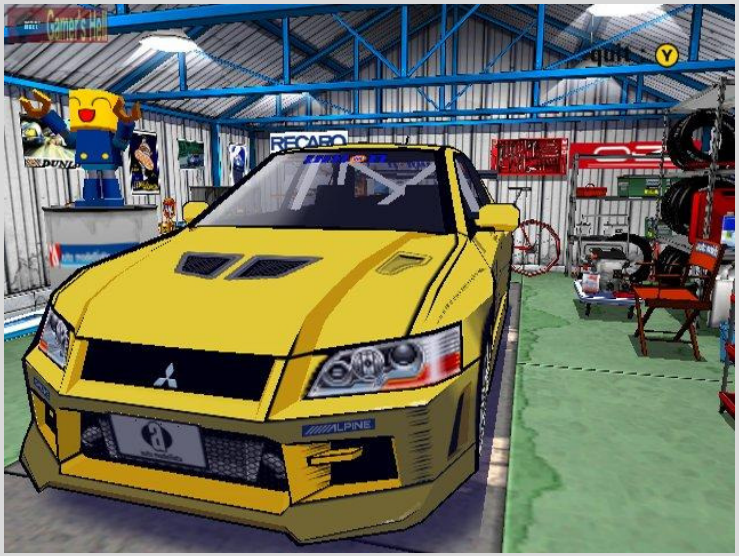
Illumination & Shading

Lecturer: Rachel McDonnell
Assistant Professor of Creative Technologies
Rachel.McDonnell@cs.tcd.ie

Course www: https://www.scss.tcd.ie/Rachel.McDonnell/
Credits: Some slides from Carol O'Sullivan

Introduction

- Realistic image synthesis
  - Photorealism vs. physically-based realism
  - Film and visual effects, architecture, ergonomic design of buildings and offices, computer games, lighting engineering, predictive simulations, flight and car simulators, advertising
- Non-photorealistic rendering
  - Suited for an artistic, technical, or educational approach
  - Pen-and-ink drawings, cartoon-style drawings, technical illustrations, watercolour painting etc.

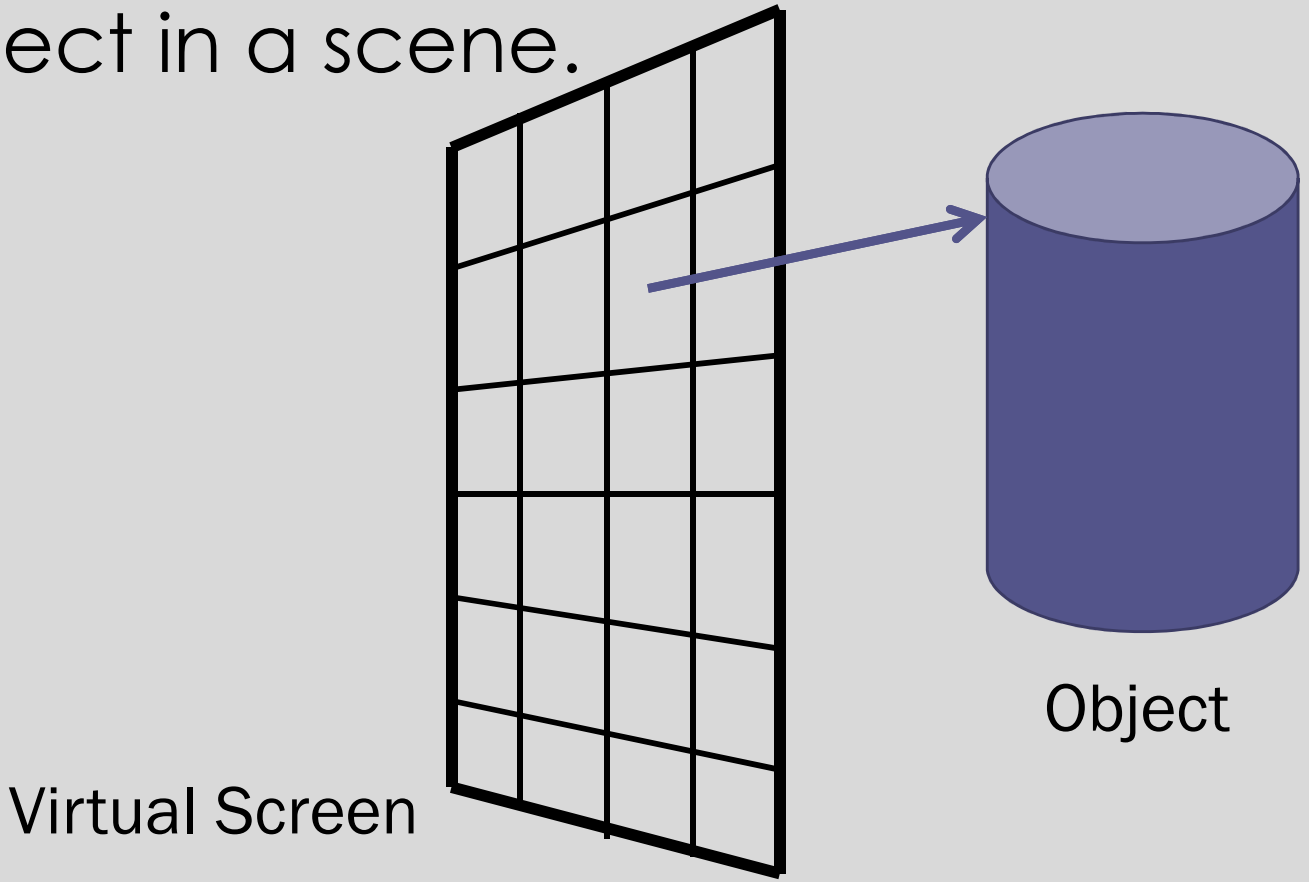


Overview

- Rendering algorithms (local, global, view dependent, view independent)
- Light, colour, spectra
- Surface reflectance
- Solid angle, radiometric units, radiance
- Inverse square law, the cosine rule
- BRDF, BRDF approximations
- Reflectance equation, radiance equation
- Light sources, Lambertian illumination model

Rendering

- Rendering is fundamentally concerned with determining the *most appropriate colour* (i.e. RGB tuple) to assign to a pixel associated with an object in a scene.



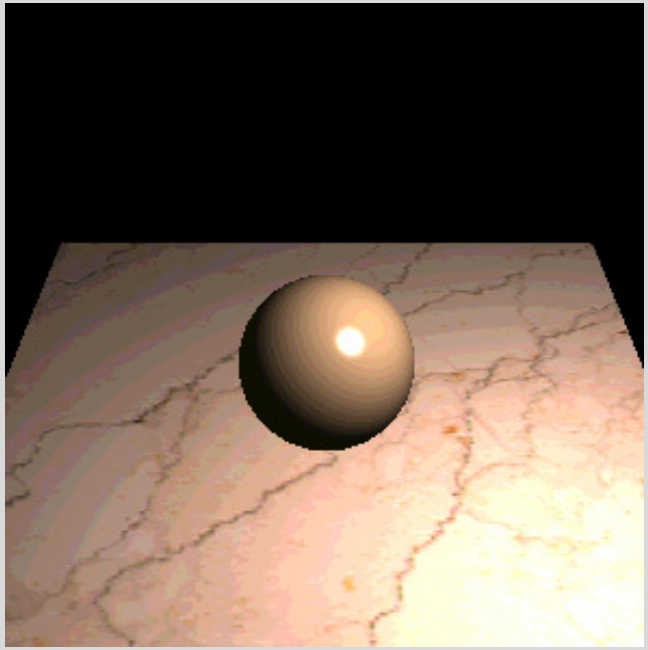
Rendering

- The colour of an object at a point depends on:
  - geometry of the object at that point (*normal* direction)
  - position, geometry and colour of the light sources (*luminaires*)
  - position and visual response of the viewer
  - *surface reflectance properties* of the object at that point
  - scattering by any *participating media* (e.g. smoke, rising hot air)

Rendering Algorithms

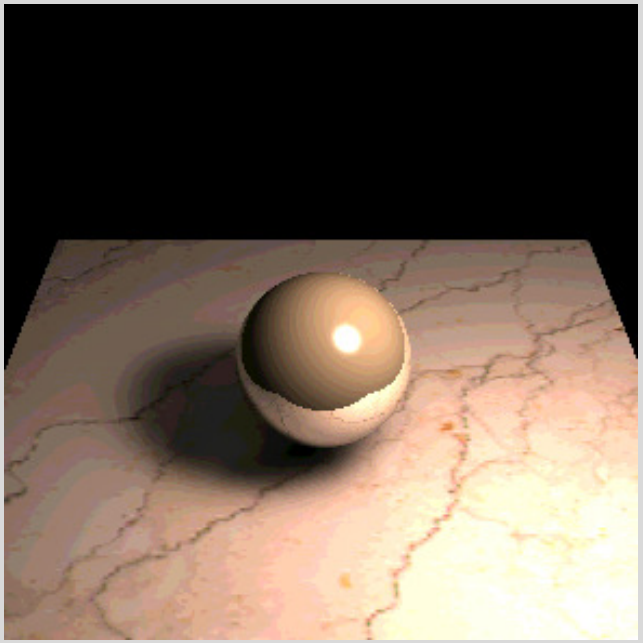
- Rendering algorithms differ in the assumptions made regarding lighting and reflectance in the scene and in the solution space:
  - **local illumination** algorithms: consider lighting only from the light sources and ignore the effects of other objects in the scene (i.e. reflection off other objects or shadowing)
  - **global illumination** algorithms: account for all modes of *light transport*
  - **view dependent** solutions: determine an image by solving the illumination that arrives through the viewport only.
  - **view independent** solutions: determine the lighting distribution in an entire scene regardless of viewing position. Views are then taken after lighting simulation by sampling the full solution to determine the view through the viewport.

Local vs. Global Illumination



Local

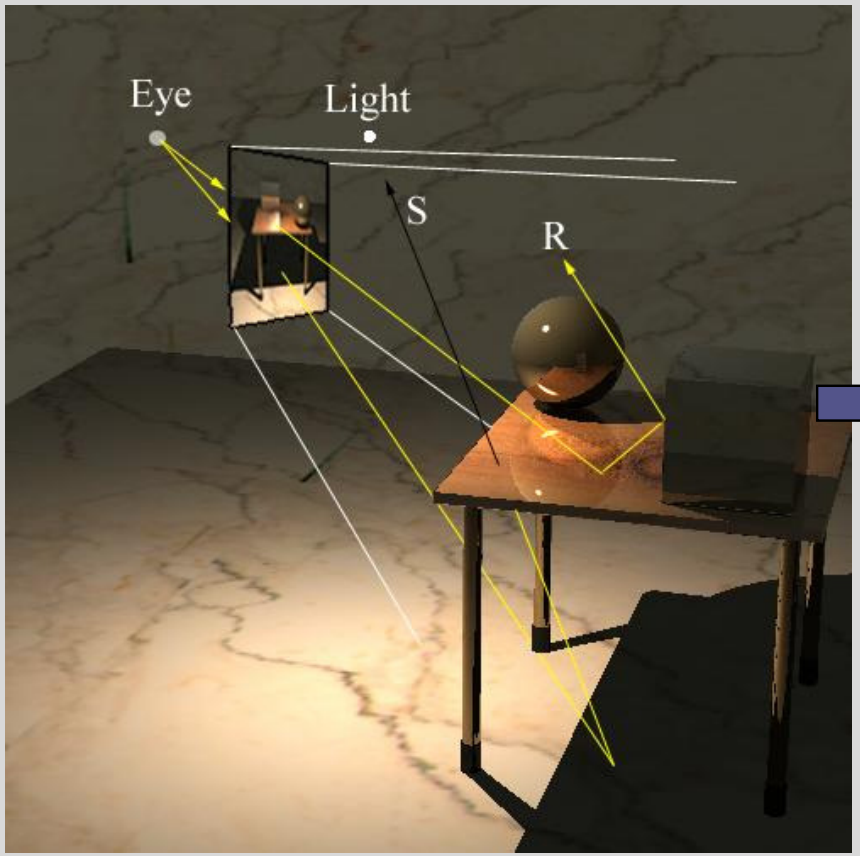
Illumination depends on local object & light sources only

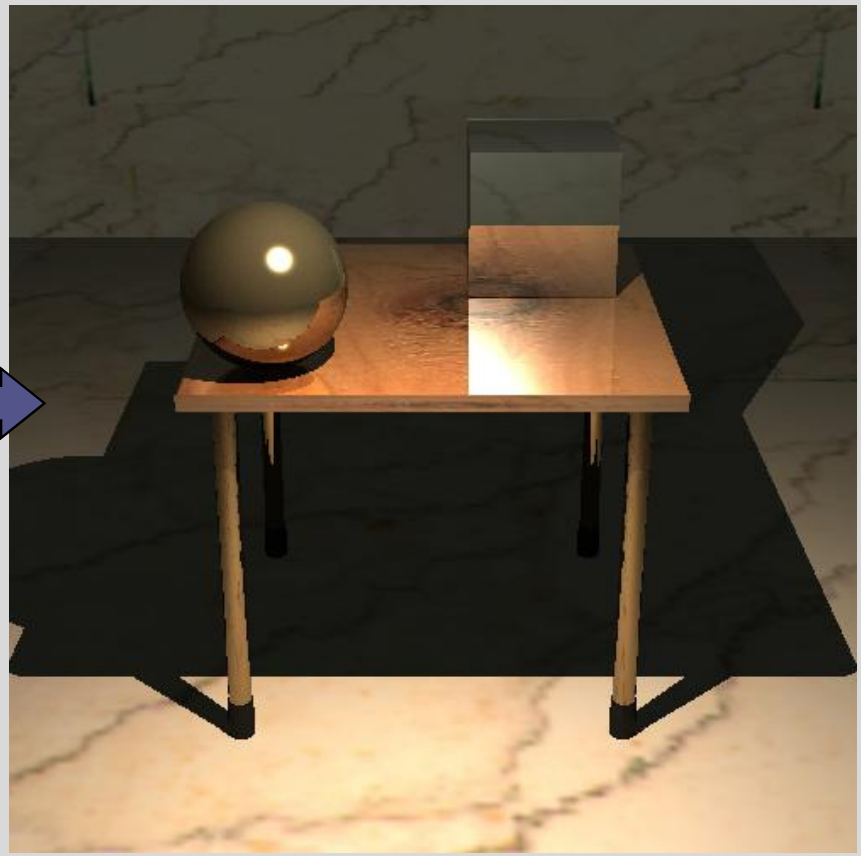


Global

Illumination at a point can depend on any other point in the scene

View Dependent Solution (Ray Traced)





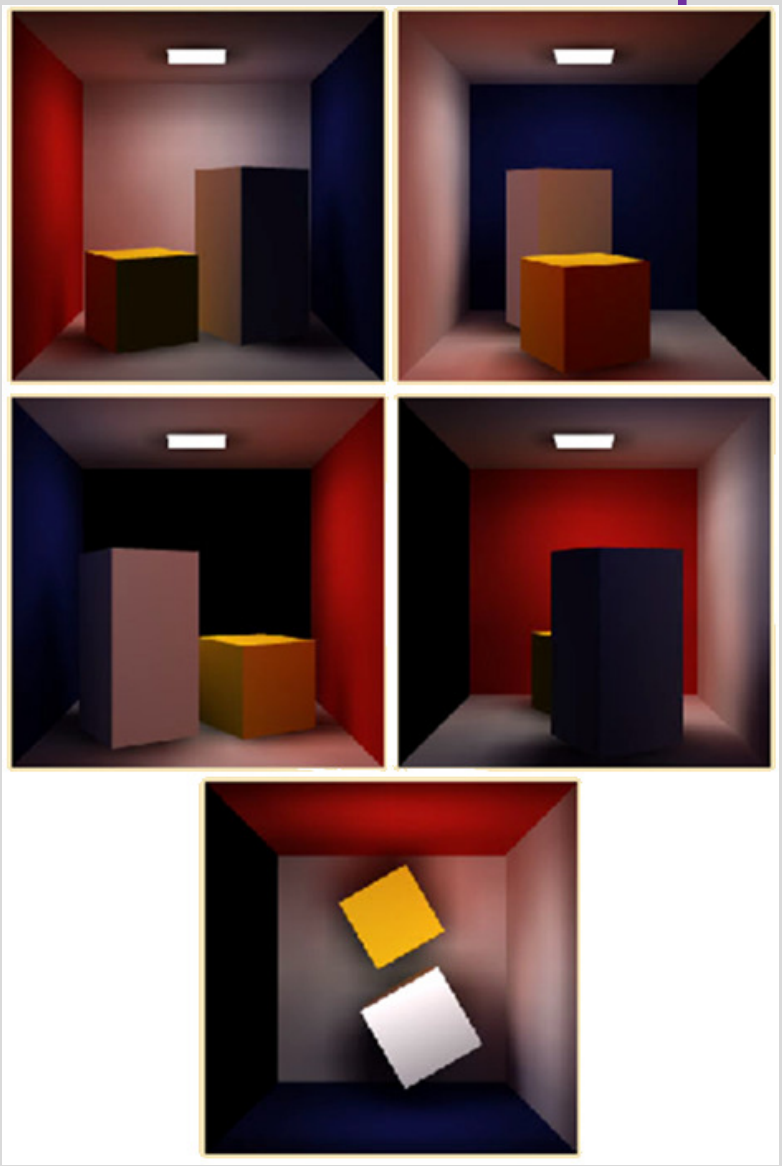
Scene Geometry

Solution determined only for directions through pixels in the viewport

View Dependent

- Advantages
  - Only the relevant parts of the scene are taken into consideration
  - Can work with any kind of geometry
  - Can produce very high-quality results
  - In some methods, view-dependent portions of the solution can be cached as well (glossy reflections, refractions etc).
  - Require less memory than a view-independent solution.
- Disadvantages
  - Requires updating for different camera positions; still, in some implementations portions of the solution may be re-used.

View Independent (Radiosity)



A single solution for the light distribution in the entire scene is determined in advance.

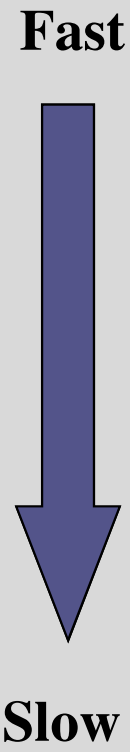
Then we can take different snapshots of the solution from different viewpoints by sampling the complete solution for specific positions and directions.

View Independent

- Advantages
  - Solution needs to be computed only once.
- Disadvantages
  - All of the scene geometry must be considered, even though some of it may never be visible.
  - The type of geometry in the scene is usually restricted to trianglular or quadrangular meshes (no procedural or infinite geometry allowed).
  - Detailed solutions require lots of memory.
  - Only the diffuse portion of the solution can be cached; view-dependent portions (glossy reflections) must still be computed.

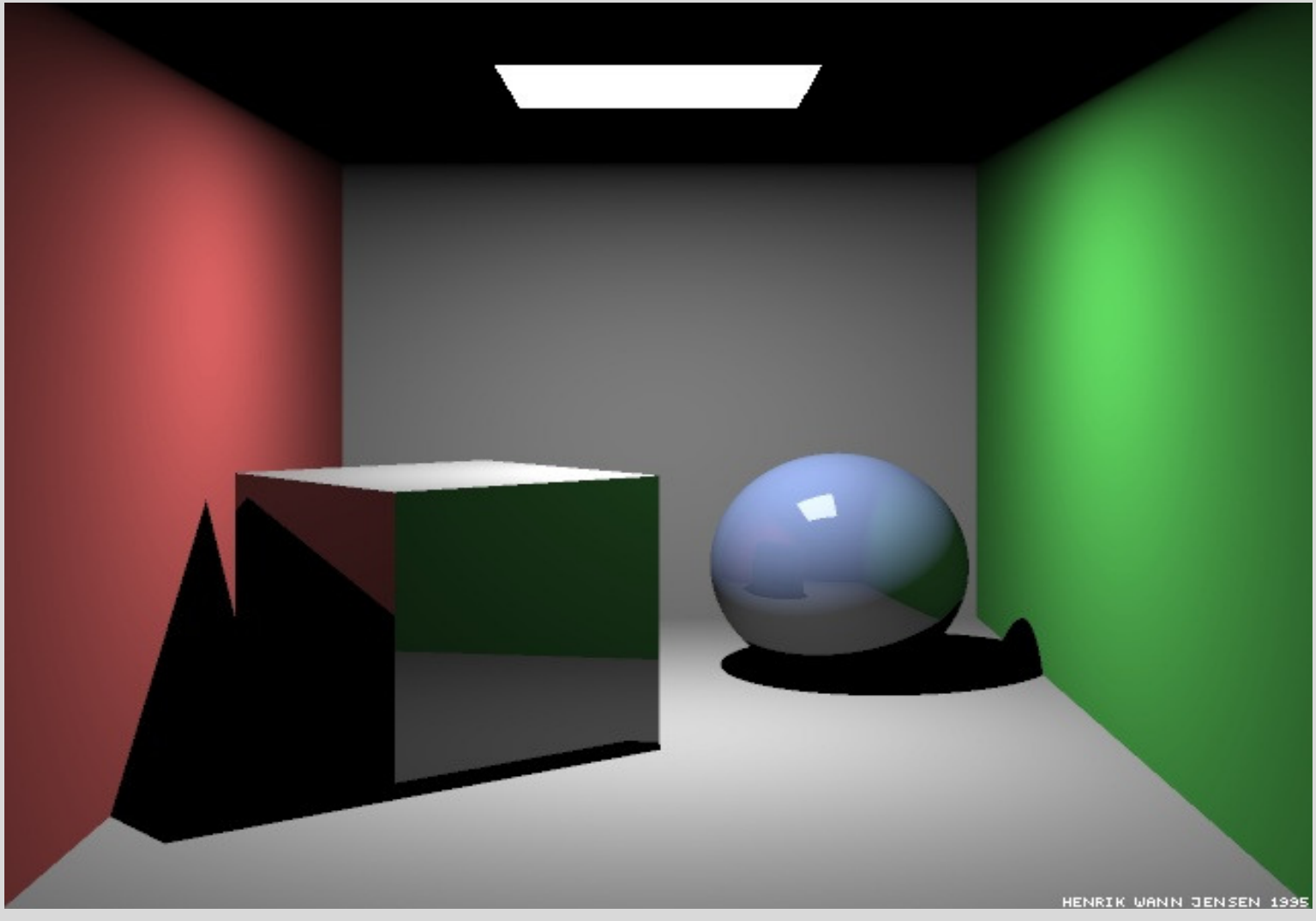
Global Illumination Algorithms

- Different algorithms solve the illumination problem with making different assumptions to vary the speed/accuracy tradeoff.
  - ***Z-Buffer Algorithms***:
    - can compute approximate shadows and reflection from planar surfaces
  - ***Ray Tracing Algorithms***:
    - determine exact shadows, reflections and refractive effects (transparency) assuming point light sources (no volume).
  - ***Radiosity Algorithms***:
    - computes approximate solutions assuming no *shiny* surfaces, but light sources can be arbitrarily large and all surfaces polygonal.
  - ***Path Tracing Algorithms***:
    - employing an expensive Monte-Carlo solution to handle arbitrary geometries, reflectance and lighting.

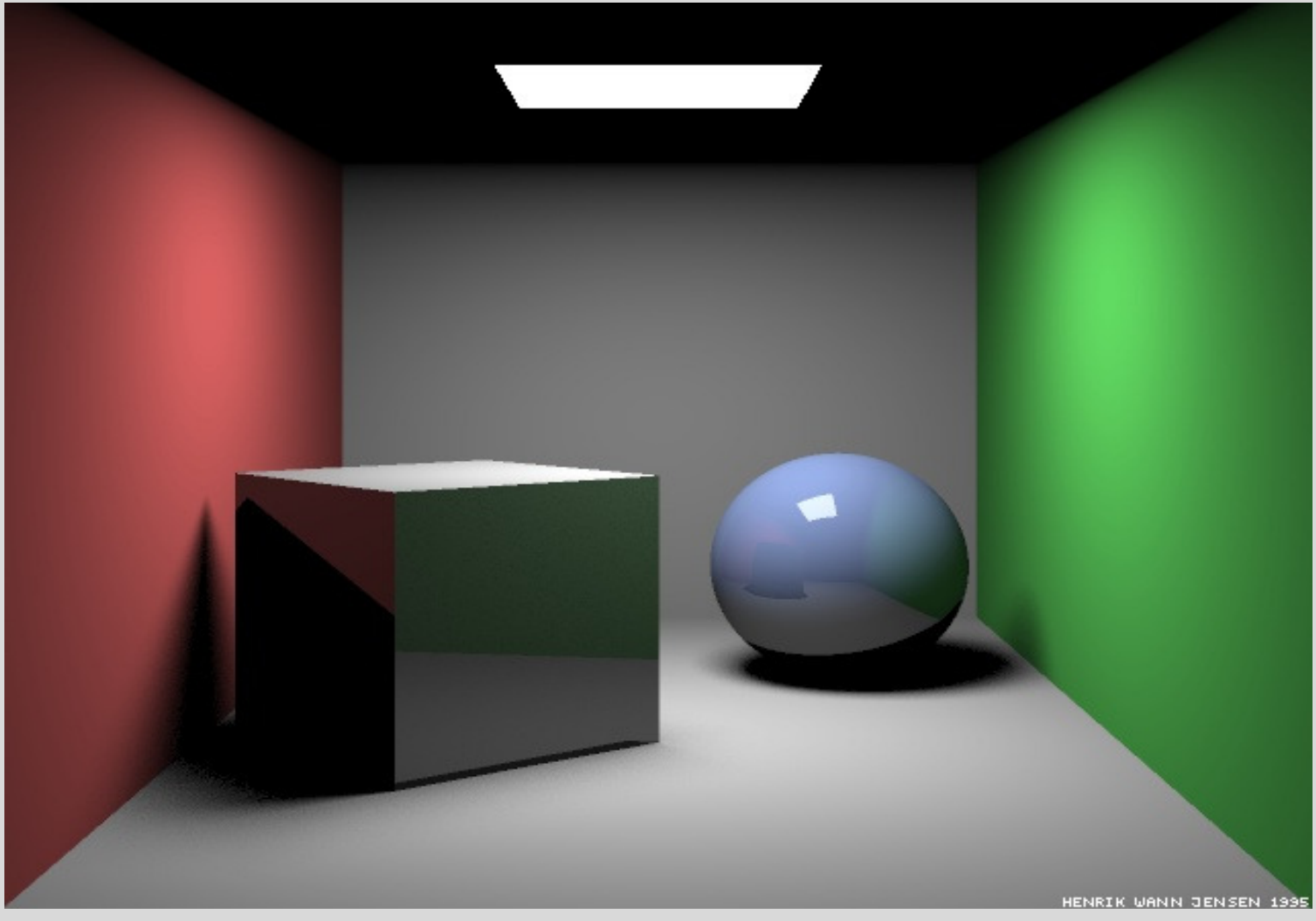


Z-buffer methods only

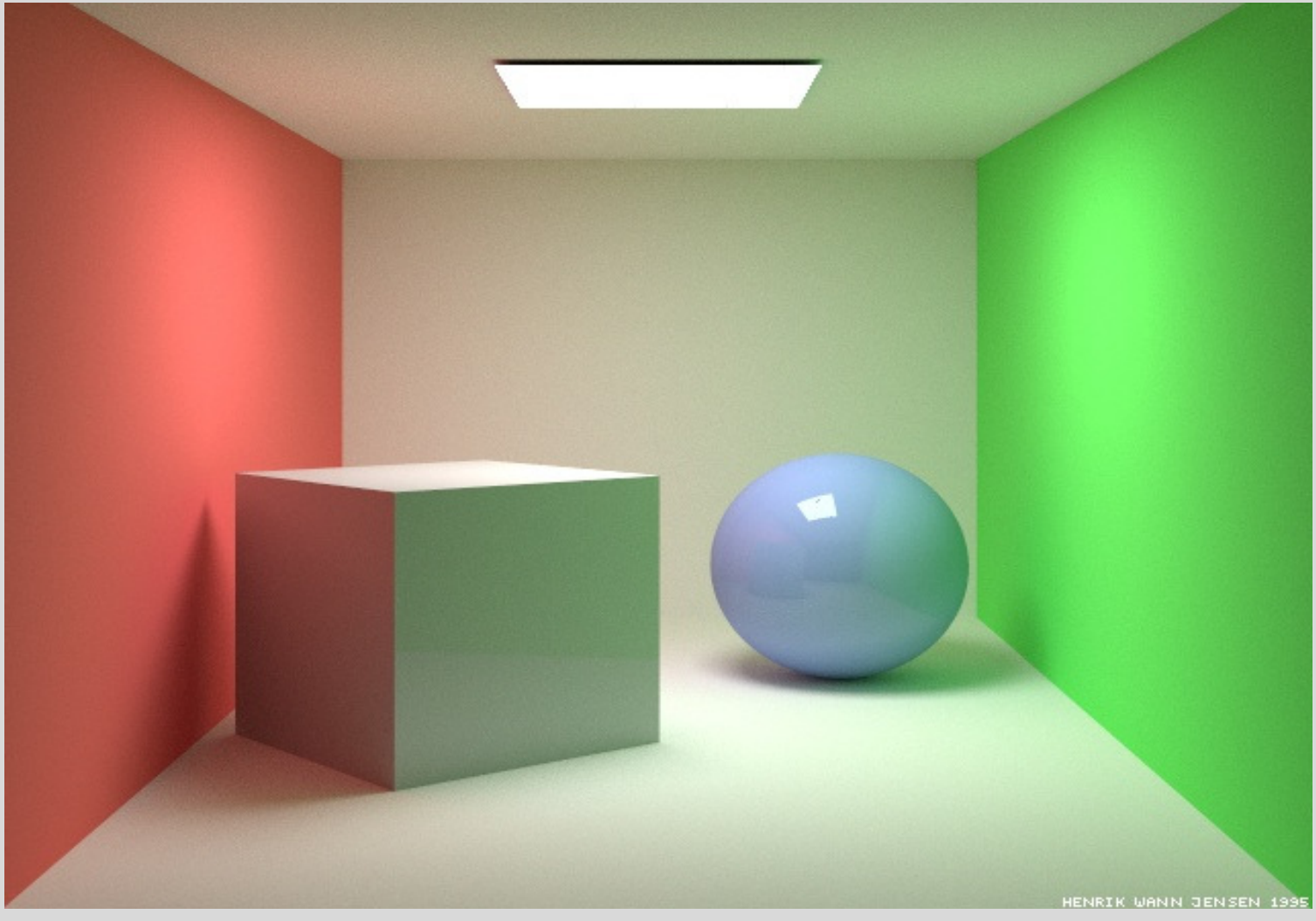
Crystal Glass Rendering using Path Tracing



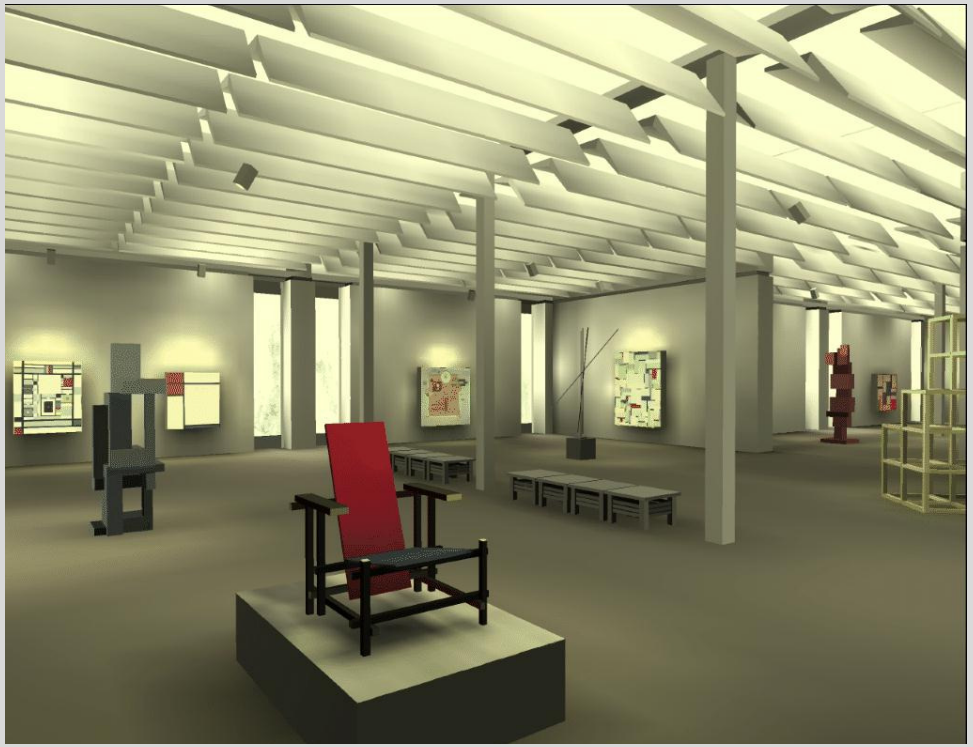
Ray traced image



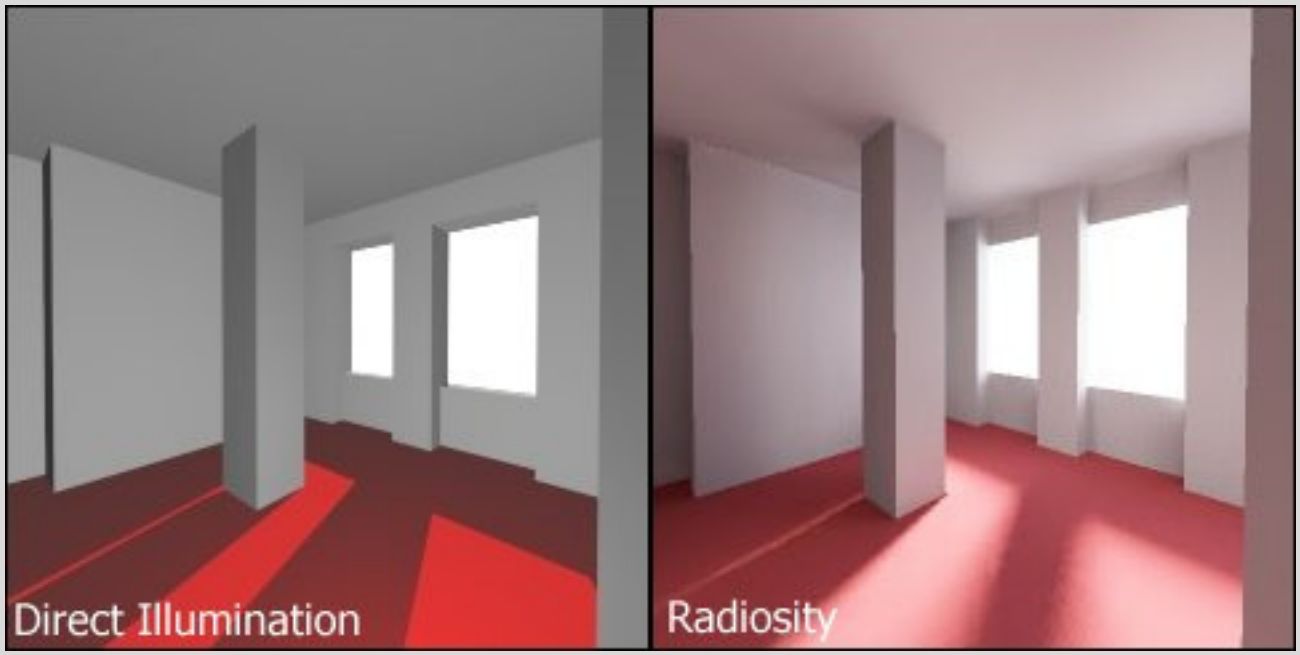
Stochastic ray traced image



Path traced image



Typical Radiosity Scene



Radiometry & Photometry

- *Radiometry* = The science of the physical measurement of electromagnetic energy.

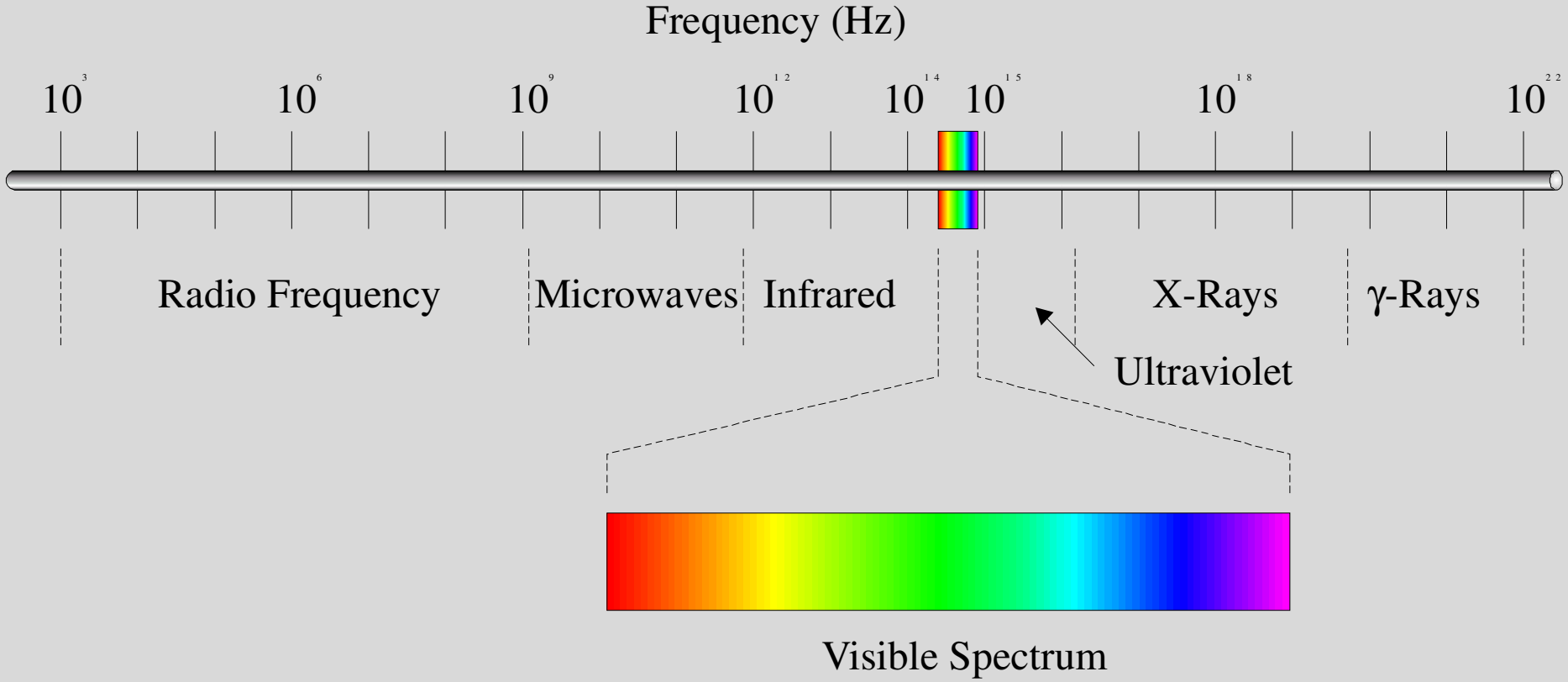
- *Photometry* = the psychophysical measurement of the visual sensation produced by the electromagnetic spectrum.
- In photometric quantities every wavelength is weighted according to how sensitive the human eye is to it, while radiometric quantities use un-weighted absolute power.

Radiometry

- Radiometry deals with the measurement of electromagnetic radiation
- Flow of *photons*
- Usually think of photons as particles
  - Except when interaction of photons with sensors in the human eye
  - Different frequencies of photons are perceived as light of different colours

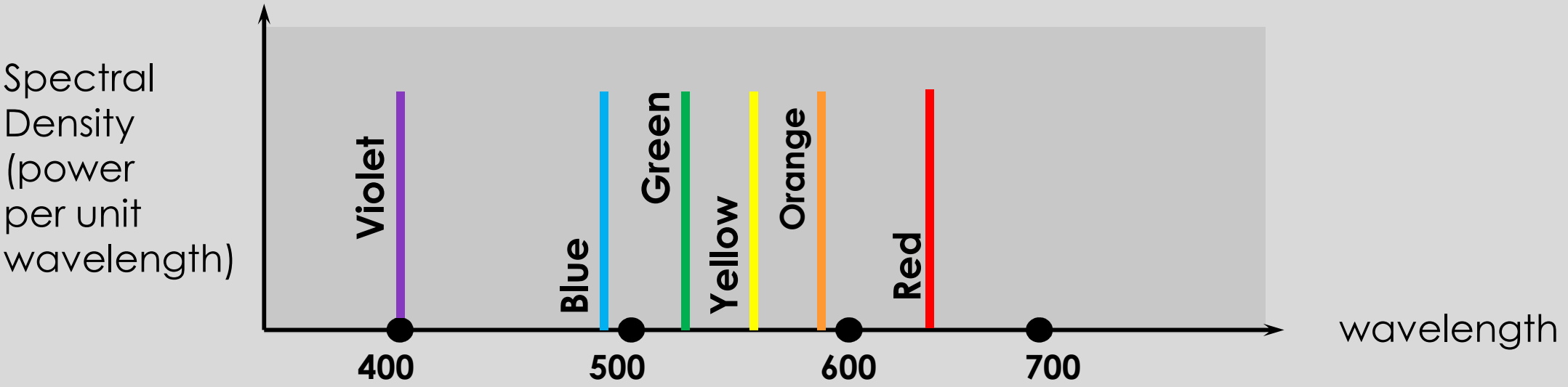
Light

- An electromagnetic phenomenon, like television waves, infrared radiation, and x-rays
- Light means those waves that lie in a narrow band of wavelengths in the "visible spectrum"



Pure Spectral light

- The eye responds to light with wavelengths between approximately 400 and 700 nm
- Some light sources emit light of essentially a single wavelength (e.g. lasers)
- Here are some spectral densities for pure spectral light, and how we perceive them:

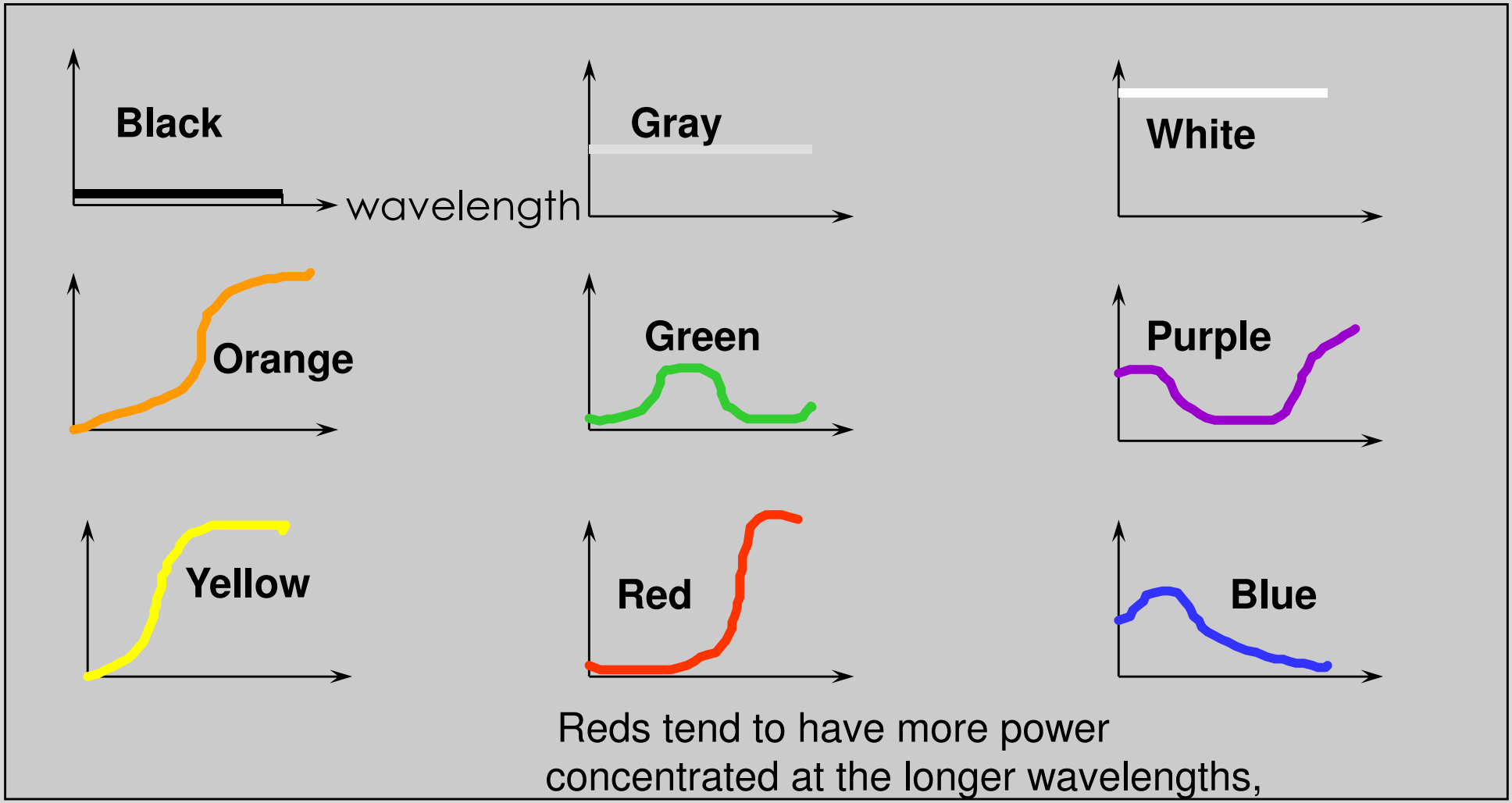


Perceived Colour

- The light from most sources, however, does not consist of only one wavelength
- It consists of power of various amounts over a continuous set of wavelengths
- Their spectral densities (Spectra) cover a **band of wavelengths**
- An enormous variety of spectra is perceived by the eye as having the same colour

Example spectra

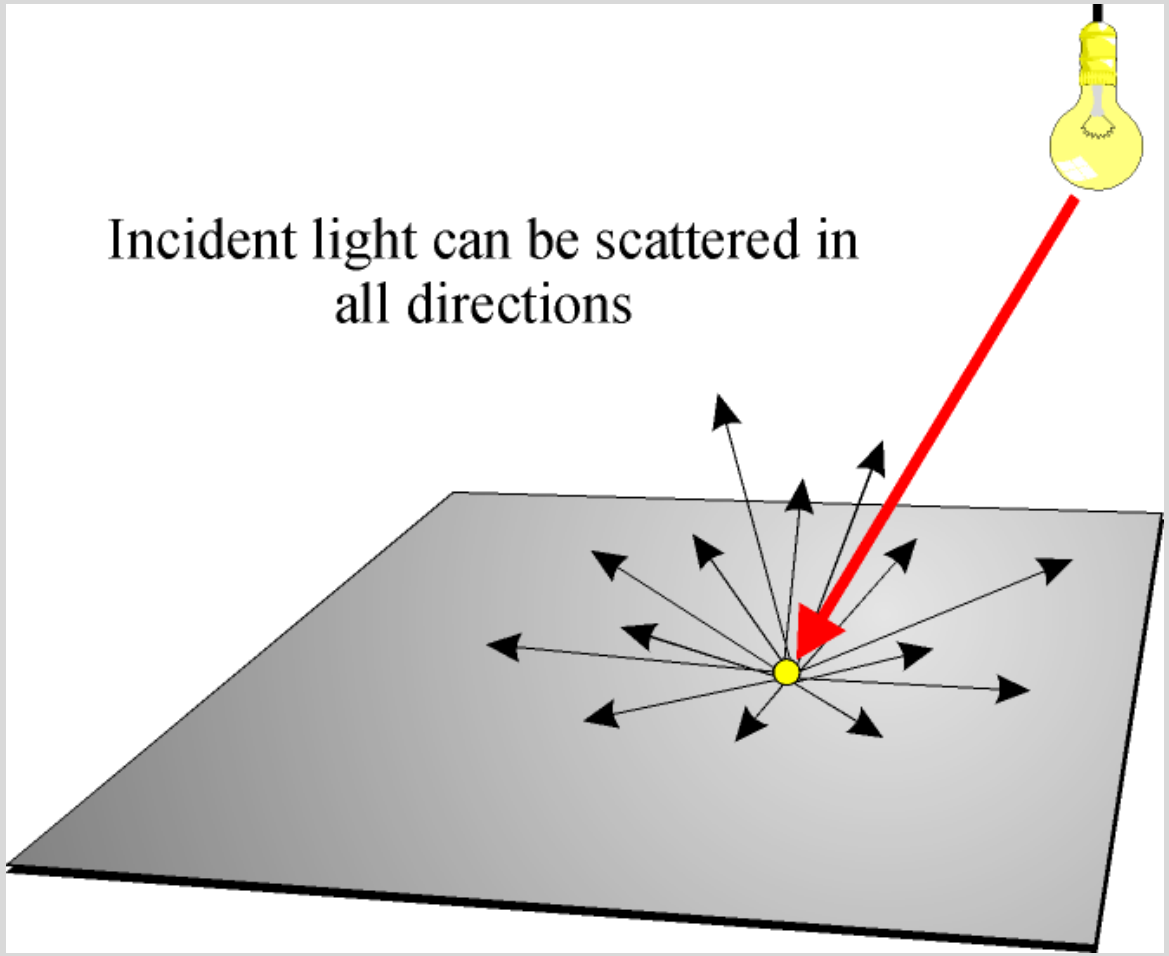
The total power of the light in any band of wavelengths is found as the area under the density curve over that band

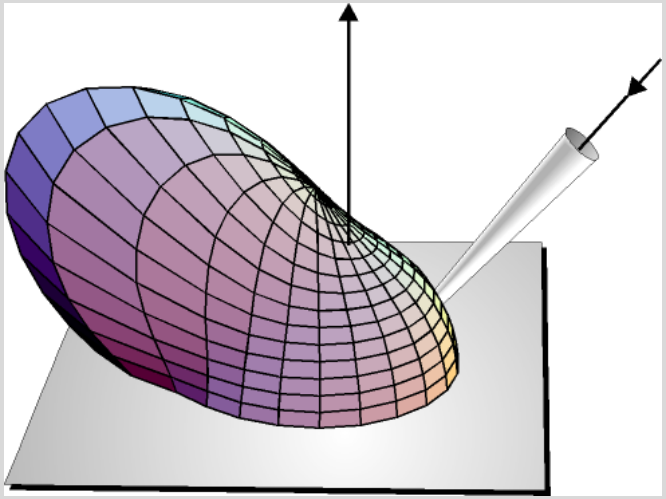


Reds tend to have more power concentrated at the longer wavelengths, and blue at shorter wavelengths

Surface Reflectance

- Much of current realistic image synthesis research is devoted to the modeling of surfaces
  - in particular the solving of ***light reflection*** off arbitrarily complex surface geometries.
- A surface ***scatters*** light that is incident on it.
  - This scattering can theoretically distribute the scattered light in any direction from the scattering point.
- Most algorithms make **assumptions** regarding the directions through which the light is scattered = ***scattering distribution***
- View dependent algorithms must determine the point in the scene which is visible through each pixel and then *determine the light that is scattered from here towards the pixel*.

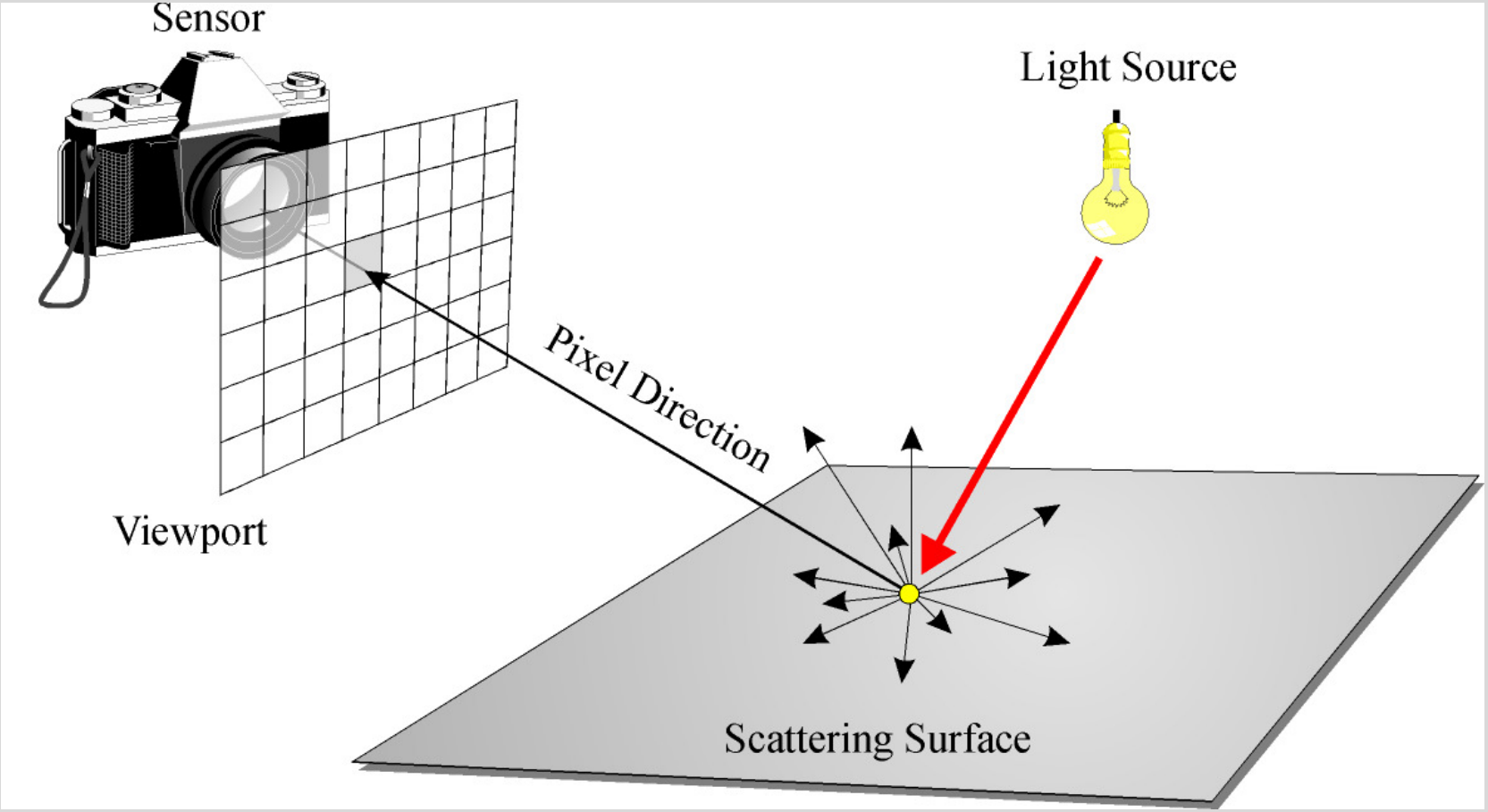




BRDF

- Energy is scattered from a surface in a distribution that depends on the surface's **microscopic geometry**.
- This distribution can be described as a function (strictly positive) that records the reflected energy per direction

= BRDF: Bidirectional Reflectance Distribution Function



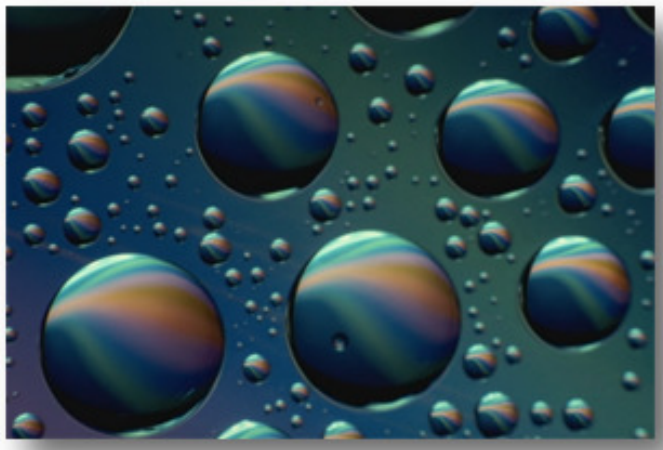
- An image is formed when light energy is scattered by surfaces in the scene towards the viewport
  - (or emitted directly towards the viewport)

Multi-layered Surfaces

Coloured Glass

Human Skin

Stone

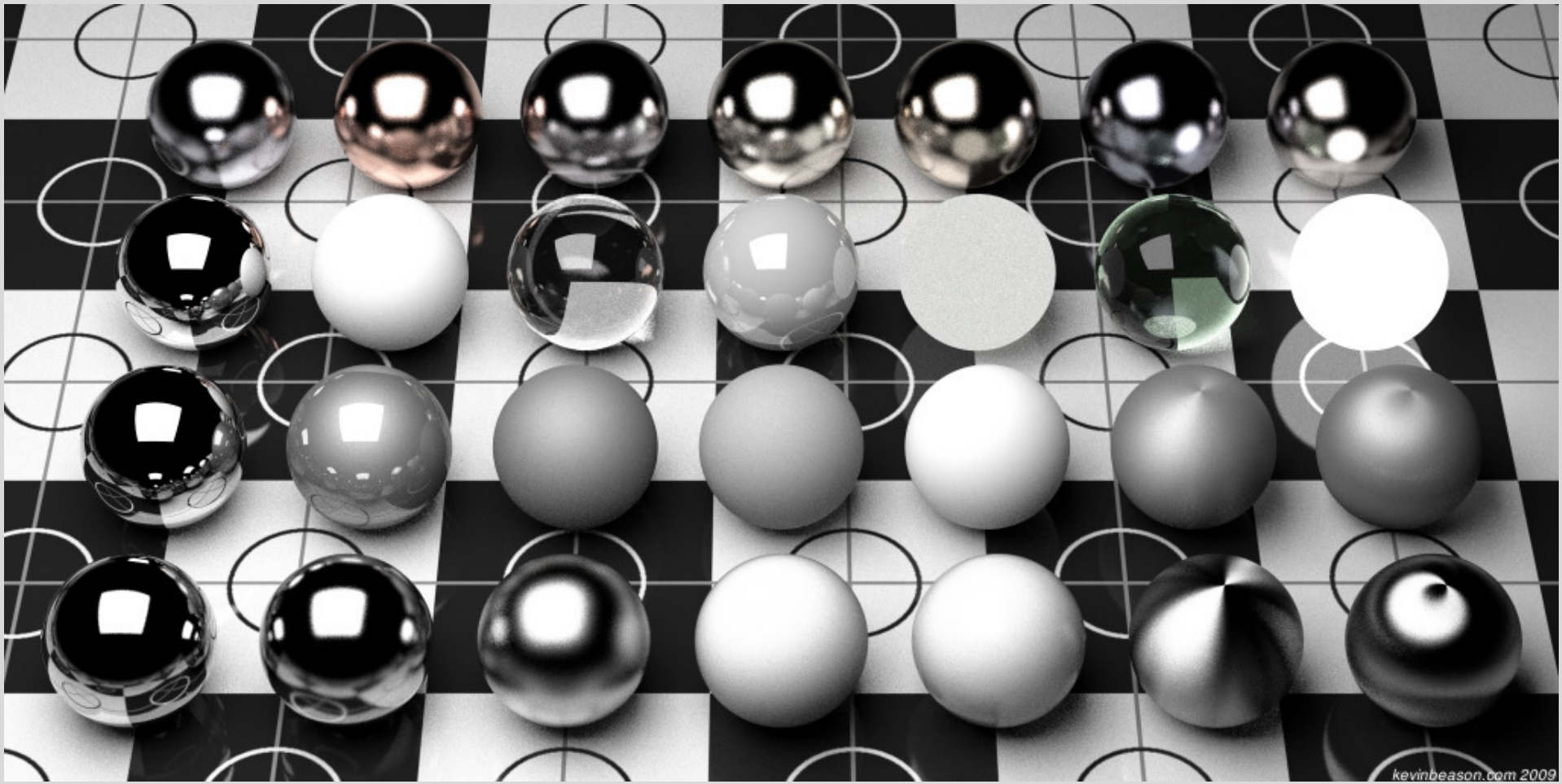


Thin Film

Tarnished Metal

Complex Surface Scattering Examples

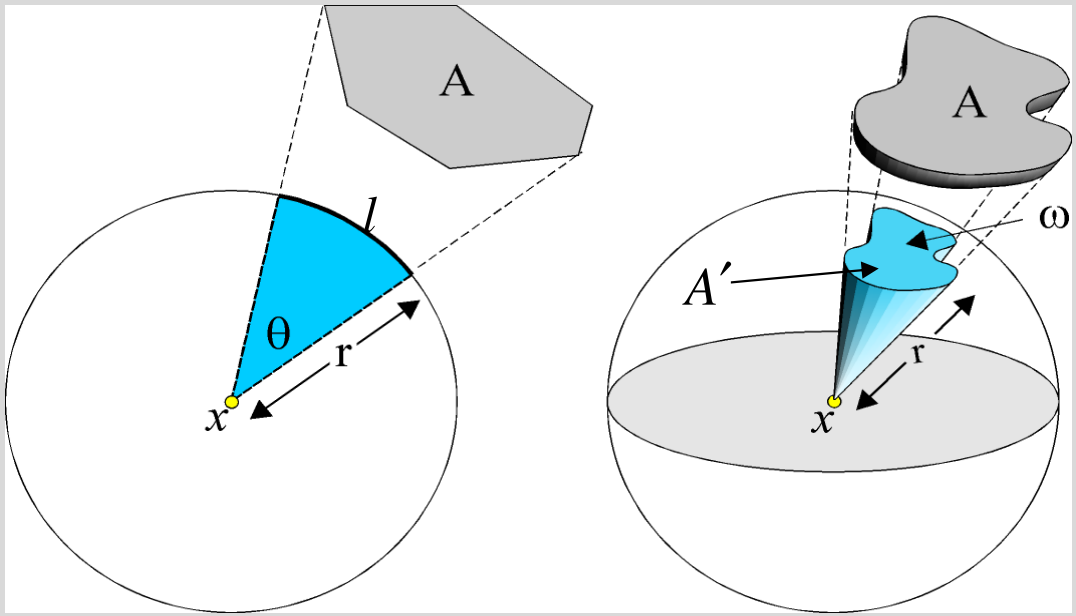
Different BRDFs



Subsurface scattering, NVIDIA

Solid Angle

- We need to define a set of directions using some metric:
- *solid angle* = 3D equivalent of the angle (units = steradians *sr*)
- The solid angle subtended by surface *A* at a point x is related to the area of *A* when projected towards x onto a *unit sphere* centered at x



$$\text{angle } \theta = \frac{l}{r}$$

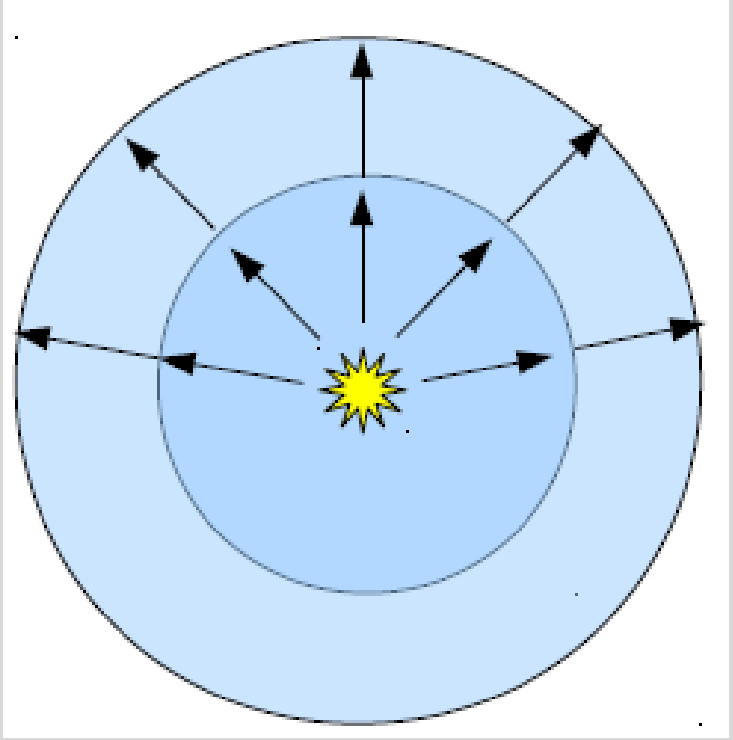
$$\text{solid angle } \omega = \frac{A'}{r^2}$$

Radiometric Units

- The total *energy* (Joules) leaving a surface per unit time:
  $\Rightarrow$ *power* or ***Flux*** (Watts) = Φ
- The flux leaving a surface can change with position on the surface (i.e. some points are brighter), so flux per unit area:
  $\Rightarrow$ ***Radiosity*** (Watts/m^2) = **B**
- Flux arriving per unit area:
  $\Rightarrow$ ***Irradiance*** (Watts/m^2) = **E**
- The point on the surface might emit different energy in different directions so radiosity per direction:
  $\Rightarrow$ ***Radiance*** (Watts/m^2/sr) = **L**
- Radiance is usually of most interest in computer graphics
  - i.e. flux per area reflected towards the viewer

Radiometric Units

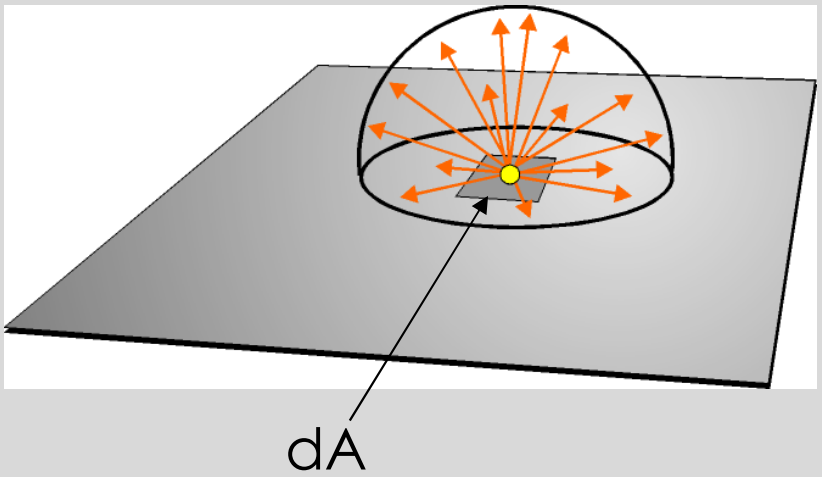
- The total *energy* (Joules) leaving a surface per unit time:
  - ⇒ *power* or ***Flux*** = Φ
  - ⇒ Has units of Power (Watts (W))



Radiometric Units

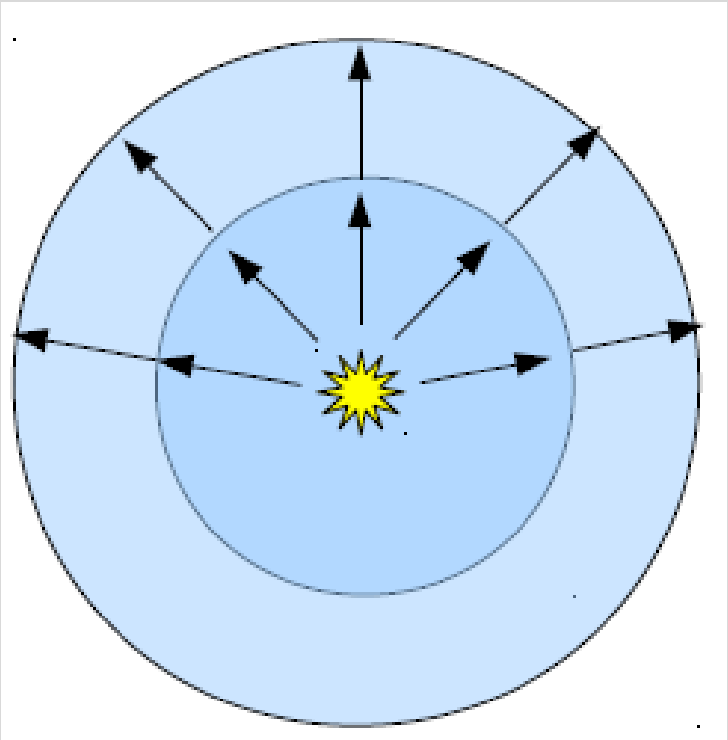
- The flux leaving a surface can change with position on the surface (i.e. some points are brighter), so flux per unit area:

  $\Rightarrow$ ***Radiosity*** (Watts/m^2) = **B**



Radiometric Units

- Flux arriving per unit area:
  $\Rightarrow$ ***Irradiance*** (Watts/m^2) = **E**



Irradiance at a point on the outer sphere is less than irradiance at a point on the inner one.

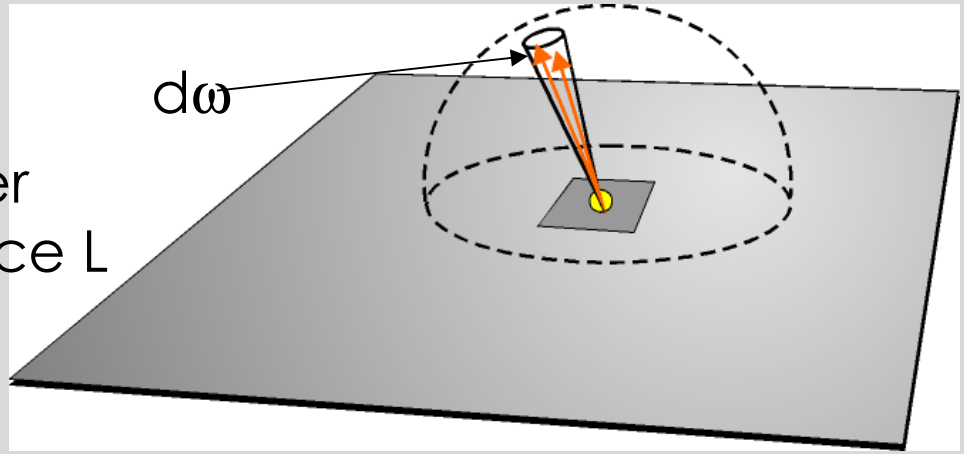
Energy received from a light source falls off with squared distance from it.

Think of it as a flux of photons bombarding the surface…

Radiometric Units

- The point on the surface might emit different energy in different directions so radiosity per direction:
  - ⇒ ***Radiance*** (Watts/m^2/sr) = **L**
  - ⇒ Remains constant along rays of light through empty space
- Radiance is usually of most interest in computer graphics
  - i.e. flux per area reflected towards the viewer

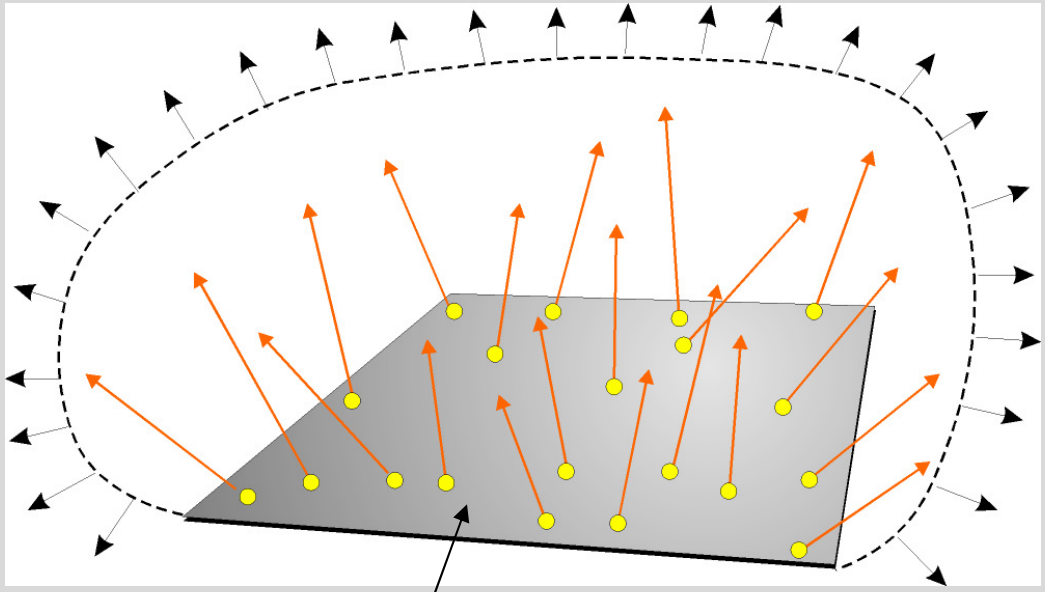
Flux per unit area per unit direction = Radiance L



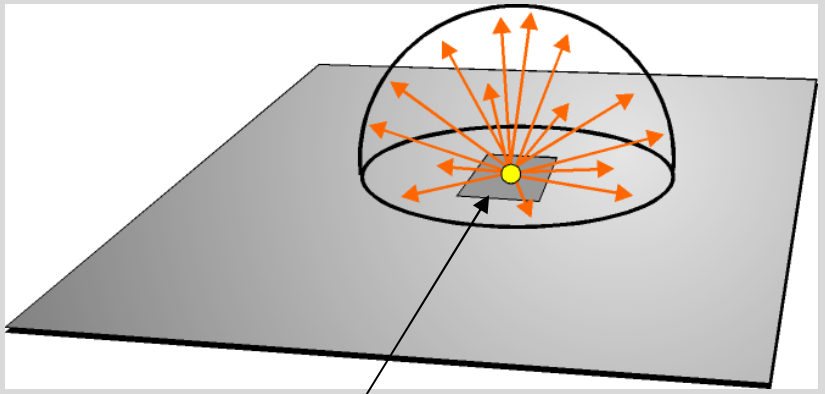
Typical Values of Luminance

- Surface of the sun 2,000,000,000
- Sunlight clouds 30,000
- Clear day 3,000
- Overcast day 300
- Street lighting 10

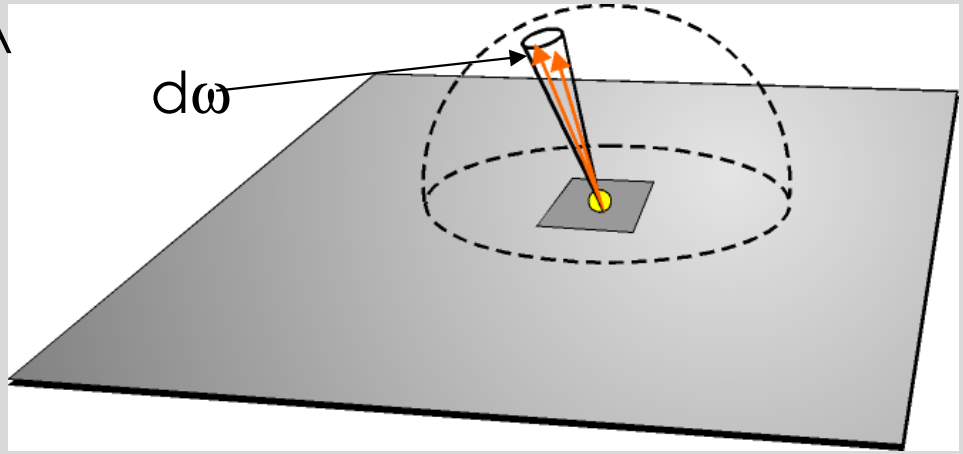
Radiometric Units – which?



Area = A



dA

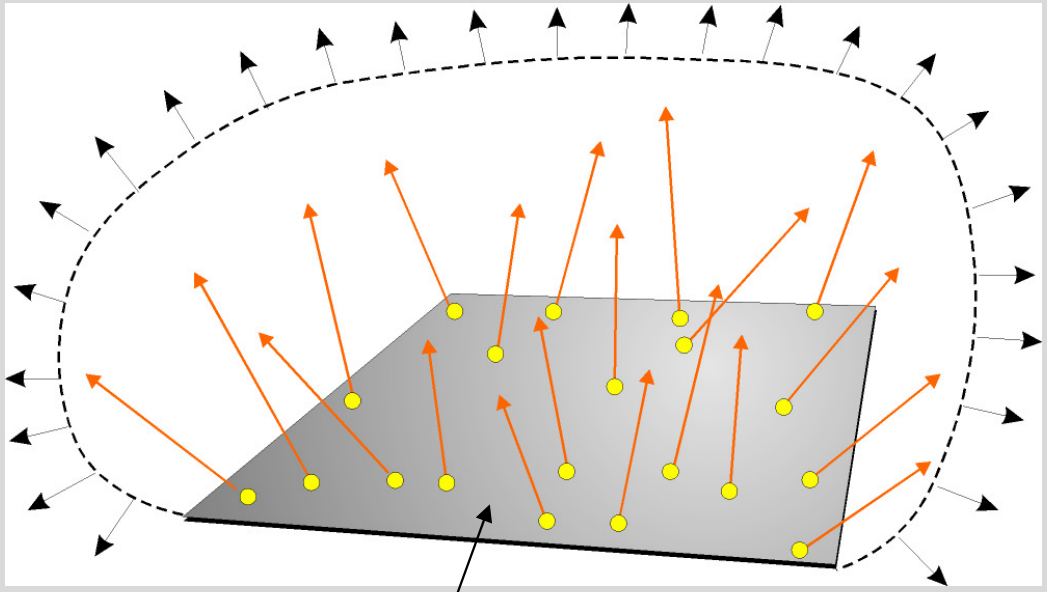


dω

Radiance?
Flux?
Radiosity?

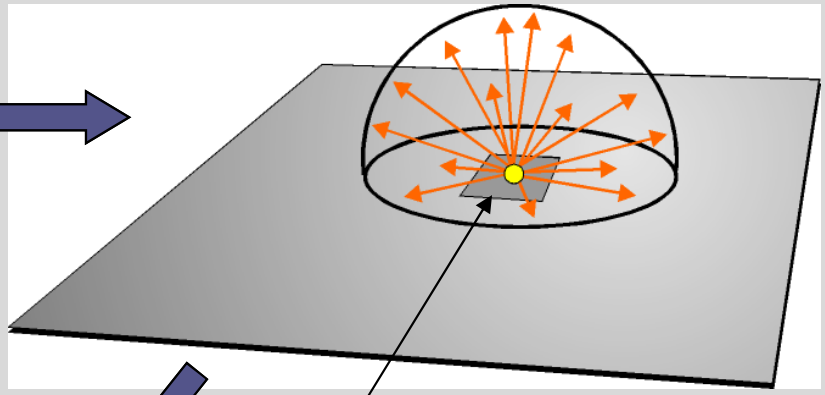
Radiometric Units

Total flux leaving surface = Φ



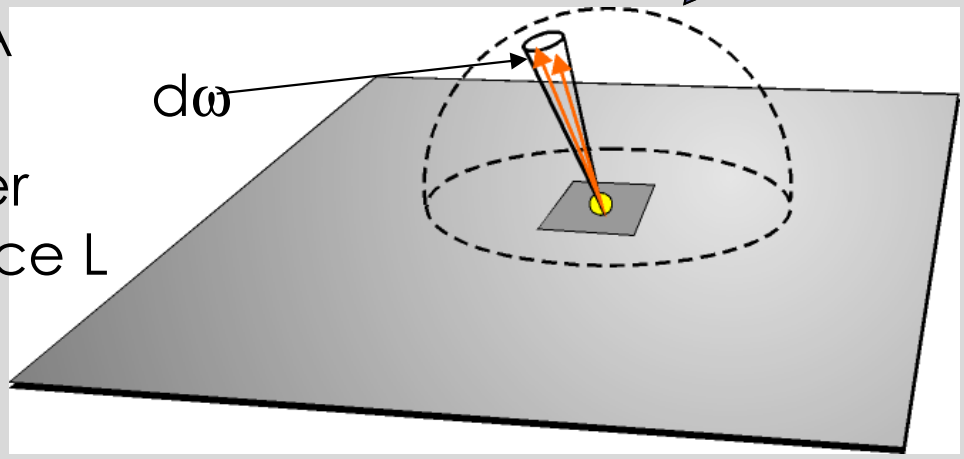
Area = A

Flux per unit area = Radiosity B



dA

Flux per unit area per unit direction = Radiance L



$d\omega$

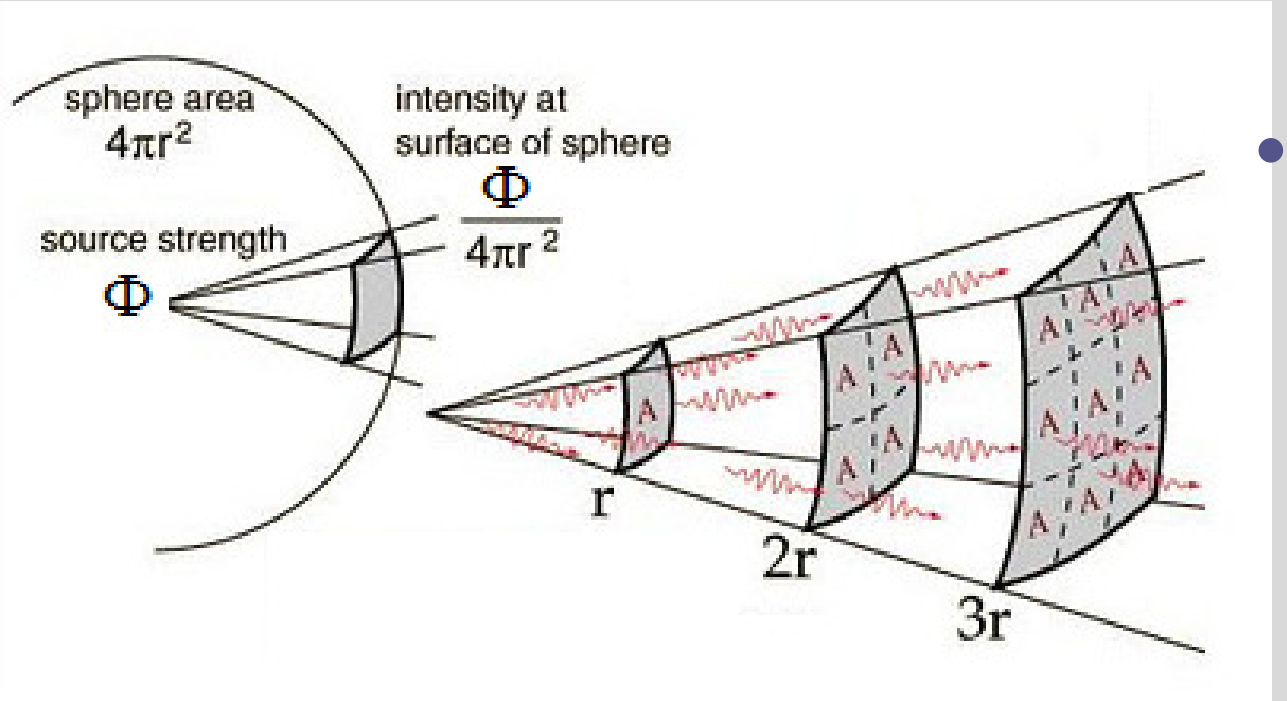
Inverse Square Law

The flux per unit area (irradiance *E*) at a point x at distance *r* from the point light source can now be calculated:

- we know the source radiates Φ Watts in all directions
- therefore this power is radiated through the sphere centred at the lightsource
- at a distance *r* from the source, the surface area of this sphere is $4\pi r^2$
- therefore the power per unit area at x is: $E = \frac{\Phi}{4\pi r^2}$

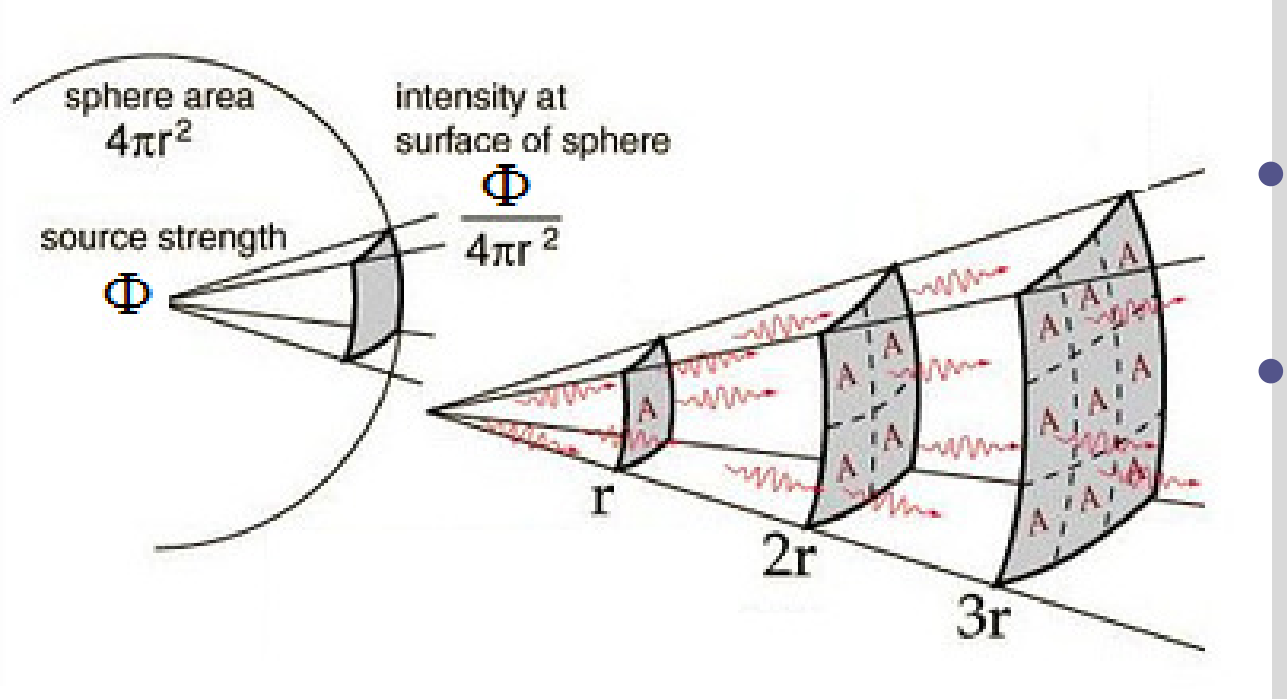
- note: this assumes the surface at x is perpendicular to the direction towards the light source.

Inverse Square Law



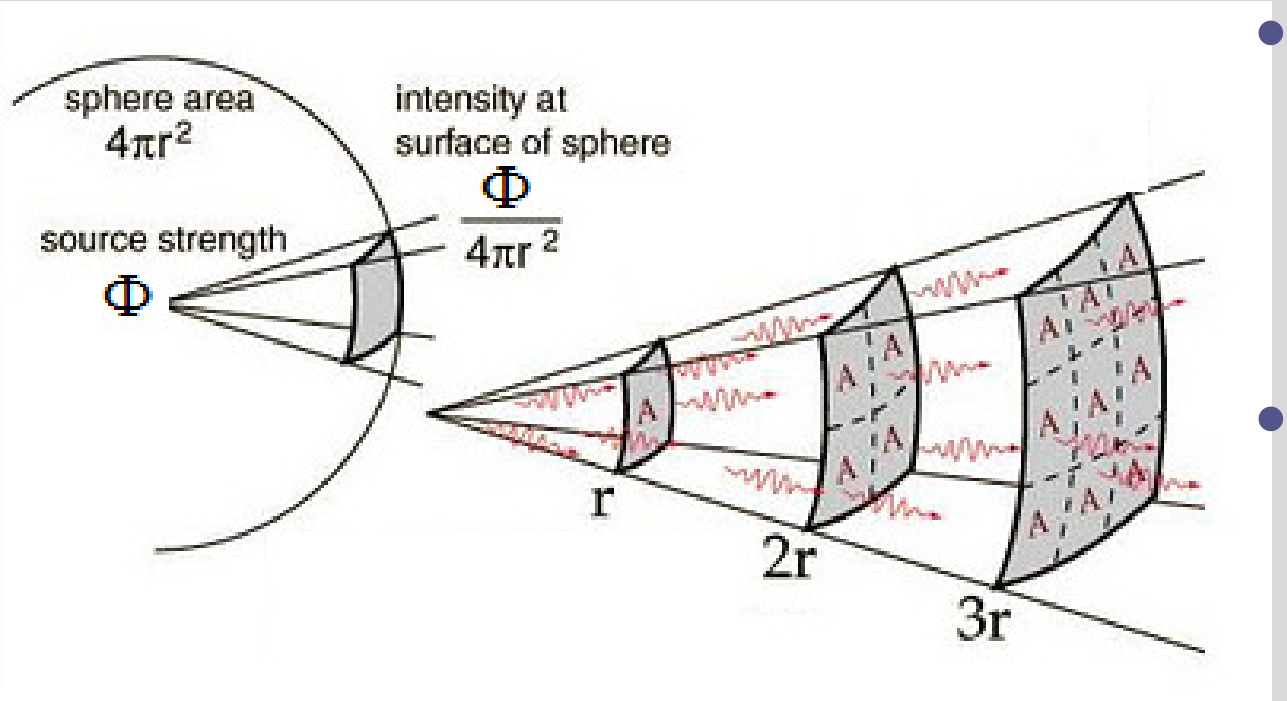
- An object that is twice the distance from a point source of light will receive a quarter of the illumination

Problem



- **50W spherical blub of diameter 20cm**
- Calculate irradiance on the bulb surface
- Calculate irradiance arriving on a surface 10cm away

Solution



- 50W spherical blub of diameter 20cm
- Calculate irradiance on the bulb surface
  - 50 / 4 * 3.14 * 0.1^2 =
  - 398 W/m^2
- Calculate irradiance arriving on a surface 10cm away
  - 50 / 4 * 3.14 * 0.2^2 =
  - 99.5 W/m^2
  - Quarter the illumination

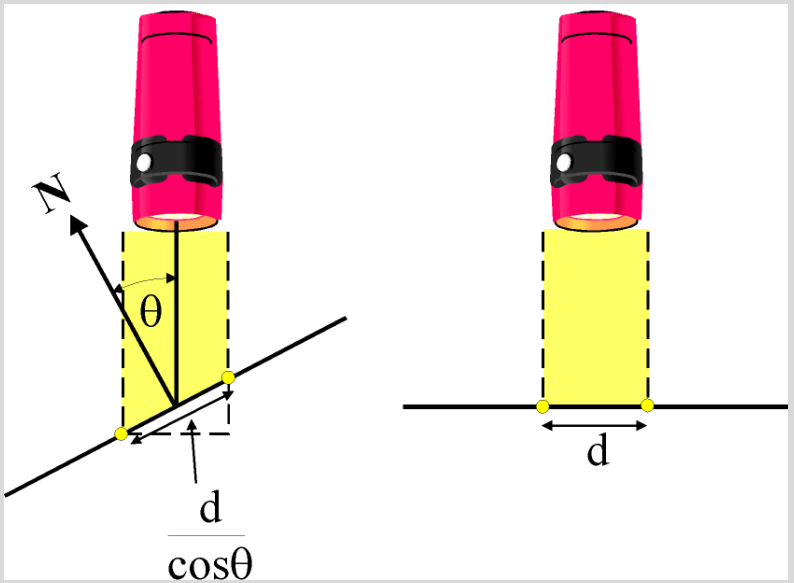
The Cosine Rule

- A surface which is oriented perpendicular to a light source will receive more energy per unit area (and thus appear brighter) than a surface oriented at an angle to the light source.
- The irradiance *E* is proportional to $\frac{1}{area}$
- As the area increases, the irradiance decreases therefore:

$$E = \frac{\cos\theta\,\Phi}{4\pi r^2}$$

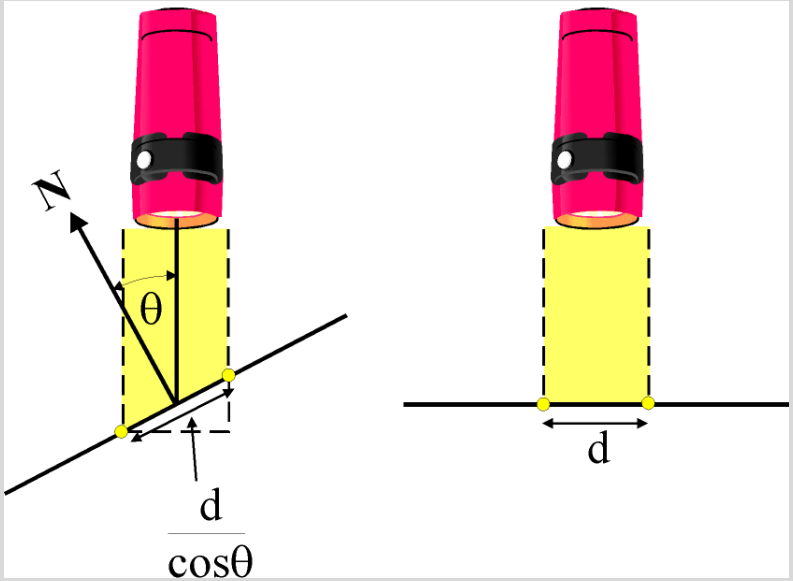
θ	Cos(θ)
0	1.00
10	0.98
30	0.87
50	0.64
70	0.34
90	0.00
110	-0.34
130	-0.64

As θ increases, the irradiance and thus the brightness of a surface decreases by cos θ



Problem

- 50W point light source
- Surface located 40cm away
- Calculate irradiance when surface is tilted 30 degrees
- = 0.87 * 50 / 4 * 3.14 * 0.2^2
- 8.65 W/m^2



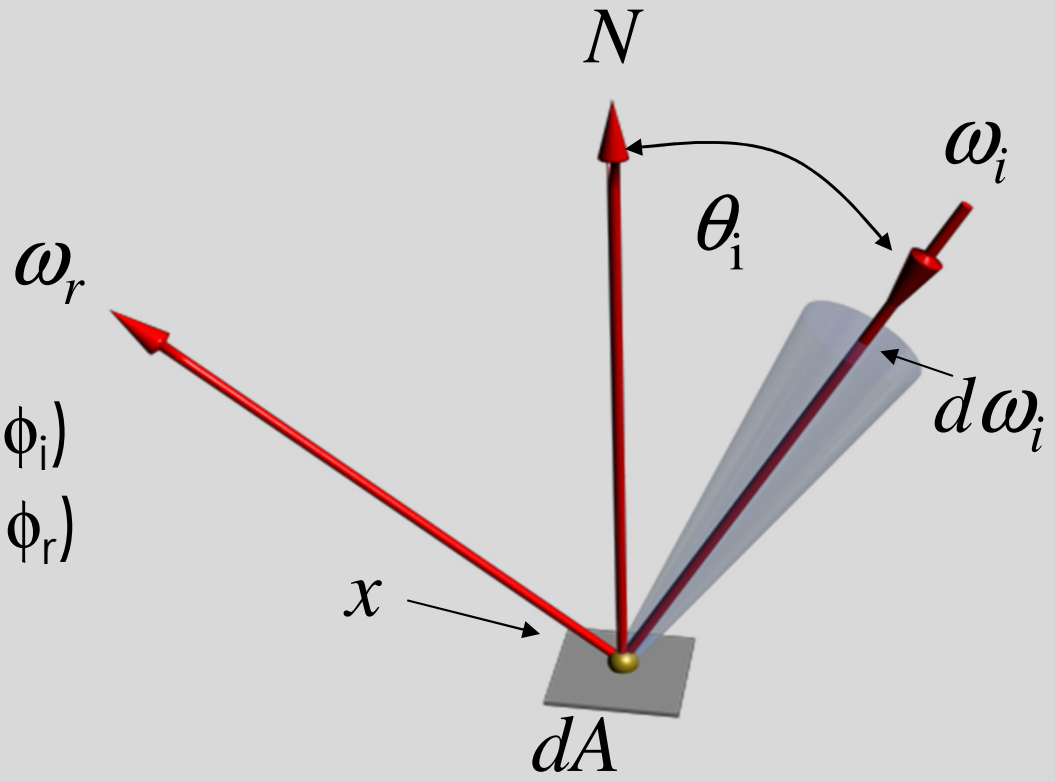
$$E = \frac{\cos\theta\,\Phi}{4\pi r^2}$$

Overview

- Rendering algorithms (local, global, view dependent, view independent)
- Light, colour, spectra
- Surface reflectance
- Solid angle, radiometric units, radiance
- Inverse square law, the cosine rule
- **BRDF, BRDF approximations**
- **Reflectance equation, radiance equation**
- Light sources, Lambertian illumination model

Bi-directional Reflectance Distribution Function (BRDF)

- Describes reflected radiance given incident radiance.
- Theoretically, a high dimension function:
  - position = x
  - incoming direction = $\omega_i = (\theta_i, \phi_i)$
  - reflected direction = $\omega_r = (\theta_r, \phi_r)$
  - wavelength = λ

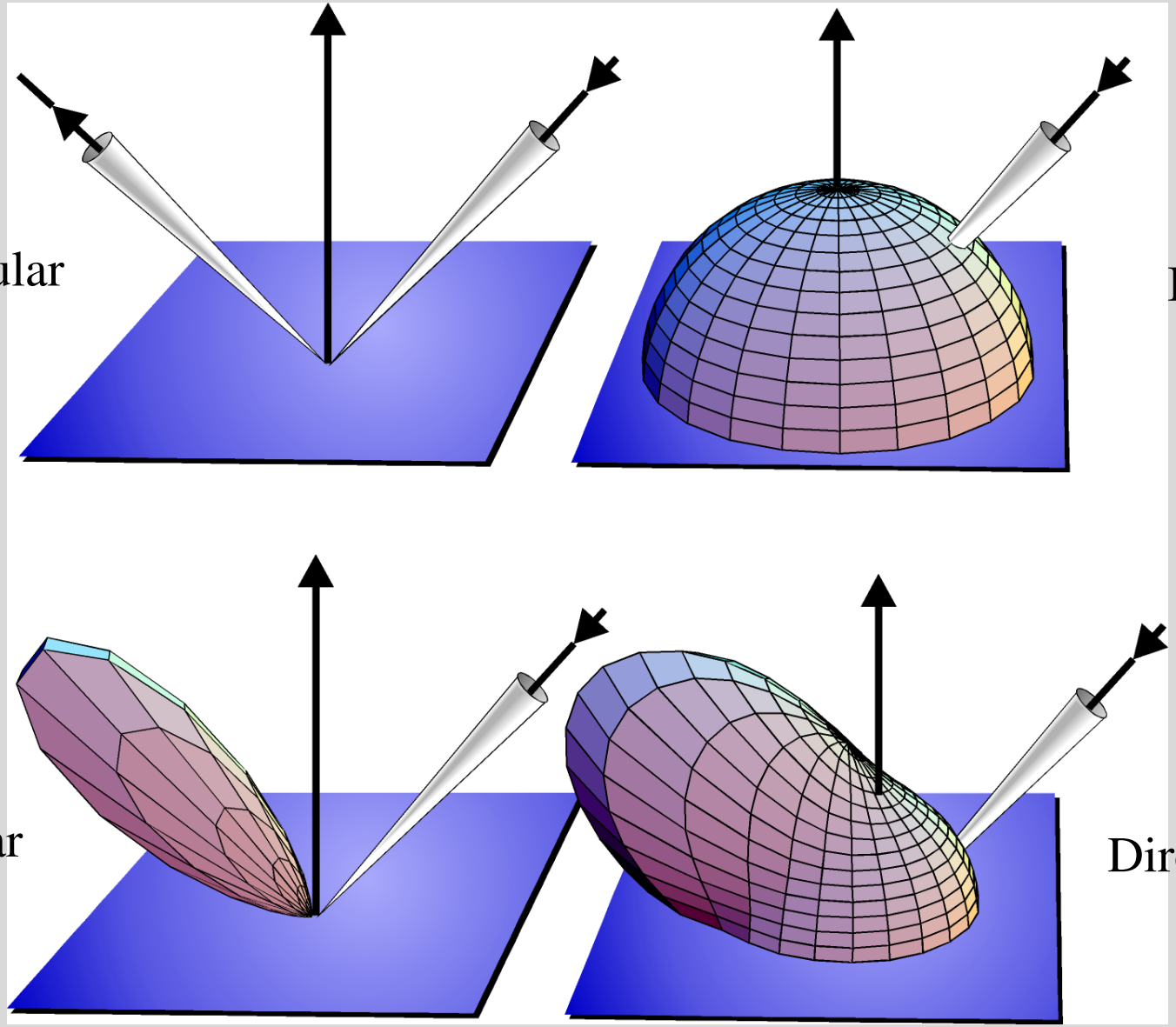


$$f_r(x, \omega_i, \omega_r) = \frac{dL_r(x, \omega_r)}{E_i(x, \omega_i)\cos\theta_i d\omega_i}$$

L = Radiance
E = Irradiance

BRDF Approximations

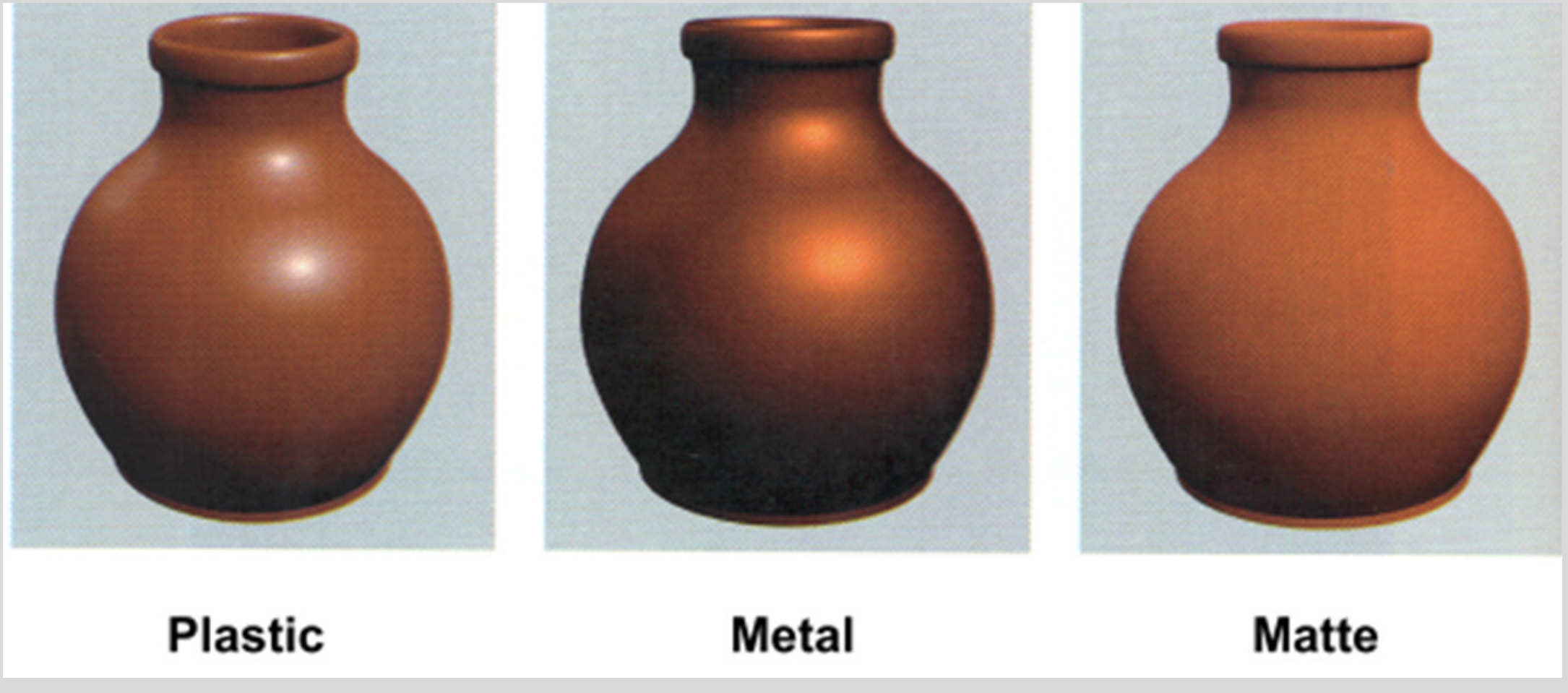
Ideal specular



Ideal diffuse

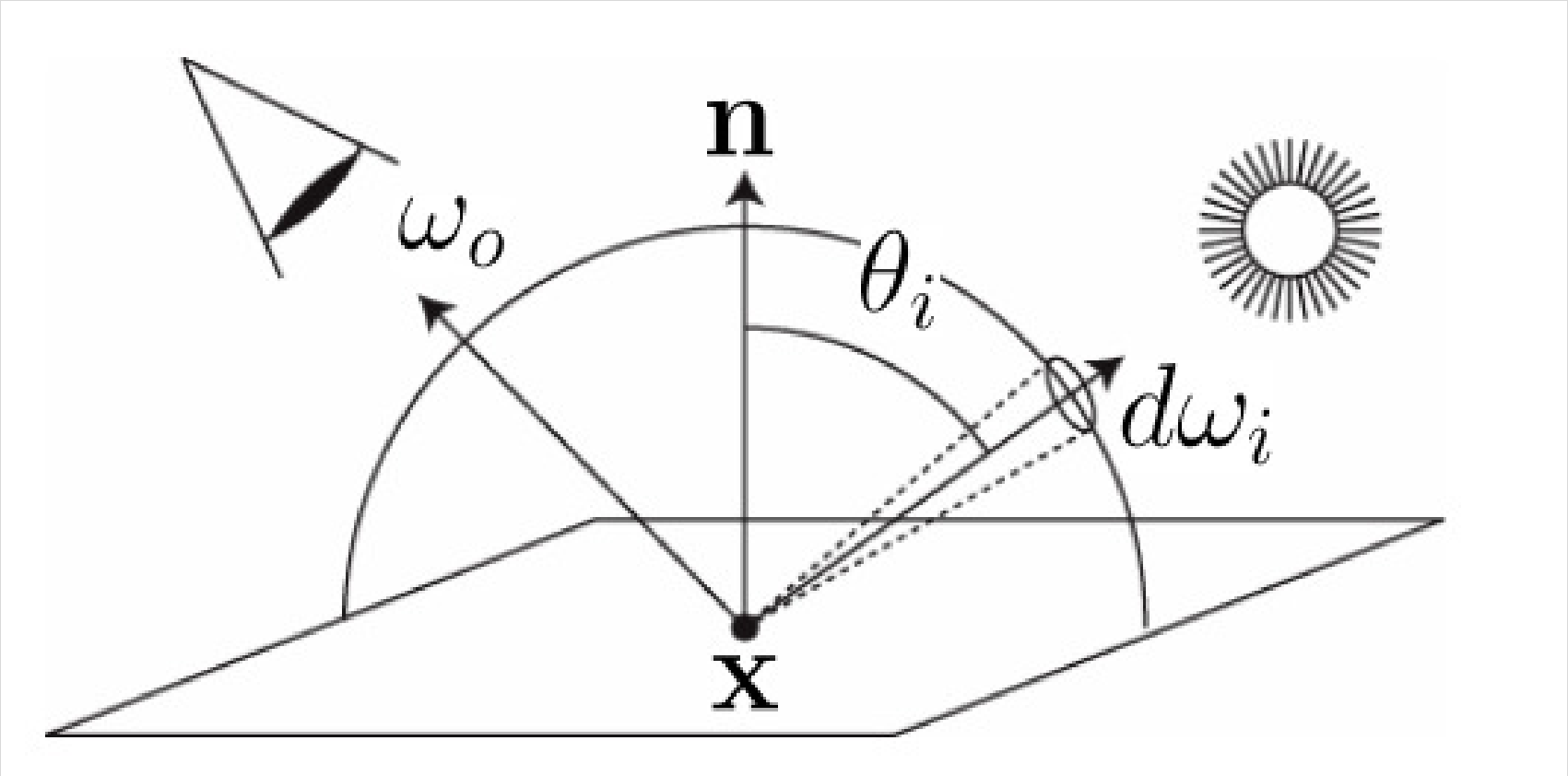
Rough specular

Directional diffuse


Plastic
Metal
Matte

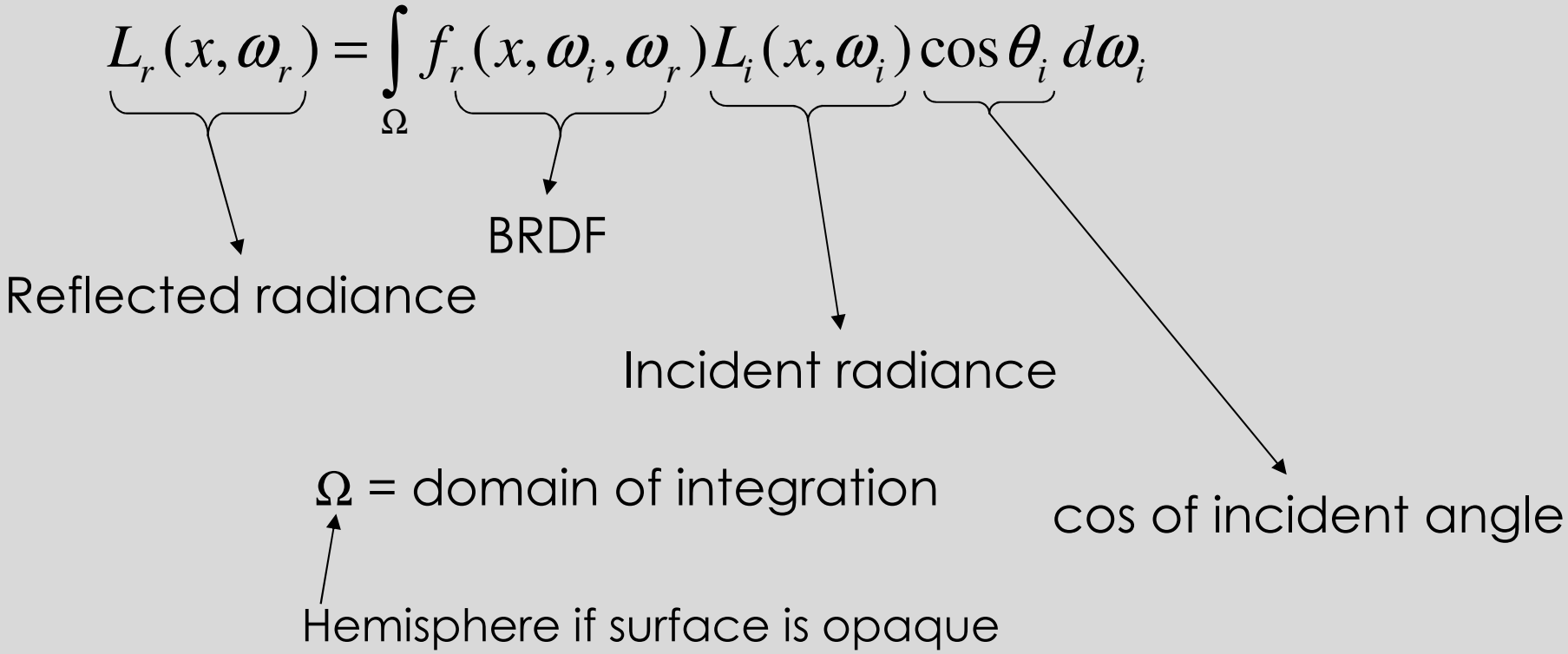
Reflectance Equation

- The *reflectance equation* relates reflected radiance to incoming radiance that is scattered according to the surface's BRDF:



Reflectance Equation

- The *reflectance equation* relates reflected radiance to incoming radiance that is scattered according to the surface's BRDF:



Radiance Equation

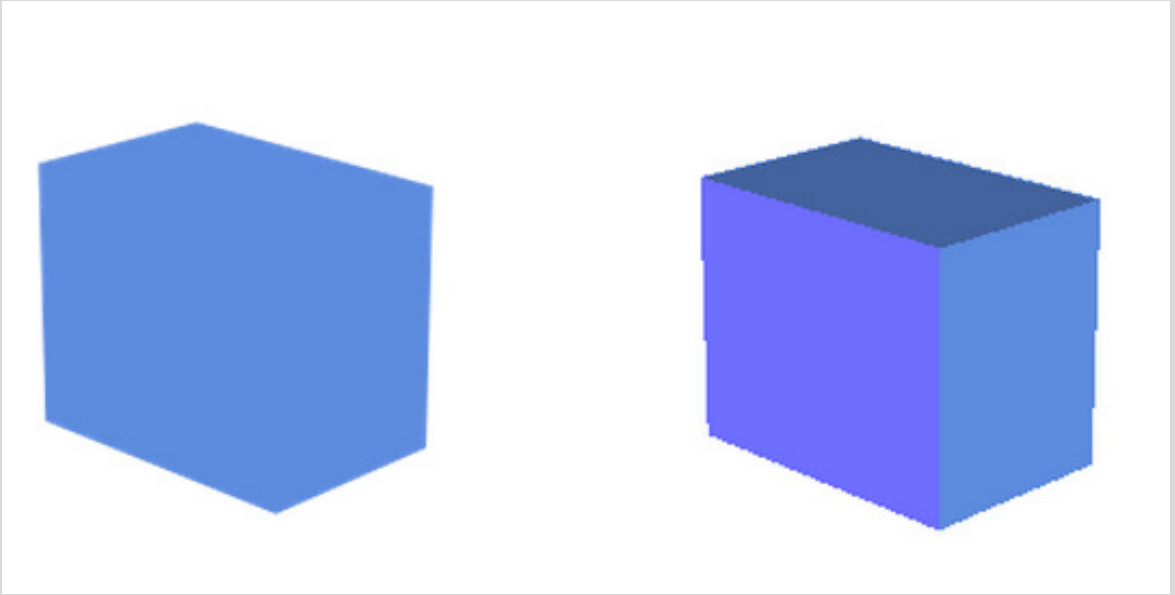
- The radiance equation includes a *self-emitted term* to account for light sources.
- This is the most important equation in rendering theory.
- We solve for $L_r(x, \omega_r)$ at each visible point in the scene.

$$L_r(x, \omega_r) = L_e(x, \omega_r) + \int_{\Omega} f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) \cos\theta_i \, d\omega_i$$

Self emitted radiance
(non zero for light sources)

Linear Fredholm Integral of the 2nd kind

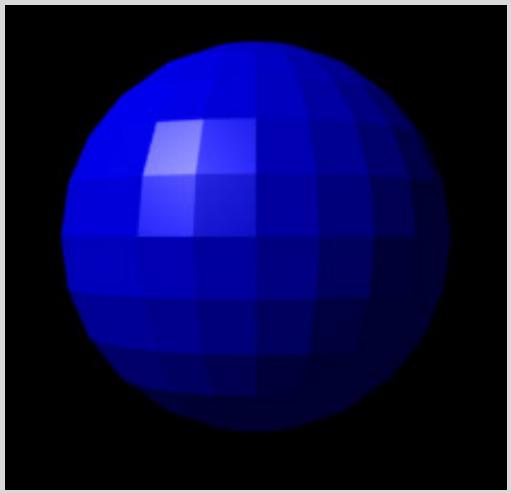
Shading



- Flat
  - Compute illumination once per polygon and apply it to whole polygon (per vertex with no interpolation)
- Smooth/Gouraud shading
  - Compute illumination at borders and interpolate (per vertex)
- More accurate/Phong shading
  - Compute illumination at every point of the polygon (per fragment)

Flat Shading

- Illumination model is applied only once per polygon
- Gives low-polygon models a faceted look.
- Works poorly if the model represents a curved surface.
- Smooth appearance implies large number of polygons.
- Adding more facets helps but… slows down the rendering.
- Advantageous in modeling boxy objects.
- Effectiveness is tempered by “Mach banding”.



Mach Banding

- Optical illusion named after physicist Ernst Mach
- Perceived intensity change at edges are exaggerated by receptors in our eyes, making the dark facet look darker and the light facet look lighter.

Mach Banding

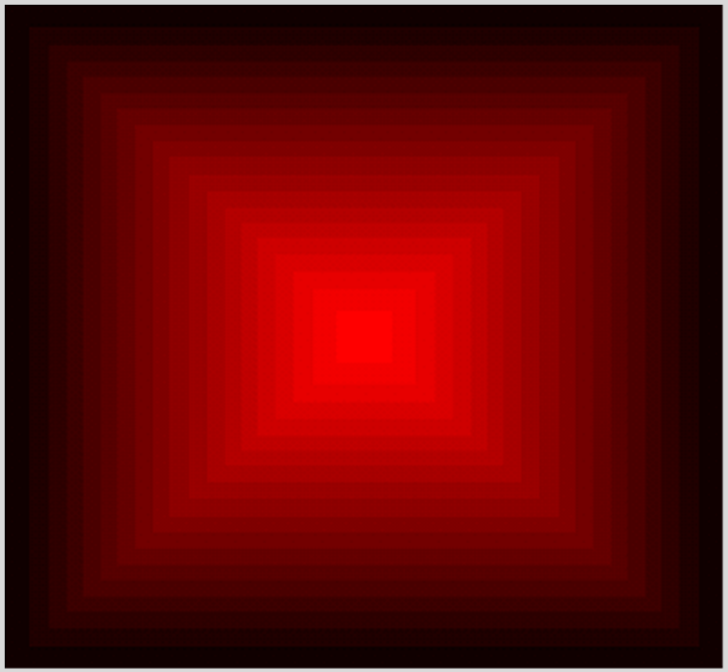
- Brightness perceived by the eye tends to overshoot at the boundaries of regions of constant intensity
- Abrupt changes in the shading of two adjacent polygons are perceived to be even greater

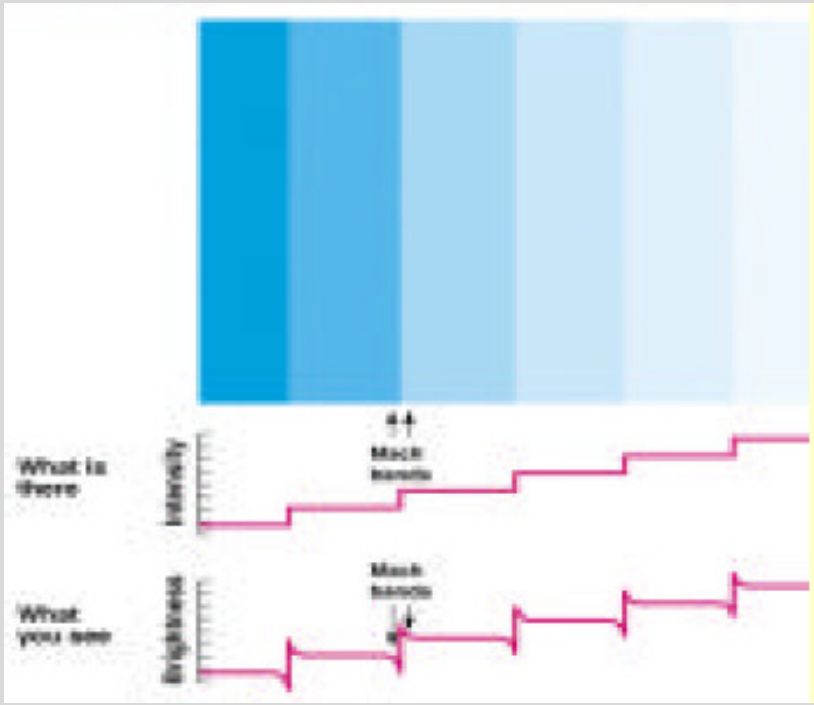
Mach Band Effect

- These "Mach Bands" are not physically there. Instead, they are illusions due to excitation and inhibition in our neural processing

The bright bands at 45 degrees (and 135 degrees) are illusory.

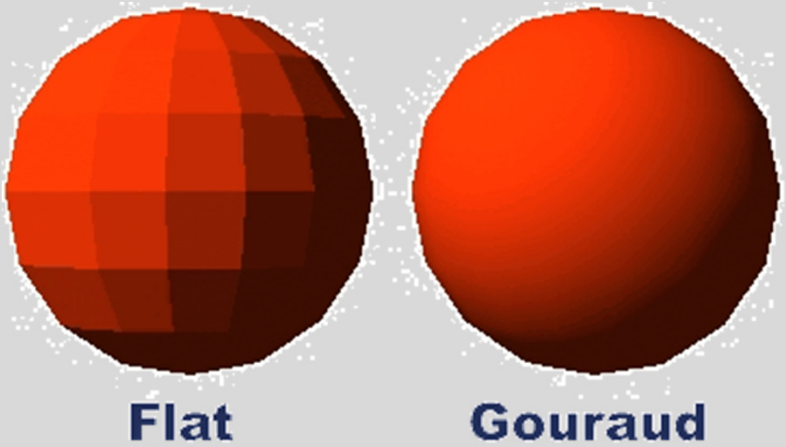
The intensity of each square is the same.





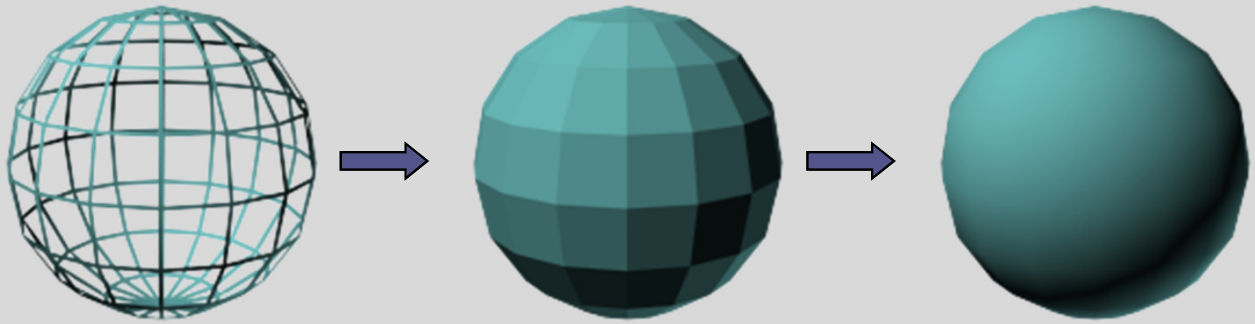
Smooth Shading

- Used to approximate curved surfaces with a collection of polygons.
- Calculate illumination based on approximation of curved surface.
- Does not change geometry, silhouette is still polygonal.



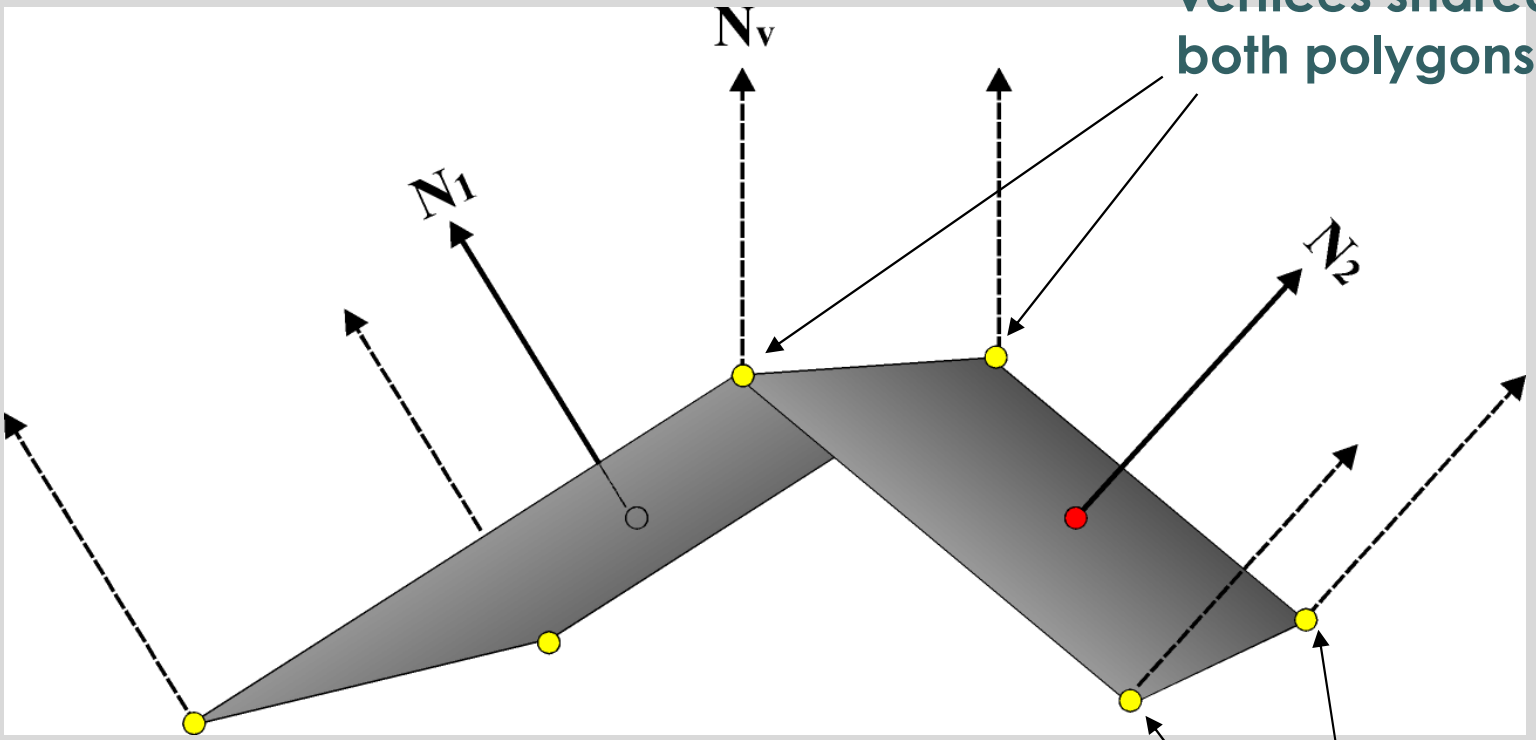
Surface Normal Interpolation

- Often we are approximating curved/smooth surfaces with polygons ⇒ we get *edge artifacts* at polygon boundaries:



- To combat this we determine average normals at the vertices of the polygons by averaging the normals of each polygon that shares a vertex and storing the result with that vertex.

Determining Vertex Normals



$$N_v = \frac{\frac{1}{2}(N_1 + N_2)}{\left|\frac{1}{2}(N_1 + N_2)\right|}$$

Gouraud Shading

- Gouraud shading is a method for linearly interpolating a colour or shade across a polygon.
- It was invented by Henri Gouraud in 1971.
- It is a very simple and effective method of adding a curved feel to a polygon that would otherwise appear flat.

Gouraud Shading

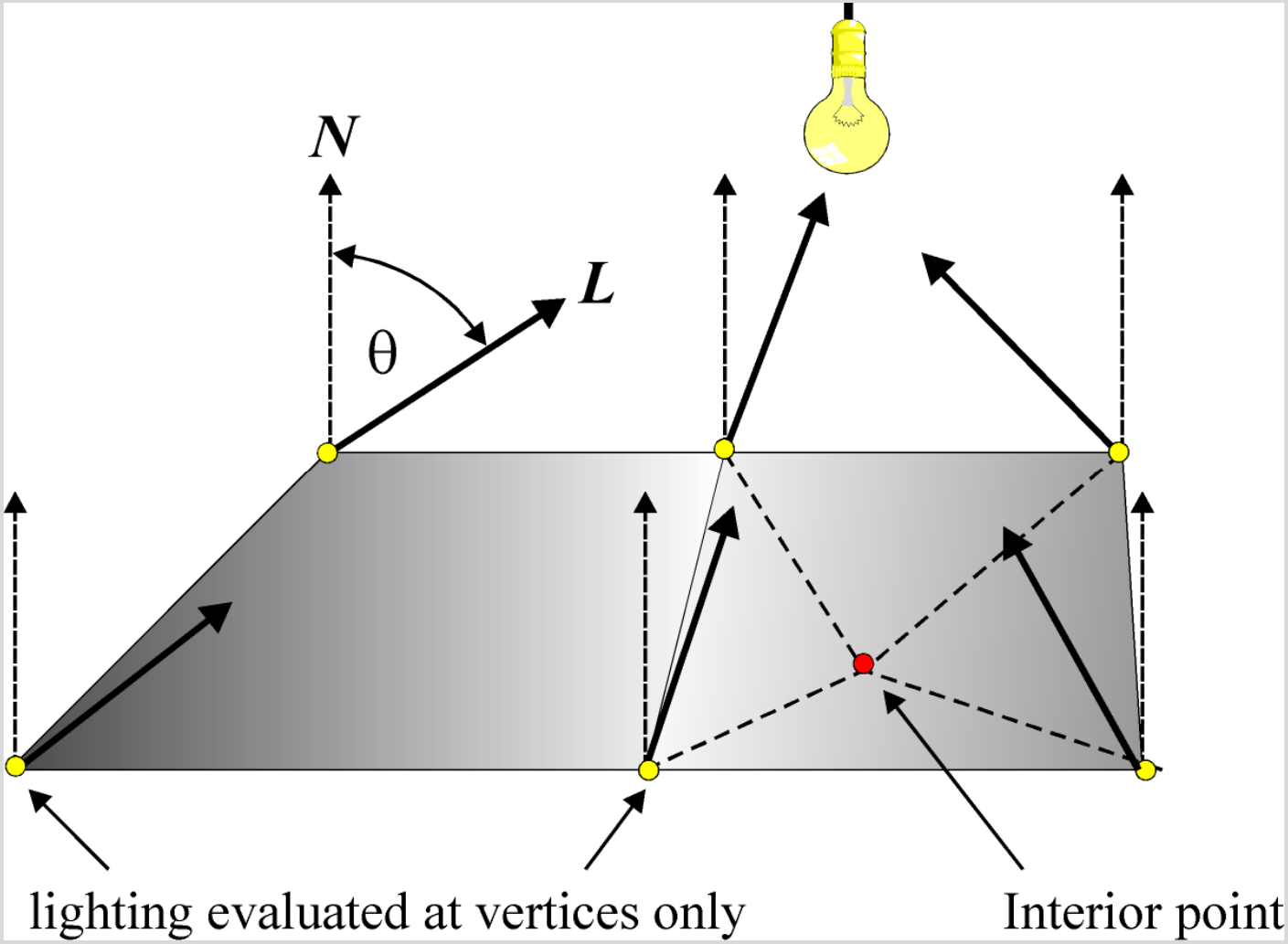
- For each interior point in the polygon being shaded we interpolate the intensity determined at the vertices.
- We do this in exactly the same way that we interpolated colour across the surface of a polygon.
- This is known as *Gouraud shading*.

$\Rightarrow$we need to do lighting calculations at vertices only

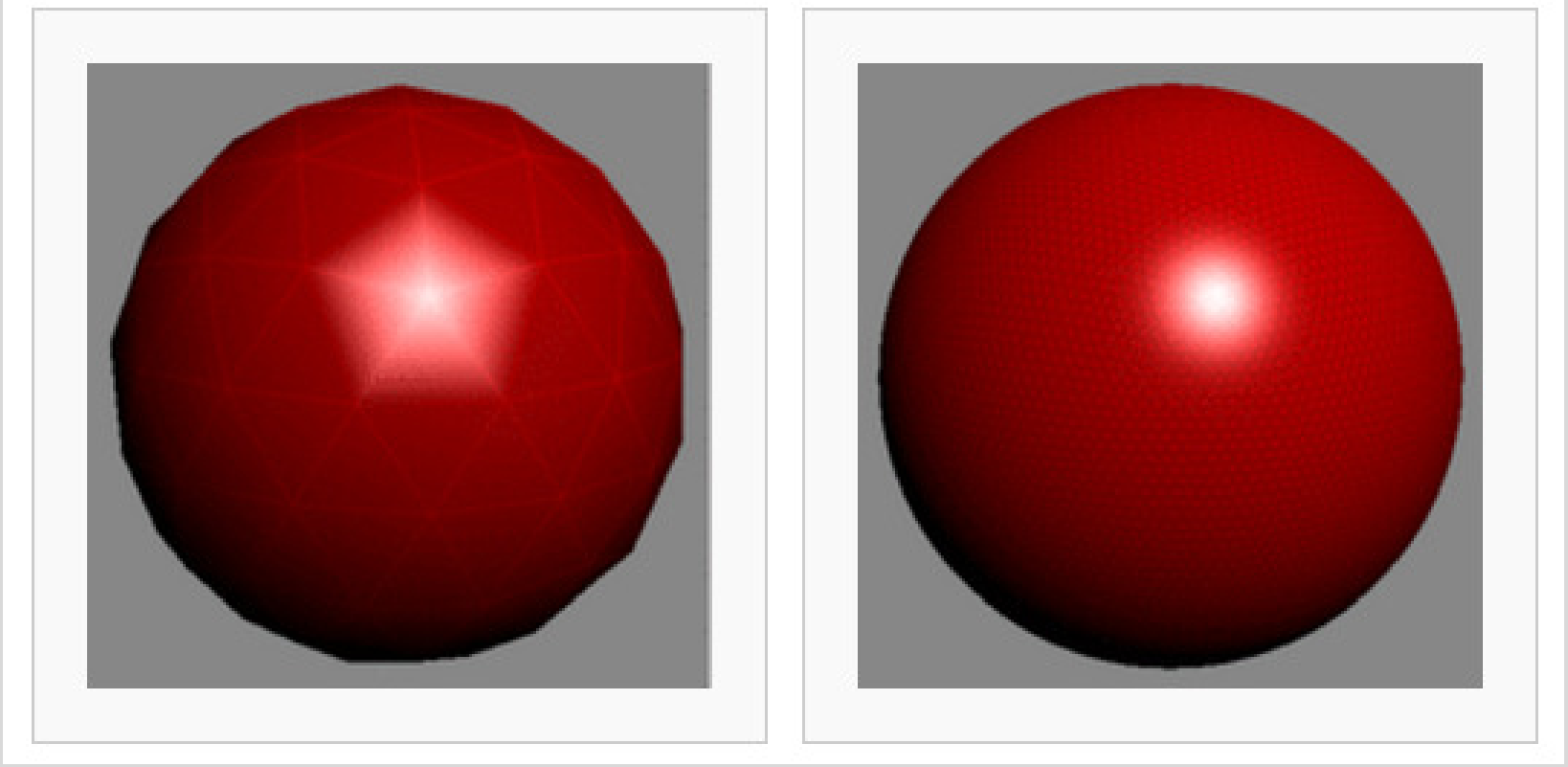
$\Rightarrow$lighting is correct at vertices only

- This is OK if the polygons are small. As polygons increase the errors also increase leading to often unacceptable results.

Gouraud Shading



Interpolation Errors



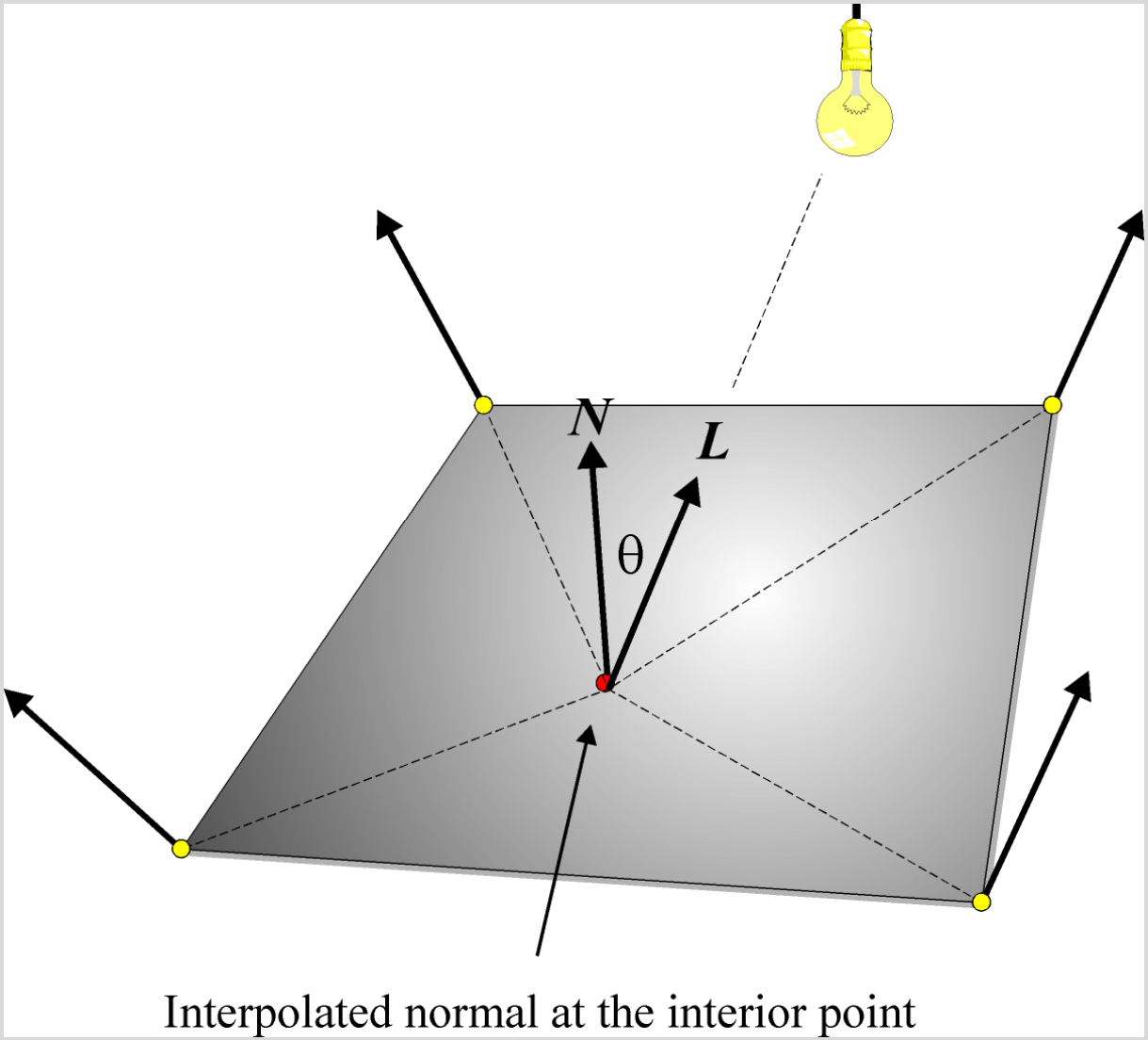
Poor behaviour of the specular highlight

Improvement with very high polygon count

Phong Shading

- To improve upon Gouraud shading we can *interpolate the normals* across the surface and apply the lighting model at each point in the interior.
- This assumes we are working with polygonal models.
- Care must be taken to ensure that all interpolated normals are of unit length before employing the lighting model.
- This is known as *Phong shading* (as opposed to the *Phong illumination model*).

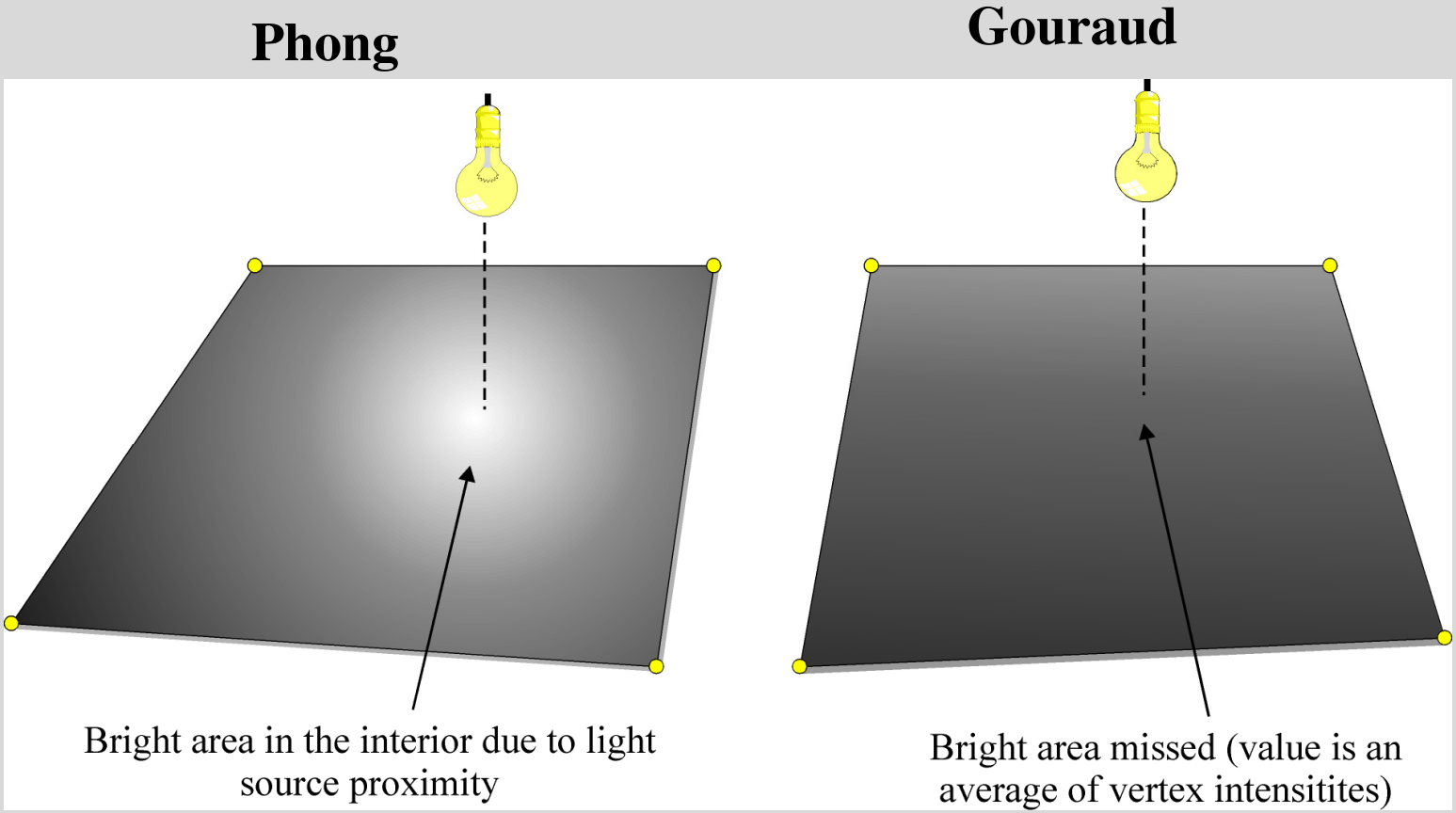
Phong Shading



Interpolated normal at the interior point

Phong Shading

- Phong shading is capable of reproducing *highlights* within the interior of a polygon that Gouraud shading will miss:



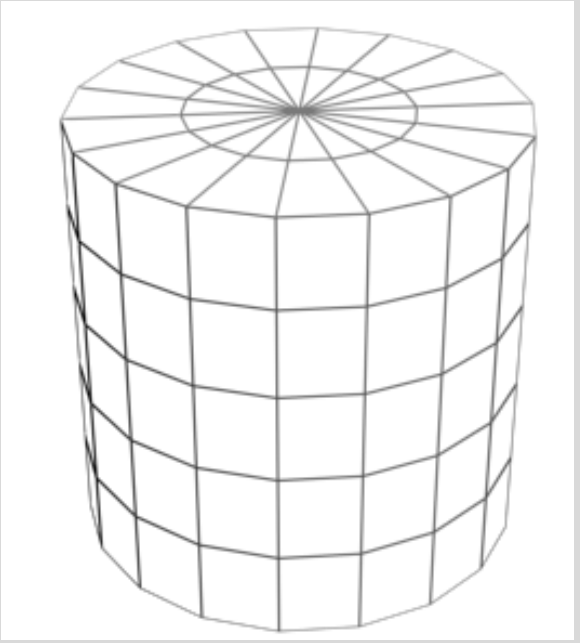
Phong vs Gouraud

- Phong shading
  - Handles specular highlights much better
  - Does a better job in handling Mach bands
  - But more expensive than Gouraud shading

Which shading was used?

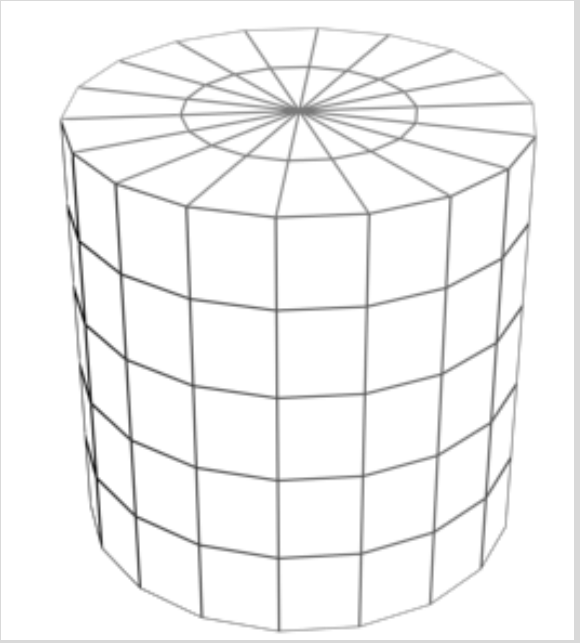
Problem

- Your friend attempts to render a shiny cylinder with Gouraud shading. His drawing code specifies a shiny material, places vertices along the upper and lower rims of the cylinder, and connects them with long, thin triangles.
- Given the position of the light, he expects to see a specular highlight in the center of the cylinder, but no highlight is present in the images generated by his renderer. Explain why this is the case, and suggest two different ways to fix the problem.



Solution

- Specular highlights that do not overlap any vertex will not be visible on meshes rendered with Gouraud shading
  - Re-tesselate the cylinder so that some vertices lie within the specular highlight
  - Switch to per-pixel Phong shading



Overview

- Solid angle, radiometric units, radiance
- Inverse square law, the cosine rule
- BRDF, BRDF approximations
- Reflectance equation, radiance equation
- Shading Models
- Illumination Models
- Practical Implementation
- Light source approximations

Shading model vs illumination model

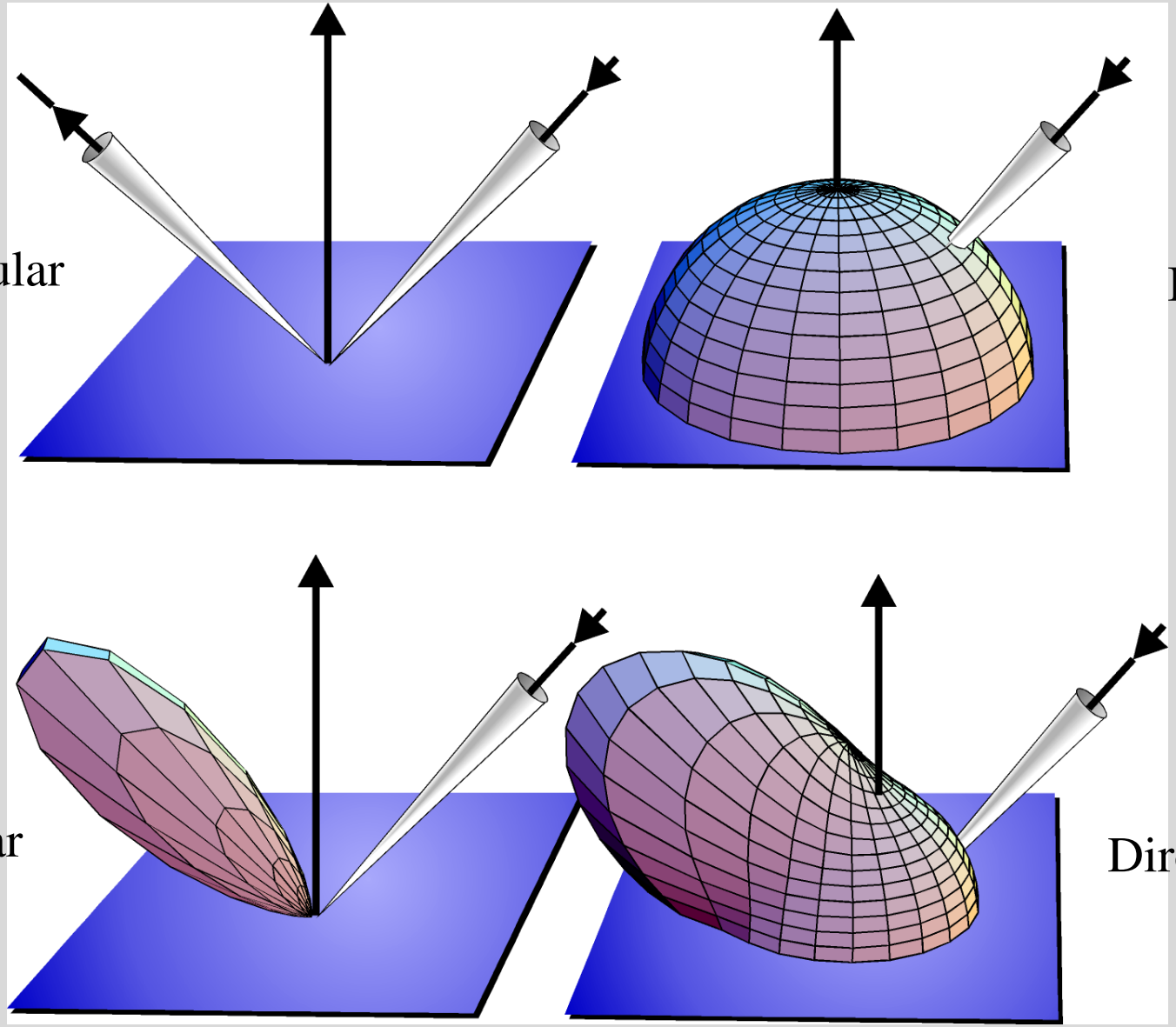
- There is a difference between the shading model and the illumination model used in rendering scenes,
  - the illumination model captures how light sources interact with object surfaces, and
  - the shading model determines how to render the faces of each polygon in the scene.
- The shading model depends on illumination model, for example
  - some shading models invoke an illumination model for every pixel (such as ray tracing),
  - others only use the illumination model for some pixels and then shade the remaining pixels by interpolation (such as Gouraud shading).

Shading model vs illumination model

- The illumination model is about determining how light sources interact with object surfaces
- Whereas the shading model is about how to interpolate over the faces of polygons, given the illumination.

BRDF Approximations

Ideal specular



Ideal diffuse

Rough specular

Directional diffuse

Reflectance Equation

- The *reflectance equation* relates reflected radiance to incoming radiance that is scattered according to the surface's BRDF:

$$\underbrace{L_r(x,\omega_r)}_{\text{Reflected radiance}} = \int_{\Omega} f_r(x,\omega_i,\omega_r) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

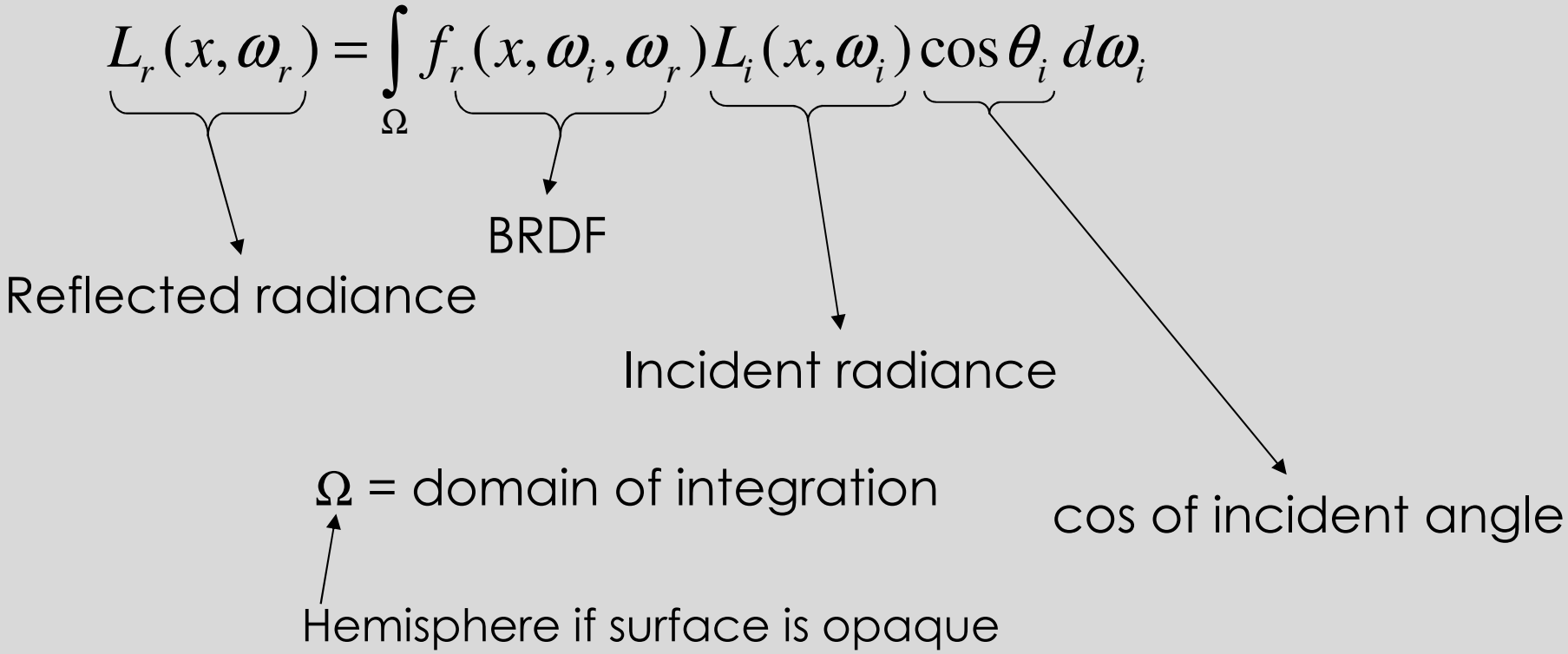
Reflectance Equation

- The *reflectance equation* relates reflected radiance to incoming radiance that is scattered according to the surface's BRDF:

$$\underbrace{L_r(x,\omega_r)}_{\text{Reflected radiance}} = \int_\Omega \underbrace{f_r(x,\omega_i,\omega_r)}_{\text{BRDF}} L_i(x,\omega_i)\cos\theta_i \, d\omega_i$$

Reflectance Equation

- The *reflectance equation* relates reflected radiance to incoming radiance that is scattered according to the surface's BRDF:



Radiance Equation

- The radiance equation includes a *self-emitted term* to account for light sources.
- This is the most important equation in rendering theory.
- We solve for $L_r(x, \omega_r)$ at each visible point in the scene.

$$L_r(x, \omega_r) = L_e(x, \omega_r) + \int_{\Omega} f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) \cos\theta_i \, d\omega_i$$

Self emitted radiance
(non zero for light sources)

Linear Fredholm Integral of the 2nd kind

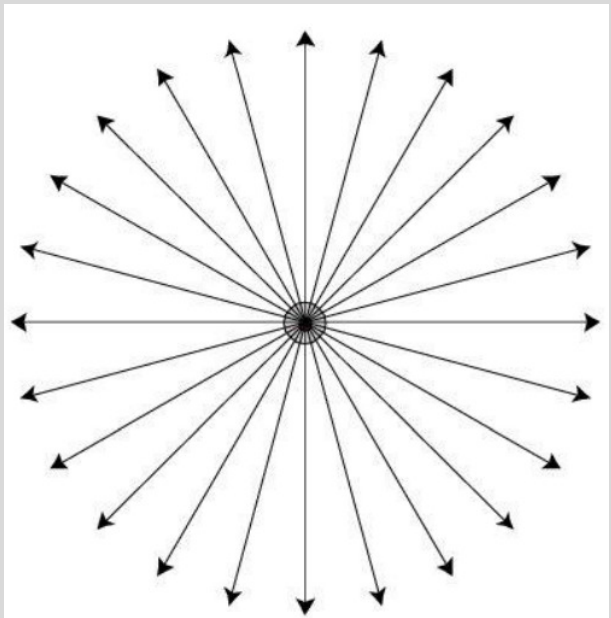
Lighting Equation Approximations

$$L_r(x,\omega_r) = \int_{\Omega} f_r(x,\omega_i,\omega_r) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

- The Rendering equation: no general solution.
- We need to make certain assumptions to solve the lighting problem. This gives rise to different illumination models.
- Factors that influence lighting: intensity of light, visibility, BRDF
- Light-material interactions: specular reflection, diffuse reflection, transmission

Light Sources

- To simplify the solution to the radiance equation we normally employ ***isotropic point light sources*** defined by:
  - position
  - colour
- *Isotropic* $\Rightarrow$ radiates energy equally in all directions.



Light Sources

- Normally we wish to associate a radiance with a light source but the definition of radiance assumes an area over which energy is emitted/distributed
  - but *point sources have no area*

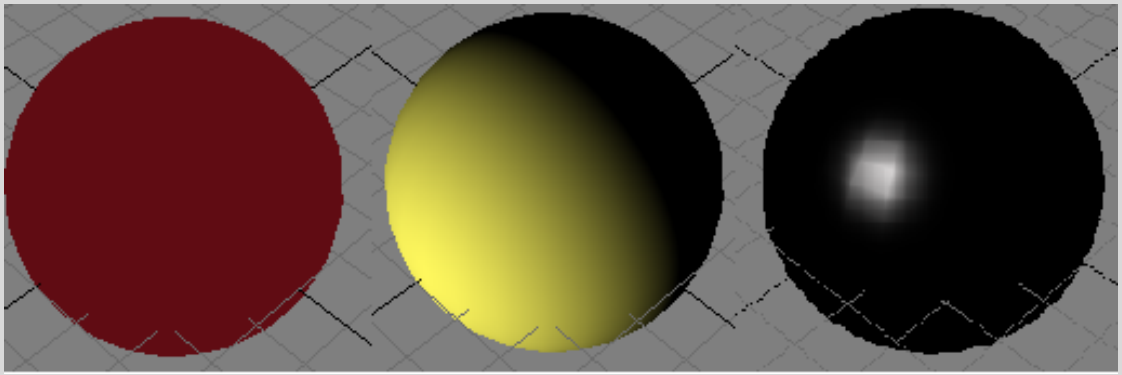
⇒Use radiant intensity *I* instead (units Watts/sr).

⇒A point source radiating energy *in all directions equally* has a radiant intensity of:

$$I = \frac{\Phi}{\omega} = \frac{\Phi}{4\pi}$$

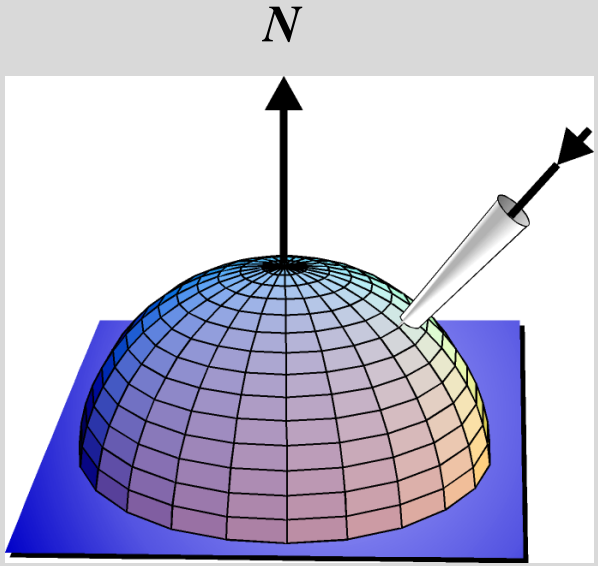
Illumination Models

- Ambient
- Lambertian
- Phong

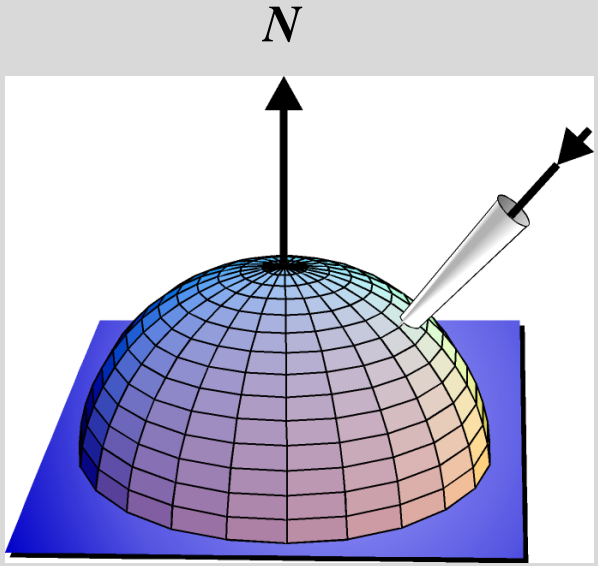


Diffuse Light

- The illumination that a surface receives from a light source and reflects equally in all directions
- This type of reflection is called Lambertian Reflection
- The brightness of the surface is independent of the observer position (since the light is reflected in all direction equally)



Lambertian Illumination Model



- We can use the cosine rule to implement shading of *Lambertian* or *diffuse* surfaces.
- Diffuse surfaces reflect light in all directions equally:
  - BRDF is a constant with respect to reflected direction
  - surface may be characterized by a *reflectance* ρ_d rather than a BRDF:

$$f_r(x) = \frac{\rho_d(x)}{\pi}$$

With unit reflectivity

  - the reflectance gives the ratio of the total reflected power to the total incident power:

$$\rho_d(x) = \frac{\Phi_i}{\Phi_r}$$

Reflectivity

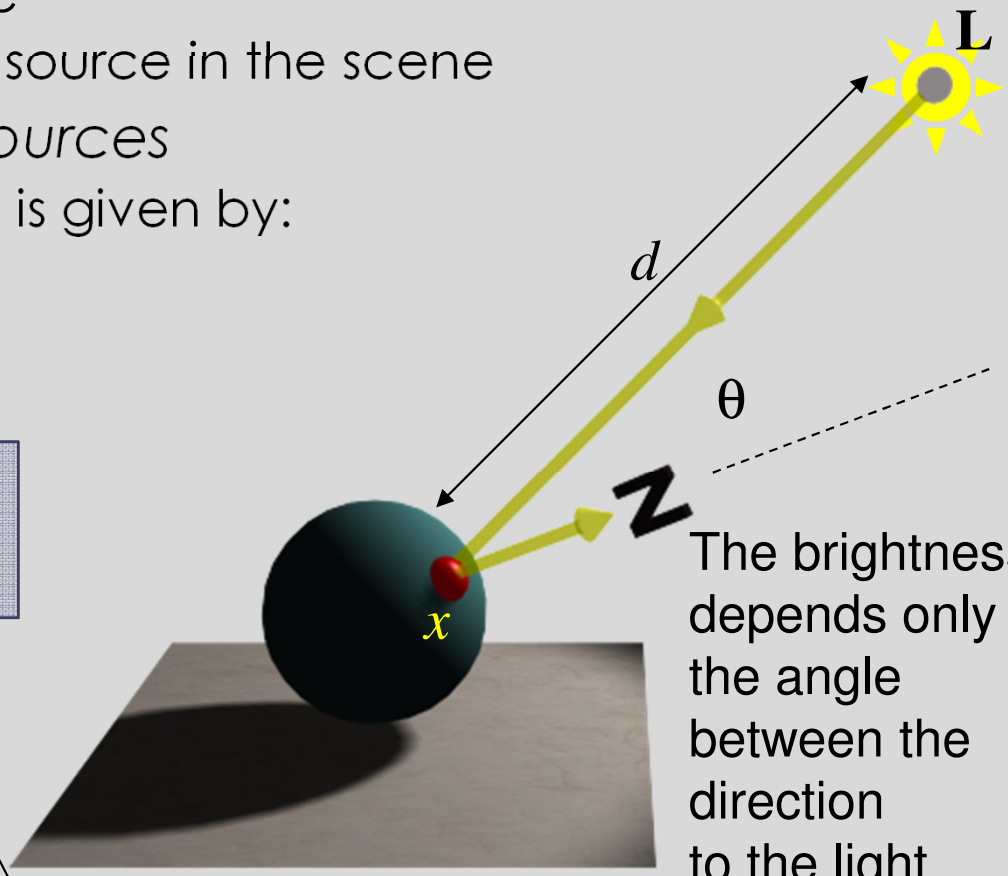
- The reflectivity varies from zero for a completely absorbing ("black") surface, to one for a completely reflecting ("white") surface.
- There are no Lambertian surfaces in nature, but matte paper is good approximation except at grazing angles and near 90 degrees), where the surface begins to look "shiny."

Lambertian Illumination Model

- To shade a diffuse surface we need to know:
  - *normal* to the surface at the point to be shaded
  - diffuse *reflectance* of the surface
  - *positions* and *powers* of the light source in the scene
- We will assume *isotropic point sources*

  ⇒ contribution from a single source is given by:

$$\underbrace{L_{r,d}(x,\cdot)}_{\text{Reflected radiance}} = \underbrace{\frac{\rho_d}{\pi}}_{\text{BRDF}} \underbrace{\cos\theta \frac{\Phi_s}{4\pi d^2}}_{\text{Incident radiance}}$$



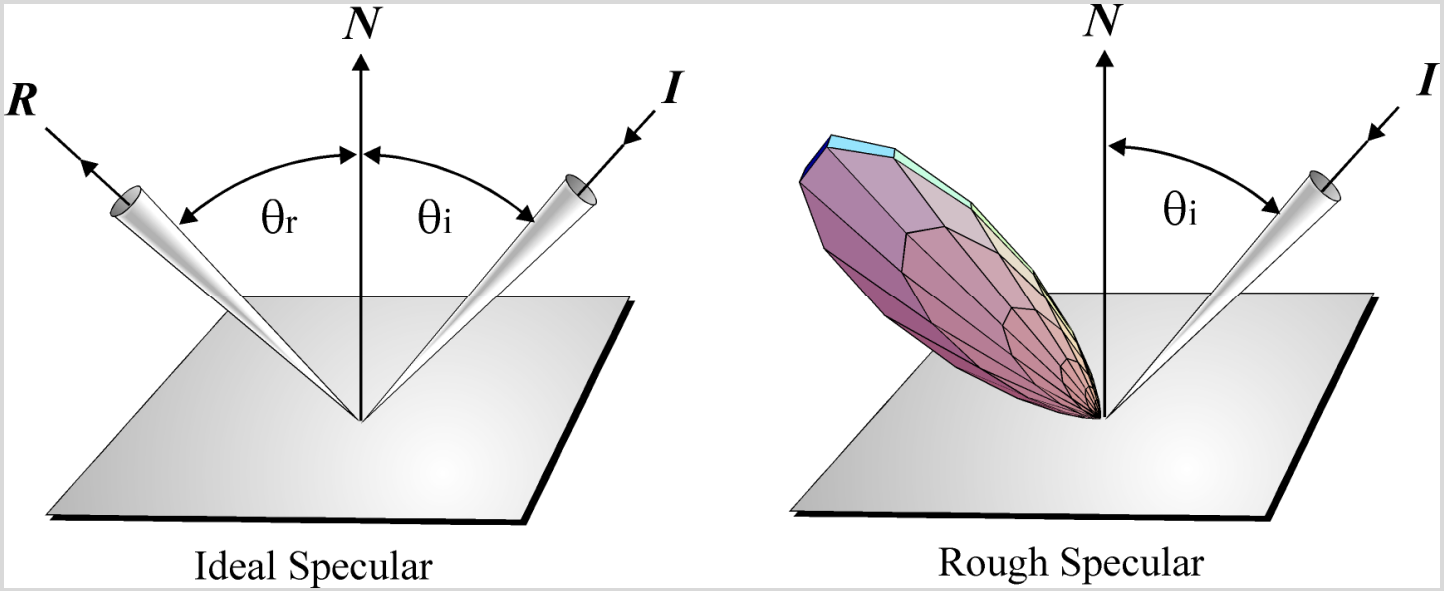
The brightness depends only on the angle between the direction to the light source and the surface normal

Lambertian surfaces

- Below are several examples of a spherical diffuse reflector with a varying lighting angles.

Phong Illumination Model

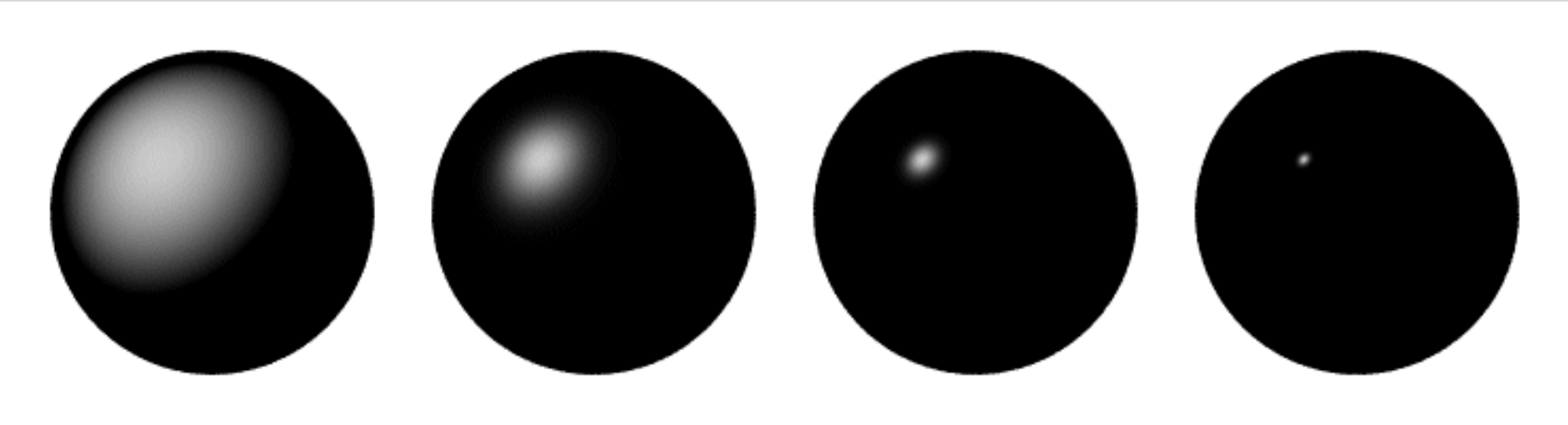
- *Specular* surfaces exhibit a high degree of *coherence* in their reflectance, i.e. the reflected radiance depends very heavily on the outgoing direction.
  - An *ideal specular* surface is *optically smooth* (smooth even at resolutions comparable to the wavelength of light).
  - Most specular surfaces (rough specular) reflect energy in a tight distribution (or *lobe*) centered on the *optical reflection direction*:



Phong Illumination Model

- To simulate reflection we should examine surfaces in the reflected direction to determine incoming flux
  $\Rightarrow$*global illumination*
- A local illumination approximation considers only reflections of light sources.
- The Phong model is an *empirical* (based on observation) local model of shiny surfaces (tend to appear like plastic).
- It is assumed that the BRDF of such surfaces may be approximated by a *spherical cosine function* raised to a power (known as the *Phong exponent*).

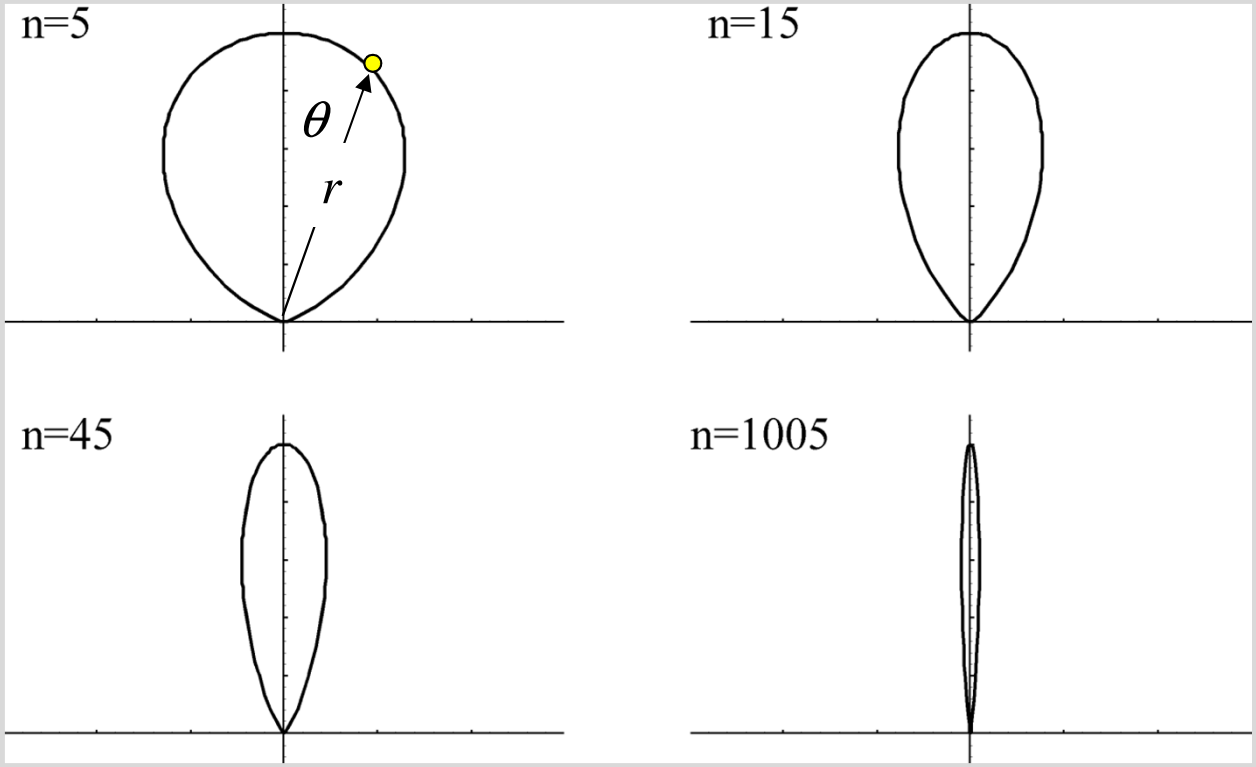
Which function?



The $\cos^n$ Function

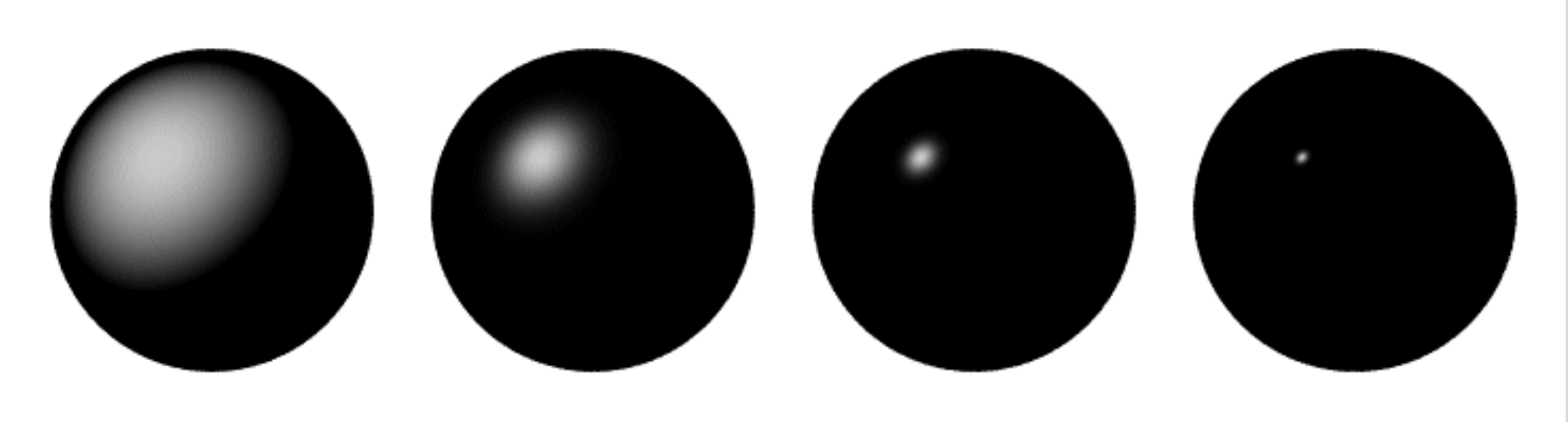
The cosine function (defined on the sphere) gives us a lobe shape which approximates the distribution of energy about a reflected direction controlled by the shininess parameter *n* known as the *Phong exponent*.

$$r = \cos^n \theta$$

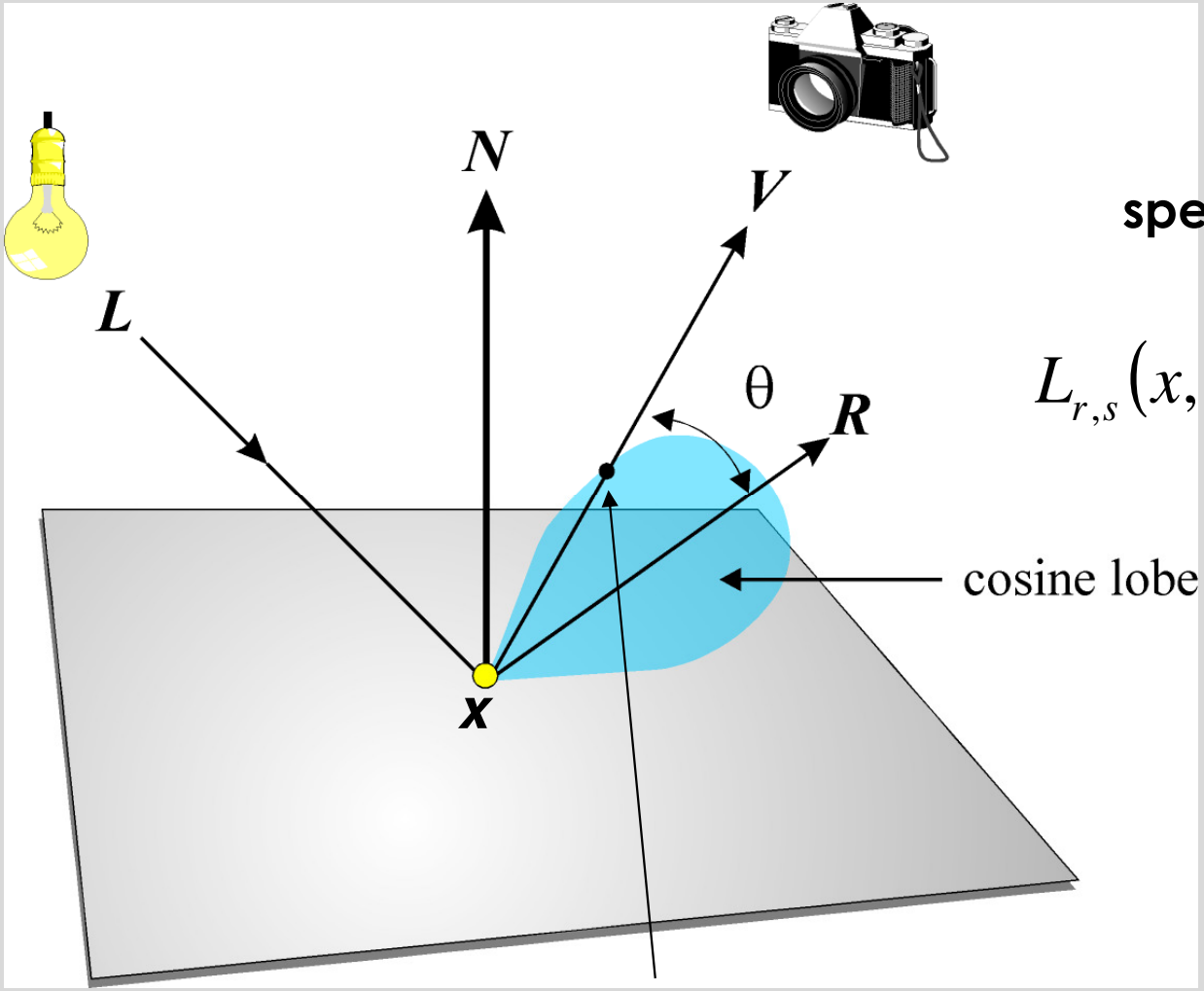


In the limit ($n \to \infty$) the function becomes a single spike (i.e. ideal specular). This is sometimes called the *delta function.*

The Cos^n Function



The Phong Model



$$L_{r,s}(x,V) = \frac{n+2}{2\pi}\rho_s \cos^n\theta \frac{\Phi_s}{4\pi d^2}$$

light source irradiance

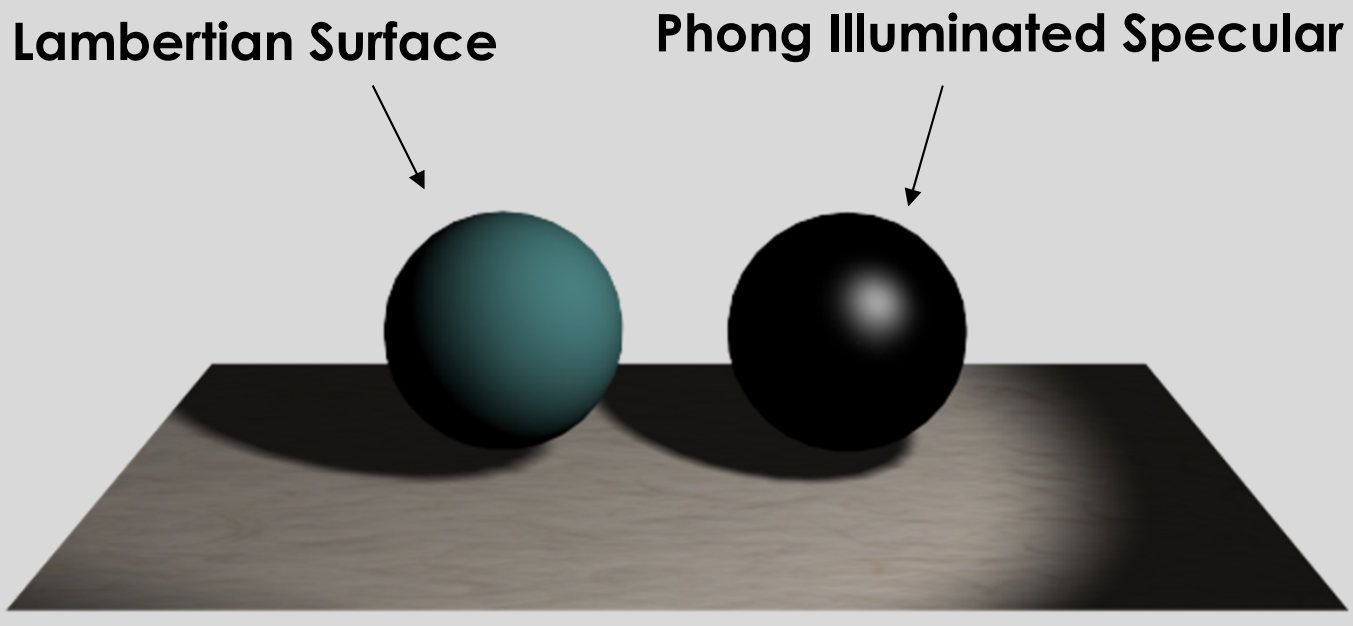
specular reflectivity

cosine lobe

Normalization term

Radiance of reflected light given by cosine function

Pure Lambertian vs. Phong



Lambertian vs. Phong

- Lambertian surfaces exhibit surface reflection independent of orientation and distance from viewer but not light source, leading to matte appearance.
- Specular surfaces exhibit surface reflection, dependent on orientation and distance of both viewer and light source, leading to glossy appearance with highlights.

Ambient Illumination

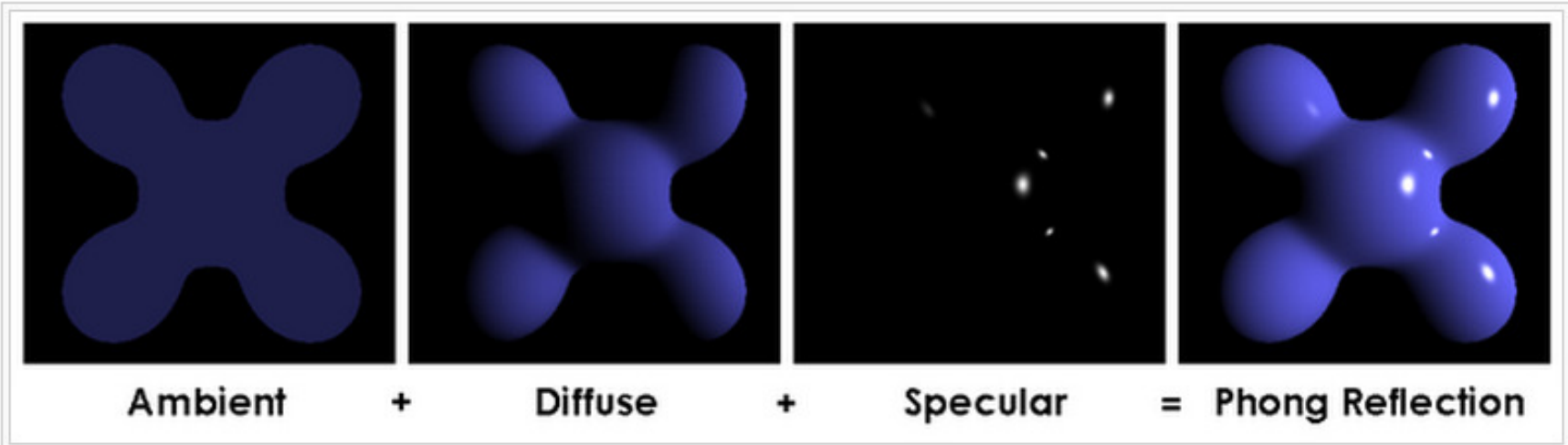
- Local illumination models account for light scattered from the light sources only.
- Light may be scattered from all surfaces in the scene
  - we are missing some light; in fact we are missing a lot of light, typically over 50%.
- *Ambient term* = a coarse approximation to this missing flux
- Ambient term defined by the ambient coefficient ρ_a:

$$I_a = \rho_a I_s$$

- This is a constant everywhere in the scene.
- The ambient term is sometimes estimated from the total powers and geometries of the light sources.

Putting it all together...

- The complete Phong Illumination Model includes
  - Lambertian model for diffuse reflection
  - Cosine lobe for specular reflection
  - Ambient term to approximate all other light



Putting it all together...

- The complete Phong Illumination Model includes
  - Lambertian model for diffuse reflection
  - Cosine lobe for specular reflection
  - Ambient term to approximate all other light
- An object must therefore have material data associated with it to define how diffuse, specular (and shiny) or ambient it is:

$$\text{Surface Data} = \begin{cases} \rho_a = \text{ambient reflectance} \\ \rho_d = \text{diffuse reflectance} \\ \rho_s = \text{specular reflectance} \\ n = \text{phong exponent} \end{cases}$$

Point Light Sources

- The irradiance E of a surface due to a point source obeys the inverse square law:

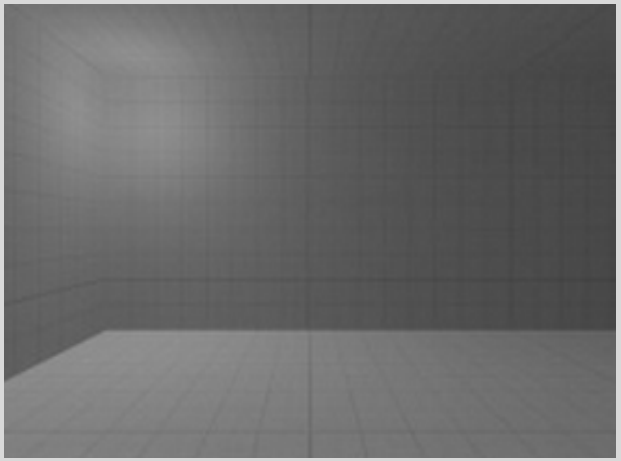
$$E = \frac{\Phi_S}{4\pi d^2} = \frac{I_S}{d^2}$$

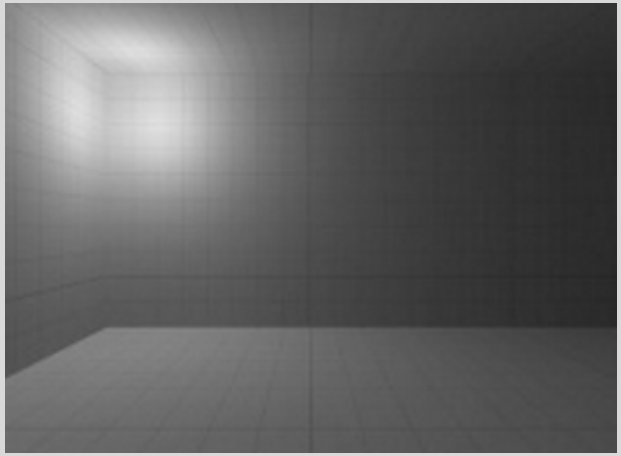
- However, this often makes it difficult to control the lighting in the scene, so we employ a less accurate but more flexible model of irradiance:

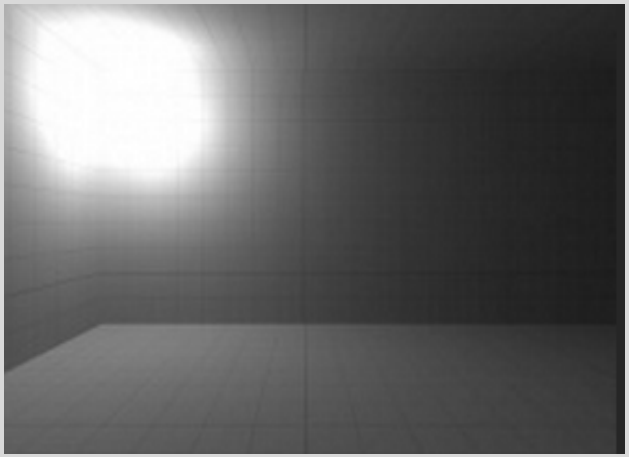
$$E = \frac{I_S}{a + bd + cd^2}$$

  - *a* = constant attenuation factor
  - *b* = linear attenuation factor
  - *c* = quadratic attenuation factor

Point Light Sources







- Constant

Linear

Quadratic

- The 100% constant attenuation will result in a light that has no attenuation at all.
- Linear light will diminish as it travels from its source.
- Quadratic: the further light travels from its source the more it will be diminished – sharp drop in light

Phong Illumination Model

$$L_r(x,V) = L_{r,a}(x) + L_{r,d}(x) + L_{r,s}(x,V) \quad = (\text{ambient} + \text{diffuse} + \text{specular})$$

$$= E\rho_a + E\rho_d\,(N \cdot L) + E\rho_s(V \cdot R)^n$$

cosΘ cos^{n}Θ

Note:
-ambient term not affected by light
-diffuse term affected by light but not by viewing angle
-specular term affected by viewing angle

Phong Illumination Model

$$L_r(x,V) = L_{r,a}(x) + L_{r,d}(x) + L_{r,s}(x,V) \quad = (\text{ambient} + \text{diffuse} + \text{specular})$$

$$= E\rho_a + E\rho_d\,(N \cdot L) + E\rho_s(V \cdot R)^n$$

$$= \frac{\Phi_s}{4\pi}\frac{1}{a + bd + cd^2}\left[\rho_a + \rho_d\,(N \cdot L) + \rho_s(V \cdot R)^n\right]$$

Phong Illumination Model

$$L_r(x,V) = L_{r,a}(x) + L_{r,d}(x) + L_{r,s}(x,V) \quad = (\text{ambient} + \text{diffuse} + \text{specular})$$

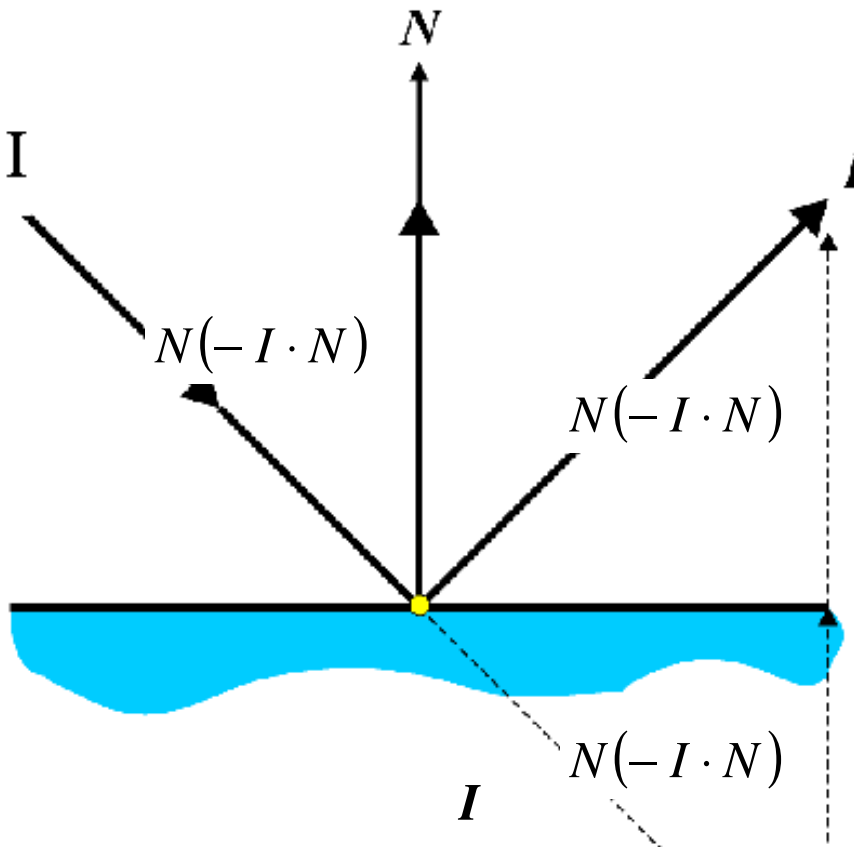
$$= E\rho_a + E\rho_d\,(N \cdot L) + E\rho_s(V \cdot R)^n$$

$$= \frac{\Phi_s}{4\pi}\frac{1}{a + bd + cd^2}\left[\rho_a + \rho_d\,(N \cdot L) + \rho_s(V \cdot R)^n\right]$$

For multiple light sources:

$$L_r(x,V) = \sum_{i=1}^{N}\left[\frac{\Phi_i}{4\pi}\frac{1}{a + bd_i + cd_i^2}\left[\rho_a + \rho_d\,(N \cdot L_i) + \rho_s(V \cdot R_i)^n\right]\right]$$

Determining the Reflected Vector

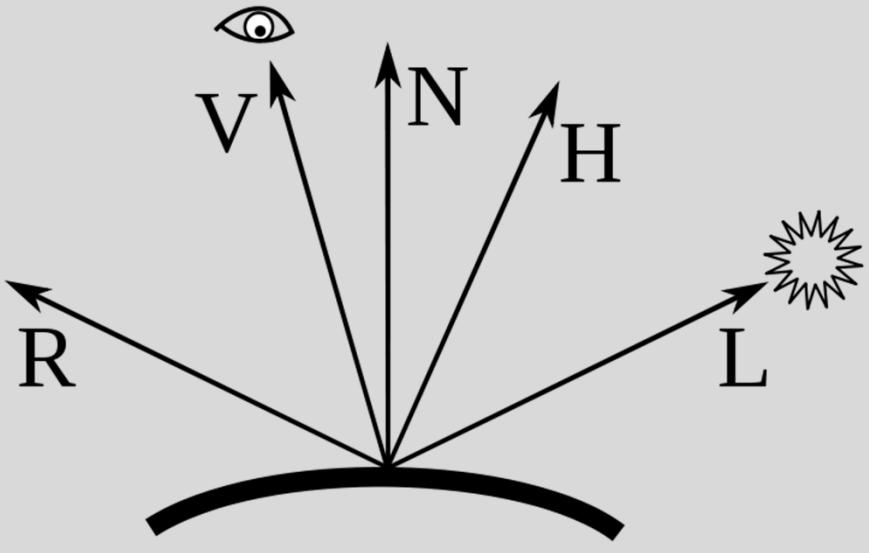


$$R = I + 2N(-I \cdot N)$$
$$= I - 2N(I \cdot N)$$

Blinn-Phong

- A problem with computing the perfect reflection direction, R, is that N is different for every point on the surface, so we must recompute R for every polygon – this is slow
- Blinn proposed an alternative formulation which employs the *half vector H* in place of *R*:

$$H = \frac{V + L}{|V + L|}$$



- The specular term is

$$(N \cdot H)^n$$

Half vector is a vector with a direction half-way between the eye vector and the light vector

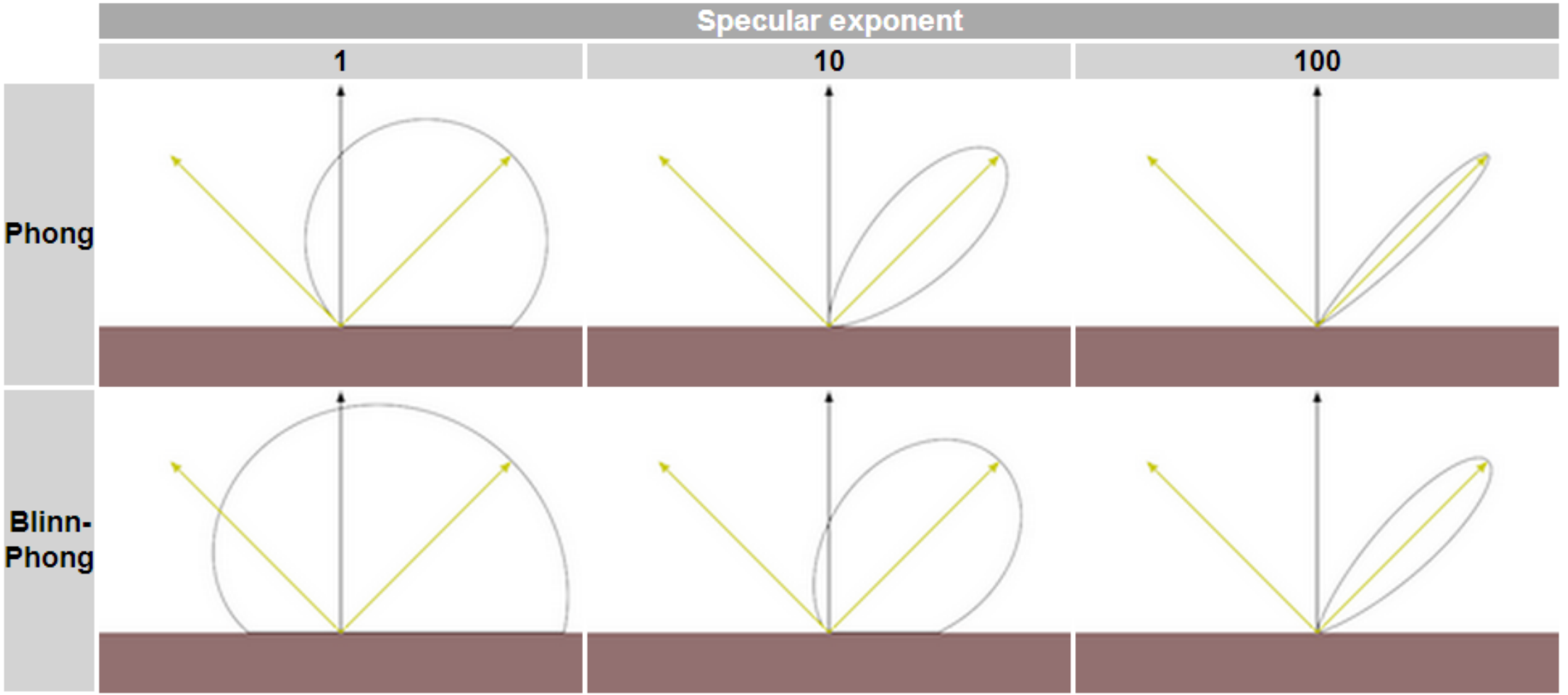
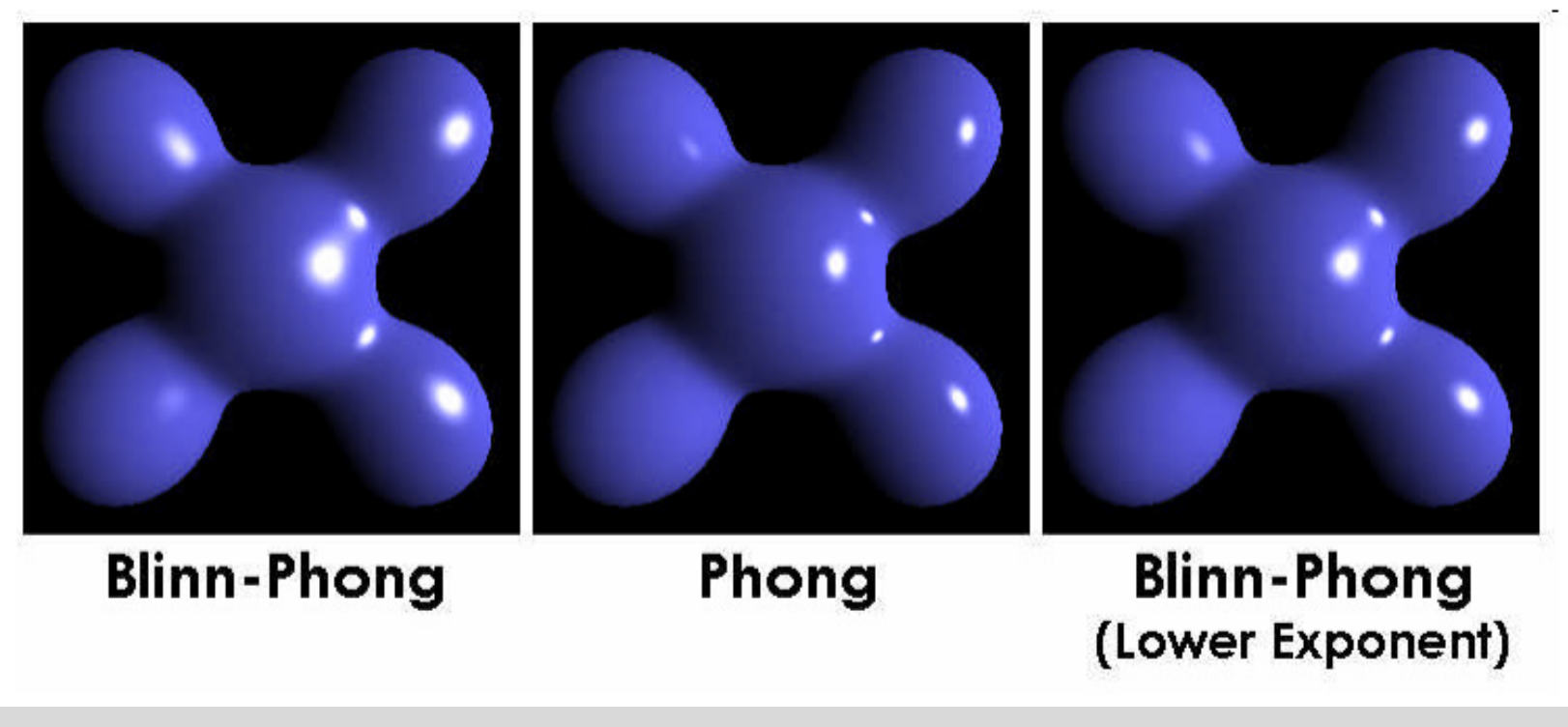


Figure 5: Phong vs Blinn-Phong With Varying Shininess Values.

Both the Phong and Blinn-Phong reflectance functions cause a highlight to appear around the direction of reflection. Blinn-Phong is cheaper to calculate, but appears more spread out at the same shininess.

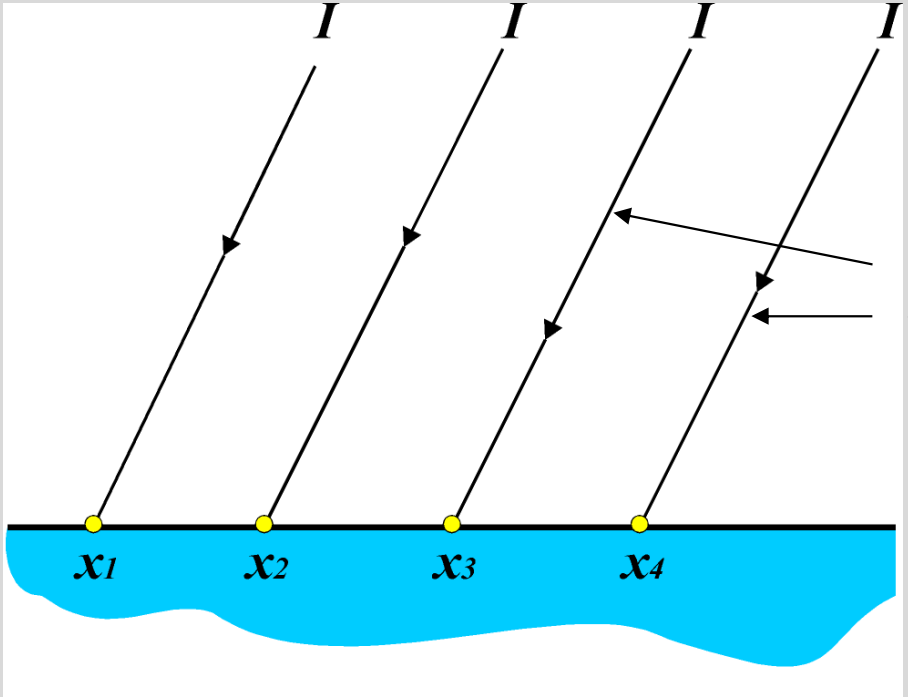
Approximating Phong

- Change the exponent so that the Blinn-Phong model matches Phong



Other Light Source Types

- We can extend the functionality of the illumination model by admitting a number of other light source types:
  - *directional*: the source is assumed to be at infinity and therefore is represented by a direction rather than a position.
  - *spot lights*: we can admit illumination only within a restricted solid angle.



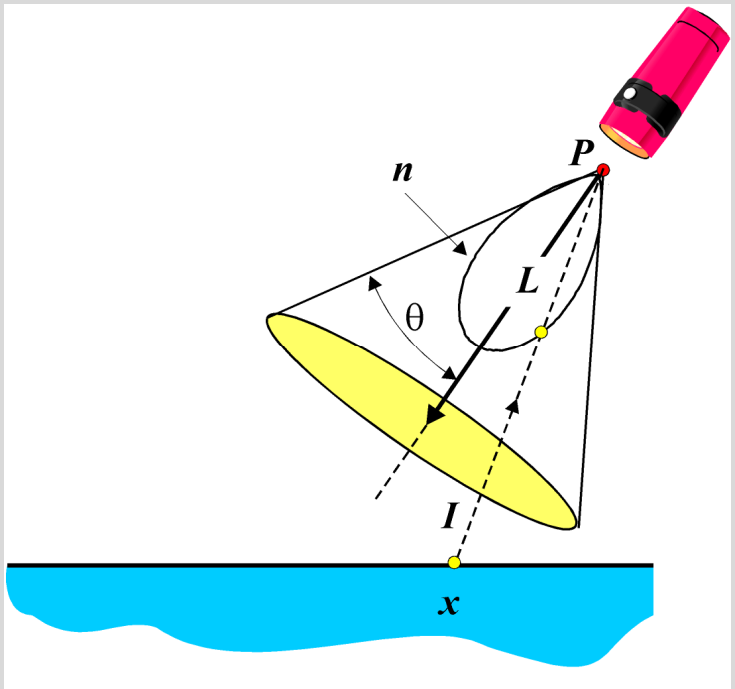
Directional Source

direction to light source
is constant everywhere

⇒ don't need to calculate *I* for each x_i
⇒ light radiant intensity is constant
(i.e. does not vary with distance)

Spot Lights

- Originally proposed by Warn (1983): a light source is defined by a *position*, a *direction*, a *cutoff angle* and an *exponent*.
- The radiant intensity of the spot light in a given direction is defined by a model similar to Phong's illumination model:



$$d = |P - x| \quad I = \frac{P - x}{d}$$

$$\text{if } \cos^{-1}(-I \cdot L) < \theta \text{ then}$$

$$E_s = \frac{\Phi_s}{4\pi(a + bd + cd^2)} \left[\frac{(-I \cdot L)}{\cos\theta} \right]^n$$

spotlight attentuation

Spot Lights

Incorporating Colour

- In fact, many of the systems that implement the Phong model differ only in the manner in which colour is handled.
- Typically we handle diffuse and specular illumination separately:
  - **diffusely** reflected light results from the reflection via multiple scattering events in the micro-scale geometry ⇒ *reflected light is coloured by selective absorption by the surface* i.e. a green surface absorbs all wavelengths except green
  - **specularly** reflected light interacts once with the surface and is thus *not coloured by the surface* i.e. the reflection of a light source takes on the colour of the source

Incorporating Colour

- **Therefore the Phong model becomes:**

$$L_r(x,V) = L_{r,a}(x) + L_{r,d}(x) + L_{r,s}(x,V)$$

$$= EC_{ambient}\rho_a + EC_{surface}\,\rho_d\,(N \cdot L) + EC_{light}\,\rho_s(V \cdot R)^n$$

- **Usually however, for flexibility, the ambient, diffuse and specular reflections are scaled independently by colour vectors:**

$$L_r(x,V) = EC_{amb}\rho_a + EC_{diff}\,\rho_d\,(N \cdot L) + EC_{spec}C_{light}\;\rho_s(V \cdot R)^n$$

- **The model is applied using colour vectors yielding the final colour vector to assign to a pixel.**
- **Note that only the specular term is scaled by the light's colour.**

Lighting in OpenGL

- Blinn-Phong used to be standard in computer graphics
- Specified as part of the OpenGL fixed-function pipeline
- Now, we are free to implement other lighting models and where to apply them
- Must specify a group of lighting and material parameter and use them in the application code or send them to the shaders.

Light source

- Must specify
  - Colour (diffuse, specular, ambient)
  - Location/direction

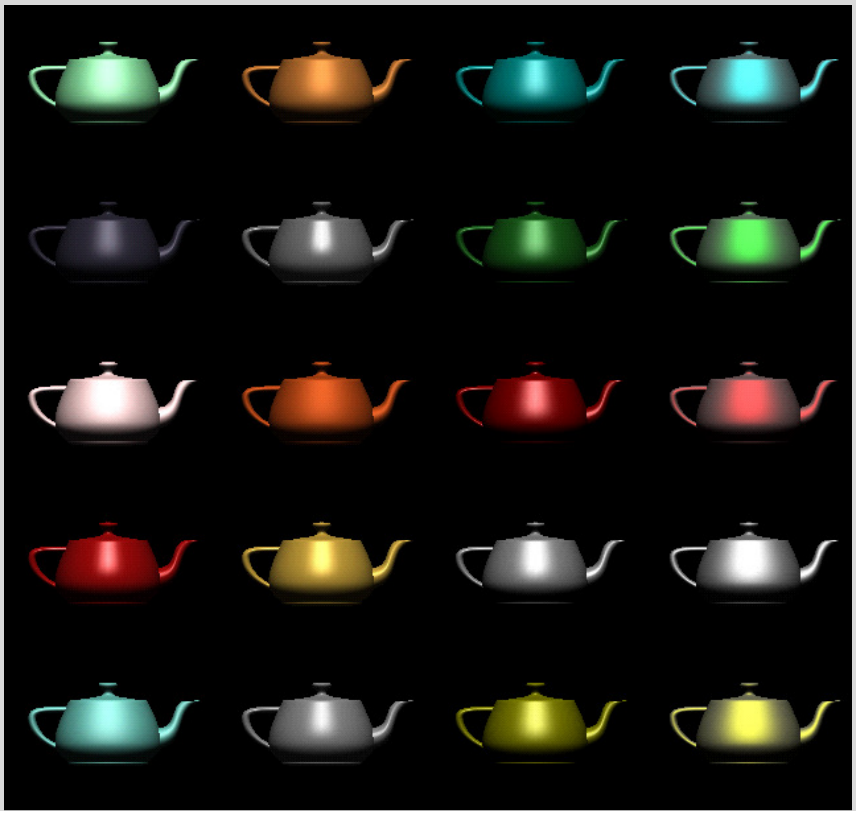
- `color4 light_diffuse, light_specular, light_ambient;`

- `point4 light_position;`

- `float attenuation_const, attenuation_linear, attenuation_quadratic;`

Materials

- `color3 ambient = color3(0.2, 0.2, 0.2);`
- `color3 diffuse = color3(0.2, 0.2, 0.2);`
- `color3 specular = color3(0.2, 0.2, 0.2);`
- `float shininess; //phong exponent`



Implementing a lighting model

- Take the simple version of Blinn-Phong model using a single light source
- Multiple sources is additive, we can repeat calculation for each source and add up the contributions
- 3 choices as to where to do calculation
  - Application
  - Vertex shader
  - Fragment shader
- Difference in efficiency and appearance depending where calculation is done

Vertex Shader Code

```
in vec3 vertex_position;
in vec3 vertex_normal;
uniform mat4 projection_mat, view_mat, model_mat;
out vec3 position_eye, normal_eye;

void main () {
        position_eye = vec3 (view_mat * model_mat * vec4 (vertex_position, 1.0));
        normal_eye = vec3 (view_mat * model_mat * vec4 (vertex_normal, 0.0));
        gl_Position = projection_mat * vec4 (position_eye, 1.0);
}
```

Normals

- **Important note:** If we are doing any scaling in the model matrix where the axes are different (e.g. a horizontal stretch) then the model matrix will incorrectly scale the normal.
- In this case we would need to use a new "normal matrix" which is the **inverse * transpose** of the view matrix * model matrix to correct the problem.

Ambient Intensity Term

```
in vec3 position_eye, normal_eye;

// fixed point light properties
vec3 light_position_world  = vec3 (0.0, 0.0, 2.0);
vec3 Ls = vec3 (1.0, 1.0, 1.0); // white specular colour
vec3 Ld = vec3 (0.7, 0.7, 0.7); // dull white diffuse light colour
vec3 La = vec3 (0.2, 0.2, 0.2); // grey ambient colour

// surface reflectance
vec3 Ks = vec3 (1.0, 1.0, 1.0); // fully reflect specular light
vec3 Kd = vec3 (1.0, 0.5, 0.0); // orange diffuse surface
reflectance
vec3 Ka = vec3 (1.0, 1.0, 1.0); // fully reflect ambient light
float specular_exponent = 100.0; // specular 'power'

out vec4 fragment_colour; // final colour of surface
```

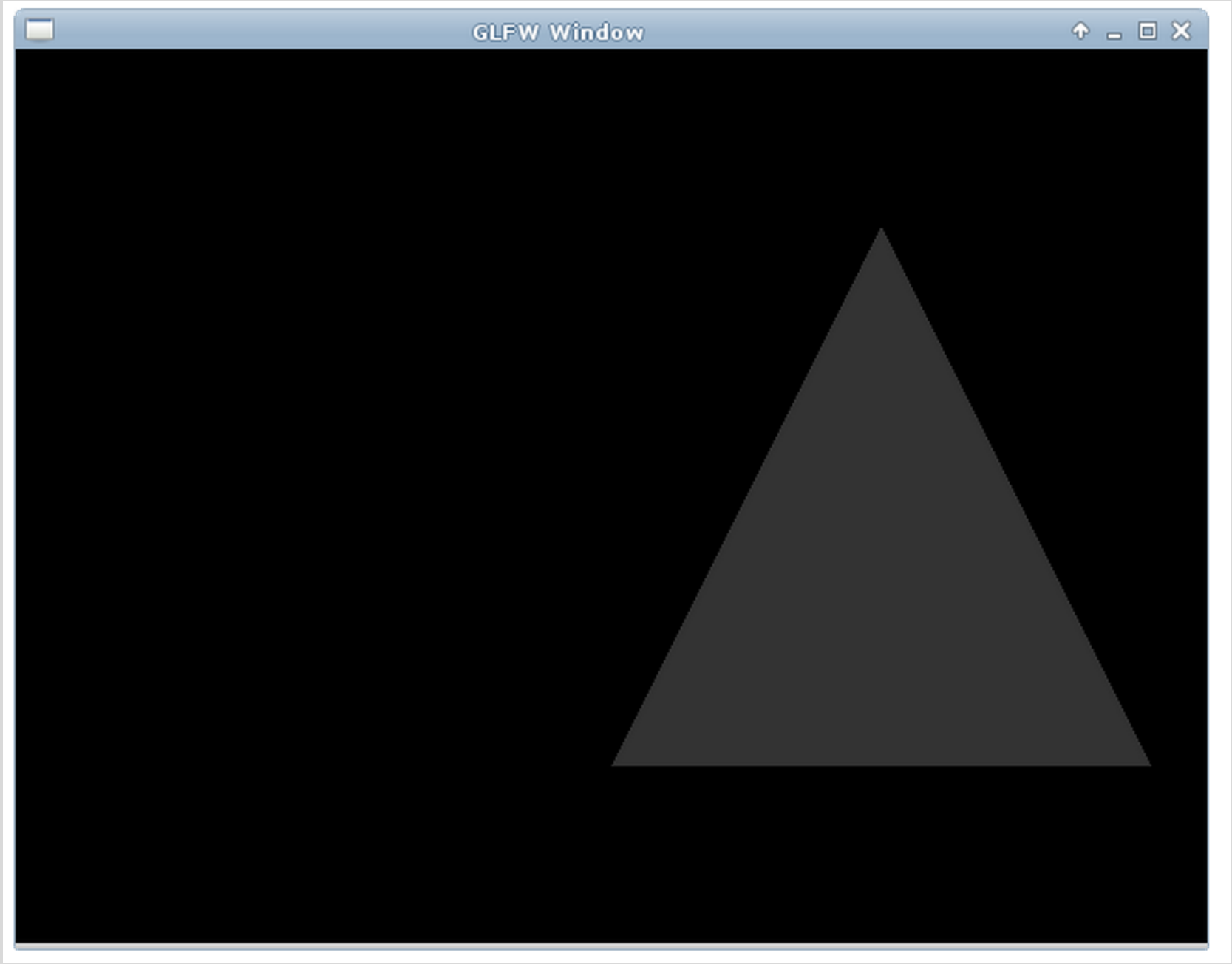
Ambient Intensity Term

```
void main () {
	// ambient intensity
	vec3 Ia = La * Ka;

	// diffuse intensity
	vec3 Id = vec3 (0.0, 0.0, 0.0); // replace me later

	// specular intensity
	vec3 Is = vec3 (0.0, 0.0, 0.0); // replace me later

	// final colour
	fragment_colour = vec4 (Is + Id + Ia, 1.0);
}
```


GLFW Window

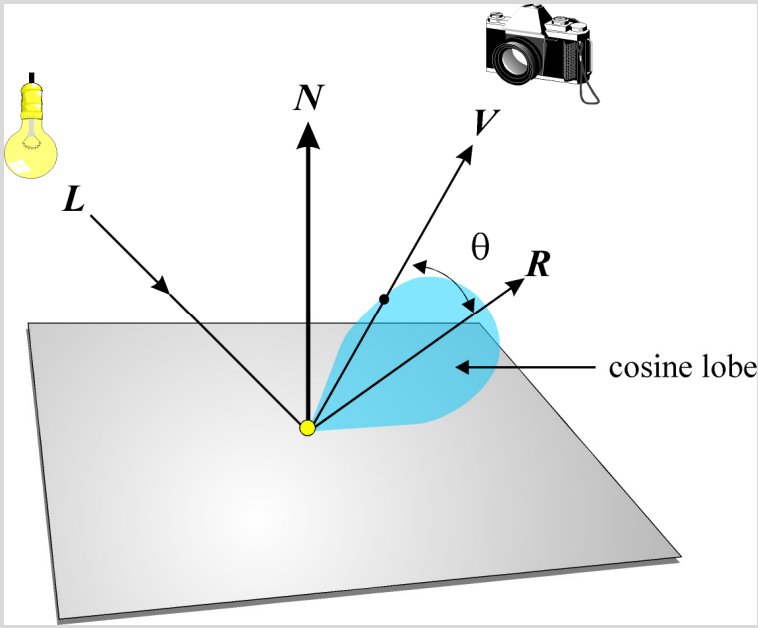
Diffuse Intensity Term

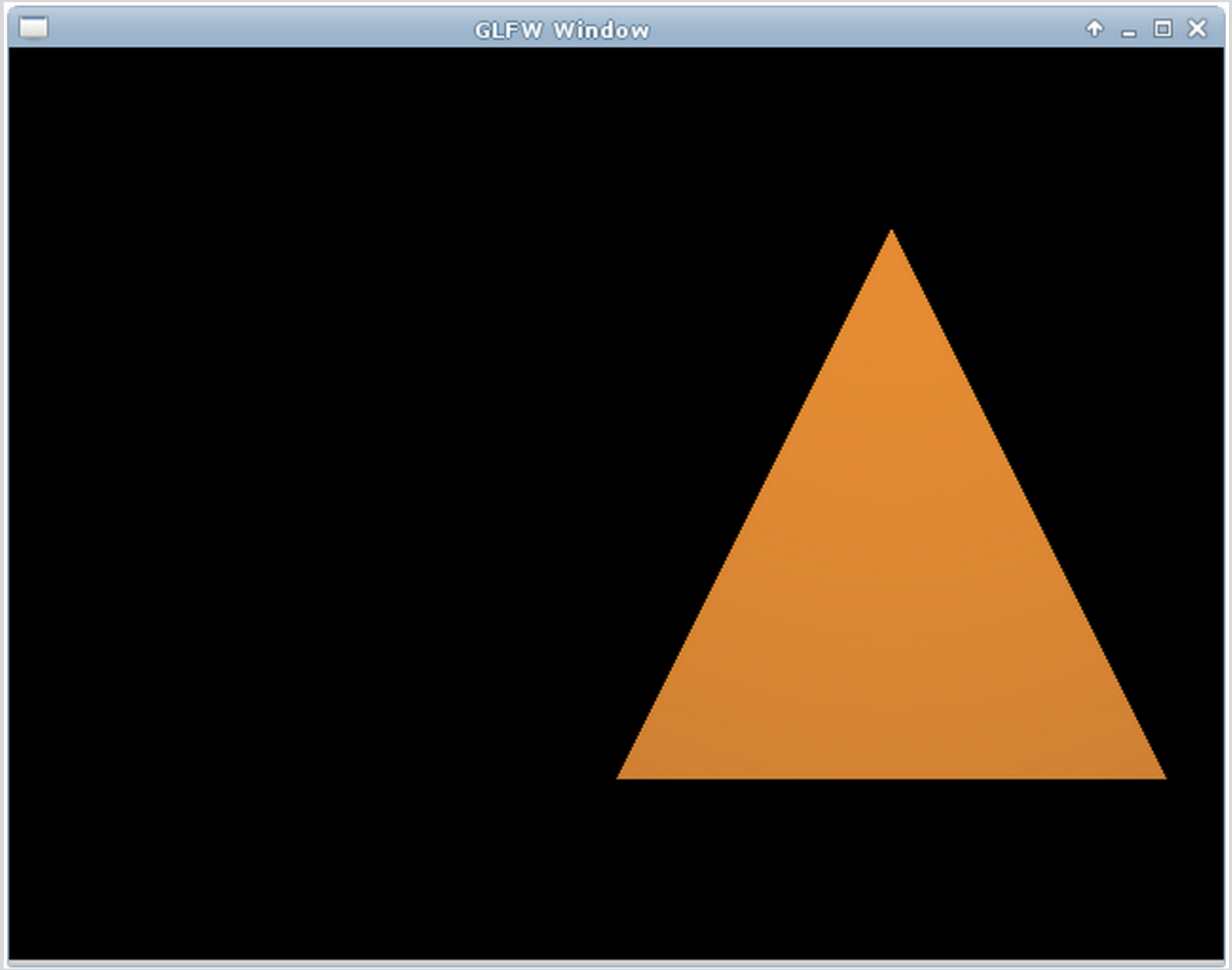
- Diffuse light should be brightest when the surface is facing the light (factor of 1.0), and not lit at all when the surface is perpendicular to the light (factor of 0.0).
  - USE Dot product

Diffuse Intensity Term

```
    ...
  // raise light position to eye space
  vec3 light_position_eye = vec3 (view_matrix * vec4
(light_position_world, 1.0));
  vec3 distance_to_light_eye = light_position_eye - position_eye;
  vec3 direction_to_light_eye = normalize (distance_to_light_eye);
  float dot_prod = dot (direction_to_light_eye, normal_eye);
  dot_prod = max (dot_prod, 0.0);
  vec3 Id = Ld * Kd * dot_prod; // final diffuse intensity
  ...
```

It is possible to produce a negative dot product. We can get around this by making the dot product a minimum of 0.0; I used the max() function to do this.




GLFW Window

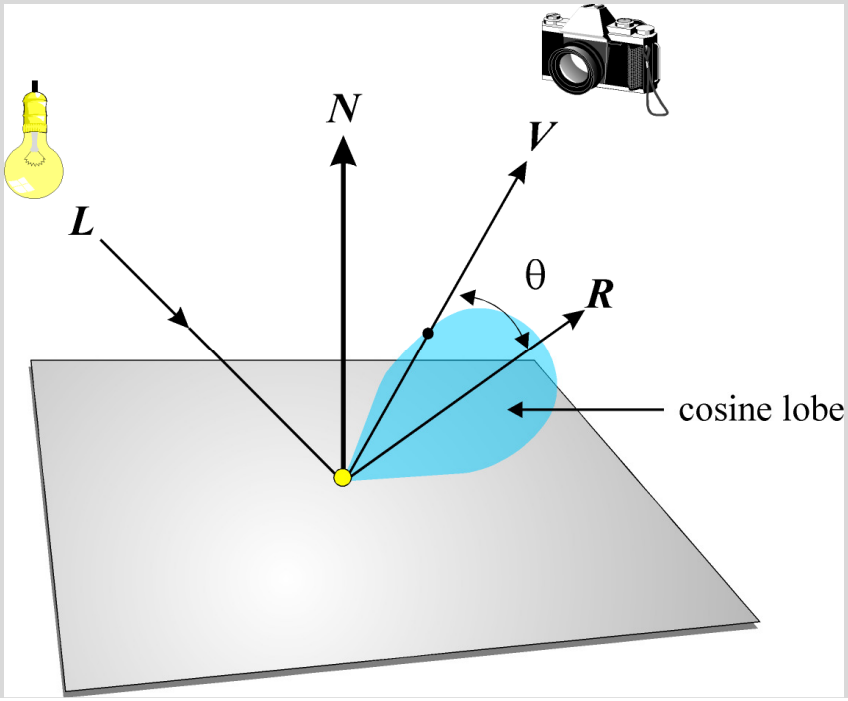
Specular Intensity Term

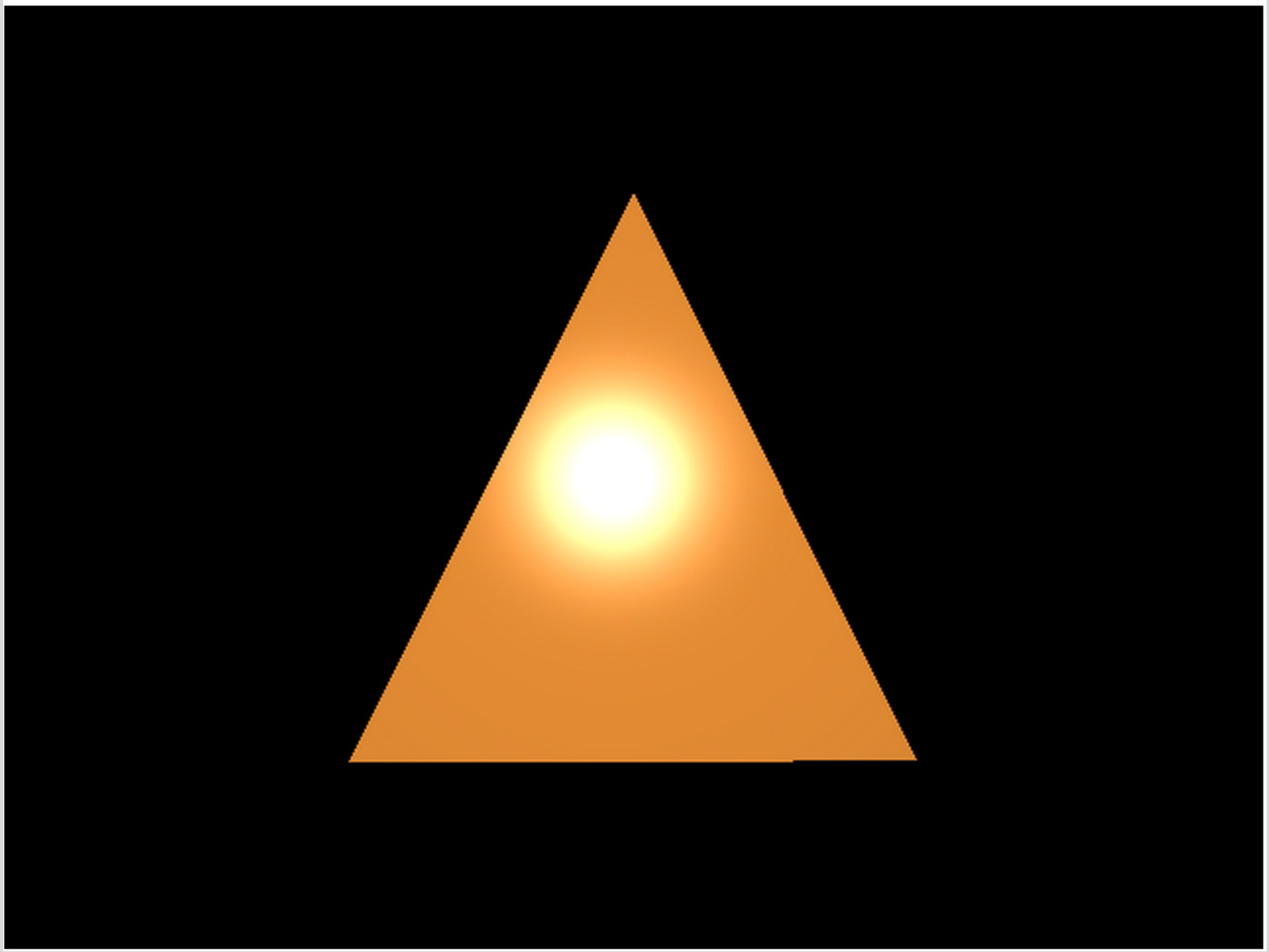
- Specular light should directly reflect around the surface normal.
- There is a built-in GLSL function called reflect() to calculate this for us.
- Then we can compare the angle between the viewer and the surface with this new, reflected direction. If they are the same then the specular colour should be factored by 1.0. If they are perpendicular then there should be no specular light observed. Once again - the dot product.
- The main difference between specular and diffuse light is that it now depends on the angle between the light, the surface, and the observer. This means that, as the object or the camera rotate, the specular highlight will change. The specular highlight should also affect only a focused area of the surface - we can determine how big or small the highlight should be by raising it to a power given by the specular_exponent variable.

Specular Intensity Term

```
...
  vec3 reflection_eye = reflect (-direction_to_light_eye, normal_eye);
  vec3 surface_to_viewer_eye = normalize (-position_eye);
  float dot_prod_specular = dot (reflection_eye, surface_to_viewer_eye);
  dot_prod_specular = max (dot_prod_specular, 0.0);
  float specular_factor = pow (dot_prod_specular, specular_exponent);
  vec3 Is = Ls * Ks * specular_factor; // final specular intensity
  ...
```

Built-in pow() function





Extra Reading

- Real-Time Rendering, 3rd Edition, Moller, Haines & Hoffman
- OpenGL SuperBible, 6th Edition, Sellers, Wright & Haemel
- Anton's Phong Tutorial
- http://antongerdelan.net/opengl/phong.html

Summary

- Rendering algorithms (local, global, view dependent, view independent)
- Light, colour, spectra
- Surface reflectance
- Solid angle, radiometric units, radiance
- Inverse square law, the cosine rule
- BRDF, BRDF approximations
- Reflectance equation, radiance equation
- Light sources
- Shading Models
- Illumination Models
- Practical Implementation
- Light source approximations