Path Tracing Workshop
Part 2: Path Tracing

Christoph Peters
Intel Graphics Research Organization

Recap

ShaderToy

Runs a full-viewport fragment shader in WebGL

Program that runs once per pixel to compute its color

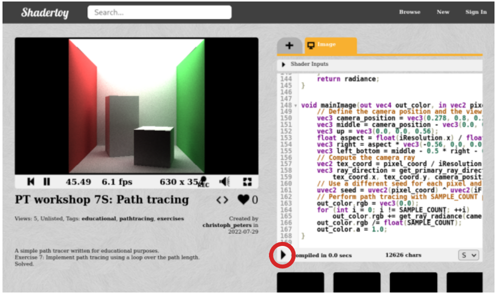
In each exercise you complete 1 function in a ShaderToy (`// TODO`)

Exercise N+1 has a reference solution for exercise N (no peeking)

To change the code, just type

To recompile/run, click play

To save, copy your code to a text file



ShaderToy

Runs a full-viewport fragment shader in WebGL

Program that runs once per pixel to compute its color

In each exercise you complete 1 function in a ShaderToy (`// TODO`)

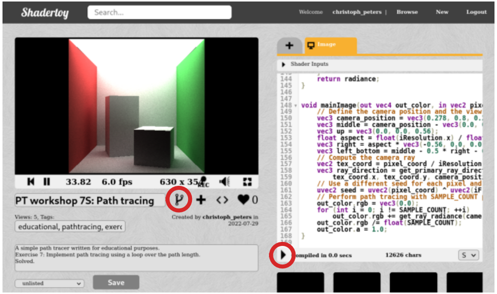
Exercise N+1 has a reference solution for exercise N (no peeking)

To change the code, just type

To recompile/run, click play

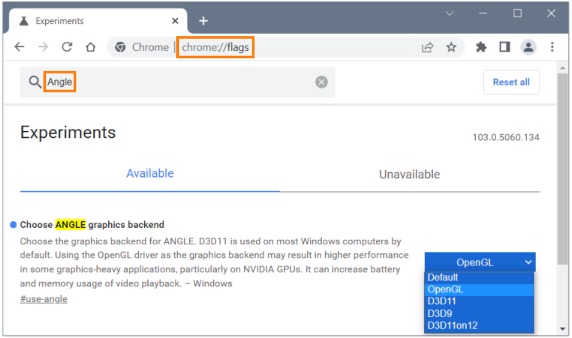
To save, copy your code to a text file

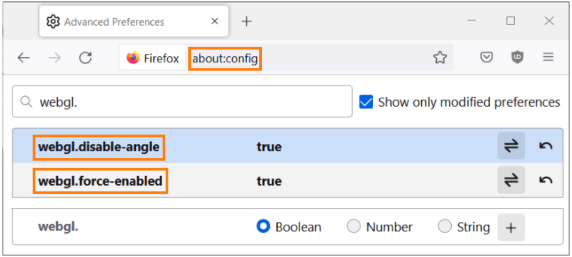
Or create an account and a fork



Proper WebGL config

By default, ANGLE makes big WebGL shaders run slowly on Windows





Then restart Chrome

Then reload ShaderToy tabs

Ray-mesh intersection test

What do we see along a ray?

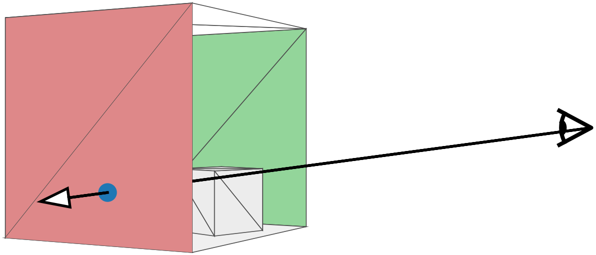
The closest intersected triangle!

Ray tracing finds this closest hit

Foundation of path tracing

Implemented in hardware

But we do it in software



Our goals

Learn rendering basics

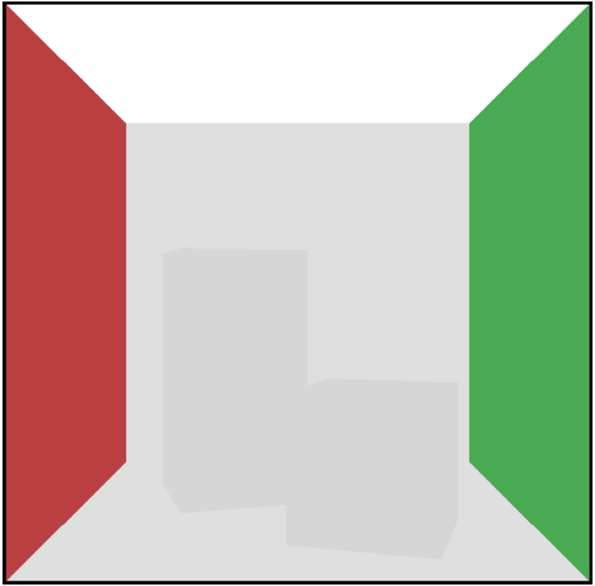
Write a path tracer

In GLSL on ShaderToy

Have fun

Part 1: Ray tracing

Part 2: Path tracing



Our goals

Learn rendering basics

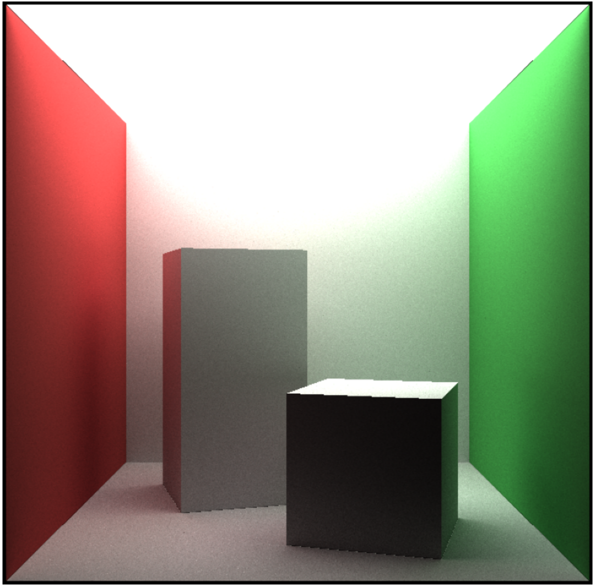
Write a path tracer

In GLSL on ShaderToy

Have fun

Part 1: Ray tracing

Part 2: Path tracing

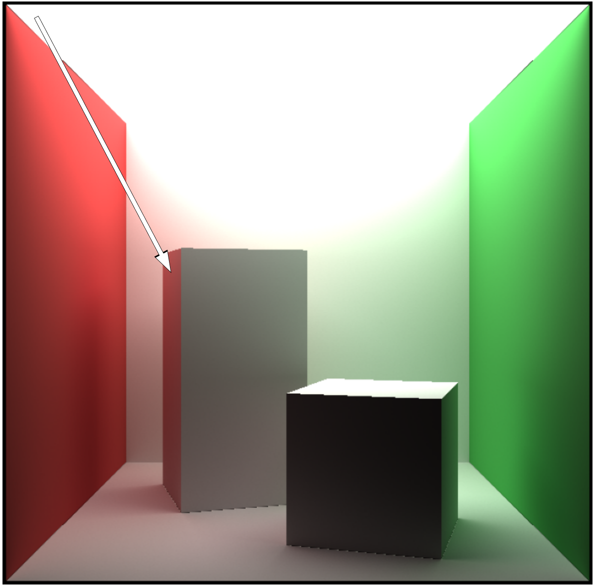


Global illumination

Global illumination

Top of the box is an area light

Surfaces can be lit directly

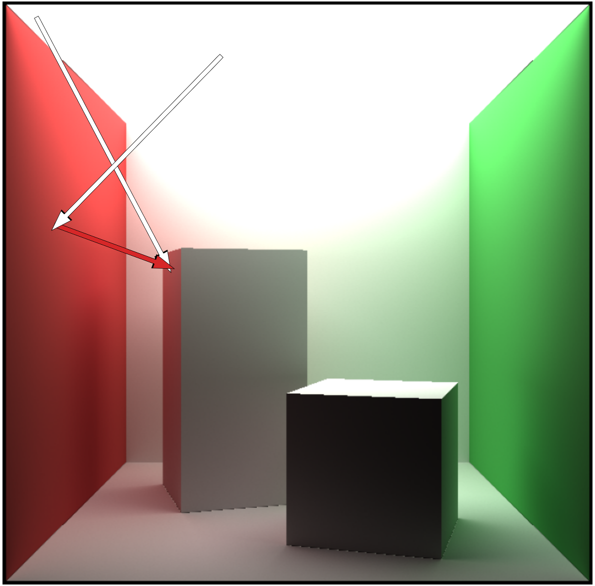


Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly



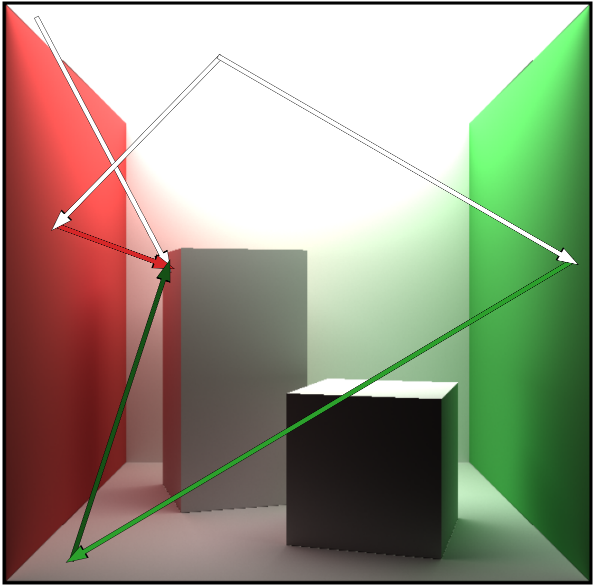
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



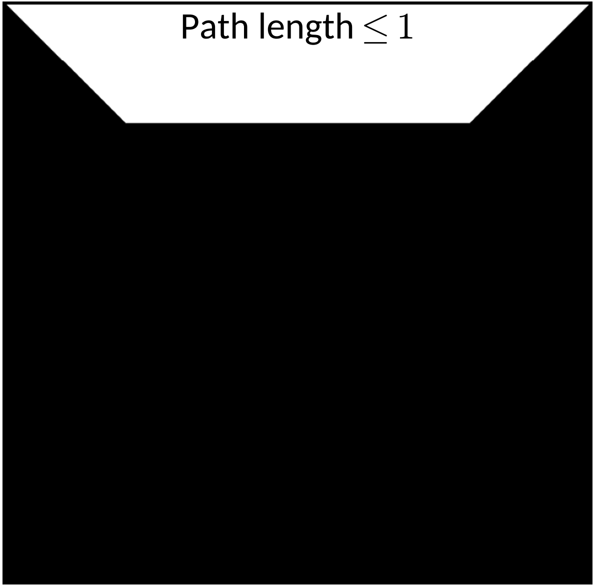
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



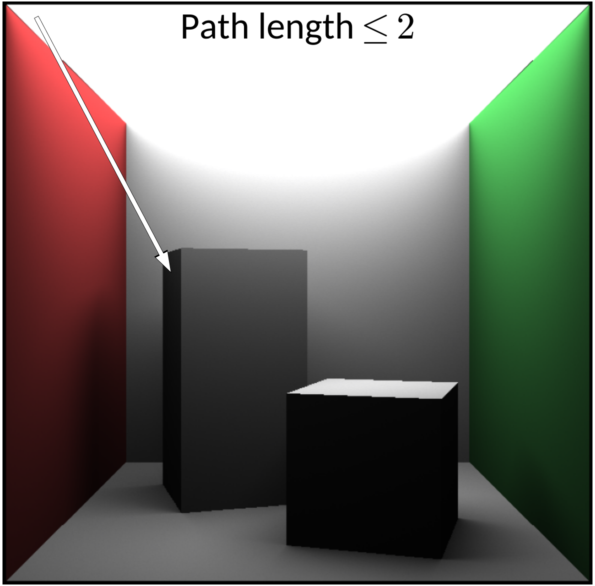
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



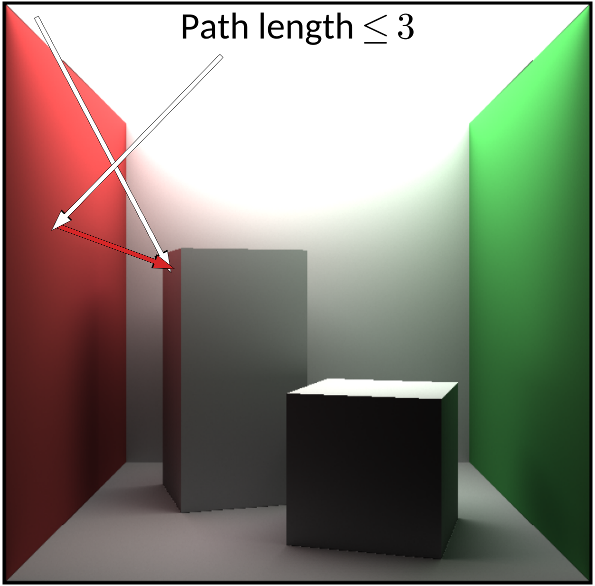
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



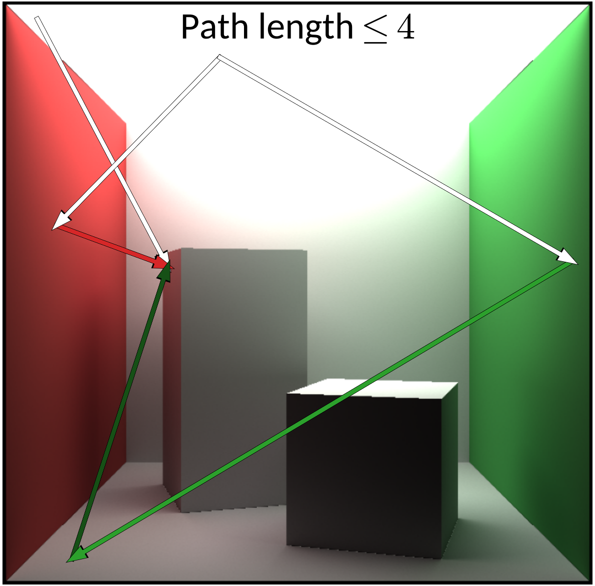
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



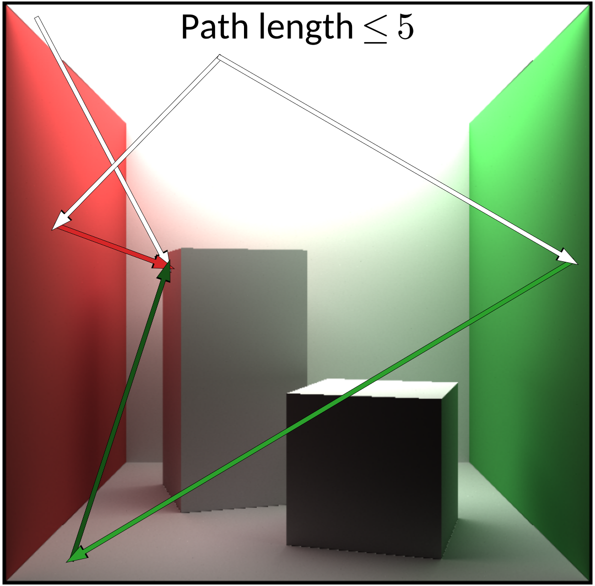
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



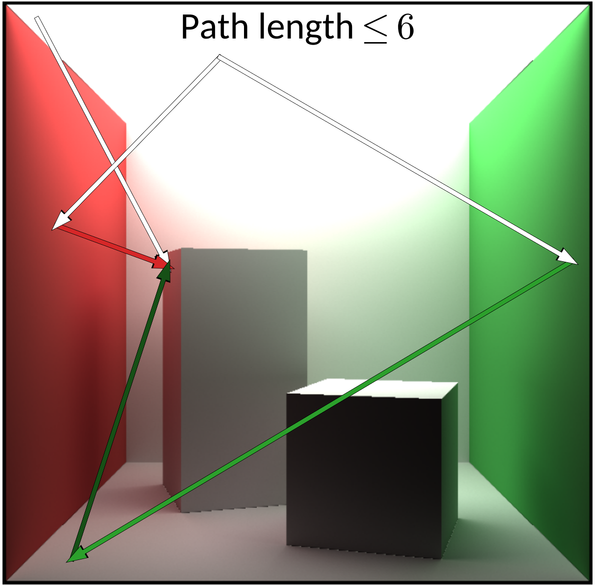
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



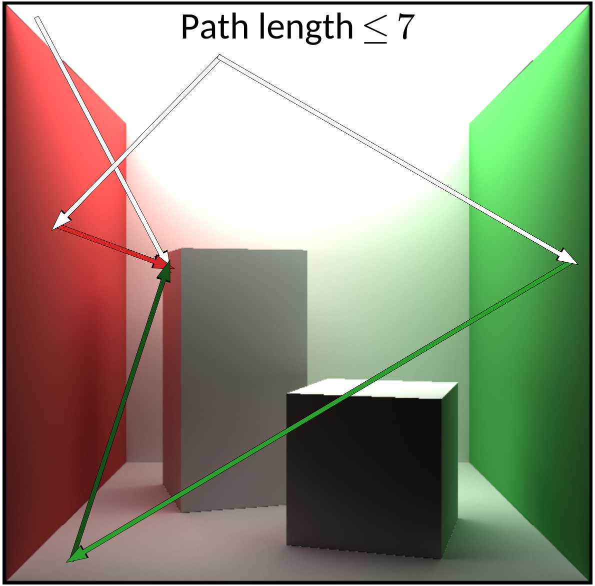
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



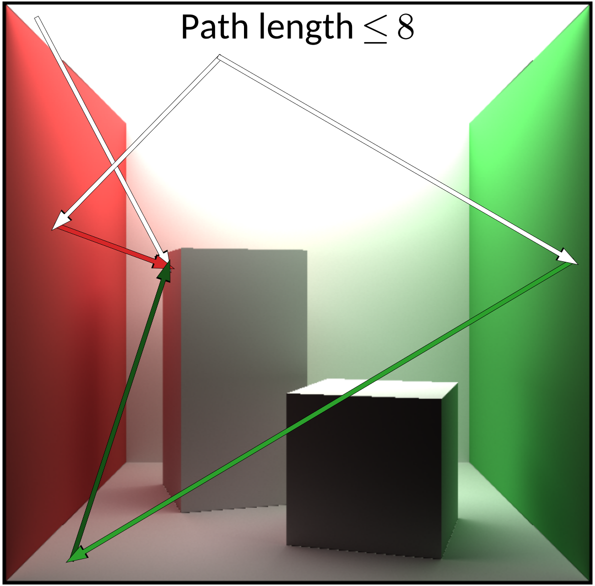
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



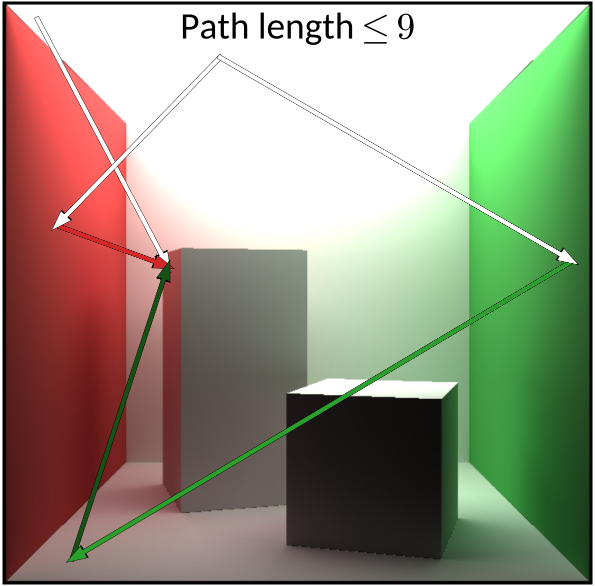
Global illumination

Top of the box is an area light

Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length



Global illumination

Top of the box is an area light

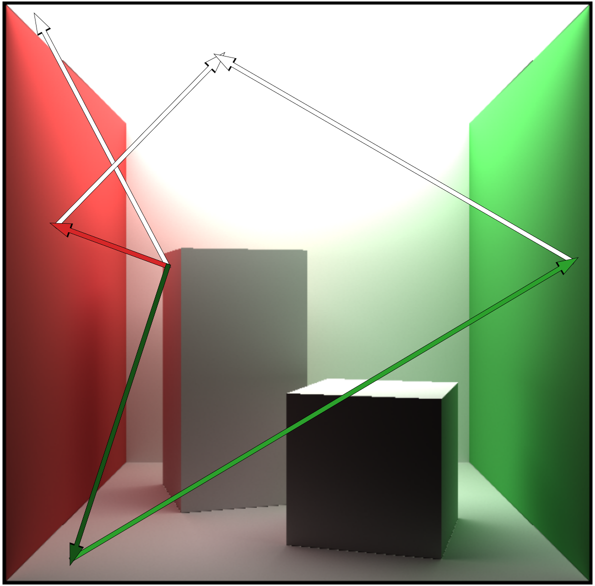
Surfaces can be lit directly

But also indirectly

Via paths of arbitrary length

Path tracing starts at camera

Finds a light when it is lucky



Global illumination

Top of the box is an area light

Surfaces can be lit directly

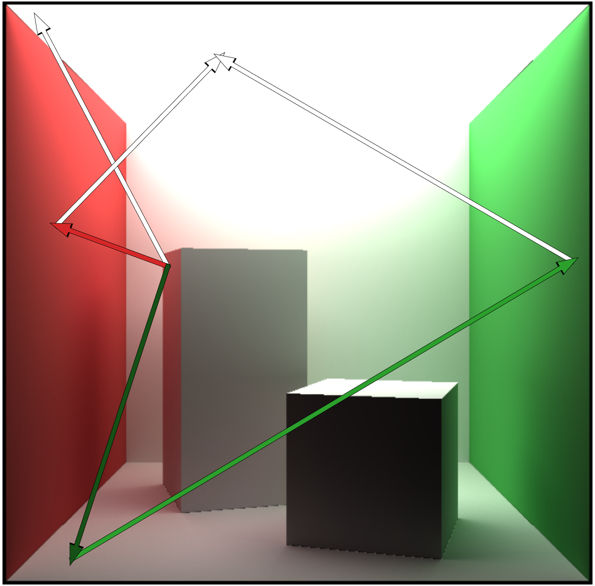
But also indirectly

Via paths of arbitrary length

Path tracing starts at camera

Finds a light when it is lucky

The colors are called "radiance"



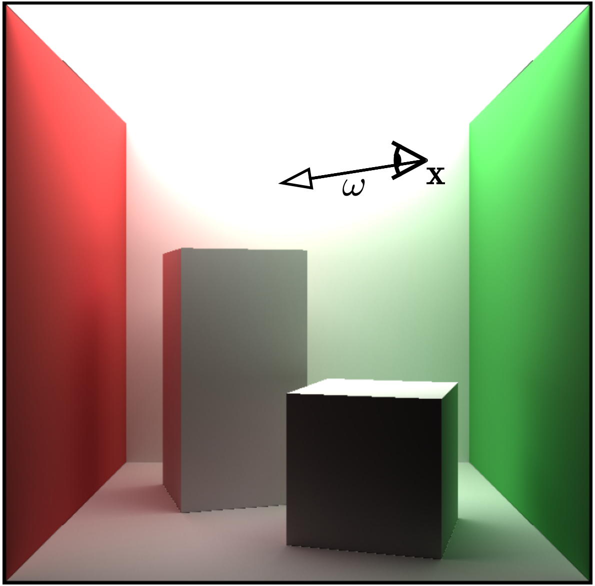
intel.

Radiance

$L(\mathbf{x}, \omega) =$ color for ray $\mathbf{x} + t\omega$

Pixel $=$ radiance for camera ray

Plenoptic function/radiance field



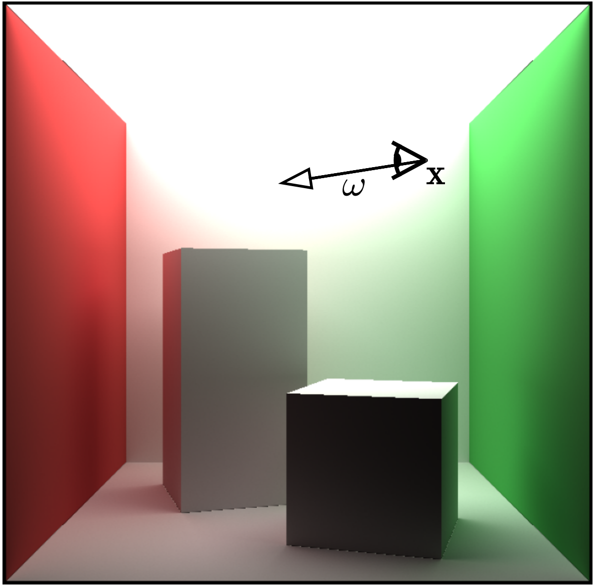
Radiance

$L(\mathbf{x}, \omega) =$ color for ray $\mathbf{x} + t\omega$

Pixel $=$ radiance for camera ray

Plenoptic function/radiance field

Constant along rays in vacuum



Radiance

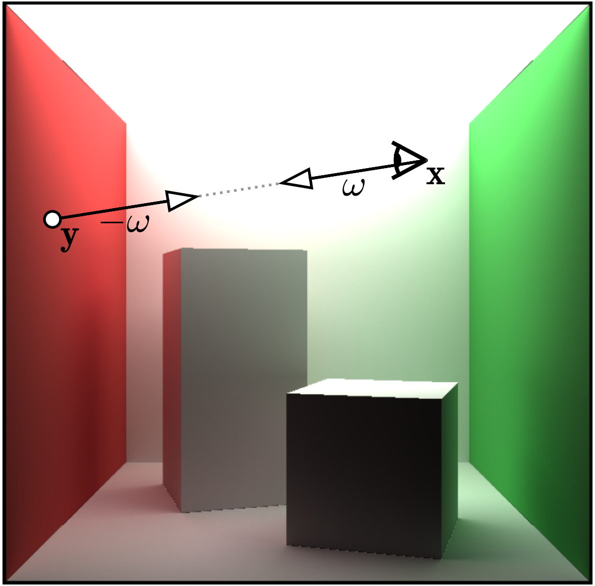
$L(\mathbf{x}, \omega) =$ color for ray $\mathbf{x} + t\omega$

Pixel $=$ radiance for camera ray

Plenoptic function/radiance field

Constant along rays in vacuum

$L(\mathbf{y}, \omega) = L(\mathbf{x}, \omega)$



Radiance

$L(\mathbf{x}, \omega) =$ color for ray $\mathbf{x} + t\omega$

Pixel $=$ radiance for camera ray

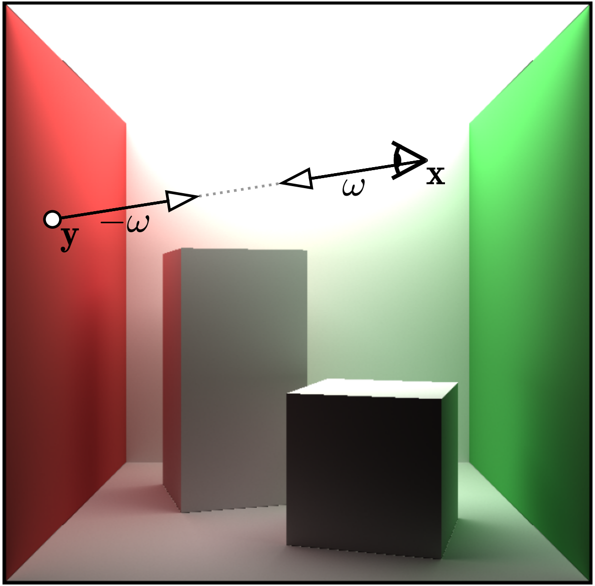
Plenoptic function/radiance field

Constant along rays in vacuum

$L(\mathbf{y}, \omega) = L(\mathbf{x}, \omega)$

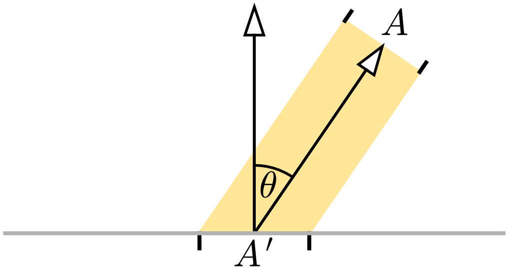
Ray tracing transports radiance

Light transport 😀



Irradiance

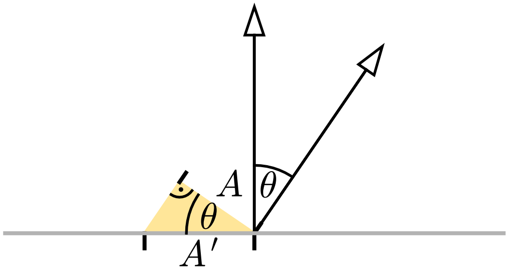
Beam of cross-sectional area A hits surface area A'



Irradiance

Beam of cross-sectional area A hits surface area A'

$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{A}{A'}$$

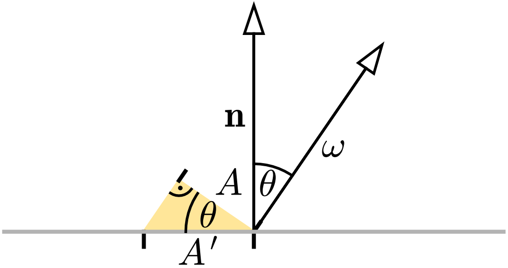


Irradiance

Beam of cross-sectional area A hits surface area A'

$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{A}{A'} = \mathbf{n} \cdot \omega$$

Because we ensure $\|\mathbf{n}\| = \|\omega\| = 1$



Irradiance

Beam of cross-sectional area A hits surface area A'

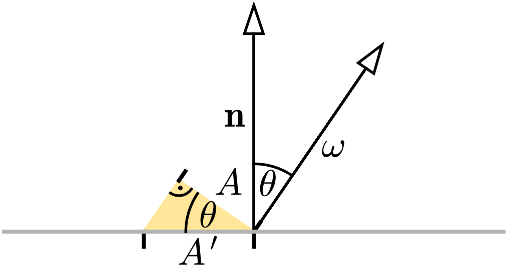
$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{A}{A'} = \mathbf{n} \cdot \omega$$

Because we ensure $\|\mathbf{n}\| = \|\omega\| = 1$

Irradiance gathers all light at point $\mathbf{x}$

Weighted integral over radiance:

$$E(\mathbf{x}, \mathbf{n}) = \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n} \cdot \omega \, \mathrm{d}\omega$$



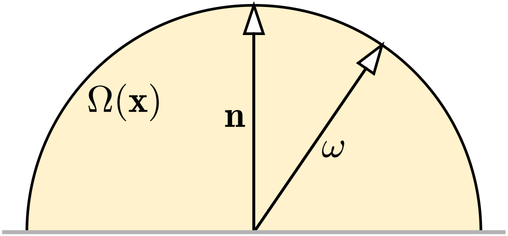
Irradiance

Beam of cross-sectional area A hits surface area A'

$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{A}{A'} = \mathbf{n} \cdot \omega$$

Because we ensure $\|\mathbf{n}\| = \|\omega\| = 1$

Irradiance gathers all light at point $\mathbf{x}$



Weighted integral over radiance:

$$E(\mathbf{x}, \mathbf{n}) = \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n} \cdot \omega \, \mathrm{d}\omega$$

Where $\Omega(\mathbf{x}) \subseteq \mathbb{R}^3$ is a hemisphere: $\omega \in \Omega(\mathbf{x}) \;\Leftrightarrow\; \|\omega\| = 1,\ \mathbf{n} \cdot \omega \geq 0$

The rendering equation

$$\boxed{L_o(\mathbf{x})} = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

Result: Outgoing radiance $L_o(\mathbf{x})$ for diffuse surface at $\mathbf{x}$

The rendering equation

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \boxed{\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega}$$

Result: Outgoing radiance $L_o(\mathbf{x})$ for diffuse surface at $\mathbf{x}$

Compute incoming irradiance $E(\mathbf{x}, \mathbf{n}(\mathbf{x}))$, i.e. total light reaching $\mathbf{x}$

The rendering equation

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \boxed{\frac{a(\mathbf{x})}{\pi}} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

Result: Outgoing radiance $L_o(\mathbf{x})$ for diffuse surface at $\mathbf{x}$

Compute incoming irradiance $E(\mathbf{x}, \mathbf{n}(\mathbf{x}))$, i.e. total light reaching $\mathbf{x}$

Multiply by the surface color $a(\mathbf{x})$ (component-wise)

The rendering equation

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\boxed{\pi}} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

Result: Outgoing radiance $L_o(\mathbf{x})$ for diffuse surface at $\mathbf{x}$

Compute incoming irradiance $E(\mathbf{x}, \mathbf{n}(\mathbf{x}))$, i.e. total light reaching $\mathbf{x}$

Multiply by the surface color $a(\mathbf{x})$ (component-wise)

Divide by π to ensure energy conservation

The rendering equation

$$L_o(\mathbf{x}) = \boxed{L_e(\mathbf{x})} + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \,\mathrm{d}\omega$$

Result: Outgoing radiance $L_o(\mathbf{x})$ for diffuse surface at $\mathbf{x}$

Compute incoming irradiance $E(\mathbf{x}, \mathbf{n}(\mathbf{x}))$, i.e. total light reaching $\mathbf{x}$

Multiply by the surface color $a(\mathbf{x})$ (component-wise)

Divide by π to ensure energy conservation

Add light emitted at $\mathbf{x}$ (0 if there is no light source at $\mathbf{x}$)

Mesh representation

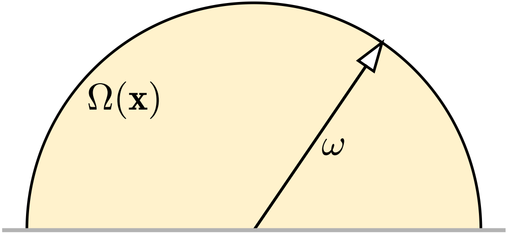
$\mathbf{x}$ $\mathbf{n}(\mathbf{x})$ $a(\mathbf{x})$ $L_e(\mathbf{x})$

```
// A triangle along with some shading parameters
struct triangle_t {
    // The positions of the three vertices (v_0, v_1, v_2)
    vec3 positions[3];
    // A vector of length 1, orthogonal to the triangle (n)
    vec3 normal;
    // The albedo of the triangle (i.e. the fraction of
    // red/green/blue light that gets reflected) (a)
    vec3 color;
    // The radiance emitted by the triangle (for light sources) (L_e)
    vec3 emission;
};
```

The rendering equation: Challenges

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

We have to integrate over $\Omega(\mathbf{x})$

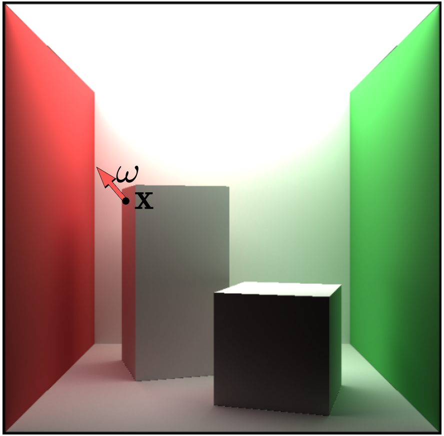


The rendering equation: Challenges

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

We have to integrate over $\Omega(\mathbf{x})$

We need $L(\mathbf{x}, \omega)$



The rendering equation: Challenges

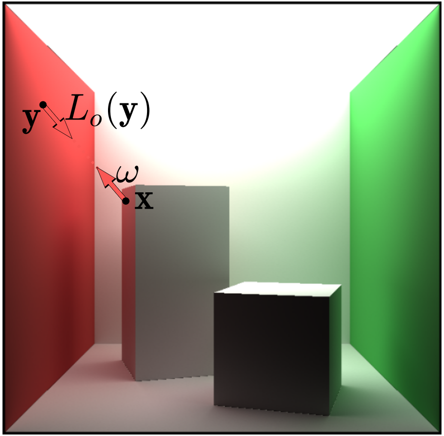
$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \,\mathrm{d}\omega$$

We have to integrate over $\Omega(\mathbf{x})$

We need $L(\mathbf{x}, \omega)$

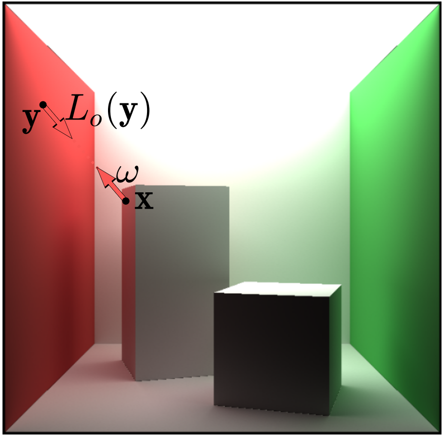
$$\mathbf{y} = \mathrm{ray\_intersection}(\mathbf{x}, \omega) = \mathbf{x} + t\omega$$

$$L(\mathbf{x}, \omega) = L_o(\mathbf{y})$$



The rendering equation: Challenges

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \,\mathrm{d}\omega$$



We have to integrate over $\Omega(\mathbf{x})$

We need $L(\mathbf{x}, \omega)$

$$\mathbf{y} = \mathrm{ray\_intersection}(\mathbf{x}, \omega) = \mathbf{x} + t\omega$$

$$L(\mathbf{x}, \omega) = L_o(\mathbf{y})$$

So we need $L_o(\mathbf{y})$ to compute $L_o(\mathbf{x})$

Monte Carlo integration

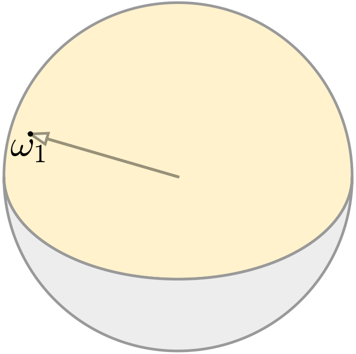
Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

Instead, pick $\omega_1 \in \boldsymbol{\Omega}(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \,\mathrm{d}\omega$$

$$\approx 2\pi\, L(\mathbf{x}, \omega_1)\, \mathbf{n}(\mathbf{x}) \cdot \omega_1$$



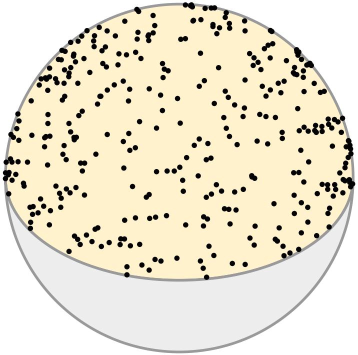
Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

Instead, pick $\omega_1, \ldots, \omega_N \in \boldsymbol{\Omega}(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \,\mathrm{d}\omega$$

$$\approx 2\pi \, \frac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$



Equal for $N \to \infty$ (with 100% probability)

Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

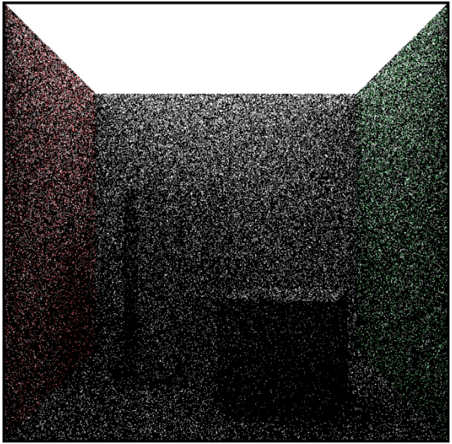
Instead, pick $\omega_1, \ldots, \omega_N \in \mathbf{\Omega}(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

$$\approx 2\pi \, \tfrac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$

Equal for $N \to \infty$ (with 100% probability)

Error is zero-mean noise



$N = 1$

Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

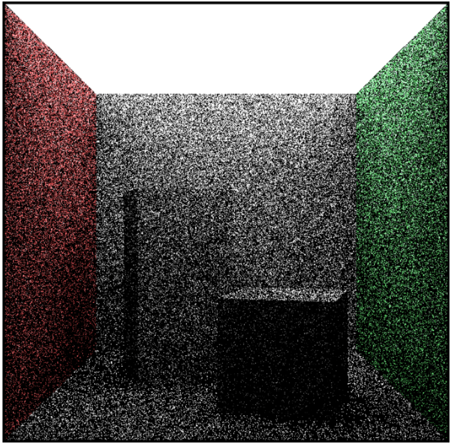
Instead, pick $\omega_1, \ldots, \omega_N \in \Omega(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

$$\approx 2\pi \, \tfrac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$

Equal for $N \to \infty$ (with 100% probability)

Error is zero-mean noise



$N = 2$

Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

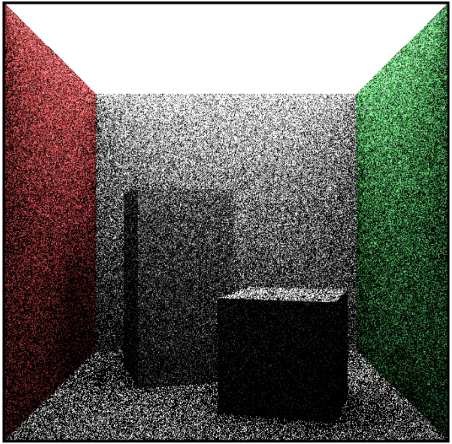
Instead, pick $\omega_1, \ldots, \omega_N \in \Omega(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

$$\approx 2\pi \, \tfrac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$

Equal for $N \to \infty$ (with 100% probability)

Error is zero-mean noise



$N = 4$

intel.

Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

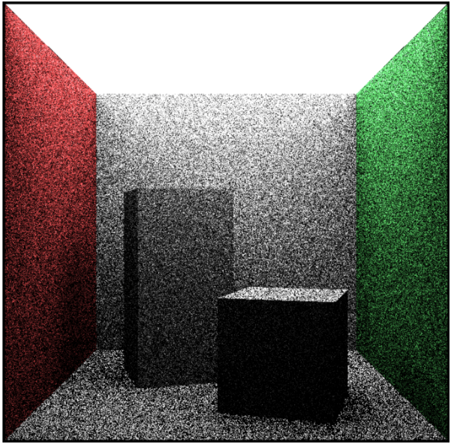
Instead, pick $\omega_1, \ldots, \omega_N \in \Omega(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \,\mathrm{d}\omega$$

$$\approx 2\pi \, \tfrac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$

Equal for $N \to \infty$ (with 100% probability)

Error is zero-mean noise



$N = 8$

Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

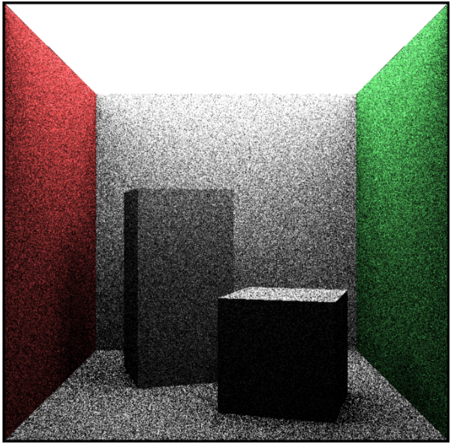
Instead, pick $\omega_1, \ldots, \omega_N \in \boldsymbol{\Omega}(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

$$\approx 2\pi \, \tfrac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$

Equal for $N \to \infty$ (with 100% probability)

Error is zero-mean noise



$N = 16$

Monte Carlo integration

We cannot integrate over ∞ many $\omega \in \Omega(\mathbf{x})$ exactly

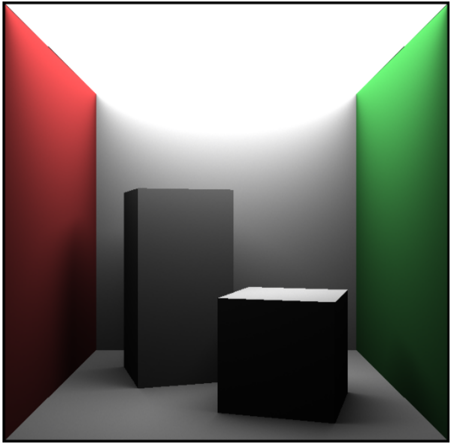
Instead, pick $\omega_1, \ldots, \omega_N \in \boldsymbol{\Omega}(\mathbf{x})$ at random

$$\int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

$$\approx 2\pi \, \tfrac{1}{N} \sum_{j=1}^{N} L(\mathbf{x}, \omega_j)\, \mathbf{n}(\mathbf{x}) \cdot \omega_j$$

Equal for $N \to \infty$ (with 100% probability)

Error is zero-mean noise



$N = 2048$

Non-uniform sphere sampling

Random generator gives uniform $u_0, u_1 \in [0, 1)$

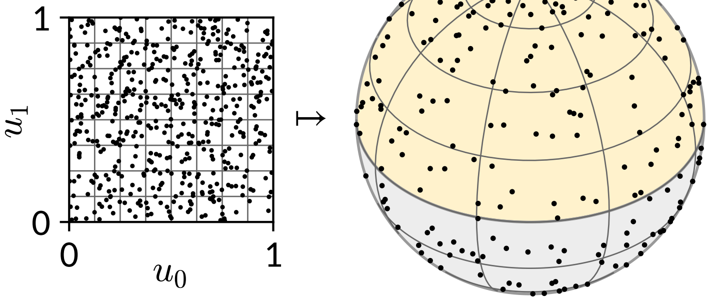
Map to sphere with spherical coordinates:

$$\varphi = 2\pi u_0, \;\; \theta = \pi u_1$$

$$\omega_{\mathrm{x}} = \cos(\varphi)\,\sin(\theta)$$

$$\omega_{\mathrm{y}} = \sin(\varphi)\,\sin(\theta)$$

$$\omega_{\mathrm{z}} = \cos(\theta)$$



Problem: Too many samples at the top

Uniform sphere sampling

Pick $\omega_\mathrm{z} \in [-1, 1)$ uniformly:

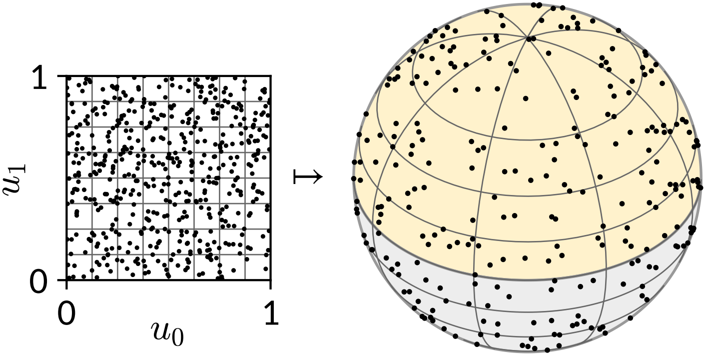
$$\omega_\mathrm{z} = 2u_1 - 1$$

$$\varphi = 2\pi u_0$$

$$\omega_\mathrm{x} = \cos(\varphi)\sqrt{1 - \omega_\mathrm{z}^2}$$

$$\omega_\mathrm{y} = \sin(\varphi)\sqrt{1 - \omega_\mathrm{z}^2}$$

Looks right



Derivation in chapter 13.6.1 of pbr-book.org

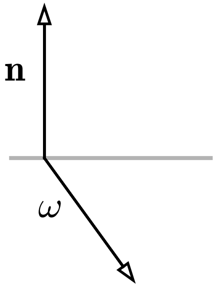
Uniform hemisphere sampling

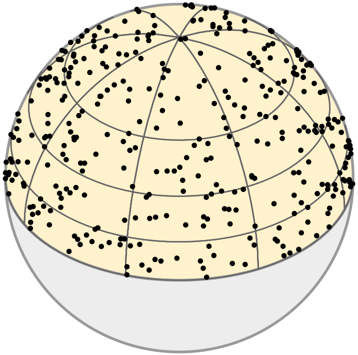
Directions with $\mathbf{n} \cdot \omega < 0$ contribute nothing

Start with ω on the sphere

Mirror if $\mathbf{n} \cdot \omega < 0$:

$$\omega' = \omega - 2\,(\mathbf{n} \cdot \omega)\,\mathbf{n}$$





Uniform hemisphere sampling

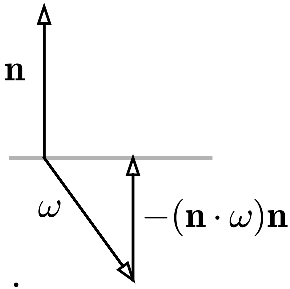
Directions with $\mathbf{n} \cdot \omega < 0$ contribute nothing

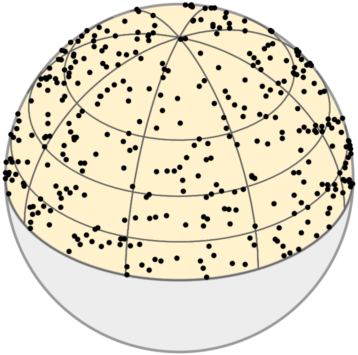
Start with ω on the sphere

Mirror if $\mathbf{n} \cdot \omega < 0$:

$$\omega' = \omega - 2\,(\mathbf{n} \cdot \omega)\,\mathbf{n}$$

i.e. subtract $\mathbf{n}$-component twice





Uniform hemisphere sampling

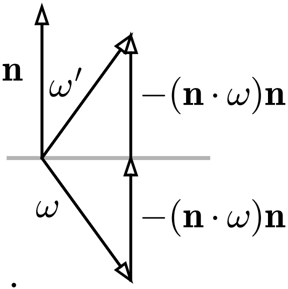
Directions with $\mathbf{n} \cdot \omega < 0$ contribute nothing

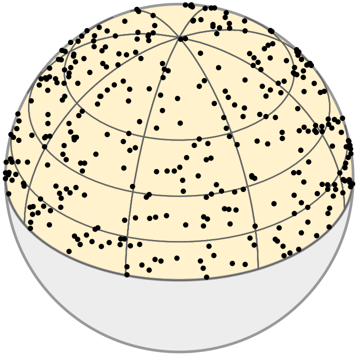
Start with ω on the sphere

Mirror if $\mathbf{n} \cdot \omega < 0$:

$\omega' = \omega - 2\,(\mathbf{n} \cdot \omega)\,\mathbf{n}$

i.e. subtract $\mathbf{n}$-component twice





Exercise 4: Uniform sphere sampling

Complete `sample_sphere()`

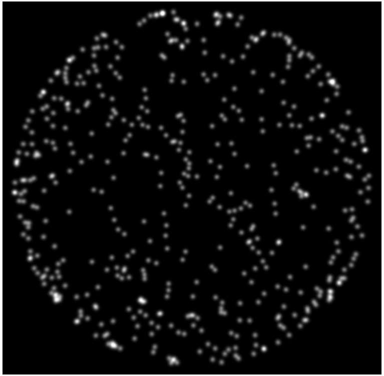
Inputs: Uniform $u_0, u_1 \in [0,1)$

Output: Uniform random direction ω

The framework displays 512 samples

Use the formulas discussed 2 slides ago

Use `cos()`, `sin()`, `sqrt()`, `vec3()`



Correct result

shadertoy.com/view/ssKBD3

Exercise 4: Uniform sphere sampling

Complete `sample_sphere()`

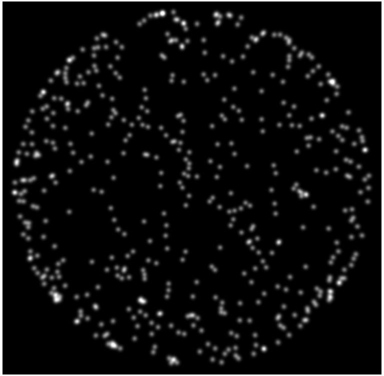
Inputs: Uniform $u_0, u_1 \in [0, 1)$

Output: Uniform random direction ω

The framework displays 512 samples

Use the formulas discussed 2 slides ago

Use `cos()`, `sin()`, `sqrt()`, `vec3()`



Correct result

shadertoy.com/view/ssKBD3

intel

Exercise 5: Uniform hemisphere sampling

Complete `sample_hemisphere()`

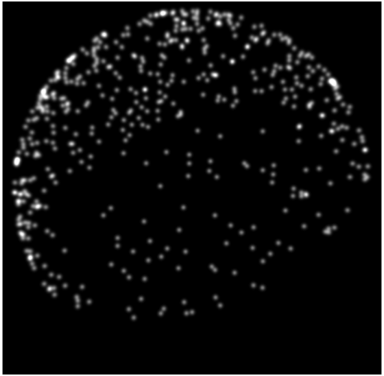
Inputs: Uniform $u_0, u_1 \in [0, 1)$, normal $\mathbf{n}(\mathbf{x})$

Output: Uniform random direction $\omega \in \Omega(\mathbf{x})$

The framework displays 512 samples

Use the formulas discussed 2 slides ago

Use `if`, `dot()`, `*`, `-`



Correct result

shadertoy.com/view/7sKBD3

Exercise 5: Uniform hemisphere sampling

Complete `sample_hemisphere()`

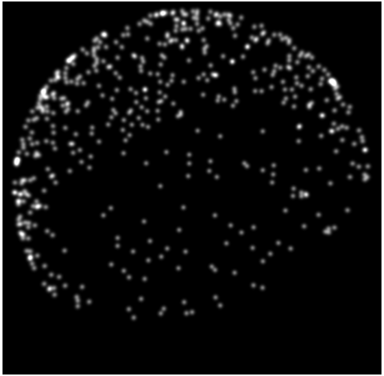
Inputs: Uniform $u_0, u_1 \in [0, 1)$, normal $\mathbf{n}(\mathbf{x})$

Output: Uniform random direction $\omega \in \Omega(\mathbf{x})$

The framework displays 512 samples

Use the formulas discussed 2 slides ago

Use `if`, `dot()`, `*`, `-`



Correct result

shadertoy.com/view/7sKBD3

Pseudorandom number generator

```
// A pseudo-random number generator
// \param seed Numbers that are different for each invocation. Gets updated so
//             that it can be reused.
// \return Two independent, uniform, pseudo-random numbers in [0,1) (u_0, u_1)
vec2 get_random_numbers(inout uvec2 seed) {
    // This is PCG2D: https://jcgt.org/published/0009/03/02/
    seed = 1664525u * seed + 1013904223u;
    seed.x += 1664525u * seed.y;
    seed.y += 1664525u * seed.x;
    seed ^= (seed >> 16u);
    seed.x += 1664525u * seed.y;
    seed.y += 1664525u * seed.x;
    seed ^= (seed >> 16u);
    // Convert to float. The constant here is 2^-32.
    return vec2(seed) * 2.32830643654e-10;
}
```

Pseudorandom number generator

```
// A pseudo-random number generator
// \param seed Numbers that are different for each invocation. Gets updated so
//             that it can be reused.
// \return Two independent, uniform, pseudo-random numbers in [0,1) (u_0, u_1)
vec2 get_random_numbers(inout uvec2 seed) {
    // ...
}


// Use a different seed for each pixel and each frame
uvec2 seed = uvec2(pixel_coord) ^ uvec2(iFrame << 16);
// This gives us 2 uniform random numbers in [0,1)
vec2 rands_0 = get_random_numbers(seed);
// These are different random numbers because seed has changed
vec2 rands_1 = get_random_numbers(seed);
```

intel

Exercise 6: Direct illumination

Complete `compute_direct_illumination()`

Inputs: A triangle and a point $\mathbf{x}$ on it

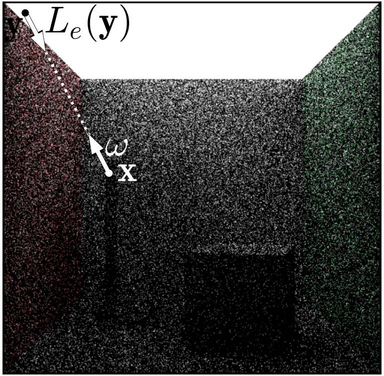
Output: Radiance (emission + direct illum.)

Use $N = 1$ random samples $\omega \in \Omega(\mathbf{x})$

Trace ray $\mathbf{x}, \omega$ to find $L_e(\mathbf{y})$ at hit $\mathbf{y}$

Compute: $L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} 2\pi\ L_e(\mathbf{y})\ \mathbf{n}(\mathbf{x}) \cdot \omega$

Use `sample_hemisphere()`, `ray_mesh_intersection()`



Correct result
`SAMPLE_COUNT=1`

shadertoy.com/view/sdVBD3

Exercise 6: Direct illumination

Complete `compute_direct_illumination()`

Inputs: A triangle and a point $\mathbf{x}$ on it

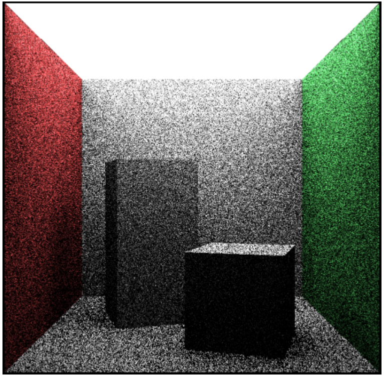
Output: Radiance (emission + direct illum.)

Use $N = 1$ random samples $\omega \in \Omega(\mathbf{x})$

Trace ray $\mathbf{x}, \omega$ to find $L_e(\mathbf{y})$ at hit $\mathbf{y}$

Compute: $L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} 2\pi \; L_e(\mathbf{y}) \; \mathbf{n}(\mathbf{x}) \cdot \omega$

Use `sample_hemisphere()`, `ray_mesh_intersection()`



Correct result
`SAMPLE_COUNT=8`

shadertoy.com/view/sdVBD3

Exercise 6: Direct illumination

Complete `compute_direct_illumination()`

Inputs: A triangle and a point $\mathbf{x}$ on it

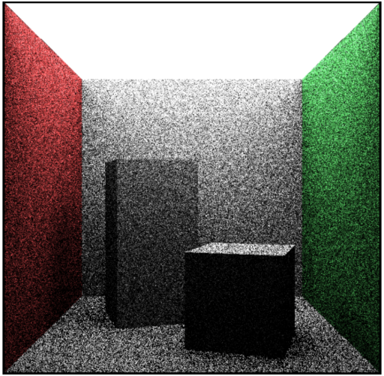
Output: Radiance (emission + direct illum.)

Use $N = 1$ random samples $\omega \in \Omega(\mathbf{x})$

Trace ray $\mathbf{x}, \omega$ to find $L_e(\mathbf{y})$ at hit $\mathbf{y}$

Compute: $L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} 2\pi \; L_e(\mathbf{y}) \; \mathbf{n}(\mathbf{x}) \cdot \omega$

Use `sample_hemisphere()`, `ray_mesh_intersection()`



Correct result
SAMPLE_COUNT=8

shadertoy.com/view/sdVBD3

Path tracing

Path tracing

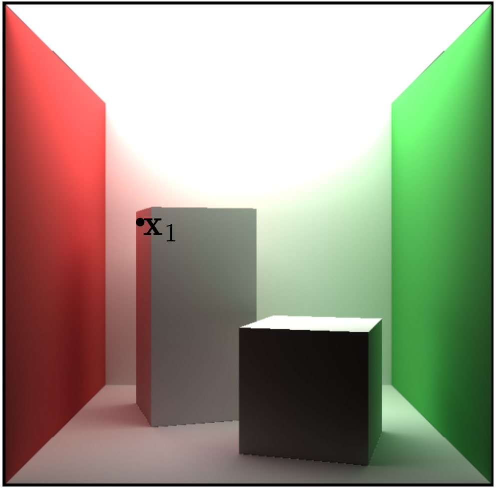
Given camera ray $\mathbf{x}_0, \omega_0$

Want to approximate $L(\mathbf{x}_0, \omega_0)$

$\mathbf{x}_1 = \text{ray\_intersection}(\mathbf{x}_0, \omega_0)$

Monte Carlo estimate with $N = 1$:

$\omega_1 \in \Omega(\mathbf{x}_1)$ random sample



$$L(\mathbf{x}_0, \omega_0) = L_o(\mathbf{x}_1) \approx L_e(\mathbf{x}_1) + \frac{a(\mathbf{x}_1)}{\pi} 2\pi\, L(\mathbf{x}_1, \omega_1)\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1$$

Path tracing

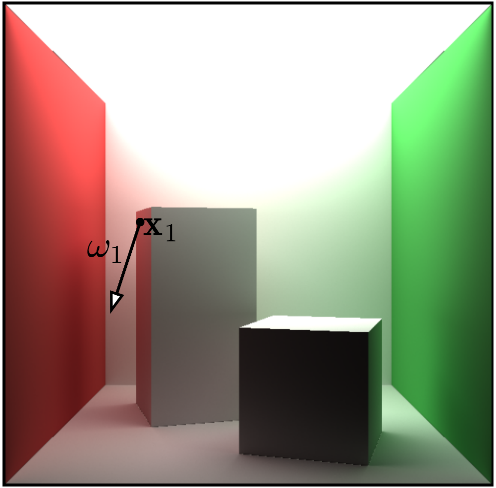
Given camera ray $\mathbf{x}_0, \omega_0$

Want to approximate $L(\mathbf{x}_0, \omega_0)$

$\mathbf{x}_1 = \mathrm{ray\_intersection}(\mathbf{x}_0, \omega_0)$

Monte Carlo estimate with $N = 1$:

$\omega_1 \in \Omega(\mathbf{x}_1)$ random sample



$$L(\mathbf{x}_0, \omega_0) = L_o(\mathbf{x}_1) \approx L_e(\mathbf{x}_1) + \frac{a(\mathbf{x}_1)}{\pi} 2\pi\, L(\mathbf{x}_1, \omega_1)\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1$$

Path tracing

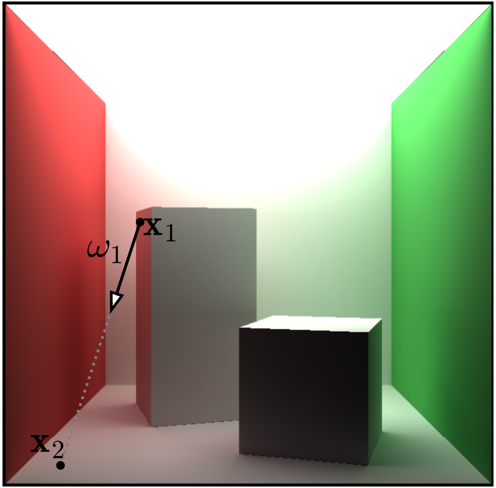
Given ray $\mathbf{x}_1, \omega_1$

Want to approximate $L(\mathbf{x}_1, \omega_1)$

$\mathbf{x}_2 = \mathrm{ray\_intersection}(\mathbf{x}_1, \omega_1)$

Monte Carlo estimate with $N = 1$:

$\omega_2 \in \Omega(\mathbf{x}_2)$ random sample



$$L(\mathbf{x}_1, \omega_1) = L_o(\mathbf{x}_2) \approx L_e(\mathbf{x}_2) + \frac{a(\mathbf{x}_2)}{\pi} 2\pi \, L(\mathbf{x}_2, \omega_2) \, \mathbf{n}(\mathbf{x}_2) \cdot \omega_2$$

Path tracing

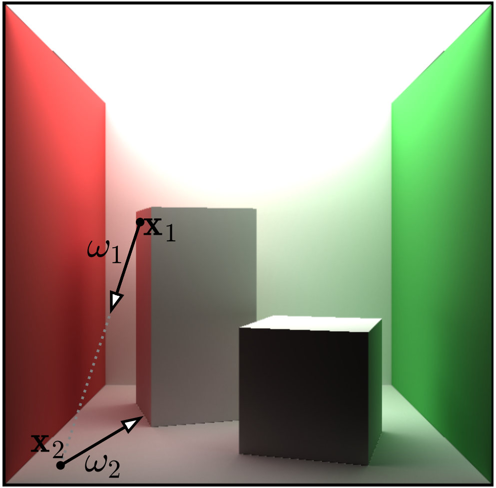
Given ray $\mathbf{x}_1, \omega_1$

Want to approximate $L(\mathbf{x}_1, \omega_1)$

$\mathbf{x}_2 = \mathrm{ray\_intersection}(\mathbf{x}_1, \omega_1)$

Monte Carlo estimate with $N = 1$:

$\omega_2 \in \Omega(\mathbf{x}_2)$ random sample



$$L(\mathbf{x}_1, \omega_1) = L_o(\mathbf{x}_2) \approx L_e(\mathbf{x}_2) + \frac{a(\mathbf{x}_2)}{\pi} 2\pi \, L(\mathbf{x}_2, \omega_2) \, \mathbf{n}(\mathbf{x}_2) \cdot \omega_2$$

intel.

Path tracing

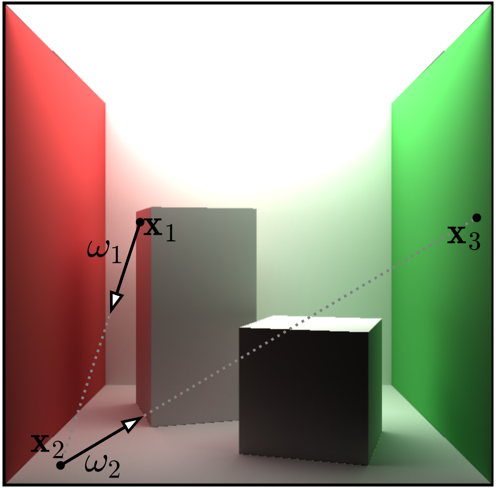
Given ray $\mathbf{x}_2, \omega_2$

Want to approximate $L(\mathbf{x}_2, \omega_2)$

$\mathbf{x}_3 = \text{ray\_intersection}(\mathbf{x}_2, \omega_2)$

Monte Carlo estimate with $N = 1$:

$\omega_3 \in \Omega(\mathbf{x}_3)$ random sample



$$L(\mathbf{x}_2, \omega_2) = L_o(\mathbf{x}_3) \approx L_e(\mathbf{x}_3) + \frac{a(\mathbf{x}_3)}{\pi} 2\pi\, L(\mathbf{x}_3, \omega_3)\, \mathbf{n}(\mathbf{x}_3) \cdot \omega_3$$

Path tracing

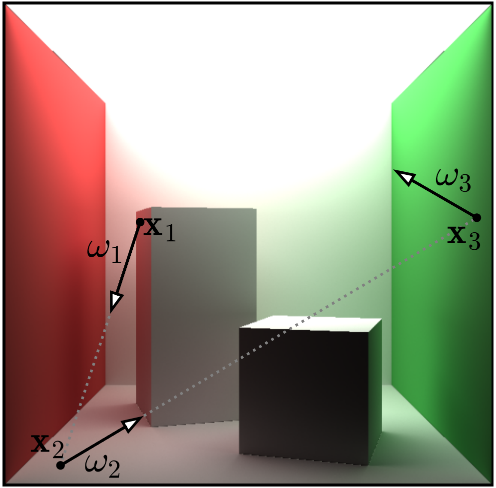
Given ray $\mathbf{x}_2, \omega_2$

Want to approximate $L(\mathbf{x}_2, \omega_2)$

$\mathbf{x}_3 = \mathrm{ray\_intersection}(\mathbf{x}_2, \omega_2)$

Monte Carlo estimate with $N = 1$:

$\omega_3 \in \Omega(\mathbf{x}_3)$ random sample



$$L(\mathbf{x}_2, \omega_2) = L_o(\mathbf{x}_3) \approx L_e(\mathbf{x}_3) + \frac{a(\mathbf{x}_3)}{\pi} 2\pi\, L(\mathbf{x}_3, \omega_3)\, \mathbf{n}(\mathbf{x}_3) \cdot \omega_3$$

Path tracing

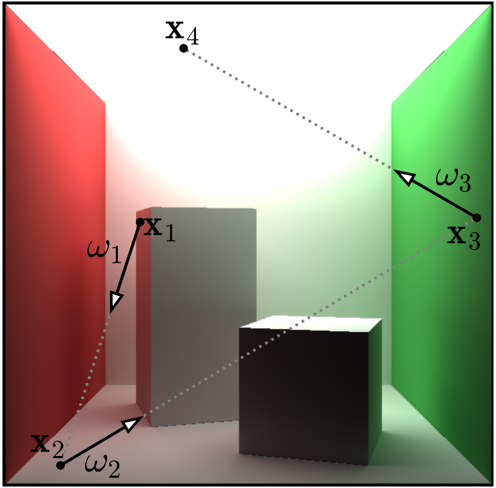
Given ray $\mathbf{x}_3, \omega_3$

Want to approximate $L(\mathbf{x}_3, \omega_3)$

$\mathbf{x}_4 = \text{ray\_intersection}(\mathbf{x}_3, \omega_3)$

Monte Carlo estimate with $N = 1$:

$\omega_4 \in \Omega(\mathbf{x}_4)$ random sample



$$L(\mathbf{x}_3, \omega_3) = L_o(\mathbf{x}_4) \approx L_e(\mathbf{x}_4) + \frac{a(\mathbf{x}_4)}{\pi} 2\pi\, L(\mathbf{x}_4, \omega_4)\, \mathbf{n}(\mathbf{x}_4) \cdot \omega_4$$

Path tracing recursion

GLSL spec: "Static and dynamic recursion is not allowed." 😕

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed." 😕

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed." 😕

$$L(\mathbf{x}_0, \omega_0) \approx L_e(\mathbf{x}_1)$$

intel.

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed." 😕

$$L(\mathbf{x}_0, \omega_0) \approx L_e(\mathbf{x}_1) + \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) L_e(\mathbf{x}_2)$$

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed." 😕

$$
\begin{aligned}
L(\mathbf{x}_0, \omega_0) \approx L_e(\mathbf{x}_1) \\
&+ \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) L_e(\mathbf{x}_2) \\
&+ \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) \left(a(\mathbf{x}_2) 2\, \mathbf{n}(\mathbf{x}_2) \cdot \omega_2\right) L_e(\mathbf{x}_3)
\end{aligned}
$$

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed." 😕

$$
\begin{aligned}
L(\mathbf{x}_0, \omega_0) \approx\ & L_e(\mathbf{x}_1) \\
& + \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) L_e(\mathbf{x}_2) \\
& + \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) \left(a(\mathbf{x}_2) 2\, \mathbf{n}(\mathbf{x}_2) \cdot \omega_2\right) L_e(\mathbf{x}_3) \\
& \vdots
\end{aligned}
$$

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed." 😕

$$
\begin{aligned}
L(\mathbf{x}_0, \omega_0) \approx L_e(\mathbf{x}_1)& \\
&+ \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) L_e(\mathbf{x}_2) \\
&+ \underbrace{\left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) \left(a(\mathbf{x}_2) 2\, \mathbf{n}(\mathbf{x}_2) \cdot \omega_2\right)}_{T_2} L_e(\mathbf{x}_3) \\
&\vdots
\end{aligned}
$$

Path tracing ~~recursion~~ loop

GLSL spec: "Static and dynamic recursion is not allowed."

$$
\begin{aligned}
L(\mathbf{x}_0, \omega_0) \approx\ & L_e(\mathbf{x}_1) \\
& + \left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) L_e(\mathbf{x}_2) \\
& + \underbrace{\left(a(\mathbf{x}_1) 2\, \mathbf{n}(\mathbf{x}_1) \cdot \omega_1\right) \left(a(\mathbf{x}_2) 2\, \mathbf{n}(\mathbf{x}_2) \cdot \omega_2\right)}_{T_2} L_e(\mathbf{x}_3) \\
& \vdots
\end{aligned}
$$

Add emission and update throughput weight T_j in each iteration:

$$
\begin{aligned}
L_{j+1} &= L_j + T_j L_e(\mathbf{x}_{j+1}), & L_0 &= 0 \\
T_{j+1} &= T_j\ a(\mathbf{x}_{j+1}) 2\, \mathbf{n}(\mathbf{x}_{j+1}) \cdot \omega_{j+1}, & T_0 &= 1
\end{aligned}
\qquad j = 0, \ldots
$$

Exercise 7: Path tracing

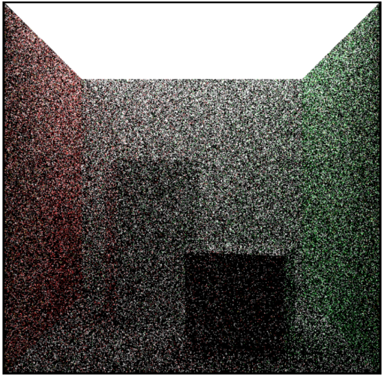
Complete `get_ray_radiance()`

Input: A ray $\mathbf{x}_0, \omega_0$

Output: $L(\mathbf{x}_0, \omega_0)$ (Monte Carlo estimate)

Use a for-loop with `MAX_PATH_LENGTH` iterations

- Trace ray $\mathbf{x}_j, \omega_j$, break if it hits nothing
- Update the ray origin: $\mathbf{x}_{j+1} = \mathbf{x}_j + t_j\omega_j$
- Add $T_j L_e(\mathbf{x}_{j+1})$ to the radiance
- Sample a direction $\omega_{j+1} \in \Omega(\mathbf{x}_{j+1})$
- Mul. throughput by $a(\mathbf{x}_{j+1}) 2\,\mathbf{n}(\mathbf{x}_{j+1}) \cdot \omega_{j+1}$



Correct result
`SAMPLE_COUNT=1`

shadertoy.com/view/flcczr

Exercise 7: Path tracing

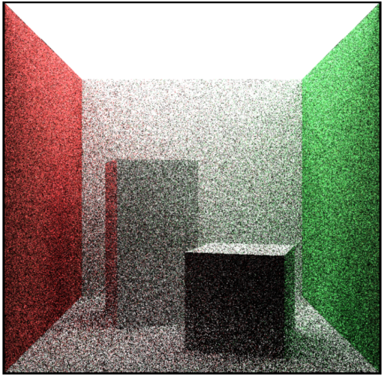
Complete `get_ray_radiance()`

Input: A ray $\mathbf{x}_0, \omega_0$

Output: $L(\mathbf{x}_0, \omega_0)$ (Monte Carlo estimate)

Use a for-loop with `MAX_PATH_LENGTH` iterations

- Trace ray $\mathbf{x}_j, \omega_j$, break if it hits nothing
- Update the ray origin: $\mathbf{x}_{j+1} = \mathbf{x}_j + t_j\omega_j$
- Add $T_j L_e(\mathbf{x}_{j+1})$ to the radiance
- Sample a direction $\omega_{j+1} \in \Omega(\mathbf{x}_{j+1})$
- Mul. throughput by $a(\mathbf{x}_{j+1}) 2\, \mathbf{n}(\mathbf{x}_{j+1}) \cdot \omega_{j+1}$



Correct result
`SAMPLE_COUNT=8`

shadertoy.com/view/flcczr

Exercise 7: Path tracing

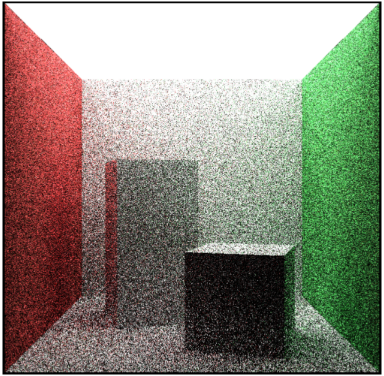
Complete `get_ray_radiance()`

Input: A ray $\mathbf{x}_0, \omega_0$

Output: $L(\mathbf{x}_0, \omega_0)$ (Monte Carlo estimate)

Use a for-loop with `MAX_PATH_LENGTH` iterations

- Trace ray $\mathbf{x}_j, \omega_j$, break if it hits nothing
- Update the ray origin: $\mathbf{x}_{j+1} = \mathbf{x}_j + t_j \omega_j$
- Add $T_j L_e(\mathbf{x}_{j+1})$ to the radiance
- Sample a direction $\omega_{j+1} \in \Omega(\mathbf{x}_{j+1})$
- Mul. throughput by $a(\mathbf{x}_{j+1}) 2\, \mathbf{n}(\mathbf{x}_{j+1}) \cdot \omega_{j+1}$



Correct result
`SAMPLE_COUNT=8`

shadertoy.com/view/flcczr

intel.

Progressive rendering

Taking more samples is the outer-most loop:

```
out_color.rgb = vec3(0.0);
for (int i = 0; i != SAMPLE_COUNT; ++i)
    out_color.rgb += get_ray_radiance(camera_position, ray_direction, seed);
out_color.rgb /= float(SAMPLE_COUNT);
```

A large sample count hangs/crashes your browser

Instead, distribute work over frames

ShaderToy technicalities, not an exercise

Keep it running for better images

Path tracing is

General

Predictable

Scalable

Parallelizable

Extendable

Efficient

Path tracing is

General 1 framework for all light transport

Predictable

Scalable

Parallelizable

Extendable

Efficient

Path tracing is

General	1 framework for all light transport
Predictable	Physically correct image + noise
Scalable	
Parallelizable	
Extendable	
Efficient	

Path tracing is

General	1 framework for all light transport
Predictable	Physically correct image + noise
Scalable	Sample count allows tradeoffs
Parallelizable	
Extendable	
Efficient	

Path tracing is

General	1 framework for all light transport
Predictable	Physically correct image + noise
Scalable	Sample count allows tradeoffs
Parallelizable	Across samples or pixels
Extendable	
Efficient	

Path tracing is

General	1 framework for all light transport
Predictable	Physically correct image + noise
Scalable	Sample count allows tradeoffs
Parallelizable	Across samples or pixels
Extendable	To spectral/volumetric/differentiable rendering
Efficient	

Path tracing is

General	1 framework for all light transport
Predictable	Physically correct image + noise
Scalable	Sample count allows tradeoffs
Parallelizable	Across samples or pixels
Extendable	To spectral/volumetric/differentiable rendering
Efficient	When effort is focused on important work

intel.

Path tracing is

General	1 framework for all light transport
Predictable	Physically correct image + noise
Scalable	Sample count allows tradeoffs
Parallelizable	Across samples or pixels
Extendable	To spectral/volumetric/differentiable rendering
Efficient	When effort is focused on important work

The default in offline rendering, the future in real-time rendering

intel.

Faster path tracers

Acceleration structures and traversal

Stratification: Quasi-random numbers that improve convergence

Importance sampling:

- Light sampling, a.k.a. next event estimation
- (Specular) BRDF importance sampling
- Path guiding
- Multiple importance sampling

Spatiotemporal (neural) denoising

Faster path tracers

Acceleration structures and traversal

Stratification: Quasi-random numbers that improve convergence

Importance sampling:

- Light sampling, a.k.a. next event estimation
- (Specular) BRDF importance sampling
- Path guiding
- Multiple importance sampling

Part 3?

Spatiotemporal (neural) denoising

Thanks!

Backup

The rendering equation

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) + \frac{a(\mathbf{x})}{\pi} \int_{\Omega(\mathbf{x})} L(\mathbf{x}, \omega)\, \mathbf{n}(\mathbf{x}) \cdot \omega \, \mathrm{d}\omega$$

$\mathbf{n}(\mathbf{x})$ is the normal vector at $\mathbf{x}$

$a(\mathbf{x})$ is the albedo at $\mathbf{x}$

$L(\mathbf{x}, \omega)$ is incoming radiance at $\mathbf{x}$ from ω

$L_o(\mathbf{x})$ is the outgoing radiance at $\mathbf{x}$

$L_e(\mathbf{x})$ is emitted radiance at $\mathbf{x}$