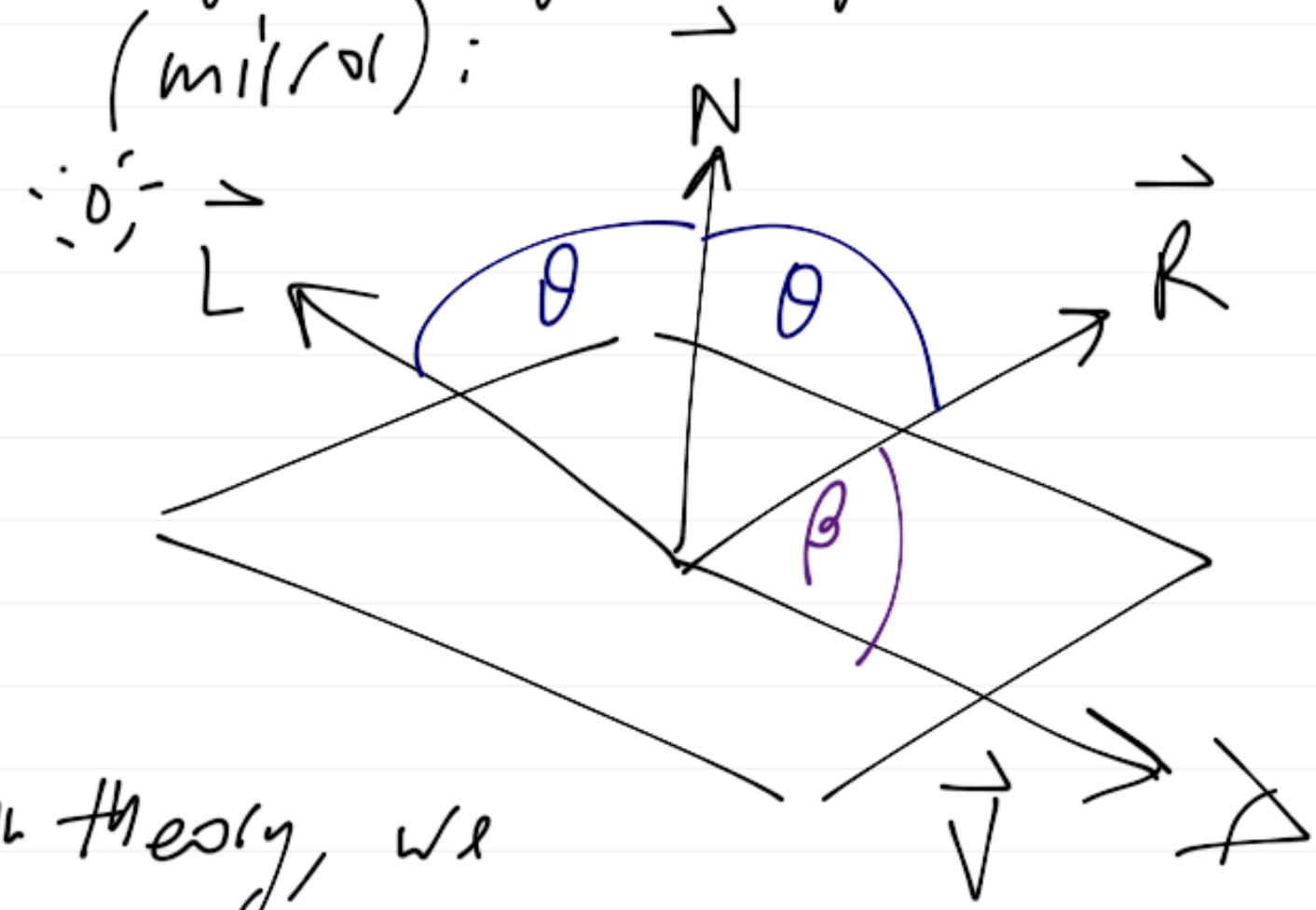
- last time: ambient & diffuse components
- today: specular component, specular highlights, & reflection (recursion — ray_trace calls itself)

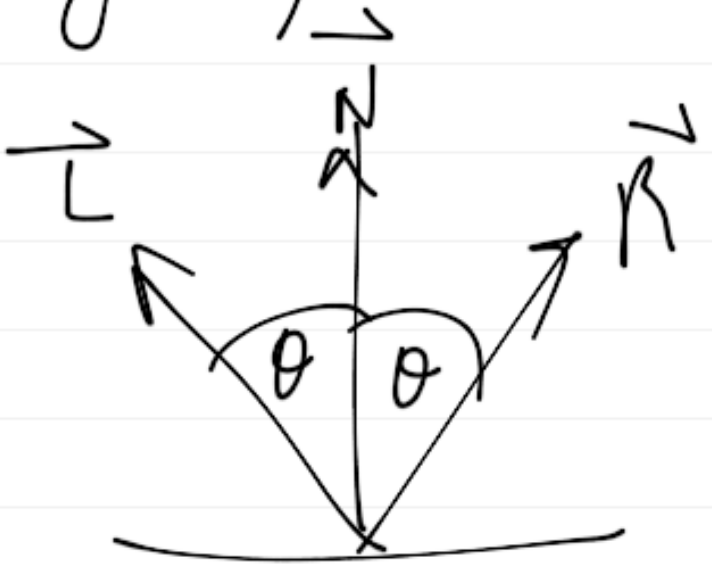
- specular (shiny) component:
- model of a "perfect reflector" (mirror):

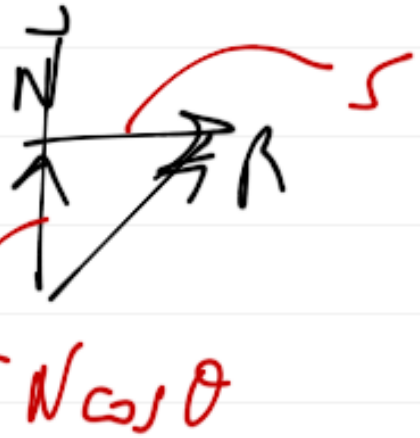


- in theory, we should only see perfect reflection if $\beta = 0$, looking down $\vec{R}$ direction

- to "fudge" this a bit, modeling non-perfect reflectors, intensity of reflected light falls off sharply as β increases

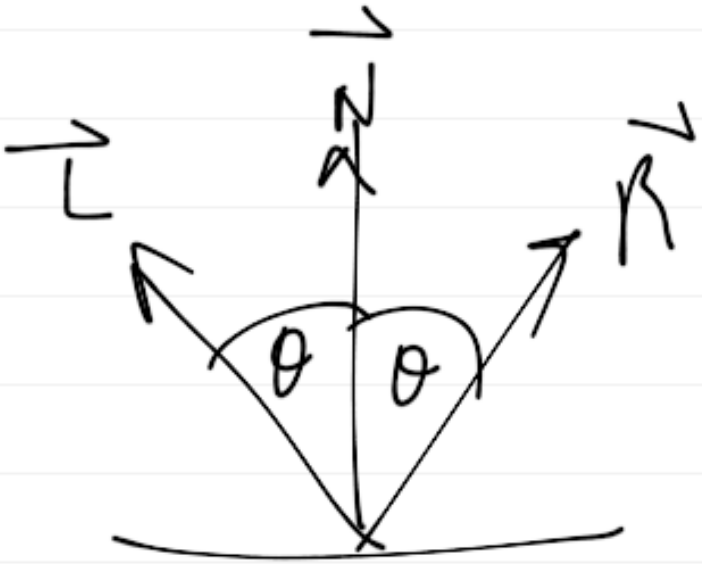
- first, need to calculate $\vec{R}$





$R = N\cos\theta + S$

$\angle$ incidence = $\angle$ reflection



$R = N\cos\theta + S$

$\angle$ incidence = $\angle$ reflection

$S = N\cos\theta - L$

(using similar $\triangle$)

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

$= 2N\cos\theta - L$

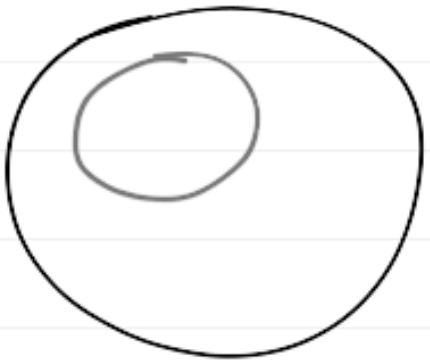
$= 2N(N \cdot L) - L$

} dir. of reflection

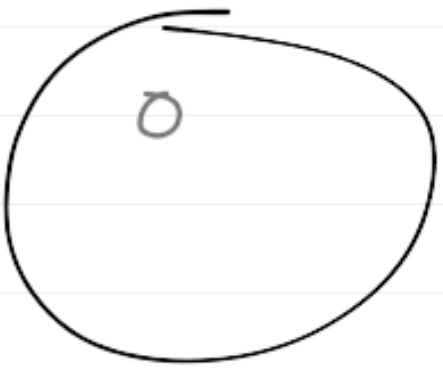
- remember angle β and sharp falloff : approximate this via $\cos^n \beta = (R \cdot V)^n$

- We now know what $\vec{R}$ and $\vec{V}$ are

- n is the "shininess" factor



$n = 2$



$n = 32$

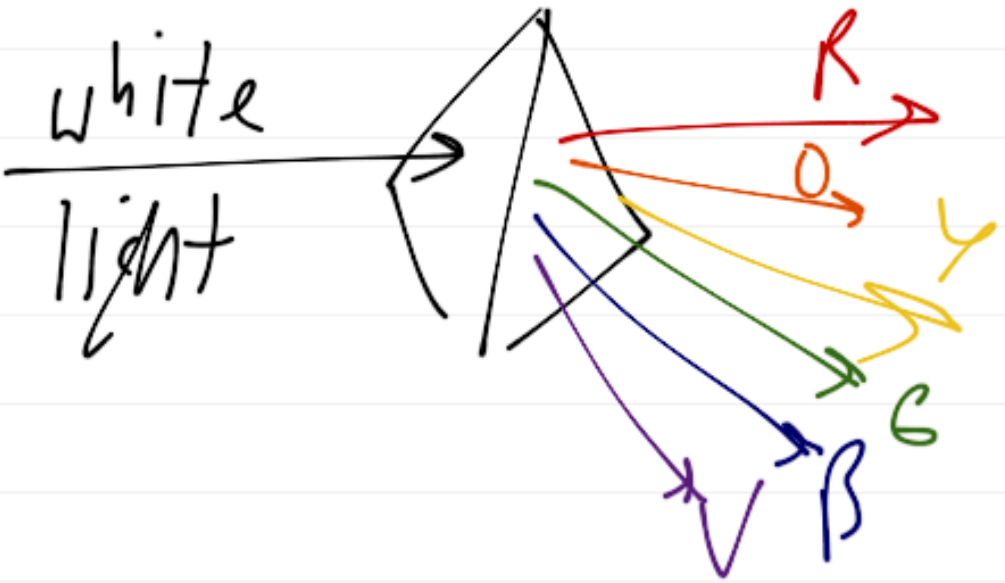
- for real materials, amount of incident light specularly reflected depends on both angle of incidence θ and wavelength:

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

↑

this would require a ray per wavelength for as many wavelengths you want to model

- when doing refraction, this leads to prismatic effect



- our model (phong) is a bit simpler:

$$I = \frac{K_a I_a}{R} + \sum_l \frac{I_l}{r}\left(K_d(N \cdot L) + K_s(R \cdot V)^n\right)$$

- Phong:

$$I = \frac{K_a I_a}{R} + \sum_l \frac{I_l}{r}\left(K_d (N \cdot L) + K_s (R \cdot V)^n\right)$$

- an alternative, Blinn:

$$I = \frac{K_a I_a}{R} + \sum_l \frac{I_l}{r}\left(K_d (N \cdot L) + K_s (N \cdot H)^n\right)$$

where H is the bisector of $\vec{L}$ and $\vec{V}$:

$$\vec{H} = \frac{1}{2}(\vec{L} + \vec{V})$$

- back to code:

in ray_trace:

```
if (!(obj = [model find_closest:
        pos: &div: &dis: &hit: &N))
            || rayDist > MAX_DIST)
    return NULL;
if (dis > 0) {
    rayDist += div;
    ambient = [[obj; mat] getAmb];
    V = ...
    if ([diffVal nonzero]) { ... }
    if ([specular nonzero]) { ... }
}
```

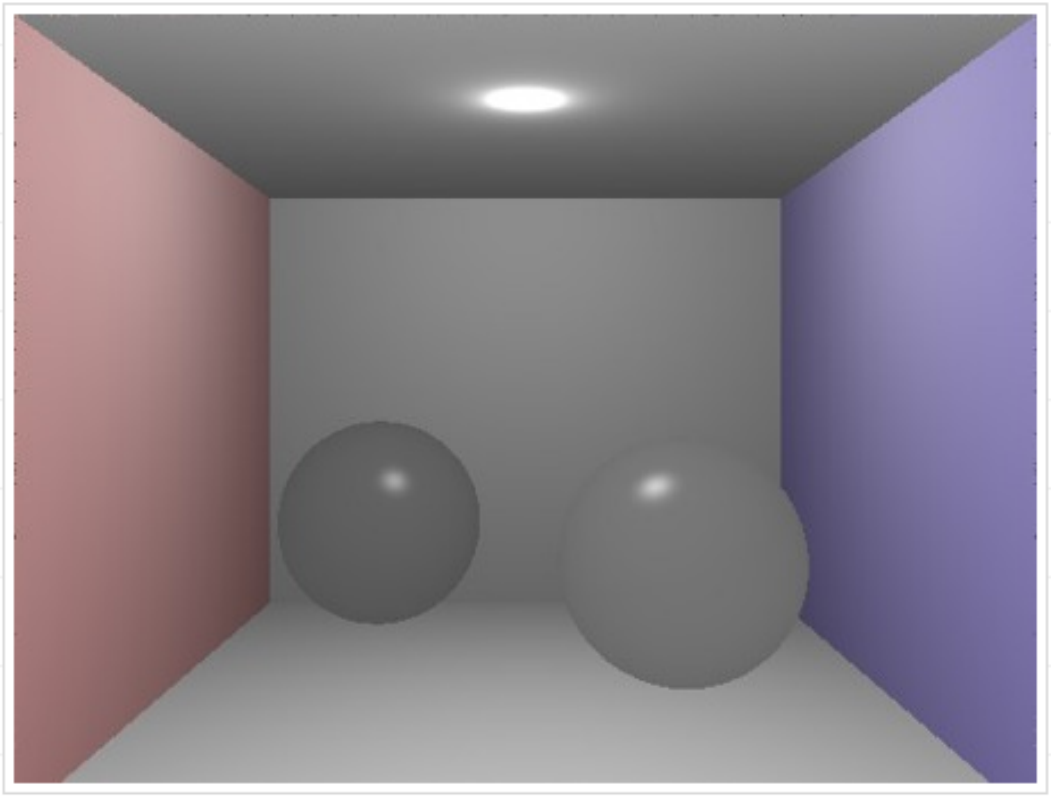
last time

hit time

```
if ([specular nonzero]) {
    // compute R = 2N(N·L) - L
    // reflect ray - I'll go over this later
    // add specular highlights
    for([[model lgts] reset];
        ![[model lgts] end];
            [[model lgts] next]) {
        lgt = [[model lgts] data];
```

```
[L autorelease];
L = [[[lgt getLightVec:
        hit] retain];
ndotL = [N dot L];
if (0.0 < ndotL && ndotL < 1.0) {
  [R autorelease];
  R = 2N(N·L) - L
  [I_s autorelease];
  I_s = [[[lgt getColor] scale:
    1.0/r * pow([R dot:V], n)]
          retain];
  [color autorelease];
  color = [[color add:[I_s mul:
    specular]] retain];
}
```

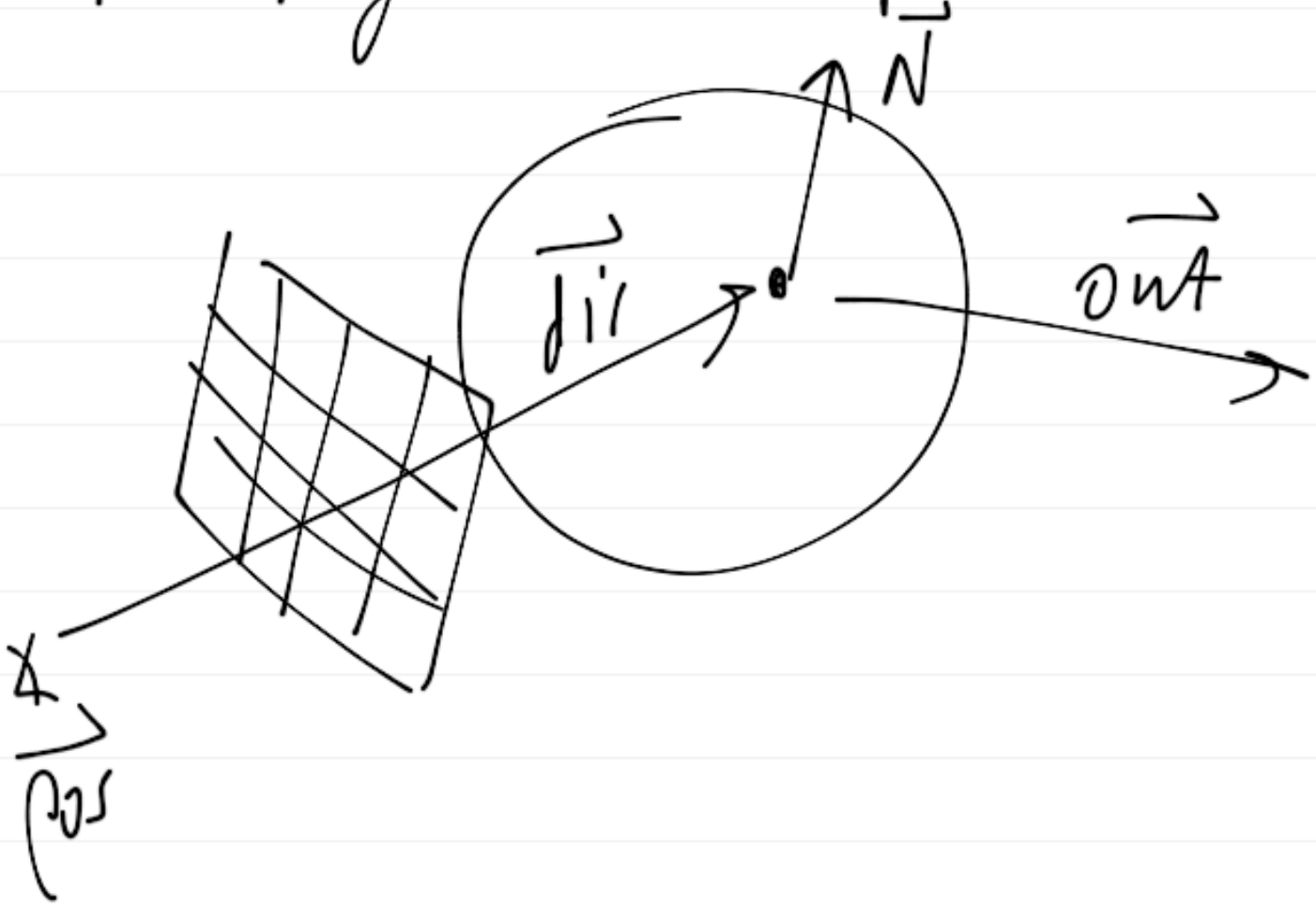
- that bit of code should produce specular highlights:



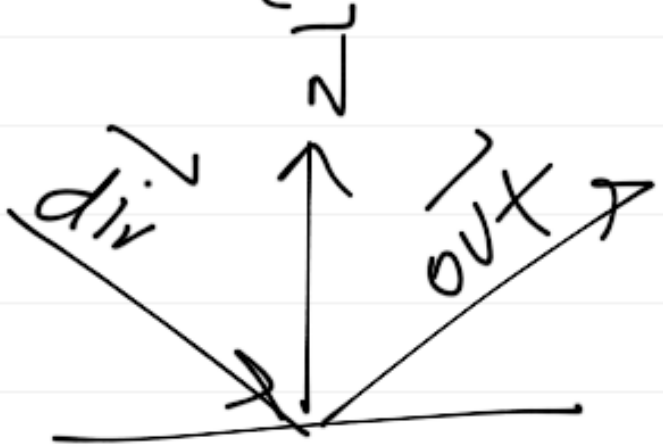
- all that remains is to reflect the ray <u>prior</u> to addition of the highlights

// reflect ray — I'll go over this later

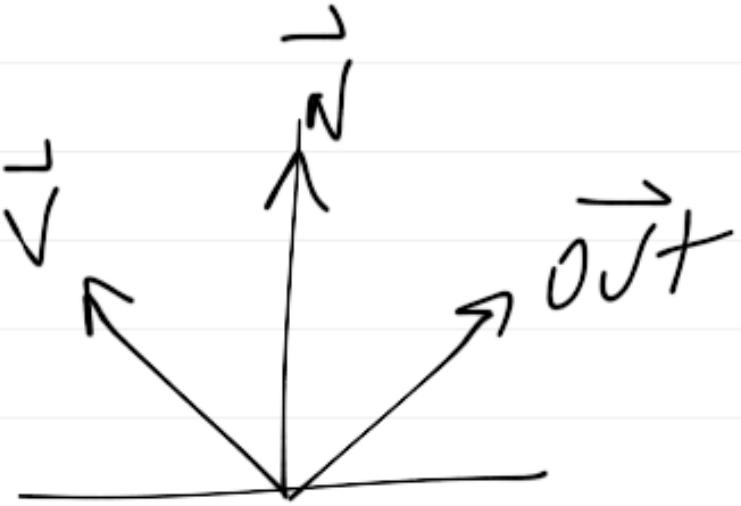
- here we want to reflect the ray at hit point



- Geometry is the same as mirror, but need to take care to negate direction (view) vector



$$\vec{V} = -\vec{dir}$$



$\overrightarrow{out} = 2N(N \cdot V) - V$

- rest of code is:

```
pixel_t *refcolor =
[(pixel_t *)[pixel_t alloc]
        init: 0.0: 0.0: 0.0];
[refcolor autorelease];
refcolor = [ray_trace (model, hit,
        out, ray dist) retain];
```

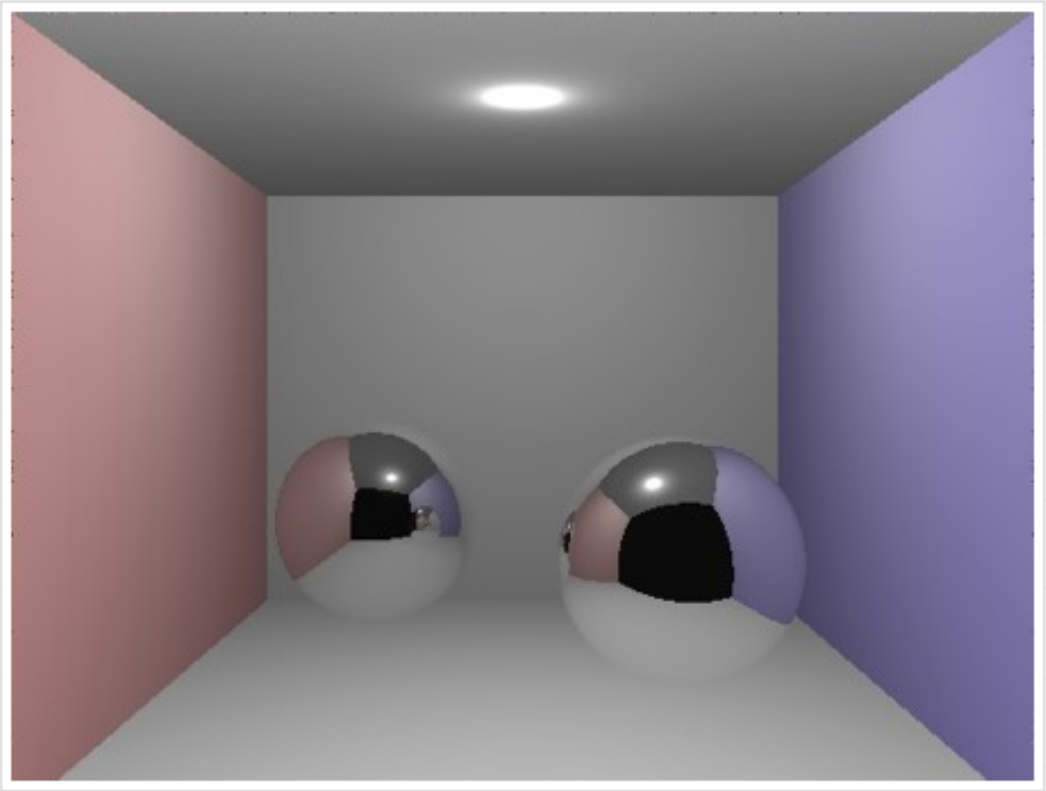
- then blend in color:

```
[color autorelease];
color = [[[color mul: diffuse]
    add: [reflect mul: specular]]
        retain];
```

- now add spec. highlights
- at the end of ray_trace

```
return [color autorelease];
```

- result is complete ray tracer:



- details:
  - arguments to ray_trace
  - local variables

- arguments

model_t *model,

vec_t *pos,

vec_t *dir,

double raydist

} ray

All should really be its own object

- local variables:

```
t_pixel *color = alloc, fix it
    (what gets returned)
double dis = 0.0
    (distance to current obj)
t_vec *hit
    (hit point, i.e. surface point)
t_vec *N
    (normal at hit point)
```

```
object_t *obj = NULL;
```
(object hit)
```
pixel_t *ambient,
        *diffuse,
        *specular;
```
(surface properties)

```
light_t *lgt;
```

(pointer to light)

```
vec_t *L,
      *V,
      *R,
      *H;
```

} alloc and init each

```
pixel_t *I_d,
        *I_s
```

} alloc & init

```
double r, ndotl = 0.0;
double n = 32.0; //shininess
```