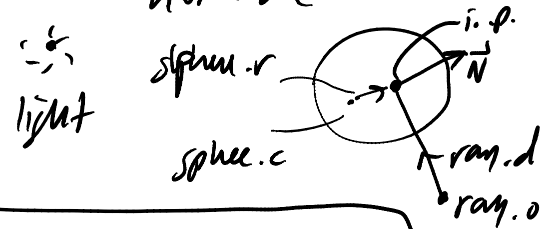
rendering equation (and light_t object)

Wednesday, October 27, 2010
8:52 AM

Today: computing light's reflectivity at surface point

Last time: sphere intersection & normal at intersection



light name
{

position x y z — vec_t

emissivity r g b — drgb_t (or rgb_t<double> later)

}

- emissivity is the light color, e.g. 1 1 1 is white

Wednesday, October 27, 2010
9:06 AM

Rendering equation: (at surface point, calculate intensity I)

$$I = \underbrace{\frac{K_a I_a}{R}}_{\text{ambient light term}} + \sum_{l} \frac{I_L}{r} \Big(\underbrace{K_d (N \cdot L)}_{\text{diffuse term}} + \underbrace{K_s (R \cdot V)^n}_{\text{specular term}} \Big)$$

I_L: light color (intensity, emissivity)

r: distance to light

ambient light term,

diffuse term, K_d = obj→getdiffuse

specular term, K_s = obj→getspecular

R is the ray distance (what you already have)

in code: color = $\underbrace{\text{obj→getambient()}}_{K_a}$ / raydist;

K_a (surface property of object, basically the surface color)

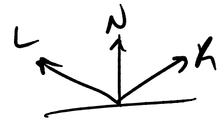
here $I_a \approx (1\ 1\ 1)$ white light (a bit of a hack)

where

N: surface normal

L: direction to light (from surface point) (normalized)

R: L dir reflected about N



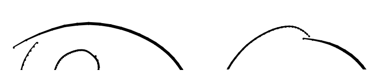
$R = 2(N \cdot L)N - L$

V: direction to camera (normalized)

n: shininess term just a double,

e.g. $n = 1.0$, $n = 32.0$ — the larger n is, the smaller the spec. highlight

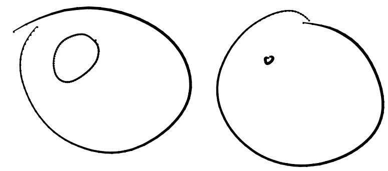
— this is called the Phong illumination model — an estimate of physical light transport — basically what OpenGL uses for direct illum. (local)



[illegible]

(local)

ray tracer, by calculating ray reflections, produces global illumination that GL doesn't do



n=2 n=32

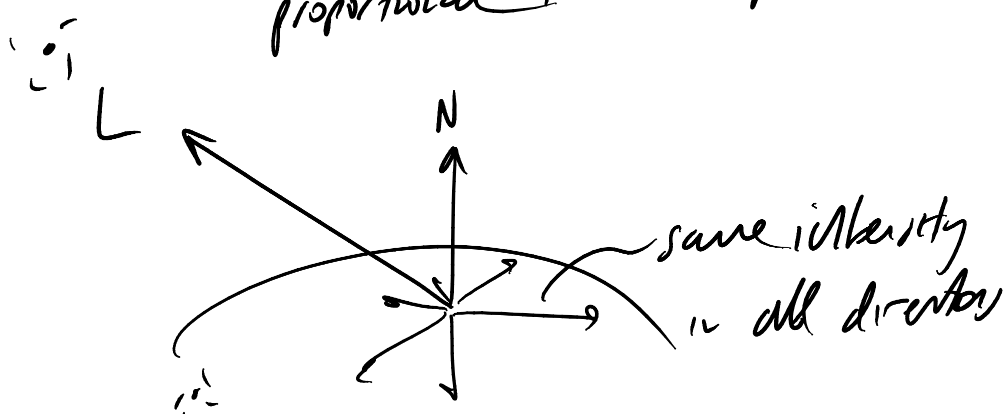
Wednesday, October 27, 2010
9:22 AM

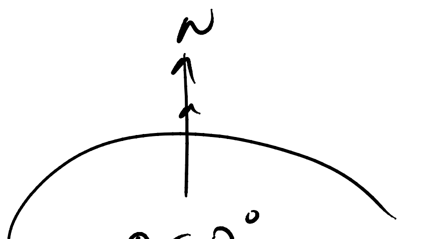
- Model derivation:
- start with diffuse scattering: light arriving at surface scattered equally in all directions (matte surface)
- perfect diffuse reflector: reflects light according to Lambert's Law:

$$I \propto L \cdot N$$

$\propto$: proportional to

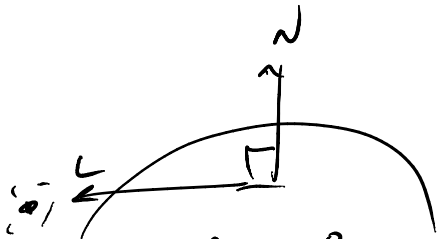
$L \cdot N$: cosine of the angle between light dir & N





$\theta = 0°$

$\cos(\theta) = 1 = N \cdot L$



$\theta = 90°$

$\cos(\theta) = 0 = N \cdot L$

(fully illuminated) (no reflection)

if ($0 \leq N \cdot L \leq 1$)
surface visible by light

else
surface hidden
(only gets ambient contrib.)

- Lambertian lighting model (model 1)

$$I = I_L \, k_d \, (N \cdot L)$$

obj → diffuse

with ambient term looks like this:

$$I = I_a k_a + \sum_{l} I_L k_d (N \cdot L)$$

Next refinement: scale by distance to light

Next refinement: scale by distance to light

in code: L = light → getPosition() − last_hit

r = L.len(); // get dist to light

L = L.norm(); // normalize L

attenuated light model (model 2)

$$I = I_a k_a + \sum_{l} \frac{I_l}{r} (k_d (N \cdot L))$$

OpenGL uses something weirder here

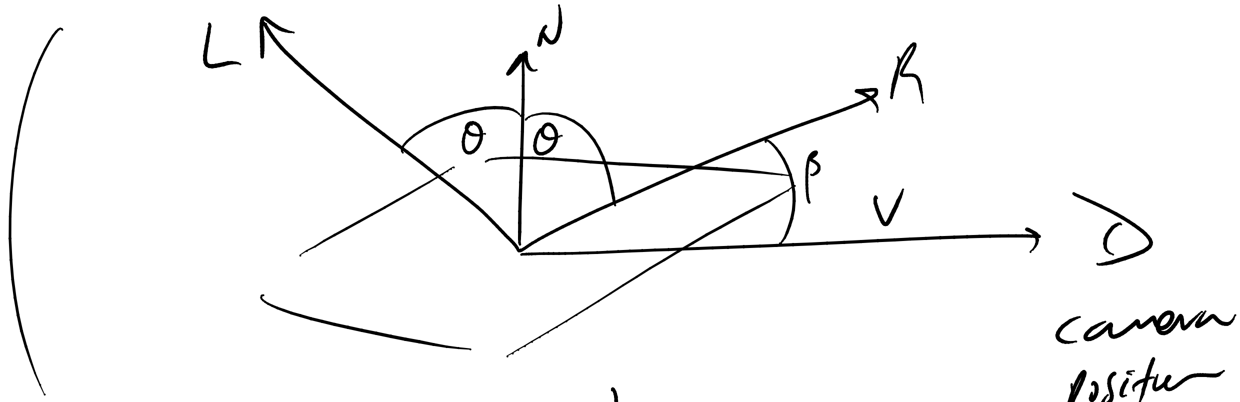
$(r + k_1 + k_2)$ (?)

- still diffuse (matte surface)

Wednesday, October 27, 2010
9:38 AM

(specular)

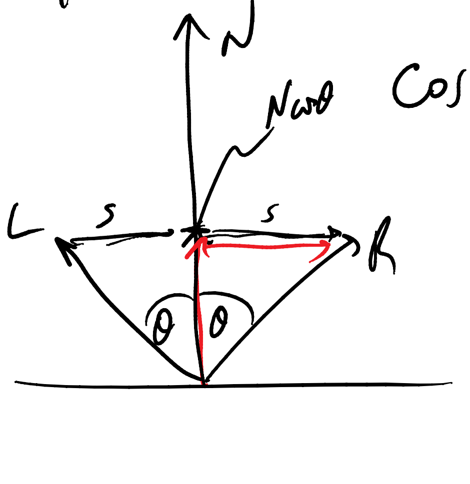
- adding in shiny component:
- models the "perfect reflector" (mirror)



should only see perfect reflection if $\beta = 0$, looking down at R

- for non-perfect reflectors, intensity of reflected light falls off sharply w/ β increases
- falloff is approximated by

$$\cos^n \beta = (R \cdot V)^n$$



$R = N\cos\theta + s$

but $s = N\cos\theta - L$ via similar Δ

so $R = N\cos\theta + N\cos\theta - L$

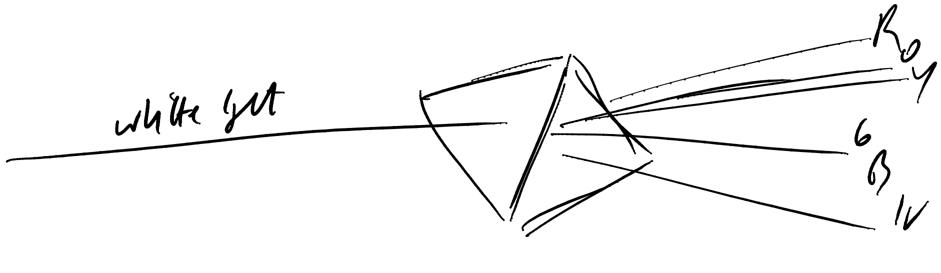
$= 2N\cos\theta - L$

$= 2N(N\cdot L) - L$

- for real materials, amount of incident light specularly reflected depends on both angle of incidence θ and wavelength

$$I_s = I_L\, w(\lambda, \theta) \cos^n \beta$$

this requires a ray per wavelength for all wavelengths you'd like to model



- another circle:

Phong

$$I = I_a k_a + \sum_l \frac{I_L}{r}\left(k_d (N\cdot L) + k_s (R\cdot V)^n\right)$$

Blinn

Blinn

$$I = I_a k_a + \sum_{l} \frac{I_l}{r}\left(k_d (N \cdot L) + k_s (N \cdot H)^n \right)$$

bisector of L & V

$$H = \frac{L+V}{\|L+V\|} \text{ or } \frac{1}{2}(L+V)$$