CVPR Tutorial June 20, 2021

CVPR 2021 Tutorial

Physics-Based Differentiable Rendering

Speakers

Shuang Zhao

Assistant Professor
University of California, Irvine

Ioannis Gkioulekas

Assistant Professor
Carnegie Mellon University

Sai Bangaru

Ph.D. student
Massachusetts Institute of Technology

Talk Outline

- Introduction
- Differentiable rendering theory and algorithms
- Differentiable rendering systems and applications
- Q&A

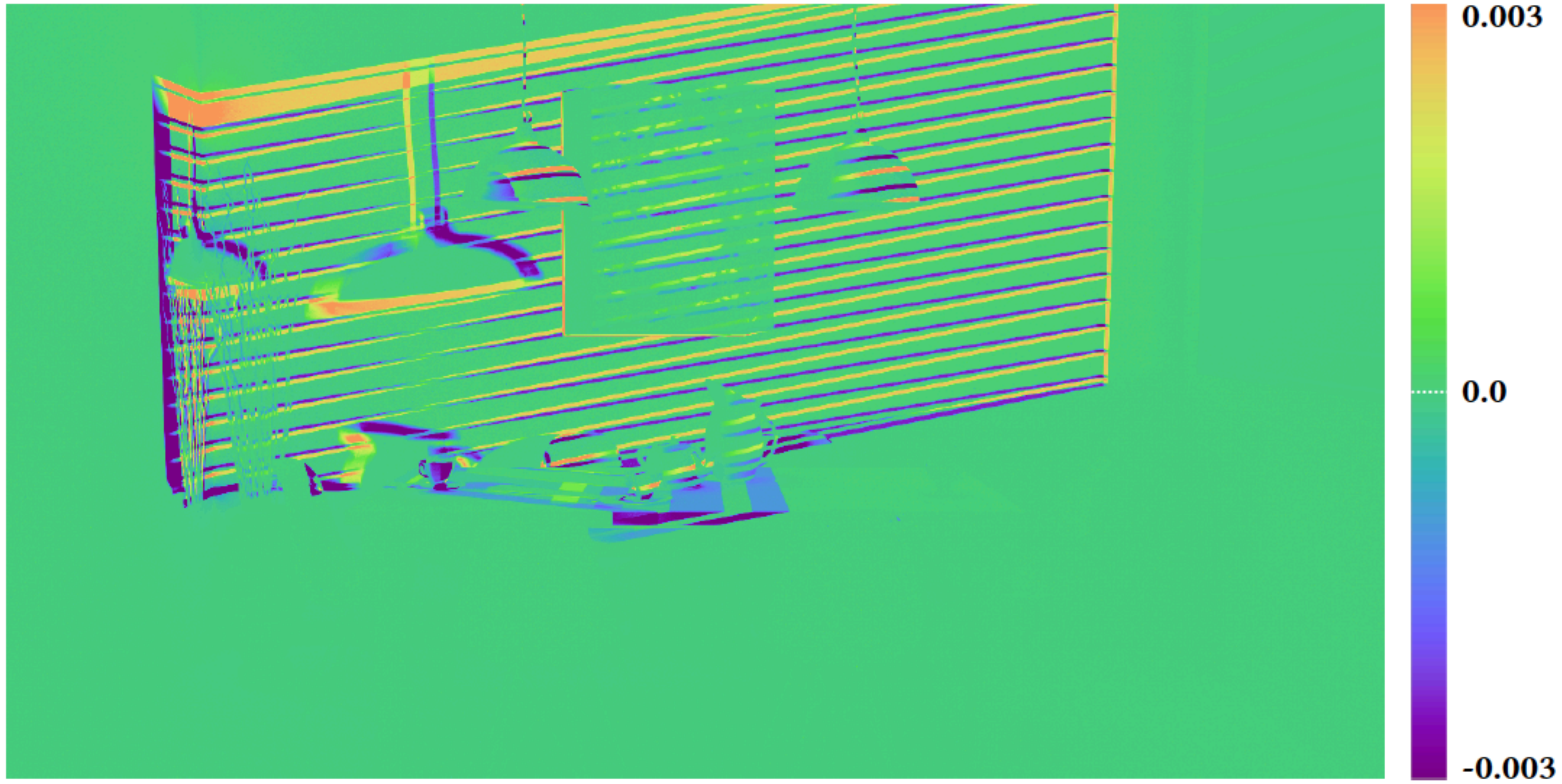
Introduction

What is Differentiable Rendering?

- Computing **derivative** images (with respect to various parameters)

Original

Forward-rendering result



Derivative with respect to sun location

Differentiable-rendering result

Why Use Differentiable Rendering?

- Solving **inverse-rendering** problems
  - i.e., inferring **scene parameters** based on **images** of the scene
- Integrating **forward rendering** into **probabilistic inference** and **machine learning** pipelines
  - e.g., backpropagating losses during training
- Numerous applications in computer vision, computer graphics, computational imaging, VR/AR, …

Forward and Inverse Rendering

Scene parameters

Geometry, materials, lighting, ...

Forward rendering

$$I = \mathscr{R}(\boldsymbol{\theta})$$

Inverse rendering

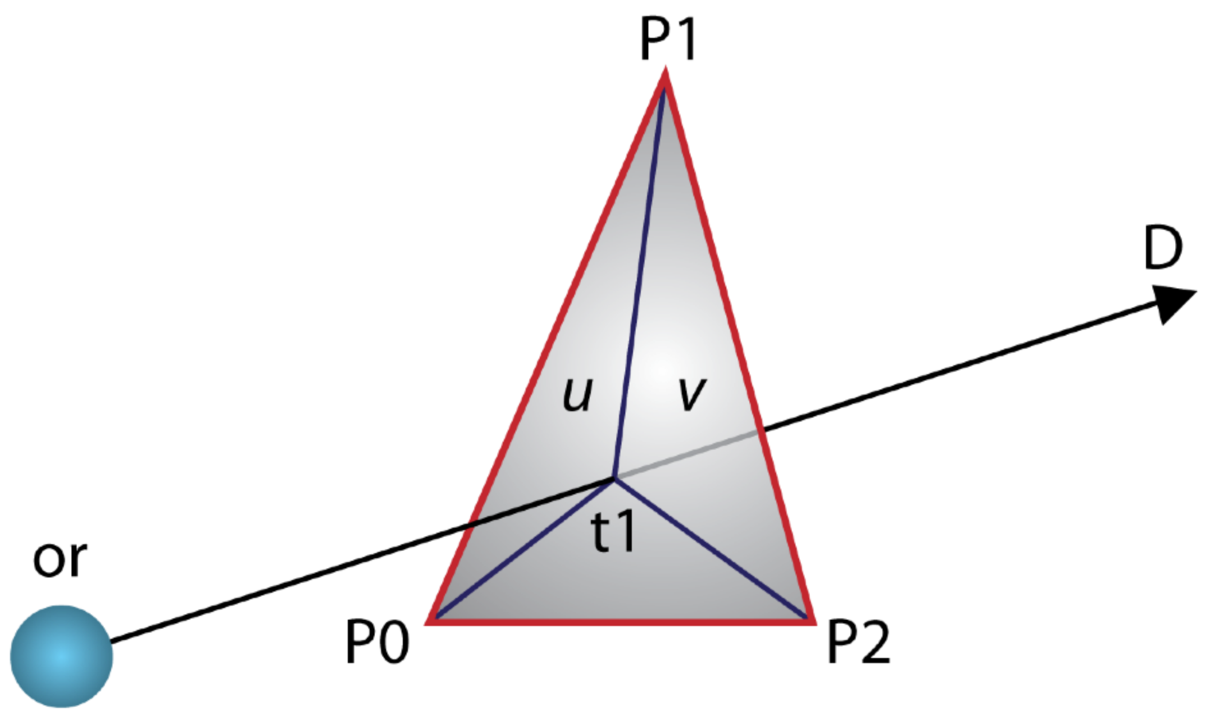
$$\boldsymbol{\theta} = \mathscr{R}^{-1}(I)?$$

Rendered image

Scene: "bed classic" from Jiraniano

Ray Tracing

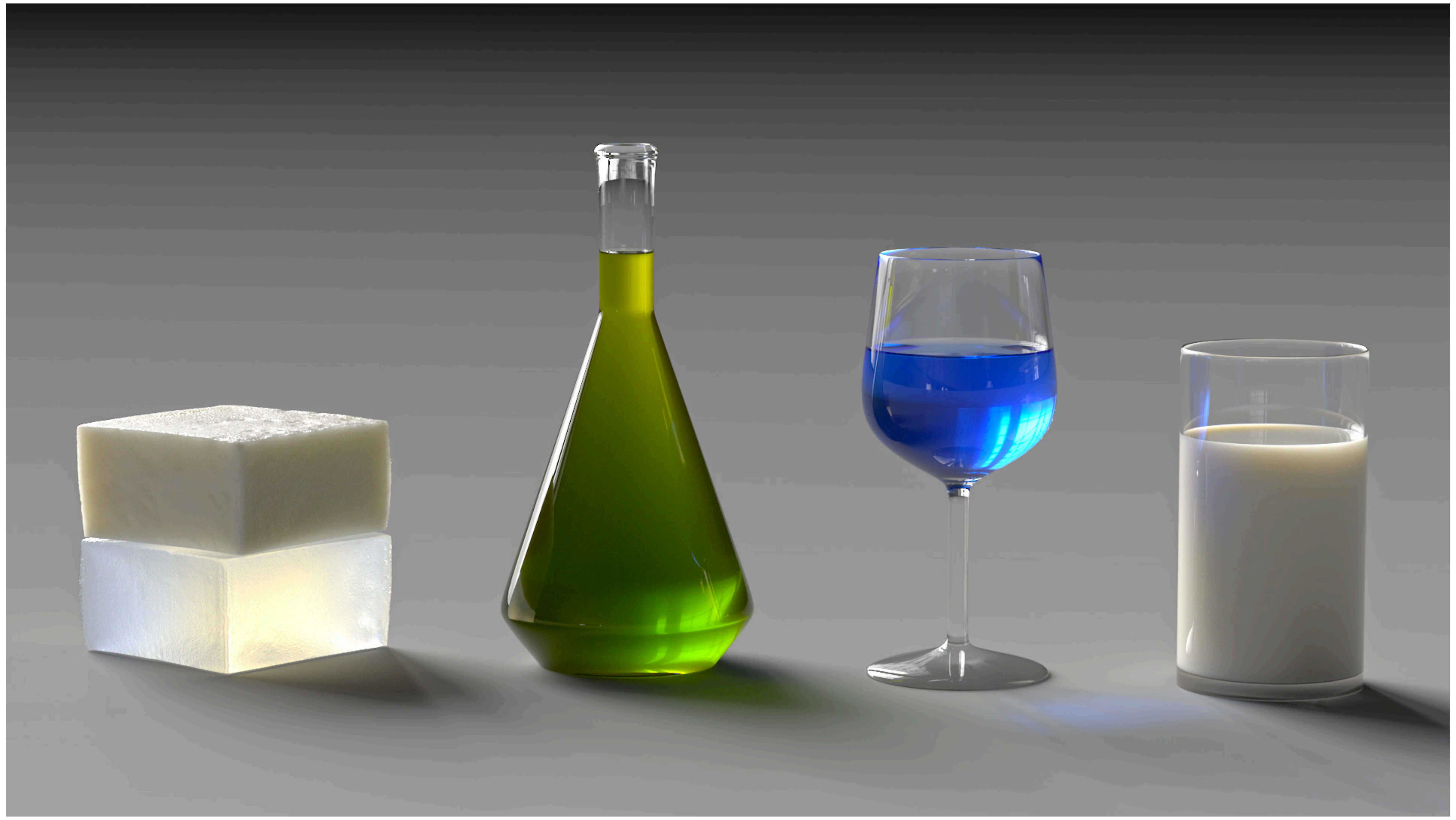
- A heavily abused term in graphics and vision
- We use **ray tracing** to mean ray-surface intersection computations
  - Applicable to both **explicit** (e.g., mesh) and **implicit** (e.g., SDF) surfaces



- *Basic building block* for most (if not all) physics-based rendering algorithms
  - e.g., path tracing, bidirectional path tracing, …

Physics-Based Forward Rendering

- Relies heavily on Monte Carlo integration
- Can capture **complex** light-transport effects
  - Soft shadows, interreflection, subsurface scattering, …



[Gkioulekas et al. 2013]

Physics-Based Inverse Rendering

Scene parameters

Geometry, materials, lighting, …

Inverse rendering

$$\boldsymbol{\theta} = \mathscr{R}^{-1}(\boldsymbol{I})\,?$$

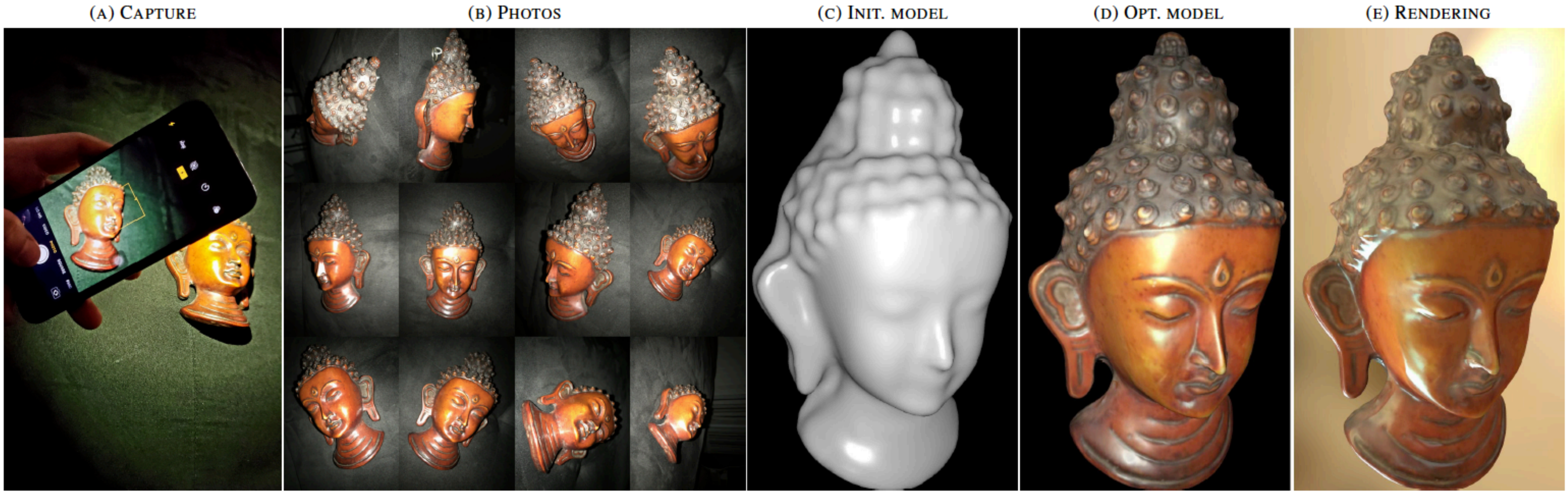
- Inverting **physics-based** forward rendering
- Crucial to many applications

Rendered image

Scene: "bed classic" from Jiraniano

Shape and Material Reconstruction

Joint optimization of *object shape* and *spatially varying reflectance* (our recent work)



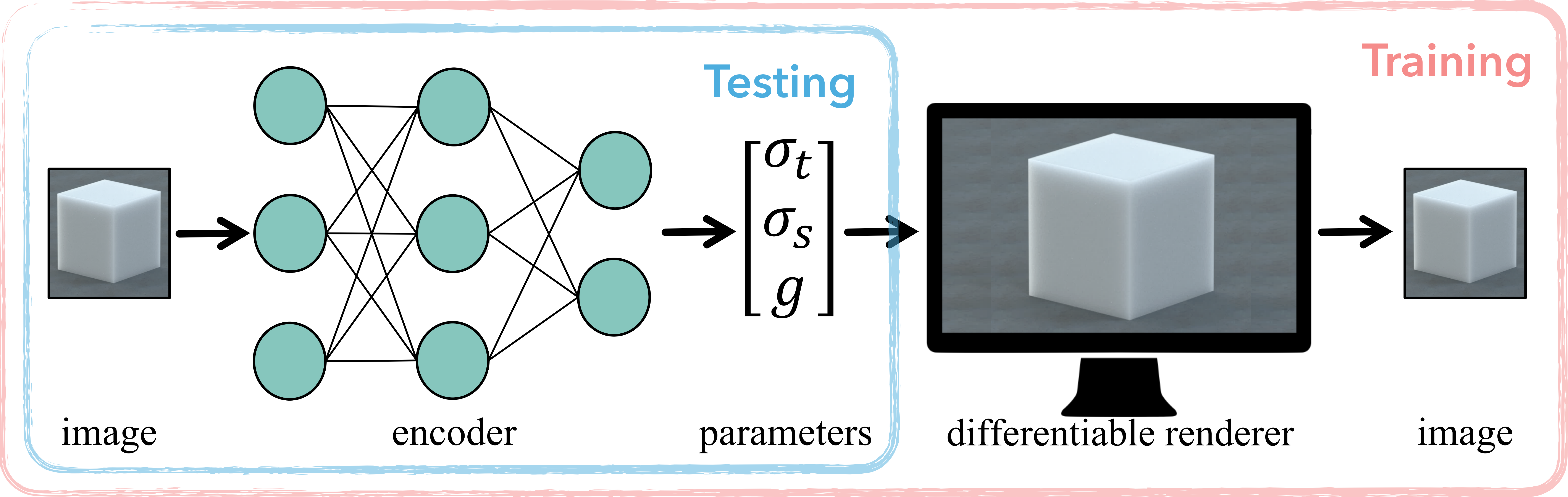
Computational Fabrication

Determining the material configuration for individual voxels in full-color inkjet 3D printing

[Nindel et al. 2021]

Physics-Based Learning

- Integrating physics-based rendering into **machine learning** and **probabilistic inference** pipelines
- Inverse subsurface scattering [Che et al. 2020]



- Utilizing *image loss* provided by a volume path tracer to regularize training
- Use the trained encoder to solve inverse problems during testing

Why is Physics-Based Differentiable Rendering Hard?

- Need to differentiate solutions of integral equations (or path integrals)
  - e.g., the rendering equation: $L(\boldsymbol{x}, \boldsymbol{\omega}_\mathrm{o}) = \int_{\mathbb{S}^2} f_\mathrm{s}(\boldsymbol{x}, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) \, L_\mathrm{i}(\boldsymbol{x}, \boldsymbol{\omega}_\mathrm{o}) \, \mathrm{d}\boldsymbol{\omega}_\mathrm{i} + L_\mathrm{e}(\boldsymbol{x}, \boldsymbol{\omega}_\mathrm{o})$
  - The relation between such solutions and scene parameters can be highly complex
- Requires handling very large gradient matrices (e.g., with 10^{12} or more entries)
- Can be tricky to implement correctly

Handling Many Parameters

- Forward-rendering function: $\boldsymbol{I} = \mathscr{R}(\boldsymbol{\theta})$
  - $\boldsymbol{\theta} \in \mathbb{R}^n$ (n: number of parameters)
  - $\boldsymbol{I} \in \mathbb{R}^m$ (m: number of pixels)
- Gradient matrix: $\dfrac{\mathrm{d}\mathscr{R}}{\mathrm{d}\boldsymbol{\theta}}(\boldsymbol{x}) \in \mathbb{R}^{m \times n}$
- Challenges:
  - m and n can both be large ($\sim 10^6$)
  - $(\mathrm{d}\mathscr{R}/\mathrm{d}\boldsymbol{\theta})$ can involve 10^{12} entries
  - Reverse-mode automatic differentiation can easily run out of memory

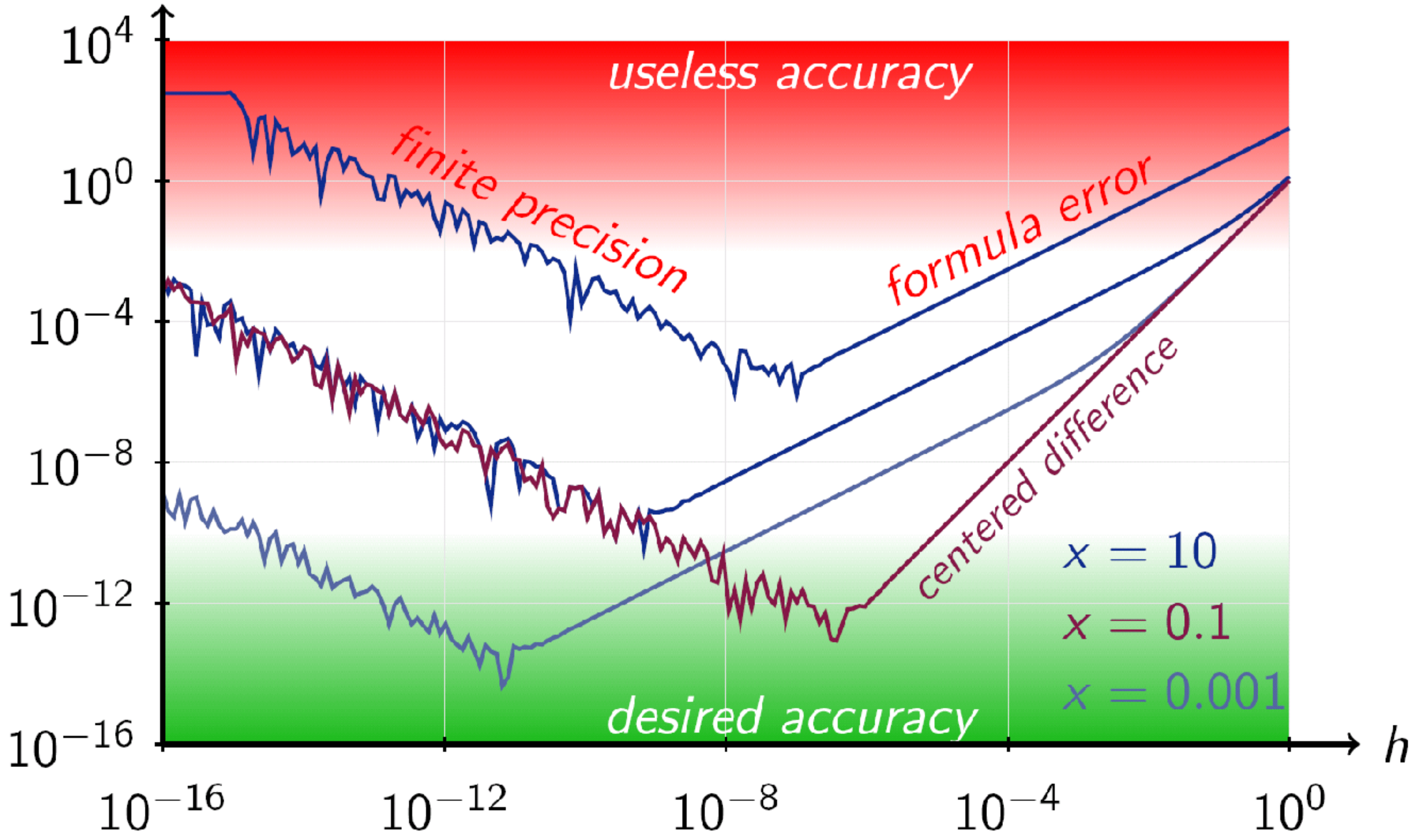
Precautions Must Be Taken

- Precautions must be taken to ensure **correctness**
  - E.g., applying automatic differentiation to a path tracer does not always work
- Should the PDF (used by a Monte Carlo estimator) be differentiated?
  - Can go either way…
    (More on this later.)
- Discontinuities
  - Differentiating only the integrand is insufficient
    (More on this later.)

Why Not Simply Use Finite Differences?

Finite difference:

$$\frac{\partial \mathscr{R}}{\partial \theta_i}(\boldsymbol{\theta}) \approx \frac{\mathscr{R}(\boldsymbol{\theta} + \varepsilon \boldsymbol{e}_i) - \mathscr{R}(\boldsymbol{\theta} - \varepsilon \boldsymbol{e}_i)}{2\varepsilon}$$



[Wikipedia]

Potential problems:

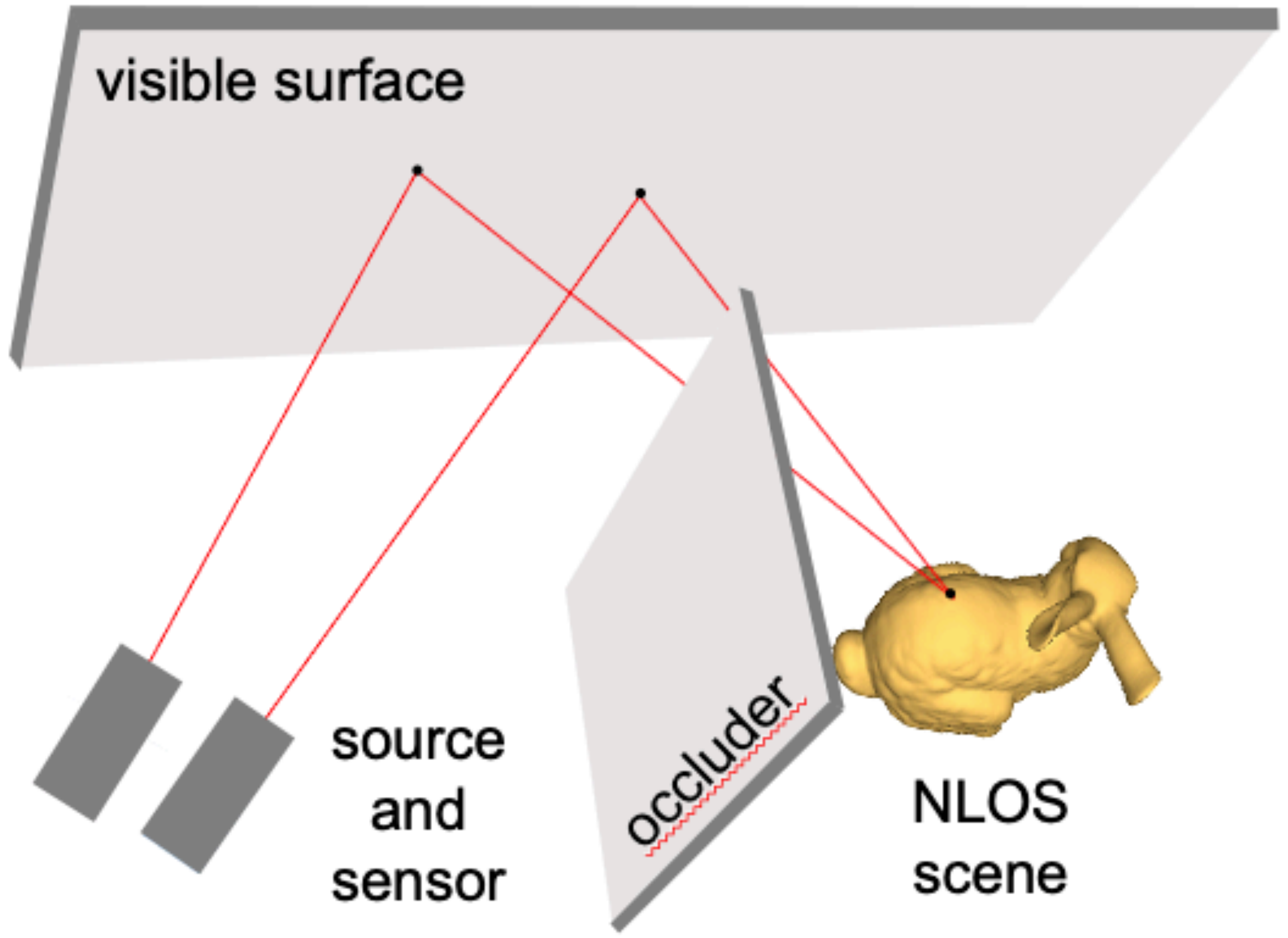
- High bias (large ε), rounding error (small ε)
- Need to correlate Monte Carlo samples
- Scales poorly with the number of parameters

Global Illumination

- Can be simulated with modern differentiable renderers
- Required when solving many inverse-rendering problems

[theawesomer.com]

Computational fabrication



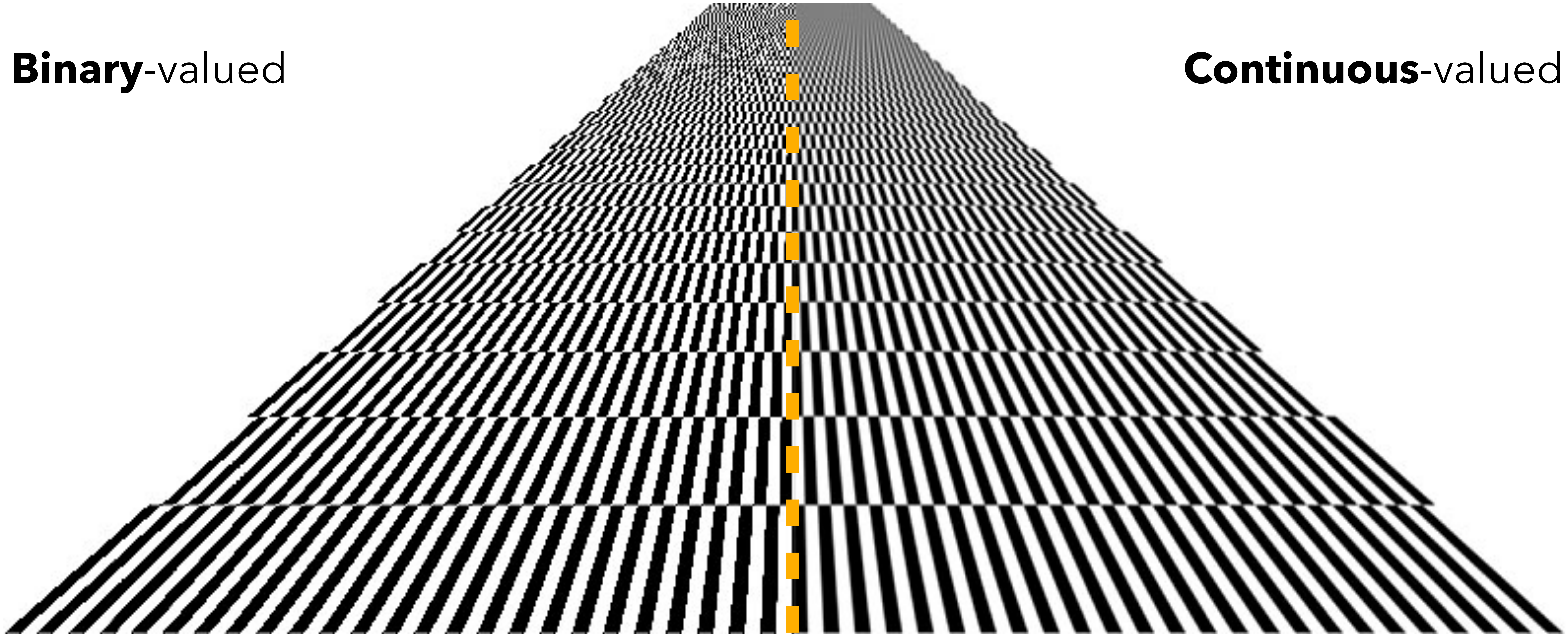
[Tsai et al. 2019]

Non-line-of-sight imaging

Pixel-Level Antialiasing Matters

Binary-valued

Continuous-valued

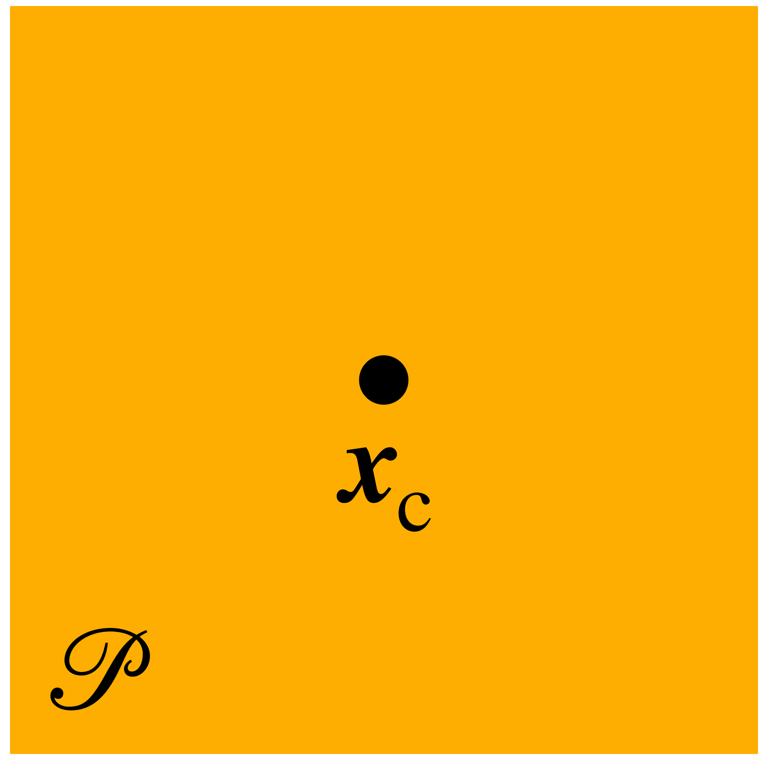


[zone-10.com]

No antialiasing

Perfect antialiasing

Pixel value = $I(\boldsymbol{x}_{\mathrm{c}})$



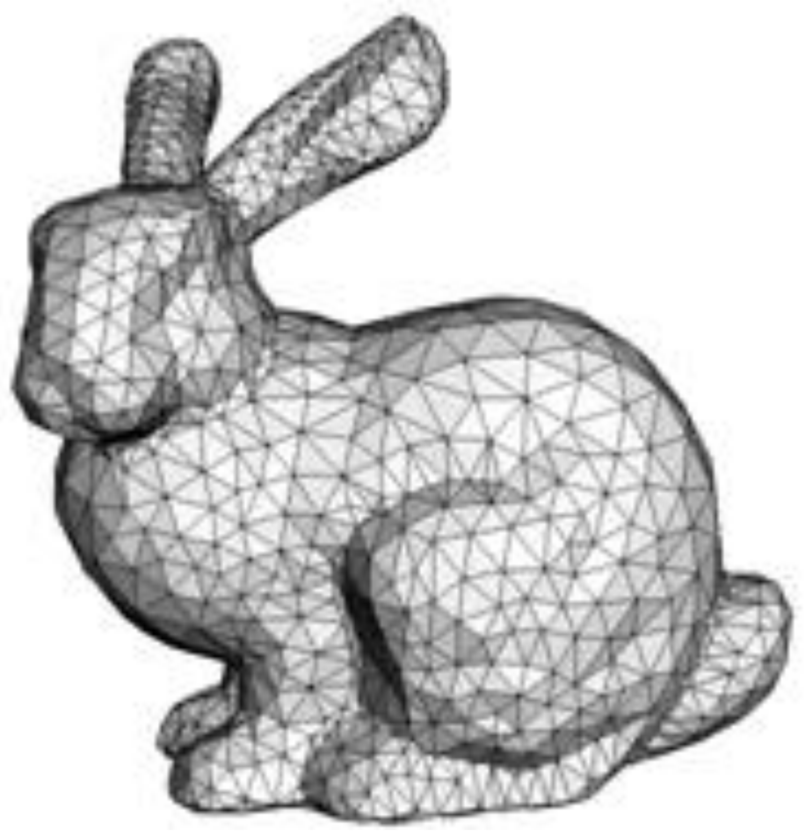
One pixel

$$\boxed{\text{Pixel value} = \frac{1}{|\mathscr{P}|}\int_{\mathscr{P}} I(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}}$$

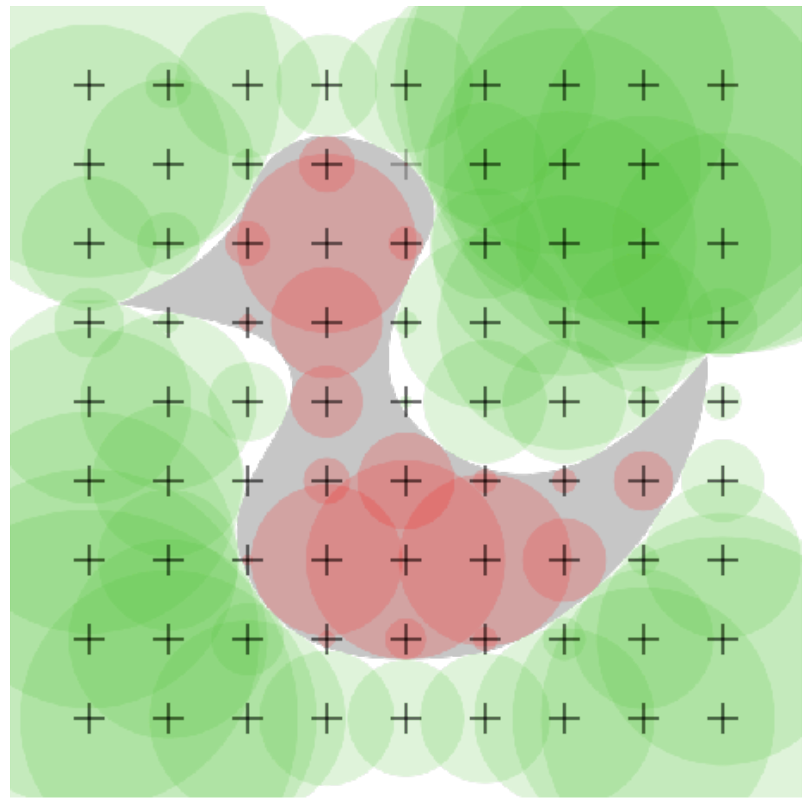
More information, more differentiable!

Can make inverse-rendering optimizations more robust

Geometric Representations



Explicit
(e.g., polygonal meshes)



Implicit
(e.g., signed distance functions)

- *Ray-tracing-based* **forward** rendering is agnostic to geometric representations
- The situation is more complex for **differentiable** rendering
  - Due to the need to handle discontinuities (will discuss in details later)

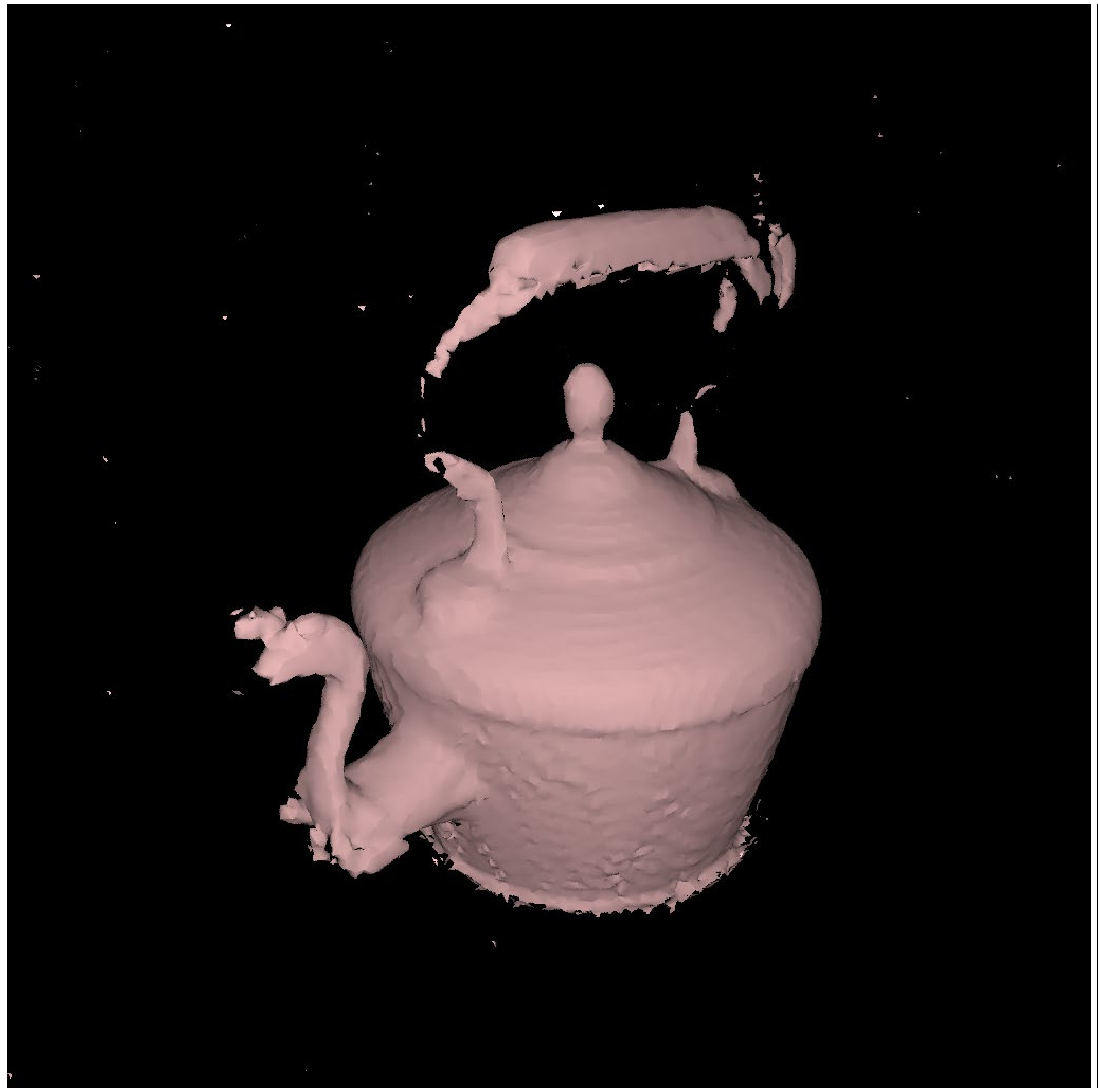
Why you should use ray-tracing-based differentiable rendering

Ray Tracing vs. Rasterization

- We believe that **ray tracing** is the way to go for future **differentiable** renderers
- Ray-tracing-based methods are **not** much slower than rasterization
  - Hardware-accelerated ray tracing has been improving rapidly (e.g., Nvidia RTX)
  - Visibility checks and intersections are typically not the performance bottleneck

Ray Tracing vs. Rasterization

23823 vertices, 44702 faces



Initial

Target

1024x1024 at 2 spp (Titan RTX) **differentiable** render time:

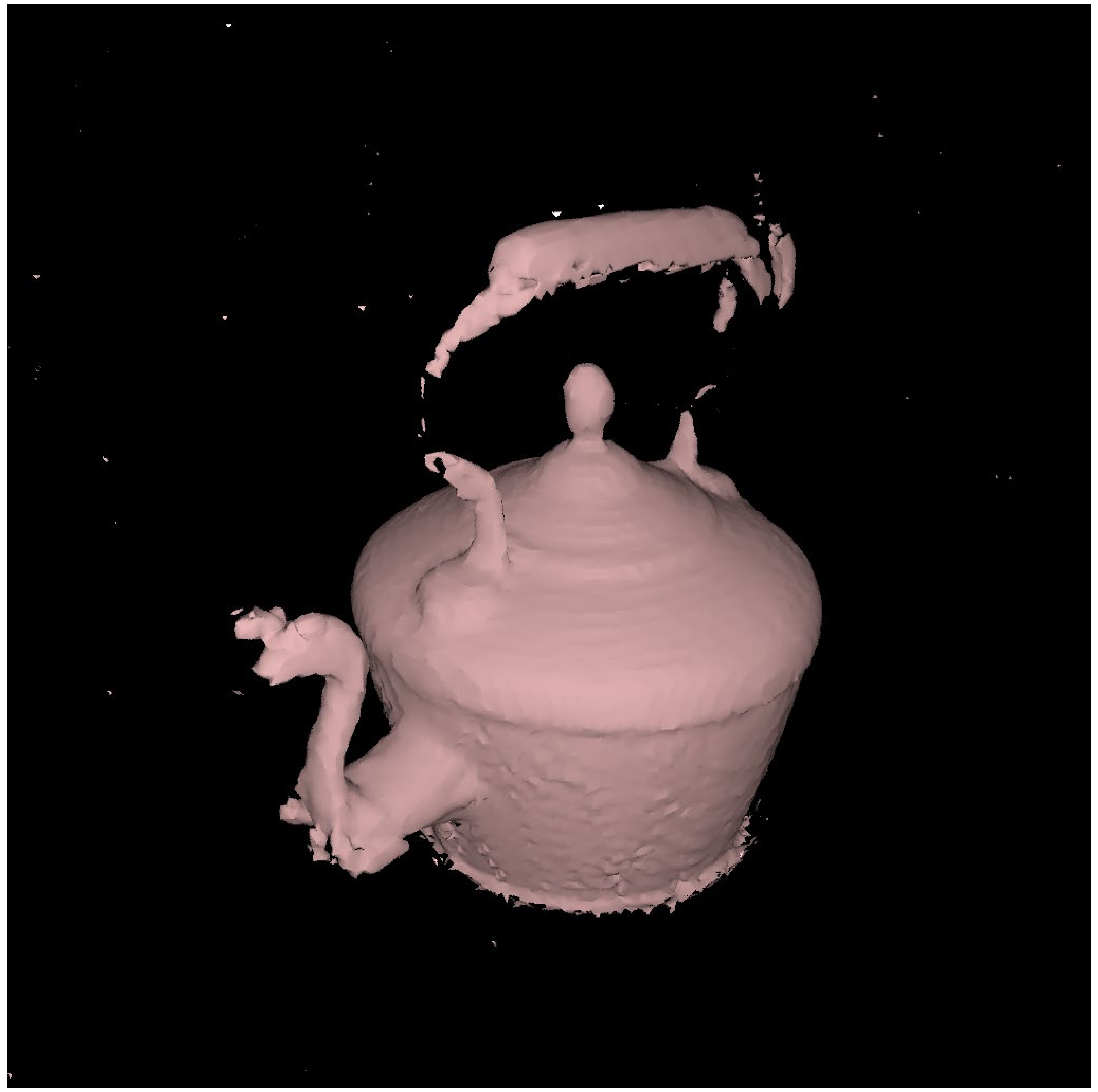
- **psdr-cuda** (ray-tracing-based)*: 2.8 msec
- **PyTorch3D** (soft rasterizer): 52.5 msec

Other computations (loss backpropagation, mesh evolution and remeshing):
~ 1000 msec

*Luan et al., EGSR 2021 *(to appear)*

Ray Tracing vs. Rasterization

23823 vertices, 44702 faces



Initial

Optimized (psdr-cuda)

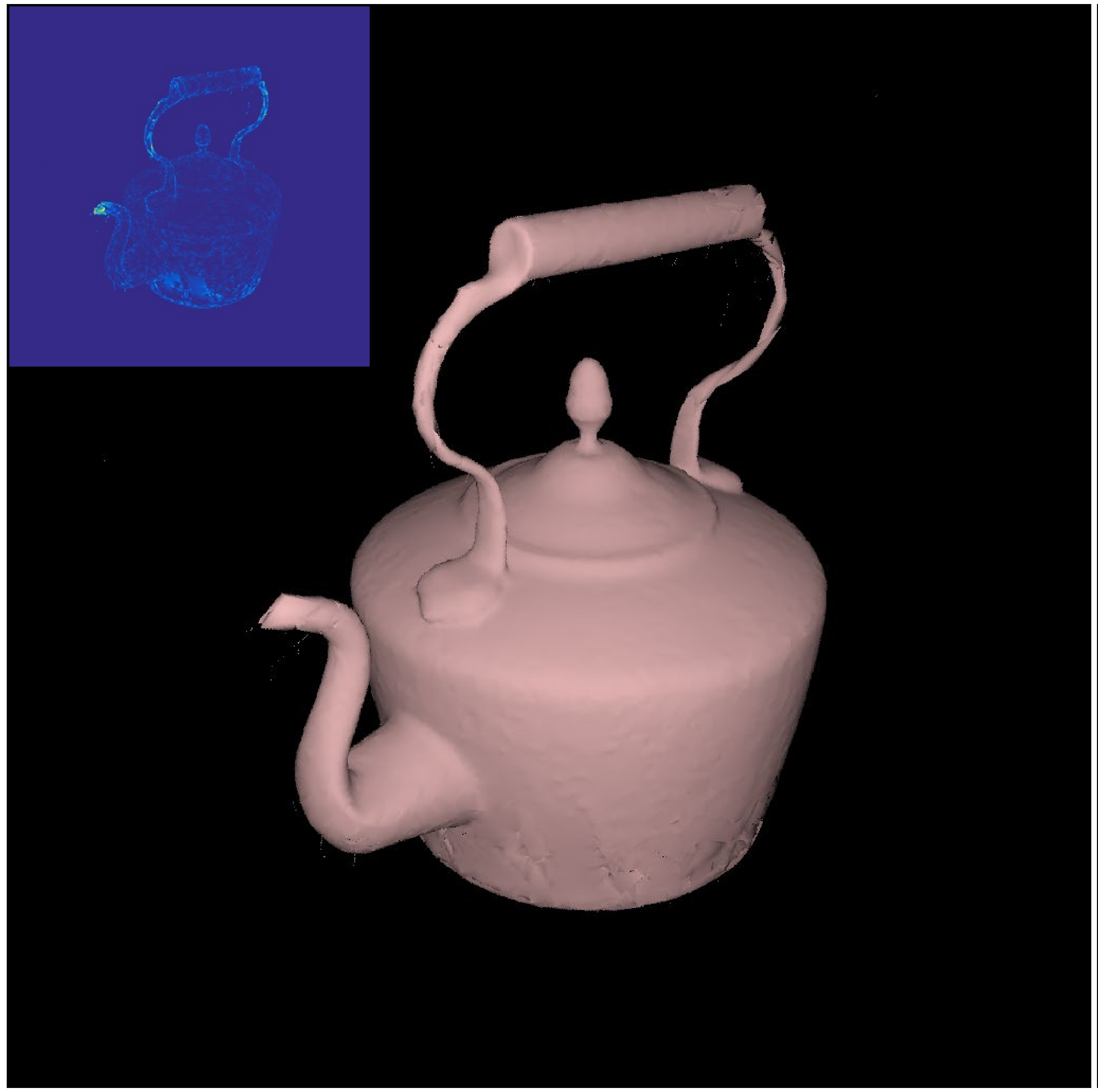
Absolute error

Ray Tracing vs. Rasterization

- We believe that **ray racing** is the way to go for future **differentiable** renderers
- Ray-tracing-based methods are **not** much slower than rasterization
  - Hardware-accelerated ray tracing has been improving rapidly (e.g., Nvidia RTX)
  - Visibility checks and intersections are typically not the performance bottleneck
- Ray-tracing-based methods can compute **correct** (i.e., unbiased) gradients
  - Correct gradients matter in optimization!

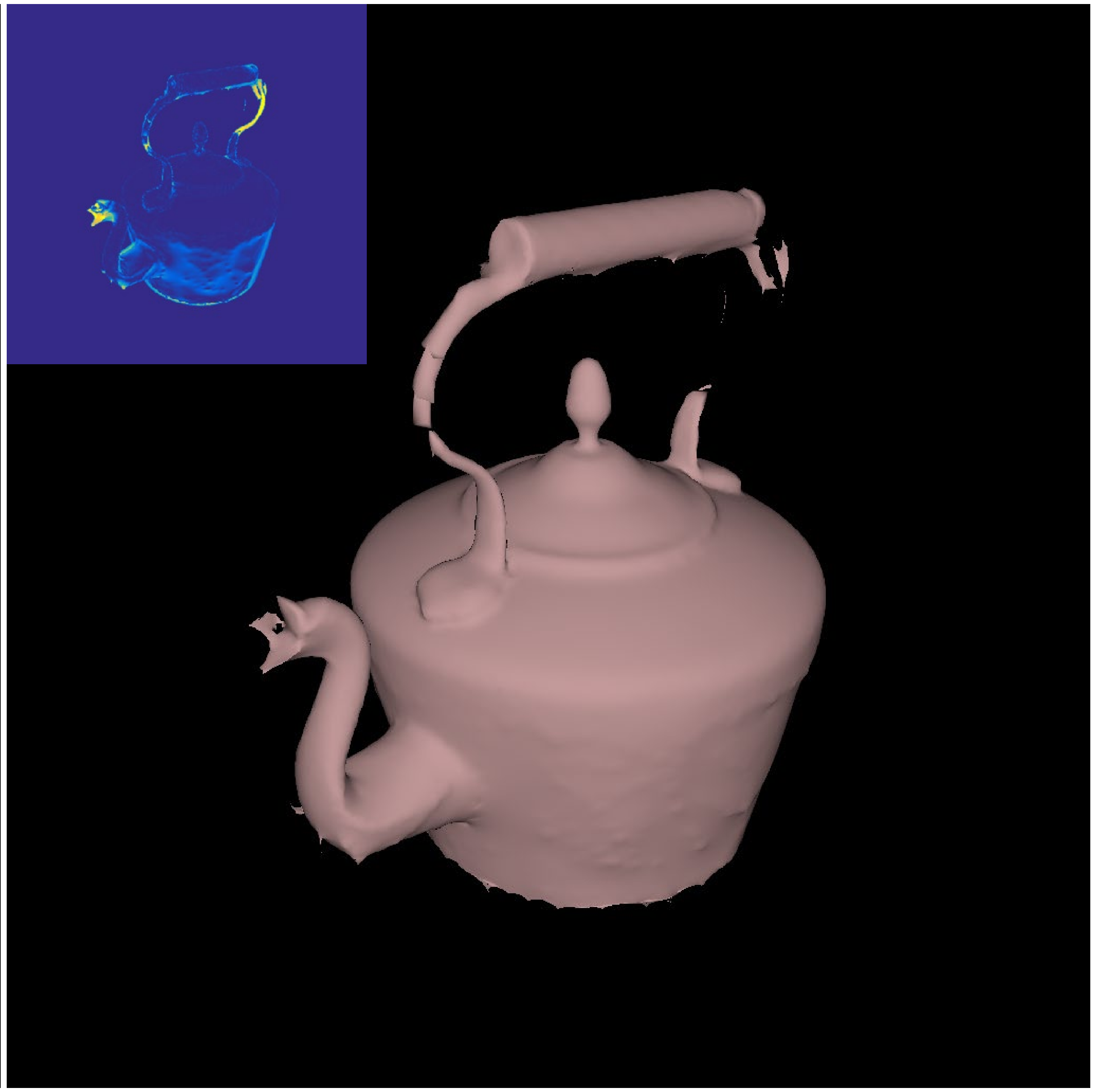
Ray Tracing vs. Rasterization

Optimization results after 5000 iterations (w/ identical settings)



Optimized (psdr-cuda)

Target



Optimized (PyTorch3D)

Ray Tracing vs. Rasterization

- We believe that **ray racing** is the way to go for future **differentiable** renderers

- Ray-tracing-based methods are **not** much slower than rasterization
  - Hardware-accelerated ray tracing has been improving rapidly (e.g., Nvidia RTX)
  - Visibility checks and intersections are typically not the performance bottleneck
- Ray-tracing-based methods can compute **correct** (i.e., unbiased) gradients
  - Correct gradients matter in optimization!

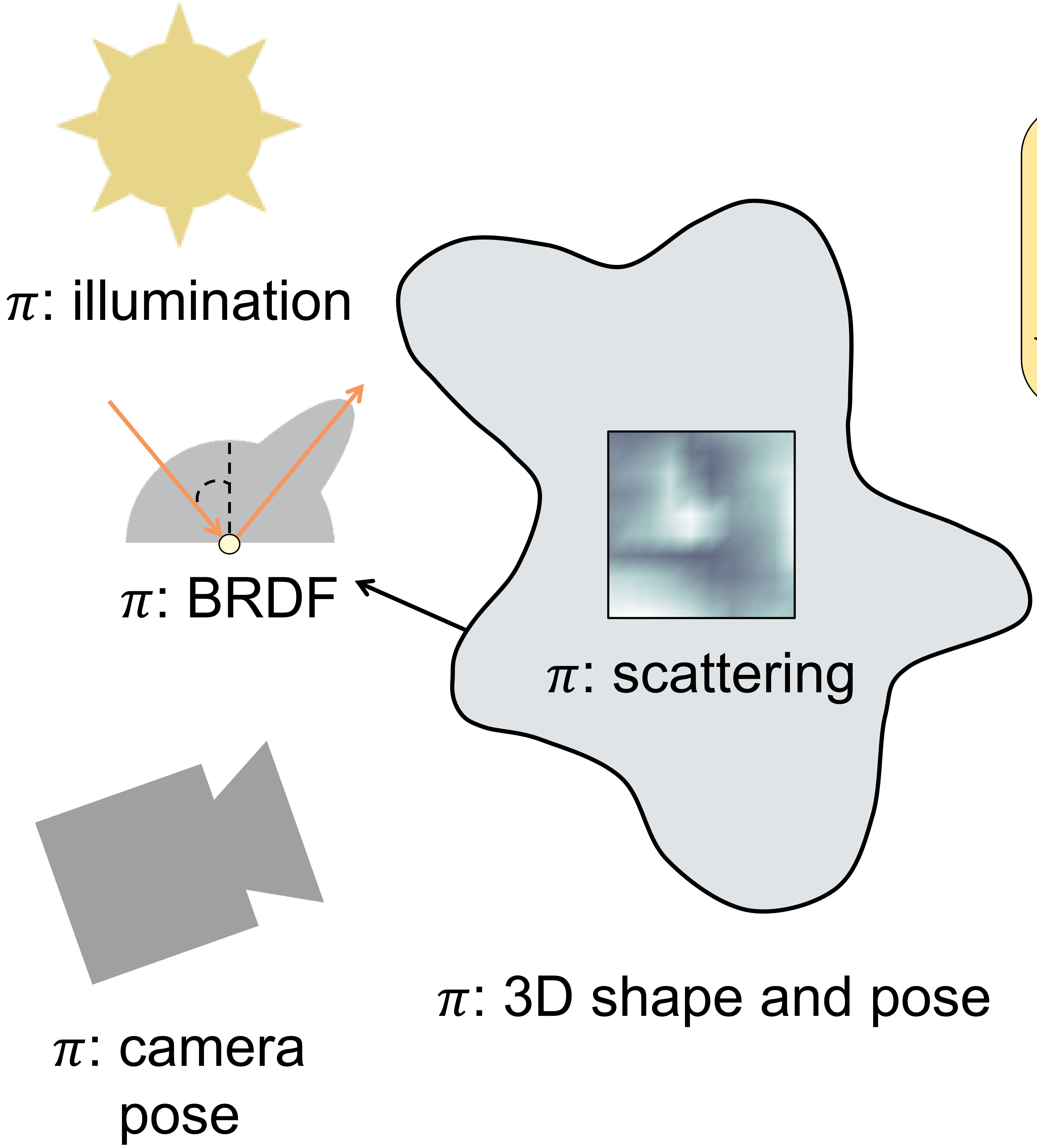
Second part of this tutorial

- Ray-tracing-based methods can handle **complex** light-transport effects
  - Soft shadows, environmental illumination
  - Inter-reflections, radiative transfer (e.g., subsurface scattering), caustics
- Ray-tracing-based methods can provide gradients in **general** scenes
  - Different shape representations, including point clouds, explicit (e.g., meshes), implicit (e.g., neural SDFs)
  - Different types of cameras (e.g., intensity, lightfield, polarization, time-of-flight, hyperspectral, ...)

Third part of this tutorial

What differentiable rendering does not give us

Inverse rendering (a.k.a. analysis by synthesis)



Analysis-by-synthesis optimization:

$\min\limits_{\substack{\text{scene}\\ \text{unknowns } \pi}} \text{loss}\,[$  $, \text{render}\left(\substack{\text{scene}\\ \text{unknowns } \pi}\right)]$

Stochastic gradient descent (e.g., Adam):

$\boxed{\text{initialize } \pi \leftarrow \pi_0}$

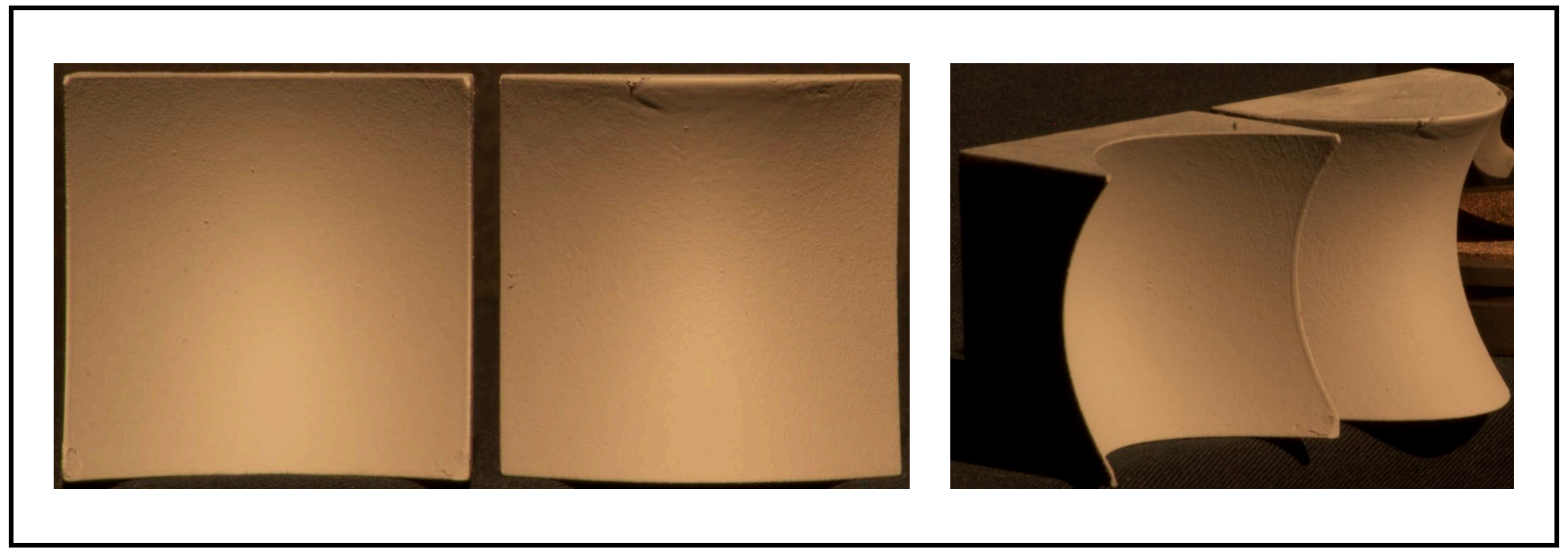
while (not converged)

update $\pi \leftarrow \pi + \eta \cdot \boxed{\dfrac{\mathrm{d}\,\text{loss}(\pi)}{\mathrm{d}\pi}}$

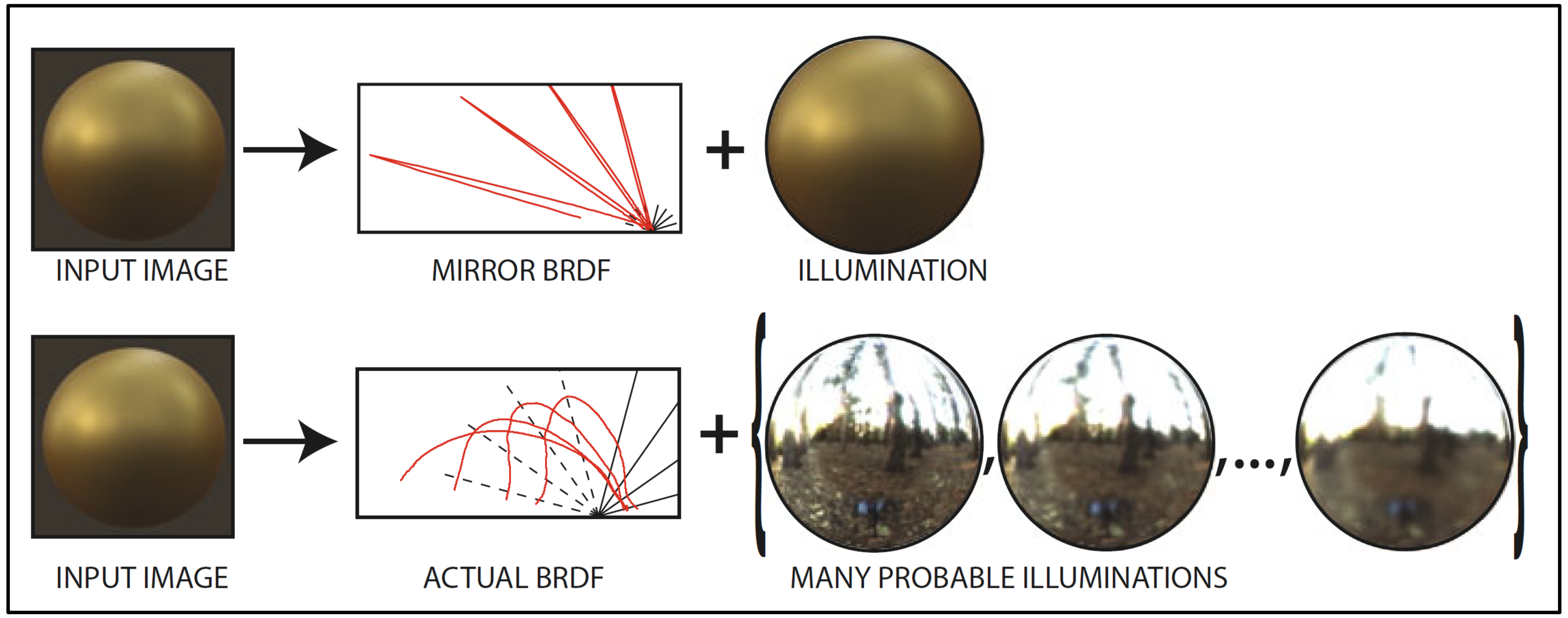
Differentiable rendering

Why we need good initializations

- Analysis-by-synthesis objectives are highly non-convex, non-linear
  - Multiple *local* minima
- Ambiguities exist between different parameters
  - Multiple *global* minima



Ambiguities between shape and lighting [Xiong et al. 2015]



Ambiguities between BRDF and lighting [Romeiro and Zickler 2010]

Ambiguities between scattering parameters [Zhao et al. 2014]

Inverse rendering (a.k.a. analysis by synthesis)

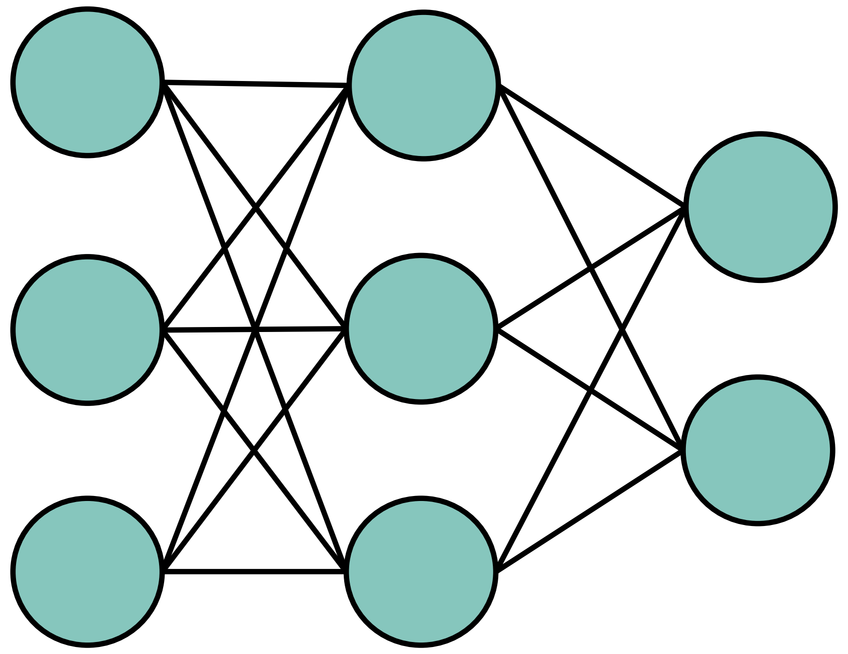
Analysis-by-synthesis optimization:

$$\min_{\text{scene unknowns } \pi} \boxed{\text{loss}} \Big[$$  $$, \text{render}\left(\text{scene unknowns } \pi\right)\Big]$$

Learned initializations help:

- avoid local minima
- accelerate convergence

→



→

Neural network

Stochastic gradient descent (e.g., Adam):

$\boxed{\text{initialize } \pi \leftarrow \pi_0}$

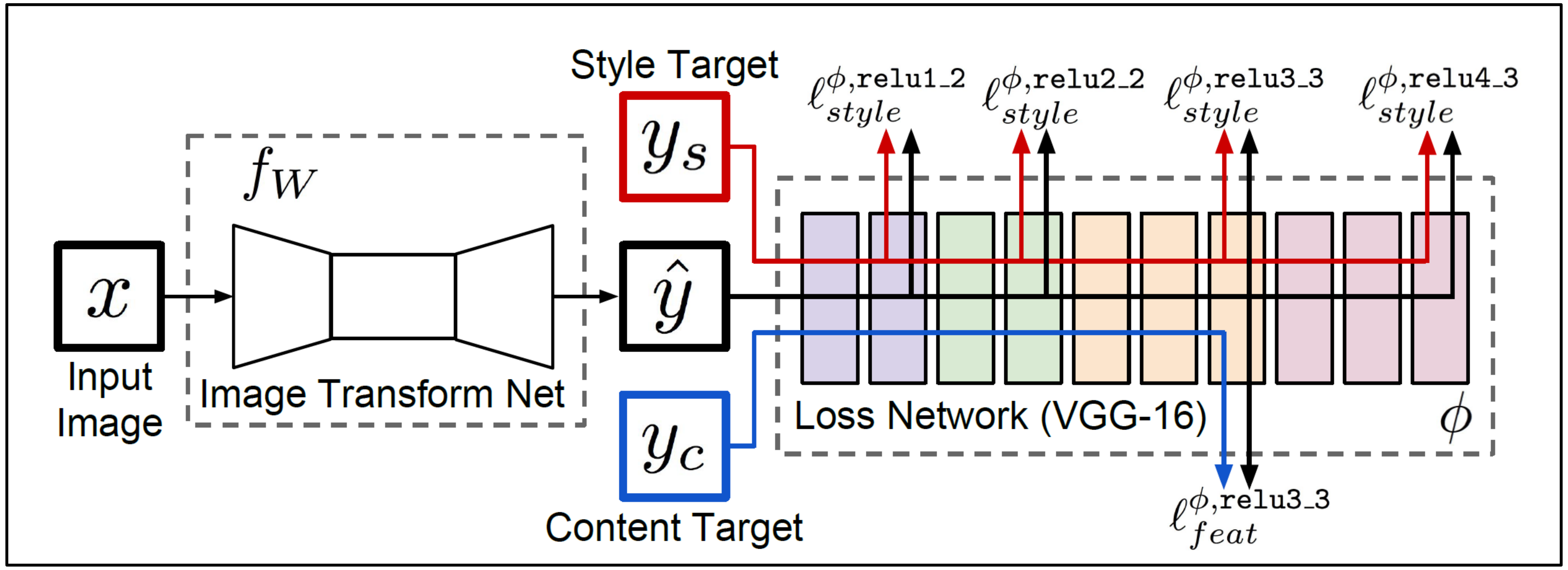
while (not converged)

update $\pi \leftarrow \pi + \eta \cdot \boxed{\dfrac{\mathrm{dloss}(\pi)}{\mathrm{d}\pi}}$

Differentiable rendering

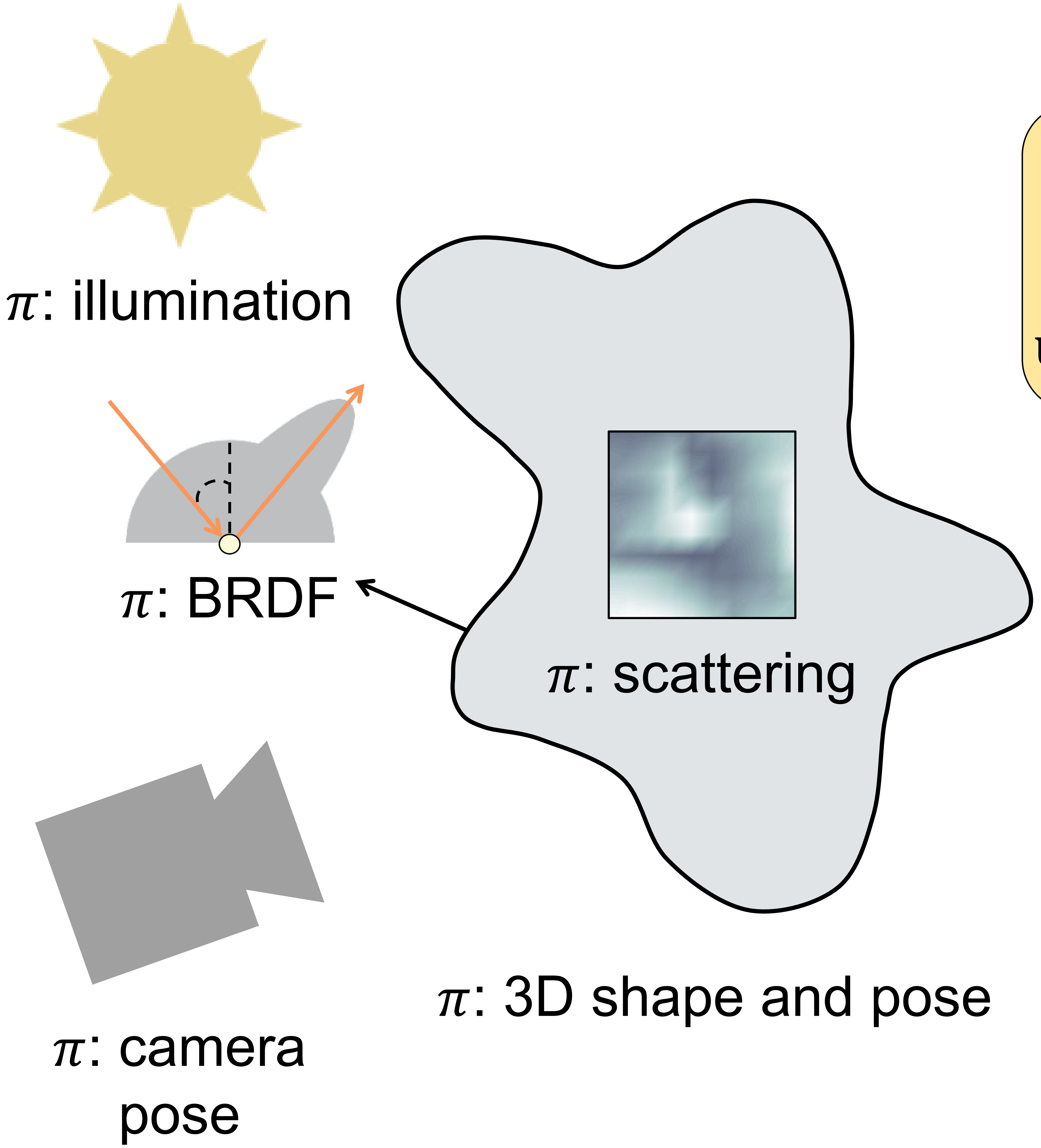
Why we need discriminative loss functions

- Well-designed loss functions can help reduce ambiguities
- Perceptual losses can help emphasize design aspects that matter
- Differentiable rendering can be combined with any loss function that can be backpropagated through



VGG-based *perceptual loss* [Johnson et al. 2016]

Inverse rendering (a.k.a. analysis by synthesis)



Analysis-by-synthesis optimization:

$$\min_{\substack{\text{scene}\\ \text{unknowns } \pi}} \boxed{\text{loss}}\left[\ \boxed{}\ , \text{render}\left(\substack{\text{scene}\\ \text{unknowns } \pi}\right)\right]$$

Stochastic gradient descent (e.g., Adam):

$\boxed{\text{initialize } \pi \leftarrow \pi_0}$

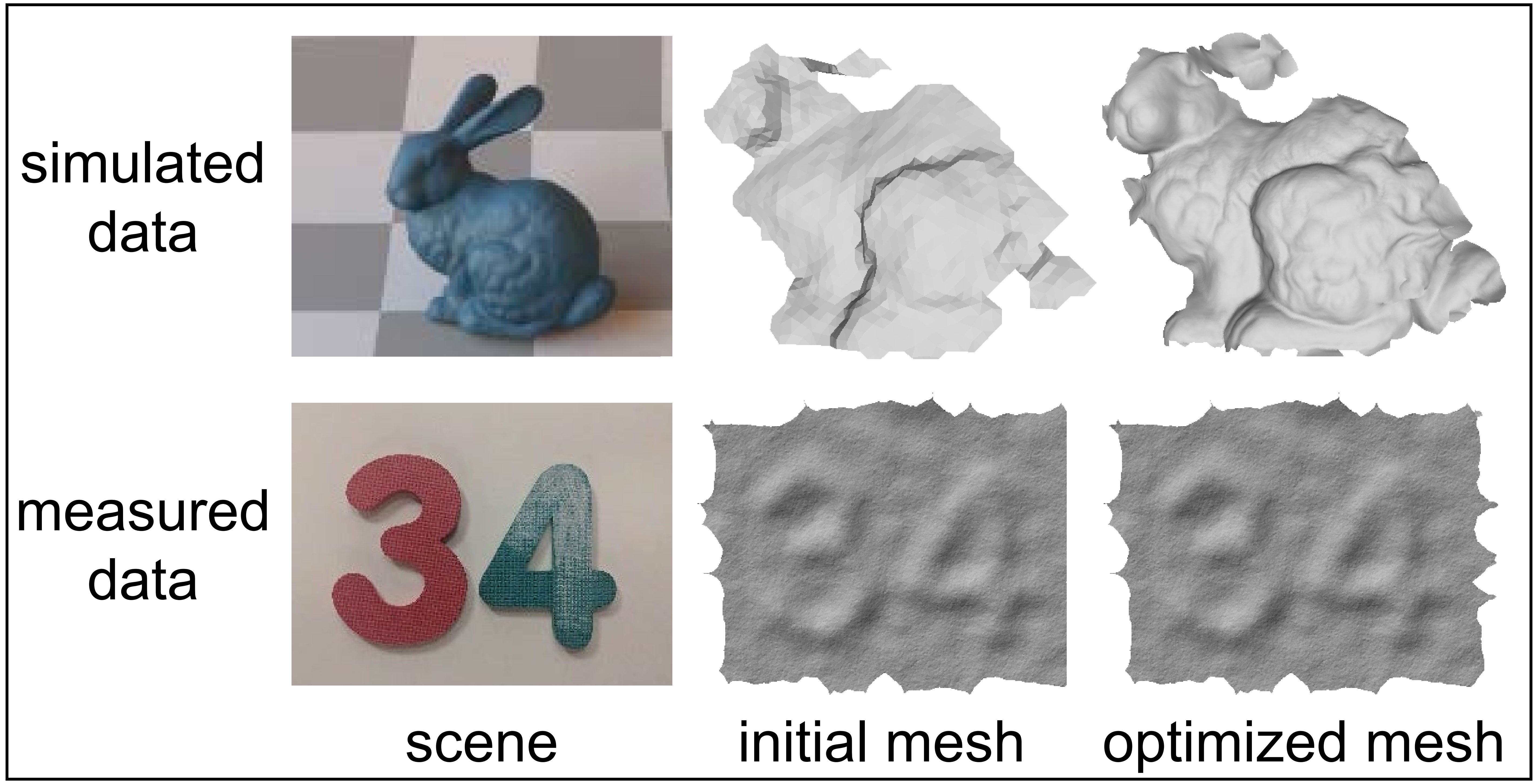
while (not converged)

update $\pi \leftarrow \pi + \eta \cdot \boxed{\dfrac{\mathrm{dloss}(\pi)}{\mathrm{d}\pi}}$

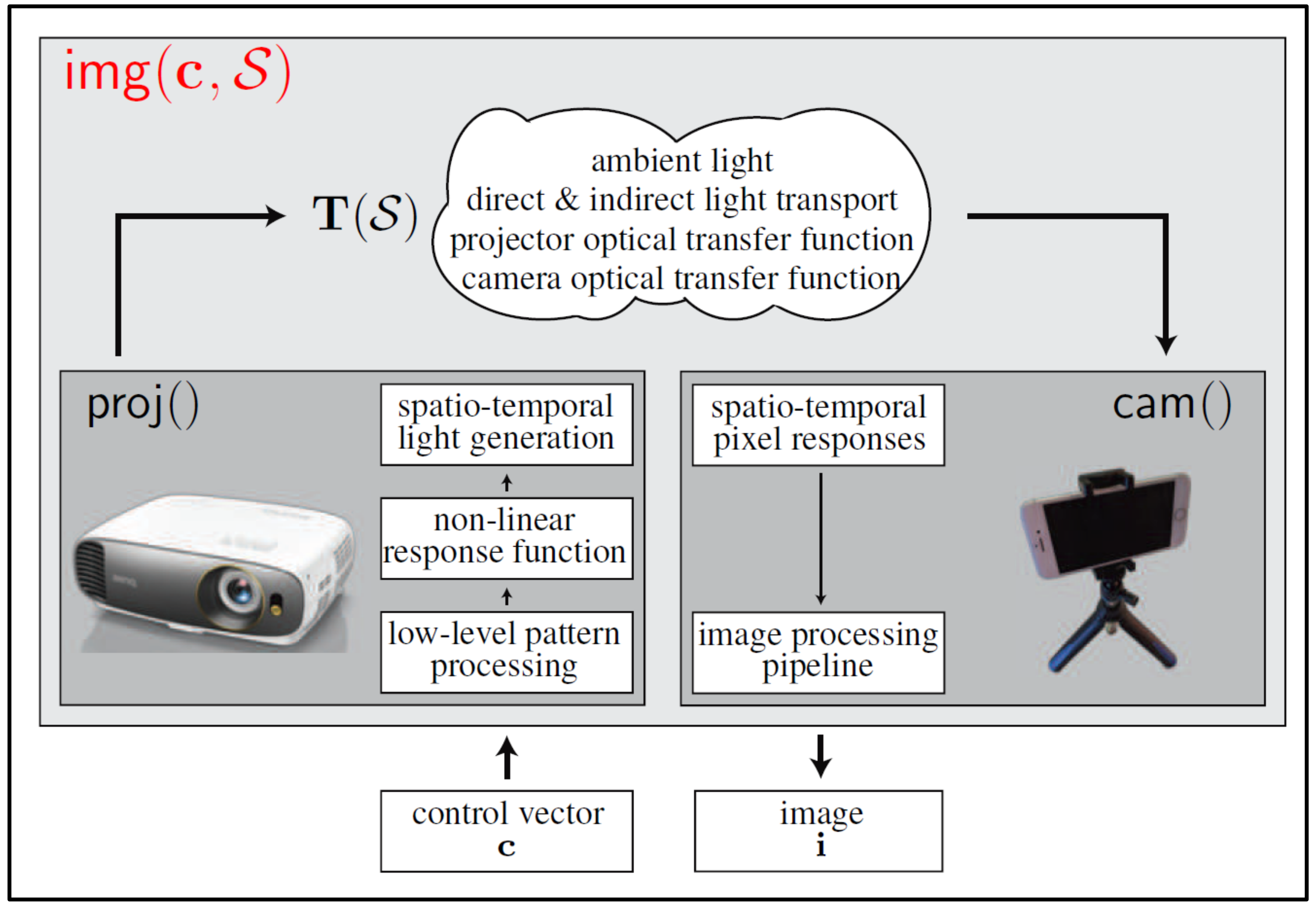
Differentiable rendering

High signal-to-noise ratio is critical

- The extent to which we can improve upon an initialization strongly depends on the signal-to-noise ratio of our measurements
- We need reliable camera models (noise, aberrations, other non-idealities)



Non-line-of-sight imaging [Tsai et al. 2019]



Optical gradient descent [Chen et al. 2020]

Differential Direct Illumination

Reminder from calculus

Differentiation under the integral sign
Also known as the Leibniz integral rule

$$\frac{\mathrm{d}}{\mathrm{d}\pi}\int_{a(\pi)}^{b(\pi)} f(x,\pi)\mathrm{d}x \stackrel{?}{=} \int_{a(\pi)}^{b(\pi)} \frac{\mathrm{d}}{\mathrm{d}\pi} f(x,\pi)\mathrm{d}x$$

Move derivative inside integral

Account for changes in integration limits

$$+ \; f(b(\pi),\pi)\frac{\mathrm{d}b(\pi)}{\mathrm{d}\pi} - f(\alpha(\pi);\pi)\frac{\mathrm{d}a(\pi)}{\mathrm{d}\pi}$$

Account for discontinuities of integrand that depend on π

$$+ \; \sum_i \left(f(c_i(\pi)^-,\pi) - f(c_i(\pi)^+,\pi)\right)\frac{\mathrm{d}c_i(\pi)}{\mathrm{d}\pi}$$

A simple example

$$f(x,\pi) = \begin{cases} 0 & \text{if } x < 2\pi \\ 1 & \text{if } x \geq 2\pi \end{cases}$$

$$\frac{\mathrm{d}}{\mathrm{d}\pi}\int_{0}^{4\pi} f(x,\pi)\mathrm{d}x = \int_{0}^{2\pi} \frac{\mathrm{d}}{\mathrm{d}\pi} 0\mathrm{d}x + \int_{\pi}^{4\pi} \frac{\mathrm{d}}{\mathrm{d}\pi} 1\mathrm{d}x$$

Move derivative inside integral

Account for changes in integration limits

$$+ \; 1\frac{\mathrm{d}(4\pi)}{\mathrm{d}\pi} - 0\frac{\mathrm{d}0}{\mathrm{d}\pi}$$

Account for discontinuities of integrand that depend on π

$$+ \; (0-1)\frac{\mathrm{d}(2\pi)}{\mathrm{d}\pi}$$

Leibniz integral rule

Differentiation under the integral sign
Also known as the Leibniz integral rule

Interior integral

$$\frac{\mathrm{d}}{\mathrm{d}\pi}\int_{a(\pi)}^{b(\pi)} f(x,\pi)\mathrm{d}x = \int_{a(\pi)}^{b(\pi)} \frac{\mathrm{d}}{\mathrm{d}\pi} f(x,\pi)\mathrm{d}x$$

Move derivative inside integral

Boundary terms

Account for changes in integration limits

$$+ \; f(b(\pi),\pi)\frac{\mathrm{d}b(\pi)}{\mathrm{d}\pi} - f(\alpha(\pi);\pi)\frac{\mathrm{d}a(\pi)}{\mathrm{d}\pi}$$

Account for discontinuities of integrand that depend on π

$$+ \; \sum_i \left(f(c_i(\pi)^-,\pi) - f(c_i(\pi)^+,\pi)\right)\frac{\mathrm{d}c_i(\pi)}{\mathrm{d}\pi}$$

Simplified Leibniz integral rule

Differentiation under the integral sign
Also known as the Leibniz integral rule

Interior integral

$$\frac{\mathrm{d}}{\mathrm{d}\pi}\int_a^b f(x,\pi)\,\mathrm{d}x = \boxed{\int_a^b \frac{\mathrm{d}}{\mathrm{d}\pi} f(x,\pi)\,\mathrm{d}x}$$

Move derivative inside integral

Differentiation wrt $\boldsymbol{\pi}$ simplifies to just moving derivative inside integral when:

- Integration limits are independent of $\boldsymbol{\pi}$.
- Integrand discontinuities are independent of $\boldsymbol{\pi}$.

Reynolds transport theorem

Interior integral

Boundary integral

$$\frac{\mathrm{d}}{\mathrm{d}\pi}\int_{\Omega(\pi)} f(x,\pi)\,\mathrm{d}A(x) \overset{?}{=} \int_{\Omega(\pi)} \frac{\mathrm{d}f(x,\pi)}{\mathrm{d}\pi}\,\mathrm{d}A(x) + \int_{\partial\Omega(\pi)} g(x,\pi)\,\mathrm{d}l(x)$$

Boundary domain

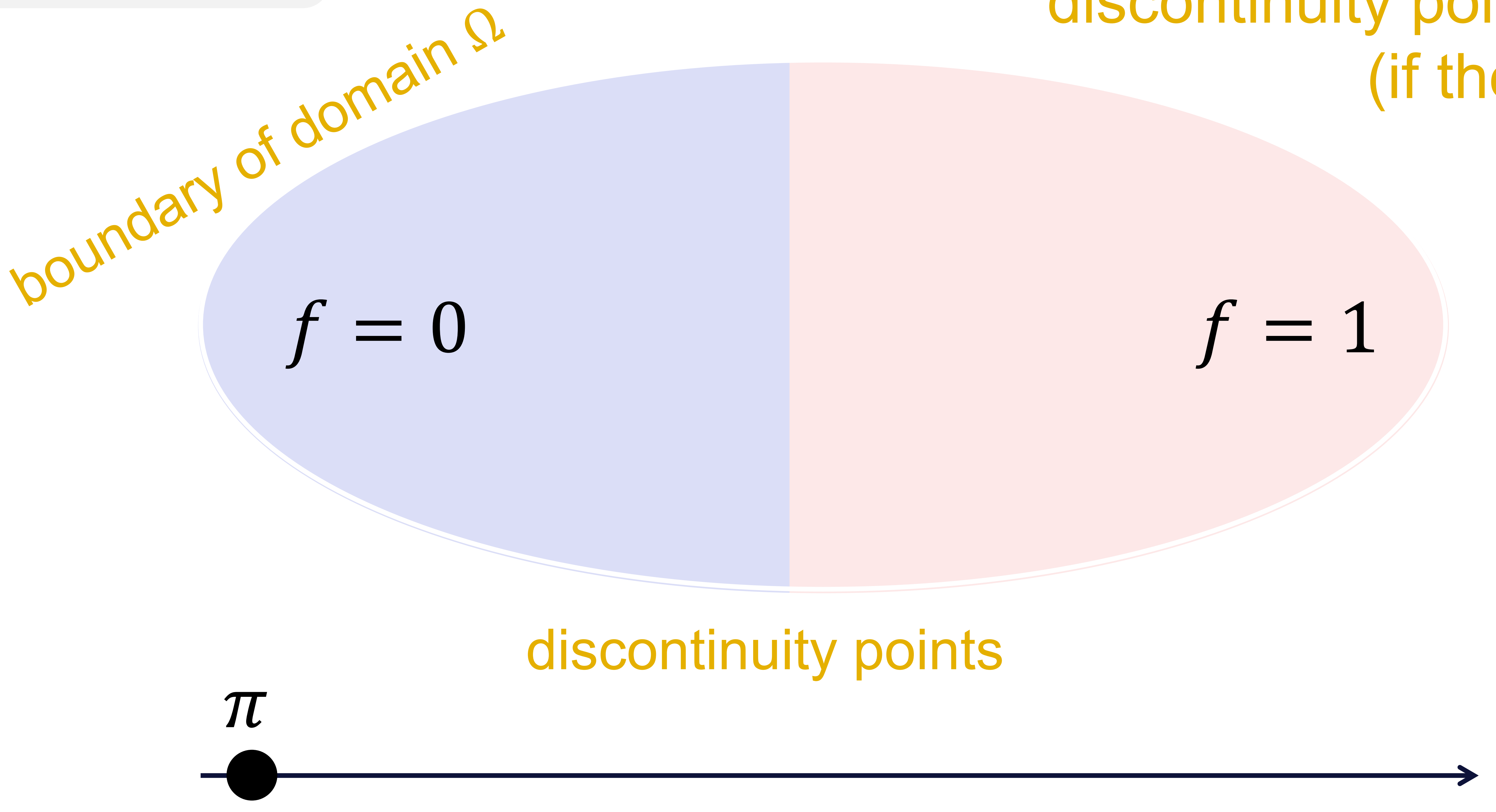
||

discontinuity points ∪ boundary of domain Ω

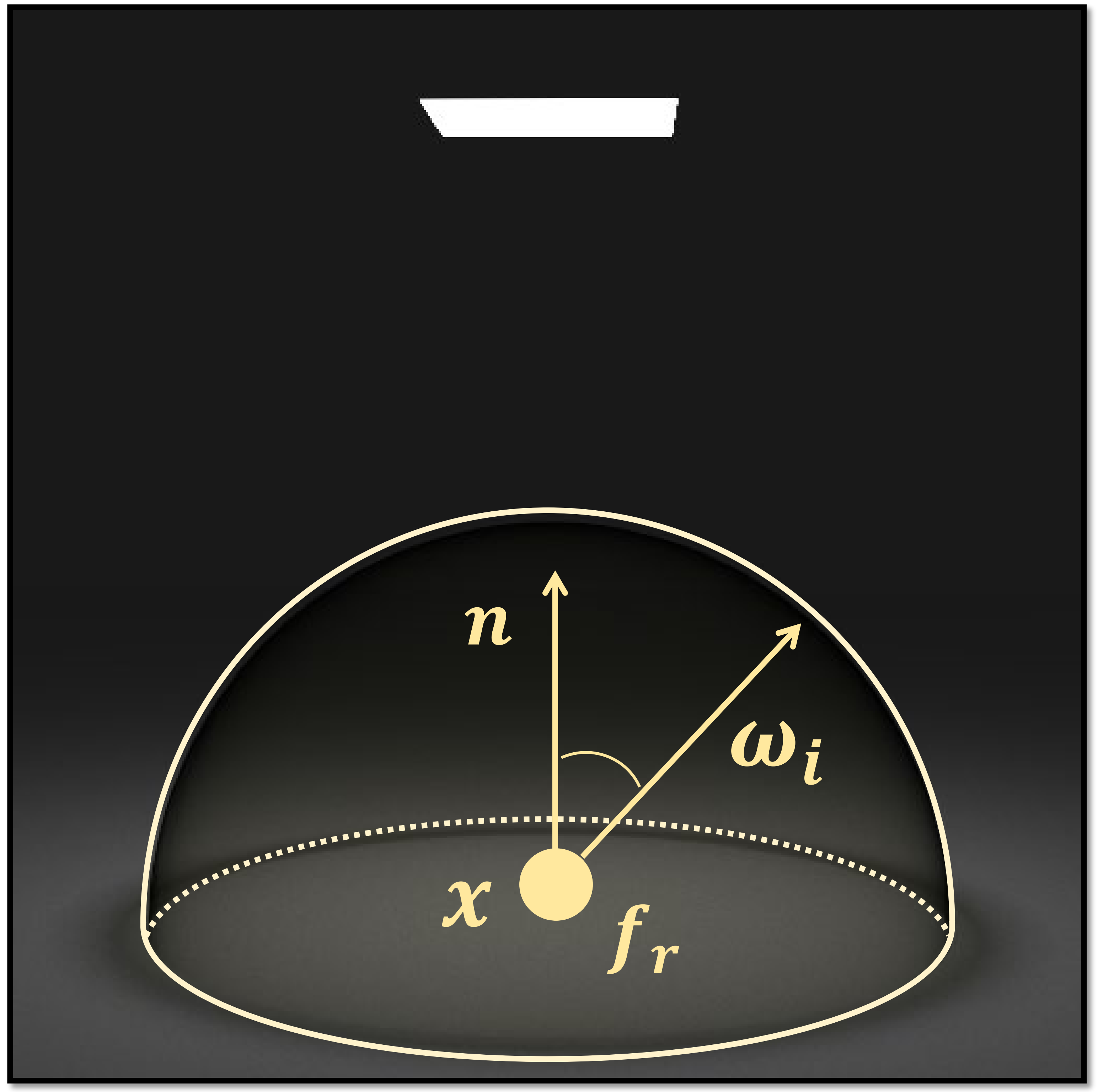
(if they depend on π)

Reynolds transport theorem [1903]

Generalization of the Leibniz rule



Direct illumination integral



Radiance from x:

$$I = \int_{\mathbb{H}^2} \underbrace{f_r(\omega_i, \omega_o)}_{\text{Reflectance (BRDF)}} \underbrace{L_i(\omega_i)}_{\text{Incident radiance}} \underbrace{(n \cdot \omega_i)}_{\text{Shading wrt normal } \boldsymbol{n}} \mathrm{d}\sigma(\omega_i)$$

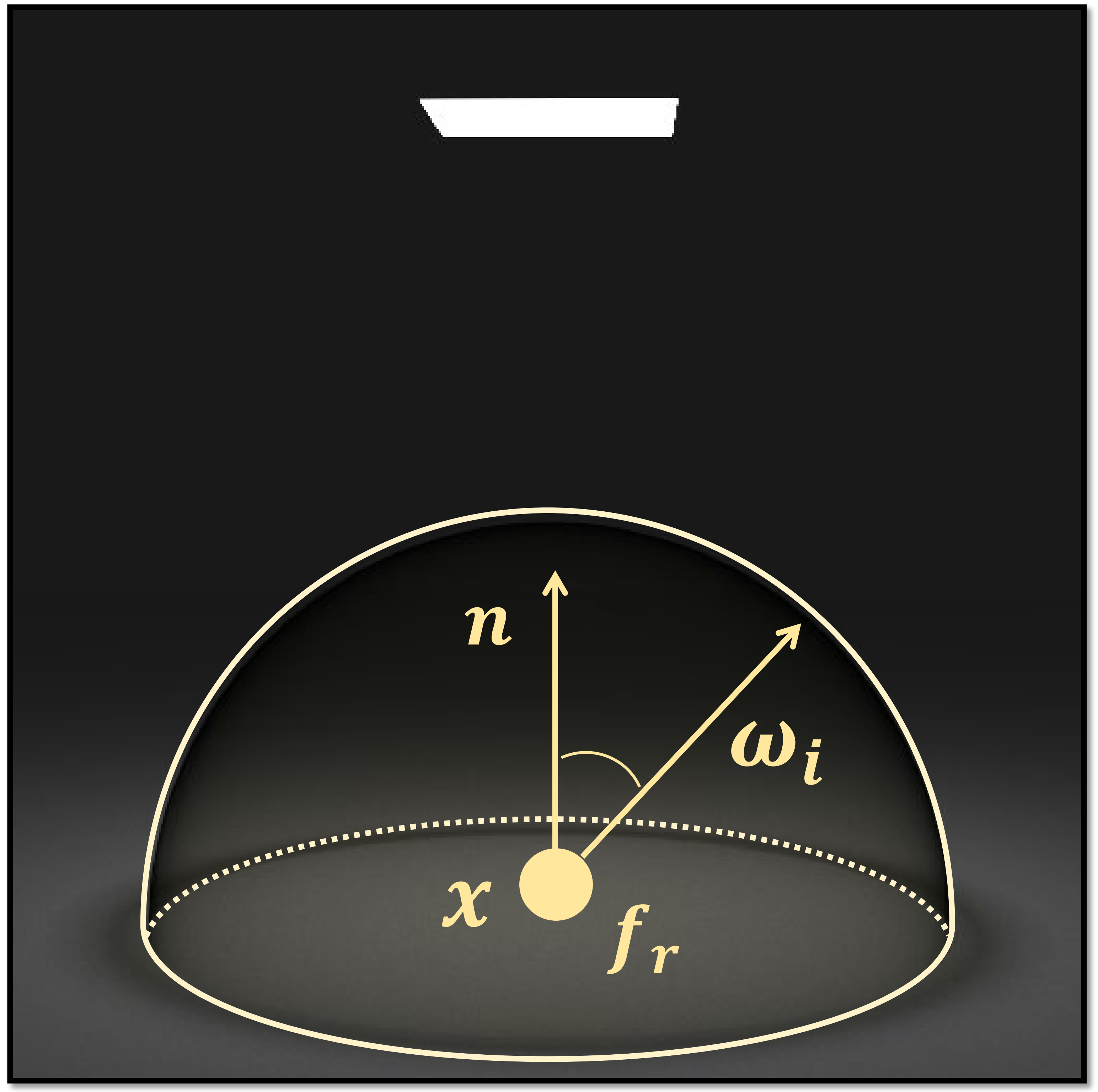
Unit hemisphere

Monte Carlo rendering:

- Sample random directions ω_i^s from PDF $p(\omega_i)$
- Form estimator

$$I \approx \sum_s \frac{f_r(\omega_i^s, \omega_o)\, L_i(\omega_i^s)\, (n \cdot \omega_i^s)}{p(\omega_i^s)}$$

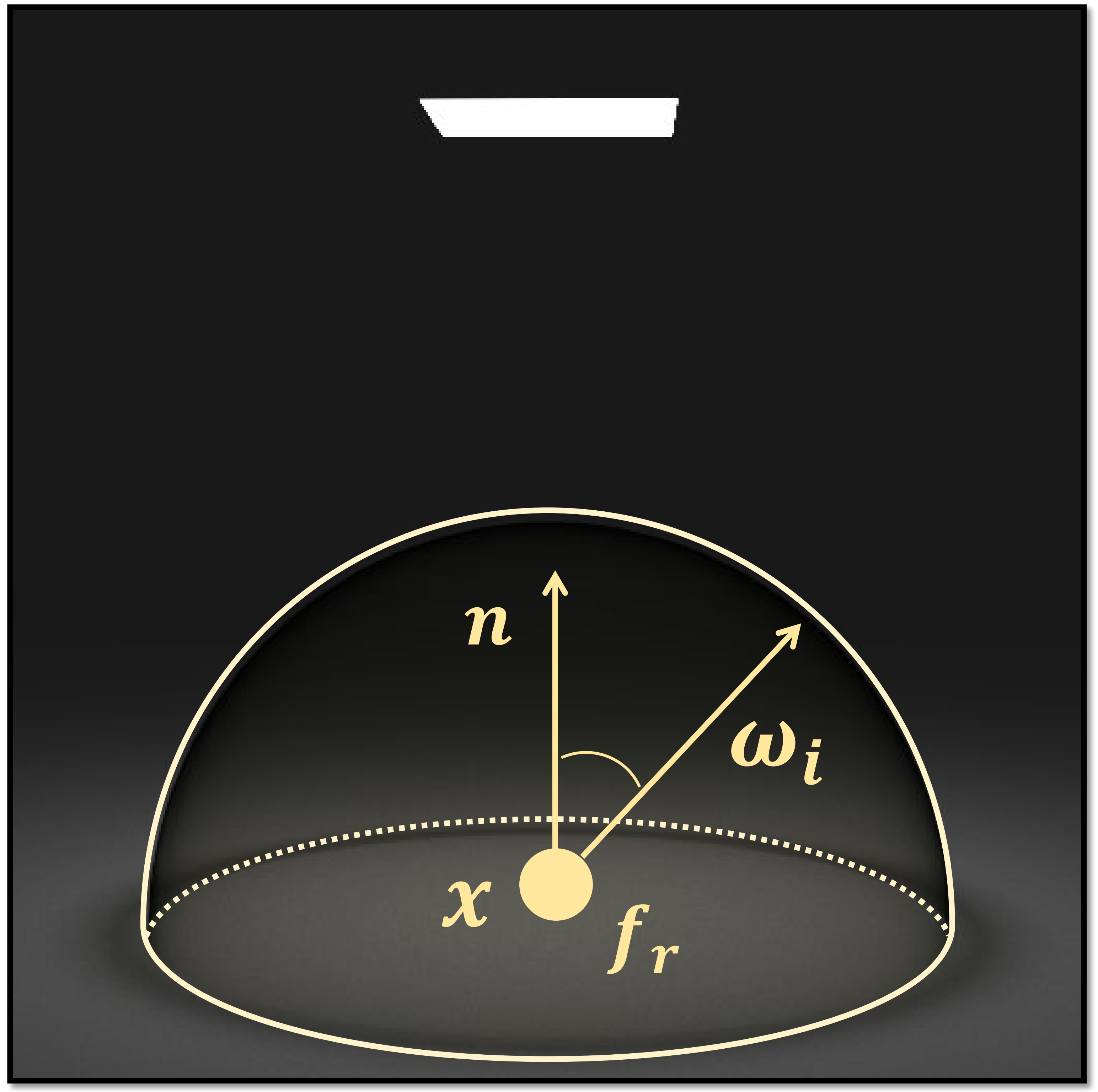
Differential direct illumination



Differential radiance from x:

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \frac{\mathrm{d}}{\mathrm{d}\pi} \int_{\mathbb{H}^2} f_r(\omega_i, \omega_o)\, L_i(\omega_i)\, (n \cdot \omega_i)\, \mathrm{d}\sigma(\omega_i)$$

Differential direct illumination: local parameters



π: *local* parameters
- BRDF parameters
- *shading* normal
- illumination brightness

Differential radiance from x:

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\mathbb{H}^2} \frac{\mathrm{d}}{\mathrm{d}\pi}\{f_r(\omega_i, \omega_o)\, L_i(\omega_i)\, (n \cdot \omega_i)\}\, \mathrm{d}\sigma(\omega_i)$$

Just move derivative inside integral

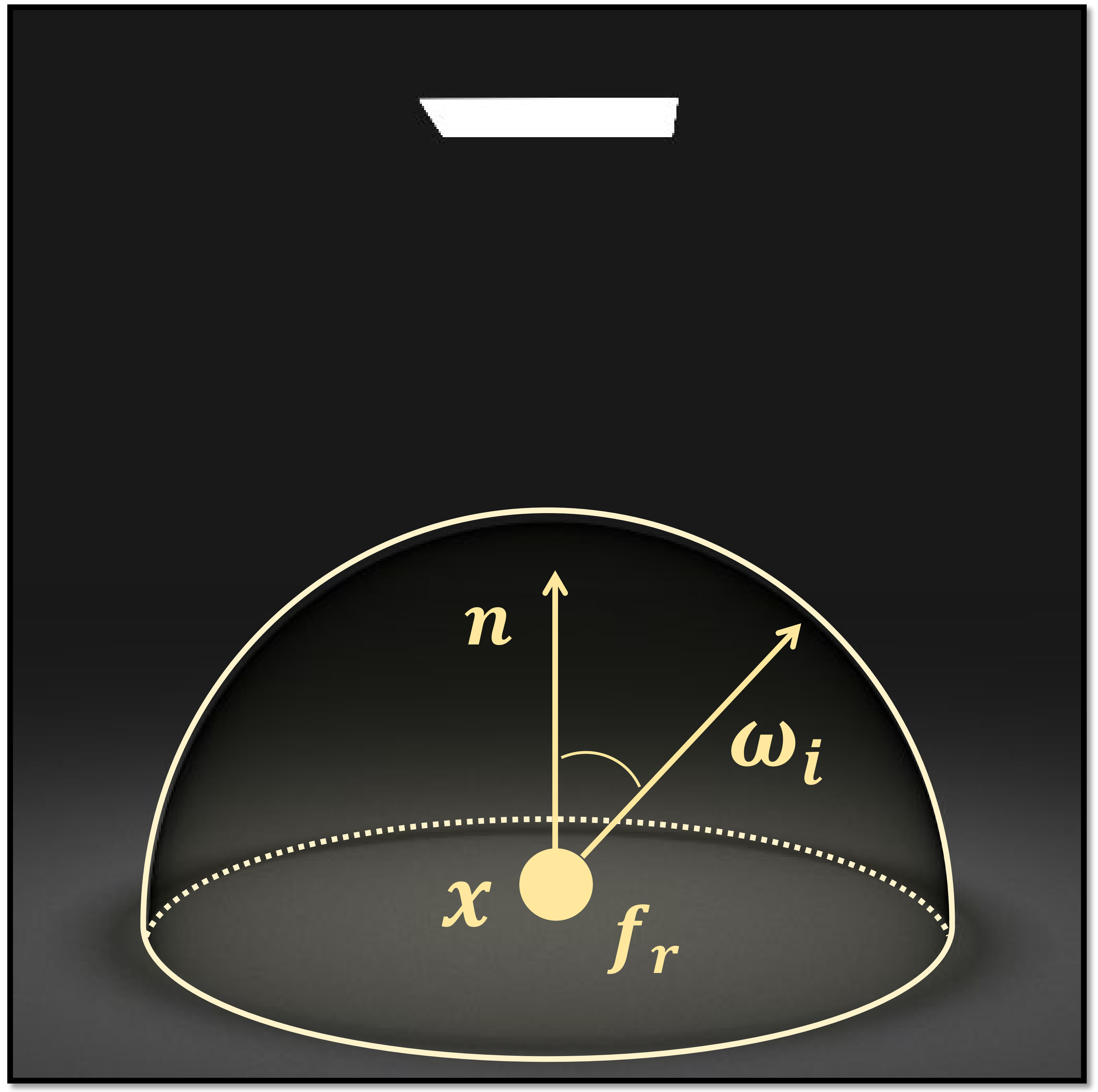
Monte Carlo differentiable rendering:

- Sample random directions ω_i^s from PDF $p(\omega_i)$
- Form estimator

Just differentiate numerator
[Khungurn et al. 2015, Gkioulekas et al. 2015]

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} \approx \sum_s \frac{\frac{\mathrm{d}}{\mathrm{d}\pi}\{f_r(\omega_i^s, \omega_o)\, L_i(\omega_i^s)\, (n \cdot \omega_i^s)\}}{p(\omega_i^s)}$$

Alternative estimator



π: *local* parameters
- BRDF parameters

Differential radiance from x:

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\mathbb{H}^2} \frac{\mathrm{d}}{\mathrm{d}\pi}\{f_r(\omega_i, \omega_o, \pi) L_i(\omega_i)(n \cdot \omega_i)\}\mathrm{d}\sigma(\omega_i)$$

Just move derivative inside integral

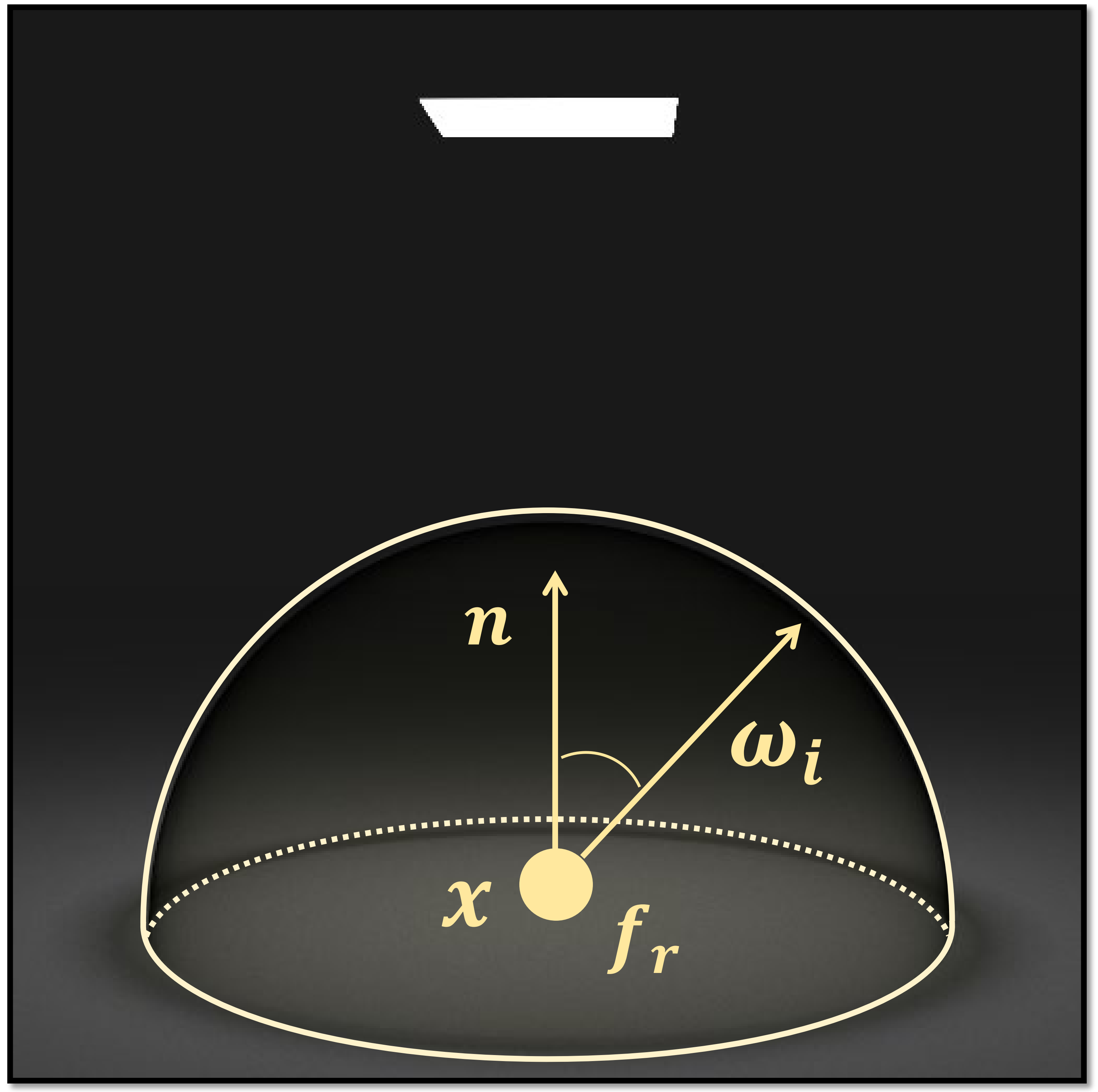
Monte Carlo estimation:

- Sample random directions ω_i^s from PDF $p(\omega_i, \pi)$
- Form estimator

Differentiate entire contribution [Zeltner et al. 2021]

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} \approx \sum_s \frac{\mathrm{d}}{\mathrm{d}\pi}\left\{\frac{f_r(\omega_i^s, \omega_o, \pi)\, L_i(\omega_i^s)\, (n \cdot \omega_i^s)}{p(\omega_i^s, \pi)}\right\}$$

Differential direct illumination: global parameters



π: *global* parameters

- shape and pose of different scene elements (camera, sources, objects)

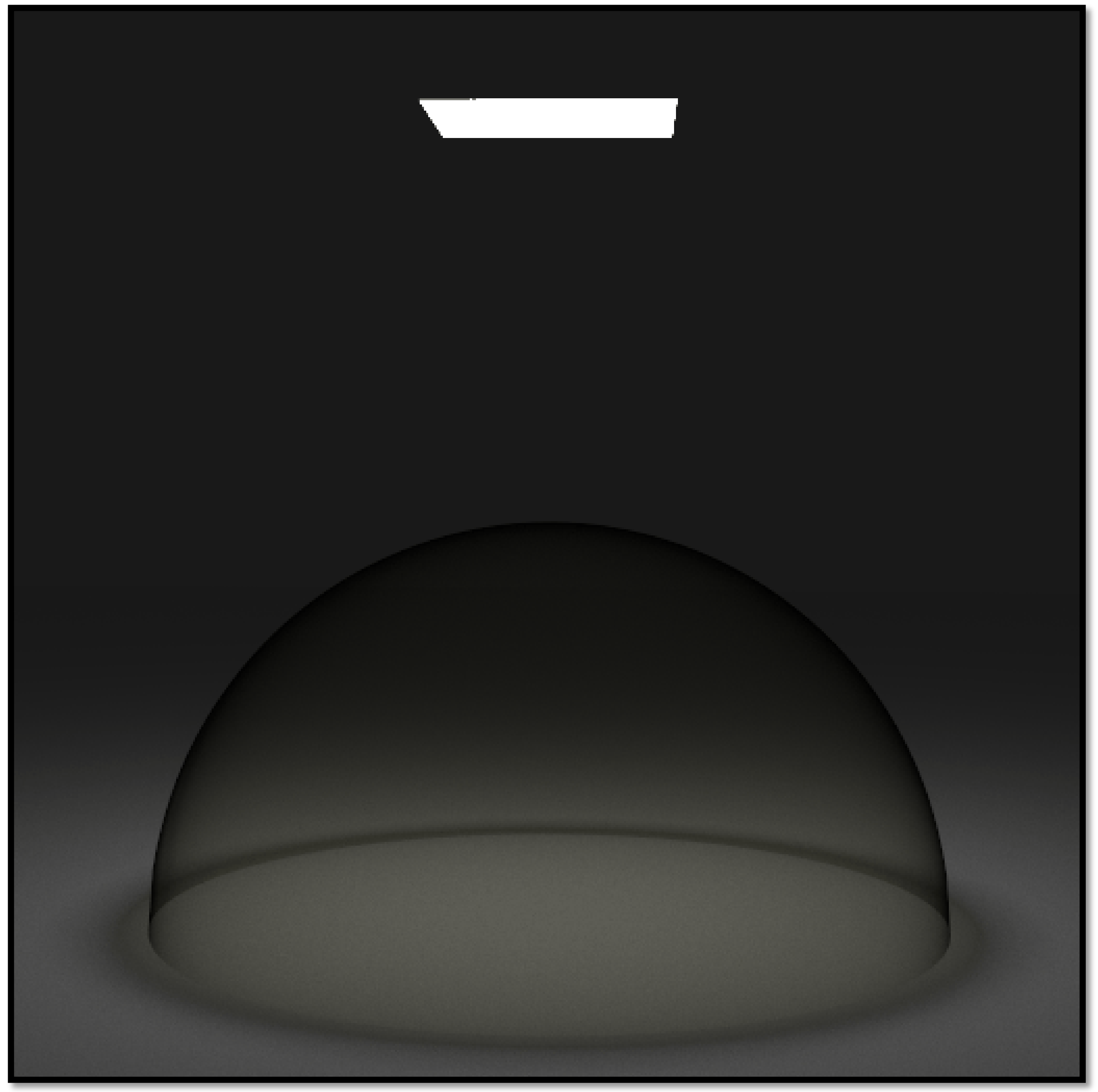
Differential radiance from x:

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \frac{\mathrm{d}}{\mathrm{d}\pi} \int_{\mathbb{H}^2} f_r(\omega_i, \omega_o)\, L_i(\omega_i)\, (n \cdot \omega_i)\, \mathrm{d}\sigma(\omega_i)$$

$$\cancel{= \int_{\mathbb{H}^2} \frac{\mathrm{d}}{\mathrm{d}\pi} \{f_r(\omega_i, \omega_o)\, L_i(\omega_i)\, (n \cdot \omega_i)\}\, \mathrm{d}\sigma(\omega_i)}$$

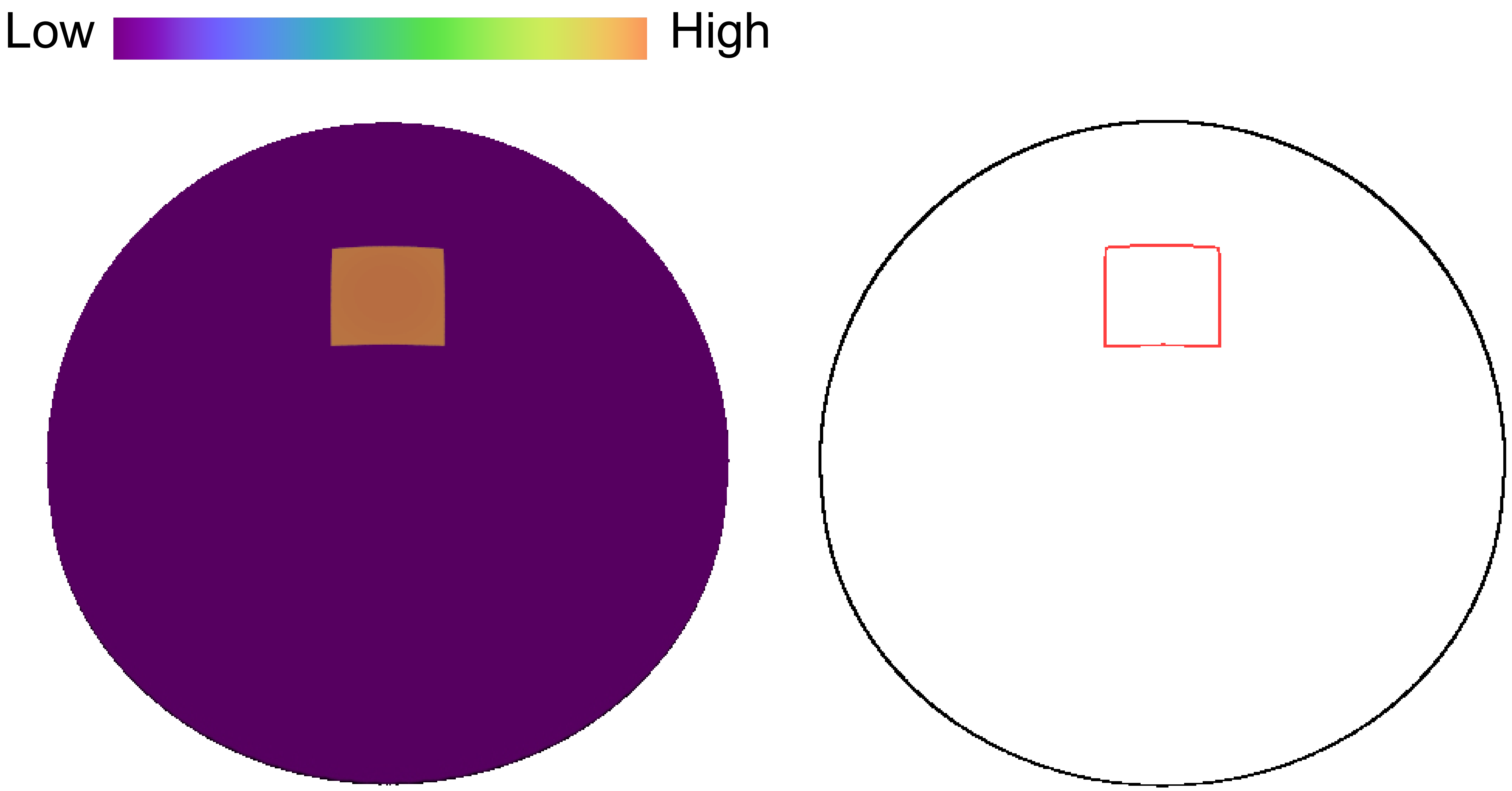
Need to use full Reynolds transport theorem

Discontinuities in the integrand



π: size of the emitter

$$I = \int_{\mathbb{H}^2} \underbrace{f_r(\omega_i, \omega_o) L_i(\omega_i)(n \cdot \omega_i)}_{f(\omega_i)} \mathrm{d}\sigma(\omega_i)$$



Integrand
$f(\omega_i)$

Discontinuous points
(π-dependent)

Applying the Reynolds transport theorem

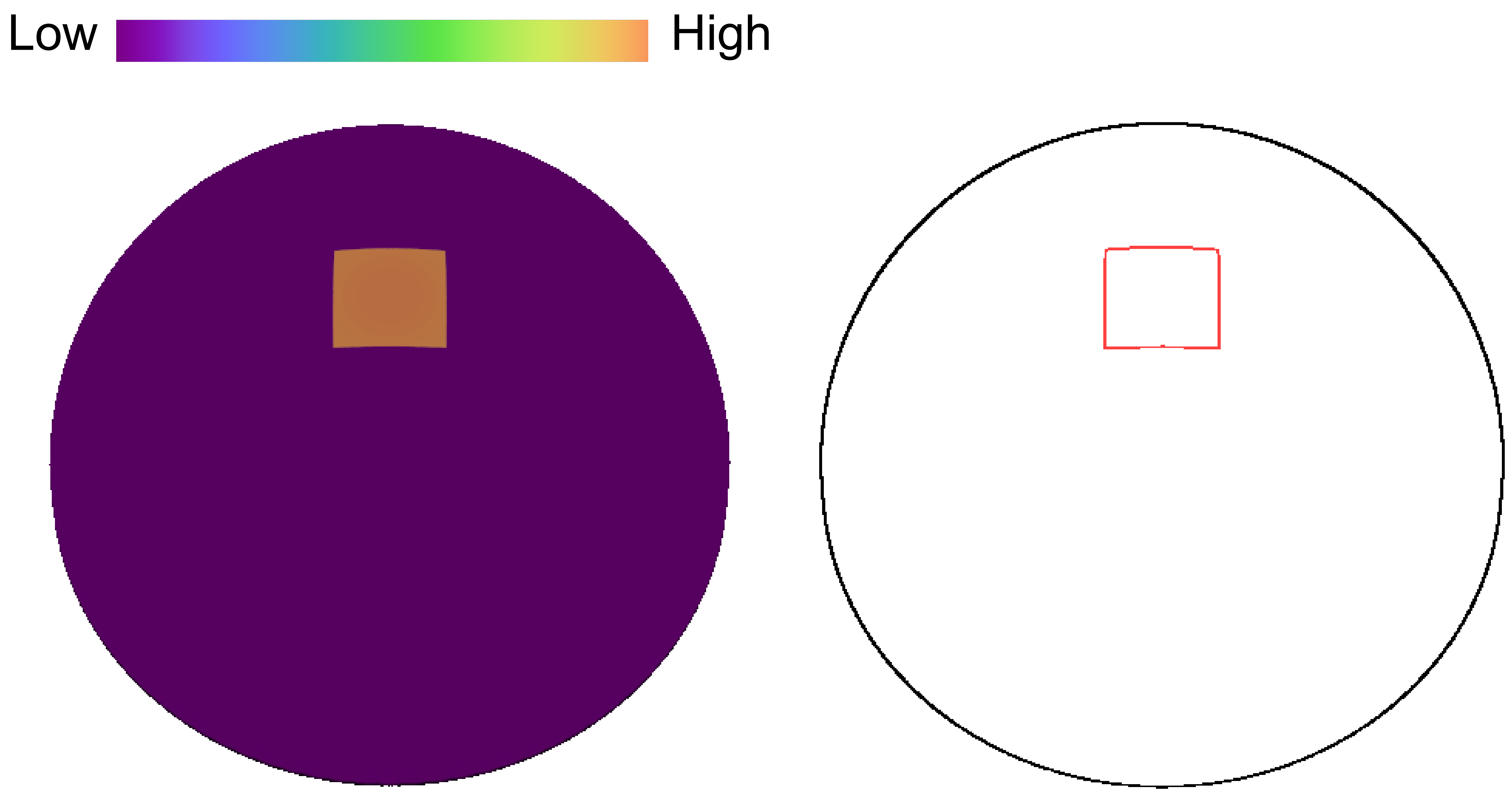
$$I = \int_{\mathbb{H}^2} f(\omega_i, \omega_o) \mathrm{d}\sigma(\omega_i)$$

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \boxed{\int_{\mathbb{H}^2} \frac{\mathrm{d}f}{\mathrm{d}\pi}\, \mathrm{d}\sigma} + \boxed{\int_{\partial\mathbb{H}^2} g\, \mathrm{d}l}$$

Interior integral (same as for local parameters)

Boundary integral

[Ramamoorthi et al. 2007, Li et al. 2019]

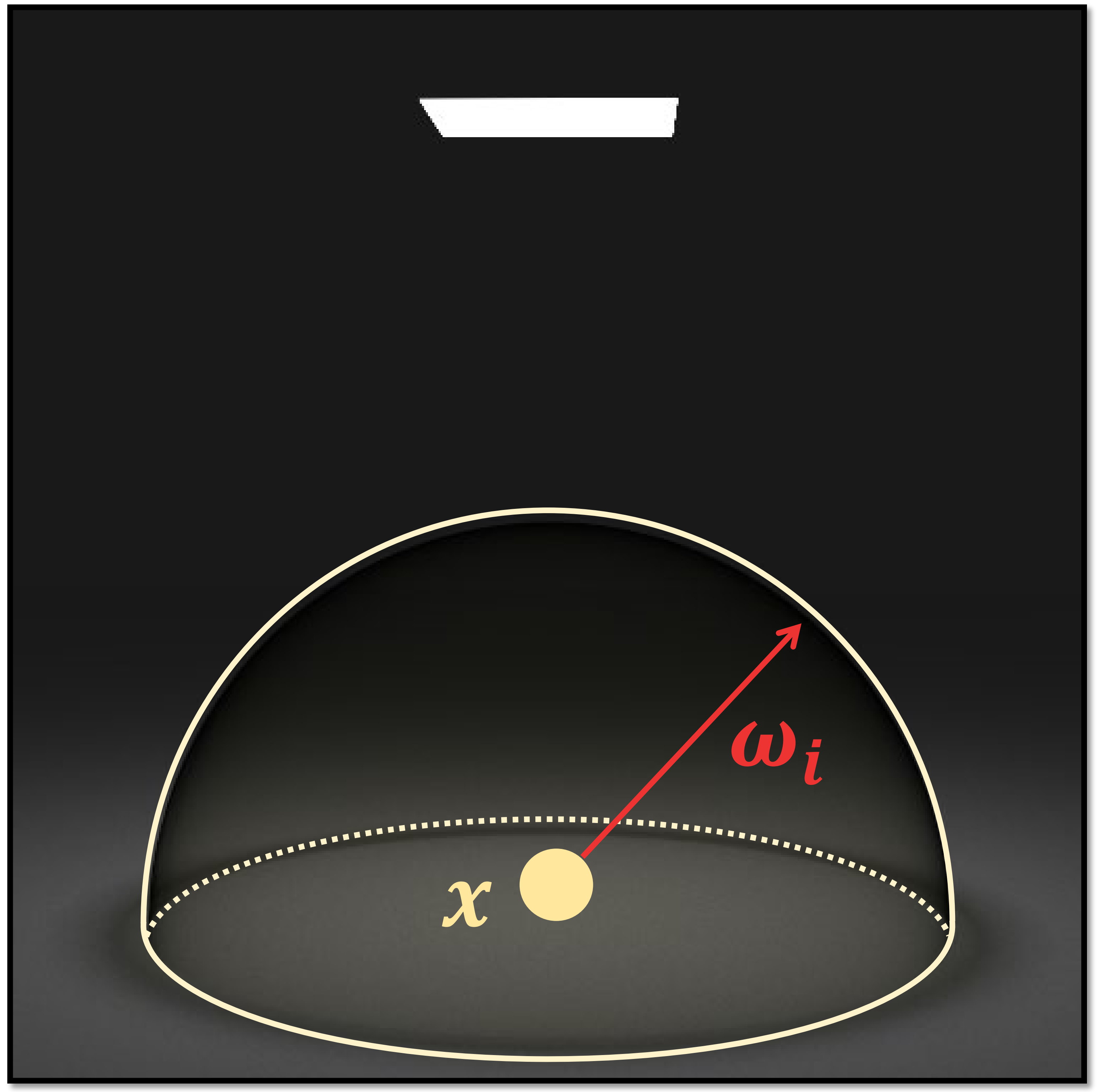


Integrand $f(\omega_i)$

Discontinuous points (π-dependent)

Reparameterizing the direct illumination integral

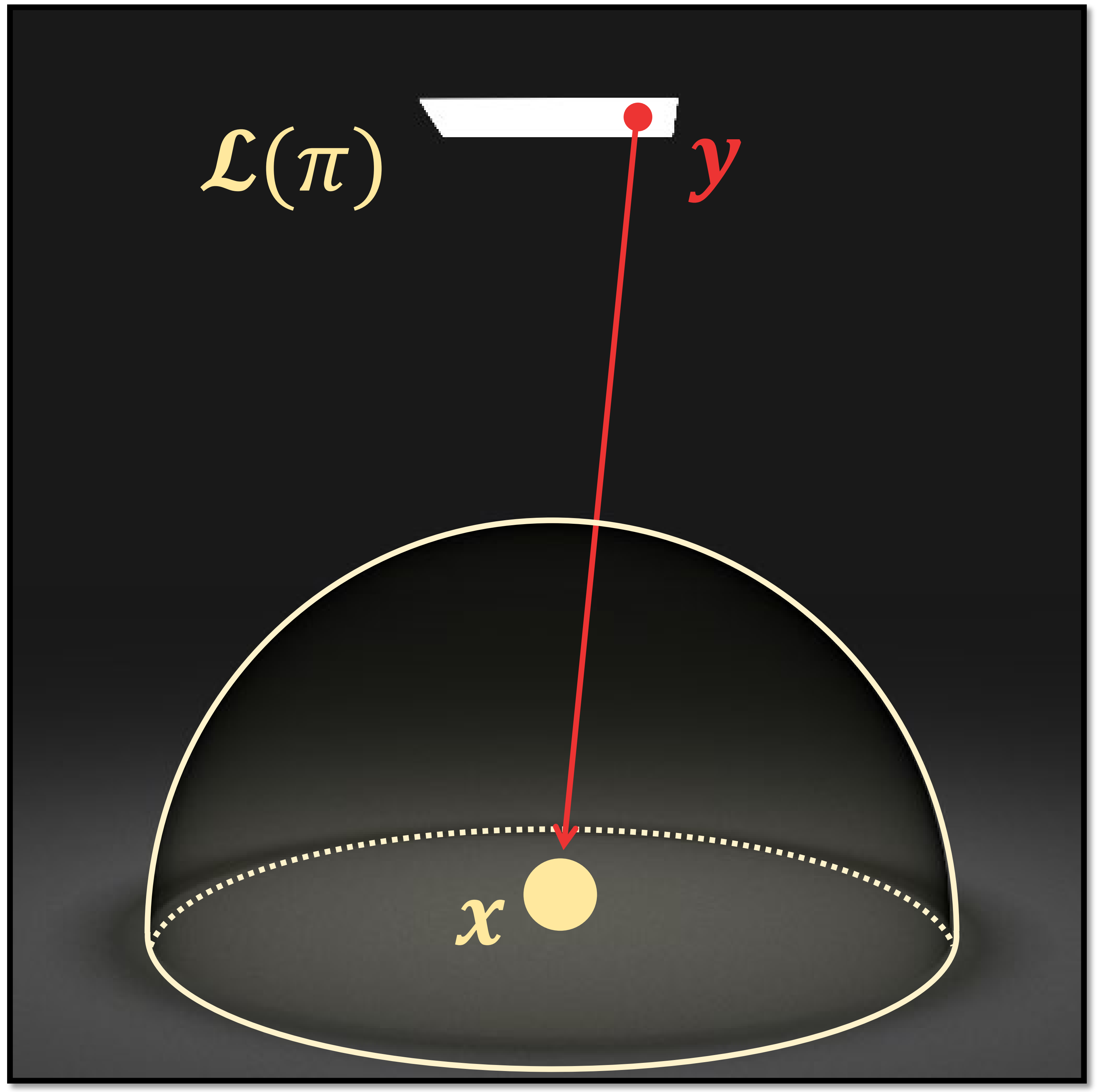
Hemispherical integral



$$I = \int_{\mathbb{H}^2} f(\omega_i)\,\mathrm{d}\sigma(\omega_i)$$

Change of variables

Surface integral

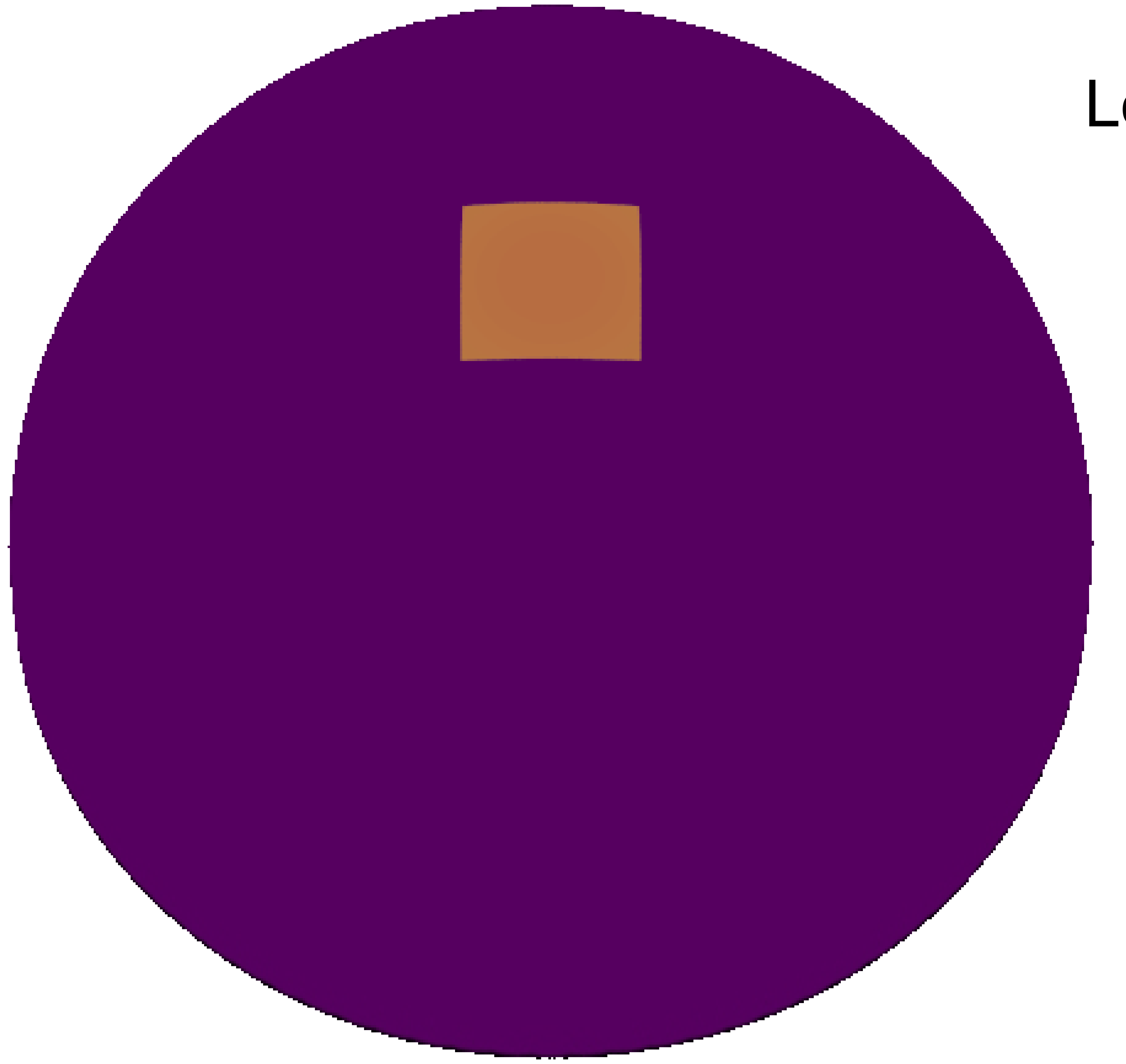


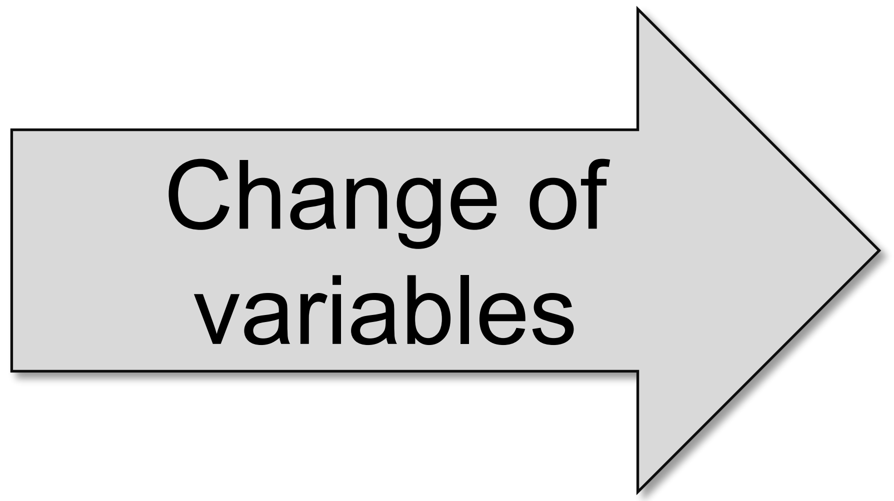
$$I = \int_{\mathcal{L}(\pi)} f(y \to x)\,\boxed{G(x, y)}\,\mathrm{d}A(y)$$

Includes visibility, fall-off, and foreshortening terms

Reparameterizing the direct illumination integral

Hemispherical integral





Surface integral

discontinuous

$$I = \int_{\mathbb{H}^2} f(\omega_i)\, \mathrm{d}\sigma(\omega_i)$$

constant domain

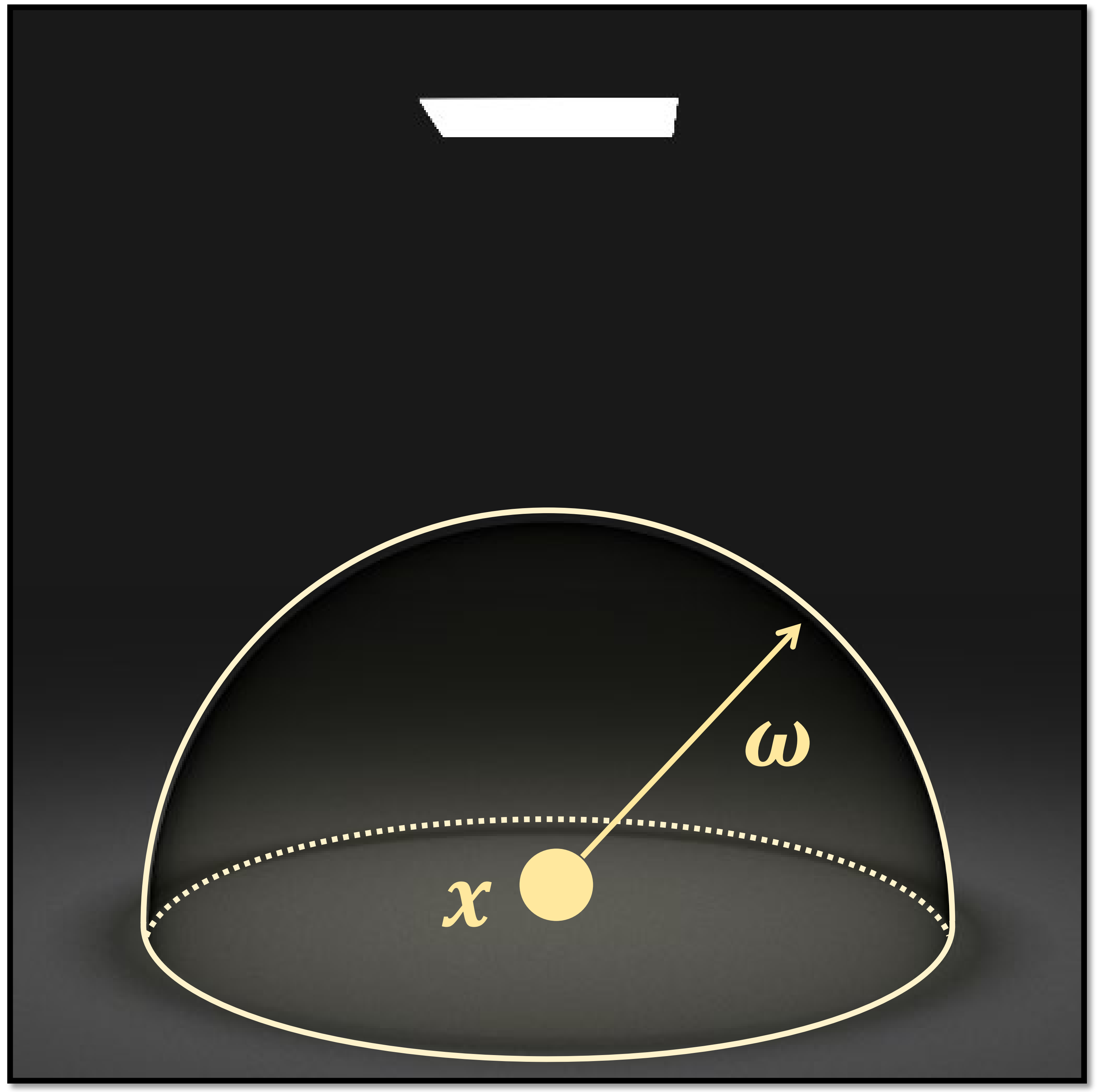
continuous

$$I = \int_{\mathcal{L}(\pi)} f(y \to x)\, G(x, y)\, \mathrm{d}A(y)$$

evolving domain

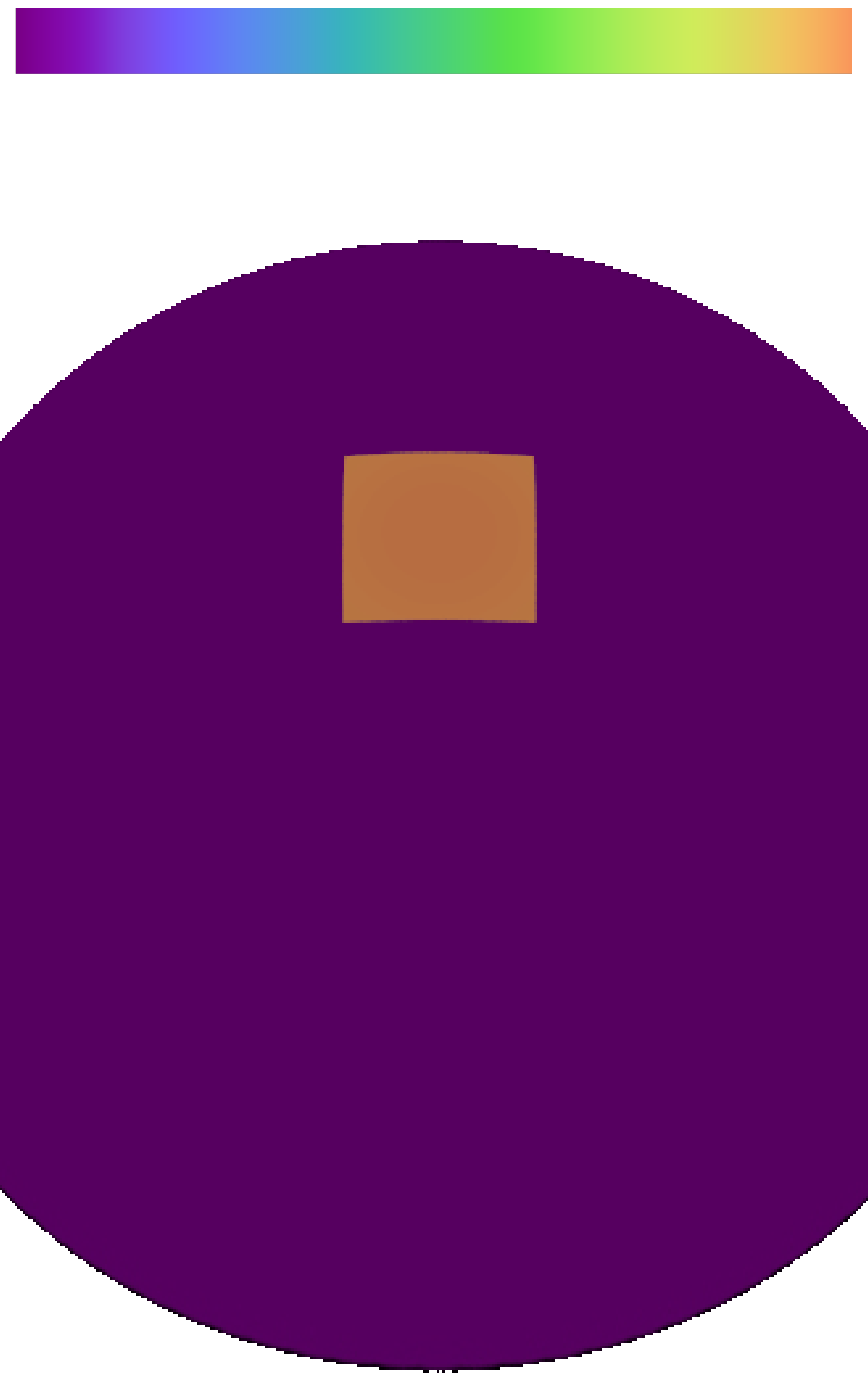
Differentiating the hemispherical integral

π: size of the emitter



$$I = \int_{\mathbb{H}^2} f(\omega_i)\,\mathrm{d}\sigma(\omega_i)$$

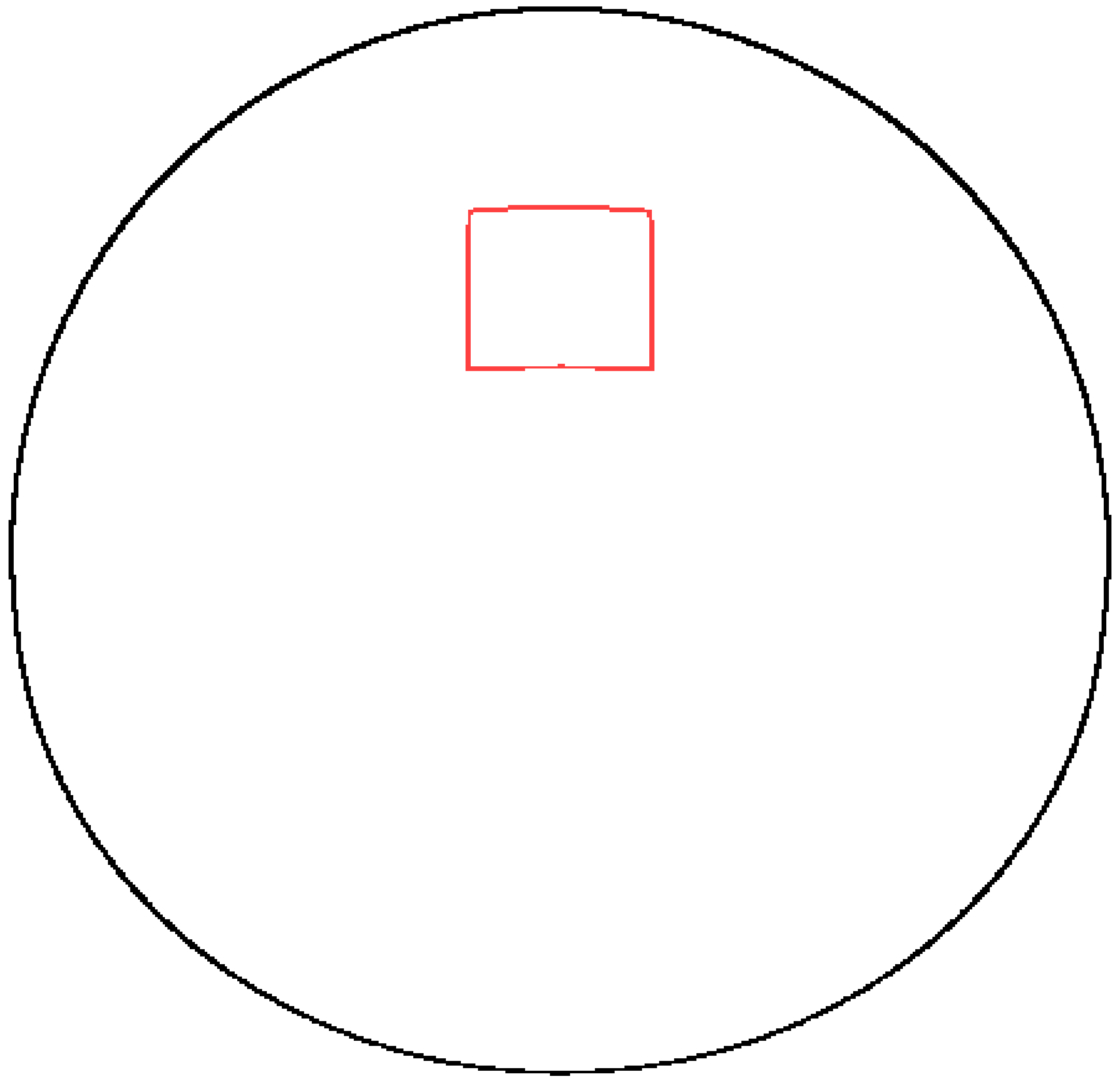
Low High



Differentiation

Reynolds transport theorem

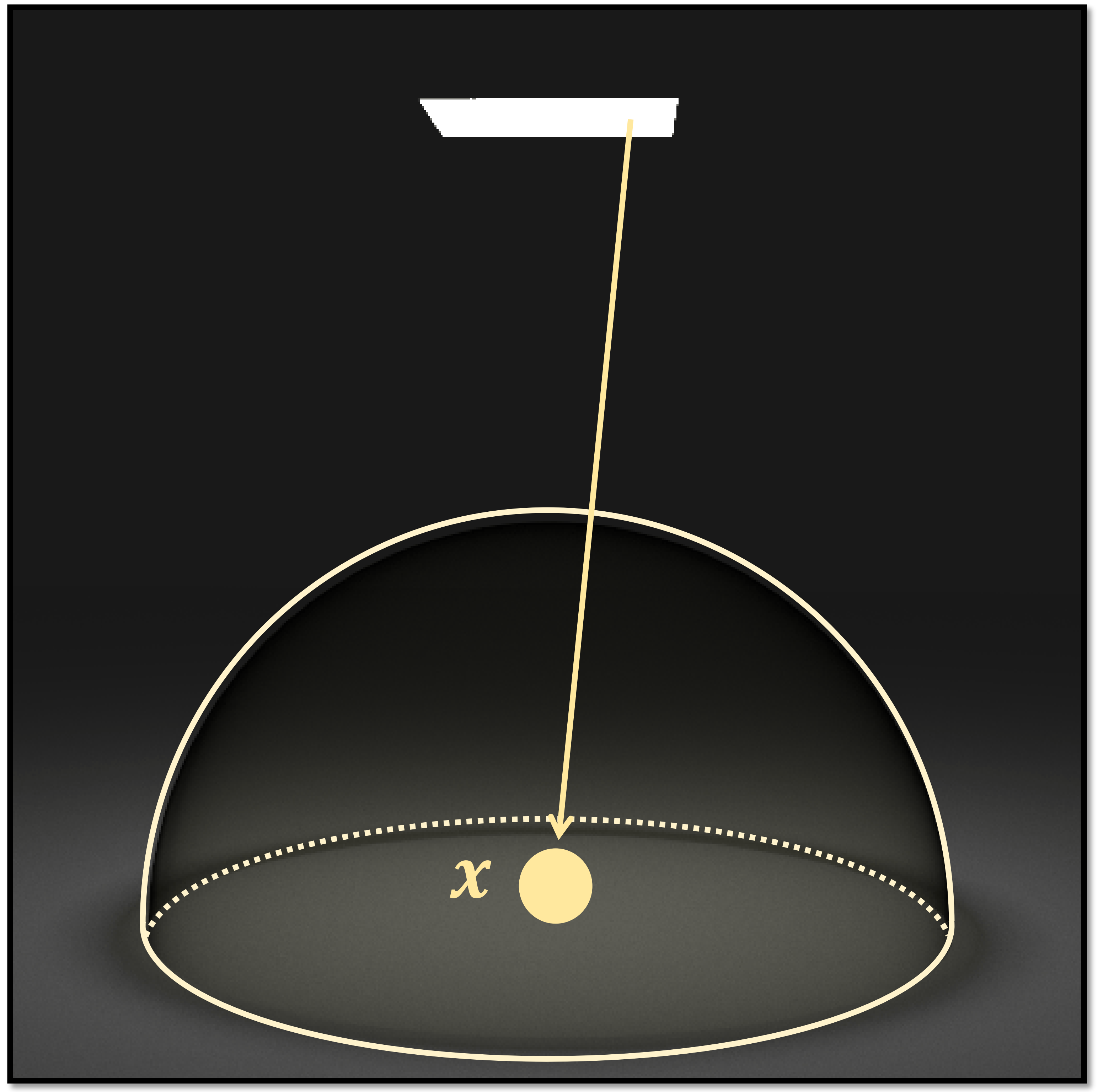
Discontinuities of f



$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \underbrace{\int_{\mathbb{H}^2} \frac{\mathrm{d}(f)}{\mathrm{d}\pi}\,\mathrm{d}\sigma}_{\text{Interior}} + \underbrace{\int_{\partial\mathbb{H}^2} g\,\mathrm{d}l}_{\text{Boundary}}$$

Differentiating the area integral

π: size of the emitter



$$I = \int_{\mathcal{L}(\pi)} f(y \to x) G(x, y) \mathrm{d}A(y)$$

Low High

Boundary of $\mathcal{L}(\pi)$

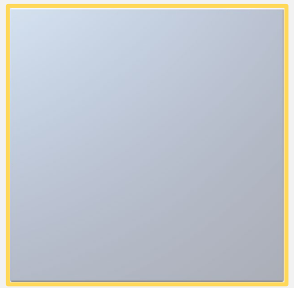
Differentiation

Reynolds transport theorem

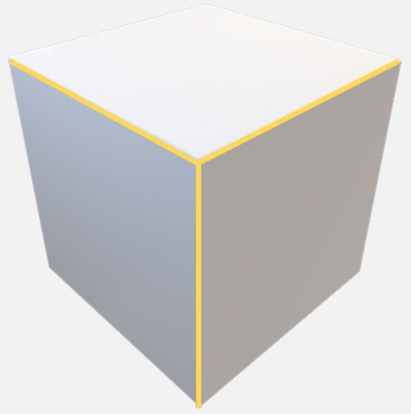
$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \underbrace{\int_{\mathcal{L}(\pi)} \frac{\mathrm{d}(fG)}{\mathrm{d}\pi} \mathrm{d}A}_{\text{Interior}} + \underbrace{\int_{\partial\mathcal{L}(\pi)} g \, \mathrm{d}l}_{\text{Boundary}}$$

Sources of discontinuities

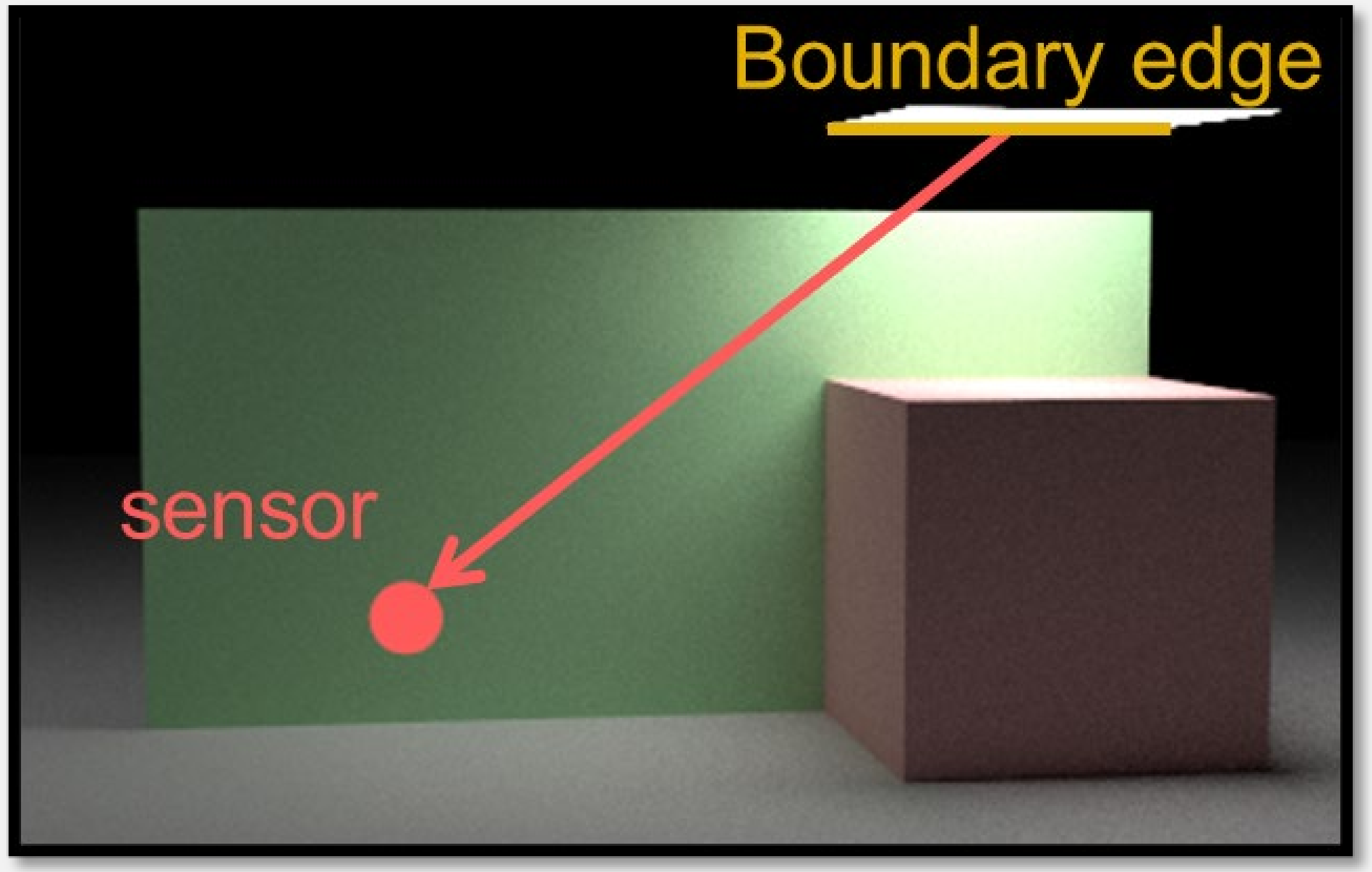
Boundary edge

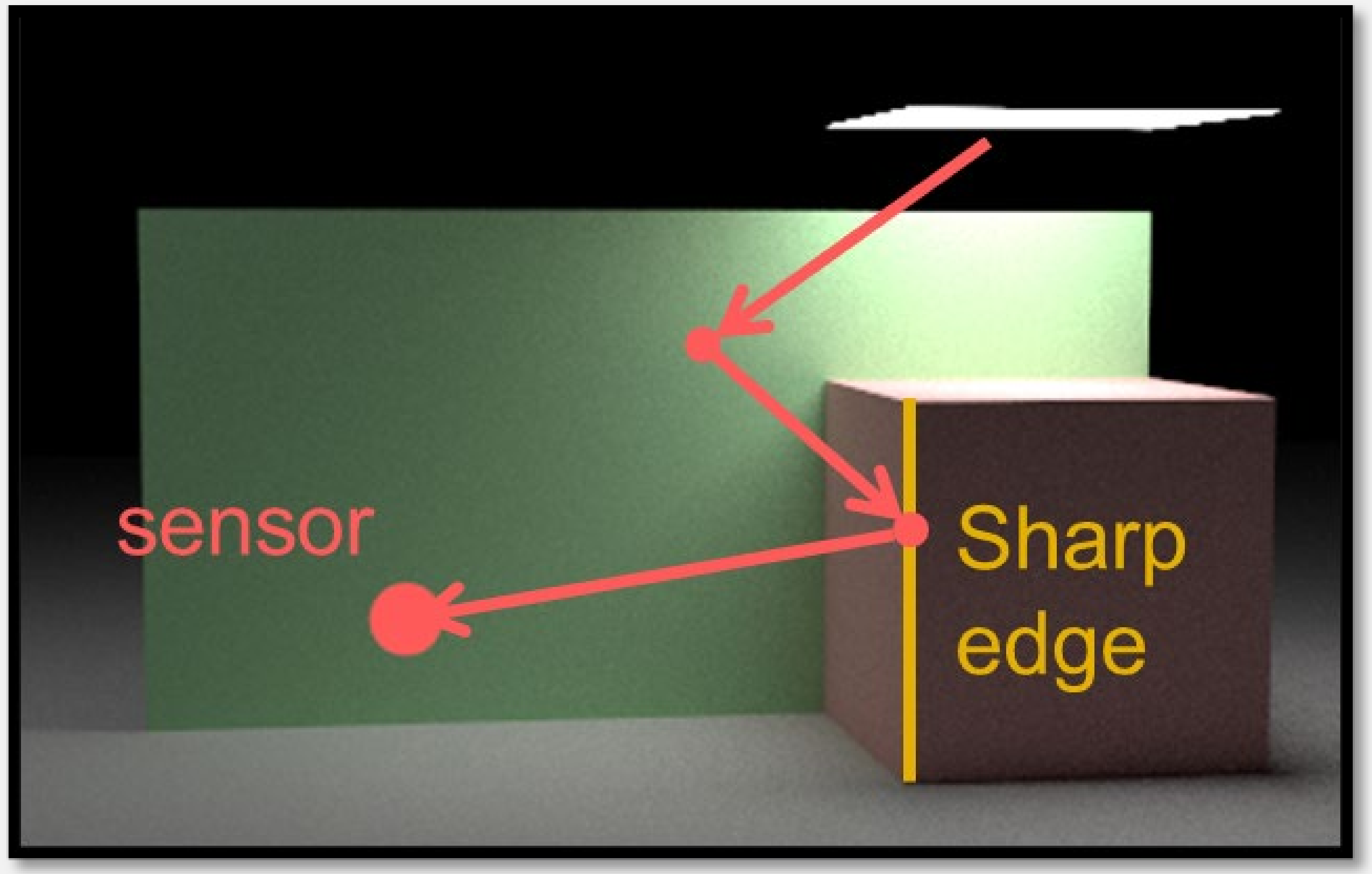


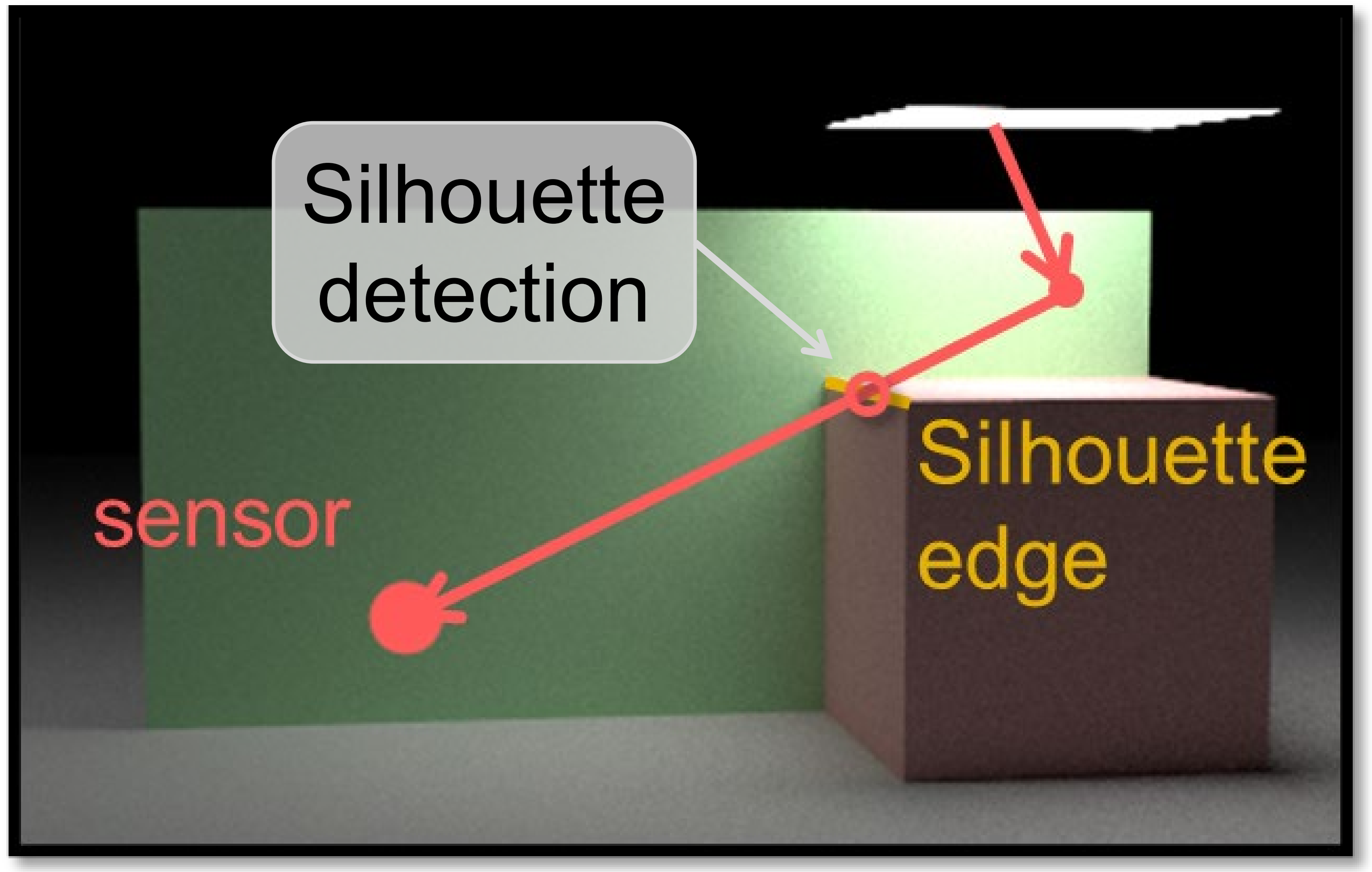
Sharp edge



Silhouette edge



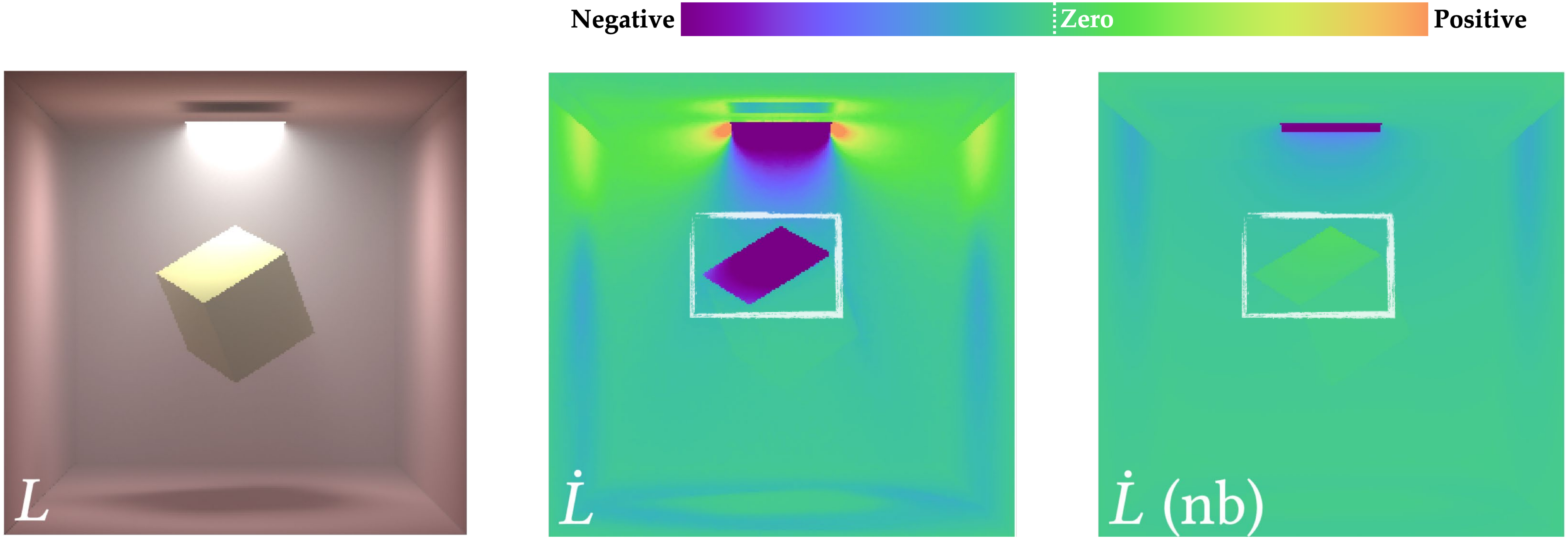




Topology-driven

Visibility-driven

Significance of the boundary integral



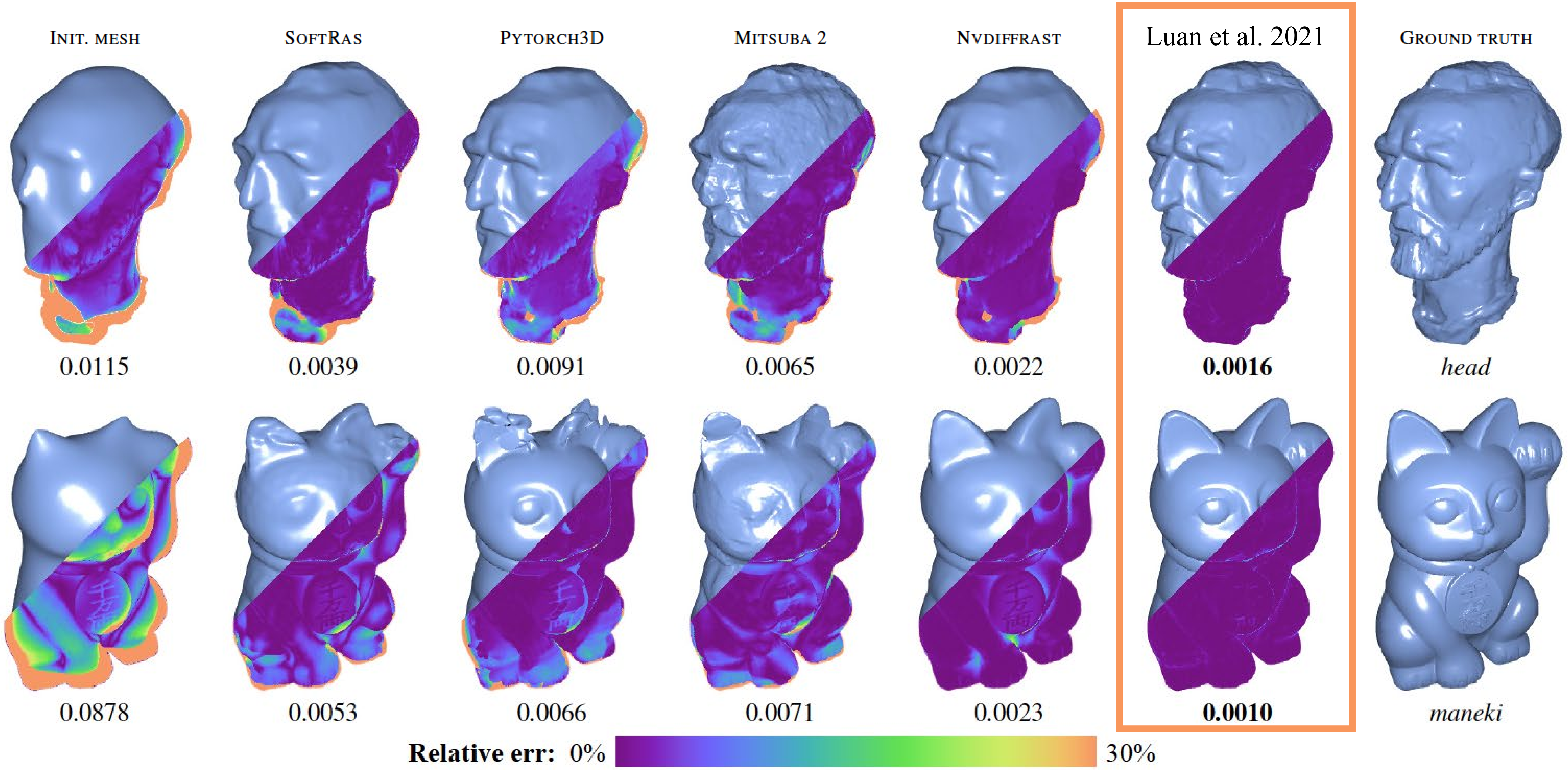
Original image

Derivative image
w.r.t. vertical offset of
the area light and the cube

Derivative image
w/o boundary integral

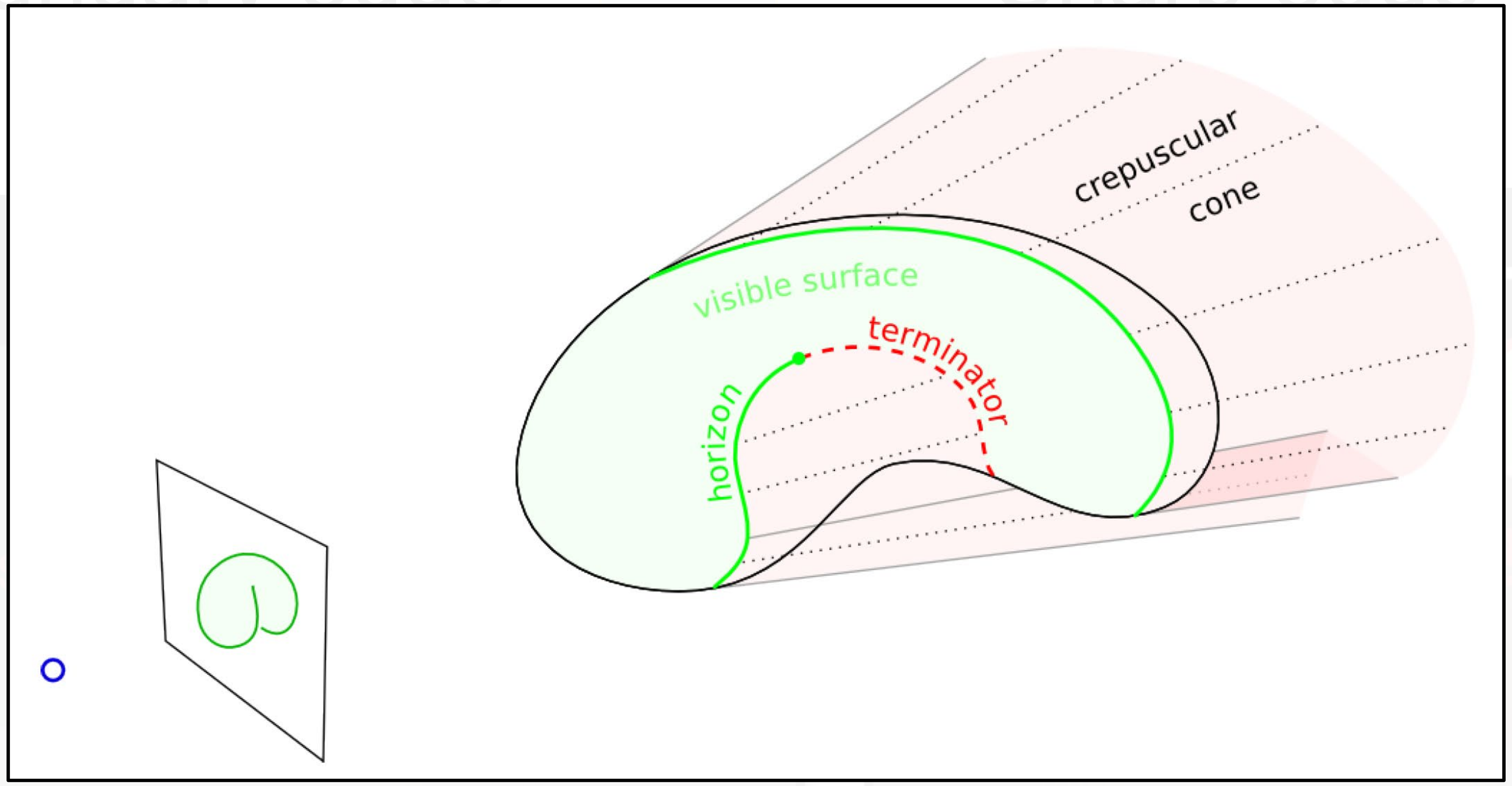
Gradient Accuracy Matters

Inverse-rendering results with *identical* optimization settings



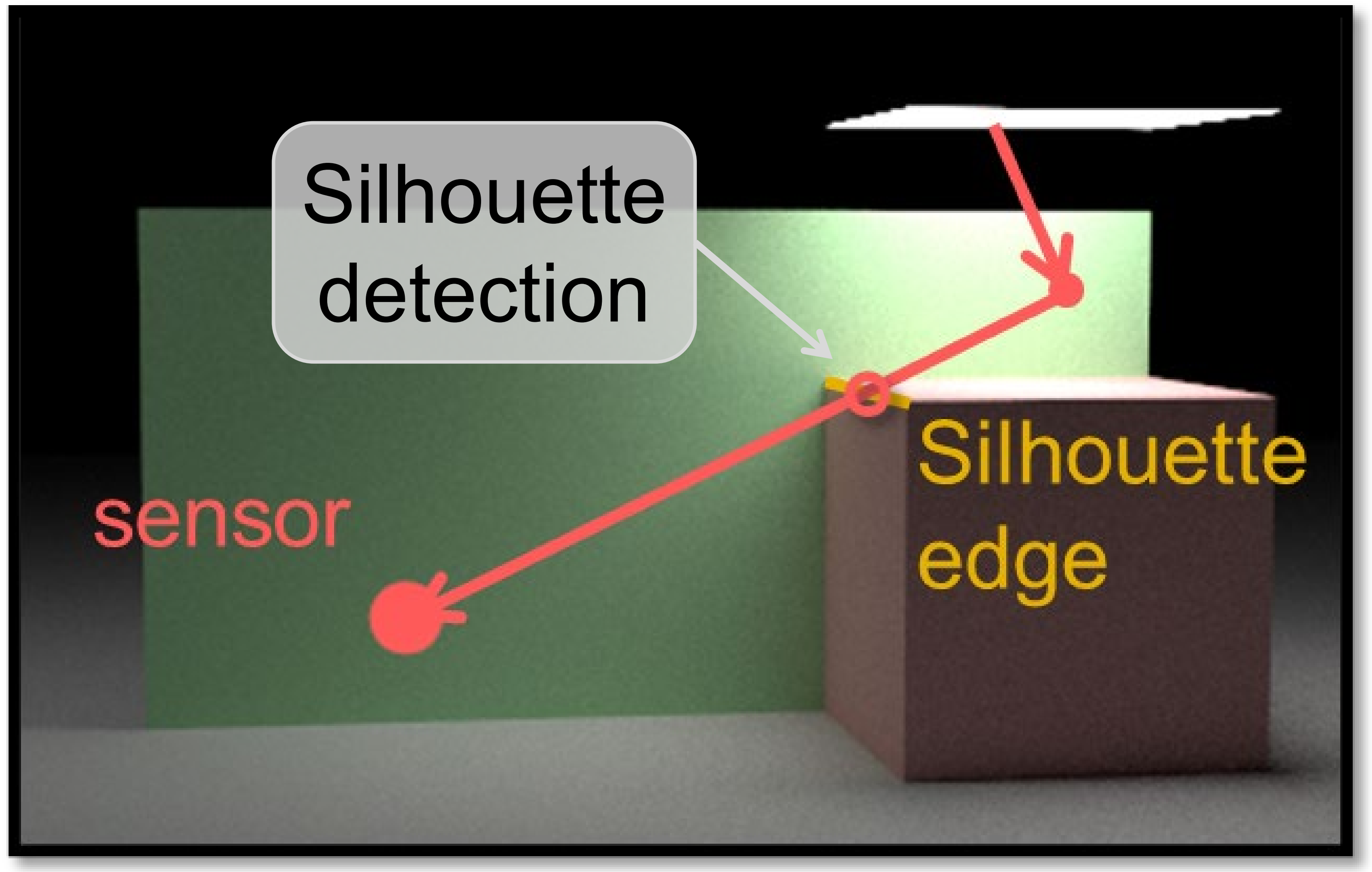
Sources of discontinuities

- We still need to account for visibility discontinuities when using smooth closed surfaces (e.g., neural SDFs)



[Gargallo et al., ICCV 2007]

Silhouette edge



Visibility-driven

Handling Global Illumination

Background: Path Integral for Global Illumination

Pixel value

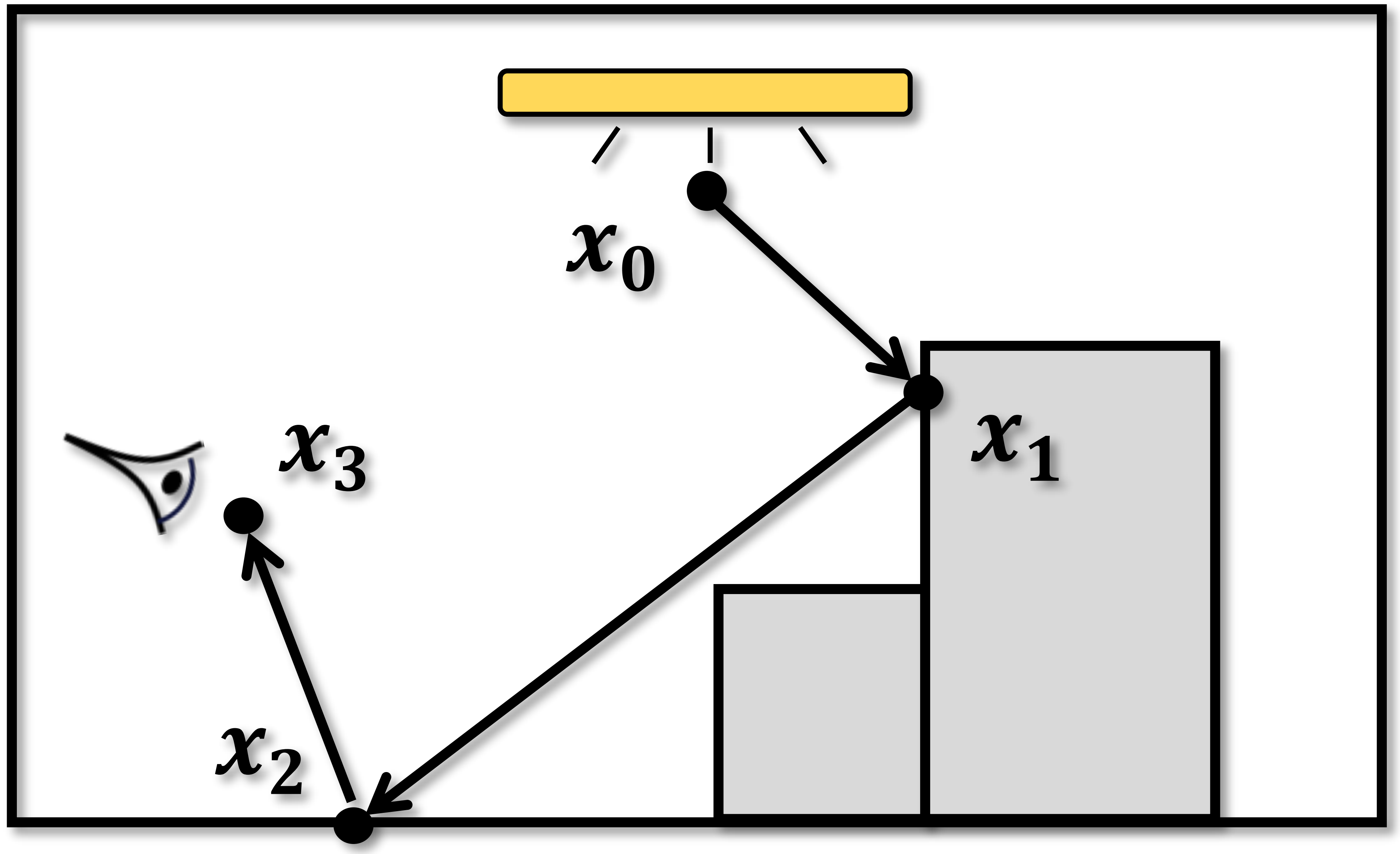
Measurement contribution

$$I = \int_{\Omega} f(\bar{x})\, \mathrm{d}\mu(\bar{x})$$

Path space

Area-product measure

- Introduced by Veach [1997] and extended by Pauly et al. [2000]
- Can capture both surface reflection/refraction and volumetric (i.e., subsurface) scattering
- Theoretical foundation of most modern forward rendering techniques



Light path $\bar{x} = (x_0, x_1, x_2, x_3)$

Background: Estimating Path Integrals

Measurement contribution

Pixel value

$$I = \int_{\Omega} f(\bar{\boldsymbol{x}}) \, \mathrm{d}\mu(\bar{\boldsymbol{x}})$$

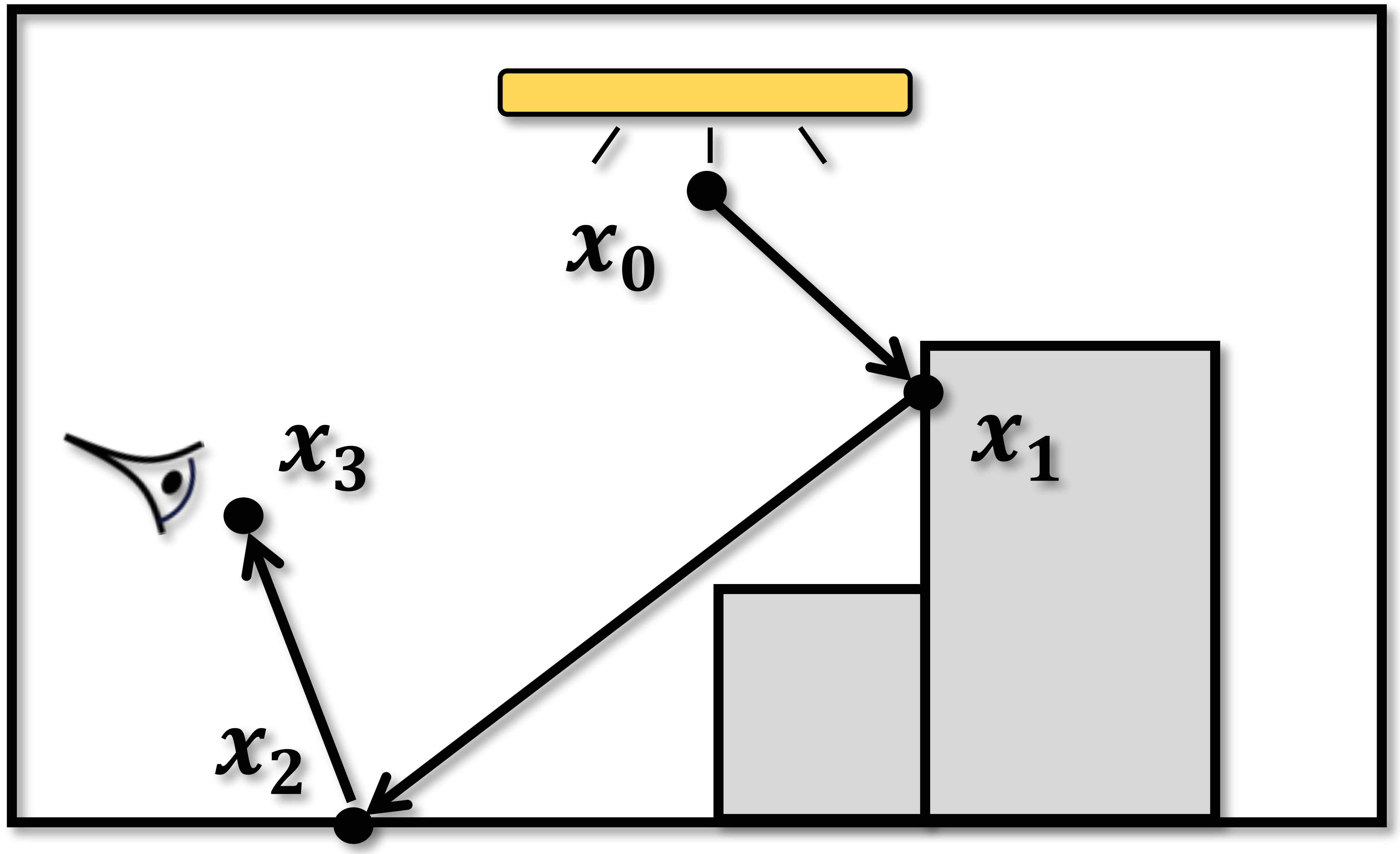
Path space

Area-product measure

Monte Carlo estimator:

$$\langle I \rangle = \frac{f(\bar{\boldsymbol{x}})}{p(\bar{\boldsymbol{x}})}$$

Probability density for sampling path $\bar{\boldsymbol{x}}$



Light path $\bar{\boldsymbol{x}} = (x_0, x_1, x_2, x_3)$

Differential Path Integral

Path-space differentiable rendering [Zhang et al. 2020, 2021]

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right) = \int_{\Omega} \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}}) + \int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})$$

Interior integral **Boundary** integral

We now derive $\partial I_N/\partial\pi$ in Eq. (25) using the recursive relations provided by Eqs. (21) and (24). Let

$$h_n^{(0)} := \left[\prod_{n'=n+1}^N g(\boldsymbol{x}_{n'};\boldsymbol{x}_{n'-2},\boldsymbol{x}_{n'-1})\right] W_e(\boldsymbol{x}_N \to \boldsymbol{x}_{N-1}), \quad (52)$$

$$h_n^{(1)} := \sum_{n'=n+1}^N \kappa(\boldsymbol{x}_{n'})\,V(\boldsymbol{x}_{n'}), \quad (53)$$

$$\Delta h_{n,n'}^{(0)} := h_n^{(0)}\,\Delta g(\boldsymbol{x}_{n'};\boldsymbol{x}_{n'-2},\boldsymbol{x}_{n'-1})/g(\boldsymbol{x}_{n'};\boldsymbol{x}_{n'-2},\boldsymbol{x}_{n'-1}), \quad (54)$$

for $0 \le n < n' \le N$. We omit the dependencies of $h_n^{(0)}$, $h_n^{(1)}$, and $\Delta h_{n,n'}^{(0)}$ on $\boldsymbol{x}_{n+1},\ldots,\boldsymbol{x}_N$ for notational convenience.

We now show that, for all $0 \le n < N$, it holds that

$$h_n(\boldsymbol{x}_n;\boldsymbol{x}_{n-1}) = \int_{\mathcal{M}^{N-n}} h_n^{(0)} \prod_{n'=n+1}^N \mathrm{d}A(\boldsymbol{x}_{n'}), \quad (55)$$

and

$$\dot{h}_n(\boldsymbol{x}_n;\boldsymbol{x}_{n-1}) = \int_{\mathcal{M}^{N-n}} \left[\left(h_n^{(0)}\right)^{\boldsymbol{\cdot}} - h_n^{(0)} h_n^{(1)}\right] \prod_{n'=n+1}^N \mathrm{d}A(\boldsymbol{x}_{n'}) + \sum_{n'=n+1}^N \int \Delta h_{n,n'}^{(0)}\, V_{\overline{\partial\mathcal{M}}_{n'}}(\boldsymbol{x}_{n'})\,\mathrm{d}\ell(\boldsymbol{x}_{n'}) \prod_{\substack{n<i\le N\\ i\ne n'}} \mathrm{d}A(\boldsymbol{x}_i), \quad (56)$$

where the integral domain of the second term on the right-hand side, which is omitted for notational clarity, is $\mathcal{M}(\pi)$ for each $\boldsymbol{x}_i$ with $i \ne n'$ and $\overline{\partial\mathcal{M}}_{n'}(\pi)$, which depends on $\boldsymbol{x}_{n'-1}$, for $\boldsymbol{x}_{n'}$.

It is easy to verify that Eqs. (55) and (56) hold for $n = N-1$. We now show that, if they hold for some $0 < n < N$, then it is also the case for $n-1$. Let $g_{n-1} := g(\boldsymbol{x}_n;\boldsymbol{x}_{n-2},\boldsymbol{x}_{n-1})$ for all $0 < n \le N$. Then,

$$h_{n-1}(\boldsymbol{x}_{n-1};\boldsymbol{x}_{n-2}) = \int_{\mathcal{M}} g_{n-1} \int_{\mathcal{M}^{N-n}} h_n^{(0)} \prod_{n'=n+1}^N \mathrm{d}A(\boldsymbol{x}_{n'})\,\mathrm{d}A(\boldsymbol{x}_n) = \int_{\mathcal{M}^{N-n+1}} h_{n-1}^{(0)} \prod_{n'=n}^N \mathrm{d}A(\boldsymbol{x}_{n'}), \quad (57)$$

and

$$\begin{aligned}
&\dot{h}_{n-1}(\boldsymbol{x}_{n-1};\boldsymbol{x}_{n-2})\\
&= \int_{\mathcal{M}} \left[\dot{g}_{n-1}\,h_n + g_{n-1}(\dot{h}_n - h_n\,\kappa(\boldsymbol{x}_n)\,V(\boldsymbol{x}_n))\right] \mathrm{d}A(\boldsymbol{x}_n)\\
&\quad + \int_{\overline{\partial\mathcal{M}}_n} \Delta g_{n-1}\,h_n\,V_{\overline{\partial\mathcal{M}}_n}\,\mathrm{d}\ell(\boldsymbol{x}_n)\\
&= \int_{\mathcal{M}^{N-n+1}} \left\{\dot{g}_{n-1}\,h_n^{(0)} + g_{n-1}\left[\left(h_n^{(0)}\right)^{\boldsymbol{\cdot}} - h_n^{(0)} h_{n-1}^{(1)}\right]\right\} \prod_{n'=k}^N \mathrm{d}A(\boldsymbol{x}_{n'})\\
&\quad + \sum_{n'=n+1}^N \int g_{n-1}\,\Delta h_{n,n'}^{(0)}\,V_{\overline{\partial\mathcal{M}}_{n'}}(\boldsymbol{x}_{n'})\,\mathrm{d}\ell(\boldsymbol{x}_{n'}) \prod_{\substack{n\le i\le N\\ i\ne n'}} \mathrm{d}A(\boldsymbol{x}_i)\\
&\quad + \int \Delta g_{n-1}\,h_n^{(0)}\,V_{\overline{\partial\mathcal{M}}_n}\,\mathrm{d}\ell(\boldsymbol{x}_n) \prod_{n'=n+1}^N \mathrm{d}A(\boldsymbol{x}_{n'})\\
&= \int_{\mathcal{M}^{N-n+1}} \left[\left(h_{n-1}^{(0)}\right)^{\boldsymbol{\cdot}} - h_{n-1}^{(0)} h_{n-1}^{(1)}\right] \prod_{n'=n}^N \mathrm{d}A(\boldsymbol{x}_{n'})\\
&\quad + \sum_{n'=n}^N \int \Delta h_{n-1,n'}^{(0)}\,V_{\overline{\partial\mathcal{M}}_{n'}}(\boldsymbol{x}_{n'})\,\mathrm{d}\ell(\boldsymbol{x}_{n'}) \prod_{\substack{n\le i\le N\\ i\ne n'}} \mathrm{d}A(\boldsymbol{x}_i). \quad (58)
\end{aligned}$$

Thus, using mathematical induction, we know that Eqs. (55) and (56) hold for all $0 \le n < N$.

Notice that $h_0^{(0)} = f$ and $\Delta h_{0,n'}^{(0)} = \Delta f_{n'}$, where $\Delta f_{n'}$ follows the definition in Eq. (28). Letting $n = 0$ in Eq. (56) yields

$$\dot{h}_0(\boldsymbol{x}_0) = \int_{\mathcal{M}^N} \left[\dot{f}(\bar{\boldsymbol{x}}) - f(\bar{\boldsymbol{x}}) \sum_{n'=1}^N \kappa(\boldsymbol{x}_{n'})\,V(\boldsymbol{x}_{n'})\right] \prod_{n'=1}^N \mathrm{d}A(\boldsymbol{x}_{n'}) + \sum_{n'=1}^N \int \Delta f_{n'}(\bar{\boldsymbol{x}})\,V_{\overline{\partial\mathcal{M}}_{n'}}\,\mathrm{d}\ell(\boldsymbol{x}_{n'}) \prod_{\substack{0<i\le N\\ i\ne n'}} \mathrm{d}A(\boldsymbol{x}_i). \quad (59)$$

Lastly, based on the assumption that h_0 is continuous in $\boldsymbol{x}_0$, Eq. (25) can be obtained by differentiating Eq. (23):

$$\begin{aligned}
\frac{\partial I_N}{\partial\pi} &= \frac{\partial}{\partial\pi}\int_{\mathcal{M}} h_0(\boldsymbol{x}_0)\,\mathrm{d}A(\boldsymbol{x}_0)\\
&= \int_{\mathcal{M}} \left[\dot{h}_0(\boldsymbol{x}_0) - h_0(\boldsymbol{x}_0)\,\kappa(\boldsymbol{x}_0)\,V(\boldsymbol{x}_0)\right] \mathrm{d}A(\boldsymbol{x}_0)\\
&\quad + \int_{\overline{\partial\mathcal{M}}_0} h_0(\boldsymbol{x}_0)\,V_{\overline{\partial\mathcal{M}}_0}(\boldsymbol{x}_0)\,\mathrm{d}\ell(\boldsymbol{x}_0) \qquad (60)\\
&= \int_{\Omega_N} \left[\dot{f}(\bar{\boldsymbol{x}}) - f(\bar{\boldsymbol{x}}) \sum_{K=0}^N \kappa(\boldsymbol{x}_K)\,V(\boldsymbol{x}_K)\right] \mathrm{d}\mu(\bar{\boldsymbol{x}})\\
&\quad + \sum_{K=0}^N \int_{\Omega_{N,K}} \Delta f_K(\bar{\boldsymbol{x}})\,V_{\overline{\partial\mathcal{M}}_K}\,\mathrm{d}\mu'_{N,K}(\bar{\boldsymbol{x}}).
\end{aligned}$$

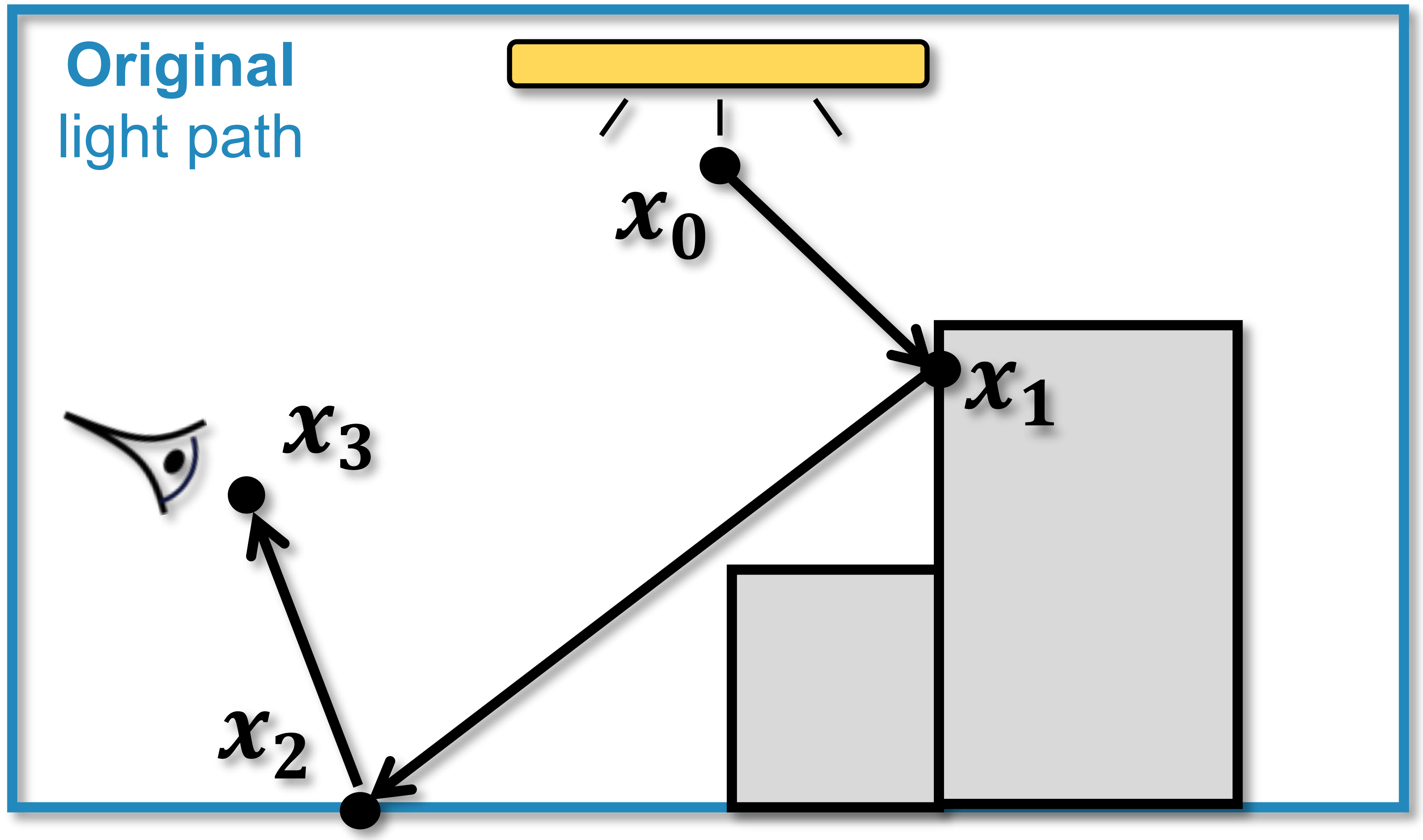
(The full derivation is quite involved…)

Differential Path Integral

Path-space differentiable rendering [Zhang et al. 2020, 2021]

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right) = \boxed{\int_{\Omega} \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})} + \int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})$$

Interior integral



Interior integral

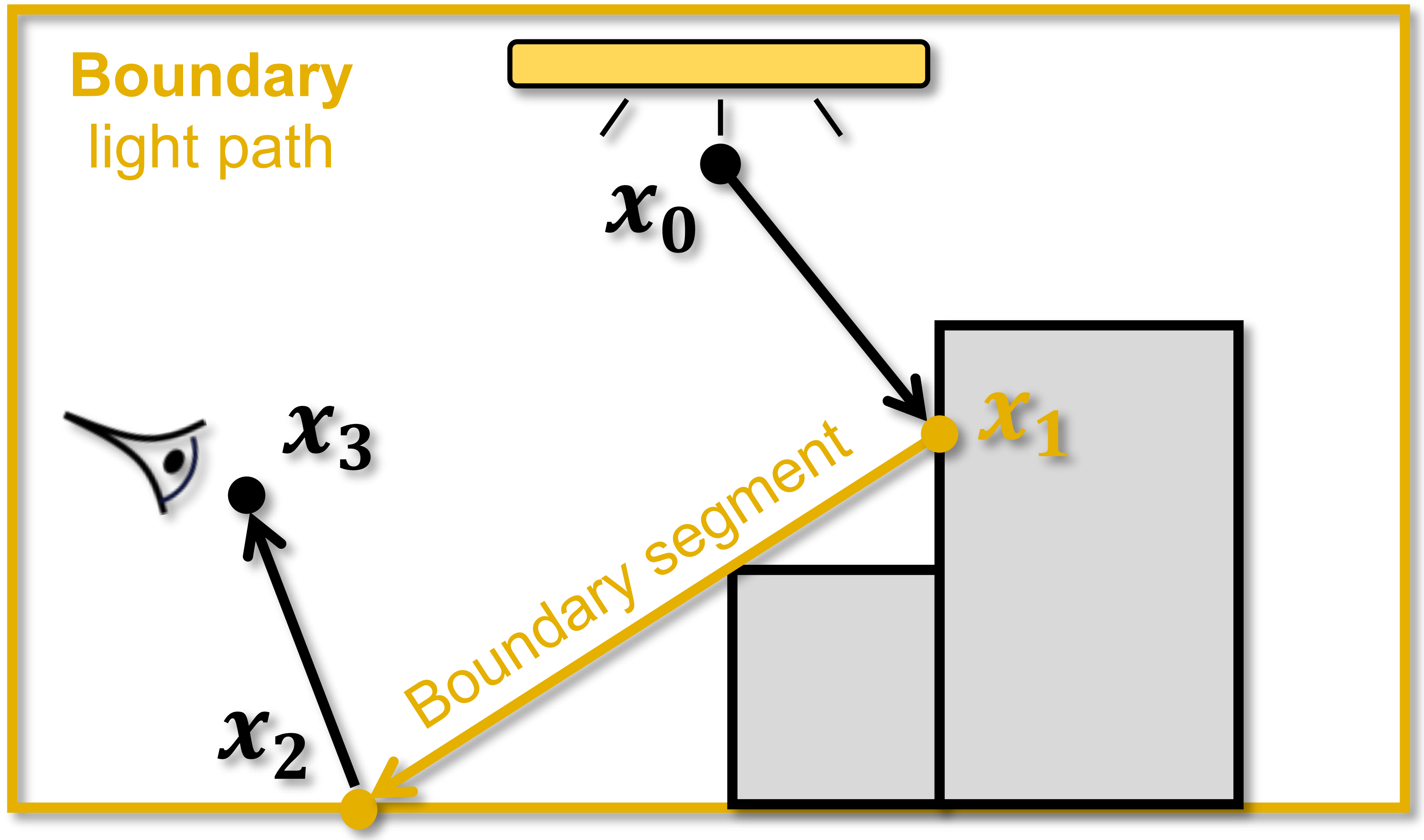
- Defined on the ordinary path space $\boldsymbol{\Omega}$
- The integrand $\dot{f}$ can be obtained by differentiating the ordinary *measurement contribution function f*

Differential Path Integral

Path-space differentiable rendering [Zhang et al. 2020, 2021]

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right) = \int_{\Omega} \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}}) + \boxed{\int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})}$$

Boundary integral



Boundary integral

- Defined on the boundary path space $\partial\Omega$
- A **boundary** light path is the same as an original one except having exactly one boundary segment

Differential Path Integral

Path-space differentiable rendering [Zhang et al. 2020, 2021]

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right) = \underbrace{\int_{\Omega} \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})}_{\text{Interior integral}} + \underbrace{\int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})}_{\text{Boundary integral}}$$

Physics-based **differentiable** rendering generally requires estimating **both integrals**

Challenges:

- Differentiating f w.r.t. many parameters (interior)
- Handling discontinuities (boundary)

Differential Interior Path Integral

Path-space differentiable rendering [Zhang et al. 2020, 2021]

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right) = \underbrace{\boxed{\int_{\Omega} \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})}}_{\textbf{Interior}\text{ integral}} + \int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})$$

- Computing $\dot{f}$ requires differentiating f w.r.t. θ
- This can be done via **automatic differentiation**, but …
  - We have many (e.g., 10^6) path integrals to evaluate (one per pixel)
  - There can be many (e.g., 10^6) parameters
  - Huge gradient matrices (e.g., with 10^{12} entries), not enough memory!

Specialized *computational differentiation* methods have been developed [Nimier-David et al. 2020, Vicini et al. 2021]

Differential Path Integral

Path-space differentiable rendering [Zhang et al. 2020, 2021]

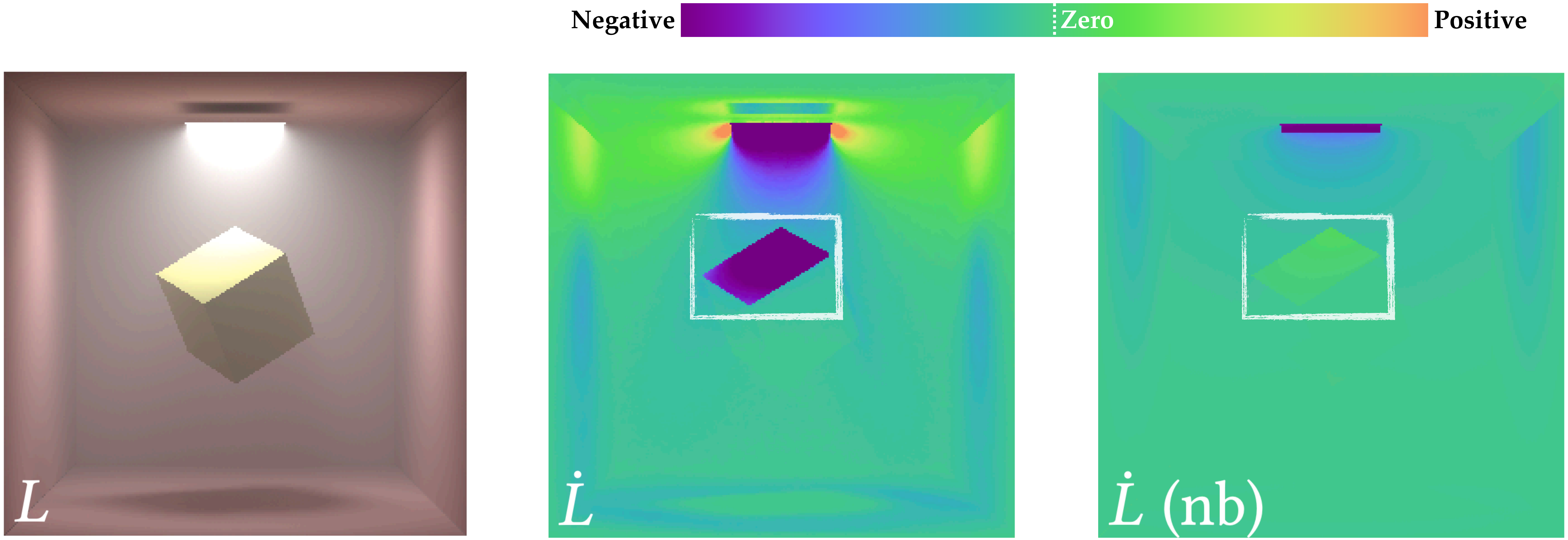
$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right) = \underbrace{\int_{\Omega} \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})}_{\textbf{Interior}\text{ integral}} + \underbrace{\int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})}_{\textbf{Boundary}\text{ integral}}$$

Physics-based **differentiable** rendering generally requires estimating **both integrals**

Challenges:

- Differentiating f w.r.t. many parameters (interior)
- Handling discontinuities (boundary)

Recap: Significance of the Boundary Integral



Original image

Derivative image
w.r.t. vertical offset of
the area light and the cube

Derivative image
w/o boundary integral

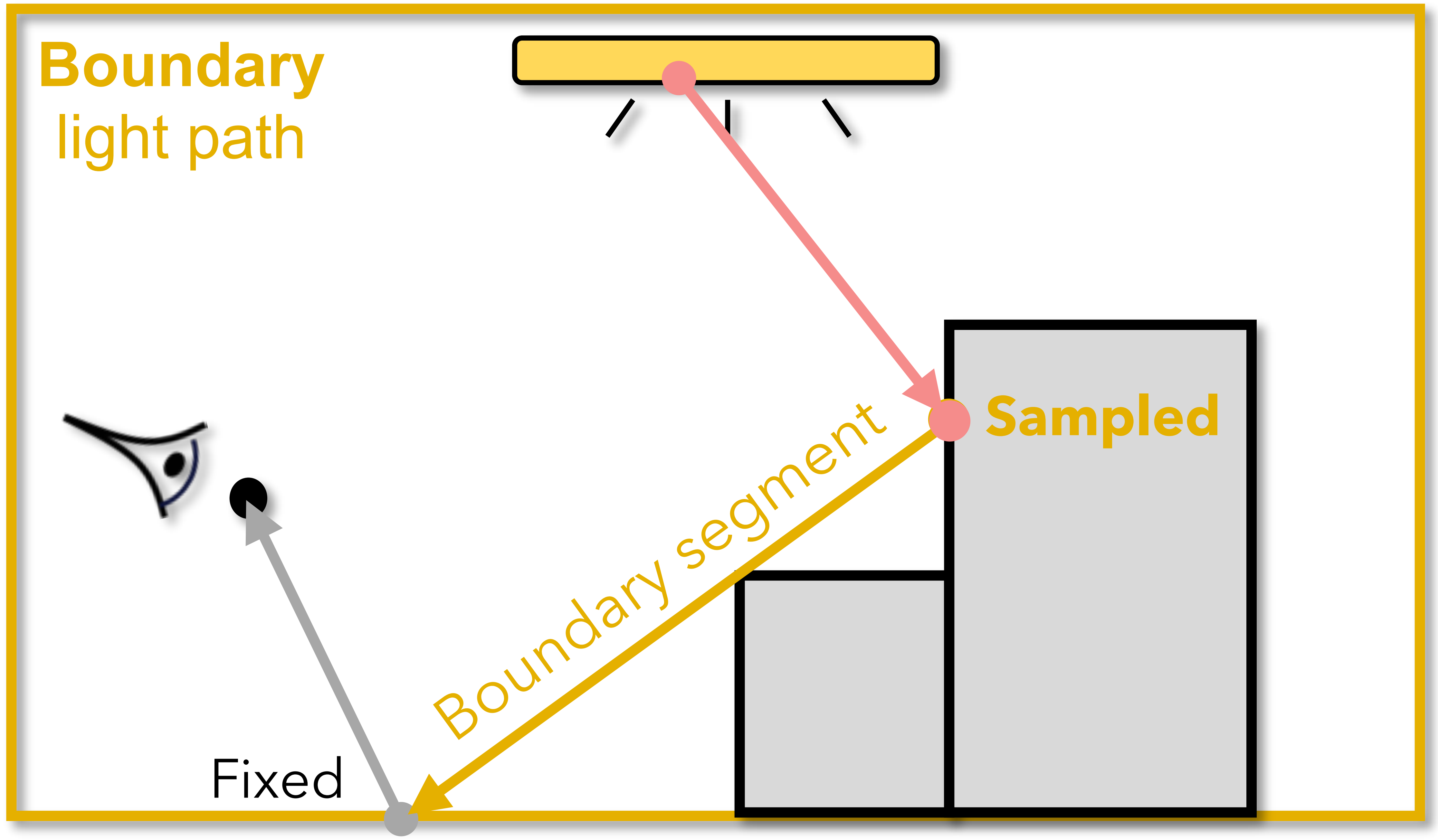
Handling Discontinuities

- **Objective:** estimating the integral over all **boundary** light paths (that are the same as an original one except having exactly one boundary segment)

- (Solution 1) Monte Carlo **edge sampling**
  - Introduced by Li et al. [2018]
  - Also used by Zhang et al. [2019]

To sample a **boundary** segment:

- Fix one endpoint
- Sample the other from **discontinuity boundaries**

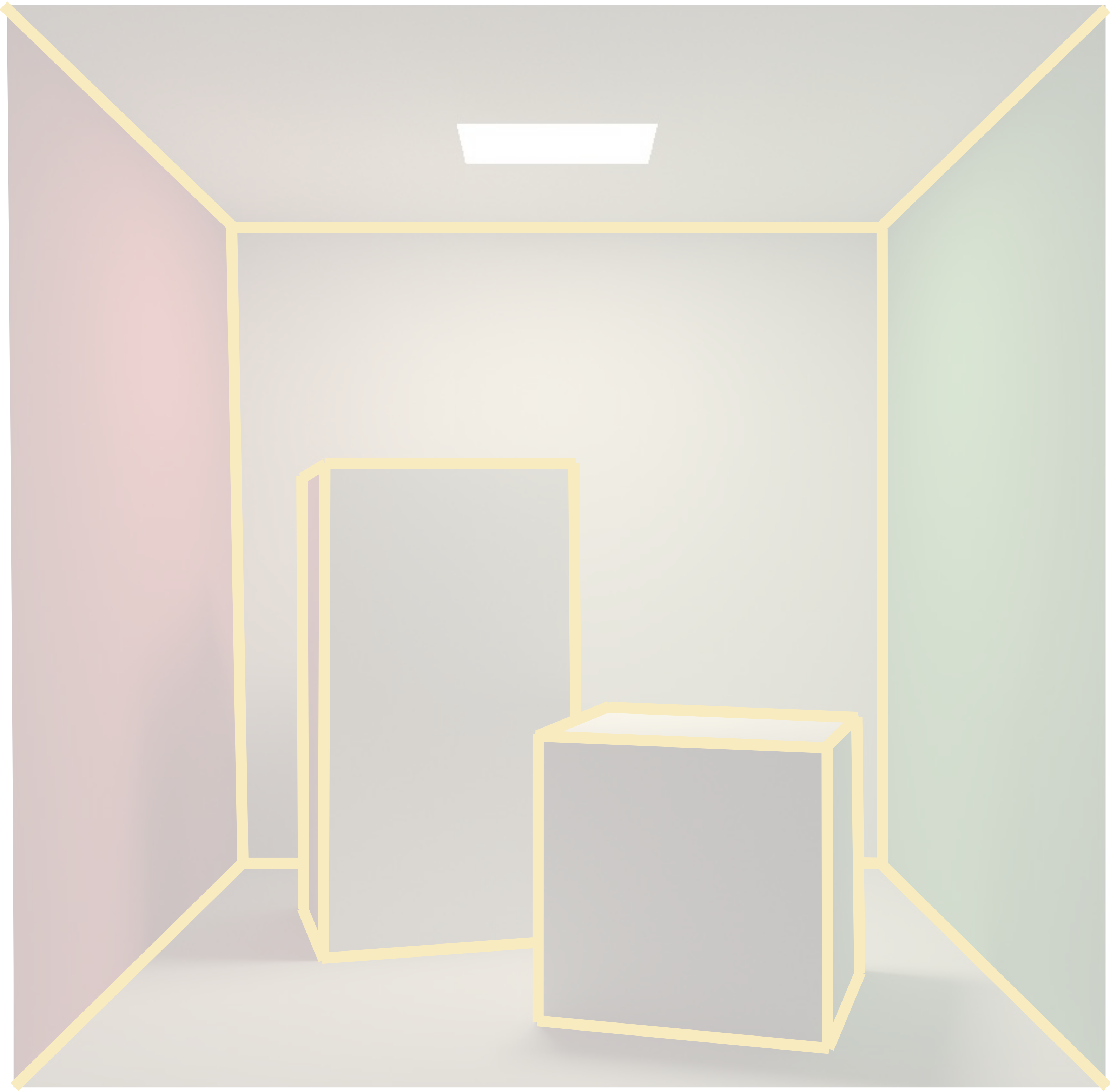


Recap: Sources of Discontinuities

Boundary edges

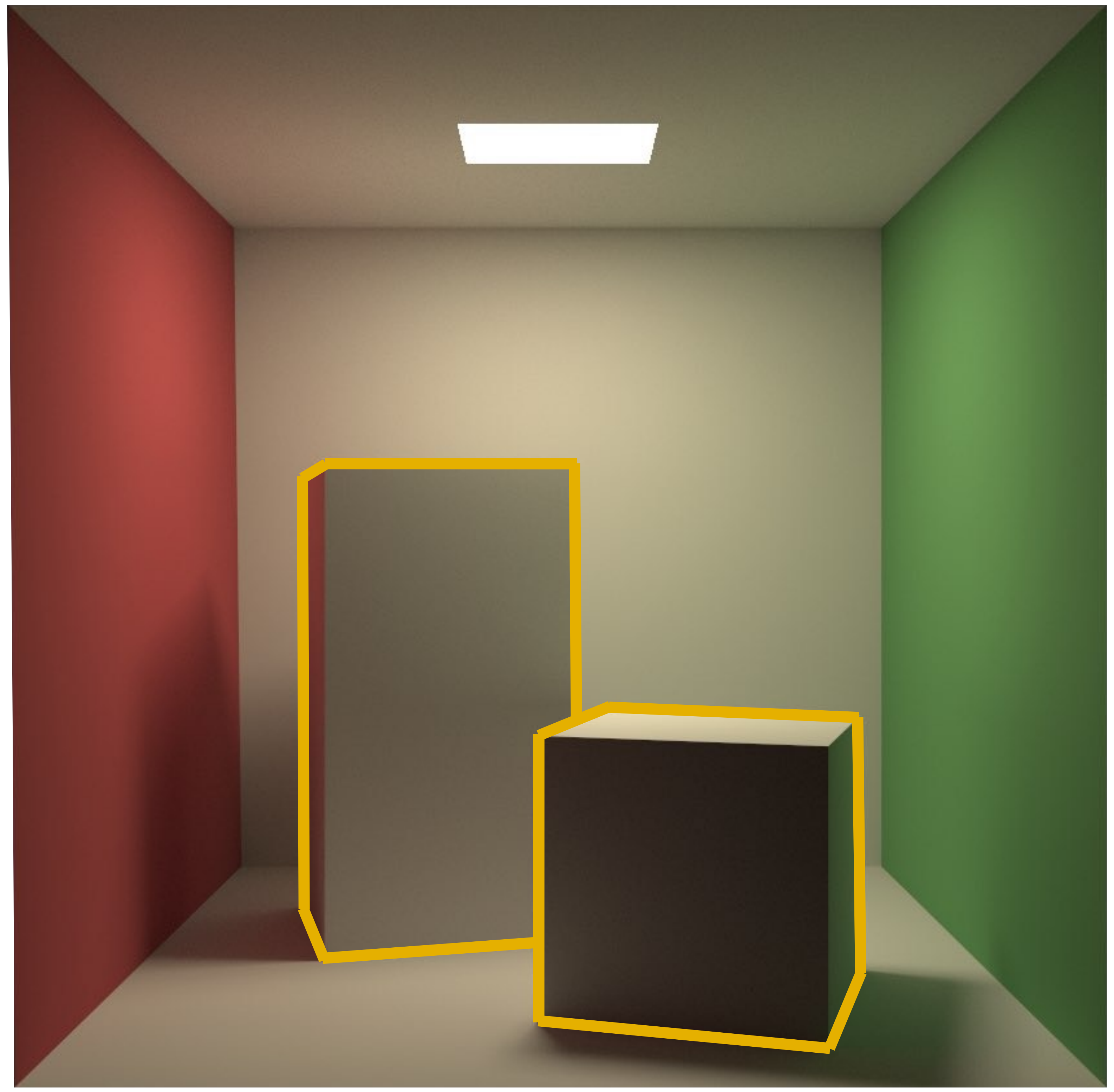
(Topological) boundary of an object

Sharp edges



Surface-normal discontinuities (e.g., face edges)

Silhouette edges



View-dependent object silhouettes

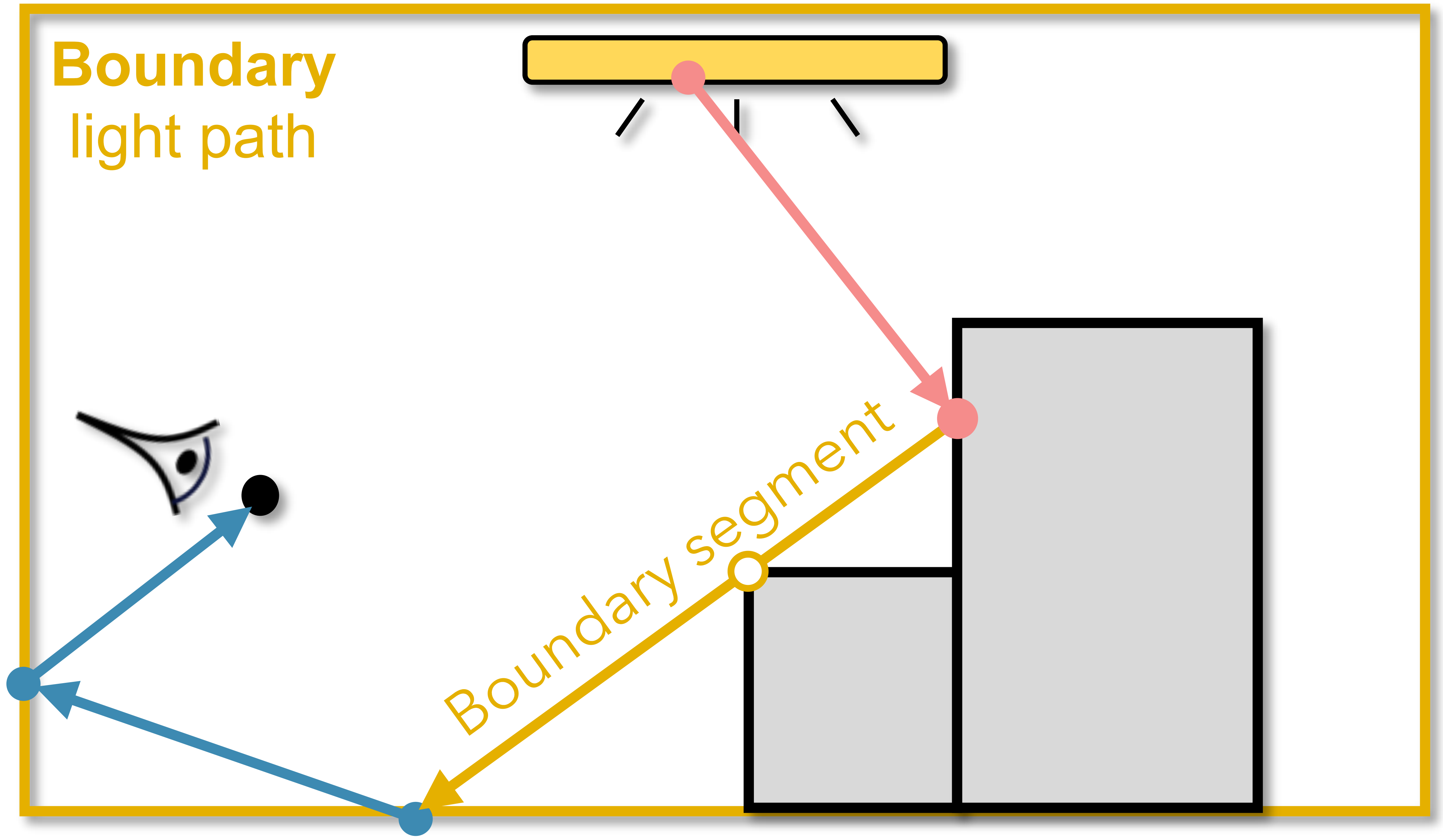
Handling Discontinuities

- **Objective:** estimating the integral over all **boundary** light paths (that are the same as an original one except having exactly one boundary segment)

- (Solution 2) **multi-directional sampling** of **boundary** paths
  - Enabled by the path-integral formulation [Zhang et al. 2020, 2021]

To sample a **boundary** path:

- Start from the **boundary** segment in the middle
- Then construct the **source** and **sensor** subpaths



Physics-Based Differentiable Rendering Algorithms

- **Boundary-sampling differentiable** rendering
  - Path tracing with **edge sampling** [Li et al. 2018, Zhang et al. 2019] (solution 1)
  - **Path-space differentiable** rendering [Zhang et al. 2020, 2021] (solution 2)

To be discussed next

- **Area-sampling differentiable** rendering
  - Avoids boundary integrals altogether (Sai will cover this later)

Differentiable Path Tracing with Edge Sampling

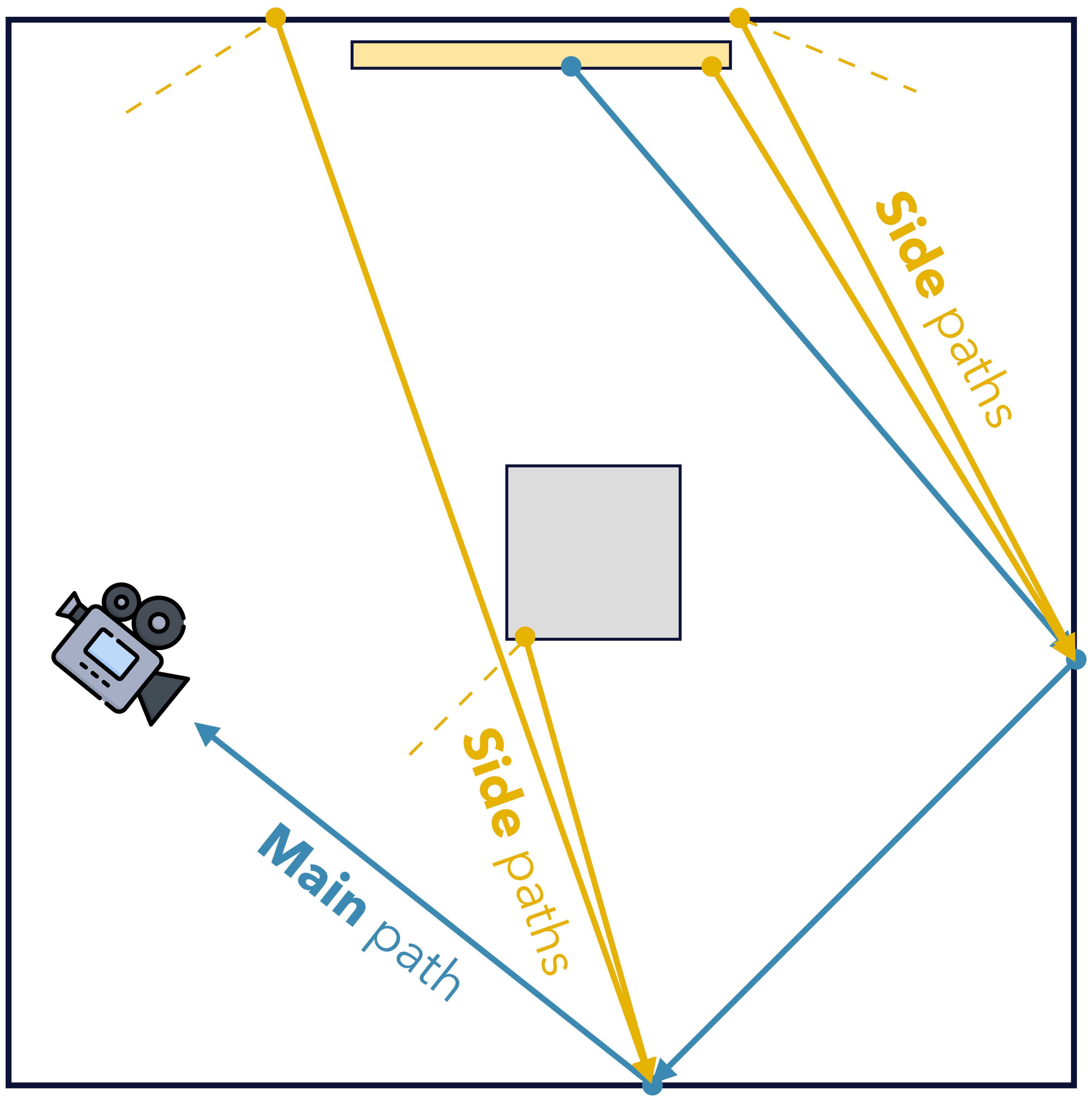
$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_\Omega f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})\right)$$

$$= \int_\Omega \dot{f}(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}}) + \int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})$$

Interior integral **Boundary** integral

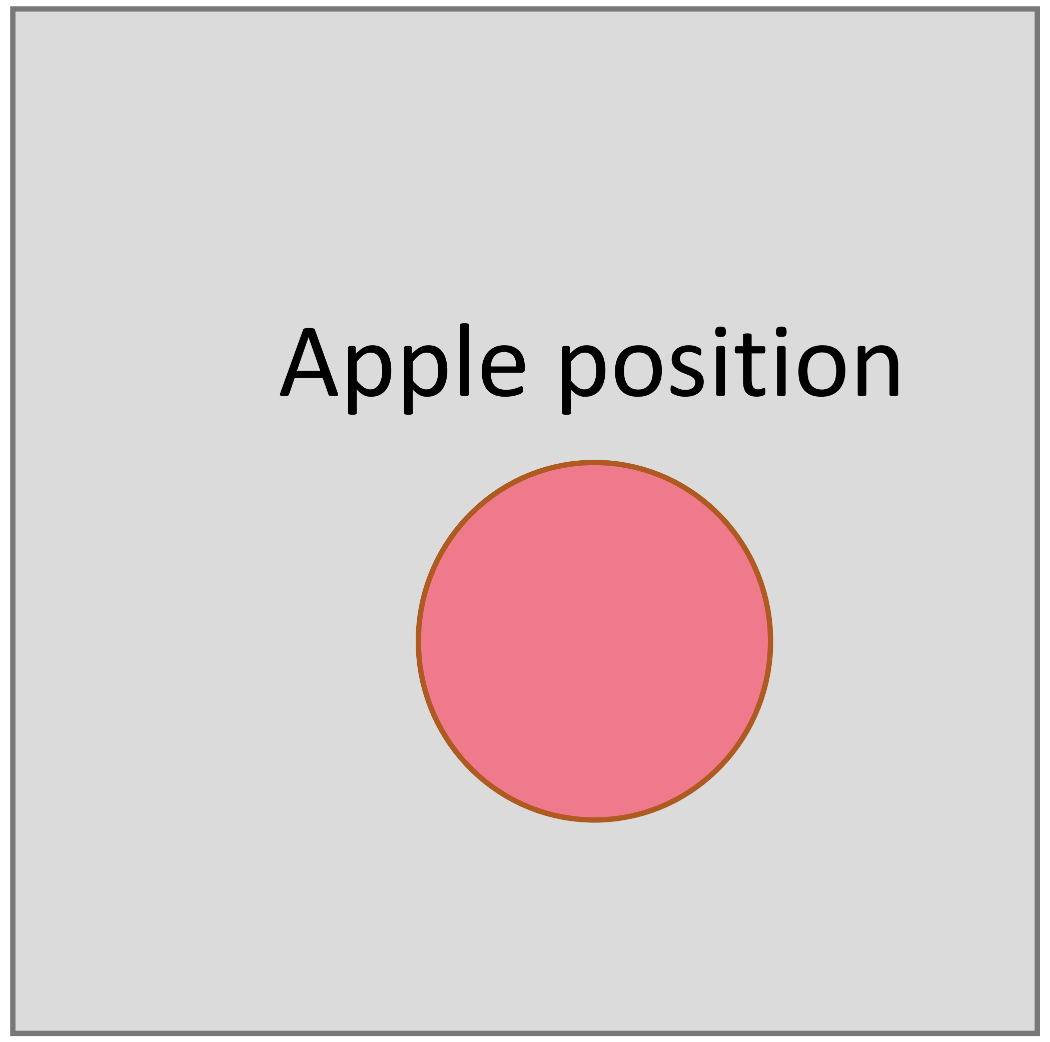
Differentiable path tracing with **edge sampling**

- Trace **main** paths to estimate the **interior** integral
  - Same as ordinary path tracing (for forward rendering)
- Trace additional **side** paths for the **boundary** integral
  - Each **side** path begins with a **boundary** segment (obtained with edge sampling)



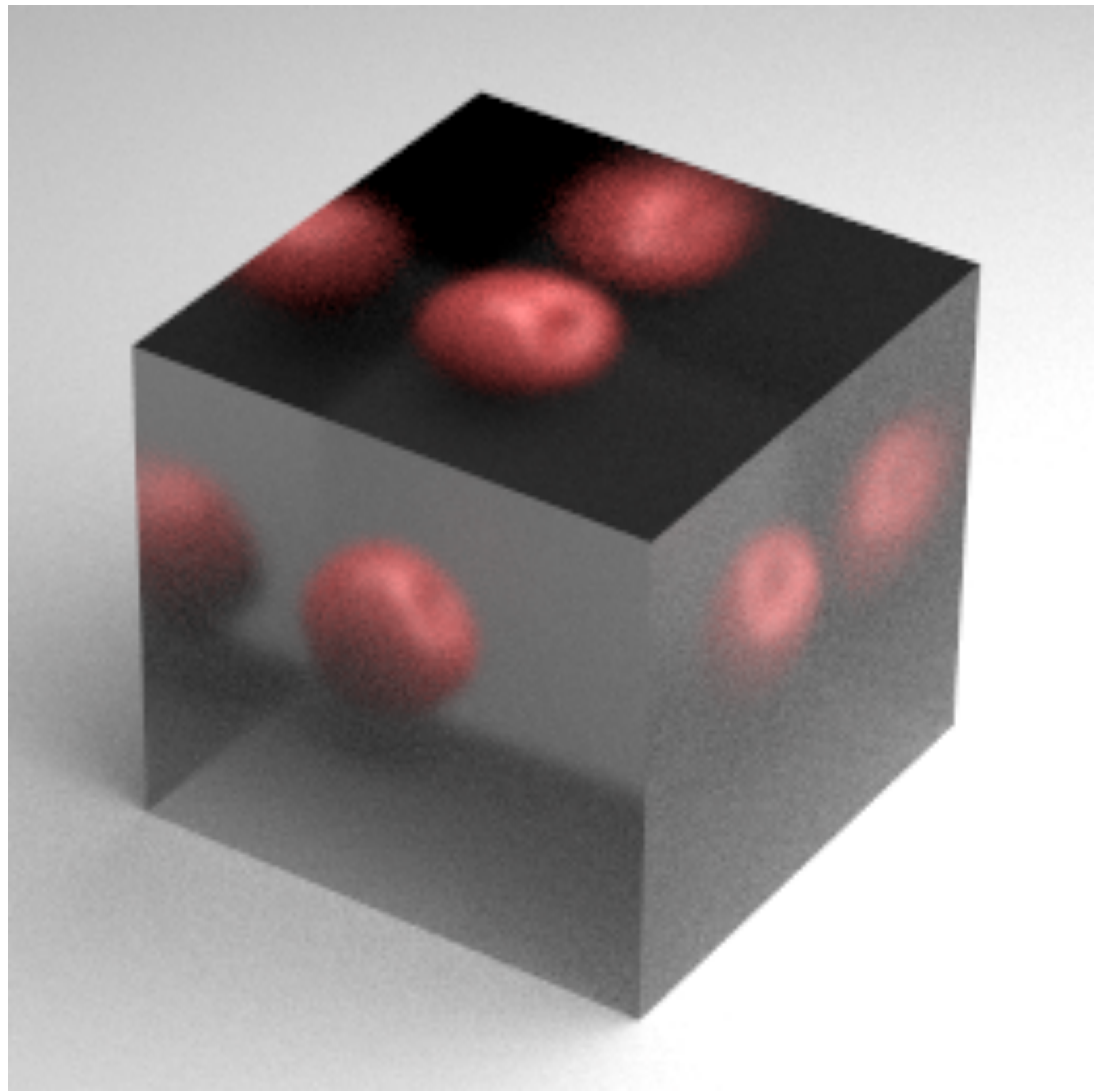
Inverse-Rendering Result [Zhang et al. 2019]

Parameters

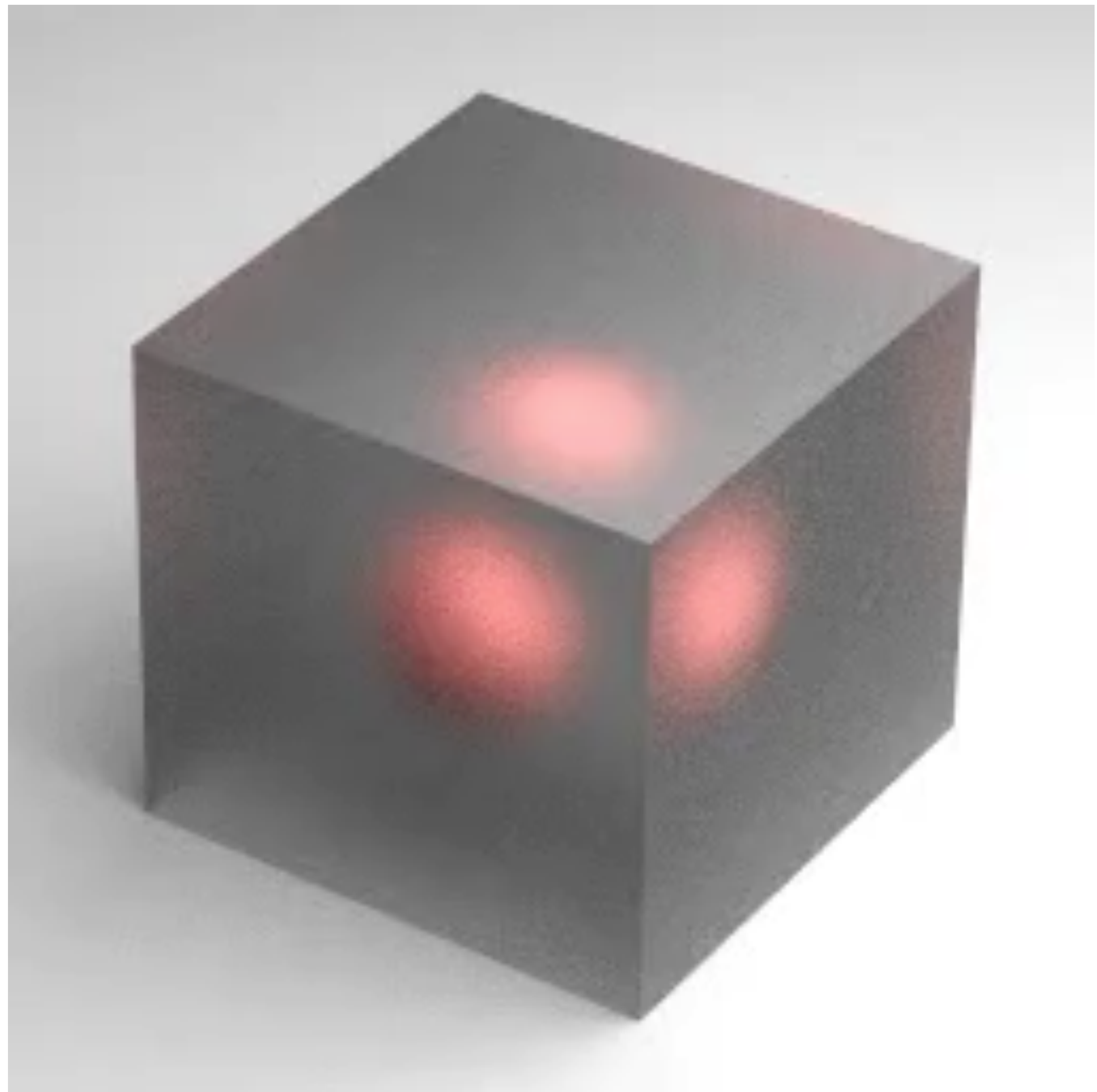


Cube roughness

Target



Optimization process

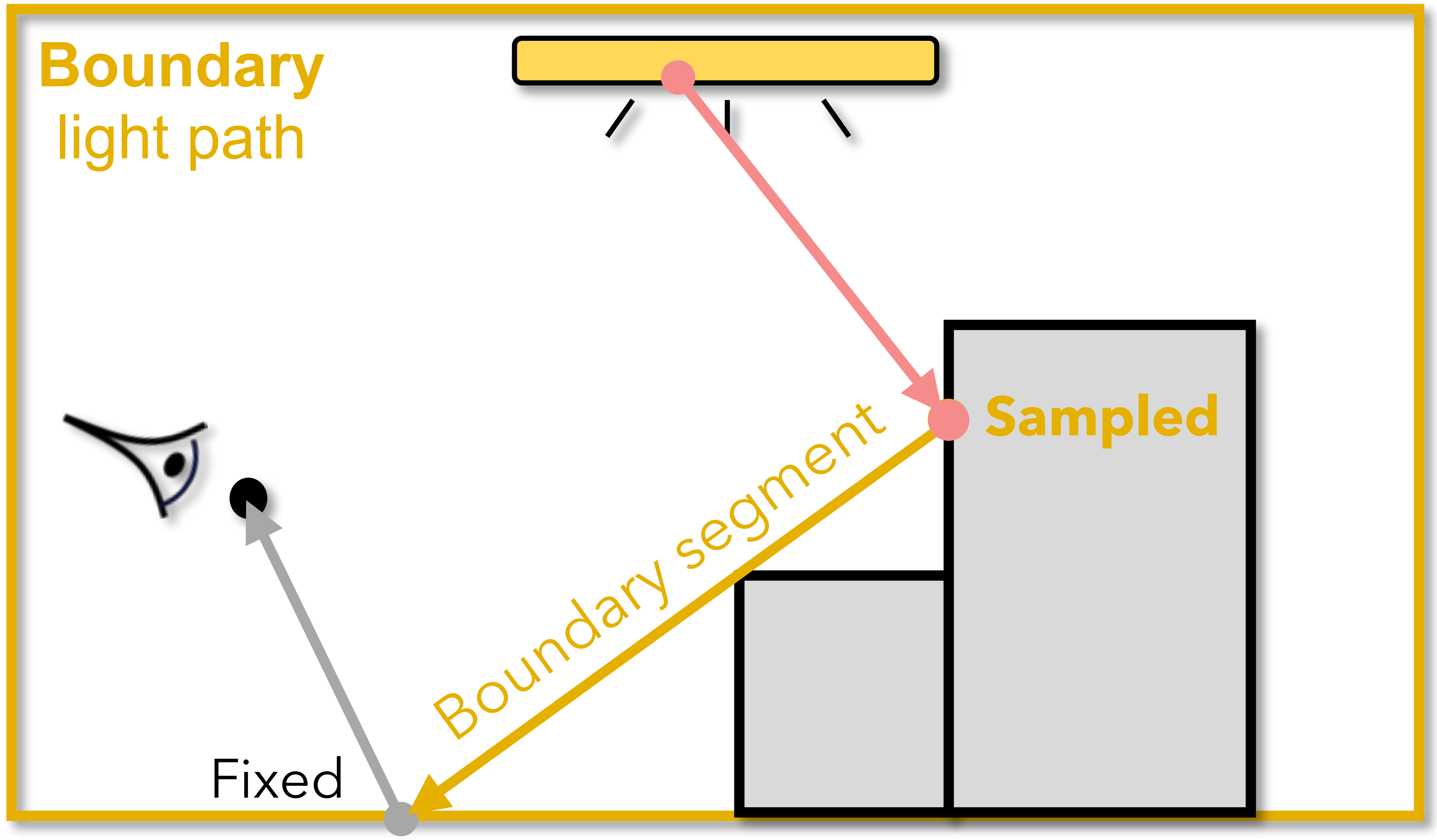


Light-transport phenomena: rough reflection and refraction
subsurface scattering

Differentiable Path Tracing with Edge Sampling

To sample a **boundary** segment:

- Fix one endpoint
- Sample the other from **discontinuity boundaries**



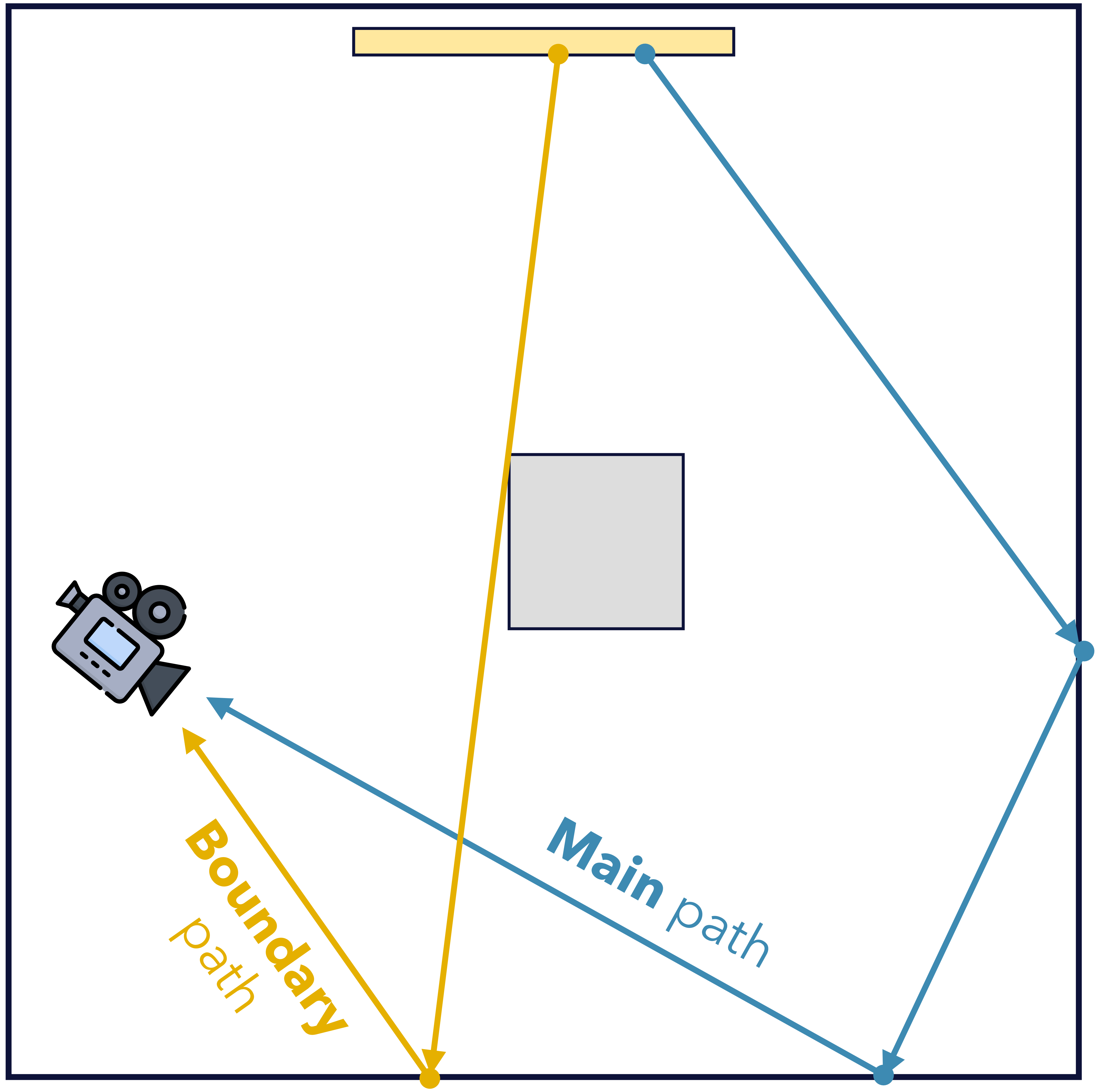
Requires **silhouette detection**, which can be **expensive**!

Path-Space Differentiable Path Tracing

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\left(\int_\Omega f(\bar{x})\,\mathrm{d}\mu(\bar{x})\right)$$

$$= \int_\Omega \dot{f}(\bar{x})\,\mathrm{d}\mu(\bar{x}) + \int_{\partial\Omega} g(\bar{x})\,\mathrm{d}\mu'(\bar{x})$$

Interior integral **Boundary** integral



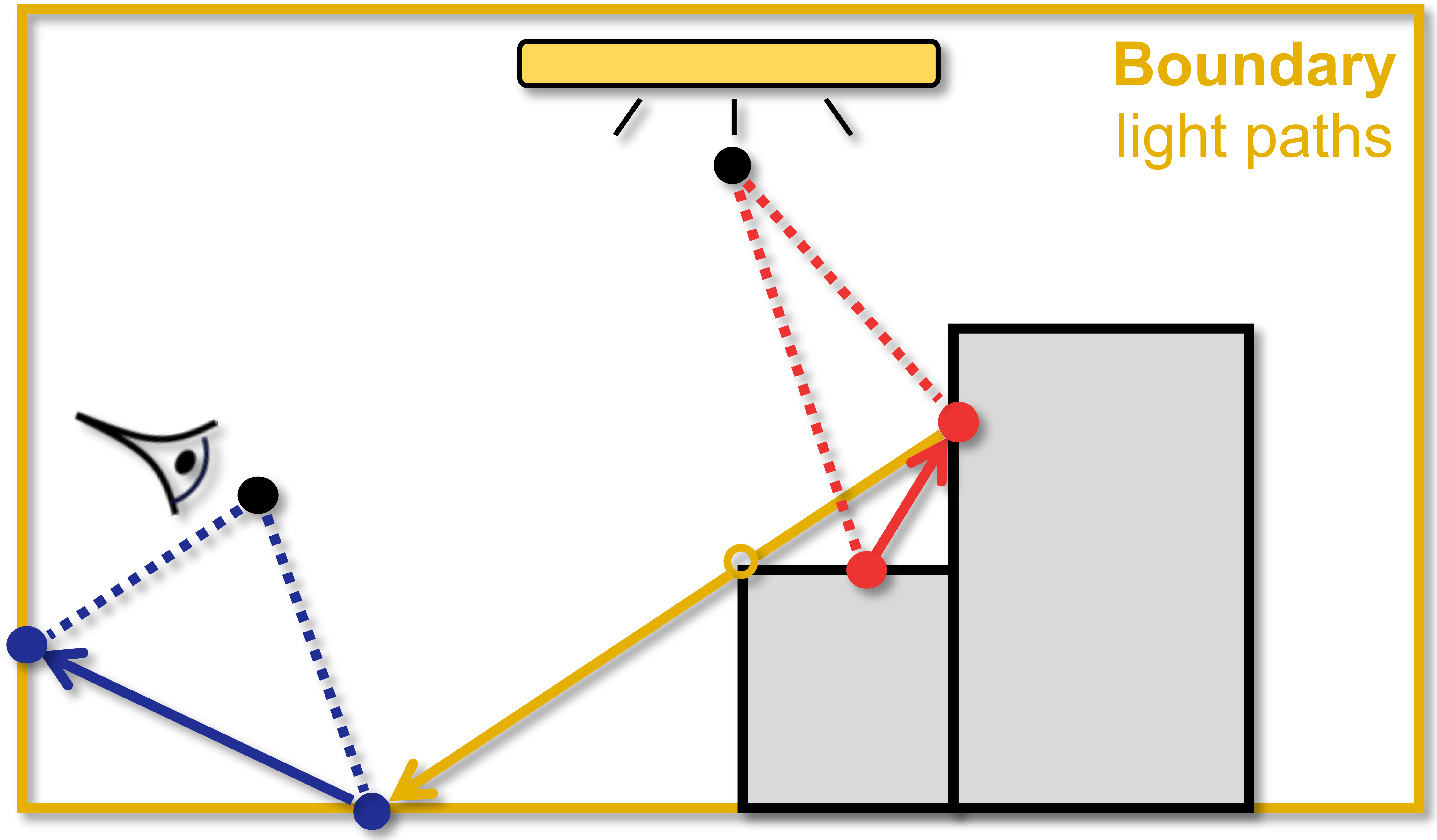
Path-space differentiable path tracing

- Trace **main** paths to estimate the **interior** integral
  - Same as forward rendering
- Trace additional **boundary** paths for the **boundary** integral separately (using multi-directional sampling)

Path-Space Differentiable Path Tracing

Unidirectional estimator

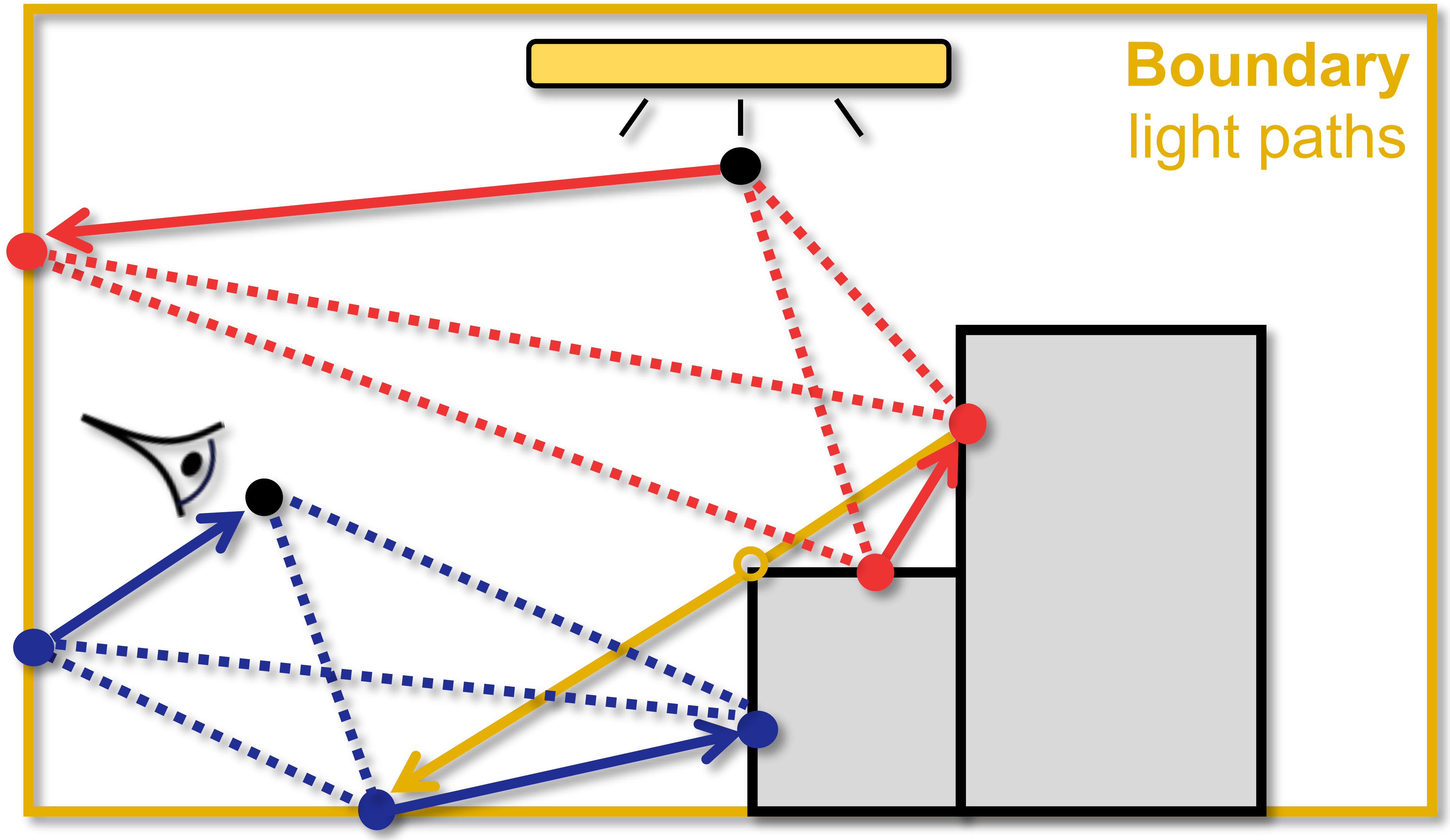
- **Interior**: *unidirectional* path tracing
- **Boundary**: *unidirectional* sampling of subpaths



Unidirectional path tracing + NEE

Bidirectional estimator

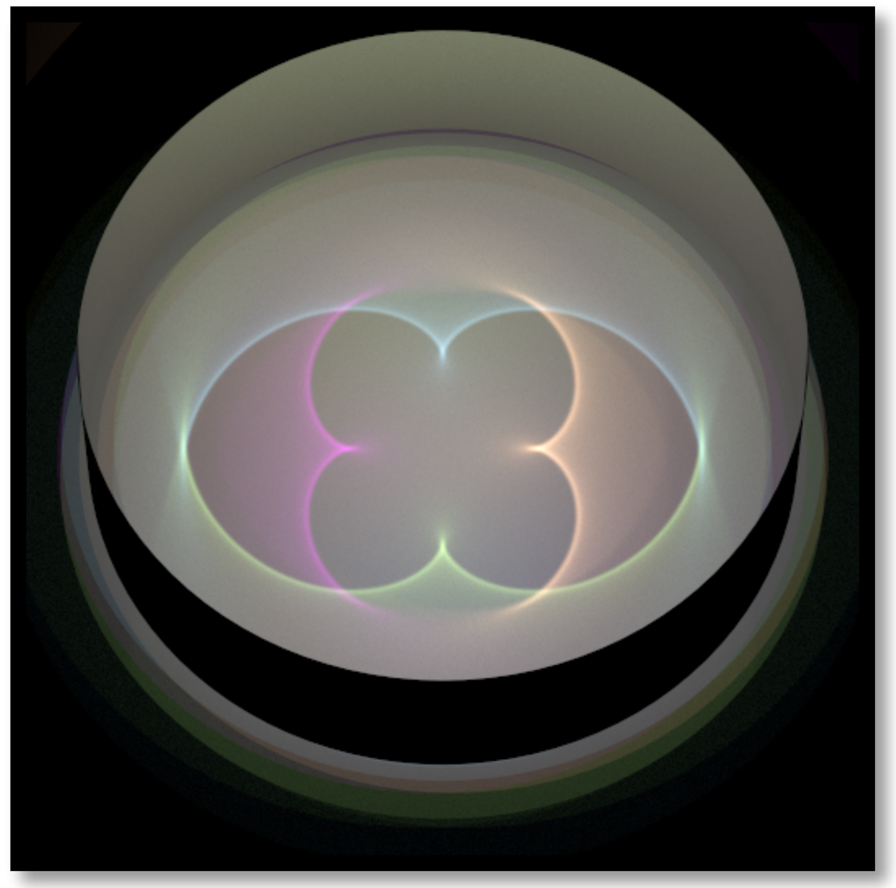
- **Interior**: *bidirectional* path tracing
- **Boundary**: *bidirectional* sampling of subpaths



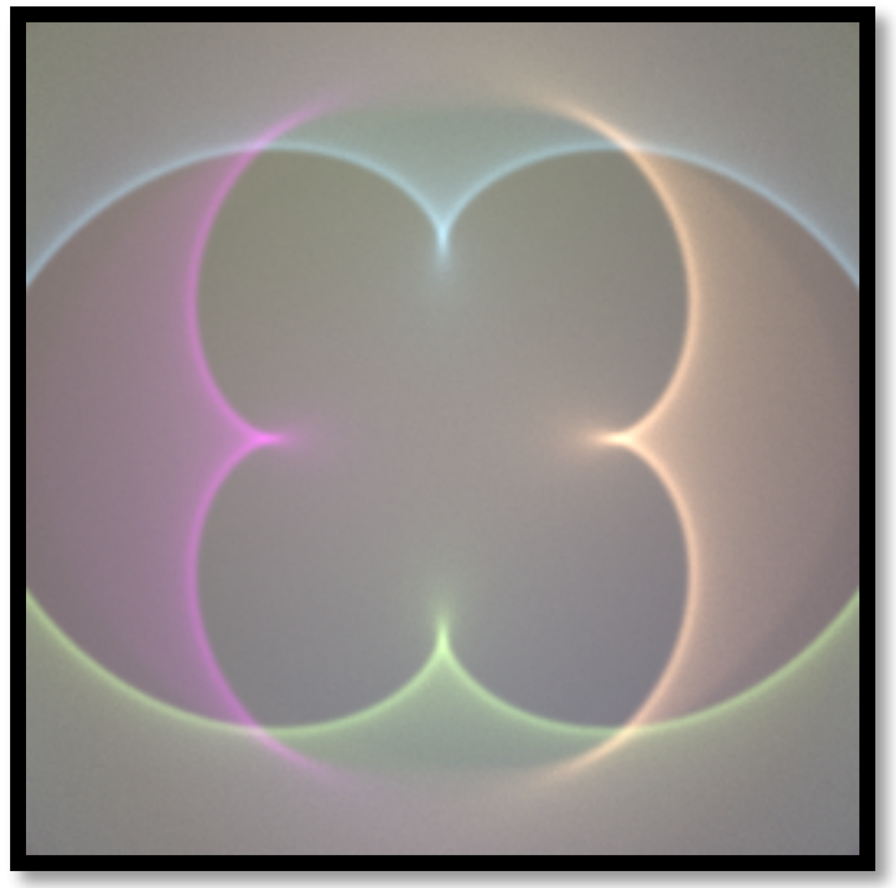
Bidirectional path tracing

Inverse-Rendering Result [Zhang et al. 2020]

Config.



Initial



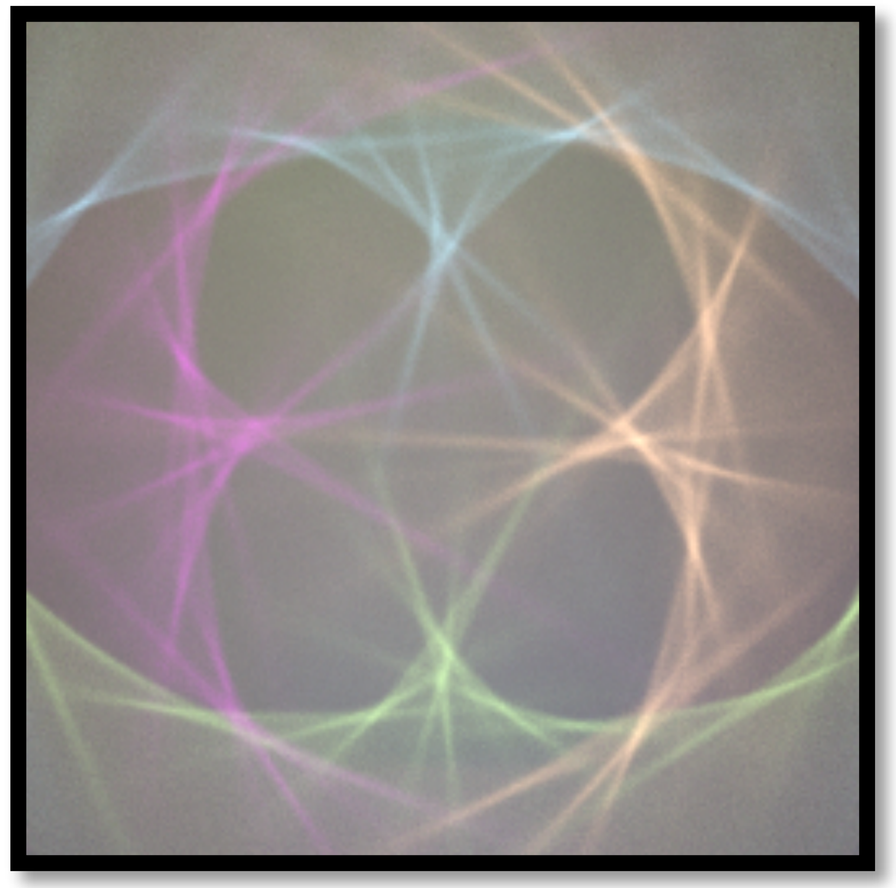
Scene configuration:

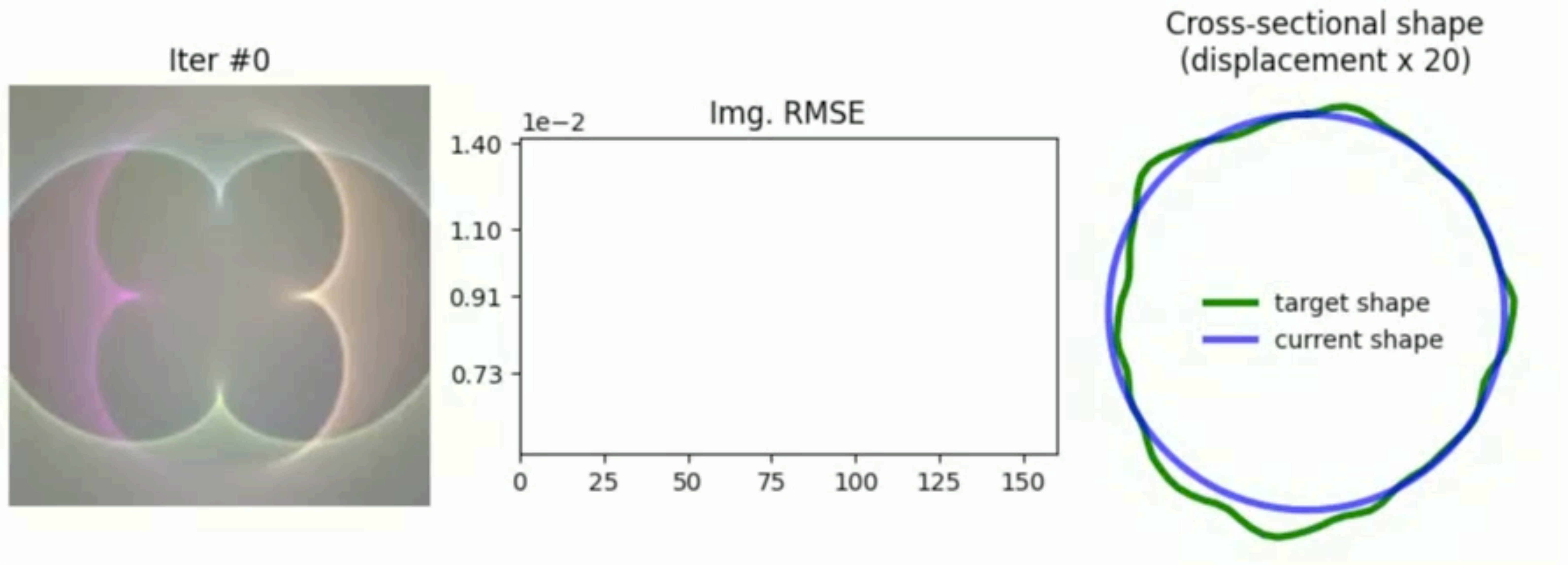
- A glossy ring lit by four colored light sources
- Optimize **cross-sectional shape** of the ring

Light-transport phenomenon:

- Caustics

Target

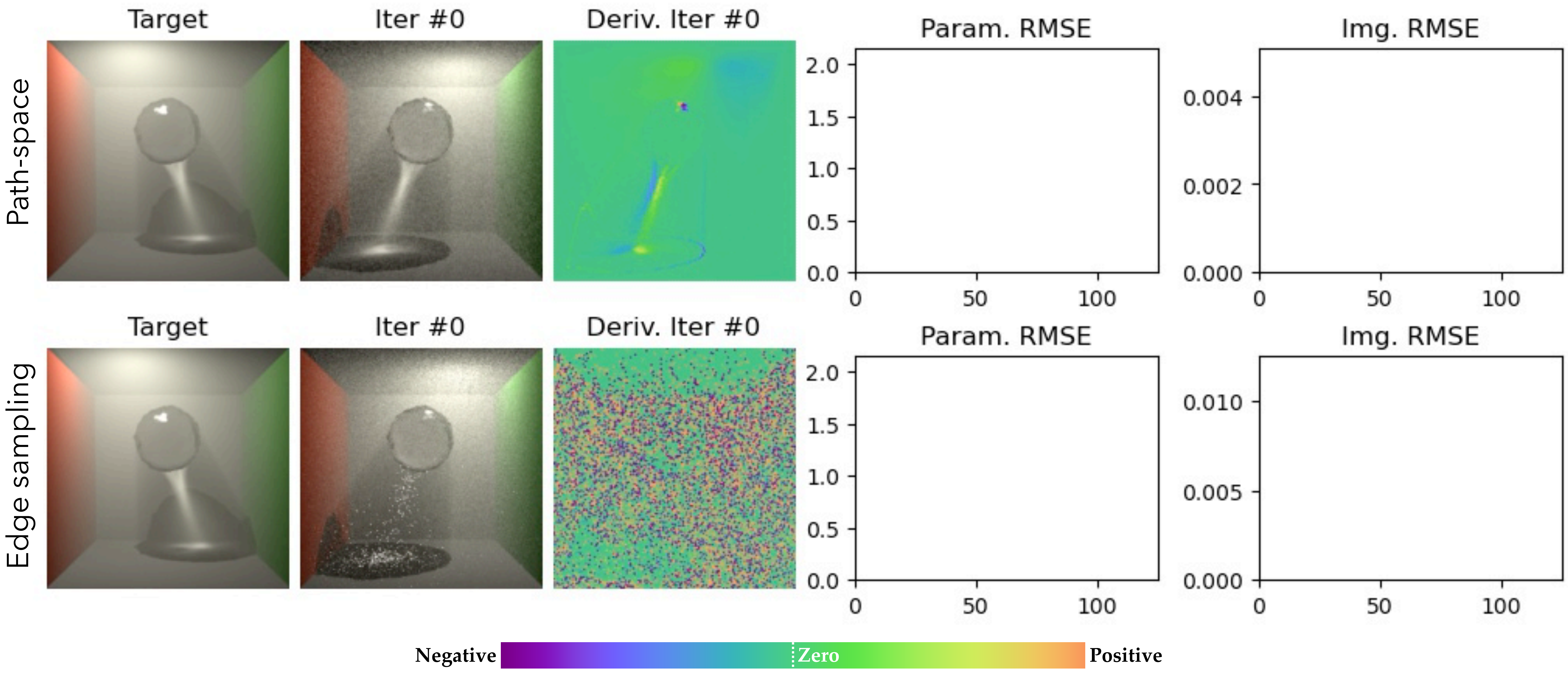




Inverse-Rendering Comparison [Zhang et al. 2021]

Optimizing the position of a small area light

(identical inverse-rendering configurations, equal-time per iteration)



Inverse-Rendering Result [Zhang et al. 2021]

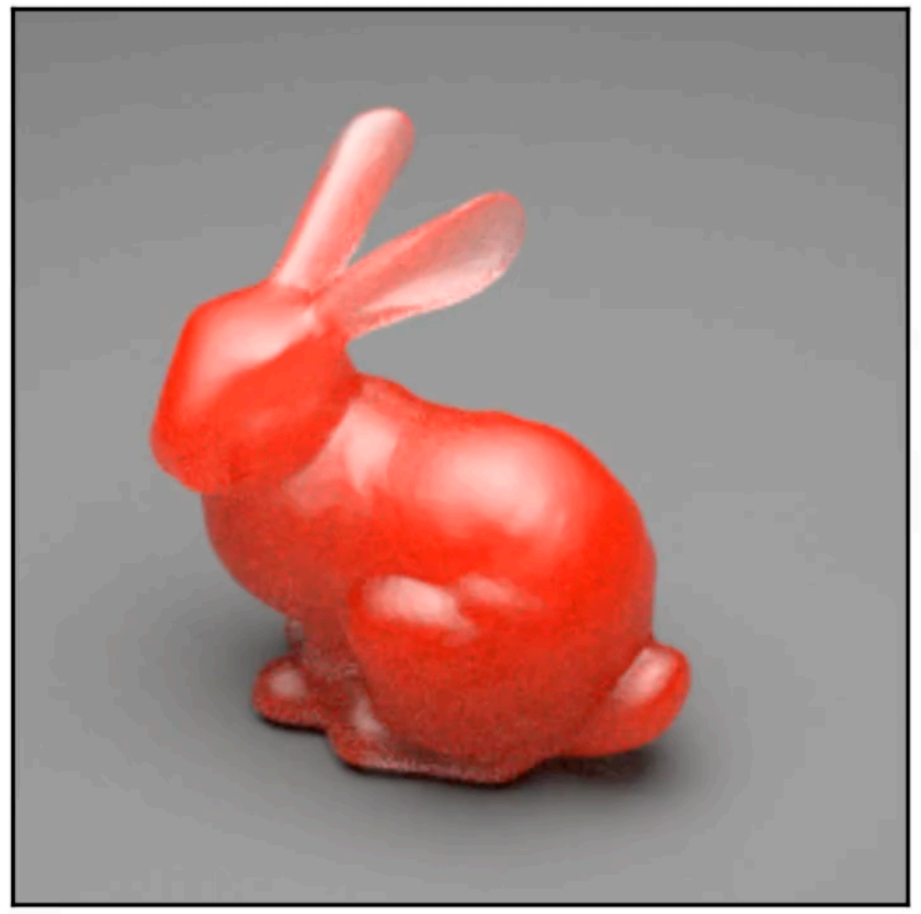
Initial

Target

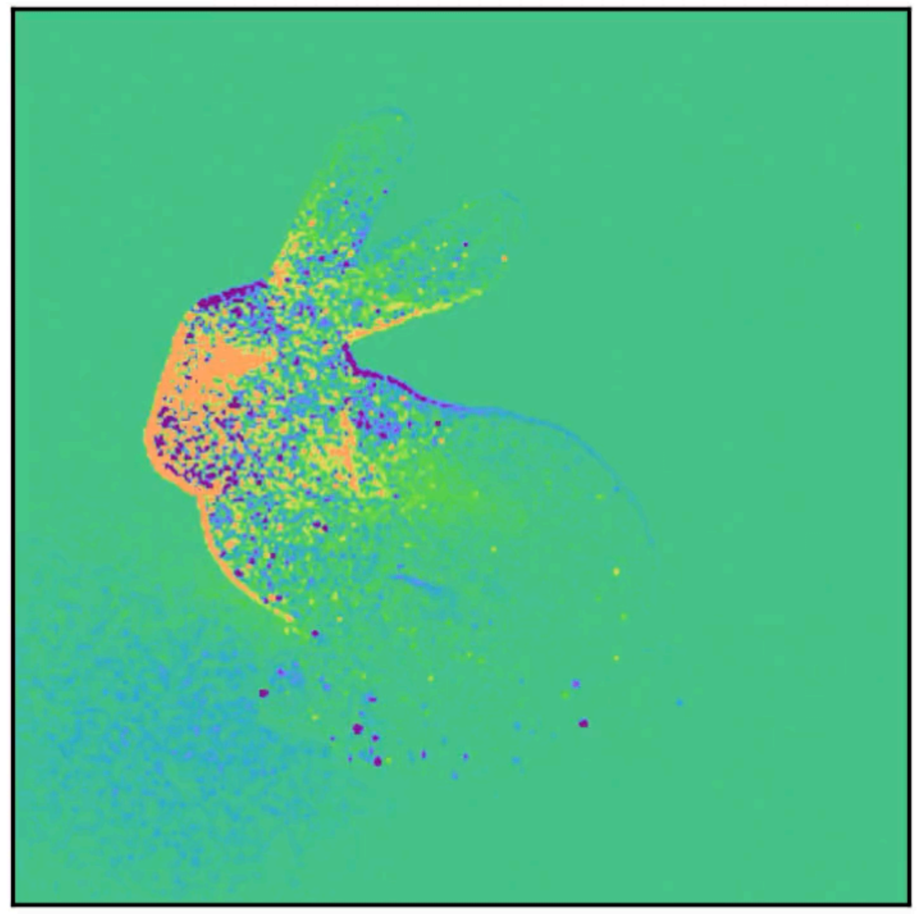
Jointly optimizing of the bunny's:

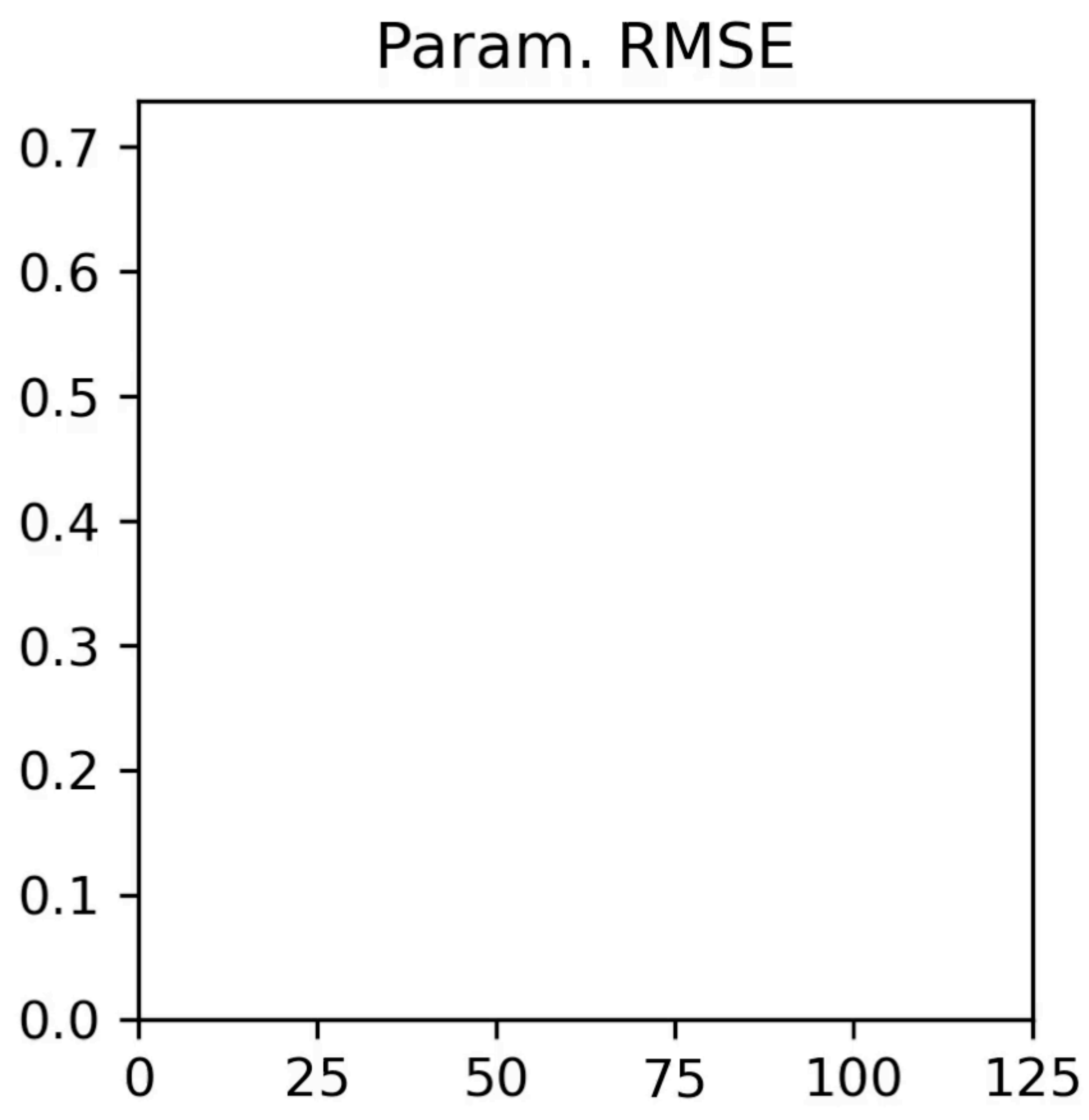
- Shape
- Surface roughness
- Optical density

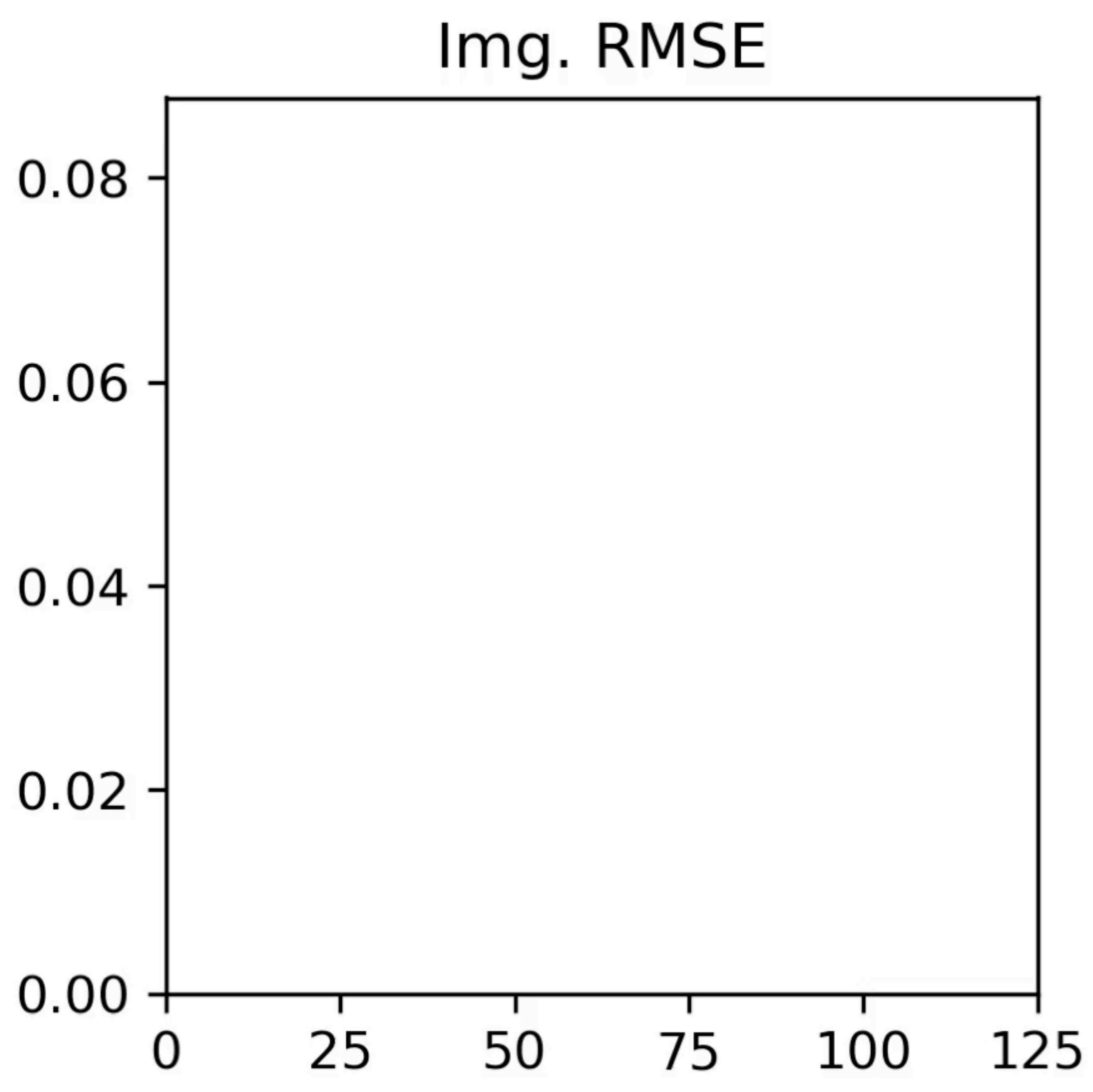
Iter #0



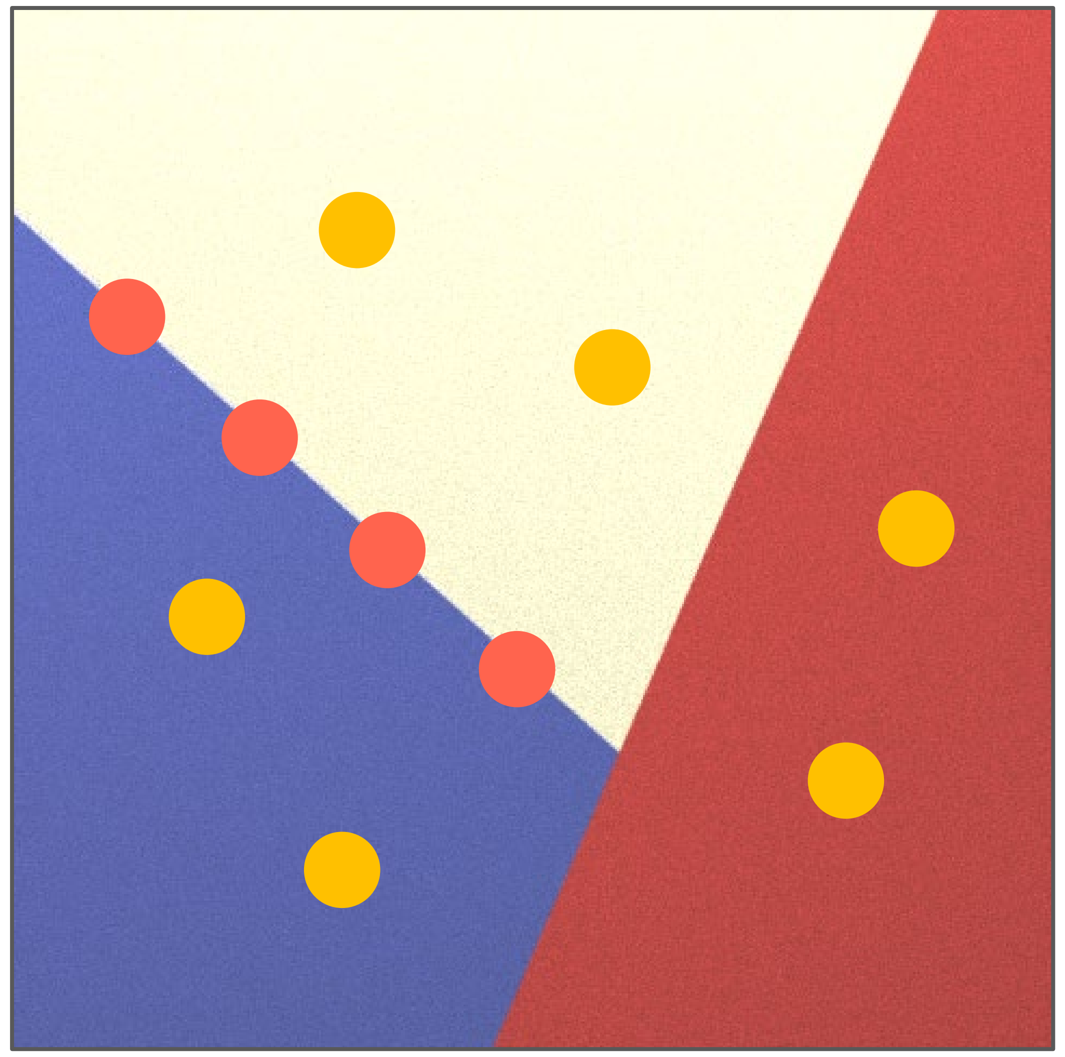
Deriv. Iter #0

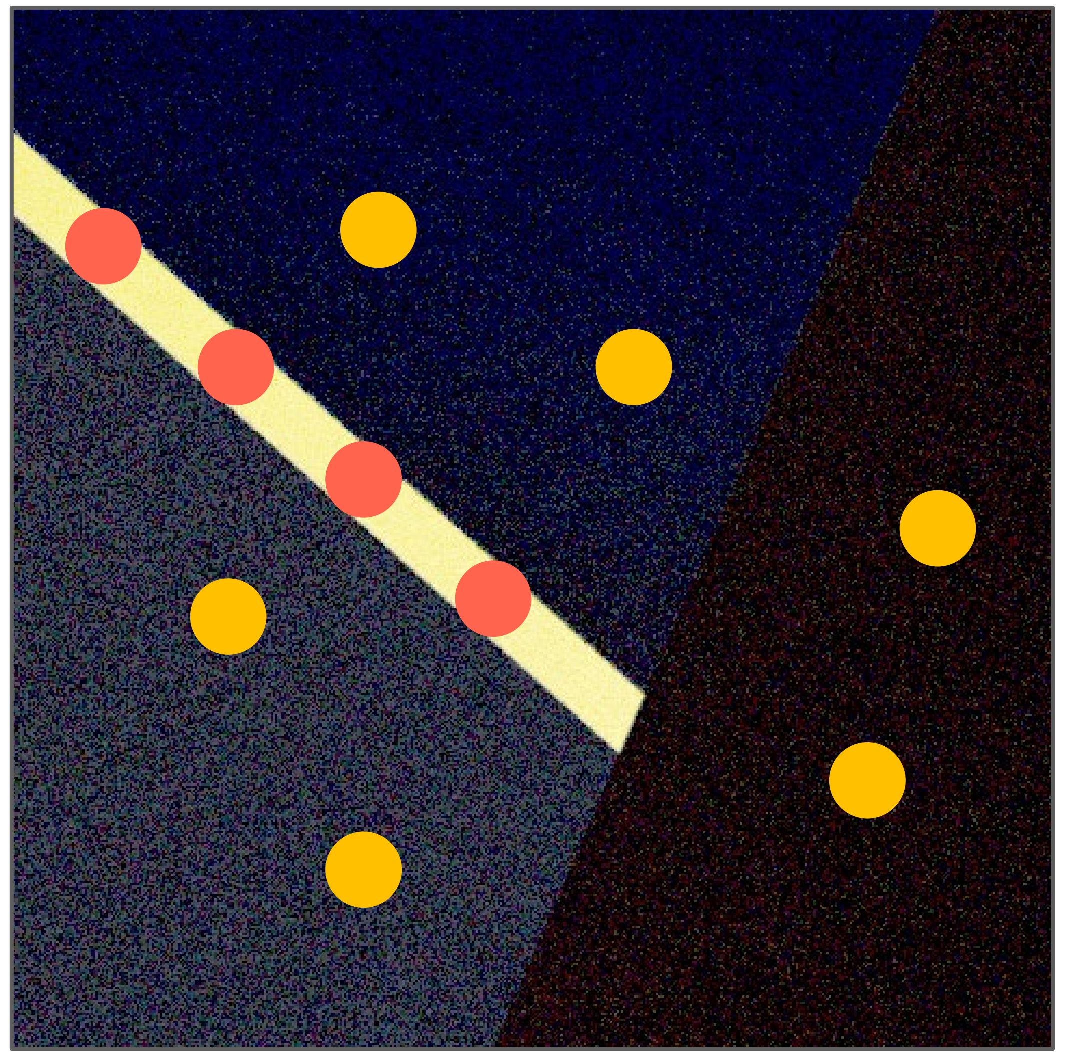






Boundary methods can run into some problems



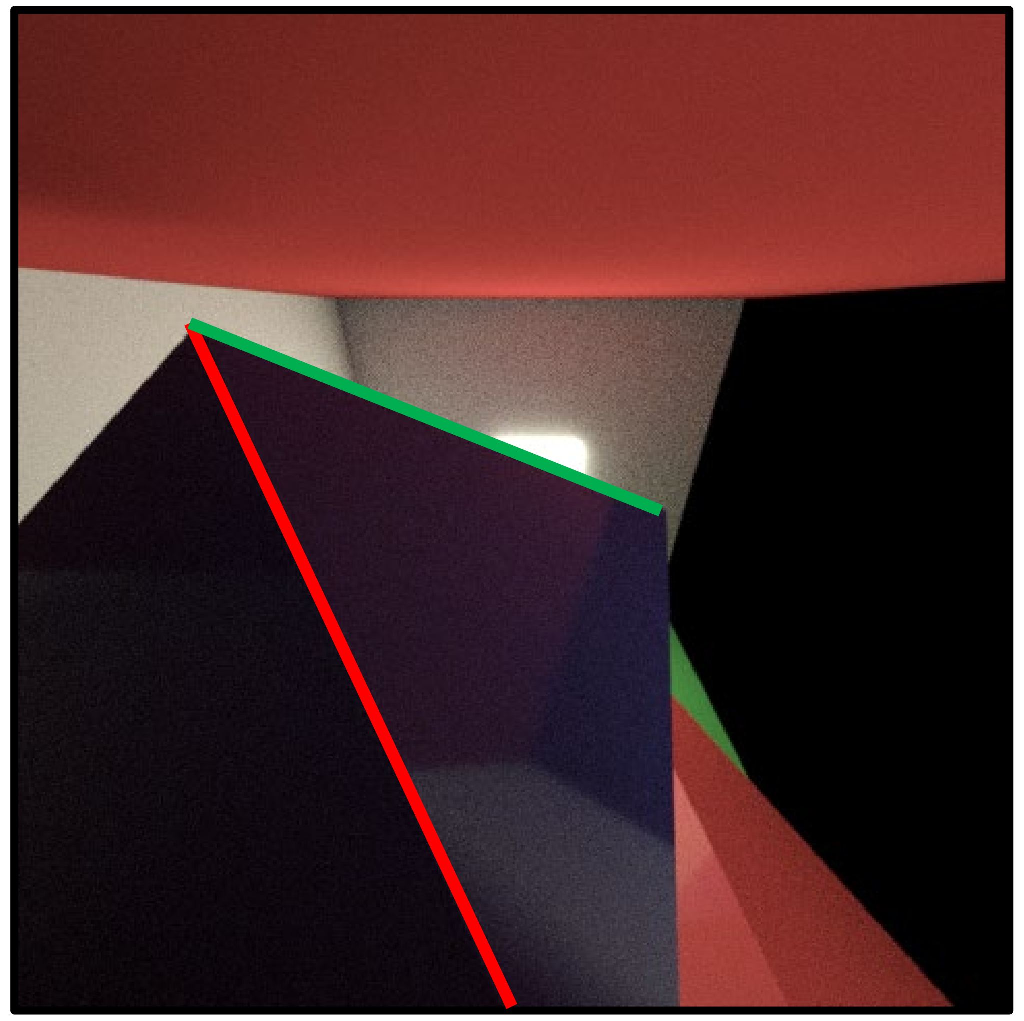


: Boundary samples

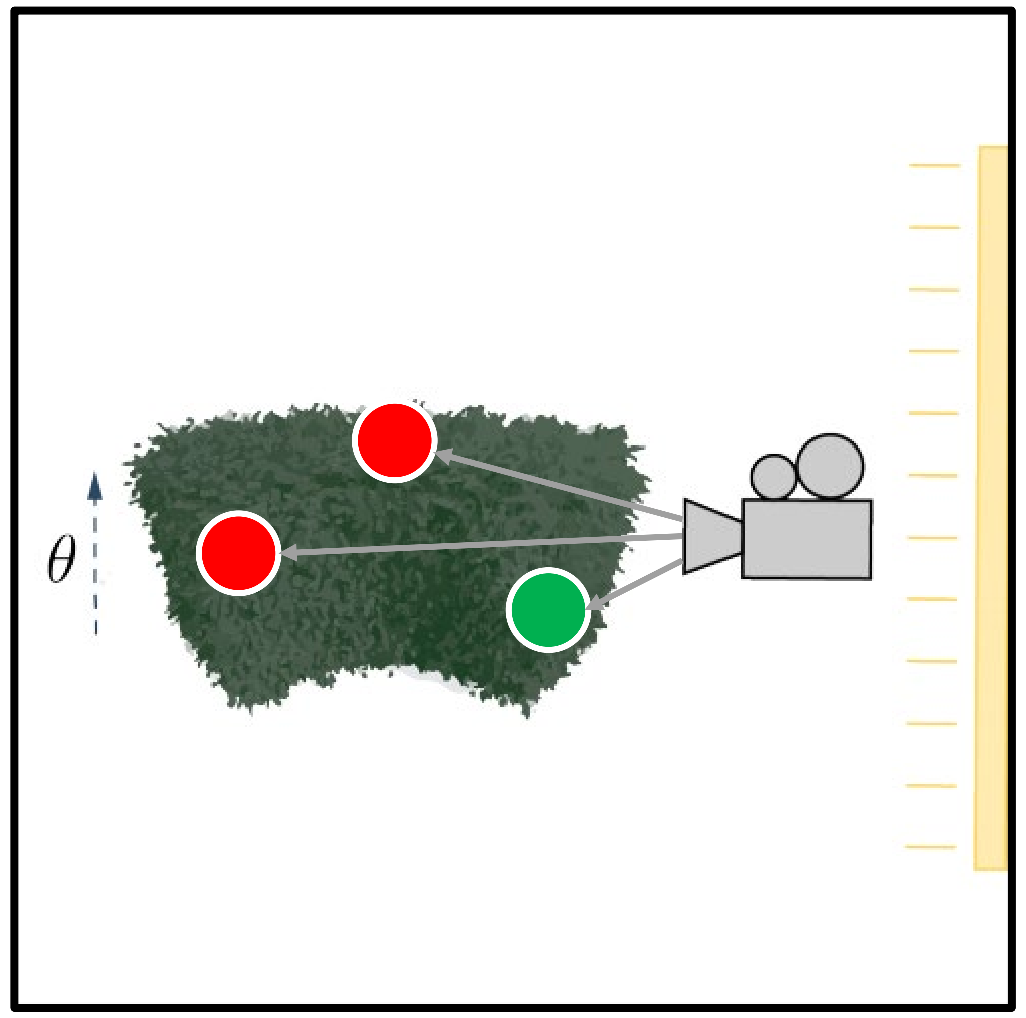
: Area samples

Boundary methods can run into some problems

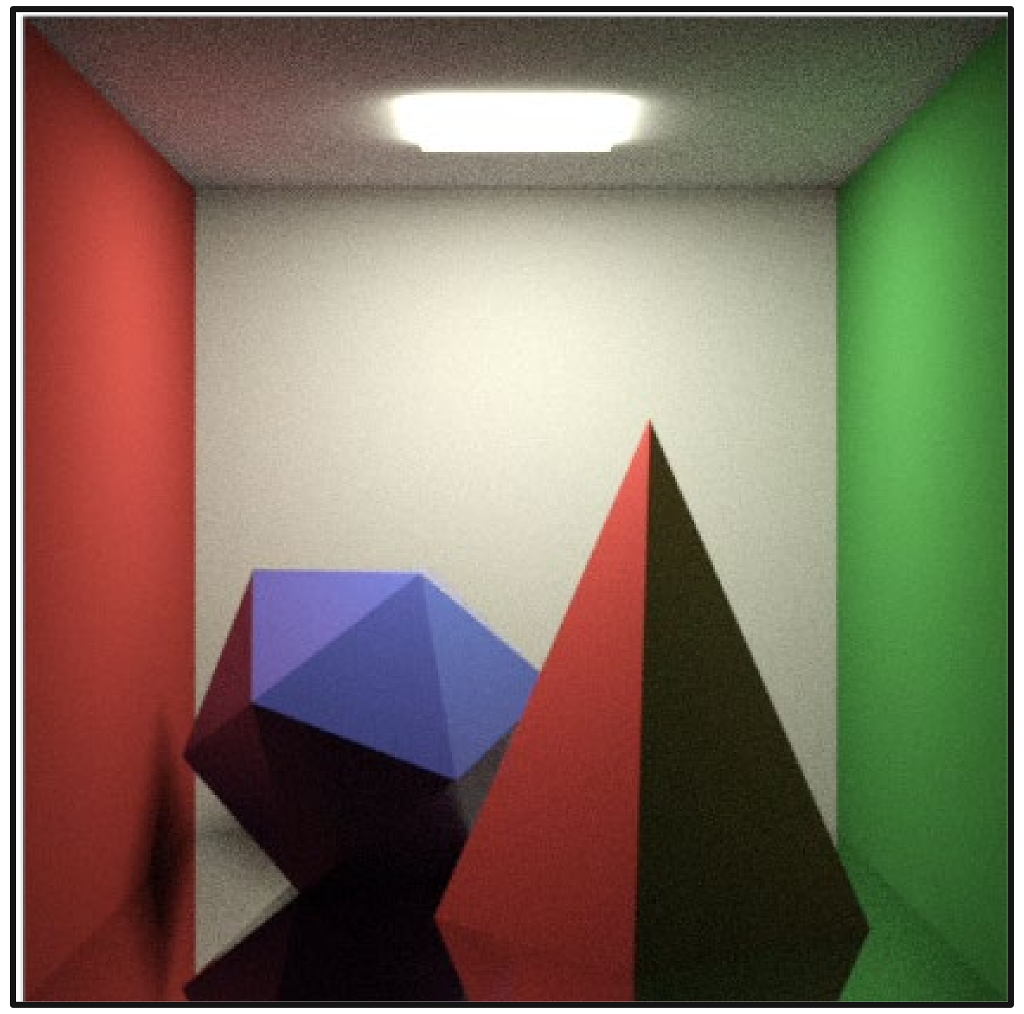
Some challenges faced



Silhouette sampling



Depth complexity



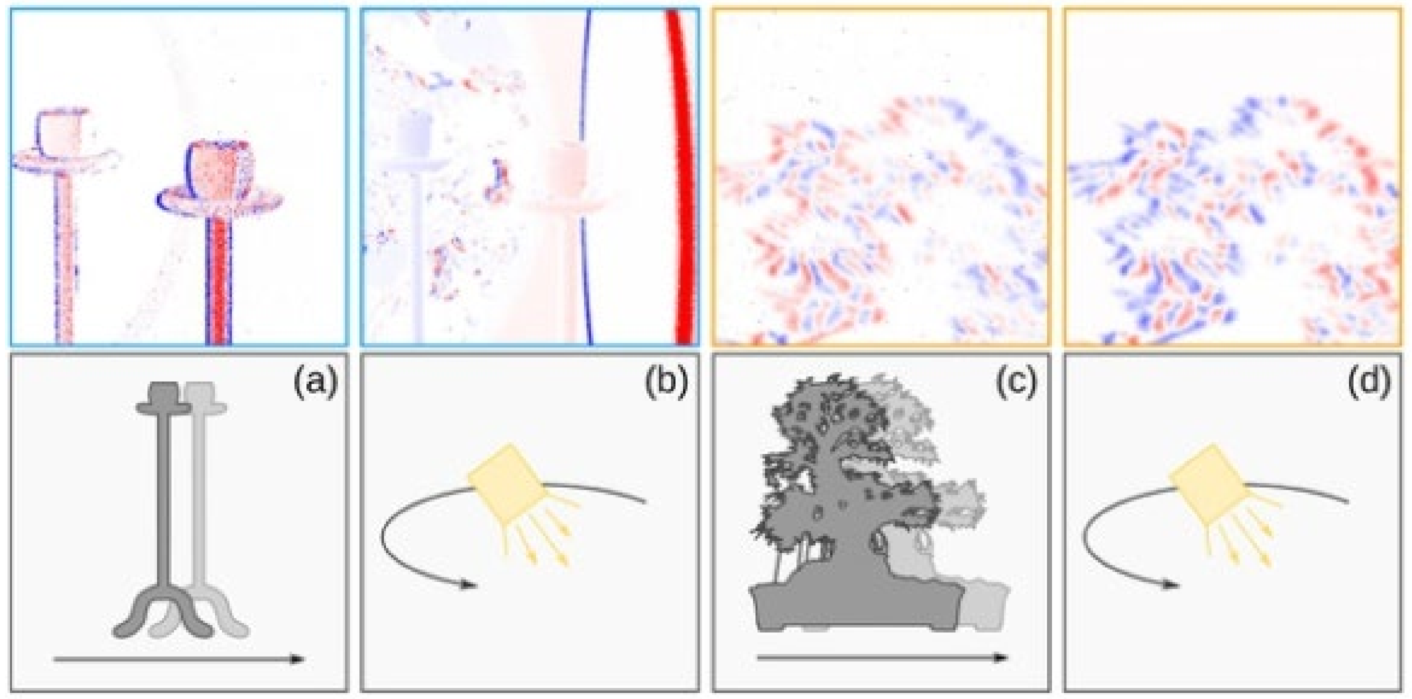
Perfect specularities

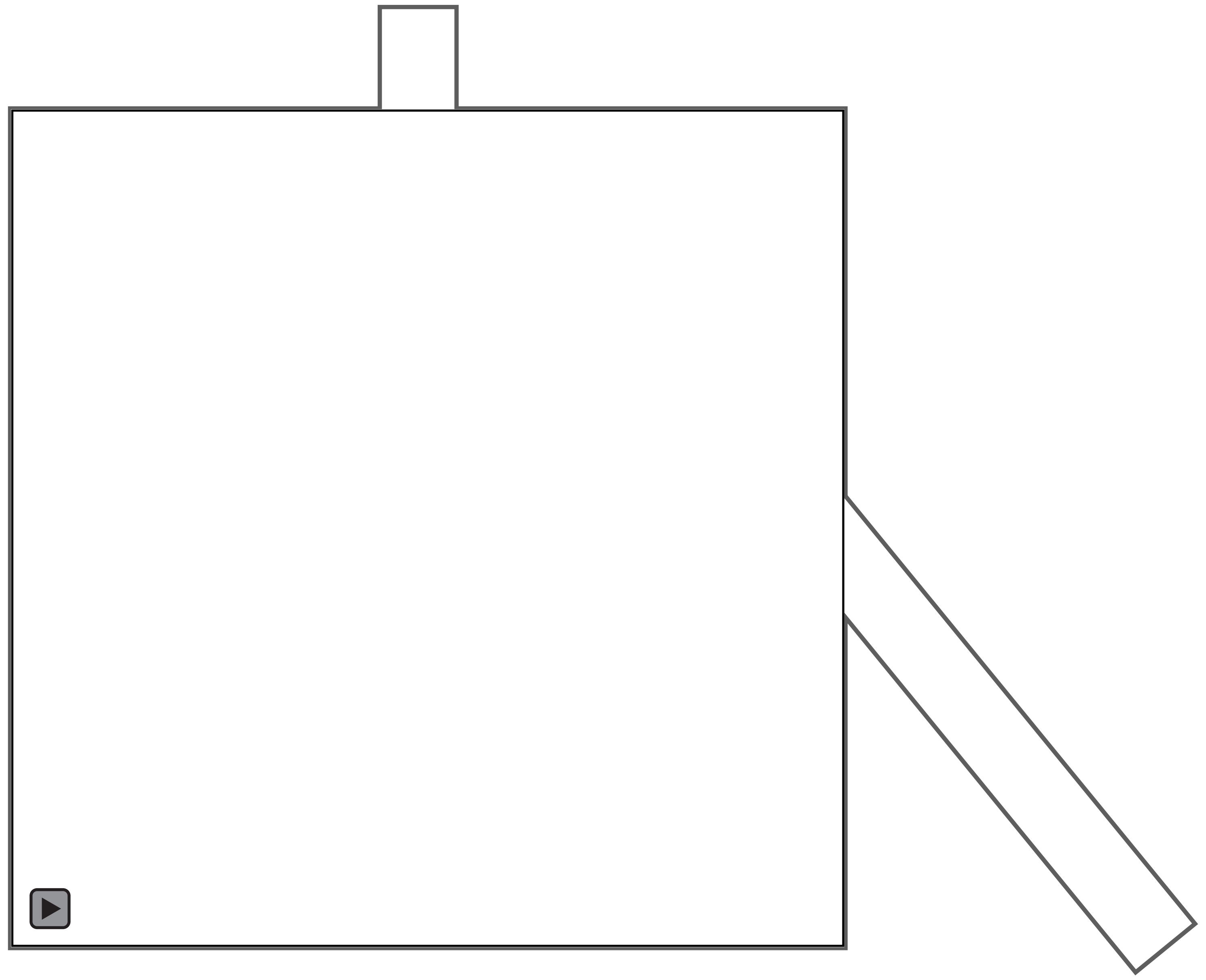
Integrating over arbitrary discontinuities can be tricky

Avoid discontinuities through reparameterization

Transform the space with **θ**.
“Cancels” discontinuities.

Reparameterizing Discontinuous Integrands for Differentiable Rendering
[Loubet 2019]





Heuristic Approximation!
May not work for all samples.

Unbiased Warped-Area Sampling for Differentiable Rendering

SAI PRAVEEN BANGARU, Massachusetts Institute of Technology
TZU-MAO LI, Massachusetts Institute of Technology
FRÉDO DURAND, Massachusetts Institute of Technology

Fig. 1. Differentiable rendering computes derivatives of the light transport equation. To differentiate with the existence of visibility, recent physically-based differentiable renderers require either explicitly finding boundary points [Li et al. 2018; Zhang et al. 2020], or approximating the boundary contribution through heuristics [Loubet et al. 2019]. We develop from first principles an unbiased estimator that computes the boundary contribution from interior (area) samples. Our approach can be easily integrated with existing importance sampling methods and computes accurate and low variance gradients. For instance, the *edge sampling method* [Li et al. 2018] finds it difficult to consistently sample boundary points that contribute to the derivative in the soft reflection, especially because of the high complexity of the scene. Our method, on the other hand, uses samples from a standard path tracer and takes advantage of BSDF and light source importance sampling to compute a robust estimate for the derivative. We validate our derivatives against the finite difference image computed w.r.t the hedge's translation in the upward direction. Both our method and edge sampling used an equal number of samples.

Differentiable rendering computes derivatives of the light transport equation with respect to arbitrary 3D scene parameters, and enables various applications in inverse rendering and machine learning. We present an unbiased and efficient differentiable rendering algorithm that does not require explicit boundary sampling. We apply the divergence theorem to the derivative of the rendering integral to convert the boundary integral into an area integral. We rewrite the converted area integral to a form that is suitable for Monte Carlo rendering. We then develop an efficient Monte Carlo sampling algorithm for solving the area integral. Our method can be easily plugged into a traditional path tracer and does not require dedicated data structures for sampling boundaries.

We analyze the convergence properties through bias-variance metrics, and demonstrate our estimator's advantages over existing methods for some synthetic inverse rendering examples.

CCS Concepts: • **Computing methodologies** → *Computer vision*; **Rendering**; **Visibility**.

Additional Key Words and Phrases: inverse graphics, differentiable rendering, light transport, differentiating visibility

ACM Reference Format:
Sai Praveen Bangaru, Tzu-Mao Li, and Frédo Durand. 2020. Unbiased Warped-Area Sampling for Differentiable Rendering. *ACM Trans. Graph.* 39, 6, Article 245 (December 2020), 18 pages. https://doi.org/10.1145/3414685.3417833

Authors' addresses: Sai Praveen Bangaru, Massachusetts Institute of Technology, Cambridge, MA, sbangaru@mit.edu; Tzu-Mao Li, Massachusetts Institute of Technology, Cambridge, MA, tzumao@mit.edu; Frédo Durand, Massachusetts Institute of Technology, Cambridge, MA, fredo@mit.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2020 Copyright held by the owner/author(s).
0730-0301/2020/12-ART245
https://doi.org/10.1145/3414685.3417833

1 INTRODUCTION

Differentiable rendering – the task of computing derivatives of the light transport equation [Kajiya 1986] with respect to scene parameters such as camera position, triangle mesh positions, and texture parameters, has become increasingly important for solving inverse rendering problems and training 3D deep learning models. The discontinuities introduced by visibility pose a central challenge

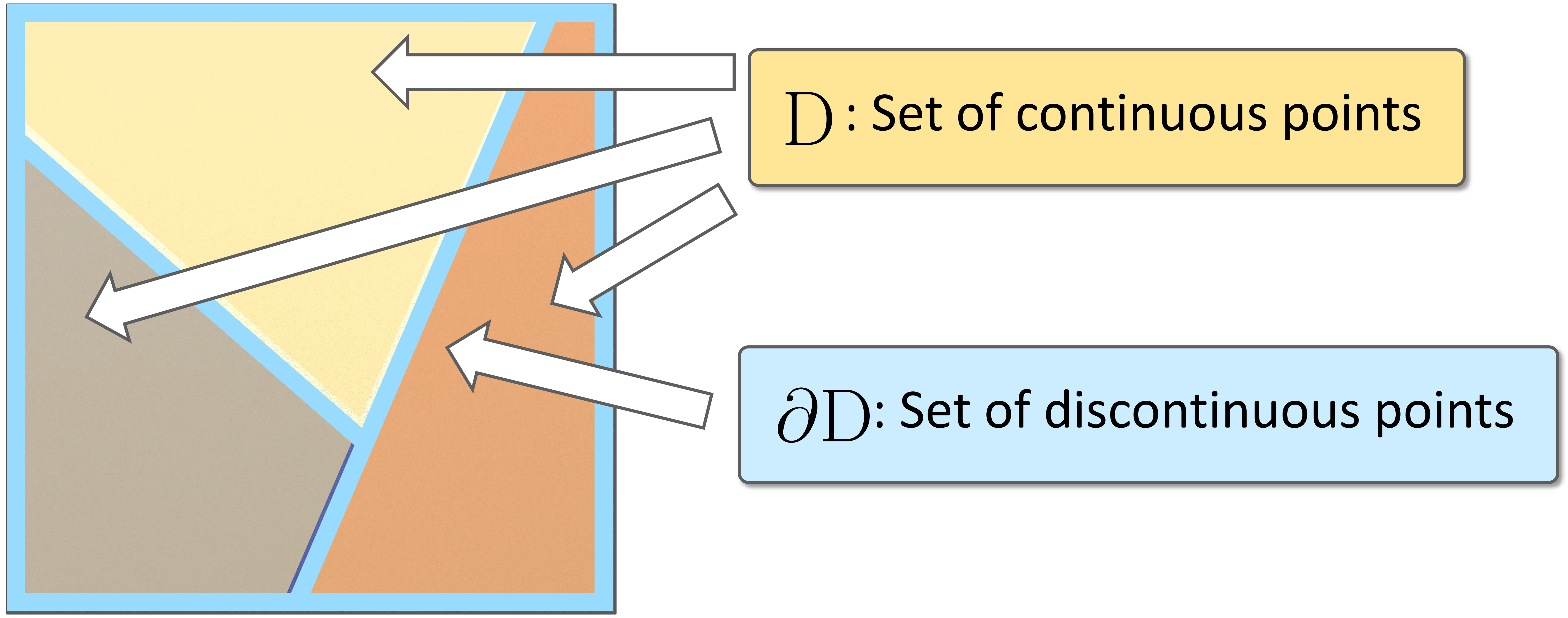
ACM Trans. Graph., Vol. 39, No. 6, Article 245. Publication date: December 2020.

Unbiased Warped-Area Sampling for Differentiable Rendering

Sai **Bangaru**, Tzu-Mao **Li**, Fredo **Durand**
(MIT CSAIL)

SIGGRAPH Asia 2020

The Reynolds Transport Theorem



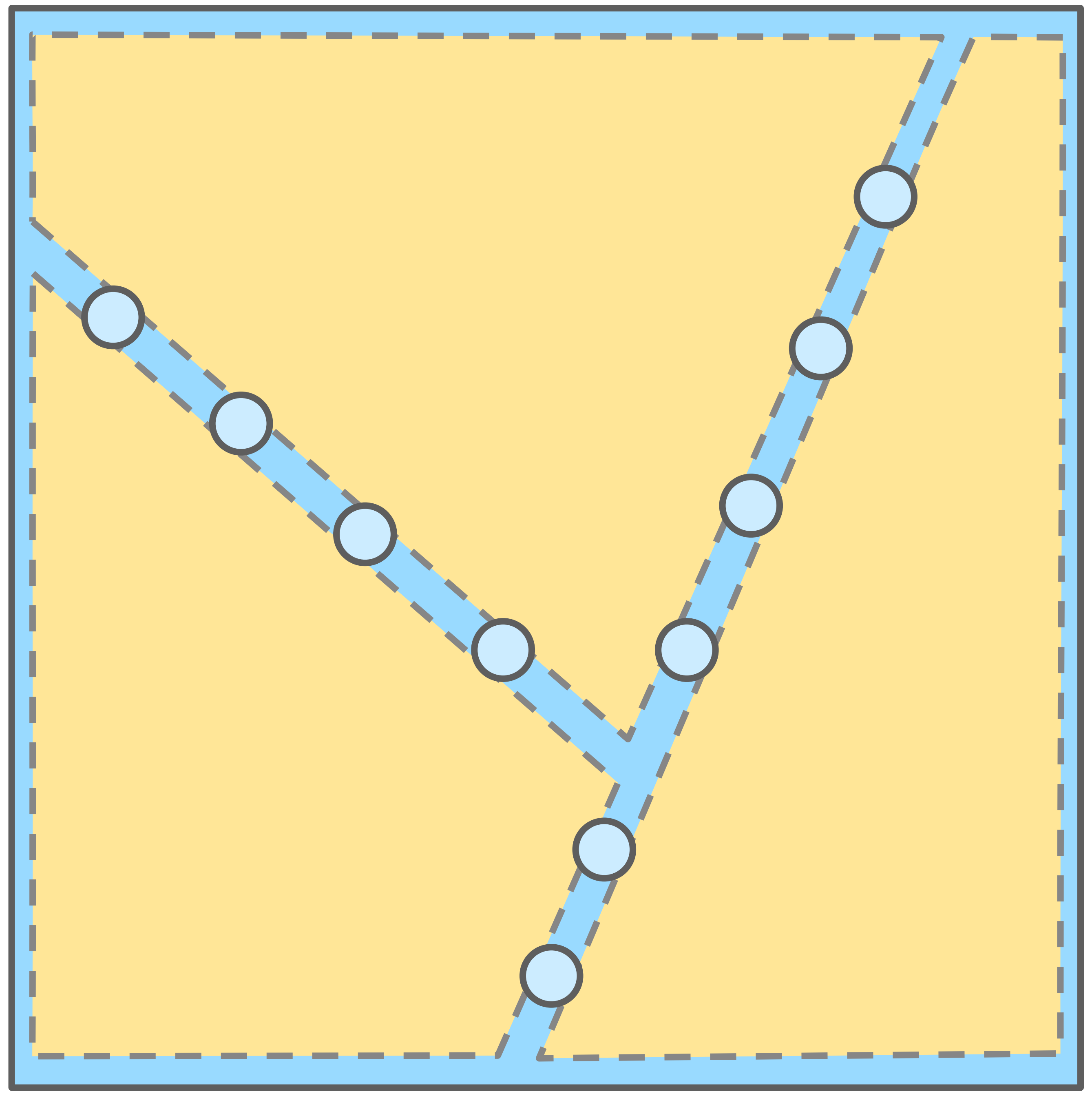
$$\partial_\theta \int_{\mathrm{D}} f = \int_{\mathrm{D}} \partial_\theta f + \int_{\partial\mathrm{D}} f \vec{\mathbf{v}} \cdot \vec{\mathbf{n}}$$

Interior term — Edge term

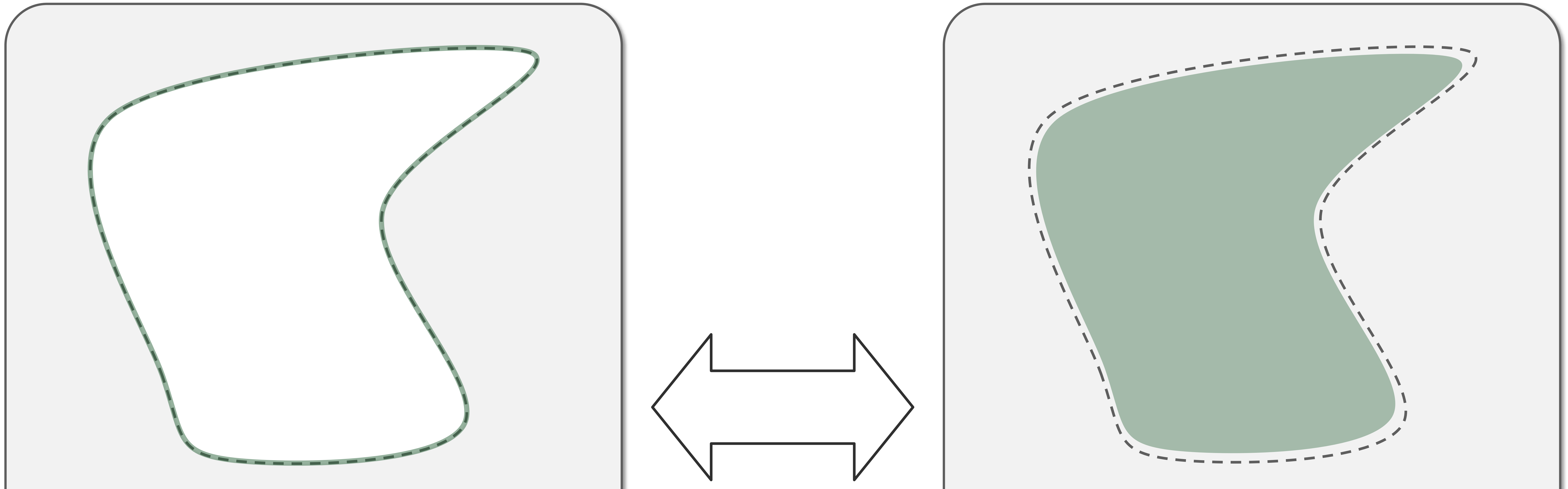
Converting Edge-Samples to Area-SAmples

$\int_{\partial D} f \vec{\mathbf{v}} \cdot \vec{\mathbf{n}}$ is estimated through edge-samples

Goal: Rewrite $\int_{\partial D} f \vec{\mathbf{v}} \cdot \vec{\mathbf{n}}$ into area integral $\int_{D} g$



The Divergence Theorem [Gauss 1813]



$$\int_{\partial D} \vec{f} \cdot \vec{n}$$

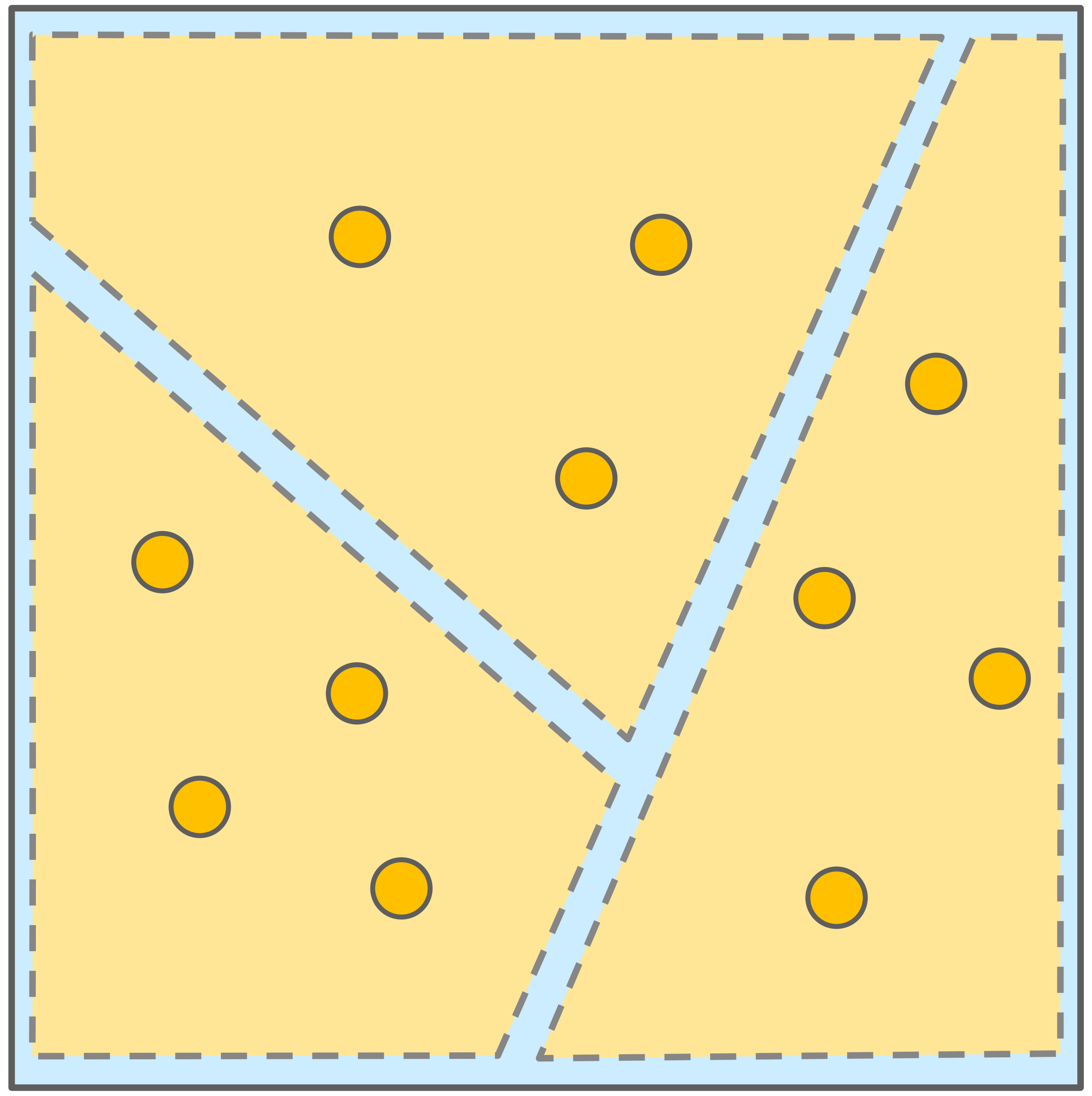
$$\int_{D} \nabla \cdot \vec{f}$$

Applying the divergence theorem to the Edge Integral

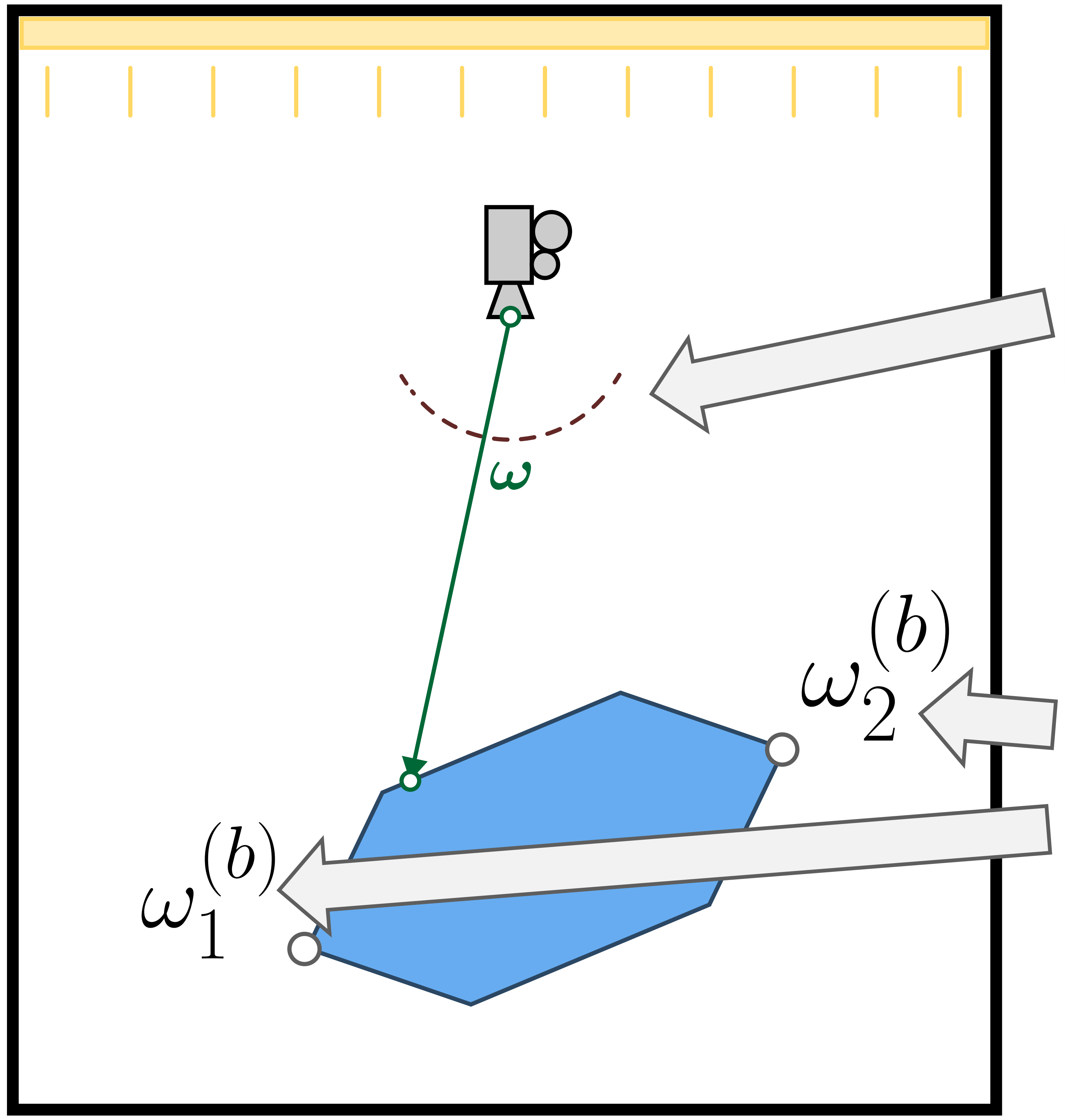
Goal: Rewrite $\int_{\partial \mathrm{D}} f \vec{\mathbf{v}} \cdot \vec{\mathbf{n}}$ into area integral $\int_{\mathrm{D}} g$

Solution: Rewrite $\int_{\partial \mathrm{D}} f \vec{\mathbf{v}} \cdot \vec{\mathbf{n}}$ into $\int_{\mathrm{D}} \nabla \cdot (\vec{\mathcal{V}}_\theta f)$

$\int_{\mathrm{D}} \nabla \cdot (\vec{\mathcal{V}}_\theta f)$ can be estimated through area-samples



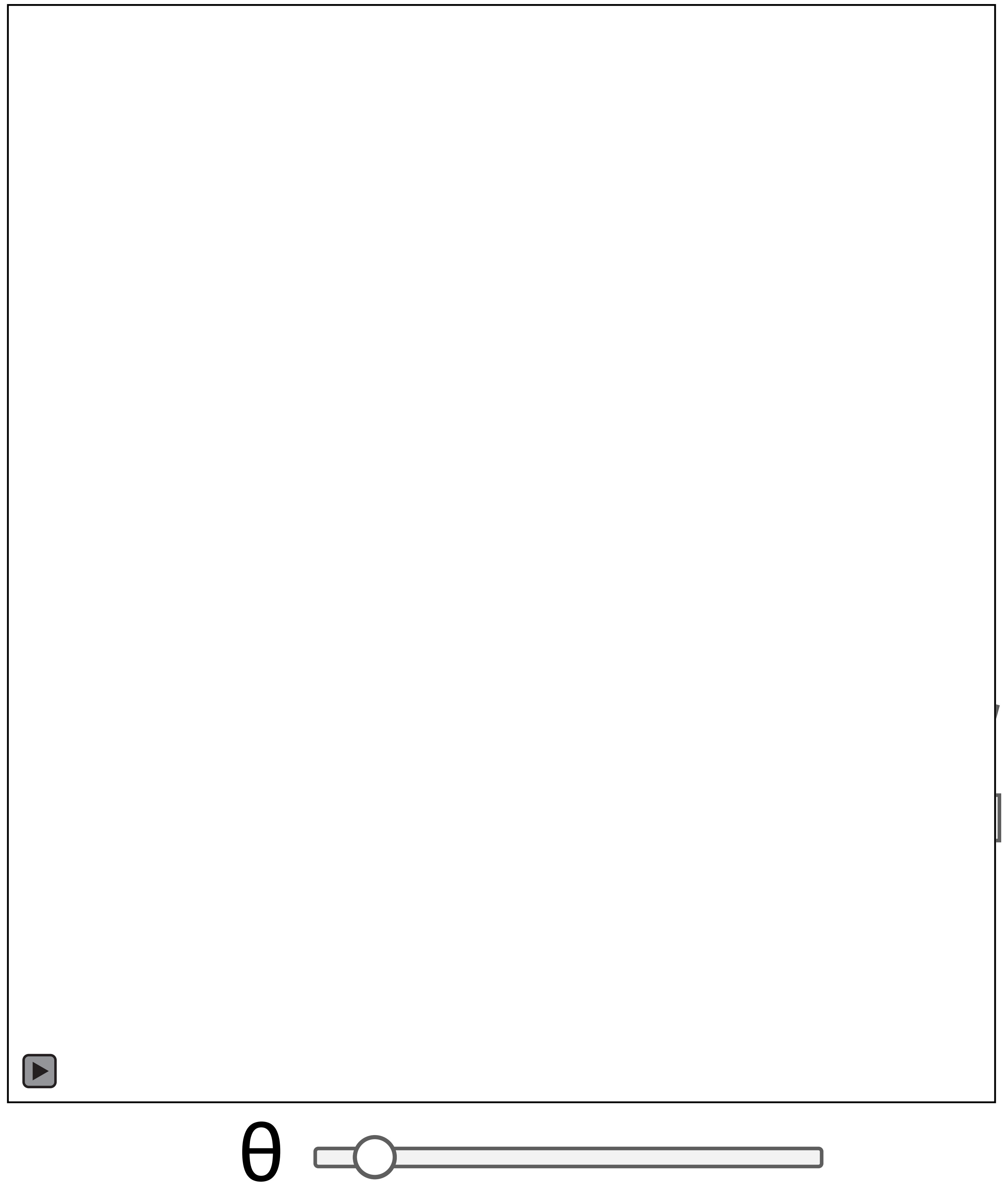
A 2D Example Scene



$\omega \in \Omega$, the domain of integration

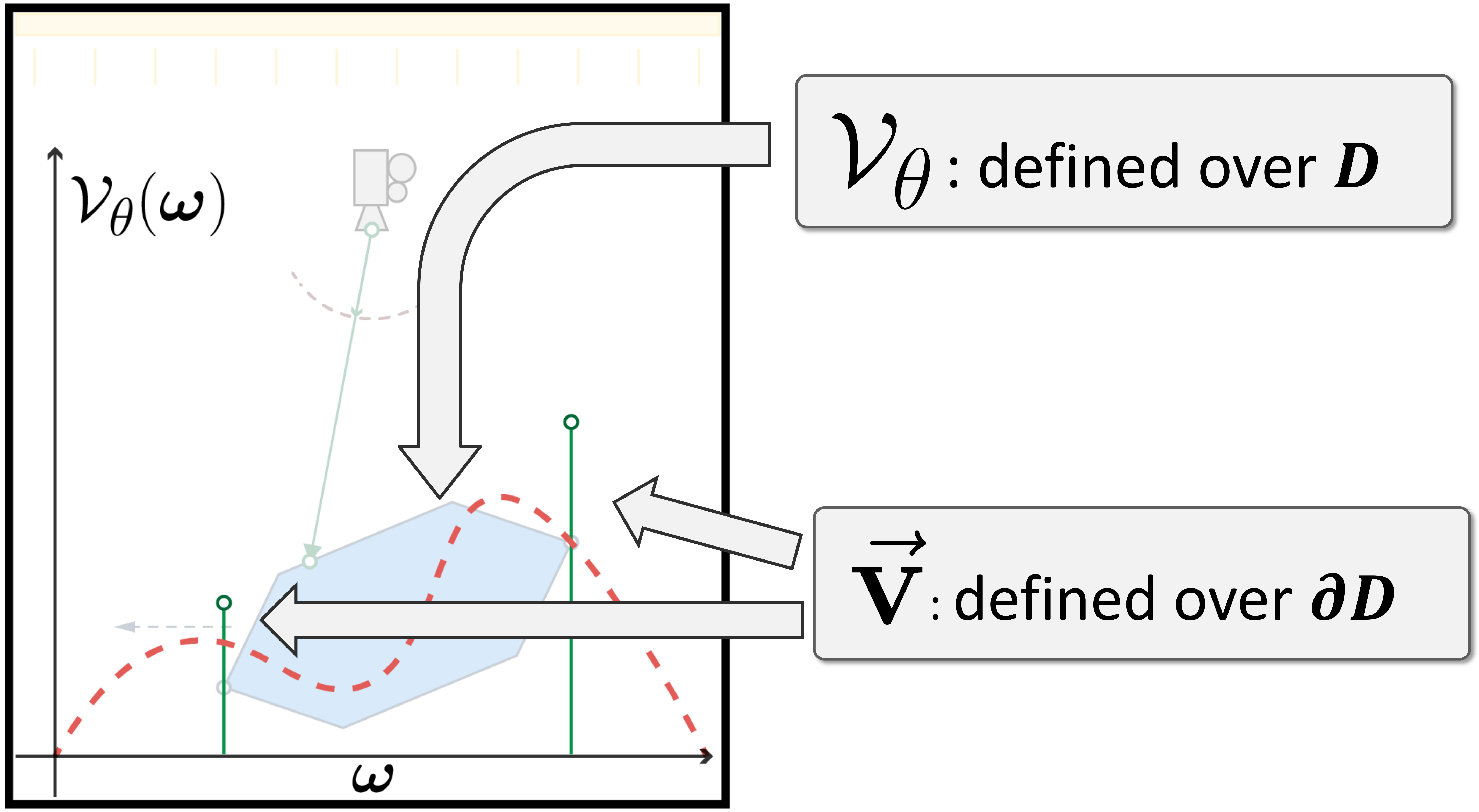
$\omega_1^{(b)}, \omega_2^{(b)} \in \partial\Omega$, the discontinuous set

Velocity $\vec{v}$: the Boundary derivative



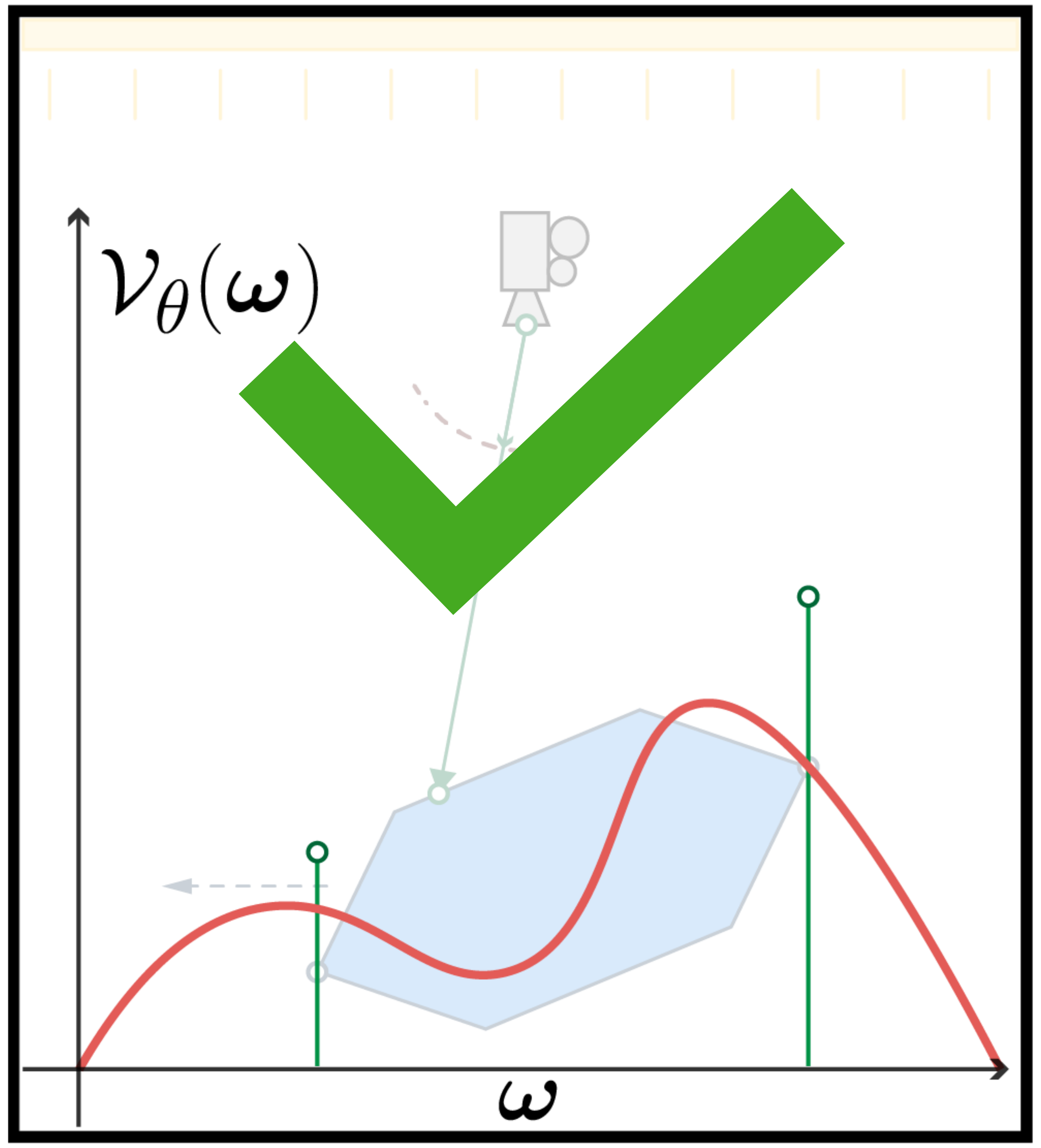
$\partial_\theta \omega_i^{(b)}$: Derivative of boundary position w.r.t **θ**

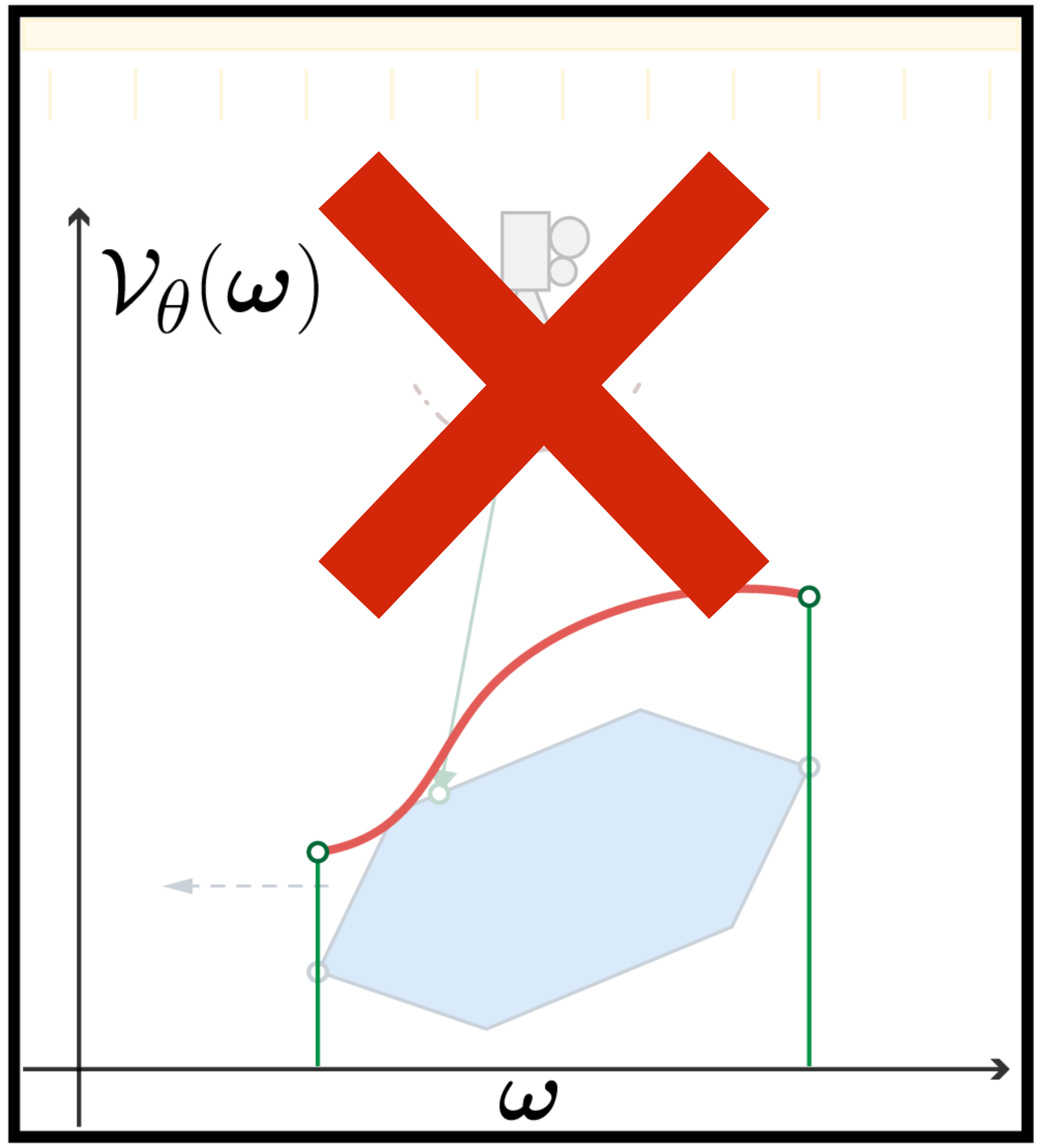
Warp Field $\vec{\mathcal{V}}_\theta$: Extension of $\vec{\mathrm{v}}$ to all points



Validity of $\vec{\mathcal{V}}_\theta$

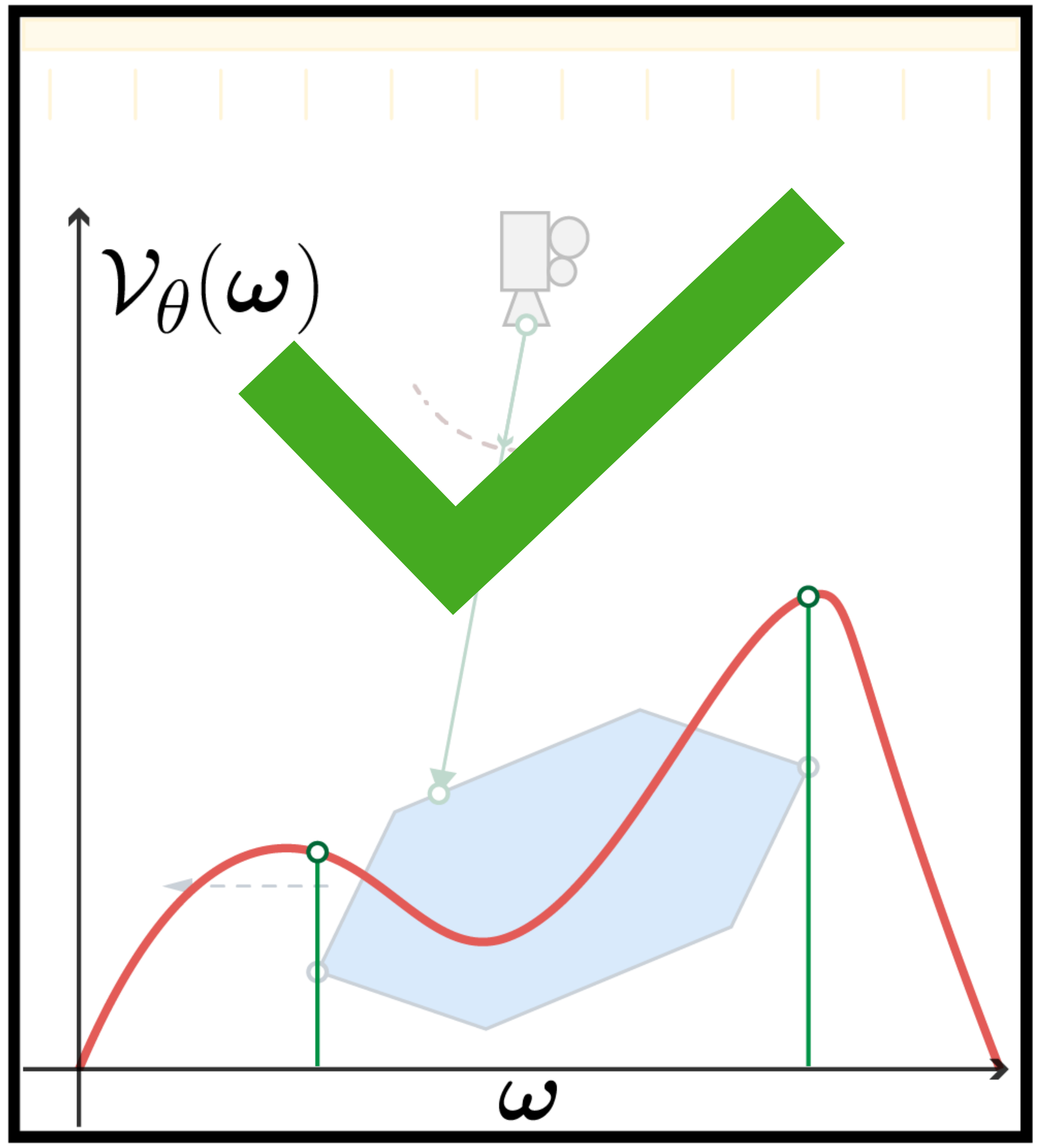
Rule 1: Continuous

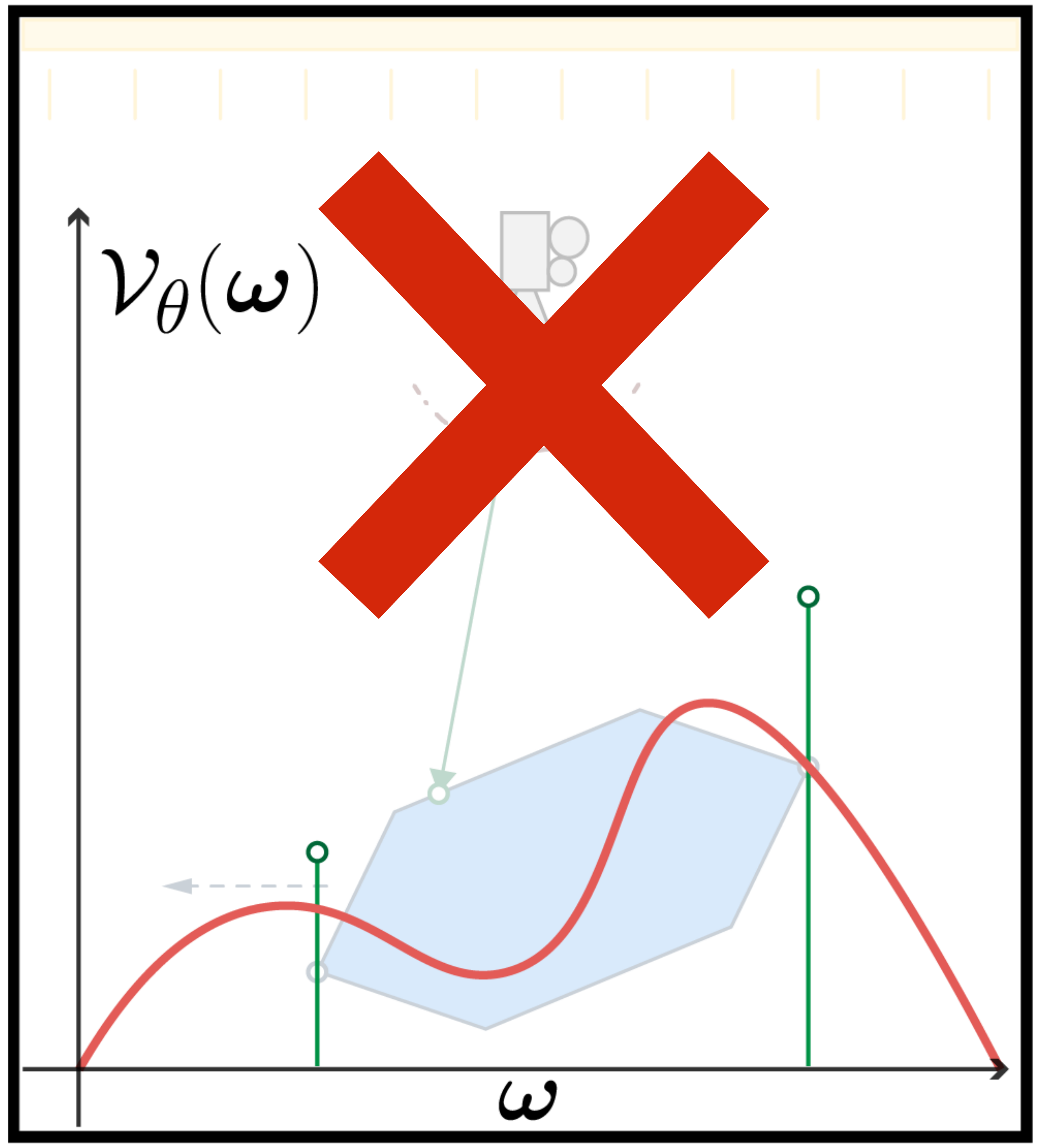




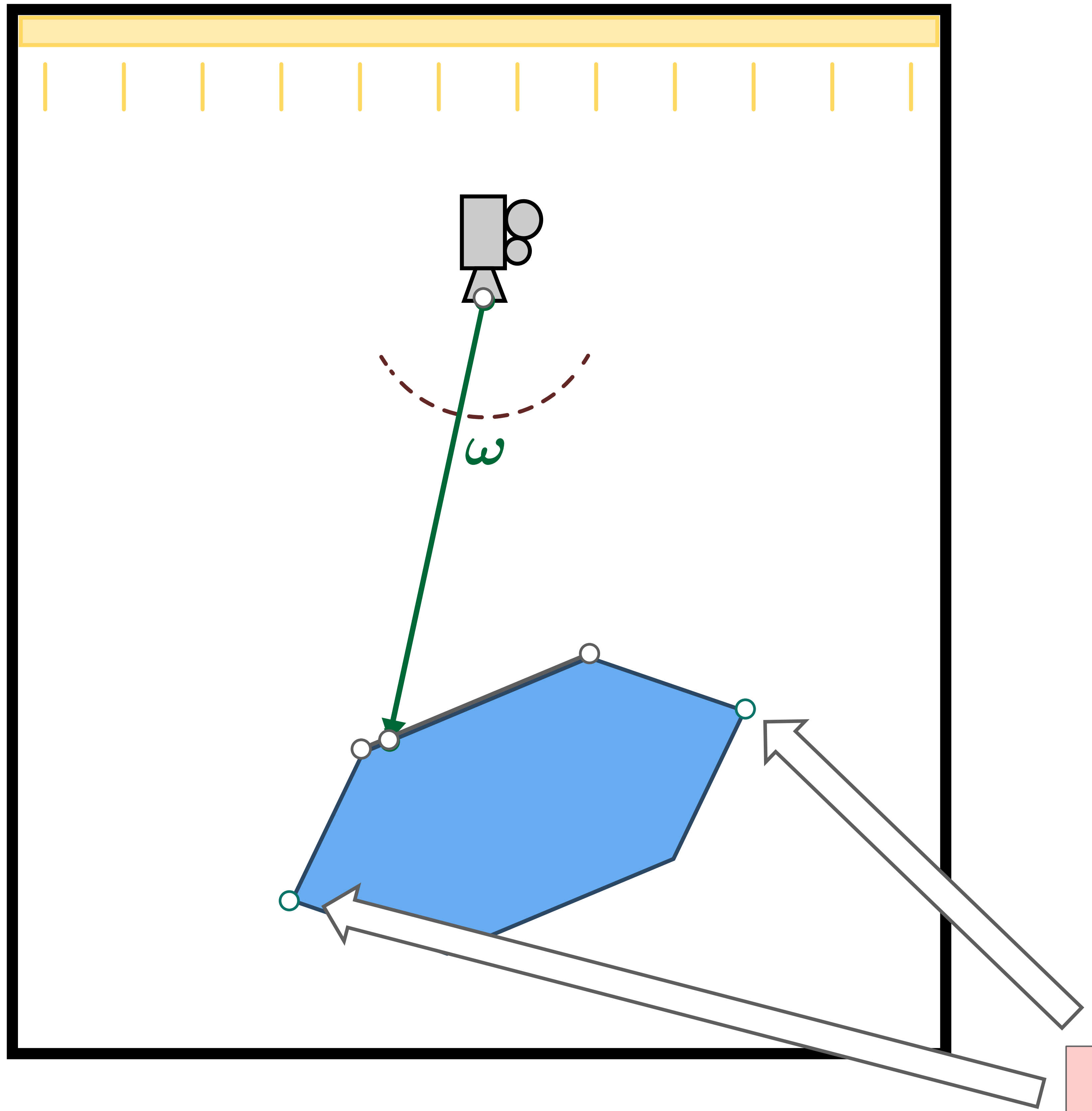
Validity of $\vec{\mathcal{V}}_\theta$

Rule 2: Boundary Consistent





Interpolation without knowledge of boundaries



Available quantities

- Origin point
- Ray
- Intersection
- Primitive

No access to discontinuity points

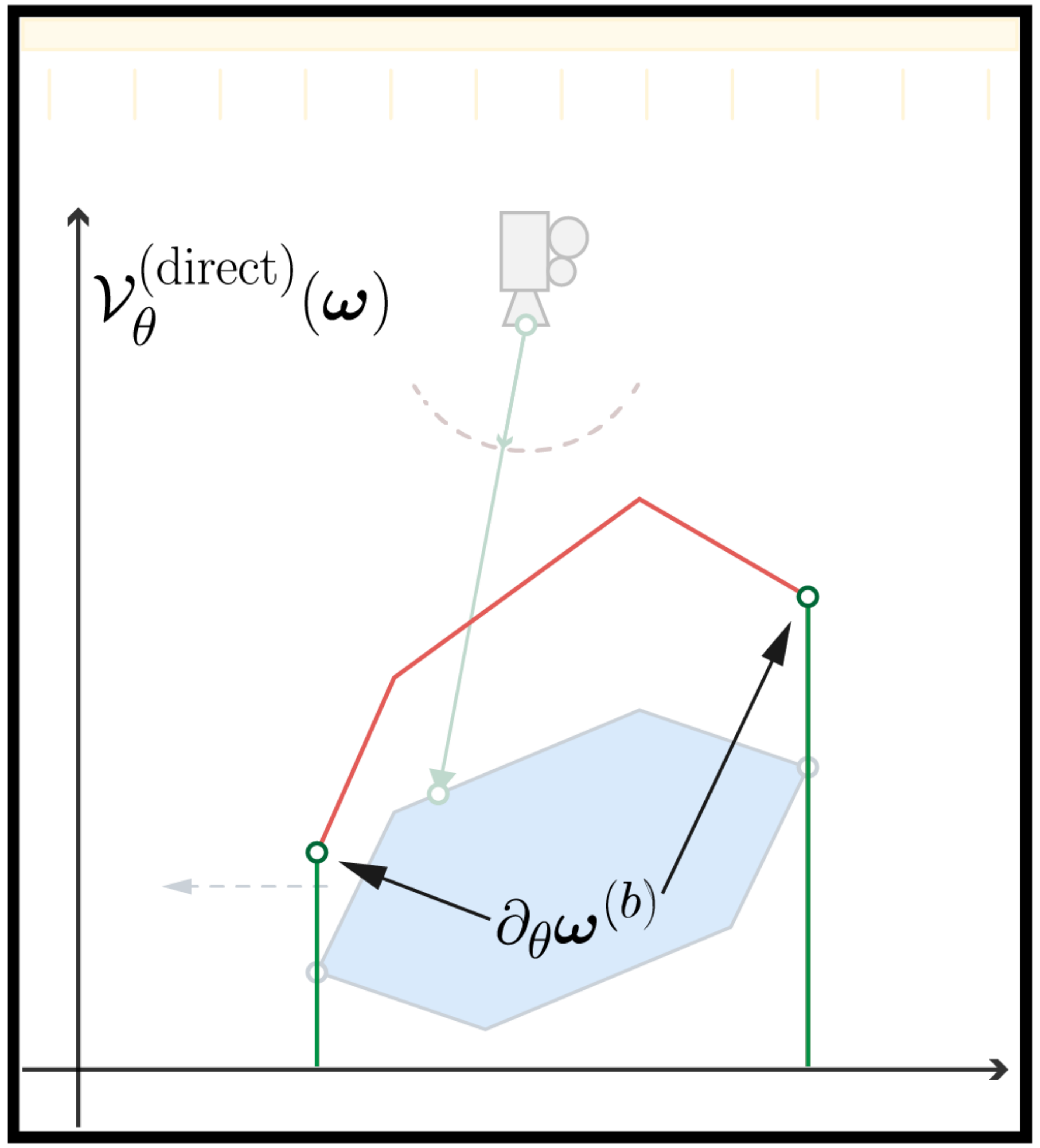
Constructing $\vec{\mathcal{V}}_\theta$

Attempt 1 → Find $\partial_\theta \boldsymbol{\omega}$ through *implicit derivative*

(Incorrect)

$$\mathbf{y} = \textsc{Intersect}(\boldsymbol{\omega}, \theta) \Longrightarrow \partial_\theta \boldsymbol{\omega} = \frac{\partial_{\boldsymbol{\omega}} \mathbf{y}}{\partial_\theta \mathbf{y}}$$

At all points (not just boundaries)



+ Boundary consistent

- Not continuous

Constructing $\vec{\mathcal{V}}_\theta$

Attempt 2 ➡ Filter *Attempt 1* with a Gaussian filter

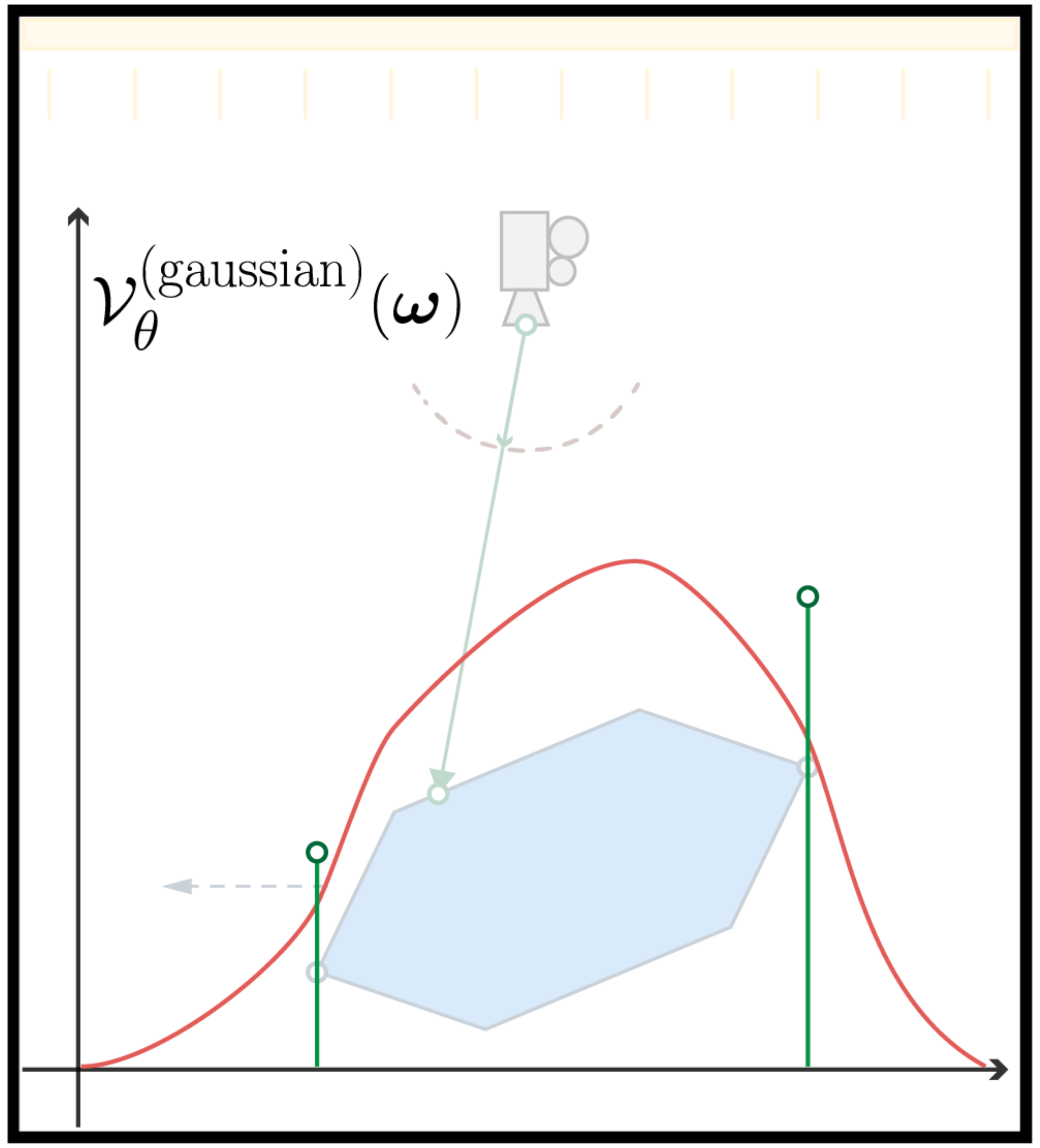
(Incorrect)

$$\int_{\Omega'} k(\boldsymbol{\omega}, \boldsymbol{\omega}') \frac{\partial_{\boldsymbol{\omega}} \mathbf{y}}{\partial_\theta \mathbf{y}}$$

k(.,.) = Gaussian filter

+ Continuous

\- Not boundary consistent



Boundary-Aware Weighting

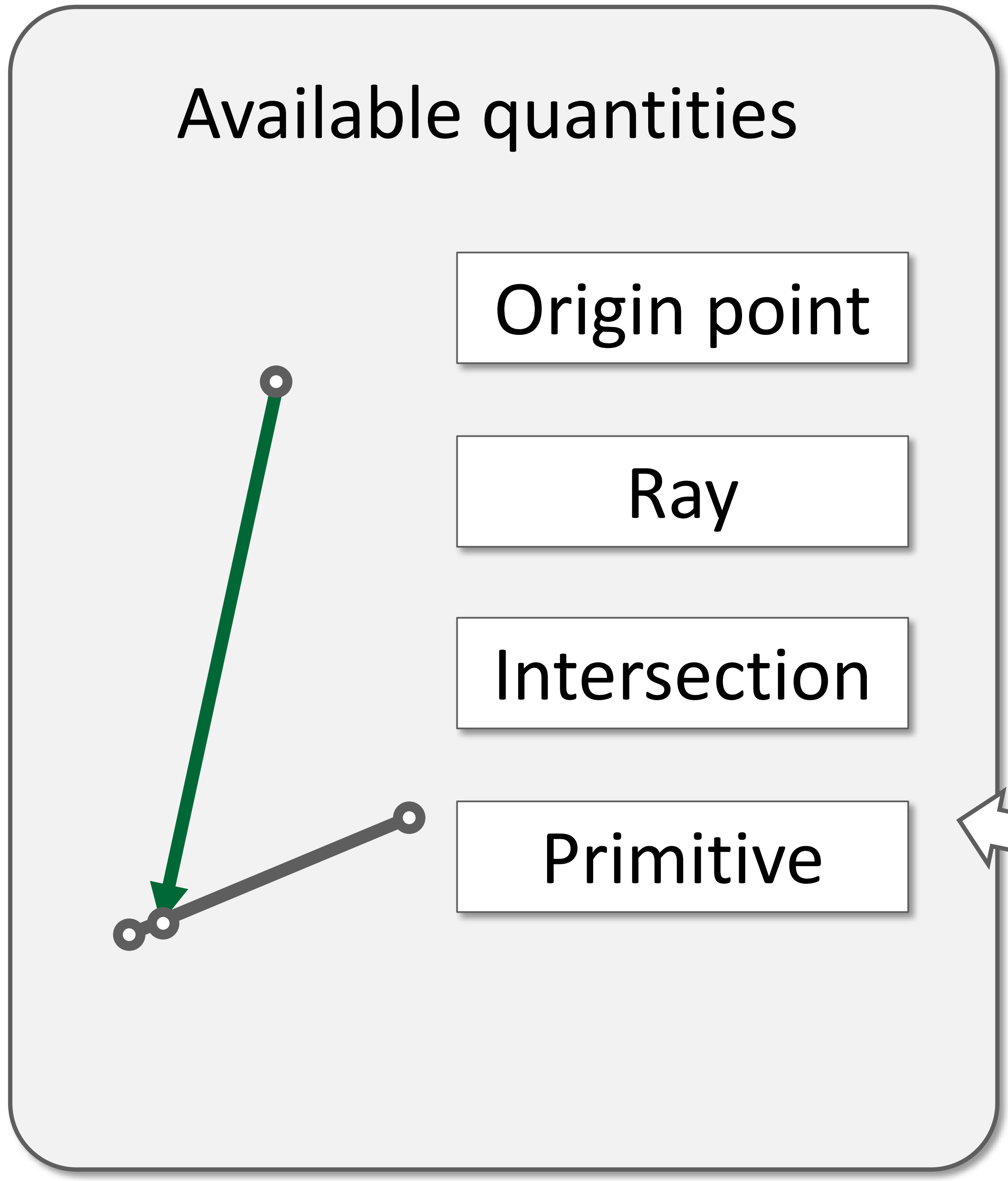
<u>Goal</u>: Find weights $k(\boldsymbol{\omega}, \boldsymbol{\omega}')$ s.t. $\vec{\mathcal{V}}_\theta = \frac{\partial_\omega \mathbf{y}}{\partial_\theta \mathbf{y}}$ at boundaries.

Ideal weighting function

$\boldsymbol{\omega}$

Approach Dirac delta near boundaries

Boundary-Aware Weighting



Don't have — Discontinuity set *(Boundary sampling)*

Do have — Implicit function of the boundary *(Boundary testing)*

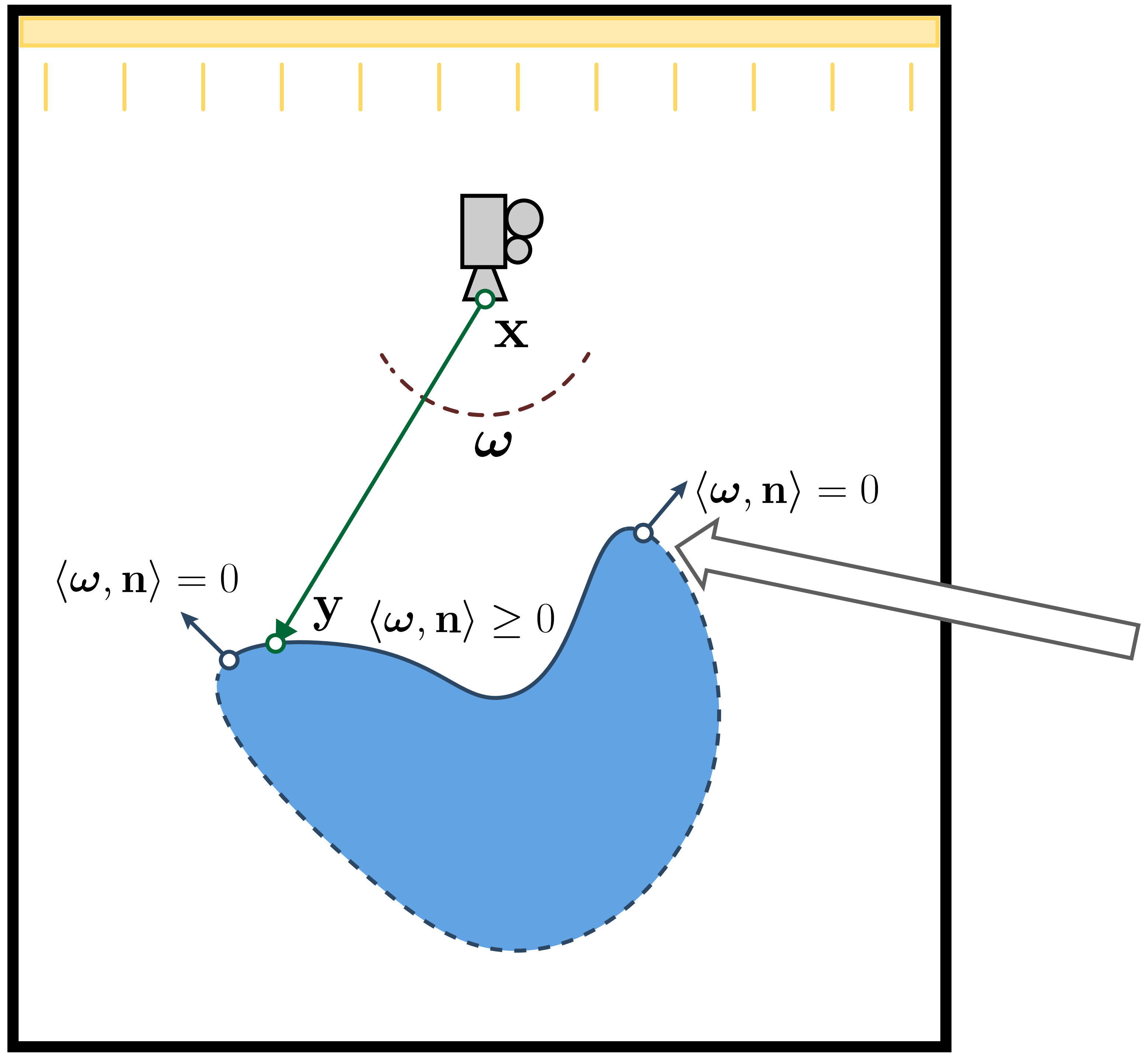
$\mathcal{B}(\boldsymbol{\omega}')$: Boundary test function

such that

$$\mathcal{B}(\boldsymbol{\omega}') = 0 \text{ for } \boldsymbol{\omega}' \in \partial\Omega$$

Boundary-Aware Weighting

Implicit Boundary through *geometric normals*



$$\mathcal{B}(\boldsymbol{\omega}) = \sigma(\langle \boldsymbol{\omega}, \mathbf{n} \rangle)$$

$$\langle \boldsymbol{\omega}, \mathbf{n} \rangle = 0$$

at boundaries

y sampling)

boundary

g)

Constructing $\vec{\mathcal{V}}_\theta$

Our Approach ⟶ Filter *Attempt 1* with harmonic weights

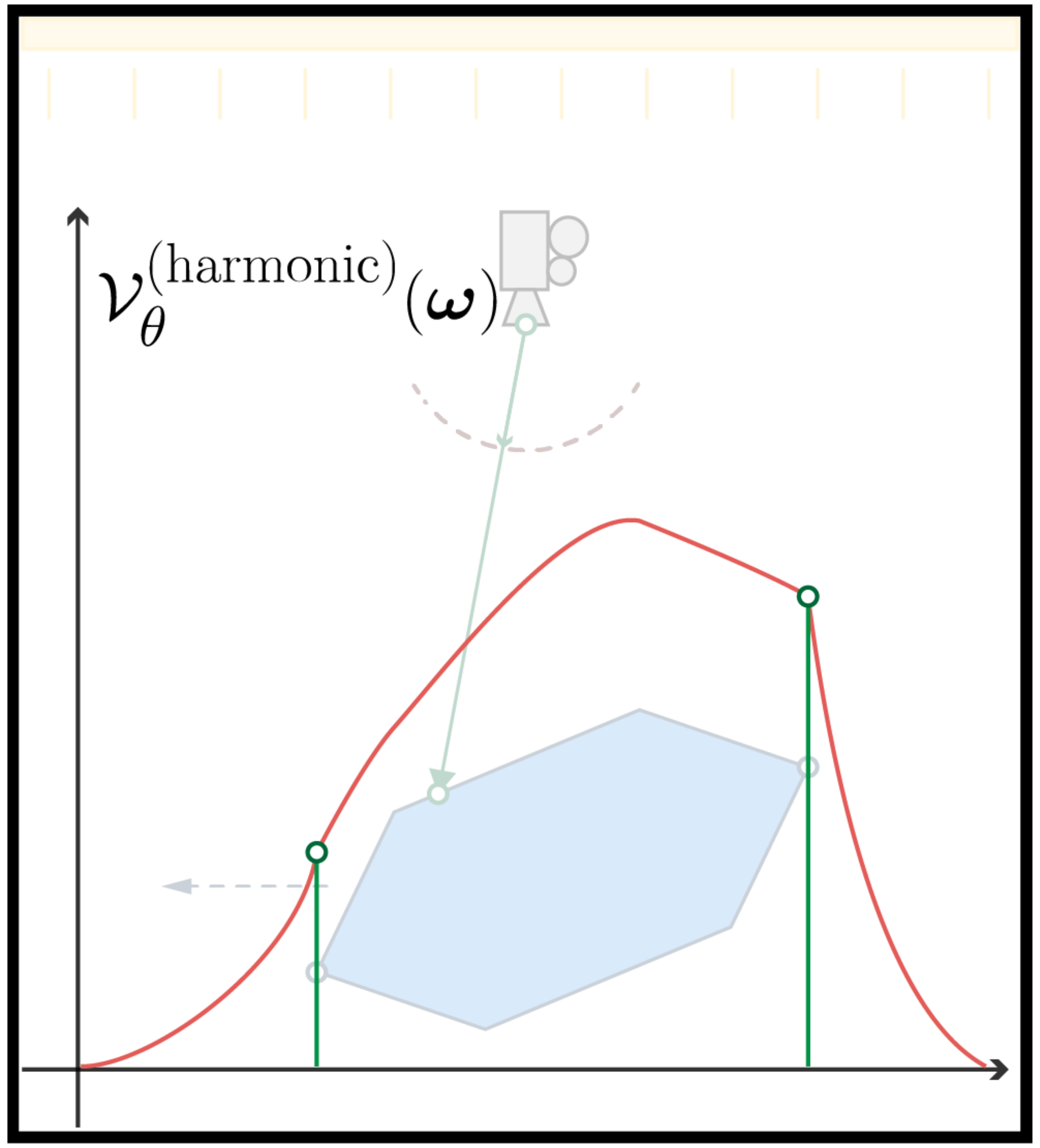
$$k(\boldsymbol{\omega}, \boldsymbol{\omega}') = \frac{1}{\boxed{\mathcal{D}(\boldsymbol{\omega}, \boldsymbol{\omega}')} + \boxed{\mathcal{B}(\boldsymbol{\omega}')}}$$

Distance function

Boundary test

+ Boundary consistent

+ Continuous



Computing $\vec{\mathcal{V}}_\theta$

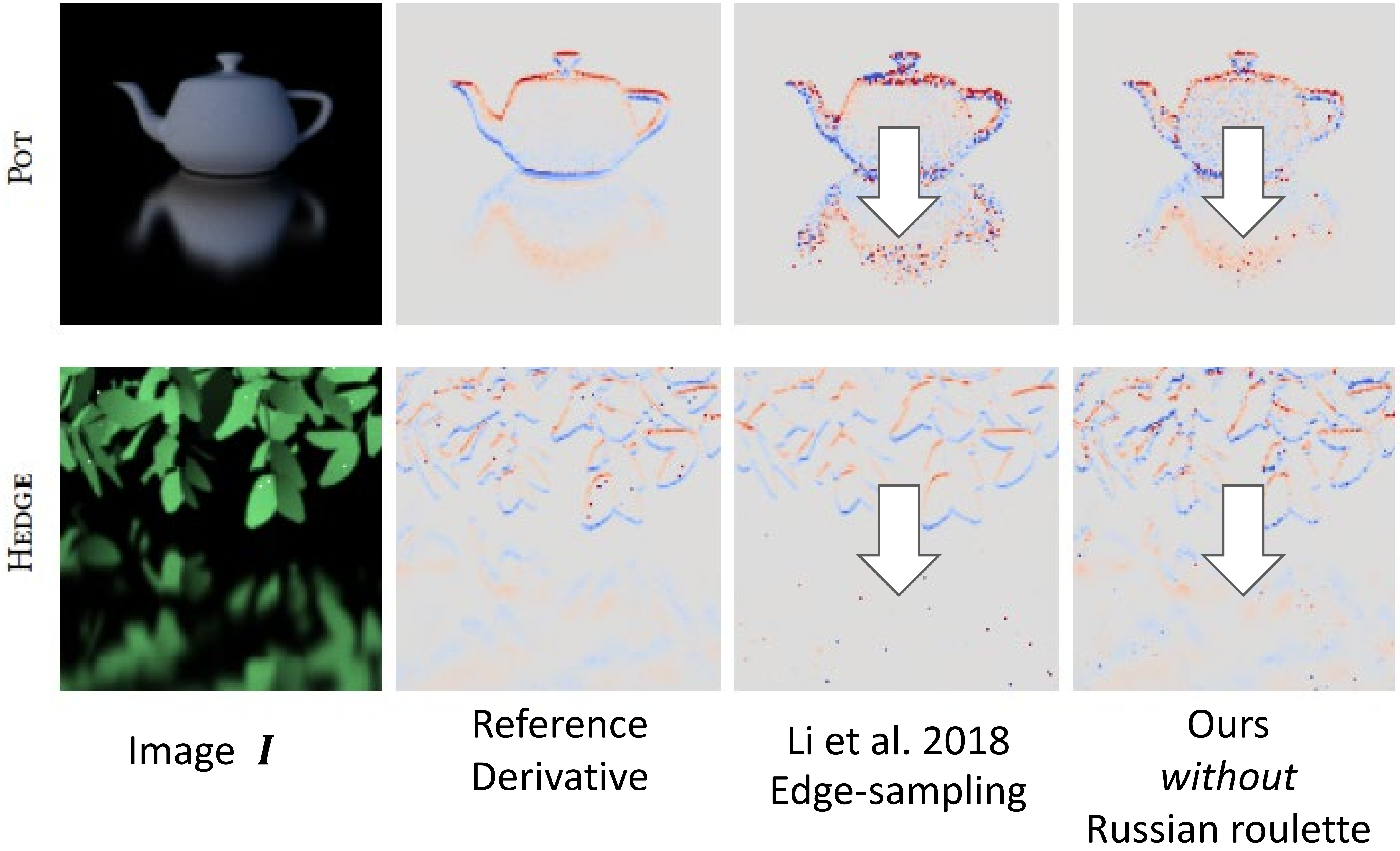
1. Sample **path** using path tracer *(N paths)*

For each bounce:

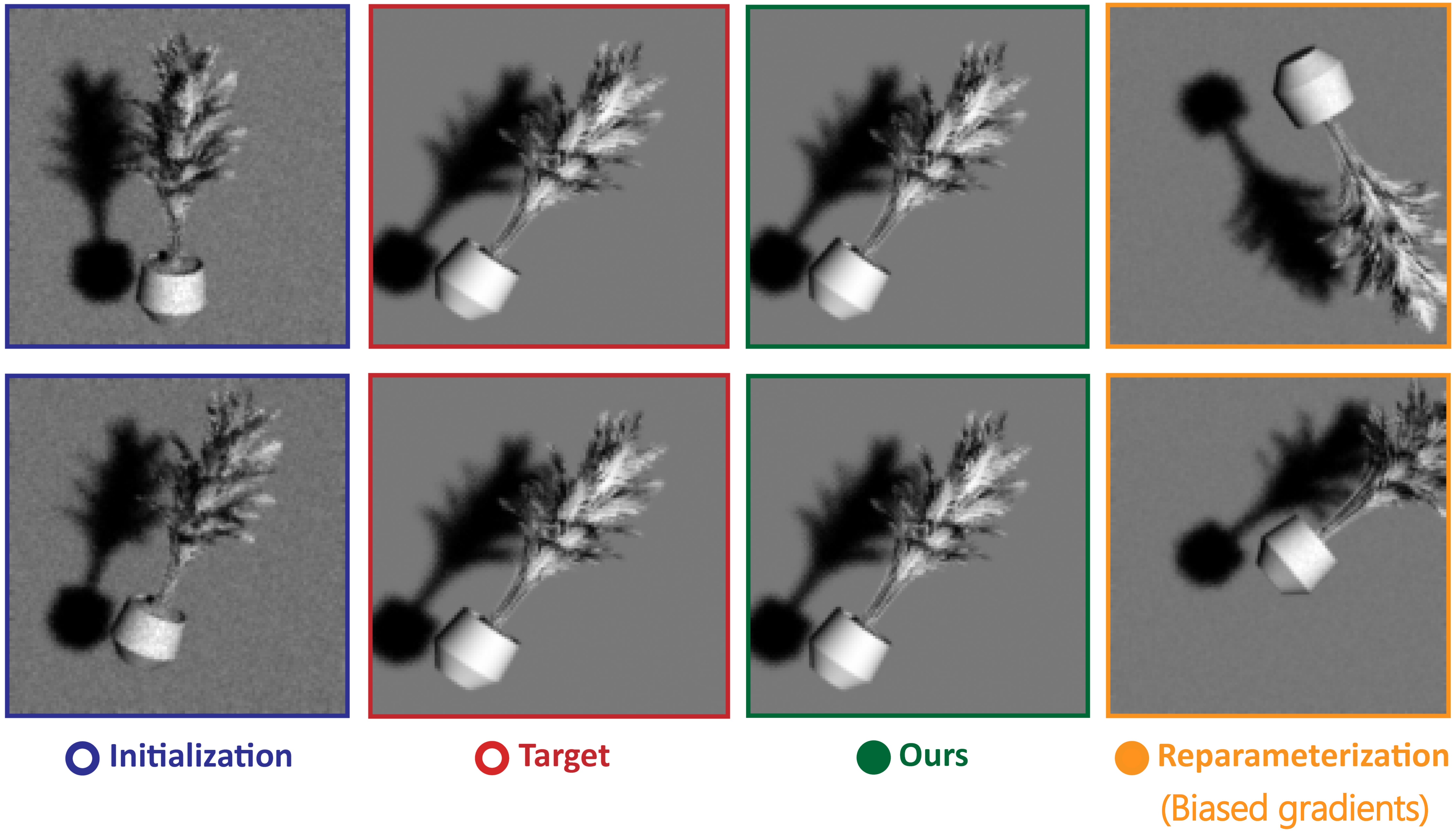
2. Sample **auxiliary** rays *(N’ rays)*
3. Compute boundary term ***B()*** locally
4. Compute weight **k(.,.)** and $\partial_\theta \boldsymbol{\omega}$
5. Find weighted mean

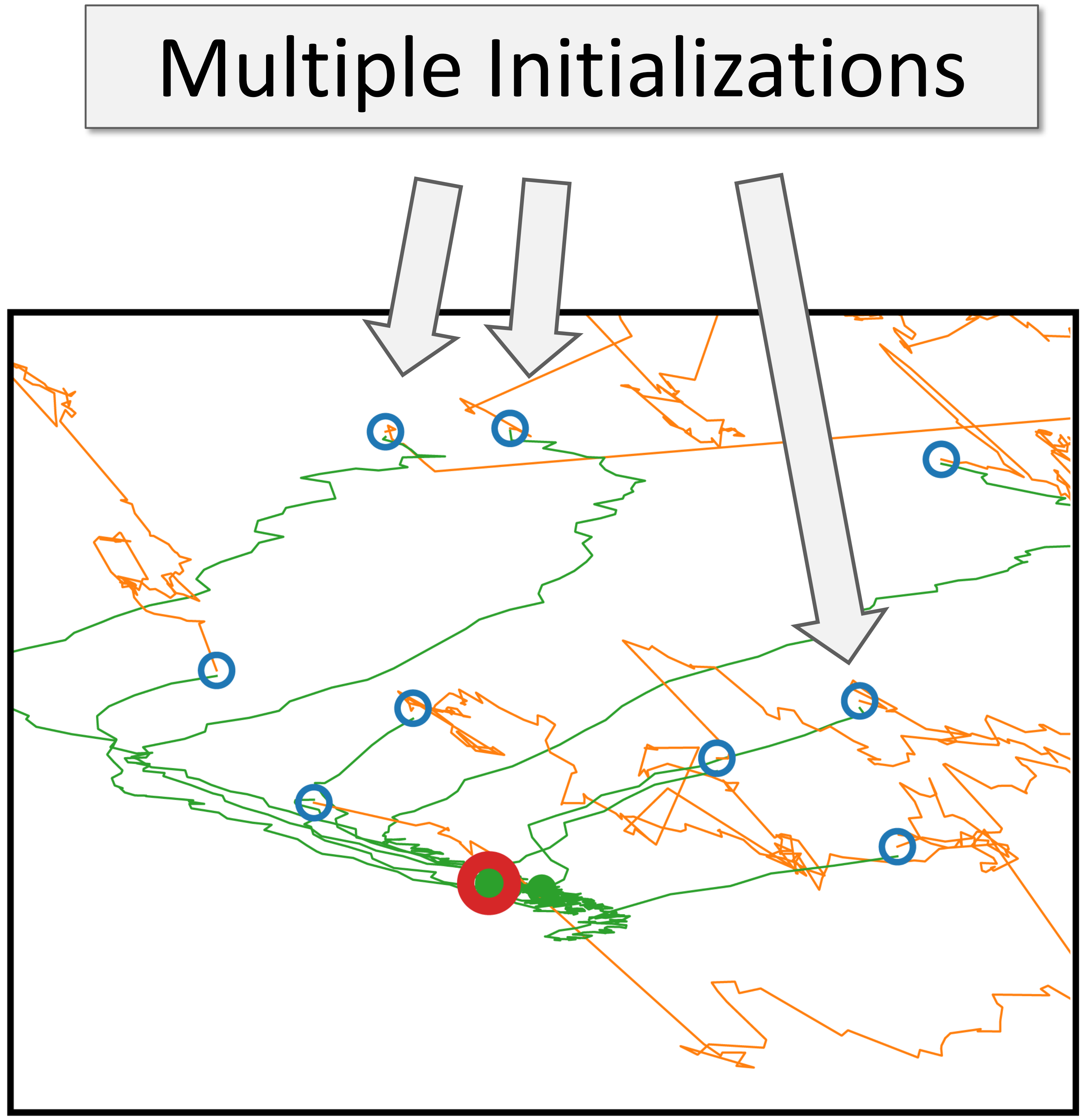
RESULTS

Area-sampling handles higher-order effects better



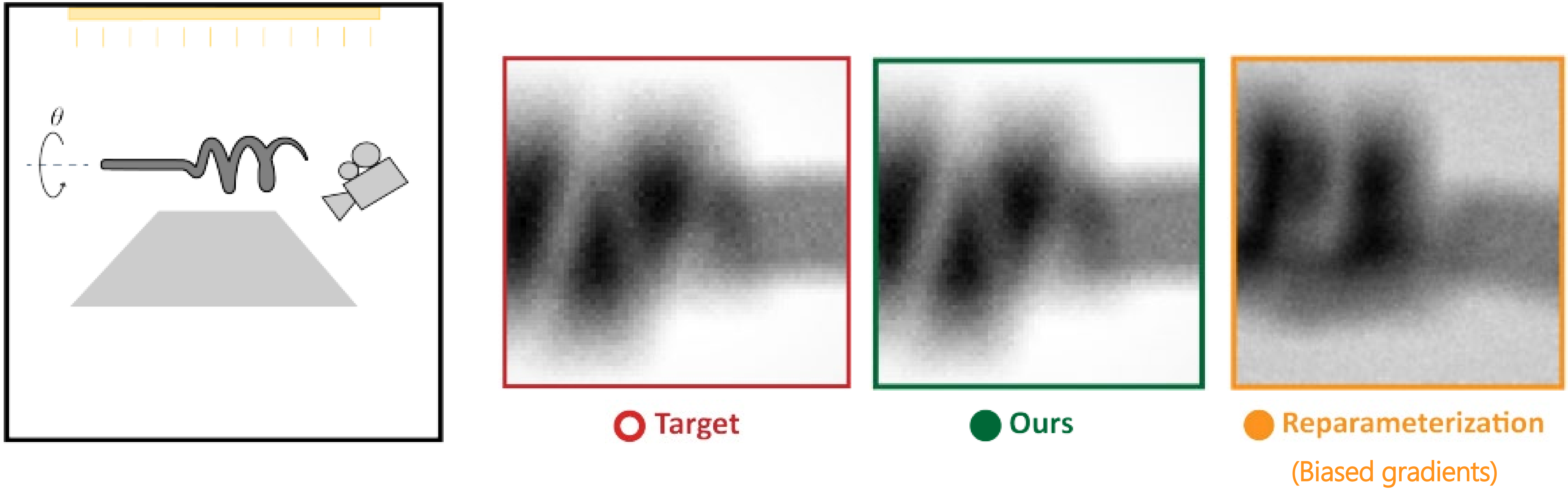
Pose estimation can fail with biased gradients





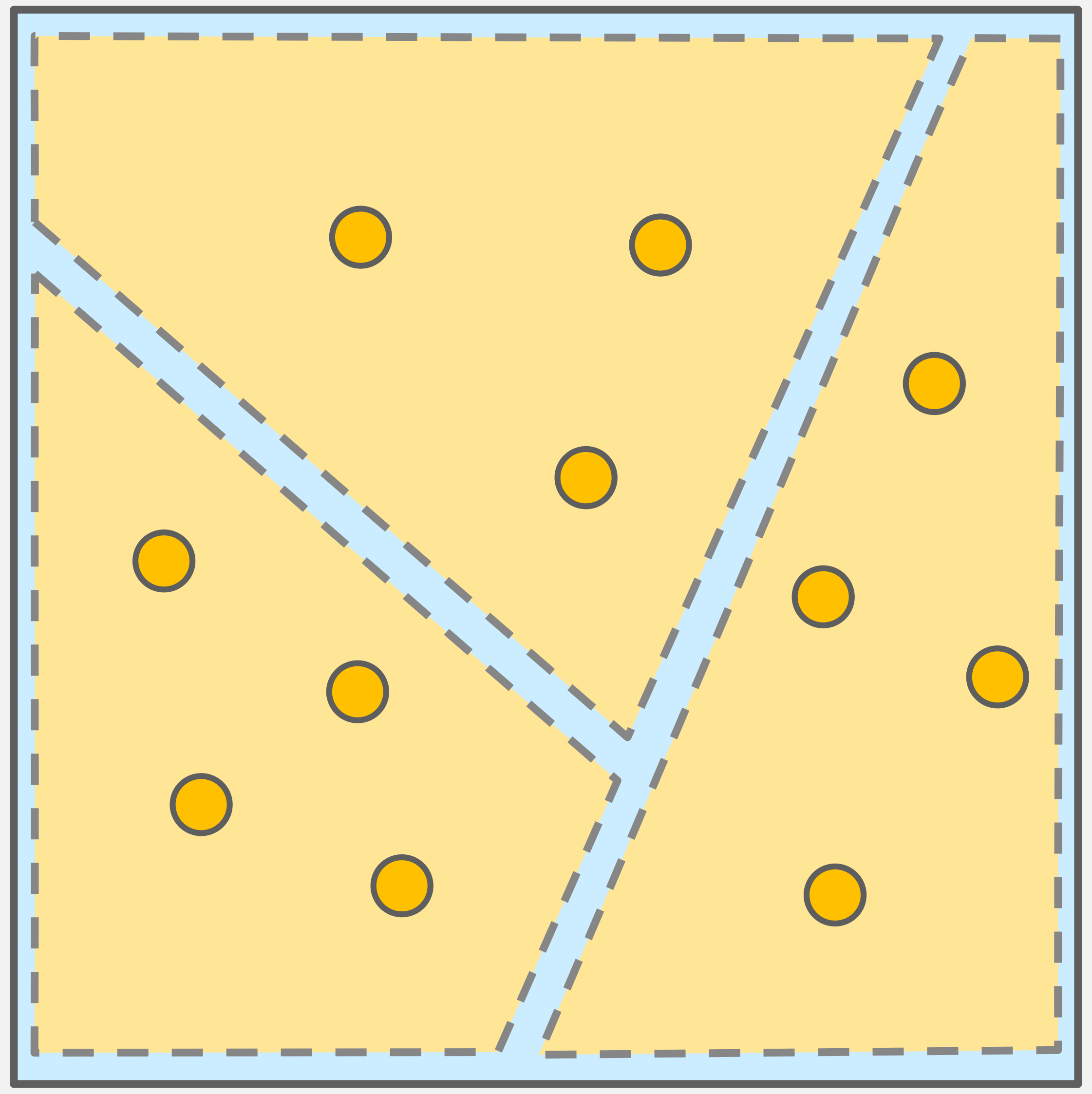
Optimization trajectories

Pose estimation can fail with biased gradients

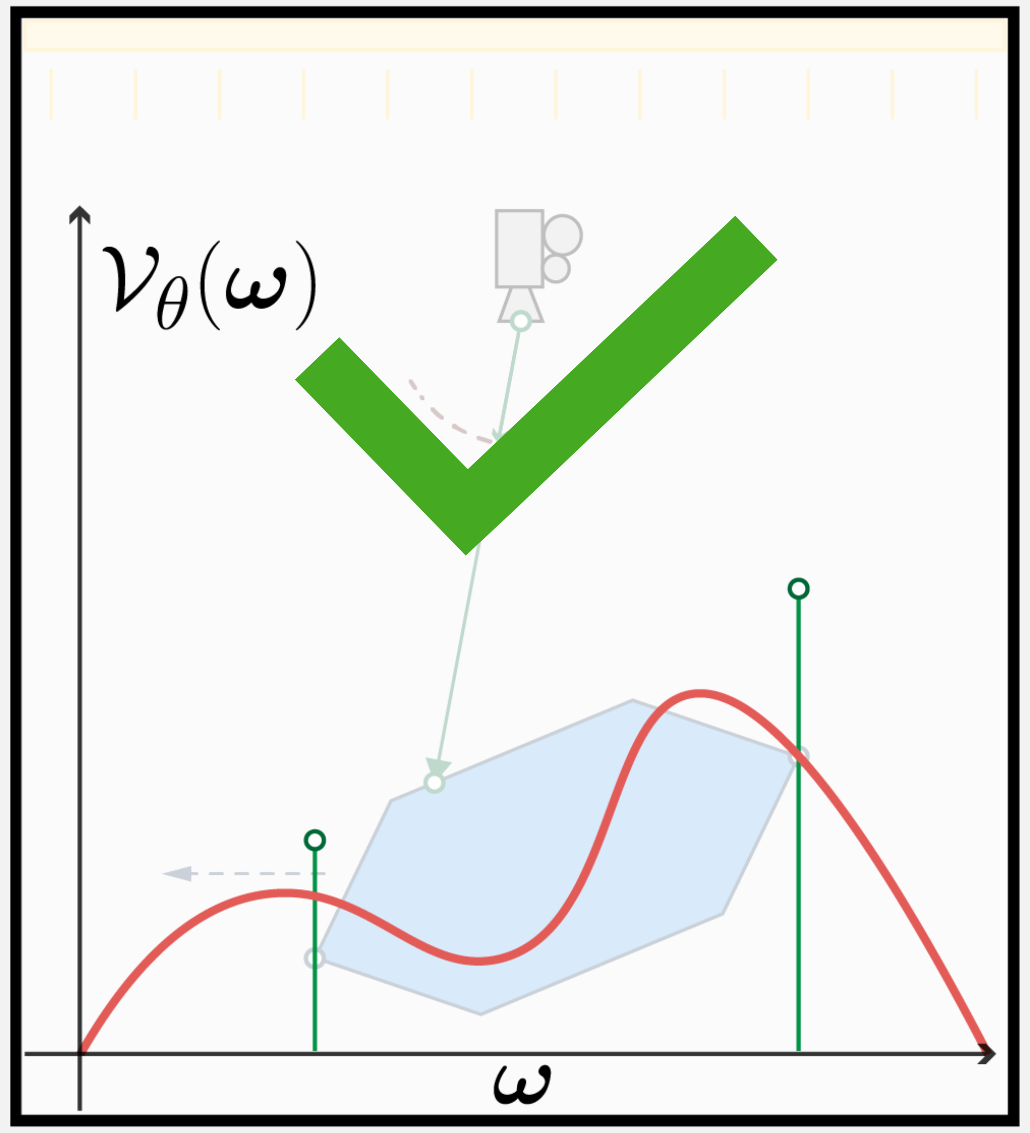


Summary of Warped-area sampling

Edge-integral to *Area*-integral

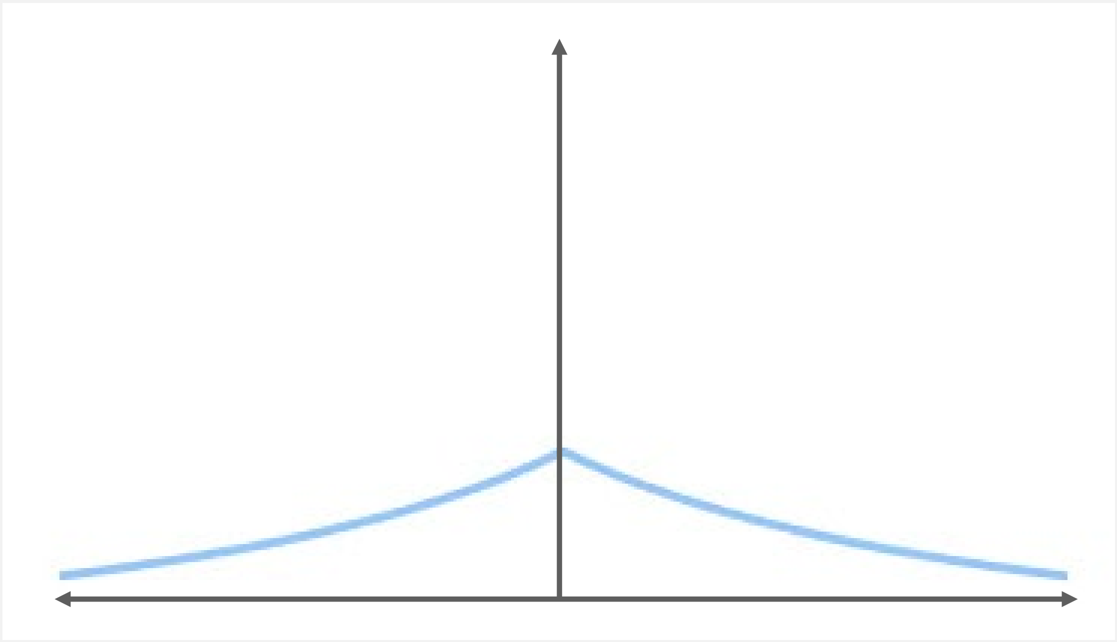


Warp field conditions

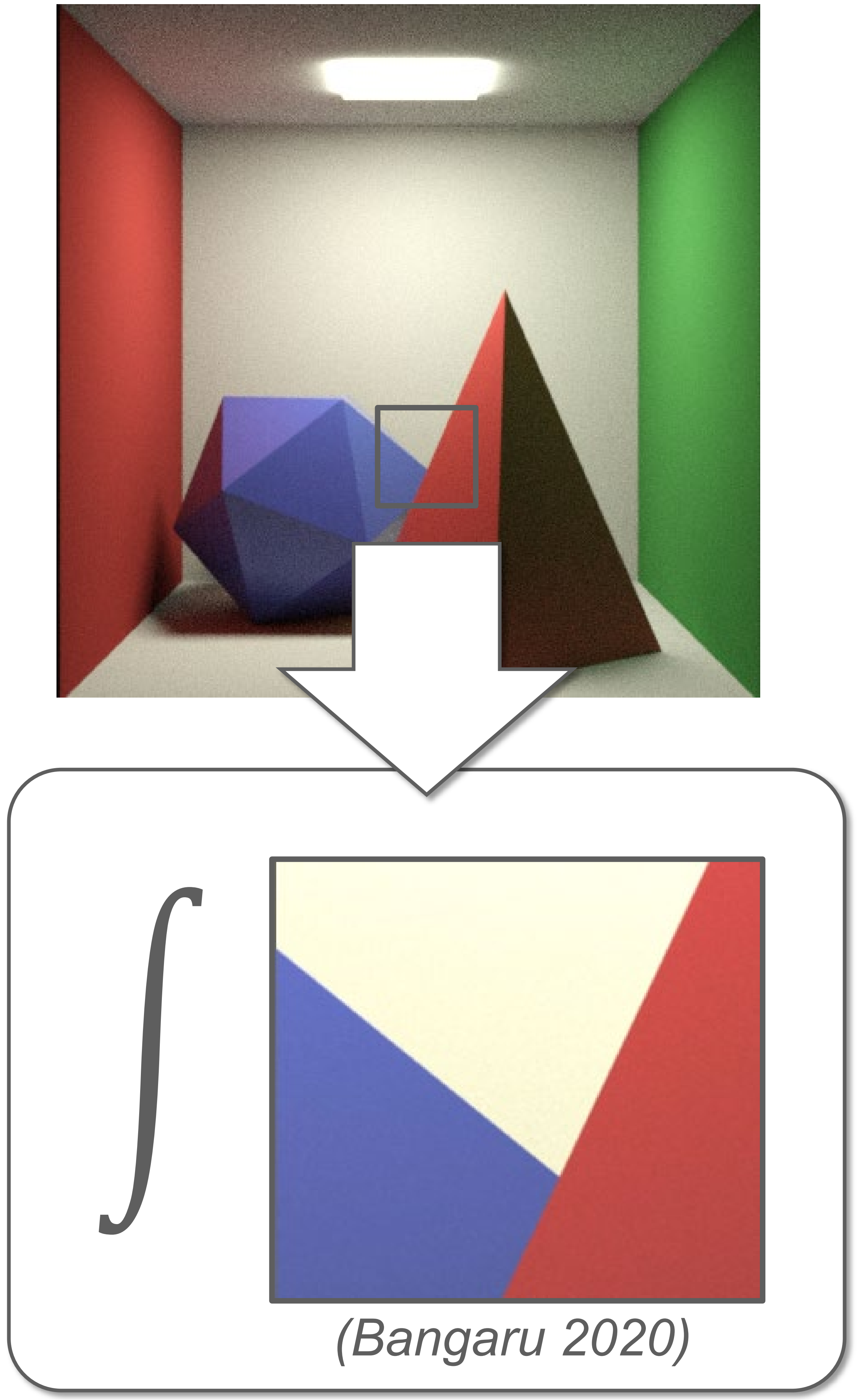


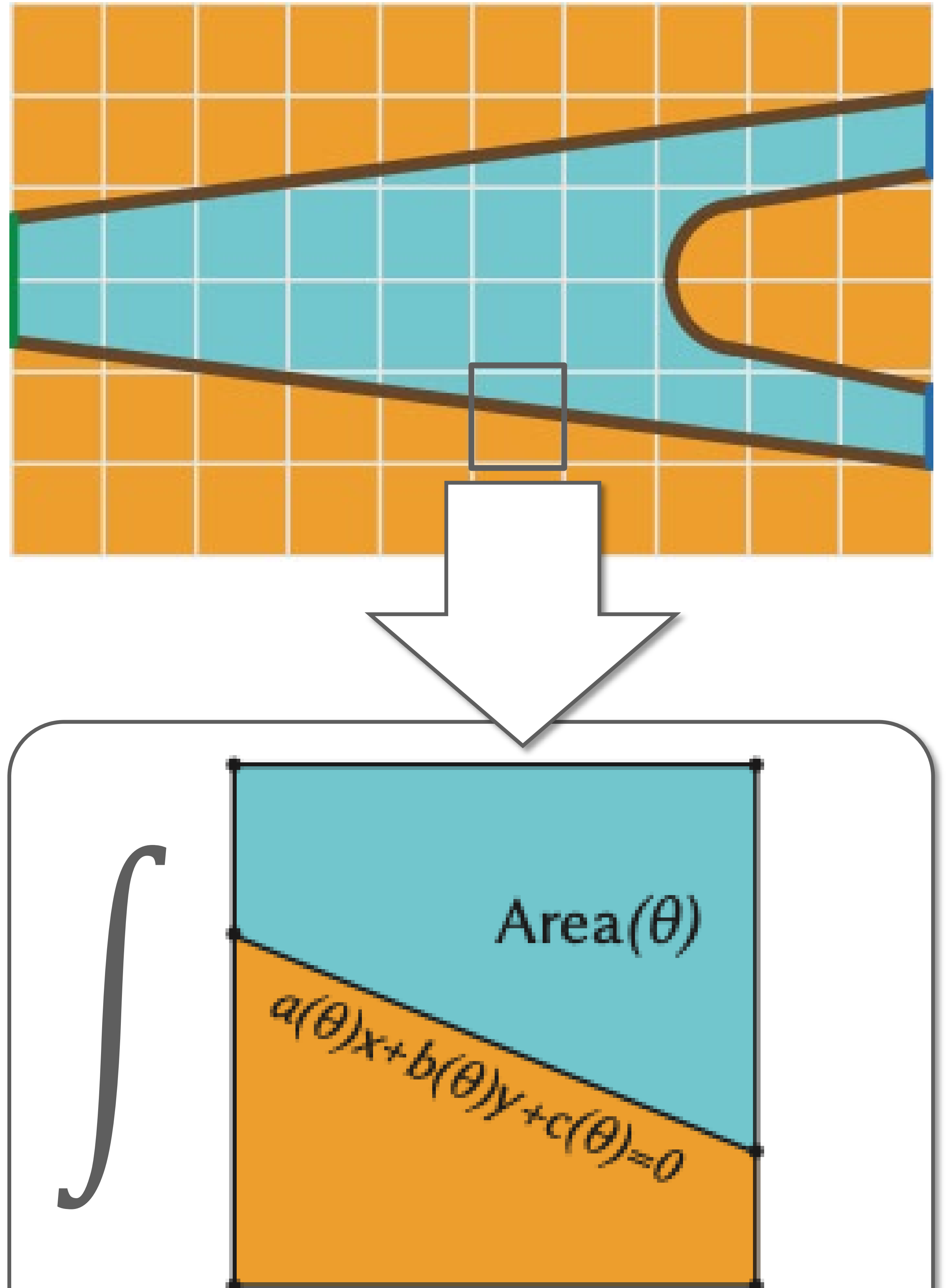
Harmonic interpolation

$$k(\boldsymbol{\omega}, \boldsymbol{\omega}') = \frac{1}{\mathcal{D}(\omega, \omega') + \mathcal{B}(\omega')}$$

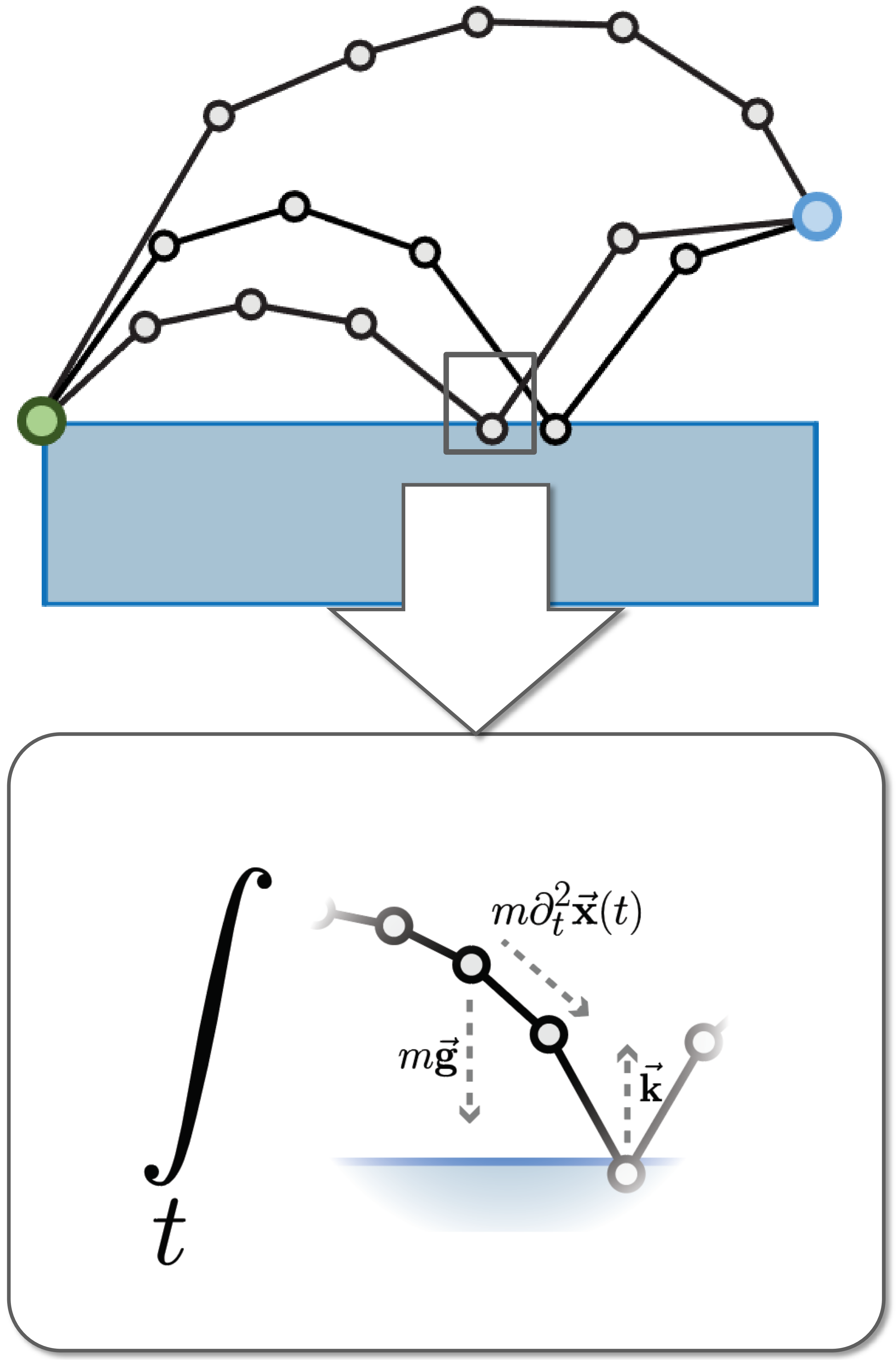


Many programs in graphics have this problem

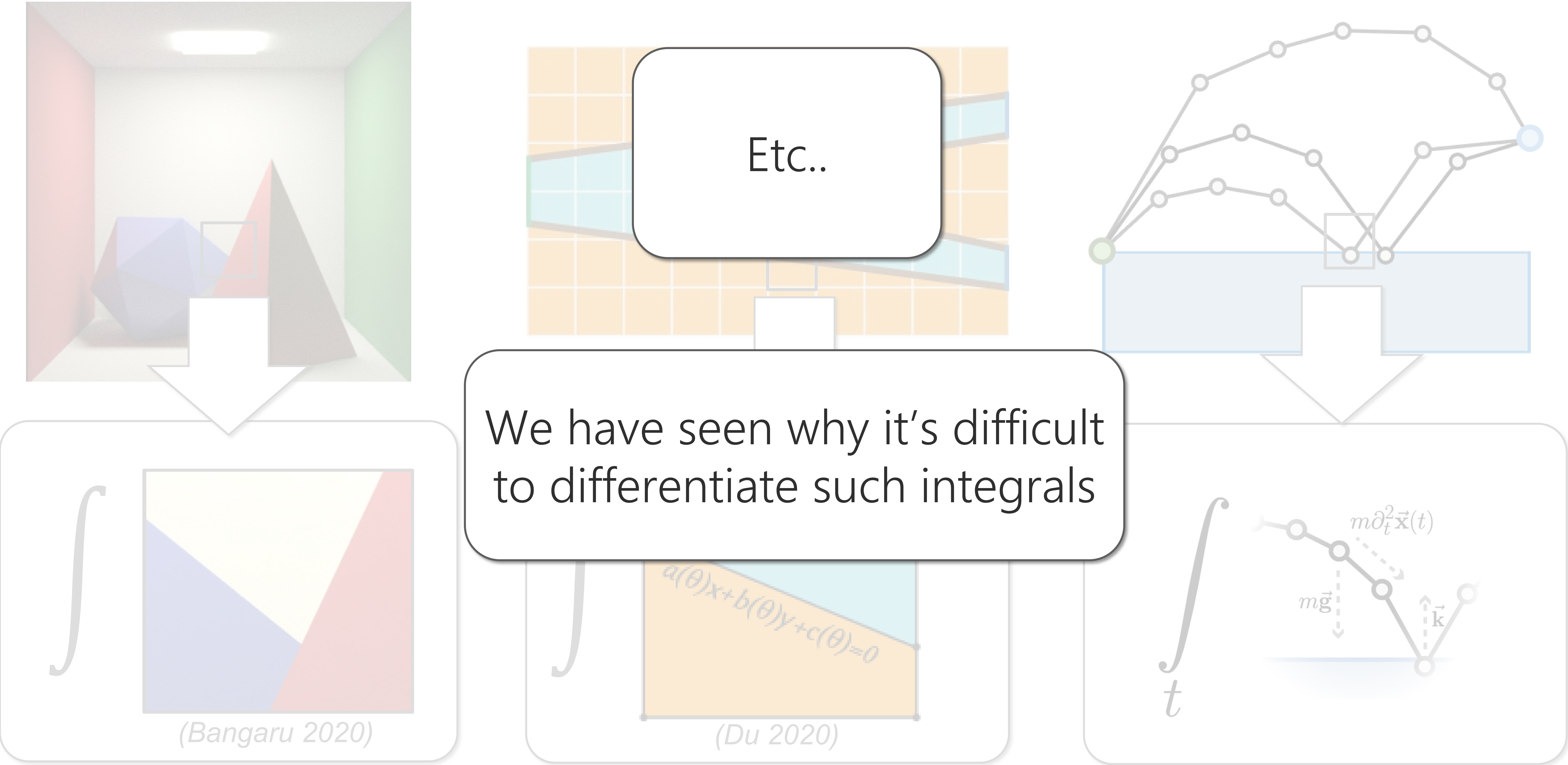

(Bangaru 2020)



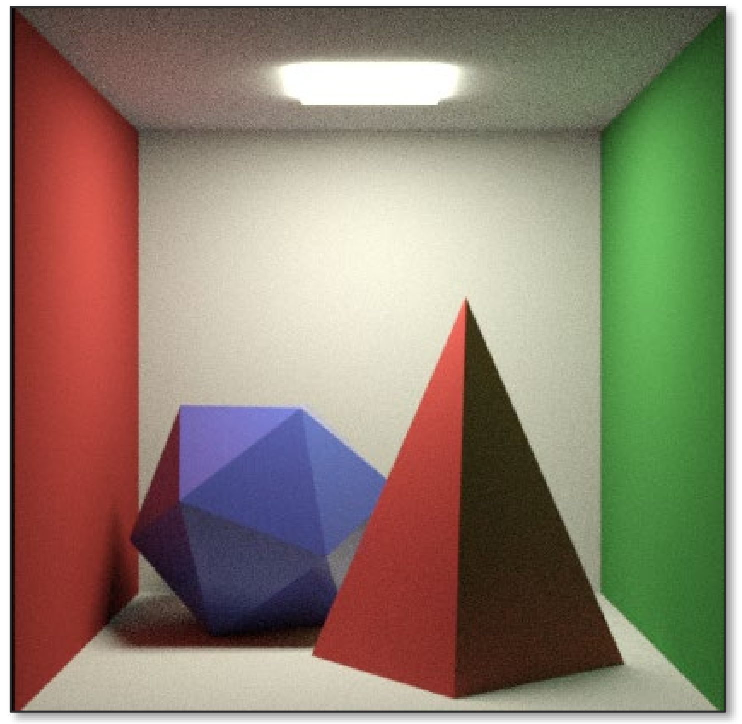
(Du 2020)

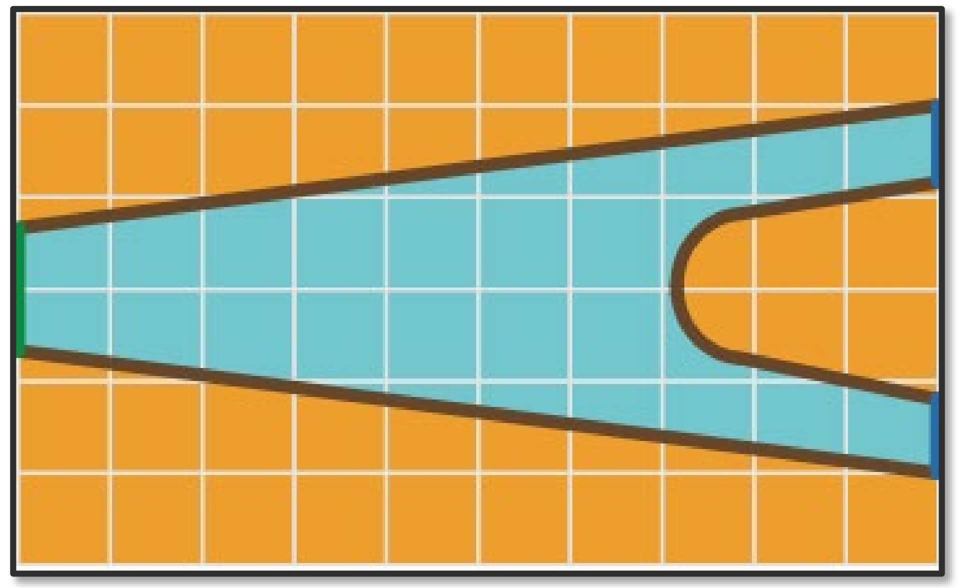


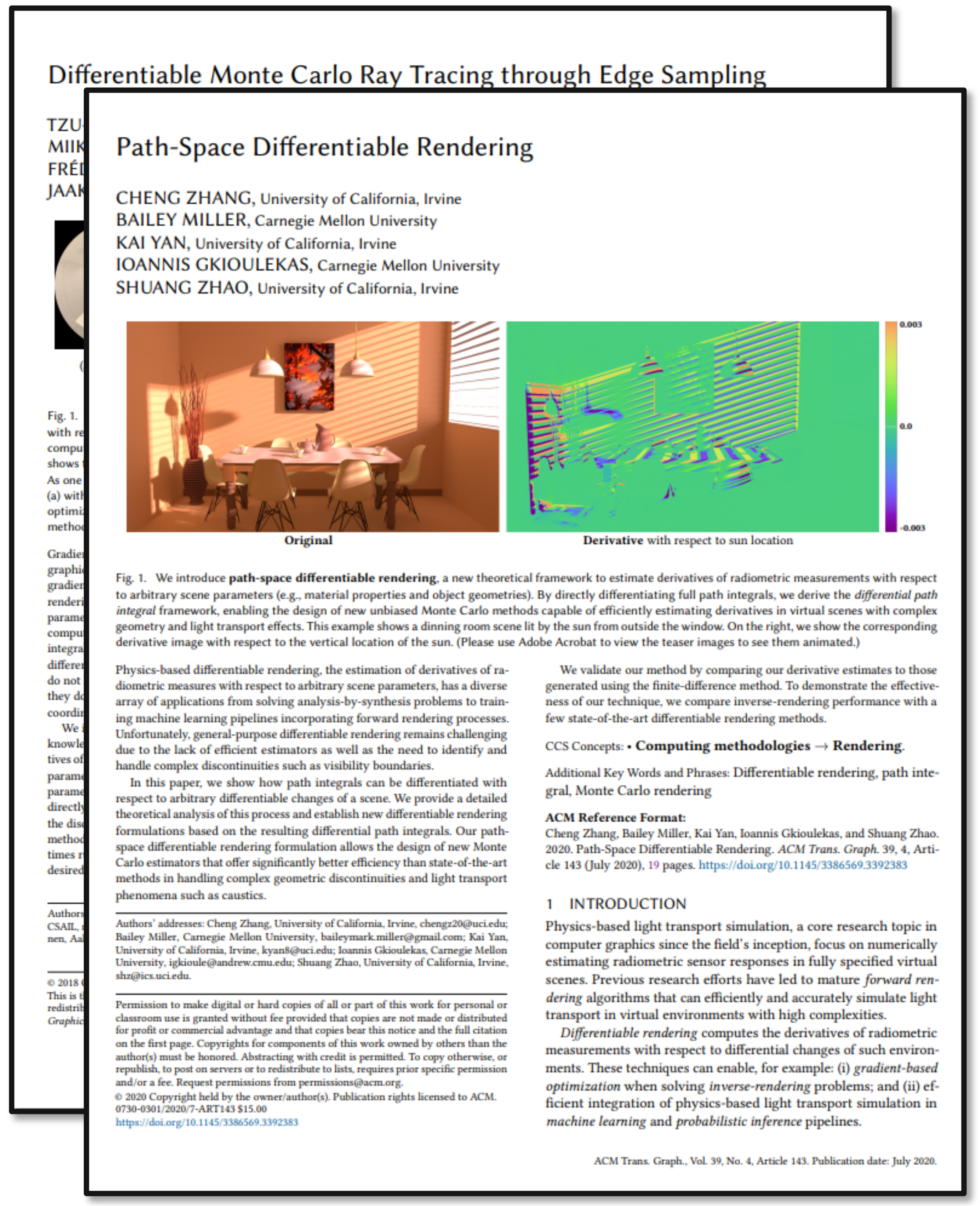
Many programs in graphics have this problem



Existing solutions to this specific problem



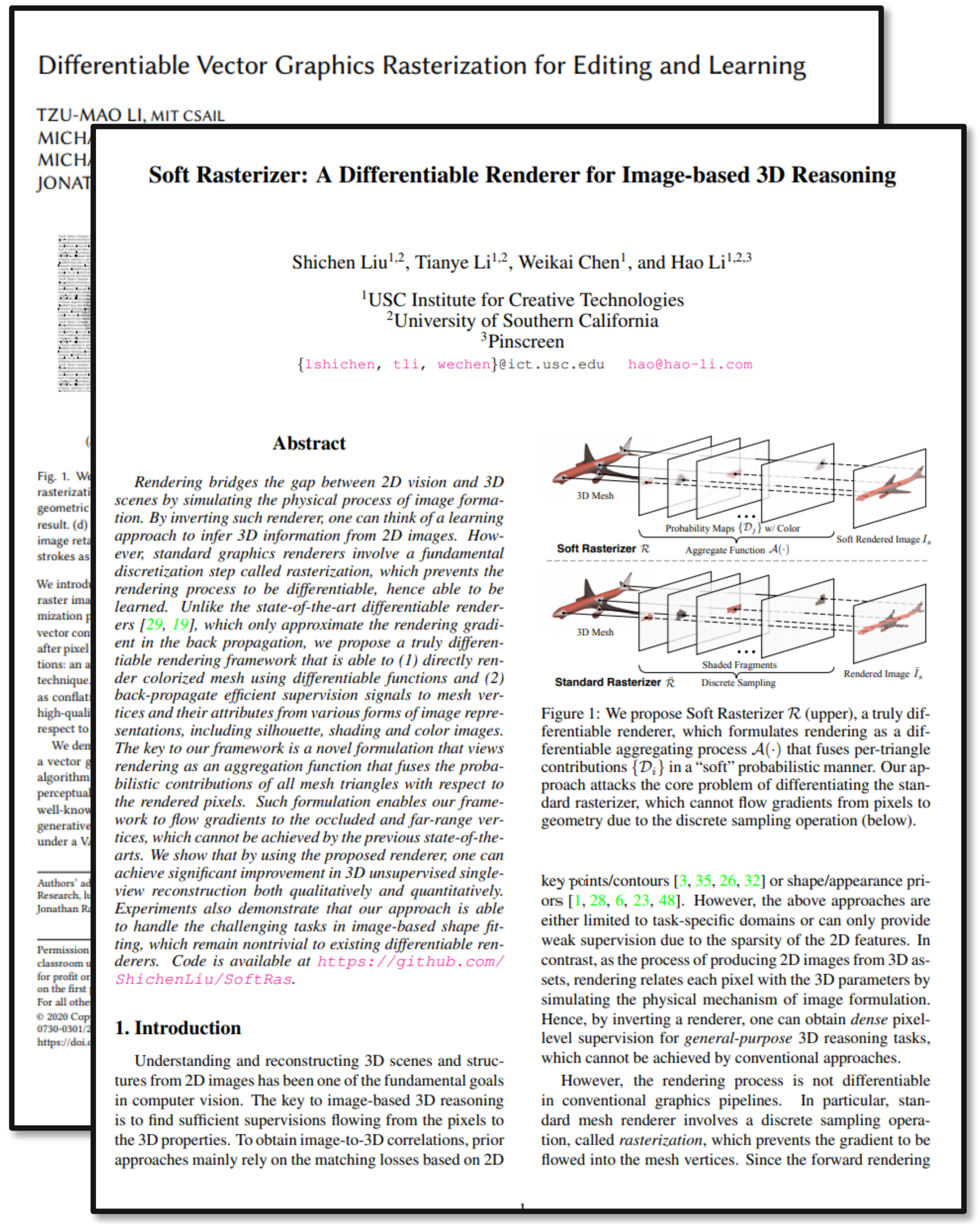




Differentiable Monte Carlo Ray Tracing through Edge Sampling

Path-Space Differentiable Rendering

CHENG ZHANG, University of California, Irvine
BAILEY MILLER, Carnegie Mellon University
KAI YAN, University of California, Irvine
IOANNIS GKIOULEKAS, Carnegie Mellon University
SHUANG ZHAO, University of California, Irvine



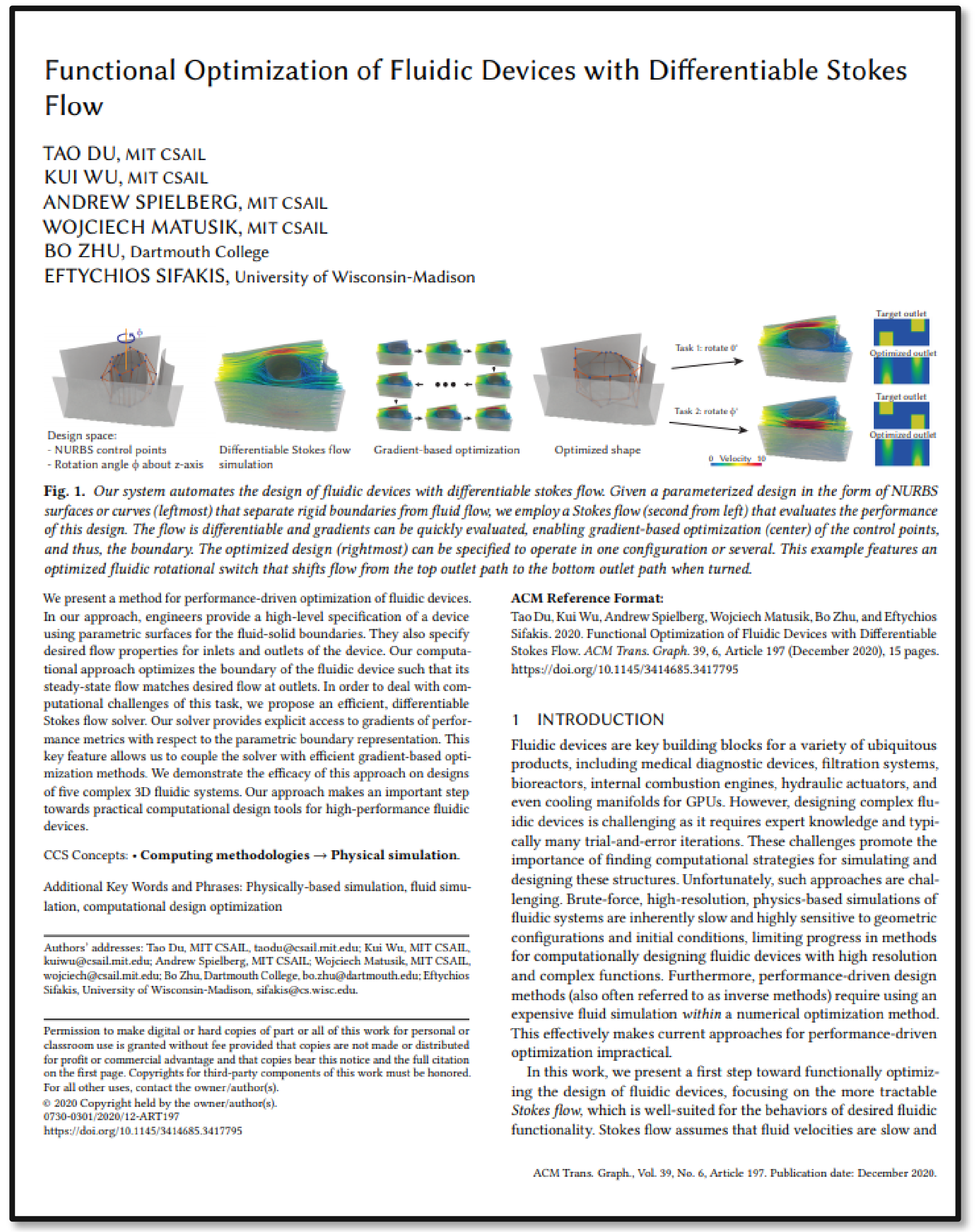
Differentiable Vector Graphics Rasterization for Editing and Learning

TZU-MAO LI, MIT CSAIL

Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning

Shichen Liu[1,2], Tianye Li[1,2], Weikai Chen[1], and Hao Li[1,2,3]

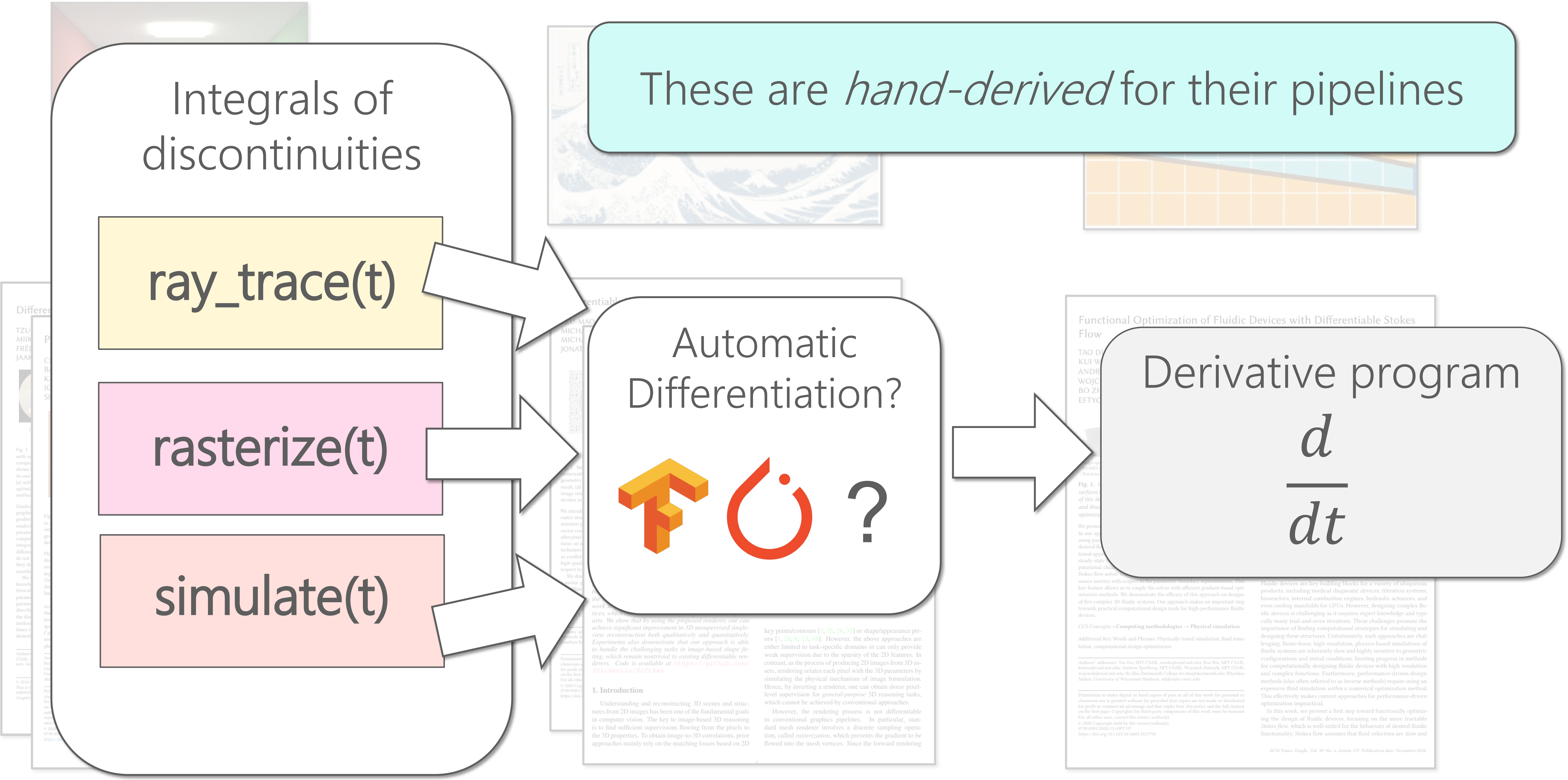
[1]USC Institute for Creative Technologies
[2]University of Southern California
[3]Pinscreen



Functional Optimization of Fluidic Devices with Differentiable Stokes Flow

TAO DU, MIT CSAIL
KUI WU, MIT CSAIL
ANDREW SPIELBERG, MIT CSAIL
WOJCIECH MATUSIK, MIT CSAIL
BO ZHU, Dartmouth College
EFTYCHIOS SIFAKIS, University of Wisconsin-Madison

Existing solutions to this specific problem



Systematically Differentiating Parametric Discontinuities

SAI PRAVEEN BANGARU*, MIT CSAIL
JESSE MICHEL*, MIT CSAIL
KEVIN MU, MIT CSAIL
GILBERT BERNSTEIN, UC Berkeley and MIT CSAIL
TZU-MAO LI, MIT CSAIL
JONATHAN RAGAN-KELLEY, MIT CSAIL

Fig. 1. We propose a language for the automatic differentiation of integrals with discontinuities. Existing auto-diff frameworks require integrals to be discretized into summations prior to differentiation, and therefore lose the derivative contribution from discontinuities. Our method produces a statistically consistent derivative program by introducing integration as a language primitive, which allows us to differentiate discontinuities in continuous space, before discretizing them into summations over discrete samples.

Emerging research in computer graphics, inverse problems, and machine learning requires us to differentiate and optimize parametric discontinuities. These discontinuities appear in object boundaries, occlusion, contact, and sudden change over time. In many domains, such as rendering and physics simulation, we differentiate the parameters of models that are expressed as integrals over discontinuous functions. Ignoring the discontinuities during differentiation often has a significant impact on the optimization process. Previous approaches either apply specialized hand-derived solutions, smooth out the discontinuities, or rely on incorrect automatic differentiation.

We propose a systematic approach to differentiating integrals with discontinuous integrands, by developing a new differentiable programming language. We introduce integration as a language primitive and account for the Dirac delta contribution from differentiating parametric discontinuities in the integrand. We formally define the language semantics and prove the correctness and closure under the differentiation, allowing the generation of gradients and higher-order derivatives. We also build a system, TEG, implementing these semantics. Our approach is widely applicable to a variety of tasks, including image stylization, fitting shader parameters, trajectory optimization, and optimizing physical designs.

CCS Concepts: • **Theory of computation** → Denotational semantics; • **Mathematics of computing** → **Differential calculus**; Stochastic control and optimization; *Probabilistic inference problems*; • **Computing methodologies** → **Computer graphics**; **Visibility**; **Animation**; Computer vision; **Modeling and simulation**.

Additional Key Words and Phrases: Automatic differentiation, differentiable programming, differentiable graphics, differentiable rendering, differentiable physics, domain-specific language.

ACM Reference Format:
Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically Differentiating Parametric Discontinuities. *ACM Trans. Graph.* 40, 4, Article 107 (August 2021), 17 pages. https://doi.org/10.1145/3450626.3459775

*Both authors contributed equally to this research.

Authors' addresses: Sai Praveen Bangaru, MIT CSAIL, Cambridge, MA, sbangaru@mit.edu; Jesse Michel, MIT CSAIL, Cambridge, MA, jmmichel@mit.edu; Kevin Mu, MIT CSAIL, Cambridge, MA, kmu@csail.mit.edu; Gilbert Bernstein, UC Berkeley, Berkeley, CA, MIT CSAIL, Cambridge, MA, gilbo@berkeley.edu; Tzu-Mao Li, MIT CSAIL, Cambridge, MA, tzumao@mit.edu; Jonathan Ragan-Kelley, MIT CSAIL, Cambridge, MA, jrk@csail.mit.edu.

0730-0301/2021/8-ART107
https://doi.org/10.1145/3450626.3459775

ACM Trans. Graph., Vol. 40, No. 4, Article 107. Publication date: August 2021.

Systematically Differentiating Parametric Discontinuities

Sai **Bangaru***, Jesse **Michel***, Kevin **Mu**, Gilbert **Bernstein**, Tzu-Mao **Li**, Jonathan **Ragan-Kelley**
(MIT CSAIL & UC Berkeley)

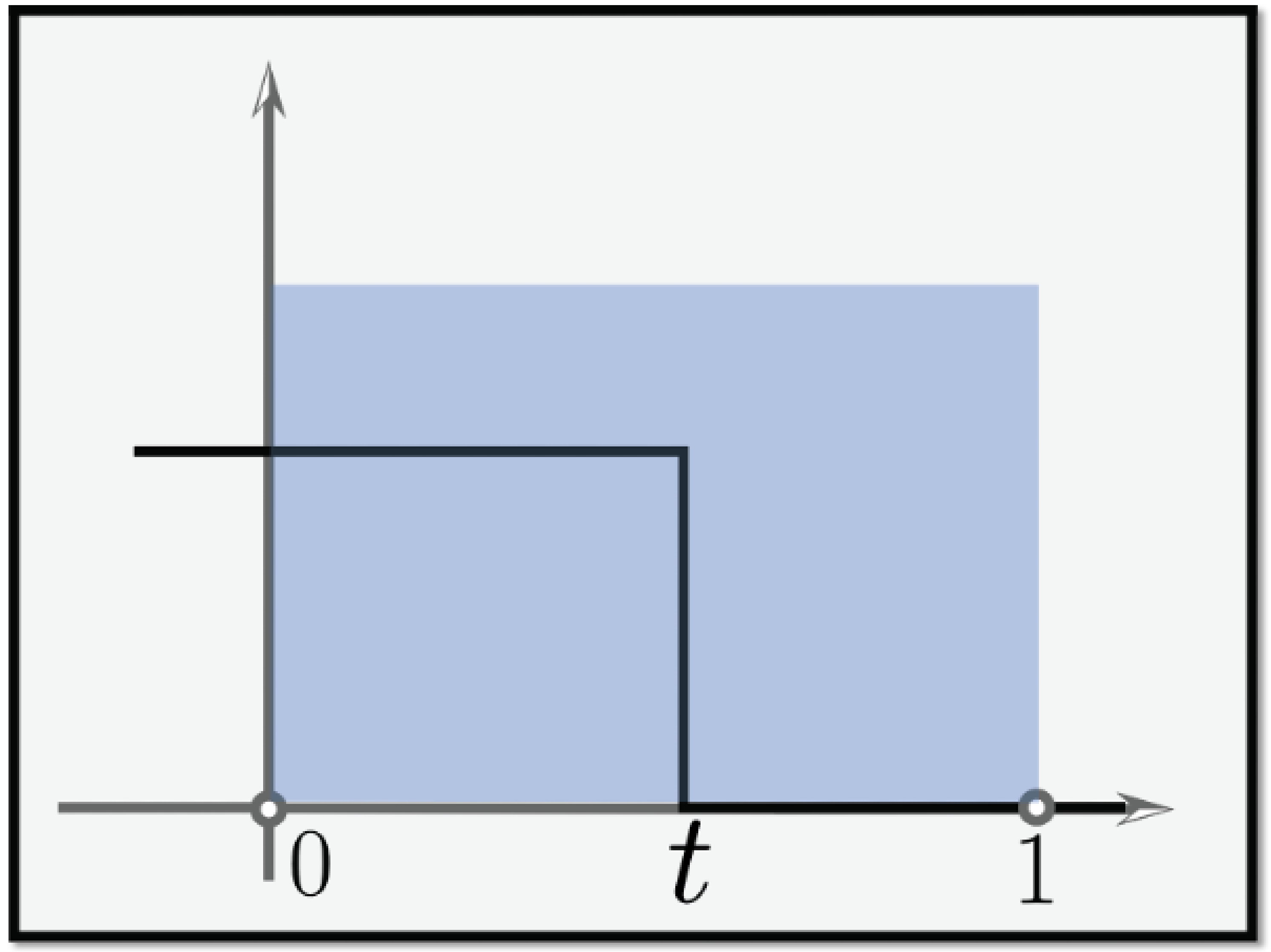
SIGGRAPH 2021 (to appear)

A simple demonstration

$$\frac{\mathtt{d}}{\mathtt{dt}}\int_0^1 [\mathtt{x} < \mathtt{t}]\,\mathtt{dx}$$

Derivative of the analytical integral

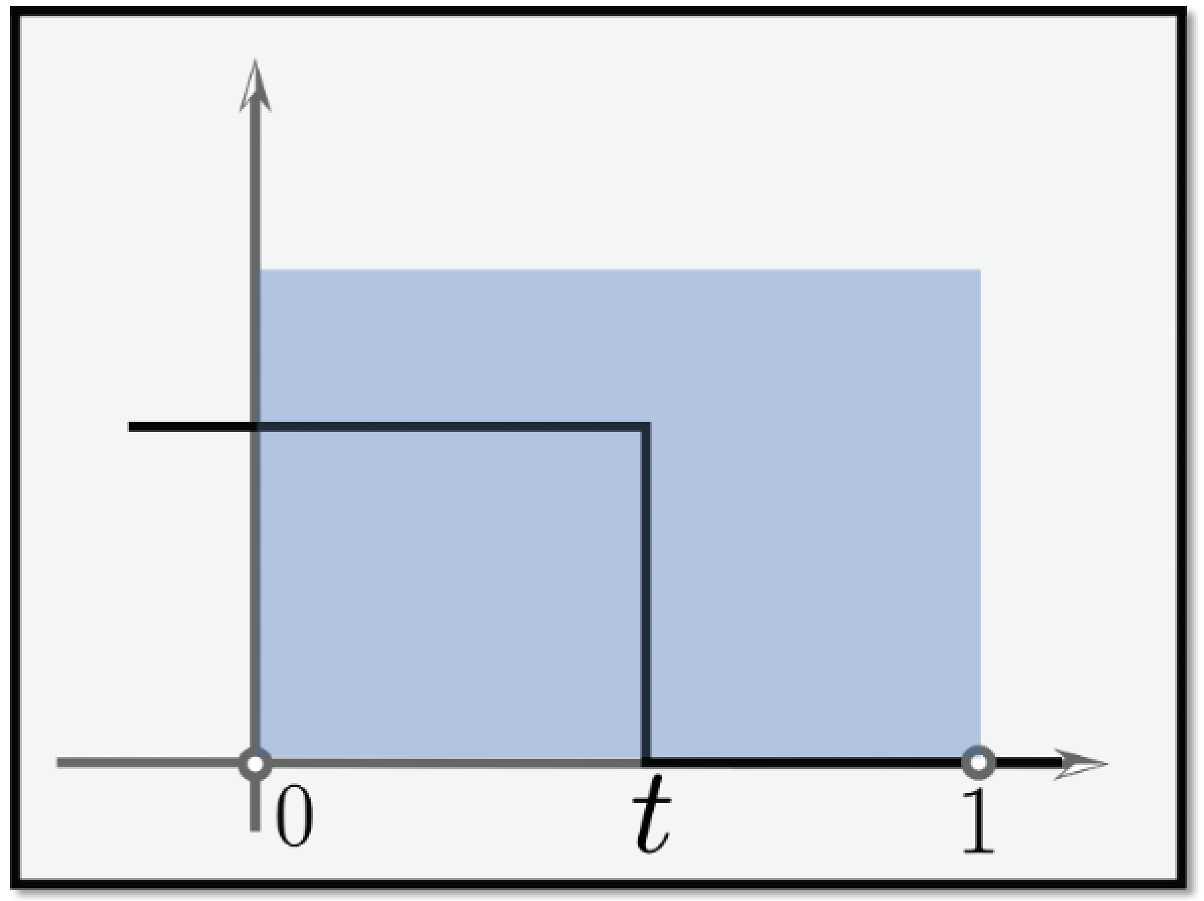
$$\int_0^1 [x < t]dx = \begin{cases} 0 & \text{if } t \leq 0 \\ t & \text{if } 0 < t < 1 \\ 1 & \text{if } 1 \geq t \end{cases}$$



$$\frac{d}{dt}\int_0^1 [x < t]dx = \begin{cases} 0 & \text{if } t \leq 0 \\ 1 & \text{if } [0 < t < 1] \\ 0 & \text{if } 1 \geq t \end{cases}$$

Naïve autodiff of integrals with derivatives

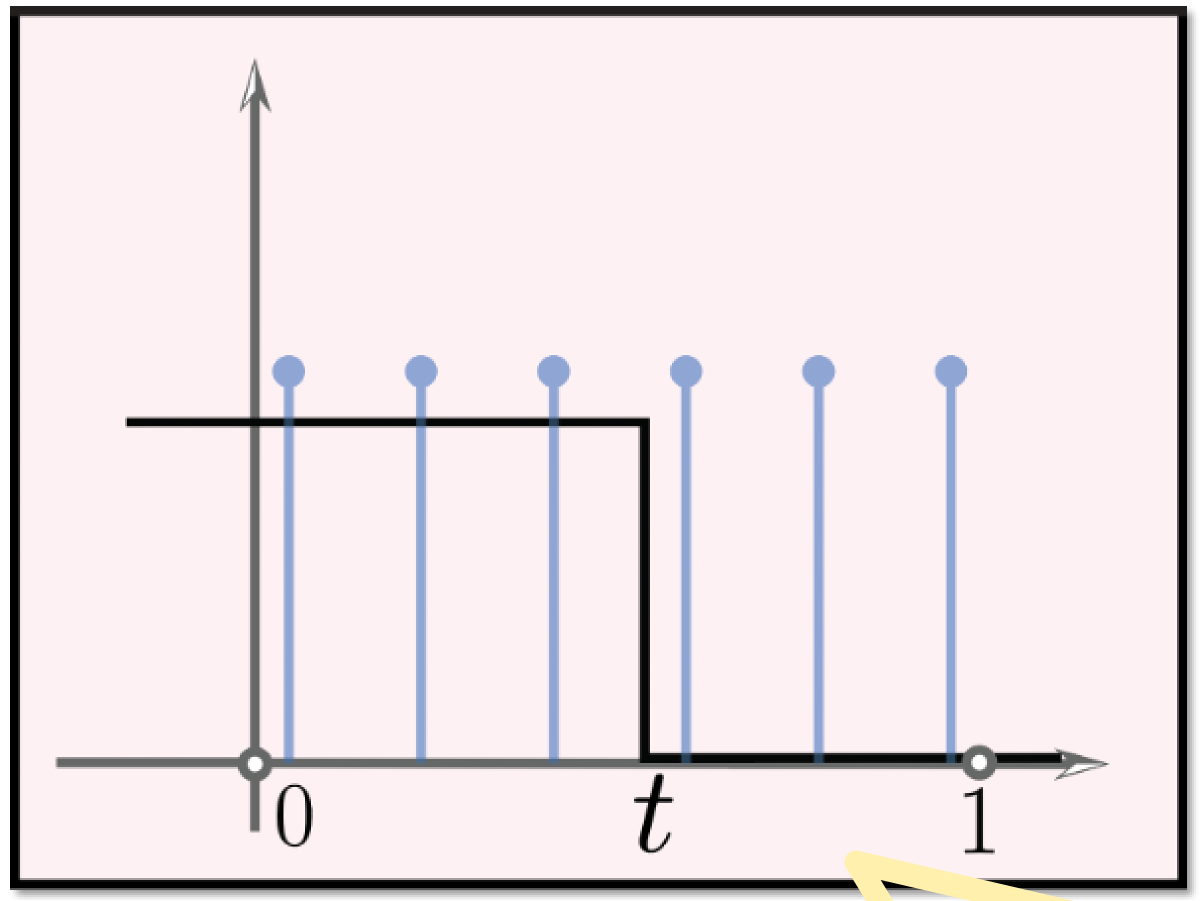
$$\int_0^1 [x < t]\,dx$$



$\int \to \sum$

DISCRETIZE

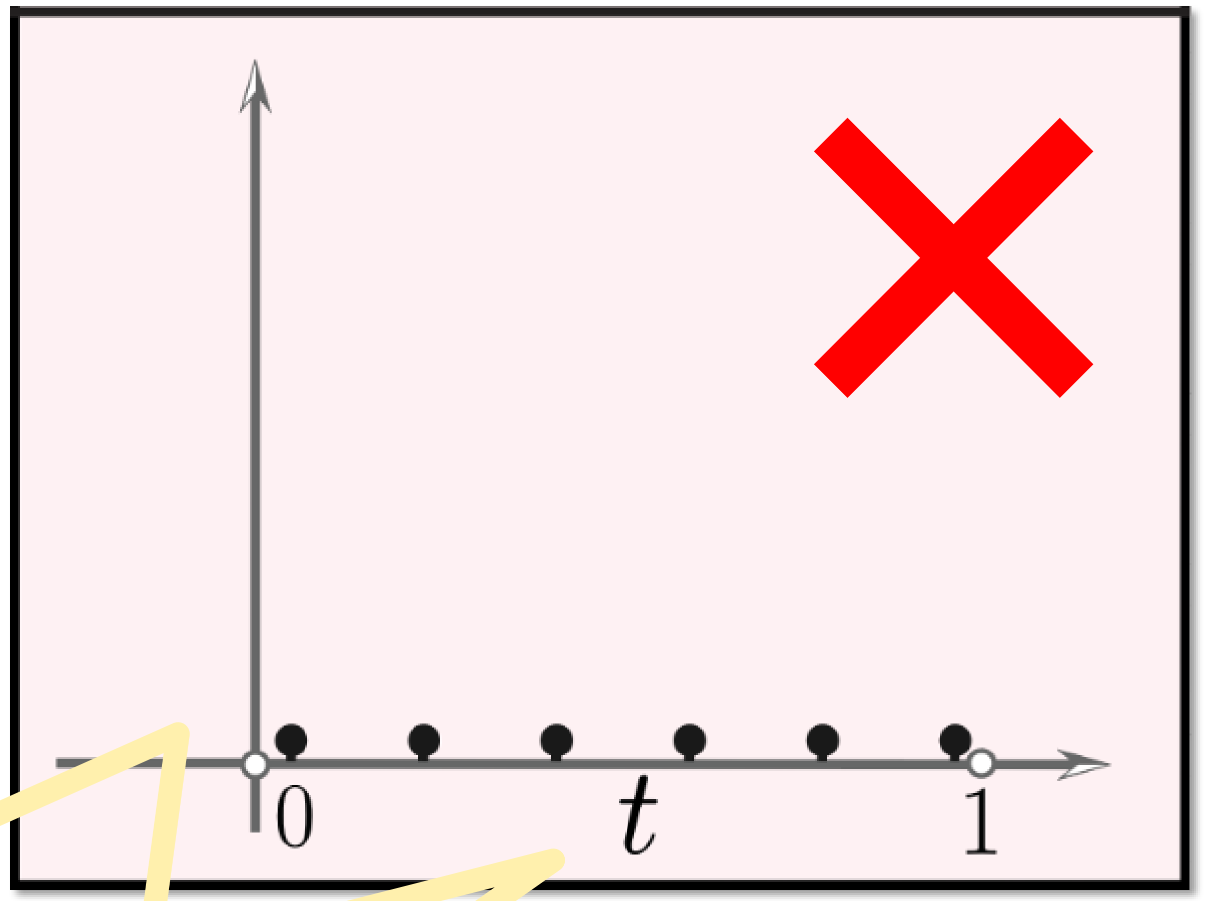
$$\sum_{i=0}^{N} \frac{1}{N}\left[\frac{i}{N} < t\right]$$



$\frac{d}{dt}$

DIFFERENTIATE

$$\sum_{i=0}^{N} 0$$

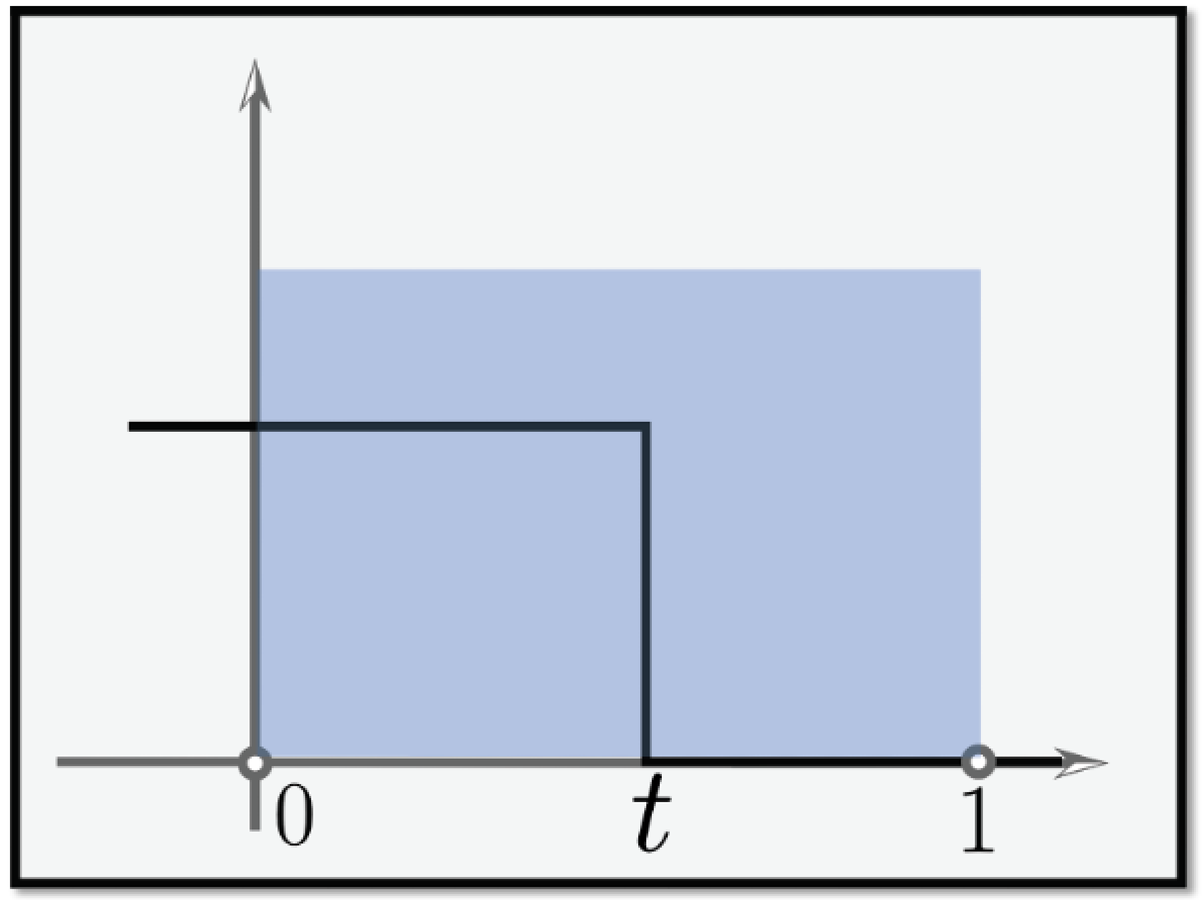


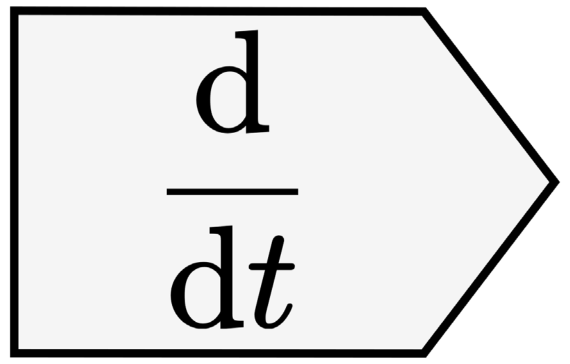
$$[0 < t < 1]$$

Correct derivatives of integrals with discontinuities

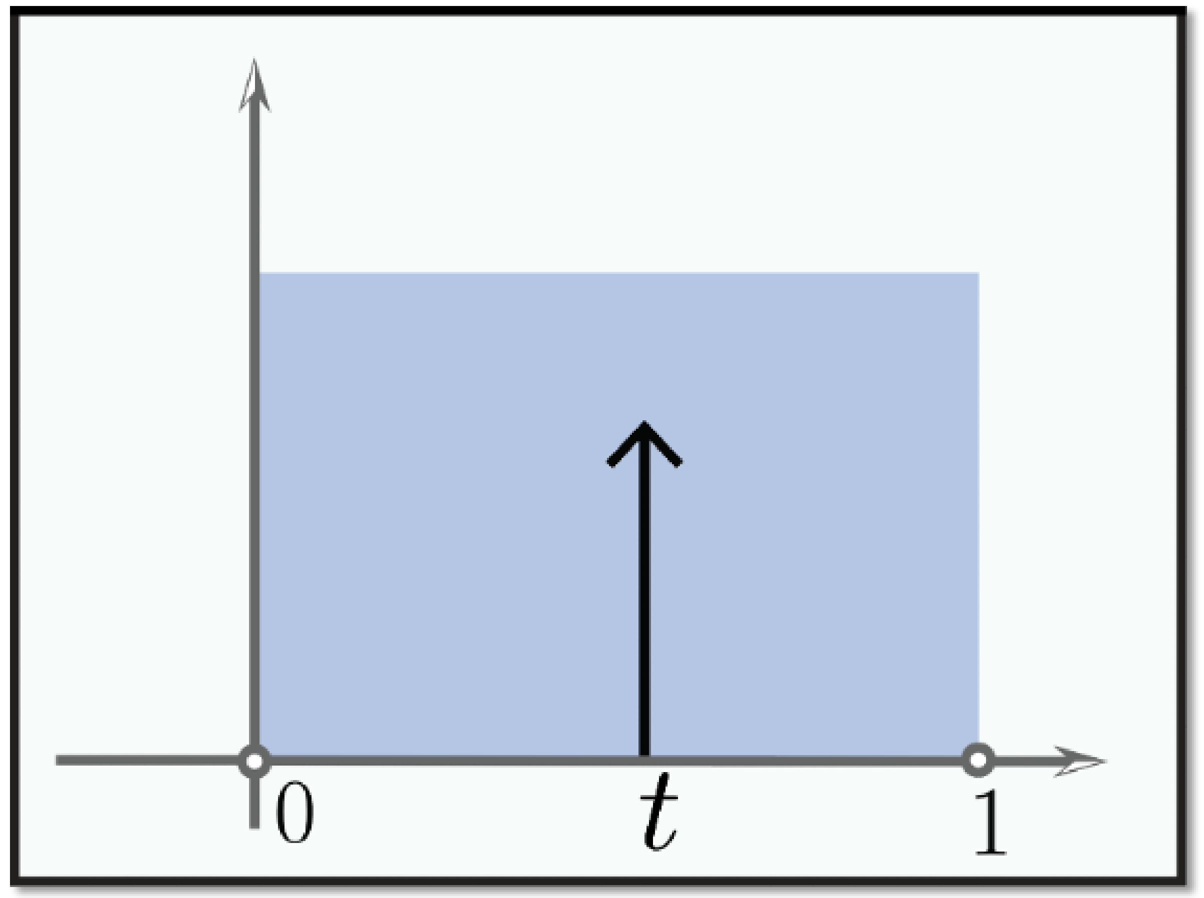
$$\int_0^1 [x < t]\,dx$$

$$\int_0^1 \delta(t - x)\,dx$$

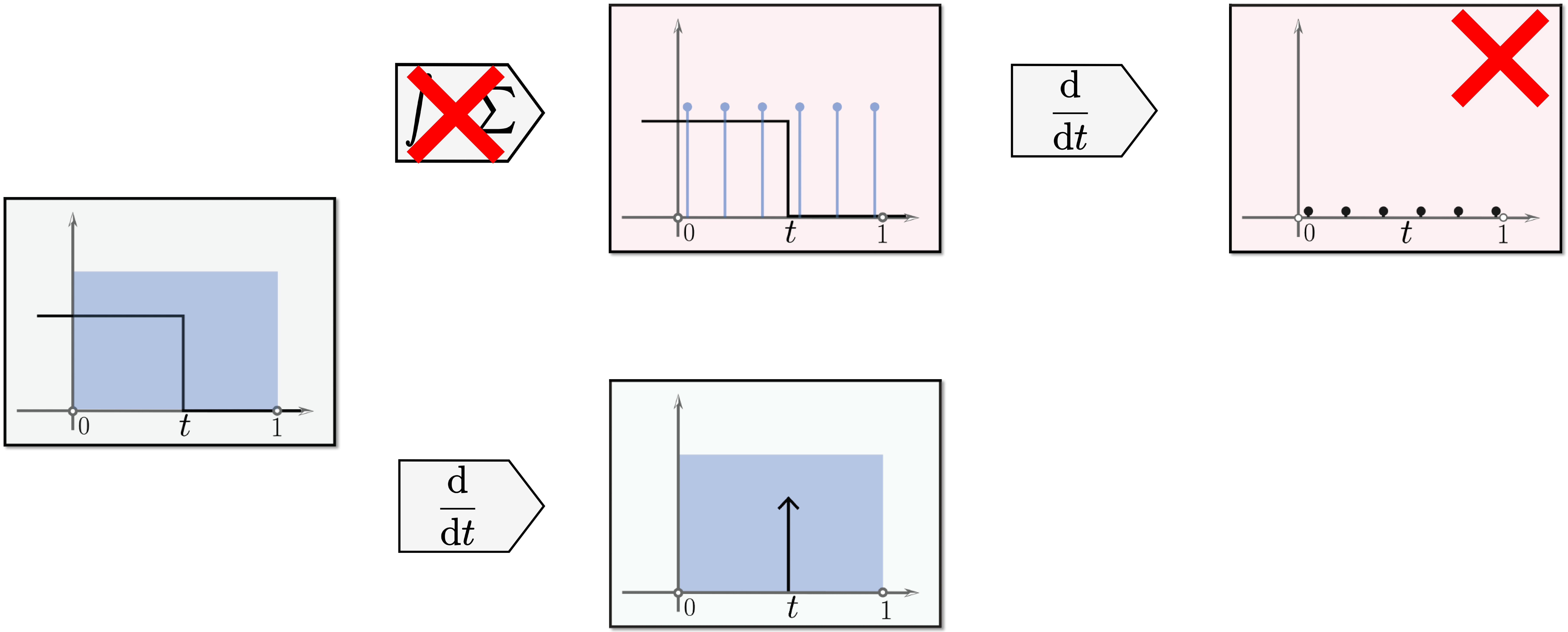




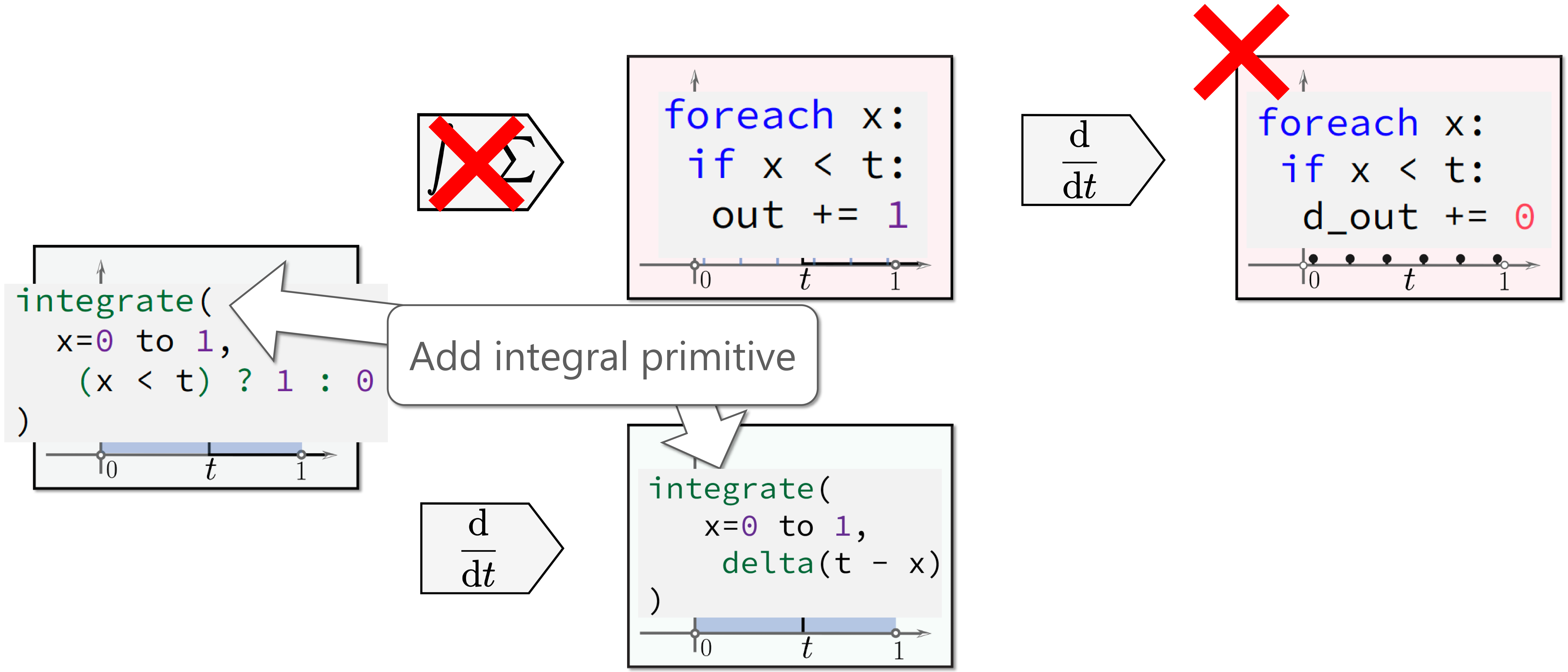
DIFFERENTIATE



Integrals with discontinuities break auto-diff



The need for an integral primitive



Discontinuities now need a delta operation

$\cancel{\int \Sigma}$

```
foreach x:
 if x < t:
  d_out += 0
```

Not a well-defined operation

```
integrate(
  x=0 to 1,
   (x < t) ? 1 : 0
)
```

Add a delta operator

$\frac{d}{dt}$

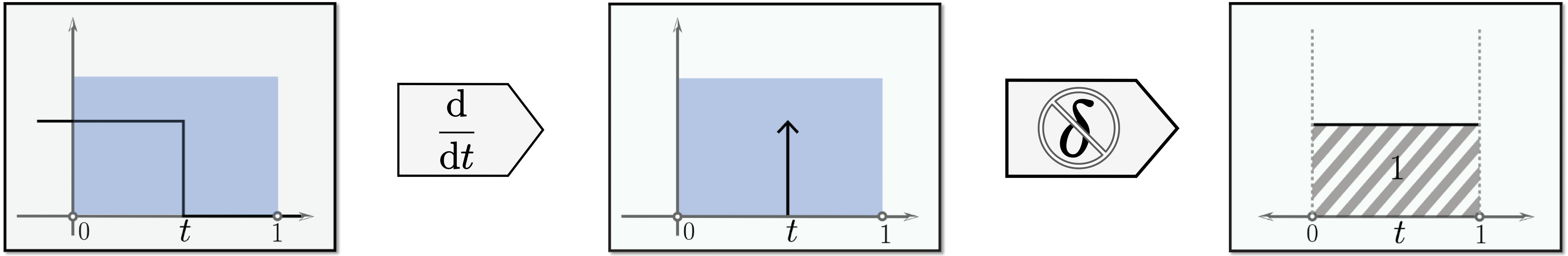
```
integrate(
  x=0 to 1,
   delta(t-x)
)
```

Recap: Differentiate first, then discretize

$$\int_0^1 [x < t] \, dx$$

$$\int_0^1 \delta(t - x) \, dx$$

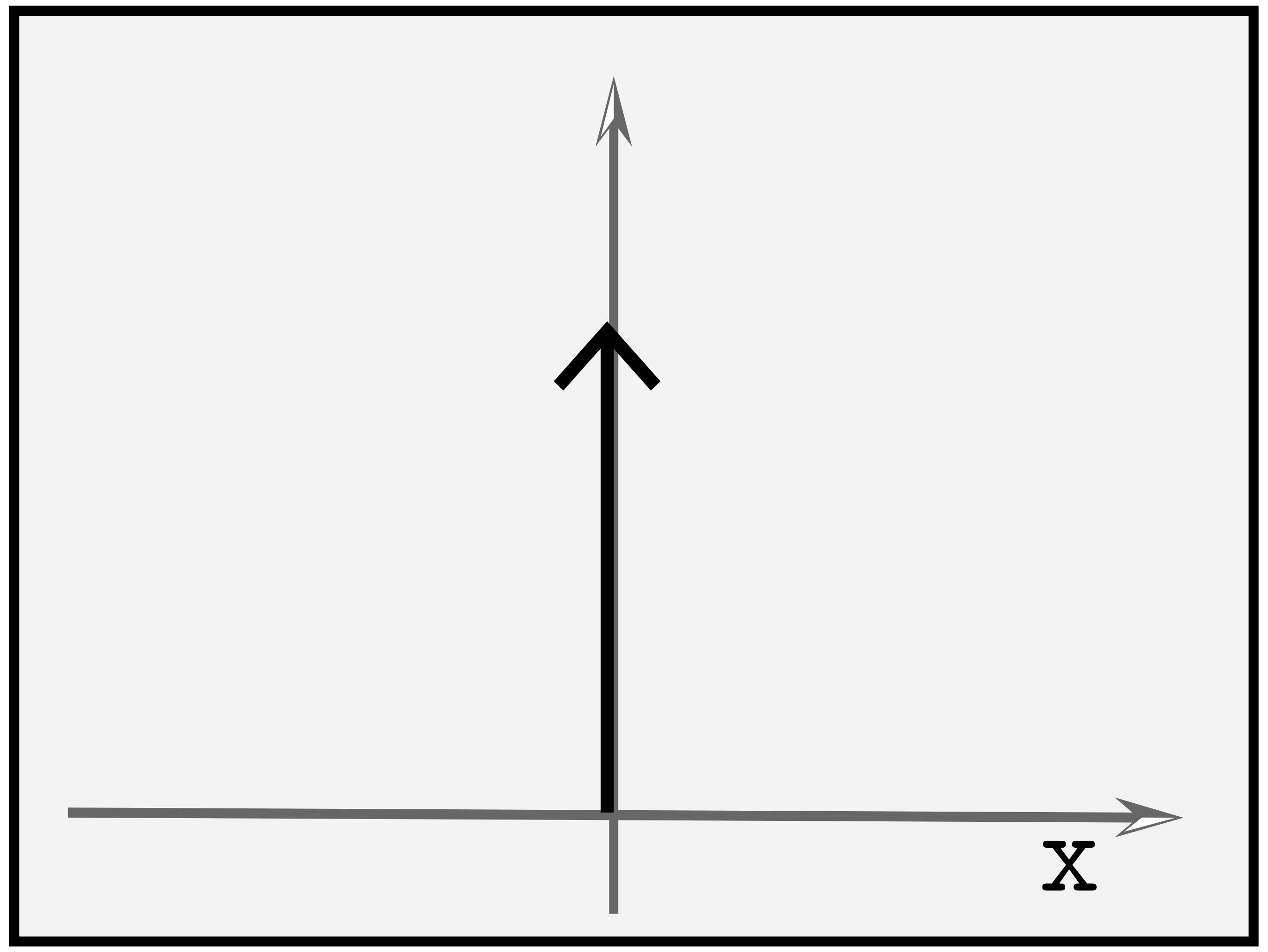
$$[0 < t < 1]$$



Eliminating deltas with the Sifting property

$$\int_{x=a}^{b} \delta(x) f(x) = [a < 0 < b] \cdot f(0)$$

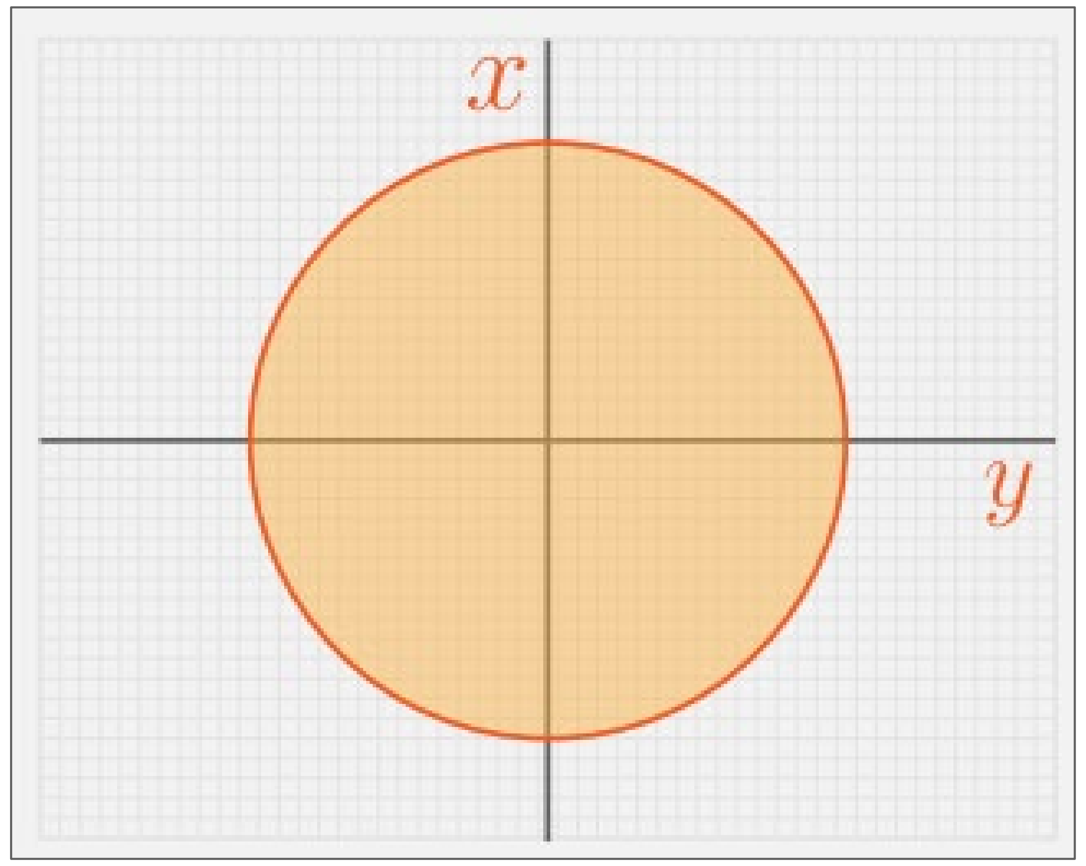
Single variable of integration



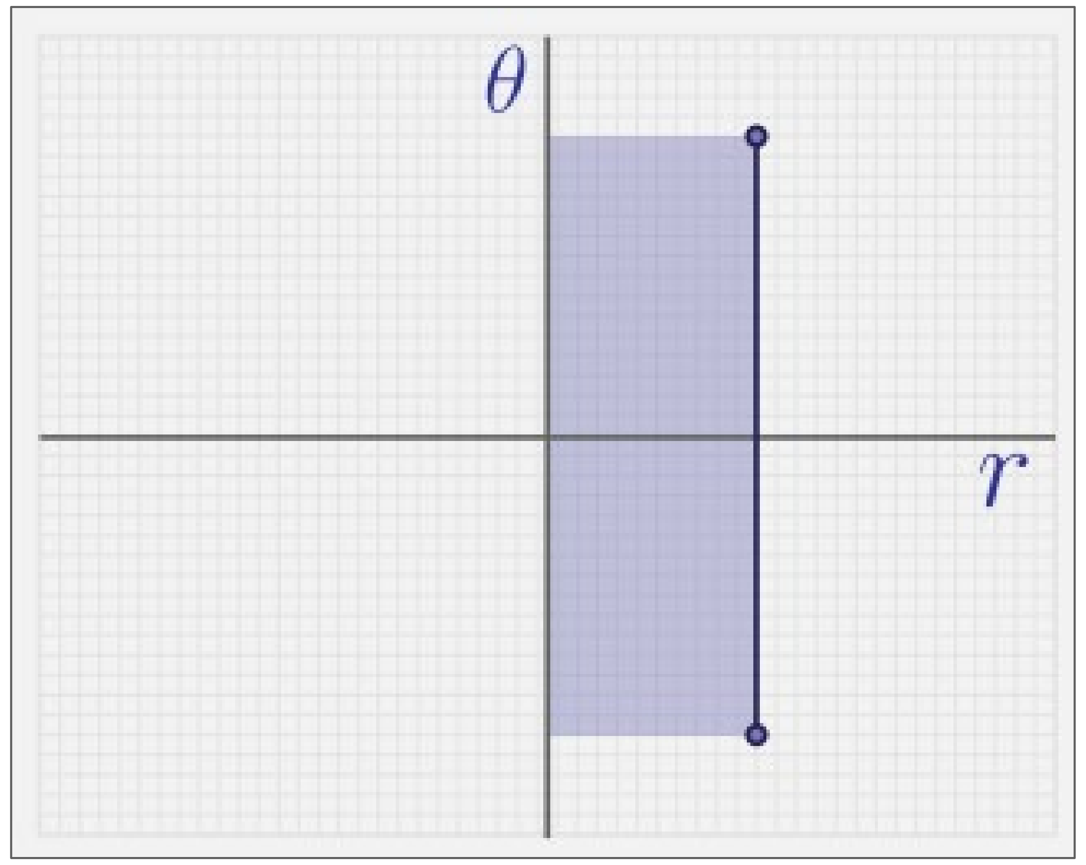
Pass 2: Delta Normalization

$$\int_{-1}^{1}\int_{-1}^{1}\delta\left(2x^2+2y^2-t\right)dydx$$

Not in normal form



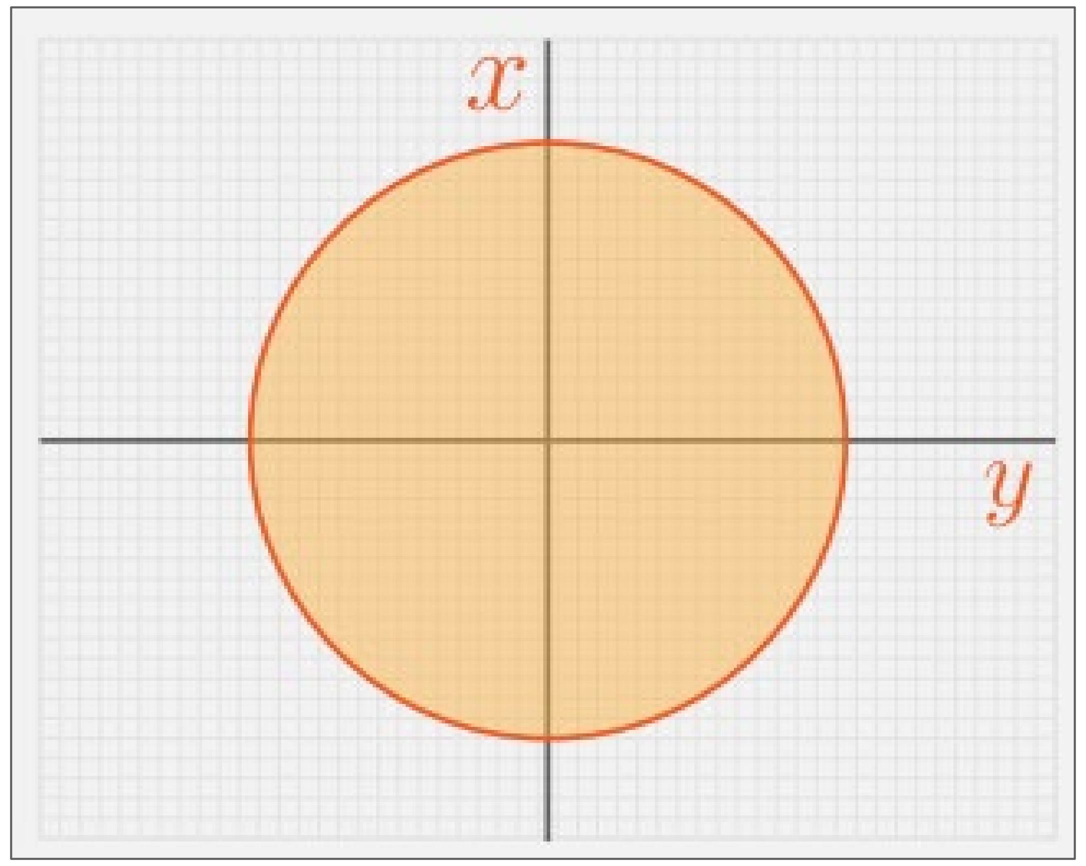
$(x, y) \leftrightarrow (r, \theta)$



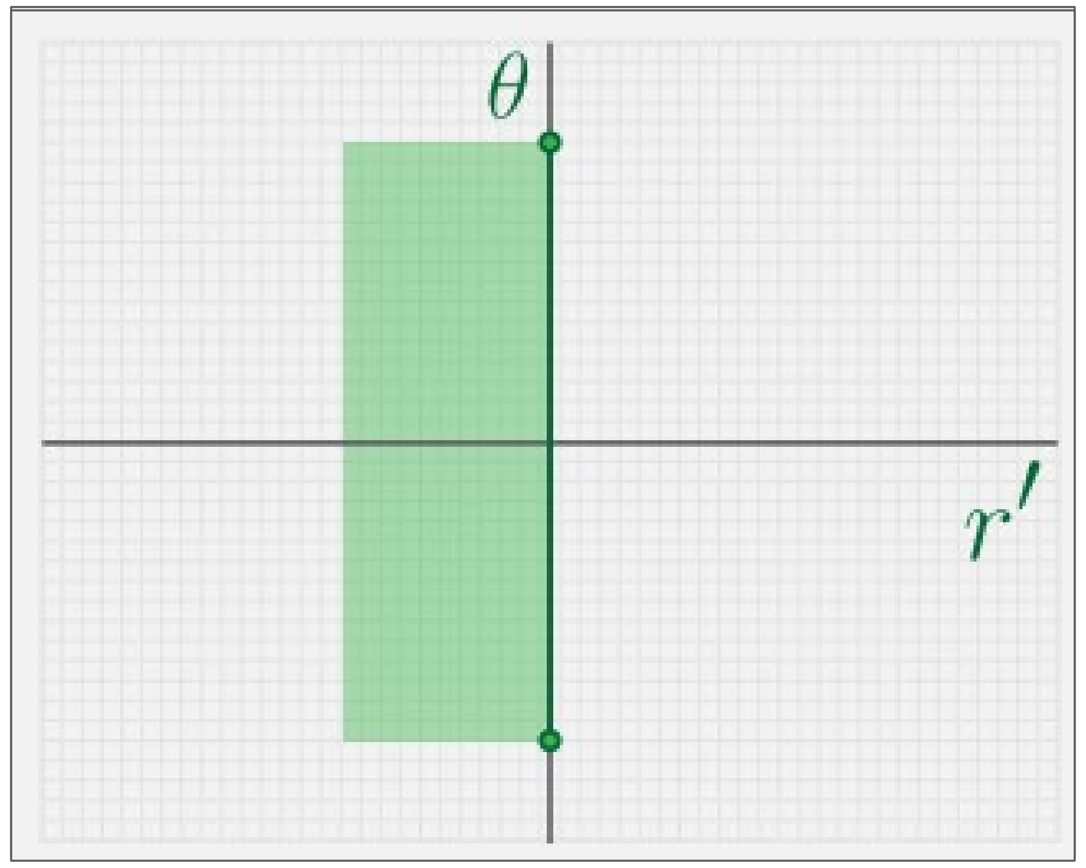
Pass 2: Delta Normalization

$$\int_{-1}^{1}\int_{-1}^{1}\delta\left(2r - t\right)\,\mathrm{d}y\,\mathrm{d}x$$

Still not in normal form



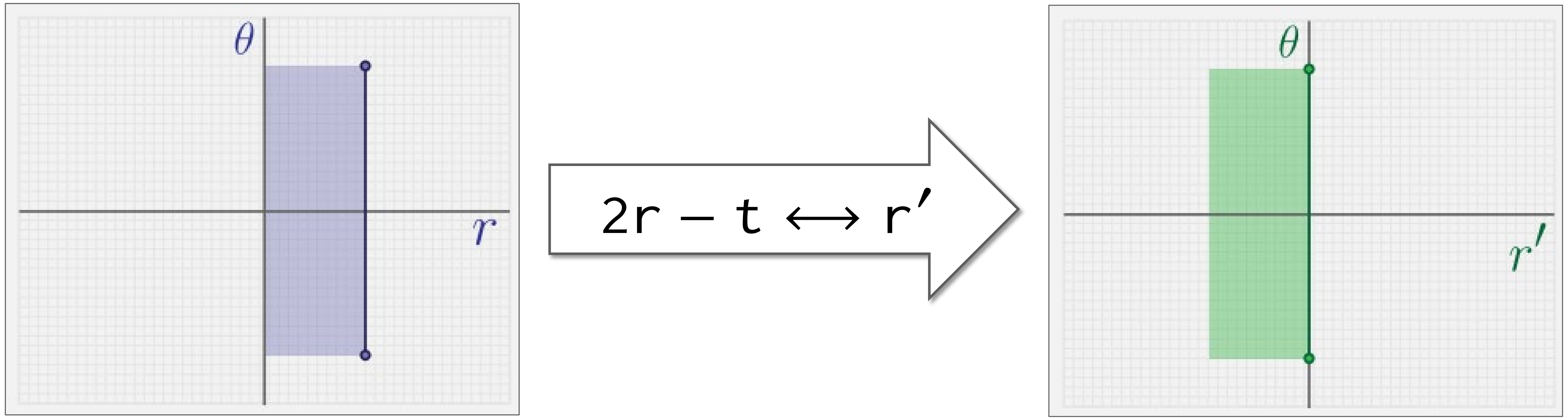
$$2r - t \longleftrightarrow r'$$



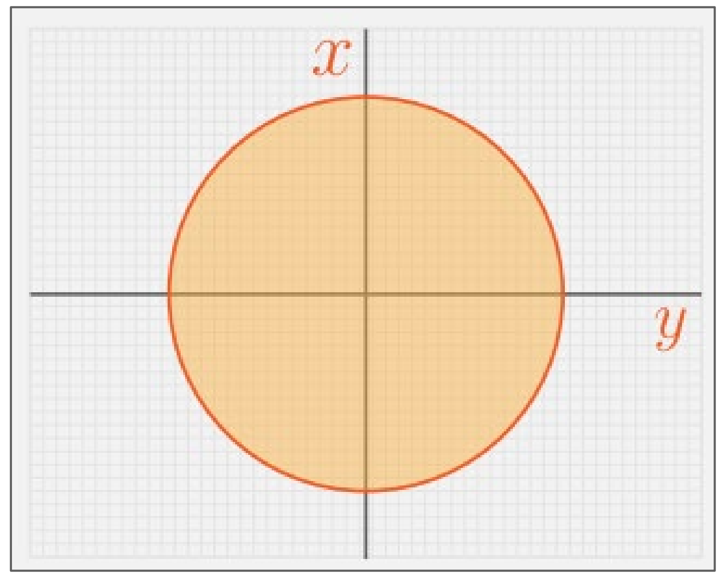
Pass 2: Delta Normalization

$$\int_{-1}^{1}\int_{-1}^{1}\delta\left(r'\right)\,dydx$$

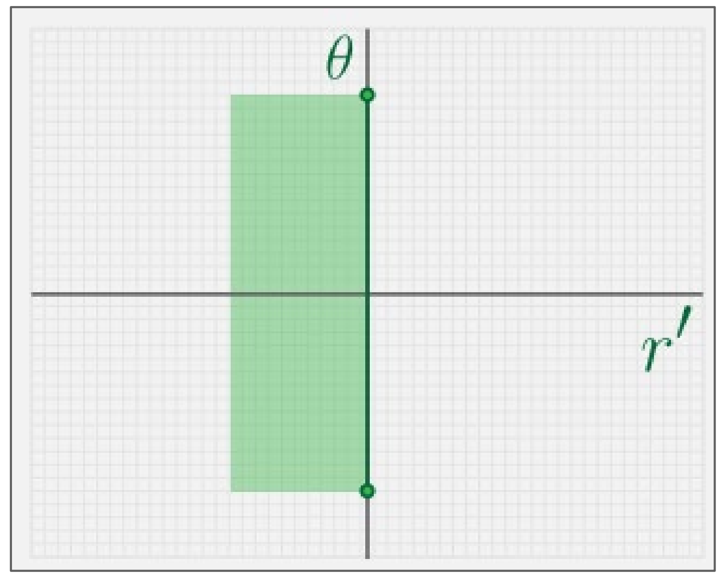
Final coordinates are in normal form!



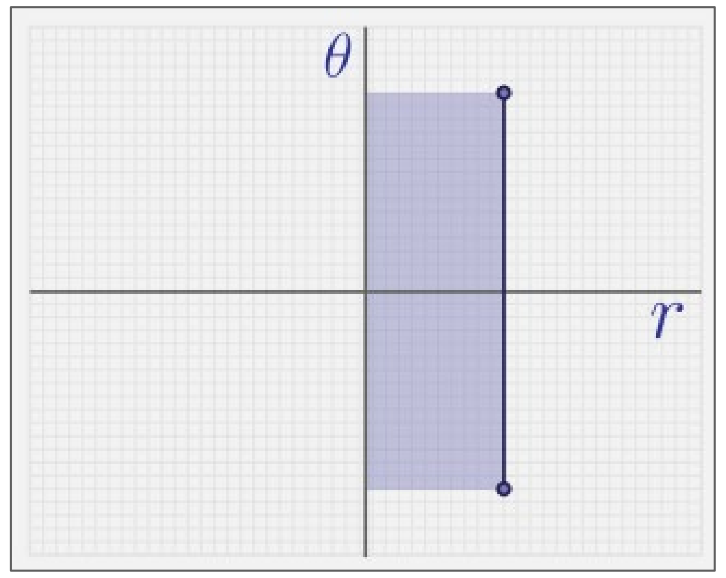
Pass 2: Delta Normalization



$$\int_{-1}^{1}\int_{-1}^{1}\delta\left(2x^2+2y^2-t\right)dydx$$



$$\int_{0}^{2\pi}\int_{0}^{1}\delta\left(2r-t\right)rdrd\theta$$



$$\int_{0}^{2\pi}\int_{-t}^{2-t}\delta\left(r'\right)(r'+t)/2\,(dr'/2)d\theta$$

Pass 2: Delta Normalization

$$\int_0^{2\pi} \int_0^1 \delta(2r - t)\, r\, dr\, d\theta$$

$$2r - t \longrightarrow r'$$

$$\int_0^{2\pi} \int_{-t}^{2-t} \delta(r')\,(r' + t)/2\,(dr'/2)\,d\theta$$

Pass 2: Delta Normalization

$$\int_0^{2\pi}\int_0^1 \delta(2r - t)\, r\, dr\, d\theta$$

$$r \longrightarrow (r' + t)/2$$

Inverse

$$\int_0^{2\pi}\int_{-t}^{2-t} \delta(r')\,(r' + t)/2\,(dr'/2)\,d\theta$$

Pass 2: Delta Normalization

$$\int_0^{2\pi}\int_0^1 \delta(2r - t)\,r\,dr\,d\theta$$

$$dr \longrightarrow dr'/2$$

Derivative

$$\int_0^{2\pi}\int_{-t}^{2-t} \delta(r')\,(r' + t)/2\,(dr'/2)\,d\theta$$

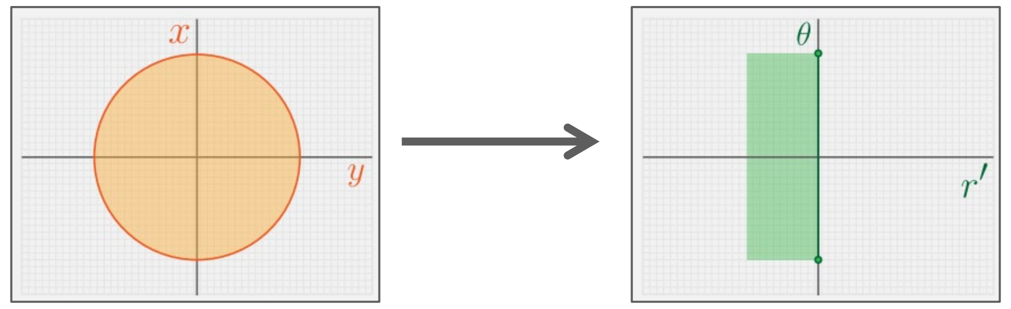
All passes together

$$\int \cdots + e * \left(\delta(\phi_1) + \delta(\phi_2)\cdots\right)$$

1. Normalize expression

$$\int e * \delta(\phi_1) + \int e * \delta(\phi_2) + \cdots$$

2. Change-of-coordinates



$$\int_x \delta(x) * \cdots$$

3. Annihilate (Sifting)

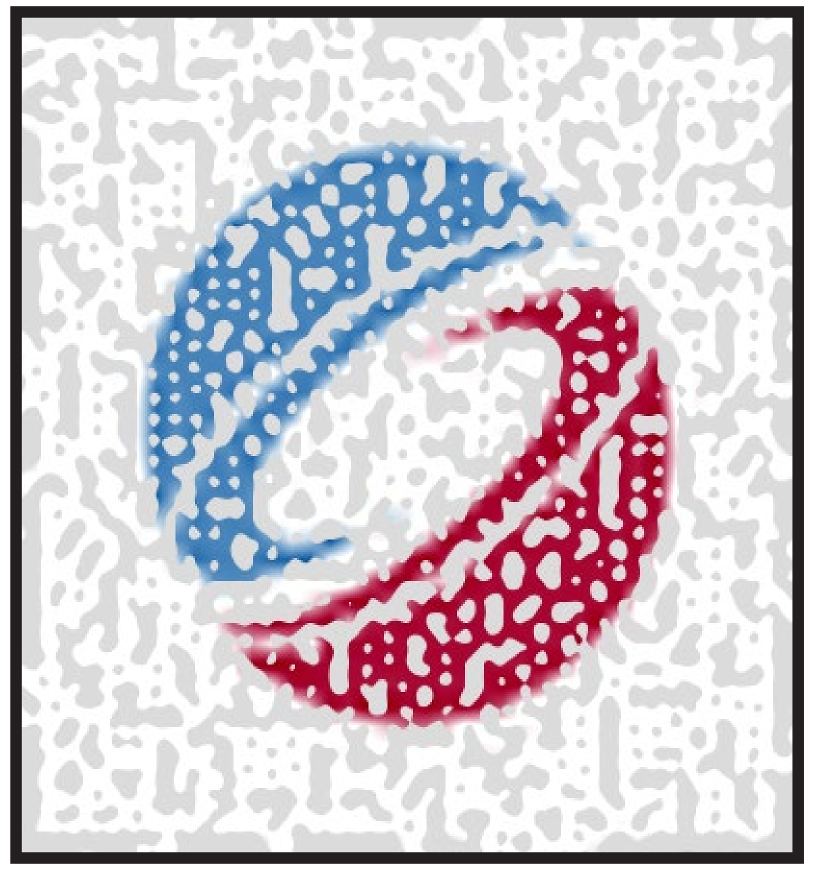
$$\cdots [a < 0 < b] \cdots$$

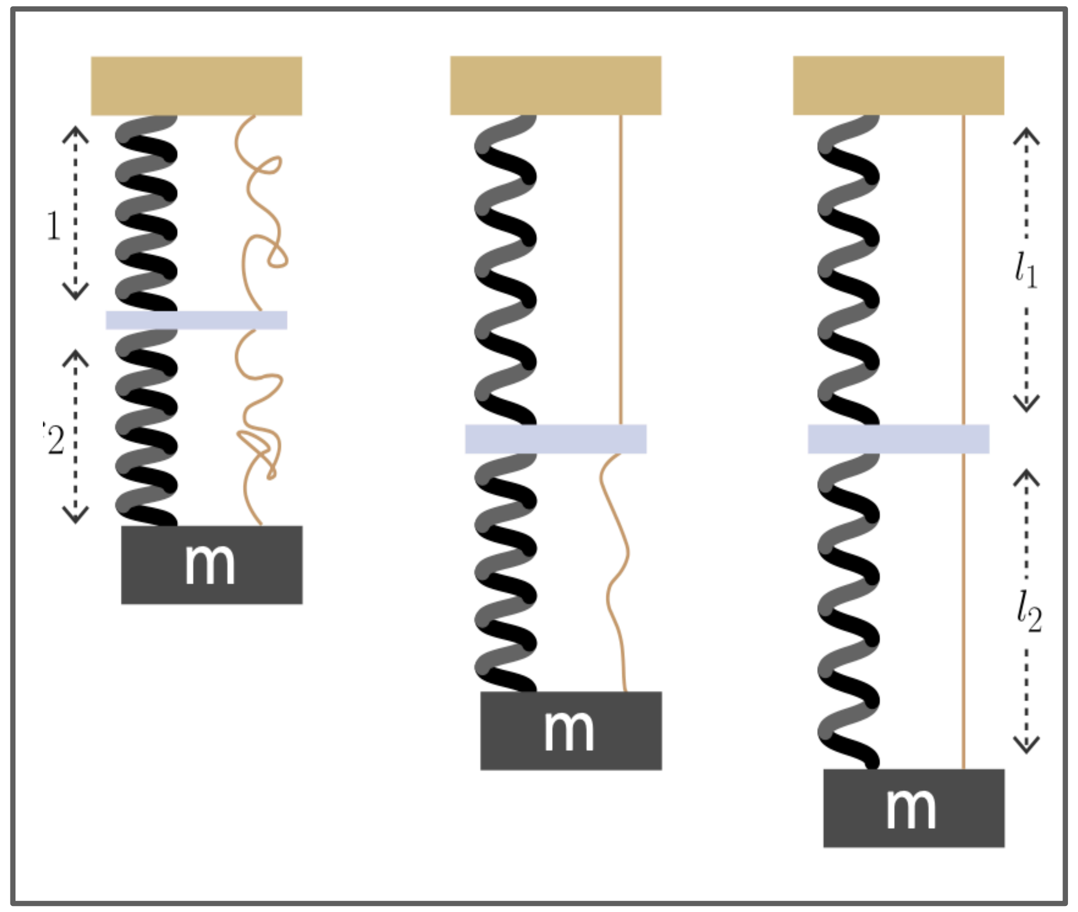
Delta-free!

$\cancel{\delta}$

Systems and Applications

Applications

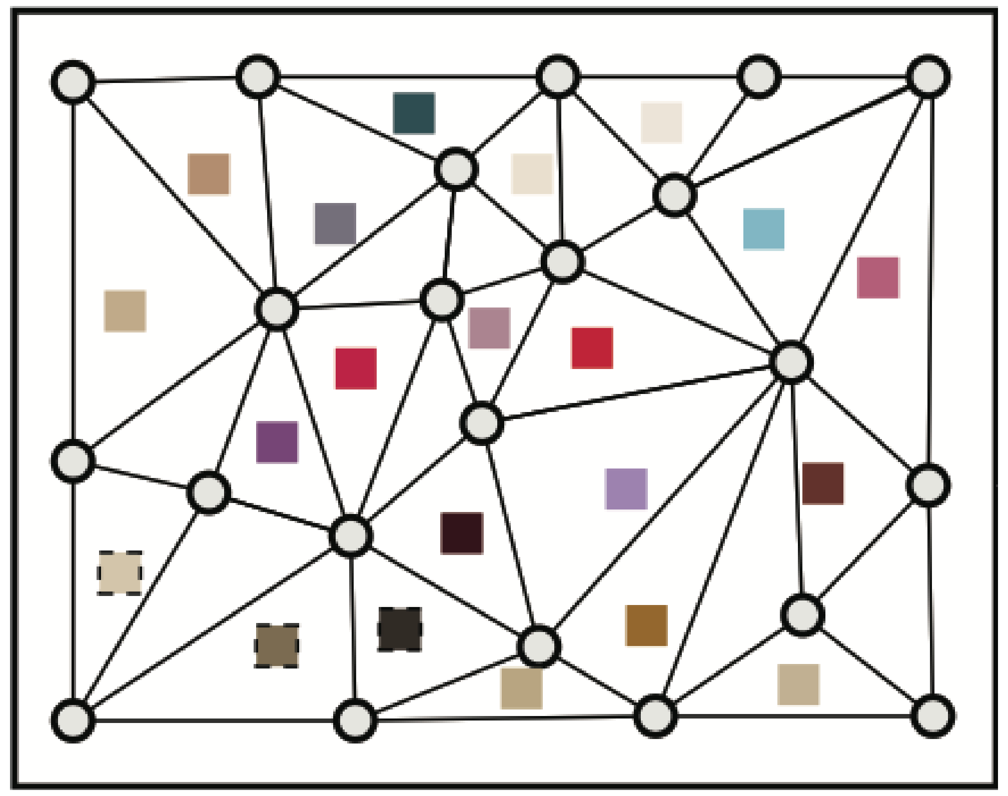




Application: Image Style Filters

Target image

rasterize(t)

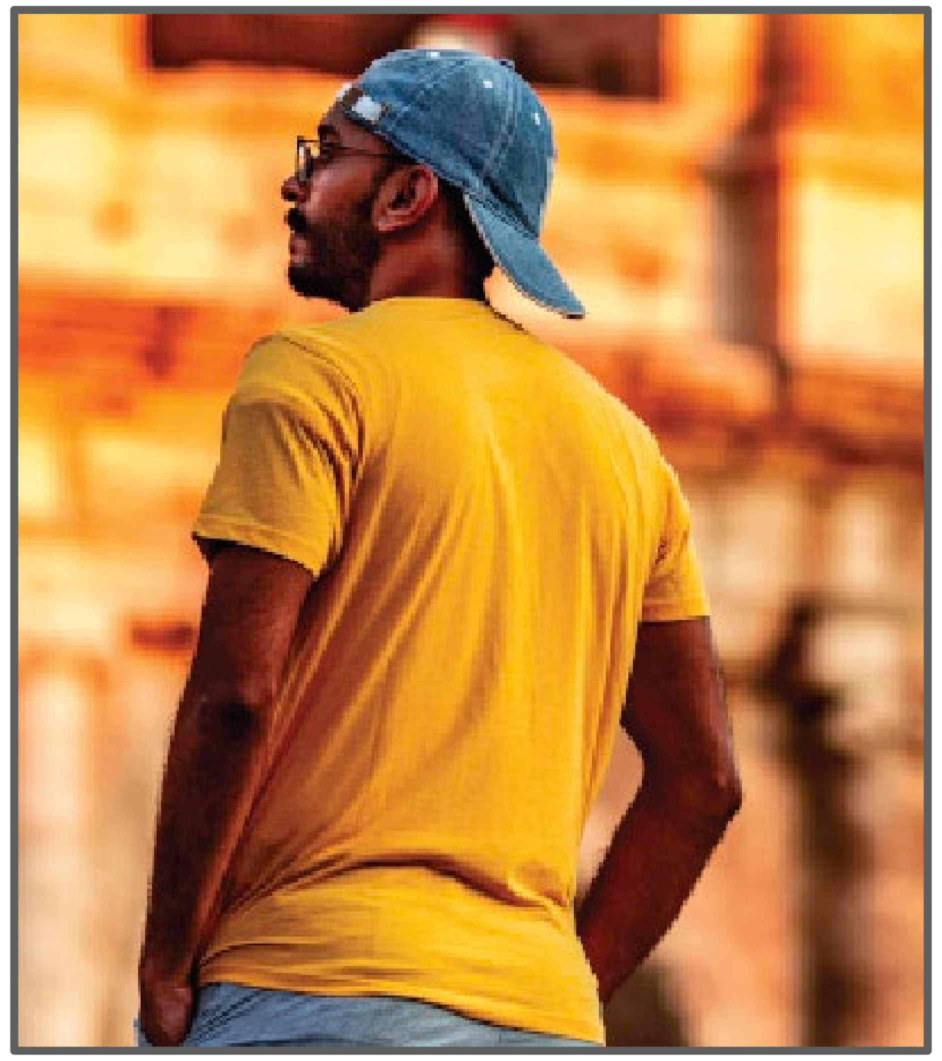


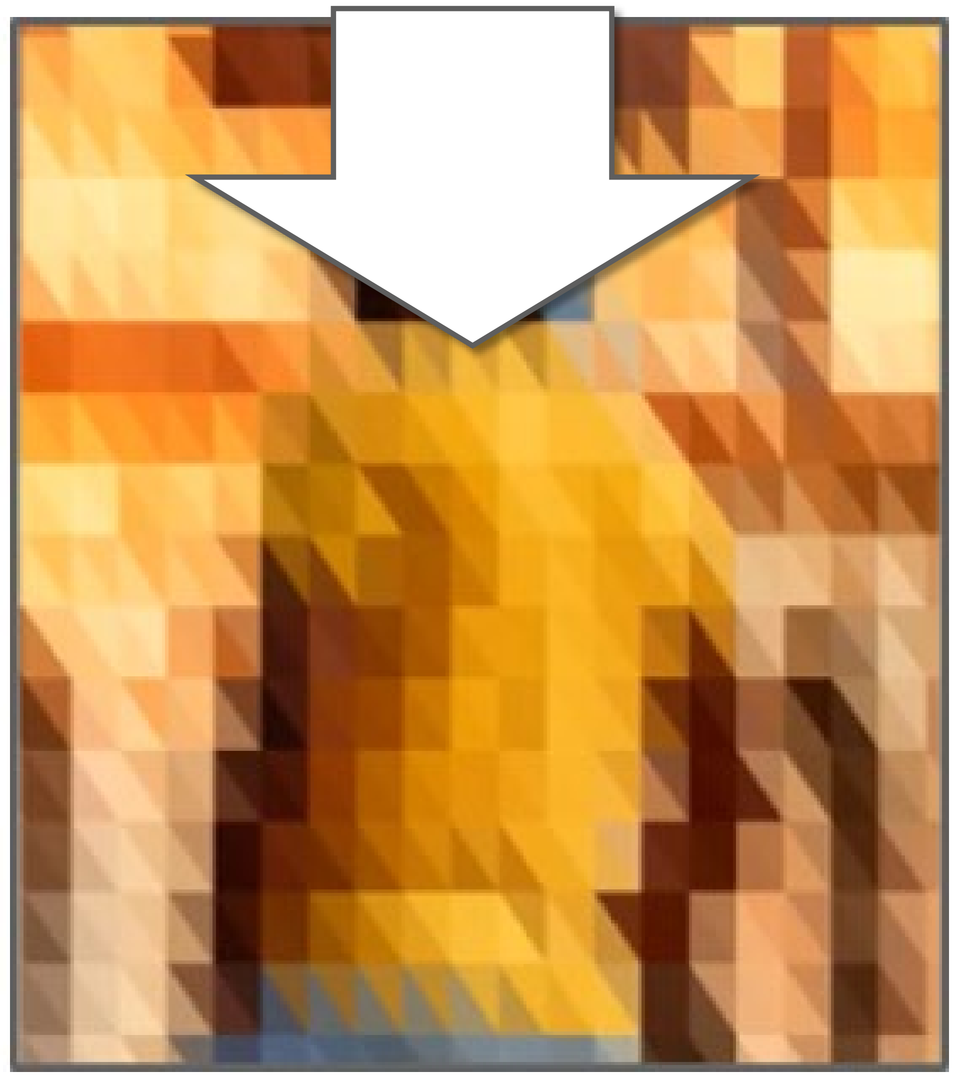
Triangulated image

Ignoring δ-terms produces 0 gradient

δs account for discontinuities

No gradient





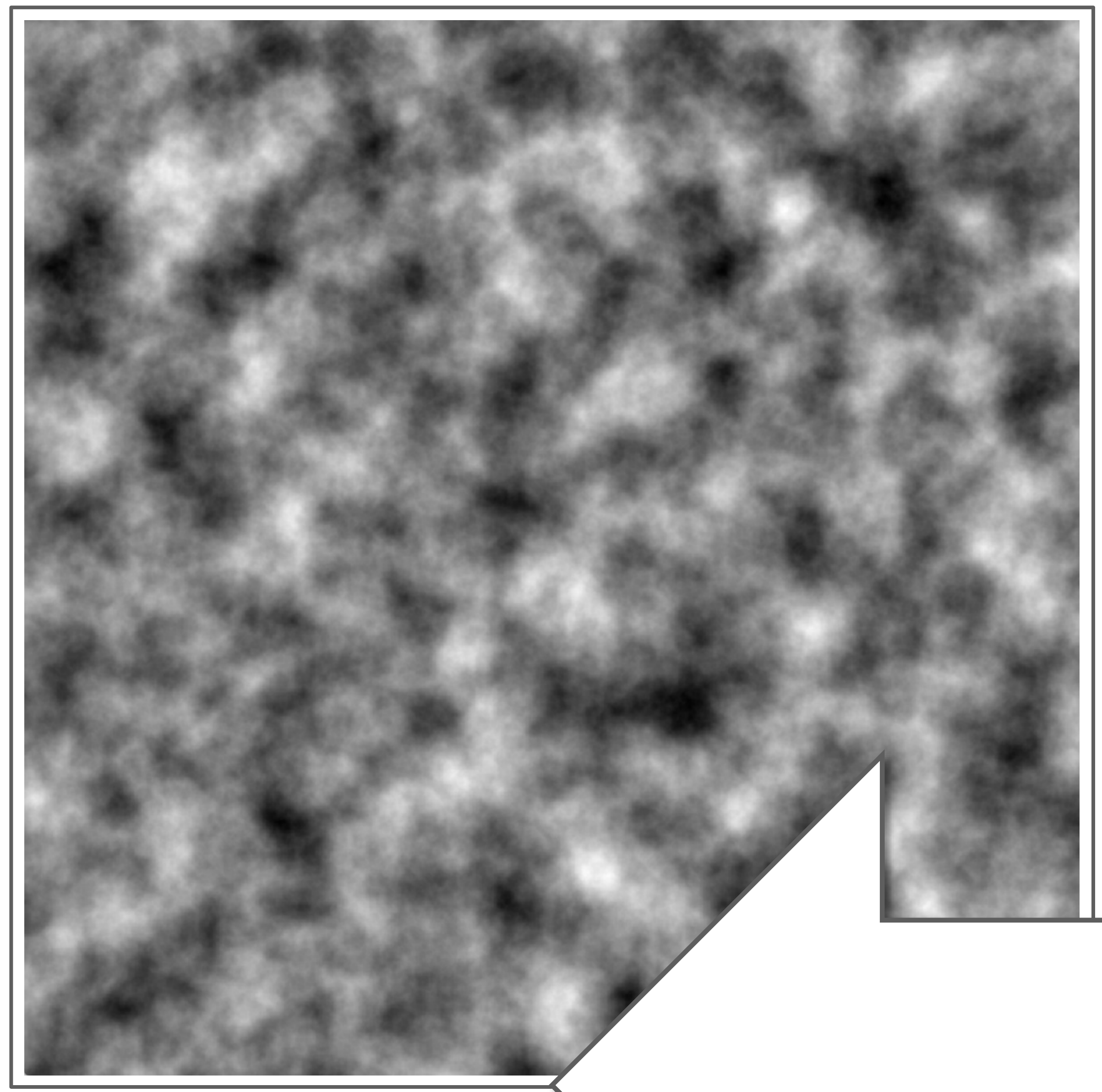
Target Image

Optimize with **Ours**

Optimize with **Traditional Auto-diff**

Thresholded noise shaders

Perlin noise map



Threshold

*... * [noise > t] * ...*

Inverse problem:

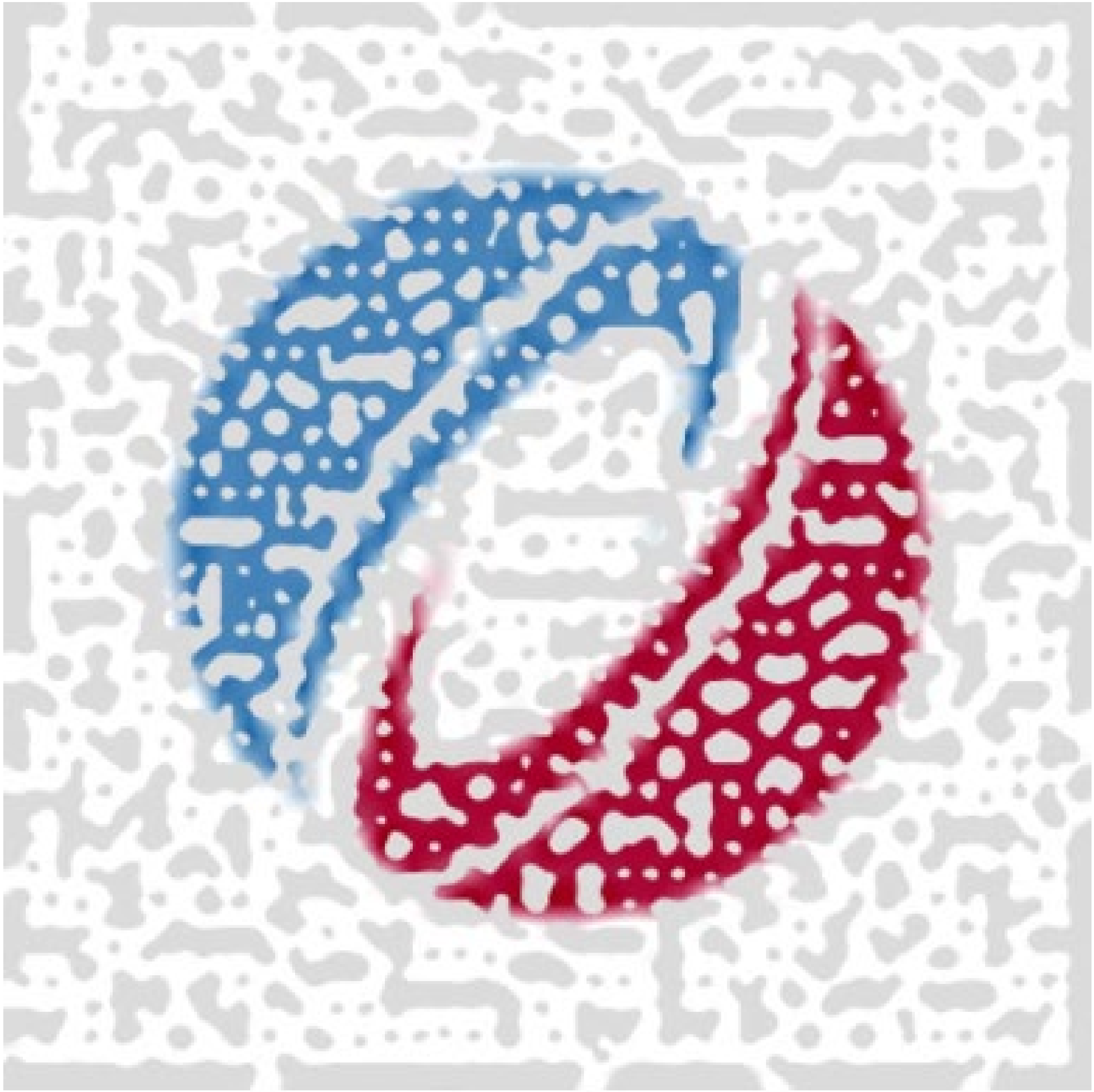
Optimize *noise* to fit a *target pattern*

Thresholded noise
(Discontinuous)

Bozo's Donut
(Perlin 1985)

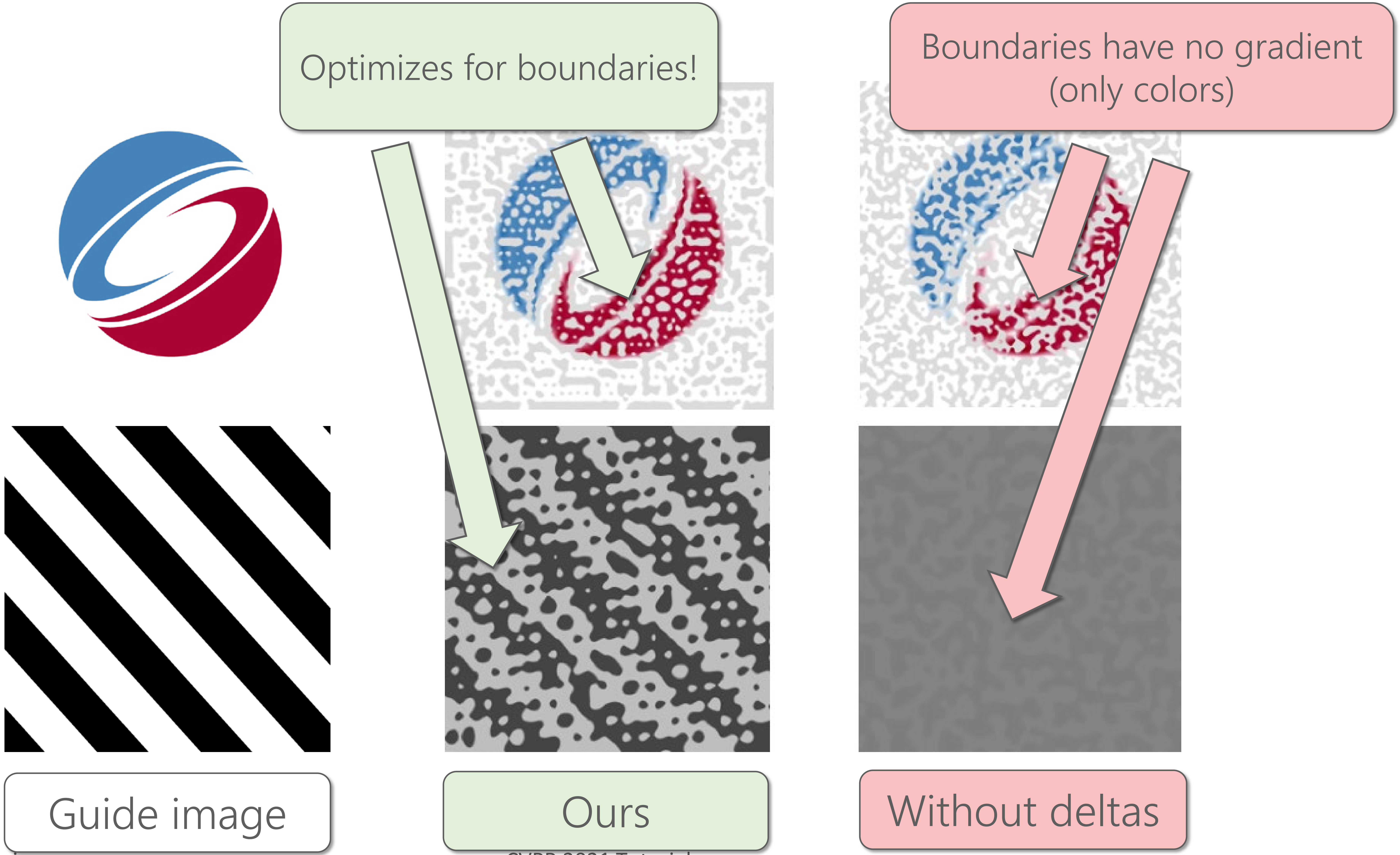
Inverse shader design using our approach

Guide image

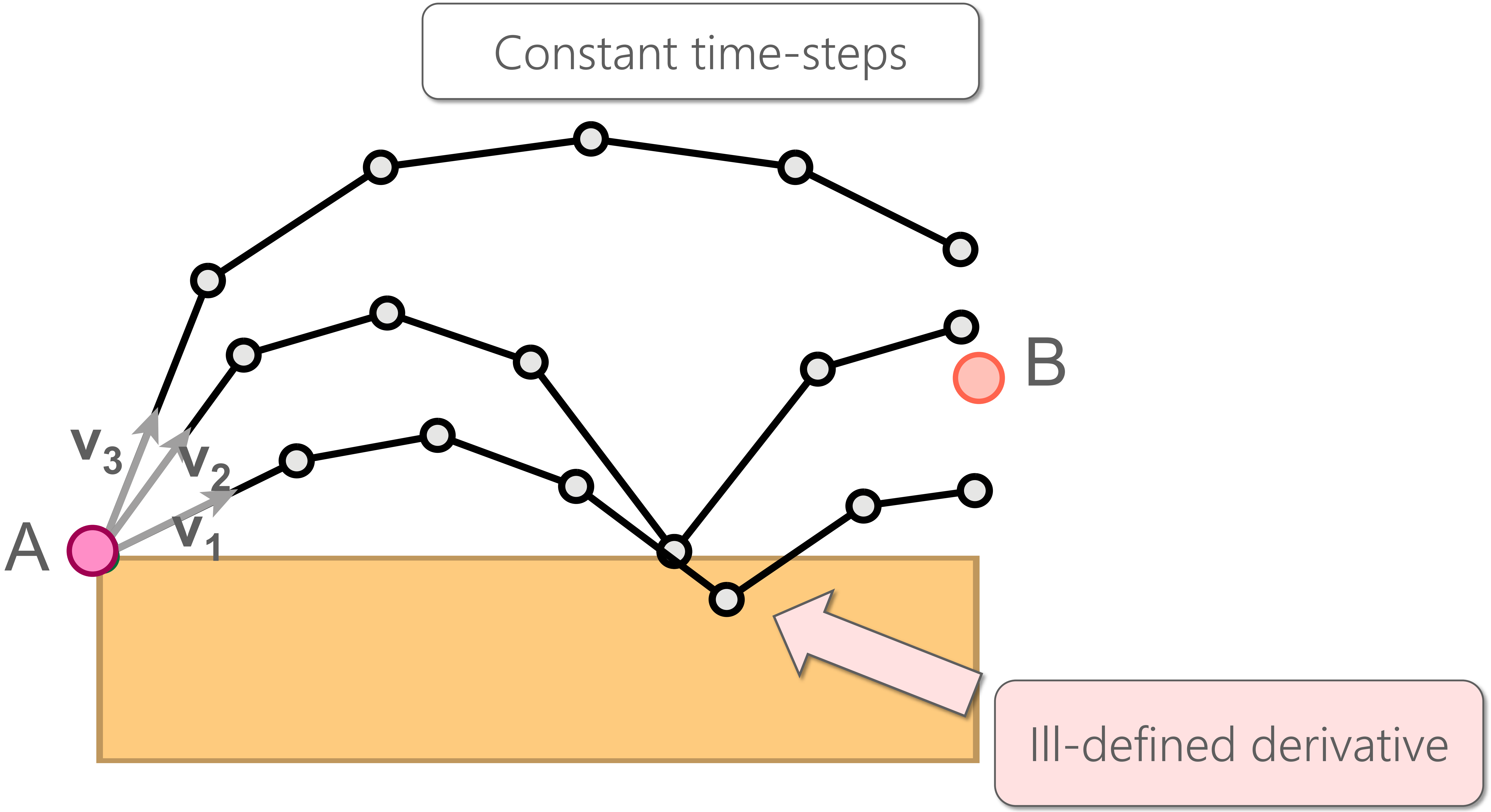


Optimized with
Ours

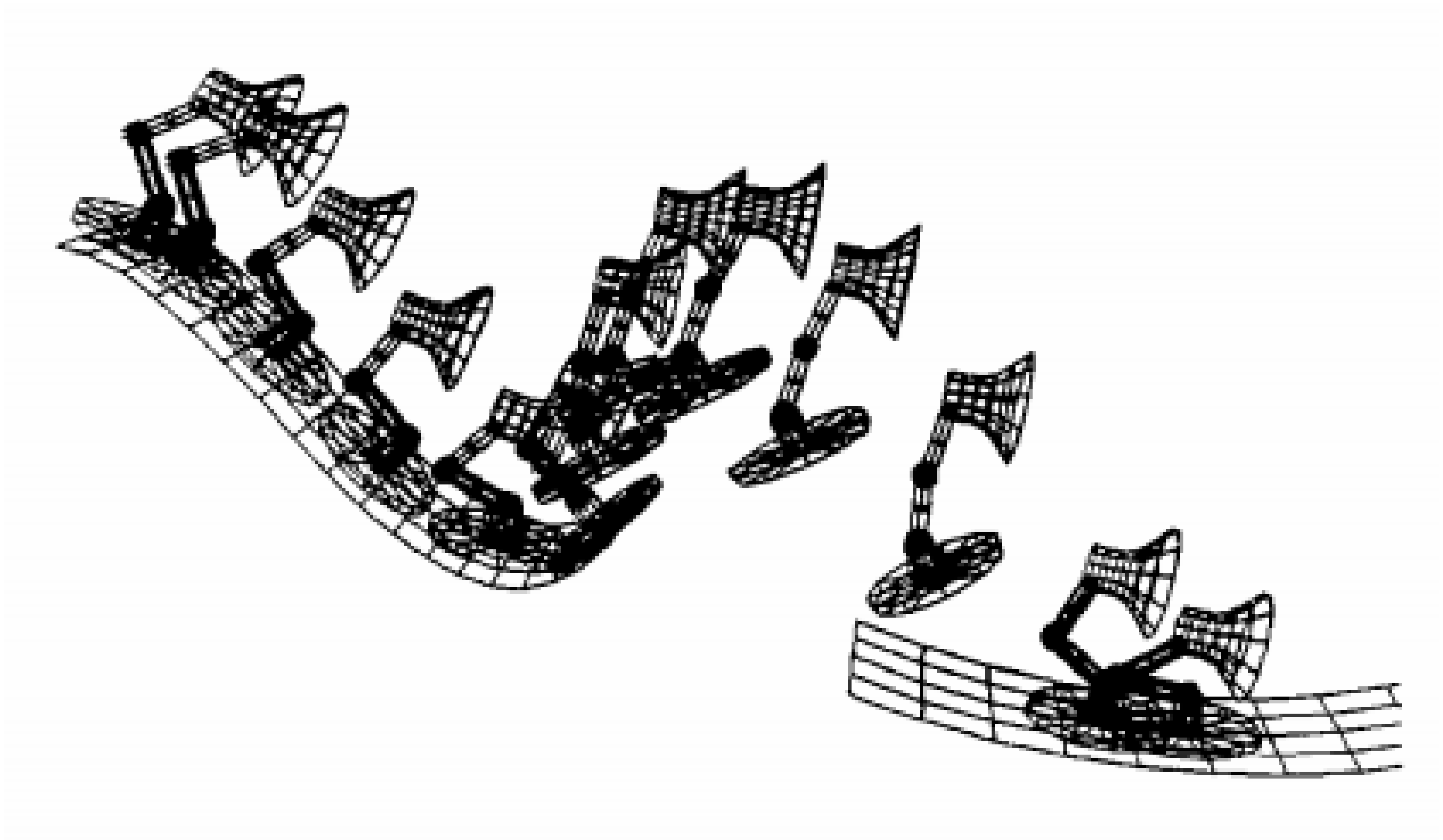
Ignoring delta terms produces incorrect results!



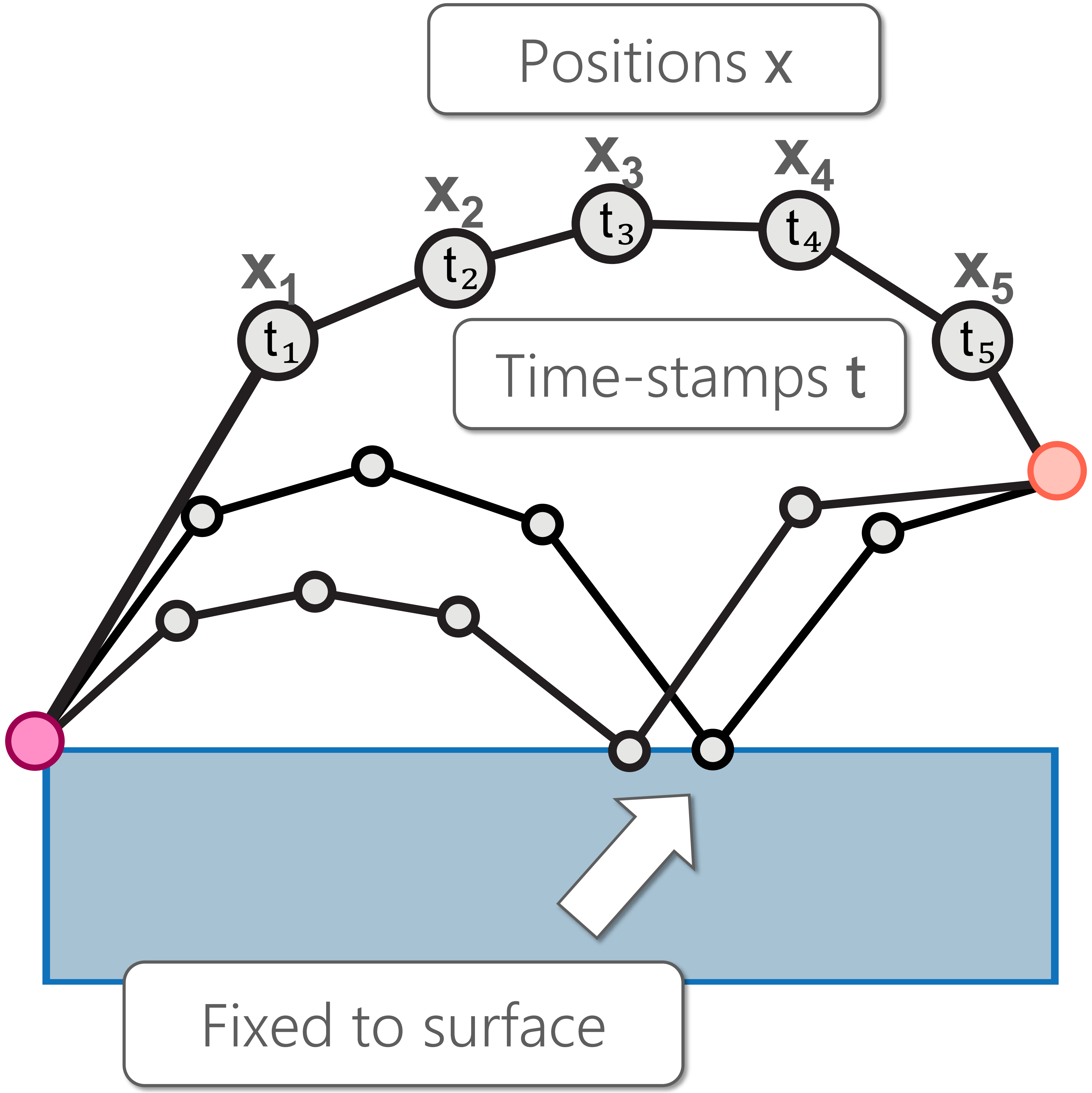
Application: Animation with hard contact



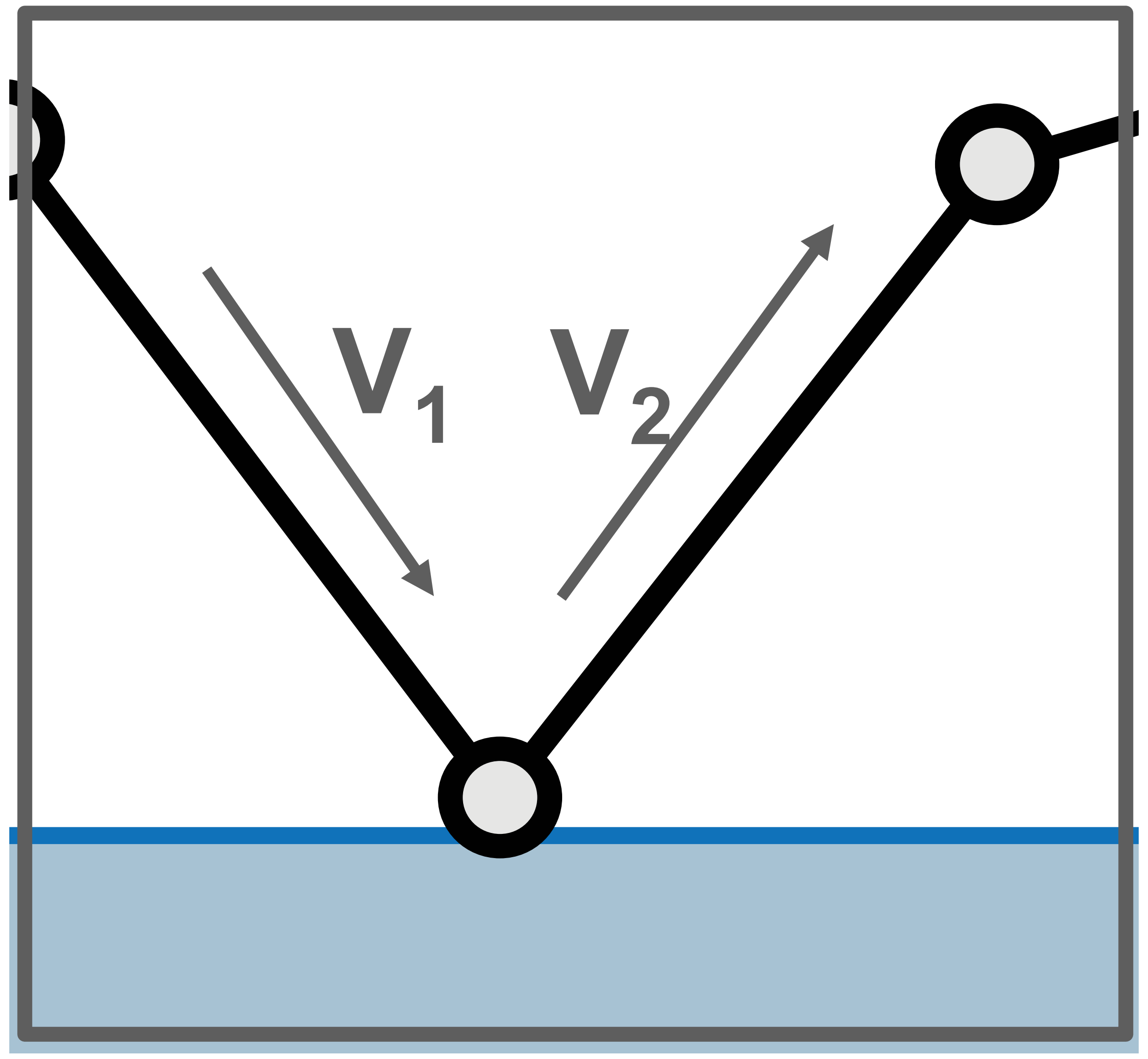
Inverse physical simulation with *space-time constraints*



Spacetime constraints (1988)

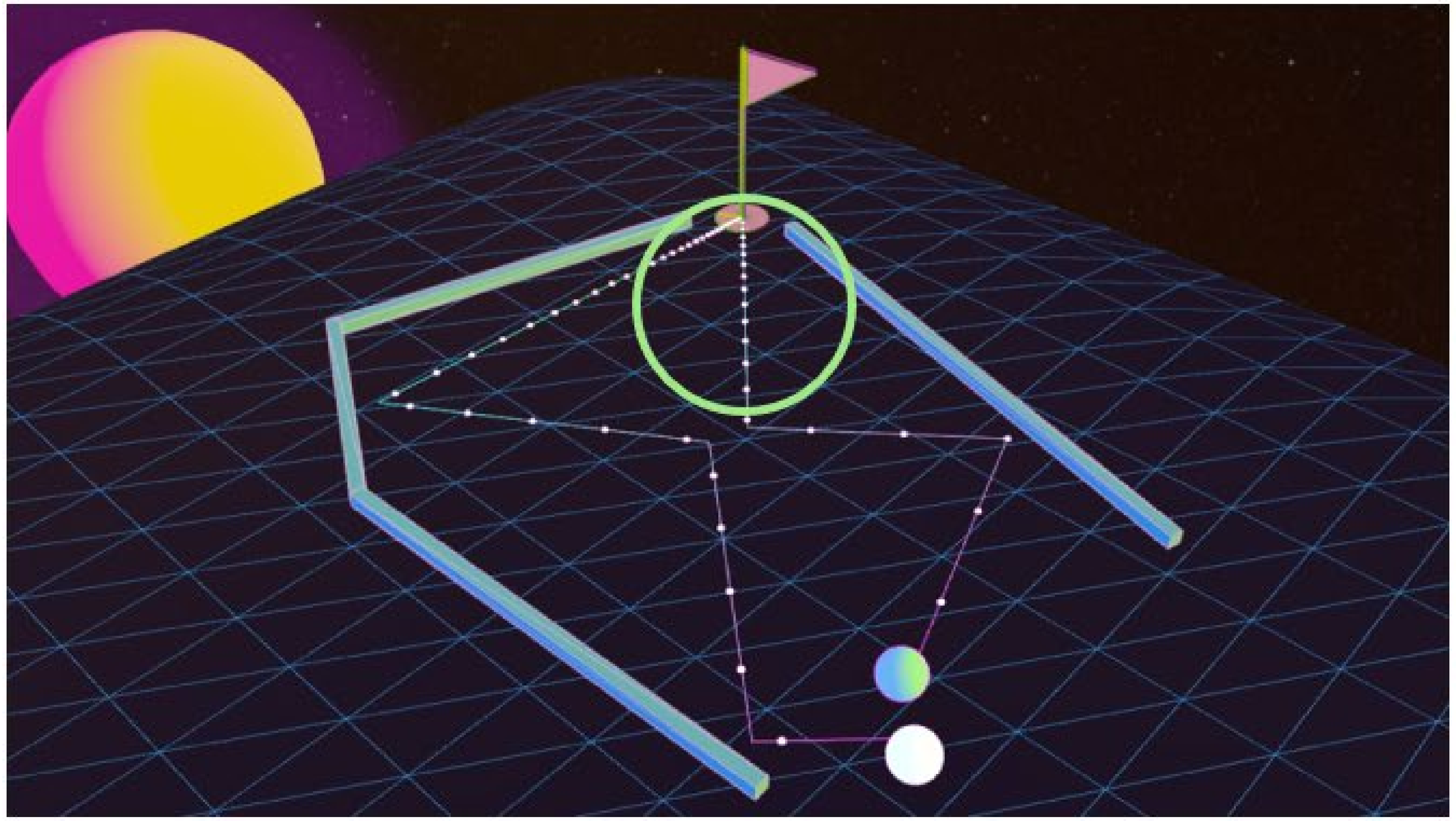


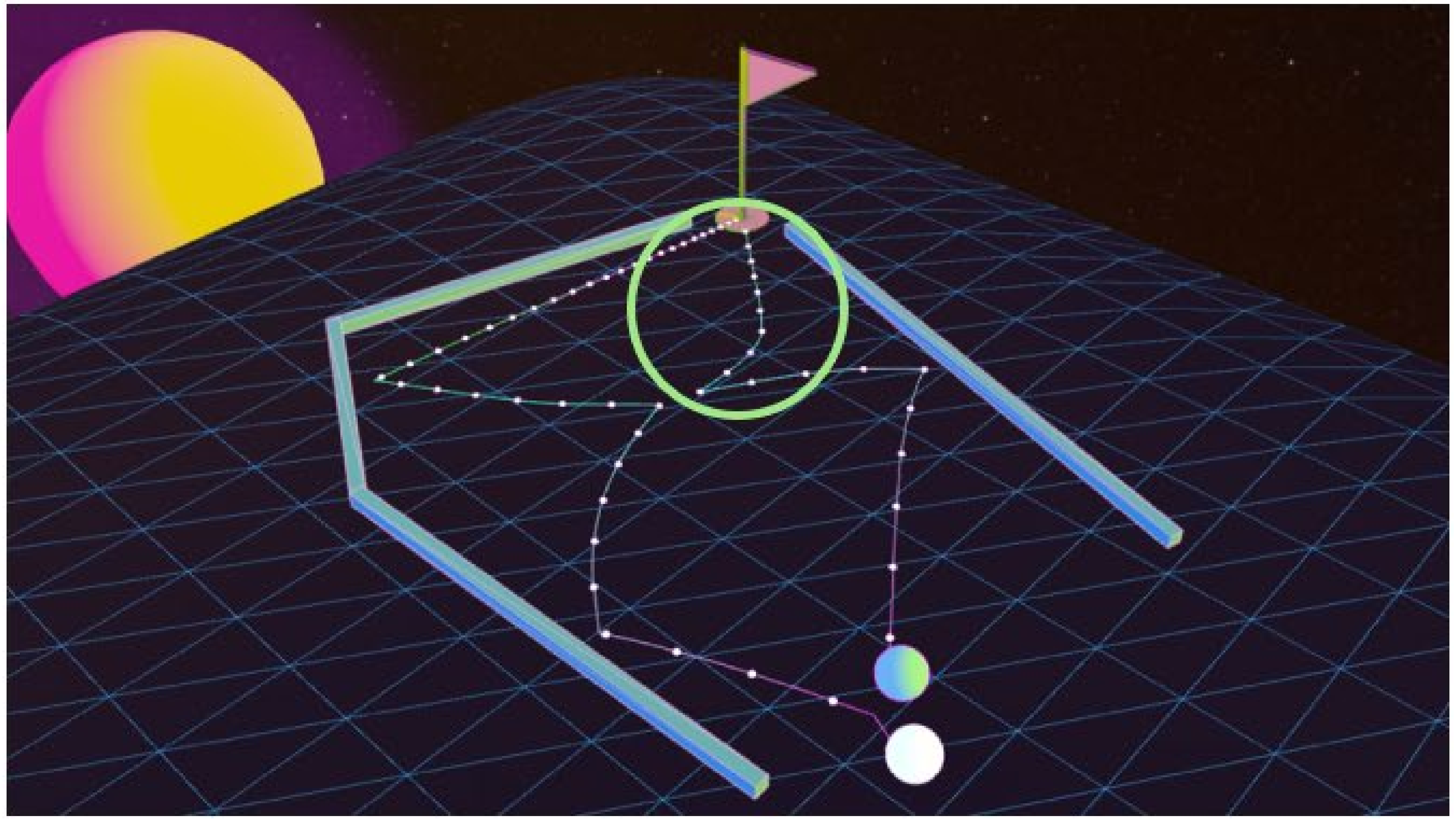
Physical simulation with space-time constraints



Discontinuous velocity

Comparison with ignoring boundaries



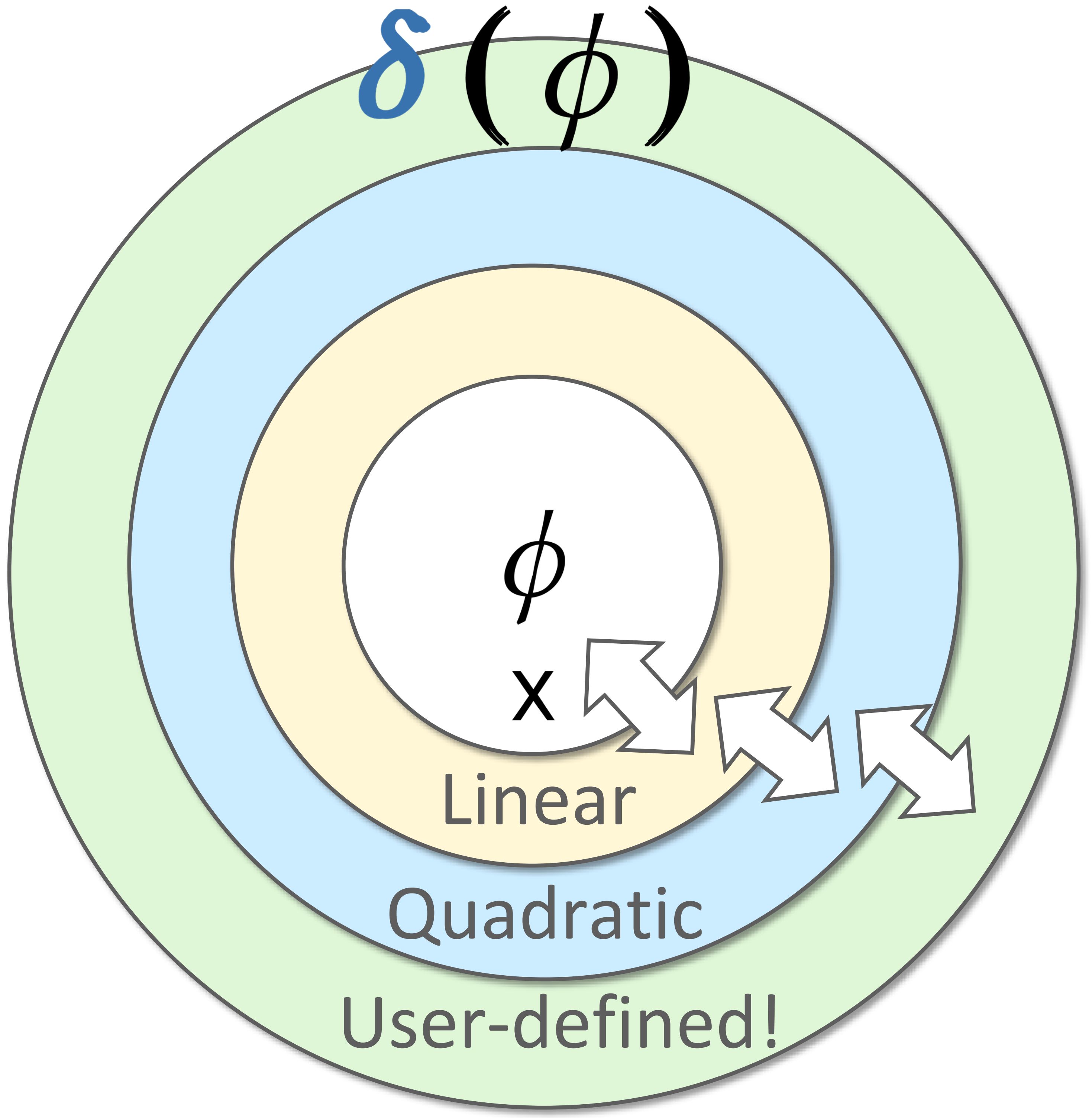


Optimized with **Ours**
(Physically *correct*)

Optimized with **Traditional Auto-diff**
(Physically *incorrect*)

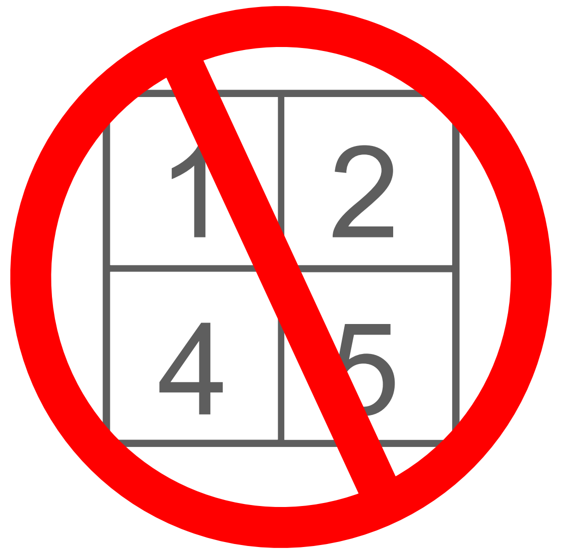
Limitations & Future Directions

Diffeomorphisms



Scalability

Scalar variables
(no indexing)



Expression lang
(no looping)

Modularity

$$\int_{x=a}^{b} \ldots \; \delta(\phi)$$

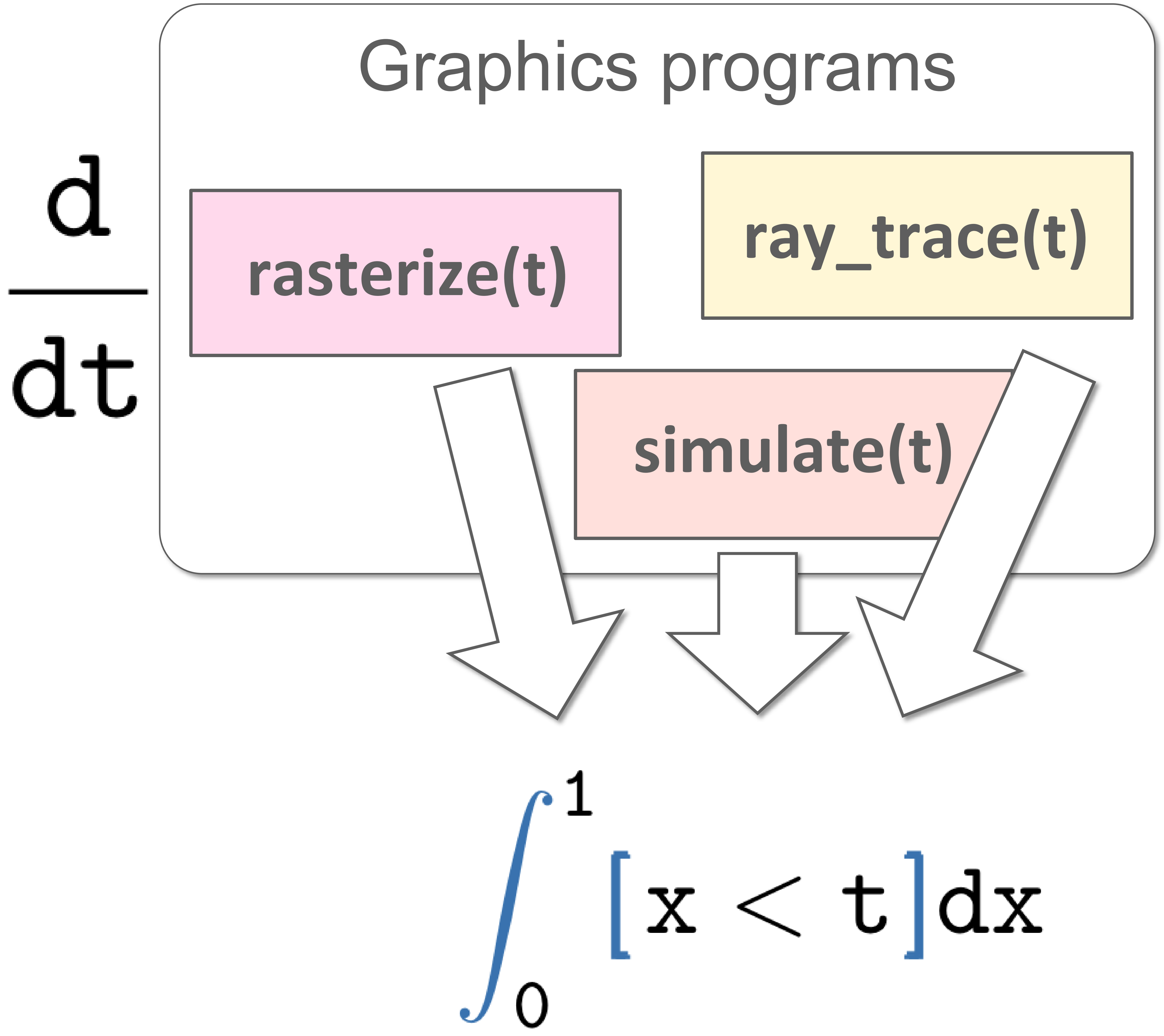
Requires global transforms

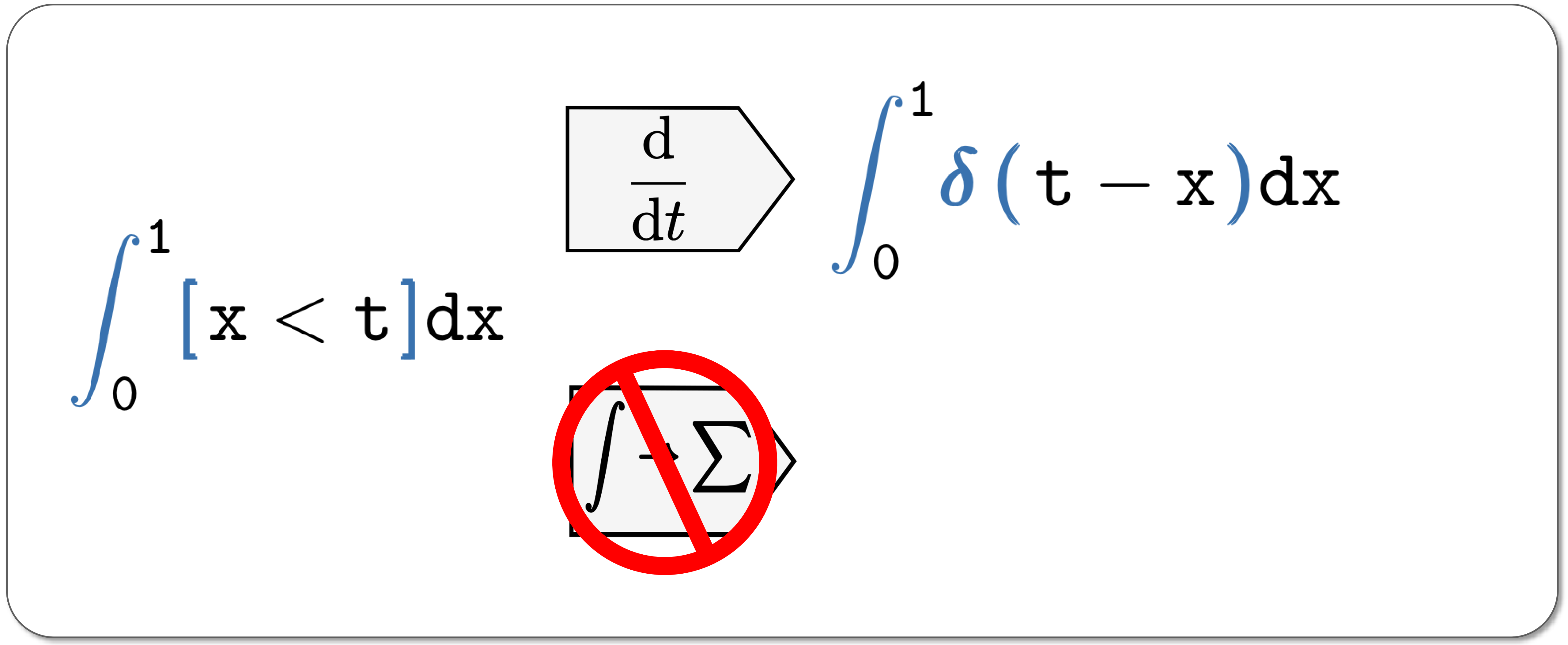
$$\int e * \left(\delta(\phi_1) + \delta(\phi_2)\right)$$

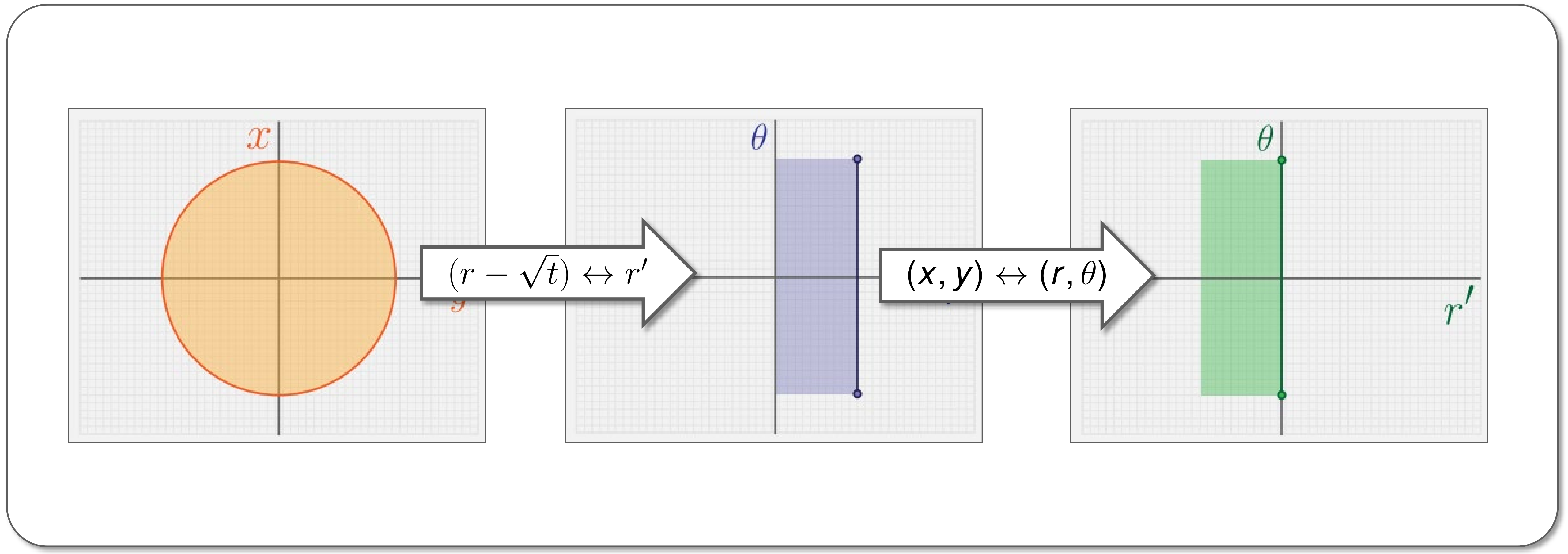
Code duplication

$$\int e * \delta(\phi_1) + e * \delta(\phi_2)$$

Systematically handling discontinuities: A Summary







Mitsuba 2

- A **general-purpose** **differentiable** renderer developed by Jakob et al.

- **Strengths**
  - Feature-rich (e.g., supports hyper-spectral and polarized rendering)
  - Efficient at handling many (e.g., millions) of parameters

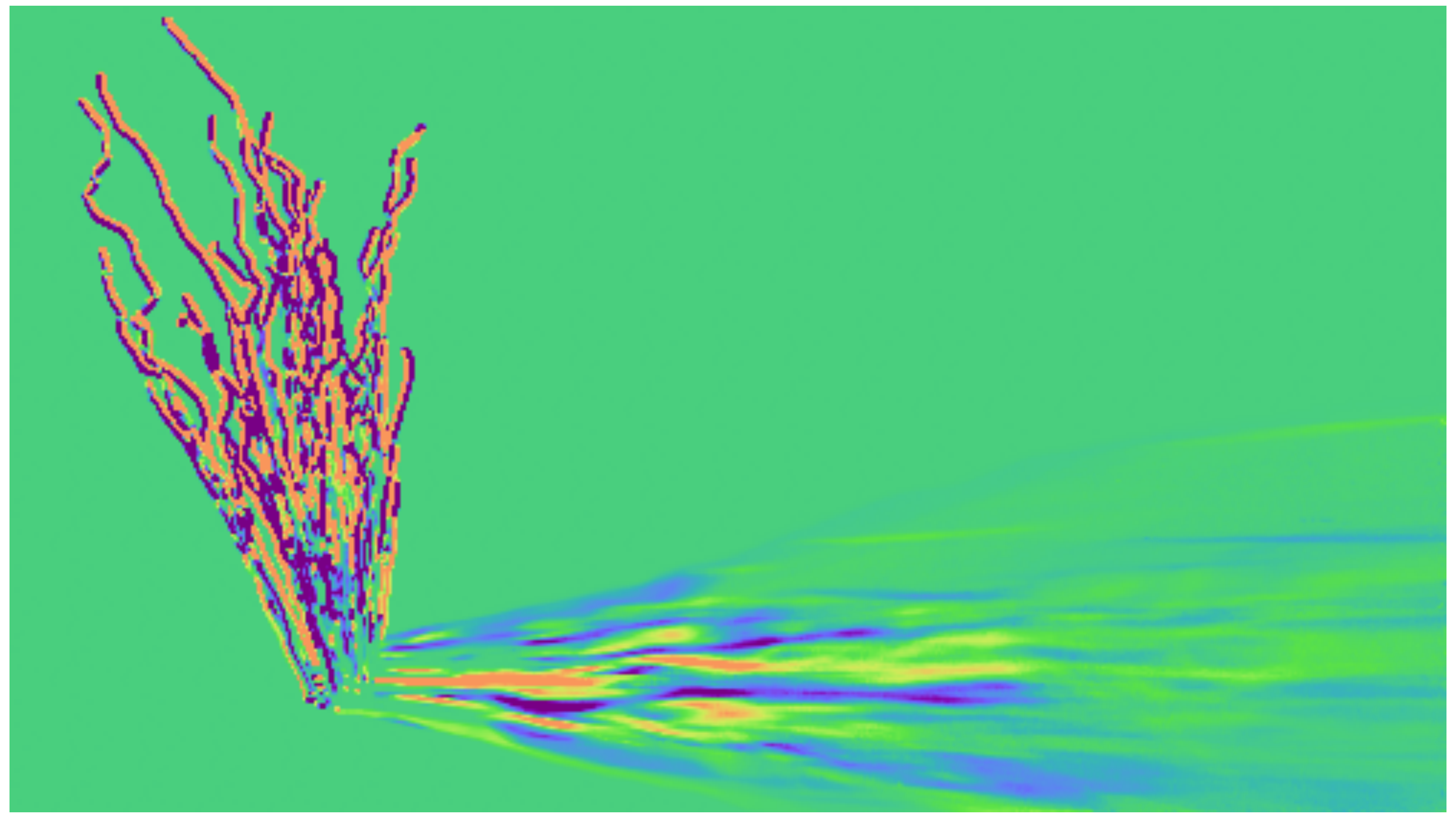
- **Weaknesses**
  - Currently offers limited support for differentiation with respect to geometry

PSDR-CUDA

- A GPU-based **general-purpose** **differentiable** renderer
- Built upon the same numerical backend (i.e., `Enoki`) as Mitsuba 2
  - Much lighter weighted
  - Python bindings via `pybind11`
- Implements **path-space** **differentiable** path tracing [Zhang et al. 2020, 2021]
  - Fast and unbiased geometric gradients

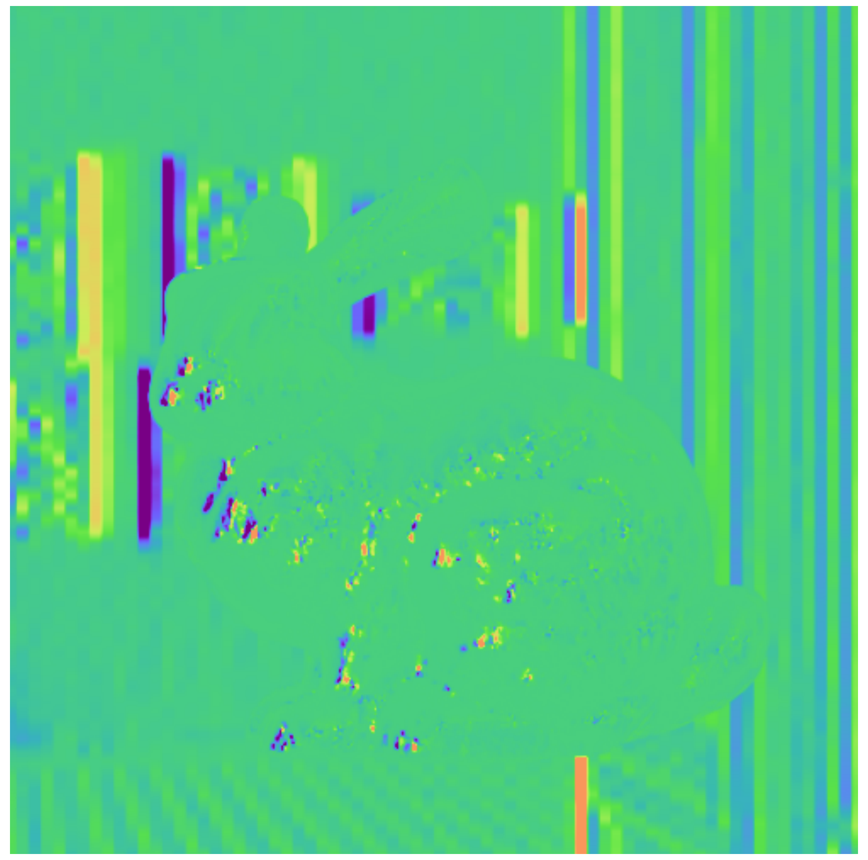
Try PSDR-CUDA!

Original image



Derivative image
(w.r.t. rotation of the object)

Original image



Derivative image
(w.r.t. rotation of the env. map)

Differentiating Image RMSE using PSDR-CUDA

```python
import enoki as ek
from enoki.cuda_autodiff import Float32 as FloatD, Vector3f as Vector3fD, Matrix4f as Matrix4fD
import psdr_cuda
```

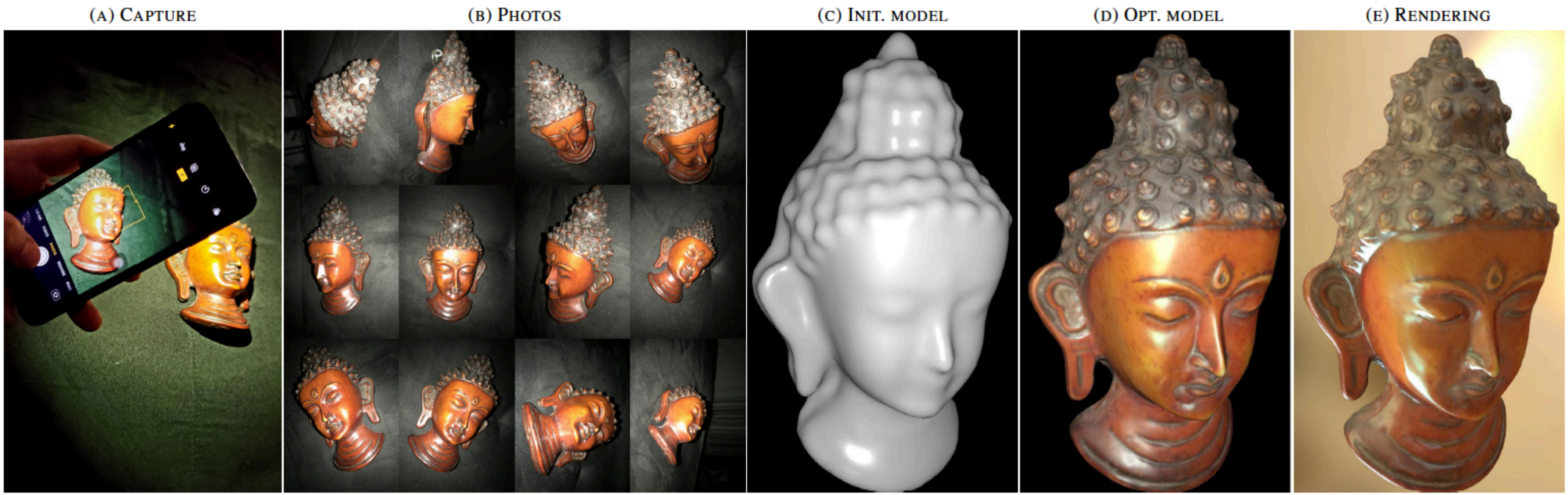
```python
# Load the scene without configuring it
scene = psdr_cuda.Scene()
scene.load_file('scene.xml', auto_configure=False)
```

```python
# Compute gradient with respect to mesh vertex positions
ek.set_requires_gradient(scene.param_map["Mesh[0]"].vertex_positions)
```

```python
# Configure the scene
scene.configure()

# Start rendering!
image = psdr_cuda.DirectIntegrator().renderD(scene, sensor_id=0)
```

```python
# Compute the RMSE image loss
loss = ek.sqrt(ek.hmean(ek.squared_norm(target_image - image)))

# Reverse-mode autodiff
ek.backward(loss)

# Obtain the gradient of the loss
grad = ek.gradient(scene.param_map["Mesh[0]"].vertex_positions)
```

Application: Shape and Material Reconstruction



To appear at *Eurographics Symposium on Rendering (EGSR) 2021*

Joint work with Fujun Luan, Kavita Bala, and Zhao Dong

Application: Shape and Material Reconstruction

- We solve an **analysis-by-synthesis** problem by jointly optimizing:
  - Object shape (i.e., positions of all mesh vertices)
  - Object reflectance (as diffuse/specular albedo and roughness maps)

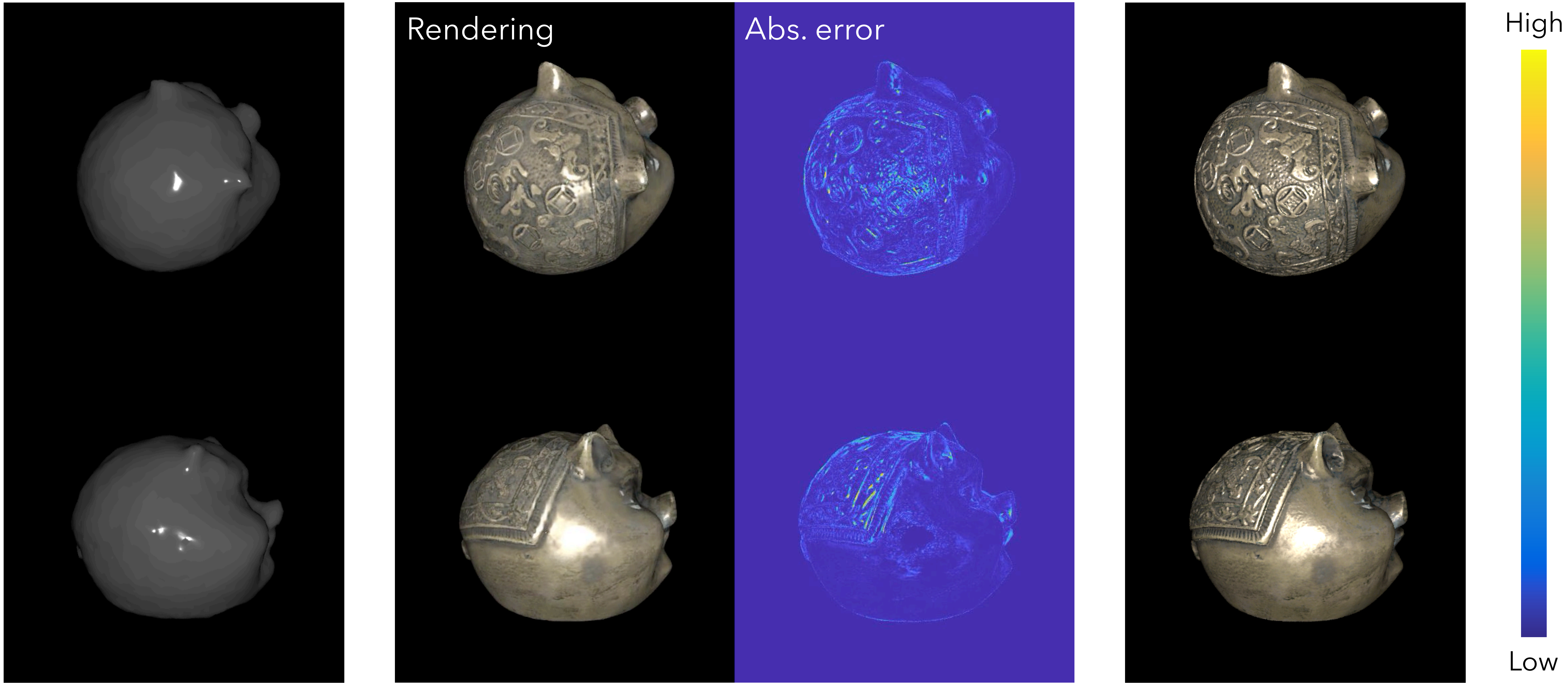
> **1M** parameters!

- Losses:
  - **Rendering** loss (computed & differentiated using PSDR-CUDA)
  - Regularization losses (e.g., mesh Laplacian, map smoothness)

- For improved **robustness**:
  - Coarse-to-fine scheme
  - When updating vertex positions, use elTopo [Brochu et al. 2009] to avoid self-intersections

Application: Shape and Material Reconstruction

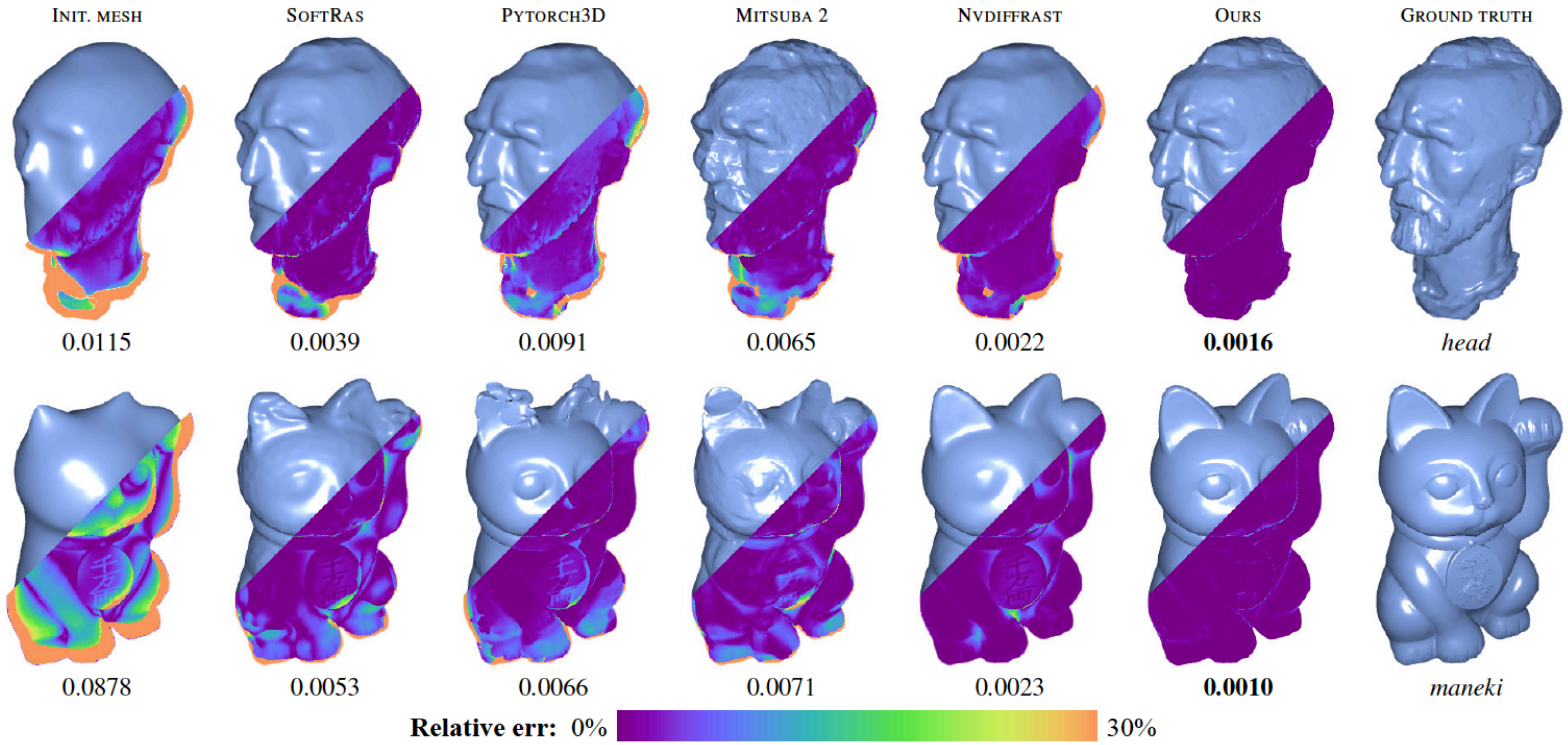
Joint optimization of *object shape* and *spatially varying reflectance* (100 views used, 2 shown)



Initial (Kinect Fusion) — **Optimized** (using gradients generated by PSDR-CUDA) — **Target**

Gradient Accuracy Matters!

Inverse-rendering results with *identical* optimization settings



Reconstruction Results of Real Objects

(No normal mapping is used, all geometric details emerge from actual mesh geometries.)

Reconstruction Results of Real Objects

Re-rendering in novel 3D scene

Object insertion in augmented reality (AR)

Inverse-Rendering Performance

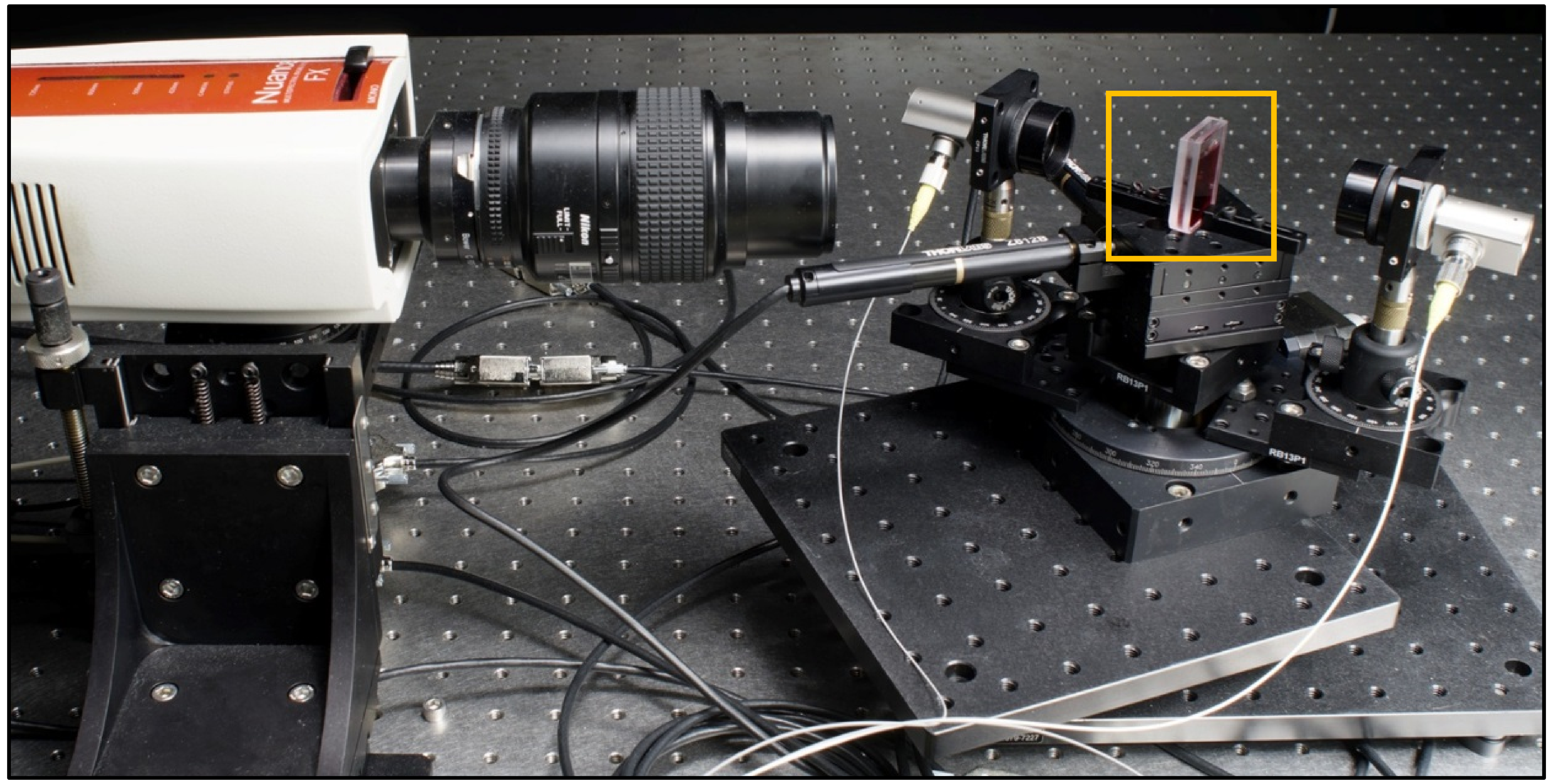
- Each inverse-rendering optimization takes 15–100 minutes

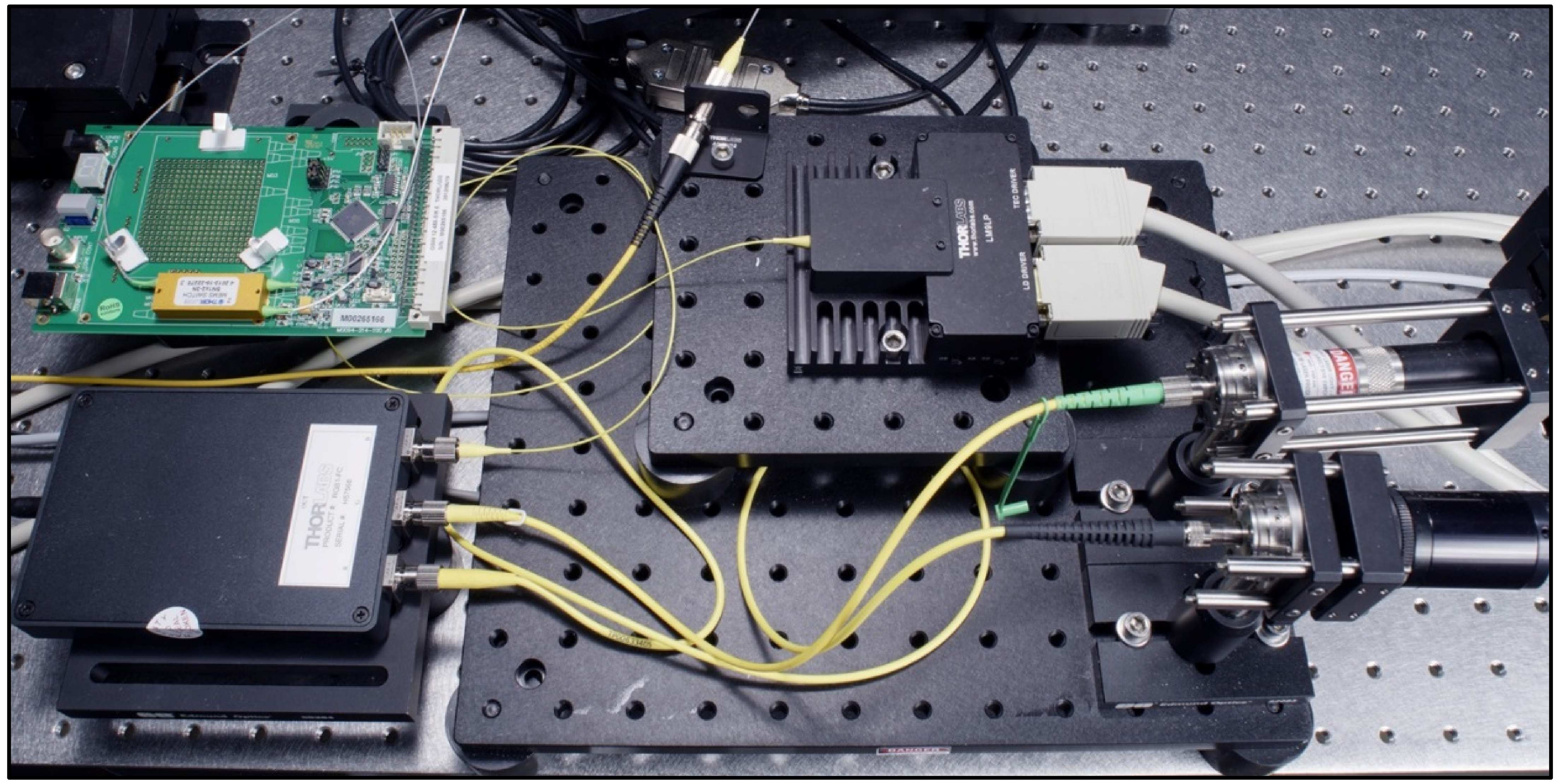
- **Inverse**-rendering performance ≠ **differentiable**-rendering performance
  - *Differentiable rendering* only accounts for **<4%** of total optimization time
  - *Geometric processing* (e.g., collision detection) takes up to **70%**

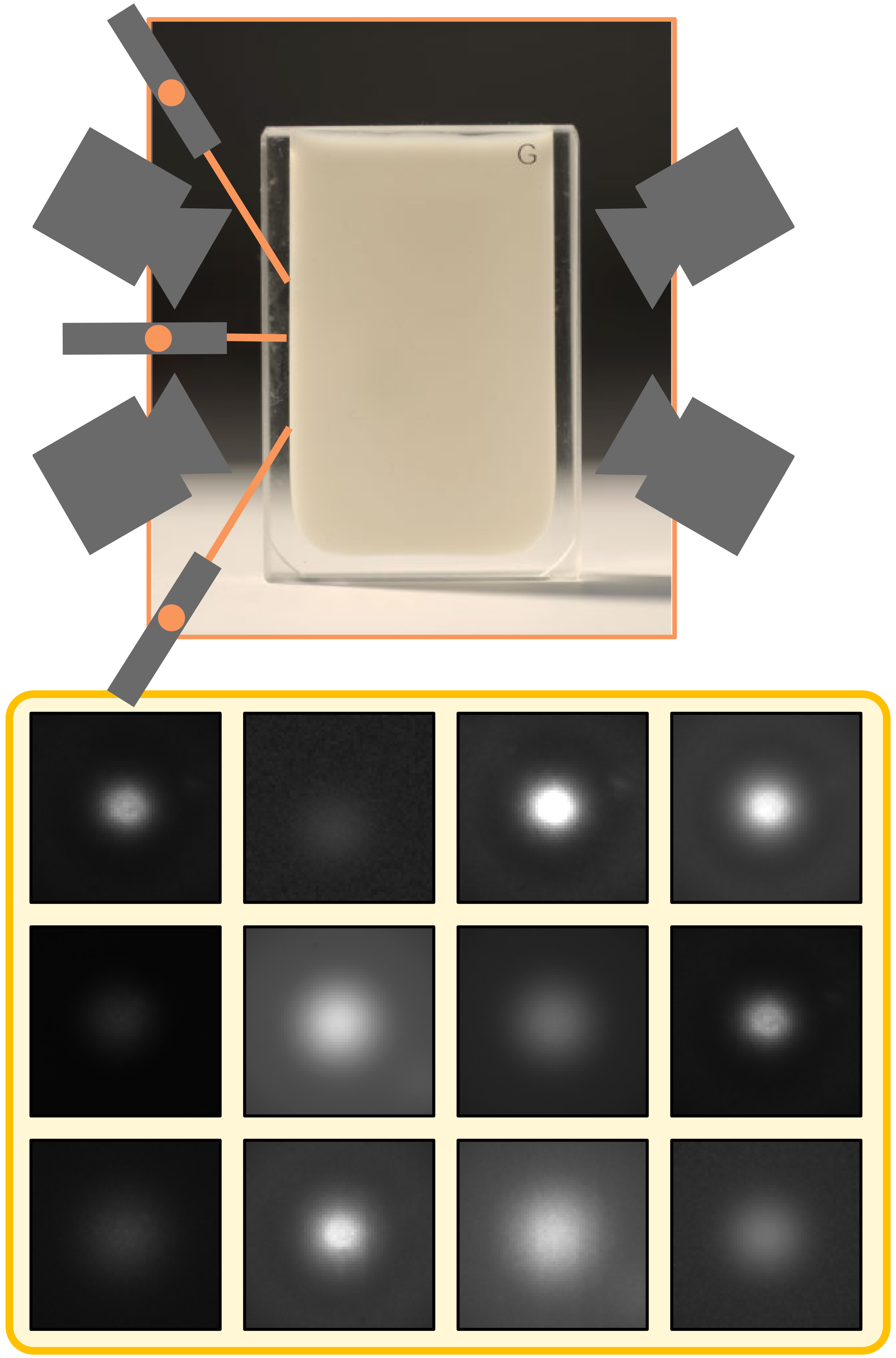
- We need much better geometric processing systems!
  - e.g., elTopo [Brochu et al. 2009] is CPU-based and single-threaded

Inverse scattering [Gkioulekas et al. 2013]

Acquisition setup

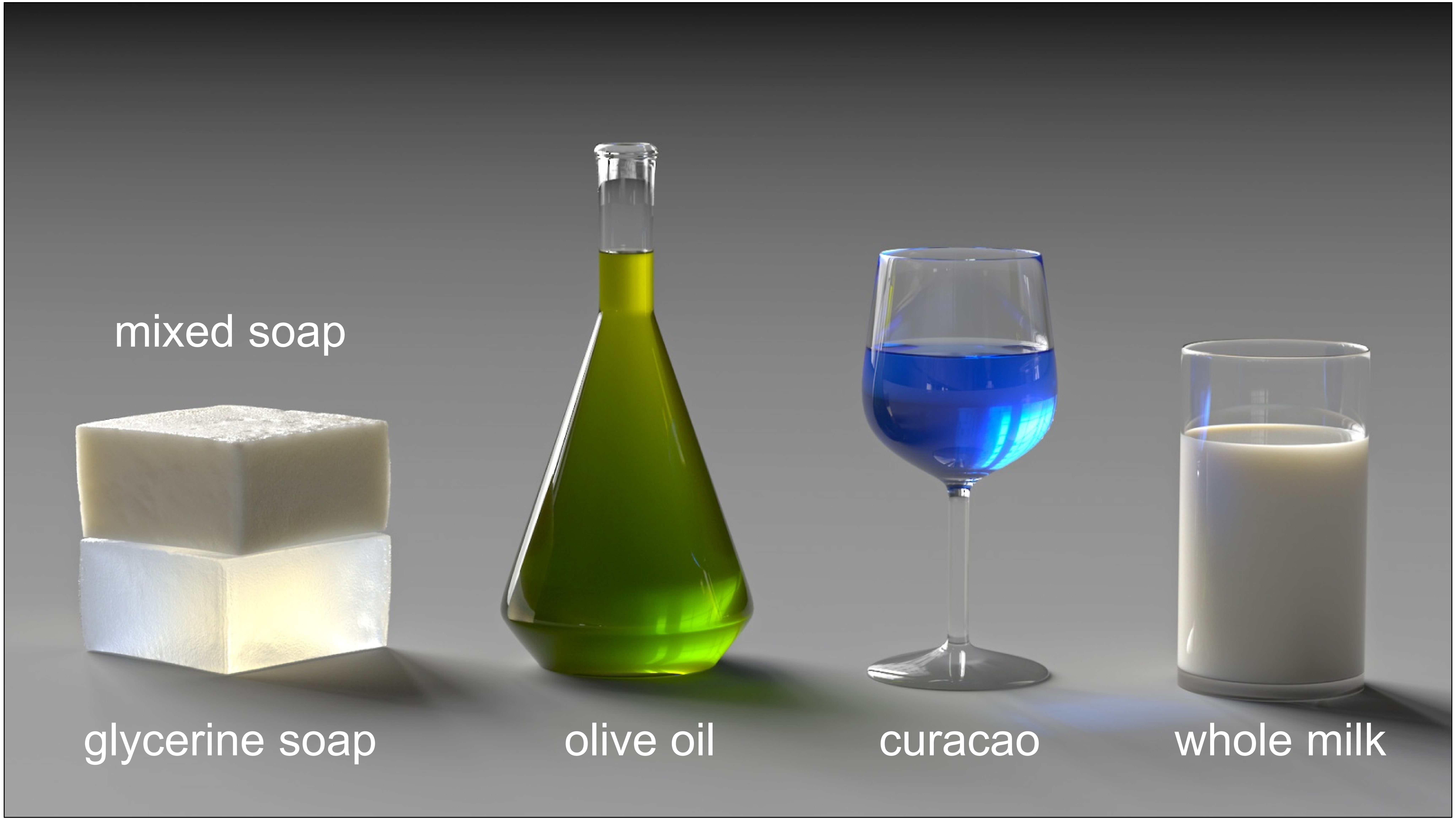






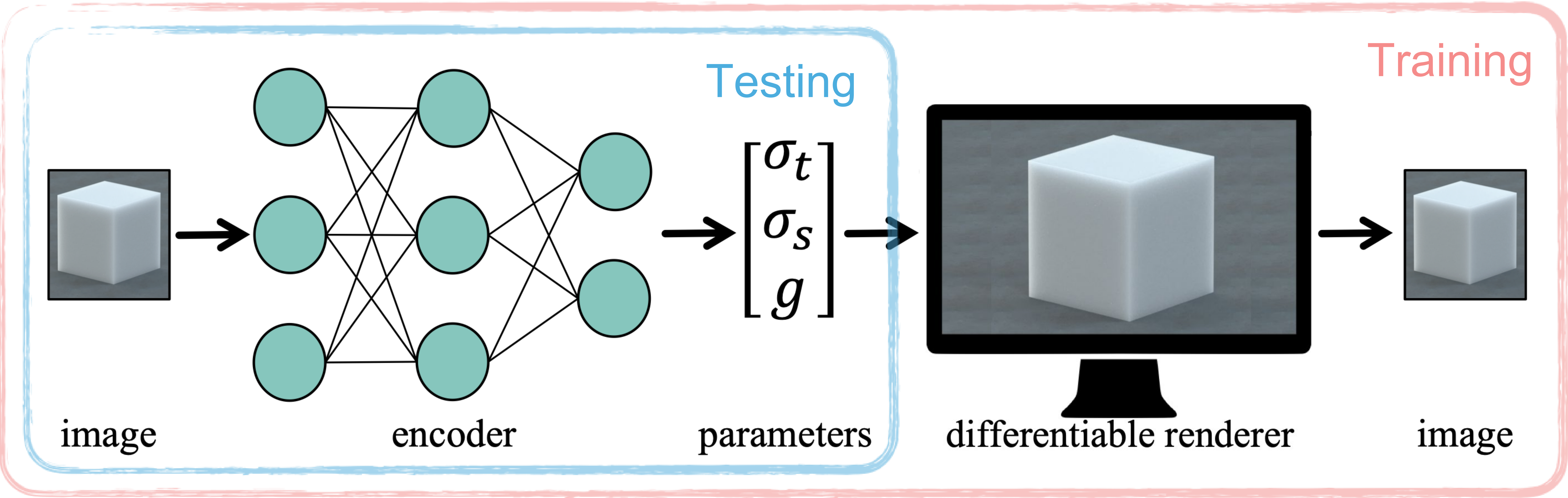
Invert using differentiable rendering

Synthetic renderings



Inverse transport networks [Che et al. 2020]

- Integrate physics-based rendering into **machine learning** pipeline
- Predict scattering parameters from images



- Utilize *image loss* provided by a volume path tracer to regularize training
- Use the trained encoder to perform inverse scattering during testing

Examples

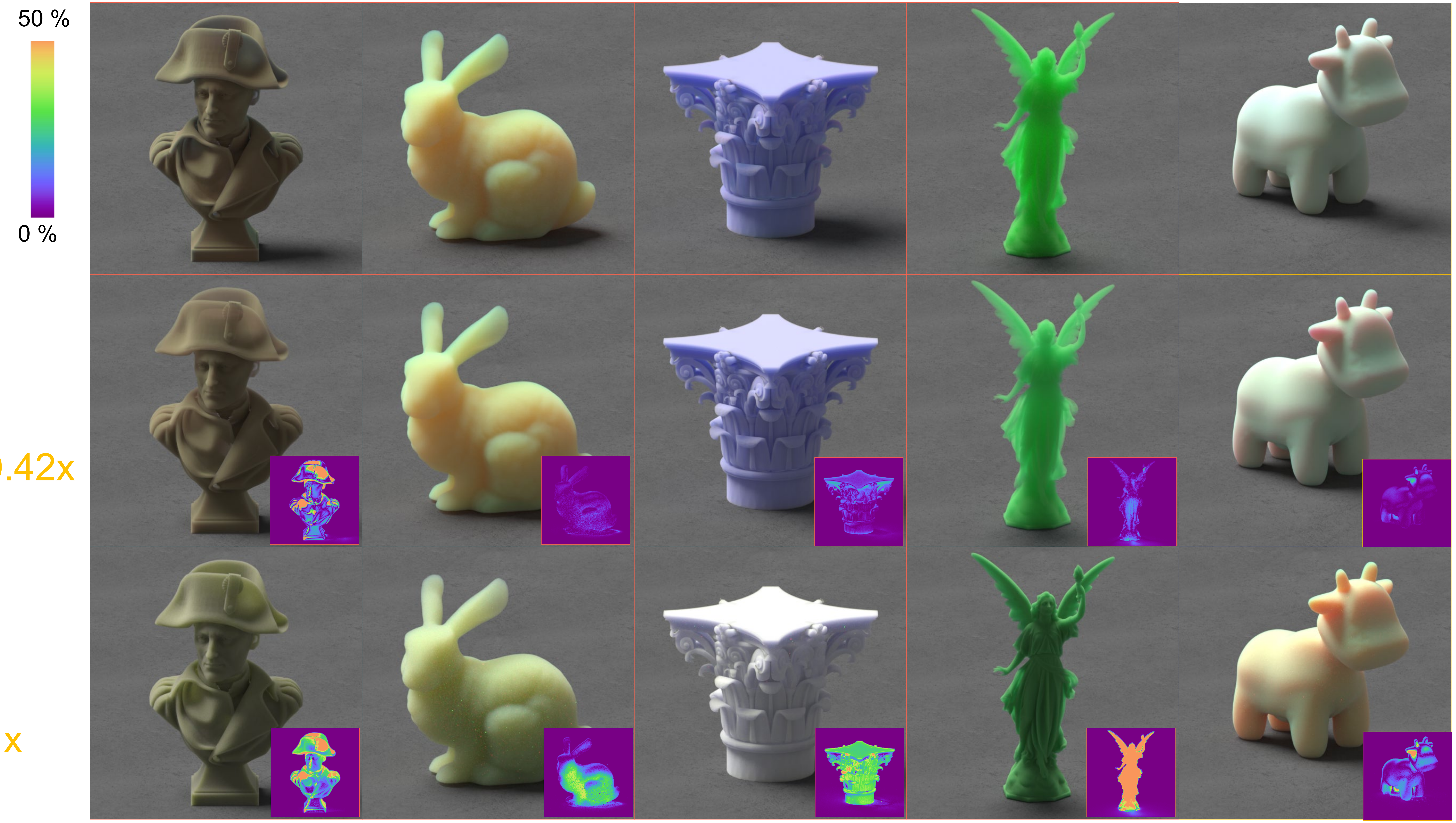
Groundtruth

Inverse transport network

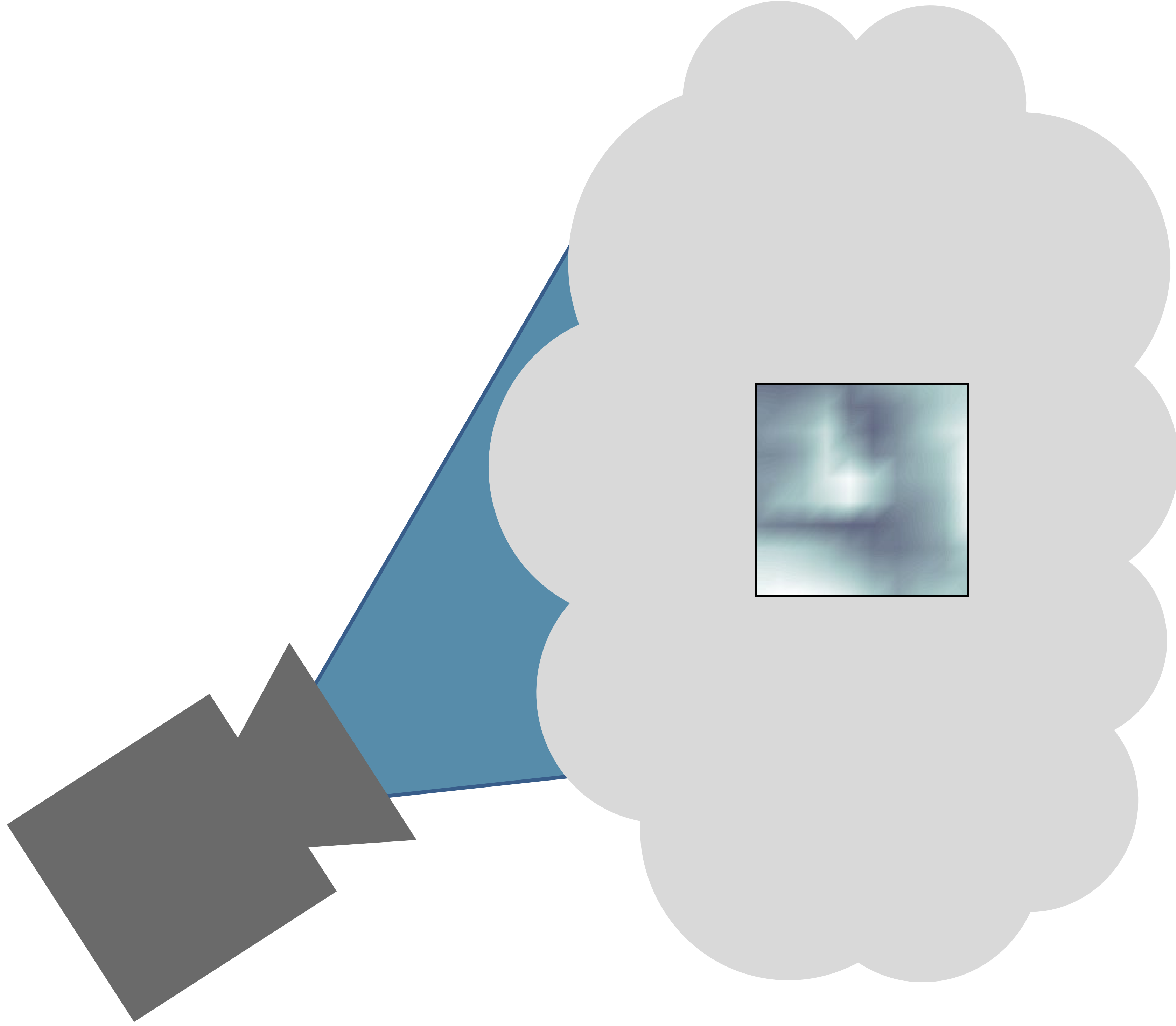
parameter loss: 0.60x
appearance loss: 0.40x
novel appearance loss: 0.42x

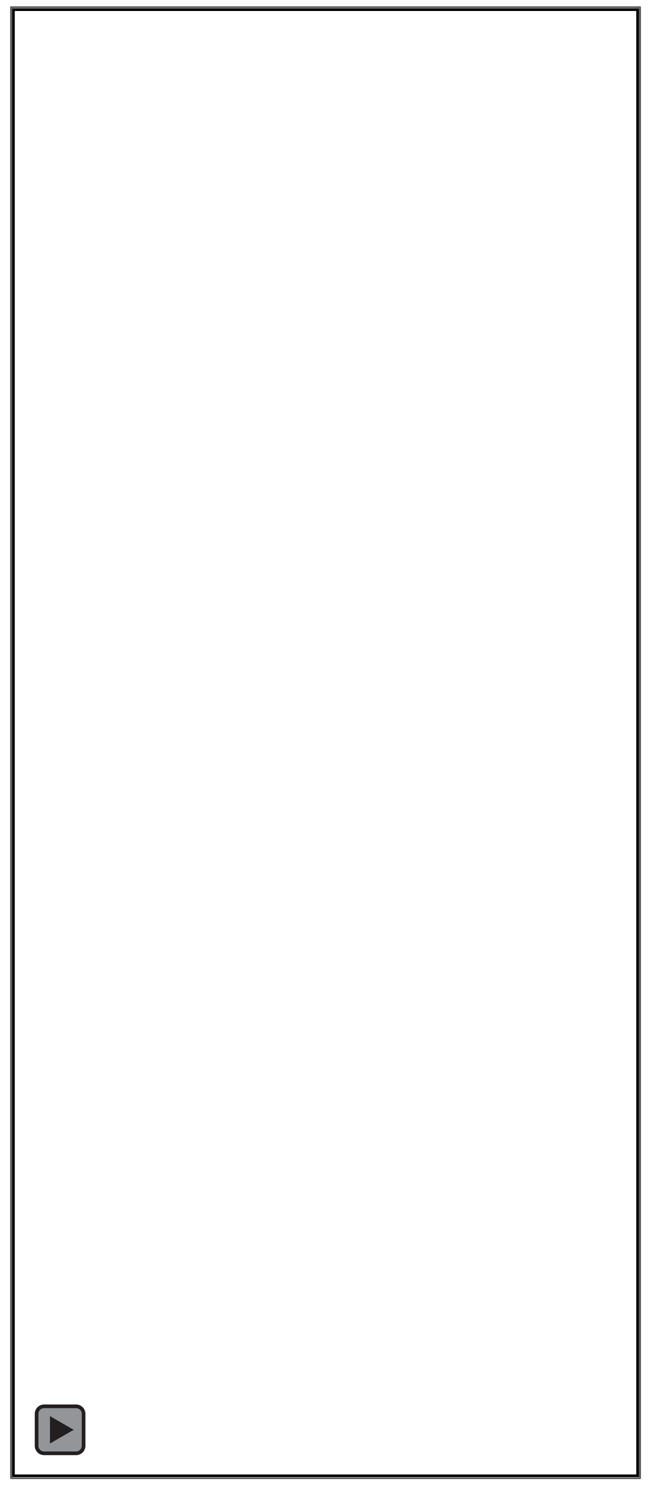
Baseline

parameter loss: 1x
appearance loss: 1x
novel appearance loss: 1x



Optical tomography [Gkioulekas et al. 2015]

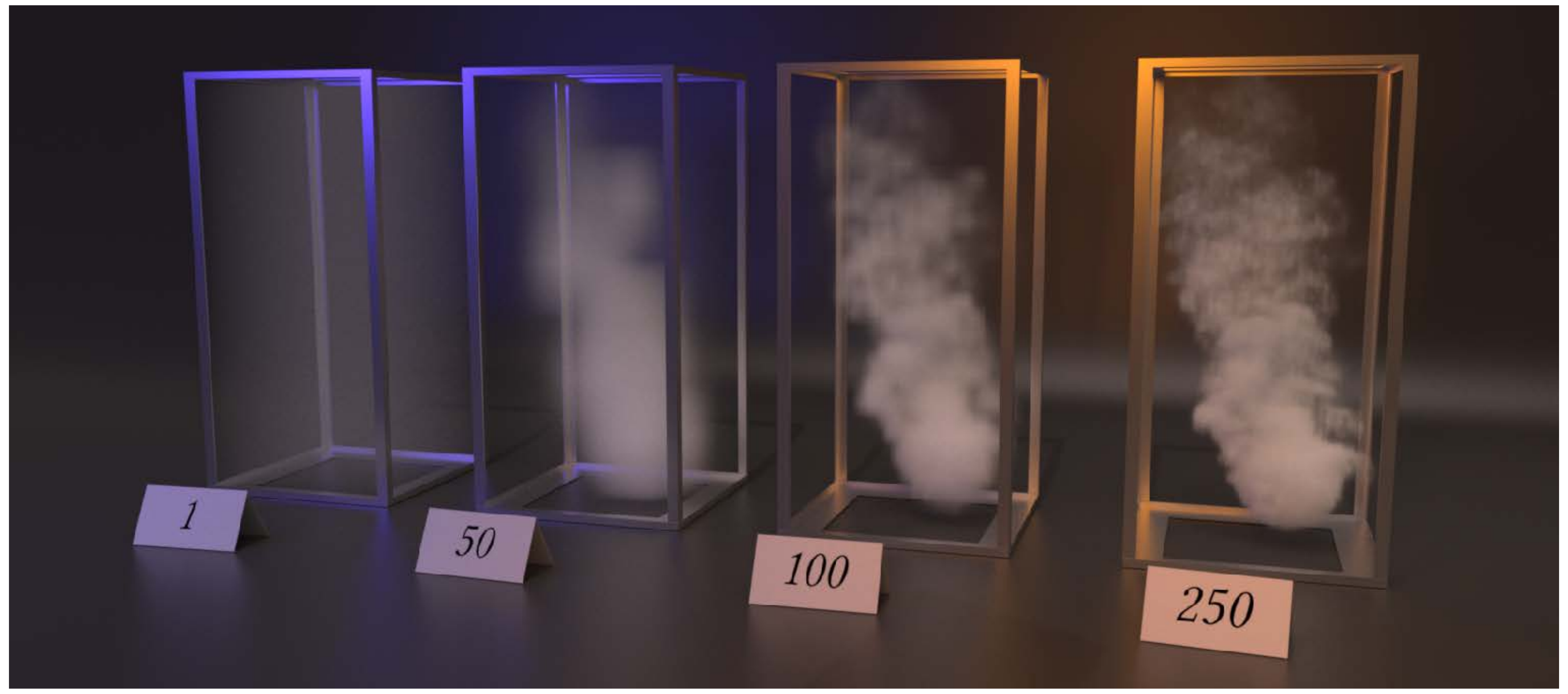




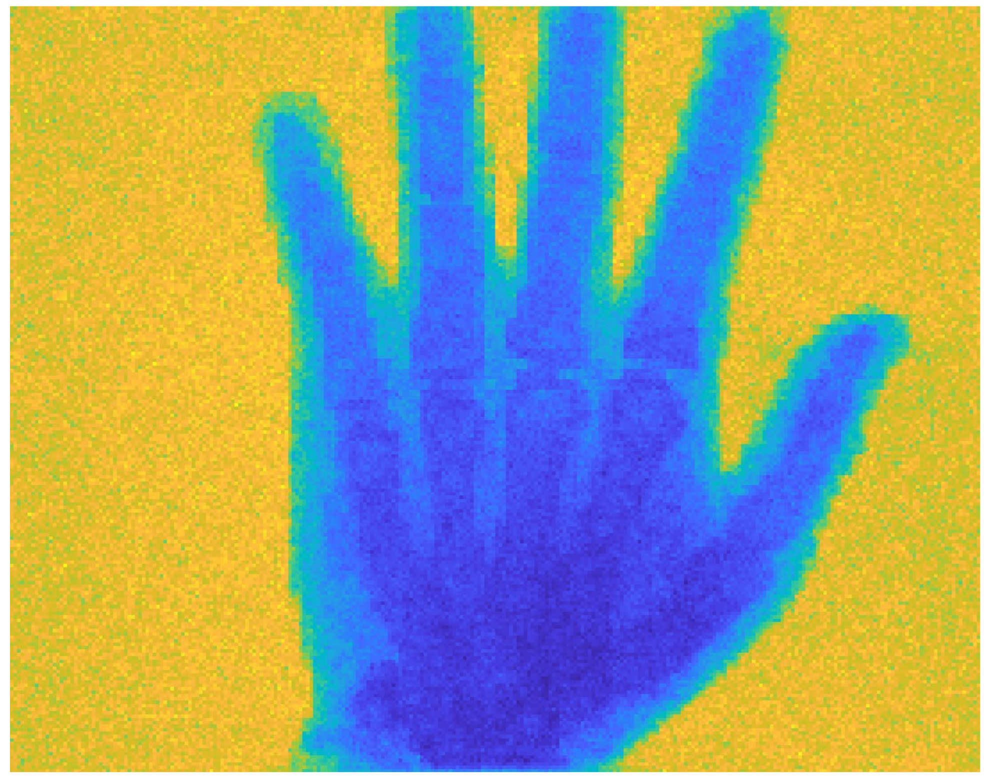
camera | thick smoke cloud | simulated camera measurements | reconstructed cloud volume | slice through the cloud

Active area of research

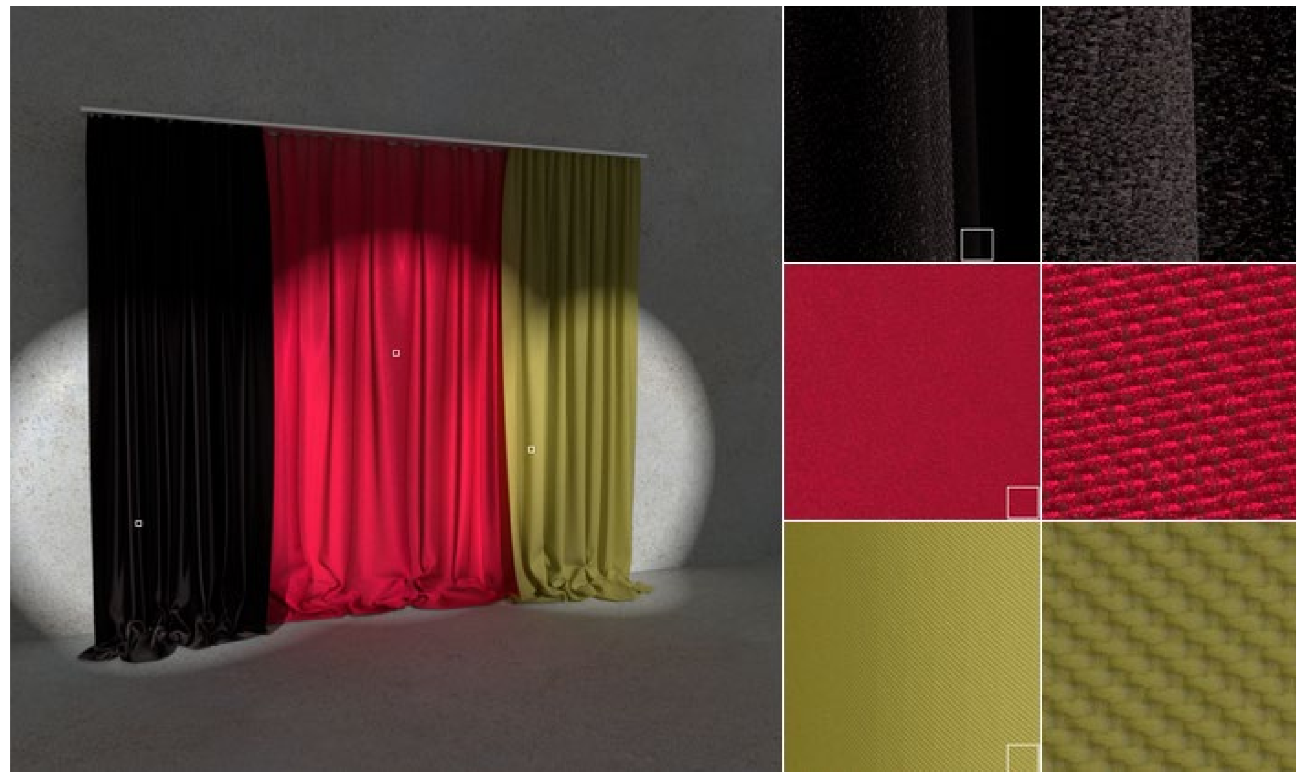
industrial dispersions
[Gkioulekas et al. 2013]



efficient algorithms
[Nimier-David et al. 2019, 2020]

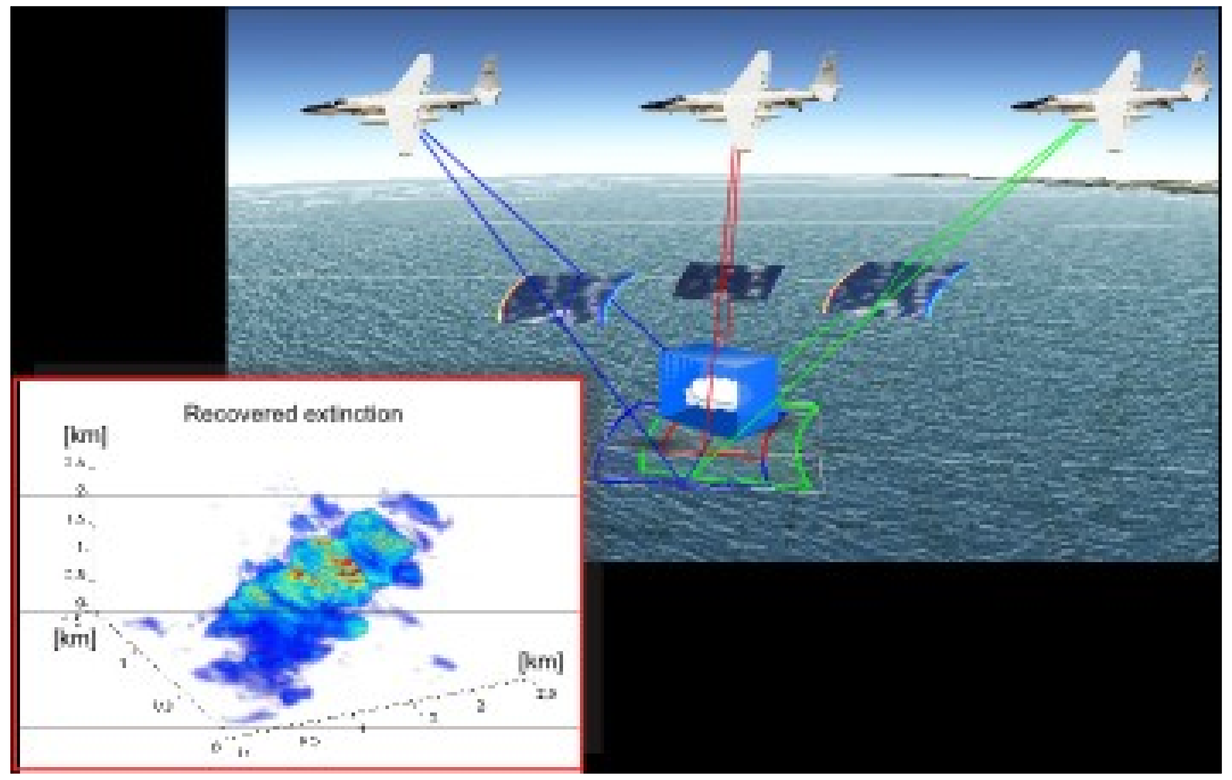


computed tomography
[Geva et al. 2018]



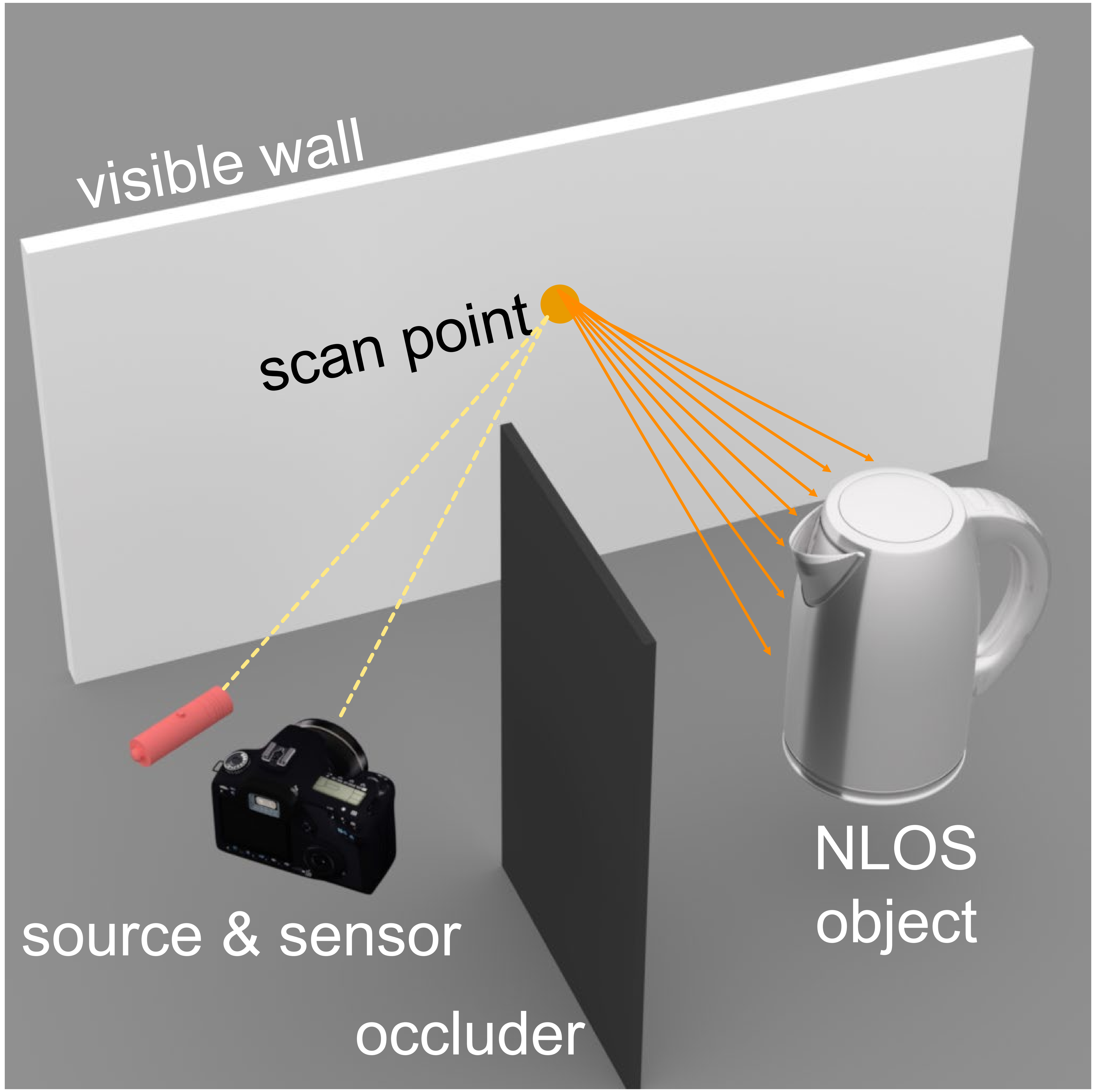
woven fabrics
[Khungurn et al. 2015,
Zhao et al. 2016]

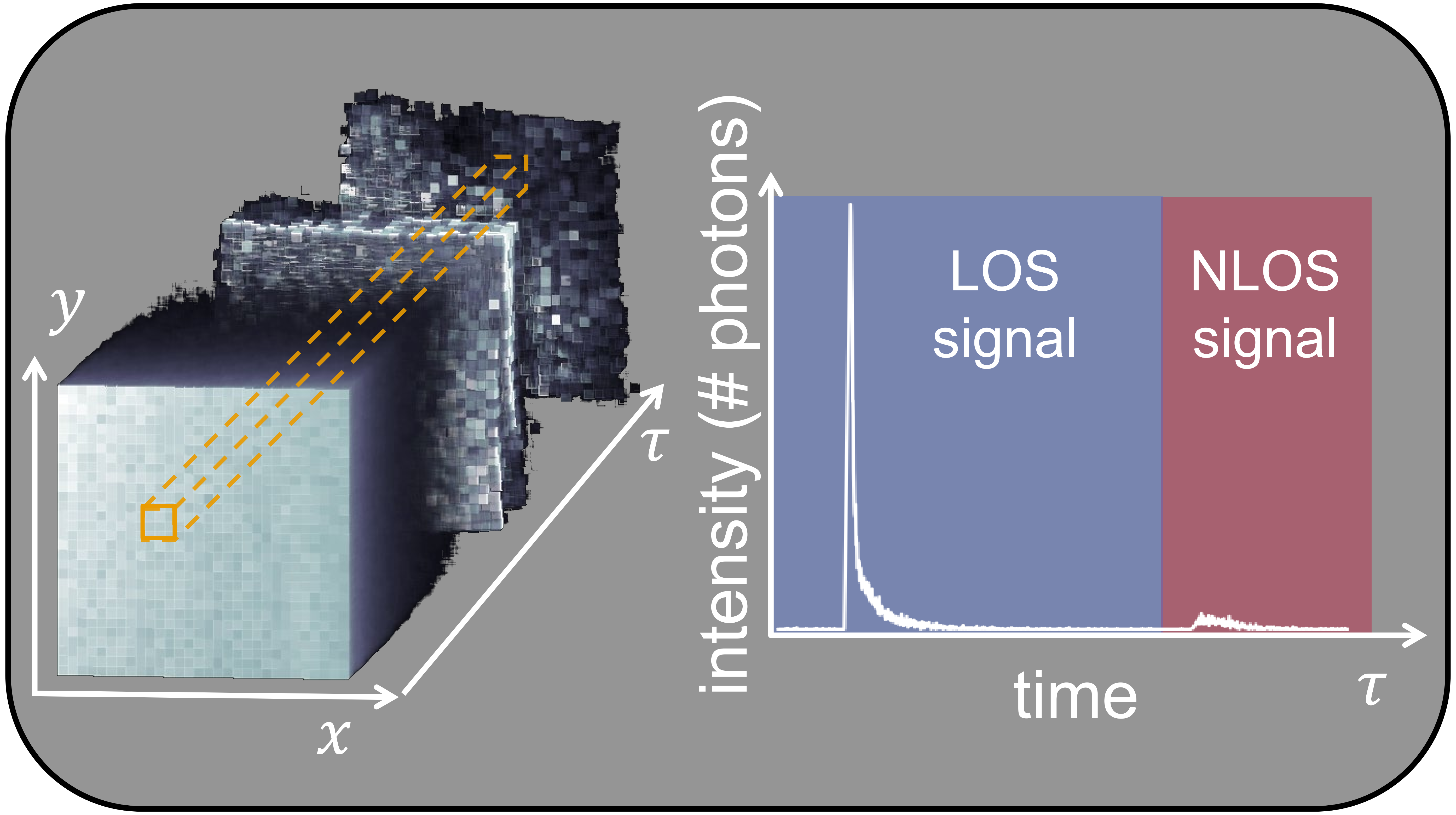
3D printing
[Elek et al. 2019,
Nindel et al. 2021]



cloud tomography
[Levis et al. 2015,
2017, 2020]

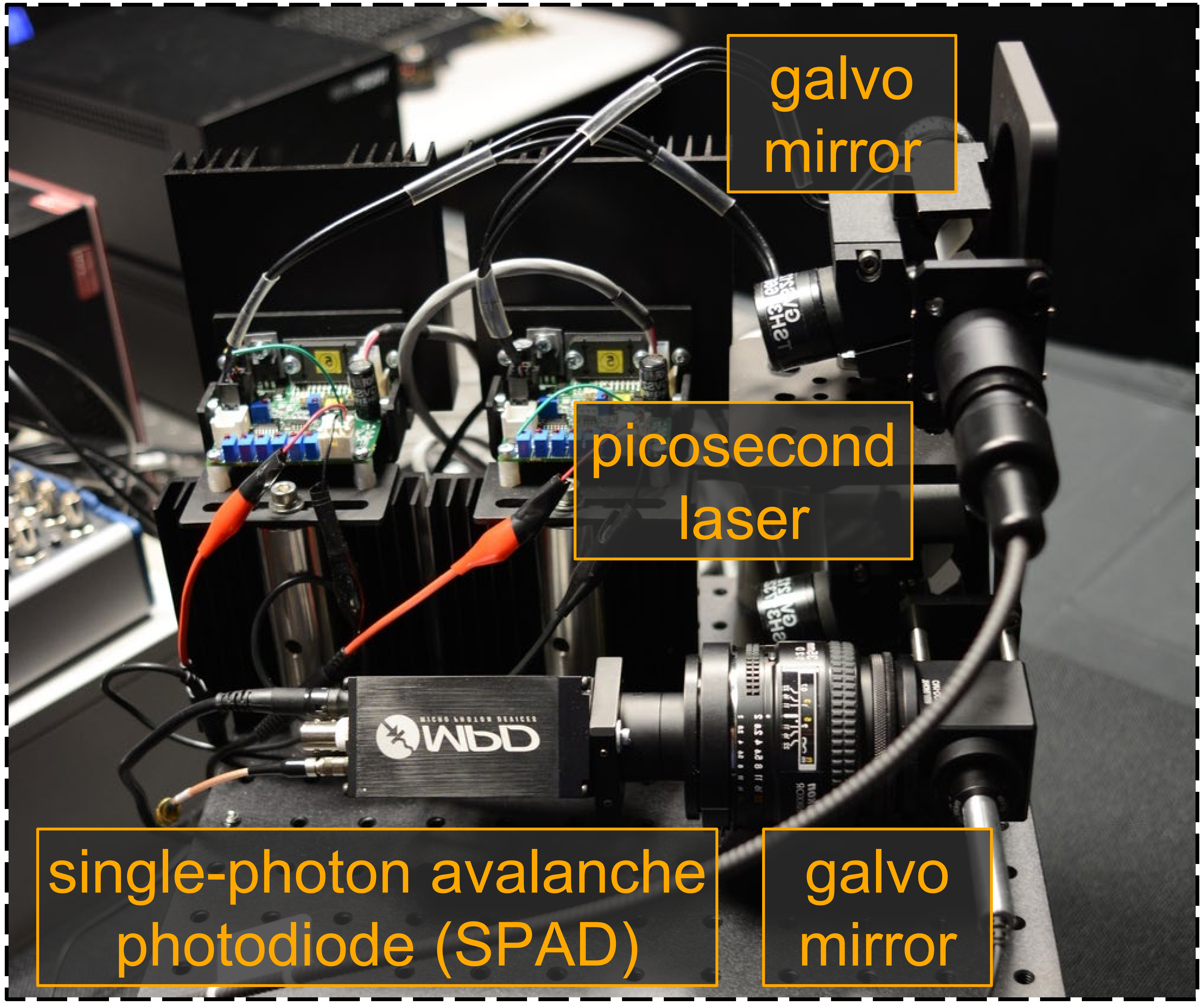
Non-line-of-sight (NLOS) imaging

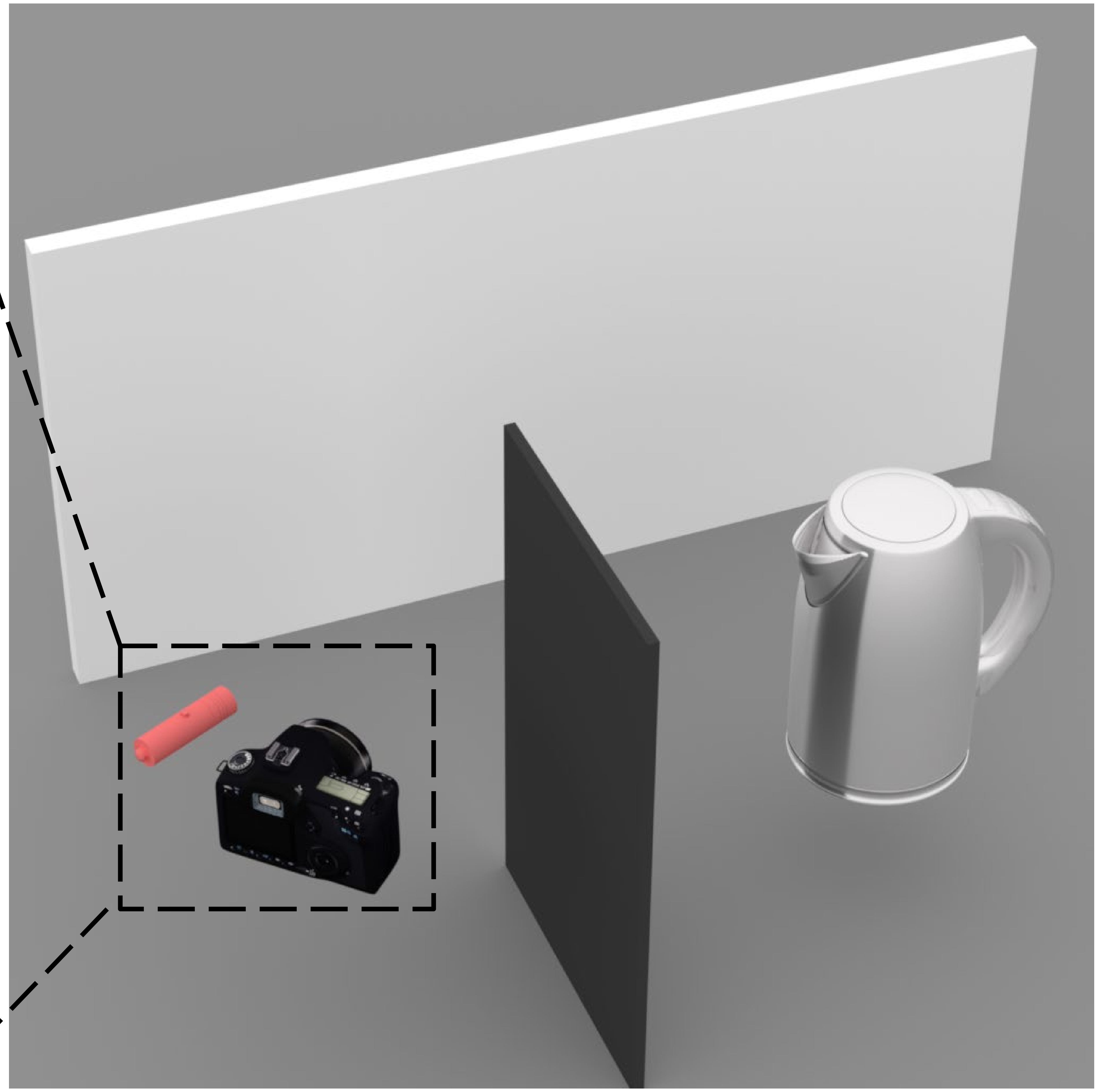




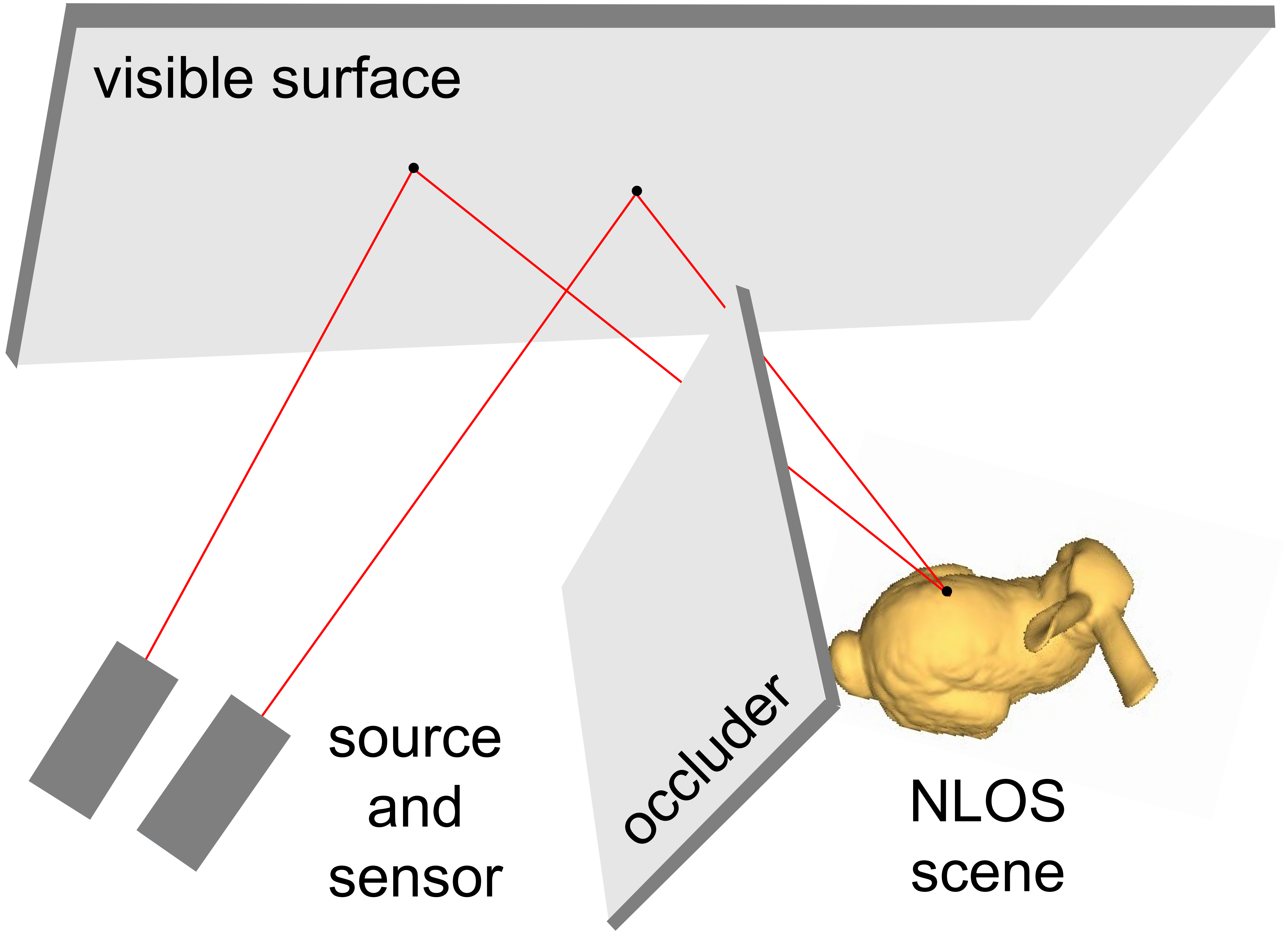
Time-of-flight measurements

SPAD-based lidar

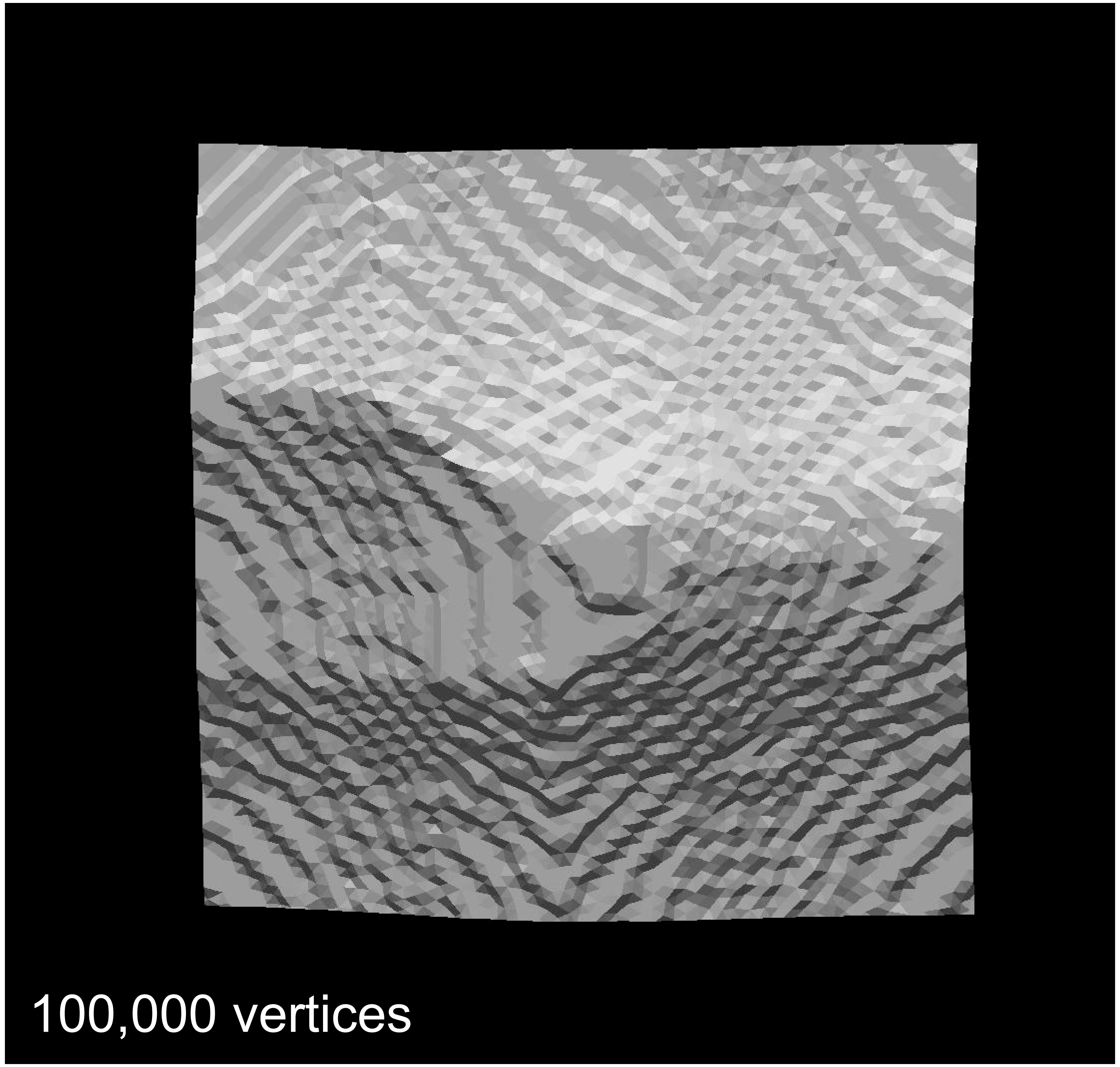




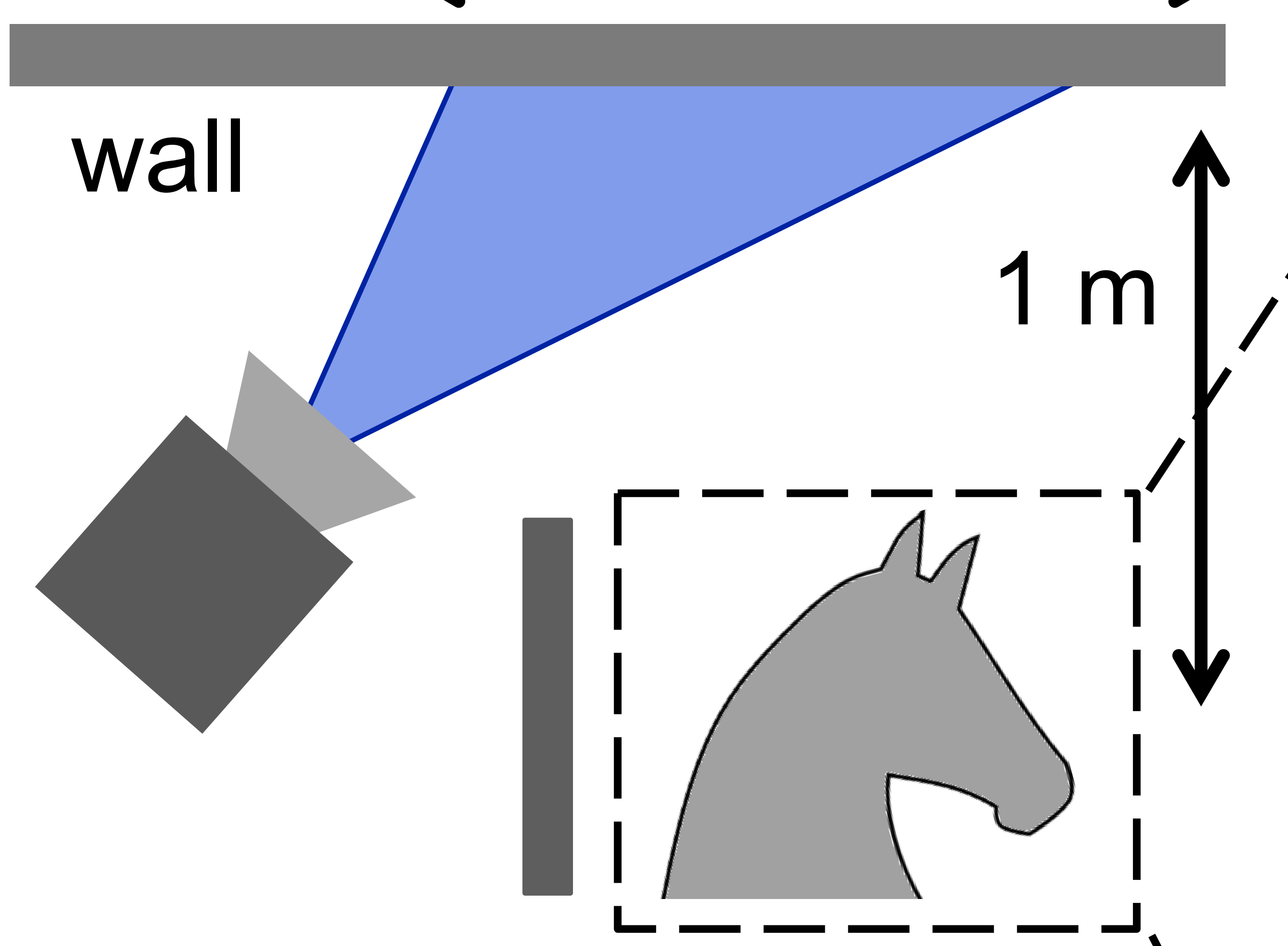
NLOS shape optimization [Tsai et al. 2019]



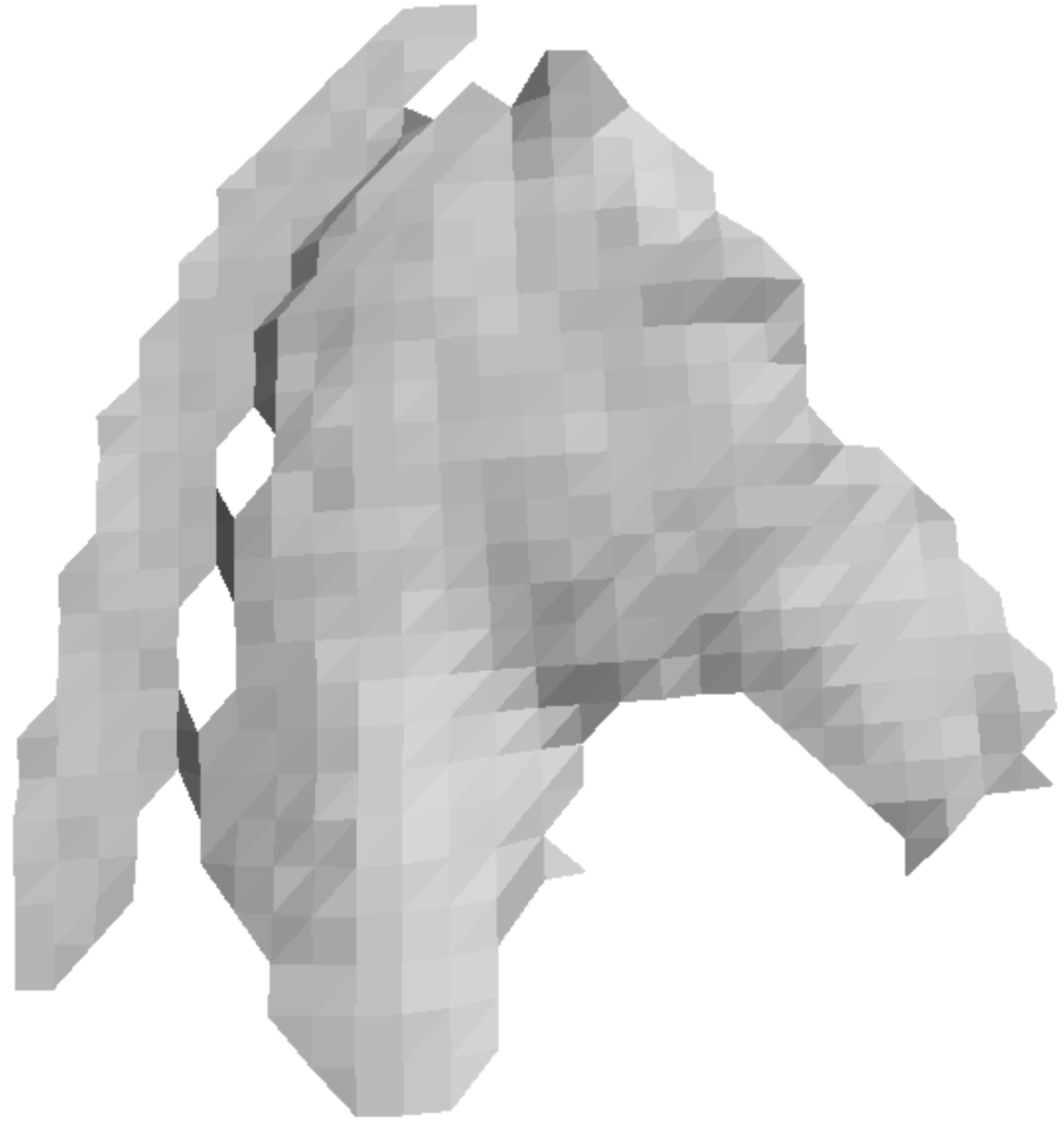
Simulated time-of-flight data



NLOS shape optimization [Tsai et al. 2019]



scene



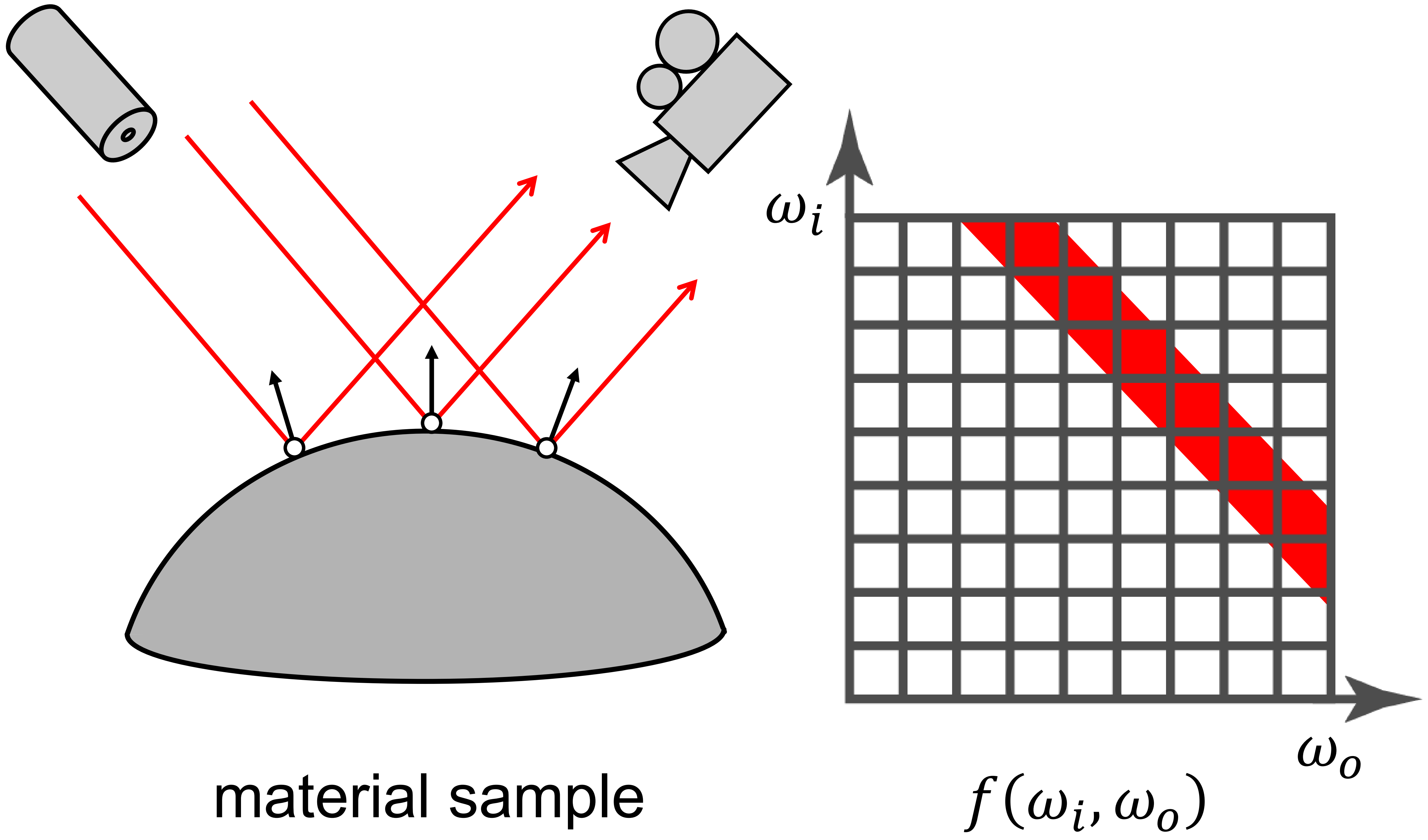
initial mesh
[O'Toole et al. 2018]

optimized mesh

Measured time-of-flight data

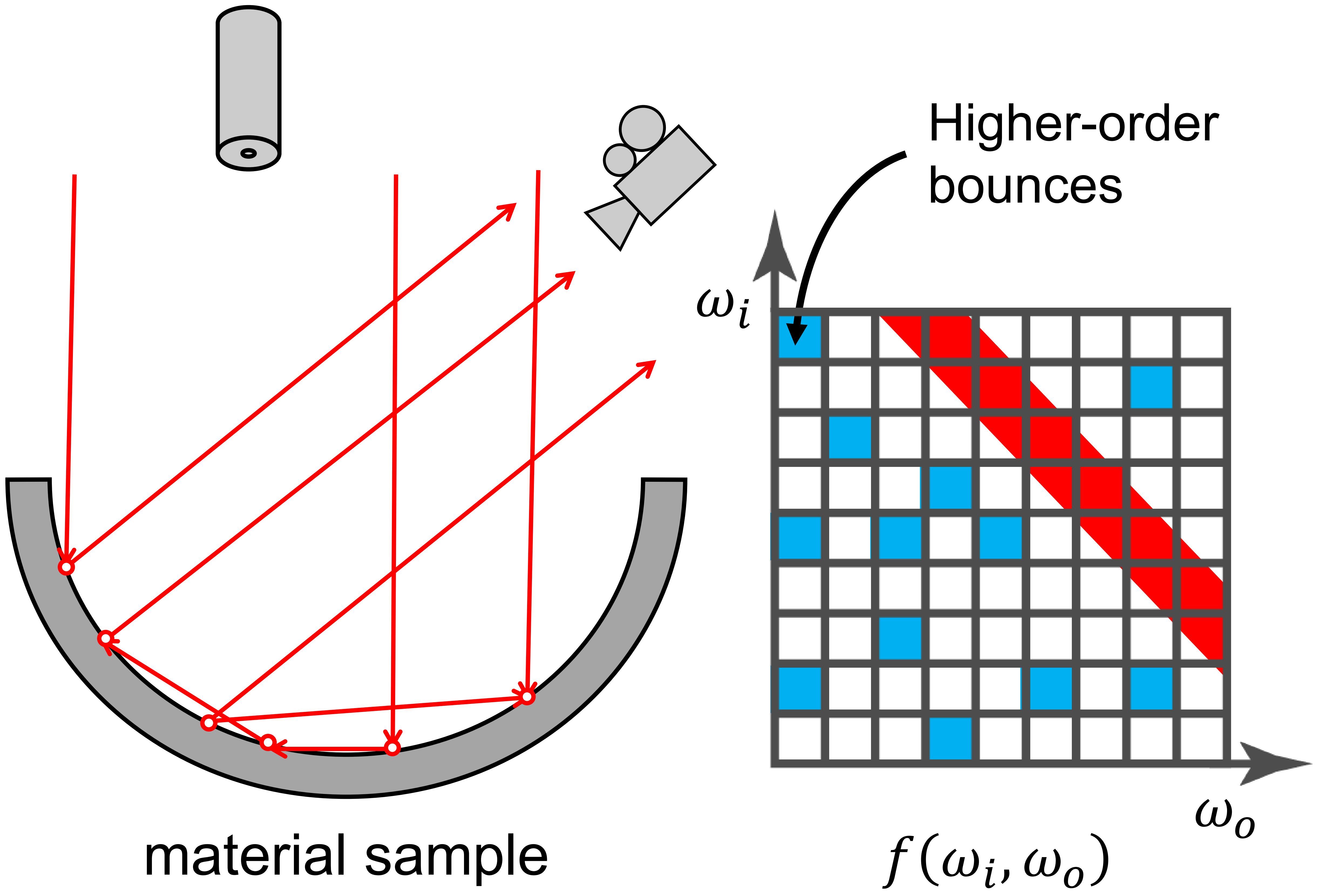
Reflectometry from interreflections [Shem-Tov et al. 2020]

Direct illumination measurements



+ Intensities map directly to BRDF entries
- Many measurements (2D scan of light & camera)

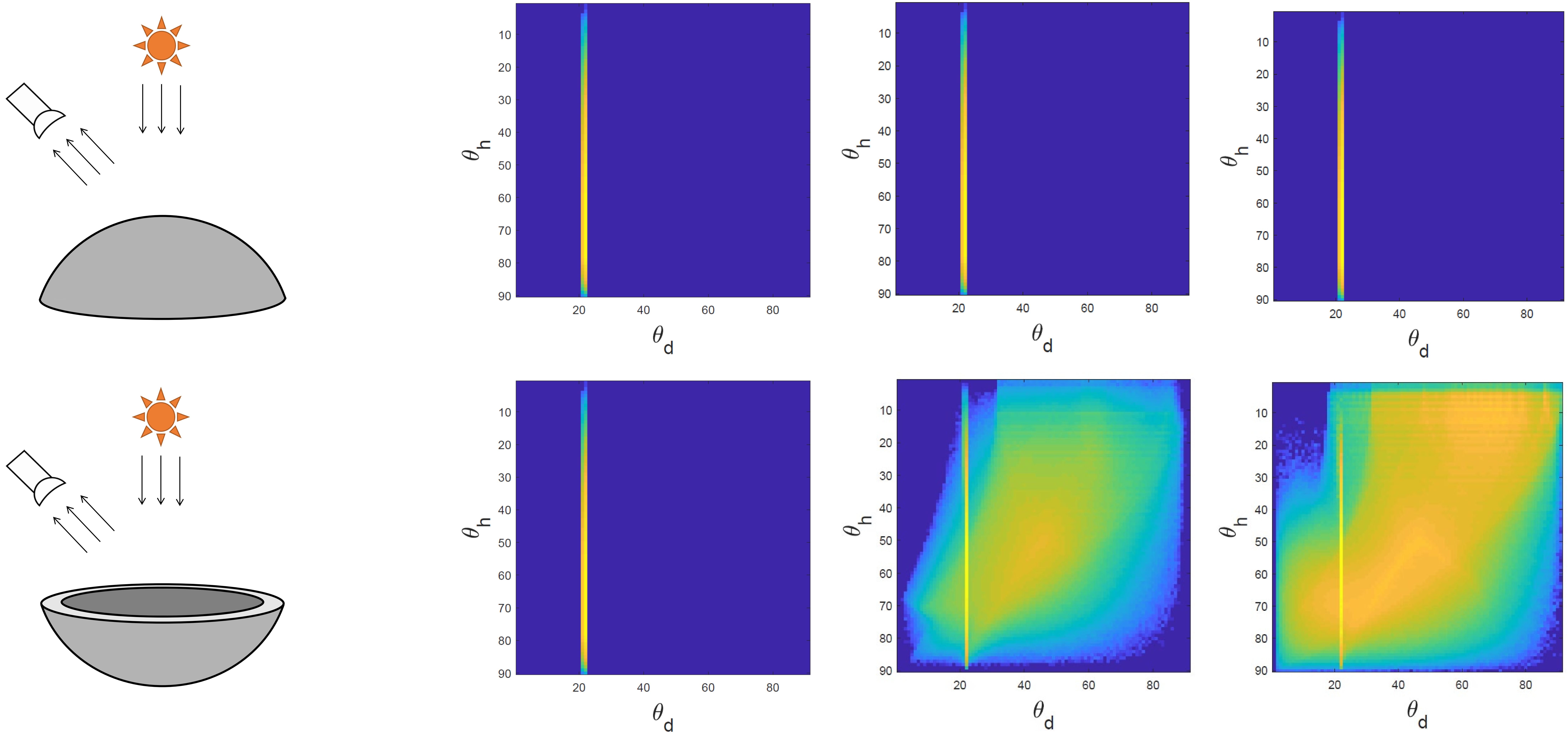
Global illumination measurements



+ Fewer measurements (single image)
- Non-linear analysis-by-synthesis optimization

Solvable using differentiable rendering

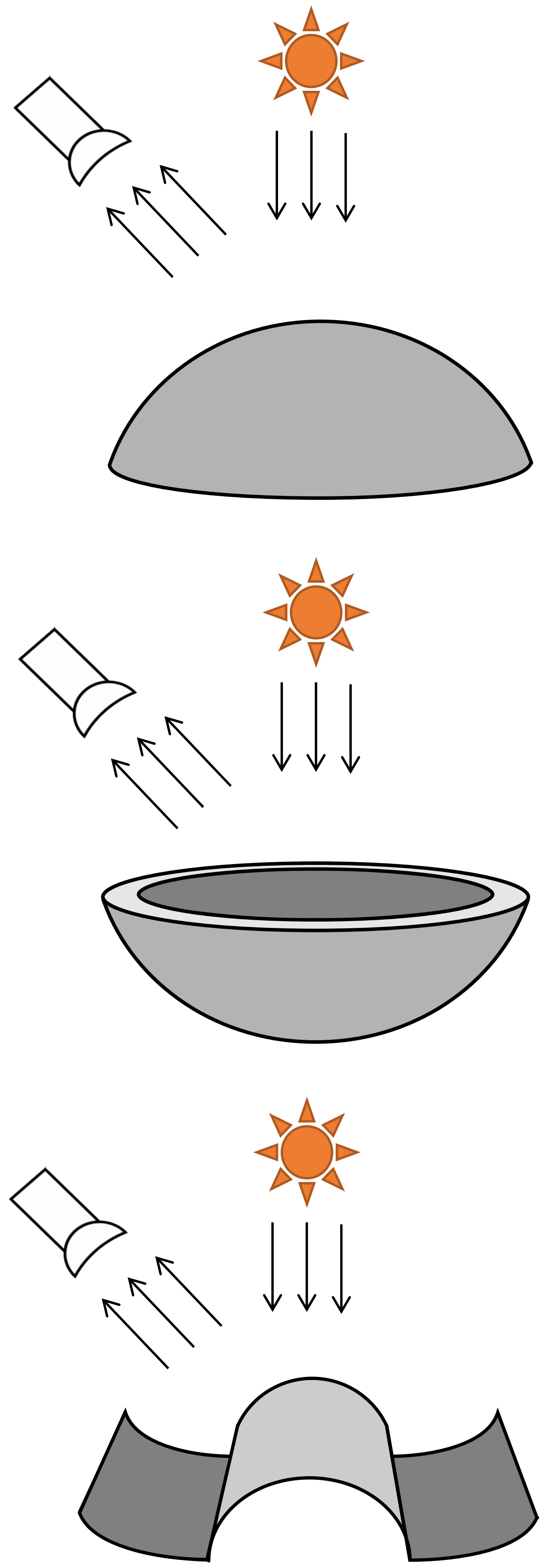
Single-image dense BRDF sampling



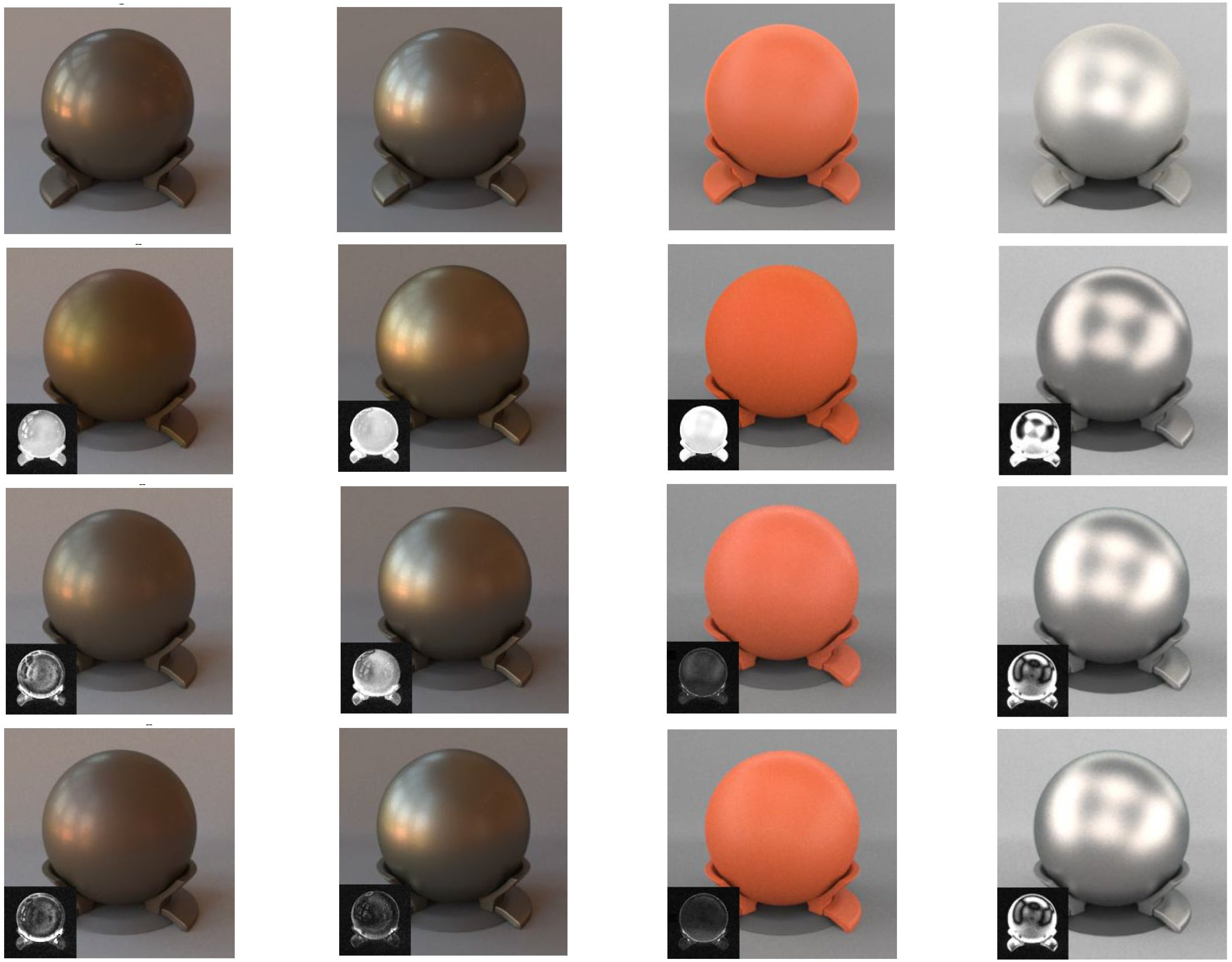
Single-bounce paths Two-bounce paths All-bounce paths

Results on MERL dataset

Groundtruth



Optimized shape

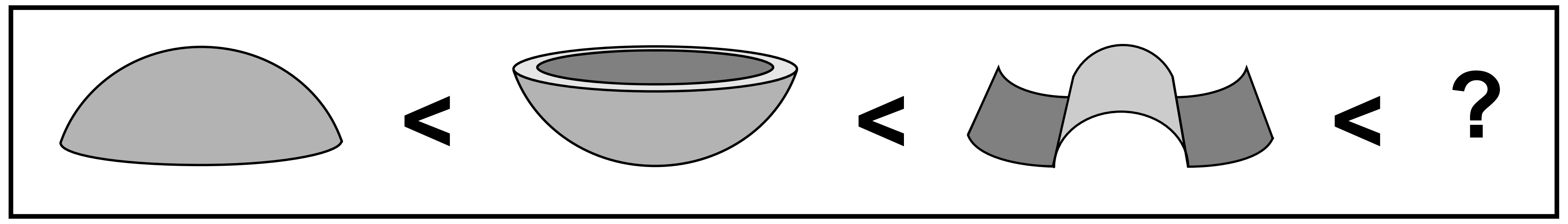


~ 6.3x better parameter recovery

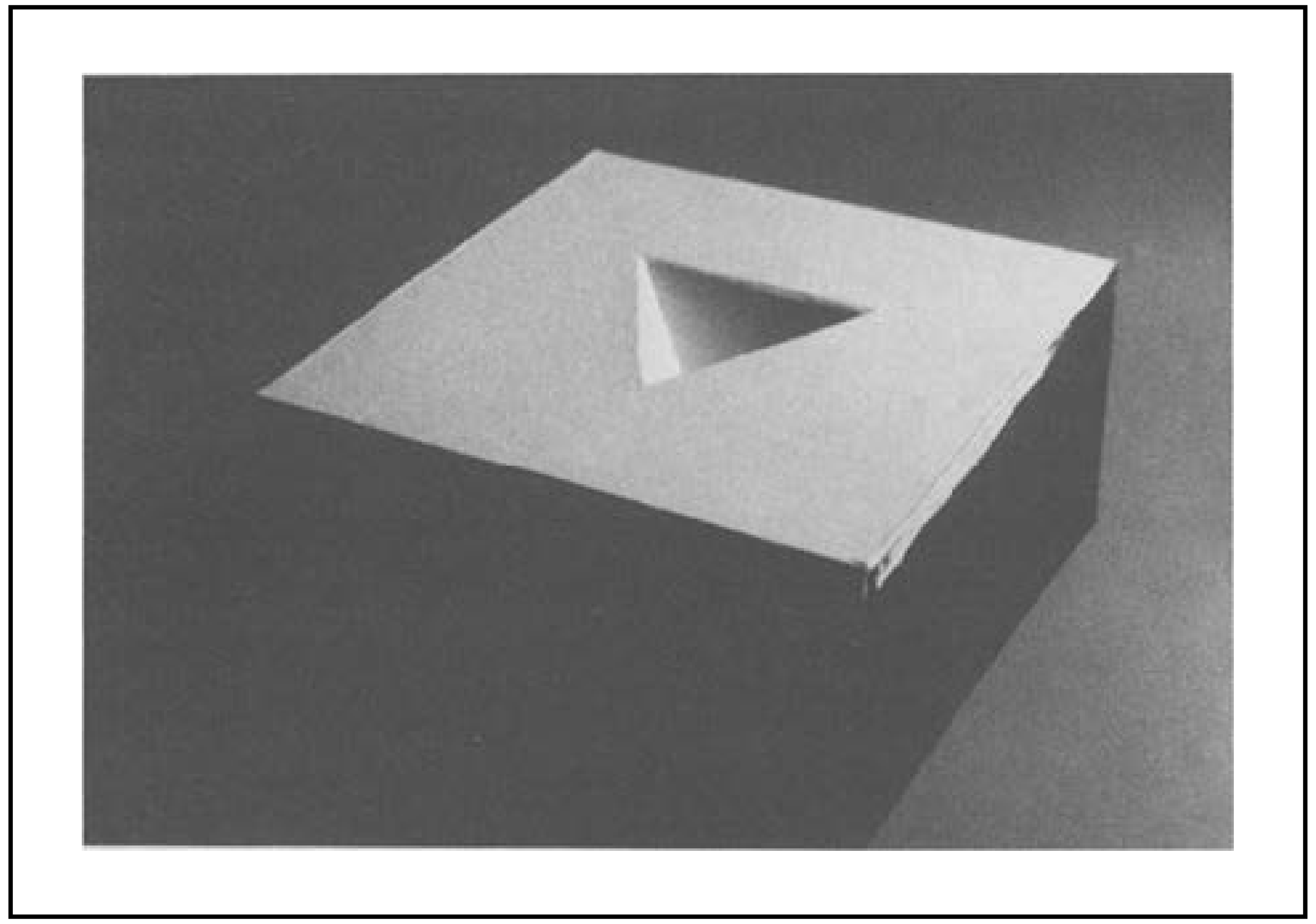
~ 11.2x better parameter recovery

Global illumination can help…

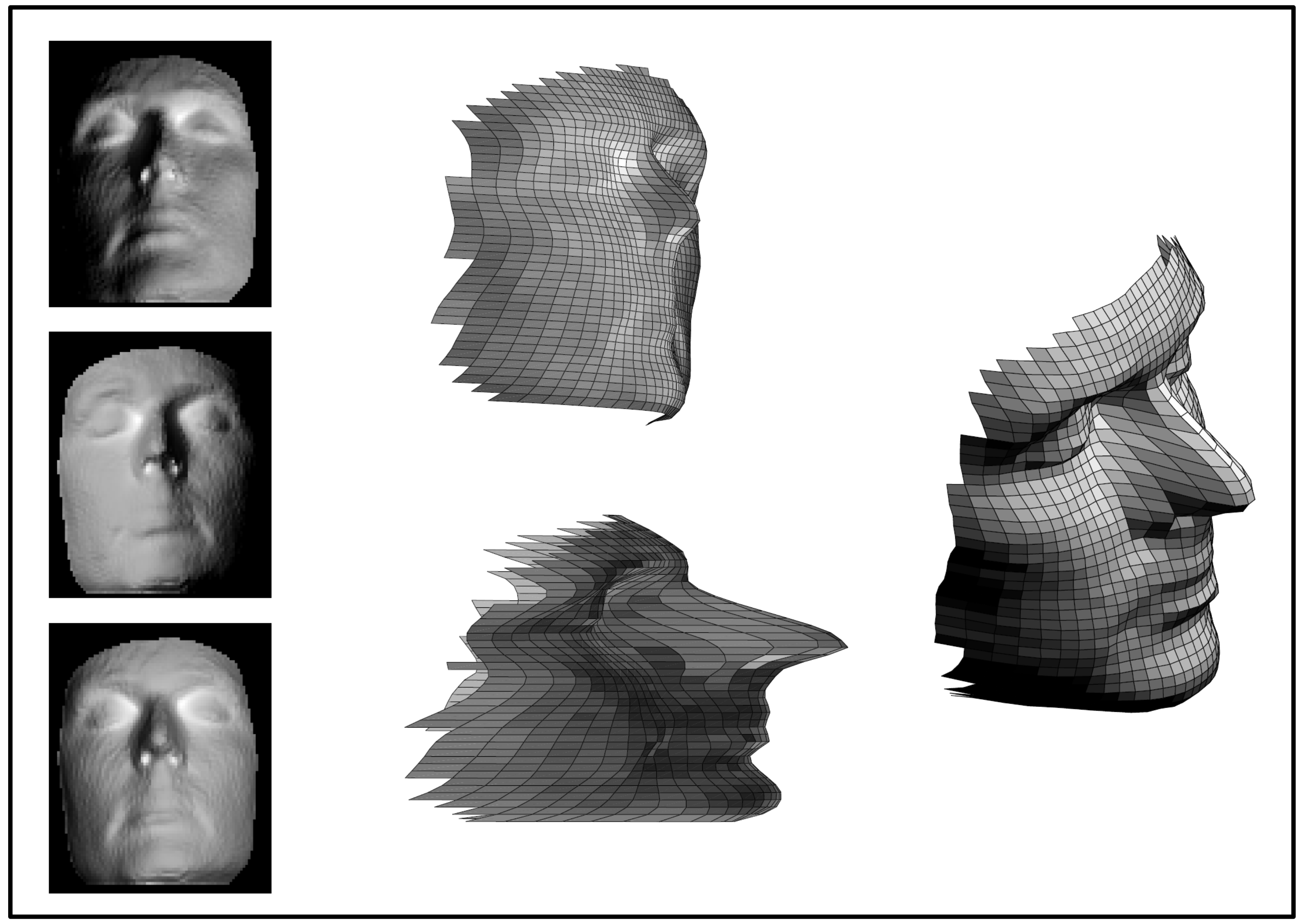
- Reduce number of measurements required for inverse rendering
  - We should rethink "optimal" acquisition systems



- Resolve ambiguities between different types of parameters
  - We should revisit theory problems on uniqueness results



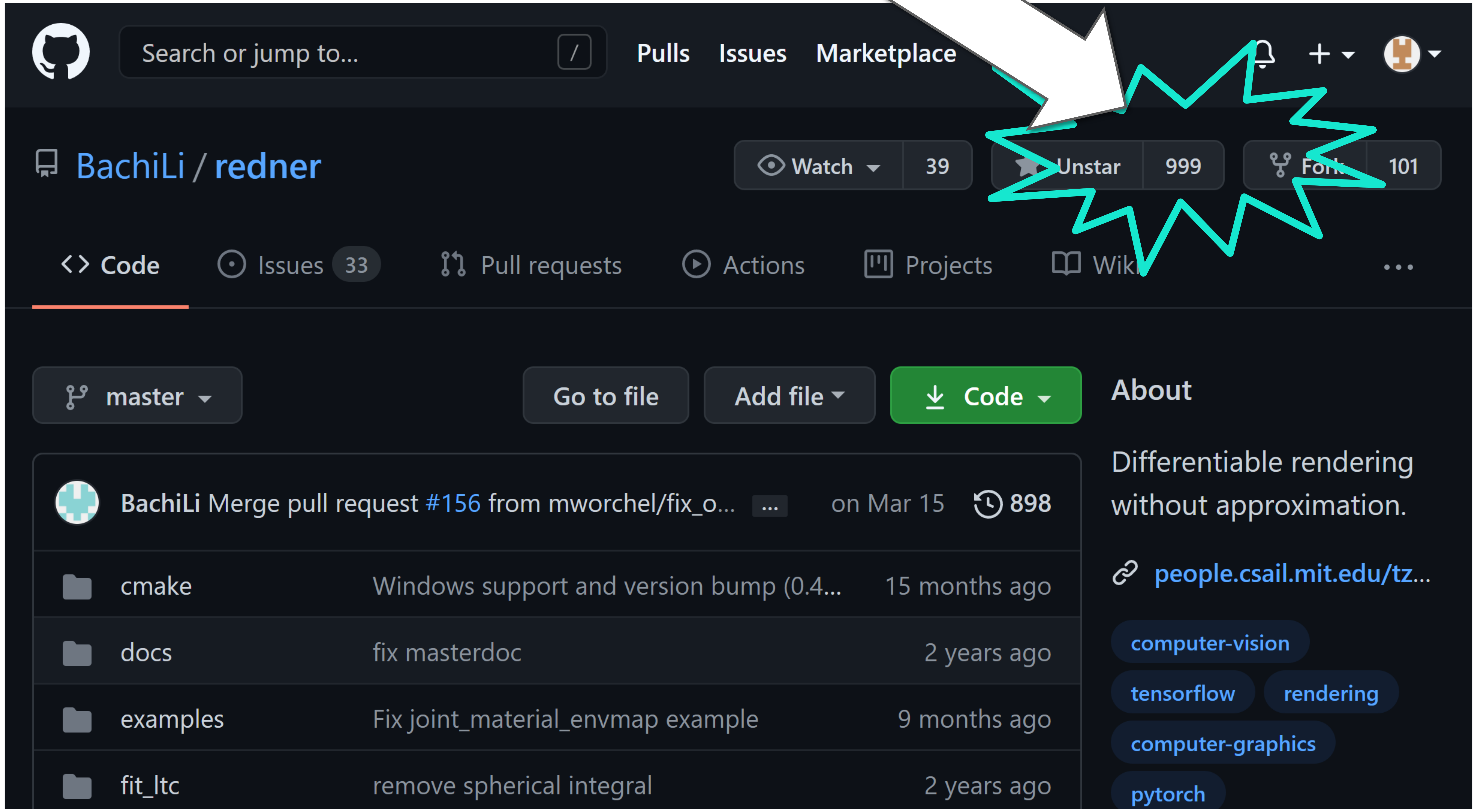
Shape from interreflections
[Nayar et al. 1990, Marr Prize]

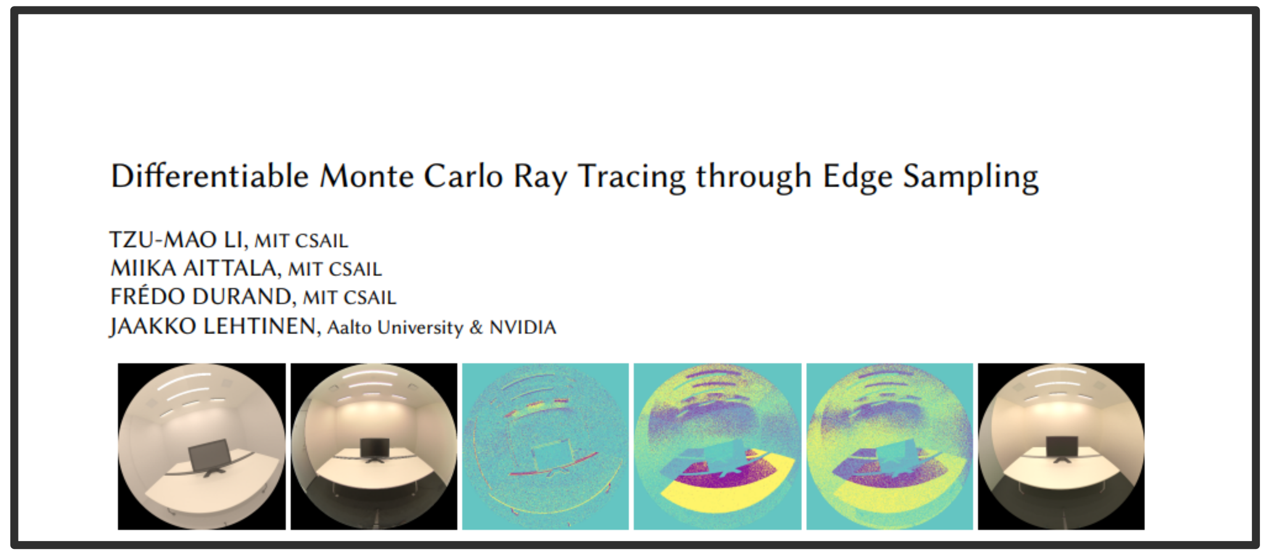


Interreflections resolve the GBR ambiguity
[Chandraker et al. 2005]

The first physically-based differentiable renderer!

And also among the most widely used:





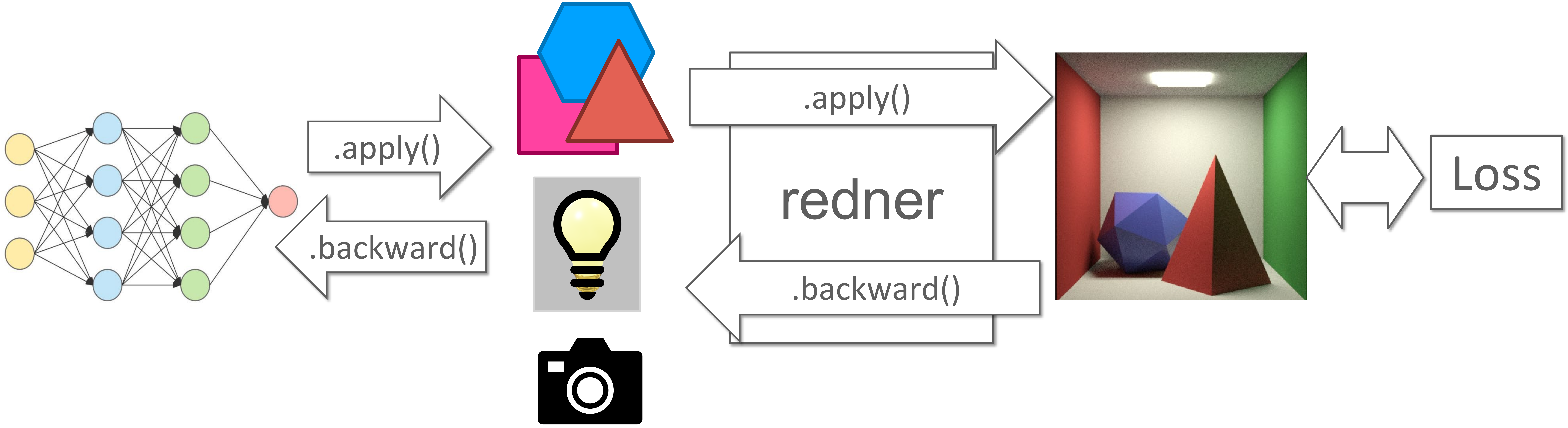
Introduced with the first unbiased differentiable rendering algorithm [Li. 2018]

Redner is built from the ground up for ML applications

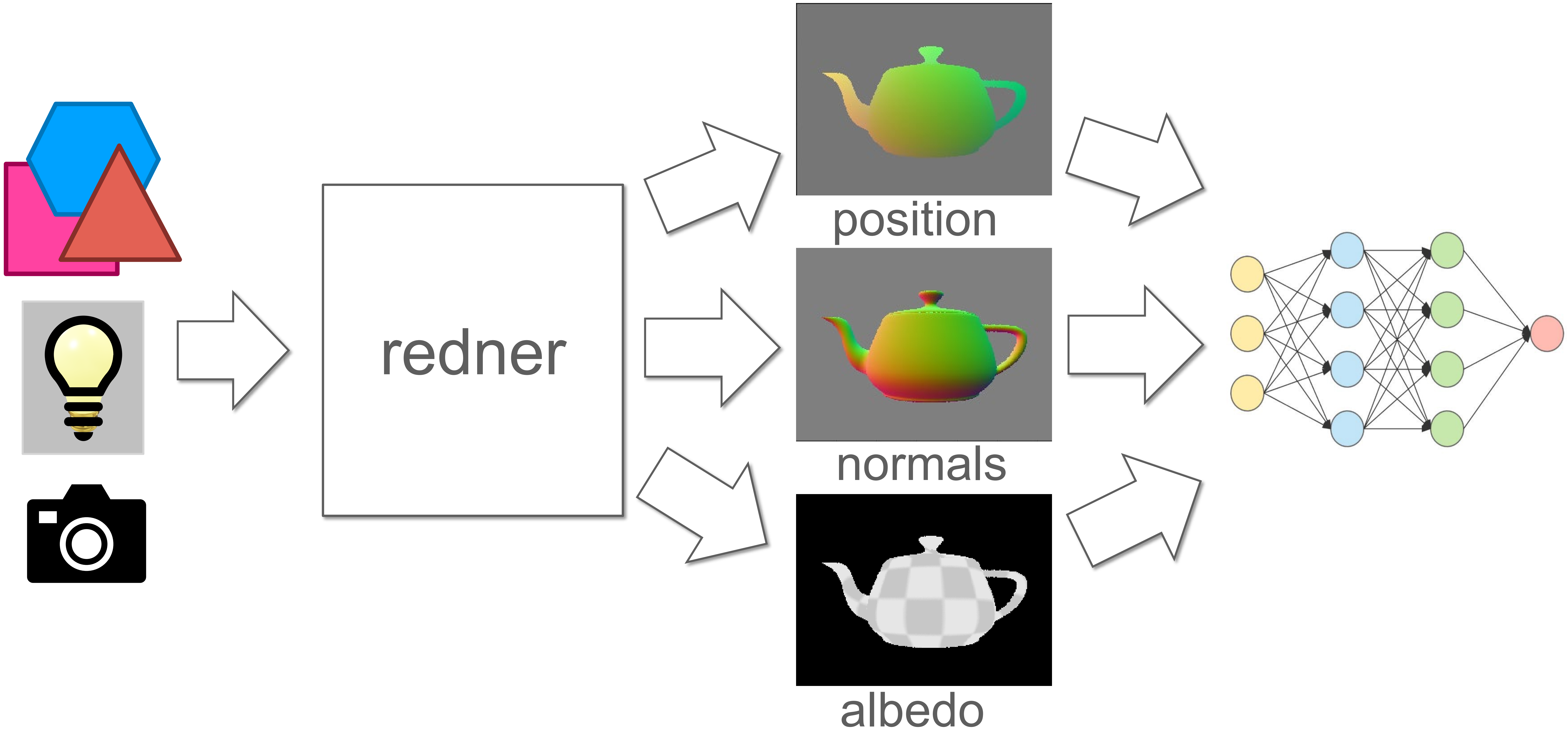
PyTorch

TensorFlow

("Eager" mode only)



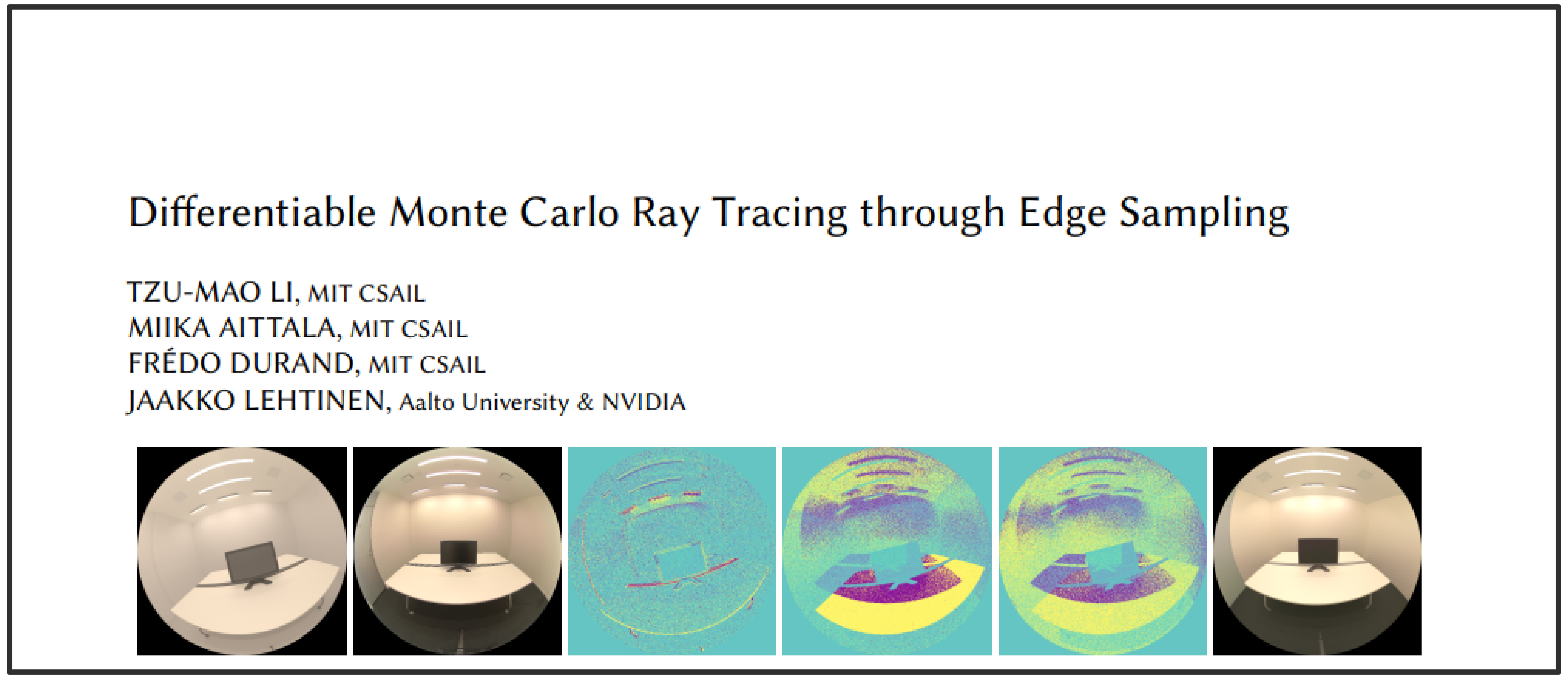
Redner supports *deferred* rendering (if realistic rendering is not the goal)



(say we can just use one bounce lighting)

Redner now contains *two* styles of differentiable rendering

`pyredner.integrators.EdgeSamplingIntegrator()`

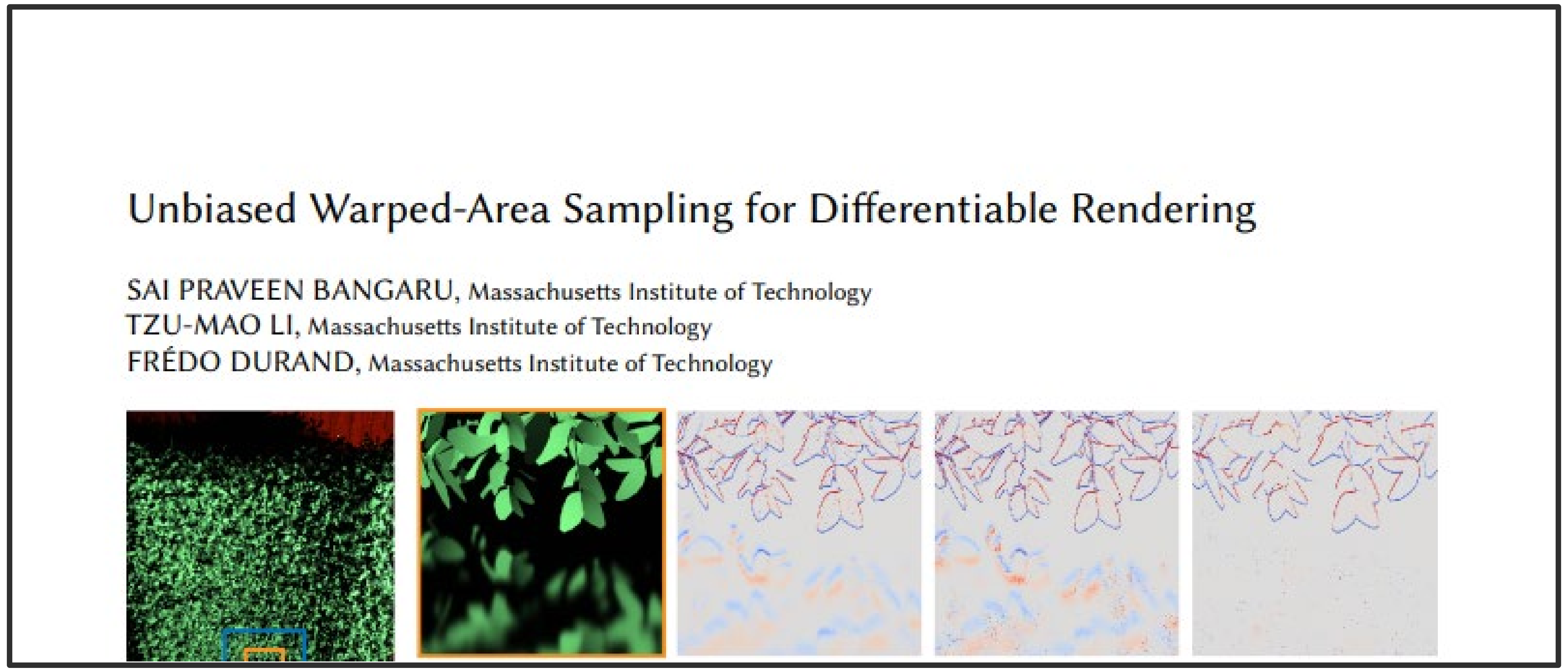


Differentiable Monte Carlo Ray Tracing through Edge Sampling

TZU-MAO LI, MIT CSAIL
MIIKA AITTALA, MIT CSAIL
FRÉDO DURAND, MIT CSAIL
JAAKKO LEHTINEN, Aalto University & NVIDIA

Edge-sampling *[Li et al. 2018]*

`pyredner.integrators.WarpFieldIntegrator()`



Unbiased Warped-Area Sampling for Differentiable Rendering

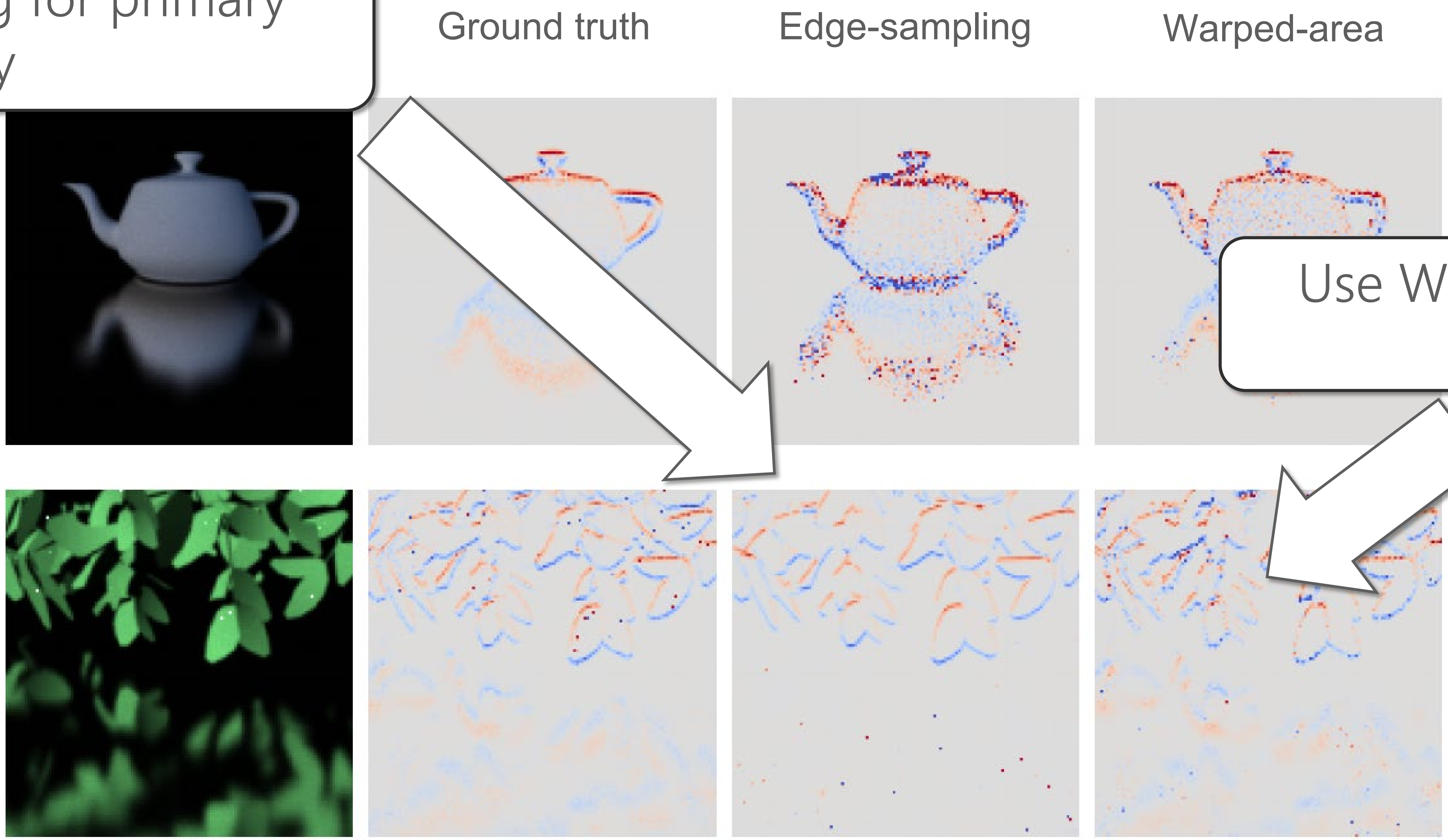
SAI PRAVEEN BANGARU, Massachusetts Institute of Technology
TZU-MAO LI, Massachusetts Institute of Technology
FRÉDO DURAND, Massachusetts Institute of Technology

Warped-area sampling *[Bangaru et al. 2020]*

Swap freely between two methods

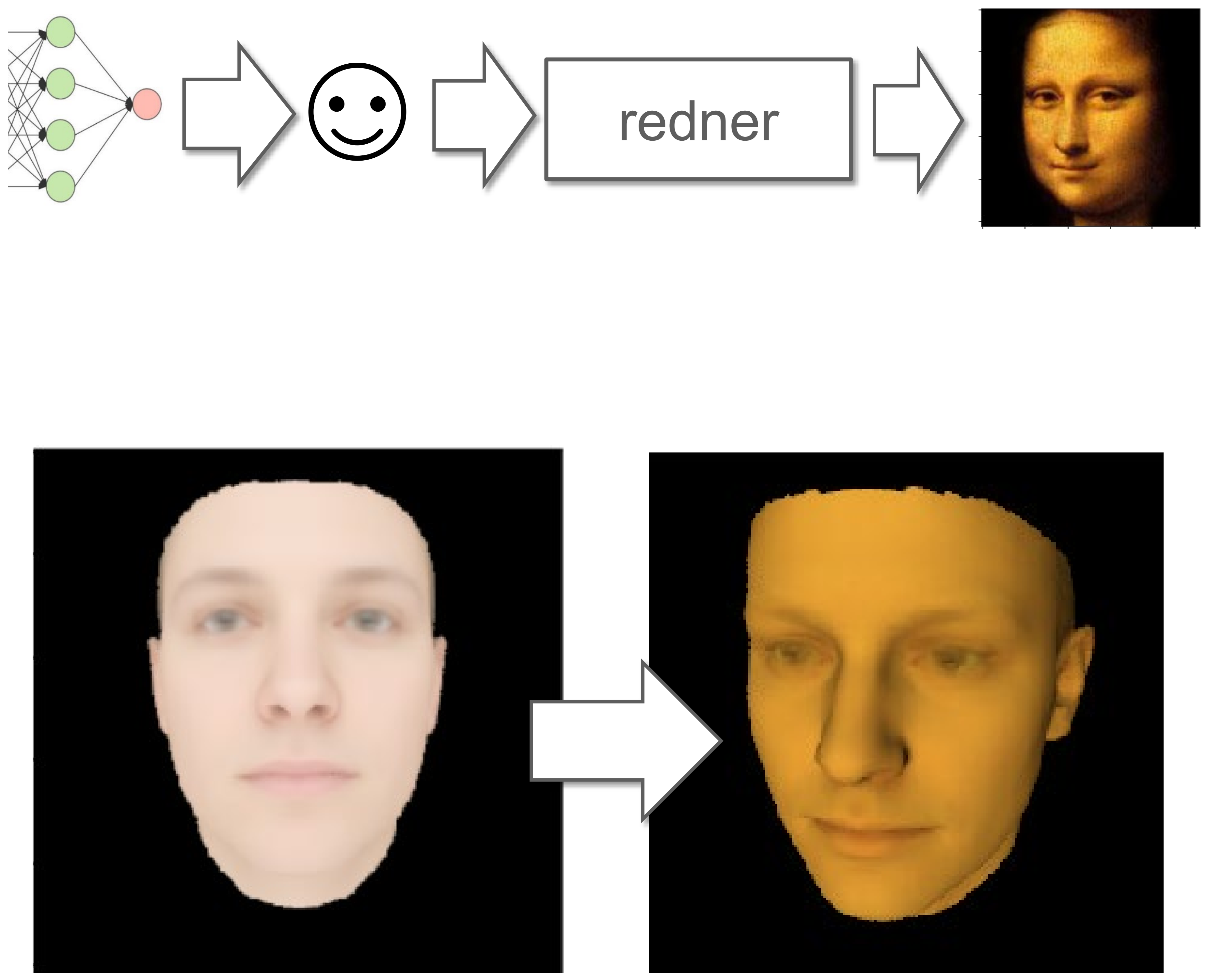
`pyredner.integrators.EdgeSamplingIntegrator()` OR `pyredner.integrators.WarpFieldIntegrator()`

Use edge-sampling for primary visibility

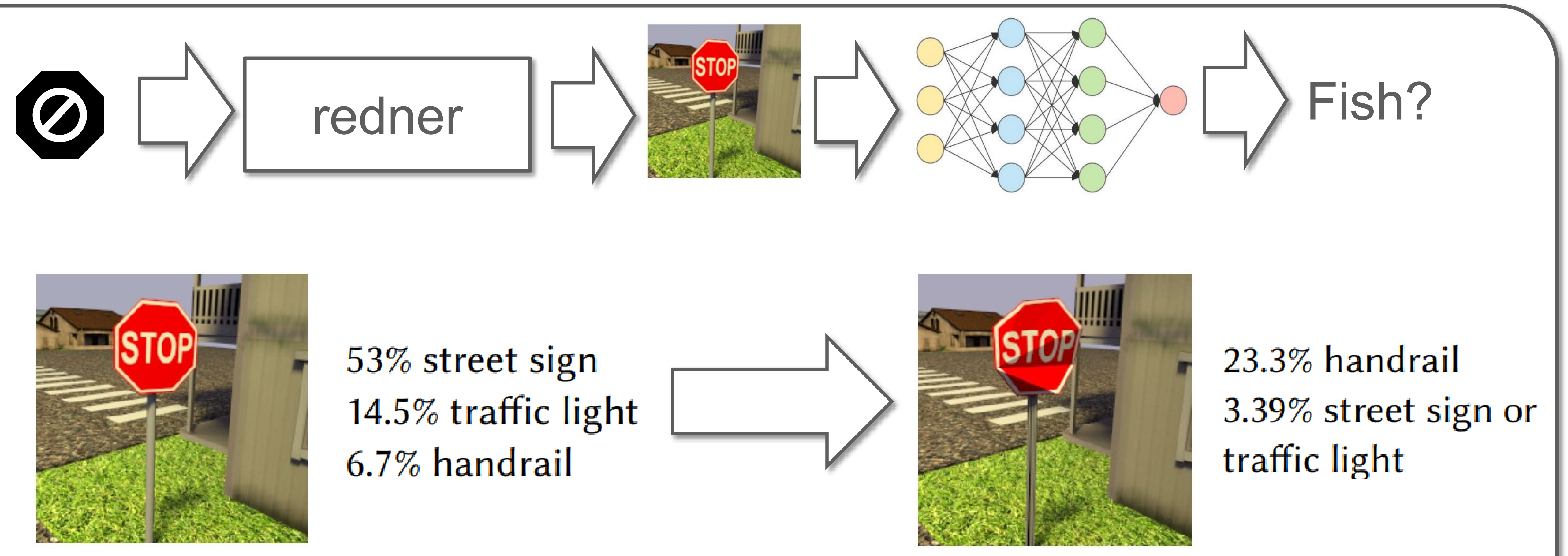


Use WAS for higher-order effects

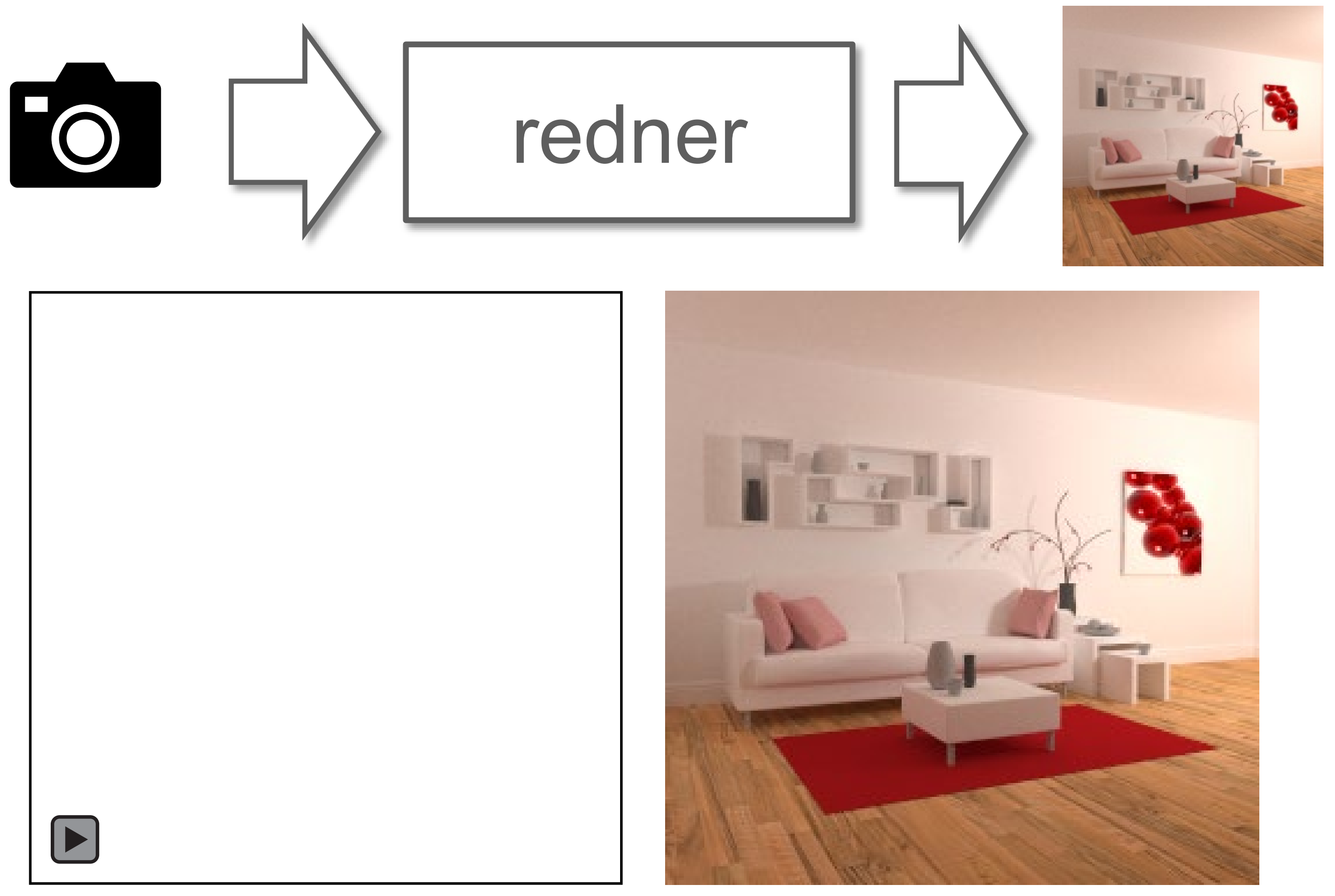
Use redner anywhere in your pipeline



Optimize/Train morphable models



Find adversarial examples



Optimize for fine-grained pose

Use Redner today!

SCAN ME

Github codebase

Start using on your system:

```
$pip install redner
```

```
$pip install redner-gpu
```

Or on Google Colab immediately:

SCAN ME

Sample notebook

Summary

Take-Home Messages

- Great progress has been made in physics-based **differentiable** rendering
  - Now capable of handling global illumination, arbitrary types of camera (e.g., transient), and global scene parameters (e.g., object geometry) with decent efficiency
  - Can be applied to solve many general inverse problems
- **Ray tracing** is no longer slow
  - Many efficient systems are being actively developed (e.g., Redner, PSDR-CUDA, Mitsuba 2, Teg)
  - And differentiable rendering is usually not the performance bottleneck
- **Gradient accuracy** matters!
  - Approximated gradients can yield reduced result quality
- **Discontinuities** always exist (due to visibility) and need to be properly handled
  - Auto-diffing a path tracer may not always work

Thank you!

Funding agencies

Tutorial website

https://diff-render.org