CENG 477
Introduction to Computer Graphics

Ray Tracing: Shading

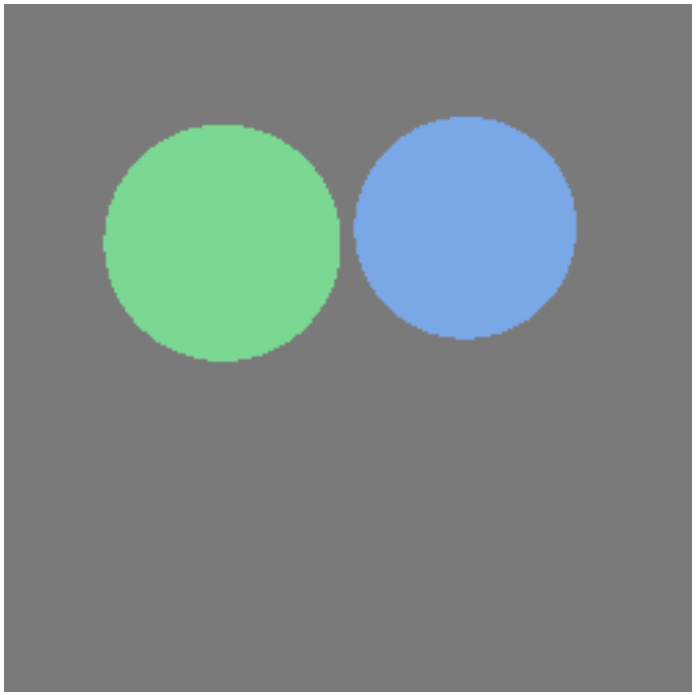
Last Week

- Until now we learned:
  - How to create the primary rays from the given camera and image plane parameters
  - How to intersect these rays with various objects:
    - Planes
    - Spheres
    - Triangles

Algorithm So Far

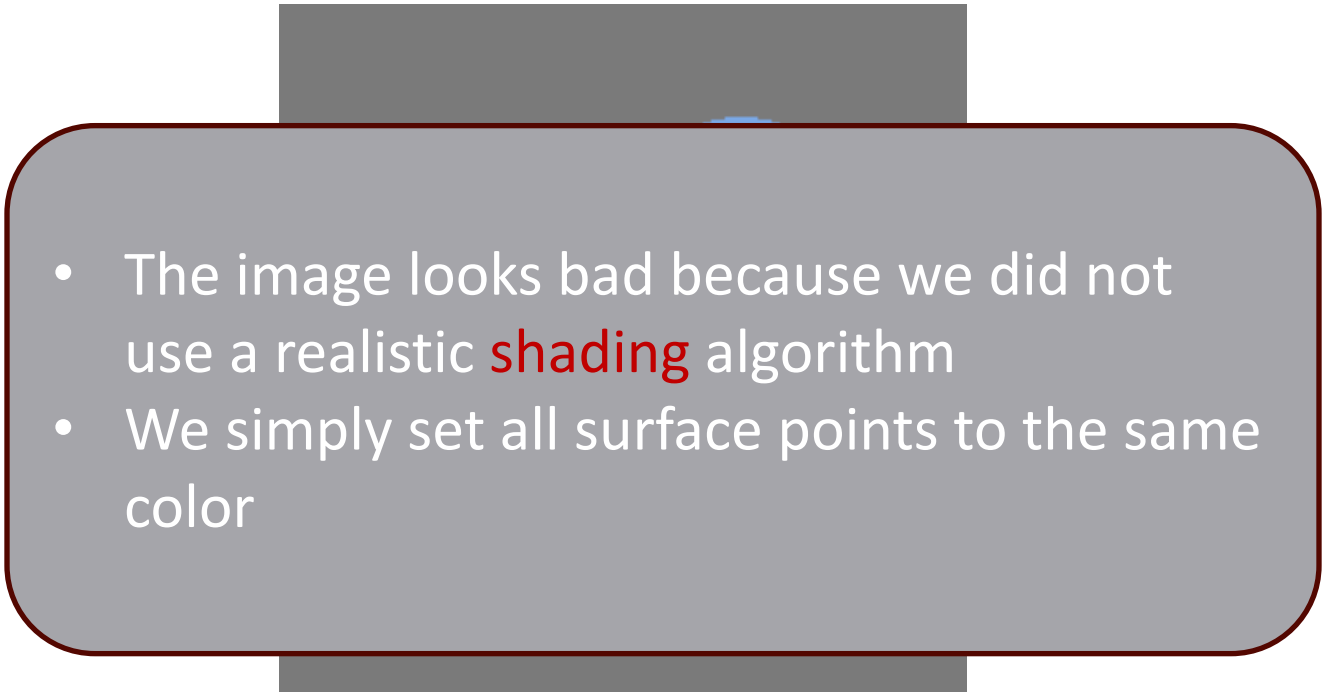
foreach pixel s:
    compute the viewing ray **r** (from **e** to **s**)
    $t_{min} = \infty$, obj = NULL
    foreach object o:
        if r intersects o:
            if $t < t_{min}$:
                $t_{min} = t$, obj = o
    if obj not NULL:
        pixel color = color of obj
    else
        pixel color = color of background

Image So Far



- Two spheres with different colors and a gray background
- **So much work for such a bad image!**

Image So Far

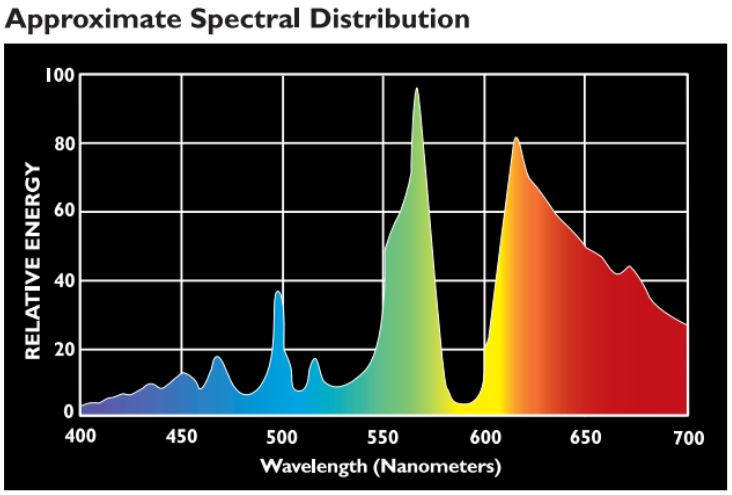


- The image looks bad because we did not use a realistic shading algorithm
- We simply set all surface points to the same color

- Two spheres with different colors and a gray background
- **So much work for such a bad image!**

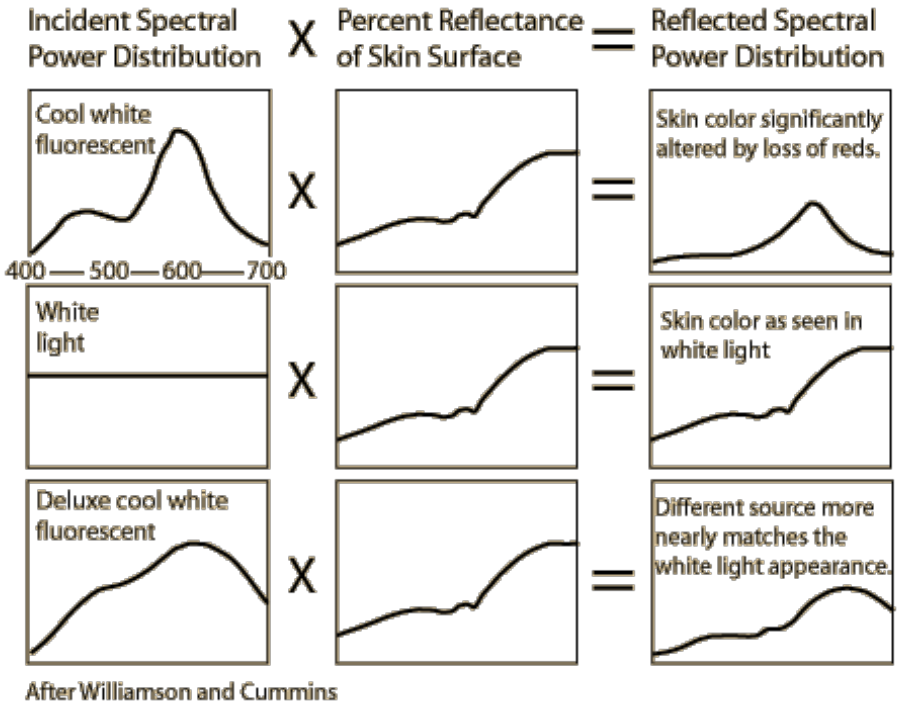
Reflectance and Power

- We assumed that each object has a color
- This is not how it is in reality
- Each object has a reflectance distribution
- Each light source has a power distribution
- Normally, both are functions of wavelength:



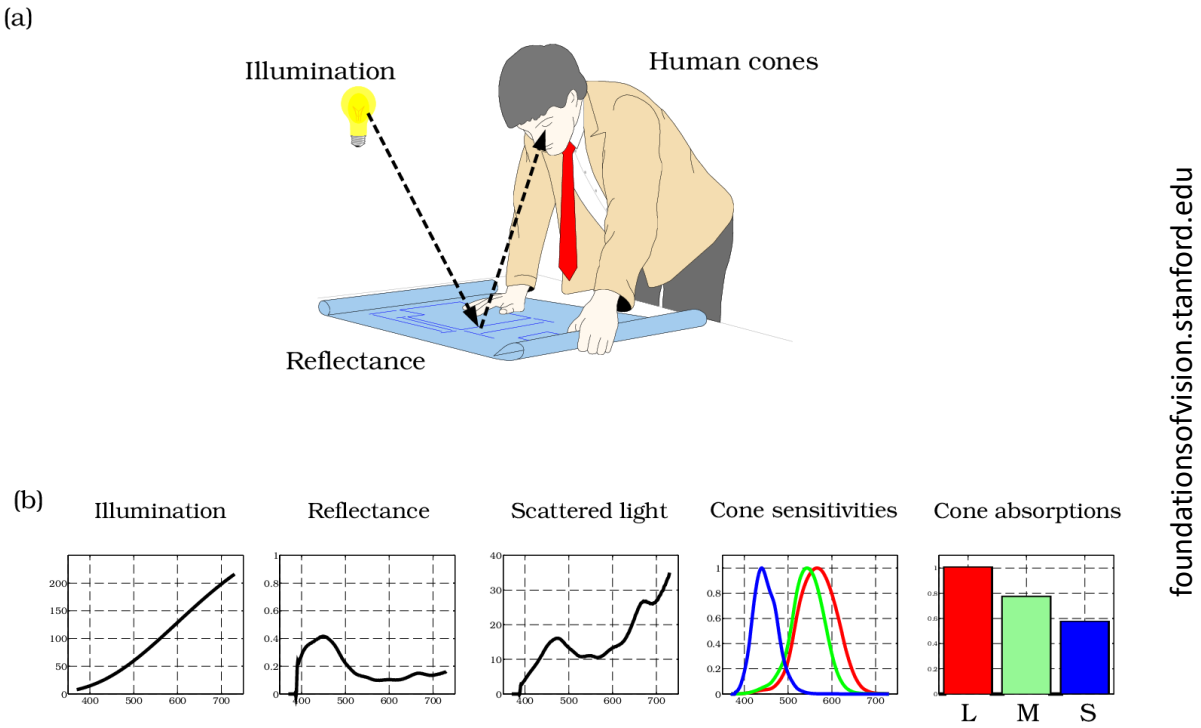
From forums.gardenweb.com

Reflectance and Power



- The object color occurs as a result of their interaction
- Same object can appear different under different lights

Reflectance, Power, Cones



foundationsofvision.stanford.edu

- Color is perceived due to the interaction of this reflected light by our cone pigments

Spectral Ray Tracing

- Accurately modeling this process requires us to represent all objects and light sources using spectral (i.e. wavelength-based) distributions
- Some ray-tracers do this:
  - Indigo renderer (http://www.indigorenderer.com/)
  - Lux renderer (http://www.luxrender.net/)
  - Mental ray (http://www.nvidia-arc.com/products/nvidia-mental-ray.html)
  - …

RGB Model

- We will make a simplifying assumption and represent all objects and light source with three components
- Reflectance:
  - R_r: How much red light it reflects, a value between [0,1]
  - R_g: How much green light it reflects, a value between [0,1]
  - R_b: How much blue light it reflects, a value between [0,1]
- Power:
  - I_r: How much red light it emits, a value between [0,∞)
  - I_g: How much green light it emits, a value between [0,∞)
  - I_b: How much blue light it emits, a value between [0,∞)

Terms

- The following terms are important when talking about light-surface interaction:
  - Power
  - Intensity
  - Radiance
  - Irradiance

Power

- Power is the time-rate of energy
- It measures how much energy (i.e. in Joules) a light source emits in all possible directions per second
- Power is measured in Watts (W = J/s)
- Power (P) is also known as flux (Φ)

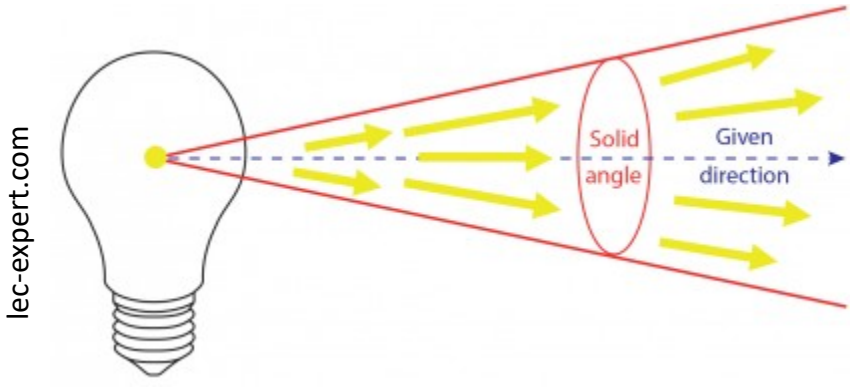
$$P = \frac{dQ}{dt}$$

Energy

time

Intensity

- Intensity is defined as power per solid angle
- Typically defined for point light sources and measured in W/sr



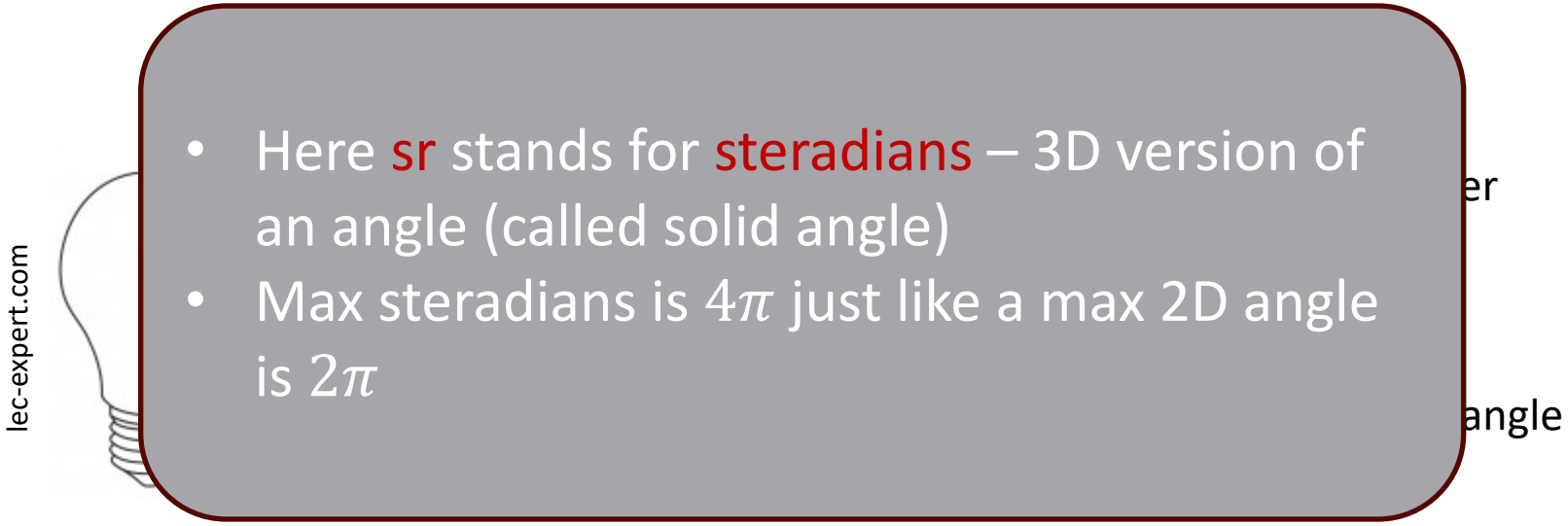
$$I = \frac{dP}{dw}$$

Power

Solid angle

Intensity

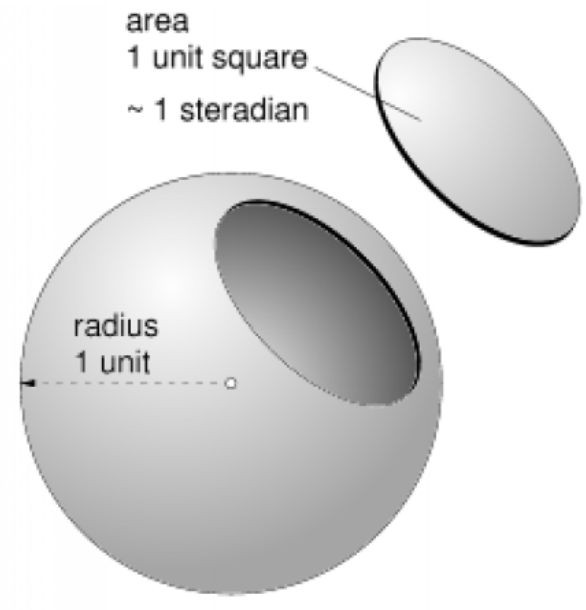
- Intensity is defined as power per solid angle
- Typically defined for point light sources and measured in W/sr



- Here sr stands for steradians – 3D version of an angle (called solid angle)
- Max steradians is 4π just like a max 2D angle is 2π

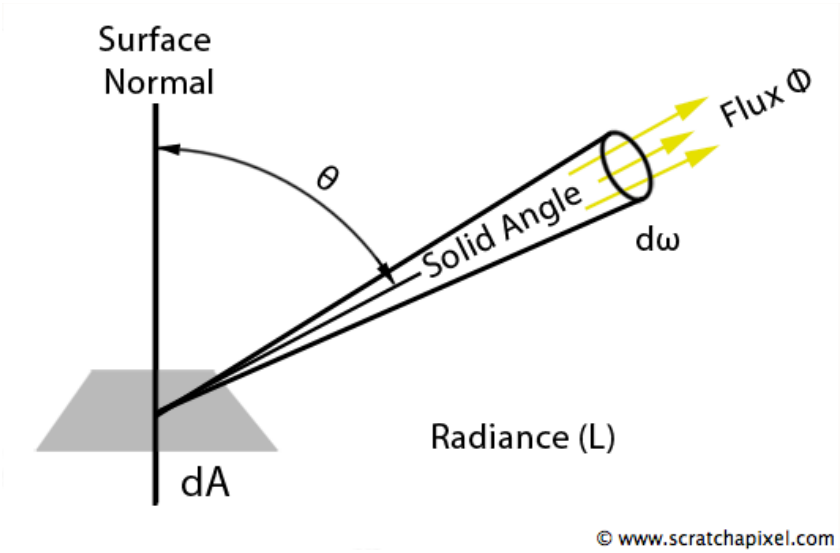
Solid Angle

- Solid angle is the 3D generalization of the angle
- It corresponds to a surface area on a unit sphere
  - similar to angle corresponding to a length on a unit circle
  - measured in steradians



Radiance

- Radiance is defined as power per solid angle per projected area
- Measured in Watts per steradian per meter squared ($W/sr/m^2$)



$$L = \frac{dP}{dwdA\cos\theta} = \frac{dI}{dA\cos\theta}$$

Intensity

Projected area

Radiance

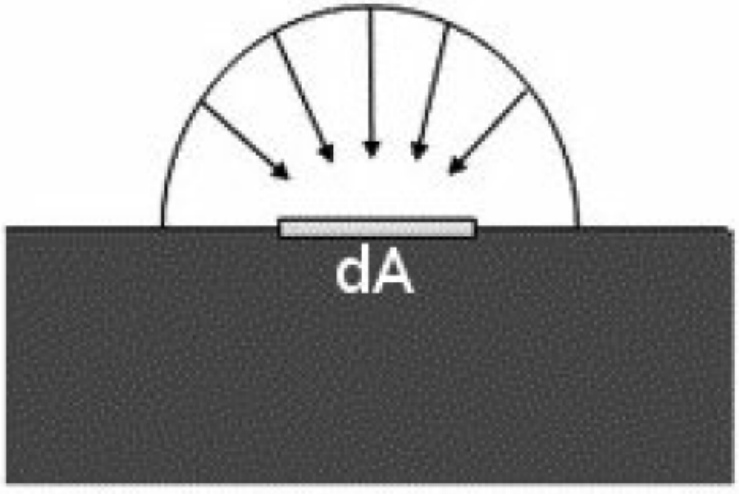
- Radiance is the most important unit in ray tracing
- Radiance of a ray does not change as the ray travels in empty space

The Goal of Ray Tracing

Compute the radiance along each viewing ray

Irradiance

- Irradiance measures the total incident power per unit area
- Measured in W/m^2



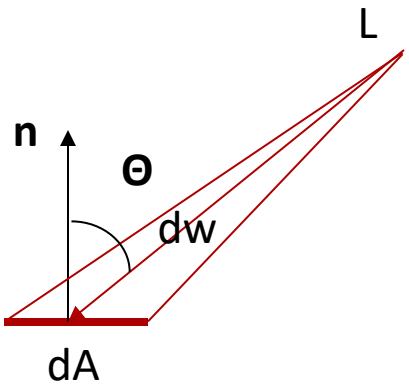
$$E = \frac{dP}{dA}$$

Power

Area

Irradiance from Radiance

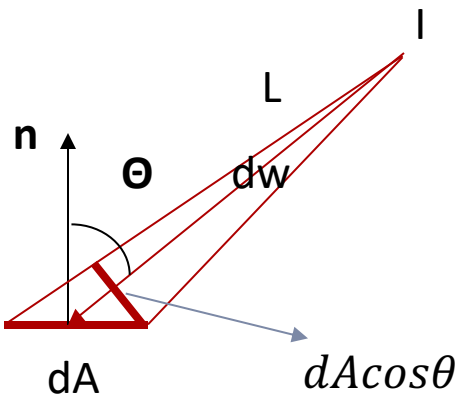
- Irradiance can be computed from radiance:



$$dE = \frac{dP}{dA} = Ldwcos\theta$$

Radiance from Intensity

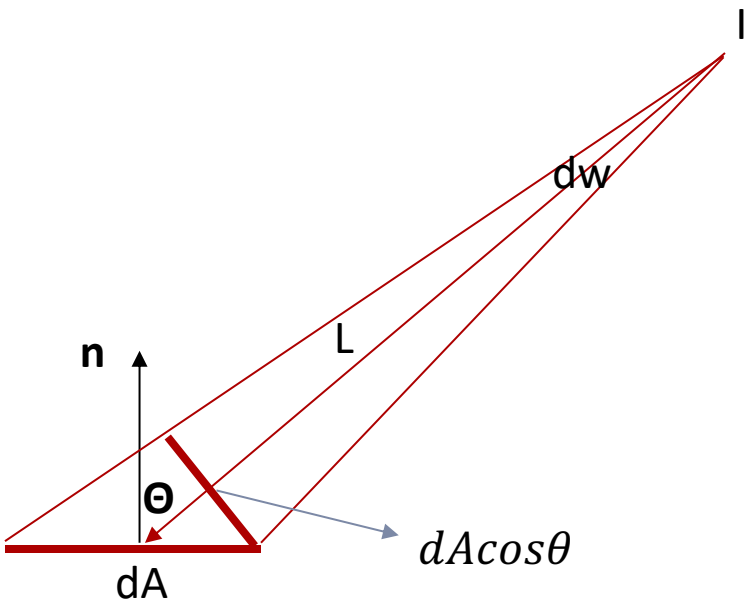
- Finally, radiance can be computed from intensity:



$$L = \frac{dI}{dA\cos\theta}$$

Radiance from Intensity

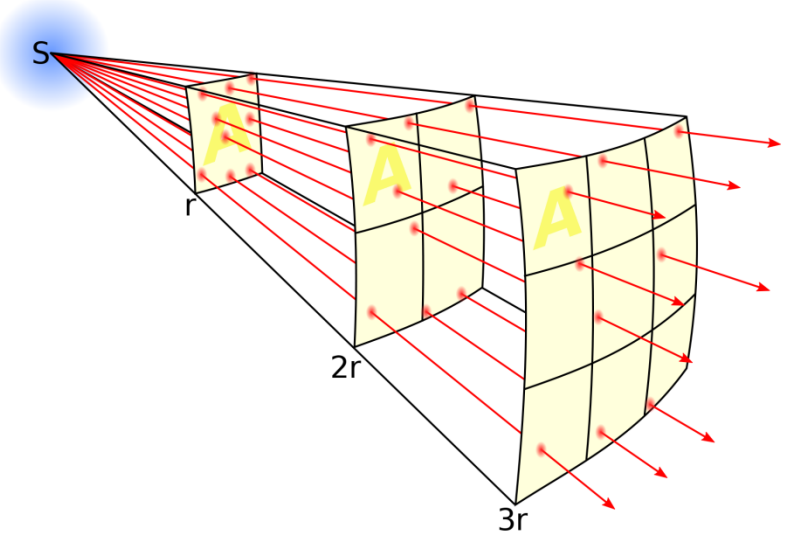
- Now, imagine that dA was further away
- The same intensity would be spread over a larger area
- This relationship is governed by the inverse square law



$$L = \frac{dI}{dAcos\theta}$$

Inverse Square Law

- The irradiance emitted by a point light source is reduced by the square of the distance from the source

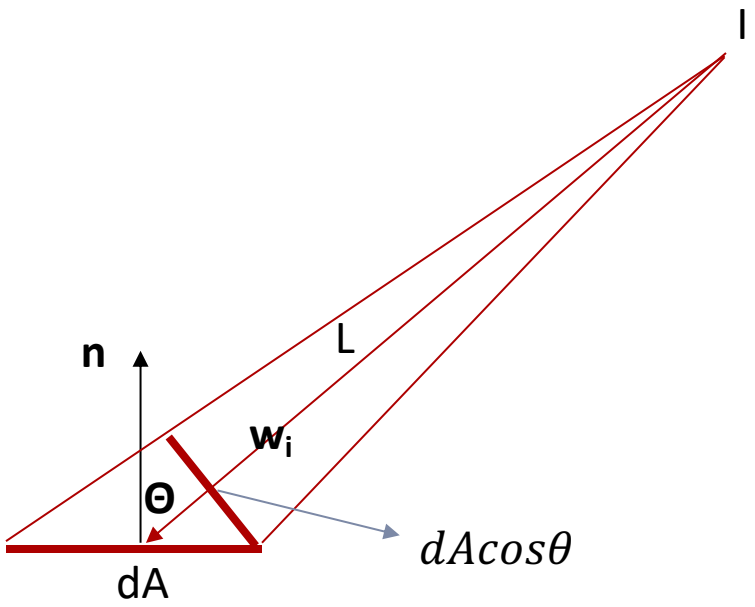


Inverse Square Law

- Objects further away from the light source appear dimmer as a result
- This is why the distance of the stars can be judged by their brightness

The Cosine Law

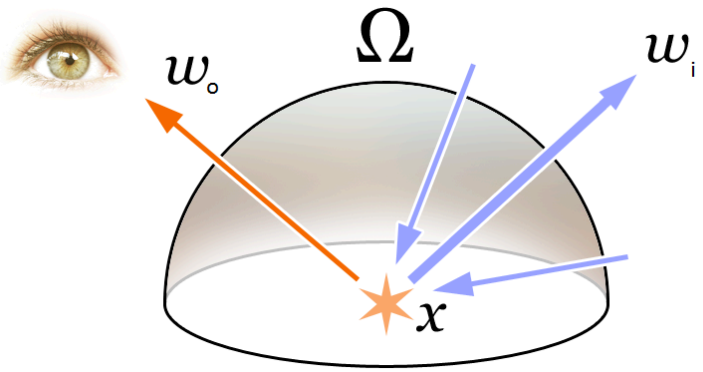
- The cosine law states that a surface can receive (or emit) radiation only in proportion to its area projected in the direction of the light:



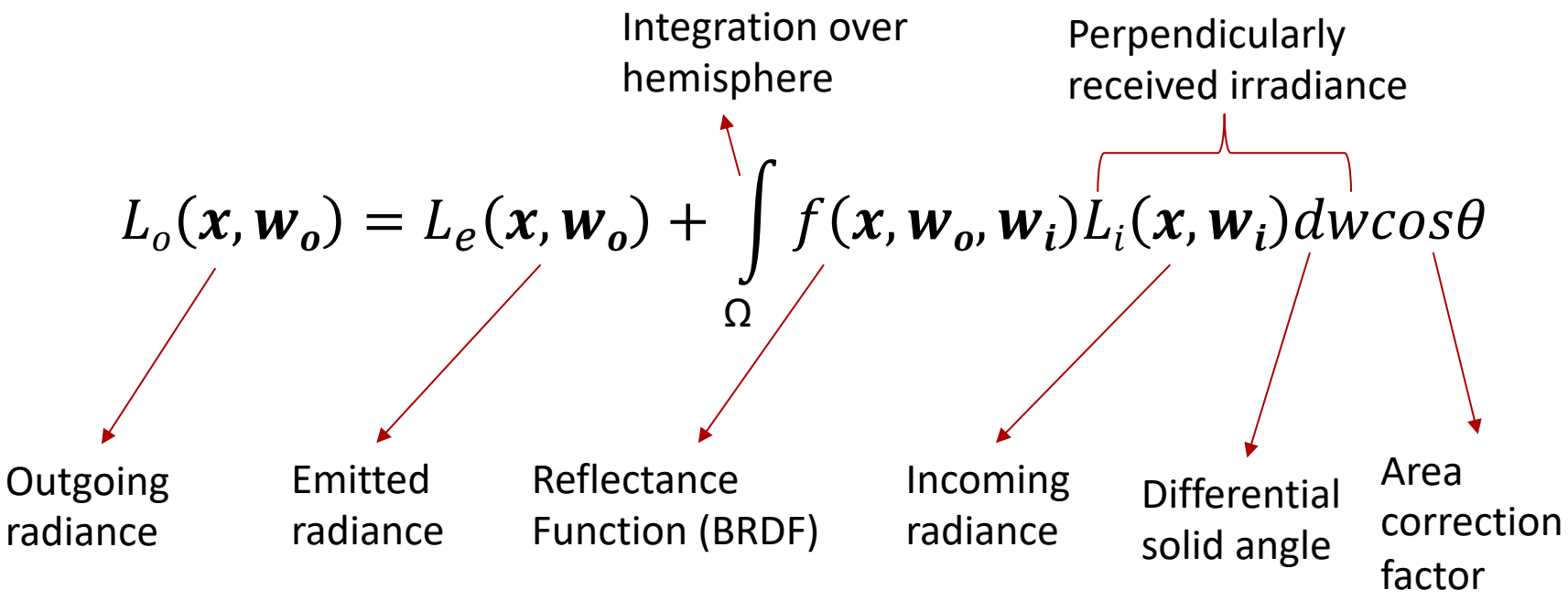
$$cos\theta = \frac{-\boldsymbol{w_i}.\boldsymbol{n}}{|\boldsymbol{w_i}||\boldsymbol{n}|}$$

Rendering Equation

- Putting together all these concepts gives us an equation known as the rendering equation
- It models how much light arriving from an incoming direction ($\mathbf{w_i}$) is reflected toward an outgoing direction ($\mathbf{w_o}$)



Rendering Equation

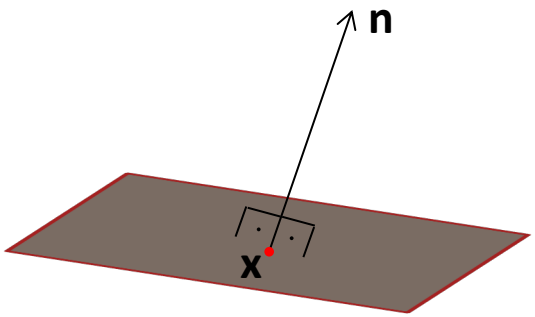


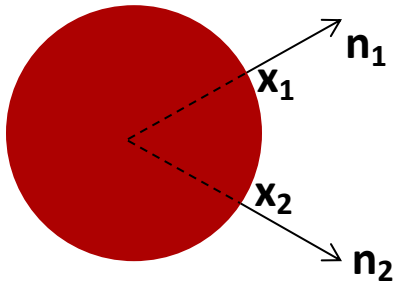
Rendering Equation

- Too costly to evaluate
- For each surface point **x** we need to integrate over the entire hemisphere surrounding **x**
- Simplification:
  - Exclude all directions except light sources
  - Add an ambient term to simulate what was excluded

Surface Normals (Reminder)

- A surface normal is a unit vector which is orthogonal to the surface and points outward from the surface

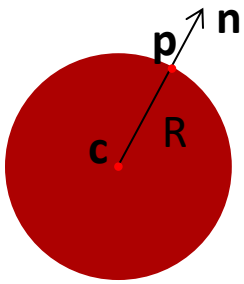




Surface Normals (Reminder)

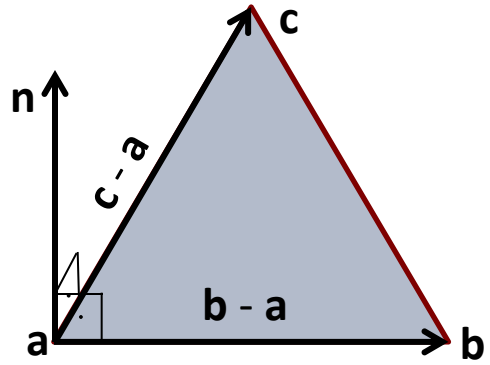
- Depends on the type of the object:
  - For a plane, surface normal is given so you don't have to do anything (except to normalize it if it isn't normalized).
  - For a sphere, compute:

$$\boldsymbol{n} = \frac{\boldsymbol{p} - \boldsymbol{c}}{R}$$



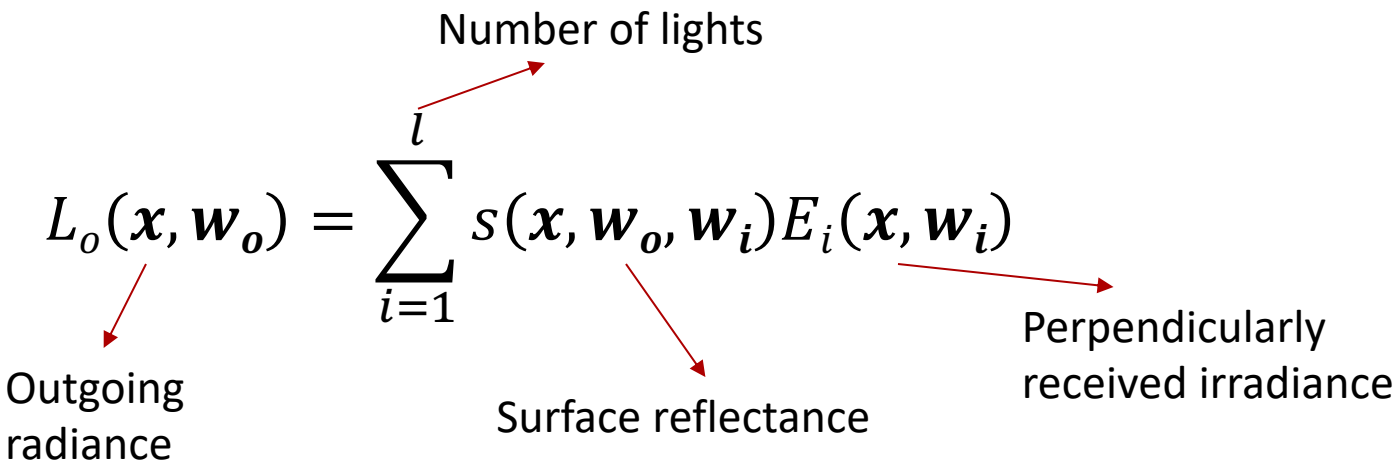
  - For a triangle, compute cross product of two edge vectors:

$$\boldsymbol{n} = \frac{(\boldsymbol{b} - \boldsymbol{a}) x (\boldsymbol{c} - \boldsymbol{a})}{\|(\boldsymbol{b} - \boldsymbol{a}) x (\boldsymbol{c} - \boldsymbol{a})\|}$$



Shading Models

- We will approximate the rendering equation as a sum of three simple shading models:
  - Diffuse shading
  - Specular shading
  - Ambient shading
- All models are based on a modified rendering equation:



Shading Models

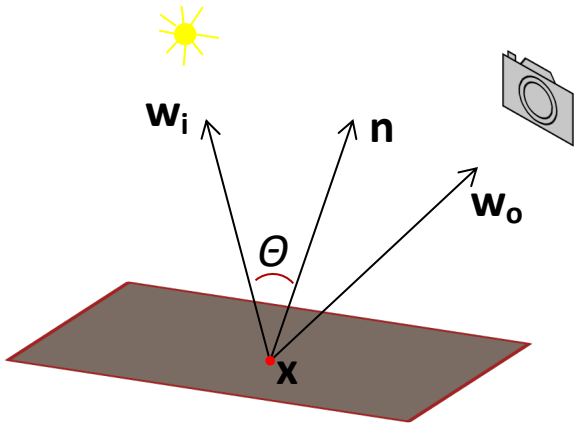
- Compare the surface shading equation with the rendering equation

$$L_o(\boldsymbol{x}, \boldsymbol{w_o}) = \sum_{i=1}^{l} s(\boldsymbol{x}, \boldsymbol{w_o}, \boldsymbol{w_i}) E_i(\boldsymbol{x}, \boldsymbol{w_i})$$

$$L_o(\boldsymbol{x}, \boldsymbol{w_o}) = L_e(\boldsymbol{x}, \boldsymbol{w_o}) + \int_{\Omega} f(\boldsymbol{x}, \boldsymbol{w_o}, \boldsymbol{w_i}) cos\theta L_i(\boldsymbol{x}, \boldsymbol{w_i}) dw$$

Diffuse Shading

- Simulates the phenomenon that a surface can receive radiation only in proportion to its area projected in the direction of the light:



$$L_o^d(x, wo) = k_d cos\theta' E_i(x, w_i)$$

Outgoing radiance

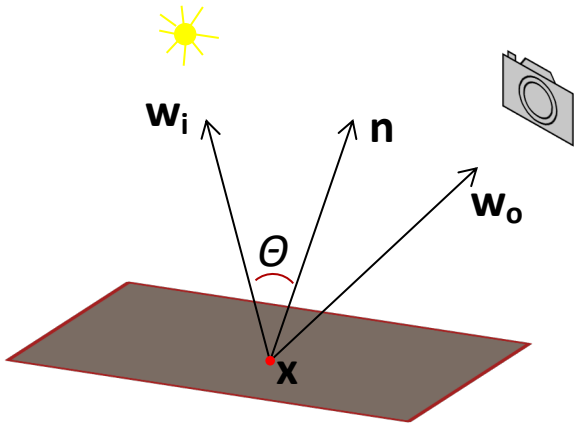
Diffuse reflectance coefficient

Perpendicularly received irradiance

$$cos\theta' = \max(0, w_i . n)$$

Diffuse Shading

- Note that a diffuse surface reflects equal radiance in all directions (looks equally bright from all viewing directions)
- Such a surface is called a Lambertian surface (1760)



$$L_o^d(x, wo) = k_d cos\theta' E_i(x, w_i)$$

Outgoing radiance

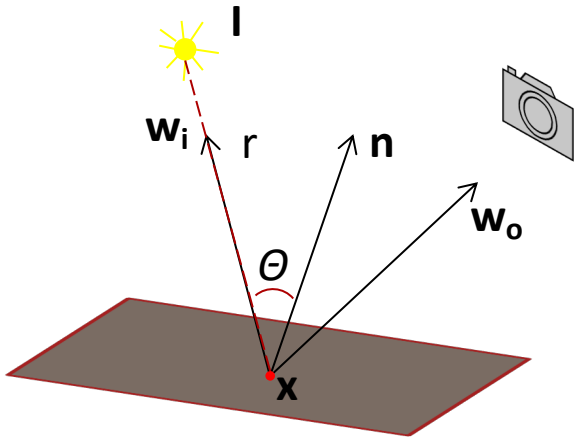
Diffuse reflectance coefficient

Received irradiance

$$cos\theta' = \max(0, w_i . n)$$

Diffuse Shading

- Also, for a point light source, we must compute irradiance at a distance r from the light source: inverse square law
- This improves realism



Light intensity

$$L_o^d(x, wo) = k_d cos\theta' \frac{I}{r^2}$$

Outgoing radiance

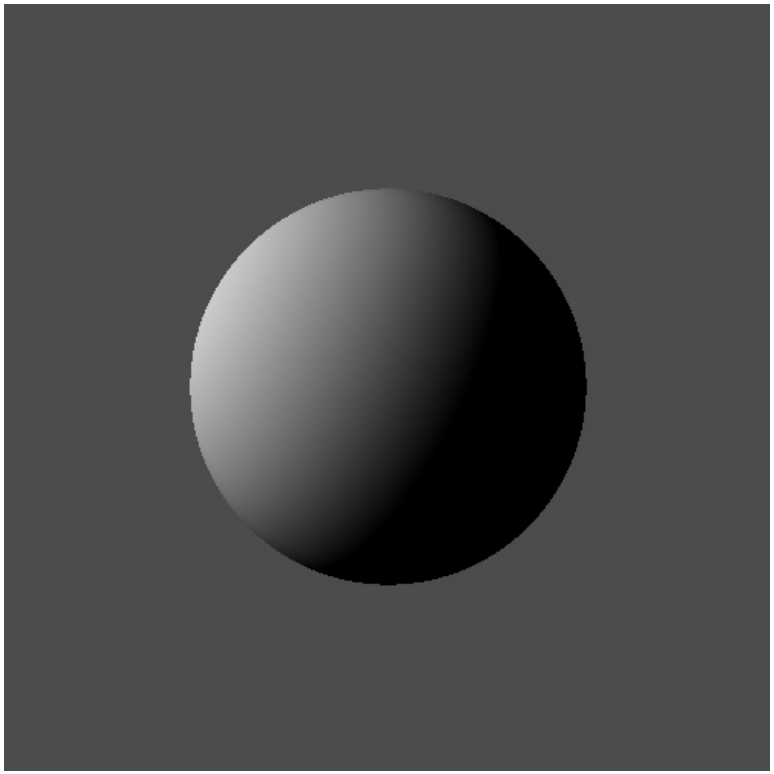
Diffuse reflectance coefficient

Light distance

$$cos\theta' = \max(0, w_i . n)$$

Diffuse Shading Example

- A sphere shaded by diffuse shading:



directxtutorial.com

Ambient Shading

- With diffuse shading parts of the sphere that do not face the light source are all **black**
- Ambient shading is used as a very crude approximate of the integral in rendering equation:

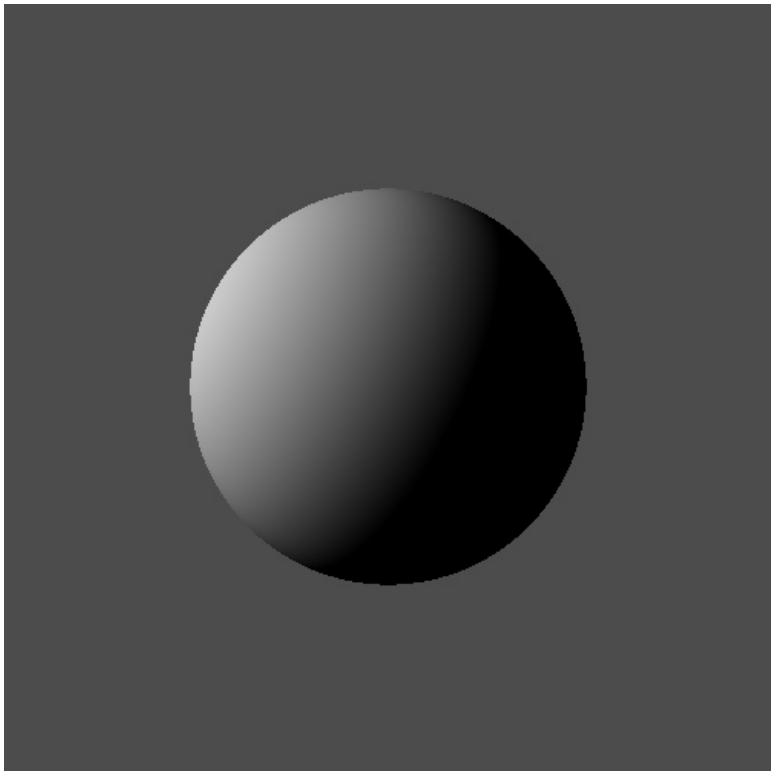
$$L_o^a(x, wo) = k_a I_a$$

Outgoing radiance

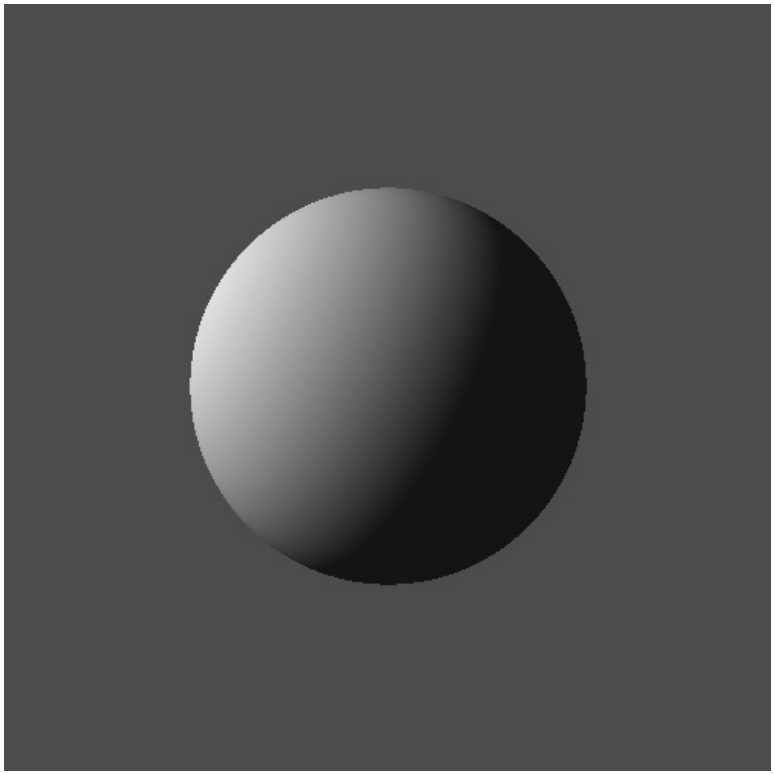
Ambient reflectance coefficient

Ambient radiance

Ambient Shading



Diffuse only



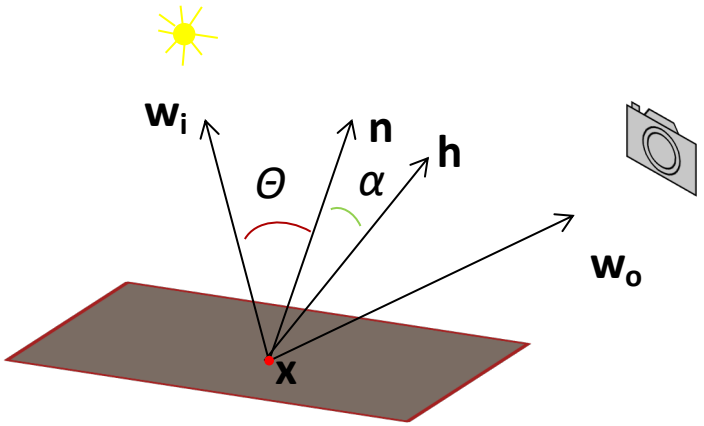
Diffuse + Ambient

Specular Shading

- Simulates shiny surfaces
- Unlike diffuse shading, specular shading is view-dependent
- Several models exist with the most commonly used (and one of the simplest) being Blinn-Phong shading

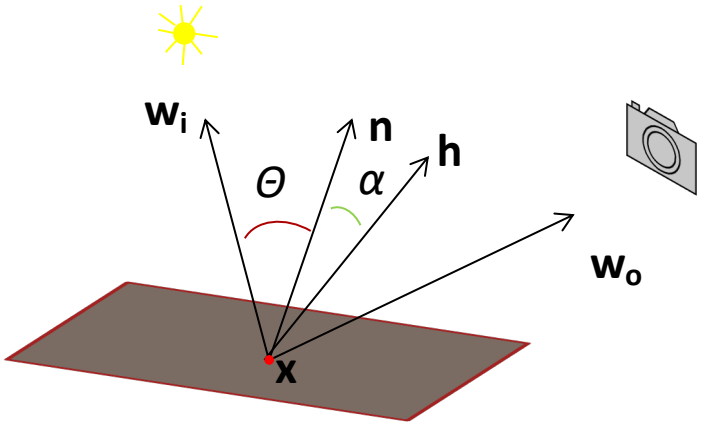
Blinn-Phong (Specular) Shading

- Assumes perfect reflection occurs along the mirror direction of the incoming light
- But for efficiency, a vector known as the half vector is computed
- The angle between the surface normal and the half vector determines the shininess



$$\boldsymbol{h} = \frac{\boldsymbol{w_i} + \boldsymbol{w_o}}{\|\boldsymbol{w_i} + \boldsymbol{w_o}\|}$$

Blinn-Phong (Specular) Shading



$$L_o^s(x, wo) = k_s cos\alpha' E_i(x, w_i)$$

Outgoing radiance

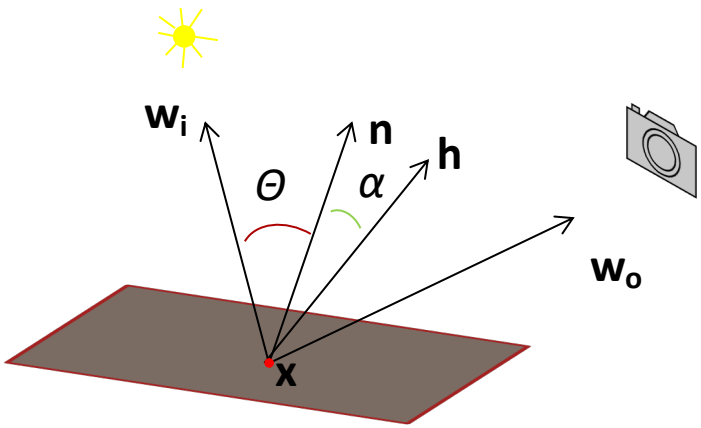
Specular reflectance coefficient

Received irradiance

$$cos\alpha' = \max(0, n.h)$$

Blinn-Phong (Specular) Shading

- To control the shininess, an exponent known as the Phong exponent is introduced:



Phong exponent

$$L_o^s(x, wo) = k_s(\cos\alpha')^p E_i(x, w_i)$$

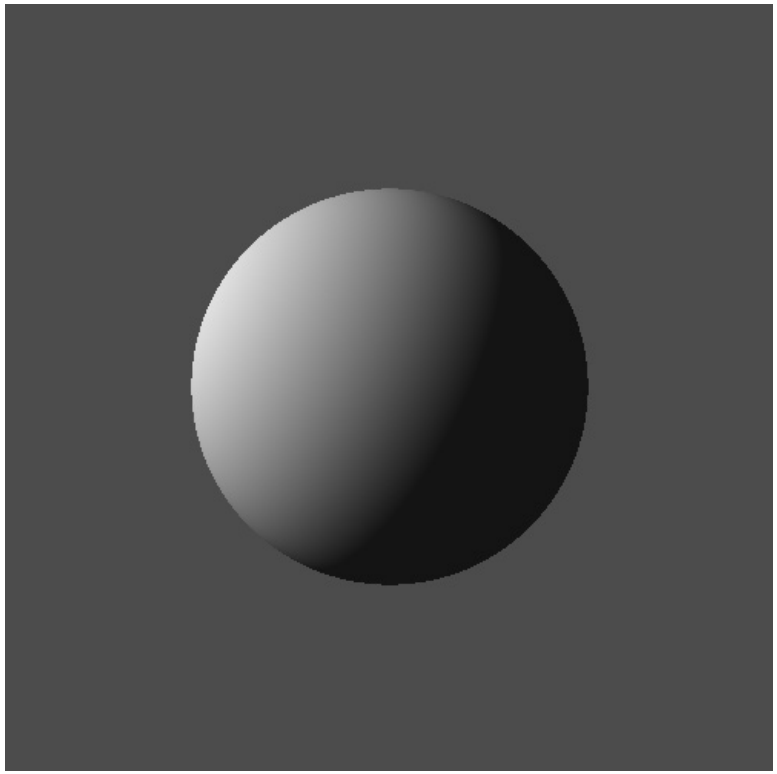
Outgoing radiance

Specular reflectance coefficient

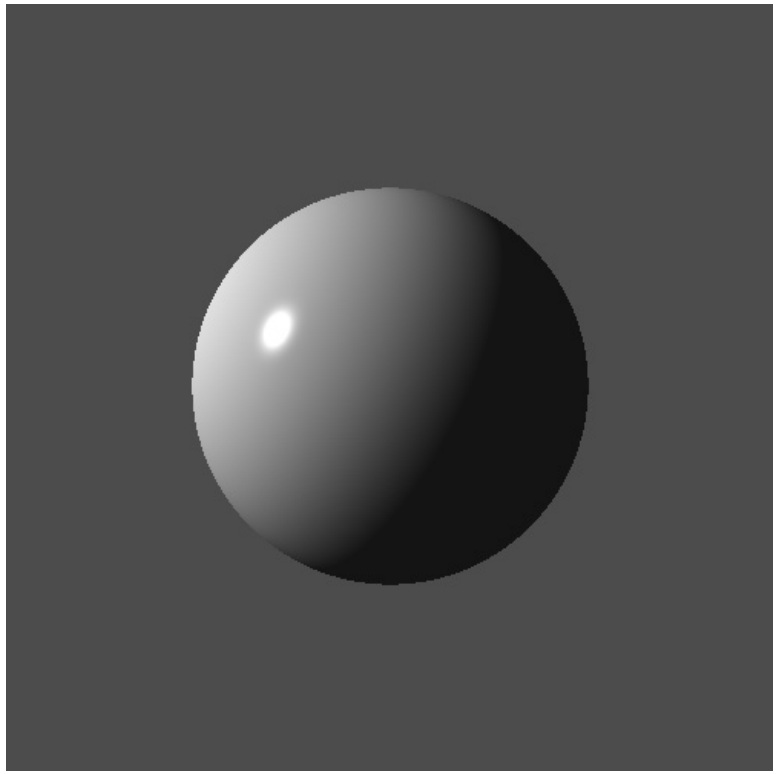
Received irradiance

$$cos\alpha' = \max(0, n.h)$$

Blinn-Phong (Specular) Shading



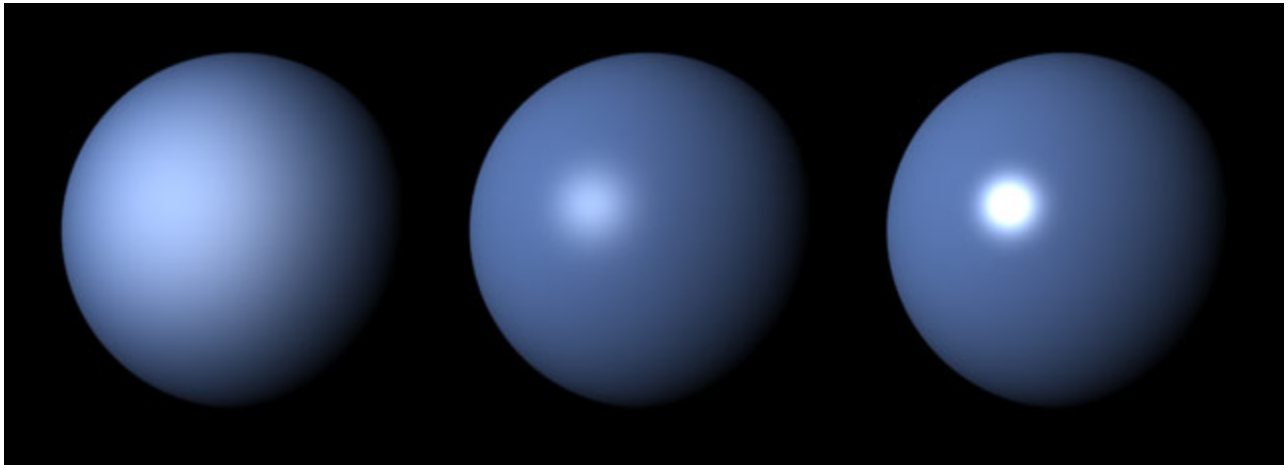
Diffuse + Ambient



Diffuse + Ambient + Specular

Phong Exponent

- Larger Phong exponent makes the highlight more focused, giving the object a shinier appearance



http://cgru.sourceforge.net/

Putting it All Together

- The color of an object at any point can be computed by combining diffuse, ambient, and specular components:

$$\boldsymbol{L_o}\ (\boldsymbol{x}, \boldsymbol{wo}) = \boldsymbol{L_o^d}(\boldsymbol{x}, \boldsymbol{w_o}) + \boldsymbol{L_o^a}(\boldsymbol{x}, \boldsymbol{wo}) + \boldsymbol{L_o^s}(\boldsymbol{x}, \boldsymbol{wo})$$

- Note that **L** is vector-valued in which each component represents the radiance for each color channel

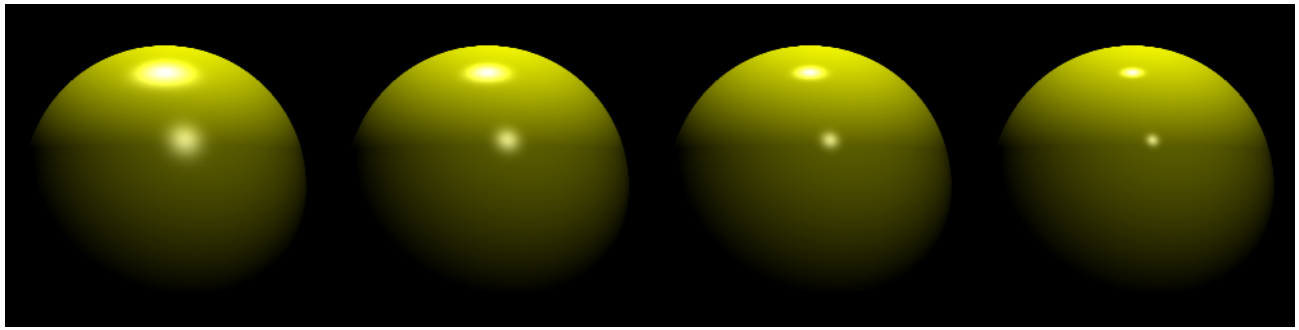
$$\boldsymbol{L_o}(\boldsymbol{x}, \boldsymbol{wo}) = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Multiple Light Sources

- Diffuse and specular contribution of each light source must be accumulated:

Number of light sources

$$\boldsymbol{L_o(x, w_o) = L_o^a(x, w_o) + \sum_{i=1}^{l} L_o^d(x, w_o)_i + L_o^s(x, w_o)_i}$$



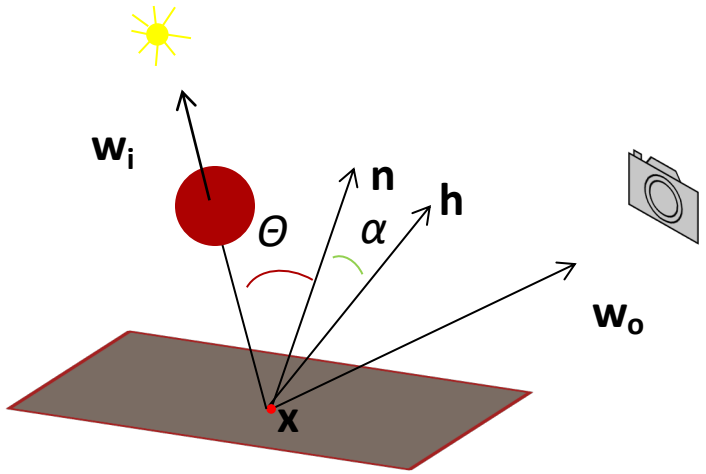
L2program.com

The Ray Tracing Algorithm – So Far

foreach pixel s:
  compute the viewing ray **r** (from **e** to **s**)
  $t_{min} = \infty$, obj = NULL
  foreach object o:
    if r intersects o at point $\boldsymbol{x}$:
      if $t < t_{min}$:
        t_{min} = t, obj = o
  if obj not NULL:
    pixel color = $\boldsymbol{L_o(x, w_o)}$ // $\boldsymbol{w_o}$ is the unit vector from **s** to **e**
  else
    pixel color = color of background (or black)

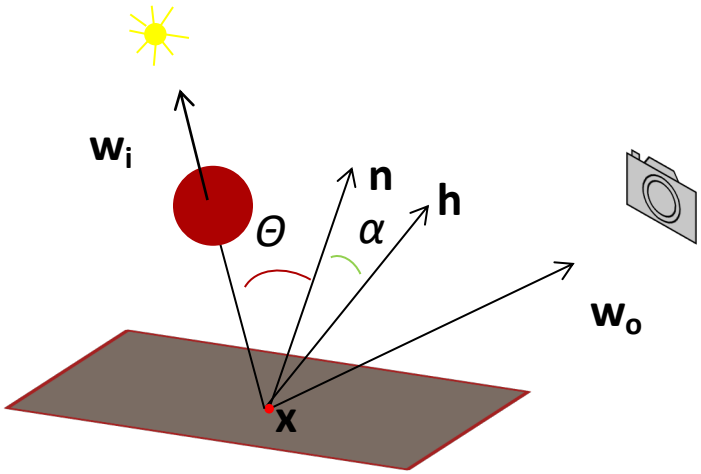
Shadows

- Shadows can easily be added by checking if the light source is visible from the intersection point
- In the configuration below, point ***x*** is in shadow:



Shadows

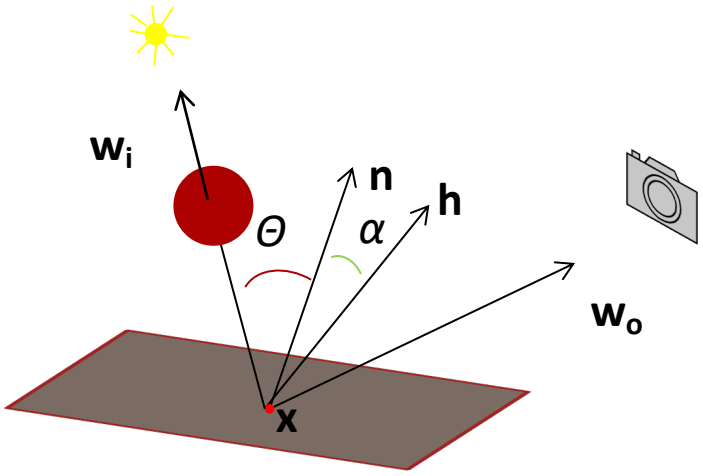
- To check if the light source is visible from the intersection point, we can create shadow rays, one for each light source



$$\boldsymbol{s(t) = x + tw_i}$$

Shadows

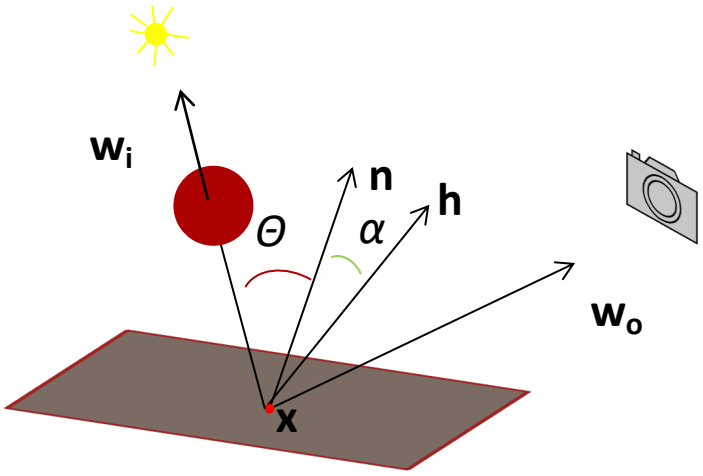
- Intersect s(t) with all the objects in the scene. If there is no intersection before the light source, the point is not in shadow. Otherwise it is in shadow



$$\boldsymbol{s(t) = x + tw_i}$$

Shadows

- To avoid self intersection due to floating point precision problems, the origin is offset by a very small amount:



$$\boldsymbol{s(t)} = (x + \boldsymbol{w_i}\epsilon) + t\boldsymbol{w_i}$$

Epsilon (ϵ) is a very small number such as 0.0001

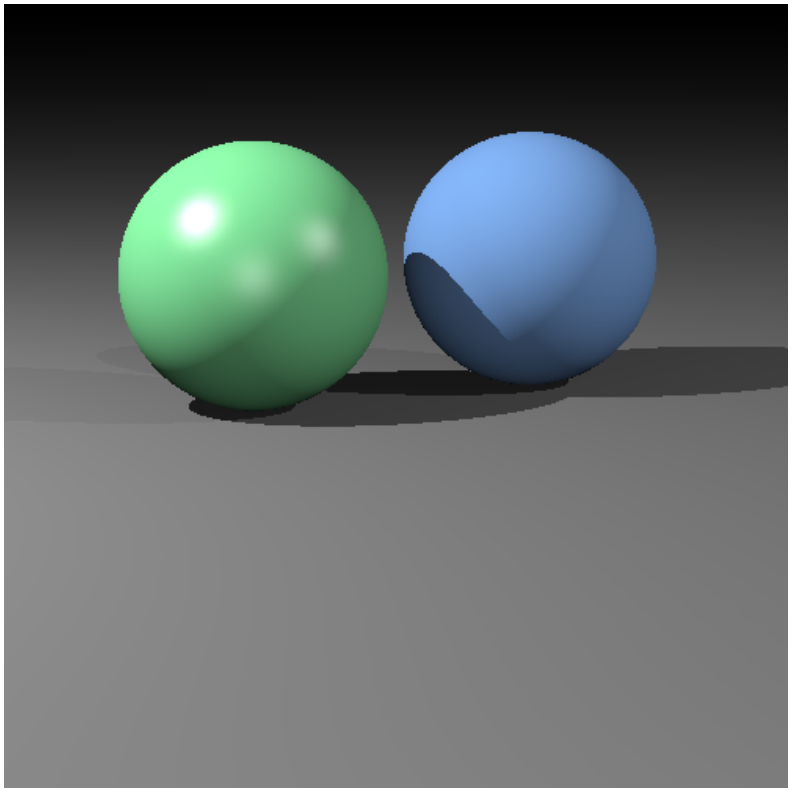
The Ray Tracing Algorithm – So Far

```
foreach pixel s:
    compute the viewing ray r (from e to s)
    t_min = ∞, obj = NULL
    foreach object o:
        if r intersects o at point x:
            if t < t_min:
                t_min = t, obj = o
    if obj not NULL: // viewing ray intersected with an object
        pixel color = L_a // ambient shading is not affected by shadows
        foreach light l:
            compute the shadow ray s from x to l
            foreach object p:
                if s intersects p before the light source:
                    continue the light loop; // point is in shadow – no contribution from this light
            pixel color += L_d + L_s // add diffuse and specular components for this light source
    else
        pixel color = color of background (or black)
```

Do not forget clamping the pixel value to [0, 255] range and rounding it to the nearest integer!

Image So Far

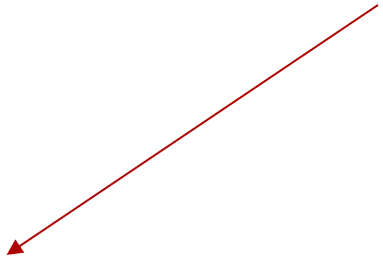
- This looks much better! ☺



Recursive Ray Tracing

- Ray tracing is recursive by nature
- Remember the rendering equation

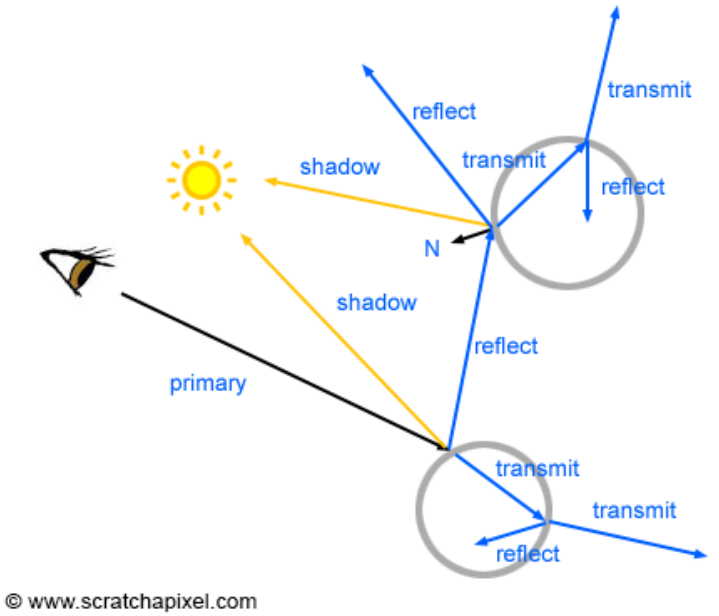
$$L_o(\boldsymbol{x}, \boldsymbol{w_o}) = L_e(\boldsymbol{x}, \boldsymbol{w_o}) + \int f(\boldsymbol{x}, \boldsymbol{w_o}, \boldsymbol{w_i}) L_i(\boldsymbol{x}, \boldsymbol{w_i}) cos\theta dw$$



This term itself could be due to reflection (or refraction) from another surface

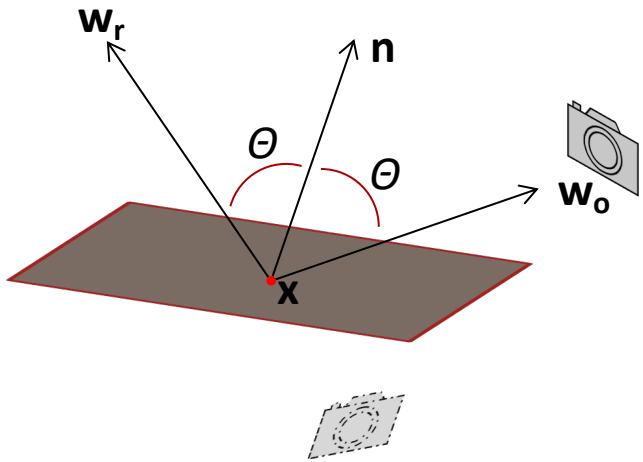
Recursive Ray Tracing

- Tracing the incident ray back into the scene (not just directly towards light sources) is called path tracing
- This is extremely costly so will only do it for perfect mirrors



Ideal Specular Reflection (Mirrors)

- Mirror-like objects reflect colors of other objects
- Given $\mathbf{w_o}$, we must compute $\mathbf{w_r}$ (reflection direction)



$\mathbf{w_r} = -\mathbf{w_o} + 2\mathbf{n}\cos\Theta = -\mathbf{w_o} + 2\mathbf{n}(\mathbf{n}.\mathbf{w_o})$

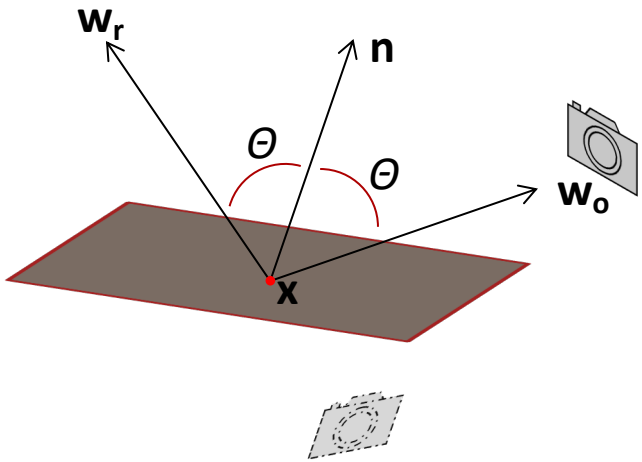
$\mathbf{n}$, $\mathbf{w_o}$, $\mathbf{w_r}$ are all unit vectors

$$L_o^m(x, wo) = k_m L_i(x, w_r)$$

Mirror reflection coefficient

Ideal Specular Reflection (Mirrors)

- For these surfaces the total radiance is equal to local shading radiance plus the radiance from the mirrored location

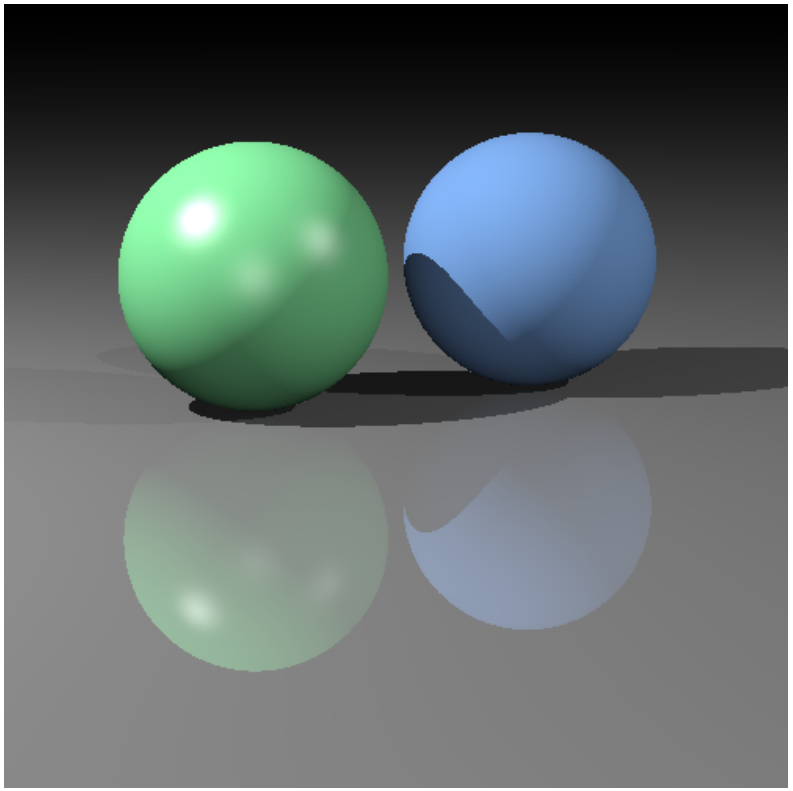


Need to stop after a fixed number of bounces to avoid potentially infinite recursion

$$L_o\,(x, wo) = L_o^d(x, w_o) + L_o^a(x, wo) + L_o^s(x, wo) + L_o^m(x, wo)$$

Final Image

- This looks much better! ☺

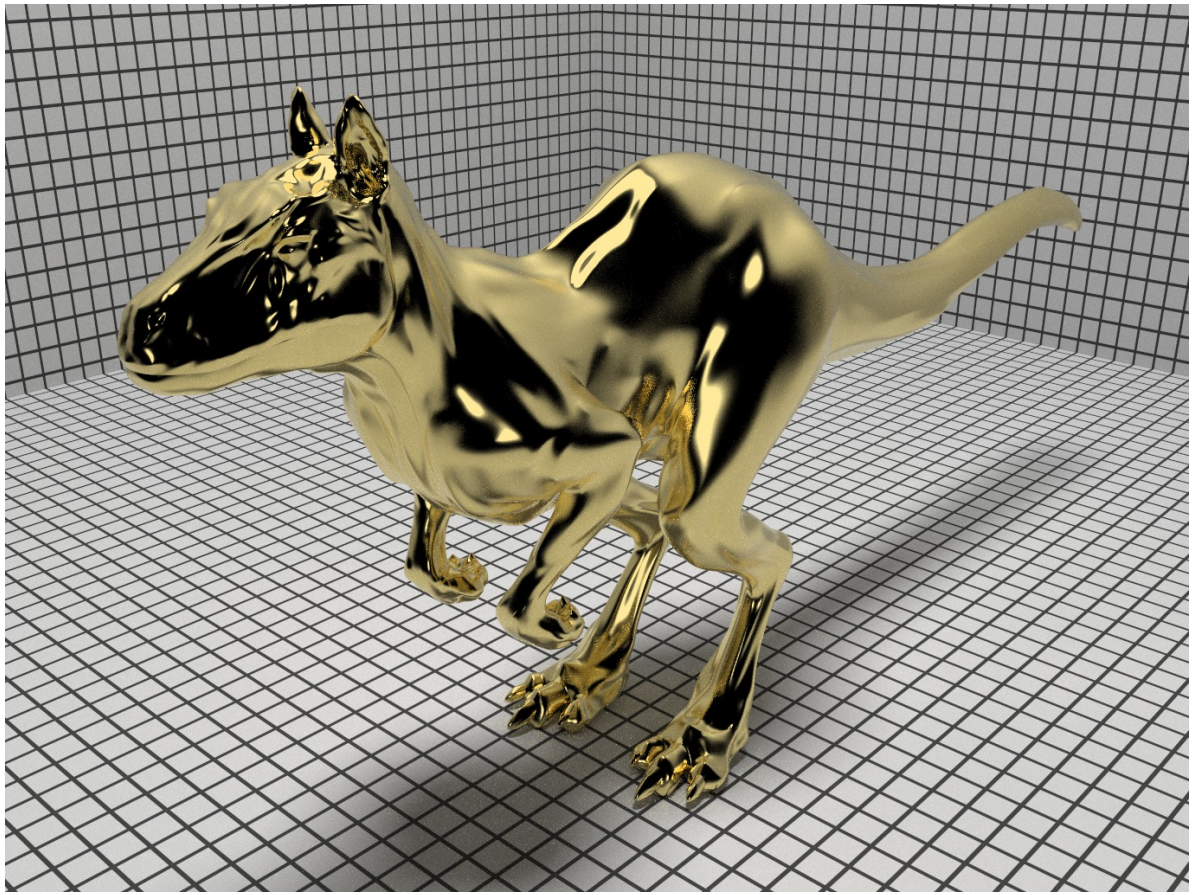


Ray Tracing

- We have only scratched the surface of ray tracing.
- There is so much more that can be done:
  - Acceleration structures
  - Adding texture
  - Modeling non-point light sources (area lights)
  - Refraction (glass-like materials)
  - Participating media (fog, smoke)
  - Subsurface scattering (candles)
  - Complex surface models
  - Parallel and interactive ray tracing
  - Ray tracing on the GPU
  - Photon mapping
  - Global illumination
  - Image-based lighting
  - …

Some Ray-traced Images

Some Ray-traced Images

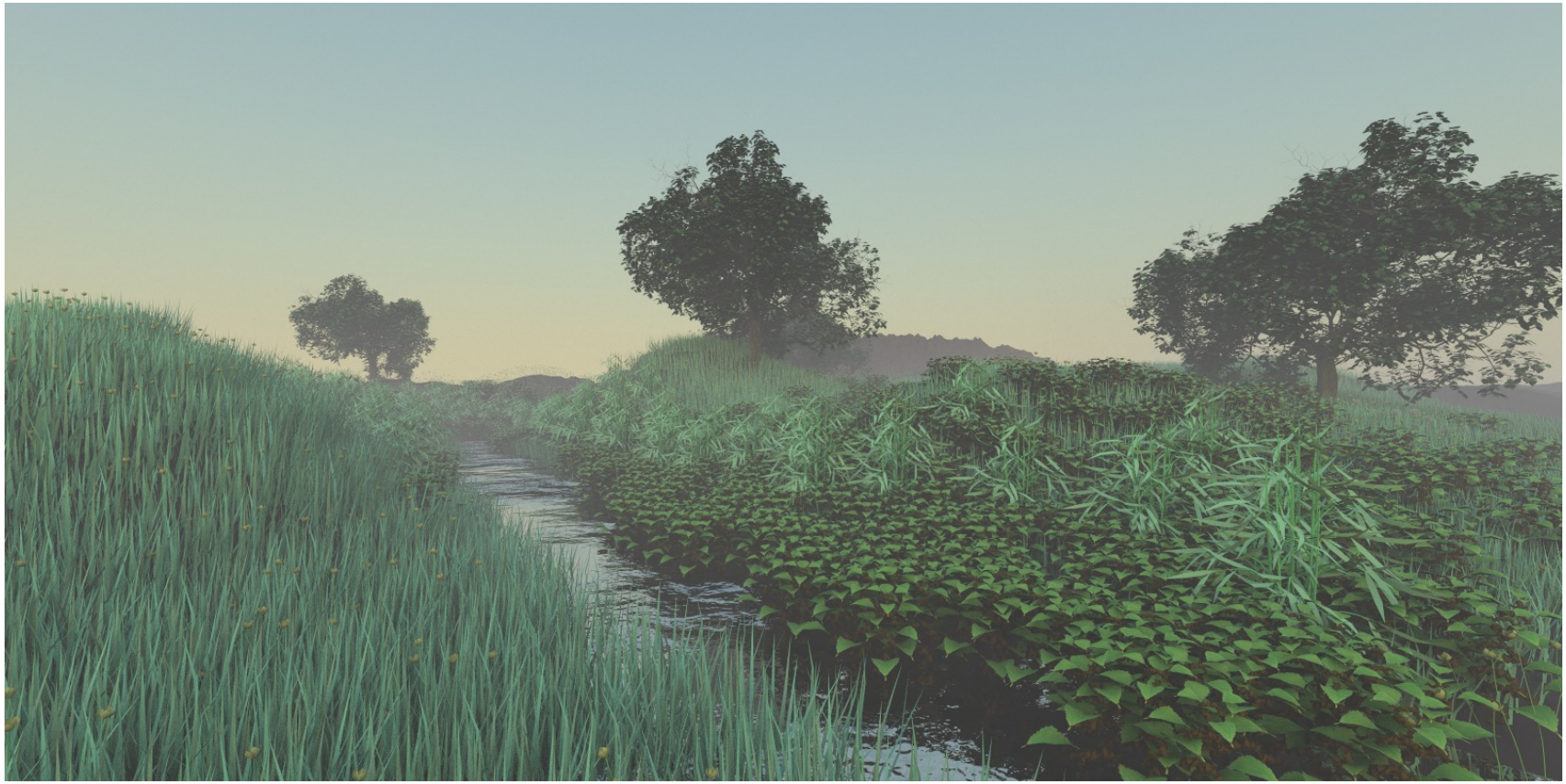


Physically Based Ray Tracer

Some Ray-traced Images

Physically Based Ray Tracer

Some Ray-traced Images



Physically Based Ray Tracer

Some Ray-traced Images

Radiance Ray Tracer

Some Ray-traced Images

METU Ray Tracer