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

specular reflection

surface scattering

surface / sub-surface scattering

specular reflection & sub-surface scattering
Diffusion

Diffuse reflection.

Absorption 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

Reflectivity vs. Positivity

- Chrome
- Rubber

Base Reflectivities

"Center" vs. "Edge"
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

“Blurry” Reflection

“Gloss”

“Smoothness”

“Roughness”
Microsurface

Specular reflection.

Semi glossy surface, reflects diffused and near specular light.
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

Increasing Gloss (Constant Reflectivity)
\[ L_o(p, \omega_o) = \int_{\Omega} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \mathbf{n} \cdot \omega_i d\omega_i \]
\[ L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t) (\omega_i \cdot \mathbf{n}) \, d\omega_i \]

- \( L_o(\mathbf{x}, \omega_o, \lambda, t) \) is the total \textit{spectral radiance} of wavelength \( \lambda \) directed outward along direction \( \omega_o \) at time \( t \), from a particular position \( \mathbf{x} \)
- \( \mathbf{x} \) is the location in space
- \( \omega_o \) is the direction of the outgoing light
- \( \lambda \) is a particular wavelength of light
- \( t \) is time
- \( L_e(\mathbf{x}, \omega_o, \lambda, t) \) is emitted \textit{spectral radiance}
- \[ \int_{\Omega} \ldots \, d\omega_i \] is an \textit{integral} over \( \Omega \)
- \( \Omega \) is the unit \textit{hemisphere} centered around \( \mathbf{n} \) containing all possible values for \( \omega_i \)
- \( f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) \) is the \textit{bidirectional reflectance distribution function}, the proportion of light reflected from \( \omega_i \) to \( \omega_o \) at position \( \mathbf{x} \), time \( t \), and at wavelength \( \lambda \)
- \( \omega_i \) is the negative direction of the incoming light
- \( L_i(\mathbf{x}, \omega_i, \lambda, t) \) is \textit{spectral radiance} of wavelength \( \lambda \) coming inward toward \( \mathbf{x} \) from direction \( \omega_i \) at time \( t \)
- \( \mathbf{n} \) is the \textit{surface normal} at \( \mathbf{x} \)
- \( \omega_i \cdot \mathbf{n} \) is the weakening factor of outward \textit{irradiance} due to \textit{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 \( \cos \theta_i \).
\[ L_o(p, \omega_o) = \int_{\Omega} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \mathbf{n} \cdot \omega_i \, d\omega_i \]
Rendering Equation

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

- 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: \( f_r(p, \omega_i, \omega_o) = \frac{c}{\pi} \)

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

\[
L_o(p, \omega_o) = \frac{c}{\pi} \int_{\Omega} L_i(p, \omega_i)(\mathbf{n} \cdot \omega_i) d\omega_i
\]
Solving the integral fast enough

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

- 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

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

\[ L_o(p, \theta_o, \phi_o) = \frac{c}{\pi} \frac{2\pi}{N_1} \frac{2\pi}{2N_2} \sum_{N_1} \sum_{N_2} L_i(p, \theta_i, \phi_i) \cos(\theta_i) \sin(\theta_i) \]

\[ L_o(p, \theta_o, \phi_o) = \frac{\pi c}{N_1 N_2} \sum_{N_1} \sum_{N_2} L_i(p, \theta_i, \phi_i) \cos(\theta_i) \sin(\theta_i) \]
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);
}
...

\[ L(p, \omega_o) = \text{texCUBE}(\text{lambertCubemap}, \omega_{op}) \]
The result
Unity 5
Overgrowth vs PBR
Overgrowth vs PBR

Diffuse

Specular

Combined
References

- Basic Theory of Physically-Based Rendering by Jeff Russell
- Interaction of Light and Materials
  - http://www.pernroth.nu/lightandmaterials/
- Article - Physically Based Rendering
  - http://www.codinglabs.net/article_physically_based_rendering.aspx
  - https://hal.inria.fr/hal-01224702/file/RR-8807%20(1).pdf
- Background: Physics and Math of Shading by Naty Hoffman
References

- Physically Based Shading in Unity 5: A Primer

- Global Illumination in Unity 5

- Physically-Based Rendering, and You Can Too!

- “Real-Time Rendering, 3rd edition” by Tomas Akenine-Möller, Eric Haines, and Naty Hoffman

References

- https://www.marmoset.co/skyshop/
- 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)