CS4620/5620: Lecture 35

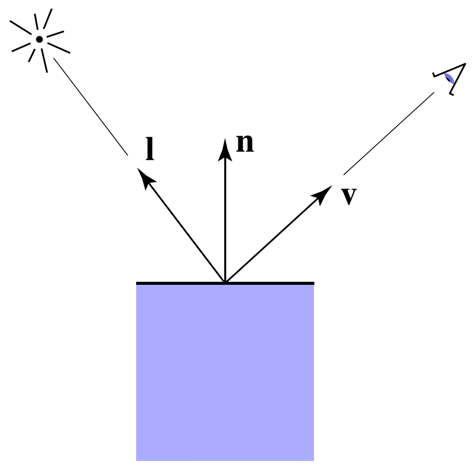
Ray Tracing (Shading)

Announcements

- 4621
  - Class today
- Turn in HW3
- PPA3 is going to be out today
- PA3A is out

Shading

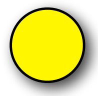
- Compute light reflected toward camera
- Inputs:
  - eye direction
  - light direction (for each of many lights)
  - surface normal
  - surface parameters (color, shininess, …)

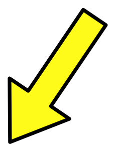


Light

- Local light
  - Position

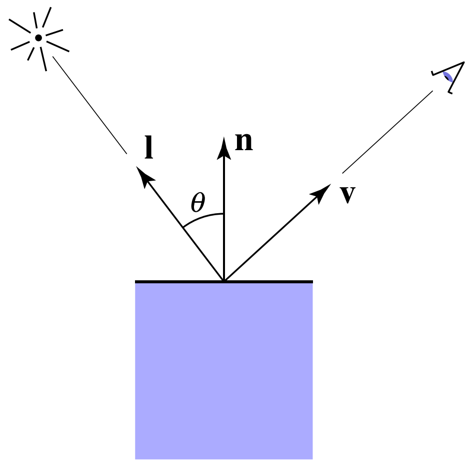
- Directional light (e.g., sun)
  - Direction, no position





Lambertian shading

- Shading independent of view direction



$$L_d = k_d \, I \max(0, \mathbf{n} \cdot \mathbf{l})$$

illumination from source

diffuse coefficient

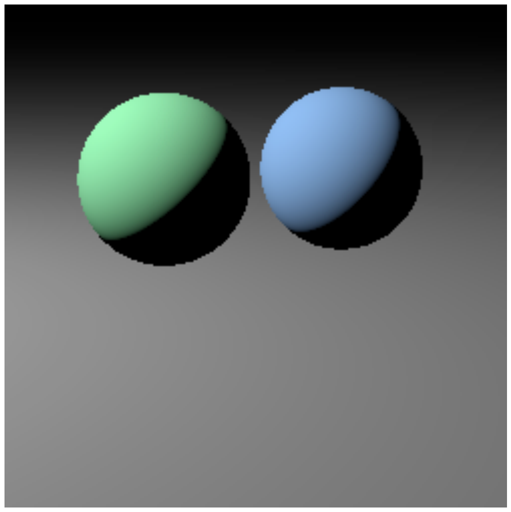
diffusely reflected light

Image so far

```
Scene.trace(Ray ray, tMin, tMax) {
    surface, t = hit(ray, tMin, tMax);
    if surface is not null {
        point = ray.evaluate(t);
        normal = surface.getNormal(point);
        return surface.shade(ray, point,
            normal, light);
    }
    else return backgroundColor;
}

...

Surface.shade(ray, point, normal, light) {
    v = –normalize(ray.direction);
    l = normalize(light.pos – point);
    // compute shading
}
```

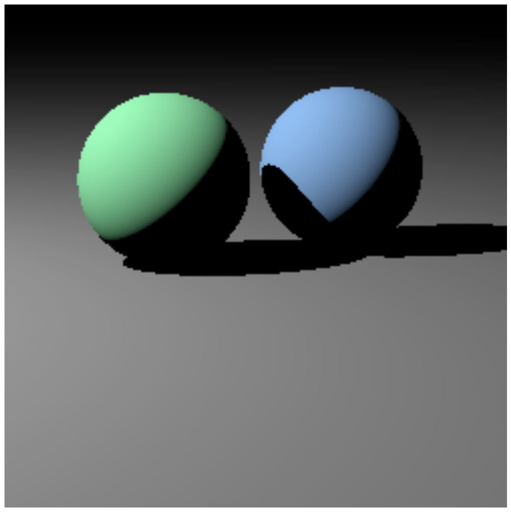


Shadows

- Surface is only illuminated if nothing blocks its view of the light.
- With ray tracing it's easy to check
  - just intersect a ray with the scene!

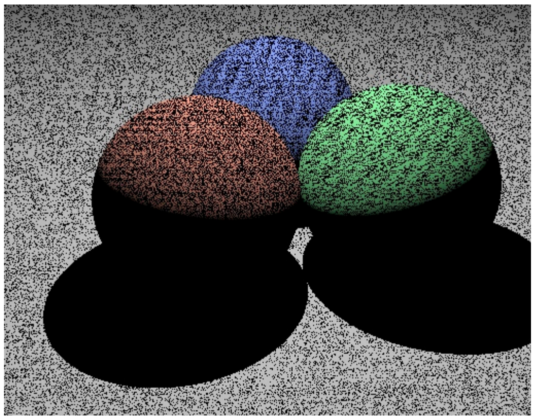
Image so far

```
Surface.shade(ray, point, normal, light) {
    shadRay = (point, light.pos – point);
    if (shadRay not blocked) {
        v = –normalize(ray.direction);
        l = normalize(light.pos – point);
        // compute shading
    }
    return black;
}
```



Shadow rounding errors

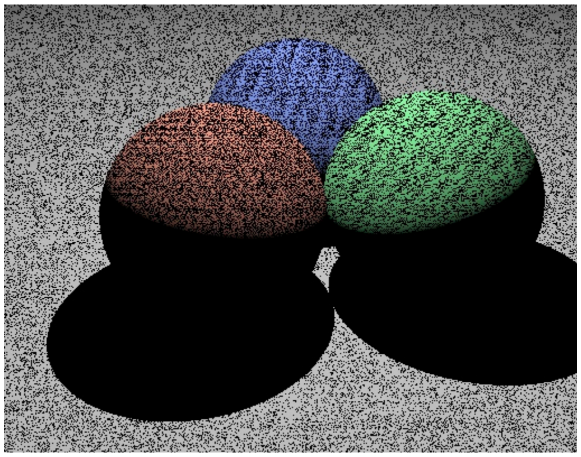
- Sounds like it should work, but hmm....



- What’s going on?

Shadow rounding errors

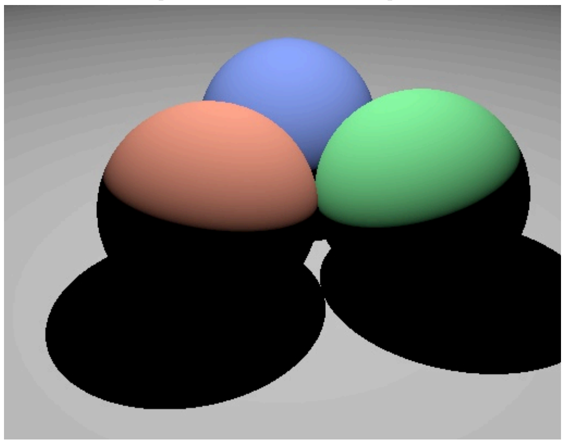
- Don’t fall victim to one of the classic blunders:



- What’s going on?
  - hint: at what t does the shadow ray intersect the surface you’re shading?

Shadow rounding errors

- Solution: shadow rays start a tiny distance from the surface



- Do this by moving the start point, or by limiting the *t* range

Multiple lights

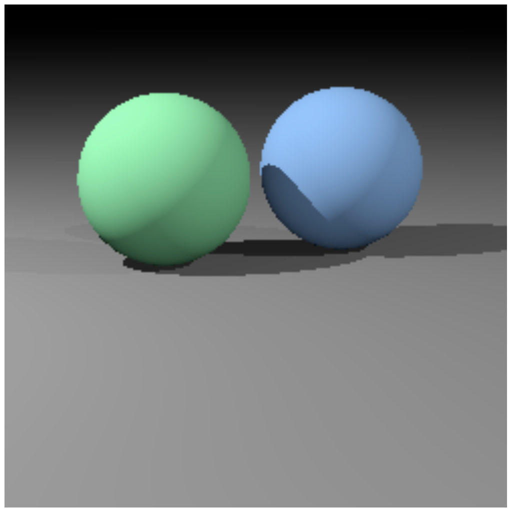
- Just loop over lights, add contributions
- Important to fill in black shadows

Multiple lights

- Important to fill in black shadows
- Just loop over lights, add contributions
- Ambient shading
  - black shadows are not really right
  - one solution: dim light at camera
  - alternative: add a constant "ambient" color to the shading…

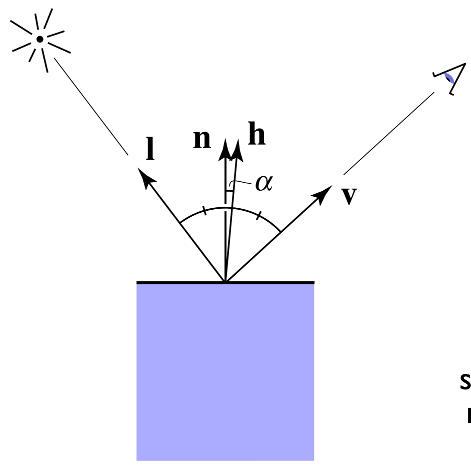
Image so far

```
shade(ray, point, normal, lights) {
    result = ambient;
    for light in lights {
        if (shadow ray not blocked) {
            result += shading contribution;
        }
    }
    return result;
}
```



Specular shading (Blinn-Phong)

- Close to mirror ⇔ half vector near normal
  - Measure "near" by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v}+\mathbf{l}}{\|\mathbf{v}+\mathbf{l}\|}$$

$$\underset{\uparrow}{L_s} = k_s\, I \max(0, \cos\alpha)^n$$

$$= \underset{\uparrow}{k_s}\, I \max(0, \mathbf{n}\cdot\mathbf{h})^n$$

specularly reflected light

specular coefficient

Putting it together

- Usually include ambient, diffuse, Phong in one model

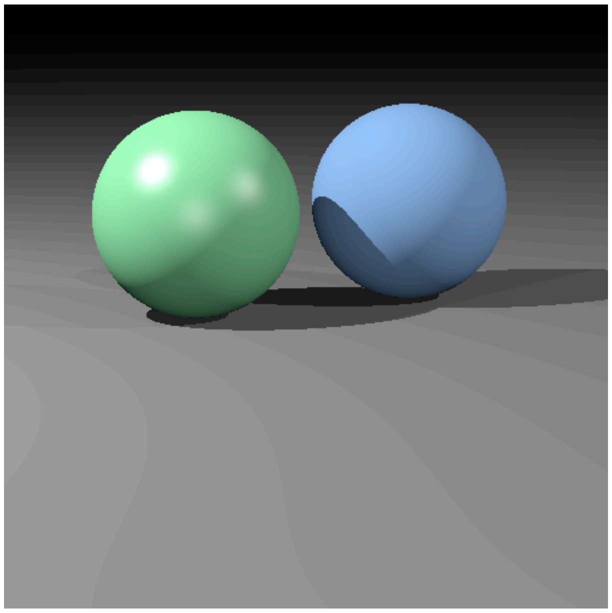
$$L = L_a + L_d + L_s$$
$$= k_a\, I_a + k_d\, I \max(0, \mathbf{n}\cdot\mathbf{l}) + k_s\, I \max(0, \mathbf{n}\cdot\mathbf{h})^n$$

- The final result is the sum over many lights

$$L = L_a + \sum_{i=1}^{N} \left[(L_d)_i + (L_s)_i\right]$$

$$L = k_a\, I_a + \sum_{i=1}^{N} \left[k_d\, I_i \max(0, \mathbf{n}\cdot\mathbf{l}_i) + k_s\, I_i \max(0, \mathbf{n}\cdot\mathbf{h}_i)^n\right]$$

Diffuse + Phong shading



Mirror reflection

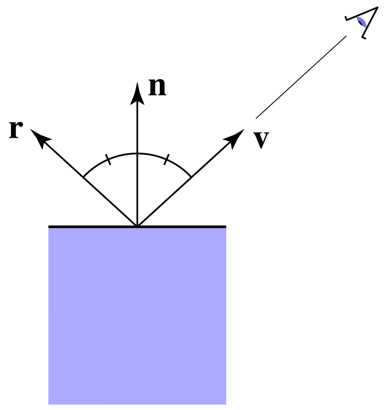
- Consider perfectly shiny surface
  - there isn't a highlight
  - instead there's a reflection of other objects
- Can render this using recursive ray tracing
  - to find out mirror reflection color, ask what color is seen from surface point in reflection direction
  - already computing reflection direction for Phong…
- "Glazed" material has mirror reflection and direct

$$L = L_a + L_d + L_m$$

  - where L_m is evaluated by tracing a new ray

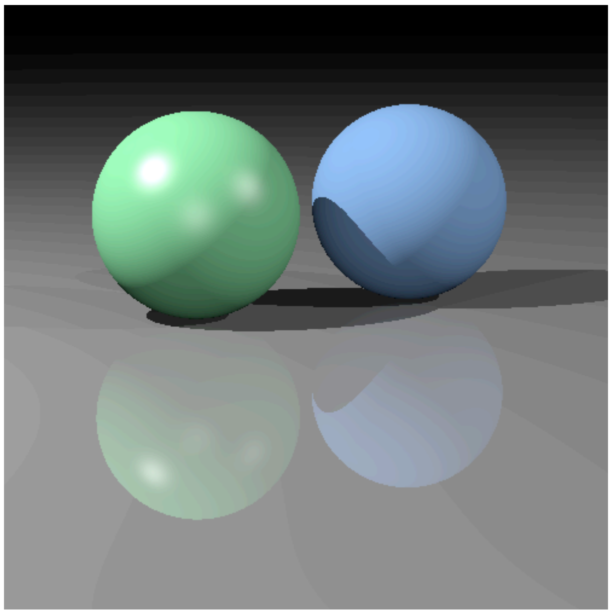
Mirror reflection

- Intensity depends on view direction
  - reflects incident light from mirror direction



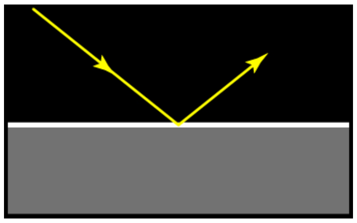
$$\begin{aligned}\mathbf{r} &= \mathbf{v} + 2((\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}) \\ &= 2(\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}\end{aligned}$$

Diffuse + mirror reflection (glazed)

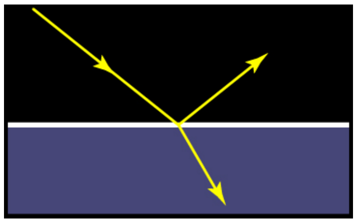

(glazed material on floor)

Simple materials

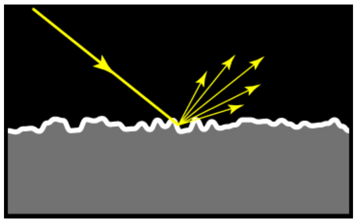
metal

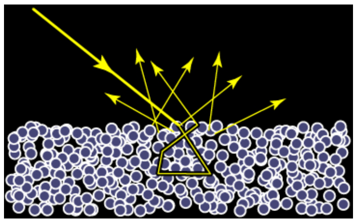


dielectric

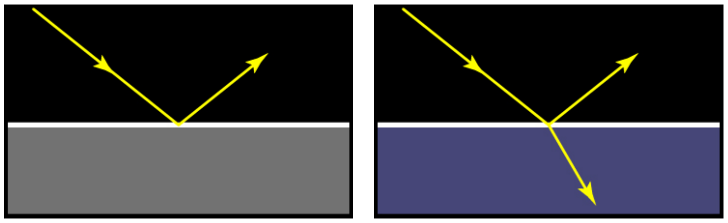


Adding microgeometry

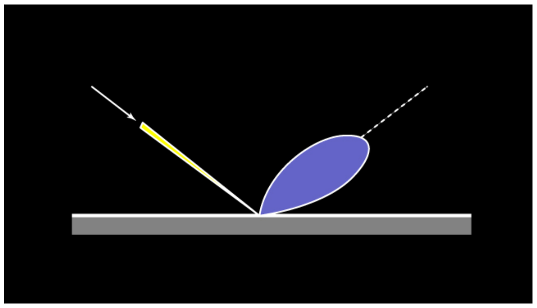




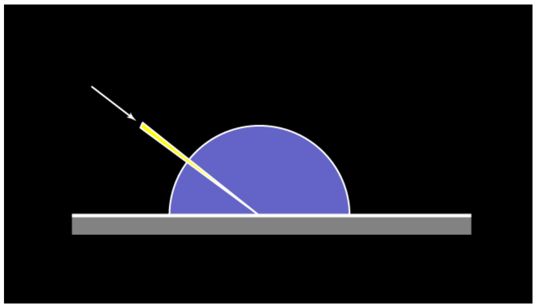
Classic reflection behavior



ideal specular (Fresnel)



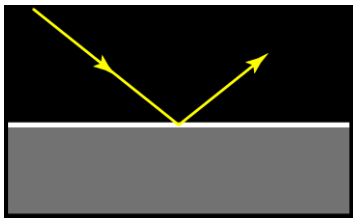
rough specular



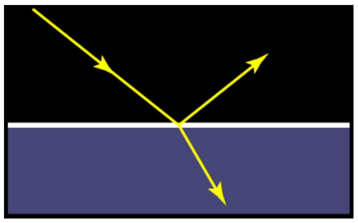
Lambertian

Specular reflection

- Smooth surfaces of pure materials have ideal specular reflection (said this before)
  - Metals (conductors) and dielectrics (insulators) behave differently
- Reflectance (fraction of light reflected) depends on angle



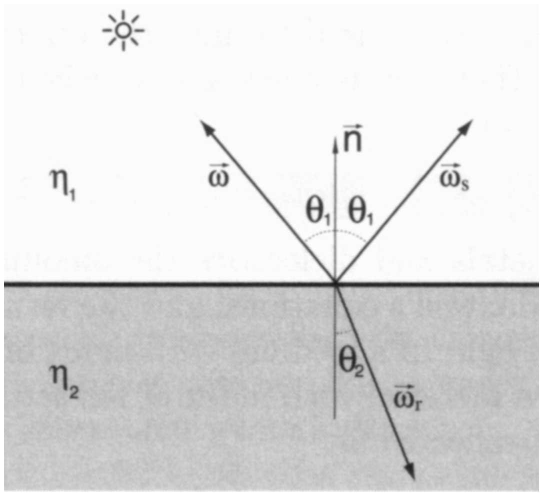
metal



dielectric

Snell's Law

- Tells us where the refracted ray goes
  - a.k.a. transmission vector
- Computation
  - ratio of sines is ratio of in-plane components
  - project to surface; scale by eta ratio; recompute normal-direction component
  - total internal reflection



$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

Example values of *n*:
air: 1.00;
water: 1.33–1.34;
window glass: 1.51;
optical glass: 1.49–1.92;
diamond: 2.42.

Computing the Transmission Vector, t

$$n \sin\theta = n_t \sin\phi.$$

Computing the sine of an angle between two vectors is usually not as convenient as computing the cosine, which is a simple dot product for the unit vectors such as we have here. Using the trigonometric identity $\sin^2\theta + \cos^2\theta = 1$, we can derive a refraction relationship for cosines:

$$\cos^2\phi = 1 - \frac{n^2\left(1 - \cos^2\theta\right)}{n_t^2}.$$

Note that if n and n_t are reversed, then so are θ and ϕ as shown on the right of Figure 13.1.

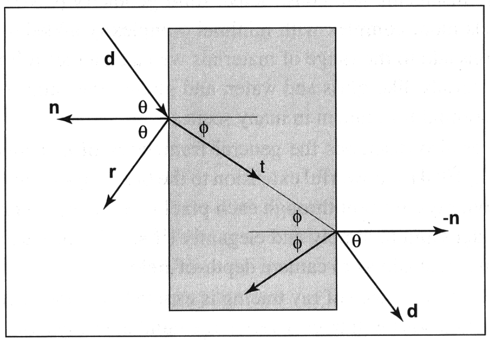
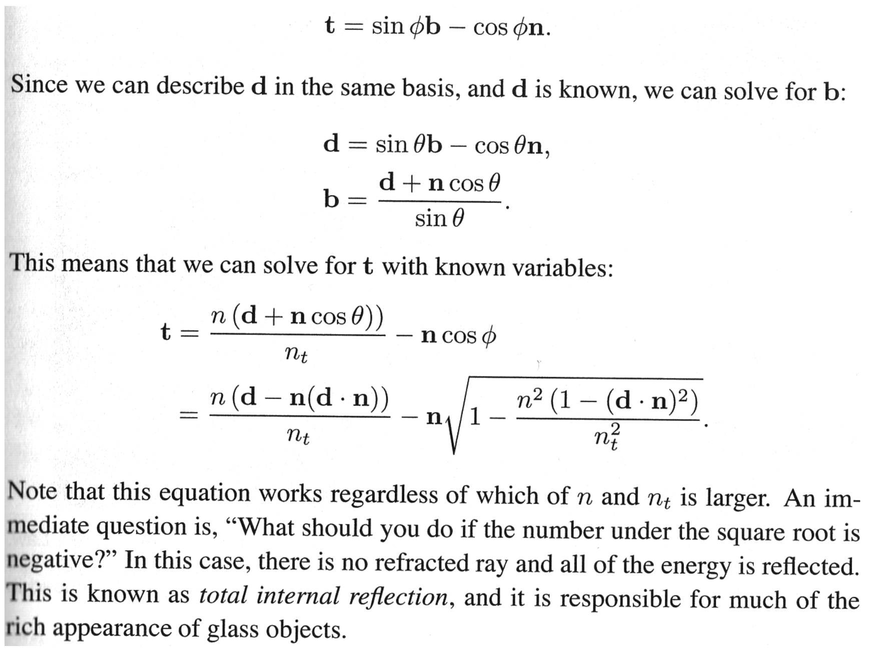


Figure 13.1. Snell's Law describes how the angle ϕ depends on the angle θ and the refractive indices of the object and the surrounding medium.

Computing the Transmission Vector, t



$$\mathbf{t} = \sin\phi\mathbf{b} - \cos\phi\mathbf{n}.$$

Since we can describe $\mathbf{d}$ in the same basis, and $\mathbf{d}$ is known, we can solve for $\mathbf{b}$:

$$\mathbf{d} = \sin\theta\mathbf{b} - \cos\theta\mathbf{n},$$
$$\mathbf{b} = \frac{\mathbf{d} + \mathbf{n}\cos\theta}{\sin\theta}.$$

This means that we can solve for $\mathbf{t}$ with known variables:

$$\begin{aligned}\mathbf{t} &= \frac{n\left(\mathbf{d} + \mathbf{n}\cos\theta\right)}{n_t} - \mathbf{n}\cos\phi \\ &= \frac{n\left(\mathbf{d} - \mathbf{n}(\mathbf{d}\cdot\mathbf{n})\right)}{n_t} - \mathbf{n}\sqrt{1 - \frac{n^2\left(1 - (\mathbf{d}\cdot\mathbf{n})^2\right)}{n_t^2}}.\end{aligned}$$

Note that this equation works regardless of which of n and n_t is larger. An immediate question is, "What should you do if the number under the square root is negative?" In this case, there is no refracted ray and all of the energy is reflected. This is known as *total internal reflection*, and it is responsible for much of the rich appearance of glass objects.

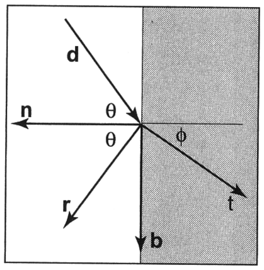


Figure 13.2. The vectors **n** and **b** form a 2D orthonormal basis that is parallel to the transmission vector **t**.

Ray tracing dielectrics

- Like a simple mirror surface, use recursive ray tracing
- But we need two rays
  - One reflects off the surface (same as mirror ray)
  - The other crosses the surface (computed using Snell's law)
    - Doesn't always exist (total internal reflection)
- Splitting into two rays, recursively, creates a ray tree
  - Very many rays are traced per viewing ray
  - Ways to prune the tree
    - Limit on ray depth
    - Limit on ray attenuation