Tutorial 9

TA: Mengyun Liu, Hongtu Xu

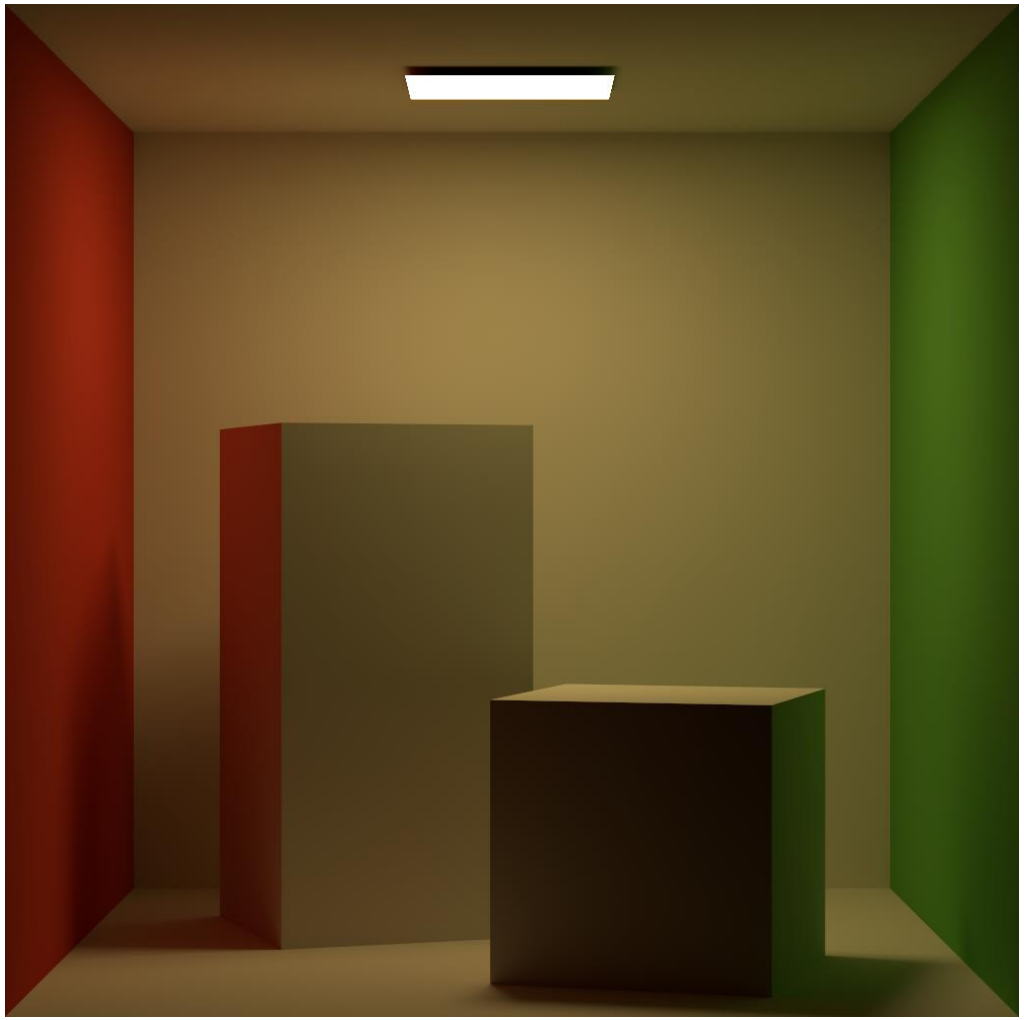
Homework 4

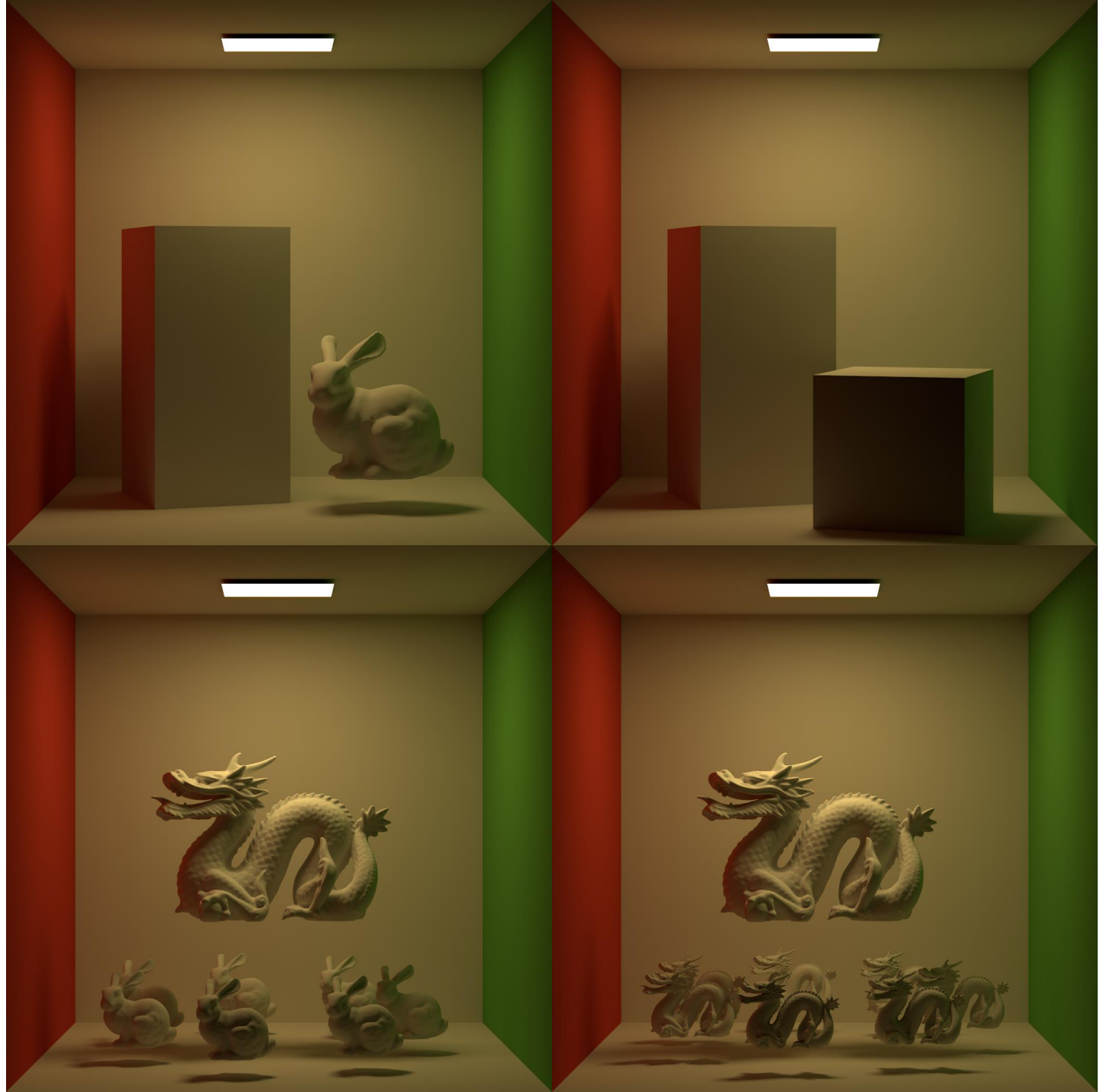
- Deadline: **22:00, Dec 9, 2021**
- Requirements:
  - Implement ray-mesh intersection test with a k-d tree for acceleration.
  - Implement the ideal diffusion BRDF with an area light source.
  - Implement the Monte-Carlo path tracing algorithm for global illumination: direct + indirect lighting.

Homework 4

- **[must]** Implement ray-mesh intersection test with a k-d tree for acceleration. [40%]
- **[must]** Implement the ideal diffusion BRDF with an area light source. [20%]
- **[must]** Implement the Monte-Carlo path tracing algorithm for global illumination: direct + indirect lighting. [40%]
- **[optional]** Implement hybrid spatial partitioning, i.e., uniform grid with a k-d tree. [10%]
- **[optional]** Implement the ideal specular and glossy specular BRDF. In particular, you need to implement the multiple importance sampling to handle these BRDFs. [10%]
- **[optional]** Implement the translucent BRDF with refraction, e.g. glasses and diamonds. [10%]
- **[optional]** Implement the bidirectional path tracing. [10%]
- **[optional]** Implement the Metropolis light transport. [5%]

Homework 4 gallery





Code Structures

- accel.h/cpp: defines acceleration structures, AABB and K-D tree.
- brdf.h/cpp: defines classes and interfaces for BRDFs, e.g., IdealDiffusion
- geometry.h/cpp: triangles and triangle mesh
- integrator.h/cpp: PathIntegrator
- interaction.h/cpp: define a data structure used for recording intersection info
  - New properties: wi (direction of incoming radiance), wo (direction of outcoming radiance), brdf (BRDF at the intersecting point);
- light.h/cpp: defines classes for modeling lights, e.g., AreaLight
- utils.h: define utility functions (unif)

Interfaces to Implement

- KdTreeNode::KdTreeNode
  - construct kd tree with given triangles, space (AABB), depth
  - divide the space into two parts according to one specific dimension
  - recursively build left and right
- KdTreeNode::intersect
  - traverse the k-d tree to check whether ray intersect with triangles
  - first check whether ray hit the bounding box
  - recursively test intersection with left and right children
  - do ray-triangle intersection in the leaf node

Interfaces to Implement

- IdeaDiffusion::sample
  - Sample a direction according to the BRDF, store the sampled direction in the given interaction. Also, the PDF of this sample should be returned.
  - `Float sample(Interaction &interact);`
- IdealDiffusion::pdf
  - Compute the PDF of the given BRDF sample at the specified interaction. You may need to use the interact.wi and interact.wo.
  - `Float IdealDiffusion::pdf(const Interaction &interact);`
- IdealDiffusion::eval
  - Evaluate the BRDF, namely, return the BRDF value at the given interaction
  - `vec3 IdealDiffusion::eval(const Interaction &interact);`

Interfaces to Implement

- AreaLight::emission
  - Get the emission at the specified position along the given direction.
  - `vec3 AreaLight::emission(vec3 pos, vec3 dir);`
- AreaLight::sample
  - Sample a position on the light and obtain the corresponding PDF.
  - `vec3 AreaLight::sample(Interaction &refIt, Float *pdf);`
- AreaLight::pdf
  - Compute the PDF of the given light sample.
  - `Float AreaLight::pdf(const Interaction &refIt, vec3 pos);`

Interfaces to Implement

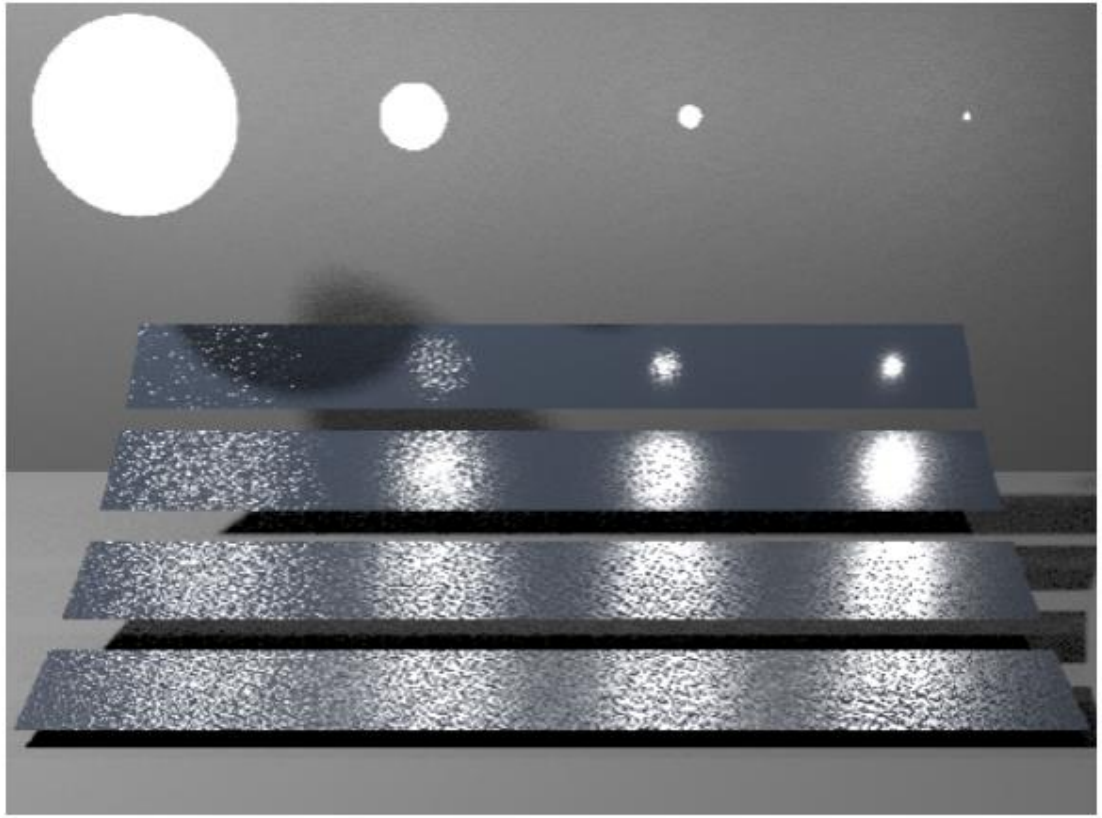
- PathIntegrator::render
  - put your anti-aliasing code inside the render loop
- PathIntegrator::radiance
  - Compute the radiance brought by the given ray.
  - `vec3 PathIntegrator::radiance(Scene &, const Ray &) const;`
- You may need to add your own functions.
- Recall basic ray tracing work flow in assignment 3.

Path tracing

- Sample multiple paths for each pixel and compute the contribution of each path. The contribution is weighted by the sampling Pdf.
- Set a max depth. Depth = times of reflection
- Implementation (iterative)
  - Beta = 1, L = 0
  - For each reflection at p
  - (1) L += Beta * Direct lighting at p
  - (2) Sample the next ray according to BRDF and find the corresponding Pdf
  - (3) Beta *= BRDF *cosΘ / Pdf
  - (4) Spawn the new ray

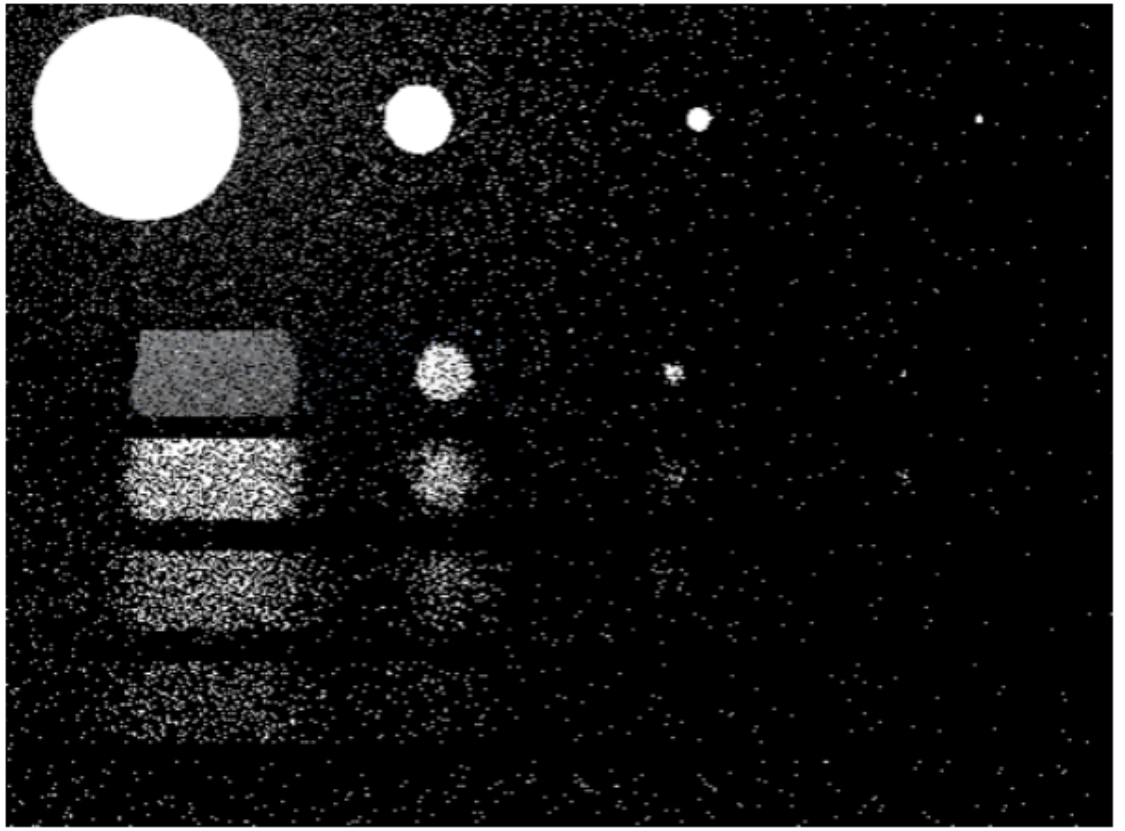
Direct Lighting

- Sampling-based method:
  - Sample light
    - Sample a point from the area light and find the corresponding Pdf.
    - If the light sample is not occluded
      - L = (Compute the radiance produced by the light sample according to the rendering equation) / Pdf.



Direct Lighting

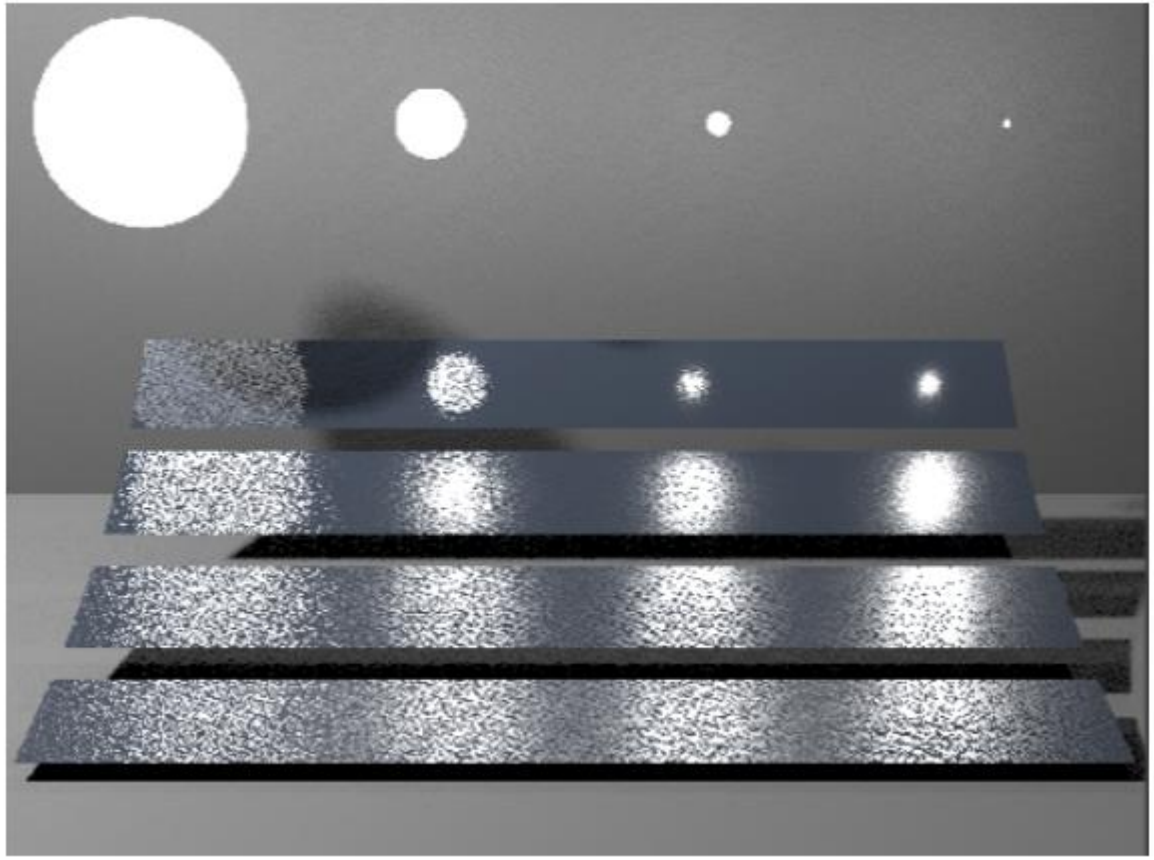
- Sampling-based method:
  - Sample BRDF (optional)
    - Sample a ω_i according to the BRDF and find the corresponding Pdf.
    - If there is a light along the ω_i without occlusion
      - L = (Compute the radiance produced by the light sample according to the rendering equation) / Pdf.



Direct Lighting

- Sampling-based method:
  - Sample light and BRDF (Multiple importance sampling) (optional)
    - Find the radiance of sampling light.
    - Find the radiance of sampling brdfs.
    - Weighted sum ($p_f(x), p_g(x)$ are Pdfs for the two sampling methods):

$$\frac{1}{n_f}\sum_{i=1}^{n_f}\frac{f(X_i)g(X_i)w_f(X_i)}{p_f(X_i)}+\frac{1}{n_g}\sum_{j=1}^{n_g}\frac{f(Y_j)g(Y_j)w_g(Y_j)}{p_g(Y_j)}$$



Notes for Sampling

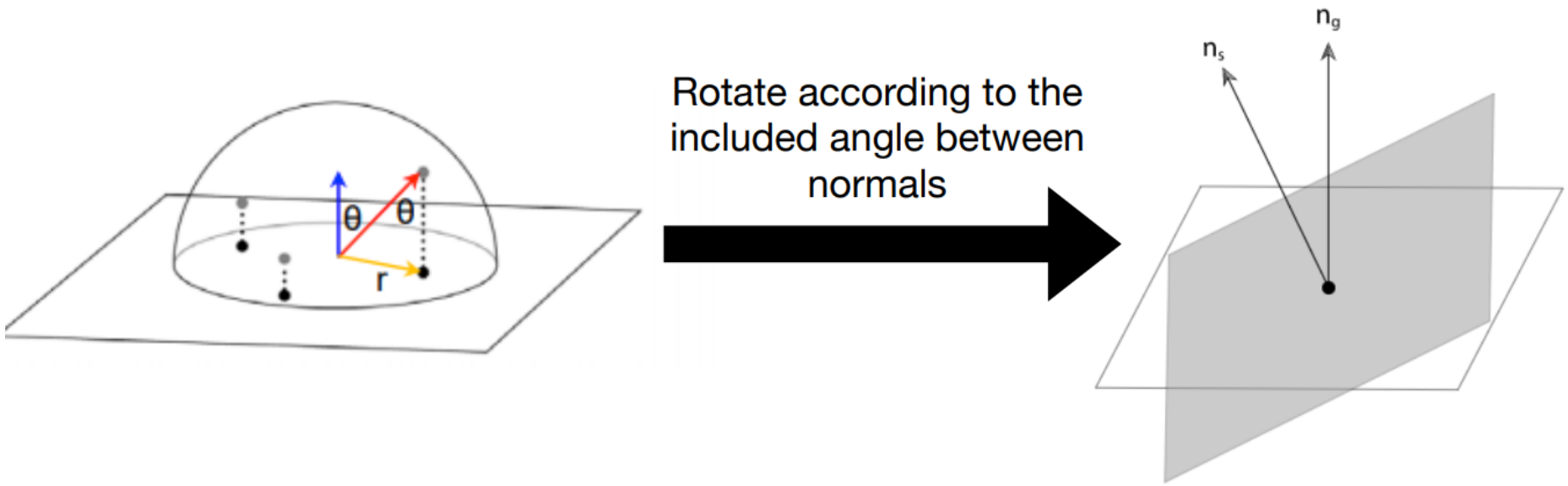
- utils.h: unif(a, b, N)
  - Get N uniform samples with range [a, b]

- Not efficient, especially for multithreading.

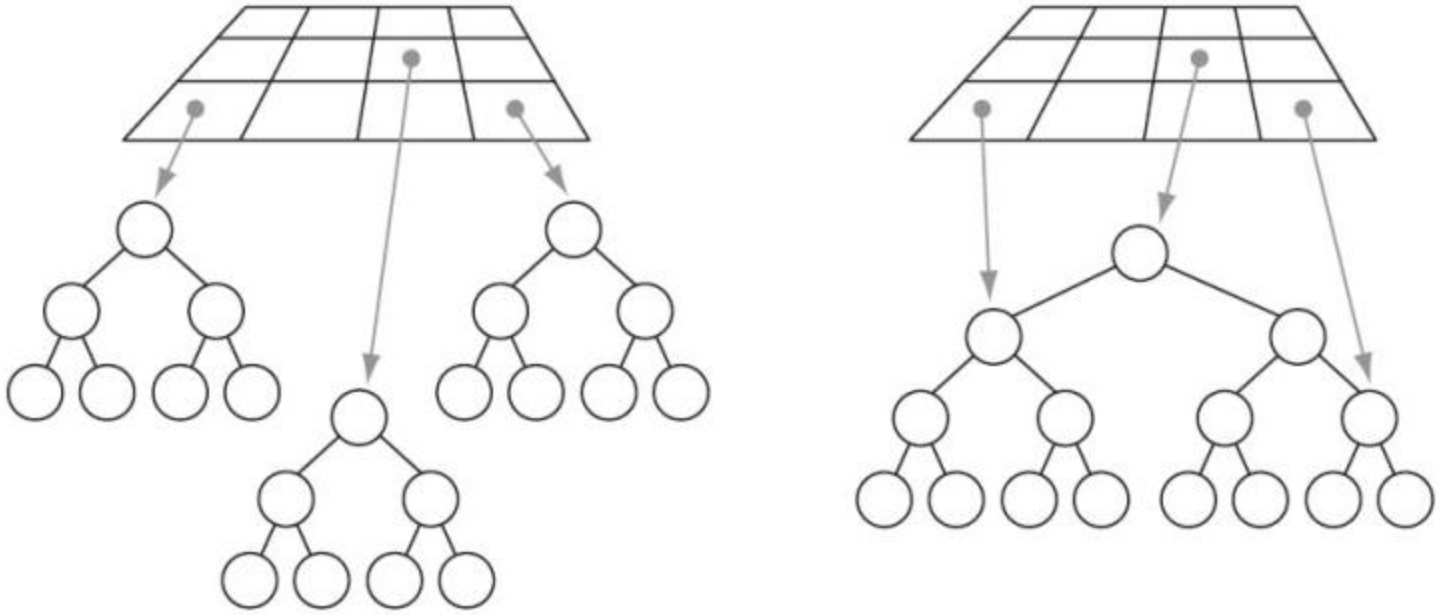
- You can implement a sampler class to maintain a random number generator per thread.

Notes for Sampling Directions

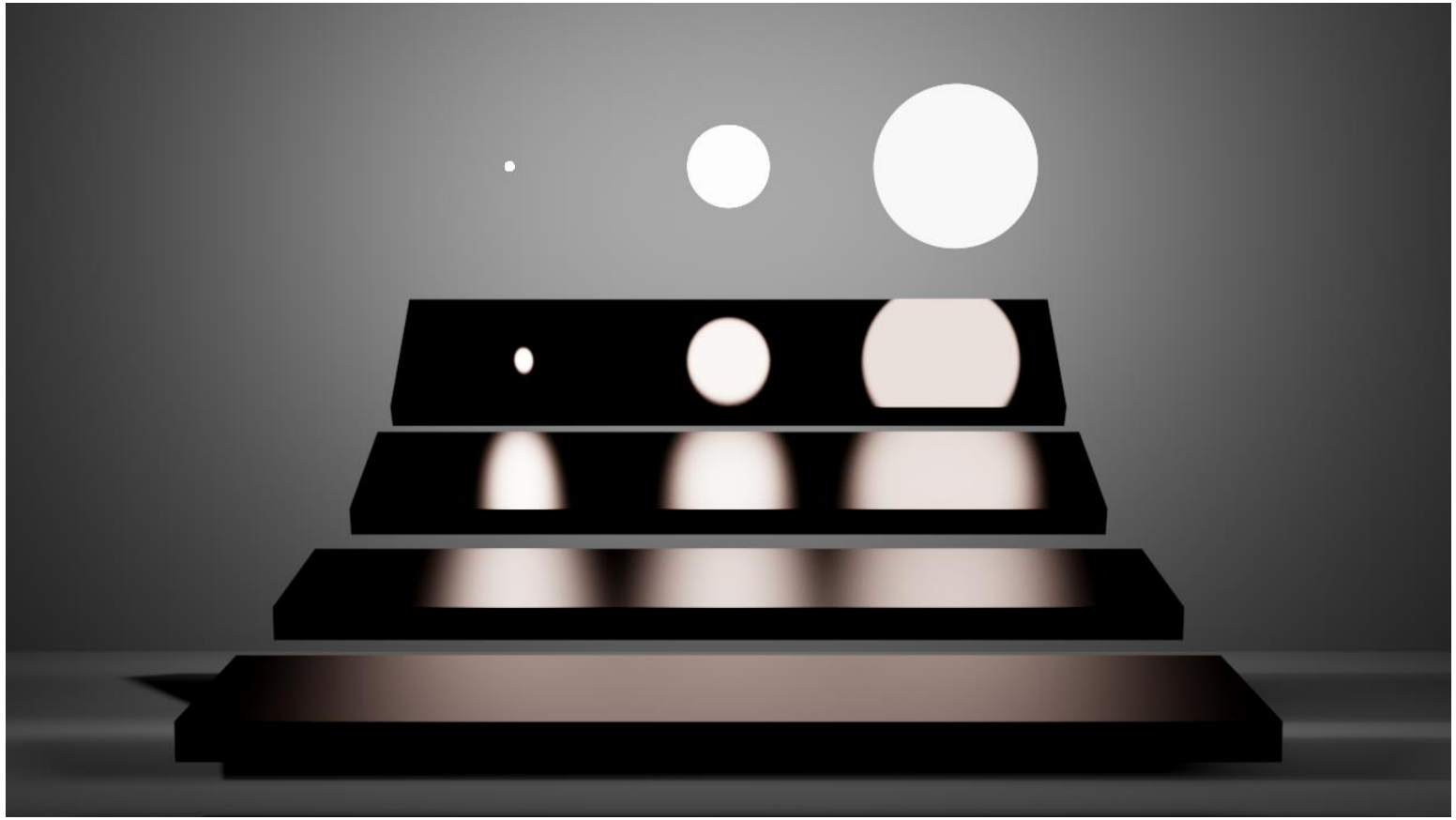
- The original sampled directions are relative to the local coordinate systems.
- You need to transform samples to the world coordinate systems.
- Useful function: `Eigen::Quaternionf::FromTwoVectors(src, des)`



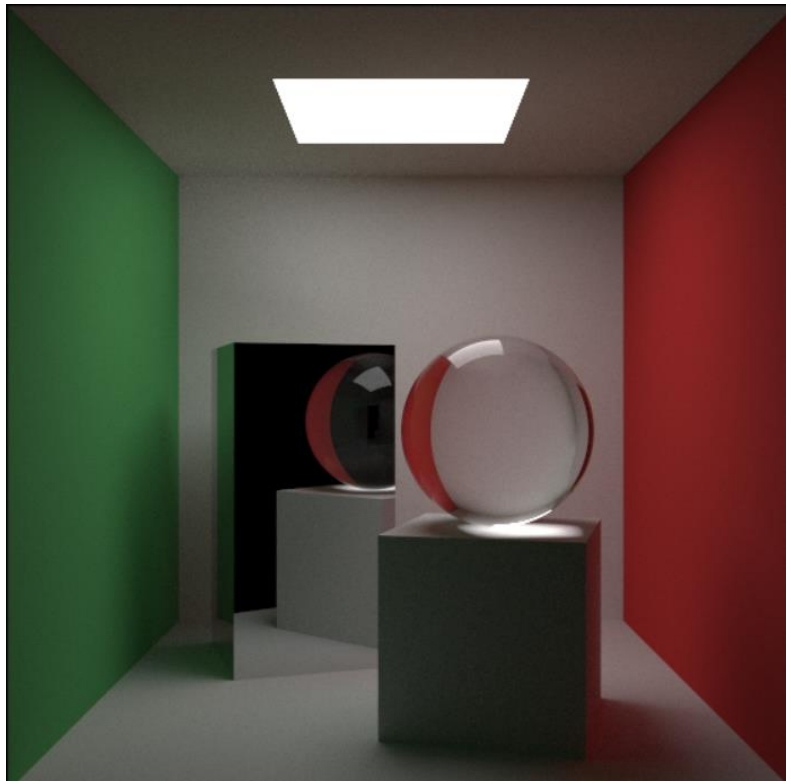
Hybrid spatial partitioning

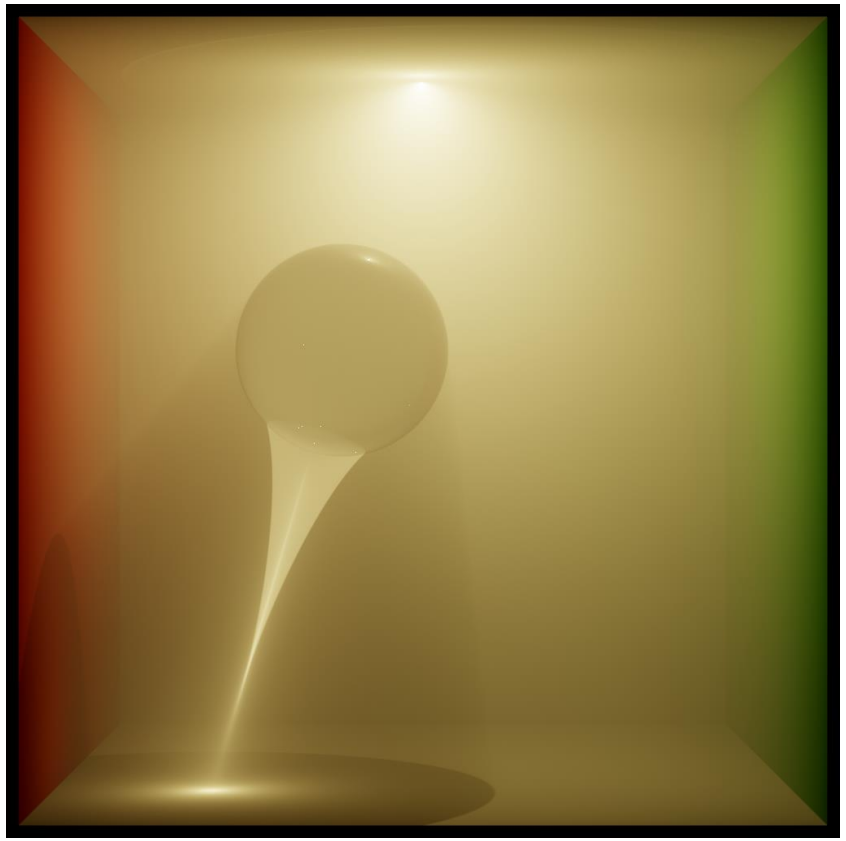


Glossy Specular BRDF (MIS)



Translucent BRDF (caustics)





Bidirectional path tracing

Metropolis light transport

Thanks