# Monte Carlo Path Tracing

COS 526: Advanced Computer Graphics

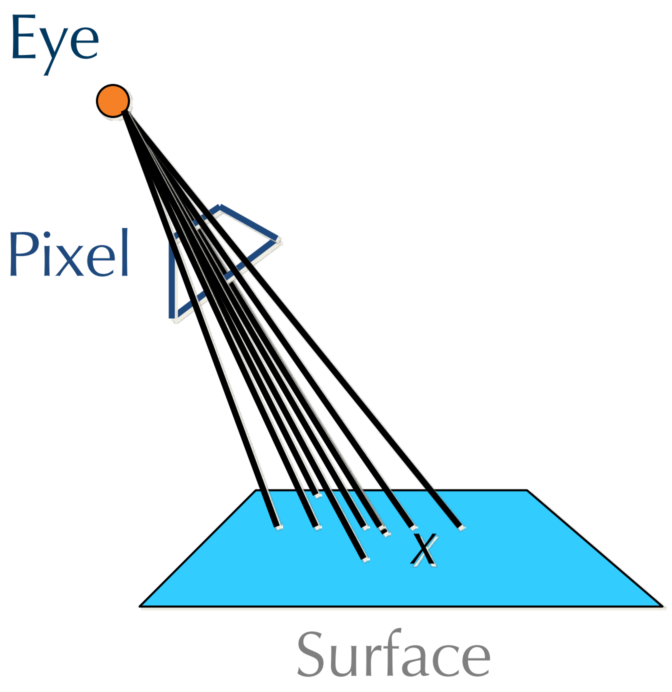
PRINCETON UNIVERSITY

## Monte Carlo Global Illumination

- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics

# Monte Carlo Global Illumination

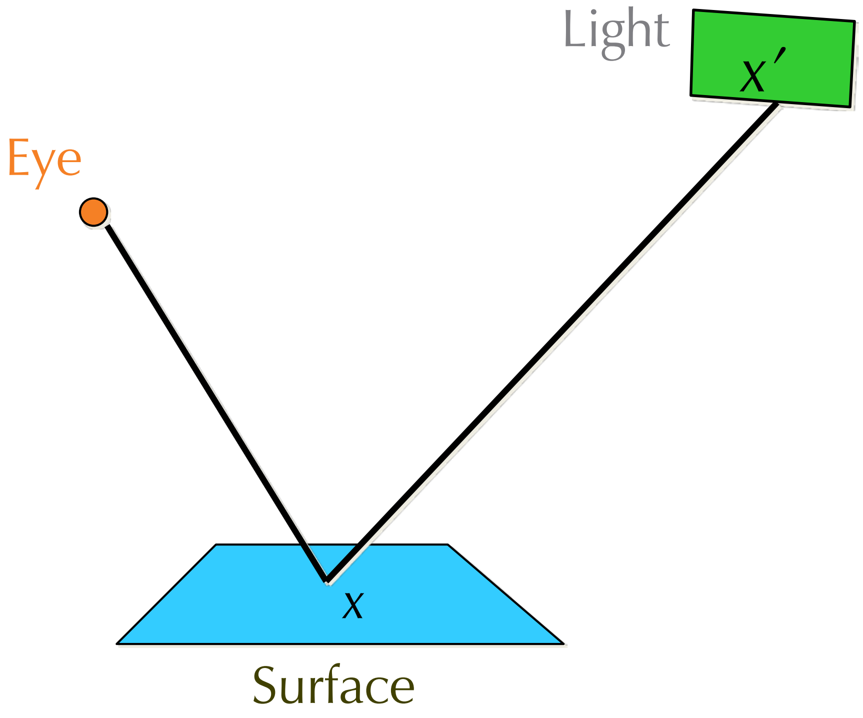
- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics



$$L_P = \int_S L(x \to e)\,dA$$

# Monte Carlo Global Illumination

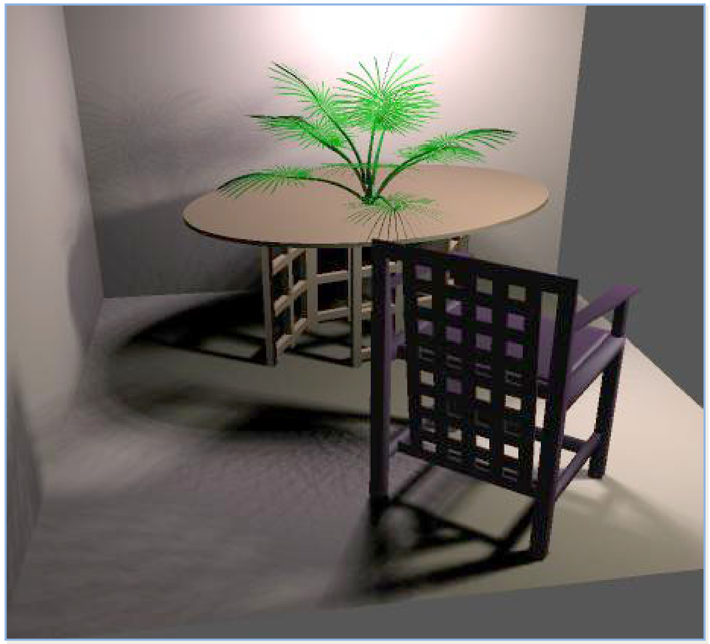
- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics



$$L(x,\vec{w}) = L_e(x,x \rightarrow e) + \int_S f_r(x,x' \rightarrow x, x \rightarrow e) L(x' \rightarrow x) V(x,x') G(x,x') dA$$

# Monte Carlo Global Illumination

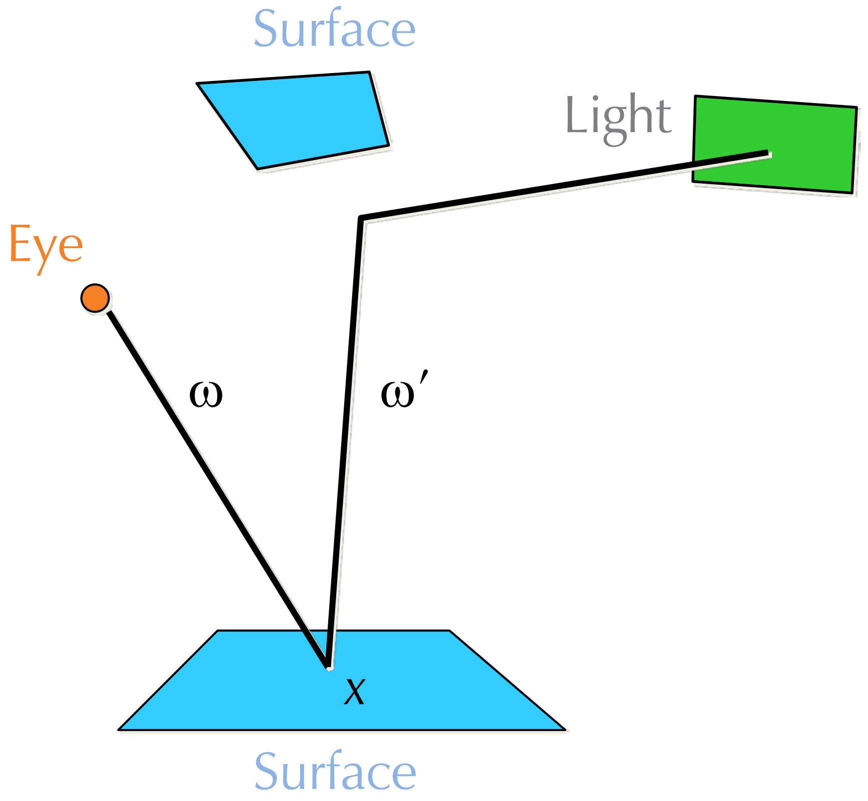
- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics


Herf

$$L(x,\vec{w}) = L_e(x,x \rightarrow e) + \int_S f_r(x,x' \rightarrow x, x \rightarrow e) L(x' \rightarrow x) V(x,x') G(x,x') dA$$

# Monte Carlo Global Illumination

- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics



$$L_o(x,\vec{w}) = L_e(x,\vec{w}) + \int_{\Omega} f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}' \bullet \vec{n}) d\vec{w}$$

# Monte Carlo Global Illumination

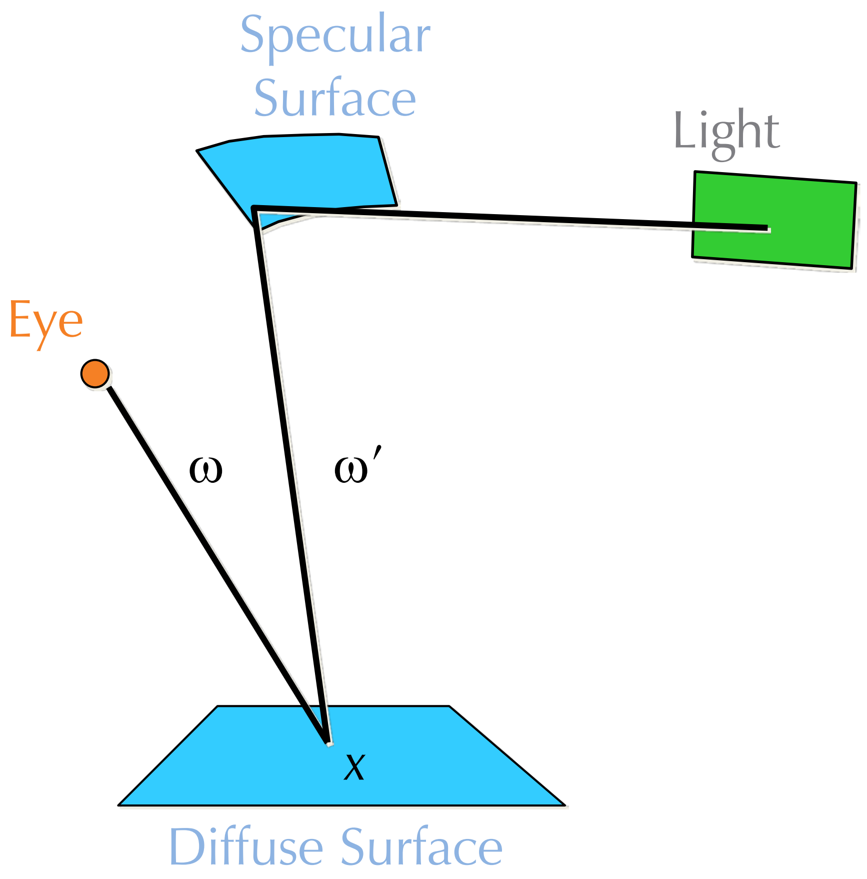
- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics

Debevec

$$L_o(x,\vec{w}) = L_e(x,\vec{w}) + \int_{\Omega} f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}' \bullet \vec{n}) d\vec{w}$$

# Monte Carlo Global Illumination

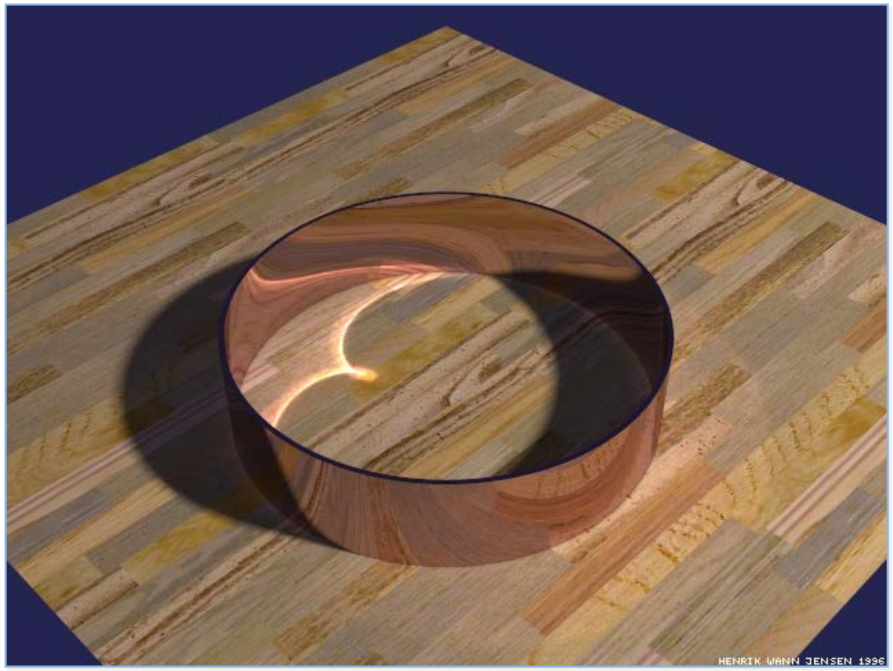
- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics



$$L_o(x,\vec{w}) = L_e(x,\vec{w}) + \int_{\Omega} f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}' \bullet \vec{n}) d\vec{w}$$

# Monte Carlo Global Illumination

- Rendering = integration
  - Antialiasing
  - Soft shadows
  - Indirect illumination
  - Caustics

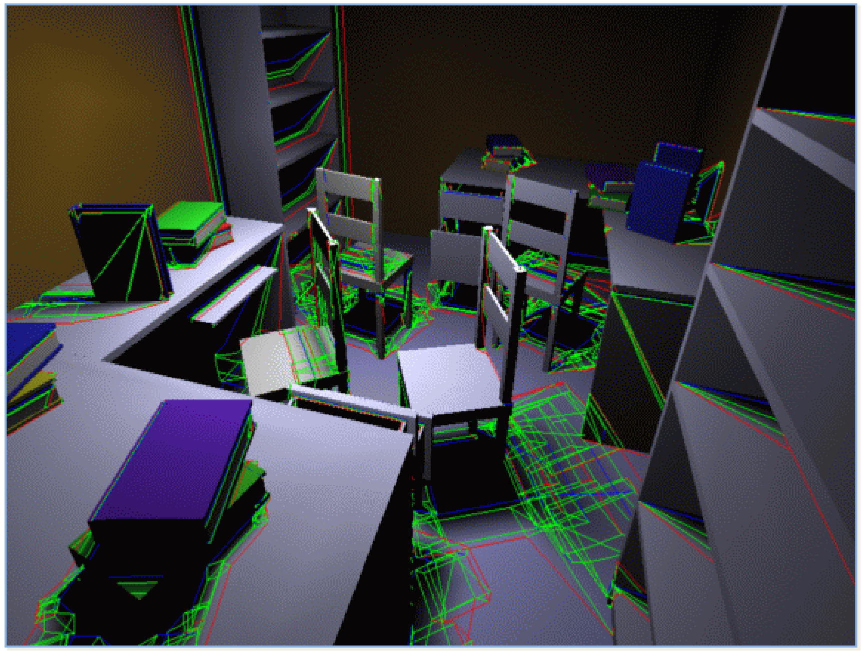


Jensen

$$L_o(x,\vec{w}) = L_e(x,\vec{w}) + \int_{\Omega} f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}' \bullet \vec{n}) d\vec{w}$$

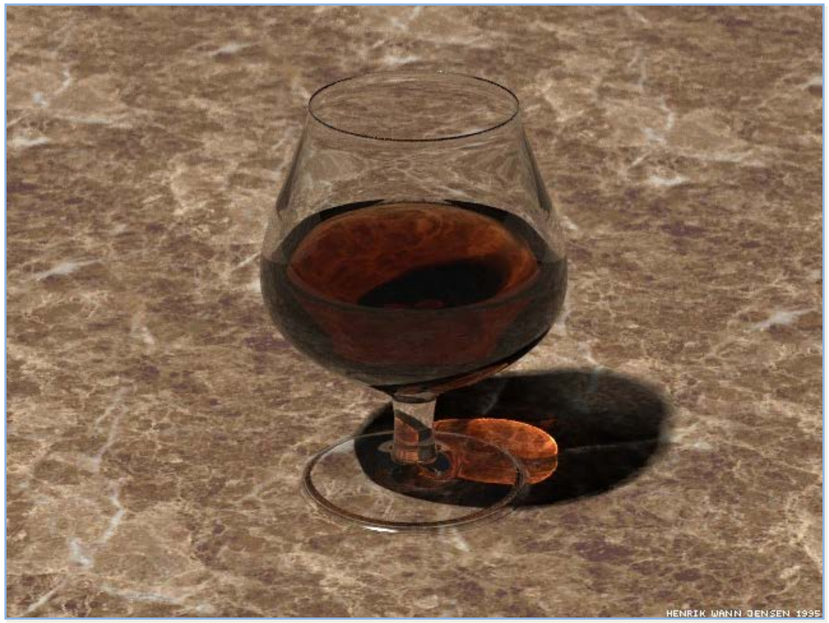
## Challenge

- Rendering integrals are difficult to evaluate
  - Multiple dimensions
  - Discontinuities
    - Partial occluders
    - Highlights
    - Caustics
  - Significant energy carried by "rare" paths


Drettakis

## Challenge

- Rendering integrals are difficult to evaluate
  - Multiple dimensions
  - Discontinuities
    - Partial occluders
    - Highlights
    - Caustics
  - Significant energy carried by "rare" paths



Jensen

## Challenge

- Rendering integrals are difficult to evaluate
  - Multiple dimensions
  - Discontinuities
    - Partial occluders
    - Highlights
    - Caustics
  - Significant energy carried by "rare" paths

Heinrich

## Challenge

- Rendering integrals are difficult to evaluate
  - Multiple dimensions
  - Discontinuities
    - Partial occluders
    - Highlights
    - Caustics
  - Significant energy carried by "rare" paths

RENDERED USING DALI - HENRIK WANN JENSEN 2000

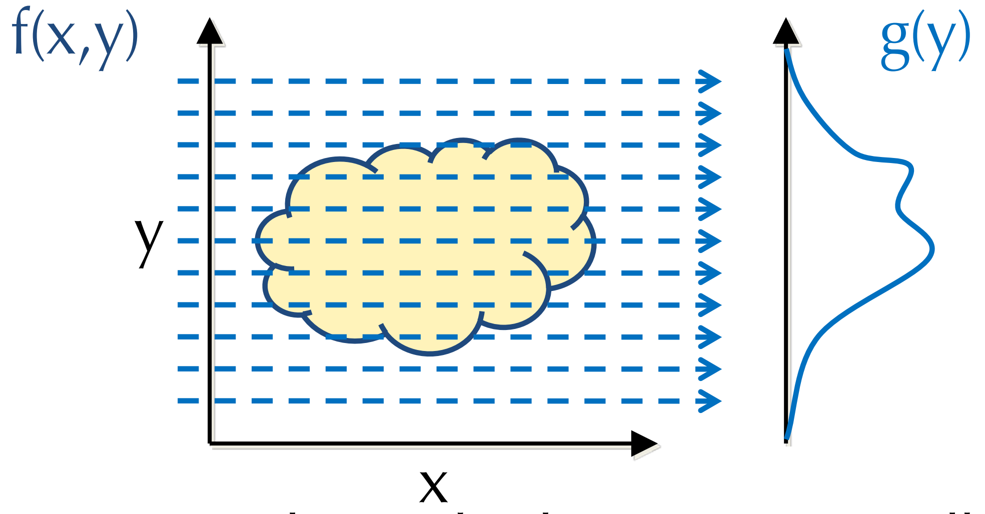
Jensen

## Outline

- Motivation
- Monte Carlo integration
- Variance reduction techniques
- Monte Carlo path tracing
- Sampling techniques
- Conclusion

## Integration in $d$ Dimensions?

- One option: nested 1-D integration



$$\iint f(x,y)\,dx\,dy = \int g(y)\,dy$$

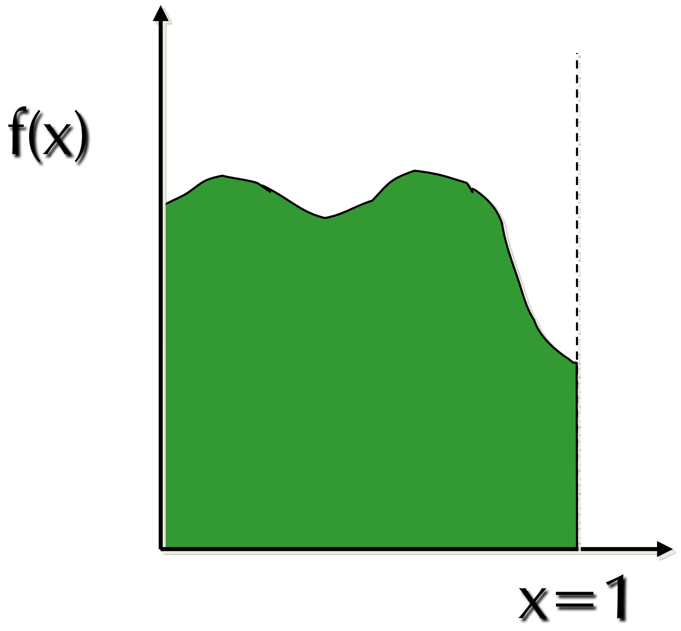
Evaluate the latter numerically, but each "sample" of g(y) is itself a 1-D integral, done numerically

## Integration in $d$ Dimensions?

- Midpoint / trapezoid / Simpson's rule in *d* dimensions?
  - In 1D: $(b-a)/h$ points
  - In 2D: $(b-a)/h^2$ points
  - In general: $O(1/h^d)$ points
- Required # of points ***grows exponentially with dimension***, for a fixed order of method
  - "Curse of dimensionality"
- Other problems, e.g. non-rectangular domains

# Rethinking Integration in 1D

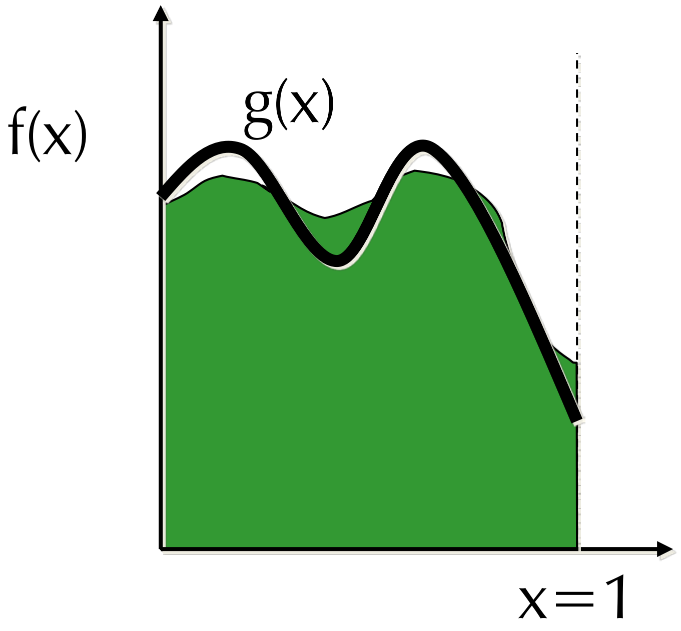
$$\int_0^1 f(x)dx = ?$$



Slide courtesy of
Peter Shirley

# We Can Approximate…

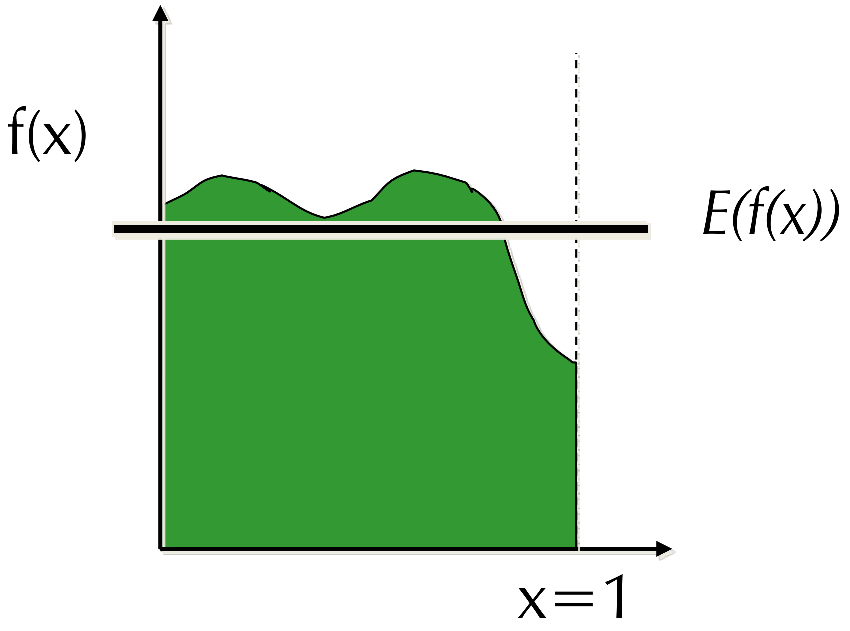
$$\int_0^1 f(x)dx = \int_0^1 g(x)dx$$



Slide courtesy of
Peter Shirley

# Or We Can Average

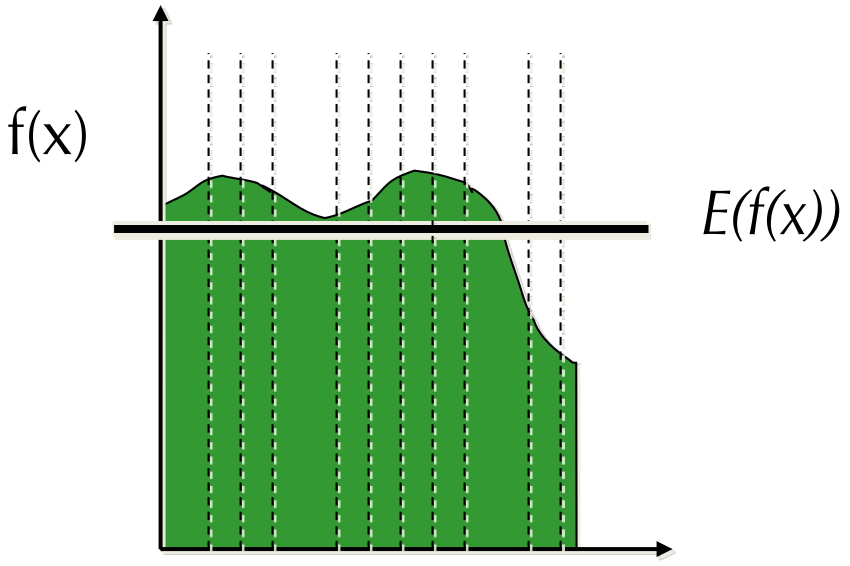
$$\int_0^1 f(x)dx = E(f(x))$$



Slide courtesy of
Peter Shirley

# Estimating the Average

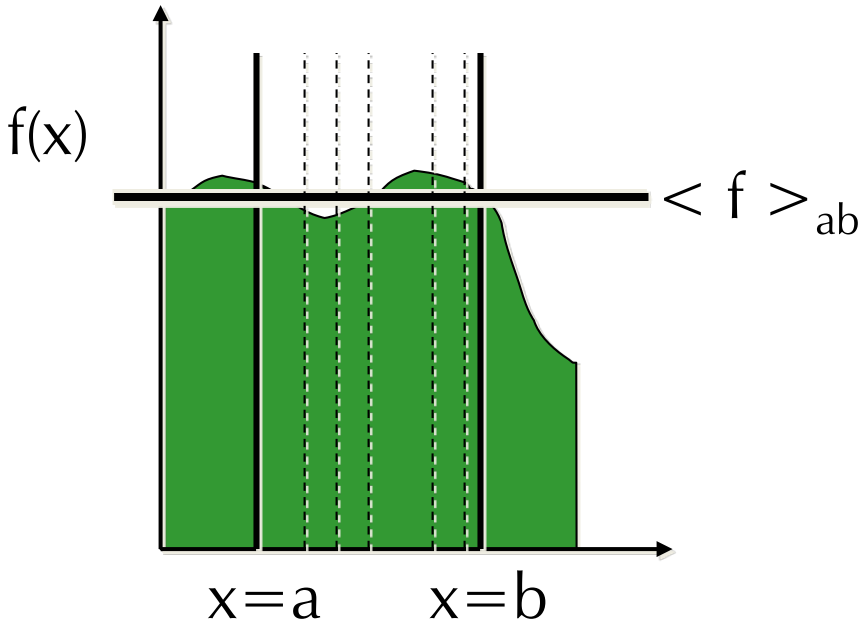
$$\int_0^1 f(x)dx \cong \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$



Slide courtesy of
Peter Shirley

## Other Domains

$$\int_a^b f(x)dx \cong \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$



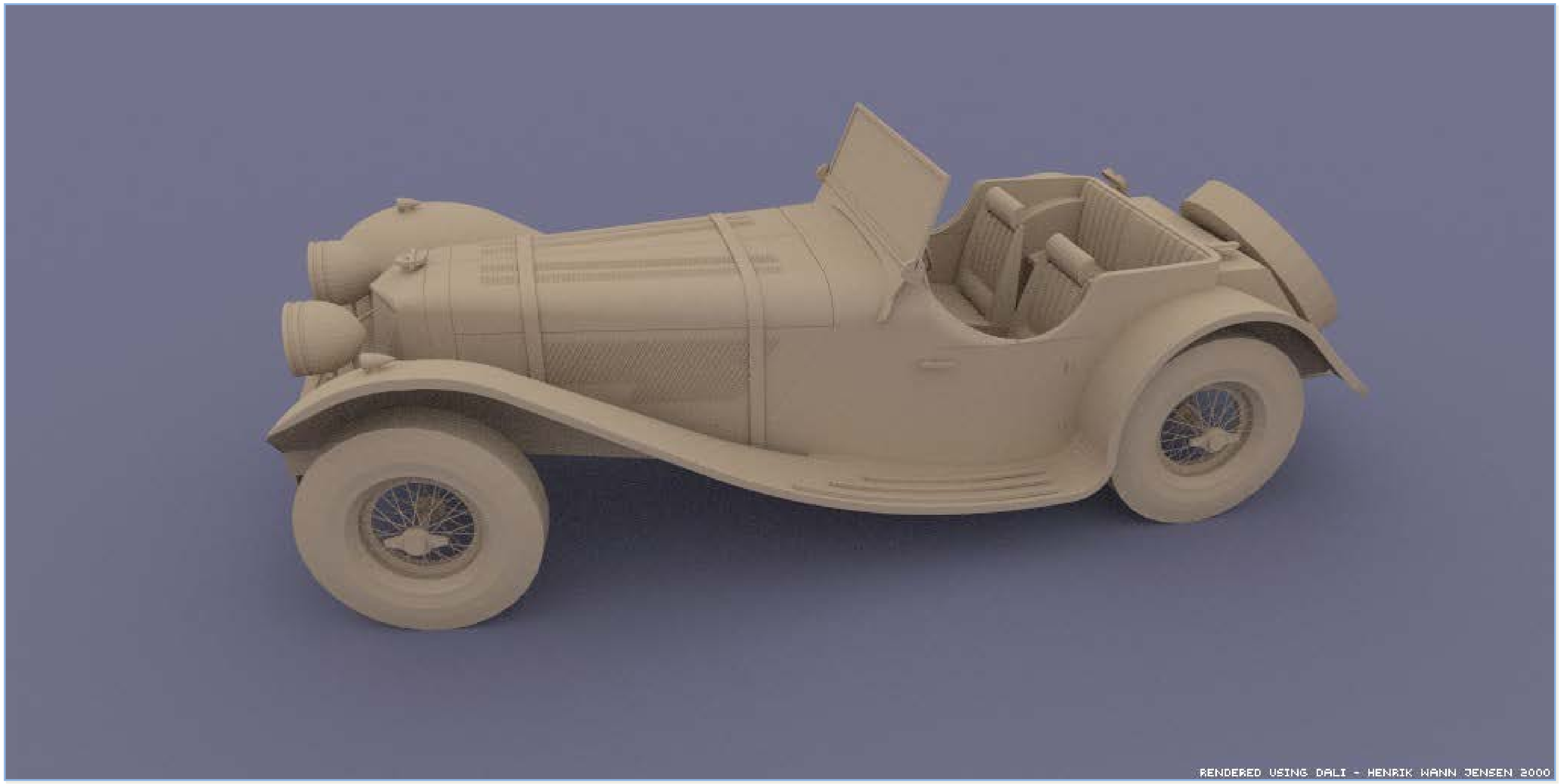
Slide courtesy of Peter Shirley

## “Monte Carlo” Integration

- No “exponential explosion” in required number of samples with increase in dimension
- (Some) resistance to badly-behaved functions

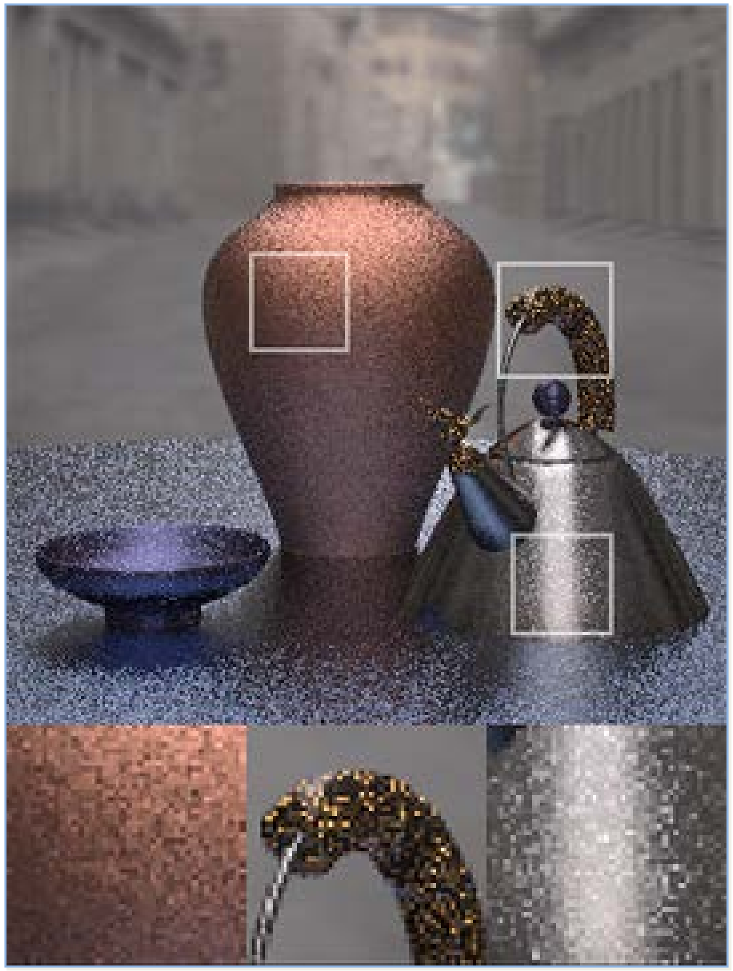
Le Grand Casino de Monte-Carlo

## Monte Carlo Path Tracing



# Monte Carlo Path Tracing

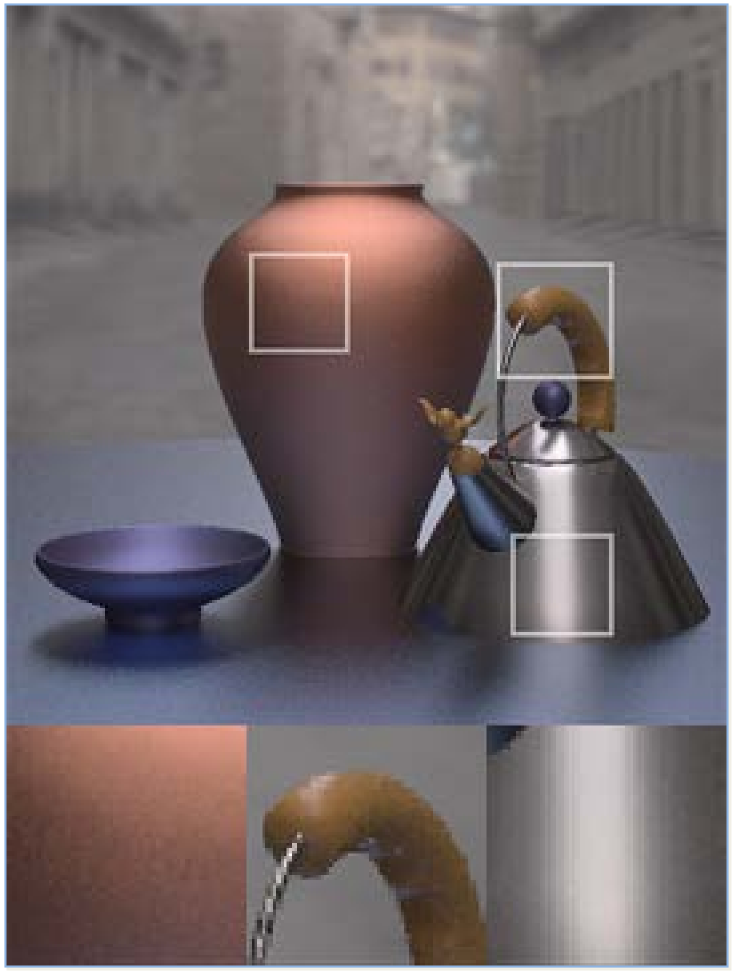
- Drawback: can be noisy unless *lots* of paths simulated
- 40 paths per pixel:



Lawrence

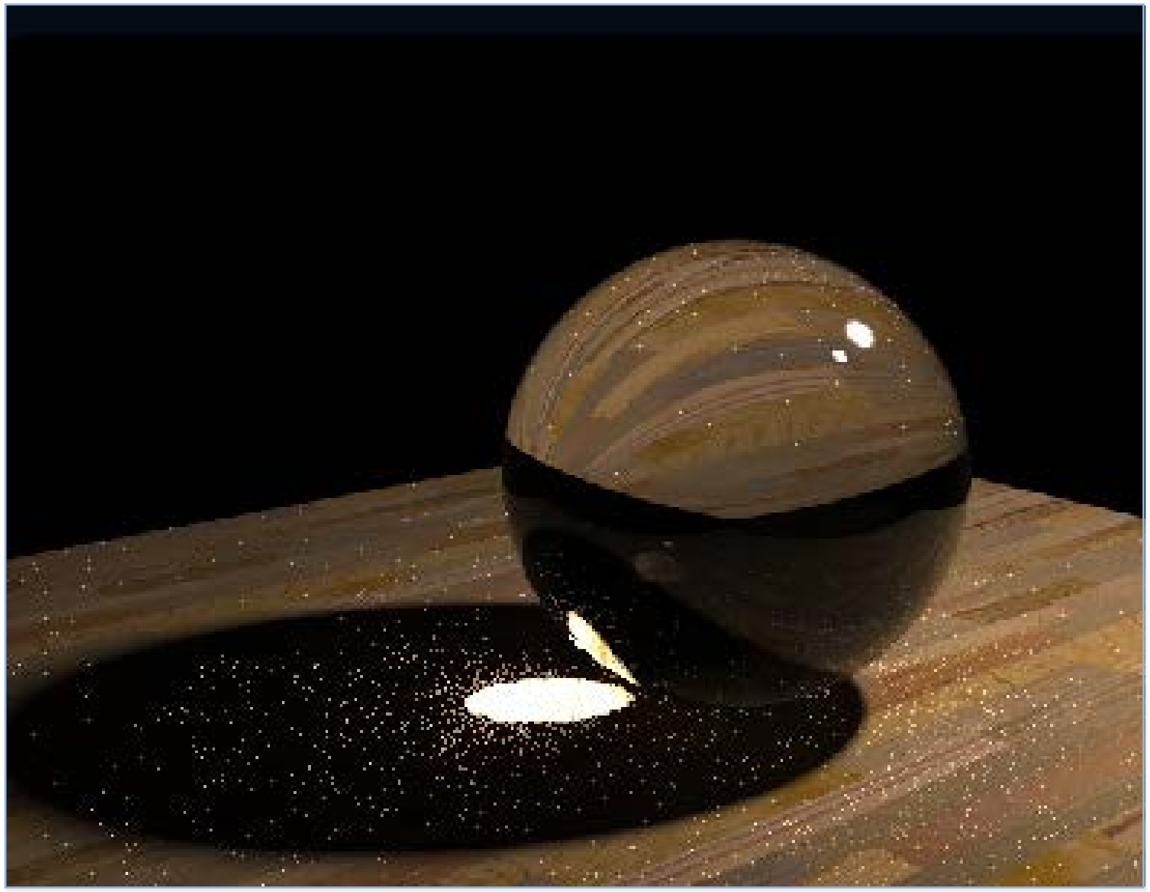
## Monte Carlo Path Tracing

- Drawback: can be noisy unless *lots* of paths simulated
- 1200 paths per pixel:



Lawrence

# Monte Carlo Path Tracing

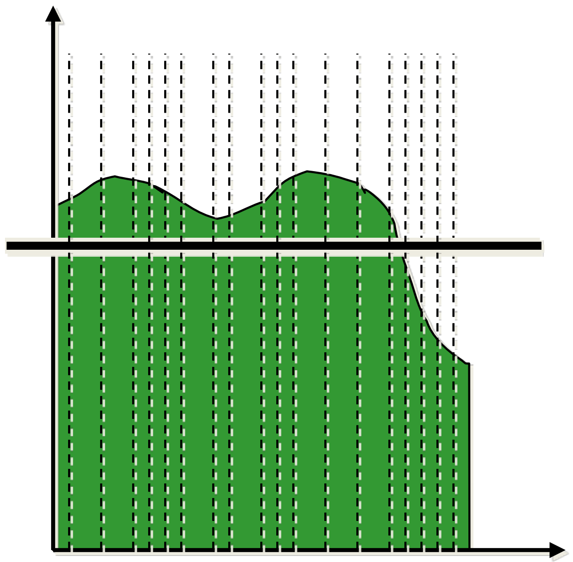

1000 paths/pixel

Jensen

## Outline

- Motivation
- Monte Carlo integration
- Variance reduction techniques
- Monte Carlo path tracing
- Sampling techniques
- Conclusion

# Variance



$$\int_a^b f(x)dx \cong \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$

$$Var\left[\frac{b-a}{N}\sum_{i=1}^{N} f(x_i)\right] = \left(\frac{b-a}{N}\right)^2 \sum_{i=1}^{N} Var[f(x_i)]$$

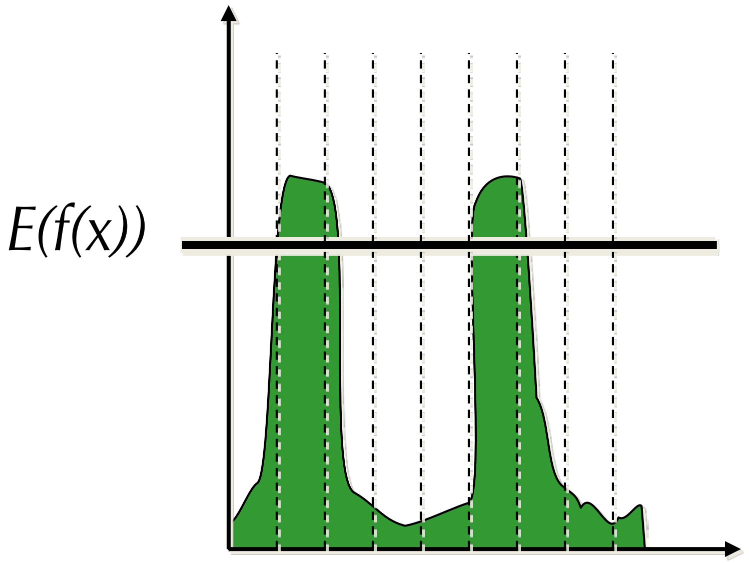
$$= \frac{(b-a)^2}{N} Var[f(x_i)]$$

* with a correction of $\sqrt{\frac{N}{N\text{-}1}}$
(consult a statistician for details)

Variance decreases as 1/N
Error of E decreases as 1/sqrt(N)

## Variance

- Problem: variance decreases with 1/N
  - Increasing # samples removes noise slowly



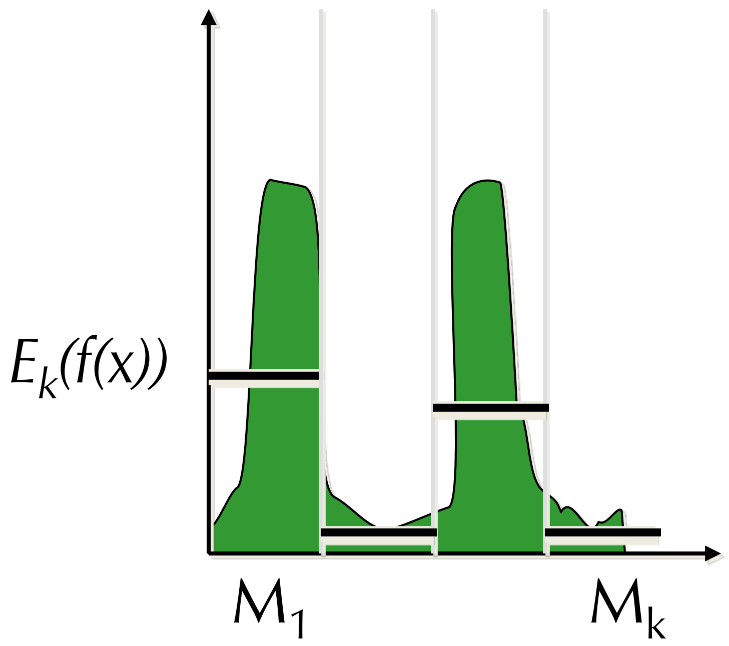
## Variance Reduction Techniques

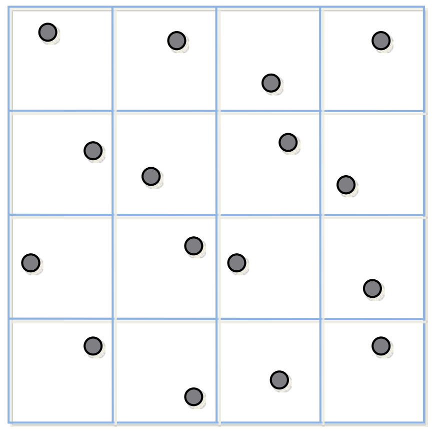
- Problem: variance decreases with 1/N
  - Increasing # samples removes noise slowly

- Variance reduction:
  - Stratified sampling
  - Importance sampling

## Stratified Sampling

- Estimate subdomains separately

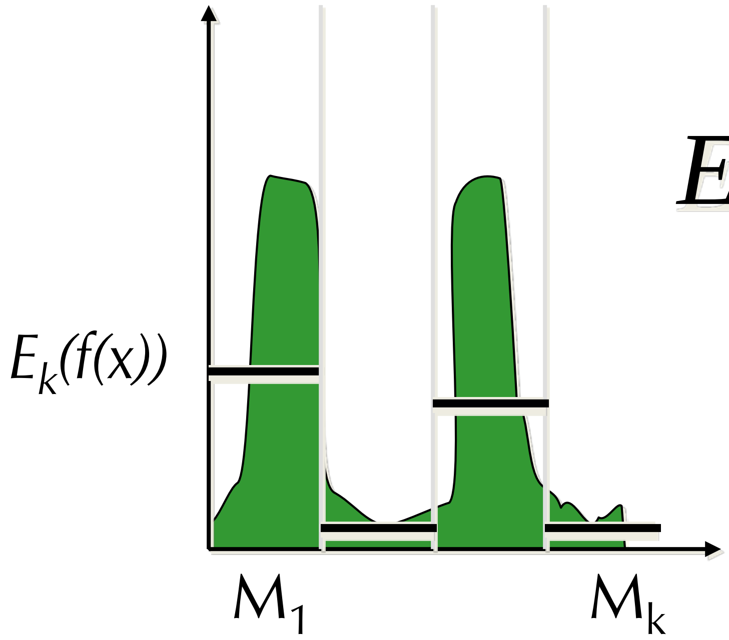




Can do this recursively!

# Stratified Sampling

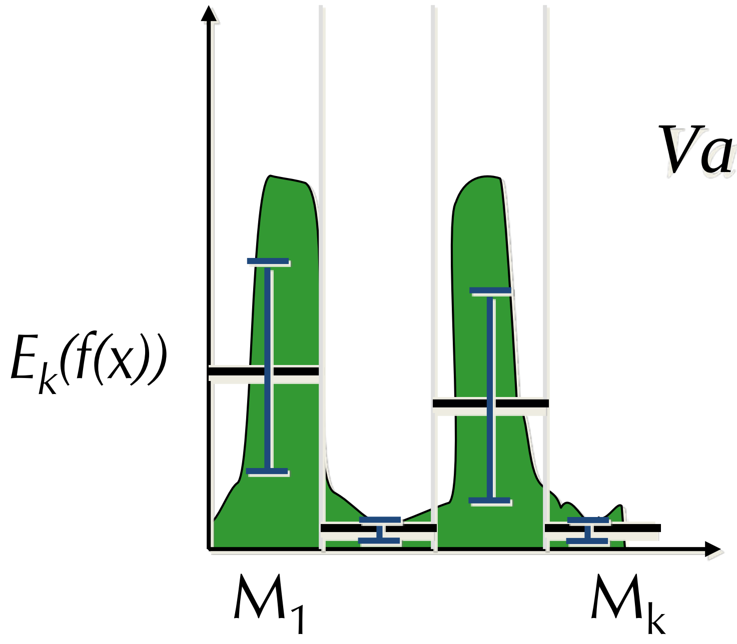
- This is still unbiased



$$E = \sum_{j=1}^{k} \frac{vol(M_j)}{N_j} \sum_{n=1}^{N_j} f(x_{jn})$$

# Stratified Sampling

- Less overall variance if less variance in subdomains

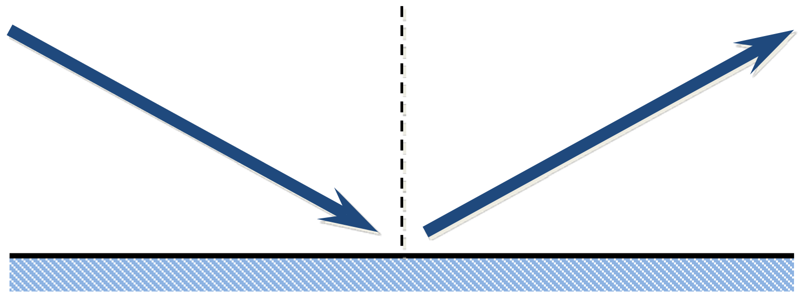


$$Var[E] = \sum_{j=1}^{k} \frac{vol(M_j)^2}{N_j} Var[f(x)]\Big|_{M_j}$$

Total variance minimized when number of points in each subvolume $M_j$ proportional to error in $M_j$.

# Reducing Variance

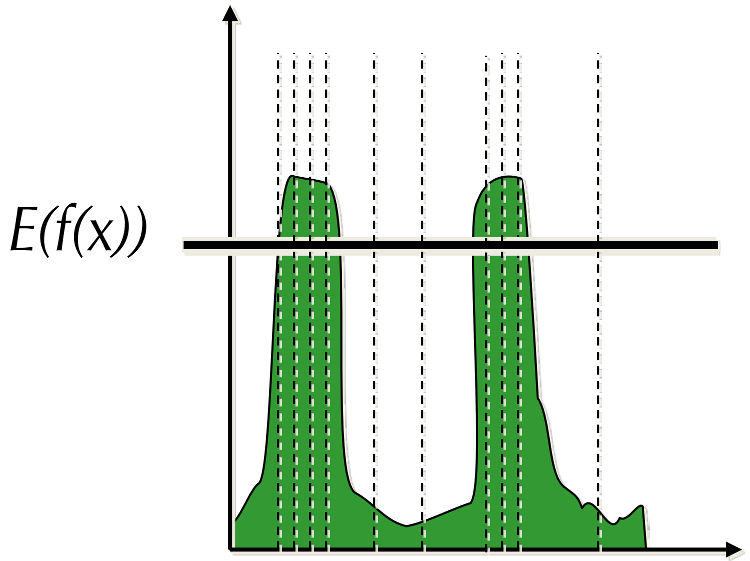
- Observation: some paths more important (carry more energy) than others
  - For example, shiny surfaces reflect more light in the ideal "mirror" direction



- Idea: put more samples where f(x) is bigger

# Importance Sampling

- Put more samples where f(x) is bigger



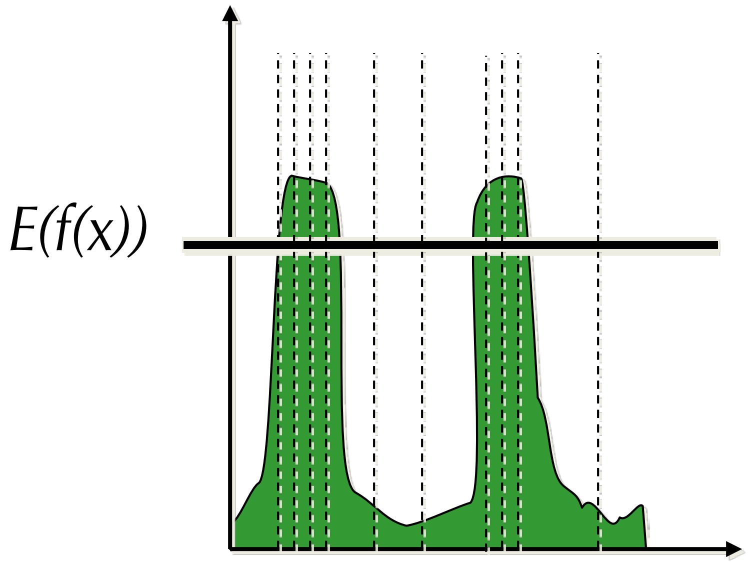
$$\int_{\Omega} f(x)dx = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

$$\text{where } Y_i = \frac{f(x_i)}{p(x_i)}$$

$$\text{and } x_i \text{ drawn from } P(x)$$

# Importance Sampling

- This is still unbiased



$$E[Y_i] = \int_\Omega Y(x)\,p(x)\,dx$$

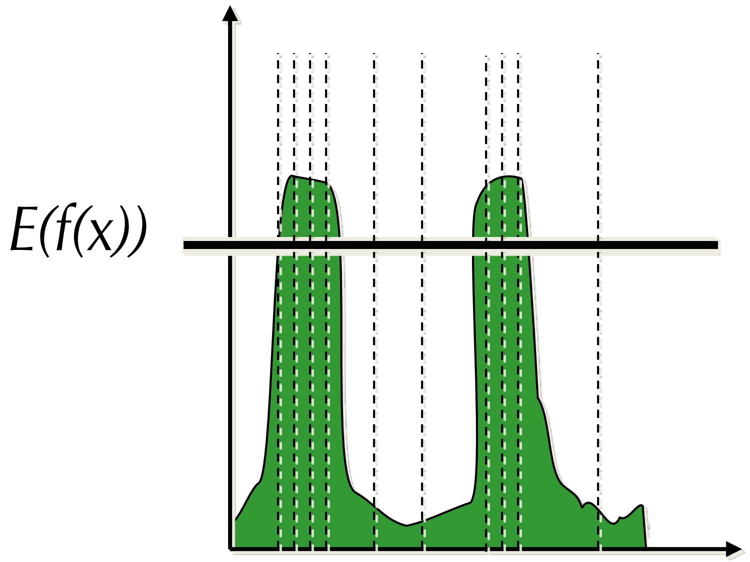
$$= \int_\Omega \frac{f(x)}{p(x)}\,p(x)\,dx$$

$$= \int_\Omega f(x)\,dx$$

for all N

## Importance Sampling

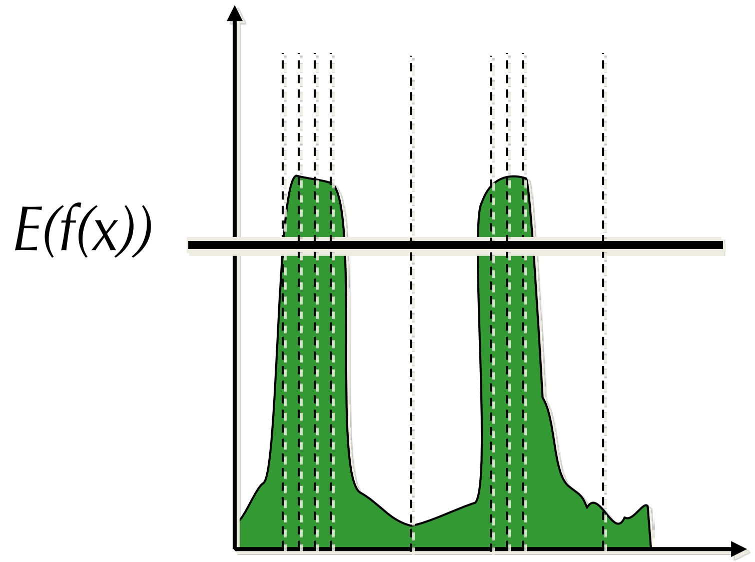
- Variance depends on choice of p(x):



$$Var(E) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{f(x_n)}{p(x_n)}\right)^2 - E^2$$

# Importance Sampling

- Zero variance if p(x) ~ f(x)



$$p(x) = cf(x)$$

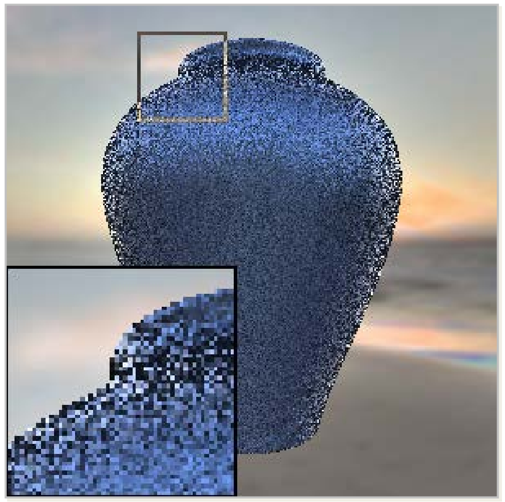
$$Y_i = \frac{f(x_i)}{p(x_i)} = \frac{1}{c}$$

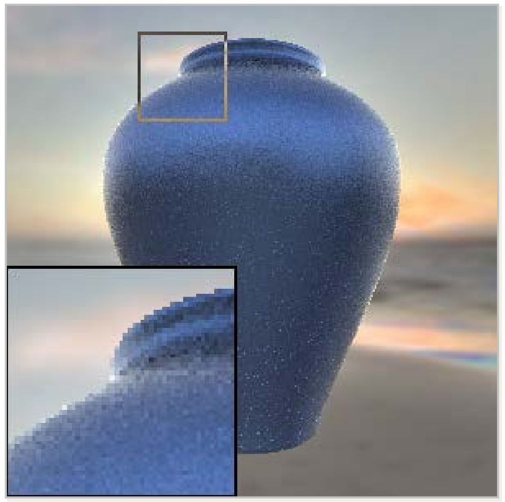
$$Var(Y) = 0$$

Less variance with better importance sampling

## Effect of Importance Sampling

- Less noise at a given number of samples


Uniform random sampling


Importance sampling

- Equivalently, need to simulate fewer paths for some desired limit of noise

## Random number generation

## True random numbers

- http://www.random.org/

10101111 00101011 10111000 11110110 10101010 00110001 01100011 00010001

00000011 00000010 00111111 00010011 00000101 01001100 10000110 11100010

10010100 10000101 10000011 00000100 00111011 10111000 00110000 11001010

11011101 11101111 00100010 10101011 00100110 10101111 00001011 10110100

00011100 00001111 11001001 11001100 01111101 10000100 10111000 01101011

01101011 01111101 11001010 11101110 11101110 00100010 10110100 01001000

11010111 11011011 11100100 01010010 10111101 01011010 01001110 01110000

00100010 11000111 01010000 10110011 01001011 00110001 01011100 10001111

11111000 10101011 01011011 01010000 01101111 00011001 00000011 00110000

10000001 00000110 11010011 00011110 11101101 00000011 00100110 01010011

11010111 10010001 10000111 01010010 01101010 00100101 10011111 01000111

10101001 01100001 01010011 01001000 11010110 01111110 11010011 01110110

00000001 01001110 00011001 00111001

## Pseudorandom Numbers

- Deterministic, but have statistical properties resembling true random numbers
- Common approach: each successive pseudorandom number is function of previous

## Desirable properties

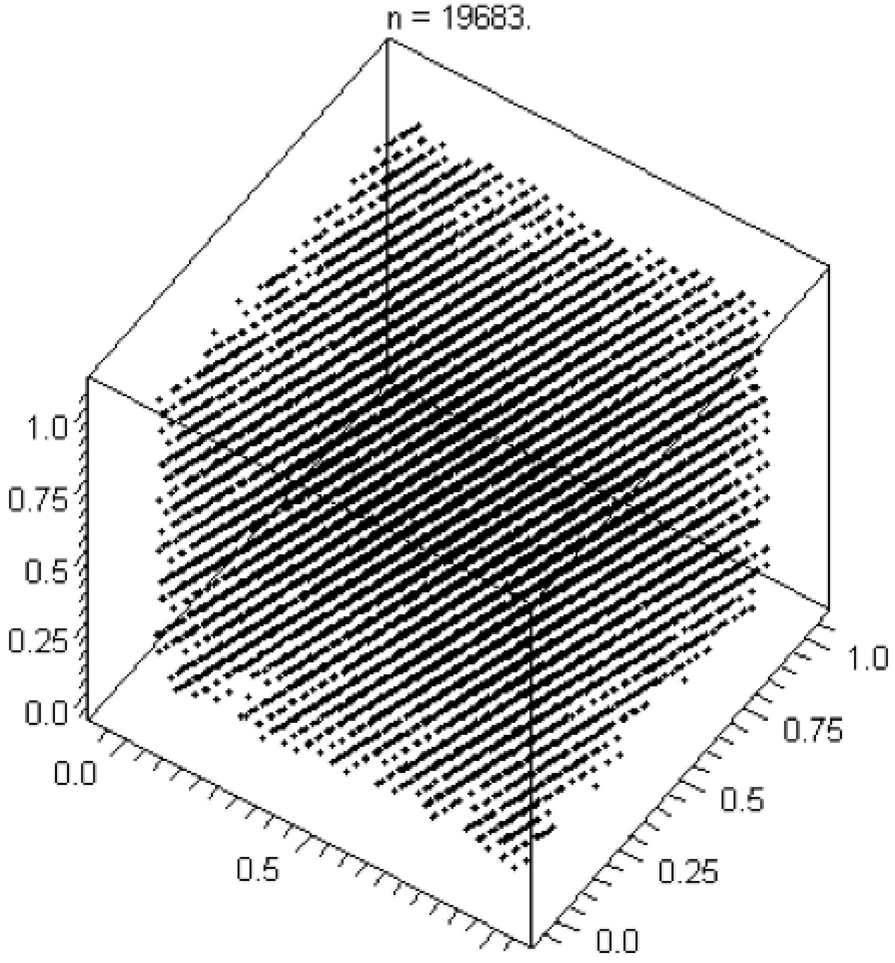
- Random pattern: Passes statistical tests (e.g., can use chi-squared)
- Long period: As long as possible without repeating
- Efficiency
- Repeatability: Produce same sequence if started with same initial conditions (for debugging!)
- Portability

## Linear Congruential Methods

$$x_{n+1} = (ax_n + b) \bmod c$$

- Choose constants carefully, e.g.
  a = 1664525
  b = 1013904223
  c = $2^{32}$
- Results in integer in [0, c)
- Simple, efficient, but often unsuitable for MC:
  e.g. exhibit serial correlations

## Problem with LCGs



## Lagged Fibonacci Generators

- Takes form $x_n = (x_{n-j} \square x_{n-k}) \bmod m$, where operation $\square$ is addition, subtraction, or XOR
- Standard choices of (j, k): e.g., (7, 10), (5,17), (6,31), (24,55), (31, 63) with $m = 2^{32}$
- Proper initialization is important and hard
- Built-in correlation!
- Not totally understood in theory (need statistical tests to evaluate)

## Seeds

- Why?

- Approaches:
  - Ask the user (for debugging)
  - Time of day
  - True random noise: from radio turned to static, or thermal noise in a resistor, or…

## Seeds

- Lava lamps!

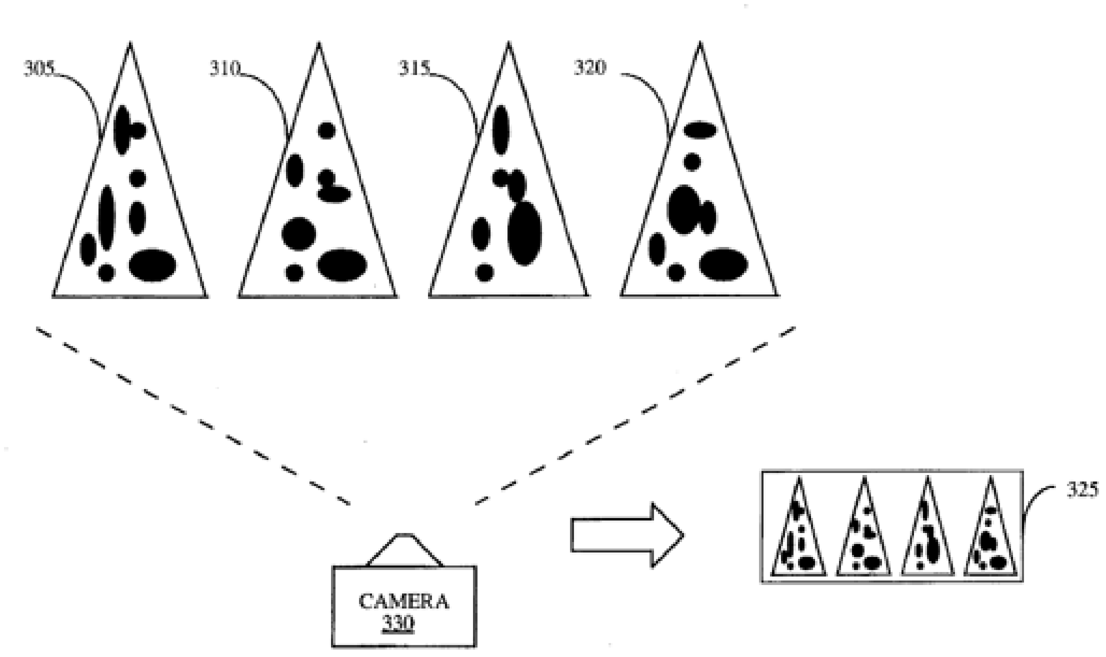


FIG. 3

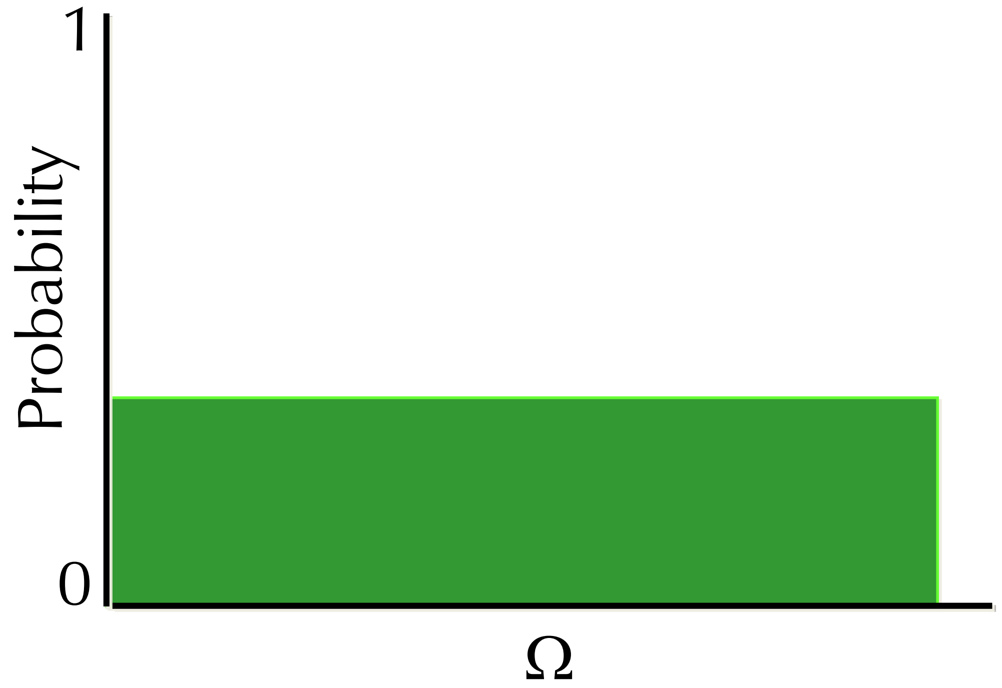
http://www.google.com/patents/about/5732138_Method_for_seeding_a_pseudo_rand.html?id=ou0gAAAAEBAJ

## Pseudorandom Numbers

- Most methods provide integers in range [0..c)
- To get floating-point numbers in [0..1), divide integer numbers by *c*
- To get integers in range [u..v], divide by c/(v–u+1), truncate, and add u
  - Better statistics than using modulo (v–u+1)
  - Only works if u and v small compared to c

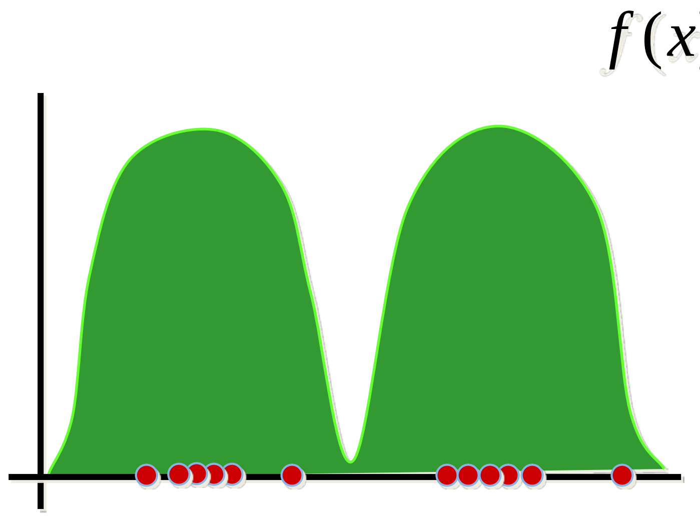
## Generating Random Points

- Uniform distribution:
  - Use pseudorandom number generator



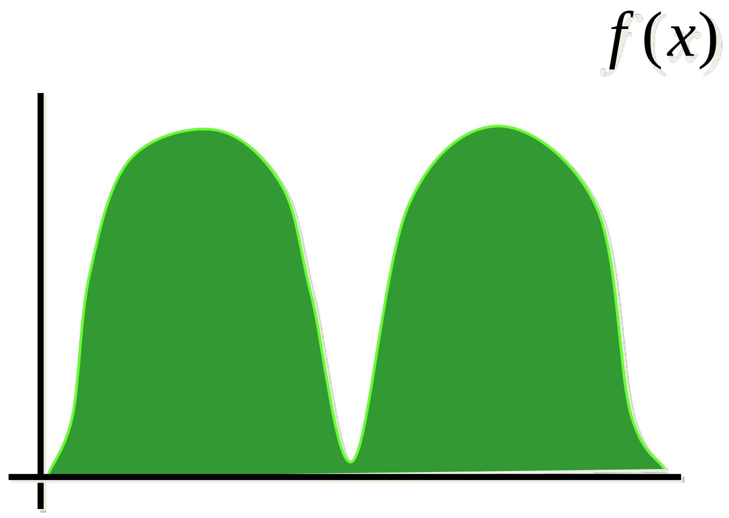
# Sampling from a non-uniform distribution

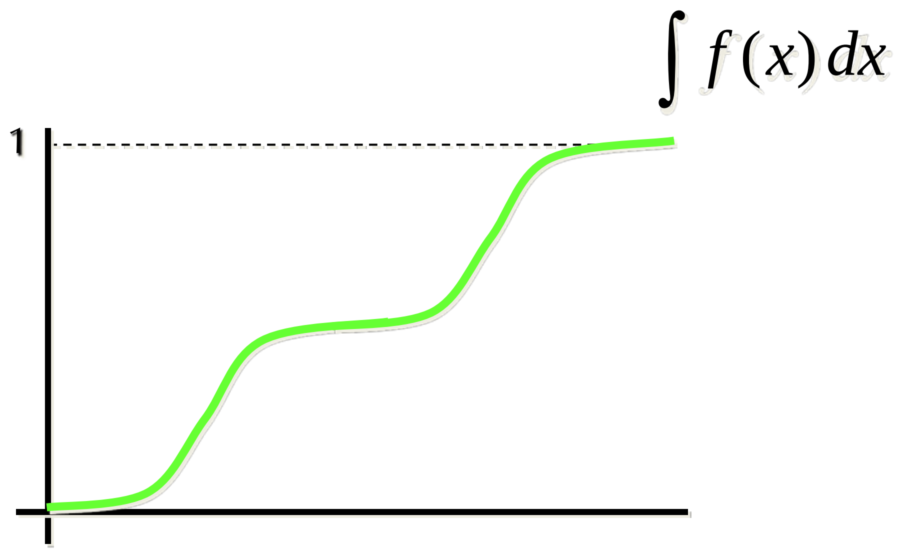
- Specific probability distribution:
  - Function inversion
  - Rejection



## Sampling from a non-uniform distribution

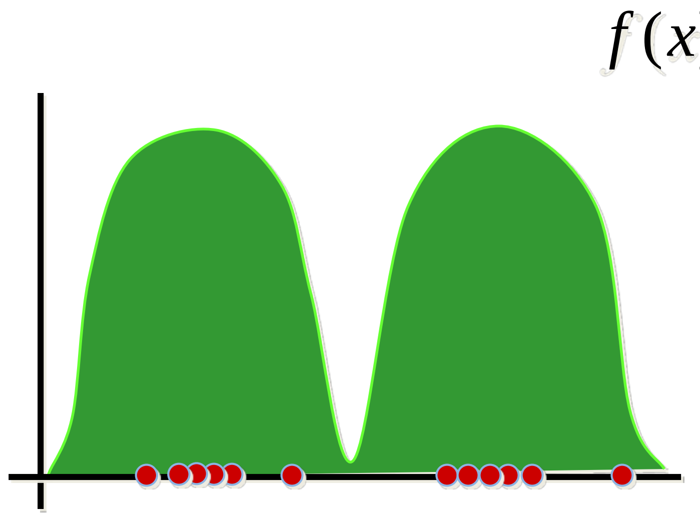
- "Inversion method"
  - Integrate *f*(*x*): Cumulative Distribution Function

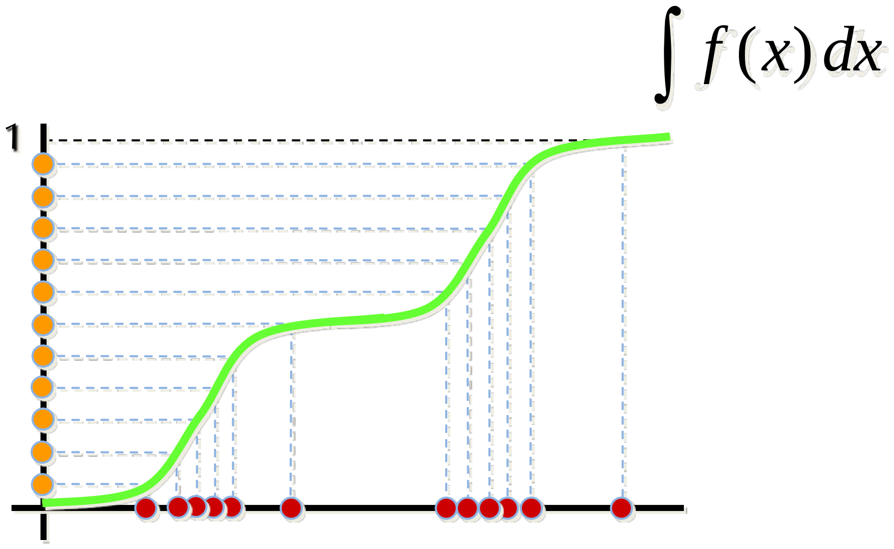




## Sampling from a non-uniform distribution

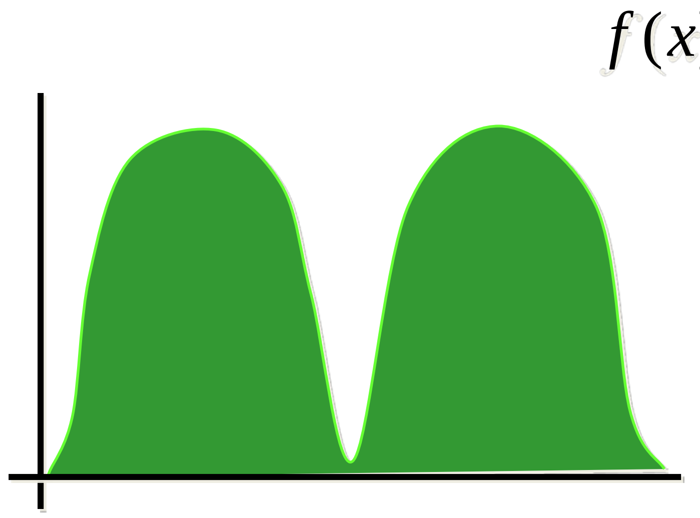
- "Inversion method"
  - Integrate *f*(*x*): Cumulative Distribution Function
  - Invert CDF, apply to uniform random variable





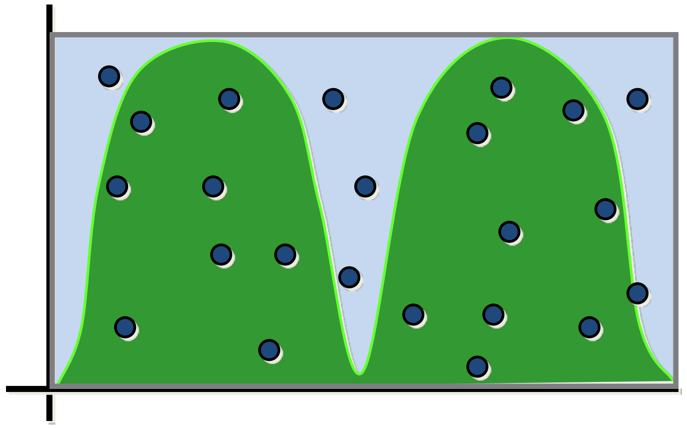
## Sampling from a non-uniform distribution

- Specific probability distribution:
  - Function inversion
  - Rejection



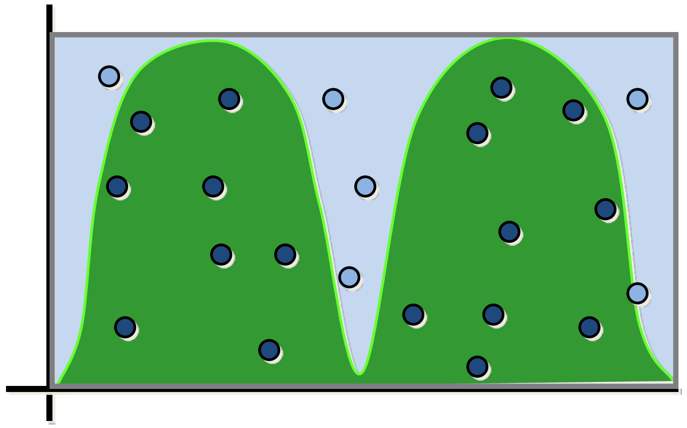
## Sampling from a non-uniform distribution

- "Rejection method"
  - Generate random (x,y) pairs, y between 0 and max(f(x))



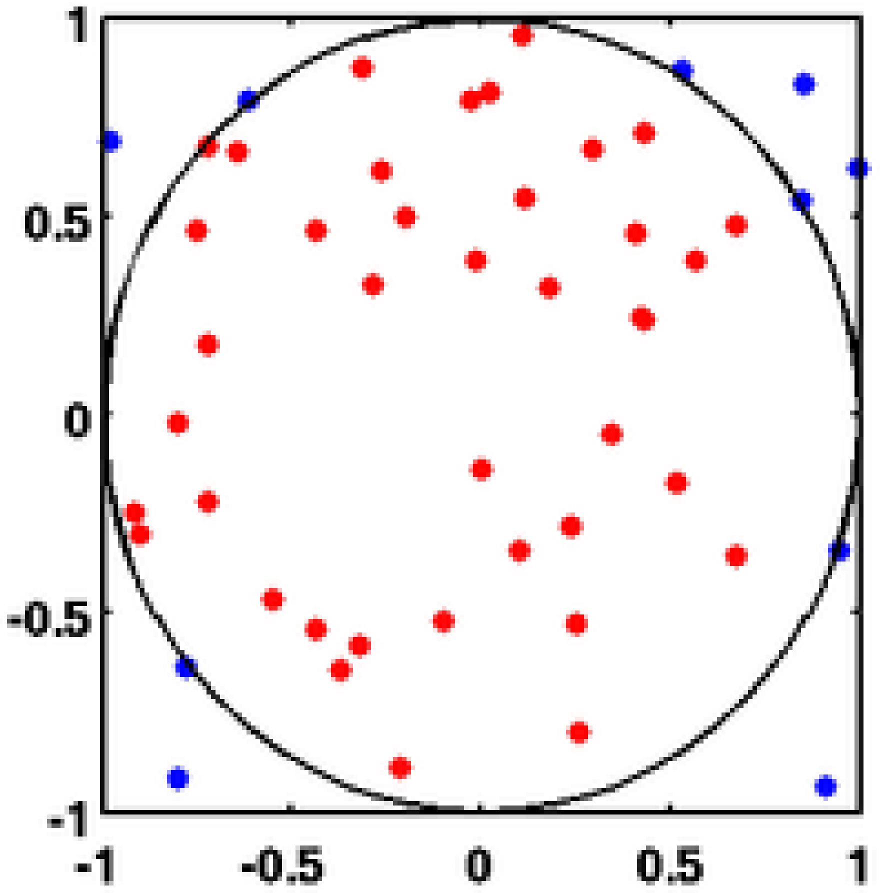
## Sampling from a non-uniform distribution

- "Rejection method"
  - Generate random (x,y) pairs, y between 0 and max(f(x))
  - Keep only samples where y < f(x)

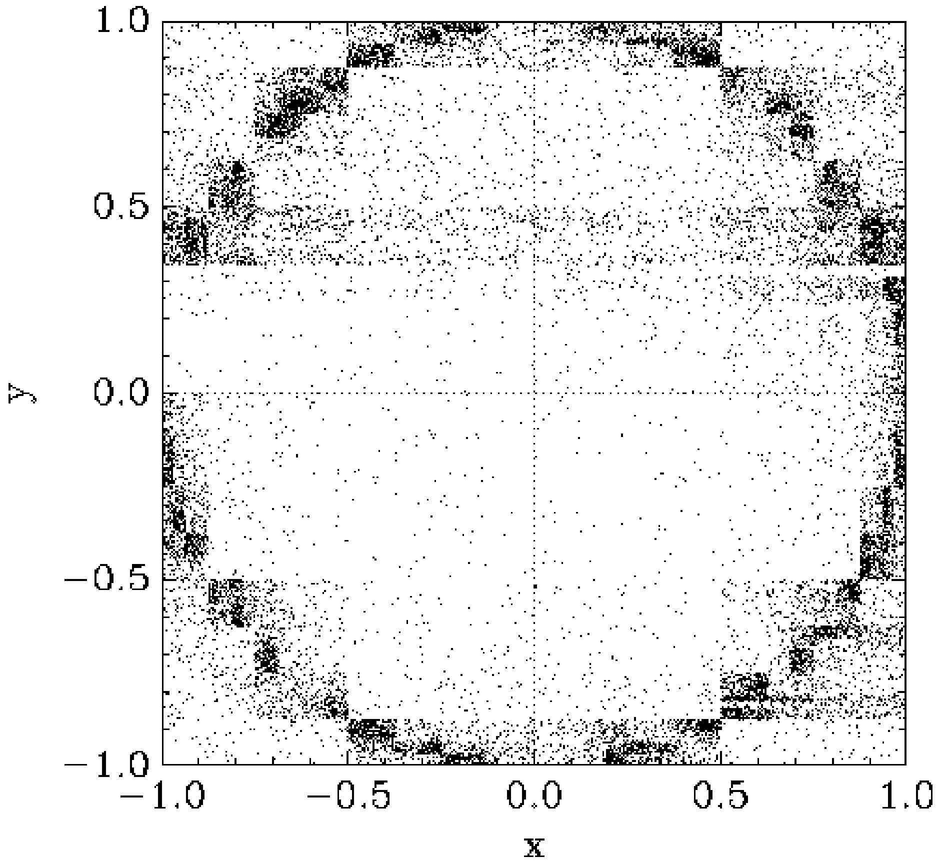


Doesn't require cdf: Can use directly for importance sampling.

## Example: Computing pi



## With Stratified Sampling

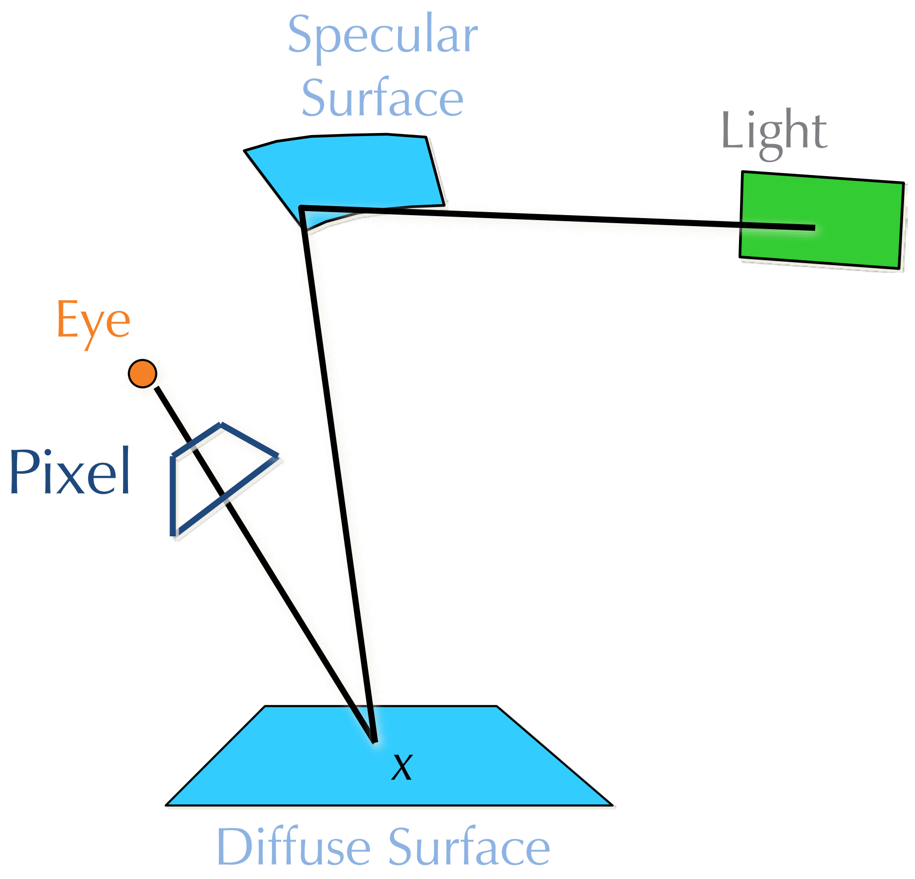


## Outline

- Motivation
- Monte Carlo integration
- Variance reduction techniques
- Monte Carlo path tracing
- Sampling techniques
- Conclusion

# Monte Carlo Path Tracing

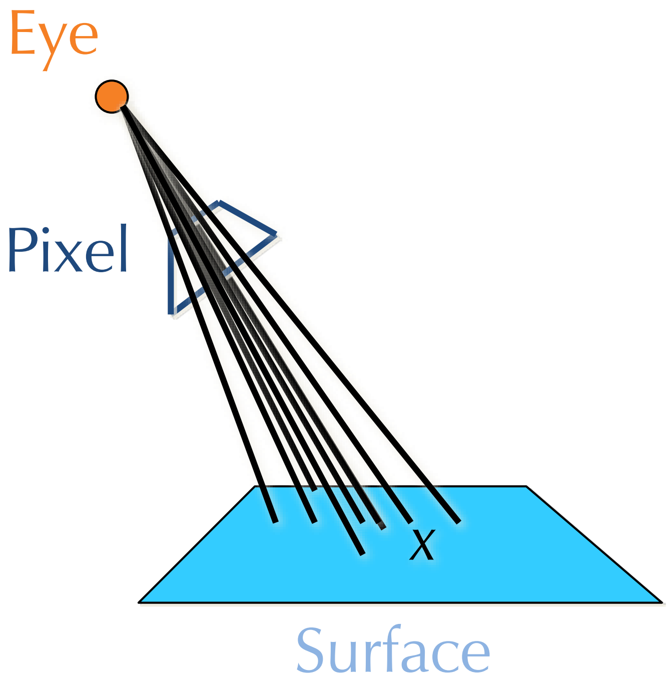
- Integrate radiance for each pixel by sampling paths randomly



# Monte Carlo Path Tracer

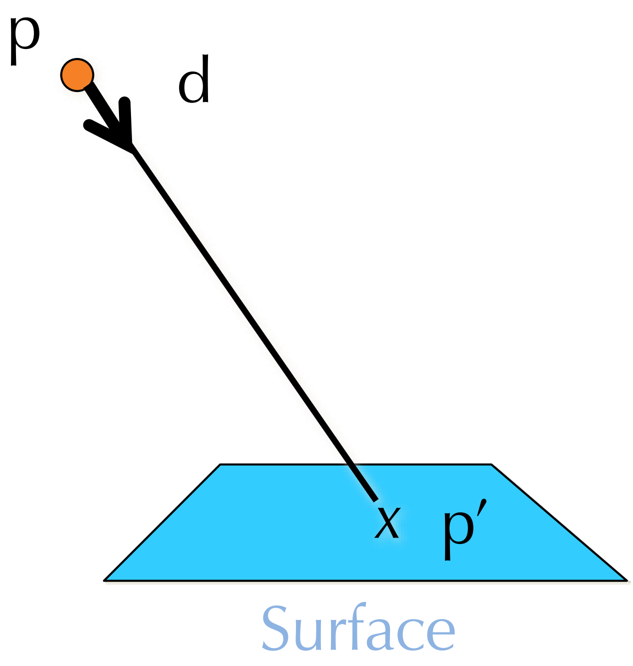
- For each pixel, repeat *n* times:
  - Choose a ray with *p*=camera, *d*=(θ,ϕ) within pixel
  - Pixel color += (1/n) * TracePath(*p*, *d*)

- Use stratified sampling to select rays within each pixel



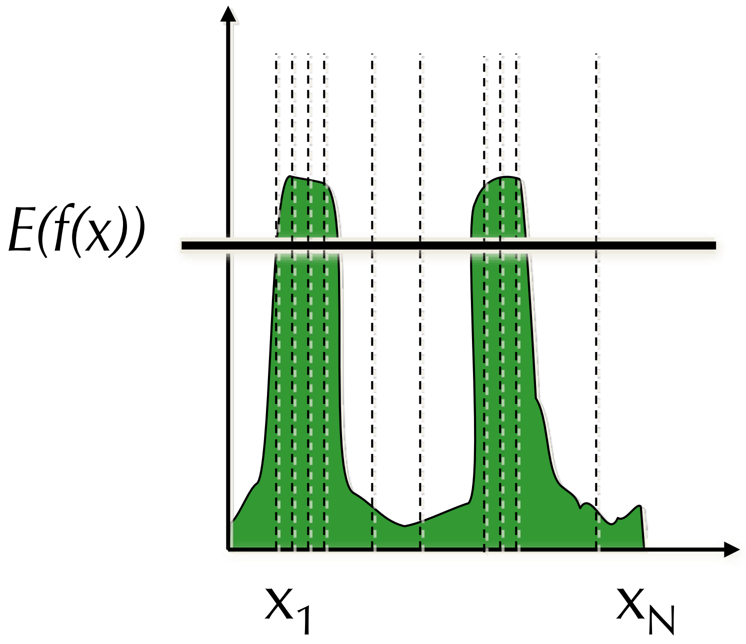
## TracePath

- TracePath(*p*, *d*) returns (r,g,b):
  - Trace ray (*p*, *d*) to find nearest intersection *p′*
  - Sample radiance leaving p′ towards p



# TracePath

- Can sample radiance however we want, but contribution weighted by 1/probability



$$\int_{\Omega} f(x)dx = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

$$\text{where } Y_i = \frac{f(x_i)}{p(x_i)}$$

# TracePath

- TracePath($p$, $d$) returns (r,g,b):
  - Trace ray ($p$, $d$) to find nearest intersection $p'$
  - If random() < $p_{emit}$ then
    - Emitted:

      return (1/ $p_{emit}$) * ($Le_{red}$, $Le_{green}$, $Le_{blue}$)
    - Reflected:

      generate ray in random direction $d'$

      return (1/ (1–$p_{emit}$)) * $f_r(d \rightarrow d')$ * $(n \cdot d')$ * TracePath($p'$, $d'$)

# TracePath

- TracePath($p$, $d$) returns (r,g,b):
  - Trace ray ($p$, $d$) to find nearest intersection $p'$
  - If Le = (0,0,0) then $p_{emit} = 0$
    else if $f_r = (0,0,0)$ then $p_{emit} = 1$
    else $p_{emit} = .9$
  - If random() < $p_{emit}$ then
    - Emitted:

      return $(1/\ p_{emit}) * (Le_{red}, Le_{green}, Le_{blue})$
    - Reflected:

      generate ray in random direction $d'$
      return $(1/\ (1-p_{emit})) * f_r(d \rightarrow d') * (n \cdot d') *$ TracePath($p'$, $d'$)

## TracePath

- Reflected case:
  - Pick a light source
  - Trace a ray towards that light
  - Trace a ray anywhere except for that light
    - Rejection sampling
  - Divide by probabilities
    - $p_{light}$ = 1/(solid angle of light) for ray to light source
    - (1 – the above) for non-light ray

# TracePath

- TracePath($p$, $d$) returns (r,g,b):
  - Trace ray ($p$, $d$) to find nearest intersection $p'$
  - If Le = (0,0,0) then $p_{emit} = 0$
    else if $f_r = (0,0,0)$ then $p_{emit} = 1$
    else $p_{emit} = .9$
  - If random() < $p_{emit}$ then
    - Emitted:

      return $(1/ p_{emit}) * (Le_{red}, Le_{green}, Le_{blue})$
    - Reflected:

```
generate ray in random direction d′ towards a light
$L_r = (1/2 * p_{light}) * f_r(d \rightarrow d') * (n \cdot d') *$ TracePath($p'$, $d'$)

generate ray in random direction d′ not towards the light
$L_r += (1/2*(1-p_{light})) * f_r(d \rightarrow d') * (n \cdot d') *$ TracePath($p'$, $d'$)
```

return $(1/ (1-p_{emit})) * L_r$

## Reflected Ray Sampling

- Uniform directional sampling:
  how to generate random ray on hemisphere?

## Reflected Ray Sampling

- Option #1: rejection sampling
  - Generate random numbers (x,y,z), with x,y,z in –1..1
  - If $x^2+y^2+z^2 > 1$, reject
  - Normalize (x,y,z)
  - If pointing into surface (ray dot n < 0), flip

## Reflected Ray Sampling

- Option #2: inversion method
  - In polar coords, density must be proportional to sin $\theta$ (remember $d$(solid angle) = sin $\theta\, d\theta\, d\phi$)
  - Integrate, invert $\rightarrow$ $\cos^{-1}$
- So, recipe is
  - Generate $\phi$ in 0..2π
  - Generate $z$ in 0..1
  - Let $\theta = \cos^{-1} z$
  - $(x,y,z) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$

## BRDF Importance Sampling

- Better than uniform sampling:
  *importance sampling*
- Because you divide by probability, ideally:
  probability $\propto f_r * \cos\theta_i$
- [Lafortune, 1994]:

$$f_r(x,\vec{\omega}_i,\vec{\omega}_o) = k_d\frac{1}{\pi} + k_s\frac{n+2}{2\pi}\cos^n\alpha$$

## BRDF Importance Sampling

- For cosine-weighted Lambertian:
  - Density = $\cos\theta \sin\theta$
  - Integrate, invert $\rightarrow \cos^{-1}(\text{sqrt})$
- So, recipe is:
  - Generate $\phi$ in $0..2\pi$
  - Generate $z$ in $0..1$
  - Let $\theta = \cos^{-1}(\text{sqrt}(z))$

## BRDF Importance Sampling

- Phong BRDF: $f_r \propto \cos^n \alpha$ where $\alpha$ is angle between outgoing ray and ideal mirror direction
- Constant scale = $k_s(\mathrm{n}+2)/(2\pi)$
- Ideally we would sample this times $\cos\ \theta_i$
  - Difficult!
  - Easier to sample BRDF itself, then multiply by $\cos\ \theta_i$
  - That's OK – still better than random sampling

## BRDF Importance Sampling

- Recipe for sampling specular term:
  - Generate $z$ in 0..1
  - Let $\alpha = \cos^{-1}(z^{1/(n+1)})$
  - Generate $\phi_\alpha$ in $0..2\pi$
- This gives direction w.r.t. ideal mirror direction

## BRDF Importance Sampling

- Recipe for combining terms:
  - r = random()
  - If (r < $k_d$) then
    - d′ = sample diffuse direction
    - weight = $1/k_d$
  - else if (r < $k_d + k_s$) then
    - d′ = sample specular direction
    - weight = $1/k_s$
  - else
    - terminate ray

## Recap

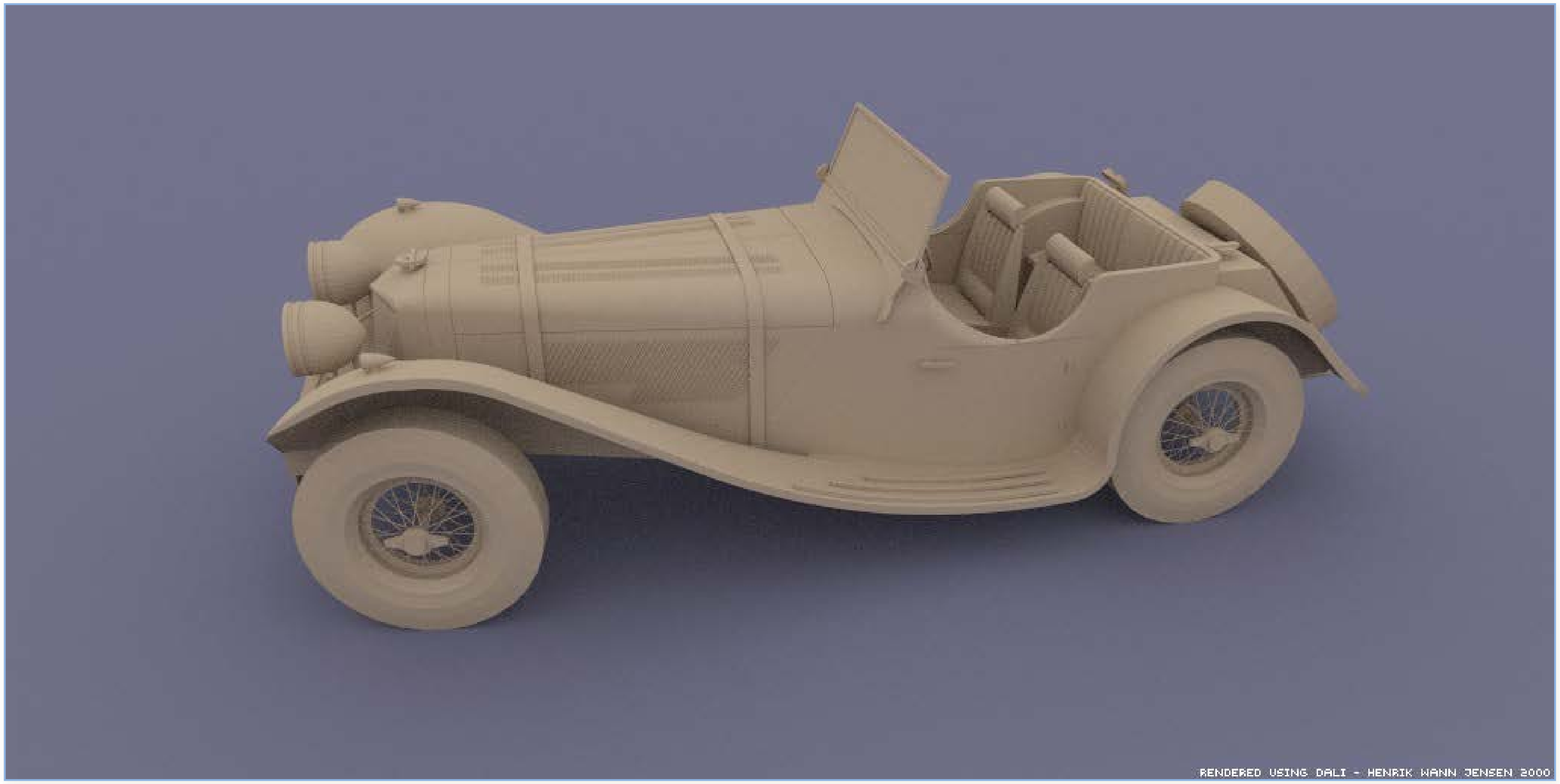
- TracePath($p$, $d$) returns (r,g,b):

```
– Trace ray ($p$, $d$) to find nearest intersection $p'$
– If Le = (0,0,0) then $p_{emit} = 0$
  else if $f_r = (0,0,0)$ then $p_{emit} = 1$
  else $p_{emit} = .9$
– If random() < $p_{emit}$ then
    • Emitted:
          return $(1/ p_{emit}) * (Le_{red}, Le_{green}, Le_{blue})$
    • Reflected:
          generate ray in random direction $d'$ towards a light
          $L_r = (1/2 * p_{light}) * f_r(d \rightarrow d') * (n \cdot d') *$ TracePath($p'$, $d'$)

          generate ray in random direction $d'$ not towards the light
          $L_r += (1/2*(1-p_{light})) * f_r(d \rightarrow d') * (n \cdot d') *$ TracePath($p'$, $d'$)

          return $(1/ (1-p_{emit})) * L_r$
```

## Monte Carlo Path Tracing

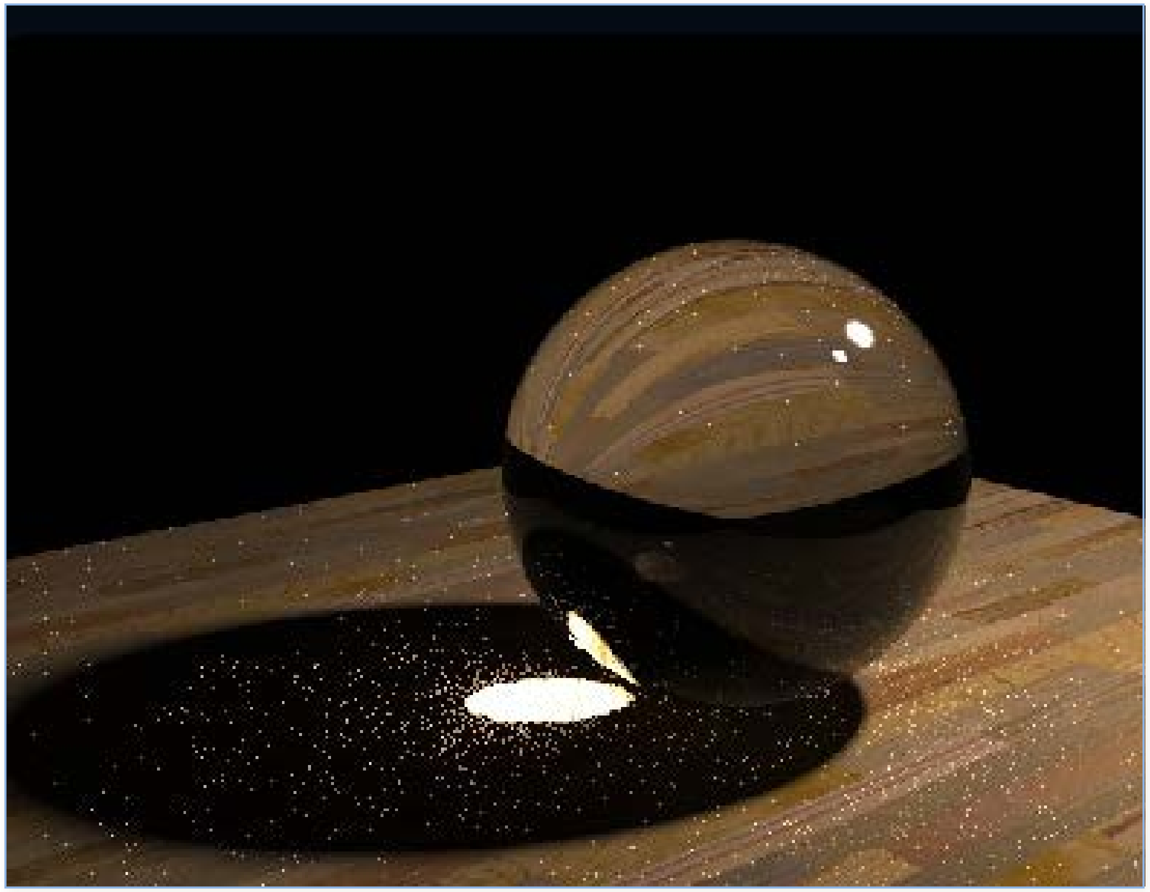
- Advantages
  - Any type of geometry (procedural, curved, ...)
  - Any type of BRDF (specular, glossy, diffuse, ...)
  - Samples all types of paths (L(SD)*E)
  - Accuracy controlled at pixel level
  - Low memory consumption
  - Unbiased - error appears as noise in final image
- Disadvantages
  - Slow convergence
  - Noise in final image

## Monte Carlo Path Tracing



Big diffuse light source, 20 minutes

## Monte Carlo Path Tracing


1000 paths/pixel

Jensen

## Summary

- Monte Carlo Integration Methods
  - Very general
  - Good for complex functions with high dimensionality
  - Converge slowly (but error appears as noise)
- Conclusion
  - Preferred method for difficult scenes
  - Noise removal (filtering) and
    irradiance caching (photon maps)
    used in practice

## More Information

- Books
  - *Realistic Ray Tracing*, Peter Shirley
  - *Realistic Image Synthesis Using Photon Mapping*, Henrik Wann Jensen
- Theses
  - *Robust Monte Carlo Methods for Light Transport Simulation*, Eric Veach
  - *Mathematical Models and Monte Carlo Methods for Physically Based Rendering*, Eric La Fortune
- Course Notes
  - *Mathematical Models for Computer Graphics*, Stanford, Fall 1997
  - *State of the Art in Monte Carlo Methods for Realistic Image Synthesis*, Course 29, SIGGRAPH 2001