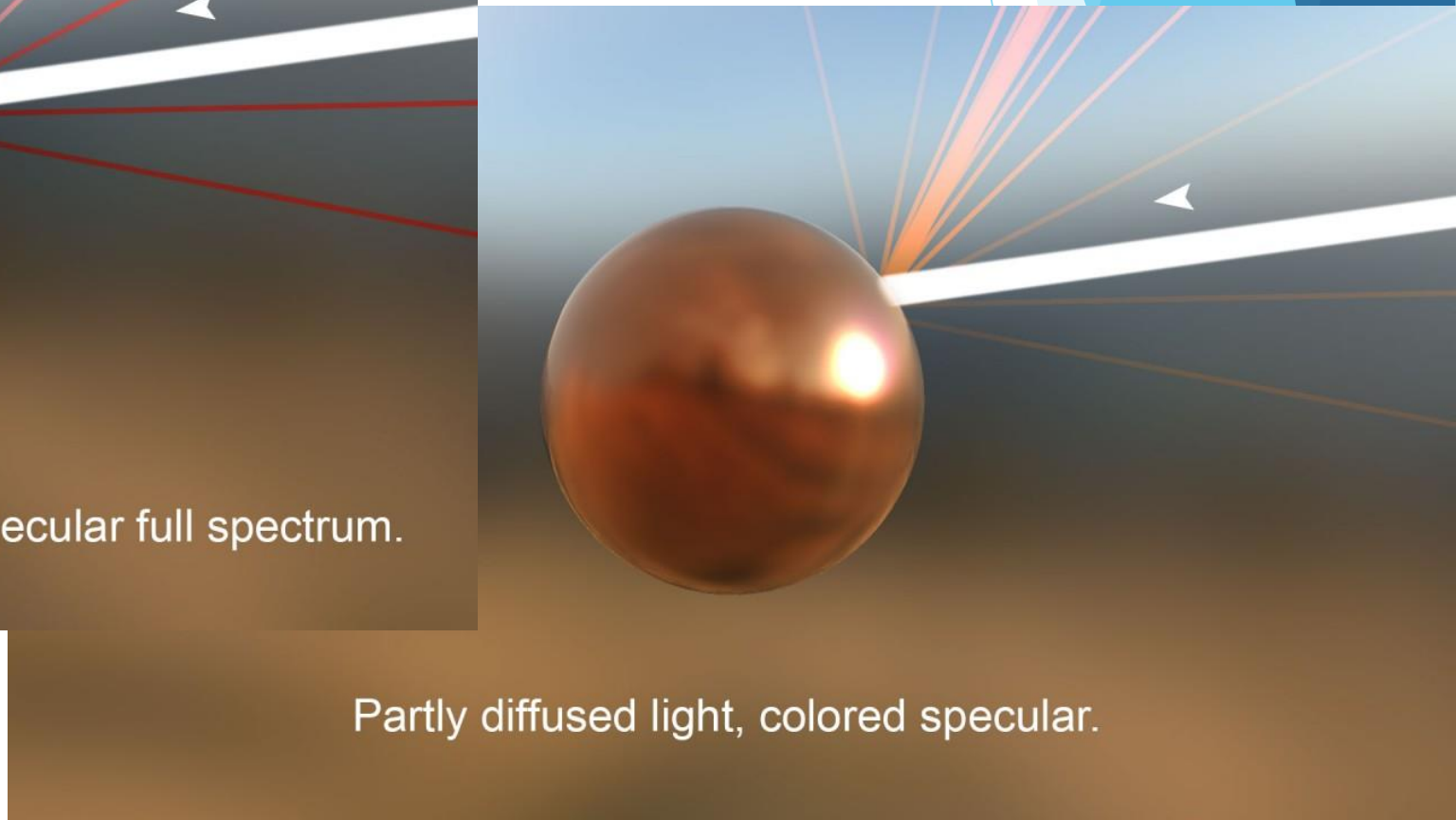

Red light diffusely reflected, full spectrum specular reflections.


Partial spectrum specular reflections, no diffused light.

Reflection & Diffusion



Partly diffused light, diffuse reflections red, specular full spectrum.

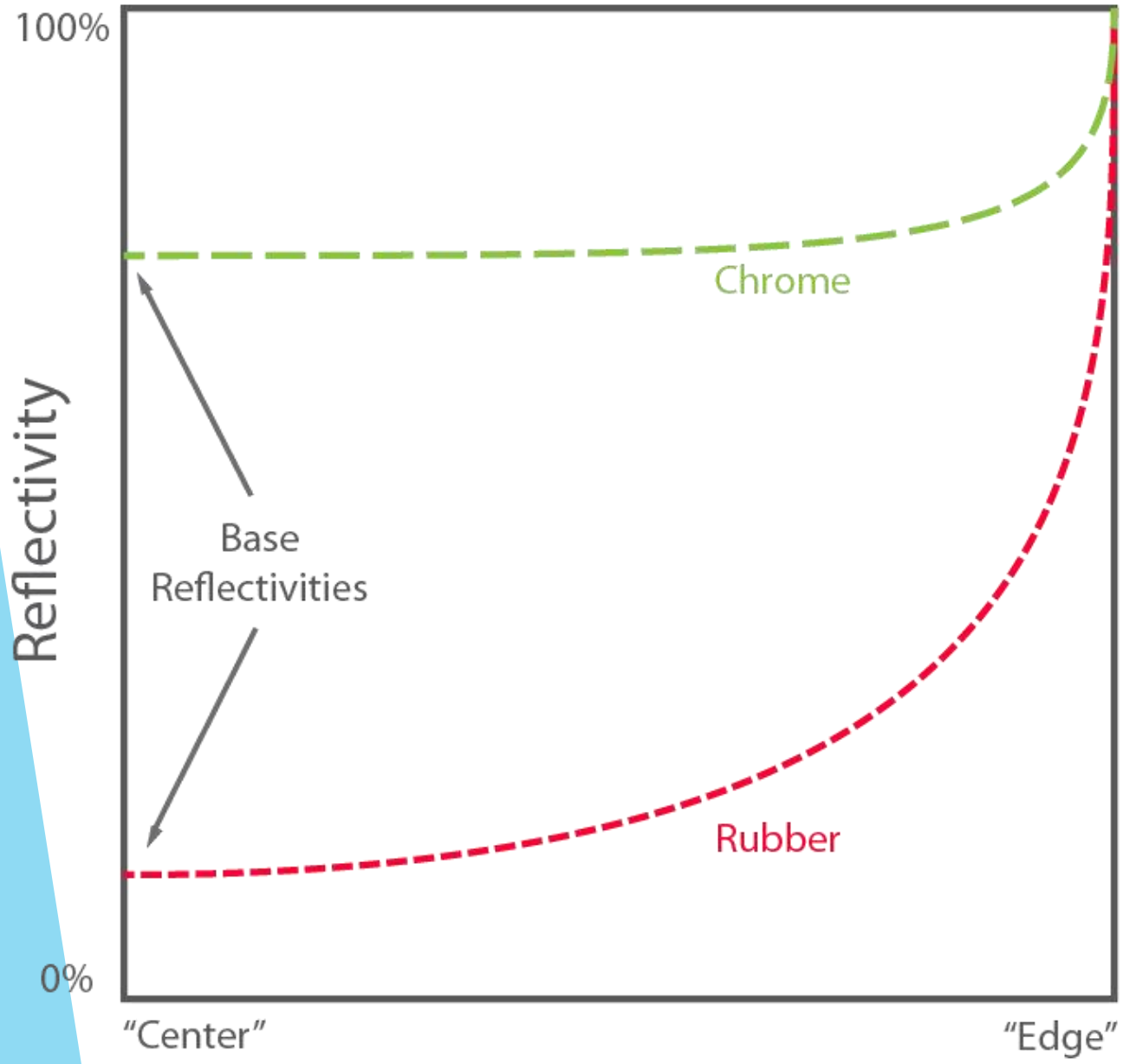


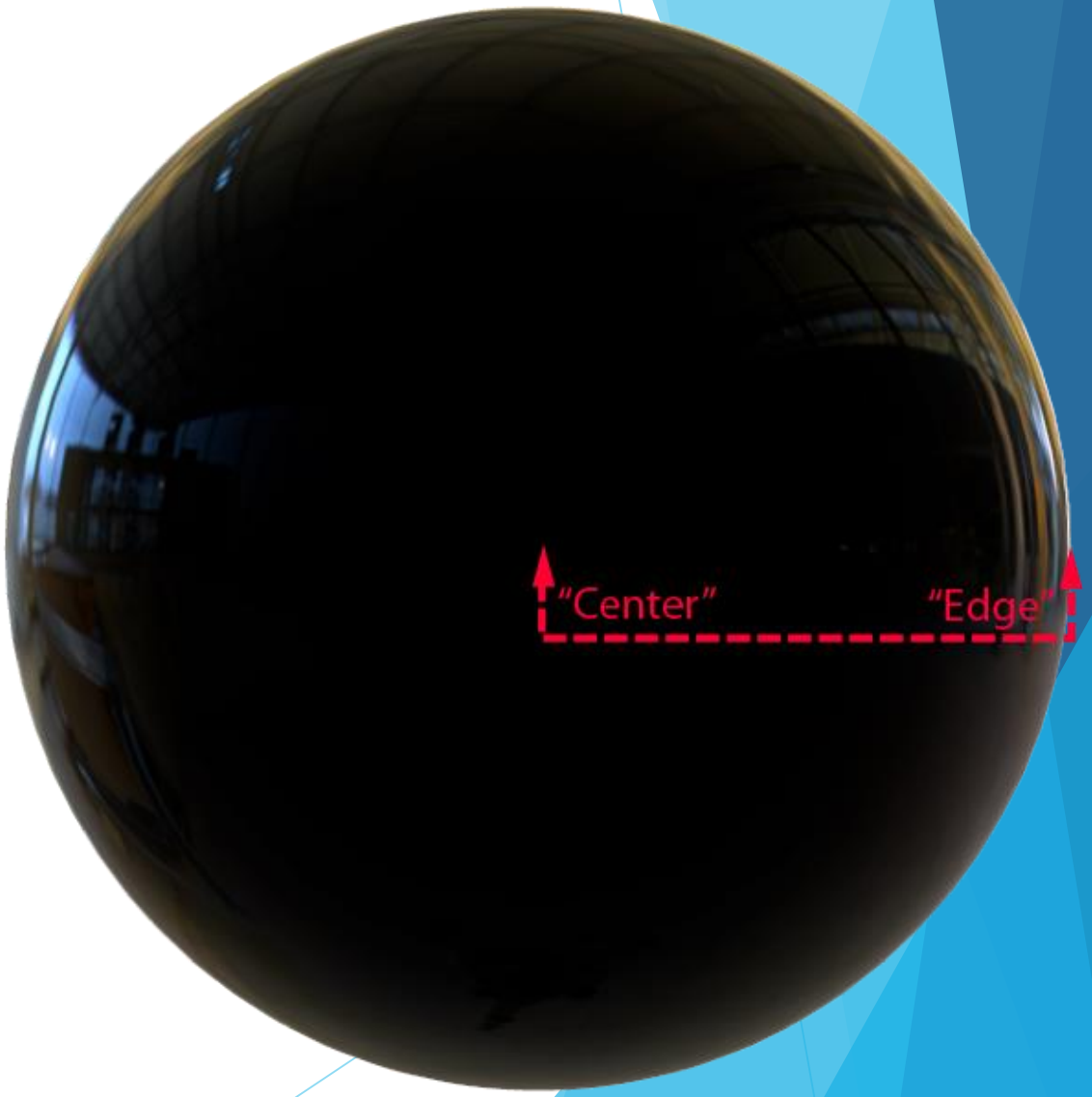
Partly diffused light, colored specular.

Fresnel Effect

- Any material can act as a perfect mirror if it is smooth and viewed at the right angle

Fresnel Effect

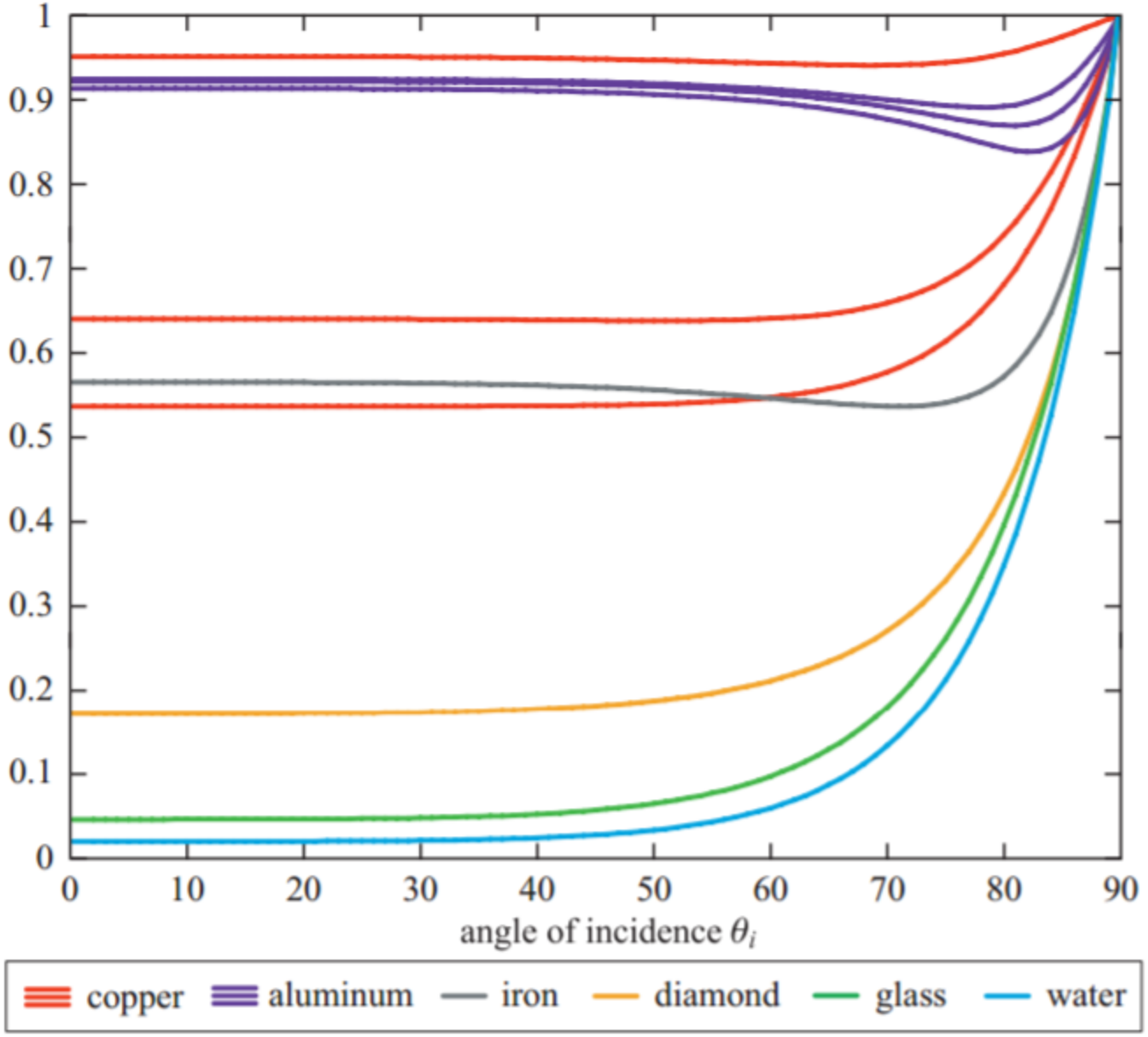




Fresnel Effect

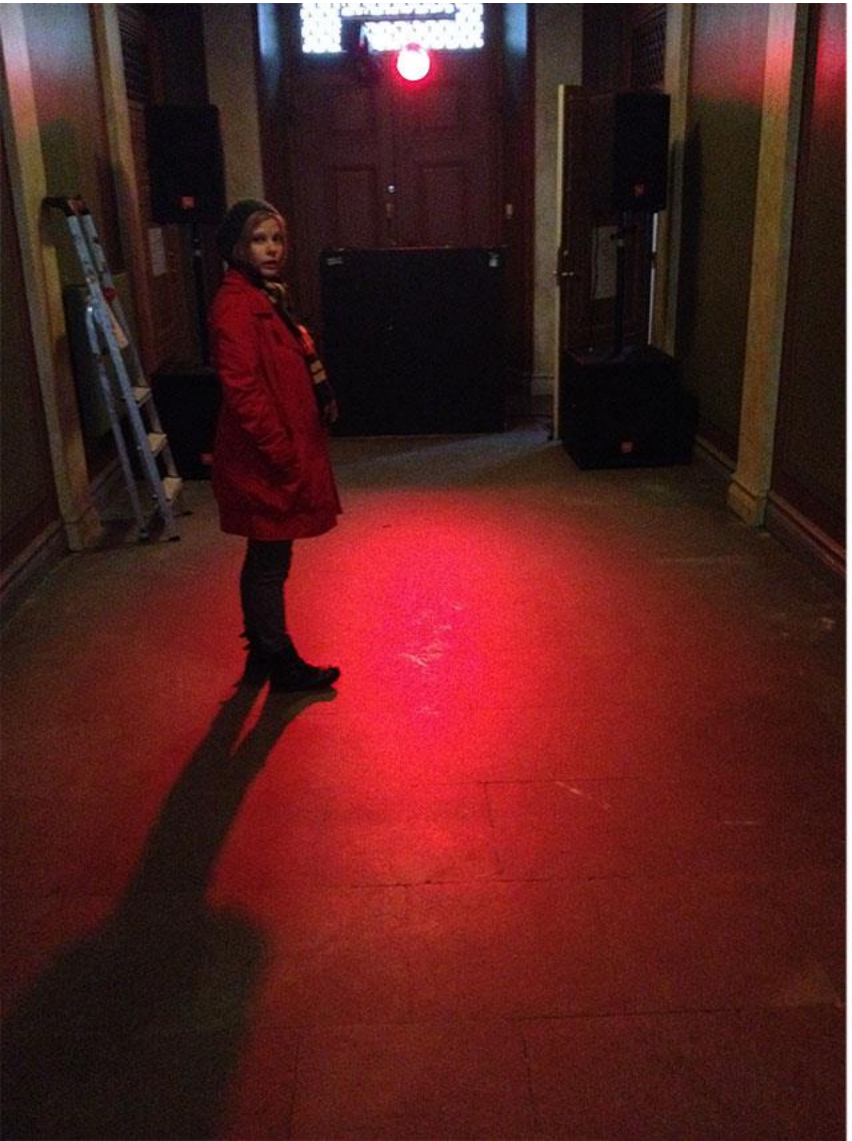
- Any material can act as a perfect mirror if it is smooth and viewed at the right angle
- The curve or gradient between the angles does not vary much from material to material

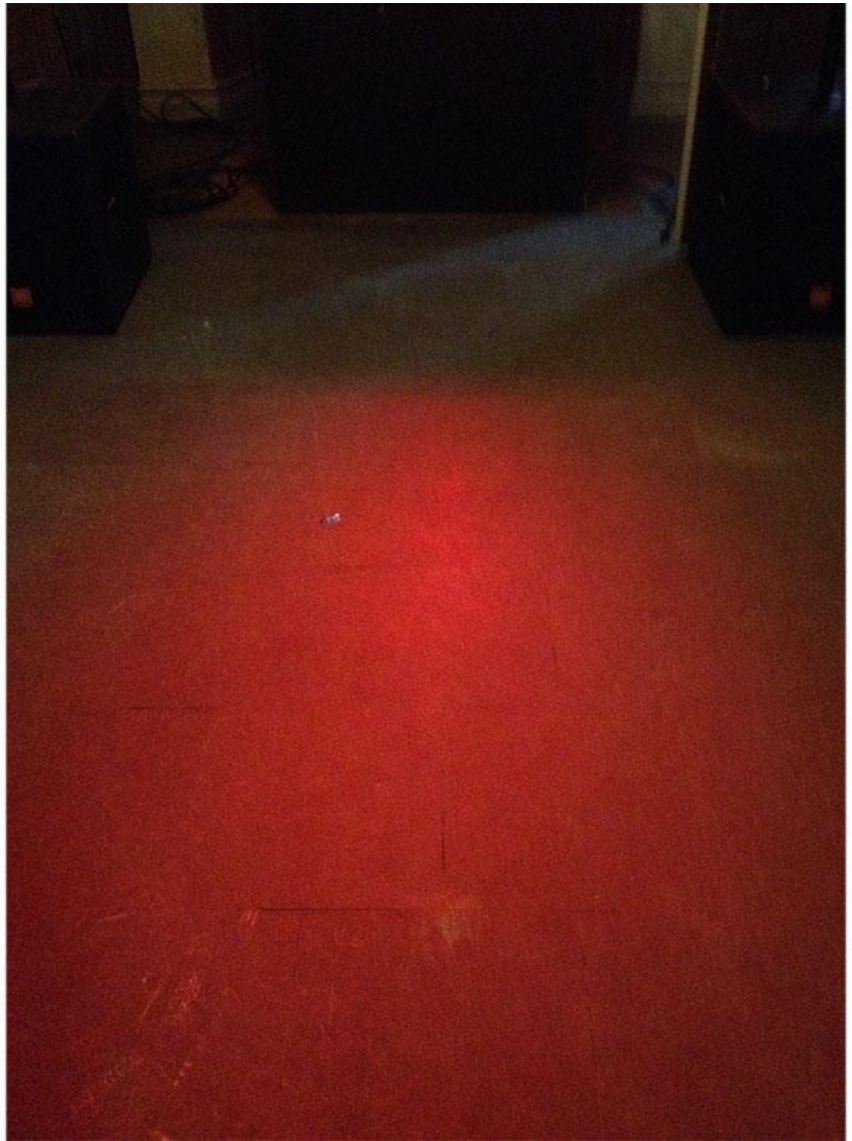
Fresnel Effect

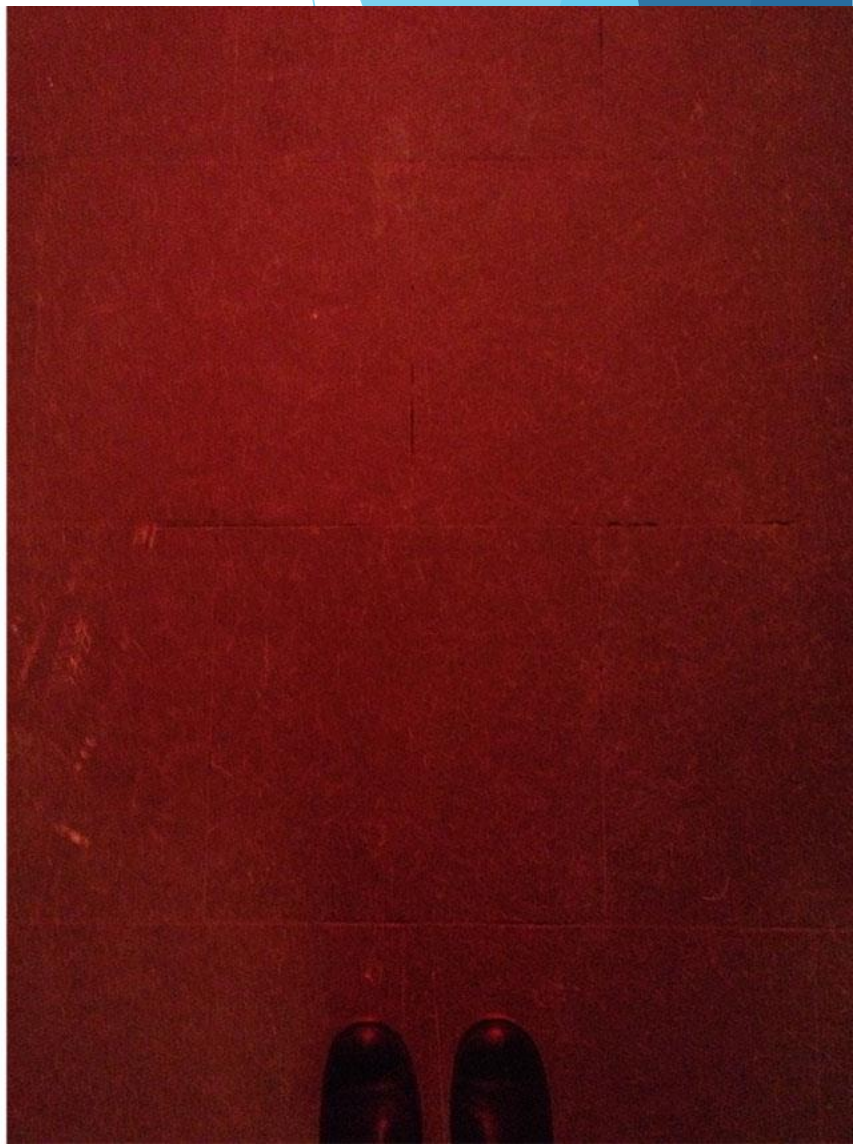


Fresnel Effect

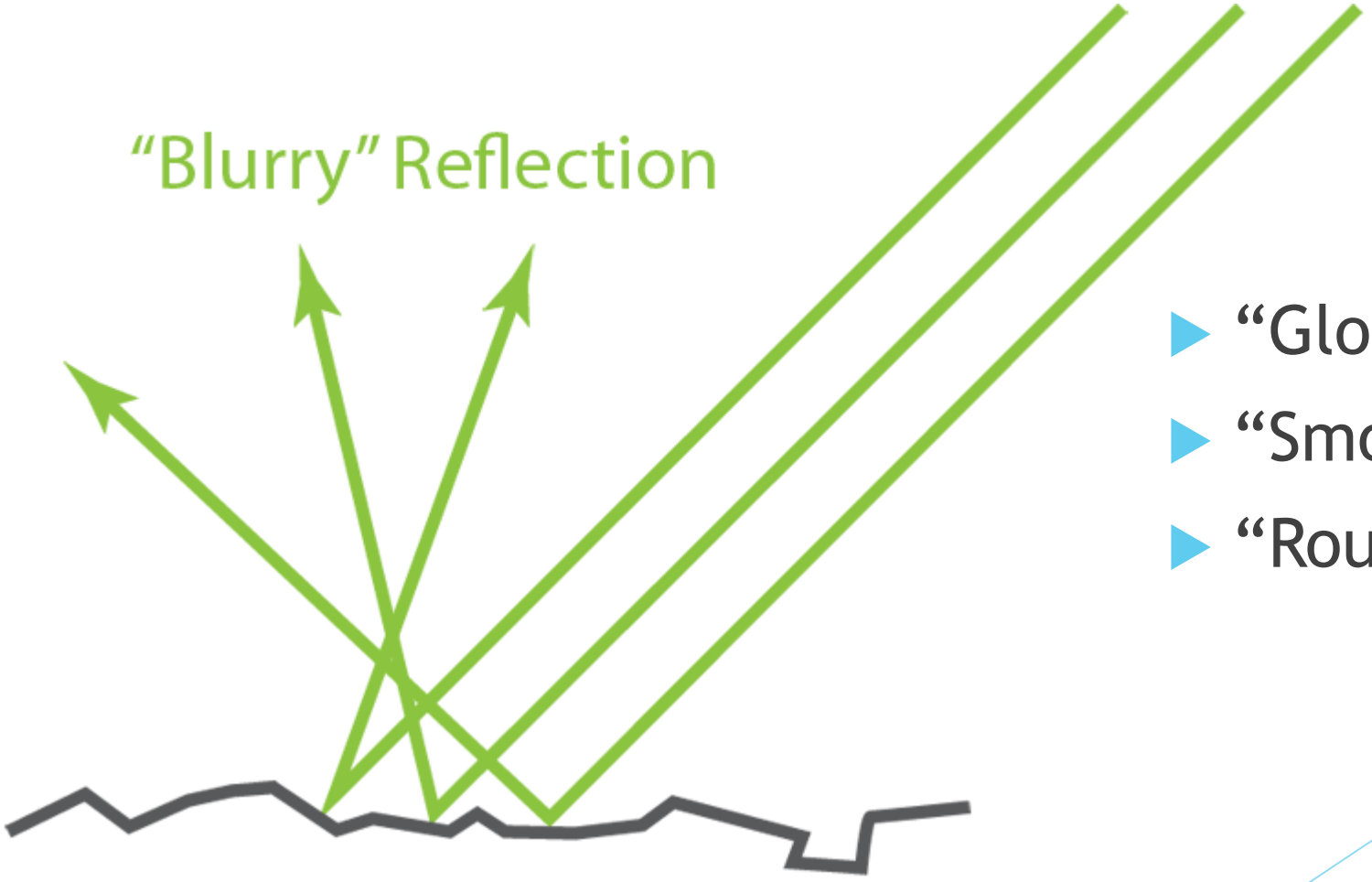
Fresnel Effect





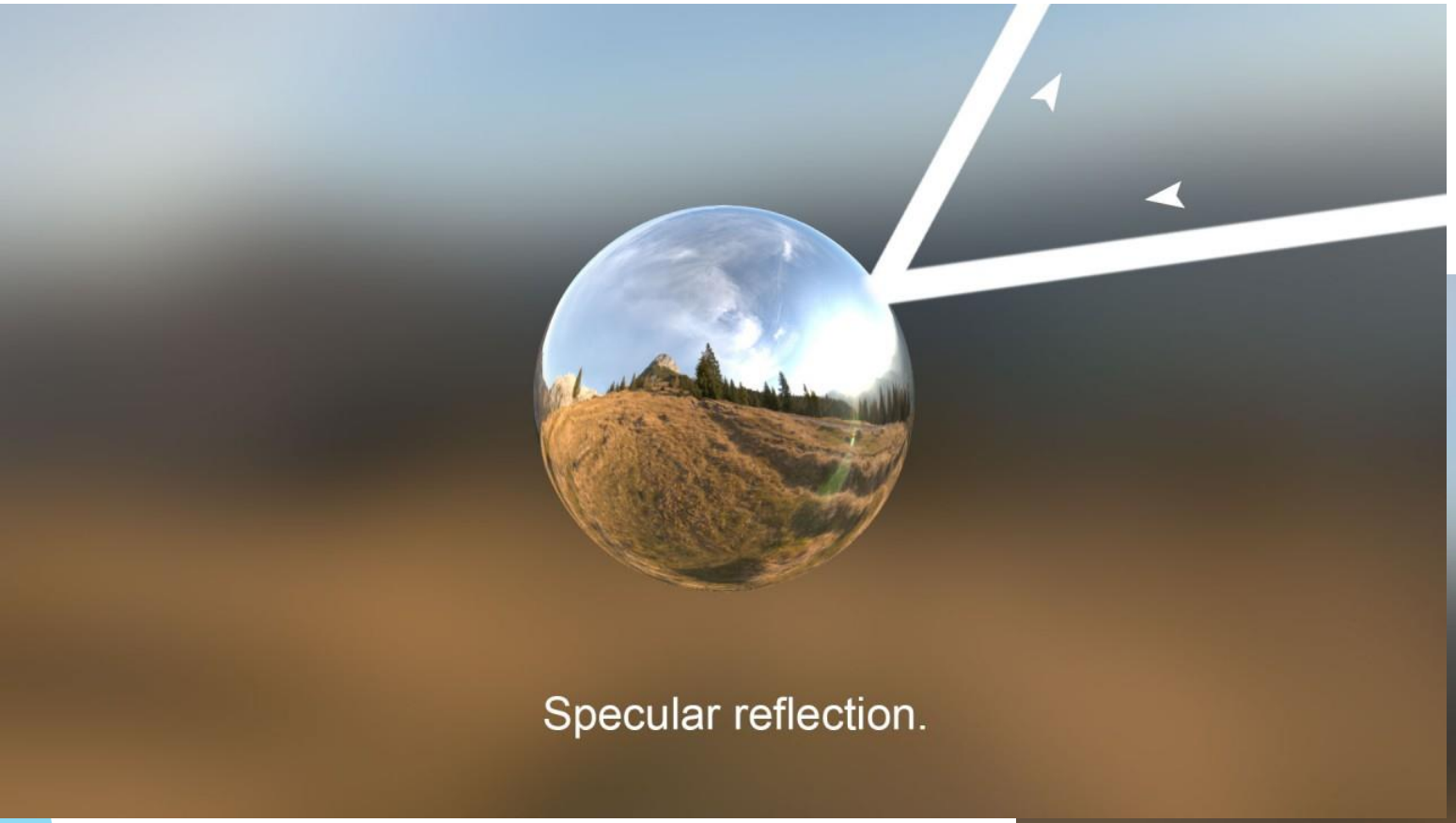


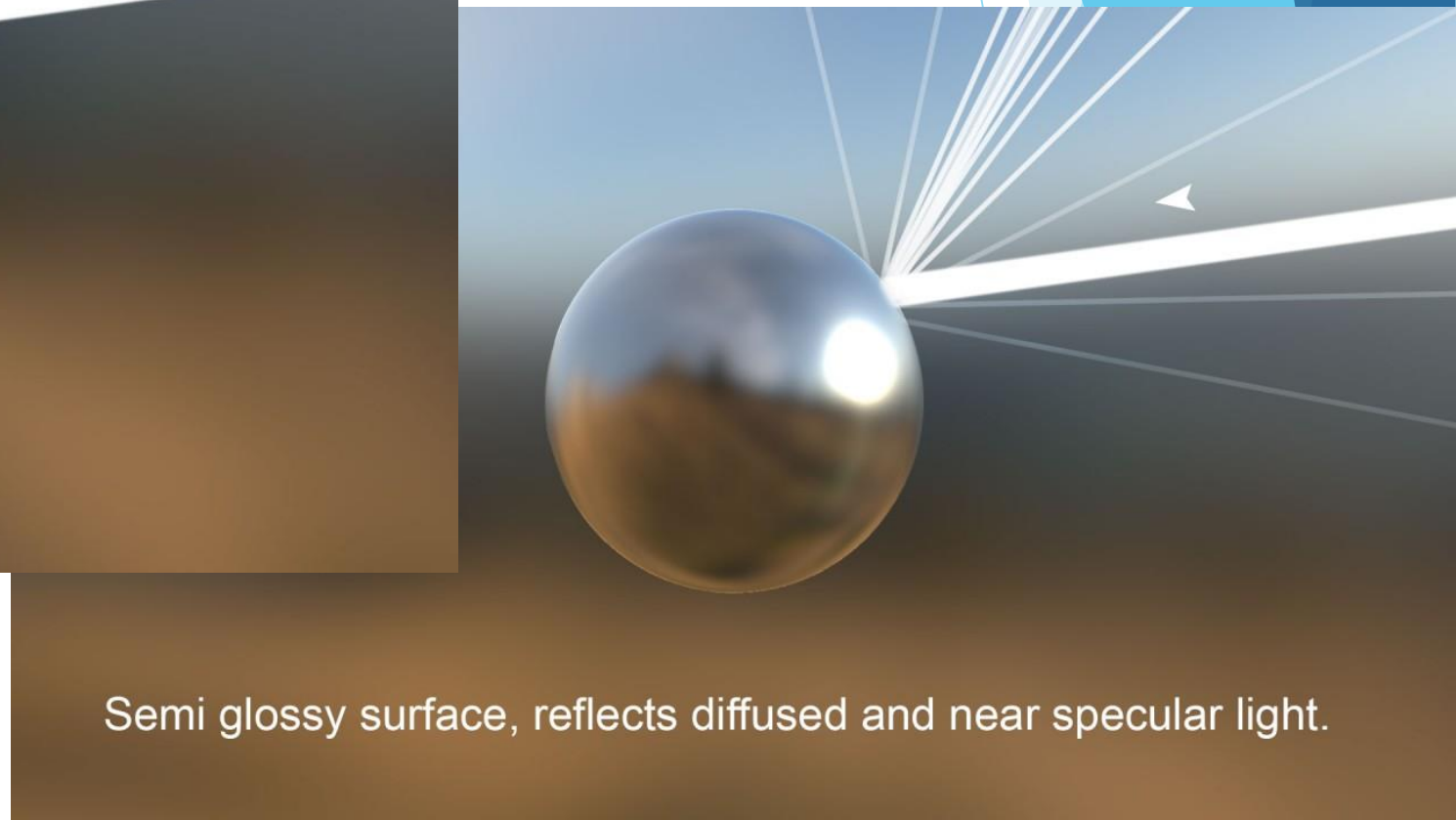
Microsurface



- "Gloss"
- "Smoothness"
- "Roughness"

Microsurface

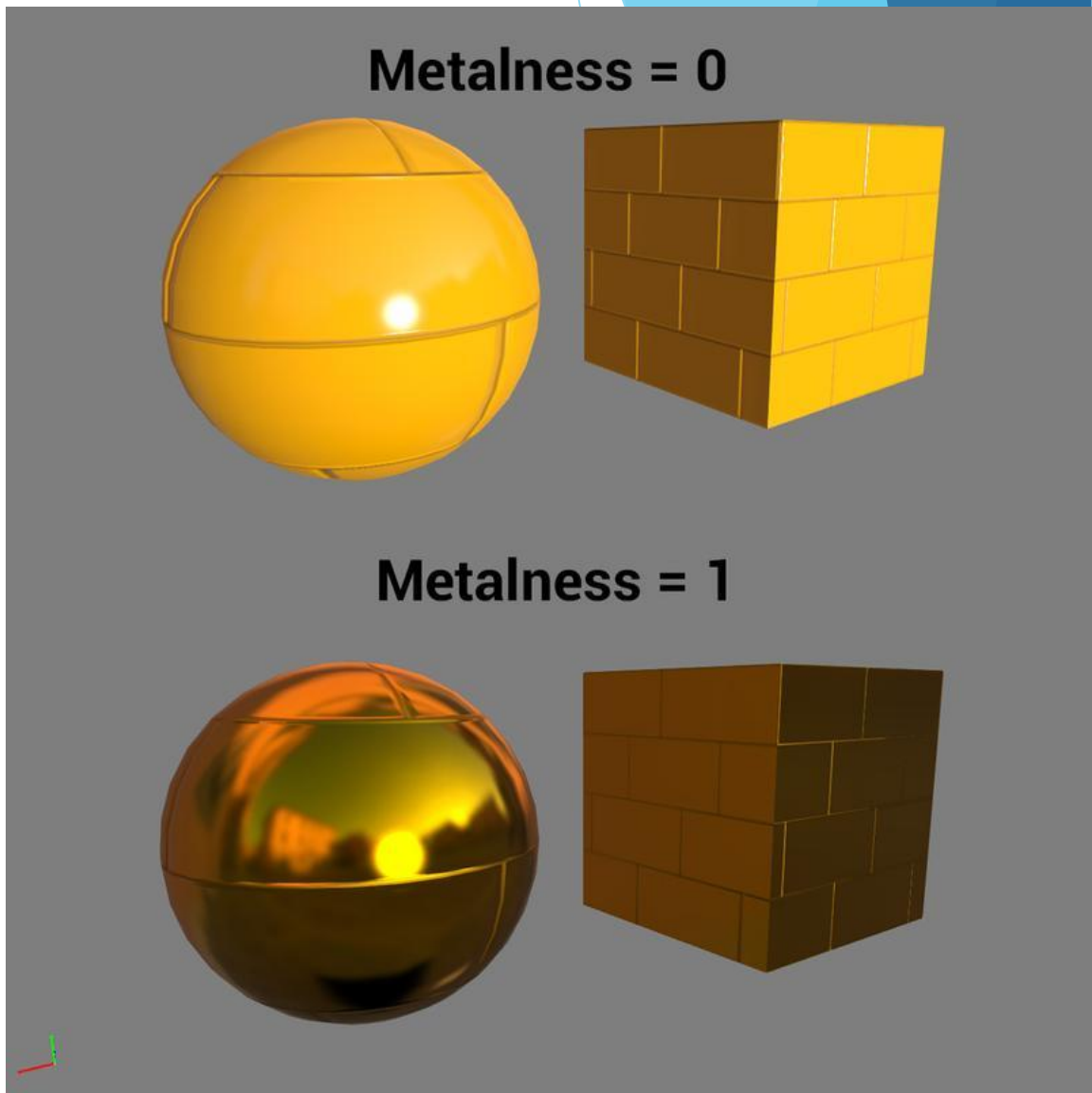




Conductors vs Insulators

- Conductors – 60-90% reflectiveness
- Insulators – 0-20% reflectiveness
- Conductors – reflectivity can vary across the visible spectrum -> tinted reflections
- Conductors will absorb rather than scatter light

- -> metalness



Energy Conservation I

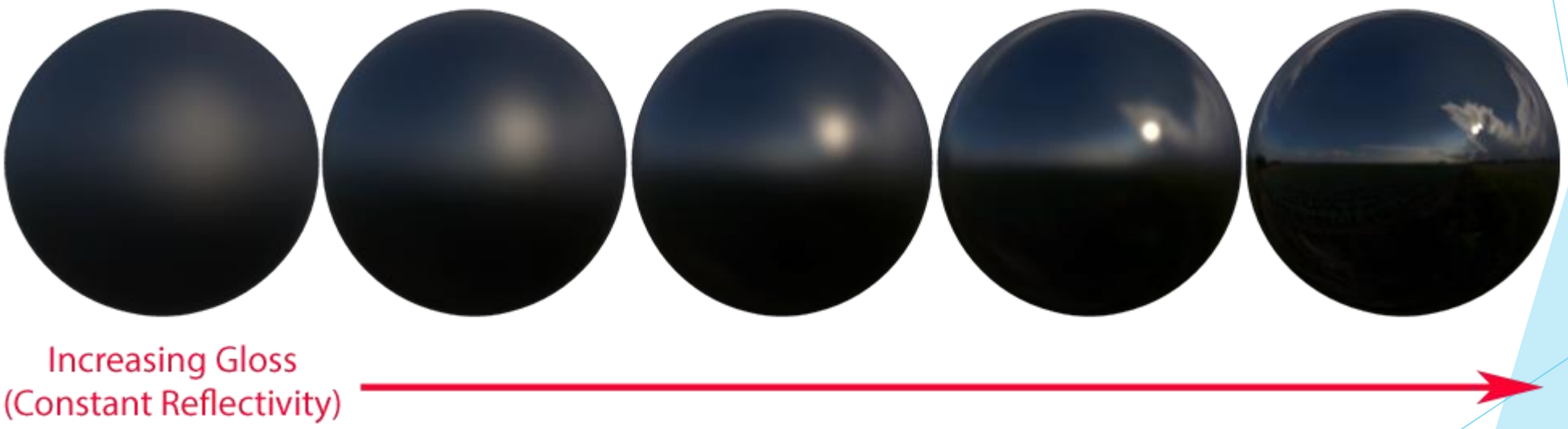
- Objects never reflect more light than they receive

Increasing Reflectivity
(Constant Albedo)

- Before doing diffuse shading, simply subtract the reflected light

Energy Conservation II

- Rougher surface -> smoother surface
  Large, dim highlights -> small, sharp highlights




Microsurface Example: Water & Mud
Rendered in Marmoset Toolbag 2
Mud
Reflectivity: 5%
Gloss: 16%
Water
Reflectivity: 5%
Gloss: 100%

Rendering Equation

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

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

- $L_o(\mathbf{x}, \omega_o, \lambda, t)$ is the total spectral radiance of wavelength λ directed outward along direction ω_o at time t, from a particular position $\mathbf{x}$
- $\mathbf{x}$ is the location in space
- ω_o is the direction of the outgoing light
- λ is a particular wavelength of light
- t is time
- $L_e(\mathbf{x}, \omega_o, \lambda, t)$ is emitted spectral radiance
- $\int_\Omega \ldots \mathrm{d}\omega_i$ is an integral over Ω
- Ω is the unit hemisphere centered around $\mathbf{n}$ containing all possible values for ω_i
- $f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)$ is the bidirectional reflectance distribution function, the proportion of light reflected from ω_i to ω_o at position $\mathbf{x}$, time t, and at wavelength λ
- ω_i is the negative direction of the incoming light
- $L_i(\mathbf{x}, \omega_i, \lambda, t)$ is spectral radiance of wavelength λ coming inward toward $\mathbf{x}$ from direction ω_i at time t
- $\mathbf{n}$ is the surface normal at $\mathbf{x}$
- $\omega_i \cdot \mathbf{n}$ is the weakening factor of outward irradiance due to incident angle, as the light flux is smeared across a surface whose area is larger than the projected area perpendicular to the ray. This is often written as $\cos\theta_i$.

Rendering Equation

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

Rendering Equation

$$L_o(p,\omega_o) = \int_{\Omega} \boxed{f_r(p,\omega_i,\omega_o)} L_i(p,\omega_i)\mathbf{n}\cdot\omega_i d\omega_i$$

- BRDF - Bidirectional reflectance distribution function
- Conservation of energy:

$$\forall\omega_i \int_{\Omega} f_r(p,\omega_i,\omega_o)(\mathbf{n}\cdot\omega_i)d\omega_o \leq 1$$

Translating it to code

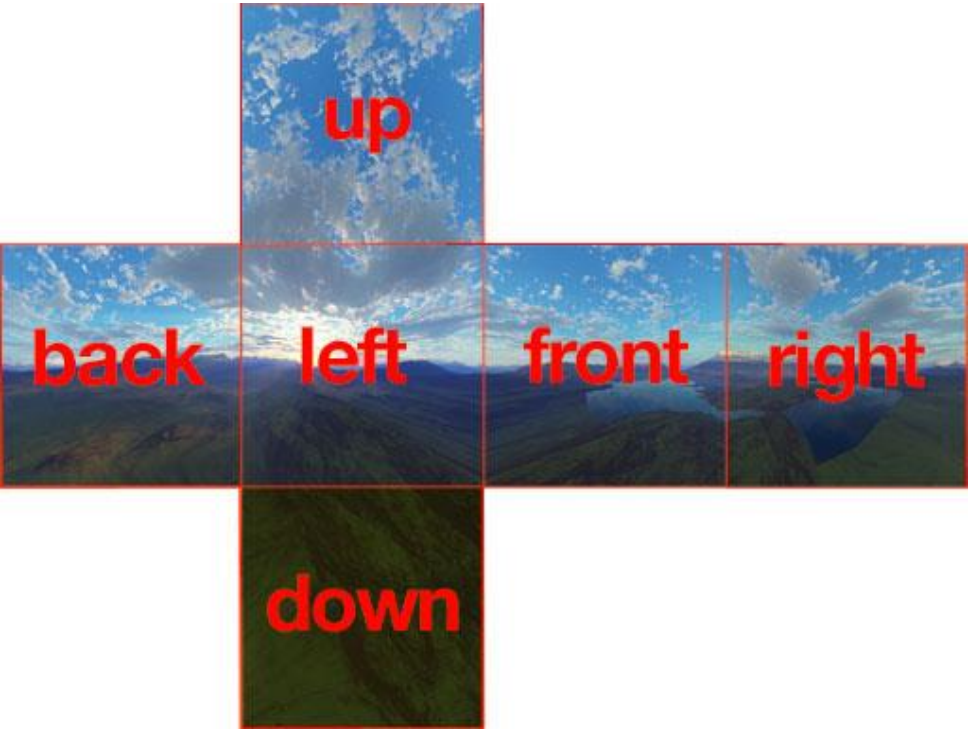
$$L_o(p, \omega_o) = \int_{\Omega} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \mathbf{n} \cdot \omega_i d\omega_i$$

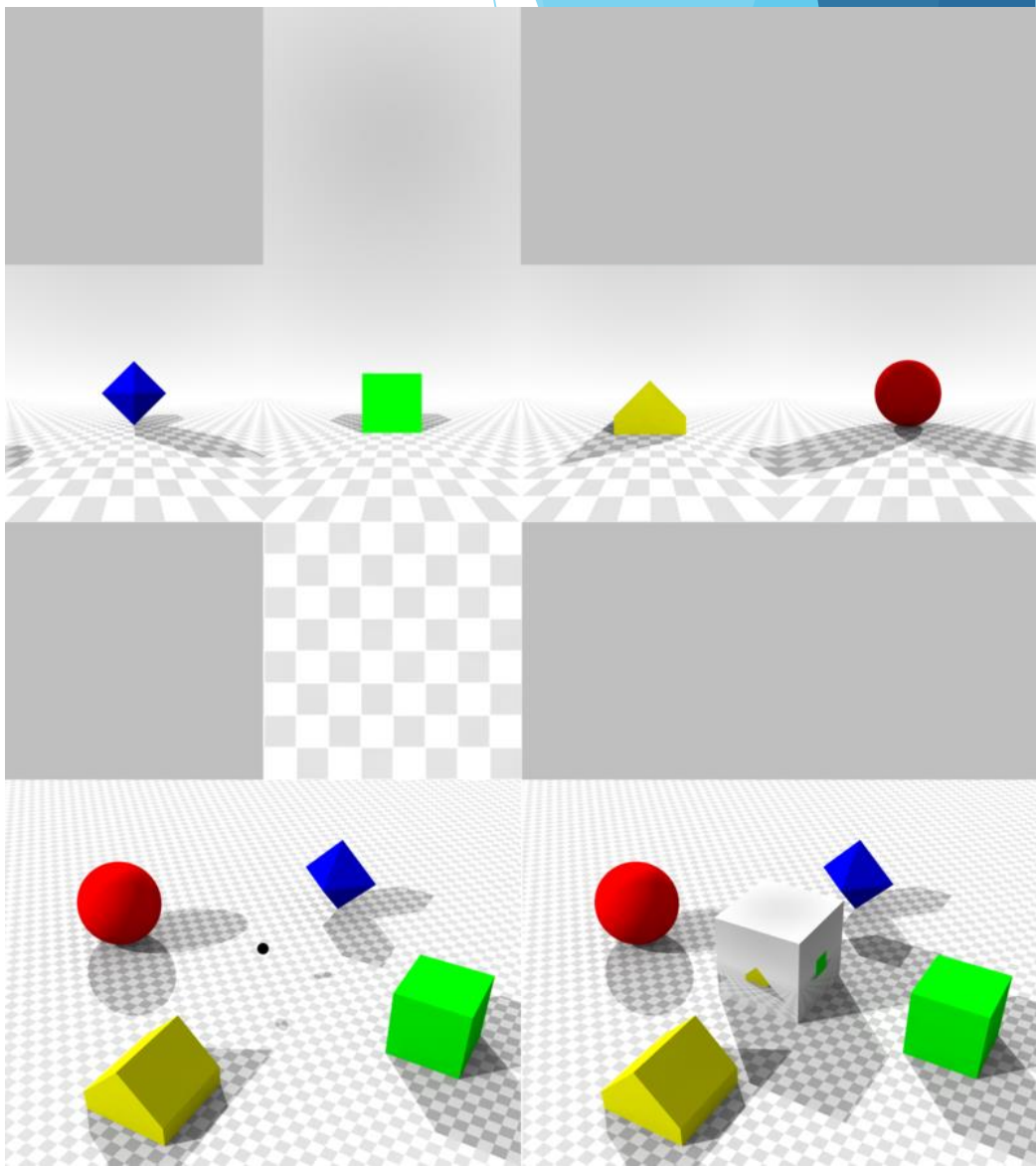
- Two problems:
  - Representing all the radiance functions
  - Solving the integral fast enough

Representing all the radiance functions

- Environment maps (cube maps)

$$L(p,\omega) = texCUBE(cubemap, \omega_p)$$





Solving the integral fast enough

- **Optimize**
- BRDF depends on incoming radiance (which could be constant)
- Using Lambert's BRDF: $f_r(p, \omega_i, \omega_o) = \frac{c}{\pi}$

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

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

Solving the integral fast enough

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

- Now integral depends on only the incidence angle of the light ray
- Precalculate it and store the result in a cubemap

$$L(p, \omega_o) = texCUBE(lambertCubemap, \omega_{op})$$

Solving the integral fast enough

$$L_o(p,\theta_o,\phi_o) = \frac{c}{\pi}\int_{\Phi}\int_{\Theta} L_i(p,\theta_i,\phi_i)cos(\theta_i)sin(\theta_i)d\theta_i d\phi_i$$

$$L_o(p,\theta_o,\phi_o) = \frac{c}{\pi}\frac{2\pi}{N_1}\frac{\pi}{2N_2}\sum^{N_1}\sum^{N_2} L_i(p,\theta_i,\phi_i)cos(\theta_i)sin(\theta_i)$$

$$L_o(p,\theta_o,\phi_o) = \frac{\pi c}{N_1 N_2}\sum^{N_1}\sum^{N_2} L_i(p,\theta_i,\phi_i)cos(\theta_i)sin(\theta_i)$$

The code

```
...
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR
{
    float3 normal = normalize( float3(input.InterpolatedPosition.xy, 1) );
    if(cubeFace==2)
        normal = normalize( float3(input.InterpolatedPosition.x,  1, -input.InterpolatedPosition.y) );
    else if(cubeFace==3)
        normal = normalize( float3(input.InterpolatedPosition.x, -1,  input.InterpolatedPosition.y) );
    else if(cubeFace==0)
        normal = normalize( float3(  1, input.InterpolatedPosition.y,-input.InterpolatedPosition.x) );
    else if(cubeFace==1)
        normal = normalize( float3( -1, input.InterpolatedPosition.y, input.InterpolatedPosition.x) );
    else if(cubeFace==5)
        normal = normalize( float3(-input.InterpolatedPosition.x, input.InterpolatedPosition.y, -1) );

    float3 up = float3(0,1,0);
    float3 right = normalize(cross(up,normal));
    up = cross(normal,right);

    float3 sampledColour = float3(0,0,0);
    float index = 0;
    for(float phi = 0; phi < 6.283; phi += 0.025)
    {
        for(float theta = 0; theta < 1.57; theta += 0.1)
        {
            float3 temp = cos(phi) * right + sin(phi) * up;
            float3 sampleVector = cos(theta) * normal + sin(theta) * temp;
            sampledColour += texCUBE( diffuseCubemap_Sampler, sampleVector ).rgb *
                                        cos(theta) * sin(theta);
            index ++;
        }
    }

    return float4( PI * sampledColour / index), 1 );
}
...
```

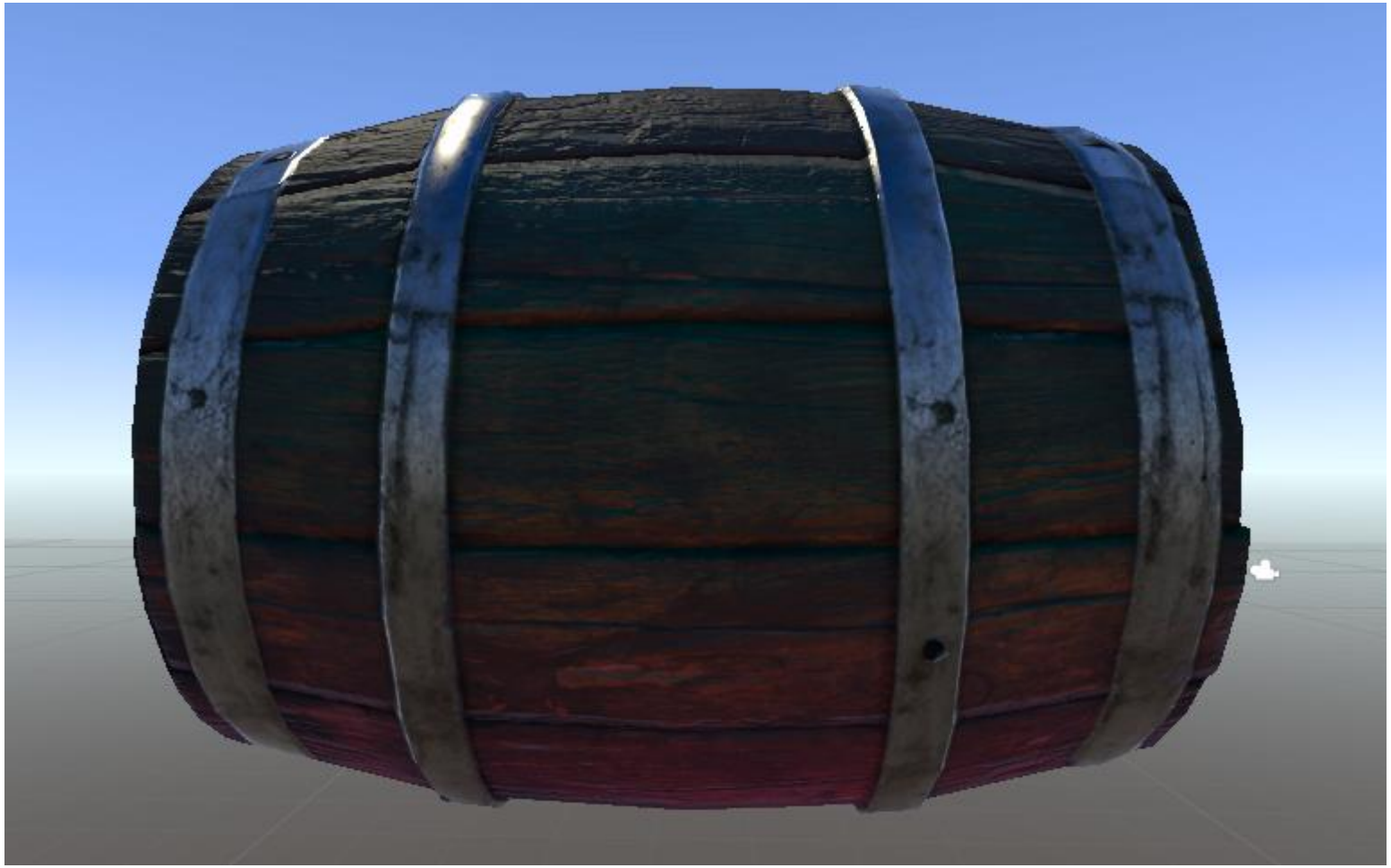
The code

```
...
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR
{
    float3 irradiance= texCUBE(irradianceCubemap_Sampler, input.SampleDir).rgb;
    float3 diffuse = materialColour * irradiance;
    return float4( diffuse , 1);
}
...
```

$$L(p, \omega_o) = texCUBE(lambertCubemap, \omega_{op})$$

The result

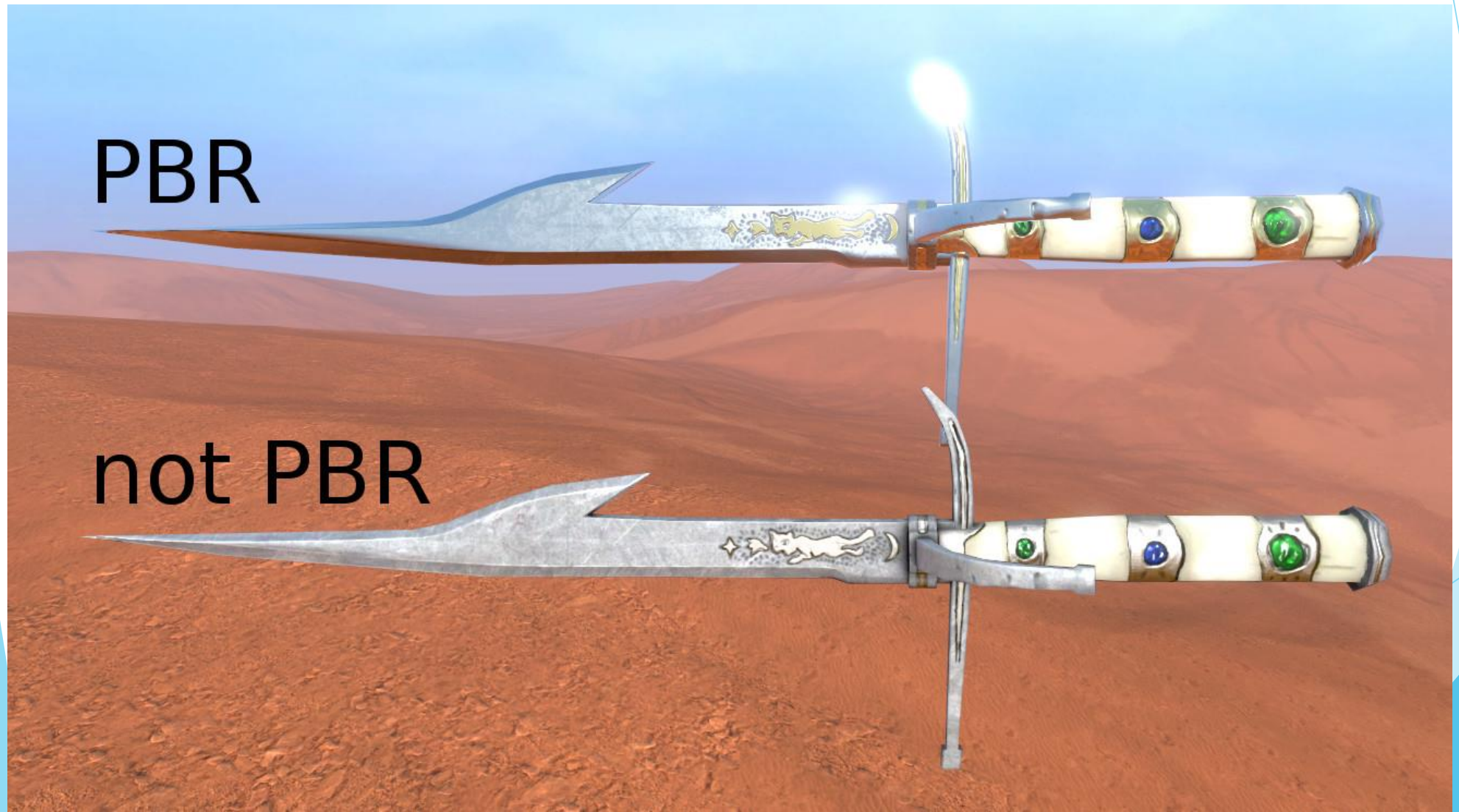
Unity 5




FM 88 92 96 100 104 108
AM 54 60 70 80 100 120 140 160
FM
AM
Volume

Art

Overgrowth vs PBR



Overgrowth vs PBR


FIND A SAFE SPOT TO CROSS THE RIVER

References

- Basic Theory of Physically-Based Rendering by Jeff Russell
  - https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/
- Interaction of Light and Materials
  - http://www.pernroth.nu/lightandmaterials/
- Article - Physically Based Rendering
  - http://www.codinglabs.net/article_physically_based_rendering.aspx
- Nicolas Holzschuch, Romain Pacanowski. A physically accurate reflectance model combining reflection and diffraction.
  - https://hal.inria.fr/hal-01224702/file/RR-8807%20(1).pdf
- Background: Physics and Math of Shading by Naty Hoffman
  - http://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf

References

- Physically Based Shading in Unity 5: A Primer
  - https://blogs.unity3d.com/2014/10/29/physically-based-shading-in-unity-5-a-primer/
- Global Illumination in Unity 5
  - https://blogs.unity3d.com/2014/09/18/global-illumination-in-unity-5/
- Physically-Based Rendering, and You Can Too!
  - https://www.marmoset.co/posts/physically-based-rendering-and-you-can-too/
- “Real-Time Rendering, 3rd edition” by Tomas Akenine-Möller, Eric Haines, and Naty Hoffman
  - http://www.realtimerendering.com/
- http://blog.teamtreehouse.com/beginners-guide-physically-based-rendering-unity

References

- https://www.marmoset.co/skyshop/
- https://en.wikipedia.org/wiki/Rendering_equation
- http://www.ogre3d.org/forums/viewtopic.php?f=25&t=82936
- http://wiki.ktxsoftware.com/Physically-Based-Rendering.html
- http://blog.wolfire.com/2015/10/Physically-based-rendering
- Picture from the game Ryse (and many more examples from games)
  - http://www.neogaf.com/forum/showthread.php?t=1031164
- https://media1.britannica.com/eb-media/80/150780-004-83AAD576.jpg