Ray Tracing: shading

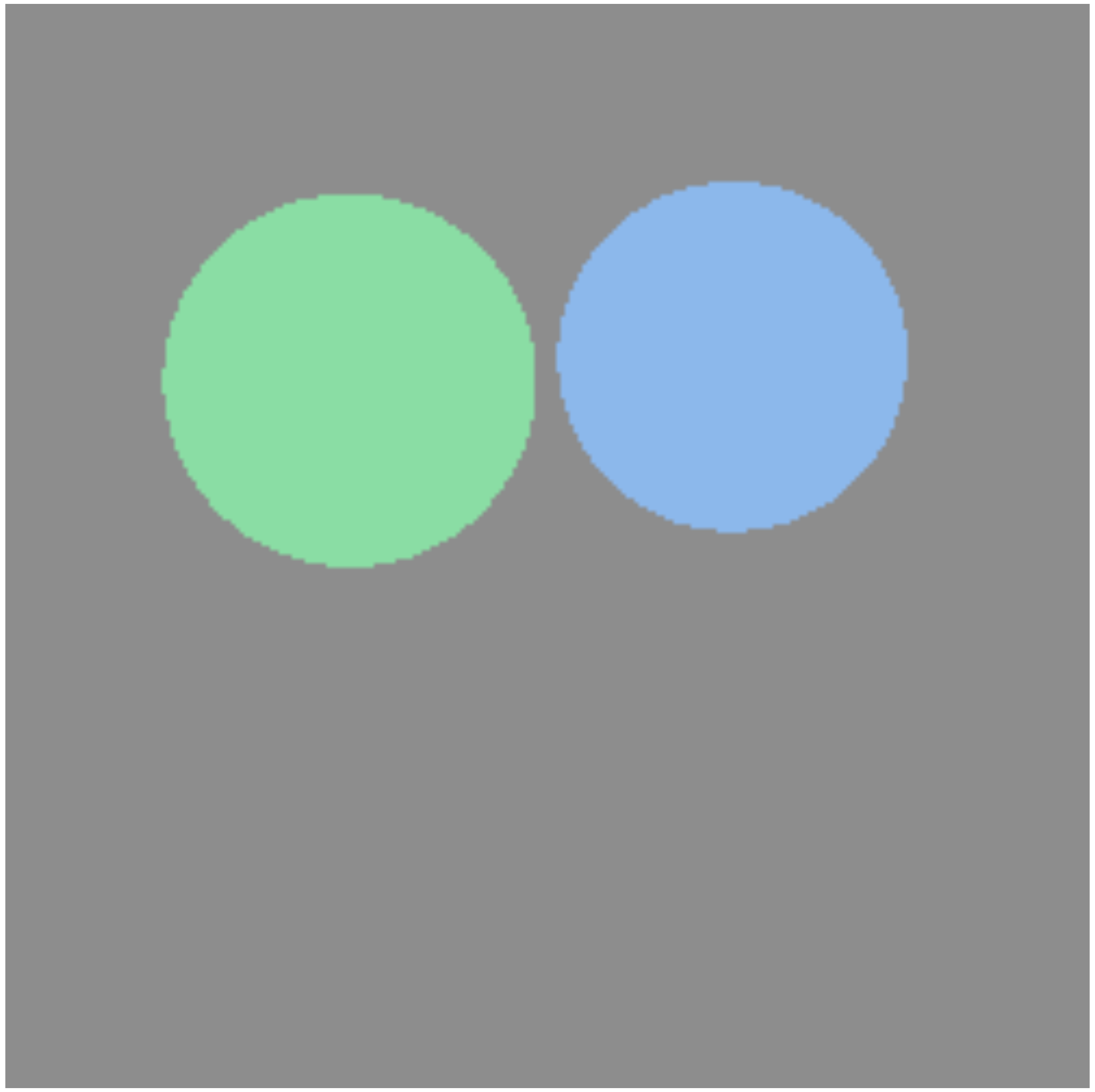
CS 4620 Lecture 6

© 2018 Steve Marschner •

Image so far

- **With eye ray generation and scene intersection**

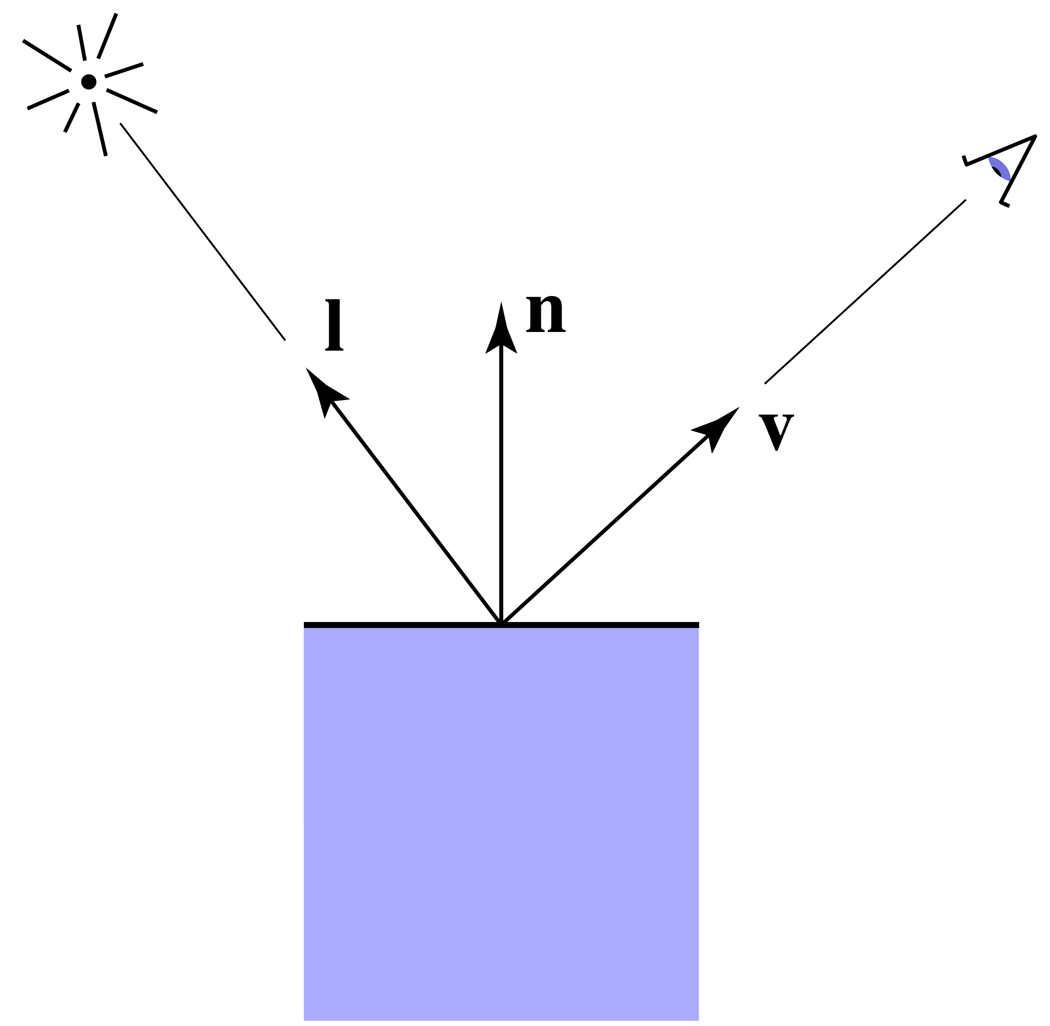
```
for 0 <= iy < ny
    for 0 <= ix < nx {
        ray = camera.getRay(ix, iy);
        c = scene.trace(ray, 0, +inf);
        image.set(ix, iy, c);
    }

...

Scene.trace(ray, tMin, tMax) {
    surface, t = surfs.intersect(ray, tMin, tMax);
    if (surface != null) return surface.color();
    else return black;
}
```



© 2018 Steve Marschner

Shading

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



© 2018 Steve Marschner

Shading philosophy

- **Goals of shading depend on purpose of image**
  - visualization, CAD: maximize visual clarity
  - visual effects, advertising: maximize resemblance to reality
  - animation, games: somewhere in between
- **Basic starting point: physics of light reflection**
  - a set of useful approximations to real surfaces
  - can remove things for simplicity/clarity
  - can add things for increased accuracy/realism

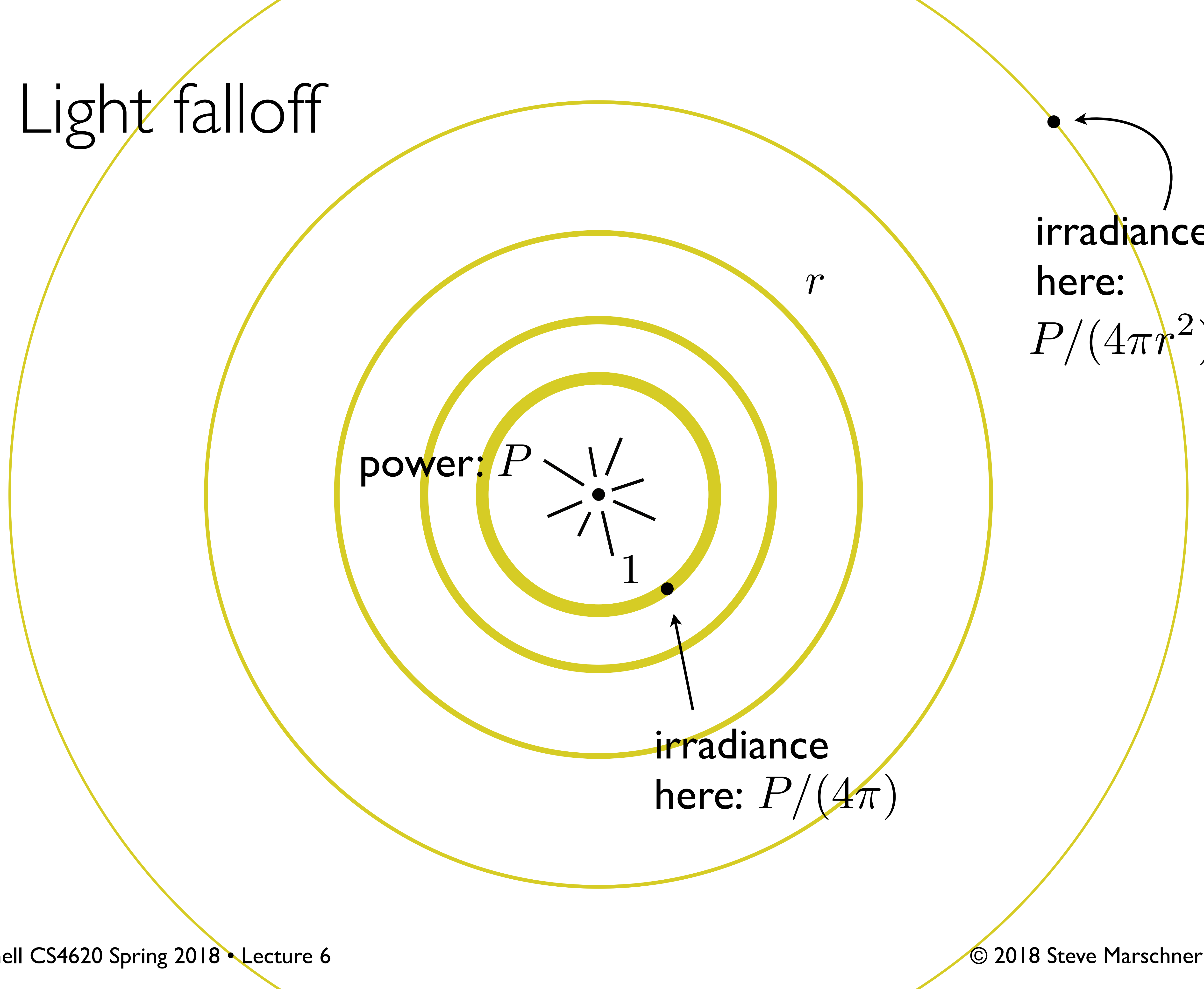
© 2018 Steve Marschner •

Light

- **Think of light as a flow of particles through space**
  - disregarding wave nature: polarization, interference, diffraction
  - for now disregarding color: only how much light
- **Sources of light**
  - point sources (a flashlight) ← we will stick to this for now.
  - directional sources (the sun)
  - area sources (a fluorescent tube)
  - environment sources (the sky)

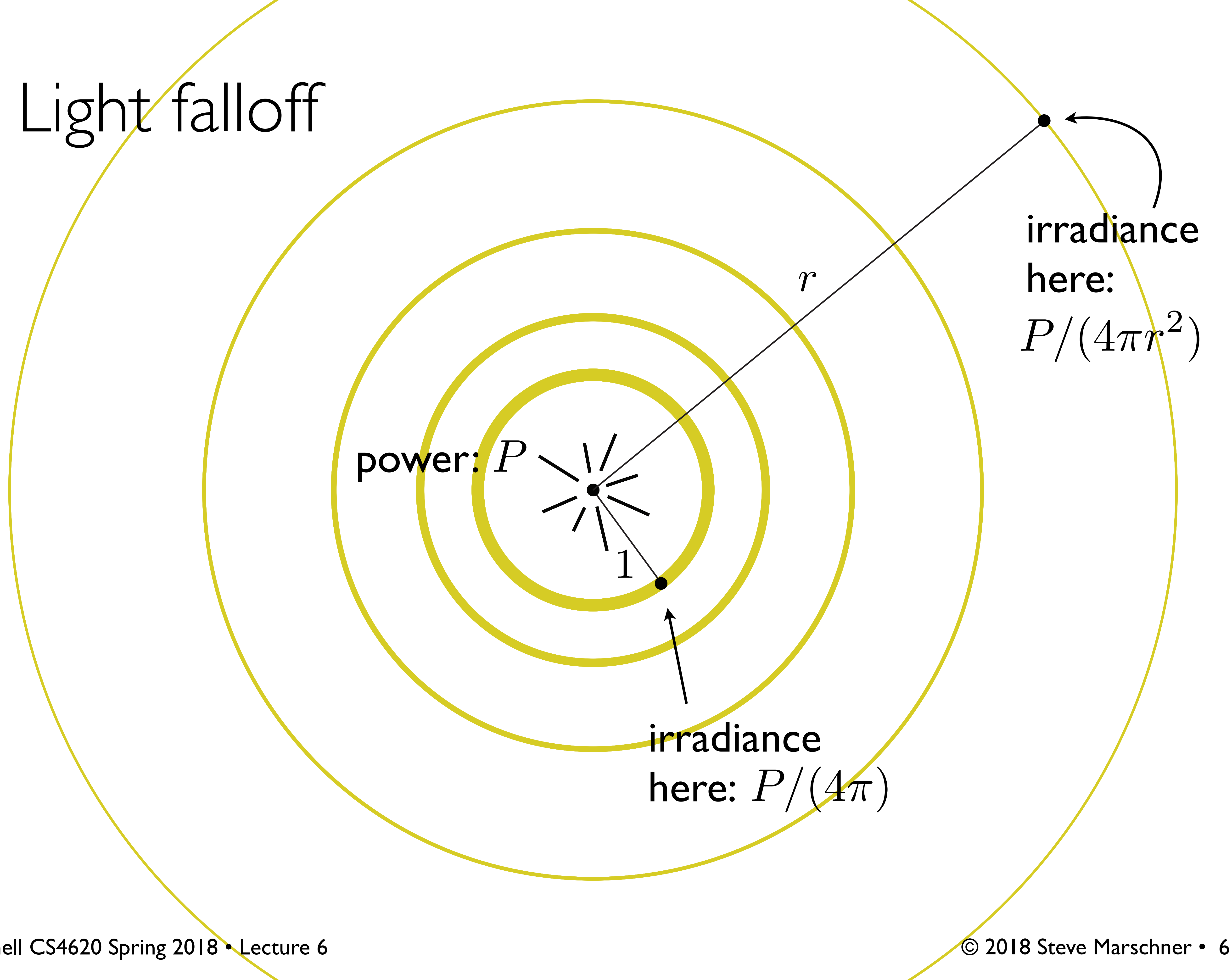
© 2018 Steve Marschner •

Light falloff



© 2018 Steve Marschner

Light falloff



© 2018 Steve Marschner

Irradiance from isotropic point source

- **A sphere surrounding the source receives all the power**
- **A small, flat surface of area *A* facing the source receives a fraction (area of surface) / (area of sphere) of that power:**

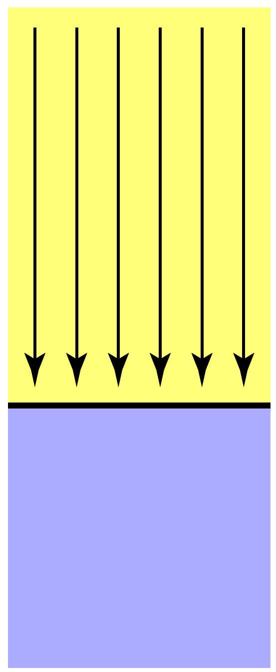
$$P_A = P \frac{A}{4\pi r^2}$$

- **Irradiance is power per unit area:**

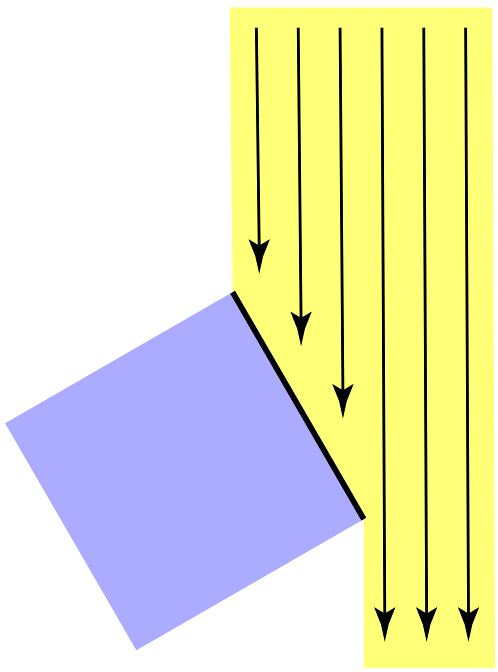
$$E = P_A / A = \frac{P}{4\pi r^2} = \underbrace{\frac{P}{4\pi}}_{\text{intensity}} \underbrace{\frac{1}{r^2}}_{\text{geometry factor}}$$

© 2018 Steve Marschner

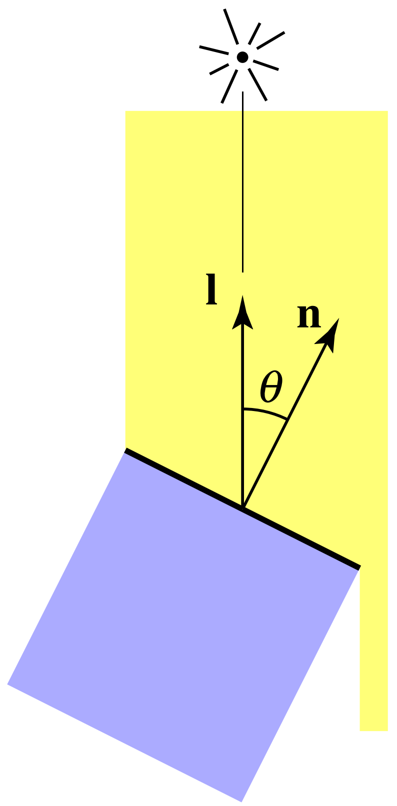
Lambert's cosine law



Top face of cube receives a certain amount of light



Top face of 60° rotated cube intercepts half the light



In general, light per unit area is proportional to cos θ = **l • n**

© 2018 Steve Marschner

Irradiance from isotropic point source

- **A surface of area *A* facing at an angle to the source receives a factor of $\cos\theta$ less light:**

$$P_A = P\frac{A\cos\theta}{4\pi r^2}$$

- **Irradiance is power per unit area:**

$$E = P_A/A = \underbrace{\frac{P}{4\pi}}_{\text{intensity}}\underbrace{\frac{\cos\theta}{r^2}}_{\text{geometry factor}}$$

© 2018 Steve Marschner •

Diffuse reflection

- **Simplest reflection model**
- **Reflected light is independent of view direction**
- **Reflected light is proportional to irradiance**
  - constant of proportionality is the diffuse reflection coefficient

$$L_d = k_d E$$

- **More useful to think in terms of reflectance**
  - reflectance is the fraction reflected (between 0 and 1)

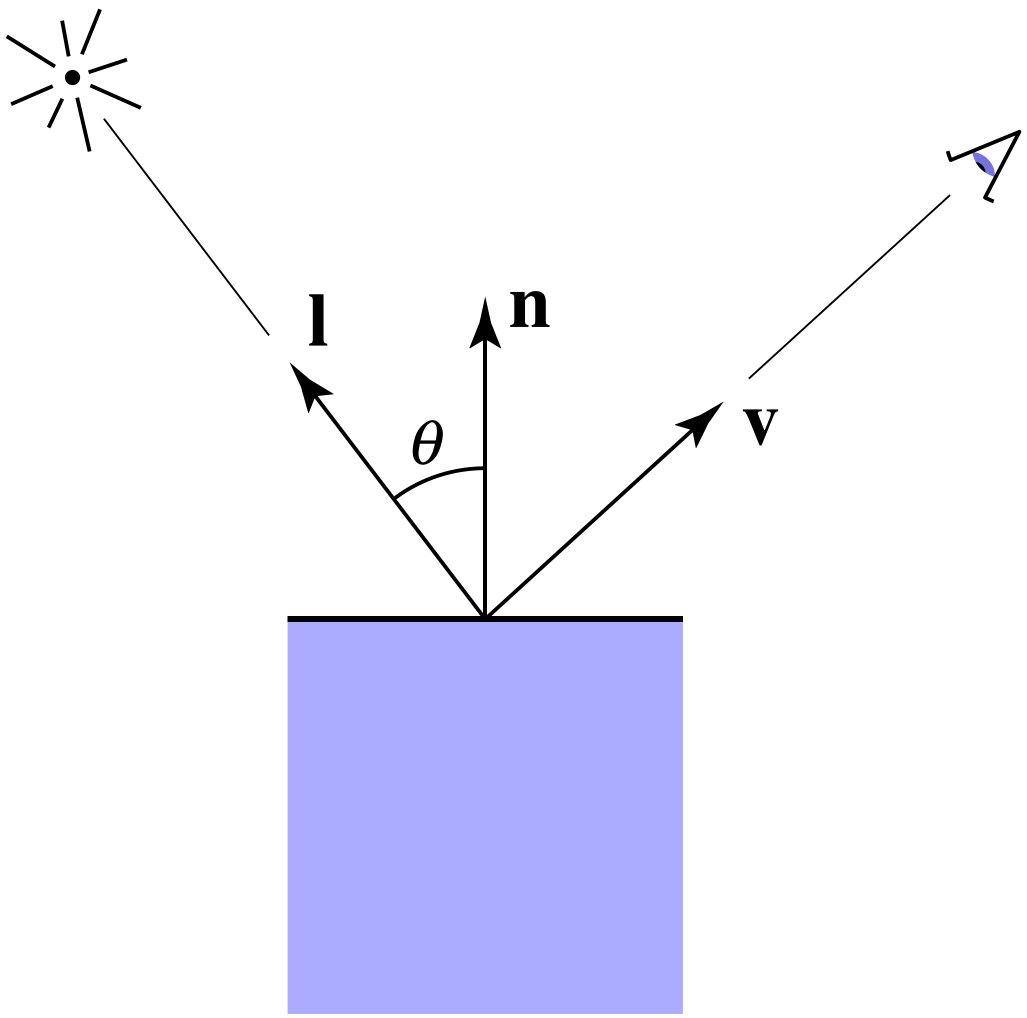
$$L_d = \frac{R_d}{\pi} E$$

  - will have to explain the factor of pi later

© 2018 Steve Marschner

Lambertian shading

- **Shading independent of view direction**



diffuse reflectance

$$L_d = \underbrace{\frac{R}{\pi}}_{\text{diffuse coefficient}} \overbrace{\frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} I}^{\text{irradiance from source}}$$

diffusely reflected radiance

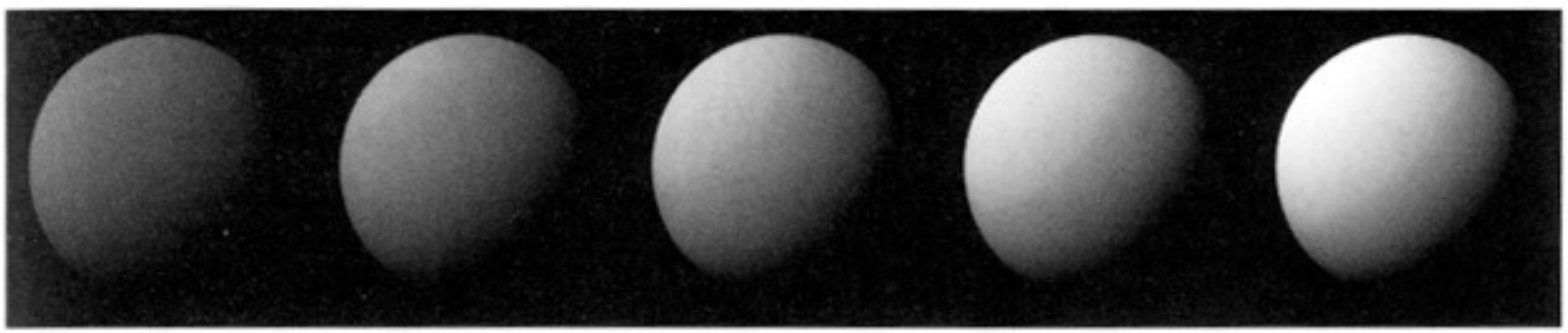
distance to source

intensity of source

© 2018 Steve Marschner

Lambertian shading

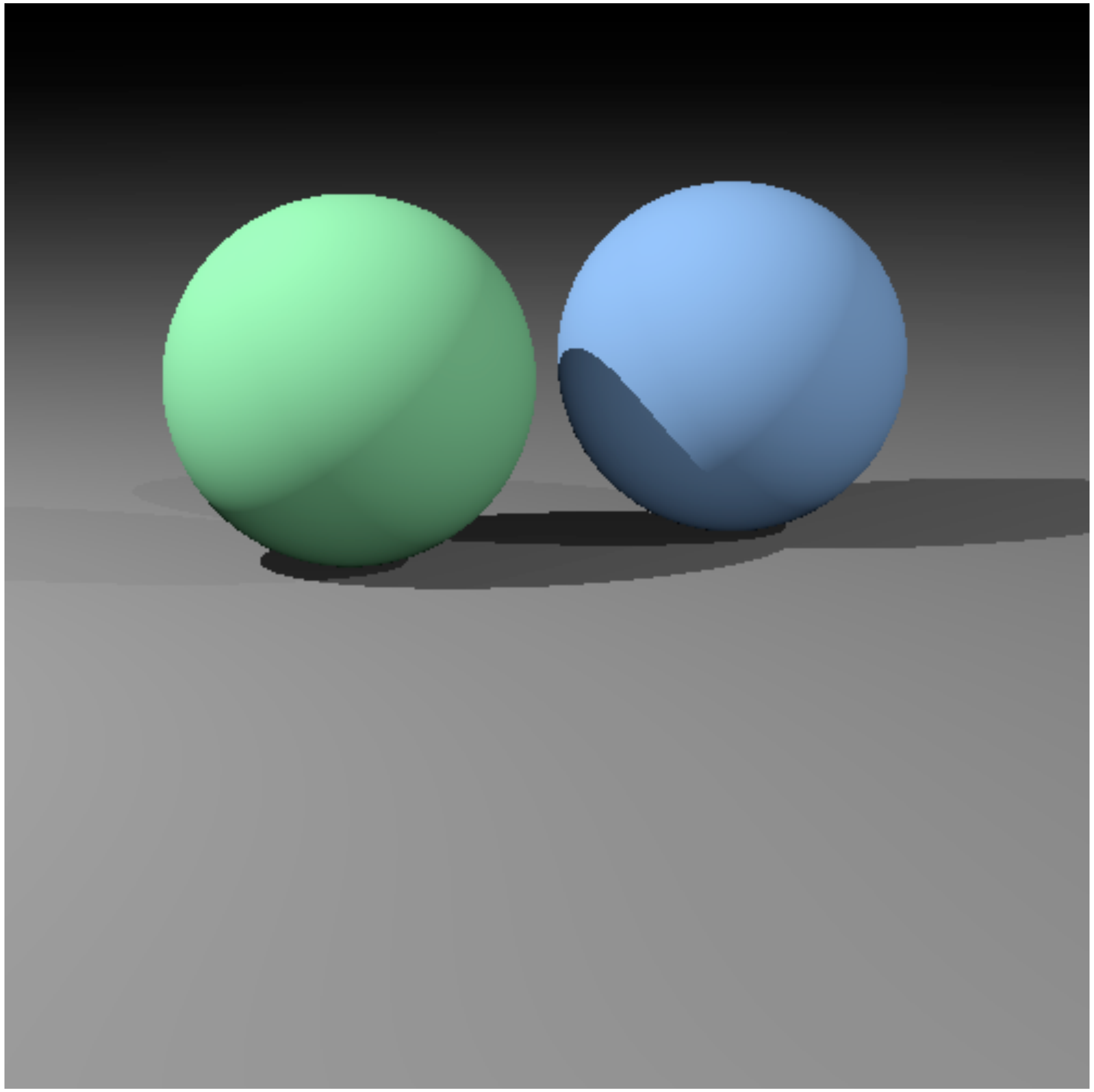
- **Produces matte appearance**



$k_d \longrightarrow$

[Foley et al.]

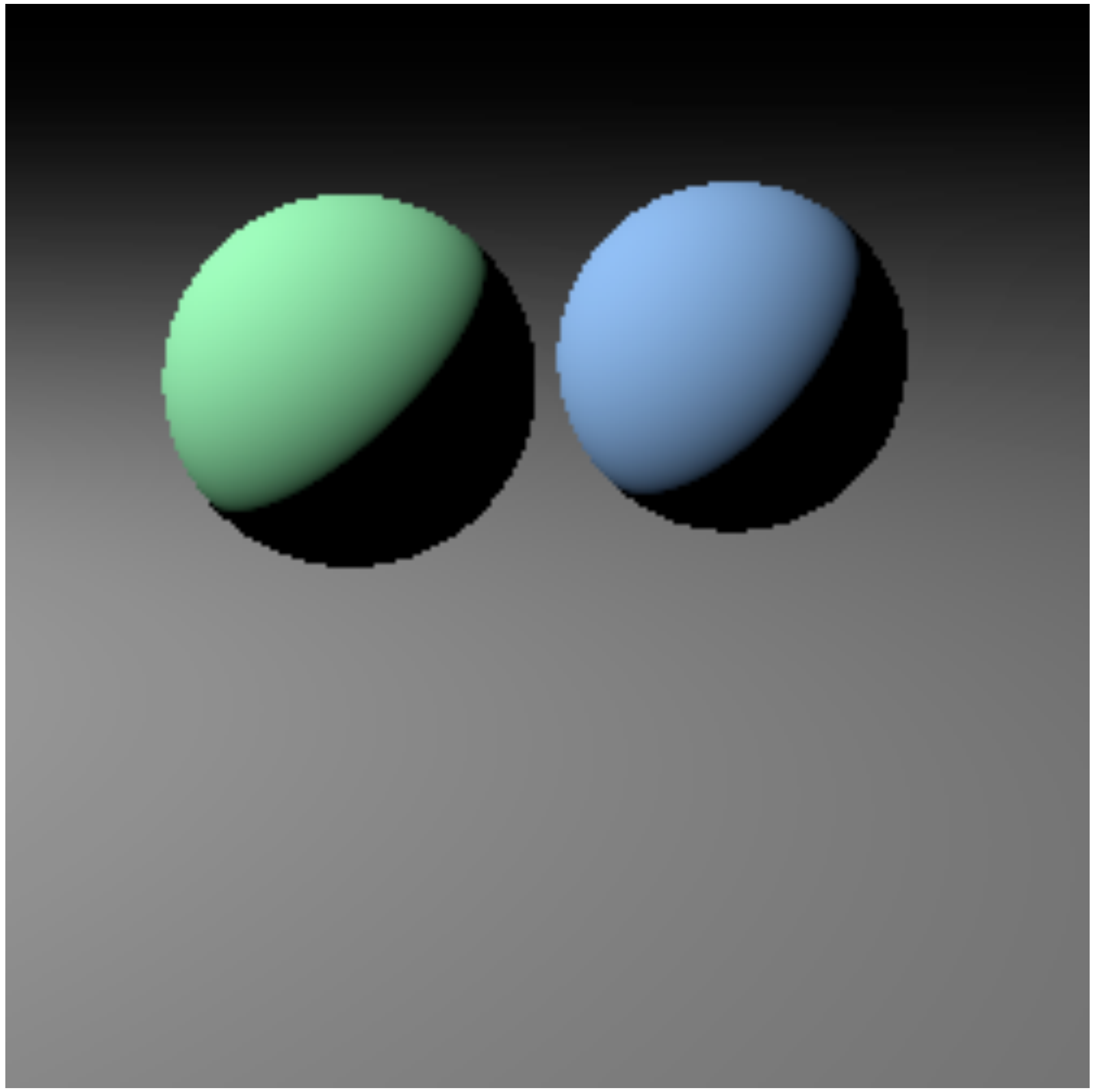
Diffuse shading



© 2018 Steve Marschner

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
}
```



© 2018 Steve Marschner

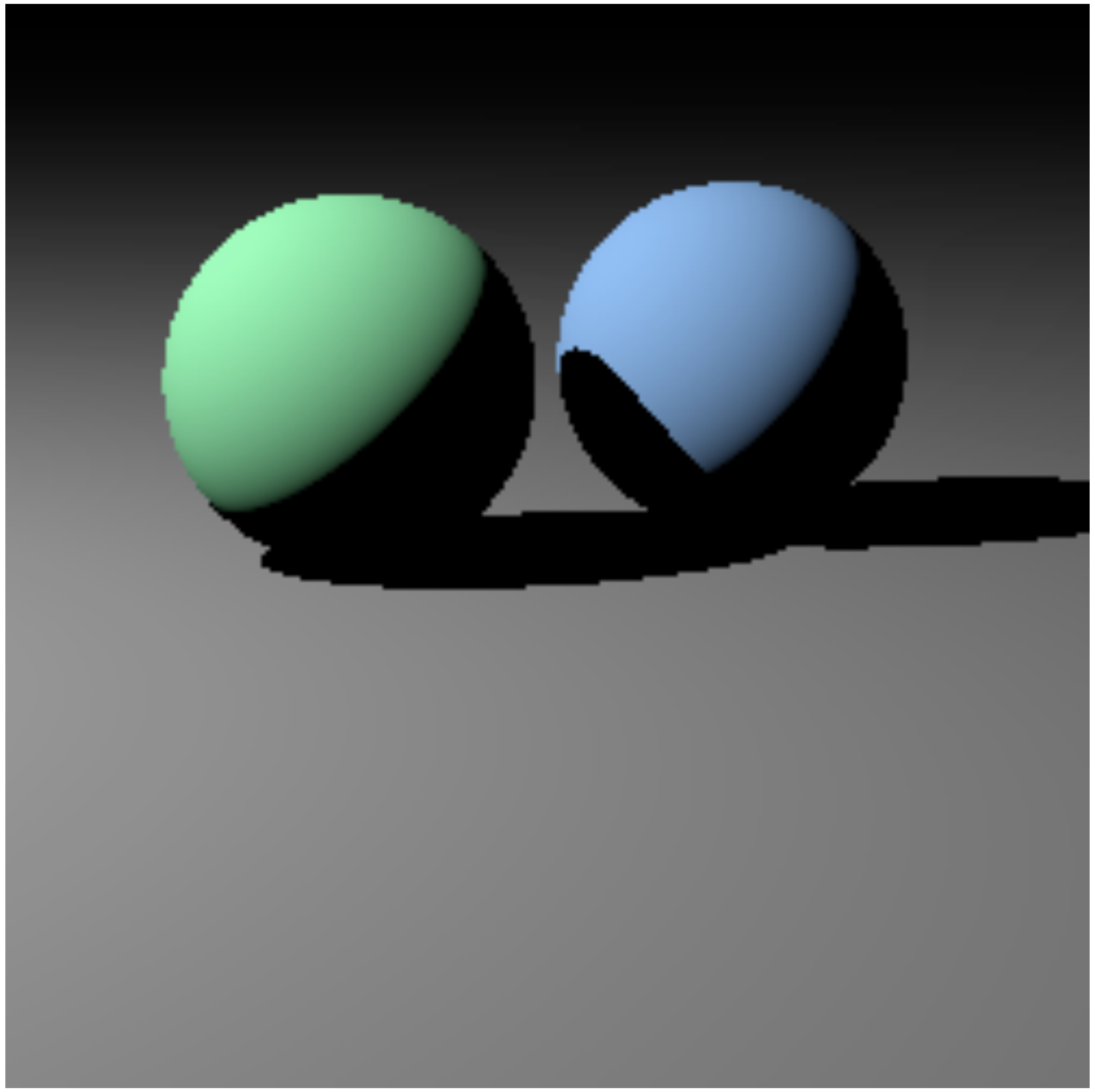
Shadows

- **Surface is only illuminated if nothing blocks the light**
  – i.e. if the surface can "see" the light
- **With ray tracing it's easy to check**
  – just intersect a ray with the scene!

© 2018 Steve Marschner

Image so far

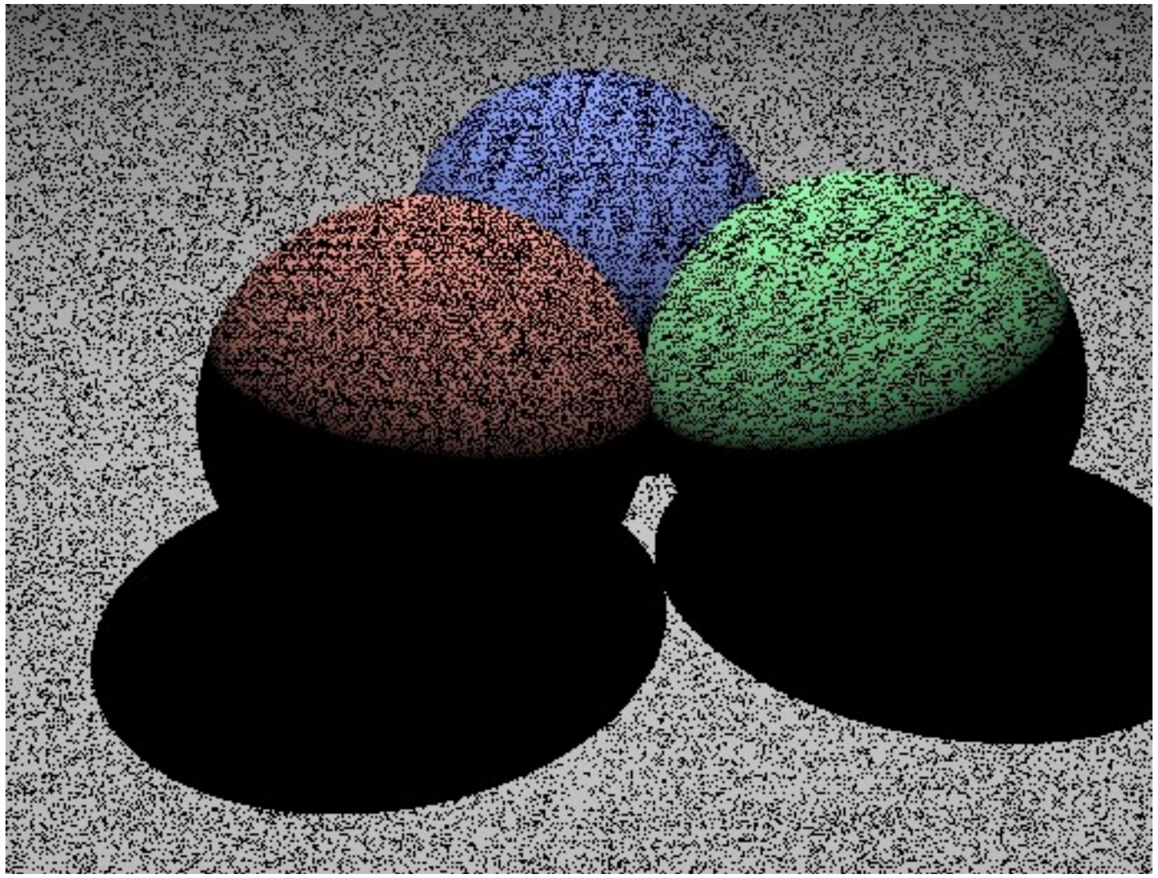
```
Surface.shade(ray, point, normal, light) {
    shadRay = (point, light.pos – point);
    if (shadRay not blocked) {
        v = –normalize(ray.direction);
        l = normalize(light.pos – point);
        // compute shading
    }
    return black;
}
```



© 2018 Steve Marschner

Shadow rounding errors

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

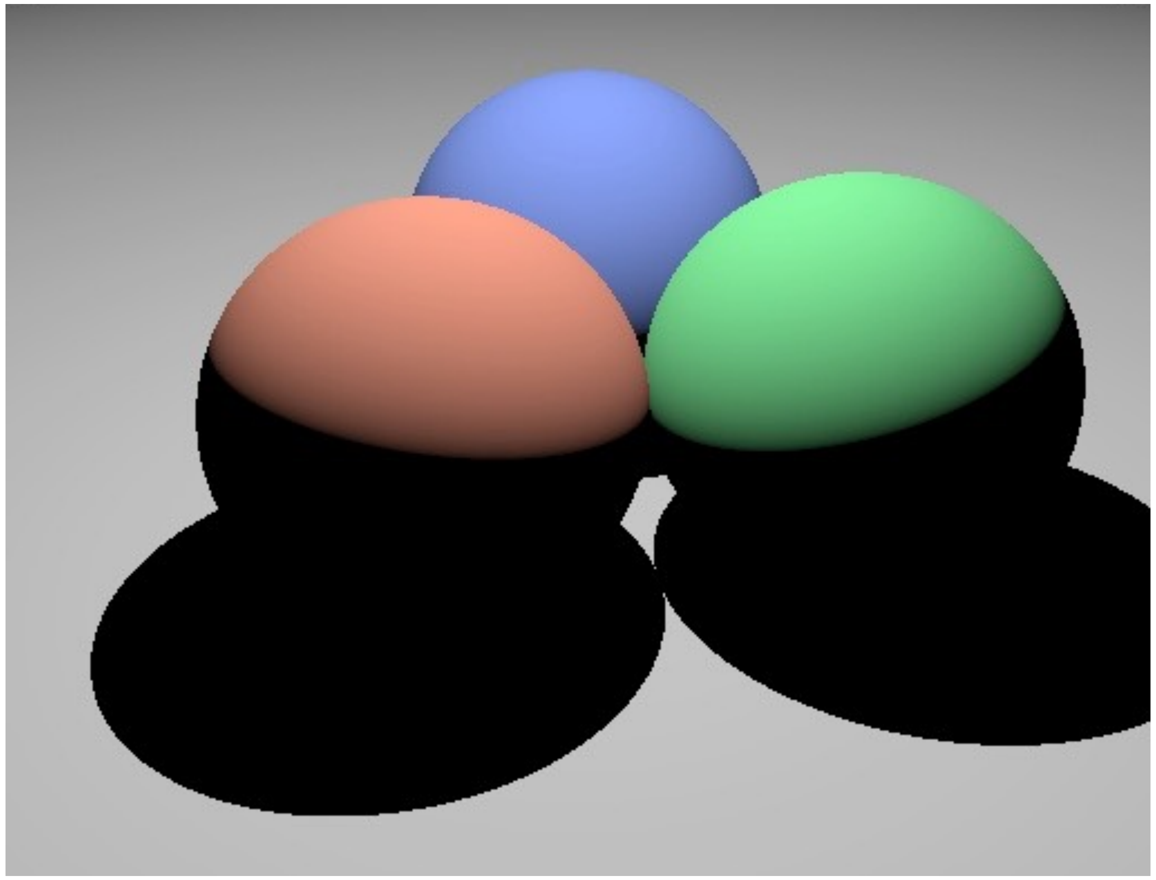


- **What's going on?**
  - hint: at what ***t*** does the shadow ray intersect the surface you're shading?

© 2018 Steve Marschner

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**

© 2018 Steve Marschner

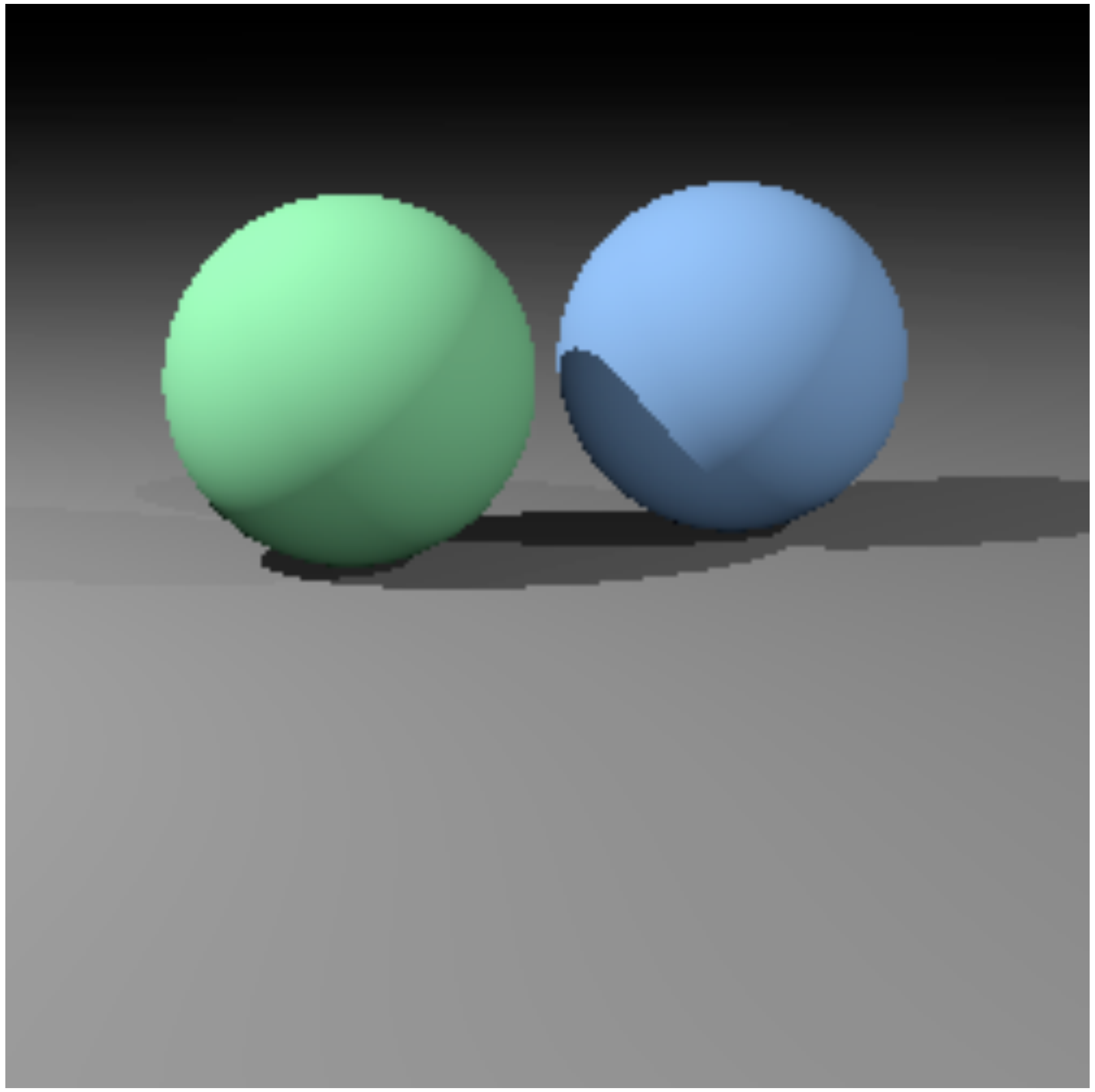
Multiple lights

- **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…

© 2018 Steve Marschner

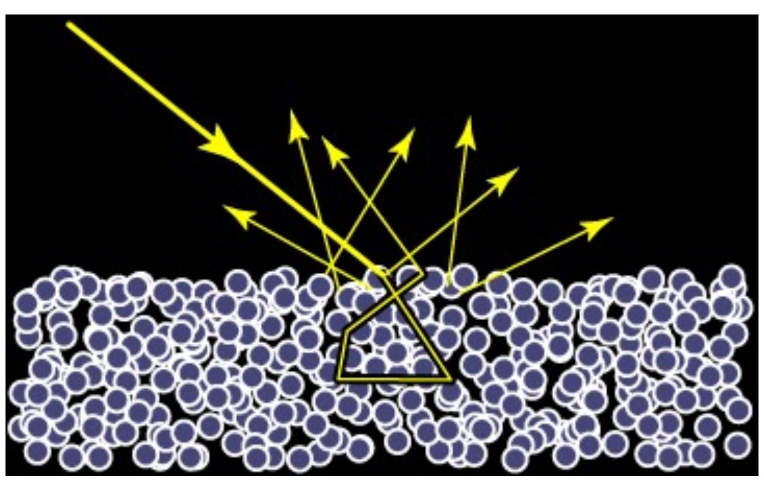
Image so far

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

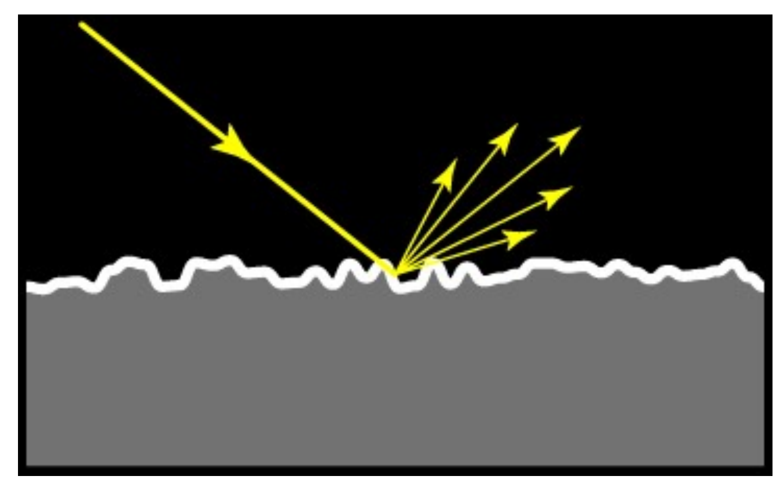


© 2018 Steve Marschner

Specular shading



diffuse

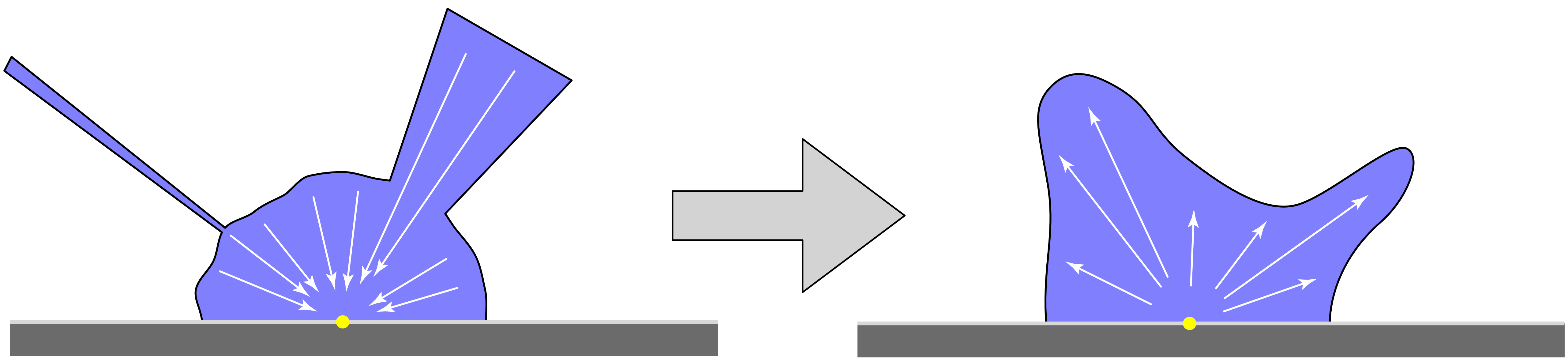


specular

© 2018 Steve Marschner

Light reflection: full picture

- **when writing a shader, think like a bug standing on the surface**
  - bug sees an incident distribution of light arriving at the surface
  - physics question: what is the outgoing distribution of light?

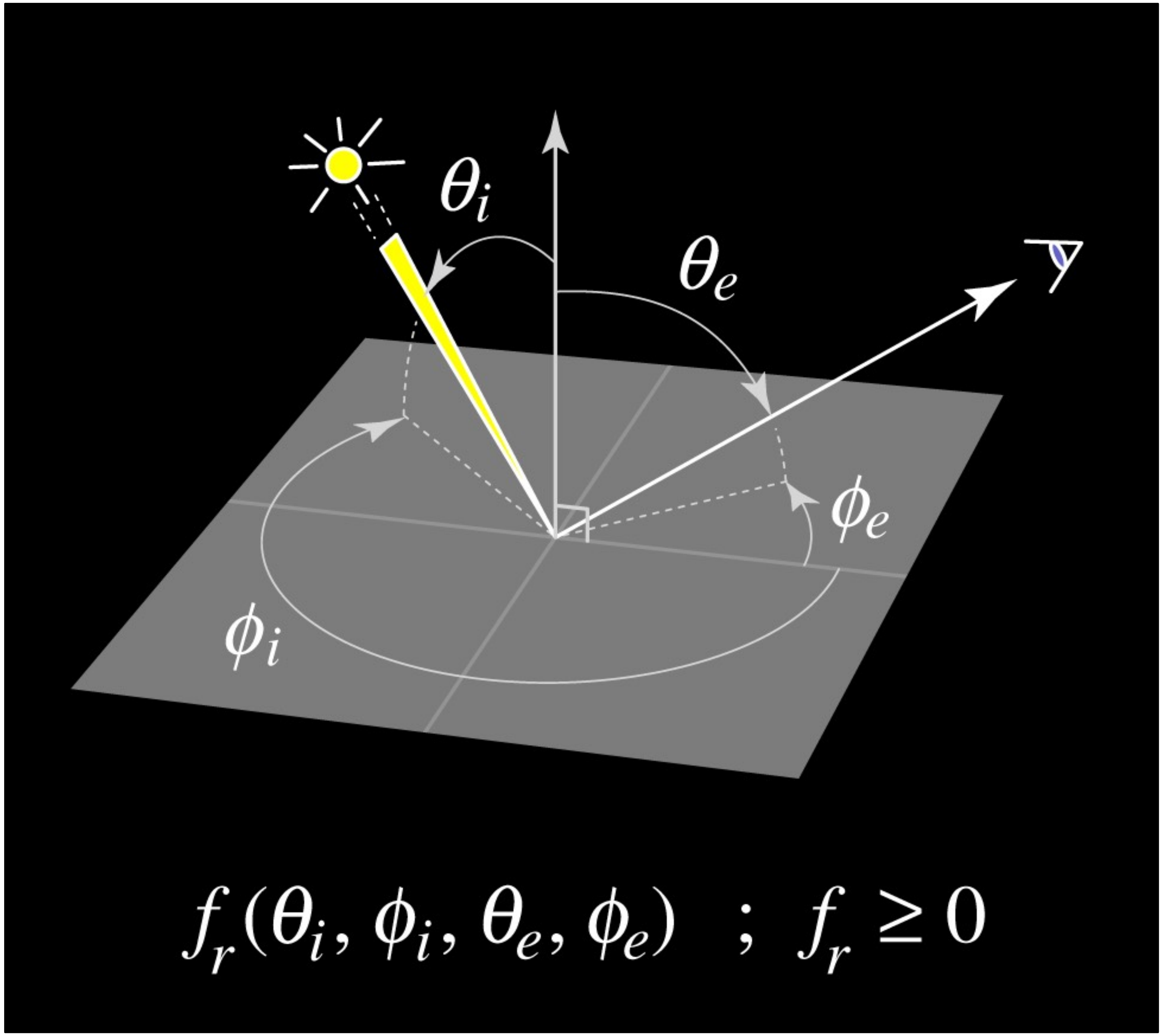


incident distribution
(function of direction)

reflected distribution
(function of direction)

© 2018 Steve Marschner

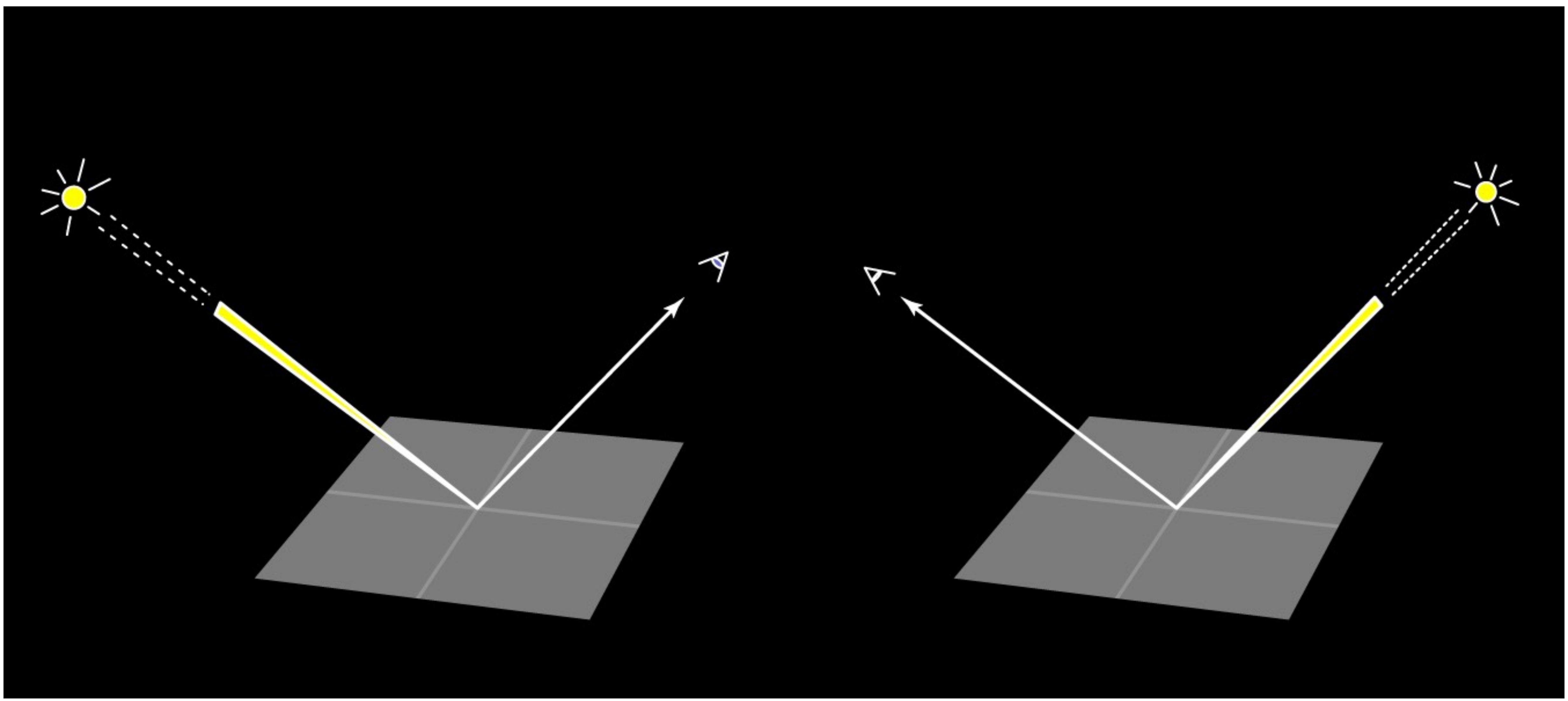
BRDF



Bidirectional Reflectance Distribution Function

Reciprocity

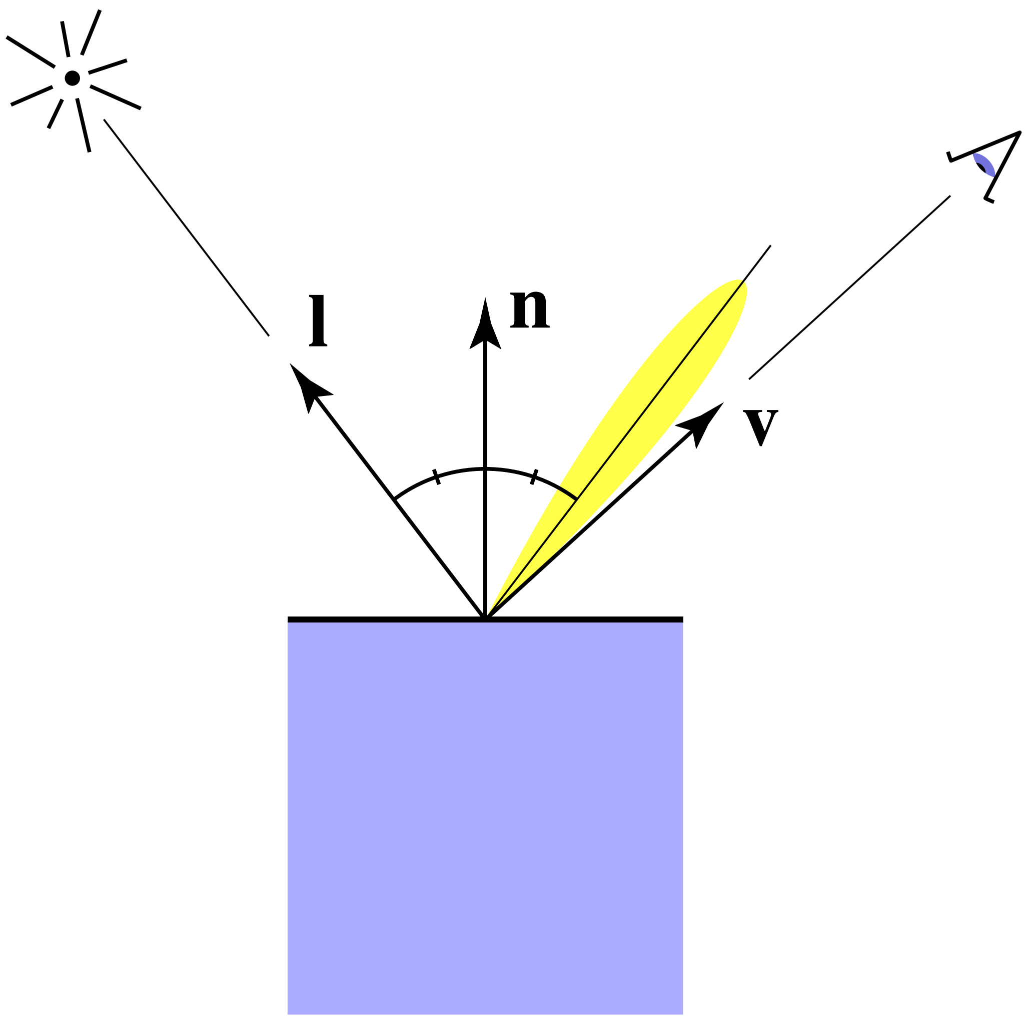
- **Interchanging arguments**
- **Physical requirement**



© 2018 Steve Marschner

Specular reflection

- **Intensity depends on view direction**
  - bright near mirror configuration

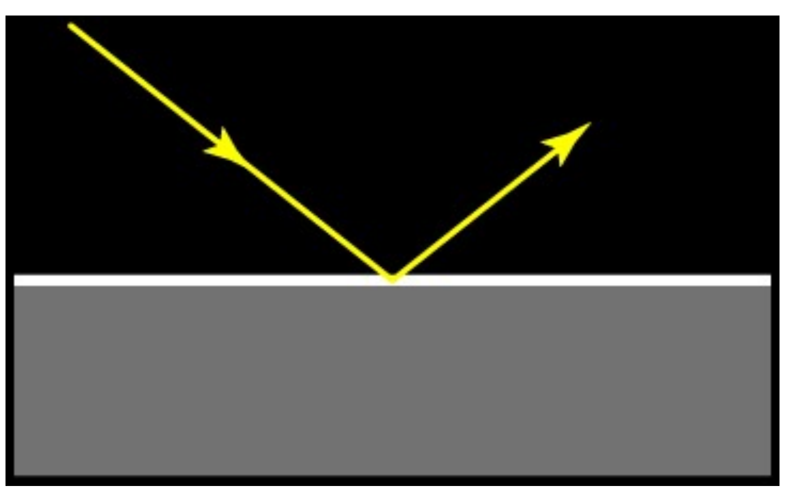


Caution: in notes and assignment, $\mathbf{v}$ is called ω_r and $\mathbf{l}$ is called ω_i. No meaningful difference, just notational.

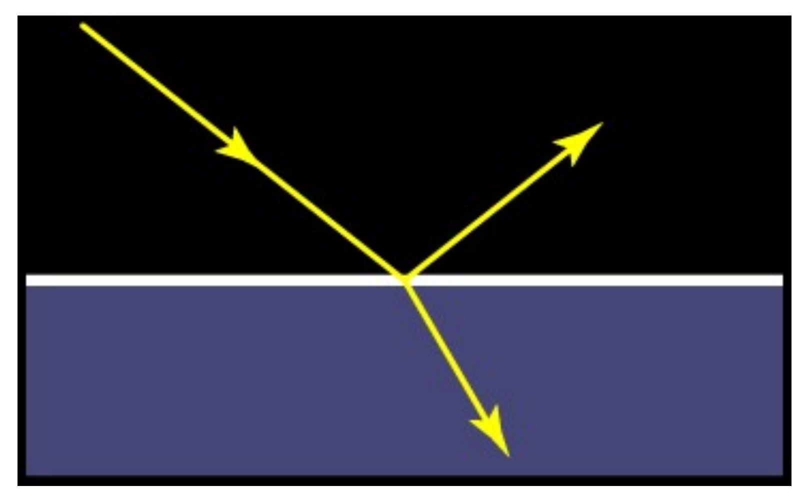
© 2018 Steve Marschner • 25

Smooth surfaces

metal



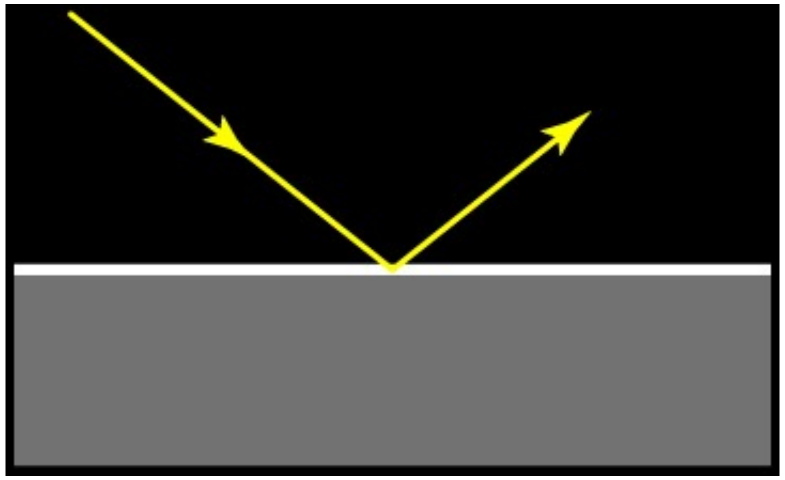
dielectric



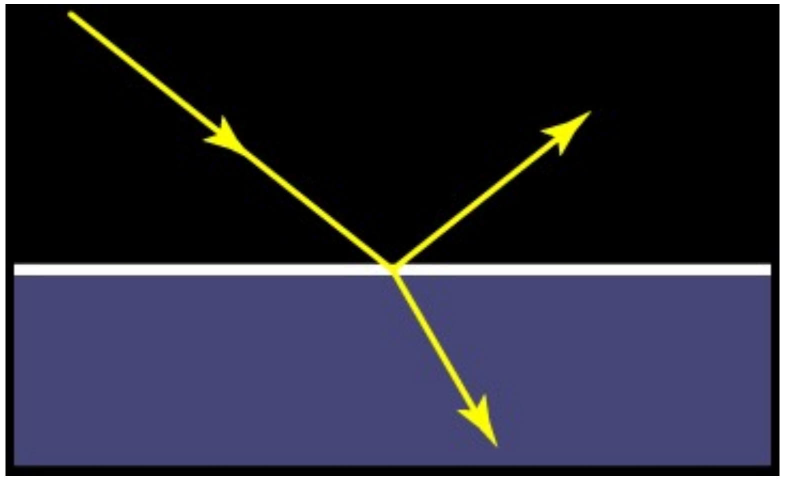
© 2018 Steve Marschner

Ideal specular reflection

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



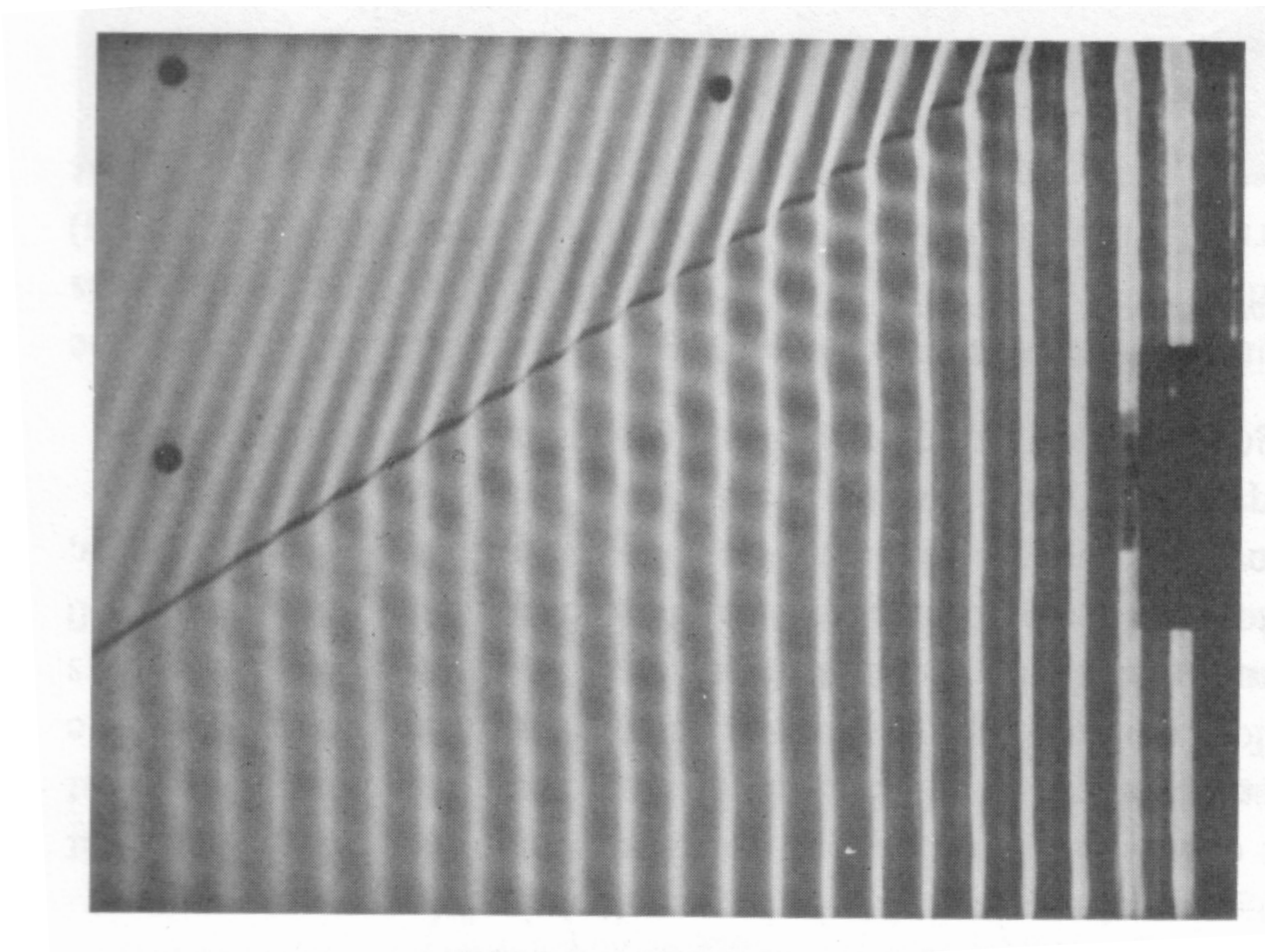
metal



dielectric

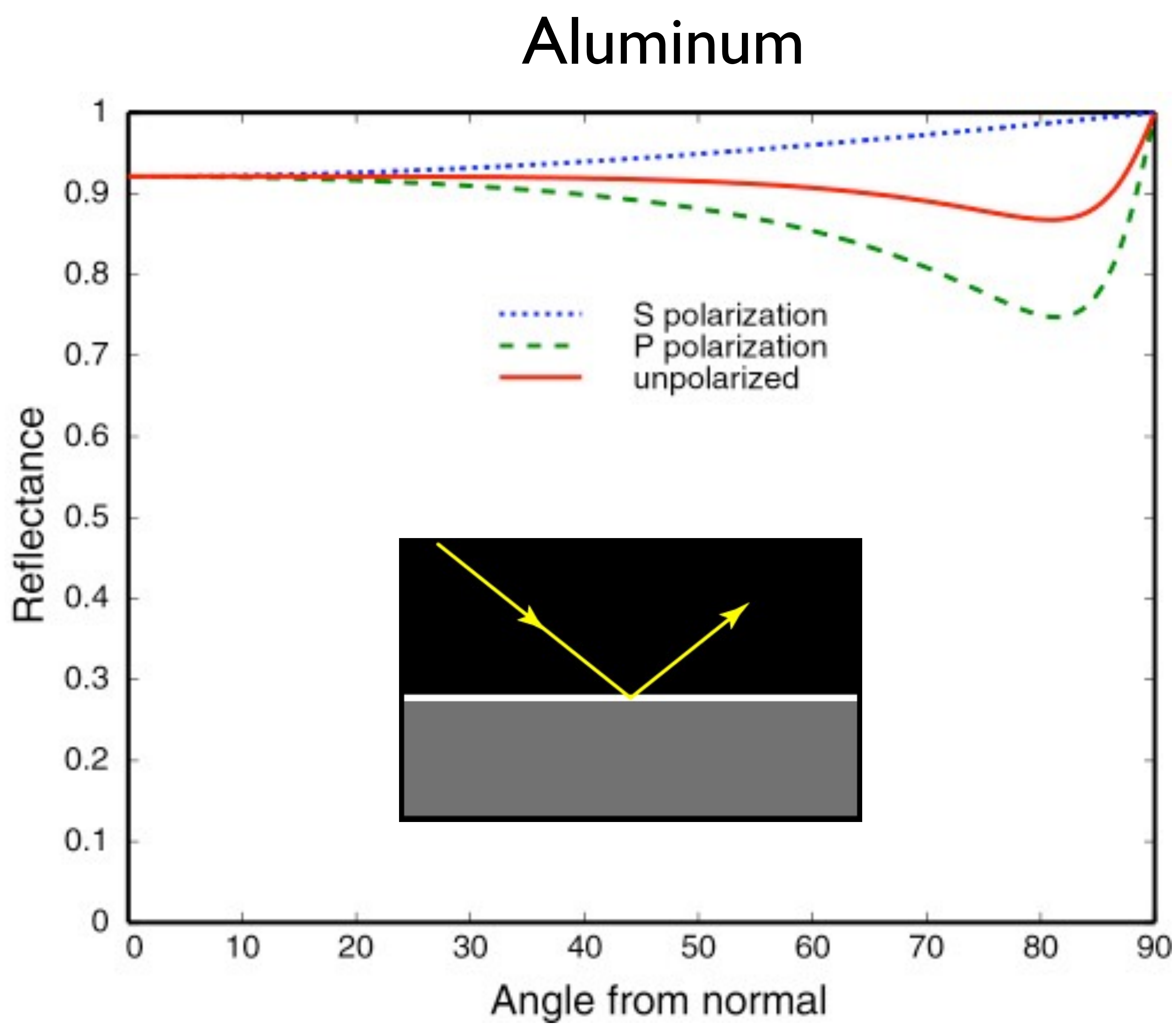
© 2018 Steve Marschner

Refraction at boundary of media



Specular reflection from metal

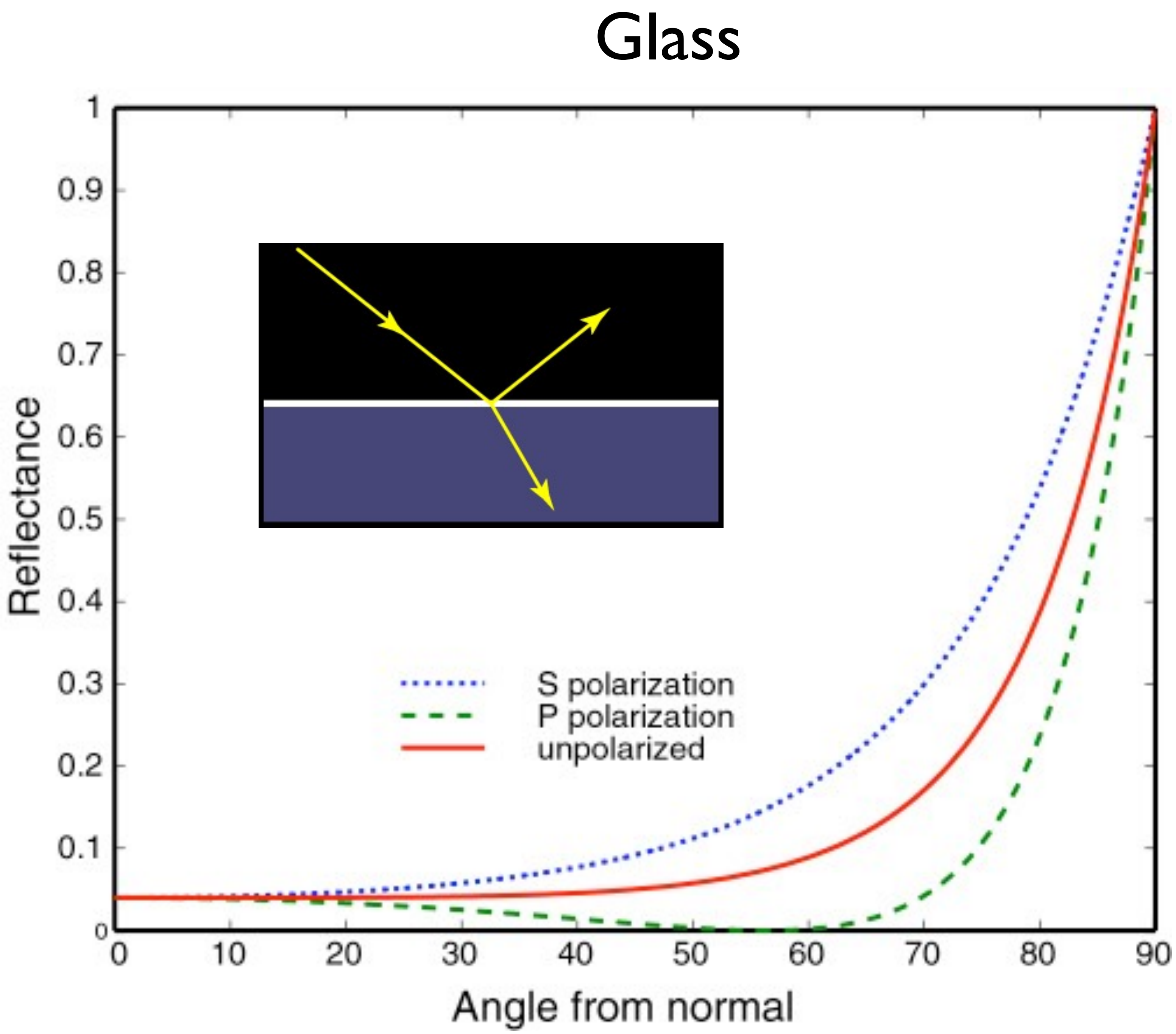
- **Reflectance does depend on angle**
  – but not much
  – safely ignored in basic rendering



© 2018 Steve Marschner

Specular reflection from glass/water

- **Dependence on angle is dramatic!**
  - about 4% at normal incidence
  - always 100% at grazing
  - remaining light is transmitted
- **This is important for proper appearance**



Fresnel's formulas

- **They predict how much light reflects from a smooth interface between two materials**
  – usually one material is empty space

$$F_p = \frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2}$$

$$F_s = \frac{\eta_1 \cos\theta_1 - \eta_2 \cos\theta_2}{\eta_1 \cos\theta_1 + \eta_2 \cos\theta_2}$$

$$R = \frac{1}{2}\left(F_p^2 + F_s^2\right)$$

where

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

note: the formula in the notes and assignment is different but equivalent.

  – R is the fraction that is reflected
  – $(1 - R)$ is the fraction that is transmitted

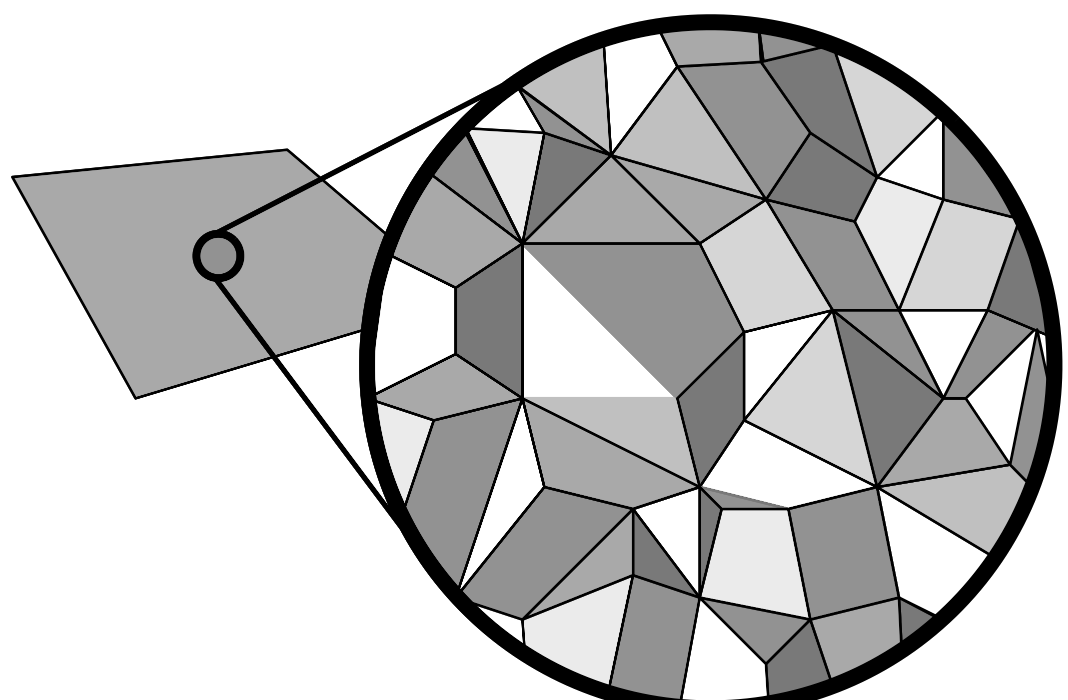
© 2018 Steve Marschner

Fresnel reflection

[Mike Hill & Gaain Kwan | Stanford cs348 competition 2001]

Microfacet BRDF Model

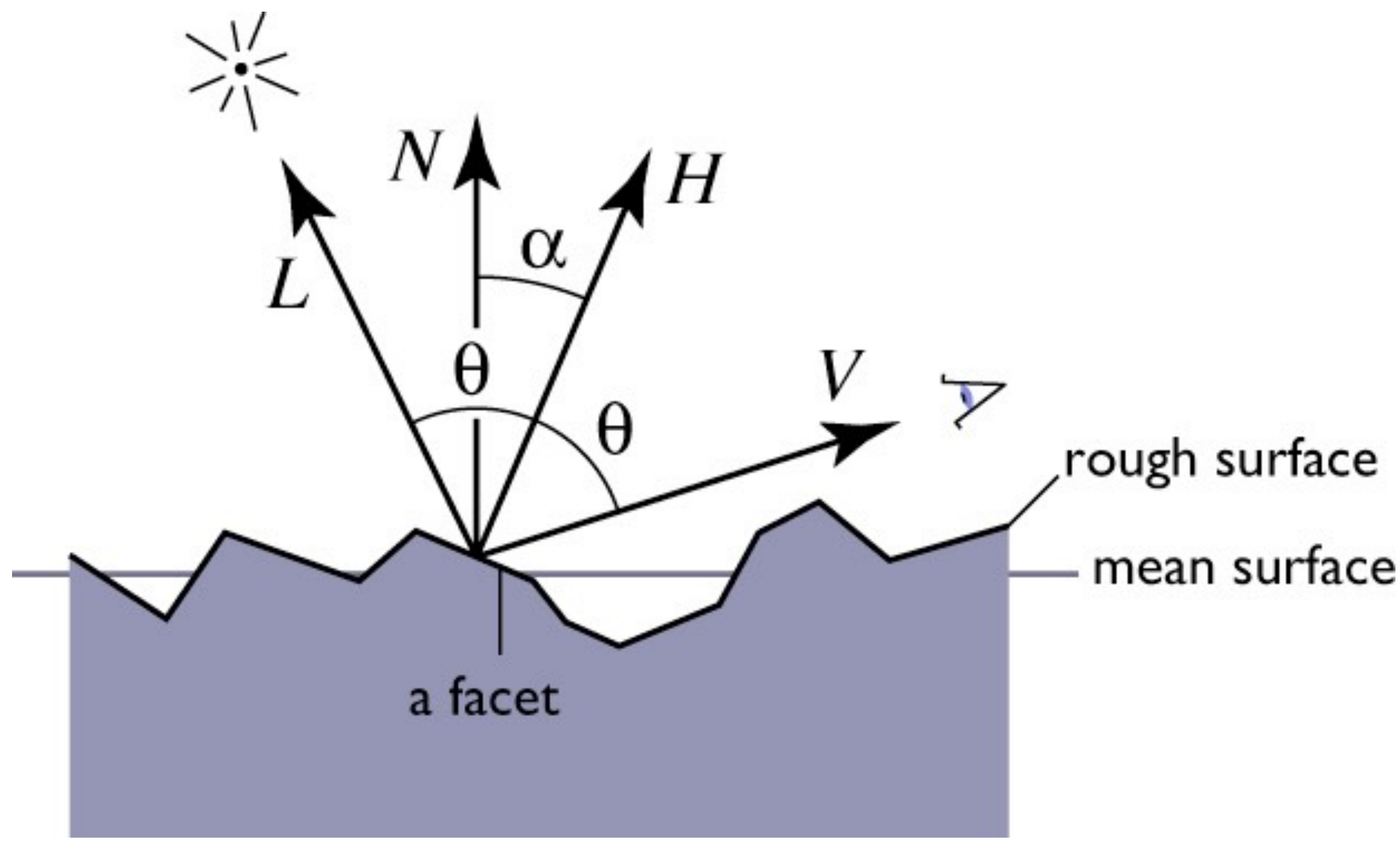
- **The *microfacet* idea**
  - surface modeled as random collection of planar facets
  - an incoming ray hits exactly one facet, at random
- **Key input: probability distribution of facet angle**



[Stephen Westin]

Facet Reflection

- ***H* vector used to define facets that contribute**
  - L and V determine H; only facets with that normal matter
  - reflected light is proportional to number of facets



© 2018 Steve Marschner

Microfacet BRDF Model

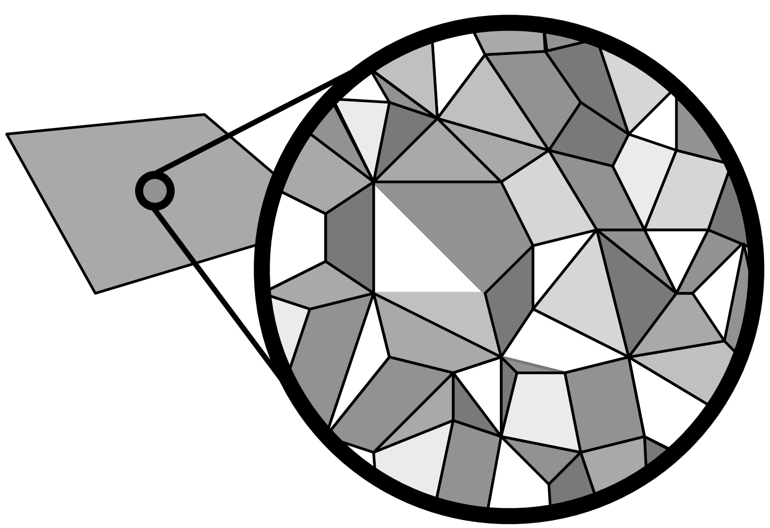
$$f_r(\mathbf{n}, \mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h}) D(\mathbf{h}) G(\mathbf{l}, \mathbf{v}, \mathbf{h})}{4 |\mathbf{n} \cdot \mathbf{l}| |\mathbf{n} \cdot \mathbf{v}|}$$

© 2018 Steve Marschner

Microfacet BRDF Model

Facet distribution

$$f_r(\mathbf{n}, \mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h})D(\mathbf{h})G(\mathbf{l}, \mathbf{v}, \mathbf{h})}{4|\mathbf{n} \cdot \mathbf{l}||\mathbf{n} \cdot \mathbf{v}|}$$



[Stephen Westin]

© 2018 Steve Marschner

Facet Distribution

- **D function describes distribution of h**
- **Many choices, depending on surface characteristics**
- **A classic choice is due to Beckmann**
  - derivation based on Gaussian random processes

$$D(\mathbf{h}) = \frac{e^{-\frac{\tan^2(\mathbf{h},\mathbf{n})}{m^2}}}{\pi m^2 \cos^4(\mathbf{h},\mathbf{n})}$$

© 2018 Steve Marschner

Cook-Torrance BRDF Model

Fresnel Reflectance

$$f_r(\mathbf{n}, \mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h}) D(\mathbf{h}) G(\mathbf{l}, \mathbf{v}, \mathbf{h})}{4|\mathbf{n} \cdot \mathbf{l}||\mathbf{n} \cdot \mathbf{v}|}$$

- **Fresnel reflectance for smooth facet**
  - more light reflected at grazing angles

© 2018 Steve Marschner

Cook-Torrance BRDF Model

Masking/shadowing

$$f_r(\mathbf{n}, \mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h})D(\mathbf{h})G(\mathbf{l}, \mathbf{v}, \mathbf{h})}{4|\mathbf{n} \cdot \mathbf{l}||\mathbf{n} \cdot \mathbf{v}|}$$

© 2018 Steve Marschner

Masking and Shadowing

- **Many options; Smith shadowing-masking follows from *D***
  - long story, it is an integral related to *D* that often doesn't have a closed form solution
  - generally it is assumed that masking and shadowing are statistically independent
  - for Beckmann, recommend using this rational approximation due to Bruce Walter:

$$G(\omega_i, \omega_r, \mathbf{h}) = G_1(\omega_i, \mathbf{h}) G_1(\omega_r, \mathbf{h})$$

$$G_1(\mathbf{v}, \mathbf{h}) \;=\; \chi^+\left(\frac{\mathbf{v}\cdot\mathbf{h}}{\mathbf{v}\cdot\mathbf{n}}\right) \begin{cases} \frac{3.535a + 2.181a^2}{1 + 2.276a + 2.577a^2}, & \text{if } a < 1.6 \\ 1 & \text{otherwise} \end{cases}$$

© 2018 Steve Marschner

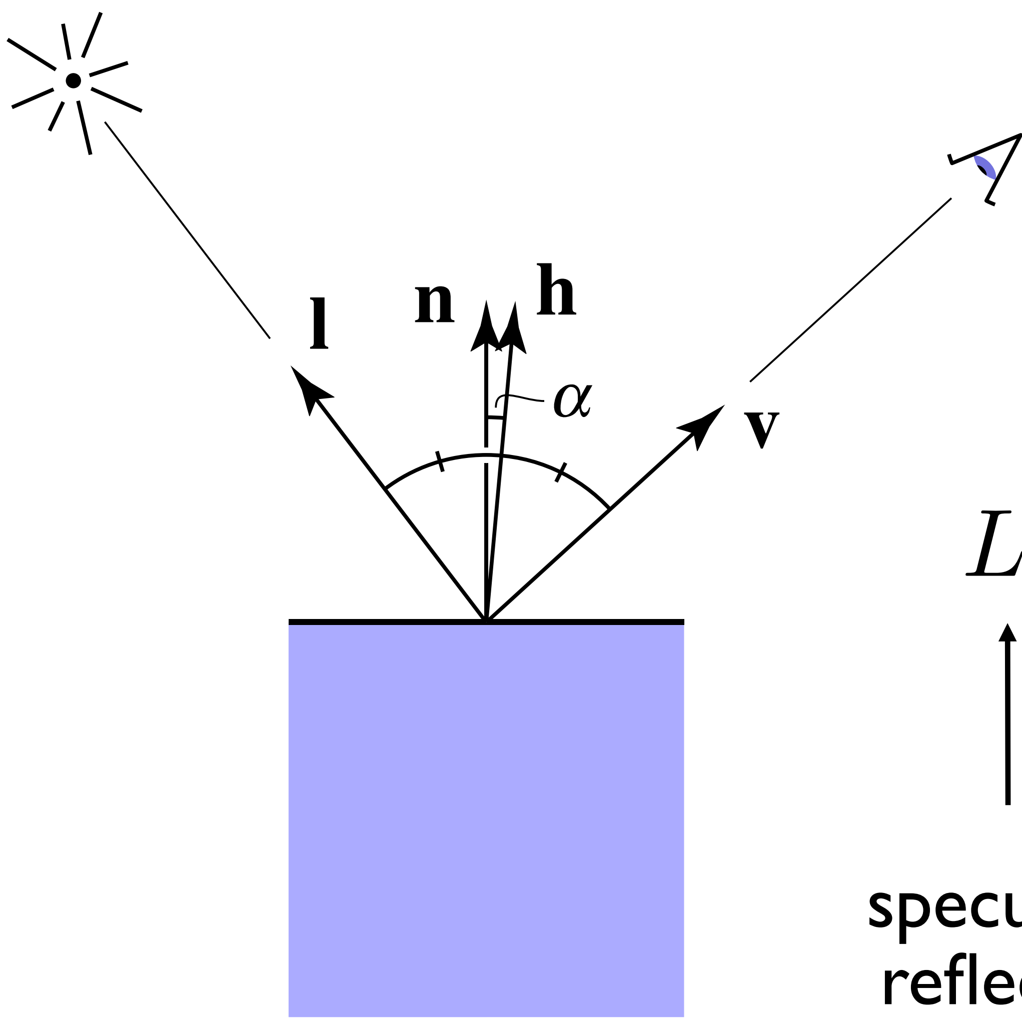
Microfacet BRDF Model

$$f_r(\mathbf{n}, \mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h}) D(\mathbf{h}) G(\mathbf{l}, \mathbf{v}, \mathbf{h})}{4|\mathbf{n} \cdot \mathbf{l}||\mathbf{n} \cdot \mathbf{v}|}$$

– reasons for cosine terms in denominator
– if one is there they clearly both have to be there (by reciprocity)

Specular shading (Microfacet)

- **Close to mirror ⇔ half vector near normal**
  - reflectance is high when **n** · **h** is near 1.0 (α is near 0)



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

irradiance from source

$$L_s = f_r(\mathbf{n}, \mathbf{l}, \mathbf{v})\, \overbrace{\frac{\max(0, \mathbf{n} \cdot \mathbf{l})}{r^2} I}$$

specularly reflected radiance

specular BRDF value

© 2018 Steve Marschner

Ray tracer architecture 101

- **You want a class called Ray**
  - point and direction; evaluate(t)
  - possible: tMin, tMax
- **Some things can be intersected with rays**
  - individual surfaces
  - groups of surfaces (acceleration goes here)
  - the whole scene
  - make these all subclasses of Surface
  - limit the range of valid *t* values (e.g. shadow rays)
- **Once you have the visible intersection, compute the color**
  - may want to separate shading code from geometry
  - separate class: Material (each Surface holds a reference to one)
  - its job is to compute the color

© 2018 Steve Marschner • 44

Architectural practicalities

- **Return values**
  – surface intersection tends to want to return multiple values
    - *t*, surface or shader, normal vector, maybe surface point
  – in many programming languages (e.g. Java) this is a pain
  – typical solution: an *intersection record*
    – a class with fields for all these things
    – keep track of the intersection record for the closest intersection
    – be careful of accidental aliasing (which is very easy if you're new to Java)
- **Efficiency**
  – in Java the (or, a) key to being fast is to minimize creation of objects
  – what objects are created for every ray? try to find a place for them where you can reuse them.
  – Shadow rays can be cheaper (any intersection will do, don't need closest)
  – but: "First Get it Right, Then Make it Fast"

© 2018 Steve Marschner