Lab #5 – Photon mapping (part 2)

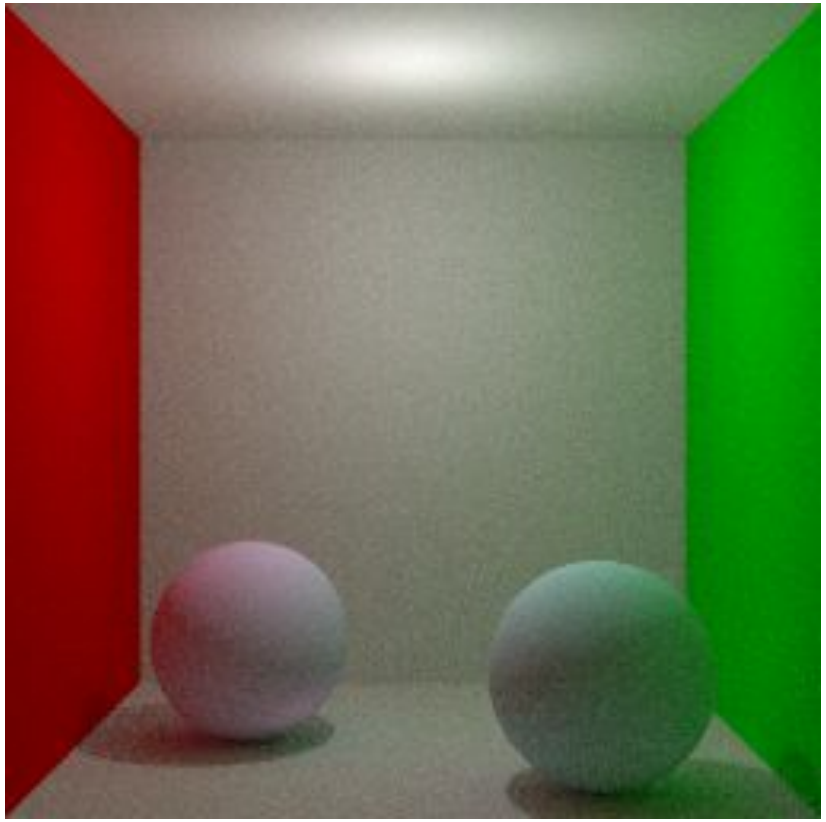
Informática Gráfica

Adolfo Muñoz - Julio Marco
Pablo Luesia - J. Daniel Subías – Óscar Pueyo

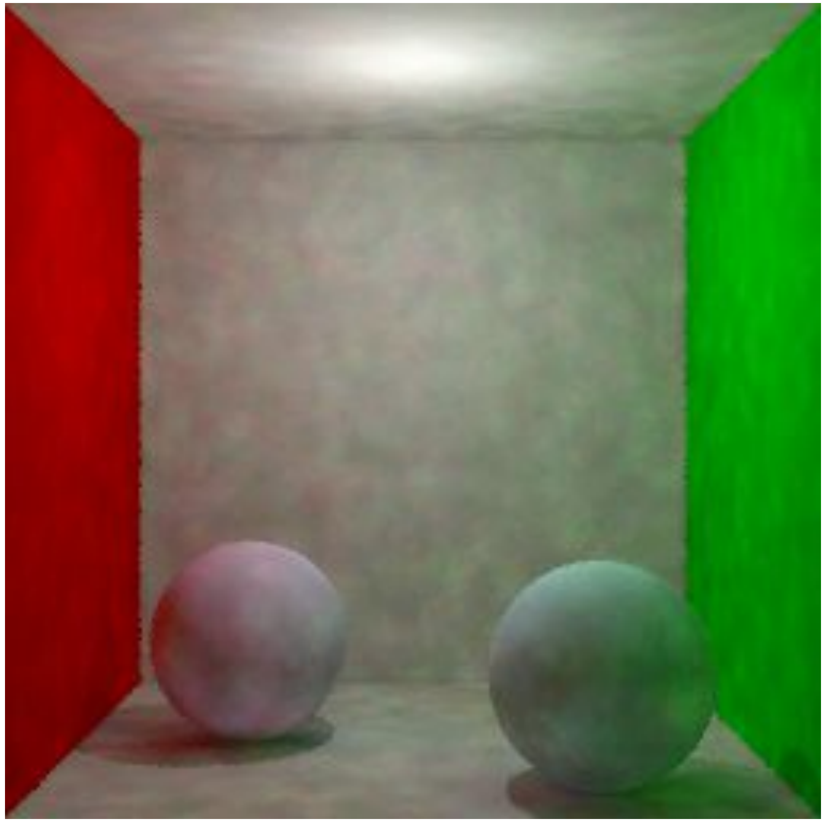
Graphics and Imaging Lab

Before we begin…

- Today: render the first image with photon mapping



Path tracing



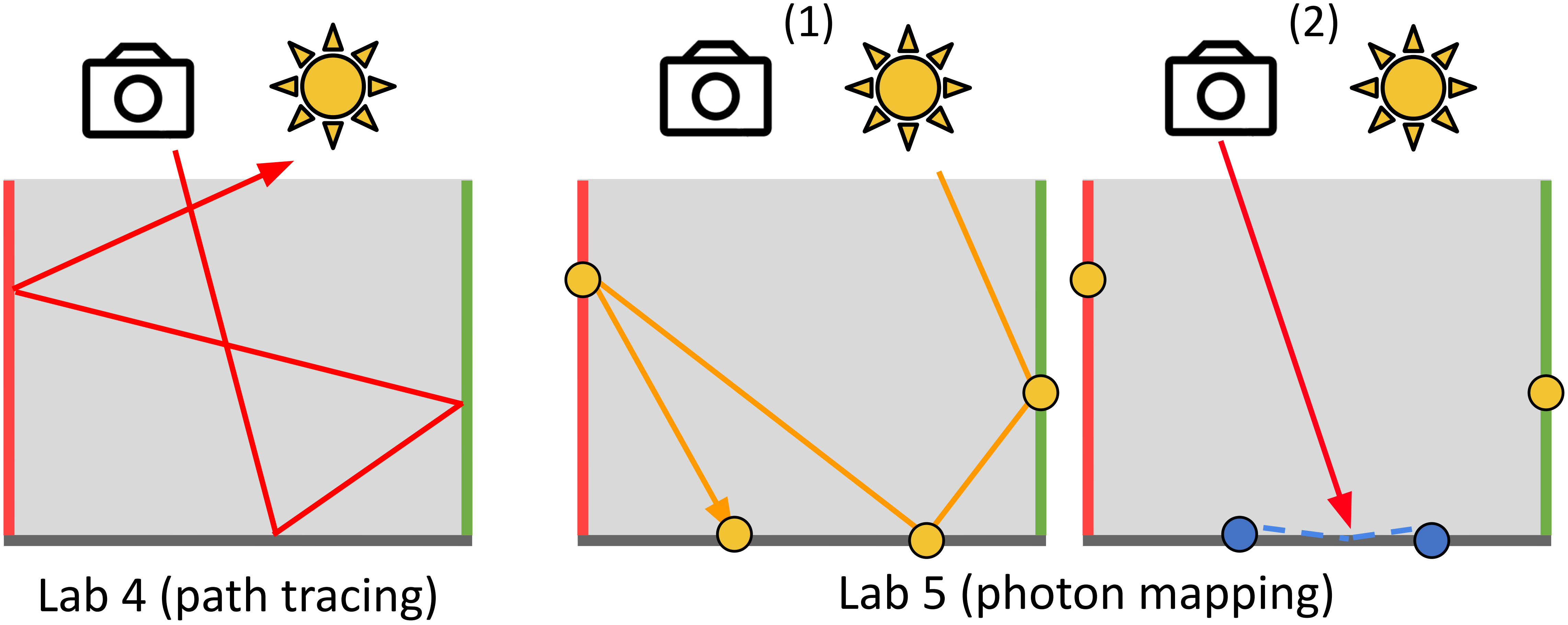
Photon mapping

Before we begin…

- Lab 5 (photon mapping) **is the second submitted work**
  - Recommended deadline: December 4th
  - Moodle: January 11th
- You can probably **reuse most of your code** for this assignment
- Remember: Final work is 80% of the final grade

Recap: photon mapping basics

- Photon mapping is a **two-pass** algorithm
  - Uses an **intermediate storage** known as the **photon map**

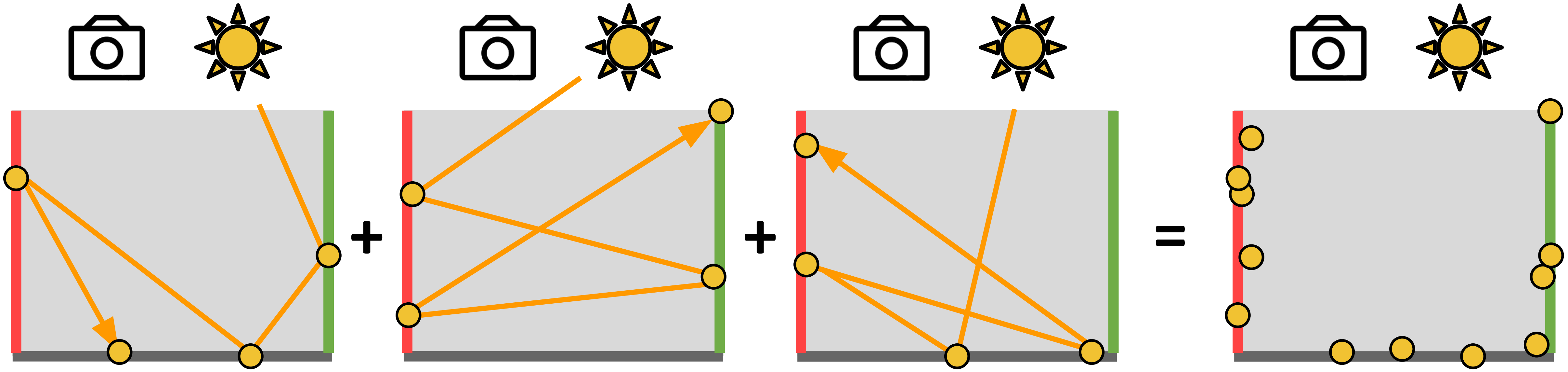


Recap: photon map construction

Graphics and Imaging Lab

- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map
- Photons are sent out from the light sources (**photon random walks**)

Final photon map



Recap: photon map construction

The origin of the photons is the light source

- **Conservation of energy:**

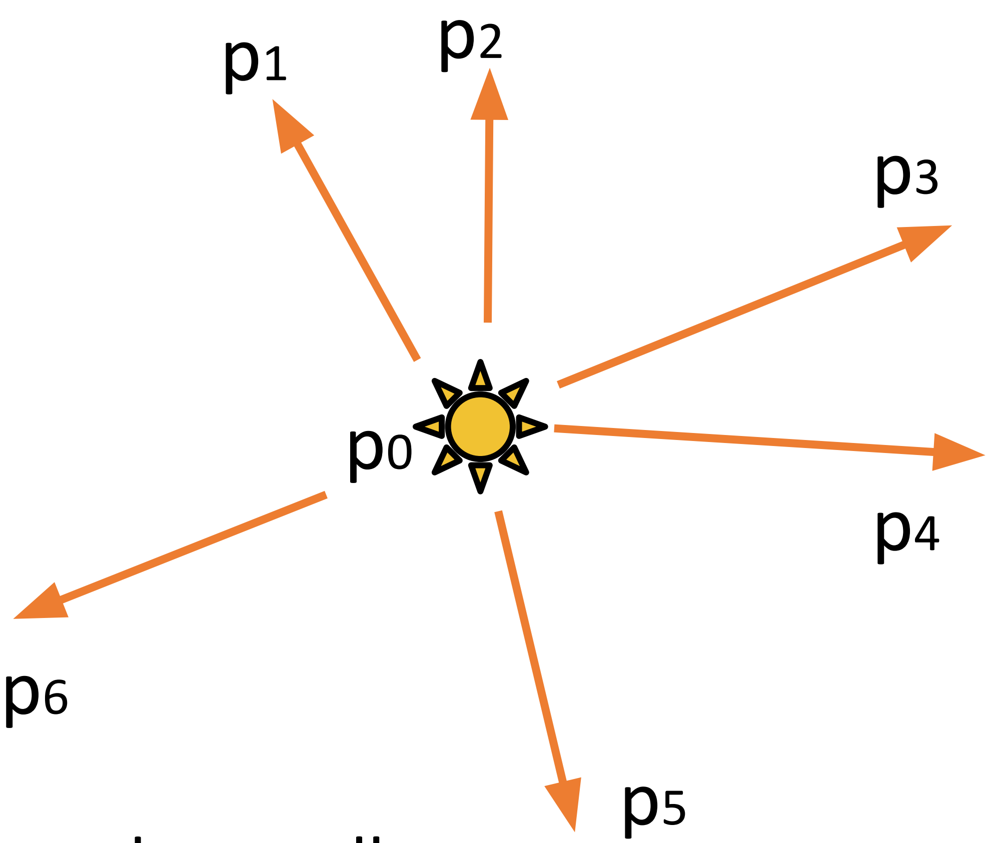
  $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 4\pi p_0$

The light source emitted *S* photons.

A **photon's flux** is $p_i = 4\pi p_0 / S$

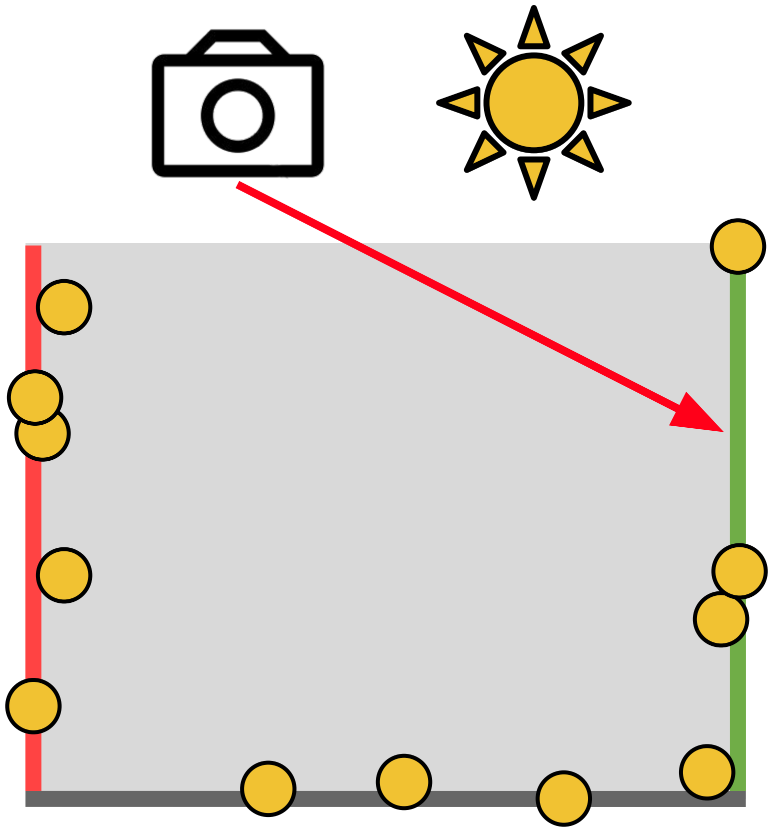
Careful with the photon map's size limit

You might need to update S after doing the random walks



Recap: how the photon map is used

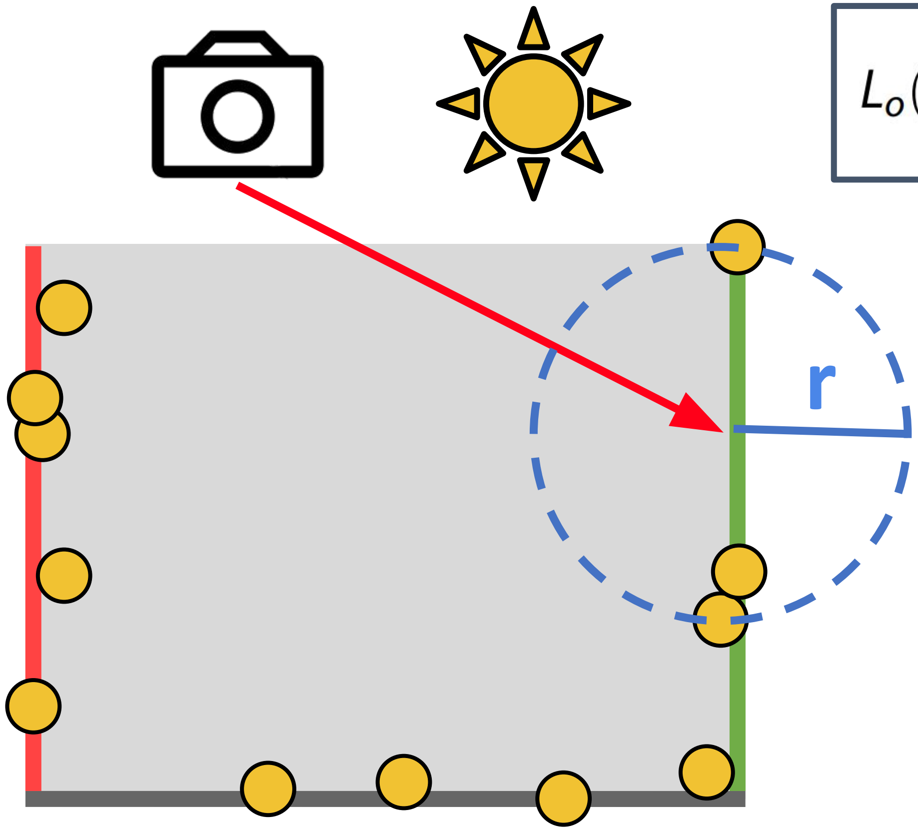
- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map



$$L_o(\mathbf{x},\omega_\mathbf{o}) = L_e(\mathbf{x},\omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x},\omega_\mathbf{i}) f_r(\mathbf{x},\omega_\mathbf{i},\omega_\mathbf{o}) |\mathbf{n}\cdot\omega_\mathbf{i}| d\omega_\mathbf{i}}$$

Recap: how the photon map is used

- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map

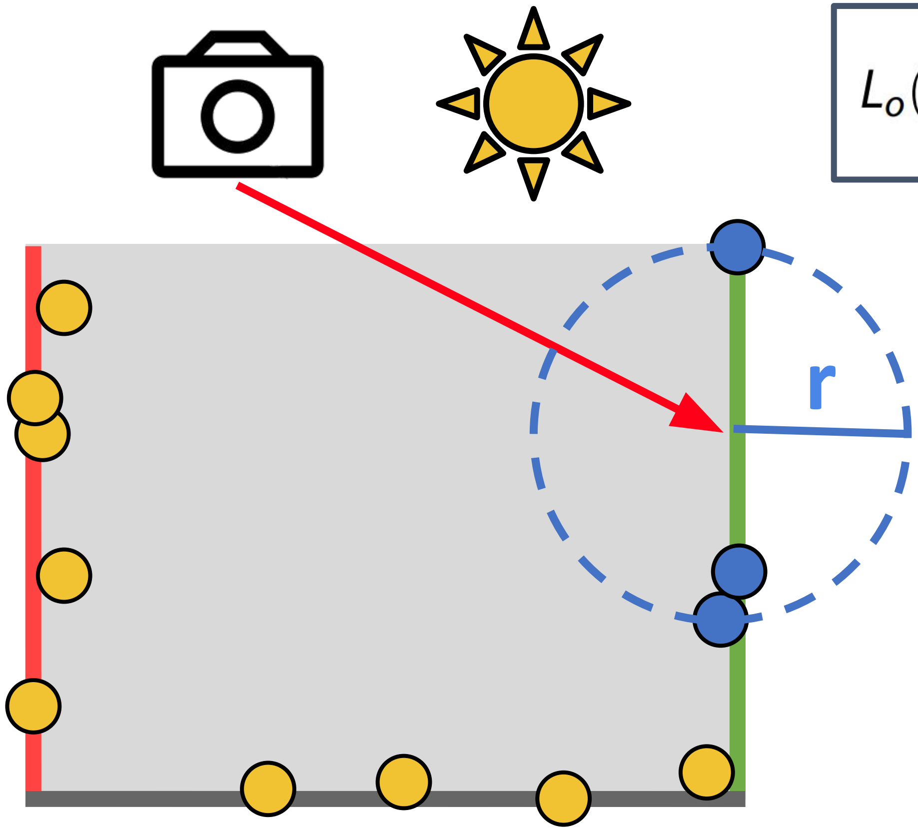


$$L_o(\mathbf{x},\omega_\mathbf{o}) = L_e(\mathbf{x},\omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x},\omega_\mathbf{i}) f_r(\mathbf{x},\omega_\mathbf{i},\omega_\mathbf{o}) |\mathbf{n}\cdot\omega_\mathbf{i}| d\omega_\mathbf{i}}$$

- Idea: use **k nearby** photons (distance < r) to approximate the Render Equation

Recap: how the photon map is used

- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map



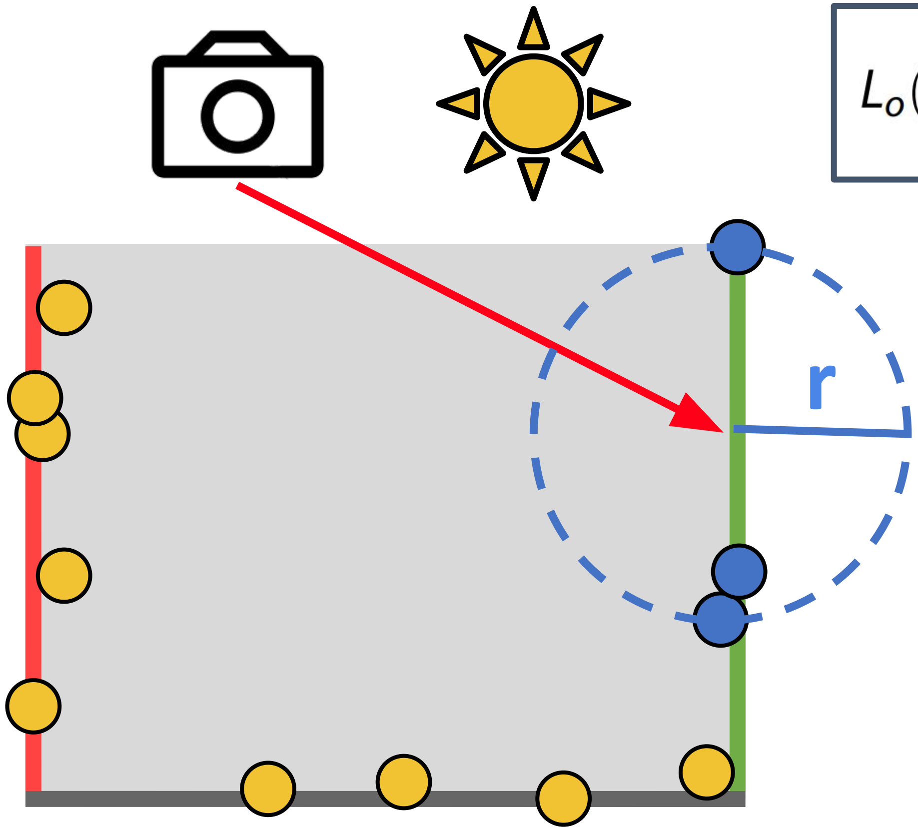
$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

- Idea: use **k nearby** photons (distance < r) to approximate the Render Equation

Recap: how the photon map is used

- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

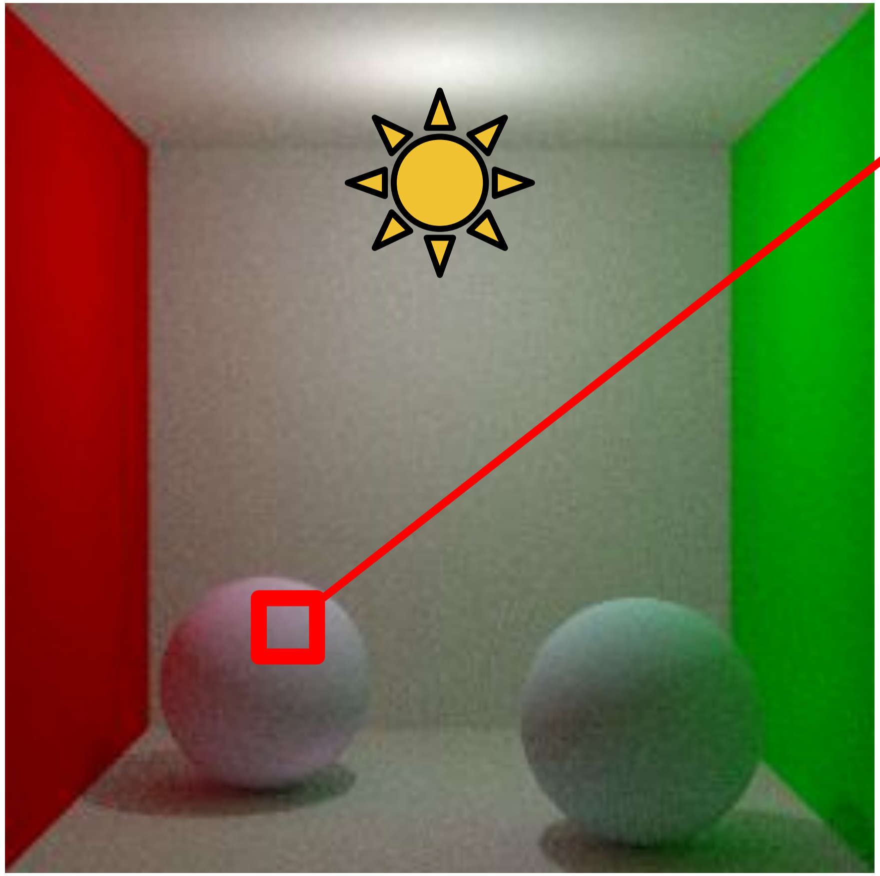


- Idea: use **k nearby** photons (distance < r) to approximate the Render Equation
- Integral is approximated as the **sum of k = 3 contributions**

Which color do we fill each pixel with?

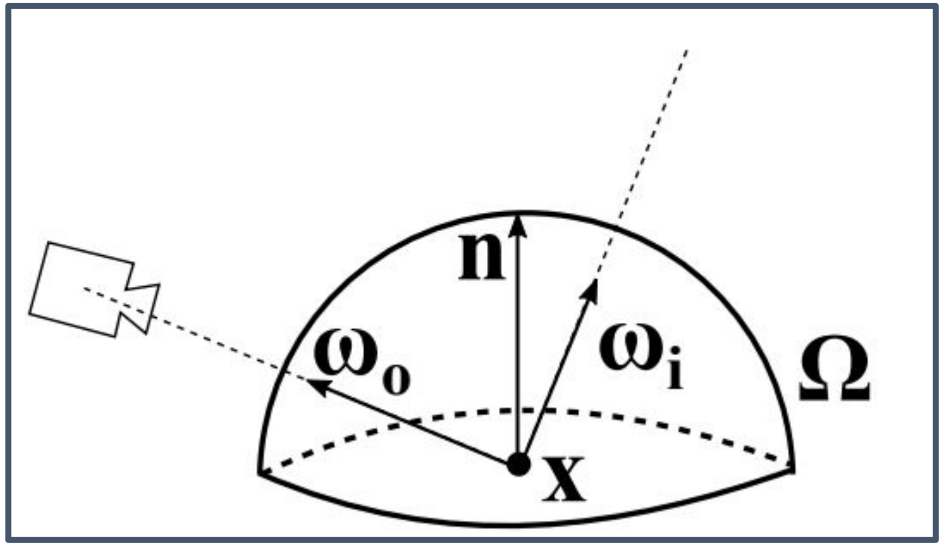
Graphics and Imaging Lab

- For path tracing:



$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

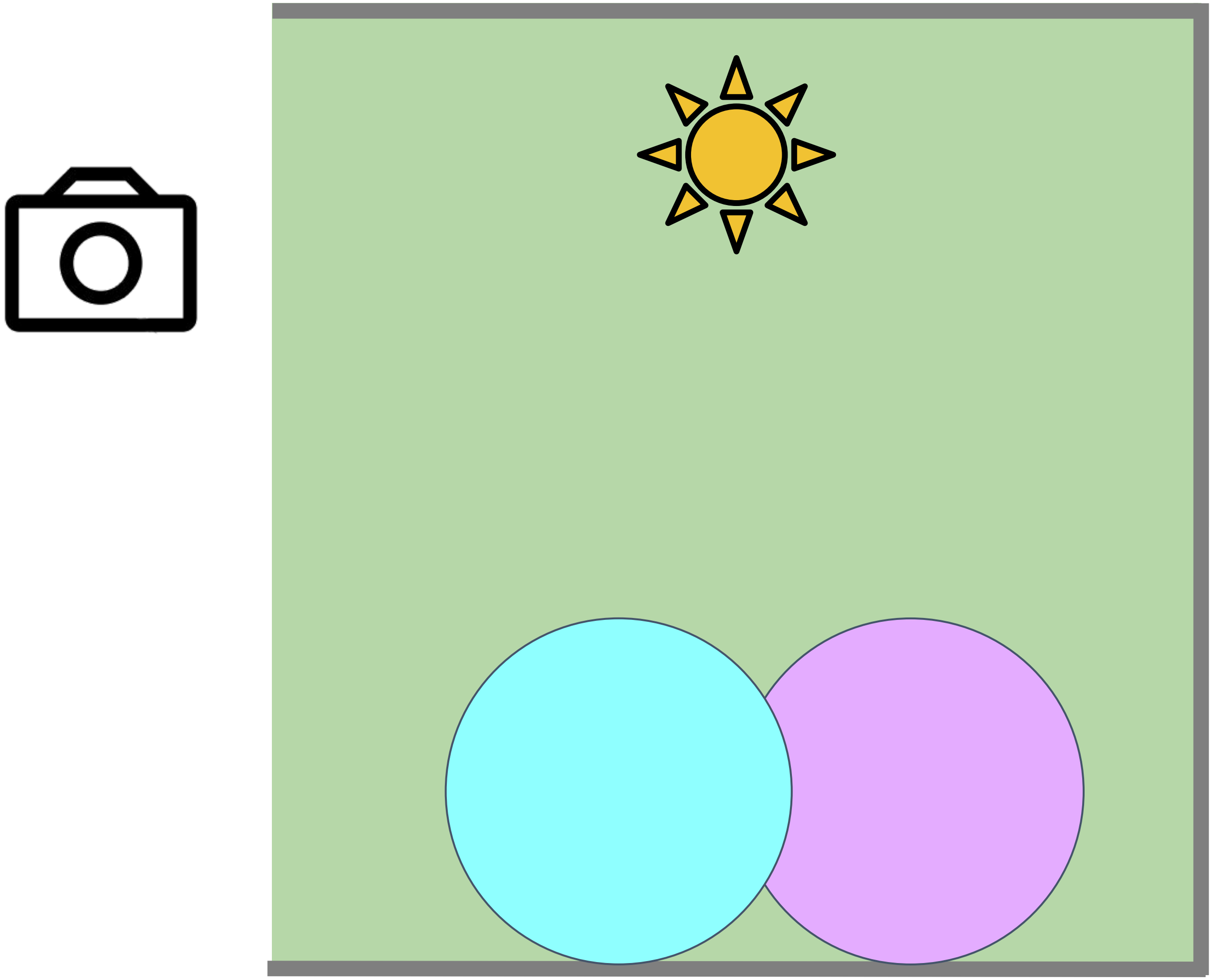
The full integral



Which color do we fill each pixel with?

Graphics and Imaging Lab

- For photon mapping:

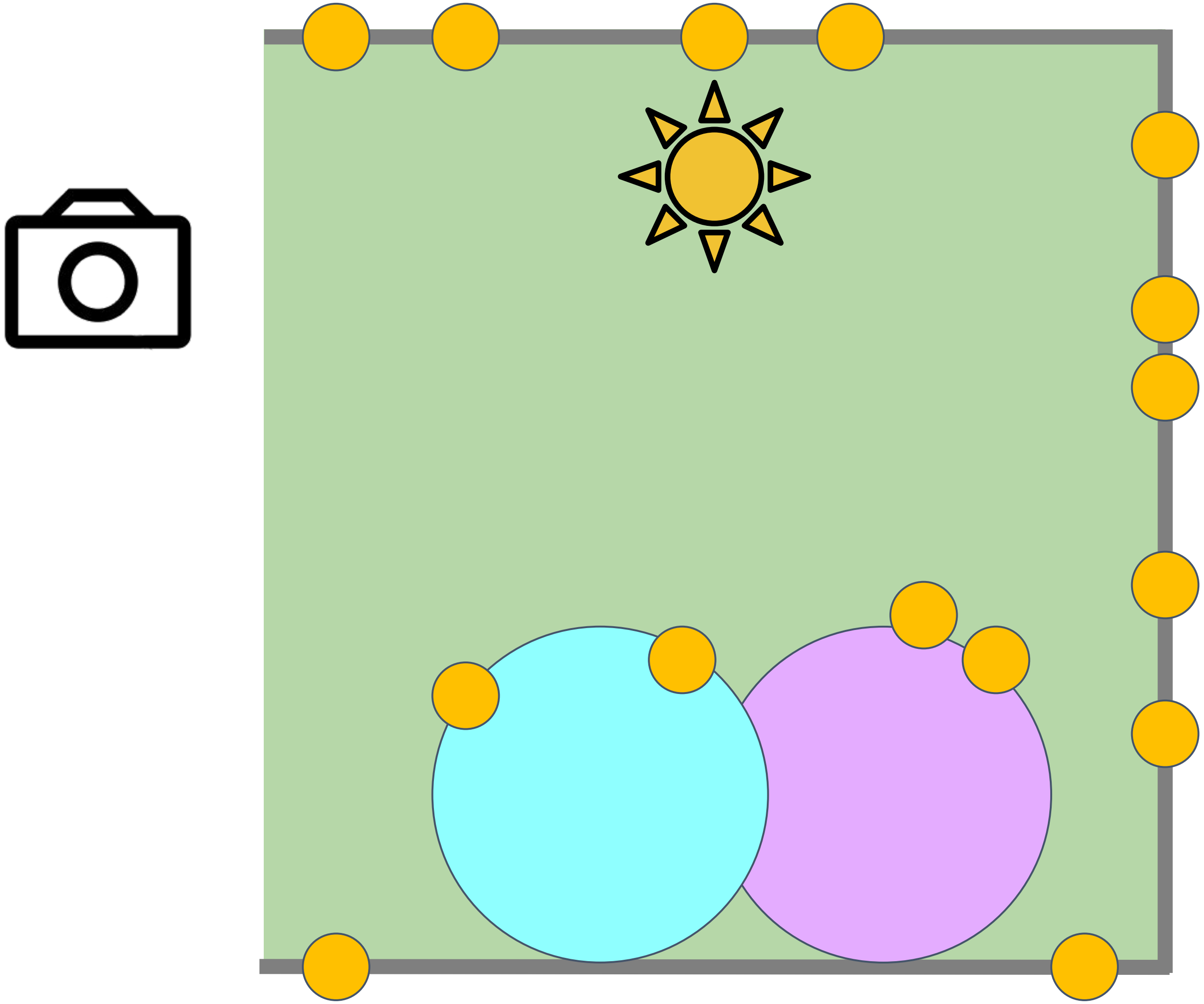


Which color do we fill each pixel with?

- For photon mapping:

1) Calculate the photon map

(previous session)

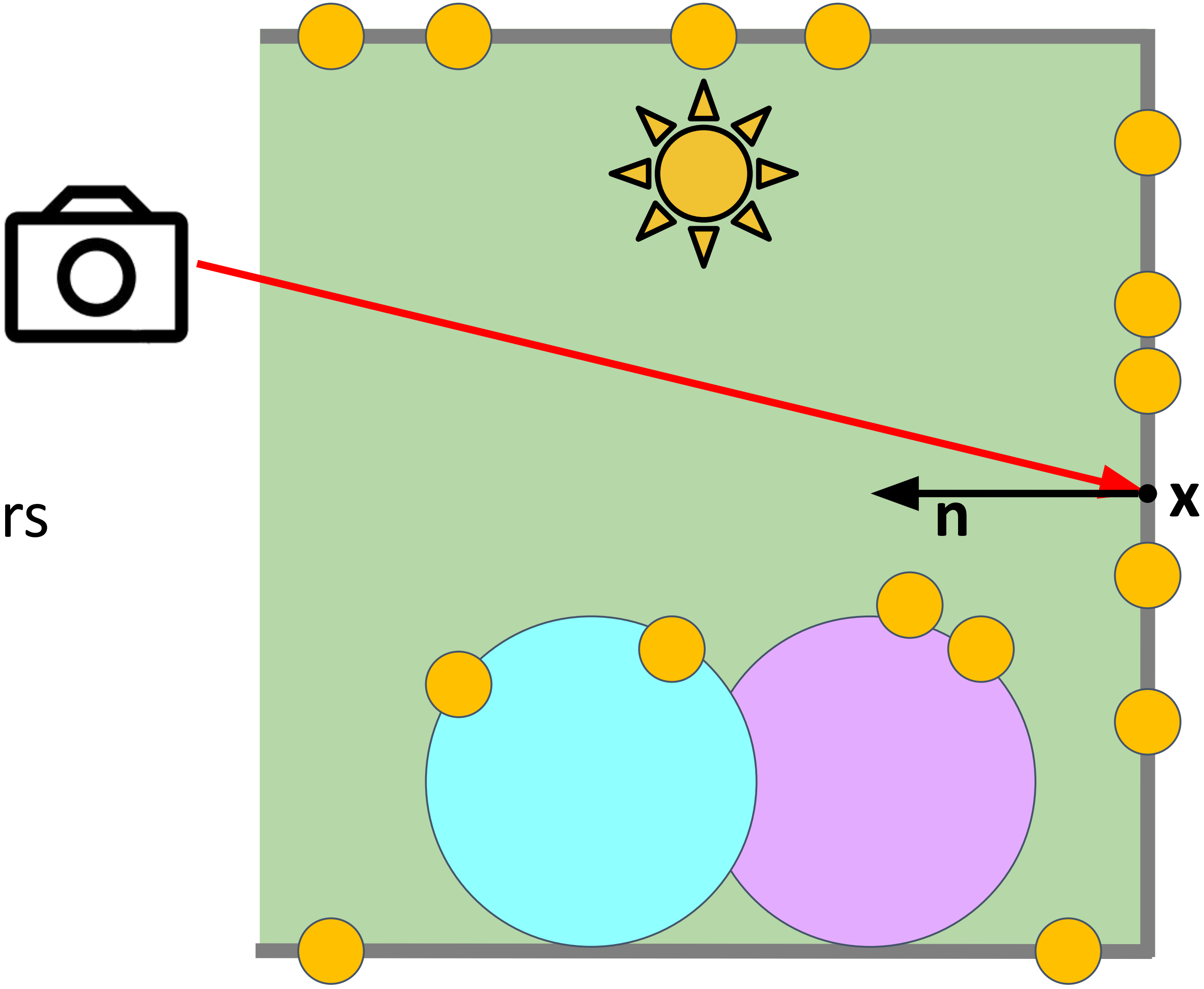


Which color do we fill each pixel with?

- For photon mapping:

2) Launch rays from the camera (**same as path tracing**)

- Intersect at point **x**
- Search for nearest neighbours of **x** in the photon map:



Which color do we fill each pixel with?

Graphics and Imaging Lab

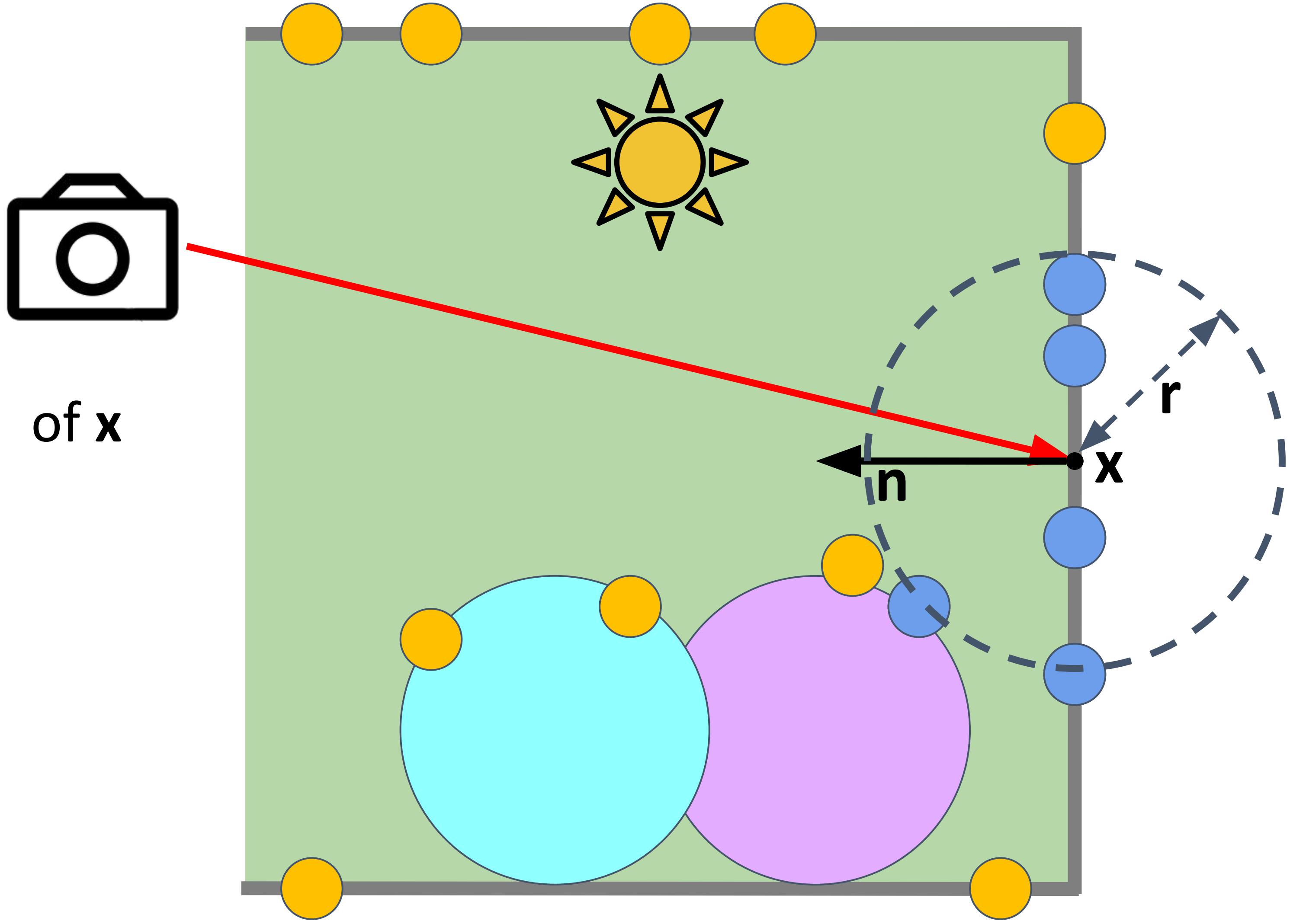
- For photon mapping:

2) Launch rays from the camera (**same as path tracing**)

- Intersect at point **x**
- Search for nearest neighbours of **x** in the photon map:

a) **k** nearest photons

b) photons at distance <= **r**



Which color do we fill each pixel with?

Graphics and Imaging Lab

- For photon mapping:

2) Launch rays from the camera (**same as path tracing**)

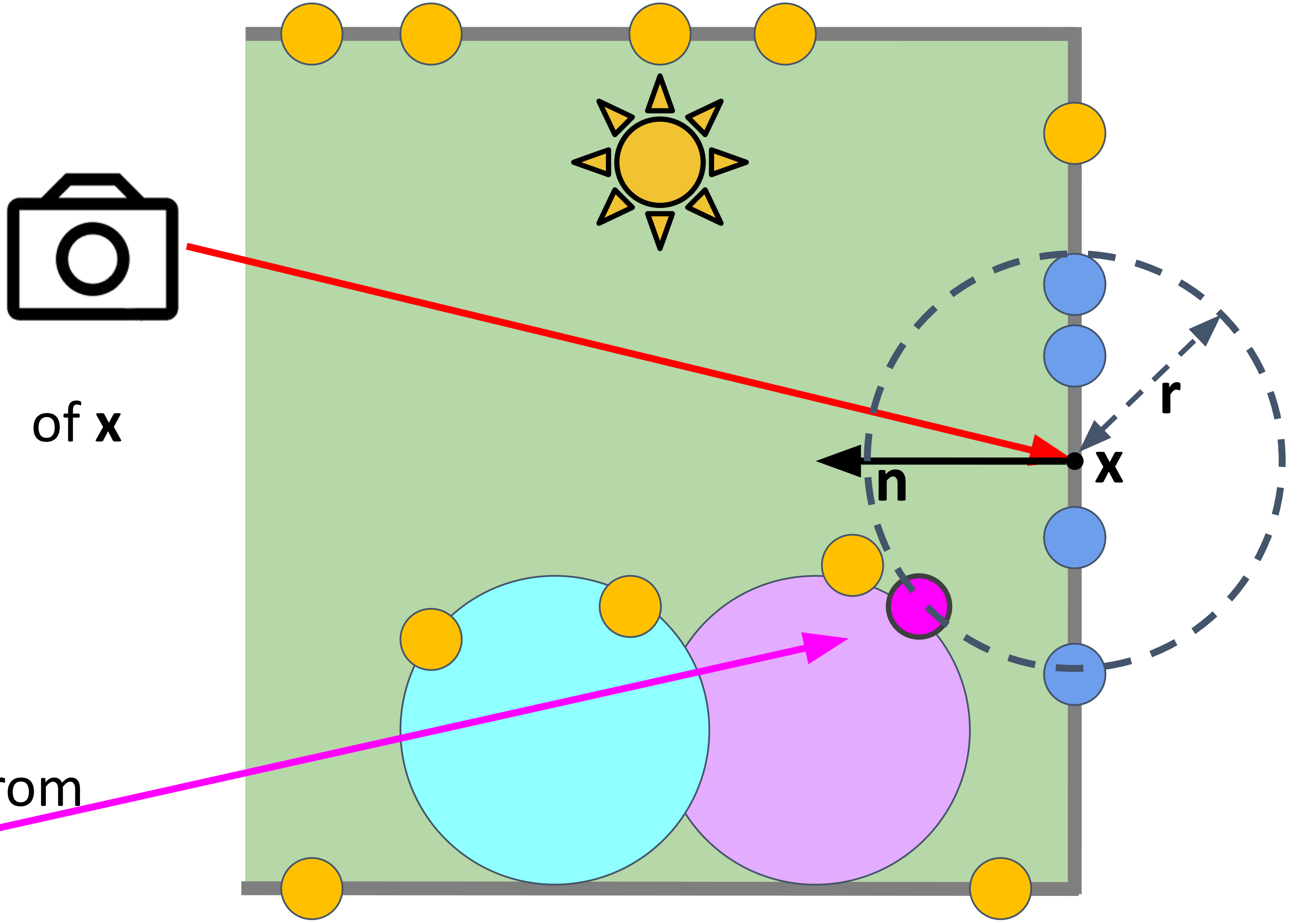
- Intersect at point **x**
- Search for nearest neighbours of **x** in the photon map:

  a) **k** nearest photons

  b) photons at distance <= **r**

- Note how some photons are from **other surfaces**

(you can try to solve this, or not)



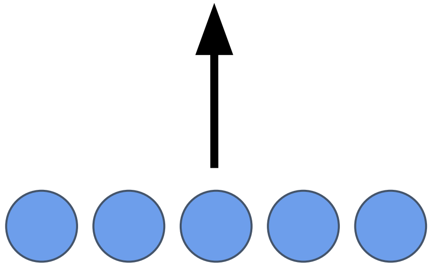
Which color do we fill each pixel with?

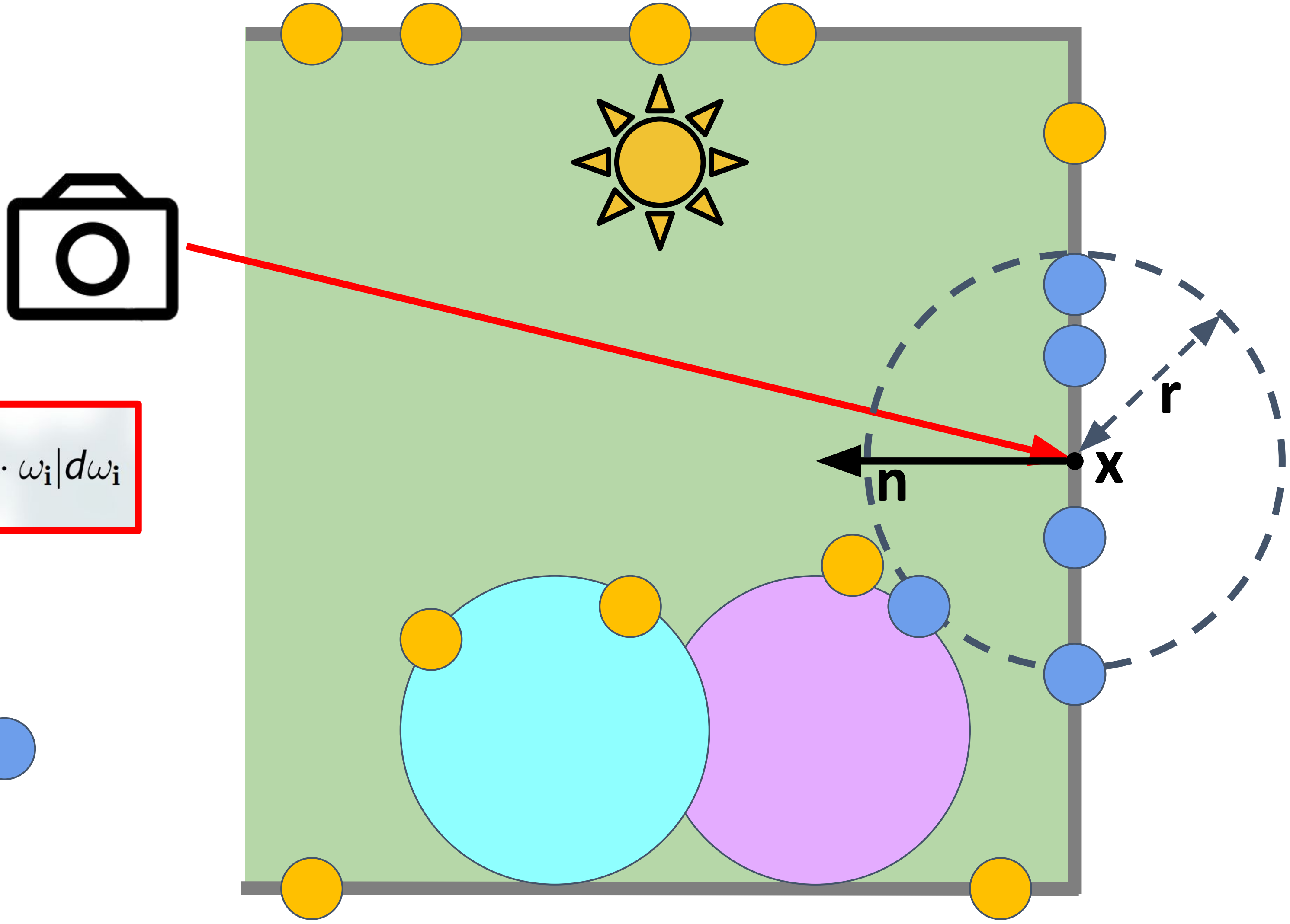
- For photon mapping:

3) Estimate color of the pixel

with nearby photons

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$



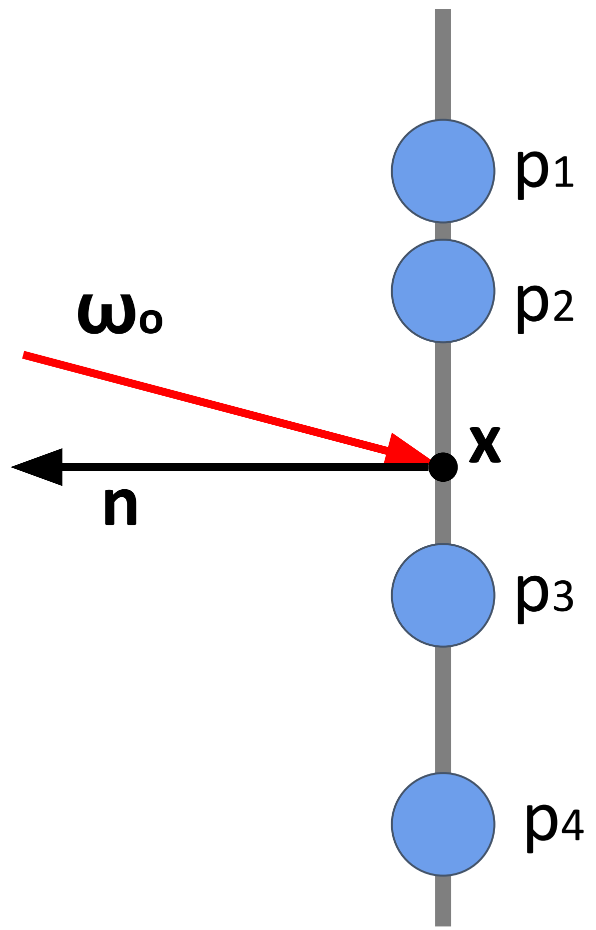


Kernel density estimation

$$\boxed{L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}}$$

Each photon p contains:

- Φ_p flux
- $\mathbf{x}_p$ position
- $\boldsymbol{\omega}_p$ direction



Kernel density estimation

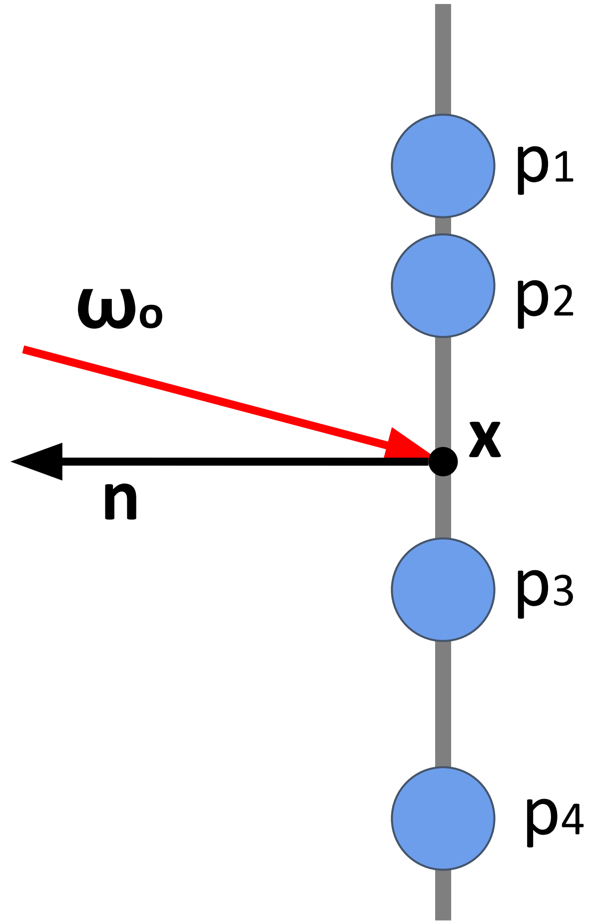
$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

Each photon p contains:

- Φ_p flux
- $\mathbf{x}_p$ position
- $\boldsymbol{\omega}_p$ direction

Constant density estimation: box kernel

$$\boxed{L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \omega_\mathbf{p}, \omega_\mathbf{o}) \frac{\Phi_p}{\pi r_k^2}}$$



Kernel density estimation

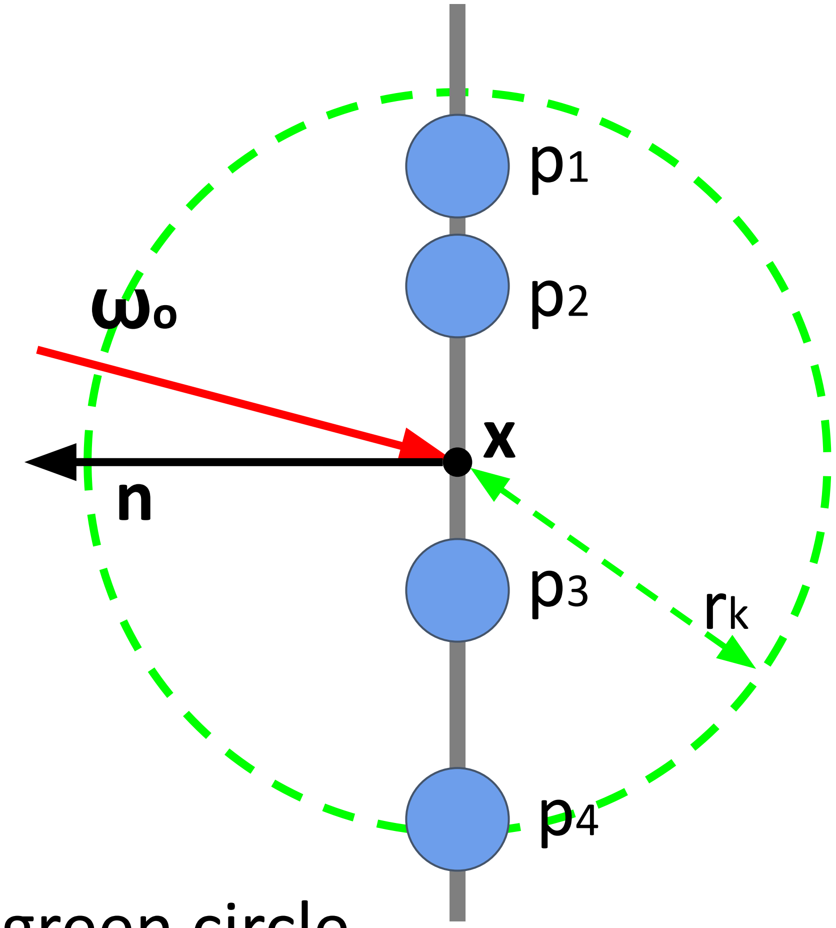
$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

Each photon p contains:

- Φ_p flux
- $\mathbf{x}_p$ position
- $\boldsymbol{\omega}_p$ direction

Constant density estimation: box kernel

$$\boxed{L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \omega_\mathbf{p}, \omega_\mathbf{o}) \frac{\Phi_p}{\boxed{\pi r_k^2}}}$$



area of the green circle

r_k is the radius that contains all photons

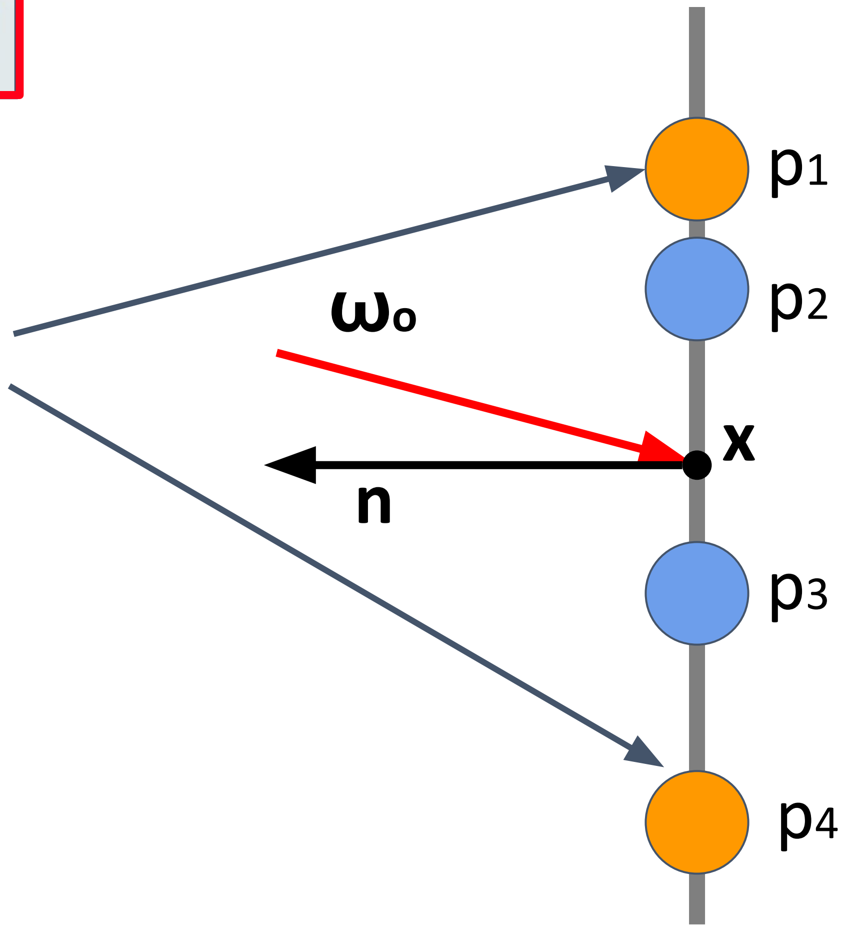
Kernel density estimation

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

Each photon p contains:

- Φ_p flux
- $\mathbf{x}_p$ position
- ω_p direction

Far away from the intersection point



Constant density estimation: box kernel

$$\boxed{L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \omega_\mathbf{p}, \omega_\mathbf{o}) \frac{\Phi_p}{\pi r_k^2}}$$

Kernel density estimation

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

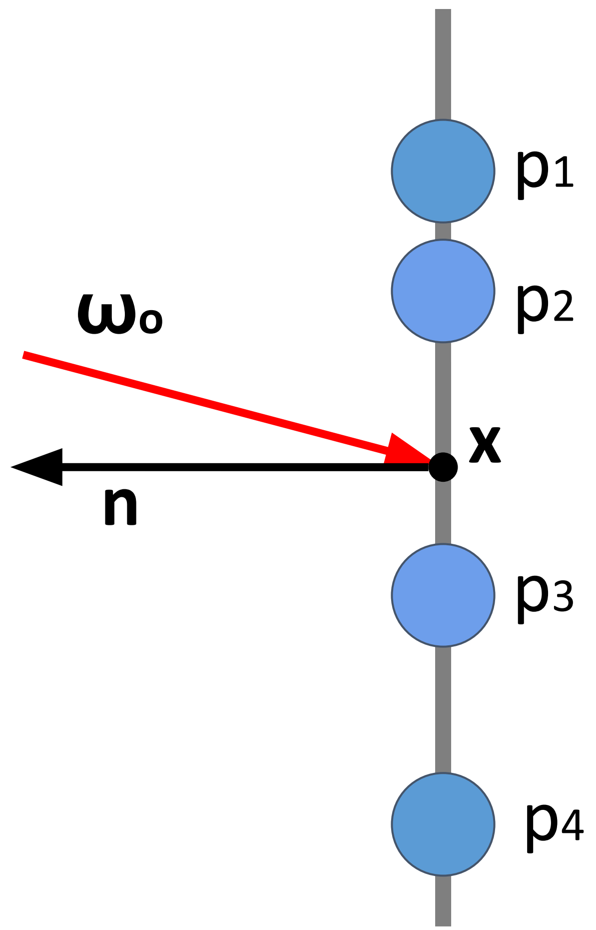
Each photon p contains:

- Φ_p flux
- $\mathbf{x}_p$ position
- $\boldsymbol{\omega}_p$ direction

Non-constant density estimation (optional)

Gives more weight to photons closer to **x**

$$\boxed{L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \omega_\mathbf{p}, \omega_\mathbf{o}) \Phi_p \underline{K_{2D}(|\mathbf{x} - \mathbf{x_p}|, r_k)}}$$



Kernel density estimation

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

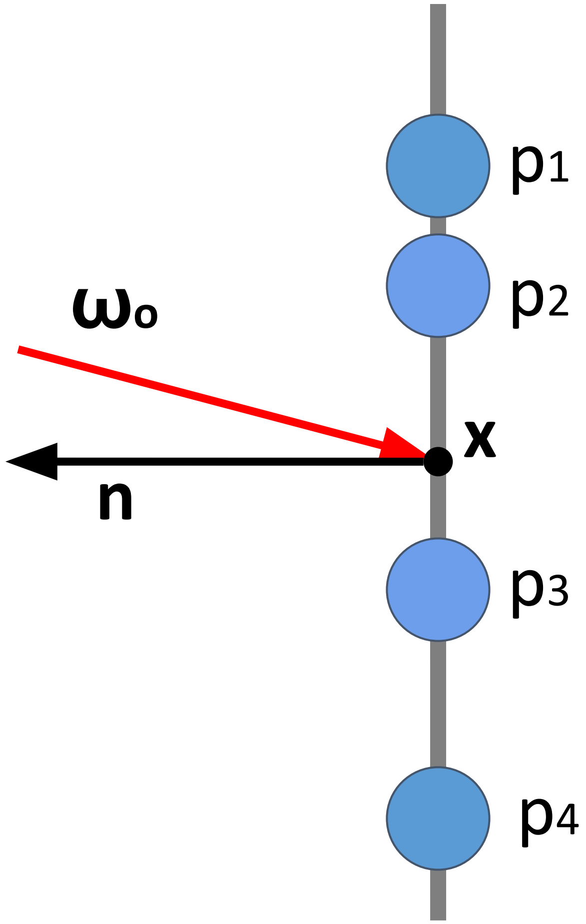
Each photon p contains:

- Φ_p flux
- $\mathbf{x}_p$ position
- $\boldsymbol{\omega}_p$ direction

Non-constant density estimation (optional)

Gives more weight to photons closer to **x**

$$\boxed{L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \sum_{p=1}^{k} f_r(\mathbf{x}, \omega_\mathbf{p}, \omega_\mathbf{o}) \Phi_p K_{2D}(\boxed{|\mathbf{x} - \mathbf{x_p}|}, r_k)}$$

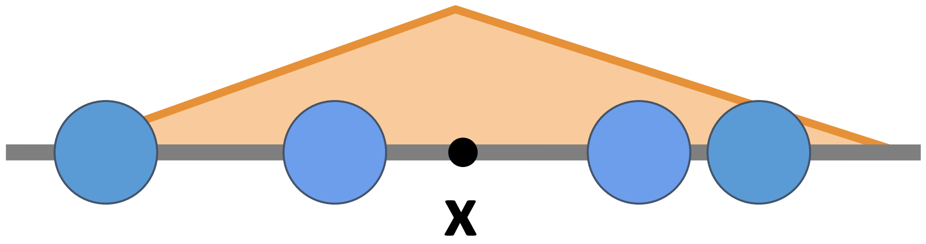


Kernel density estimation

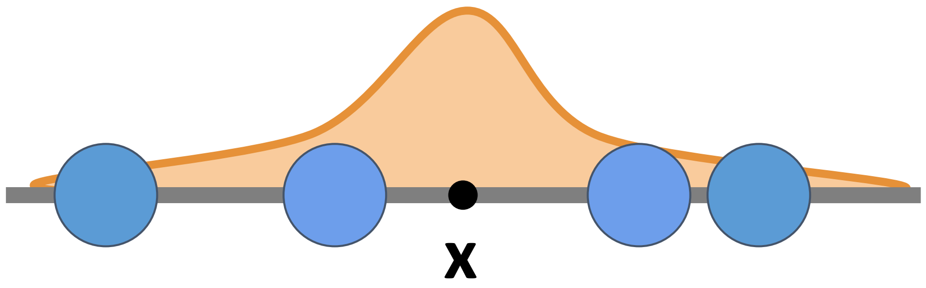
Graphics and Imaging Lab

Non-constant density estimation

Cone kernel

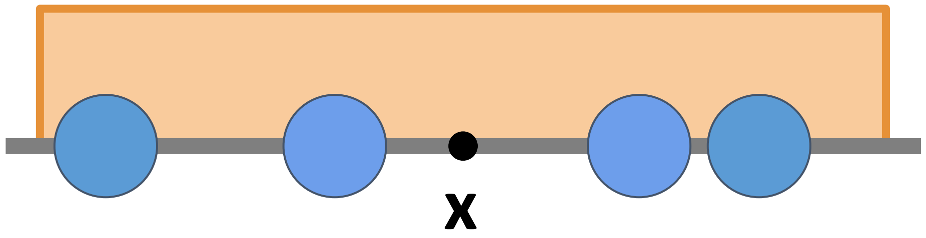


Gaussian kernel

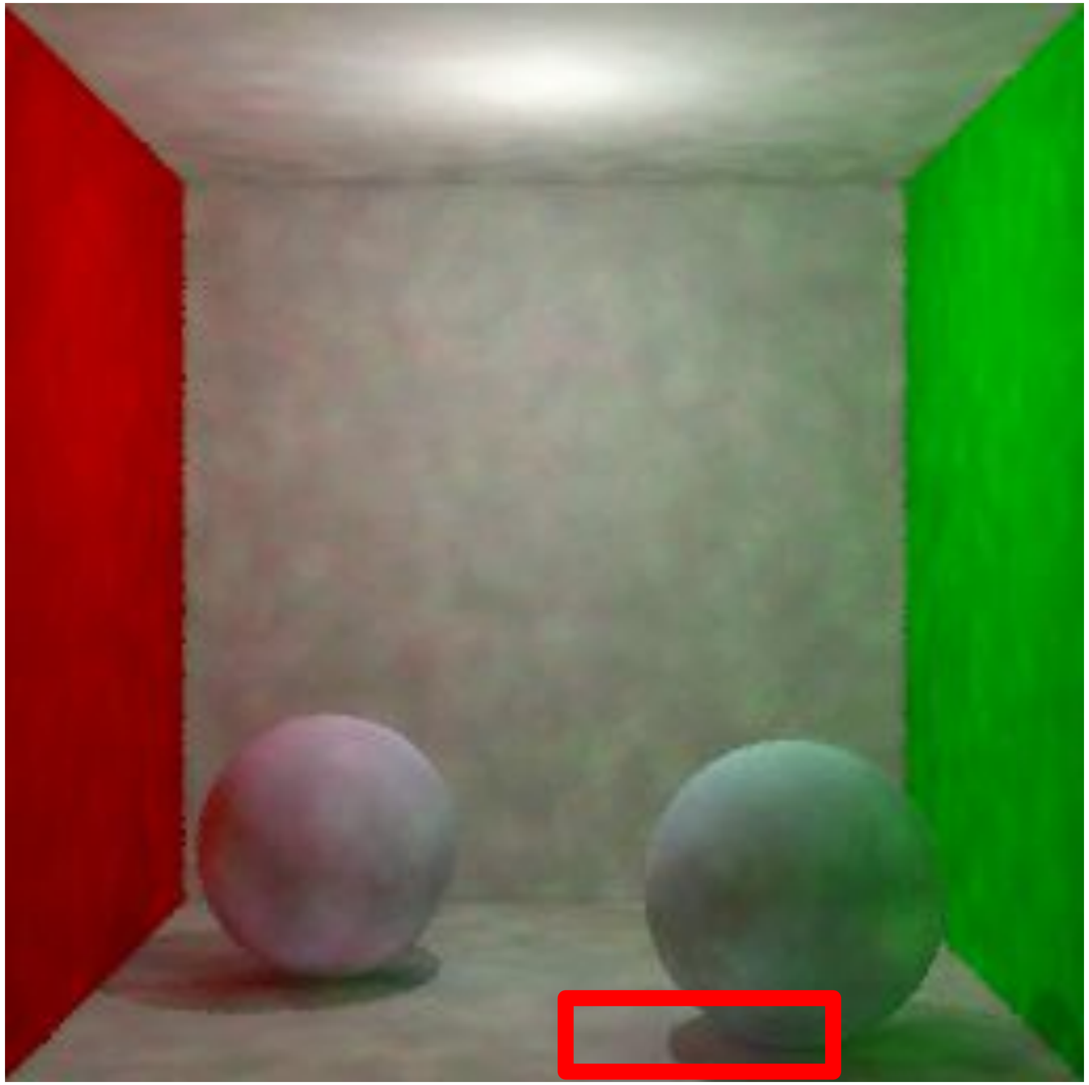


Constant density estimation

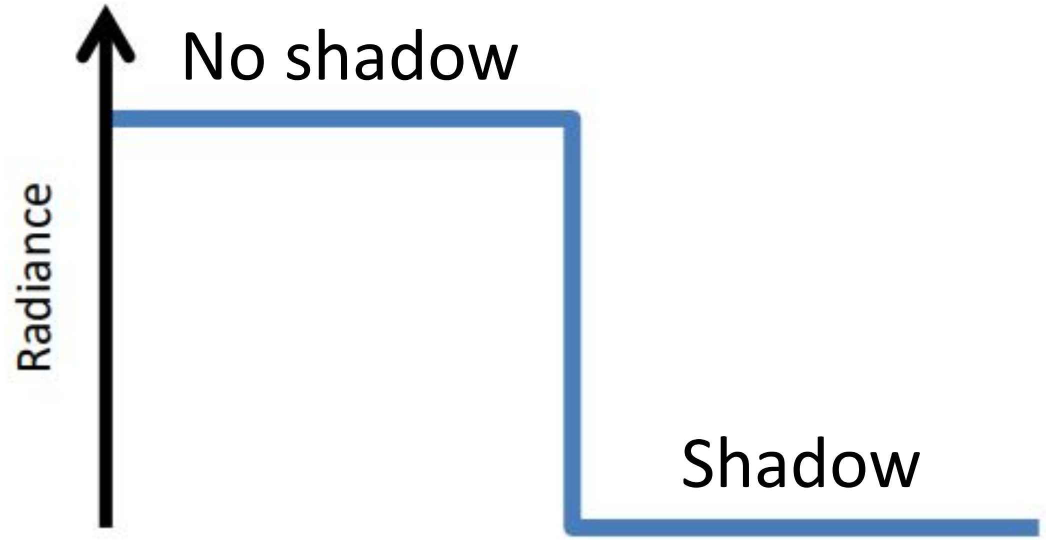
Box kernel

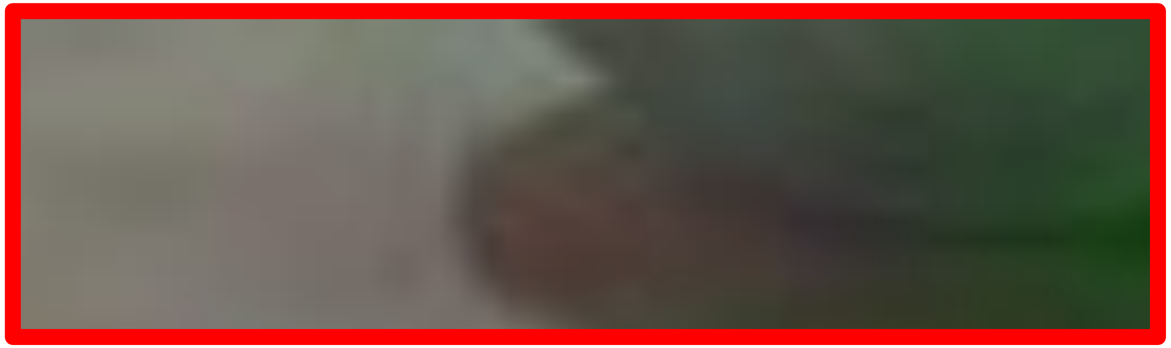


Photon mapping is biased



Photon mapping is biased

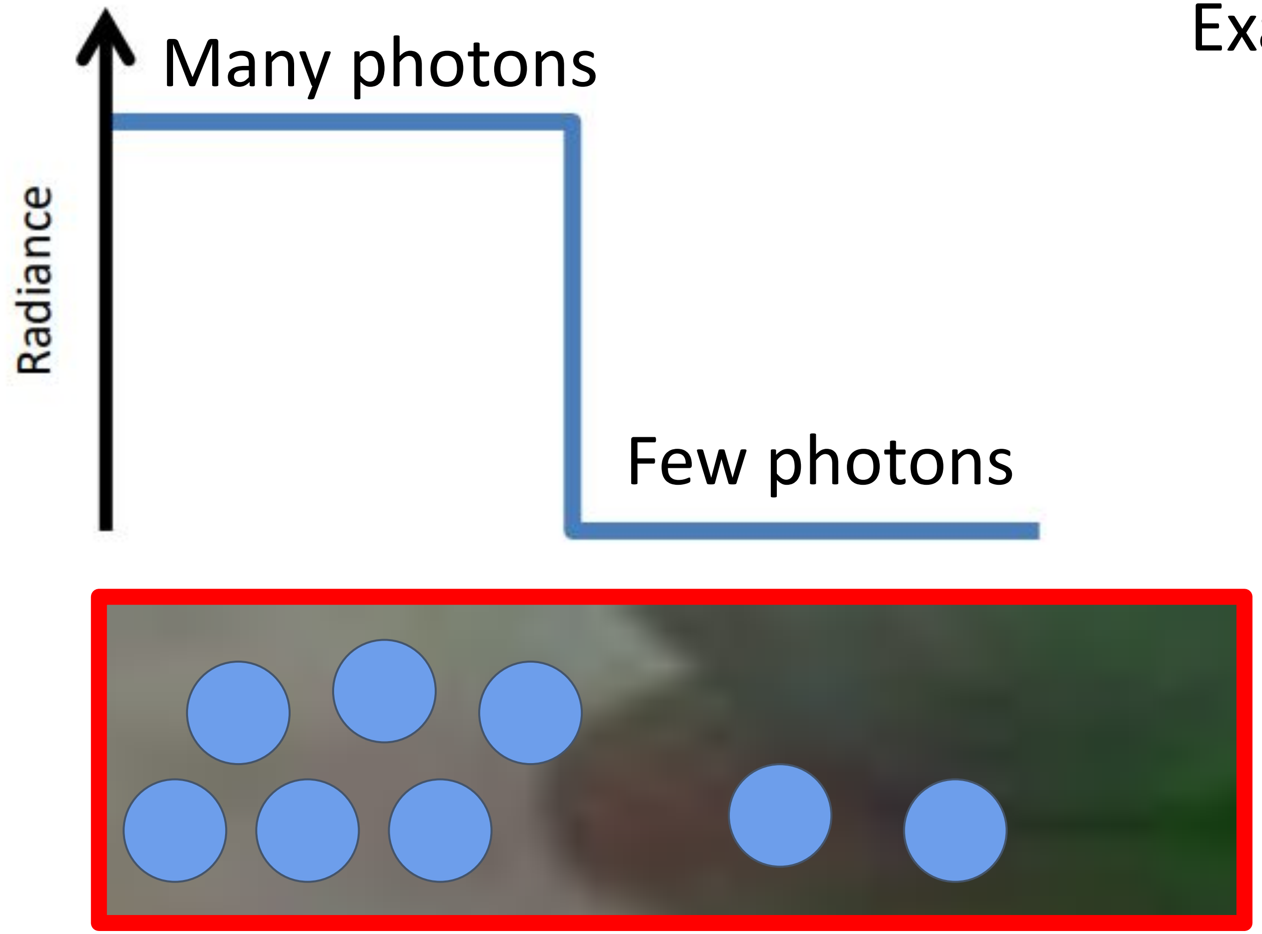




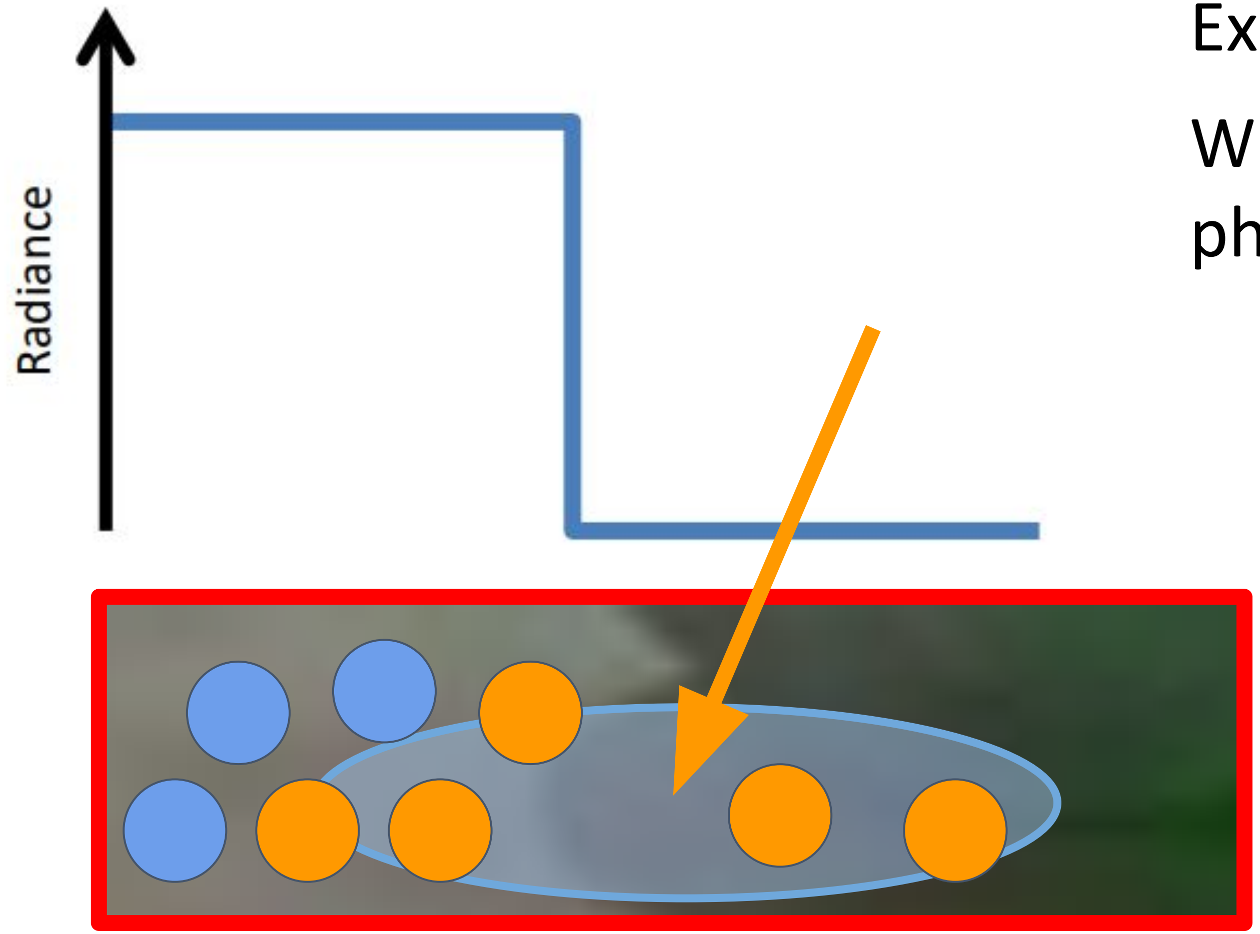
Example: area with hard shadows

Photon mapping is biased

Example: area with hard shadows



Photon mapping is biased

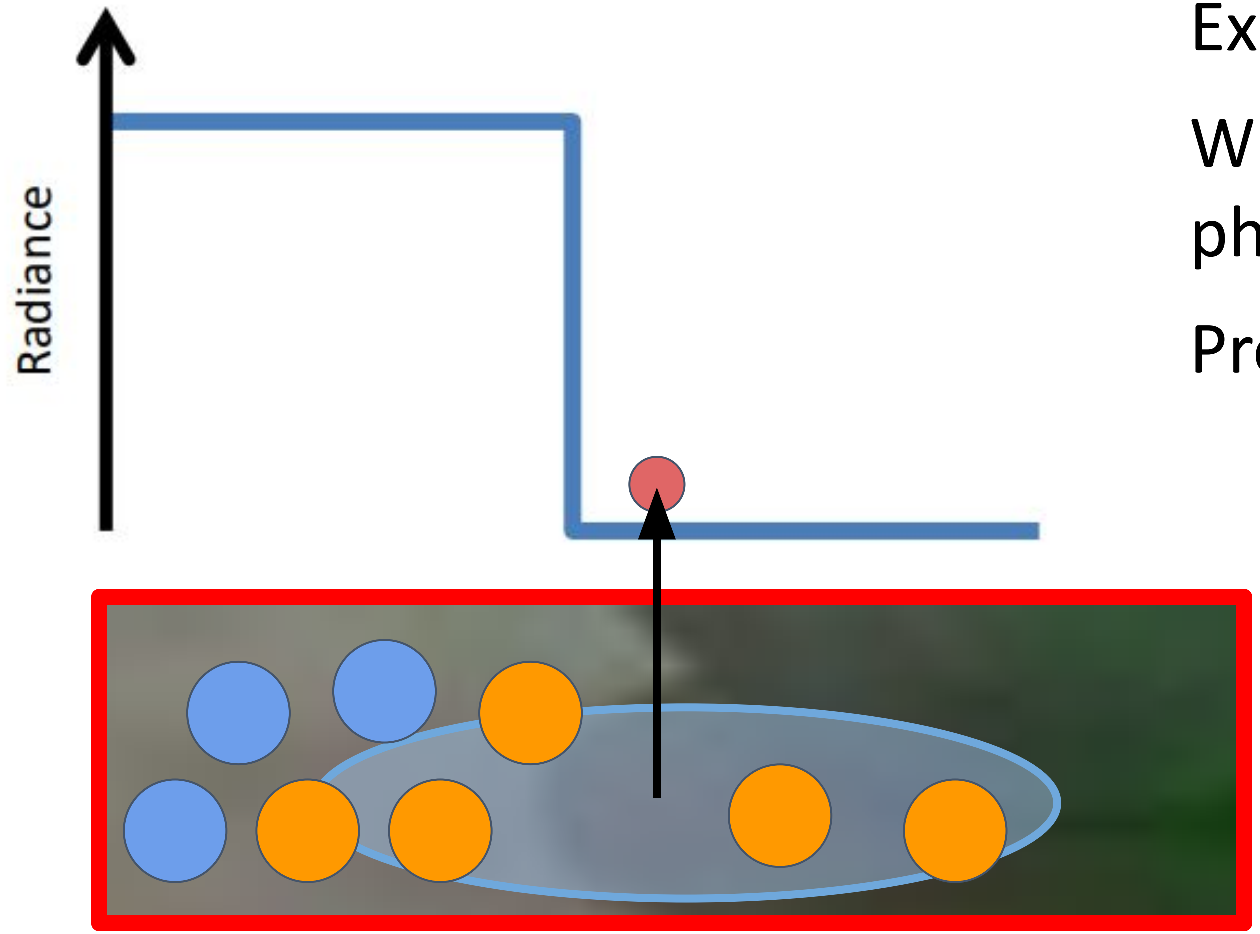


Example: area with hard shadows

When a ray intersects, it picks up photons from both areas

Photon mapping is biased

Graphics and Imaging Lab

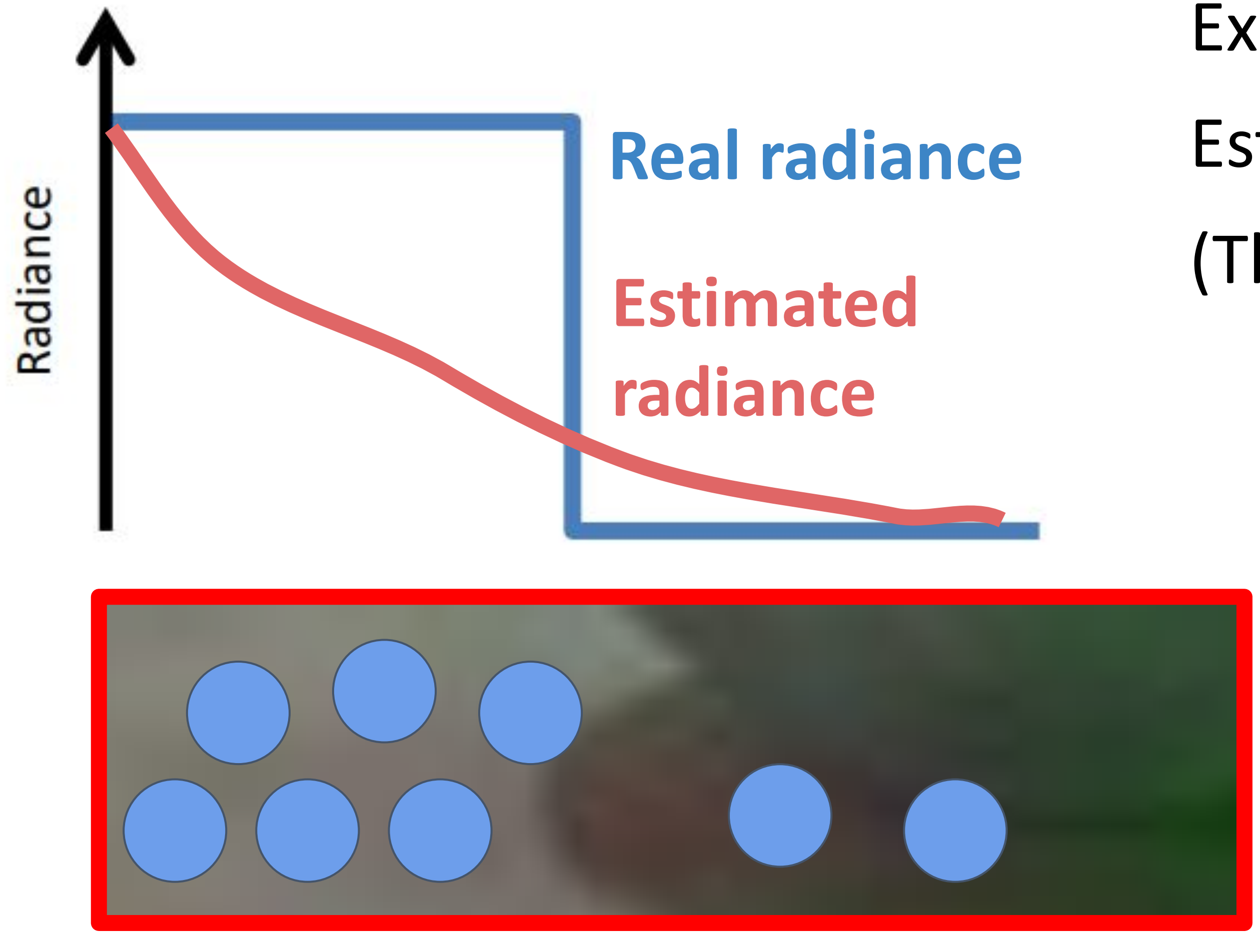


Example: area with hard shadows

When a ray intersects, it picks up photons from both areas

Produces a **biased estimation**

Photon mapping is biased



Example: area with hard shadows

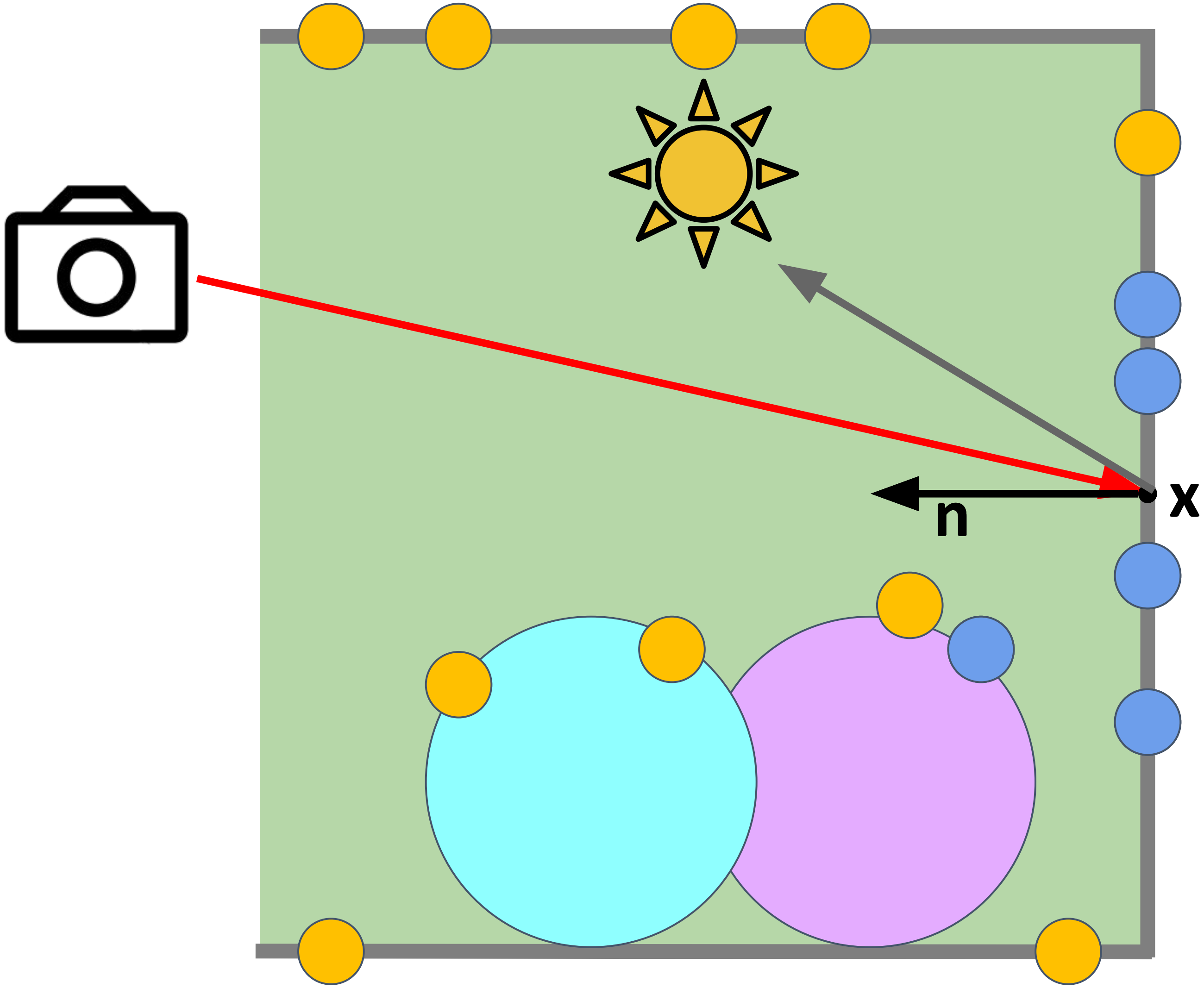
Estimated result is **biased**

(This did not happen for path tracing)

How to compute direct light

We can compute direct light

without using a photon map:



How to compute direct light

We can compute direct light

without using a photon map:

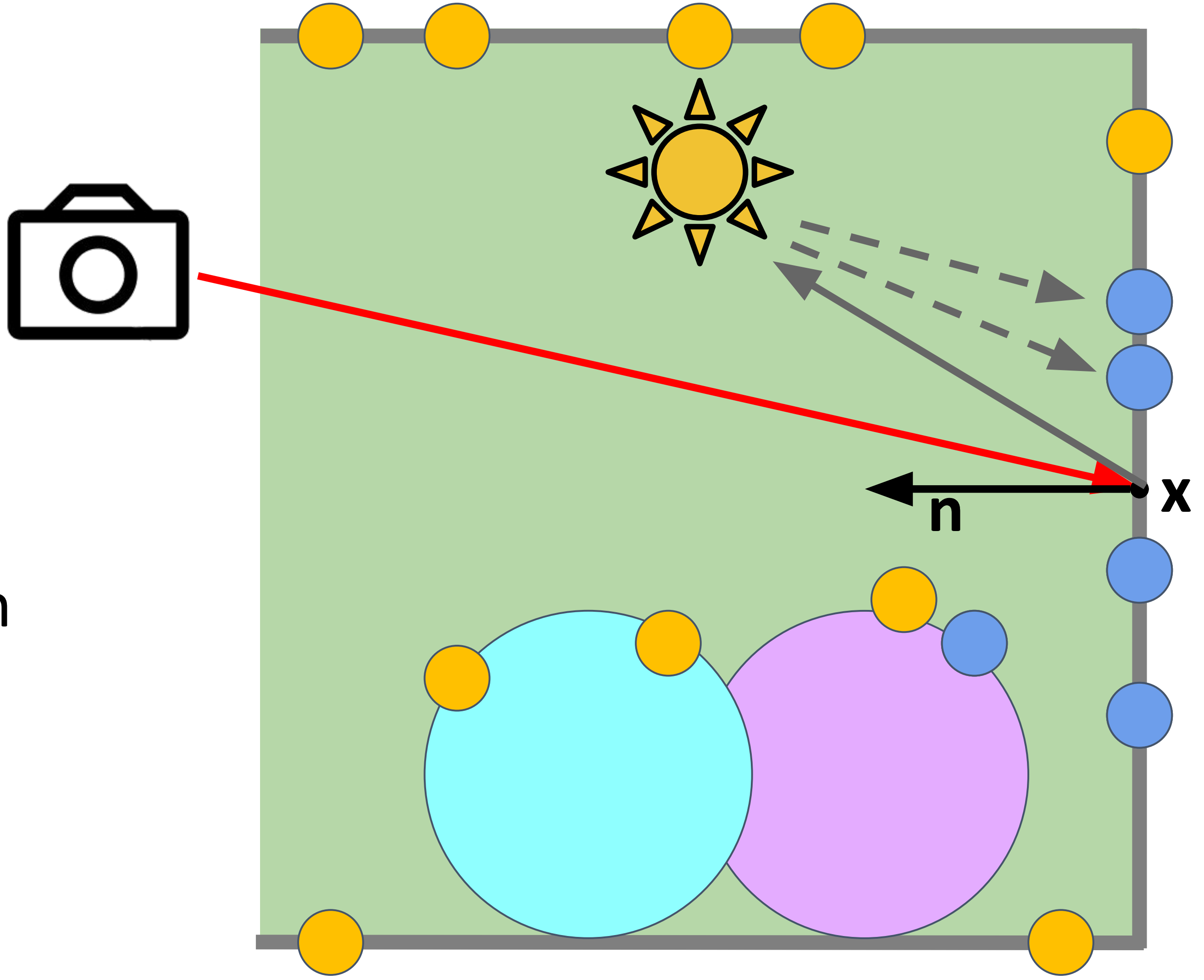
However

You are duplicating energy

+

Next-event estimation

First bounce photons



How to compute direct light

We can compute direct light
without using a photon map:

However

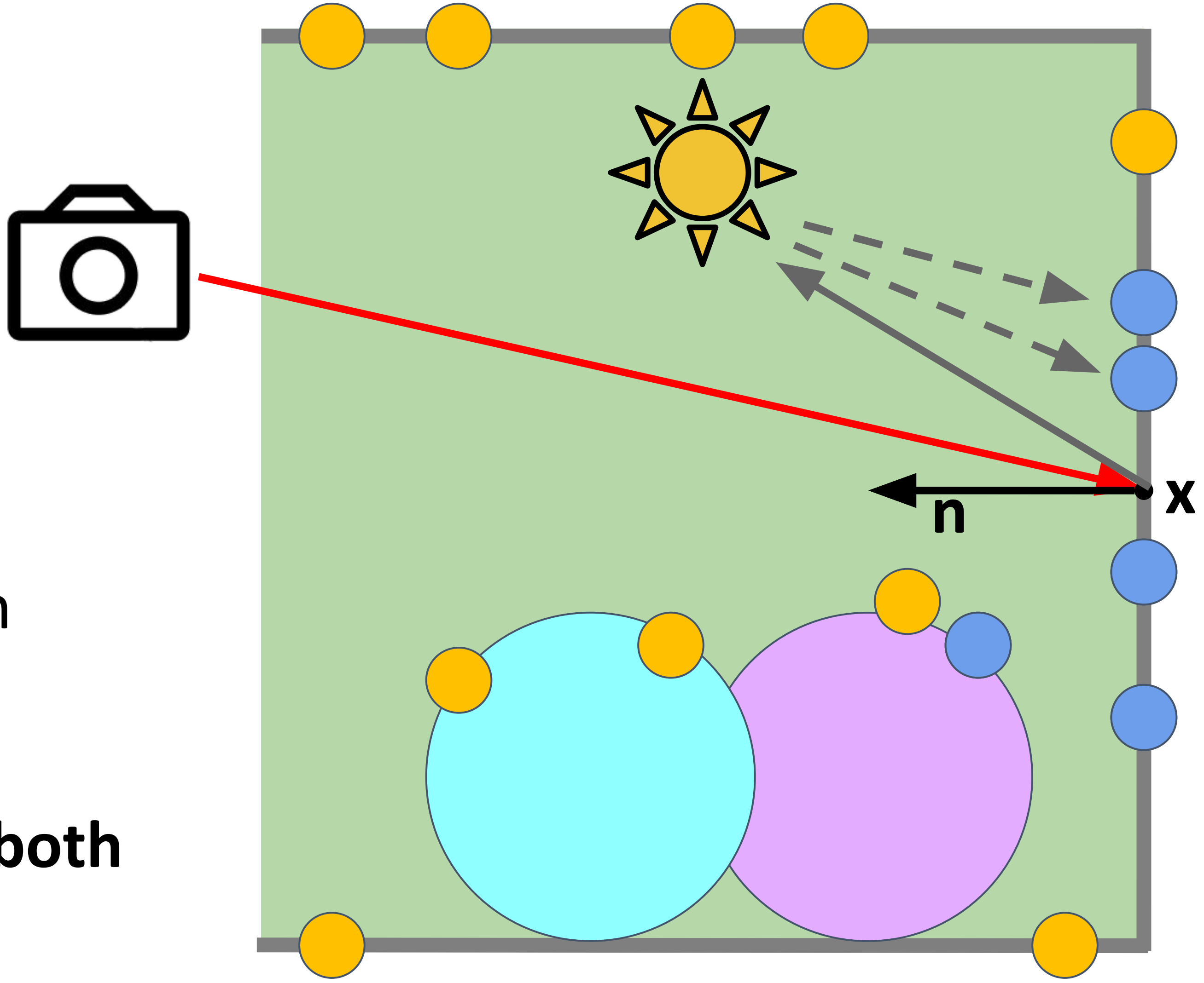
You are duplicating energy

+

Next-event estimation

First bounce photons

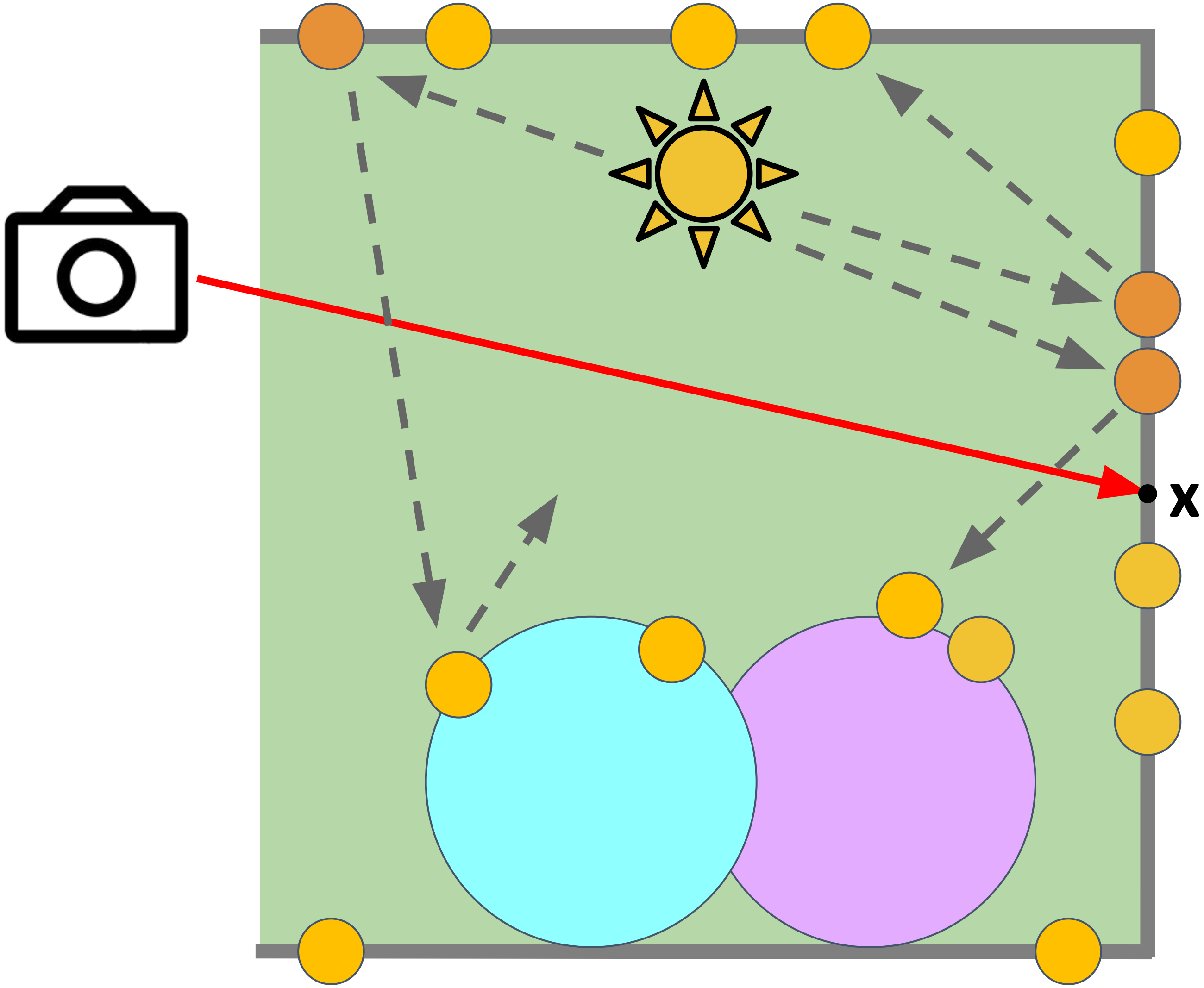
Only choose one method, not both



How to compute direct light

Method 1:

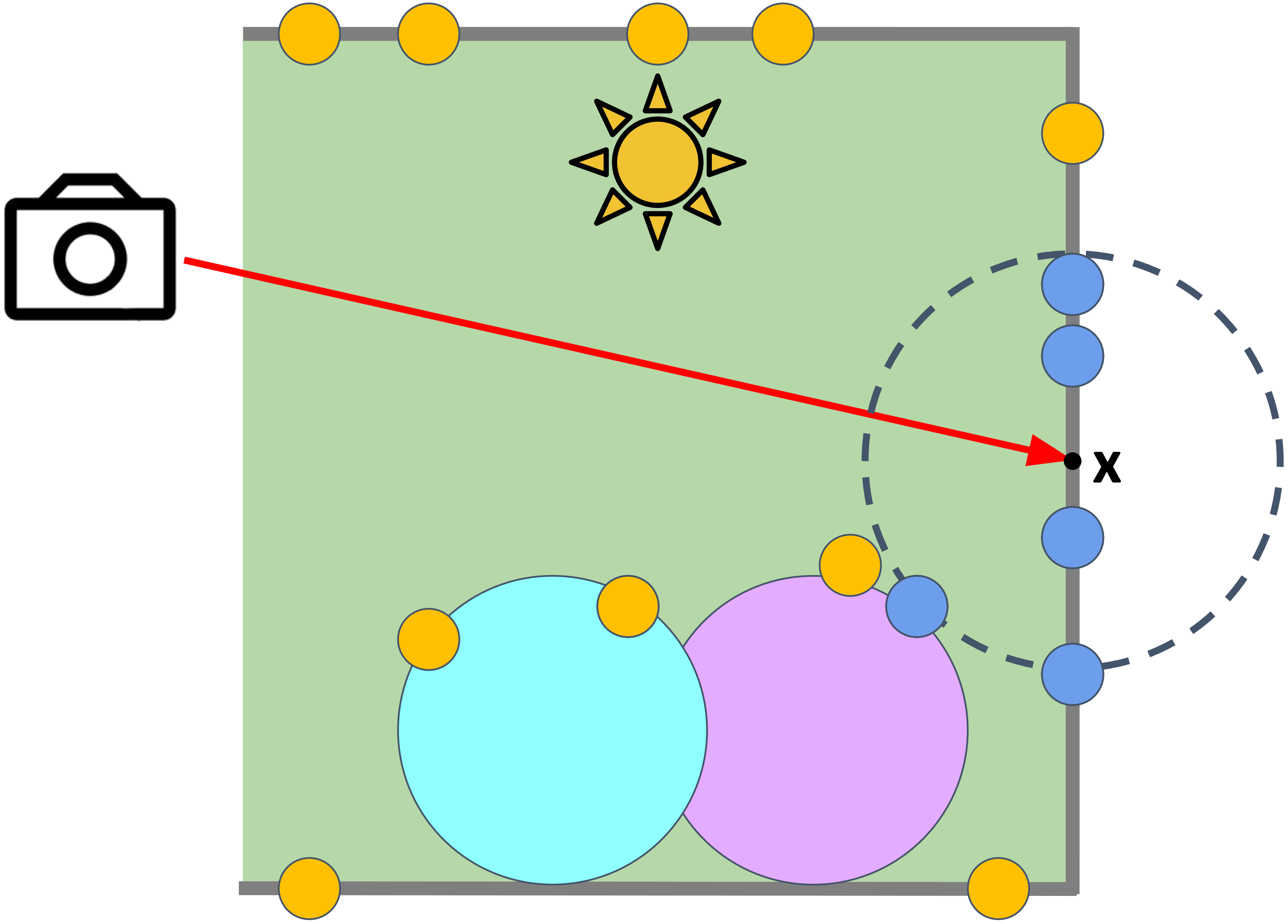
- When generating the photon map, store photons on every diffuse interaction, including the first one



How to compute direct light

Method 1:

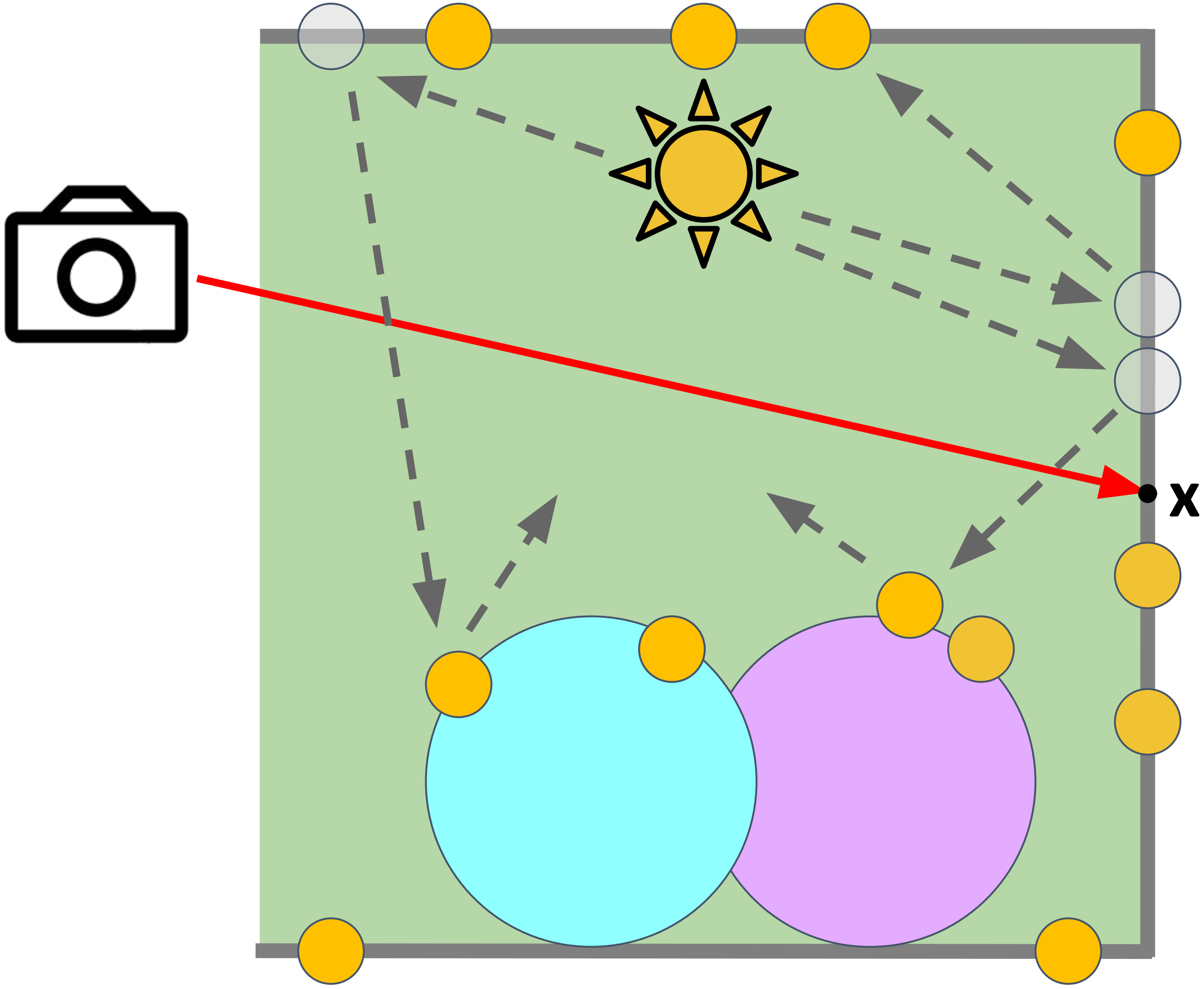
- When generating the photon map, store photons on every diffuse interaction, including the first one
- Later, compute direct light using kernel density estimation



How to compute direct light

Method 2:

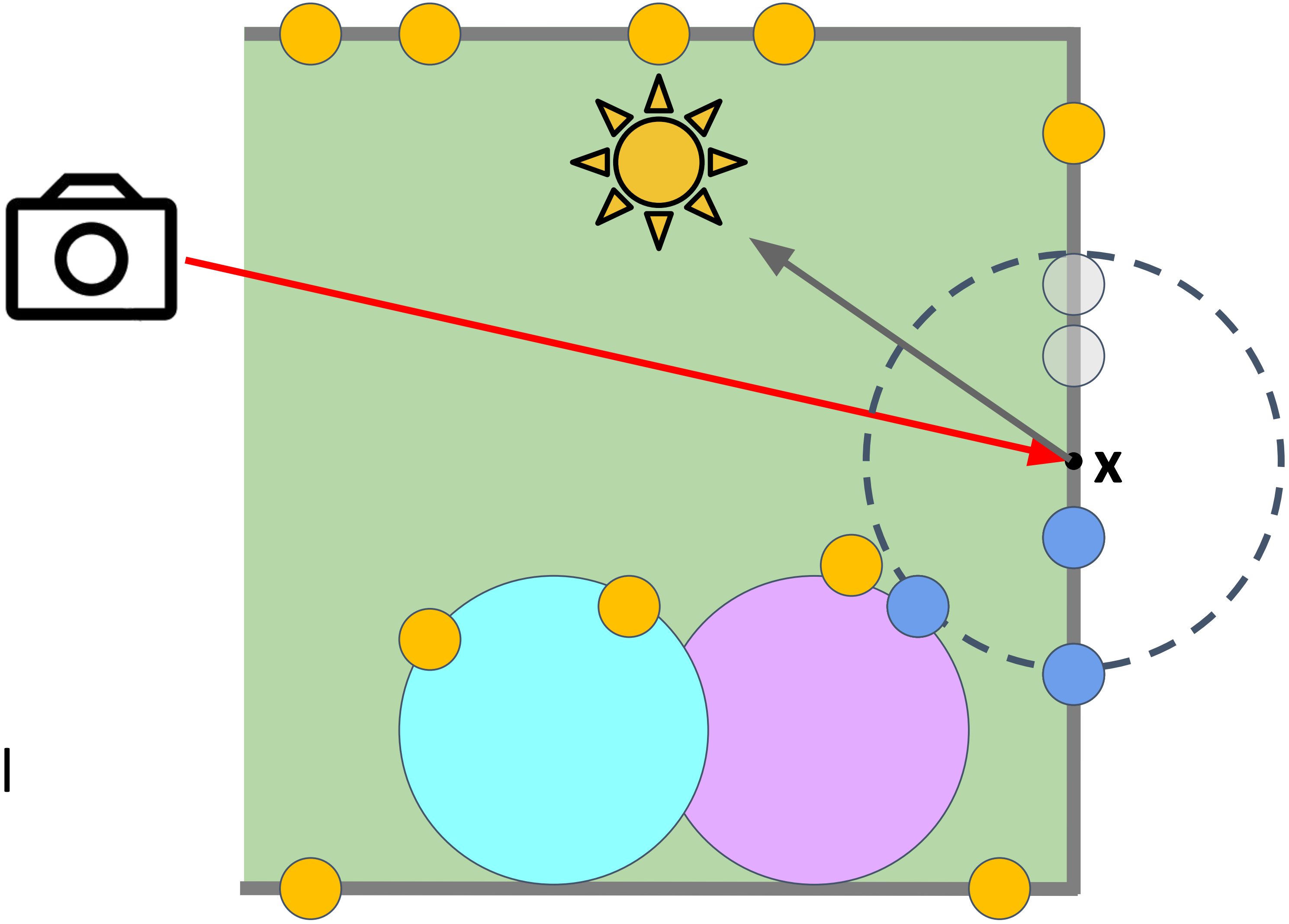
- When generating the photon map, do not store the first bounce (direct) photons



How to compute direct light

Method 2:

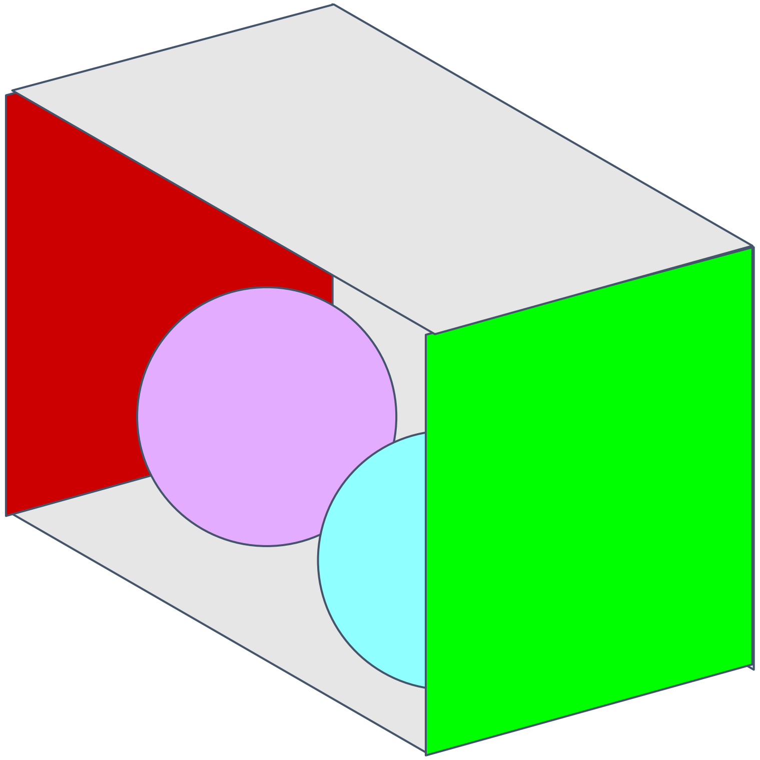
- When generating the photon map, do not store the first bounce (direct) photons
- Later, compute direct light using next-event estimation and indirect light with kernel density estimation



Example scene: Cornell Box

Graphics and Imaging Lab

- **Geometry**



Planes defined by normal (n) and distance (d)

Left plane	n = (1, 0, 0), d = 1
Right plane	n = (-1, 0, 0), d = 1
Floor plane	n = (0, 1, 0), d = 1
Ceiling plane	n = (0, -1, 0), d = 1
Back plane	n = (0, 0, -1), d = 1

Spheres defined by center (c) and radius (r)

Left sphere c = (-0.5, -0.7, 0.25), r = 0.3

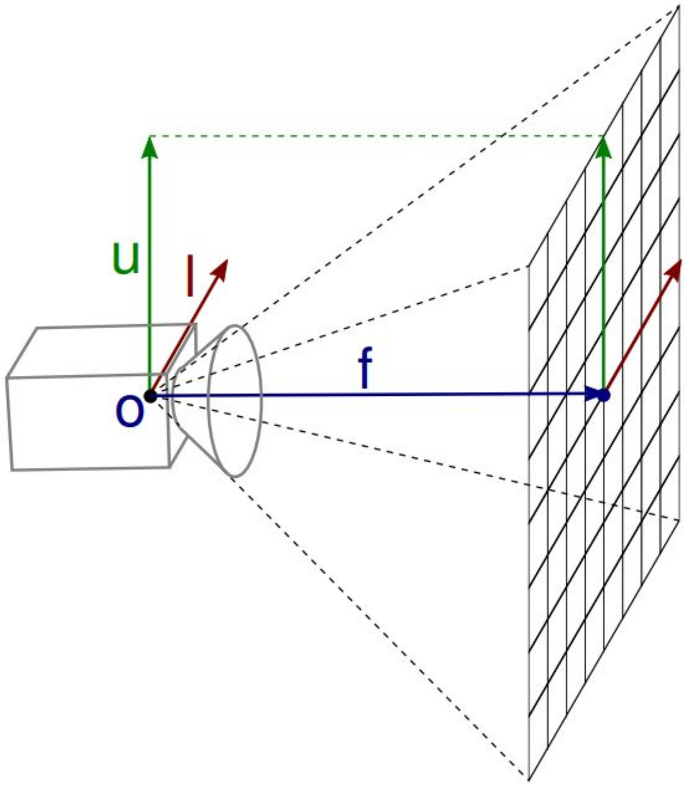
- Diffuse

Right sphere c = (0.5, -0.7, -0.25), r = 0.3

- Diffuse

Example scene: Cornell Box

- **Camera & light sources**

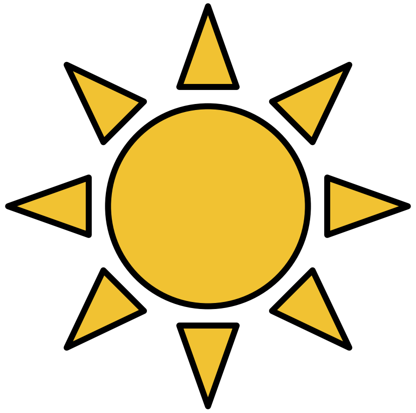


Camera and image plane defined by

Origin	O = (0, 0, -3.5)
Left	L = (-1, 0, 0)
Up	U = (0, 1, 0)
Forward	F = (0, 0, 3)
Size	256x256 pixels

Example scene: Cornell Box

- **Light sources**



Center and power (emission)

Center $c = (0, 0.5, 0)$

Power can be any number e.g. $p = (1, 1, 1)$

Just be careful with the #MAX

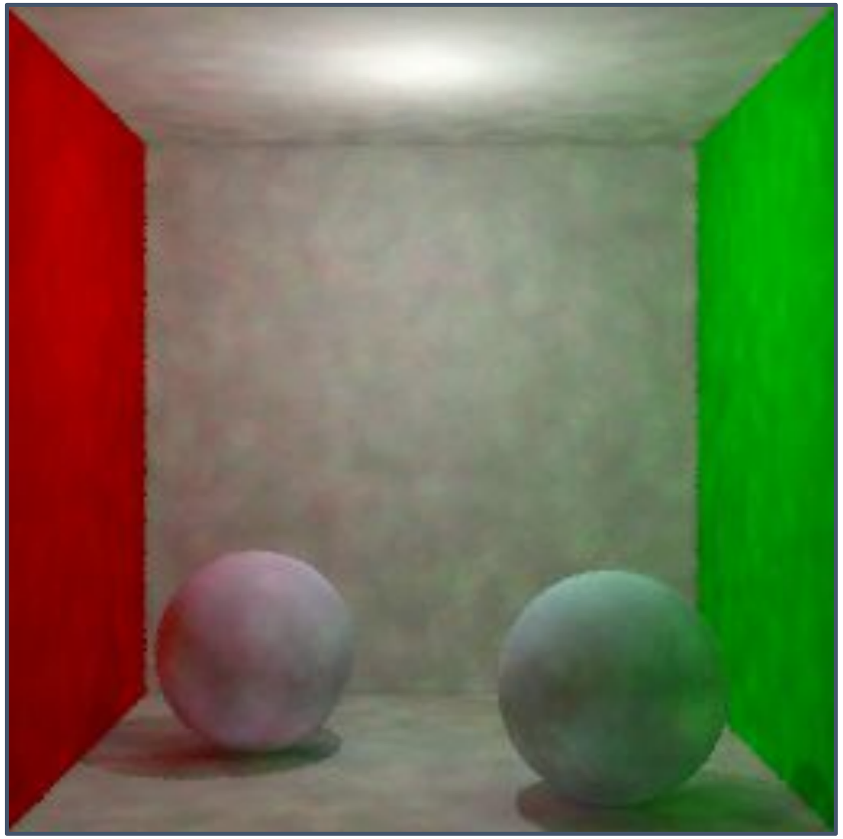
```
P3
# feep.ppm
#MAX=<maximum of your RGB memory values>
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
```

Example scene: Cornell Box

- **Results (direct light is computed using next-event estimation)**

Using a point light

Equalization
Gamma = 2.2



With tone mapping

Questions

DO ASK questions, either now or after the lab

But be reasonable, please :)

pluesia@unizar.es | dsubias@unizar.es | o.pueyo@unizar.es

What to expect from this session

In the programming language of your choice implement:

- Generate an image from your photon map:
    - Launch rays from the camera, intersect at point **x**, find nearest photons
    - Use constant density estimation (box kernel) to estimate the Render Equation
    - Careful with direct light calculation!
- Recommended deadline: November 27th (moodle: January 11th)
    - Extensions (do not count towards recommended deadline):
        - **Recommended to finish base photon mapper before any optionals**
        - Try **different kernels** (cone, Gaussian, or more sophisticated ones)
        - Use an **adaptive kernel bandwidth** (radius)
        - Others: participating media, transient photon mapping, etc. (talk with us before)