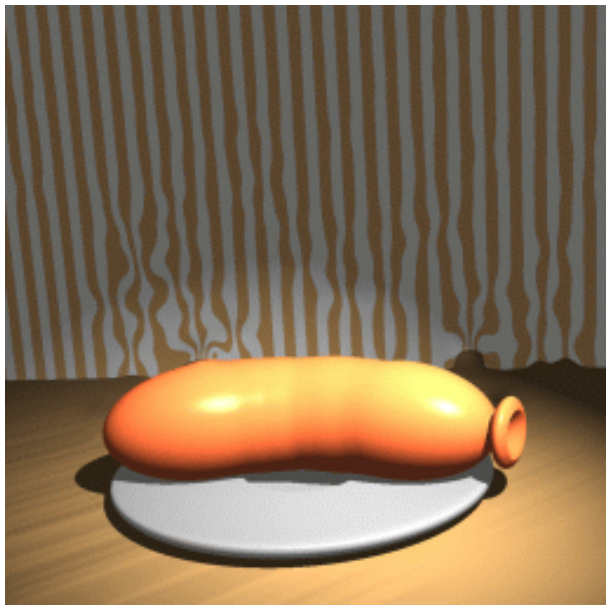
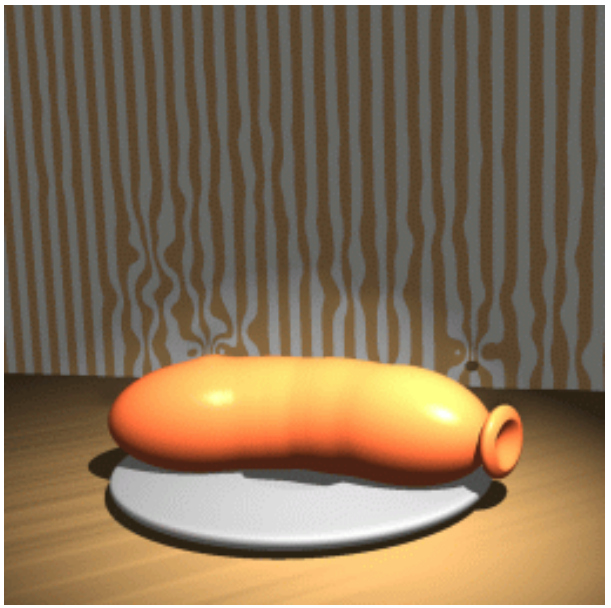
Path Tracing

Models of light

- Geometric optics
  - light travel in straight lines
  - light particles, i.e. photons, do not interact with each other
  - describes: emission, reflection/refraction, absorption

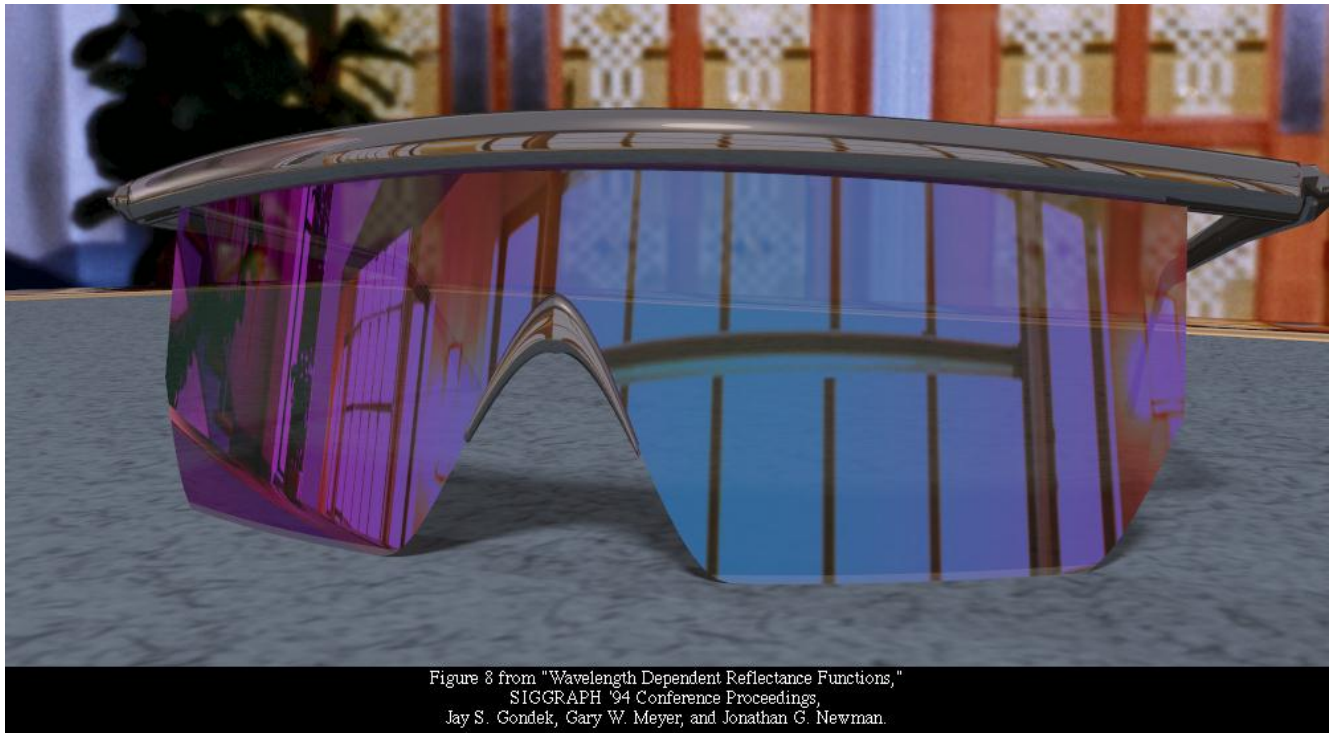




[Stam et al., 1996]

Models of light

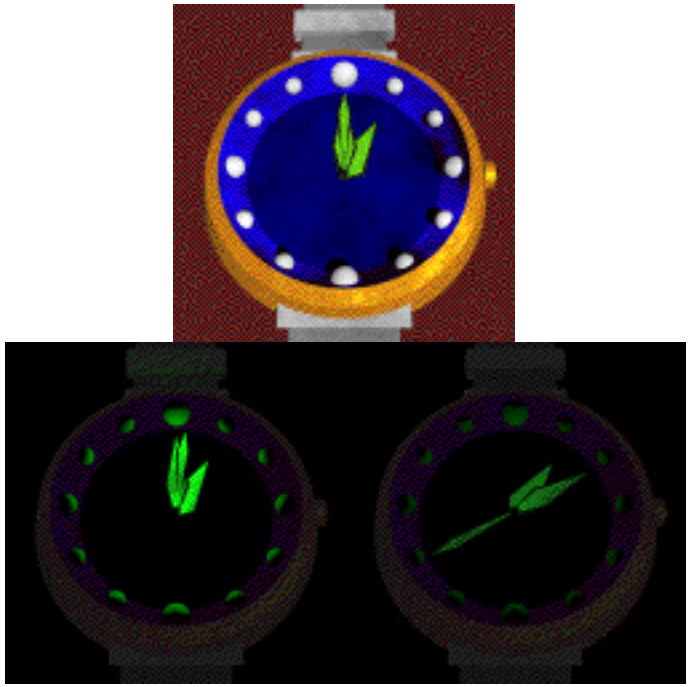
- Wave optics
  - light particles interact with each other
  - describes: diffraction, interference, polarization



[Gondek et al., 1997]

Models of light

- Quantum optics
  - light particles are like any other quantum particle
  - describes: fluorescence, phosphorescence



[Glassener et al., 1997]

Real-world image formation

- Assumption: geometric optics
- Images are formed by a continuous process
  - photons are emitted from light sources
  - they travel in straight lines until the hit a surface
  - at the surfaces, they are either absorbed or reflected
  - the reflected ones continue the same process
  - eventually, some hit the eye
  - the eye measures how many photons hit it

Physically-based rendering

- *All realistic image synthesis methods* simulate, to a different degree of approximation, the physics of lights
  - main idea: simulating the physics of light ensures realistic images
    - obviously we also need realistic input scenes
  - some algorithms simulate "virtual photons", tracing them from the lights to the eye as real photons would behave

- *Rendering equation:* describes formally the physics of light
  - common formulation based on geometric optics
  - describes nearly all illumination effects and surface materials

Physically-based rendering

- *Path tracing:* an algorithm that solves the rendering equation
  - produces *physically-correct* images
  - simulates the physics of light using backwards raytracing, i.e. tracing paths from eye to lights
  - used in architecture, product design and movies
  - drawback: slow to compute
  - many other algorithms exists, but not covered here

Path tracing – informal

[Disney/Youtube]

Path tracing – informal

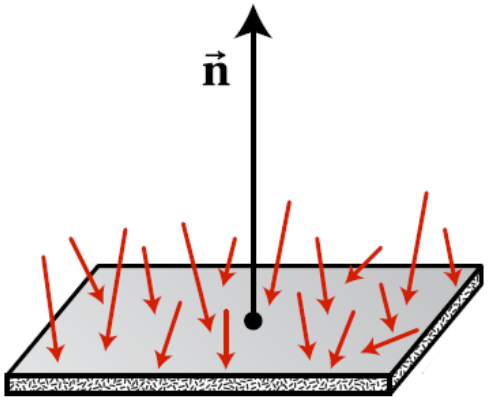
[Disney/Youtube]

Rendering Equation

Radiant power

- Energy *Q* [J]: illumination energy
  - what a detector measures
  - intuition: how many photons hit a detector
- Power *P* or flux [W=Js^{-1}]: energy per unit time
  - intuition: how many photons hit a surface per second
  - other units derived from this

$$P = \frac{dQ}{dt}$$

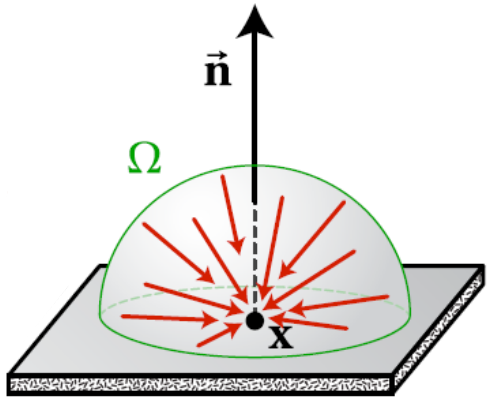


[Woiciech Jaroz]

Irradiance

- Irradiance *E* [Wm^{-2}]: power per unit area
  - measured at each location on a surface
  - intuition: considers photons coming from all direction to a small area around a single surface point
  - irradiance is used for incoming light
  - radiosity is the term used for outgoing light

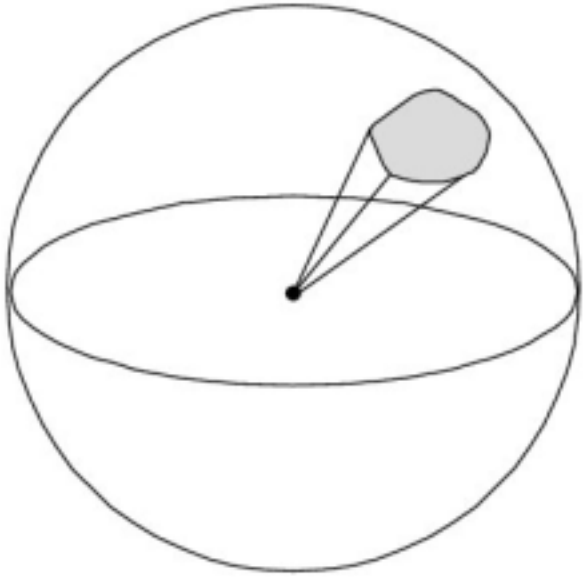
$$E(\mathbf{x}) = \frac{dP}{dA_{\mathbf{x}}}$$



[Woiciech Jaroz]

Solid Angle

- Solid angle [sr]: area of the unit sphere subtended by an object
  - analogous to angles in 2D that are the arc length of the unit circle
  - intuition: measure the "size" of a light beam
- Differential solid angle [sr]: infinitesimal solid angle around a direction
  - for the direction $\mathbf{d}$, indicated as $d\omega_{\mathbf{d}}$



[Jaroslav Krivanec]

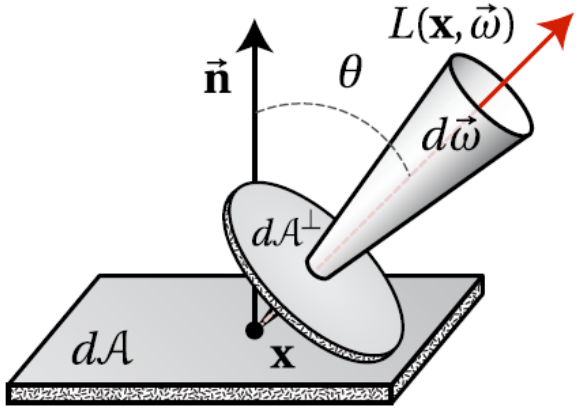
Radiance

- Radiance L [$Wm^{-2}sr^{-2}$]: power per unit projected area per unit solid angle
  - measured at each surface location and for each direction
  - intuition: consider only light in a thin beam around a direction
  - "projected area": account for orientation of surface
  - radiance is defined for incoming, outgoing and reflected light
  - most important quantity: *what our eyes see*
  - HDRs store radiance (scaled)

$$L(\mathbf{x}, \mathbf{i}) = \frac{d^2 P}{dA_{\mathbf{x}} d\omega_{\mathbf{i}} (\mathbf{i} \cdot \mathbf{n})}$$

$$dA_{\mathbf{x}}^{\perp} = dA(\mathbf{i} \cdot \mathbf{n})$$

$$d\omega_{\mathbf{i}}^{\perp} = d\omega_{\mathbf{i}}(\mathbf{i} \cdot \mathbf{n})$$



[Woiciech Jaroz]

Radiance

- Typical values [cd m^{-2}]
  - surface of the sun: 2,000,000,000
  - cloudy sky: 30,000
  - clear day: 3,000
  - overcast day: 300
  - surface of the moon: 0.03

[Pat Hanrahan]

Radiance properties

- Radiance depends on the wavelength
  - we just use RGB vector to represent it that samples the spectrum at three wavelength
  - good enough for most cases

$$L(\mathbf{x}, \mathbf{o}, \lambda) \approx [L_R(\mathbf{x}, \mathbf{o}), L_G(\mathbf{x}, \mathbf{o}), L_B(\mathbf{x}, \mathbf{o})]^T$$

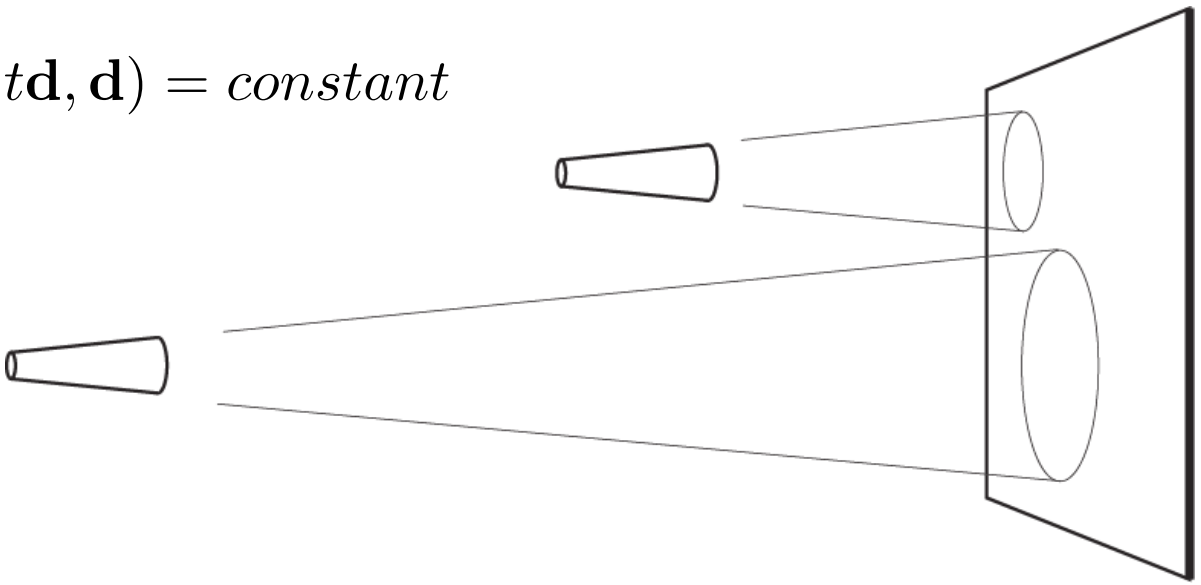
- Incoming radiance L_i is not the same as outgoing radiance L_o
  - photons that reach the surface are not the same as the ones that leave it
  - if not specified, we mean outgoing radiance

$$L_o(\mathbf{x}, \mathbf{d}) = L(\mathbf{x} \rightarrow \mathbf{d}) \neq L(\mathbf{x} \leftarrow \mathbf{d}) = L_i(\mathbf{x}, \mathbf{d})$$

Radiance properties

- Radiance does not change along straight lines in empty space
  - comes from conservation of energy
  - approximately true also in air for short distances
  - reason why radiance is associated with rays in a raytracer
  - corollary: surface colors do not change when we move away

$$L(\mathbf{o} + t\mathbf{d}, \mathbf{d}) = constant$$

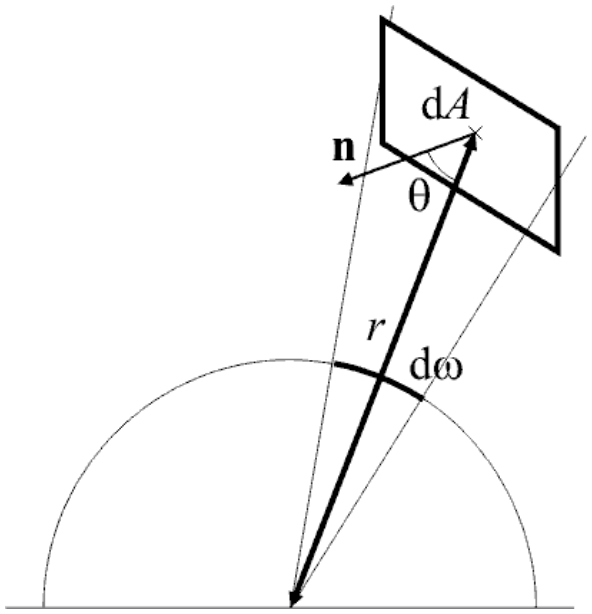


[Shirley]

Differential solid angle of surface

- Differential solid angle of a surface
  - consider a small surface area *dA*
  - project it onto the unit sphere by dividing by distance squared r^2
  - take into account orientation by correcting with the cosine

$$d\omega = \frac{dA \cos\theta}{r^2}$$



[Jaroslav Krivanec]

Proof of radiance invariance

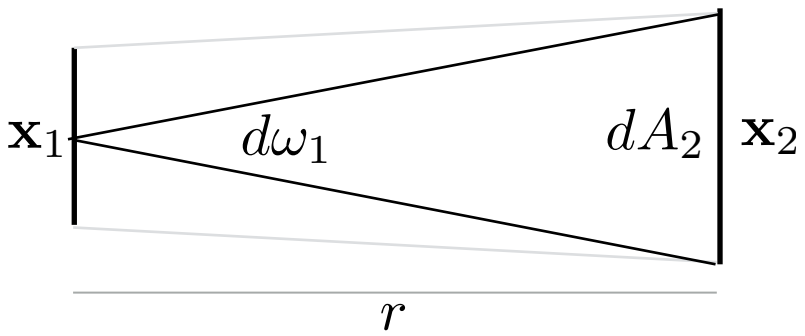
- Consider two surfaces orthogonal to each other
- For energy conservation all photons leaving a surface within the solid angle of the other have to reach the other surface

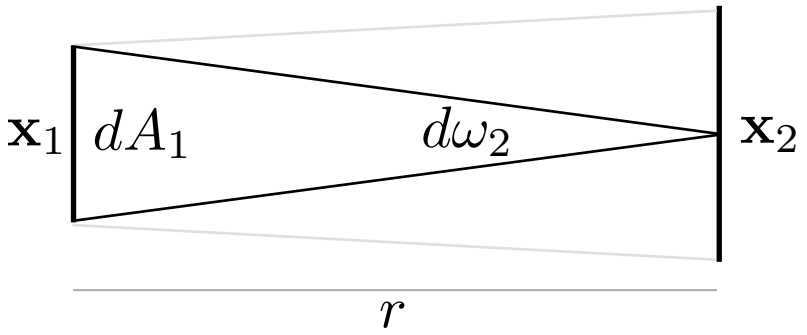
$$d^2P_1 = d^2P_2 \qquad \text{energy conservation}$$

$$L_1 d\omega_1 dA_1 = L_2 d\omega_2 dA_2 \qquad \text{by substitution}$$

$$L_1 (dA_2/r^2) dA_1 = L_2 (dA_1/r^2) dA_2 \qquad \text{by construction}$$

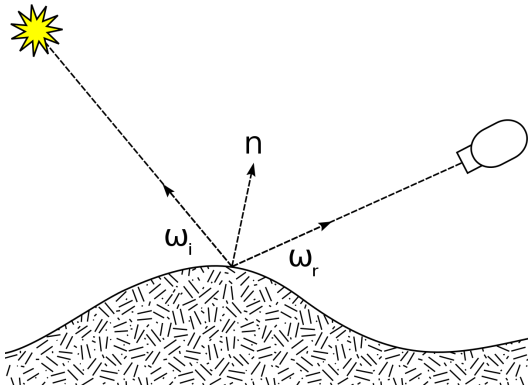
$$L_1 = L_2 = d^2P \frac{dA_1 dA_2}{r^2} \qquad \text{invariance along rays}$$

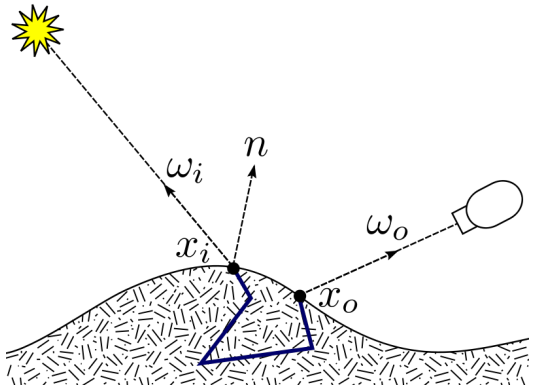




Scattering

- Photons that come to a surface from one direction are scattered in many other directions
- In graphics, we consider mostly three main scattering "modes"
- *Surface scattering* at the surface, e.g. metals and dielectrics
- *Subsurface scattering* in a thin region right below the surface, e.g. skin
- *Volumetric scattering* in a 3D volume, e.g. smoke and fog
- We'll focus on surface scattering

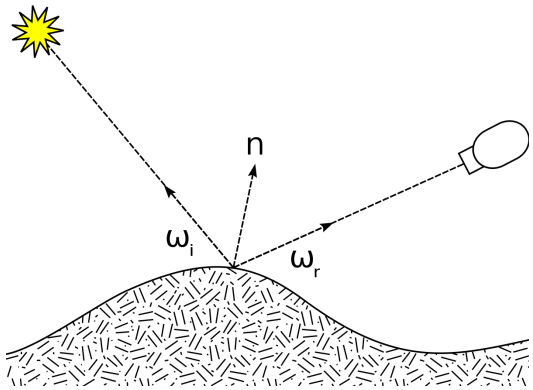




Surface scattering: BRDF

- BRDF [sr^{-1}]: *bidirectional reflectance distribution function*
- Defined as the ratio of the differential reflected radiance over the differential incident irradiance
- Depends on incoming and outgoing directions, and wavelength of light
  - do not track wavelength explicitly: just use colors for BRDF coeff.
- Intuition: probability density that a photon from $d\omega_i$ is reflected in direction $d\omega_o$

$$f(\mathbf{x}, \mathbf{i}, \mathbf{o}) = \frac{dL_r(\mathbf{x}, \mathbf{o})}{dE_i(\mathbf{x}, \mathbf{i})} = \frac{dL_r(\mathbf{x}, \mathbf{o})}{L_i(\mathbf{x}, \mathbf{i})(\mathbf{n_x} \cdot \mathbf{i}) d\omega_\mathbf{i}}$$



Reflected radiance with BRDF

- From the BRDF definition, we write the differential reflected radiance as the product of the BRDF and the incoming radiance weighted by the cosine
  - write the point explicitly

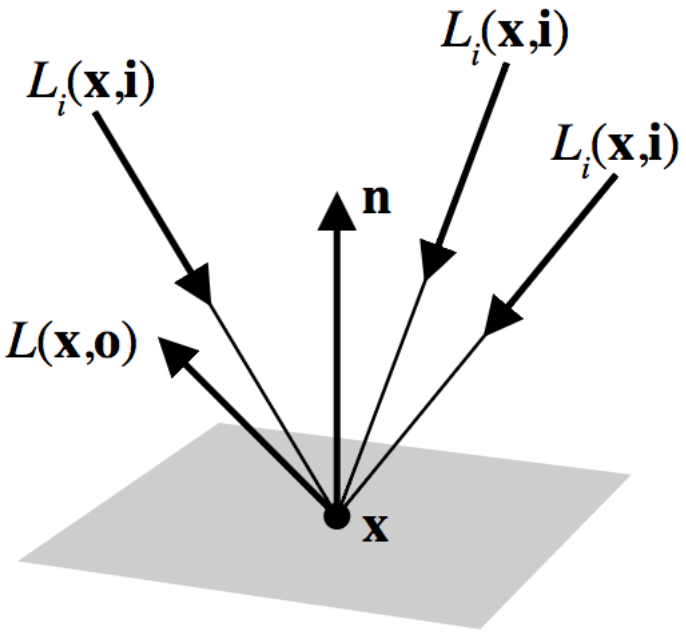
$$dL_r(\mathbf{x}, \mathbf{o}) = L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o})(\mathbf{n_x} \cdot \mathbf{i}) d\omega_\mathbf{i}$$

- The total reflected radiance is the integral of the differential reflected radiance for all directions over the hemisphere

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H_\mathbf{x}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o})(\mathbf{n_x} \cdot \mathbf{i}) d\omega_\mathbf{i}$$

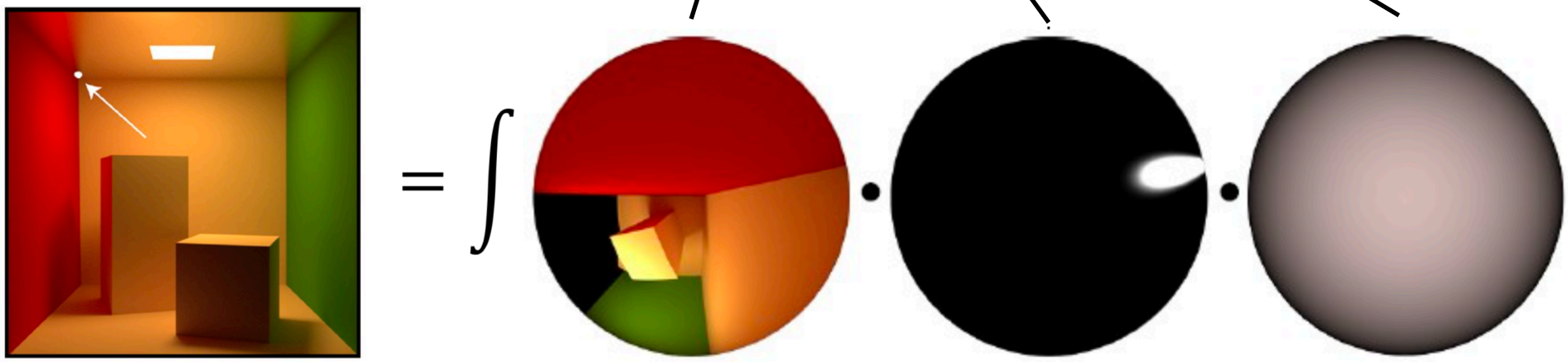
Reflected radiance

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H_\mathbf{x}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o})(\mathbf{n_x} \cdot \mathbf{i}) d\omega_\mathbf{i}$$



Reflected radiance

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H_\mathbf{x}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o})(\mathbf{n_x} \cdot \mathbf{i}) d\omega_\mathbf{i}$$



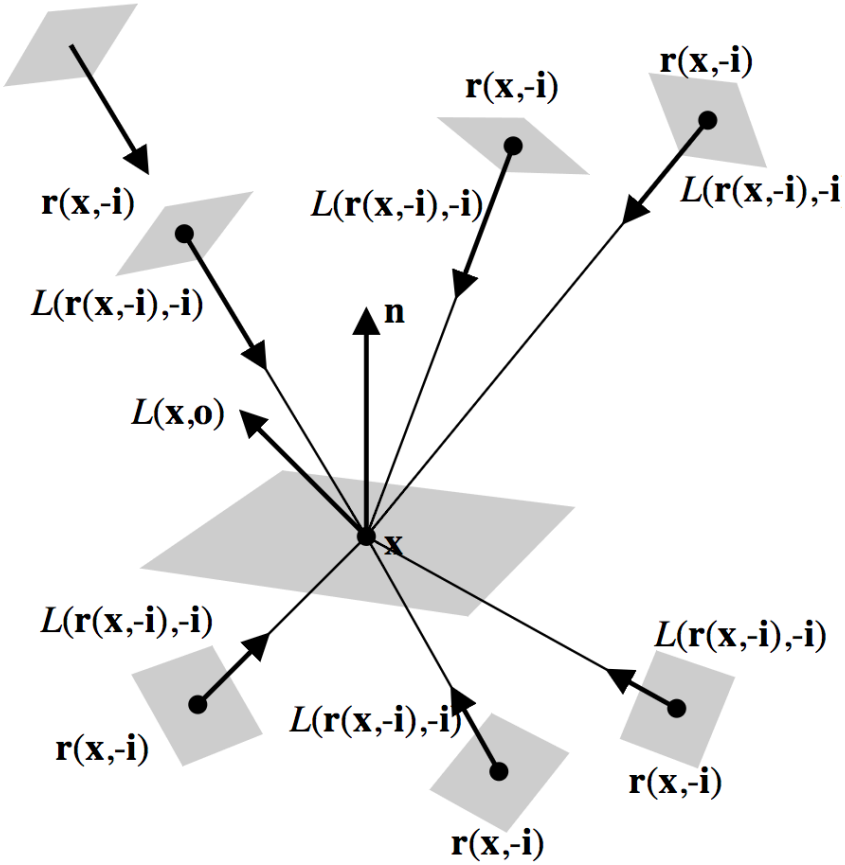
[Krivanec]

BSDF

- So far we only discussed opaque objects
- For translucent objects, such as glass, we have similar definitions
- BTDF: *bidirectional transmittance distribution function*
  - equiv. to BRDF but bottom hemisphere
- BSDF: *bidirectional scattering distribution function*
  - considers both top and bottom hemisphere
  - sum of BRDF and BTDF
- BRDF reflection integral is over full sphere around a point and takes the absolute value of the cosine

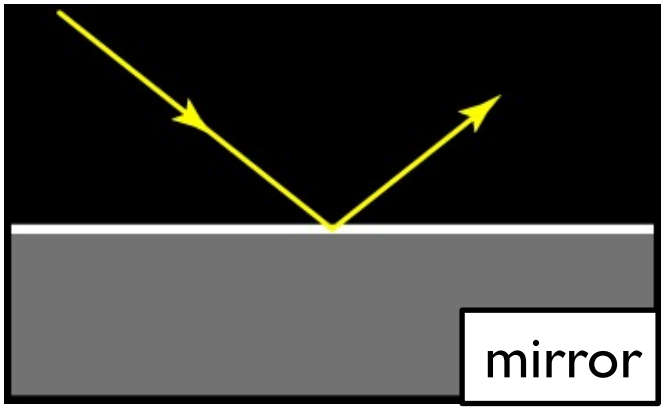
Reflected radiance with BSDF

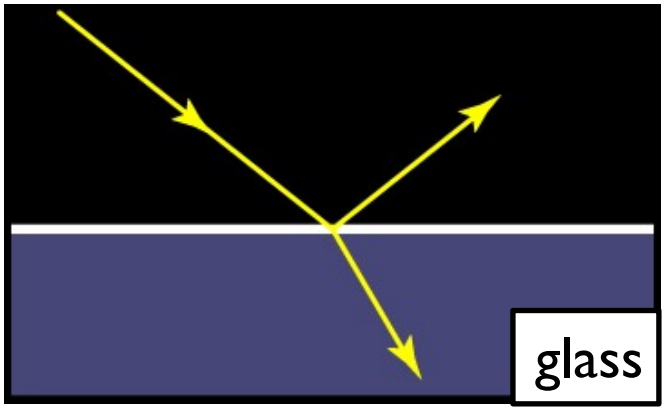
$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H^2_{\mathbf{x}}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

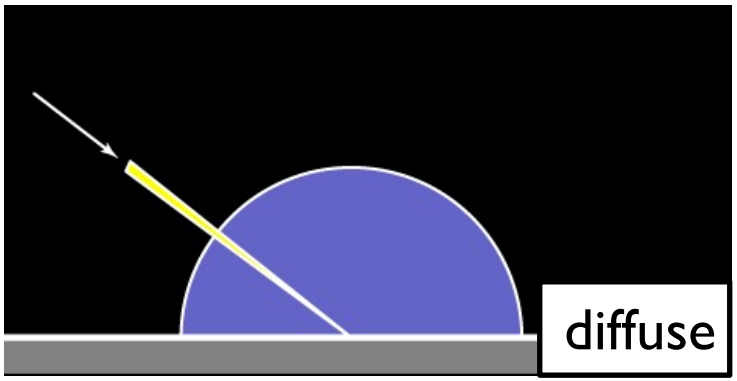


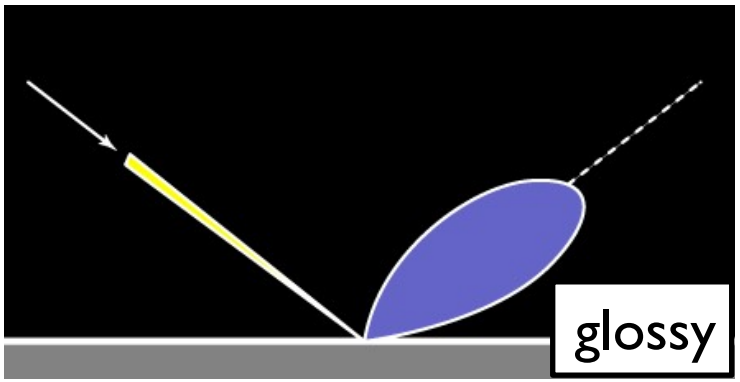
BSDFs for different materials

- Different materials are described by different scattering distributions









Properties of the BSDF

- Positivity: since photons cannot be "negatively" reflected

$$f(\mathbf{x}, \mathbf{i}, \mathbf{o}) \geq 0$$

- Helmotz reciprocity: can invert light paths

$$f(\mathbf{x}, \mathbf{i}, \mathbf{o}) = f(\mathbf{x}, \mathbf{o}, \mathbf{i})$$

- Energy conservation: all photons that comes in are either reflected or absorbed and no new photons is emitted

$$\forall \mathbf{i}. \int_{H^2} f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{o} \cdot \mathbf{n}| d\omega_{\mathbf{o}} \leq 1$$

Rendering equation

- Start from the equation for reflected radiance

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H^2_{\mathbf{x}}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

- Consider also the light that is naturally emitted by some surfaces, such as light source with an additional term

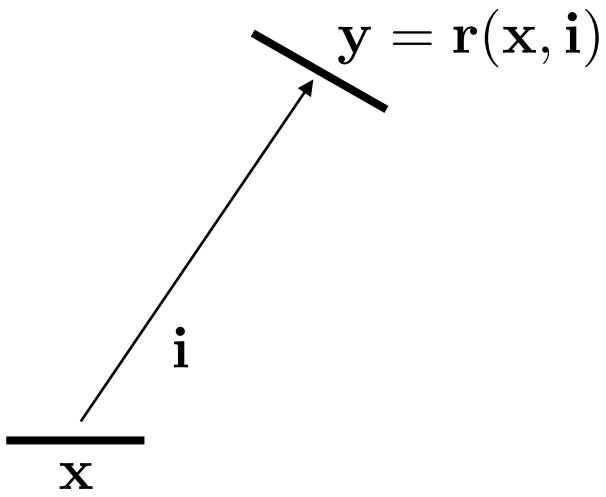
$$L_e(\mathbf{x}, \mathbf{o})$$

- Now we want to express the incoming radiance w.r.t. the outgoing radiance, using the property that radiance along a ray does not change

Radiance along a ray

- From the invariance of radiance along a ray, we can express the radiance incoming to a point as the radiance leaving other points
- Express this with recasting to find the first visible point

$$L_i(\mathbf{x}, \mathbf{i}) = L(\mathbf{r}(\mathbf{x}, \mathbf{i}), -\mathbf{i}))$$



Rendering equation

- Start from the equation for reflected radiance

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H^2_{\mathbf{x}}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

- Write incoming as the outgoing radiance os the first visible surface along the ray

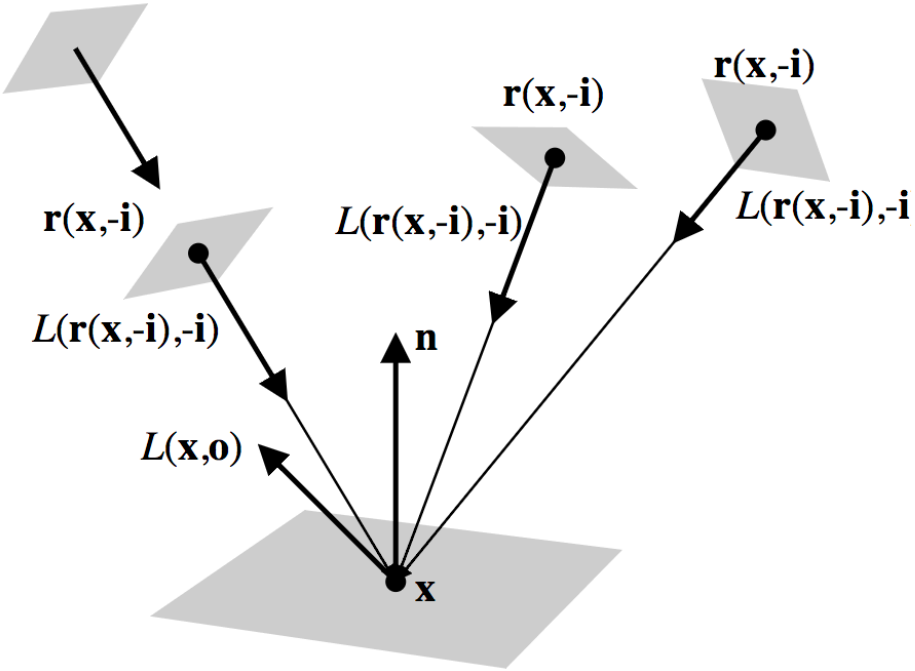
$$L_i(\mathbf{x}, \mathbf{i}) = L(\mathbf{r}(\mathbf{x}, \mathbf{i}), -\mathbf{i})$$

- Substitute the latter in the former and add the emitted radiance

$$L(\mathbf{x}, \mathbf{o}) = L_e(\mathbf{x}, \mathbf{o}) + \int_{H^2_{\mathbf{x}}} L(\mathbf{r}(\mathbf{x}, \mathbf{i}), -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

Rendering equation

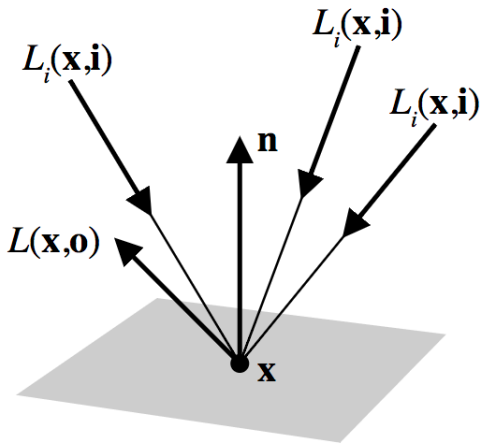
$$L(\mathbf{x},\mathbf{o}) = L_e(\mathbf{x},\mathbf{o}) + \int_{H^2_{\mathbf{x}}} L(\mathbf{r}(\mathbf{x},\mathbf{i}),-\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

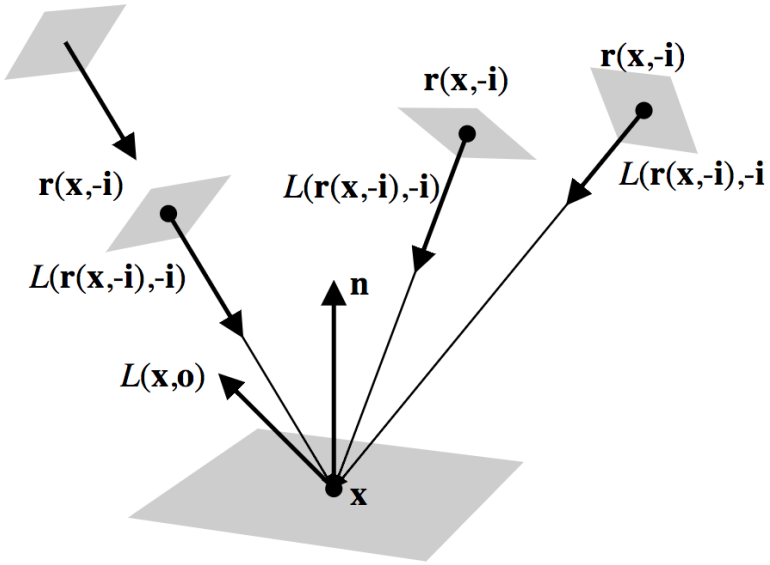


Reflection vs rendering equation

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{H^2_{\mathbf{x}}} L_i(\mathbf{x}, \mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

$$L(\mathbf{x}, \mathbf{o}) = L_e(\mathbf{x}, \mathbf{o}) + \int_{H^2_{\mathbf{x}}} L(\mathbf{r}(\mathbf{x}, \mathbf{i}), -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$



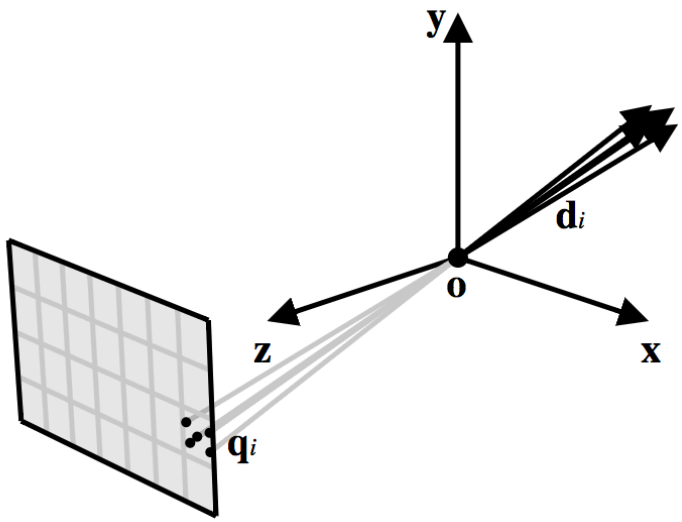


Rendering vs reflection equation

- **Reflection equation**: describes *local light reflection* at a point
  - *Integral* can be used to compute the reflected radiance given the *known incoming radiance*
- **Rendering equation**: condition of the global distribution of light
  - describe the energy balance in the scene at steady state
  - *Integral equation* with *unknown outgoing radiance* on both sides
- **Rendering**: compute the outgoing radiance for all visible points in the images by solving the rendering equation for those points and their view directions

Pixel radiance

- All previous formulation specify the radiance at a point in the scene
- To compute the pixel radiance we can think of measuring the incoming radiance in the sensor area covered by the pixel that can reach the pixel through the lens aperture
- Let us consider first the case of a pinhole camera depicted here



Pixel radiance

- We measure the pixel radiance as the integral, over the pixel surface, of the incoming radiance to each product weighted by the sensitivity of the sensor *s* and the cosine; this gives us proper anit-aliasing
- To be independent of the image resolution we divide by the pixel area

$$L_{pixel} = \frac{1}{A_{pixel}} \int_{\mathbf{q} \in pixel} L_i(\mathbf{q}, \mathbf{d}) s(\mathbf{q}, \mathbf{d}) |\mathbf{d} \cdot \mathbf{n_q}| dA_{\mathbf{q}} \text{ with } \mathbf{d} = \frac{\mathbf{e} - \mathbf{q}}{|\mathbf{e} - \mathbf{q}|}$$

- We now make the approximation that sensor has constant sensitivity and corrects for the cosine (this is your vignetting by the way)

$$L_{pixel} = \frac{1}{A_{pixel}} \int_{\mathbf{q} \in pixel} L_i(\mathbf{q}, \mathbf{d}) dA_{\mathbf{q}} \text{ with } \mathbf{d} = \frac{\mathbf{e} - \mathbf{q}}{|\mathbf{e} - \mathbf{q}|}$$

Monte Carlo integration

Discreet random variable

- Given a random processes with a finite number of outcomes
- A *discreet random variable* is a function that maps outcomes to measurable values, typically real numbers

$$X : \Omega \rightarrow D$$

- Example: rolling a dice
  - outcomes: the face of the dice
  - values of the variable: numbers 1, 2, 3, 4, 5, 6
- For simplicity in these notes, we consider just the range of the random variable and discuss its property over that
  - we assume there is a proper random process whose outcomes can be mapped in the range
- Disclaimer: next slides have math that is correct but not as formal!

Discreet random variable

- A *discreet random variable* is a variable that can randomly take one of a finite number of values with fixed probabilities

$$X \in D = \{v_i\}$$

- *Probability mass function*: probability of each value

$$X \sim p_X \quad p(x) = \Pr(X = x) \quad \sum_i p(v_i) = 1$$

- *Cumulative distribution function*: probability to be within an interval

$$c(x) = \Pr(X \leq x) \quad c(x) = \sum_{v_i \leq x} p(v_i) \quad c(\max(v_i)) = 1$$

- Example: rolling a dice

$$p(v_i) = 1/6 \qquad v_i = \{1, 2, 3, 4, 5, 6\}$$

Expected value and variance

- Expected value: weighted average of the variable values

$$E[X] = \sum_i v_i p(v_i)$$

- Variance: the expected value of the difference from the average

$$\sigma^2[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

- Example: expected value of rolling a dice

$$E[X] = (1 + 2 + 3 + 4 + 5 + 6)/6 = 3.5$$

$$\sigma^2[X] = \ldots \approx 2.92$$

Estimating expected value

- Estimate expected value by running an experiment
  - pick at random one of the variable values with probably *p*
  - lets call this value x_i
  - repeat *n* times and average the results

$$E[X] \approx \frac{1}{N} \sum_i^N x_i$$

- Larger *n* give better estimates
- Example: rolling a dice
  - roll 3 times: $\{3, 1, 6\} \rightarrow E[X] \approx (3 + 1 + 6)/3 = 3.33$
  - roll 9 times: $\{3, 1, 6, 2, 5, 3, 4, 6, 2\} \rightarrow E[X] \approx 3.51$

Law of large numbers

- As the number of triangles goes to infinity, the probability that the average opt the observations is equal to the expected value will be equal to one.

$$\Pr\left(E[X] = \lim_{N\to\infty} \frac{1}{N}\sum\nolimits_i x_i\right) = 1$$

- This means that *the estimate of the expected value converges to the correct value*

Continuous random variable

- *One-dimensional continuous random variables*: similar to discreet random variables but the values are in the continuous domain

$$X \in D = [a, b)$$

- *Probability density function (pdf)*: probability that the variable is in within a differential interval around a given value

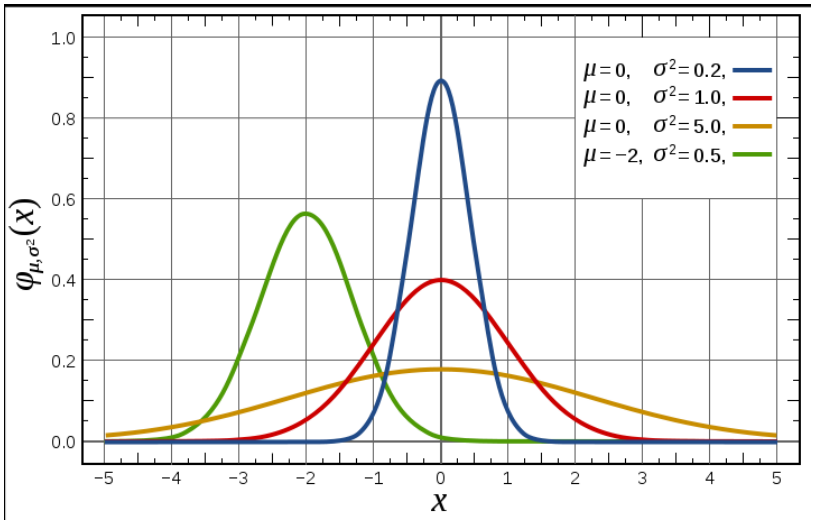
$$X \sim p_X \quad p(x)dx = d\Pr(x \leq X \leq x + dx) \quad \int_a^b p(x)dx = 1$$

- *Cumulative distribution function*: probability to be within an interval

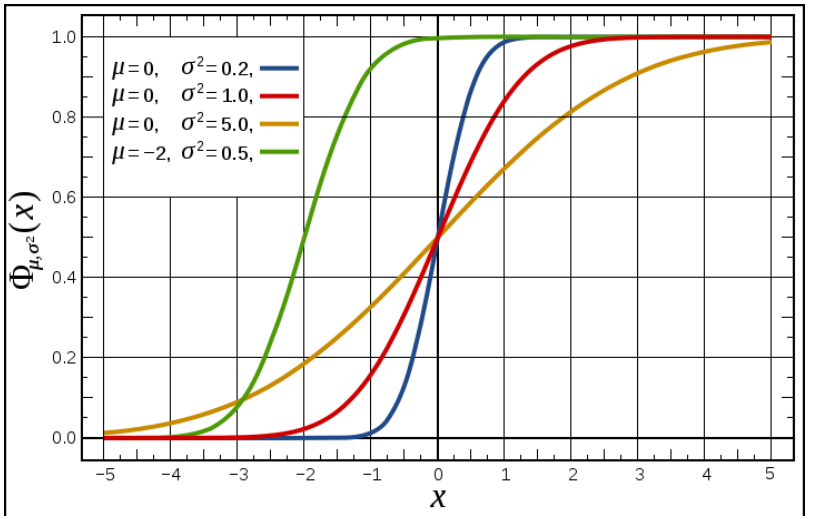
$$c(x) = \Pr(X \leq x) \quad c(x) = \int_a^x p(y)dy \quad c(b) = 1$$

Continuous random variable

example pdf: gaussian *p(x)*



example cdf: erf *c(x)*



Continuous random variable

- More generally, a *continuous random variable* is a multi-dimensional variable that can take a value in a arbitrary continuous domain

$$\mathbf{X} \in D$$

- *Probability density function*: probability that the variable takes a value within a differential domain around a point

$$\mathbf{X} \sim p_{\mathbf{X}} \quad p(\mathbf{x}) dA_{\mathbf{x}} = d\Pr(\mathbf{x} \in dD_{\mathbf{x}}) \quad \int_D p(\mathbf{x}) dA_{\mathbf{x}} = 1$$

- *Cumulative distribution function*: general formulation is complex
- Expected values and variance

$$E[\mathbf{X}] = \int_{\mathbf{x} \in D} \mathbf{x} p(\mathbf{x}) dA_{\mathbf{x}} \quad \sigma[\mathbf{X}] = \int_{\mathbf{x} \in D} |\mathbf{x} - E[\mathbf{X}]|^2 p(\mathbf{x}) dA_{\mathbf{x}}$$

Expected value of a function

- Expected value of a function of a continuous random variable

$$E[g(\mathbf{X})] = \int_{\mathbf{x} \in D} g(\mathbf{x}) p(\mathbf{x}) dA_{\mathbf{x}}$$

- Variance

$$\sigma^2[g(\mathbf{X})] = E[g(\mathbf{X}) - E[g(\mathbf{X})]]^2 = \int_{\mathbf{x} \in D} (g(\mathbf{x}) - E[g(\mathbf{X})])^2 p(\mathbf{x}) dA_{\mathbf{x}}$$

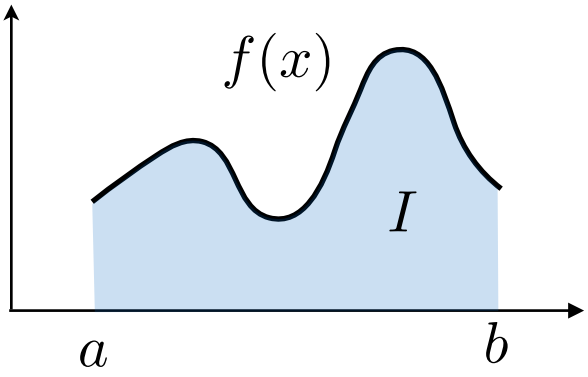
- Estimated expected value

$$E[g(\mathbf{X})] \approx \frac{1}{N} \sum\nolimits_i g(\mathbf{x}_i)$$

Numerical integration

- Rendering amounts to solving numerical integrals
- Let us consider first the case of one dimensional integrals
  - in this case, the integral is the area below the curve

$$I = \int_a^b f(x)dx$$



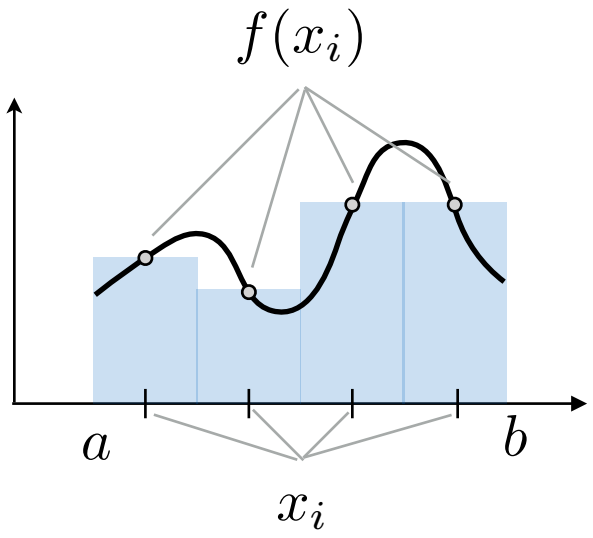
Numerical quadrature

- The classic method for solving numerical integrals is to sample the function at *n* equal spaces positions and approximate the integral with the corresponding boxes

$$I = \int_a^b f(x)dx$$

$$x_i = a + (i - 0.5) \cdot \frac{b-a}{N}$$

$$I \approx \sum_{i=1}^{N} f(x_i) \cdot \frac{b-a}{N}$$



Monte Carlo integration

we want to evaluate $$I = \int_a^b f(x)dx$$

by definition $$E[g(X)] = \int_a^b g(x)p(x)dx$$

can be estimated as $$E[g(X)] \approx \frac{1}{n}\sum_{i=0}^{n-1} g(x_i)$$

by setting $$g(x) = \frac{f(x)}{p(x)} \text{ with uniform } p(x) = \frac{1}{b-a}$$

we have that $$I = \int_a^b \frac{f(x)}{p(x)}p(x)dx$$

can be evaluated as $$I \approx \frac{1}{n}\sum_{i}^{n} \frac{f(x_i)}{p(x_i)}$$

Monte Carlo integration

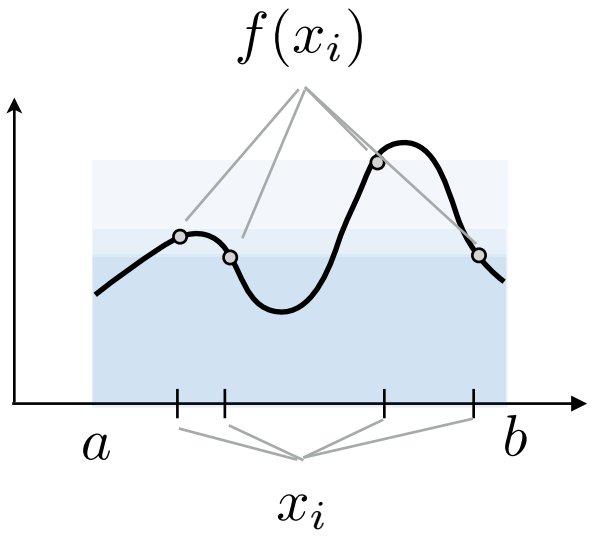
- Approximately evaluate the integral using a *uniform random variable*, by averaging function values at random positions in [a,b)

$$I = \int_a^b f(x)dx$$

$$x \sim p(x) = \frac{1}{b-a}$$

$$I \approx \frac{1}{N}\sum_i \frac{f(x_i)}{p(x_i)} =$$

$$= \frac{1}{N}\sum_i f(x_i)(b-a)$$



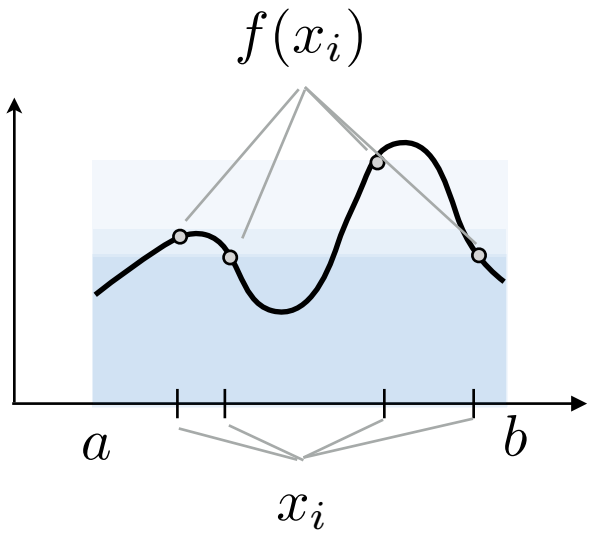
Monte Carlo integration

- Approximately evaluate the integral using a *uniform random variable*, by averaging function values at random positions in [a,b)

$$I = \int_a^b f(x)dx$$

$$x \sim p(x) = \frac{1}{b-a}$$

$$I \approx \frac{1}{N} \sum_i f(x_i)(b-a)$$



Monte Carlo vs. quadrature

- Evaluate the same integral

$$I = \int_a^b f(x)dx$$

- with stochastic vs deterministic positions

$$x \sim p(x) = \frac{1}{b-a} \qquad x_i = a + (i + 0.5) \cdot \frac{b-a}{n}$$

- using averaging vs reconstruction

$$I \approx \frac{1}{N} \sum_i f(x_i)(b-a) \qquad I \approx \sum_{i=1}^{N} f(x_i) \cdot \frac{b-a}{N}$$

Curse of dimensionality

- Evaluate an integral of a function of *k* variables, here in a "box" domain

$$I = \int_{\mathbf{x} \in D} f(\mathbf{x}) dA_{\mathbf{x}} = \int_{a_1}^{b_1} \int_{a_2}^{b_2} ... \int_{a_k}^{b_k} f(x_1, x_2, ..., x_k) dx_1 dx_2 ... dx_k$$

- With numerical quadrature we split each axis separately
- The integral is then approximated by summing the function values of the n^k *k*-dimensional intervals

$$I \approx \sum_{i_1}^{N_1} \sum_{i_2}^{N_2} ... \sum_{i_k}^{N_k} f(x_{1,i_1}, x_{2,i_2}, ..., x_{k,i_k}) w_1 w_2 ... w_k$$

- Error in the estimate goes down as

$$error \propto \frac{1}{\sqrt[k]{N}}$$

Monte Carlo integration

- The estimation of the expected values works in *any dimension*
  - just pick random numbers in the proper high dimensional domain

$$I = \int_{\mathbf{x} \in D} f(\mathbf{x}) dA_{\mathbf{x}} \approx \frac{1}{N} \sum_i \frac{f(\mathbf{x})}{p(\mathbf{x})}$$

- Works for any shape of the domain, provided that you can picks random numbers in it
- The expected relative error of the random estimate depends only on the number of samples, and not the dimensionality of the domain, and goes down with the square root of *n*

$$error \propto \frac{1}{\sqrt{N}}$$

Example: computing *pi*

- *Pi* can be evaluated by computing the integral of a function that is one inside a unit circle and 0 outside

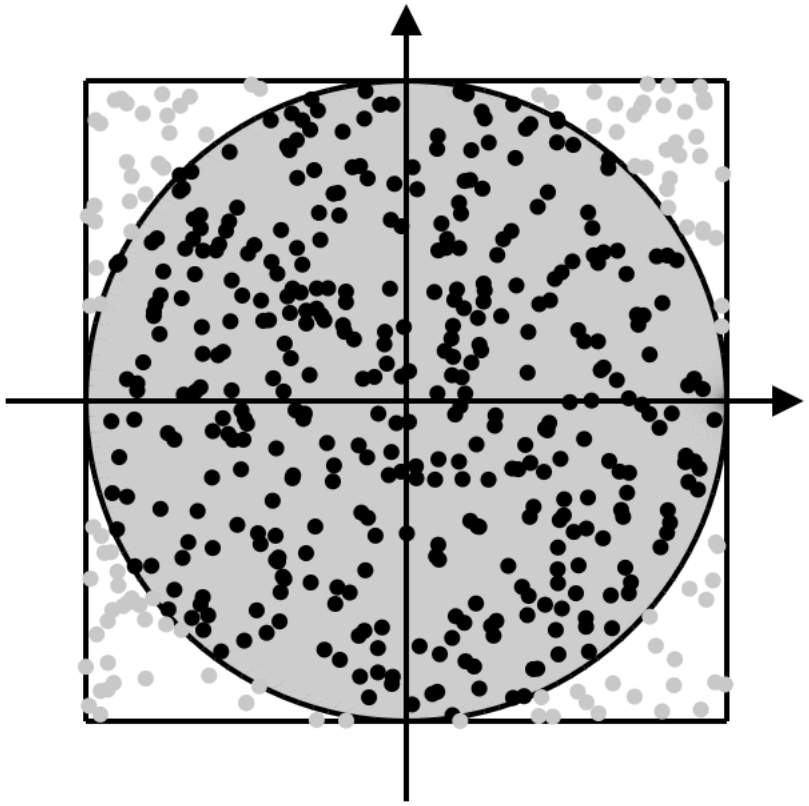
$$\pi = \int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy \text{ with } f(x,y) = \begin{cases} 1 & \text{if } x^2+y^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- With Monte Carlo integration, we can estimate *pi* as

$$\pi \approx \frac{4}{N}\sum_{i}^{N} f(x_i,y_i) \text{ with } (x_i,y_i) \in [-1,1]^2, p(x_i,y_i) = \frac{1}{4}$$

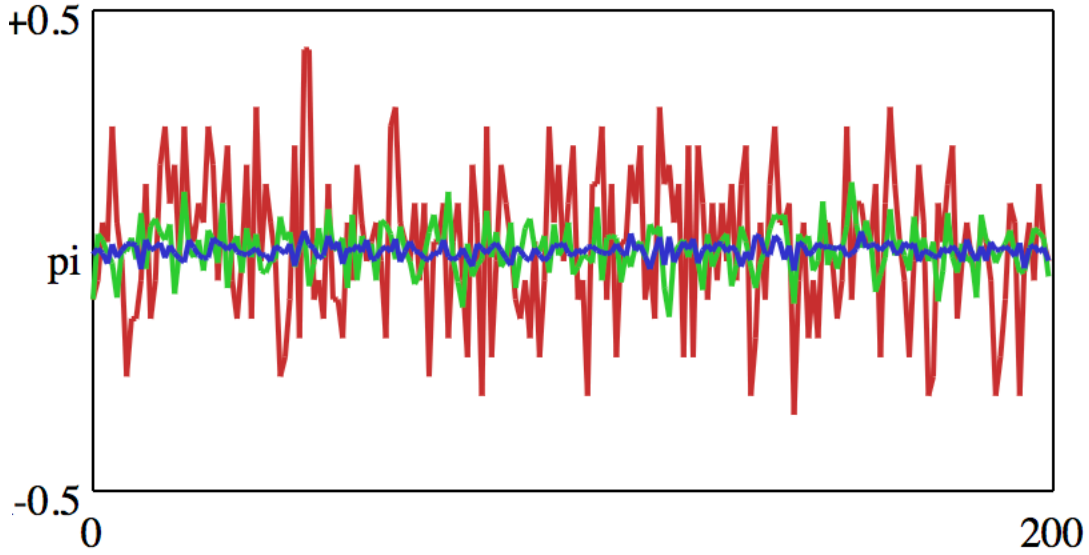
Example: computing *pi*

- If we run one experiment we obtain the following image



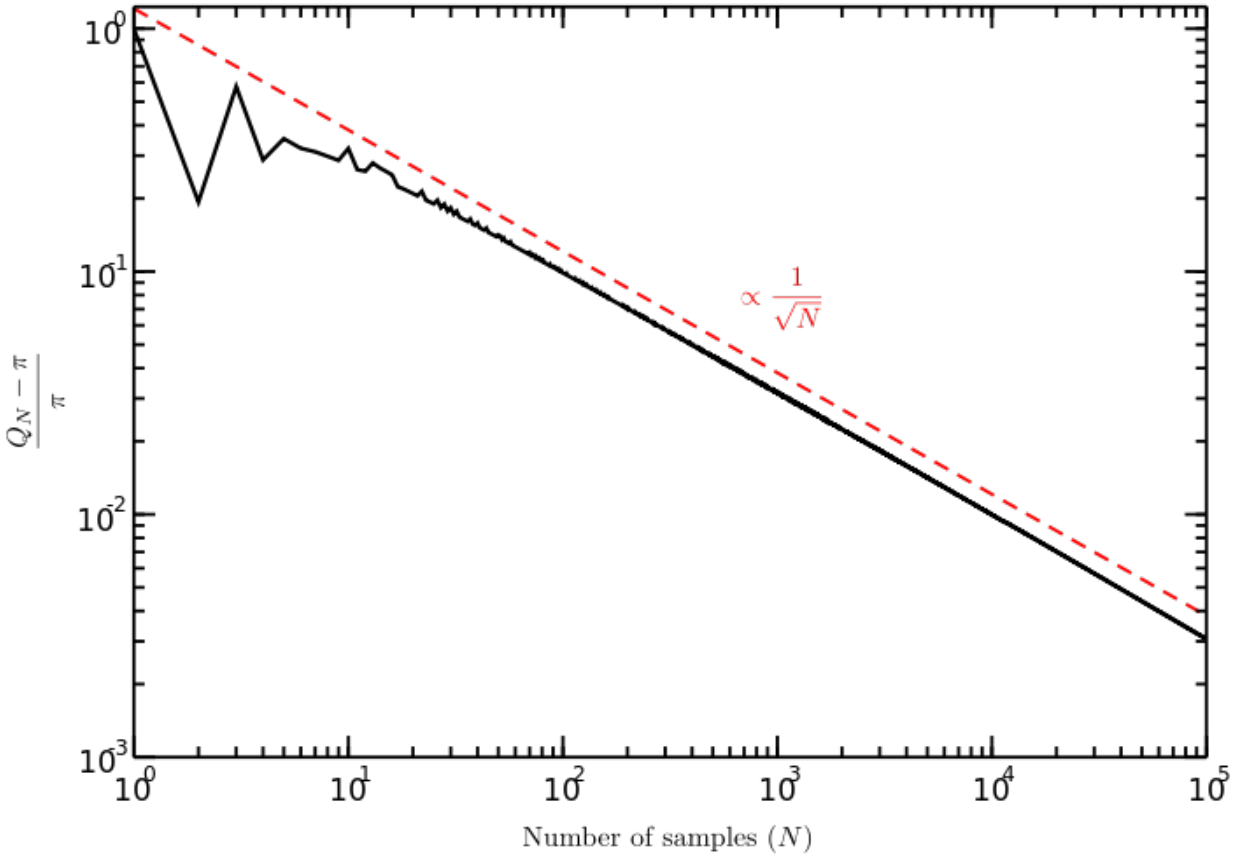
Example: computing *pi*

- Each time we run an experiment we get a slightly different value of *pi*
- The estimates get better as *N* increases



Example: computing *pi*

- Variance decreases with the square root of *N*



[Wikipedia]

Generating random numbers

- Monte Carlo methods requires random numbers
- In general, we use *pseudo random numbers*, i.e. numbers that are deterministically generated, but appear random
  - true random numbers are not always available and are expensive
- Many generators exists with different tradeoffs between random quality and speed/memory
- All generators produce sequences pseudo-random bits, that can be used to generate floating point values in the [0,1) range
- I particularly like the PCG family of random numbers by Melissa E. O'Neill, since they are simple, have good statistical properties, are efficient and they allow for multiple independent streams of random numbers

Implementing the *pi* estimate

- Now we have a way to generate uniform random floats in [0,1)
- But we need uniform values in [-1,1]x[-1,1]
- To generate random numbers in higher dimension, we simply generate multiple random numbers with our generators since the numbers are statistically independent
  - the pdf is the product of the one-dimensional pdfs
- To generate random numbers in a different domain, we *warp* the random numbers to that domain; in out case this is just scaling
  - the pdf needs to be appropriately "rescaled" to take into account the new domain size and the change of variable
- If *r* is our generator of uniform random numbers in [0,1) we can write

$$r \sim p(r) = 1 \rightarrow \mathbf{x} = (2r_1 - 1, 2r_2 - 1) \sim p(\mathbf{x}) = 1/4$$

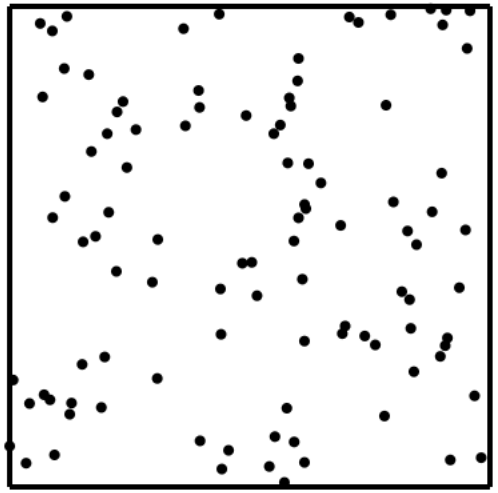
Implementing the *pi* estimate

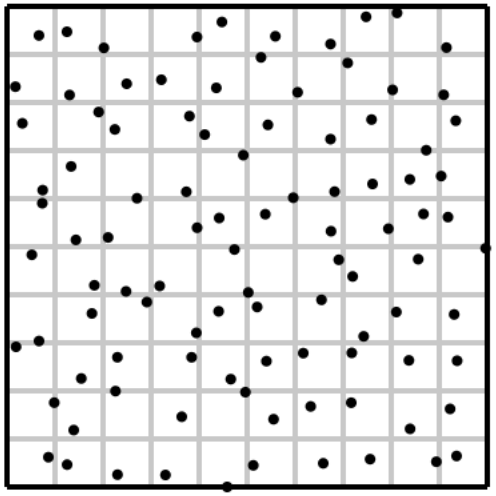
```
float f(vec2f x) {
    return (x.x * x.x + x.y * x.y < 1) ? 1 : 0;
}

float pi(int nsamples, rng_t& rng) {
    auto integral = 0.0f;
    for (auto s = 0; s < nsamples; s++) {
       integral += f({randf(rng)*2-1,randf(rng)*2-1});
    }
    integral *= 4.0f / nsamples;
    return integral;
}
```

Stratified sampling

- Main issue with Monte Carlo is the noise in the solution of integrals
- Several methods exists to reduce noise, which amount to choosing samples at better locations, but still randomly
- *Stratified sampling:* avoid "clumps" in uniform random numbers by forcing them to be distributed in smaller domains
  - not so convenient to implement in rendering

$$\mathbf{x} = \left(\frac{i + r_1}{N_1}, \frac{j + r_2}{N_2} \right)$$





Importance sampling

- *Importance sampling:* sample with a pdf that is as close as possible to the function we want to integrate
  - intuition: the closer the pdf is to the integrand, the smaller in the ratios of f(x)/p(x)
  - most important variance reduction in rendering

$$\mathbf{x} \sim p(\mathbf{x}) \approx f(\mathbf{x})$$

- We will introduce later appropriate warp functions to sample according the advantageous pdfs

Monte Carlo integration

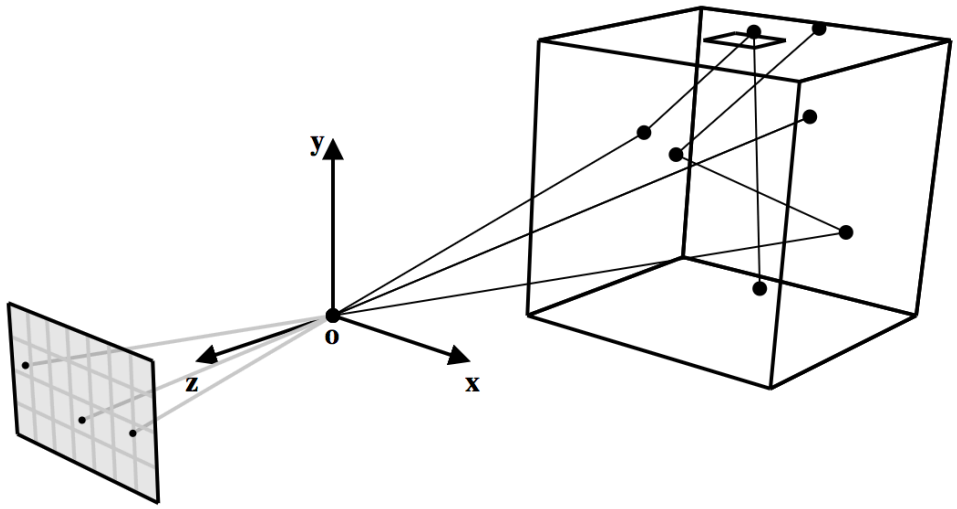
- Pros: works for every domain
- Pros: convergence independent of dimensionality
- Pros: works for functions with discontinuities
- Pros: simple implementation

- Cons: relatively slow convergence
- Cons: variance (noise) in the estimate

Naive Path Tracing

Path tracing

- Solves the rendering equation with Monte Carlo integration supporting all lighting effects shown above
- For each pixel independently, we solve the rendering equation by integrating the incoming radiance over several paths that go through that pixel – paths are traced backwards starting at the pixel and going towards the scene



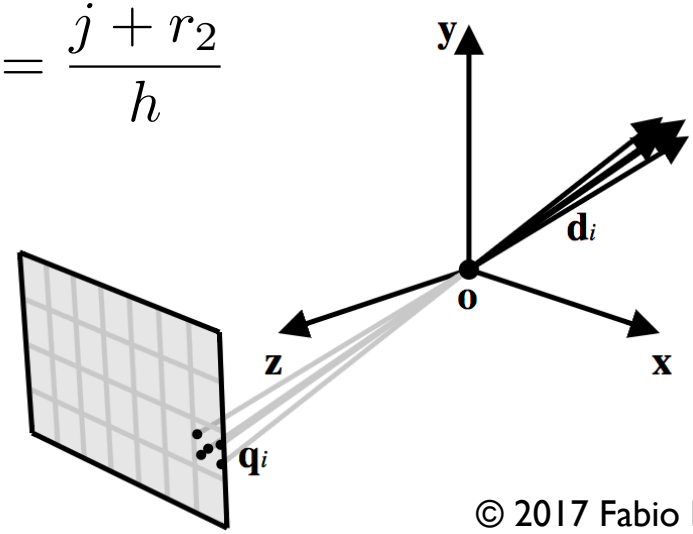
Pixel sampling

- We compute the pixel integral by straightforward Monte Carlo integration of the incoming radiance

$$L_{pixel} = \frac{1}{A_{pixel}} \int_{\mathbf{q} \in pixel} L_i(\mathbf{q}, \overrightarrow{\mathbf{eq}}) dA_{\mathbf{q}} \approx \frac{1}{N} \sum_{i}^{N} L_i(\mathbf{q}_i, \overrightarrow{\mathbf{eq}_i})$$

- To generate the random positions $\mathbf{q}_i$ we pick two random numbers in the unit interval and warp them to the camera *(u,v)* coordinates

$$u = \frac{i + r_1}{w} \qquad v = \frac{j + r_2}{h}$$



Estimating incoming radiance

- We estimate the incoming radiance by tracing a ray toward the scene
- If the ray hits, we return an estimate of the outgoing radiance computed at the intersection point and with the intersection BRDF
- If not, we return the background radiance at that direction

$$L_i(\mathbf{q}, \mathbf{d}) = \begin{cases} L(\mathbf{r}(\mathbf{q}, \mathbf{d}), -\mathbf{d}) & \text{if ray hits} \\ L_{environment}(\mathbf{d}) & \text{otherwise} \end{cases}$$

Solving the reflection equation

- We first give an "intuitive" solution to the rendering equation that we will then formally prove later
- Consider the rendering equation

$$L(\mathbf{x}, \mathbf{o}) = L_e(\mathbf{x}, \mathbf{o}) + \int_{H^2_{\mathbf{x}}} L(\mathbf{r}(\mathbf{x}, \mathbf{i}), -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

- We can solve this equation by picking random direction in the hemisphere above the point – we use the hemisphere to handle opaque objects
- A Monte Carlo estimate for integral can be written as

$$L(\mathbf{x}, \mathbf{o}) \approx L_e(\mathbf{x}, \mathbf{o}) + \frac{L(\mathbf{r}(\mathbf{x}, \mathbf{i}_i), -\mathbf{i}_i) f(\mathbf{x}, \mathbf{i}_i, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}_i|}{p(\mathbf{i}_i)}$$

Hemispherical sampling

- To generate random directions in the hemisphere, we warp two uniform random numbers to an hemispherical direction
- To lower variance, we choose a distribution proportional to the cosine
- In local coordinates, we can write

$$\mathbf{i}^l = [\sqrt{1-r_2}\cos(2\pi r_1), \sqrt{1-r_2}\sin(2\pi r_1), r_2]$$

$$p(\mathbf{i}^l) = \frac{\mathbf{i}^l_z}{\pi}$$

- To get the world coordinates, we construct a frame with z aligned to the normal and transform by it

$$\mathbf{i} = \sqrt{1-r_2}\cos(2\pi r_1)\cdot\mathbf{t_x} + \sqrt{1-r_2}\sin(2\pi r_1)\cdot\mathbf{t_y} + r_2\cdot\mathbf{n}$$

$$p(\mathbf{i}) = \frac{\mathbf{n}\cdot\mathbf{i}}{\pi}$$

Recursive evaluation

- The equation above cannot be directly evaluated since the unknown radiance is present on the right side
- To solve the equation, we recursively evaluate the integral using the same formulation
- The recursion terminates is we do not hit a surface, in which case we accumulate the radiance of the environment map
- But in most cases, say closed rooms, an infinite recursion will happen
  - for now, we stop evaluation after a fixed number of recursive calls
- We start the recursion at each camera pixel

Path tracing – main loop

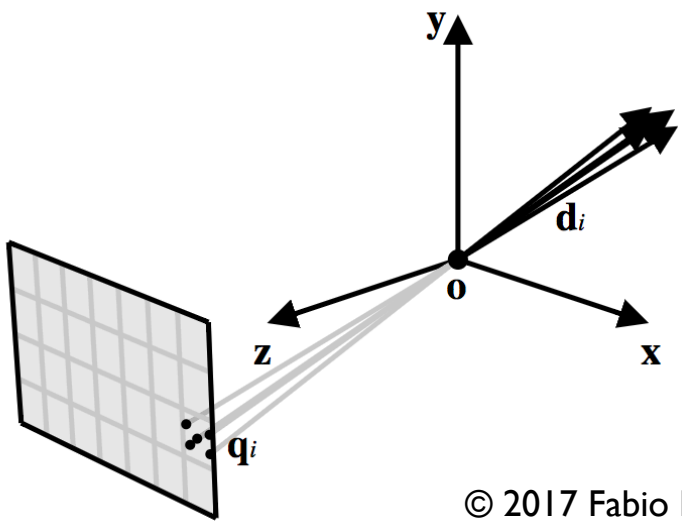
- We can now sketch the pseudocode for a naive path tracer starting from the outer loop

```
void pathtrace(scene* scn, image4f& img) {
    auto rng = init_random();
    image.set(0);
    for(auto j : range(img.height))
        for(auto i : range(img.width))
            for(auto s : range(nsamples)) {
                auto uv = {(i+randf())/img.width,
                           (j+randf())/img.height};
                auto [q, i] = sample_camera(scn->cam, uv);
                auto li = estimate_li(scn, q, i);
                image[i,j] += li / nsamples;
            }
}
```


Path tracing – pixel sampling

- We then sample the evaluate the camera accordingly

```
ray3f sample_camera(camera* cam, vec2f uv) {
    auto ql ={(uv.x-0.5) * cam->w, (uv.y-0.5) * cam->h, -1};
    auto ol = vec3f{0, 0, 0};
    return {transform_point(cam->frame, ol),
        transform_direction(cam->frame, normalize(-ql - ol))};
}
```



Path tracing – recursive

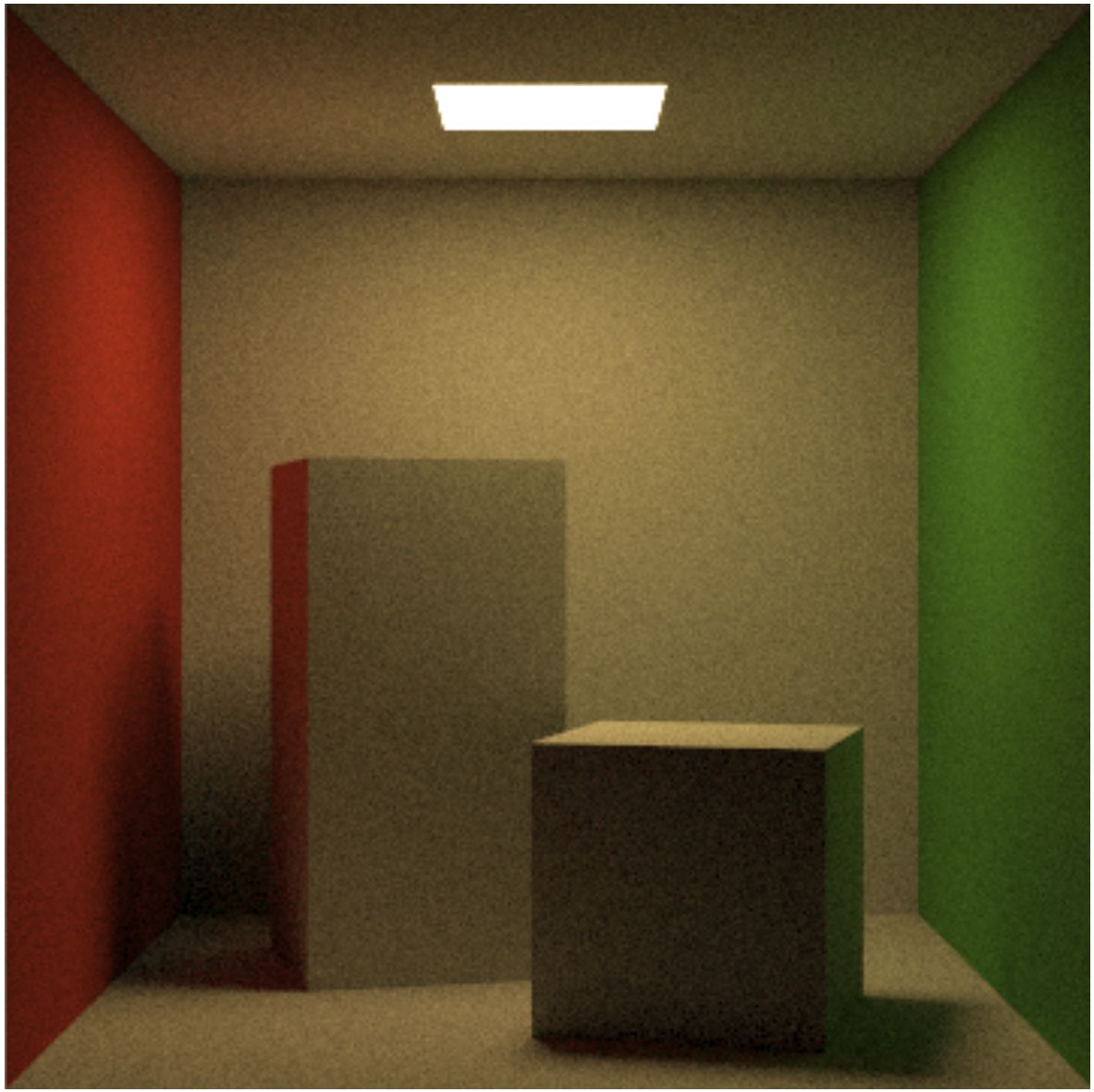
- We can now write the path tracer with the previous equation

```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto isec = intersect(scn, {q, i});
    if(!isec) return eval_env(scn, i); // not hit

    auto x, n = eval_point(isec);     // eval pos, norm
    auto f = eval_brdf(isec);         // evaluate textures here
    auto o = -i;

    auto le = eval_emission(brdf, o);
    if(bounce >= max_bounce) return le; // exit

    auto i = sample_hemisphere(n);
    auto li = estimate_li(scn, o, i); // recursive
    auto lr = li * eval_brdf(f,i,o) * dot(i,n) / p(i);
    return le + lr;
}
```

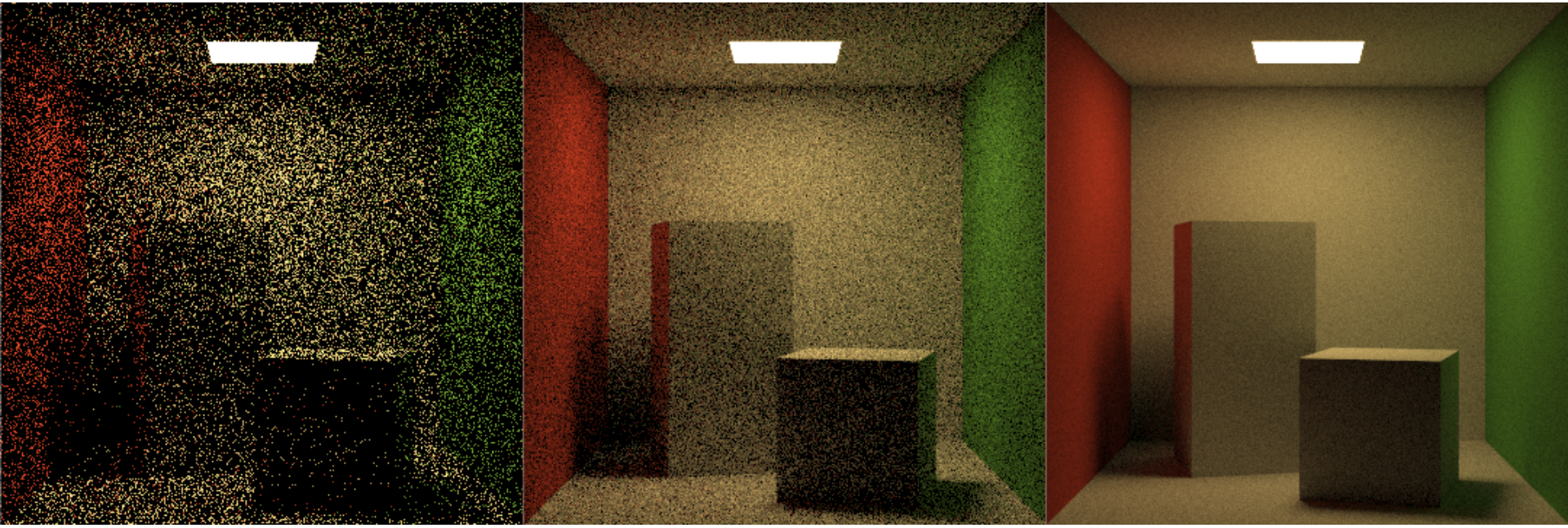
Path tracing – point formulation

- For convenience, define an intersection "point" structure
  - if no intersection, set the BSDF to zero but store *Le*
- For convenience, fold the dot product into BSDF
- Finally, pick sample according to the BSDF (as shown later)

```
struct point { vec3f x, n, le, o; bsdf f; };

vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    if(!pt.f) return pt.le;

    if(bounce >= max_bounce) return pt.le;

    auto i = sample_brdfcos(pt);
    auto li = estimate_li(scn, {pt.x,i});
    auto lr = li * eval_brdfcos(pt,i) / pdf(i);
    return le + lr;
}
```



Naive path tracing – convergence

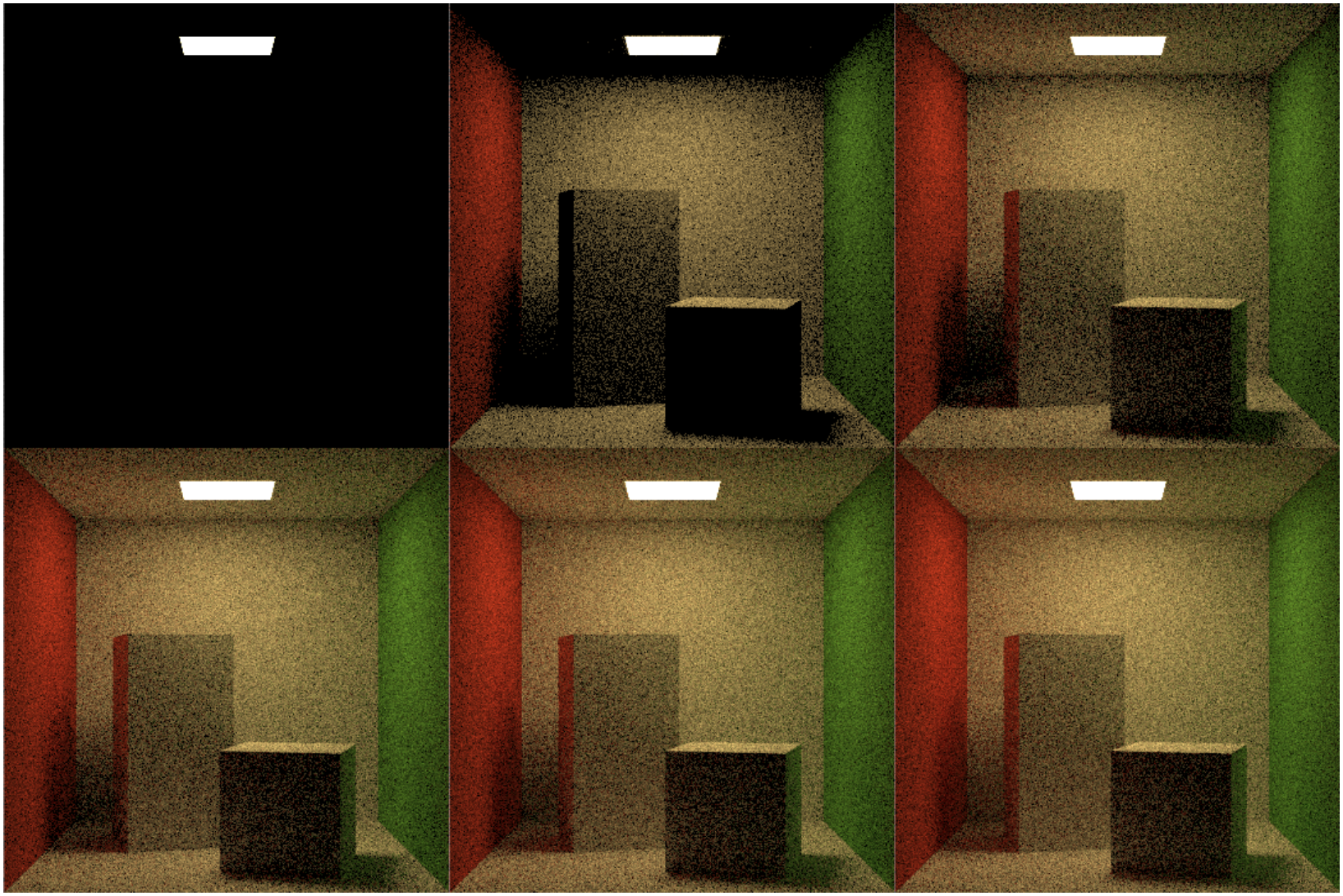


16 samples

256 samples

4096 samples

Naive path tracing – bounces



Russian roulette

- We want to avoid stopping at the arbitrary bounce since the energy associate with it depends on the scene configuration
- *Russian roulette*: stop with a given probability and raise the value of the remaining samples to the inverse of that probability to account for the energy loss due to stopping

$$g(\mathbf{x}) \text{ is an estimator for I}$$

$$g_r(\mathbf{x}) = \begin{cases} \frac{1}{p_r} g(\mathbf{x}) & \text{with probability } p_r \\ 0 & \text{with probability } 1 - p_r \end{cases}$$

$$E[g_r(\mathbf{x})] = p_r \cdot E[(1/p_r) g(\mathbf{x})] + (1 - p_r) \cdot 0 = E[g(\mathbf{x})]$$

Russian roulette

- In the context of Path tracing, we check whether a path needs to be terminate and, if not, correct the recursive radiance
- For the probability of termination, we could use the surface reflectivity since that would match the real world behavior
- But computing that might not be easy – alternatively, we compute the survival probability as

$$p_r = \min\left(1, \frac{f(\mathbf{i},\mathbf{o})|\mathbf{i}\cdot\mathbf{n}|}{p(\mathbf{i})}\right)$$

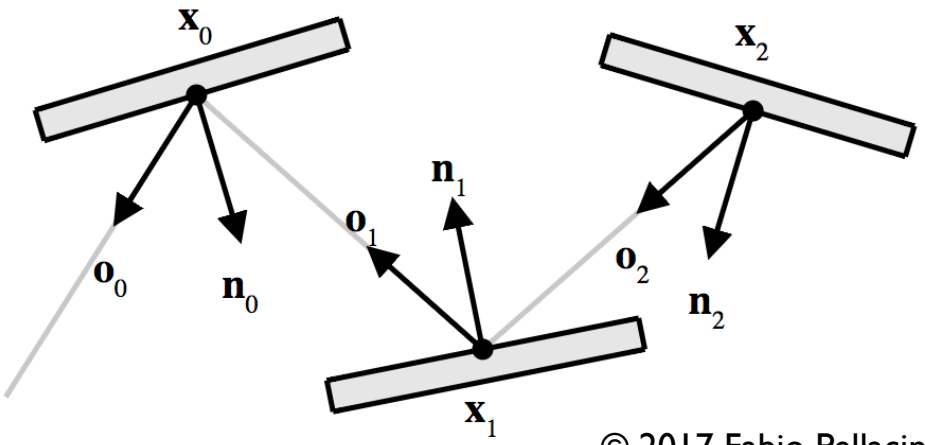
Path tracing – Russian roulette

```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    if(!pt.f) return le;

    if(bounce >= max_bounce) return pt.le;
    if(randf() >= pr(pt.f)) return l;       // russian roulette

    auto i = sample_brdfcos(n);
    auto li = estimate_li(scn, pt.x i);
    auto lr = li * eval_brdfcos(pt,i) / (pr(pt.f)*pdf(i));
    return le + lr;
}
```


Path tracing – product form

- One problem of the recursive evaluation is that it is harder to improve since the recursion "hides" the overall path behavior
- An alternative way to write a path tracer is to unroll the recursion
- Let us write the Monte Carlo estimator for the *j*-th point along a path

$$L(\mathbf{x}_j, \mathbf{o}_j) = L_e(\mathbf{x}_j, \mathbf{o}_j) + \left[f(\mathbf{x}_j, \mathbf{i}_j, \mathbf{o}_j)(\mathbf{n}(\mathbf{x}_j) \cdot \mathbf{i}_j)/p(\mathbf{i}_j) \right] L(\mathbf{x}_{j+1}, \mathbf{o}_{j+1}) =$$
$$= L_e(\mathbf{x}_j, \mathbf{o}_j) + w_j L(\mathbf{x}_{j+1}, \mathbf{o}_{j+1}) \quad \text{with } \mathbf{i}_j = -\mathbf{o}_{j+1}$$

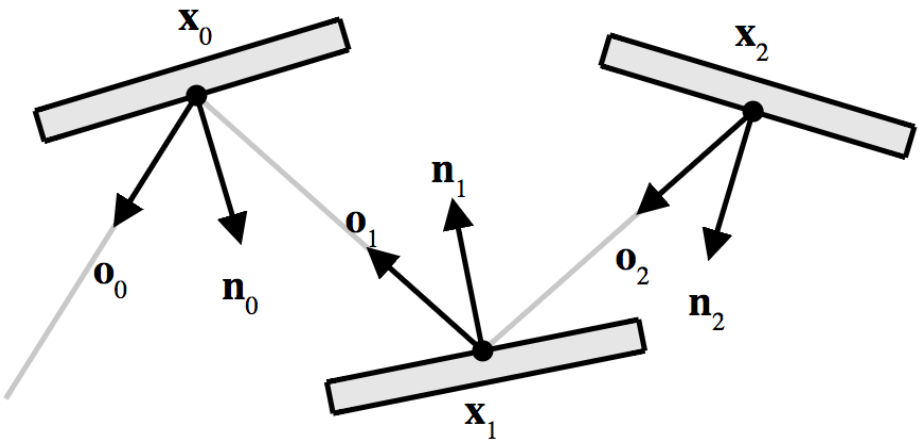


Path tracing – product form

- To compute the radiance of the *whole path* we unroll the previous recursion as

$$L_{path} = L_e(\mathbf{x}_0, \mathbf{o}_0) + w_1 L_e(\mathbf{x}_1, \mathbf{o}_1) + w_1 w_2 L_e(\mathbf{x}_2, \mathbf{o}_2) + \ldots =$$

$$= \sum_j W_j L_e(\mathbf{x}_j, \mathbf{o}_j) \quad \text{with } W_j = w_j W_{j-1} = \prod_{k=0}^{j} w_k$$



Path tracing - product form

```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    auto li = pt.le, w = {1,1,1};
    for(auto bounce : range(max_bounces)) {
        if(!pt.f) break;
        if(randf() > pr(f)) break; // russian roulette

        auto i = sample_brdfcos(n);
        auto bpt = intersect(scn, {pt.x, i});
        w *= eval_brdfcos(pt,i) / (pr(pt.f)*p(i)); // update w
        li += w * bpt.le;           // accumulate li

        pt = bpt;                   // “recurse”
    }
    return li;
}
```


Direct Illumination

Splitting direct and indirect

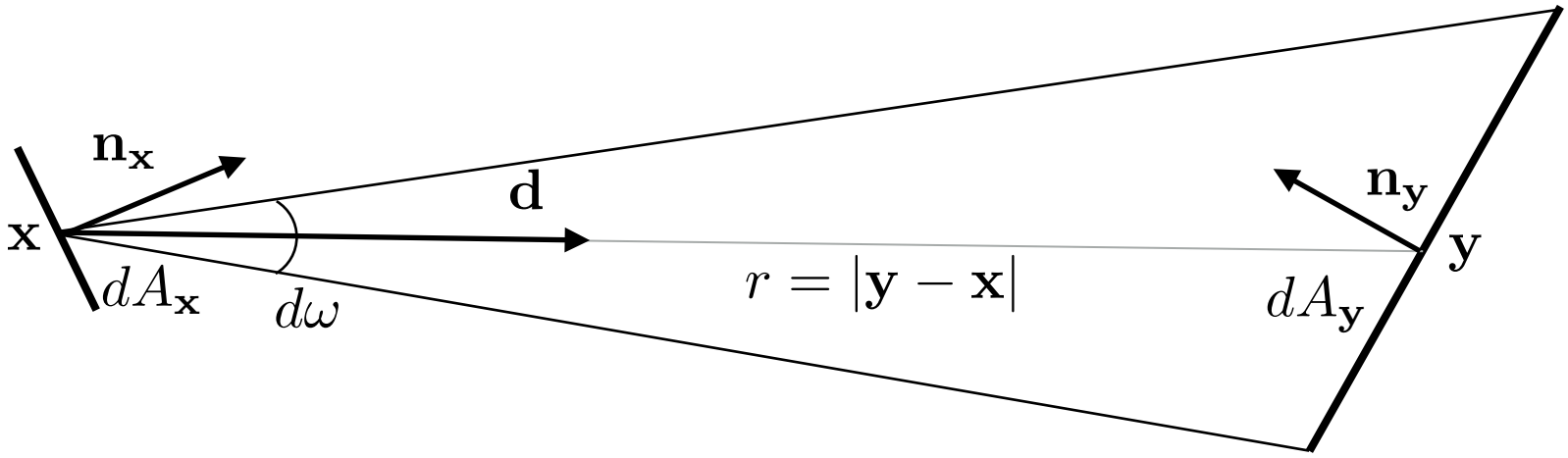
- The main concern of the previous renderer is that all paths that do not touch the light will end up with zero contribution
- Solution: connect each point in the path to at least one light
  - in jargon this is called next event estimation
- To do this, we split the integral of the reflected radiance *direct illumination*, i.e. the reflected radiance originating directly from light sources, and *indirect illumination*, i.e. all the rest
- For indirect illumination, we will proceed as before
- For direct illumination, we want to sample the lights directly to ensure that we hit one
  - for this we need a new formulation of the rendering equation

Radiance between differential areas

- Radiance between surfaces can be computed considering the solid angle subtended by one surface w.r.t. the other

$$L(\mathbf{x}, \mathbf{d}_{\mathbf{x}\to\mathbf{y}}) = \frac{d^2P}{dA_{\mathbf{x}}(\mathbf{d}\cdot\mathbf{n}_{\mathbf{x}})d\omega_{\mathbf{d}}} = \frac{d^2P|\mathbf{y}-\mathbf{x}|^2}{dA_{\mathbf{x}}(\mathbf{d}\cdot\mathbf{n}_{\mathbf{x}})dA_{\mathbf{y}}(-\mathbf{d}\cdot\mathbf{n}_{\mathbf{x}})} =$$

$$\text{with } \mathbf{d}_{\mathbf{x}\to\mathbf{y}} = \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|} \quad d\omega_{\mathbf{d}} = \frac{dA_{\mathbf{y}}(-\mathbf{d}\cdot\mathbf{n}_{\mathbf{x}})}{|\mathbf{y}-\mathbf{x}|^2}$$



Rendering equation – area form

- Angular form of the rendering equation

$$L(\mathbf{x},\mathbf{o}) = L_e(\mathbf{x},\mathbf{o}) + \int_{H^2_{\mathbf{x}}} L(\mathbf{r}(\mathbf{x},\mathbf{i}),-\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) |\mathbf{n_x}\cdot\mathbf{i}| d\omega_{\mathbf{i}}$$

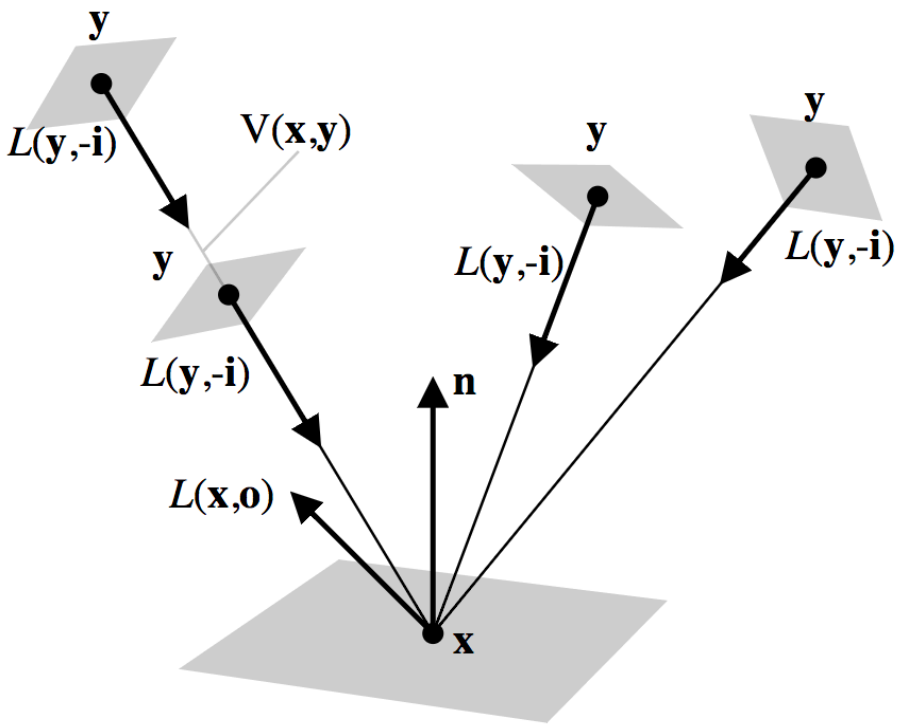
- Express the integral in terms of integrating over all possible surfaces in the scene; change the variable of integration from solid angle to area using the expression of the solid angle w.r.t. area
  - include a term *V* to decide whether two points are visible

$$L(\mathbf{x},\mathbf{o}) = L_e(\mathbf{x},\mathbf{o}) + \int_{\mathbf{y}\in S} L(\mathbf{y},-\mathbf{i}) V(\mathbf{x},\mathbf{y}) f(\mathbf{x},\mathbf{i},\mathbf{o}) G(\mathbf{x},\mathbf{y}) dA_{\mathbf{y}}$$

$$\text{with } \mathbf{i} = \frac{\mathbf{y}-\mathbf{x}}{|\mathbf{y}-\mathbf{x}|} \quad d\omega_{\mathbf{i}} = \frac{dA_{\mathbf{y}}|-\mathbf{i}\cdot\mathbf{n_y}|}{|\mathbf{y}-\mathbf{x}|^2} \quad G(\mathbf{x},\mathbf{y}) = \frac{|\mathbf{i}\cdot\mathbf{n_x}||-\mathbf{i}\cdot\mathbf{n_y}|}{|\mathbf{y}-\mathbf{x}|^2}$$

Rendering equation – area form

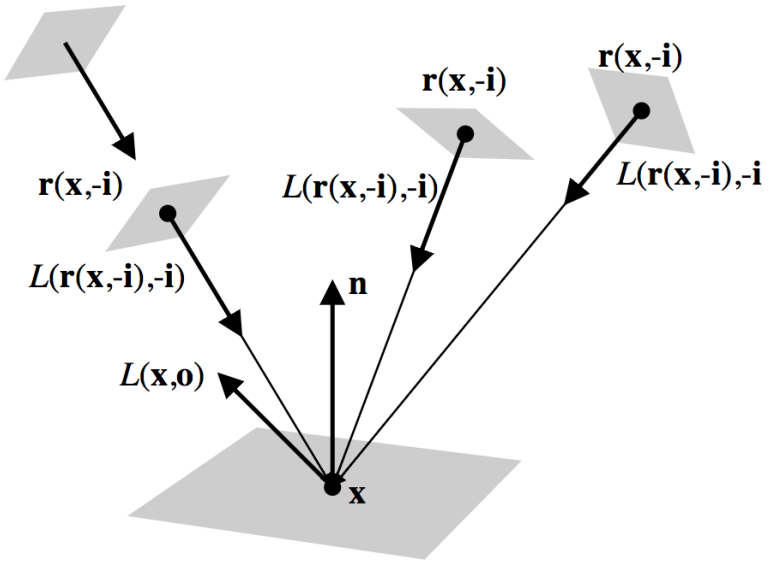
$$L(\mathbf{x},\mathbf{o}) = L_e(\mathbf{x},\mathbf{o}) + \int_{\mathbf{y}\in S} L(\mathbf{y},-\mathbf{i})V(\mathbf{x},\mathbf{y})f(\mathbf{x},\mathbf{i},\mathbf{o})G(\mathbf{x},\mathbf{y})dA_{\mathbf{y}}$$

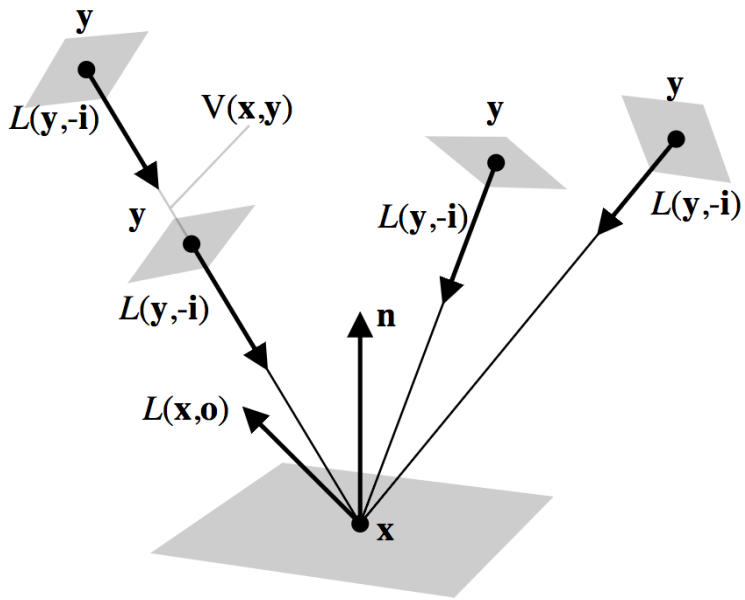


Angular vs area form

$$L(\mathbf{x},\mathbf{o}) = L_e(\mathbf{x},\mathbf{o}) + \int_{H^2_{\mathbf{x}}} L(\mathbf{r}(\mathbf{x},\mathbf{i}),-\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) |\mathbf{n_x}\cdot\mathbf{i}| d\omega_{\mathbf{i}}$$

$$L(\mathbf{x},\mathbf{o}) = L_e(\mathbf{x},\mathbf{o}) + \int_{\mathbf{y}\in S} L(\mathbf{y},-\mathbf{i}) V(\mathbf{x},\mathbf{y}) f(\mathbf{x},\mathbf{i},\mathbf{o}) G(\mathbf{x},\mathbf{y}) dA_{\mathbf{y}}$$





Angular vs area form

- **Angular form**: integrate over incoming angle
  - use raytracing to determine the visible surface from where the light comes from
- **Area form**: integrate over all scene's surfaces
  - use raytracing to determine whether a given point is visible from another, i.e. whether a point contributes to the integral (shadow rays)

Splitting direct and indirect

- Let us now write the reflection equation by splitting direct and indirect illumination

$$
\begin{aligned}
L_r(\mathbf{x},\mathbf{o}) &= \int_{H^2} L_i(\mathbf{x},\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) |\mathbf{n_x}\cdot\mathbf{i}| d\omega_\mathbf{i} = \\
&= \int_{H^2} L_e(\mathbf{x},\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) |\mathbf{n_x}\cdot\mathbf{i}| d\omega_\mathbf{i} + \\
&+ \int_{H^2} L_r(\mathbf{x},\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) |\mathbf{n_x}\cdot\mathbf{i}| d\omega_\mathbf{i} = \\
&= L_d(\mathbf{x},\mathbf{o}) + L_{ind}(\mathbf{x},\mathbf{o})
\end{aligned}
$$

Direct illumination – hemisphere

- Direct illumination can be thought of as the solution of the reflection equation where the incoming illumination is just emission term

$$L_d(\mathbf{x}, \mathbf{o}) = \int_{H^2_{\mathbf{x}}} L_e(\mathbf{r}(\mathbf{x}, \mathbf{i}), -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}| d\omega_{\mathbf{i}}$$

- We can solve the direct illumination equation with Monte Carlo by picking random directions and evaluating the integrand function

$$L_d(\mathbf{x}, \mathbf{o}) \approx \frac{L_e(\mathbf{r}(\mathbf{x}, \mathbf{i}_i), -\mathbf{i}_i) f(\mathbf{x}, \mathbf{i}_i, \mathbf{o}) |\mathbf{n_x} \cdot \mathbf{i}_i|}{p(\mathbf{i}_i)}$$

- The problem here is becomes obvious – each ray with L_e zero has no contribution

Direct illumination – area

- Consider the same solution but now with the area formulation

$$L_d(\mathbf{x}, \mathbf{o}) = \int_{\mathbf{y} \in A_{light}} L_e(\mathbf{y}, -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) V(\mathbf{x}, \mathbf{y}) G(\mathbf{x}, \mathbf{y}) dA_{\mathbf{y}}$$

- The estimator for this formulation is

$$L_d(\mathbf{x}, \mathbf{o}) \approx \frac{L_e(\mathbf{y}_i, -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) V(\mathbf{x}, \mathbf{y}_i) G(\mathbf{x}, \mathbf{y}_i)}{p(\mathbf{y}_i)}$$

- To evaluate it, we have to pick points uniformly over the light surface

Sampling the light surface

- To evaluate it, we have to pick points uniformly over the light surface
- For a square light of area *A*, we can warp two random numbers as

$$\mathbf{y}^l = \sqrt{A}[r_1 - 0.5, r_2 - 0.5, 0]$$

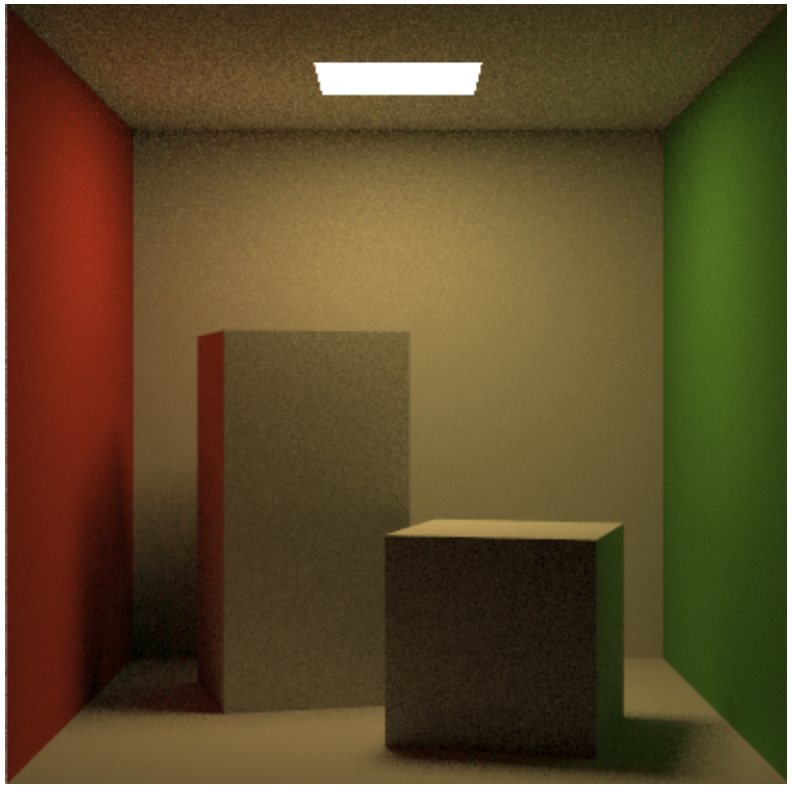
$$\mathbf{y}^l \sim p(\mathbf{y}^l) = \frac{1}{A}$$

- To move to world coordinates, we use a frame that describes the orientation of the square light

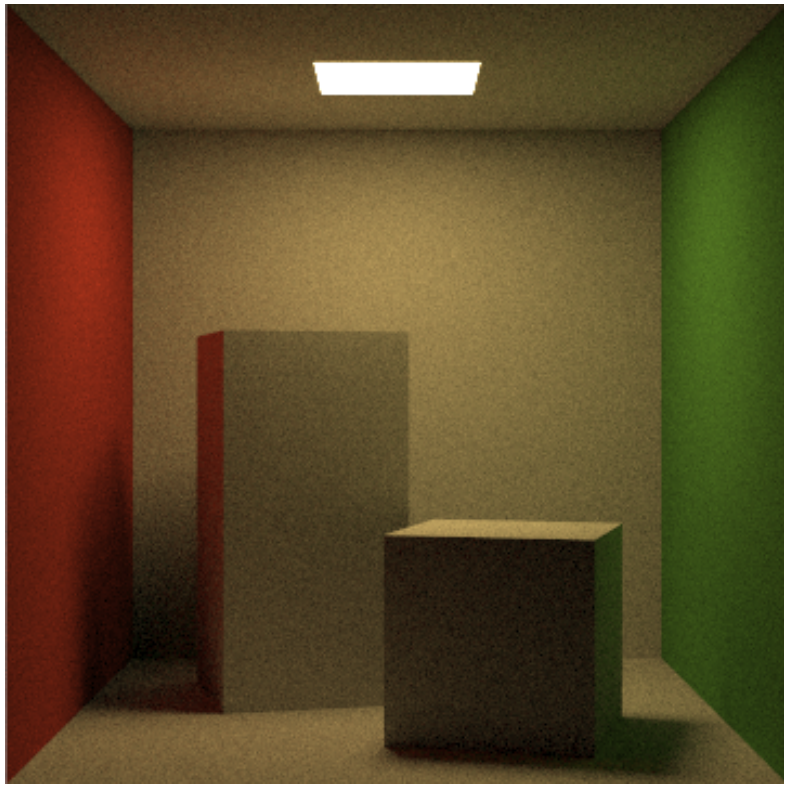
$$\mathbf{y} = \sqrt{A}(r_1 - 0.5) \cdot \mathbf{F_x} + \sqrt{A}(r_2 - 0.5) \cdot \mathbf{F_y} + \mathbf{F_o}$$

$$\mathbf{y} \sim p(\mathbf{y}) = \frac{1}{A}$$

Hemispherical vs area

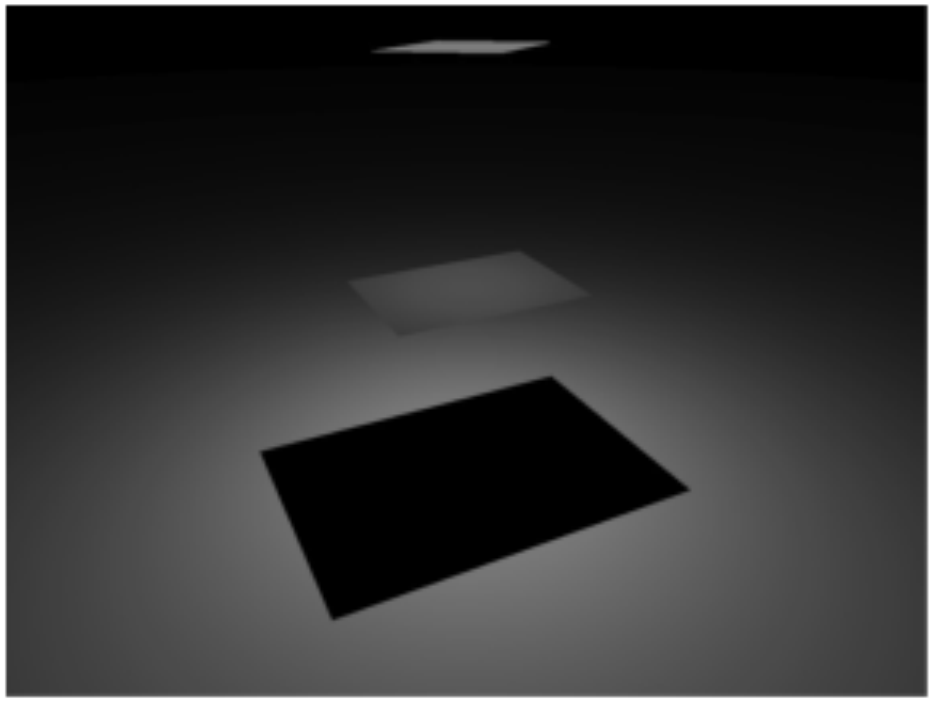


64 area samples

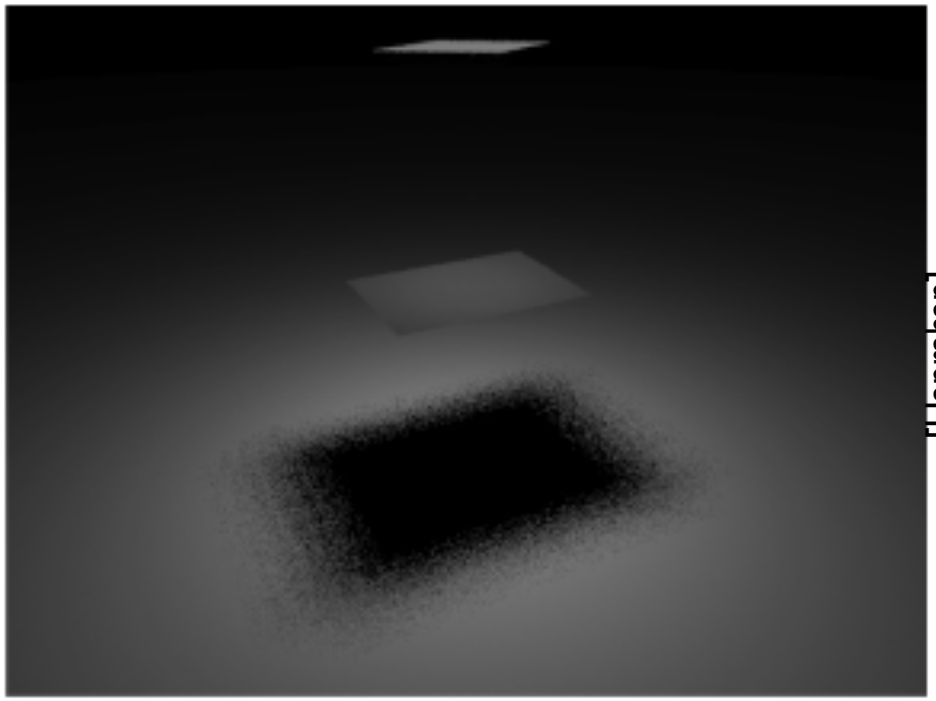


4096 hemispherical samples

Monte Carlo vs quadrature



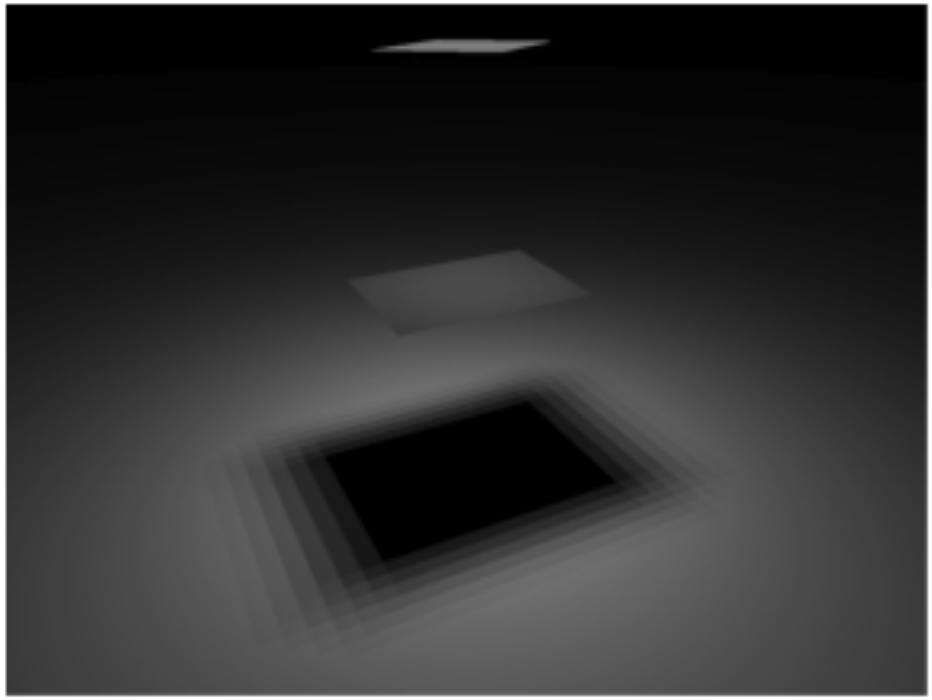
1 sample – center



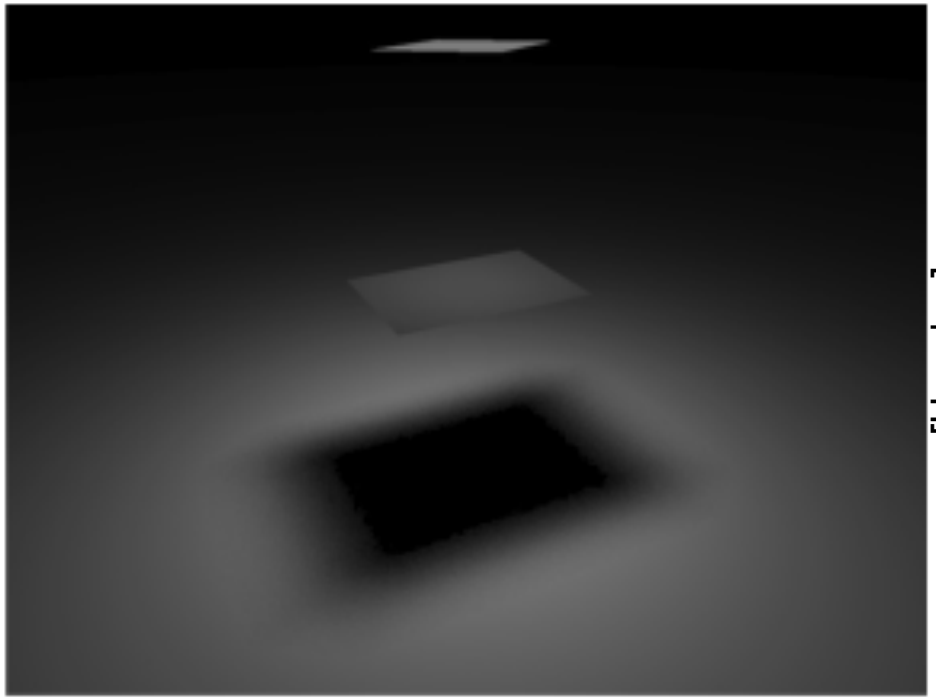
16 samples – random

[Hanrahan]

Monte Carlo vs quadrature



16 samples – center



16 samples – stratified

[Hanrahan]

Sampling multiple lights

- To handle multiple lights, we can estimate their contributions independently since each light integral is independent

$$L_d(\mathbf{x},\mathbf{o}) = \int_{\mathbf{y}\in A_{lights}} L_e(\mathbf{y},-\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) V(\mathbf{x},\mathbf{y}) G(\mathbf{x},\mathbf{y}) dA_{\mathbf{y}} =$$

$$= \sum_l^{n_l} \int_{\mathbf{y}\in A_l} L_e(\mathbf{y},-\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) V(\mathbf{x},\mathbf{y}) G(\mathbf{x},\mathbf{y}) dA_{\mathbf{y}}$$

$$L_d(\mathbf{x},\mathbf{o}) \approx \sum_l^{n_l} \frac{A_{l_i}}{N} \sum_i A_{l_i} L_e(\mathbf{y}_i,-\mathbf{i}) f(\mathbf{x},\mathbf{i},\mathbf{o}) V(\mathbf{x},\mathbf{y}_i) G(\mathbf{x},\mathbf{y}_i)$$

Sampling multiple lights

- As the number of lights increases, this formulation becomes inefficient
- The alternative is to pick a *light at random for each sample*
- Here we are sampling from a uniform *discreet* distribution

$$L_d(\mathbf{x}, \mathbf{o}) = \sum_l^{n_l} \int_{\mathbf{y} \in A_l} L_e(\mathbf{y}, -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) V(\mathbf{x}, \mathbf{y}) G(\mathbf{x}, \mathbf{y}) dA_{\mathbf{y}}$$

$$L_d(\mathbf{x}, \mathbf{o}) \approx A_{l_i} L_e(\mathbf{y}_i, -\mathbf{i}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) V(\mathbf{x}, \mathbf{y}_i) G(\mathbf{x}, \mathbf{y}_i)$$

$$l_i = \text{floor}(r \cdot n_l) \quad l \sim p(l) = \frac{1}{n_l}$$

Path tracing – direct + indirect

```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    auto li = le, w = {1,1,1};
    for(auto bounce : range(max_bounces)) {
        if(!pt.f) break;
        auto lgt = sample_lights(scn->lights);
        auto lpt = sample_light(lgt, pt);
        if(!intersect(scn, {pt.x, -lpt.o}))
            li += w * lpt.le * eval_brdfcos(pt,-lpt.o) *
                      (dot(lpt.n,lpt.o) / |lpt.x-pt.x|^2) /
                      (pdf(lgt)*pdf(lpt));
        if(randf() > pr(f)) break; // russian roulette
        auto bpt = intersect(scn, {pt.x, sample_brdfcos(n)});
        w *= eval_brdfcos(pt,i) / (pr(pt.f)*pdf(i)); // update w
        // do not accumulate emission since done before
        pt = bpt;                          // "recurse"
    }
    return li;
}
```

DIRECT

INDIRECT

Path tracing – direct + indirect

- The previous formulation mixes area and hemispherical sampling
- While correct, this is prime to error as we continue adding features
- For convenience, fold the cosine and distance squared in the light pdf
  - this is equivalent to a change of variable from area to hemisphere
- Also fold the PDF of closing a single light into point pdf

$$p_h(\mathbf{y}) = \frac{|\mathbf{x} - \mathbf{y}|^2}{n_l A |\mathbf{n_y} \cdot \mathbf{o_y}|}$$

Path tracing – direct + indirect

```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    auto li = le, w = {1,1,1};
    for(auto bounce : range(max_bounces)) {
        if(!pt.f) break;
        auto lpt = sample_lights(scn->lights, pt);
        if(!intersect(scn, {pt.x, -lpt.o}))
            li += w*lpt.le * eval_brdfcos(pt,-lpt.o) / pdf(lpt);
        if(randf() > pr(f)) break; // russian roulette
        auto bpt = intersect(scn, {pt.x, sample_brdfcos(n)});
        w *= eval_brdfcos(pt,i) / (pr(pt.f)*pdf(i)); // update w
        // do not accumulate emission since done before
        pt = bpt;                          // "recurse"
    }
    return li;
}
```

DIRECT

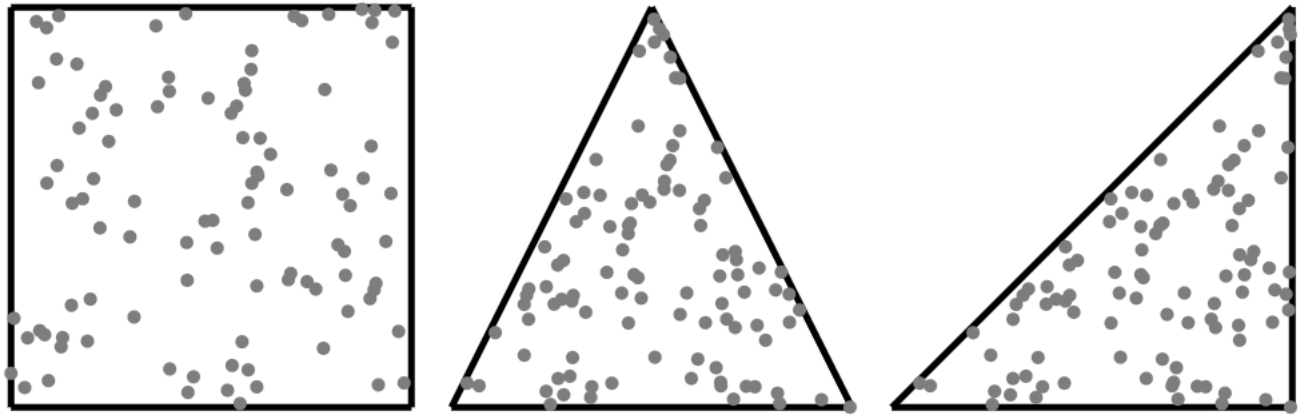
INDIRECT

Mesh Area Lights

Sampling triangle meshes

- In general, our surfaces are described by triangle meshes
- We want to a procedure to pick point *uniformly* on the entire surface
- Let us start by picking points on a triangle with uniform probability by warping from the unit square

$$\mathbf{y} = \sqrt{r_0}(1 - r_1)\mathbf{v}_0 + (1 - \sqrt{r_1})\mathbf{v}_1 + r_1\sqrt{r_0}\mathbf{v}_2$$



Sampling triangle meshes

- Now we want to pick a triangle at random from the set of triangles with probability proportional to its area
- This is a non-uniform discreet distribution
- To sample it, we warp a random number in [0,1) to the distribution by
  - compute the triangle areas
  - from these, derive the cdf of the distribution
  - for each sample, we use a random number in [0,1) to find the smallest cdf value that is larger than the given one
  - its index is the wanted triangle index

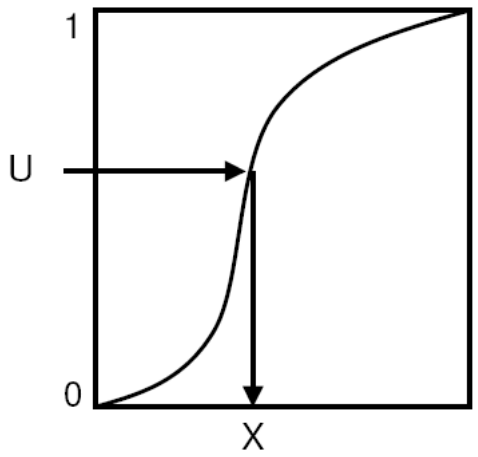
$$p_j = \frac{A_j}{\sum_k A_k} \quad c_j = \sum_{k=0}^{j} p_k \quad t_i = \arg\min_j c_j \geq r$$

Sampling discreet distribution

- Theorem: given the uniform random variable *U* in [0,1) and a cumulative distribution function *c*, then the random variable $X=c^{-1}(U)$ has *c* as its distribution

$$U \in [0,1) \sim p_U = 1 \rightarrow X = c^{-1}(U) \sim p_X = c$$

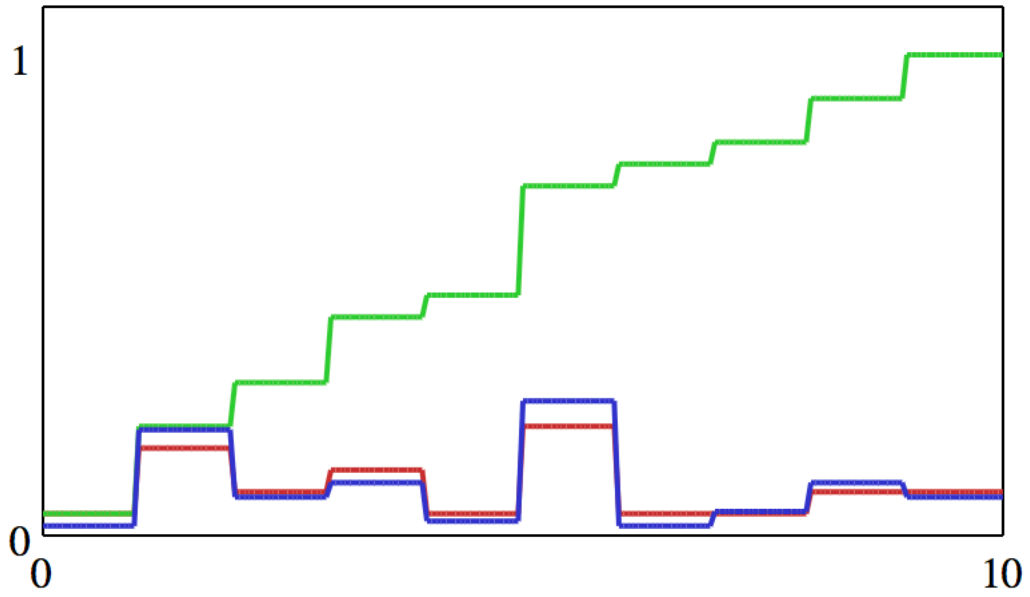
- To sample a random variable according to a distribution *p* do
  - compute its cumulative distribution function *c* and its inverse c^{-1}
  - use c^{-1} to warp uniform samples to the new domain



[Krivanec]

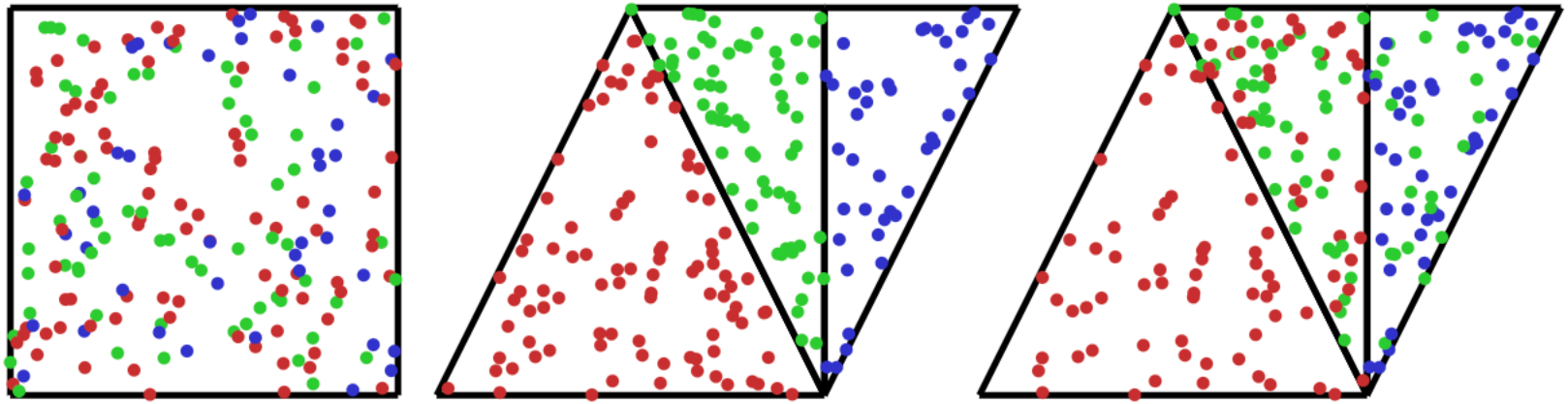
Sampling triangle meshes

- Example case with 10 triangles



Sampling triangle meshes

- Comparison between sampling uniformly the triangle index or sampling proportionally to area



Sampling triangles meshes

```
vector<float> triangles_cdf(vector<vec3i> triangles,
                            vector<vec3f> pos) {
    auto cdf = vector<float>();
    for (auto t : triangles) {
       cdf.push_back(
           triangle_area(pos[t.x], pos[t.y], pos[t.z]));
    }
    for (auto i : range(1, cdf.size()) cdf[i] += cdf[i - 1];
    return cdf;
}

int sample_triangles(vector<float> cdf, float rn) {
    rn *= cdf.back();
    return (int)(std::upper_bound(cdf.begin(),
        cdf.end(), rn) - cdf.begin());
}
```


Environment map

- Use HDR texture as background illumination
- For LatLong parametrization, the outgoing radiance is written as

$$L_{env}(\mathbf{i}^l) = k_e * txt \left[\frac{\text{atan2}(\mathbf{i}^l_z, \mathbf{i}^l_x)}{2\pi}, \frac{\text{acos}(\mathbf{i}^l_y)}{\pi} \right]$$

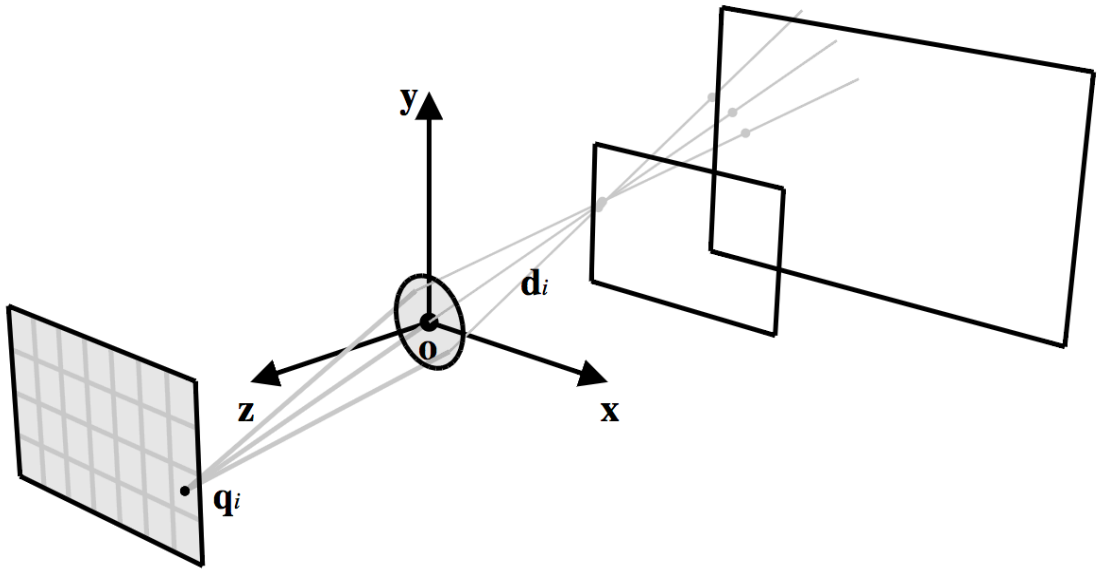
- We can also consider environment maps as light source, in which case we would sample outgoing radiance as a 2D piecewise-linear distribution
  - not covered in these slides

Realistic camera

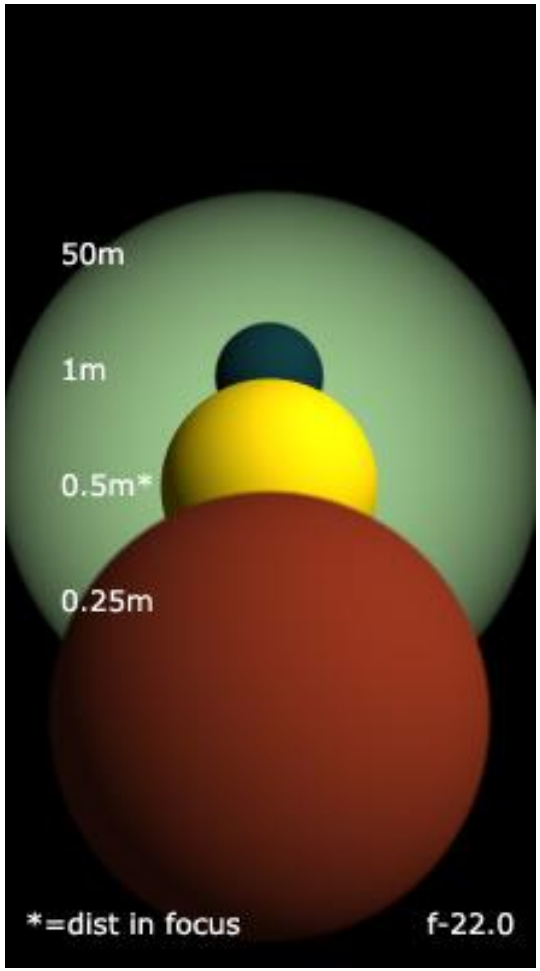
Pixel radiance

- We account for lenses with finite aperture by also integrating for all possible positions on the lens itself; this give us defocus blur

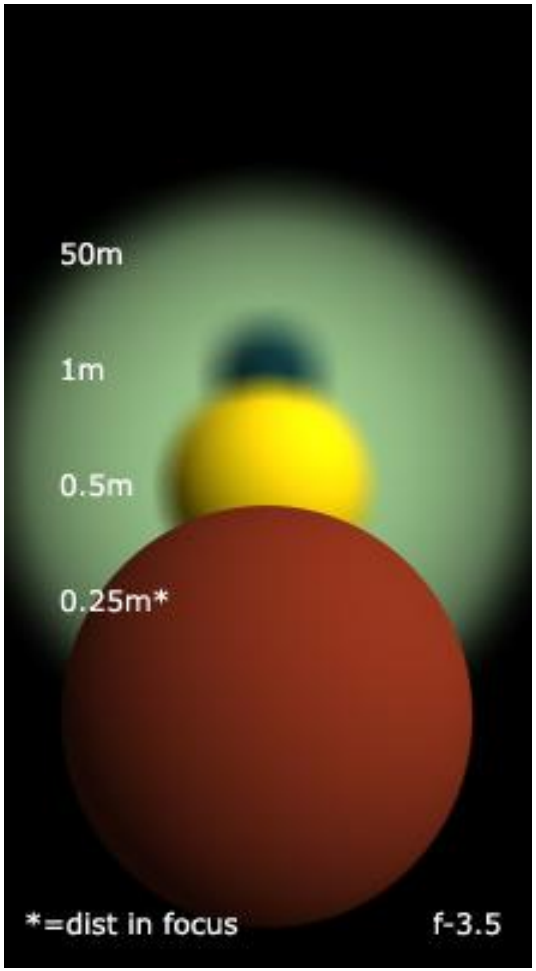
$$L_{pixel} = \frac{1}{A_{pixel}} \frac{1}{A_{lens}} \int_{\mathbf{q} \in pixel} \int_{\mathbf{e} \in lens} L_i(\mathbf{q}, \mathbf{d}) dA_{\mathbf{q}} dA_{\mathbf{e}} \text{ with } \mathbf{d} = \frac{\mathbf{e} - \mathbf{q}}{|\mathbf{e} - \mathbf{q}|}$$



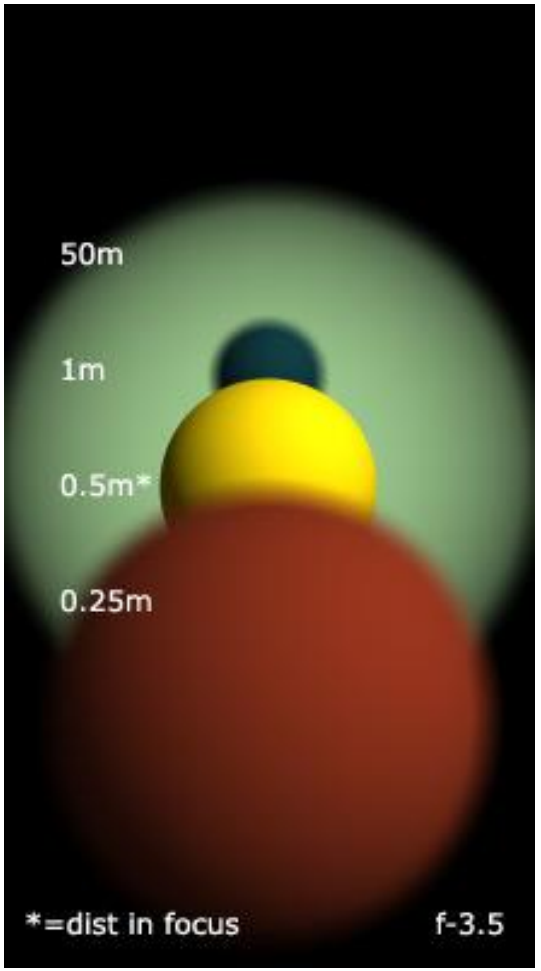
Pixel radiance



pinhole



close focus

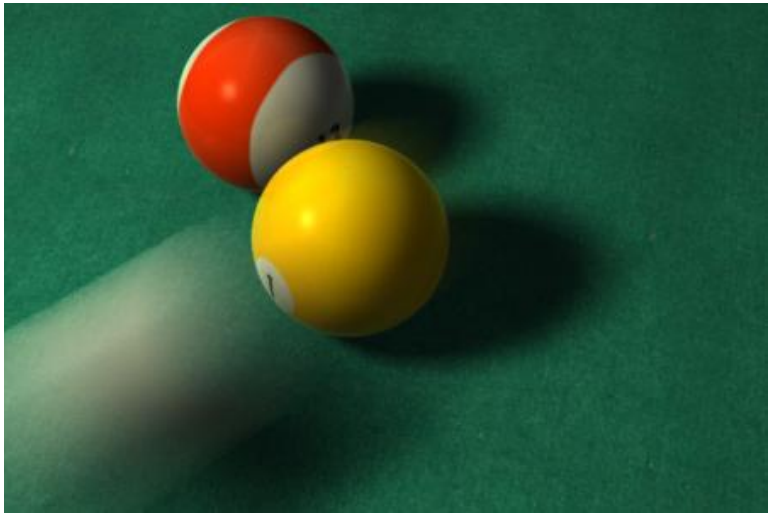


far focus

Rendering animation

- In general the camera stays open for a finite amount of time while the objects may move in the scene; this gives us motion blur
- We can compute this effect by integrating over time and considering a time-varying scene, say due to animation

$$L_{pixel} = \frac{1}{A_{pixel}} \frac{1}{A_{lens}} \frac{1}{t_1 - t_0} \int_{t_0}^{t_1} \int_{\mathbf{q} \in pixel} \int_{\mathbf{e} \in lens} L_i(\mathbf{q}, \mathbf{d}, t) dA_{\mathbf{q}} dA_{\mathbf{e}} dt$$

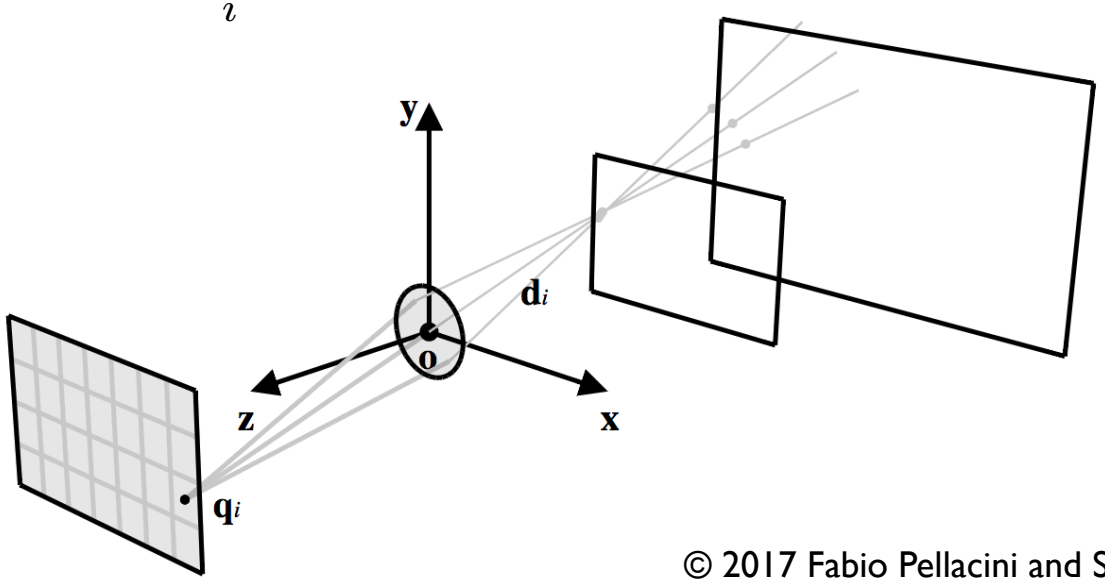


Sampling the lens

- Monte Carlo integration works just fine when considering the lens too
  - note how we numerically compute two integrals at once

$$L_{pixel} = \frac{1}{A_{pixel}} \frac{1}{A_{lens}} \int_{\mathbf{q} \in pixel} \int_{\mathbf{e} \in lens} L_i(\mathbf{q}, \overrightarrow{\mathbf{eq}}) dA_{\mathbf{q}} dA_{\mathbf{e}}$$

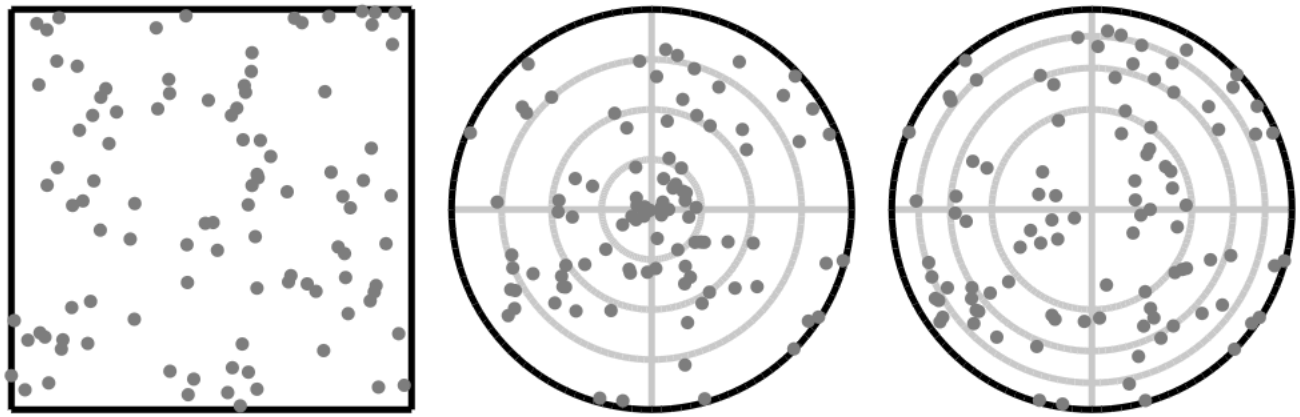
$$\approx \frac{1}{N} \sum_{i}^{N} L_i(\mathbf{q}_i, \overrightarrow{\mathbf{e}_i \mathbf{q}_i})$$



Sampling the lens

- For square lens, we could generate the lens positions by warping to a small square like we did not far – but lenses are really circular
- If we warp with polar coordinates we get points on the circles that are not equally distributed since the areas change
- To sample the circle, we *warp with an area-preserving method*

$$\mathbf{e}^l = [ar\cos\theta, ar\sin\theta, 0] \quad \text{with } \theta = 2\pi r_1, r = \sqrt{r_2}$$



Sampling the lens

```
ray3f sample_camera(camera* cam, vec2f uv, vec2f luv) {
    auto ql = vec3f{(uv.x-0.5),(uv.y-0.5),1} * cam->focus;
    auto ll = vec3f{cosf(2 * pif * luv.x) * sqrt(luv.y),
                    sinf(2 * pif * luv.x) * sqrt(luv.y), 0} *
              cam->aperture;
    return {transform_point(cam->frame, ll),
        transform_direction(cam->frame, normalize(-ql - ll))};
}
```


Proof of Recursive Solution

Rendering eq. - operator form

- We can think of reflection as an operator *T* that takes as input the radiance function and return a new radiance function

$$T : L(\mathbf{x}, \mathbf{o}) \rightarrow L_r(\mathbf{x}, \mathbf{o})$$

$$(TL)(\mathbf{x}, \mathbf{o}) = \int_{\mathbf{y} \in S} L(\mathbf{y}, -\mathbf{i}) V(\mathbf{x}, \mathbf{y}) f(\mathbf{x}, \mathbf{i}, \mathbf{o}) G(\mathbf{x}, \mathbf{y}) dA_{\mathbf{y}}$$

- Rendering equation

$$L(\mathbf{x}, \mathbf{o}) = L_e(\mathbf{x}, \mathbf{o}) + (TL)(\mathbf{x}, \mathbf{o})$$

- In shorthand notation

$$L = L_e + TL$$

Rendering eq. - operator form

- Formal solution of the rendering equation is

$$L = L_e + TL \rightarrow (I - T)L = L_e \rightarrow L = (I - T)^{-1} L_e$$

- Unusable in practice since we cannot compute the inverse
- Solve by recursive substitution

$$L = L_e + TL = L_e + T(L_e + TL) = L_e + TL_e + T^2 L =$$
$$= L_e + TL_e + T^2(L_e + TL) = L_e + TL_e + T^2 L_e + T^3 L = \ldots$$

- This leads to the solution written as a Neumann series

$$L = \sum_{i=0}^{\infty} T^i L_e$$

Rendering eq. - operator form

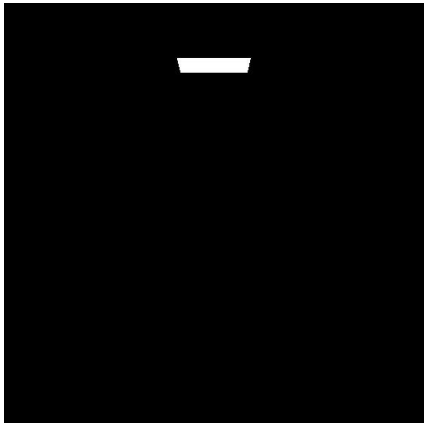
- We can show that the Neumann series is the right solution by equating it in the formal solution

$$L = \sum_{i=0}^{\infty} T^i L_e \qquad L = (I - T)^{-1} L_e$$

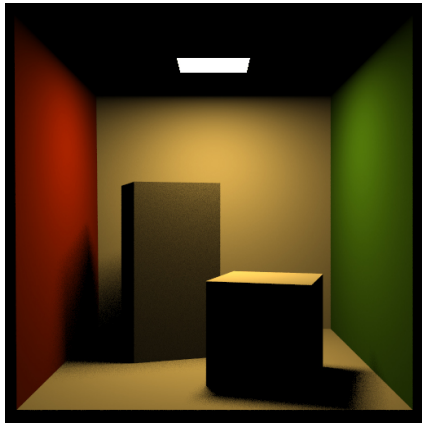
- Since these equations are true for every *Le*, we have that

$$\begin{aligned}
(I - T)^{-1} &= \sum_{i=0}^{\infty} T^i L_e \\
(I - T)(I - T)^{-1} &= (I - T) \sum_{I=0}^{\infty} T^i L_e \\
I &= \sum_{i=0}^{\infty} T^i L_e - \sum_{i=1}^{\infty} T^i L_e \\
I &= I
\end{aligned}$$

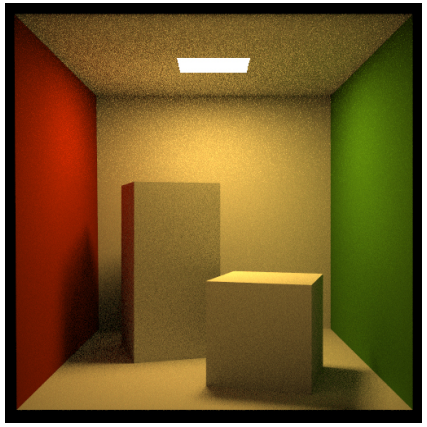
Rendering eq. - operator form



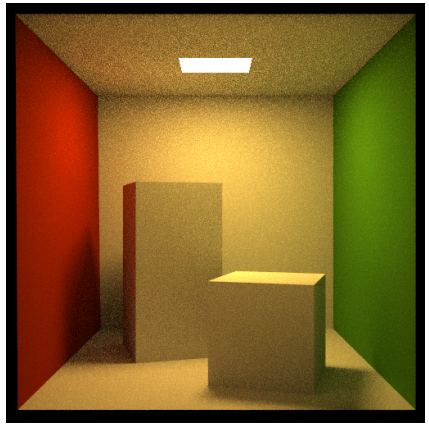
L_e



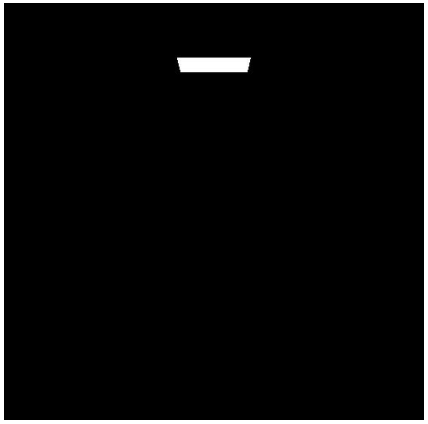
$L_e + TL_e$



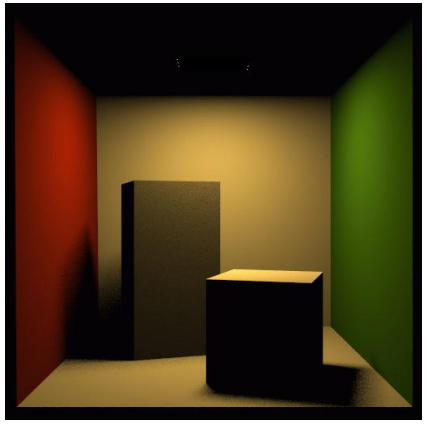
$L_e + TL_e + T^2L_e$



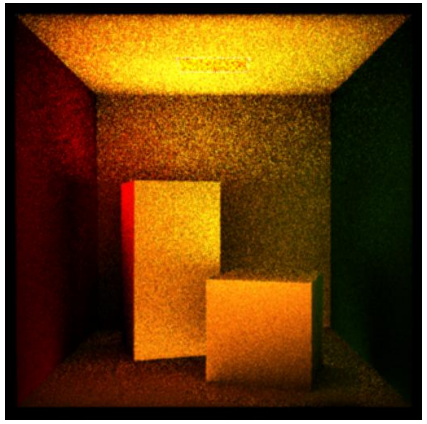
$L_e + ... + T^3L_e$



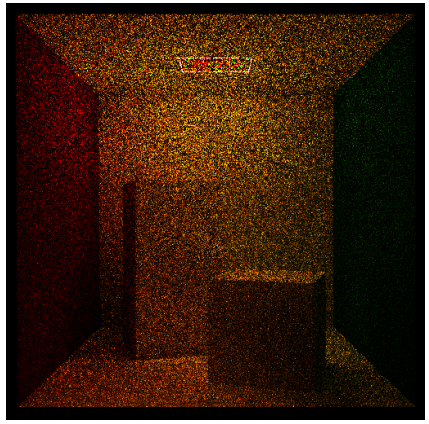
L_e



TL_e



T^2L_e (16x)



T^3L_e (16x)

[Dutrè, Bala]

Properties of TL

- Intuition: *TL* bounces light once
- From the conservation of energy, we know that the norm of *TL* will be less or equal to the one of *L*
  - since light is either reflected of absorbed
  - similar to energy conservation for the BSDF
- The higher the BSDFs of the scene the more bounces we need since energy decreases less at each bounce

Recursive solution

- We know now how to recursively solve the rendering equation

$$L = \sum_{i=0}^{\infty} T^i L_e$$

- Let us rewrite this in terms of the original integrals

$$\begin{aligned} L(\mathbf{x}_0, \mathbf{o}_0) = L_e(\mathbf{x}_0, \mathbf{o}_0) + \\ + \int_{\mathbf{x}_1} L_e(\mathbf{x}_1, \mathbf{o}_1) V(\mathbf{x}_0, \mathbf{x}_1) f(\mathbf{x}_0, \mathbf{i}_0, \mathbf{o}_0) G(\mathbf{x}_0, \mathbf{x}_1) dA_{\mathbf{x}_1} + \\ + \int_{\mathbf{x}_1} \left(\int_{\mathbf{x}_2} L_e(\mathbf{x}_2, \mathbf{o}_2) V(\mathbf{x}_1, \mathbf{x}_2) f(\mathbf{x}_1, \mathbf{i}_1, \mathbf{o}_1) G(\mathbf{x}_1, \mathbf{x}_2) dA_{\mathbf{x}_2} \right) \cdot \\ \cdot V(\mathbf{x}_0, \mathbf{x}_1) f(\mathbf{x}_0, \mathbf{i}_0, \mathbf{o}_0) G(\mathbf{x}_1, \mathbf{x}_2) dA_{\mathbf{x}_2} + \ldots \end{aligned}$$

Recursive solution

- We can write the recursive solution by aggregating the product of all visibility, BRDF and geometric terms in a single expression called *path throughput*

$$\mathrm{Tr}(\mathbf{x}_0, ..., \mathbf{x}_i) = \prod_{j=0}^{i-1} V(\mathbf{x}_j, \mathbf{x}_j + 1) f(\mathbf{x}_j, \mathbf{i}_j, \mathbf{o}_j) G(\mathbf{x}_j, \mathbf{x}_{j+1})$$

$$L(\mathbf{x}, \mathbf{o}) = \sum_{i=0}^{\infty} \int_{\mathbf{x}_1} ... \int_{\mathbf{x}_i} L_e(\mathbf{x}_i, \mathbf{o}_i) \mathrm{Tr}(\mathbf{x}_0, ..., \mathbf{x}_i) dA_{\mathbf{x}_1} ... dA_{\mathbf{x}_i}$$

Rendering eq. – path integral form

- From the previous formulation, we can get some intuition on yet another form of the rendering equation called *path formulation*
- The basic idea is to think about integrating not recursively over position but in *space of all possible paths* that the light can travel into
- A light path is a sequence of points of *any length*

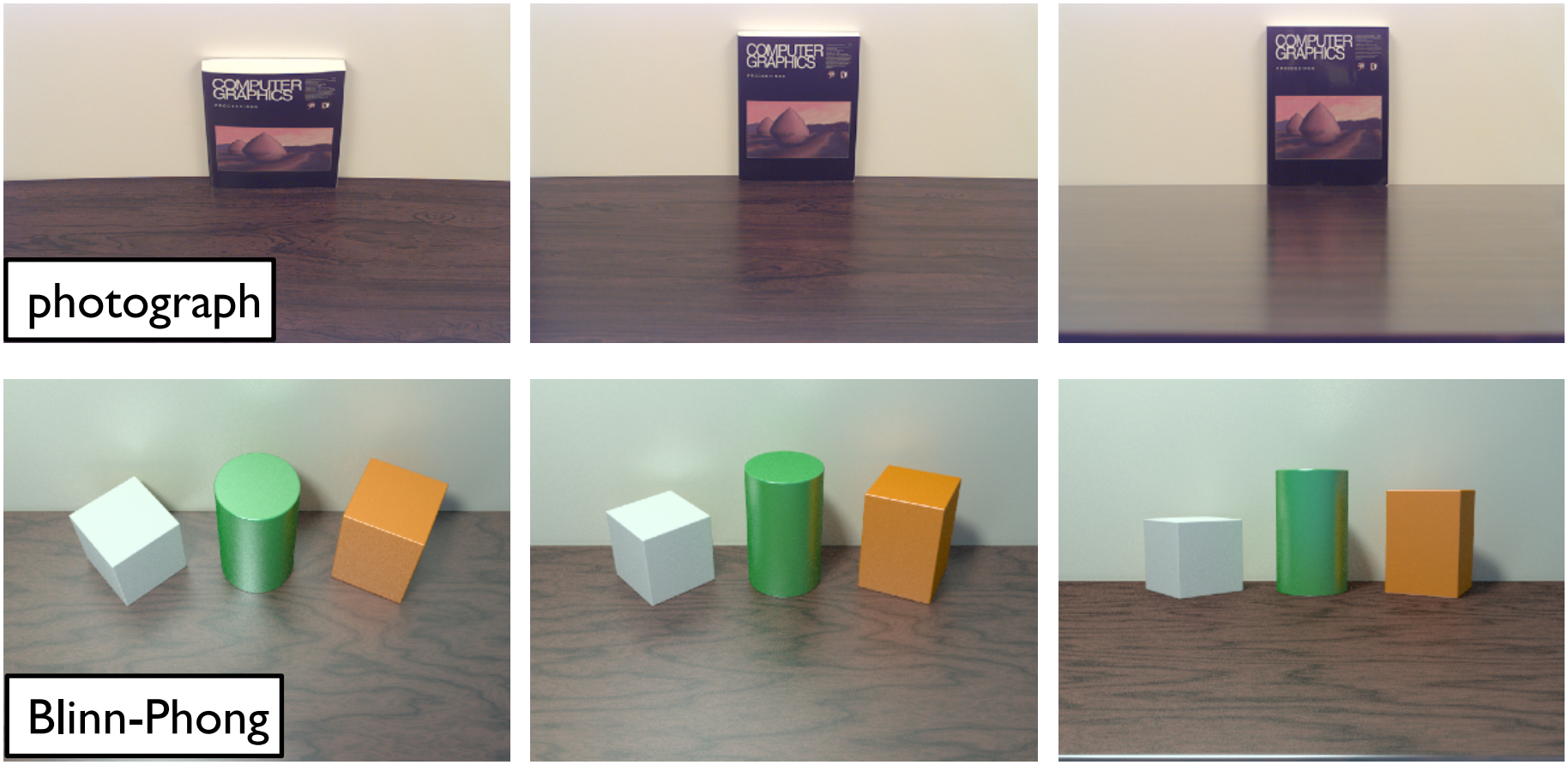
$$\mathbf{p} = \{\mathbf{x}_0, ..., \mathbf{x}_i\}$$

- We can rewrite the previous solution as the integral of the radiance of all possible path that end at the camera

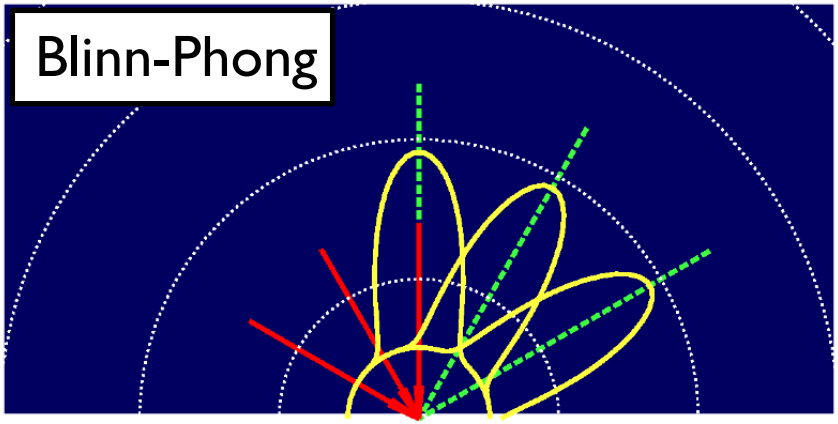
$$L(\mathbf{x}, \mathbf{o}) = \int_{\mathbf{p} \in \mathcal{P}} L_e(\mathbf{p}_{end}) \mathrm{Tr}(\mathbf{p}) d\mu_{\mathbf{p}}$$

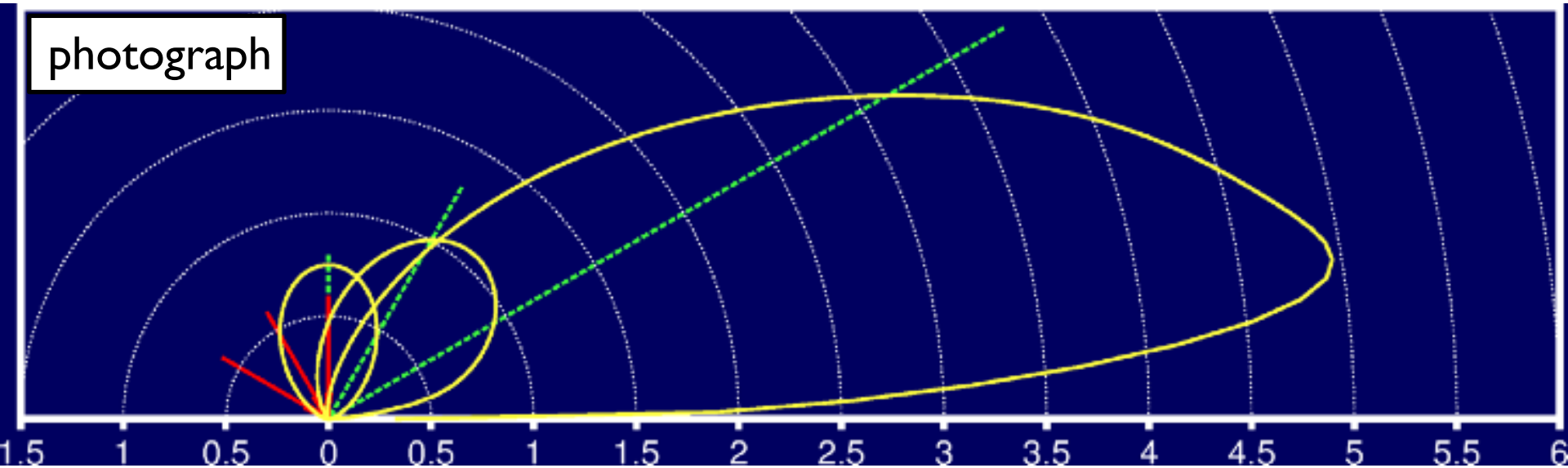
Materials

Blinn-Phong vs physically-based



Blinn-Phong vs physically-based

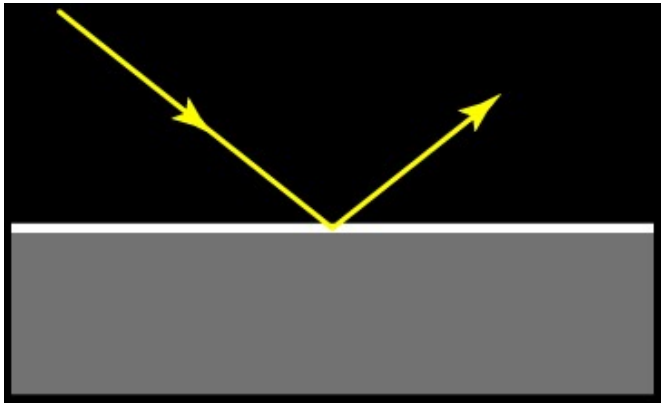


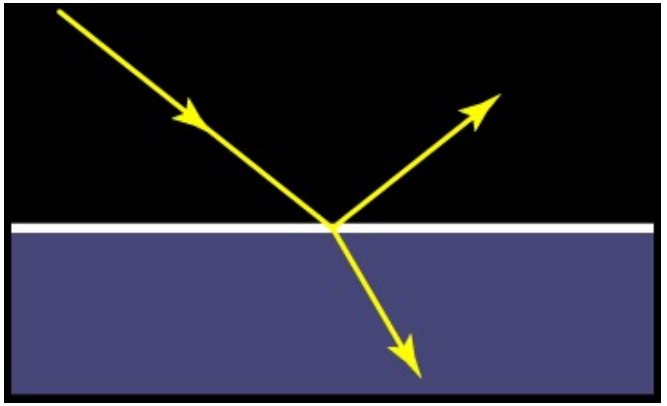


Materials

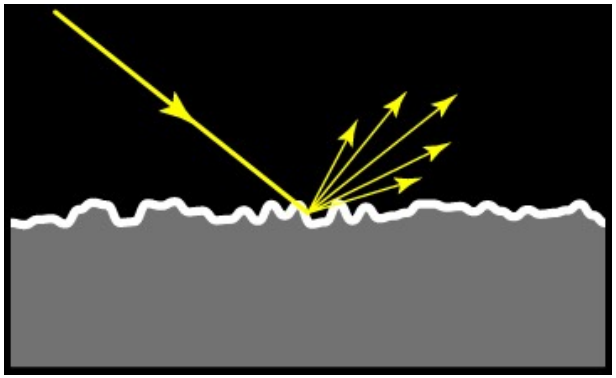
- We'll focus on BSDF and in particular in covering opaque materials
- Different BSDFs comes from different optical behavior and "structure" of the surface
- In theory, most materials would need an entirely different BSDF to account for their peculiarities
  - not practical since editing and exchanging them becomes hard
- In practice, use one "good-enough" physical model for most cases and custom materials only when absolutely necessary

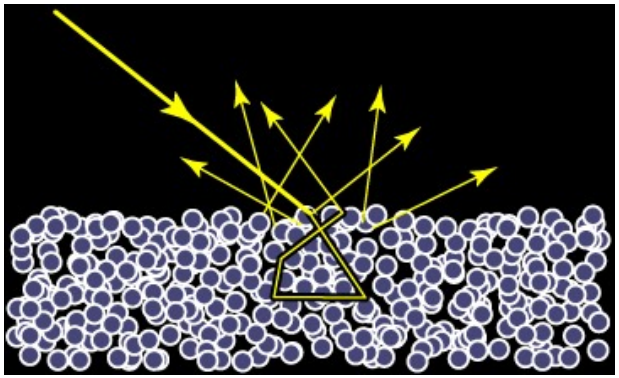
Smooth surfaces





Rough surfaces

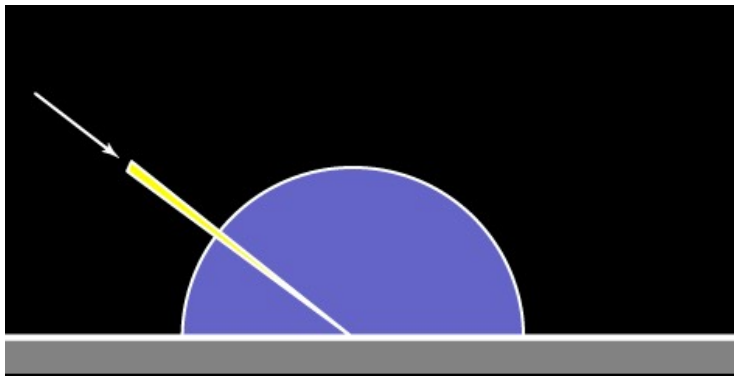


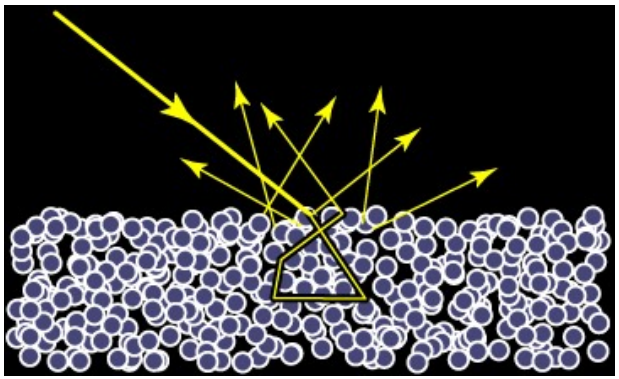


Ideal reflection

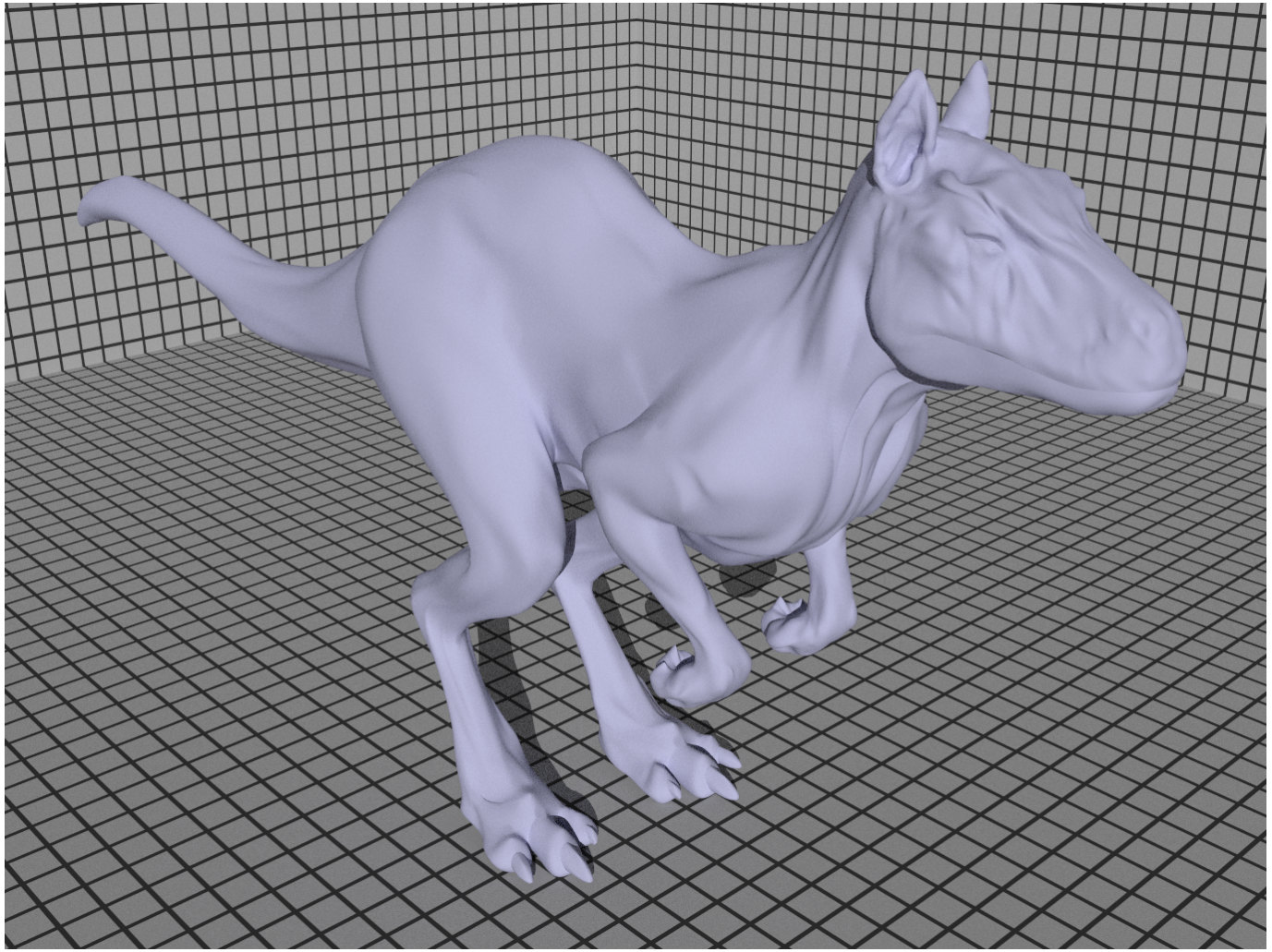
- Ideal reflection from a completely matte surface
- Light is scattered in all direction uniformly
- BRDF is a constant
- For energy conservation, k_d is in [0,1] and normalized by *pi*

$$f_d(\mathbf{i}, \mathbf{o}) = \frac{k_d}{\pi}$$





Ideal diffuse



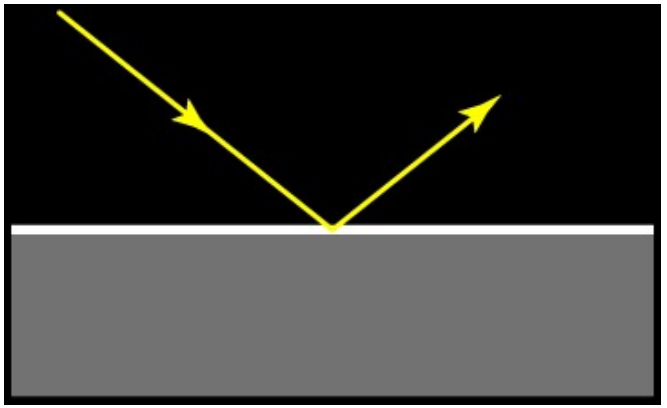
[Pharr et al./pbrt]

Ideal reflection

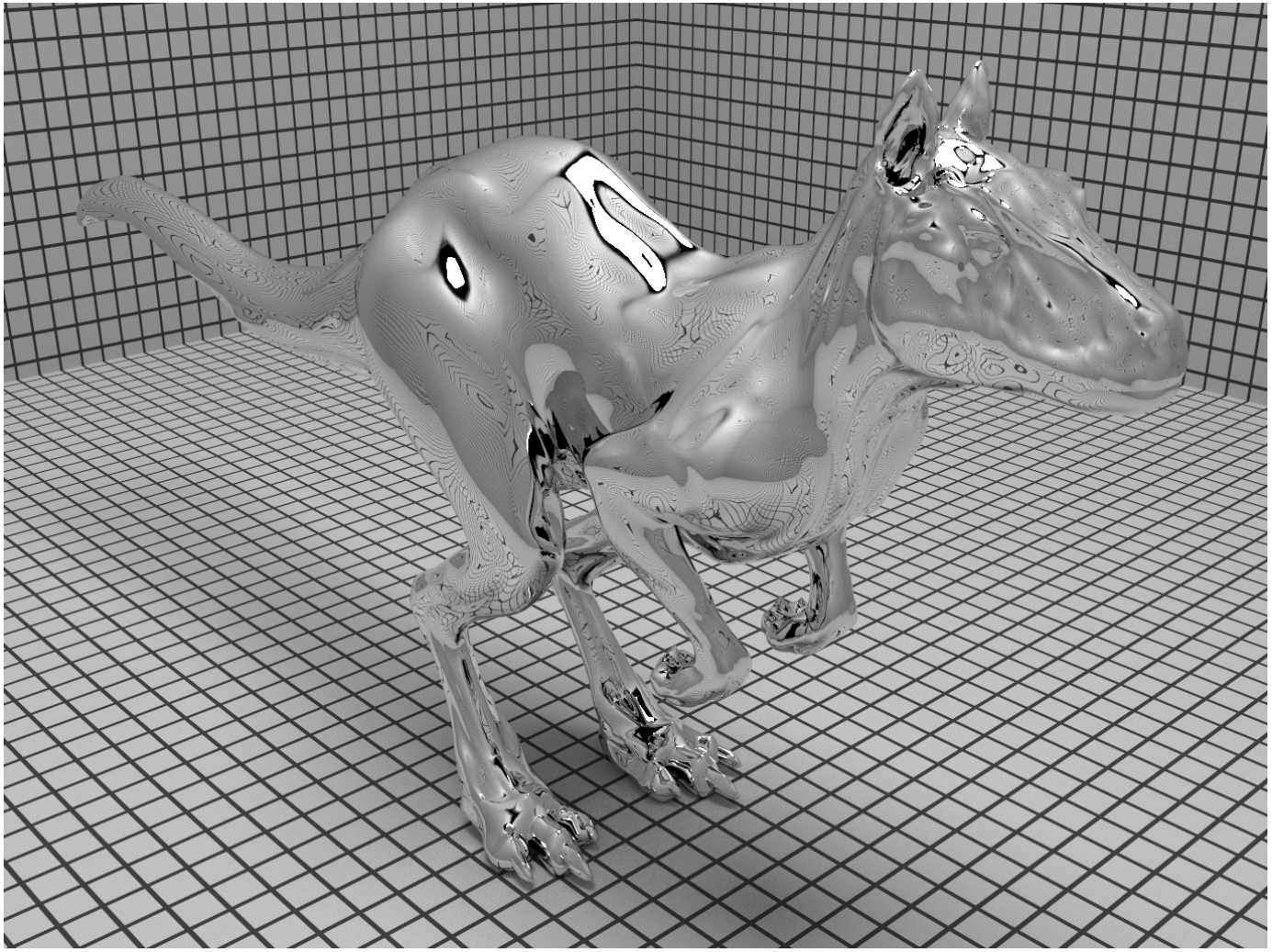
- Ideal surface with no imperfection made of an ideal conductor
- Light is scattered entirely along the mirror direction
- BRDF is a "delta function" — informally, zero everywhere except at one point, and with finite integral

$$f_r(\mathbf{i},\mathbf{o}) = k_r \frac{\delta_{\omega_\mathbf{o}}(\mathbf{r},\mathbf{o})}{|\mathbf{n}\cdot\mathbf{i}|} \qquad \mathbf{r} = -\mathbf{i} + 2(\mathbf{i}\cdot\mathbf{n})\mathbf{n}$$

$$\int_{H^2} g(\mathbf{o})\delta_{\omega_\mathbf{o}}(\mathbf{d},\mathbf{o})d\omega_\mathbf{o} = g(\mathbf{d})$$



Ideal reflection



[Pharr et al./pbrt]

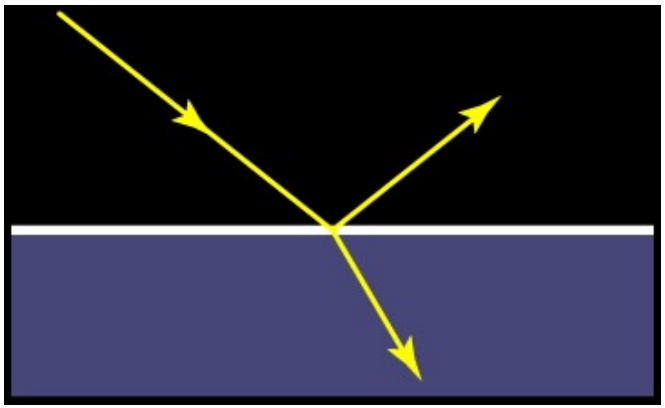
Ideal refraction

- Ideal surface made of a dialectic has both reflection and refraction
- Light is scattered in two directions, above and below the surface
- BRDF is a "delta function" around the refracted direction

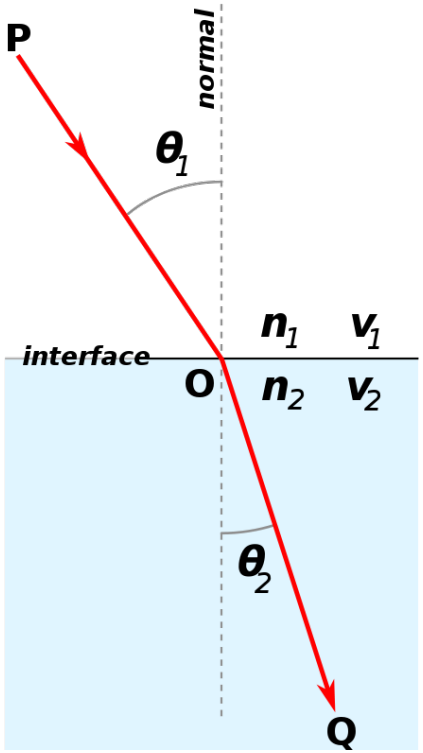
$$f_t(\mathbf{i},\mathbf{o}) = k_t \frac{1}{\eta^2} \frac{\delta_{\omega_\mathbf{o}}(\mathbf{t},\mathbf{o})}{|\mathbf{n}\cdot\mathbf{i}|} \qquad \mathbf{t} = -\eta\mathbf{i} + \left[\eta(\mathbf{n}\cdot\mathbf{i}) - \sqrt{1-\eta^2(1-(\mathbf{n}\cdot\mathbf{i})^2)}\right]\mathbf{n}$$

$$\eta = \frac{\eta_i}{\eta_o}$$

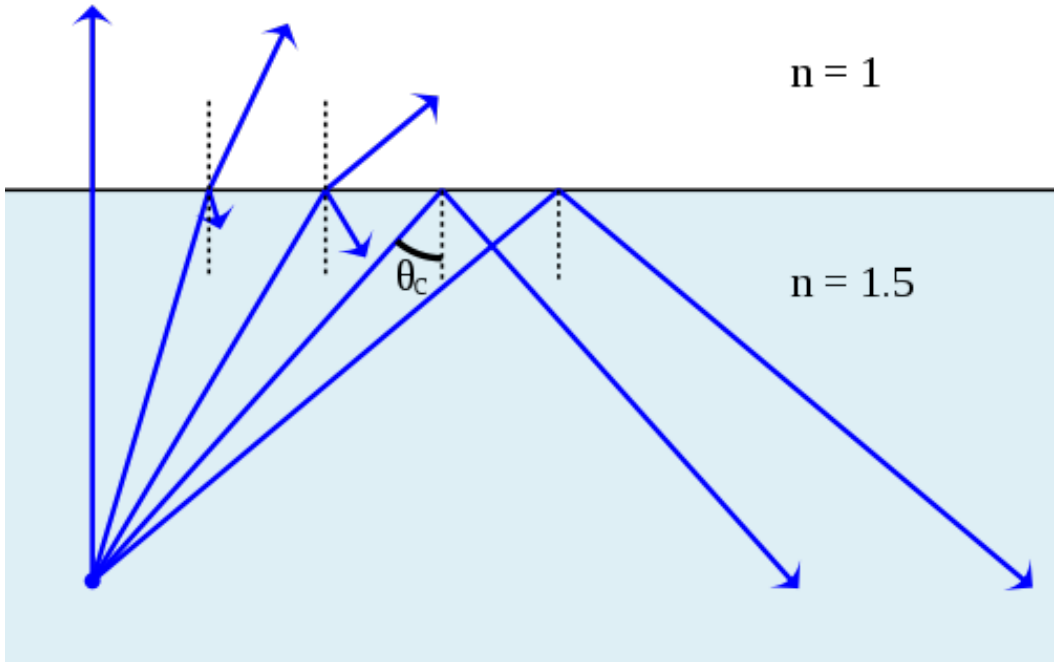
$$\int_{H^2} g(\mathbf{o})\delta_{\omega_\mathbf{o}}(\mathbf{d},\mathbf{o})d\omega_\mathbf{o} = g(\mathbf{d})$$



Ideal refraction

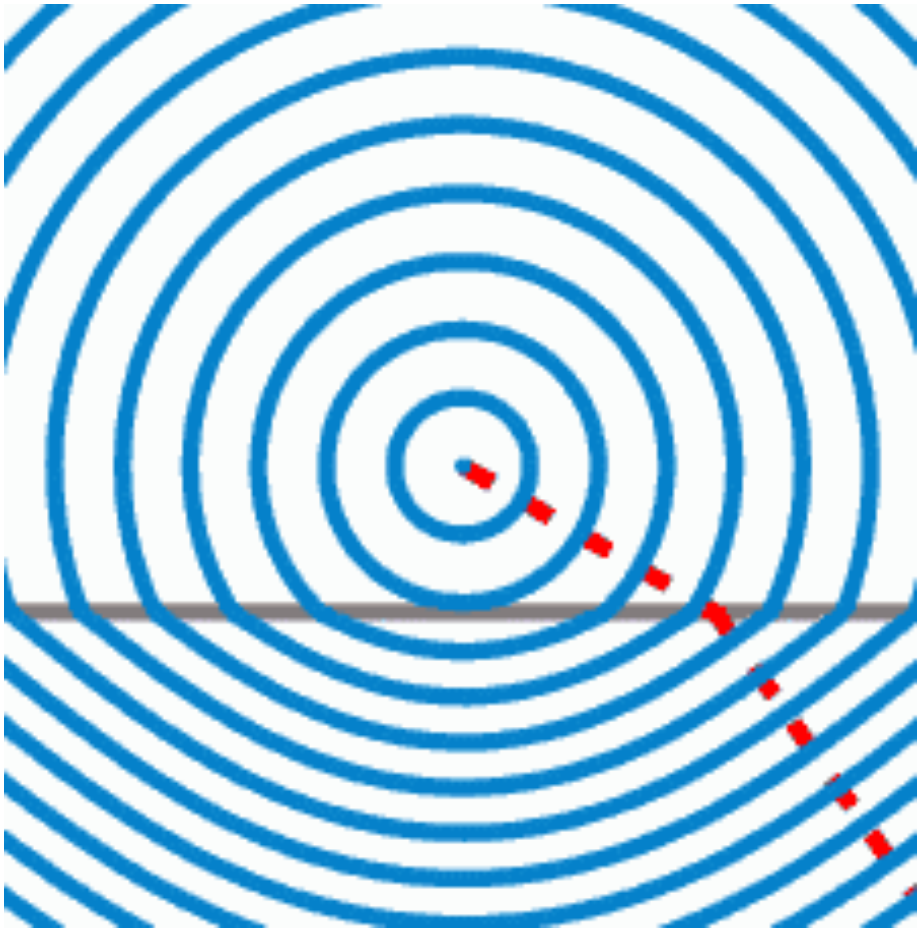


$$\mathbf{t} = -\eta\mathbf{i} + \left[\eta(\mathbf{n}\cdot\mathbf{i}) - \sqrt{1-\eta^2(1-(\mathbf{n}\cdot\mathbf{i})^2)}\right]\mathbf{n}$$



[Wikipedia]

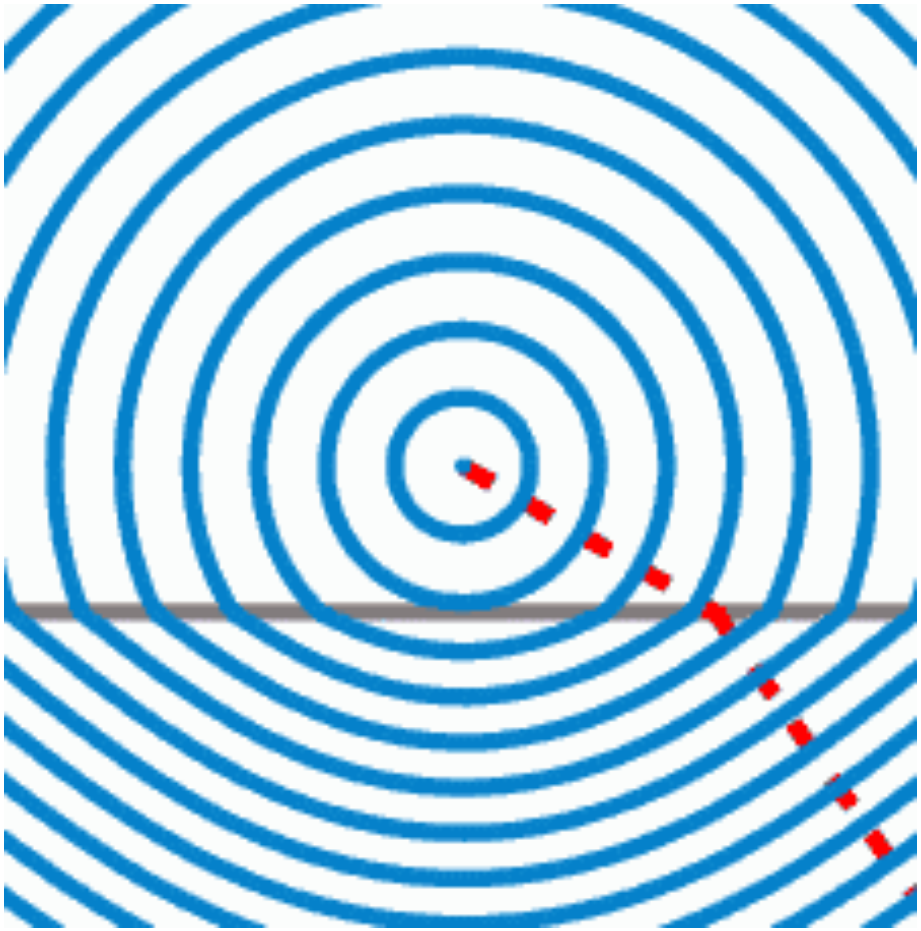
Ideal refraction



$$f_t(\mathbf{i},\mathbf{o}) = k_t \frac{1}{\eta^2} \frac{\delta_{\omega_\mathbf{o}}(\mathbf{t},\mathbf{o})}{|\mathbf{n}\cdot\mathbf{i}|}$$

[Wikipedia]

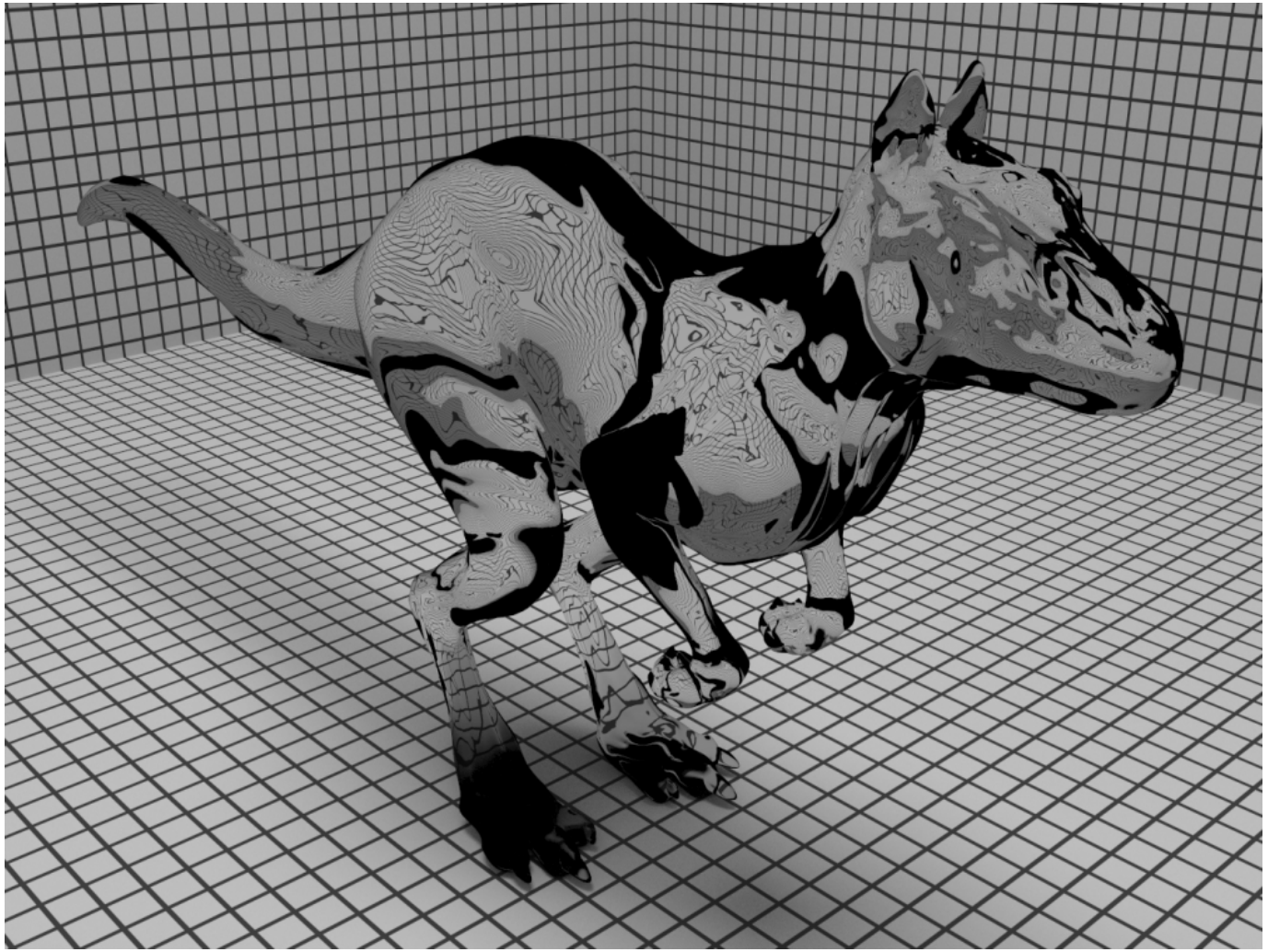
Ideal refraction



$$f_t(\mathbf{i},\mathbf{o}) = k_t \frac{1}{\eta^2} \frac{\delta_{\omega_\mathbf{o}}(\mathbf{t},\mathbf{o})}{|\mathbf{n}\cdot\mathbf{i}|}$$

[Wikipedia]

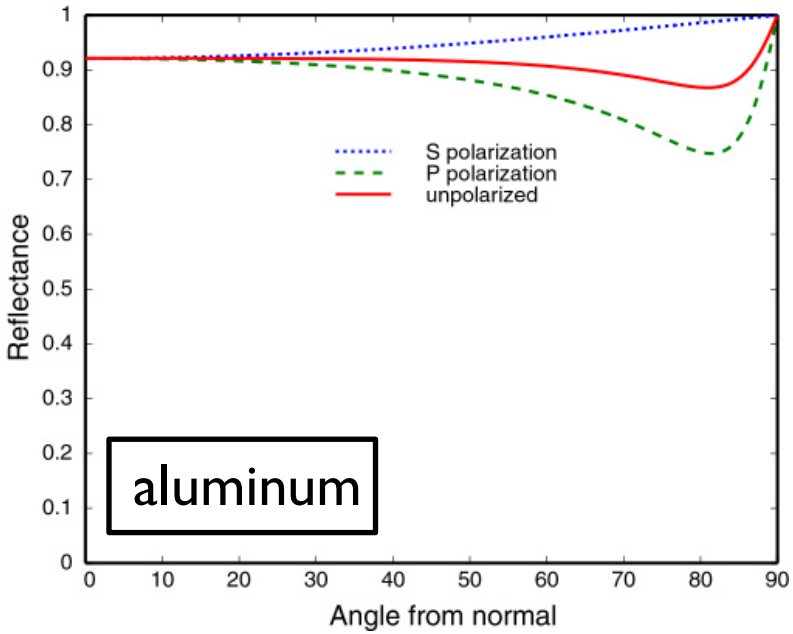
Ideal refraction

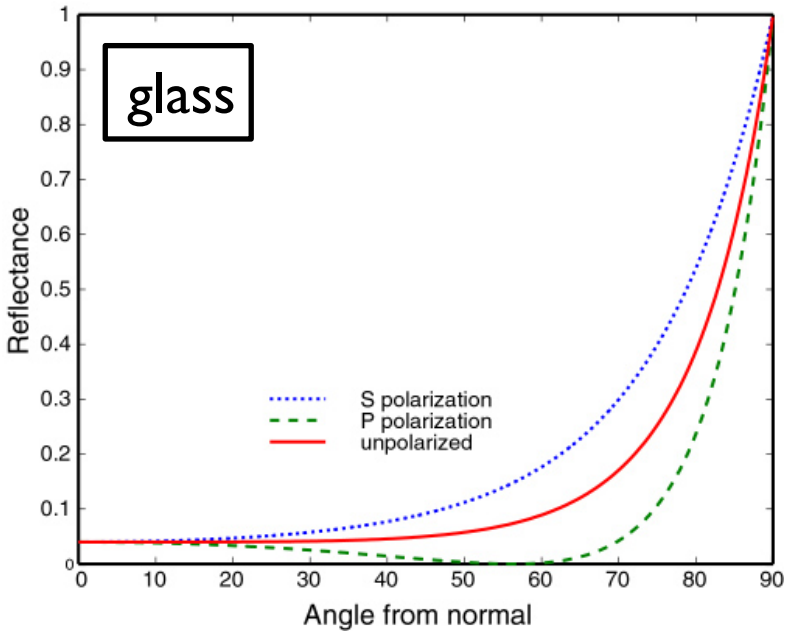


[Pharr et al./pbrt]

Fresnel reflection and transmission

- Fresnel term: determines k_r and k_t from intrinsic material properties
- Conductors: small variation from surface color to 1
- Dielectrics: drastic change from small values (~0.04) to 1
- Depends on wavelength





[Westin]

Fresnel reflection and transmission

- Full definition differ from conductors and dielectrics
- Difficult to find values for real surfaces, beside pure materials
- In graphics, we commonly use Schlick's approximation

$$k_r = k_s^0 + (1 - k_s^0)(1 - |\mathbf{n} \cdot \mathbf{i}|)^5 \qquad k_t = 1 - k_r$$

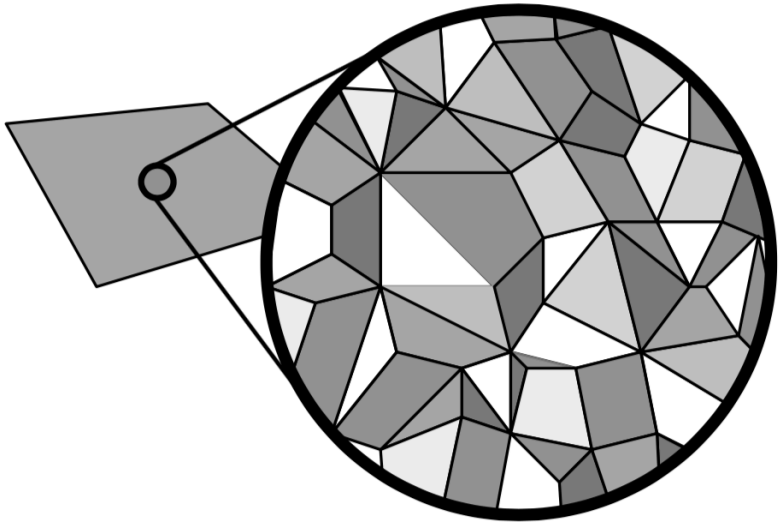
- Defined as is the reflection at normal incidence since easy to measure
  - dielectrics: 0.04-1.5; most commonly 0.04
  - conductors: surface color

Fresnel term

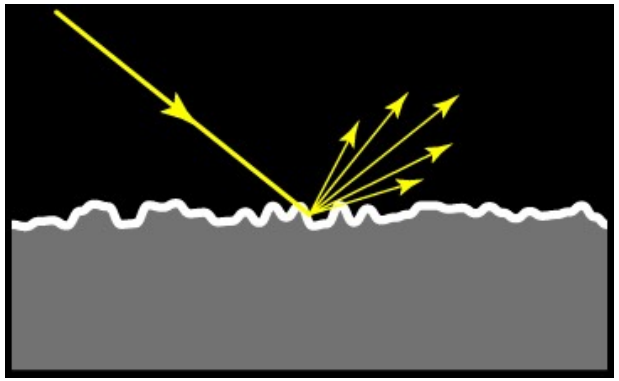
[Wikipedia]

Microfacet reflection

- Surface modeled as a collection of randomly oriented microfacet
- When light hits a surface, it hits one microfacet
- BRDF is the "average" value of all these reflection events
  - reflection centered around the bisector h



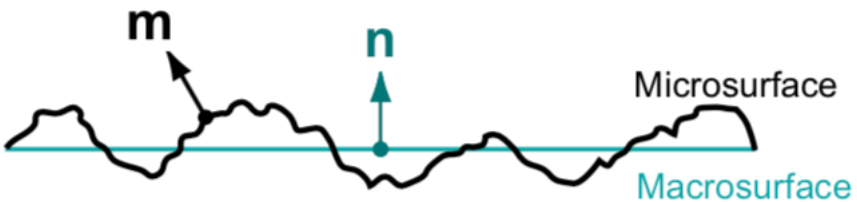
[Westin]

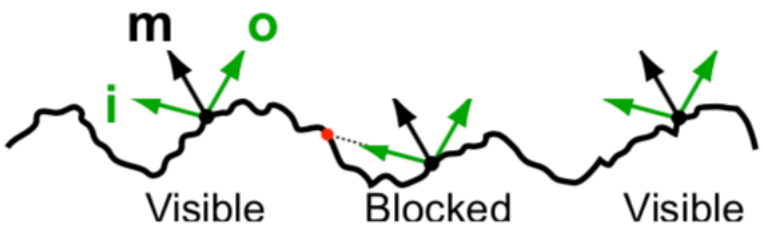


Microfacet reflection

- Normalized product of three terms: fresnel *F*, microfacet distribution *D* and shadow-masking term *G*
- Fresnel *F* accounts for difference at grazing angle
- Distribution *D* describes the statistical distribution of microfacets
- Shadow-making *G* corrects for facets not visible from light or view

$$f_r(\mathbf{i},\mathbf{o}) = \frac{F(\mathbf{i},\mathbf{h})D(\mathbf{h})G(\mathbf{h},\mathbf{i},\mathbf{o})}{4|\mathbf{n}\cdot\mathbf{i}||\mathbf{n}\cdot\mathbf{o}|} \qquad \mathbf{h} = \frac{\mathbf{i}+\mathbf{o}}{|\mathbf{i}+\mathbf{o}|}$$





[Walter et al.]

Microfacet reflection

- Different choices for *D*, mostly quite similar
- Blinn-Phong: simple, but lacks physical basis

$$D_p(\mathbf{h}, n) = \frac{n+2}{2\pi}(\mathbf{n}\cdot\mathbf{h})^n \qquad D_p(\mathbf{h}, \alpha) = \frac{1}{\pi\alpha^2}(\mathbf{n}\cdot\mathbf{h})^{\frac{2}{\alpha^2}-2}$$

- Beckman: first distribution used in graphics

$$D_b(\mathbf{h}, \alpha) = \frac{1}{\pi\alpha^2(\mathbf{n}\cdot\mathbf{h})^4}\exp\left(\frac{(\mathbf{n}\cdot\mathbf{h})^2 - 1}{\alpha^2(\mathbf{n}\cdot\mathbf{h})^2}\right)$$

- GGX: currently the most used one – slightly better fit to real data

$$D_g(\mathbf{h}, \alpha) = \frac{\alpha^2}{\pi((\mathbf{n}\cdot\mathbf{h})^2(\alpha^2 - 1) + 1)^2}$$

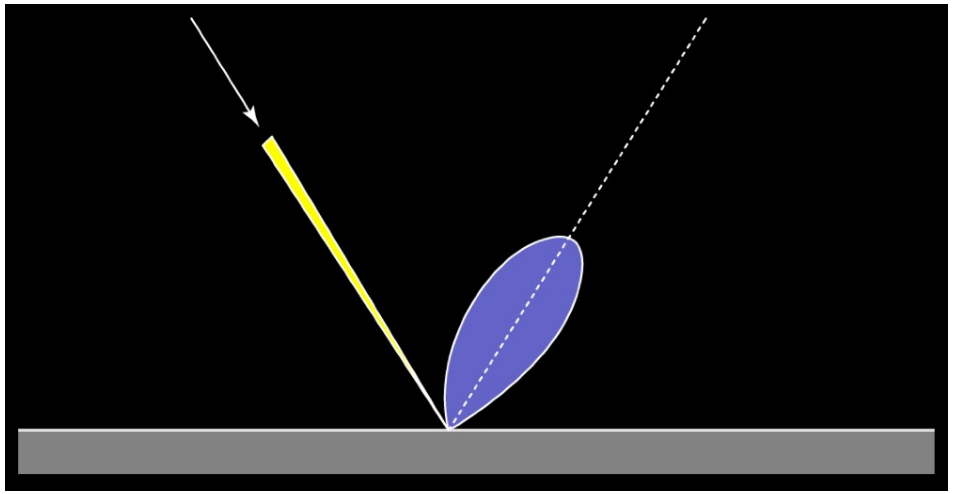
Microfacet reflection

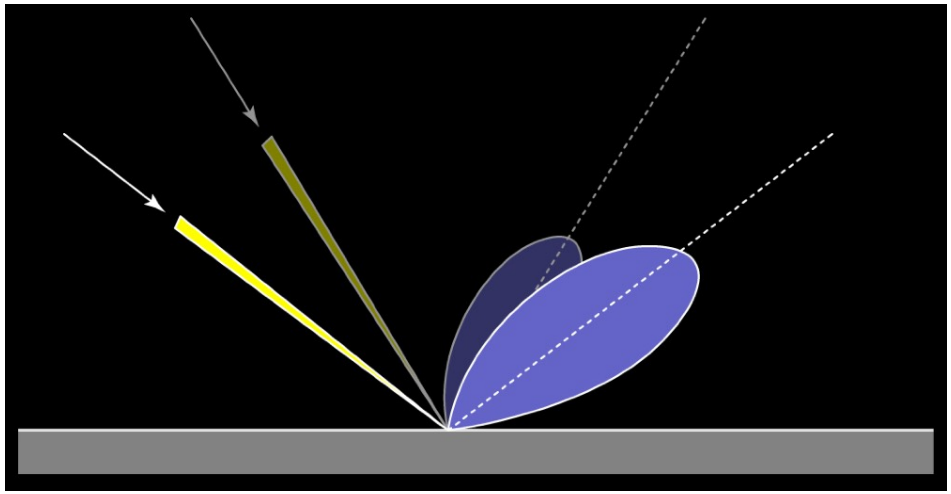
- Different choices for *G*, quite a bit different – no clear winner
- Cook-Torrance: first shadow-masking used in graphics

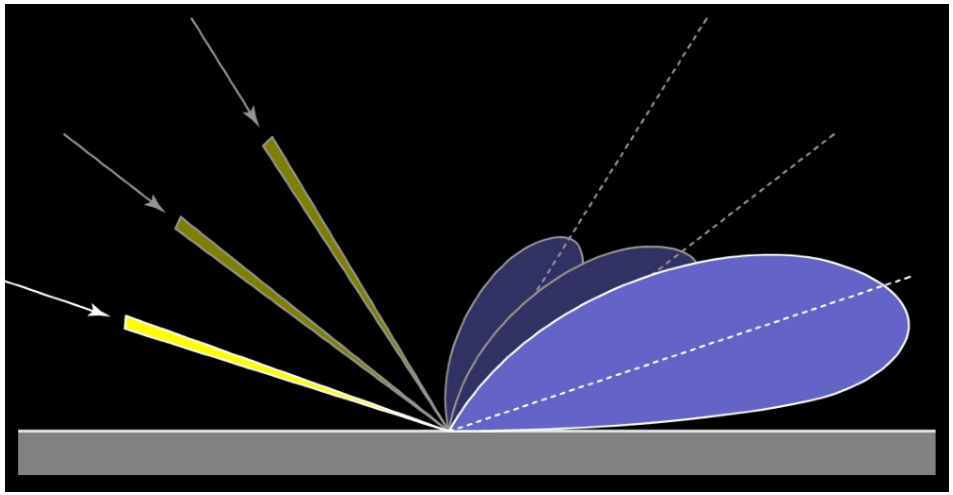
$$G(\mathbf{h},\mathbf{i},\mathbf{o}) = \min\left\{1, \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{o})}{\mathbf{h}\cdot\mathbf{o}}, \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{i})}{\mathbf{h}\cdot\mathbf{i}}\right\}$$

- Smith: more physically-sound – depends on *D*
- *T*ogether with normalization, this is responsible for better behavior at grazing angles

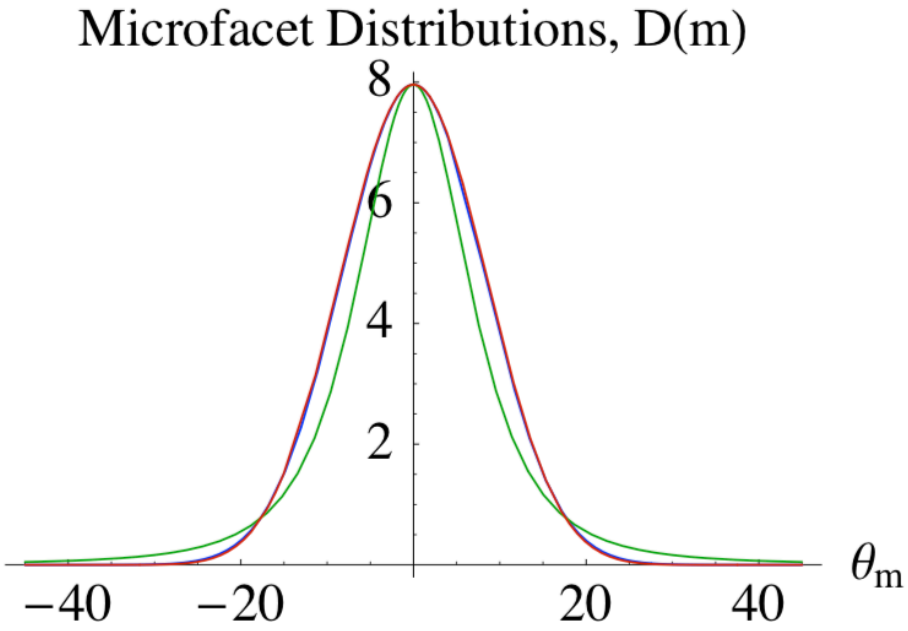
Microfacet reflection

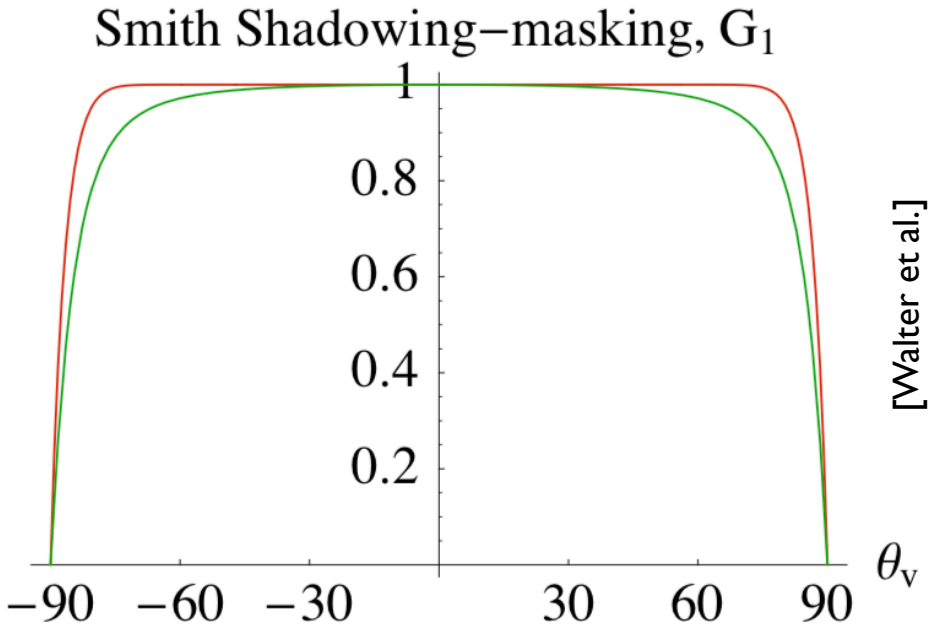






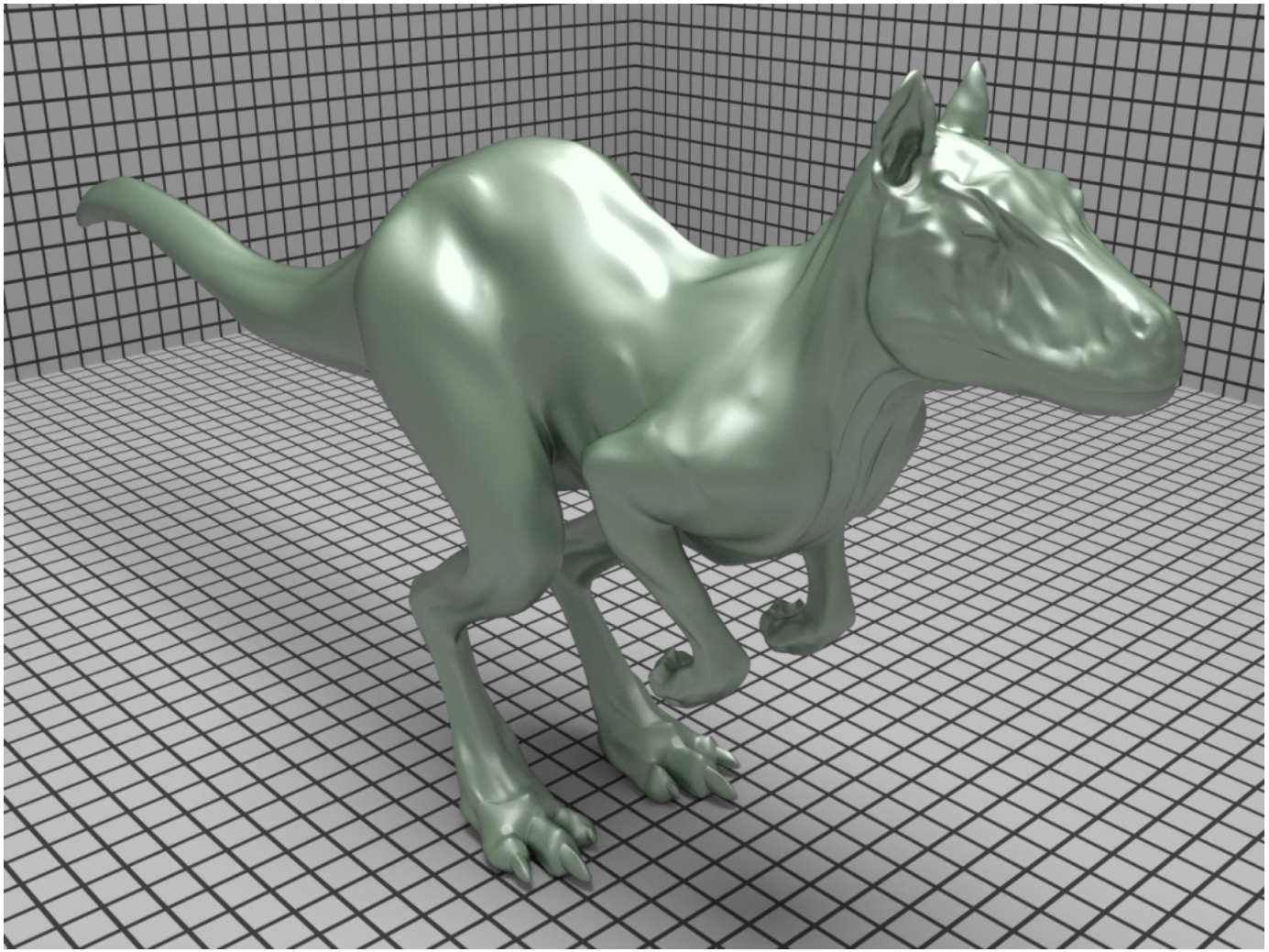
Microfacet reflection





[Walter et al.]

Microfacet reflection



[Pharr et al./pbrt]

Microfacet transmission

- Derived from previous ideas while correctly accounting for change of variable when moving below the surface

$$f_t(\mathbf{i},\mathbf{o}) = \frac{|\mathbf{h}\cdot\mathbf{i}||\mathbf{h}\cdot\mathbf{o}|}{|\mathbf{n}\cdot\mathbf{i}||\mathbf{n}\cdot\mathbf{o}|}\frac{\eta_o^2(1-F(\mathbf{i},\mathbf{h}))D(\mathbf{h})G(\mathbf{h},\mathbf{i},\mathbf{o})}{(\eta_i(\mathbf{h}\cdot\mathbf{i})+\eta_o(\mathbf{h}\cdot\mathbf{o}))^2}$$

Generic materials

- These scattering behaviours can be combined together to *fit* data from real-world surfaces
- Polished/rough glass: combine reflection and transmission terms
- Many opaque objects: weighted sum of diffuse and specular reflections

$$f(\mathbf{i},\mathbf{o}) = \frac{k_d}{\pi} + \frac{k_s D(\mathbf{h}) G(\mathbf{h},\mathbf{i},\mathbf{o})}{4|\mathbf{n}\cdot\mathbf{i}||\mathbf{n}\cdot\mathbf{o}|}$$

- Accounting for Fresnel still an open problem here, but we can use a simple solution that conserve energy

$$k_s = F(k_s^0, \mathbf{i}, \mathbf{h}) \quad k_d = k_d^0 + (k_s^0 - F(k_s^0, \mathbf{i}, \mathbf{h}))$$

Sampling BSDF-cosine

- When picking directions for hemispherical sampling, we want to sample a function that is close to the integrand
- Often, best we can do is to sample the BSDF scaled by the cosine
- For mirror reflection and transmission, we only have one direction
- For diffuse reflection, we use the cosine sampling presented previously
- For convenience, we can rewrite that sampling in spherical coordinates

$$\mathbf{i}^l = (\phi, \theta) = (2\pi r_1, \mathrm{acos}(r_2))$$

$$p(\mathbf{i}^l) = \frac{\mathbf{i}^l_z}{\pi}$$

Sampling BSDF-cosine

- For microfacet BRDFs, we sample the *D* term since this is dominant
- We proceed by picking a half-vector **h** according to *D* as

$$\mathbf{h}_p^l = \left(2\pi r_1, \operatorname{acos}\left(r_2^{\alpha/2}\right)\right)$$

$$\mathbf{h}_b^l = \left(2\pi r_1, \operatorname{atan}\left(\sqrt{-\alpha^2 \log(1-r_2)}\right)\right)$$

$$\mathbf{h}_g^l = \left(2\pi r_1, \operatorname{atan}\left(\frac{\alpha\sqrt{r_2}}{\sqrt{1-r_2}}\right)\right)$$

$$p(\mathbf{h}) = D(\mathbf{h})|\mathbf{n}\cdot\mathbf{h}|$$

Sampling BSDF-cosine

- For reflection, we derive the incoming angle from the half-vector and correct the pdf for the change of variable as

$$\mathbf{i} = -\mathbf{o} + (\mathbf{h} \cdot \mathbf{o})\mathbf{h} \qquad p(\mathbf{i}) = \frac{D(\mathbf{h})|\mathbf{n} \cdot \mathbf{h}|}{4|\mathbf{h} \cdot \mathbf{i}|}$$

- For refraction, the incoming angle and pdf are written as

$$\mathbf{i} = -\eta\mathbf{o}\left(\eta(\mathbf{o} \cdot \mathbf{h}) - \sqrt{1 + \eta((\mathbf{o} \cdot \mathbf{h})^2 - 1)}\right)\mathbf{h}$$

$$p(\mathbf{i}) = \frac{D(\mathbf{h})|\mathbf{n} \cdot \mathbf{h}|}{4|\mathbf{h} \cdot \mathbf{i}|} \frac{\eta_o^2|\mathbf{o} \cdot \mathbf{h}|}{(\eta_i(\mathbf{i} \cdot \mathbf{h}) + \eta_o(\mathbf{o} \cdot \mathbf{h}))}$$

Sampling BSDF-cosine

- For BSDF that are sums of lobes, we use a third random number to choose the lobe just like a discreet distribution

$$f = k_1 f_1 + k_2 f_2$$

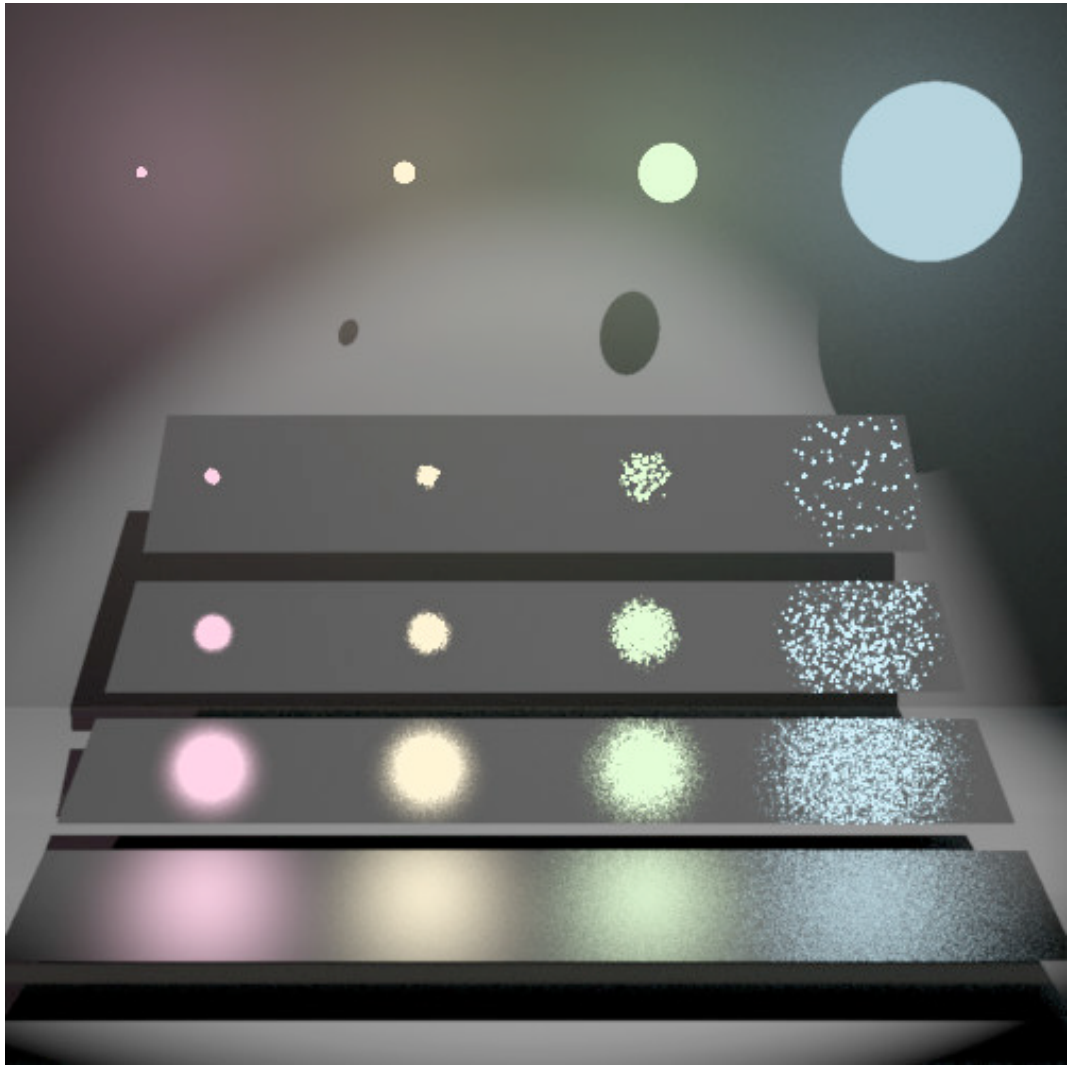
$$\mathbf{i}_f = \begin{cases} \mathbf{i}_{f_1} & r_3 < w_1 \\ \mathbf{i}_{f_2} & \text{otherwise} \end{cases}$$

$$w_1 = \frac{\max(k_1)}{\max(k_1) + \max(k_2)}$$

$$p(\mathbf{i}) = w_1 p(\mathbf{i}_{f_1}) + (1 - w_1) p(\mathbf{i}_{f_2})$$

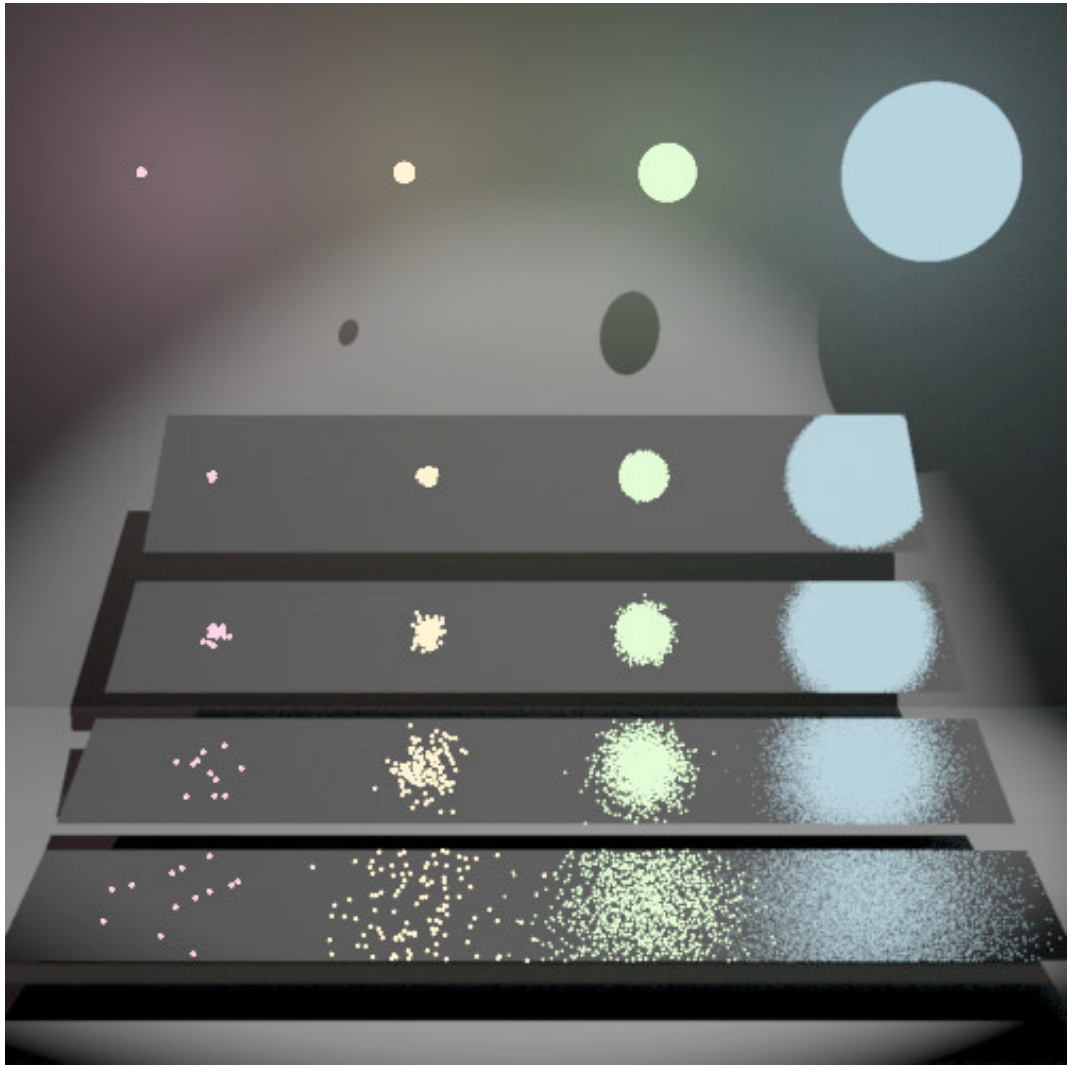
Multiple Importance Sampling

Direct by sampling light



[Veach]

Direct by sampling BSDF



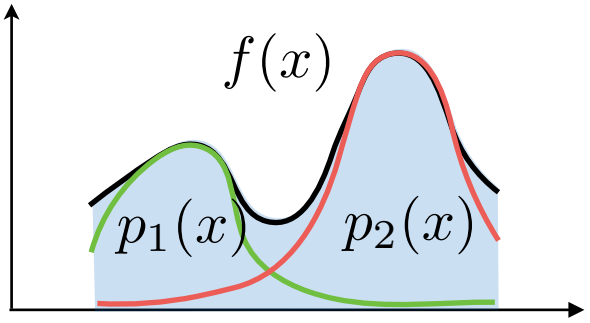
[Veach]

Direct by sampling both

[Veach]

Multiple importance sampling

- We want to sample a complex function with importance sampling, but we do not know hoe to sample it well over the entire domain
- But we can sample different parts of the domain well with different sampling techniques
- We want to combine all techniques into a single robust one
- *Multiple importance sampling*: pick a technique at random and sample according to that that one
- Insight: avoid issues when a technique would not sample well a region



Multiple importance sampling

- Consider *n* sampling techniques with pdfs *pi*
- Pick N_i samples from each technique
- The combined estimator for these samples is

$$E[I] = \sum_i^n \frac{1}{N_i} \sum_{j=1}^{N_i} w_i(\mathbf{x}_{i,j}) \frac{f(\mathbf{x}_{i,j})}{p_i(\mathbf{x}_{i,j})} \qquad \sum_i w_i(\mathbf{x}) = 1$$

- Choose weights w_i to reduce variance
- Balance heuristic: almost optimal weights

$$w_i(\mathbf{x}) = \frac{N_i p_i(\mathbf{x})}{\sum_k N_k p_k(\mathbf{x})}$$

- Other heuristic possible (power, maximum, etc.)

Multiple importance sampling

- In the context of path tracing, use this to estimate the direct illumination according to both light and BSDF
- Draw a sample from each distribution and combine the samples
- When combining samples, remember to convert the light pdf into an hemispherical measure

$$L_d(\mathbf{x},\mathbf{o}) \approx \frac{L_e(\mathbf{r}(\mathbf{x},\mathbf{i}_b),-\mathbf{i}_b) f(\mathbf{x},\mathbf{i}_b,\mathbf{o}) |\mathbf{n}_\mathbf{x}\cdot\mathbf{i}_b|}{p_b(\mathbf{i}_b)+p_l(\mathbf{i}_b)} + \frac{L_e(\mathbf{y}_l,-\mathbf{i}_l) f(\mathbf{x},\mathbf{i}_l,\mathbf{o}) |\mathbf{n}_\mathbf{x}\cdot\mathbf{i}_l| V(\mathbf{y}_l,\mathbf{x})}{p_b(\mathbf{i}_l)+p_l(\mathbf{i}_l)}$$

Path tracing – MIS

```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    auto li = le, w = {1,1,1};
    for(auto bounce : range(max_bounces)) {
        if(!pt.f) break;
        auto lpt_l= sample_lights(scn->lights, pt);
        if(!intersect(scn, {pt.x, -lpt_l.o}))
            li += w * lpt_l.le * eval_brdfcos(pt,-lpt_l.o) /
                  (pdf_l(lpt_l)*pdf_b(-lpt_l.o));
        auto lpt_b = intersect(scn, {pt.o, sample_brdfcos(pt)});
        li += w * lpt_b.le * eval_brdfcos(pt,-lpt_b.o) /
                  (pdf_l(lpt_b)*pdf_b(-lpt_b.o));
        if(randf() > pr(f)) break; // russian roulette
        auto bpt = intersect(scn, {pt.x, sample_brdfcos(pt)});
        w *= eval_brdfcos(pt,i) / (pr(pt.f)*pdf(i)); // update w
        // do not accumulate emission since done before
        pt = bpt;                       // "recurse"
    }
    return li;
}
```

DIRECT MIS

INDIRECT

Path tracing – MIS with reuse

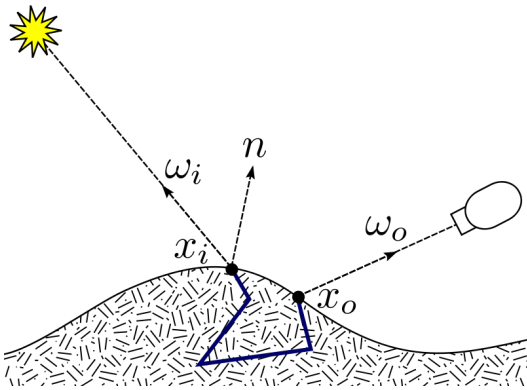
```
vec3f estimate_li(scene* scn, vec3f q, vec3f i) {
    auto pt = intersect(scn, {q, i});
    auto li = le, w = {1,1,1};
    for(auto bounce : range(max_bounces)) {
        if(!pt.f) break;
        auto lpt= sample_lights(scn->lights, pt);
        if(!intersect(scn, {pt.x, -lpt.o}))
            li += w * lpt.le * eval_brdfcos(pt,-lpt.o) /
                  (pdf_l(lpt)*pdf_b(-lpt.o));
        auto bpt = intersect(scn, {pt.x, sample_brdfcos(pt)});
        li += w * bpt.le * eval_brdfcos(pt,-bpt.o) /
                  (pdf_l(bpt)*pdf_b(-bpt.o));
        if(randf() > pr(f)) break; // russian roulette
        w *= eval_brdfcos(pt,i) / (pr(pt.f)*pdf(i)); // update w
        pt = bpt;                    // “recurse”
    }
    return li;
}
```


Advanced Rendering

Subsurface scattering: BSSRDF

- BSSRDF [$m^{-2}sr^{-1}$]: *bidirectional scattering surface-reflectance distr. func.*
- Defined as the ratio of the differential reflected radiance over the differential incident flux
- Depends on incoming and outgoing directions, incoming and outgoing positions, and wavelength of light
- Main different w.r.t. BSDF: photons leave at different points then they enter, so consider two both locations in the equation

$$s(\mathbf{x_i}, \mathbf{x_o}, \mathbf{i}, \mathbf{o}) = \frac{dL_r(\mathbf{x_o}, \mathbf{o})}{dP_i(\mathbf{x_i}, \mathbf{i})} = \frac{dL_r(\mathbf{x_o}, \mathbf{o})}{L_i(\mathbf{x_i}, \mathbf{i})|\mathbf{i} \cdot \mathbf{n_i}| d\omega_\mathbf{i} dA_{\mathbf{x_i}}}$$



Subsurface scattering: BSSRDF

- Reflected radiance can be obtained by integrating the above equation over all points on the surface and for each incoming direction around each point

$$L_r(\mathbf{x_o}, \mathbf{o}) = \int_{\mathbf{x_i} \in S} \int_{\mathbf{i} \in H^2} L_i(\mathbf{x_i}, \mathbf{i}) s(\mathbf{x_i}, \mathbf{x_o}, \mathbf{i}, \mathbf{o}) |\mathbf{i} \cdot \mathbf{n_i}| d\omega_\mathbf{i} dA_{\mathbf{x_i}}$$

Properties of the BSSRDF

- Same properties of BSSRDF
- Positivity: since photons cannot be "negatively" reflected

$$s(\mathbf{y}, \mathbf{x}, \mathbf{i}, \mathbf{o}) \geq 0$$

- Helmotz reciprocity: can invert light paths

$$s(\mathbf{y}, \mathbf{x}, \mathbf{i}, \mathbf{o}) = s(\mathbf{x}, \mathbf{y}, \mathbf{o}, \mathbf{i})$$

- Energy conservation: all photons that comes in are either reflected or absorbed and no new photons is emitted

$$\int_S \int_{H^2} s(\mathbf{y}, \mathbf{x}, \mathbf{i}, \mathbf{o}) |\mathbf{i} \cdot \mathbf{o}| d\omega_{\mathbf{i}} dA_{\mathbf{y}} \leq 1$$

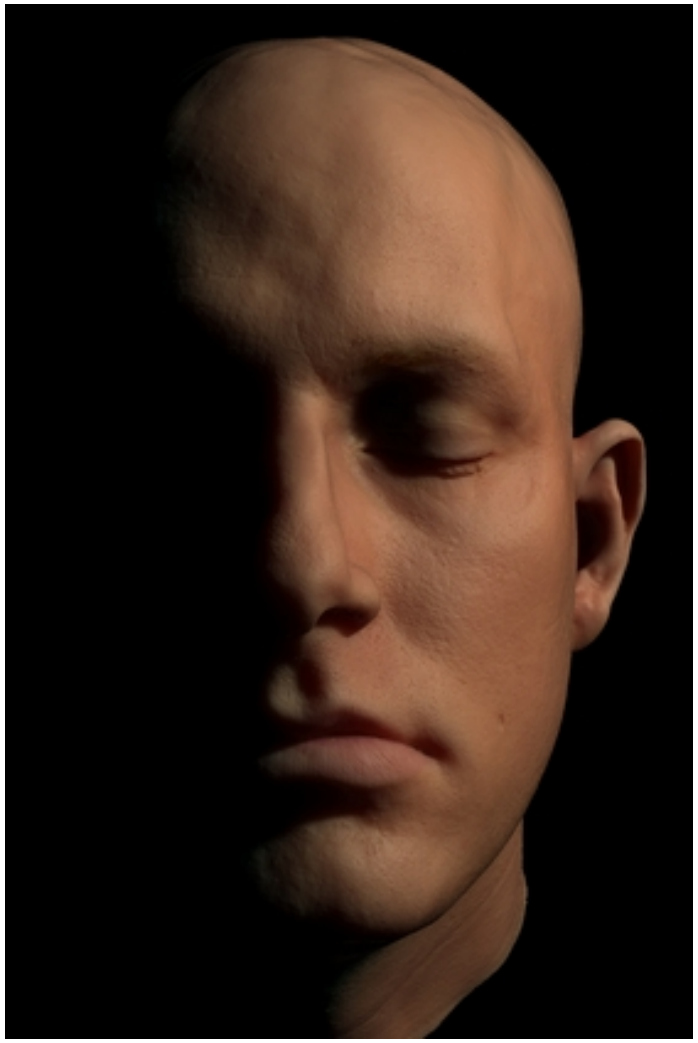
Rendering with BSSRDF

BRDF

BSSRDF

[Jensen et al]

Rendering with BSSRDF



[Conner et al]

[Jensen et al]

Volumetric scattering: phase func.

- When rendering natural effects such as smoke or dust, scattering does not happen at a particular surface but anywhere within the 3D volume
- We do not integrate over the surface but along the ray from its origin to infinity, or better from when it enters to when it leaves the medium
- *Phase function p* [$m^{-1}sr^{-1}$]: describes the scattering at a location in the volume, akin to the BRDF and BSSRDF
- Reflected radiance integrates along the ray and in the sphere of incoming directions

$$L_r(\mathbf{x}, \mathbf{o}) = \int_{t=0}^{\infty} \int_{\mathbf{i} \in H^2} L_i(\mathbf{x}(t), \mathbf{i}) p(\mathbf{x}(t), \mathbf{i}, \mathbf{o}) |\mathbf{i} \cdot \mathbf{d}| d\omega_{\mathbf{i}} dt$$

$$\text{with } \mathbf{x}(t) = \mathbf{x} + t\mathbf{d}$$

Volumetric scattering: phase func.

- Phase function is often written as product of the *normalized phase function p'* [sr^{-1}], that integrates to 1, and the *scattering coefficient* σ [m]
  - angular behavior does not change in medium, but scattering does
  - phase function depends on material type, scattering coefficient on material density

$$p(\mathbf{x}, \mathbf{i}, \mathbf{o}) = \sigma(\mathbf{x}) p'(\mathbf{i}, \mathbf{o})$$