Phong Shading and Lighting

Lighting formulas

Light equation

- The two shaders must put together some equation to come up with pixel brightness

- We can find out what sort of equation we can make by looking at what the inputs of the shader are - what information is available?

Light equation starts with just all this info

- Elements to be drawn
  - Per vertex in each element:
    - Positions - all of them
    - Normals (the full set of perpendicular vectors for all points)
    - Colors if you want some per vertex
    - Texture coords

- Per whole draw call:
  - Matrices
  - Flags
    - Does this shape consult a texture image or not?
    - Color this shape solid? Set color = normals?
    - Specialized program just for this shape? etc.
  - Light position (and specify if point or vector)
  - Material properties of shape - how chalky/shiny its interaction with light source is

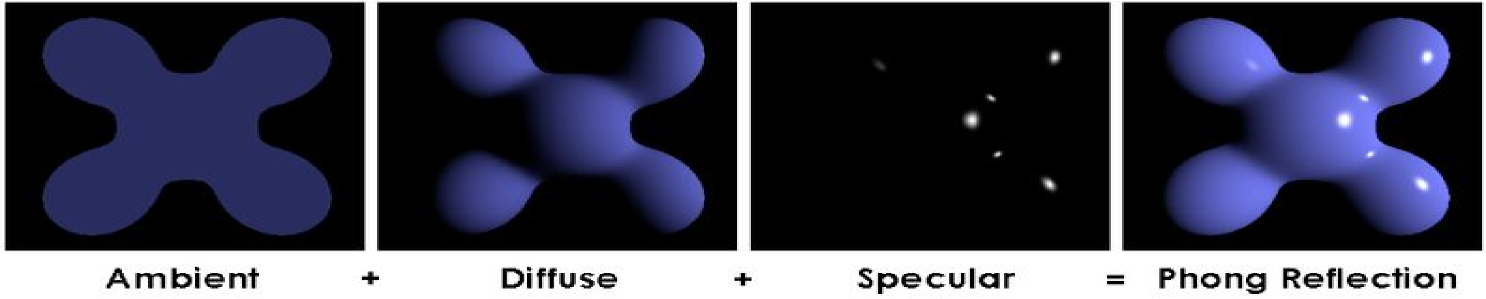
Light equation starts with just all this info

- Normal is the most important input to light equation

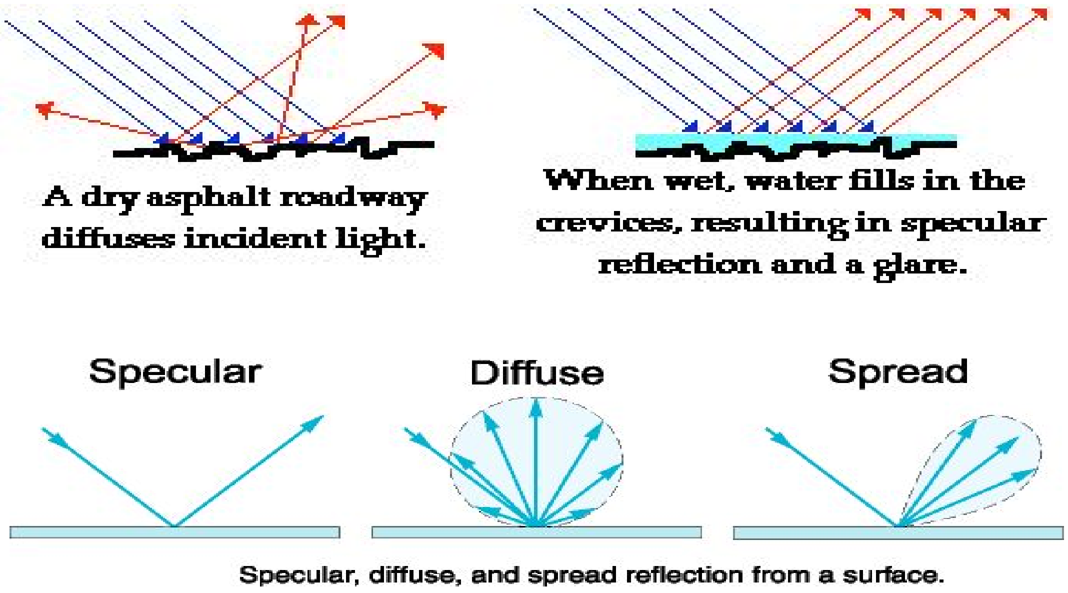
- Intermediate calculations: Compute eye, L

- Our equation choice: Phong model

Components of light

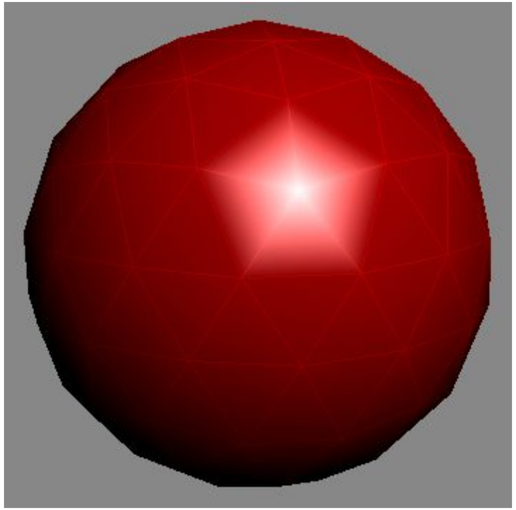


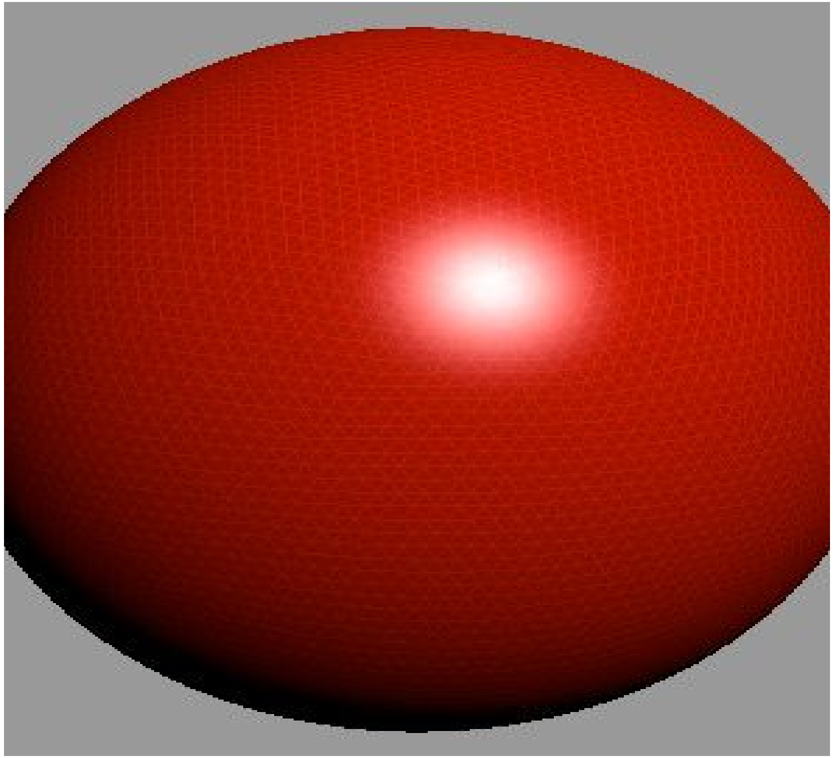
Combining components of light



Gouraud shading

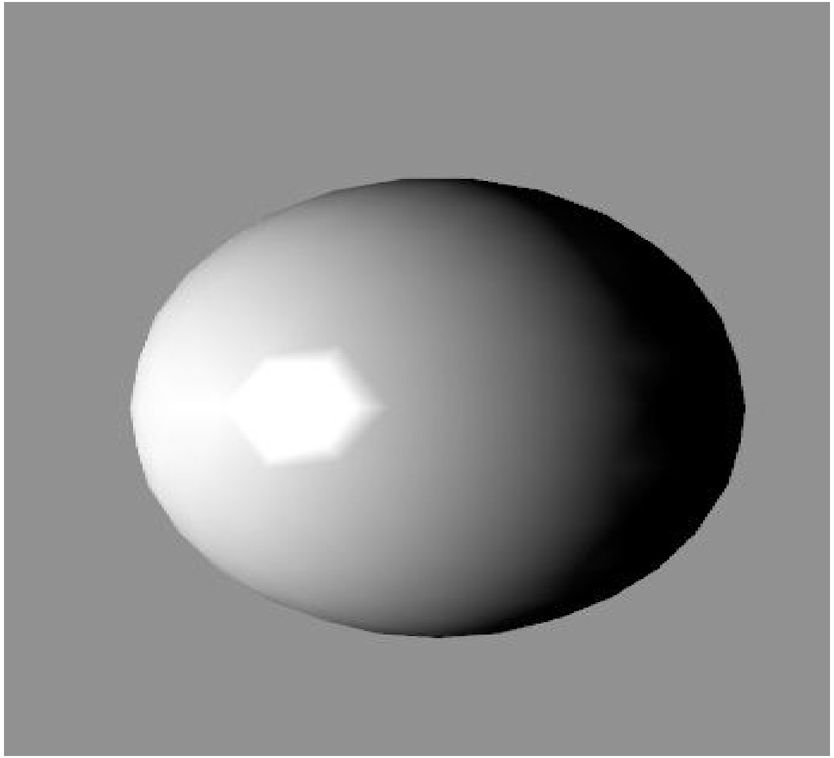
- Simplest method: Just assign brightnesses per vertex and interpolate
- The specular highlight performs poorly at edges
- Only solution - make edges matter less by increasing polygon count





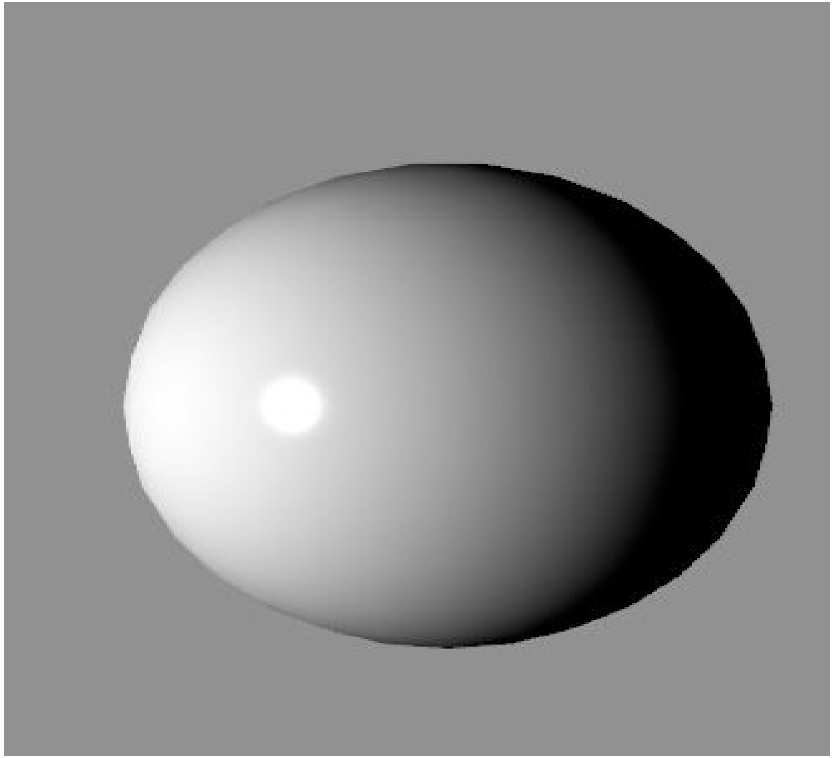
Light equation

- Gouraud shading
  - Diffuse and specular components calculated at every point and added together
  - Linearly interpolate the brightnesses



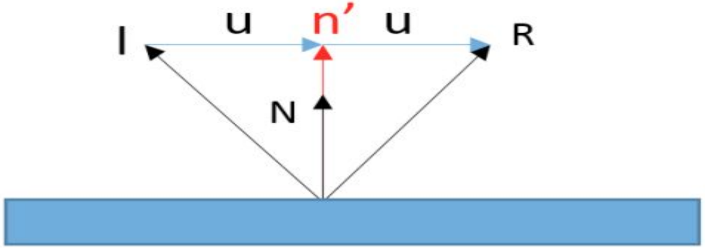
Light equation

- Phong shading
  - Linearly interpolate the <u>normals</u> across each triangle
  - Only when you get to an actual pixel do you calculate the specular and diffuse brightnesses
  - At <u>every pixel</u>, a much finer scale than Gouraud



Light equation

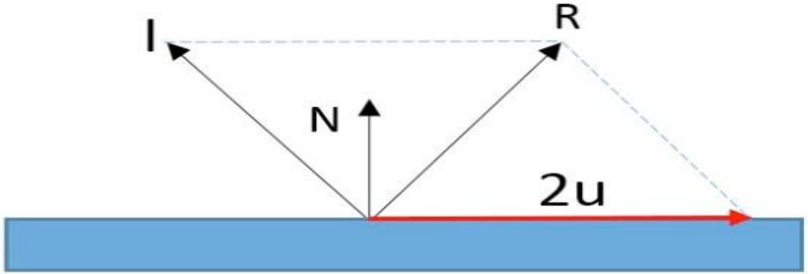
- Calculating R, the (non-physical, made-up) reflection of the point light source



The $\vec{n'}$ is the projection of $\vec{I}$ on $\vec{N}$

$$\vec{n'} = (\vec{N} \cdot \vec{I})\, \vec{N}, \text{ with } ||\vec{N}||^2 = 1$$

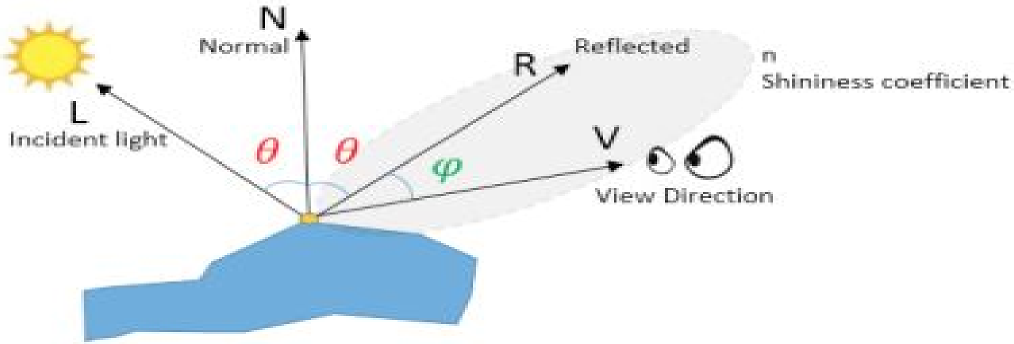
$$\vec{u} = \vec{n'} - \vec{I}$$



$$\vec{R} = \vec{I} + 2\vec{u} = \vec{I} + 2(\vec{n'} - \vec{I})$$

$$\vec{R} = 2(\vec{N} \cdot \vec{I})\, \vec{N} - \vec{I}$$

Light equation



I = emissive + ambient + diffuse + specular

$$\text{emissive} = k_e$$

$$\text{ambient} = k_a * ambientColor$$

$$\text{diffuse} = k_d * lightColor * \cos(\theta)$$

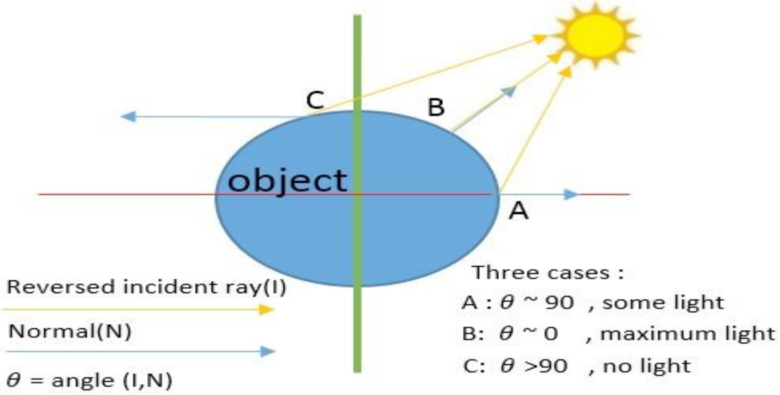
$$= k_d * lightColor * \max(0, N \cdot L)$$

$$\text{specular} = k_s * lightColor * \cos(\varphi)^n$$

$$= k_s * lightColor * \max(0, R \cdot V)^n$$

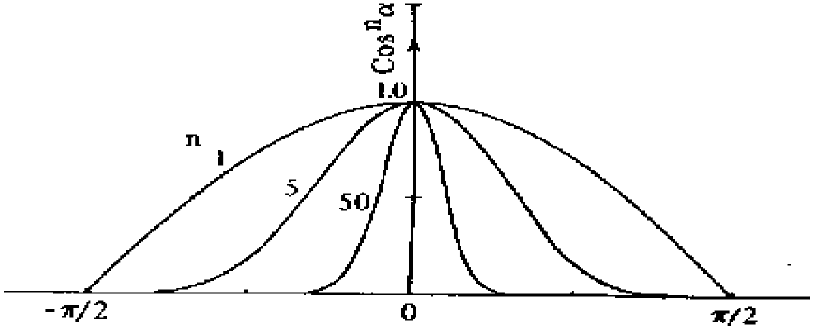
Lambert’s law

“The amount of reflected light is proportional with the cosine (dot product) of the angle between the normal and incident vector”



Specular term - Smoothness exponent effect

- Exponentiating a function that has values < 1 draws those values closer to zero
- Higher exponent = smaller region where point light's reflection is considered "aligned" with the viewer.
- Smaller shiny spot



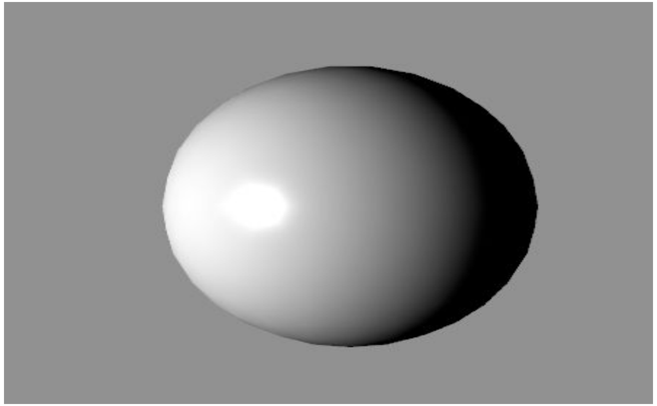
Material properties - coefficients

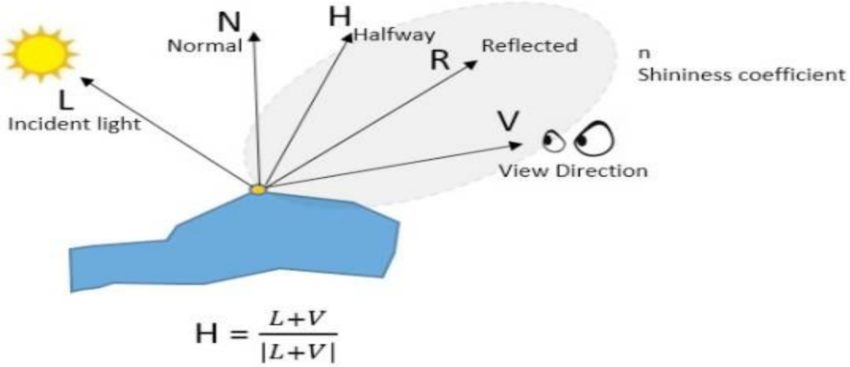
Name	Ambient			Diffuse			Specular			Shininess
emerald	0.0215	0.1745	0.0215	0.07568	0.61424	0.07568	0.633	0.727811	0.633	0.6
jade	0.135	0.2225	0.1575	0.54	0.89	0.63	0.316228	0.316228	0.316228	0.1
obsidian	0.05375	0.05	0.06625	0.18275	0.17	0.22525	0.332741	0.328634	0.346435	0.3
pearl	0.25	0.20725	0.20725	1	0.829	0.829	0.296648	0.296648	0.296648	0.088
ruby	0.1745	0.01175	0.01175	0.61424	0.04136	0.04136	0.727811	0.626959	0.626959	0.6
turquoise	0.1	0.18725	0.1745	0.396	0.74151	0.69102	0.297254	0.30829	0.306678	0.1
brass	0.329412	0.223529	0.027451	0.780392	0.568627	0.113725	0.992157	0.941176	0.807843	0.21794872
bronze	0.2125	0.1275	0.054	0.714	0.4284	0.18144	0.393548	0.271906	0.166721	0.2
chrome	0.25	0.25	0.25	0.4	0.4	0.4	0.774597	0.774597	0.774597	0.6
copper	0.19125	0.0735	0.0225	0.7038	0.27048	0.0828	0.256777	0.137622	0.086014	0.1
gold	0.24725	0.1995	0.0745	0.75164	0.60648	0.22648	0.628281	0.555802	0.366065	0.4
silver	0.19225	0.19225	0.19225	0.50754	0.50754	0.50754	0.508273	0.508273	0.508273	0.4
black plastic	0.0	0.0	0.0	0.01	0.01	0.01	0.50	0.50	0.50	.25
cyan plastic	0.0	0.1	0.06	0.0	0.50980392	0.50980392	0.50196078	0.50196078	0.50196078	.25
green plastic	0.0	0.0	0.0	0.1	0.35	0.1	0.45	0.55	0.45	.25
red plastic	0.0	0.0	0.0	0.5	0.0	0.0	0.7	0.6	0.6	.25
white plastic	0.0	0.0	0.0	0.55	0.55	0.55	0.70	0.70	0.70	.25
yellow plastic	0.0	0.0	0.0	0.5	0.5	0.0	0.60	0.60	0.50	.25
black rubber	0.02	0.02	0.02	0.01	0.01	0.01	0.4	0.4	0.4	.078125
cyan rubber	0.0	0.05	0.05	0.4	0.5	0.5	0.04	0.7	0.7	.078125
green rubber	0.0	0.05	0.0	0.4	0.5	0.4	0.04	0.7	0.04	.078125
red rubber	0.05	0.0	0.0	0.5	0.4	0.4	0.7	0.04	0.04	.078125
white rubber	0.05	0.05	0.05	0.5	0.5	0.5	0.7	0.7	0.7	.078125
yellow rubber	0.05	0.05	0.0	0.5	0.5	0.4	0.7	0.7	0.04	.078125

Multiply the shininess by 128!

Phong-Blinn

- Combine V and L, the two constants in the scene, into one vector
- Given H = halfway between V and L, Use (H dot N) instead of (R dot V)
- If directional light, you can compute H once per frame per light source and it's the same everywhere in the scene - no dependence on normal, just viewer and light
- Re-use it instead of re-calcuating in shader - shader only has to dot H with each N - cheap
- Also behaves better at glancing angles





Light equation

- How many times to do the lighting equation?
  - Once per triangle - flat
  - Once per point - gouraud
  - Once per pixel - smooth / "phong shading"