CS4620/5620: Lecture 34

Ray Tracing, Light Reflection, Advanced Shading

Announcements

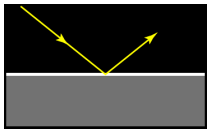
- Review session this evening
  - 7pm Olin 165
- Prelim on Wed
  - In Class
  - Today's class: not on prelim
  - Bring notes?

Shading for Computer Graphics

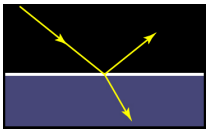
- Need to compute an image
  - of particular geometry
  - under particular illumination
  - from a particular viewpoint
- Basic question: how much light reflects from an object toward the viewer?

Simple materials

metal



dielectric



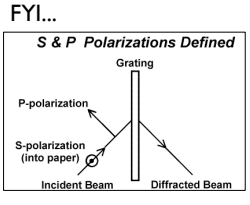
Ray tracing dielectrics

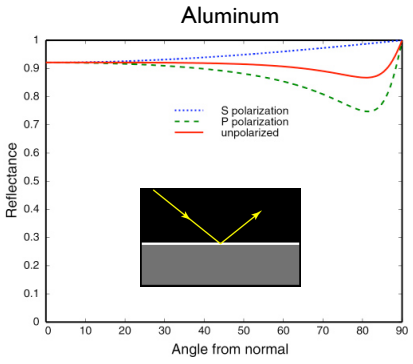
- Like a simple mirror surface, use recursive ray tracing
- But we need two rays
  - One reflects off the surface (same as mirror ray)
  - The other crosses the surface (computed using Snell's law)
    - Doesn't always exist (total internal reflection)
- Splitting into two rays, recursively, creates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

Specular reflection from metal

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

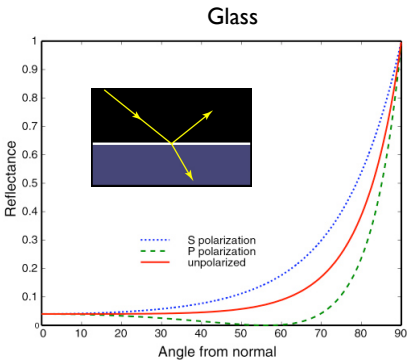
FYI...





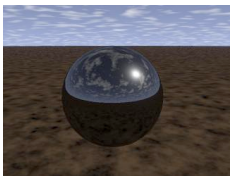
Specular reflection from glass/water

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

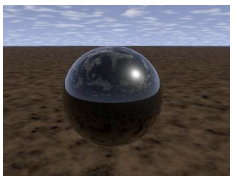


Fresnel reflection

- Black glazed sphere
  - reflection from glass surface
  - transmitted ray is discarded



Constant reflectance



Fresnel reflectance

Fresnel's formulas

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

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

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

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

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

Schlick's approximation

- For graphics, a quick hack to get close with less computation:

$$\tilde{R} = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

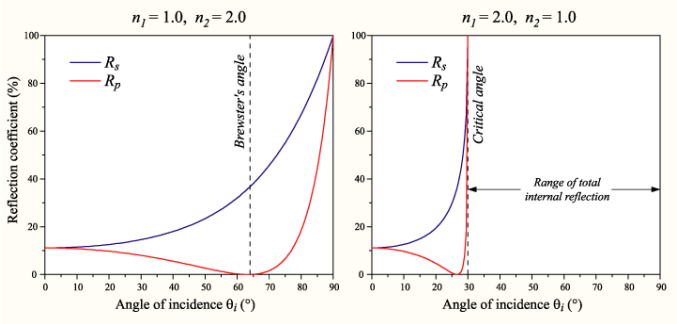
- R_0 is easy to compute:

$$F_p = \frac{\eta_2 - \eta_1}{\eta_2 + \eta_1}$$

$$F_s = \frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}$$

$$R_0 = \left(\frac{\eta_2 - \eta_1}{\eta_2 + \eta_1}\right)^2$$

Fresnel reflection



http://en.wikipedia.org/wiki/File:Fresnel2.png

Fresnel reflection

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

Hmm, ... how do we render beer?

Beer's Law!

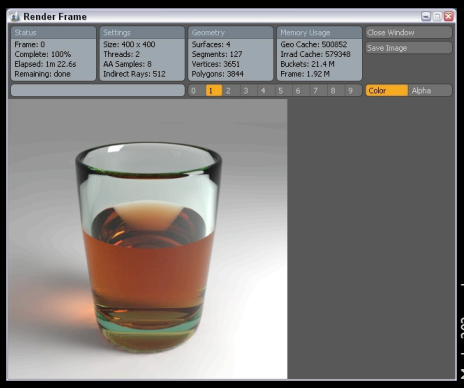
Light-carrying ray intensity I is attenuated with distance traveled x according to $$\frac{dI}{dx} = -bI, b \geq 0$$

Therefore, the attenuated per-channel intensity is $$I(x) = I(0)e^{-bx}, x > 0$$

Evaluated on a per-channel basis, e.g., the color of white light after traveling unit distance is

$$I(1) = (e^{-b_R}, e^{-b_G}, e^{-b_B})$$
$$\xrightarrow{ln} (b_R, b_G, b_B) = -ln(I(1))$$

[Josh Wills | 2003 UCSD Rendering Competition]



Modo 202 render

Shader for Transparent Objects (pp. 306-7)

To add transparent materials to our code, we need a way to determine when a ray is going "into" an object. The simplest way to do this is to assume that all objects are embedded in air with refractive index very close to 1.0, and that surface normals point "out" (toward the air). The code segment for rays and dielectrics with these assumptions is:

if ($\mathbf{p}$ is on a dielectric) **then**
  $\mathbf{r}$ = reflect($\mathbf{d}$, $\mathbf{n}$)
  if ($\mathbf{d} \cdot \mathbf{n} < 0$) **then**
    refract($\mathbf{d}, \mathbf{n}, n, \mathbf{t}$)
    $c = -\mathbf{d} \cdot \mathbf{n}$
    $k_r = k_g = k_b = 1$
  else
    $k_r = \exp(-a_r t)$
    $k_g = \exp(-a_g t)$
    $k_b = \exp(-a_b t)$
    if refract($\mathbf{d}, -\mathbf{n}, 1/n, \mathbf{t}$) **then**
      $c = \mathbf{t} \cdot \mathbf{n}$
    else
      return $k *$ color($\mathbf{p} + t\mathbf{r}$)
  $R_0 = (n-1)^2/(n+1)^2$
  $R = R_0 + (1 - R_0)(1 - c)^5$
  return $k(R$ color($\mathbf{p} + t\mathbf{r}$) + $(1 - R)$ color($\mathbf{p} + t\mathbf{t}$))

Basic ray traced image

[Glassner 89]

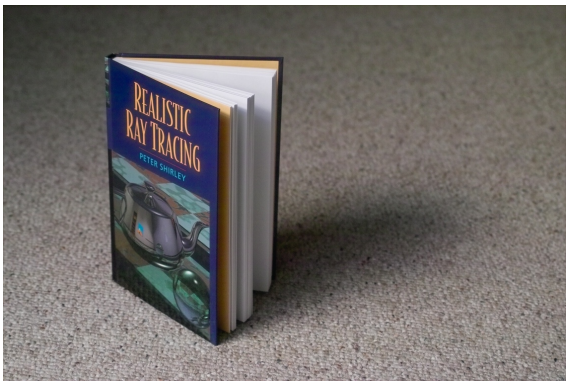
Basic ray tracing

- Basic ray tracer: one sample for everything
  - one ray per pixel
  - one shadow ray for every point light
  - one reflection ray per intersection
    - one refraction ray (if necessary) per intersection
- Many advanced methods build on the basic ray tracing paradigm

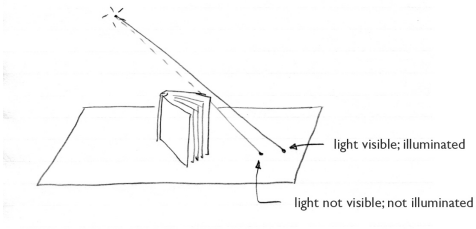
Discontinuities in basic RT

- Perfectly sharp object silhouettes in image
  - leads to aliasing problems (stair steps)
- Perfectly sharp shadow edges
  - everything looks like it's in direct sun
- Perfectly clear mirror reflections
  - reflective surfaces are all highly polished
- Perfect focus at all distances
  - camera always has an infinitely tiny aperture
- Perfectly frozen instant in time (in animation)
  - motion is frozen as if by strobe light

Soft shadows

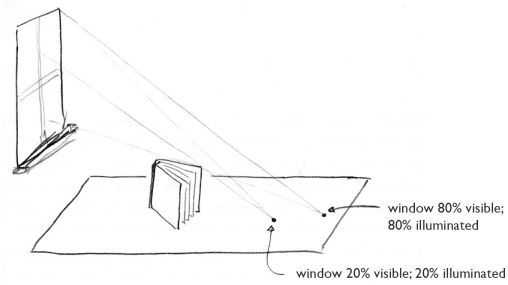


Cause of soft shadows



point lights cast hard shadows

Cause of soft shadows

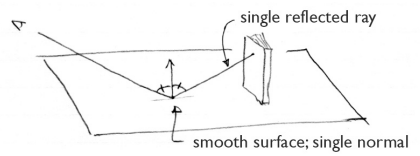


area lights cast soft shadows

Glossy reflection

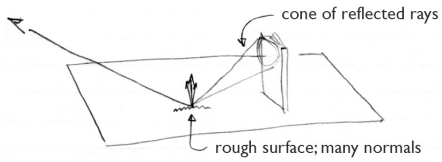
[Lafortune et al. 97]

Cause of glossy reflection



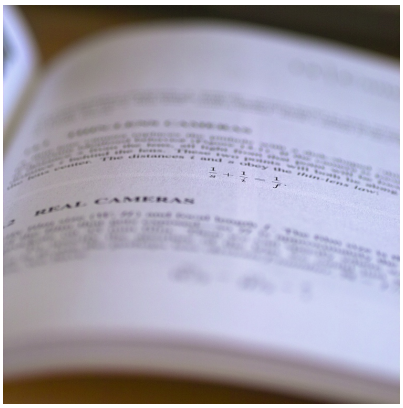
smooth surfaces produce sharp reflections

Cause of glossy reflection

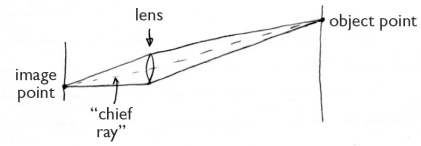


rough surfaces produce soft (glossy) reflections

Depth of field



Cause of focusing effects



what lenses do (roughly)

Cause of focusing effects

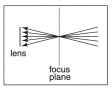
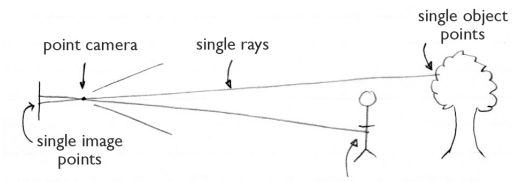
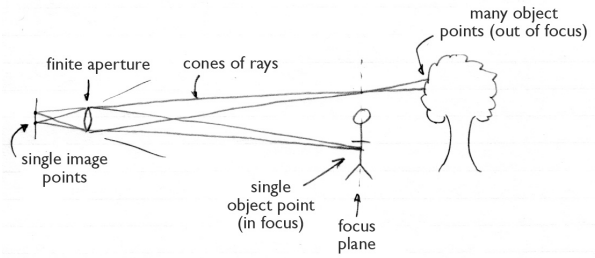


Figure 13.15. The lens averages over a cone of directions that hit the pixel location being sampled.



point aperture produces always-sharp focus

Cause of focusing effects

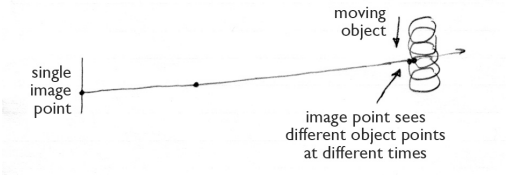


finite aperture produces limited depth of field

Motion blur

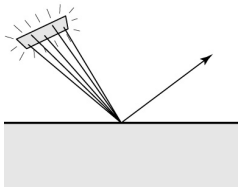
[Cook, Porter, Carpenter 1984]

Cause of motion blur



Creating soft shadows

- For area lights: use many shadow rays
  - and each shadow ray gets a different point on the light
- Choosing samples
  - general principle: start with uniform in square



Creating soft shadows

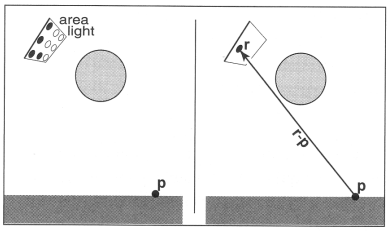
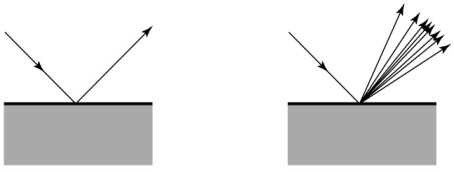


Figure 13.13. Left: an area light can be approximated by some number of point lights; four of the nine points are visible to **p** so it is in the penumbra. Right: a random point on the light is chosen for the shadow ray, and it has some chance of hitting the light or not.

Creating glossy reflections

- Jitter the reflected rays
  - Not exactly in mirror direction; add a random offset
  - Can work out math to match Phong exactly
  - Can do this by jittering the normal if you want



Creating glossy reflections

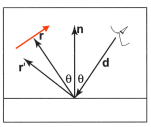


Figure 13.18. The reflection ray is perturbed to a random vector $\mathbf{r}'$.

To choose $\mathbf{r}'$, we again sample a random square. This square is perpendicular to $\mathbf{r}$ and has width a which controls the degree of blur. We can set up the square's orientation by creating an orthonormal basis with $\mathbf{w} = \mathbf{r}$ using the techniques in Section 2.4.6. Then, we create a random point in the 2D square with side length a centered at the origin. If we have 2D sample points $(\xi, \xi') \in [0, 1]^2$, then the analogous point on the desired square is

$$u = -\frac{a}{2} + \xi a,$$

$$v = -\frac{a}{2} + \xi' a.$$

Because the square over which we will perturb is parallel to both the $\mathbf{u}$ and $\mathbf{v}$ vectors, the ray $\mathbf{r}'$ is just

$$\mathbf{r}' = \mathbf{r} + u\mathbf{u} + v\mathbf{v}.$$

Note that $\mathbf{r}'$ is not necessarily a unit vector and should be normalized if your code requires that for ray directions.

Motion blur

- Caused by finite shutter times
- Introduce time as a variable throughout the system
  - object are hit by rays according to their position at a given time
- Then generate rays with times distributed over shutter interval

$$T = T_0 + \xi(T_1 - T_0)$$

Generating samples

- A complicated question in general
- Basic idea: start with random points in a square
- Monte Carlo methods—see 600-level graphics courses