Ray Tracing for Games

Dr. Jacco Bikker - IGAD/BUAS, Breda, February 6

Welcome!

9

Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation

Ray Tracing for Games

Wednesday 13:00 – 17:00

course intro
LH2
template
Whitted
refactoring
RT-centric games

LAB 1

Thursday 09:00 – 14:00

advanced Whitted
audio, AI & physics
faster Whitted
Heaven7

LAB 2

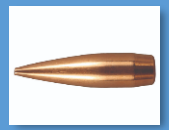
work @ home

End result day 2:

A solid Whitted-style ray tracer, as a basis for subsequent work.

Friday 09:00 – 17:00

optimization
profiling, rules of engagement
threading



LAB 3

SIMD
applied SIMD
SIMD triangle
SIMD AABB

LAB 4

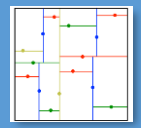
End result day 3:

A 5x faster tracer.

GLOBAL GAME JAM

Monday 09:00 – 17:00

acceleration
grid, BVH, kD-tree
SAH
binning



LAB 5

refitting
top-level BVH
threaded building

LAB 6

End result day 4:

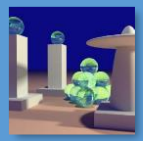
A real-time tracer.

Tuesday 09:00 – 17:00

Monte-Carlo
Cook-style
glossy, AA
area lights, DOF

LAB 7

path tracing



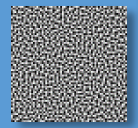
LAB 8

End result day 5:

Cook or Kajiya.

Thursday 09:00 – 17:00

random numbers
stratification
blue noise



LAB 9

importance
sampling
next event
estimation

LAB 10

End result day 6:

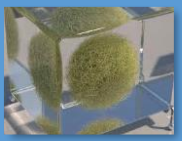
Efficiency.

Friday 09:00 – 17:00

future work

LAB 11

path guiding



LAB 10

End result day 6:

Great product.

Path Tracing

$$L_o(x, \omega_o) = L_E(x, \omega_o) + \int_\Omega f_r(x, \omega_o, \omega_i)\, L_i(x, \omega_i) \cos\theta_i \; d\omega_i$$

The rendering equation allows us to accurately simulate light transport:

- Given a sensible scene description
- Given a sensible camera model
- Ignoring some complex effects.

We evaluate it using Monte Carlo integration:

$$L_o(p, \omega_o) \approx \frac{2\pi}{N} \sum_{i=1}^{N} f_r(p, \omega_o, \Omega_i)\, L_d(p, \Omega_i) \cos\theta_i$$

➔ The result is always an estimate, with variance.

www.mitsuba-renderer.org

Noise in the Real World

Noise in a digital photo can be the result of short exposure.

www.photoreview.com.au/tips/shooting/how-to-control-image-noise

Noise in the Real World

Noise in a digital photo can be the result of short exposure.

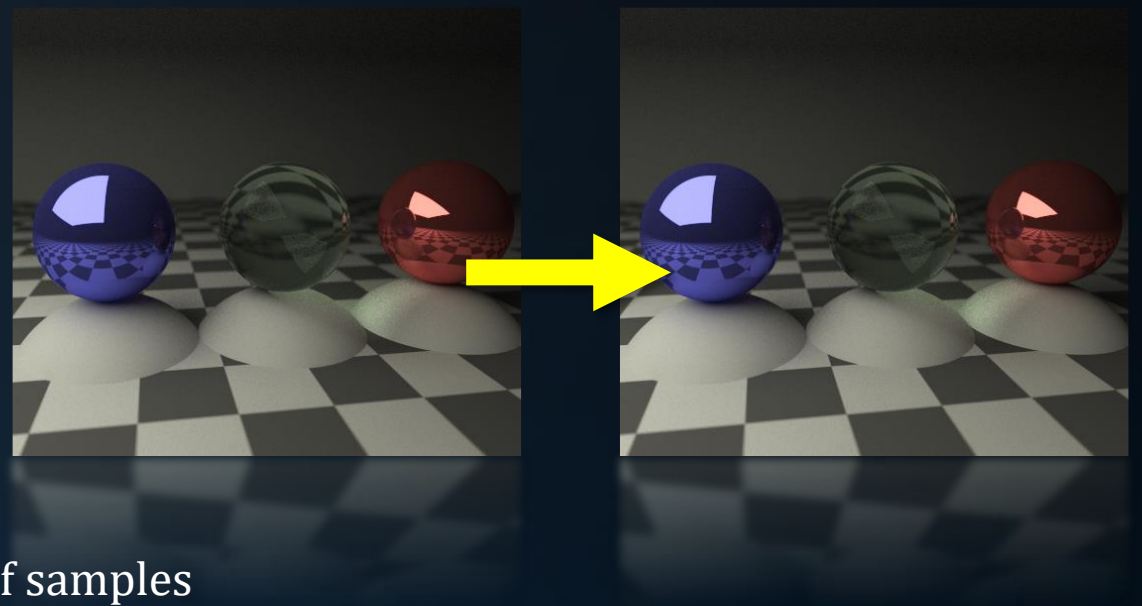
We solve this by lowering our ISO settings, so we gather more photons per pixel.

(that obviously has consequences)

www.photoreview.com.au/tips/shooting/how-to-control-image-noise

Today:

- Stratification
- Blue noise
- Next Event Estimation
- Importance Sampling



Aim:

- to get a better image with the same number of samples
- to increase the efficiency of a path tracer
- to reduce variance in the estimate

Requirement:

- *produce the correct image*

Agenda:

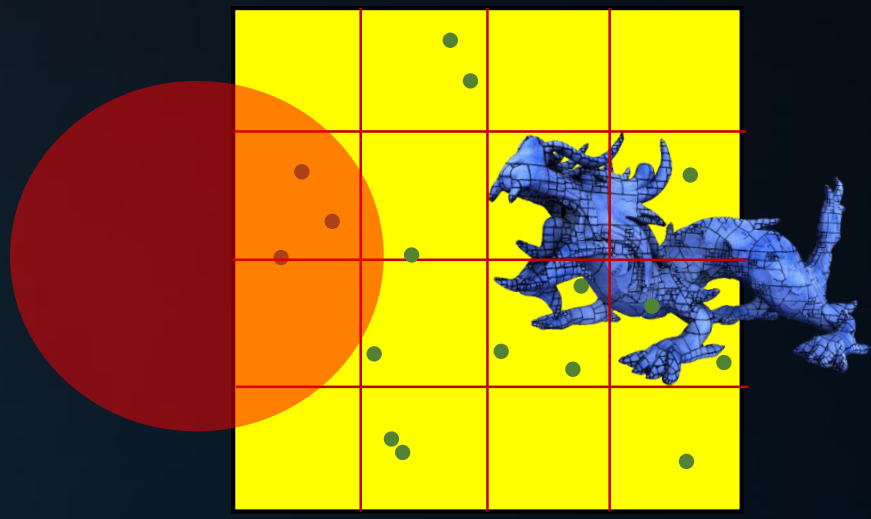
- Introduction
- Stratification
- Blue Noise
- Next Event Estimation

Uniform Random Sampling

To sample a light source, we draw two random values in the range 0..1.

The resulting 2D positions are not uniformly distributed over the area.

We can improve uniformity using *stratification*: one sample is placed in each stratum.



Uniform Random Sampling

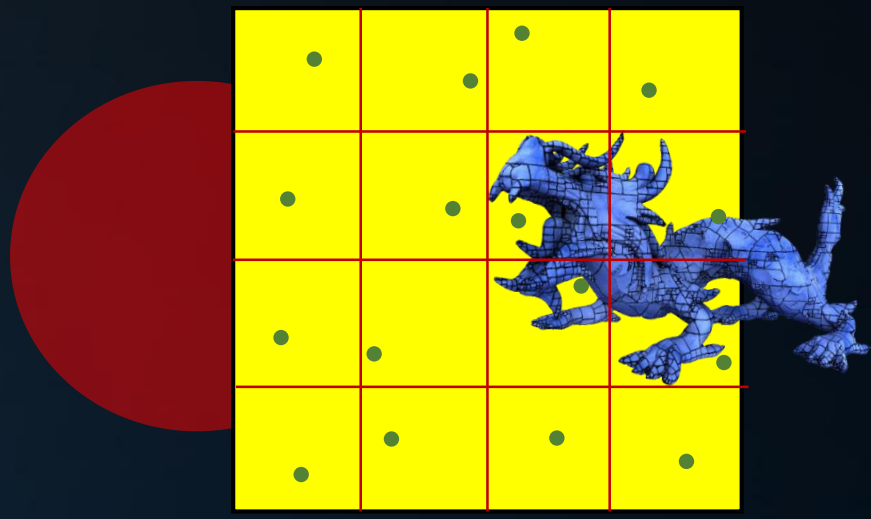
To sample a light source, we draw two random values in the range 0..1.

The resulting 2D positions are not uniformly distributed over the area.

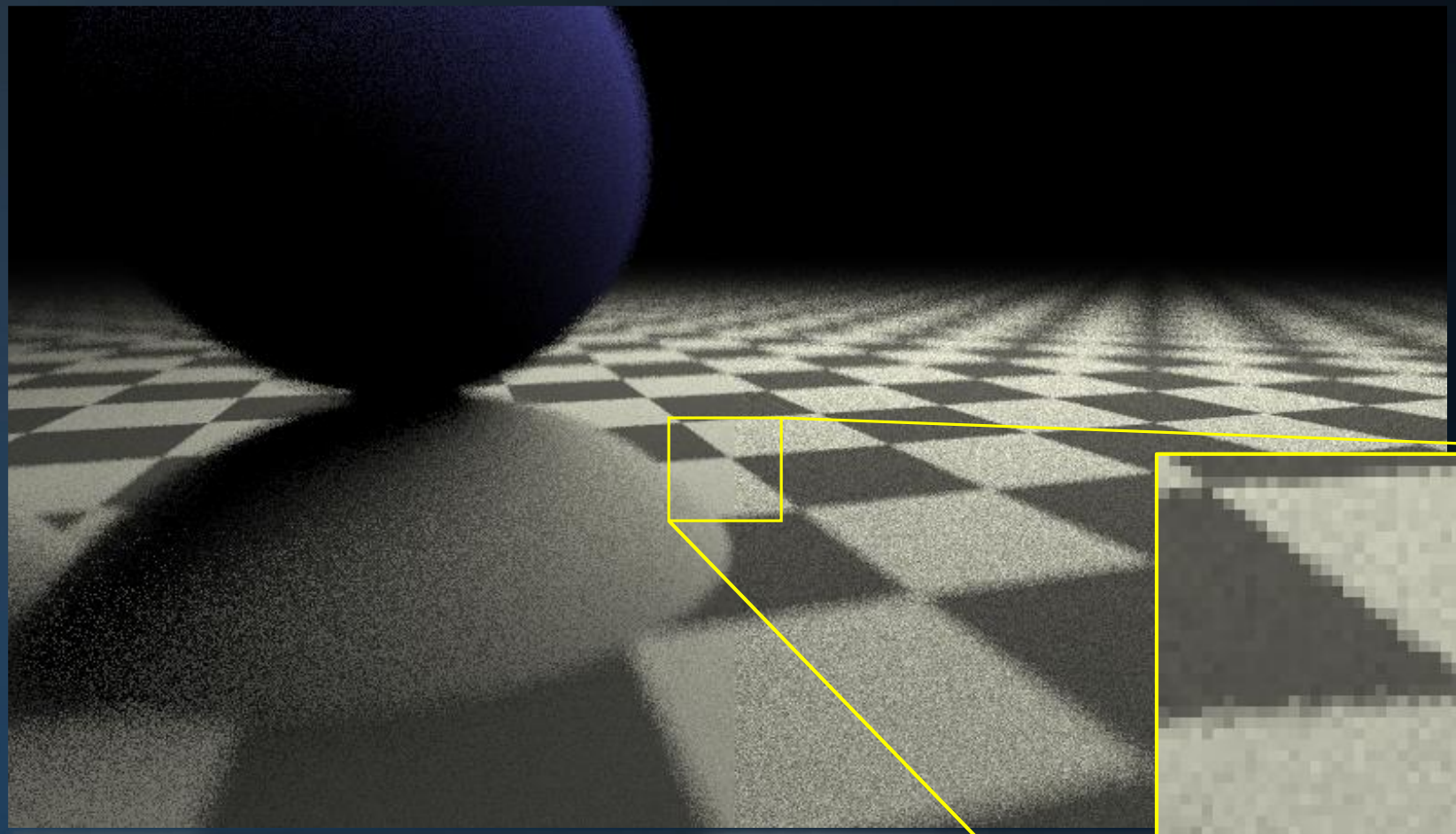
We can improve uniformity using *stratification*: one sample is placed in each stratum.

For 4x4 strata:

```
stratum_x = (idx % 4) * 0.25 // idx = 0..15
stratum_y = (idx / 4) * 0.25
r0 = Rand() * 0.25
r1 = Rand() * 0.25
P = vec2( stratum_x + r0, stratum_y + r1 )
```



Ray Tracing for Games



Troubleshooting Path Tracing Experiments

When experimenting with stratification and other variance reduction methods you will frequently produce incorrect images.

Tip:

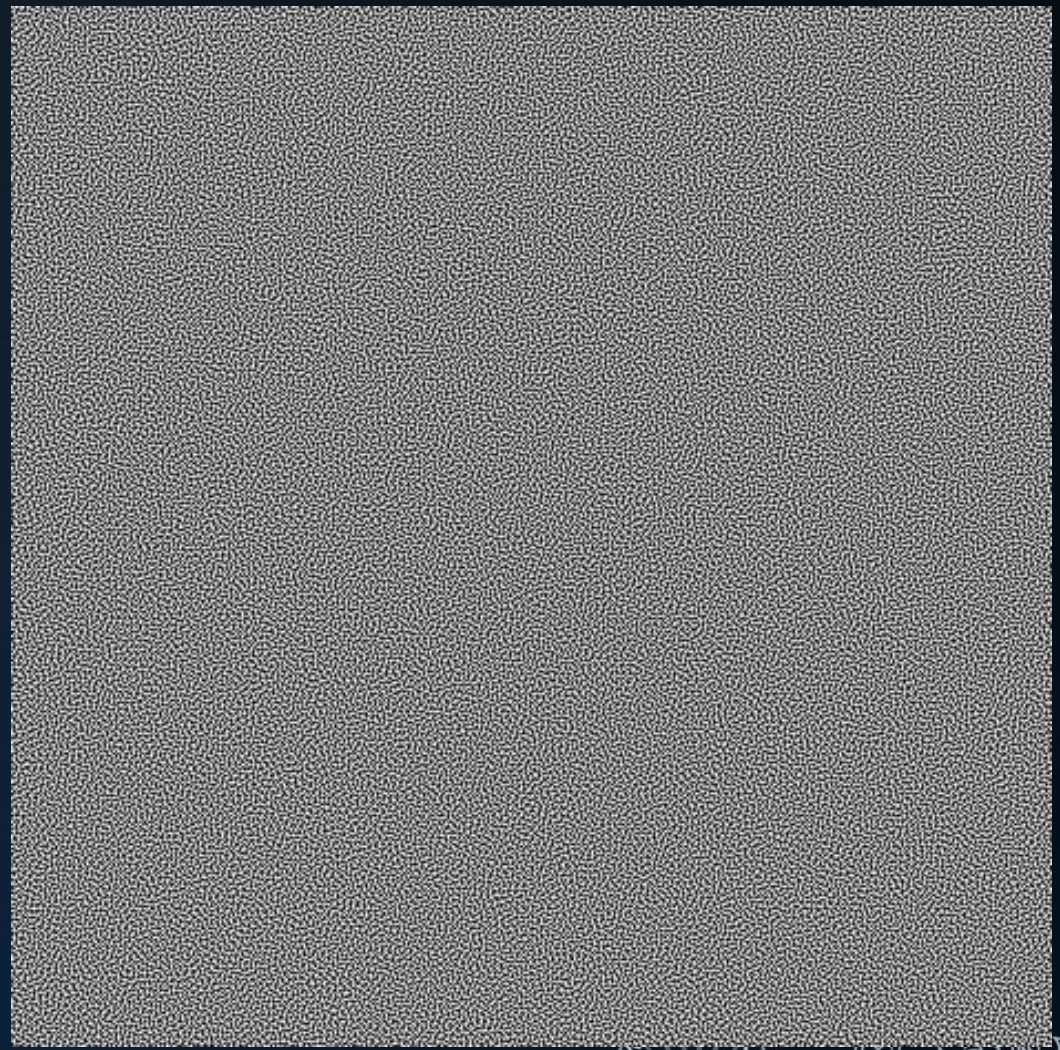
Keep a simple reference path tracer without any tricks.
Compare your output to this reference solution frequently.

Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation

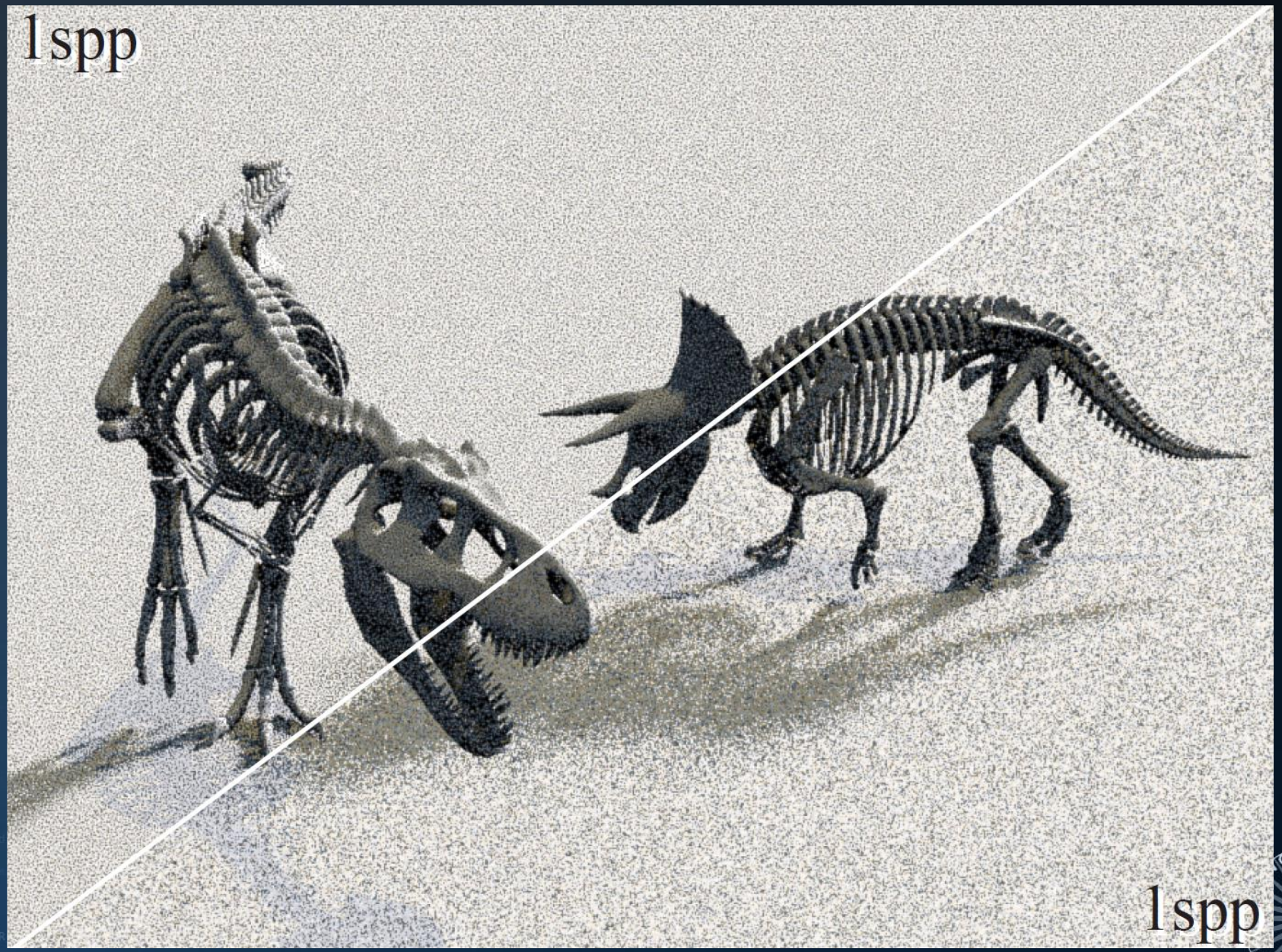
Not All Noise is Equal

Our eyes expect noise. ‘Good noise’ is uniform, high-frequent, and… blue*.

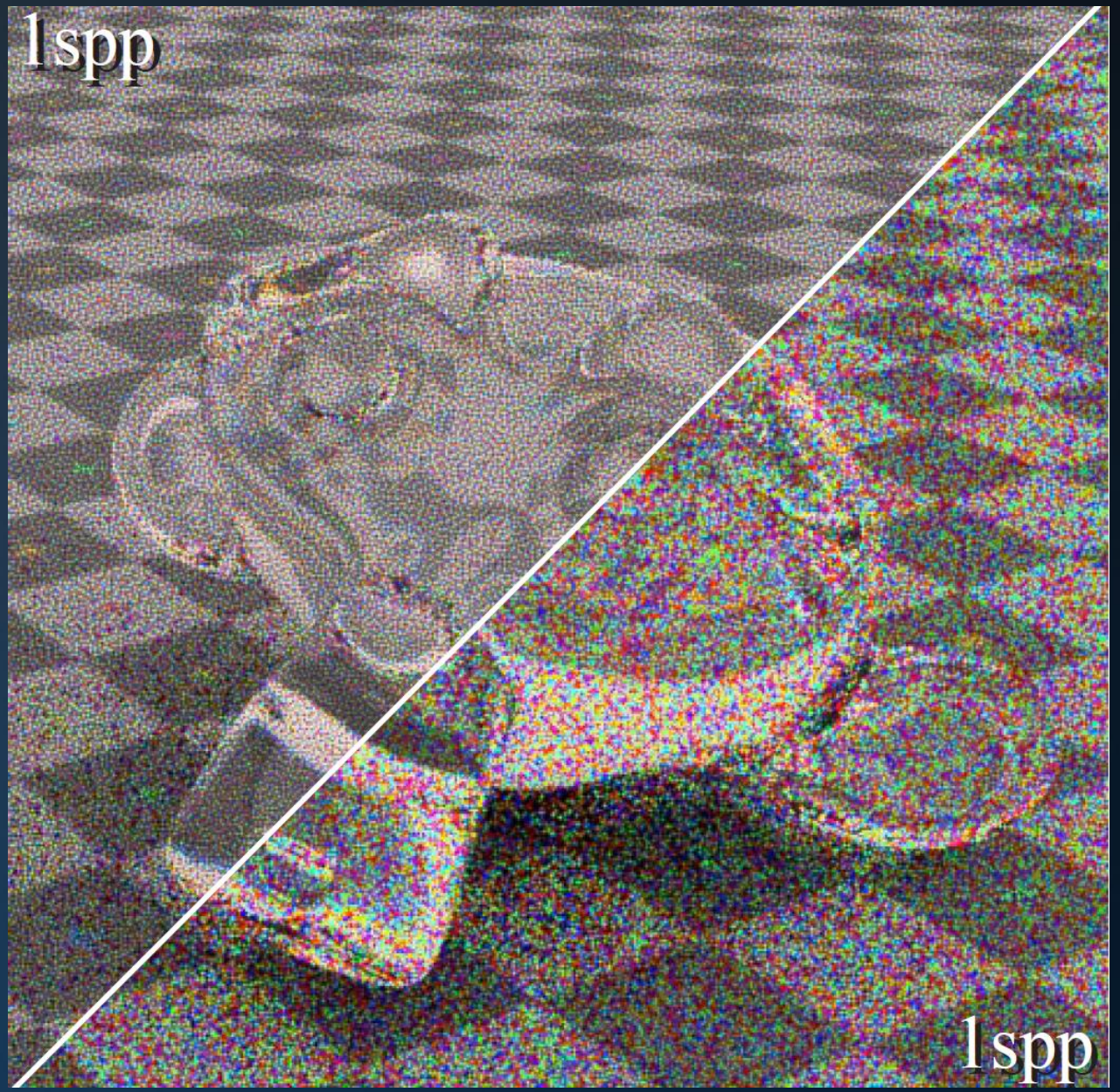


*: www.arnoldrenderer.com/arnold/research

Ray Tracing for Games



Ray Tracing for Games



Agenda:

- Introduction
- Stratification
- Blue Noise
- Next Event Estimation

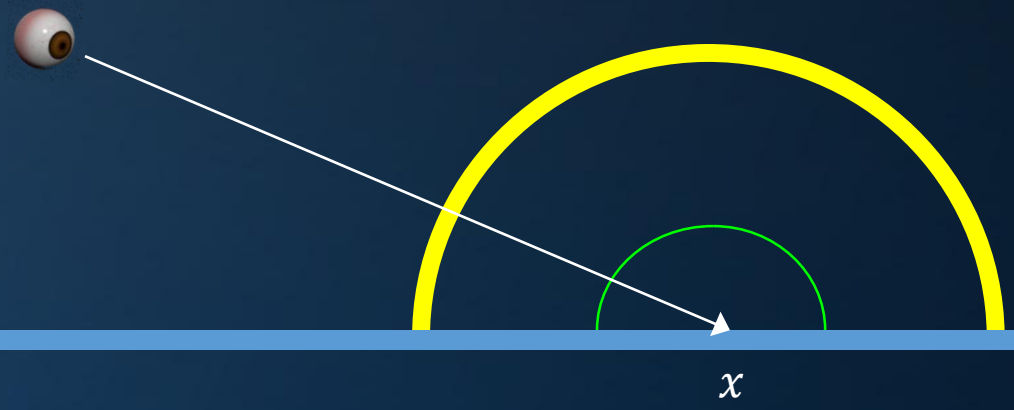
Ray Tracing for Games

Incoming <u>direct light</u>

$$= \int_{\Omega} L_d(x, \omega_i) \cos\theta_i \, d\omega_i$$

$$\approx \frac{2\pi}{N} \sum_{i=1}^{N} L_d(p, \omega_i) \cos\theta_i$$

$-\frac{1}{2}\pi$ $+\frac{1}{2}\pi$

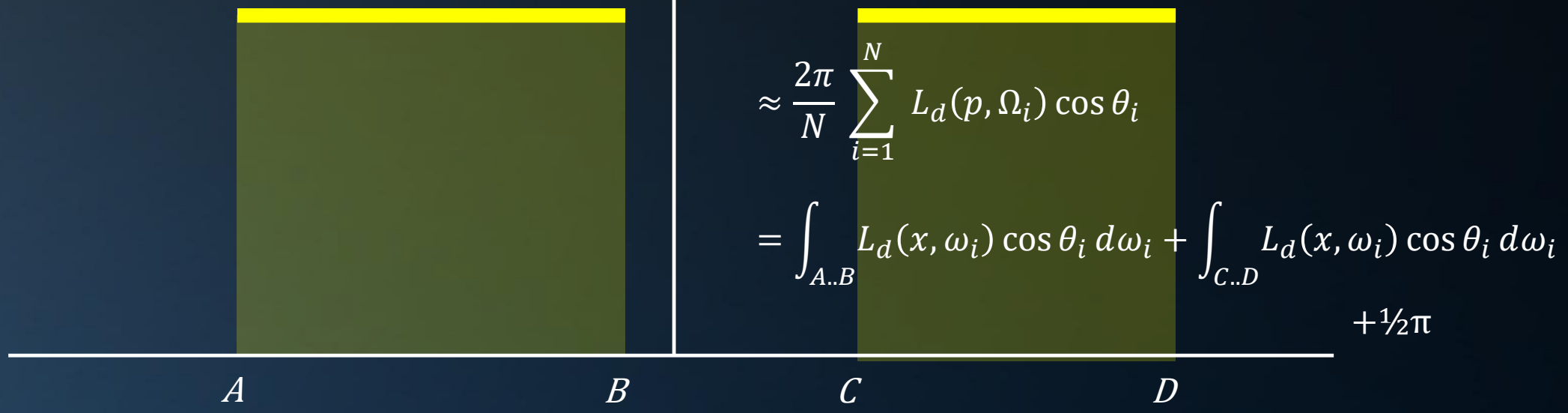


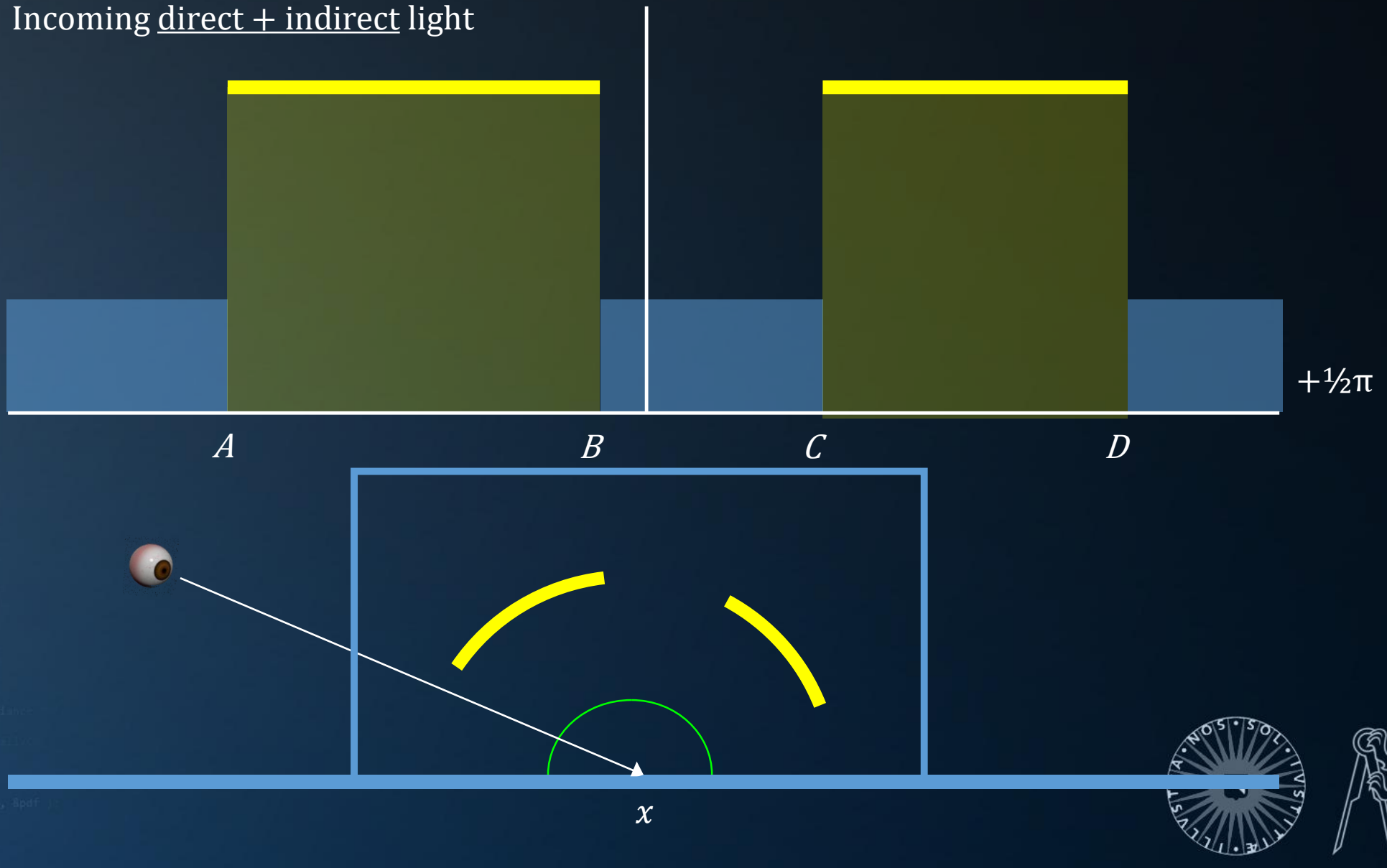
Incoming <u>direct light</u>

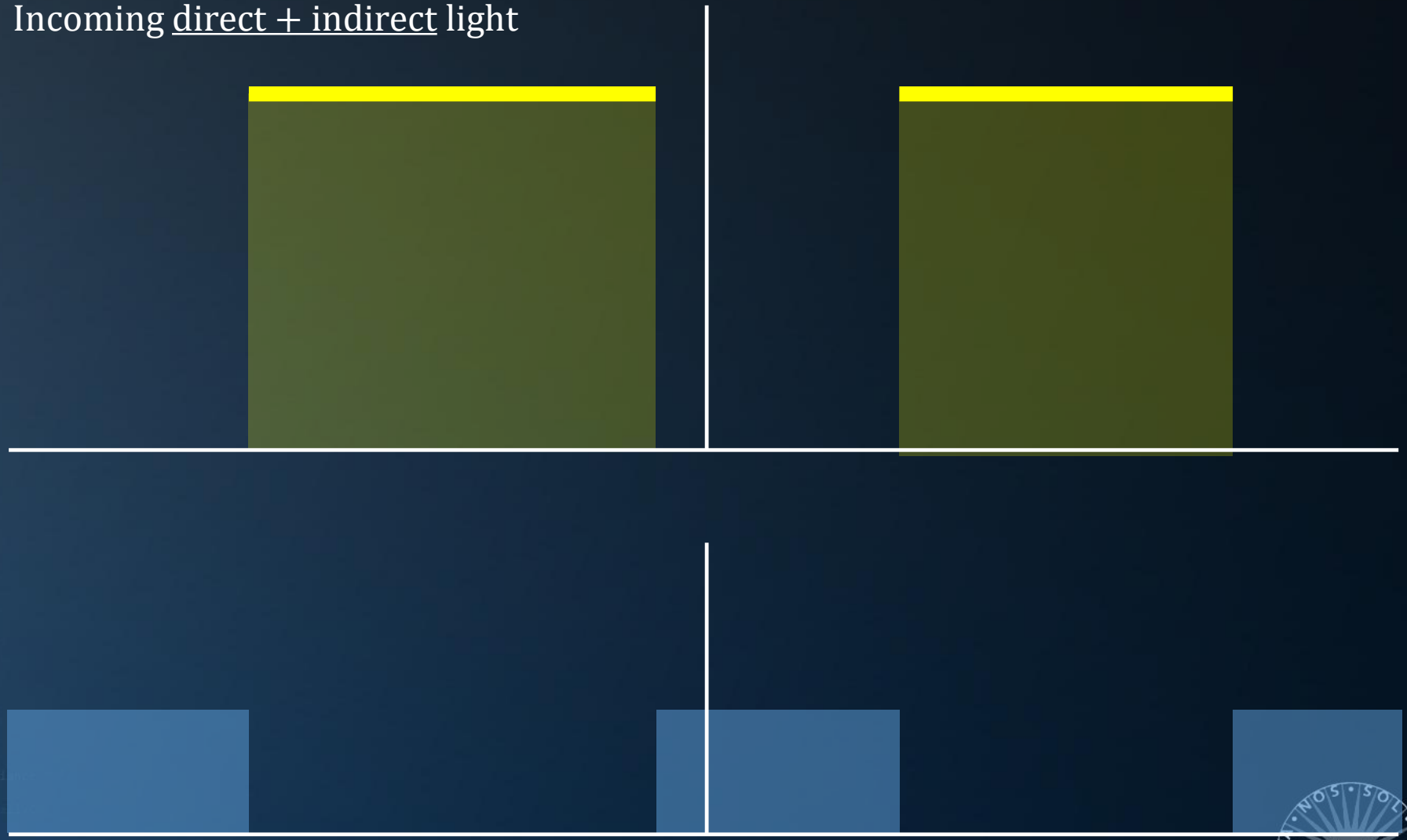
$$= \int_{\Omega} L_d(x, \omega_i) \cos\theta_i \, d\omega_i$$

$$\approx \frac{2\pi}{N} \sum_{i=1}^{N} L_d(p, \Omega_i) \cos\theta_i$$

$$= \int_{A..B} L_d(x, \omega_i) \cos\theta_i \, d\omega_i + \int_{C..D} L_d(x, \omega_i) \cos\theta_i \, d\omega_i$$



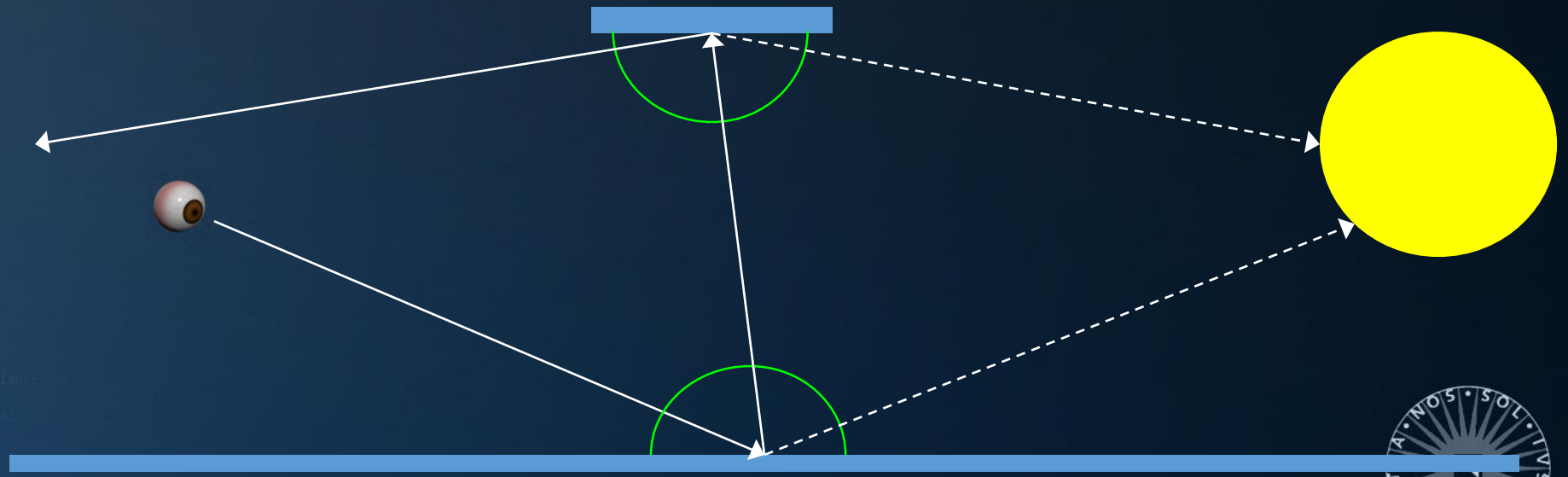

Incoming direct + indirect light
−½π
+½π
A
B
C
D
x


Incoming direct + indirect light
−½π
+½π
−½π
+½π

Next Event Estimation

Observation: light travelling via any vertex on the path consists of indirect light and direct light *for that vertex.*

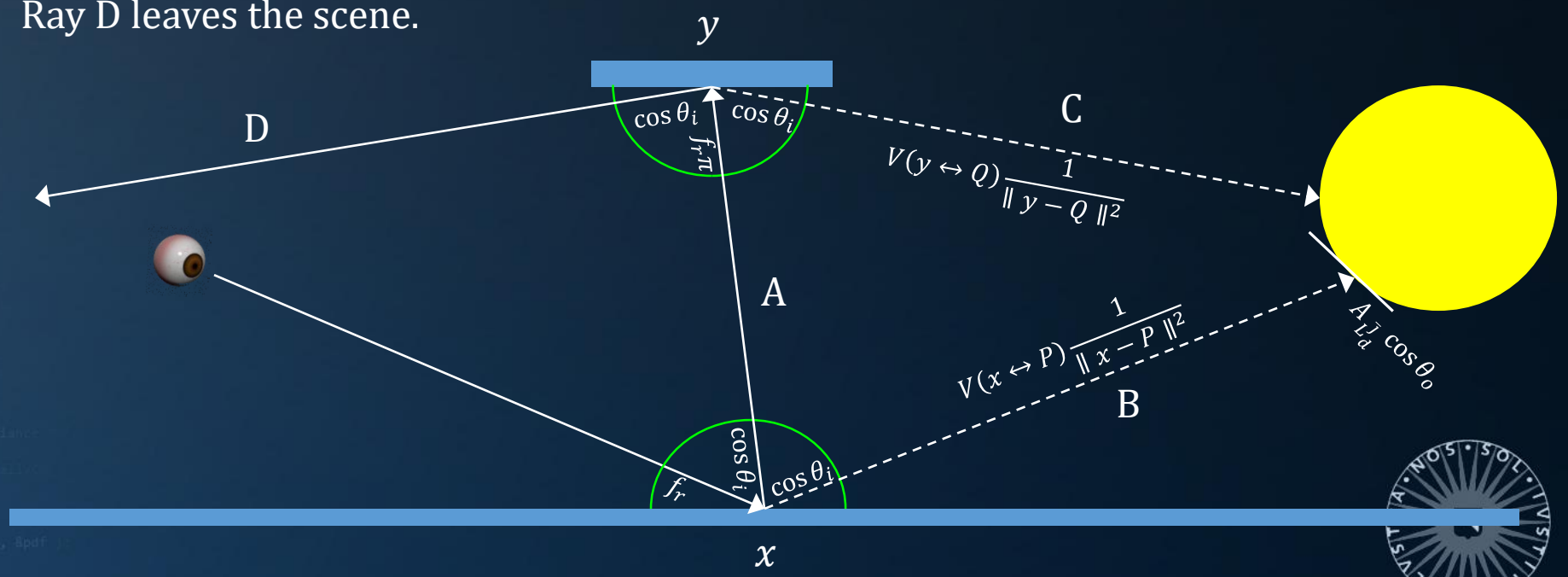
Next Event Estimation: sampling direct and indirect separately.



Next Event Estimation

Per surface interaction, we trace *two* random rays.

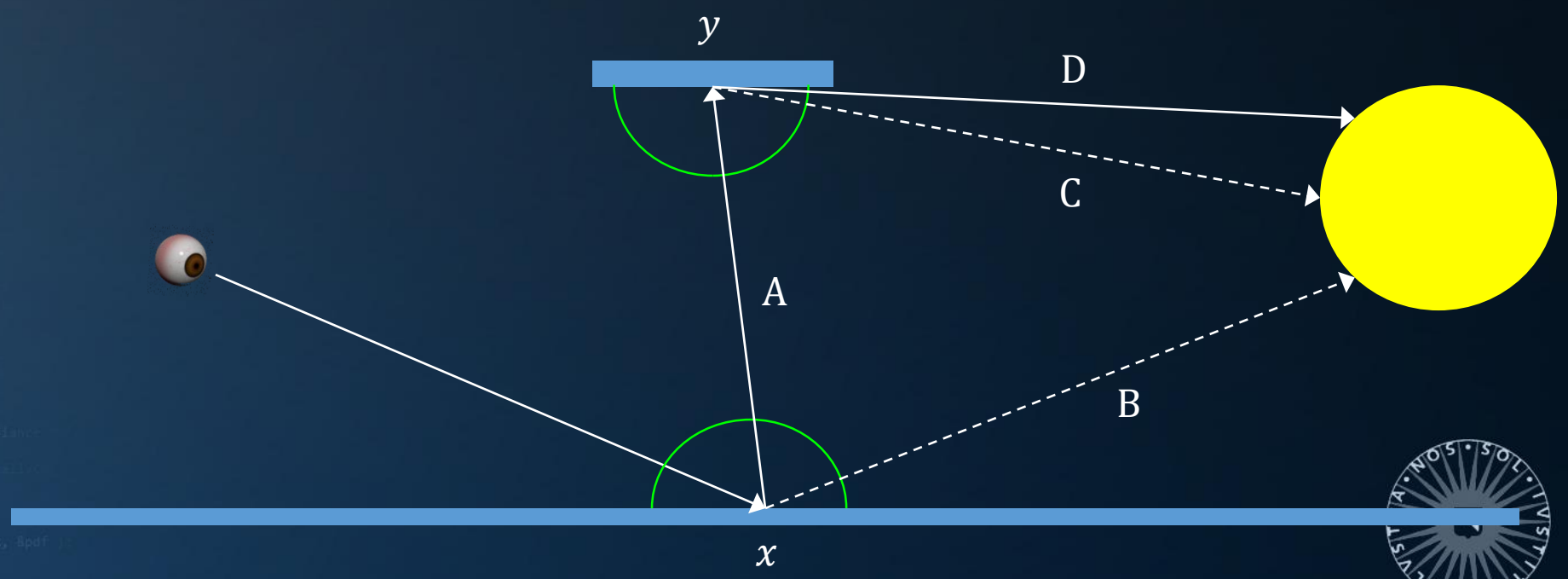
- Ray A returns (via point x) the energy reflected by y (estimates indirect light for x).
- Ray B returns the direct illumination on point x (estimates direct light on x).
- Ray C returns the direct illumination on point y, which will reach the sensor via ray A.
- Ray D leaves the scene.



Next Event Estimation

When a ray for indirect illumination stumbles upon a light, the path is terminated and no energy is transported via ray D:

This way, we prevent accounting for direct illumination on point y twice.



Next Event Estimation

We thus split the hemisphere into two distinct areas:

1. The area that has the projection of the light source on it;
2. The area that is not covered by this projection.

We can now safely send a ray to each of these areas and sum whatever we find there.

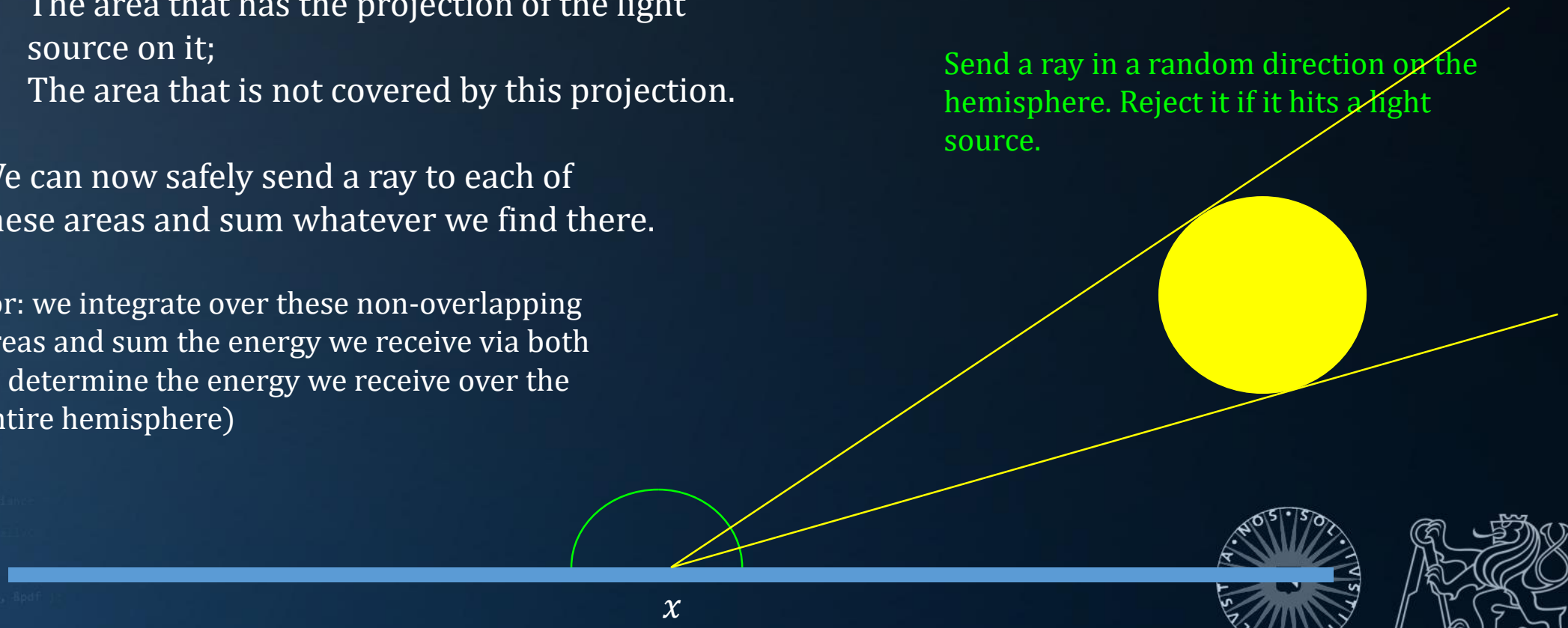
(or: we integrate over these non-overlapping areas and sum the energy we receive via both to determine the energy we receive over the entire hemisphere)

Area 1:

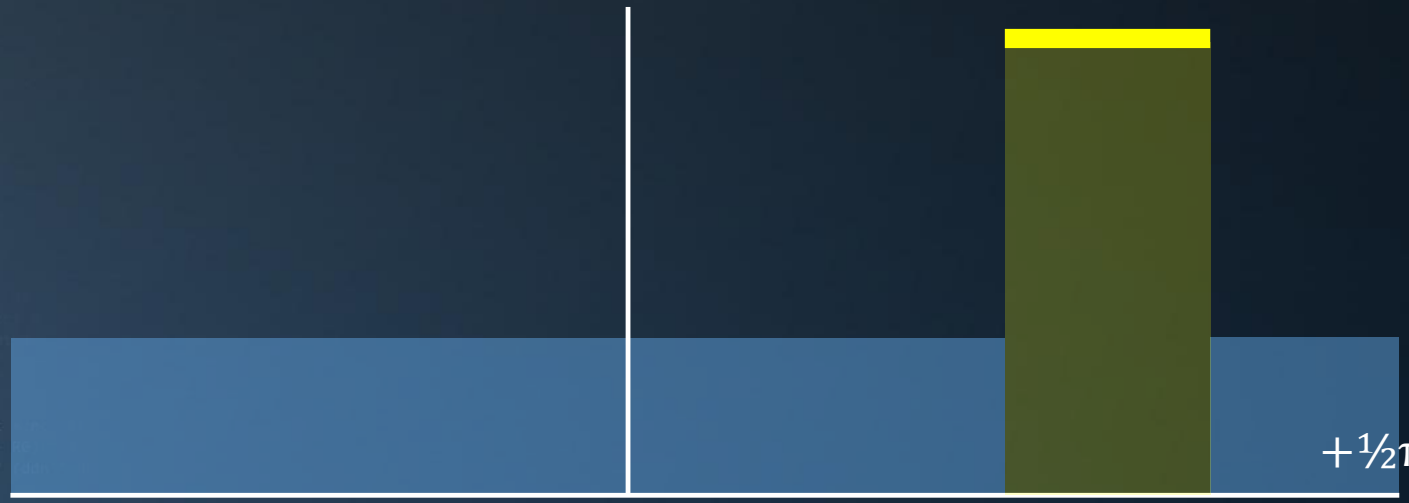
Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:

Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.



Ray Tracing for Games

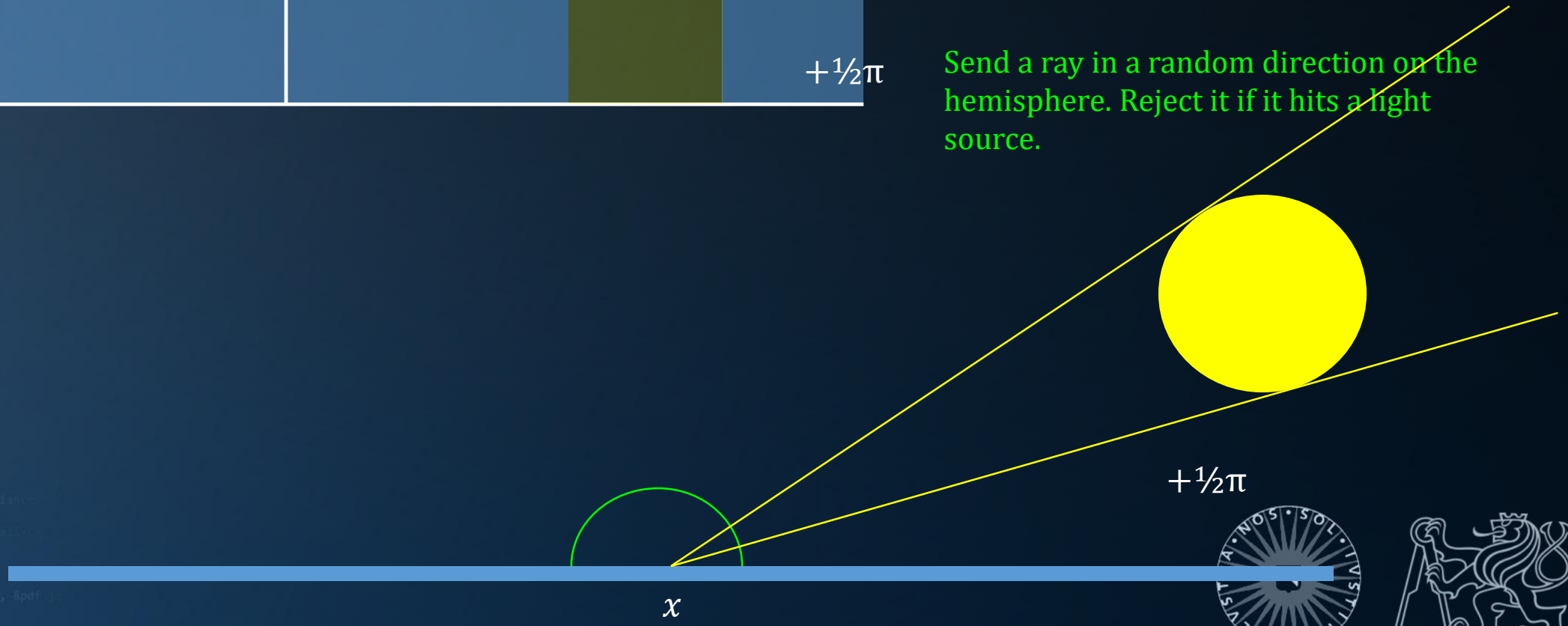


Area 1:

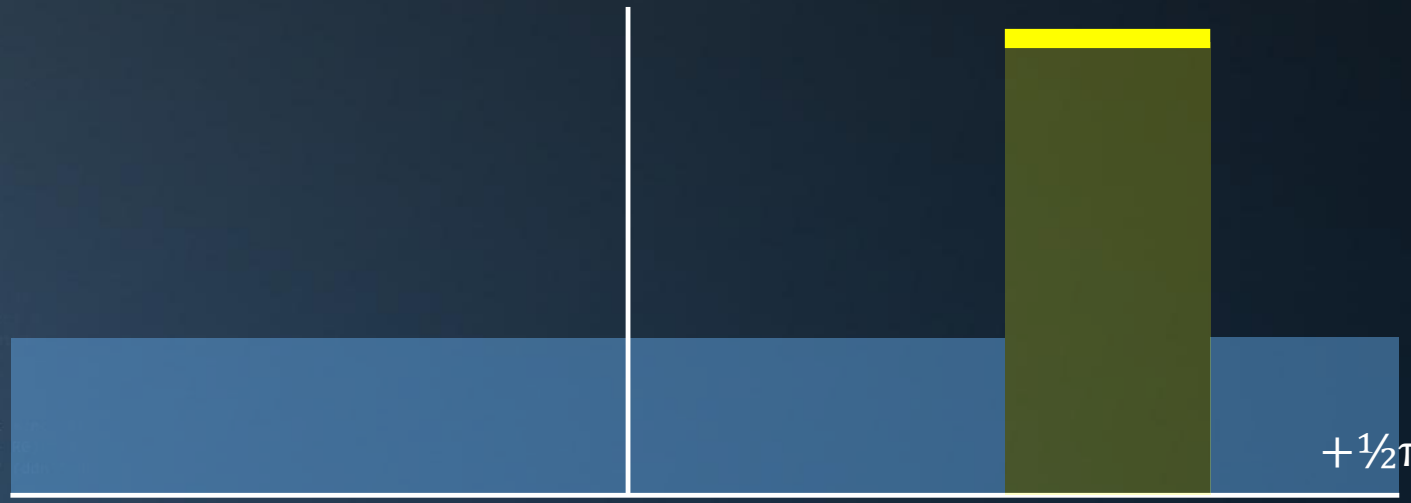
Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:

Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.



Ray Tracing for Games

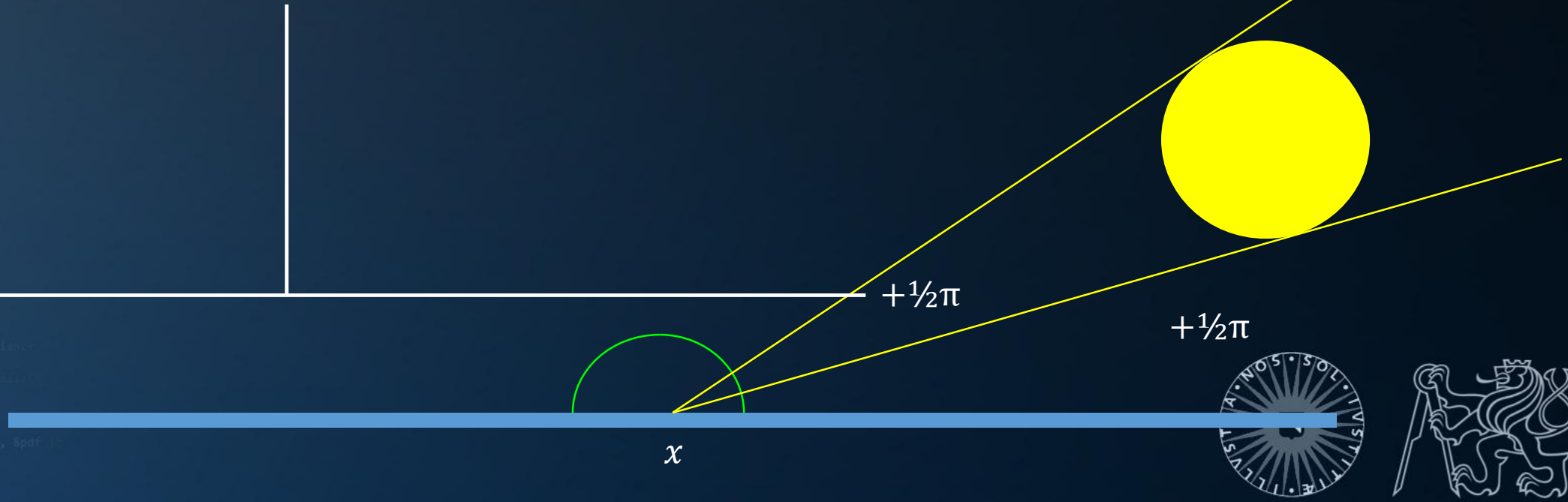


Area 1:

Send a ray directly to a random light source. Reject it if it hits anything else than the targeted light.

Area 2:

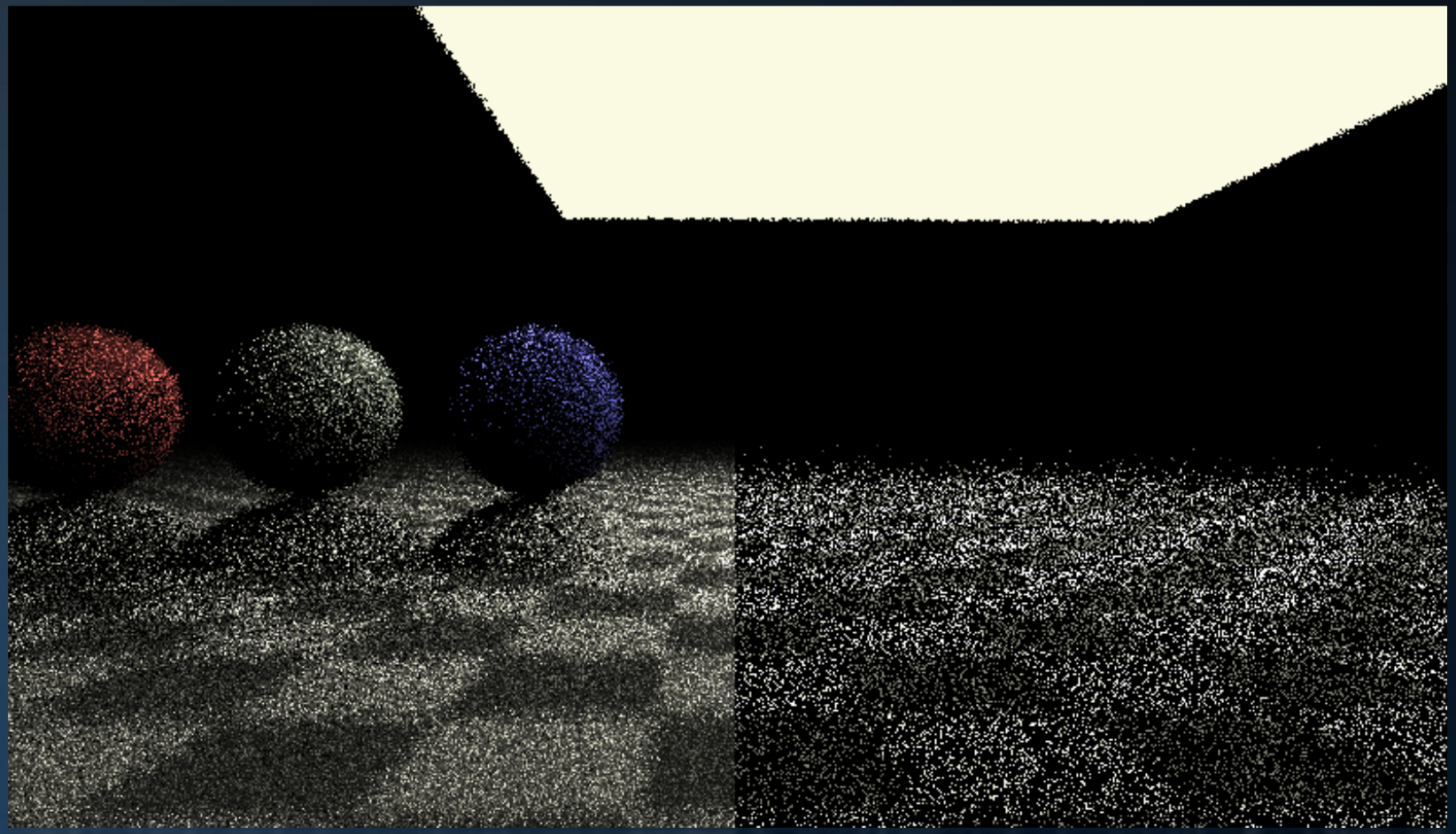
Send a ray in a random direction on the hemisphere. Reject it if it hits a light source.



Next Event Estimation

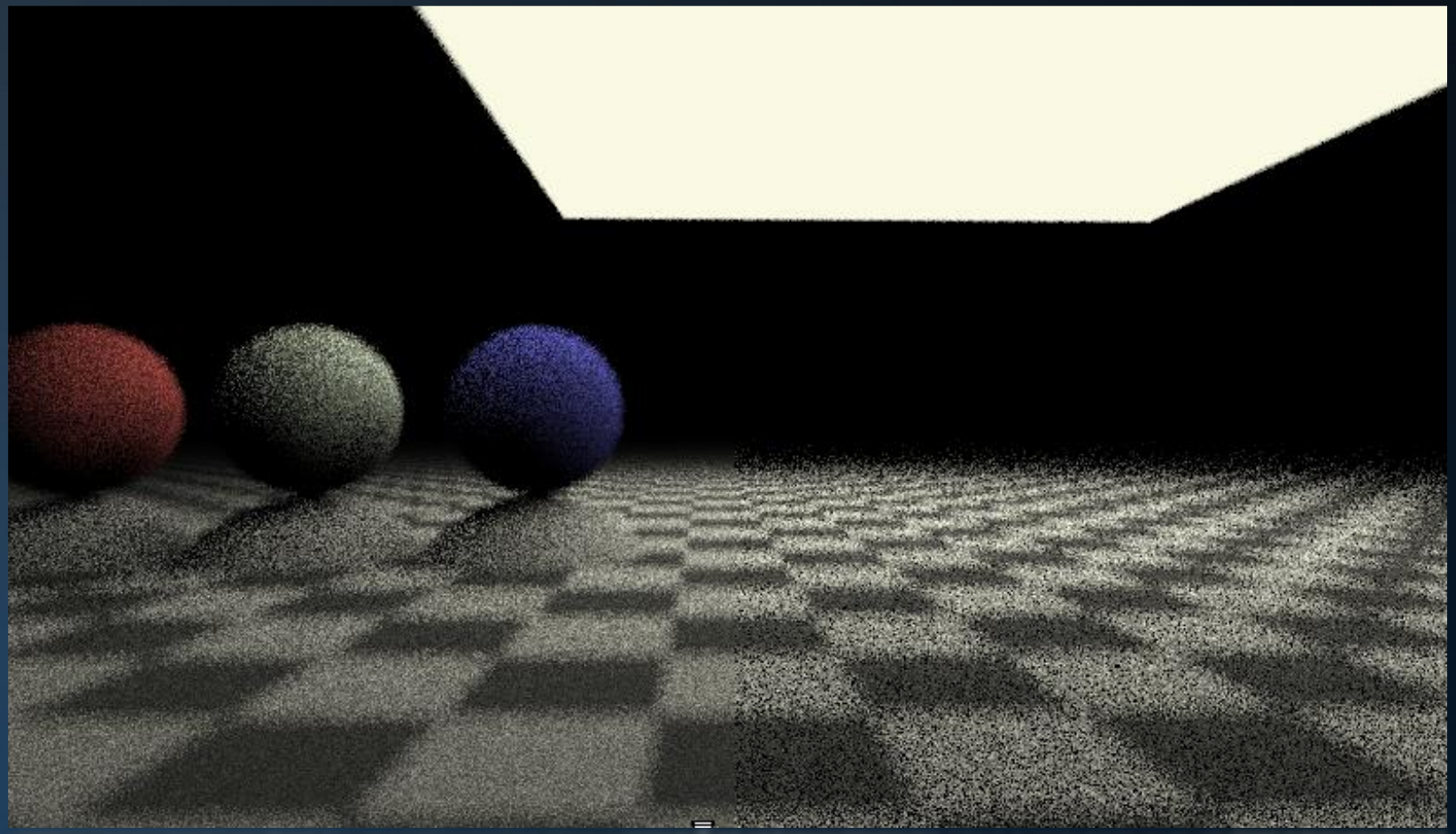
```
Color Sample( Ray ray )
{
   // trace ray
   I, N, material = Trace( ray );
   BRDF = material.albedo / PI;
   // terminate if ray left the scene
   if (ray.NOHIT) return BLACK;
   // terminate if we hit a light source
   if (material.isLight) return BLACK;
   // sample a random light source
   L, Nl, dist, A = RandomPointOnLight();
   Ray lr( I, L, dist );
   if (N·L > 0 && Nl·-L > 0) if (!Trace( lr ))
   {
      solidAngle = ((Nl·-L) * A) / dist²;
      Ld = lightColor * solidAngle * BRDF * N·L;
   }
   // continue random walk
   R = DiffuseReflection( N );
   Ray r( I, R );
   Ei = Sample( r ) * (N·R);
   return PI * 2.0f * BRDF * Ei + Ld;
}
```

Ray Tracing for Games



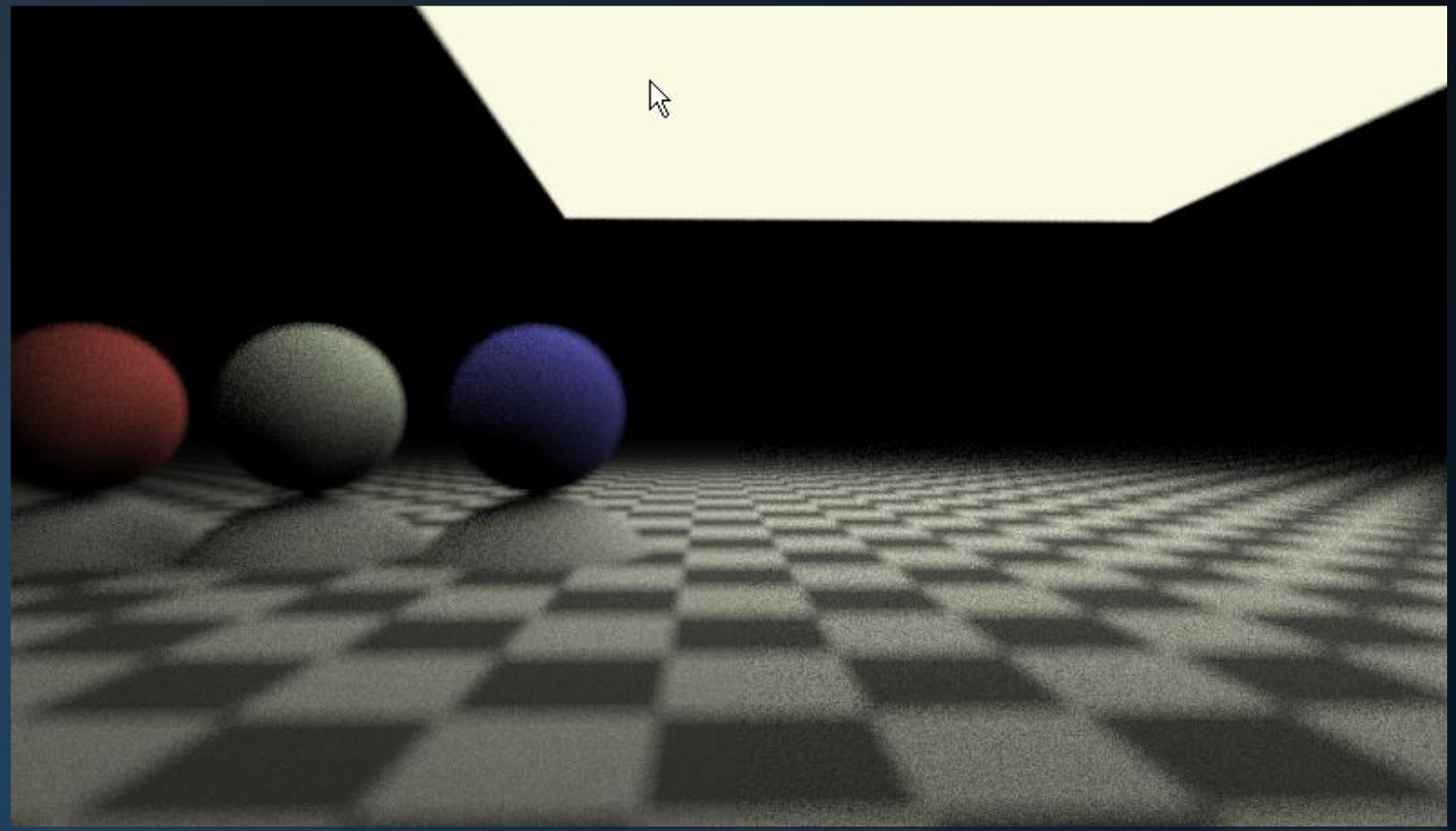
0.1s

Ray Tracing for Games



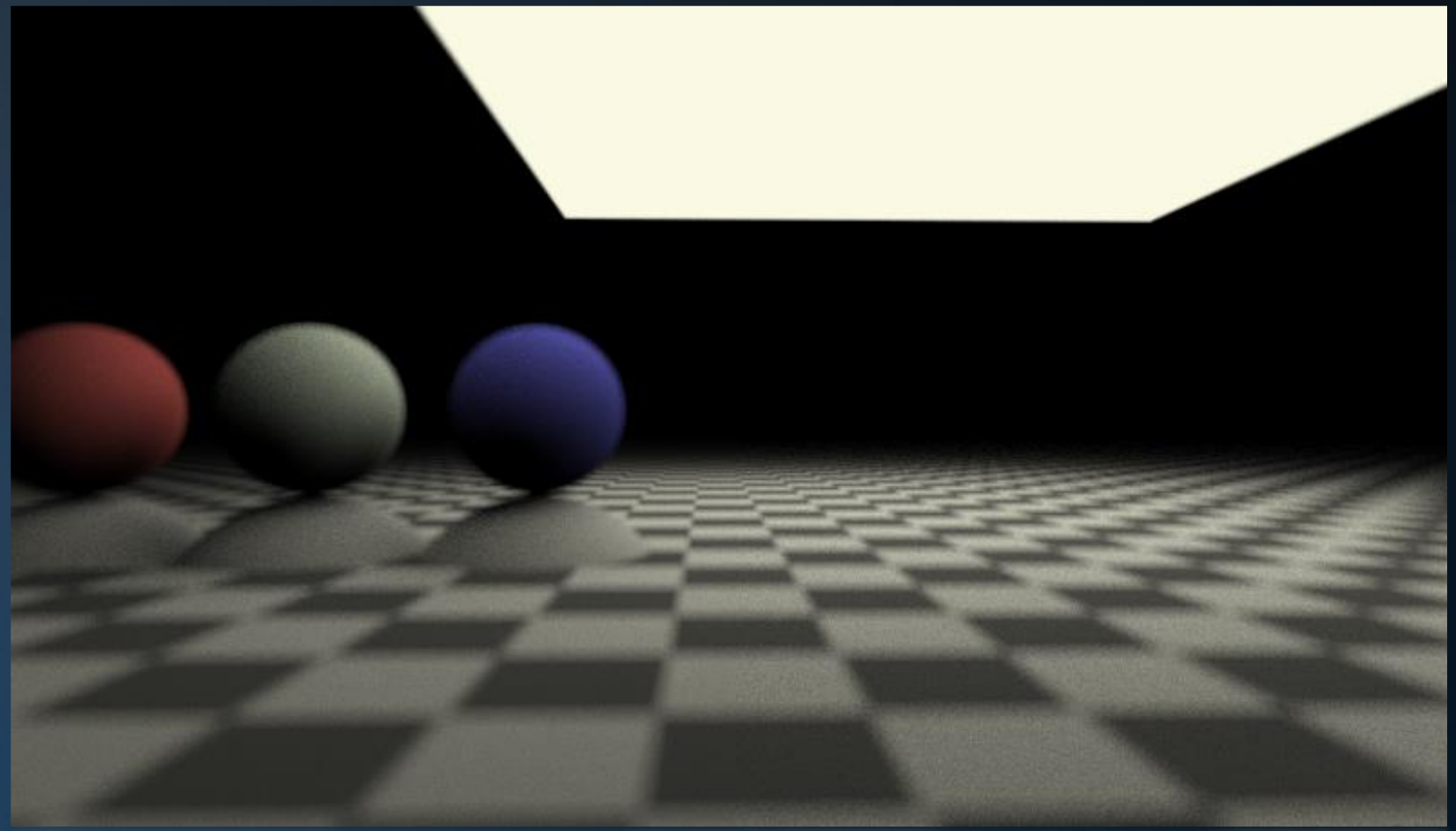
0.5s

Ray Tracing for Games



2.0s

Ray Tracing for Games



30.0s

Next Event Estimation

Some vertices require special attention:

- If the first vertex after the camera is emissive, its energy can't be reflected to the camera.
- For specular surfaces, the BRDF to a light is always 0.

Since a light ray doesn't make sense for specular vertices, we will include emission from a vertex directly following a specular vertex.

The same goes for the first vertex after the camera: if this is emissive, we will also include this.

This means we need to keep track of the type of the previous vertex during the random walk.

Ray Tracing for Games

```
Color Sample( Ray ray, bool lastSpecular )
{
   // trace ray
   I, N, material = Trace( ray );
   BRDF = material.albedo / PI;
   // terminate if ray left the scene
   if (ray.NOHIT) return BLACK;
   // terminate if we hit a light source
   if (material.isLight)
      if (lastSpecular) return material.emissive;
                        else return BLACK;
   // sample a random light source
   L, Nl, dist, A = RandomPointOnLight();
   Ray lr( I, L, dist );
   if (N·L > 0 && Nl·-L > 0) if (!Trace( lr ))
   {
      solidAngle = ((Nl·-L) * A) / dist²;
      Ld = lightColor * solidAngle * BRDF * N·L;
   }
   // continue random walk
   R = DiffuseReflection( N );
   Ray r( I, R );
   Ei = Sample( r, false ) * (N·R);
   return PI * 2.0f * BRDF * Ei + Ld;
}
```

End of PART 9.