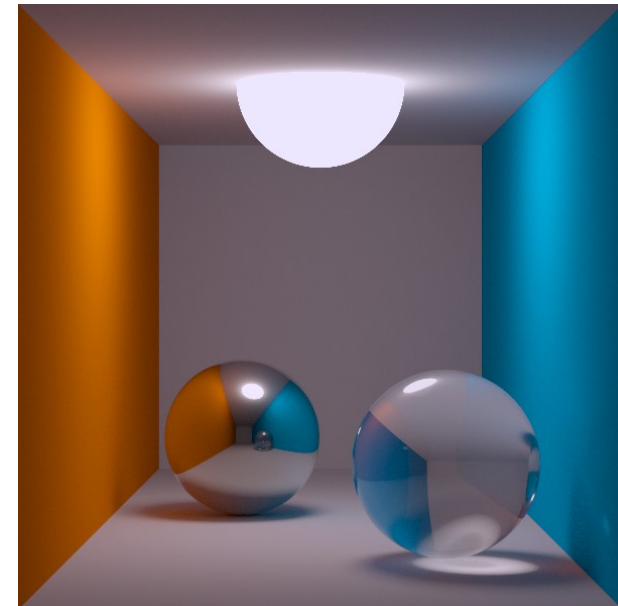
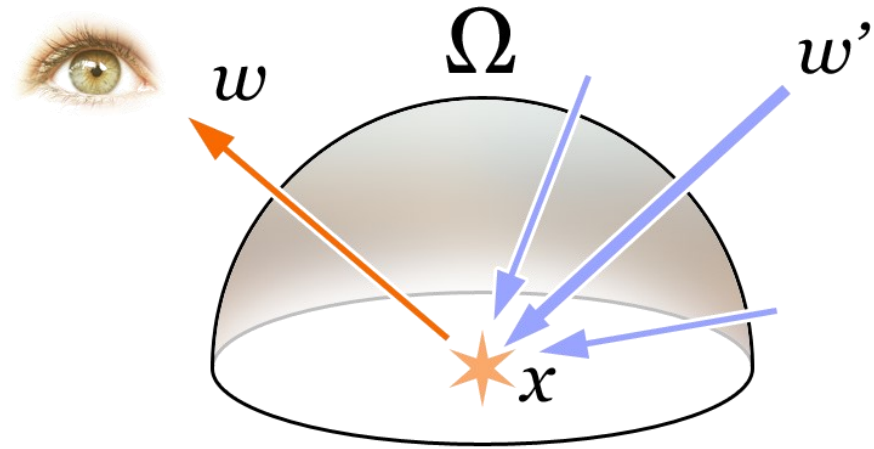


Radiometry and Monte Carlo integration

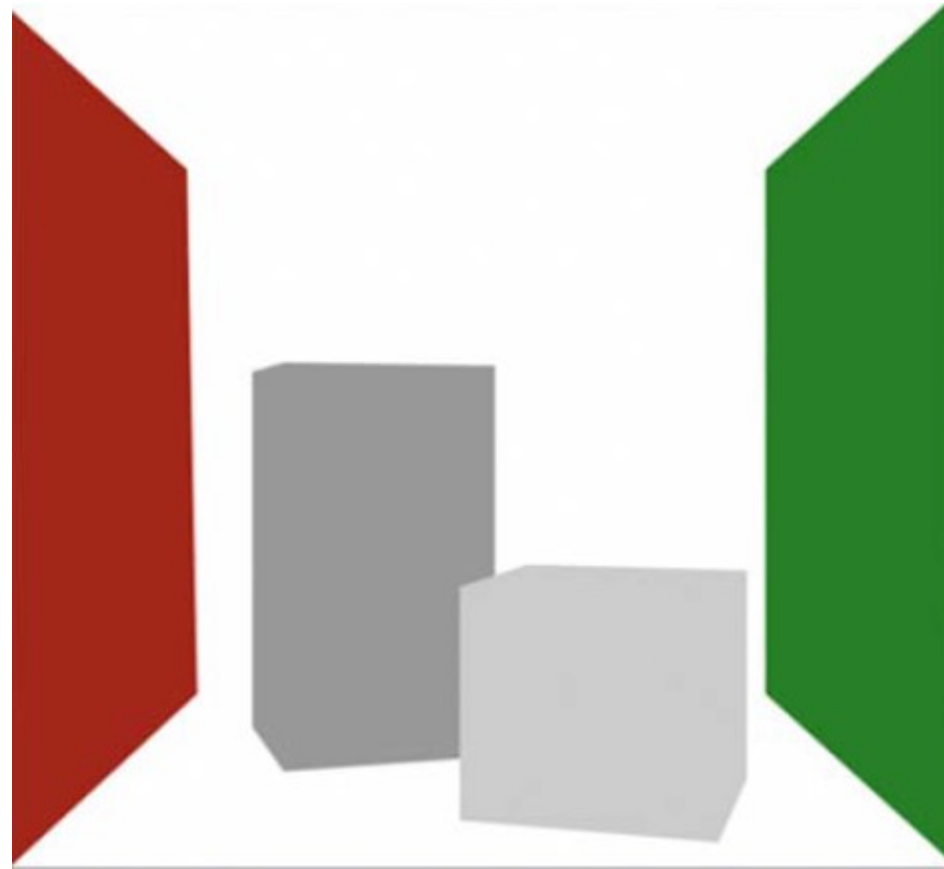
Adapted from slides by Derek Nowrouzezahrai

What will we cover?

- The theory of physically-based rendering (PBR)
 - Radiometry
 - Monte Carlo integration
 - Ambient occlusion
 - Direct illumination
 - Global illumination
- How to go from theory to simple, practical algorithms
 - You'll be able to generate images like these:

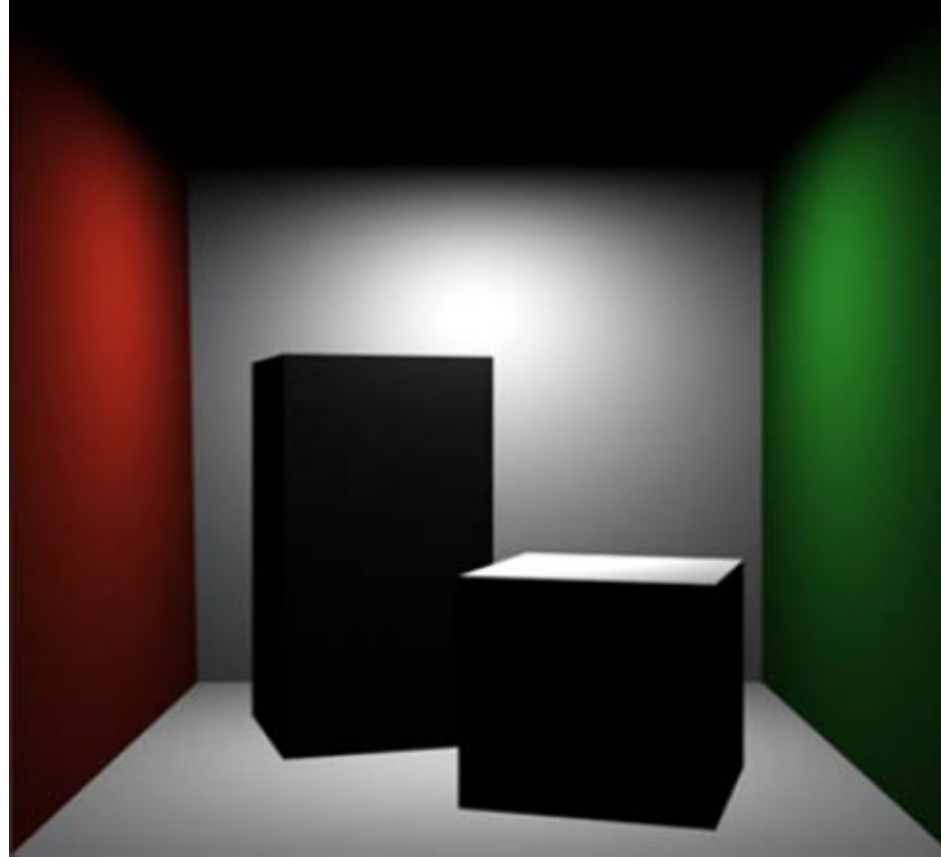


PBR: a (visual) implementation guide



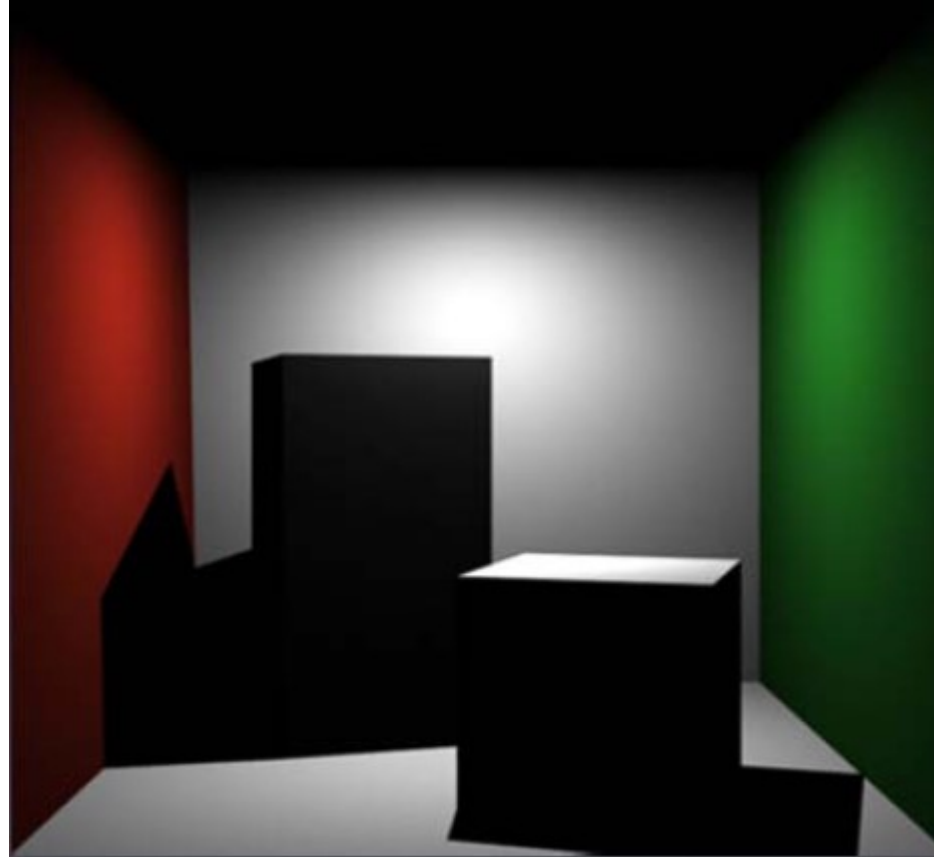
Basic raytracing framework (without shading)
[primary/eye rays]

PBR: a (visual) implementation guide



Simple shading calculations
[still just primary rays]

PBR: a (visual) implementation guide



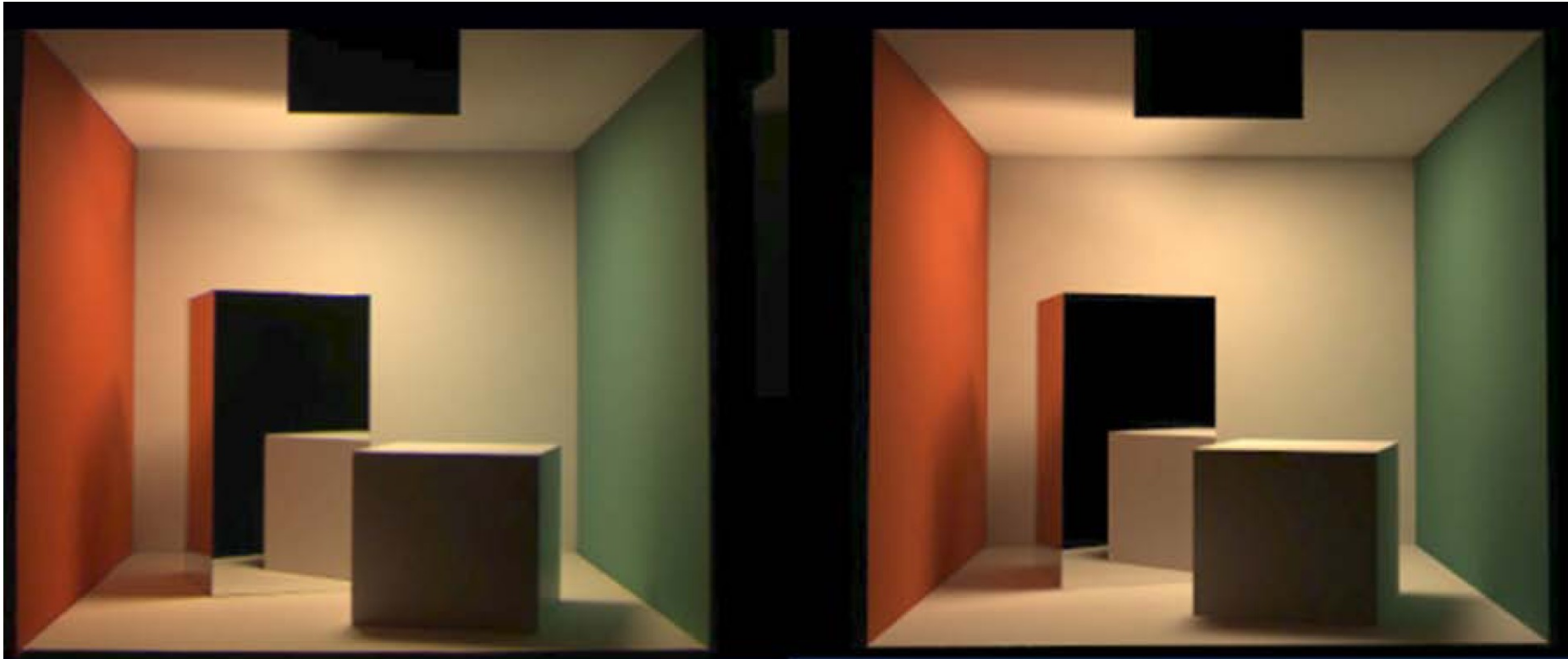
Hard shadows [(one) secondary ray]

PBR: a (visual) implementation guide



Soft shadows [**many secondary rays**]

PBR: a (visual) implementation guide



Global illumination [**recursive secondary rays**]

<http://www.graphics.cornell.edu/online/box/compare.html>

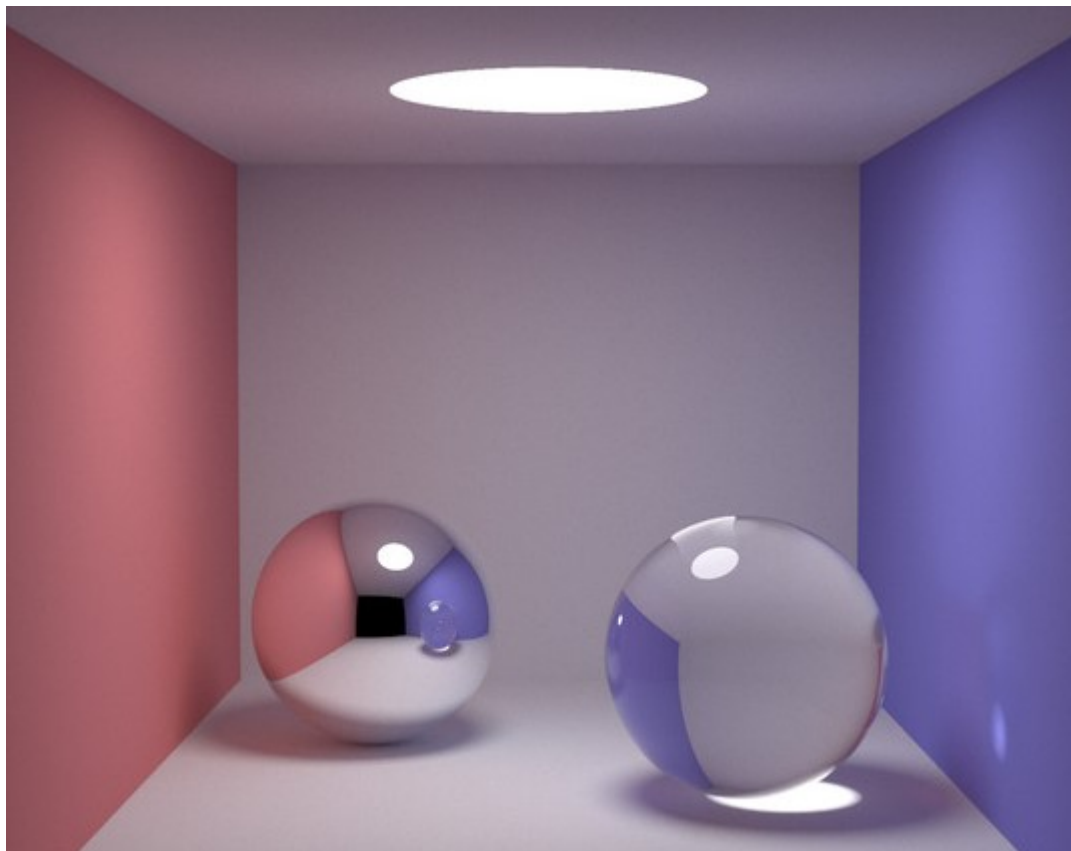
<http://area.autodesk.com/fakeorfoto> (I got 84%)

PBR: a (visual) implementation guide

- Devil's in the details:
 - Where do the secondary rays start from?
 - Where do they go to?
 - What values do they “carry” and “propagate”
- We will bridge the answers to these questions using the theory of realistic image synthesis
 - Good news: in the end, the code isn't that bad!
- For example: see smallpt (a 99 LoC path tracer!)

PBR: a (visual) implementation guide

- For example: see smallpt (a 99 LoC path tracer!)

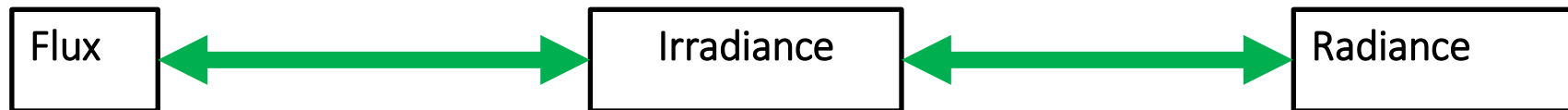


```
1. #include <math.h> // smallpt, a Path Tracer by Kevin Beason, 2008
2. #include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
3. #include <stdio.h> // Remove "-fopenmp" for g++ version < 4.2
4. struct vec { // usage: time ./smallpt 5000 44 xv image.ppm
5.     double x, y, z; // position, also color (x,g,b)
6.     Vec(double x=0, double y=0, double z=0) { x=x; y=y; z=z; }
7.     Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
8.     Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
9.     Vec operator*(double s) const { return Vec(x*s,y*s,z*s); }
10.    Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
11.    Vec norm() { return *this * (1/sqrt(x*x+y*y+z*z)); }
12.    double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
13.    Vec operator^(Vec&b) { return Vec(y*b.z-z*b.y,x*b.z-x*b.y,y*b.x-x*b.y); }
14. };
15. struct Ray { Vec o, d; Ray(Vec o, Vec d) : o(o), d(d) {} };
16. enum Refl_t { DIFF, SPEC, REFR }; // material types, used in radiance()
17. struct Sphere {
18.     double rad; // radius
19.     Vec p, e, c; // position, emission, color
20.     Refl_t refl; // reflection type (DIFFuse, SPECular, REFRactive)
21.     Sphere(double rad, Vec p, Vec e, Vec c, Refl_t refl):
22.         rad(rad), p(p), e(e), c(c), refl(refl) {}
23.     double intersect(const Ray &r) const { // solve t^2+2*o.p.d+(o.p-o.p).d^2=0 if nohit
24.         Vec op = p-r.o; // Solve t^2+2*o.p.d+2*t*(o-p).d+(o-p).(o-p)-R^2=0
25.         double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
26.         if (det<0) return 0; else det=sqrt(det);
27.         return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
28.     };
29. };
30. Sphere spheres[] = { //Scene: radius, position, emission, color, material
31.     Sphere(1e5, Vec(1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF), //Left
32.     Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF), //Right
33.     Sphere(1e5, Vec(50,40.8,1e5), Vec(),Vec(.75,.75,.75),DIFF), //Back
34.     Sphere(1e5, Vec(50,40.8,-1e5+10), Vec(),Vec(),DIFF), //Front
35.     Sphere(1e5, Vec(0,1e5+1,40.8),Vec(),Vec(.75,.75,.75),DIFF), //Botm
36.     Sphere(1e5, Vec(0,1e5+1,40.8),Vec(),Vec(.75,.75,.75),DIFF), //Top
37.     Sphere(16.5,Vec(27,16.5,47), Vec(),Vec(1,1,1)*.999, SPEC), //Mirr
38.     Sphere(16.5,Vec(73,16.5,78), Vec(),Vec(1,1,1)*.999, REFR), //Glas
39.     Sphere(600, Vec(50,681.6,-27,81.6),Vec(12,12,12), Vec(),DIFF) //Lite
40. };
41. inline double clamp(double x) { return x<0 ? 0 : x>1 ? 1 : x; }
42. inline int toInt(double x) { return int(pow(clamp(x),1/2.2)*255+.5); }
43. inline bool intersect(const Ray &r, double &t, int &id) {
44.     double n=sizeof(spheres)/sizeof(Sphere), d, int n=1e20;
45.     for(int i=0;i<n;i++) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
46.     return t<inf;
47. }
48. Vec radiance(const Ray &r, int depth, unsigned short *Xi){
49.     double t; // distance to intersection
50.     int id=0; // id of intersected object
51.     if (!intersect(r, t, id)) return Vec(); // if miss, return black
52.     const Sphere &obj = spheres[id]; // the hit object
53.     Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
54.     double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
55.     if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
56.     if (obj.refl == DIFF) { // Ideal DIFFUSE reflection
57.         double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
58.         Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
59.         Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
60.         return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
61.     } else if (obj.refl == SPEC) { // Ideal SPECULAR reflection
62.         return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
63.     } else if (obj.refl == REFR) { // Ideal dielectric REFRACTION
64.         Ray reflRay(x, r.d-n*2*n.dot(r.d)); // Ray from outside going in?
65.         bool into = n.dot(nl)>0; // Total internal reflection
66.         double nc=1, nt=1.5, nnt=into*nc/nt; ddn=r.d.dot(nl), cos2t;
67.         if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0) // Total internal reflection
68.             return obj.e + f.mult(radiance(reflRay,depth,Xi));
69.         Vec tdir = (r.d*nt - n*(into?1:-1)*(ddn*nnt+sqrt(cos2t))).norm();
70.         double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(nl));
71.         double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
72.         return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ? // Russian roulette
73.             radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
74.             radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
75.     }
76. }
77. int main(int argc, char *argv[]){
78.     int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
79.     Ray cam(Vec(50,52,295.6), Vec(0,-0.042812,-1).norm()); // cam pos, dir
80.     Vec cx=Vec(w*.5135/h), cy=(cx*cam.d).norm()**.5135, z, *c=new Vec[w*h];
81.     #pragma omp parallel for schedule(dynamic, 1) private(r) // OpenMP
82.     for (int y=0; y<h; y++){ // Loop over image rows
83.         for (int x=0; x<w; x++){ // Loop over image cols
84.             for (int sx=0; sx<2; sx++){ // 2x2 subpixel rows
85.                 for (int sy=0; sy<2; sy++){ // 2x2 subpixel cols
86.                     double r1=2*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r1)-1;
87.                     double r3=2*erand48(Xi), r4=erand48(Xi), r4s=sqrt(r2)-1;
88.                     Vec d = cx*( ( (sx+.5+dx)/2 + x)/w - .5) +
89.                         cy*( ( (sy+.5+dy)/2 + y)/h - .5) + cam.d;
90.                     r = r + radiance(Ray(cam.o+d*140,d.norm(),0,Xi)*(1./samps);
91.                     // Camera rays are pushed forward to start in interior
92.                     c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z)).*.25;
93.                 }
94.             }
95.         }
96.     }
97.     FILE *f = fopen("image.ppm", "w"); // Write image to PPM file.
98.     fprintf(f, "P3\n%d %d\n255\n", w, h);
99.     for (int i=0; i<w*h; i++) {
100.         fprintf(f, "%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
101.     }
102. }
```

An Introduction to Radiometry

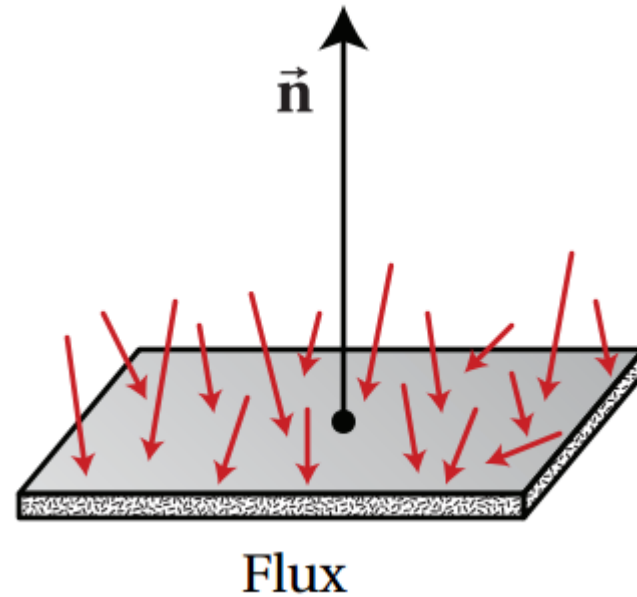
Radiometry

- Definition: the study of electromagnetic radiation, including visible light
- What are the most fundamental physical quantities used when measuring “light”?
- Which are the most important for physically-based rendering?



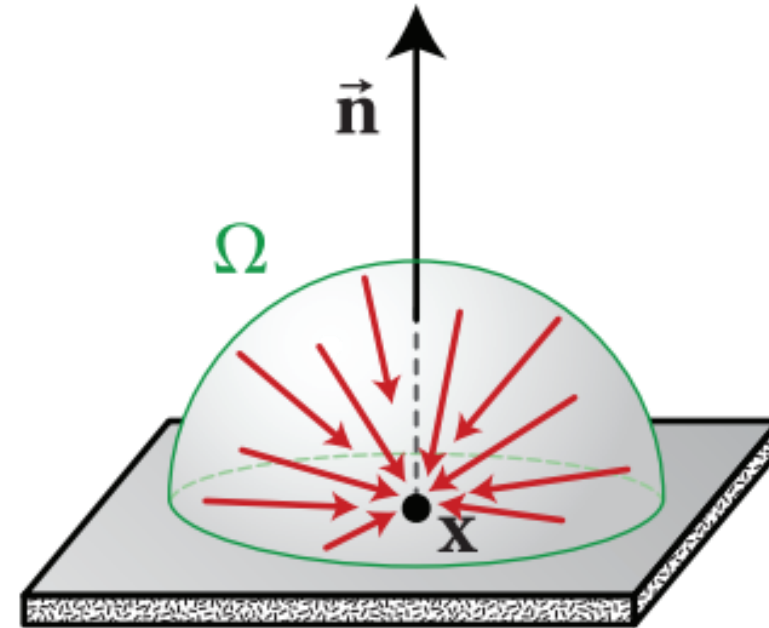
Flux Φ

- The amount of energy per unit time, a.k.a. power, hitting (or passing through) a surface
 - Measured in Watts ($W \equiv \frac{J}{s}$)



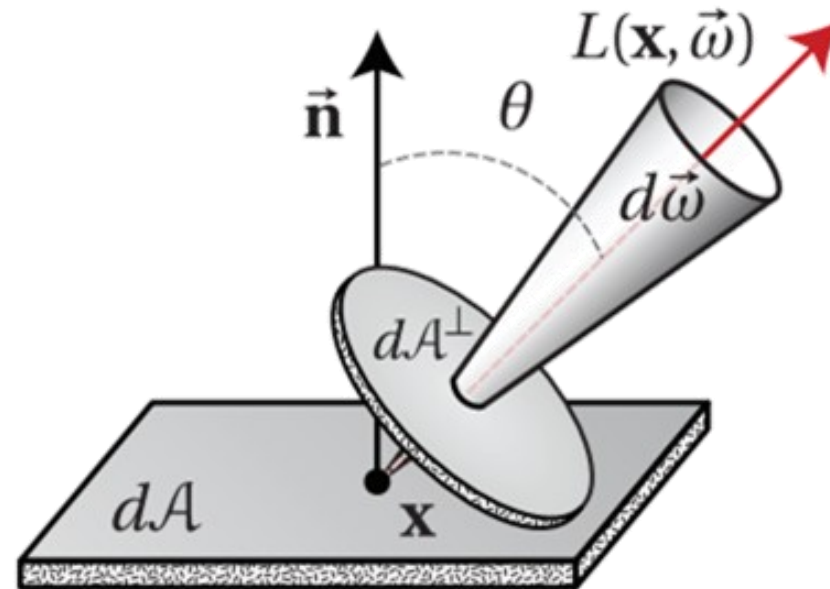
Irradiance $E = \frac{d\Phi}{dA}$

- The power (Flux) per unit (differential) surface area
 - Watts per meter squared ($E \equiv \frac{W}{m^2}$)
- Always measured at a surface point x with normal \vec{n}
- Irradiance leaving a surface area element is often called emittance M , radiant exitance or radiosity B



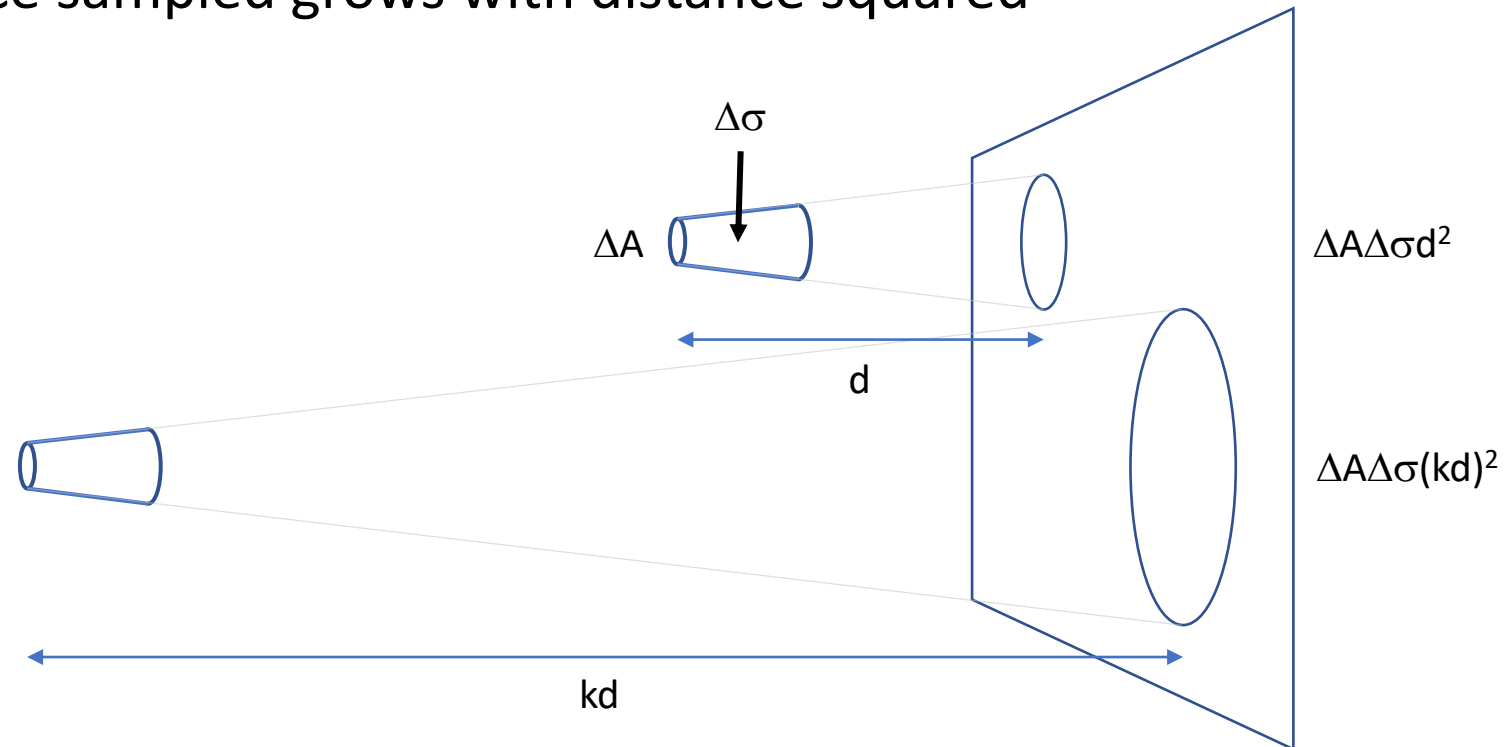
$$\text{Radiance } L = \frac{d^2\Phi}{dA^\perp d\vec{\omega}} = \frac{d^2\Phi}{dA (\vec{n} \cdot \vec{\omega}) d\vec{\omega}}$$

- Power hitting (differential) area perpendicularly from a (differential) cone of directions;
 - Watts per meter squared per steradian
 $\left(L \equiv \frac{W}{sr \cdot m^2} \equiv \frac{E}{sr}\right)$
- The most important quantity in rendering! This is the “color” or pixel intensity you observe in a photo
- Note the cosine of the angle between dA^\perp and dA



Radiance

- Radiance does not vary along a line in space
 - Light inversely proportional to distance squared
 - Area of surface sampled grows with distance squared



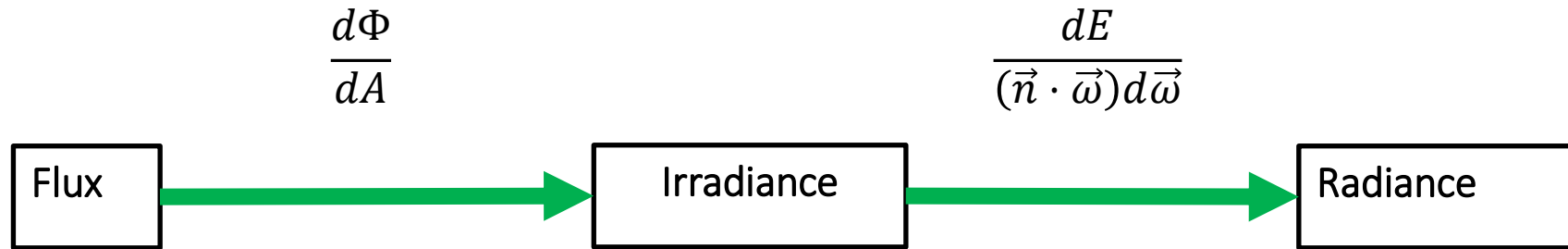
Radiometry

Summary of Terminology

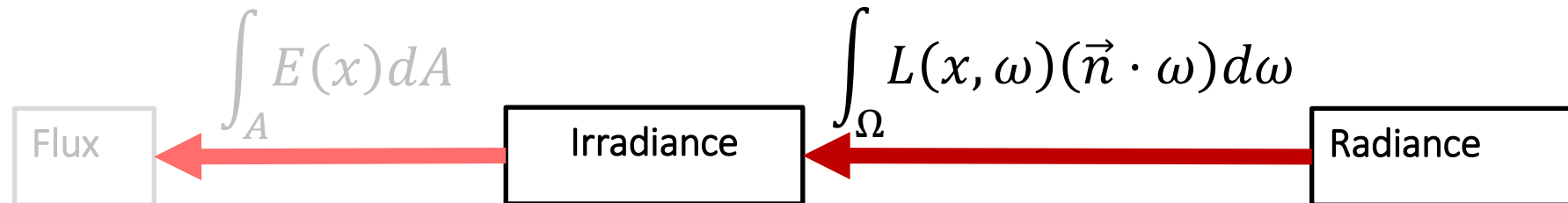
Symbol	Units	Name
Φ	Watts ($W \equiv \frac{J}{s}$)	Flux
E	$E \equiv \frac{W}{m^2}$	Irradiance
M	$E \equiv \frac{W}{m^2}$	Radiant exitance
B	$E \equiv \frac{W}{m^2}$	Radiosity
L	$L \equiv \frac{W}{sr \cdot m^2} \equiv \frac{E}{sr}$	Radiance

From Radiometry to Rendering

- In radiometry we typically move between these quantities, from left to right, through differentiation

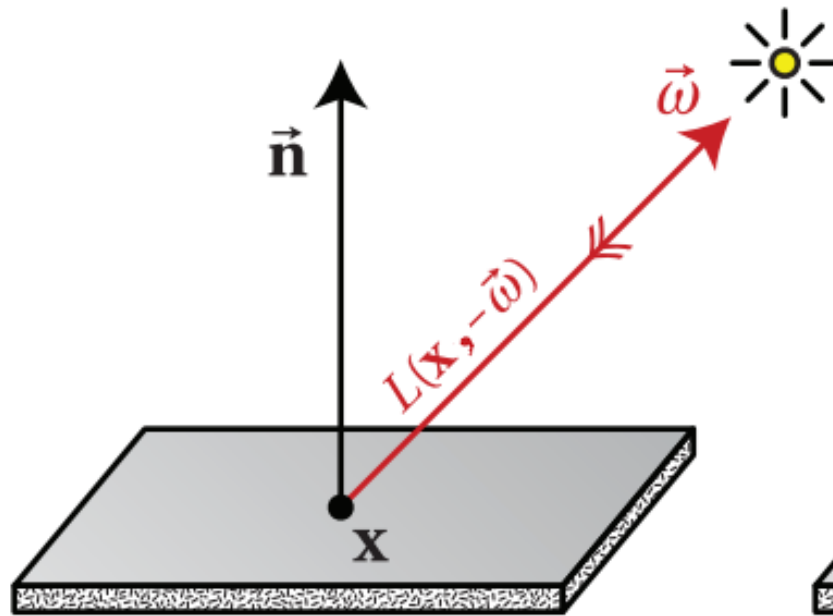


- In rendering, we usually integrate, moving from right to left

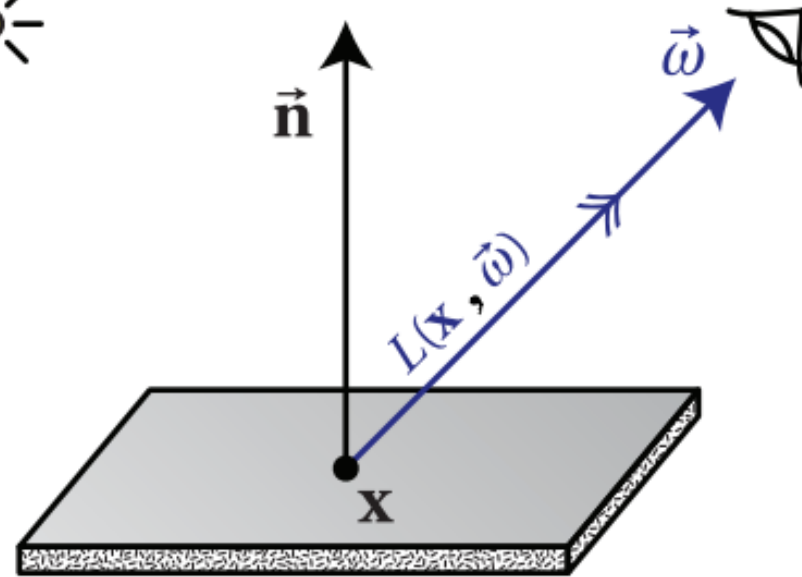


Radiance

- As with irradiance, radiance can be used to represent both incident and exitant quantities



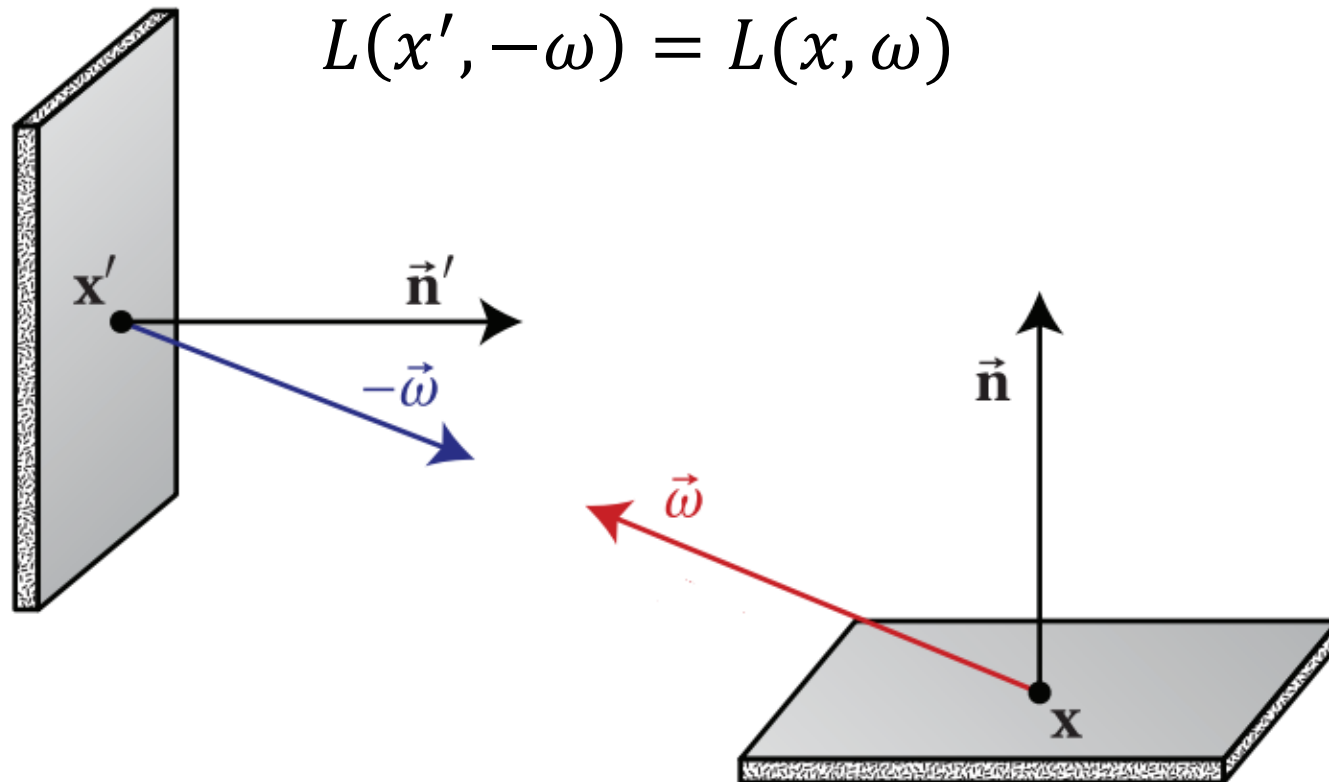
Incident Radiance



Exitant Radiance

Radiance

- In a vacuum, radiance remains constant along (unoccluded) rays

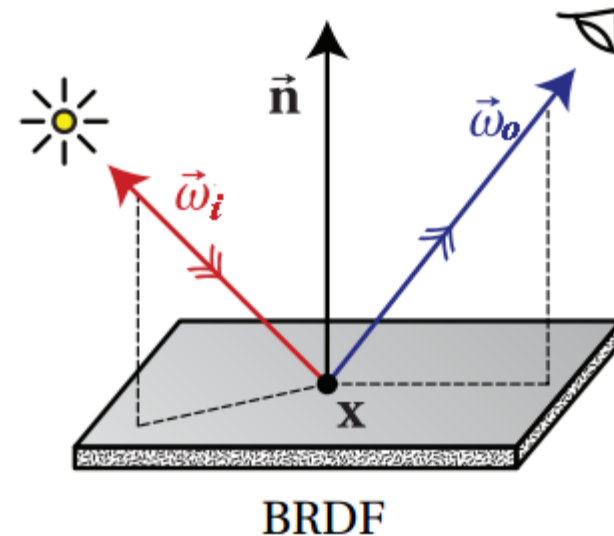


Bi-directional Reflectance Distribution Function (BRDF)

- Incident radiance at x warped to exitant/reflected (differential) radiance according to the BRDF, f_r

$$f_r(x, \omega_i, \omega_o) = \frac{dL_r(x, \omega_o)}{dE(x, \omega_i)} = \frac{dL_r(x, \omega_o)}{L(x, \omega_i)(\vec{n} \cdot \omega_i)d\omega_i}$$

- The BRDF is responsible for the appearance of different materials
- For example, the BRDF of wood is different than that of metal
- The BRDF has units sr^{-1}

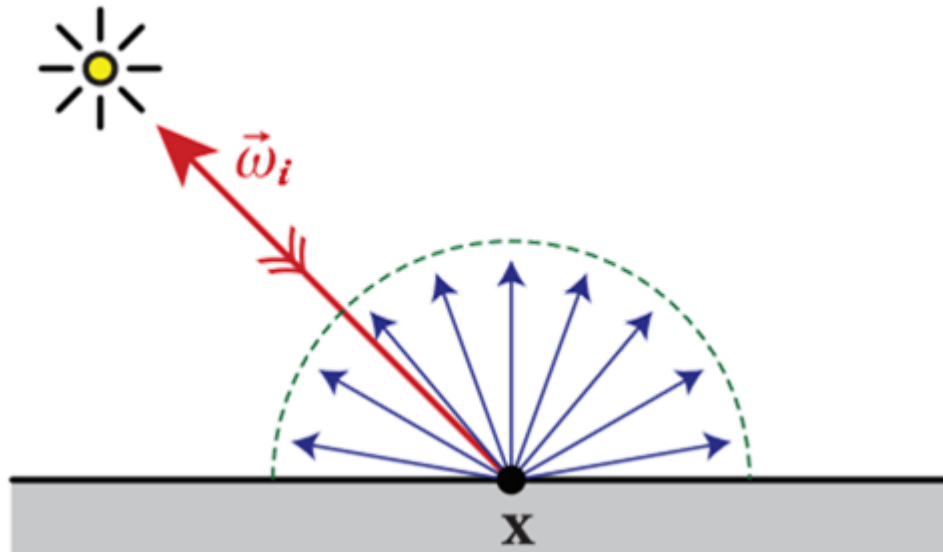


Important BRDF Properties

- At each x , the BRDF f_r is a 4 dimensional function
 - Two for incident lighting directions, and
 - Two for exitant (reflected) lighting directions
- The most important mathematical/physical properties of a BRDF are:
 - Reciprocity: $f_r(x, \omega_i, \omega_o) = f_r(x, \omega_o, \omega_i)$
 - Energy conservation: $\int_{\Omega} f_r(x, \omega_i, \omega_o) (\vec{n} \cdot \omega_i) d\omega_i \leq 1, \forall \omega_o$
 - Positivity: $f_r(x, \omega_i, \omega_o) \geq 0, \forall \omega_i \text{ and } \forall \omega_o$

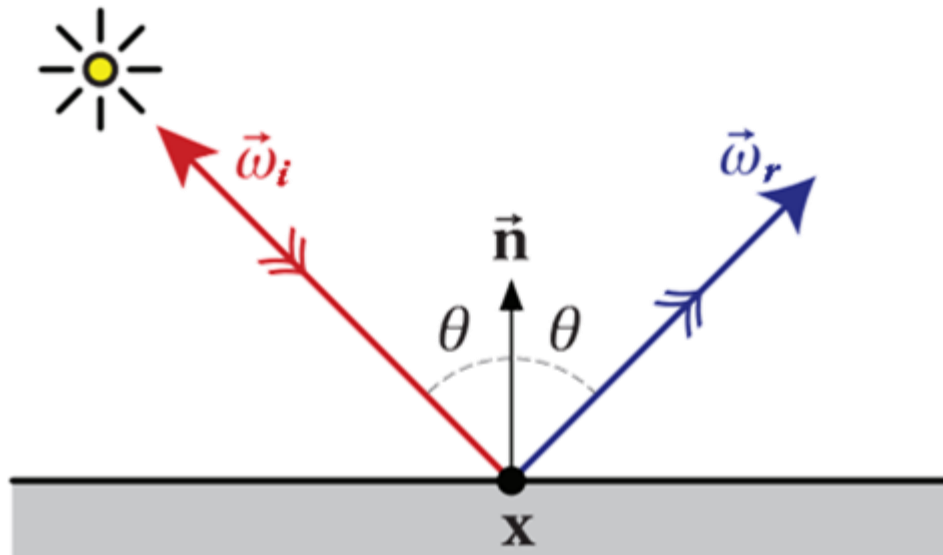
A few Examples of BRDFs

- Special case #1: Diffuse/Lambertian
 - A material that reflects light (radiance) equally in all directions
 - View-independent reflection
 - $f_r(x, \omega_i, \omega_o) = \frac{\rho}{\pi}$



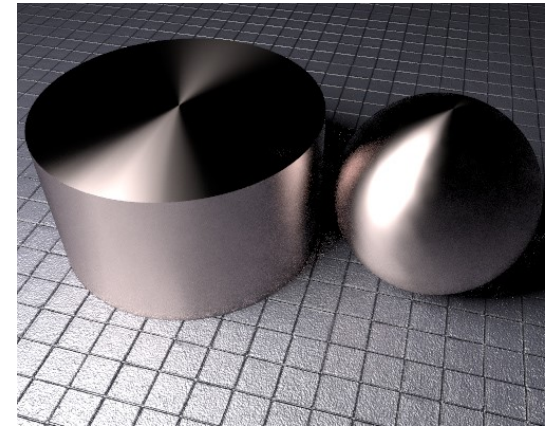
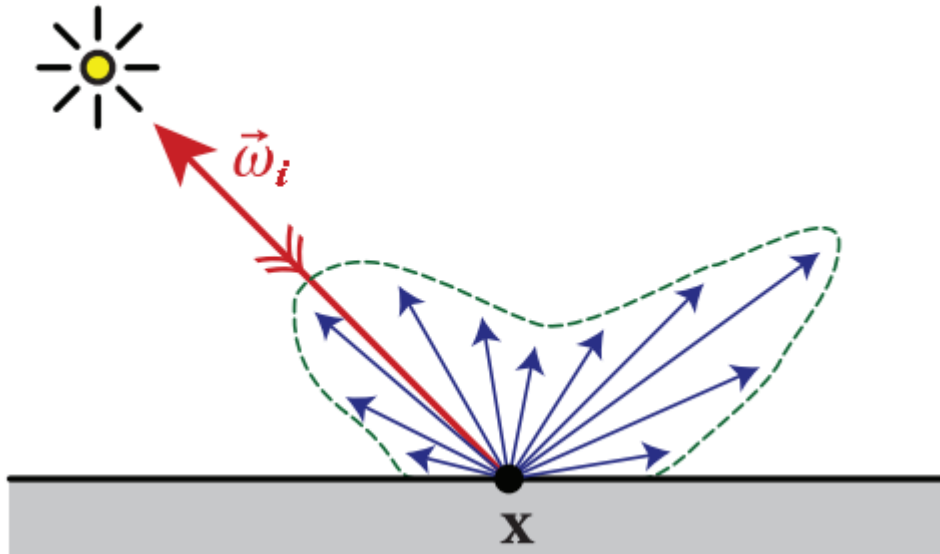
A few Examples of BRDFs

- Special-case #2: perfect specular mirror
 - Only reflects light a single direction: the perfect mirror reflection direction
 - $$f_r(x, \omega_i, \omega_o) = \frac{\delta(\omega_r(\omega_i))}{\vec{n} \cdot \omega_i}$$

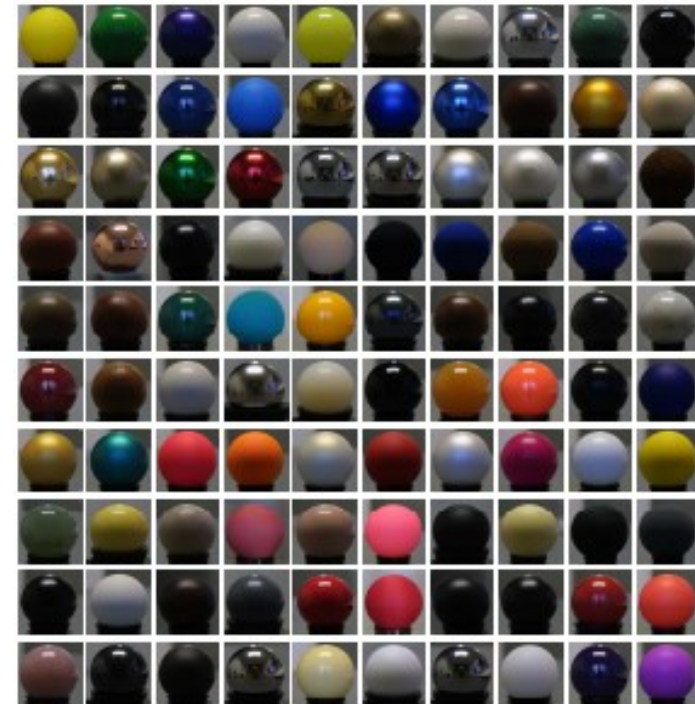


A few Examples of BRDFs

- General case:
 - For each direction of incident illumination, the BRDF specifies a (spherical) distribution of exitant illumination
 - A ratio of reflected light for any possible view direction



Example of brushed aluminum (anisotropic BRDF)



Emitted and Reflected Radiance

- Exitant radiance (from a surface, in a cone of directions) is equal to the sum of emitted (L_e) and reflected (L_r) radiance:

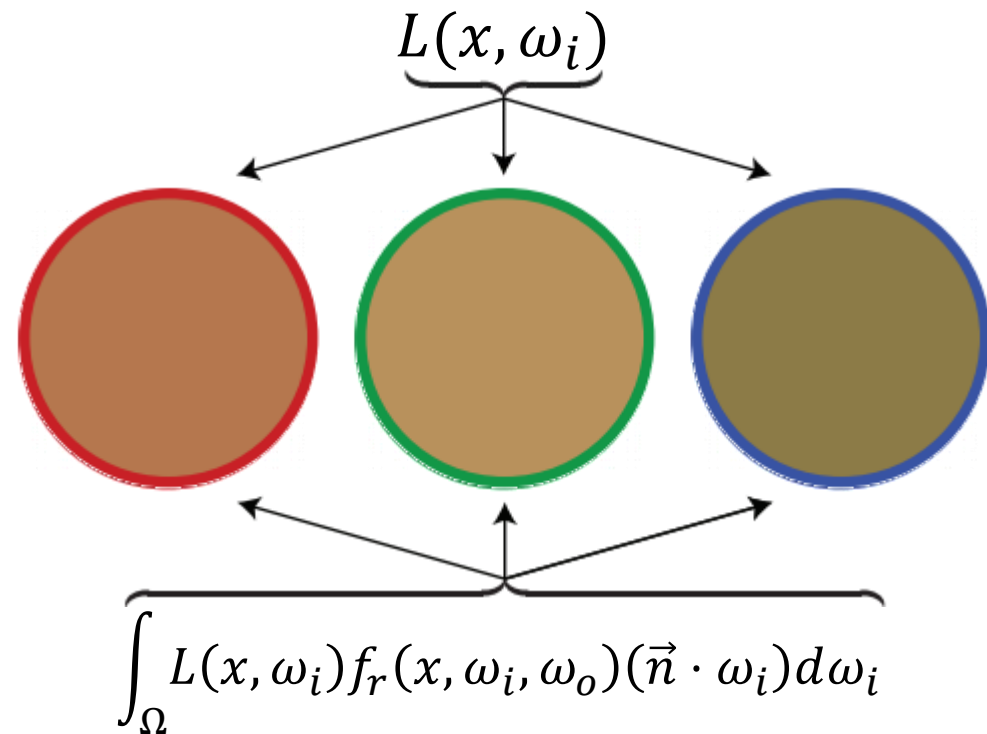
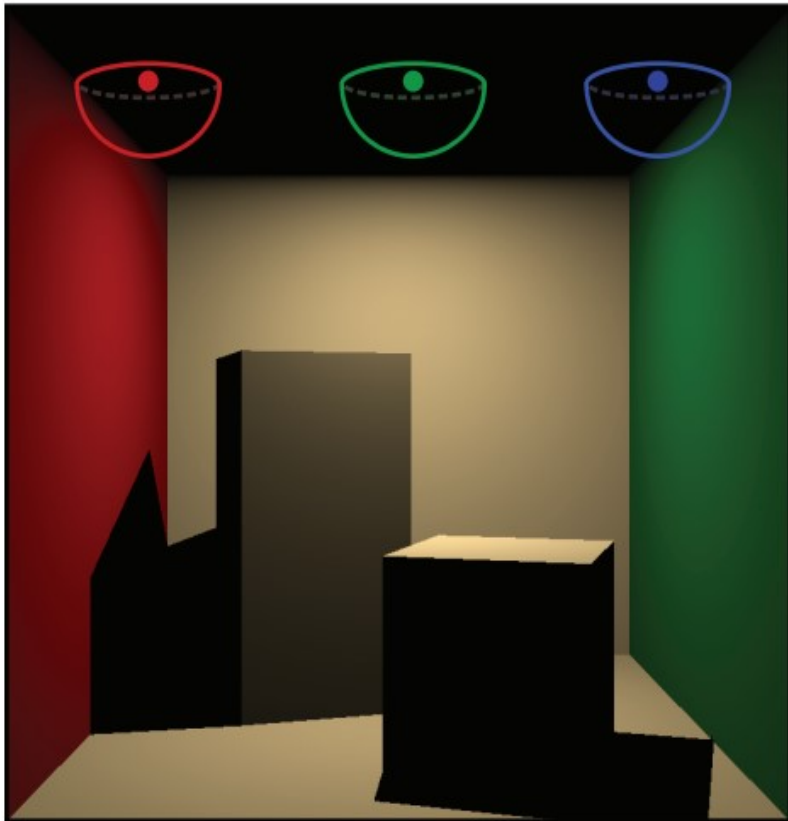
$$L(x, \omega_o) = L_e(x, \omega_o) + L_r(x, \omega_o)$$

- Only light source surfaces have a non-zero emitted radiance component ($L_e \neq 0$)
 - We typically treat these surfaces (lights) as non-reflective ($f_r = 0$)

Reflected Radiance

a visual example

- L_r is the integral, over (differential) incident directions $d\omega_i$ around \vec{n} , of incident radiance weighted by the BRDF and cosine projection



The Rendering Equation

- This finally leads us to the rendering equation

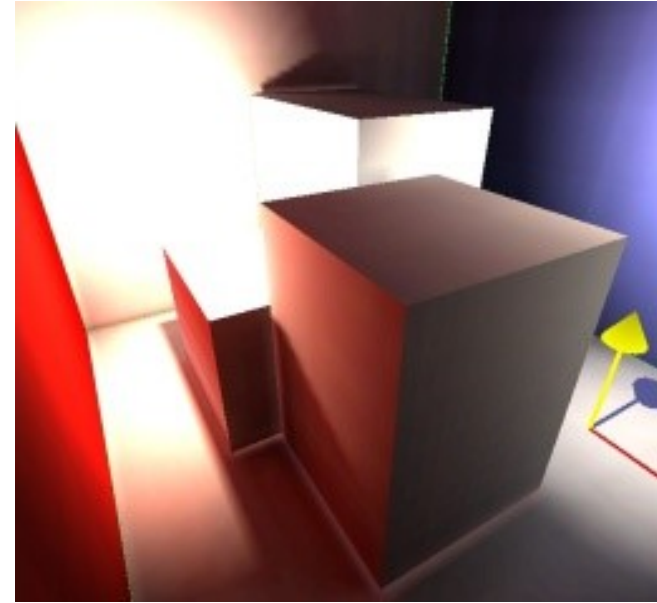
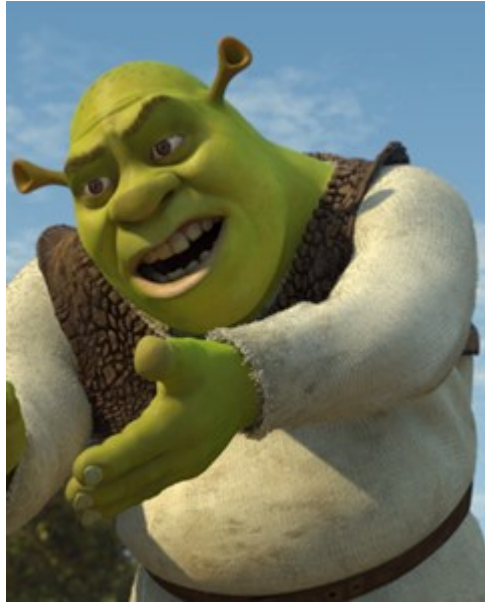
$$L(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} L(x', -\omega_i) f_r(x, \omega_i, \omega_o) (\vec{n} \cdot \omega_i) d\omega_i$$

- Note that this general formulation is recursive!
- Almost every shading algorithm you will encounter is based on (or a simplifications of) this equation

Direct vs. Global Illumination

- The rendering equation defines the global illumination of a scene
- This includes direct and indirect illumination
- Direct illumination is light that is directly reflected and/or occluded from light sources
- Indirect illumination is light reflected and/or occluded recursively off the remaining (non-light) surfaces in a scene

Direct vs. Global Illumination



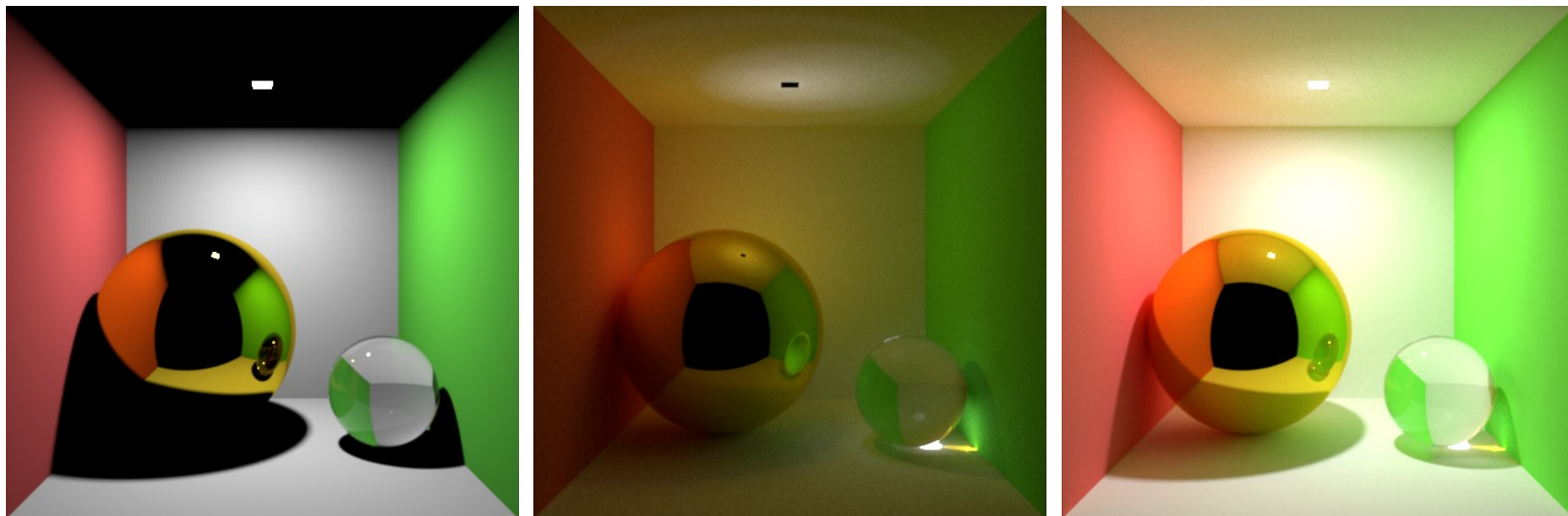
Direct vs. Global Illumination



Direct vs. Global Illumination



Direct vs. Global Illumination



Global Illumination Techniques: Overview

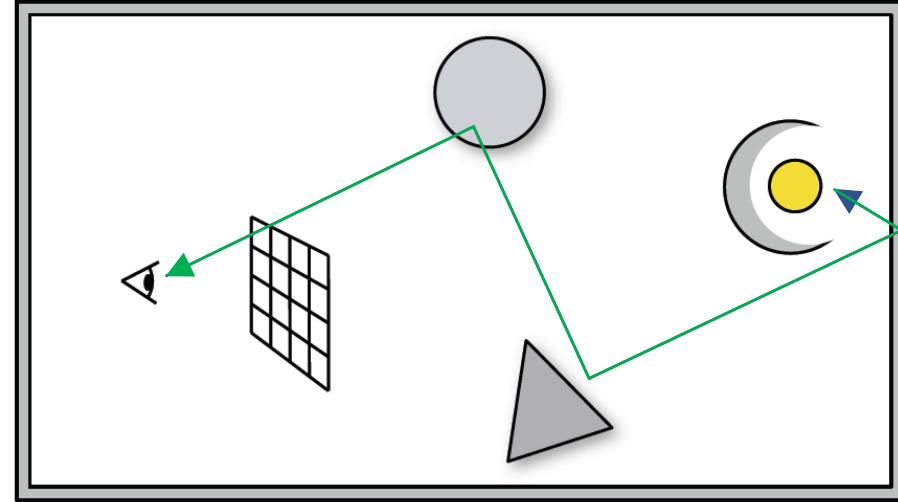
- Monte Carlo Integration
 - Ambient occlusion
 - Direct illumination
 - Indirect illumination
- Path tracing
 - implicit
 - explicit
- Photon mapping

Solving the Rendering Equation

- How to solve a multi-dimensional integral equation?
 - Analytically? Only in trivially simple scenes...
 - Numerically? How to avoid the curse of dimensionality?
- We're going to:
 - use a special type of numerical integration (MC Integration) that completely side steps the curse of dimensionality, and
 - use ray-tracing to sample/evaluate the terms in our integrand
- Conceptually, we want to connect pixels to points on the light sources using paths that light follows

Solving the Rendering Equation

- Several approaches:
 - Do we start at the eye and trace paths?
 - What about starting at the light and tracing paths?



- How do we generate these connecting paths? How can we guarantee the correctness of this process?
- To answer these questions, we first need theory of numerical integration suitable for multi-dimensional integration

An Introduction to Monte Carlo

Monte Carlo Integration

- A Monte Carlo estimator for an integral (of arbitrary dimensionality $\dim(X)$)

$$F = \int_X f(x) dx$$

- is defined as

$$\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)} \approx \int_X f(x) dx$$

- where the N samples $X_i \in X$ are drawn according to a probability distribution function pdf

$$\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)}$$

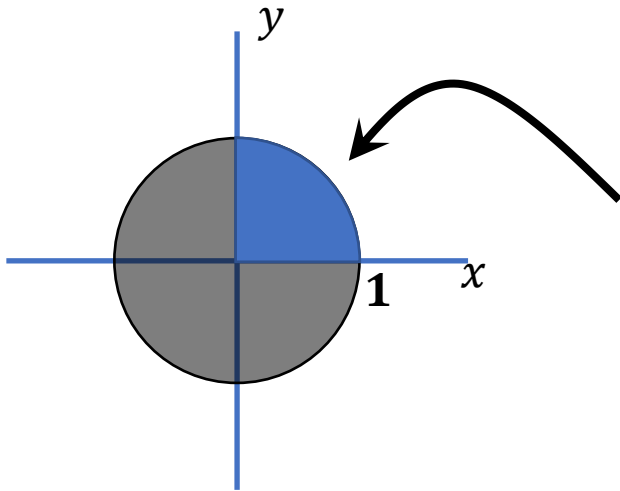
Monte Carlo Integration

- Given the definition of our Monte Carlo estimator, the process of evaluating \bar{F} is divided into 3 steps:
 - Choose* the pdf from which to draw random samples
 - Draw N random samples according to pdf (i.e., the X_i)
 - Evaluate $\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)}$
- Note that the X_i can also be generated on-the-fly in the loop in step 3

Monte Carlo Integration: an example in 1D

$$\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)}$$

- Let's consider a simple example that we can validate analytically:



$$\int_0^1 \sqrt{1-x^2} dx$$
$$= \frac{\pi}{4} \approx 0.785398$$

```
integral = 0
for( i = 0 to N-1 )
    X[i] = rand();
    pdf[i] = 1;
    integral += sqrt(1.-
        X[i]*X[i])/pdf[i]

integral = integral / N
```

1. Choose pdf : for the example, we'll just use a uniform pdf

$$pdf(x) = \frac{1}{b-a} = \frac{1}{1-0} = 1$$

2. Generate the N points X_i
3. Evaluate \bar{F}

$$\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)}$$

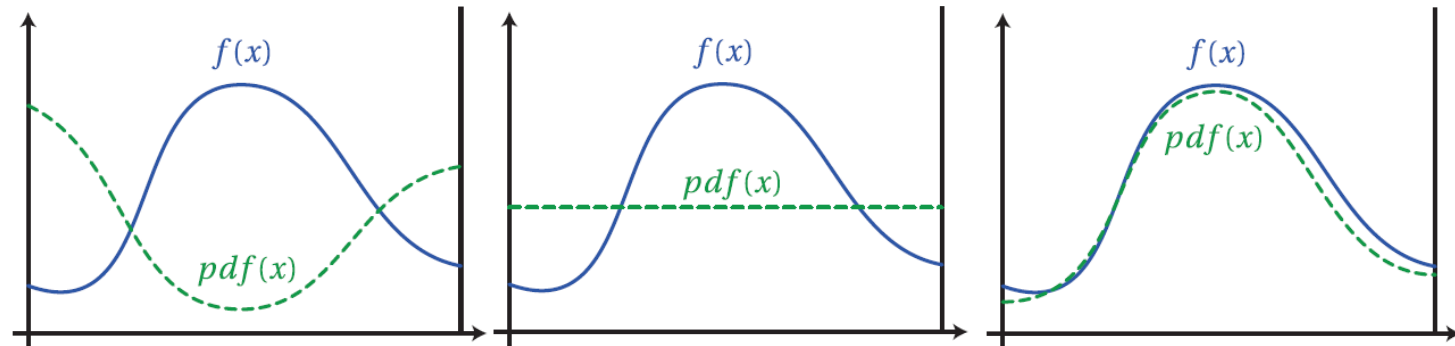
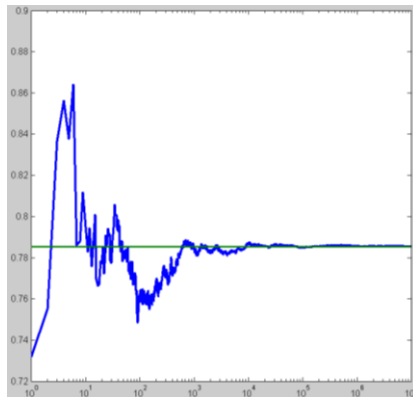
Monte Carlo Integration

- One design goal for an MC estimator is to sample according to a pdf that reduces the variance in the estimator's convergence
- Two typical variance-reduction strategies are stratification and importance sampling

$$\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)}$$

Monte Carlo Integration

- Stratification divides the integration domain into smaller sub-domains and performs MC estimation separately within each sub-domain
- Importance sampling refers to choosing a $pdf(x)$ that samples X_i preferentially where $f(x)$ has higher values

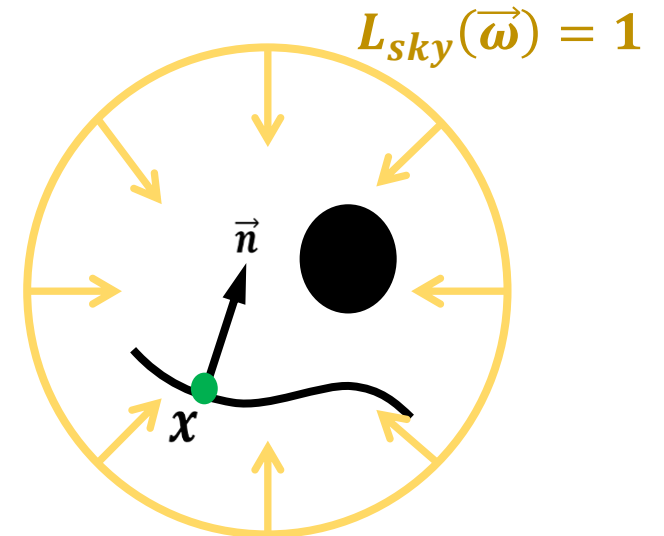


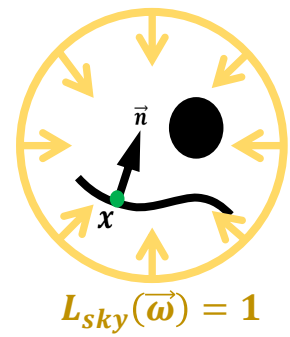
Example 1: Ambient Occlusion

- Consider the following scenario: we want to compute the shading of a scene composed of only **diffuse** objects (each with $\rho = 1$), with **direct illumination** exclusively due to **sky lighting**
 - Here, we treat the sky as an infinitely large sphere surrounding our scene, where all points on the sphere have uniform and unit-intensity emitted radiance $L_{in}(x, \vec{\omega}) = L_{sky}(\vec{\omega}) = 1$

$$L_o(x, \vec{\omega}_o) = L_{ao}(x) = \int_{\Omega} \frac{\max(\vec{n}_x \cdot \vec{\omega}, 0)}{\pi} V(x, \vec{\omega}) d\vec{\omega}$$

- We will investigate how to implement a MC estimator for AO using: a naive pdf and an optimal pdf





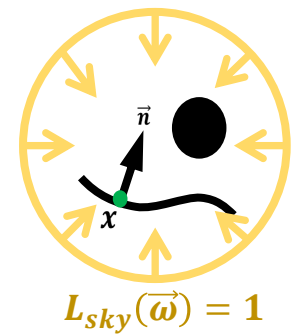
Example 1: Ambient Occlusion

- Let's start by writing the Monte Carlo estimator equation (with a general *pdf* for now) for

$$L_{ao}(x) = \int_{\Omega} \frac{\max(\vec{n}_x \cdot \vec{\omega}, 0)}{\pi} V(x, \vec{\omega}) d\vec{\omega}$$

as

$$\overline{L_{ao}}(x) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)}{\pi \text{pdf}(\vec{\omega}_i)}$$



Example 1: Ambient Occlusion

$$\overline{L}_{ao}(x) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)}{\pi \text{pdf}(\vec{\omega}_i)}$$

- And now, like before, we just need to:
 - Pick a $\text{pdf}(\vec{\omega}_i)$, and
 - Distribute the random samples $\vec{\omega}_i$ according to this pdf
- We'll investigate two choices for $\text{pdf}(\vec{\omega}_i)$, how to distribute random samples according to these pdfs, and their effect on the convergence/variance of our estimators

Ambient Occlusion – Estimator 1

(pdf = uniform spherical directions)

$$\overline{L}_{ao}(x) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)}{\pi \text{pdf}(\vec{\omega}_i)}$$

- As with our earlier 1D example, let's start with the simplest (naive) choice for our pdf: uniform distribution
 - In the 1D case we had $\text{pdf}(x_i) = \frac{1}{\text{length}(X)} = \frac{1}{\int_X dx} = \frac{1}{b-a}$ but now our integration/sampling domain Ω is the set of unit directions
 - This corresponds to the surface (not volume!) of a unit sphere
 - So, for (random) samples distributed uniformly on the surface of the unit sphere, we have $\text{pdf}(\vec{\omega}_i) = \frac{1}{\text{area}(\Omega)} = \frac{1}{\int_{\Omega} d\vec{\omega}} = ?$

Ambient Occlusion – Estimator 1

(pdf = uniform spherical directions)

$$\overline{L}_{ao}(x) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)}{\pi \text{pdf}(\vec{\omega}_i)}$$

$$\text{pdf}(\vec{\omega}_i) = \frac{1}{\text{area}(\Omega)} = \frac{1}{\int_{\Omega} d\vec{\omega}} = \frac{1}{\int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} \sin \theta \, d\theta \, d\phi} = \frac{1}{4\pi}$$

- Our estimator for L_{ao} , with this uniform pdf , becomes:

$$\overline{L}_{ao}(x) = \frac{4}{N} \sum_{i=0}^{N-1} \max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)$$

- and we only have to figure out how to distribute random samples according to $\text{pdf}(\vec{\omega}_i)$ now...

Ambient Occlusion – Estimator 1

(pdf = uniform spherical directions)

$$\overline{L_{ao}}(x) = \frac{4}{N} \sum_{i=0}^{N-1} \max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)$$

- How do we generate random $\vec{\omega}_i$ directions with a uniform distribution over the sphere?
- Method 1: I show you how to derive it from first principles and we spend the rest of the class on the whiteboard... fun...
- or...

Ambient Occlusion – Estimator 1

(pdf = uniform spherical directions)

- Method 2: we ask Google

```
 $\theta$  = acos ( 1 - 2 * rand() );  
 $\phi$  = 2 * M_PI * rand();
```

Web Images Maps More ▾ Search tools

About 2,510,000 results (0.26 seconds)

[Sphere Point Picking -- from Wolfram MathWorld](#)

[mathworld.wolfram.com](#) > ... > Random Point Picking ▾

have a **uniform** distribution on the surface of a unit **sphere**. This method can also be extended to **hypersphere** point picking. The plots above show the ...



[#AltDevBlog » Generating Uniformly Distributed Points on Sphere](#)

[www.altdevblogaday.com/.../generating-uniformly-distributed-points-on...](#) ▾

May 3, 2012 - An effective **sampling** requires a **uniform** distribution of **samples**. ... shows why this method can generate a **uniform** distribution over a **sphere**.



[Uniform Sampling of a Sphere - File Exchange - MATLAB Central](#)

[www.mathworks.com/.../37004-uniform-sampling-of-a-sphere](#) ▾

Uniform Sampling of a Sphere. by Anton Semechko. 05 Jun 2012 (Updated 23 May 2013). Create an approximately **uniform** triangular tessellation of a unit ...



Ambient Occlusion – Estimator 1

(pdf = uniform spherical directions)

- So, in summary, our first end-to-end MC estimator for AO:

1. Uses a uniform *pdf* of directions on the sphere: $pdf(\vec{\omega}_i) = \frac{1}{4\pi}$
2. Generates random samples $\vec{\omega}_i = (\theta_i, \phi_i)$ about this distribution:

$$\theta_i = \text{acos}(1 - 2 * \text{rand}());$$

$$\phi_i = 2 * \text{M_PI} * \text{rand}();$$

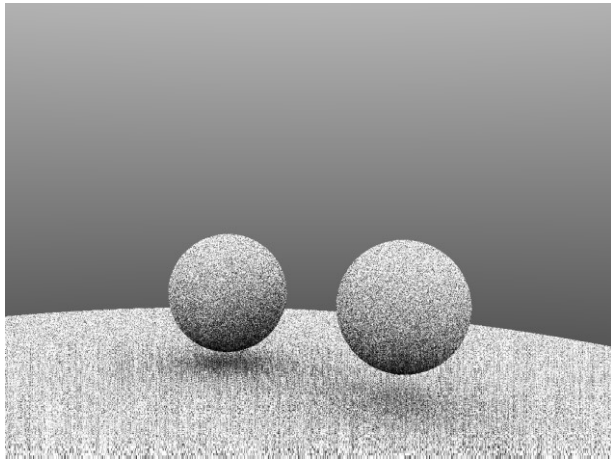
3. And finally calculates ambient occlusion as:

$$\overline{L}_{ao}(x) = \frac{4}{N} \sum_{i=0}^{N-1} \max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)$$

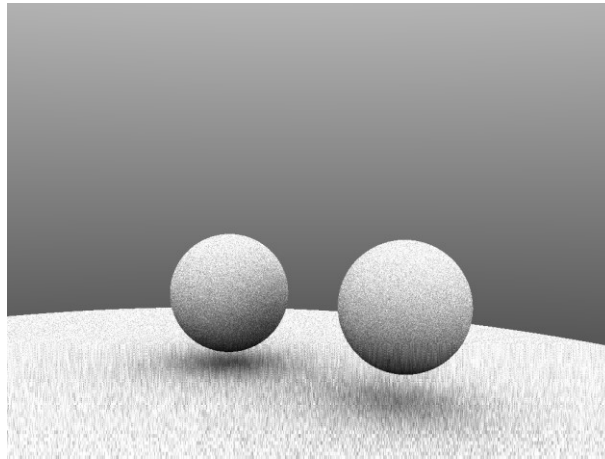
Ambient Occlusion – Estimator 1

(pdf = uniform spherical directions)

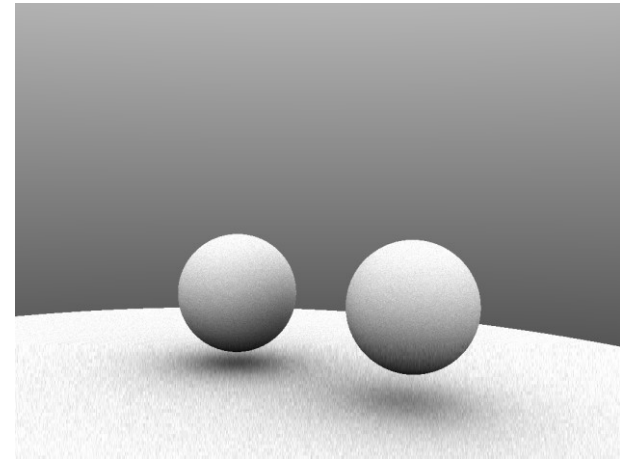
- Results (using `rand()` from `stdlib.h`):



$N = 10$



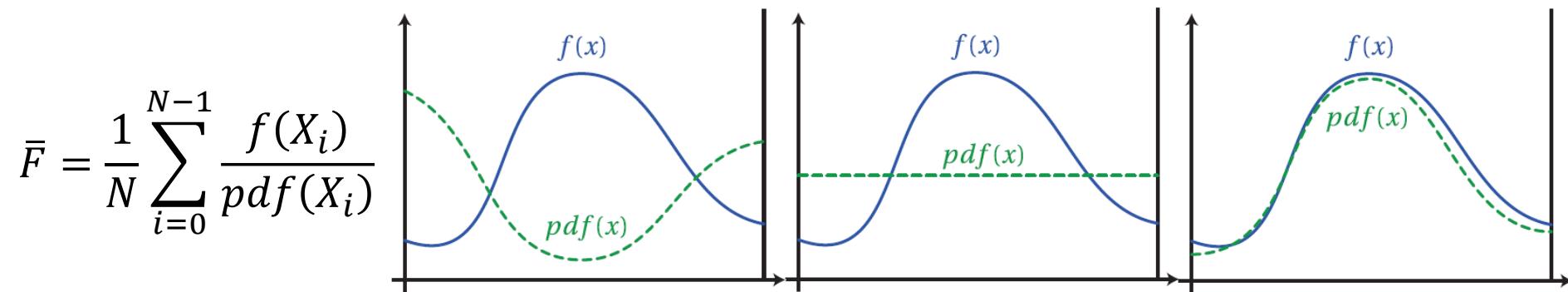
$N = 100$



$N = 1000$

Ambient Occlusion: more efficient estimators?

- Recall that we can reduce the variance of an estimator (using importance sampling) by picking a pdf that better matches the profile of our integrand

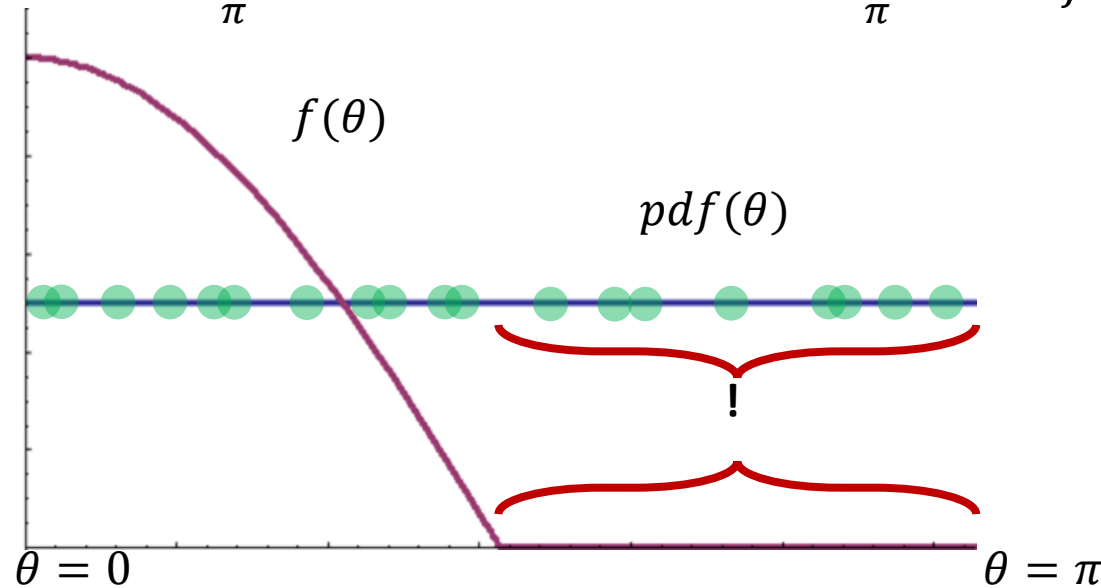


- With our first estimator, the integrand is $f(\omega) = \frac{\max(\vec{n}_x \cdot \vec{\omega}, 0) V(x, \vec{\omega})}{\pi}$ but our pdf is $\frac{1}{4\pi}$

Ambient Occlusion: reducing variance

- What does our integrand look like? Visualizing spherical functions on a slide is hard, but if we fix $\vec{n}_x = \vec{z} = (0,0,1)$ and if we *ignore the visibility*, we can study the *pdf* and integrand visually:

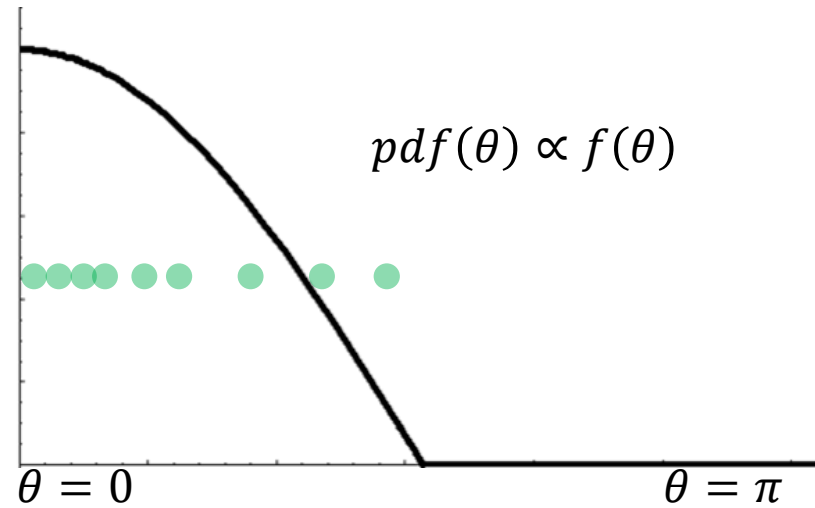
- $$f(\omega)|_{\vec{n}_x=\vec{z}} = \frac{\max((0,0,1) \cdot (\cos \phi \sin \theta, \sin \theta \sin \phi, \cos \theta), 0)}{\pi} = \frac{\max(\cos \theta, 0)}{\pi} = f(\theta)$$



- Clearly this *pdf* wastes a lot of samples!

Ambient Occlusion: reducing variance

- If we continue to ignore visibility, we know **exactly** what our integrand looks like, $f(\omega) = \frac{\max(\vec{n}_x \cdot \vec{\omega}, 0)}{\pi}$, and we can adapt our *pdf* to distribute samples according to this profile:



Ambient Occlusion – Estimator 2


(pdf = hemispherical cosine about the normal)


- So, following the 3 steps for MC integration again:


1. Pick a (better) pdf : $pdf(\vec{\omega}_i) = \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0)}{\pi}$
2. Generate random samples $\vec{\omega}_i = (\theta_i, \phi_i)$ according to this pdf :


Web Images Maps More Search tools


About 209,000 results (0.32 seconds)


[Cosine weighted hemisphere | pathtracing](#)
pathtracing.wordpress.com/2011/03/03/cosine-weighted-hemisphere/ 
Mar 3, 2011 - The top hemisphere shows cosine-weighted sampling – the bottom hemisphere shows uniform sampling (both with 100 000 sampling points).


[smallpt: Global Illumination in 99 lines of C++ - Kevin Beason](#)
www.kevinbeason.com > Software 
... Antialiasing via super-sampling with importance-sampled tent distribution, and ... scene description; Cosine importance sampling of the hemisphere for diffuse ...
You visited this page on 06/11/13.


[ambocc.cu](#)
https://www.cs.uiowa.edu/~cwman/classes/fall10-22C151/.../ambocc.cu 
... +z hemisphere distributed // based on a cosine hemisphere __device__ inline __float3 SampleCosineHemisphere(float u1, float u2) { // Uniformly sample ...

[\[PDF\] Disc and Hemisphere Sampling](#)
web.cs.wpi.edu/~emmanuel/.../wk3_p2_joe_miller_mapping_samples.pdf... 
Disc and Hemisphere Sampling ... Until now sampling has been done on the unit square }1/(e+1)}. • The book describes this as the cosine distribution function.

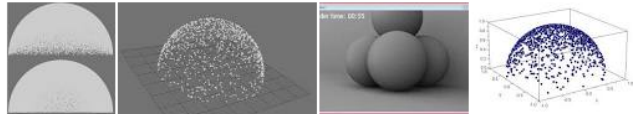
[cosine weighted hemisphere sampling | Bobobobo's Weblog](#)
bobobobo.wordpress.com/2012/.../cosine-weighted-hemisphere-samplin... 
Jun 11, 2012 - So, this post talks about cosine weighted hemisphere sampling. Really its simple. There are 2 ways to do this, both are outlined in Philip Dutre's ...

[Importance Sampling - Rendering Competition](#)
rendering.aspona.cz/ImportanceSampling.aspx 
Importance sampling really speeds up convergency of Monte Carlo based algorithms. ... Uniform sampling hemisphere using UniformSampleSphere including ...

[\[PDF\] in PDF](#)
www.eng.utah.edu/~cs6965/slides/pathtrace.pdf 
Let's start by sampling in pixels N. • Pick a random direction on the normal hemisphere. • How? ... With samples cosine-weighted, we don't need to ...

[CodeItNow » Better Sampling](#)
www.rorydriscoll.com/2009/01/07/better-sampling/ 
Jan 7, 2009 - The integral of the pdf over the hemisphere must equal one, so by switching to a cosine-weighted sample distribution, the pdf becomes ...

[Images for cosine sampling hemisphere - Report images](#)



Ambient Occlusion – Estimator 2

(pdf = hemispherical cosine about the normal)

- So, following the 3 steps for MC integration again:

1. Pick a (better) pdf : $pdf(\vec{\omega}_i) = \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0)}{\pi}$
2. Generate random samples $\vec{\omega}_i = (\theta_i, \phi_i)$ according to this pdf :

$\theta_i = \text{acos}(\text{sqrt}(\text{rand}()))$;

$\phi_i = 2 * \text{M_PI} * \text{rand}()$;

rotate points to a coordinate frame about \vec{n}_x ;

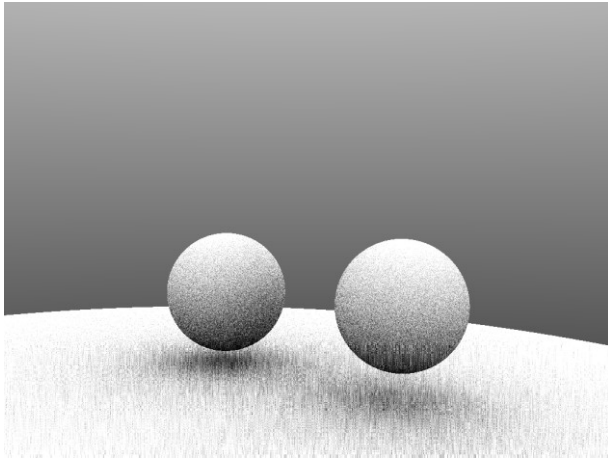
3. Calculate our estimate of the AO integral:

$$\overline{L}_{ao}(x) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0) V(x, \vec{\omega}_i)}{\pi \left(\frac{\max(\vec{n}_x \cdot \vec{\omega}_i, 0)}{\pi} \right)}$$

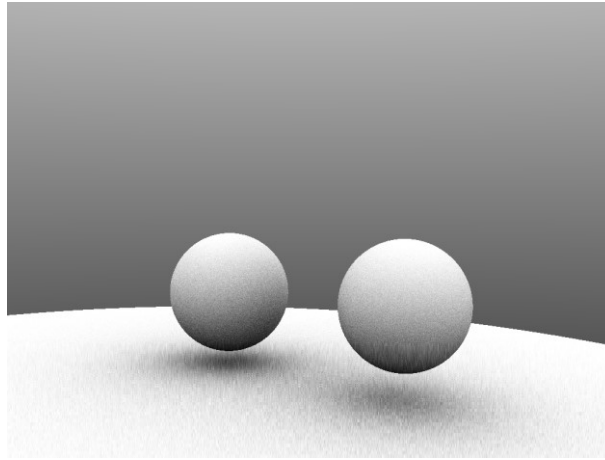
Ambient Occlusion – Estimator 2

(pdf = hemispherical cosine about the normal)

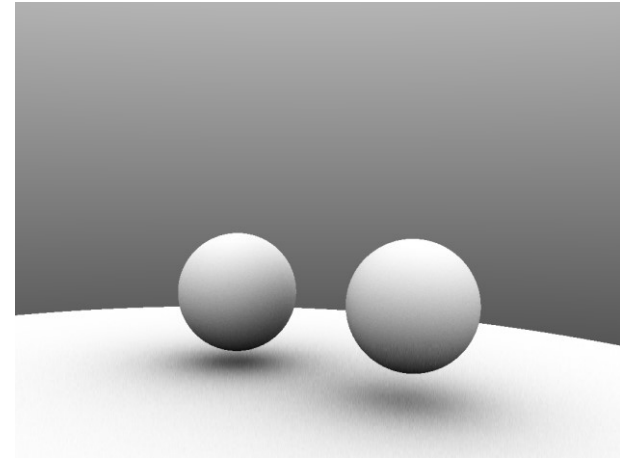
- Results of our new estimator vs. our naive estimator



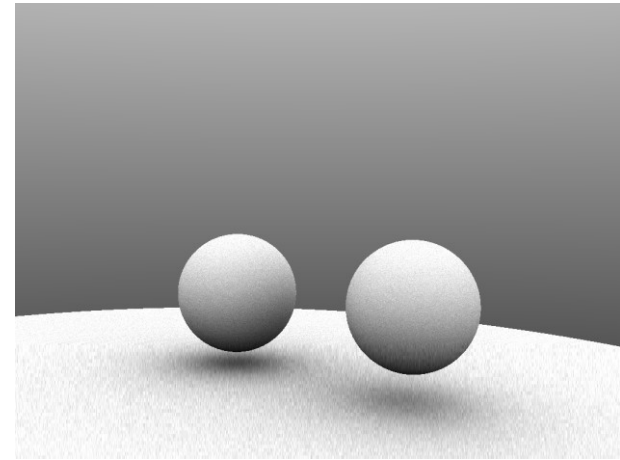
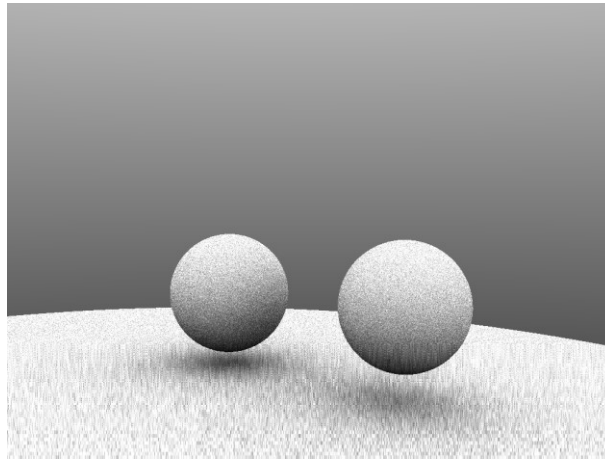
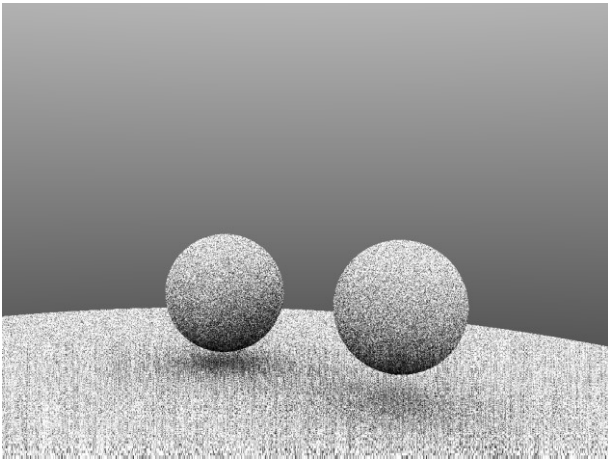
$N = 10$



$N = 100$



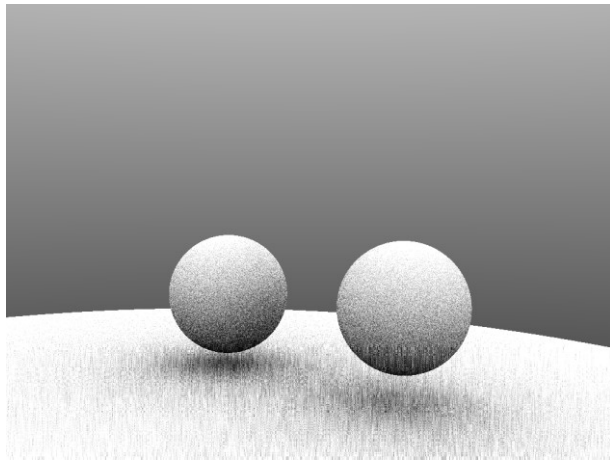
$N = 1000$



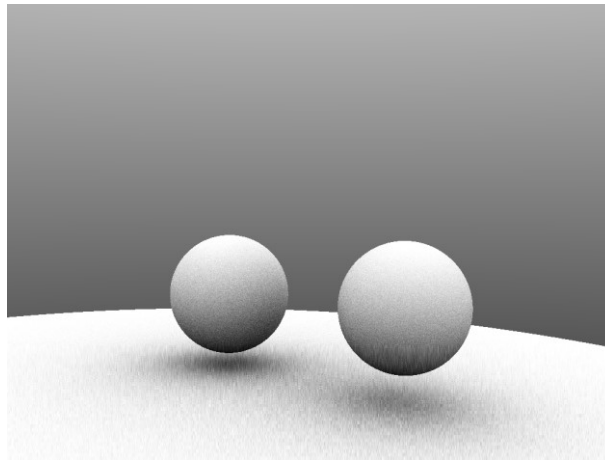
Ambient Occlusion – Estimator 2

(pdf = hemispherical cosine about the normal)

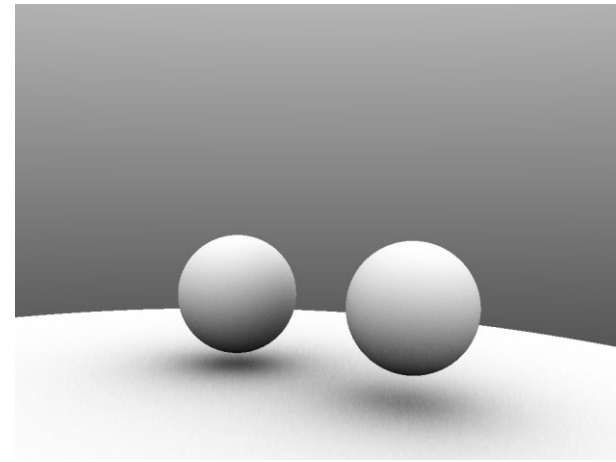
- Results of our new estimator vs. our naive estimator



$N = 10$



$N = 100$



$N = 1000$

Review and more information

- Fundamentals of Computer Graphics
 - Chapter 20 Light
 - Section 20.1 Radiometry
 - Section 20.2 Transport Equation
 - The rest of the chapter is pretty short
- Physically Based Rendering, from theory to implementation
 - <http://www.pbr-book.org/3ed-2018/contents.html>
 - <https://www.pbrt.org/>
 - Chapter 13 covers Monte Carlo Integration