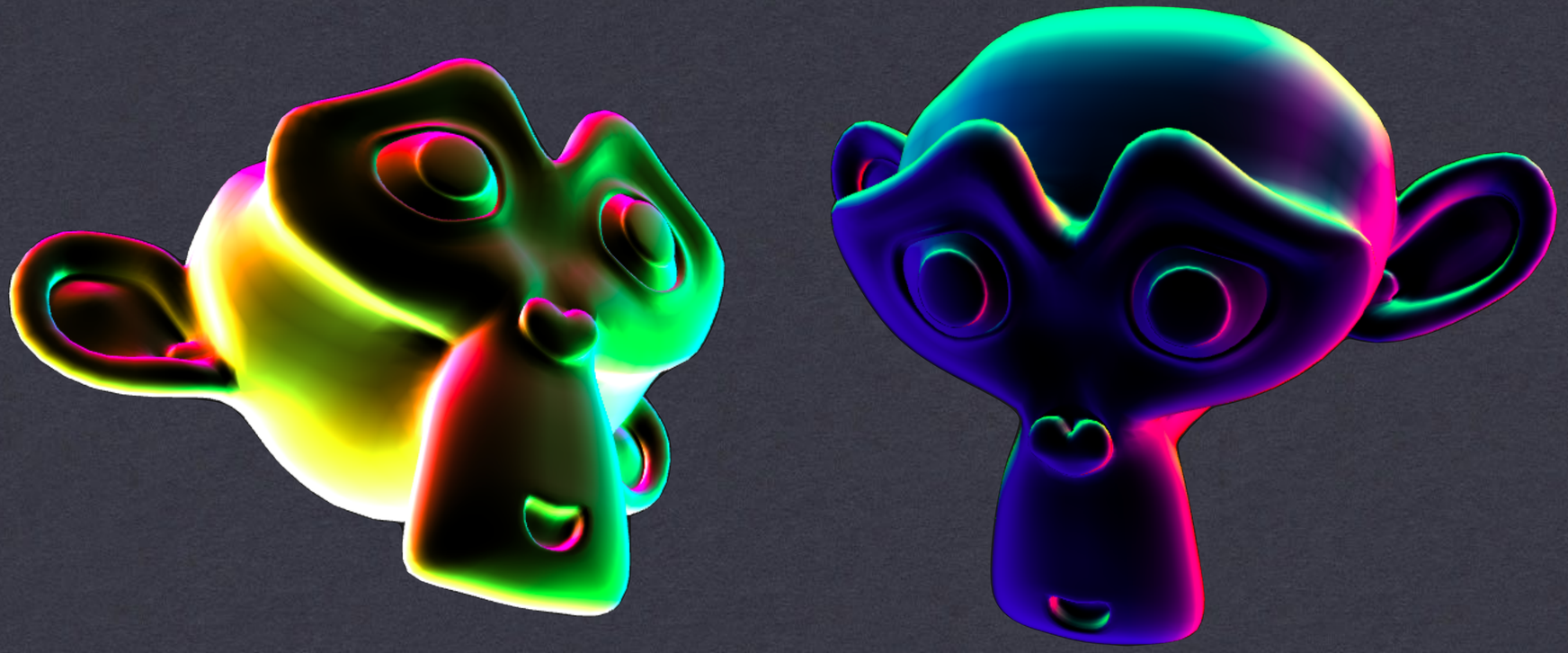


LIGHTING AND SHADING

MUSL2361 - VR ADVENTURE

香 港 資 優 教 育 學 苑
The Hong Kong Academy for Gifted Education

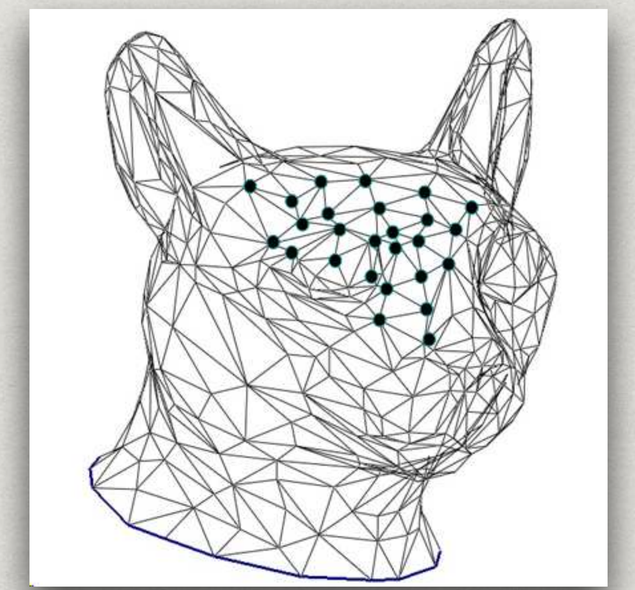
3D GRAPHICS PIPELINE

OpenGL and Shaders

- **OpenGL** began as an API that implemented a **very specific sequence of operations** used for doing 3D computer graphics.
  - Like those THREE.js functions
- Modern OpenGL has become feasible to fully **control certain portions of the graphics computation** by using **shaders**.
- Much of the 3D is now done by Shaders.
- OpenGL is more about organizing the data.

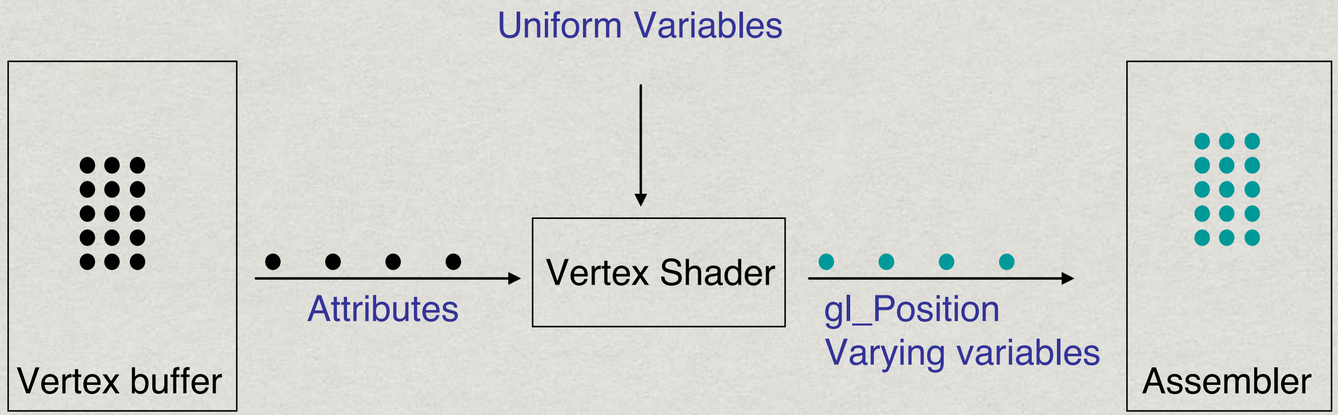
Representation of 3D Objects

- Basic representation of object starts with a **collection of triangles**
- Each triangle is formed by three vertices.
- Each vertex, obviously carries the **(x, y, z) location values**.
- May carry other **vertex attributes** including color, normal vector and material information.



- Transmitting the vertex data from the central processing unit (CPU) to the graphics hardware (GPU) is expensive.
- Thus, it is done **as infrequent as** possible.
- When we specify certain **API calls**, the vertex is transferred and stored in a **vertex buffer**.

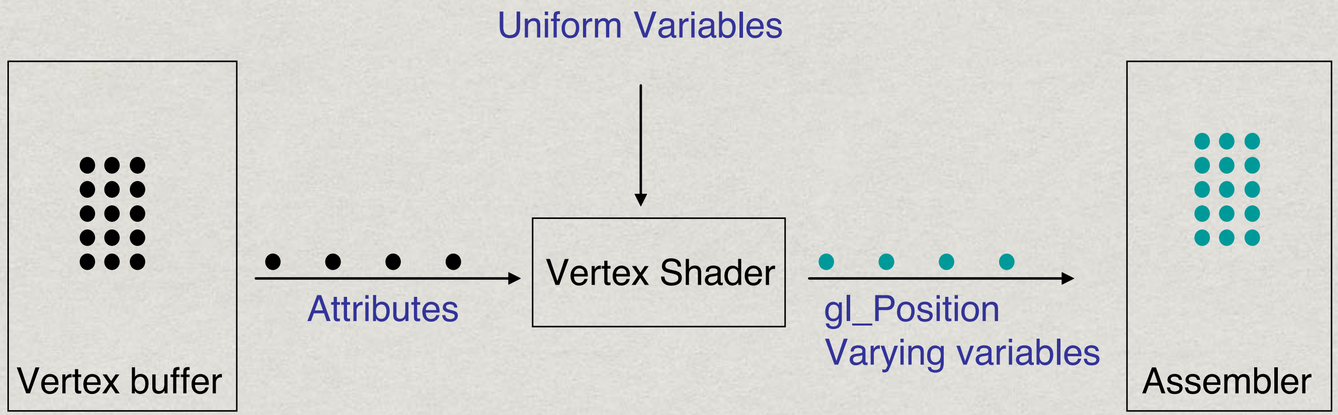
scene.add(earth)



Vertex Shader

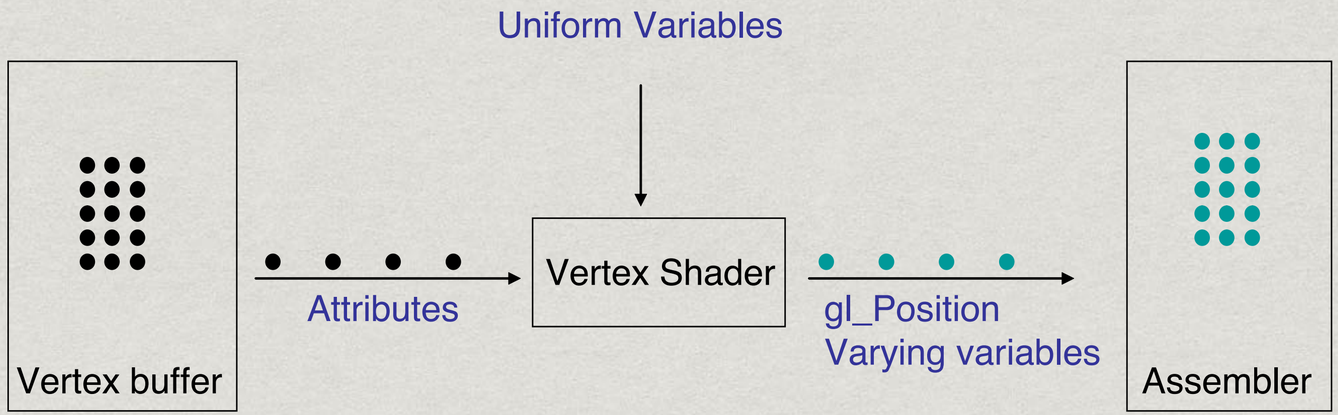
renderer.render()

- We can also issue **draw call** and each vertex (all its *attributes*) will be processed independently by a (programmable) **vertex shader**.
- The shader can also access things called **uniform variables**, which you can set between draw calls and **not per vertex**.

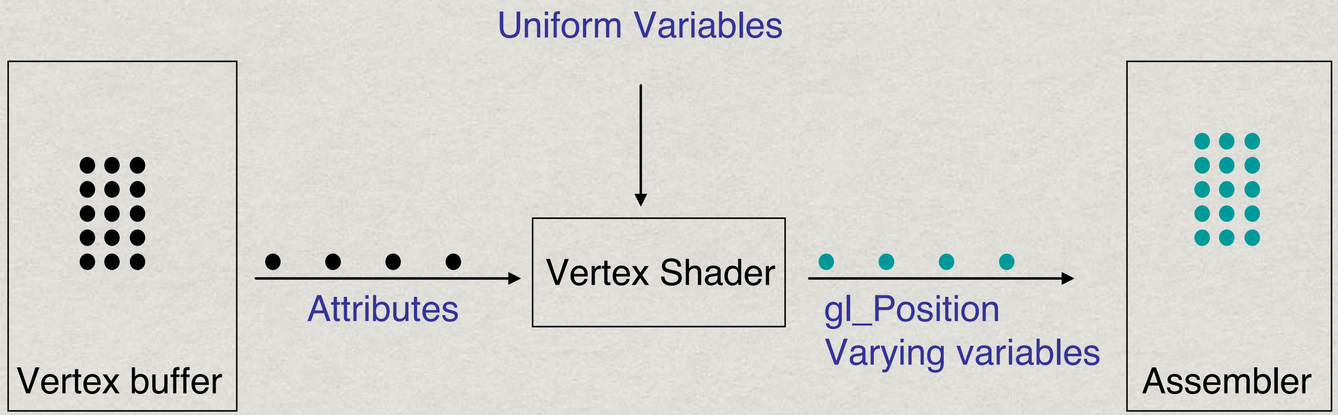


Vertex Transformations

- The most typical use of vertex shader is to **determine the final position of the vertices** on the screen.
- For example, a uniform variable can be used to **describe a virtual camera** that maps the 3D vertices to the actual 2D screen.

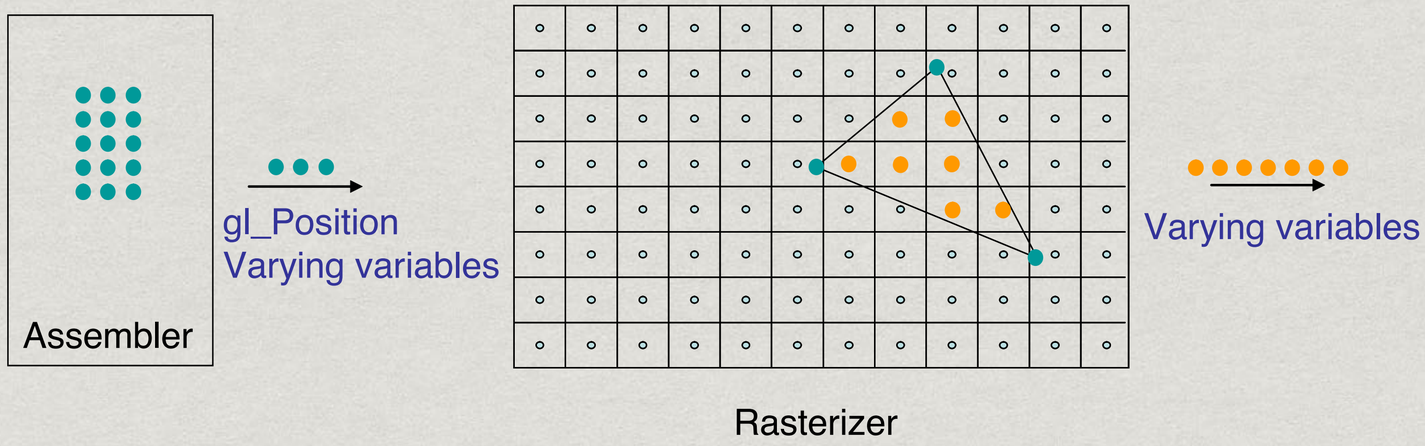


- The **new vertex locations** will be stored in output variable **gl_Position**.
- Other processed **vertex attributes** (like color) will be stored as **Varying variables**.
- Assembler will group vertices back to triangles.



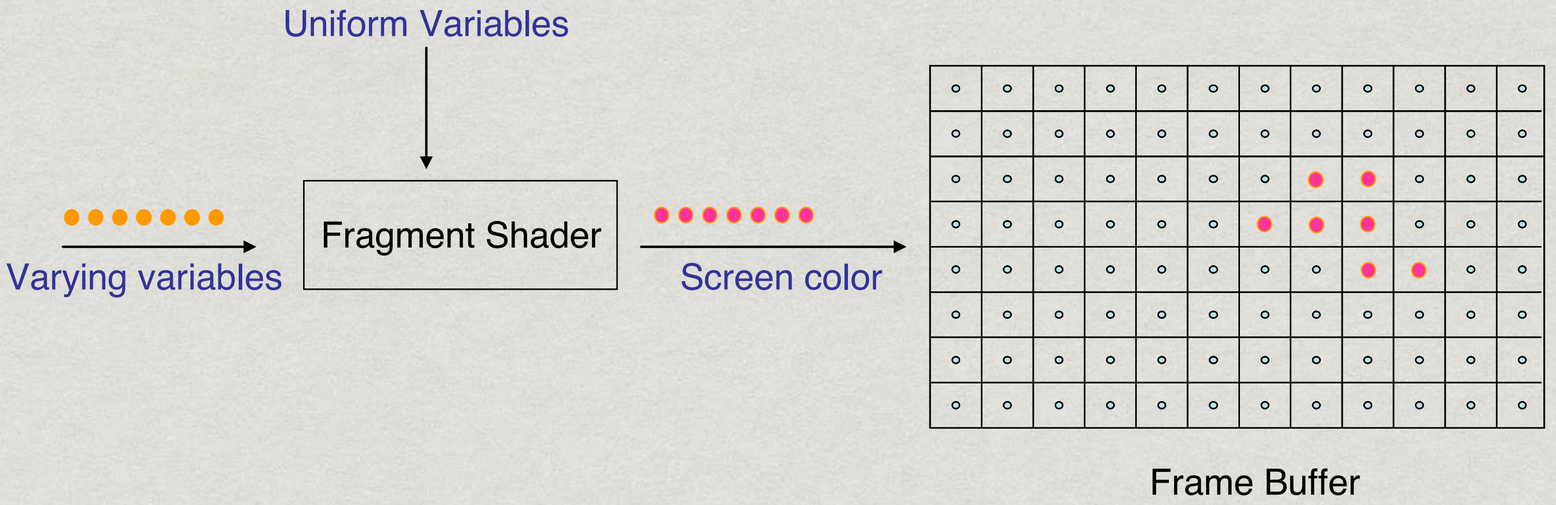
Rasterization

- Each triangle is drawn on screen.
- The **rasterizer** will compute the **interpolated values** for each **varying variable** (such as color) on each **pixel** inside the triangles.



Fragment Shader

- Finally, for each pixel, the interpolated values will pass through the **fragment shader** (also programmable).
- The job of the fragment shader is to **determine the drawn color of each pixel** based on information from the **varying variables** and **uniform variables**.



Lighting and Shading

- Uniform variable can represent the **position and color of light source**.
- In 3D graphics, we typically determine a pixel's color by computing **a few equations that simulate the way that light reflects** off of the surface of some material.

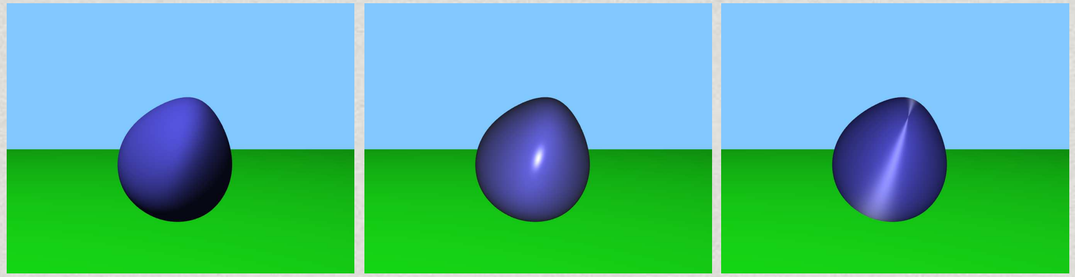


Figure 1.5: By changing our fragment shader we can simulate light reflecting off of different kinds of materials.

Texture Mapping

- Fragment shader can also **read a texture image** (as a uniform variable) and glue the image onto each triangle.

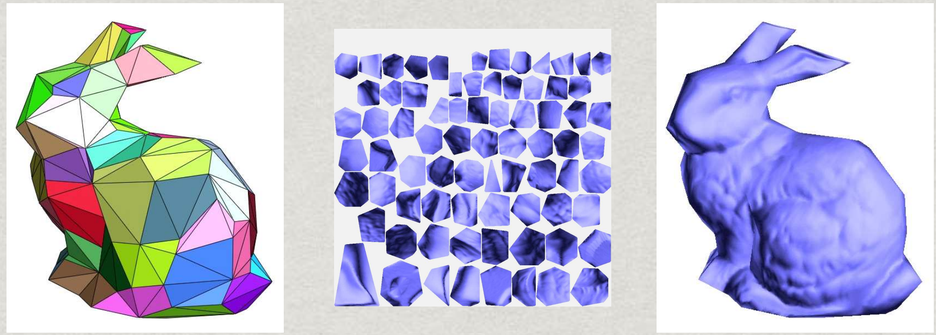


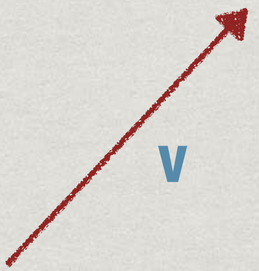
Figure 1.6: Texture mapping. Left: A simple geometric object described by a small number of triangles. Middle: An auxiliary image called a texture. Right: Parts of the texture are glued onto each triangle giving a more complicated appearance. From [65], ©ACM.

Merging

- The last step.
- When **z-buffering** is enabled, the frame-buffer is only updated when the newly processed point is **closer than** the existing point.
- Other process, named **Alpha blending**, will mix old and new values together to produce **transparent effect**.

MORE ON VECTORS

Vectors



- Physical definition: **a vector is a quantity with two attributes**
  - **Direction** and **Magnitude**
- To find the **magnitude** of a vector?
  - **Pythagorean Theorem**.
- How about **direction**?

$$|\vec{v}| = \sqrt{2^2 + 1^2 + (-2)^2} = 3$$

How about Direction?

- We can express the direction of a vector with respect to the coordinate system.
- However, we are more interested in the **angle between two vectors**.
- The angle could be obtained via the **Dot Product** between these two vectors.

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

Dot Product and Angle Between

- Given two vectors **u** and **v**
- We calculate their **dot product**.
- Also the **magnitude of each vector**.
- The **angle** between the vectors could therefore be obtained.

$$\vec{u} = \begin{pmatrix} -2 \\ 2 \\ -1 \end{pmatrix}, \quad \vec{v} = \begin{pmatrix} 2 \\ -1 \\ -2 \end{pmatrix}$$

$$\vec{u} \cdot \vec{v} = -2 \times 2 + 2 \times -1 + -1 \times -2 = -4$$

$$|\vec{u}| = |\vec{v}| = 3$$

$$\cos\theta = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|} = \frac{-4}{3 \times 3} = -\frac{4}{9}$$

$$\theta = 116.3878^{\circ}$$

Unit Vector and Vector Normalization

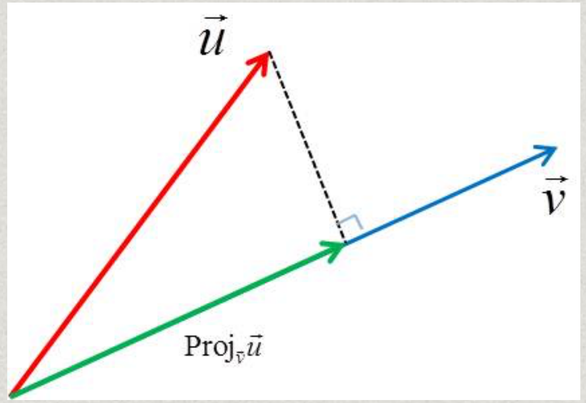
- An **unit vector** is a vector with **magnitude (or length) equals to 1**.
- It could be obtained easily with

$$\frac{\vec{u}}{|\vec{u}|}$$

- The process of turning a vector into unit length is called **normalization**.

Vector Projection

- The **vector projection** of a vector **u** on a vector **v** is the **orthogonal projection** of **u** onto a straight line parallel to **v**.
  - It could be obtained with:



$$Proj_{\vec{v}}\vec{u} = |\vec{u}|\cos\theta \times \frac{\vec{v}}{|\vec{v}|} = |\vec{u}|\frac{\vec{u}\cdot\vec{v}}{|\vec{u}||\vec{v}|} \times \frac{\vec{v}}{|\vec{v}|}$$

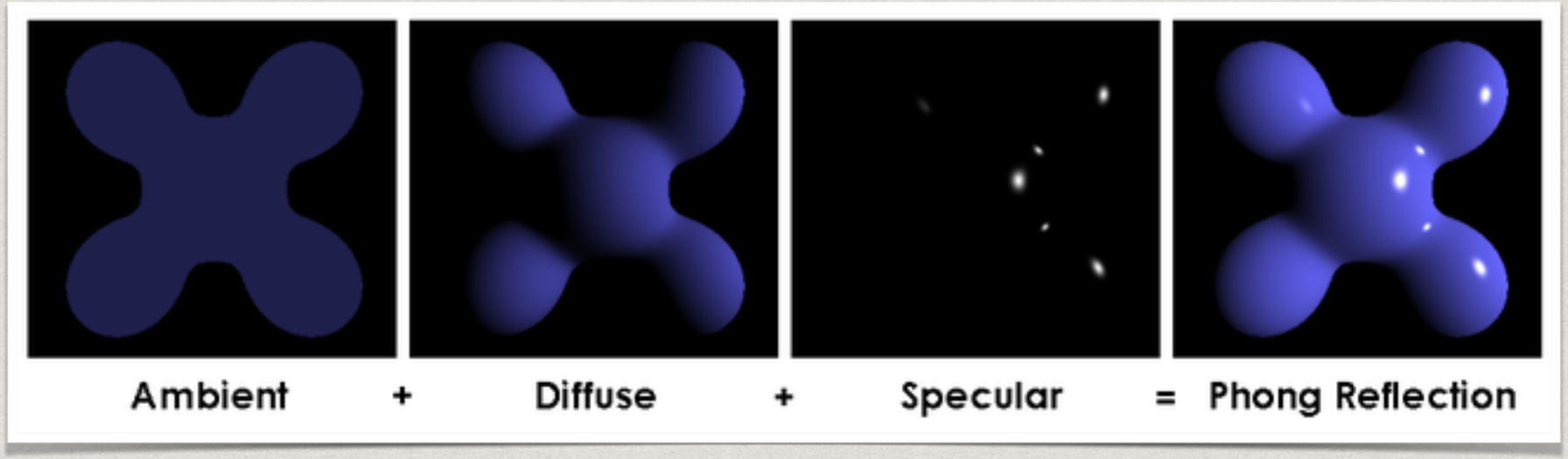
- Suppose both **u** and **v** are **unit vectors**
  - The vector projection will be

$$Proj_{\vec{v}}\vec{u} = (\vec{u}\cdot\vec{v})\vec{v}$$

LIGHTING AND SHADING

Phong Illumination Model

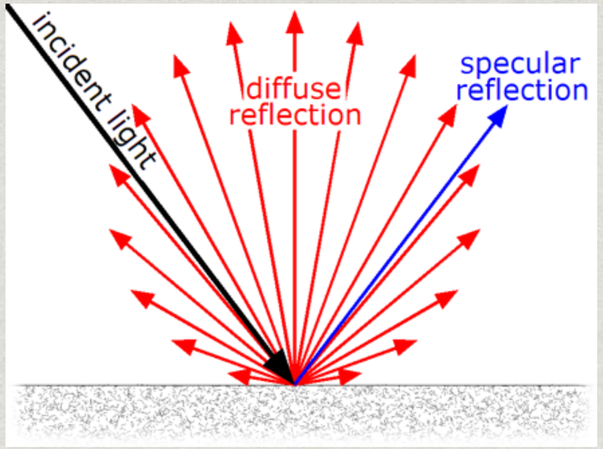
- k_A, k_D and k_S define the **color** and **reflection** of the material.

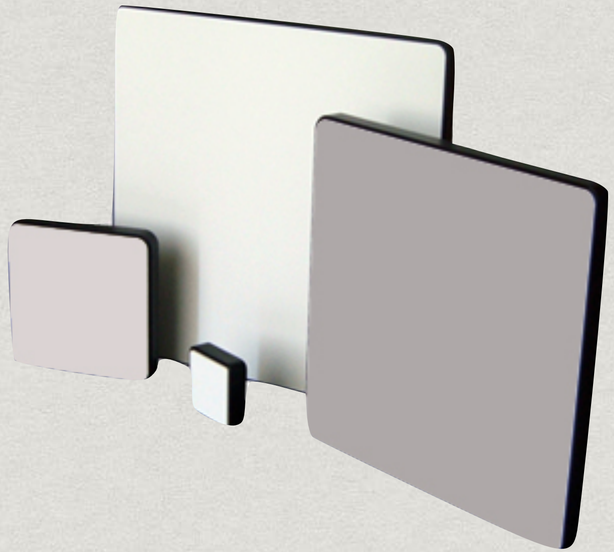


$$L_{\text{reflected}} = k_A L_{\text{ambient}} + k_D L_{\text{diffuse}} + k_S L_{\text{specular}}$$

Diffuse Reflection

- Light **scattered equally in all directions**
  - The fraction of the incident light reflected is **independent of the direction of the reflection**.
- A purely diffusive material is said to be a material that exhibits a **Lambertian** reflection.
  - Unfinished **wood**
  - **Spectralon** is a material which is designed to exhibit an almost perfect Lambertian reflectance

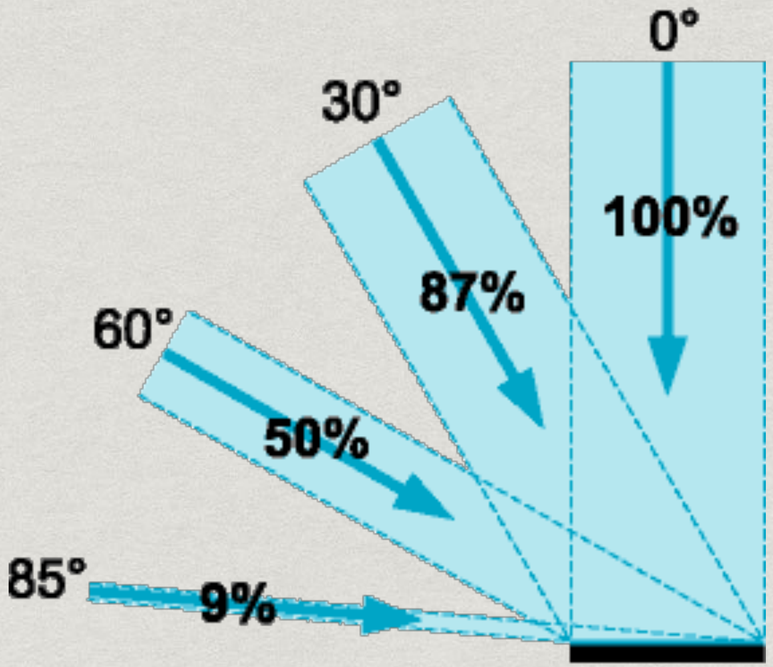




Diffuse Reflection

- The amount of light reflected only **depends on the direction of the light** incidence.
  - Reflect the most when the incident light direction is **perpendicular** to the surface.
  - Reduce as the inclination angle increases, which could be **modeled by a cosine rule**.
  - Thus, the **amount of light reflected** by a Lambertian surface is

$$L_{\text{diffuse}} = L_{\text{incident}} k_{\text{diffuse}} \cos\theta$$



Diffuse Reflection

$$L_{\text{diffuse}} = L_{\text{incident}} k_{\text{diffuse}} \cos\theta$$

- ***$L_{incident}$*** is the amount of the incident light, while ***θ* is the angle of inclination** of the incident light vector.
- We can further express cosθ as a product of **normalized vectors**.

$$\cos\theta = \boldsymbol{N} \cdot \boldsymbol{L}_{\text{incident}}$$

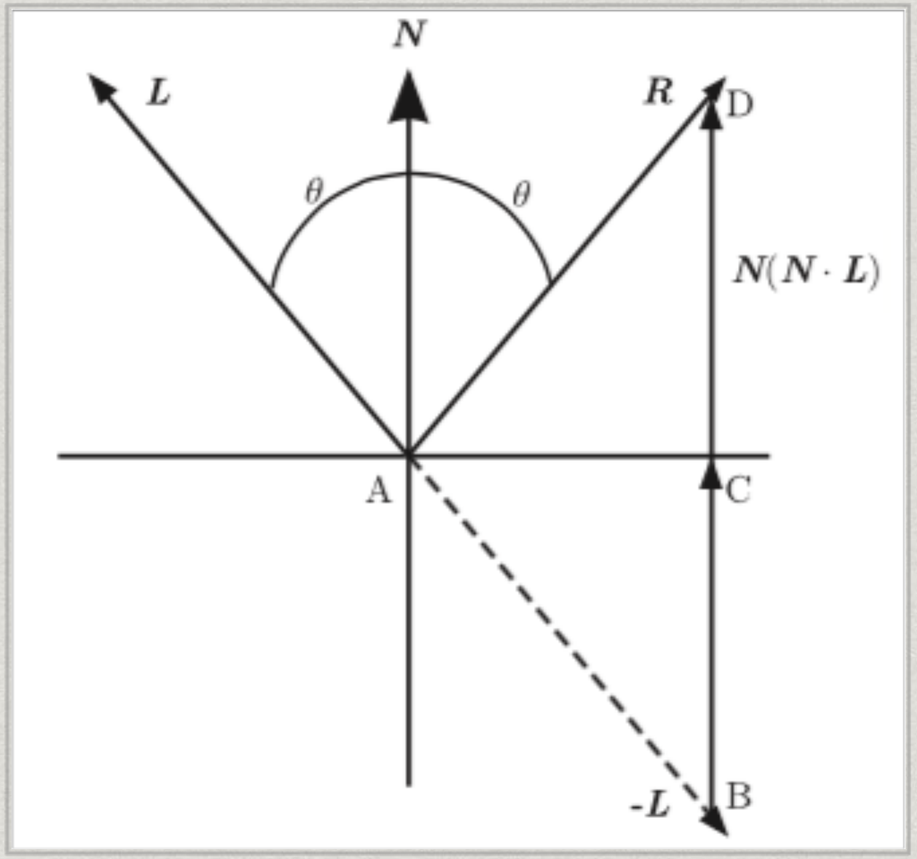
- which leads to the standard **Lambertian Reflection**

$$L_{\text{diffuse}} = L_{\text{incident}} k_{\text{diffuse}} (\boldsymbol{N} \cdot \boldsymbol{L}_{\text{incident}}).$$

Specular Reflection

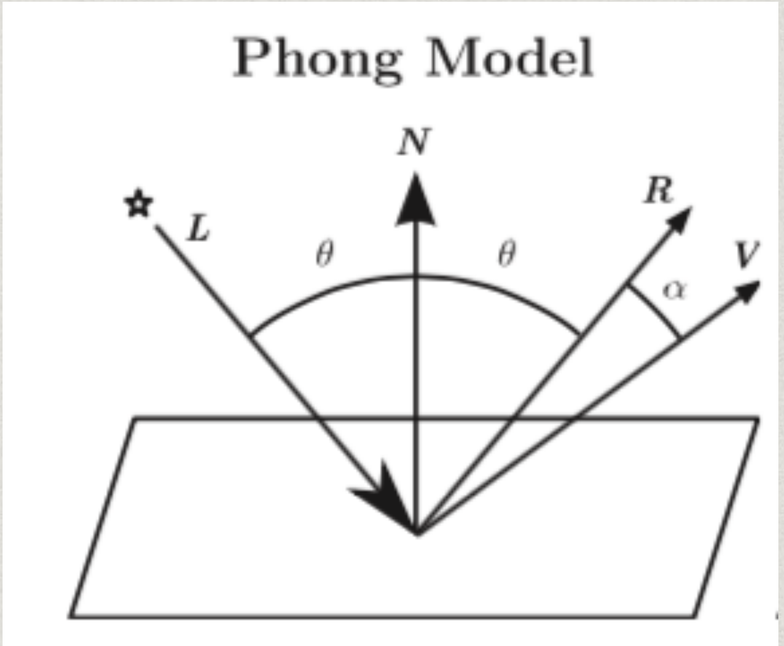
- Specular reflection is the **directionally dependent**.
- Mirror reflects the incident light with exactly **the same angle of incidence**.
- The **mirror direction** is computed by simple vector algebra.

$$\boldsymbol{R} = 2\boldsymbol{N}(\boldsymbol{N} \cdot \boldsymbol{L}) - \boldsymbol{L}$$



Modeling Specular Component

- Perfect mirrors **reflect only along the mirror reflection** of the incident light vector.
- Rough mirrors **reflect a reduced amount** along the direction close to the mirror reflection direction.
- This reduction is modeled as **a power function of the cosine function** of the reflection and view vectors.

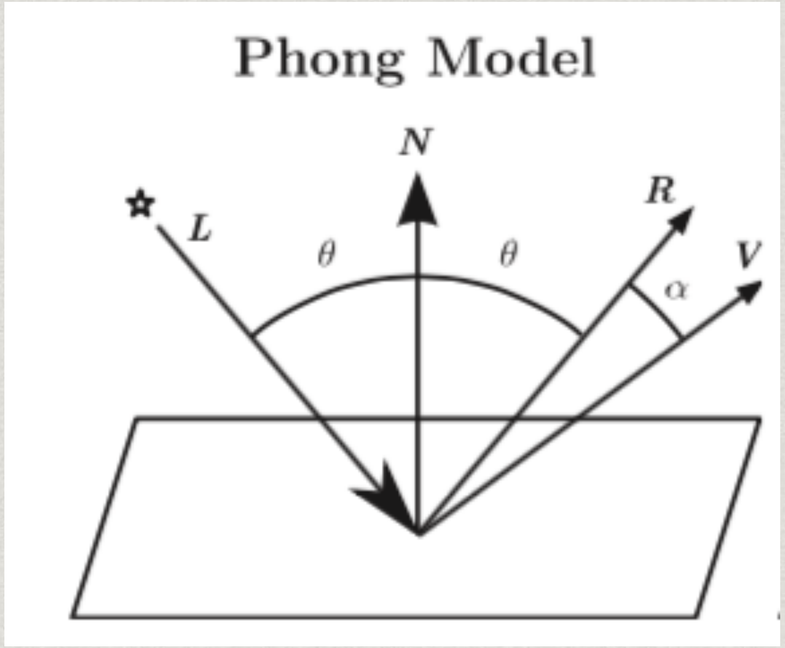


Modeling Specular Component

- So the amount of reflected light along the **view direction V** is

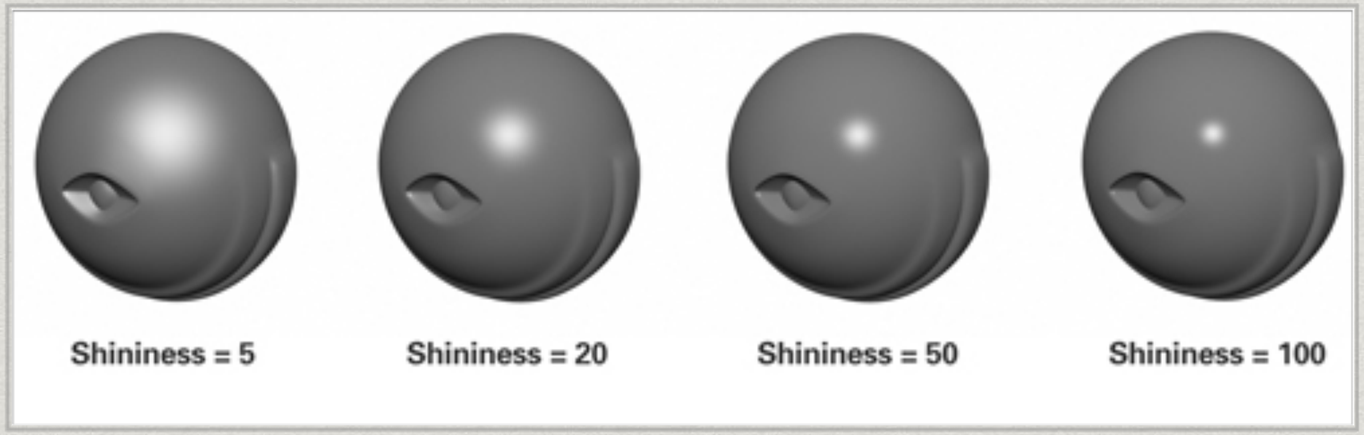
$$L_{\text{specular}} = L_{\text{incident}} \cos(\alpha)^{n_s}$$

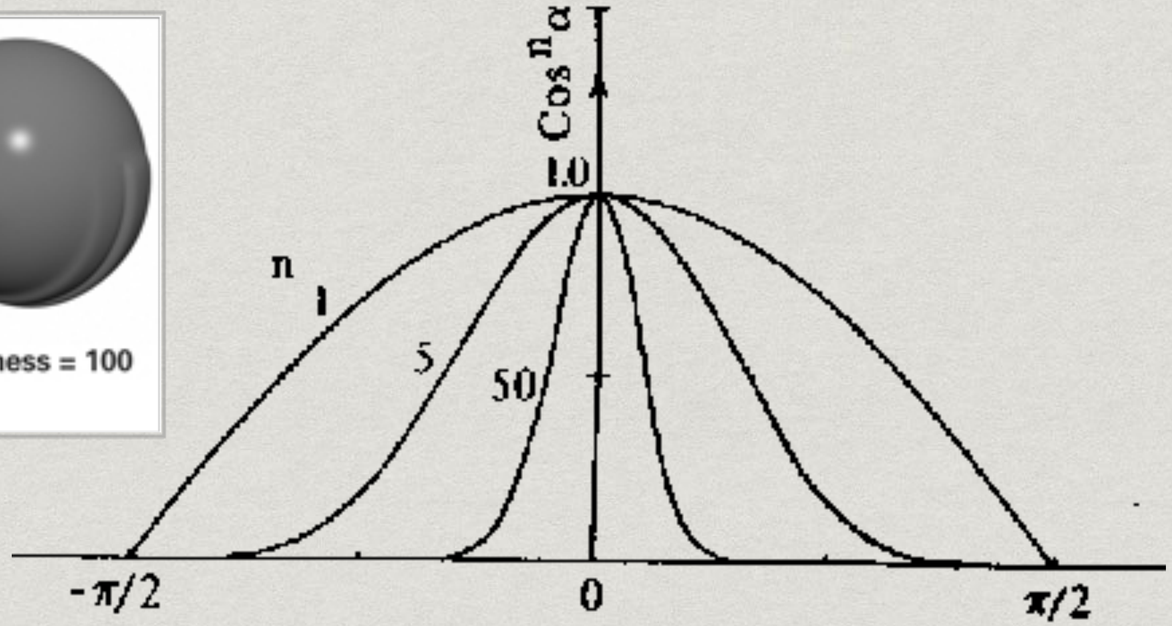
- where the **exponent $n_s > 1$** is the **shininess coefficient** of the surface.
- The larger the value of n_s, the larger is the **falloff** of the reflection.



The Shininess Coefficient

- Values of n_s between **100 and 200** correspond to **metals**
- Values between **5 and 10** give surface that look like **plastic**





Modeling Specular Component

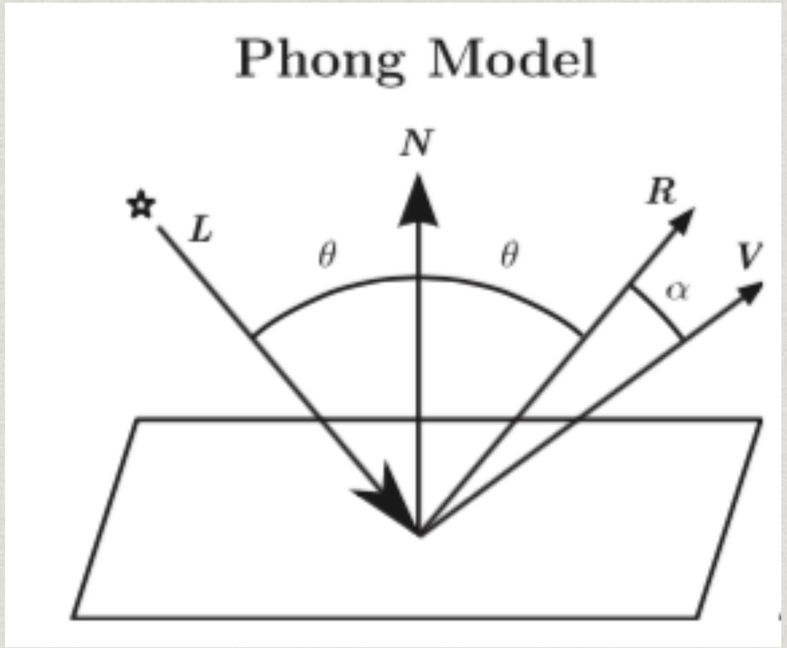
- **cos α** could be computed as

$$\cos\alpha = \boldsymbol{R} \cdot \boldsymbol{V}$$

- and **R** could be expressed as

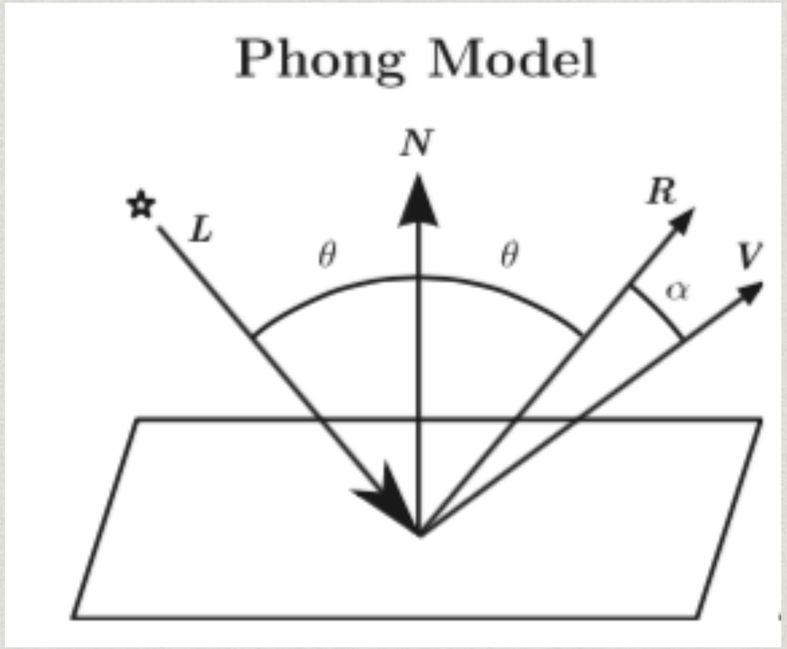
$$\boldsymbol{R} = 2\boldsymbol{N}(\boldsymbol{N} \cdot \boldsymbol{L}) - \boldsymbol{L}$$

- Thus, the specular component could be modeled with unit vectors **N, L and V**



Blinn-Phong Model

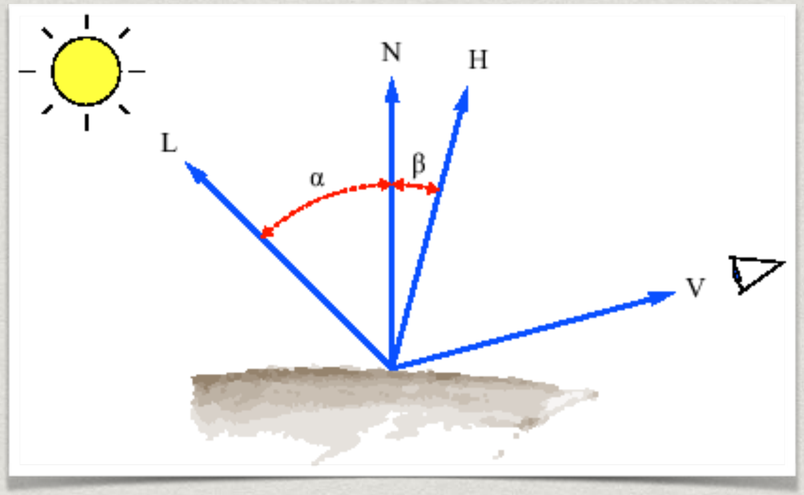
- The **specular term** in the Phong model is problematic because it **requires the calculation of a new reflection vector** and **view vector** for each vertex.
- Blinn suggested an approximation using the **halfway vector** ,which is the vector between the light vector and view vector.

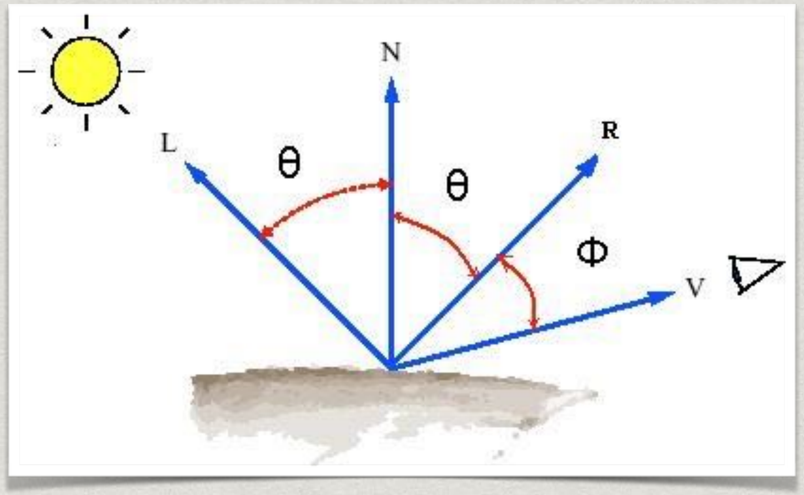


The Halfway Vector

- H is normalized vector **halfway between L and V**.

$$\boldsymbol{H} = \frac{\boldsymbol{L} + \boldsymbol{V}}{2}$$



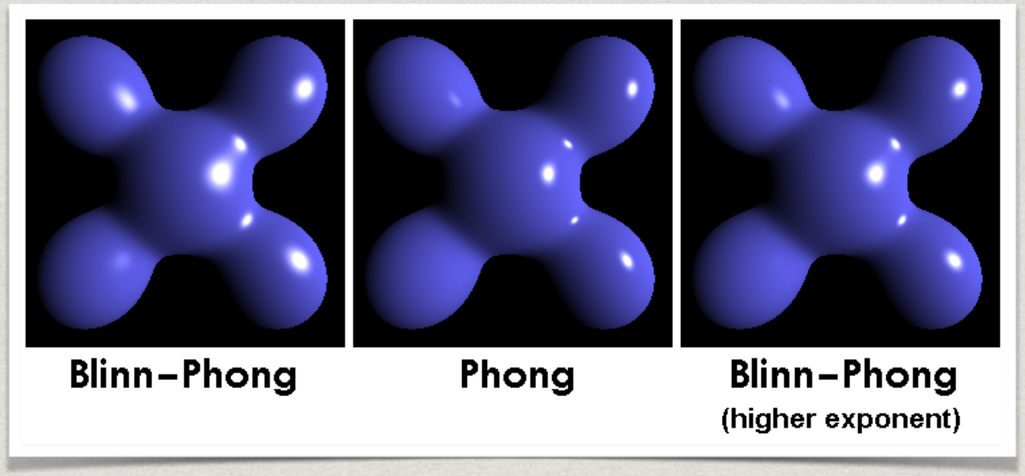


Using the Halfway Vector

- When using Blinn-Phong model, **cos α** could be computed as

$$\cos\alpha = \boldsymbol{N} \cdot \boldsymbol{H}$$

- Blinn-Phong requires **a higher shininess coefficient** to produce the same specular result with the original Phong model.



Ambient Component

- The **ambient component** is used to simulate the effect of the indirect lighting present in the scene.
- Secondary light sources, that is, the **nearby reflectors**, can make significant contributions to the illumination of surfaces.
  - Required computation can be very complex and computational expensive.
- In real-time applications, it is **approximated by an ambient term**.

$$k_A L_{\text{ambient}}$$

The Complete Model

- By putting the ambient, the diffuse and the specular component together, we obtain the **final formulation of the Phong model**:

$$L_{\text{refl}} = k_A L_{\text{ambient}} + L_{\text{incident}} \left(k_D \max(\cos\theta, 0) + k_S \max(\cos(\alpha)^{n_s}, 0) \right)$$

- The **maximum function** guarantees that the amount of reflected light due to an light source is **not less than zero**.

- Please note that this equation is **only valid for one light source**.

The Complete Model

- The effect of lighting in a scene is the **sum** of all the light contributions.
- By accumulating the **contributions of the different light sources**,

$$\begin{aligned} L_{\text{refl}} &= L_{\text{ambient}} k_{\text{ambient}} + \sum_i L_{\text{incident},i} \Big(k_D \max(\cos\theta_i, 0) \\ &+ k_S \max(\cos^{n_s} \alpha_i, 0) \Big) \end{aligned}$$

- Where ***i*** represents the *i*-th light source.