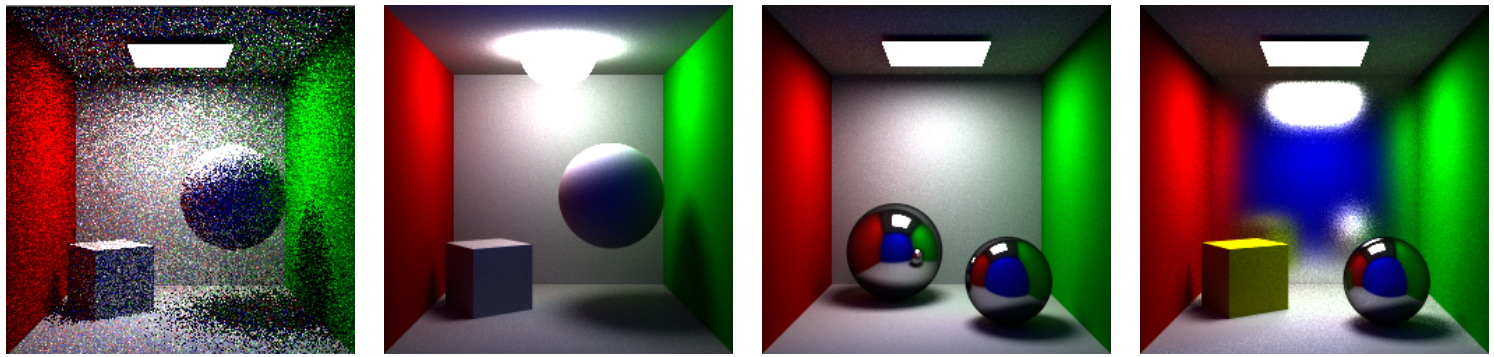
Introduction to Path Tracing

Jan Kautz
Department of Computer Science
University College London



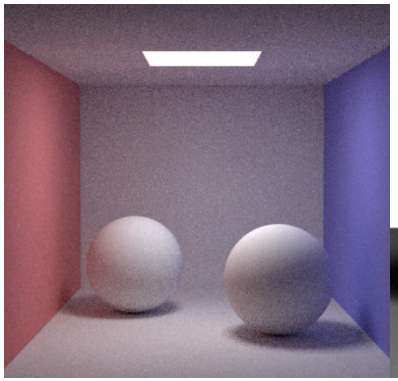
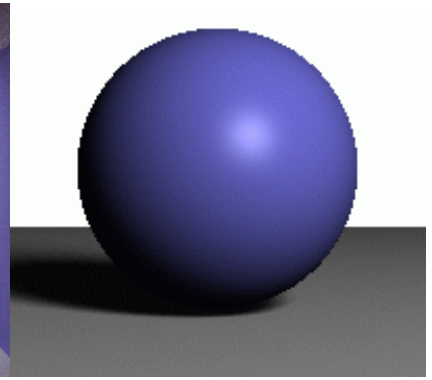
Outline

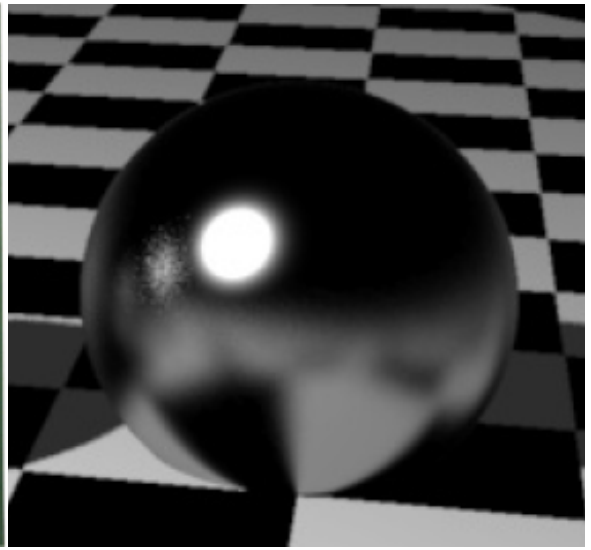
- Monte Carlo Integration
- Basics of the algorithm
- Importance Sampling
- Sampling Directions
- Russian Roulette

What can we integrate?

$$\int\int\int\int\int L(x, y, t, u, v)dx\,dy\,dt\,du\,dv$$

- Pixel: antialiasing
- Light sources: Soft shadows
- Lens: Depth of field
- Time: Motion blur
- BRDF: glossy reflection
- Hemisphere: indirect lighting

Probabilities and Expected Values

- Given a discrete random variable x_i with probabilities p_i
- Expected value: $E[x] = \sum x_i * p_i$

- E.g., for the dice we have
  - $x = \{1,2,3,4,5,6\}$, $p_i = 1/6$
  - Expected value is: $(1+2+3+4+5+6)*1/6 = 3.5$

Probabilities and Expected Values

- Given a continuous random variable x with PDF p(x)
  - Expected value: $E[x] = \int x*p(x)\,dx$

- Now given a function f(x) with random variable x, which has p(x)
  - Expected value: $E[f(x)] = \int f(x)*p(x)\,dx$
  - Discretized: $E[f(x)] \approx 1/N * \sum f(x_i)$

(works for *independent identically distributed* random variables)

Monte Carlo Integration

- This describes a simple technique for the numerical evaluation of integrals

- Suppose: $I = \int f(x)\, dx$

- The Monte-Carlo converts this to an expected value computation problem, where p(x) is some probability density function:

$$I = \int (f(x)/p(x))\, p(x)\, dx = E[f(x) / p(x)]$$

Monte Carlo Integration

- Values can be estimated by taking N samples $x_1, x_2, \ldots, x_n$ drawn from PDF p(x):

$$\langle I \rangle = E[f(x) / p(x)] \approx (1/n) \sum_i f(x_i)/p(x_i)$$

- The variance is proportional to 1/n

- Error decrease: 1/sqrt(n) (= stddev.)

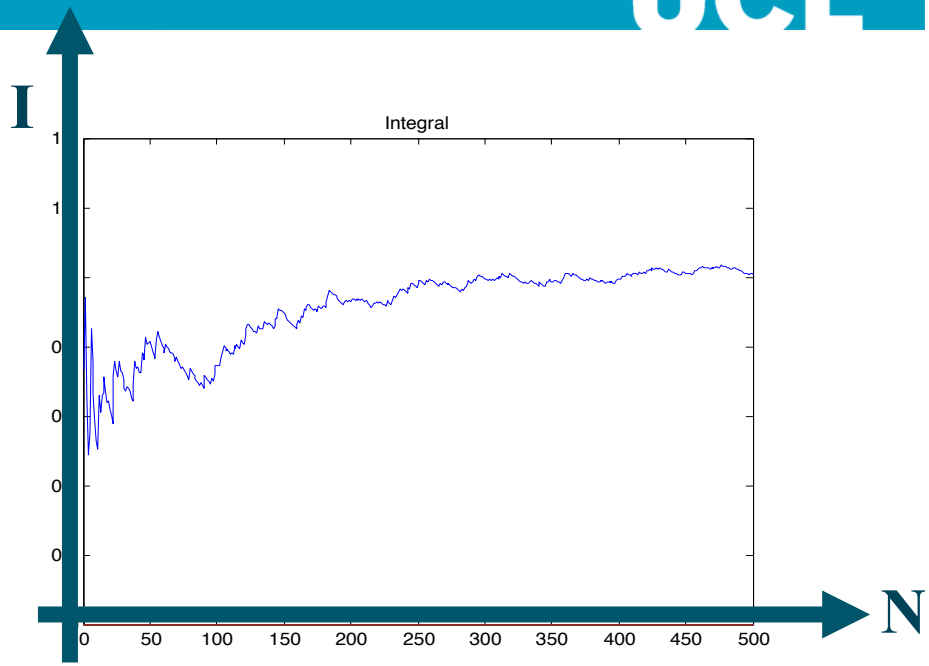
Example

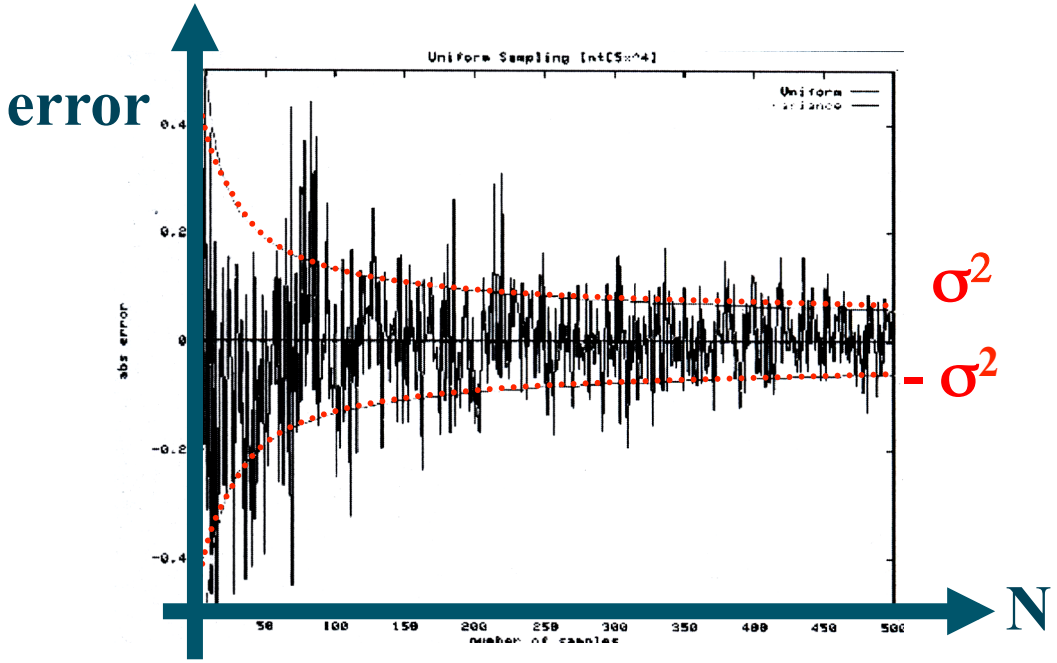
$$I = \int_0^1 5x^4 dx$$

- We know it should be 1.0

- Here with uniform samples (p(x)=1):

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{5x^4}{1}$$





Example

- Integrate from 0 to 2: $I = \int_0^2 5x^4 dx$
- Uniform samples: $p(x) = \frac{1}{2}$
- Hence: $I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{5x^4}{\frac{1}{2}} = \frac{2}{N} \sum_{i=1}^{N} 5x^4 \approx 32$

Draw Samples from PDF

- Draw random samples x_i from PDF $p(x)$ range: $[-\infty, \infty]$
- Compute CDF $F(x) = \int_{-\infty}^{x} f(t)dt$
- Sample inverse CDF uniformly:

$$x_i = F^{-1}(u_i) \qquad u_i \in [0, 1)$$

Example

- Draw random samples from PDF: (in [0,1])

$$p(x) = 5x^4$$

- CDF: $F(x) = \int_0^x p(t)dt = x^5$

- Samples: $x_i = F^{-1}(u_i) = \sqrt[5]{u_i}$

Monte Carlo with Importance Sampling

- Consider N random samples over domain **with probability *p(x)***
- Define estimator as:

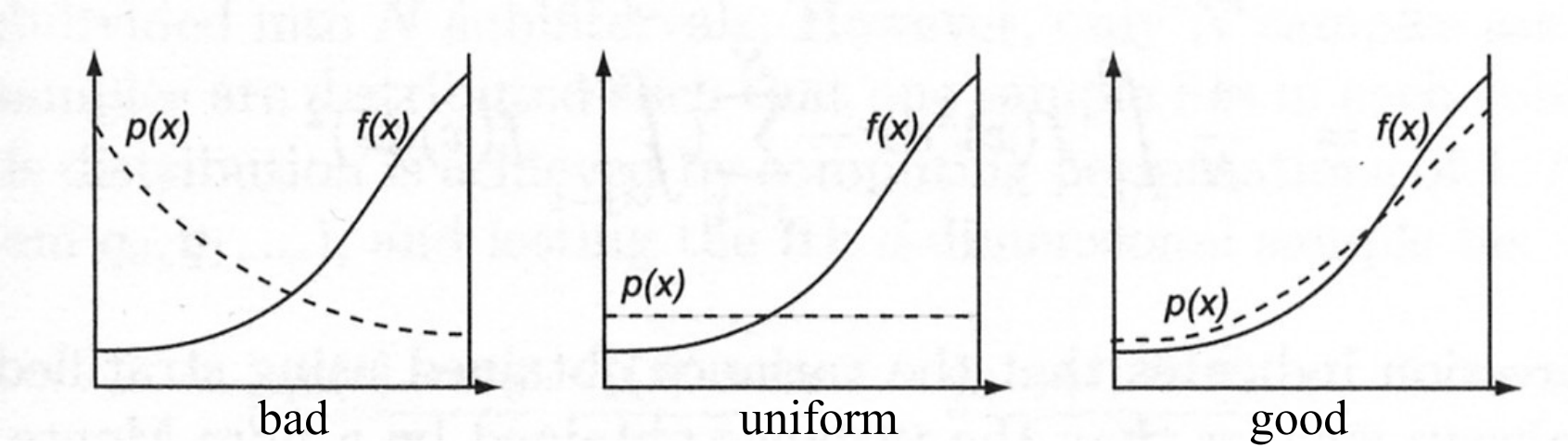
$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

- Probability *p* allows us to sample the domain more smartly

Importance sampling

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

- Choose *p* wisely to reduce variance
  - Use *p* that resembles *f*
  - Does not change convergence rate (still sqrt)

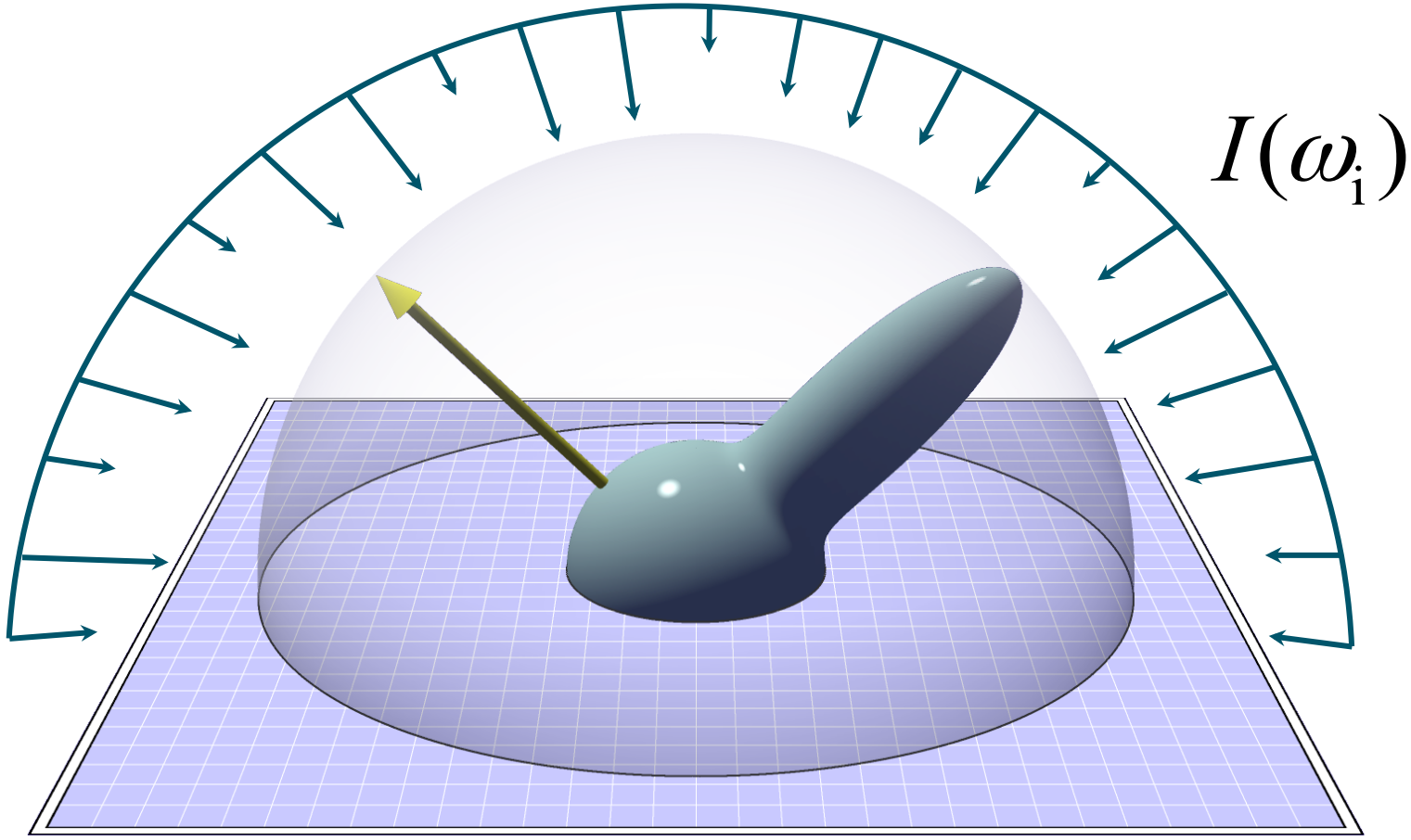


Example

- Evaluate: $I = \int_0^1 5x^4 dx$
- Use PDF $p(x) = 5x^4$
- Yields: $I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{5x^4}{5x^4} = 1$

Example: Glossy rendering

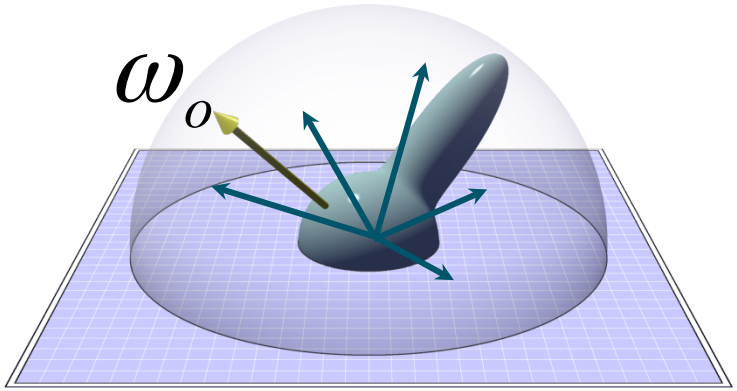
- Integrate over hemisphere
- BRDF times cosine times incoming light



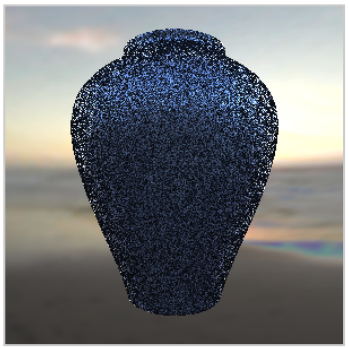
Slide courtesy of Jason Lawrence

Sampling a BRDF

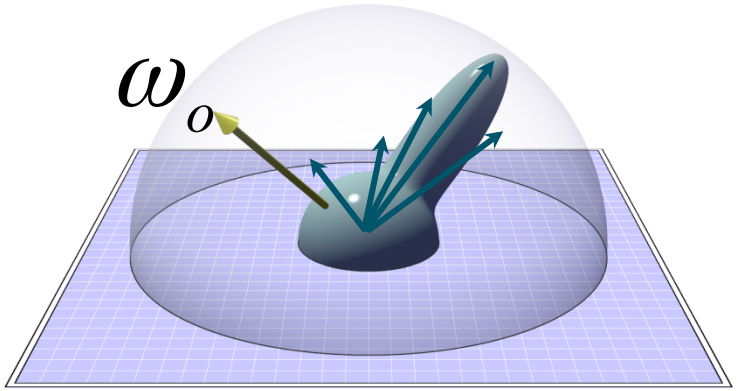
$U(\omega_i)$

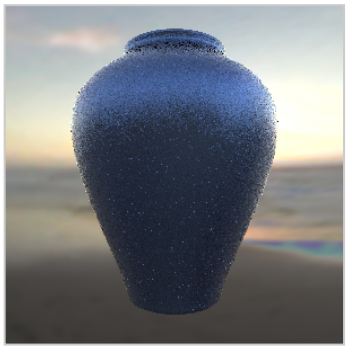


5 Samples/Pixel



$P(\omega_i)$

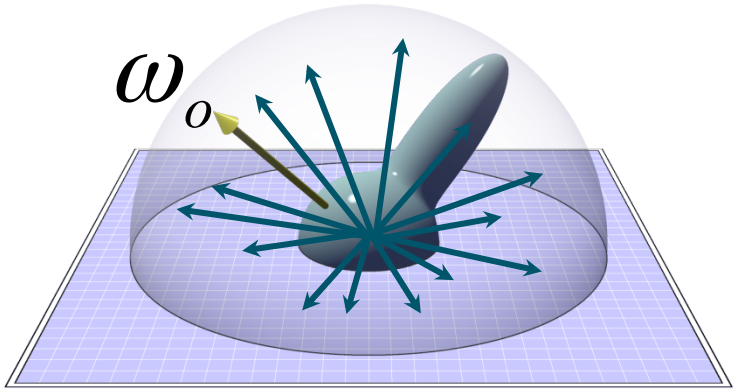




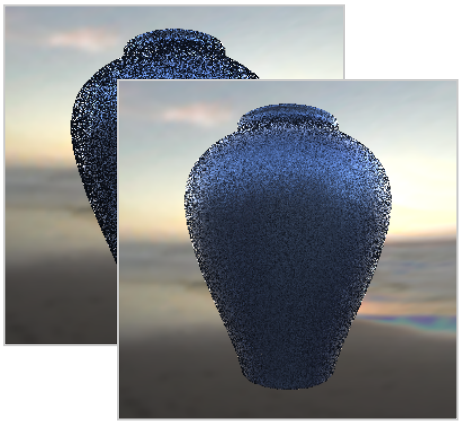
Slide courtesy of Jason Lawrence

Sampling a BRDF

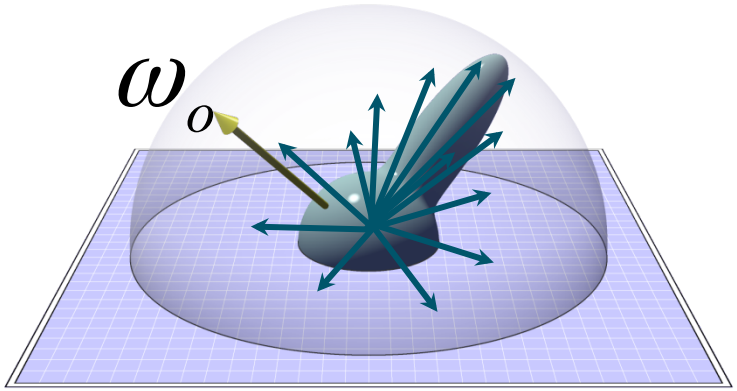
$U(\omega_i)$

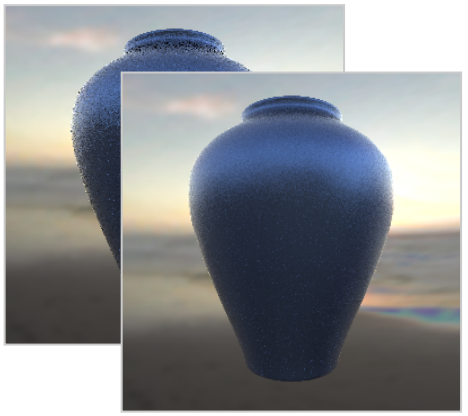


25 Samples/Pixel



$P(\omega_i)$

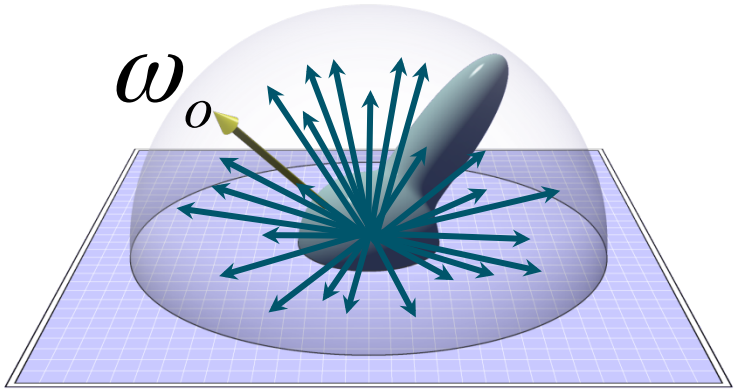




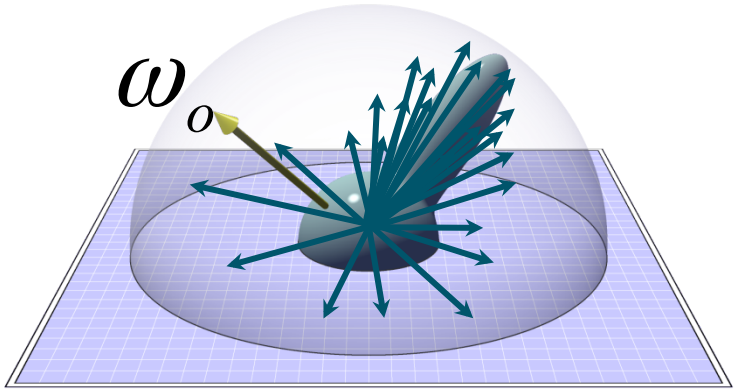
Slide courtesy of Jason Lawrence

Sampling a BRDF

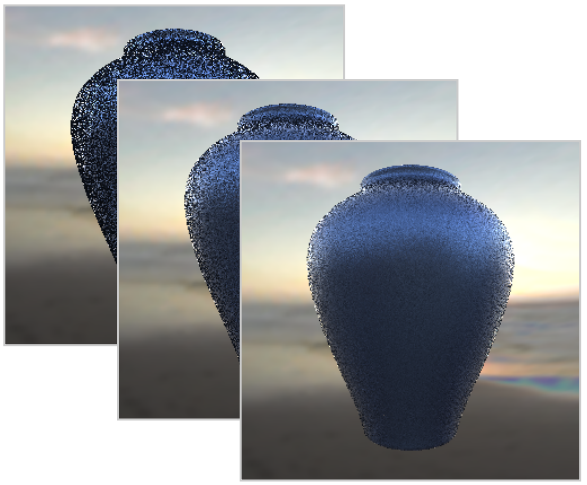
$U(\omega_i)$

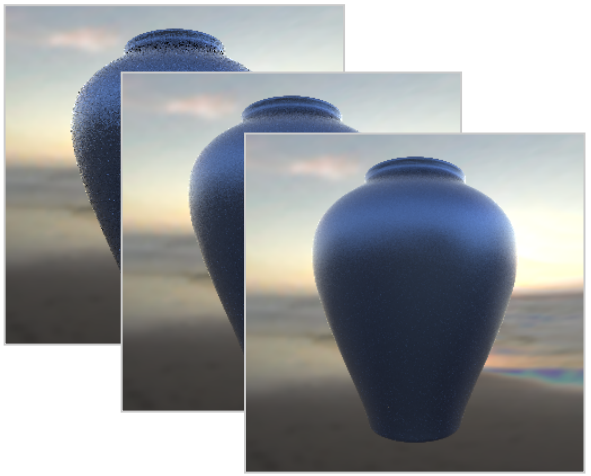


$P(\omega_i)$



75 Samples/Pixel





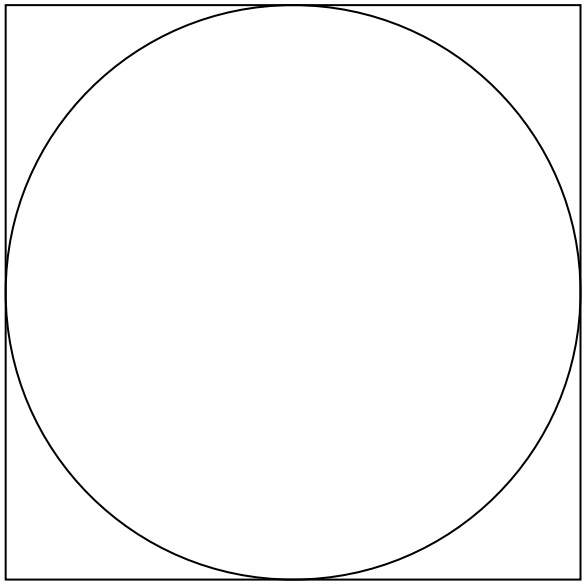
Slide courtesy of Jason Lawrence

Variance Reduction

- The convergence rate of Monte Carlo Integration can be quite slow.
- The standard error of the estimator $\propto 1/\sqrt{n}$
- The problem in MC methods is to find ways to reduce the variance.
- A standard technique is stratified sampling.

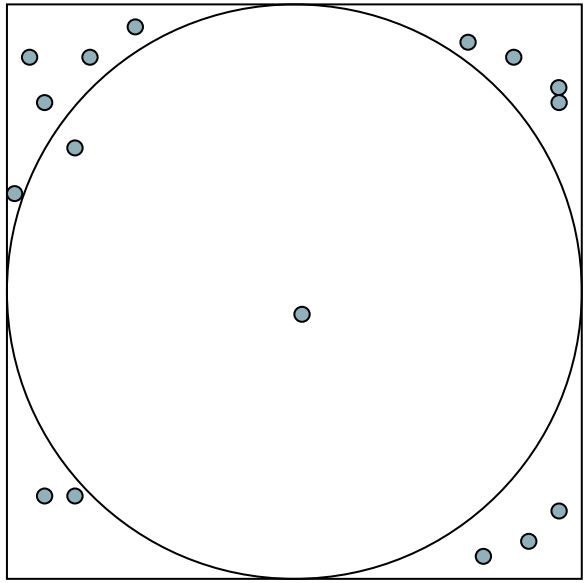
Example – Finding π

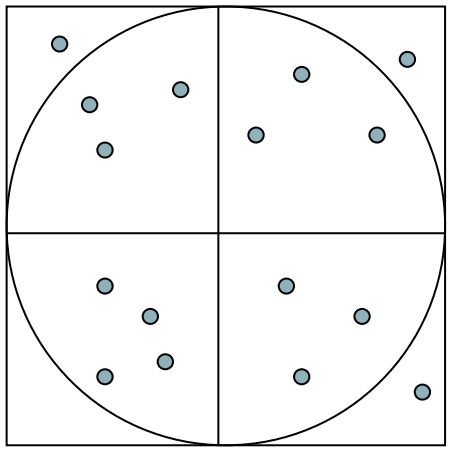
- Choose a random sample of n uniformly distributed points in the square of side 2
- Count how many (r) in the circle
- $r/n \rightarrow \pi/4$ with prob. 1 as $n \rightarrow \infty$



Stratified Sampling

- A uniform random sample is *random*!
- Each type of pattern is equally probable.
- A stratified sample, where we sample randomly within strata significantly reduces the variance.





stratified

Example

$$f(x) = e^{\sin(3x^2)}$$

N	I
1	2.75039
10	1.9893
100	1.79139
1000	1.75146
10000	1.77313
100000	1.77862

Unstratified

$$O(1/\sqrt{N})$$

$$f(x) = e^{\sin(3x^2)}$$

N	I
1	2.70457
10	1.72858
100	1.77925
1000	1.77606
10000	1.77610
100000	1.77610

Stratified

$$O(1/N)$$

Slide borrowed from Henrik Wann Jensen

Antialiasing in Ray Tracing

- In order to reduce aliasing due to undersampling in ray tracing each pixel may be sampled and then the average radiance per pixel found.

- A stratified sample over the pixel is preferable to a uniform sample – especially when the gradient within the pixel is sharply changing.

Random Numbers

- C/C++ have an inbuilt random number generator
- For example:

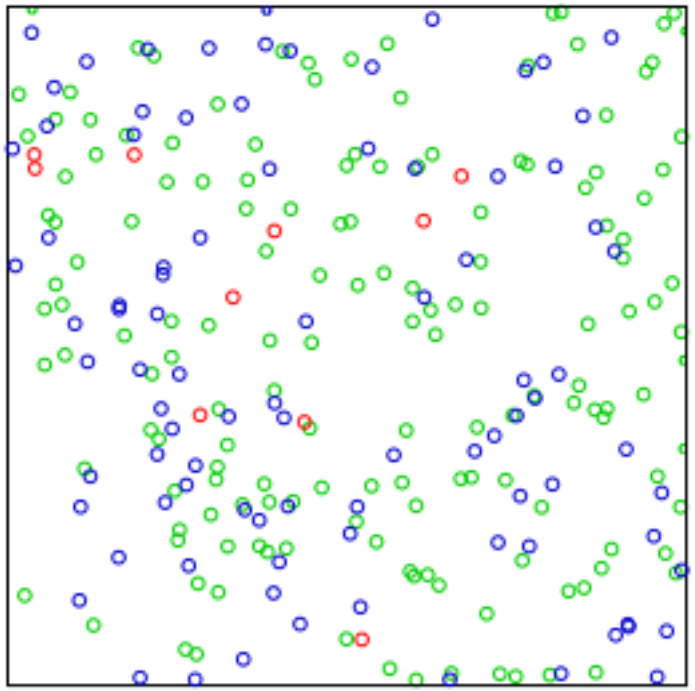
```
/*generates random number in range [0,1]*/
float uniform(void) {
      return((float)rand()/(float)(RAND_MAX));
}
```

- To choose an event with probability p, use

```
If(uniform() < p) /*do the event*/;
else /*don't*/;
```

Random Numbers

- Builtin random number generator
  - Pseudo random (it is a computer after all)
  - Usually not very good
  - Not stratified, etc.



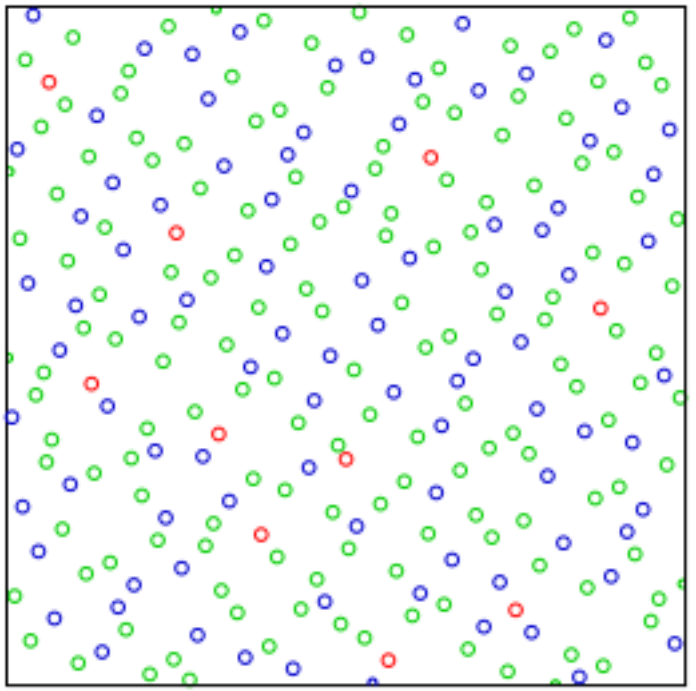
Random Numbers

- Better random numbers: *low discrepancy*
- Discrepancy:
  - Slide a small axis-aligned box across samples
  - Discrepancy: the variation in the number of samples among all boxes
- **Low** discrepancy:
  - Samples are everywhere, and mostly evenly distributed
- Note: low discrepancy by itself not necessarily good (samples can be regular and introduce aliasing)

Random Numbers

- Better (quasi-)random numbers: *Halton* sequence

```
FUNCTION Halton(index, base)
  BEGIN
      result = 0;
      f = 1 / base;
      i = index;
      WHILE (i > 0)
      BEGIN
          result = result + f * (i % base);
          i = FLOOR(i / base);
          f = f / base;
      END
      RETURN result;
  END
```



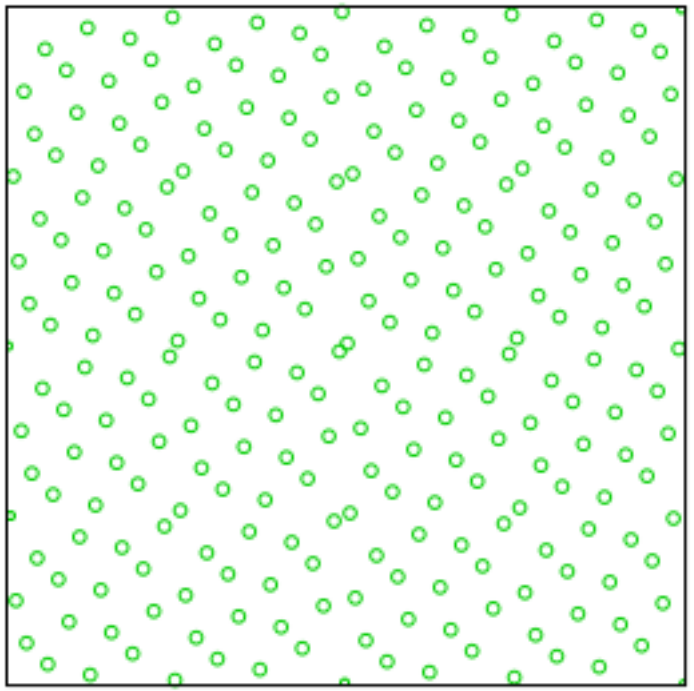
Random Numbers

- Better (quasi-)random numbers: *Hammersley* set

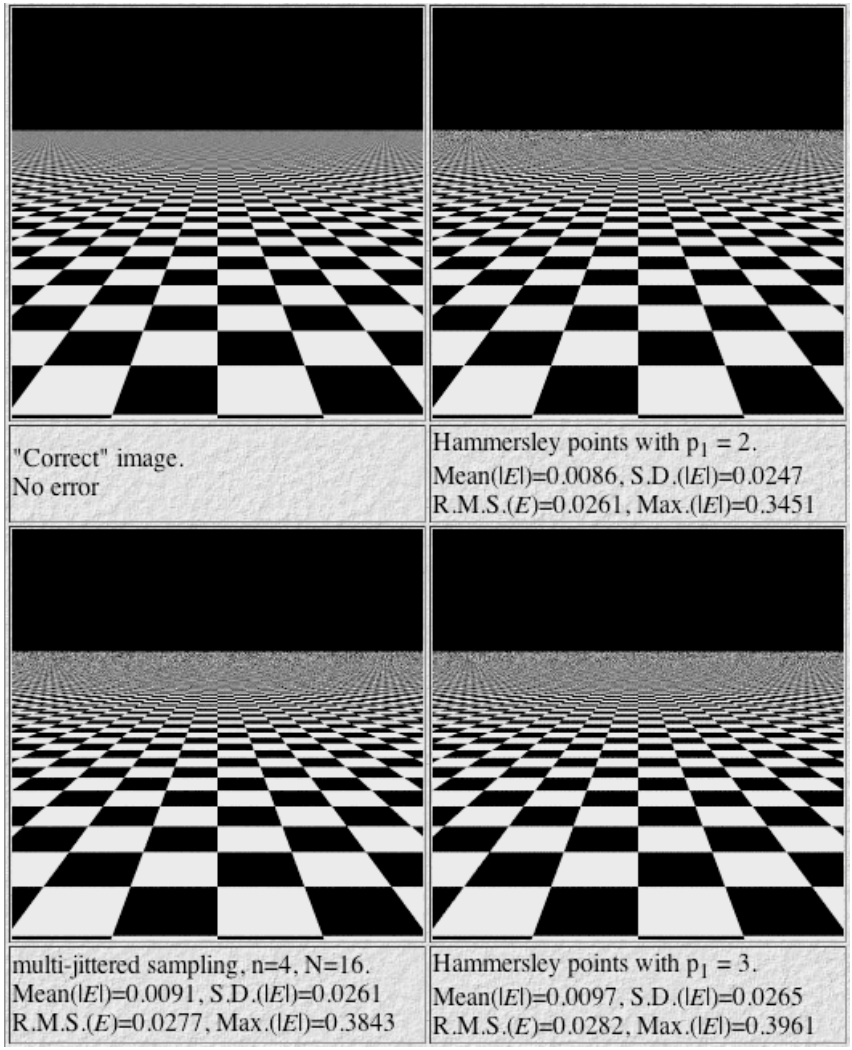
```
void PlaneHammersley(float *result, int n) {
  float p, u, v;
  int k, kk, pos;
  for (k=0, pos=0; k<n ; k++) {
    u = 0;

    for (p=0.5, kk=k; kk; p*=0.5, kk>>=1)
      if(kk&1) u += p;       // kk mod 2 == 1

    v = (k + 0.5) / n;
    result[pos++] = u;
    result[pos++] = v;
  }
}
```



Random Numbers – Differences in Rendering



"Correct" image.
No error

Hammersley points with $p_1 = 2$.
Mean(|E|)=0.0086, S.D.(|E|)=0.0247
R.M.S.(E)=0.0261, Max.(|E|)=0.3451

multi-jittered sampling, n=4, N=16.
Mean(|E|)=0.0091, S.D.(|E|)=0.0261
R.M.S.(E)=0.0277, Max.(|E|)=0.3843

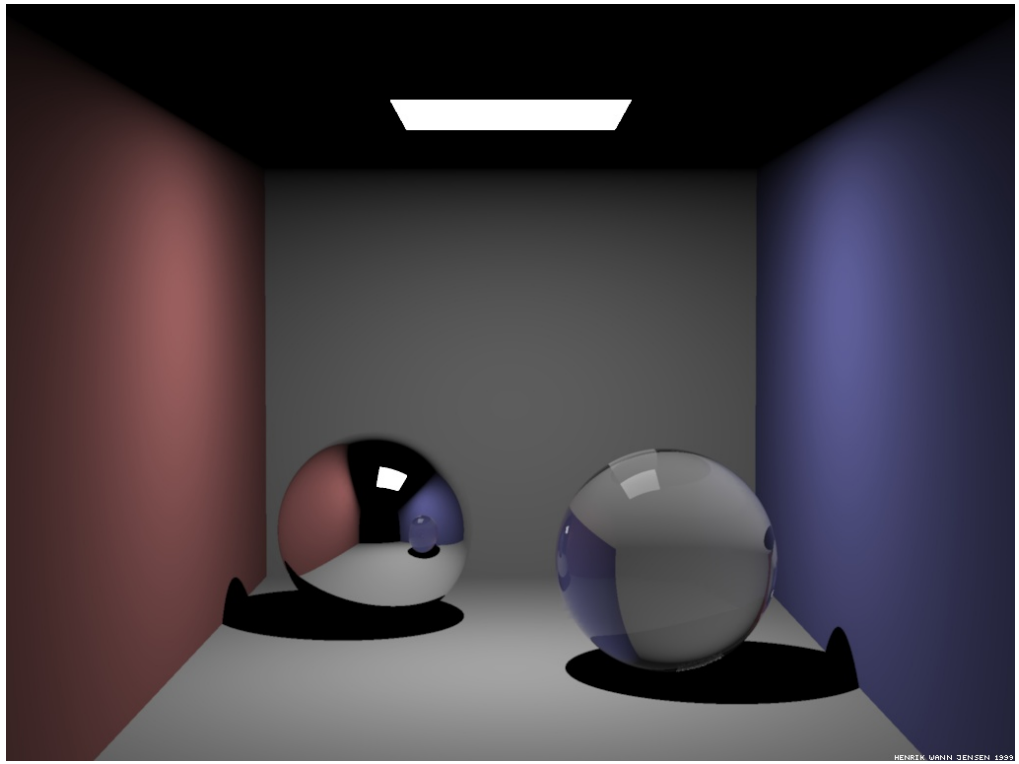
Hammersley points with $p_1 = 3$.
Mean(|E|)=0.0097, S.D.(|E|)=0.0265
R.M.S.(E)=0.0282, Max.(|E|)=0.3961

PATH TRACING

Ray Tracing Recap

- We saw from the operator expansion of the radiance equation that $L(p,\omega)$ expands into a series of paths
- In ray tracing we start from the eye and trace rays backwards from surface to surface, only following specular paths, terminating at non-specular surfaces
- From each point we obtain the local contribution of the light source to that point, plus the global contribution of the (specular) influence of other surfaces

Typical Ray Traced Image

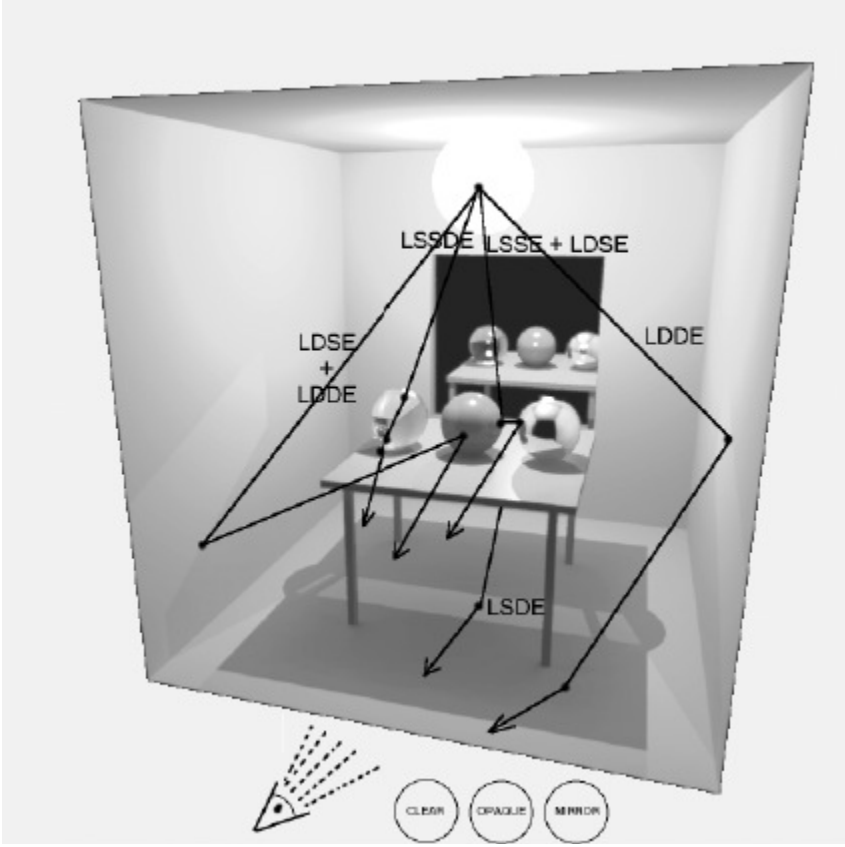


- http://graphics.stanford.edu/~henrik/images/global.html

Looking at Paths – Heckbert Notation

- Path are written as regular expressions
- **L**ight, **D**iffuse, **S**pecular and **E**ye
- Examples:
  - Ray tracing: LD[S*]E
  - Radiosity: LD*E
- Complete global illumination solution: L(D|S)*E

Heckbert Notation examples

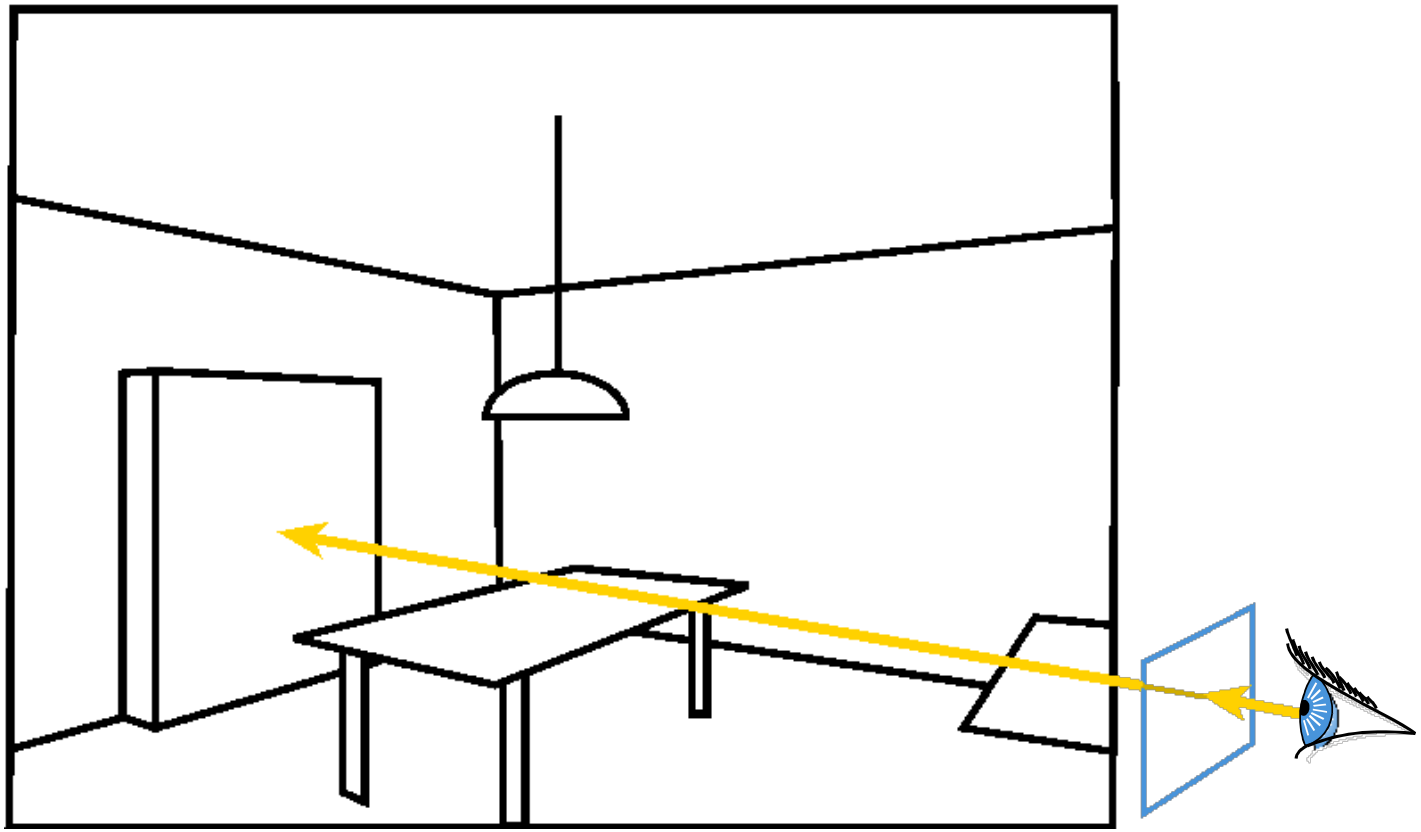


Ray Tracing

- The light paths expand in a tree-form
  - Each intersection of a ray with a surface can spawn two further rays (reflected + refracted) which each can spawn two further rays and so on.
- The contribution to the final image of deep layers of the tree can be very little, yet they still contribute very much to the time.

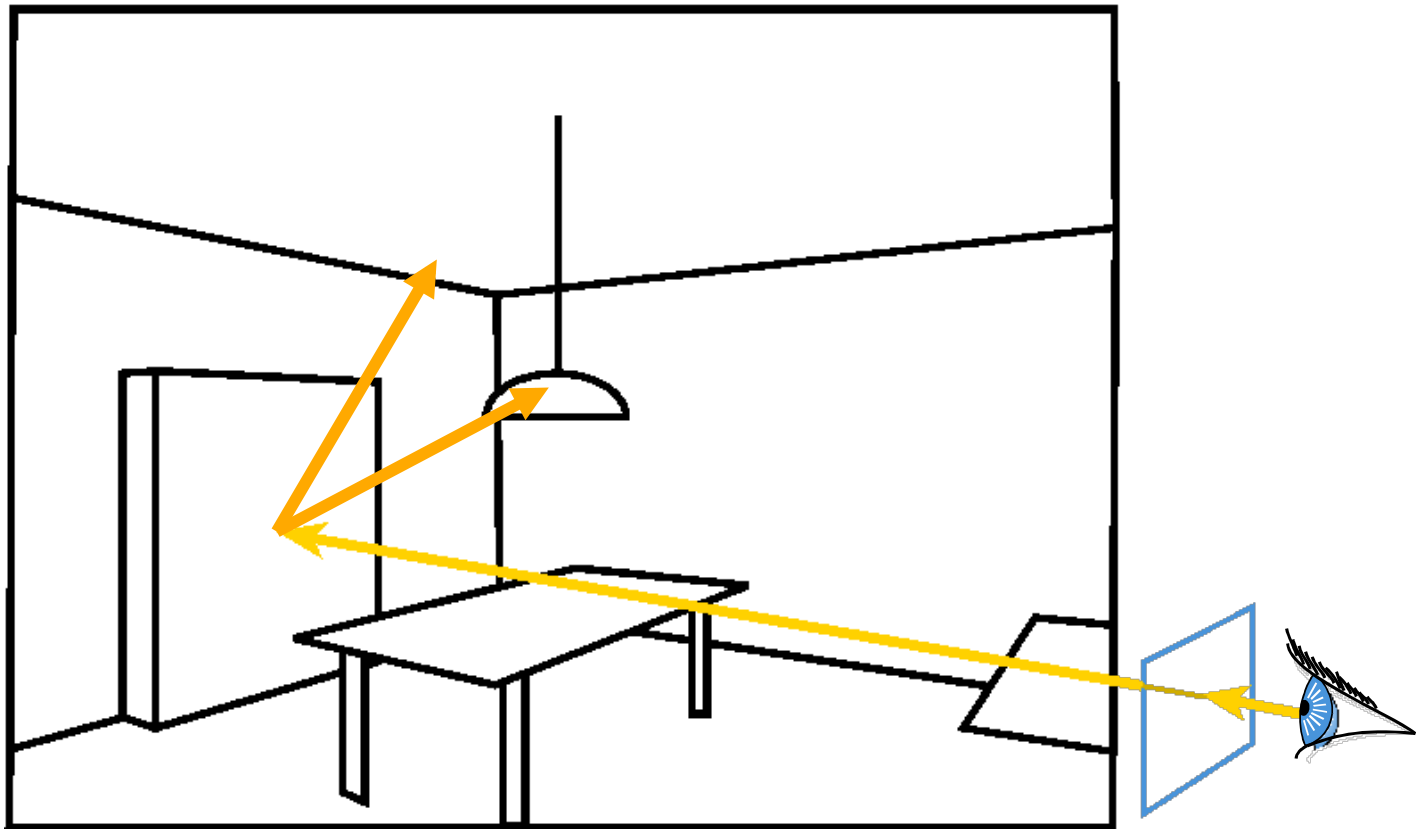
Ray Casting

- Cast a ray from the eye through each pixel



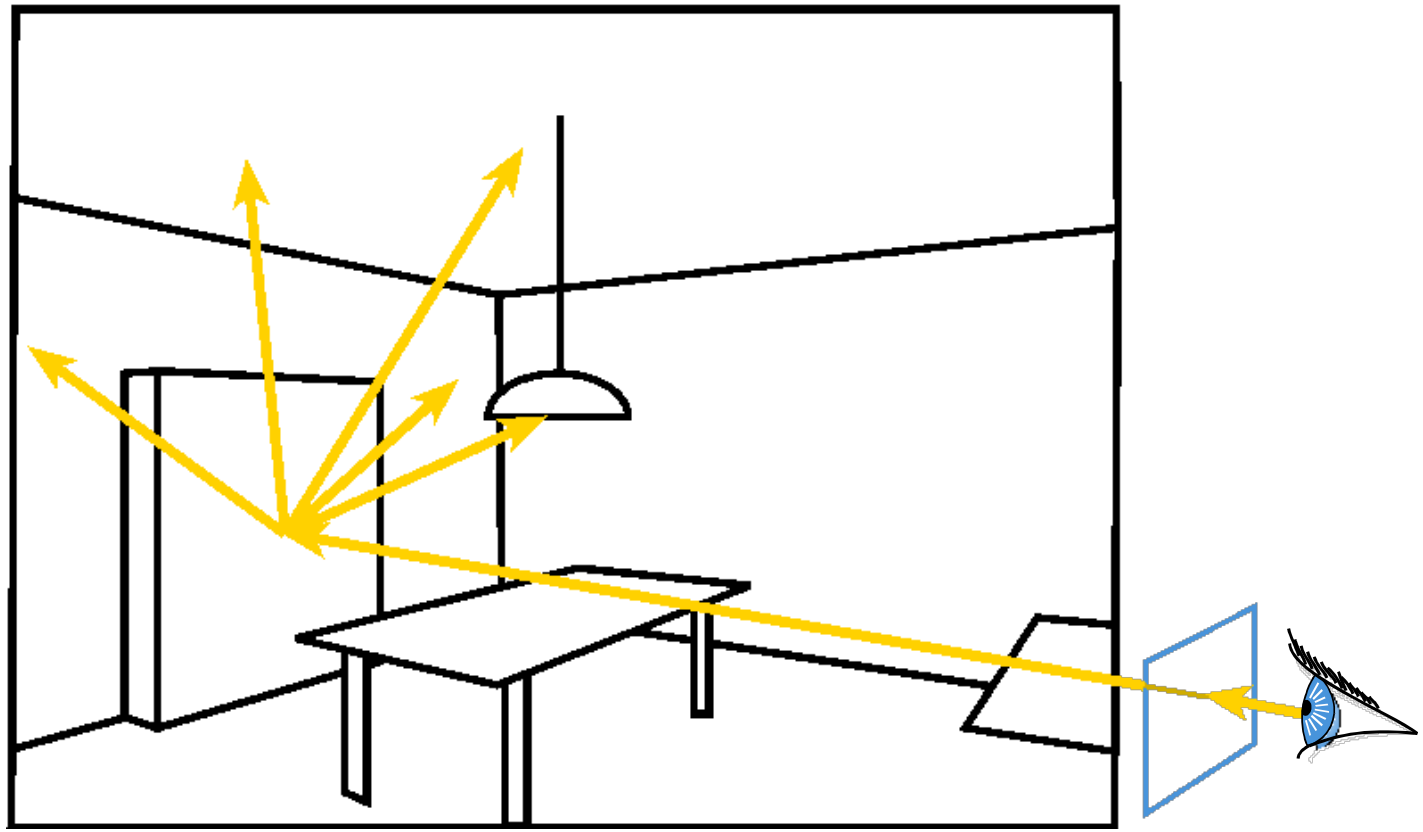
Ray Tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction)



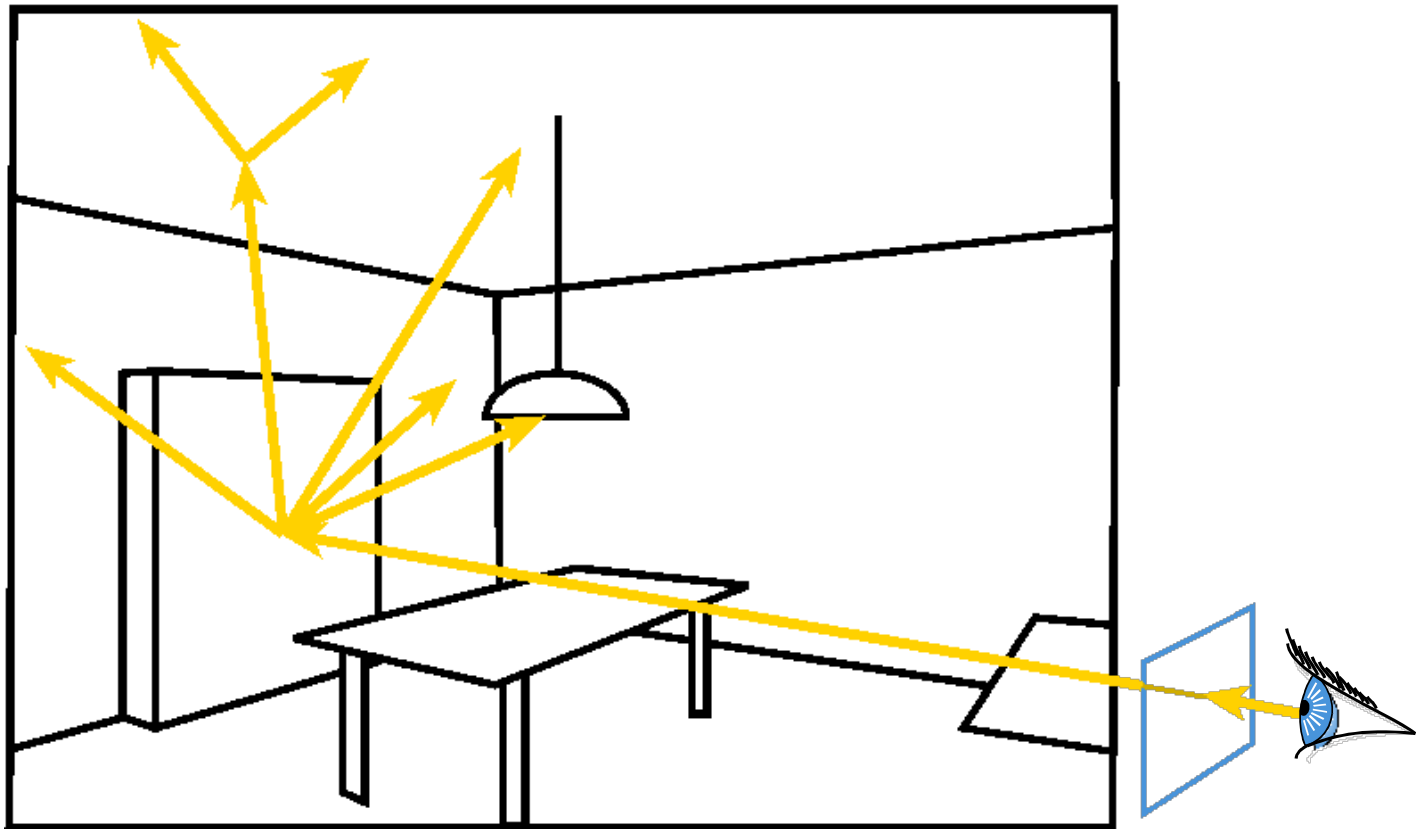
Monte-Carlo Ray Tracing (Distribution Ray Tracing)

- Cast a ray from the eye through each pixel
- Cast many random rays from the visible point
  - Accumulate radiance contribution



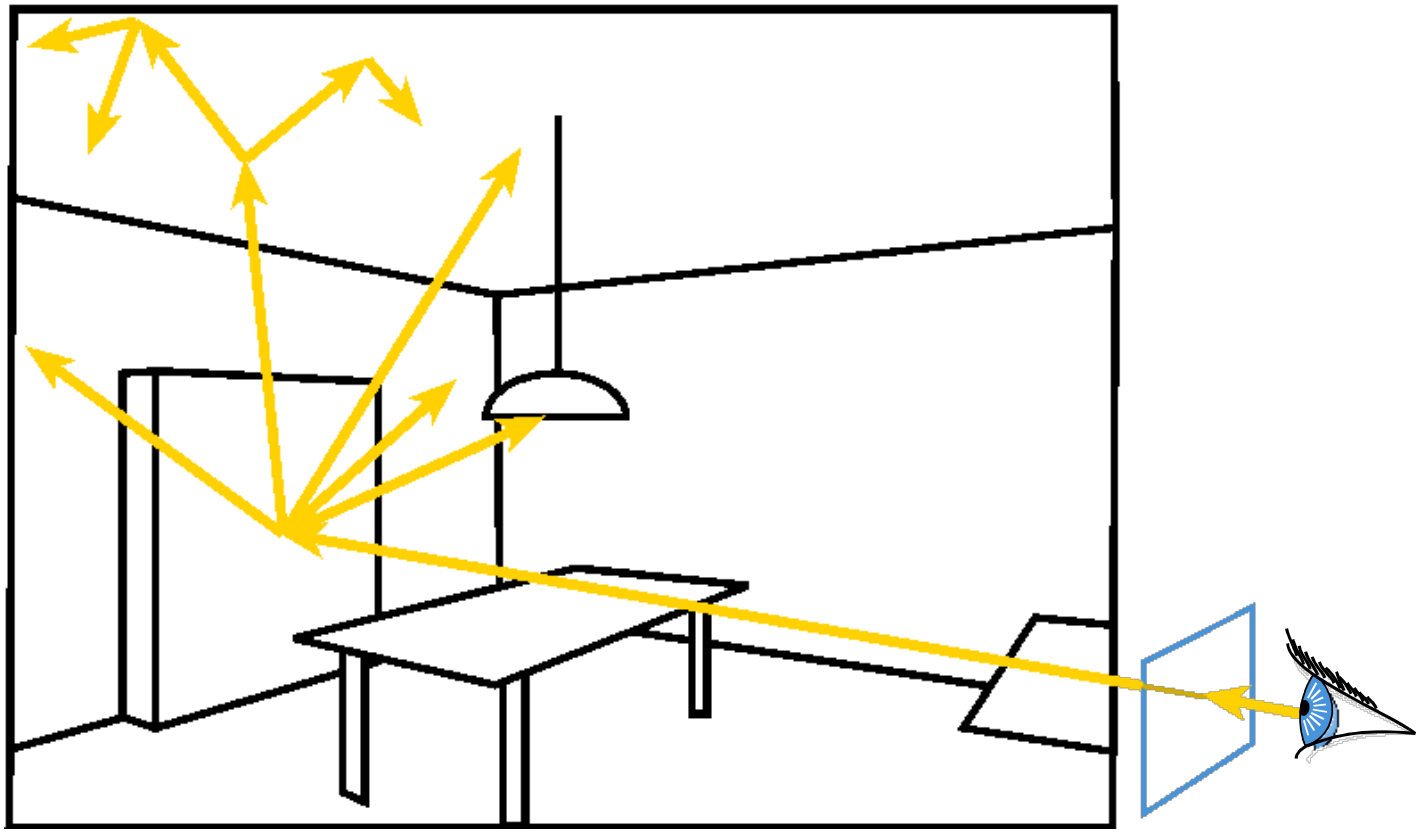
Monte-Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast many random rays from the visible point
- Recurse



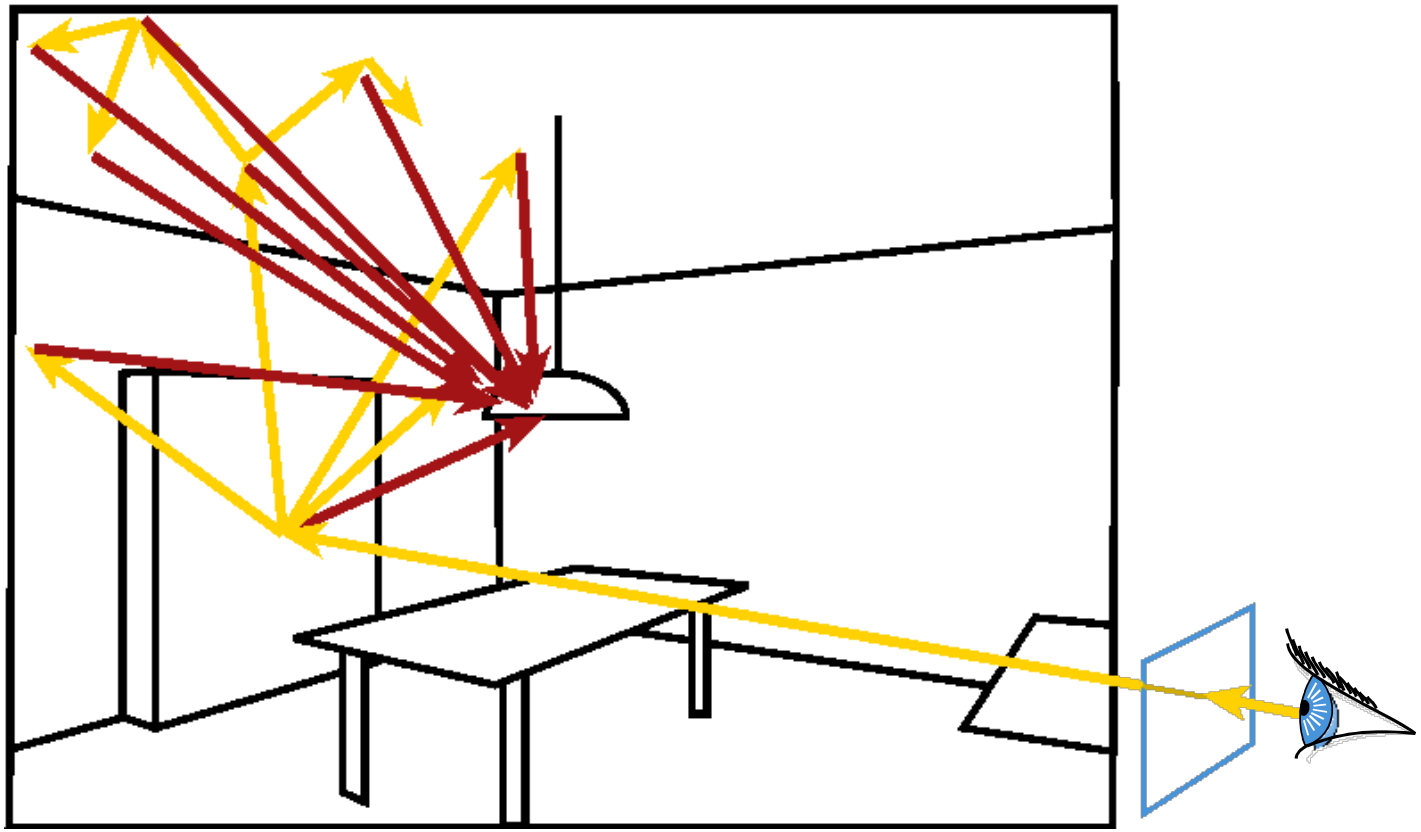
Monte-Carlo Ray Tracing

- Cast a ray from the eye through each pixel
- Cast many random rays from the visible point
- Recurse

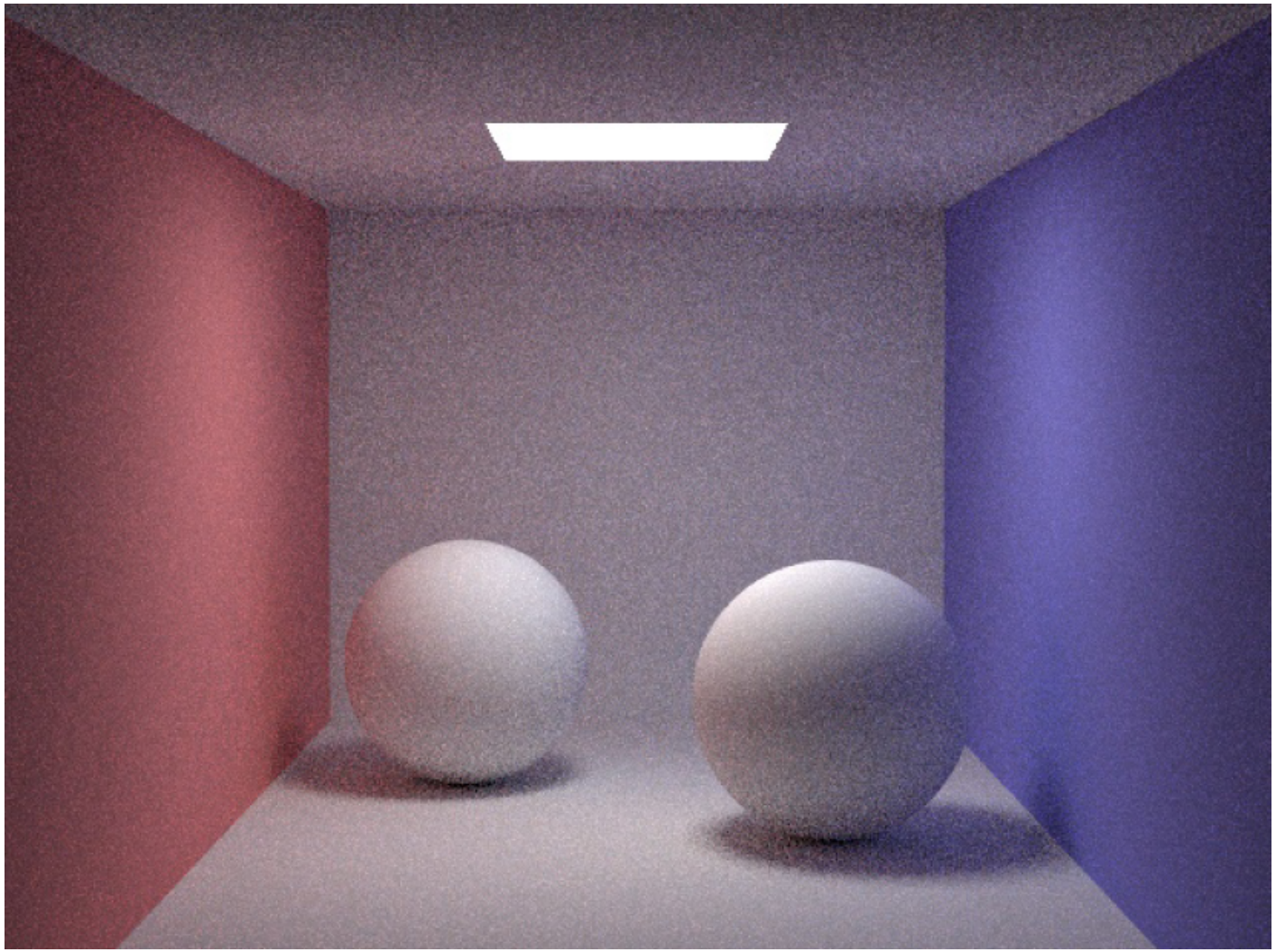


Monte-Carlo Ray Tracing

- Systematically sample primary light



Results



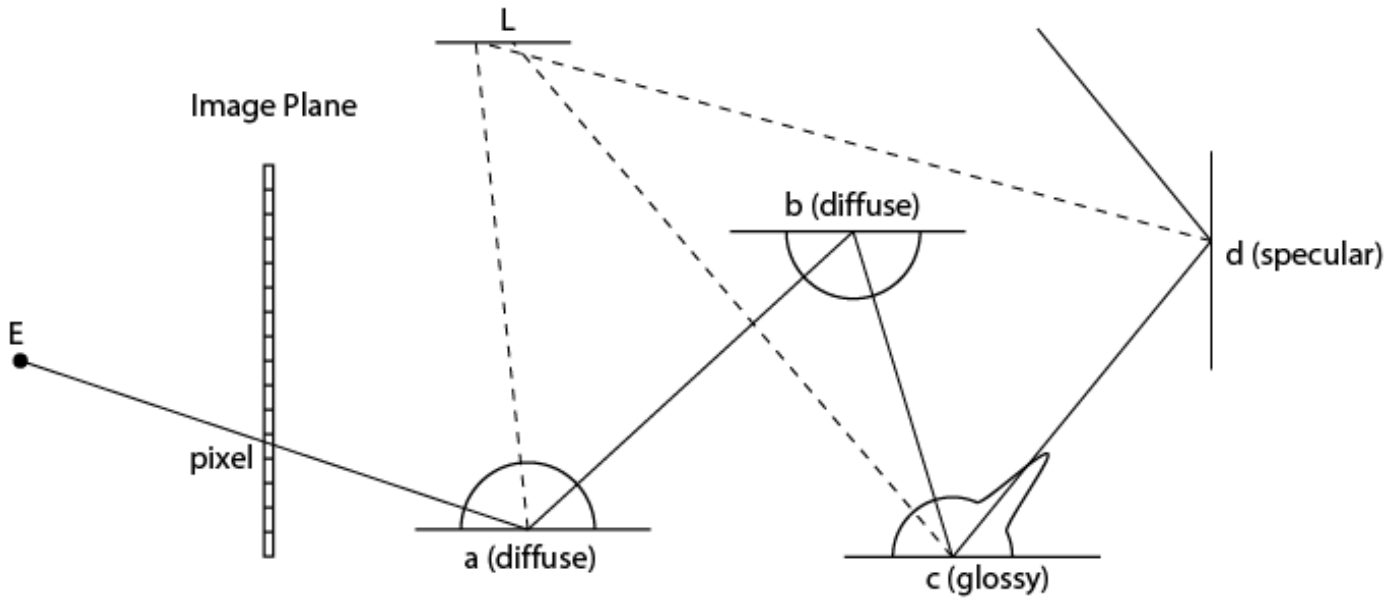
Monte-Carlo Ray Tracing

- Problem:
  - Exponential explosion of rays
  - Nearly impossible to recurse many times
- Solution:
  - Path Tracing

PATH TRACING

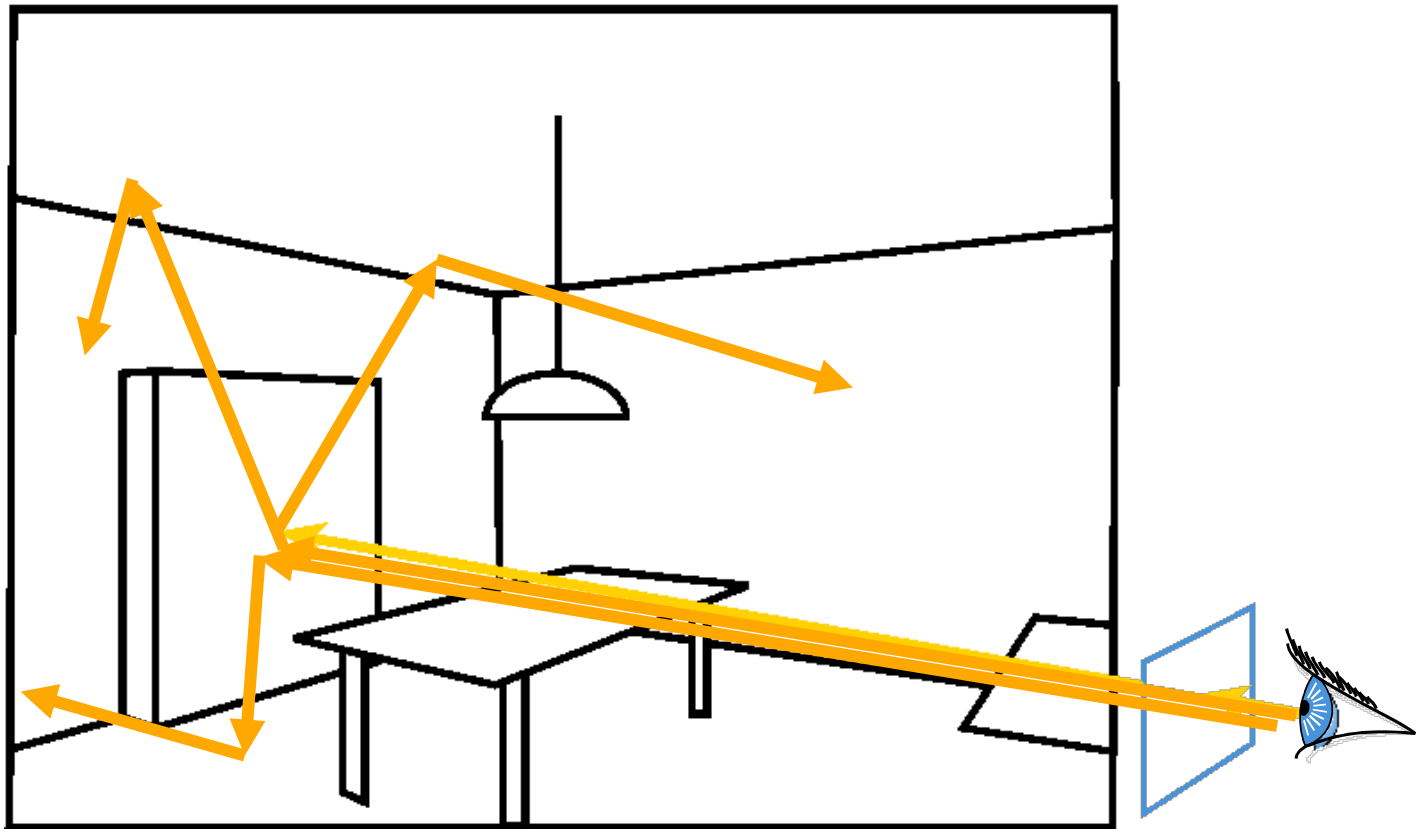
Path Tracing

- We generate paths of the form L(D|S)*E and each new ray is chosen stochastically according to the material properties (BRDF) of the surface from which it is generated.



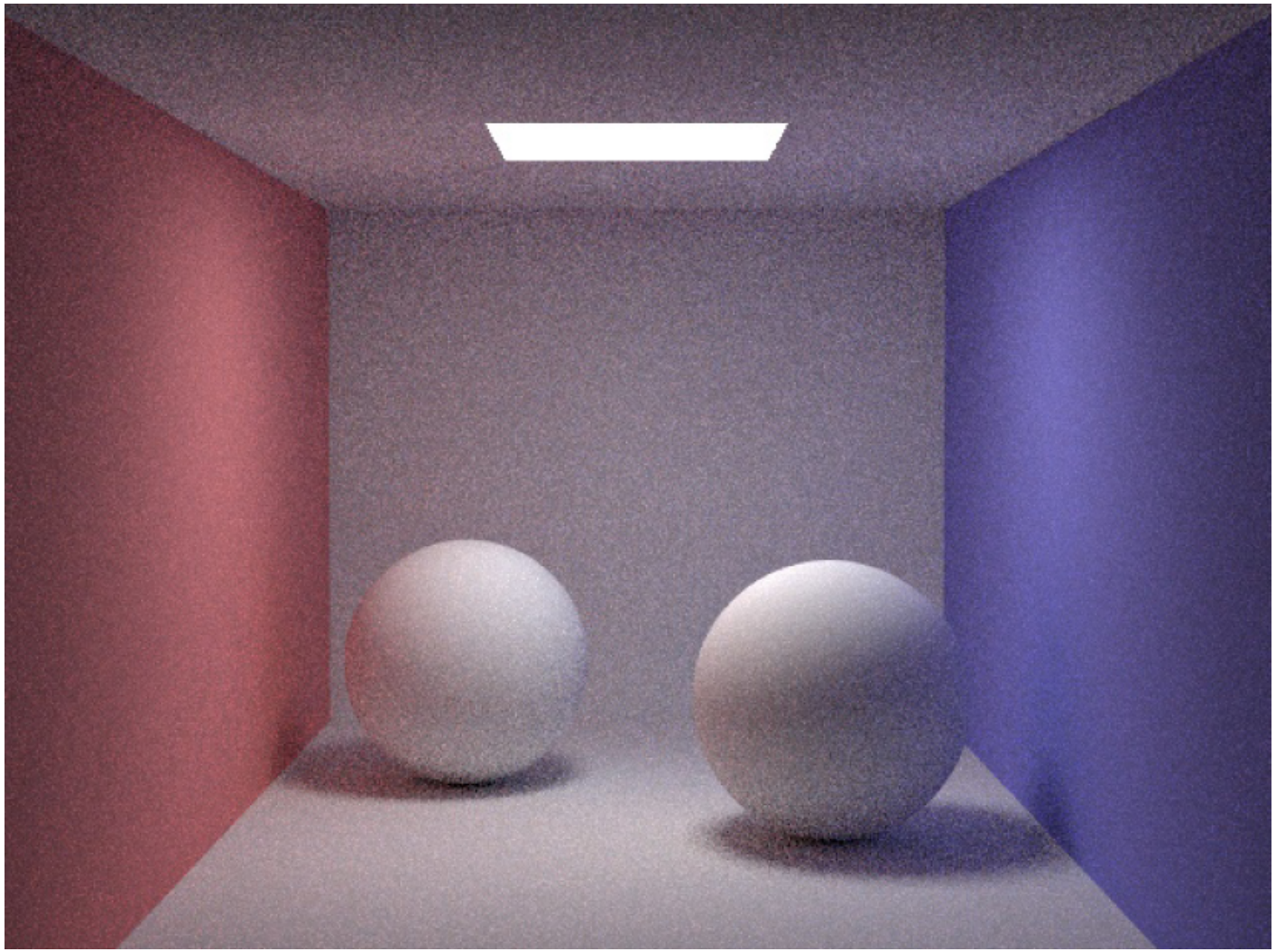
Monte Carlo Path Tracing

- Trace only ***one*** secondary ray per recursion
- But send many primary rays per pixel (performs antialiasing as well)



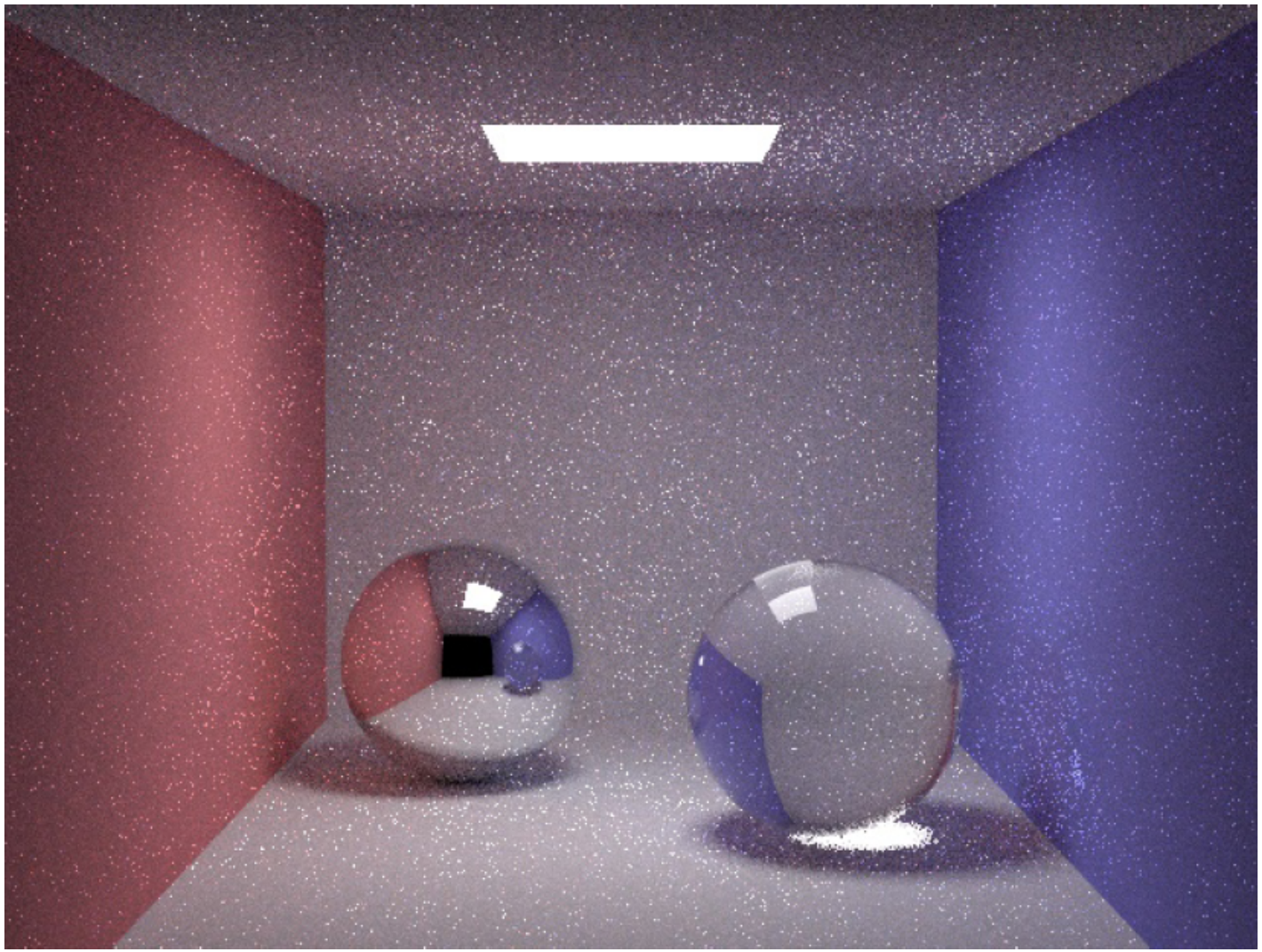
Results

Think about it : we compute an infinite-dimensional integral with 10 samples!!!



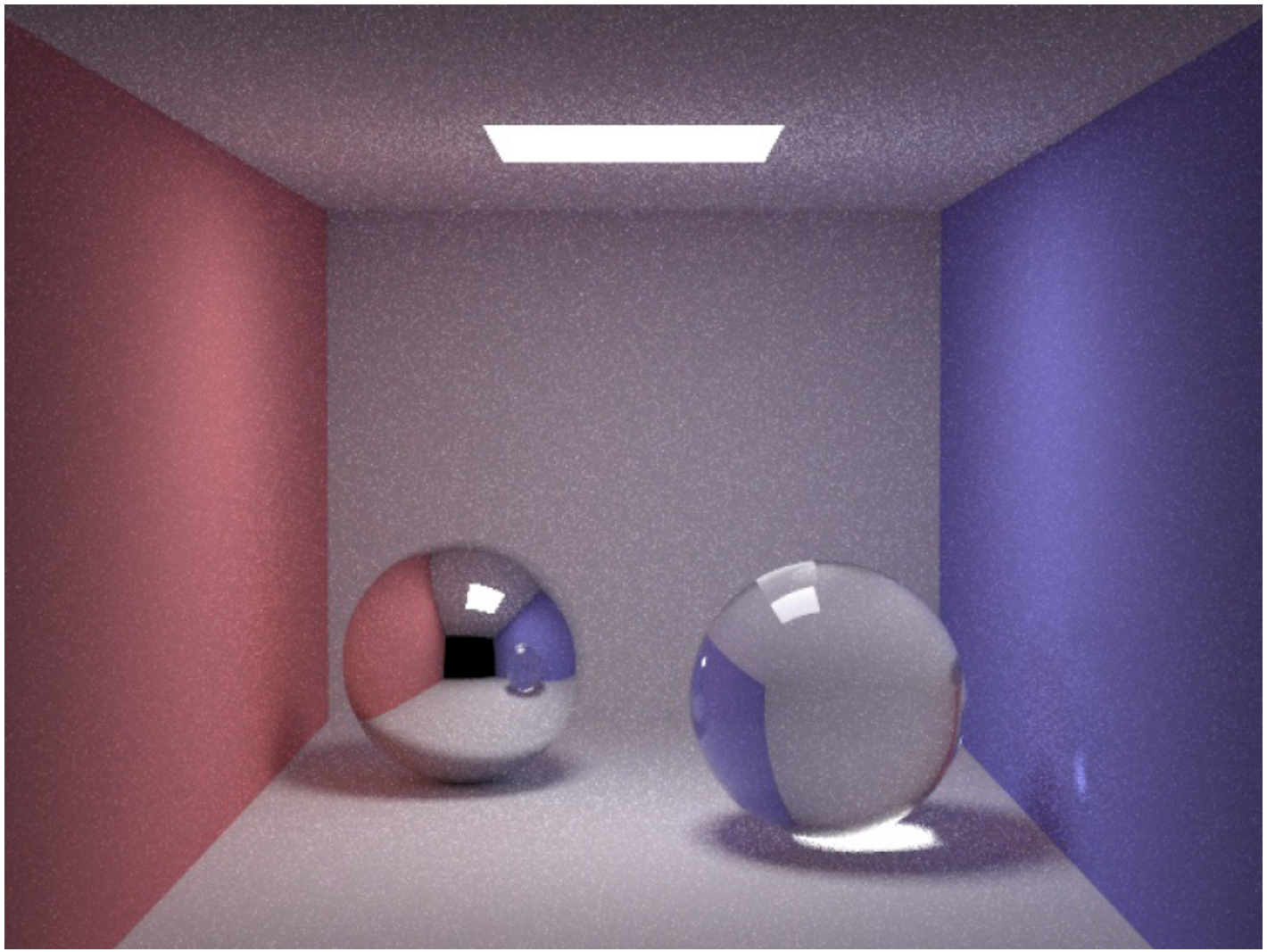
10 paths/pixel

Results: Glossy



10 paths/pixel

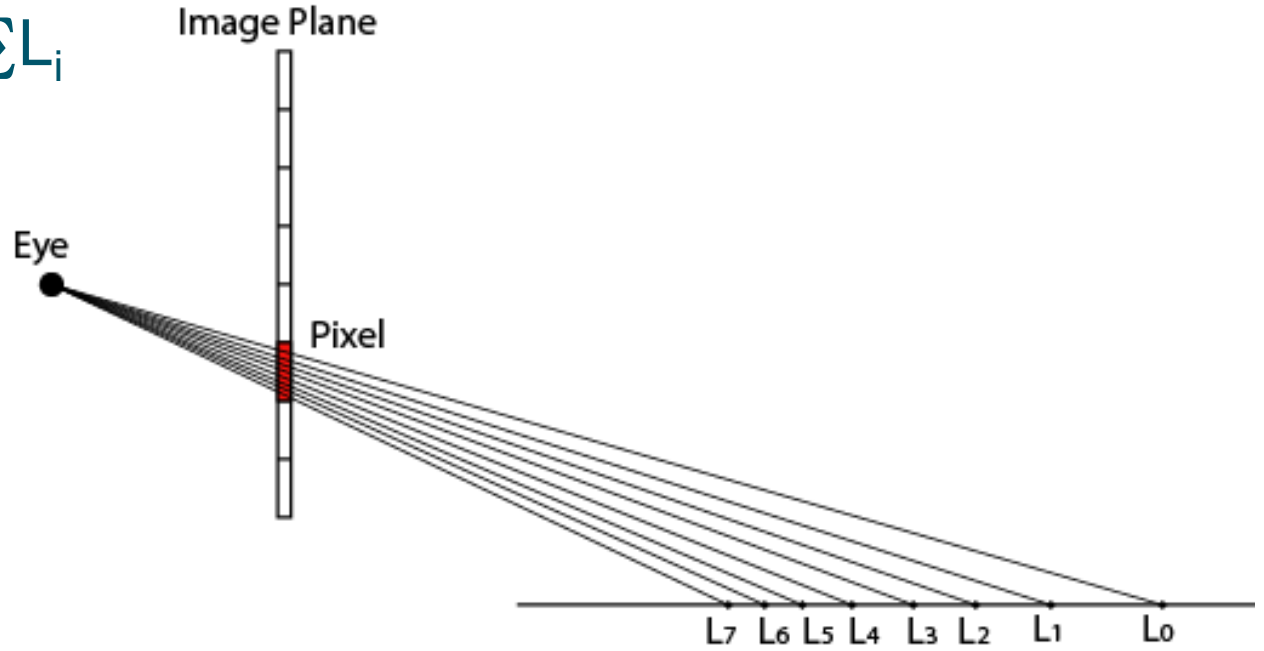
Results: Glossy



100 paths/pixel

Path Tracing

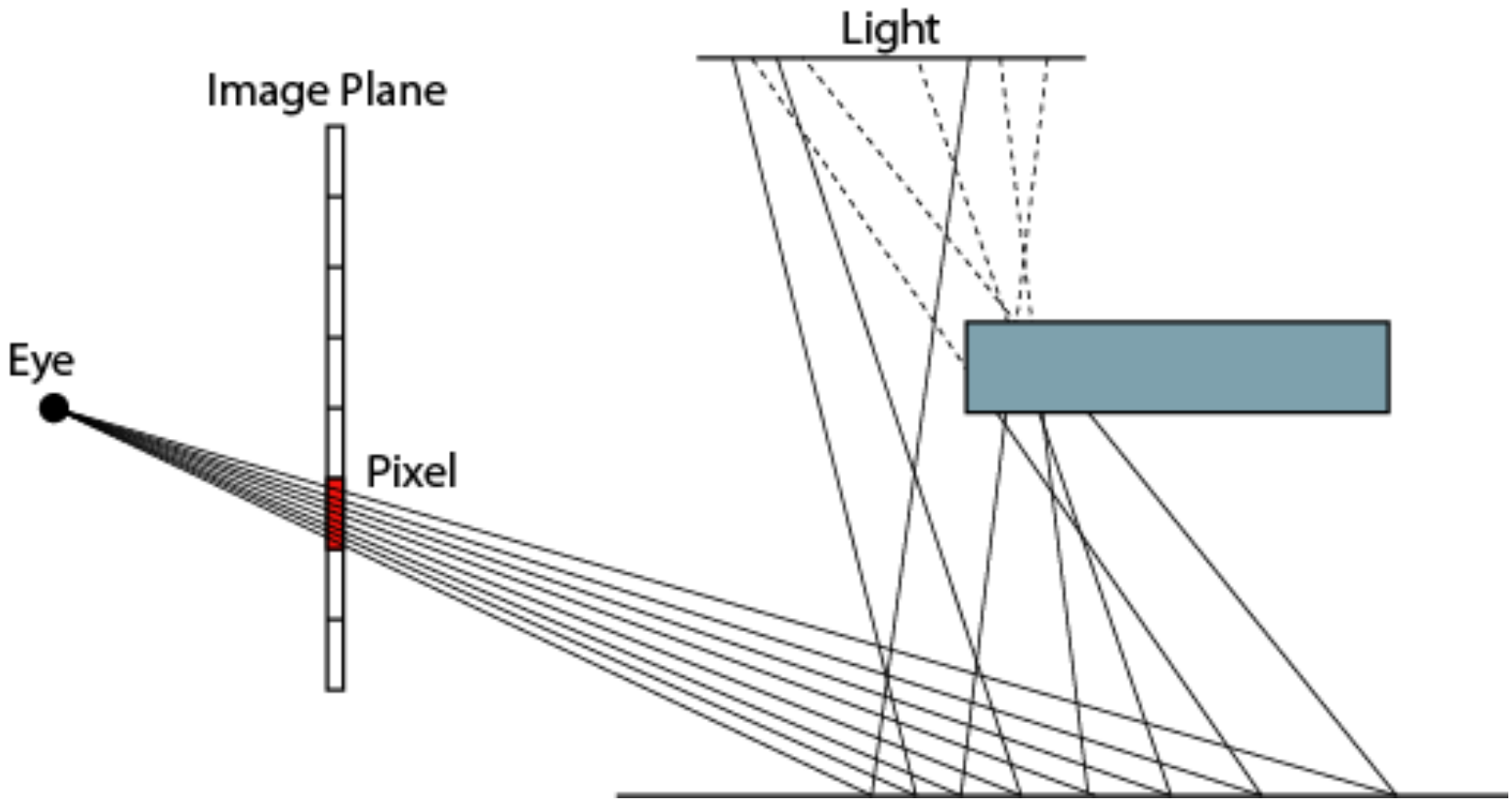
- We take n such rays through a pixel
- Suppose the i-th ray contributes L_i
- Then the overall estimator for the pixel is
  - $L^* = (1/N)\sum L_i$



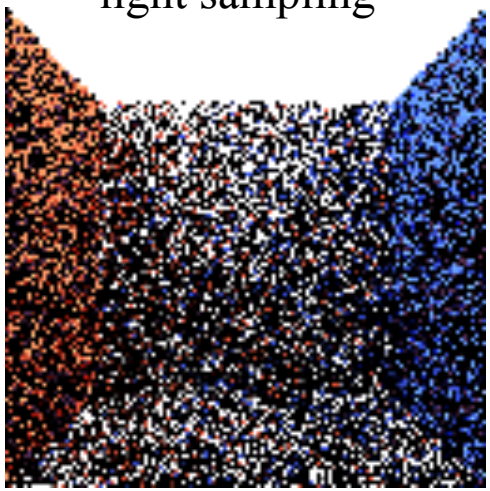
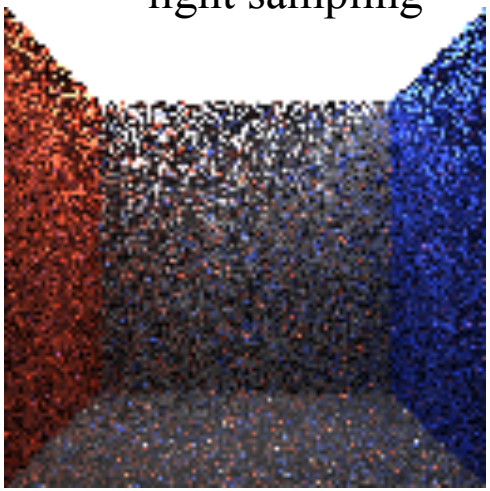
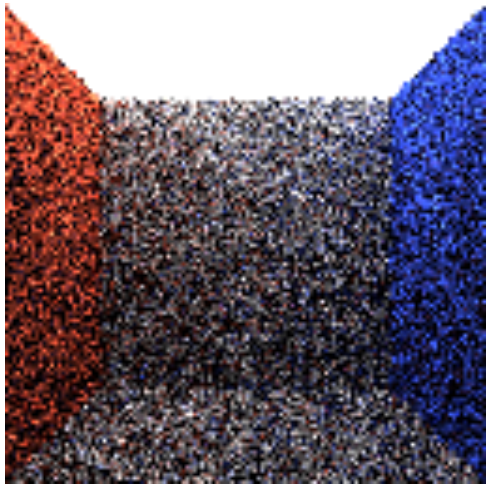
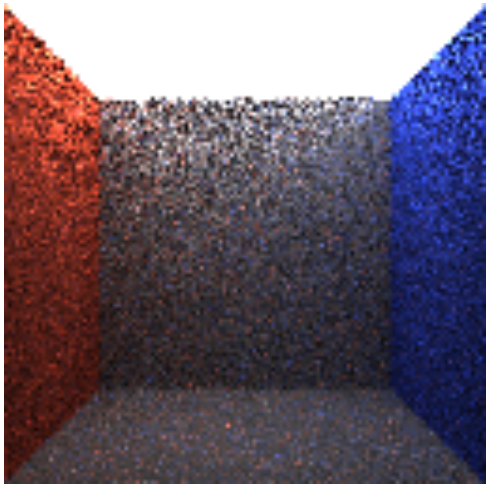
Area Light Sources

- Ray tracing assumes point lights
- Path tracing uses area lights
- Area lights are stochastically sampled
- With diffuse emitters we can uniformly sample a point on the light and shoot the ray to that point
- If the ray is not occluded we compute the local contribution from that point
- This will result in shadow penumbras (which are not possible with standard ray tracing)

Area Light Sources



Importance of sampling the light

	Without explicit light sampling	With explicit light sampling
1 path per pixel		
4 path per pixel		

Importance Sampling Multiple Area Lights

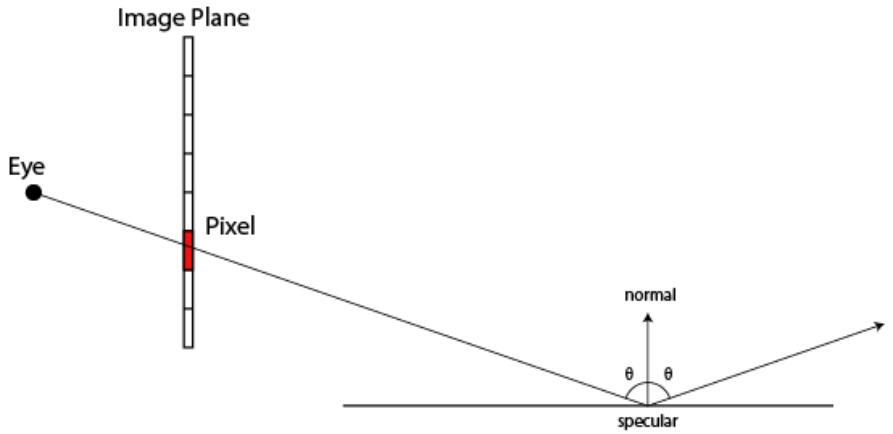
- How would you do that?

Sampling the BRDF

- Here we assume a very simple form of BRDF
  - The surface is perfectly diffuse (k_{dr}, k_{dg}, k_{db})
  - The surface is perfectly specular (k_{sr}, k_{sg}, k_{sb})
  - It is a ‘mixture’ of the above.
  - We must have $(0,0,0) \leq (k_{dr}, k_{dg}, k_{db}) + (k_{sr}, k_{sg}, k_{sb}) \leq (1,1,1)$

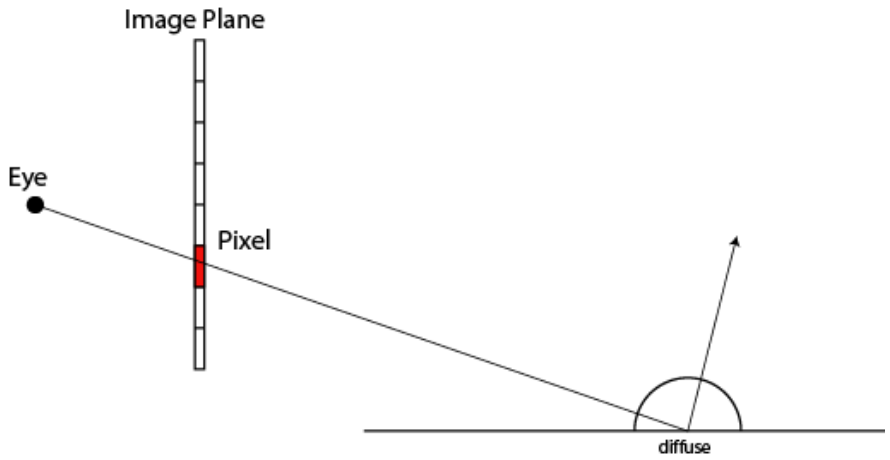
Sampling the BRDF - Specular

- The ray must follow the direction of specular reflection
- Reflect incoming ray about the normal



Sampling the BRDF - Diffuse

- Choose a random point on the hemisphere and this gives the direction for the next ray
- Can also do importance sampling, choose direction according to cosine distribution



Sampling the BRDF - Mixed

- Choose whether the next ray is to follow the specular or the diffuse path
- Let $k_d = k_{dr} + k_{dg} + k_{db}$
- Let $k_s = k_{sr} + k_{sg} + k_{sb}$
- Let $p = k_d / (k_d + k_s)$
- Then with probability p choose the diffuse path or with 1-p choose the specular path

Sampling the Hemisphere (for diffuse)

- Rejection sampling:
  - Generate uniformly random (x,y,z) with
  - $-1 \leq x \leq 1$
  - $-1 \leq y \leq 1$
  - $0 \leq z \leq 1$
- Reject this if $x^2 + y^2 + z^2 > 1$
- Otherwise use normalise(x,y,z)

Spherical Coordinates

- $x = \sin\phi \cos\theta$
- $y = \sin\phi \sin\theta$
- $z = \cos\phi$
- Let r_1 and r_2 be two uniform [0,1] observations
- $\theta = 2\pi r_1$
- $\phi = \arccos(r_2)$

Sampling a Convex Polygon

- Suppose the light source is a convex polygon and a random point must be chosen from this
- Rotate the polygon so that it is parallel to a principle plane (e.g., XY plane)
- Find the bounding rectangle
- Use rejection sampling to find uniformly random points within the polygon
- (Of course this requires clipping)

Sampling a Convex Polygon

- Another approach is to use barycentric coordinates
- Suppose the vertices of the polygon are $p_1,\ldots,p_k$
- Then for any point p in the polygon:
- $p = \sum \alpha_i p_i$, for $\sum \alpha_i = 1$, $\alpha_i \geq 0$
- Choose the α_i at random in the interval [0,1] to satisfy the summation requirement (with rejection sampling)

Russian Roulette

- Our estimator is of the form:
- $L^* = \sum L_i$
- Each L_i requires the same computation but many terms may contribute very little to the final result
- Russian Roulette is a sampling scheme that maintains an unbiased estimator but terminating a path, such that on the average important paths are sampled

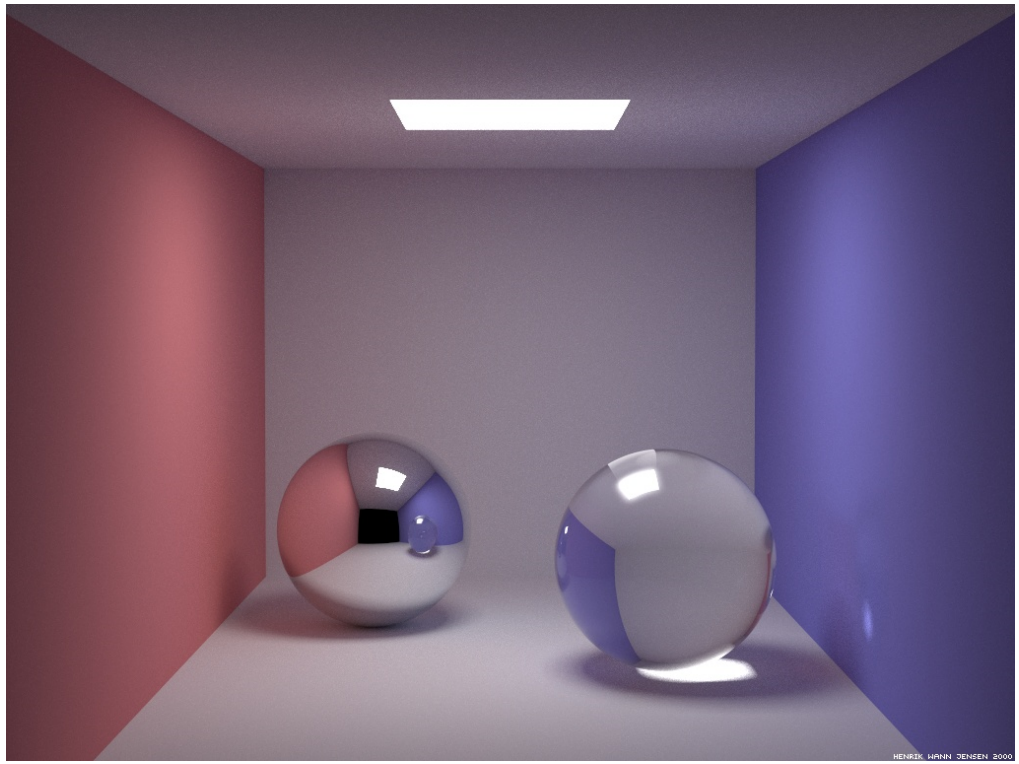
Russian Roulette

- For each term we decide whether or not to terminate
- Let q_i be a probability, and let
- $L^*_i = L_i/q_i$ with probability q_i or else 0 (termination)
  - Draw random number r in [0,1]
  - Check $r > q_i$ then terminate, otherwise continue
- $E(L^*_i) = (L_i/q_i)q_i = L_i$ so that the estimator is unbiased

Importance Sampling

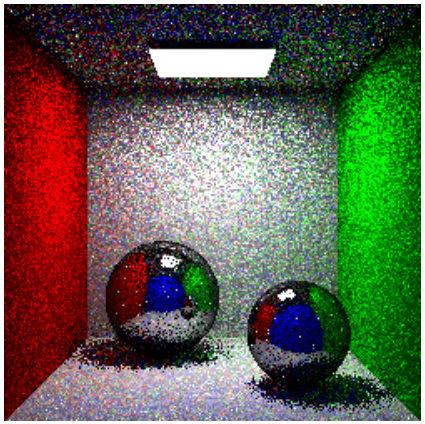
- In Monte Carlo integration the variance is reduced the closer the probability distribution p(x) is to f(x)
- This is known as importance sampling, where the probabilities are chosen to reflect the shape of the integrand
- In path tracing we implicitly use importance sampling since we are only following paths that are bound to contribute to the final image (since we are sampling the BRDF, and not following purely ‘random’ paths)

Path Traced Image

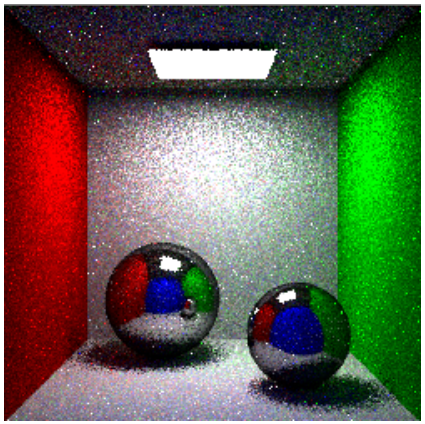


- http://graphics.stanford.edu/~henrik/images/global.html

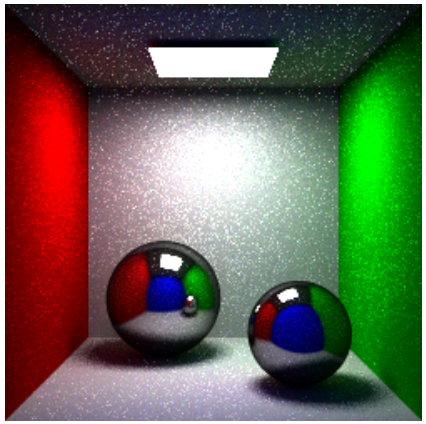
Path Tracing Undersampling



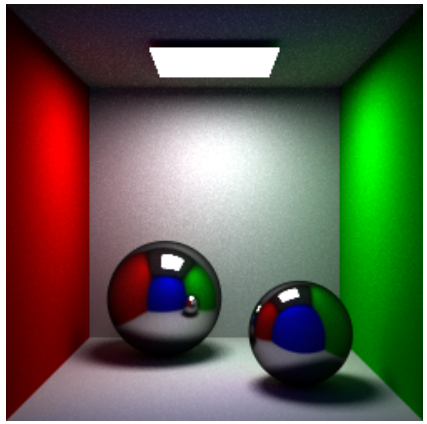
1 ray



4 rays



64 rays

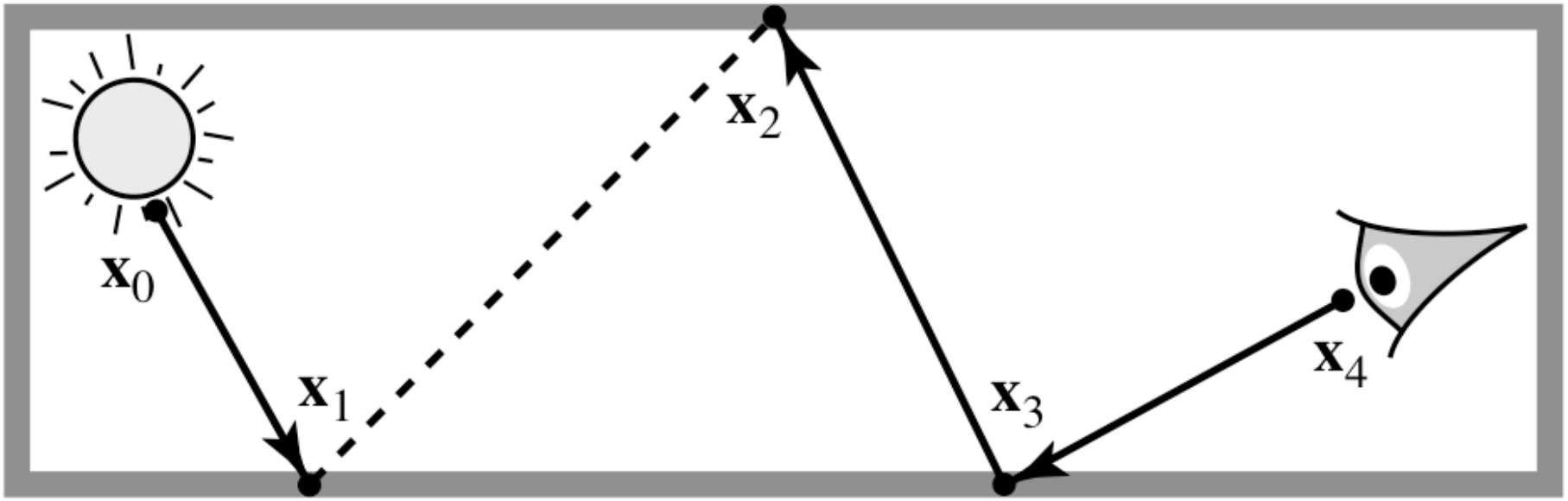


1024 rays

Path Tracing

- Relatively straightforward to implement given the basic code of a ray tracer
- Produces global illumination (i.e., any light path can be simulated)
- Can take a very long time
- Undersampling will result in speckled (noisy) images

Bi-directional Path Tracing



Conclusion

- Monte-Carlo Rendering
  - Sample all possible paths (randomly)
- Reduce variance using importance sampling