Shading

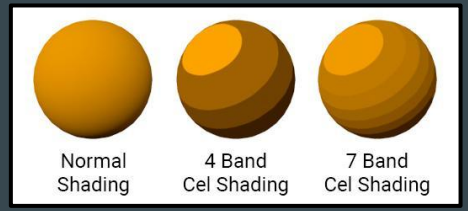
...

Carlos Fuentes

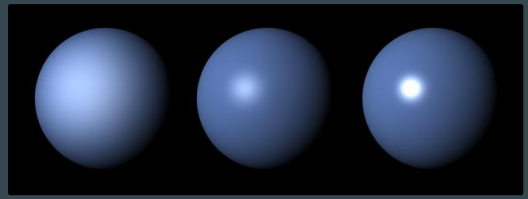
Goal

Shading models

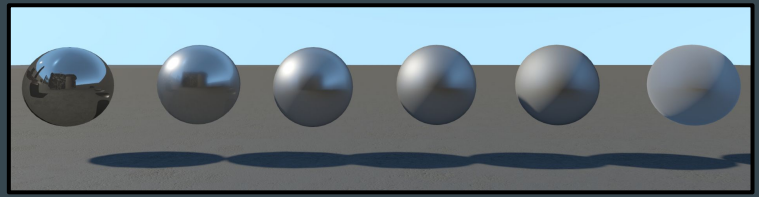
- **Shading** is variation of observed color across and object.
- Shading is caused by the interaction between **material** and **light**.
- Shading models can be divided into two types:
  - Realistic, most of them physically based:
    - **Phong**
    - Microfacet models
  - Non-photorealistic
    - Cel shading



Cel Shading



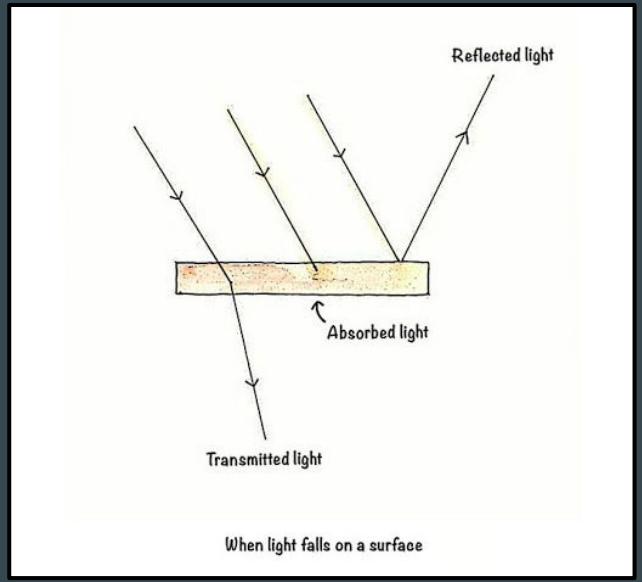
Phong



Microfacets

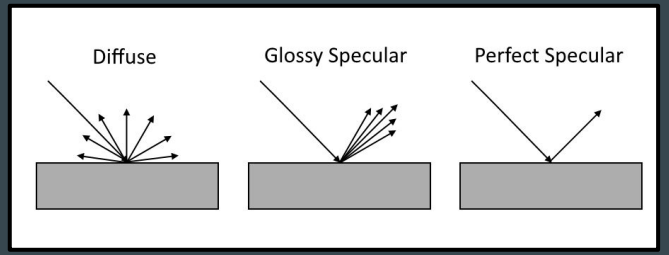
Light and matter interaction

- When light interacts with a matter
  - Some light is **absorbed** : transformed into hot
  - Some light is **reflected** : surfaces reflect light of certain wavelengths (**colors**)
  - Some light is **refracted or transmitted :** can be seen in **transparent/translucen**t materials like glass or water.



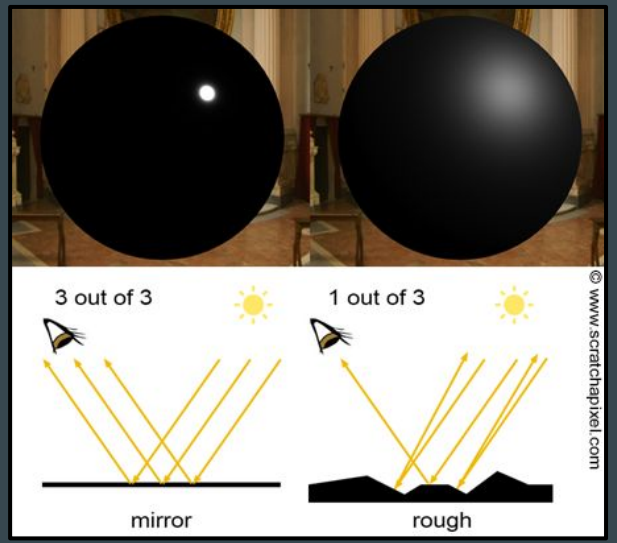
Reflection models

- When light arrives to a surface and is reflected it can be scattered in a range of directions. This scattering can be divided into 4 types:
  - **Diffuse:** light is scattered equally in all directions. Ex: blackboard
  - **Glossy specular:** light is scattered in a set of nearby directions. Shows **specular highlights**. Ex: plastic
  - **Perfect specular:** light is scattered in one single direction (depends on the normal of the surface). Ex: mirror



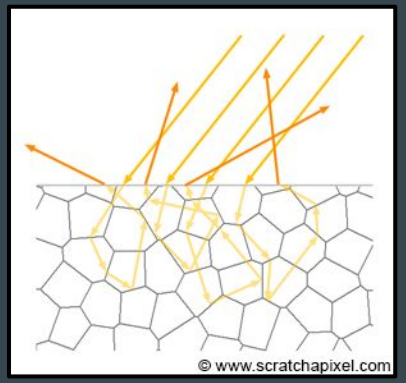
Glossy surfaces

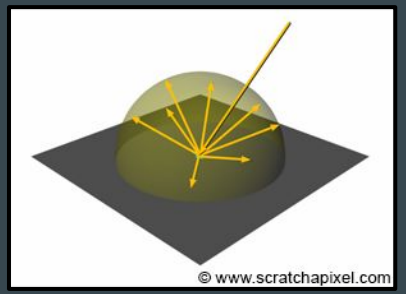
- Few surfaces are perfect specular like mirrors, most surfaces are **glossy specular or diffuse**.
- What makes a surface glossy ?
  - A glossy material is in fact a mirror whose surface isn't smooth. Its **roughness** can be seen as a collection of very small mirrors oriented in directions slightly different to surface.
  - The **rougher** the material is, the **blurrier** the reflected light.



Diffuse surfaces

- Light in **diffuse** materials are slightly transmitted into them and **re-emerges** after bouncing inside it.
- This bounces makes reflection uncorrelated with incident angle. The **direction** of a ray leaving the surface is said to be **random**.
- For this reason we see diffuse objects as reflecting light **equally in any direction**.
- This transmission and re-emerging is done in a **microscopic** scale





Phong model

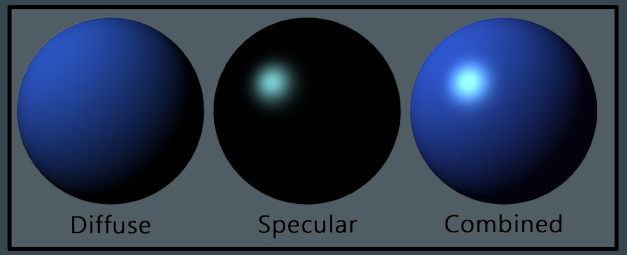
- **Phong** model is a reflection model that tries to describe how light interacts with materials that exhibit **diffuse** and **glossy specular** reflections.
- The reason why materials exhibits more than one type of reflection is not always the same, but for example it can be composite of **different materials**.

Phong model

- So, Phong model can be conceptually modelled using this **equation**

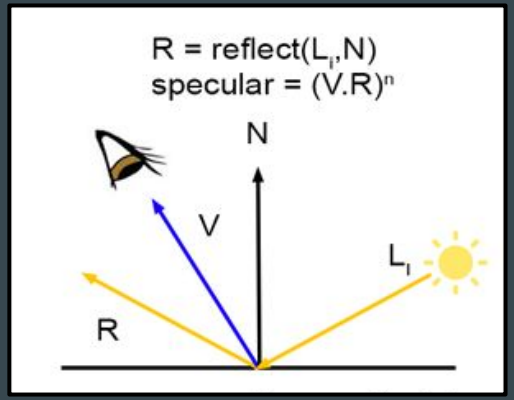
$$S_p = diffuse() * K_d + specular() * K_s$$

- Where S_p (shading at point p) is the diffuse and specular components combined using K_d and K_s constants depending on the material.



Specular

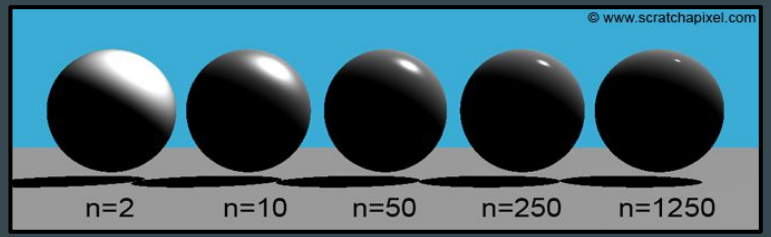
- Phong observed that can simulate glossy effect by computing the light **perfect ray reflection** R.
- A perfect specular reflection can only be seen if it coincides with **view direction** V.
- For glossy specular reflection, reflected ray directions and can be **slightly different** from R due to material roughness.



$$R = reflect(L, N)$$
$$R = L - 2 * (N \cdot L) * N$$

Specular

- **Dot product** between V and R is 1 if both directions are the same and decreases to 0 until both are perpendicular.
- Phong models specular as $specular = (V \cdot R)^n$
- Where n (**shininess**) models polishness of the material.
- The bigger shininess, the faster decrease of dot product.



Diffuse

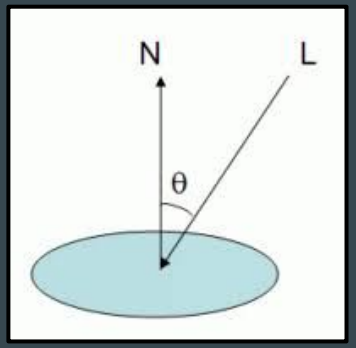
- Diffuse reflection is **equal in any direction**, so diffuse function should be

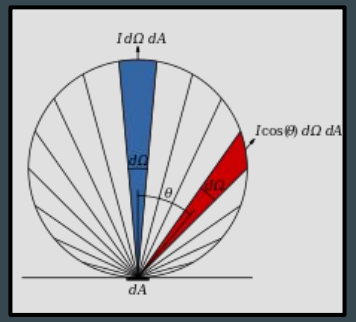
$$diffuse = 1$$

- But, it can be observed that, in perfect diffuse surface, that light reflected is

$$diffuse = cos(\phi_{incidence}) = -(N \cdot L)$$

- This is because of **Lambert cosine law**: the amount of light projected from a light source through a surface is proportional to the cosine of the angle between direction of light (L) and normal of the surface (**N**)
- Diffuse depends on **light direction** but not on **view direction**

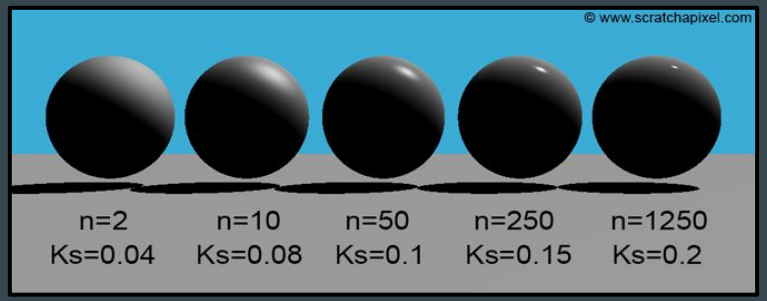




The number of light photons directed into any wedge is proportional to the area of the wedge

Constants

- Phong is an empirical model so constants $\mathbf{K_d}$ (from 0 to 1 models how much diffuse the material is) $\mathbf{K_s}$ (from 0 to 1 models how much specular the material is) and **n** (shininess) actually has **no physical meaning**
- They must be tweaked until desired result has been achieved



Colors

- **Light** has its own color $\text{Color}_{\text{light}}$
- Diffuse reflection is affected by $\text{Color}_{\text{light}}$ and object **diffuse color** $\text{Color}_{\text{diffuse}}$
- Specular reflections are of the **color of light** (so don't tint it with diffuse color)
- So finally our **Phong reflection model equation** is

$$K_d * Color_{diffuse} * Color_{light} * (N \cdot L) + K_s * Color_{light} * (V \cdot R)^n$$

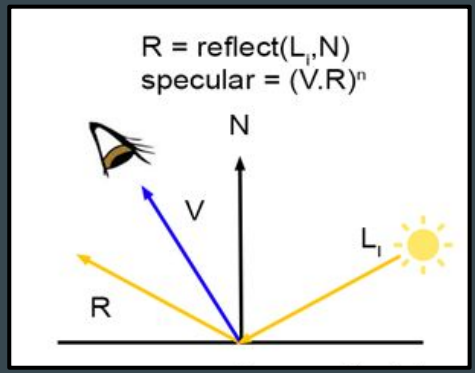
- With $\mathbf{K_d}$, $\mathbf{K_s}$ **and n** as constants used for describing different materials

Ambient color

- Lighting is only applied to faces in front of light

$$(N \cdot L) > 0$$

- So, due to lighting, faces whose normal is in the same direction of light will be shade **black**
- **Ambient color** is a dark color that is always added to our equation and helps to avoid this problem.



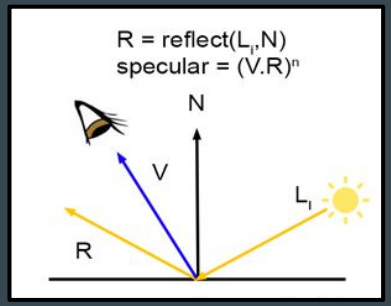
Ambient color

- Ambient color must be **tinted with diffuse color**
- You can think ambient color as **light coming from scene**
- **Phong equation** with ambient color added:

$$Color_{ambient} * Color_{diffuse} + K_d * Color_{diffuse} * Color_{light} * (N \cdot L) + K_s * Color_{light} * (V \cdot R)^n$$

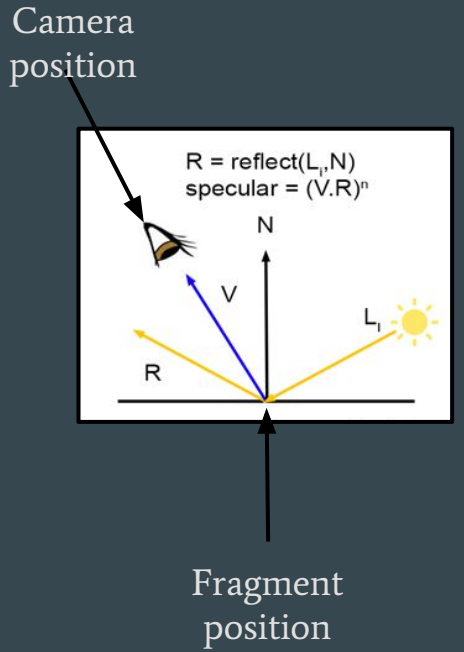
Phong implementation

- We are going to apply Phong **per fragment**
- **Light direction L** will be an uniform passed to the shader. Must be normalized.
- Per **fragment normal N** will be needed:
    a. Add vertex **normal attribute** to our VBO and vertex shader
    b. Add **out/in** normal global variable to vertex and fragment shader
    c. Vertex normal must be transformed to world coordinates using **model matrix** (remember normal transform rules from transformations class!!!!)
    d. Normal will arrive to fragment shader interpolated from 3 vertices ⇒ **normalize**



Phong implementation

- V vector needs to be computed from **fragment position** and **camera position**:
  - Create **in/out position** variable in fragment/vertex shader
  - position must be transformed by **model matrix** in vertex shader
  - V must be **normalized**
  - Camera position is an uniform computed from **camera or view transform**



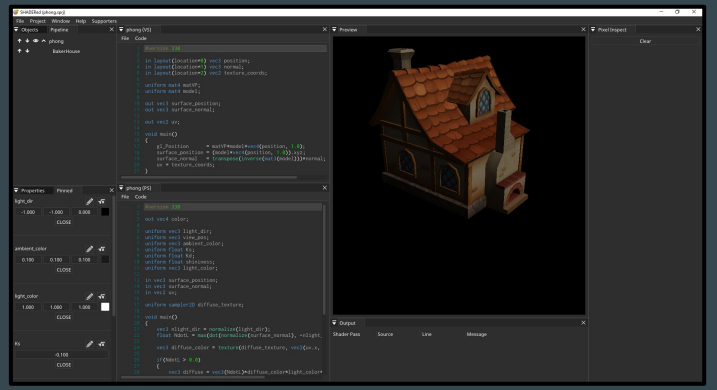
Phong implementation

- **$Color_{ambient}$, $Color_{diffuse}$ and $Color_{light}$** are uniforms passed to the shader
- $Color_{diffuse}$ can be extracted from **texture** (AKA diffuse texture) if we have it.
- **K_s and K_d and n (shininess)** are also uniforms passed to the shader
- Use **ImGui** to add all this uniforms and light direction into a menu for tweaking
- Remember **final equation**

$$Color_{ambient} * Color_{diffuse} + K_d * Color_{diffuse} * Color_{light} * (N \cdot L) + K_s * Color_{light} * (V \cdot R)^n$$

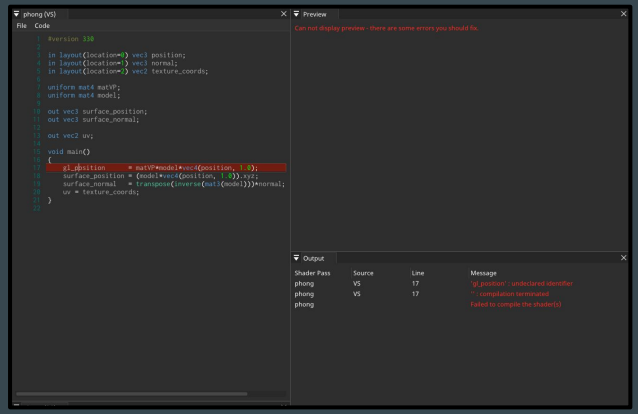
Prototyping

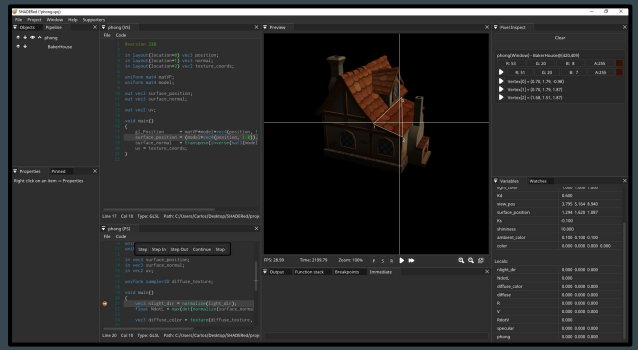
- It’s recommendable to use a shader editor to prototype shaders when possible
- They provide a platform to quickly test our shaders with common primitives: spheres, cubes, etc..
- They can also use custom geometries, textures, etc...



Prototyping

- ShaderEd allows you to:
  - Get **compiling errors** while editing
  - Display **partial results** easily
  - Test first with common primitives: **spheres**, cubes
  - **Debug shaders:** breakpoints, inspection, step by step execution.





Prototyping

- Recommended steps:
  a. Use ShaderEd to implement Phong in a **sphere** primitive (is easy to see diffuse and glossy shines)
  b. Improve ShaderEd Phong implementation using **Baker house** model and adding diffuse texture.
  c. Copy shaders to our **Engine**
     - Add normals to **VBO** and needed **uniforms**
     - Use **ImGui** to tweak needed uniforms