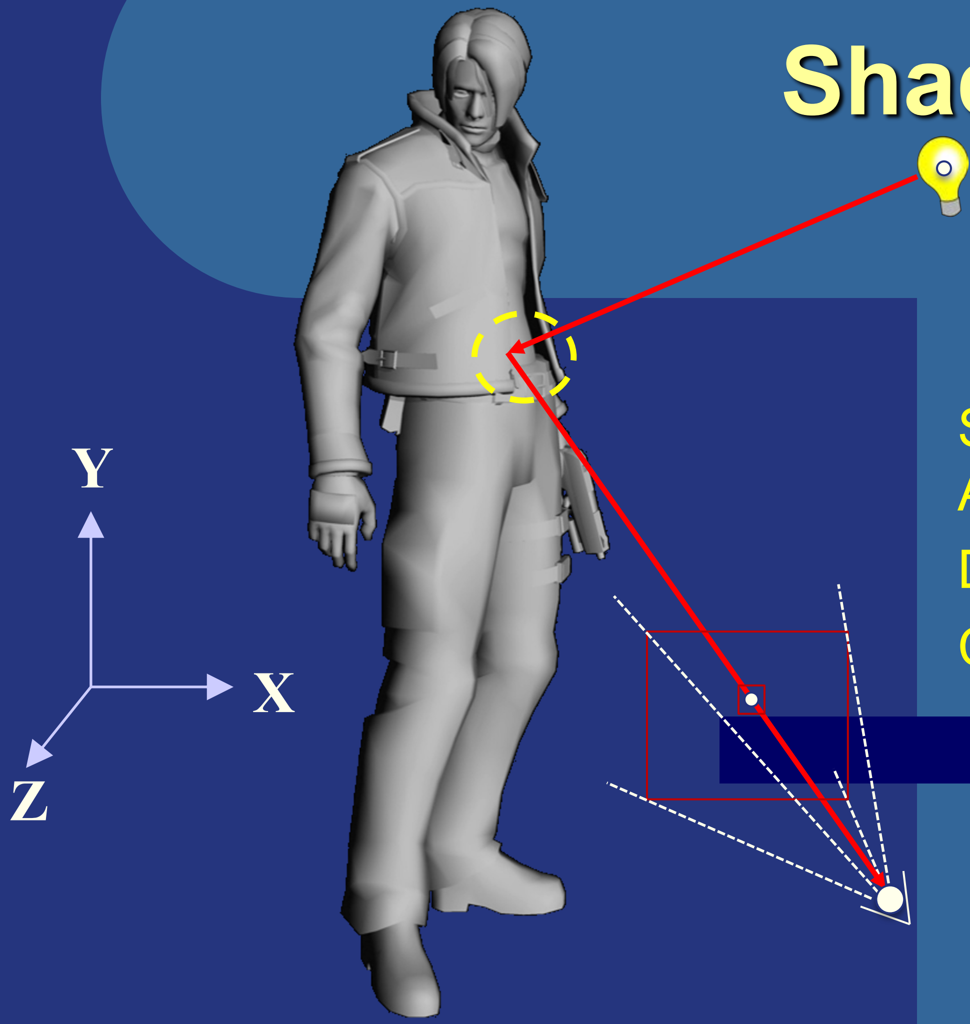
Shading



Slides by Ulf Assarsson and Tomas Akenine-Möller

Department of Computer Engineering

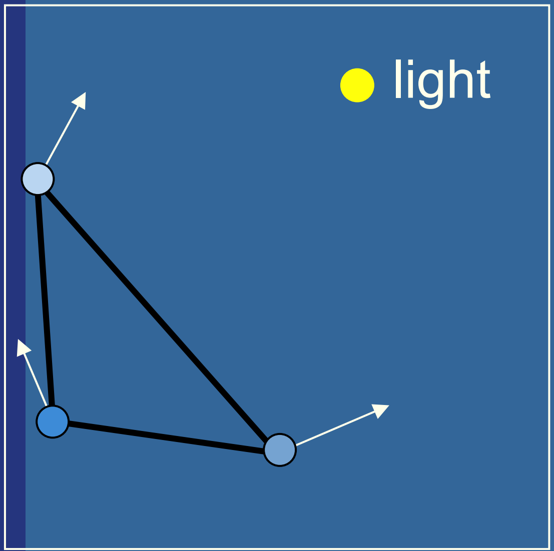
Chalmers University of Technology

Overview of today's lecture

- First, a simple most basic real-time lighting model
  - Shading parts: ambient, diffuse, specular, emission.
    - It is also OpenGL's old fixed pipeline lighting model
- Physically-based shading (PBS)
  - Metalness (vs dielectric) in percent,
  - Fresnel: F_0. (”reflection color”, base reflectance)
  - Specularity: shininess or roughness,
  - Base color: $\boldsymbol{c}_{base}$
- Fog
- Gamma correction
- Transparency and alpha

Lighting and Shading

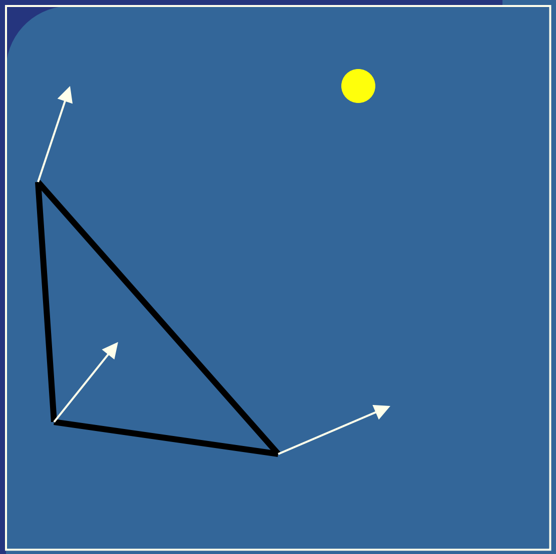
Typically done in the fragment shader.



Lighting computation

Full shading

A basic lighting model



Light: (r,g,b)

DIFFUSE SPECULAR	Base color Highlight Color
AMBIENT	Low-light Color
EMISSION	Glow Color
SHININESS	Surface Smoothness

Material:

- Ambient (r,g,b,a)
- Diffuse (r,g,b,a)
- Specular (r,g,b,a)
- Emission (r,g,b,a) =”self-glowing color”

The ambient/diffuse/specular/emission model

- The most basic real-time model:
- Light interacts with material and change color at bounces:

$$\mathbf{outColor}_{rgb} \sim \mathbf{material}_{rgb} \otimes \mathbf{lightColor}_{rgb}$$

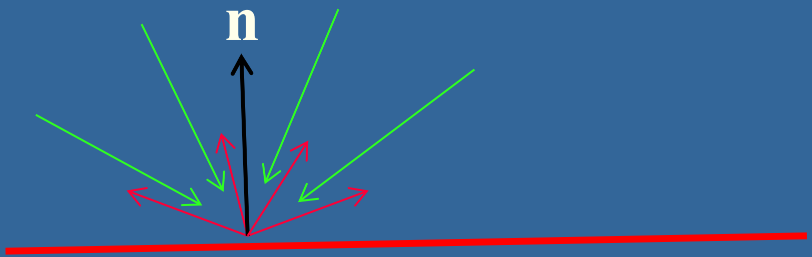
- **Ambient** light: incoming background light from all directions and spreads in all directions (view-independent and light-position independent color)

$$\mathbf{i}_{\text{amb}} = \mathbf{m}_{\text{amb}}\,\mathbf{l}_{\text{amb}}$$

Assuming homogeneous background light

i.e., $(\mathrm{i}_r, \mathrm{i}_g, \mathrm{i}_b) = (\mathrm{m}_r, \mathrm{m}_g, \mathrm{m}_b)\ (\mathrm{l}_r, \mathrm{l}_g, \mathrm{l}_b) = (\mathrm{m}_r \mathrm{l}_r, \mathrm{m}_g \mathrm{l}_g, \mathrm{m}_b \mathrm{l}_b)$


Ambient

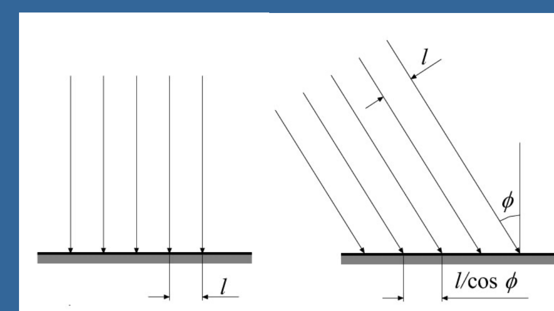


The ambient/diffuse/specular/emission model

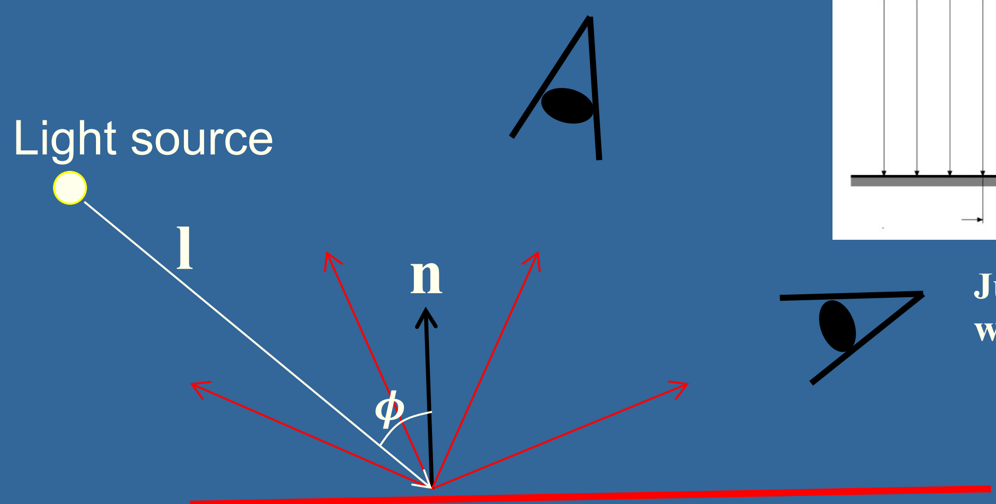
- The most basic real-time model:
- Light interacts with material and change color at bounces:

$$\mathbf{outColor}_{rgb} \sim \mathbf{material}_{rgb} \otimes \mathbf{lightColor}_{rgb}$$

- Ambient light: incoming background light from all directions and spreads in all directions (view-independent and light-position independent color)
- **Diffuse** light: the part that spreads equally in **all** directions (view independent) due to that the surface is very **rough** on microscopic level



Amb + Diff



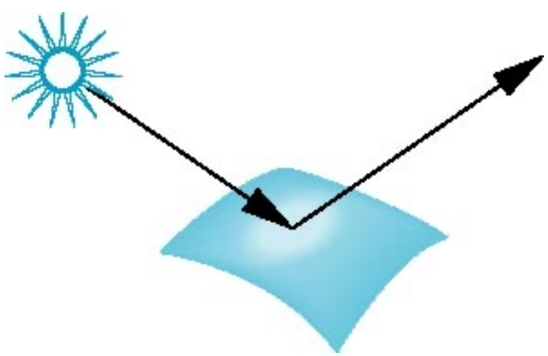
Just scale light intensity with incoming angle

$$\mathbf{i}_{diff} = (\mathbf{n} \cdot \mathbf{l})\mathbf{m}_{diff} \otimes \mathbf{s}_{diff}$$

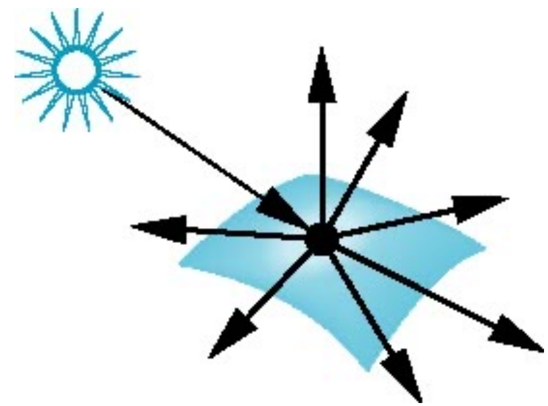
$$(\boldsymbol{n} \cdot \boldsymbol{l}) = \boldsymbol{cos\ \phi}$$

A 100% diffuse material is called a "Lambertian" Surface

- A perfectly diffuse reflector
- Light scattered equally in all directions



Highly reflective surface (specular)



Fully diffuse surface (Lambertian)

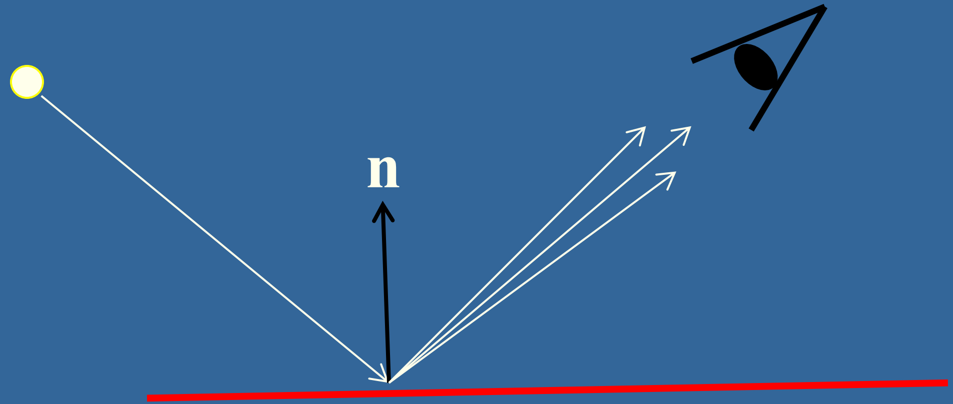
The ambient/diffuse/specular/emission model

- The most basic real-time model:
- Light interacts with material and change color at bounces:

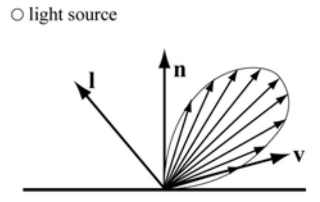
$$\mathbf{outColor}_{rgb} \sim \mathbf{material}_{rgb} \otimes \mathbf{lightColor}_{rgb}$$

- Ambient light: incoming background light from all directions and spreads in all directions (view-independent and light-position independent color)
- Diffuse light: the part that spreads equally in **all** directions (view independent) due to that the surface is very **rough** on microscopic level
- **Specular** light: the part that spreads mostly in the reflection direction (often same color as light source)

Amb + Diff + Spec



Specular: Phong's model

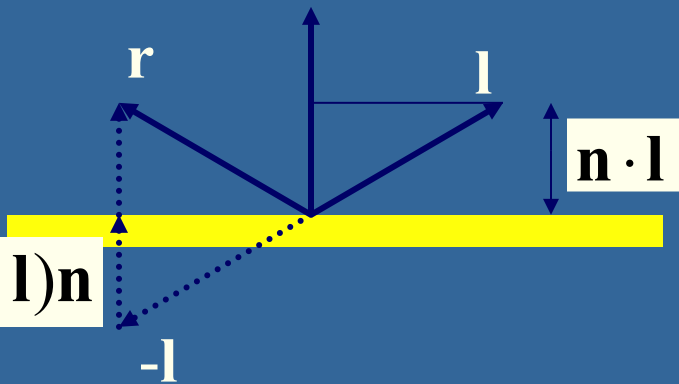


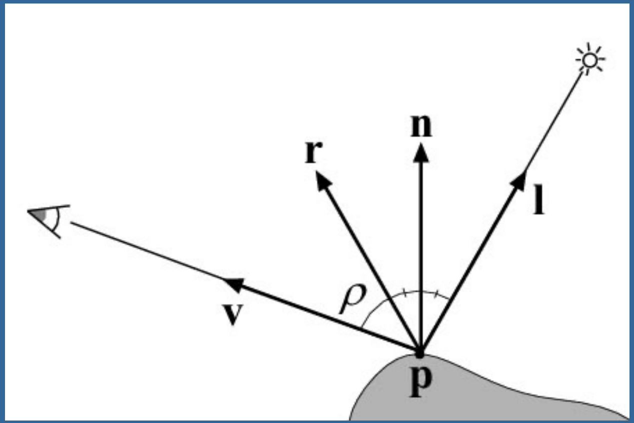
- Phong specular highlight model
- Reflect **l** around **n**:

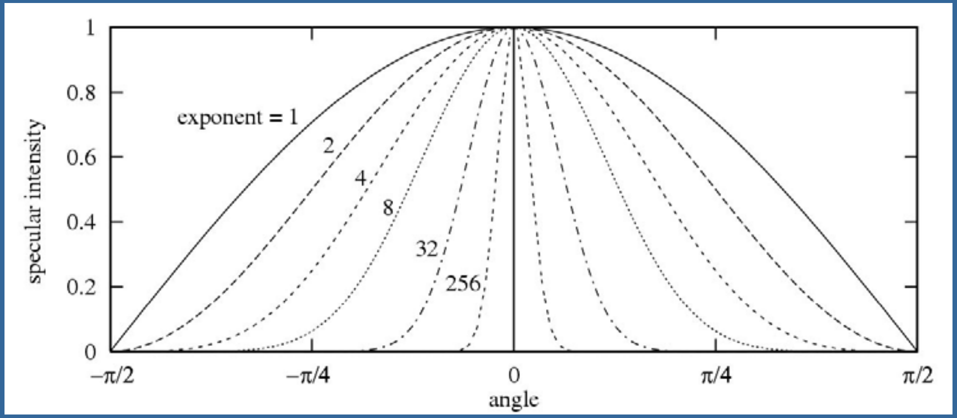
$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{n}\cdot\mathbf{l})\mathbf{n}$$

$$i_{spec} = (\mathbf{r}\cdot\mathbf{v})^{m_{shi}} = (\cos\rho)^{m_{shi}}$$

n must be unit vector







$$\mathbf{i}_{spec} = ((\mathbf{n}\cdot\mathbf{l}) < 0)\ ?\ 0\ :\ \max(0,(\mathbf{r}\cdot\mathbf{v}))^{m_{shi}}\,\mathbf{m}_{spec}\otimes\mathbf{s}_{spec}$$

- Next: Blinns highlight formula: $(\mathbf{n}\cdot\mathbf{h})^{m}$

Specular: Blinn's specular highlight model

Blinn proposed replacing **v·r** by **n·h** where

h = (**l**+**v**)/|**l** + **v**|

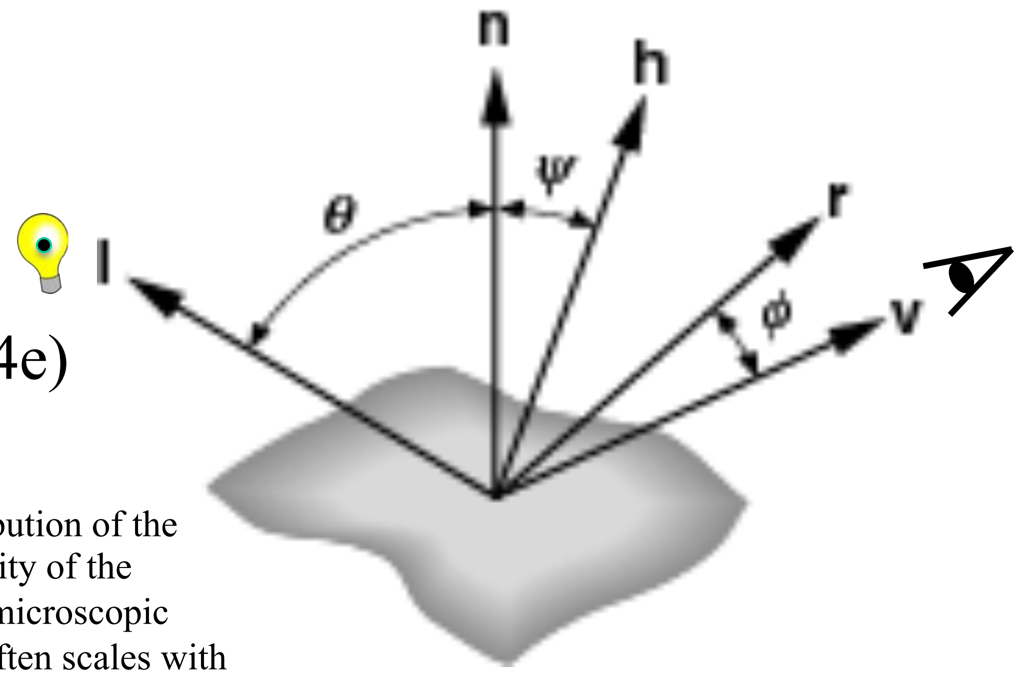
h is halfway between **l** and **v**

If **n**, **l**, and **v** are coplanar:

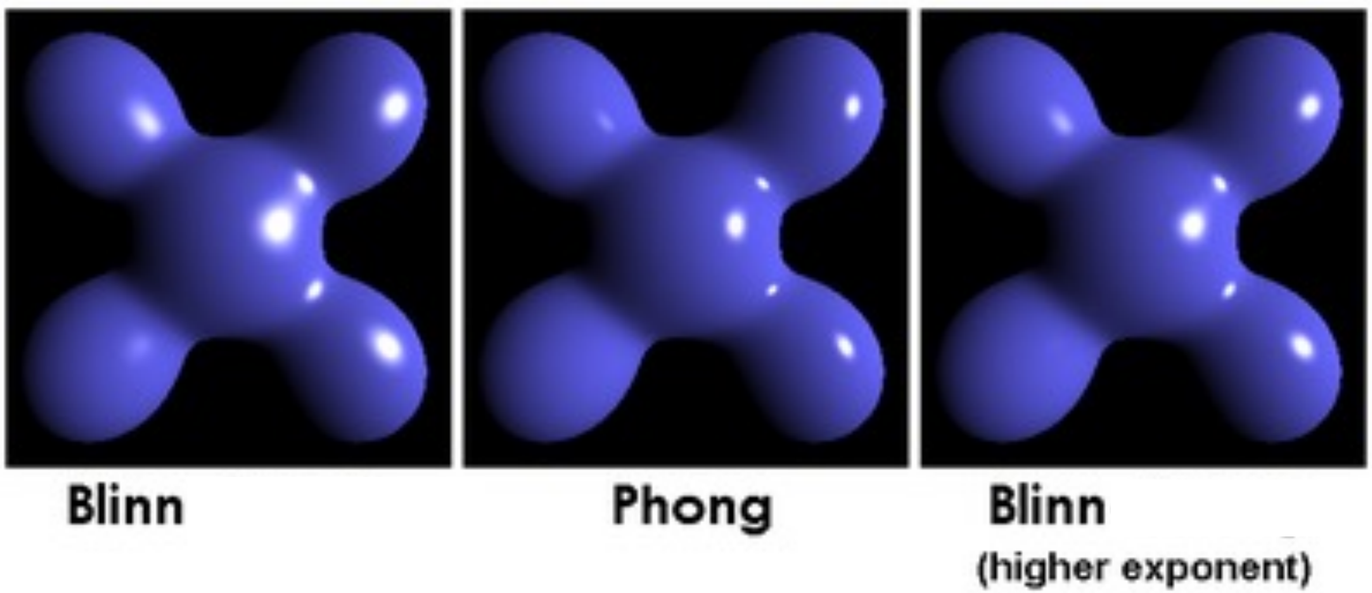
$$\psi = \phi/2$$

Must then adjust exponent

so that $(\mathbf{n}\cdot\mathbf{h})^{e'} \approx (\mathbf{r}\cdot\mathbf{v})^{e}$, $(e' \approx 4e)$



If the surface is rough, there is a probability distribution of the microscopic normals **n**. This means that the intensity of the reflection is decided by how many percent of the microscopic normals are aligned with **h**. And that probability often scales with how close **h** is to the macroscopic surface normal **n**.



$(n \cdot h)^s$ $(r \cdot v)^s$ $(n \cdot h)^{4s}$

The ambient/diffuse/specular/emission model

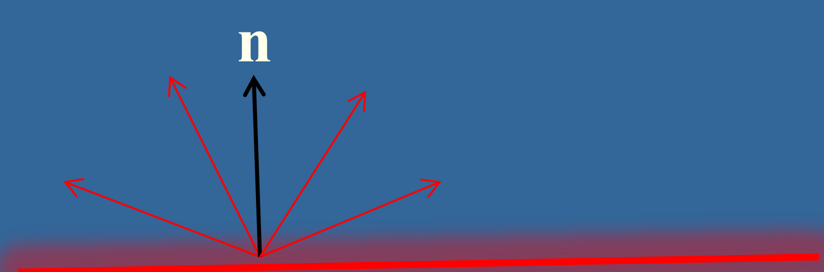
- The most basic real-time model:
- Light interacts with material and change color at bounces:

$$\mathbf{outColor}_{rgb} \sim \mathbf{material}_{rgb} \otimes \mathbf{lightColor}_{rgb}$$

- Ambient light: incoming background light from all directions and spreads in all directions (view-independent and light-position independent color)
- Diffuse light: the part that spreads equally in **all** directions (view independent) due to that the surface is very **rough** on microscopic level
- Specular light: the part that spreads mostly in the reflection direction (often same color as light source)
- **Emission**: self-glowing surface

Amb + Diff + Spec + Em

$$\mathbf{i}_{\mathrm{amb}} = \mathbf{m}_{\mathrm{emission}}$$



The ambient/diffuse/specular/emission model

- Summary of formulas:

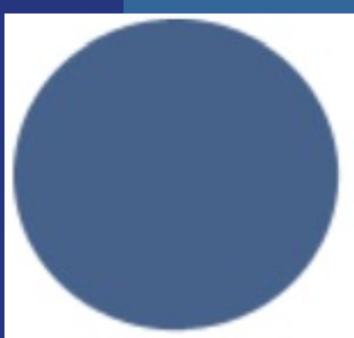
  Ambient: $\mathbf{i}_{amb} = \mathbf{m}_{amb}\,\mathbf{l}_{amb}$

  Diffuse: $(\boldsymbol{n} \cdot \boldsymbol{l})\,\mathbf{m}_{diff}\,\mathbf{l}_{diff}$

  Specular:

  - Phong: $(\boldsymbol{r} \cdot \boldsymbol{v})^{shininess}\,\mathbf{m}_{spec}\,\mathbf{l}_{spec}$
  - Blinn: $(\boldsymbol{n} \cdot \boldsymbol{h})^{shininess}\,\mathbf{m}_{spec}\,\mathbf{l}_{spec}$

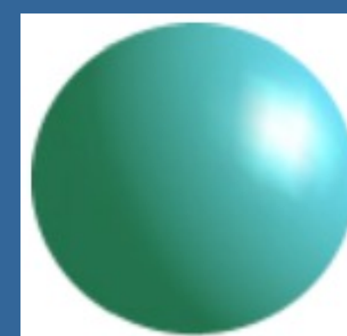
  Emission: $\mathbf{m}_{emission}$



Ambient

Amb + Diff

Amb + Diff + Spec



Amb + Diff + Spec + Em

Physically-based Shading (PBS)

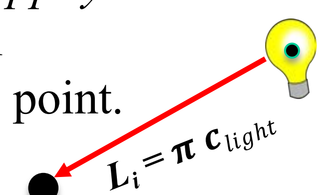
Physically-based Shading (PBS)

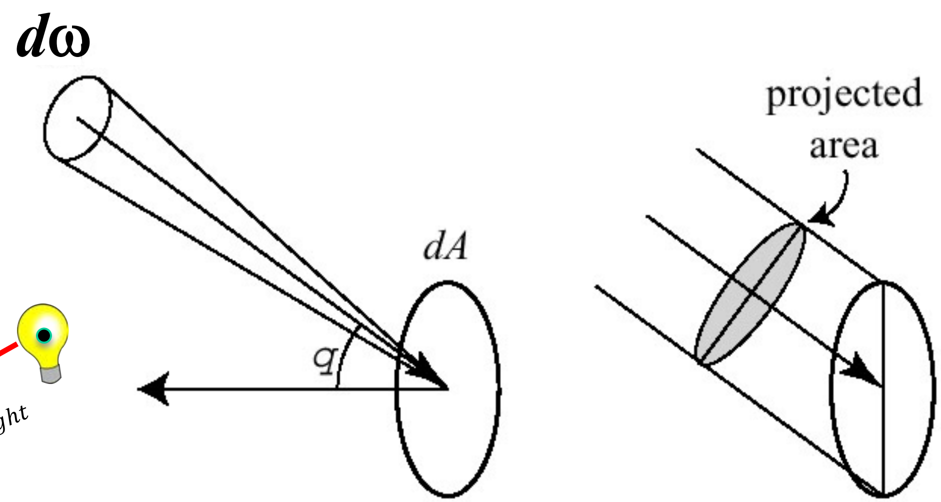
Radiance

- In graphics, we typically use rgb-colors $\boldsymbol{c} = (c_r, c_g, c_b)$ and mean the intensity or ***radiance*** for the red, green, and blue light.
- Radiance, L : a radiometric term. What we store in a pixel is the radiance towards the eye: a tripplet $\boldsymbol{L} = (L_r, L_g, L_b)$
  - Radiance = the amount of electromagnetic radiation leaving or arriving at a point on a surface (per unit solid angle per unit projected area)
- Five-dimensional (or 6, including wavelength):
  - Position (3)
  - Direction (2) – horizontal + vertical angle
- Radiance is ”power per unit projected area per unit solid angle”

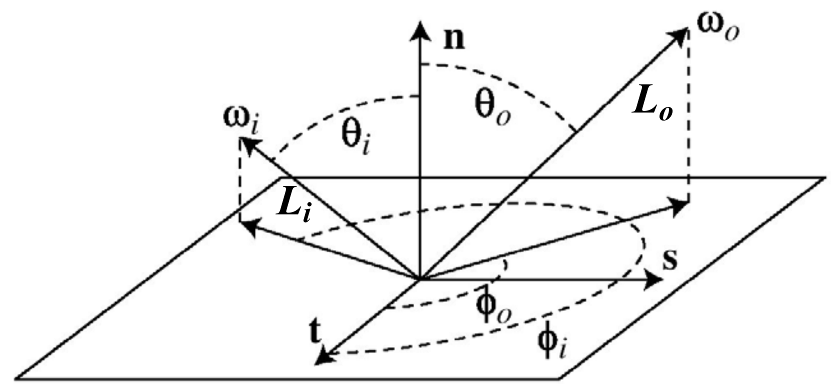
Radiance from a specific *direction* uses differentials, where the cone of the solid angle becomes an infinitesmally thin ray.

Hence, in graphics we often *sloppily* talk about the radiance from an incoming direction to a surface point.





BRDF

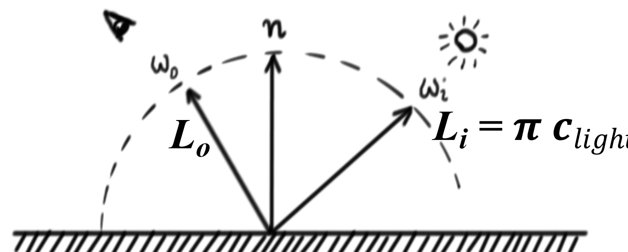


- BRDF = Bidirectional Reflection Distribution Function
- Is a material description, $f(\omega_i, \omega_o)$
- What the BRDF describes: how much of the incoming radiance $\boldsymbol{L}_i$ from a given direction $\boldsymbol{\omega}_i$ that will leave in a given outgoing direction $\boldsymbol{\omega}_o$.

How to compute color, i.e outgoing radiance $\boldsymbol{L}_o$ from a point light:

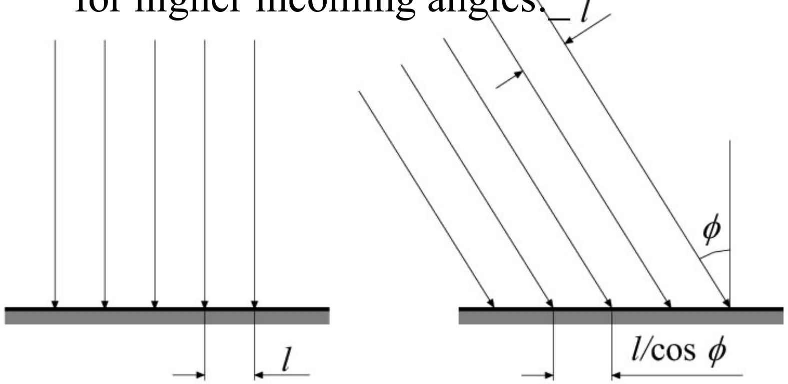
$$\boldsymbol{L}_o(\boldsymbol{\omega}_o) = f(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\boldsymbol{L}_i(\omega_i)(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)$$

$$\boldsymbol{L}_o(\boldsymbol{\omega}_o) = f(\boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\,\pi \boldsymbol{c}_{light}(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)$$



where π comes from that the definition of radiance uses differentials $d\omega_i$ and integrates a cosine factor $(\boldsymbol{n} \cdot \boldsymbol{\omega}_i)$ for the hemisphere. The brdf, *f()*, contains a division by π, which cancel out π.

The cosine comes from decreased incoming intensity for higher incoming angles:



A fully diffuse (Lambertian) brdf is then:

$$f(\omega_i, \omega_o) = \frac{\boldsymbol{c}_{diff}}{\pi}$$

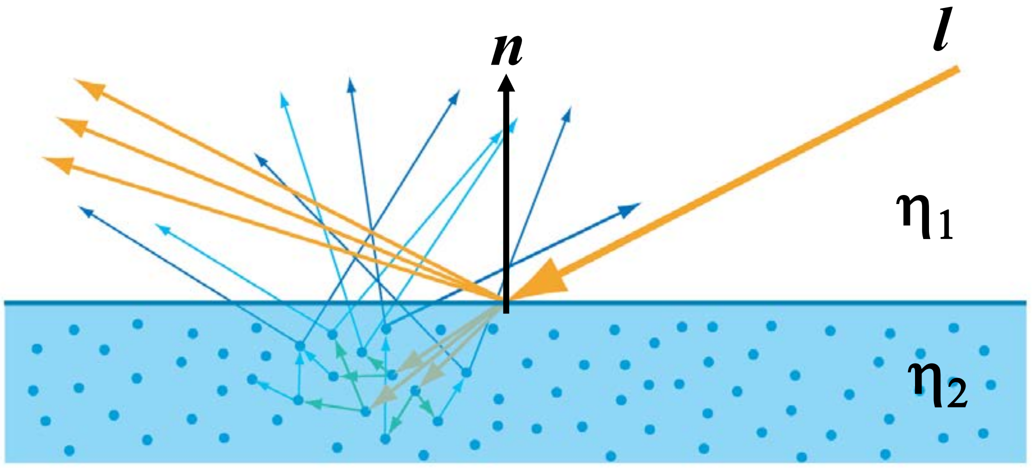
=>

diff color: $\boldsymbol{L}_o(\omega_o) = \boldsymbol{c}_{diff}\, \boldsymbol{c}_{light}(n \cdot \omega_i)$

Surfaces and materials

A common surface model:

- Some amount of incoming light from direction ω_i :
  - reflects to various outgoing directions (yellow).
  - refracts into the material, bounces around, gets color tinted, and refracts out as a fully diffuse reflection (blue), where he color tint is created by absorption.

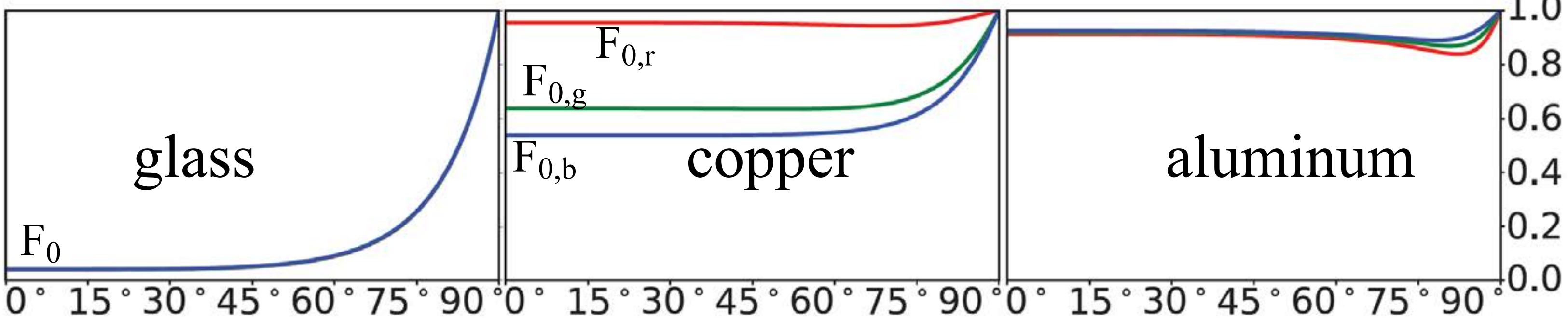


- The Fresnell equations describe how much of the incoming light that reflects or refracts. F() depends on the relative refraction index $\boldsymbol{\eta = \eta_1/\eta_2}$ and the incoming angle to the surface.

$$\boldsymbol{F}(\boldsymbol{n},\boldsymbol{l}) \approx \boldsymbol{F}_0 + (1-\boldsymbol{F}_0)\big(1-(\boldsymbol{n}\cdot\boldsymbol{l})\big)^5 \quad \text{where } F_0 = \left(\frac{\eta_1-\eta_2}{\eta_1+\eta_2}\right)^2$$

called Schlick's approximation.

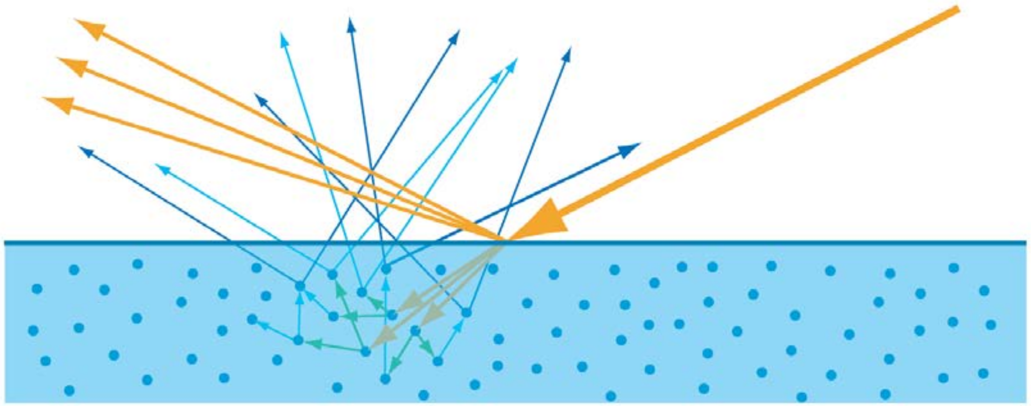
F_0 is also wavelength dependent: highly for metals, not so for dielectrics.



Surfaces and materials – dielectrics vs metals

Materials:

- **Dielectrics:**
  - The glossy reflection has the light's color.
  - The diffuse reflection is colored by the material
  - Ex: glass, skin, wood, hair, leather, plastic, stone, concrete, water,



Dielectric	Linear	Texture	Color	Notes
Water	0.02	39		
Living tissue	0.02–0.04	39–56		Watery tissues are toward the lower bound, dry ones are higher
Skin	0.028	47		
Eyes	0.025	44		Dry cornea (tears have a similar value to water)
Hair	0.046	61		

- **Metals:** has **only** reflection, no refraction (so no diffuse component)

Metal	Linear	Texture	Color
Titanium	0.542,0.497,0.449	194,187,179	
Chromium	0.549,0.556,0.554	196,197,196	
Iron	0.562,0.565,0.578	198,198,200	
Nickel	0.660,0.609,0.526	212,205,192	
Platinum	0.673,0.637,0.585	214,209,201	
Copper	0.955,0.638,0.538	250,209,194	
Palladium	0.733,0.697,0.652	222,217,211	
Mercury	0.781,0.780,0.778	229,228,228	
Brass (C260)	0.910,0.778,0.423	245,228,174	
Zinc	0.664,0.824,0.850	213,234,237	
Gold	1.000,0.782,0.344	255,229,158	
Aluminum	0.913,0.922,0.924	245,246,246	
Silver	0.972,0.960,0.915	252,250,245	

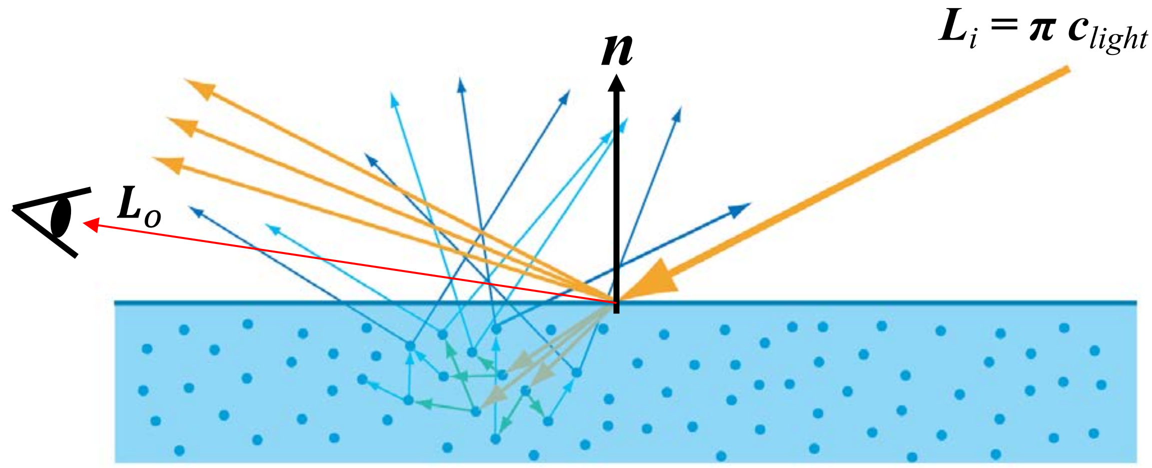
F_0 values p:322-323.

Example of material parameters:

- Metalness (vs dielectric). In percent.
  - Allows layered mtrls, e.g., metal w. lacquer layer
- shininess [0,∞] (or roughness [0,1])
- Fresnel F_0. p:322-323.
- Base_color: $\boldsymbol{c}_{base}$

A physically-based shading model

Putting it together…



Parameters:

- ***Metalness*** (vs dielectric in percent),
- ***Fresnel*** F_0, // tint by angle
- ***Specularity***: shininess or roughness,
- ***Base color***: c_{base} // tint by absorbtion

Formulas:

$L_i = \pi c_{light} * 1/r^2$ // *point light*

$L_i = \pi c_{light}$ // *directional light*

Fresnell effect: $F(n, l) \approx F_0 + (1 - F_0)(1 - (n \cdot l))^5$

diffuse_brdf = $\frac{c_{base}}{\pi}$

metal_brdf = $\frac{G(\omega_i,\omega_o)D(\omega_h)F(\omega_i)}{|n \cdot \omega_o||n \cdot \omega_i|}$ * c_{base}

metal reflection is colored by material

dielectric_brdf = $\frac{G(\omega_i,\omega_o)D(\omega_h)F(\omega_i)}{|n \cdot \omega_o||n \cdot \omega_i|}$ * **vec3(1)** + **(1-F) diffuse_brdf**

but not so for dielectrics

tot_brdf = metalness * **metal_brdf** + **(**1 - metalness**)** * **dielectric_brdf**

Specular reflection

roughness roughness

$$\frac{G(\omega_i,\omega_o)D(\omega_h)F(\omega_i)}{|n \cdot \omega_o||n \cdot \omega_i|}$$

Is explained by Erik… see his online videos 1 2 3 4.

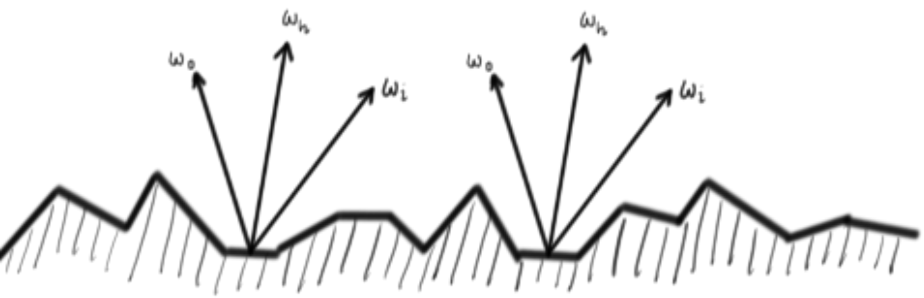
TOTAL: $L_o(\omega_o) = \sum_{i=1}^{\#lights} (\textbf{tot\_brdf})\,(L_i)\,(n \cdot \omega_i)$

A physically-based shading model

Is explained by Erik… see his online videos 1 2 3 4.

$$\frac{G(\omega_i,\omega_o)D(\omega_h)F(\omega_i)}{|n\cdot\omega_o||n\cdot\omega_i|}$$

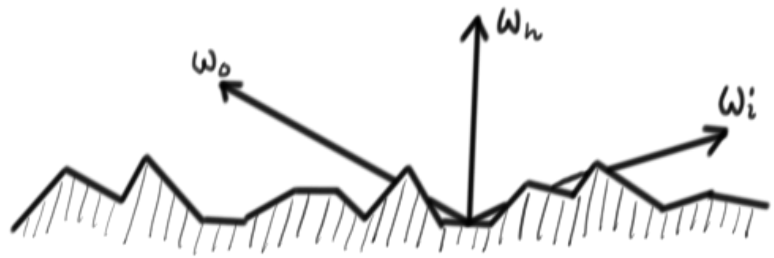
Specular lobe width:



Only the microfacets whose normal is ω_h will reflect in direction ω_o. $D(\omega_h)$ gives us the density of such facets.

$$\omega_h = \texttt{normalize}(\ \omega_i + \omega_o\ )$$
$$s = \texttt{material\_shininess}$$
$$D(\omega_h) = \frac{(s+2)}{2\pi}(n\cdot\omega_h)^s$$

Specular self-shadowing:



When ω_o or ω_i are at grazing angles, radiance might be blocked by other microfacets. This is what $G(\omega_i,\omega_o)$ models.

$$G(\omega_i,\omega_o) = \texttt{min}(1,\ \texttt{min}(\ 2\frac{(n\cdot\omega_h)(n\cdot\omega_o)}{\omega_o\cdot\omega_h}\ ,\ 2\frac{(n\cdot\omega_h)(n\cdot\omega_i)}{\omega_o\cdot\omega_h}\ ))$$

Clamp to [0,1] and avoid denominators ≈ 0

Extra… (bonus)

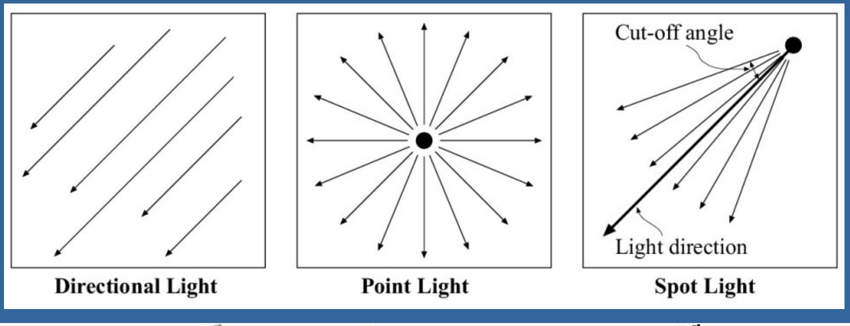
- Anisotropic Normal Distribution Functions – update D() in the microfacet model - see p343
- Multibounce surface reflections – p:346
- Subsurface Scattering: p:347 – modify the Lambertian brdf and the Fresnell factor.
- Cloth brdf:s – p:356

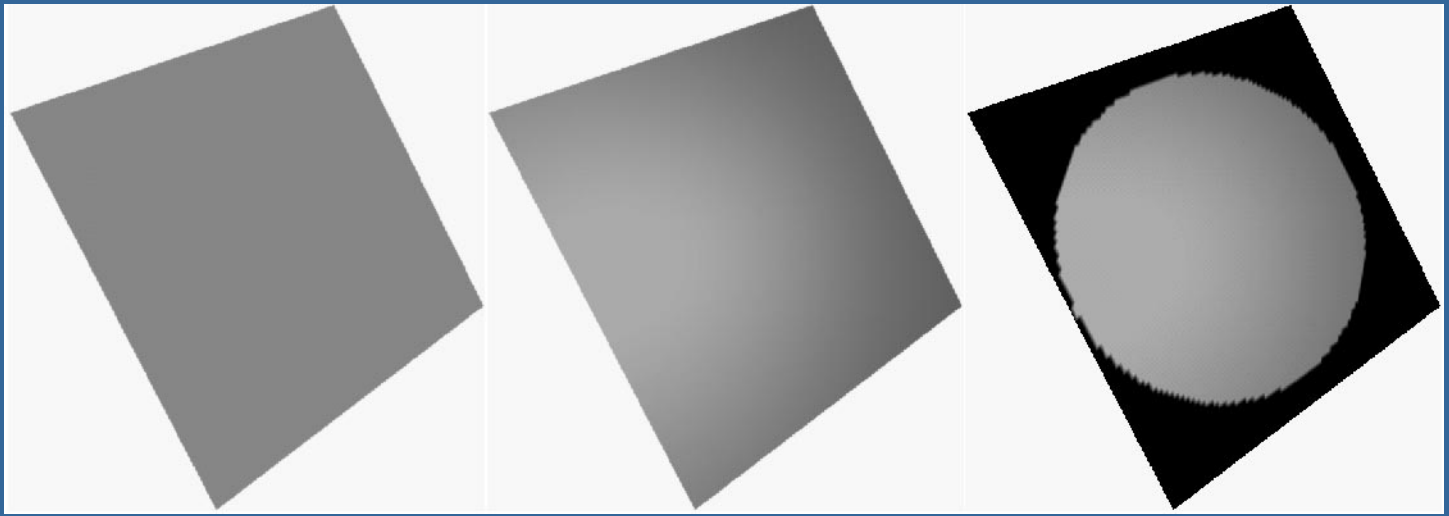
- Light falloff: page 111, Unreal, Frostbite + CryEngine
- Distance falloff function / windowing function: Just Cause 2
- Lambertian brdf. = diffuse color = albedo.
- Some light source types: point lights, area lights,
  - Incoming light from the surrounding can be captured by environment maps,

Environment maps (reflection maps)

Additions to the lighting equation

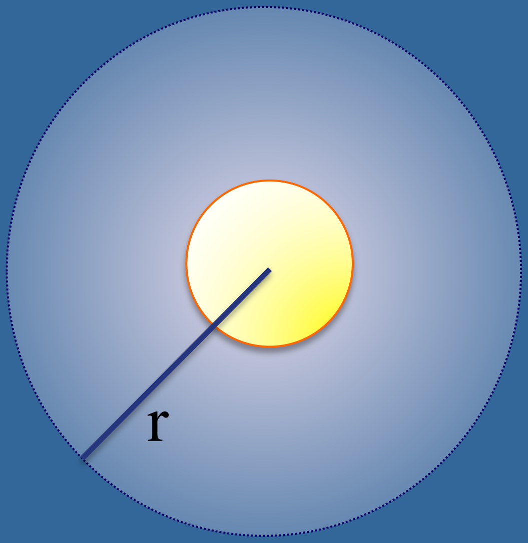
- Accounting for distance: $1/(a+bt+ct^2)$
- Several lights: just sum their respective contributions
- Different light types:





Clarification on accounting for distance

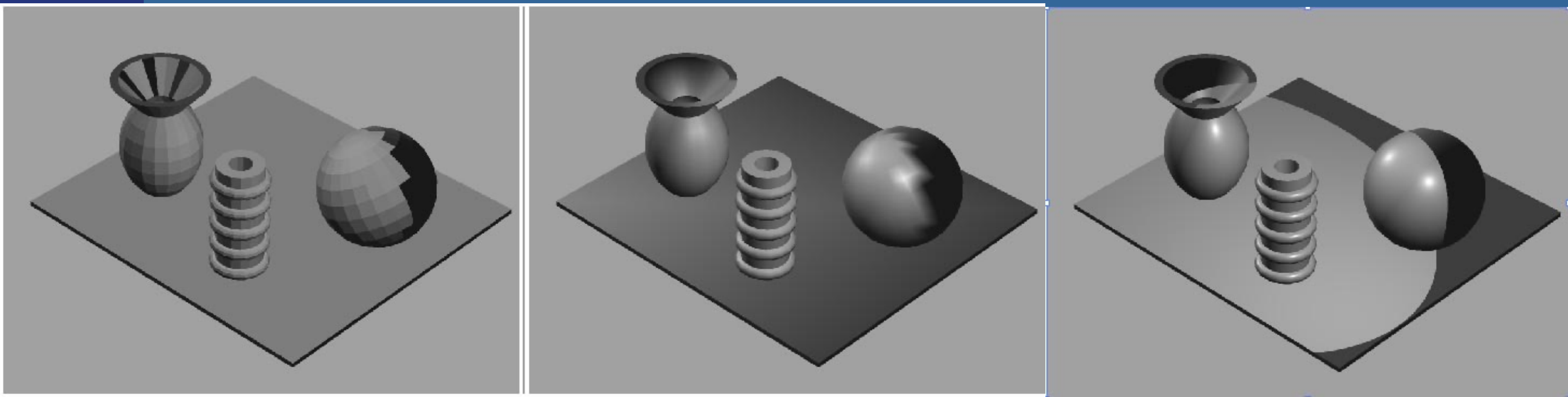
- Energy is emitted at equal proportions in all directions from a spherical radiator. Due to energy conservation, the intensity is proportional to the **spherical area** at distance r from the light center.
- $A = 4\pi r^2$
- Thus, the intensity scales $\sim 1/r^2$
- For efficiency, we often cap or limit how far the light source will affect the environment.
  - Hence, we often want to fade its intensity to zero at some finite distance r.



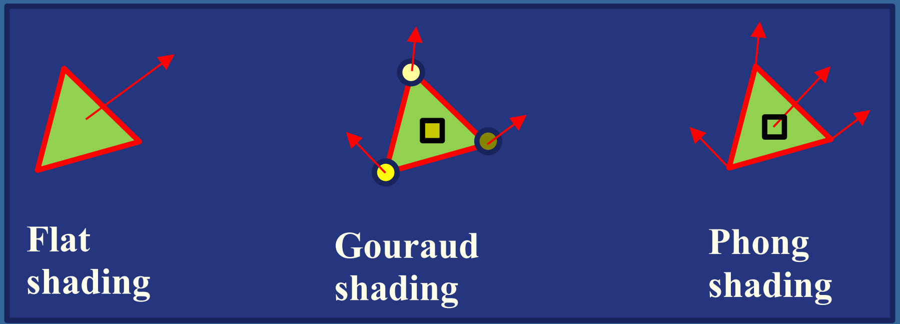
Shading

- Shading: compute the fragment's final color contribution to the pixel.
- Three types of shading

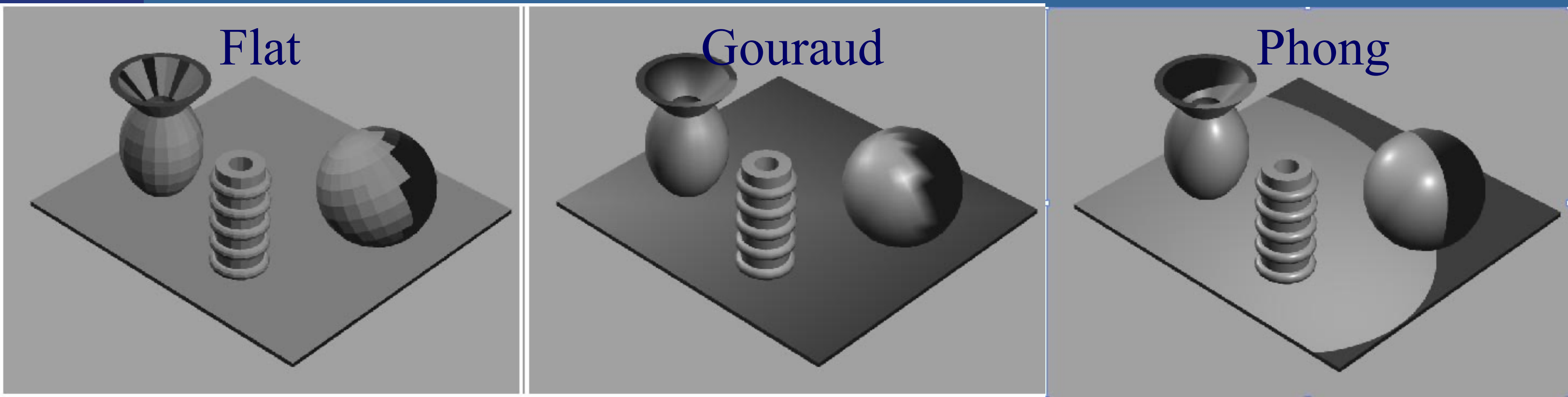
  regarding how often it is computed per triangle:
  - Flat shading: once per triangle
  - Goraud shading: once per vertex
  - Phong shading: once per pixel (standard today)



Shading



- Flat, Goraud, and Phong shading:
  - Flat shading: one normal per triangle. Lighting computed once for the whole triangle.
  - Gouraud shading: the lighting is computed per triangle vertex and for each pixel, the **color is interpolated** from the colors at the vertices.
  - Phong Shading: the lighting is **not** computed per vertex. Instead the **normal is interpolated** per pixel from the normals defined at the vertices and full lighting is computed per pixel using this normal. This is of course more expensive but looks better.



Transparency and alpha

- Transparency
  - Very simple in real-time contexts
- The tool: alpha blending (mix two colors)
- Alpha (α) is the forth color component (r,g,b,α)
  - e.g., of the material for a triangle
  - Represents the opacity
  - 1.0 is totally opaque
  - 0.0 is totally transparent
- The over operator:

Color already in the frame buffer at the corresponding position

$$\mathbf{c}_o = \alpha \mathbf{c}_s + (1-\alpha)\mathbf{c}_d$$

Rendered object

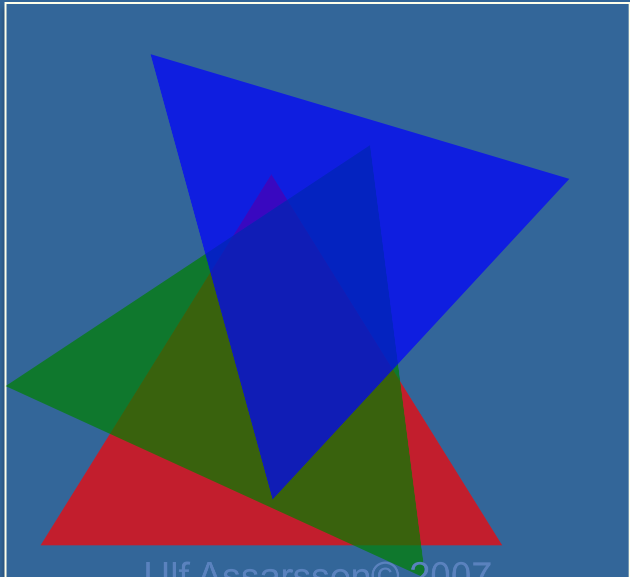
$$\mathbf{c}_o = \alpha\mathbf{c}_s + (1-\alpha)\mathbf{c}_d$$

Rendered fragment ($\mathbf{c}_s$) — Background ($\mathbf{c}_d$)

Transparency

- Need to sort the transparent objects
  - Render back to front (blending is order dep.)
    - See next slide…
- Lots of different other blending modes
- Can store RGBα in textures as well

So the texels with α=0.0 do not not hide the objects behind



Transparency

- Need to sort the transparent objects
  - **First, render all non-transparent triangles as usual.**
  - **Then, sort all transparent triangles and render them back-to-front with blending enabled.**
    - **The reason for sorting is that the blending operation (i.e., over operator) is order dependent.**

If we have high frame-to-frame coherency regarding the objects to be sorted per frame, then Bubble-sort (or Insertion sort) are really good! (superior to Quicksort).
Because, they have expected runtime of resorting already almost sorted input in O(n) instead of O(n log n), where n is number of elements.

Blending

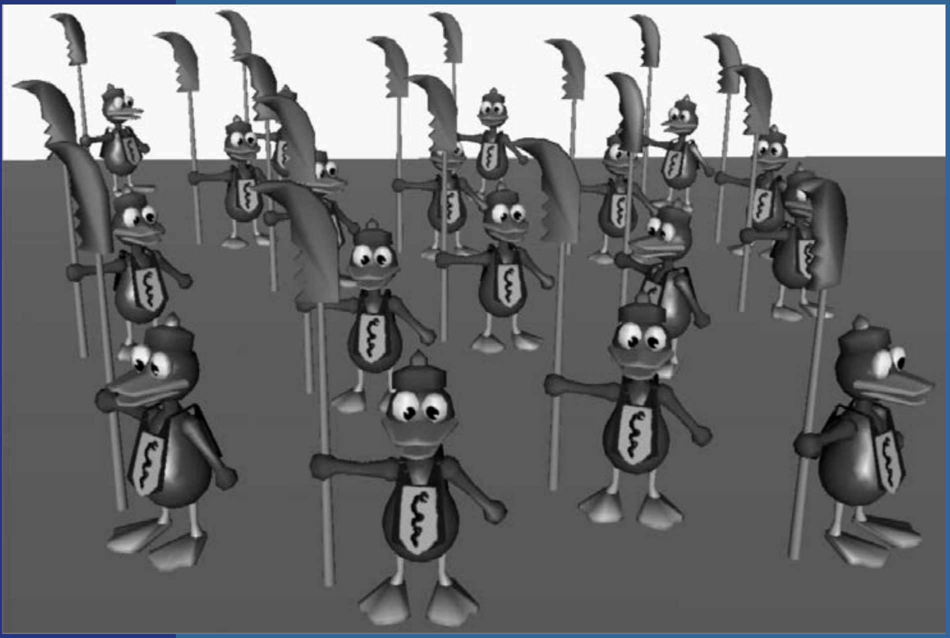
- Used for
  - Transparency

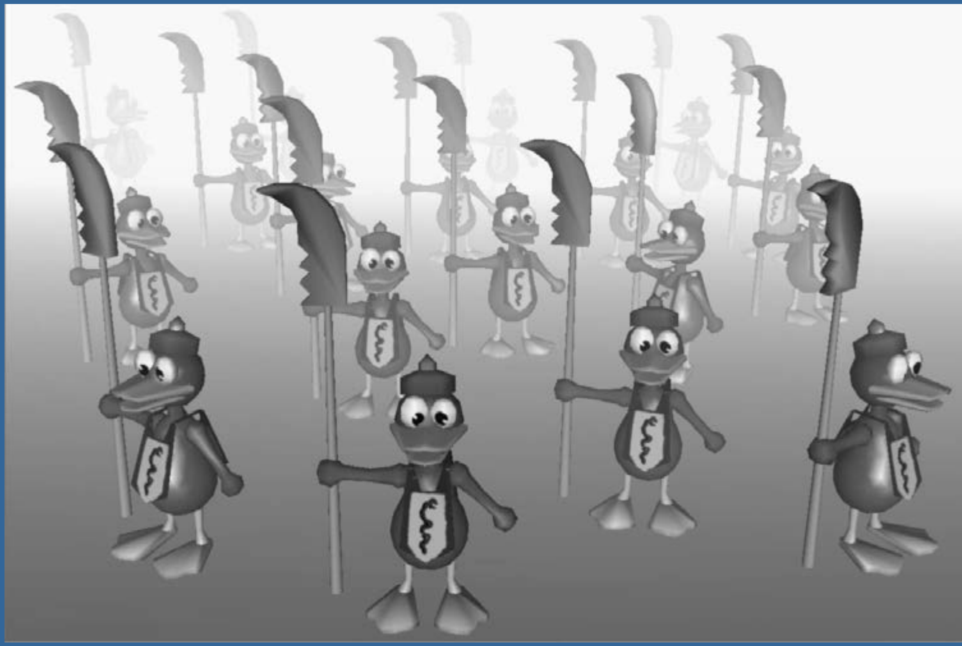
    $$\mathbf{c}_o = \alpha\mathbf{c}_s + (1-\alpha)\mathbf{c}_d$$

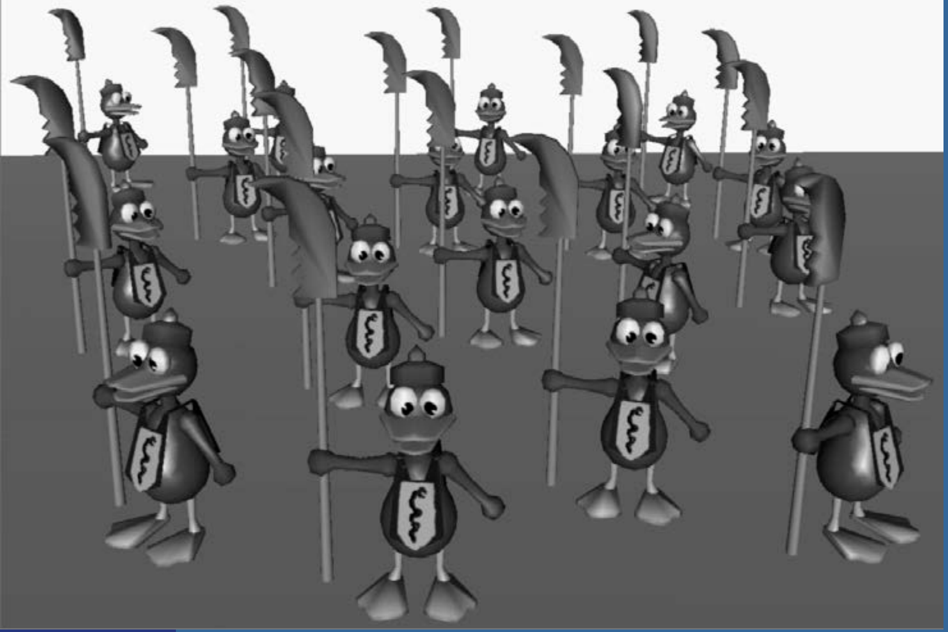
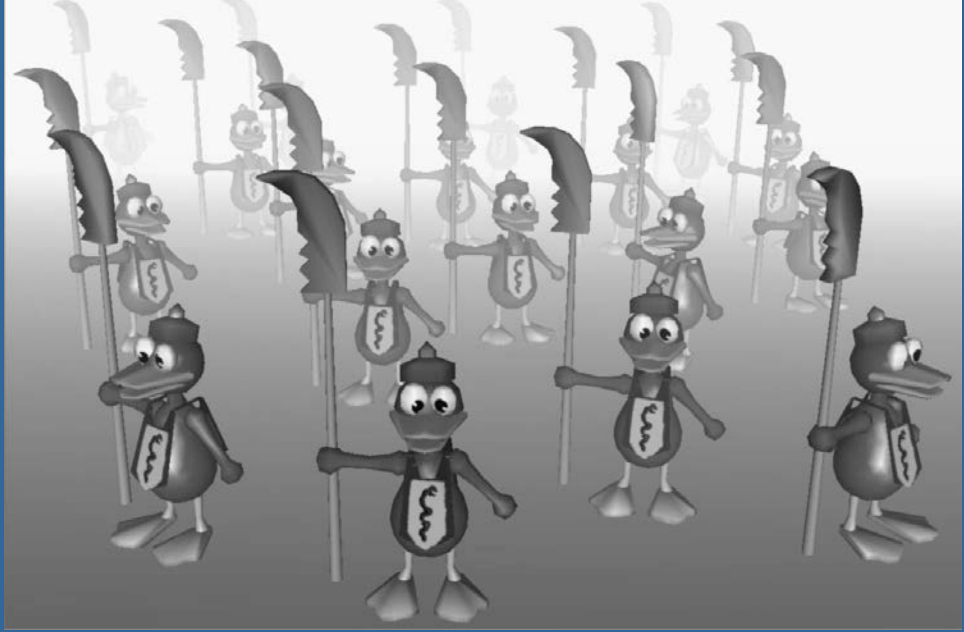
    - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
  - Effects (shadows, reflections)
  - (Complex materials)
    - Quake3 used up to 10 rendering passes, blending toghether contributions such as:
      - Diffuse lighting (for hard shadows)
      - Bump maps
      - Base texture
      - Specular and emissive lighting
      - Volumetric/atmospheric effects
  - Enable with **glEnable(GL_BLEND)**

Fog

- Simple atmospheric effect
  - A little better realism
  - Help in determining distances





- Color of fog: $\mathbf{c}_f$ color of surface: $\mathbf{c}_s$

$$\mathbf{c}_p = f\mathbf{c}_s + (1-f)\mathbf{c}_f \qquad f \in [0,1]$$

- How to compute f?
- E.g.: linear, exponential
- Linear:

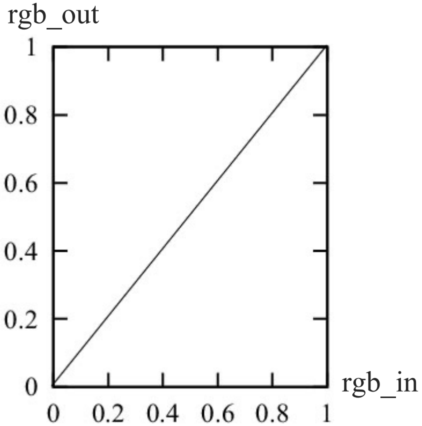
$$f = \frac{z_{end} - z_p}{z_{end} - z_{start}}$$

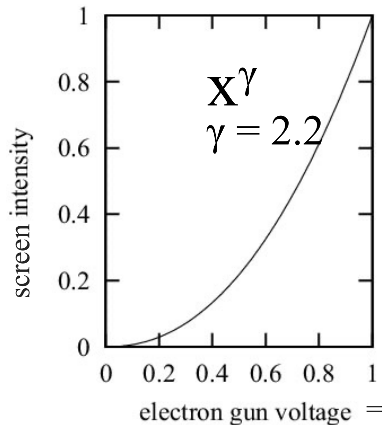
Program it yourself in the fragment shader.

(Old OpenGL – just set OpenGL parameters and turn it on)

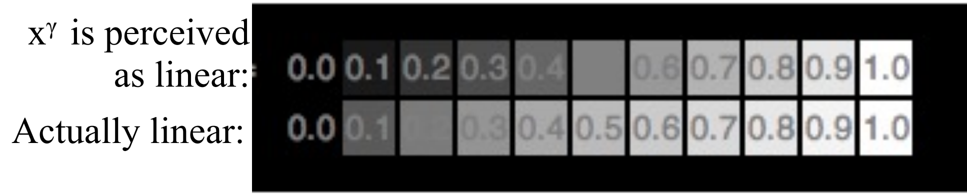
Fog in up-direction

Gamma correction





Expon. distribution better for humans. Our eyes have non-linear sensitivity and monitors have limited brightness



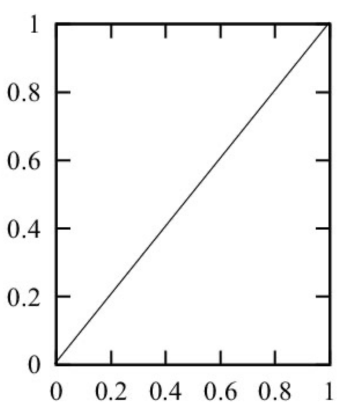
Intensities: x^γ vs linear

We compute rgb color intensities in linear space from [0,1]

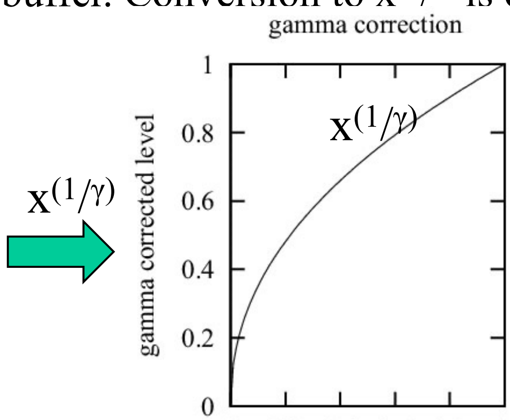
However, CRT-monitor output is exponential. Gives more precision for darker regions. Very Good! But we want linear output. Else, our images will be too dark.

Textures: store in gamma space for better ditributed precision.

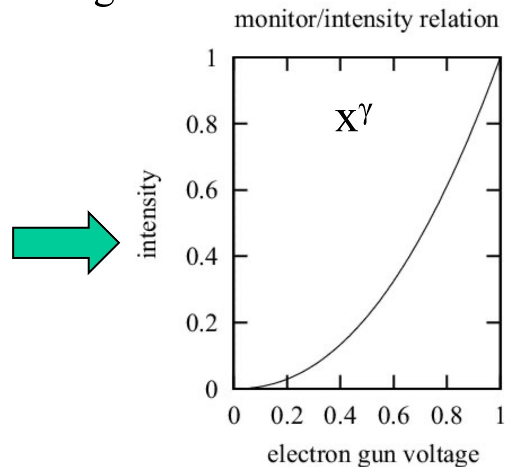
So, store color intensities with more precision for darker colors: i.e., convert color to $x^{(1/\gamma)}$ before storing in 8- bits in the frame buffer. Conversion to $x^{(1/\gamma)}$ is called gamma correction.



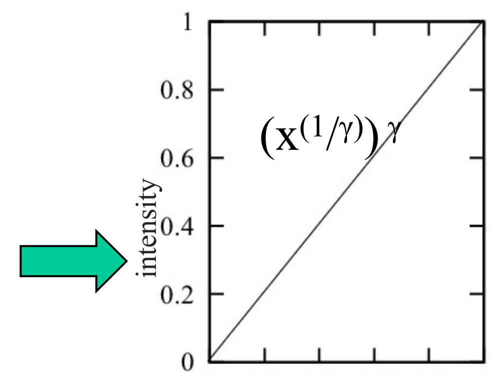
Shader rgb colors



Frame buffer rgb colors.
“Dark pixels are made brighter”

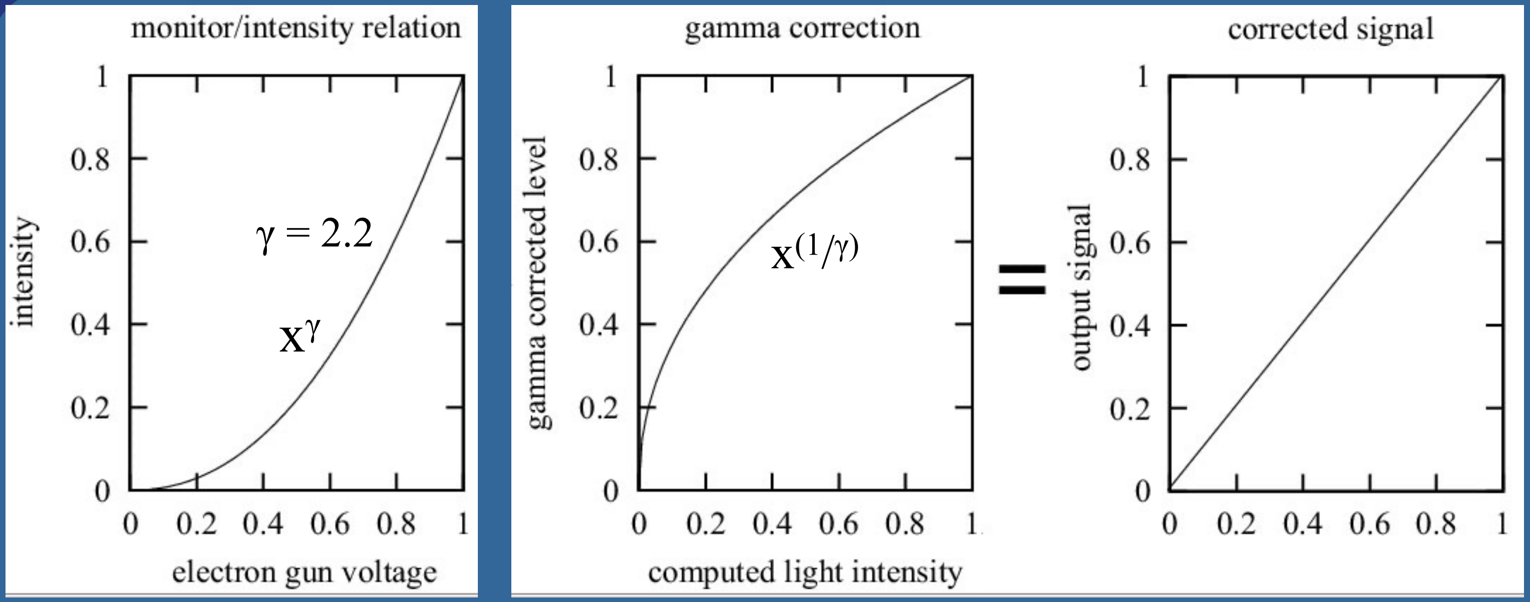


Displayed by CRT



Linear output again, but redistributed precision.

Gamma correction



- If input to gun is 0.5, then you don't get 0.5 as output in intensity
- Instead, gamma correct that signal: gives linear relationship

Gamma correction

$$I = a(V + \varepsilon)^{\gamma}$$

- *I*=intensity on screen
- *V*=input voltage (electron gun)
- *a*, ε, *and* γ *are* constants for each system
- Common gamma values: 2.2-2.6
- Assuming ε=0, gamma correction is:

$$c = c_i^{(1/\gamma)}$$

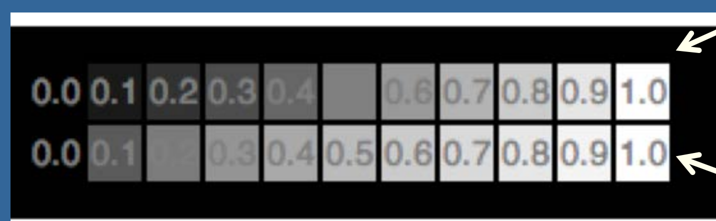
Why is it important to care about gamma correction?

- Portability across platforms
- Image quality
  - Texturing
  - Anti-aliasing
- One solution is to put gamma correction in hardware…
- sRGB asumes gamma=2.2
- Can use EXT_framebuffer_sRGB to render with gamma correction directly to frame buffer

Gamma correction today

- Reasons for wanting gamma correction (standard is 2.2):

1. Screen has non-linear color intensity
   - We often really want linear output (e.g. for correct antialiasing)
   - (But, today, screens can be made with linear output, so non-linearity is more for backwards compatibility reasons.)
2. Also happens to give more efficient color space (when compressing intensity from 32-bit floats to 8-bits). Thus, often desired when storing textures.



Gamma of 2.2. Better distribution for humans. Perceived as linear.

Truly linear intensity increase.

A linear intensity output (bottom) has a large jump in perceived brightness between the intensity values 0.0 and 0.1, while the steps at the higher end of the scale are hardly perceptible.
A nonlinearly-increasing intensity (upper), will show much more even steps in perceived brightness.

Important on Gamma correction

- Give two reasons for gamma correction:
  - screen output is non-linear so we need gamma to counter that.
  - Textures/images can be stored with better precision (for human eye) for low-intensity regions.

What is important

- Amb-, diff-, spec-, emission model + formulas
- Phong's + Blinn's highlight model:
  - Phong: scales with $(\boldsymbol{r} \cdot \boldsymbol{v})^s$
  - Blinn: scales with $(\boldsymbol{n} \cdot \boldsymbol{h})^s$, *halfvector* **h** = (**l**+**v**)/|**l** + **v**|
- Flat-, Gouraud- and Phong shading
- Transparency:
  - Draw transparent triangles back-to-front.
  - Use blending with this over operator: $\mathbf{c}_o = \alpha \mathbf{c}_s + (1-\alpha)\mathbf{c}_d$
- Two reasons for wanting gamma correction