TA 4

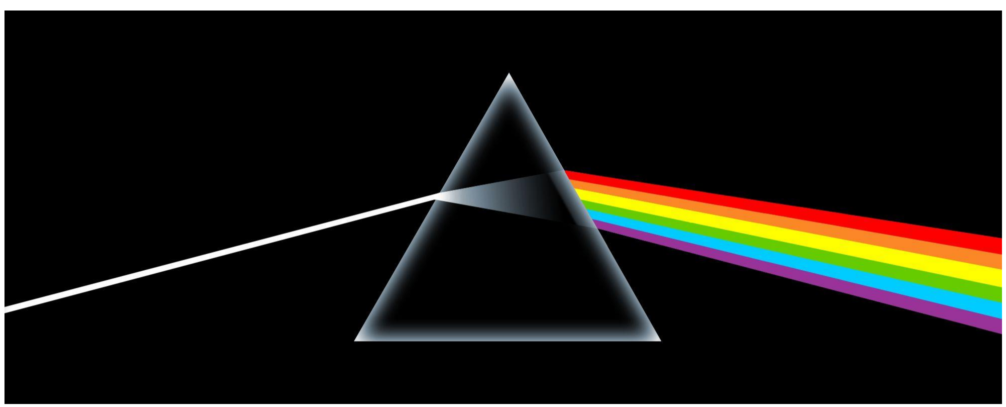
- The Rendering Equation
- The Phong Lighting Model
- Blinn Specular Lighting
- Polygon Shading Models

Lighting and Shading

Computer Graphics 2020

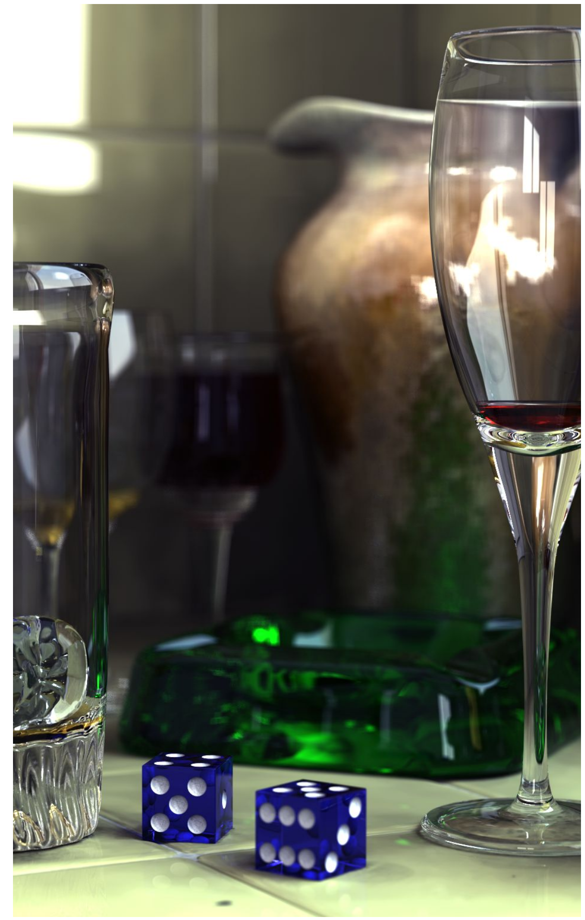
What is Light?

- Light electromagnetic radiation that can be seen by the human eye
- Before reaching our eyes, light interacts with materials in many complex processes - bouncing, refracting, being absorbed, etc.



What is Light?

- Because of the infinite complexity of light we must make approximations if we wish to simulate its effects in a virtual scene
- We get a tradeoff between realism and complexity - which directly affects rendering time



Definitions

- ***Lighting*** is the process of computing the radiance (i.e. outgoing light) from a particular 3D point
- ***Shading*** is the process of altering the color of a surface in the 3D scene, based on things like the angle to the light, the angle to the camera and material properties
- In the process of Shading we assign colors to pixels, using a program called a ***Shader***

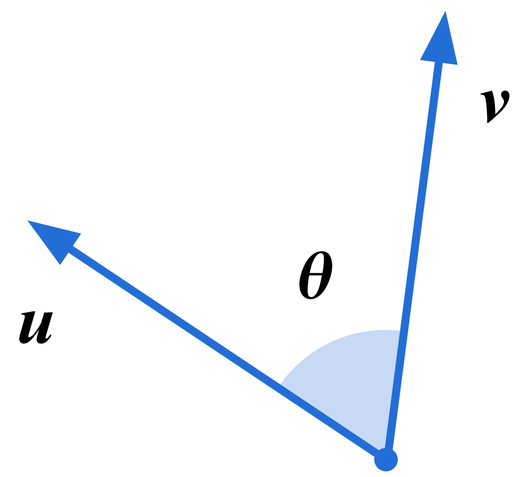
Reminder - Dot Product

- Geometrically, the dot product of two vectors $\boldsymbol{u}$, $\boldsymbol{v}$ is equal to the product of the magnitudes of the two vectors and the cosine of the angle between them:

$$\boldsymbol{u} \cdot \boldsymbol{v} = ||\boldsymbol{u}||\ ||\boldsymbol{v}||\ \cos\theta$$

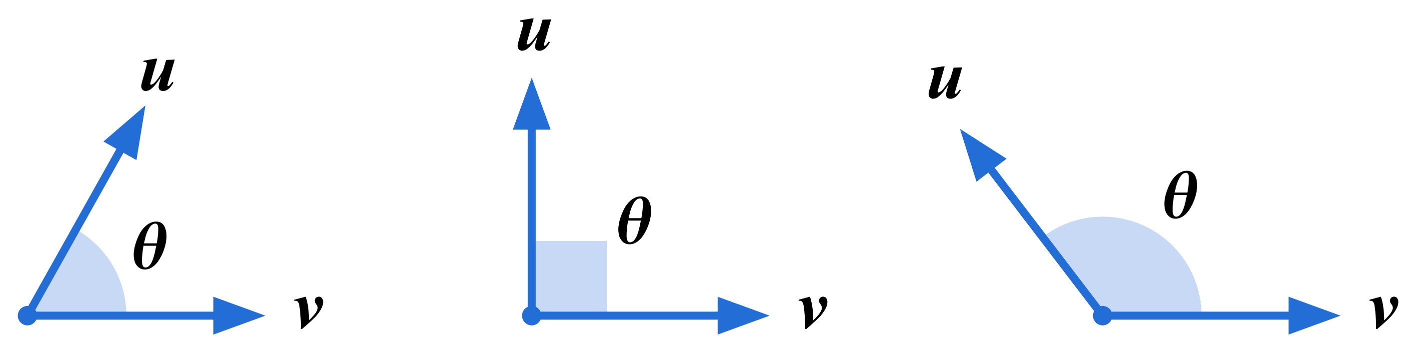
- When normalized:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \cos\theta$$



Reminder - Dot Product

- The sign of the dot product gives information about the geometric relationship of the two vectors



$\boldsymbol{u} \cdot \boldsymbol{v} > 0$ $\quad$ $\boldsymbol{u} \cdot \boldsymbol{v} = 0$ $\quad$ $\boldsymbol{u} \cdot \boldsymbol{v} < 0$

$0° \leq \boldsymbol{\theta} < 90°$ $\quad$ $\boldsymbol{\theta} = 90°$ $\quad$ $90° < \boldsymbol{\theta} \leq 180°$

The Rendering Equation

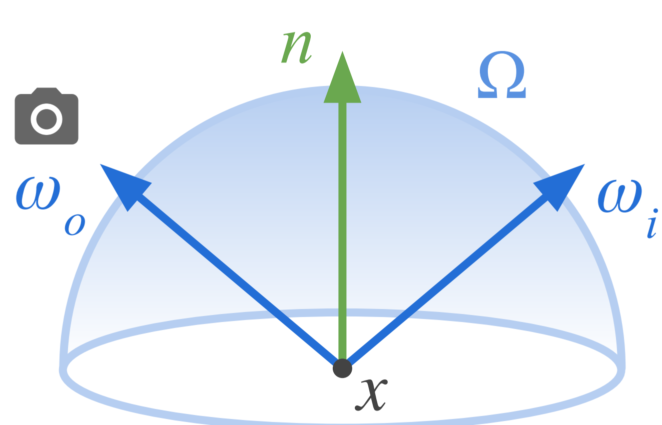
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o)(\omega_i \cdot n) L_i(x,\omega_i) d\omega_i$$

- Describes the total amount of outgoing light from point x to a view direction ω_o
- The physical basis for the rendering equation is the law of conservation of energy
- Although the equation is very general, it does not capture every aspect of light reflection

The Rendering Equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o)(\omega_i \cdot n) L_i(x, \omega_i) d\omega_i$$

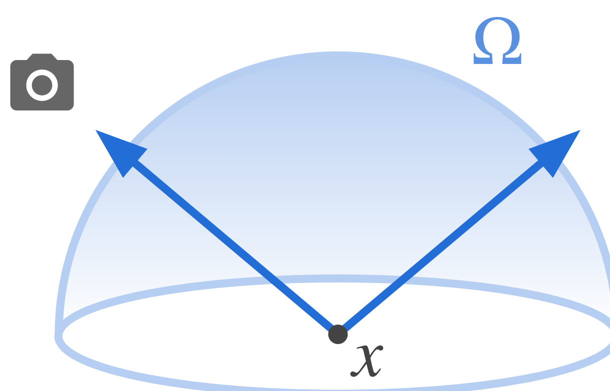
- ω_o - Outgoing direction from x to the camera
- $\omega_i \in \Omega$ - Incoming light direction to point x
- n - Surface normal at point x



The Rendering Equation

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o)(\omega_i \cdot n)L_i(x,\omega_i)d\omega_i$$

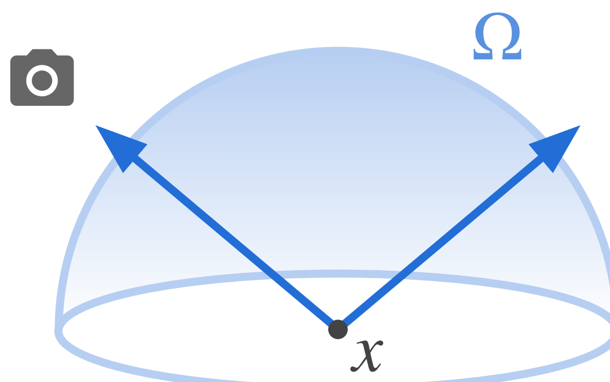
- $L_e(x, \omega_o)$ is the light emitted from x to direction ω_o
- For most surfaces, $L_e(x, \omega_o) = 0$ because they do not emit but only reflect light
- Light sources will have $L_e(x, \omega_o) > 0$



The Rendering Equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o)(\omega_i \cdot n) L_i(x, \omega_i) d\omega_i$$

- The amount of reflected light is given by integrating over Ω, the hemisphere of all possible incoming light directions to x



The Rendering Equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o)(\omega_i \cdot n) L_i(x, \omega_i) d\omega_i$$

- $f_r(x, \omega_i, \omega_o)$ is known as ***BRDF*** - the *Bidirectional Reflectance Distribution Function*
- The BRDF is *bidirectional*, meaning:

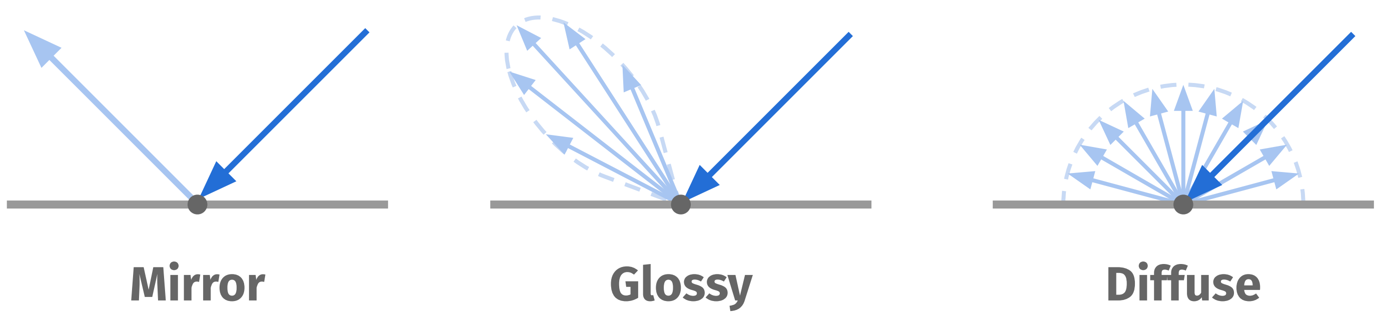
$$f_r(x, \omega_i, \omega_o) = f_r(x, \omega_o, \omega_i)$$

- Because of this we can use backwards ray-tracing (camera to light) and get correct results

The Rendering Equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o)(\omega_i \cdot n) L_i(x, \omega_i) d\omega_i$$

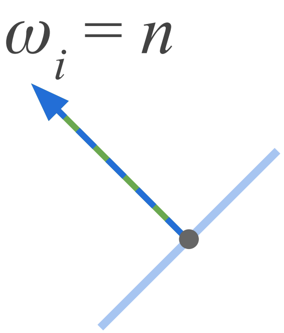
- *Reflectance distribution* means the function describes what proportion of light coming from ω_i is reflected to direction ω_o



The Rendering Equation

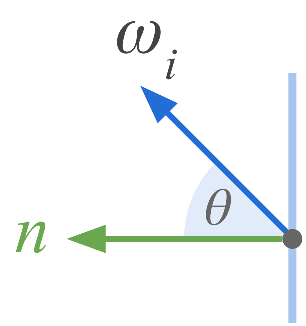
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega f_r(x,\omega_i,\omega_o)(\omega_i \cdot n) L_i(x,\omega_i) d\omega_i$$

- $(\omega_i \cdot n) = \cos\theta$ is the weakening factor of due to the incident angle θ



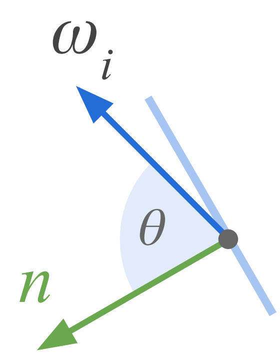
Full reflectance

$\cos\theta = 1$



Partial reflectance

$\cos\theta \approx 0.7$



Small reflectance

$\cos\theta \approx 0.1$

The Rendering Equation

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o)(\omega_i \cdot n) L_i(x, \omega_i) d\omega_i$$

- $L_i(x, \omega_i)$ - Light coming to x from direction ω_i
- To calculate this we must find the collision point in the incoming direction ω_i and use the rendering equation recursively...

The Rendering Equation

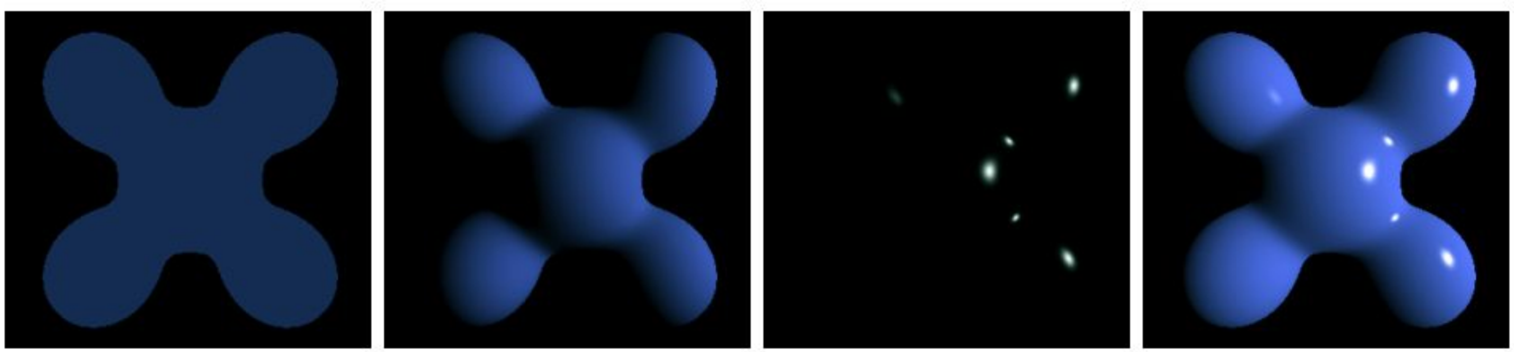
- Correct shading requires a global calculation involving all objects and light sources - very heavy and time consuming
- Solving the rendering equation for a given scene is the primary challenge in realistic rendering
- We can “cheat” and approximate the effects of the Rendering Equation to achieve real-time rendering that looks good!

The Phong Lighting Model

- A simple model that can be computed rapidly to approximate the effects of a light source on a surface
- Developed by Bui Tuong Phong at the University of Utah, published in 1975
- Considered radical at the time of introduction, but has since become the baseline shading method for many applications

The Phong Lighting Model

- In order to simplify the problem, we seperate the effects of a light on a surface into three primary lighting components:



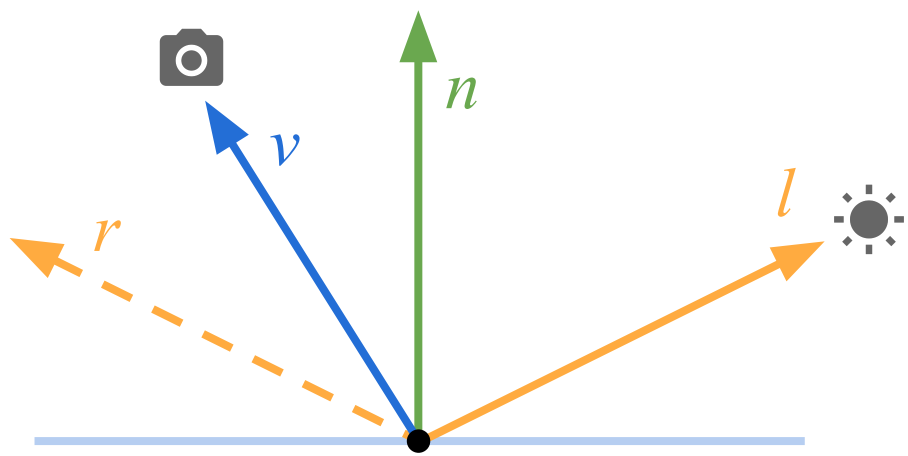
Ambient + Diffuse + Specular = Phong Lighting

The Phong Lighting Model

- In order to calculate these components we need to use 4 vectors for each point on a surface:

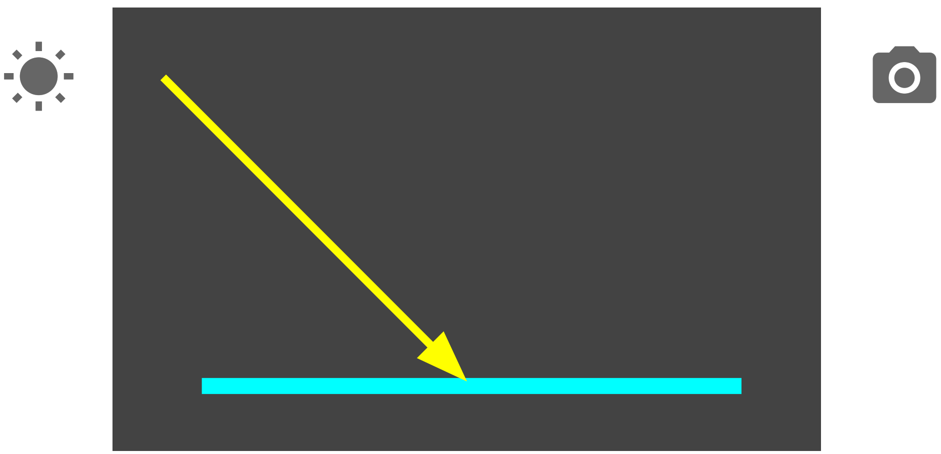
l - Light direction n - Surface normal at point

r - Reflection vector v - Viewpoint direction



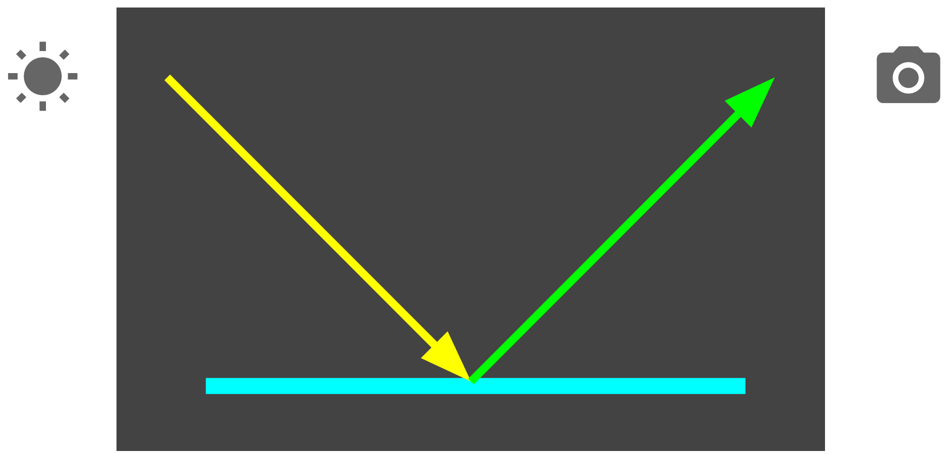
Material and Light Interaction

- The final reflected color is affected both by the color of the light and the color of the material
- What color will we see if we shine a yellow light on a cyan surface?



Material and Light Interaction

- Yellow light = $(1, 1, 0)$
- Cyan surface = $(0, 1, 1)$
- Light $*$ Surface = $(0, 1, 0)$



Material and Light Interaction

- To account for this, we define a color coefficient for each lighting component of the material and light source
- Each coefficient vector contains red, green and blue values

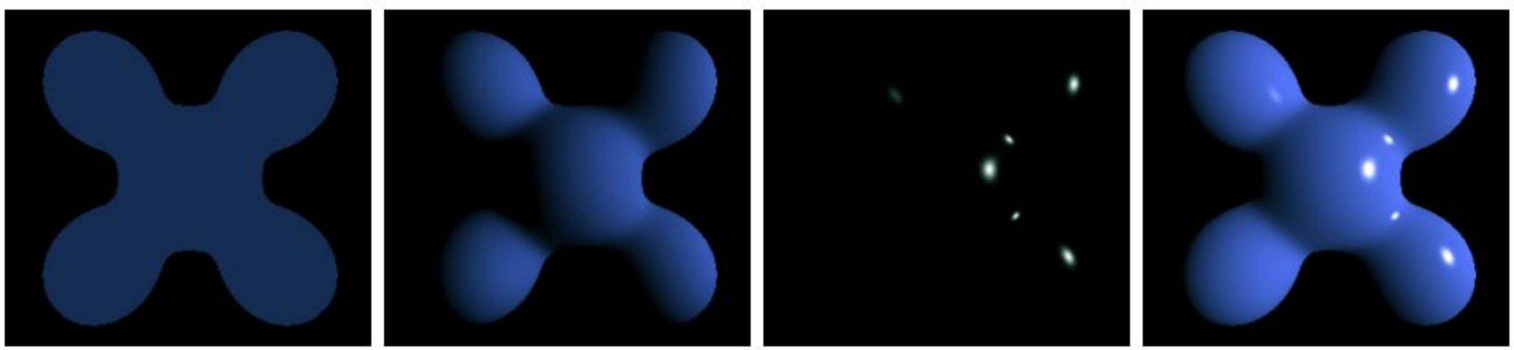
Material and Light Interaction

- Ambient - light color l_a, material color c_a
- Diffuse - light color l_d, material color c_d
- Specular - light color l_s, material color c_s

- For example, the material specular coefficient vector $c_s = [c_{s.r}, c_{s.g}, c_{s.b}]$ contains red blue and green coefficients
- Note that this is **not** physically accurate at all!

The Phong Lighting Model

- We will now see how to calculate each of the lighting components using the vectors and coefficients we defined



Ambient + Diffuse + Specular = Phong Lighting

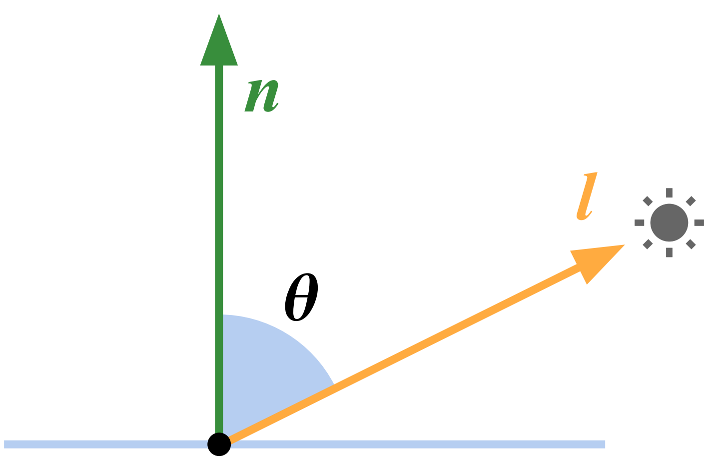
Ambient Reflectance

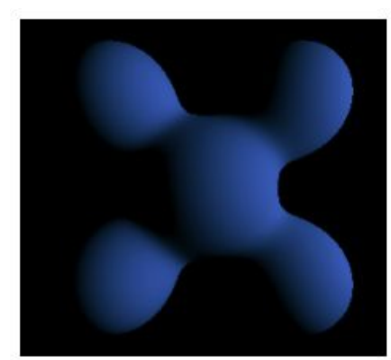
- In real life, a little bit of light almost always reaches even the darkest of shadows
- Ambient light is a result of light by bouncing around many times in the environment
- A very crude simulation of this "Global Illumination" is just to set a constant base color:

$$color_a = c_a * l_a$$

Diffuse / Lambertian Reflectance

- Represents a “matte” surface, on which light is scattered equally in all directions
- θ is the angle of surface in relation to the light
- According to *Lambert's cosine law*, the amount of reflected light is proportional to $\cos\theta$

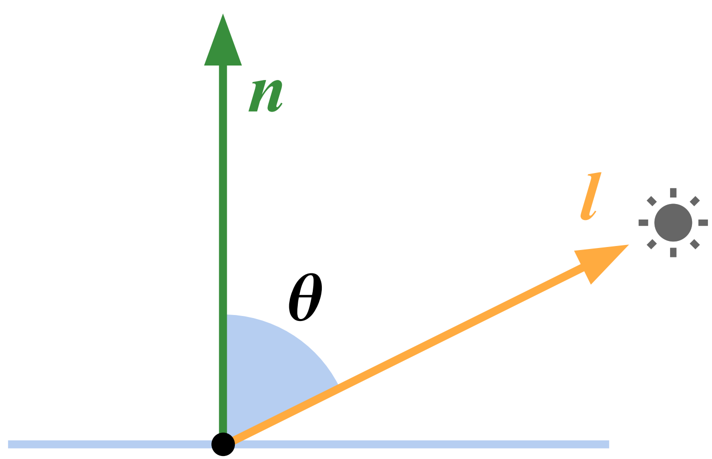


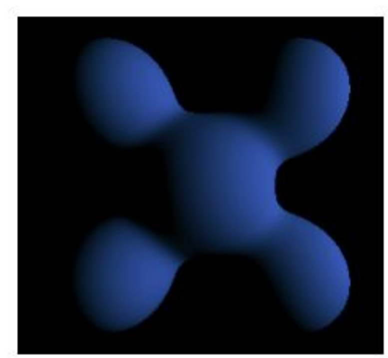


Diffuse / Lambertian Reflectance

- Same reflectance regardless of view direction!
- Assuming normalized vectors, $\cos\theta = l \cdot n$
- No such thing as negative light! So finally we get:

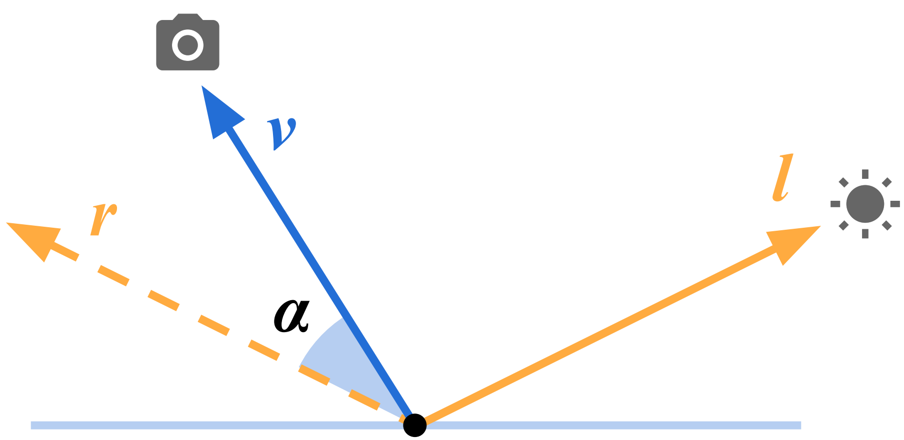
$$color_d = \max(l \cdot n, 0) * c_d * l_d$$

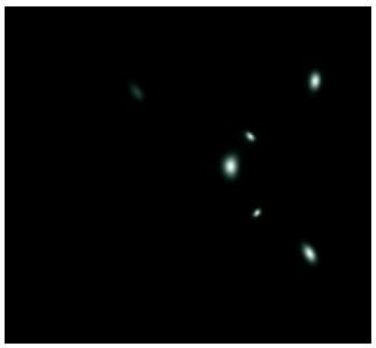




Specular Reflectance

- Represents specular highlights on the object, dependant on viewpoint direction v
- If the light reflection direction r is towards the viewpoint v, the area should appear brighter
- determined by the angle α

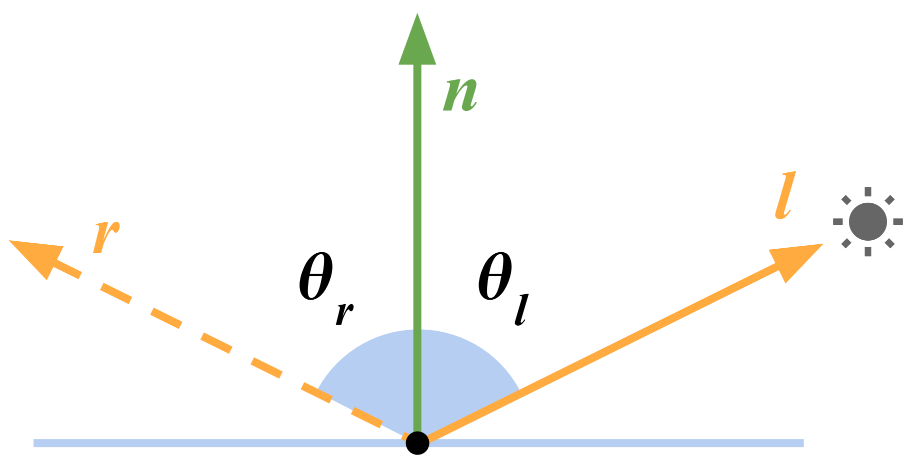




Specular Reflectance

- How do we find the reflection direction $\boldsymbol{r}$?
- From *the Law of Reflection*, we know that $\theta_r = \theta_l$

$$\Rightarrow \boldsymbol{r} \cdot \boldsymbol{n} = \cos \theta_r = \cos \theta_l = \boldsymbol{l} \cdot \boldsymbol{n}$$

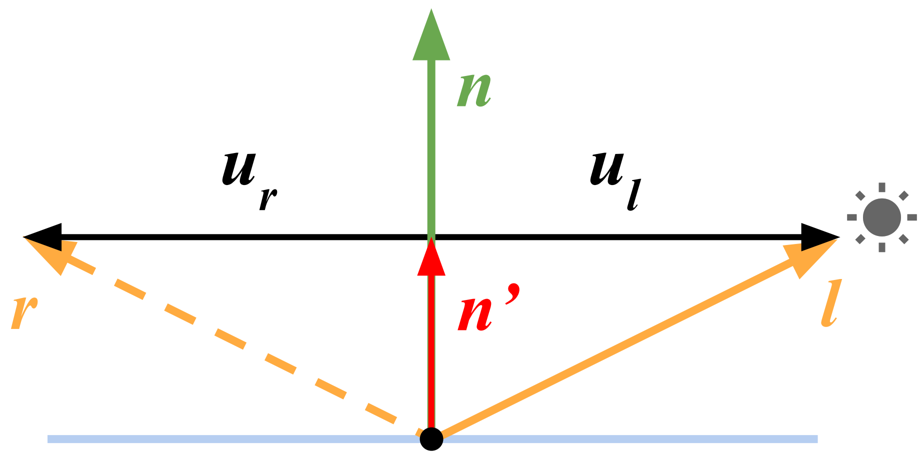


Specular Reflectance

- Take a look at $\boldsymbol{u}_r$ and $\boldsymbol{u}_l$ and note $\boldsymbol{u}_r = -\boldsymbol{u}_l$

$$\boldsymbol{u}_r = \boldsymbol{r} - \boldsymbol{n}' \quad \boldsymbol{u}_l = \boldsymbol{l} - \boldsymbol{n}' \qquad \boldsymbol{n}' = (\boldsymbol{l} \cdot \boldsymbol{n})\boldsymbol{n} = (\boldsymbol{r} \cdot \boldsymbol{n})\boldsymbol{n}$$

$$\Rightarrow \boldsymbol{r} - (\boldsymbol{l} \cdot \boldsymbol{n})\boldsymbol{n} = -(\boldsymbol{l} - (\boldsymbol{l} \cdot \boldsymbol{n})\boldsymbol{n}) = -\boldsymbol{l} + (\boldsymbol{l} \cdot \boldsymbol{n})\boldsymbol{n}$$

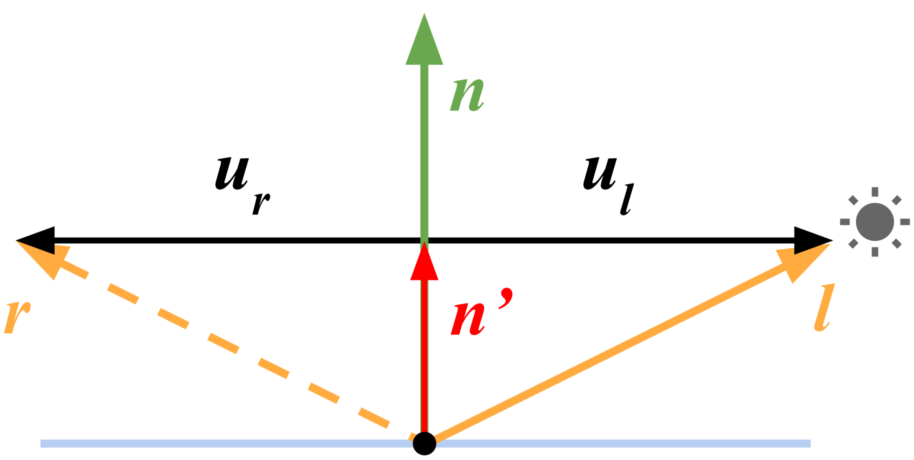


Specular Reflectance

- Rearrange and we get:

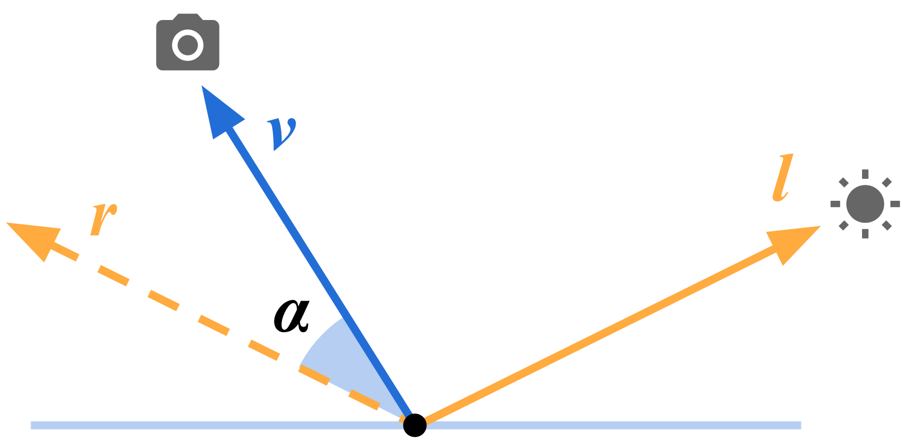
$$r - (l \cdot n)n = -l + (l \cdot n)n \quad \Rightarrow$$

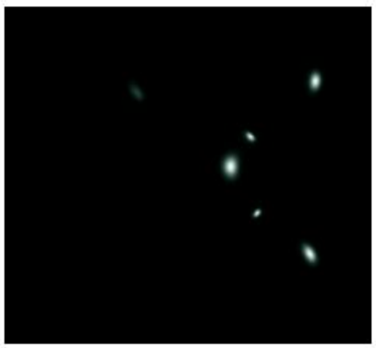
$$r = 2(l \cdot n)n - l$$



Specular Reflectance

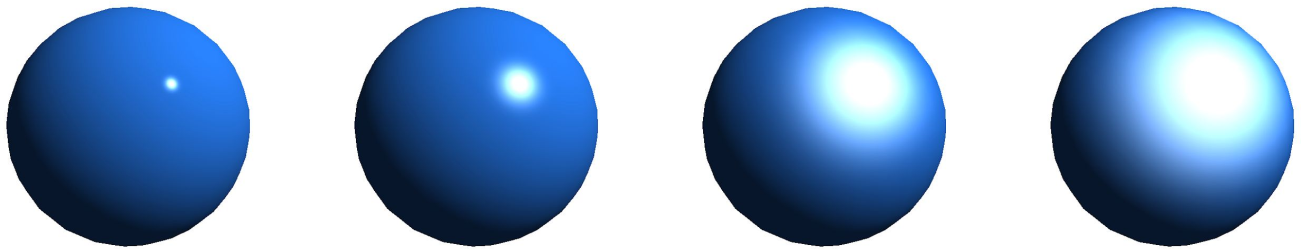
- So, assuming normalized vectors: $\cos \alpha = r \cdot v$
- To control the size of the highlight we raise to the power of the shininess coefficient σ
- So finally we get: $color_s = \max(r \cdot v, 0)^{\sigma} * c_s * l_s$





Shininess Coefficient

- The shininess coefficient describes the breadth of the angle of specular reflection
- As $\boldsymbol{\sigma}$ becomes smaller, the angle of reflection and so the highlight become larger



$\sigma = 1000$ $\sigma = 100$ $\sigma = 10$ $\sigma = 5$

Full Phong Lighting Model

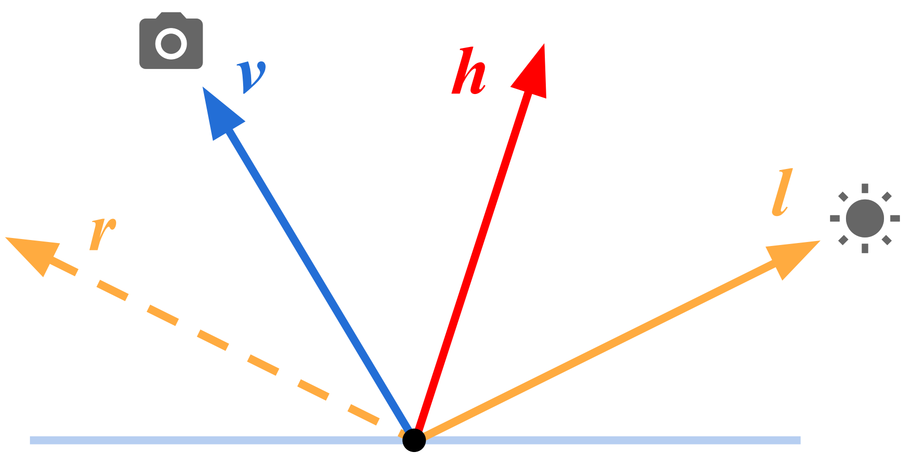
- Finally we combine all three color components to get our color:

$$final\_color = color_d + color_s + color_a =$$

$$\max(l \cdot n, 0) * c_d * l_d + \max(r \cdot v, 0)^{\sigma} * c_s * l_s + c_a * l_a$$

Blinn-Phong Specular Reflectance

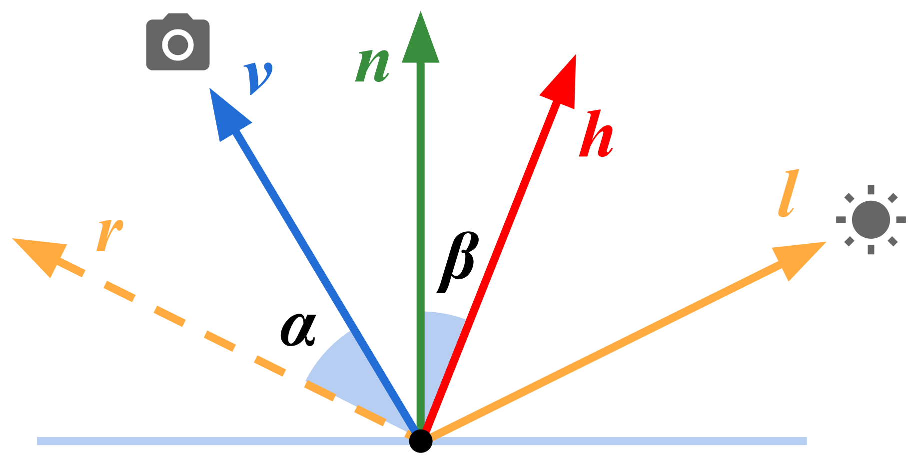
- Blinn suggested a different approach for calculating specular reflectance
- Recall Phong specular: $\boldsymbol{color_s = \max(r \cdot v, 0)^\sigma}$
- Blinn uses the *halfway vector* $\boldsymbol{h}$ instead of $\boldsymbol{r}$ and $\boldsymbol{v}$



Blinn-Phong Specular Reflectance

- The halfway vector ***h*** is between the light direction ***l*** and the viewing direction ***v***
- ***α*** is the original angle we used, take a look at ***β***

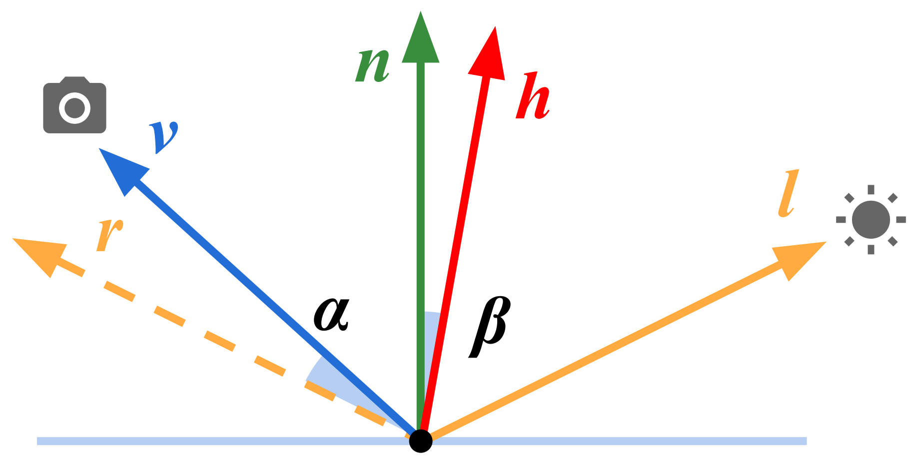
$$h = \frac{(l+v)/2}{|(l+v)/2|}$$



Blinn-Phong Specular Reflectance

- As $\boldsymbol{v}$ approaches $\boldsymbol{r}$, $\boldsymbol{\alpha}$ shrinks, and we can see that $\boldsymbol{\beta}$ also shrinks:

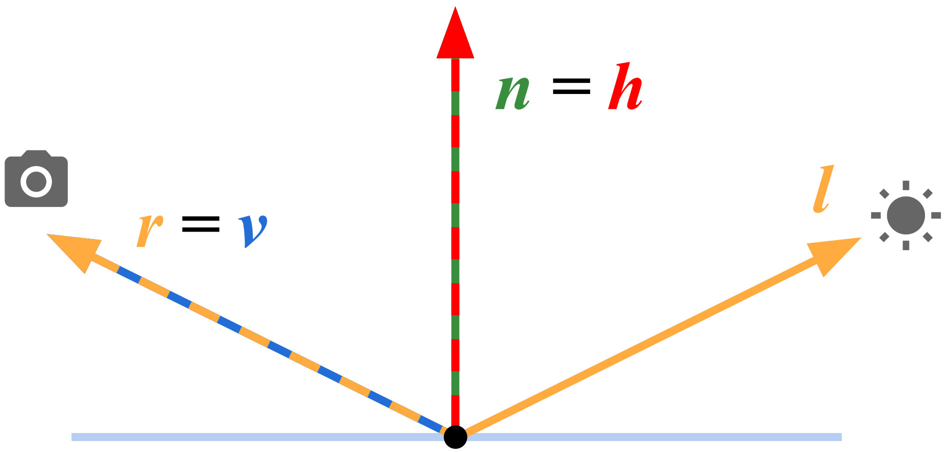
$$h = \frac{(l+v)/2}{|(l+v)/2|}$$



Blinn-Phong Specular Reflectance

- The halfway vector will coincide with the surface normal if the reflected light coincides with the viewer direction

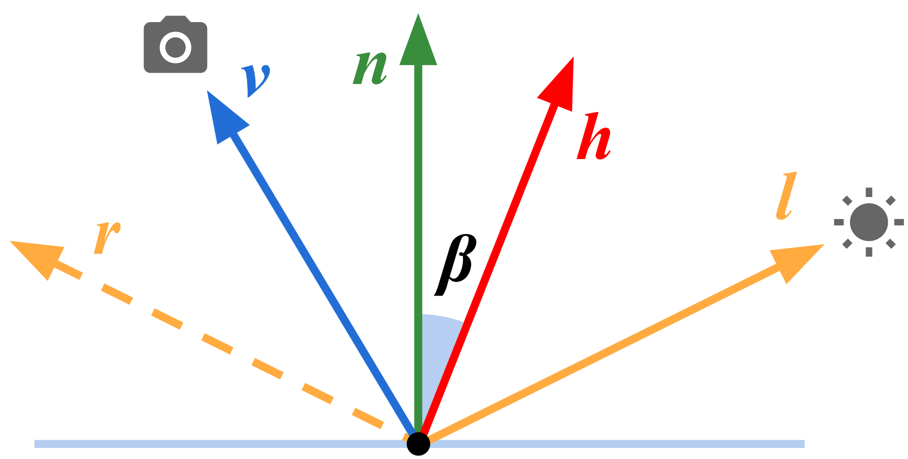
$$h = \frac{(l+v)/2}{|(l+v)/2|}$$



Blinn-Phong Specular Reflectance

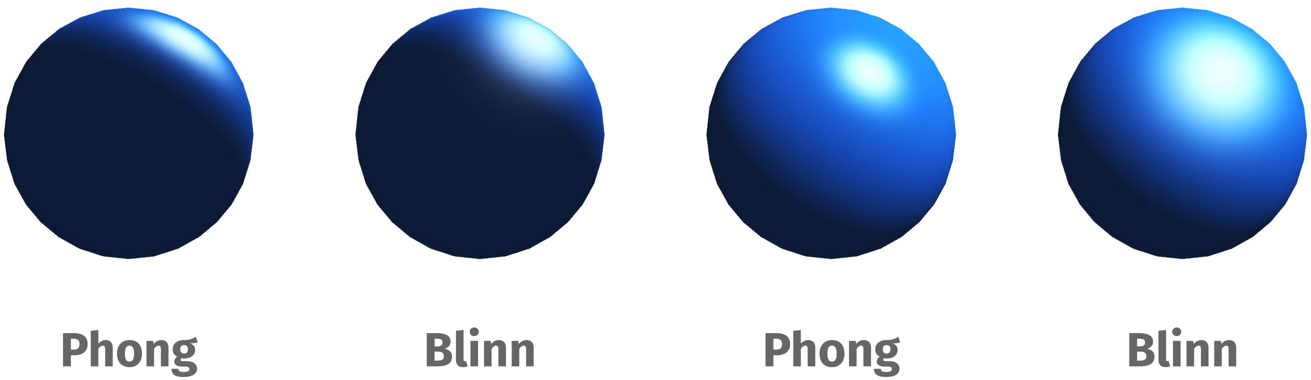
- As before, we get $\cos\beta = \boldsymbol{n} \cdot \boldsymbol{h}$
- The Blinn-Phong specular component:

$$color_s = \max(\boldsymbol{n} \cdot \boldsymbol{h}, 0)^{\sigma} * c_s * l_s$$



Blinn-Phong Specular Reflectance

- Blinn highlights will look a bit different
- For the same values of the shininess coefficient σ we get larger highlights
- For example, when $\sigma = 10$:



Blinn-Phong Specular Reflectance

- Blinn is also more efficient in a <u>particular</u> case
- If we use an orthographic camera, $\boldsymbol{v}$ remains constant at each surface point
- If we use a directional light, $\boldsymbol{l}$ remains constant at each surface point
- We get a constant $\boldsymbol{h}$ that we can compute once for the whole scene!

Blinn-Phong Lighting Model

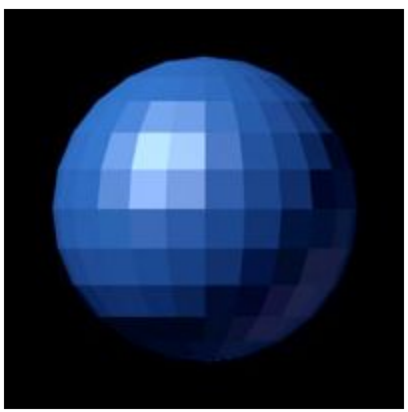
- The full Blinn-Phong lighting model:

$$final\_color = color_d + color_s + color_a =$$

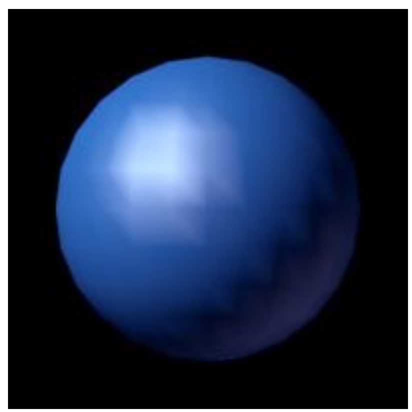
$$\max(l \cdot n, 0) * c_d * l_d + \max(n \cdot h, 0)^{\sigma} * c_s * l_s + c_a * l_a$$

Polygon Shading

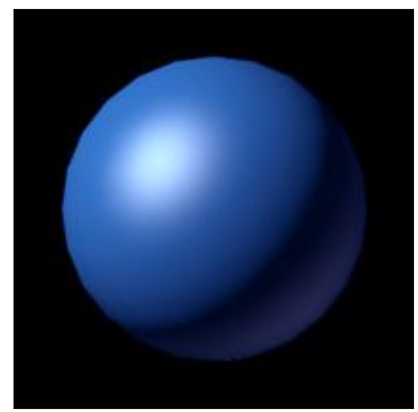
- We have learned how to calculate the color of a single point on a surface, the next step is to color a whole mesh
- There are three main shading models:



Flat Shading



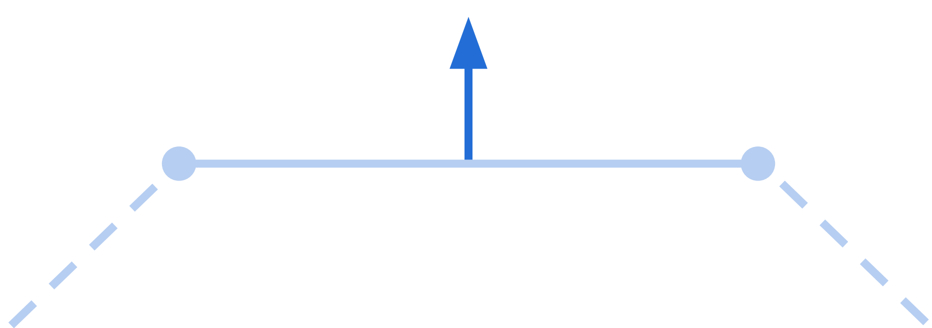
Gouraud Shading

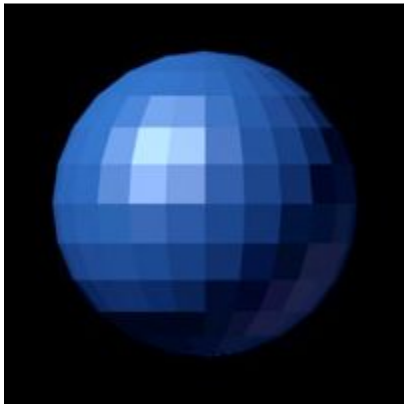


Phong Shading

Flat Shading - Per Polygon

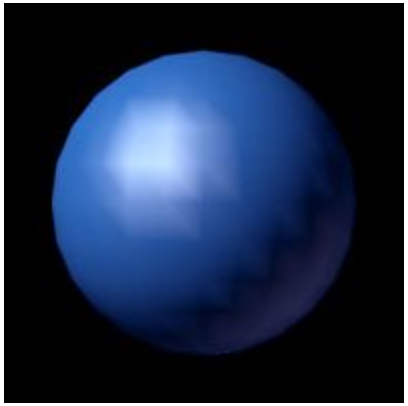
- Evaluate the lighting model once per polygon, use resulting color for all of its pixels
- the most simple and efficient way to specify color for an object
- Results in a faceted appearance





Gouraud Shading - Per Vertex

- Evaluate the lighting model once per vertex, and interpolate the resulting <u>colors</u> for each pixel in the polygon
- Results in a smoother appearance
- Bad with specular reflections!



Phong Shading - Per Pixel

- Evaluate the lighting model once per pixel, by interpolating between the normals of the polygon vertices
- Results in a smooth appearance, perfectly shades a sphere