Lab #5 – Photon mapping (part 1)

Informática Gráfica

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

Graphics and Imaging Lab

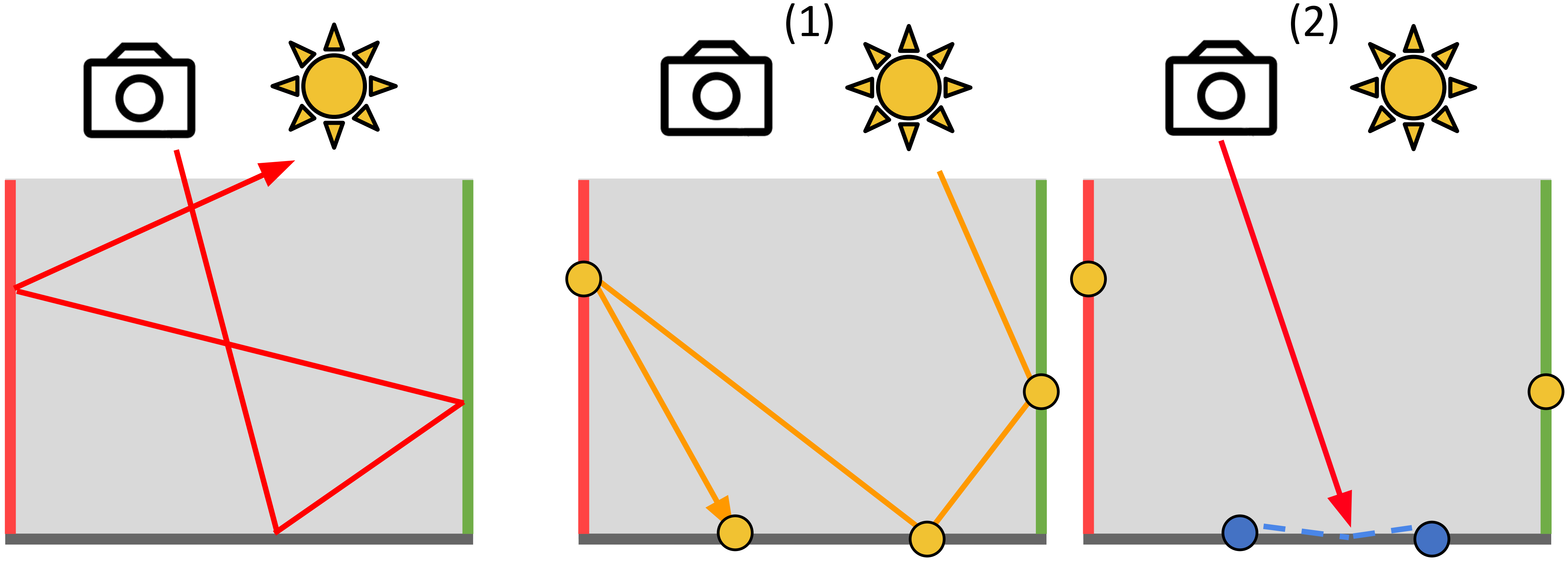
Before we begin…

Graphics and Imaging Lab

- 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

Photon mapping basics

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

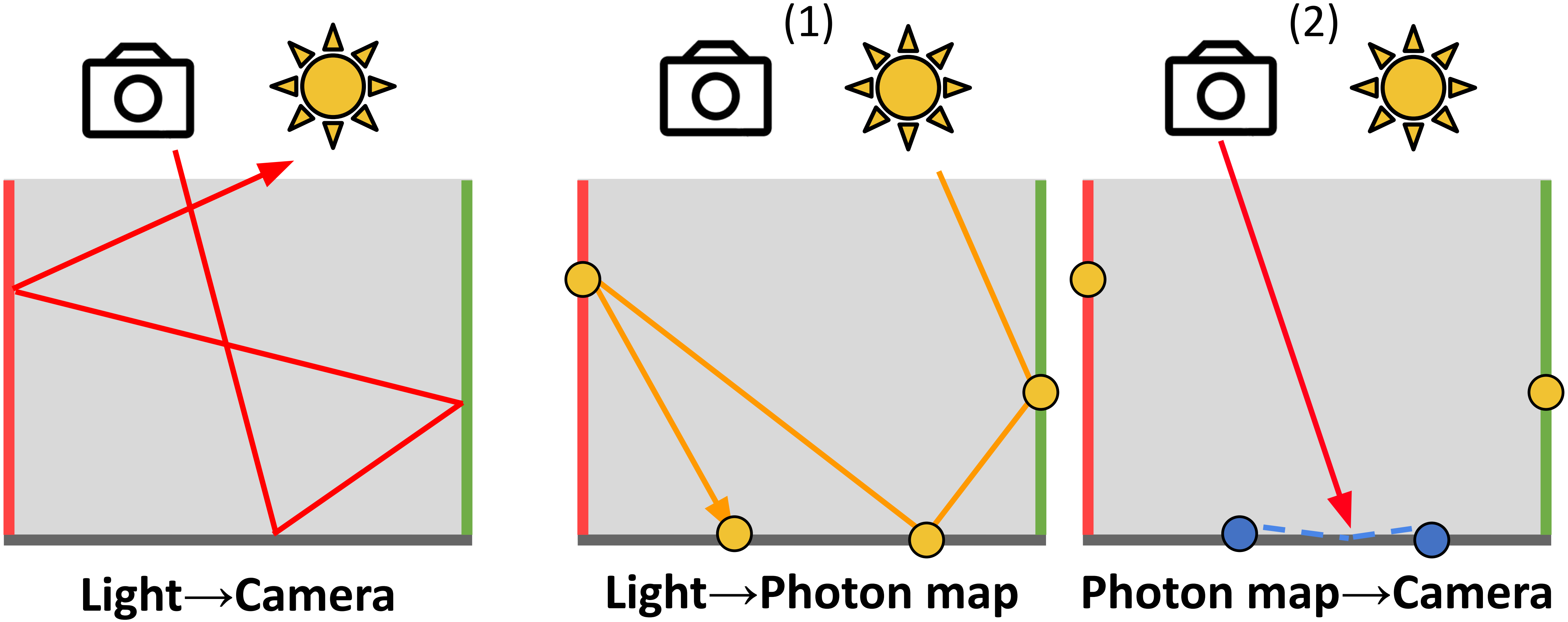


Lab 4 (path tracing)

Lab 5 (photon mapping)

Photon mapping basics

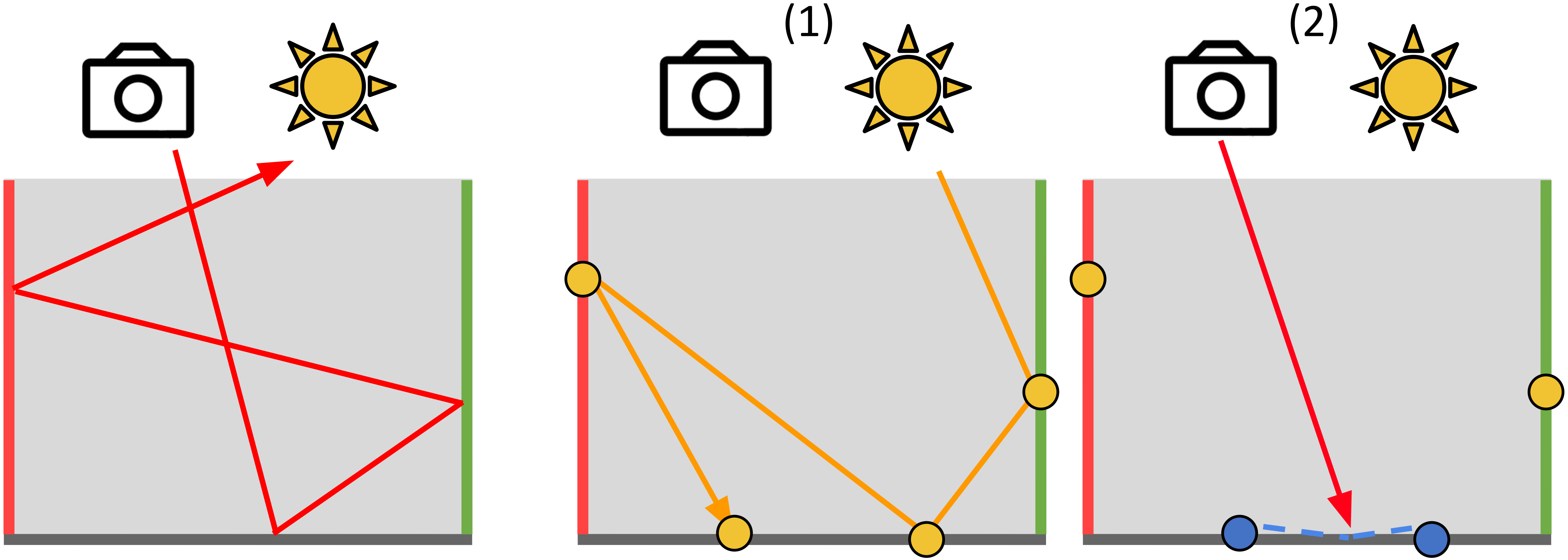
- Photon mapping is a **two-pass** algorithm
  - Uses an **intermediate storage** known as the **photon map**



Photon mapping basics

Graphics and Imaging Lab

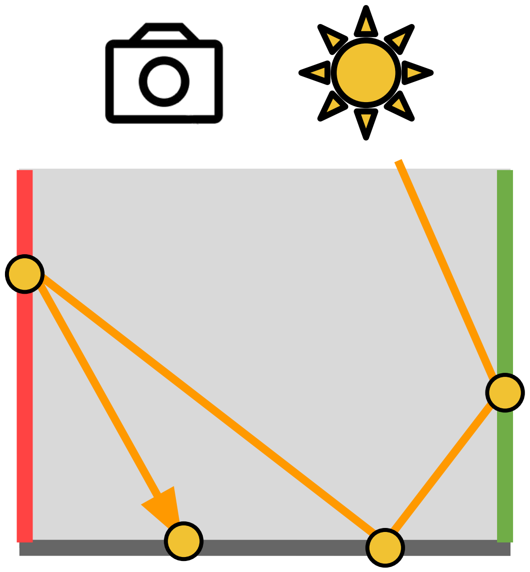
- Lights **write** into the photon map, camera rays **read** from the photon map
- **One camera ray** can read photons from **many light paths**



Light→Camera **Light→Photon map** **Photon map→Camera**

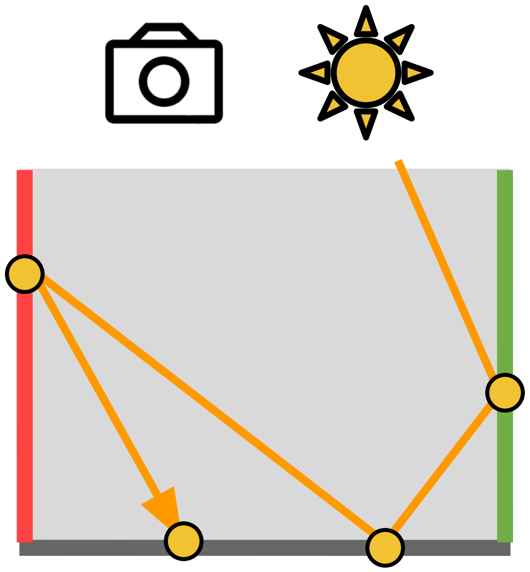
How to create the photon map

- 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**)



How to create the photon map

- 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**)

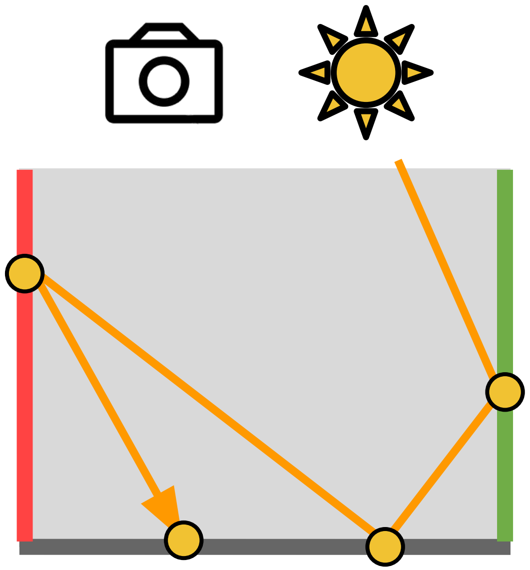


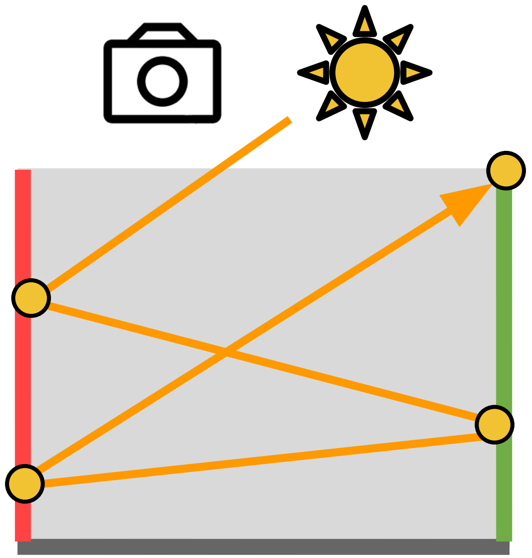
- Example: four photons in this walk
- For each one:
  - Position $\mathbf{x}$
  - Incident direction at x $\omega_{\mathbf{i}}$
  - Flux (similar to radiance)

How to create the photon map

Graphics and Imaging Lab

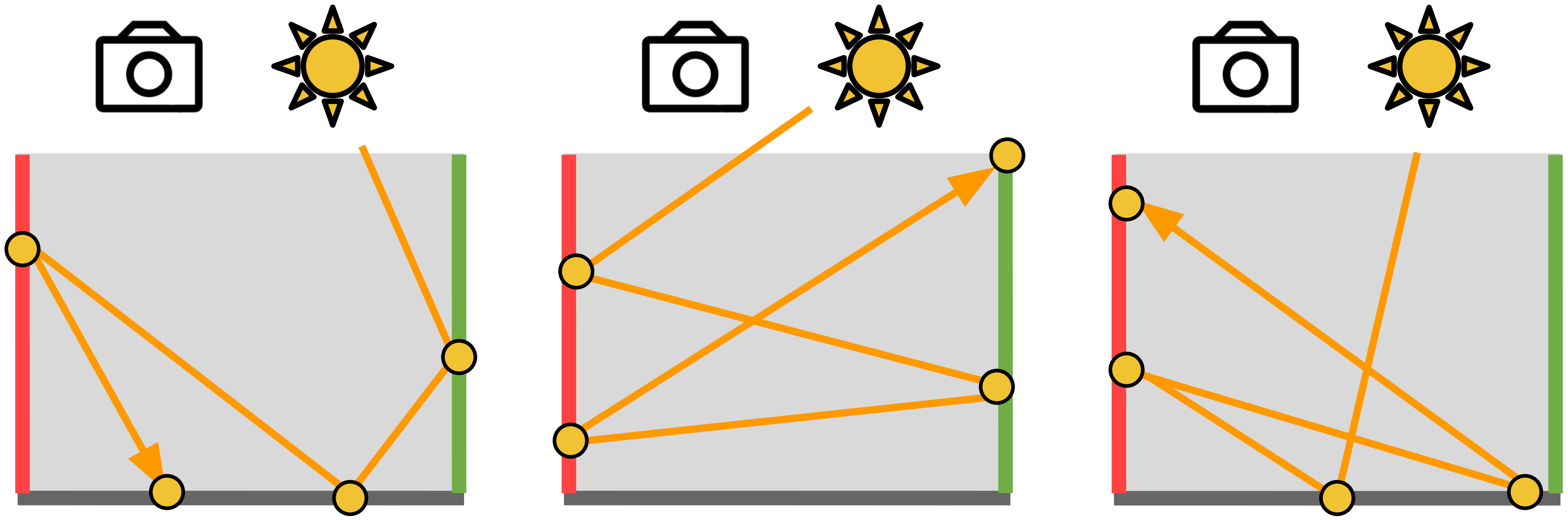
- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map
- Photons are sent out from the light sources (**photon random walks**)





How to create the photon map

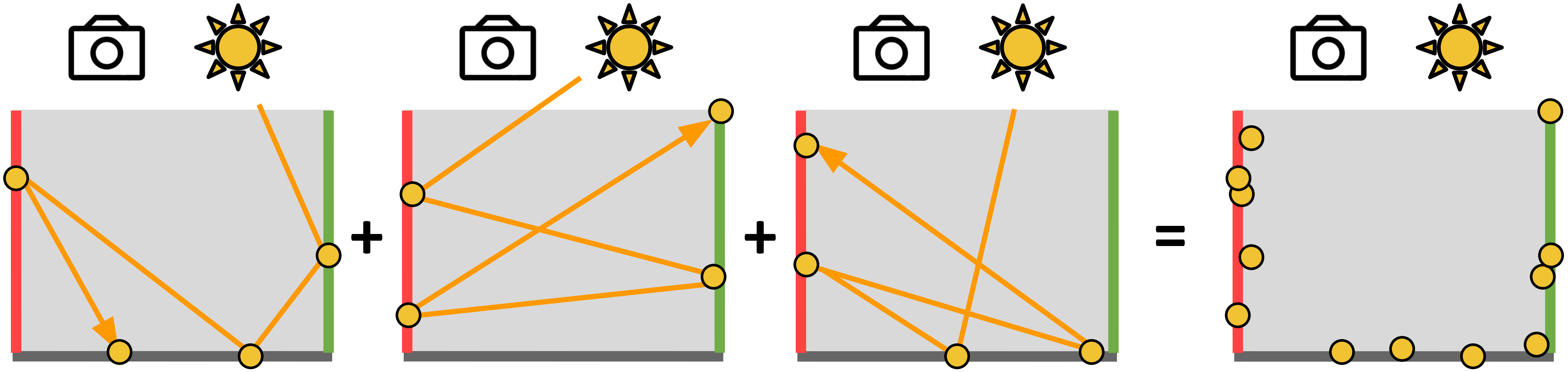
- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map
- Photons are sent out from the light sources (**photon random walks**)



How to create the photon map

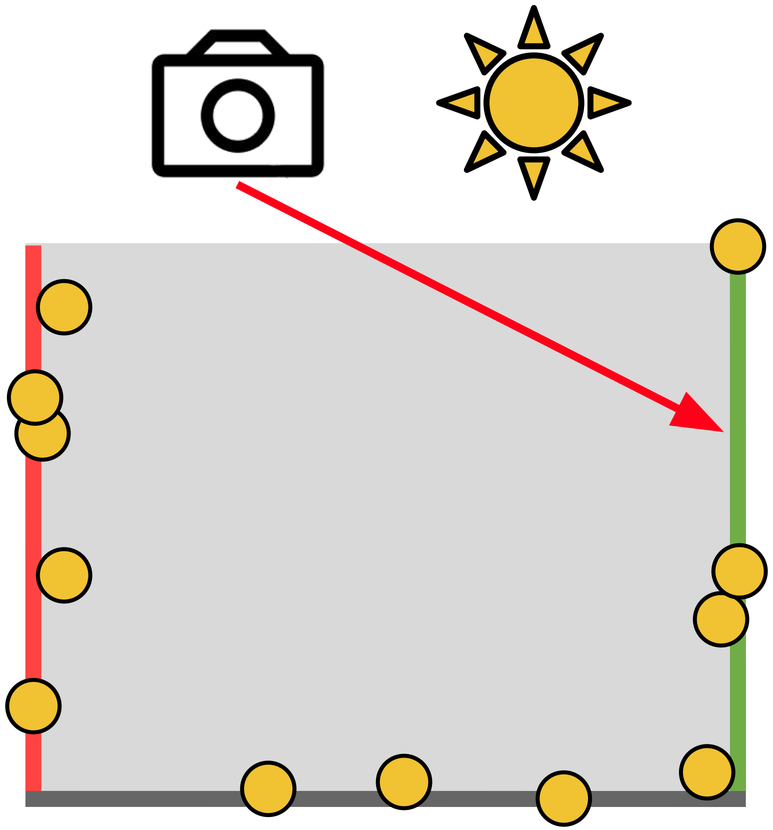
- Split a path into two:
  - (1) lights write into the photon map
  - (2) camera reads from the photon map
- Photons are sent out from the light sources (**photon random walks**)

Final photon map



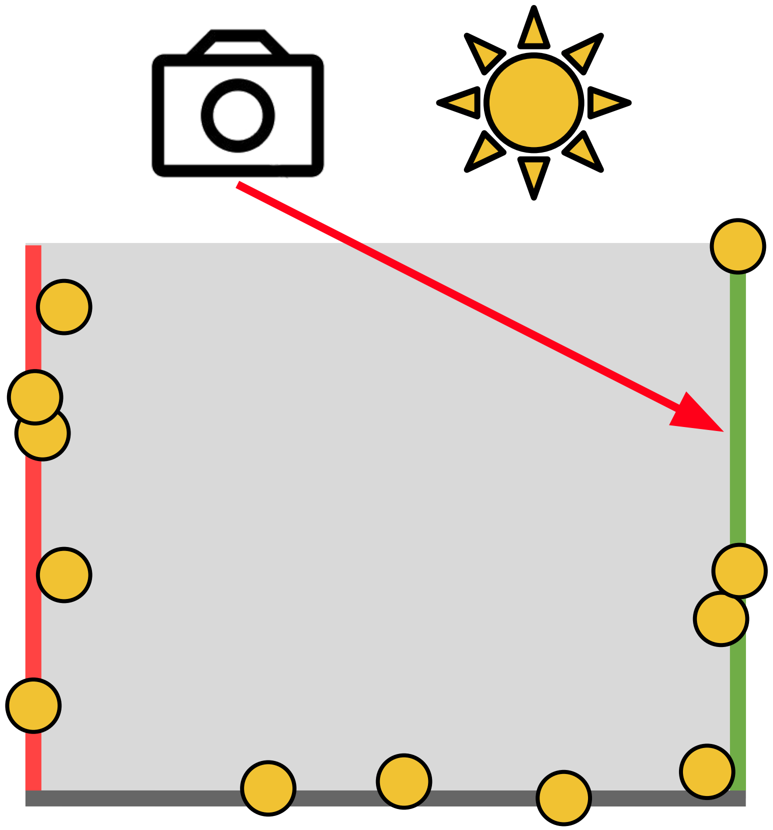
How the photon map is used

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



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

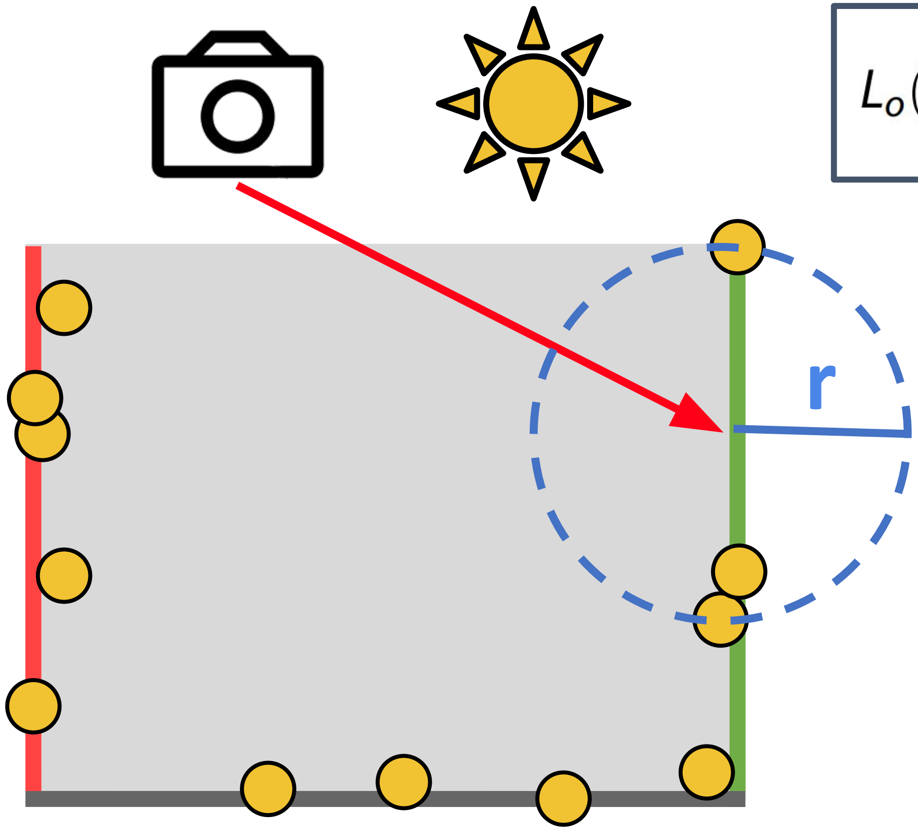


$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \boxed{\int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) |\mathbf{n} \cdot \omega_\mathbf{i}| d\omega_\mathbf{i}}$$

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

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

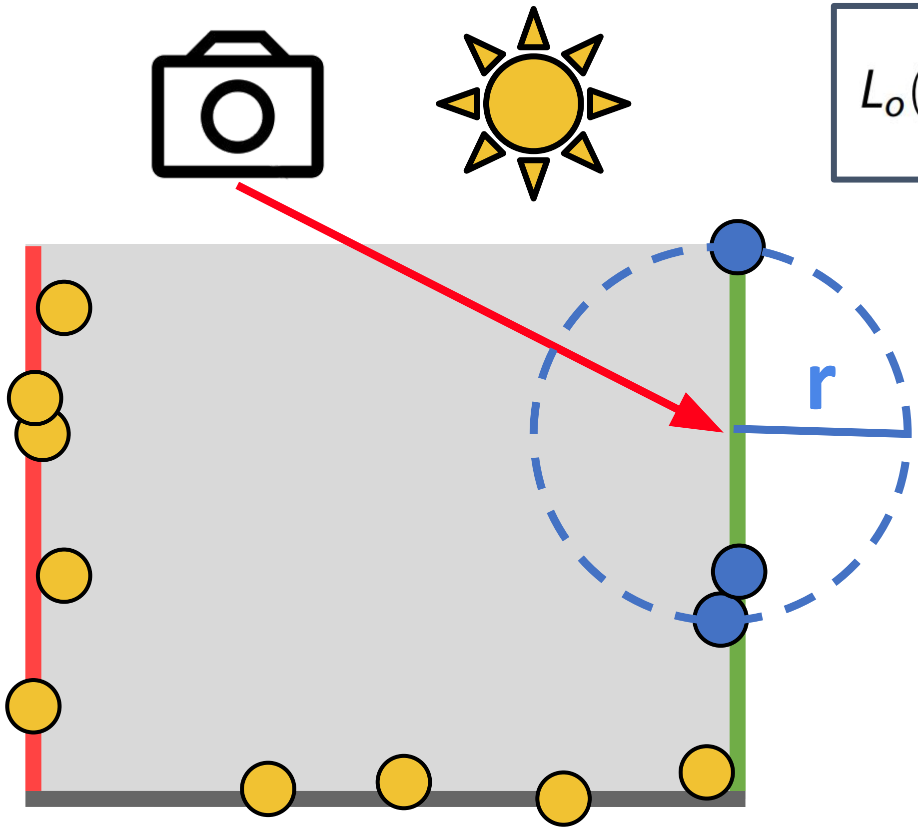


- Idea: use **k nearby** photons (distance < r) to approximate the Render Equation

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

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

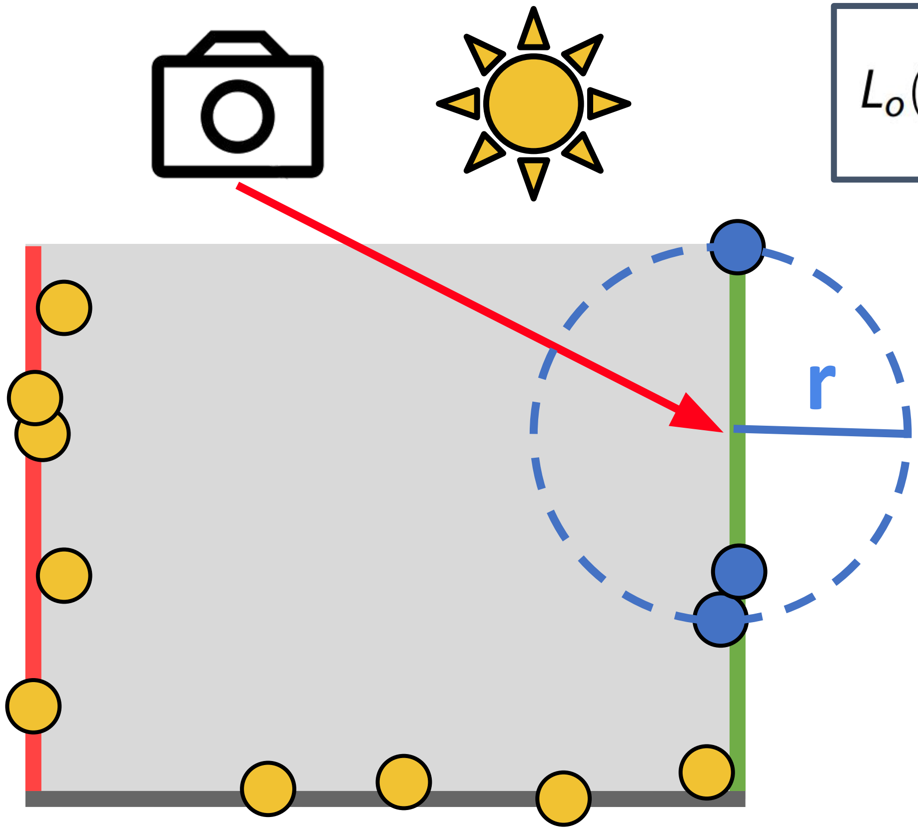


- Idea: use **k nearby** photons (distance < r) to approximate the Render Equation

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

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



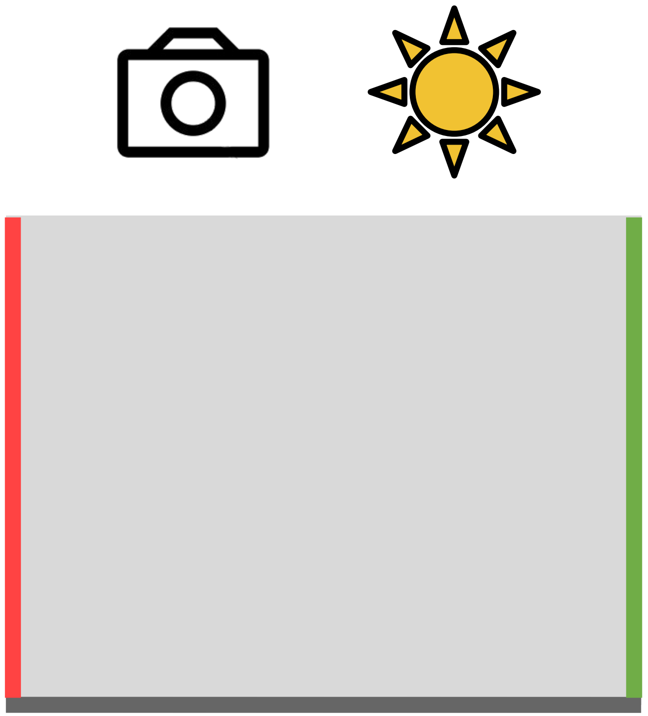
- Idea: use **k nearby** photons (distance < r) to approximate the Render Equation
- Integral is approximated as the **sum of k = 3 contributions**

Implementation of the photon map

- Photon map is a collection of N photons
    - Position
    - Incident direction
    - Flux

Implementation of the photon map

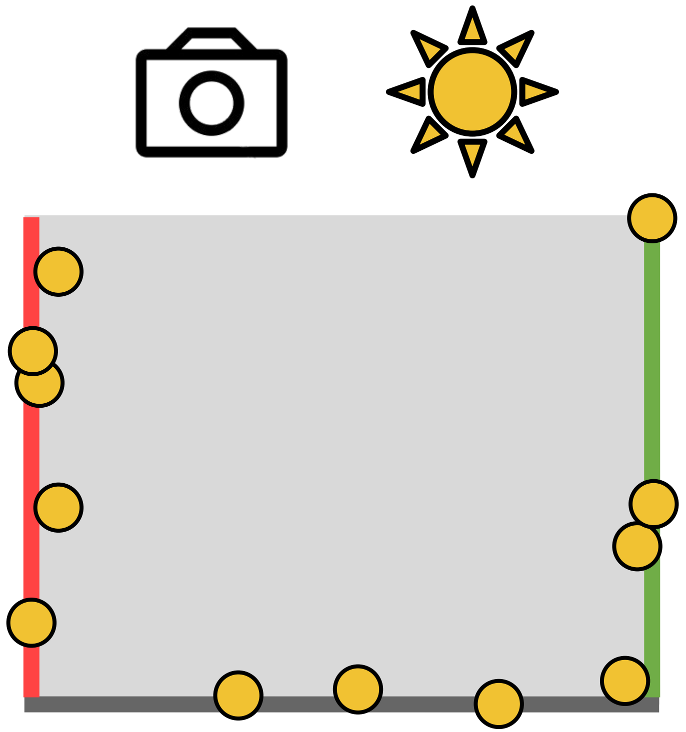
- Photon map is a collection of N photons
  - Position
  - Incident direction
  - Flux
- Operations:
  - Add a photon / a list of photons



Implementation of the photon map

- Photon map is a collection of N photons
- Operations:
  - Add a photon / a list of photons

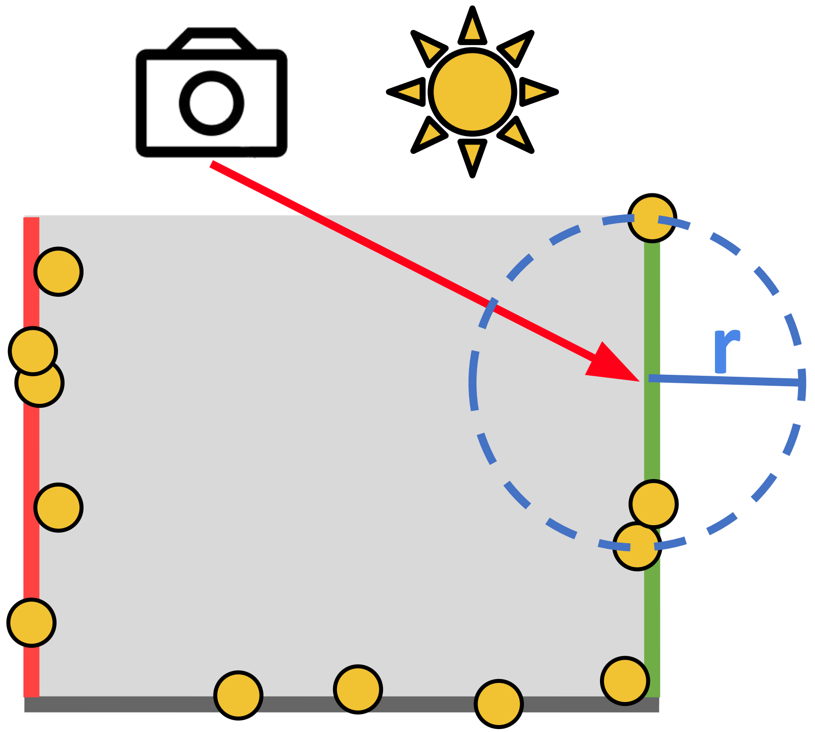
- Position
- Incident direction
- Flux



Implementation of the photon map

Graphics and Imaging Lab

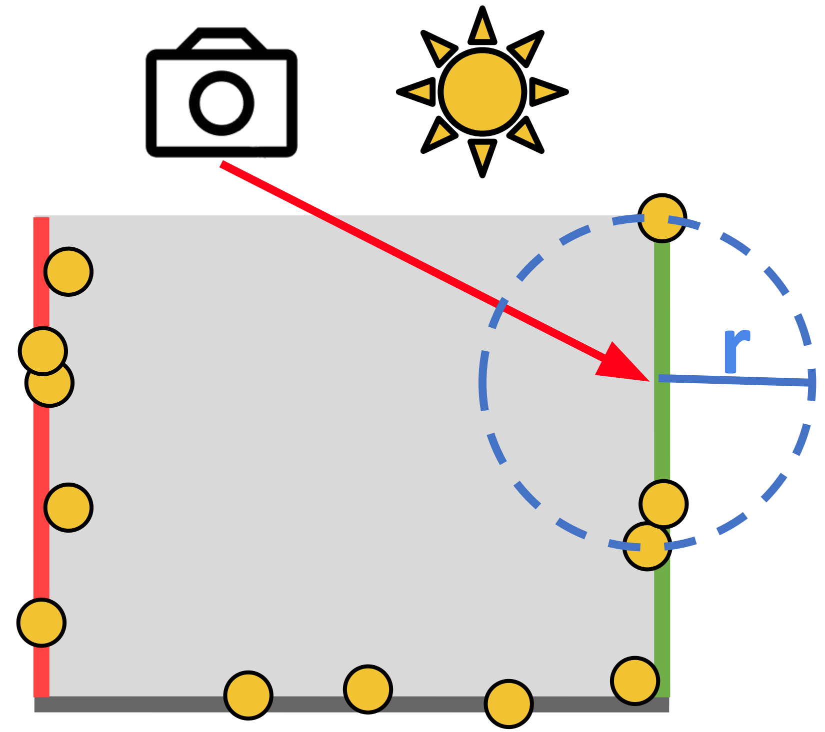
- Photon map is a collection of N photons
  - Position
  - Incident direction
  - Flux
- Operations:
  - Add a photon / a list of photons
  - Find photons near a point $\boxed{\mathbf{x}}$
    - Filter by distance < r
    - Filter by k nearest photons



Implementation of the photon map

Graphics and Imaging Lab

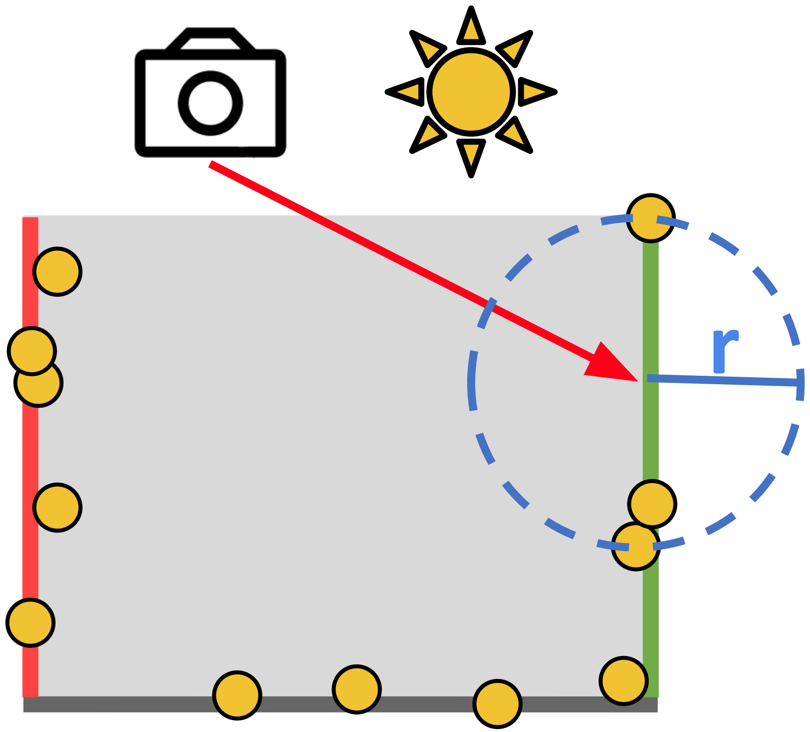
- Photon map is a collection of N photons
    - Position
    - Incident direction
    - Flux
- Operations:
    - Add a photon / a list of photons
    - Find photons near a point $\mathbf{x}$
        - Filter by distance < r
        - Filter by k nearest photons
        - Usually the number of stored photons is huge (large N)



Implementation of the photon map

Graphics and Imaging Lab

- Photon map is a collection of N photons
  - Position
  - Incident direction
  - Flux
- Operations:
  - Add a photon / a list of photons
  - Find photons near a point $\boxed{\mathbf{x}}$
    - Filter by distance < r
    - Filter by k nearest photons
    - Usually the number of stored photons is huge (large N)



- We need a data structure that implements these **two operations as fast as possible**

Implementation of the photon map

Graphics and Imaging Lab

- Data structures for the photon map

Implementation of the photon map

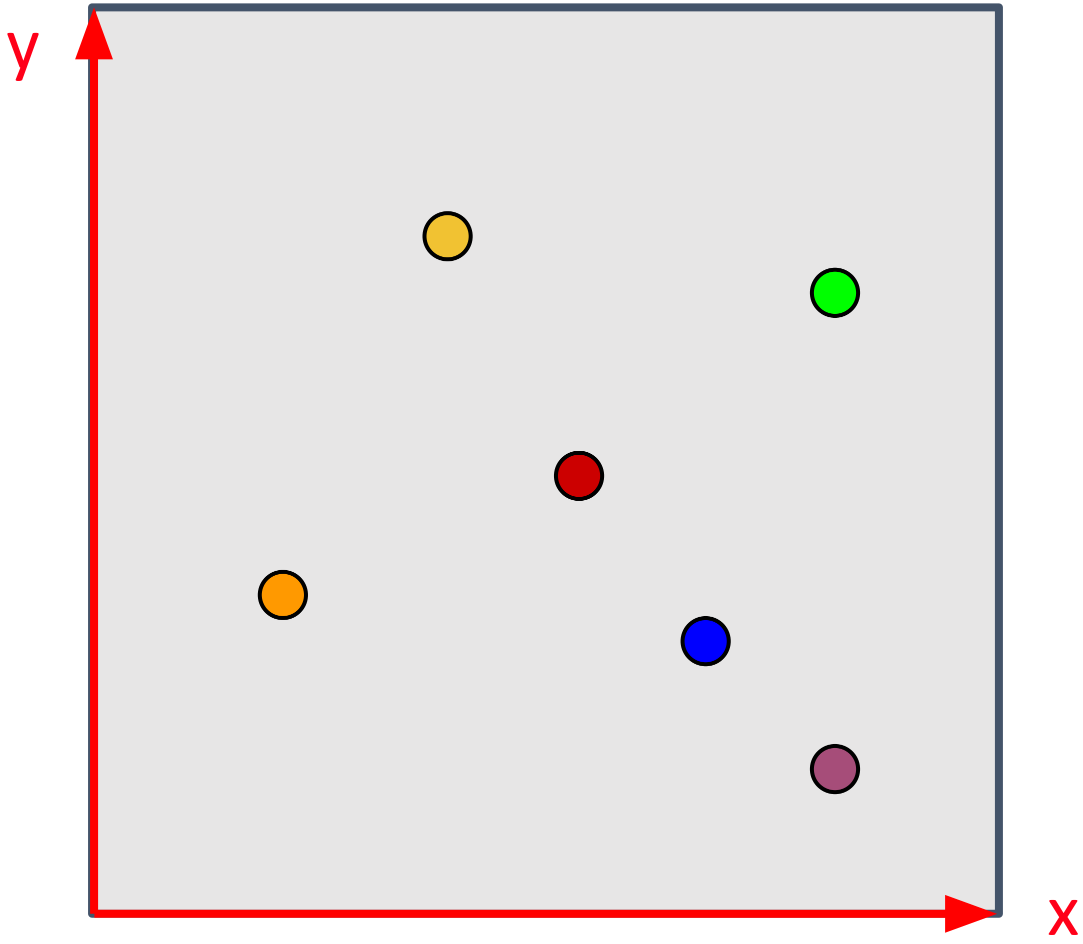
- Data structures for the photon map
- **Array**
  - Add photons → Add elements to the array
  - Search nearby photons → Loop through all N elements of the array, O(N)
  - **Very** inefficient

Implementation of the photon map

- Data structures for the photon map
- **Array**
  - Add photons → Add elements to the array
  - Search nearby photons → Loop through all N elements of the array, O(N)
  - **Very** inefficient
- **k-d tree**
  - Structure for organizing points (photons) in a k dimensional space
  - We use a space with k = 3 dimensions (x, y, z)

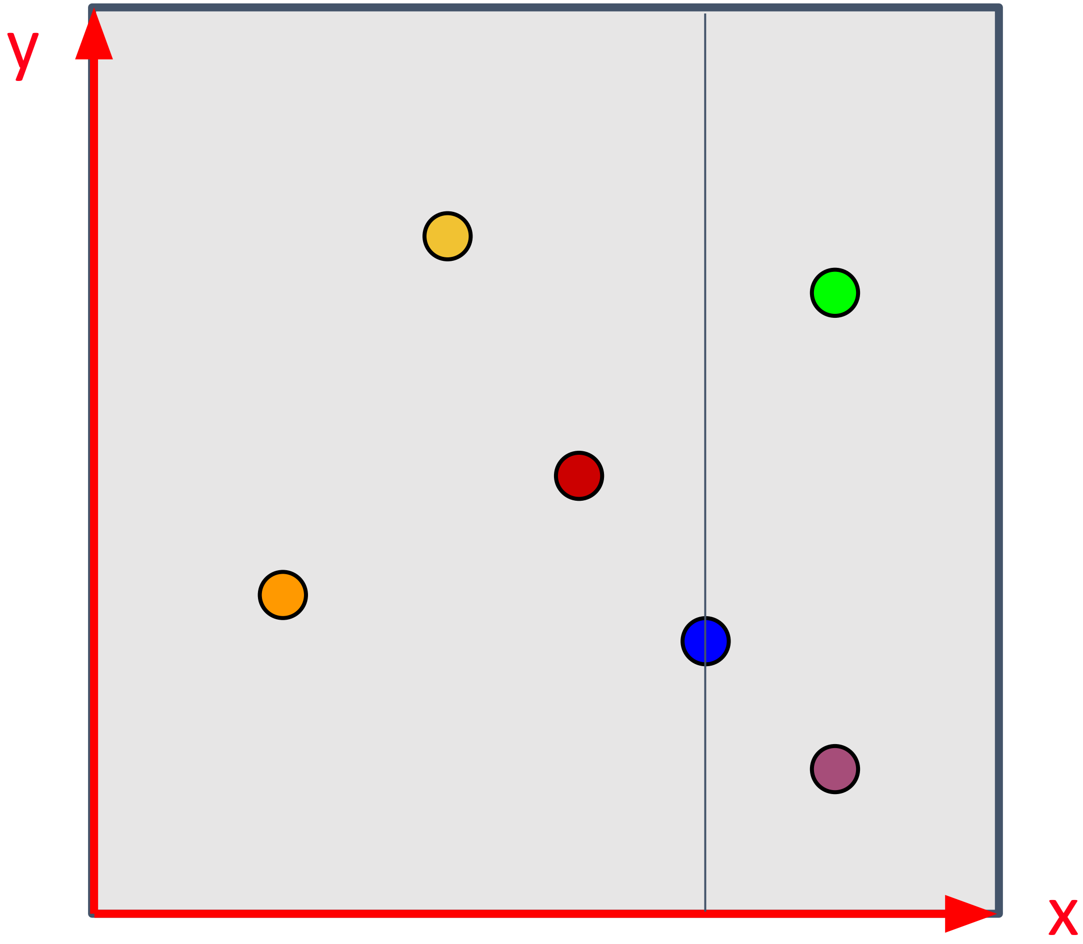
Implementation of the photon map

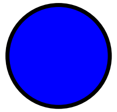
- k-d tree example with k = 2 dimensions



Implementation of the photon map

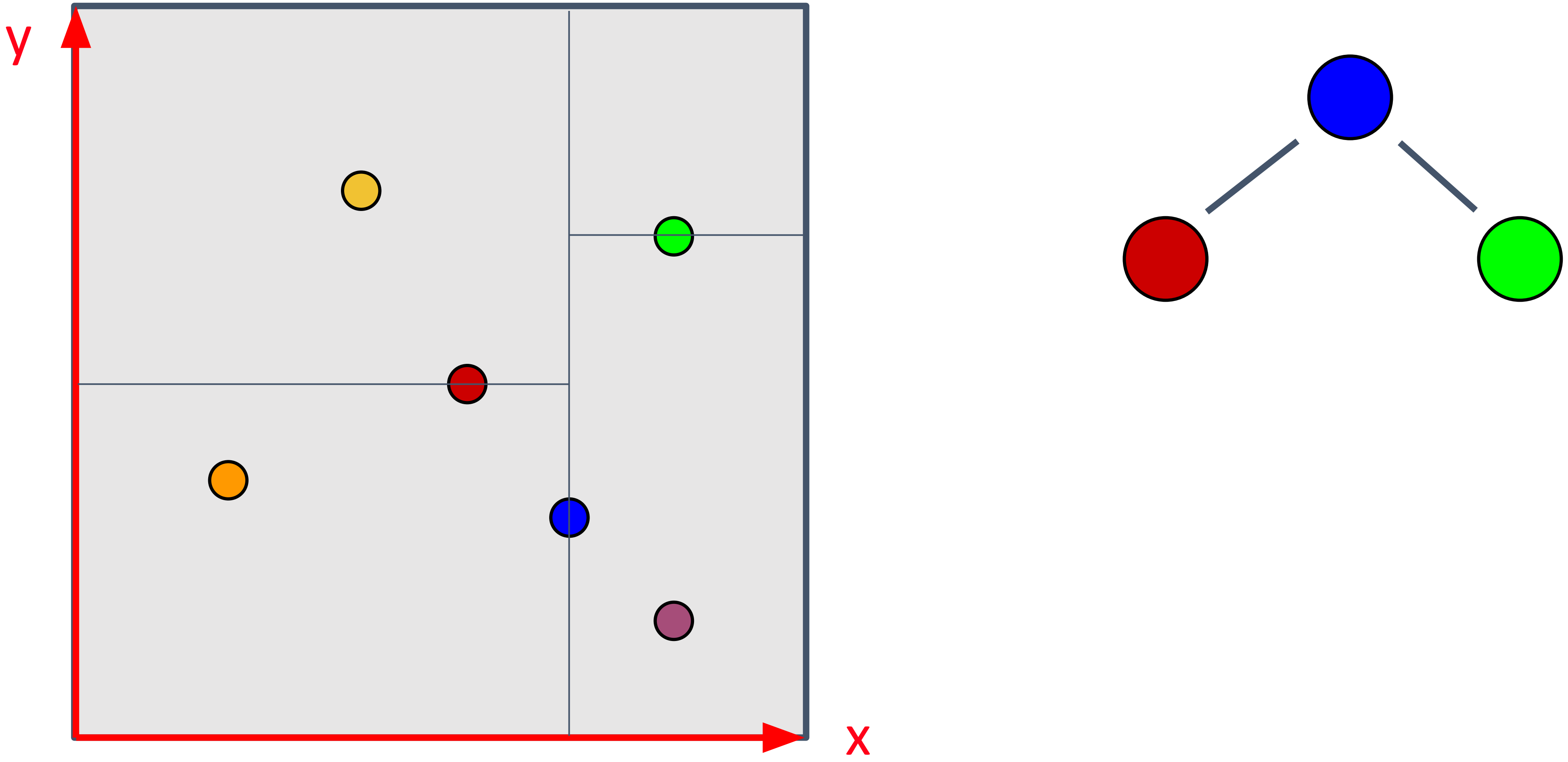
- k-d tree example with k = 2 dimensions





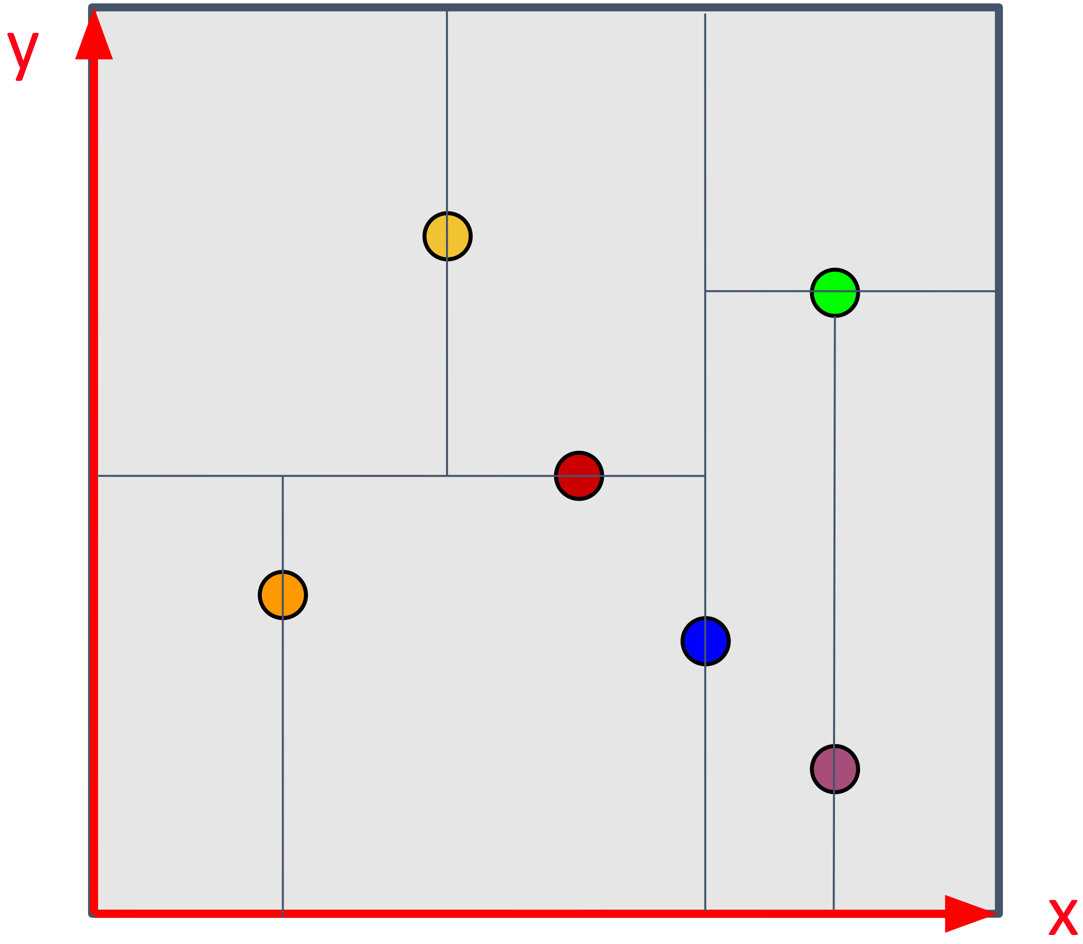
Implementation of the photon map

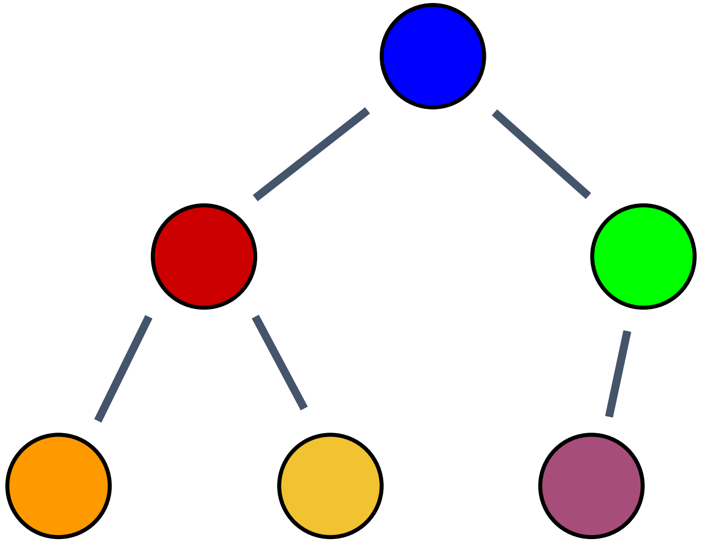
- k-d tree example with k = 2 dimensions



Implementation of the photon map

- k-d tree example with k = 2 dimensions





Implementation of the photon map

- Data structures for the photon map
- **Array**
  - Add photons → Add elements to the array
  - Search nearby photons → Loop through all N elements of the array, O(N)
  - **Very** inefficient
- **k-d tree**
  - Structure for organizing points (photons) in a k dimensional space
  - We use a space with k = 3 dimensions (x, y, z)
  - Add photons → Create a tree that partitions space, leaves/children = photons
  - Search nearby photons → **Space partitioning gives average time O(log N)**

Today: Generating the photon map

Graphics and Imaging Lab

Today we will focus on the generation of the photon map

- Photon random walk
  1) Select a (point) light source from the scene
  2) Begin the photon random walk with a random direction
  3) Scatter the photon through the scene
  4) Store the scattered photons in the photon map

Today: Generating the photon map

Graphics and Imaging Lab

Today we will focus on the generation of the photon map

- Photon random walk
  1) Select a (point) light source from the scene
  2) Begin the photon random walk with a random direction
  3) Scatter the photon through the scene
  4) Store the scattered photons in the photon map
- Repeat until
  - Maximum number of photons in the photon map (map is full)
  - **Maximum number of random walks**

Today: Generating the photon map

Today we will focus on the generation of the photon map

- Photon random walk
  1) **Select a (point) light source from the scene**
  2) Begin the photon random walk with a random direction
  3) Scatter the photon through the scene
  4) Store the scattered photons in the photon map
- Repeat until
  - Maximum number of photons in the photon map (map is full)
  - Maximum number of random walks

1. Select a light source

The user will indicate the number of photon random walks *N*

- How many photons for each light source?

1. Select a light source

Graphics and Imaging Lab

The user will indicate the number of photon random walks *N*

- How many photons for each light source?
  - Idea 1: all light sources emit same amount of photons

1. Select a light source

The user will indicate the number of photon random walks N

- How many photons for each light source?
  - Idea 1: all light sources emit same amount of photons

$N/2$ photons

$N/2$ photons

1. Select a light source

The user will indicate the number of photon random walks N

- How many photons for each light source?
  - Idea 1: all light sources emit same amount of photons

$N/2$ photons

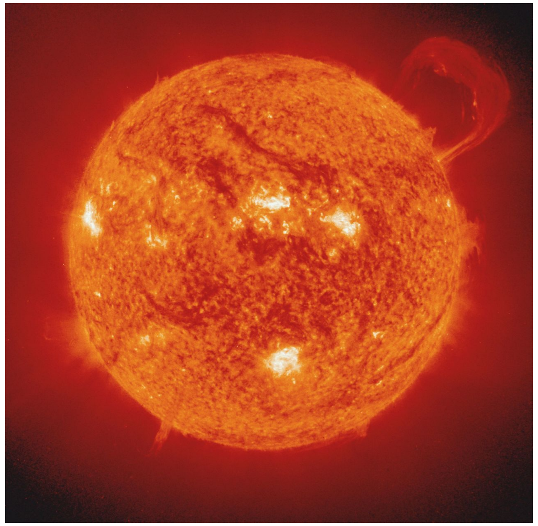
$N/2$ photons

1. Select a light source

The user will indicate the number of photon random walks N

- How many photons for each light source?
  - Idea 1: all light sources emit same amount of photons

$N/2$ photons



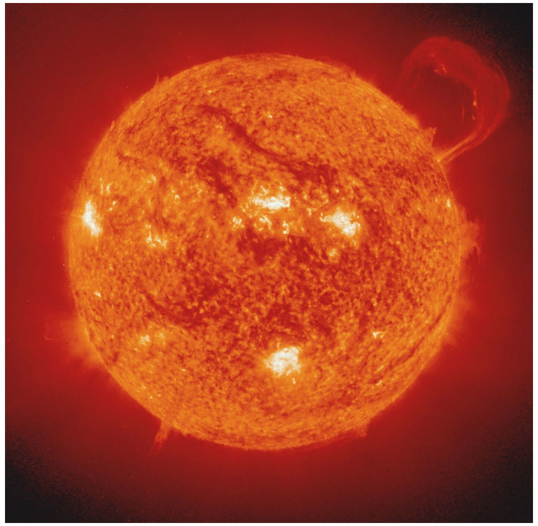
$N/2$ photons

1. Select a light source

The user will indicate the number of photon random walks N

- How many photons for each light source?
  - ~~idea 1: all light sources emit same amount of photons~~

$N/2$ photons



$N/2$ photons

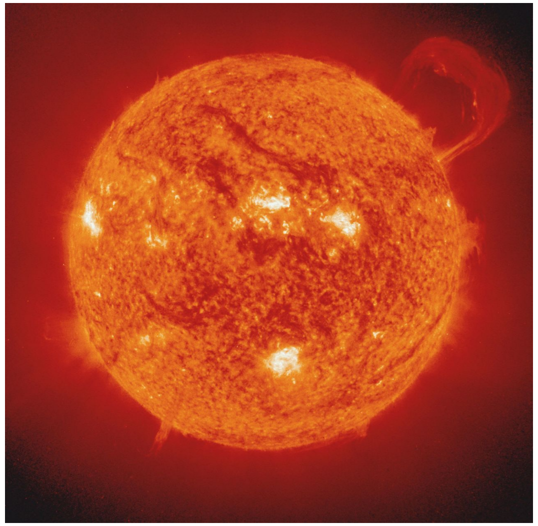
1. Select a light source

The user will indicate the number of photon random walks *N*

- How many photons for each light source?
  - Idea 2: Photons proportional to the emission/power of the light source

p_1

p_2



$N^*(p_1/(p_1 + p_2))$ photons

$N^*(p_2/(p_1 + p_2))$ photons

Today: Generating the photon map

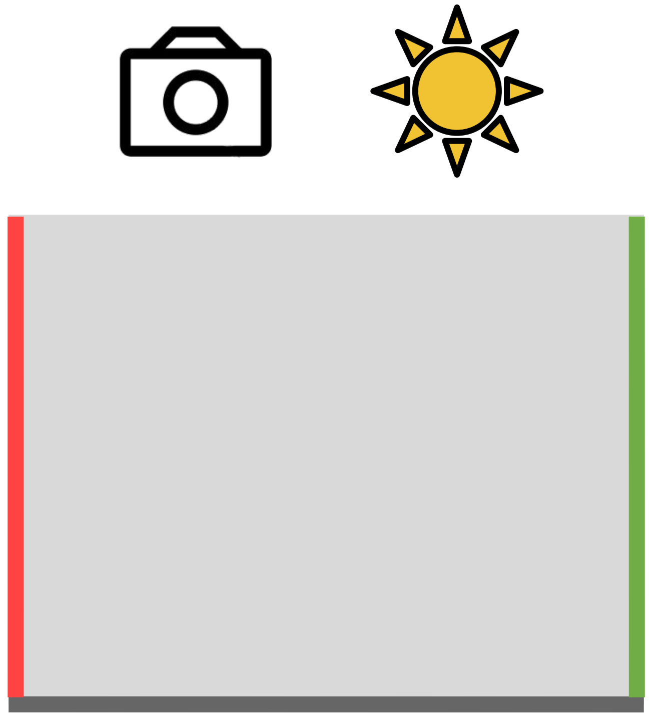
Today we will focus on the generation of the photon map

- Photon random walk
  1) Select a (point) light source from the scene
  2) **Begin the photon random walk with a random direction**
  3) Scatter the photon through the scene
  4) Store the scattered photons in the photon map
- Repeat until
  - Maximum number of photons in the photon map (map is full)
  - **Maximum number of random walks**

2. Begin the random walk

A point light emits photons in all possible (spherical) directions

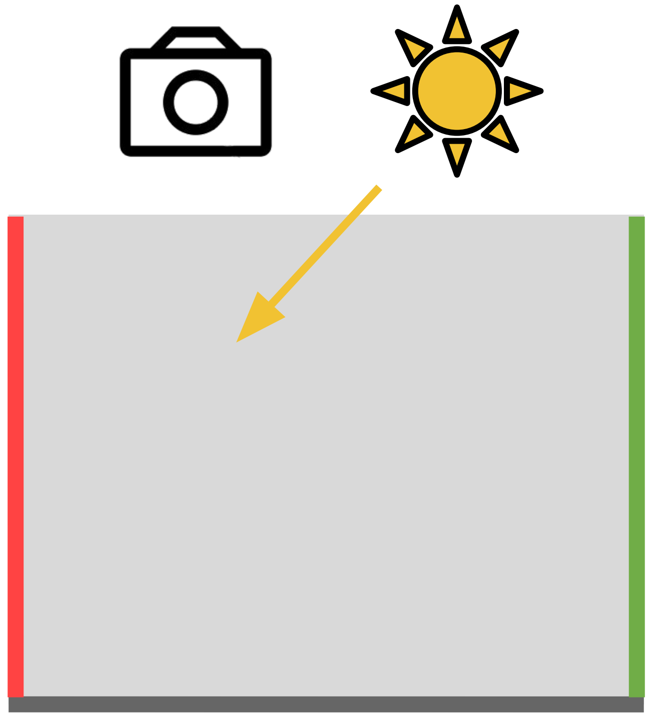
- How to choose a direction to emit each photon?



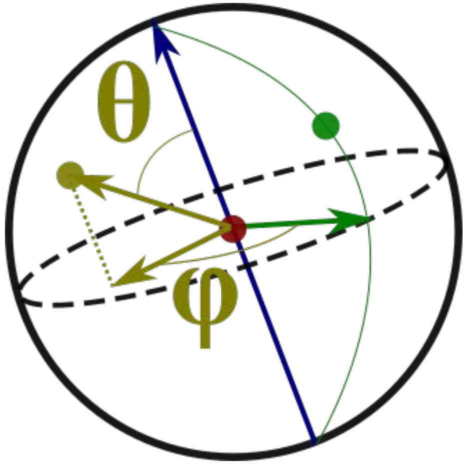
2. Begin the random walk

A point light emits photons in all possible (spherical) directions

- How to choose a direction to emit each photon?
  - Random direction (Monte Carlo)
  - **Sample a direction on the sphere**



2. Begin the random walk



Spherical coordinates:

- $\theta \in [0, \pi)$
- $\varphi \in [0, 2\pi)$

Convert spherical to cartesian

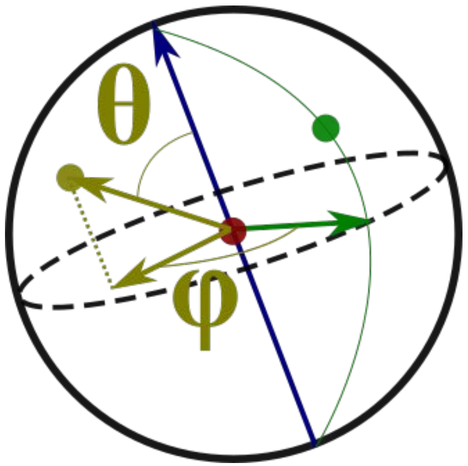
$$\left.\begin{array}{l}\xi_\theta \in [0,1] \\ \xi_\varphi \in [0,1]\end{array}\right\} \text{Uniform random}$$

Uniform angle sampling **is bad**

$\theta = \pi\xi_\theta$

$\varphi = 2\pi\xi_\varphi$

2. Begin the random walk



Spherical coordinates:

- $\theta \in [0, \pi)$
- $\varphi \in [0, 2\pi)$

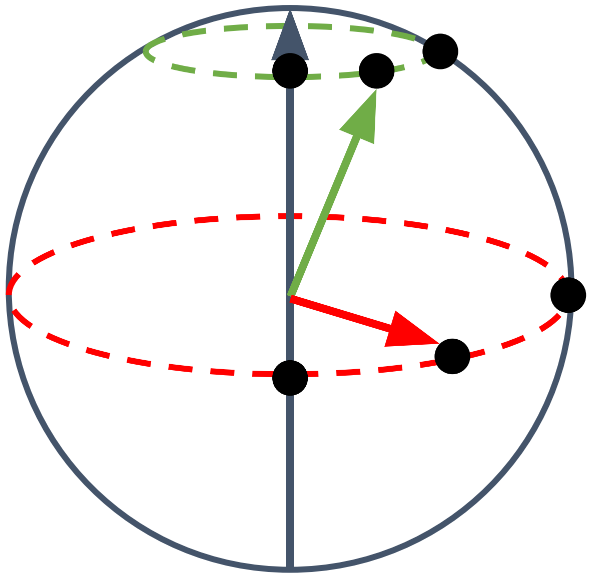
Convert spherical to cartesian

$$\left.\begin{array}{l} \xi_\theta \in [0,1] \\ \xi_\varphi \in [0,1] \end{array}\right\} \text{Uniform random}$$

Uniform angle sampling **is bad**

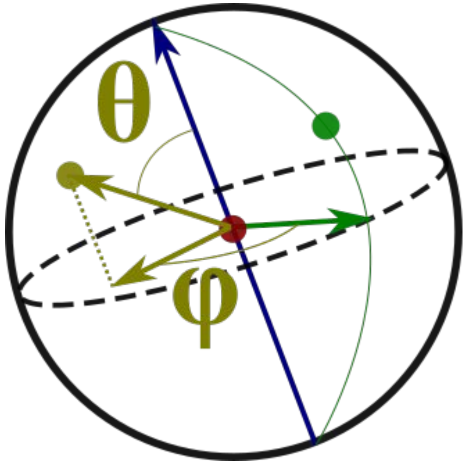
$$\theta = \pi \xi_\theta$$

$$\varphi = 2\pi \xi_\varphi$$



Uniform $\theta, \varphi \neq$ Uniform solid angle ω_i

2. Begin the random walk



Spherical coordinates:

- $\theta \in [0, \pi)$
- $\varphi \in [0, 2\pi)$

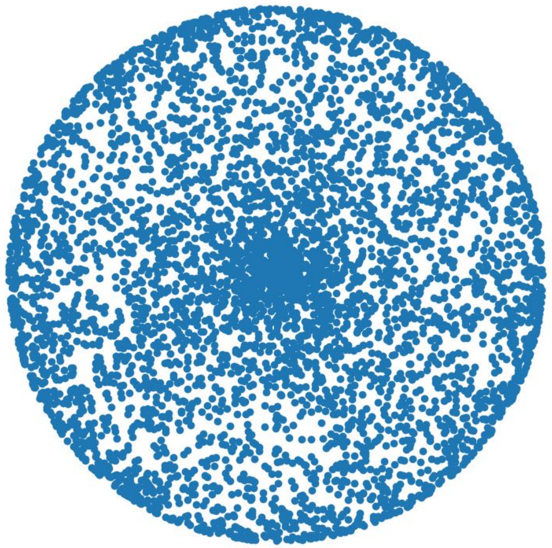
Convert spherical to cartesian

$$\left.\begin{array}{l}\xi_\theta \in [0,1] \\ \xi_\varphi \in [0,1]\end{array}\right\} \text{Uniform random}$$

Uniform angle sampling is bad

$$\theta = \pi\xi_\theta$$

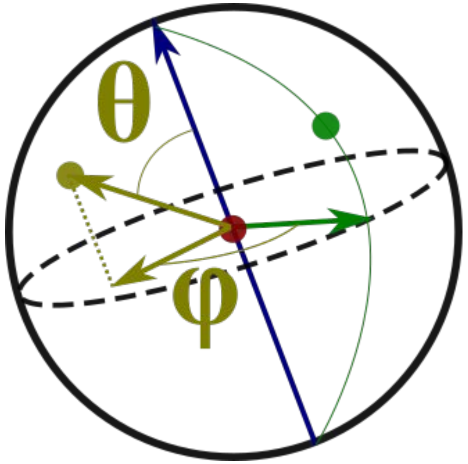
$$\varphi = 2\pi\xi_\varphi$$



Top view (more samples near **poles**)

2. Begin the random walk

Graphics and Imaging Lab



Spherical coordinates:

- $\theta \in [0, \pi)$
- $\varphi \in [0, 2\pi)$

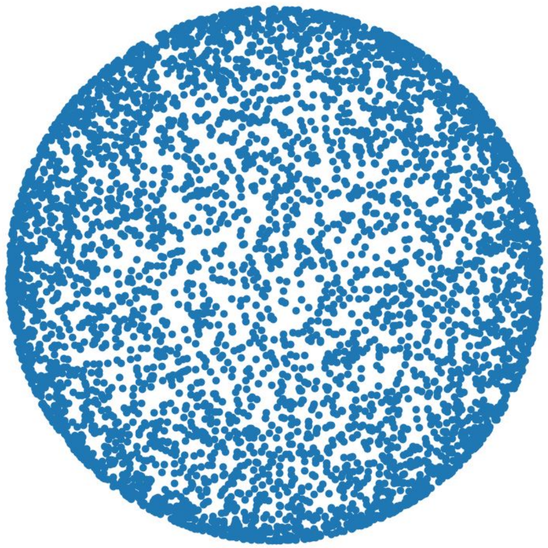
Convert spherical to cartesian

$\xi_\theta \in [0,1]$
$\xi_\varphi \in [0,1]$ Uniform random

Uniform solid angle sampling **is good**

$\theta = \mathrm{acos}(2\xi_\theta - 1)$

$\varphi = 2\pi\xi_\varphi$



Top view (uniformly distributed)

Today: Generating the photon map

Graphics and Imaging Lab

Today we will focus on the generation of the photon map

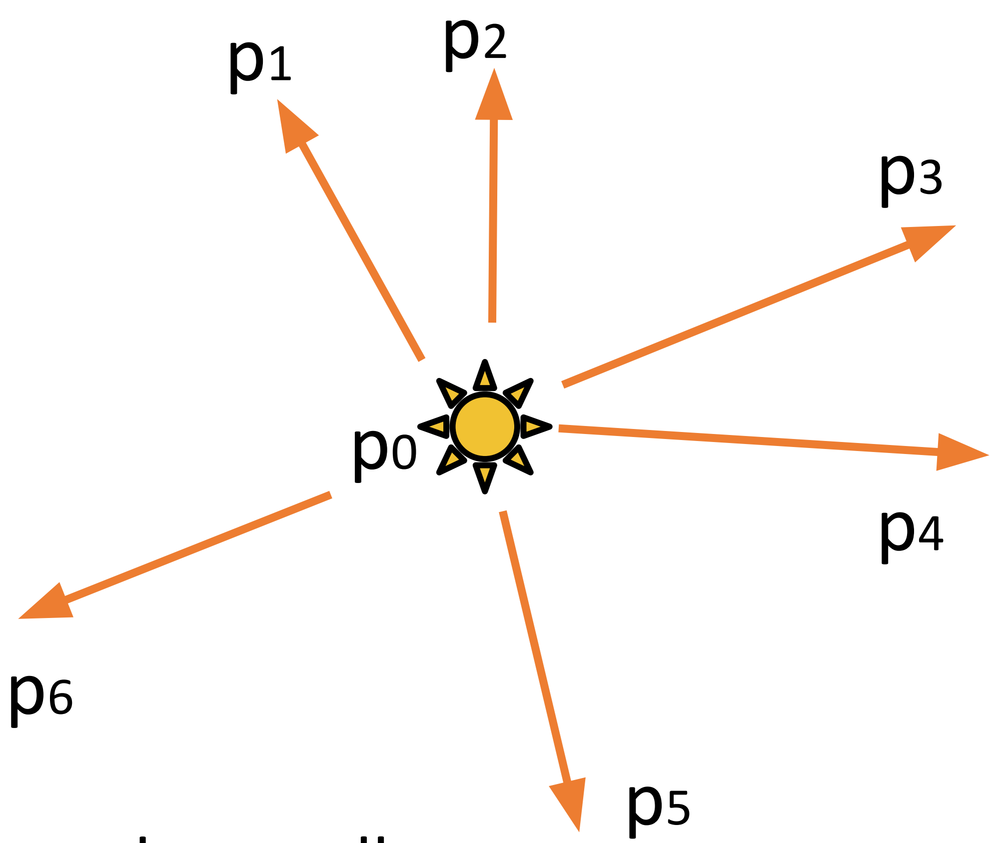
- Photon random walk
  1) Select a (point) light source from the scene
  2) Begin the photon random walk with a random direction
  3) **Scatter the photon through the scene**
  4) Store the scattered photons in the photon map
- Repeat until
  - Maximum number of photons in the photon map (map is full)
  - **Maximum number of random walks**

3. Scatter the photon through the scene

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

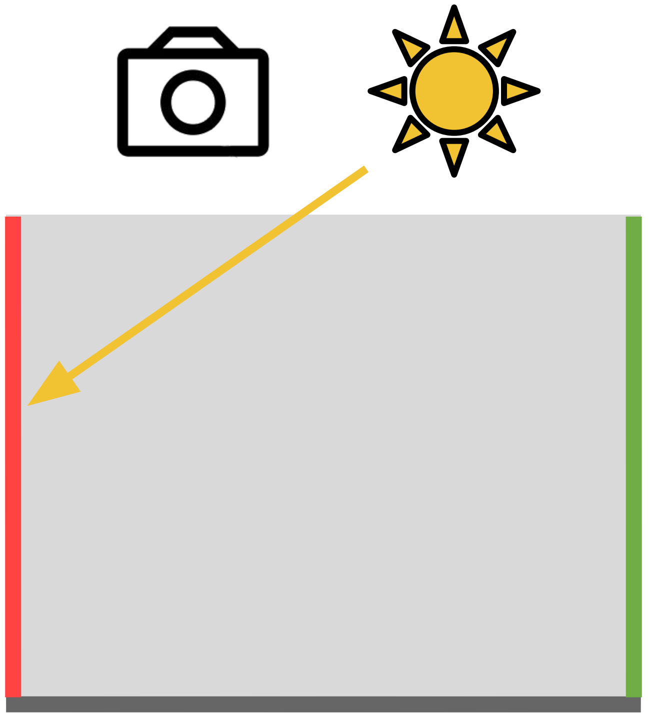
You might need to update S after doing the random walks

3. Scatter the photon through the scene

Photon hits a diffuse surface → Store photon in the photon map

Next: In which directions do photons scatter?

- Note: in this session we **only use diffuse materials**

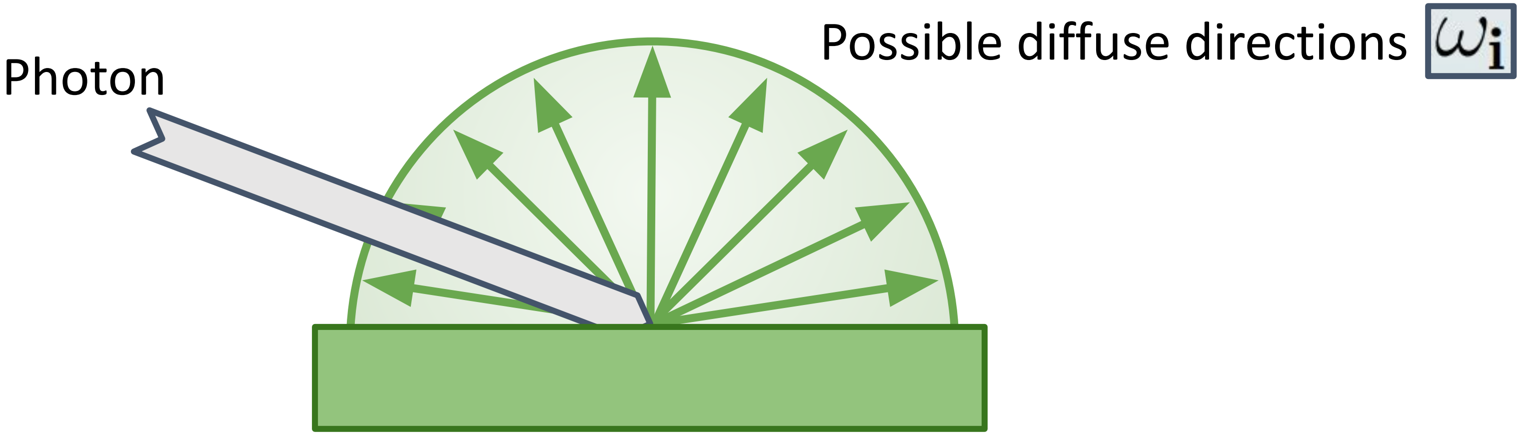


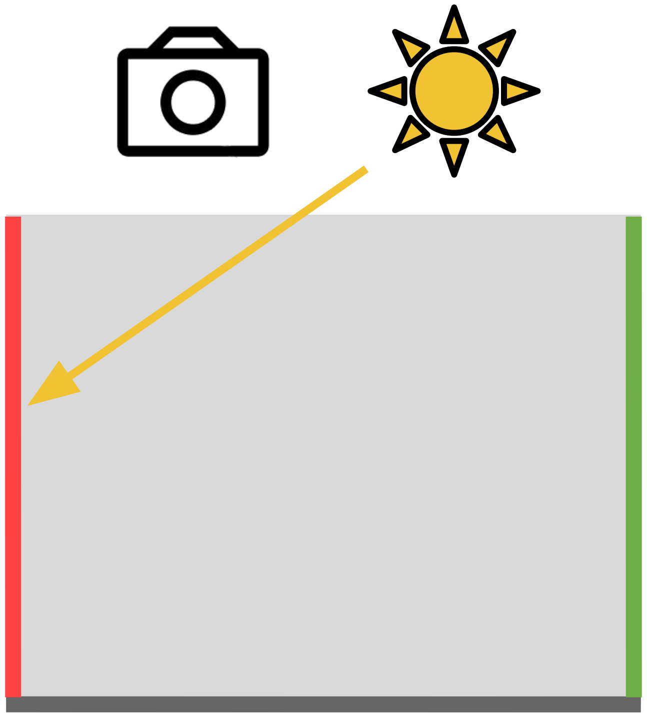
3. Scatter the photon through the scene

Photon hits a diffuse surface → Store photon in the photon map

Next: In which directions do photons scatter?

- Note: in this session we **only use diffuse materials**



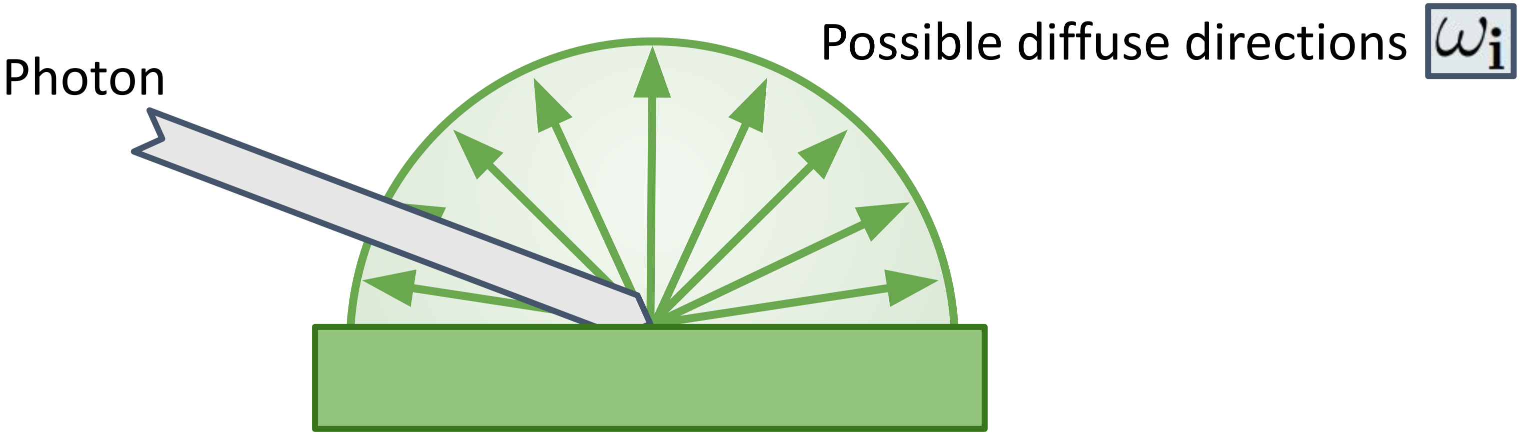


3. Scatter the photon through the scene

Photon hits a diffuse surface → Store photon in the photon map

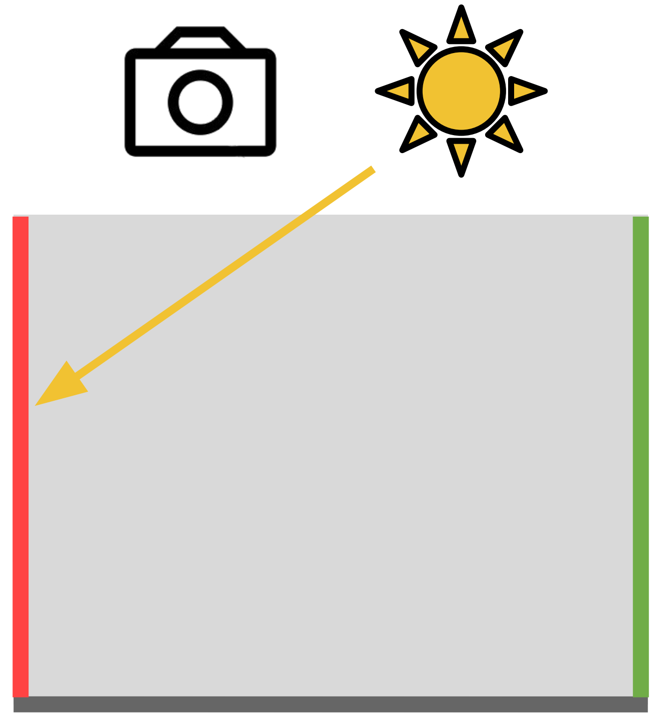
Next: In which directions do photons scatter?

- Note: in this session we **only use diffuse materials**



Sample ω_i → Uniform cosine sampling

Uniform solid angle sampling



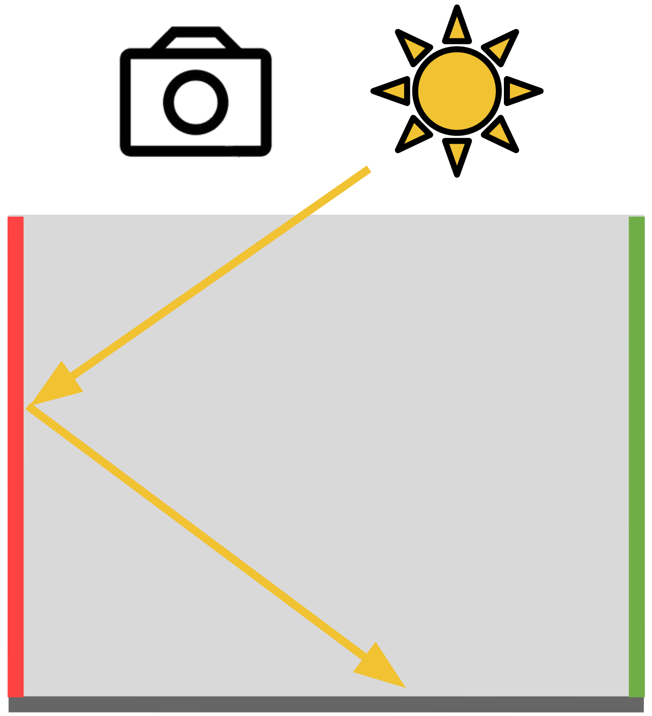
3. Scatter the photon through the scene

Photon hits a diffuse surface → Store photon in the photon map

Next: In which directions do photons scatter?

- Note: in this session we **only use diffuse materials**

- Sample $\boxed{\omega_\mathbf{i}}$
- Evaluate BRDF and cosine term
  $\boxed{f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o})|\mathbf{n} \cdot \omega_\mathbf{i}|}$
- Compute radiance for scattered photon



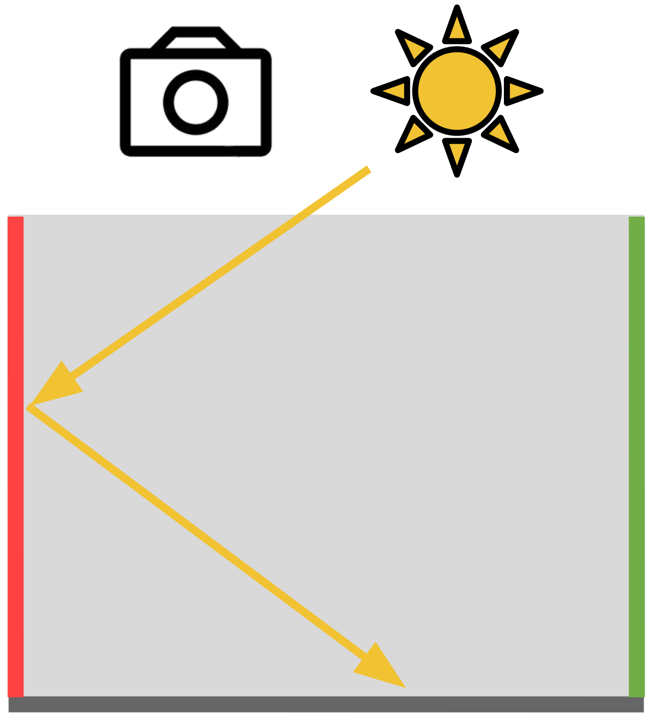
3. Scatter the photon through the scene

Photon hits a diffuse surface → Store photon in the photon map

Next: In which directions do photons scatter?

- Note: in this session we **only use diffuse materials**

- Sample $\boxed{\omega_{\mathbf{i}}}$
- Evaluate BRDF and cosine term
  $\boxed{f_r(\mathbf{x}, \omega_{\mathbf{i}}, \omega_{\mathbf{o}}) |\mathbf{n} \cdot \omega_{\mathbf{i}}|}$
- Compute radiance for scattered photon



You already implemented this for the path tracer

(reuse your code)

Today: Generating the photon map

Today we will focus on the generation of the photon map

- Photon random walk
  1) Select a (point) light source from the scene
  2) Begin the photon random walk with a random direction
  3) Scatter the photon through the scene
  4) **Store the scattered photons in the photon map**
- Repeat until
  - Maximum number of photons in the photon map (map is full)
  - **Maximum number of random walks**

4. Store photons in the photon map

You can use the provided k-d tree:

- Available in Moodle:
  - kd_tree.h
- Implemented in C++
- You need to define some custom classes (see next slides)

4. Store photons in the photon map


```
/*
    Your Photon class implementation, which stores each
    photon walk interaction
*/
class YourPhoton {
    Vec3D position_;    // 3D point of the interaction
    ...
    // It returns the axis i position (x, y or z)
    float position(std::size_t i) const { return position_[i]}
    ...
}
```

Your photon class

```
/*
    An additional struct that allows the KD-Tree to access your photon position
*/
struct PhotonAxisPositition {
    float operator()(const YourPhoton& p, std::size_t i) const {
        return p.position(i);
    }[redacted]
};
```

An auxiliary struct to access the position in your photon class

4. Store photons in the photon map

Graphics and Imaging Lab

Type definition (alias) as a shorter name

```
/*
    The KD-Tree ready to work in 3 dimensions, with YourPhoton s, under a
    brand-new name: YourPhotonMap
*/
using YourPhotonMap = nn::KDTree<YourPhoton,3,PhotonAxisPosition>;
```

4. Store photons in the photon map

You need two functions:

- 1. Construction of the k-d tree

```
/*
    Example function to generate the photon map with the given photons
*/
YourPhotonMap generation_of_photon_map(...){
    std::list<YourPhoton> photons = ...;        // Create a list of photons
    map = YourPhotonMap(photons, PhotonAxisPosition())
    return map
}
```

4. Store photons in the photon map

Graphics and Imaging Lab

You need two functions:

- 1. Construction of the k-d tree

```
map = YourPhotonMap(photons, PhotonAxisPosition())
```

- 2. Search for nearest neighbors in the k-d tree

```
// nearest is the nearest photons returned by the KDTree
auto nearest = map.nearest_neighbors(position,
                                     nphotons_estimate,
                                     radius_estimate)
```

With the provided source you can find an example of usage:

- *kd_tree_example.cpp*

4. Store photons in the photon map

```
/*
    Example method to search for the nearest neighbors of the photon map
*/
void search_nearest(YourPhotonMap map, ...){
    // Position to look for the nearest photons
    Vec3D query_position = ...;
    // Maximum number of photons to look for
    unsigned long nphotons_estimate = ...;
    // Maximum distance to look for photons
    float radius_estimate = ...;

    // nearest is the nearest photons returned by the KDTree
    auto nearest = map.nearest_neighbors(position,
                                         nphotons_estimate,
                                         radius_estimate)
}
```

4. Store photons in the photon map

Graphics and Imaging Lab

```
/*
    Example method to search for the nearest neighbors of the photon map
*/
void search_nearest(YourPhotonMap map, ...){
    // Position to look for the nearest photons
    Vec3D query_position = ...;
    // Maximum number of photons to look for
    unsigned long nphotons_estimate = ...;
    // Maximum distance to look for photons
    float radius_estimate = ...;

    // nearest is the nearest photons returned by the KDTree
    auto nearest = map.nearest_neighbors(query_position,
                                         nphotons_estimate,
                                         radius_estimate)
}
```

Careful!

Your Vec3D class must implement operator[]

4. Store photons in the photon map

```
// Position to look for the nearest photons
Vec3D query_position = ...;
```

```
class Vec3D{
private:
    // 3 dimension axis
    float x, y, z;
```

```
public:
    // Overload the [] operator for indexing
    const float& operator[](size_t t) const{
        if (t == 0)
            return x;
        else if (t == 1)
            return y;
        else if (t == 2)
            return z;
        else
            return;
    }
};
```

*For example

4. Store photons in the photon map

To compile our code, you will need C++17. You can use the flag:

-std==c++17

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:

- The generation of a photon map:
  - Select a (point) light source
  - Sampling a direction for light source to start the random walk
  - Photon scattering through the scene (**only diffuse or absorption events**)
  - Storage in the photon map: **k-d tree**
- Recommended deadline: December 4th (moodle: January 11th)
  - If you have more time, you can visualize your photon map:
    - Launch rays from the camera (exactly like path tracing, reuse your code)
    - At each intersection, find the nearest photon in the photon map
    - Use the flux from that photon to paint each pixel of your final image