CS580:
Monte Carlo Ray Tracing: Part I

Sung-Eui Yoon
(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/GCG

KAIST

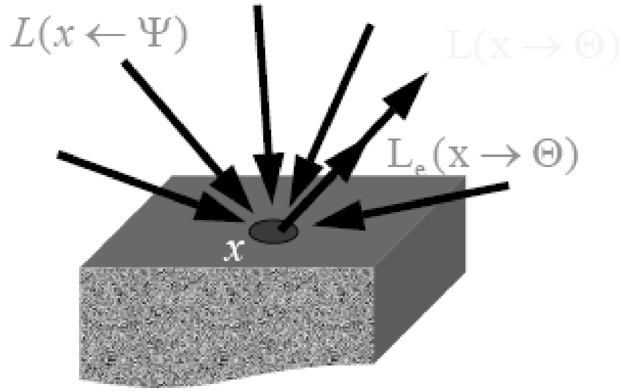
Class Objectives

- **Understand a basic structure of Monte Carlo ray tracing**
  - **Russian roulette for its termination**
  - **Stratified sampling**
- **Quasi-Monte Carlo ray tracing**

Why Monte Carlo?

- **Radiace is hard to evaluate**

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} f_r(\Psi \leftrightarrow \Theta) \cdot L(x \leftarrow \Psi) \cdot \cos(\Psi, n_x) \cdot d\omega_\Psi$$



From kavita's slides

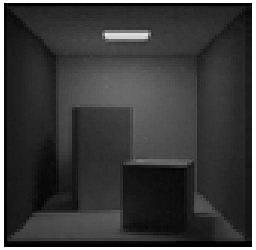
- **Sample many paths**
  - **Integrate over all incoming directions**
- **Analytical integration is difficult**
  - **Need numerical techniques**

Rendering Equation

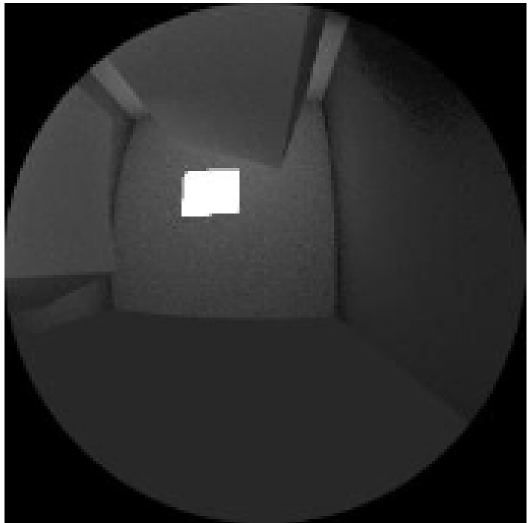
$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} f_r(\Psi \leftrightarrow \Theta) \cdot L(x \leftarrow \Psi) \cdot \cos(\Psi, n_x) \cdot d\omega_\Psi$$

function to integrate over all incoming directions over the hemisphere around x

Value we want



$$= L_e + \int_{\Omega_x}$$



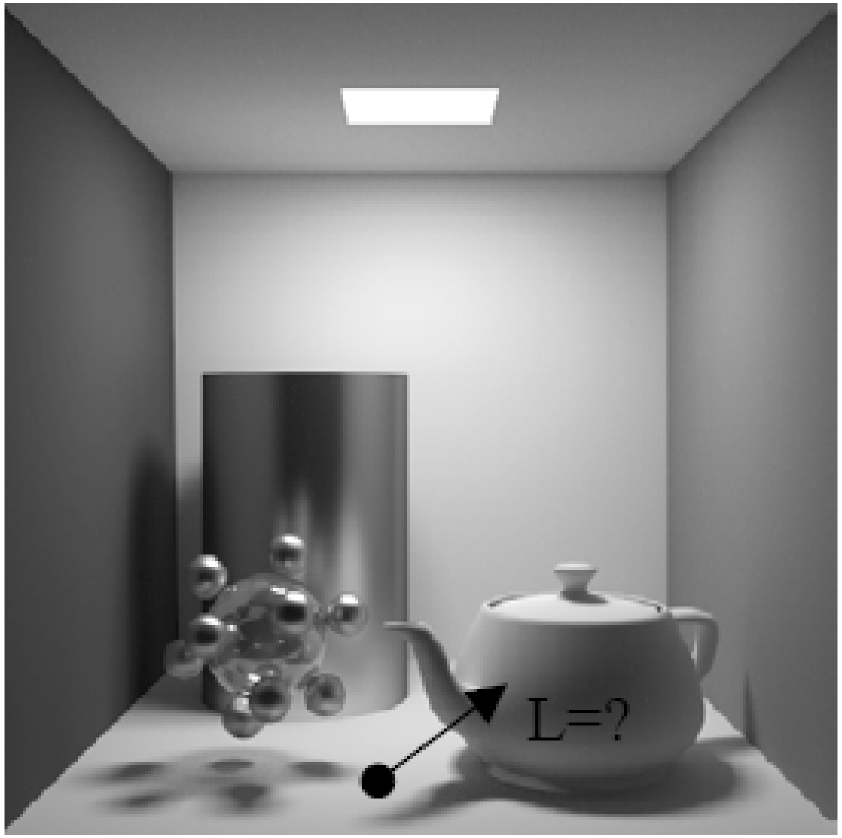
$$\cdot f_r \cdot \cos$$

How to compute?

$L(x \rightarrow \Theta) = ?$

Check for $L_e(x \rightarrow \Theta)$

Now add $L_r(x \rightarrow \Theta) =$



$$\int_{\Omega_x} f_r(\Psi \leftrightarrow \Theta) \cdot L(x \leftarrow \Psi) \cdot \cos(\Psi, n_x) \cdot d\omega_\Psi$$

How to compute?

- **Use Monte Carlo**
- **Generate random directions on hemisphere Ω_x using pdf $p(\Psi)$**

$$L(x \to \Theta) = \int_{\Omega_x} f_r(\Psi \leftrightarrow \Theta) \cdot L(x \leftarrow \Psi) \cdot \cos(\Psi, n_x) \cdot d\omega_\Psi$$

$$\langle L(x \to \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\Psi_i \leftrightarrow \Theta) \cdot L(x \leftarrow \Psi_i) \cdot \cos(\Psi_i, n_x)}{p(\Psi_i)}$$

How to compute?

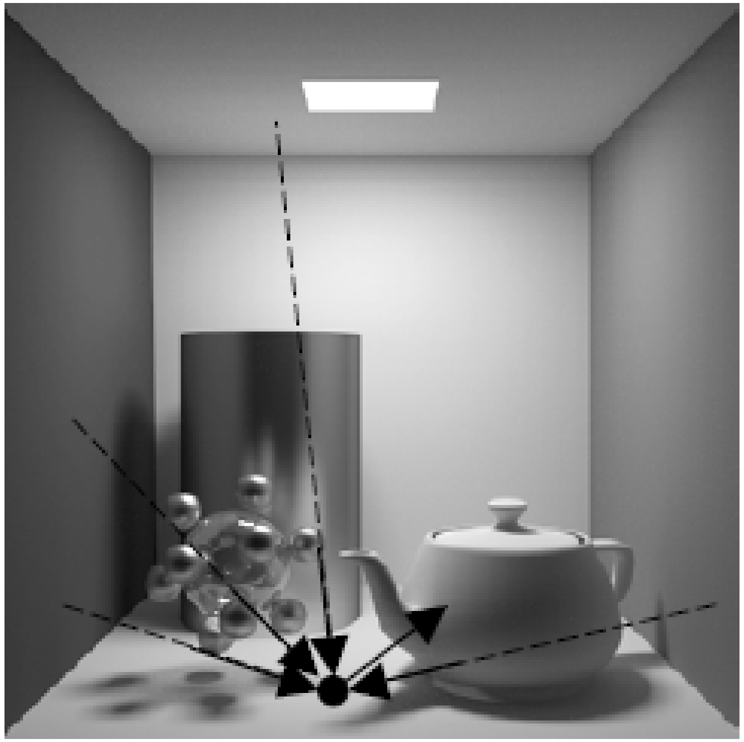
Generate random directions Ψ_i

$$\langle L \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f_r(\ldots) \cdot L(x \leftarrow \Psi_i) \cdot \cos(\ldots)}{p(\Psi_i)}$$

- evaluate brdf
- evaluate cosine term
- evaluate $L(x \leftarrow \Psi_i)$

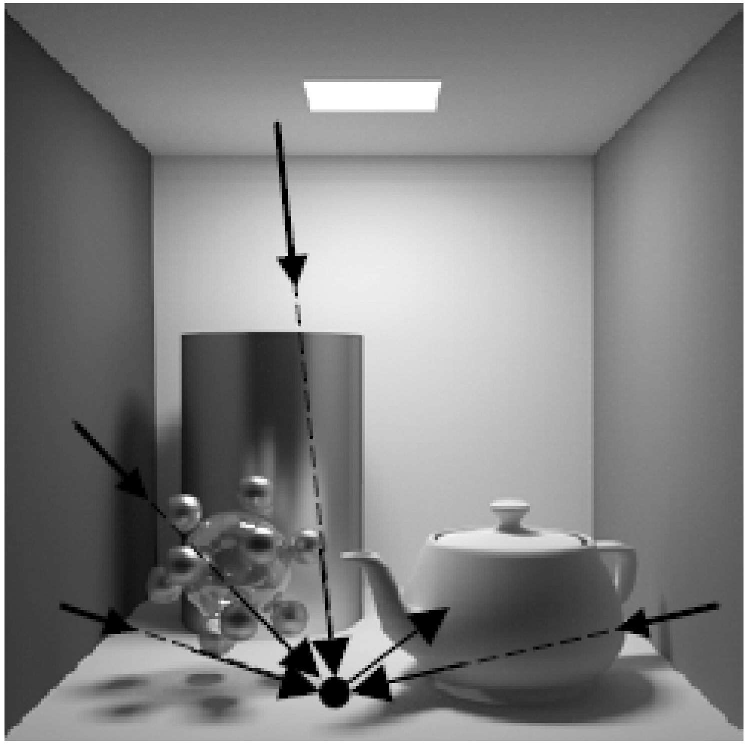
How to compute?

- evaluate $L(x \leftarrow \Psi_i)$?
- Radiance is invariant along straight paths
- $vp(x, \Psi_i)$ = first visible point
- $L(x \leftarrow \Psi_i) = L(vp(x, \Psi_i) \rightarrow \Psi_i)$

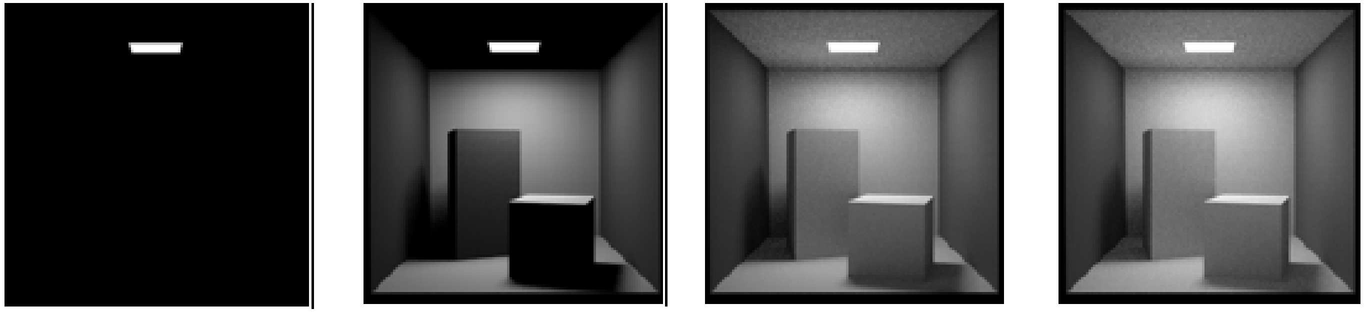


How to compute? Recursion ...

- Recursion ….
- Each additional bounce adds one more level of indirect light
- Handles ALL light transport
- "Stochastic Ray Tracing"



When to end recursion?



From kavita's slides

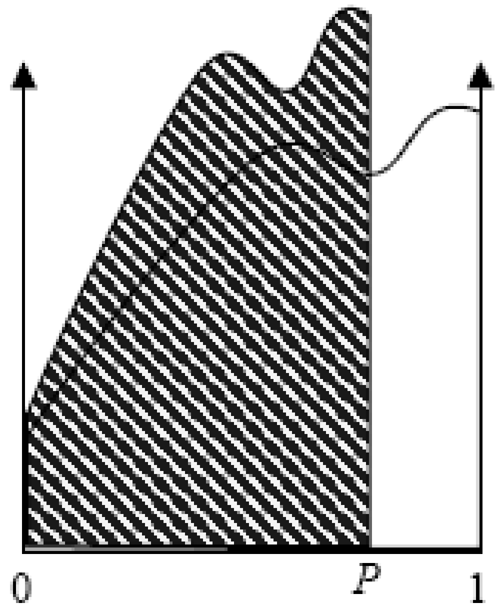
- Contributions of further light bounces become less significant
  - Max recursion
  - Some threshold for radiance value

- If we just ignore them, estimators will be biased

Russian Roulette

Integral

$$I = \int_0^1 f(x)dx = \int_0^1 \frac{f(x)}{P} P dx = \int_0^P \frac{f(y/P)}{P} dy$$



Estimator

$$\langle I_{roulette} \rangle = \begin{cases} \frac{f(x_i)}{P} & \textbf{if } x_i \le P, \\ 0 & \textbf{if } x_i > P. \end{cases}$$

Variance $\sigma_{roulette} > \sigma$

Russian Roulette

- **Pick absorption probability, α = 1-P**
  - **Recursion is terminated**

- **1- α is commonly to be equal to the reflectance of the material of the surface**
  - **Darker surface absorbs more paths**

KAIST

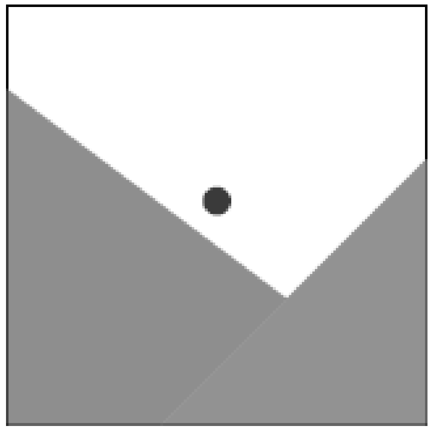
Algorithm so far

- Shoot primary rays through each pixel
- Shoot indirect rays, sampled over hemisphere
- Terminate recursion using Russian Roulette

KAIST

Pixel Anti-Aliasing

- **Compute radiance only at the center of pixel**
  - **Produce jaggies**

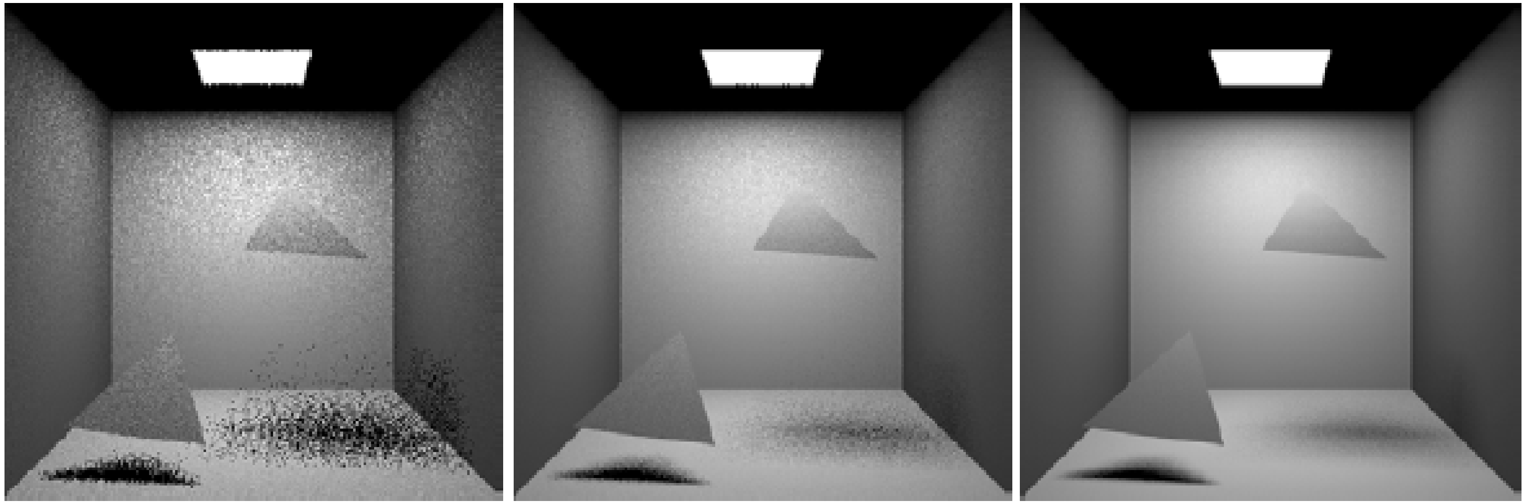


- **Simple box filter**
  - **The averaging method**
- **We want to evaluate using MC**

Stochastic Ray Tracing

- **Parameters**
  - **Num. of starting ray per pixel**
  - **Num. of random rays for each surface point (branching factor)**
- **Path tracing**
  - **Branching factor = 1**

Path Tracing



1 ray / pixel | 10 rays / pixel | 100 rays / pixel

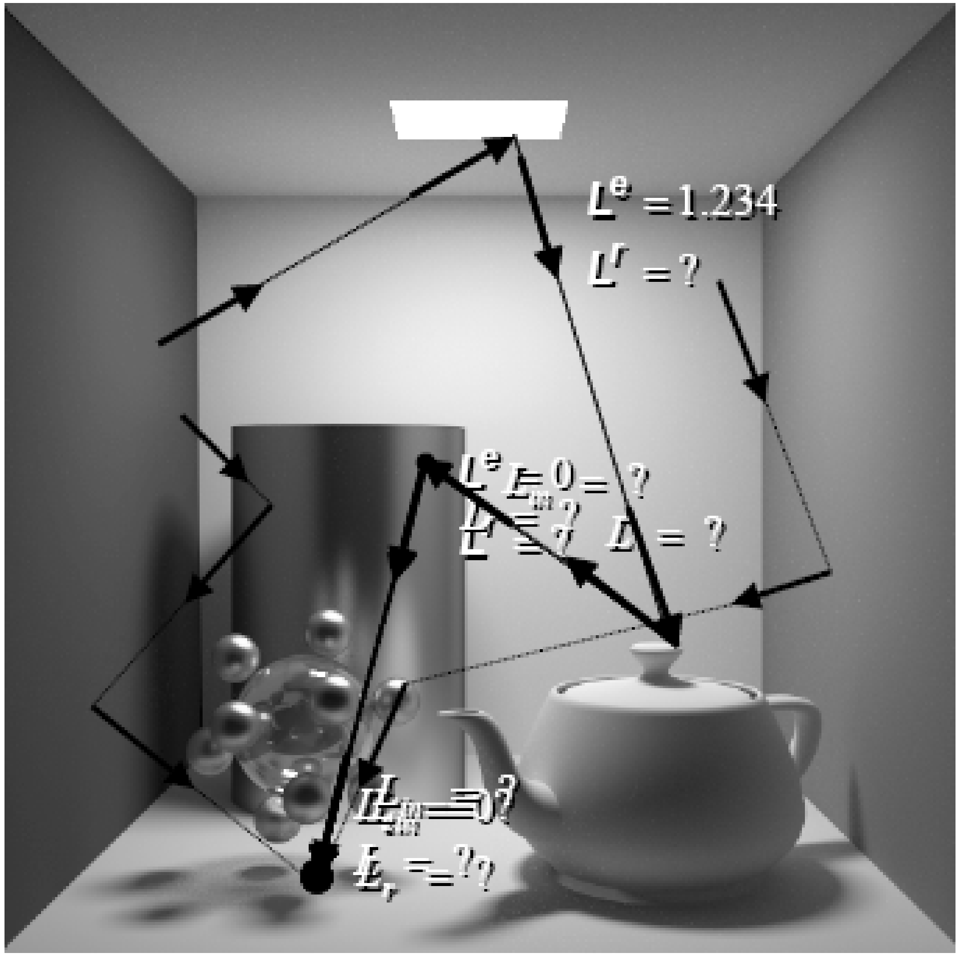
From kavita's slides

- **Pixel sampling + light source sampling folded into one method**

Algorithm so far

- Shoot primary rays through each pixel
- Shoot indirect rays, sampled over hemisphere
  - Path tracing shoots only 1 indirect ray
- Terminate recursion using Russian Roulette

Algorithm



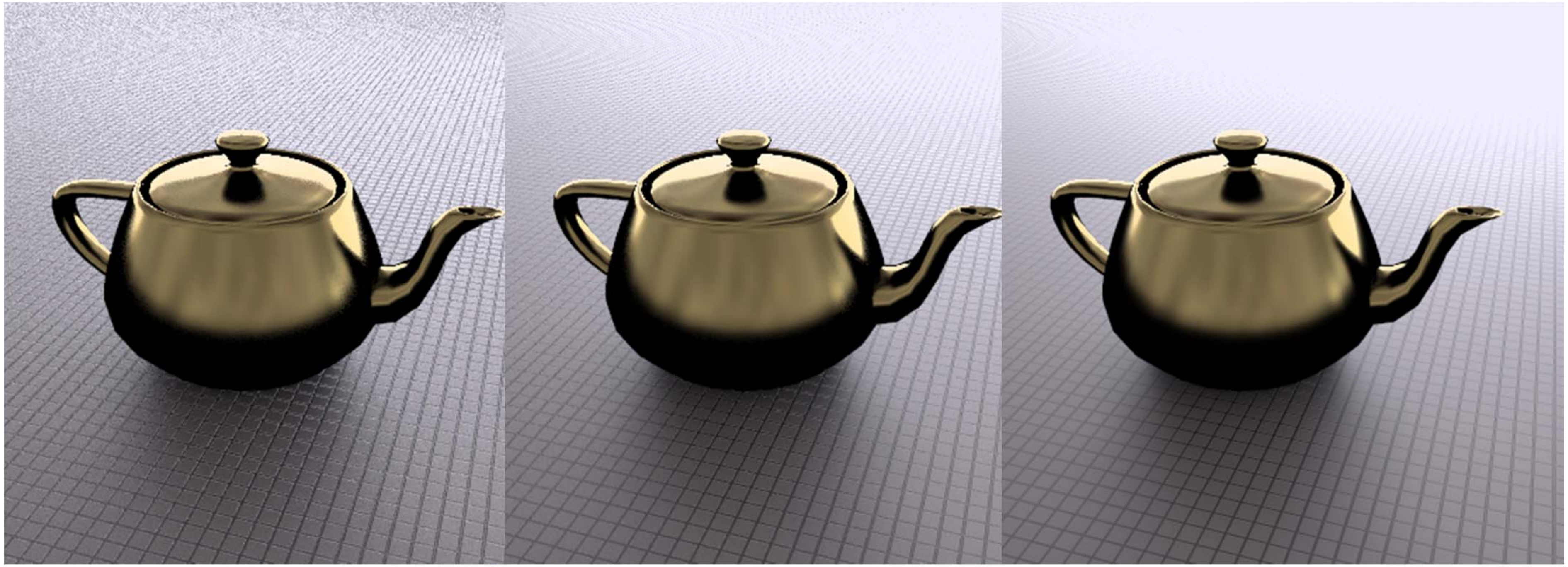
© Kavita Bala, Computer Science, Cornell University

Performance

- **Want better quality with smaller # of samples**
  - **Fewer samples/better performance**
  - **Stratified sampling**
  - **Quasi Monte Carlo: well-distributed samples**

- **Faster convergence**
  - **Importance sampling**

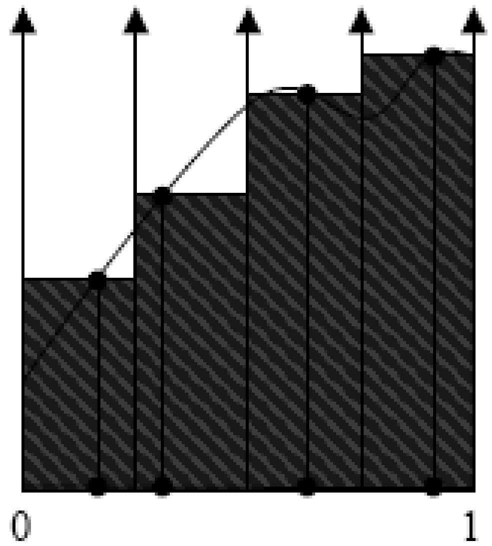
PA2



Uniform sampling (64 samples per pixel) | **Adaptive sampling** | **Reference**

Stratified Sampling

- Samples could be arbitrarily close



- Split integral in subparts

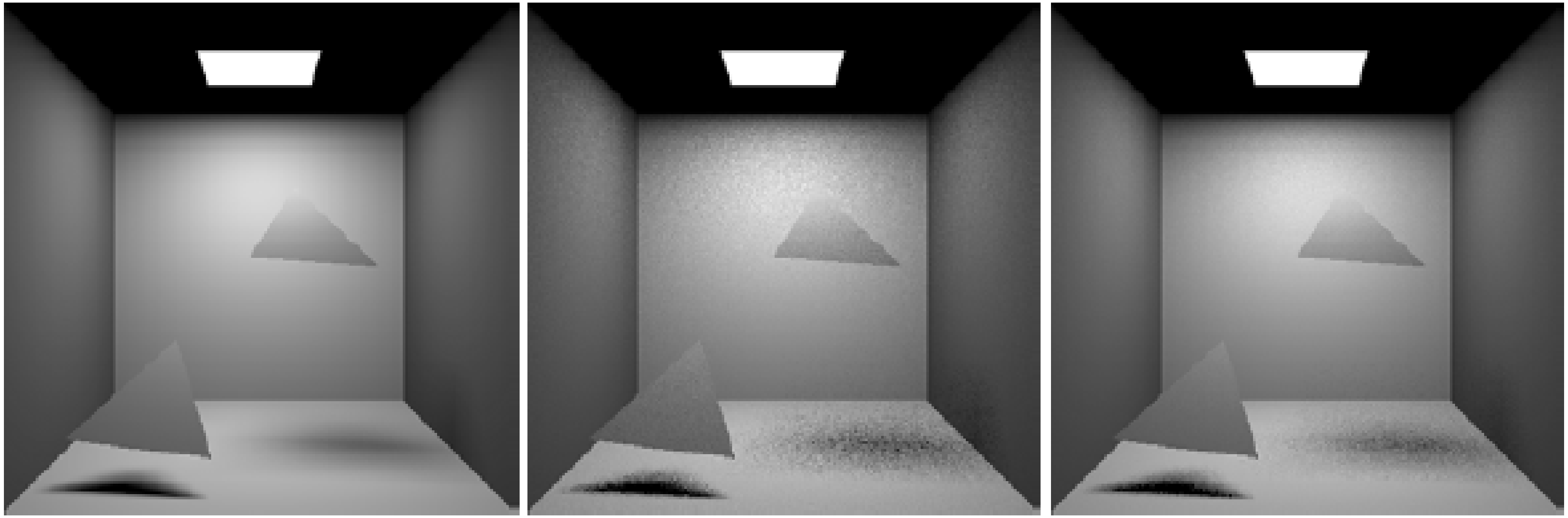
$$I = \int_{X_1} f(x)dx + \ldots + \int_{X_N} f(x)dx$$

- Estimator

$$\bar{I}_{strat} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\bar{x}_i)}{p(\bar{x}_i)}$$

- Variance: $\sigma_{strat} \leq \sigma_{sec}$

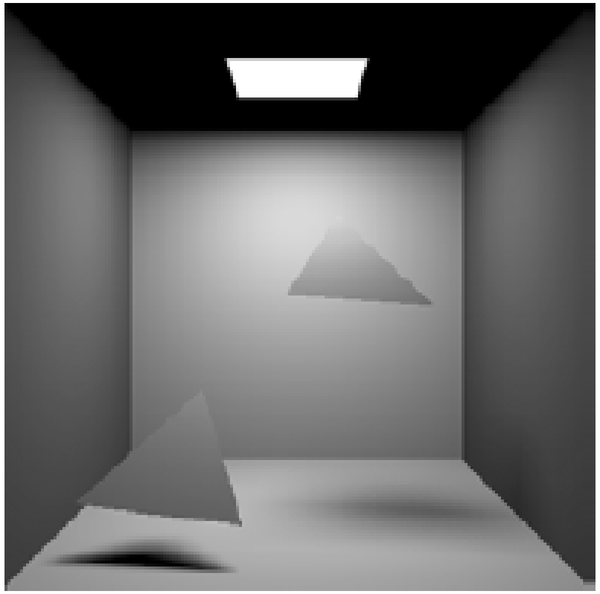
Stratified Sampling

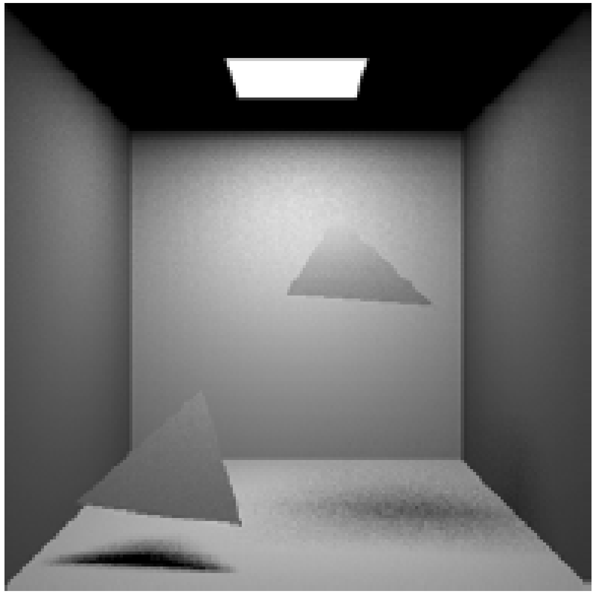


9 shadow rays
not stratified

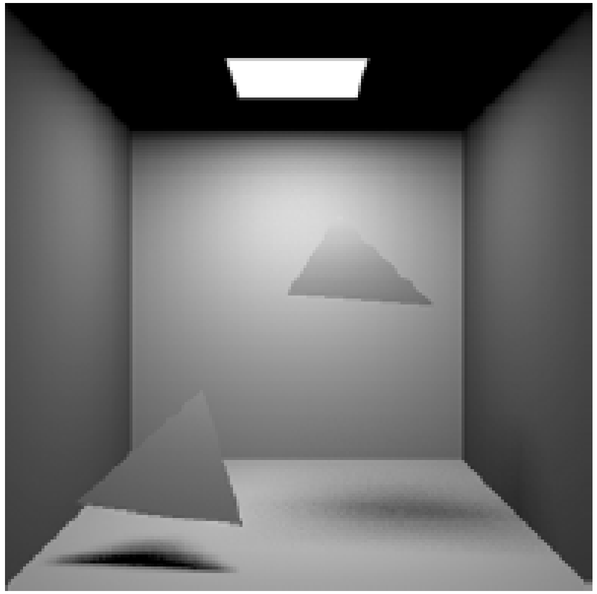
9 shadow rays
stratified

Stratified Sampling



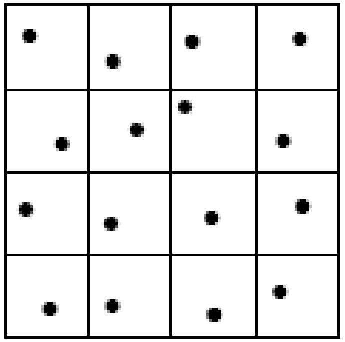


36 shadow rays
not stratified



36 shadow rays
stratified

High Dimensions



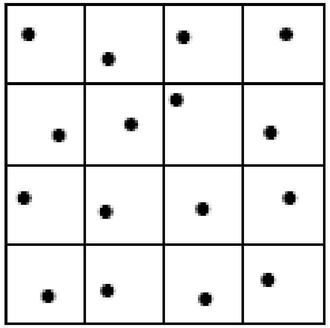
$\rightarrow N^2$ samples

- Problem for higher dimensions
- Sample points can still be arbitrarily close to each other

KAIST

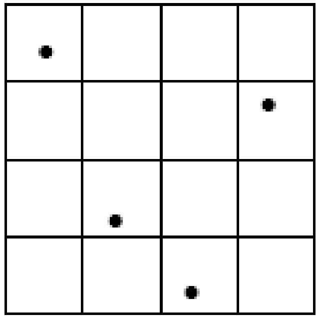
Higher Dimensions

- **Stratified grid sampling**



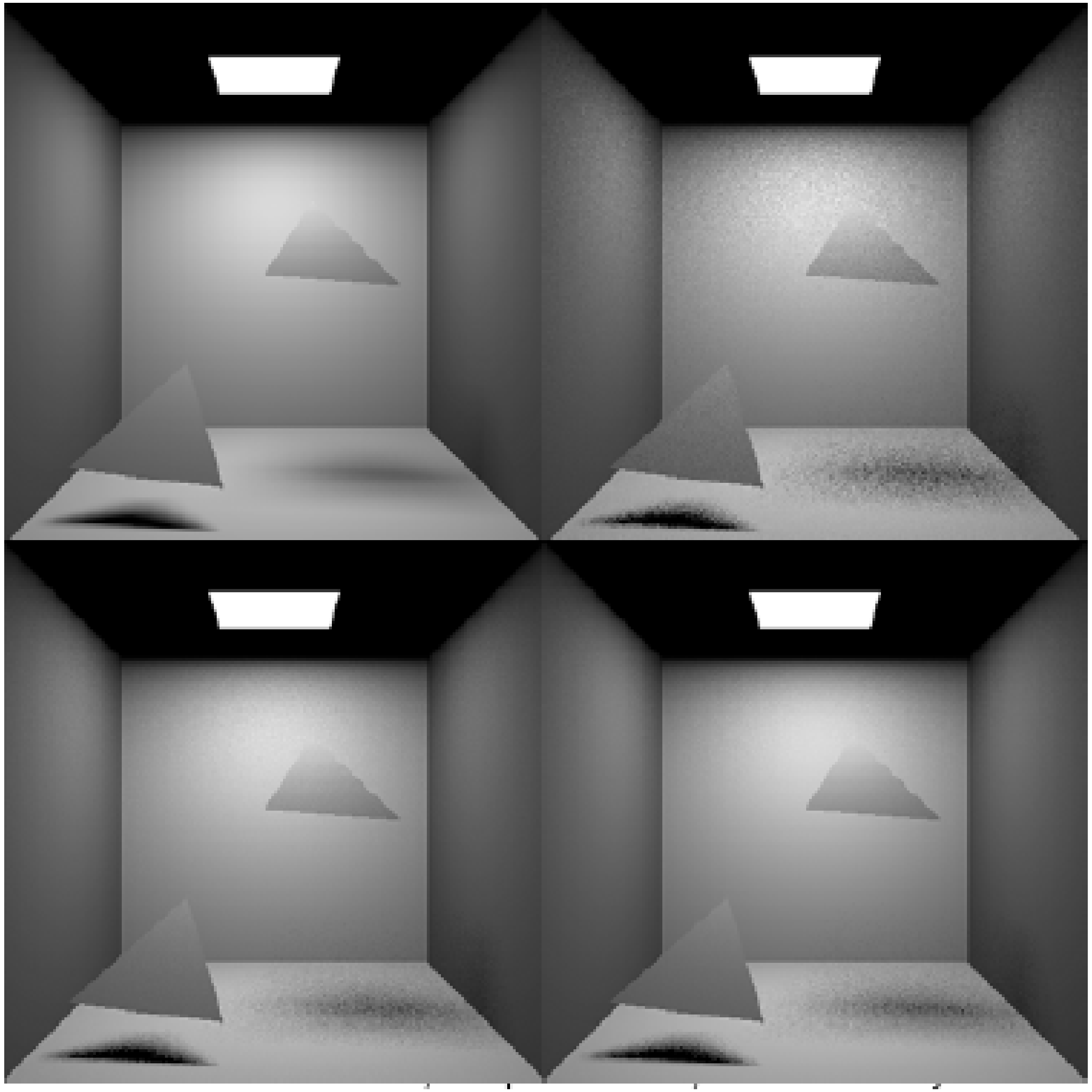
$\rightarrow N^d$ samples

- **N-rooks sampling**



$\rightarrow N$ samples

N-Rooks Sampling - 9 rays

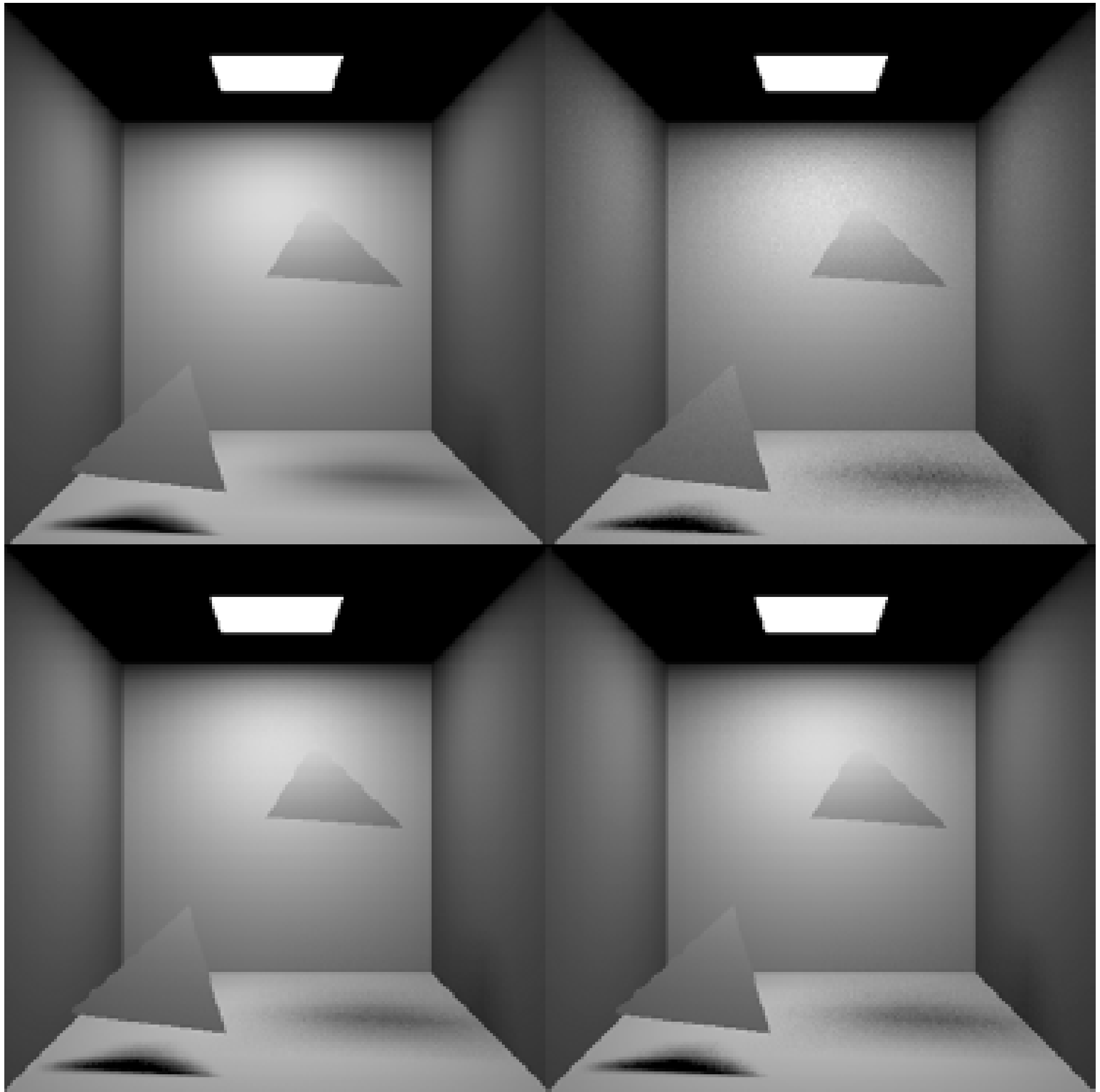


not stratified

stratified

N-Rooks

N-Rooks Sampling - 36 rays



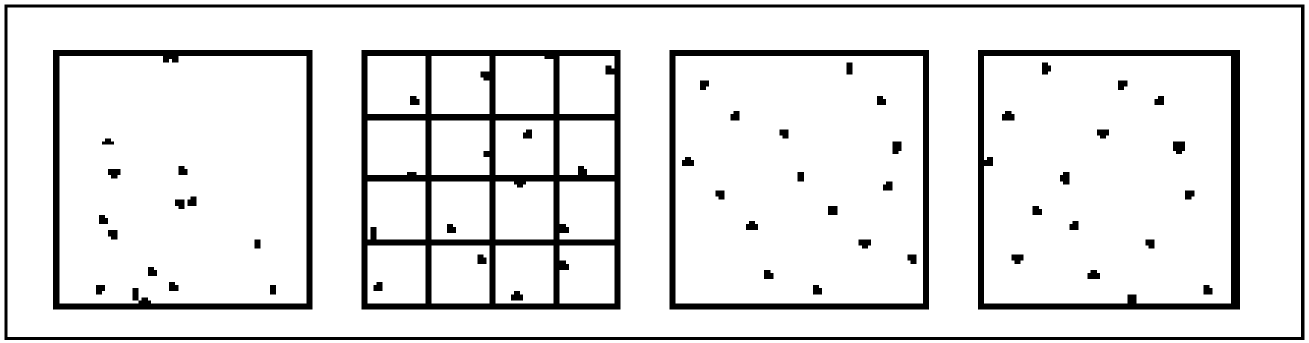
not stratified

stratified

N-Rooks

Quasi Monte Carlo

- Eliminates randomness to find well-distributed samples
- Samples are determinisitic but “appear” random



Quasi-Monte Carlo (QMC)

- Notions of variance, expected value don't apply

- Introduce the notion of discrepancy
  - Discrepancy mimics variance
  - E.g., subset of unit interval [0,x]
    - Of N samples, n are in subset
    - Discrepancy: |x-n/N|
  - Mainly: "it looks random"

Example: van der Corput Sequence

- One of simplest low-discrepancy sequences
- Radical inverse function, $\Phi_b(n)$
  - Given n = $\sum_{i=1}^{\infty} d_i b^{i-1}$,
  - $\Phi_b(n) = 0.d_1 d_2 d_3 \ldots d_n$
  - E.g., $\Phi_2(i)$: $111010_2 \rightarrow 0.010111$
- van der Corput sequence, $x_i = \Phi_2(i)$

Example: van der Corput Sequence

- **One of simplest low-discrepancy sequences**
- $x_i = \Phi_2(i)$

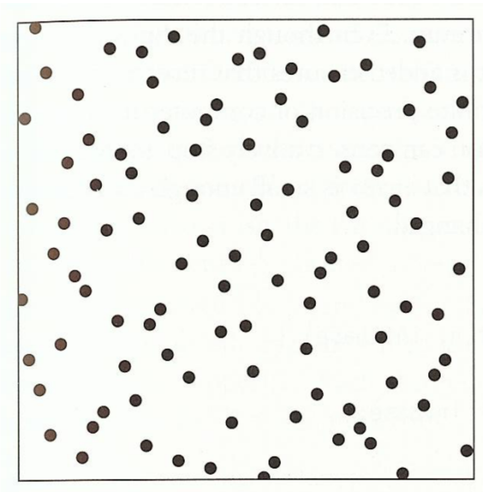
i	Base 2	$\Phi_2(i)$
1	1	.1 = 1/2
2	10	.01 = 1/4
3	11	.11 = 3/4
4	100	.001 = 1/8
5	101	.101 = 5/8
⋮	⋮	⋮

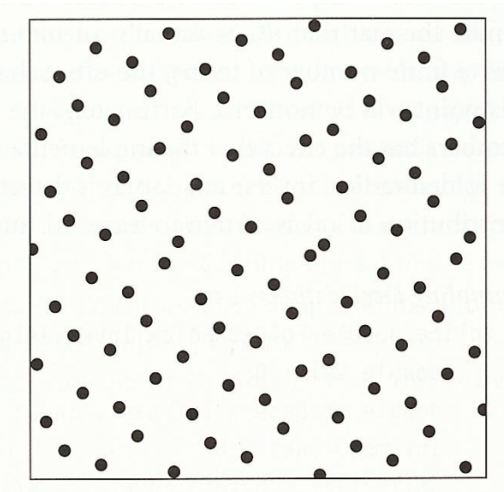
KAIST

Halton and Hammersley

- **Halton**
  - $x_i = (\Phi_2(i), \Phi_3(i), \Phi_5(i), \ldots, \Phi_{prime}(i))$
- **Hammersley**
  - $x_i = (1/N, \Phi_2(i), \Phi_3(i), \Phi_5(i), \ldots, \Phi_{prime}(i))$
  - Assume we know the number of samples, N
  - Has slightly lower discrepancy

Halton





Hammersley

Why Use Quasi Monte Carlo?

- **No randomness**
- **Much better than pure Monte Carlo method**
- **Converge as fast as stratified sampling**

Performance and Error

- **Want better quality with smaller number of samples**
  - **Fewer samples → better performance**
  - **Stratified sampling**
  - **Quasi Monte Carlo: well-distributed samples**

- **Faster convergence**
  - **Importance sampling: next-event estimation**

KAIST

Class Objectives were:

- Understand a basic structure of Monte Carlo ray tracing
  - Russian roulette for its termination
  - Stratified sampling
- Quasi-Monte Carlo ray tracing