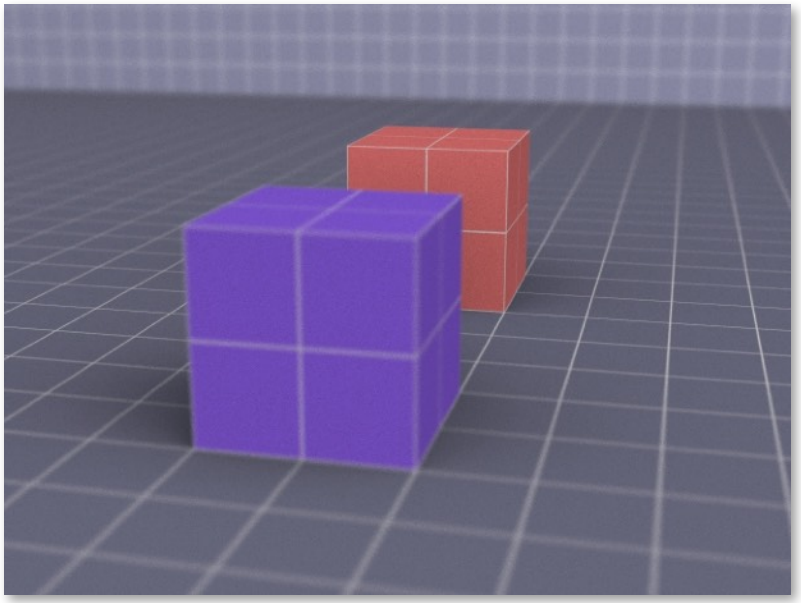
Global Illumination II

Our story so far

« Distributed Ray Tracing » Cook et al. (1984)

- Soft shadows
  - Several rays for each extended light source
- Anti-aliasing
  - Several rays for each pixel
- Glossy Reflection
  - Several rays are reflected
- Motion blur
  - Several rays during time
- Depth of field
  - Several rays per pixel, focusing with a lens



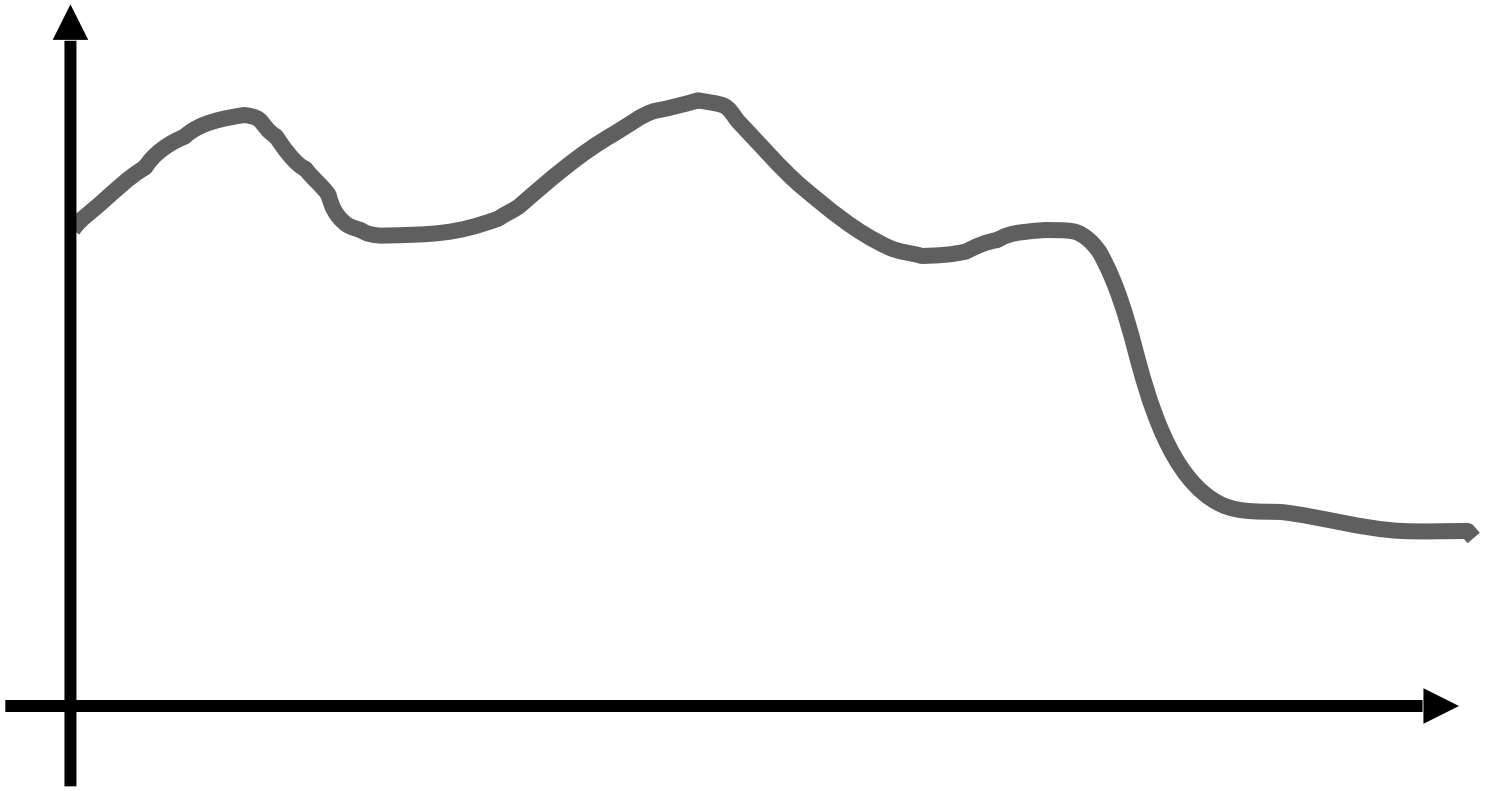
Ray tracing = integrating!

- Integrating what?
  - light sources: soft shadows
  - pixels: anti-aliasing
  - BRDF: glossy reflections
  - over Time: motion blur
  - over the lens: depth of field
  - over the hemisphere: indirect lighting
  - over light paths: global illumination
- Generic method for computing multi-dimensional integrals:

Monte Carlo Integration

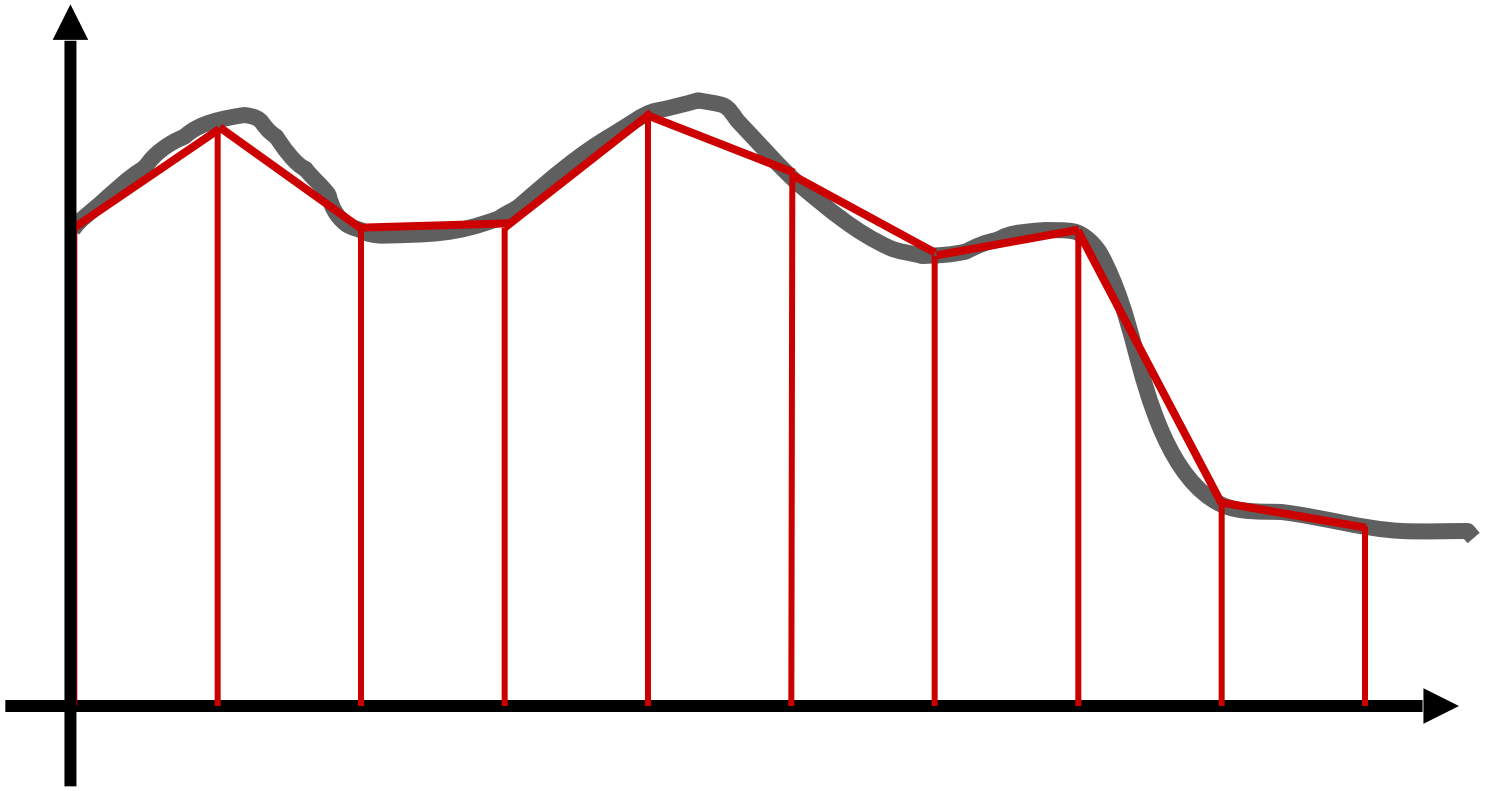
1D Integrals

- Integral of an arbitrary function
- Continuous problem ⇨ discretization



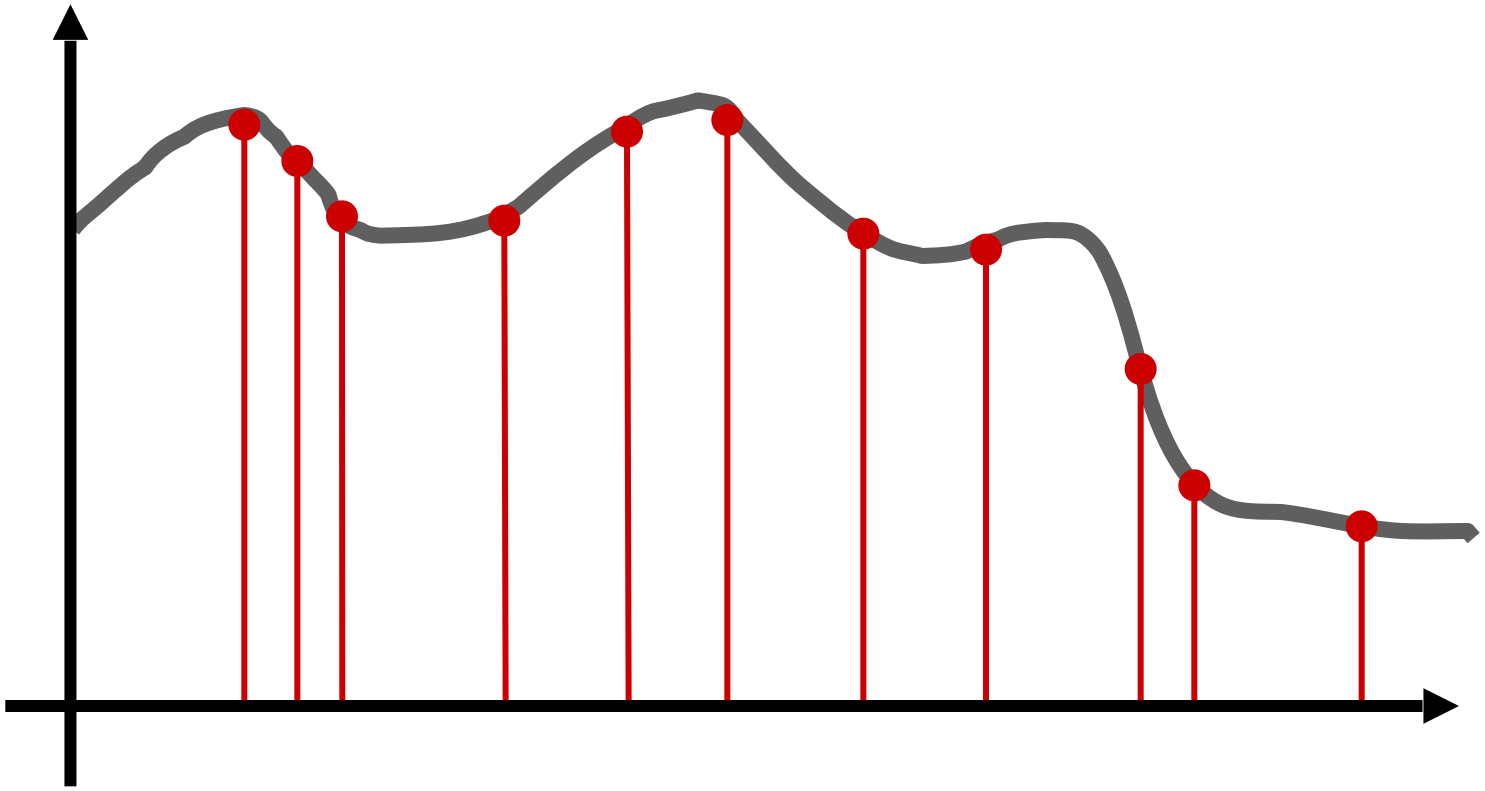
1D Integrals

- Trapezoidal approximation:
  - Also Simpson's rule, midpoint rule...



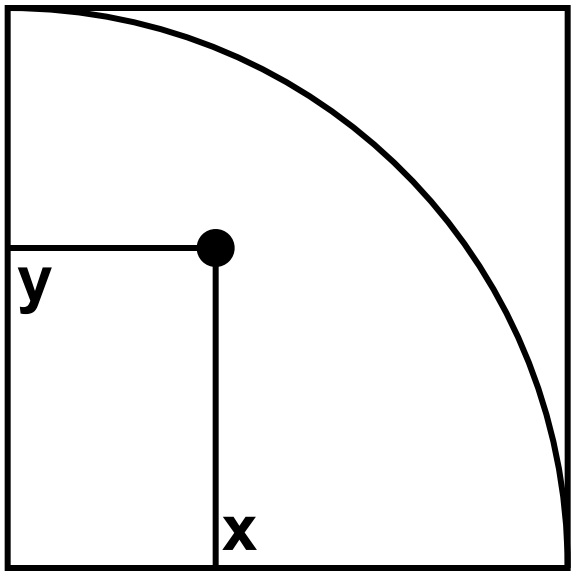
1D Integrals

- Monte Carlo: random sampling
  - Don't keep the distance between the n samples
  - But on average, expect it to be 1/n



Monte Carlo: computing π

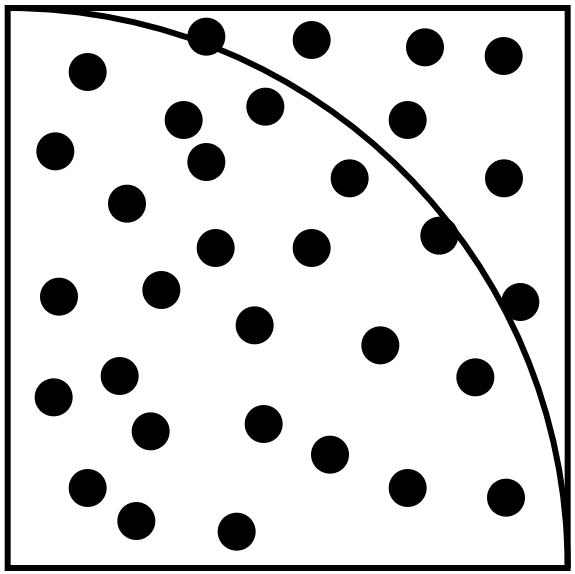
- Take a square
- Take a random point (x,y) in the square
- Test whether it is inside the ¼ disc ($x^2+y^2 < 1$)
- Probability is π /4



Integral of the function equal to 1 on the disc, 0 outside

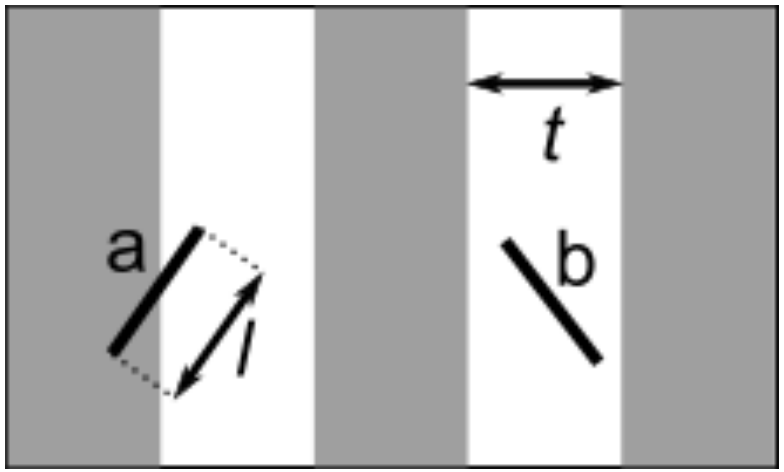
Monte Carlo: computing π

- Probability is π /4
- n = # points inside / # total points
- π ≈ n * 4
- Error depends on the number of samples



See also: Buffon's needle

- Floor made of parallel strips of wood
- Throw needles on the floor
  - The needle either crosses the line or it doesn't
  - Count number of time it crosses the line
  - Divide by total number of throws
- Result is connected to π
- $P = 2l/t\pi$



Why not use Trapezoidal rule?

- To compute π, Monte Carlo is not highly efficient
- But convergence rate independent from dimension
- ⇨ Monte Carlo integration very efficient for higher dimensions

Continuous random variables

- Random variable x
- Probability distribution: p(x)
  - Probability that this variable is between x and x+dx is p(x) dx

Expected value (mean)

$$E[x] = \int_{-\infty}^{\infty} xp(x)dx$$

$$E[f(x)] = \int_{-\infty}^{\infty} f(x)p(x)dx$$

- Expected value is linear:

  $E[f_1(x) + a\, f_2(x)] = E[f_1(x)] + a\, E[f_2(x)]$

Monte Carlo integration

- Take the function f(x) with x in [a b]
- We want to compute: $$I = \int_a^b f(x)dx$$
- Take a random variable x
- If x has a uniform distribution, I=E[f(x)]
  - By definition of expected value

Sum of random variables

- Take N random variables, independent, identical distribution (IID) x_i (N échantillons)
  - Same probability (here uniform)
- Define:

$$F_N = \frac{1}{N} \sum_{j=1}^{n} f(x_i)$$

Monte Carlo estimator

- By linearity of expected value: $E[F_N] = E[f(x)]$

Variance

$$\sigma^2 = E[(x - E[x])^2] = \int_{-\infty}^{\infty} (x - E[x])^2 p(x) dx$$

- Measures the distance to expected value
- Standard deviation σ: square root of variance
- Properties:
  - $\sigma^2[x+y] = \sigma^2[x] + \sigma^2[y] + 2\,\mathrm{Cov}[x,y]$
  - $\sigma^2[ax] = a^2\,\sigma^2[x]$

About the variance

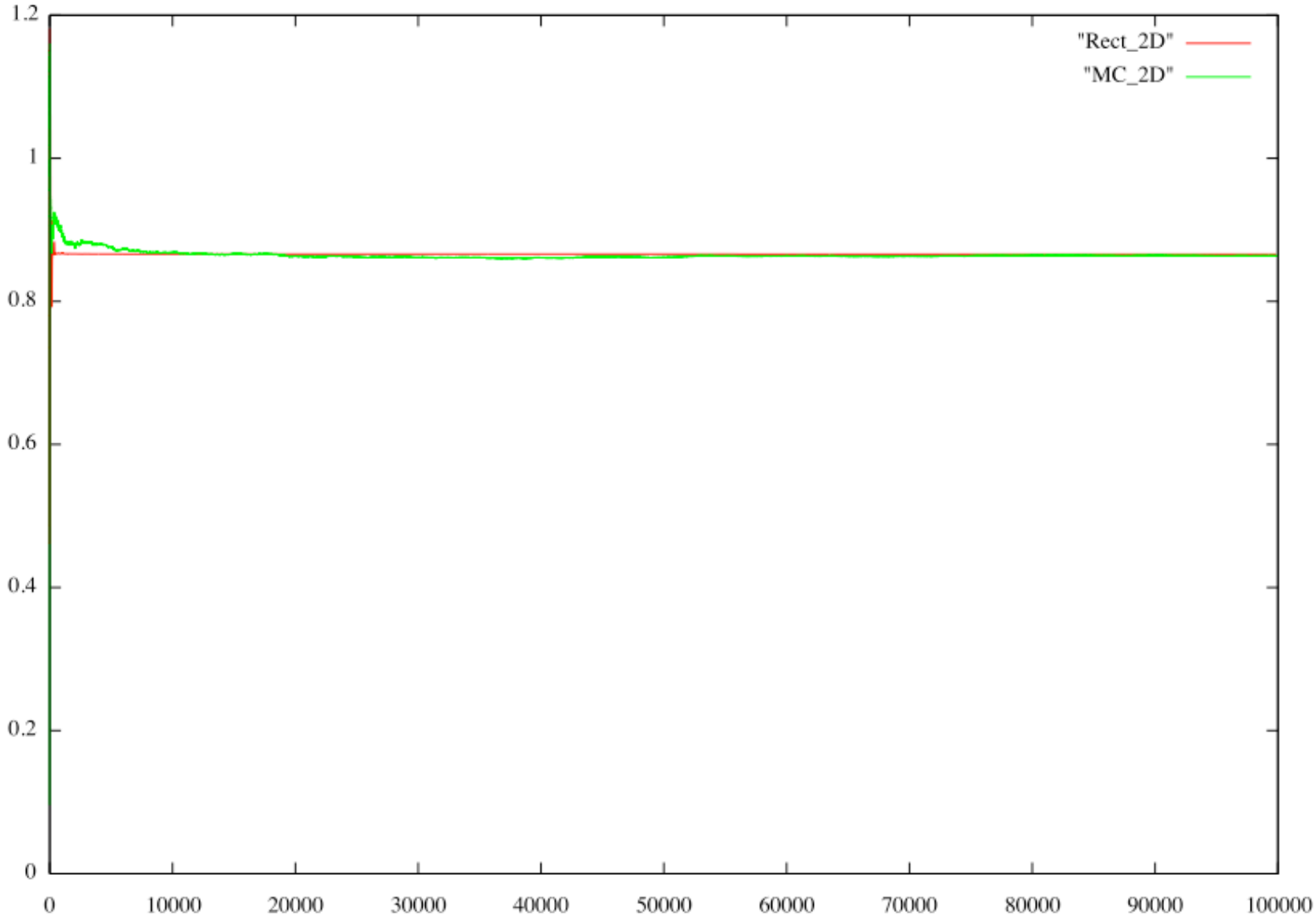
$$\sigma^2[F_N] = \sigma^2\left[\sum_{j=1}^{n} \frac{f(x_i)}{N}\right]$$

- Independent variables ⇨ Cov[x_i, x_j]=0 si i≠j

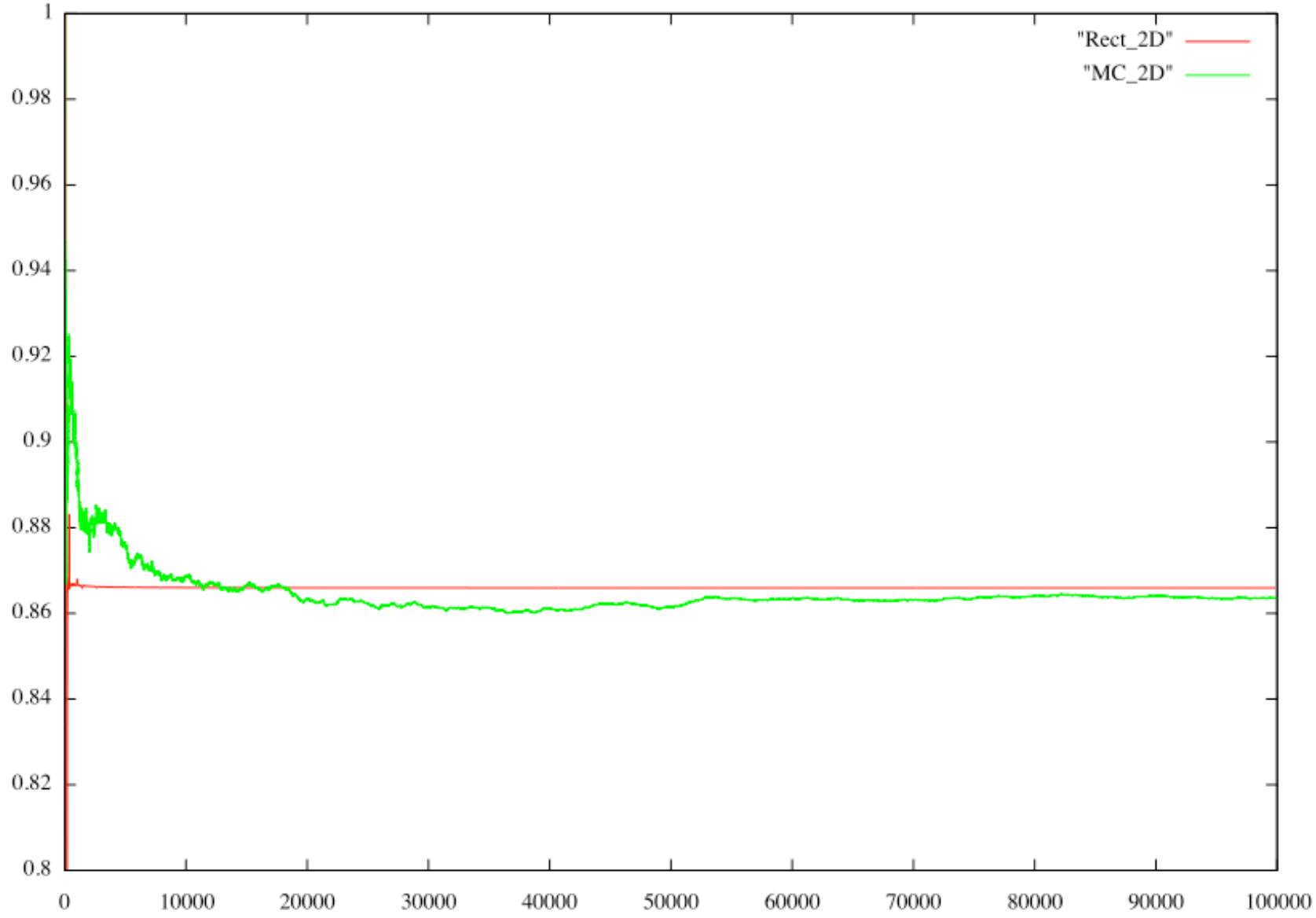
$$\sigma^2[F_N] = \frac{\sigma^2[f(x)]}{N}$$

- thus σ (error) decreases with $\sqrt{N}$
- ⇨ slow convergence

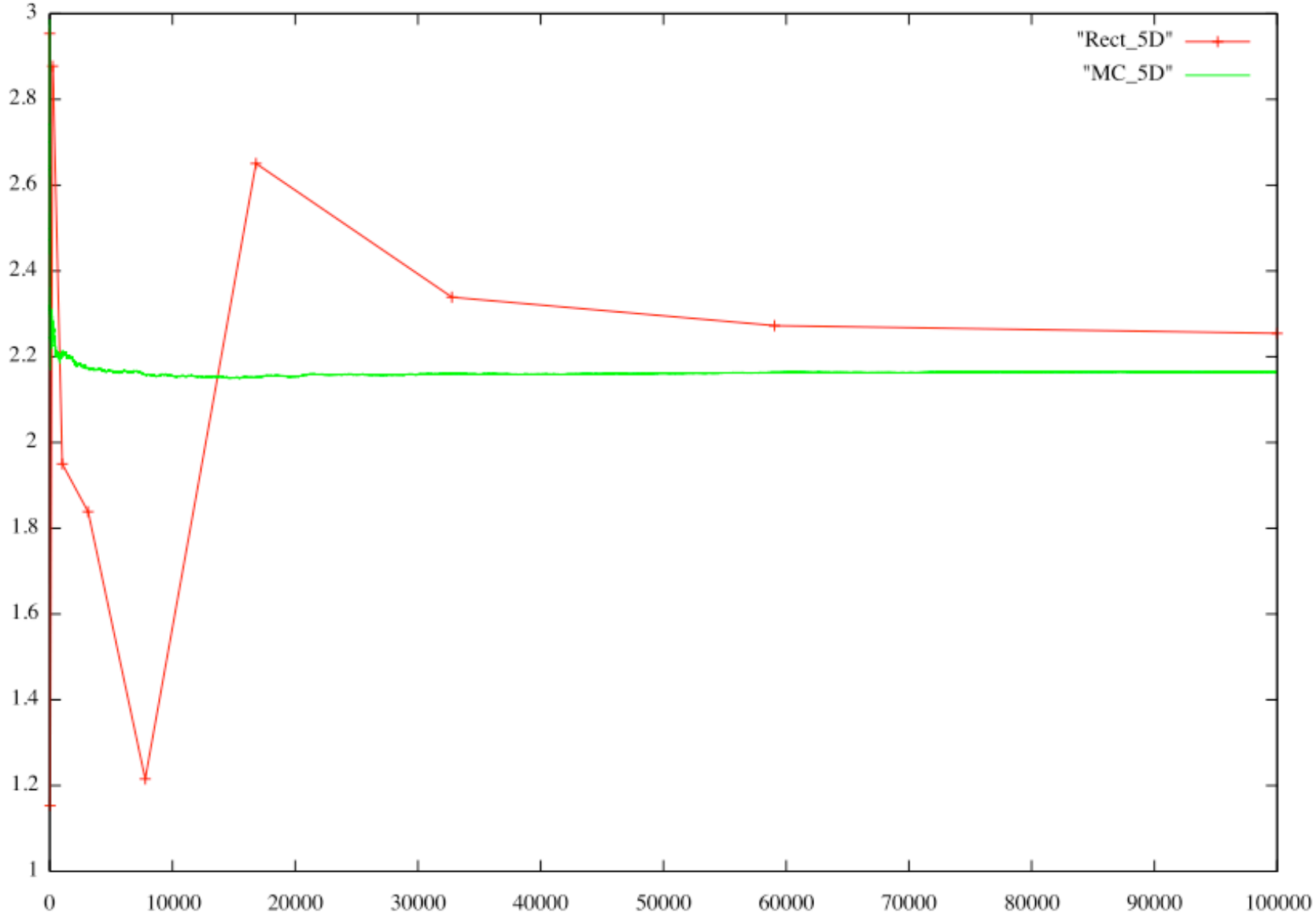
Ex.: Integral of a 2D function



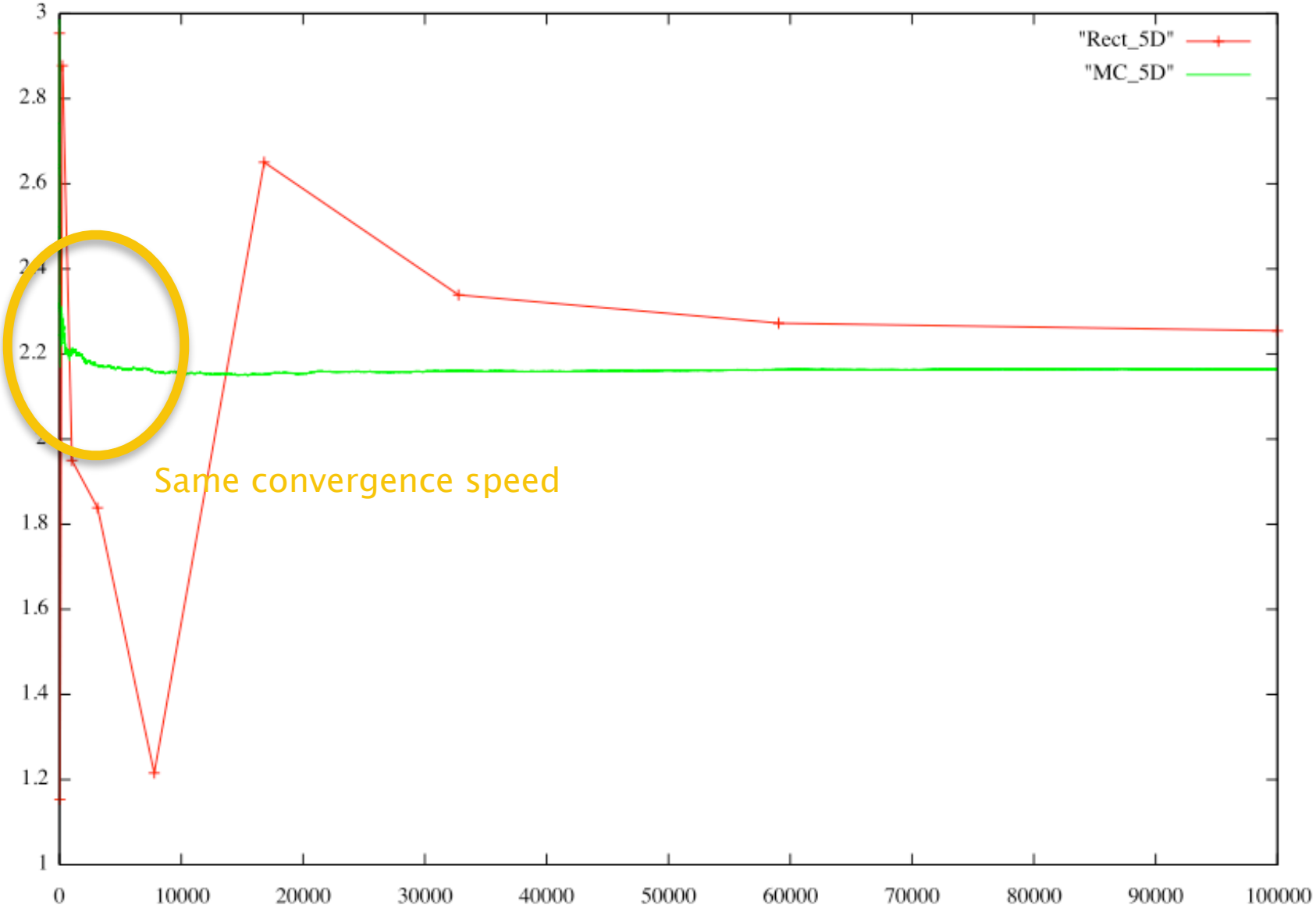
2D function: zoom-in



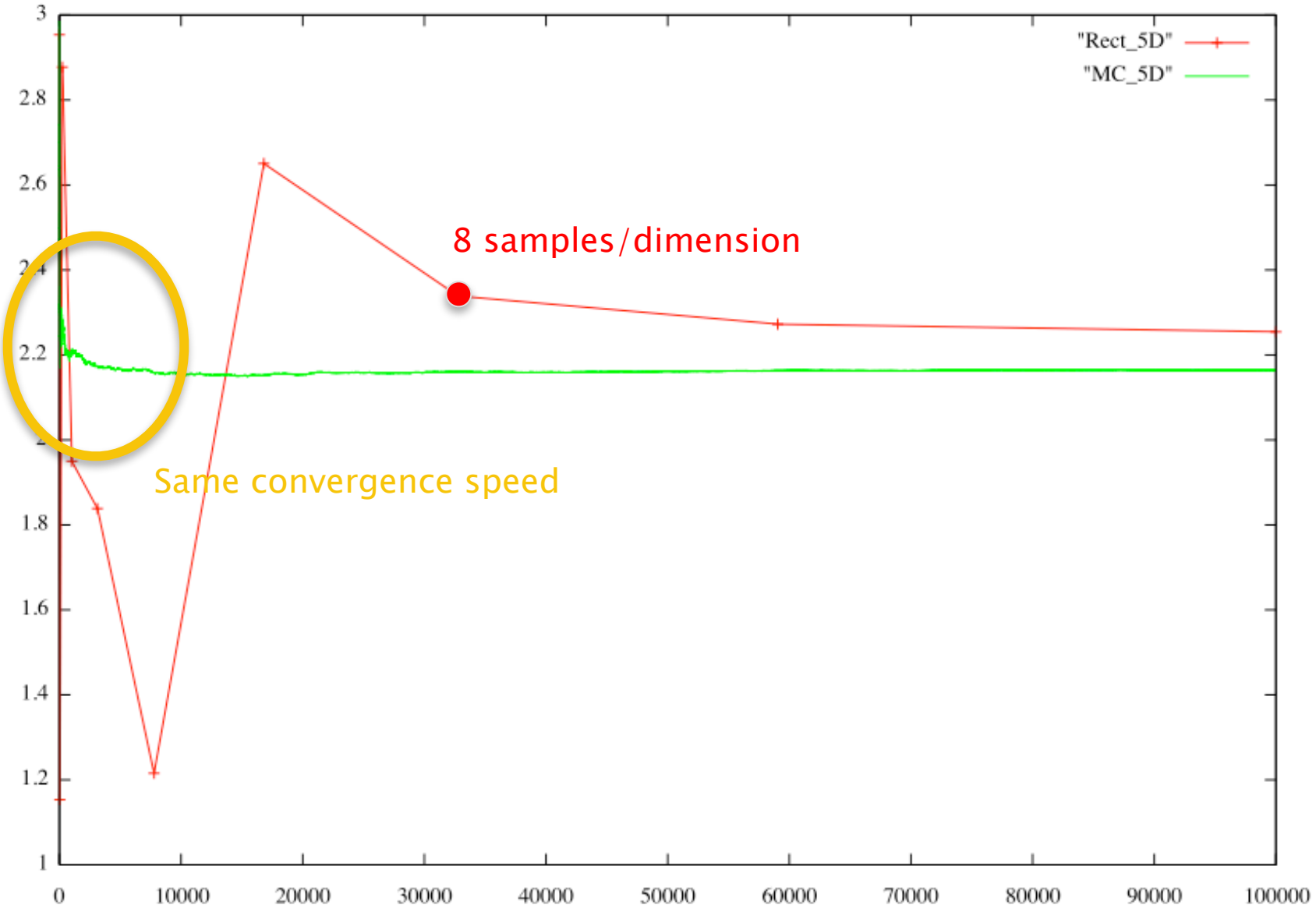
Ex.: Integral of a 5D fonction



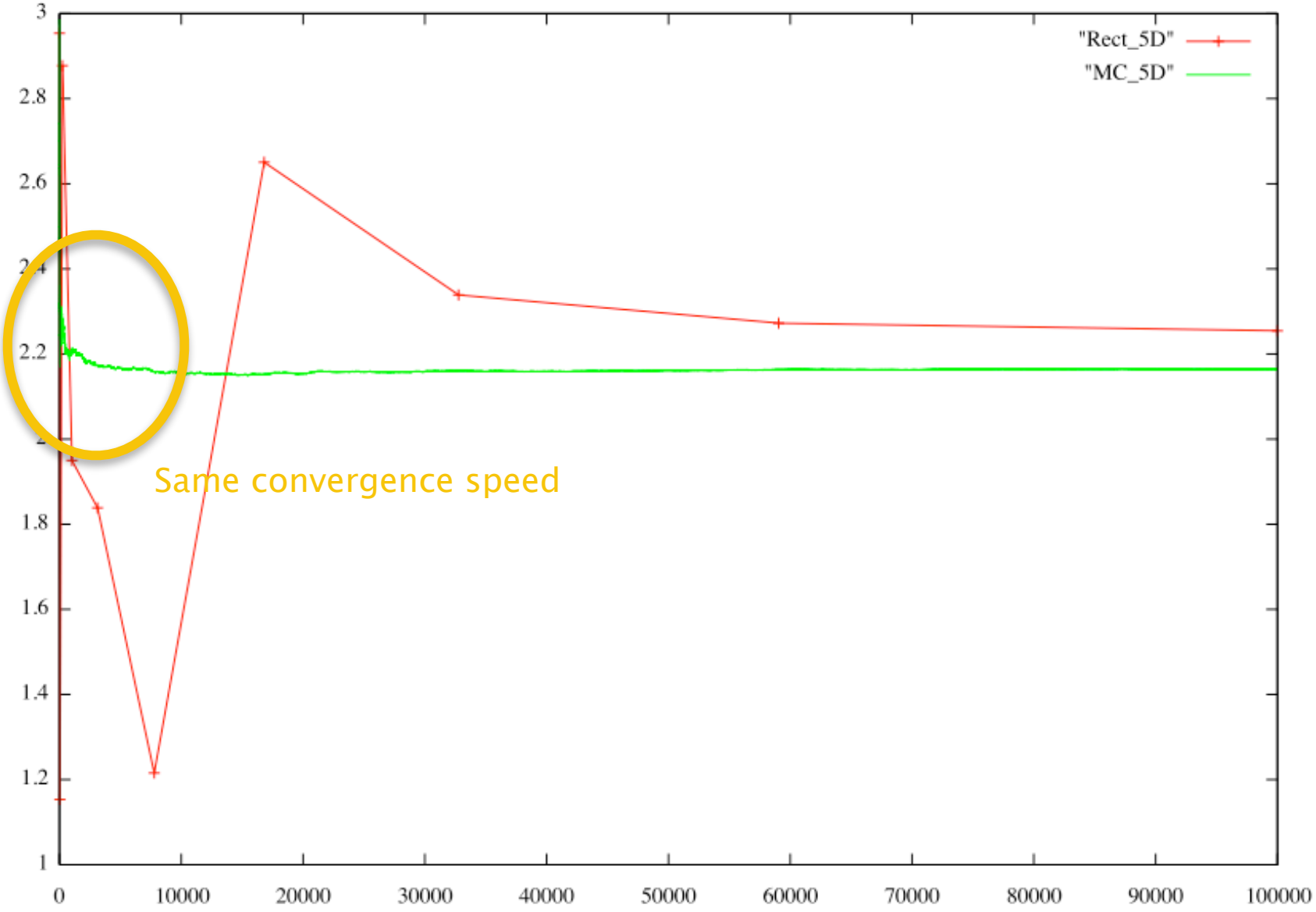
Ex.: Integral of a 5D fonction



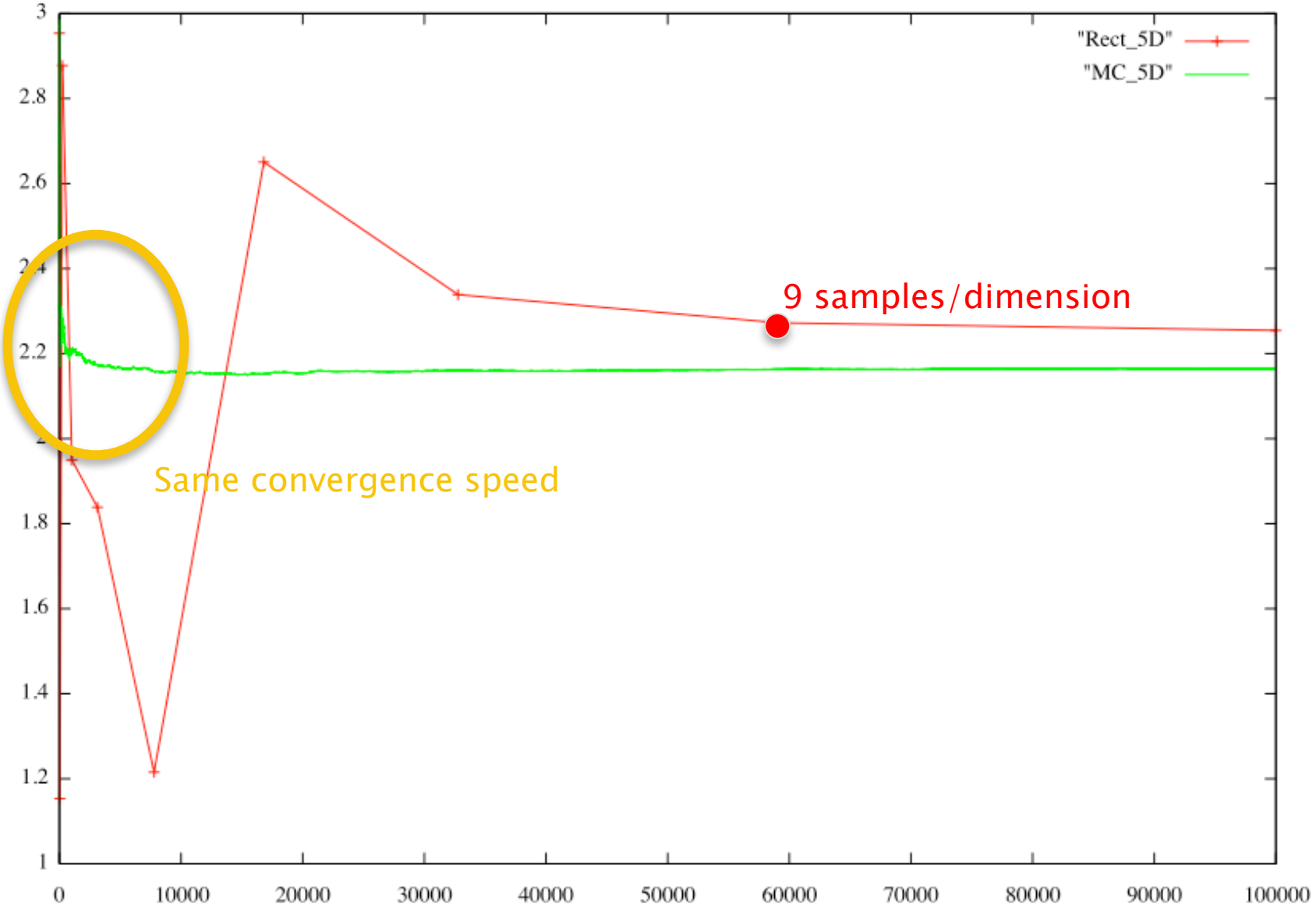
Ex.: Integral of a 5D fonction



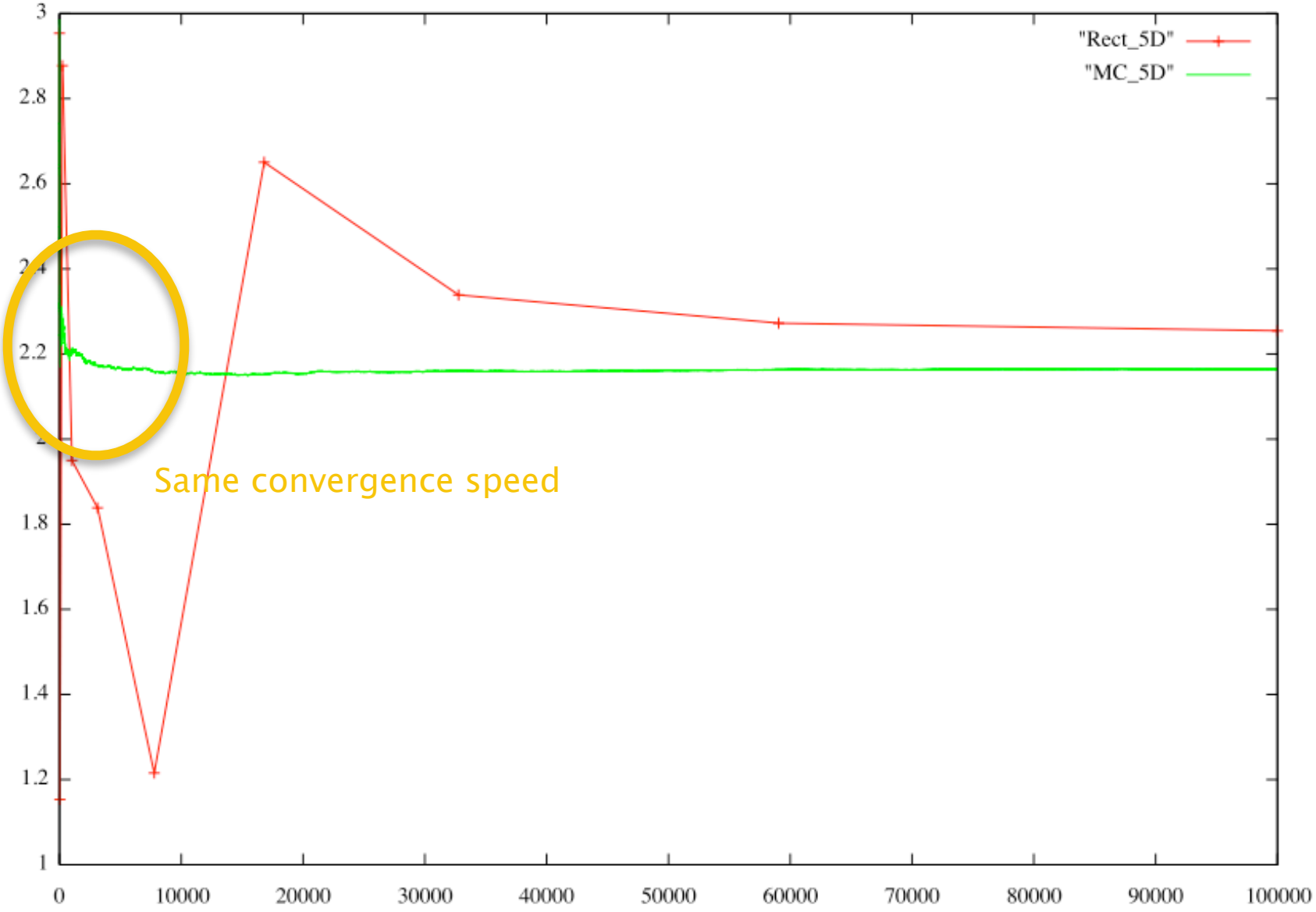
Ex.: Integral of a 5D fonction



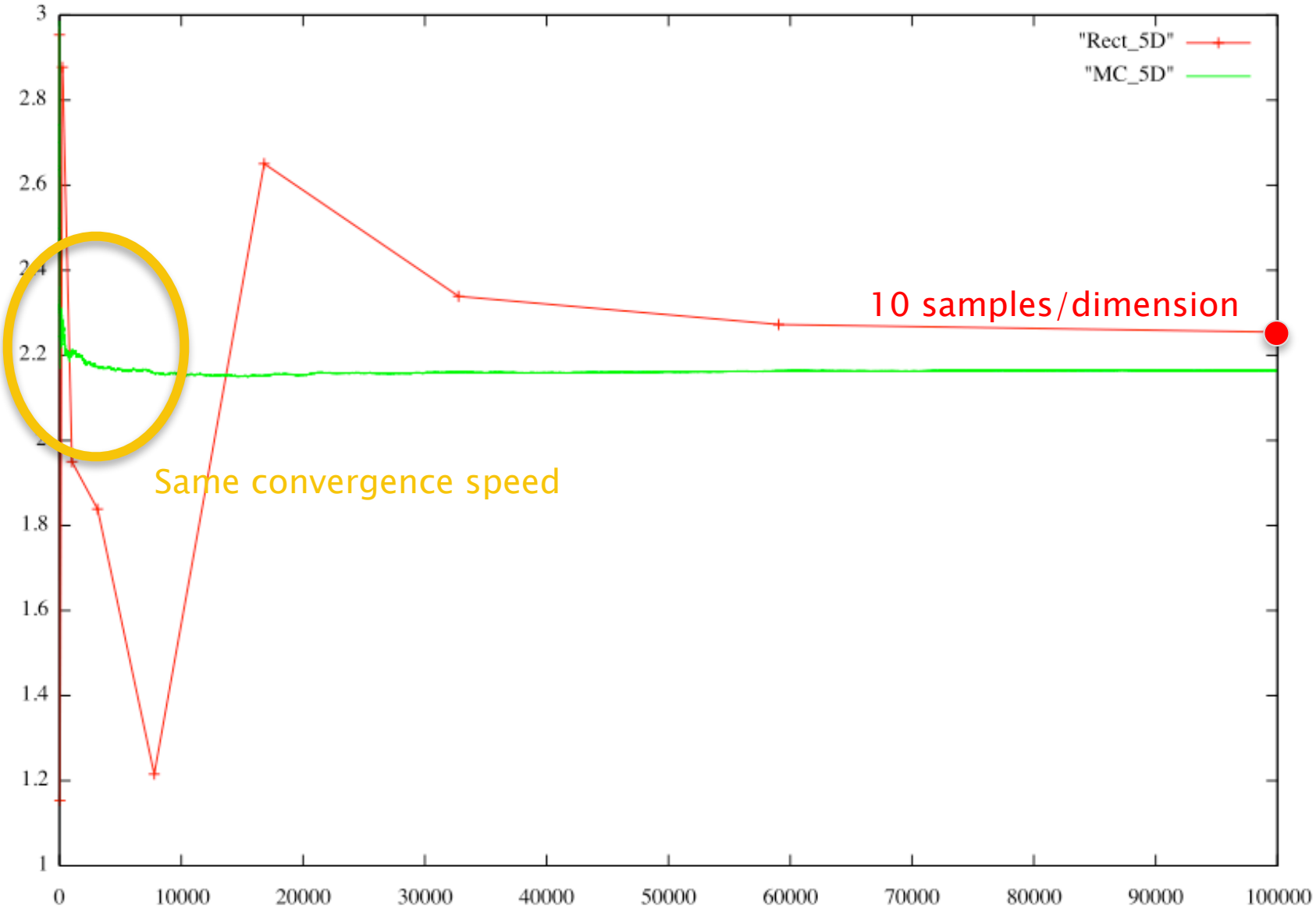
Ex.: Integral of a 5D fonction



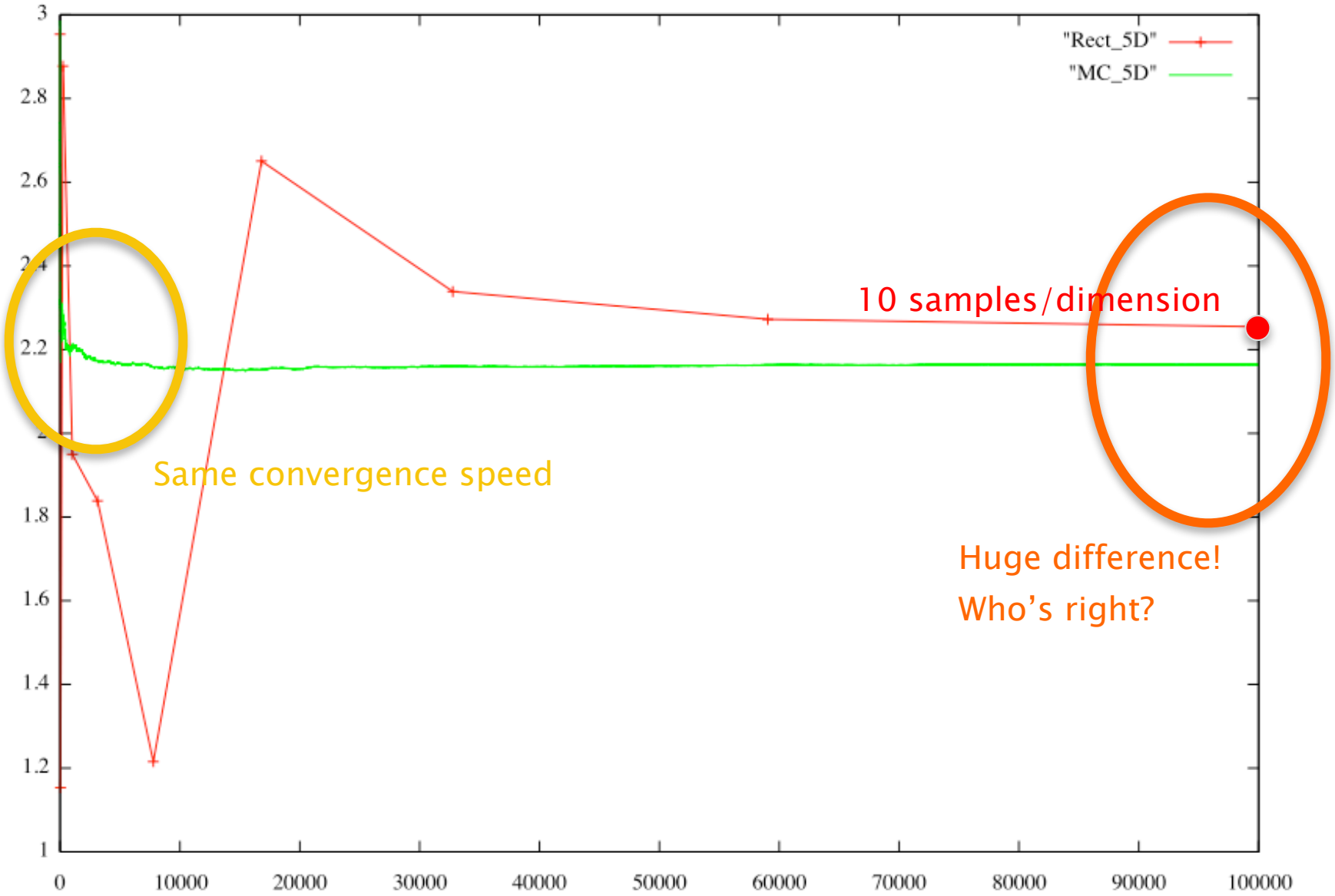
Ex.: Integral of a 5D fonction

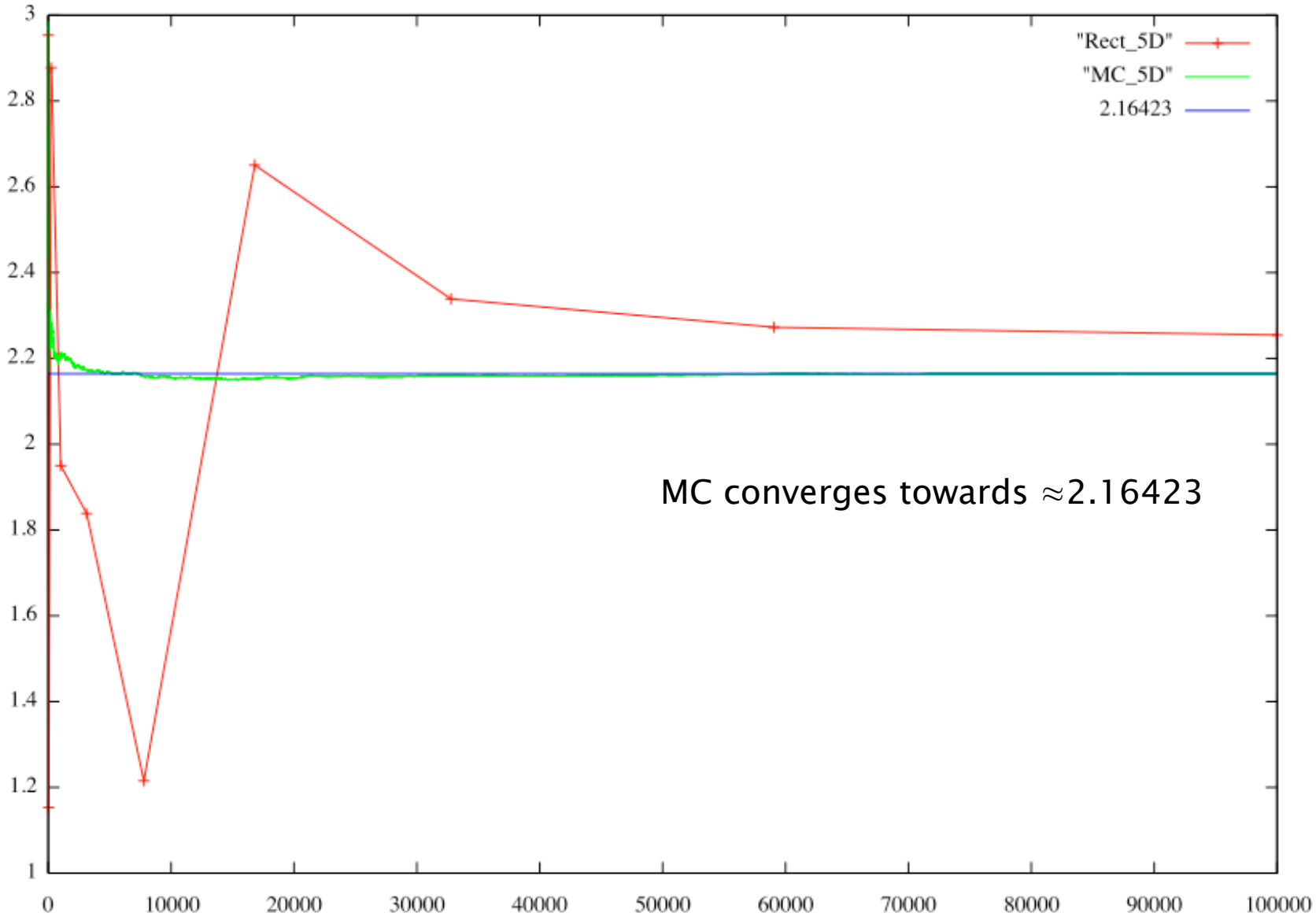


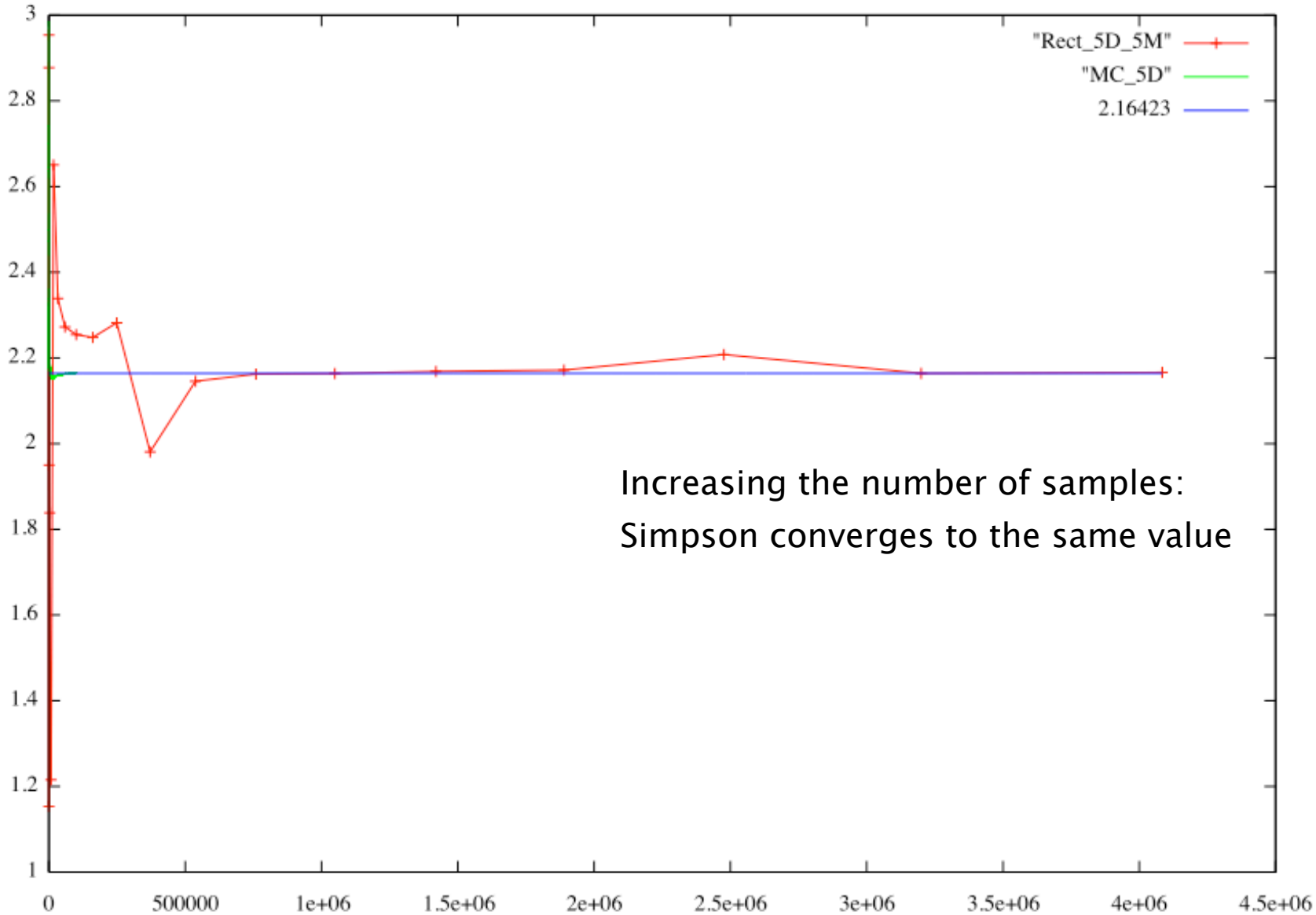
Ex.: Integral of a 5D fonction



Ex.: Integral of a 5D fonction

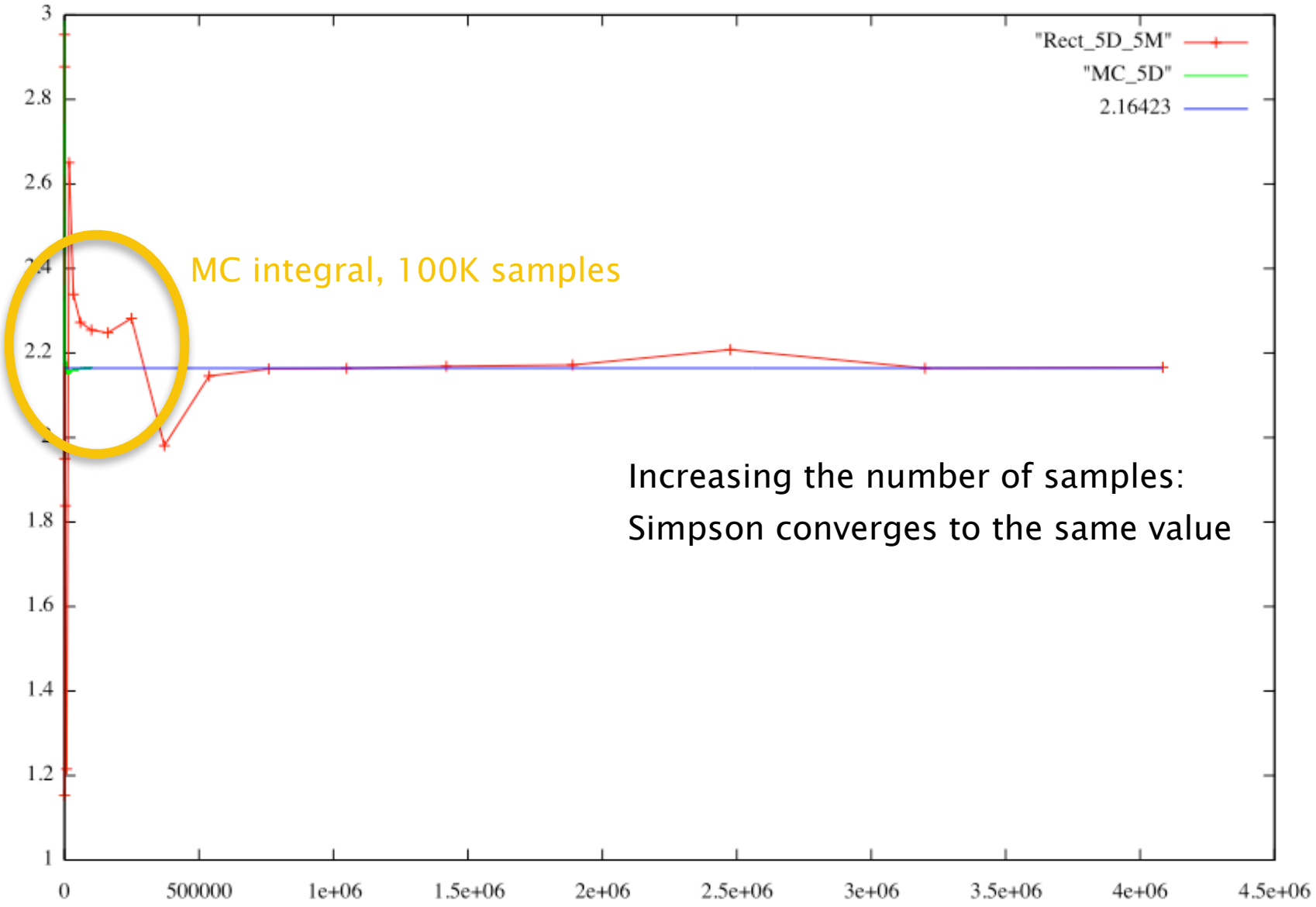



"Rect_5D"
"MC_5D"
2.16423
MC converges towards ≈2.16423
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000



Increasing the number of samples:

Simpson converges to the same value


"Rect_5D_5M"
"MC_5D"
2.16423
MC integral, 100K samples
Increasing the number of samples:
Simpson converges to the same value
3
2.8
2.6
2.2
1.8
1.6
1.4
1.2
1
0
500000
1e+06
1.5e+06
2e+06
2.5e+06
3e+06
3.5e+06
4e+06
4.5e+06

Monte Carlo integration: pros

- Few restrictions on the function to integrate
  - No requirements on continuity, regularity …
  - Only needs sampling on a single point
- Same convergence rate on higher dimensions
- Very simple

Monte Carlo integration: cons

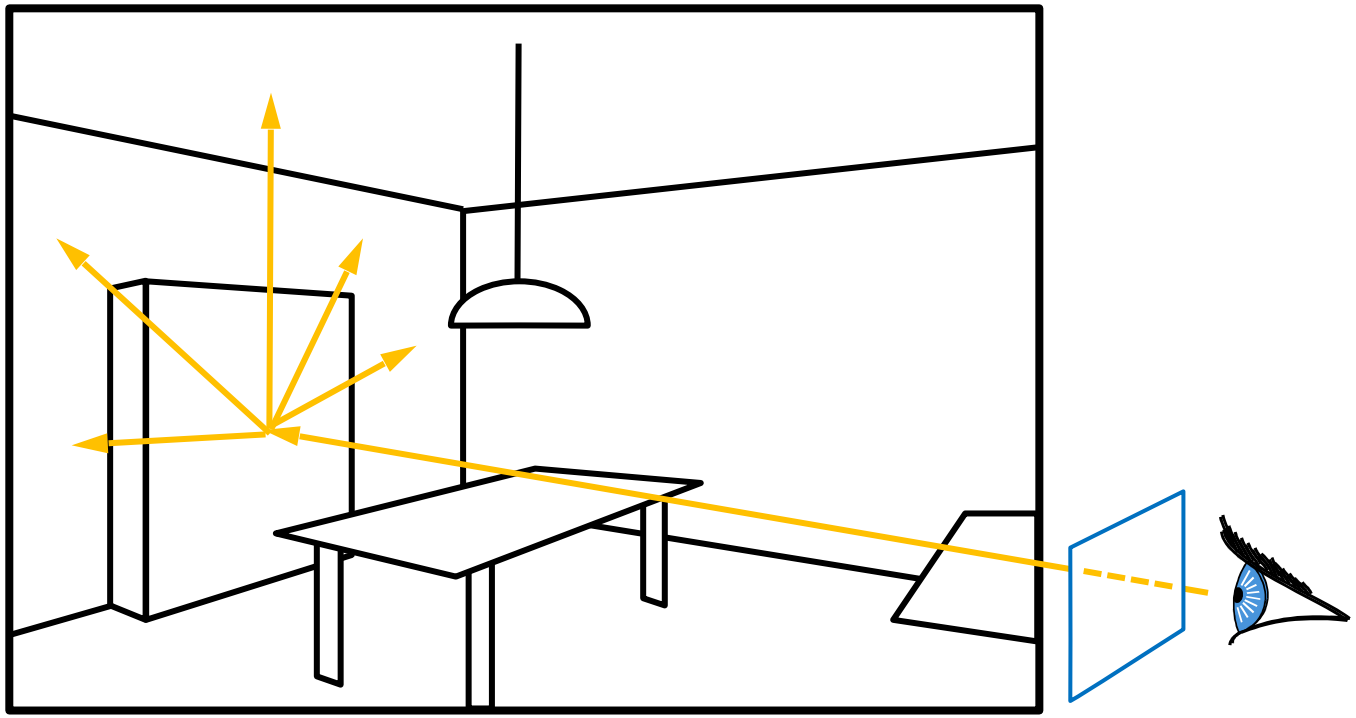
- Noise
- Slow convergence
- Efficient implementation harder

Global Illumination Simulation

- Monte-Carlo methods for illumination
- How?
- Each pixel is an integral

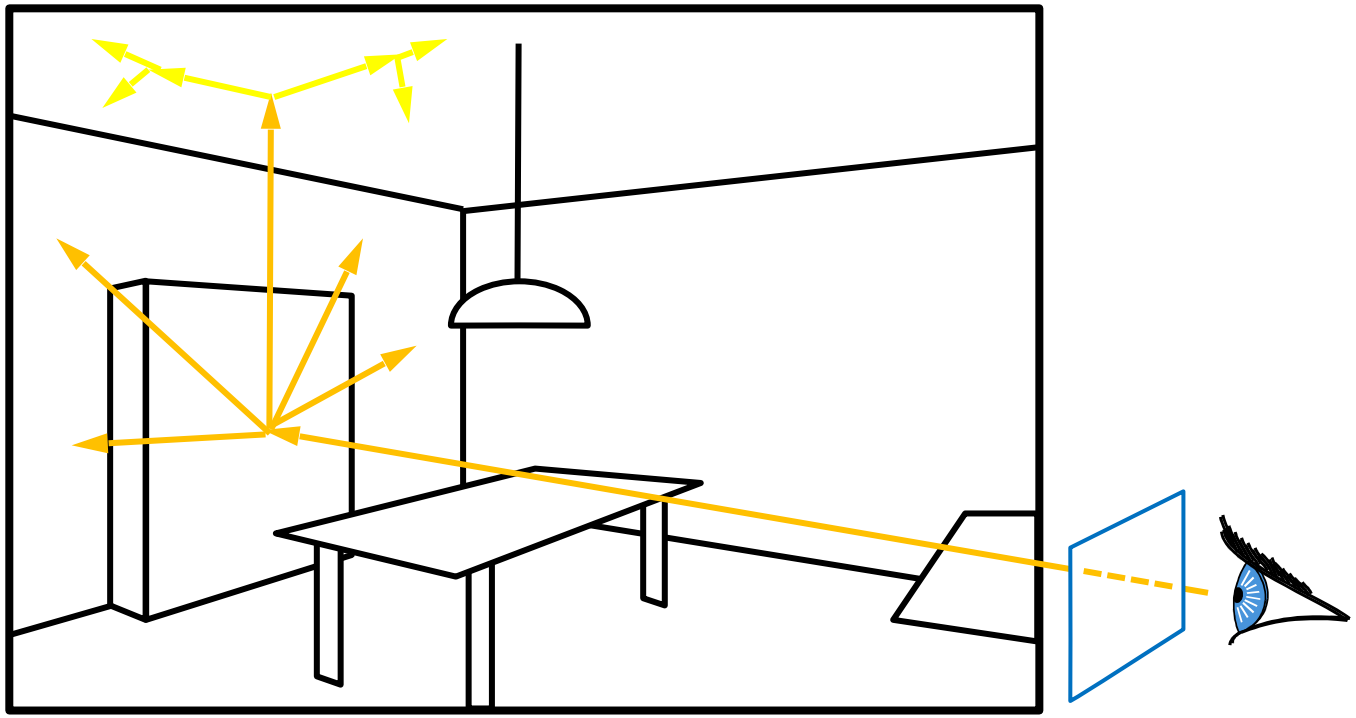
Monte Carlo methods

- One ray for every pixel
- For each visible point: random sampling of rays, accumulate radiance



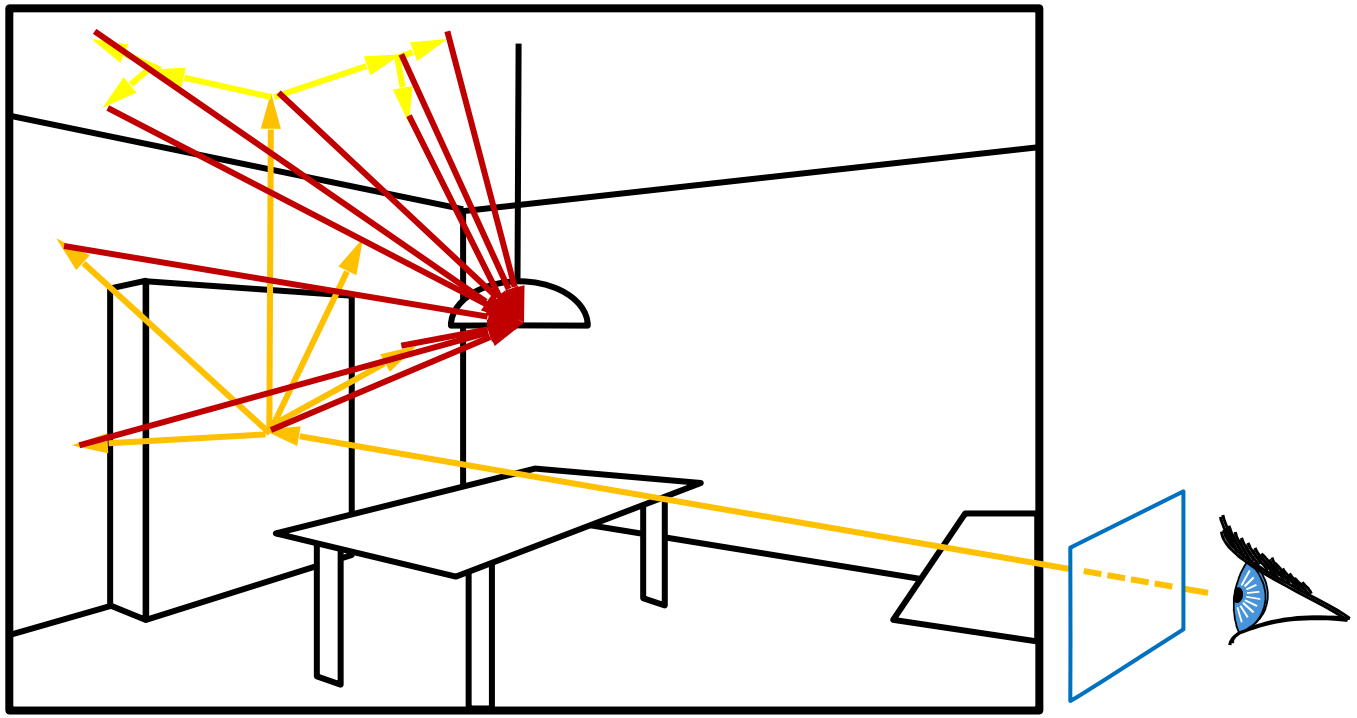
Monte Carlo methods

- One ray for every pixel
- For each visible point : random sampling of rays, accumulate radiance
- Keep going, recursively



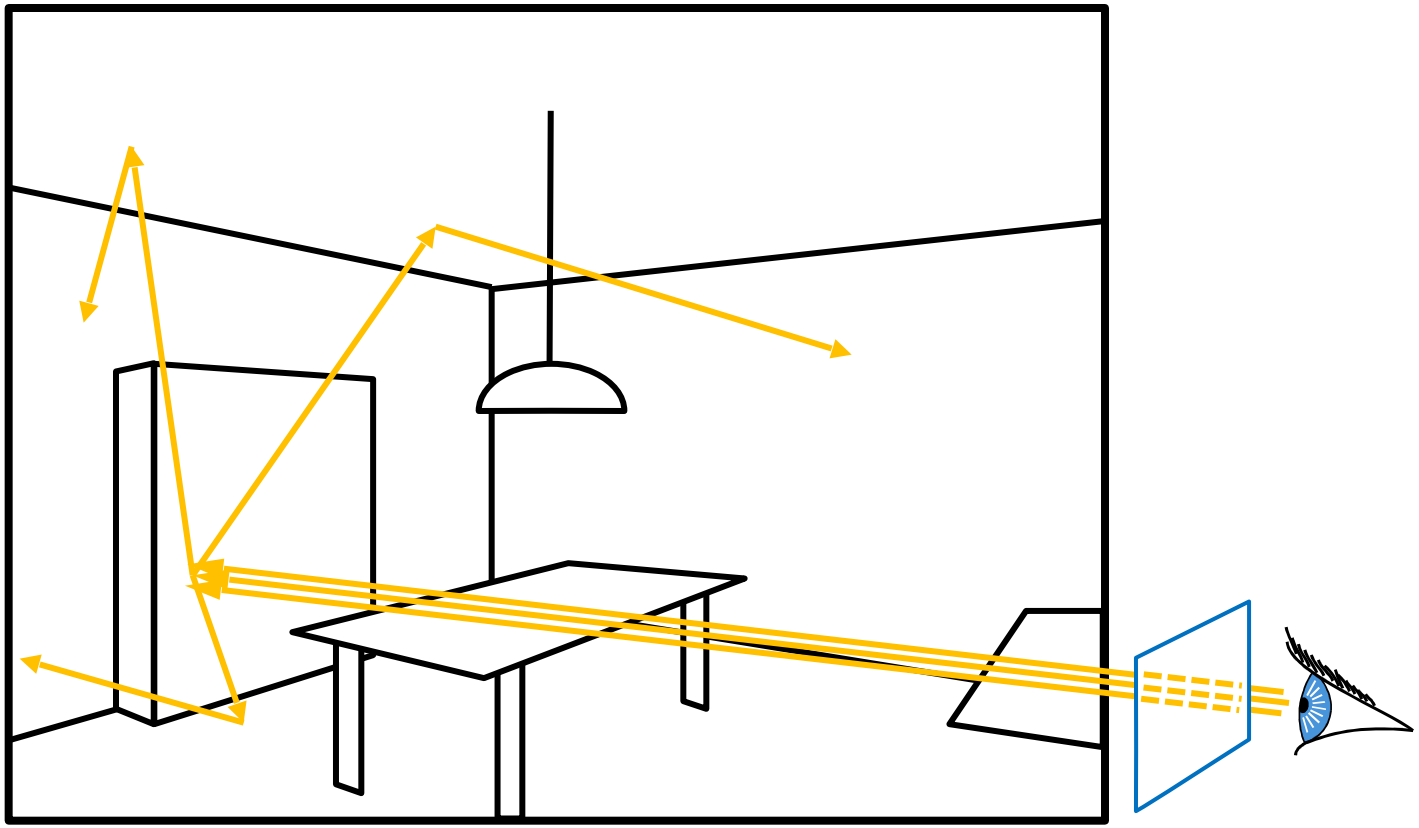
Monte Carlo methods

- One ray for every pixel
- For each visible point : random sampling of rays, accumulate radiance
- Keep going, recursively
- Sample the light source each time



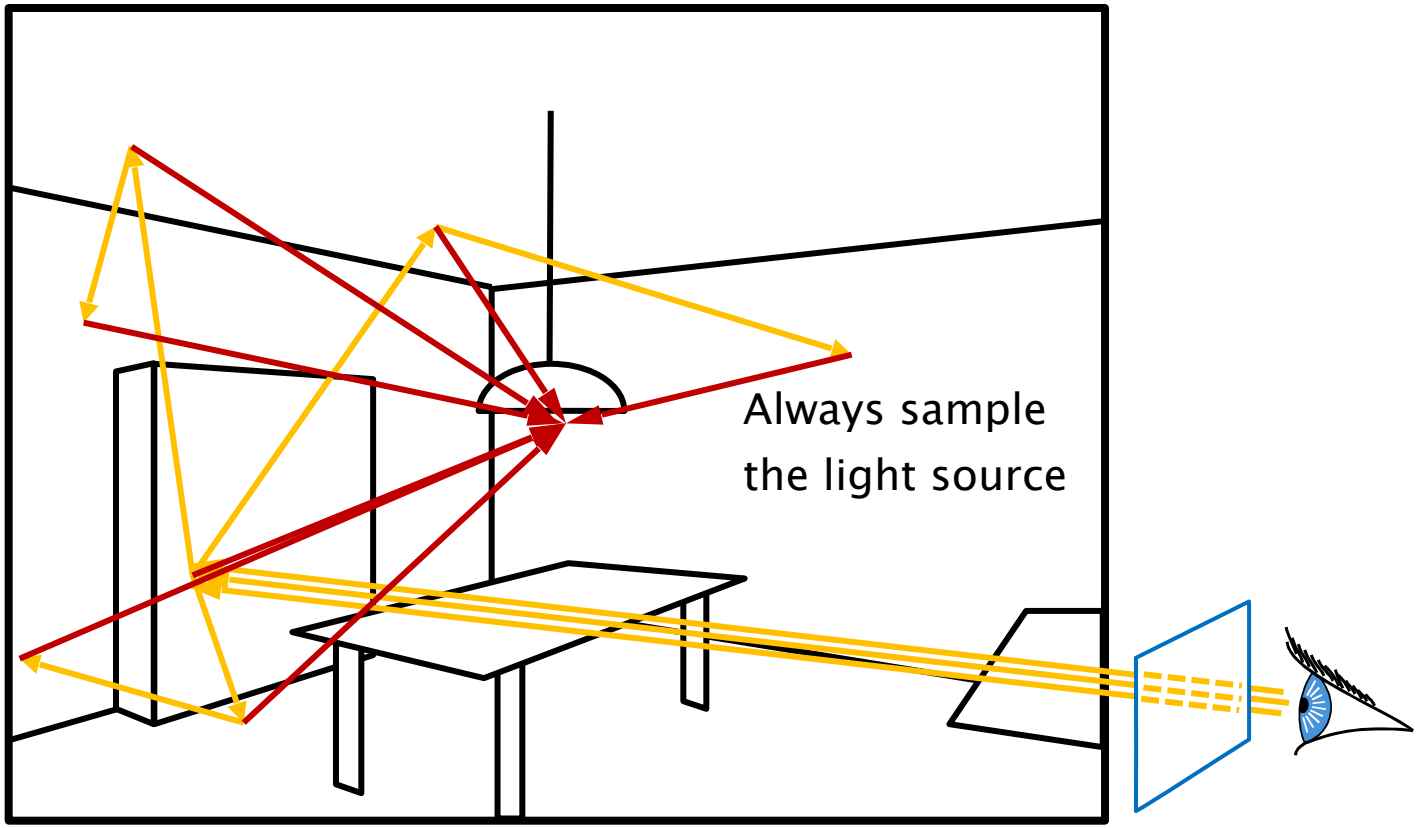
Monte Carlo Path Tracing

- Trace only one ray at each recursion
- But trace several (hundreds of) primary rays for each pixel

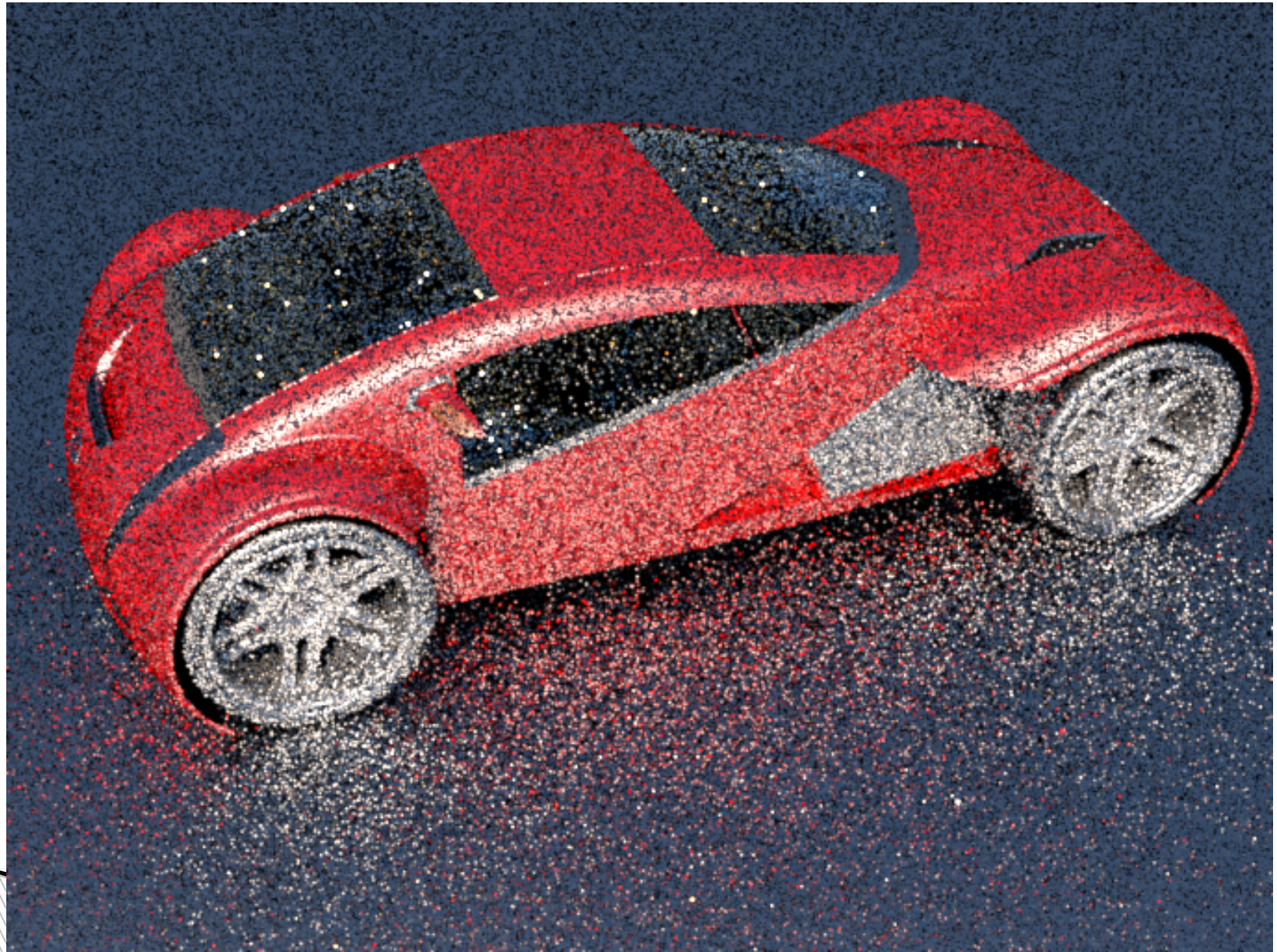


Monte Carlo Path Tracing

- Trace only one ray at each recursion
- But trace several (hundreds of) primary rays for each pixel

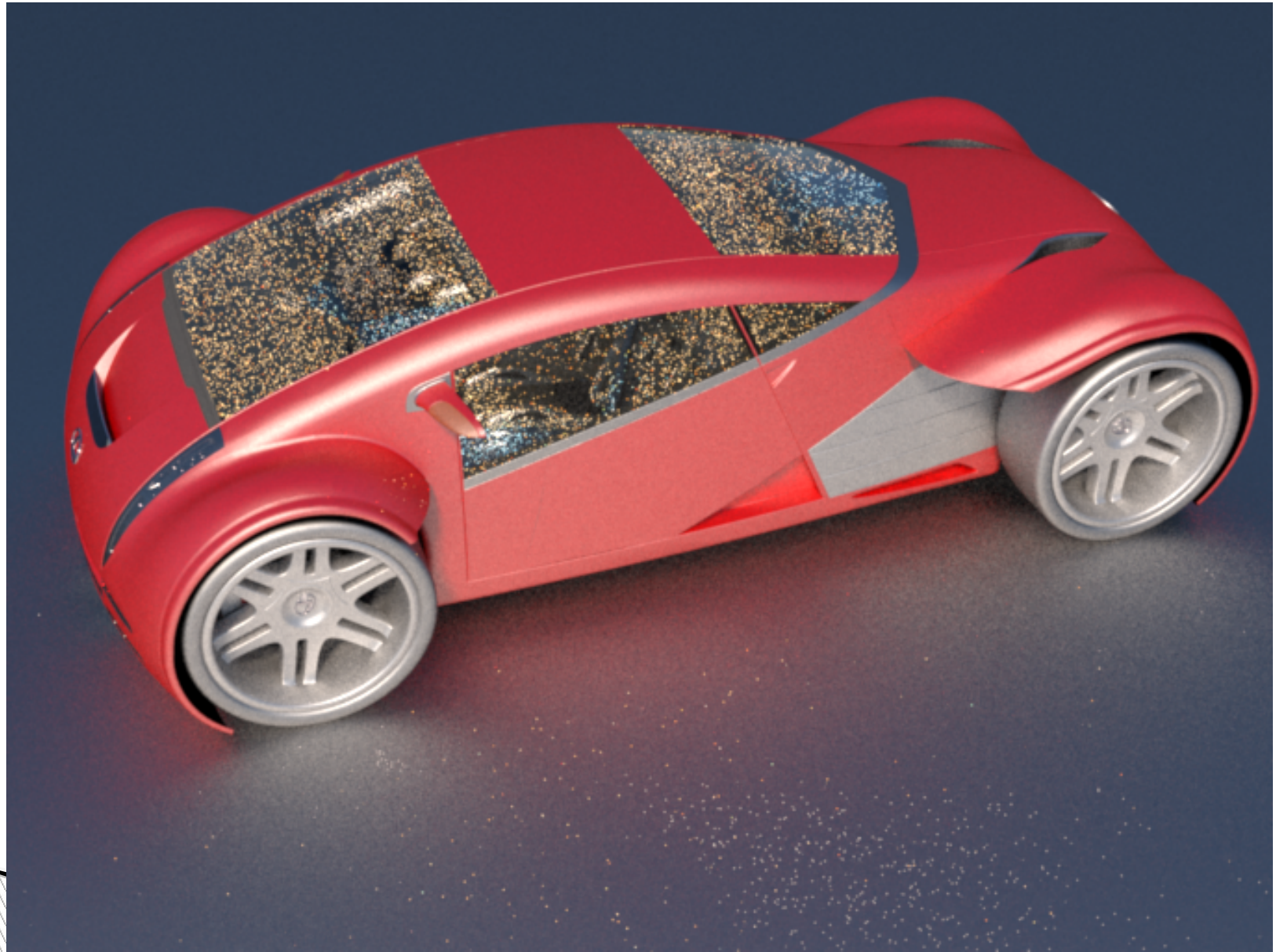


Path Tracing, 1spp (0.7 s)

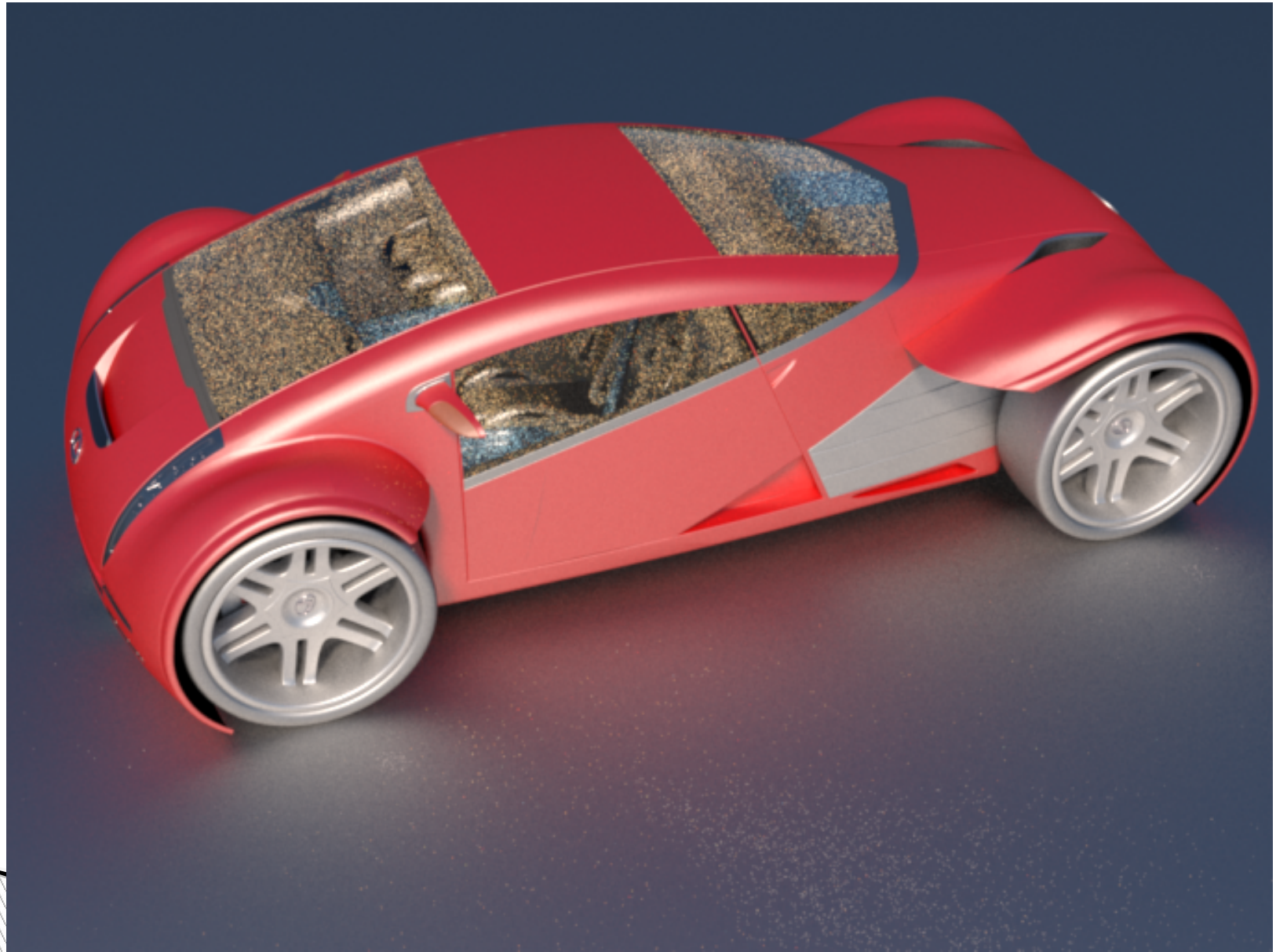


Path Tracing, 32 spp (21 s)

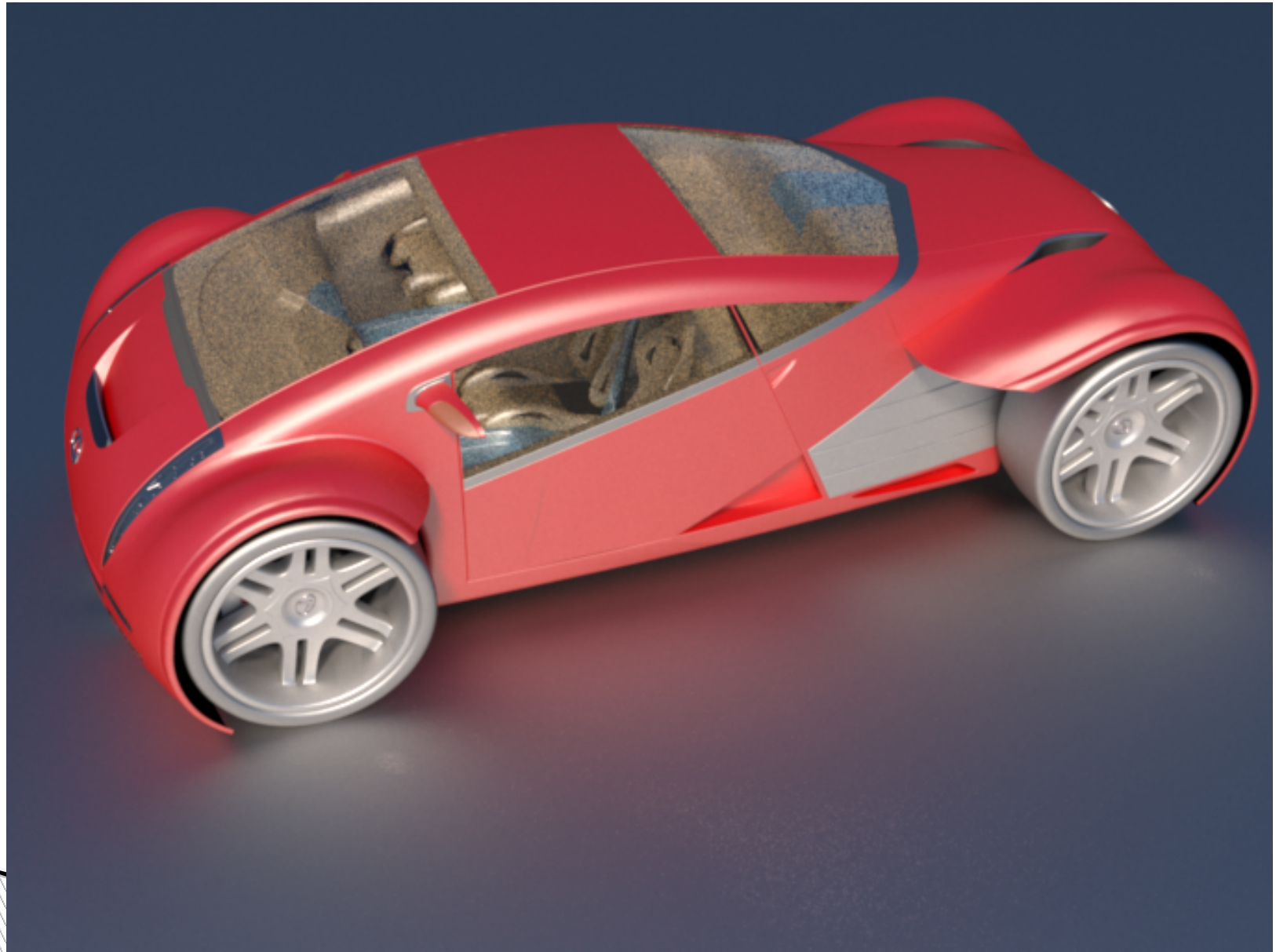
Path Tracing, 256 spp (3 mn)



Path Tracing, 1024 spp (12.4 mn)



PT, adaptive, 256-8192 spp (53 mn)

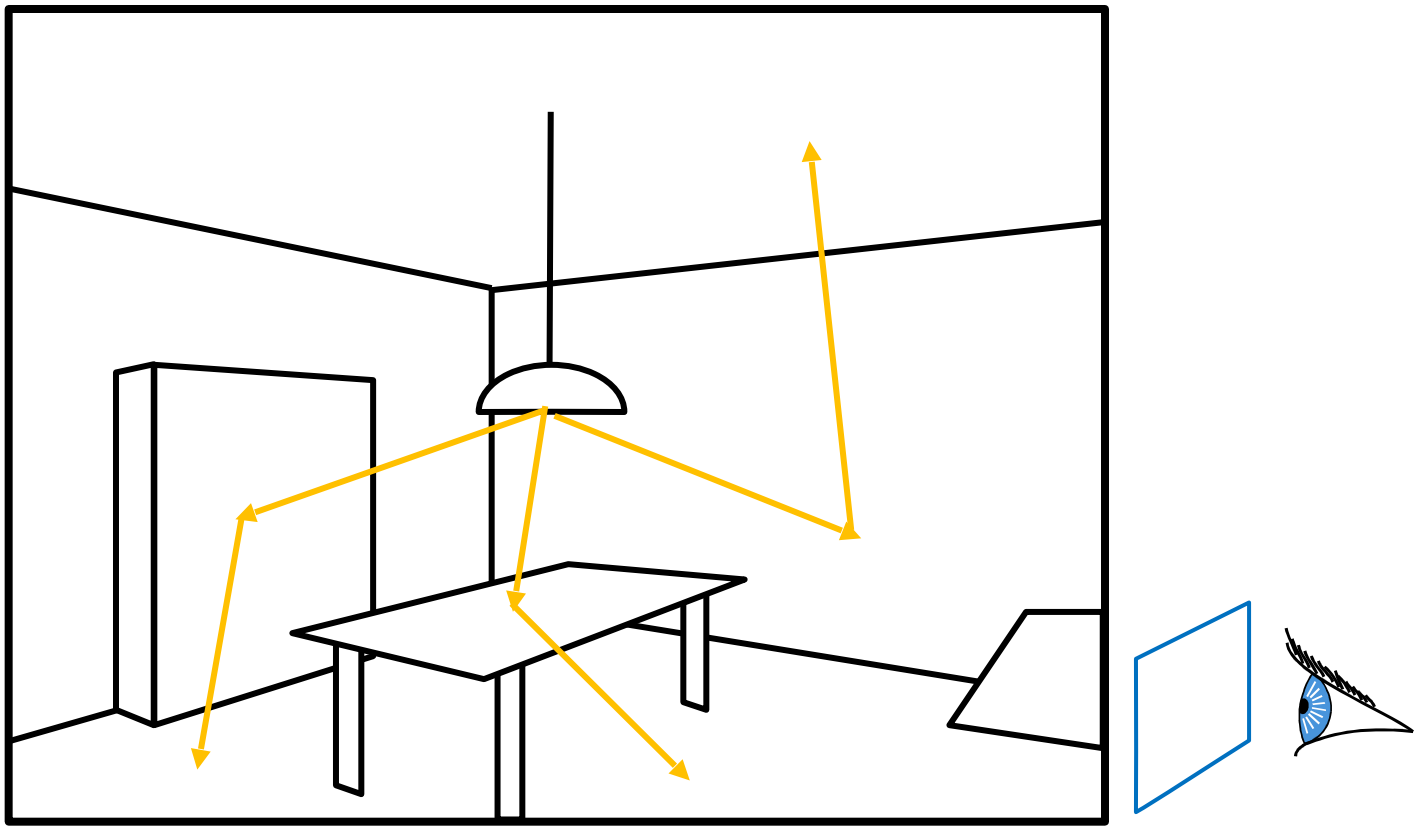


What's happening?

- Light source: sun + sky
- Fast convergence outside
- Much slower inside
  - Light is refracted by windows
  - No direct path towards the light source

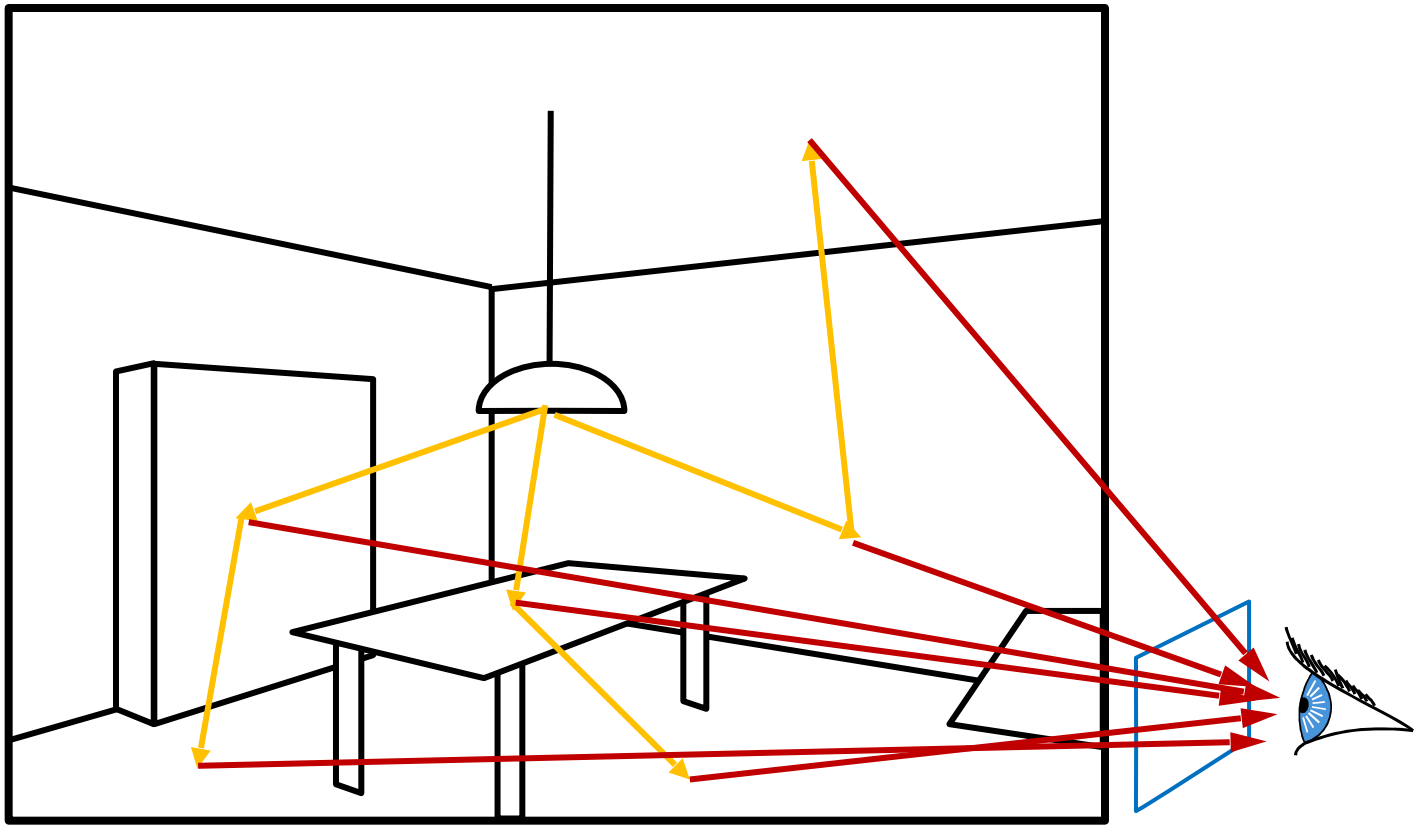
Monte Carlo Light Path Tracing

- Start from the light source
- Send particles, let them bounce around

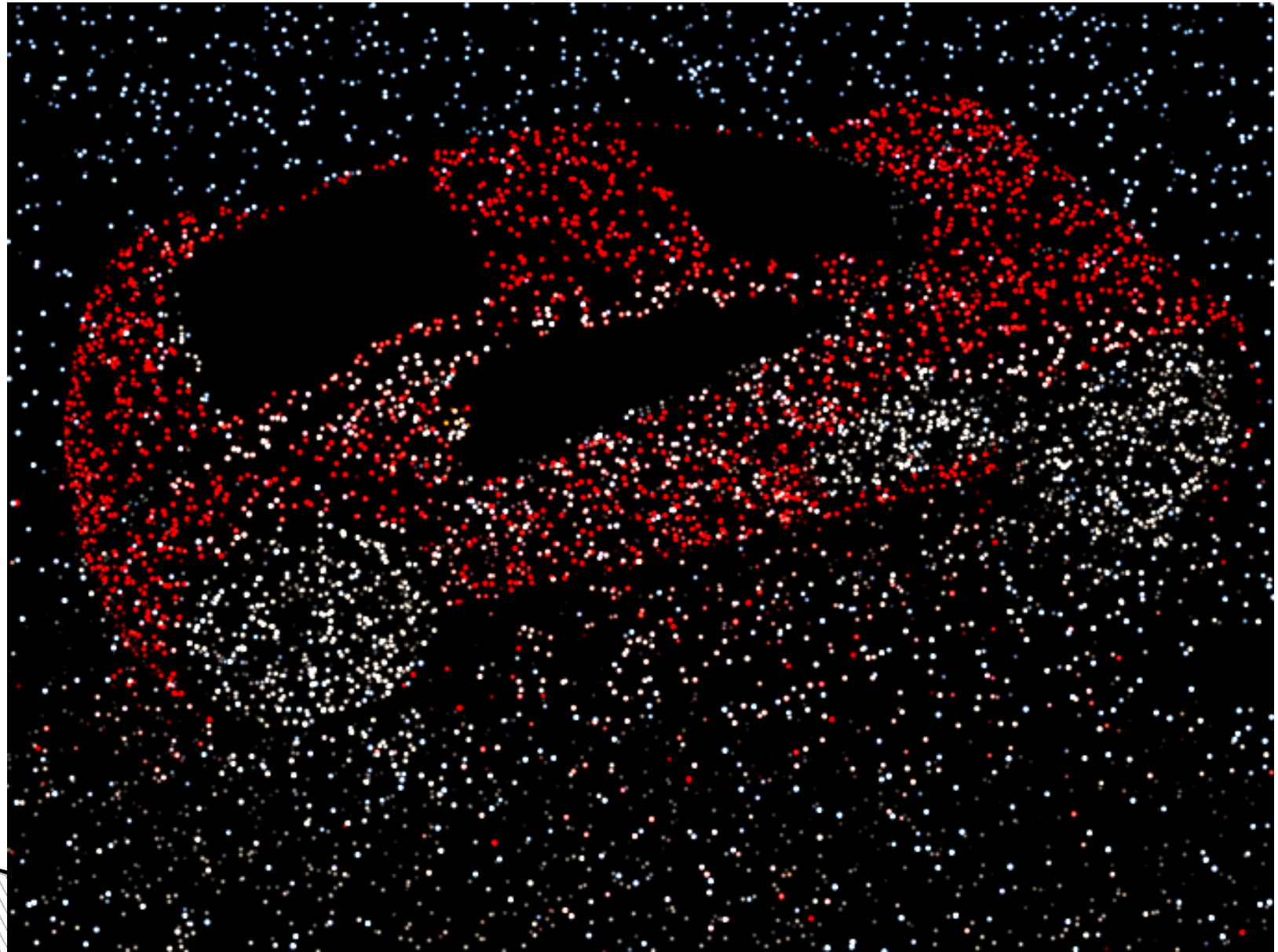


Monte Carlo Light Path Tracing

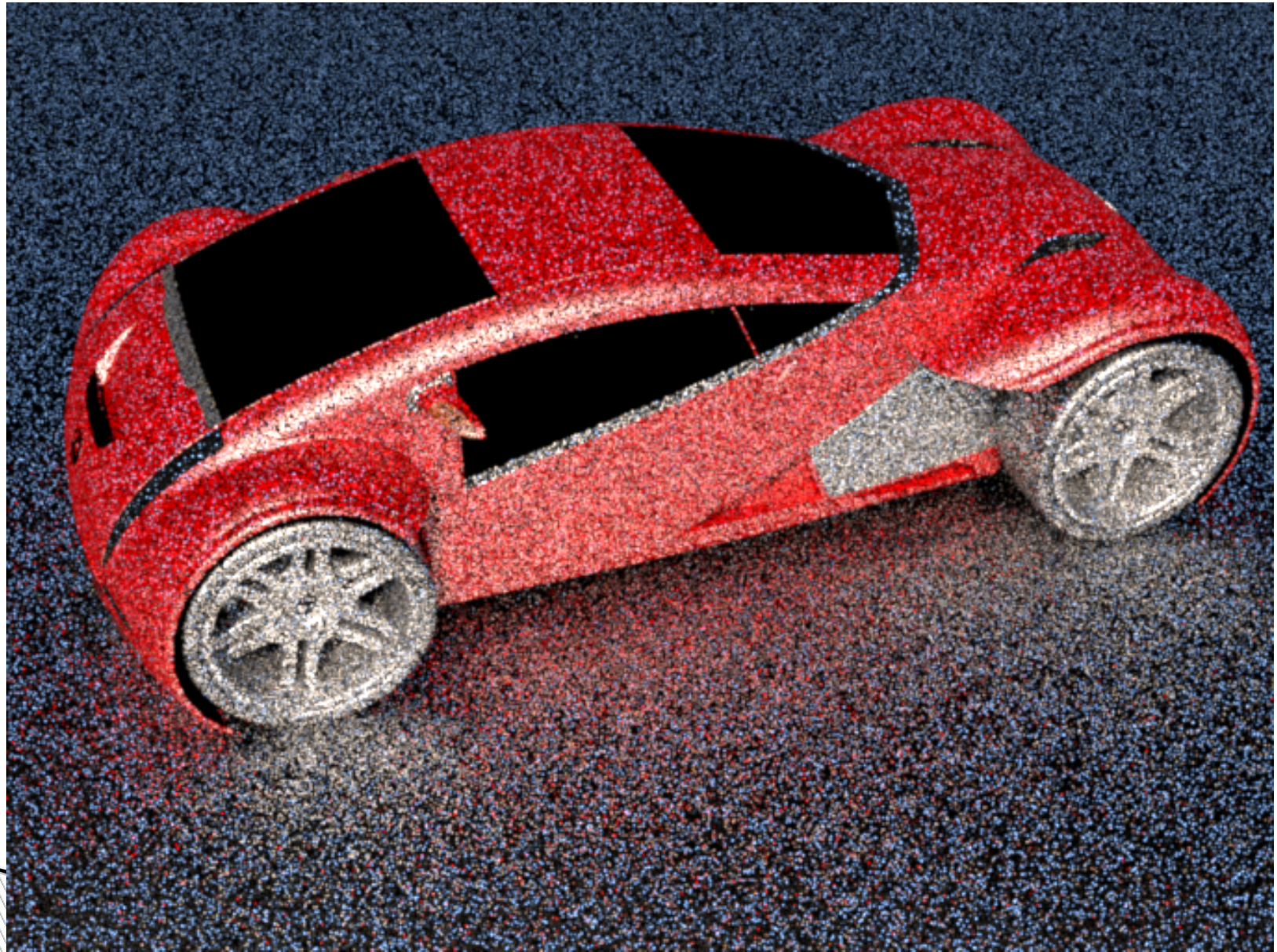
- Start from the light source
- Send particles, let them bounce around
- Sample the camera directly



Light PT: 700M particles, 4 mn

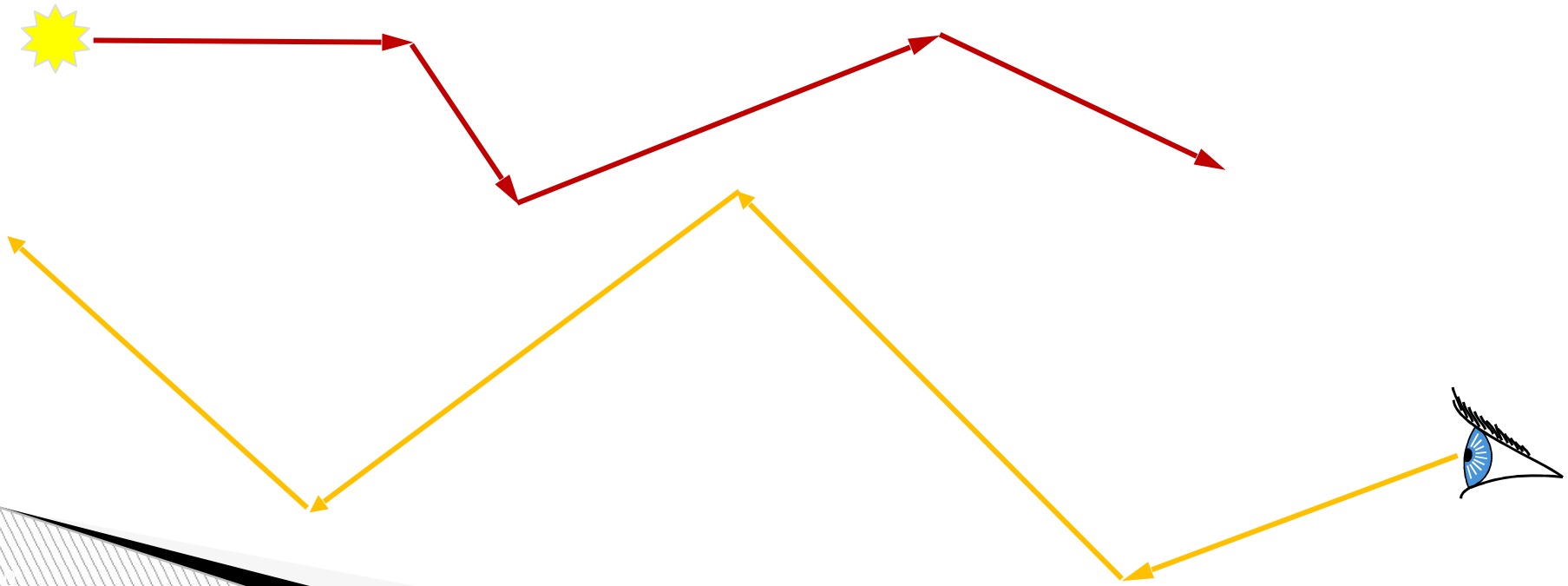


Light PT: 70G particles, 6.7 h



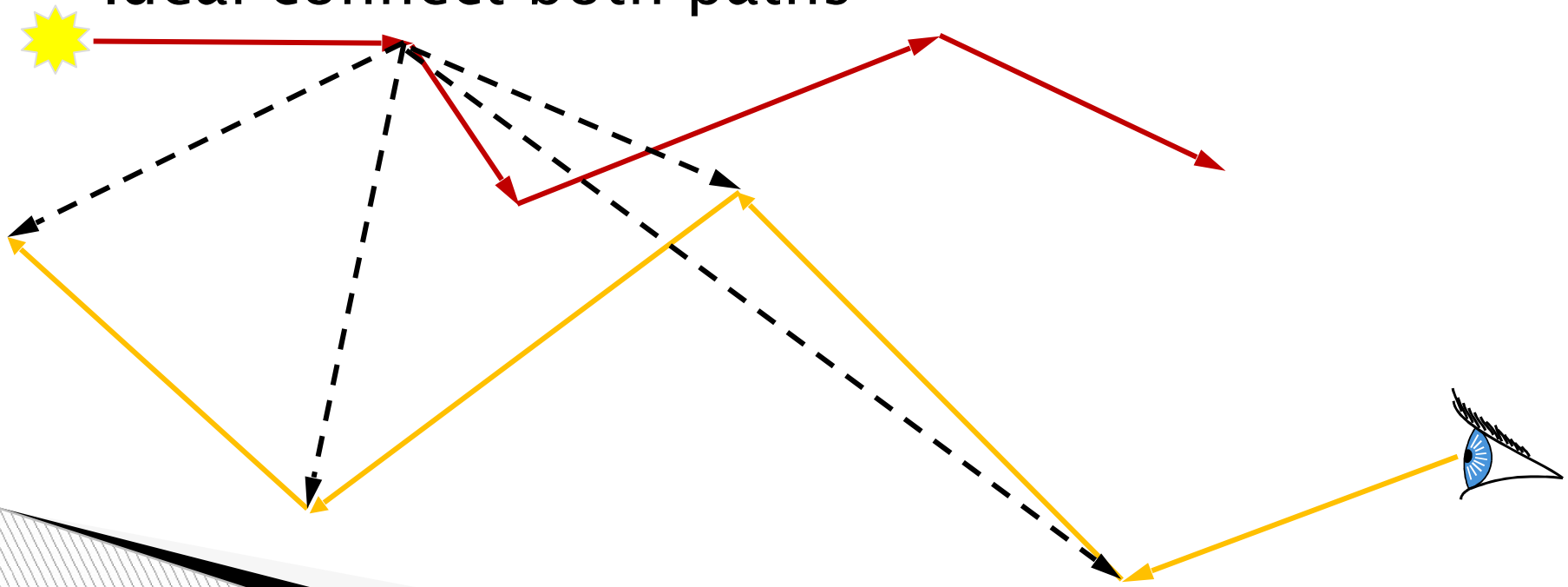
Bi-directional path tracing

- Path-tracing is not enough
  - Need a connection towards light source, camera
- Connection between eye-paths and light-paths?
- Idea: connect both paths



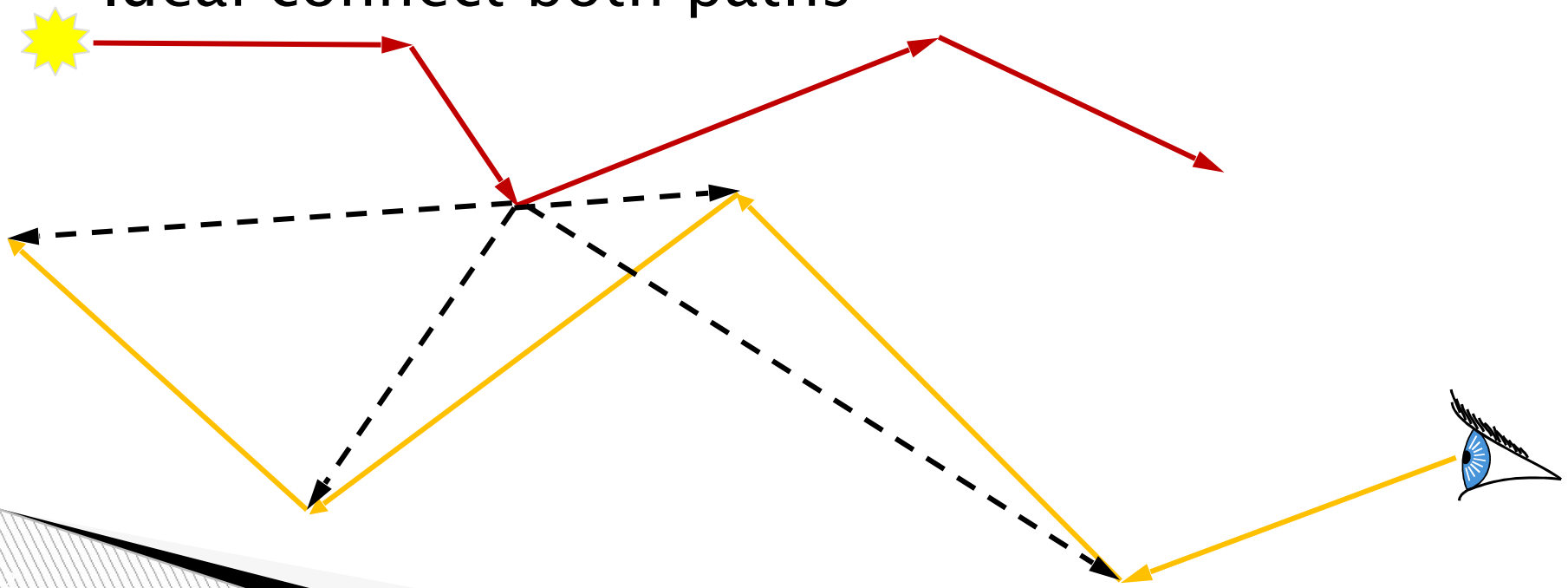
Bi-directional path tracing

- Path-tracing is not enough
  - Need a connection towards light source, camera
- Connection between eye-paths and light-paths?
- Idea: connect both paths



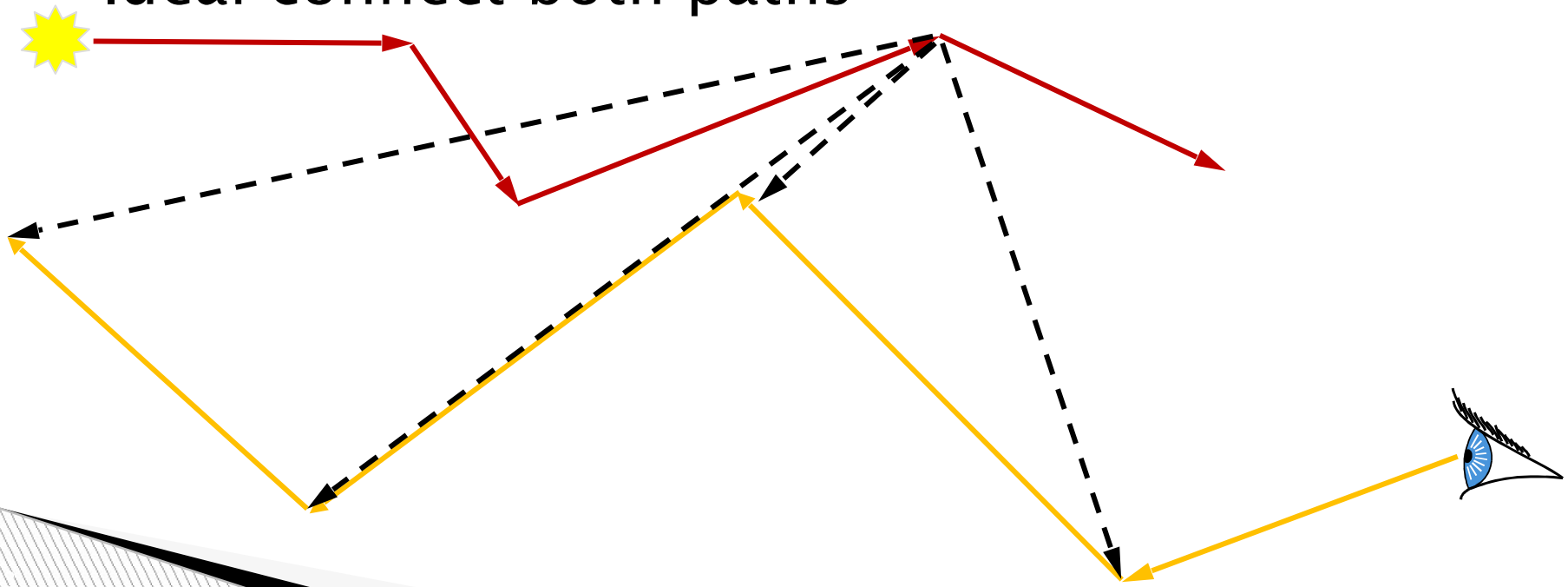
Bi-directional path tracing

- Path-tracing is not enough
  - Need a connection towards light source, camera
- Connection between eye-paths and light-paths?
- Idea: connect both paths



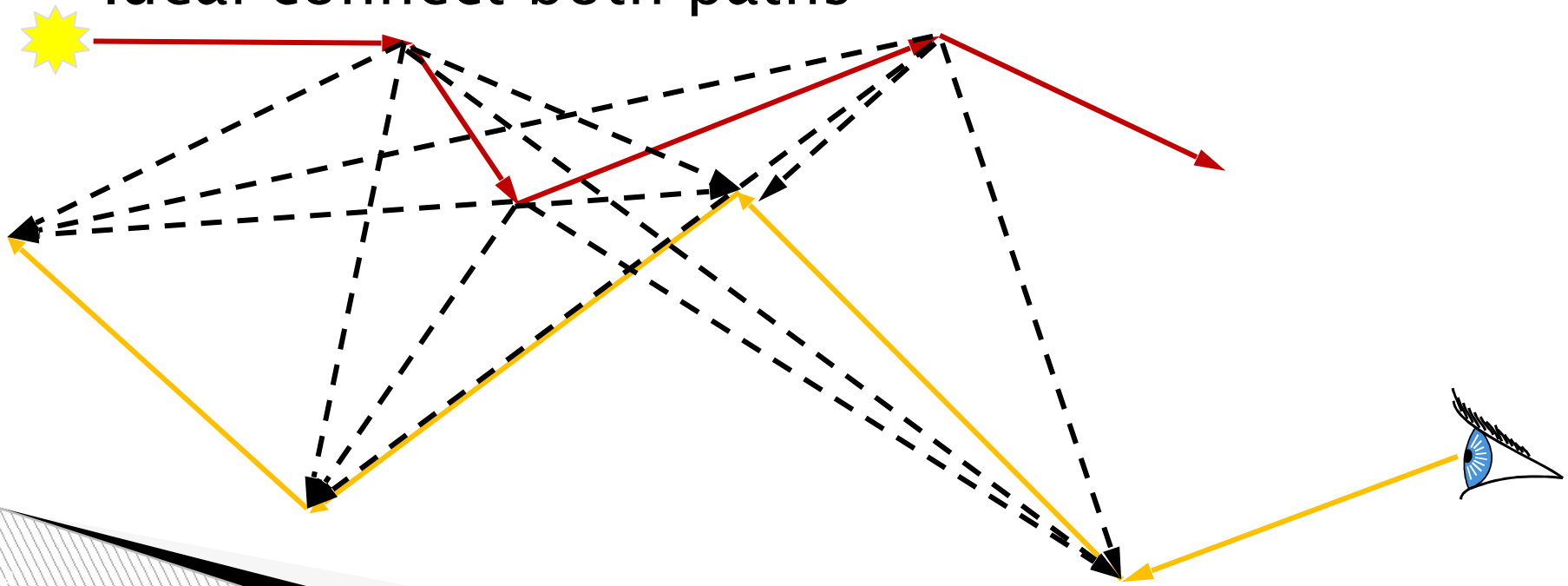
Bi-directional path tracing

- Path-tracing is not enough
  - Need a connection towards light source, camera
- Connection between eye-paths and light-paths?
- Idea: connect both paths



Bi-directional path tracing

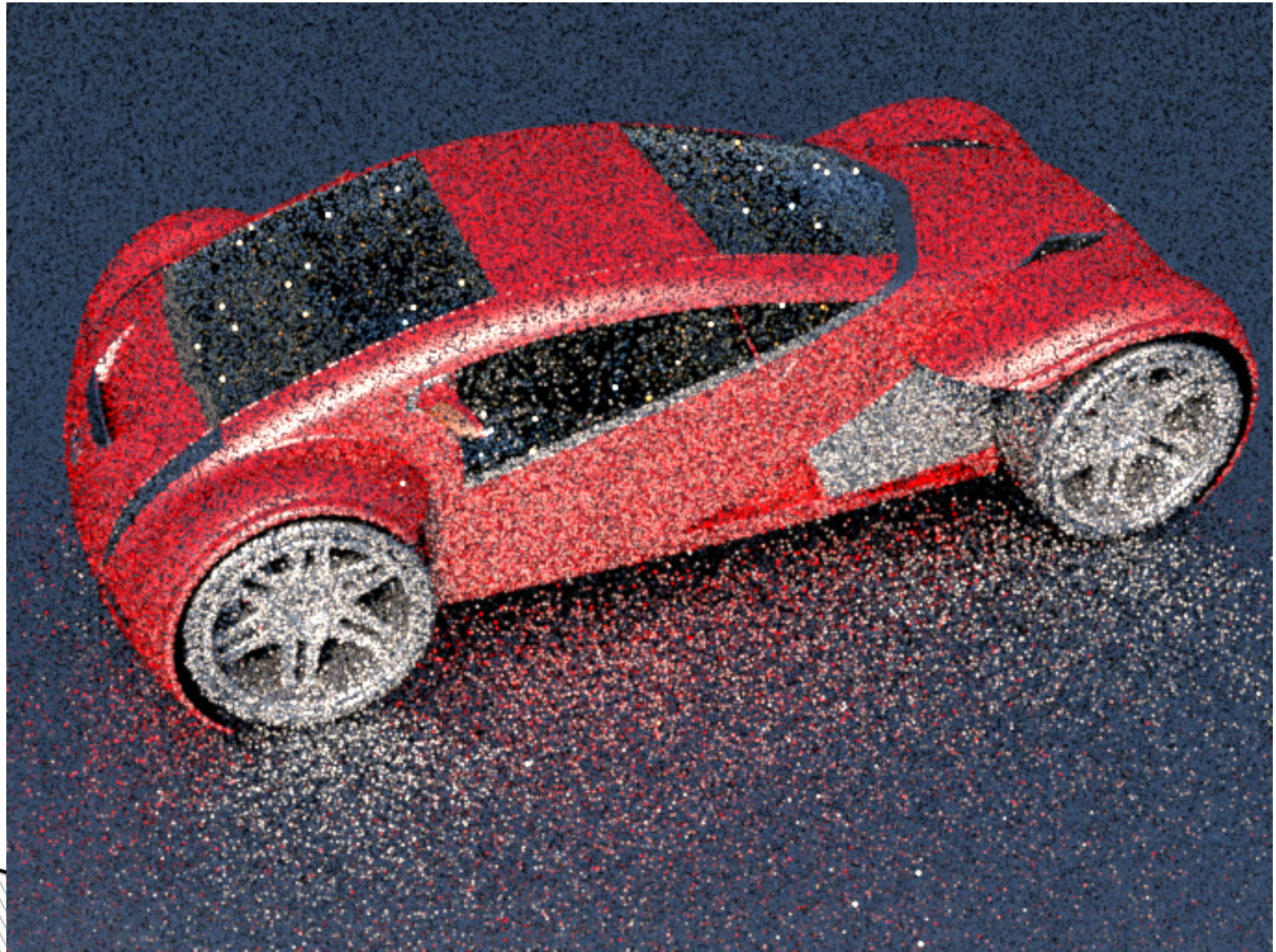
- Path-tracing is not enough
  - Need a connection towards light source, camera
- Connection between eye-paths and light-paths?
- Idea: connect both paths



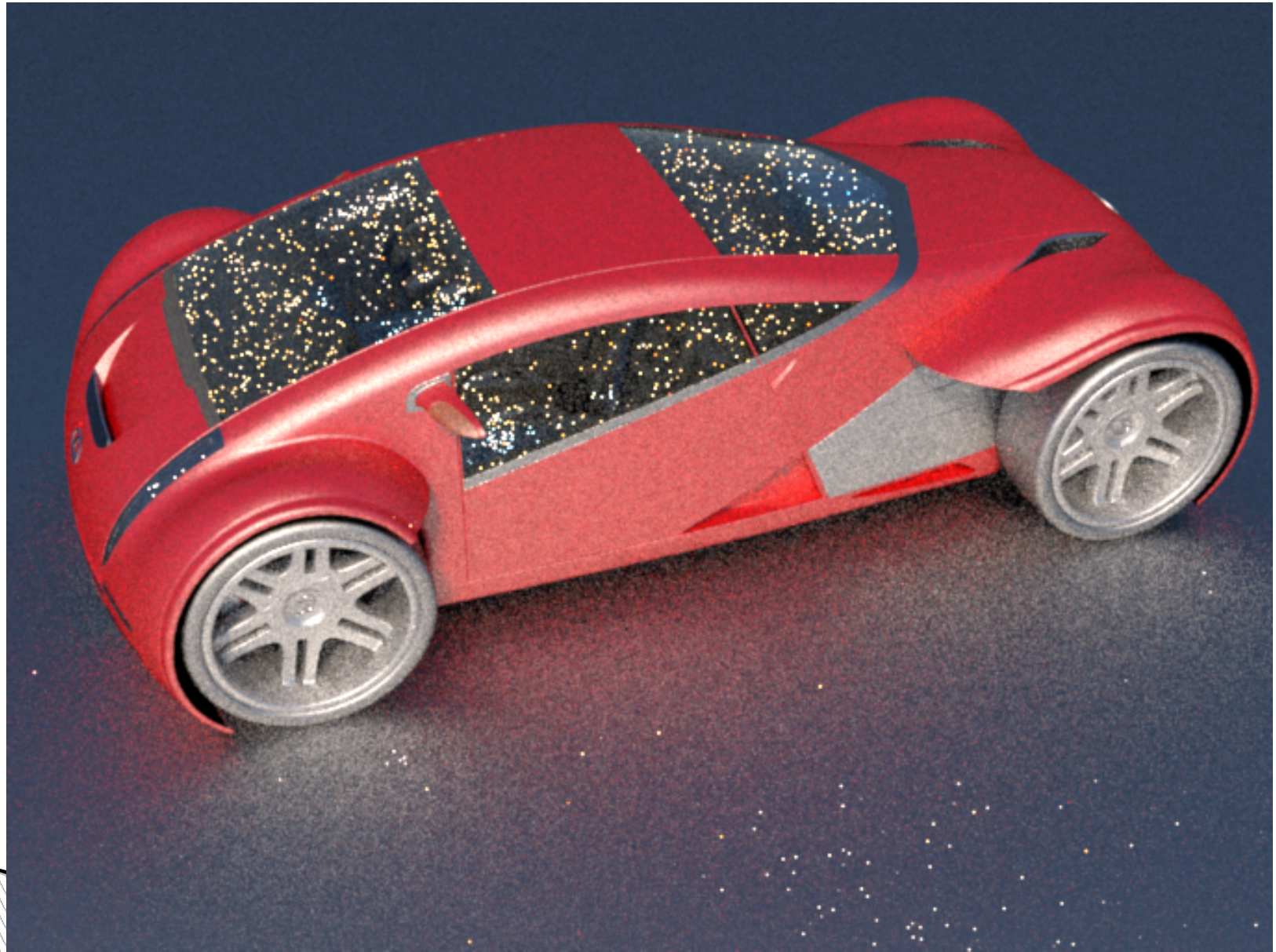
Bi-directional path-tracing

- Tracer light-paths and eye-paths
- Connect these paths together
  - Only for non-specular path vertices
- Probability? Weight?
- Express things in path-space
- Multiple Importance-sampling

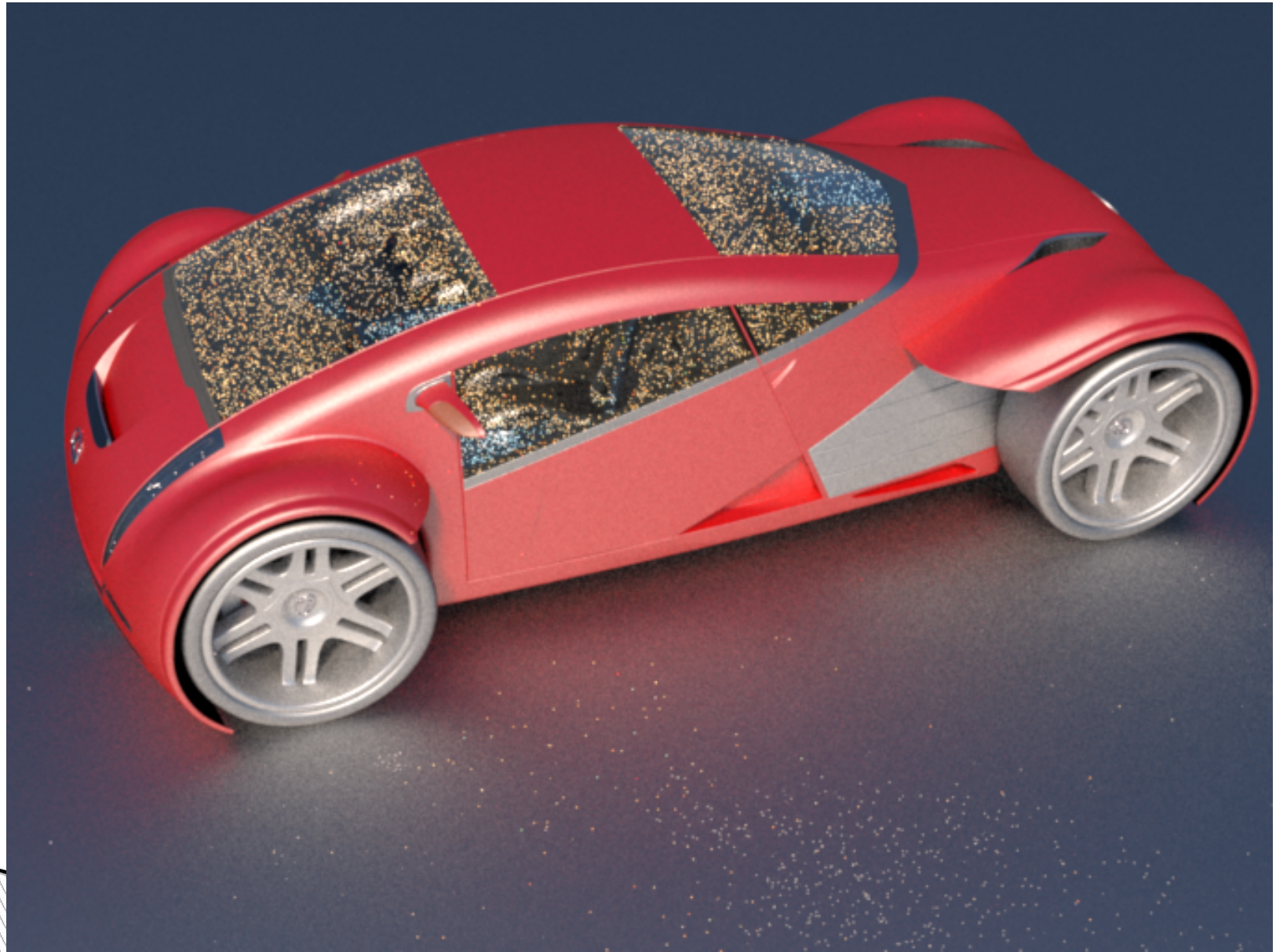
BDPT, 1 spp, 2s



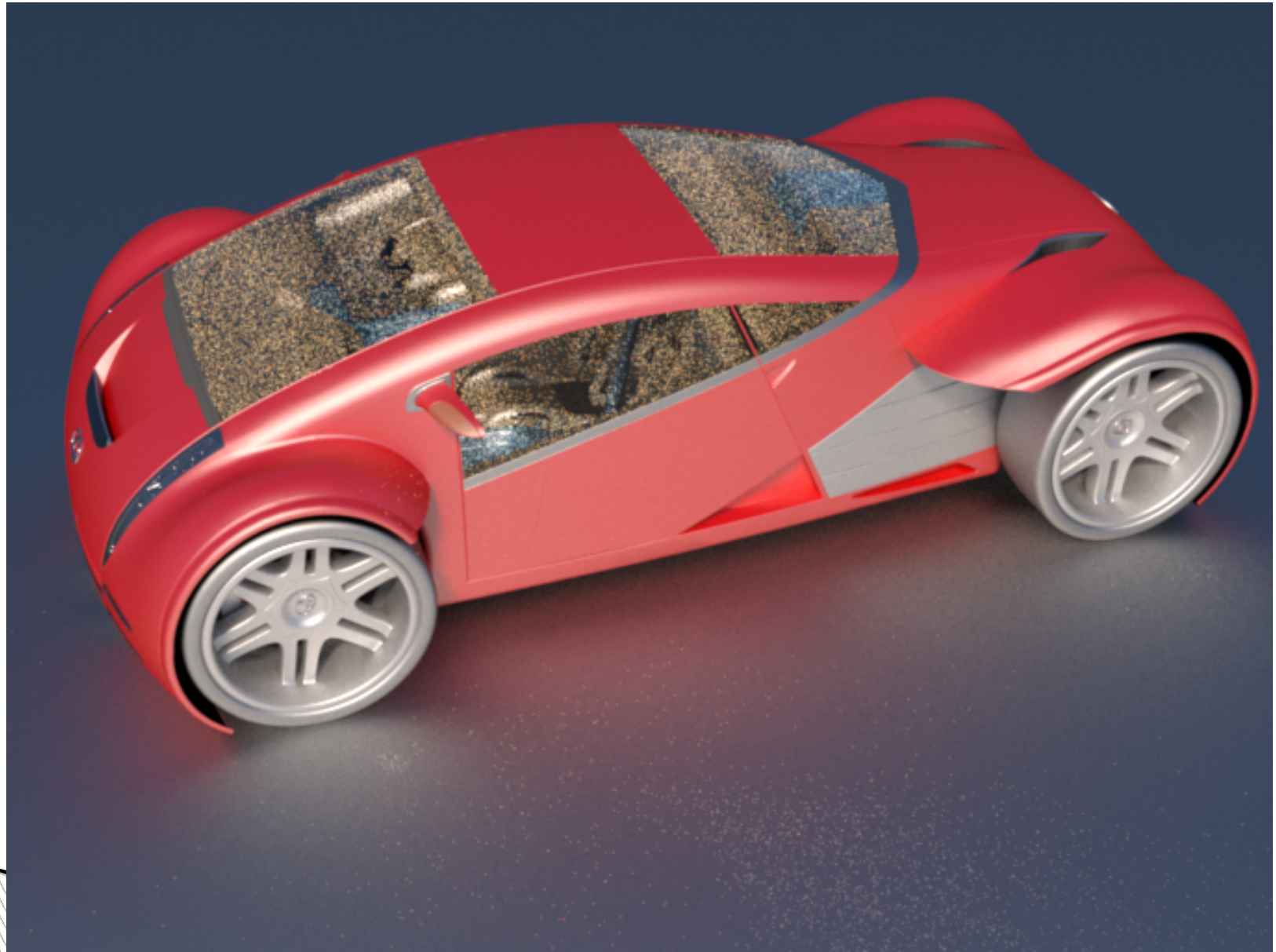
BDPT, 32 spp, 42 s



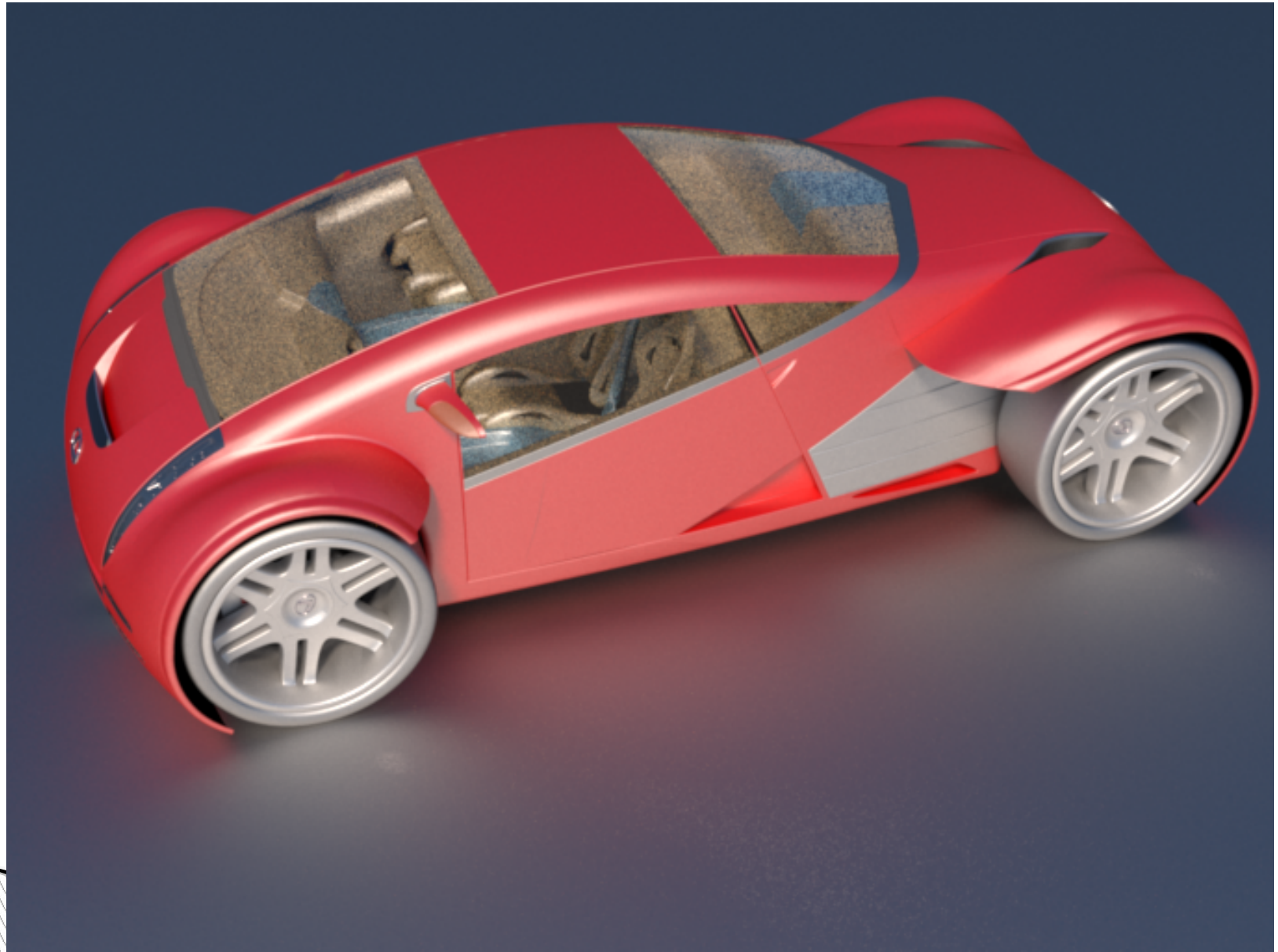
BDPT, 256 spp, 5.4 mn



BDPT, 1024 spp, 23 mn



Path-Tracing, adaptive, 53 mn

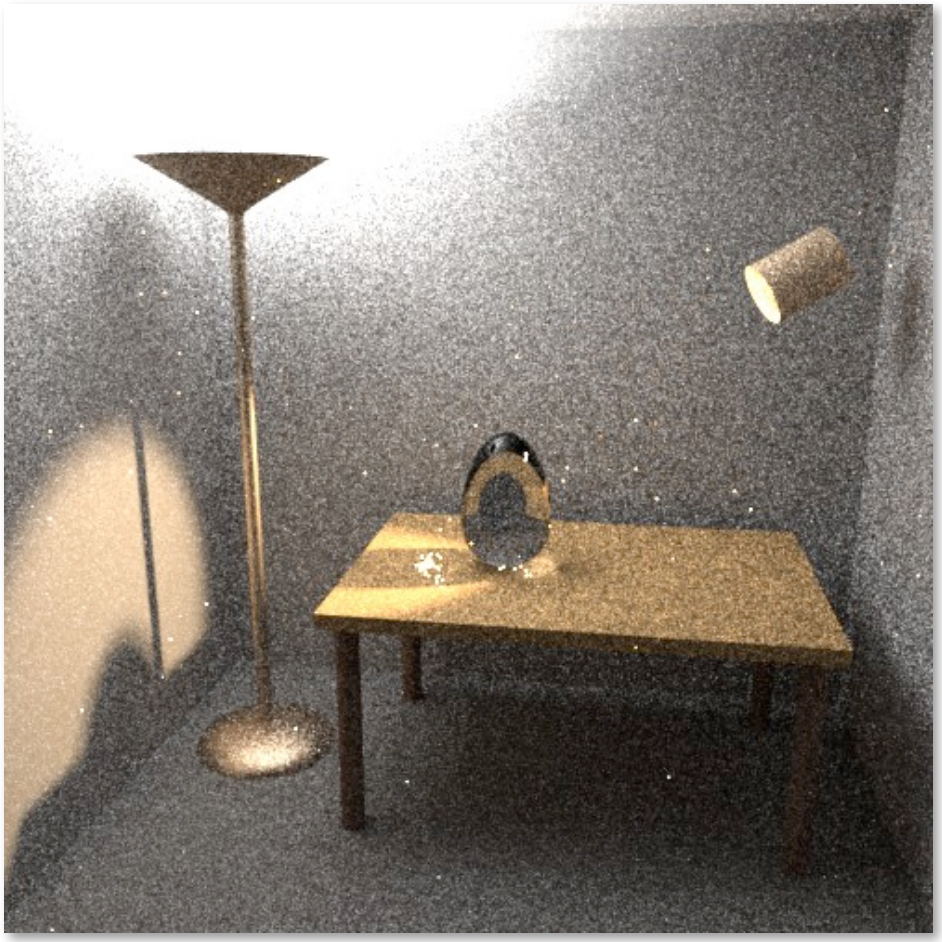


Bi-Directional Path Tracing

- Faster convergence than Path-Tracing
- Less noise

- Inside the car:
  - Light being refracted by the windows
  - Hard to find connections
  - Reverts to path-tracing
  - BDPT better than PT for small light sources

Bi-Directional Path Tracing



Path Tracing

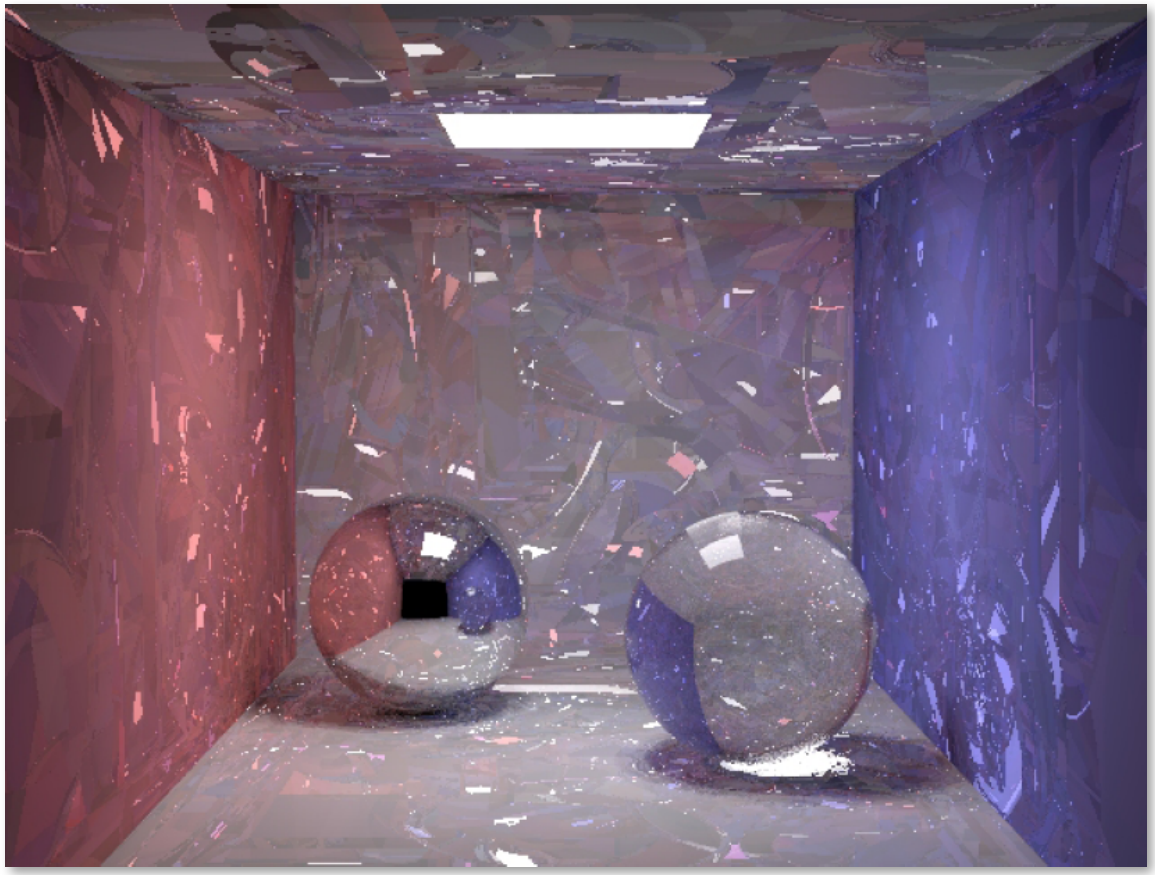
Bi-Directional Path Tracing
(Veach and Guibas 1995)

Advanced Monte-Carlo techniques

- Random samples generation
- Importance Sampling
- Irradiance Caching
- Photon Mapping
- Metropolis Light Transport

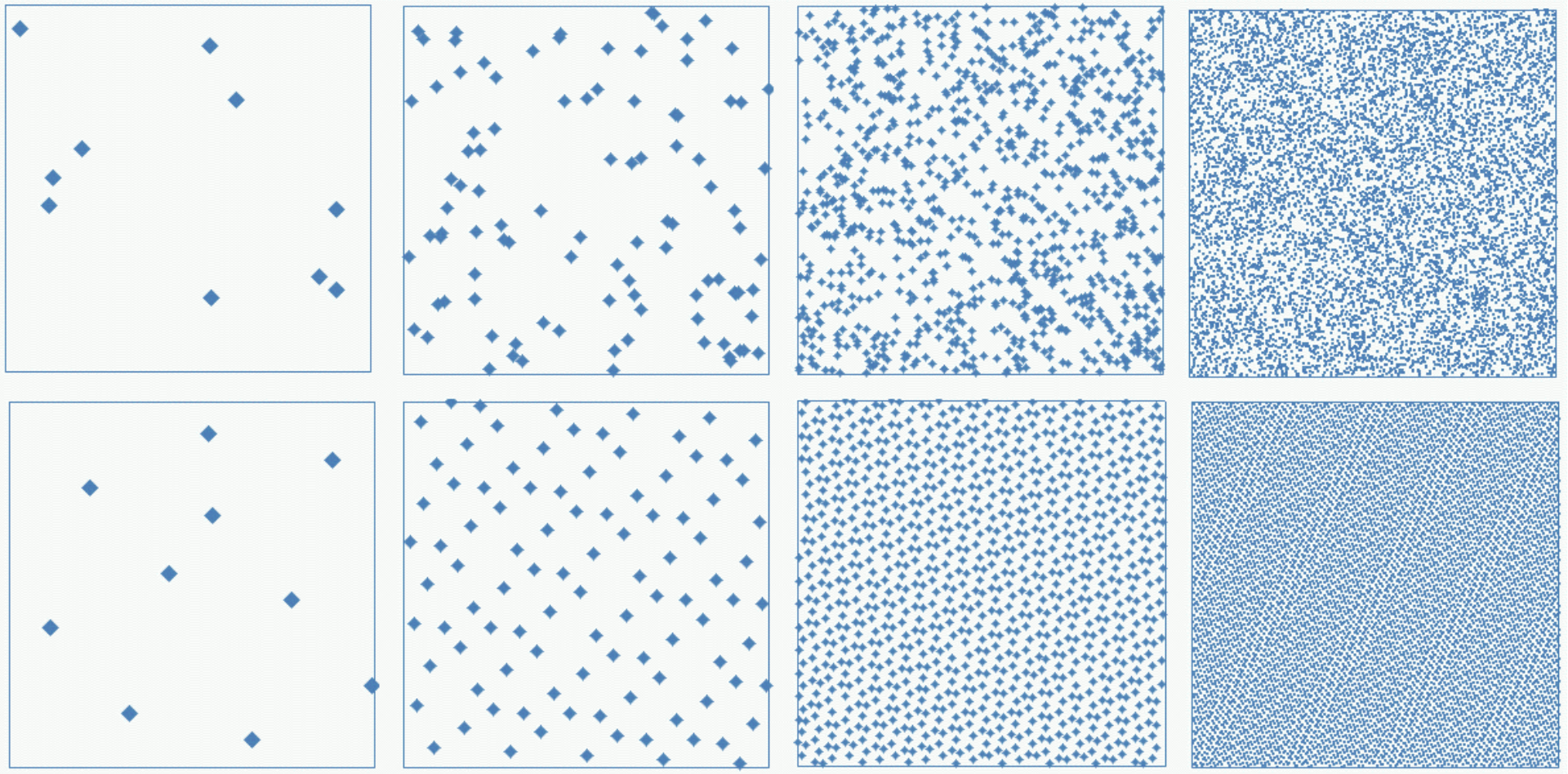
Random samples generation

- Pseudo-random sampling sequence
- The structure appears:



Quasi-random samples

Fully random sequences

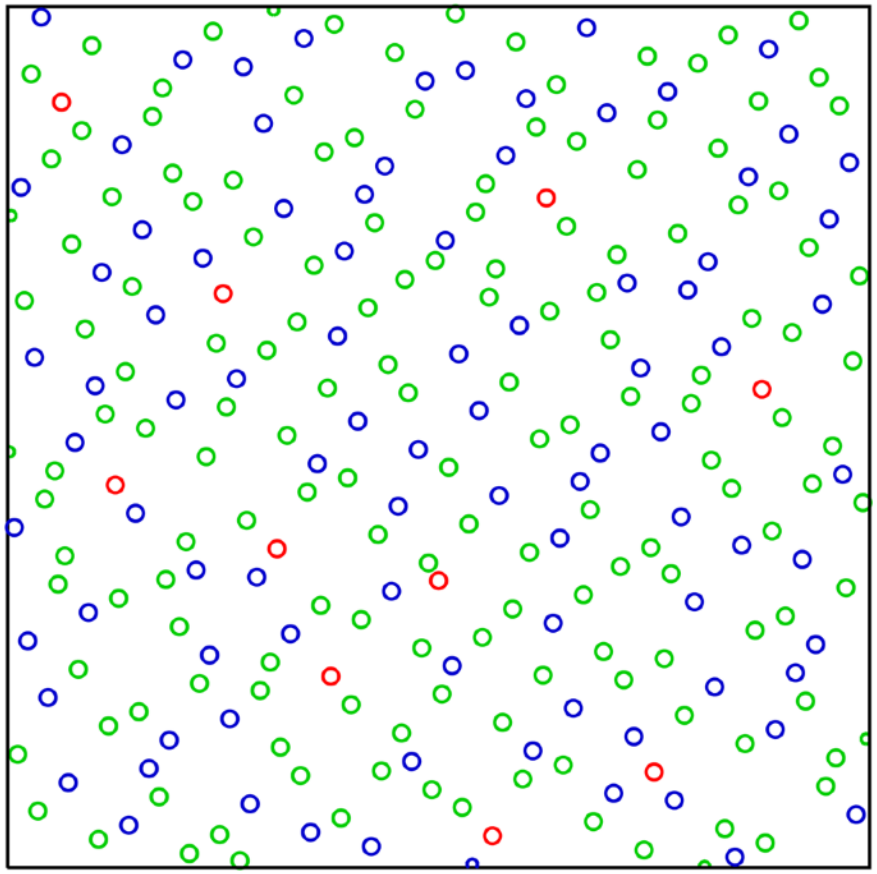


Quasi random sequences

Quasi-random sequences

- Fully random sample points:
  - Easy to generate continuously
  - No uniform coverage of sample space
  - Discrepancy: slow convergence
- Quasi-random sequences:
  - Uniform coverage
  - Low discrepancy: converges faster
  - Many versions: Halton, Sobol, Hammersley
  - Usually built so you can use a subset

Example: Halton sequence



Red= 1..10

Blue= 11..100

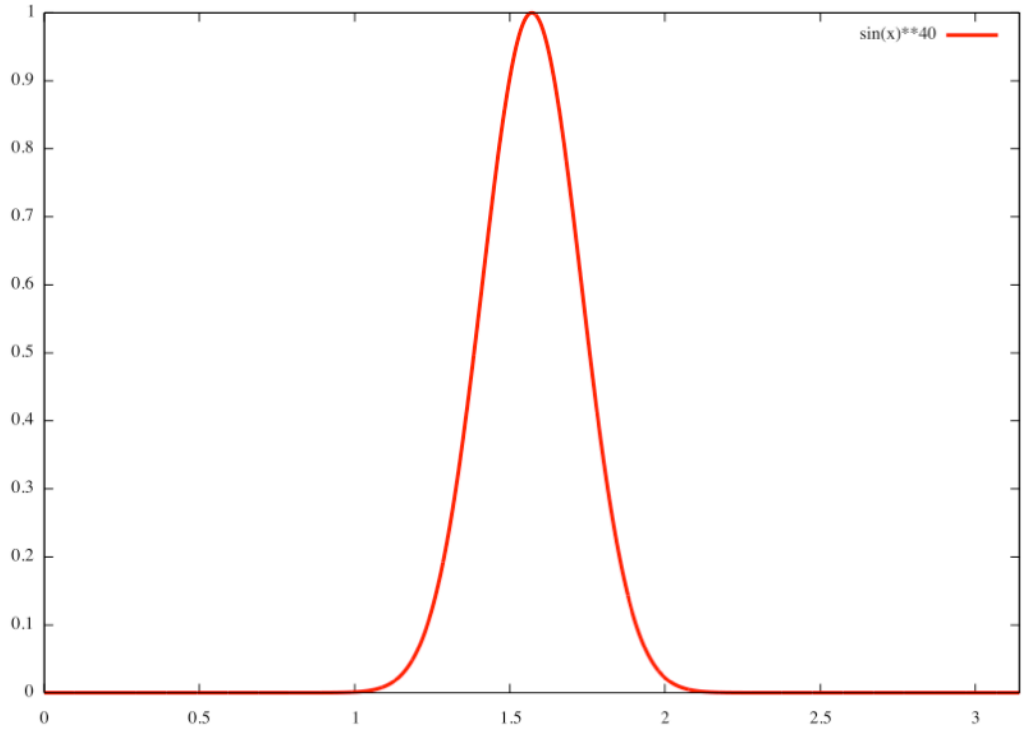
Green= 101..256

Quasi-random sequences

- There is a structure!
- It can become visible
- Don’t use always the same sequence:
  - Different starting point
  - Rotated 90°
  - Mirrored
  - …

Importance Sampling

- Functions with tight lobes/small support
- Random points outside the lobe: useless
- Can we guide randomnes?



Non-uniform distribution

- N samples with probability p(x)
- Monte Carlo estimator becomes:

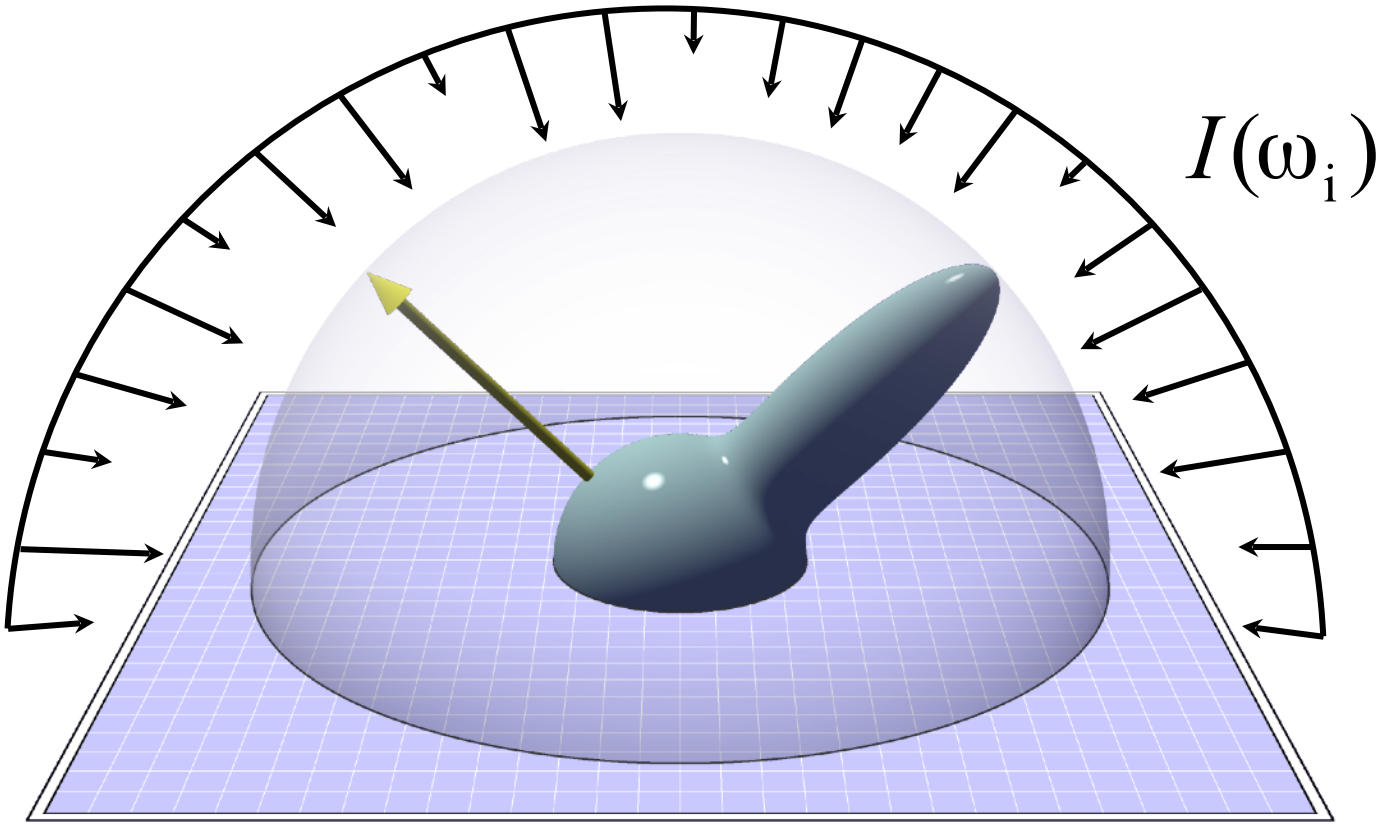
$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

- Probability p allows a better sampling of the domain

How can we choose p?

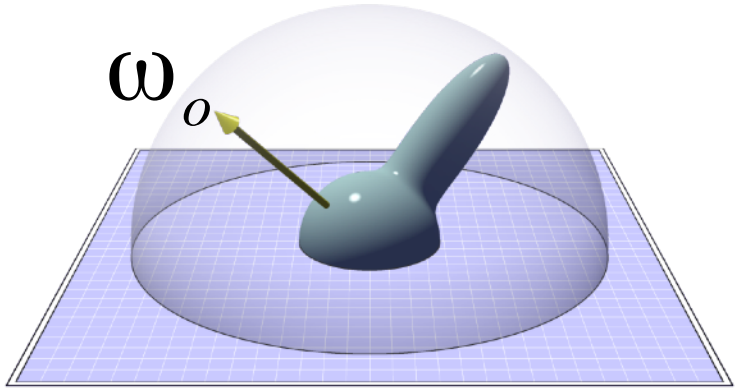
Example: glossy reflections

- Integral over hemisphere of directions
- BRDF x cosine x incoming light



Slide courtesy of Jason Lawrence

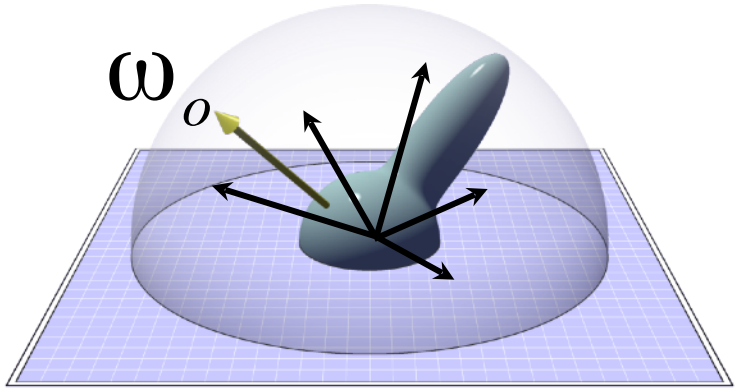
Sampling a BRDF



Slide courtesy of Jason Lawrence

Sampling a BRDF

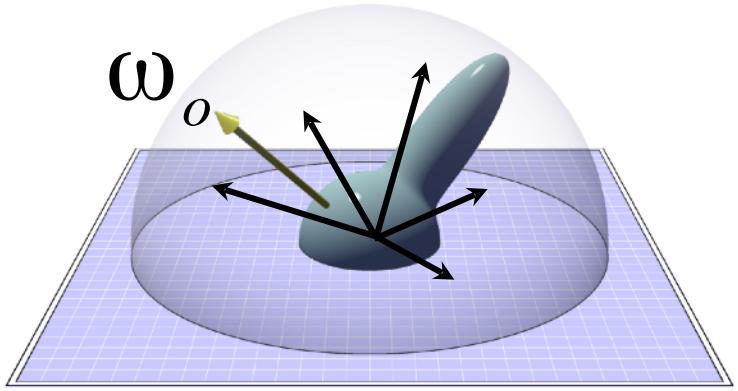
$U(\omega_i)$



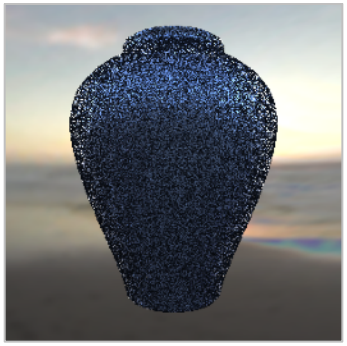
Slide courtesy of Jason Lawrence

Sampling a BRDF

$U(\omega_i)$



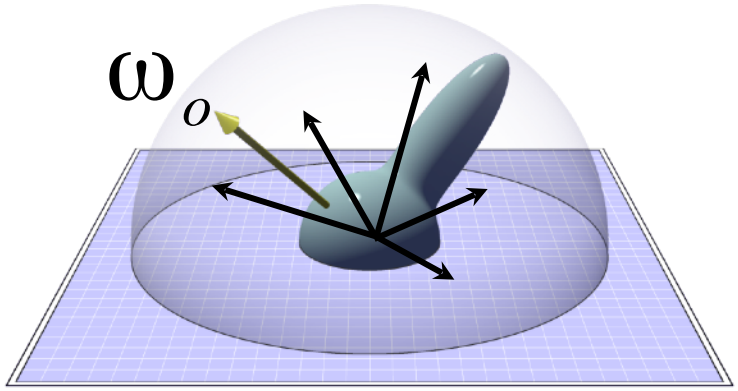
5 Samples/Pixel



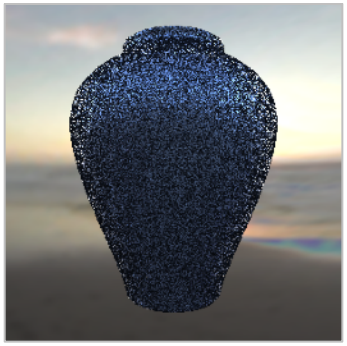
Slide courtesy of Jason Lawrence

Sampling a BRDF

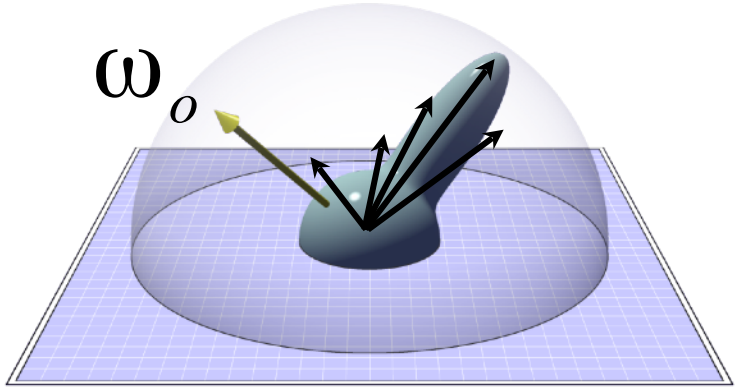
$U(\omega_i)$



5 Samples/Pixel



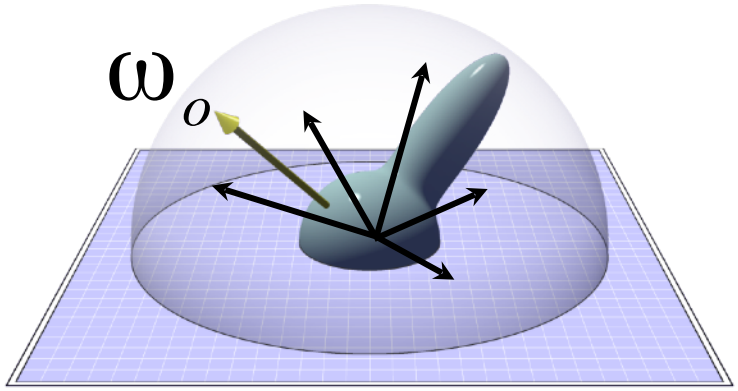
$P(\omega_i)$



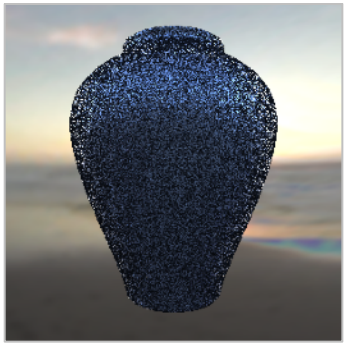
Slide courtesy of Jason Lawrence

Sampling a BRDF

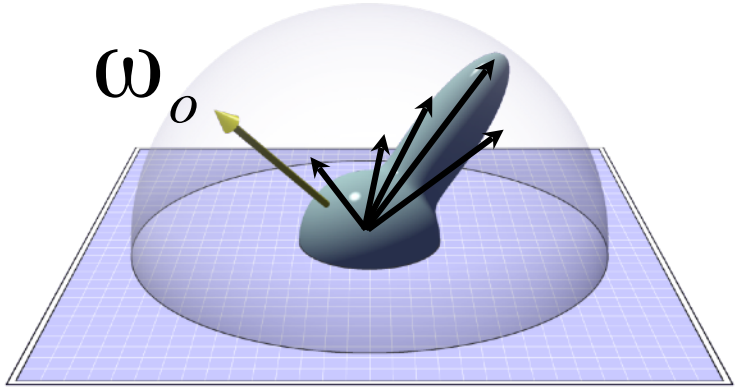
$U(\omega_i)$

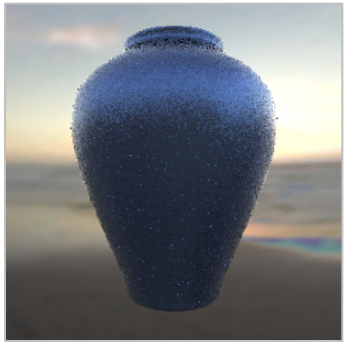


5 Samples/Pixel



$P(\omega_i)$

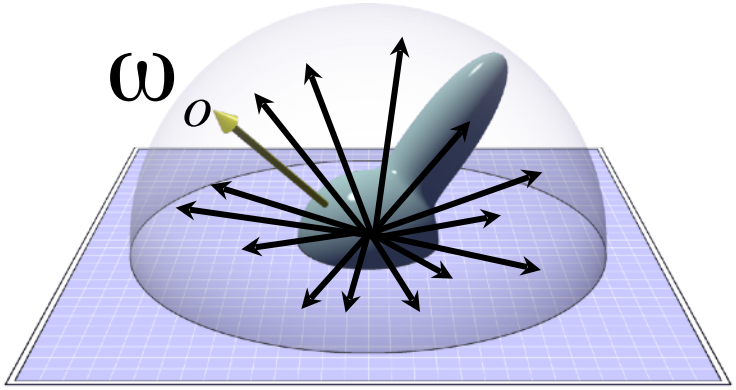




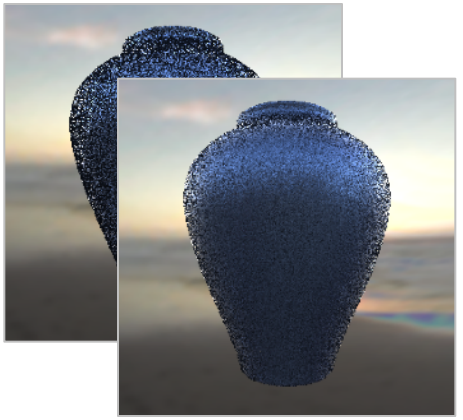
Slide courtesy of Jason Lawrence

Sampling a BRDF

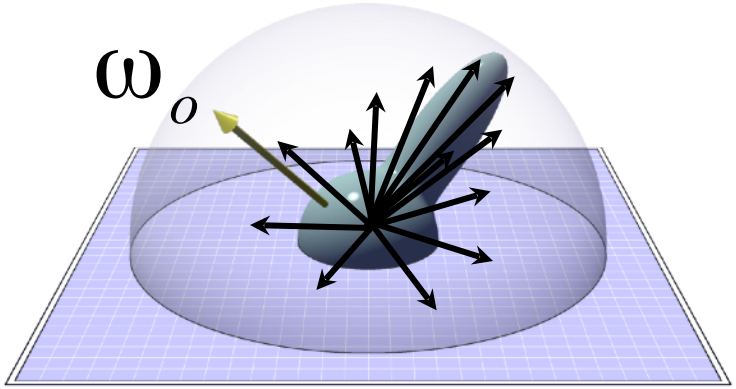
$U(\omega_i)$

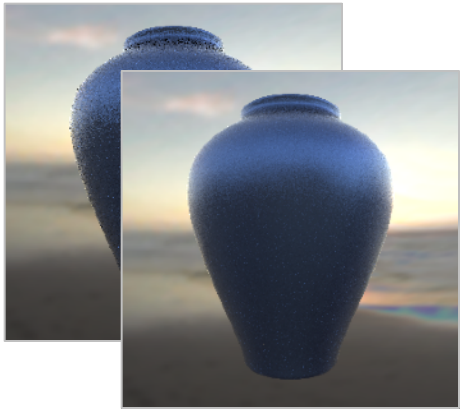


25 Samples/Pixel



$P(\omega_i)$

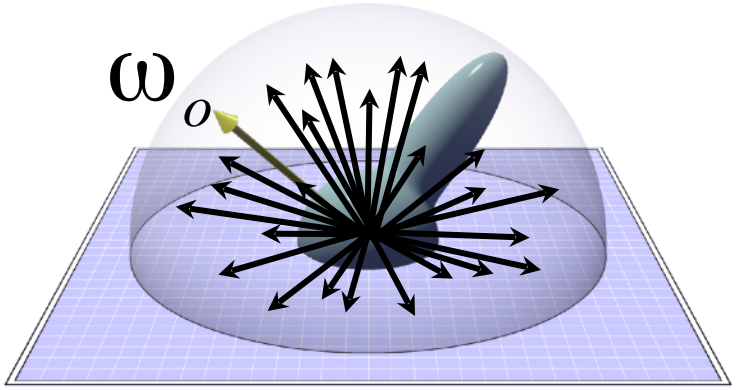




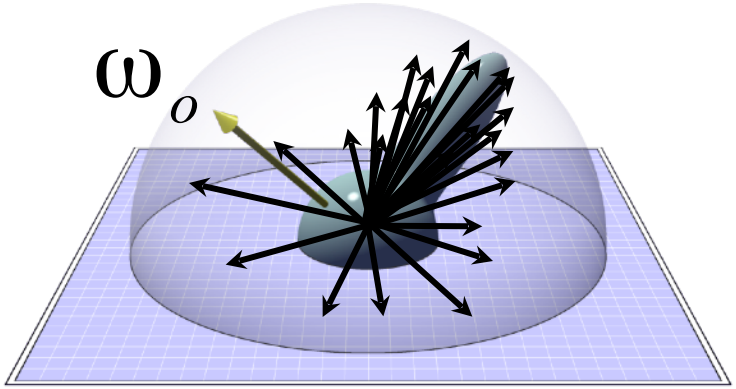
Slide courtesy of Jason Lawrence

Sampling a BRDF

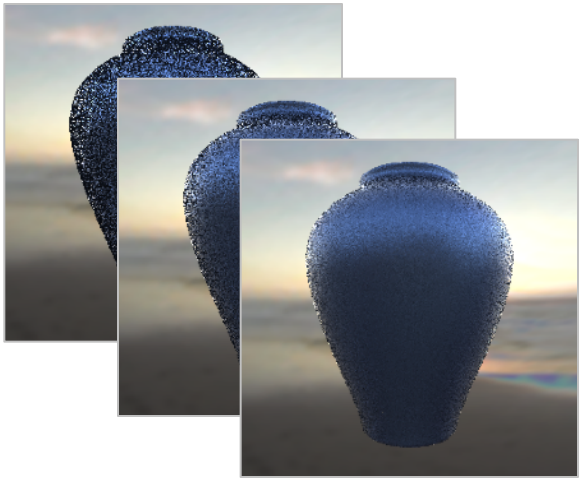
$U(\omega_i)$

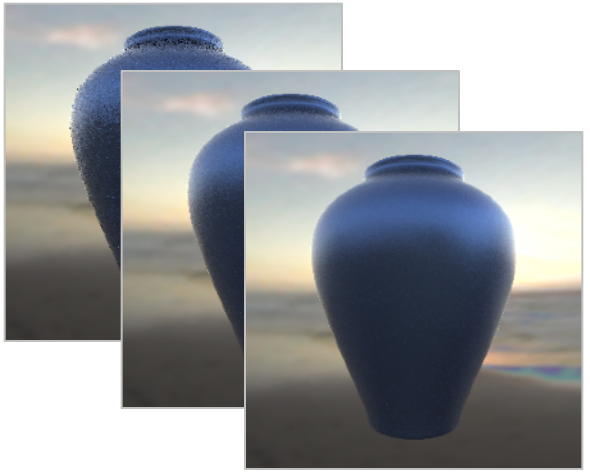


$P(\omega_i)$



75 Samples/Pixel





Slide courtesy of Jason Lawrence

Importance sampling

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

- Choose p wisely to reduce variance:
  - p must look like f
  - Doesn't change convergence with $\sqrt{N}$ (reduces the constant)
- Is present in all path-tracers

Importance sampling in practice

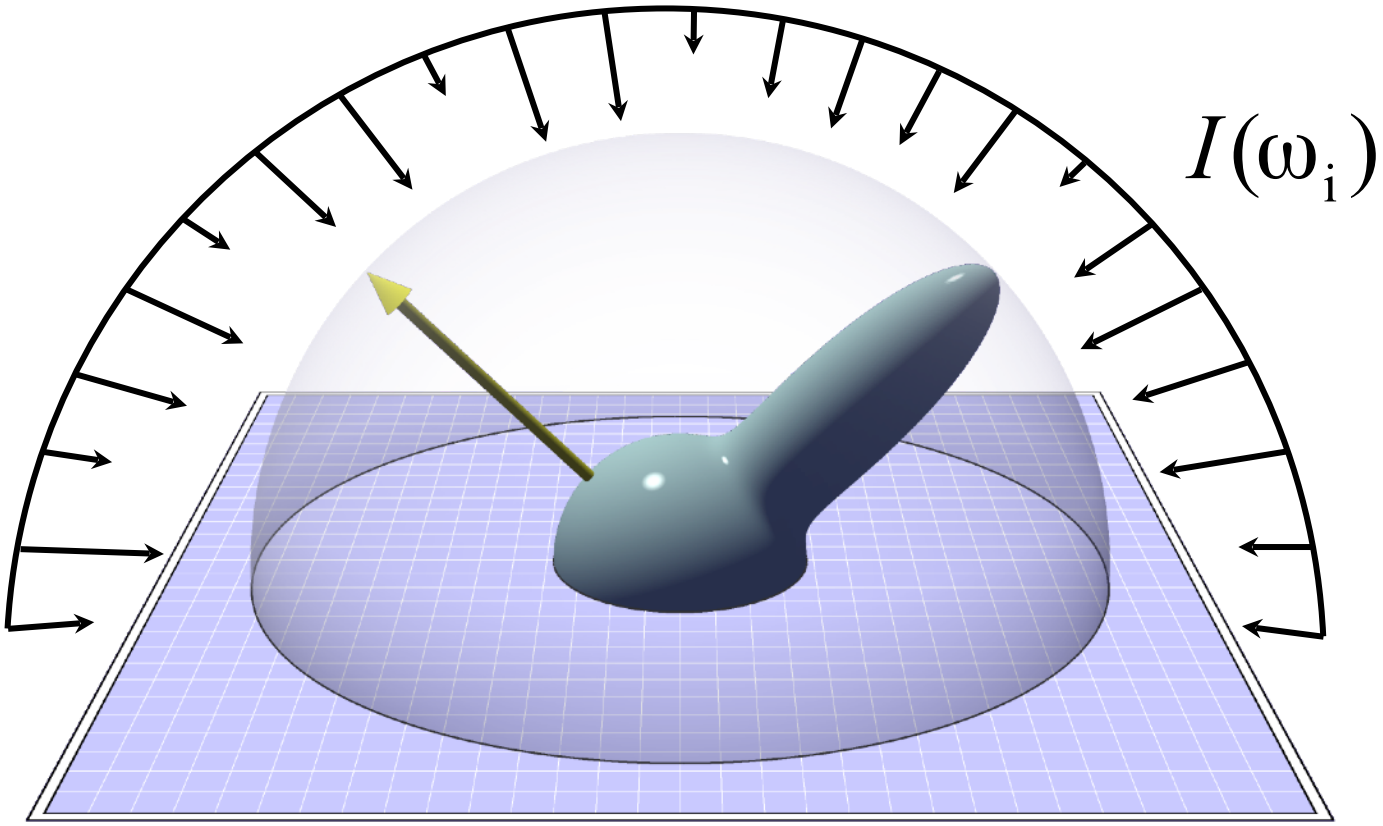
Phong + environment map, on the GPU **100 spp**

Importance sampling in practice

Phong + environment map, on the GPU **1000 spp**

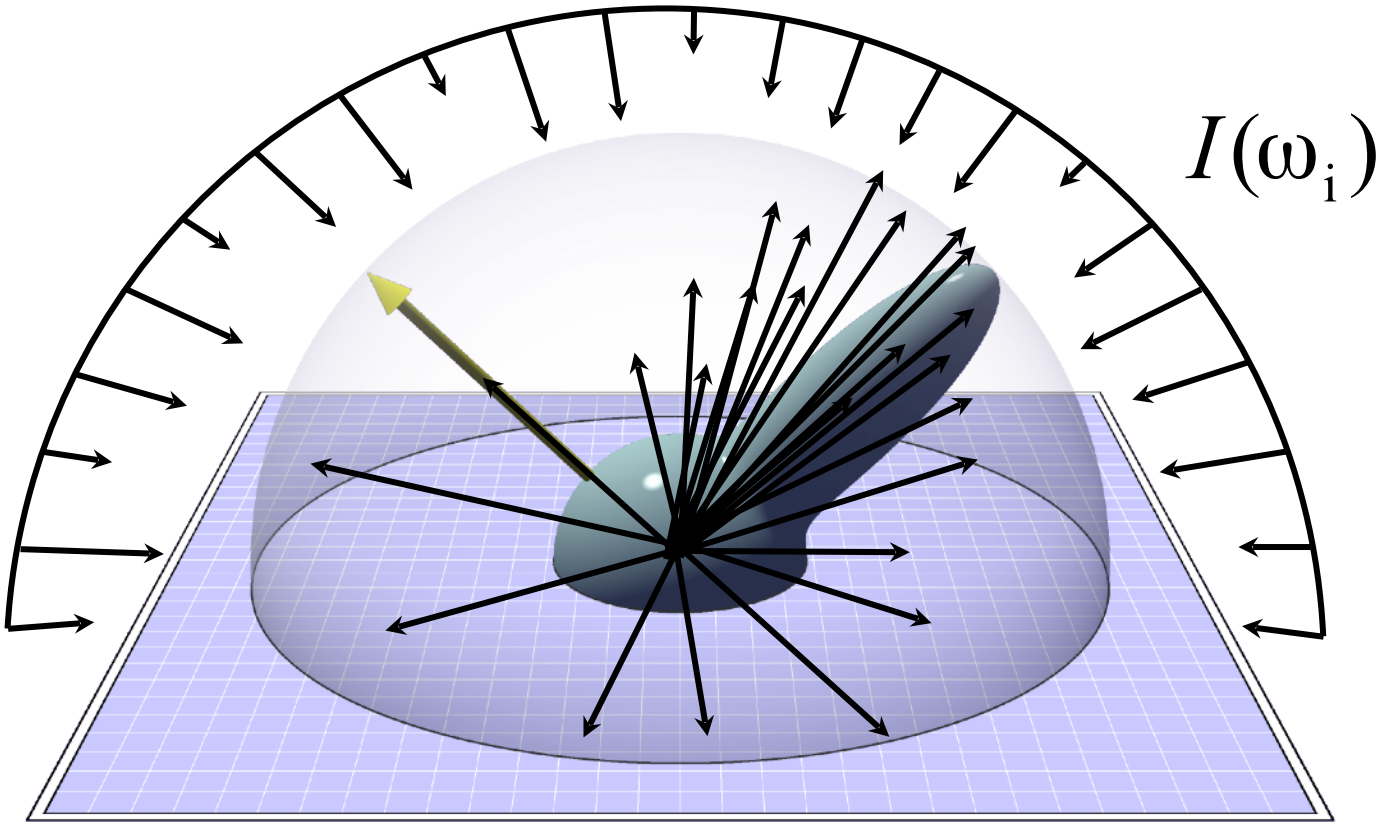
Importance sampling Phong

- Input: ray from the eye/camera
- Goal: outgoing rays, generated with IS



Importance sampling Phong

- Input: ray from the eye/camera
- Goal: outgoing rays, generated with IS



Importance sampling Phong

- Actually: generate new normal m randomly
- Incoming ray + reflection by m
  - Outgoing direction

- New normal m :
  - 2 random numbers (u_1, u_2)
  - $\phi = 2 \pi u_1$
  - $\theta = \arccos(u_2^{(s+1)/2})$
  - m = vector (θ, ϕ)
  - Probability density function for m = $\cos(\theta)^s$
  - θ = angle (n,m) = angle α ray/specular direction

Importance sampling Phong

- Contribution from one sample:

$P \mathrel{+}= F(x_i)/p(x_i)$

$P \mathrel{+}= envmap(reflected) * Phong(reflected, n) / probabilité(reflected)$

$P \mathrel{+}= envmap(reflected) * k_s (\cos \alpha)^s / (\cos \alpha)^s$

$P \mathrel{+}= envmap(reflected) * k_s$

Importance sampling Phong

- Contribution from one sample:

$P \mathrel{+}= F(x_i)/p(x_i)$

$P \mathrel{+}= \text{envmap}(\text{reflected}) * \text{Phong}(\text{reflected}, n) / \text{probabilité}(\text{reflected})$

$P \mathrel{+}= \text{envmap}(\text{reflected}) * k_s (\cos \alpha)^s / (\cos \alpha)^s$

$P \mathrel{+}= \text{envmap}(\text{reflected}) * k_s$

Wait, what?

Importance sampling Phong

- What if there is a diffuse component?
  - Or several specular components...
- Select the lobe using Monte-Carlo :
  - Energy for each lobe: $k_{d,}$ k_s,…
  - Random sample u_3 in [0,1]
  - Select sampled lobe:
    - $u_3 < k_d/(k_{s+}k_d)$: diffuse, otherwise specular
  - Find sampling direction using this lobe
  - Compute full BRDF (diffuse + specular)
  - Divide by full PDF (diffuse + spécular)
  - No simplification anymore

Importance sampling Phong

- What if there is an ambient component?
  - Doesn't depend from incoming light...
  - So no need for Monte-Carlo sampling

Monte-Carlo + Phong on the GPU

- Function random() doesn't exist in GLSL
- Pseudo-random function using sin:

```
highp float rand(vec2 co) {
    highp float a = 12.9898;
    highp float b = 78.233;
    highp float c = 43758.5453;
    highp float dt = dot(co.xy, vec2(a,b));
    highp float sn = mod(dt, 3.14);
    return fract(sin(sn) * c);
}
```

Source: byteblacksmith.com

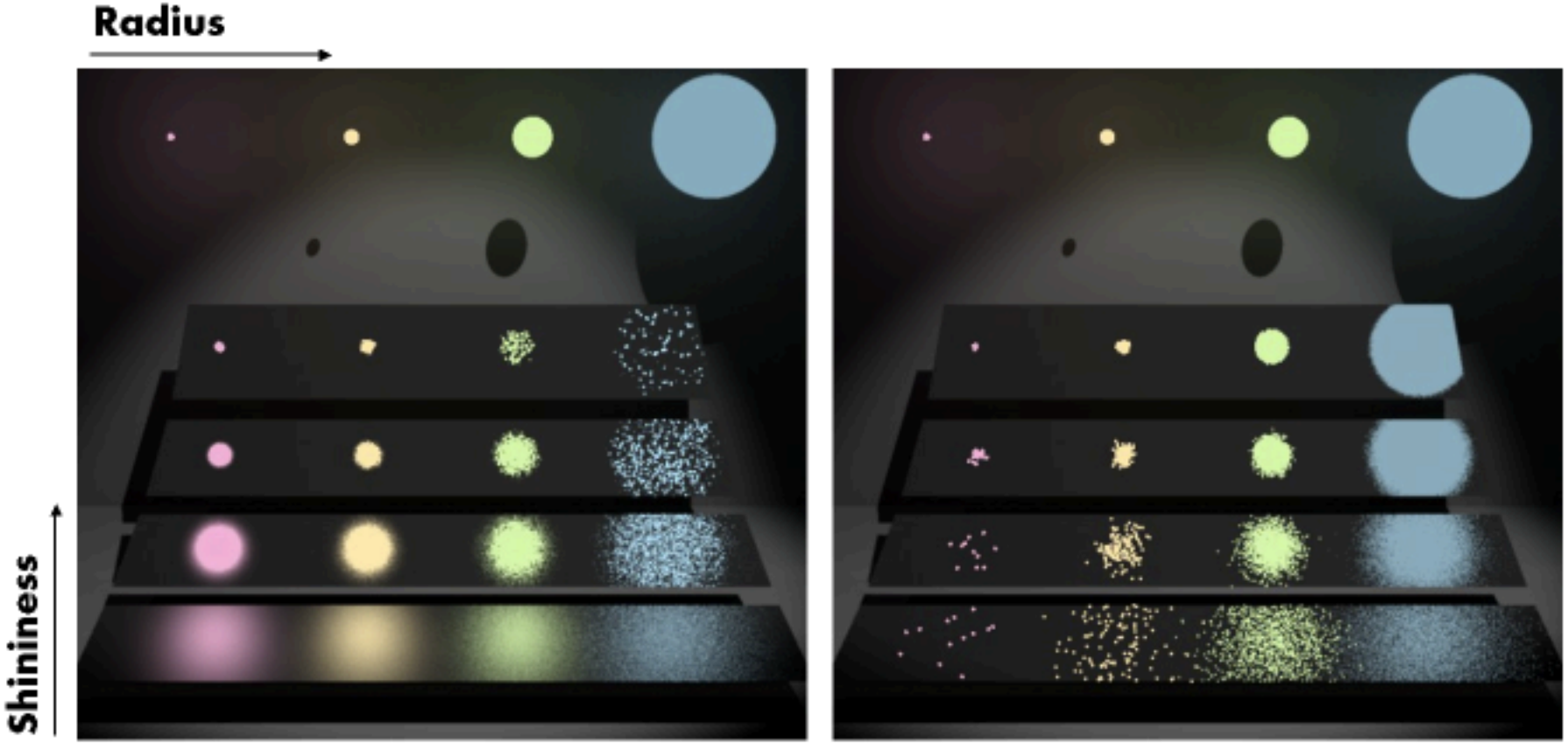
- Seed (co) :
  - Essential
  - Must be different for u_1 and u_2
  - Use your imagination

Monte-Carlo + Phong on the GPU

- New normal m : coordinates relative to n
  - Local coordinate system
  - Must move to global coordinates
  - Need a coordinate transform matrix
  - Vertical axis (z) = n (point normal)
  - Other axes (x,y) = anything
  - M = (t,b,n)

Multiple Importance Sampling

Reflection of a circular light source by a rough surface



Sampling the light source

Sampling the BRDF

$$\int f(x)g(x)dx$$

Multiple Importance Sampling

Two sampling techniques

$$X_{1,i} \sim p_1(x) \qquad X_{2,i} \sim p_2(x)$$

$$Y_{1,i} = \frac{f(X_{1,i})}{p_1(X_{1,i})} \qquad Y_{2,i} = \frac{f(X_{2,i})}{p_2(X_{2,i})}$$

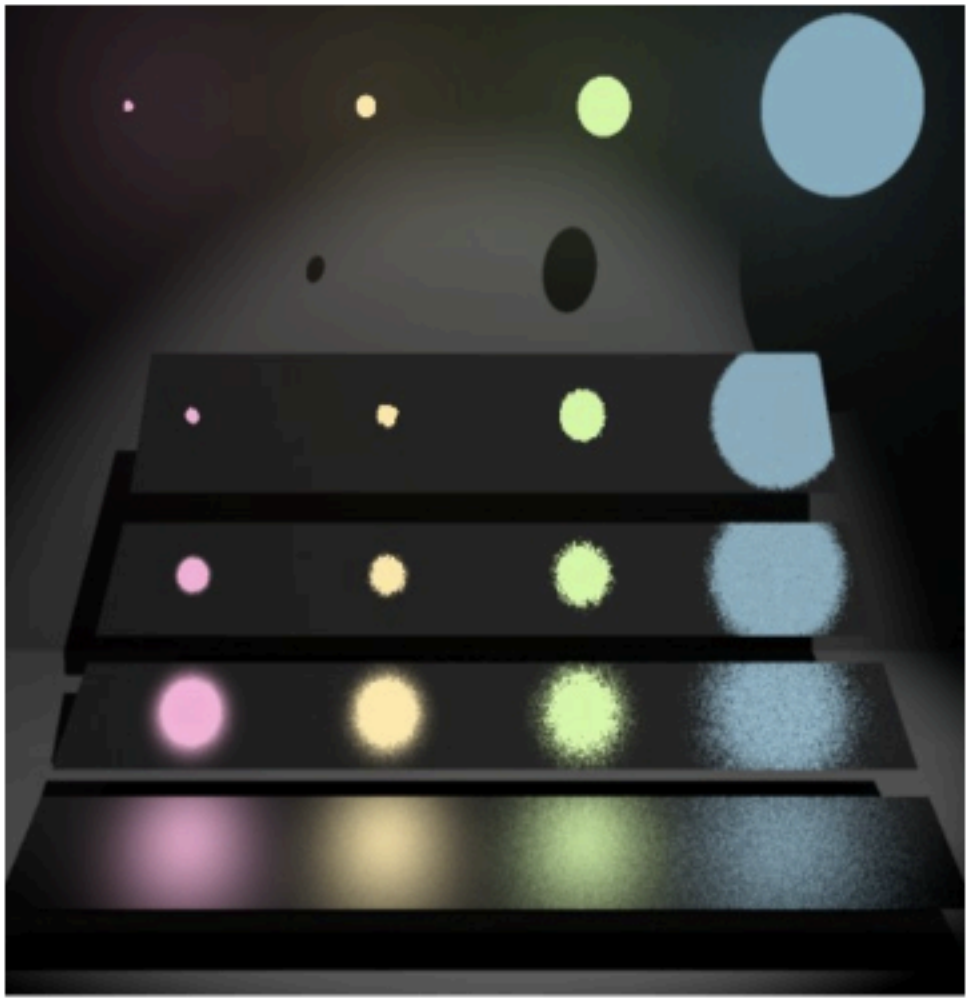
Form weighted combination of samples

$$Y_i = w_1 Y_{1,i} + w_2 Y_{2,i}$$

The balance heuristic

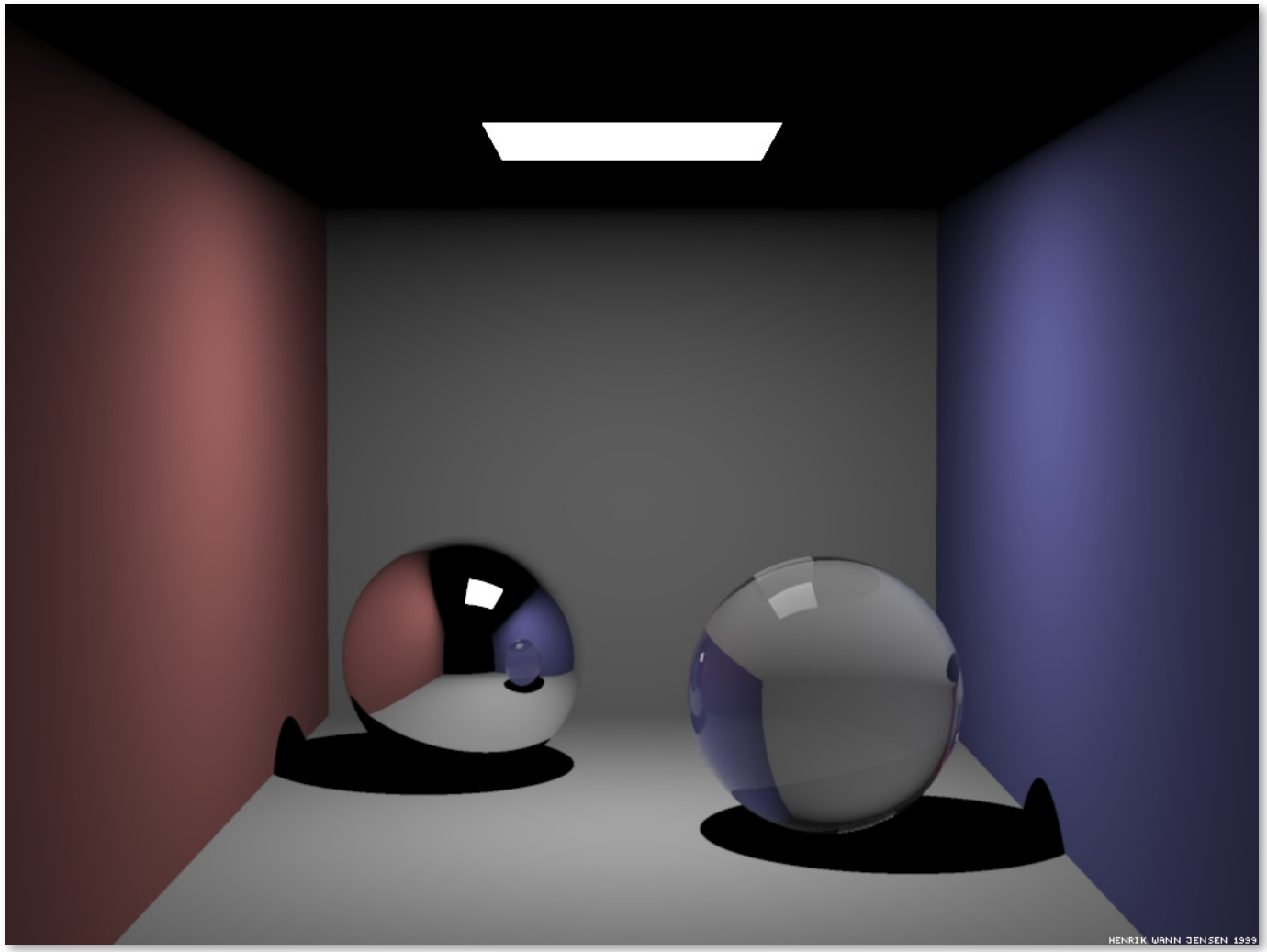
$$w_i(x) = \frac{p_i(x)}{p_1(x) + p_2(x)} \Rightarrow p(x) = w_1(x)p_1(x) + w_2(x)p_2(x)$$

Multiple Importance Sampling

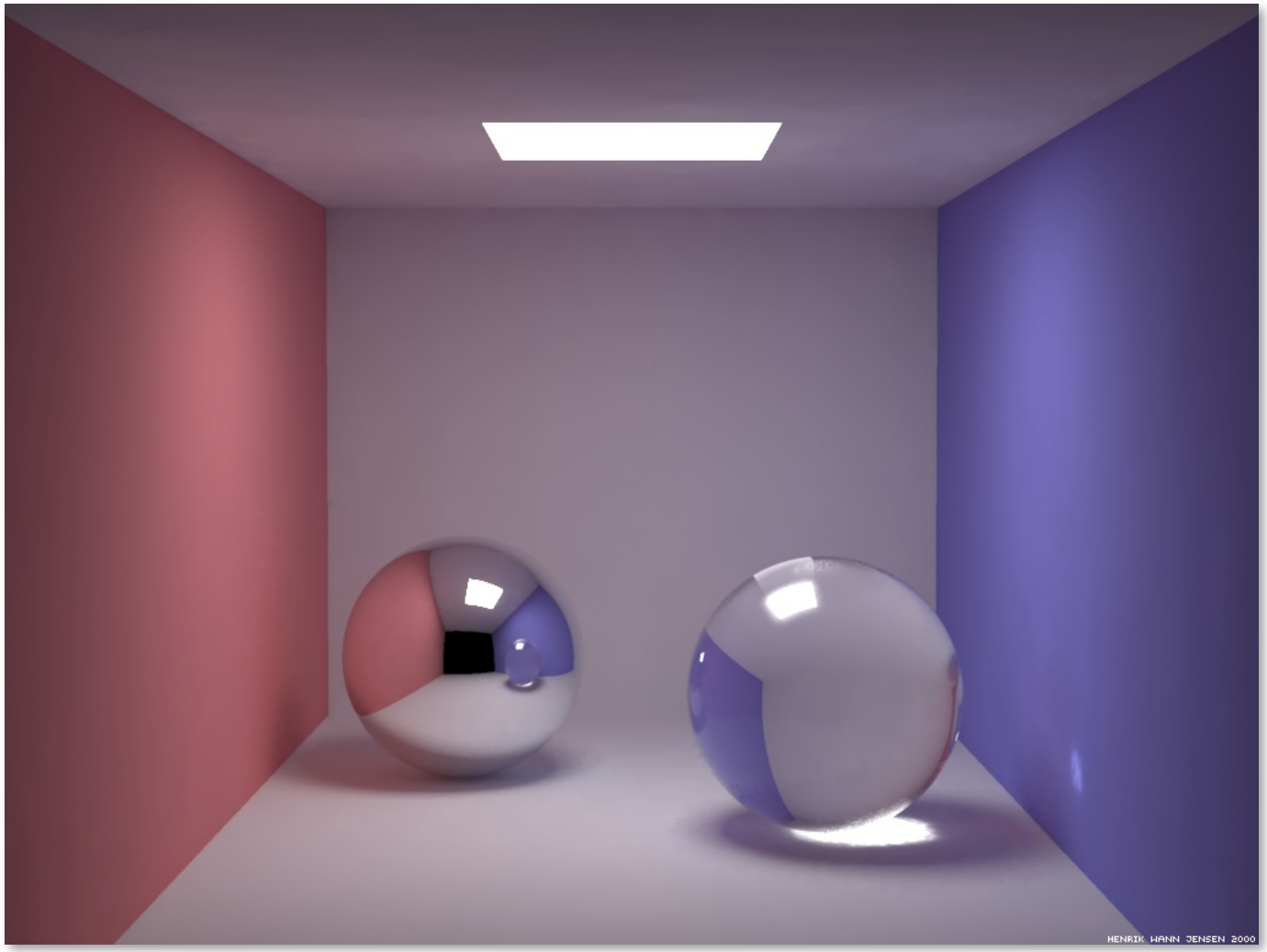


Source: Veach and Guibas

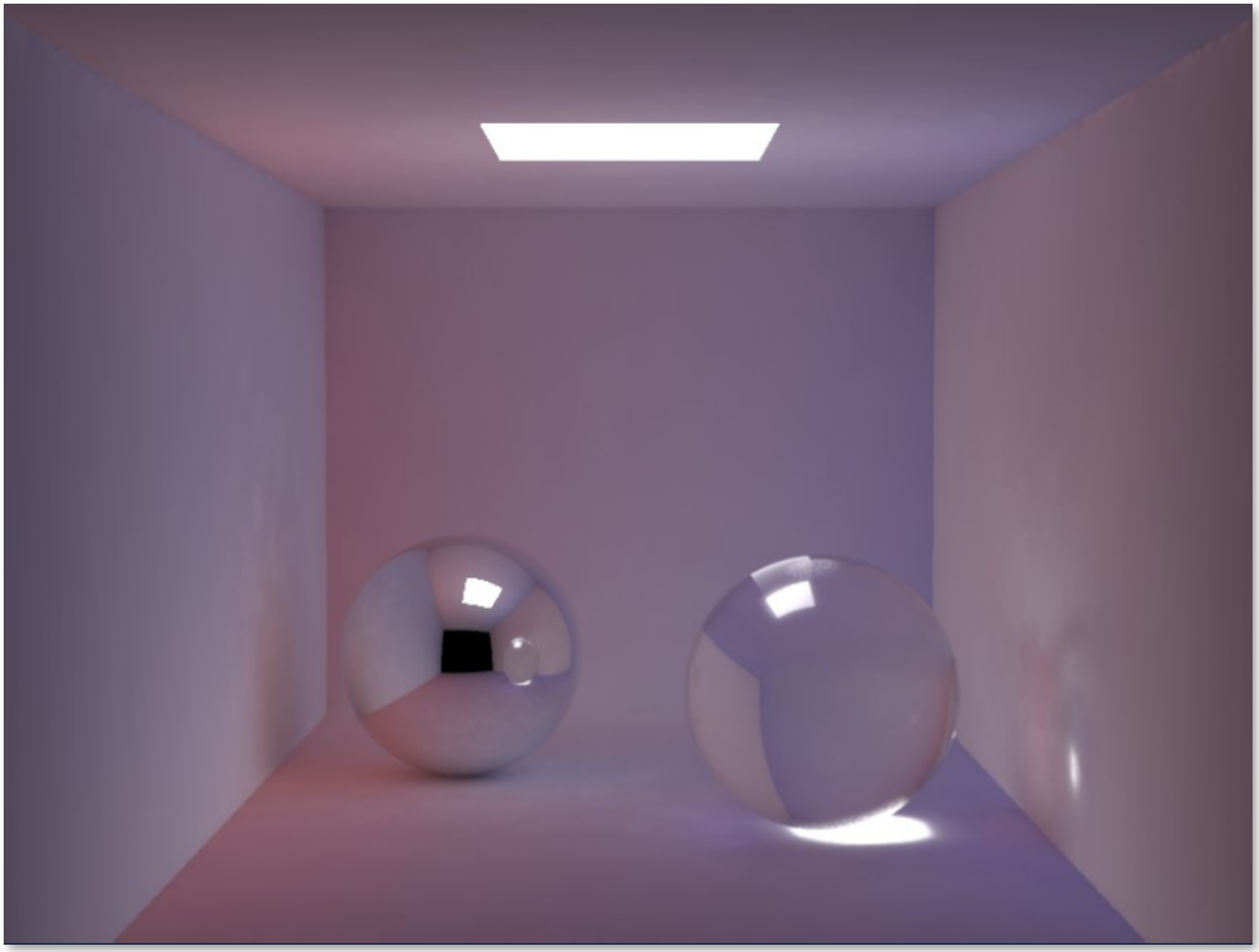
Direct lighting



Global illumination

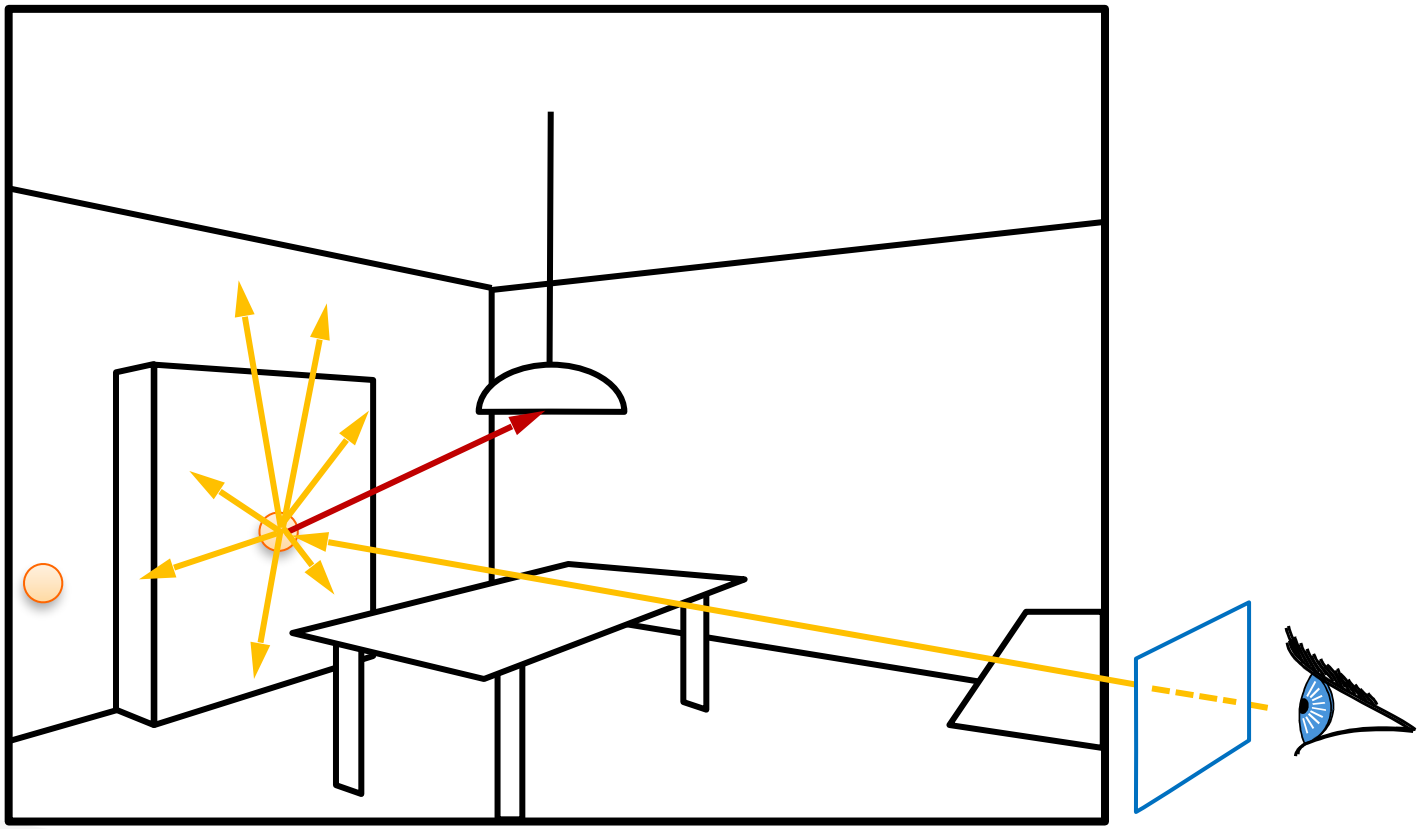


Indirect lighting only



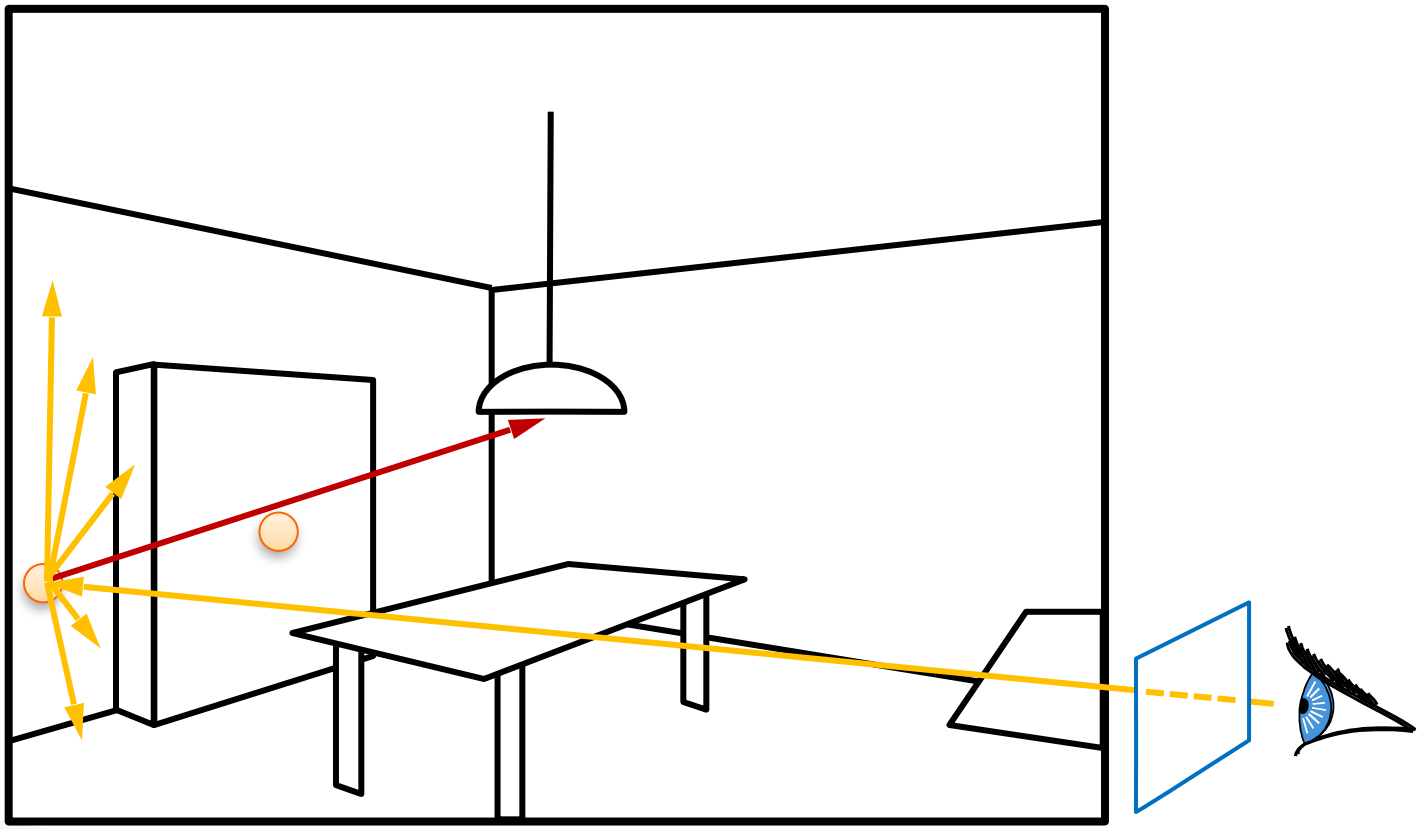
Irradiance cache

- Indirect lighting changes slowly in space



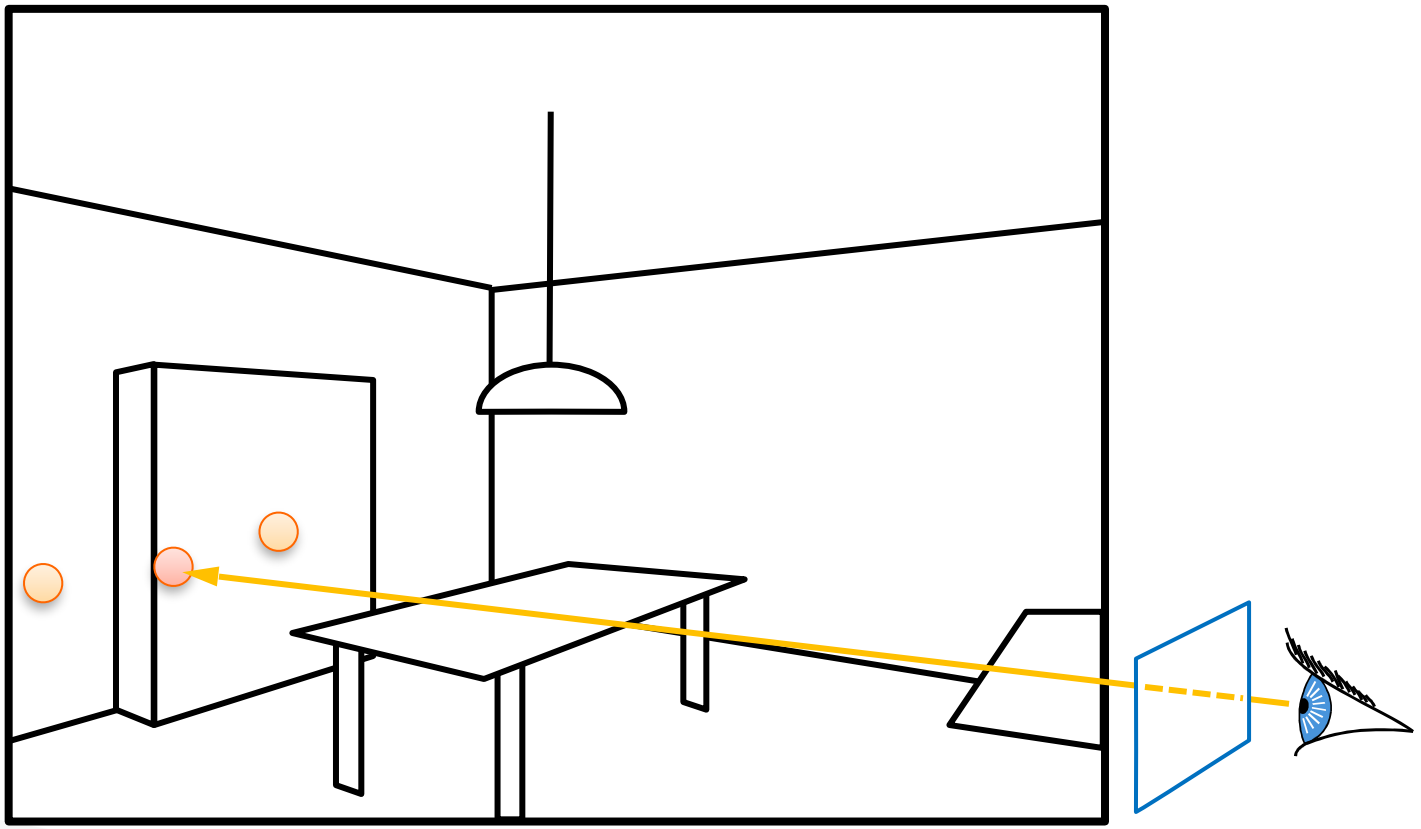
Irradiance cache

- Indirect lighting changes slowly in space
- Particularly with diffuse surfaces



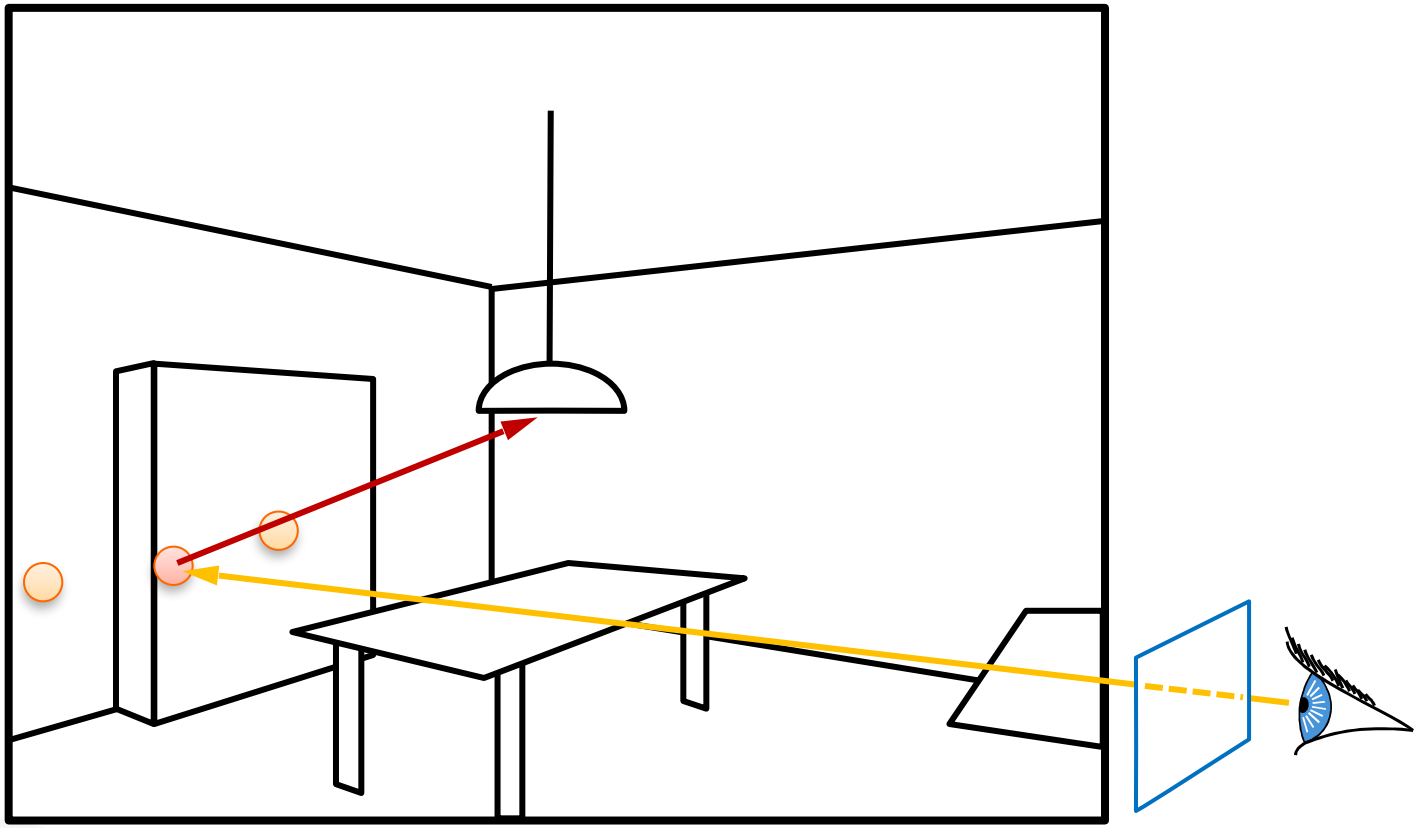
Irradiance cache

- Indirect lighting changes slowly in space
- Interpolate between neighboring values



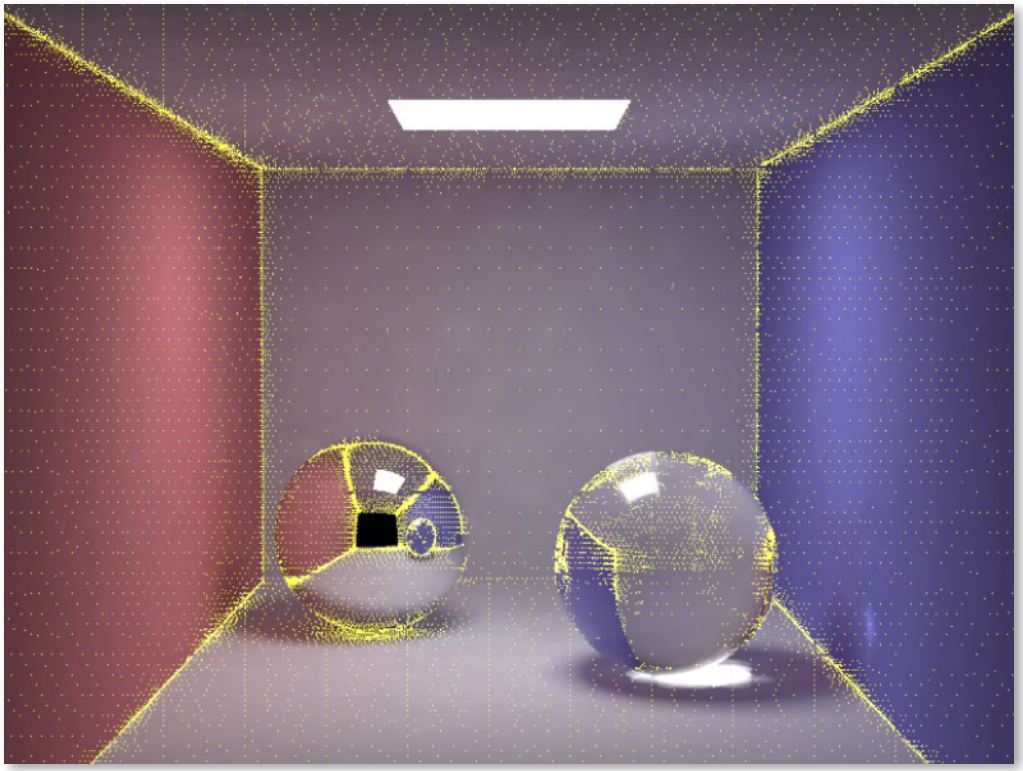
Irradiance cache

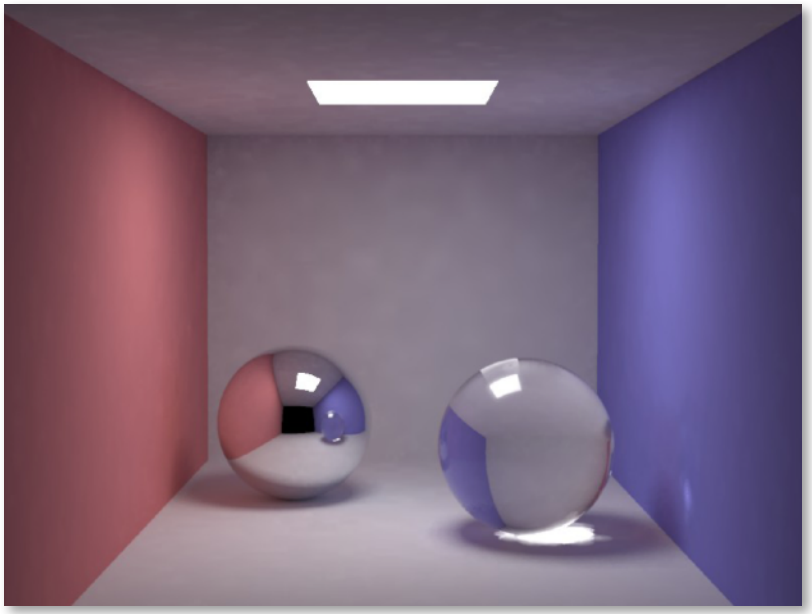
- Indirect lighting changes slowly in space
- Interpolate between cached values
- But full computation for direct lighting



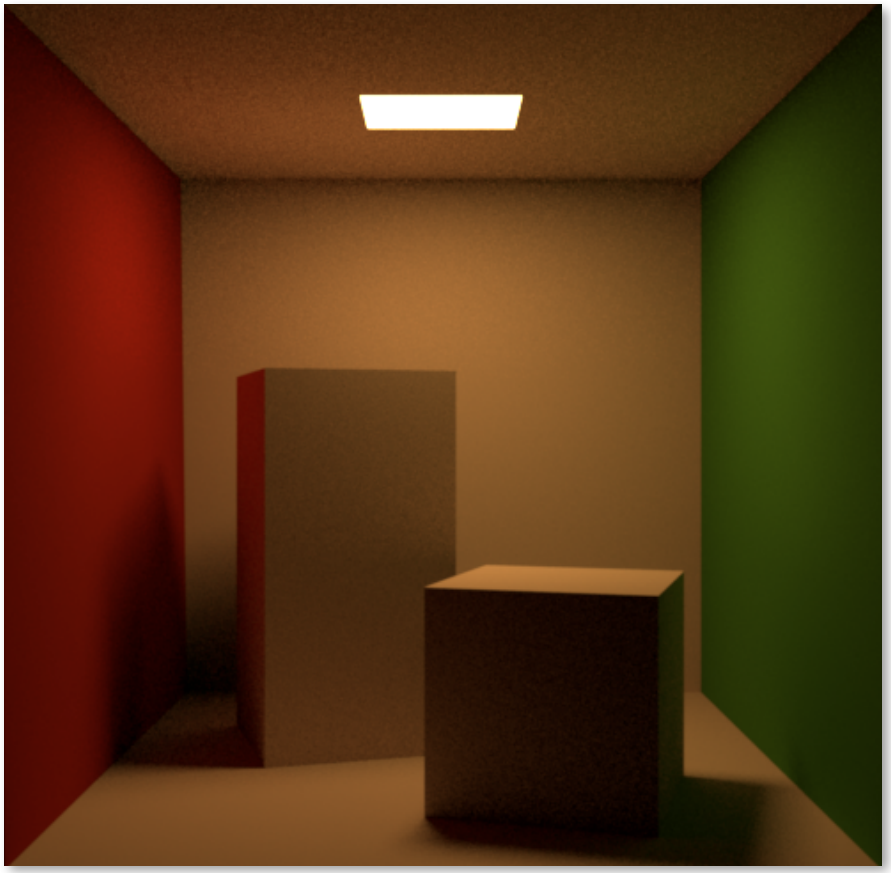
Irradiance cache

- Yellow dots: computation of indirect lighting

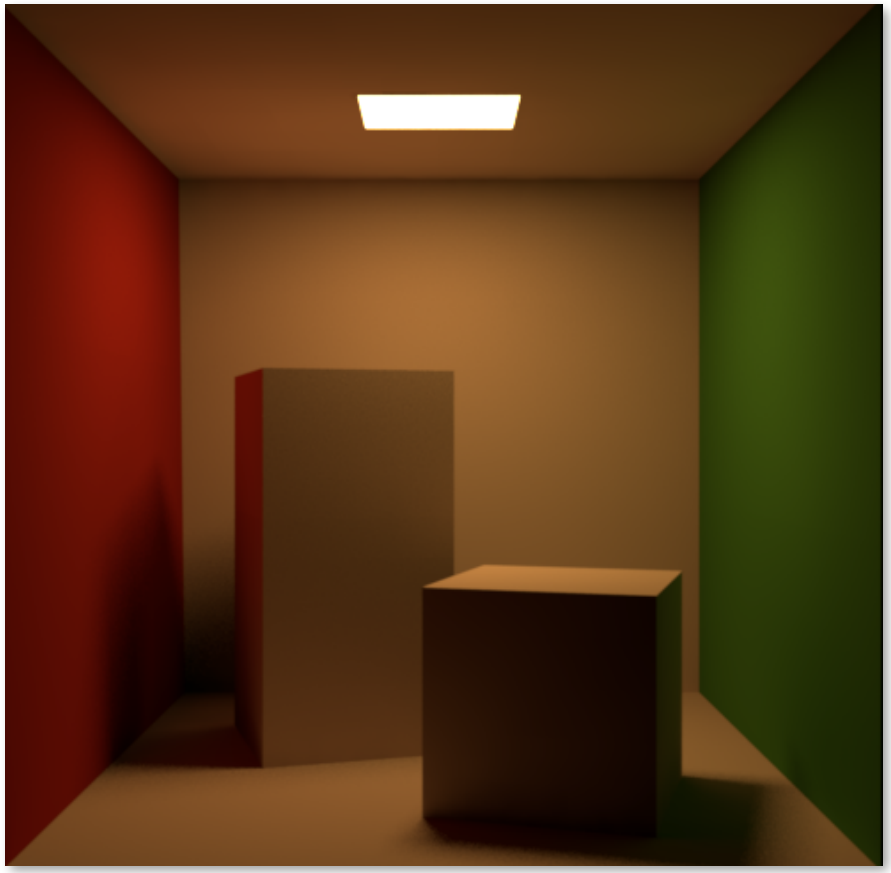




Irradiance cache



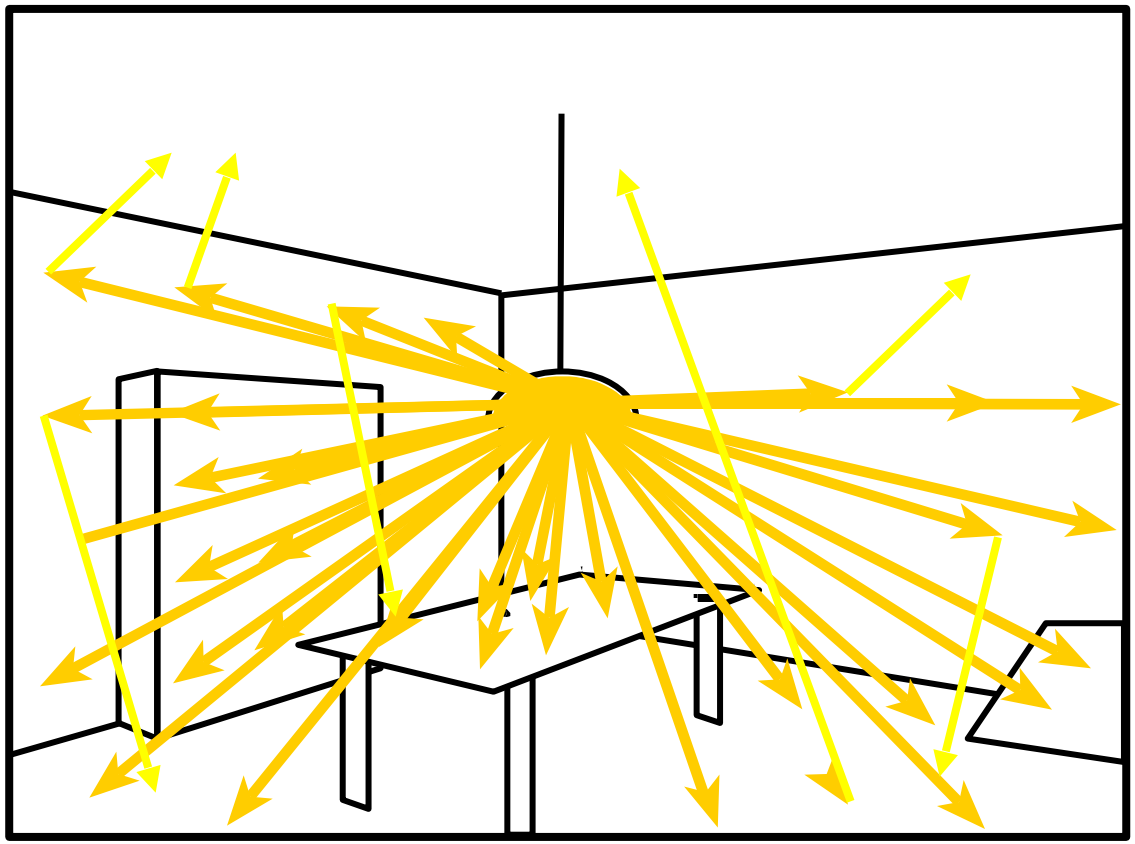
Path Tracing



Path tracing + Irradiance cache

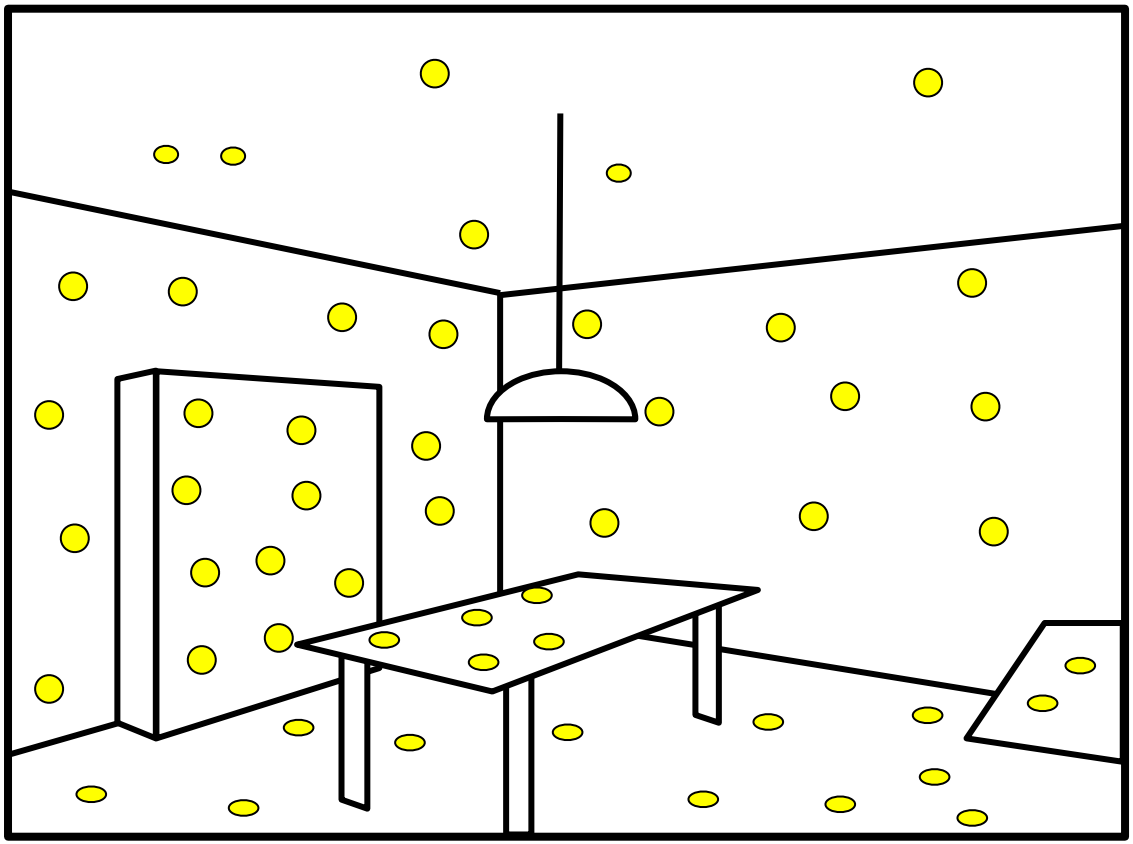
Photon mapping

- Pre-computation: throw rays from the light sources



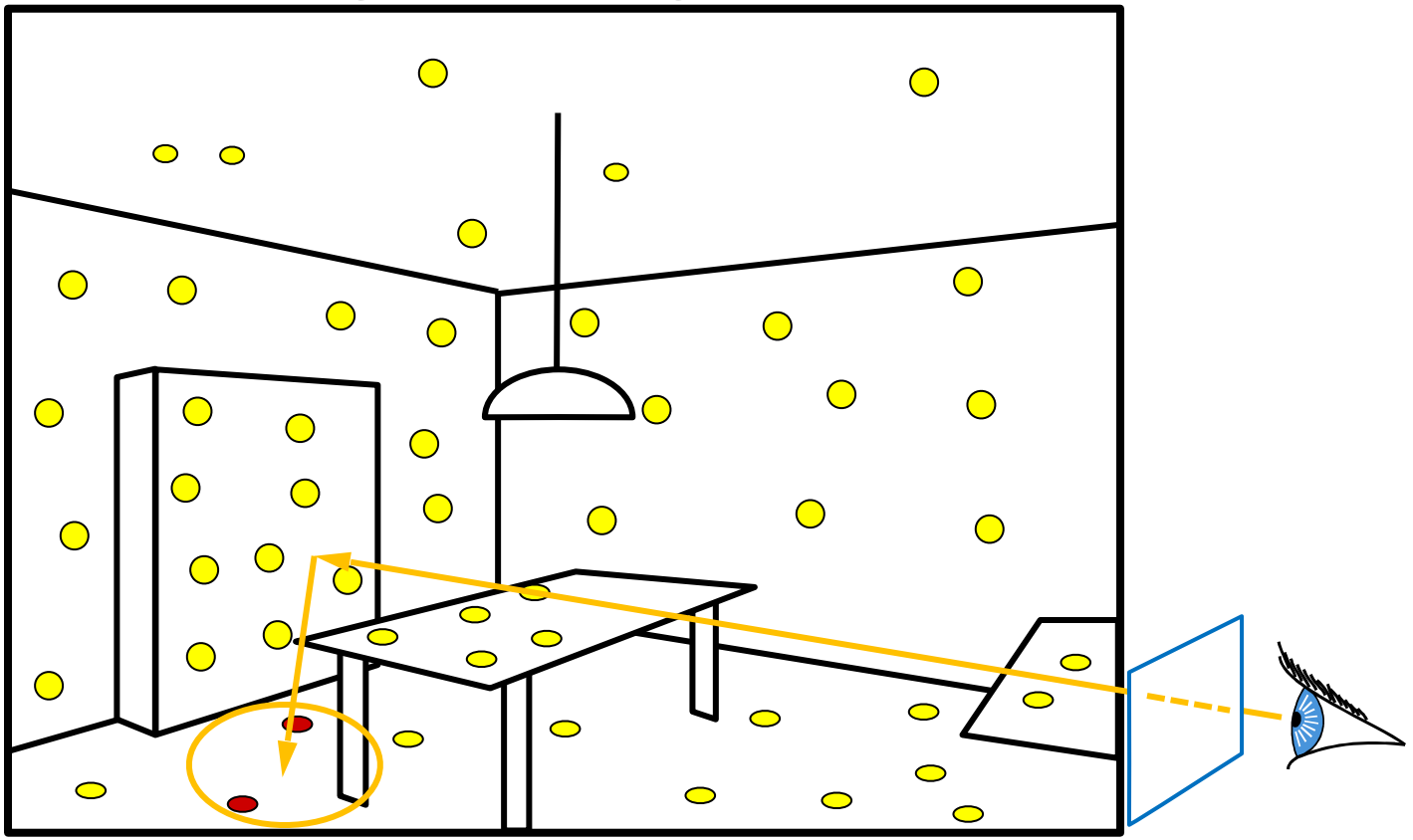
Photon mapping

- Store photons (position + intensity + direction) on the geometry or inside a data structure

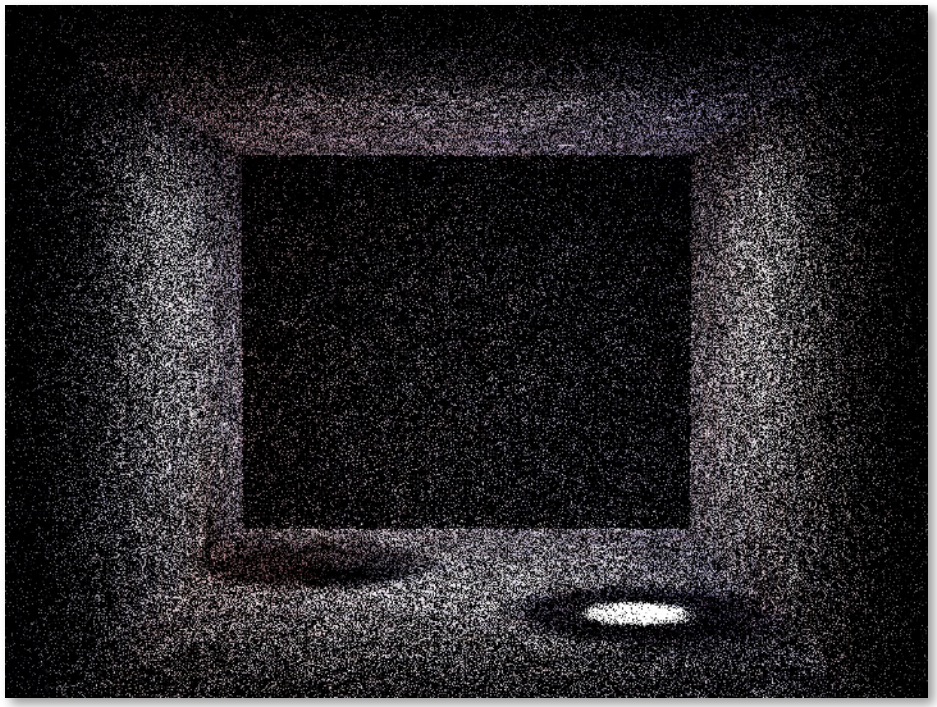


Photon mapping - rendering

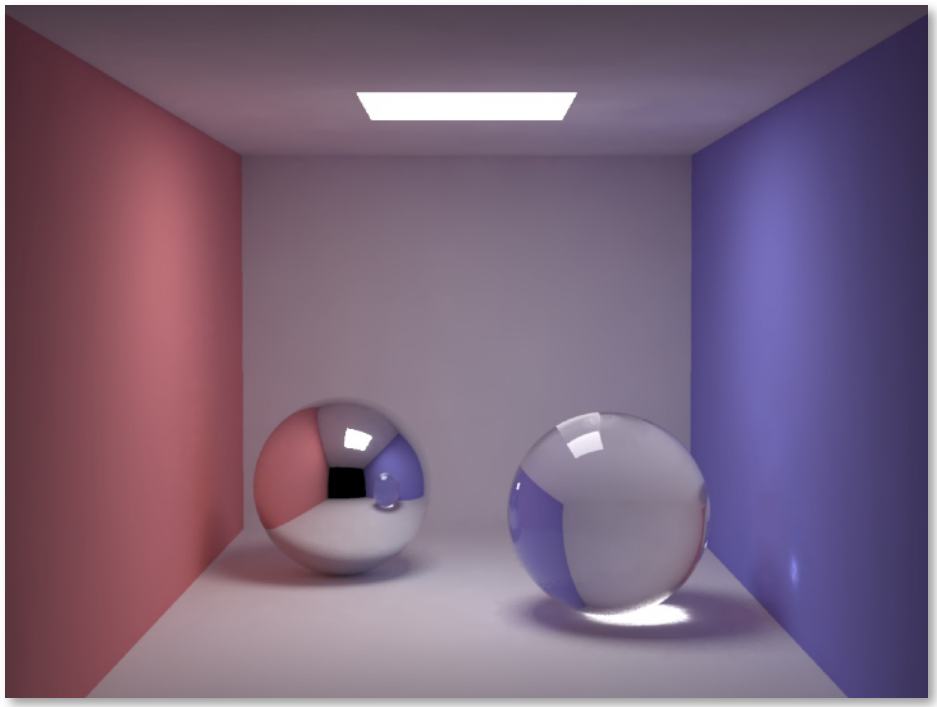
- Throw primary rays
- Radiance for secondary rays by gathering radiance from neighbouring photons



Results



Photon map



Final rendering

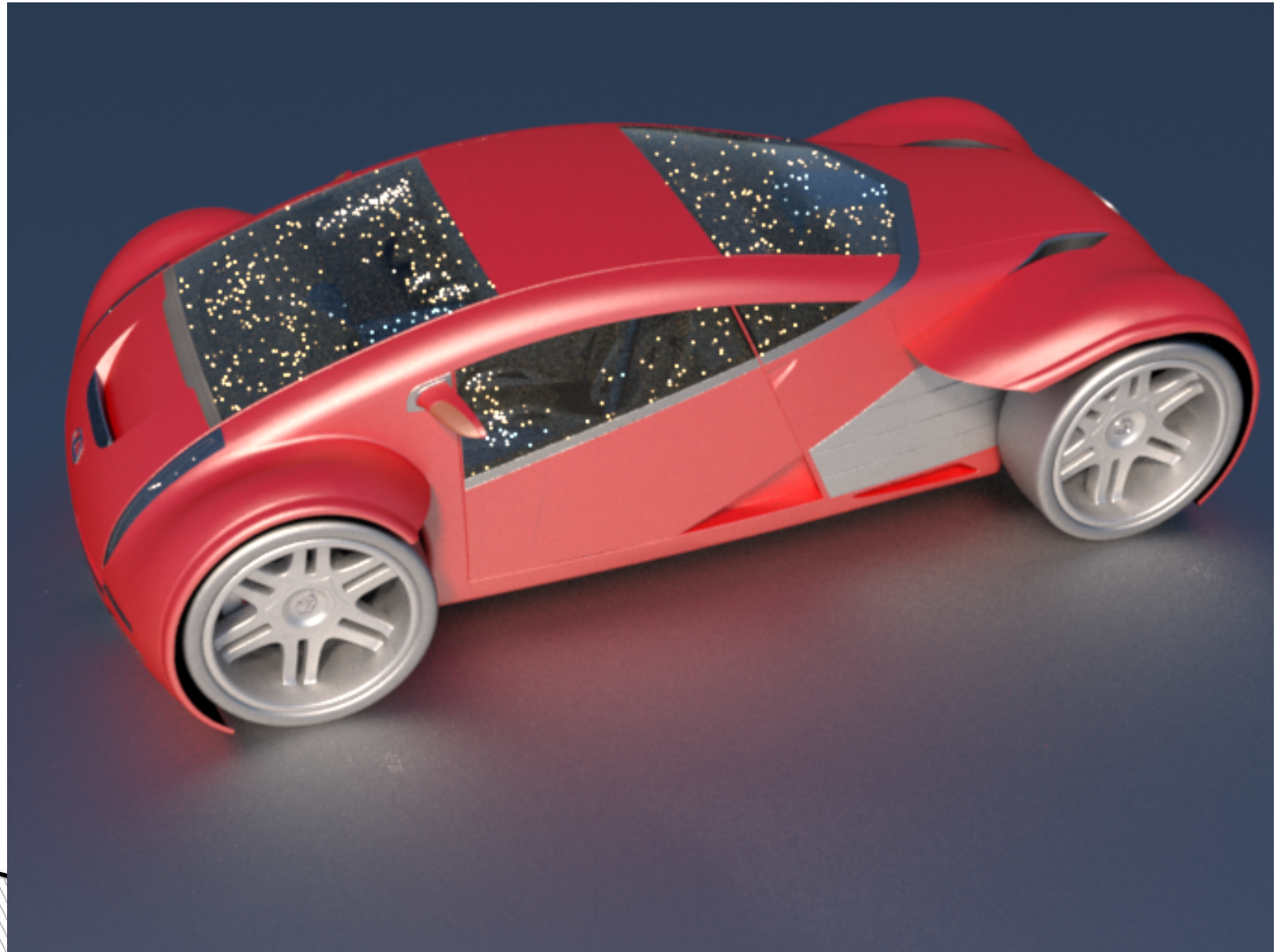
Results

- Jensen (1996)
  - Direct visualization of the photon map: 6min

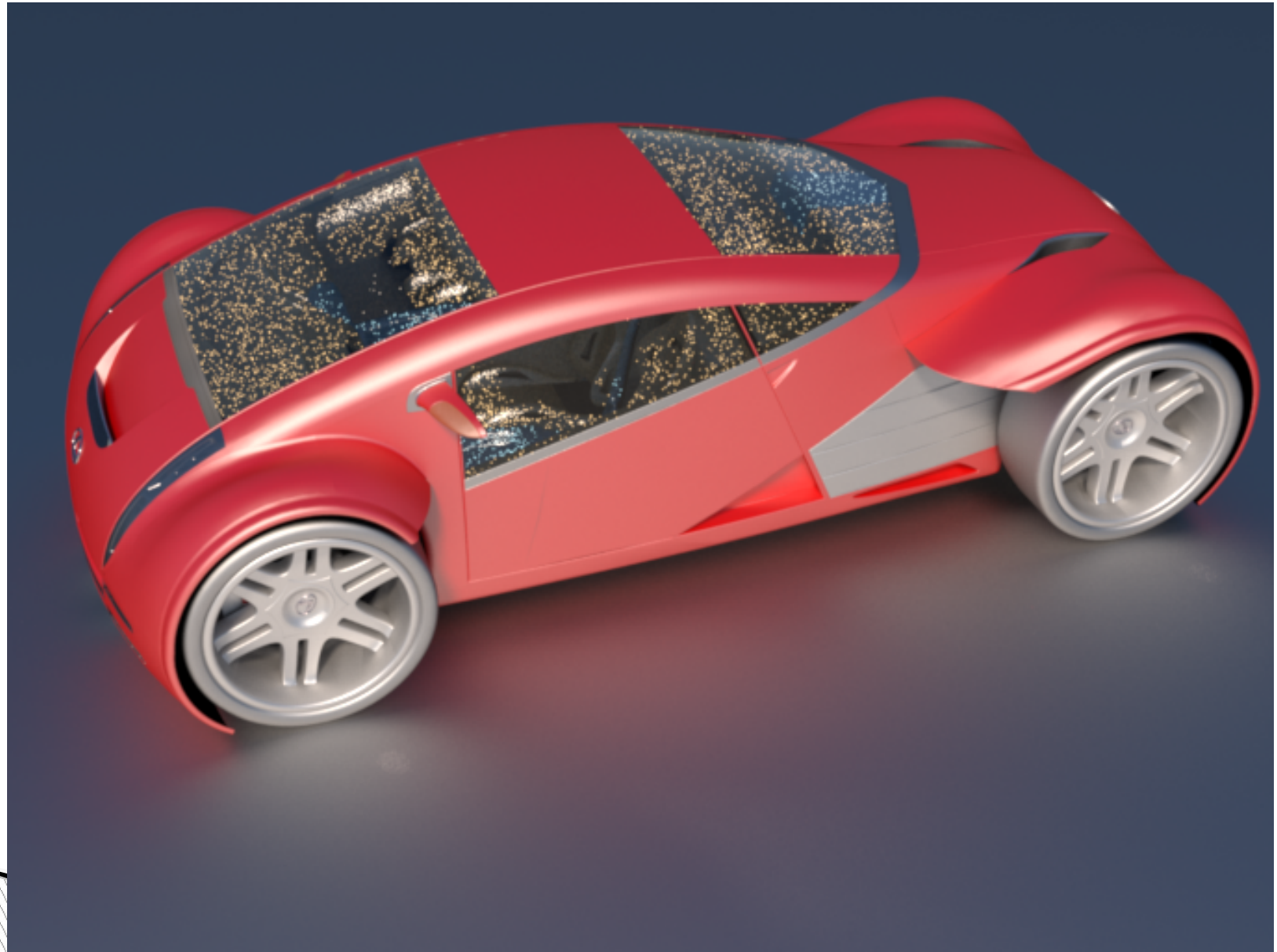
Results

- Jensen (1996)
  - Final gather pass: + 51 mn

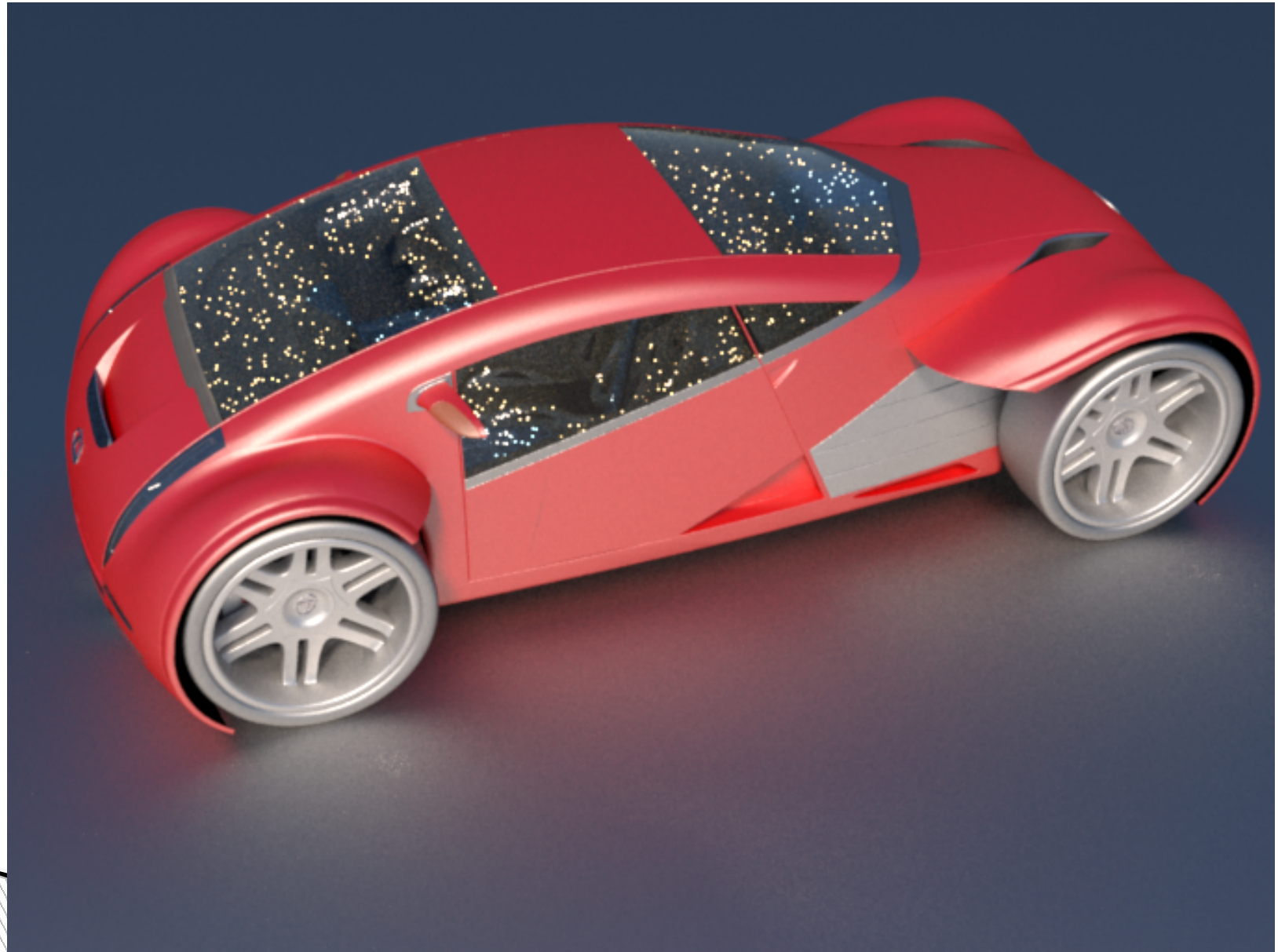
PM: 10k p, 10k cp, 32 spp, 20 mn



PM: 10k p, 10k cp, 256 spp, 1.3h



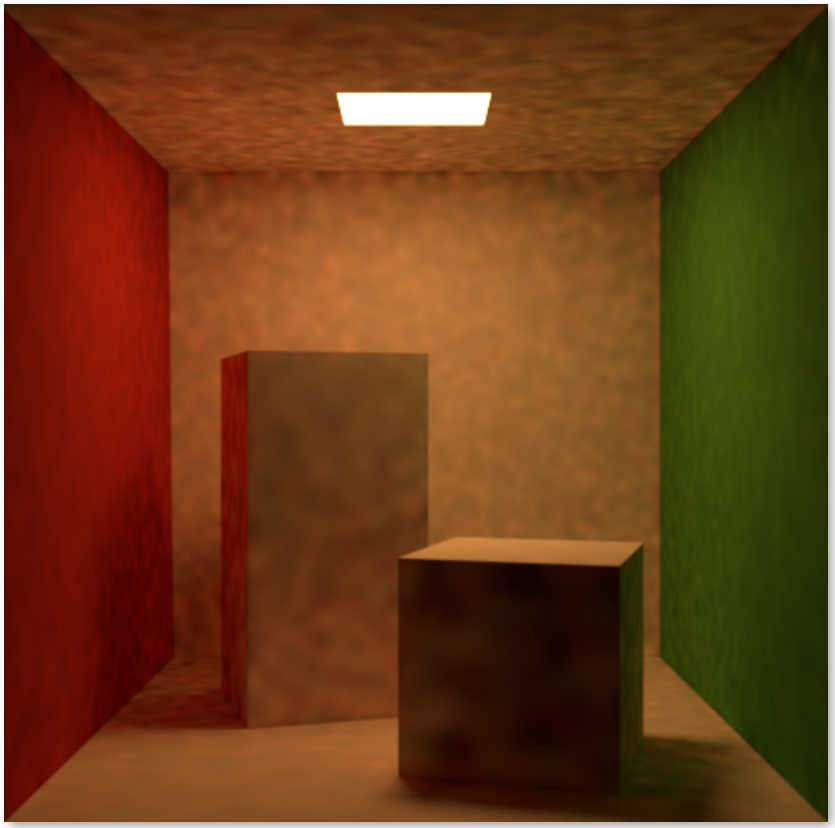
PM: 100k p, 100k cp, 32 spp, 1.8h



PM: 100k p, 100k cp, 256-8192 spp, 2.3 h

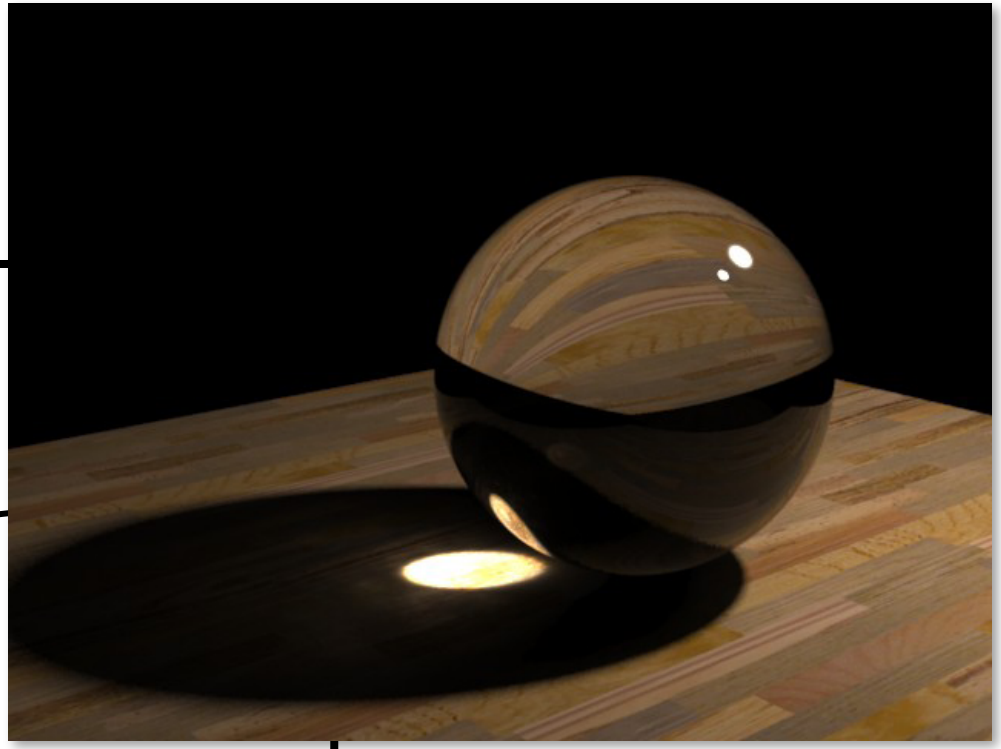
Results

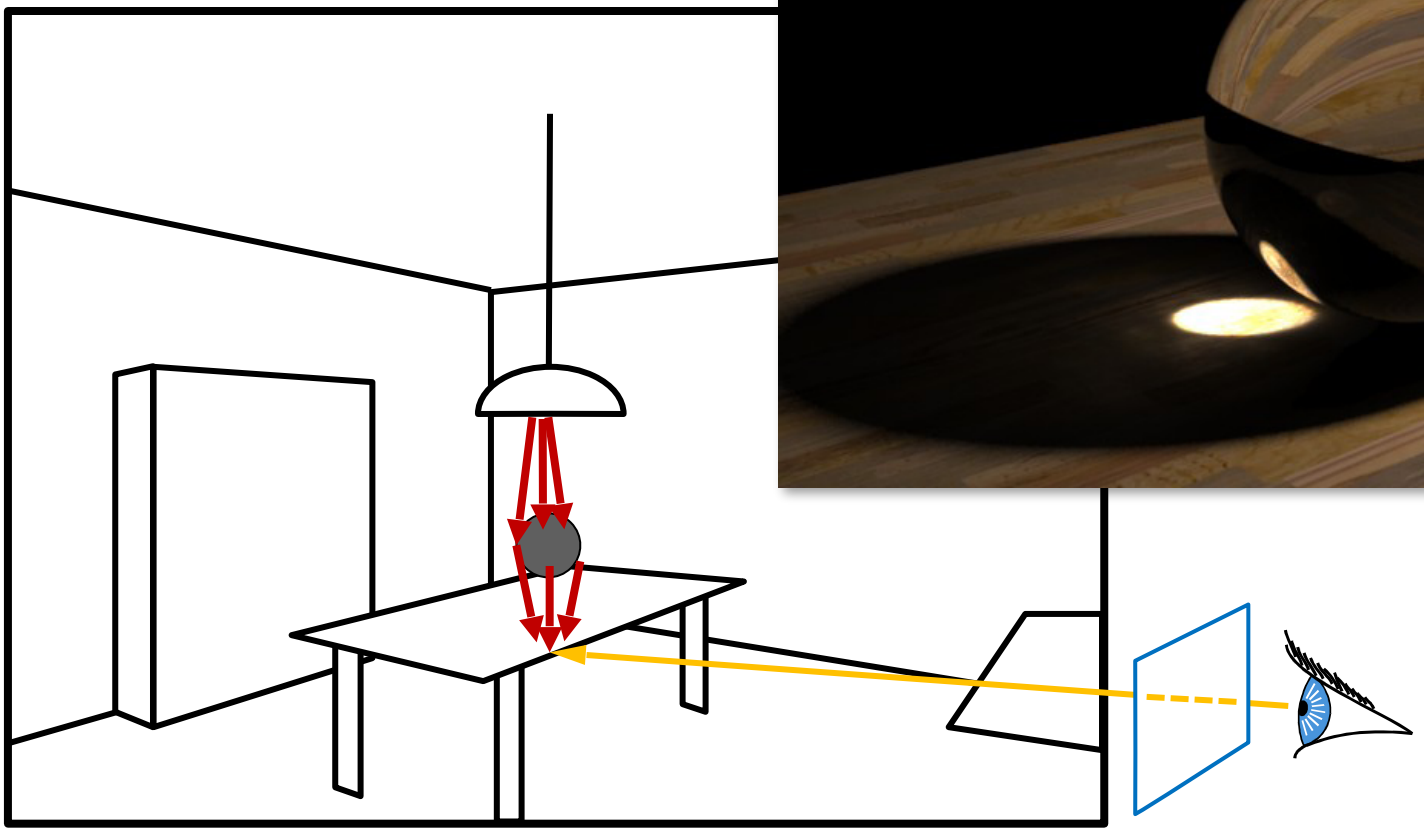
Mitsuba (http://www.mitsuba-renderer.org/)



Caustics

- Separate Photon map for refraction





Photon Mapping: summary

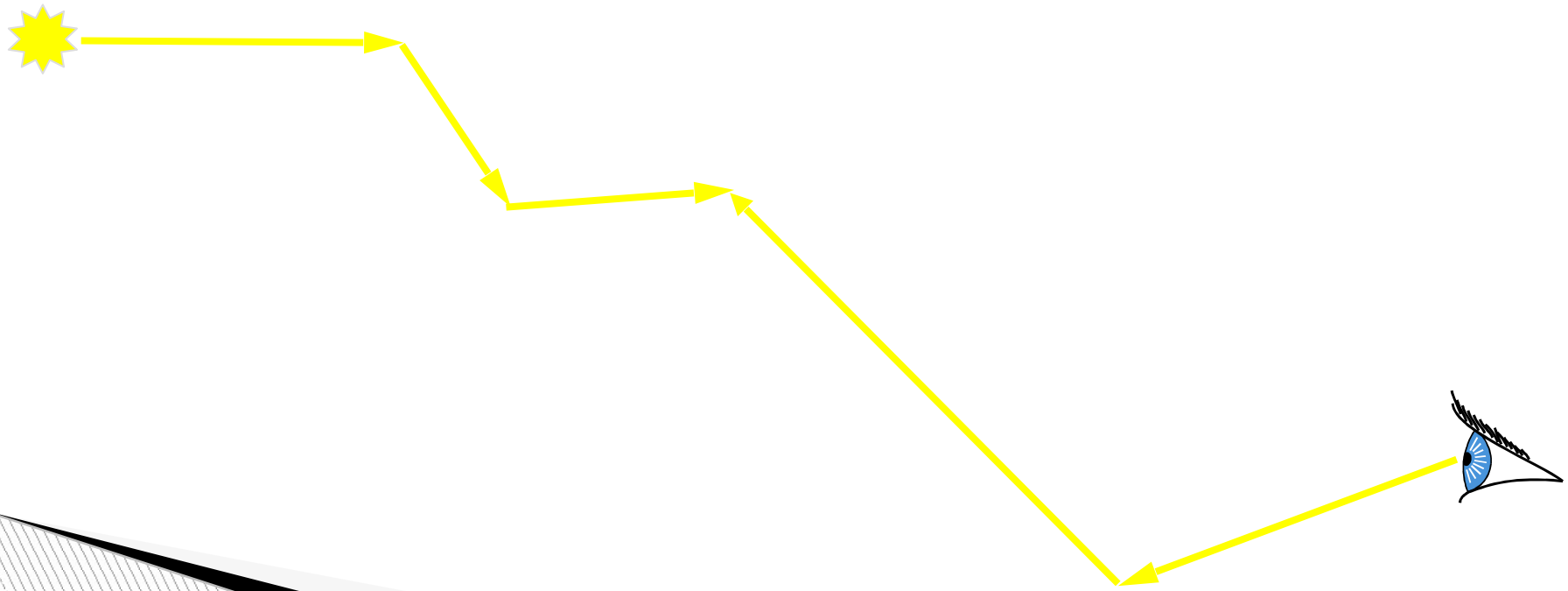
- Pre-computation view-point independent
  - Storage on the surfaces
- Good representation for caustics
- Noisy: smoothing based on the samples
  - Reconstruction of the radiance function
- Can be coded in two passes with a ray-tracer
  - One pass for each direction

Metropolis Light Transport

- Some paths are hard to find
- Explorer path space
  - BDPT: treat each path independently
  - Metropolis: use existing paths (if they're good)
- Start with a given path from eye to light
- Mutations + perturbations
  - Accept mutations: with probability
  - Importance sampling for energy carried

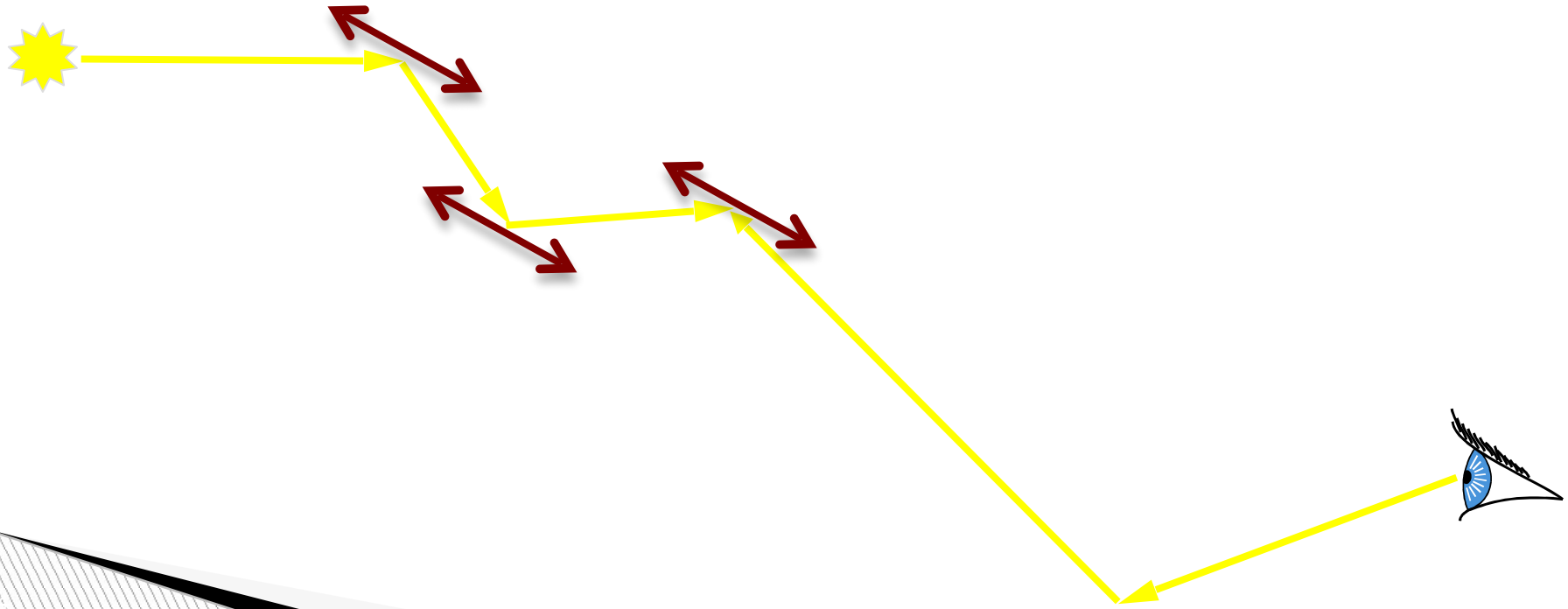
Metropolis Light Transport

- Start with a light path from eye to source:



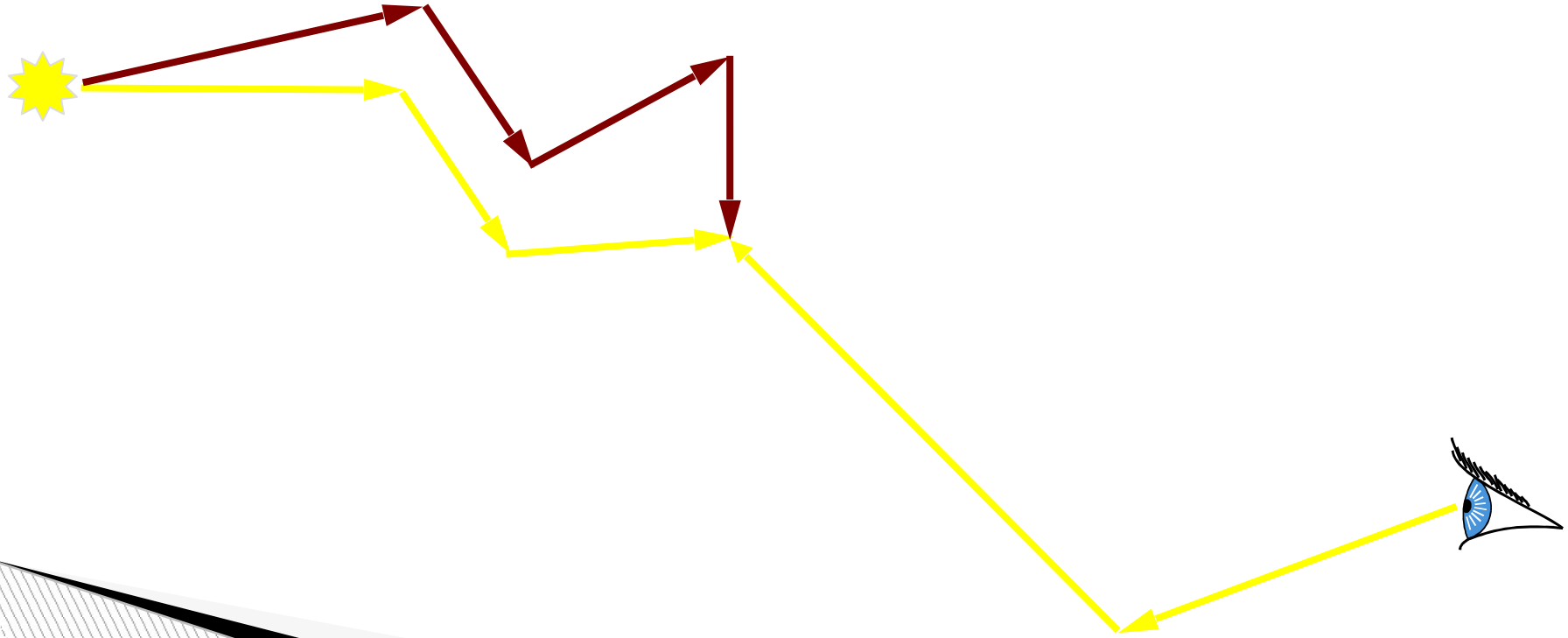
Metropolis Light Transport

- Start with a light path from eye to source
- Random perturbations
  - Keep / don't keep

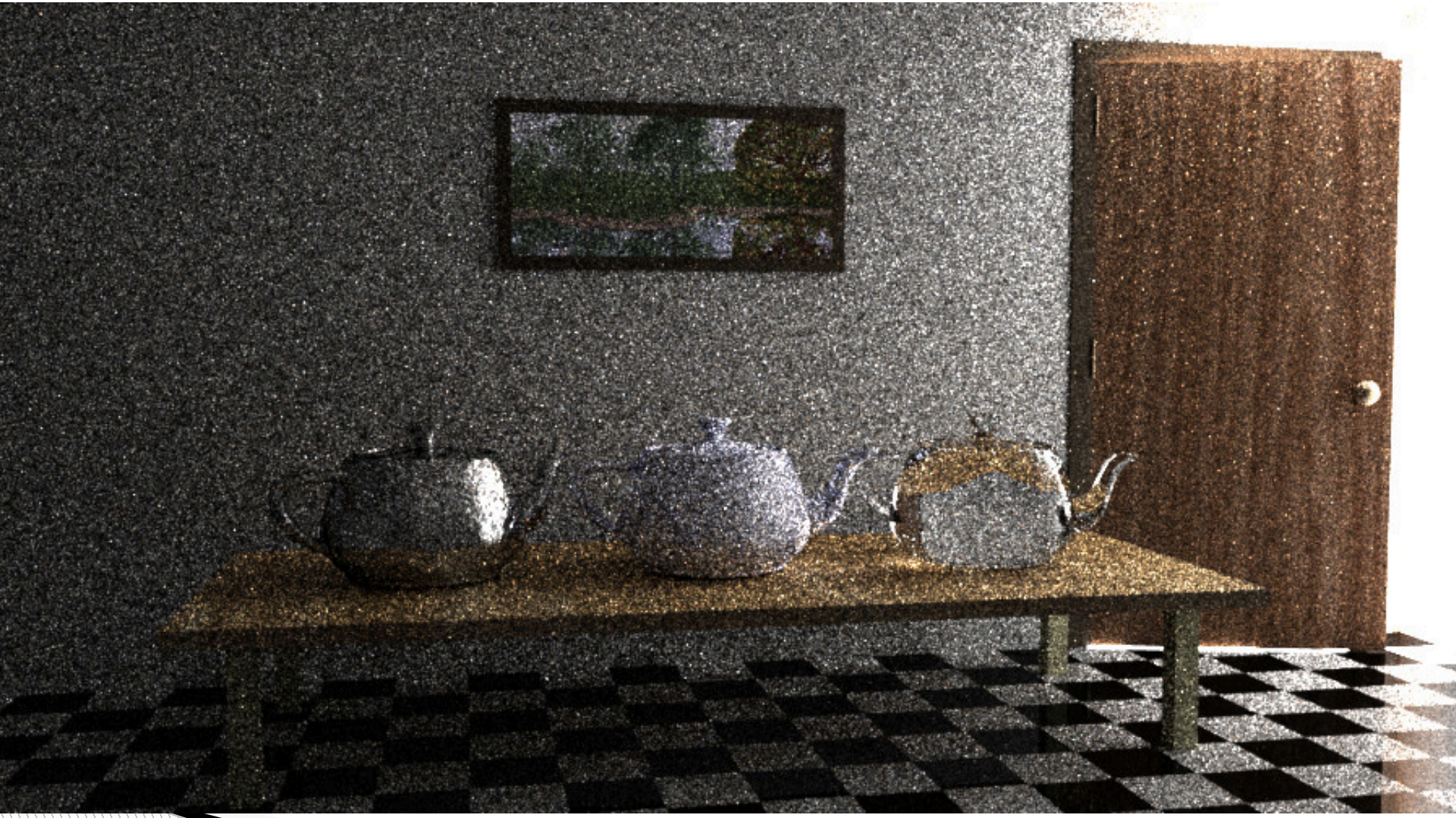


Metropolis Light Transport

- Start with a light path from eye to source
- Random mutations (adding vertices)
  - Keep / don't keep

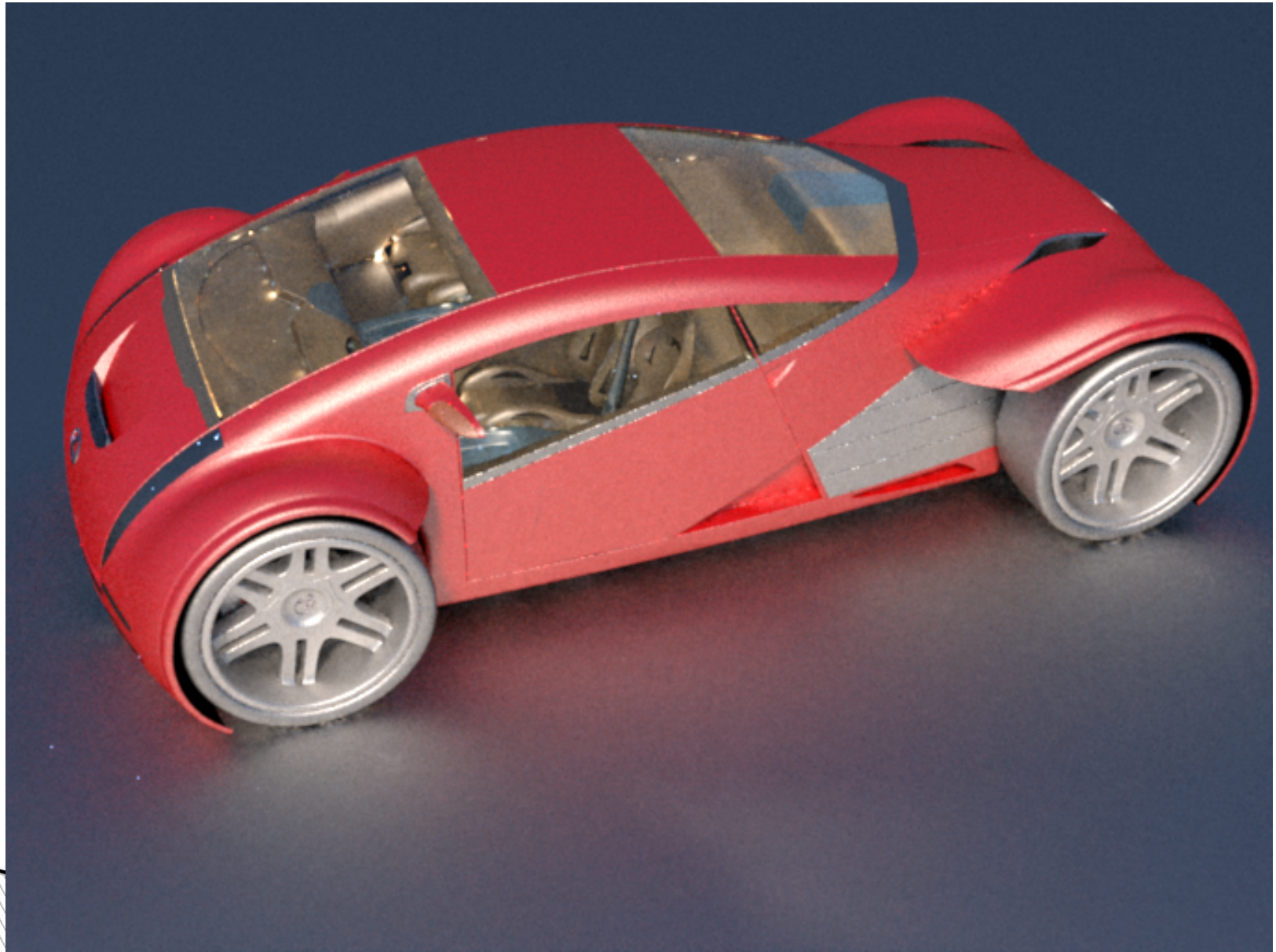


Bi-Directional Path Transport

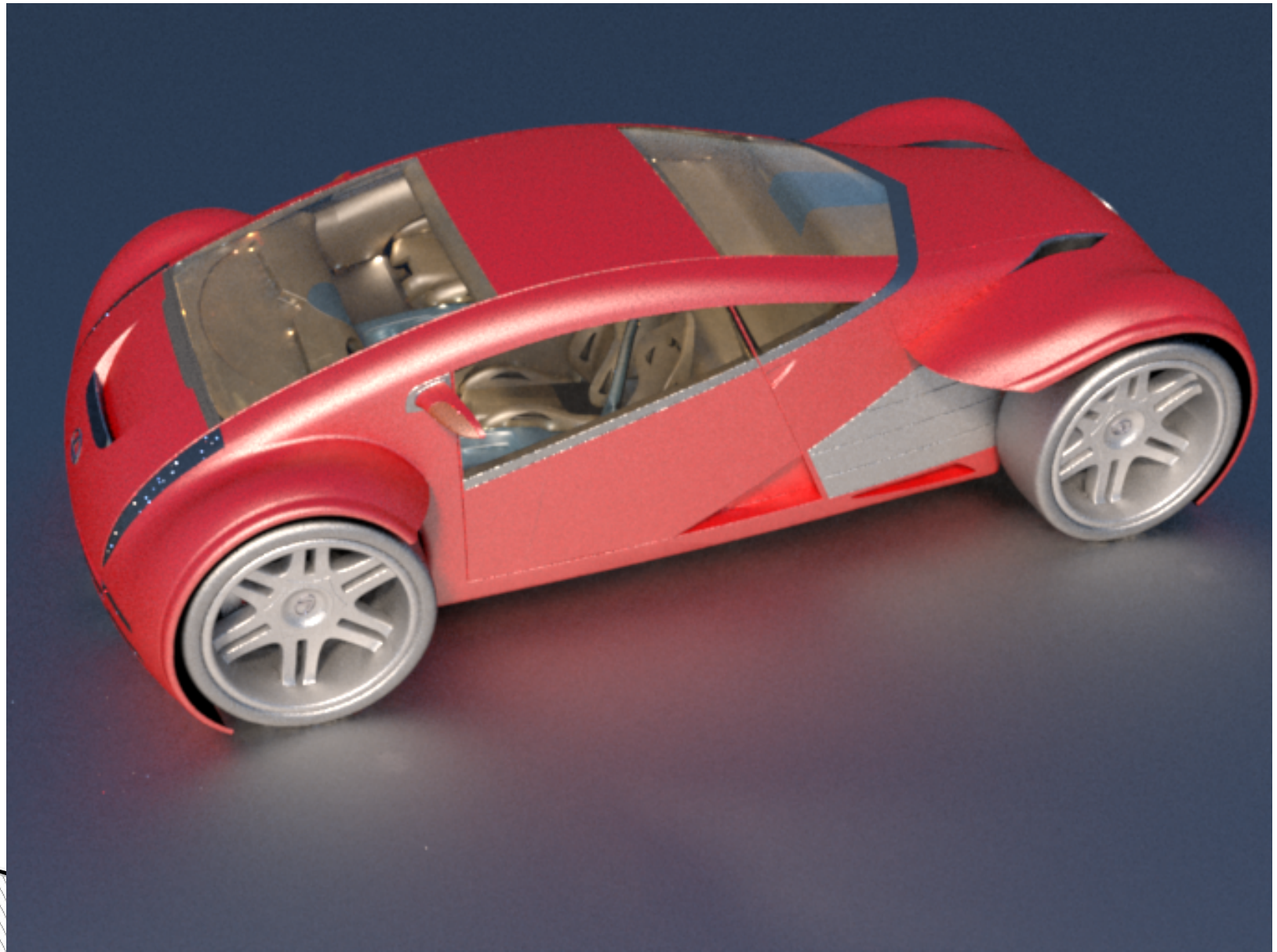


Metropolis light transport

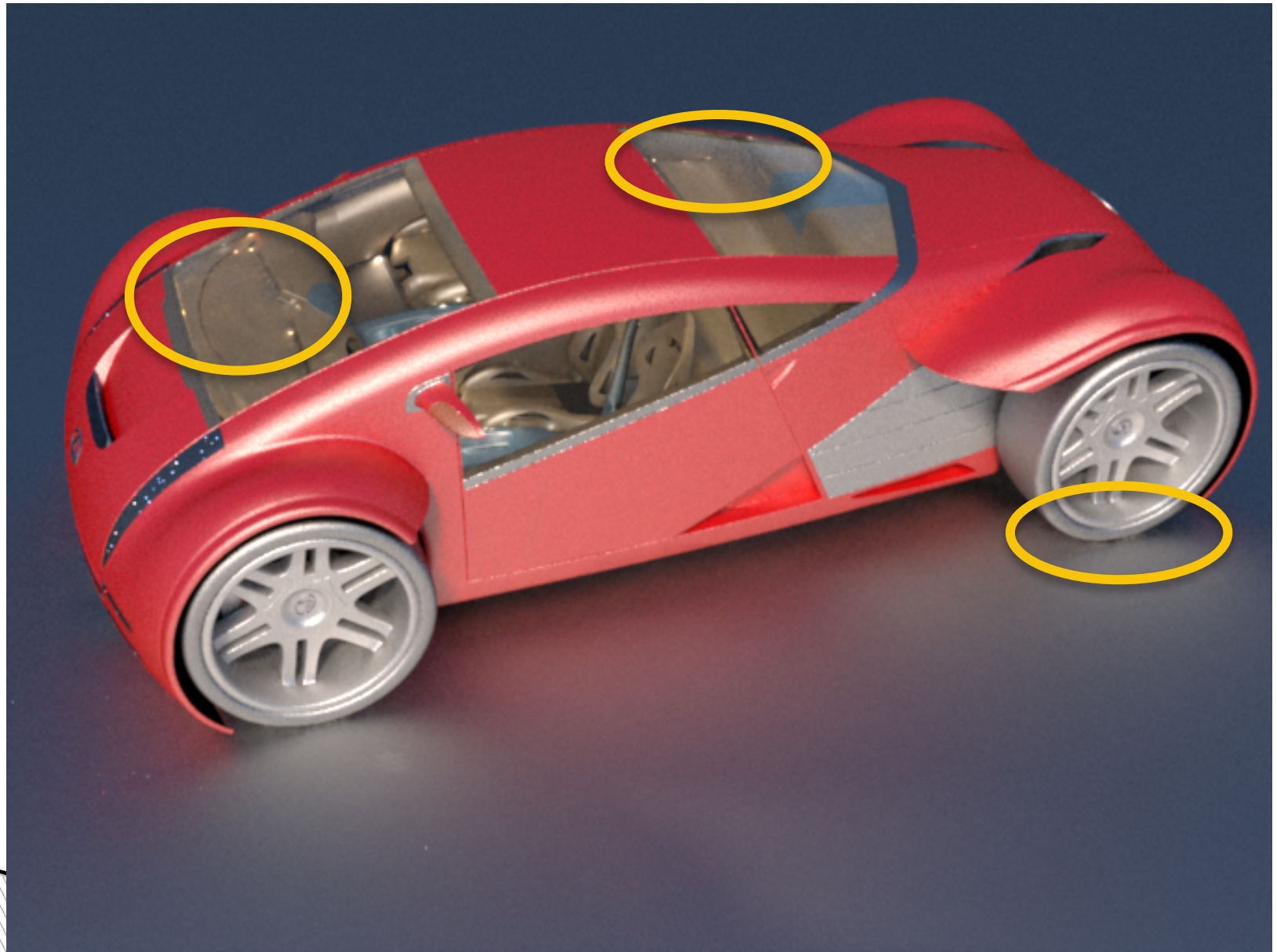
MLT, 256 mutations/pixel (2.9 mn)



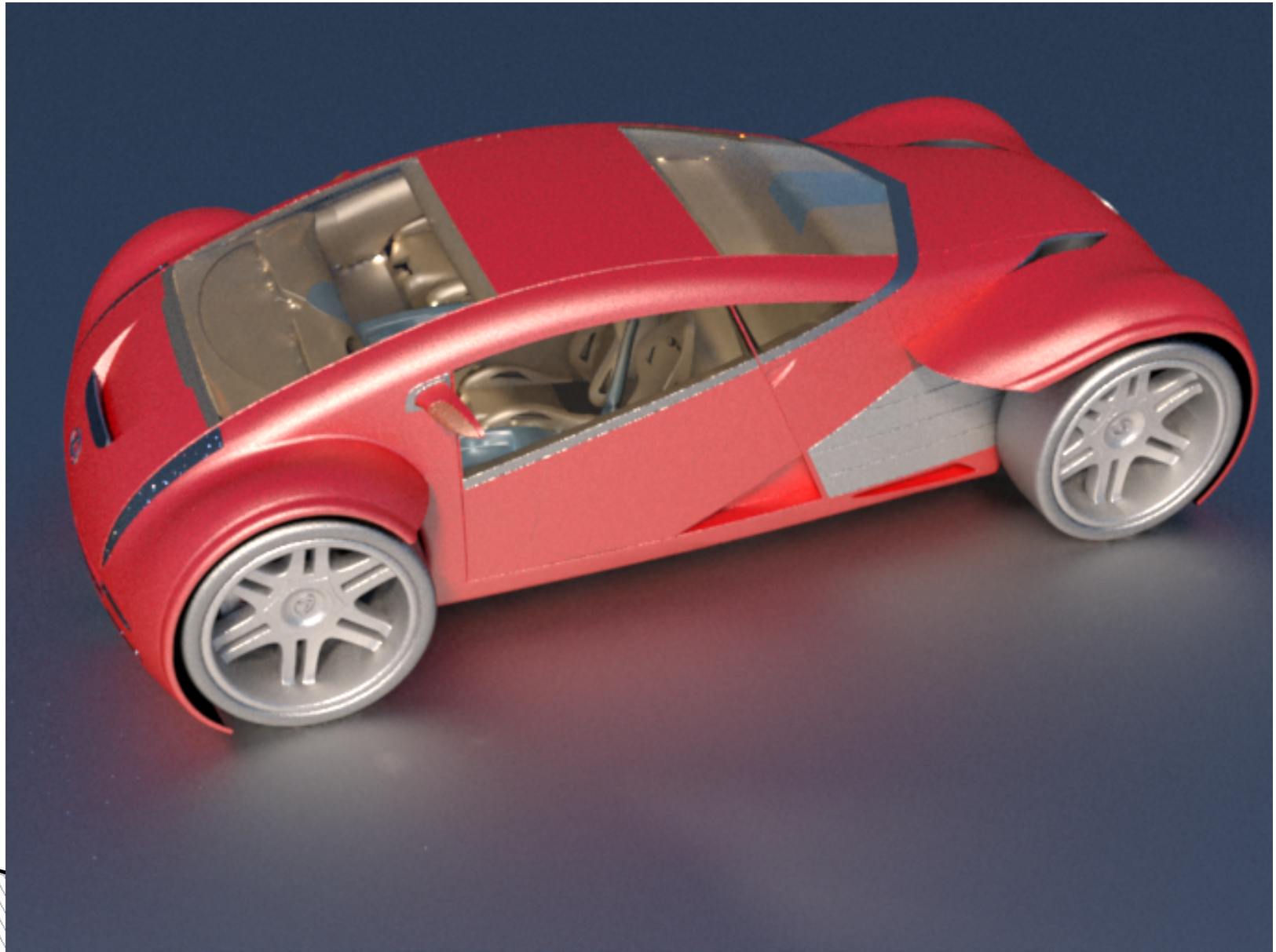
MLT, 1024 mutations/pixel (12 mn)



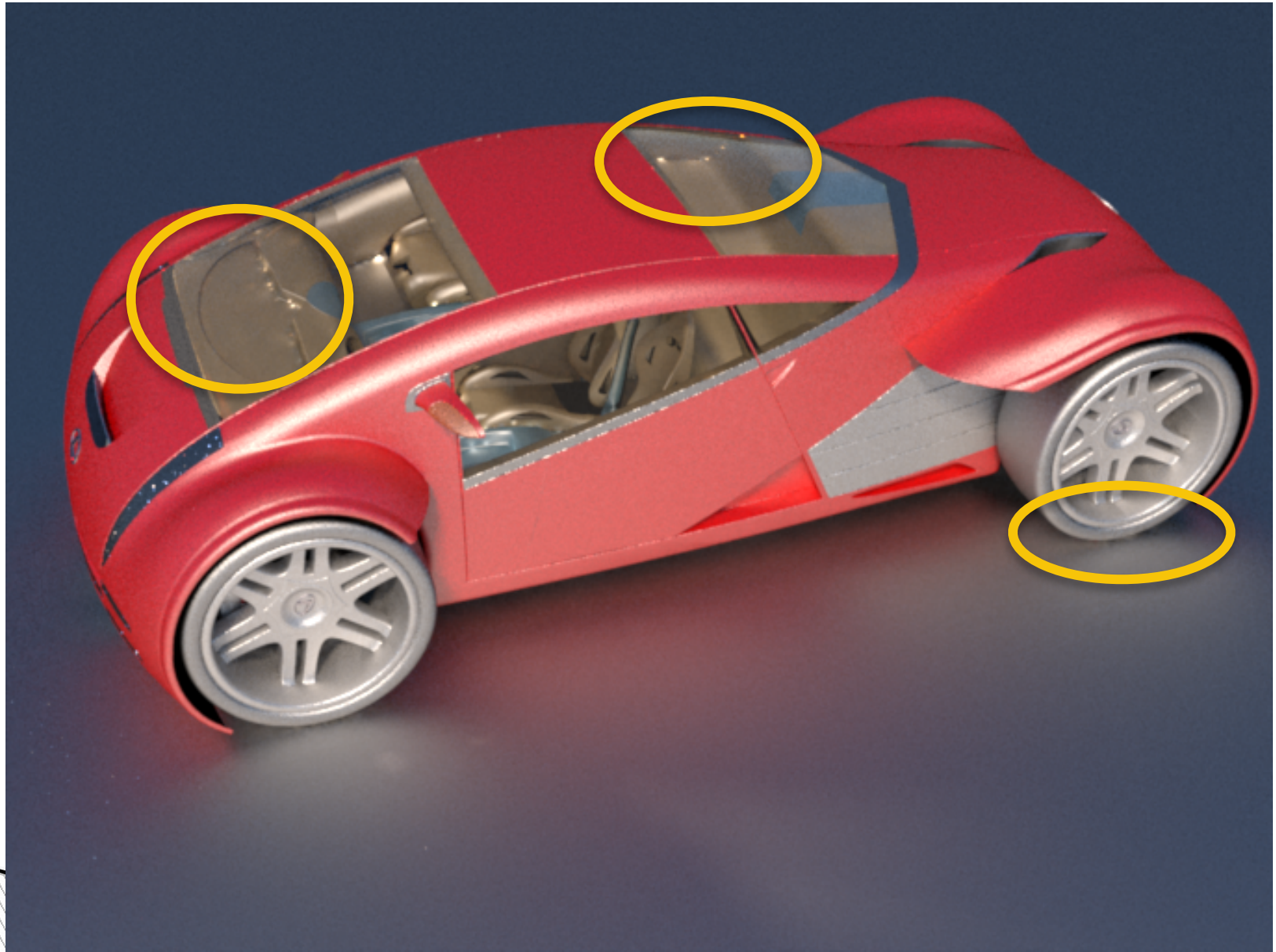
MLT, 1024 mutations/pixel (12 mn)



MLT, 2048 mutations/pixel (23 mn)



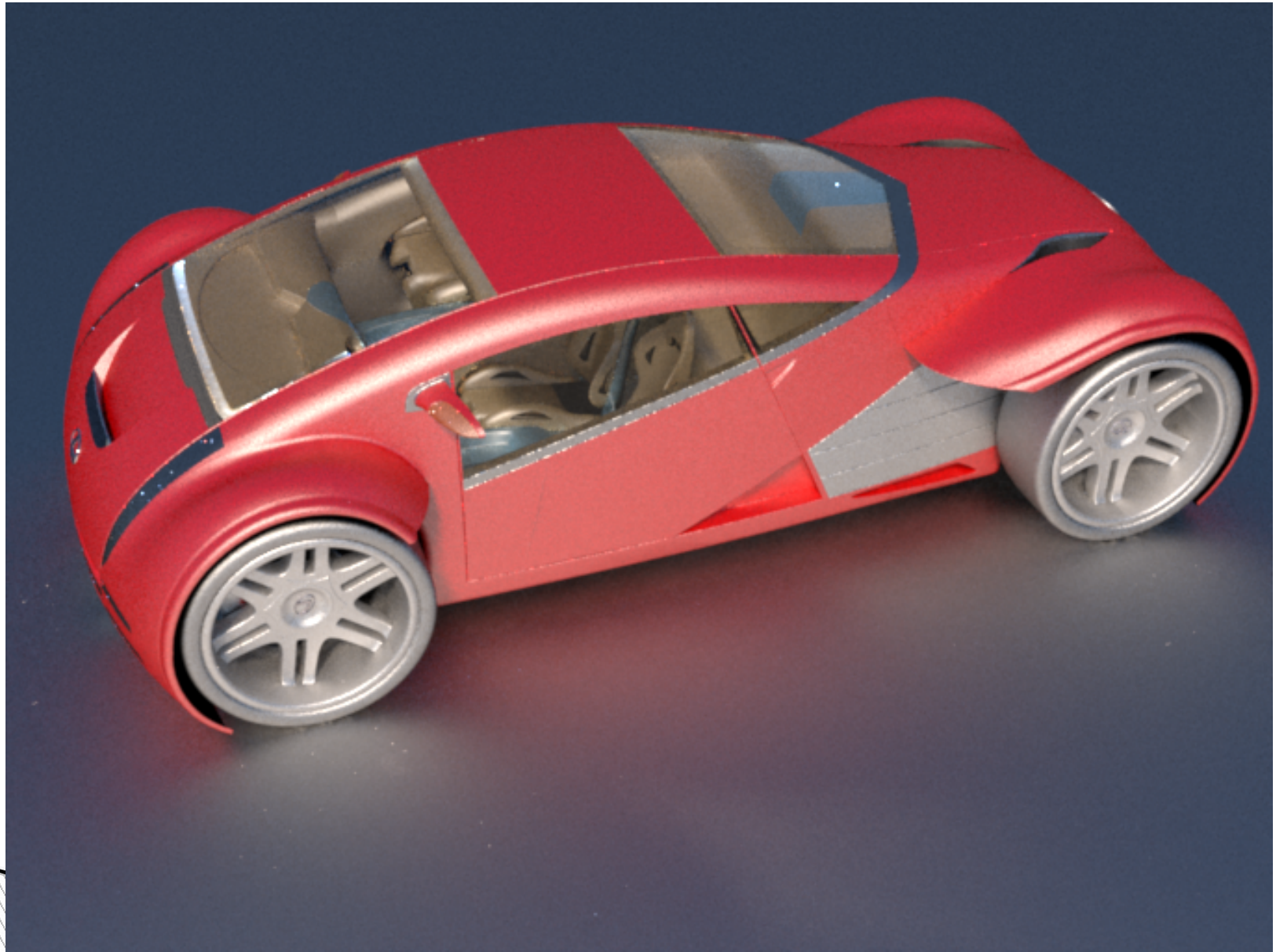
MLT, 2048 mutations/pixel (23 mn)



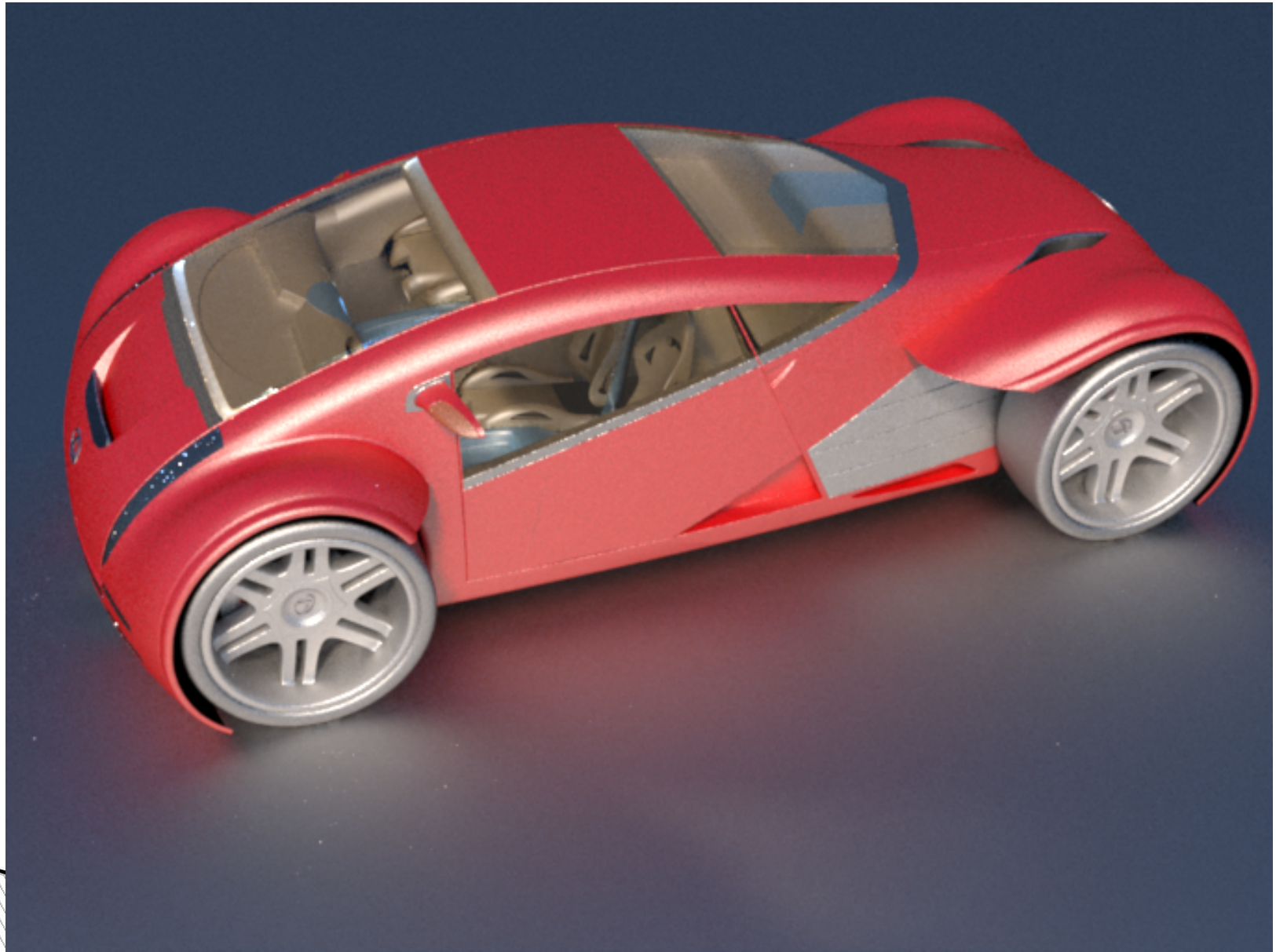
MLT: what's happening?

- Test scene n° 1: tailored for MLT/BDPT
- Scène n° 2:
  - Some paths are hard to find
  - When they're found, higher importance
- Solution: manifold exploration
  - Interesting paths = manifold (in path space)
  - Stay on this manifold
  - Constrained exploration of path space
  - Extension: Half-Vector Space exploration
- Aliasing

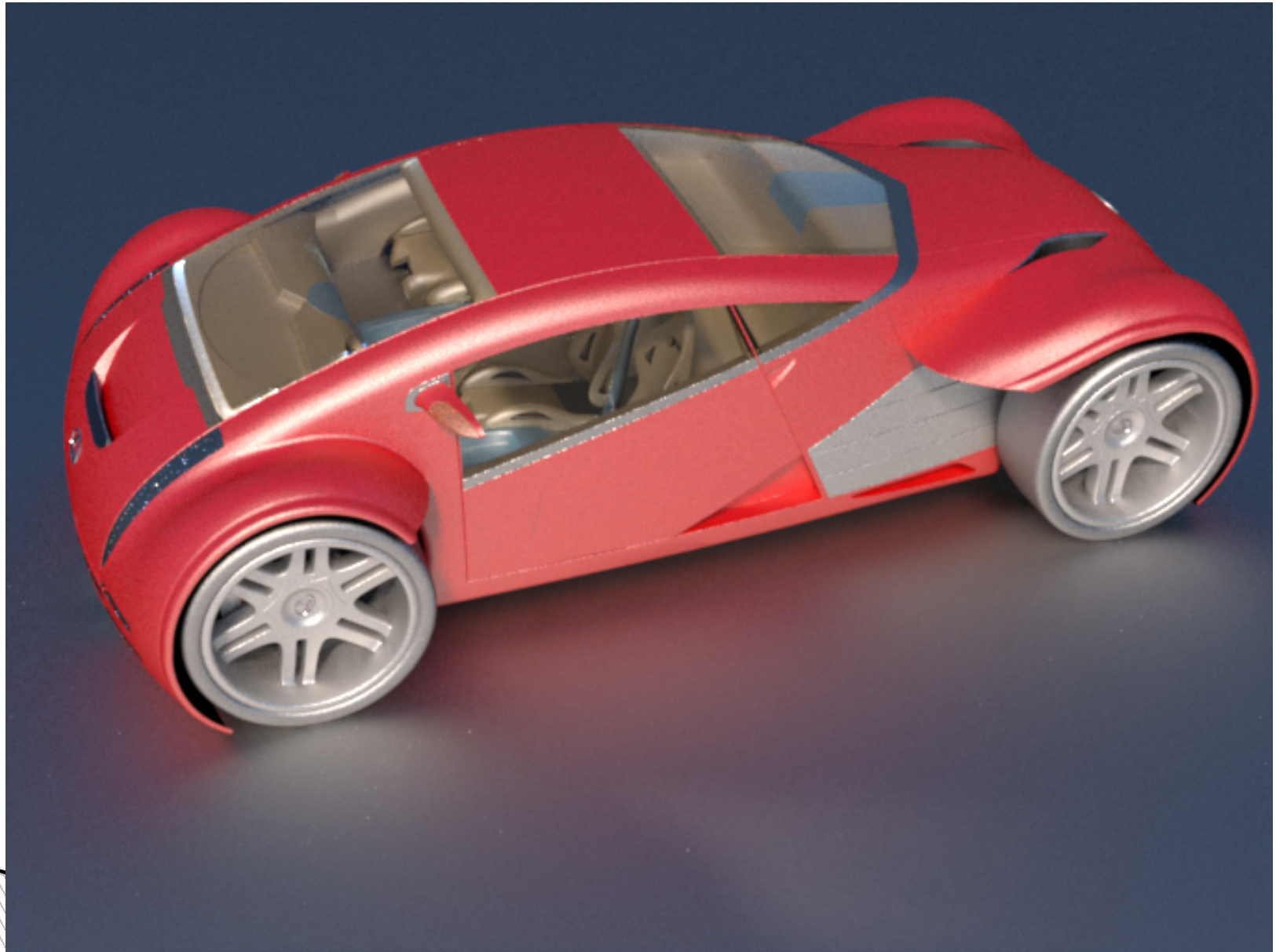
MEMLT, 1024 mut./pixel, 17 mn



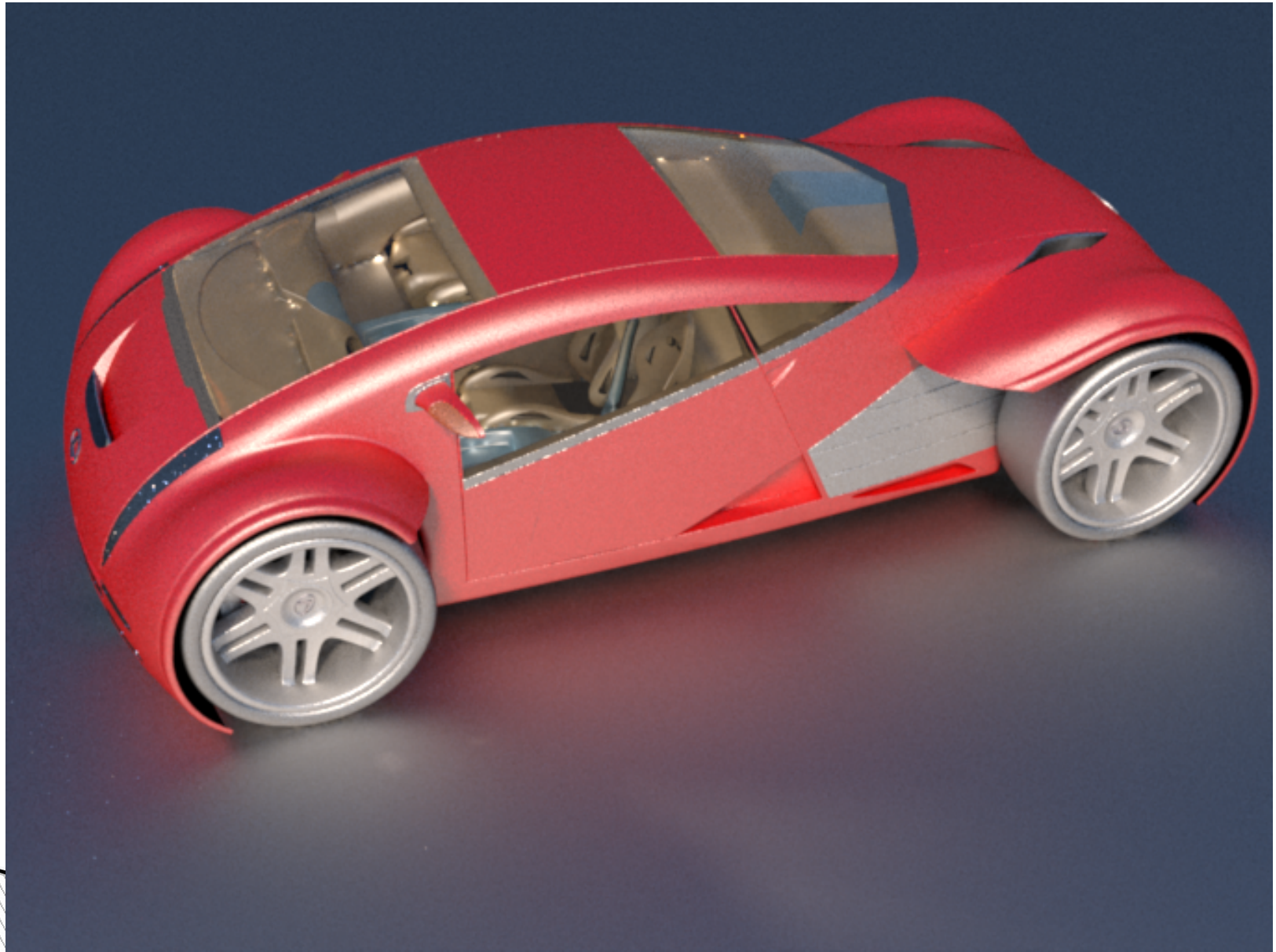
MEMLT, 2048 mut./pixel (27 mn)



MEMLT, 4096 mut./pixel (54 mn)



MLT, 2048 mutations/pixel (23 mn)



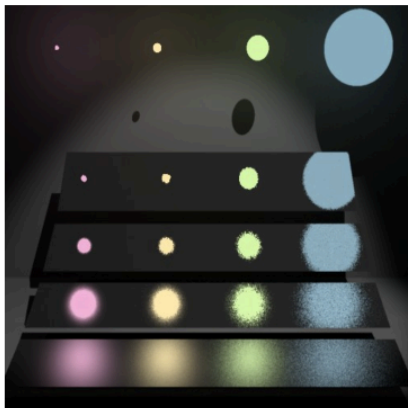
Rendering in the movie industry

(As of August 2019; sources: Weta/Disney/Sony)

- Temporal coherence is essential:
  - Similar image for following frames
  - Not necessarily *exact*, but same bugs
- Frame *consistency* also important:
  - Same time for each frame
- So... back to path tracing:
  - Path guiding / importance resampling
  - Denoising

Resampling Importance Sampling

- Importance sampling: guide rays based on BRDF
- Multiple Importance sampling:
  - Combine BRDF-sampled rays and light-sampled rays
  - Can be inefficient



- Importance resampling:
  - Trace a few rays
  - Get an approximate version of outgoing illumination
  - Importance sample based on that

Resampling Importance Sampling

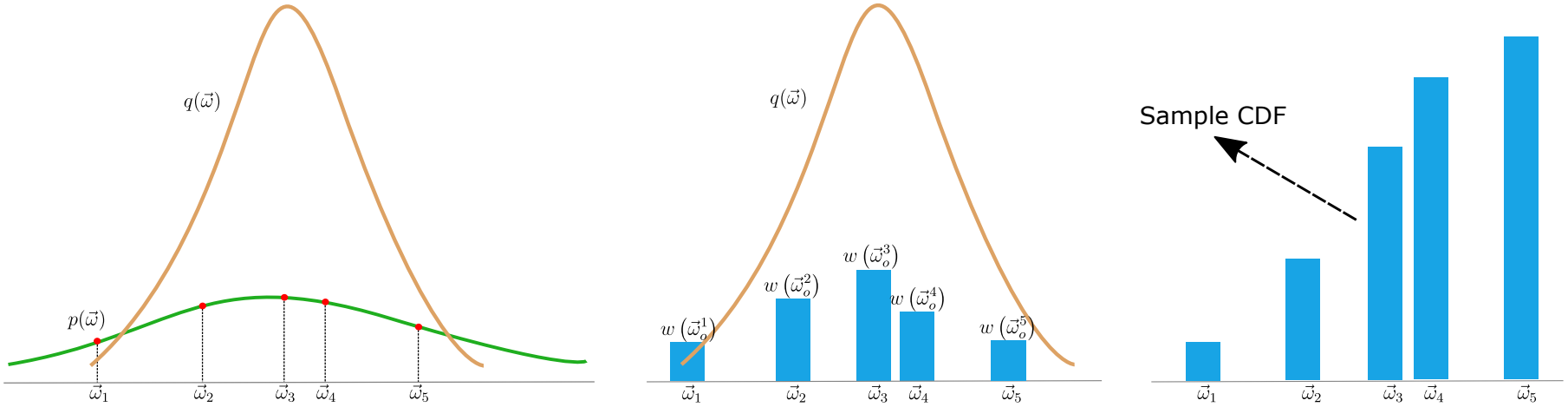


Figure 4. Example of resampling in flatland curves. To sample target function (orange curve), we first choose a function as the source function (green), and then sample the source function and get 5 outgoing direction candidates. In the second step, we evaluate the weight of each candidate which is target function divided by source function. Finally, we compute the Cumulative Distribution Function (CDF) of the weights of the candidates and use it to get the final sample.

Resampling Importance Sampling

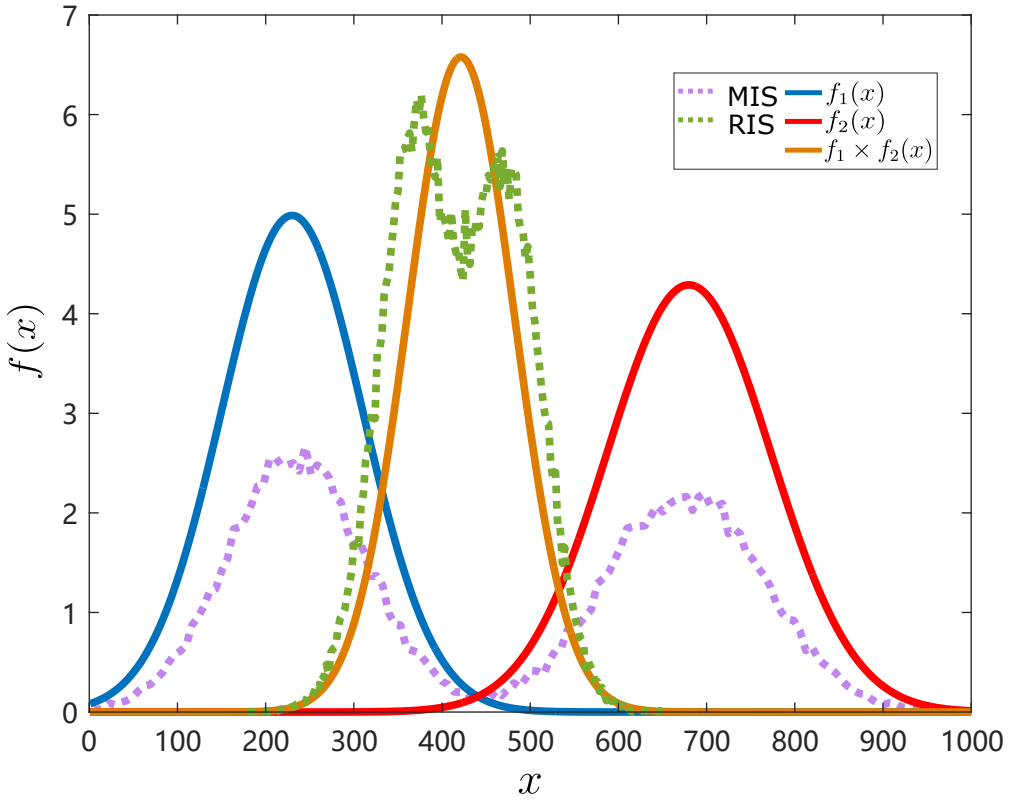


Figure 5. Comparison between MIS and RIS of two function f_1 and f_2 product. The RIS produces result much closer to the target function (product of two functions) than MIS.

Path guiding

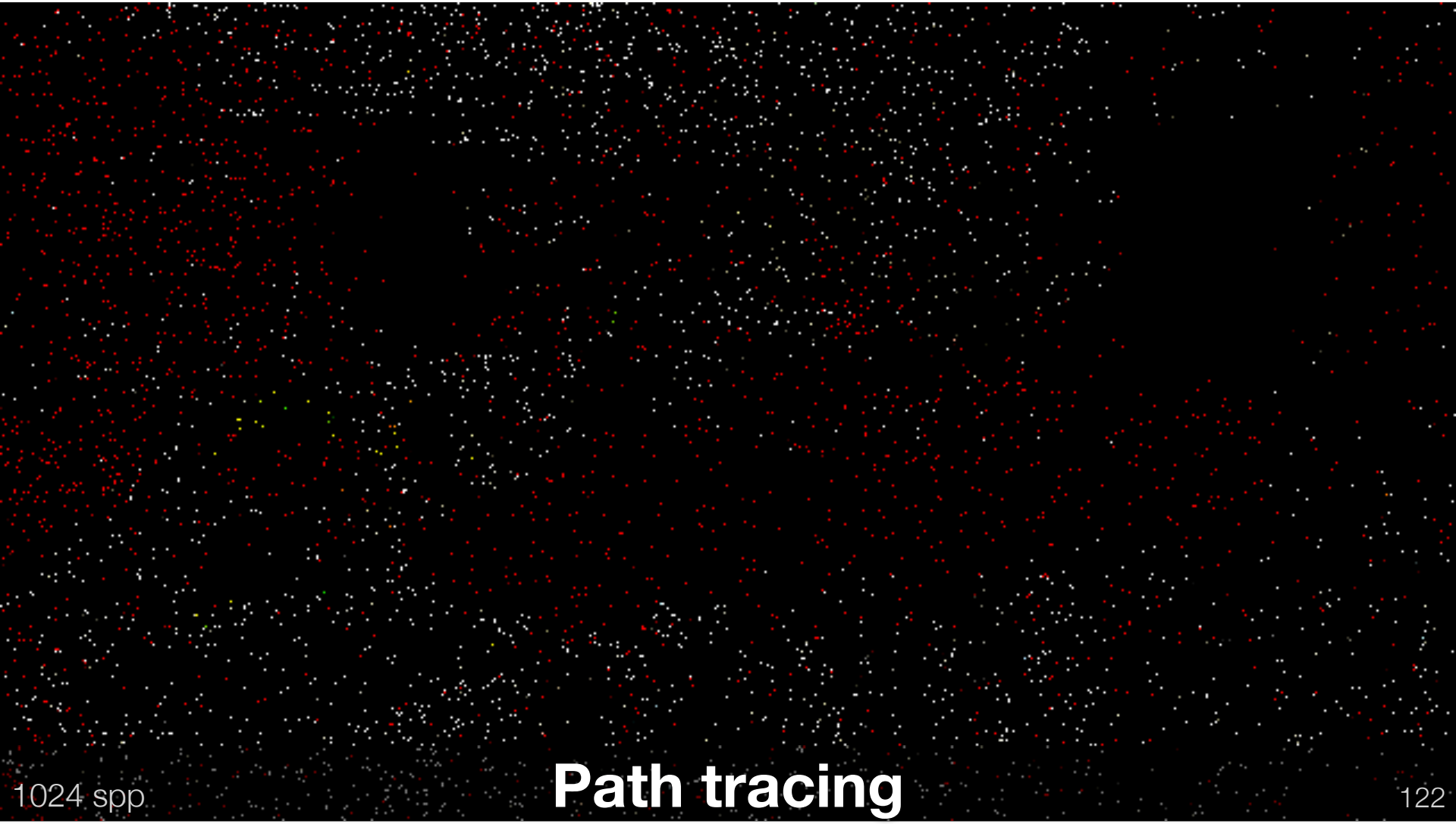
- Slides from Thomas Müller, “Practical path guiding in production”, 2019.

"Practical Path Guiding" in Production

Thomas Müller

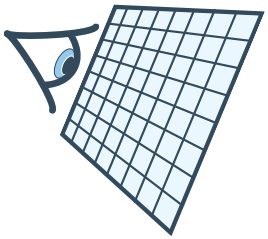
Affiliation: NVIDIA.

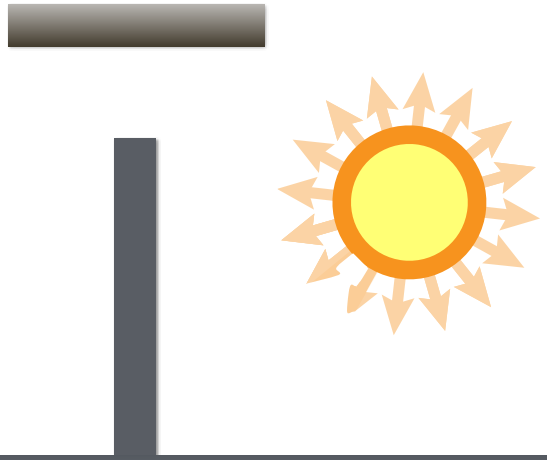
Work done while ETH zürich, Disney Research, cgl computer graphics laboratory


1024 spp
Path tracing

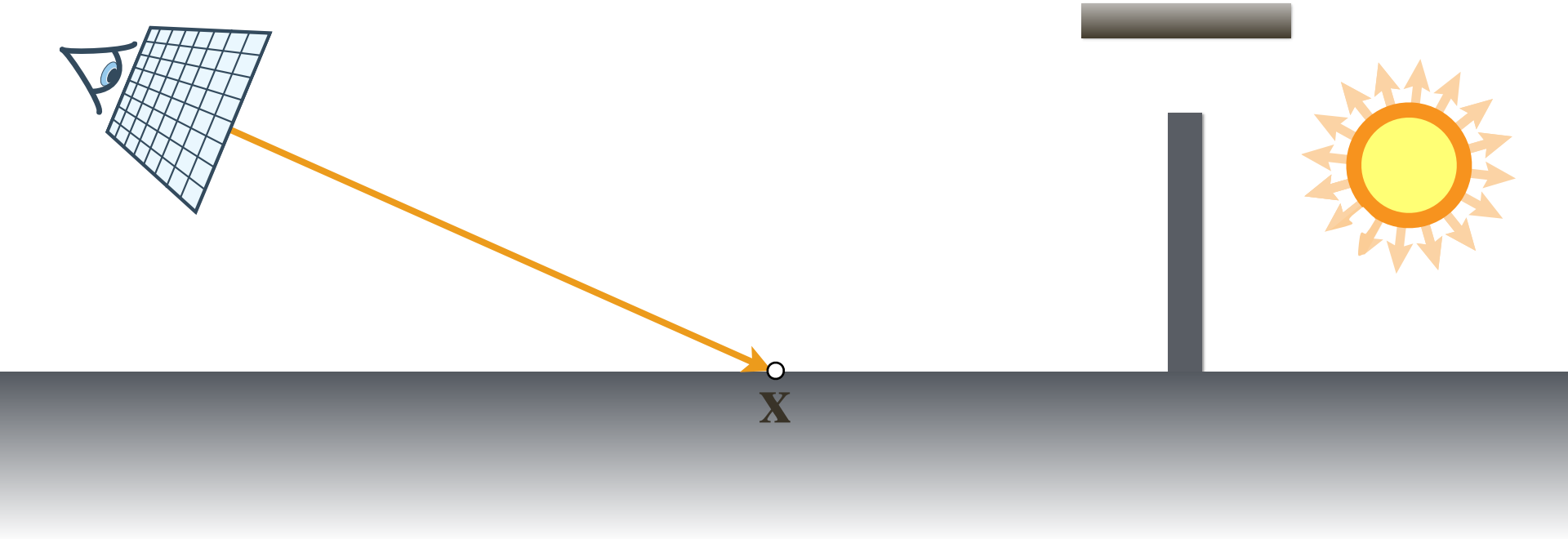

"Practical path guiding" with improvements
1024 spp

Path guiding

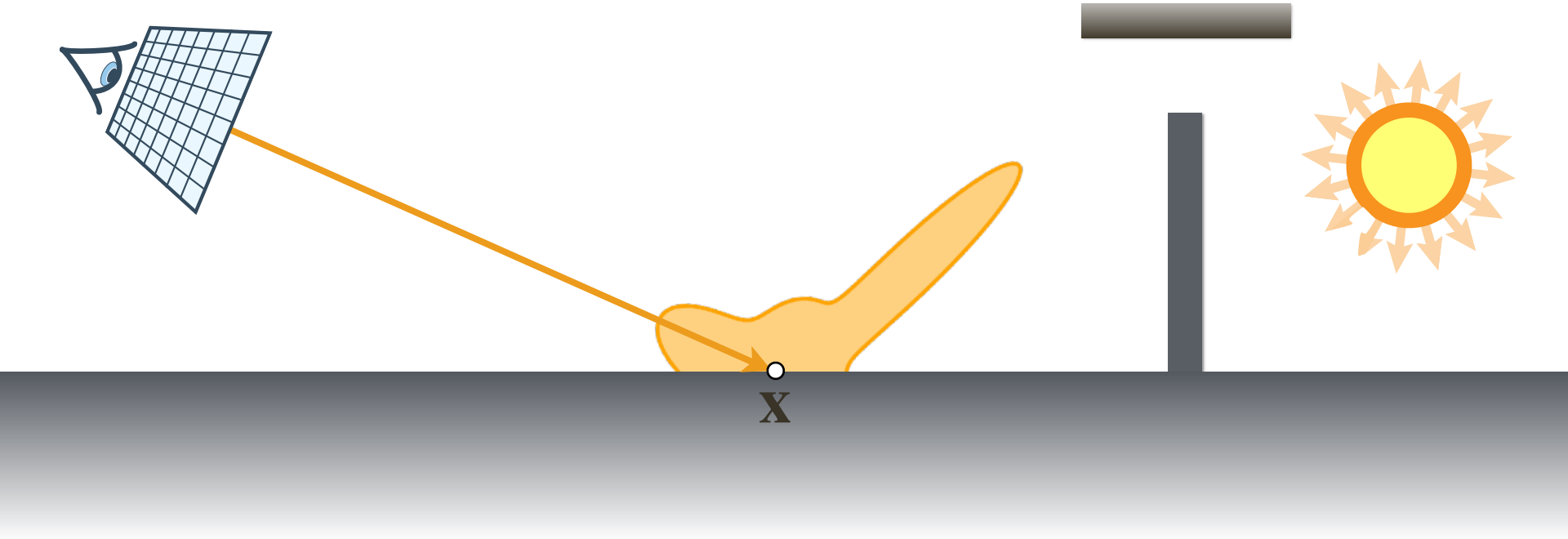




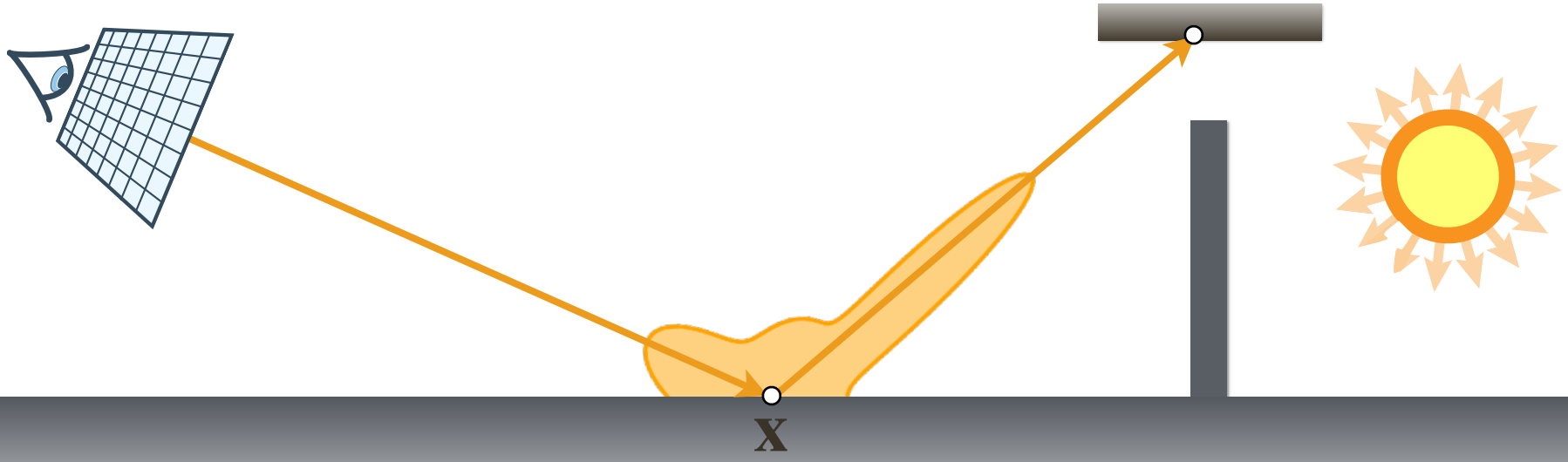
Path guiding



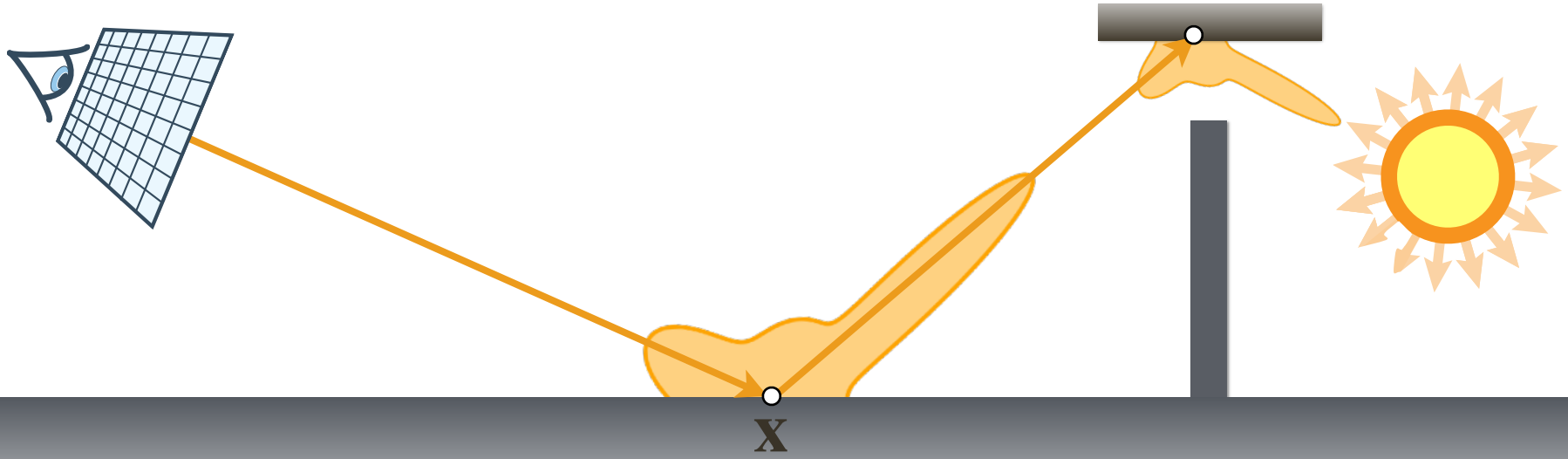
Path guiding



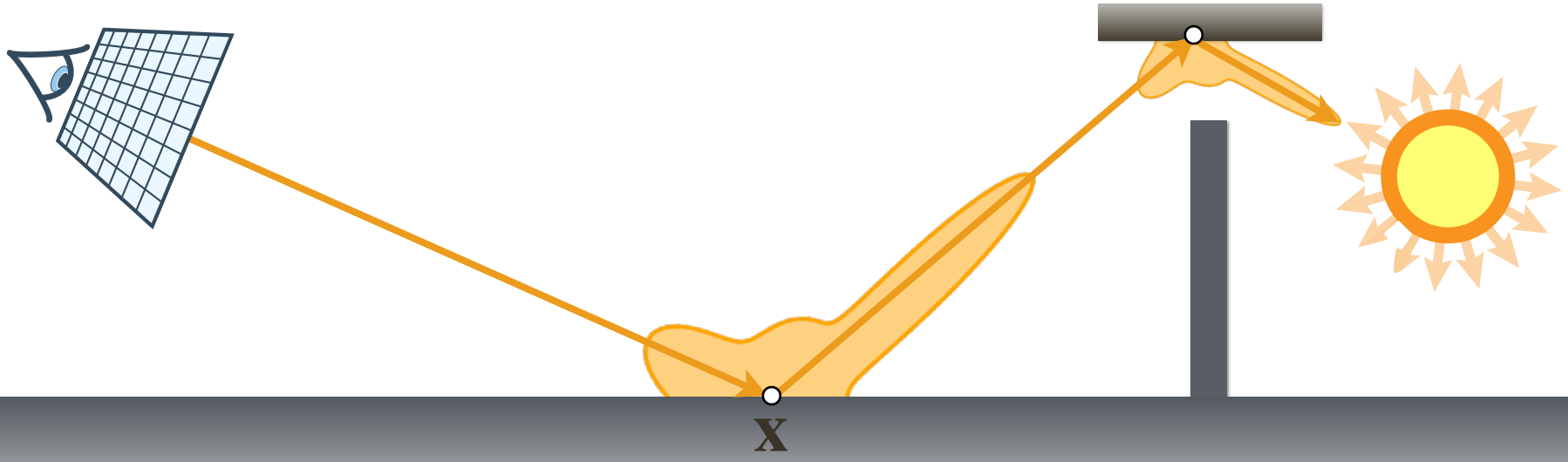
Path guiding



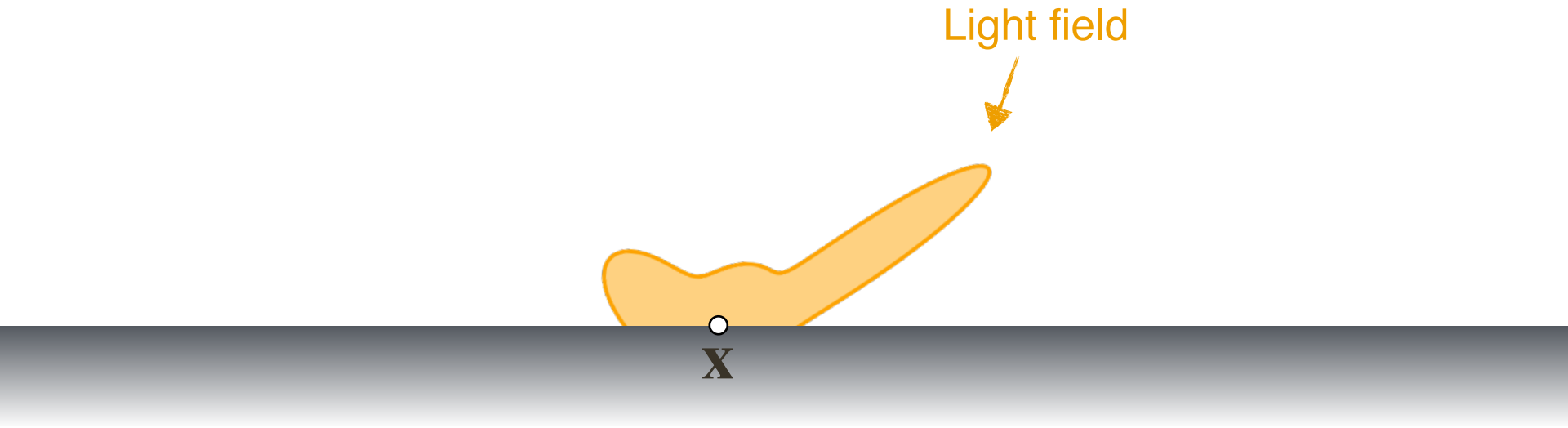
Path guiding



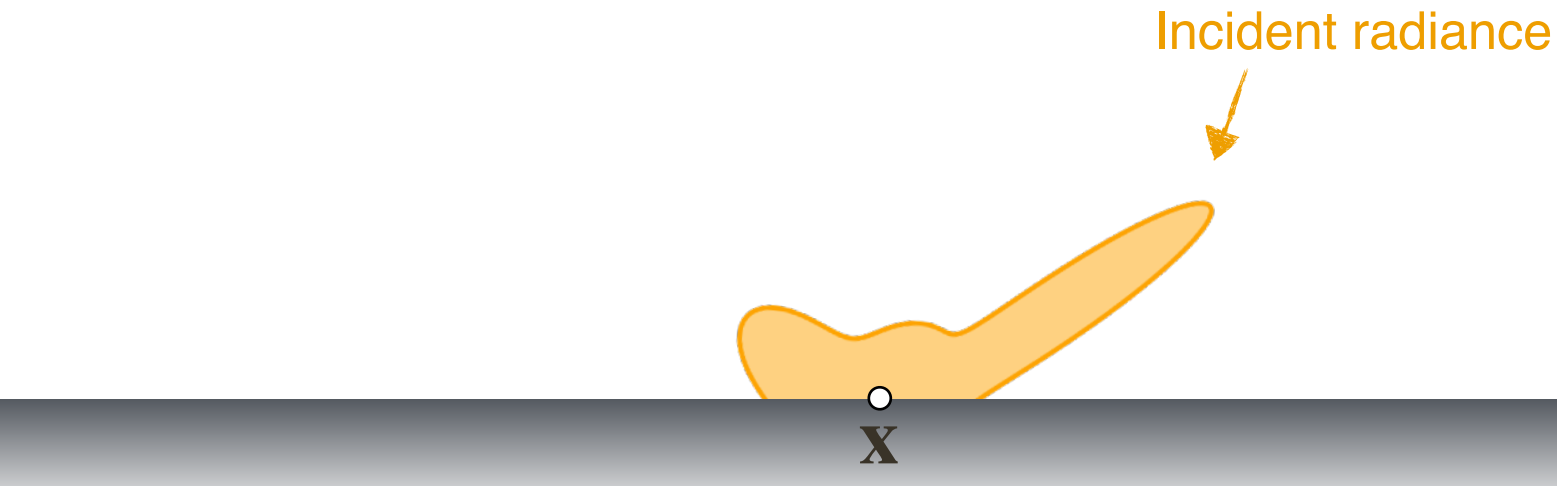
Path guiding



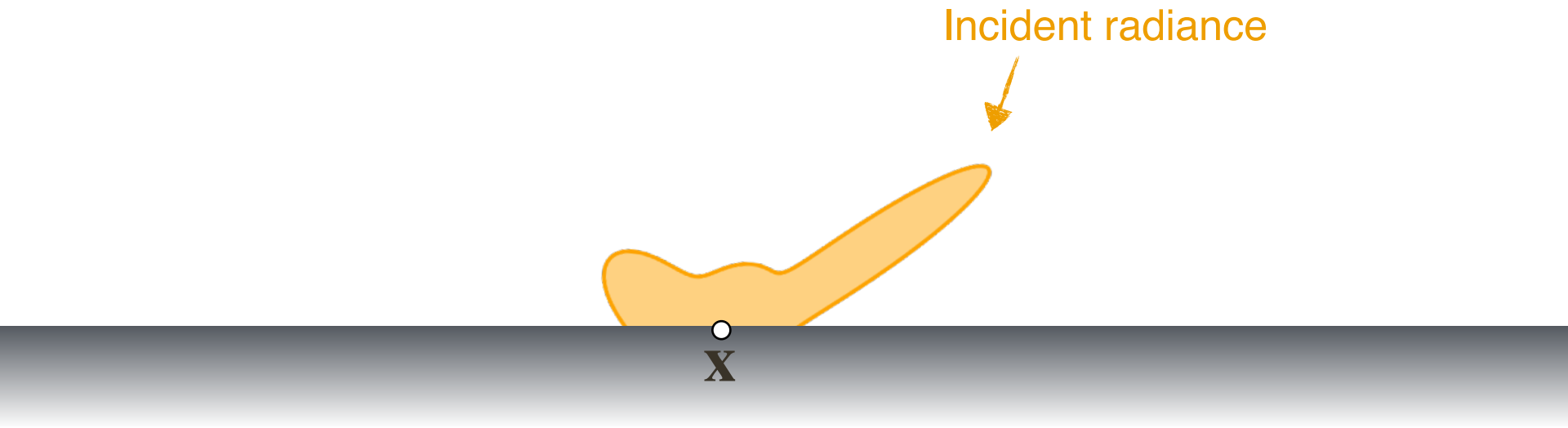
The goal is to learn the 5D = 3D + 2D light field



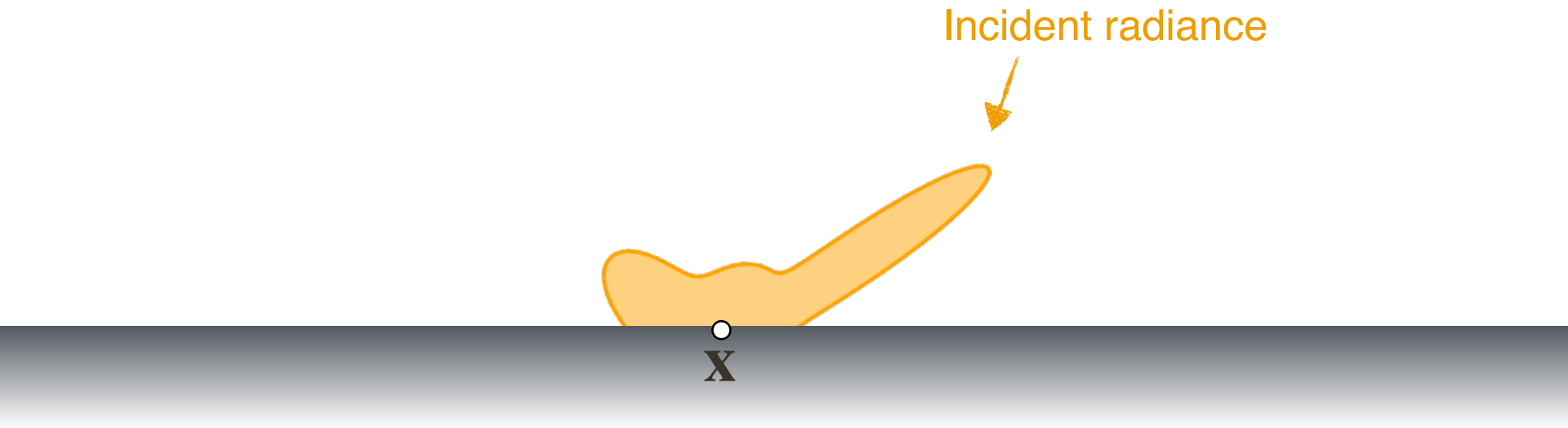
The goal is to learn the 5D = 3D + 2D light field



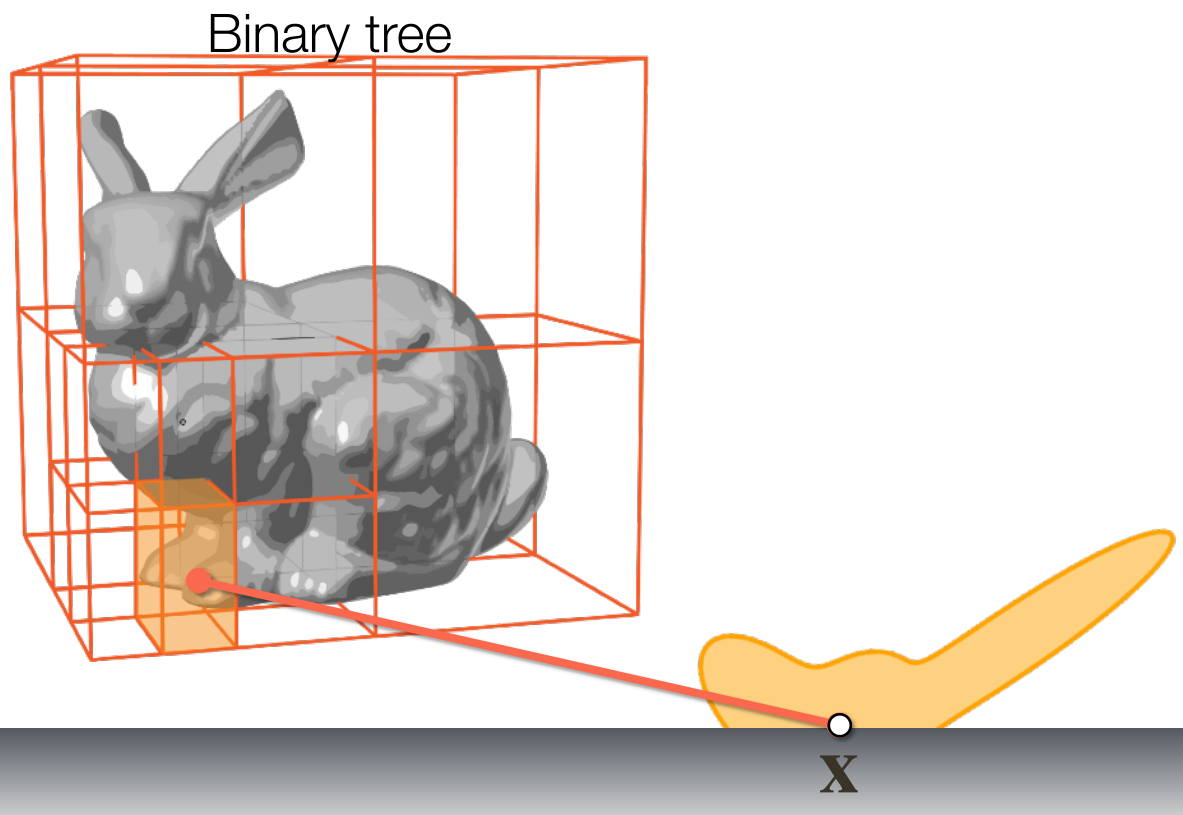
The goal is to learn the 5D = 3D + 2D light field



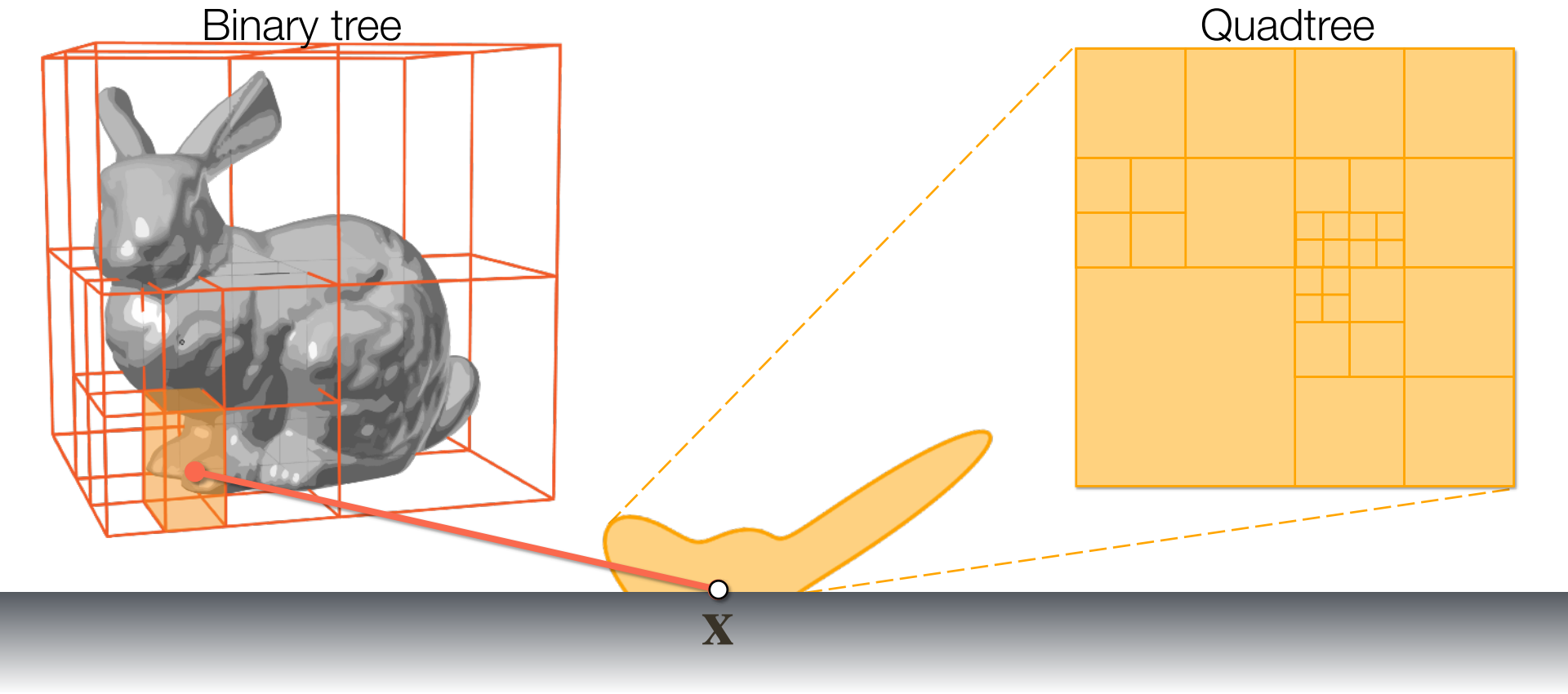
The goal is to learn the 5D = 3D + 2D light field



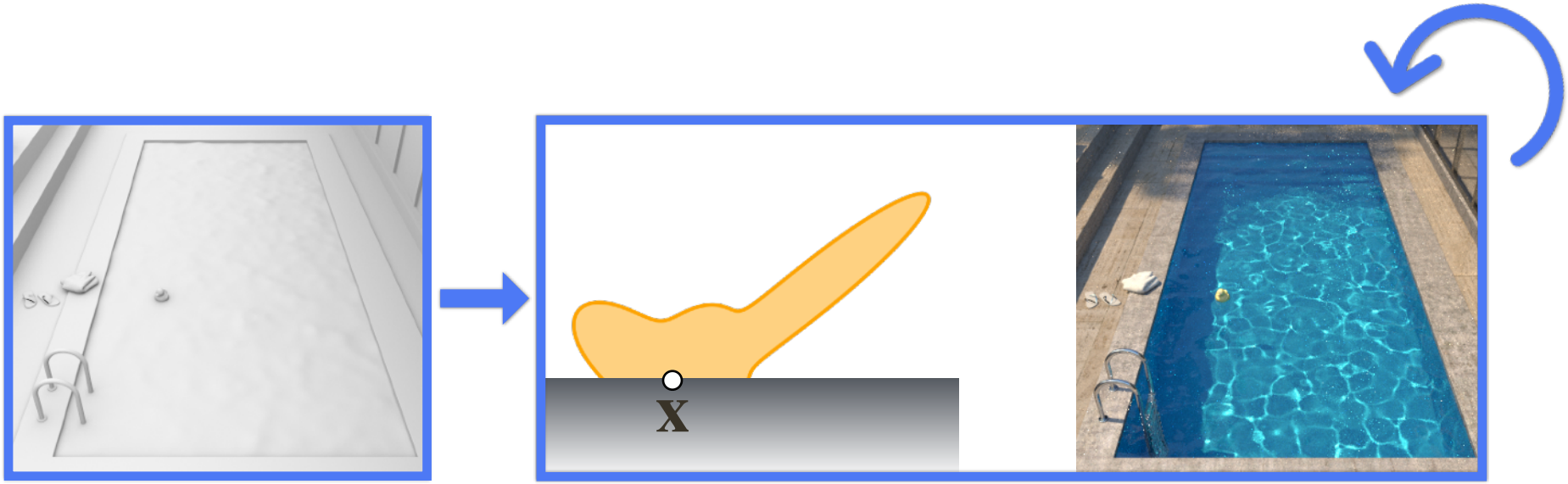
Representing the light field in a 3D + 2D tree



Representing the light field in a 3D + 2D tree



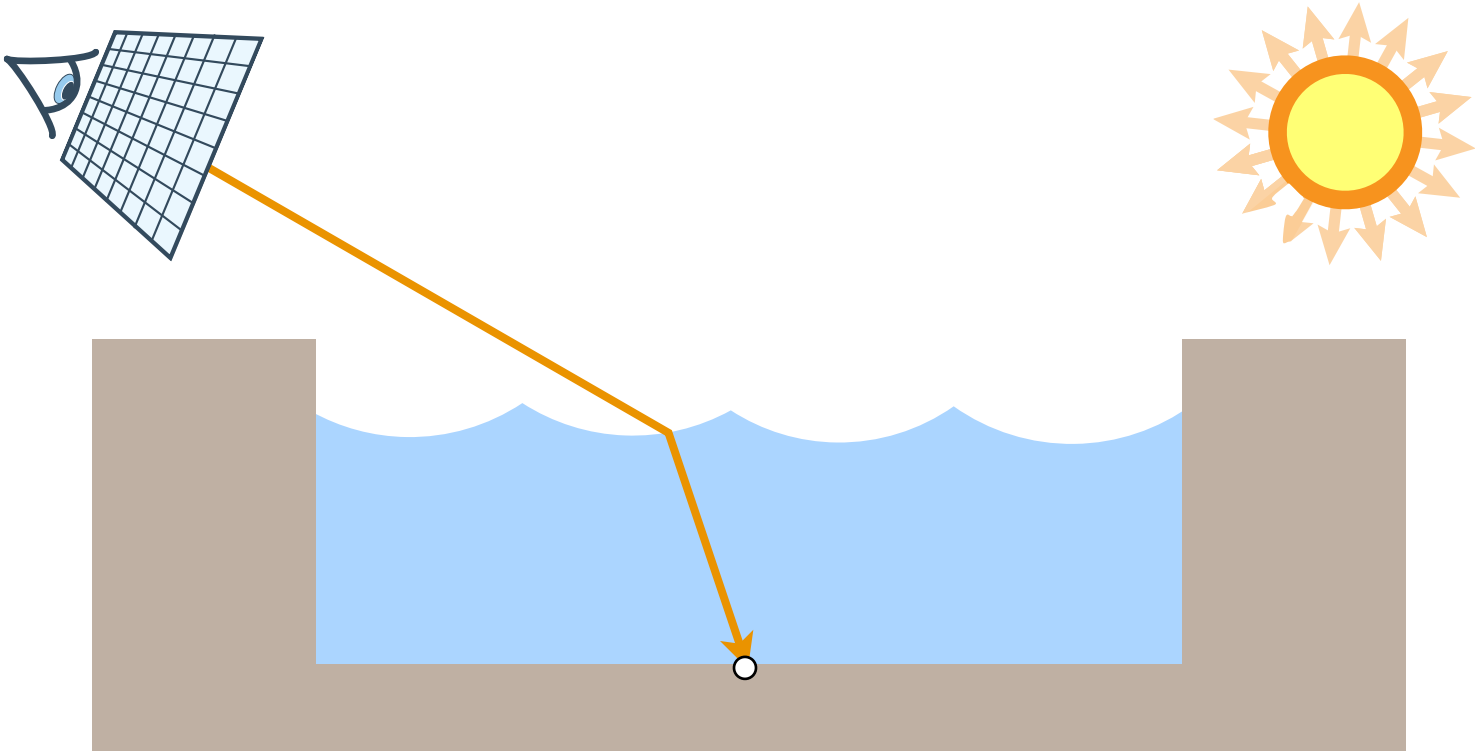
Iterative learning during rendering



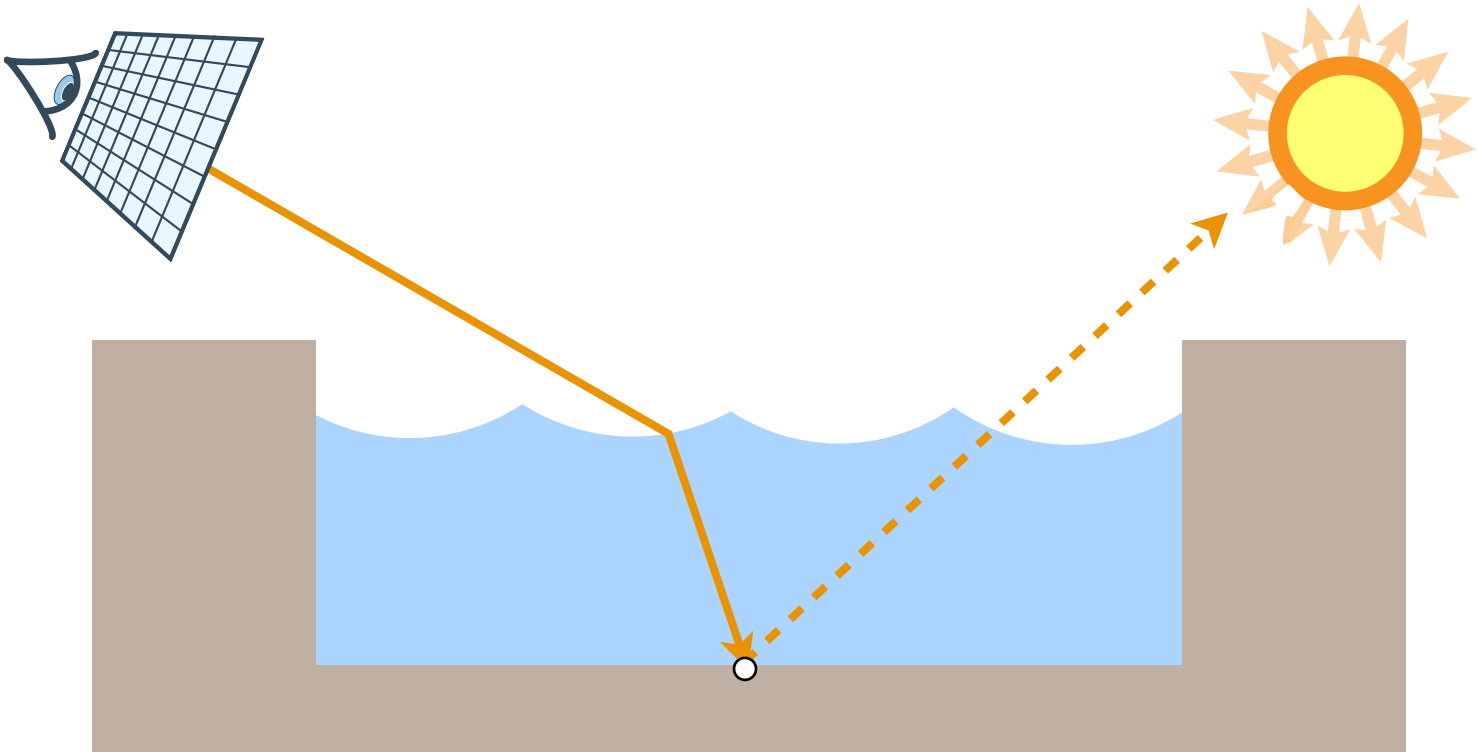
Iterative learning during rendering

Iterative learning during rendering

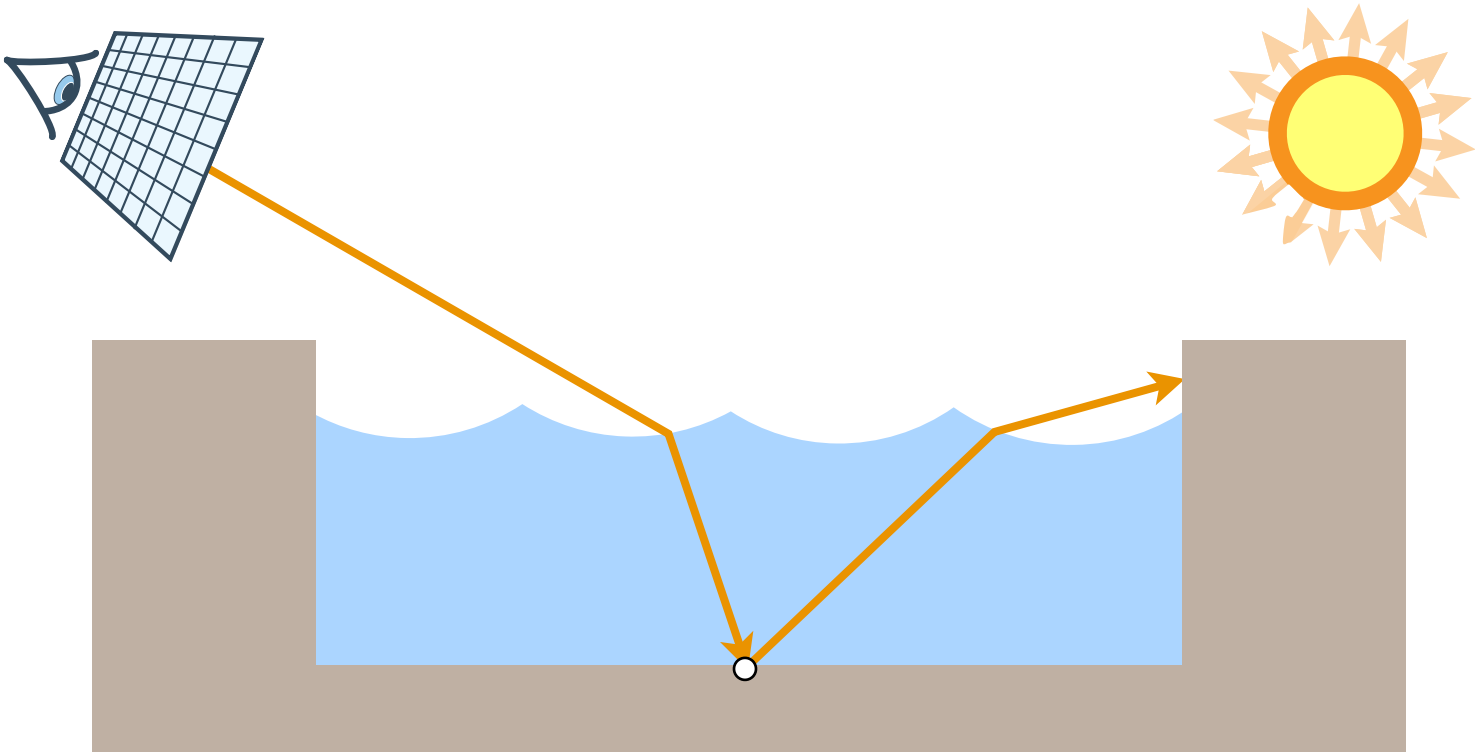
Iterative learning during rendering



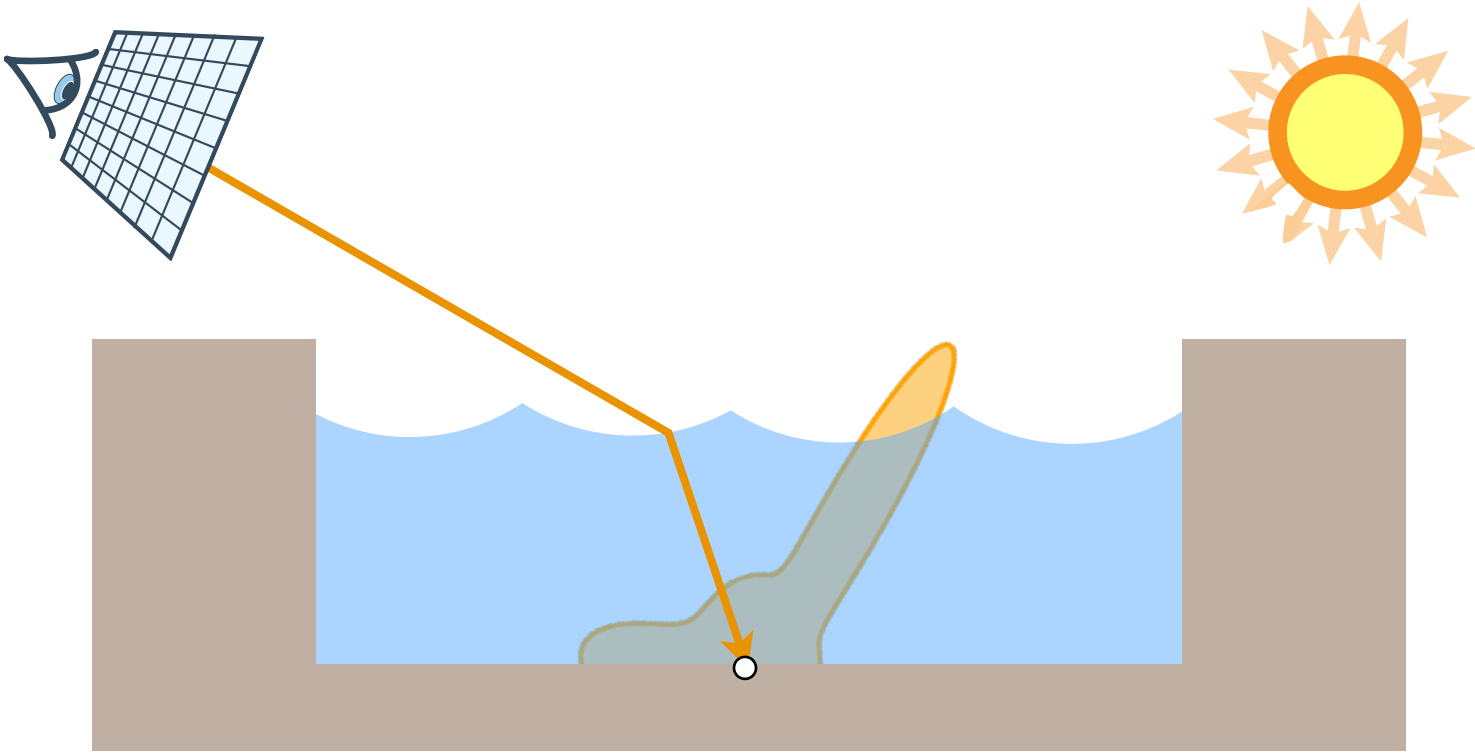
Iterative learning during rendering



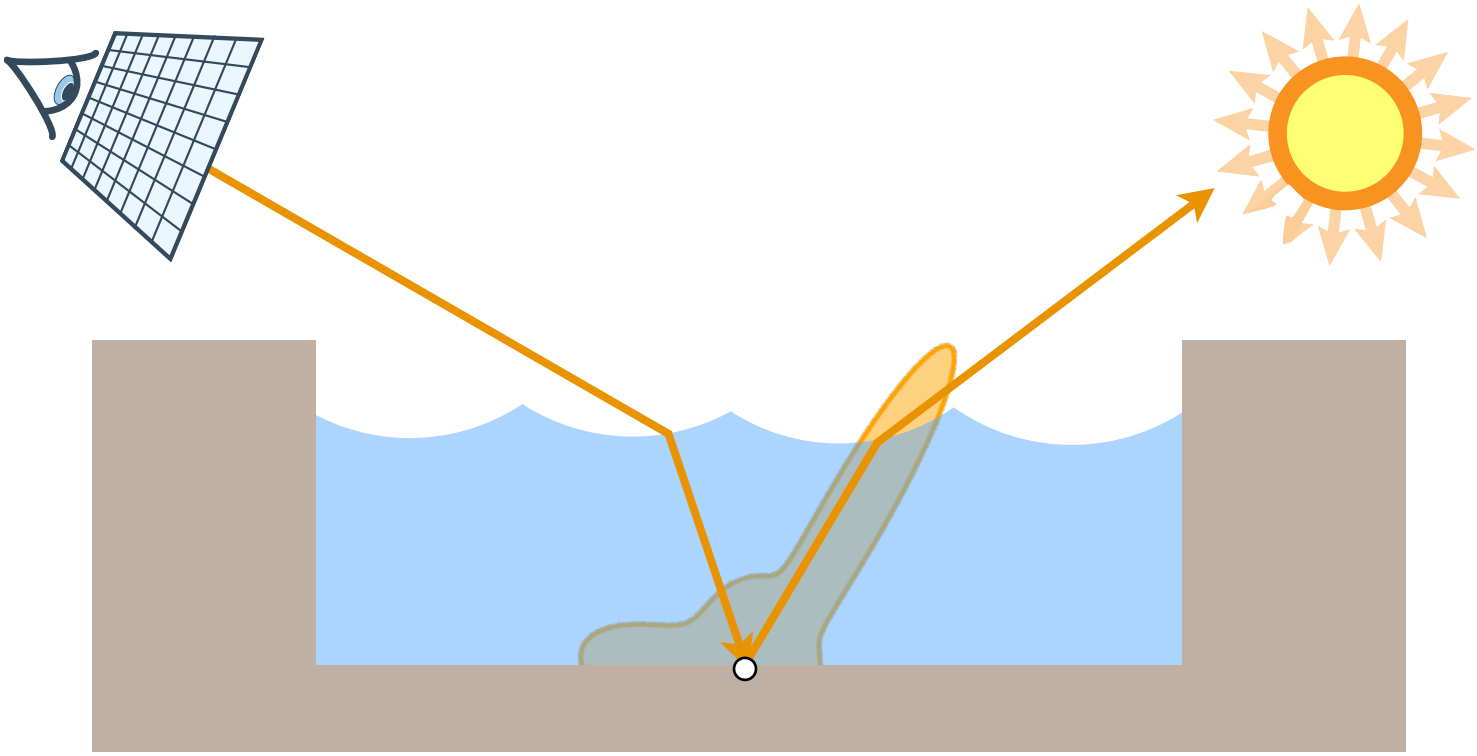
Iterative learning during rendering



Iterative learning during rendering

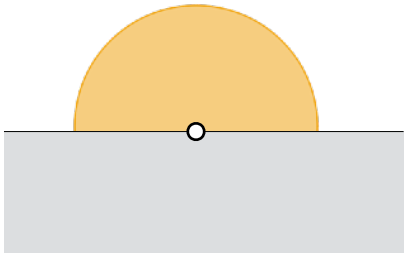


Iterative learning during rendering



Iterative learning during rendering

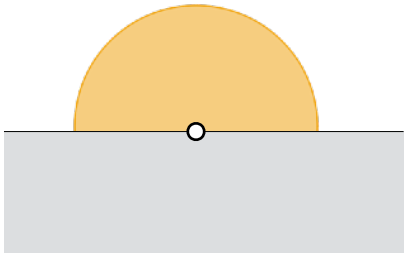
Illustration of data structure



Iteration 1

Iterative learning during rendering

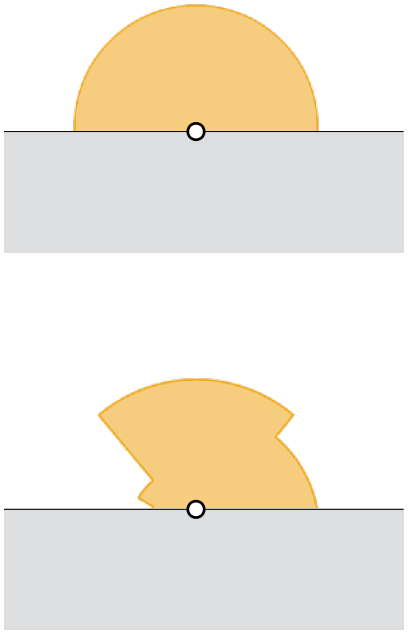
Illustration of data structure



Iteration 1

Iterative learning during rendering

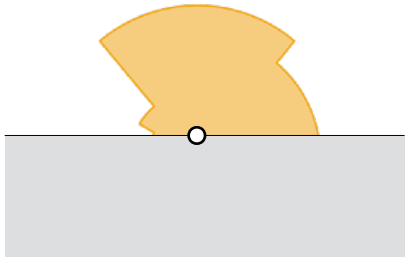
Illustration of data structure



Iteration 1

Iterative learning during rendering

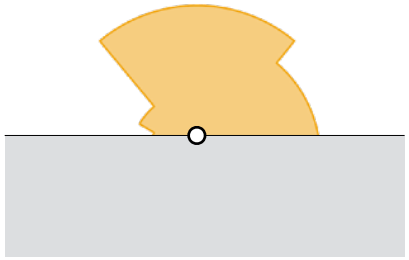
Illustration of data structure



Iteration 2

Iterative learning during rendering

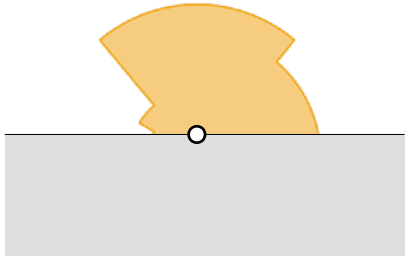
Illustration of data structure



Iteration 2

Iterative learning during rendering

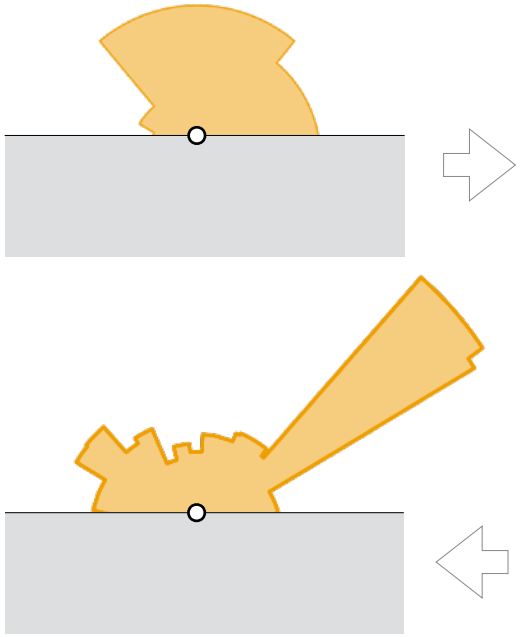
Illustration of data structure



Iteration 2

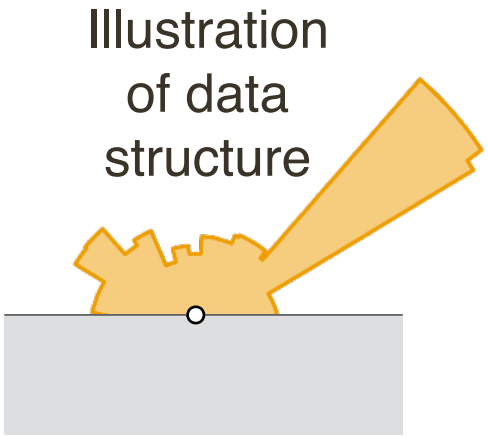
Iterative learning during rendering

Illustration of data structure



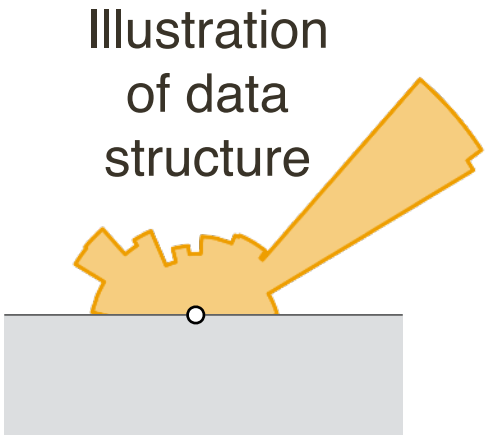
Iteration 2

Iterative learning during rendering



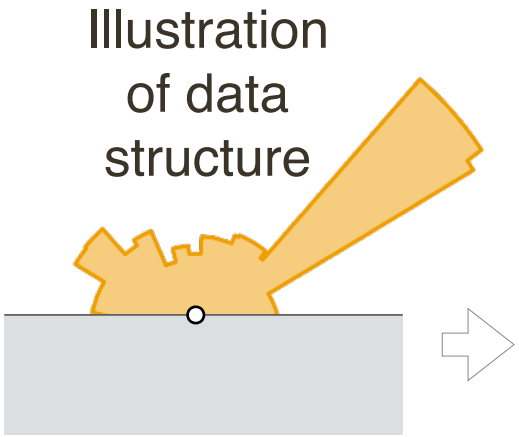
Iteration 3

Iterative learning during rendering

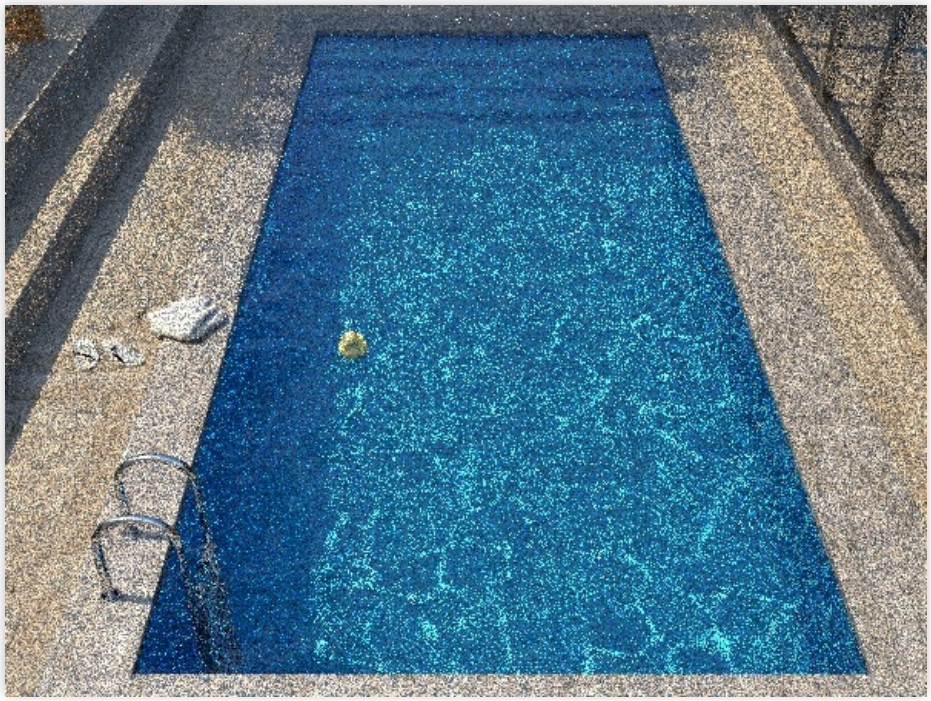


Iteration 3

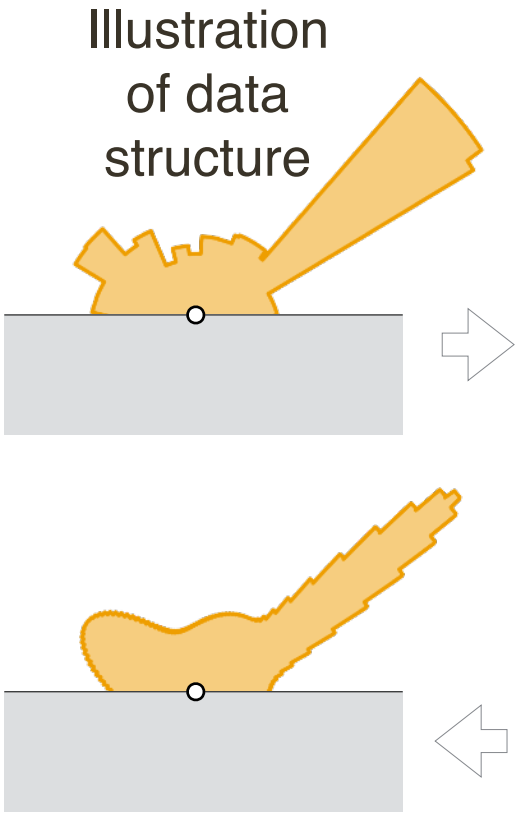
Iterative learning during rendering



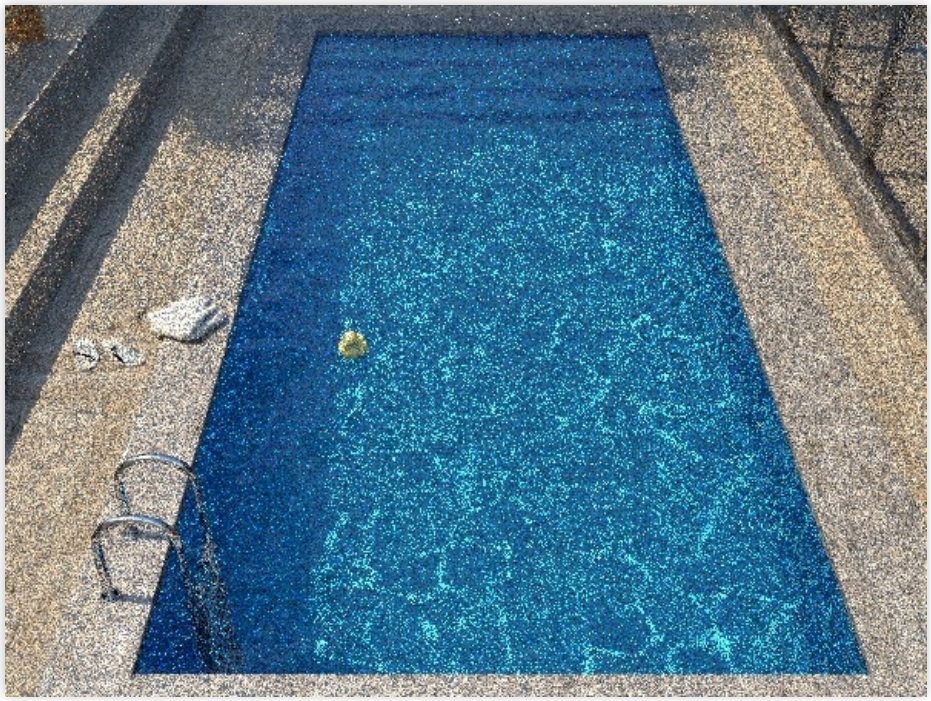
Iteration 3



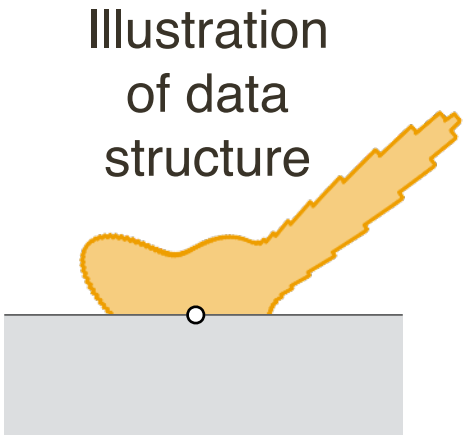
Iterative learning during rendering



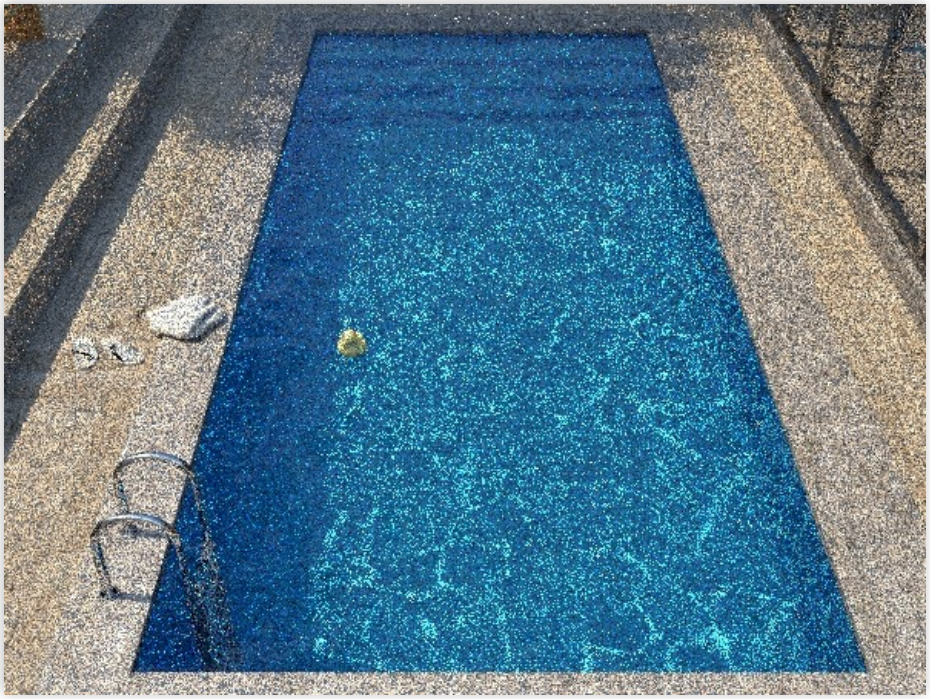
Iteration 3



Iterative learning during rendering

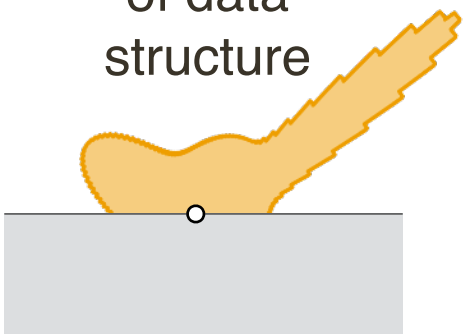


Iteration 4



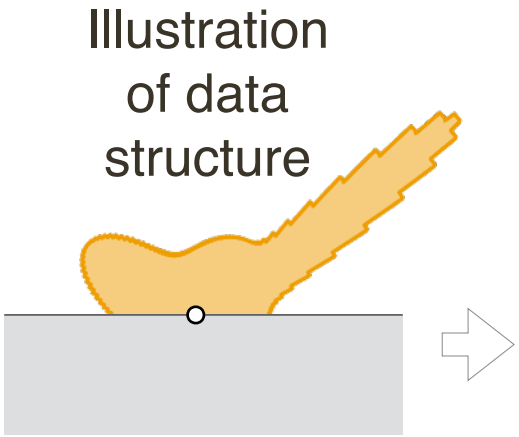
Iterative learning during rendering

Illustration of data structure



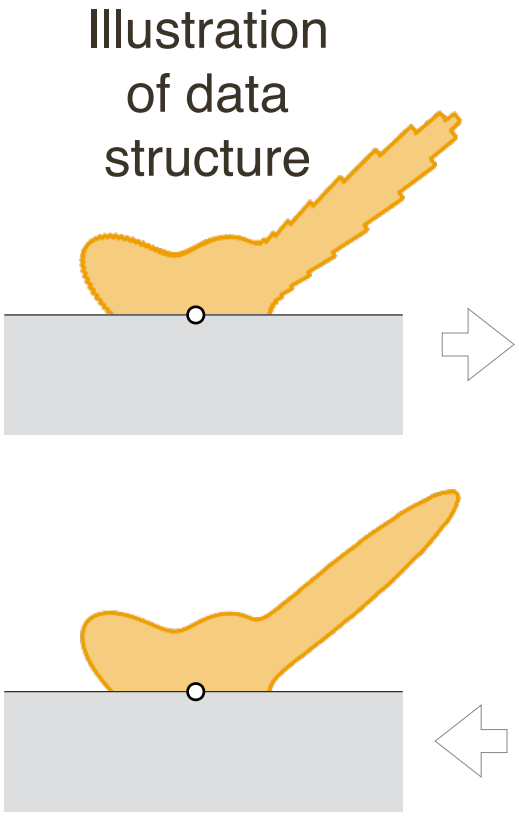
Iteration 4

Iterative learning during rendering

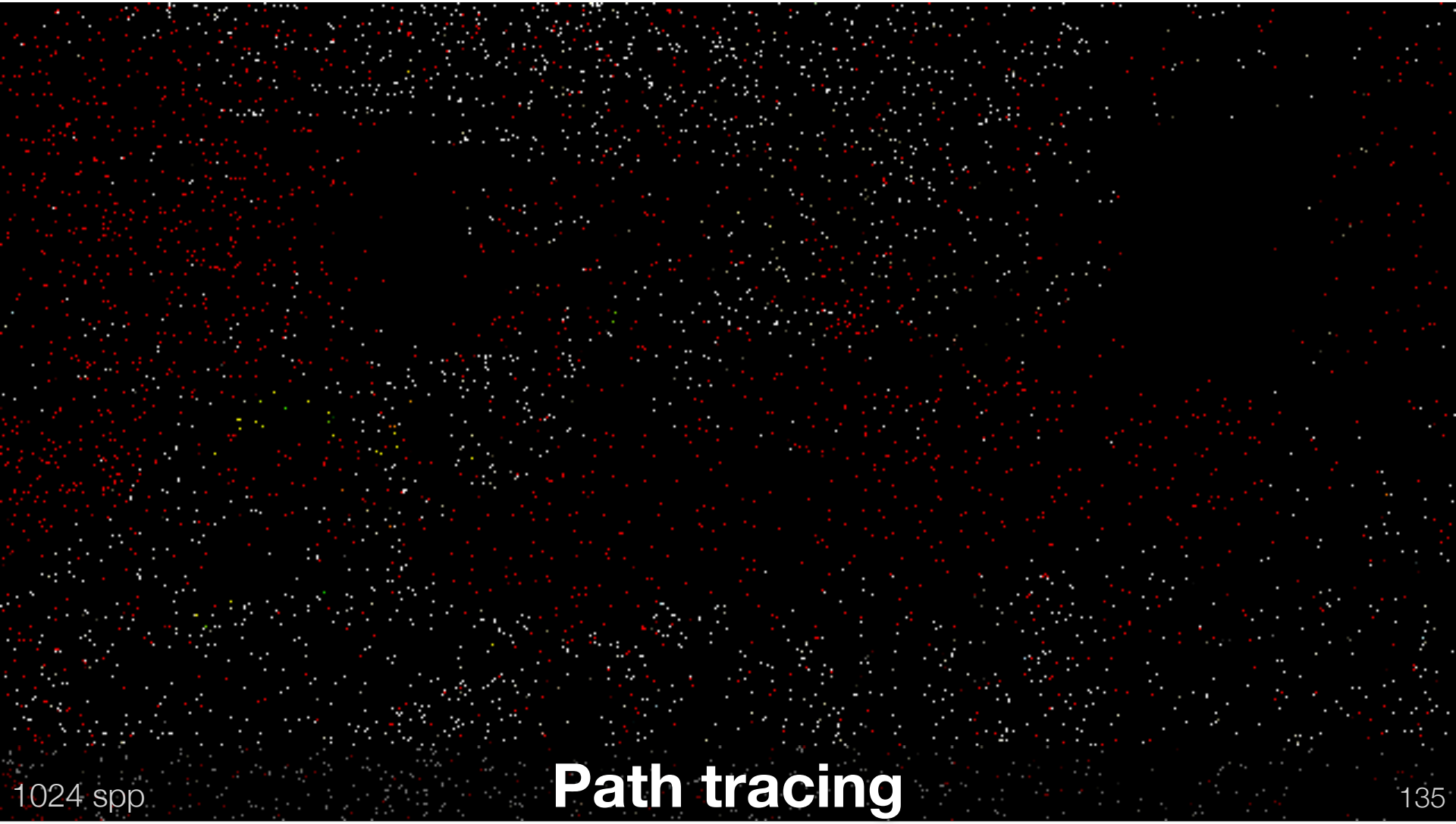


Iteration 4

Iterative learning during rendering



Iteration 4


1024 spp
Path tracing

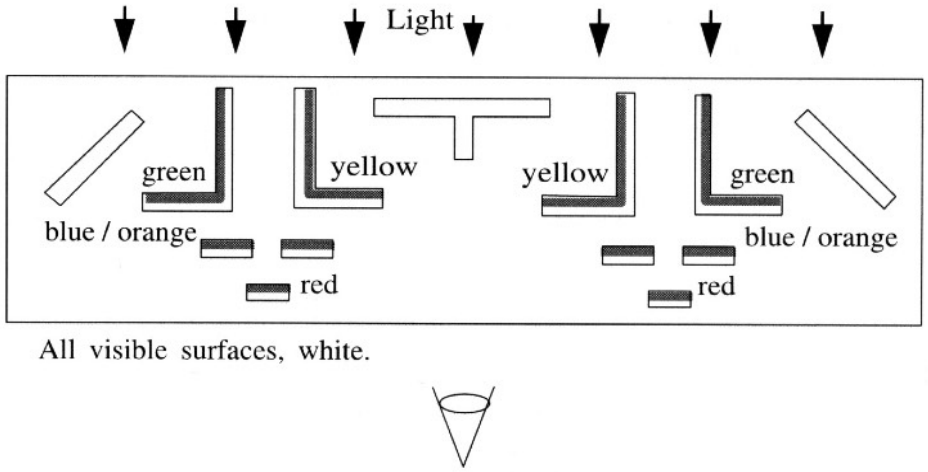

1024 spp
"Practical path guiding" with improvements

Finite-Element Methods

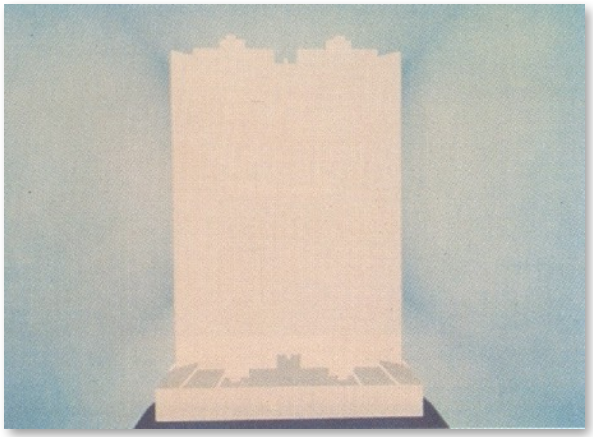
Radiosity

- Taking into account all inter-reflections

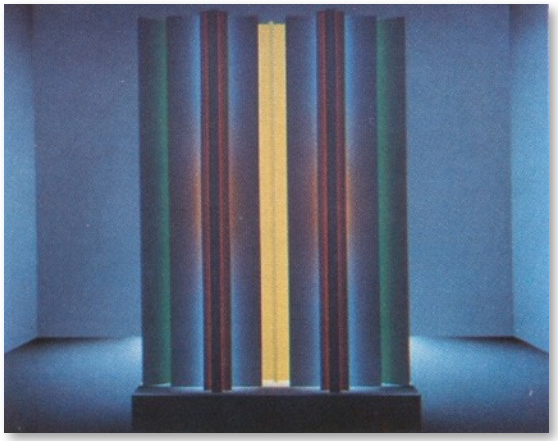
Sculpture by John Ferren



Photo



Ray-tracing



Radiosité

Radiosity methods [1984]

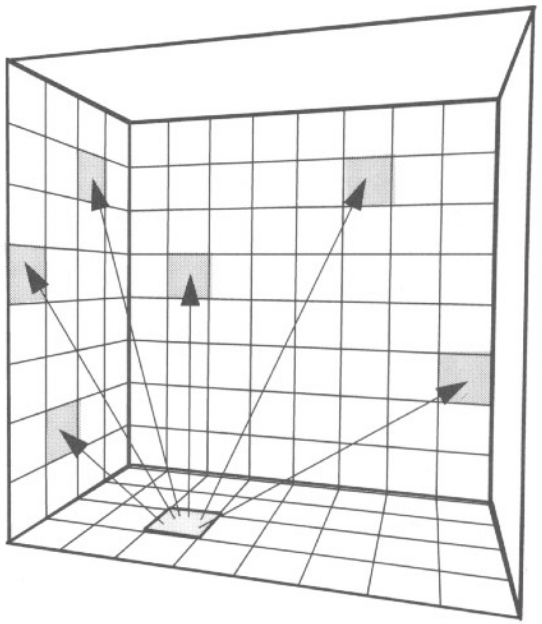
- Hypothesis: diffuse materials
- Radiance, BRDF… are independent from the direction

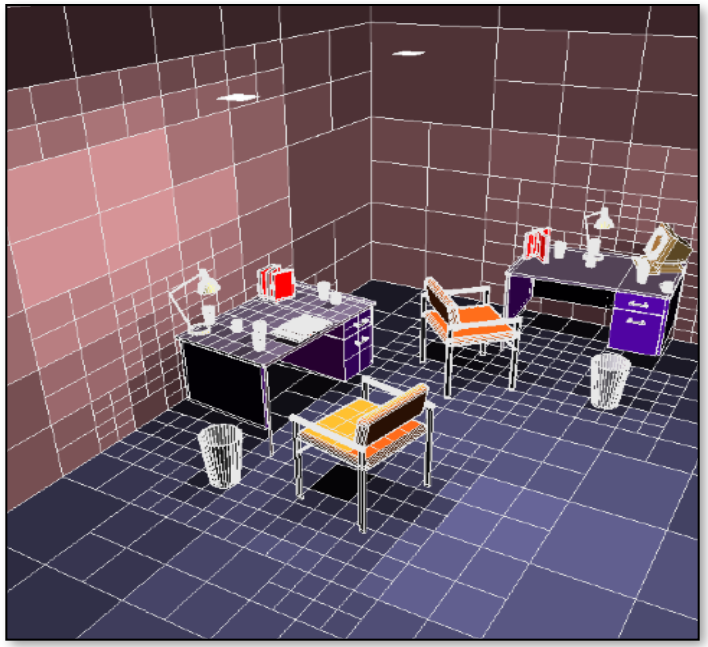
⇨ Simplifies the rendering equation

- Radiosity method:
  - Discretize this equation in object space
    - (viewpoint independent)
  - Solve the discretized equation
  - Render the scene using the illumination

Radiosity Equation

- Discretize scene into patches





Simplification and discretization

$$L(x,d) = E(x,d) + \int \rho(x,d,d') \, v(x,x') \, G(x,x') \, dA$$

- Simplified:

$$B(x) = E(x) + \rho_x \int B(\mathbf{x'}) \, v(\mathbf{x},\mathbf{x'}) \, G(\mathbf{x},\mathbf{x'}) \, dA$$

Form factor

- Discrete version:

$$B_i = E_i + \rho_i \sum_j F_{ji} \, B_j \, A_j / A_i$$

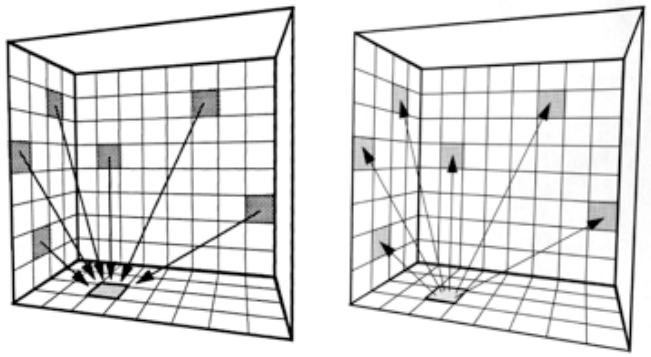
  - B_i, B_j: radiosity for patches i et j (in W/m2)
  - E_i: emission for patch i
  - F_{ji} form factor, characterizes proportion of energy leaving patch j arriving on i
  - A_i et A_j : areas for patches i and j

Matrix representation

- Grouping together all elements:

$$\begin{pmatrix} B_0 \\ \\ B_n \end{pmatrix} = \begin{pmatrix} E_0 \\ \\ E_n \end{pmatrix} + \begin{pmatrix} \rho_i F_{ji} \end{pmatrix} \begin{pmatrix} B_0 \\ \\ B_n \end{pmatrix} \quad \Leftrightarrow \quad B = E + MB$$

- Matrix equation, to solve iteratively
  - Relaxation methods (gathering / shooting)



Form factor

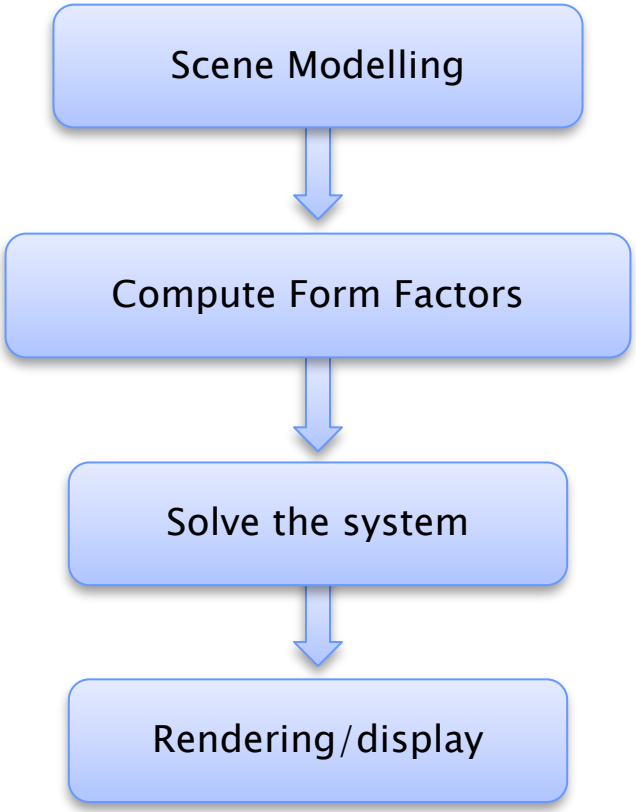
- Form factor F_{ij} from a patch A_i towards a patch A_j :

$$F_{ij} = \int_{A_i} \int_{A_j} v(x,x') \frac{\cos(\theta)\cos(\theta')}{\pi r^2} dx dx'$$

- Problem: computing this integral (4D). No analytical solution
  - ⇨Approximated solutions: projection on a hemisphere or a hemicube.

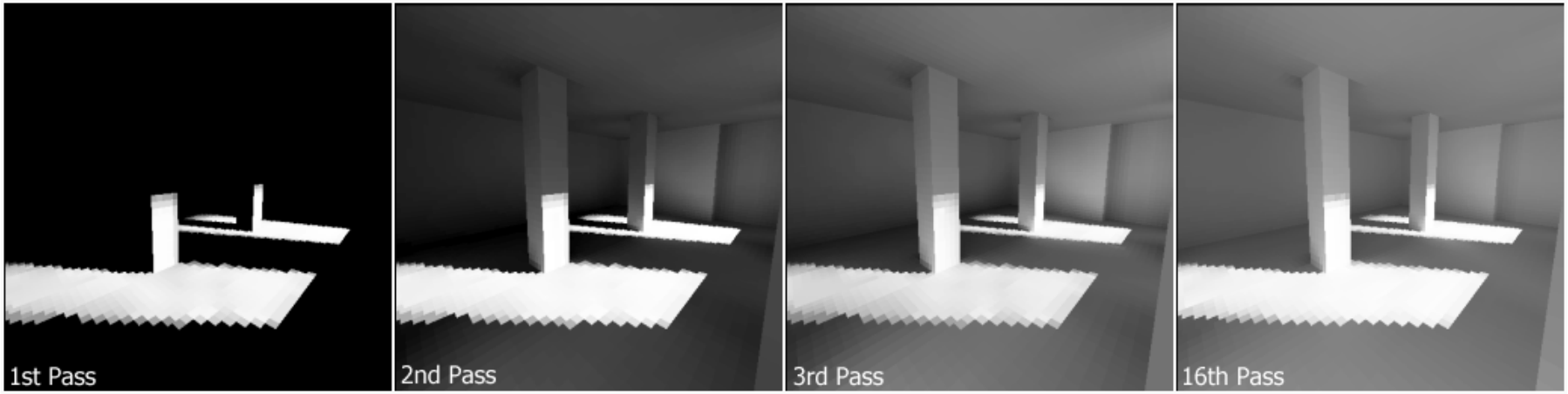
Solving radiosity

- Pipeline for computing global illumination :



Solving radiosity

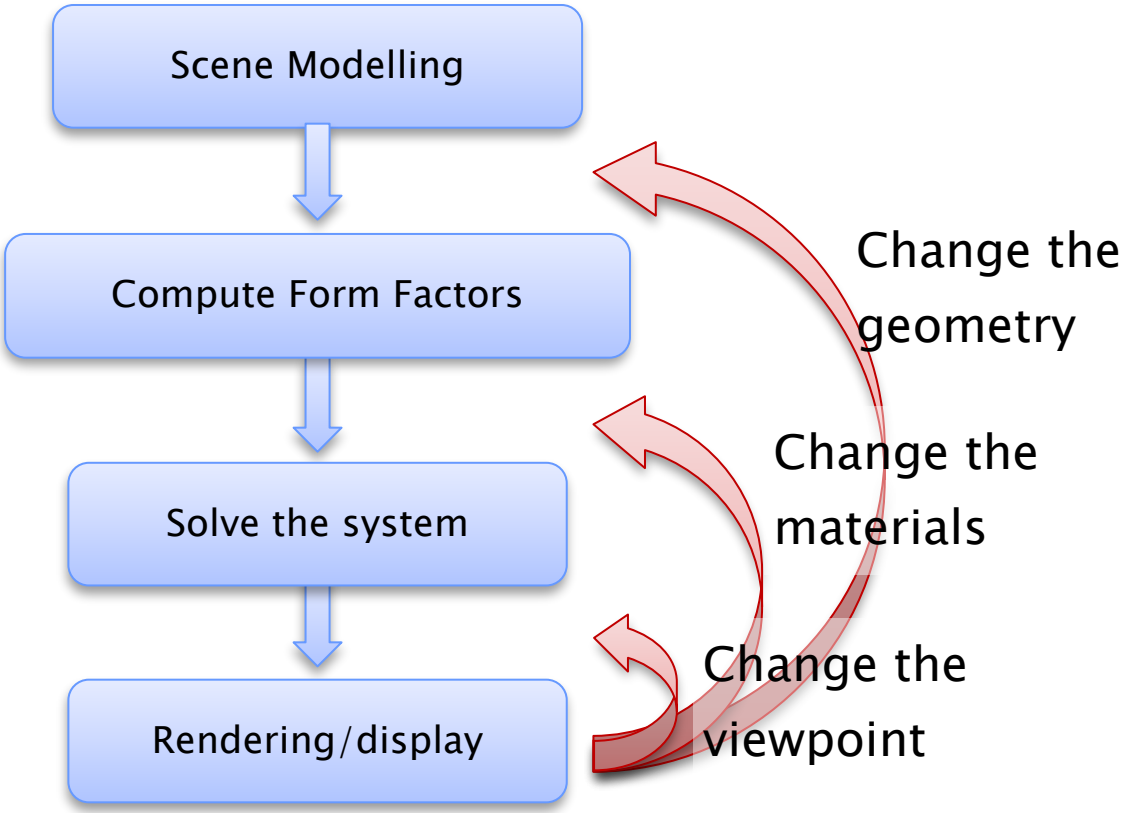
- Iterative solving:



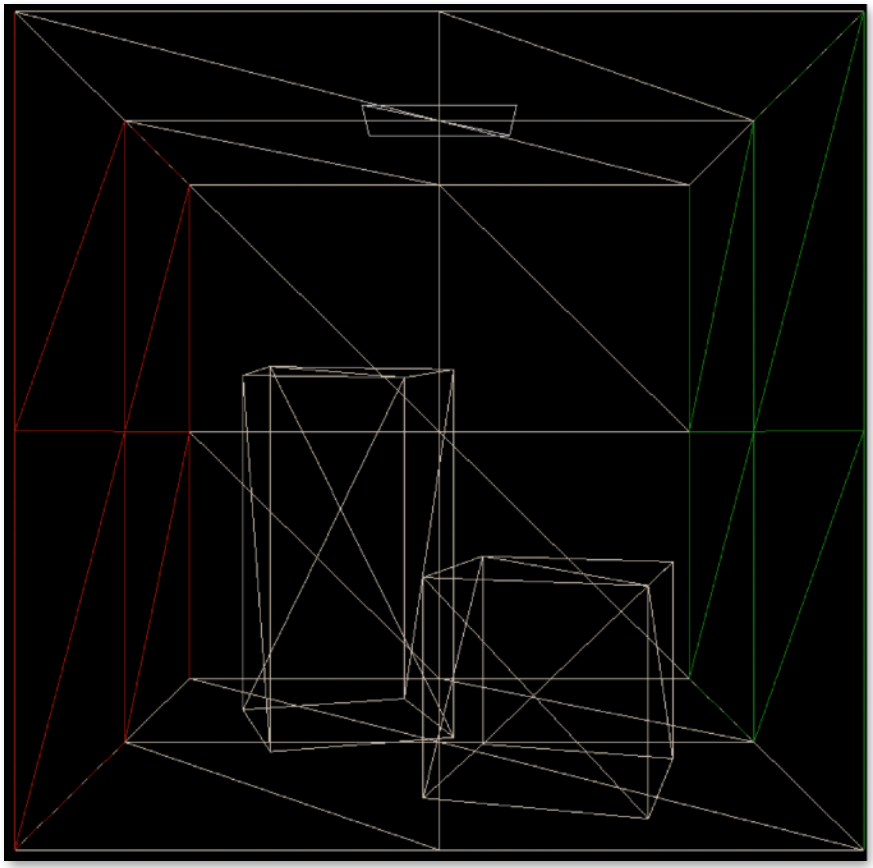
Question – 1 mn

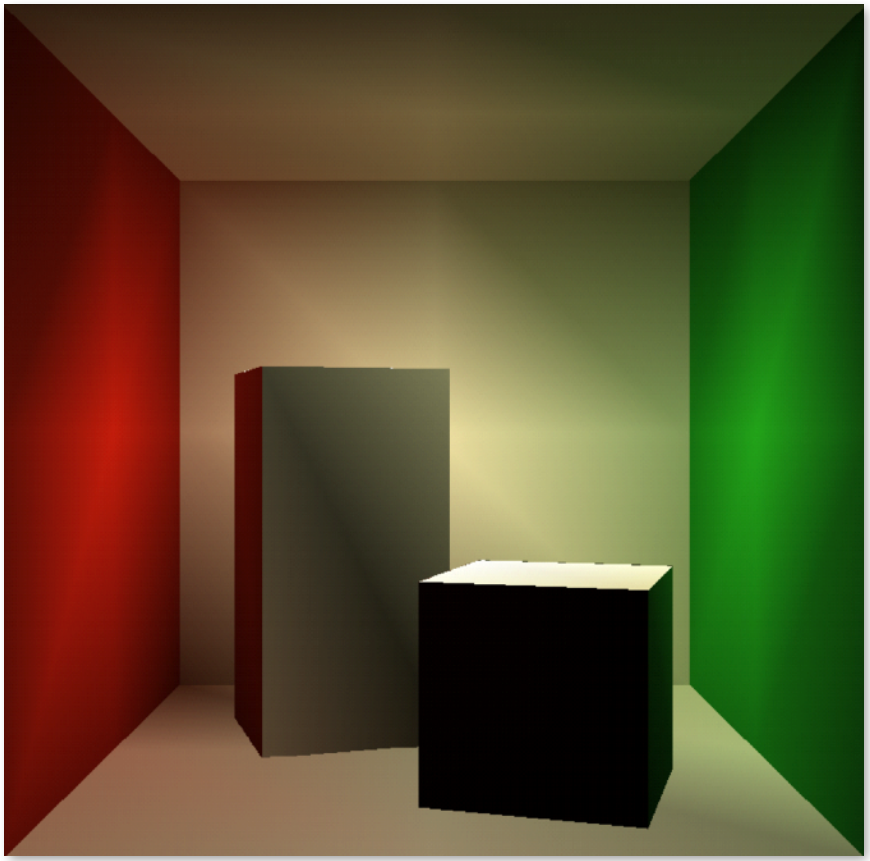
- What must be recomputed if something changes in the scene?
  - Geometry
  - Reflectance properties
  - Viewpoint

Solving radiosity

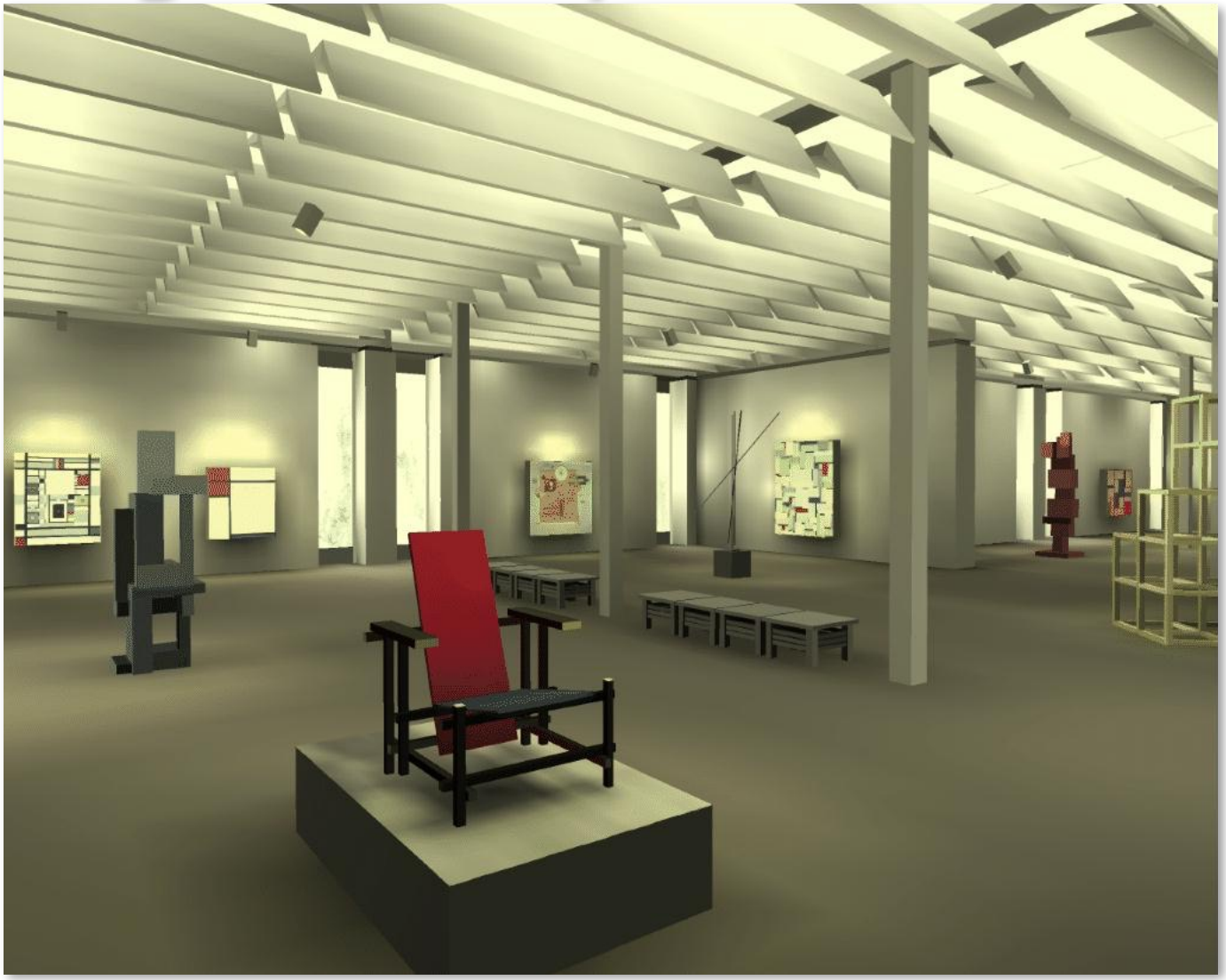


Solving radiosity





Solving radiosity



Museum simulation. Cornell University. 50,000 patches.

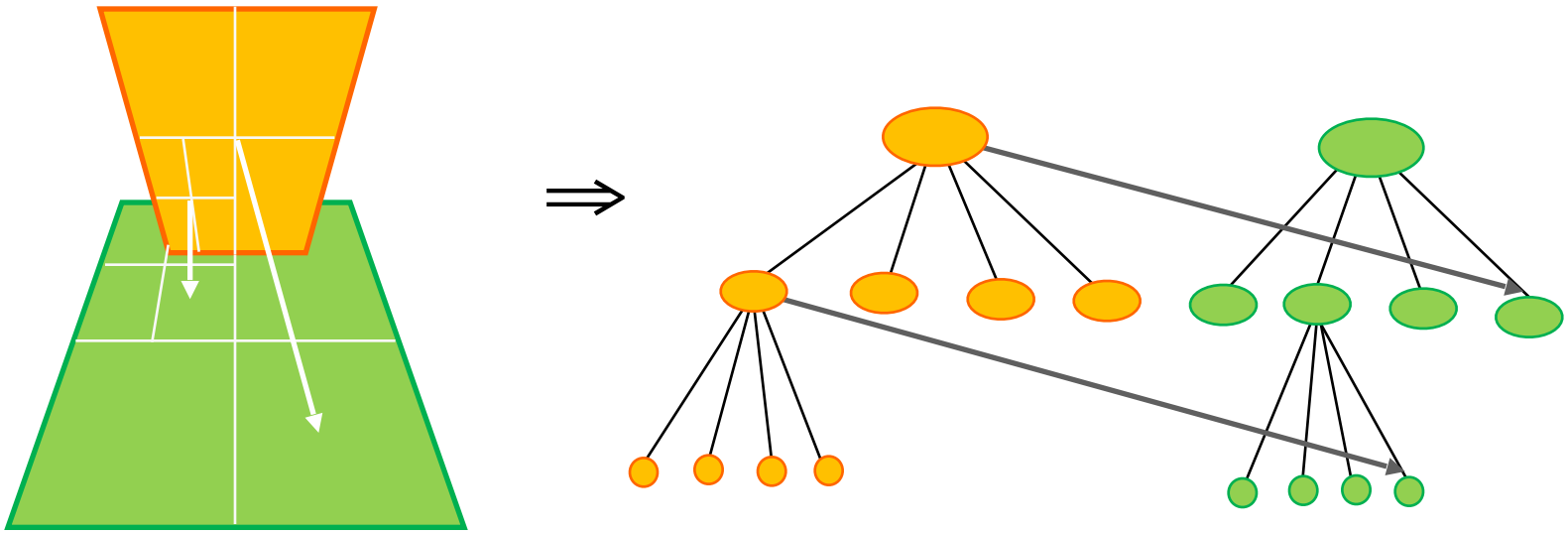
Radiosity: pros

- Computations are viewpoint independent
- Okay for complex scenes
- Solving light exchanges
  - Interactive rendering

Radiosity: cons

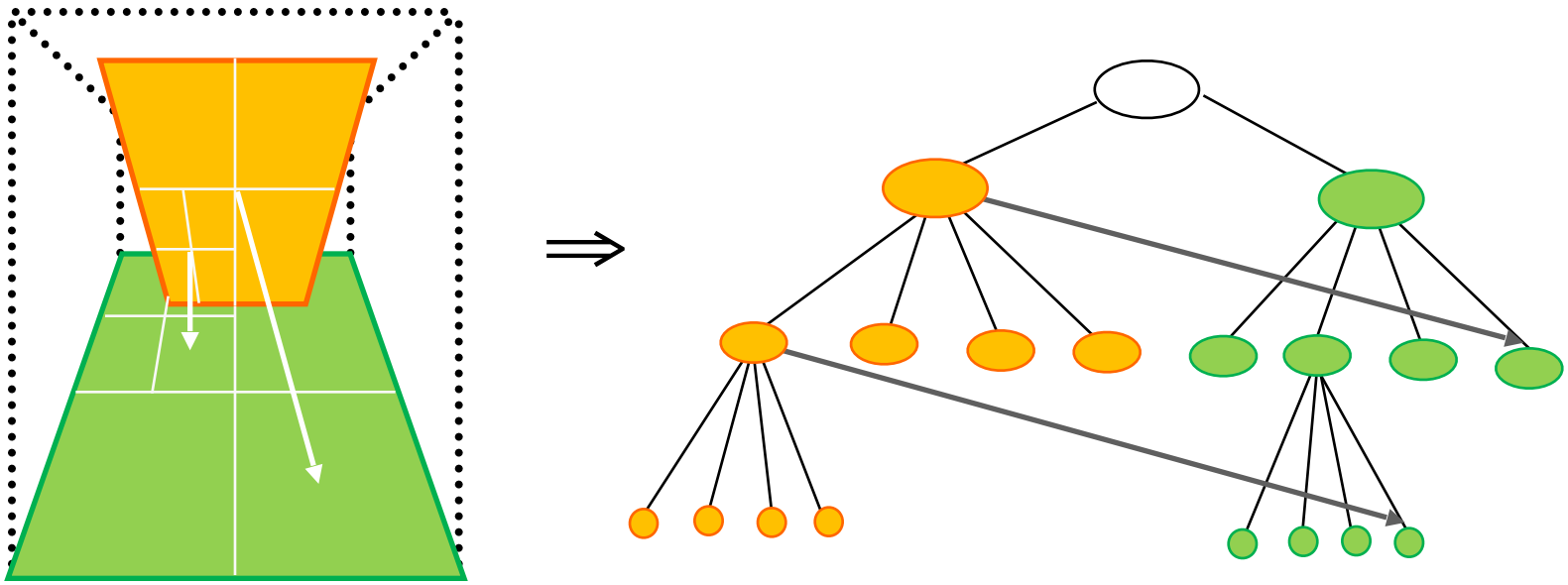
- Memory cost
- Only diffuse materials
  - « final gather » using Ray-Tracing
- Meshing
  - Discontinuity mesh
- Long pre-computations
  - Possible speed-up: hierarchical radiosity

Hierarchical radiosity [Hanrahan91]



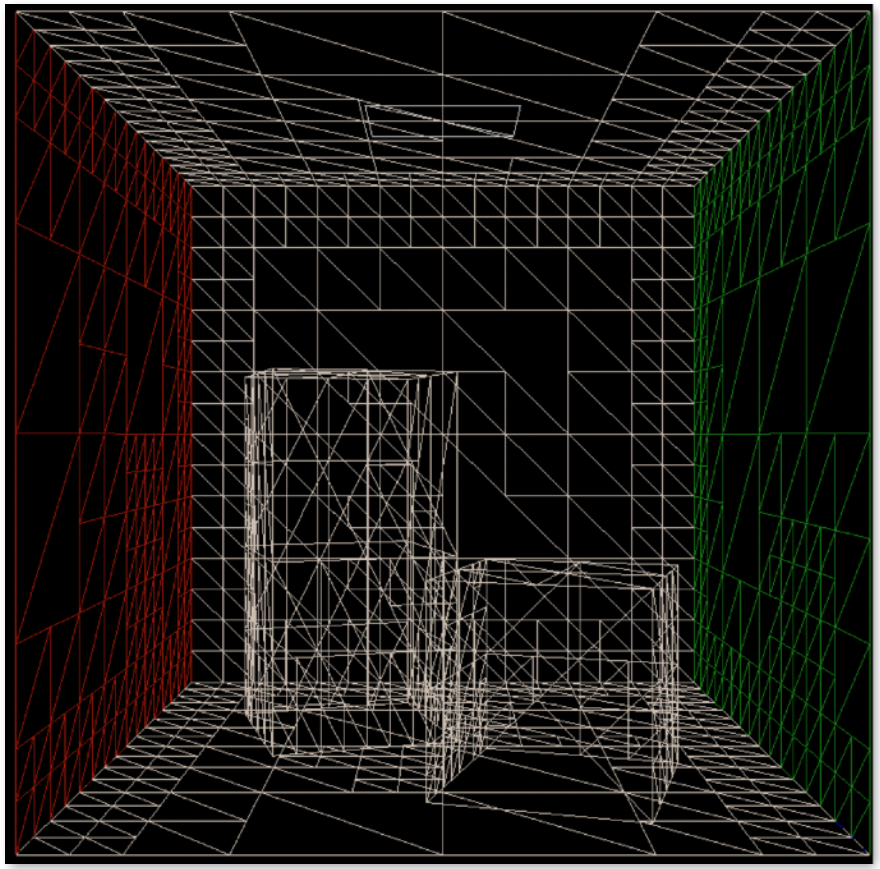
- Computation at different hierarchical levels
- Push-pull

Hierarchical radiosity [Hanrahan91]

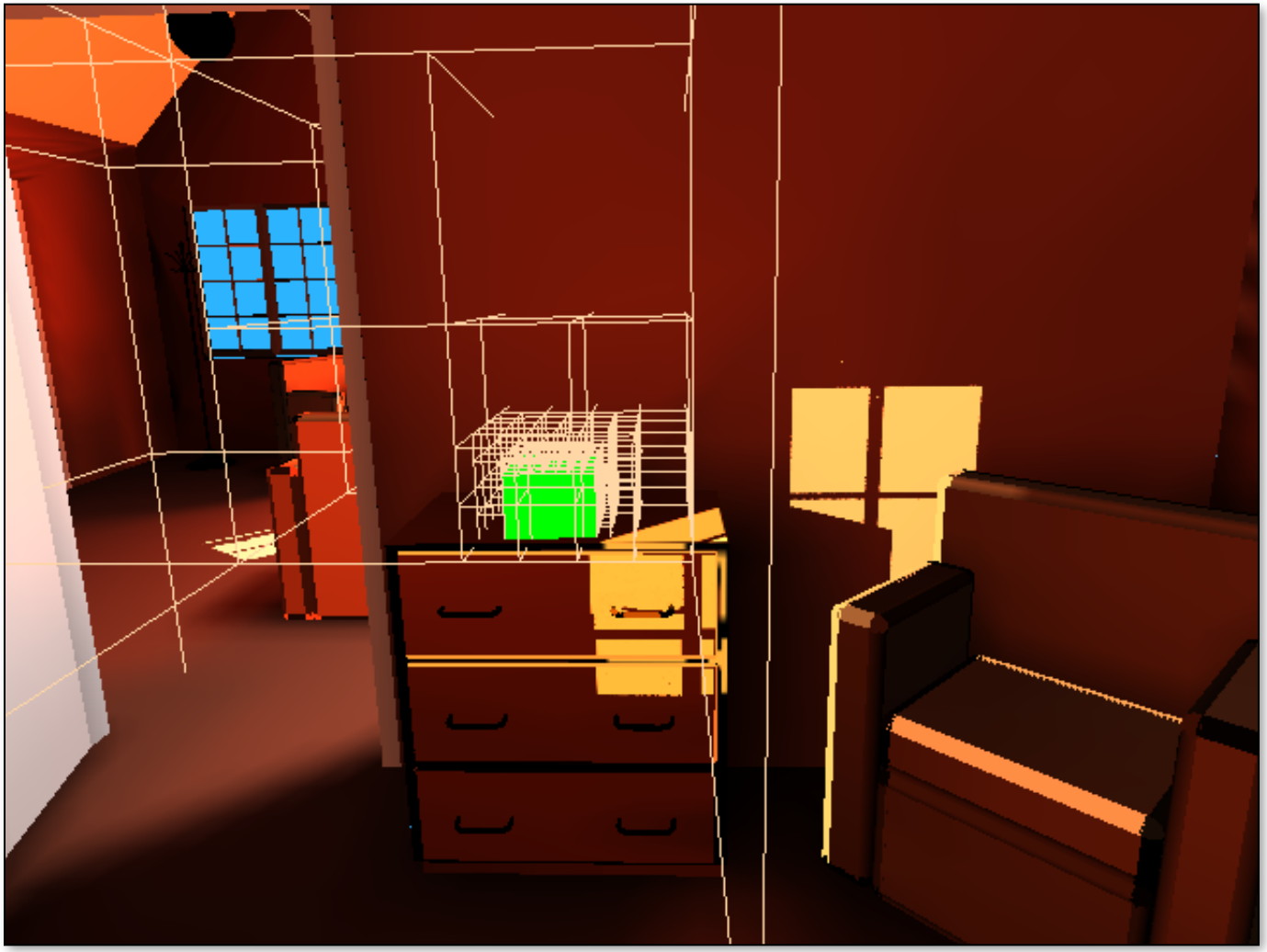


- Computation at different hierarchical levels
- Push-pull

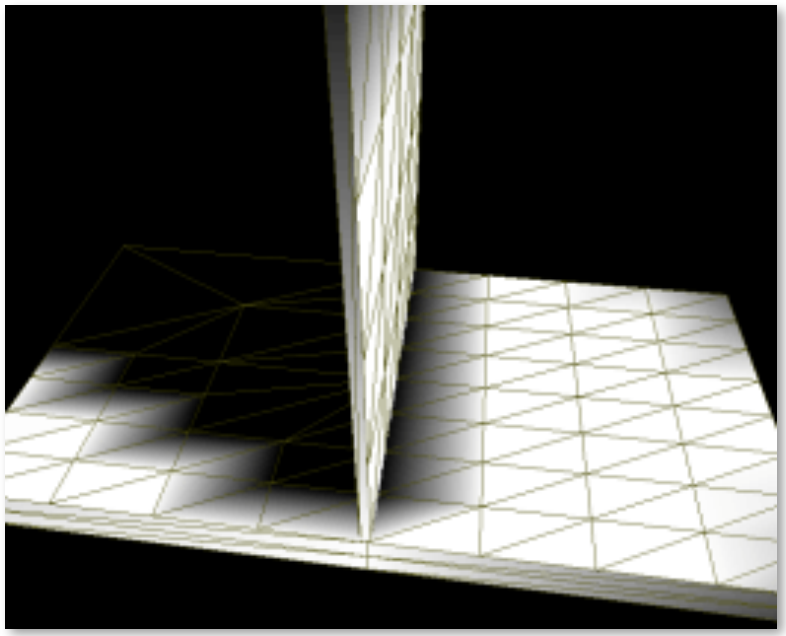
Hierarchical radiosity



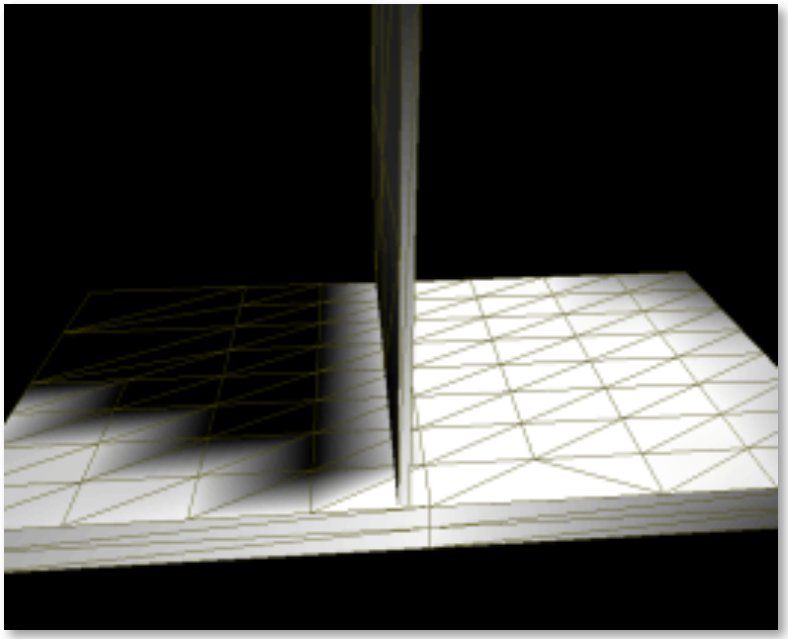
Hierarchical radiosity



Mesh quality

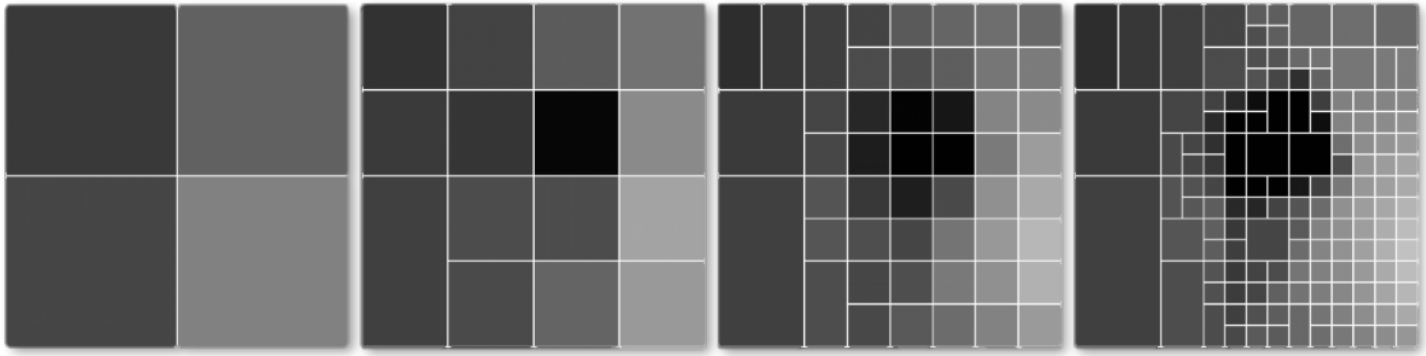


Shadow leak

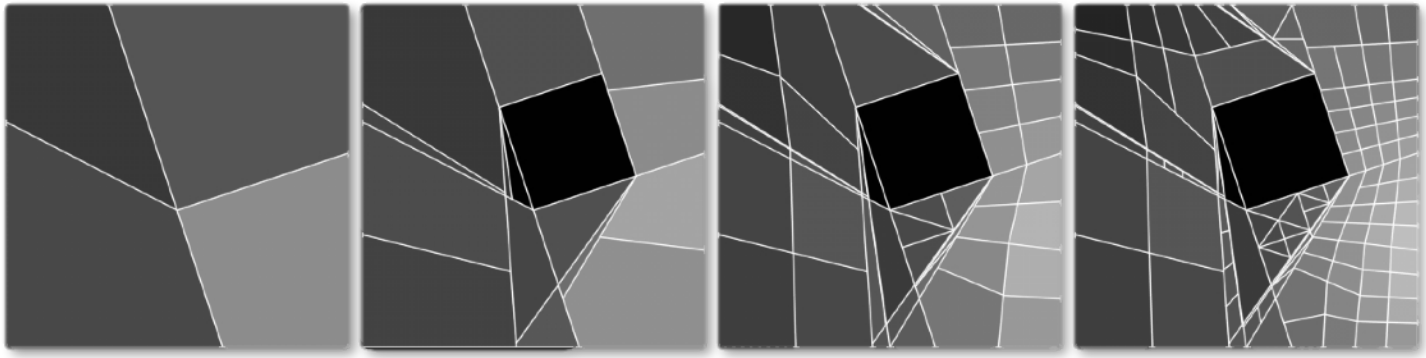


Light leak

Discontinuity meshing



Regular subdivision



Discontinuity meshing

Discontinuity meshing

Discontinuity meshing

Discontinuity meshing

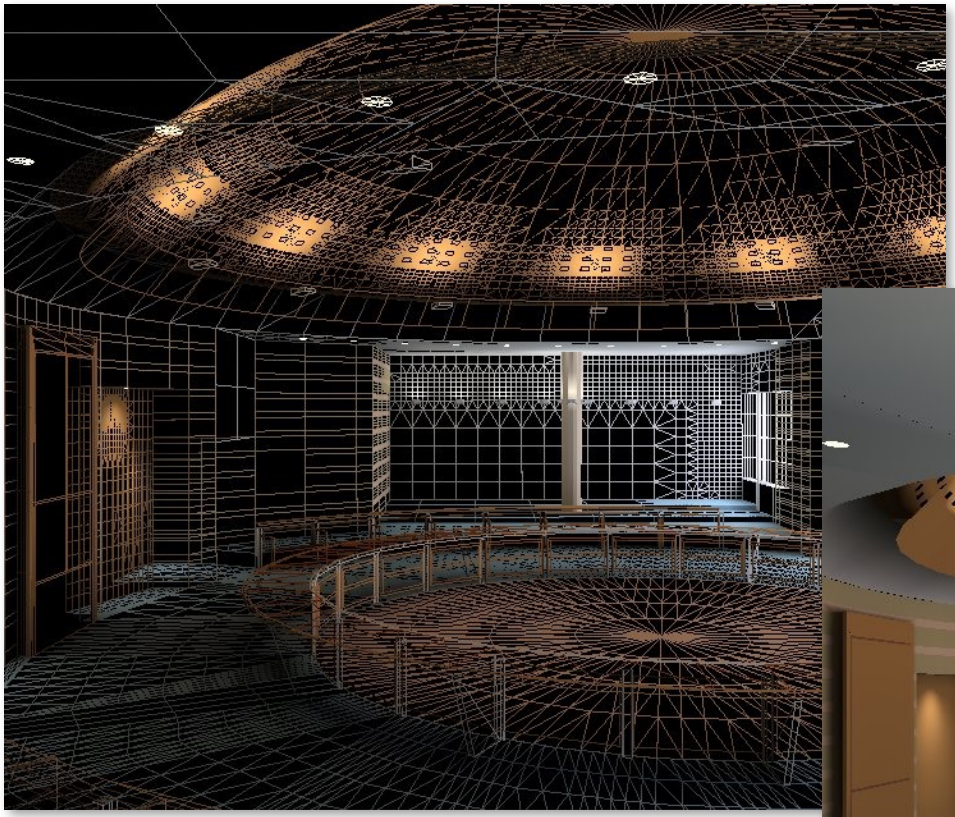
Résultats et comparaison

Rendu Scan-line.
(3DS MAX)

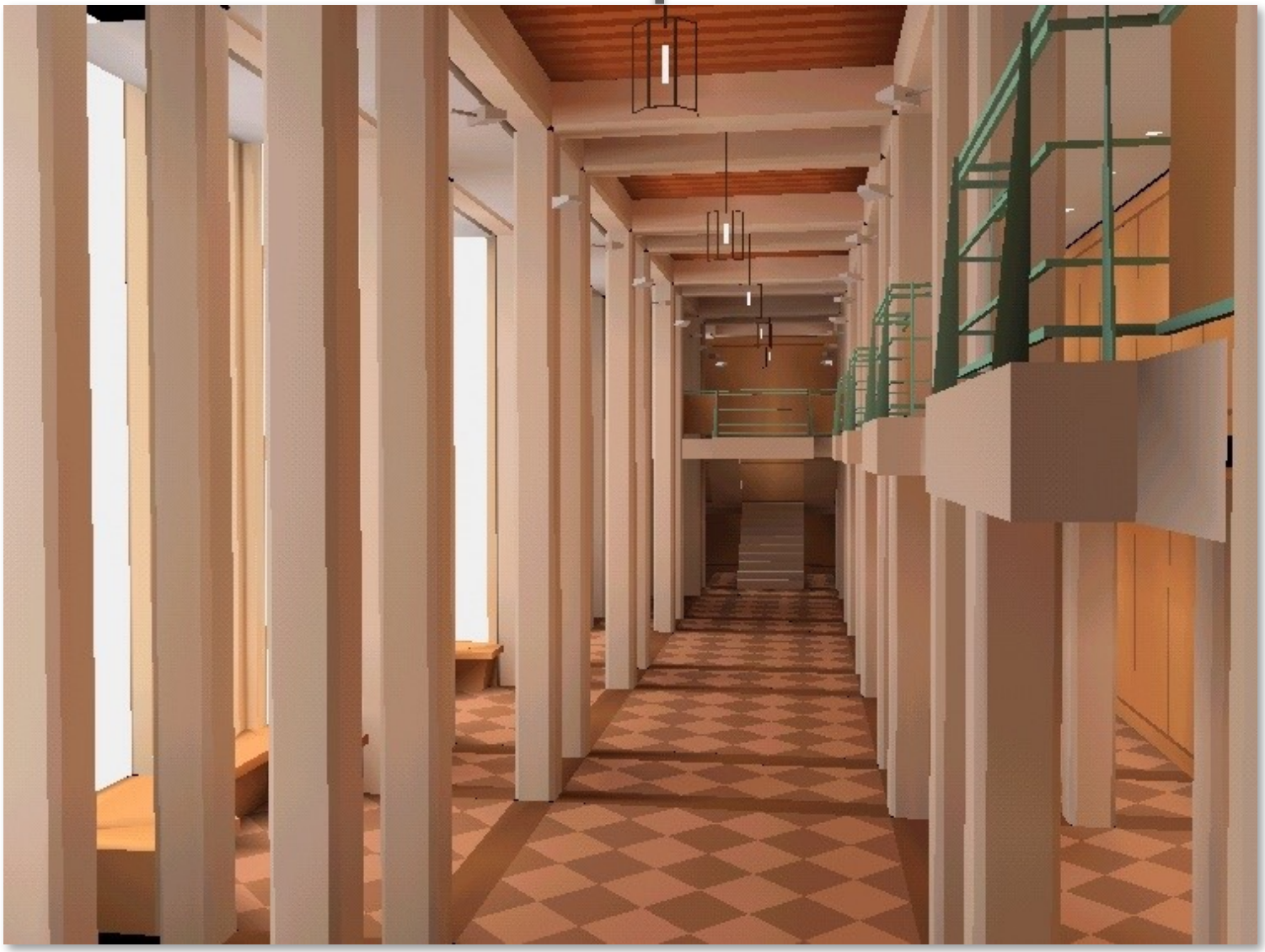
Rendu en radiosité et
lancer de rayons.

Results and comparison

Results and comparison



Results and comparison



Results and comparison

Radiosity today

- Used by architects (Lightscape)
- Used to precompute diffuse lighting for some video games (light maps)

- Not an active research topic anymore
  - Monte-Carlo is more generic
  - But pre-computed radiance transfer is similar (used e.g. in Max Payne 2)

- Complexity = $O(n^2)$ where n = geometry

Virtual Point Lights

- Initially, a technique for interactive rendering
  - “instant radiosity”
- First step:
  - Propagate light in the scene
  - Store as virtual point lights (VPL)
- Second step:
  - Direct illumination from the VPL
  - Can be done on the GPU (100-1000 VPLs/s)
  - Needs shadows

Virtual Point Lights

- Also Point-Based Global Illumination
- Separate illumination from scene geometry
- Group point lights for more efficiency:
  - Light cuts, row-column matrix sampling

Row-Column Matrix Sampling

- VPL as a method for global illumination
- Many light sources (100k)
- Matrix interpretation:
  - Columns= contribution from one light source
  - Rows= one pixel
  - Huge matrix (2m x 100k)

  -> Don't compute the whole matrix!

- Following slides: presentation of RCMS, Siggraph 2007, Miloš Hašan.

Indirect Illumination → Many Lights

=

Σ (




)

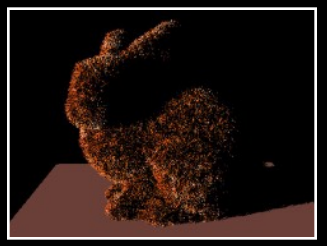
100,000 point lights

Environment Map → Many Lights

=

Σ (






)

100,000 point lights

Sun, Sky, Indirect → Many Lights

=

Σ (

)

100,000 point lights

Brute Force Takes Minutes

- Why not sum all columns?
  - With 100,000 lights, still several minutes

10 min

13 min

20 min

Our Contribution

- Fast, accurate, GPU-based approximation

Brute force:

10 min

13 min

20 min

Our result:

3.8 sec

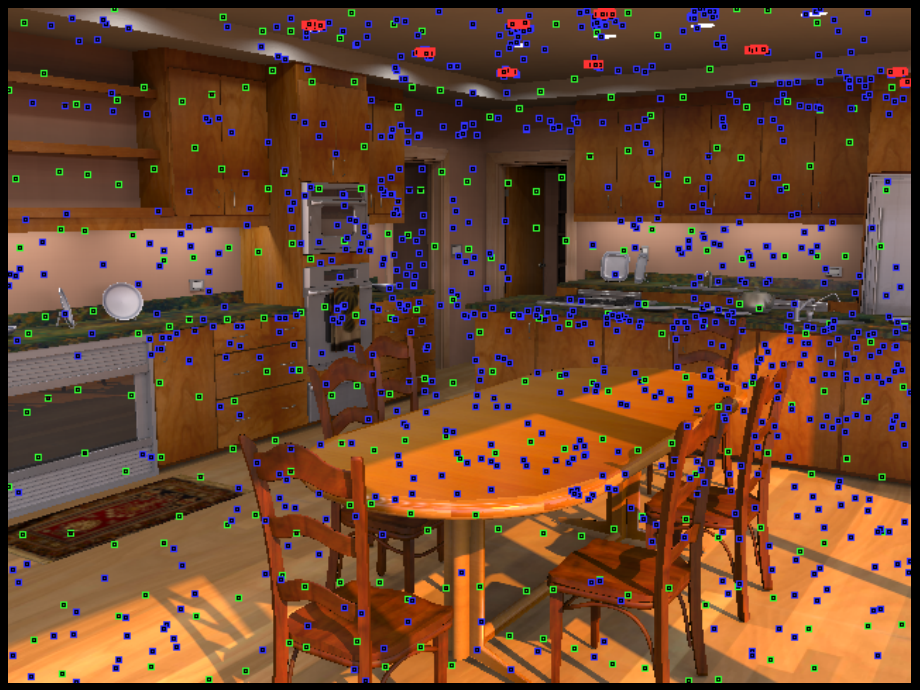
13.5 sec

16.9 sec

- Application: Preview for lighting design

Conversion to Many Lights

- **Area, indirect, sun/sky**



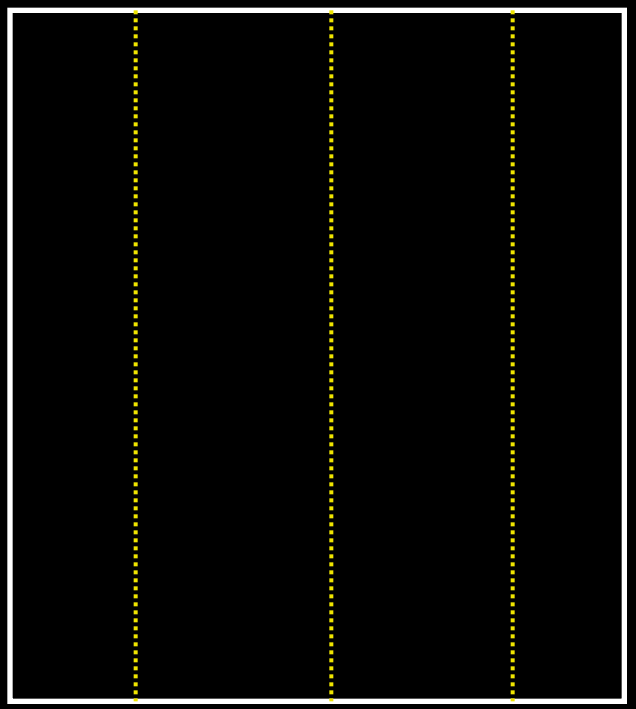
Courtesy Walter et al., Lightcuts, SIGGRAPH 05/06

A Matrix Interpretation

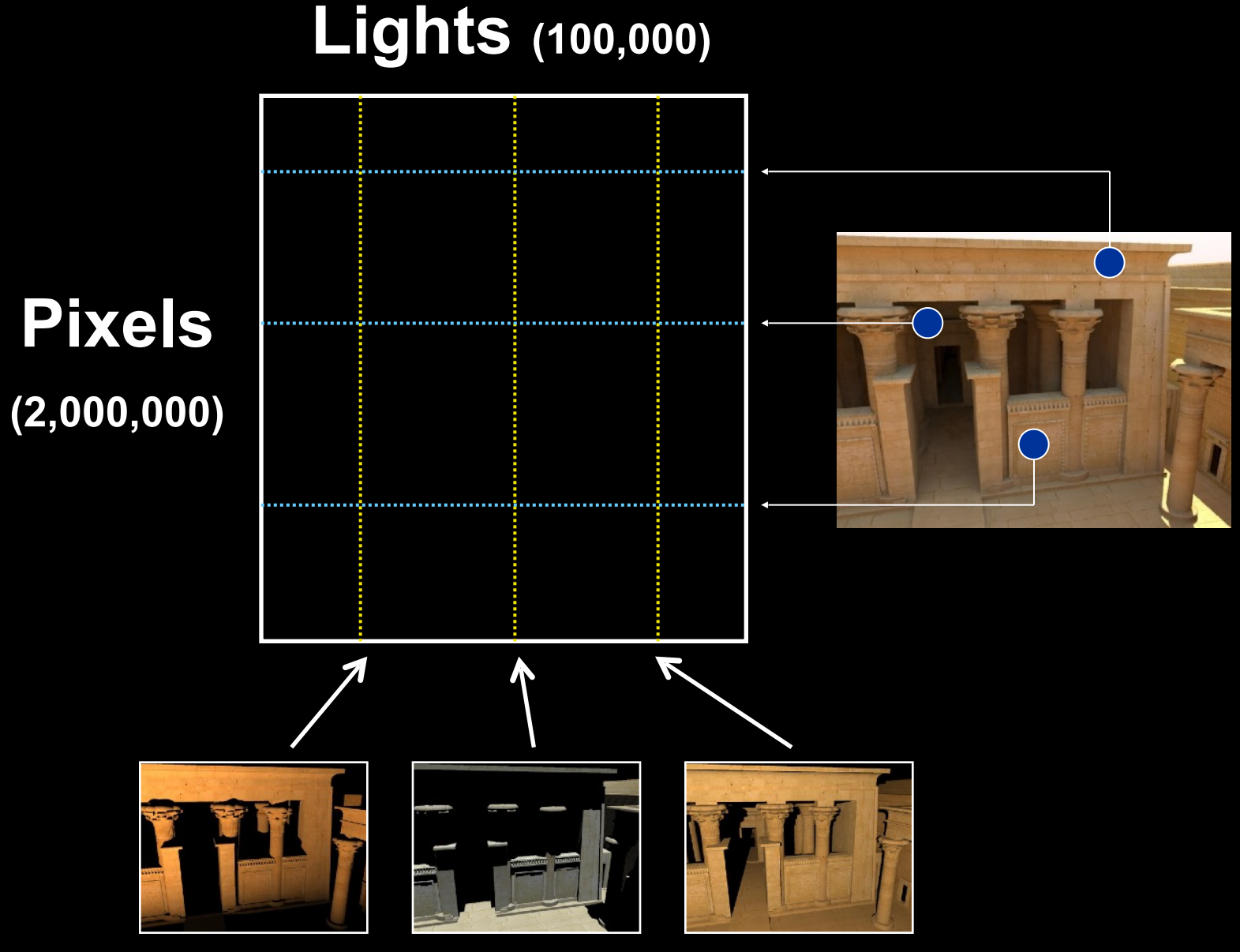
Lights (100,000)

Pixels

(2,000,000)



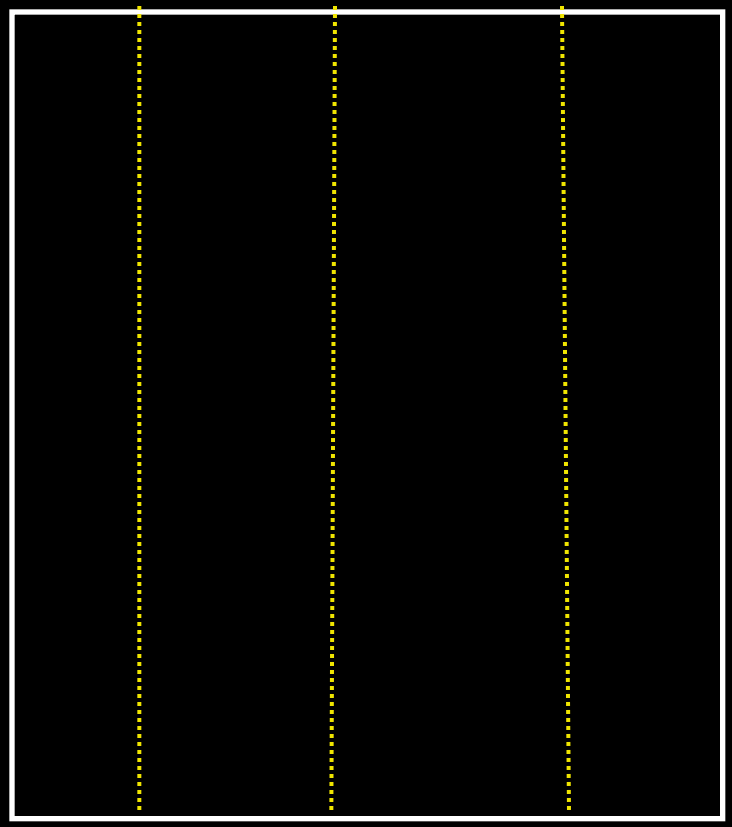
A Matrix Interpretation



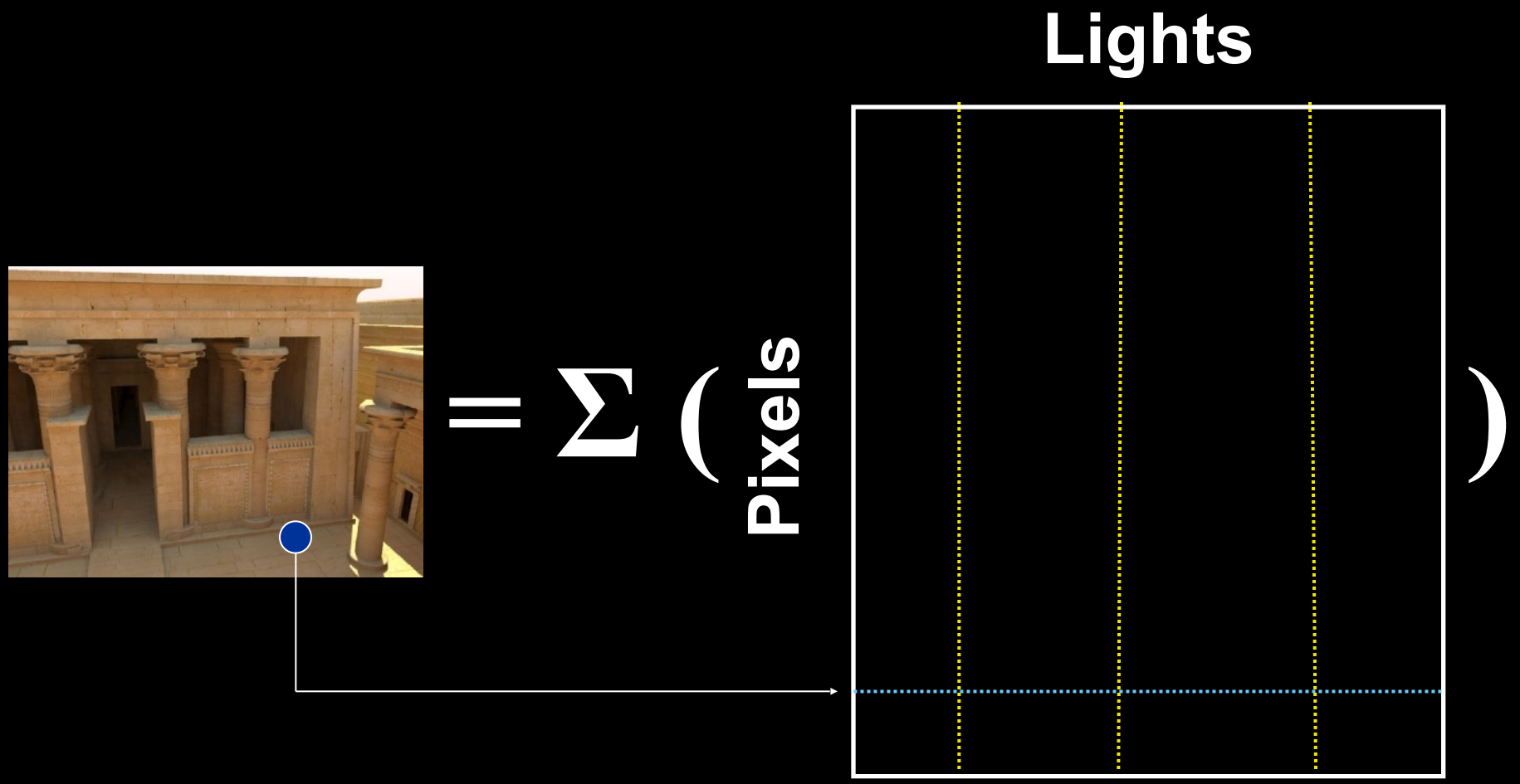
Problem Statement

$$= \sum_{\text{Pixels}} \left(\right)$$

Lights

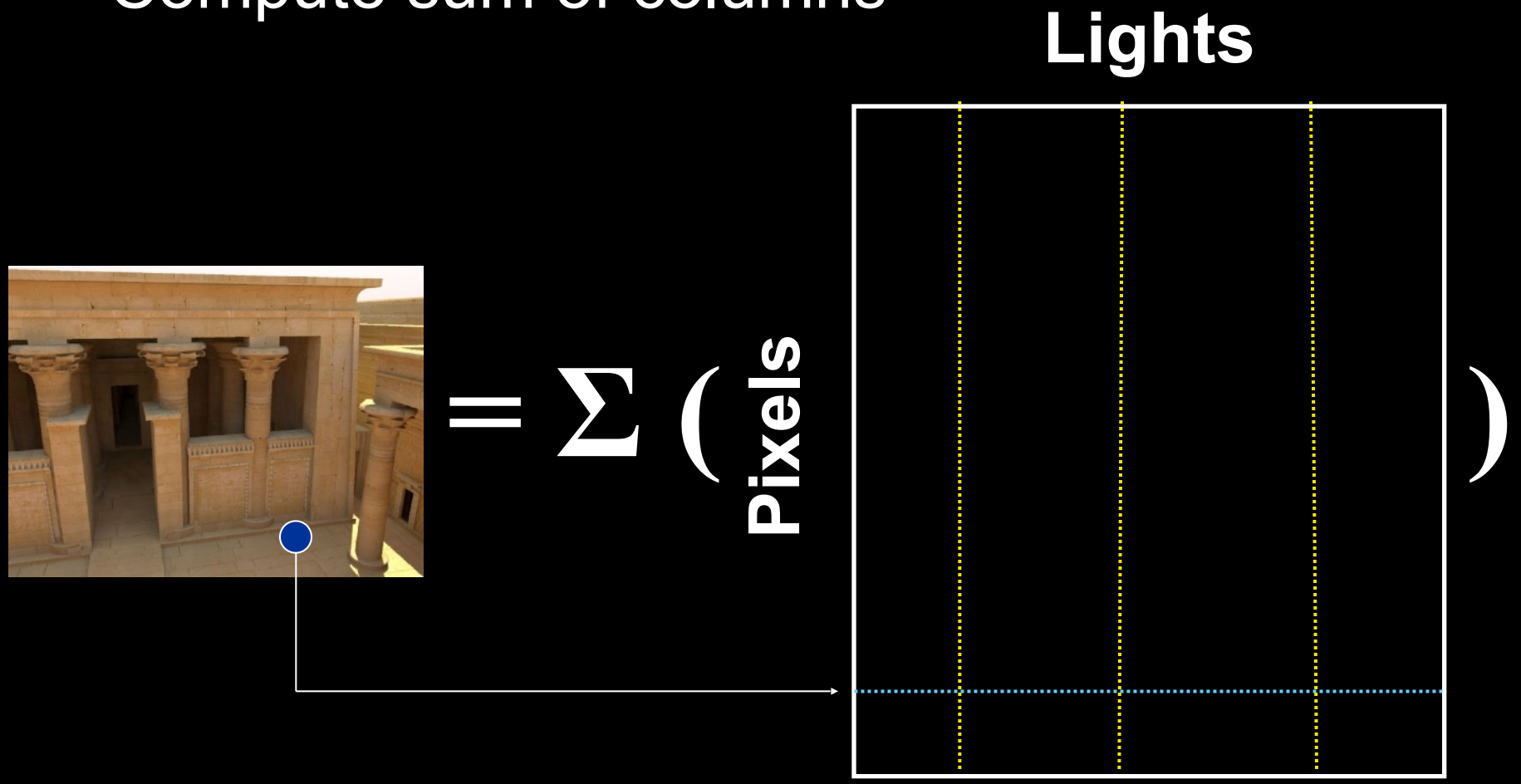


Problem Statement



Problem Statement

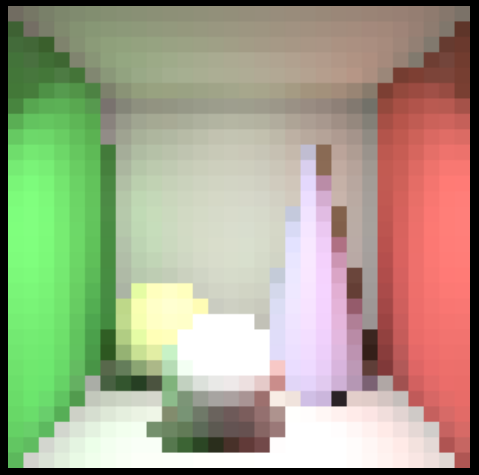
- Compute sum of columns



- Note: We don’t have the matrix data

Insight #1: Matrix has structure

- Compute small subset of elements
- Reconstruct



A simple scene

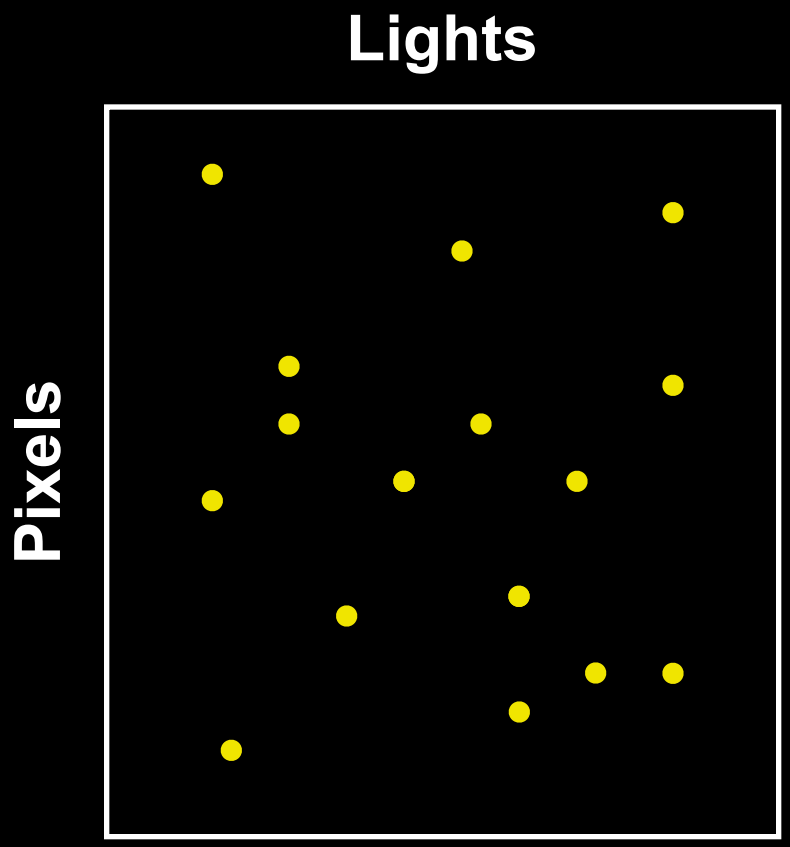
30 x 30 image

643 lights

900 pixels

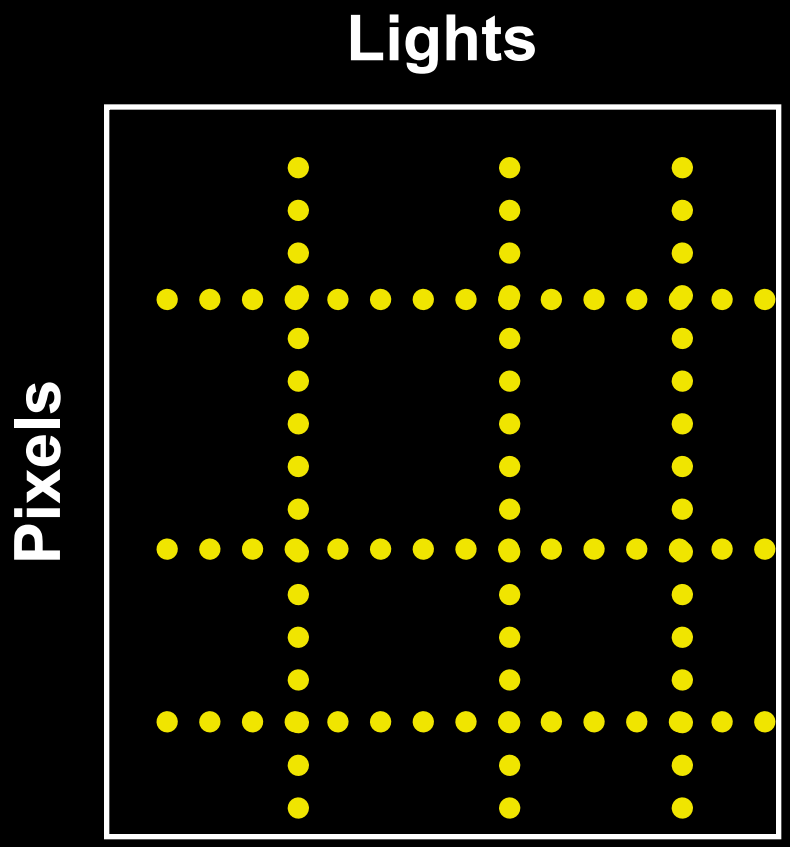
The matrix

Insight #2: Sampling Pattern Matters



Point-to-point visibility: Ray-tracing

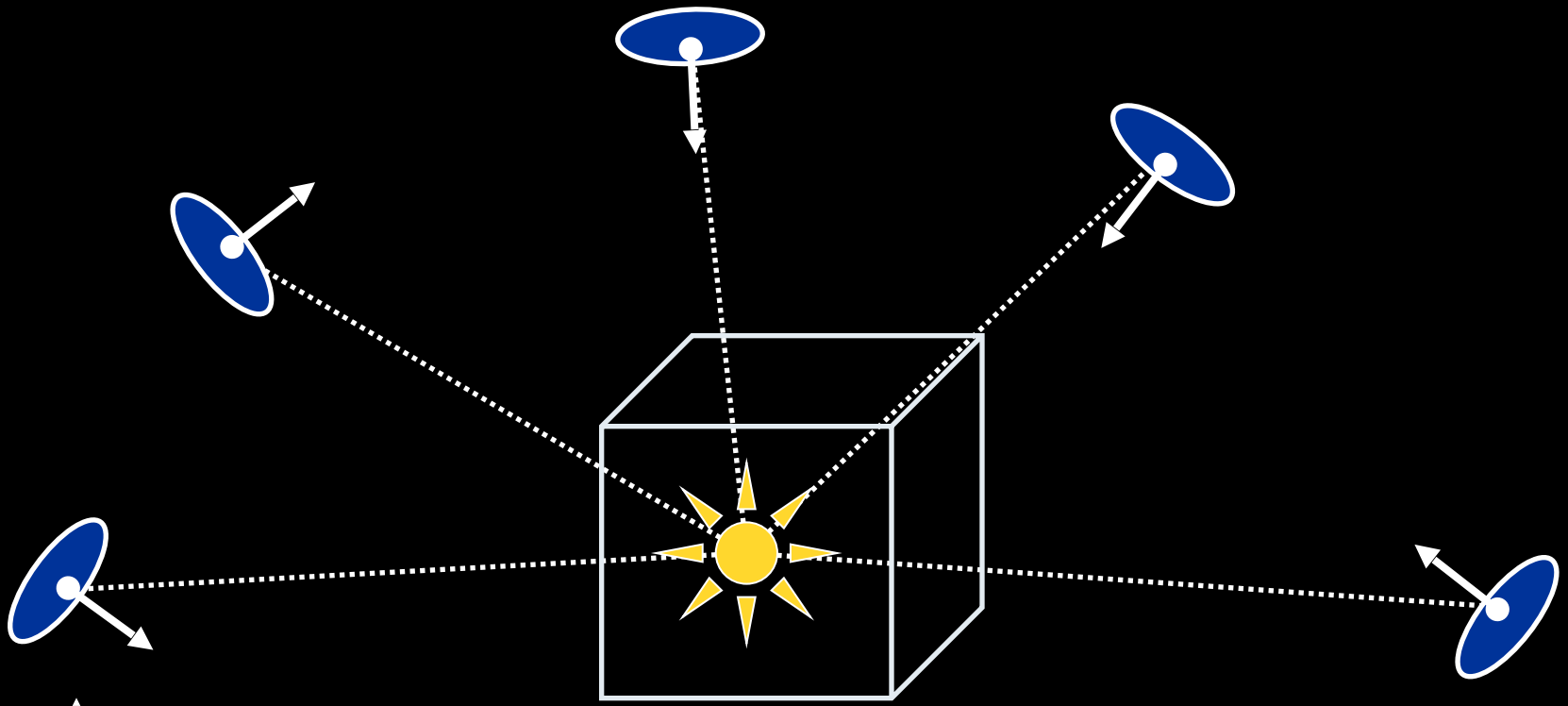
Insight #2: Sampling Pattern Matters



Point-to-many-points visibility: Shadow-mapping

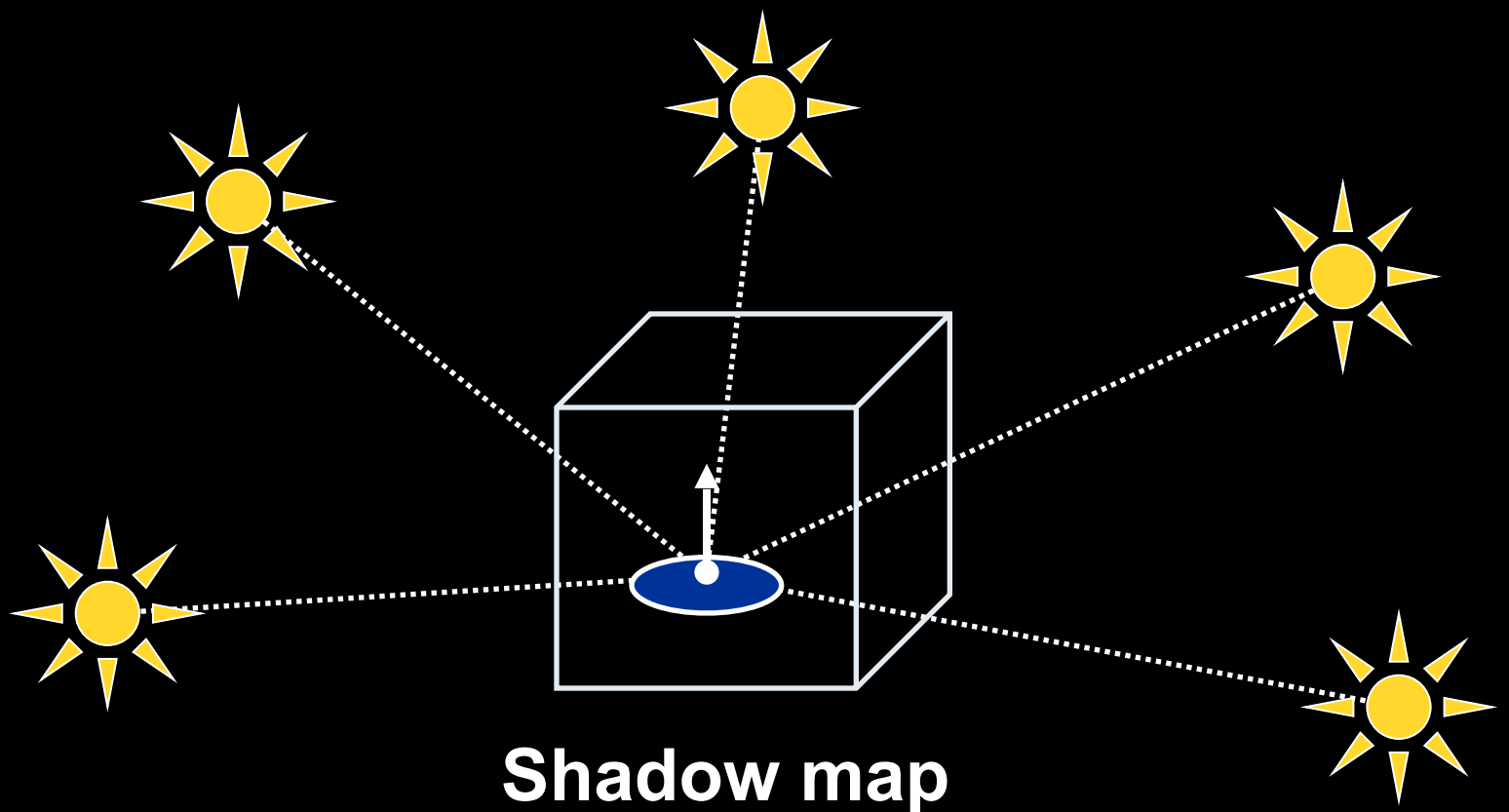
Row-Column Duality

- **Columns: Regular Shadow Mapping**



Row-Column Duality

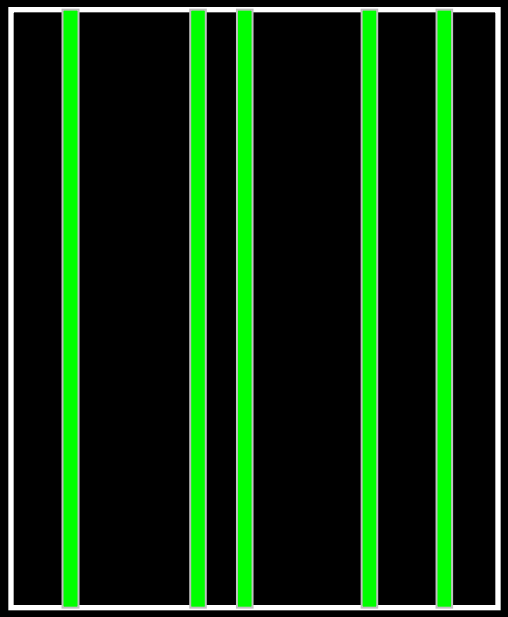
- **Rows: Also Shadow Mapping!**



Shadow map
at sample position

Image as a Weighted Column Sum

- **The following is possible:**

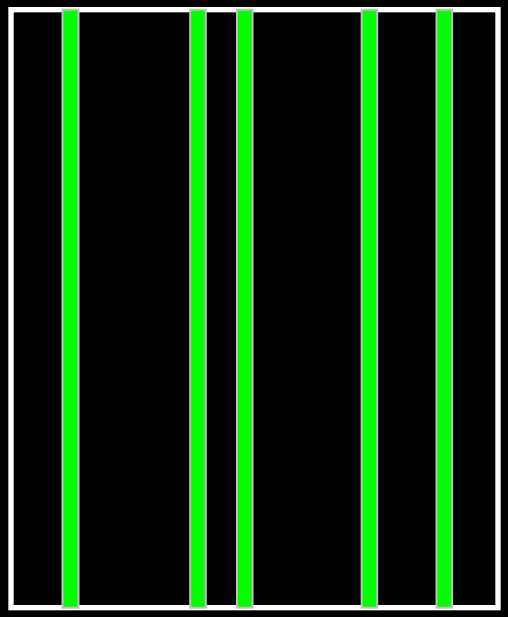


compute very small subset of columns

compute weighted sum

Image as a Weighted Column Sum

- **The following is possible:**

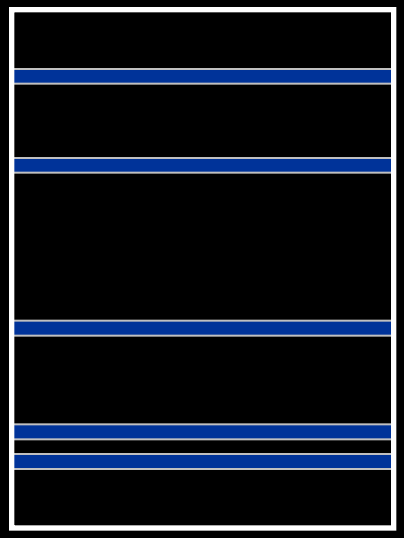


compute very small subset of columns

compute weighted sum

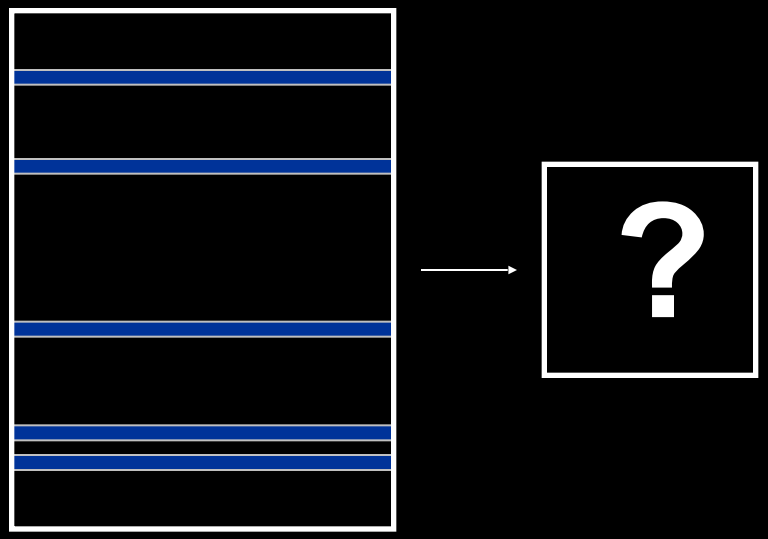
- **Use rows to choose a good set of columns!**

Exploration and Exploitation



compute rows (explore)

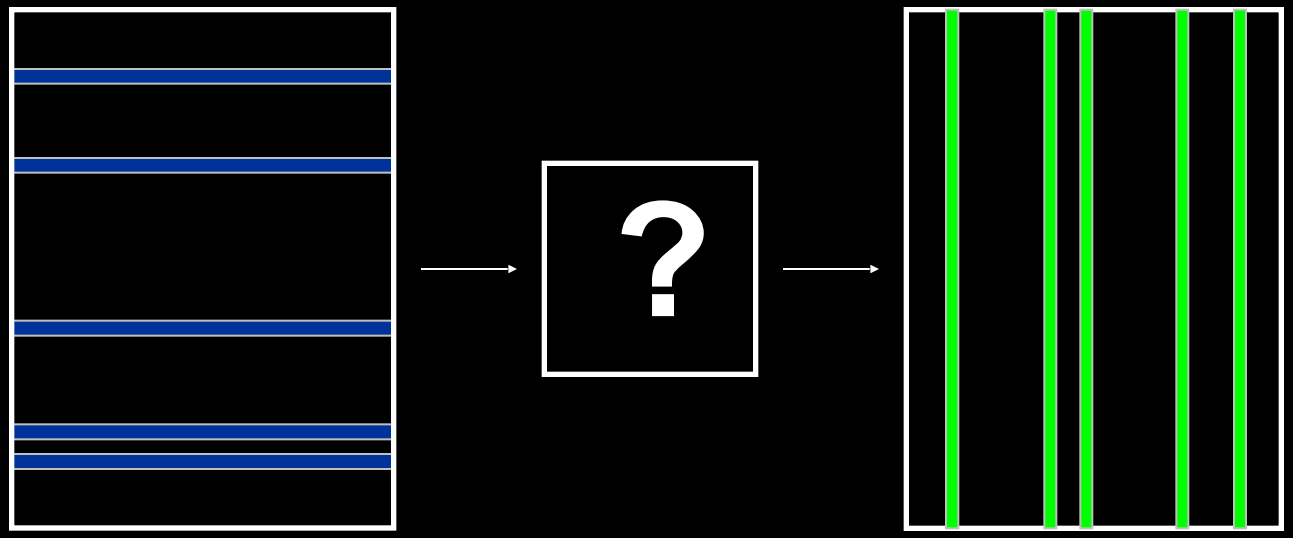
Exploration and Exploitation



compute rows (explore)

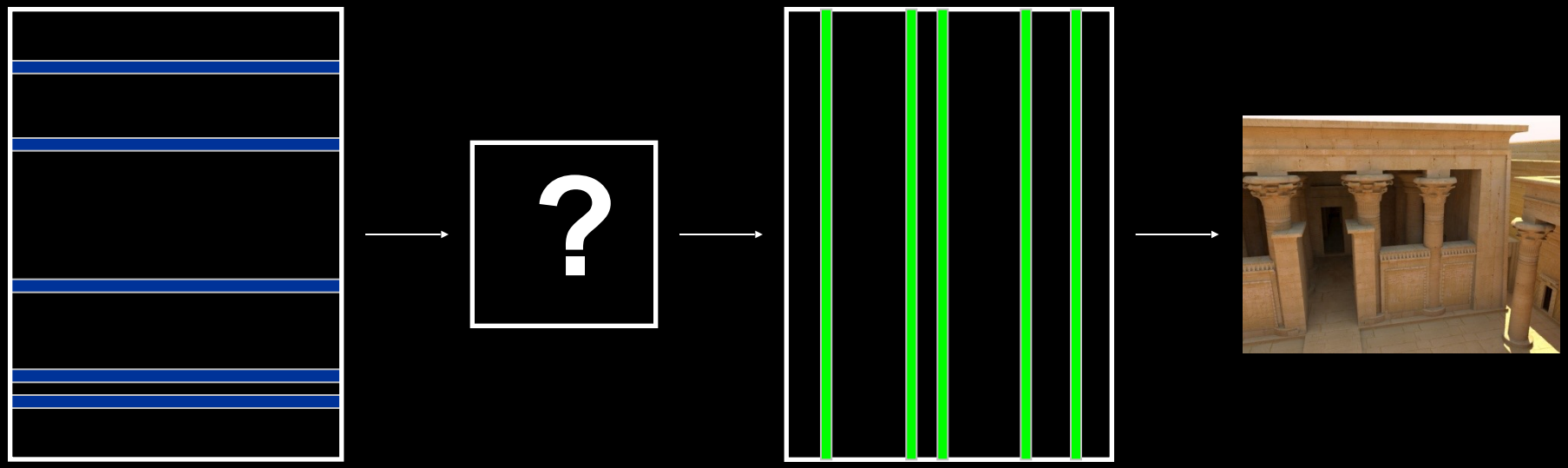
choose columns and weights

Exploration and Exploitation



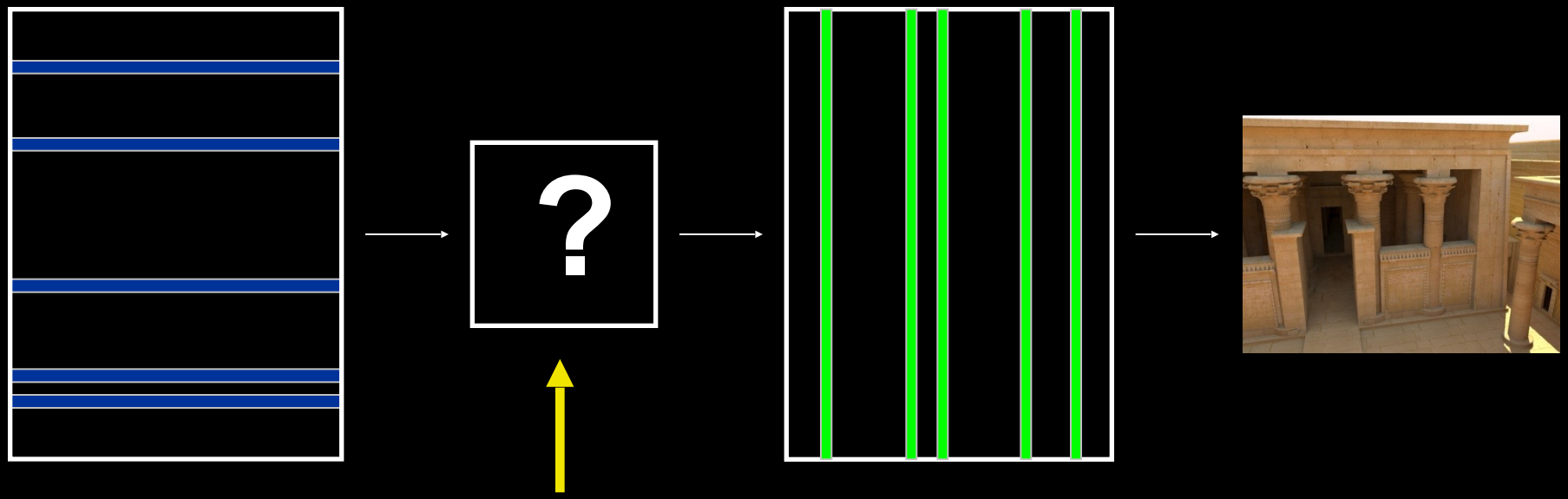
compute rows (explore) **choose columns and weights** **compute columns (exploit)**

Exploration and Exploitation



compute rows (explore) **choose columns and weights** **compute columns (exploit)** **weighted sum**

Exploration and Exploitation



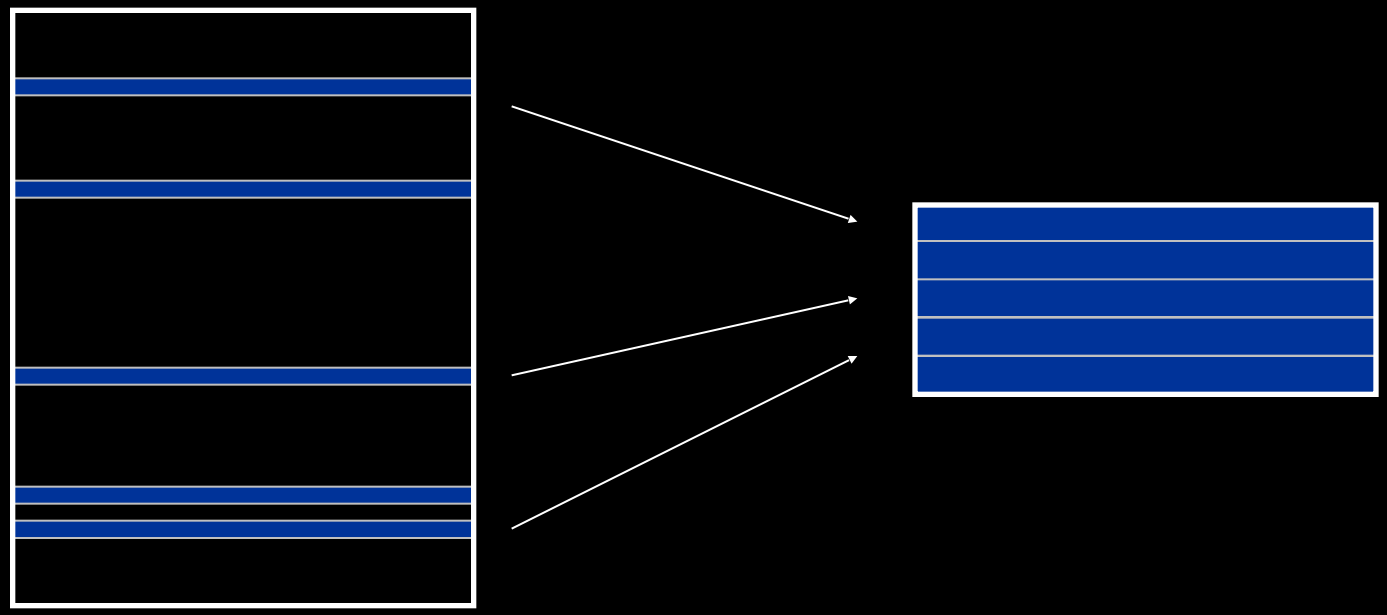
compute rows (explore)

how to choose columns and weights?

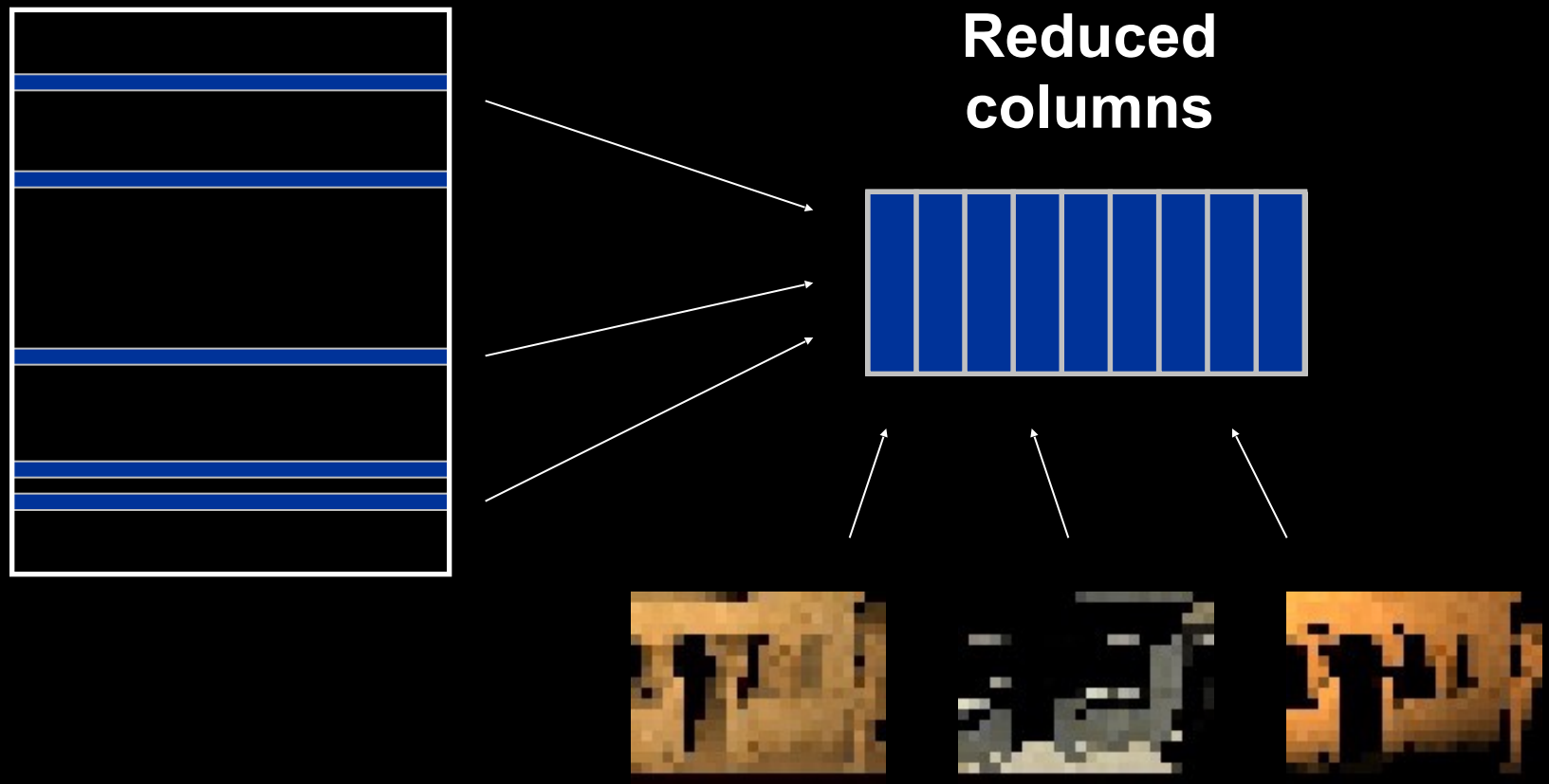
compute columns (exploit)

weighted sum

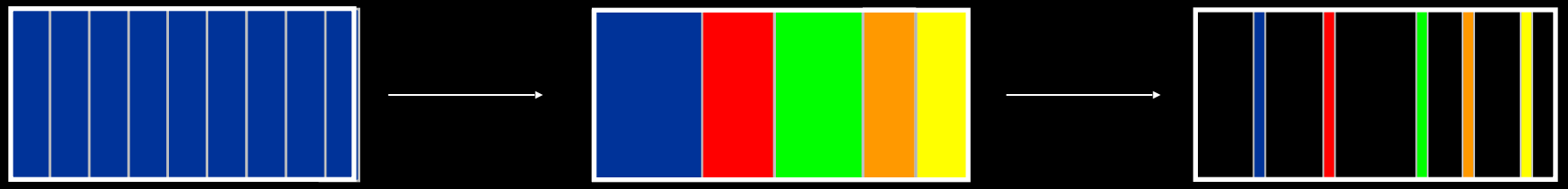
Reduced Matrix



Reduced Matrix



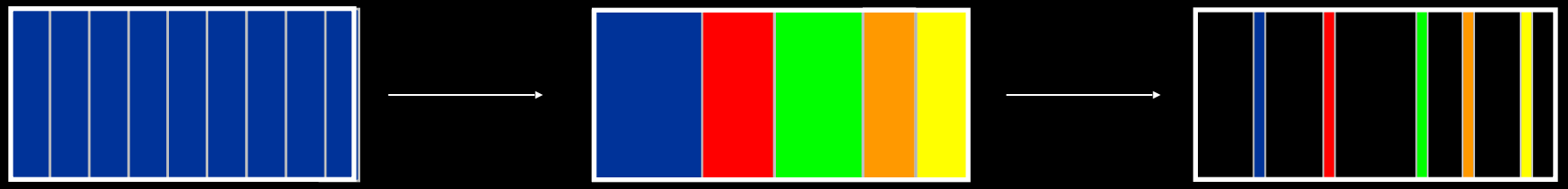
Clustering Approach



Choose k clusters

Choose representative columns

Clustering Approach

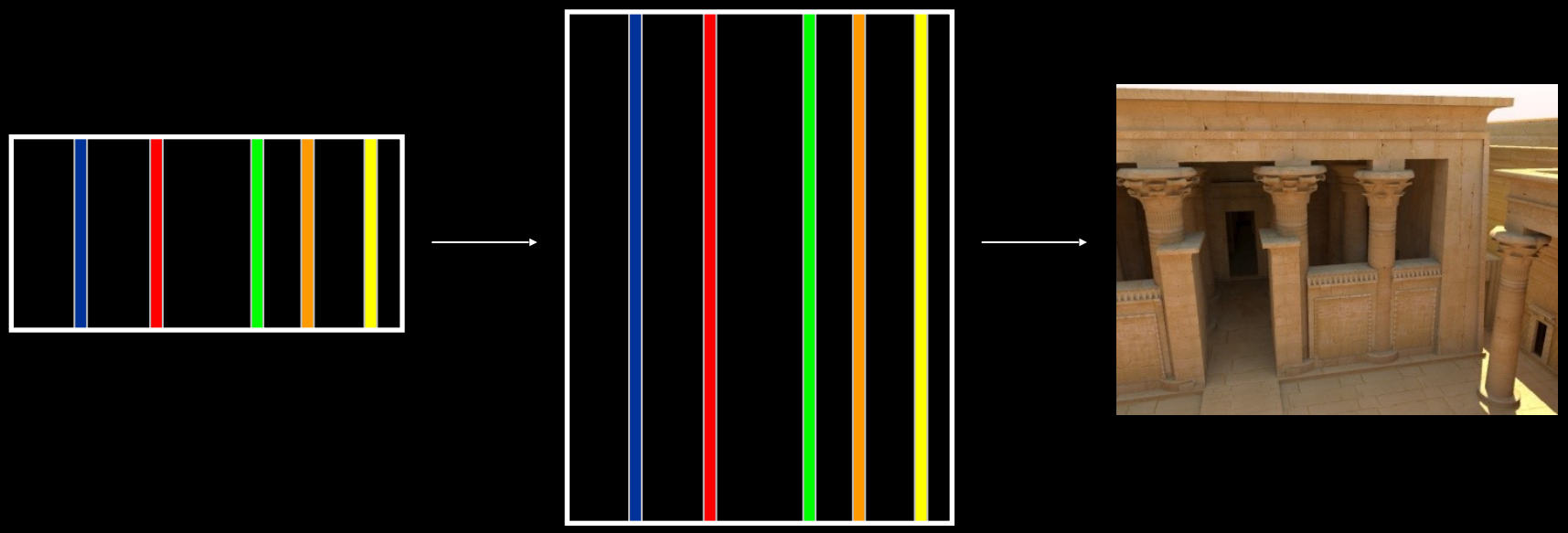


Reduced columns

Choose k clusters

Choose representative columns

Reduced → Full



Representative columns

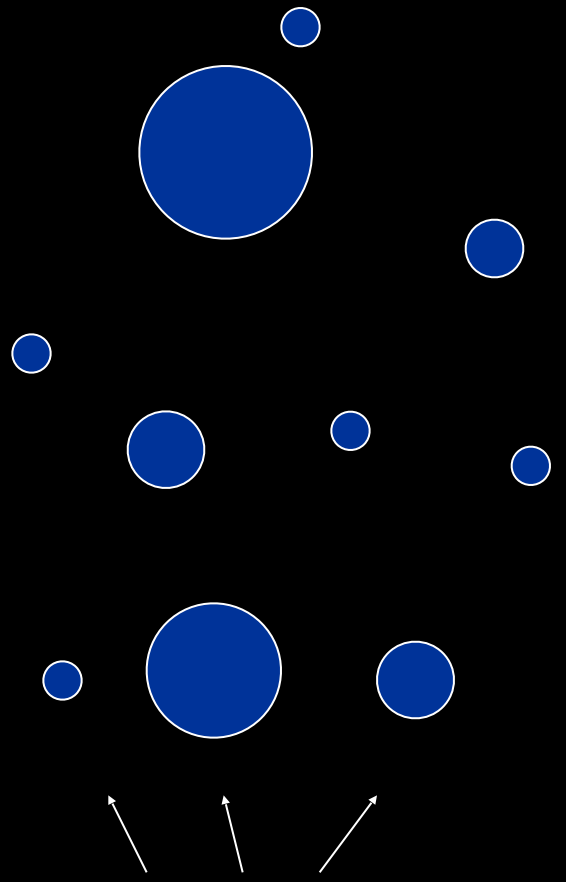
Use the same representatives for the full matrix

Weighted sum

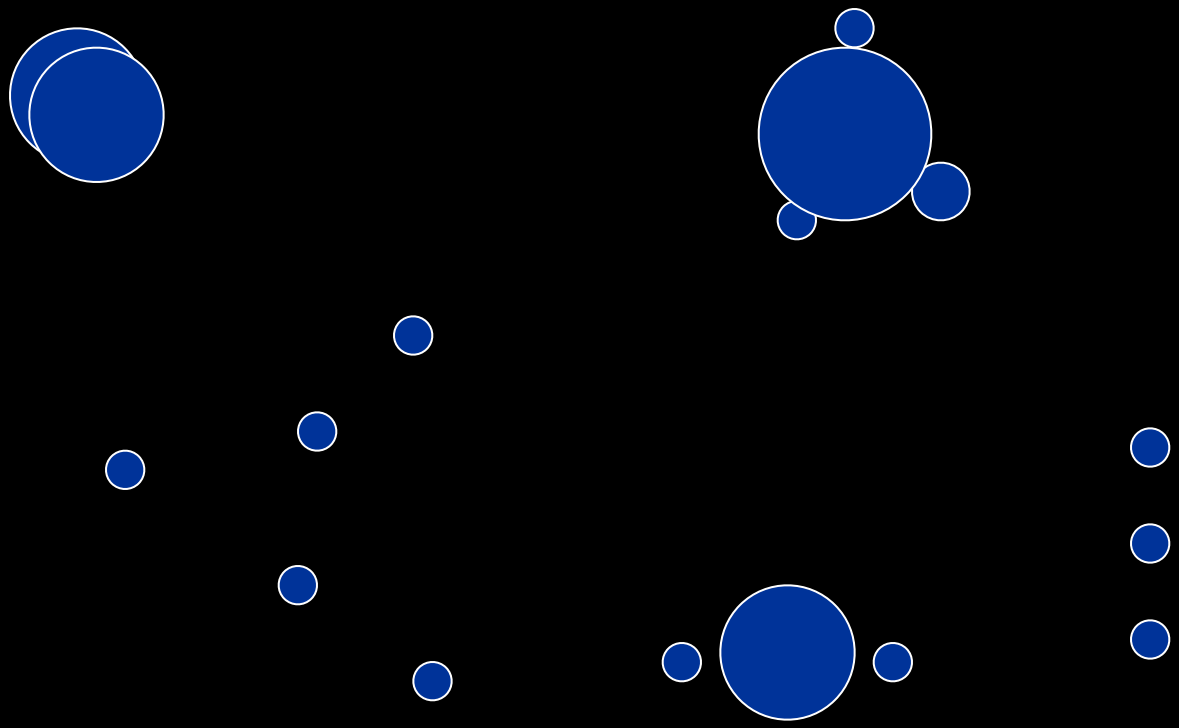
Visualizing the Reduced Columns

Reduced columns: vectors in high-dimensional space

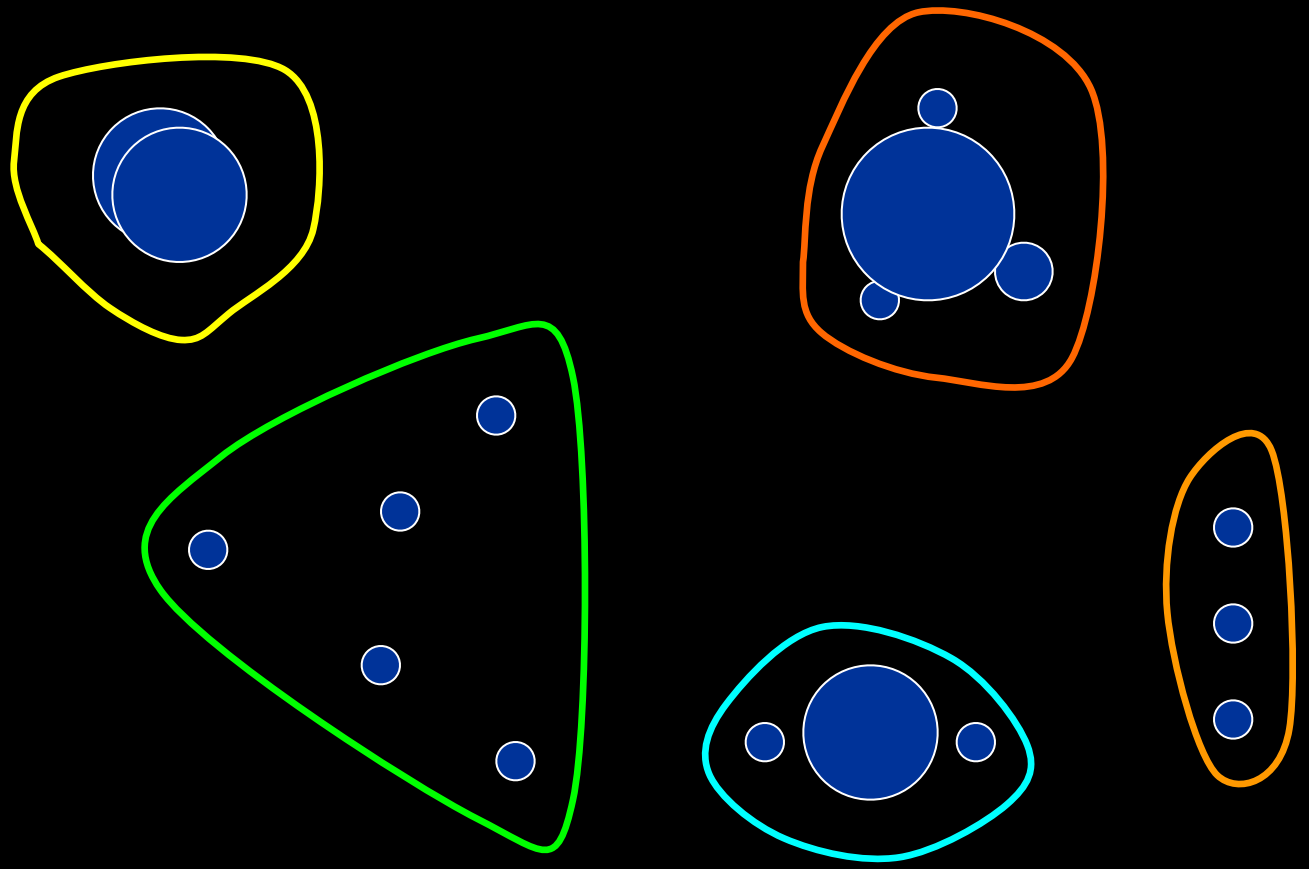
visualize as …



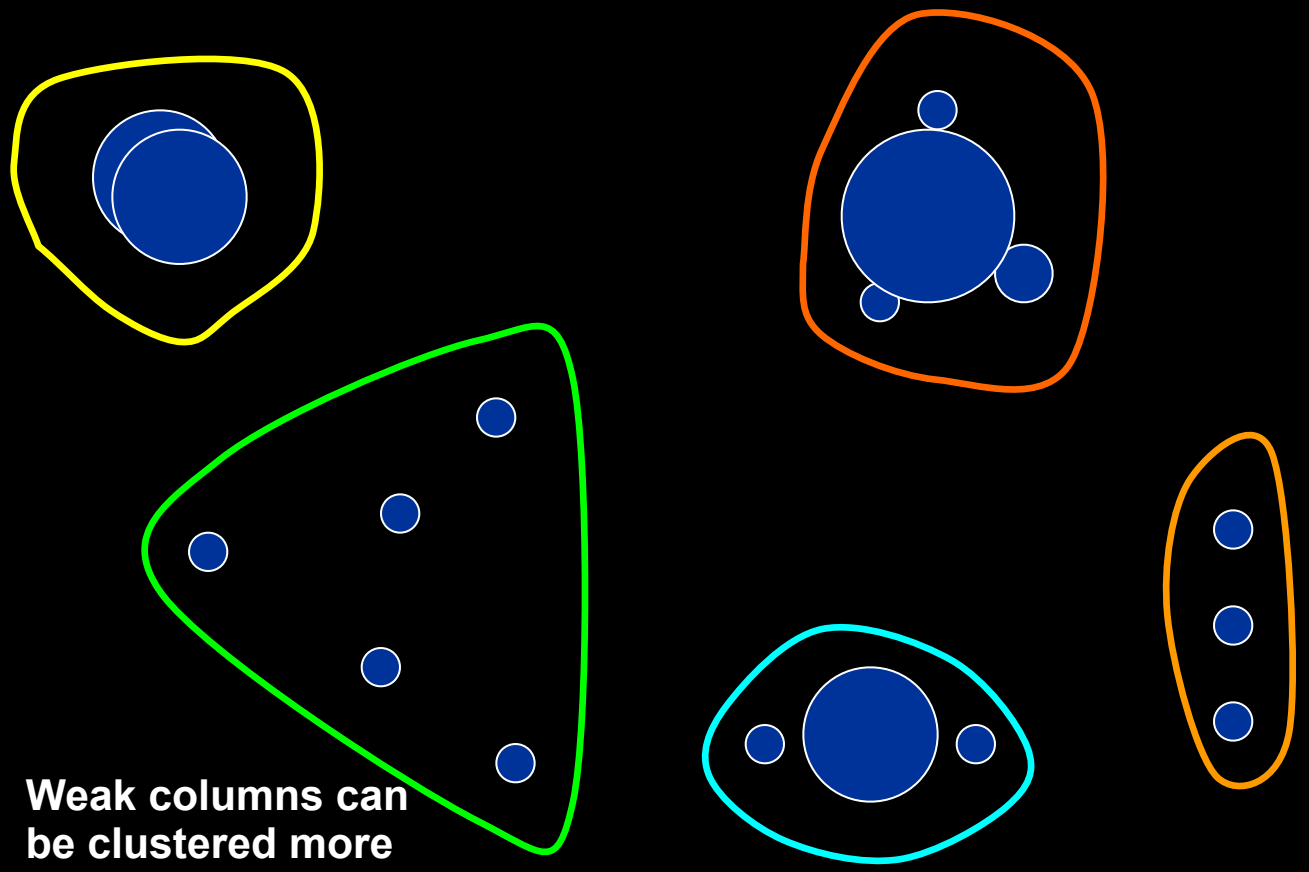
Clustering Illustration



Clustering Illustration

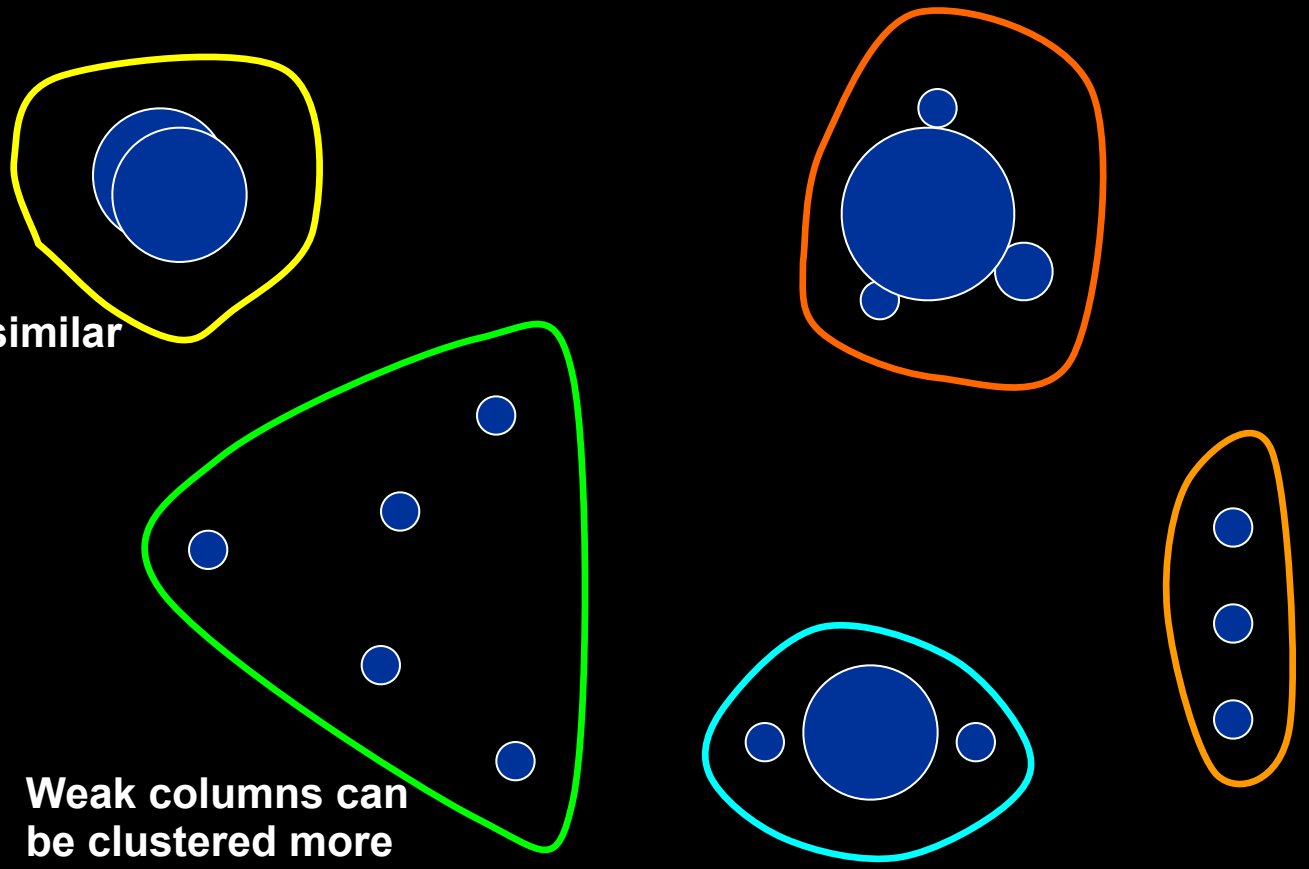


Clustering Illustration

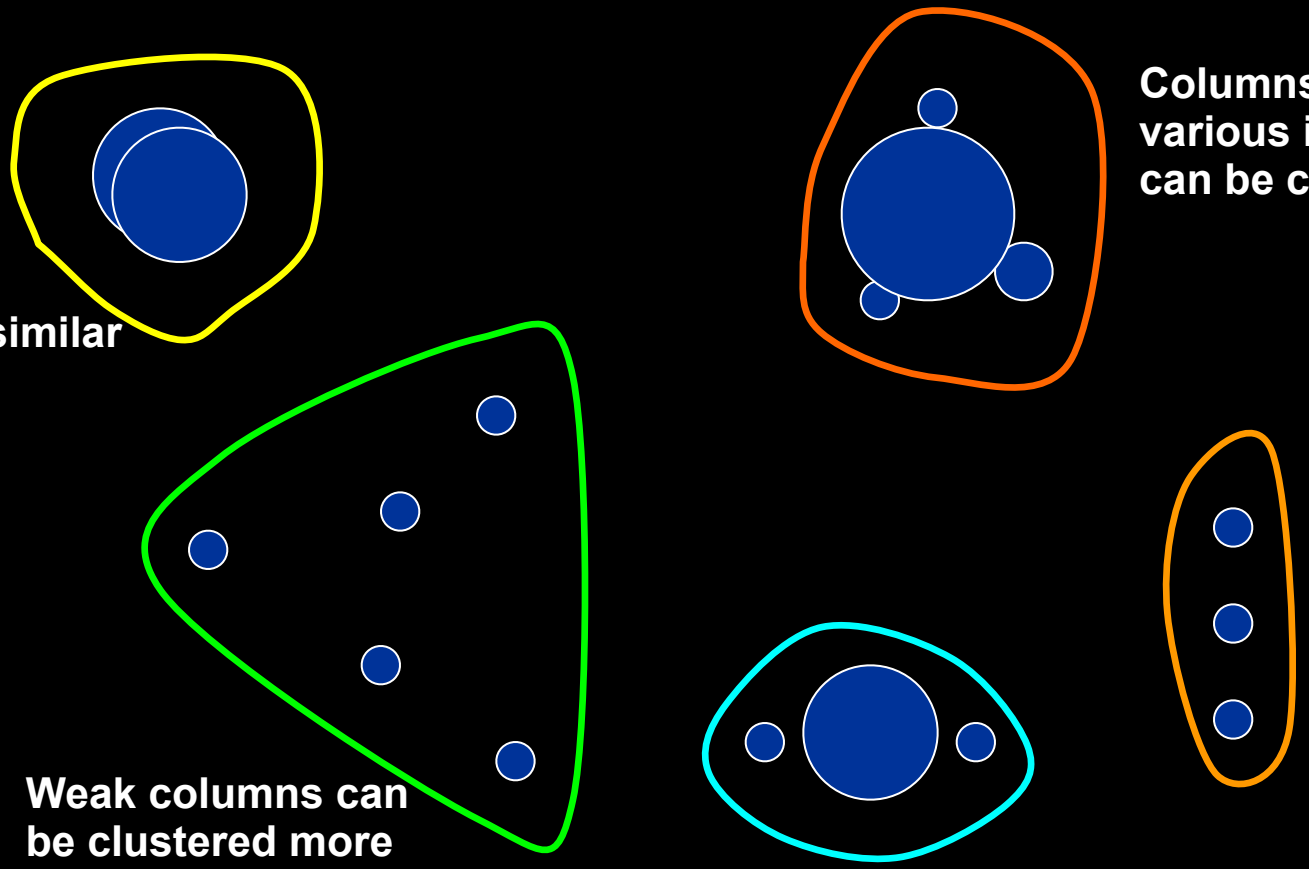


Weak columns can be clustered more easily

Clustering Illustration



Clustering Illustration



The Clustering Metric

- Minimize: $$\sum_{p=1,\ldots,k} cost(C_p)$$

- where: $$cost(C) = \sum_{i,j \in C} w_i \; w_j \; \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

The Clustering Metric

- Minimize: $\underbrace{\sum_{p=1,\ldots,k} cost(C_p)}_{\text{total cost of all clusters}}$

- where: $cost(C) = \sum_{i,j \in C} w_i \; w_j \; \|\mathbf{x}_i - \mathbf{x}_j\|^2$

The Clustering Metric

- Minimize:

$$\sum_{p=1,\ldots,k} cost(C_p)$$

total cost of all clusters

- where:

$$cost(C) = \sum_{i,j \in C} w_i \; w_j \; \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

cost of a cluster

sum over all pairs in it

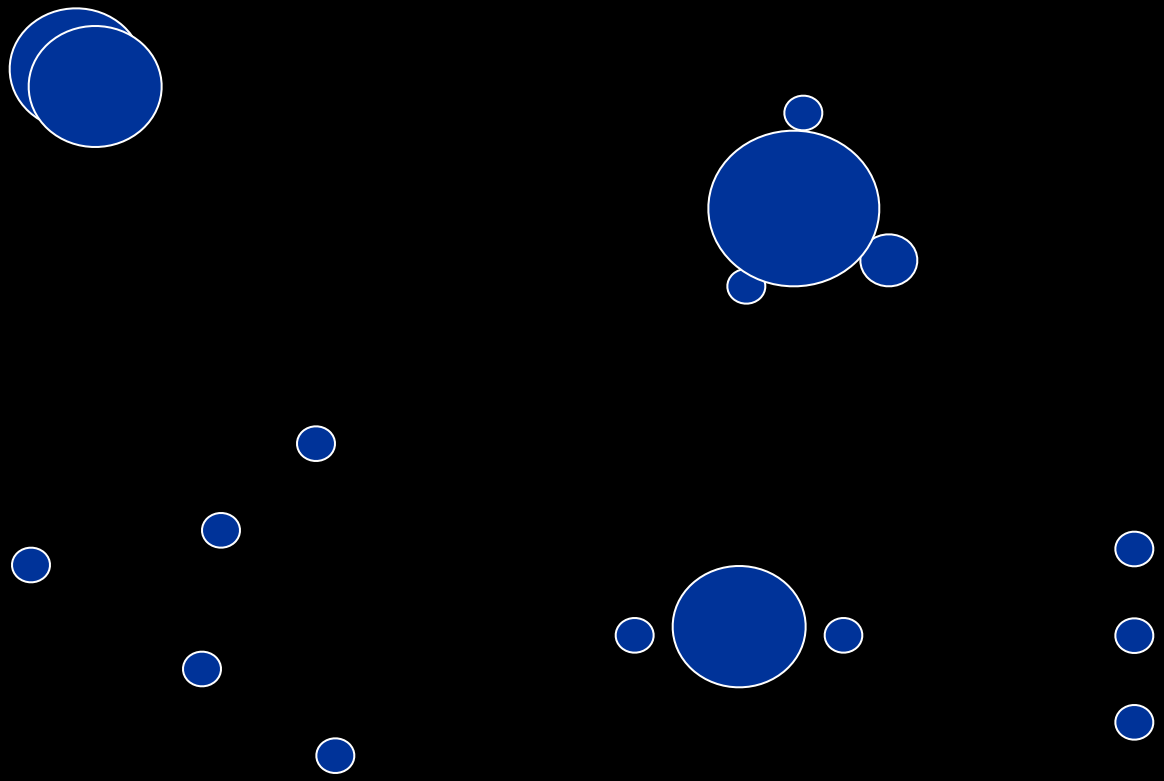
norms of the reduced columns

squared distance between normalized reduced columns

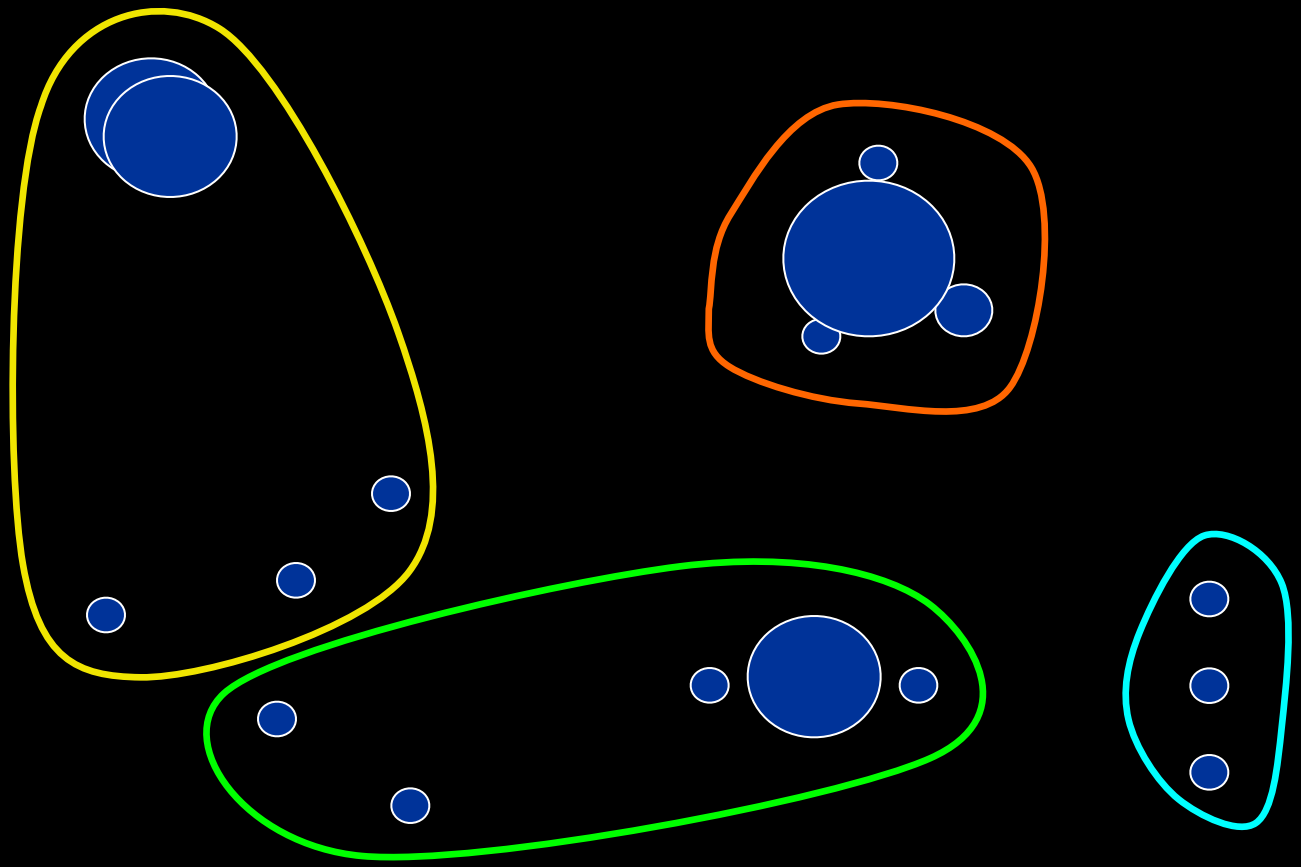
How to minimize?

- Problem is NP-hard
- Not much previous research
- Should handle large input:
  - 100,000 points
  - 1000 clusters
- We introduce 2 heuristics:
  - Random sampling
  - Divide & conquer

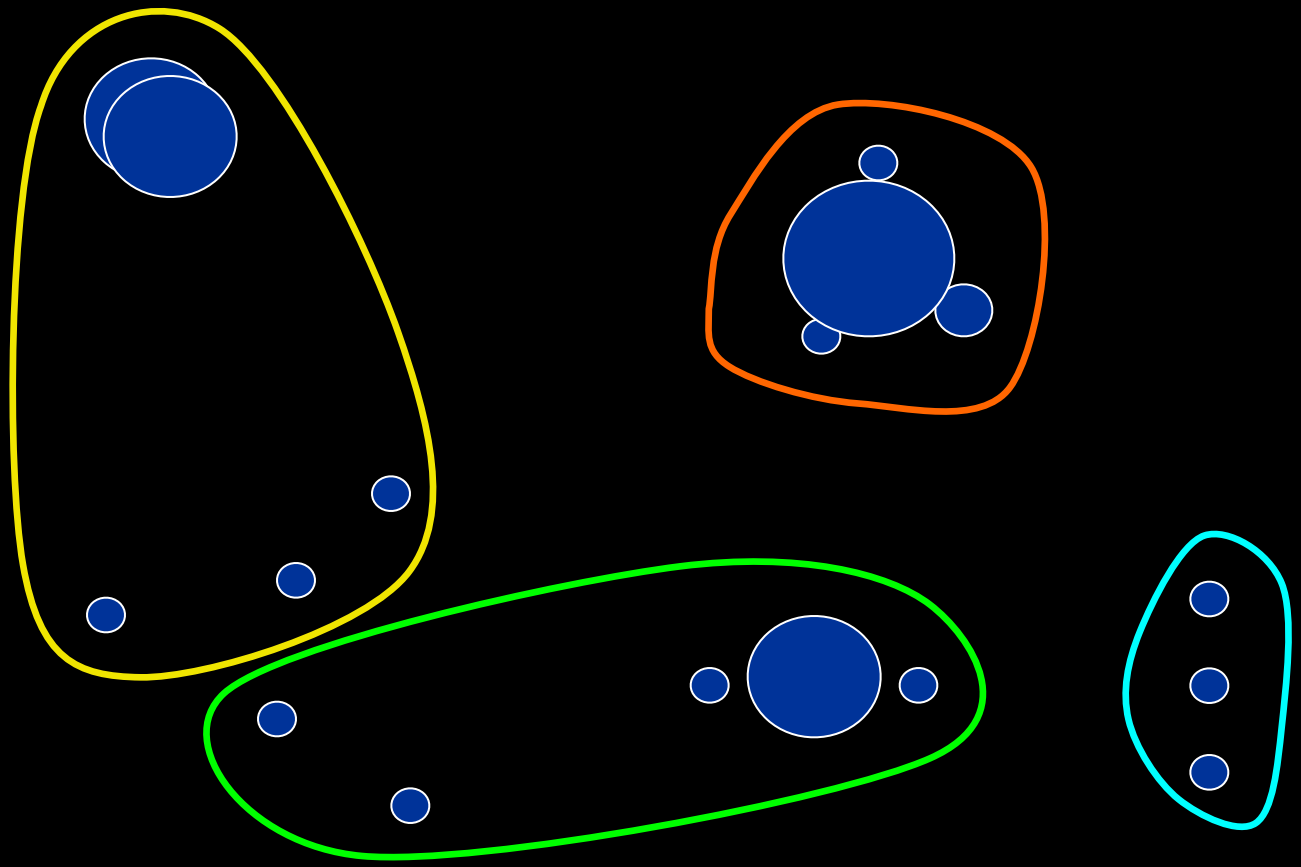
Clustering by Random Sampling



Clustering by Random Sampling

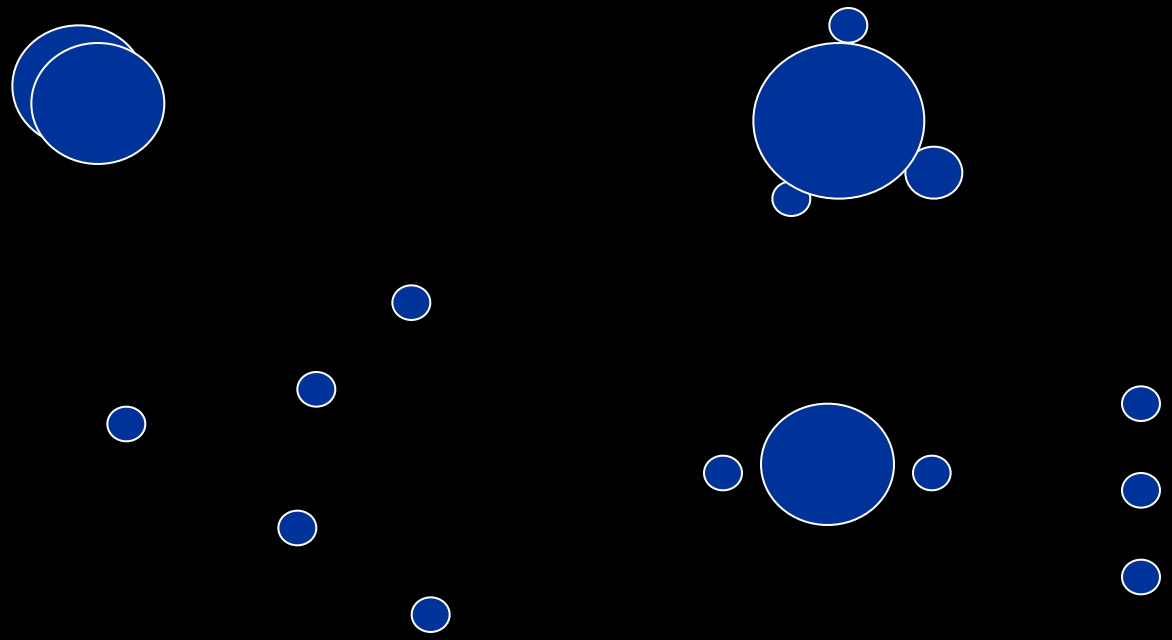


Clustering by Random Sampling

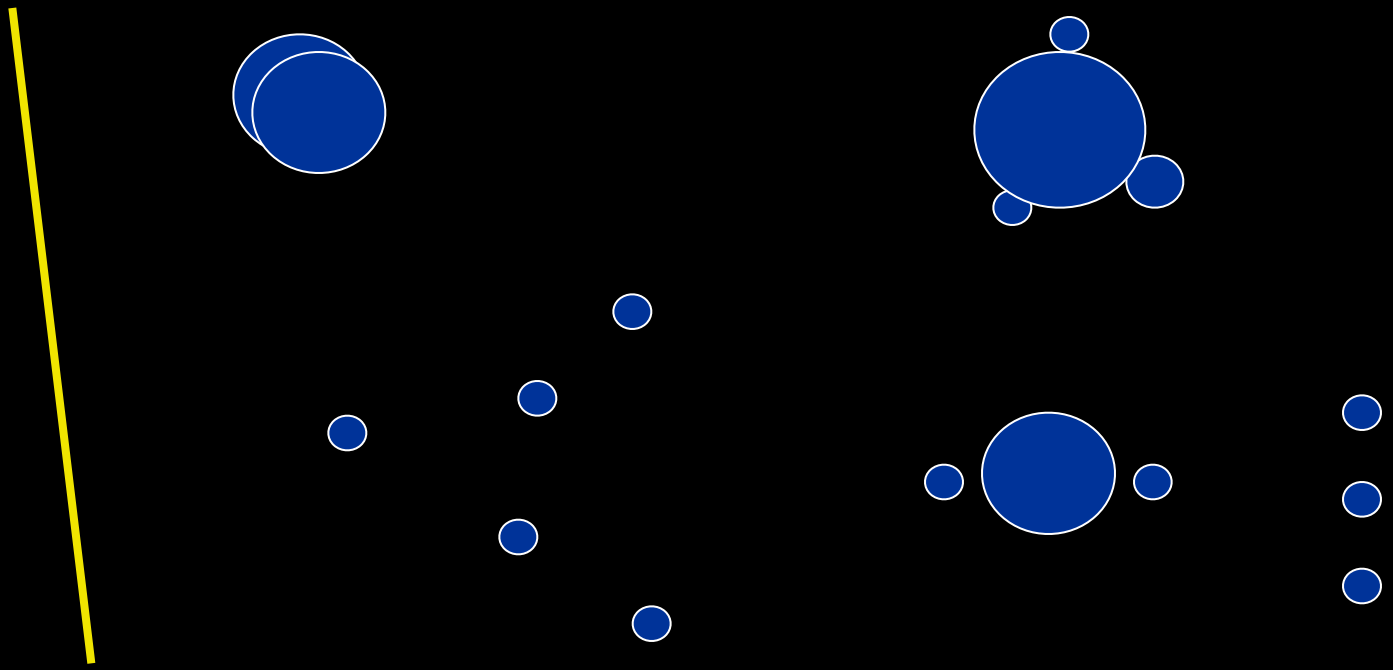


- **Very fast (use optimized BLAS)**
- **Some clusters might be too small / large**

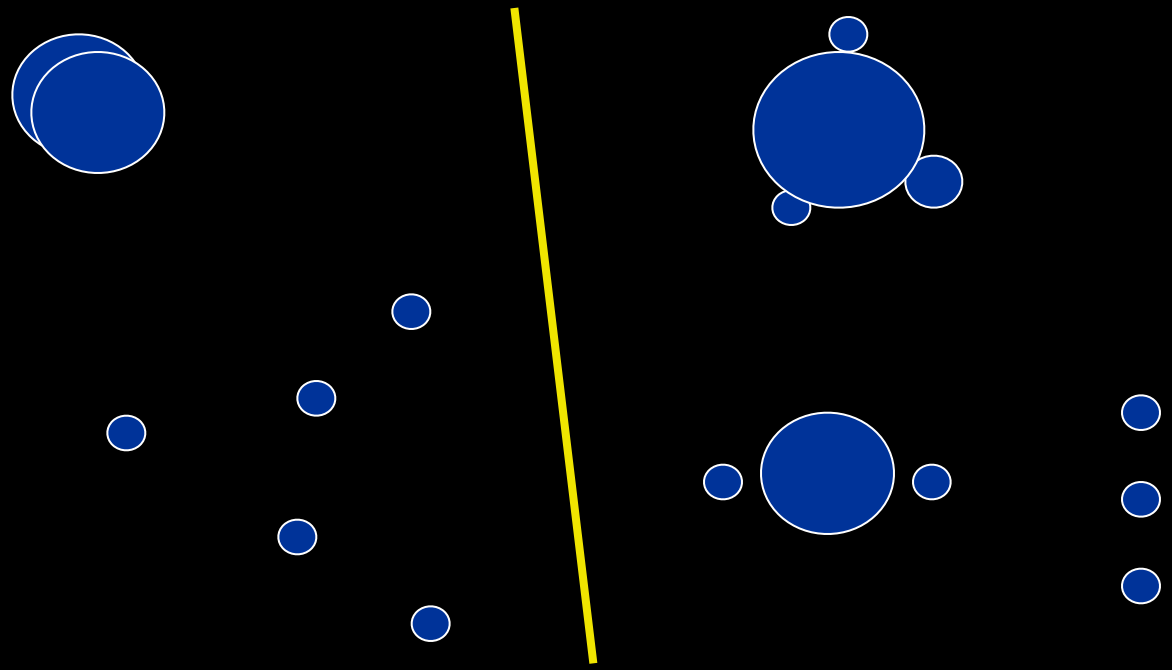
Clustering by Divide & Conquer



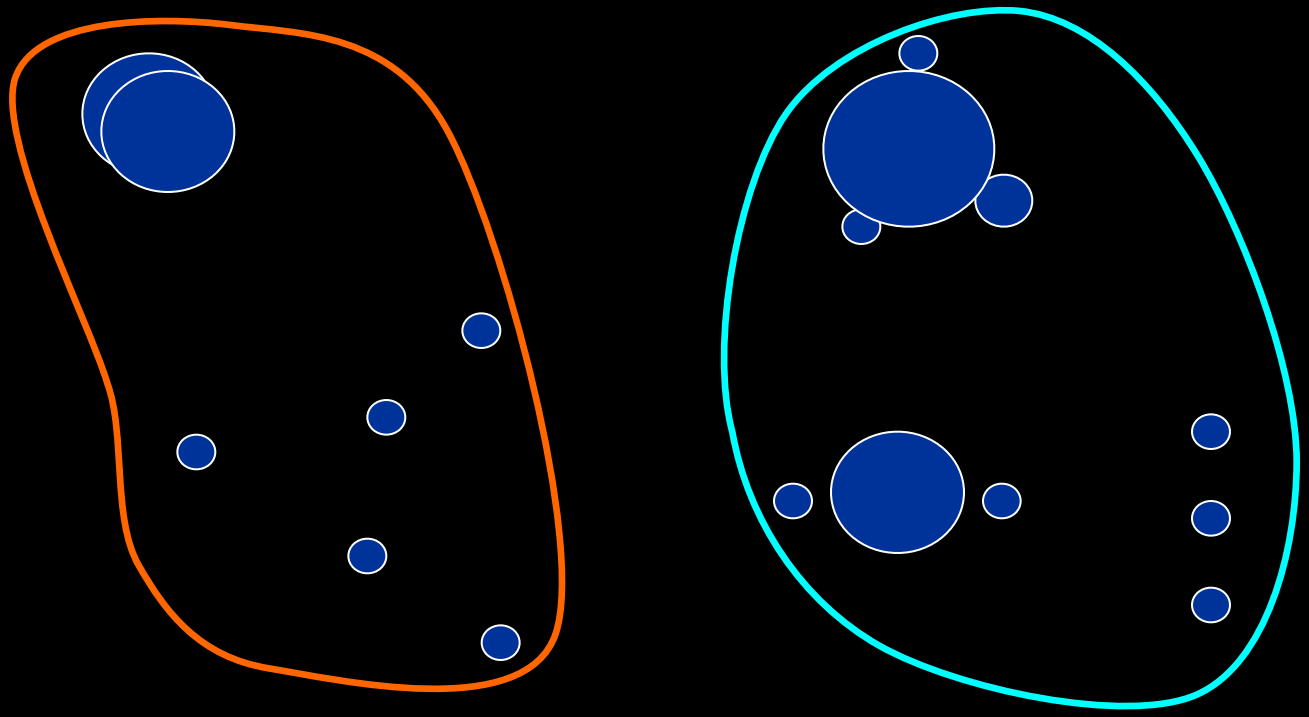
Clustering by Divide & Conquer



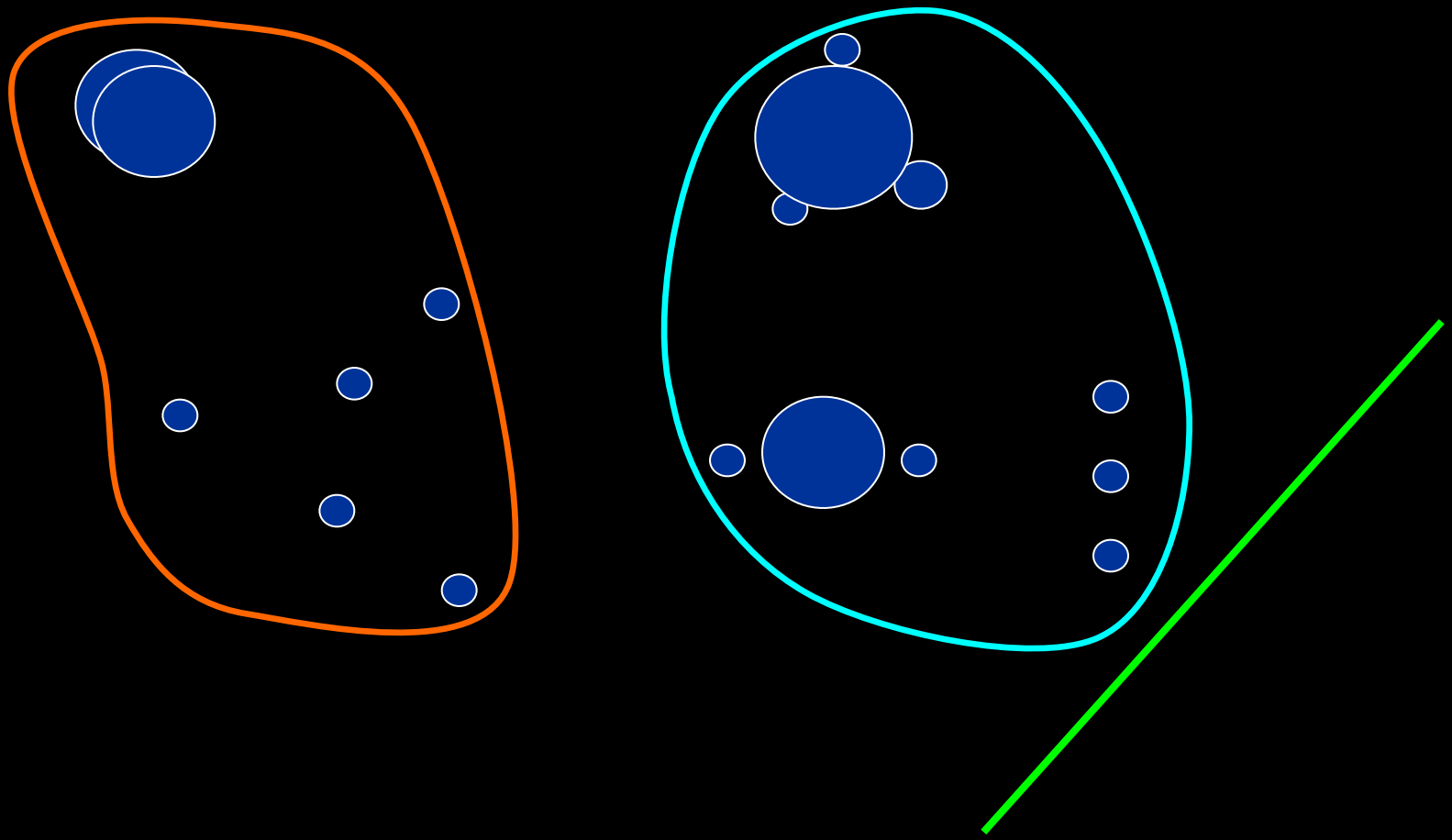
Clustering by Divide & Conquer



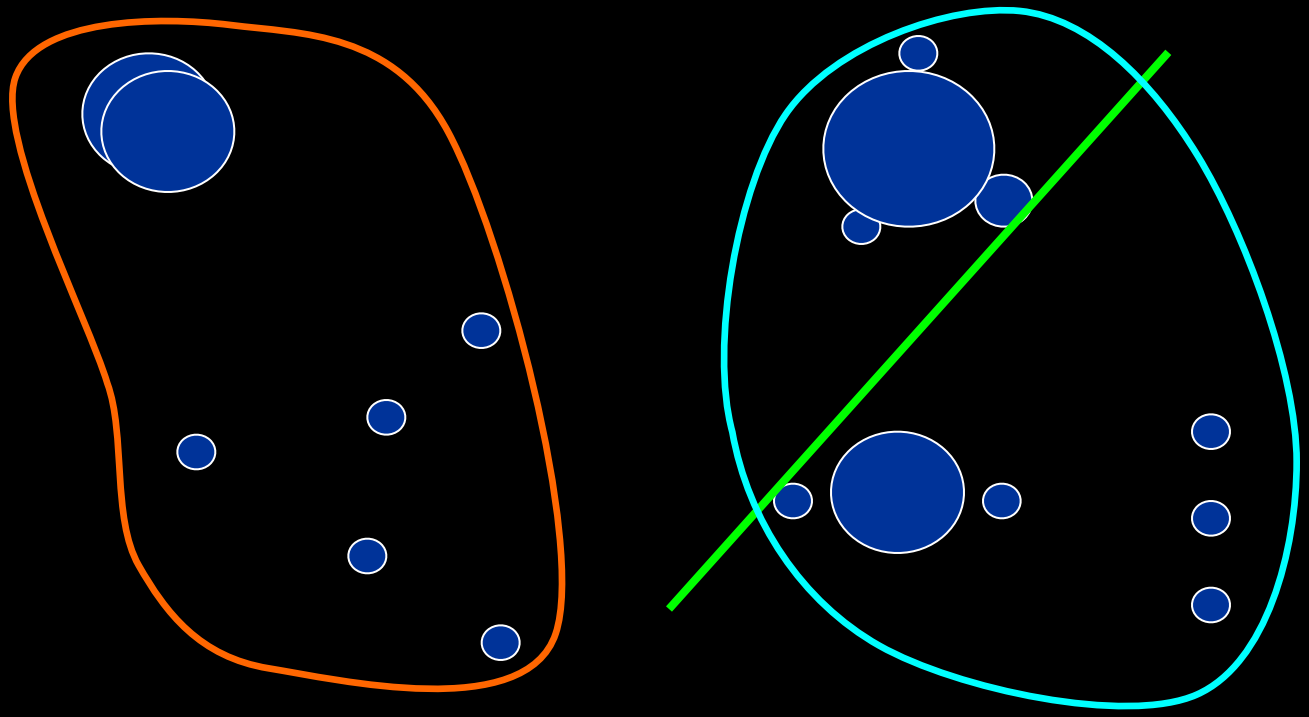
Clustering by Divide & Conquer



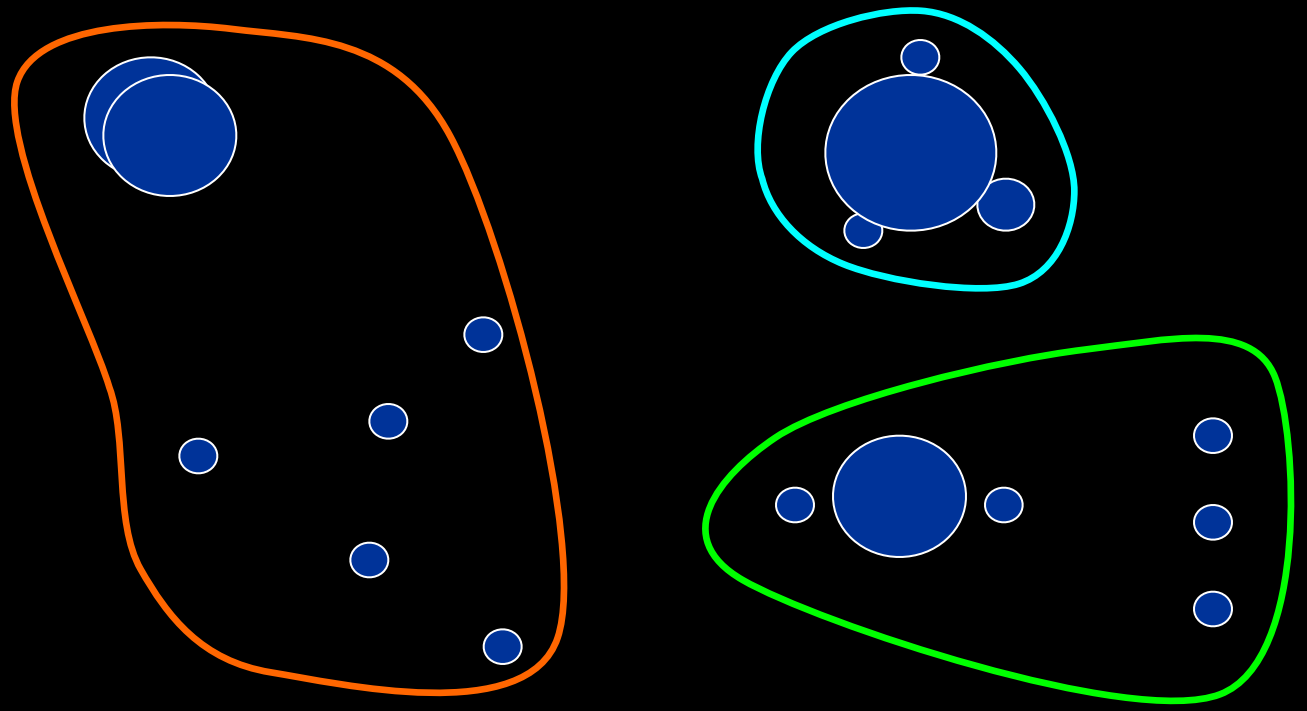
Clustering by Divide & Conquer



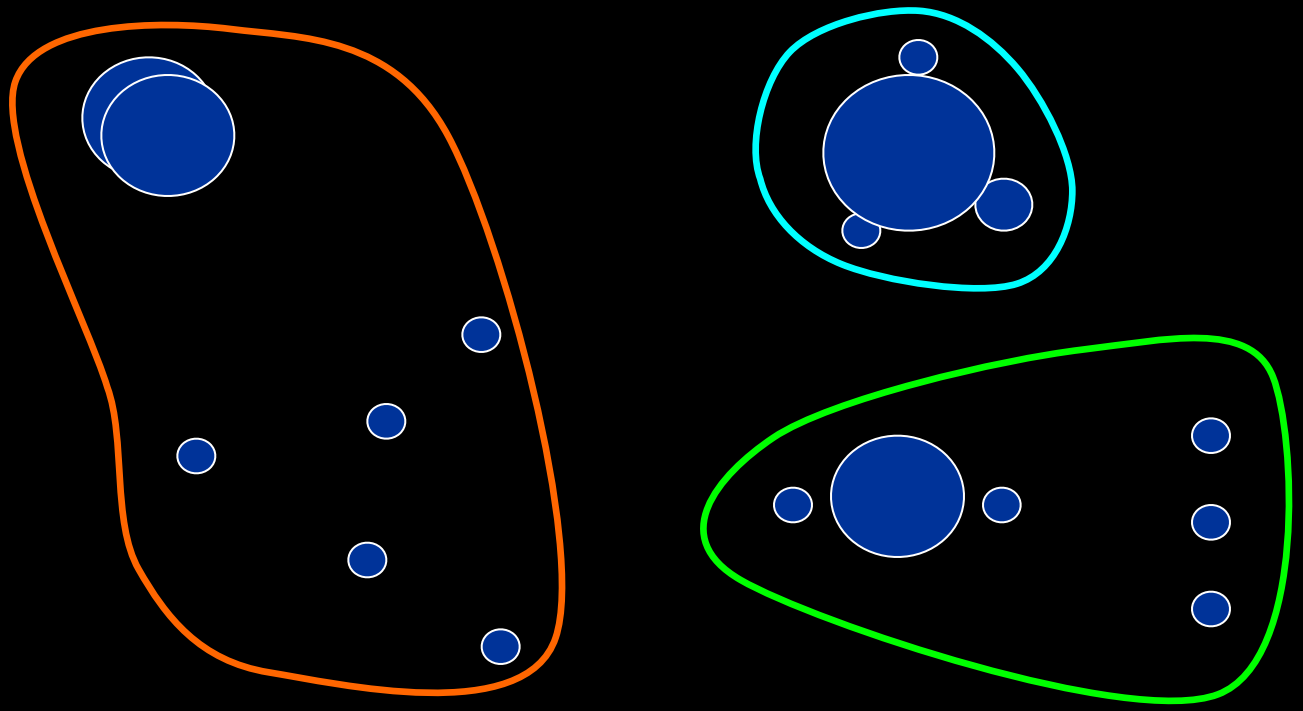
Clustering by Divide & Conquer



Clustering by Divide & Conquer



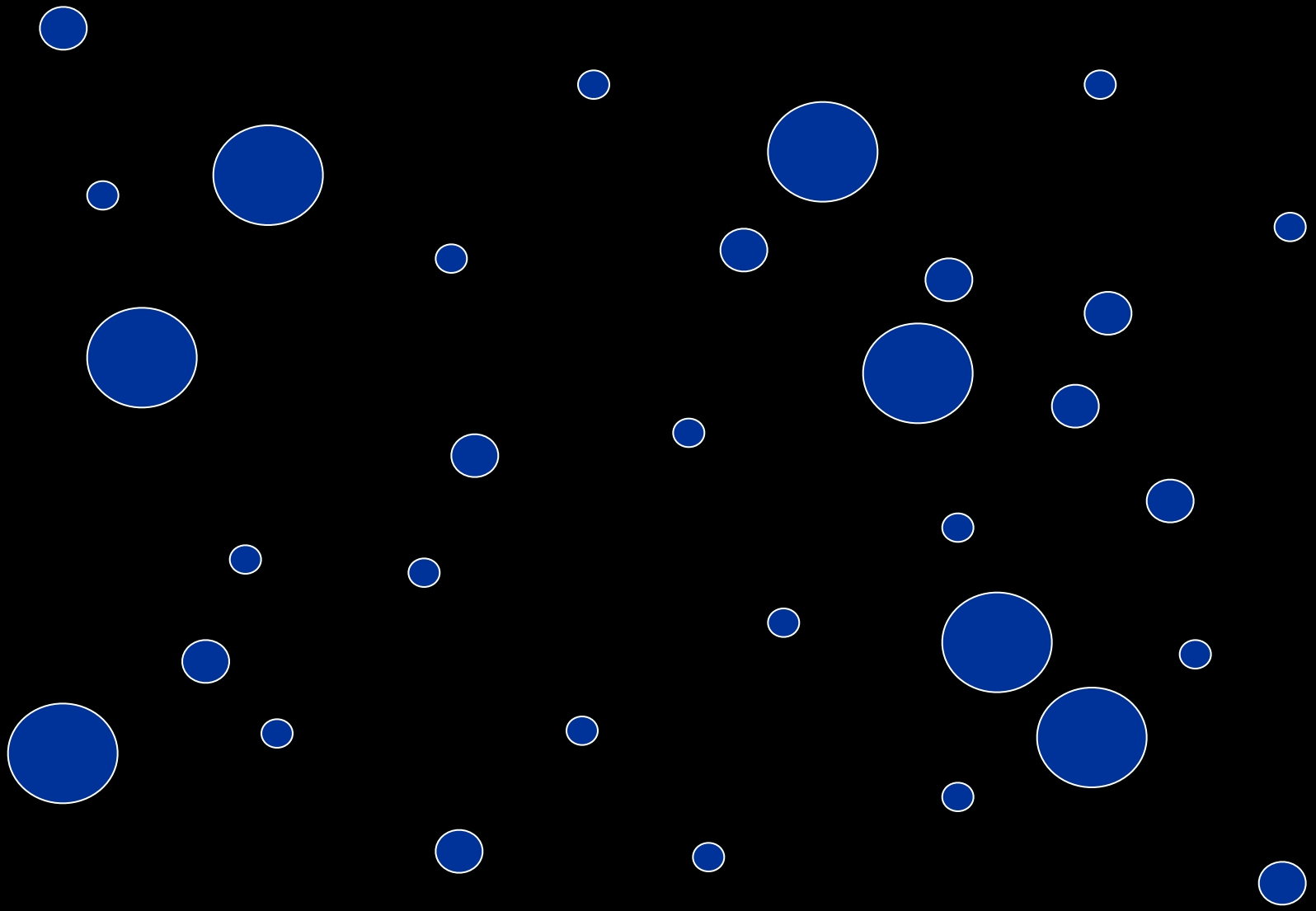
Clustering by Divide & Conquer



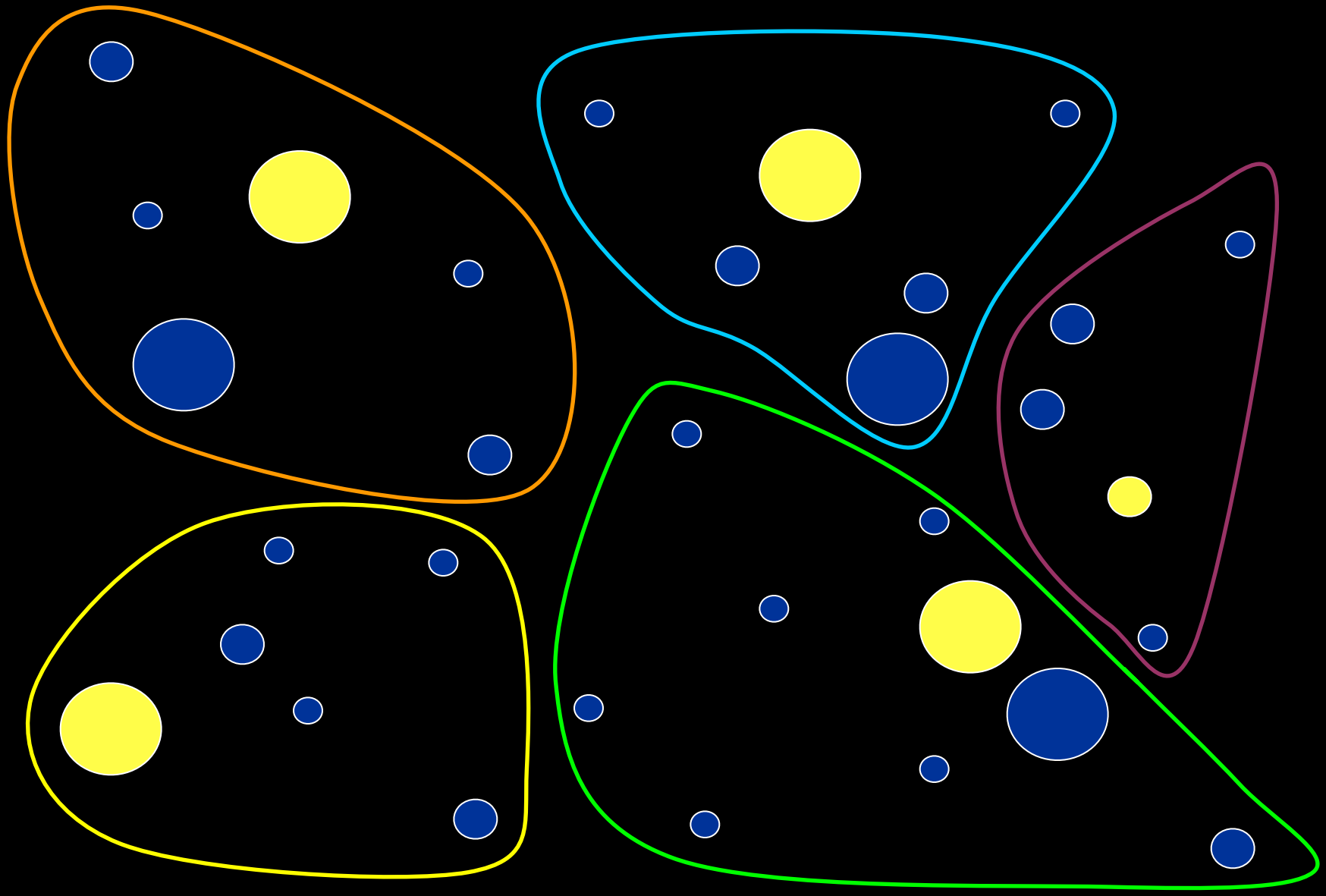
+ Splitting small clusters is fast

– Splitting large clusters is slow

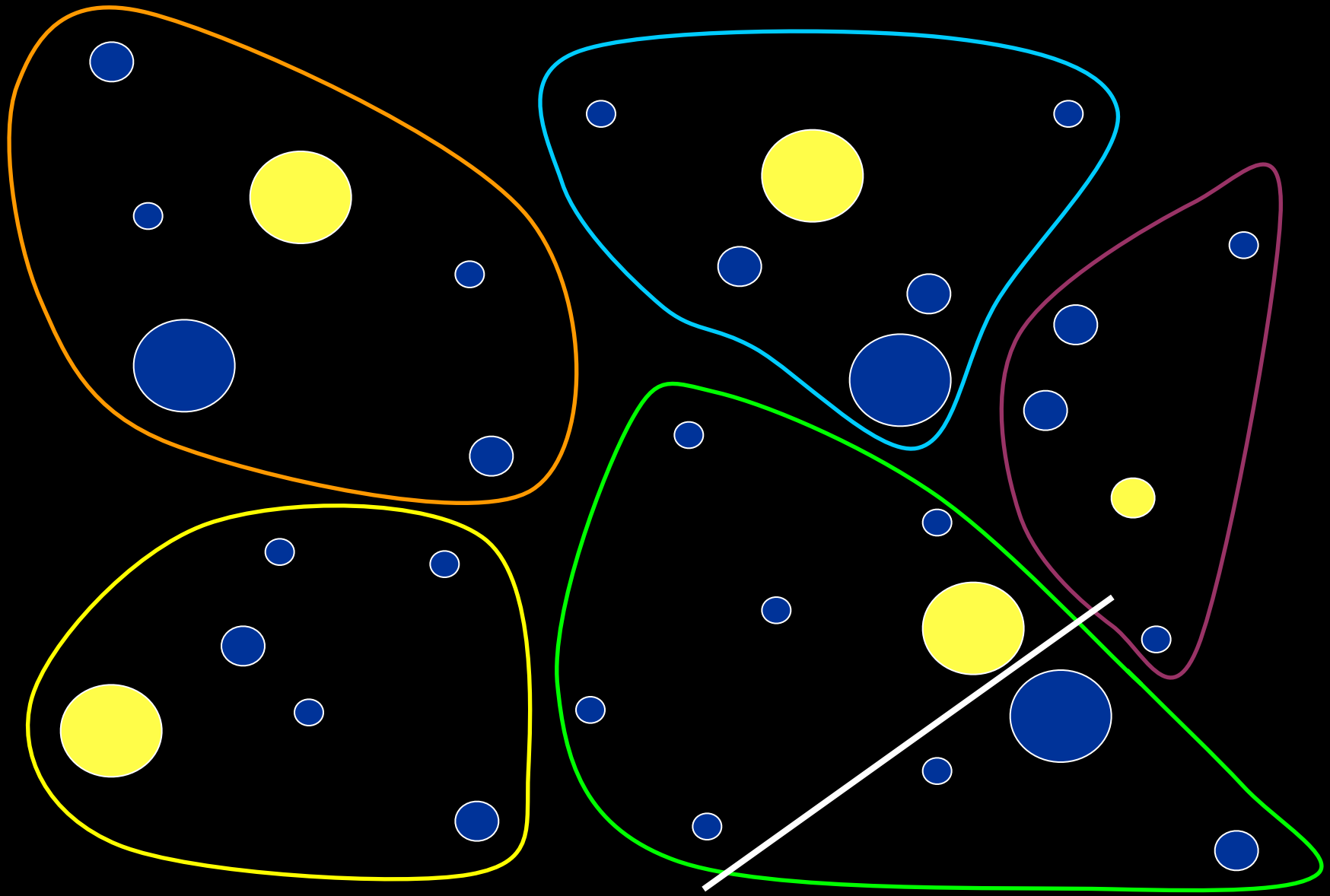
Combined Clustering Algorithm



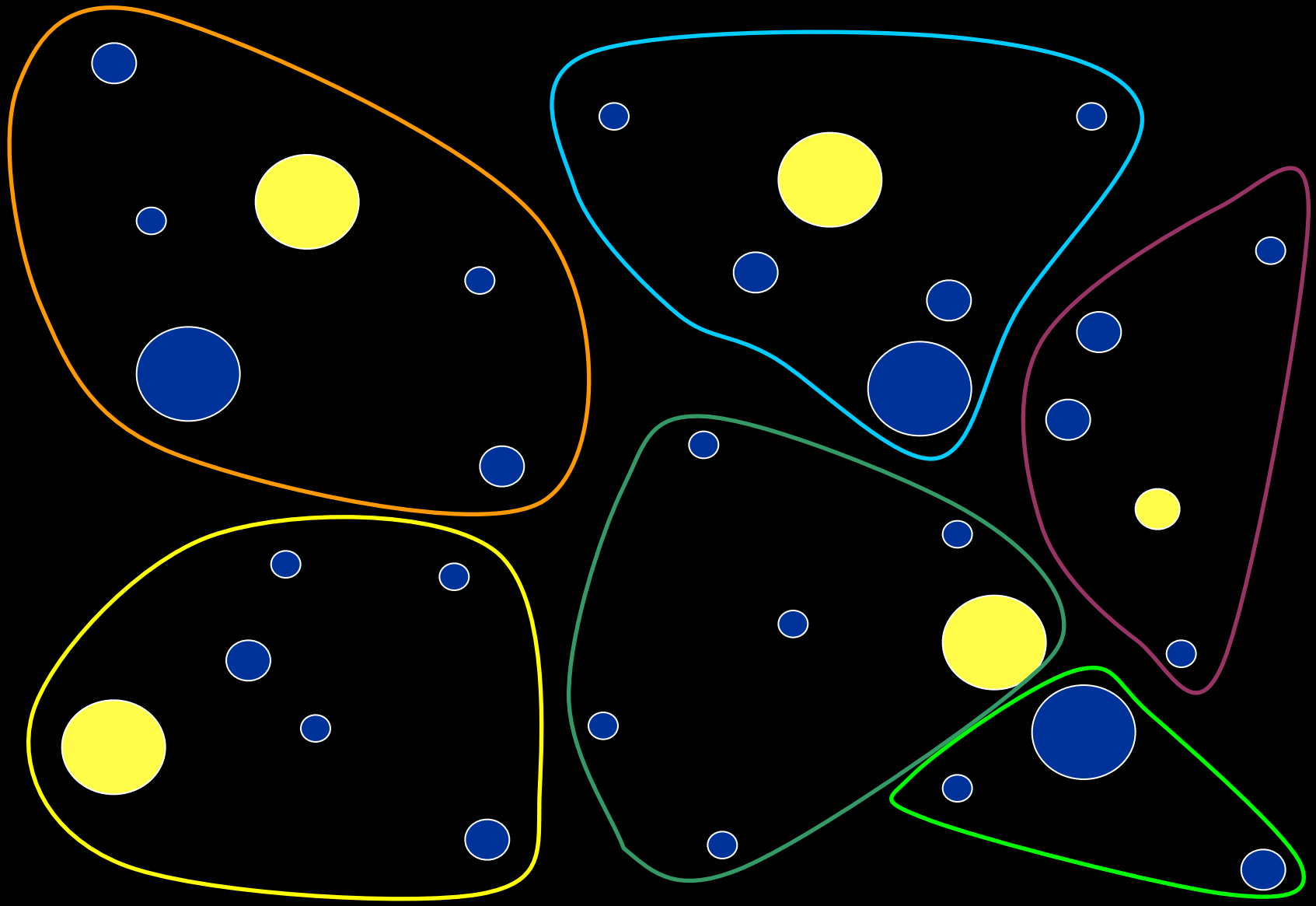
Combined Clustering Algorithm



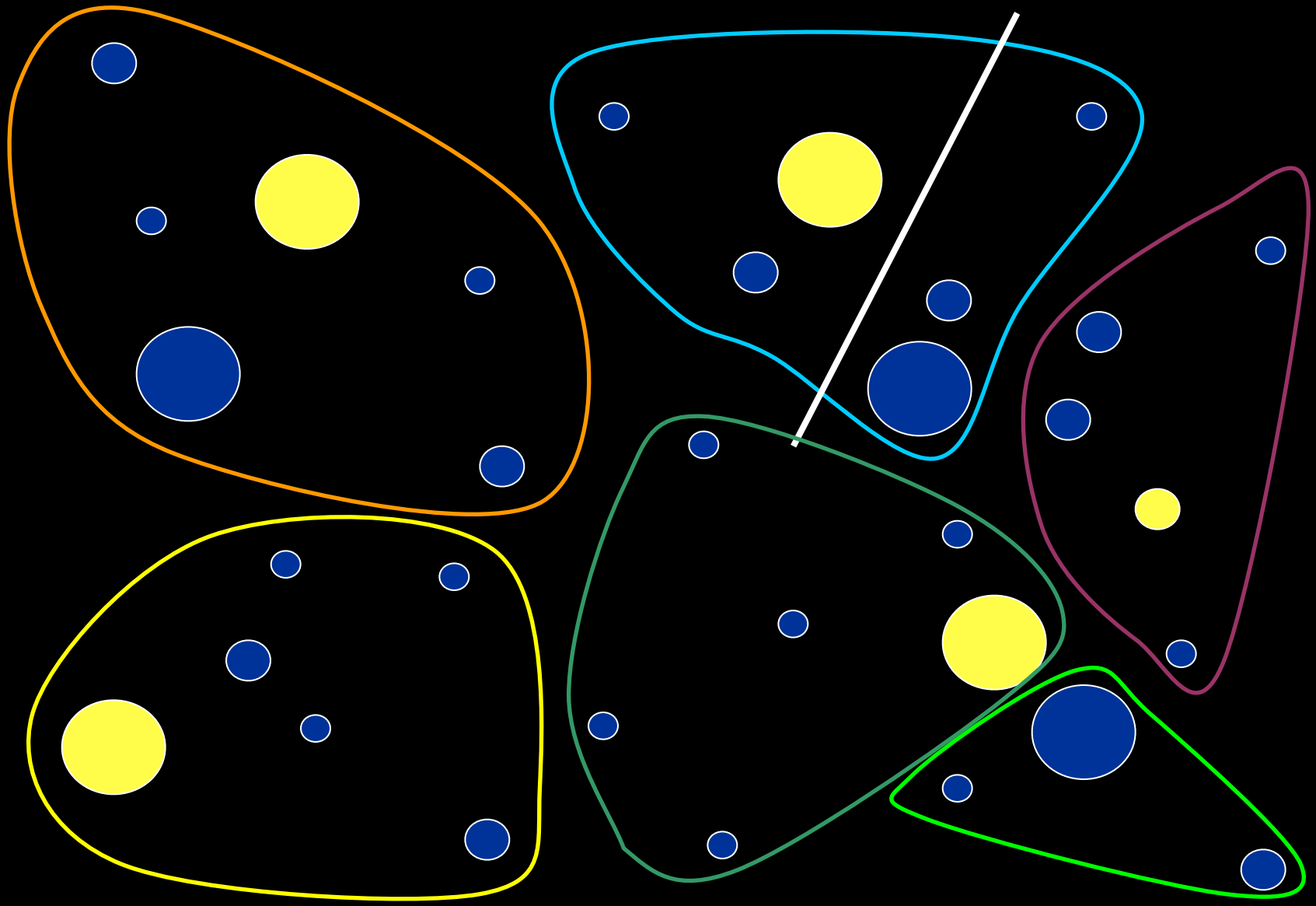
Combined Clustering Algorithm



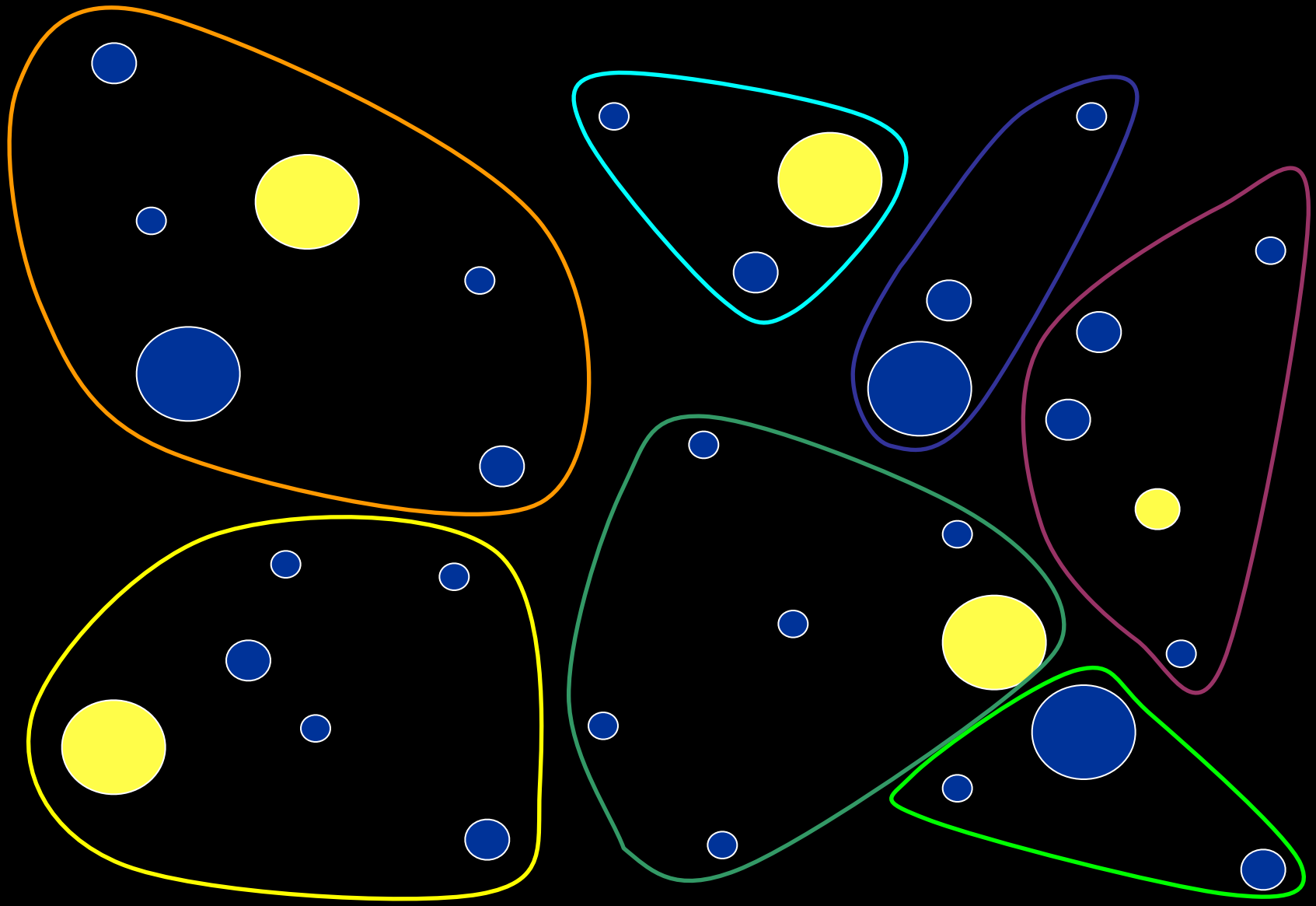
Combined Clustering Algorithm



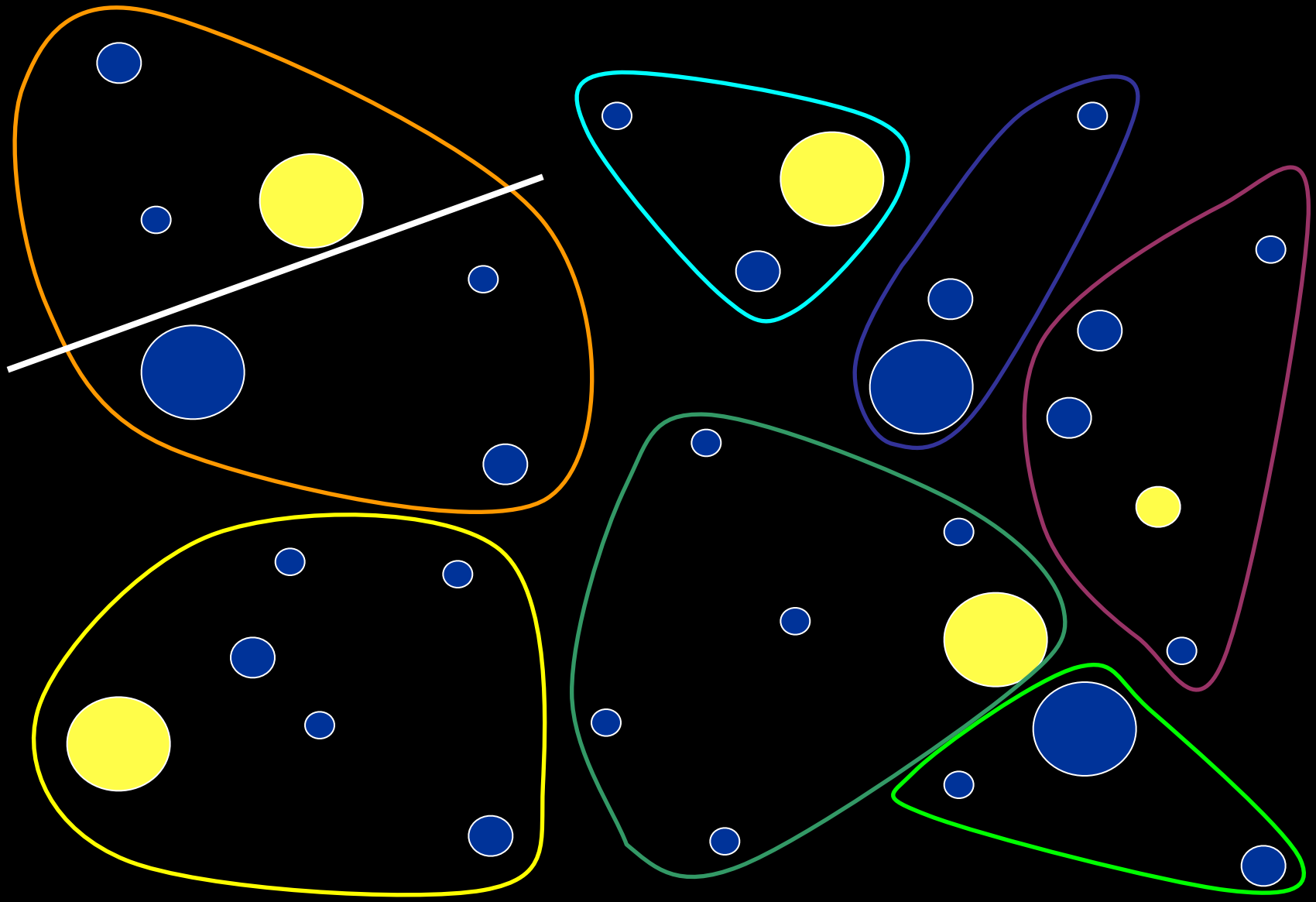
Combined Clustering Algorithm



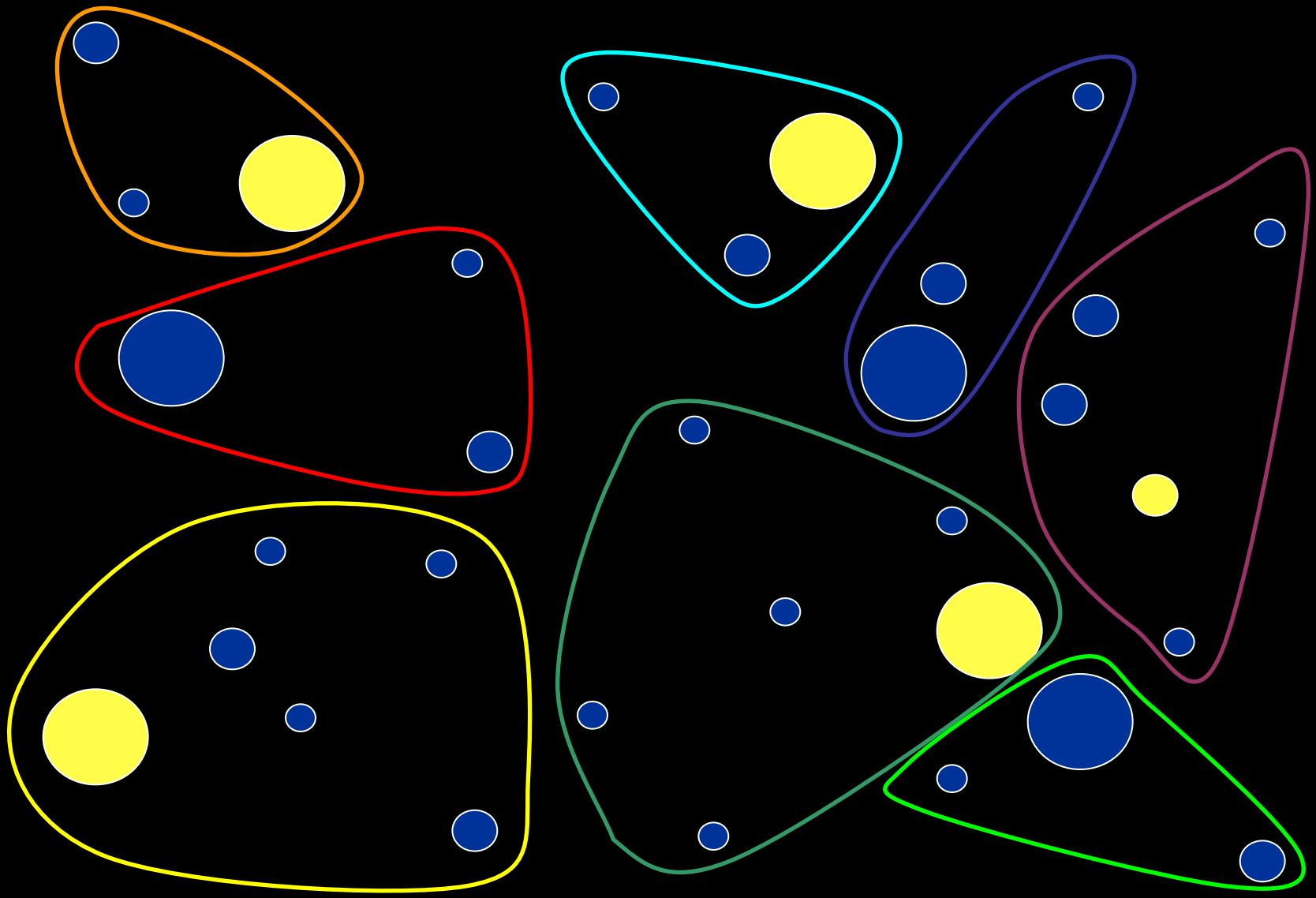
Combined Clustering Algorithm



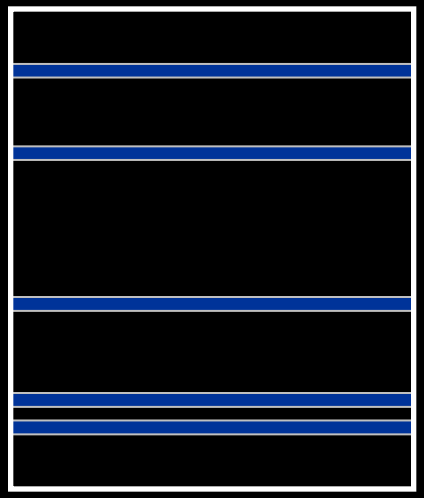
Combined Clustering Algorithm



Combined Clustering Algorithm

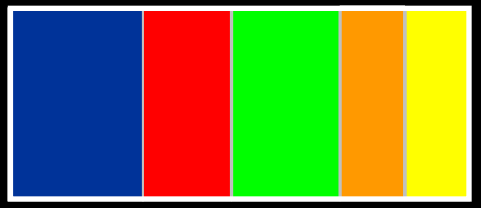


Full Algorithm

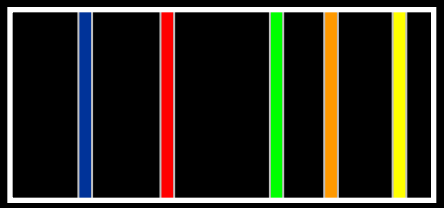


Compute rows (GPU)

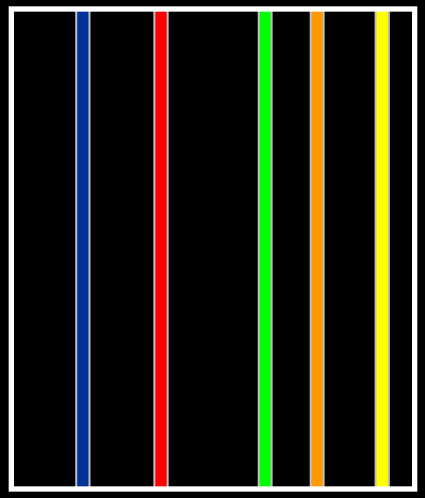
Assemble rows into reduced matrix



Cluster reduced columns



Choose representatives



Compute columns (GPU)

Weighted sum

Results

- We show 5 scenes:


Kitchen


Temple


Trees

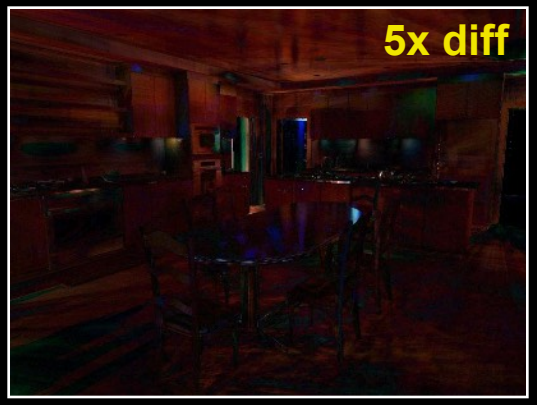

Bunny


Grand Central

- Show reference and 5x difference image
- All scenes have 100,000+ lights
- Timings
  - NVidia GeForce 8800 GTX
  - Light / surface sample creation not included

Results: Kitchen

- 388k polygons
- Mostly indirect illumination
- Glossy surfaces
- Indirect shadows

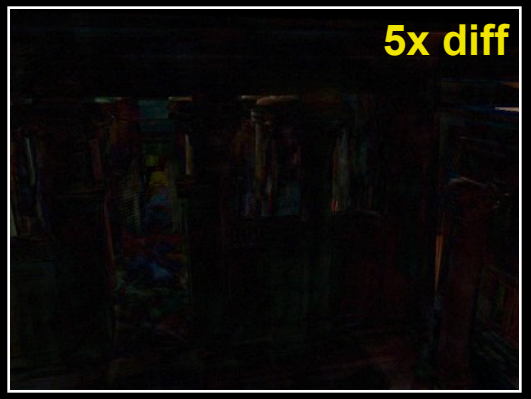


Our result: 13.5 sec
(432 rows + 864 columns)

Reference: 13 min
(using all 100k lights)

Results: Temple

- 2.1m polygons
- Mostly indirect & sky illumination
- Indirect shadows

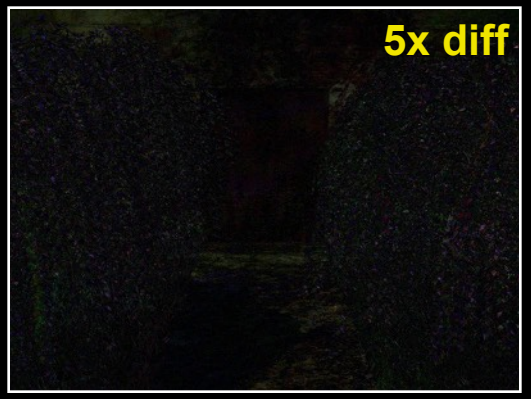


Our result: 16.9 sec
(300 rows + 900 columns)

Reference: 20 min
(using all 100k lights)

Results: Trees

- 328k polygons
- Complex incoherent geometry

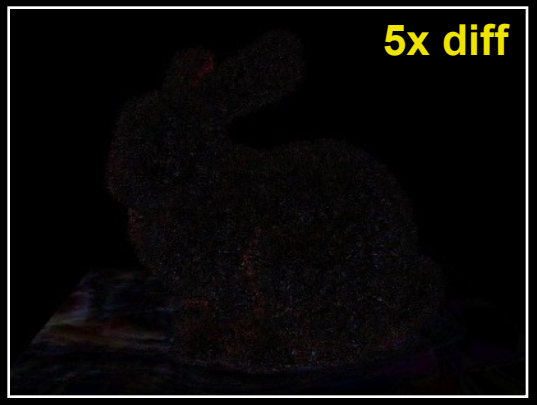


Our result: 2.9 sec
(100 rows + 200 columns)

Reference: 14 min
(using all 100k lights)

Results: Bunny

- 869k polygons
- Incoherent geometry
- High-frequency lighting
- Kajiya-Kay hair shader



Our result: 3.8 sec
(100 rows + 200 columns)

Reference: 10 min
(using all 100k lights)

Results: Grand Central

- 1.5m polygons
- Point lights between stone blocks

Our result: 24.2 sec
(588 rows + 1176 columns)

Reference: 44 min
(using all 100k lights)

The Value of Exploration

Our result

(432 rows + 864 columns)

No exploration

(Using 1455 lights)

Equal time comparison

The Value of Exploration

Our result

No exploration

Equal time comparison:
5x difference from reference

Conclusion

- Fast, high quality approximation for many lights
  - GPU-oriented
  - Sample rows to explore low-rank structure
  - Sample well-chosen columns
- Application: Preview for lighting design
  - Indirect illumination
  - Environment maps
  - Arbitrary lights and shaders

Future Work

- How many rows + columns?
  - Pick automatically
- Row / column alternation
- Progressive algorithm:
  - stop when user likes the image
- Render multiple frames at once?

Row-Column Matrix Sampling

- Visible surfaces are diffuse / low specular
  - Problems with glossy/specular surfaces
  - Extensions to spherical light sources
- Pre computations (for the light sources)

- Light transport operator
  - Part of it: last transport step
  - Operator compression
  - Highly efficient. Why?

Global Illumination II

Global Illumination II