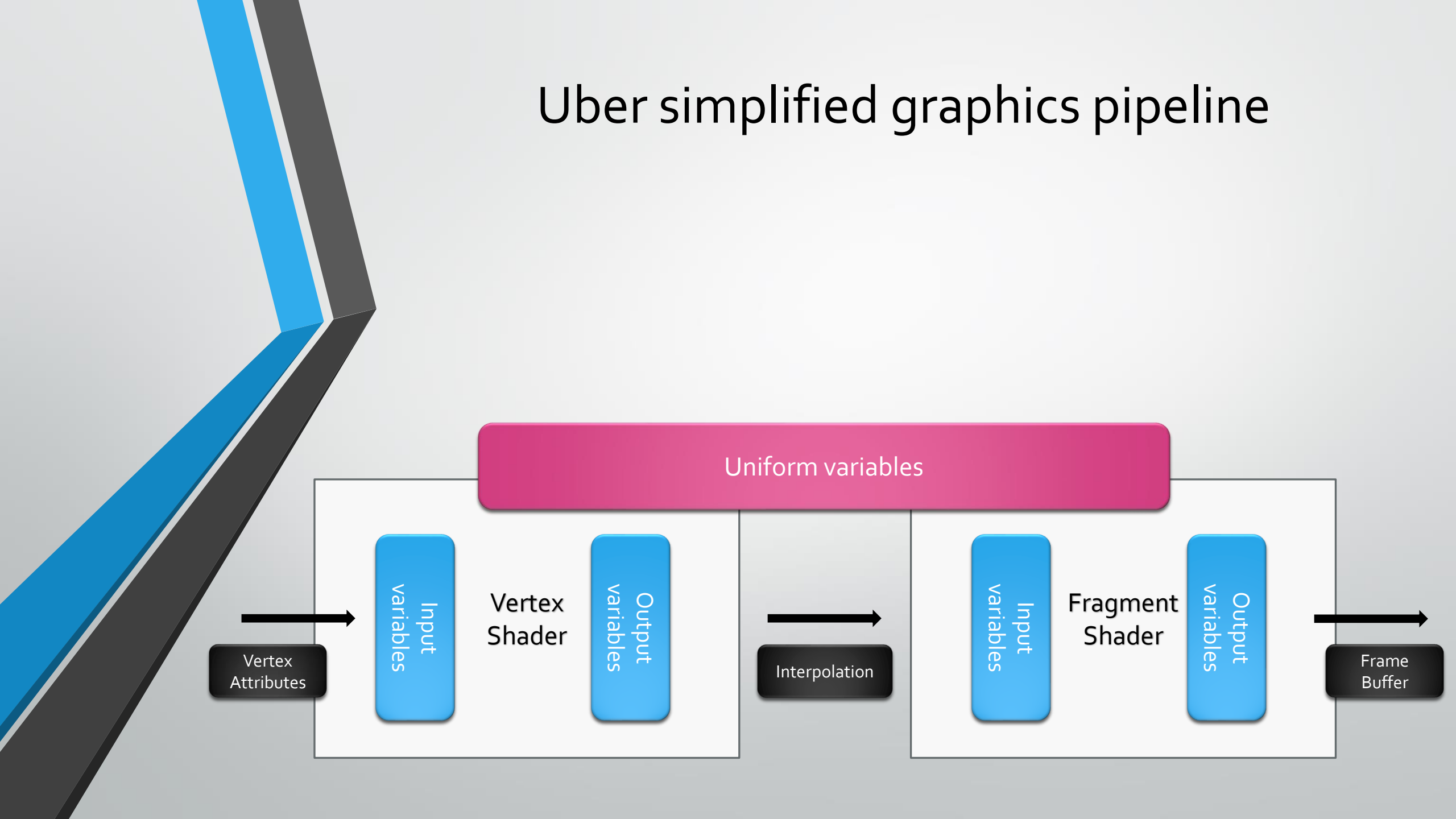
Shaders

Introduction

Uber simplified graphics pipeline



Vectors

refresher

Magnitude

- Magnitude is the length of a vector
- Use Pythagorean theorem: $c^2 = a^2 + b^2$; $c = \sqrt{(a^2 + b^2)}$;

  For our vector $c = \sqrt{(x^2 + y^2)}$;
- Magnitude of a vector - $||Vec1|| = \sqrt{(Vec1.x \cdot Vec1.x + Vec1.y \cdot Vec1.y)}$
- Vec1 = (2,3);

  $||Vec1|| = \sqrt{(2 \cdot 2 + 3 \cdot 3)} = \sqrt{(4 + 9)} = \sqrt{13} = 3.6$

Normalising

- Normalising a vector, means making something "standard", creating a unit vector -> magnitude(length) = 1
- Describes the direction of something, very useful.

$$v = vec1 / ||vec1|| ;$$

v- resulting vector

vec1 - vector

||Vec1|| - magnitude

- vec1 = (2,3); ||vec1|| = 3.6;
- v = ((2,3) / 3.6) = (2/3.6, 3/3.6) = (0.55, 0.83);

Dot Product

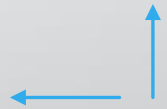
- It's the multiplication of 2 vectors the result is a float (scalar)
  - $a \cdot b = |a| \cdot |b| \cdot \cos(\theta)$
  - $|a|$ magnitude of vector a
  - $|b|$ magnitude of vector a
  - θ the angle between a and b
- If vectors are normalised it returns a value between 0-1

Dot Product

- Vectors point same direction = 1

- Vectors are at 90 degrees = 0

- Vectors are at 90 degrees = 0



- Vectors point in different directions = -1

Cross Product

- It's the multiplication of 2 vectors, the result is a third vector perpendicular on the 2 vectors

$$a \times b = |a| * |b| * \sin(\theta) * n$$

$|a|$ magnitude of vector a

$|b|$ magnitude of vector a

θ the angle between a and b

n unit vector at right angles to both a and b

- If vectors are normalised it returns a value between 0-1

Cross Product

- If you look down on the surface you should choose the vectors clockwise and you will get a positive vector (going up - left hand rule)
- If you go anticlockwise you will get a negative vector(pointing down) but the same magnitude
- Another useful element is the Area of the surface between the 2 vectors which is equal to resulting vector length divided by 2

Areatriangle = perpmagnitude / 2;

Shading models

and components

Ambient component

- Represents light that illuminates all surfaces equally and reflects equally in all directions
- It is normally used to brighten some of the darker areas within a scene

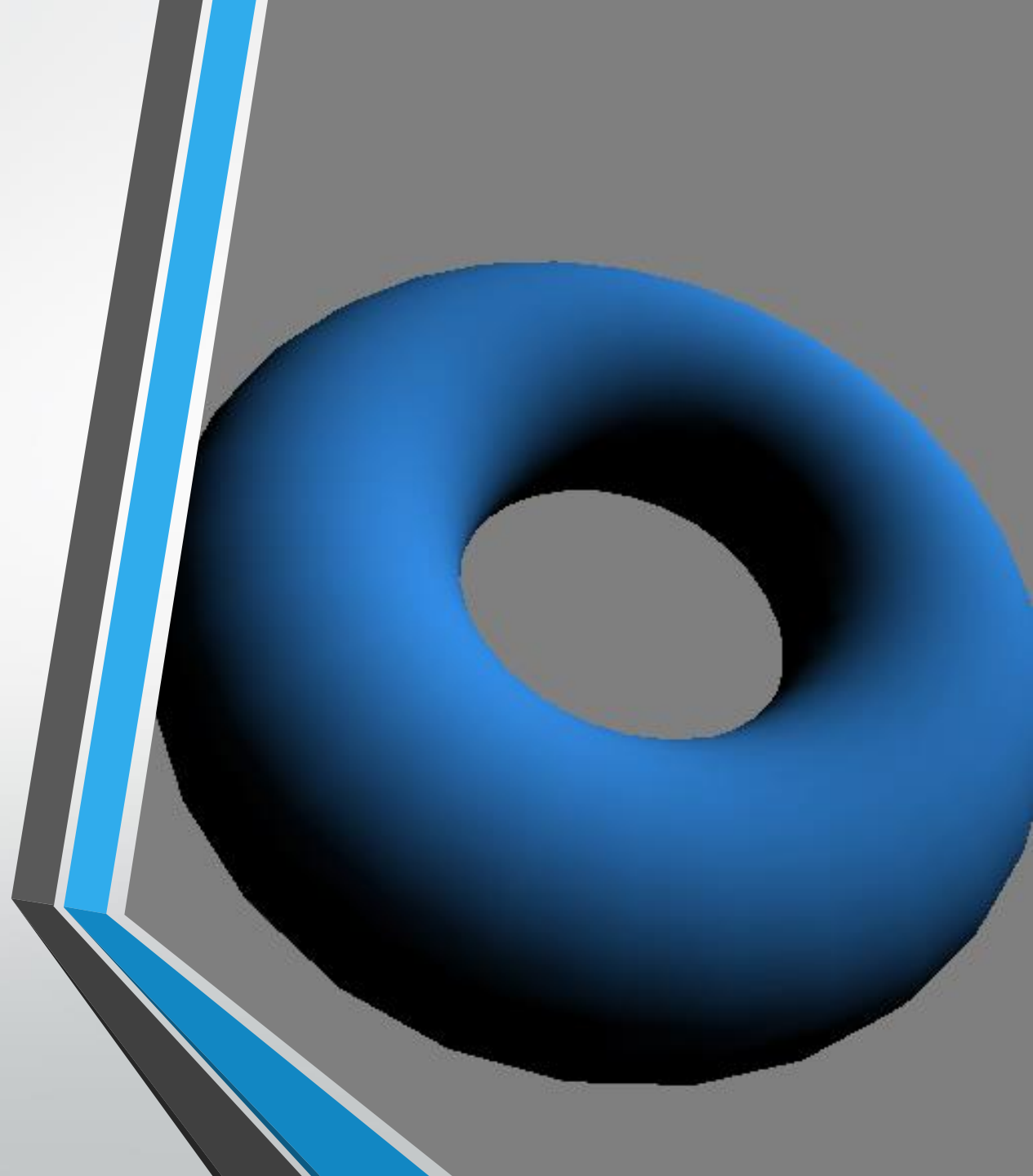
$$Ia = Ka \cdot La$$

Ka : surface reflectivity

La : light source intensity

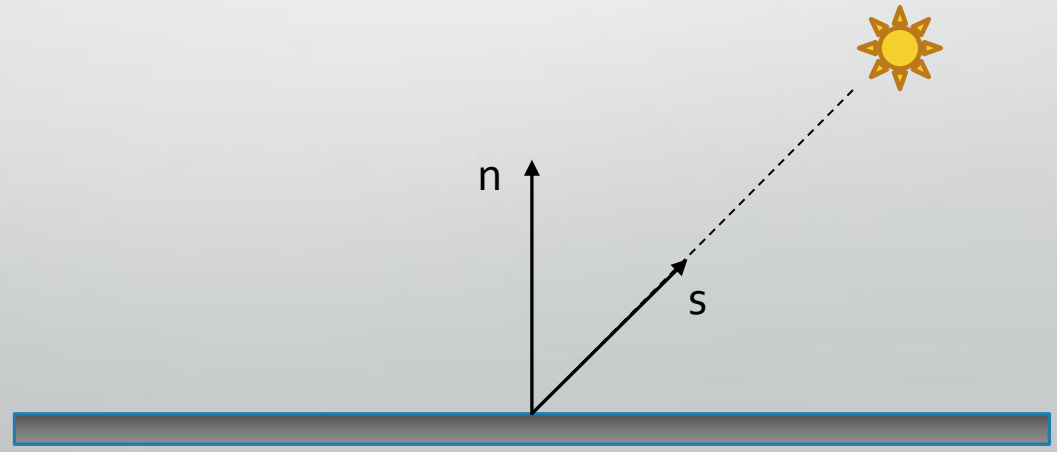
Diffuse reflection model

- The surface exhibits purely diffuse reflection (the surface appears to scatter light in all directions equally
- Imagine an object that is covered in matte paint



Diffuse reflection – vectors

- Uniform omnidirectional scattering
- 2 vectors are involved in this model:
  - n : normal vector
  - s : the direction from the surface point to the light source



Diffuse reflection – mathematical model

$L = K_d \cdot L_d \cdot (s.n)$

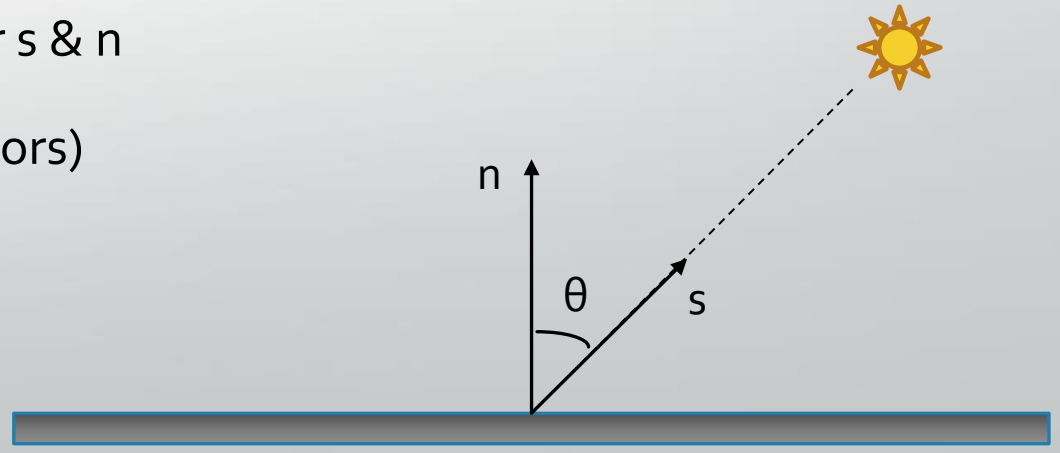
$L = K_d \cdot L_d \cdot (s \cdot n \cdot \cos\theta)$

K_d – diffuse reflectivity (scaling factor)

L_d – intensity of light source

$s.n$ – dot product between vector s & n

$s.n = s \cdot n \cdot \cos\theta$ (normalised vectors)



Phong reflection model

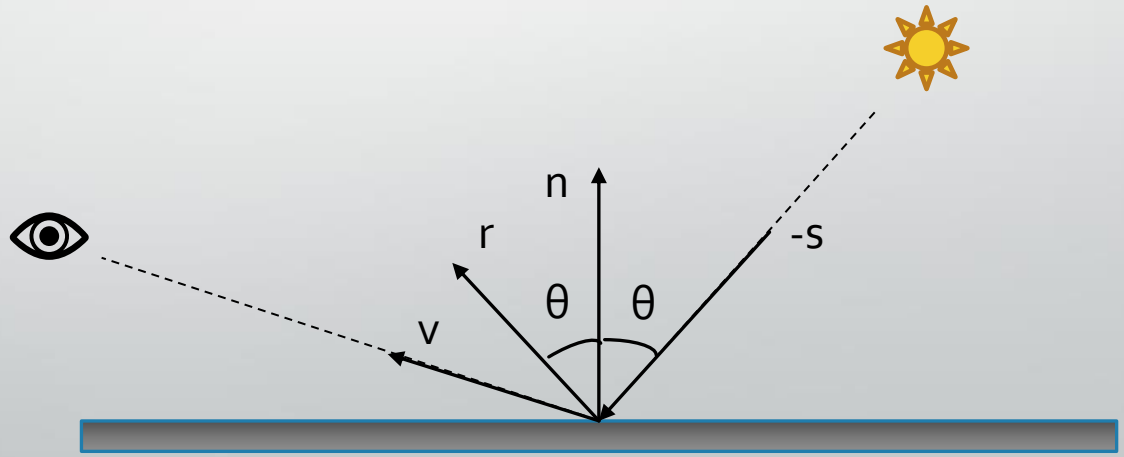
- Models the light-surface interaction as a combination of:
  - Ambient
  - Diffuse
  - Specular

Phong reflection model

- **Ambient** – intended to model light that has been reflected so many times that appears to be emanating uniformly from all directions
- **Diffuse** – represents omnidirectional reflection covered earlier in this session
- **Specular** – models the shininess of the surface and represents glossy reflection around a preferred direction.

Phong reflection model – vectors

- n: surface normal
- -s: light direction
- r: reflection vector
- v: viewing vector
- θ: angle between normal & reflection vector



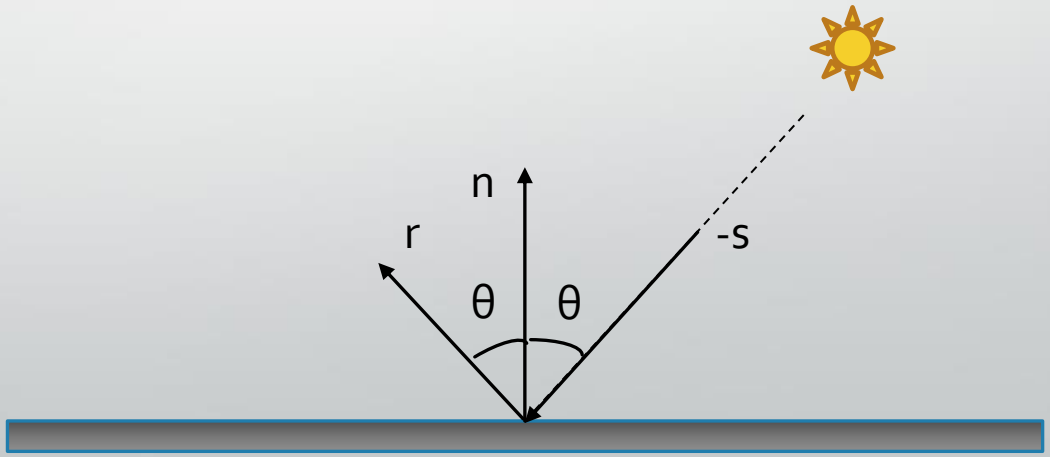
Specular component

- This component is used for modelling the shininess of a surface.
- Glossy surface: the light is reflected off the surface, scattered around some preferred direction.
- The reflected light is strongest in the direction of perfect reflection.
- For perfect reflection, the angle of incidence is the same as the angle of reflection and that the vectors are coplanar with the surface normal.

Specular component

- The reflection vector (r) can be calculated with the following formula:

$$r = -s + 2(s.n)n$$



Specular component

$$I_s = K_s \cdot L_s \cdot (r.v)^f$$

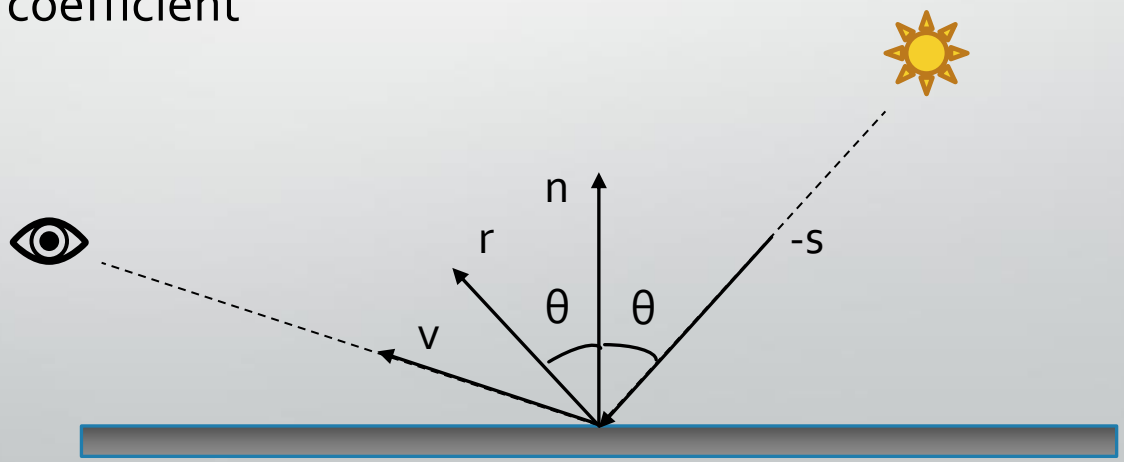
K_s : specular reflectivity (typically some grayscale value)

Ls : light intensity

R: reflection vector

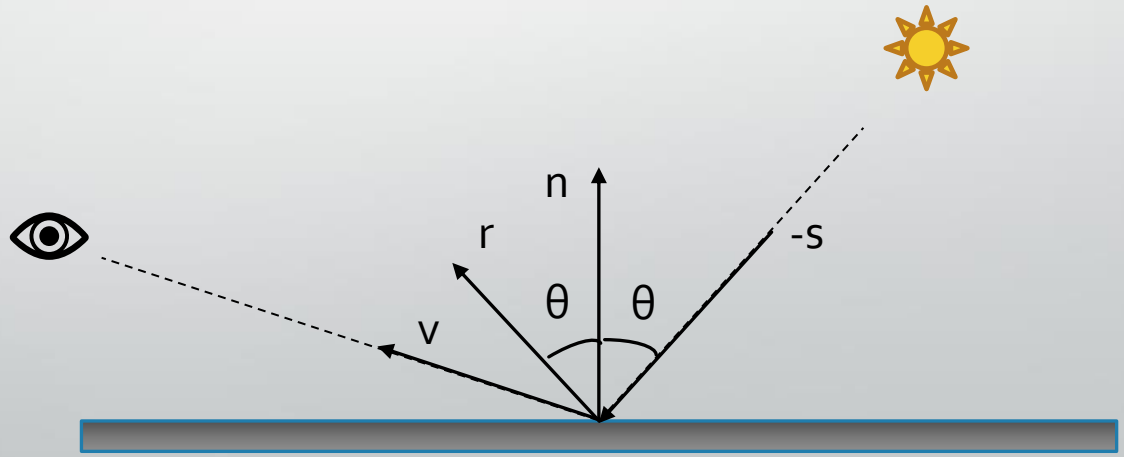
V: viewing vector

F: power coefficient



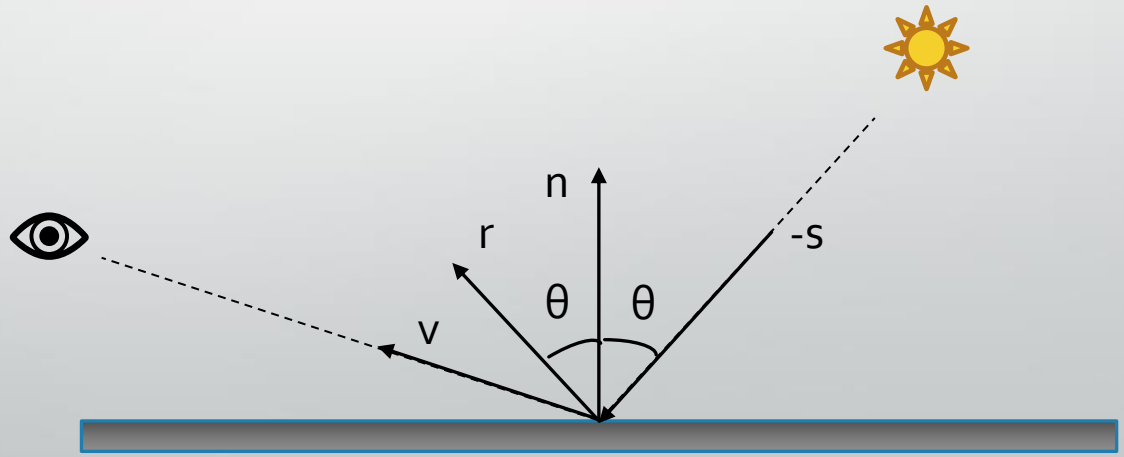
Specular component

- The reflection needs to be at its maximum when the viewer is aligned with vector r.
- We want it to fall off quickly as the viewer moves away from alignment
- Creates specular highlights (bright spots). The larger the power of f the smaller the specular highlight and the shinier the surface.



Phong reflection – mathematical model

	Ambient		Diffuse		Specular
$I =$	I_a	$+$	I_d	$+$	I_s
$I =$	$K_a \cdot L_a$	$+$	$K_d \cdot L_d \cdot (s.n)$	$+$	$K_s \cdot L_s \cdot (r.v)^f$



Per-vertex versus per-fragment

- If we do all the shading calculations within the vertex shading, we call this **per-vertex shading**.
- Per-vertex shading is also called **Gouraud** shading.
- Disadvantages: specular highlights can be warped or lost
- **Per-fragment** shading can improve realism but it is more expensive as the calculations are applied to every pixel.

Attenuation

- Light attenuation is useful for a point light (light bulb in your room), the further you are from the light source less intensity for the light
- In order to implement the fall off, all we have to do is to calculate the light intensity as we normally do and divide everything by the inverse square of the distance from the source
- Or we can use something with a less steeper curve

$$\frac{1}{(r+k)} \qquad L = (K_d \cdot L_d \cdot (s.n)) / (r+k) \quad \text{//difuse with fall off}$$

r – distance

k – constant value

Directional light

- If we use a directional light, we can avoid calculating the light direction in the shader (we can send direction directly through a uniform)
- At the moment we calculate light direction based on the position of the light and the vertex position.

s = lightPosition – vertex position; //direction of the light

Directional light has only direction, no position (saves us a calculation.)

Useful links

- To read: Chapter 3 - Introduction to 3D Graphics & Chapter 2 - Getting Started (OpenGL Superbible – see link on the DLE)
- To read: Chapter 4 - Introduction to 3D Graphics & Chapter 2 - Getting Started (OpenGL Superbible – see link on the DLE)
- Coordinate systems: https://learnopengl.com/Getting-started/Coordinate-Systems
- To read: Transformations - https://learnopengl.com/Getting-started/Transformations
- To read: The basics of GLSL in OpenGL 4 Shading Language Cookbook
- Essence of linear algebra (YouTube): https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab