SIGGRAPH 2020 S2020.SIGGRAPH.ORG THINK BEYOND

PHYSICS-BASED DIFFERENTIABLE RENDERING

A COMPREHENSIVE INTRODUCTION

Shuang Zhao — University of California, Irvine
Wenzel Jakob — EPFL, Lausanne, Switzerland
Tzu-Mao Li — MIT CSAIL, Cambridge

PRESENTERS

SHUANG ZHAO
ASSISTANT PROFESSOR
University of California, Irvine

UCI

WENZEL JAKOB
ASSISTANT PROFESSOR
EPFL, Lausanne, Switzerland

EPFL

TZU-MAO LI
POSTDOCTORAL RESEARCHER
MIT CSAIL, Cambridge
(joining UCSD as an assistant professor in 2021)

FORWARD VS. INVERSE RENDERING

Rendering

$$\mathbf{y} = f(\mathbf{x})$$

Inverse rendering

$$\mathbf{x} = f^{-1}(\mathbf{y})?$$

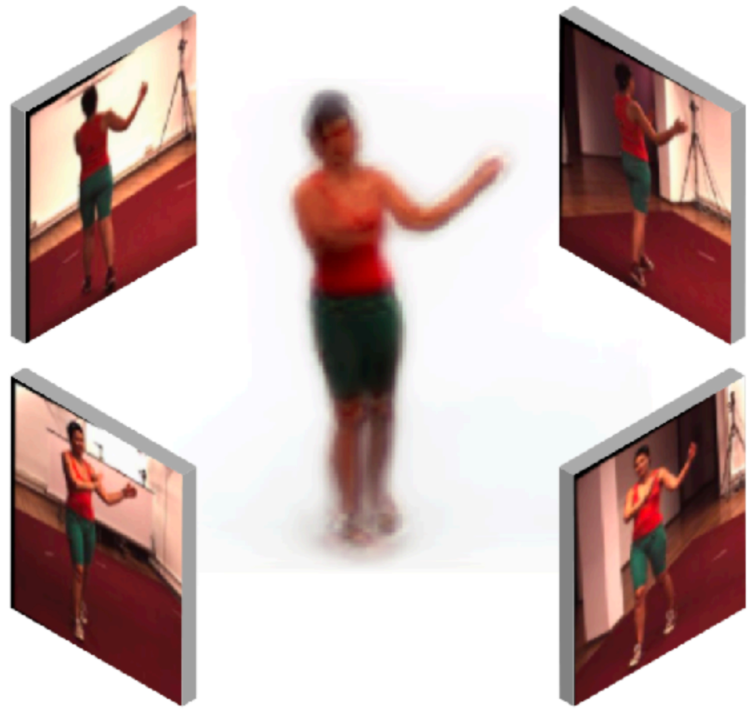
Geometry, materials, emitters, ...

Scene: "bed classic" from jiraniano

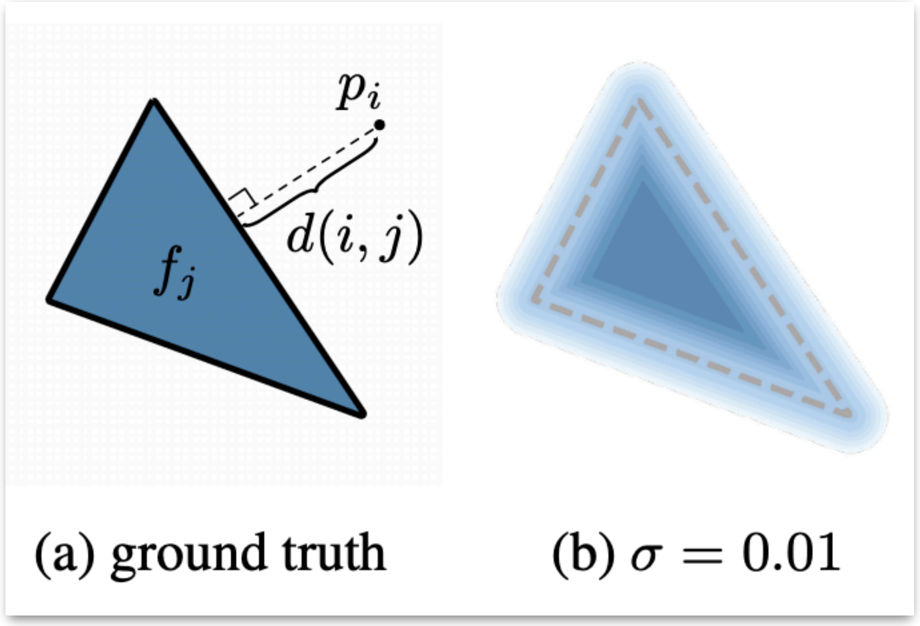
INVERSE RENDERING IN COMPUTER VISION

OpenDR: an Approximate Differentiable Renderer
[Loper et al. 2014]

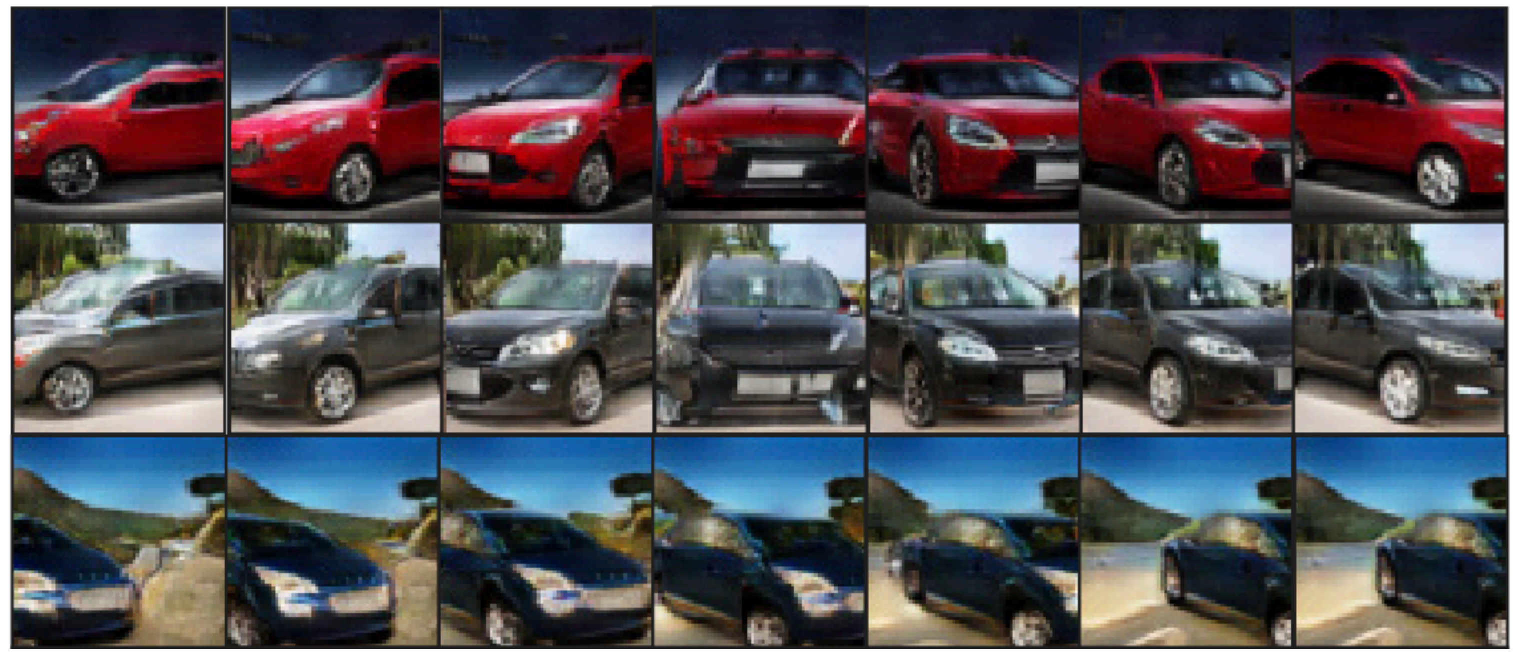
Neural 3D Mesh Renderer
[Kato et al. 2017]



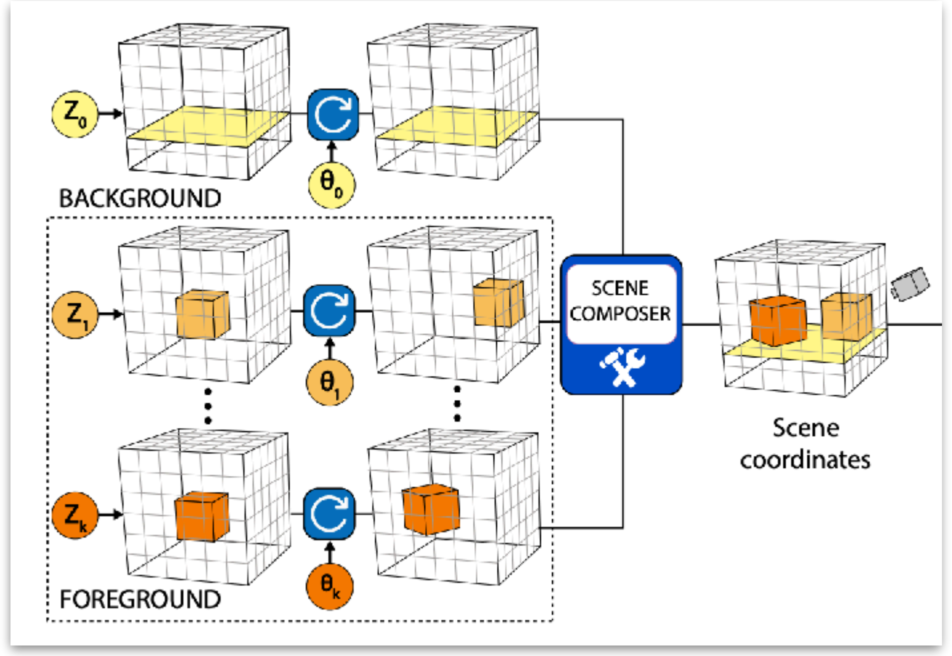
Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation
[Rhodin et al., 2016]



Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction
[Liu et al. 2019]



HoloGAN: Unsupervised Learning of 3D Representations From Natural Images.
[Nguyen-Phuoc et al. 2019]

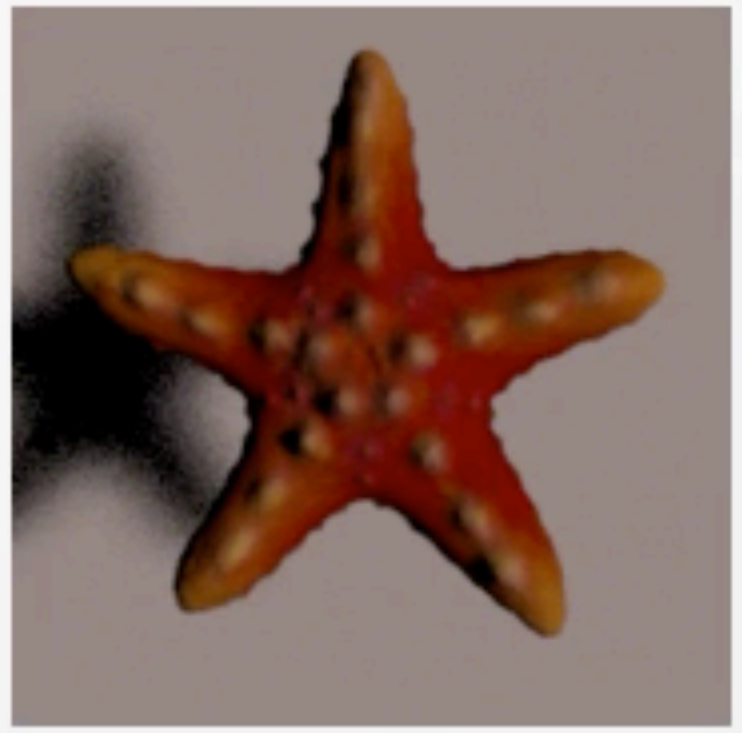


BlockGAN: Learning 3D Object-aware Scene Representations from Unlabelled Images
[Nguyen-Phuoc et al. 2020]

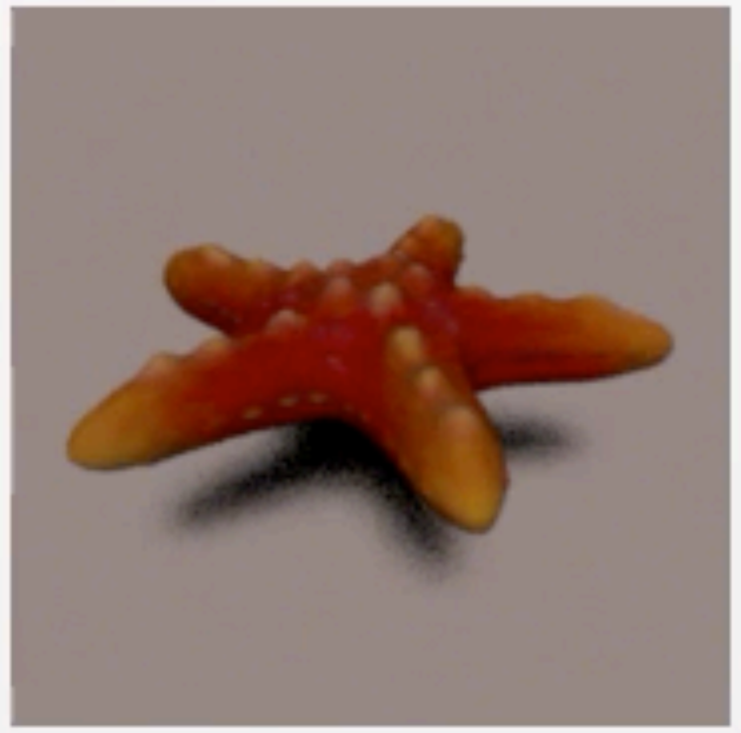
- **Focus on inverse rendering for realistic functions** $f(\mathbf{x})$

Global illumination, complex materials, participating media, polarization, color spectra, etc.

SHAPE & MATERIAL RECONSTRUCTION



Target



Target

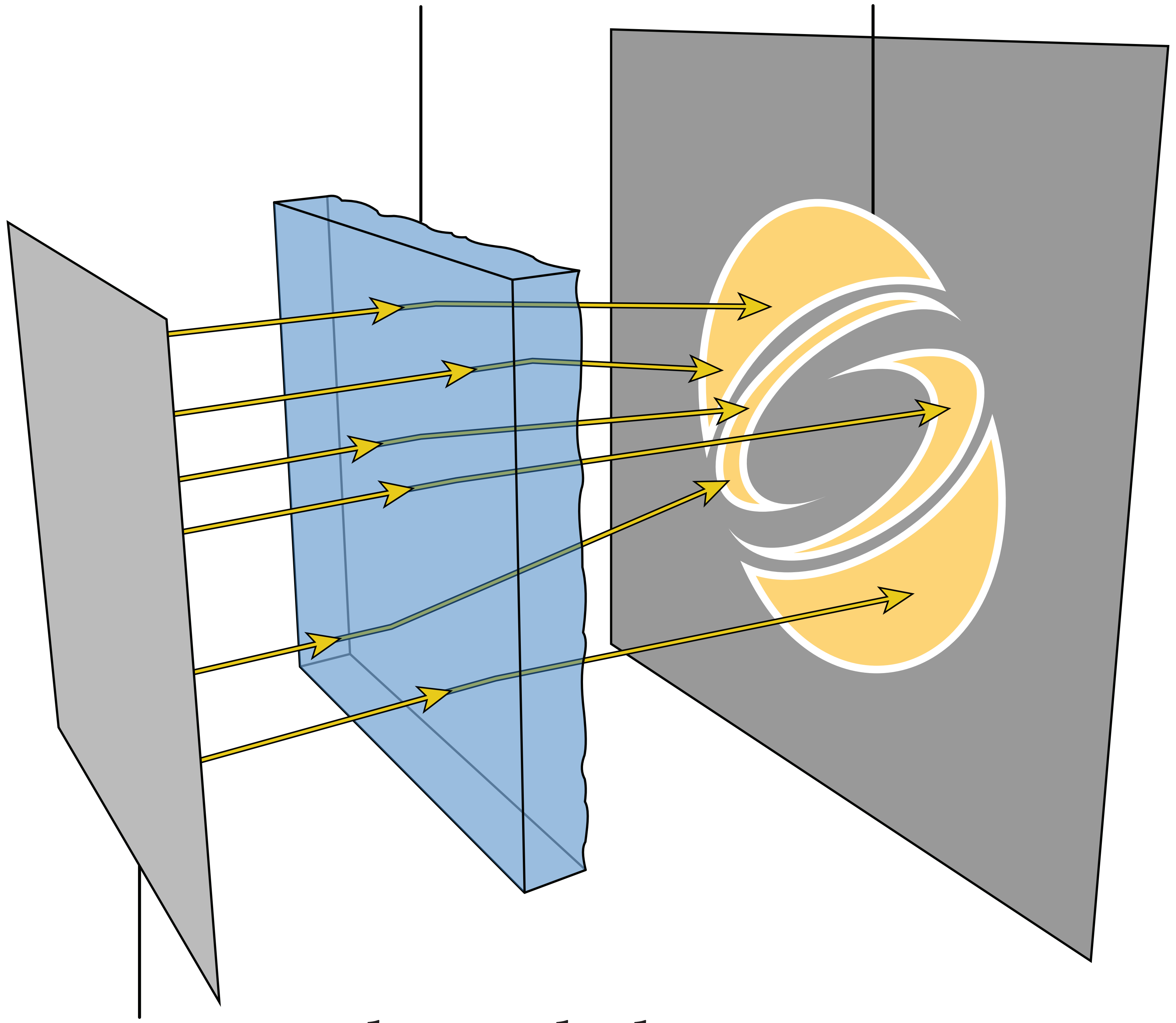
Target

Target

Reparameterizing discontinuous integrands for differentiable rendering [Loubet et al. 2019]

CAUSTIC DESIGN

Schwartzburg et al. 2014



(META-) MATERIAL DESIGN

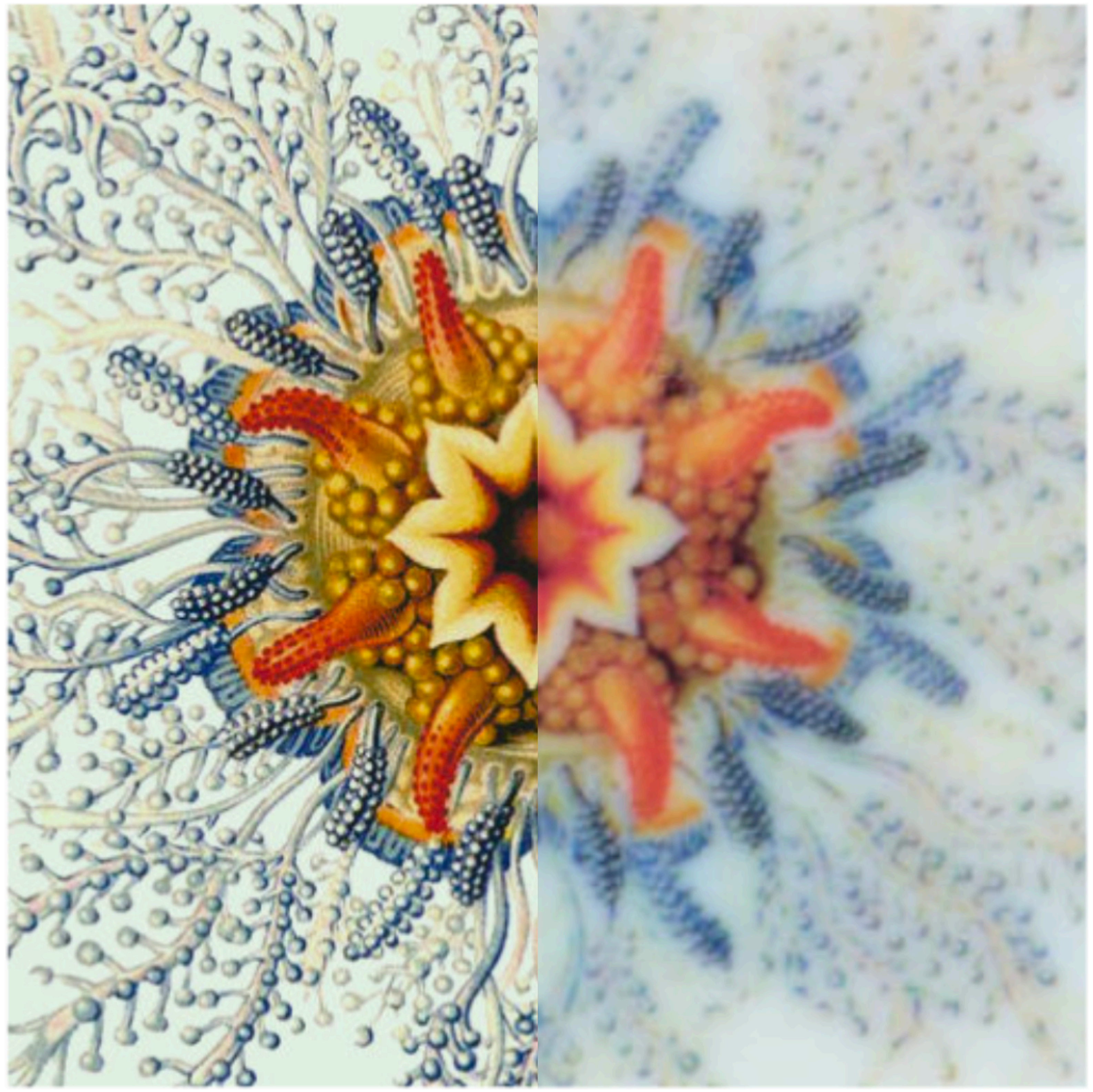
Mitsuba 2: A Retargetable Forward and Inverse Renderer [Nimier-David et al. 2019]

(META-) MATERIAL DESIGN

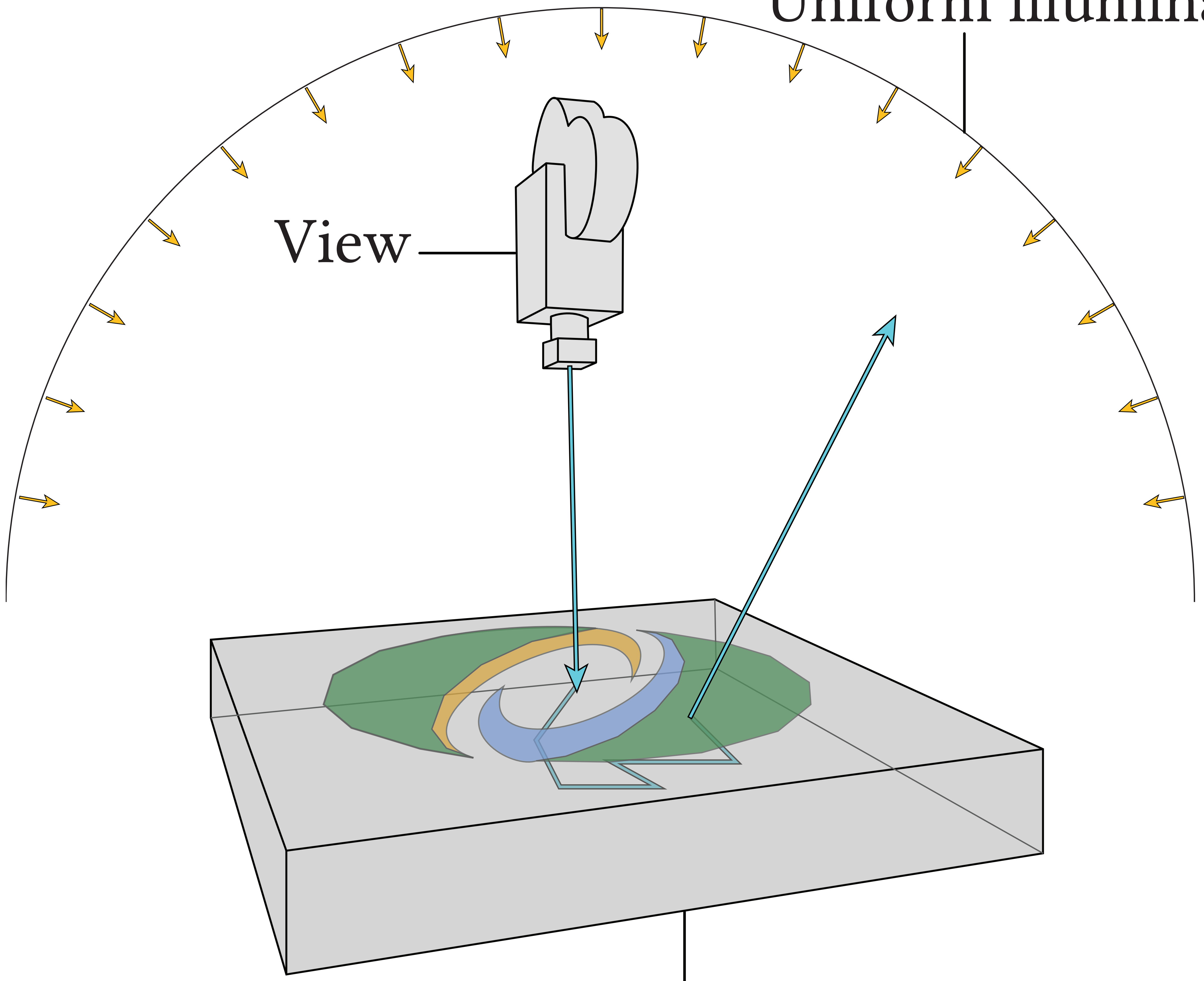
Mitsuba 2: A Retargetable Forward and Inverse Renderer [Nimier-David et al. 2019]

FABRICATION: 3D PRINT OPTIMIZATION

Elek et al. 2017



Target Naive print



Mitsuba 2: A Retargetable Forward and Inverse Renderer [Nimier-David et al. 2019]

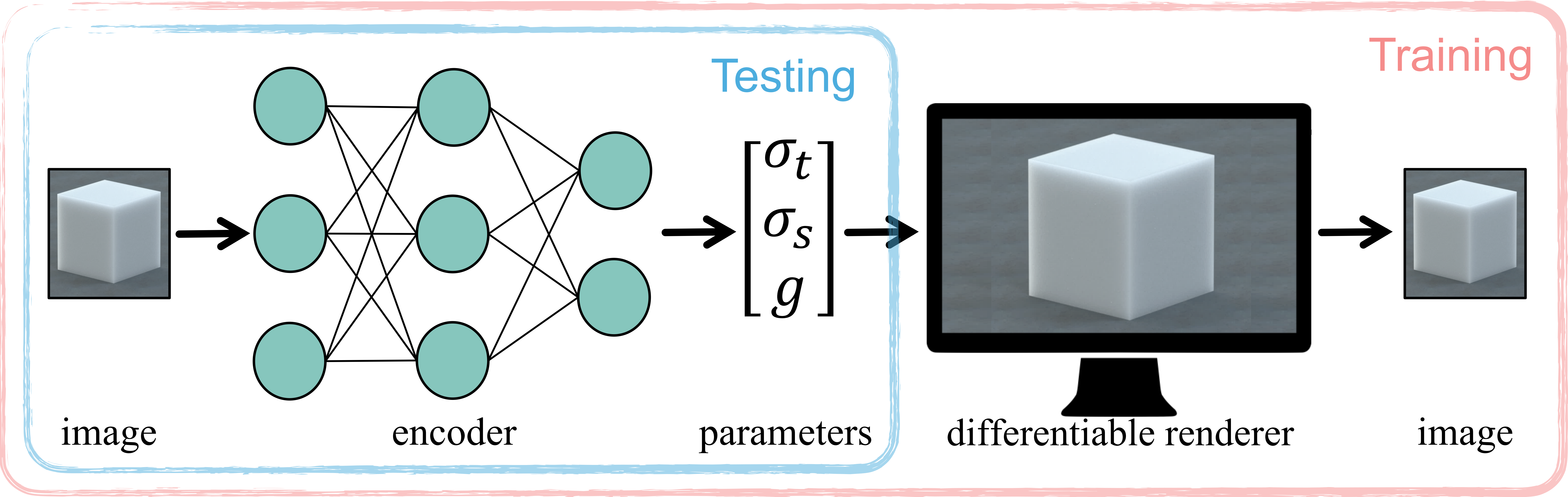
Reference: diffuse surface texture

WHY DIFFERENTIABLE RENDERING

SIGGRAPH 2020 S2020.SIGGRAPH.ORG THINK BEYOND

- Integrating physics-based rendering into **machine learning** & **probabilistic inference** pipelines

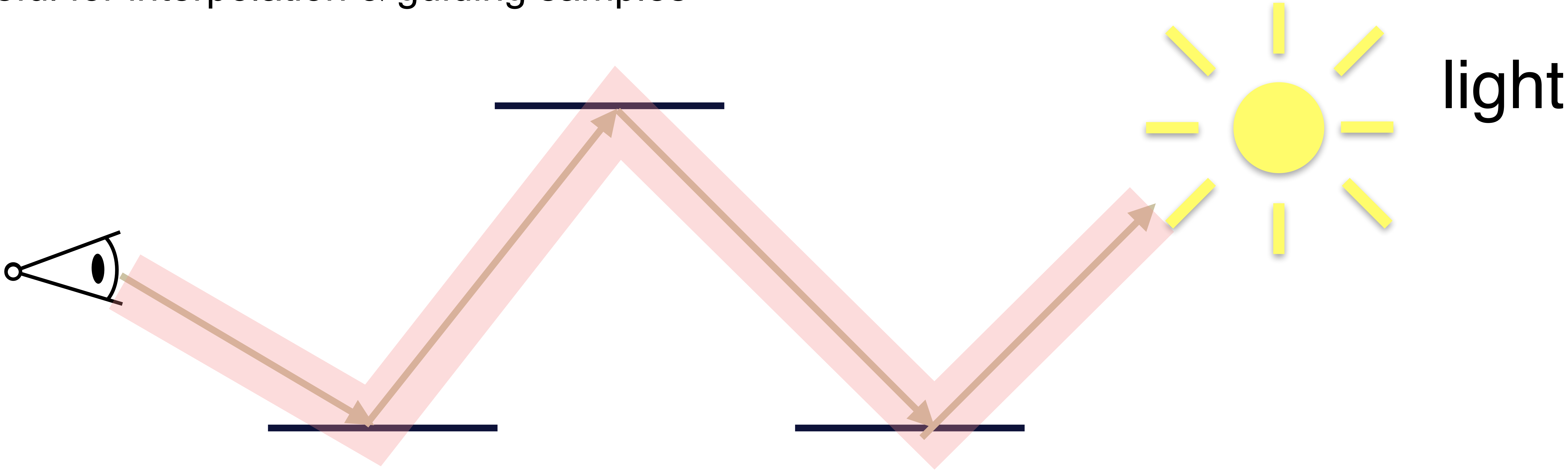
- Inverse subsurface scattering [Che et al. 2020]

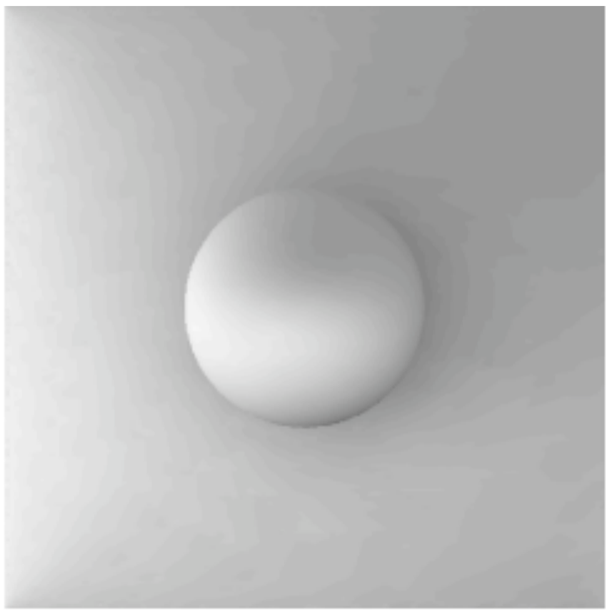


  - Utilizing *image loss* (provided by a volume path tracer) to regularize training
  - Use the trained encoder to solve inverse problems during testing

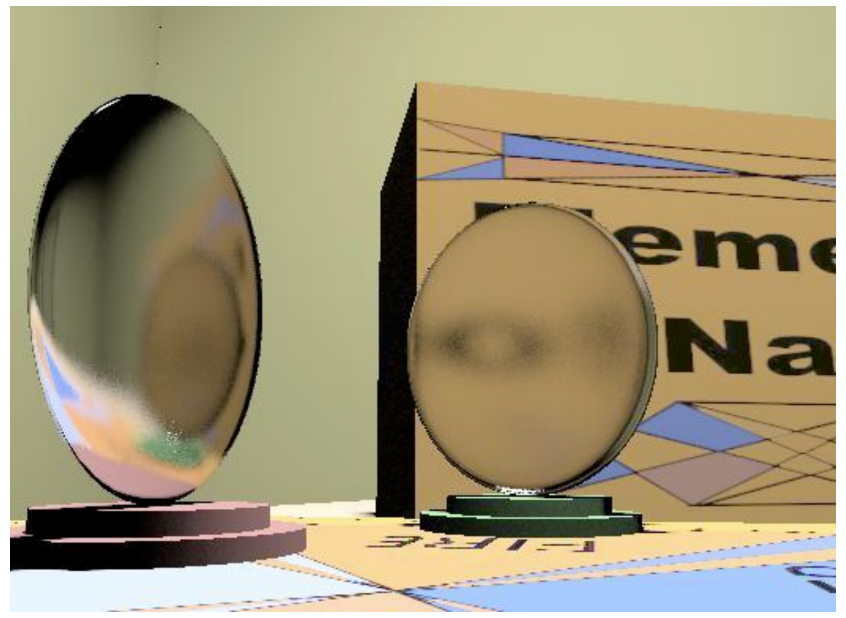
DIFFERENTIABLE RENDERING MAKES RENDERING FASTER

- Derivatives reveal neighborhood information of light paths
  - useful for interpolation & guiding samples

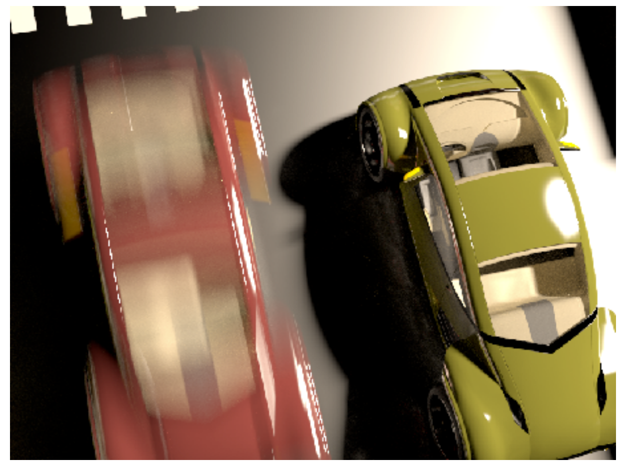




irradiance gradient
[Ward 1992]



path differentials
[Suykens and Williams 2001]

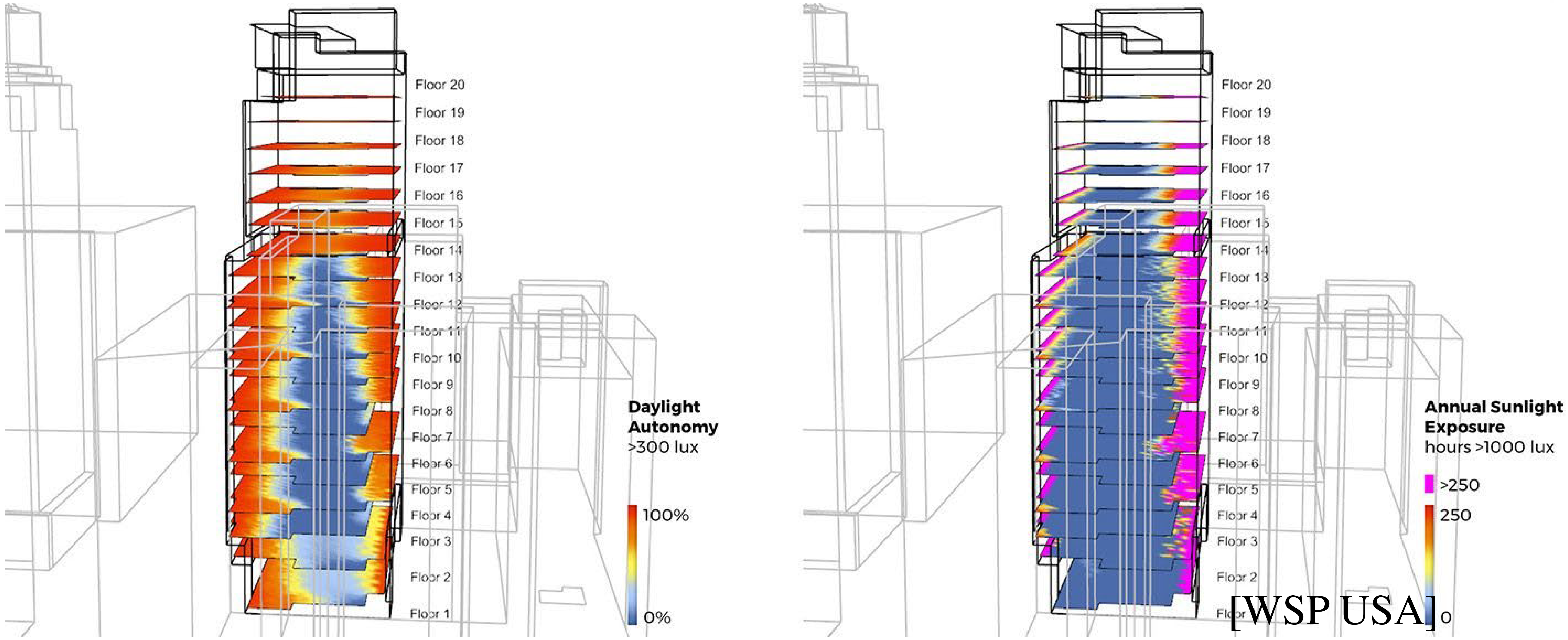


H2MC
[Li et al. 2015]

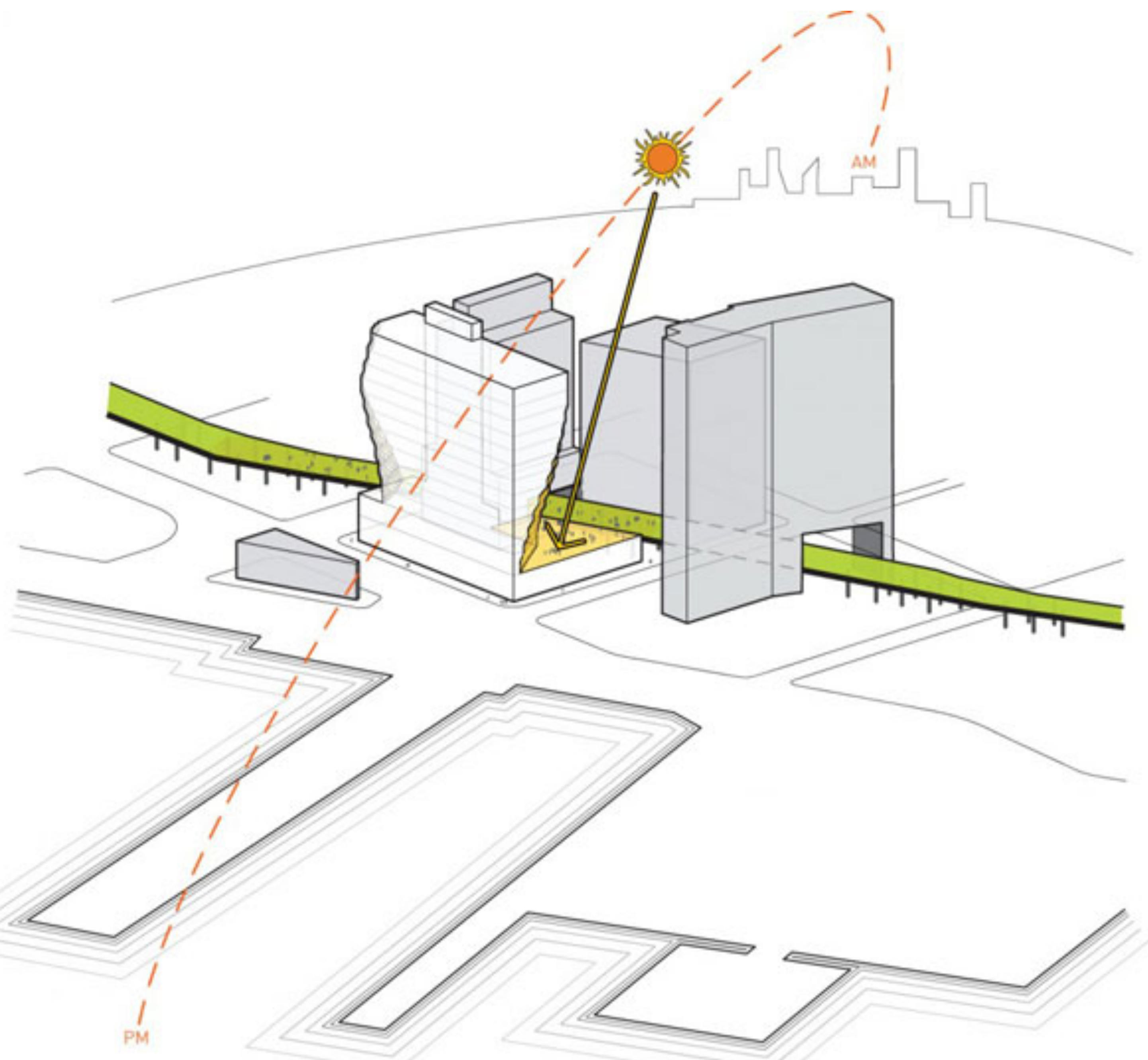
Langevin MC
[Luan et al. 2020]

BEYOND GRAPHICS: A WORLD OF APPLICATIONS

- Many disciplines rely on understanding or controlling the behavior of light in images or other kinds of measurements.



[WSP USA]

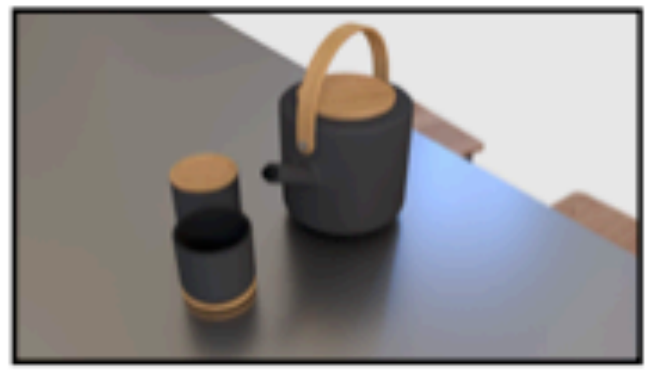
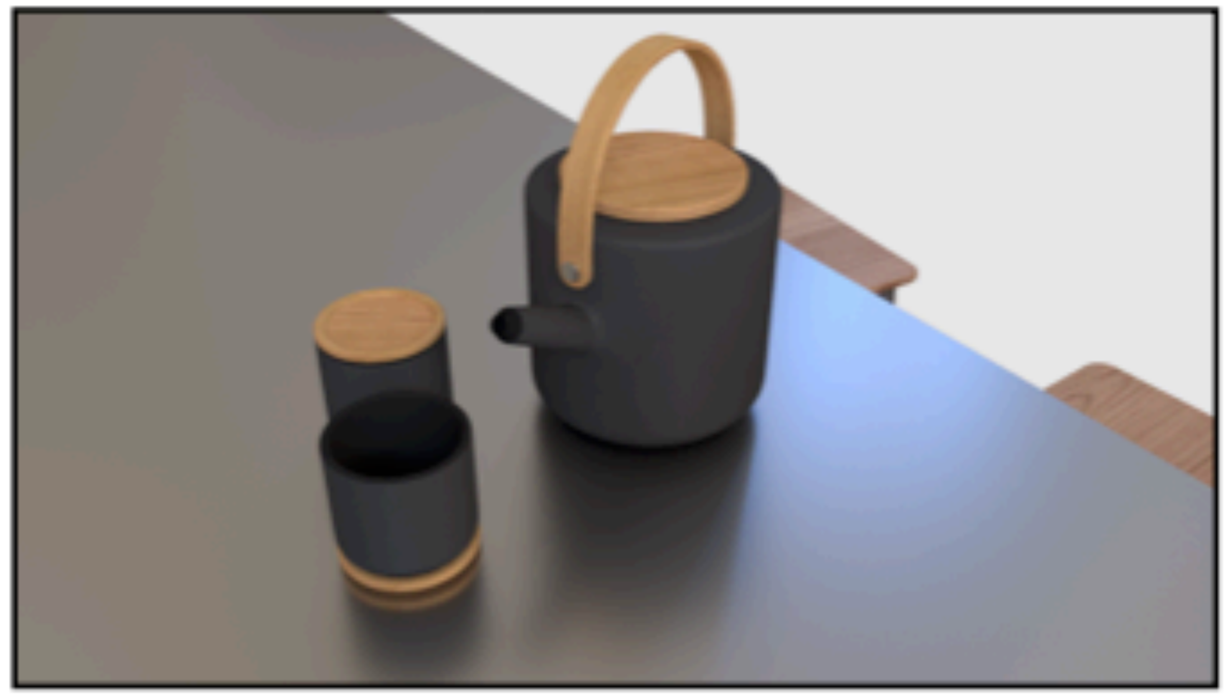
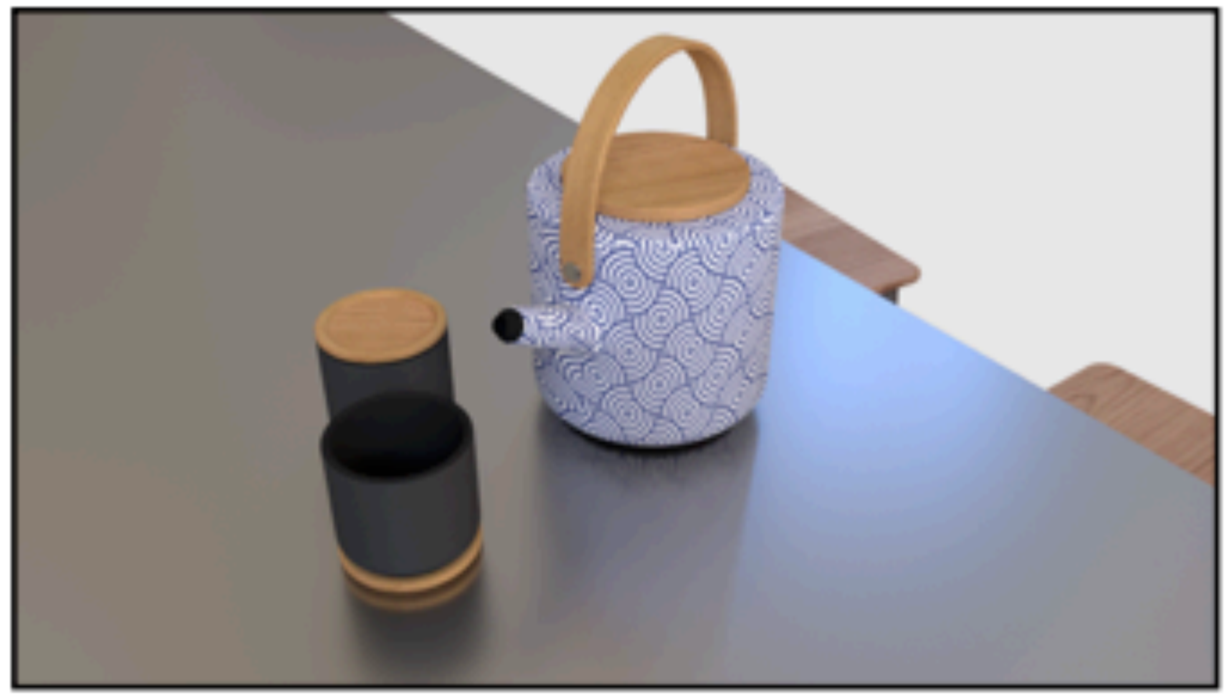


[Solar Carve Tower - Studio Gang]

Target

Current rendering

OBJECTIVE FUNCTION (A.K.A. "LOSS")

$g\Big($  $\Big) = \Bigg\|$  $-$  $\Bigg\|^2$

Rendering | Target

Scene parameters

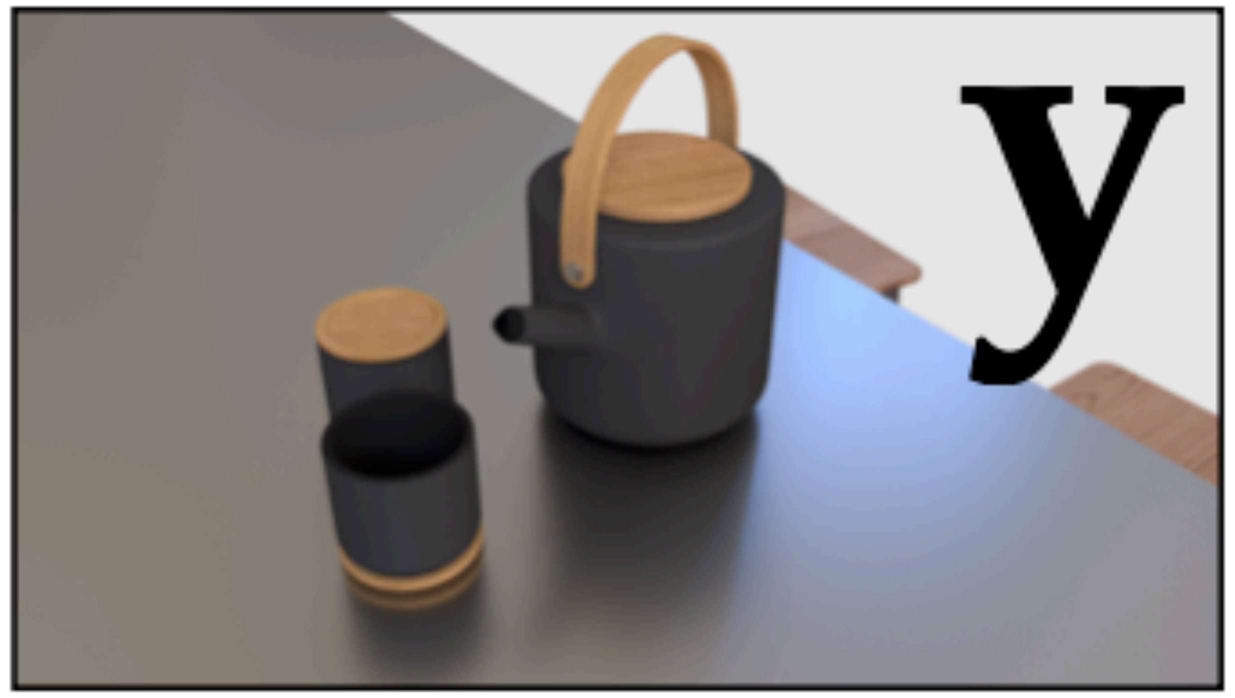
The problem: $\underset{\mathbf{x}\in\mathcal{X}}{\text{minimize}}\; g(f(\mathbf{x}))$

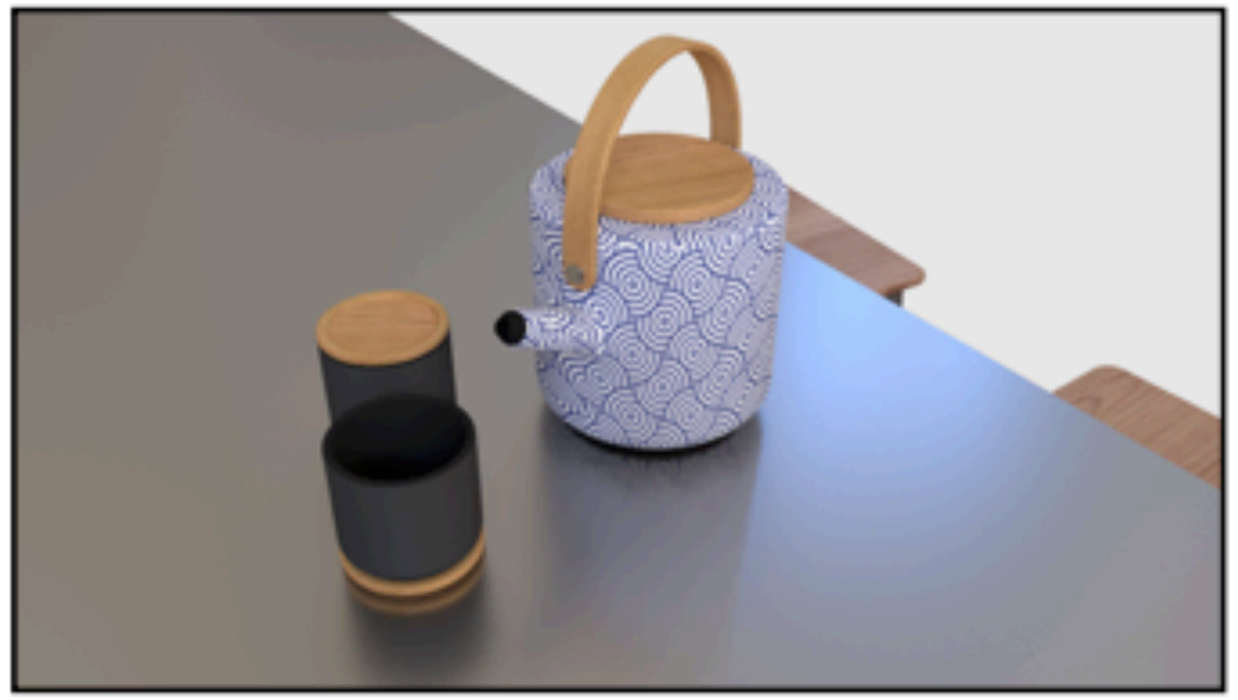
Objective | Rendering algorithm

The problem: $\underset{\mathbf{x}\in\mathcal{X}}{\text{minimize}}\; g(f(\mathbf{x}))$

$\mathbf{x}$

$f(\mathbf{x})$





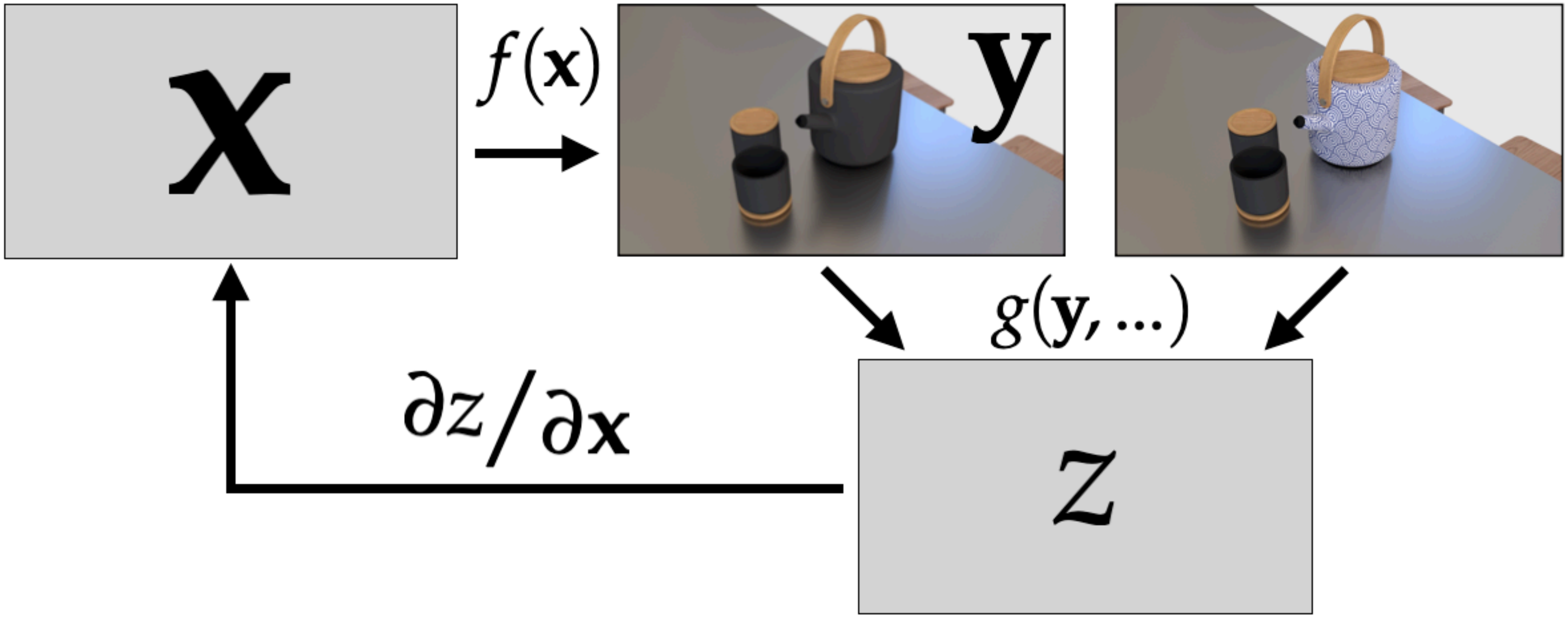
- meshes
- material (BSDF) parameters
  - textures, etc.
- parameters of procedural models
- volumes, light sources, …

$g(\mathbf{y}, \ldots)$

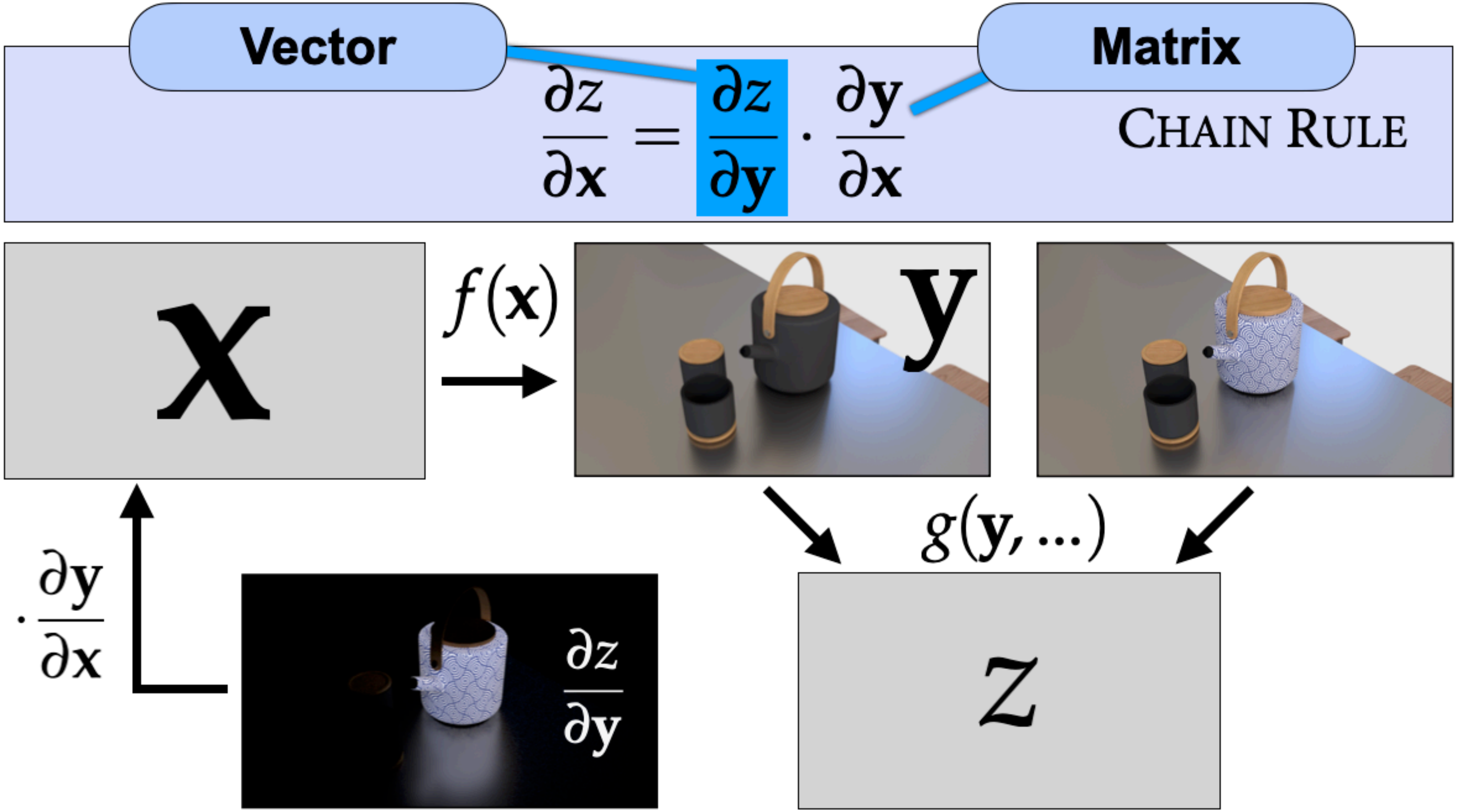
z

DIFFERENTIABLE RENDERING

The problem: $\underset{\mathbf{x} \in \mathcal{X}}{\text{minimize}}\; g(f(\mathbf{x}))$

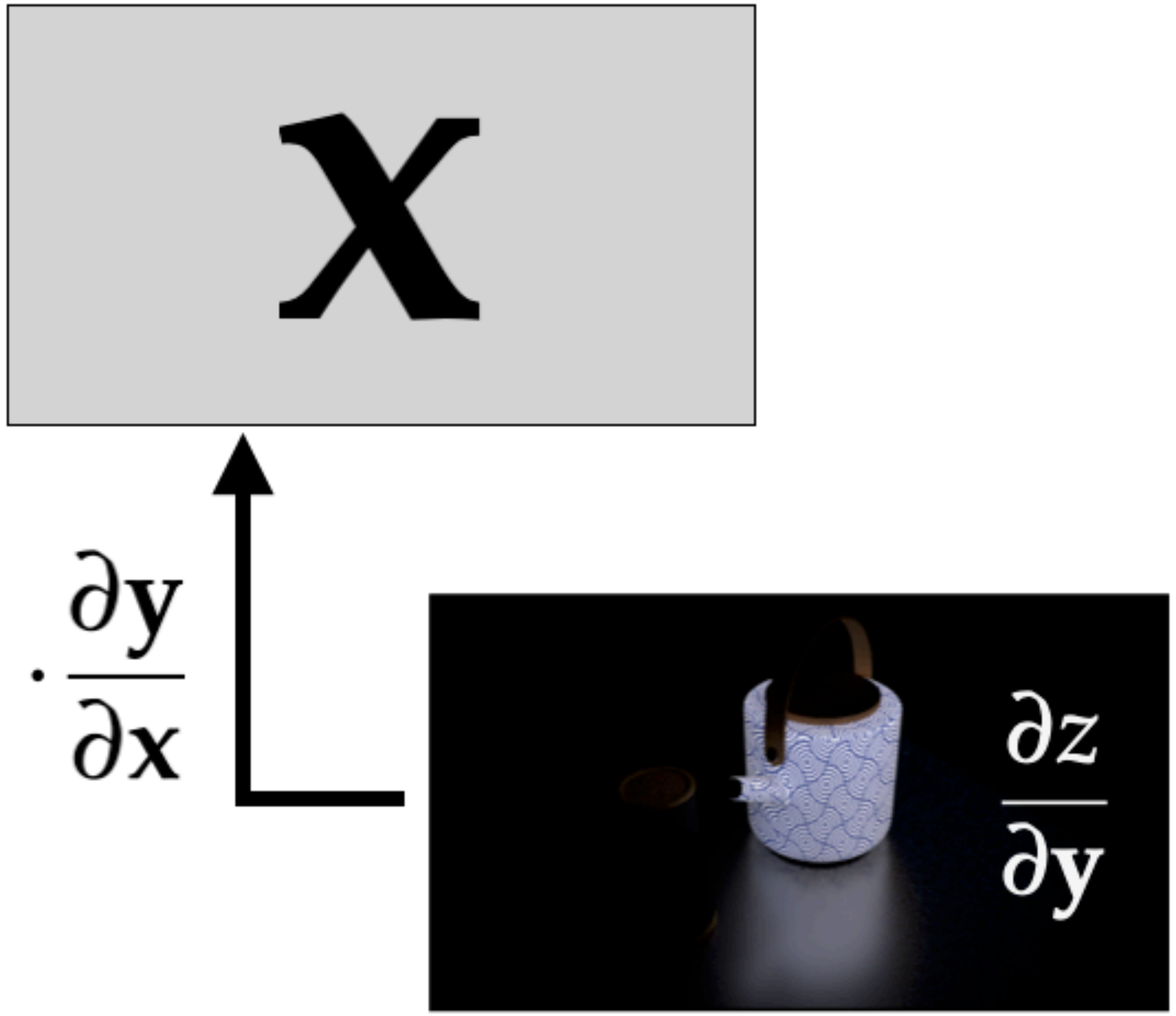


DIFFERENTIABLE RENDERING



DIFFERENTIABLE RENDERING

$$\frac{\partial z}{\partial \mathbf{x}} = \frac{\partial z}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$



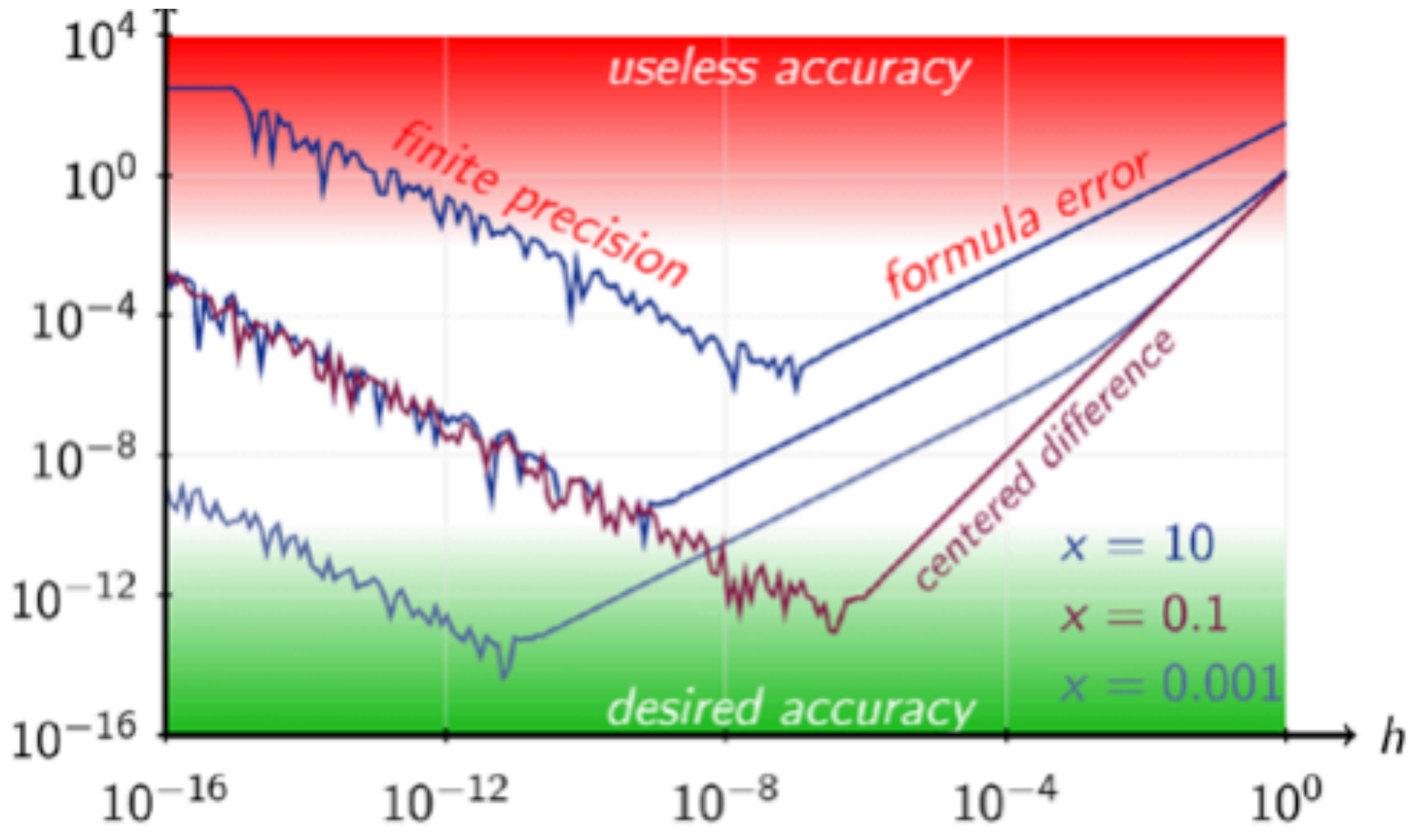
Challenges

1. Differentiating f
2. Matrix multiplication
3. Efficiency?
4. How to deal with edges?

HOW TO DO THIS (AT ALL?)

Use finite differences!

$$\frac{\partial \mathbf{y}}{\partial x_i} = \frac{f(\mathbf{x} + \varepsilon\, \mathbf{e}_i) - f(\mathbf{x} - \varepsilon\, \mathbf{e}_i)}{2\,\varepsilon}$$

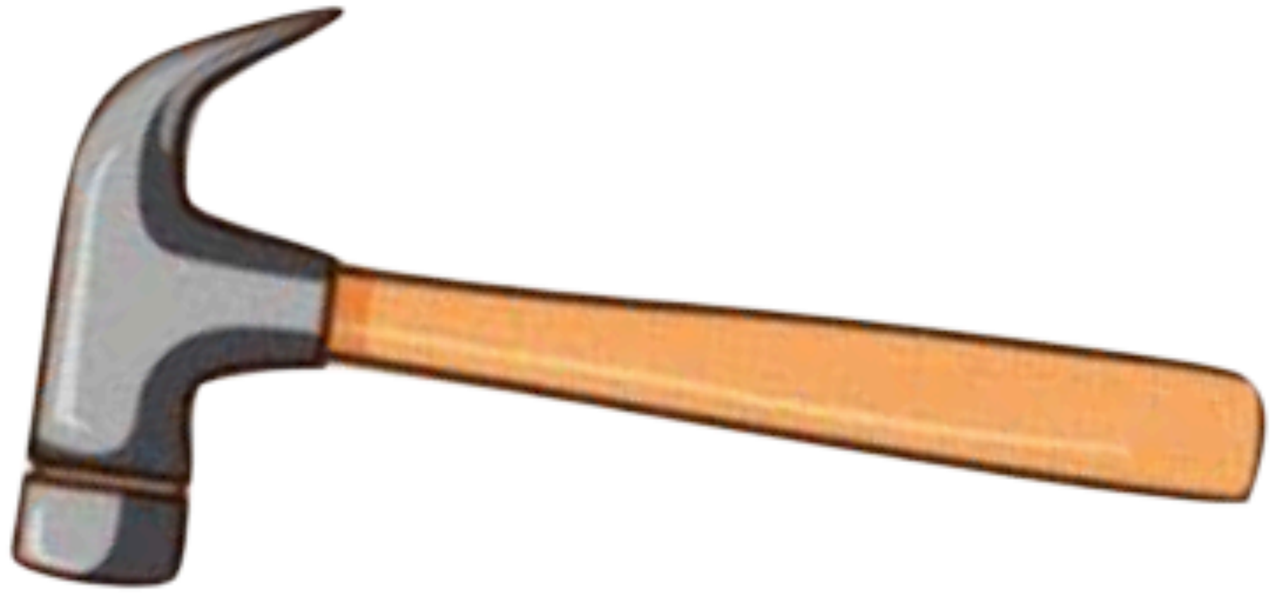


[Wikipedia]

Potential problems:

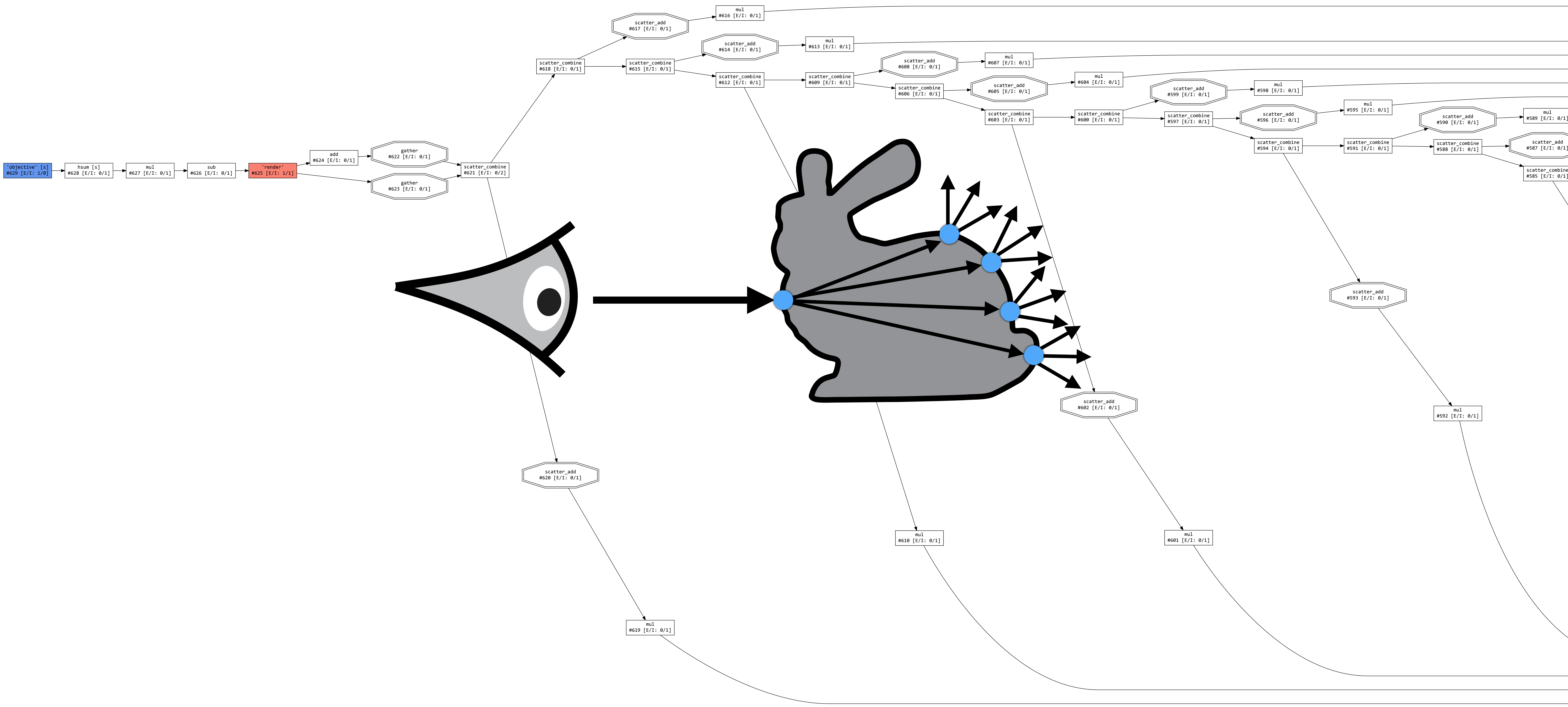
- Bad approximation (big ε), rounding error (small ε)
- Need to correlate Monte Carlo samples
- Extremely slow when many there are many parameters.

AUTOMATIC DIFFERENTIATION



$$f(\mathbf{x}) \xrightarrow[\text{AD}]{} \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x})$$

ISSUES WITH AUTOMATIC DIFFERENTIATION (AD)



- Precautions must be taken to ensure **correctness**
  - Symbolically differentiating a Monte Carlo estimator path tracer does not always work!

- **Example 1:** Distributional parameters

Estimate $\int_0^\infty f(\lambda, x)\,\mathrm{d}x$ (with λ given)

(Single-sample) Monte Carlo estimator:

- Draw $x \sim \mathrm{Exp}[\lambda]$
- $f \leftarrow f(\lambda, x)$
- $p \leftarrow \lambda \mathrm{e}^{-\lambda x}$ # This is the pdf of $\mathrm{Exp}[\lambda]$
- Return f/p

Estimate $\frac{\mathrm{d}}{\mathrm{d}\lambda}\int_0^\infty f(\lambda, x)\,\mathrm{d}x = \int_0^\infty \frac{\partial f}{\partial \lambda}(\lambda, x)\,\mathrm{d}x$

(Single-sample) Monte Carlo estimator:

- Draw $x \sim \mathrm{Exp}[\lambda]$ x has zero gradient
- $f' \leftarrow \frac{\partial f}{\partial \lambda}(\lambda, x)$
- $p \leftarrow \lambda \mathrm{e}^{-\lambda x}$ p is NOT differentiated
- Return f'/p

- Precautions must be taken to ensure **correctness**
  - Symbolically differentiating a Monte Carlo estimator path tracer does not always work!

- **Example 1:** Distributional parameters, with $\xi = \mathrm{e}^{-\lambda x}$

Estimate $\displaystyle\int_0^\infty f(\lambda, x)\,\mathrm{d}x = \int_0^1 \frac{f(\lambda, x)}{\lambda\xi}\,\mathrm{d}\xi$

(Single-sample) Monte Carlo estimator:

- Draw $\xi \sim U[0,1)$
- $x \leftarrow -\log(\xi)/\lambda$ $\quad \# \, x \sim \mathrm{Exp}(\lambda)$
- $f \leftarrow f(\lambda, x)$
- $p \leftarrow \lambda \mathrm{e}^{-\lambda x}$ $\quad \# \, p = \lambda\xi$
- Return f/p

Estimate $\displaystyle\frac{\mathrm{d}}{\mathrm{d}\lambda}\int_0^\infty f(\lambda, x)\,\mathrm{d}x = \int_0^1 \frac{\partial}{\partial\lambda}\frac{f(\lambda, x)}{\lambda\xi}\,\mathrm{d}\xi$

(Single-sample) Monte Carlo estimator:

- Draw $\xi \sim U[0,1)$
- $x \leftarrow -\log(\xi)/\lambda$ $\quad x$ has nonzero gradient
- $f \leftarrow f(\lambda, x)$
- $p \leftarrow \lambda \mathrm{e}^{-\lambda x}$ $\quad \# \, p = \lambda\xi$
- Return $\partial(f/p)/\partial\lambda$ $\quad f$ and p are both differentiated

- Precautions must be taken to ensure **correctness**
  - Symbolically differentiating a Monte Carlo estimator path tracer does not always work!

- **Example 1:** Distributional parameters

Estimate $\frac{\mathrm{d}}{\mathrm{d}\lambda}\int_0^\infty f(\lambda,x)\,\mathrm{d}x = \int_0^\infty \frac{\partial f}{\partial\lambda}(\lambda,x)\,\mathrm{d}x$

(Single-sample) Monte Carlo estimator:

- Draw $x \sim \mathrm{Exp}[\lambda]$ — x has zero gradient
- $f' \leftarrow \frac{\partial f}{\partial\lambda}(\lambda,x)$
- $p \leftarrow \lambda \mathrm{e}^{-\lambda x}$ — p is NOT differentiated
- Return f'/p

Estimate $\frac{\mathrm{d}}{\mathrm{d}\lambda}\int_0^\infty f(\lambda,x)\,\mathrm{d}x = \int_0^1 \frac{\partial}{\partial\lambda}\frac{f(\lambda,x)}{\lambda\xi}\,\mathrm{d}\xi$

(Single-sample) Monte Carlo estimator:

- Draw $\xi \sim U[0,1)$
- $x \leftarrow -\log(\xi)/\lambda$ — x has nonzero gradient
- $f \leftarrow f(\lambda,x)$
- $p \leftarrow \lambda \mathrm{e}^{-\lambda x}$ $\quad \# \, p = \lambda\xi$
- Return $\partial(f/p)/\partial\lambda$ — f and p are both differentiated

Whether to differentiate the *sampling* and the *pdf* should be **consistent**!

WHY IS DIFFERENTIABLE RENDERING DIFFICULT

- Precautions must be taken to ensure **correctness**
  - Symbolically differentiating a Monte Carlo estimator path tracer does not always work!

- **Example 2:** Discontinuities

Estimate $\int_0^1 (x < p\ ?\ 1 : 0.5)\,\mathrm{d}x$ with $0 < p < 1$

(Single-sample) Monte Carlo estimator:

- Draw $X \sim U[0, 1)$
- Return $X < p\ ?\ 1 : 0.5$

Ground-truth:

$$\int_0^1 (x < p\ ?\ 1 : 0.5)\,\mathrm{d}x = \int_0^p \mathrm{d}x + \int_p^1 0.5\,\mathrm{d}x = \frac{1+p}{2}$$

Estimate $\frac{\mathrm{d}}{\mathrm{d}p}\int_0^1 (x < p\ ?\ 1 : 0.5)\,\mathrm{d}x$ with $0 < p < 1$

(Single-sample) Monte Carlo estimator:

- Draw $X \sim U[0, 1)$
- Return $\mathrm{d}(X < p\ ?\ 1 : 0.5)/\mathrm{d}p$ Zero! (constant)

$\nparallel$

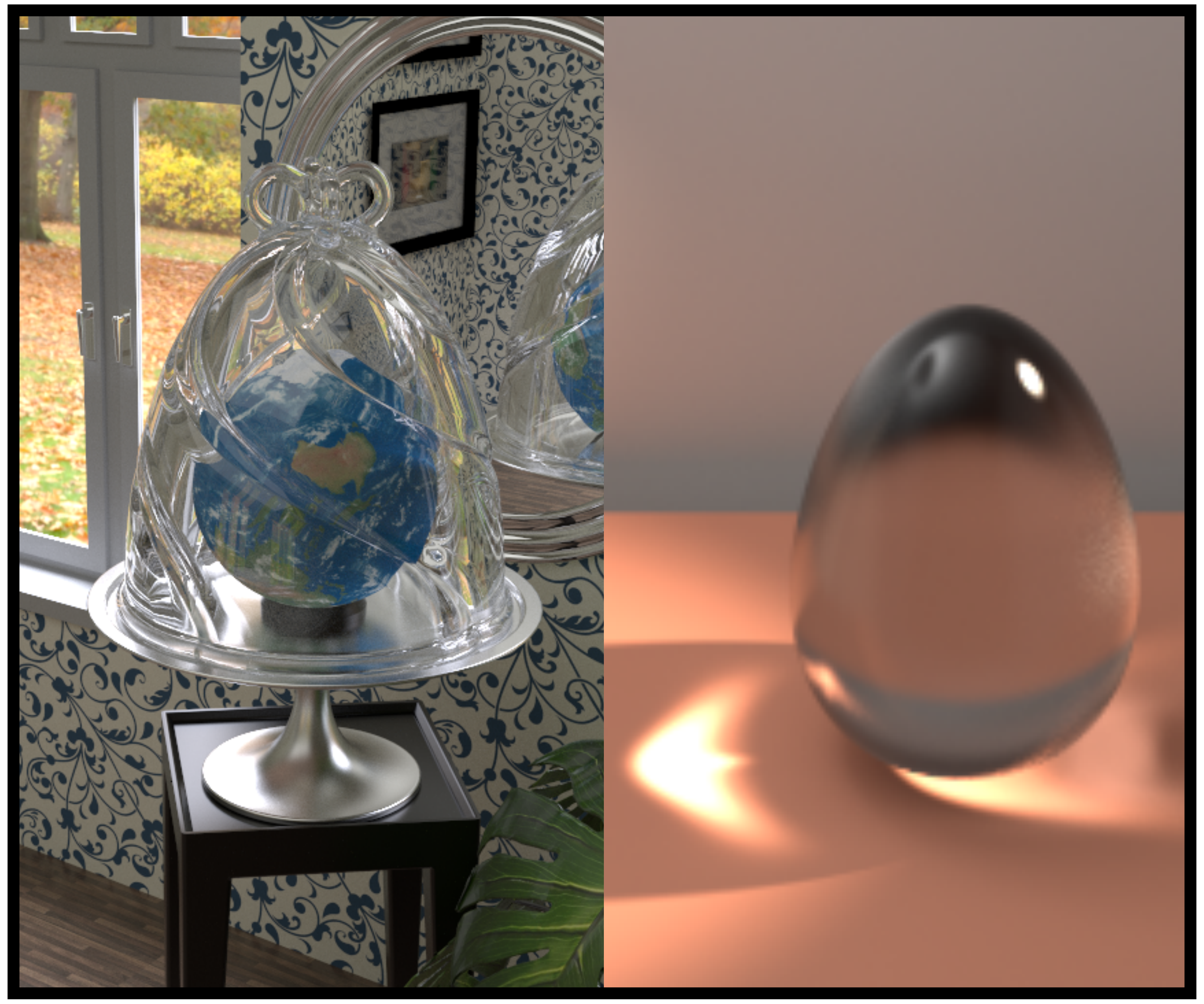
Ground-truth:

$$\frac{\mathrm{d}}{\mathrm{d}p}\int_0^1 (x < p\ ?\ 1 : 0.5)\,\mathrm{d}x = \frac{\mathrm{d}}{\mathrm{d}p}\frac{1+p}{2} = \frac{1}{2}$$

More on this example later

COURSE OUTLINE

Basics



State-of-the-art theories and algorithms

Implementation details

BASICS

DIFFERENTIATING (RENDERING) PROGRAMS

- a crash course on automatic differentiation
- differentiating discontinuities in rendering
- discussions & limitations

∇

```
auto scatter_contrib = Vector3{0, 0, 0};
auto scatter_bsdf = Vector3{0, 0, 0};
if (bsdf_isect.valid()) {
    const auto &bsdf_shape = scene.shapes[bsdf_isect.shape_id];
    auto dir = bsdf_point.position - p;
    auto dist_sq = length_squared(dir);
    auto wo = dir / sqrt(dist_sq);
    auto pdf_bsdf = bsdf_pdf(material, shading_point, wi, wo, min_rough);
    if (dist_sq > 1e-20f && pdf_bsdf > 1e-20f) {
        auto bsdf_val = bsdf(material, shading_point, wi, wo, min_rough);
        if (bsdf_shape.light_id >= 0) {
            const auto &light = scene.area_lights[bsdf_shape.light_id];
            if (light.two_sided || dot(-wo, bsdf_point.shading_frame.n) >
                auto light_contrib = light.intensity;
                auto light_pmf = scene.light_pmf[bsdf_shape.light_id];
                auto light_area = scene.light_areas[bsdf_shape.light_id];
                auto inv_area = 1 / light_area;
                auto geometry_term = fabs(dot(wo, bsdf_point.geom_normal))
                auto pdf_nee = (light_pmf * inv_area) / geometry_term;
                auto mis_weight = Real(1 / (1 + square((double)pdf_nee / (
                scatter_contrib = (mis_weight / pdf_bsdf) * bsdf_val * lig
            }
        }
        scatter_bsdf = bsdf_val / pdf_bsdf;
        next_throughput = throughput * scatter_bsdf;
```

- automatic differentiation v.s. symbolic differentiation

```
function f(x):
  result = x
  for i = 1 to 8:
    result = exp(result)
  return result
```

- automatic differentiation v.s. symbolic differentiation

symbolic differentiation (37 exponents):

$$\frac{df(x)}{dx} = e^{x+e^{e^{e^{e^{e^{e^{e^{x}}}}}}}+e^{e^{e^{e^{e^{e^{x}}}}}}+e^{e^{e^{e^{e^{x}}}}}+e^{e^{e^{e^{x}}}}+e^{e^{e^{x}}}+e^{e^{x}}+e^{x}}$$

```
function f(x):
  result = x
  for i = 1 to 8:
    result = exp(result)
  return result
```

- automatic differentiation v.s. symbolic differentiation

```
function f(x):
  result = x
  for i = 1 to 8:
    result = exp(result)
  return result
```

symbolic differentiation (37 exponents):

$$\frac{df(x)}{dx} = e^{x+e^{e^{e^{e^{e^{e^{e^{x}}}}}}}+e^{e^{e^{e^{e^{e^{x}}}}}}+e^{e^{e^{e^{e^{x}}}}}+e^{e^{e^{e^{x}}}}+e^{e^{e^{x}}}+e^{e^{x}}+e^{x}}$$

forward-mode automatic differentiation (8 exponents):

```
function d_f(x):
  result = x
  d_result = 1
  for i = 1 to 8:
    result = exp(result)
    d_result = d_result * result
  return d_result
```

- key idea: chain rules, but applied in a smart way

```
y = f(x)
z = g(y)
```

- key idea: chain rules, but applied in a smart way

```
y = f(x)
z = g(y)
```

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

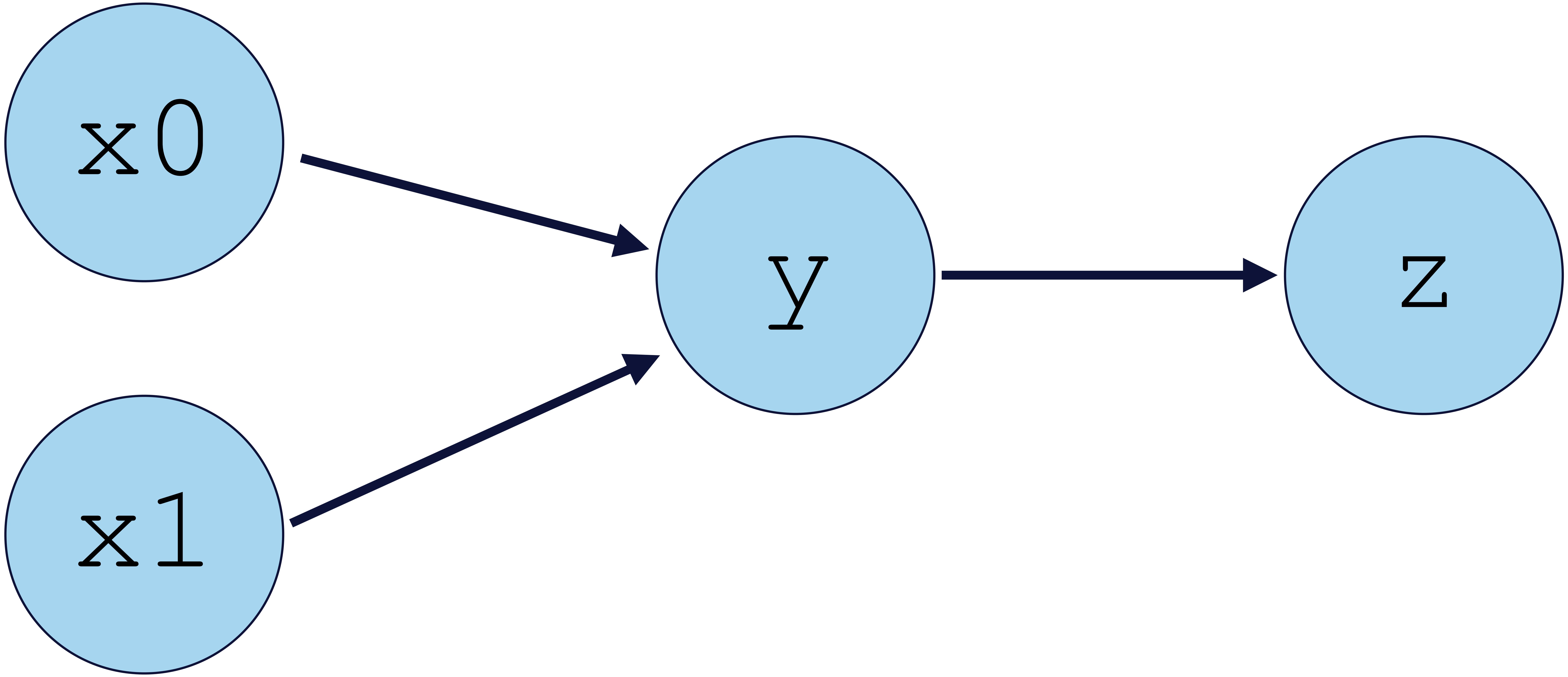
MENTAL MODEL: COMPUTATIONAL GRAPH

```
y = f(x)
z = g(y)
```

$$x \xrightarrow{\dfrac{dy}{dx}} y \xrightarrow{\dfrac{dz}{dy}} z$$

MULTIVARIATE EXAMPLE

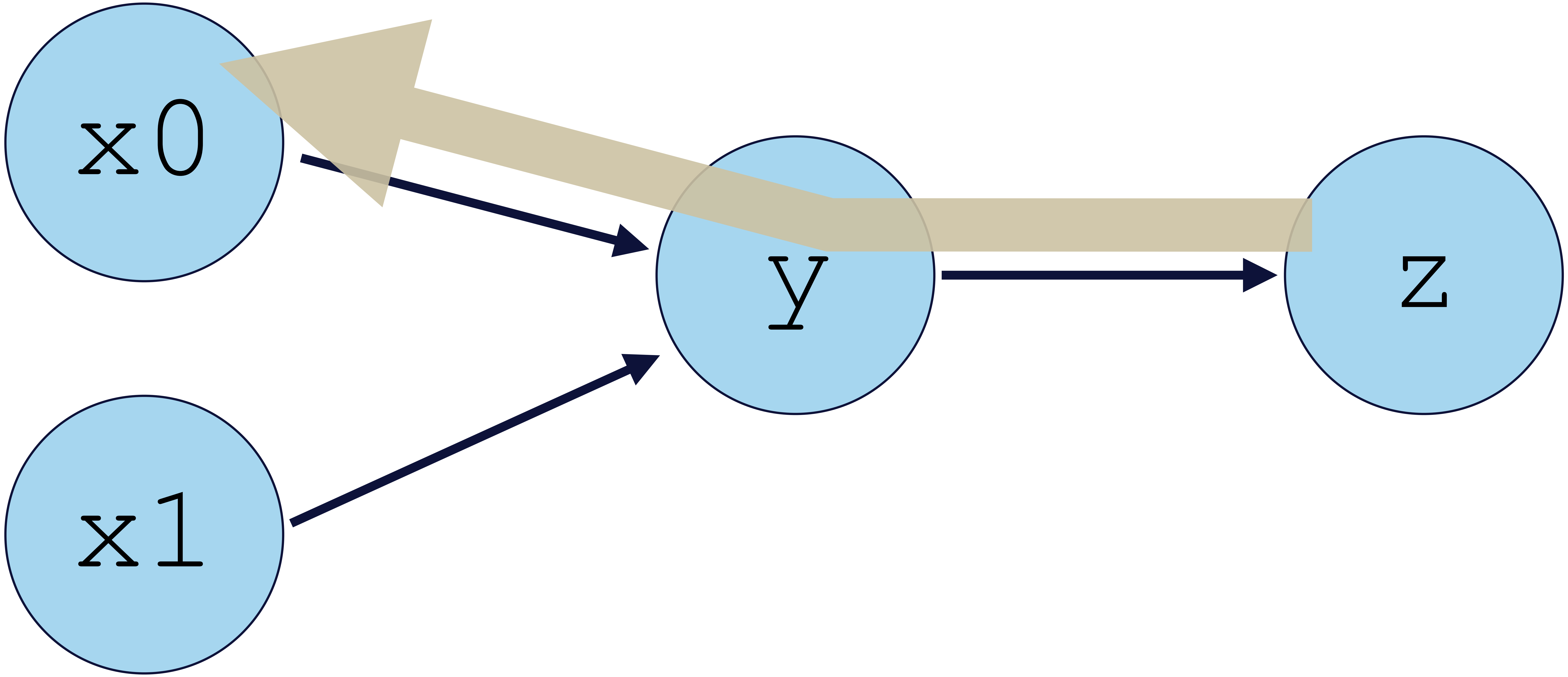
```
y = f(x0, x1)
z = g(y)
```



MULTIVARIATE EXAMPLE

```
y = f(x0, x1)
z = g(y)
```

$$\frac{\partial z}{\partial x_0} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x_0}$$

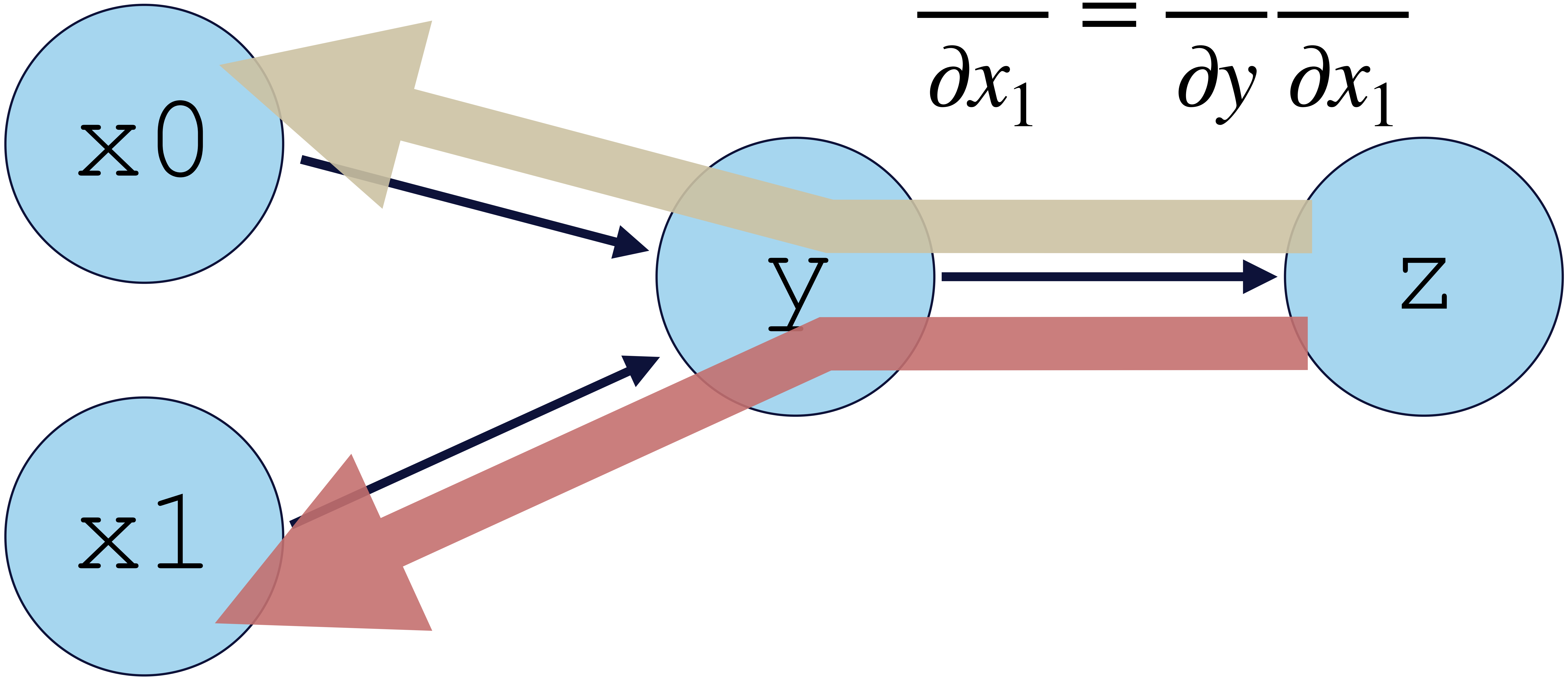


MULTIVARIATE EXAMPLE

```
y = f(x0, x1)
z = g(y)
```

$$\frac{\partial z}{\partial x_0} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x_0}$$

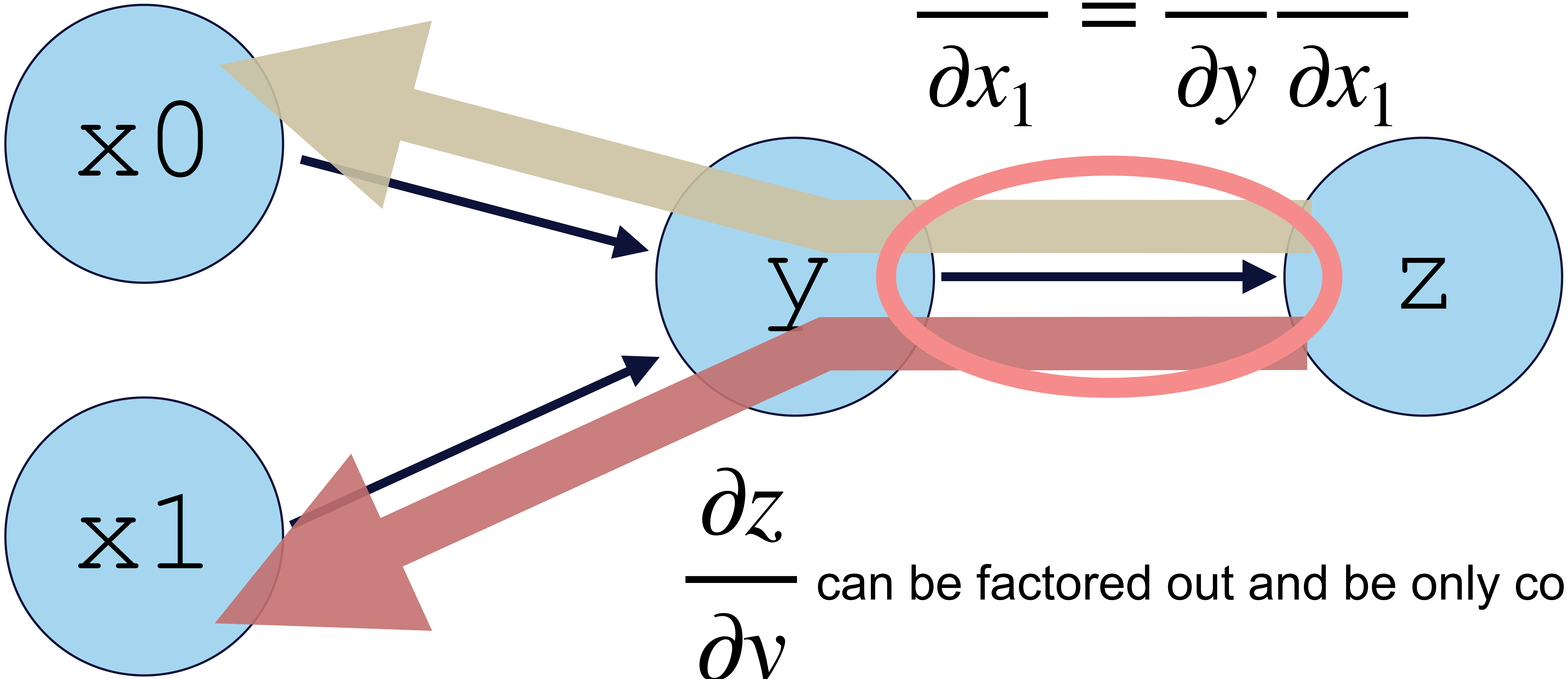
$$\frac{\partial z}{\partial x_1} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x_1}$$



```
y = f(x0, x1)
z = g(y)
```

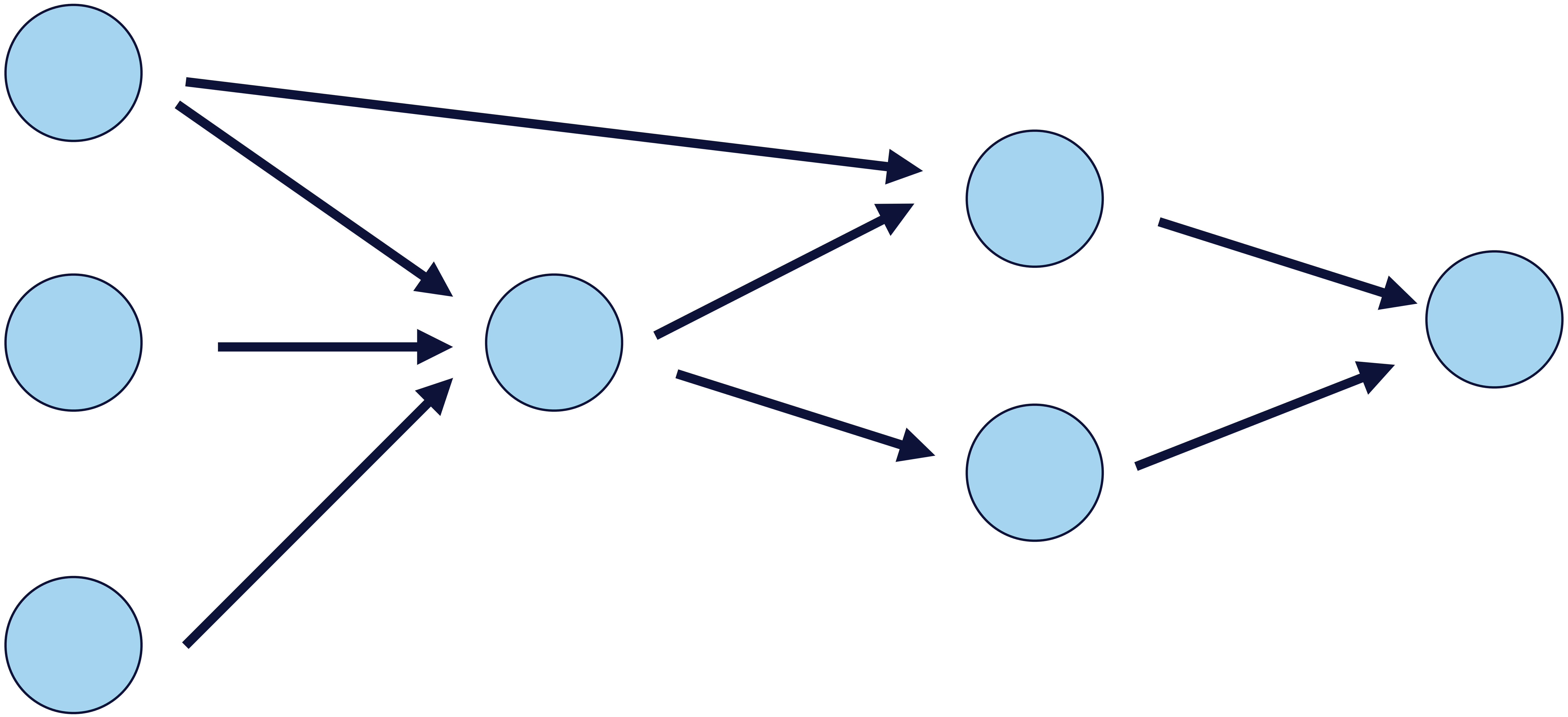
$$\frac{\partial z}{\partial x_0} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x_0}$$

$$\frac{\partial z}{\partial x_1} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x_1}$$

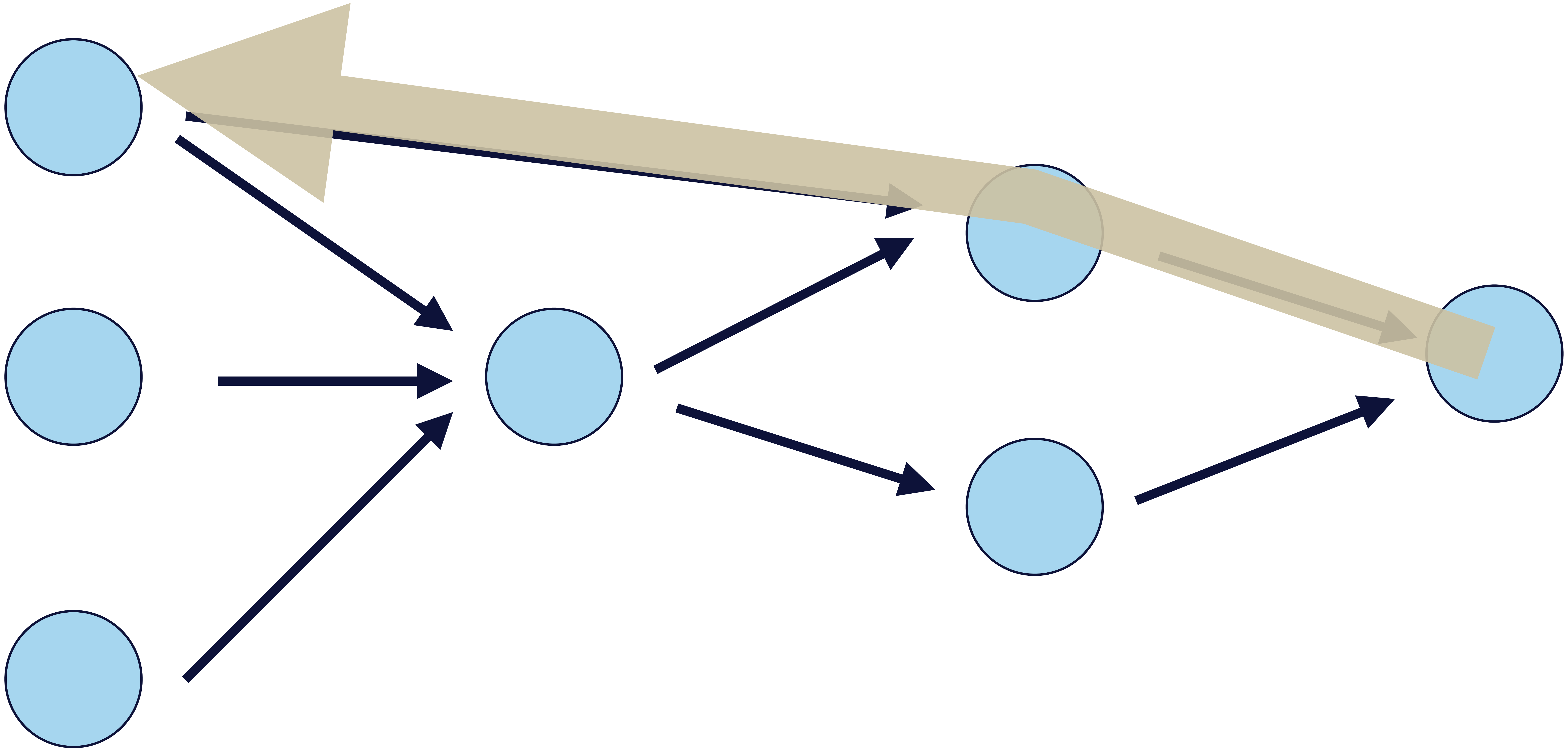


$\frac{\partial z}{\partial y}$ can be factored out and be only computed once!

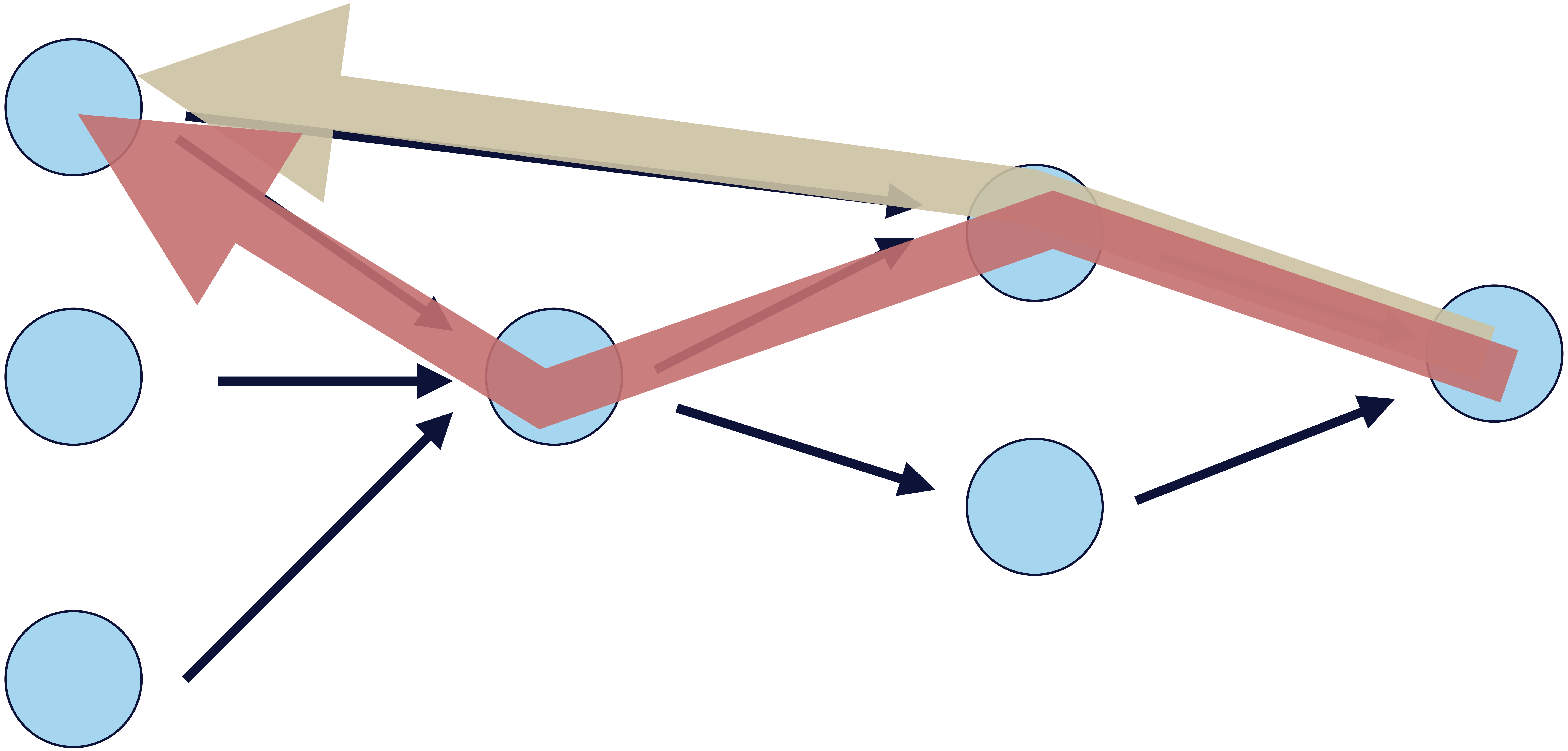
AUTODIFF = A PATH FINDING PROBLEM



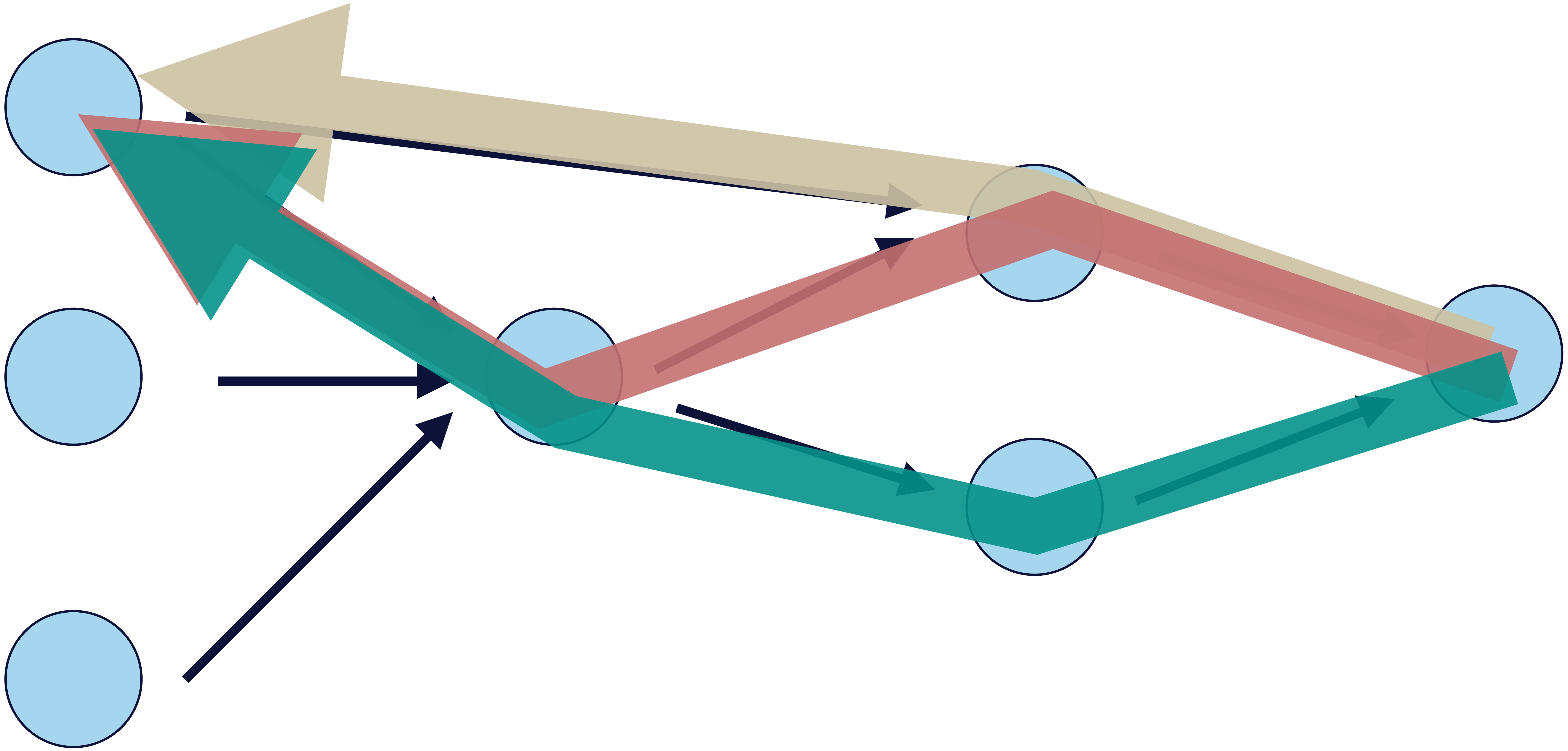
AUTODIFF = A PATH FINDING PROBLEM



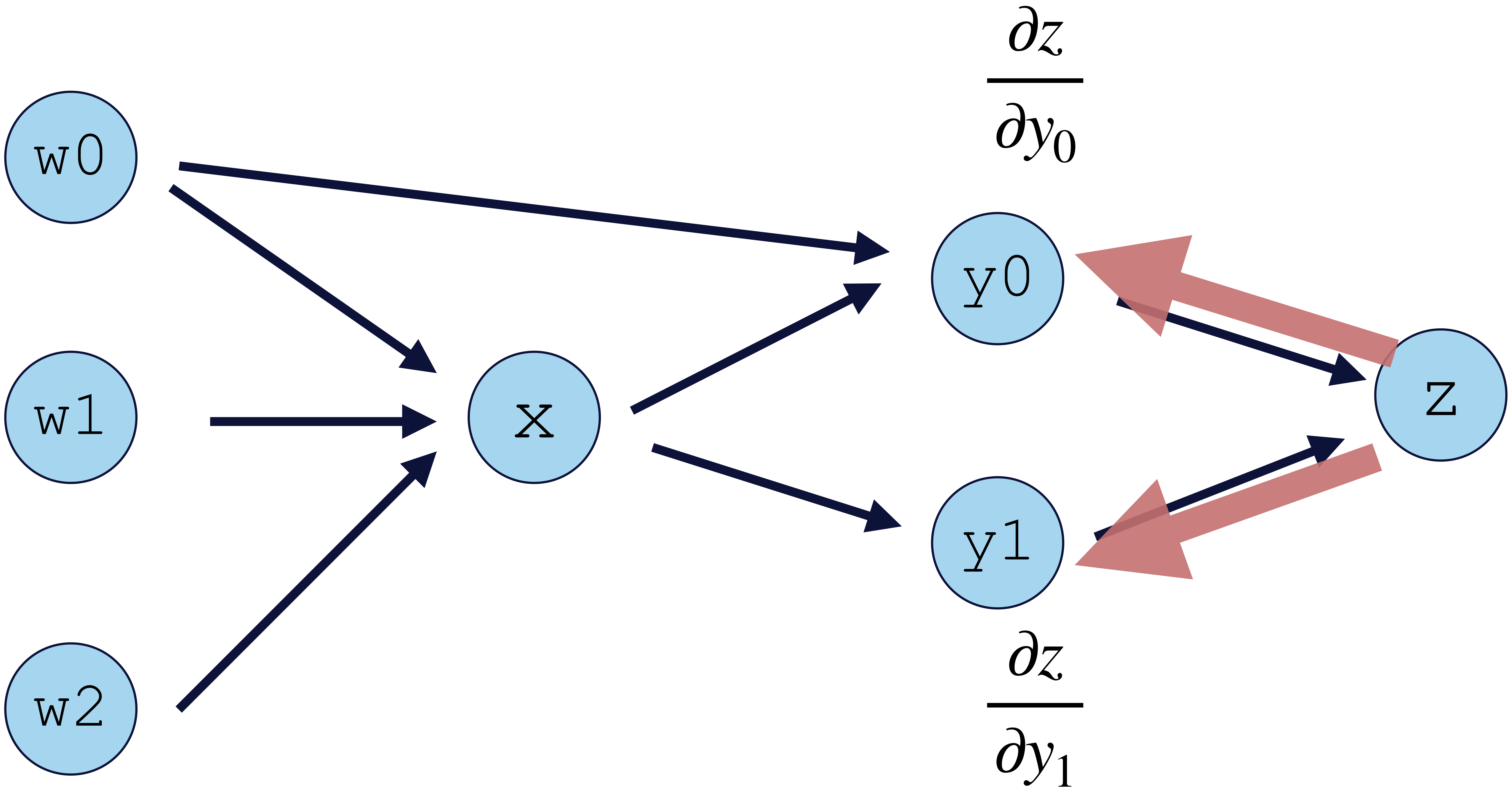
AUTODIFF = A PATH FINDING PROBLEM



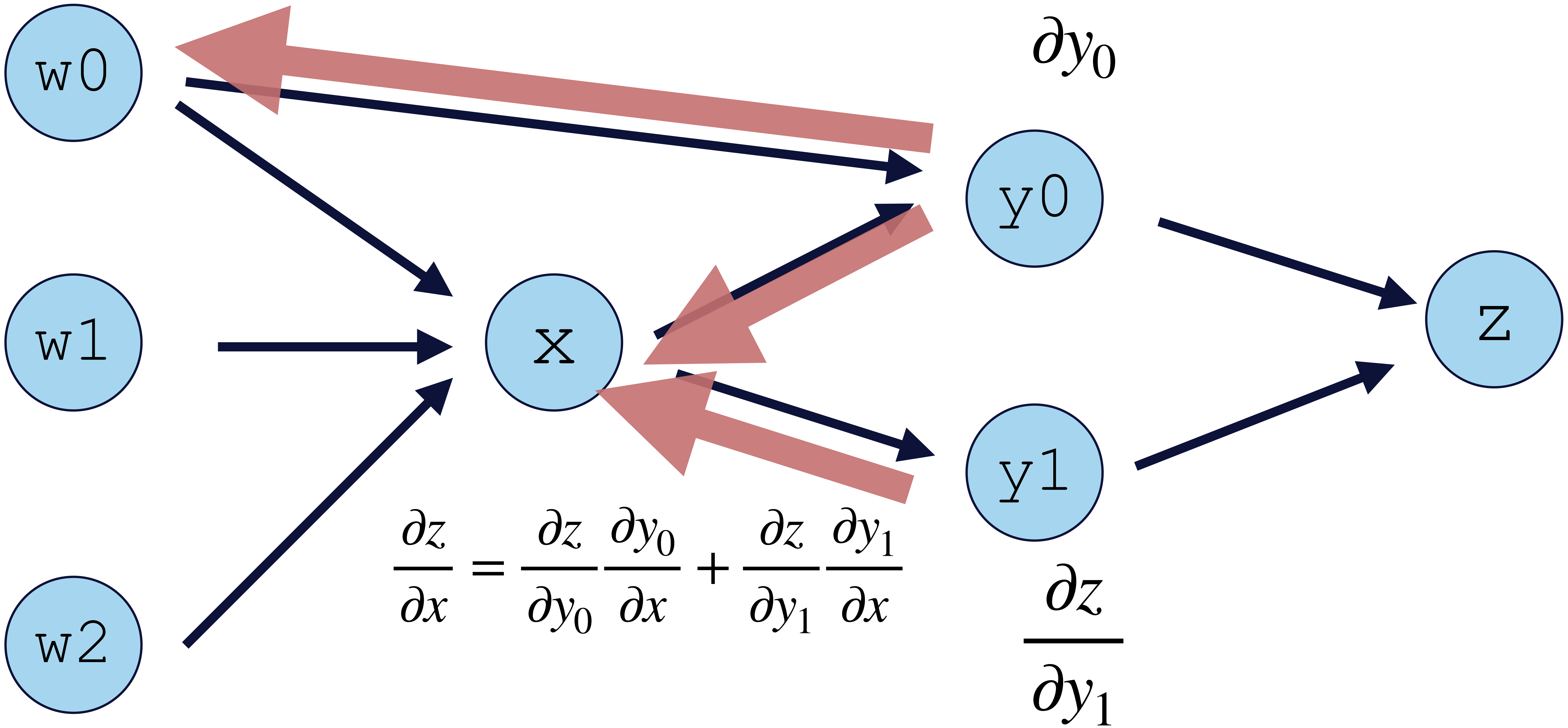
AUTODIFF = A PATH FINDING PROBLEM



REVERSE-MODE AUTOMATIC DIFFERENTIATION = A GREEDY PATH FACTORIZATION ALGORITHM



$$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial y_0}\frac{\partial y_0}{\partial w_0}$$



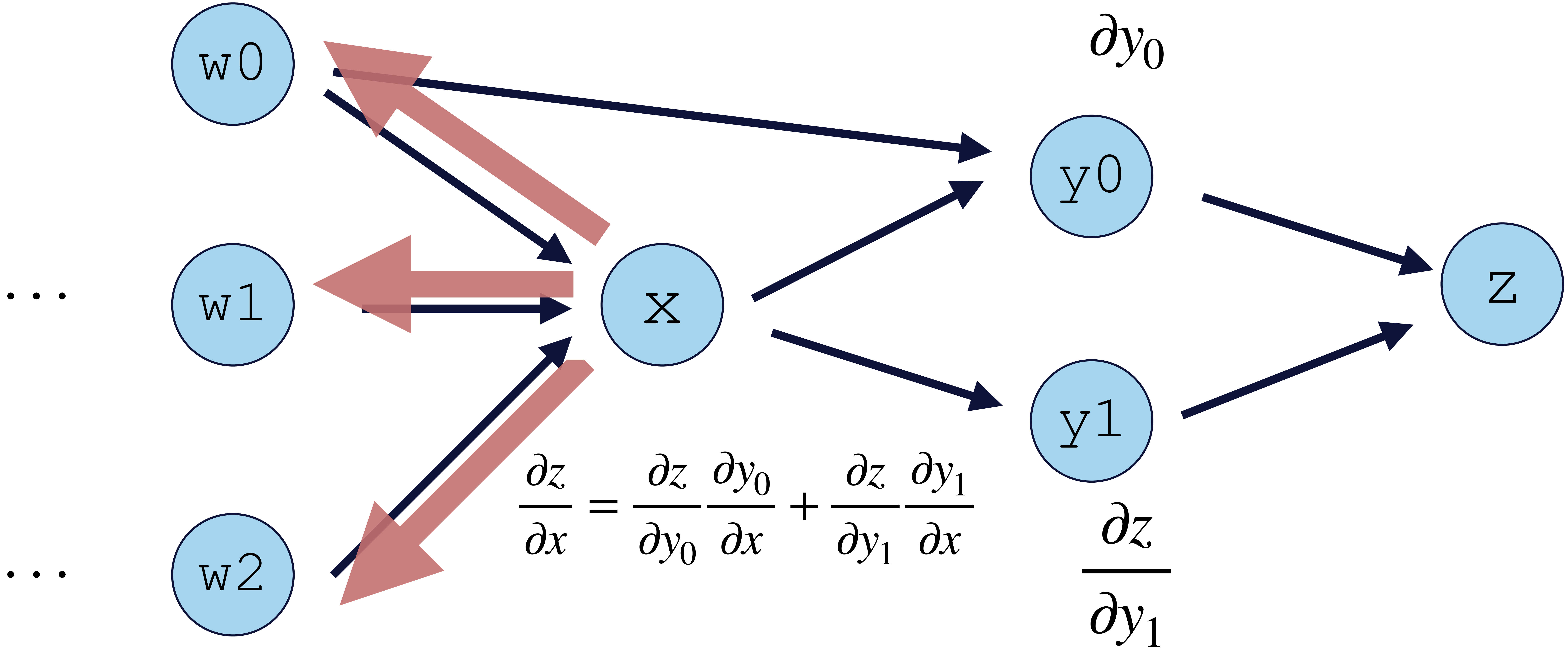
$$\frac{\partial z}{\partial y_0}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_0}\frac{\partial y_0}{\partial x} + \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x}$$

$$\frac{\partial z}{\partial y_1}$$

REVERSE-MODE AUTOMATIC DIFFERENTIATION = A GREEDY PATH FACTORIZATION ALGORITHM

$$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial y_0}\frac{\partial y_0}{\partial w_0} + \frac{\partial z}{\partial x}\frac{\partial x}{\partial w_0}$$



$$\frac{\partial z}{\partial y_0}$$

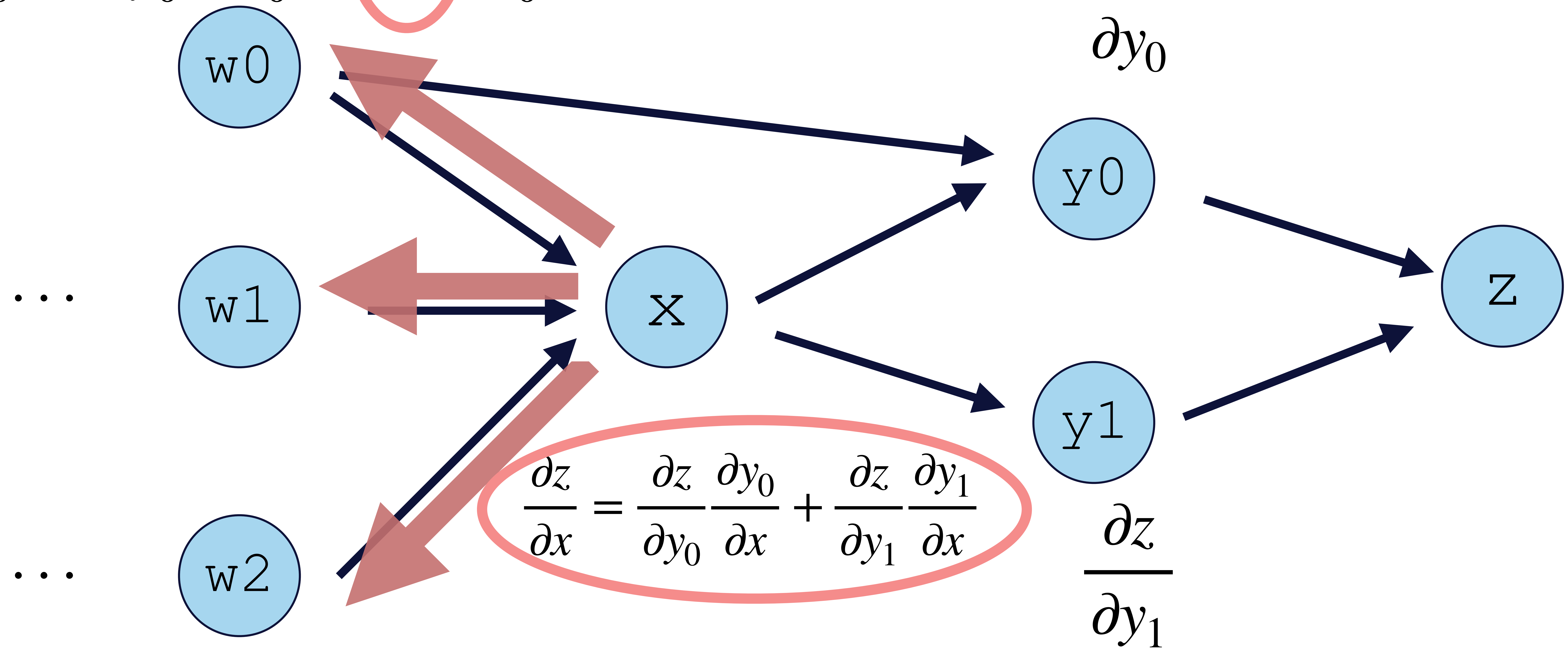
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_0}\frac{\partial y_0}{\partial x} + \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x}$$

$$\frac{\partial z}{\partial y_1}$$

REVERSE-MODE AUTOMATIC DIFFERENTIATION = A GREEDY PATH FACTORIZATION ALGORITHM

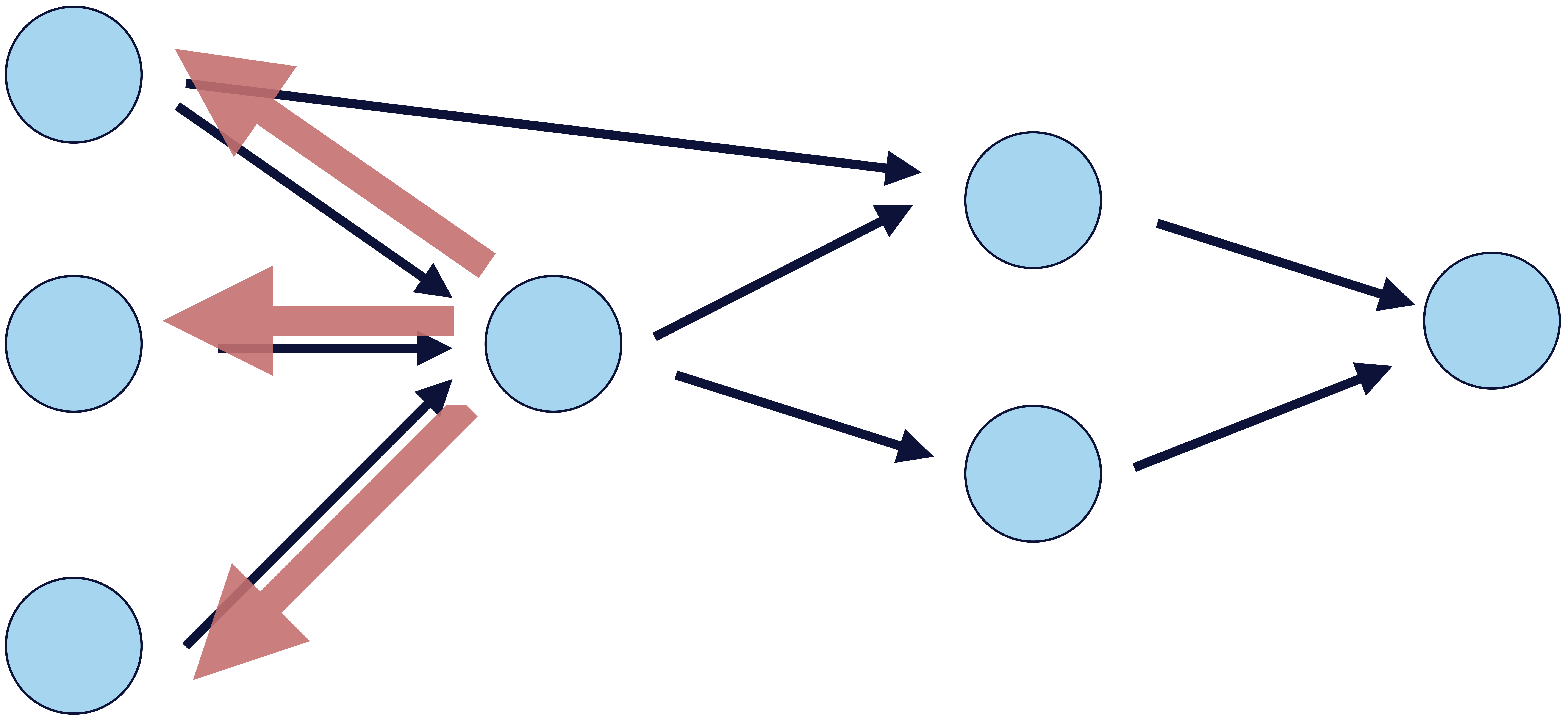
SIGGRAPH THINK BEYOND 2020 S2020.SIGGRAPH.ORG

$$\frac{\partial z}{\partial w_0} = \frac{\partial z}{\partial y_0}\frac{\partial y_0}{\partial w_0} + \frac{\partial z}{\partial x}\frac{\partial x}{\partial w_0}$$



REVERSE-MODE AUTOMATIC DIFFERENTIATION PRODUCES EFFICIENT GRADIENTS

- gradient complexity: number of edges * constant
  - same as directly computing the function ("*cheap gradient principle*")



TRANSFORMING LOOPS WITH REVERSE MODE

- remember every intermediate values in the forward pass, then run the loop backward
  - also works for recursion
  - unbounded memory usage

```
function f(x):
  result = x
  for i = 1 to 8:
    result = exp(result)
  return result
```

→

```
function d_f(x):
  result = x
  results = []
  for i = 1 to 8:
    results.push(result)
    result = exp(result)

  for i = 8 to 1:
    d_results = d_result *
      exp(results[i])
  return result
```

SOURCE TRANSFORM V.S. TAPING

- a spectrum: how much is done at compile time
  - similar to (tracing) JIT v.s. static compile

source transform

```
function f(x):
    …
```

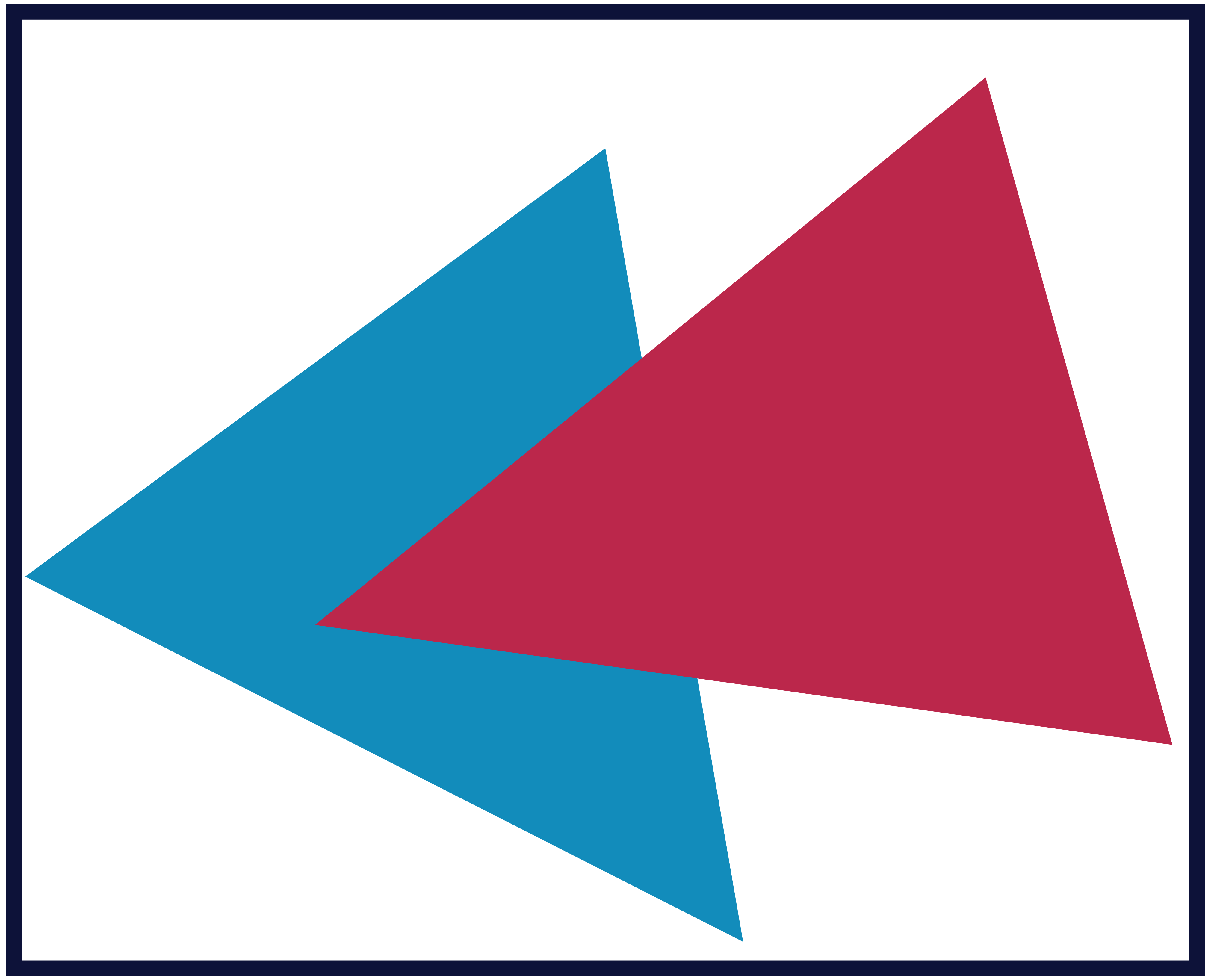
```
function d_f(x):
    …
    …
    …
```

trace

```
f(5)
```

DIFFERENTIATING CONDITIONALS

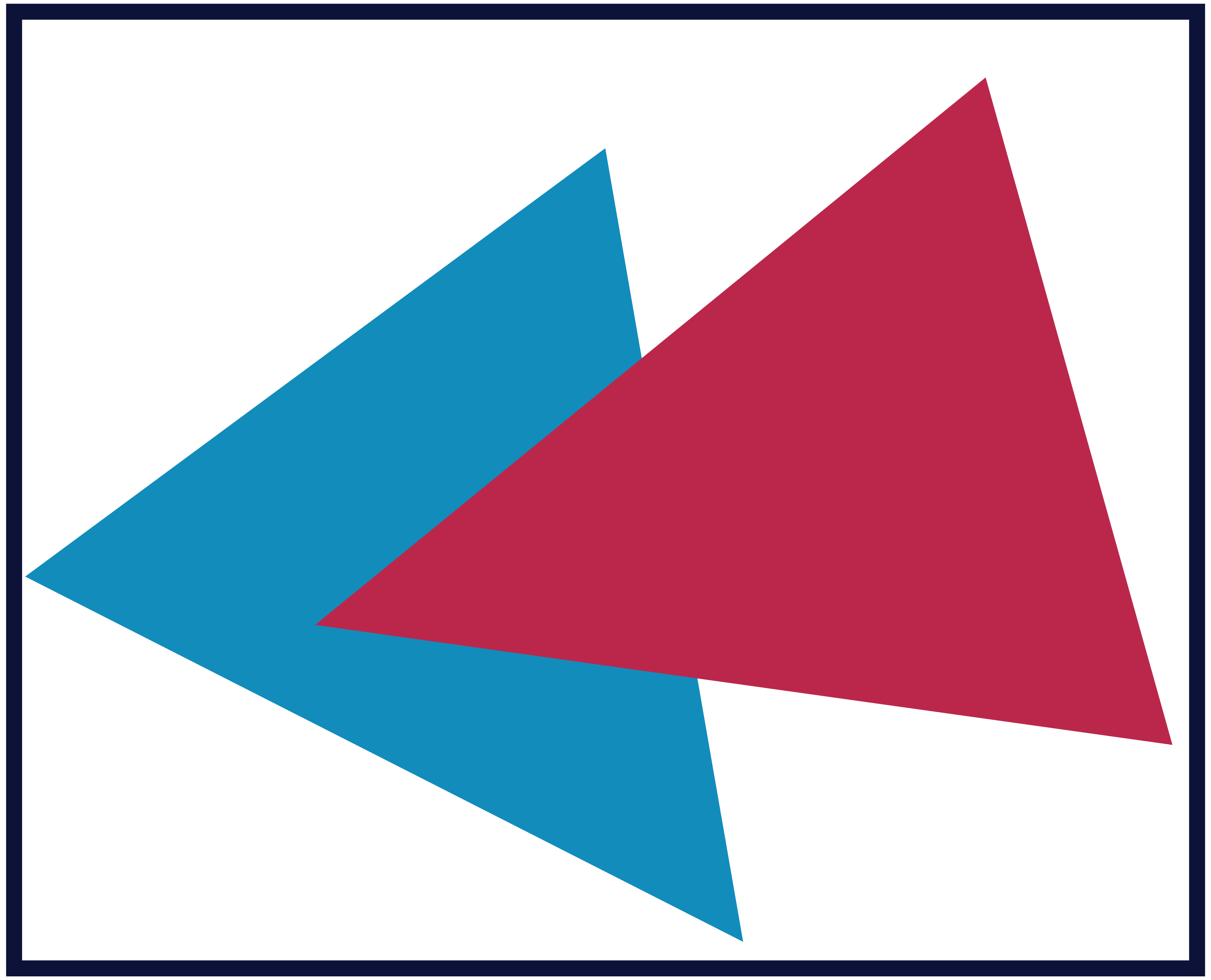
```
if (hit the red triangle)
   return red
elif (hit the blue triangle)
   return blue
else
   return white
```



DIFFERENTIATING CONDITIONALS

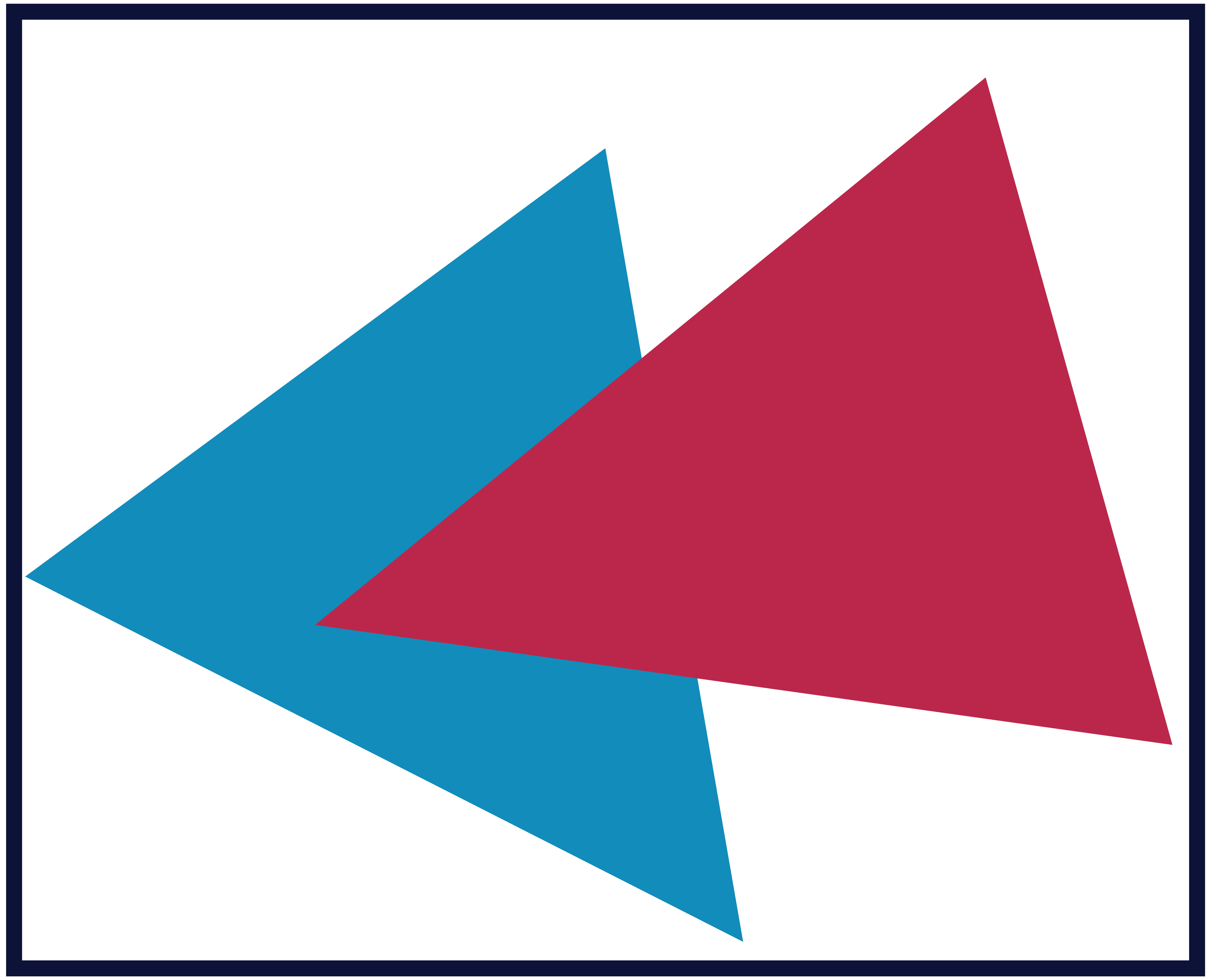
```
if (hit the red triangle)
  return red
elif (hit the blue triangle)
  return blue
else
  return white
```

- derivative of color w.r.t. triangle vertex is 0



DIFFERENTIATING CONDITIONALS

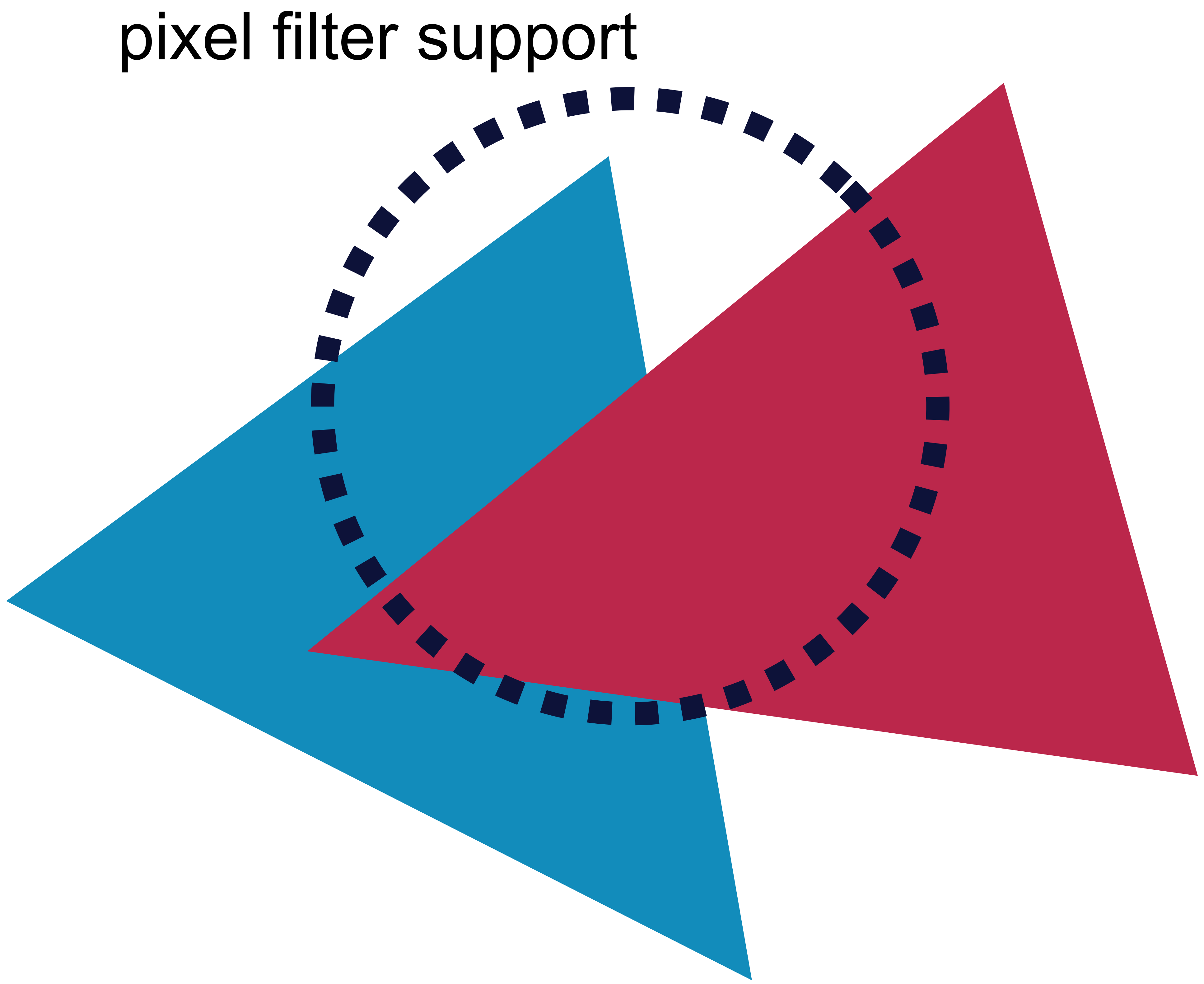
```
if (hit the red triangle)
  return red
elif (hit the blue triangle)
  return blue
else
  return white
```



- derivative of color w.r.t. triangle vertex is 0
  - or is it?

RENDERING = COMPUTING INTEGRALS

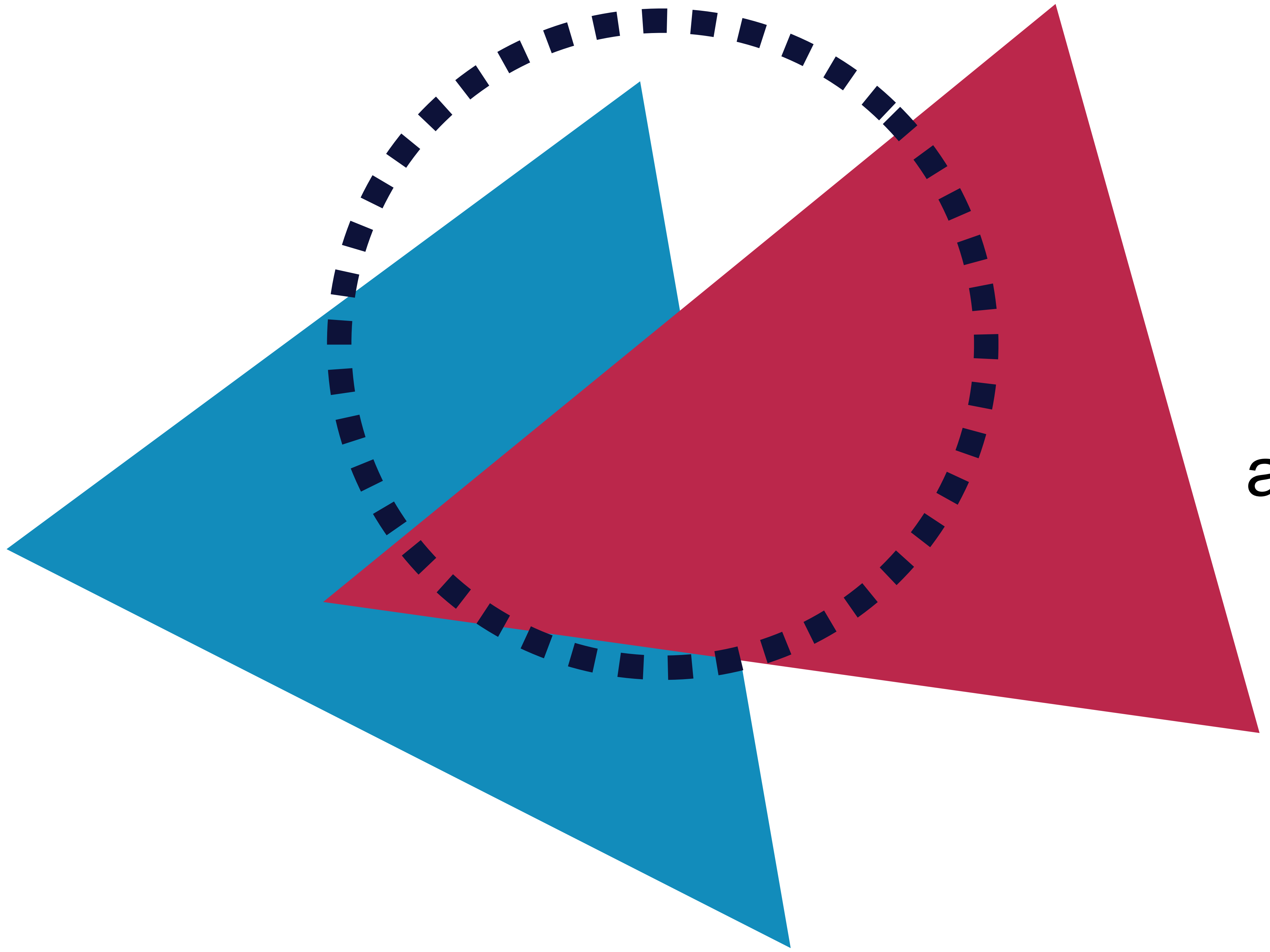
- pixel color is defined by the average color over an area
  - aka anti-aliasing



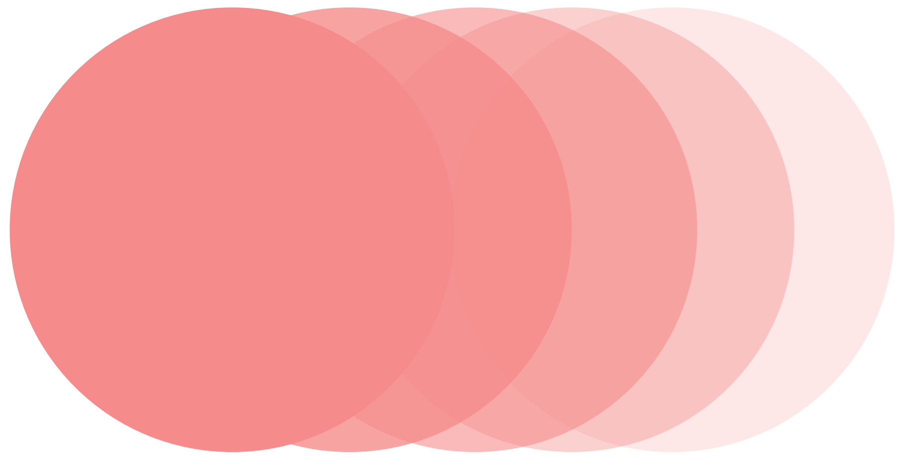
RENDERING = COMPUTING INTEGRALS

- pixel color is defined by the average color over an area
  - aka anti-aliasing

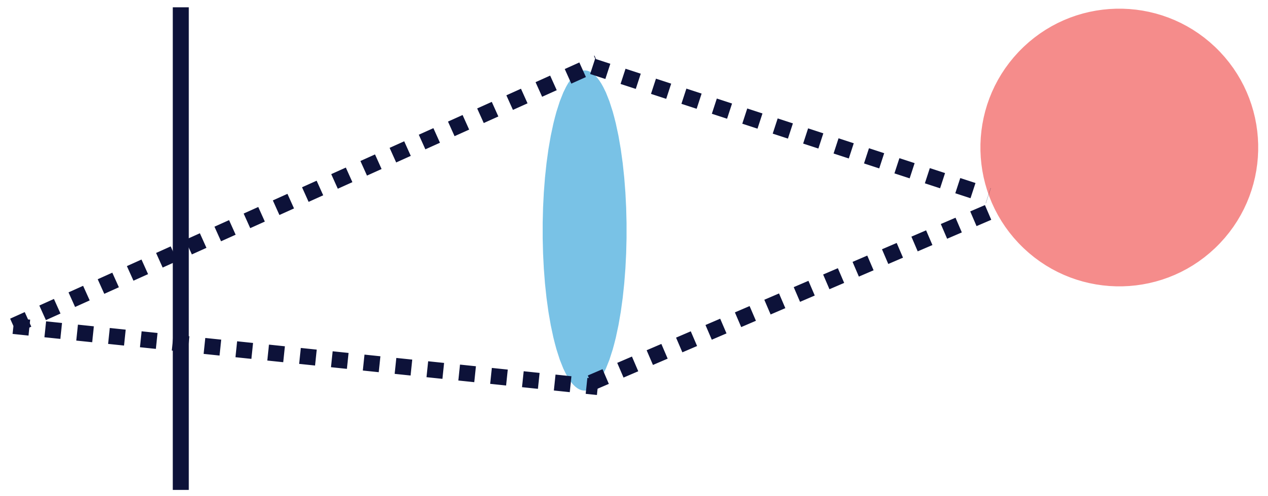
pixel filter support



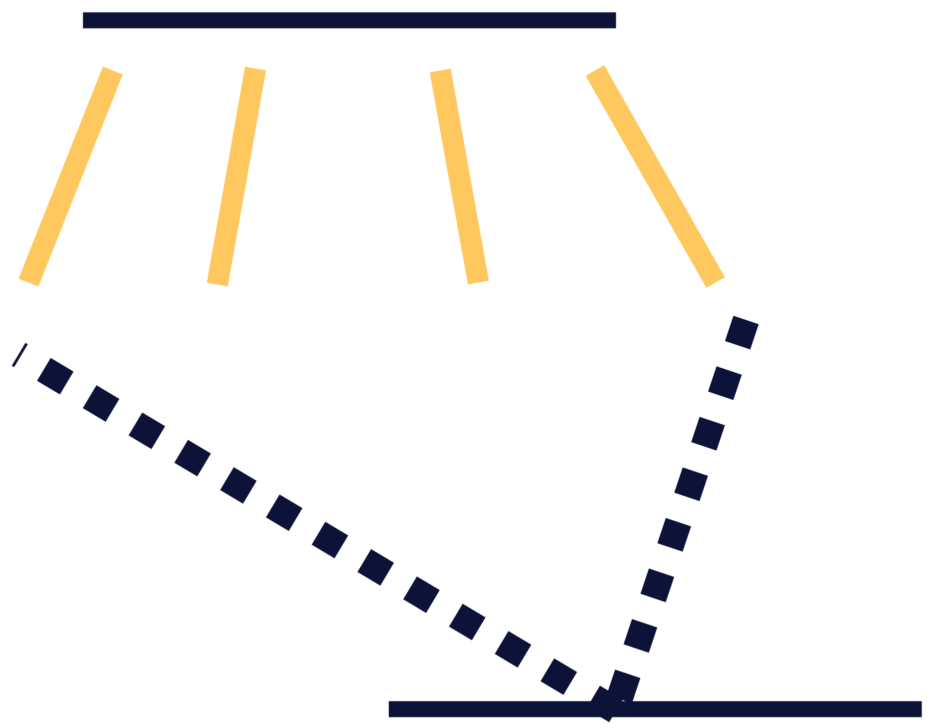
shutter time (motion blur)



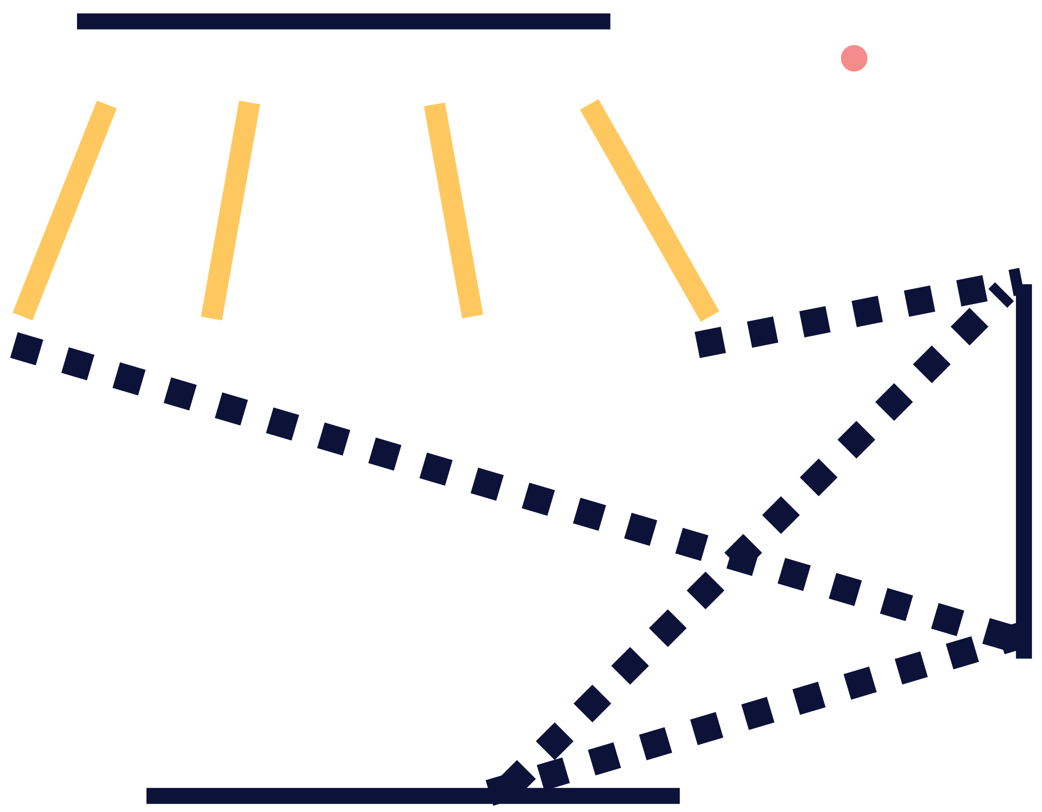
camera aperture (defocus blur)



area light



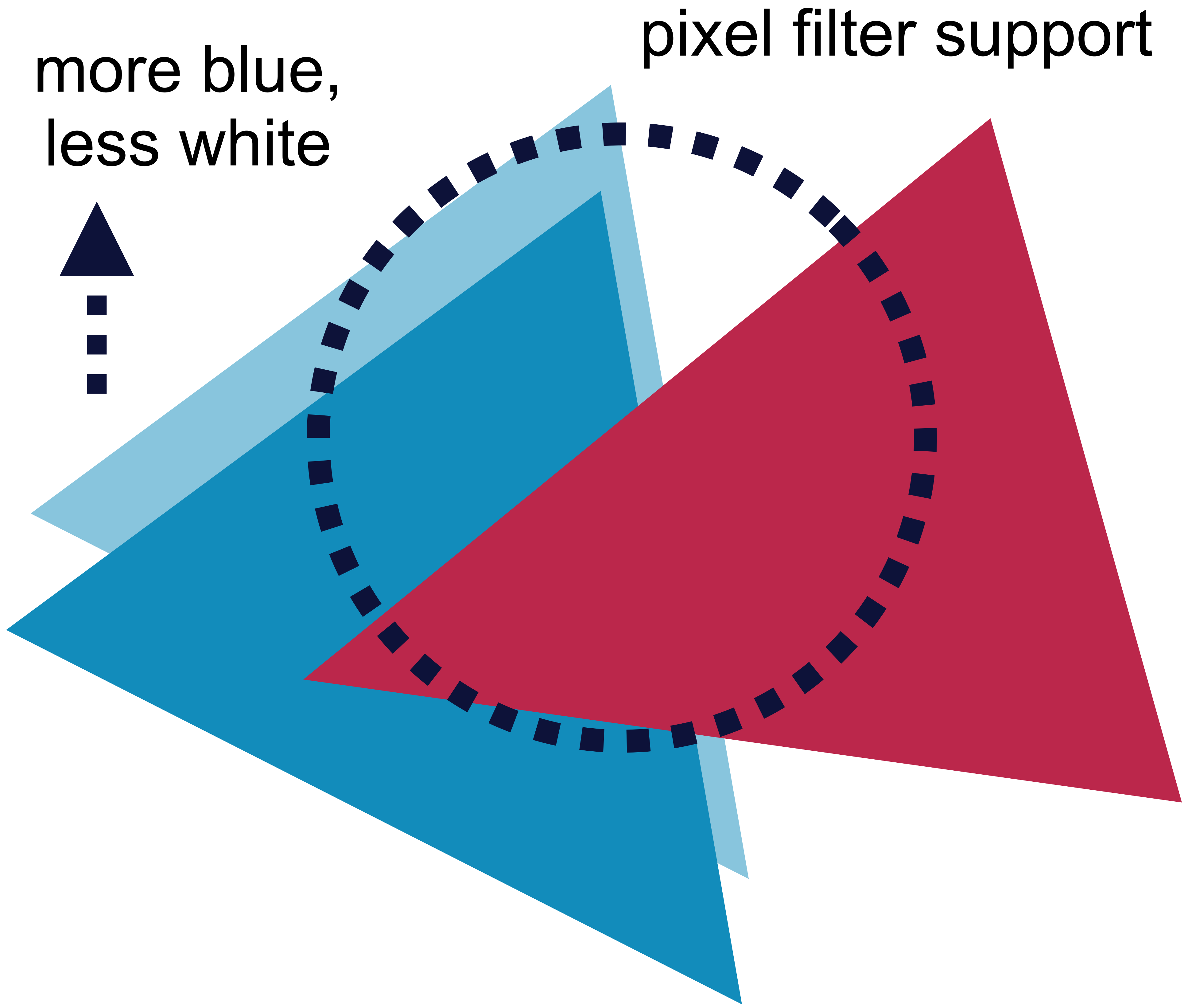
global illumination



- wavelength
- participating media
- …
- and more!

THE RENDERING INTEGRALS ARE DIFFERENTIABLE!

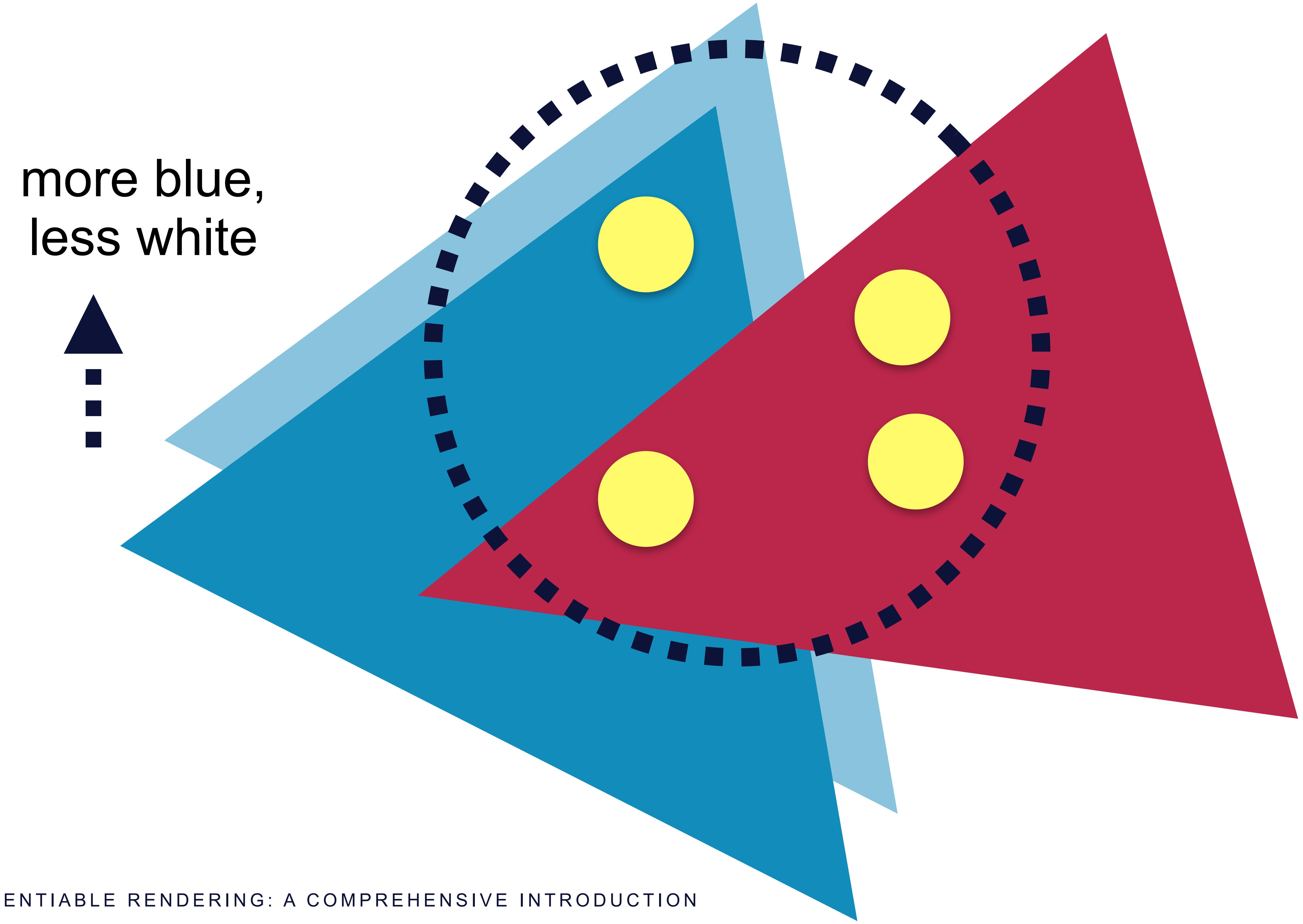
- **While the *integrand* is discontinuous, the *integral* is differentiable!**
  - the average color changes continuously as triangles move




```
∫ if (hit the red triangle)
    return red
  elif (hit the blue triangle)
    return blue
  else
    return white
```

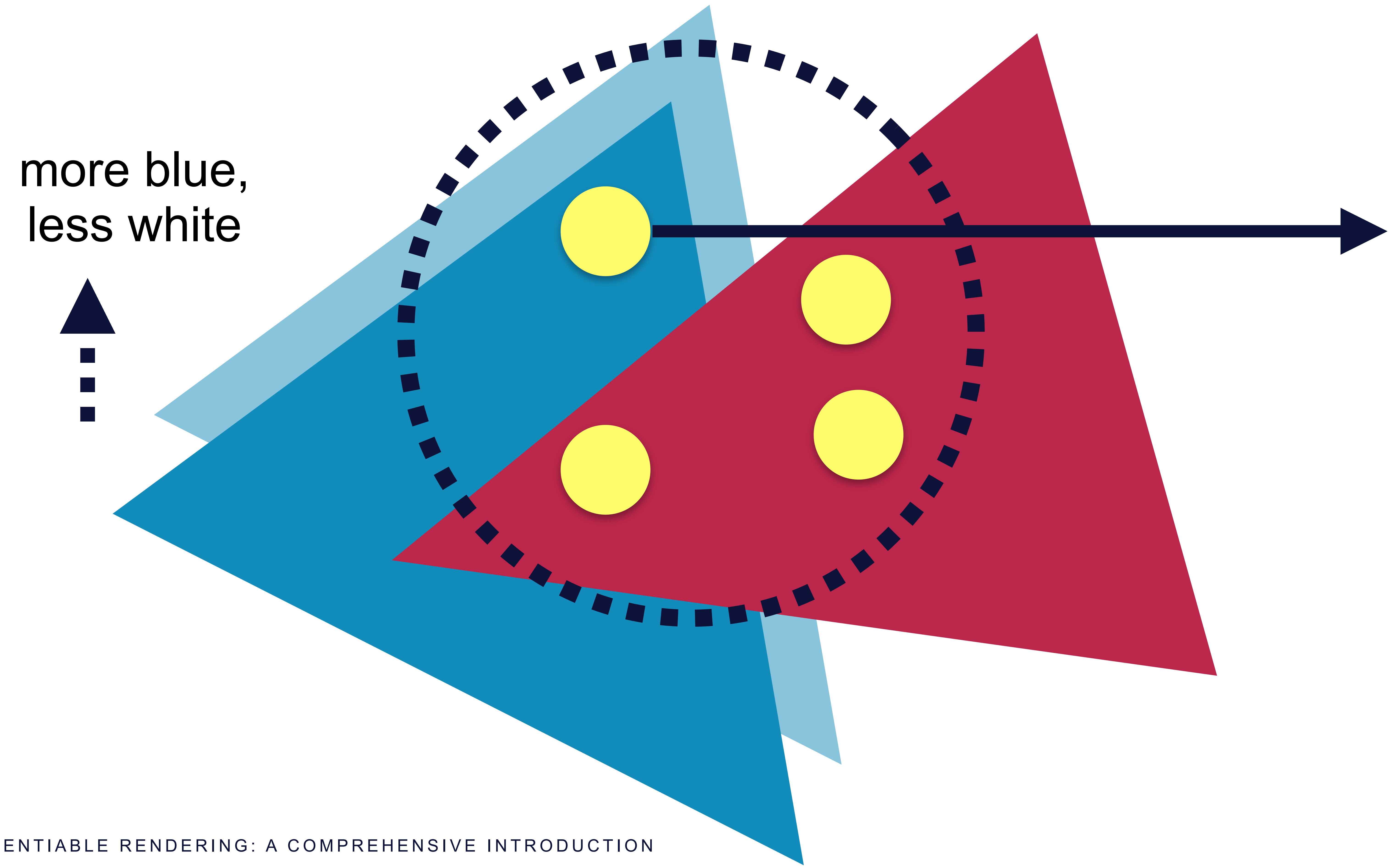
RENDERING = SAMPLING INTEGRALS

- We evaluate these integrals by sampling them



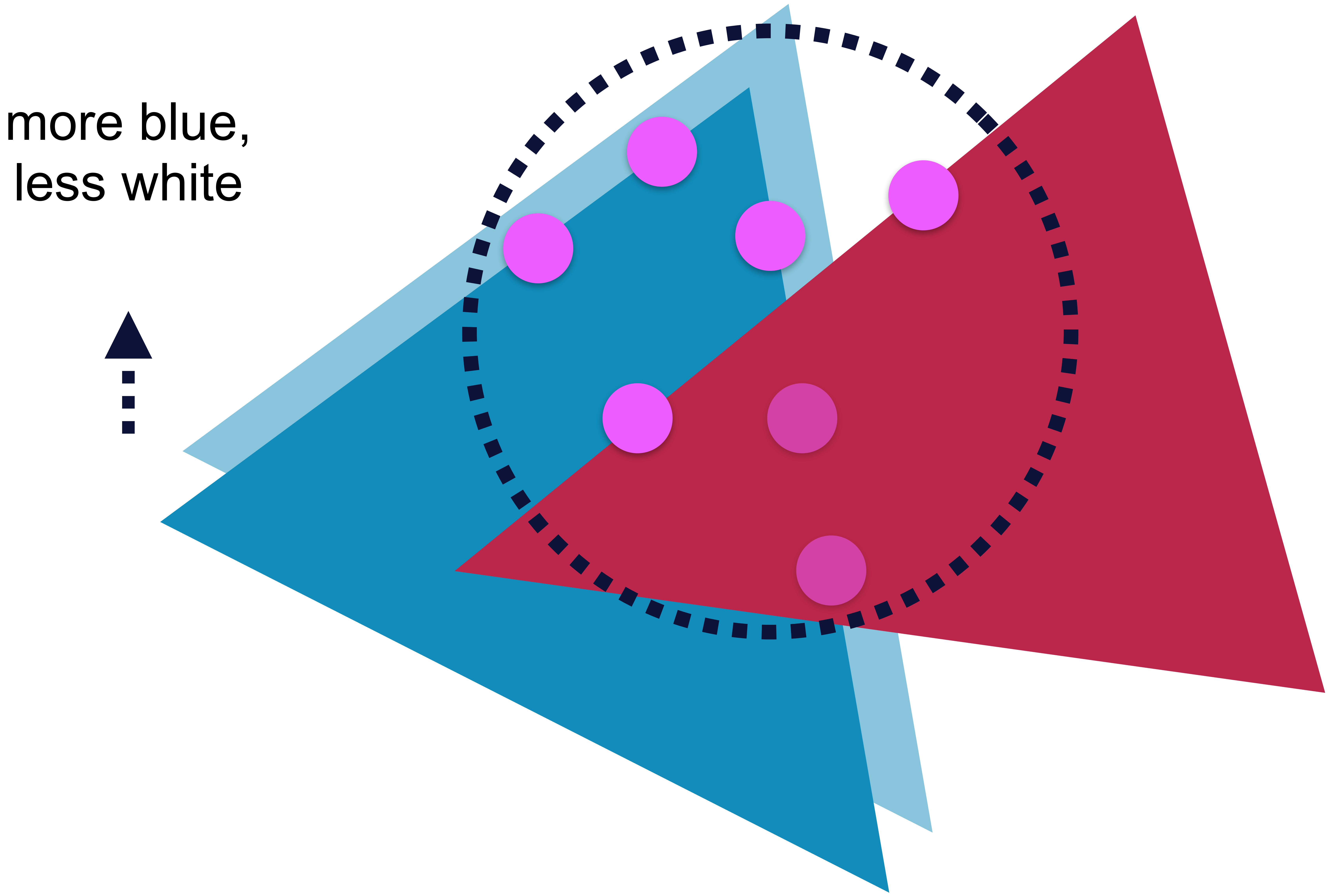
$$\int \ldots \approx \sum \ldots$$

DIFFERENTIATING INTEGRAL SAMPLES GIVES WRONG DERIVATIVES



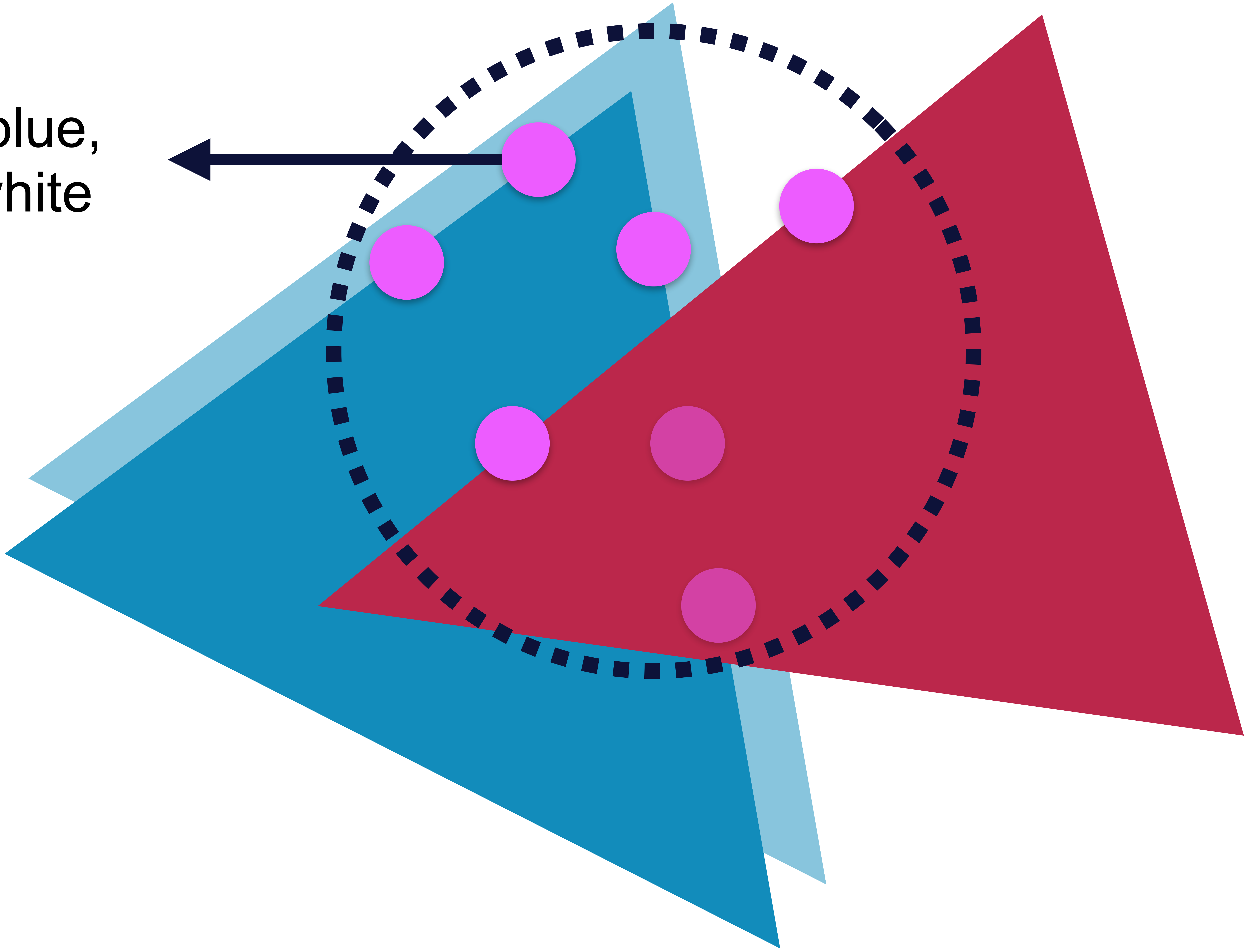
$$\frac{\partial}{\partial p} = 0$$

KEY IDEA: EXPLICITLY INTEGRATE THE BOUNDARIES

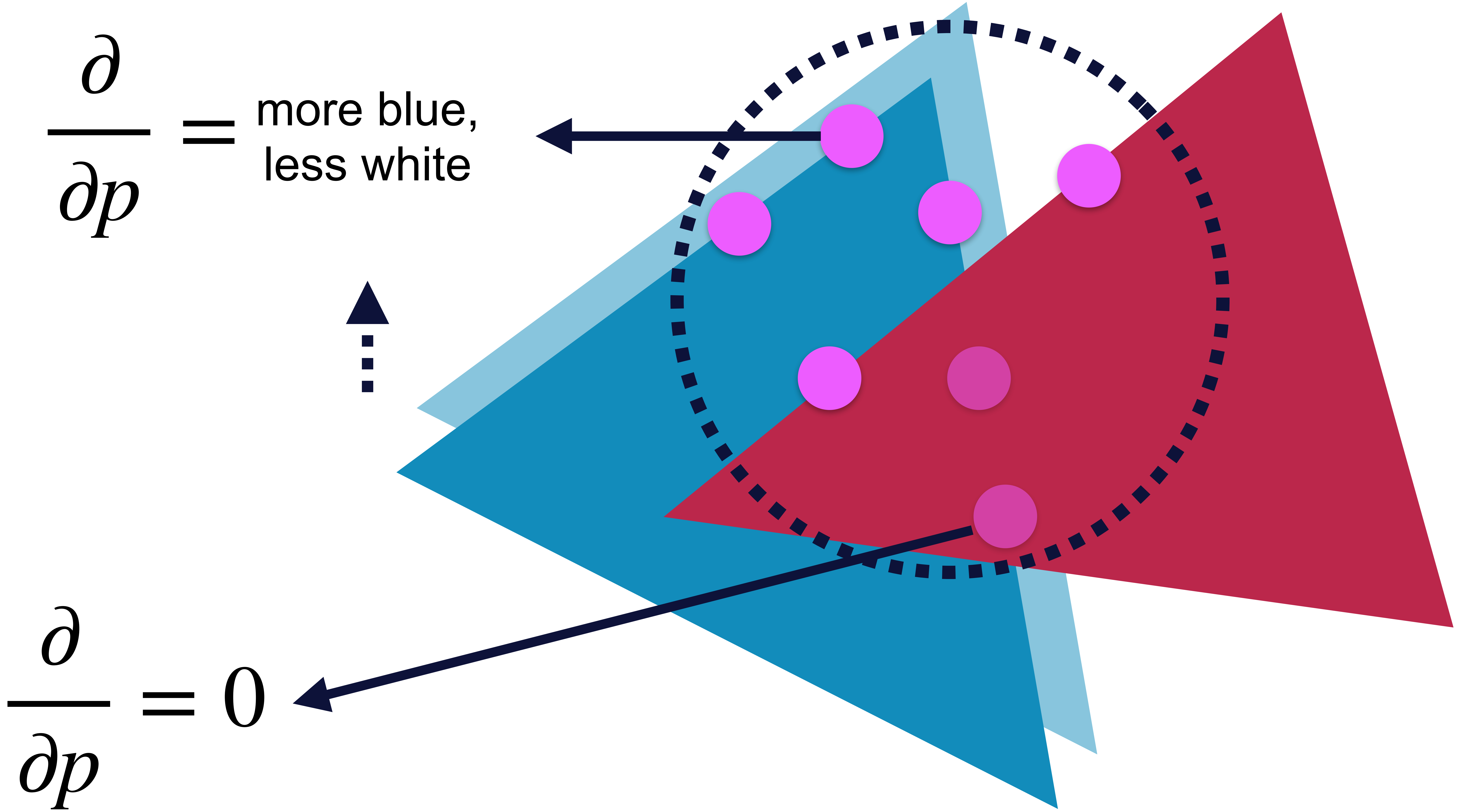


KEY IDEA: EXPLICITLY INTEGRATE THE BOUNDARIES

$$\frac{\partial}{\partial p} = \text{more blue, less white}$$

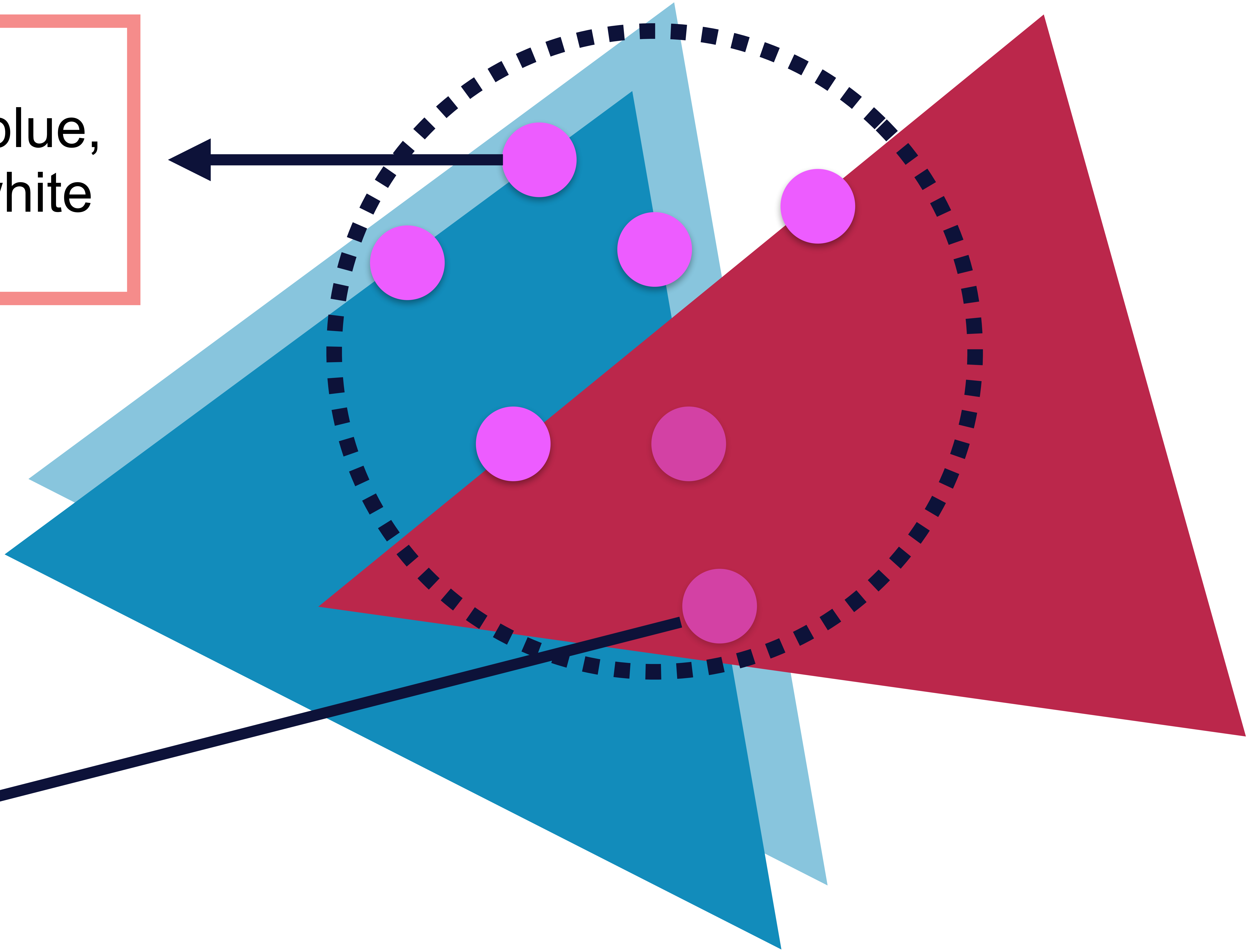


KEY IDEA: EXPLICITLY INTEGRATE THE BOUNDARIES



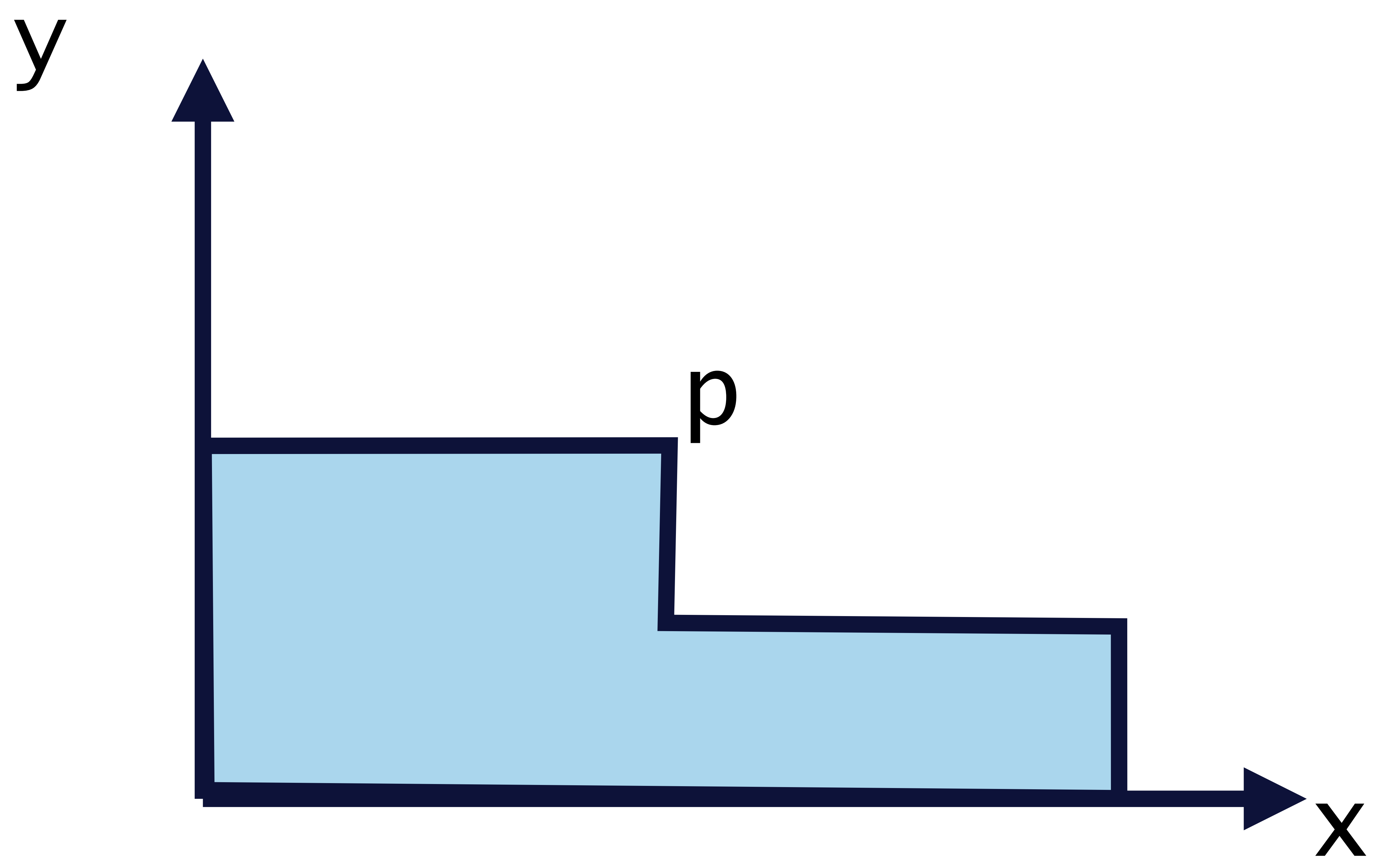
KEY IDEA: EXPLICITLY INTEGRATE THE BOUNDARIES

$$\frac{\partial}{\partial p} = \text{more blue, less white}$$



$$\frac{\partial}{\partial p} = 0$$

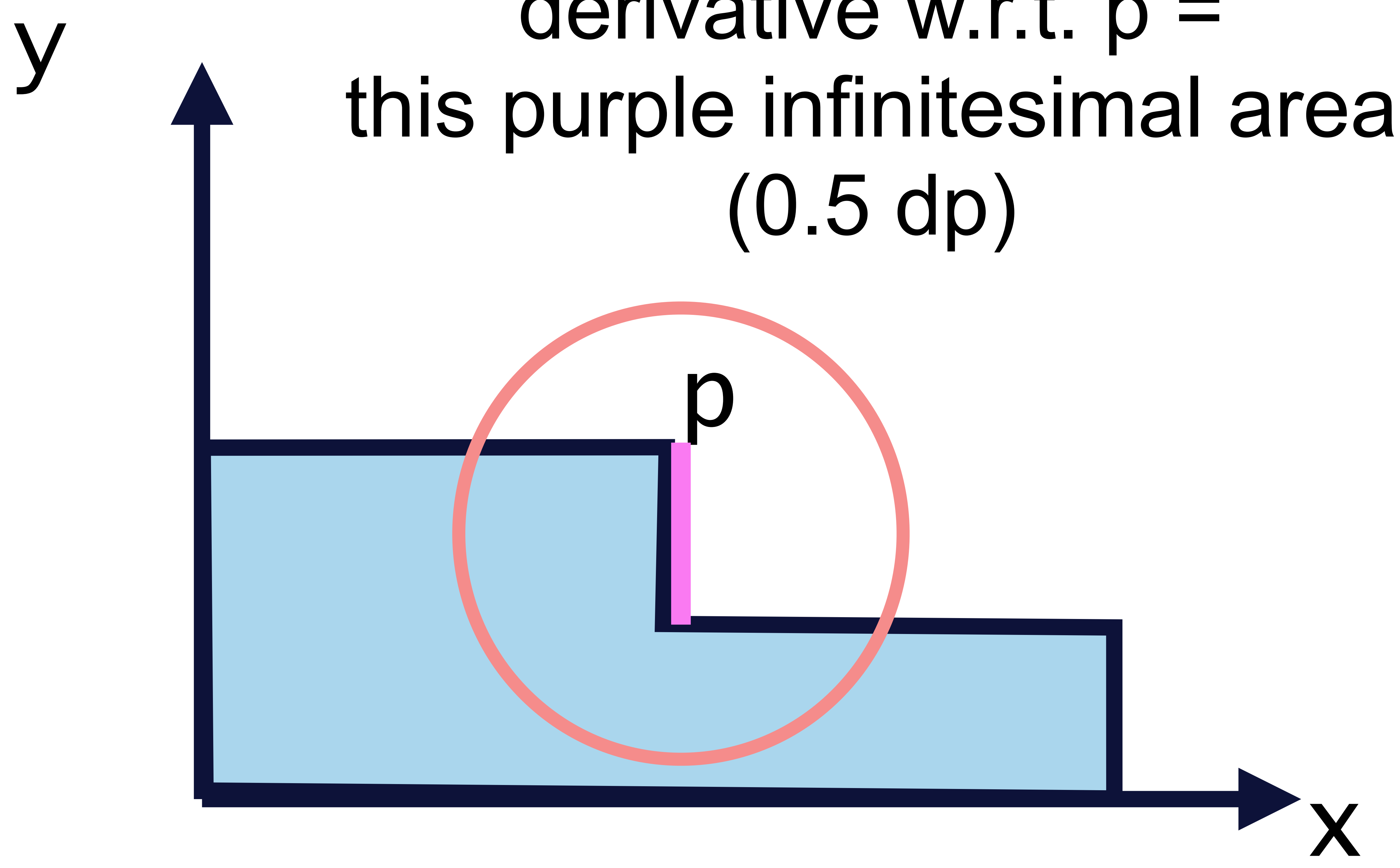
LET'S DERIVE THE DERIVATIVES IN 1D



(the blue area)

$$\int_{x=0}^{x=1} \texttt{x < p ? 1 : 0.5}$$

LET'S DERIVE THE DERIVATIVES IN 1D



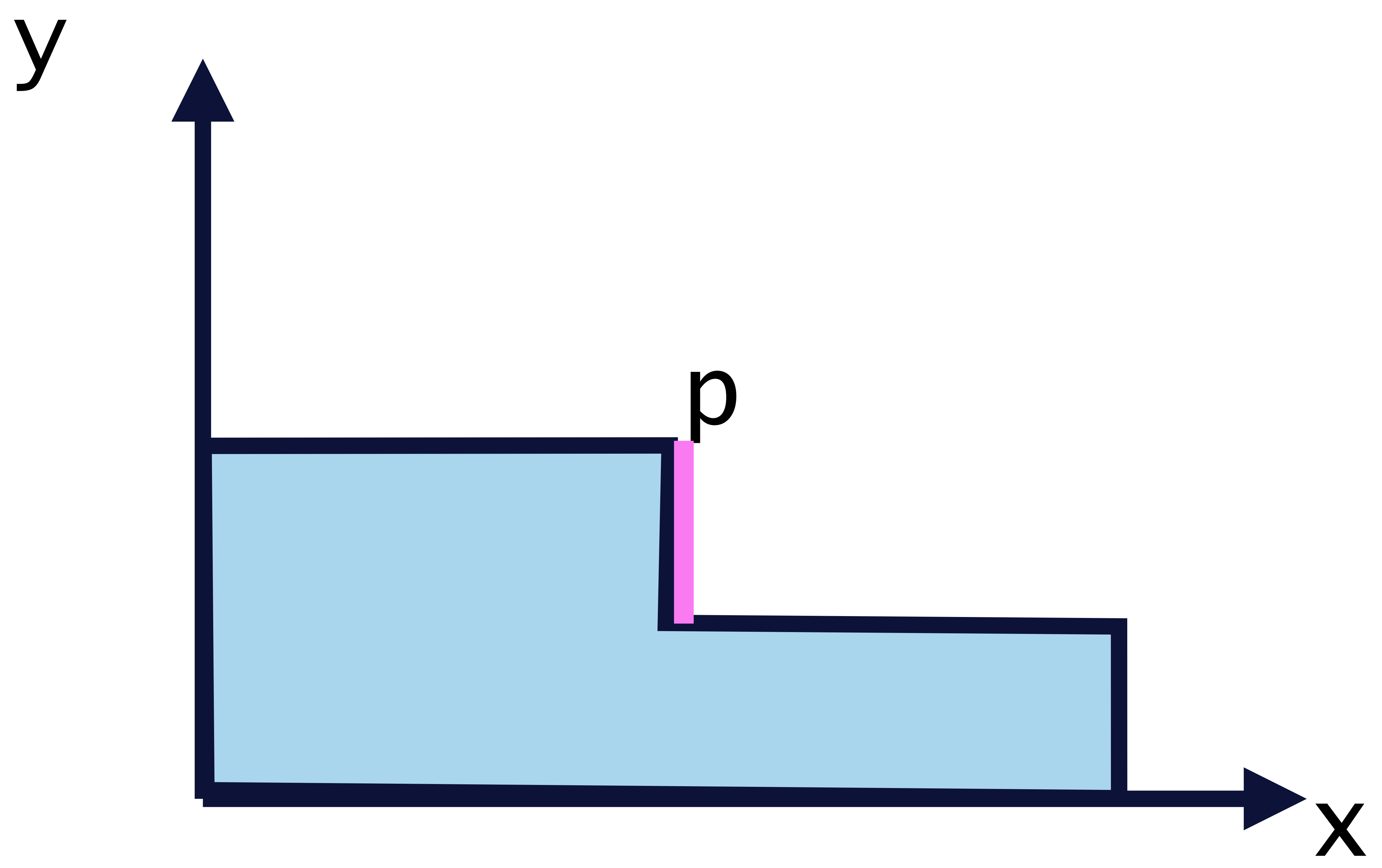
derivative w.r.t. p =
this purple infinitesimal area
(0.5 dp)

(the blue area)

$$\int_{x=0}^{x=1} \texttt{x < p ? 1 : 0.5}$$

LET'S DERIVE THE DERIVATIVES IN 1D

- Trick: move the discontinuities to the integral boundaries



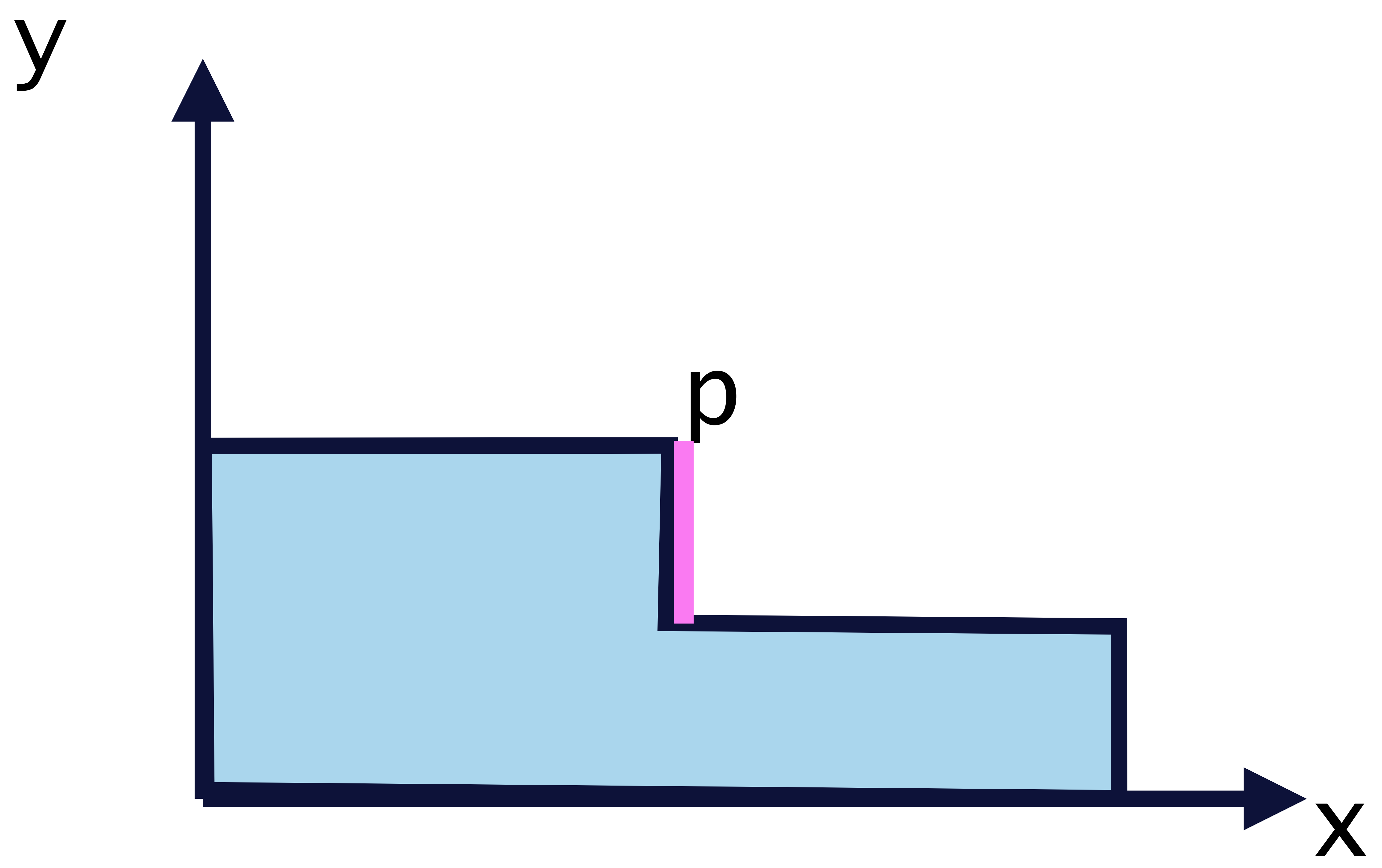
(the blue area)

$$\int_{x=0}^{x=1} \texttt{x < p ? 1 : 0.5}$$

$$= \int_{x=0}^{x=p} 1 + \int_{x=p}^{x=1} 0.5$$

LET'S DERIVE THE DERIVATIVES IN 1D

- Trick: move the discontinuities to the integral boundaries



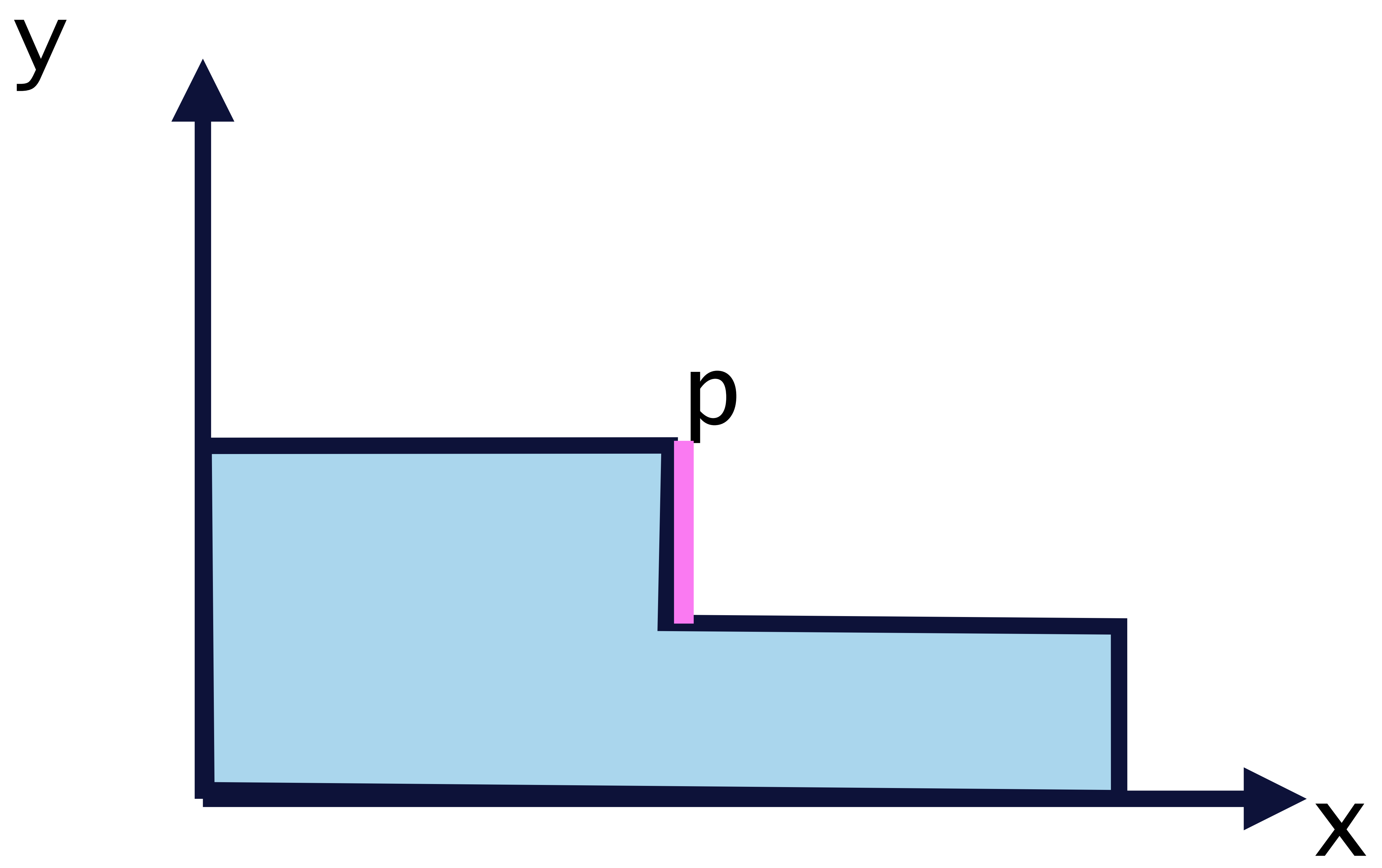
(the blue area)

$$\int_{x=0}^{x=1} \texttt{x < p ? 1 : 0.5}$$

$$= \boxed{\int_{x=0}^{x=p} 1 + \int_{x=p}^{x=1} 0.5}$$

DISCONTINUITY DERIVATIVES = DIFFERENCES AT DISCONTINUITIES

$$\int_{x=0}^{x=1} \texttt{x < p ? 1 : 0.5}$$



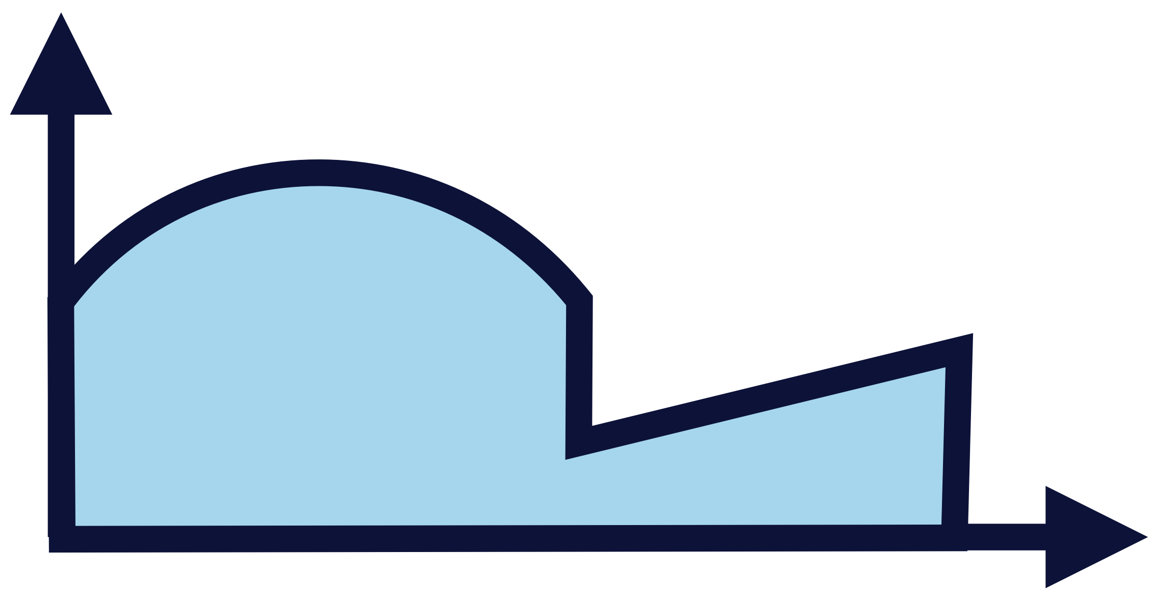
(derivative of blue area w.r.t. p)

$$\frac{\partial}{\partial p}\left(\int_{x=0}^{x=p} 1 + \int_{x=p}^{x=1} 0.5\right)$$

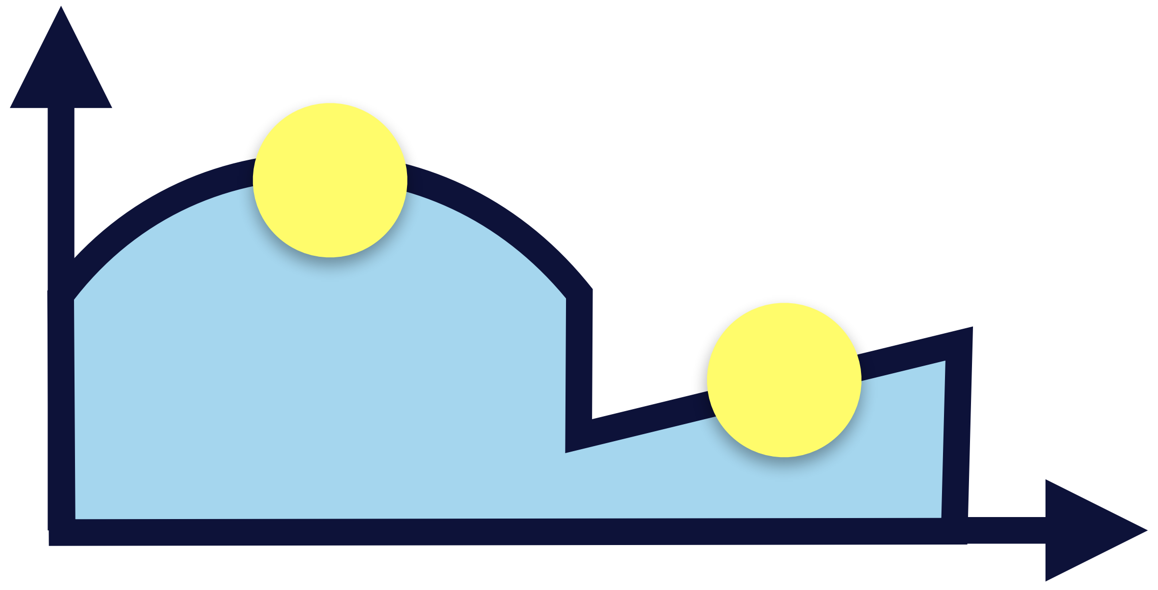
$$= 1 - 0.5$$

DISCONTINUITY DERIVATIVES = DIFFERENCES AT DISCONTINUITIES

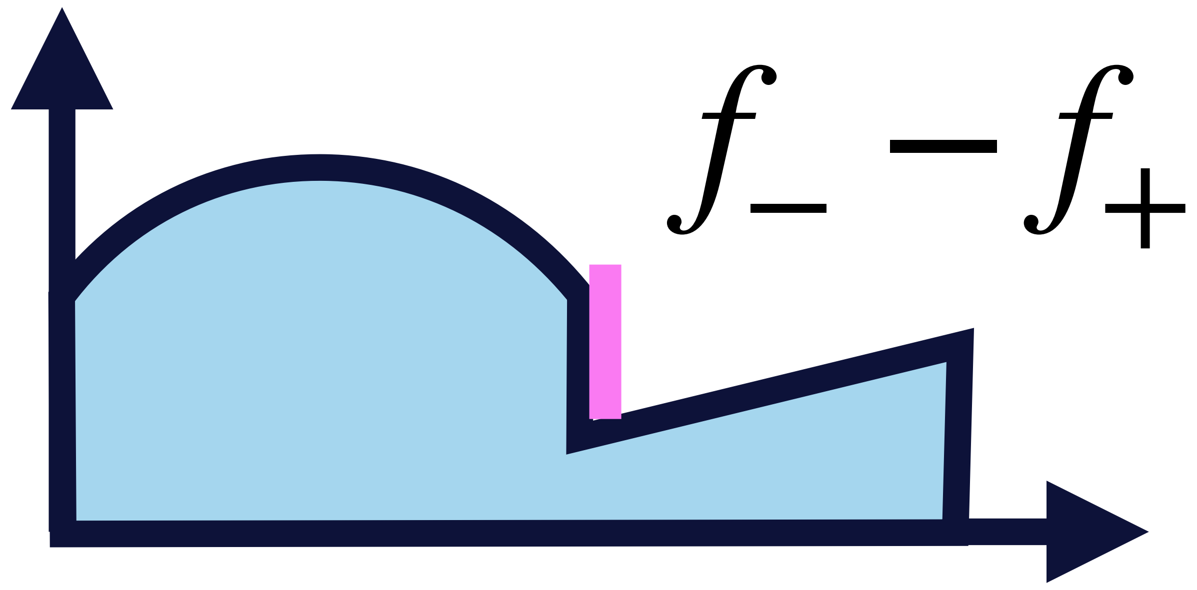
$$\frac{\partial}{\partial p} \int$$



$$= \int \frac{\partial}{\partial p}$$



$$+ \quad \sum$$

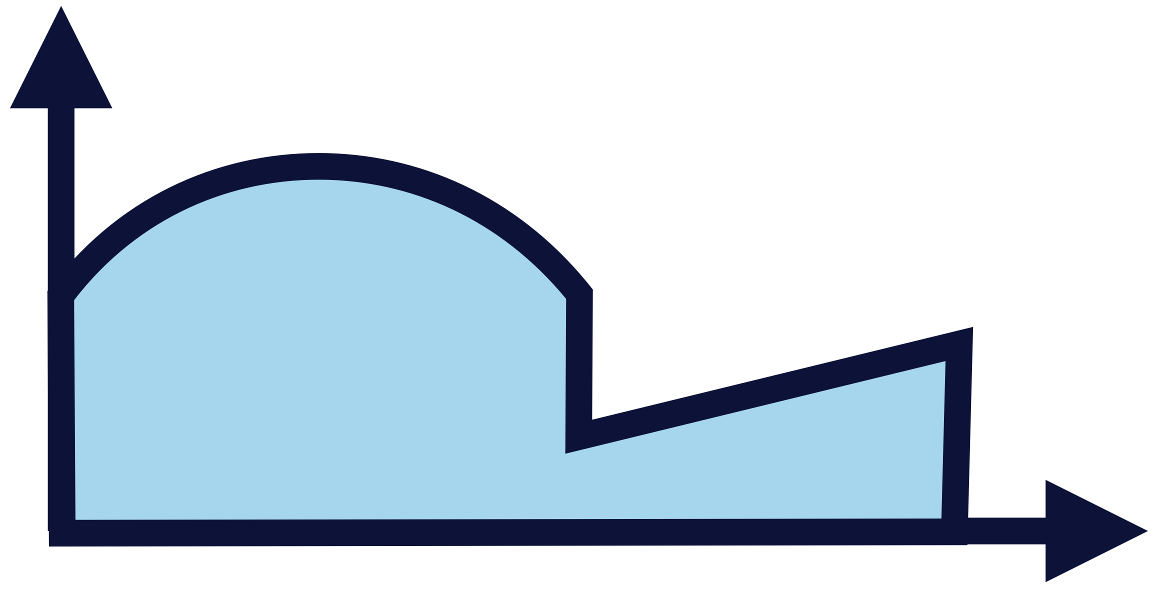


“the Leibniz’s integral rule”

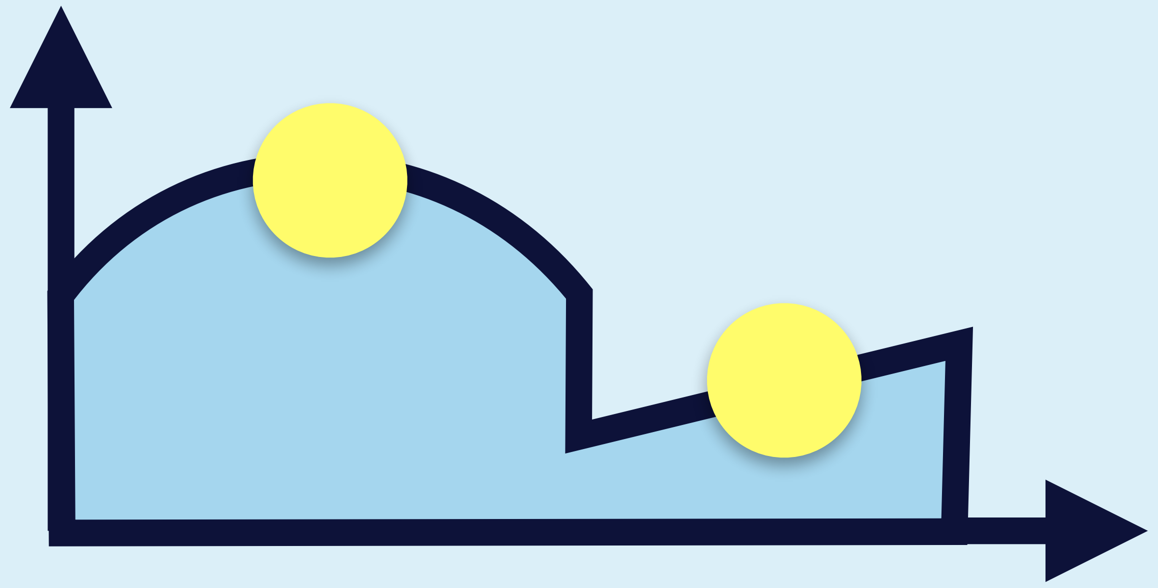
DISCONTINUITY DERIVATIVES = DIFFERENCES AT DISCONTINUITIES

interior derivative

$$\frac{\partial}{\partial p}\int$$

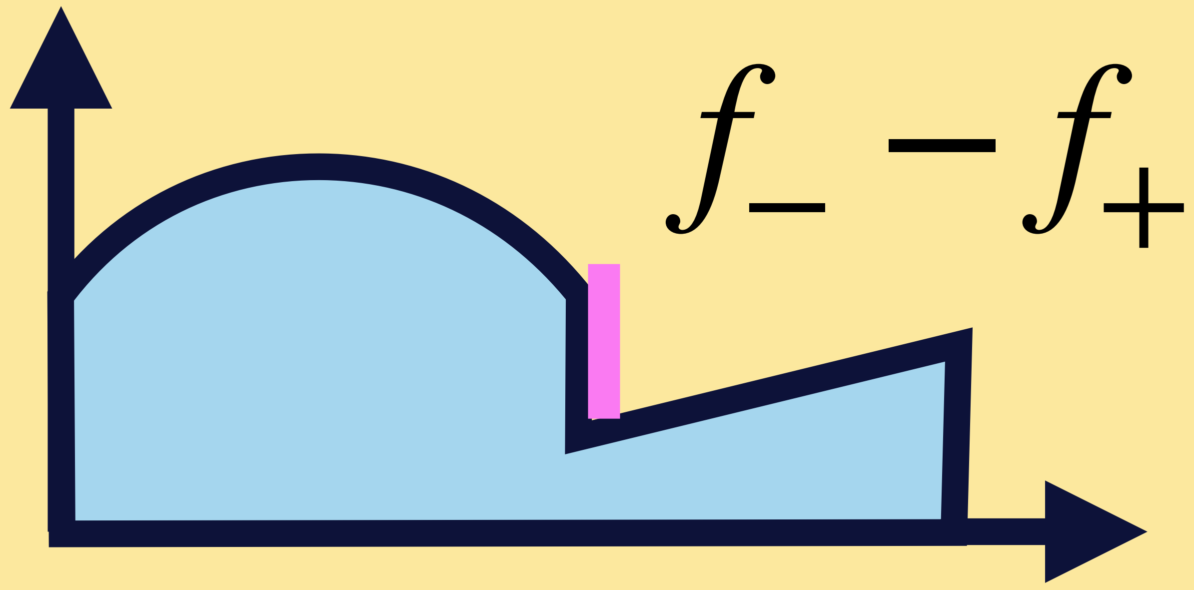


$$= \int \frac{\partial}{\partial p}$$



$$+$$

$$\sum$$



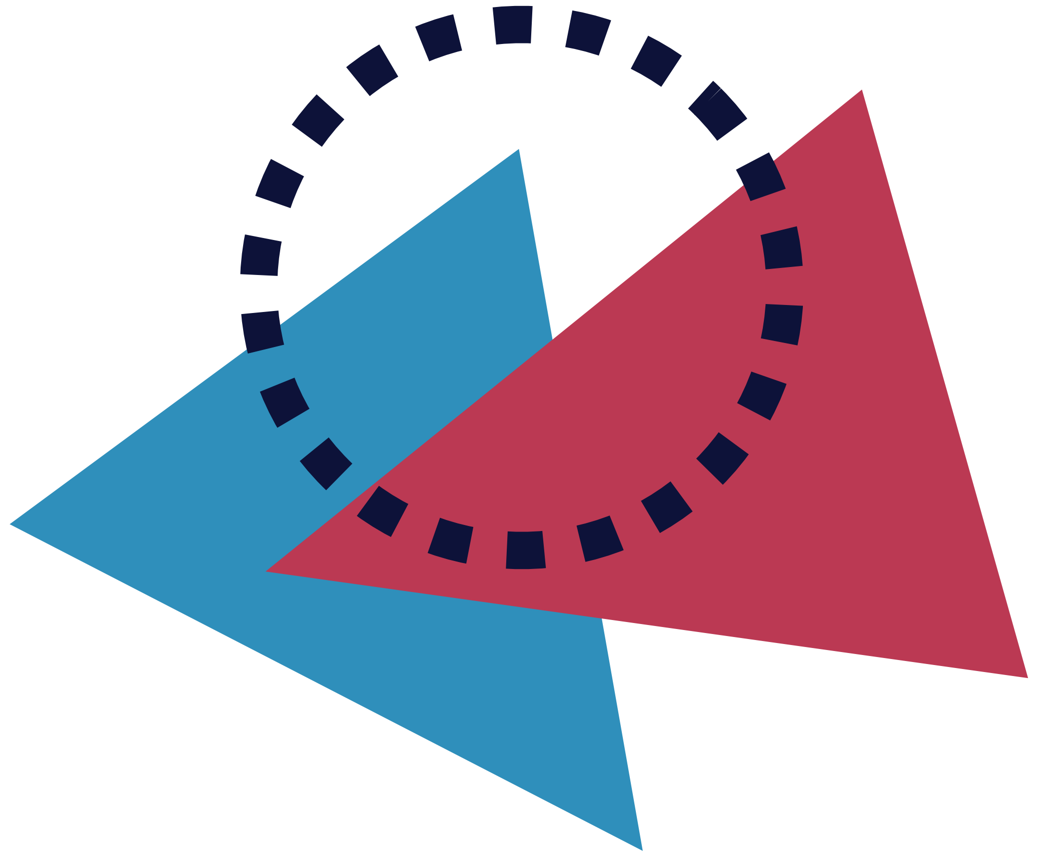
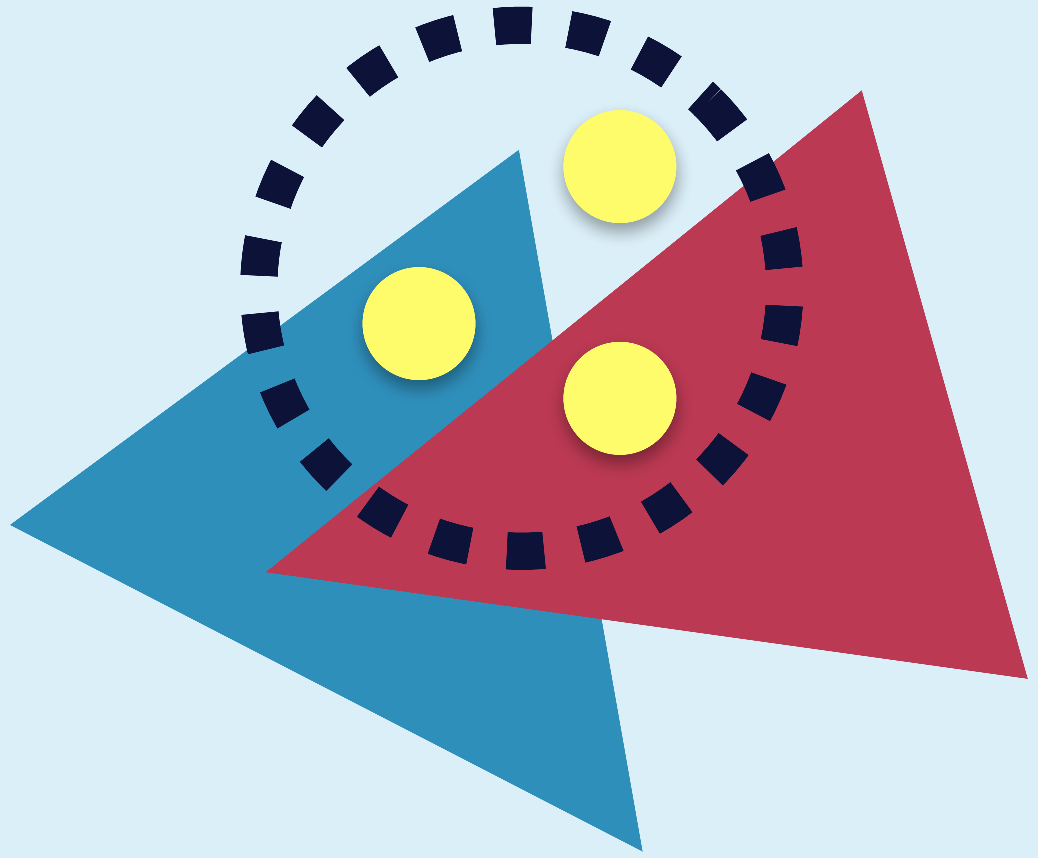
$$f_- - f_+$$

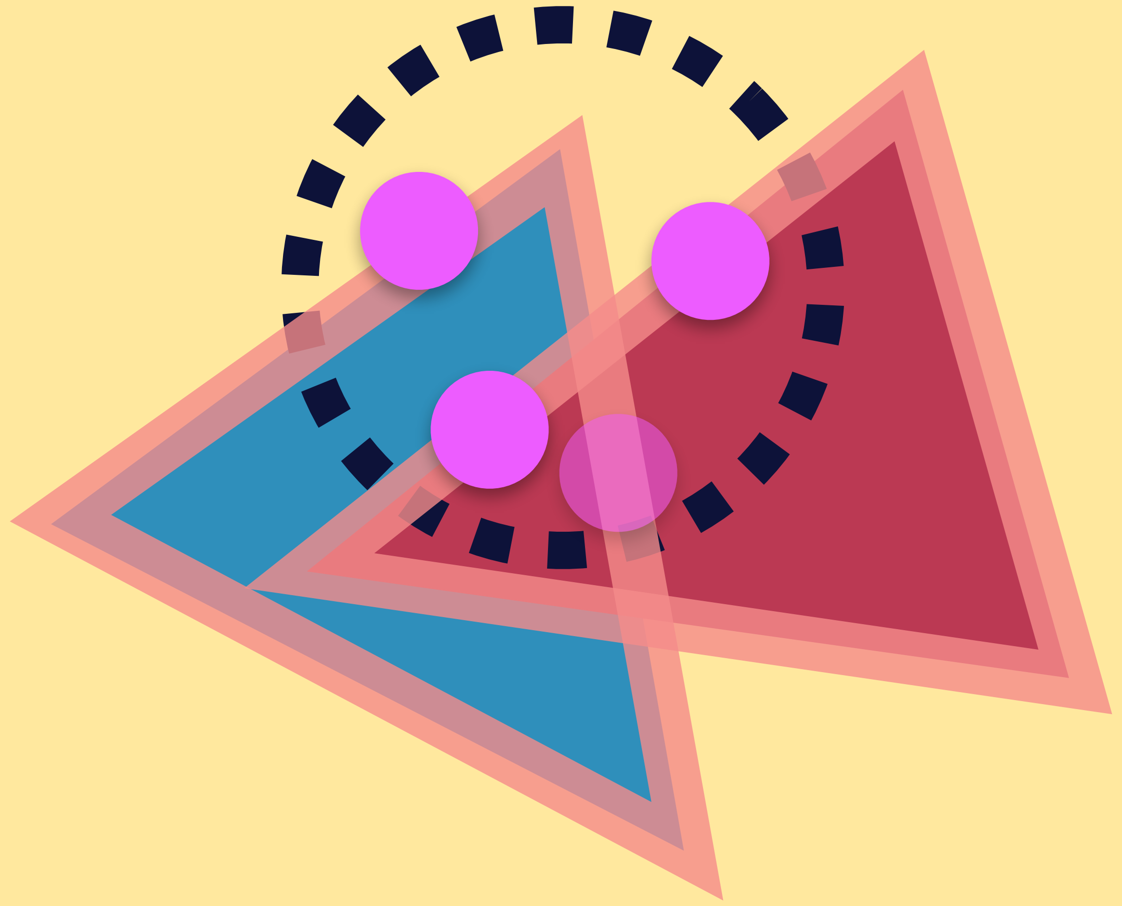
boundary derivative

“the Leibniz’s integral rule”

GENERALIZE TO 2D

interior derivative

$\frac{\partial}{\partial p}\iint$  $= \iint \frac{\partial}{\partial p}$ 

$+ \int$ 

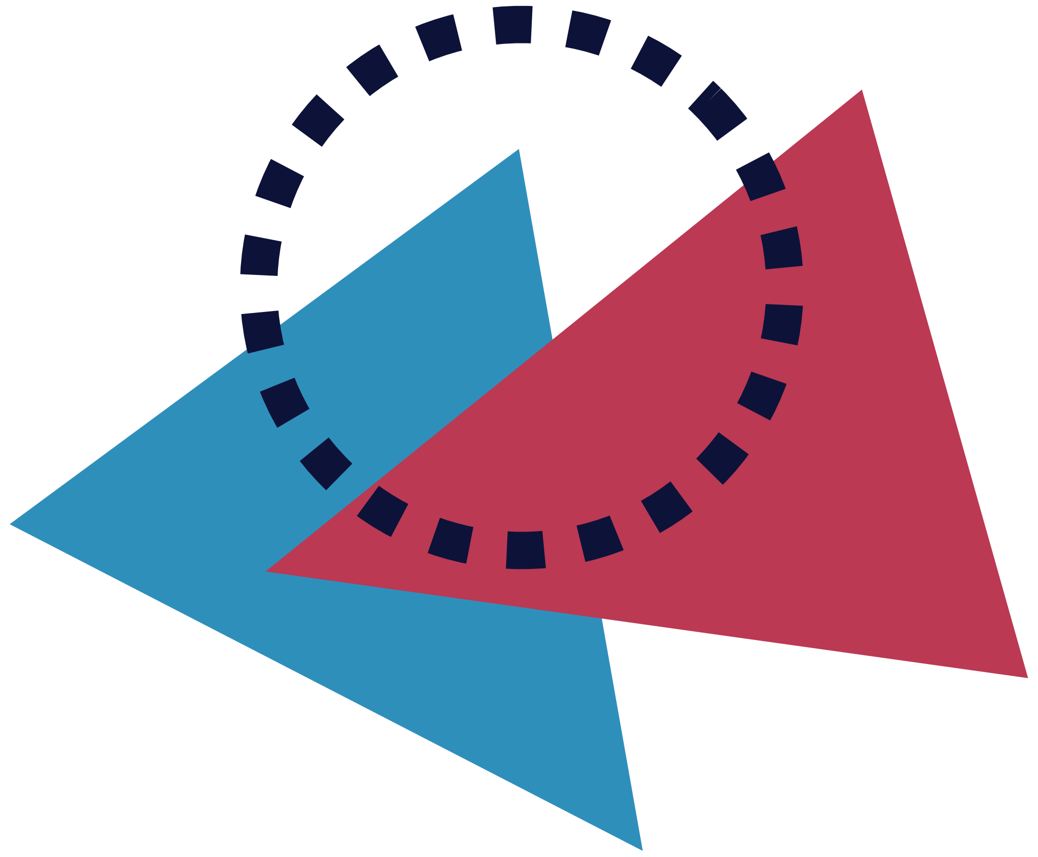
boundary derivative

Reynolds transport theorem
[Reynolds 1903]

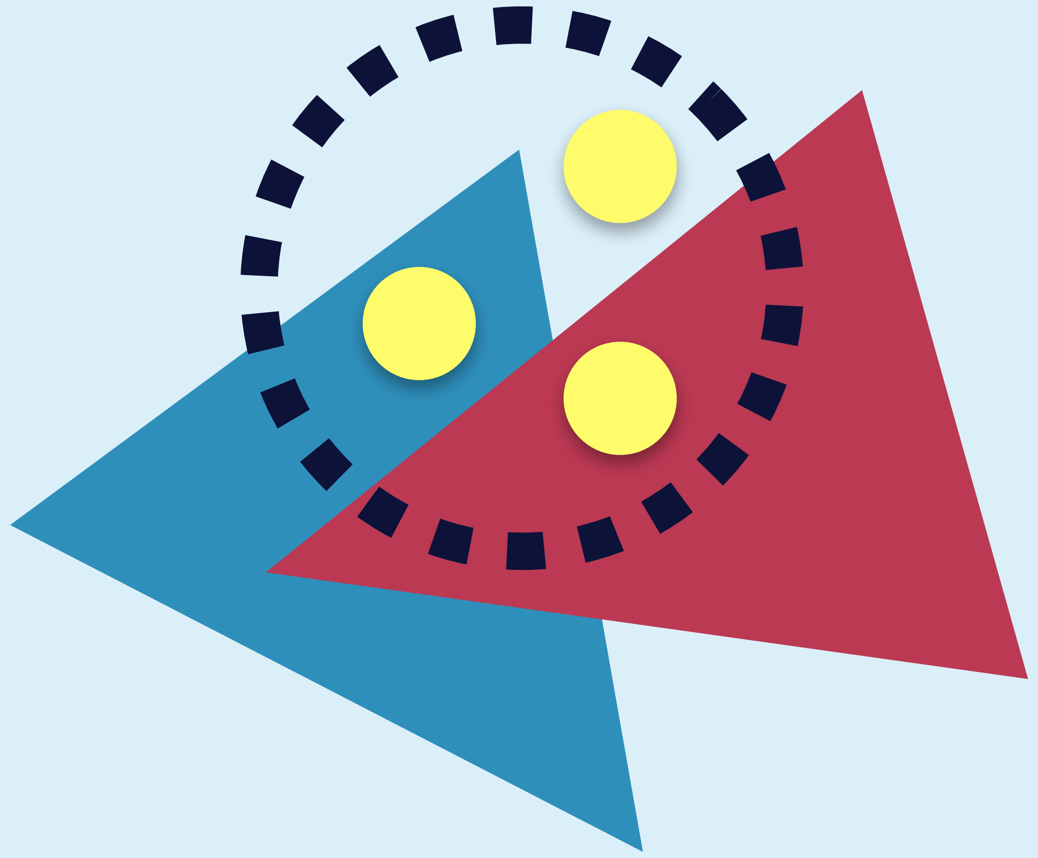
GENERALIZE TO 2D

interior derivative

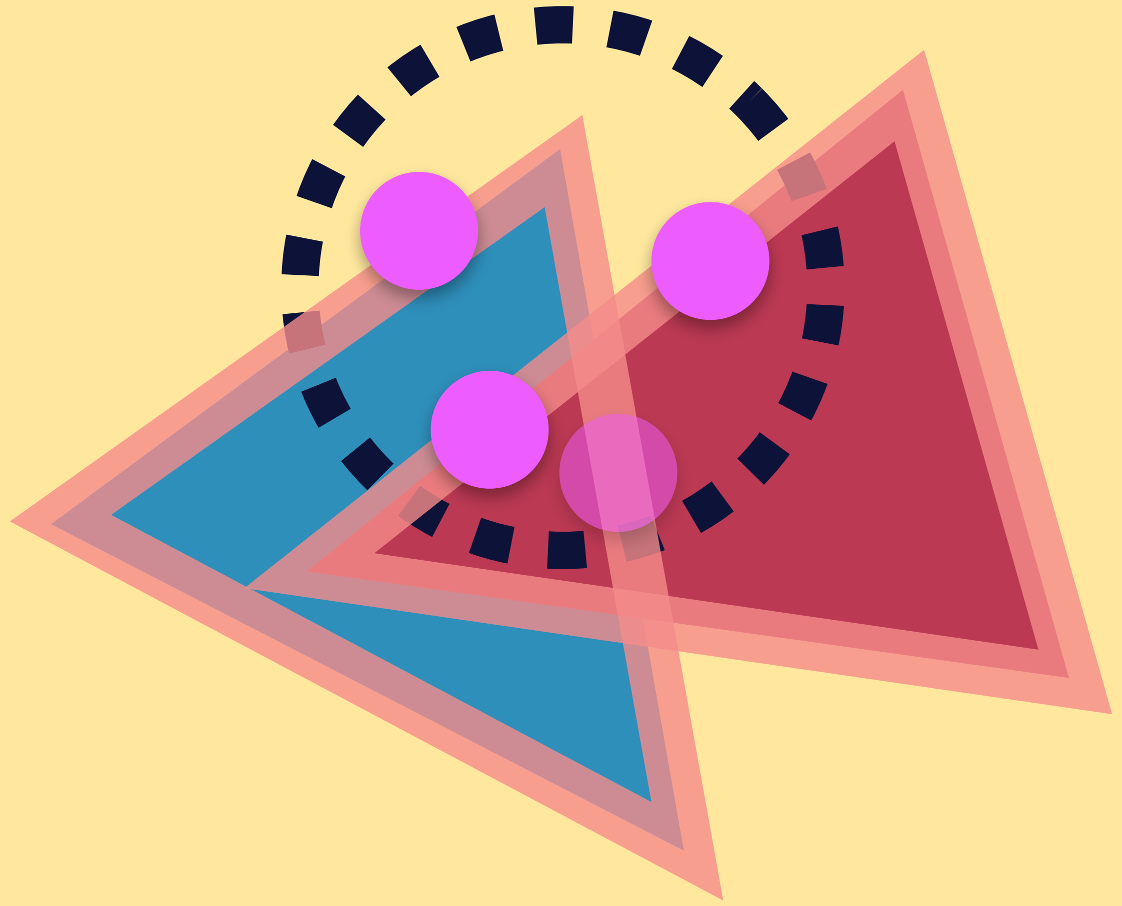
$$\frac{\partial}{\partial p} \iint$$



$$= \iint \frac{\partial}{\partial p}$$



$$+ \int$$

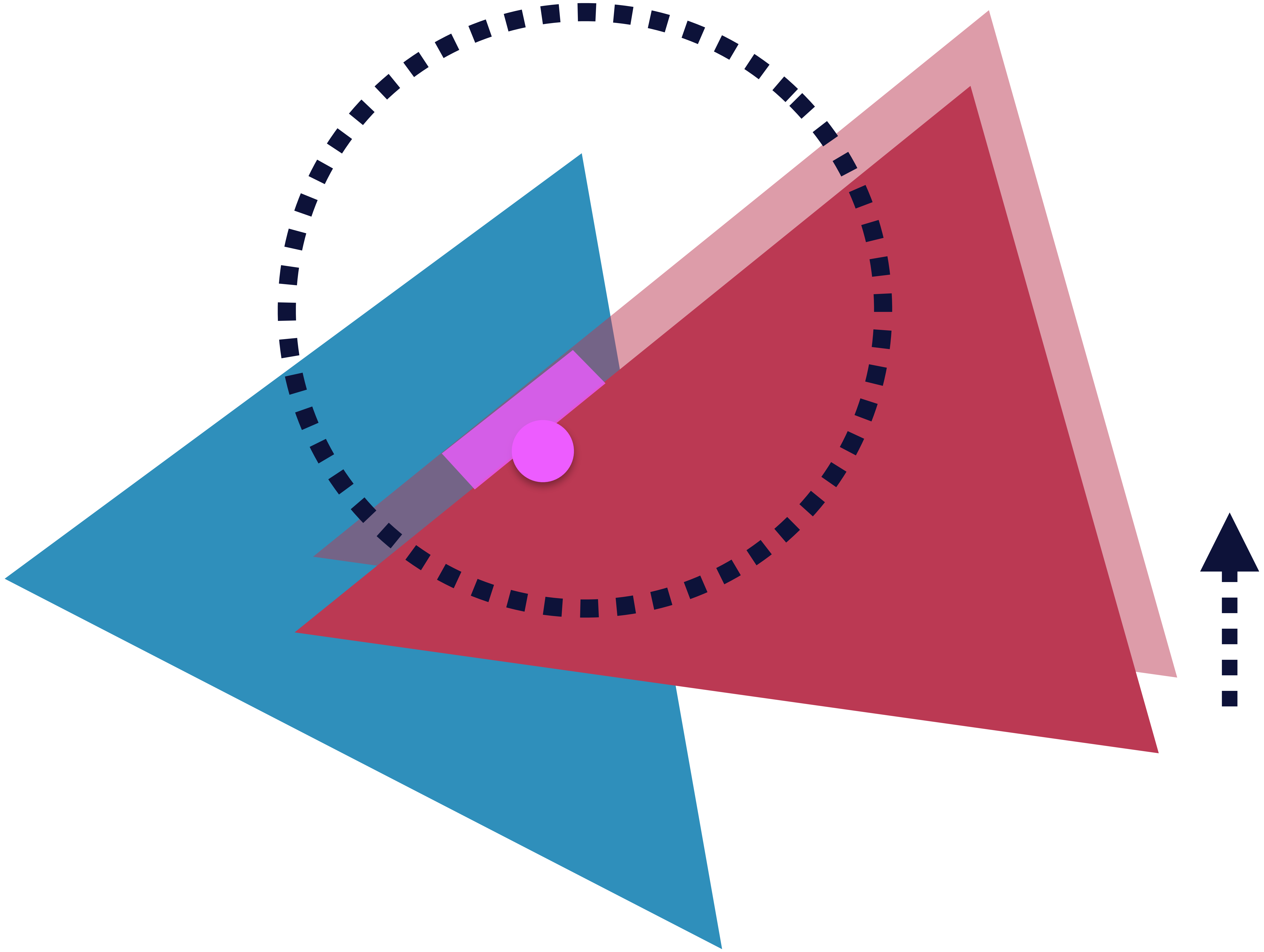


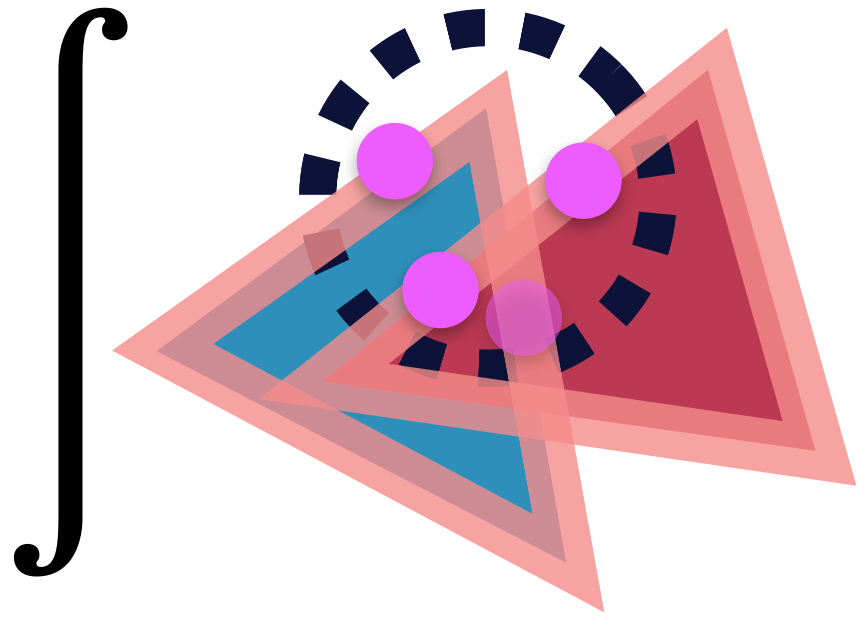
boundary derivative

Reynolds transport theorem
[Reynolds 1903]

DERIVING THE 2D BOUNDARY DERIVATIVE

- boundary derivative = infinitesimal volume change w.r.t. parameter





DERIVING THE 2D BOUNDARY DERIVATIVE

- boundary derivative = infinitesimal volume change w.r.t. parameter

3D view around the purple sample

DERIVING THE 2D BOUNDARY DERIVATIVE

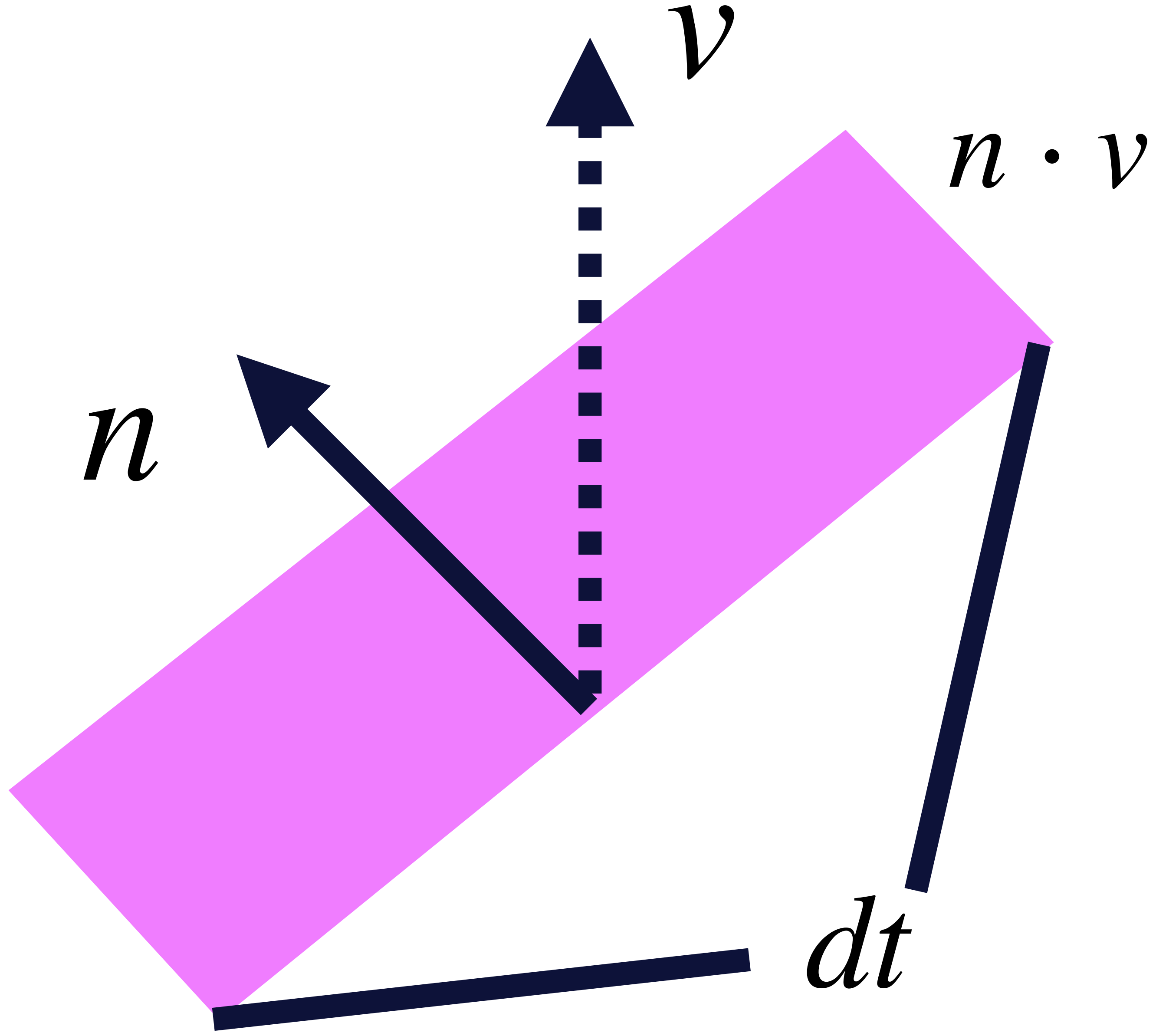
- boundary derivative = infinitesimal volume change w.r.t. parameter

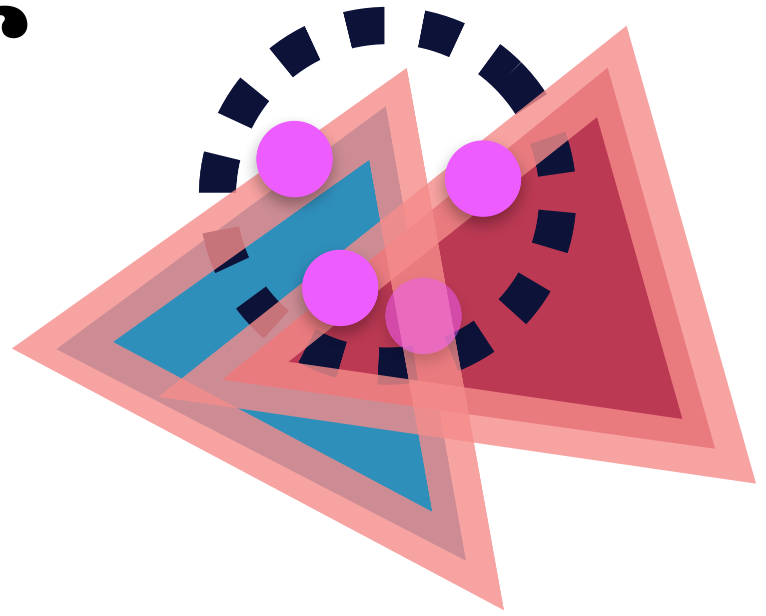
DERIVING THE 2D BOUNDARY DERIVATIVE

$v =$ boundary movement w.r.t. param

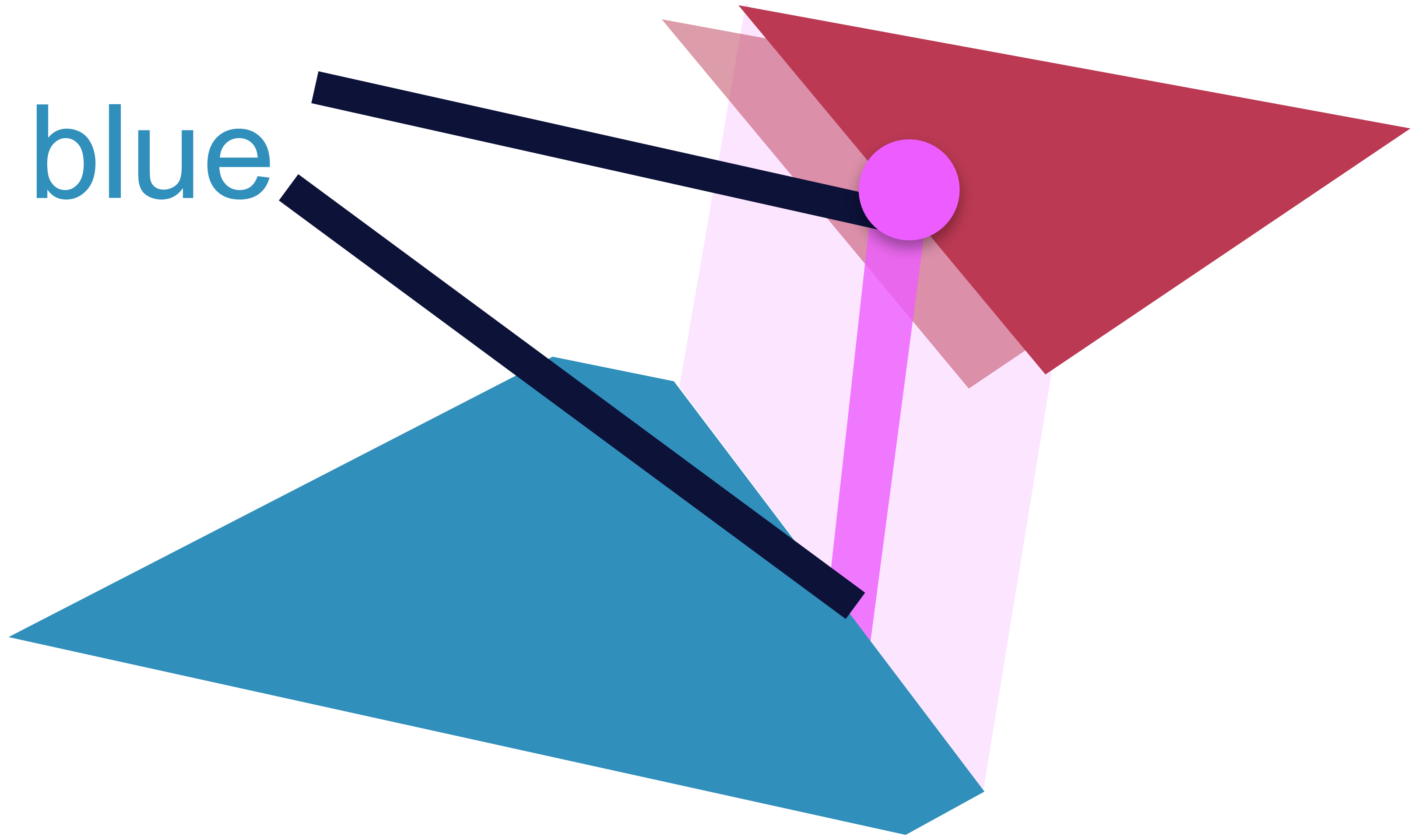
$= \dfrac{\partial x}{\partial p}$

THE INFINITESIMAL BOUNDARY VOLUME

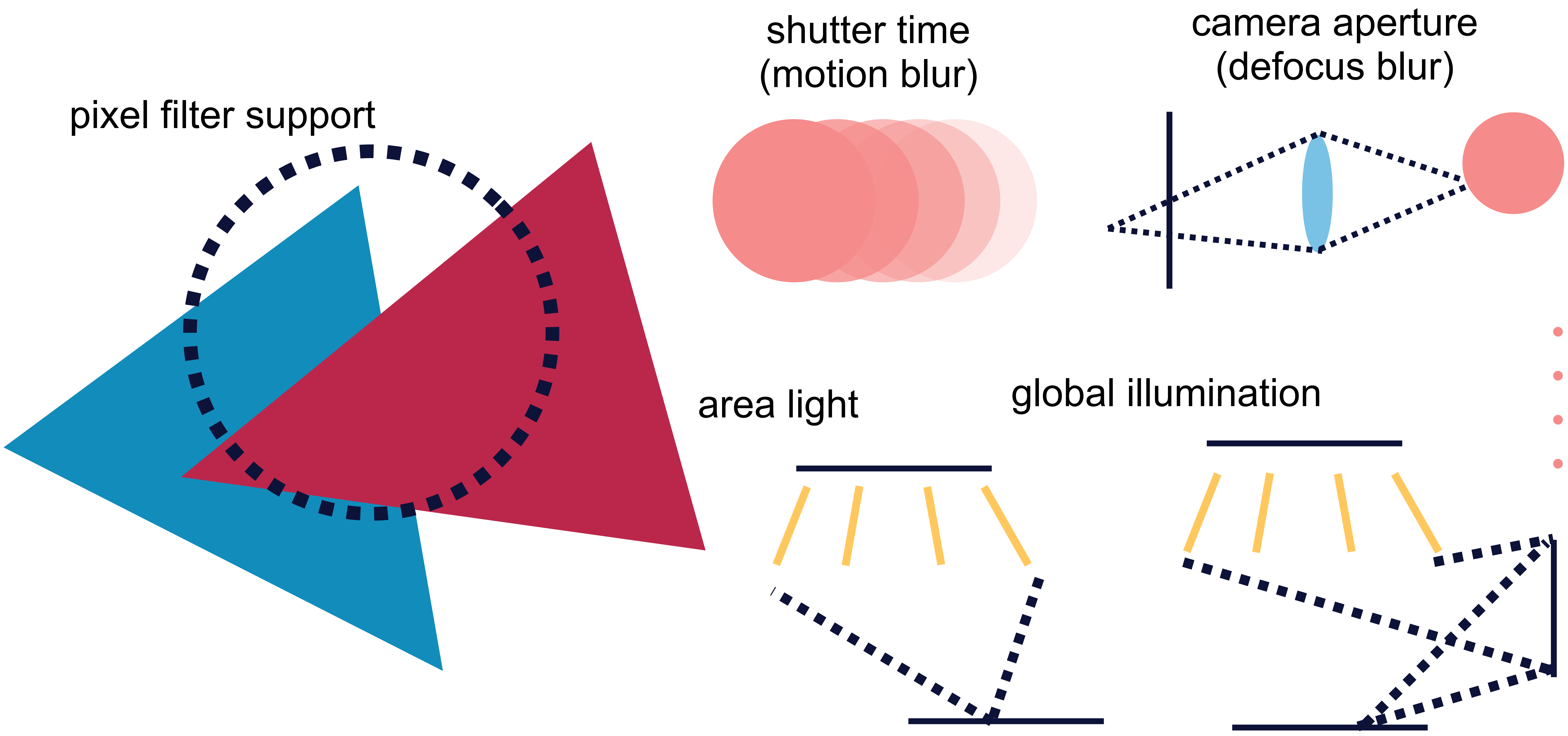


$\int$  $dt = \int \underbrace{(f_- - f_+)}_{\text{height}} \underbrace{(n \cdot v)}_{\text{width}} \underbrace{dt}_{\text{length}}$

red - blue

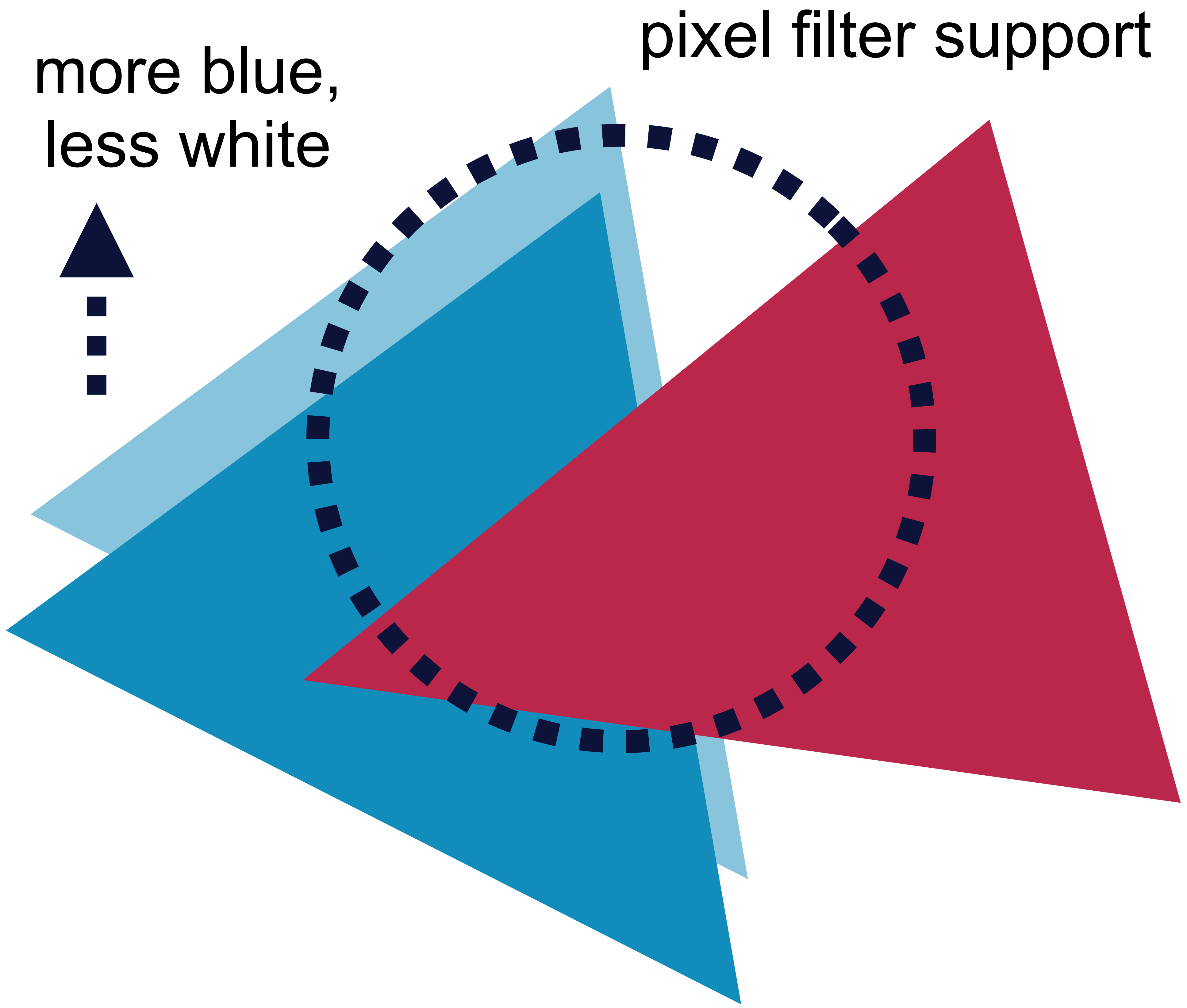


RENDERING = COMPUTING INTEGRALS

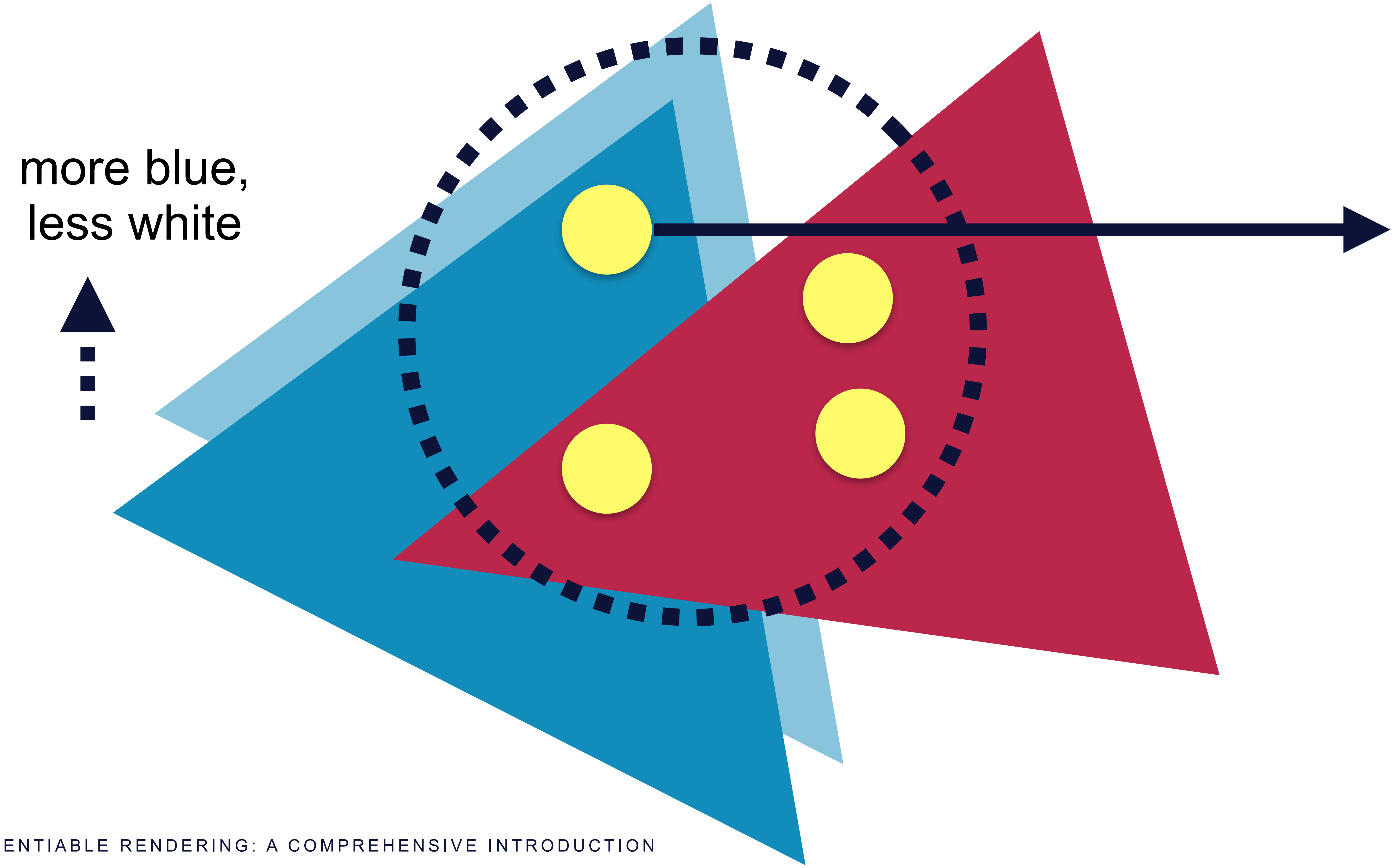


- wavelength
- transmittance
- …
- and more!

- **While the *integrand* is discontinuous, the *integral* is differentiable!**
  - the average color changes continuously as triangles move

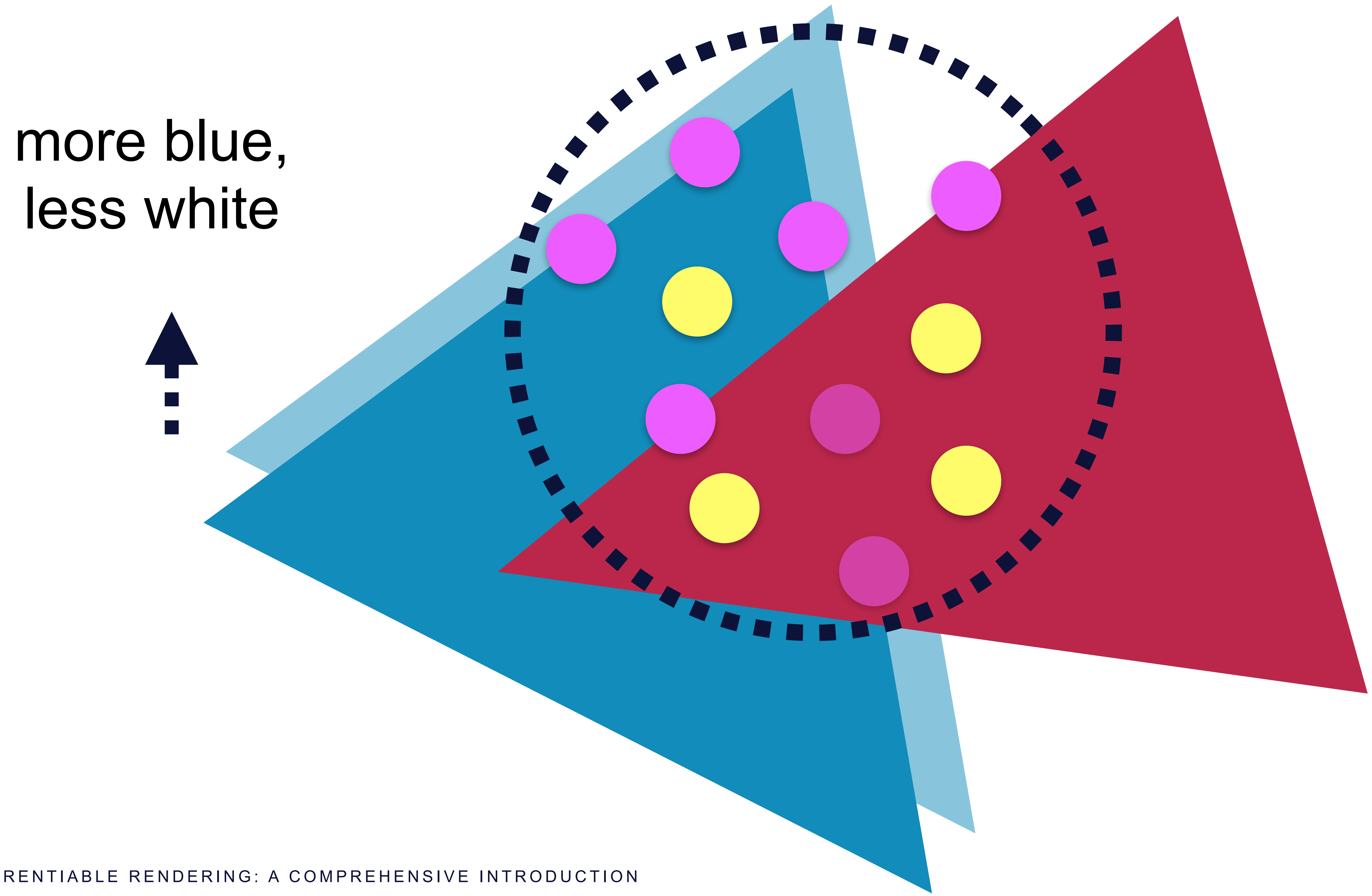


DIFFERENTIATING INTEGRAL SAMPLES GIVES WRONG DERIVATIVES



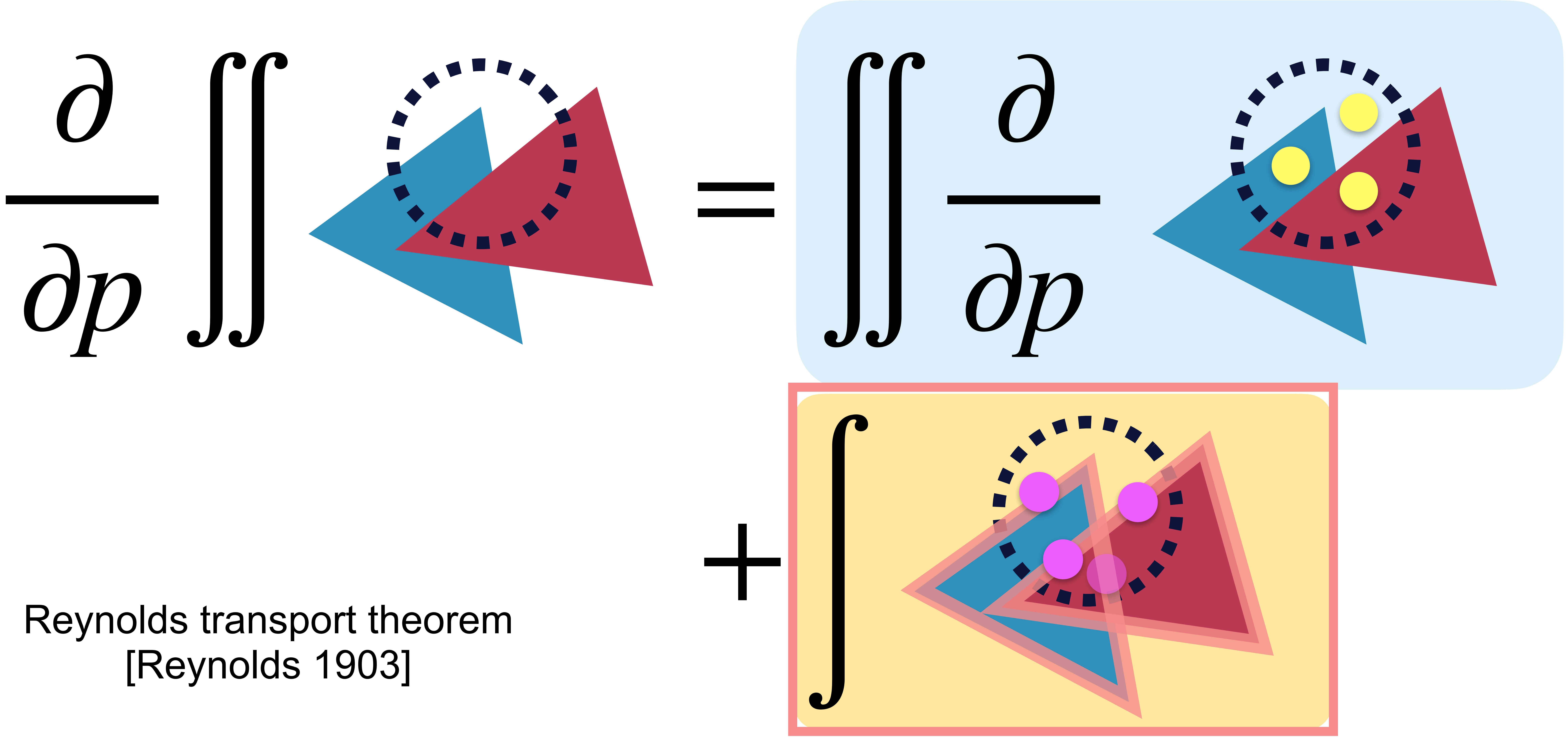
$$\frac{\partial}{\partial p} = 0$$

KEY IDEA: EXPLICITLY INTEGRATE THE BOUNDARIES



RECAP

interior derivative



boundary derivative

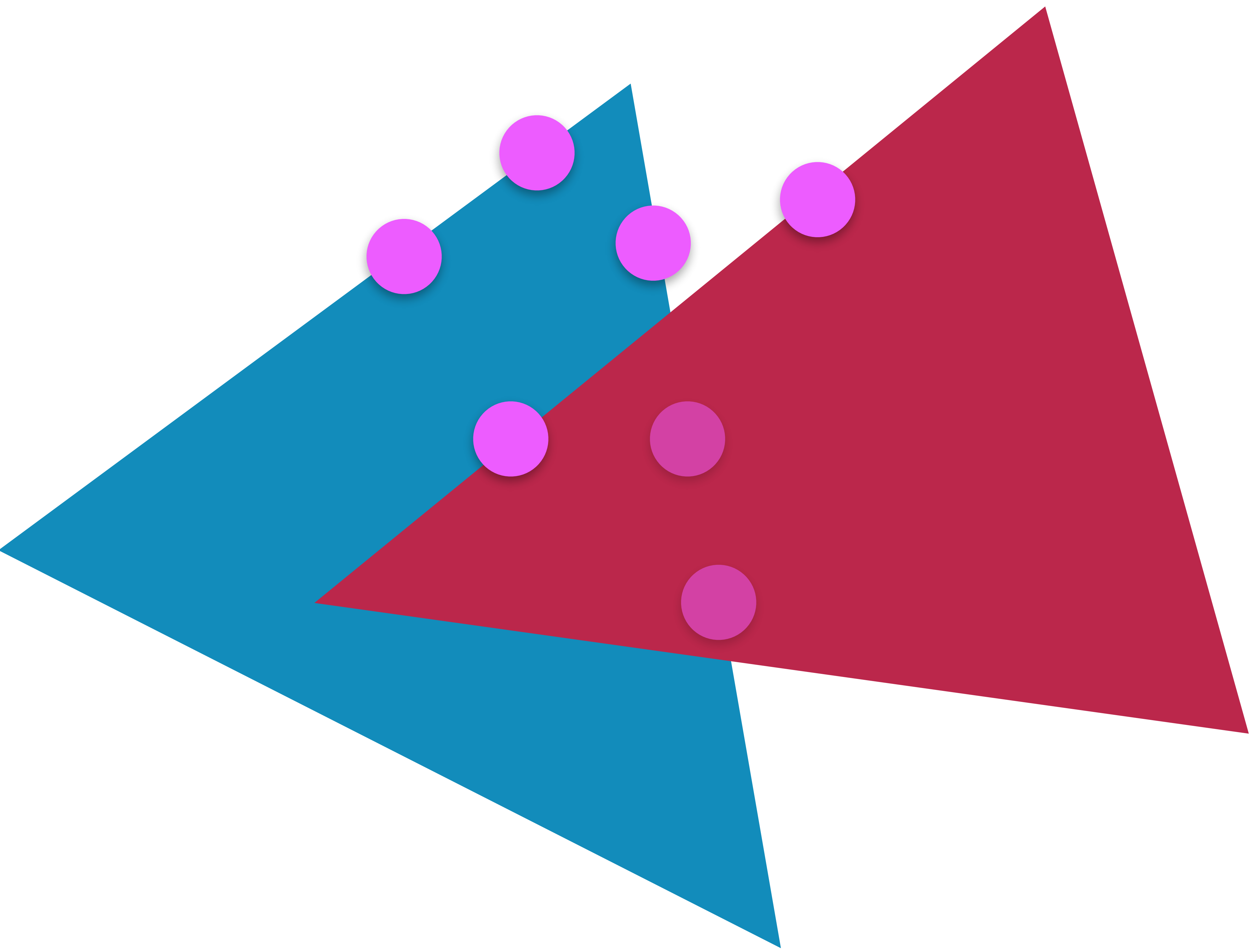
Reynolds transport theorem
[Reynolds 1903]

DISCUSSION

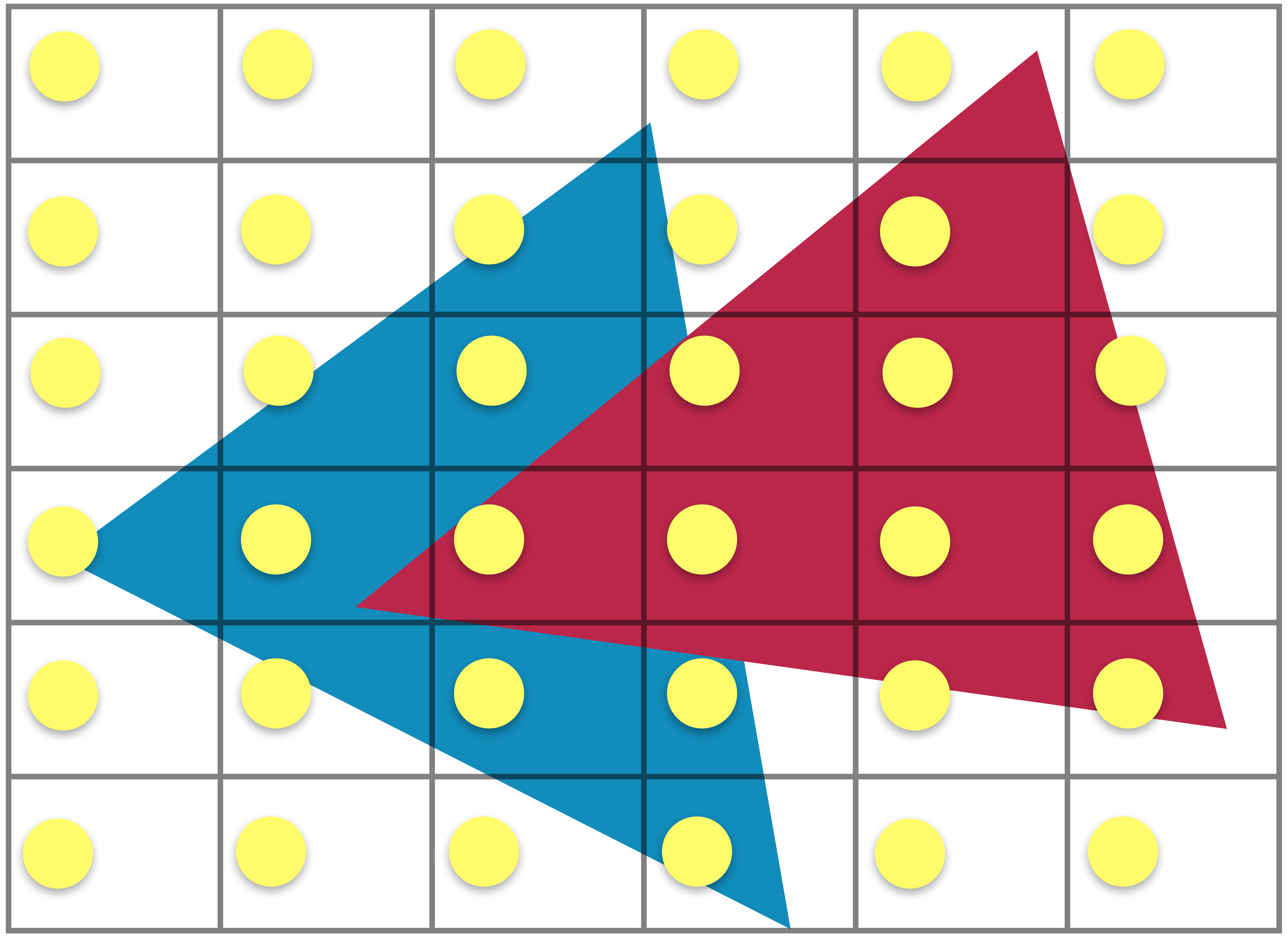
- Ray tracing vs rasterization
- Approximated solutions
- Geometry representation
- Limitations

RAY TRACING VS RASTERIZATION

- The boundary sampling is not very compatible with z-buffer rendering

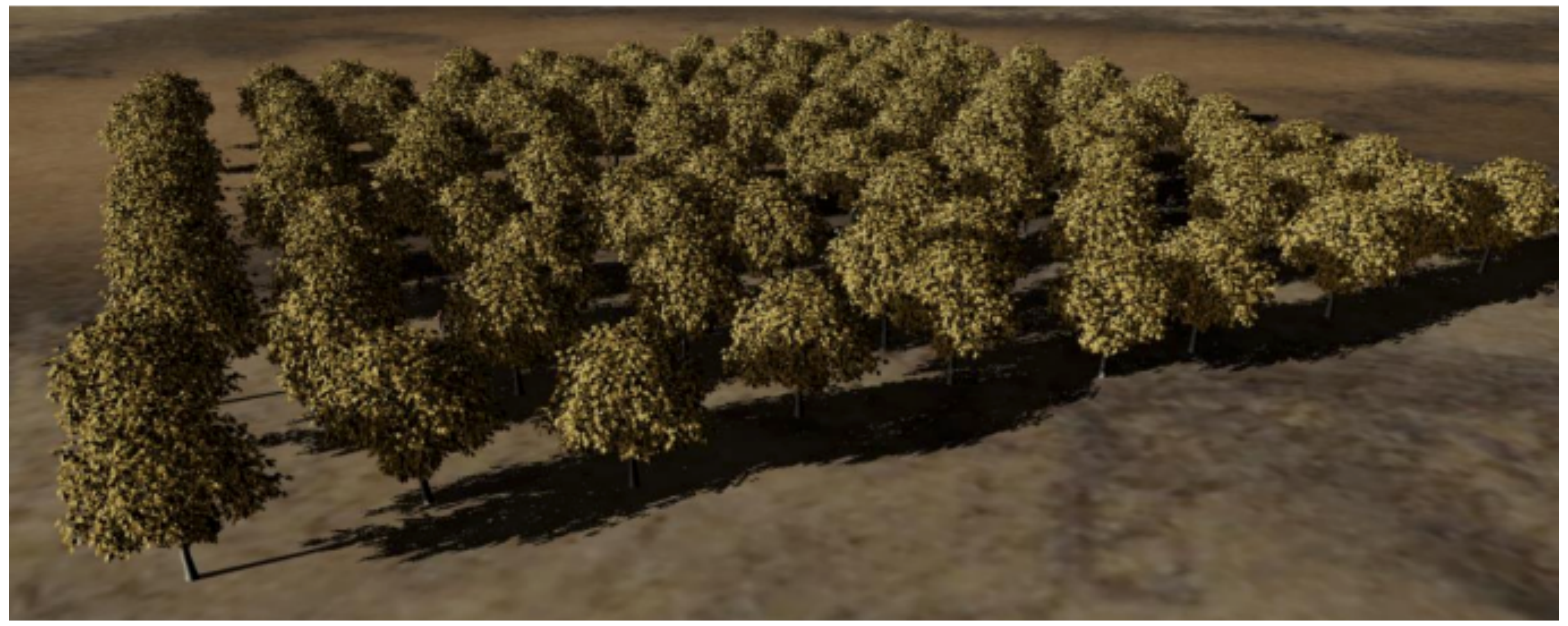


v.s.



RAY TRACING VS RASTERIZATION

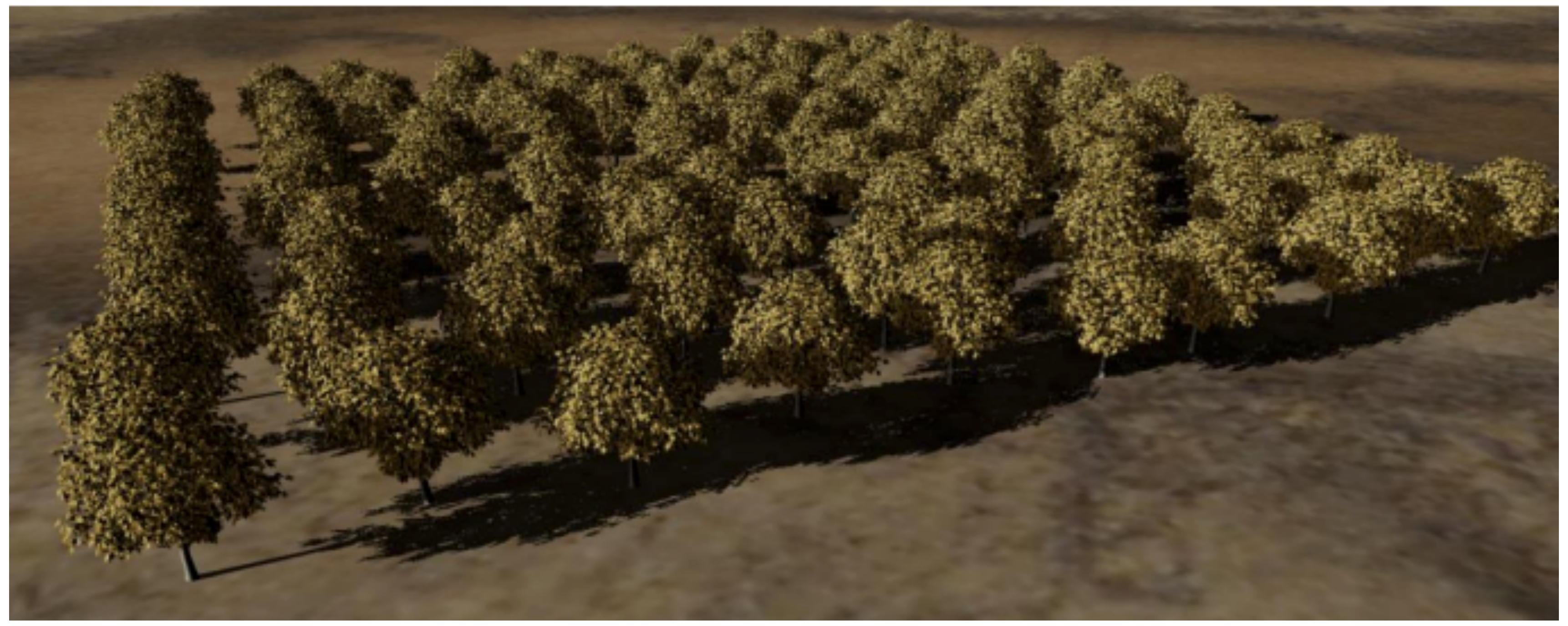
- Ray tracing is not significantly slower than rasterization
- The interior derivatives can be computed using rasterization



from Gruen 2020
1080p, ~19M triangles
raster: 2.7 ms
raytrace: 8.6 ms (2.5 ms for animation)

RAY TRACING VS RASTERIZATION

- Ray tracing is not significantly slower than rasterization
- The interior derivatives can be computed using rasterization
- Visibility queries may not be the main bottleneck

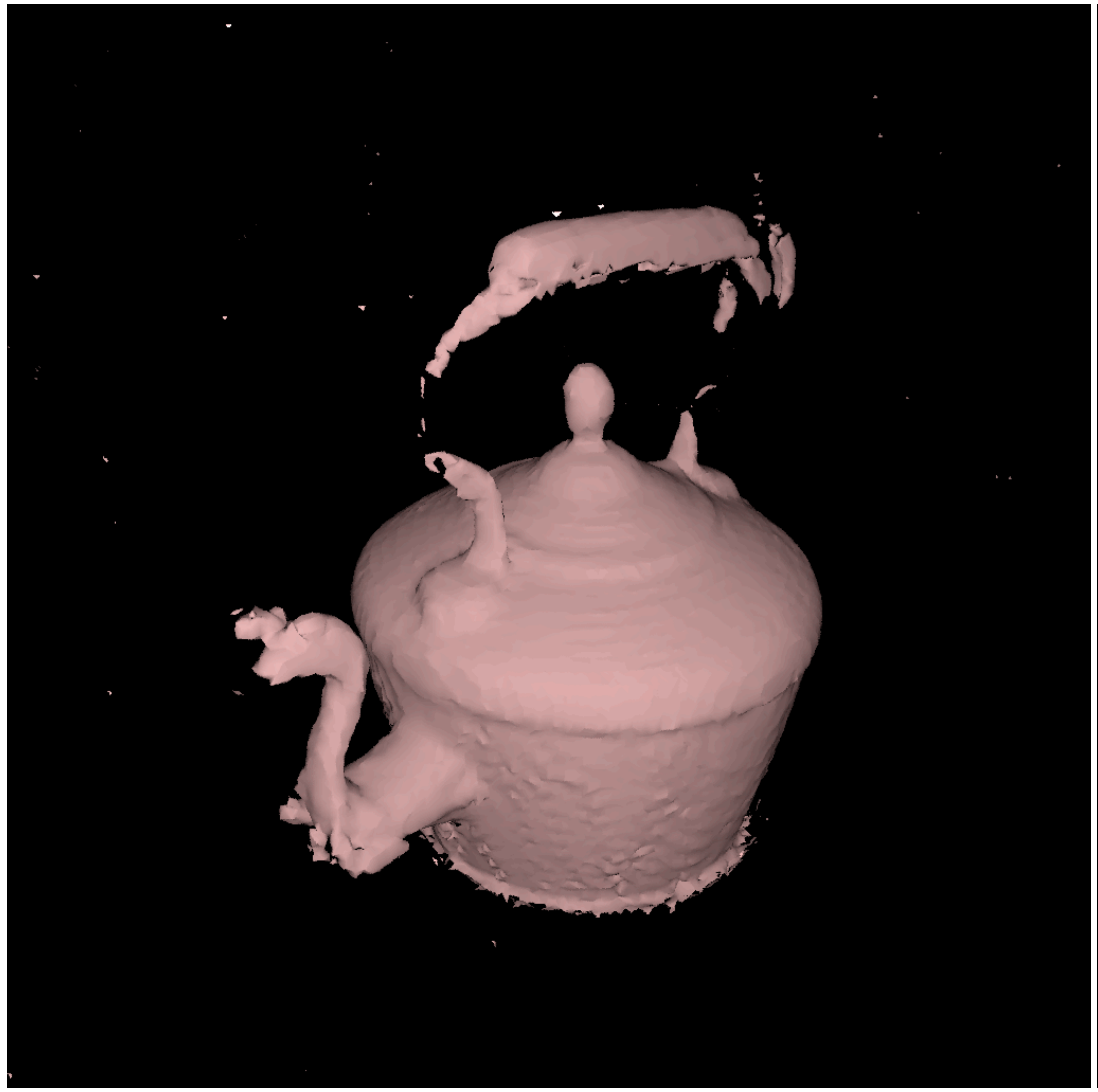


from Gruen 2020
1080p, ~19M triangles
raster: 2.7 ms
raytrace: 8.6 ms (2.5 ms for animation)

~10k faces, 256x256 (Titan Xp)
PyTorch3D (raster) 220ms
redner (raytrace) 60ms
(BVH 20ms, forward 7ms, backward 27ms)

RAY TRACING VS RASTERIZATION

23823 vertices, 44702 faces



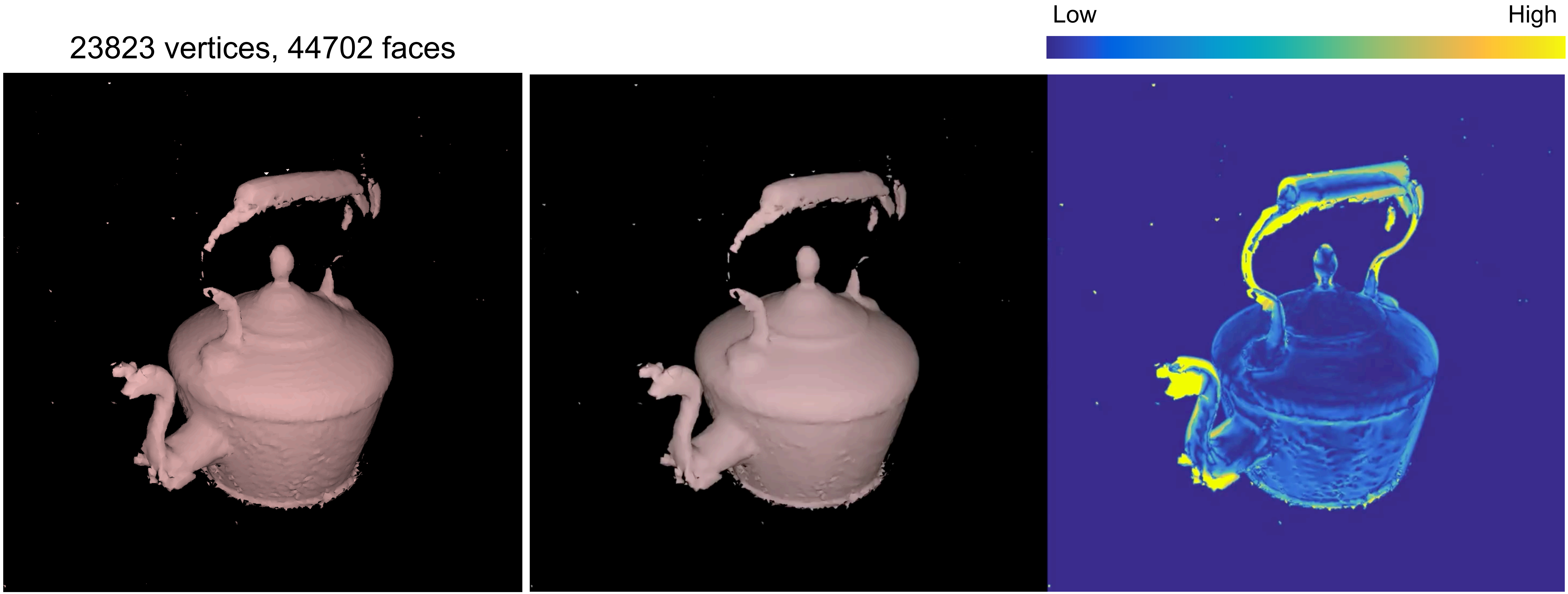
initial

target

- 1024x1024 at 2 spp (Titan Xp) forward + backward
  - Ray tracing + edge sampling: 0.05—0.1 sec
  - PyTorch3D: 0.15 sec

RAY TRACING VS RASTERIZATION

23823 vertices, 44702 faces



initial

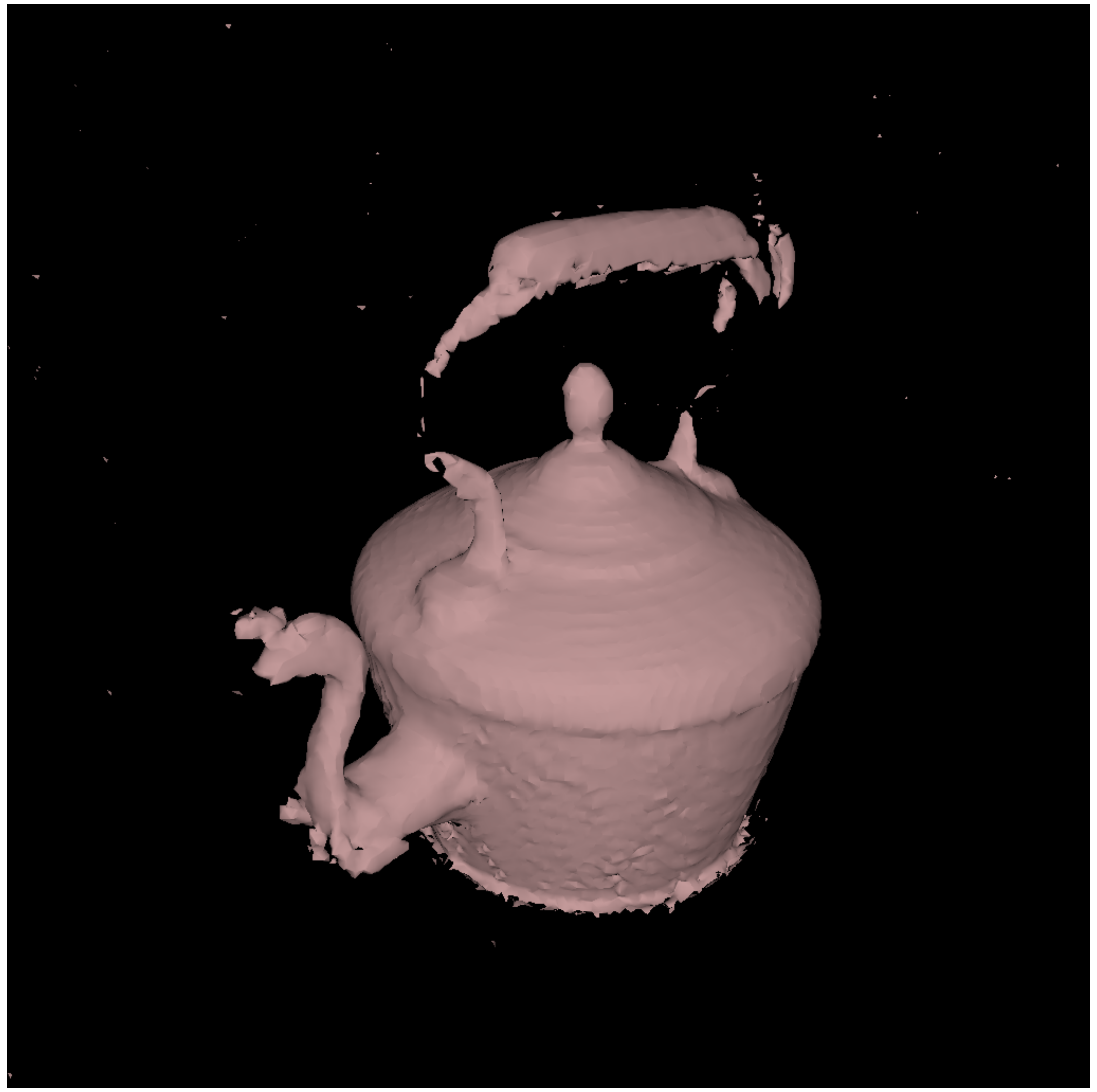
edge sampling optimization video (1 view over 20)

abs. error

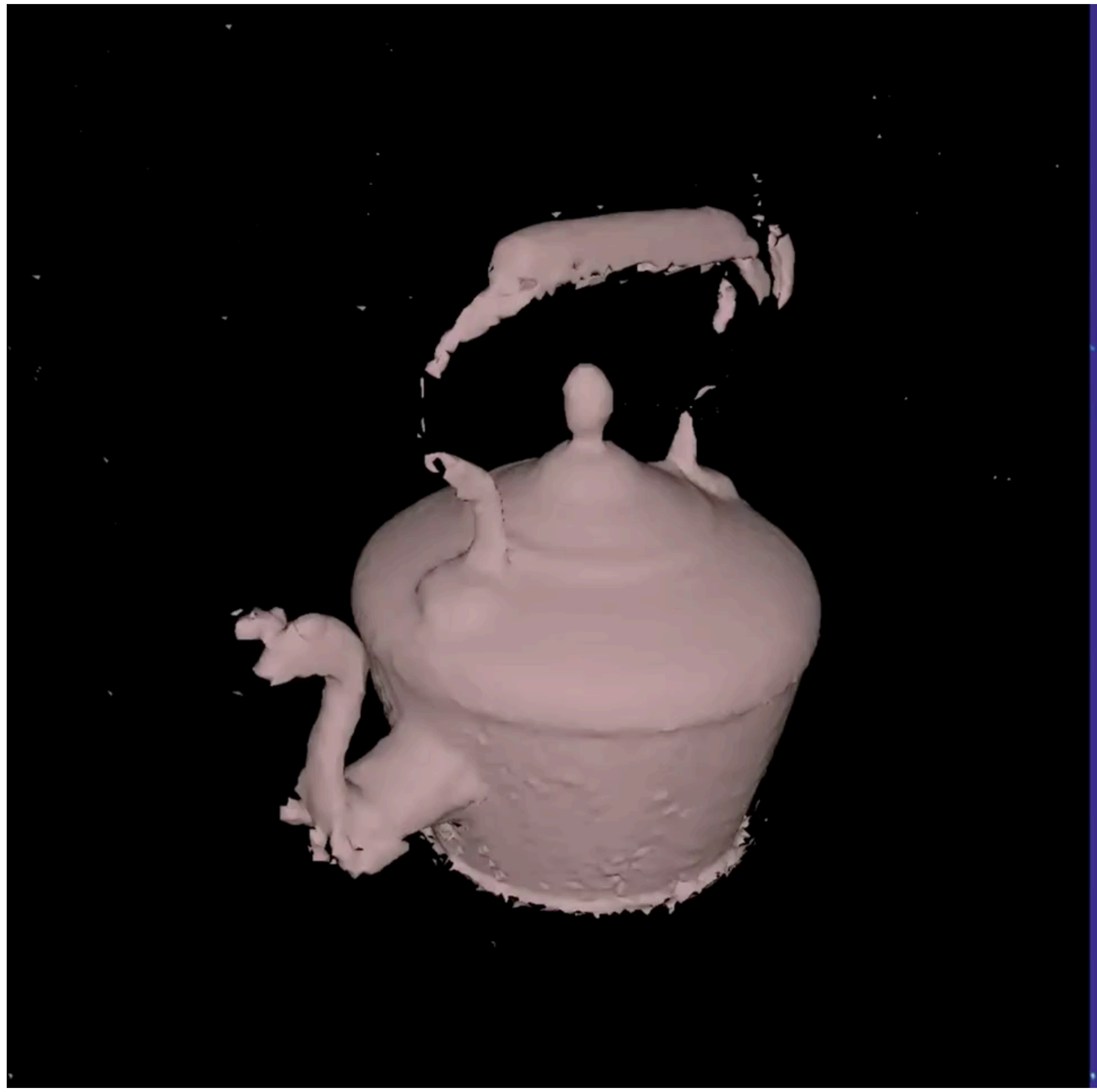
RAY TRACING VS RASTERIZATION

Low High

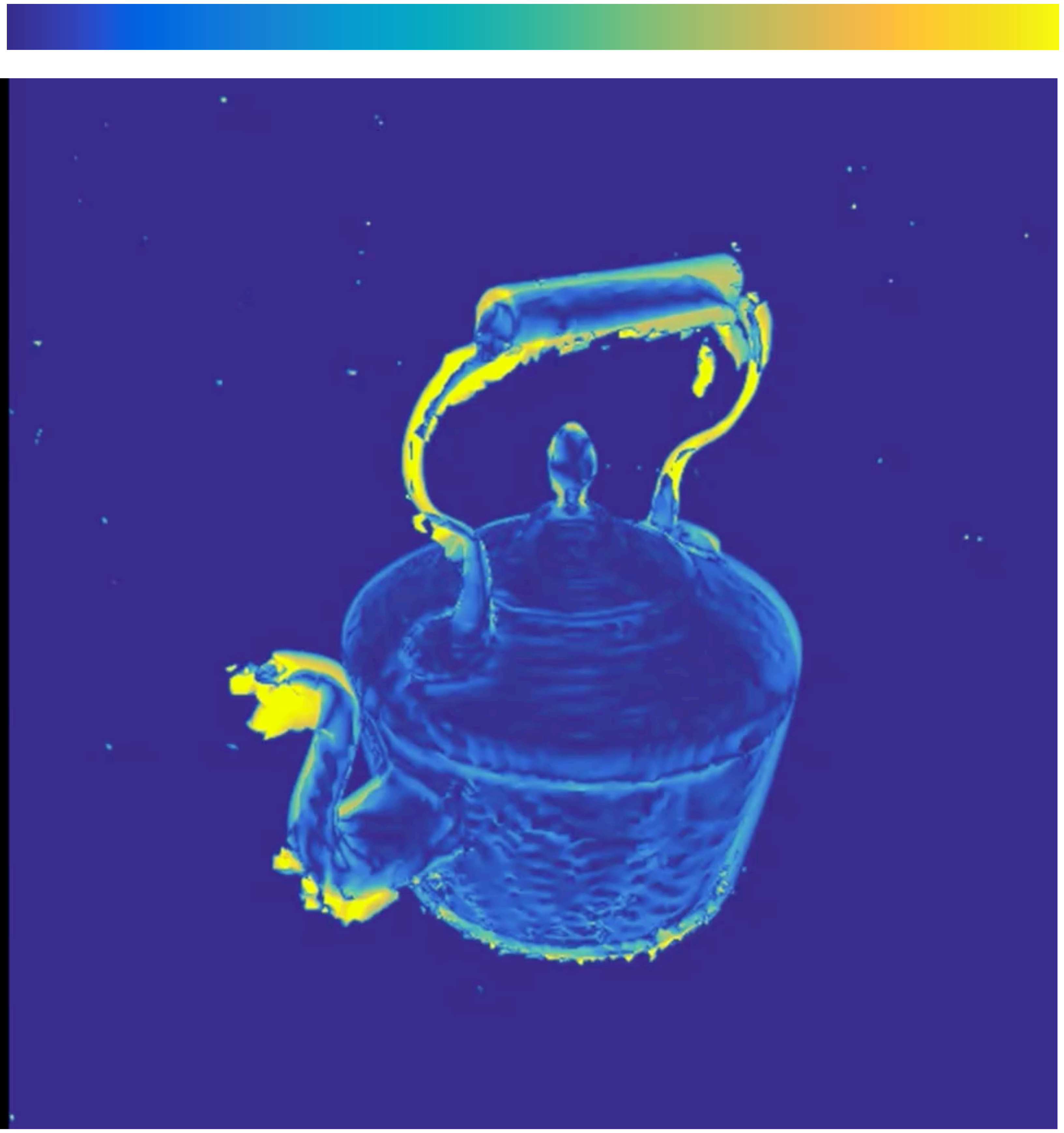
23823 vertices, 44702 faces



initial



PyTorch3D
optimization video
(1 view over 20)



abs. error

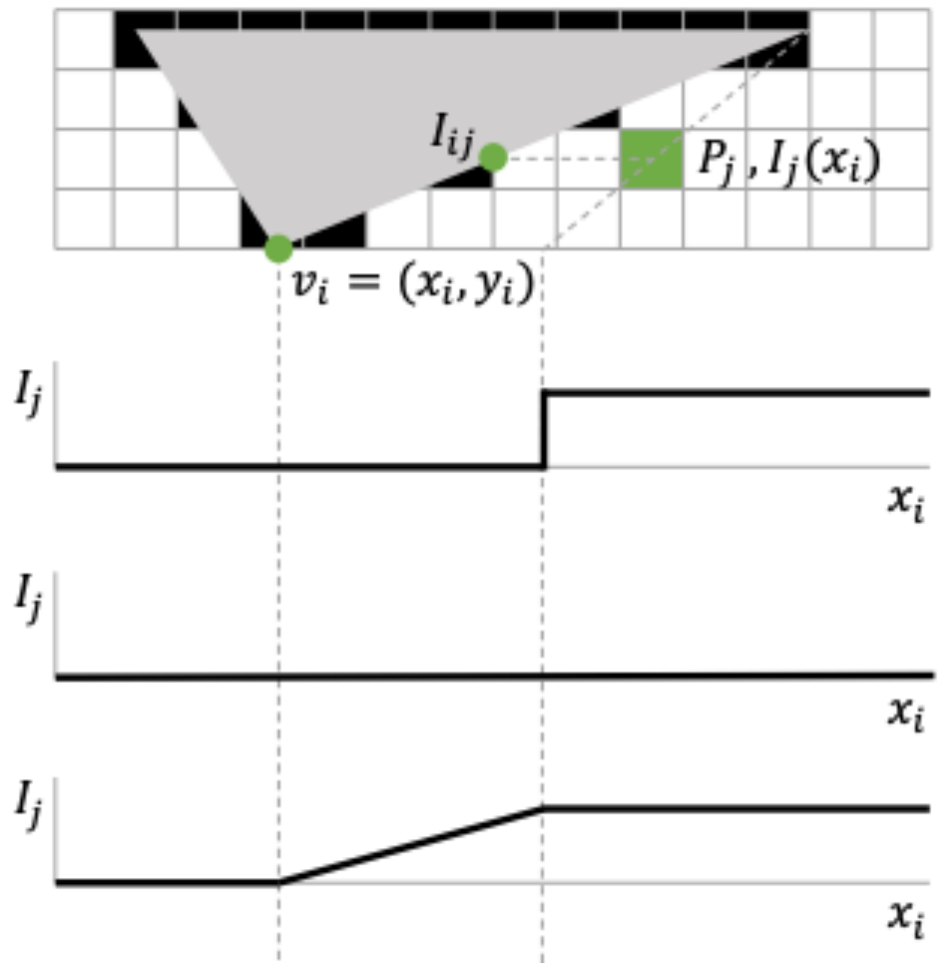
RAY TRACING VS RASTERIZATION

Optimization results after 5000 iterations (with identical settings)

optimized (ray tracing) | target | optimized (PyTorch3D)

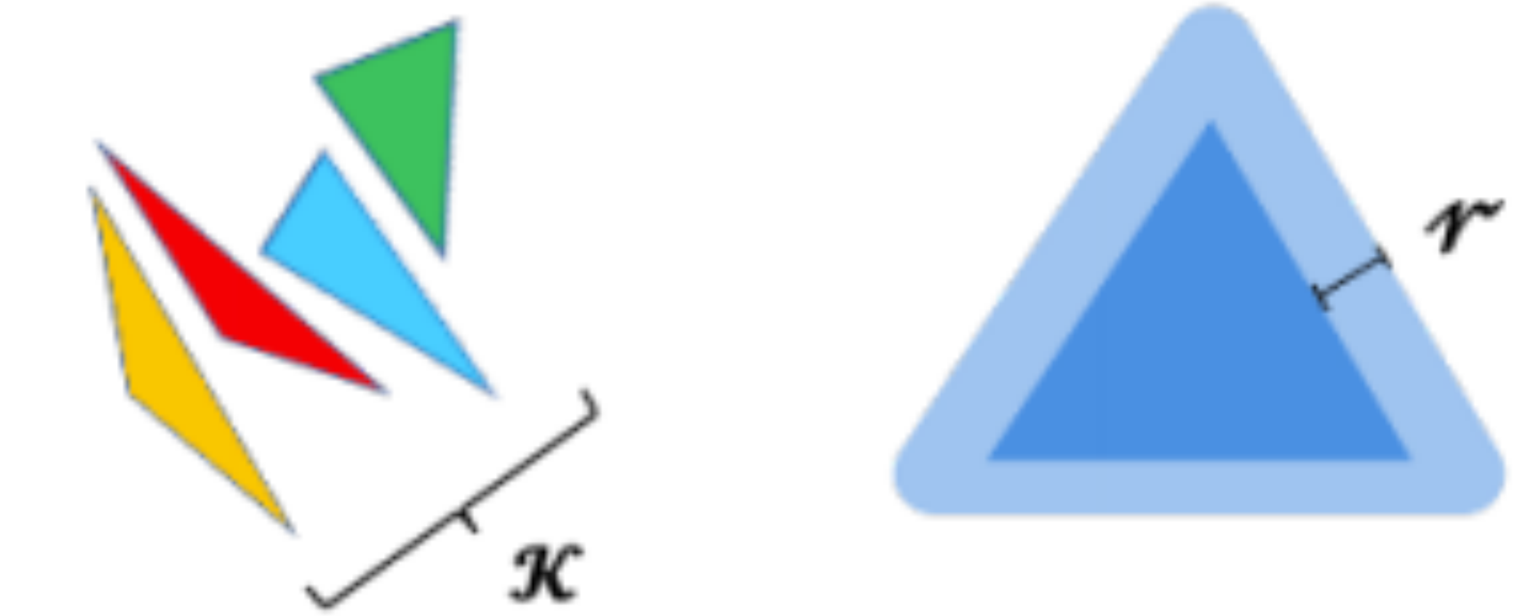
APPROXIMATED SOLUTION

- Our boundary integral is *correct, i.e.,* when the number of samples grows it converges to the integral.
- Two other kinds of approximation:
  - Keep the rendering model, approximate the derivatives (de La Gorce 2011, OpenDR 2014, Kato 2018, …)
  - Change the rendering model
    (Rhodin 2015, SoftRas 2019, PyTorch3D 2020…)



Kato 2018

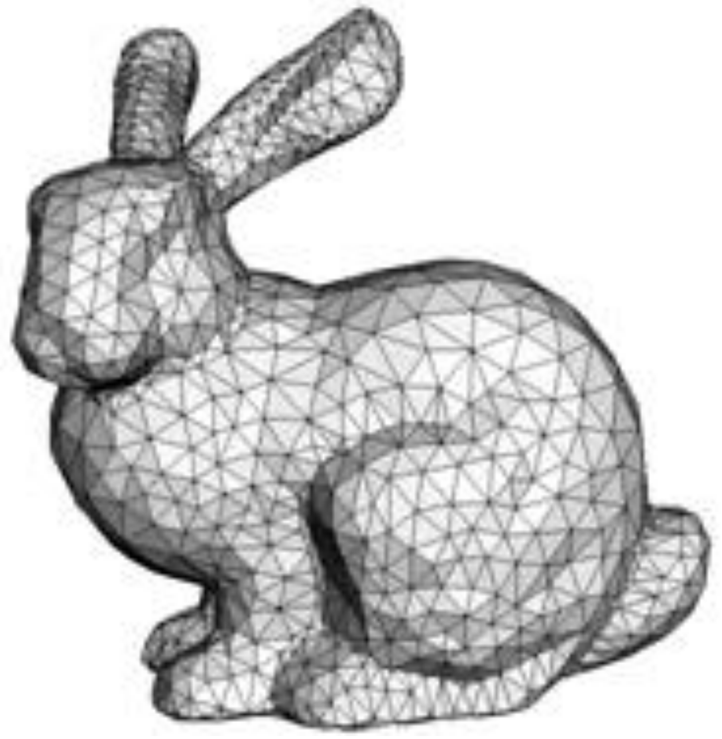

Rhodin 2015



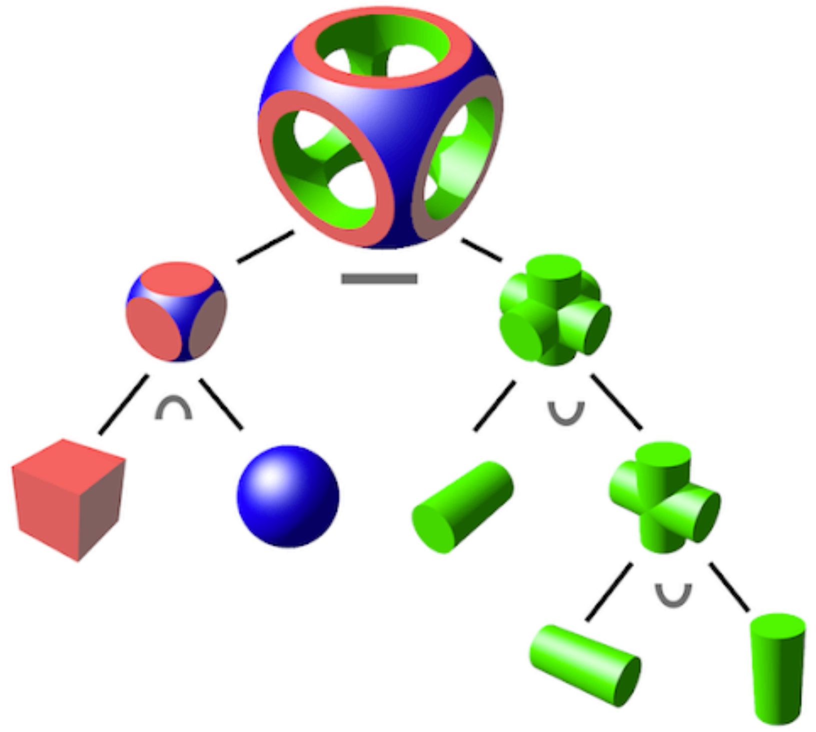
PyTorch3D [Ravi 2020]

GEOMETRY REPRESENTATION

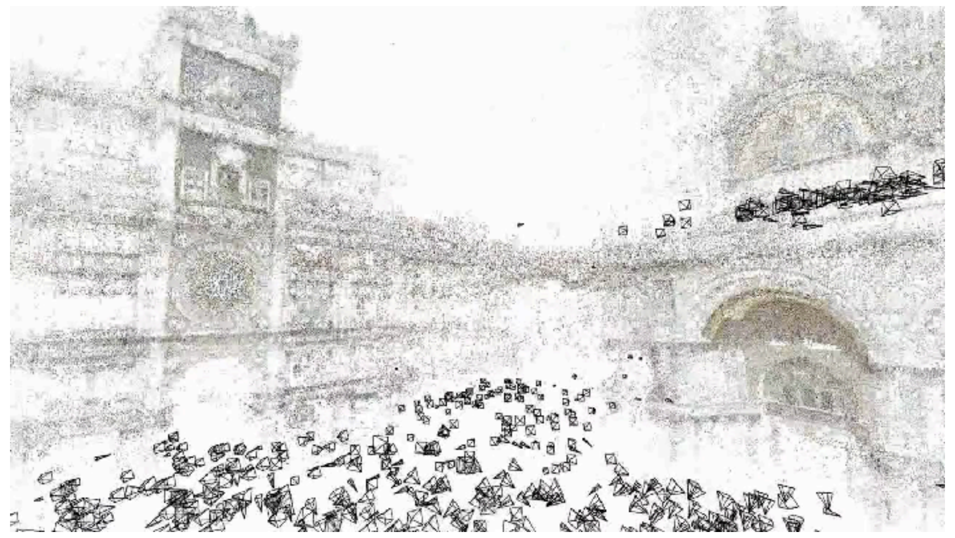
- Need boundary extraction — easier for meshes, harder for implicit representations and fractals



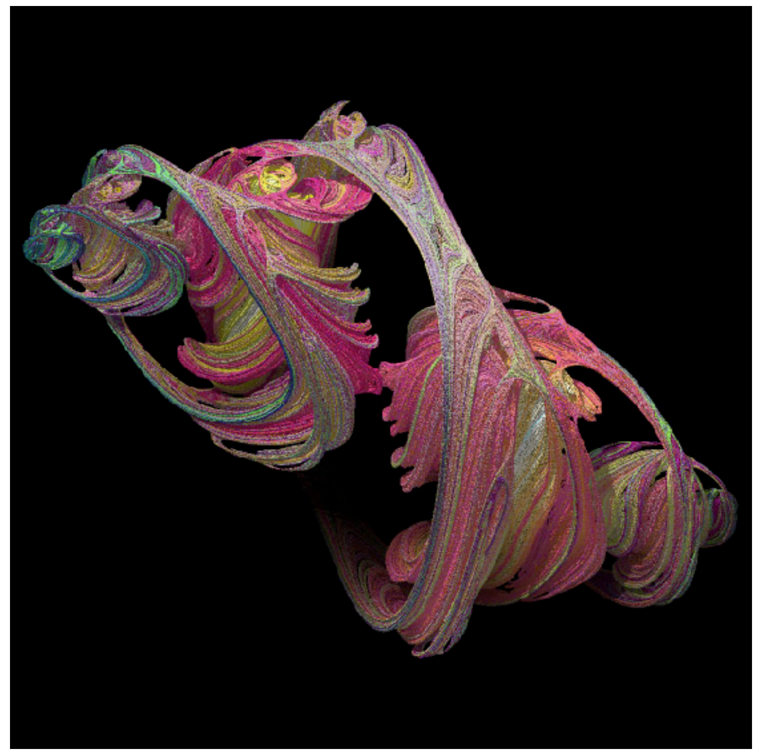
mesh



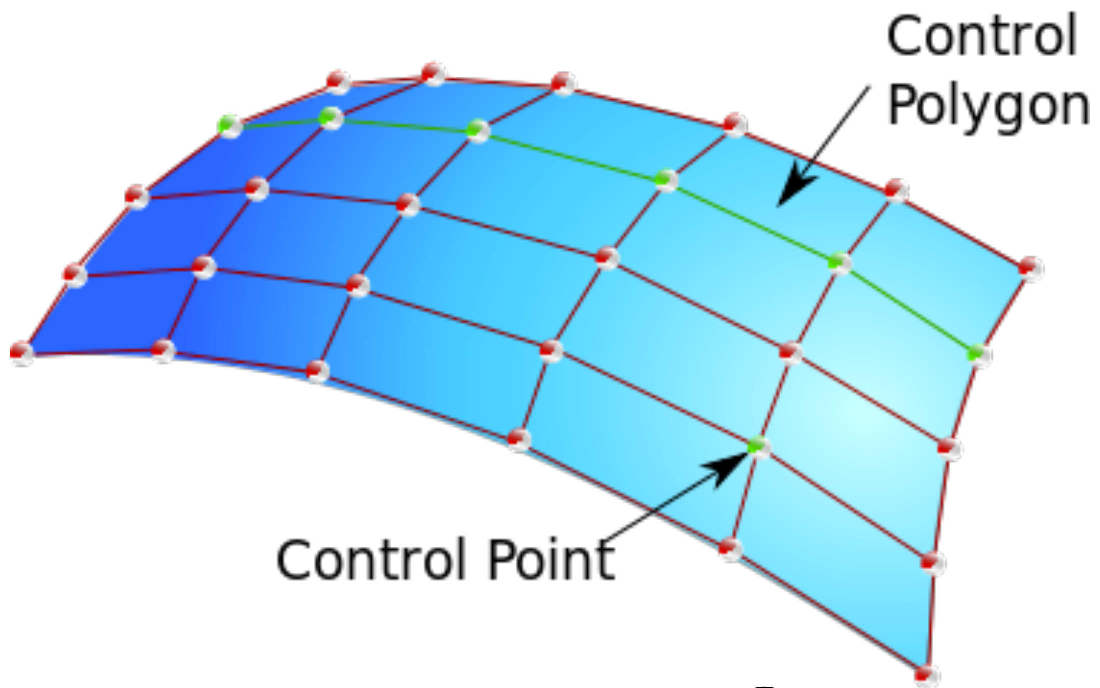
CSG



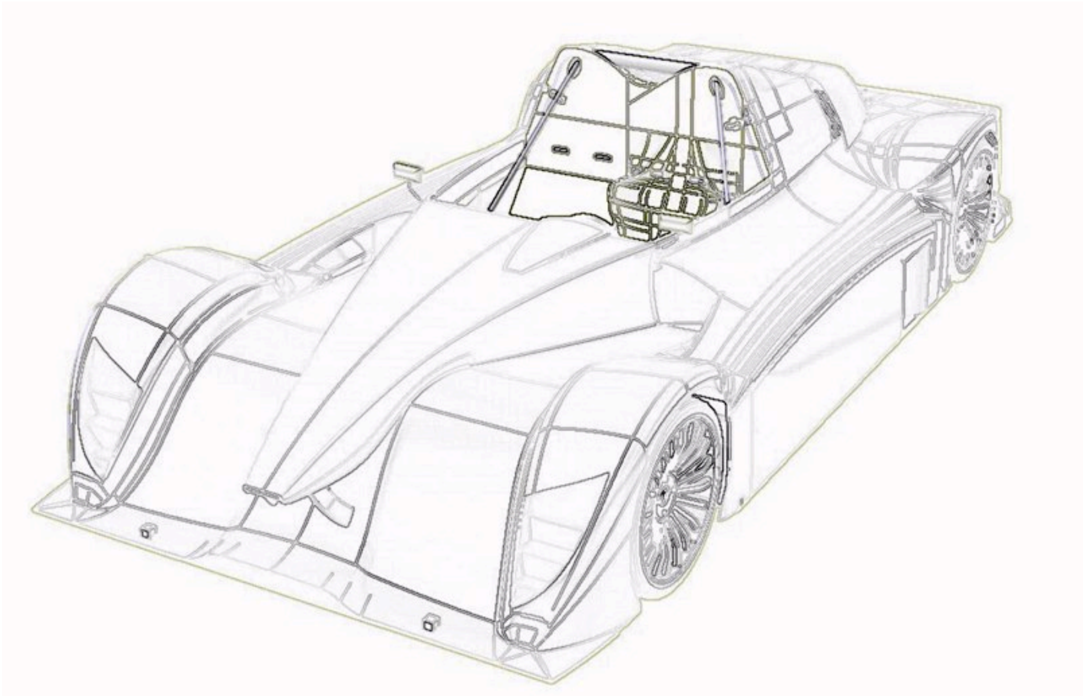
point cloud



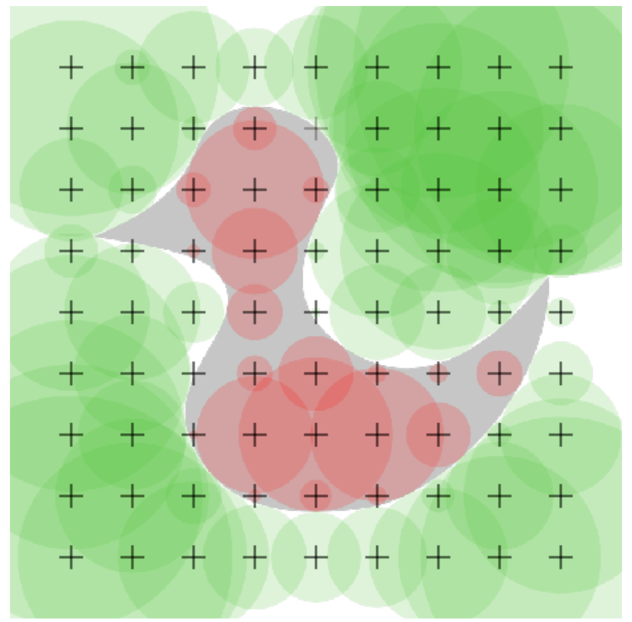
fractal



NURBS



B-rep

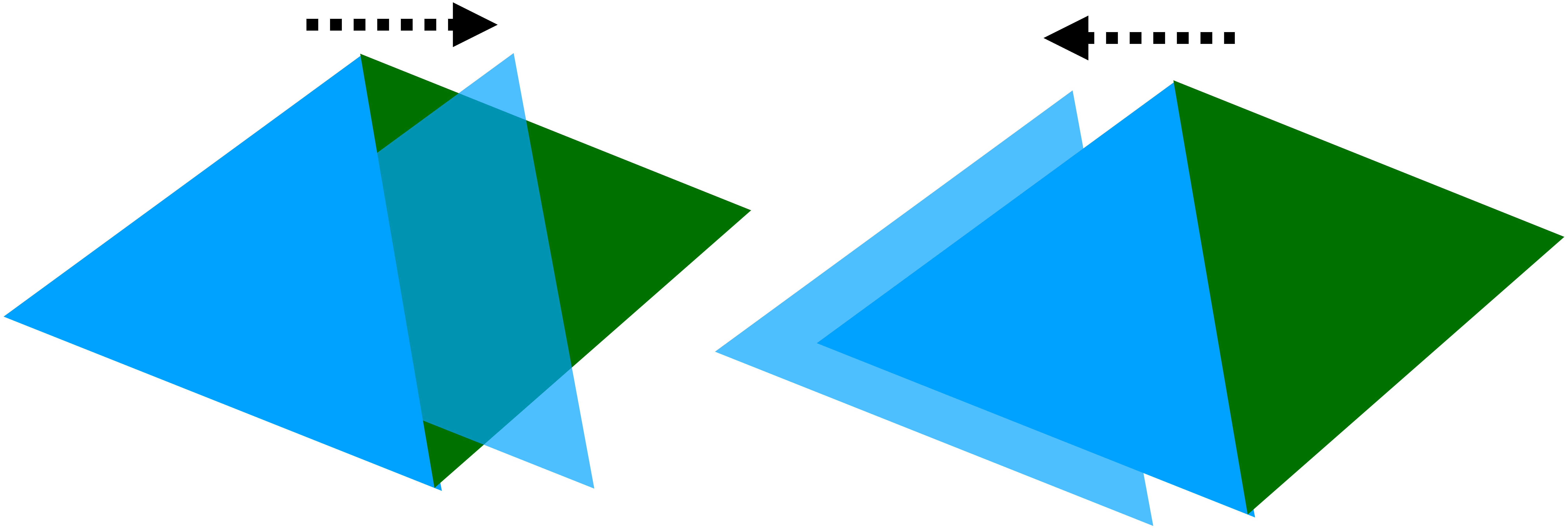


SDF

images courtesy of Carlson et al., Vladsinger, Agarwal et al., Pso, Solkoll, Zottie, Drummyfish

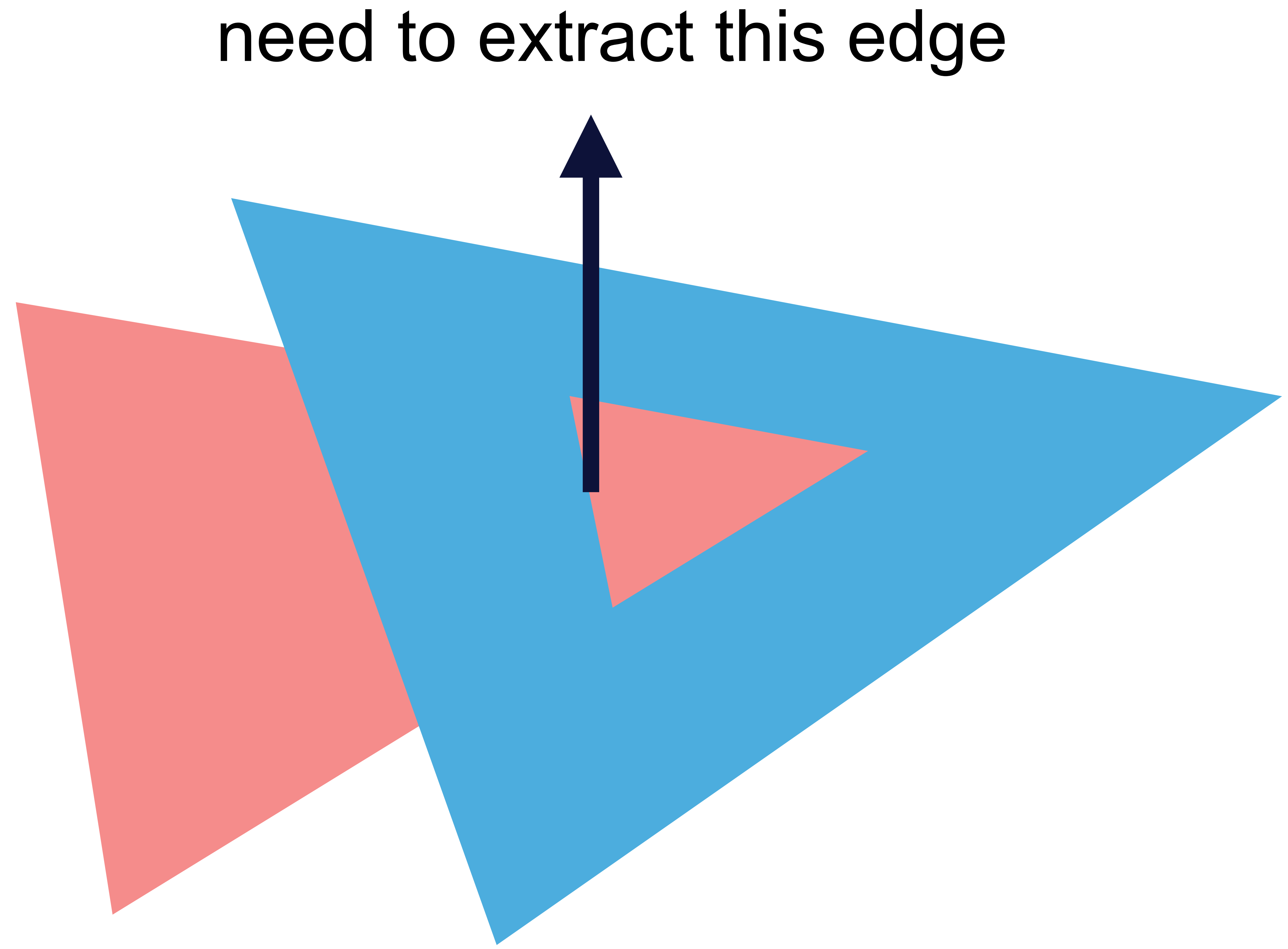
LIMITATIONS

- Non-differentiability of parallel edges of two separate triangles
  - can be resolved by applying a small perturbation to the vertices



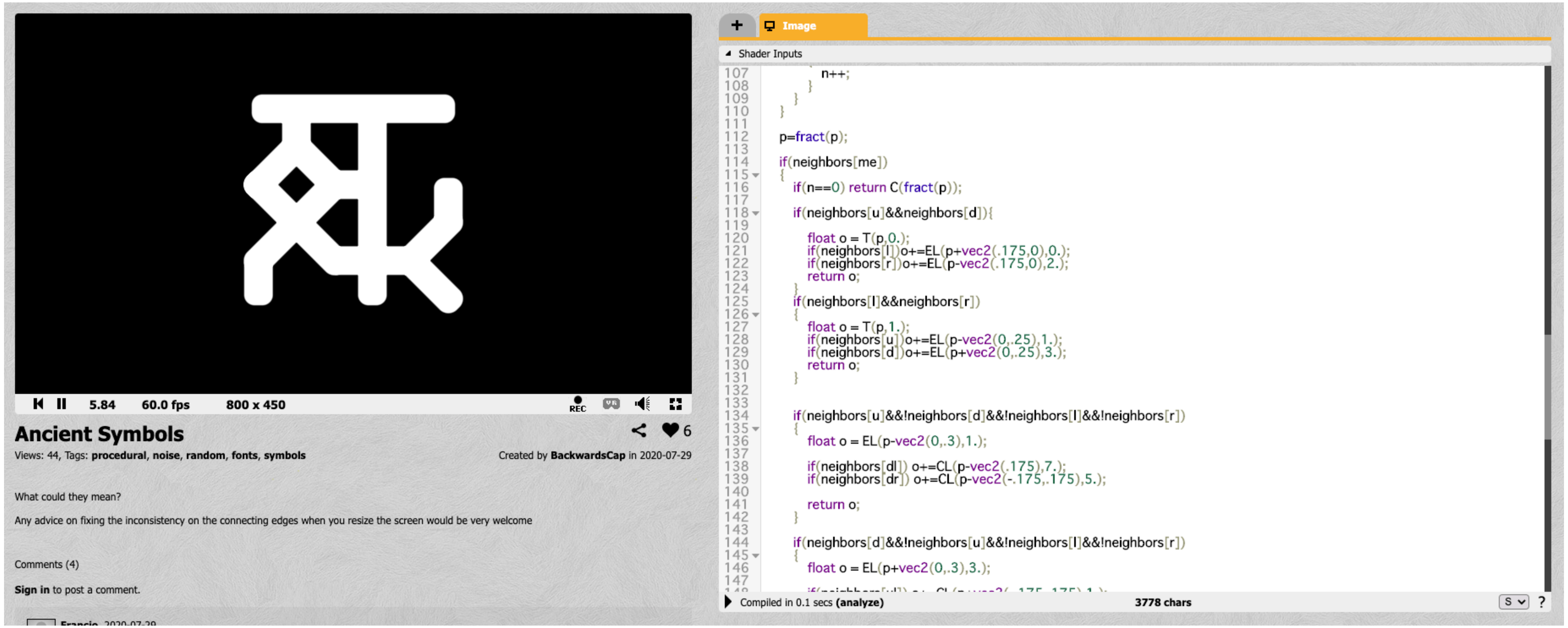
LIMITATIONS

- Non-differentiability of parallel edges of two separate triangles
  - can be resolved by applying a small perturbation to the vertices
- Interpenetration



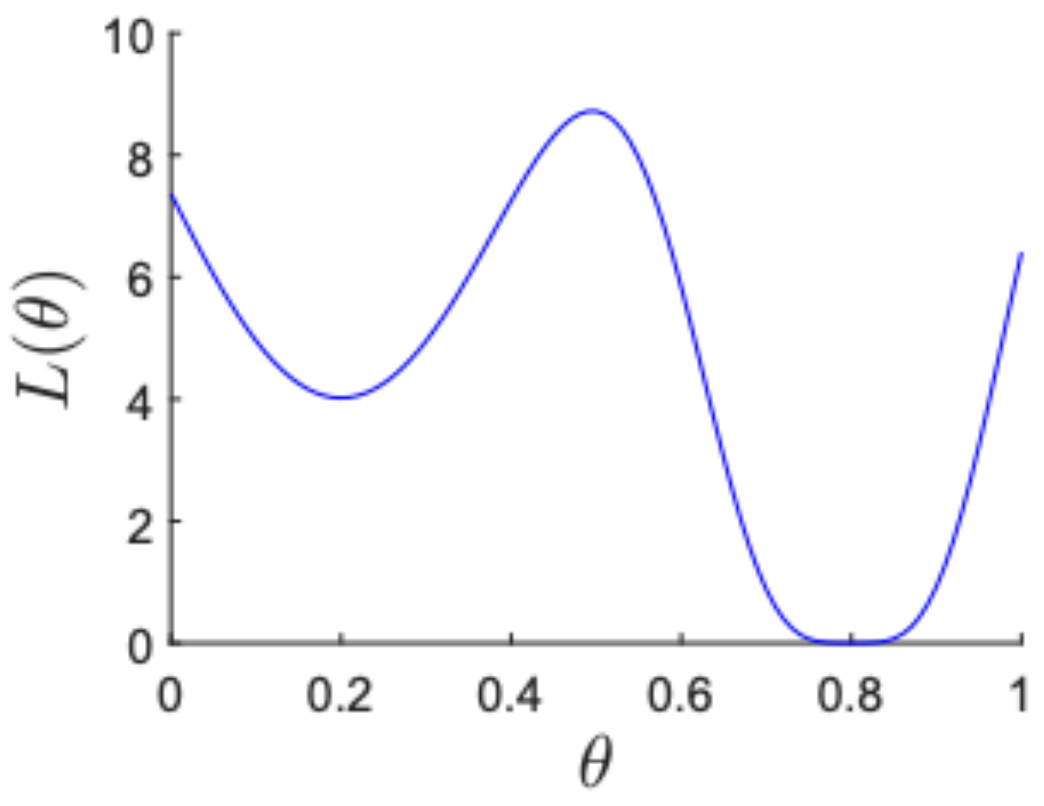
LIMITATIONS

- Non-differentiability of parallel edges of two separate triangles
  - can be resolved by applying a small perturbation to the vertices
- Interpenetration
- If/else conditions in procedural shaders (bitmap texture is 100% fine)

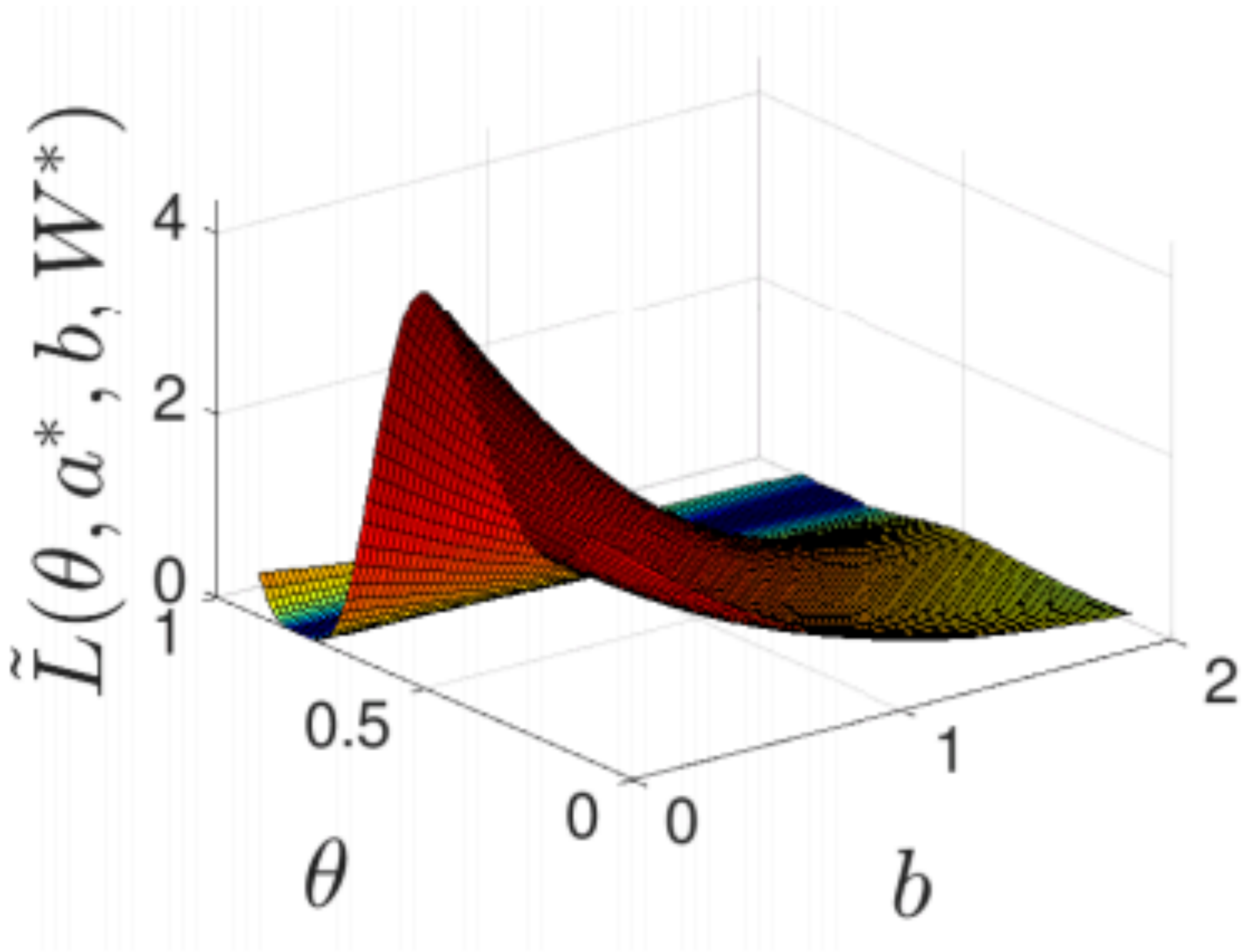


LIMITATIONS

- Non-differentiability of parallel edges of two separate triangles
  - can be resolved by applying a small perturbation to the vertices
- Interpenetration
- If/else conditions in procedural shaders (bitmap texture is 100% fine)
- Local minimum

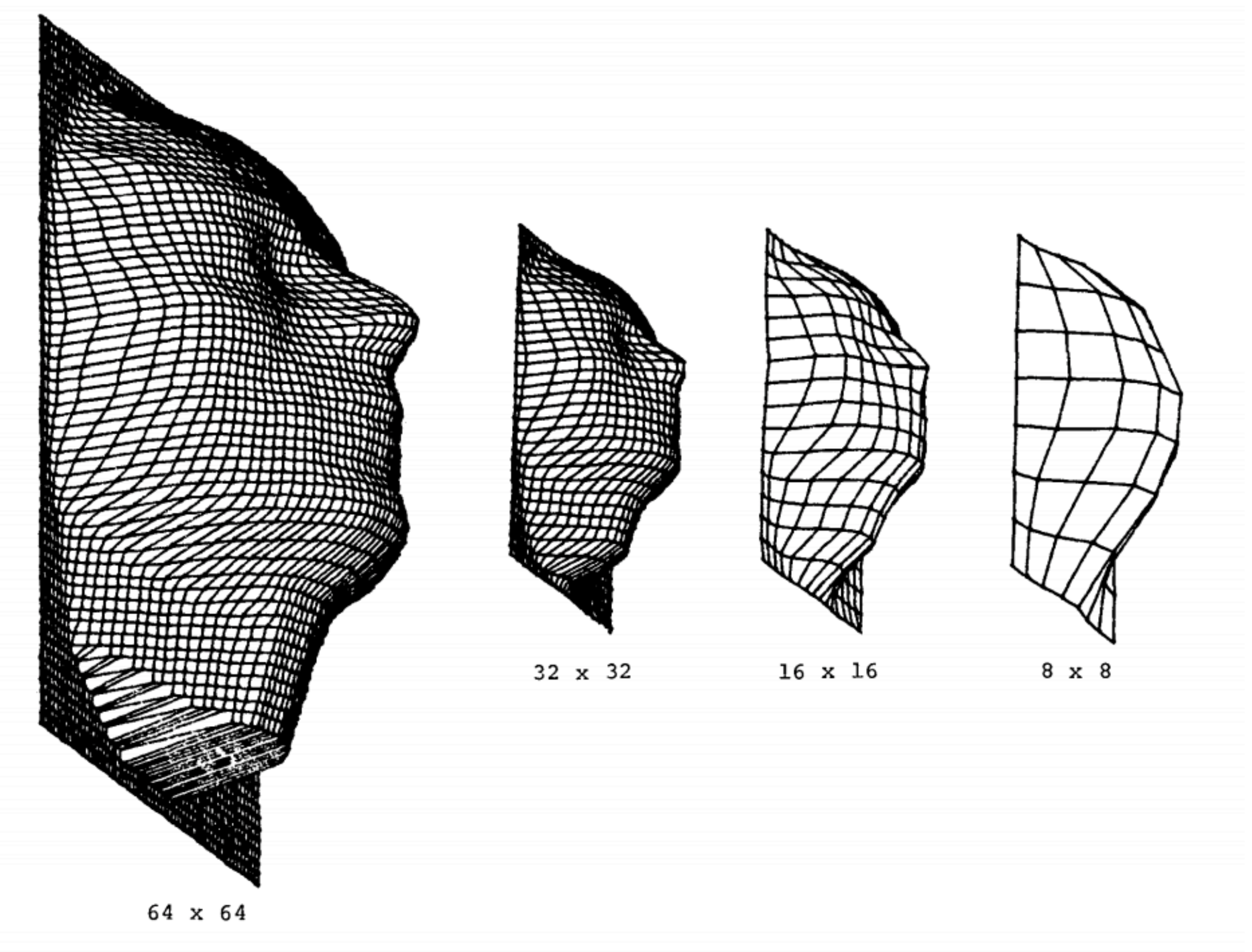


(a) original objective function L



(b) modified objective function $\tilde{L}$

Kawaguchi and Kaelbling 2019



William 1983

THEORY & ALGORITHMS

- Warm-up: differential irradiance

- Differentiable path tracing with edge sampling
- Differential radiative transfer

- Another way of dealing with discontinuities
- Radiative backpropagation

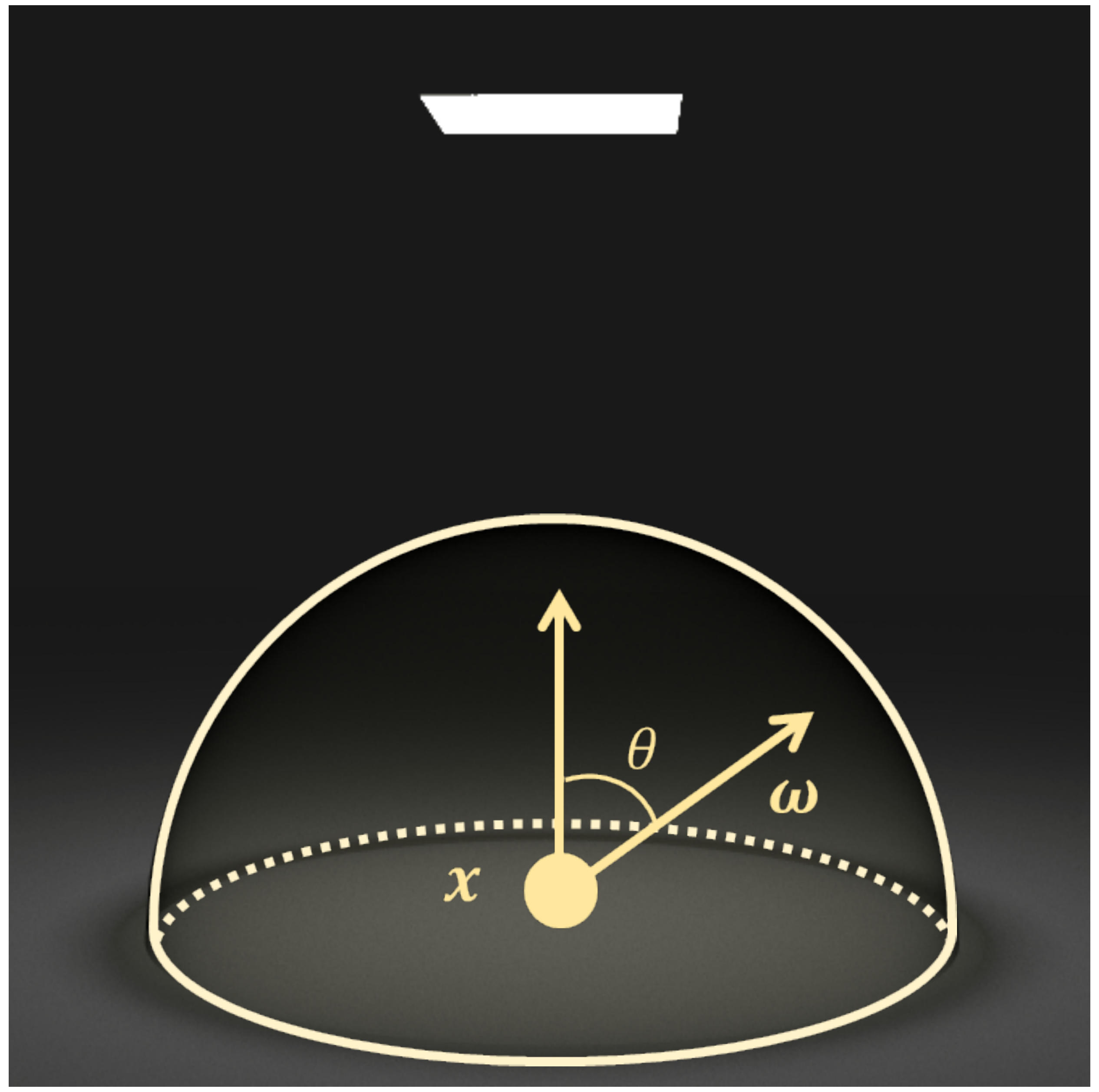
- Path-space differentiable rendering

WARM-UP: DIFFERENTIAL IRRADIANCE

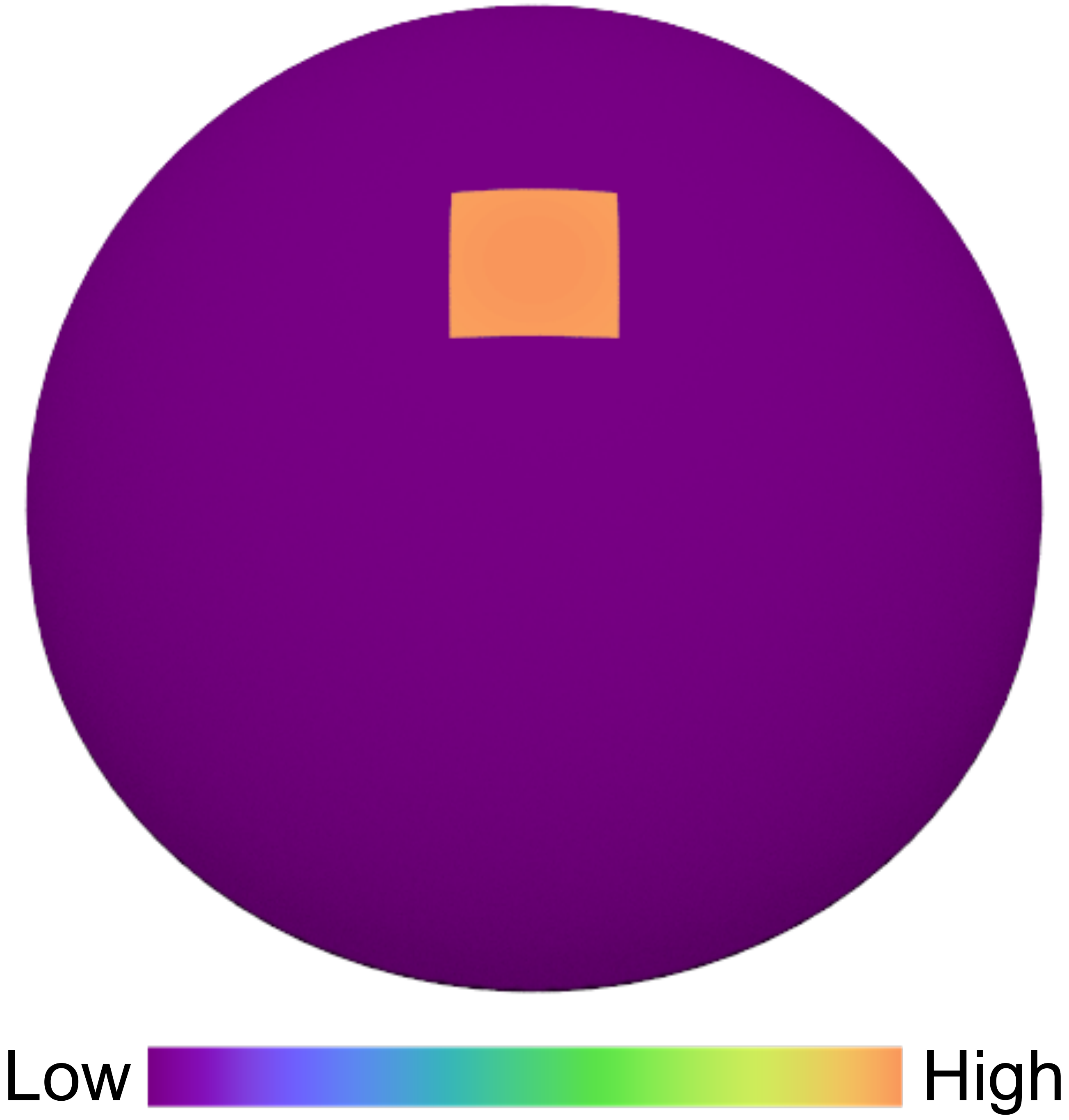
Irradiance at $\mathbf{x}$: $$E = \int_{\mathbb{H}^2} \overbrace{L_\mathrm{i}(\boldsymbol{\omega}) \, \cos\theta}^{f_E(\boldsymbol{\omega})} \, \mathrm{d}\sigma(\boldsymbol{\omega})$$

WARM-UP: DIFFERENTIAL IRRADIANCE

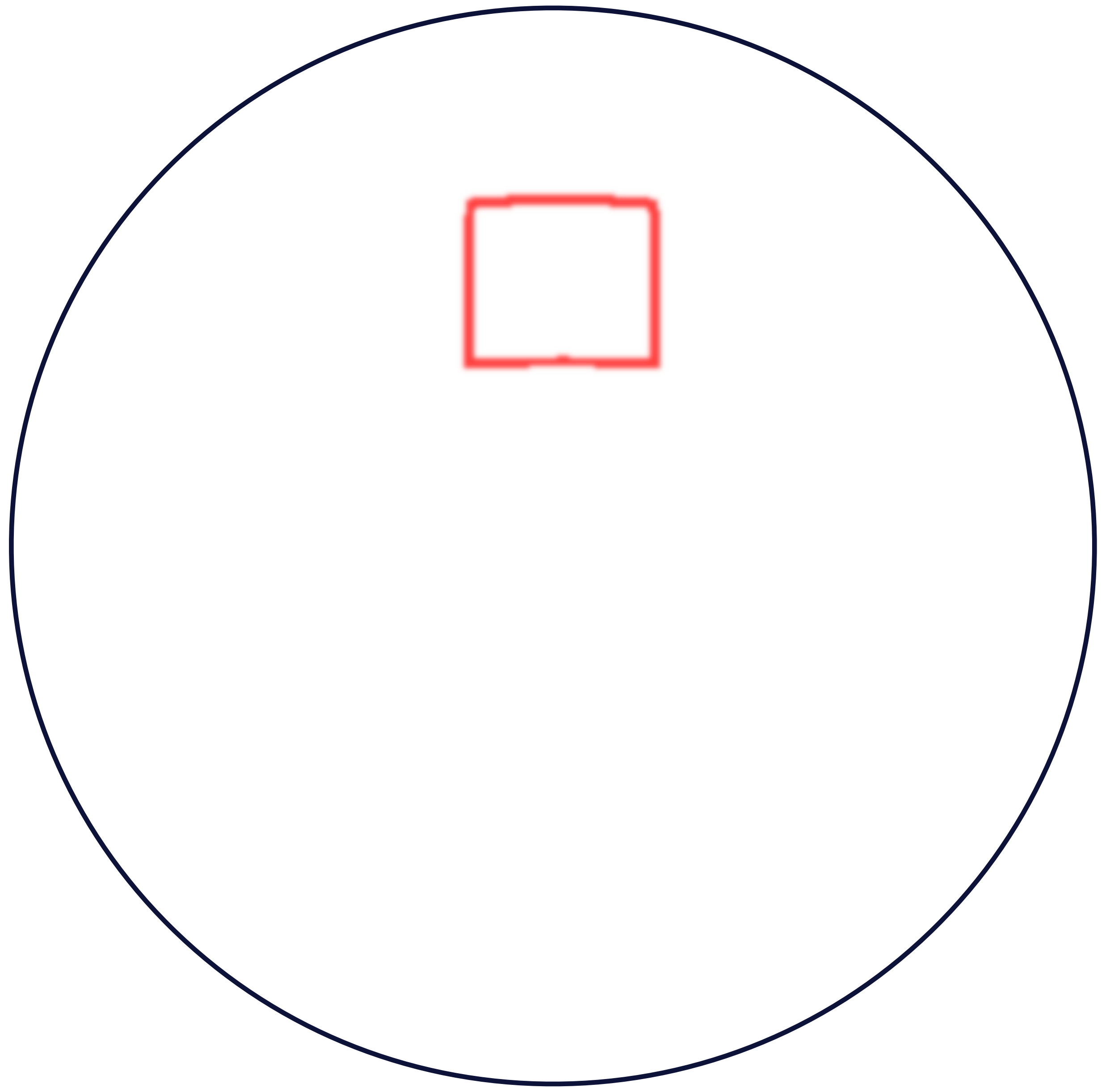
π: emitter size



$f_E(\omega)$



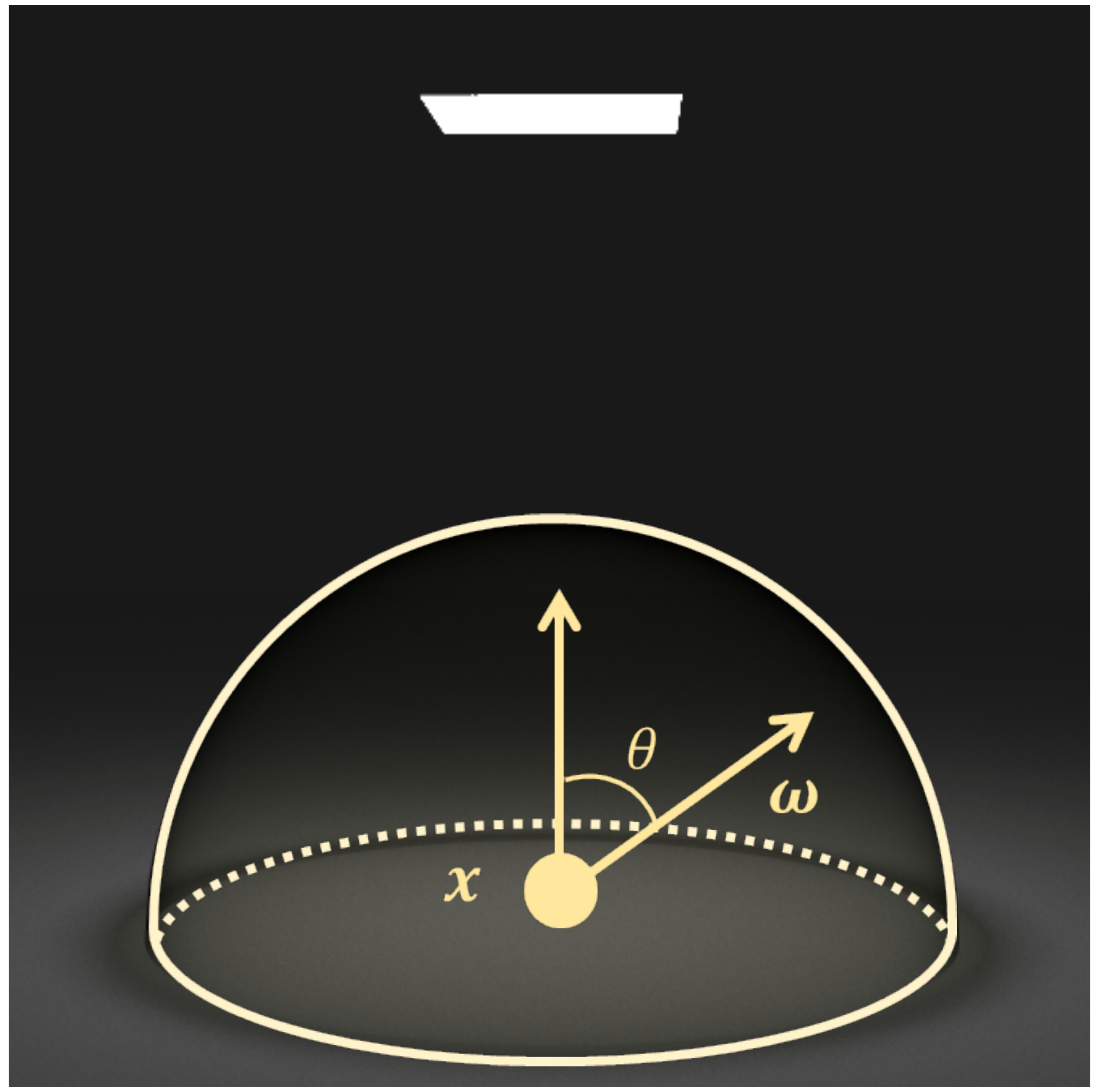
$\partial\mathbb{H}^2$



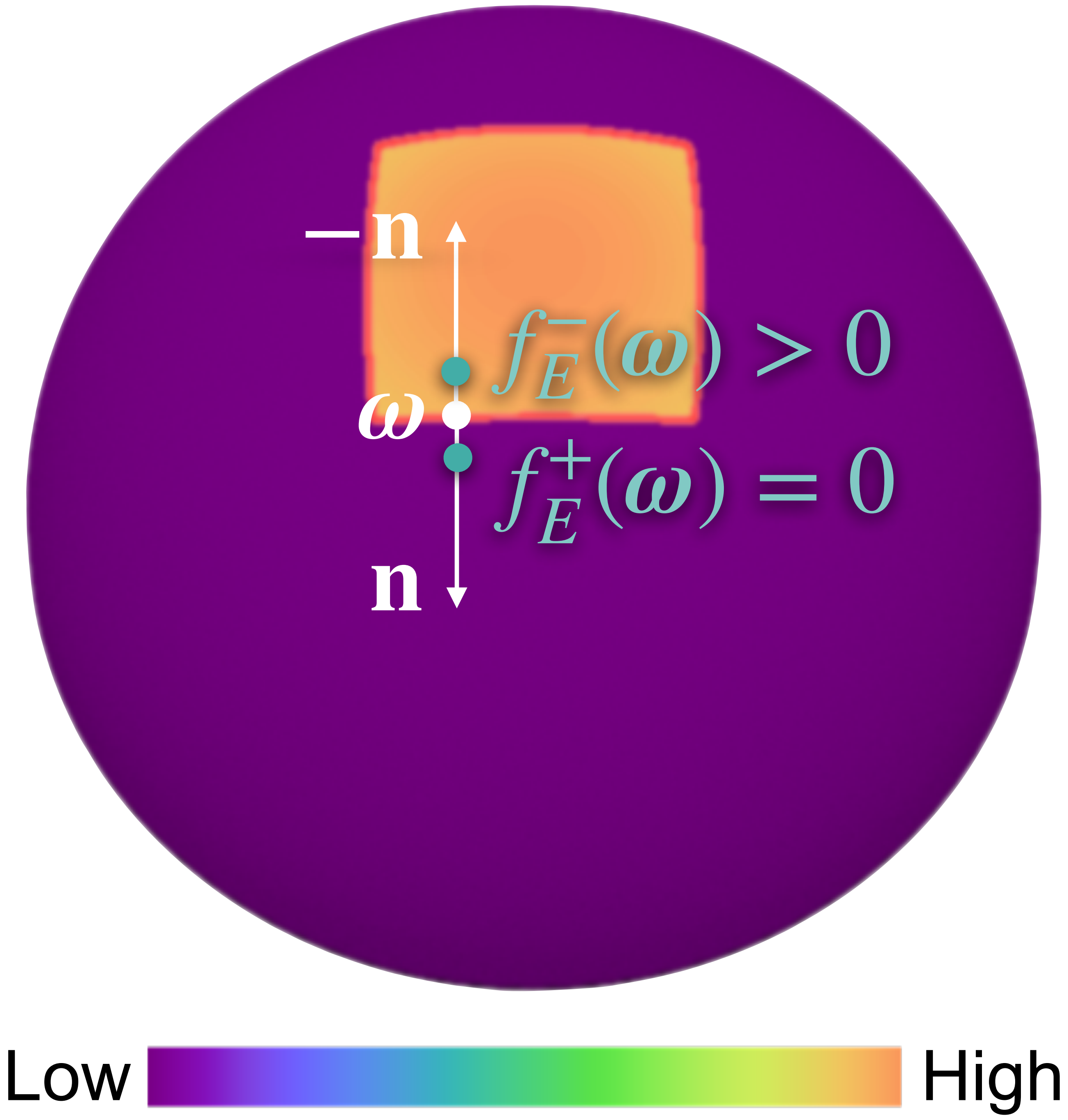
$$E = \int_{\mathbb{H}^2} \overbrace{L_i(\omega)\,\cos\theta}^{f_E(\omega)}\,\mathrm{d}\sigma(\omega) \xrightarrow{\text{Reynolds}} \frac{\mathrm{d}E}{\mathrm{d}\pi} = \underbrace{\int_{\mathbb{H}^2} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\omega)\,\mathrm{d}\sigma(\omega)}_{\text{Interior integral } = 0} + \underbrace{\int_{\partial\mathbb{H}^2} V_{\partial\mathbb{H}^2}(\omega)\,\Delta f_E(\omega)\,\mathrm{d}\ell(\omega)}_{\text{Boundary integral}}$$

WARM-UP: DIFFERENTIAL IRRADIANCE

π: emitter size



$f_E(\boldsymbol{\omega})$



Scalar normal "velocity" of $\boldsymbol{\omega}$

$$V_{\partial\mathbb{H}^2}(\boldsymbol{\omega}) = \left\langle \mathbf{n}(\boldsymbol{\omega}), \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}\pi} \right\rangle$$

independent of the parameterization of $\partial\mathbb{H}^2$

Difference of the integrand f_E across the boundary

$$\Delta f_E(\boldsymbol{\omega}) = f_E^-(\boldsymbol{\omega}) - f_E^+(\boldsymbol{\omega})$$

$$E = \int_{\mathbb{H}^2} \overbrace{L_{\mathrm{i}}(\boldsymbol{\omega}) \, \cos\theta}^{f_E(\boldsymbol{\omega})} \, \mathrm{d}\sigma(\boldsymbol{\omega})$$

Reynolds

General result

Interior integral

Boundary integral

$$\frac{\mathrm{d}E}{\mathrm{d}\pi} = \int_{\mathbb{H}^2} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\boldsymbol{\omega}) \, \mathrm{d}\sigma(\boldsymbol{\omega}) + \int_{\partial\mathbb{H}^2} V_{\partial\mathbb{H}^2}(\boldsymbol{\omega}) \, \Delta f_E(\boldsymbol{\omega}) \, \mathrm{d}\ell(\boldsymbol{\omega})$$

DIFFERENTIAL RENDERING EQUATION

$$E = \int_{\mathbb{H}^2} \overbrace{L_\mathrm{i}(\boldsymbol{\omega})\,\cos\theta}^{f_E(\boldsymbol{\omega})}\,\mathrm{d}\sigma(\boldsymbol{\omega}) \xrightarrow{\text{Reynolds}} \frac{\mathrm{d}E}{\mathrm{d}\pi} = \underbrace{\int_{\mathbb{H}^2} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\boldsymbol{\omega})\,\mathrm{d}\sigma(\boldsymbol{\omega})}_{\text{Interior integral}} + \underbrace{\int_{\partial\mathbb{H}^2} V_{\partial\mathbb{H}^2}(\boldsymbol{\omega})\,\Delta f_E(\boldsymbol{\omega})\,\mathrm{d}\ell(\boldsymbol{\omega})}_{\text{Boundary integral}}$$

This can be generalized easily to obtain the differential rendering equation:

Rendering equation

$$L(\boldsymbol{\omega}_\mathrm{o}) = \int_{\mathbb{S}^2} \overbrace{L_\mathrm{i}(\boldsymbol{\omega}_\mathrm{i})\,f_s(\boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})}^{f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})}\,\mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i}) + L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

f_s : cosine-weighted BSDF

$\Big\downarrow$ Reynolds

Differential rendering equation

$$\boxed{\frac{\mathrm{d}}{\mathrm{d}\pi}L(\boldsymbol{\omega}_\mathrm{o}) = \underbrace{\int_{\mathbb{S}^2} \frac{\mathrm{d}}{\mathrm{d}\pi} f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})\,\mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i})}_{\text{Interior integral}} + \underbrace{\int_{\partial\mathbb{S}^2} V_{\partial\mathbb{S}^2}(\boldsymbol{\omega}_\mathrm{i})\,\Delta f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})\,\mathrm{d}\ell(\boldsymbol{\omega}_\mathrm{i})}_{\text{Boundary integral}} + \frac{\mathrm{d}}{\mathrm{d}\pi}L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})}$$

SOURCES OF DISCONTINUITIES

Assumptions:

No **zero-measure** (point and directional) **lights**

(which can create *hard shadow boundaries*)

Hard-to-detect discontinuities

No **perfectly specular surfaces**

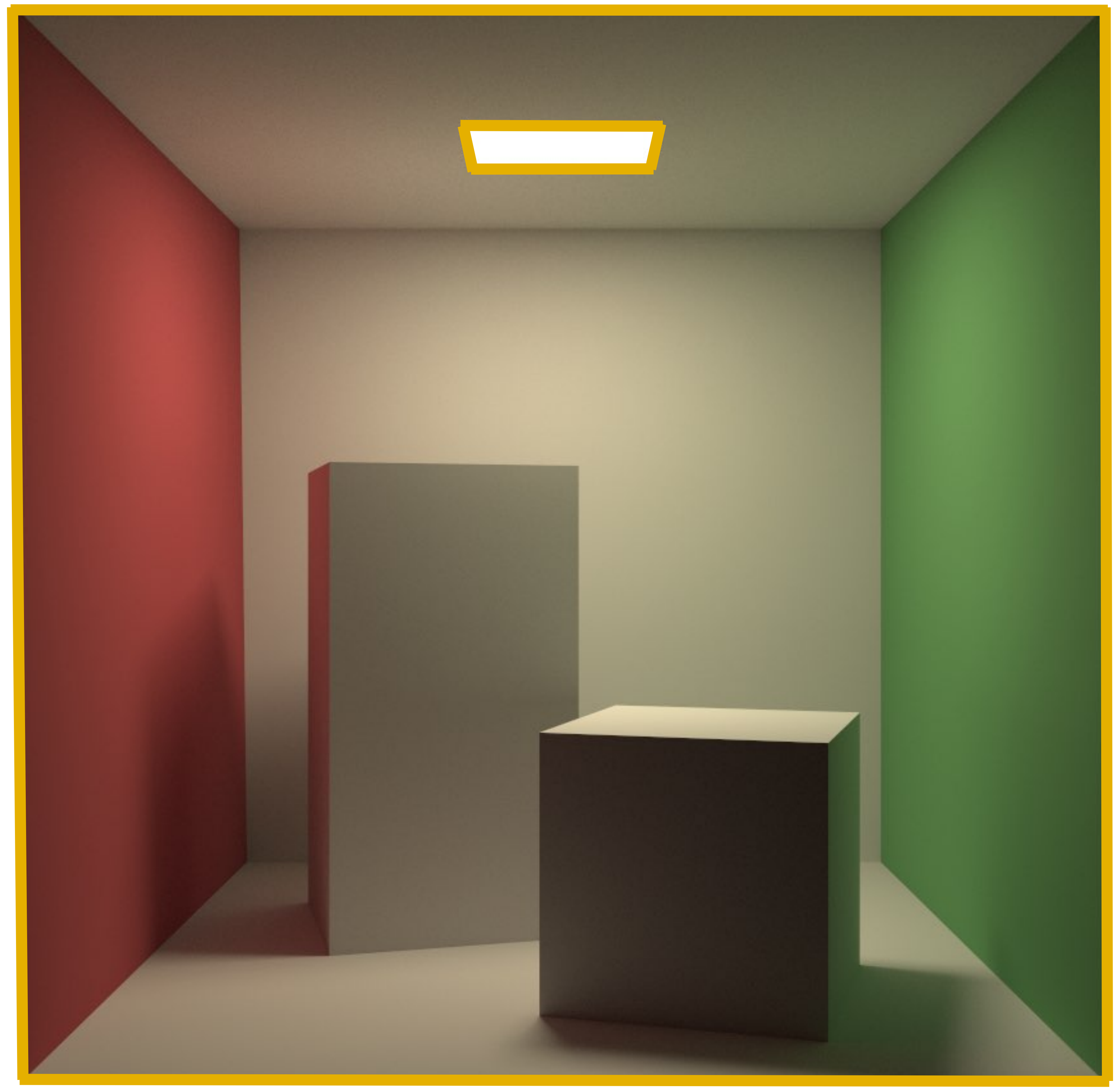
(which can create *virtual images* of other objects)

Continuous BSDFs

These limitations are largely practical and can be easily mitigated

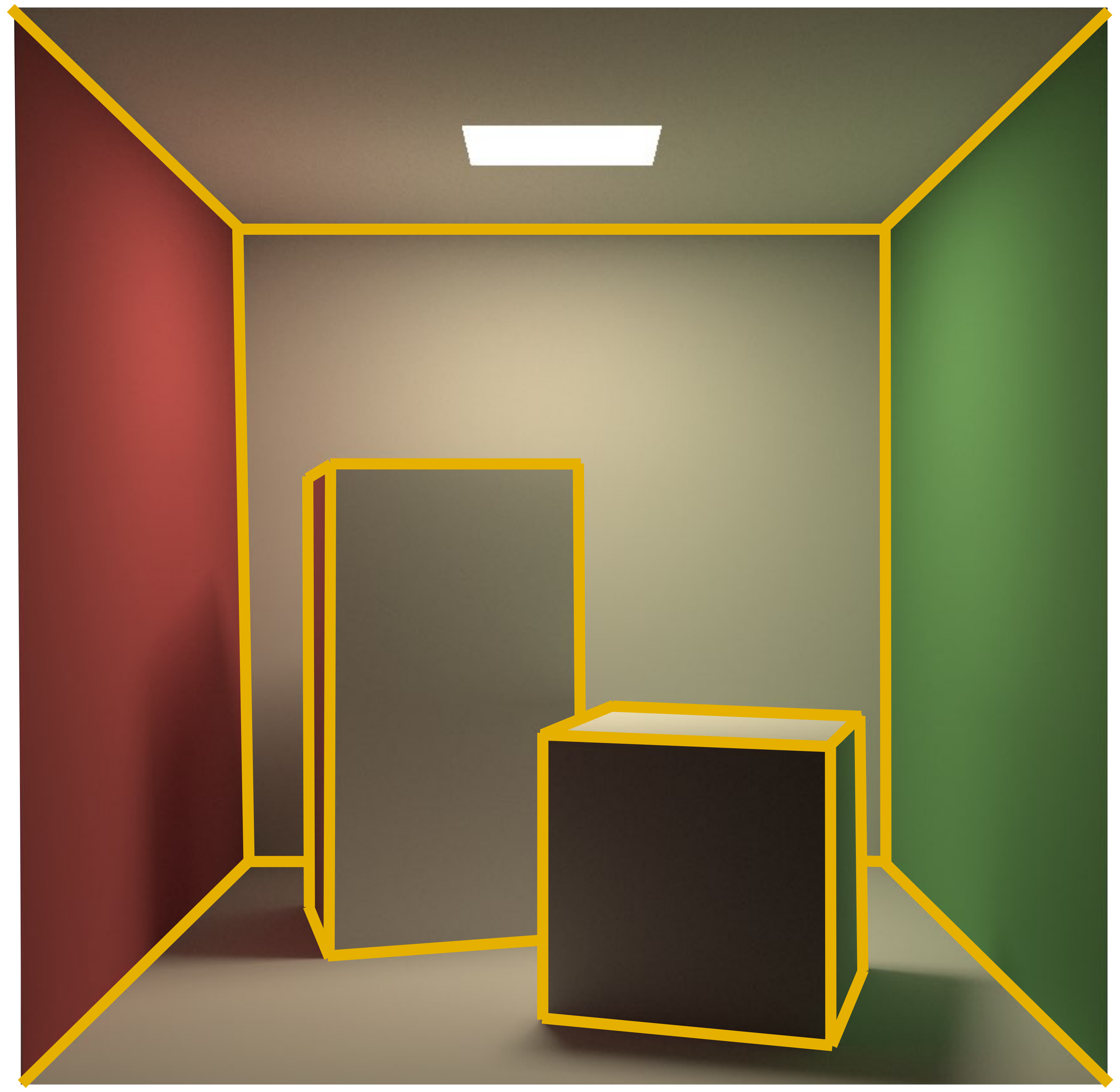
SOURCES OF DISCONTINUITIES

Boundary edges



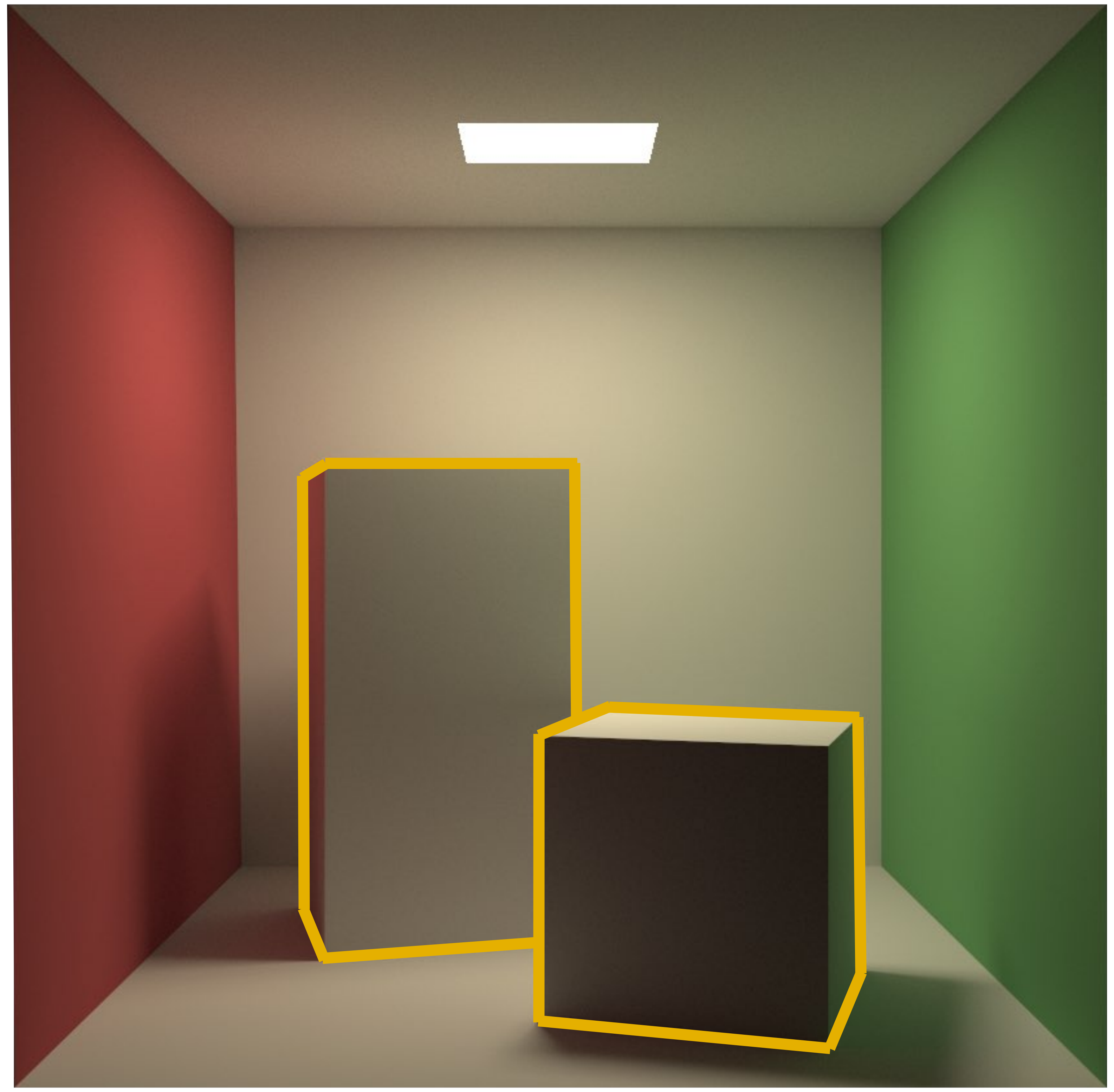
(Topological) boundary of an object

Sharp edges



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

Silhouette edges



View-dependent object silhouettes

SIGGRAPH 2020 S2020.SIGGRAPH.ORG THINK BEYOND

Path tracing can be generalized to estimate L and $\mathrm{d}L/\mathrm{d}\pi$ jointly

Rendering equation

$$L(\boldsymbol{\omega}_\mathrm{o}) = \boxed{\int_{\mathbb{S}^2} \overbrace{f_s(\boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})\, L_\mathrm{i}(\boldsymbol{\omega}_\mathrm{i})}^{f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})}\, \mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i})} + L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

Differential rendering equation

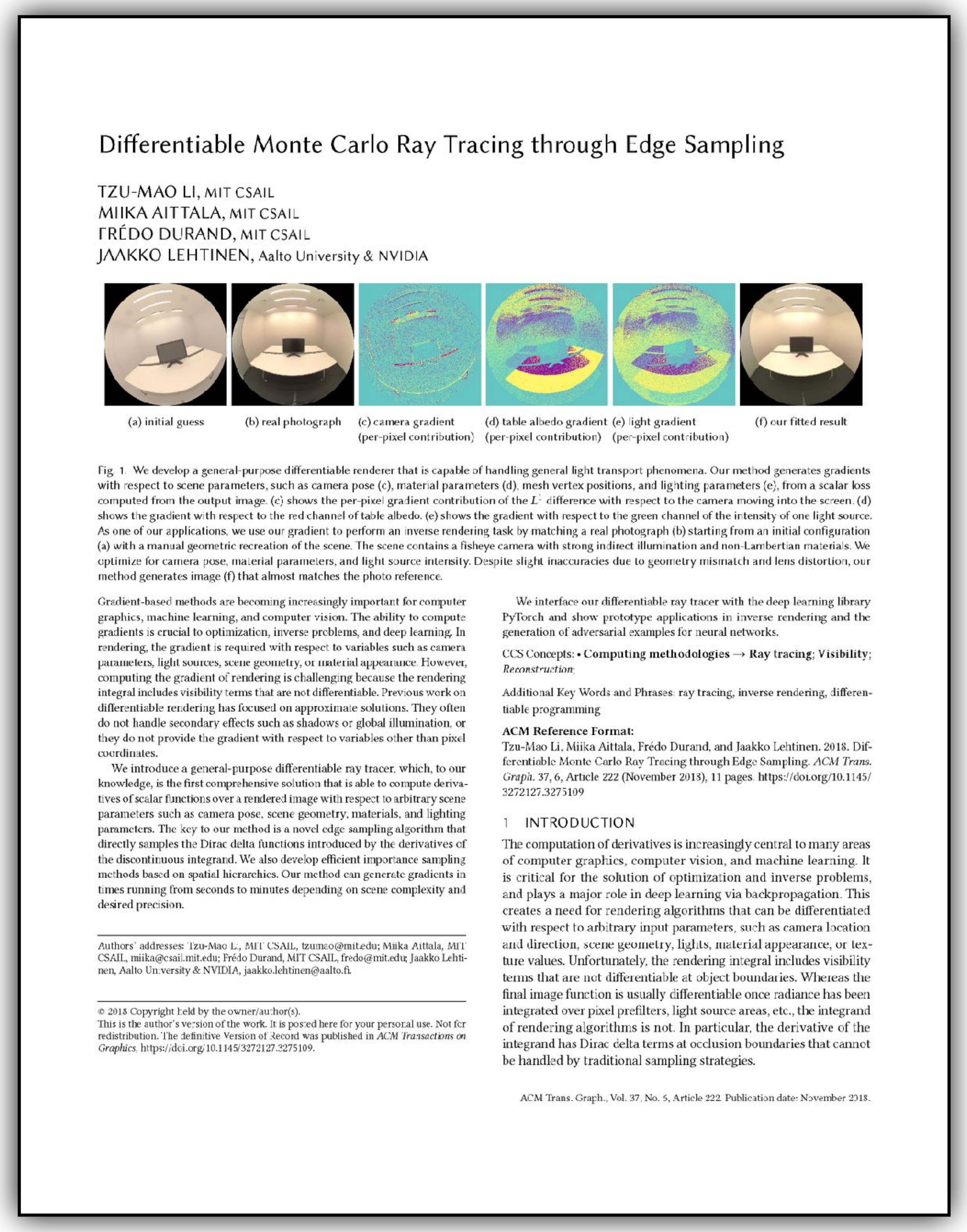
Interior integral

Boundary integral

$$\frac{\mathrm{d}}{\mathrm{d}\pi} L(\boldsymbol{\omega}_\mathrm{o}) = \boxed{\int_{\mathbb{S}^2} \frac{\mathrm{d}}{\mathrm{d}\pi} f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})\, \mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i})} + \boxed{\int_{\partial\mathbb{S}^2} V_{\partial\mathbb{S}^2}(\boldsymbol{\omega}_\mathrm{i})\, \Delta f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})\, \mathrm{d}\ell(\boldsymbol{\omega}_\mathrm{i})} + \frac{\mathrm{d}}{\mathrm{d}\pi} L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

Standard path tracing

Edge sampling



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

(a) initial guess (b) real photograph (c) camera gradient (per-pixel contribution) (d) table albedo gradient (per-pixel contribution) (e) light gradient (per-pixel contribution) (f) our fitted result

Fig. 1. We develop a general-purpose differentiable renderer that is capable of handling general light transport phenomena. Our method generates gradients with respect to scene parameters, such as camera pose (c), material parameters (d), mesh vertex positions, and lighting parameters (e), from a scalar loss computed from the output image. (c) shows the per-pixel gradient contribution of the L^1 difference with respect to the camera moving into the screen. (d) shows the gradient with respect to the red channel of table albedo. (e) shows the gradient with respect to the green channel of the intensity of one light source. As one of our applications, we use our gradient to perform an inverse rendering task by matching a real photograph (b) starting from an initial configuration (a) with a manual geometric recreation of the scene. The scene contains a fisheye camera with strong indirect illumination and non-Lambertian materials. We optimize for camera pose, material parameters, and light source intensity. Despite slight inaccuracies due to geometry mismatch and lens distortion, our method generates image (f) that almost matches the photo reference.

Gradient-based methods are becoming increasingly important for computer graphics, machine learning, and computer vision. The ability to compute gradients is crucial to optimization, inverse problems, and deep learning. In rendering, the gradient is required with respect to variables such as camera parameters, light sources, scene geometry, or material appearance. However, computing the gradient of rendering is challenging because the rendering integral includes visibility terms that are not differentiable. Previous work on differentiable rendering has focused on approximate solutions. They often do not handle secondary effects such as shadows or global illumination, or they do not provide the gradient with respect to variables other than pixel coordinates.

We introduce a general-purpose differentiable ray tracer, which, to our knowledge, is the first comprehensive solution that is able to compute derivatives of scalar functions over a rendered image with respect to arbitrary scene parameters such as camera pose, scene geometry, materials, and lighting parameters. The key to our method is a novel edge sampling algorithm that directly samples the Dirac delta functions introduced by the derivatives of the discontinuous integrand. We also develop efficient importance sampling methods based on spatial hierarchies. Our method can generate gradients in times running from seconds to minutes depending on scene complexity and desired precision.

We interface our differentiable ray tracer with the deep learning library PyTorch and show prototype applications in inverse rendering and the generation of adversarial examples for neural networks.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; **Visibility**; *Reconstruction*;

Additional Key Words and Phrases: ray tracing, inverse rendering, differentiable programming

ACM Reference Format:
Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Trans. Graph.* 37, 6, Article 222 (November 2018), 11 pages. https://doi.org/10.1145/3272127.3275109

1 INTRODUCTION

The computation of derivatives is increasingly central to many areas of computer graphics, computer vision, and machine learning. It is critical for the solution of optimization and inverse problems, and plays a major role in deep learning via backpropagation. This creates a need for rendering algorithms that can be differentiated with respect to arbitrary input parameters, such as camera location and direction, scene geometry, lights, material appearance, or texture values. Unfortunately, the rendering integral includes visibility terms that are not differentiable at object boundaries. Whereas the final image function is usually differentiable once radiance has been integrated over pixel prefilters, light source areas, etc., the integrand of rendering algorithms is not. In particular, the derivative of the integrand has Dirac delta terms at occlusion boundaries that cannot be handled by traditional sampling strategies.

Authors' addresses: Tzu-Mao Li, MIT CSAIL, tzumao@mit.edu; Miika Aittala, MIT CSAIL, miika@csail.mit.edu; Frédo Durand, MIT CSAIL, fredo@mit.edu; Jaakko Lehtinen, Aalto University & NVIDIA, jaakko.lehtinen@aalto.fi.

© 2018 Copyright held by the owner/author(s).
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM Transactions on Graphics*, https://doi.org/10.1145/3272127.3275109.

ACM Trans. Graph., Vol. 37, No. 6, Article 222. Publication date: November 2018.

Differentiable Monte Carlo Ray Tracing through Edge Sampling

Tzu-Mao Li, Miika Aittala, Frédo Durand, Jaakko Lehtinen

SIGGRAPH Asia 2018

SIGGRAPH THINK BEYOND 2020 S2020.SIGGRAPH.ORG

$\mathrm{dPT}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$: # Estimate $L(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$ and $\frac{\mathrm{d}}{\mathrm{d}\pi}[L(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})]$ jointly

sample $\boldsymbol{\omega}_{\mathrm{i},1} \in \mathbb{S}^2$ with probability $p_{\mathrm{i},1}$

$\mathbf{y} \leftarrow$ rayIntersect$(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1})$

$(L_\mathrm{i}, \dot{L}_\mathrm{i}) \leftarrow \mathrm{dPT}(\mathbf{y}, -\boldsymbol{\omega}_{\mathrm{i},1})$

$$L \leftarrow \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})\, L_\mathrm{i}}{p_{\mathrm{i},1}}$$

$$\dot{L} \leftarrow \frac{\frac{\mathrm{d}}{\mathrm{d}\pi}[f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})]\, L_\mathrm{i} + f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})\, \dot{L}_\mathrm{i}}{p_{\mathrm{i},1}}$$

Standard PT w/ symbolic differentiation

sample $\boldsymbol{\omega}_{\mathrm{i},2} \in \partial\mathbb{S}^2$ with probability $p_{\mathrm{i},2}$

$$\dot{L} \leftarrow \dot{L} + \frac{V_{\partial\mathbb{S}^2}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})\, f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2}, \boldsymbol{\omega}_\mathrm{o})\, \Delta L_\mathrm{i}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})}{p_{\mathrm{i},2}}$$

Monte Carlo edge sampling

return $\left(L + L_\mathrm{e}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o}),\ \dot{L} + \frac{\mathrm{d}}{\mathrm{d}\pi} L_\mathrm{e}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})\right)$

$\Delta f_\mathrm{RE} = \Delta(f_s\, L_\mathrm{i}) = f_s\, \Delta L_\mathrm{i}$

(assuming f_s to be continuous)

Rendering equation

$$L(\boldsymbol{\omega}_\mathrm{o}) = \int_{\mathbb{S}^2} \overbrace{f_s(\boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})\, L_\mathrm{i}(\boldsymbol{\omega}_\mathrm{i})}^{f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})}\, \mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i}) + L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

Differential rendering equation

$$\frac{\mathrm{d}}{\mathrm{d}\pi} L(\boldsymbol{\omega}_\mathrm{o}) = \int_{\mathbb{S}^2} \frac{\mathrm{d}}{\mathrm{d}\pi} f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})\, \mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i})$$

$$+ \int_{\partial\mathbb{S}^2} V_{\partial\mathbb{S}^2}(\boldsymbol{\omega}_\mathrm{i})\, \Delta f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})\, \mathrm{d}\ell(\boldsymbol{\omega}_\mathrm{i})$$

$$+ \frac{\mathrm{d}}{\mathrm{d}\pi} L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

MONTE CARLO EDGE SAMPLING

$\mathrm{dPT}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$: # Estimate $L(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$ and $\frac{\mathrm{d}}{\mathrm{d}\pi}[L(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})]$ jointly

sample $\boldsymbol{\omega}_{\mathrm{i},1} \in \mathbb{S}^2$ with probability $p_{\mathrm{i},1}$

$\mathbf{y} \leftarrow$ rayIntersect$(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1})$

$(L_\mathrm{i}, \dot{L}_\mathrm{i}) \leftarrow \mathrm{dPT}(\mathbf{y}, -\boldsymbol{\omega}_{\mathrm{i},1})$

$$L \leftarrow \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})\, L_\mathrm{i}}{p_{\mathrm{i},1}}$$

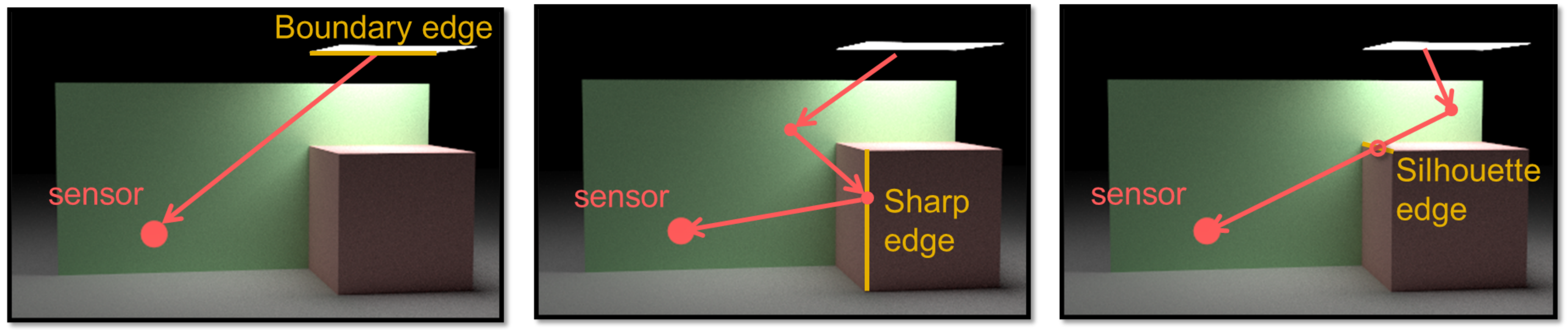
$$\dot{L} \leftarrow \frac{\frac{\mathrm{d}}{\mathrm{d}\pi}[f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})]\, L_\mathrm{i} + f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})\, \dot{L}_\mathrm{i}}{p_{\mathrm{i},1}}$$

sample $\boldsymbol{\omega}_{\mathrm{i},2} \in \partial\mathbb{S}^2$ with probability $p_{\mathrm{i},2}$

$$\dot{L} \leftarrow \dot{L} + \frac{V_{\partial\mathbb{S}^2}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})\, f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2}, \boldsymbol{\omega}_\mathrm{o})\, \Delta L_\mathrm{i}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})}{p_{\mathrm{i},2}}$$

return $\left(L + L_\mathrm{e}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o}),\ \dot{L} + \frac{\mathrm{d}}{\mathrm{d}\pi} L_\mathrm{e}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})\right)$

- A new sampling procedure introduced by Li et al. [2018]
- **Key:** determining $\partial\mathbb{S}^2$, the discontinuity points of ΔL_i (w.r.t. incident direction $\boldsymbol{\omega}_\mathrm{i}$)



- For polygonal meshes, $\partial\mathbb{S}^2$ can involve:
  - Boundary edges (associated with only one face)
  - Face edges (when not using smooth shading)
  - Silhouette edges (shared by a front and a back face)
    - Requires traversing a 6D BVH
    - Expensive for complex scenes
    - To be addressed later!

COMPUTING ΔL_{i}

$\mathrm{dPT}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{o}})$: # Estimate $L(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{o}})$ and $\frac{\mathrm{d}}{\mathrm{d}\pi}[L(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{o}})]$ jointly

sample $\boldsymbol{\omega}_{\mathrm{i},1} \in \mathbb{S}^2$ with probability $p_{\mathrm{i},1}$

$\mathbf{y} \leftarrow$ rayIntersect$(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1})$

$(L_{\mathrm{i}}, \dot{L}_{\mathrm{i}}) \leftarrow \mathrm{dPT}(\mathbf{y}, -\boldsymbol{\omega}_{\mathrm{i},1})$

$$L \leftarrow \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_{\mathrm{o}})\, L_{\mathrm{i}}}{p_{\mathrm{i},1}}$$

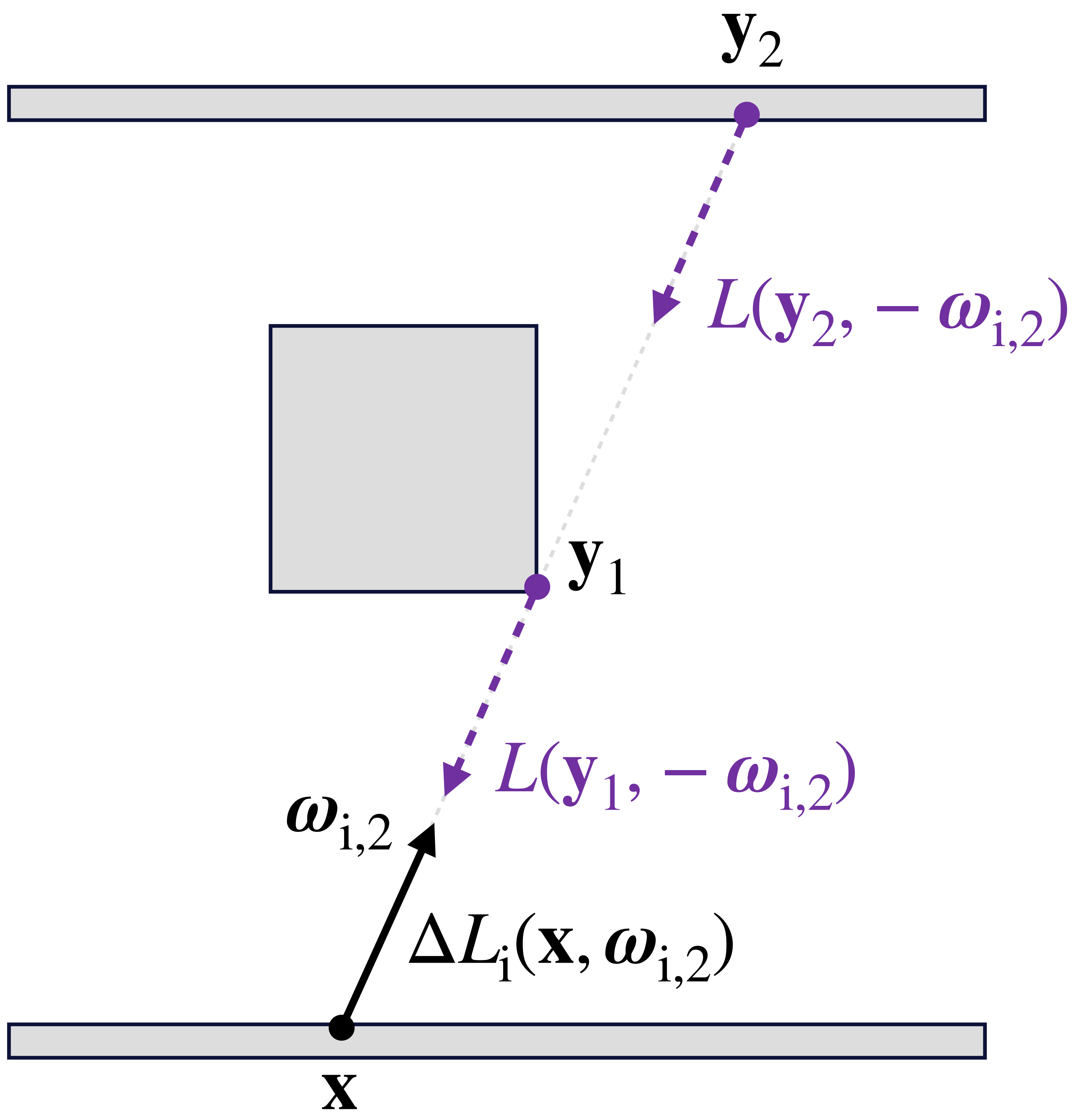
$$\dot{L} \leftarrow \frac{\frac{\mathrm{d}}{\mathrm{d}\pi}[f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_{\mathrm{o}})]\, L_{\mathrm{i}} + f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_{\mathrm{o}})\, \dot{L}_{\mathrm{i}}}{p_{\mathrm{i},1}}$$

sample $\boldsymbol{\omega}_{\mathrm{i},2} \in \partial\mathbb{S}^2$ with probability $p_{\mathrm{i},2}$

$$\boxed{\dot{L} \leftarrow \dot{L} + \frac{V_{\partial\mathbb{S}^2}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})\, f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2}, \boldsymbol{\omega}_{\mathrm{o}})\, \Delta L_{\mathrm{i}}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})}{p_{\mathrm{i},2}}}$$

Monte Carlo edge sampling

return $\left(L + L_{\mathrm{e}}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{o}}),\ \dot{L} + \frac{\mathrm{d}}{\mathrm{d}\pi} L_{\mathrm{e}}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{o}})\right)$



$$\Delta L_{\mathrm{i}}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2}) = \pm \left[L(\mathbf{y}_1, -\boldsymbol{\omega}_{\mathrm{i},2}) - L(\mathbf{y}_2, -\boldsymbol{\omega}_{\mathrm{i},2})\right]$$

Radiance values $L(\mathbf{y}_1, -\boldsymbol{\omega}_{\mathrm{i},2})$ and $L(\mathbf{y}_2, -\boldsymbol{\omega}_{\mathrm{i},2})$ can be computed by tracing additional "side" paths

$\mathrm{dPT}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$: # Estimate $L(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})$ and $\frac{\mathrm{d}}{\mathrm{d}\pi}[L(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})]$ jointly

sample $\boldsymbol{\omega}_{\mathrm{i},1} \in \mathbb{S}^2$ with probability $p_{\mathrm{i},1}$

$\mathbf{y} \leftarrow$ rayIntersect$(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1})$

$(L_\mathrm{i}, \dot{L}_\mathrm{i}) \leftarrow \mathrm{dPT}(\mathbf{y}, -\boldsymbol{\omega}_{\mathrm{i},1})$

$$L \leftarrow \frac{f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})\, L_\mathrm{i}}{p_{\mathrm{i},1}}$$

$$\dot{L} \leftarrow \frac{\frac{\mathrm{d}}{\mathrm{d}\pi}[f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})]\, L_\mathrm{i} + f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},1}, \boldsymbol{\omega}_\mathrm{o})\, \dot{L}_\mathrm{i}}{p_{\mathrm{i},1}}$$

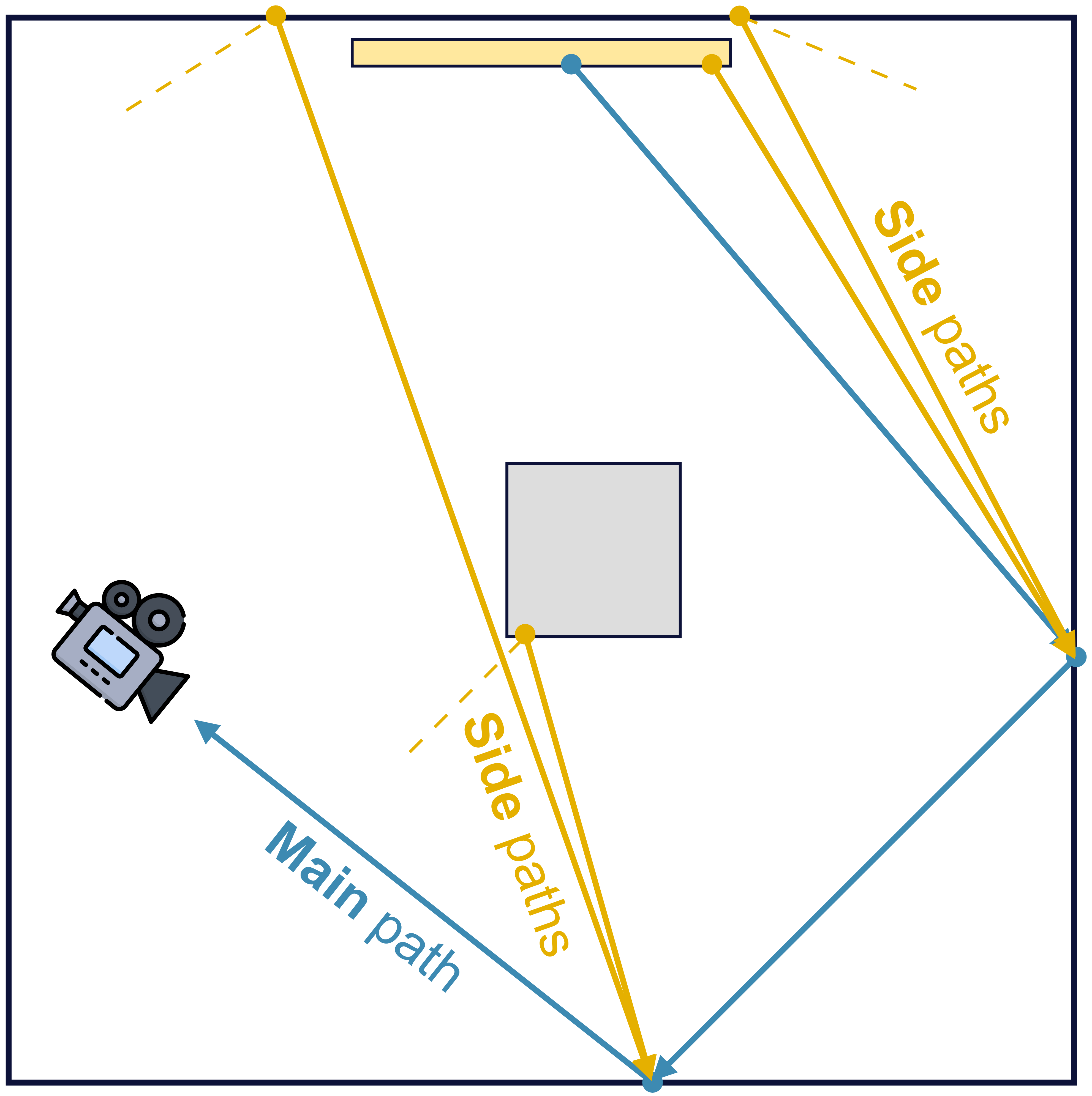
Standard PT w/ symbolic differentiation

sample $\boldsymbol{\omega}_{\mathrm{i},2} \in \partial\mathbb{S}^2$ with probability $p_{\mathrm{i},2}$

$$\dot{L} \leftarrow \dot{L} + \frac{V_{\partial\mathbb{S}^2}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})\, f_s(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2}, \boldsymbol{\omega}_\mathrm{o})\, \Delta L_\mathrm{i}(\mathbf{x}, \boldsymbol{\omega}_{\mathrm{i},2})}{p_{\mathrm{i},2}}$$

Monte Carlo edge sampling

return $\left(L + L_\mathrm{e}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o}),\ \dot{L} + \frac{\mathrm{d}}{\mathrm{d}\pi} L_\mathrm{e}(\mathbf{x}, \boldsymbol{\omega}_\mathrm{o})\right)$



A Differential Theory of Radiative Transfer

CHENG ZHANG, University of California, Irvine
LIFAN WU, University of California, San Diego
CHANGXI ZHENG, Columbia University
IOANNIS GKIOULEKAS, Carnegie Mellon University
RAVI RAMAMOORTHI, University of California, San Diego
SHUANG ZHAO, University of California, Irvine

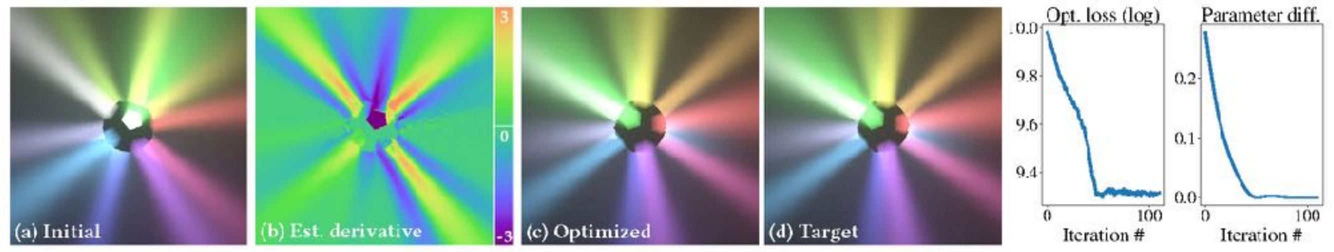


Fig. 1. We introduce a new differential theory of radiative transfer, which lays the foundation for computing the derivatives of radiometric measures with respect to arbitrary scene parameterizations (e.g., material properties and object geometries). The ability to evaluate these derivatives can facilitate gradient-based optimization for many diverse applications. As an example, here we optimize the pose of a dodecahedron emitting colored beams inside a participating medium. Given a target image (d) and an initial configuration (a), the optimization uses derivatives estimated by our method (b) to find parameters that produce rendered images (c) closely matching the target. Per-iteration optimization loss and difference between true and estimated parameters (both measured in L_2) are plotted on the right.

Physics-based differentiable rendering is the task of estimating the derivatives of radiometric measures with respect to scene parameters. The ability to compute these derivatives is necessary for enabling gradient-based optimization in a diverse array of applications: from solving analysis-by-synthesis problems to training machine learning pipelines incorporating forward rendering processes. Unfortunately, physics based differentiable rendering remains challenging, due to the complex and typically nonlinear relation between pixel intensities and scene parameters.

We introduce a differential theory of radiative transfer, which shows how individual components of the radiative transfer equation (RTE) can be differentiated with respect to arbitrary differentiable changes of a scene. Our theory encompasses the same generality as the standard RTE, allowing differentiation while accurately handling a large range of light transport phenomena such as volumetric absorption and scattering, anisotropic phase functions, and heterogeneity. To numerically estimate the derivatives given by our theory, we introduce an unbiased Monte Carlo estimator supporting arbitrary surface and volumetric configurations. Our technique differentiates path contributions symbolically and uses additional boundary integrals to capture geometric discontinuities such as visibility changes.

We validate our method by comparing our derivative estimations to those generated using the finite-difference method. Furthermore, we use a few synthetic examples inspired by real-world applications in inverse rendering, non line of sight (NLOS) and biomedical imaging, and design, to demonstrate the practical usefulness of our technique.

CCS Concepts: • **Computing methodologies → Rendering.**

Additional Key Words and Phrases: radiative transfer, differentiable rendering, Monte Carlo path tracing

ACM Reference Format:
Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A Differential Theory of Radiative Transfer. *ACM Trans. Graph.* 38, 6, Article 227 (November 2019), 16 pages. https://doi.org/10.1145/3355089.3356522

Authors' addresses: Cheng Zhang, University of California, Irvine, chengz20@uci.edu; Lifan Wu, University of California, San Diego, liw086@eng.ucsd.edu; Changxi Zheng, Columbia University, cxz@cs.columbia.edu; Ioannis Gkioulekas, Carnegie Mellon University, igkioule@andrew.cmu.edu; Ravi Ramamoorthi, University of California, San Diego, ravir@cs.ucsd.edu; Shuang Zhao, University of California, Irvine, shz@ics.uci.edu.

0730-0301/2019/11-ART227 $15.00
https://doi.org/10.1145/3355089.3356522

1 INTRODUCTION

A fundamental task of physics-based light transport simulation is to compute the radiant power (generally measured using radiance) at certain 3D locations and directions in a virtual scene, e.g., those corresponding to radiometric sensors. Such *forward* evaluations of light transport have been a focus of research efforts in computer graphics since the field's inception. These efforts have resulted in mature forward rendering algorithms, including Monte Carlo techniques, that can efficiently and accurately simulate complex light transport effects such as interreflections and subsurface scattering.

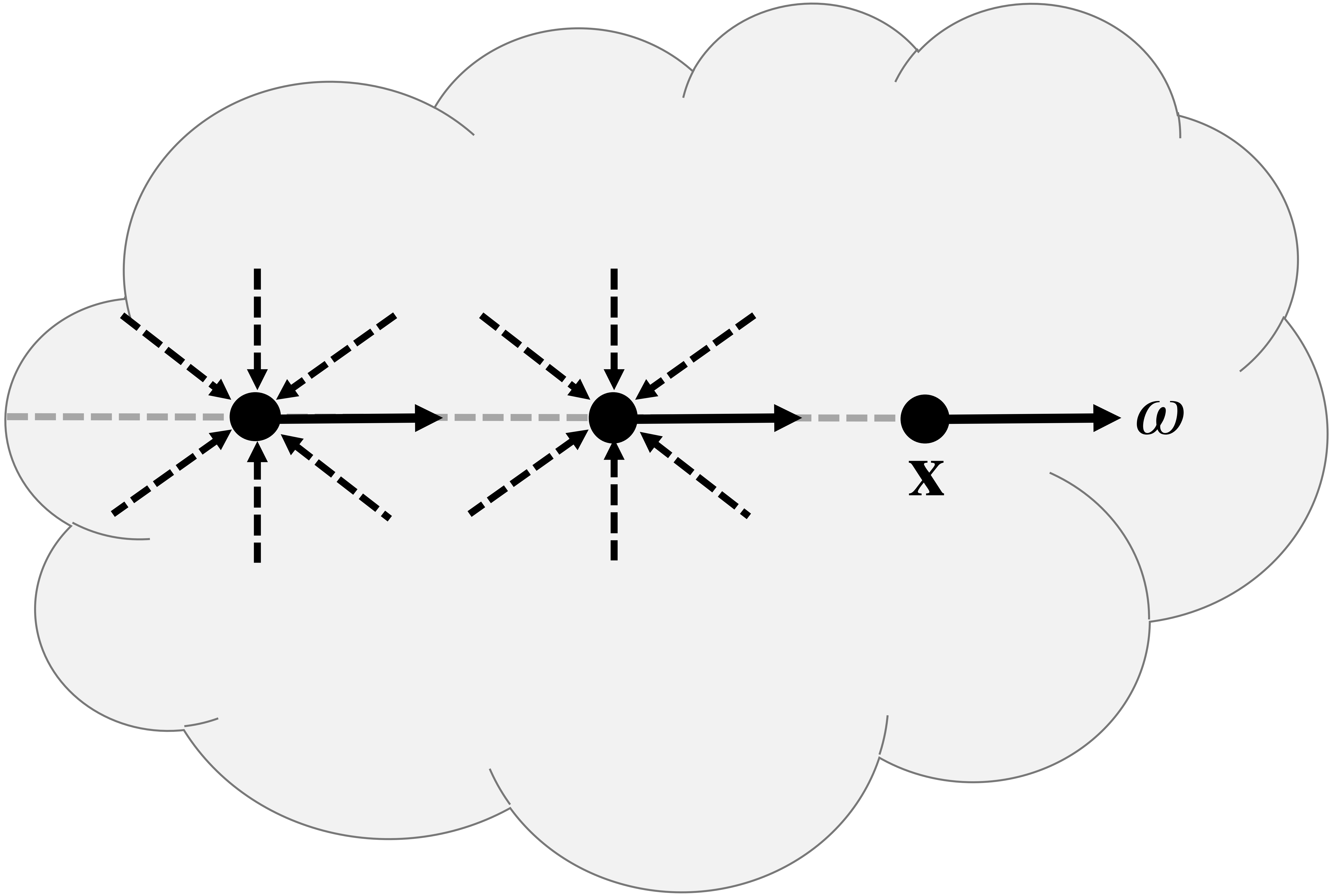
Mathematically, it is convenient to be capable of evaluating not only a given function but also its various transformations. One such

A Differential Theory of Radiative Transfer

Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, Shuang Zhao

SIGGRAPH Asia 2019

RECAP: RADIATIVE TRANSFER THEORY



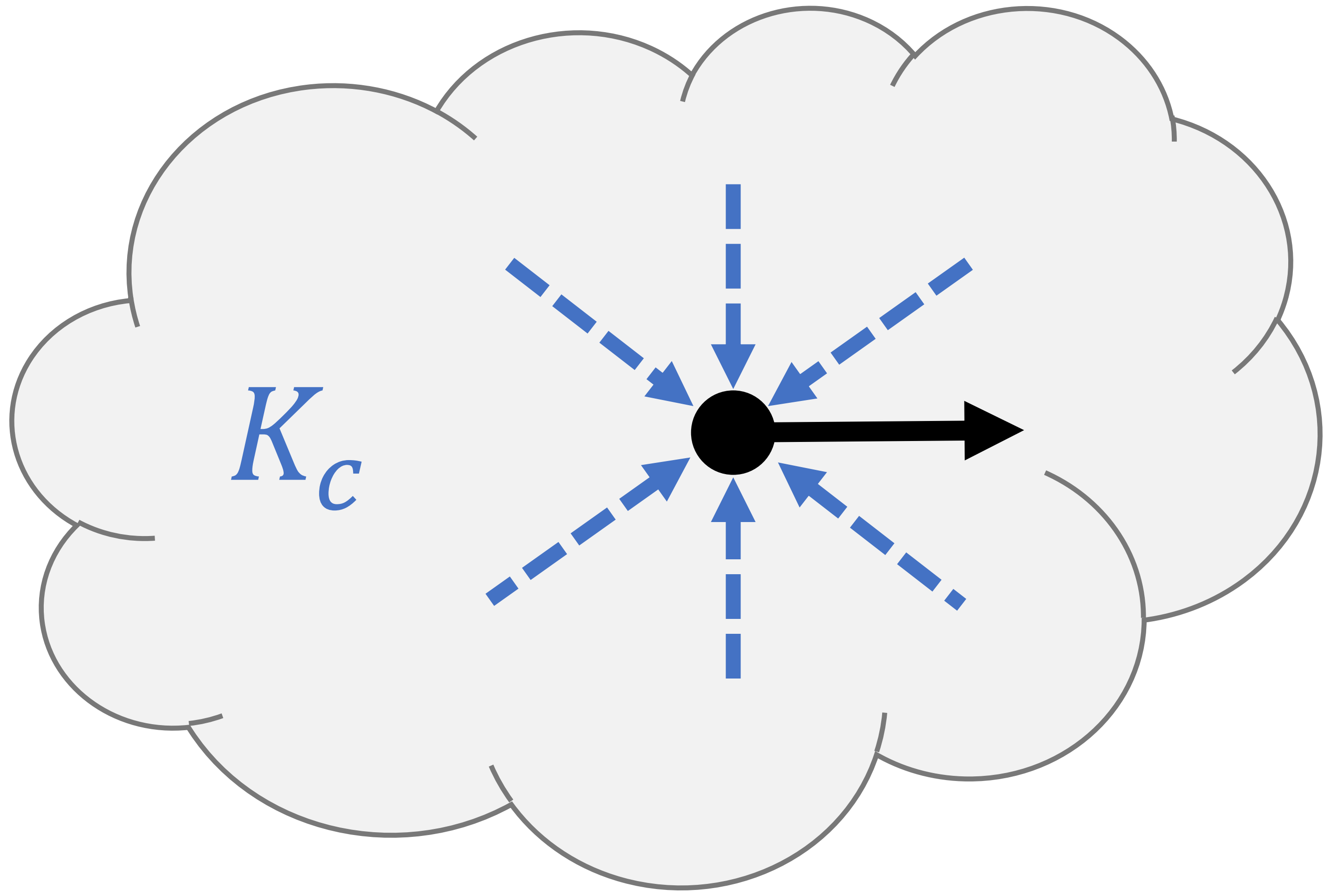
RECAP: RADIATIVE TRANSFER THEORY

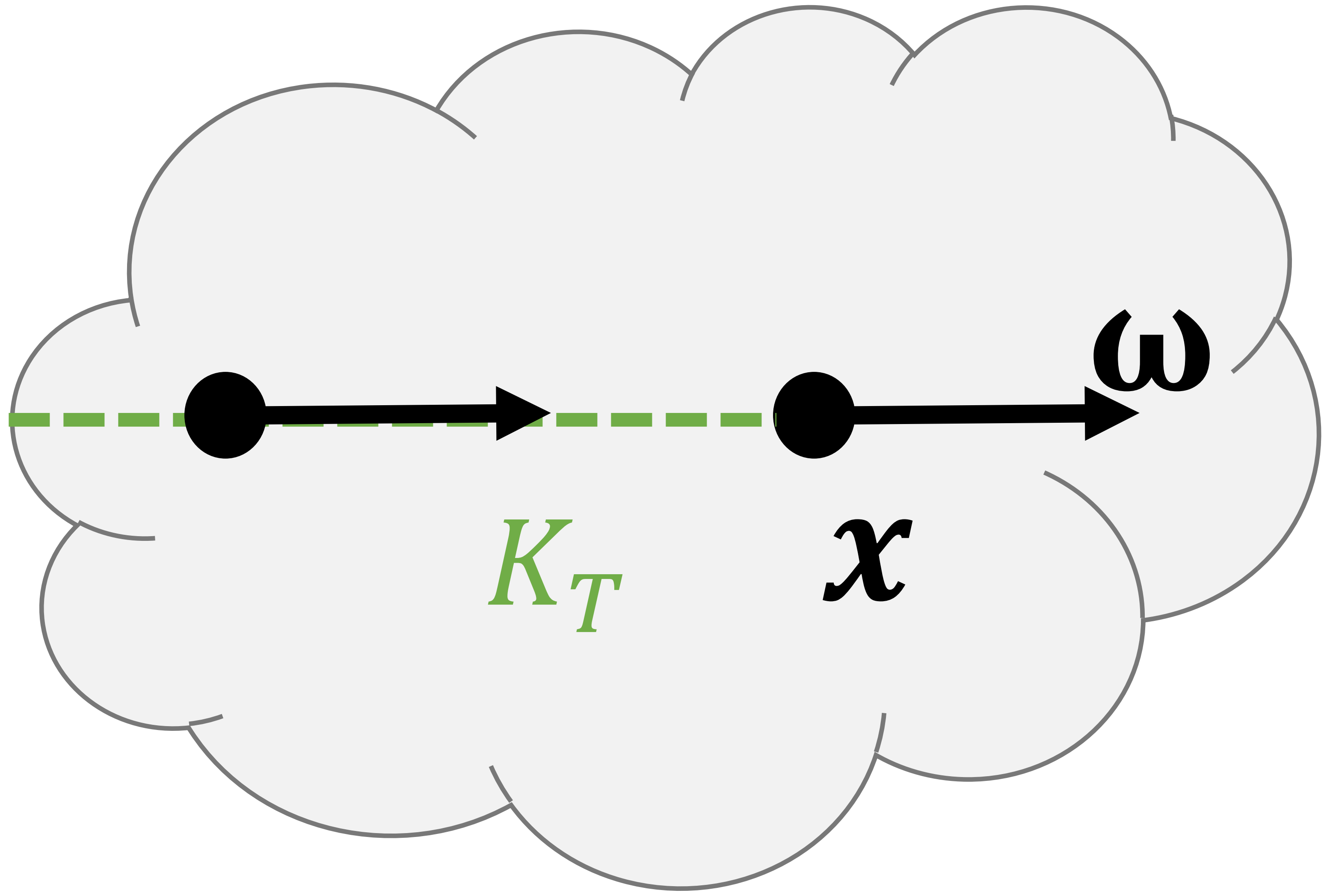
Transport operator · Collision operator · Source

$$L = K_T K_c L + Q$$

Radiative transfer equation (RTE)

in operator form





DIFFERENTIATING THE RTE

$$L = K_T K_c L + Q$$

$$\partial_\pi L = \partial_\pi (K_T K_c L) + \partial_\pi Q$$

Differentiating individual operators

DIFFERENTIATING THE COLLISION OPERATOR

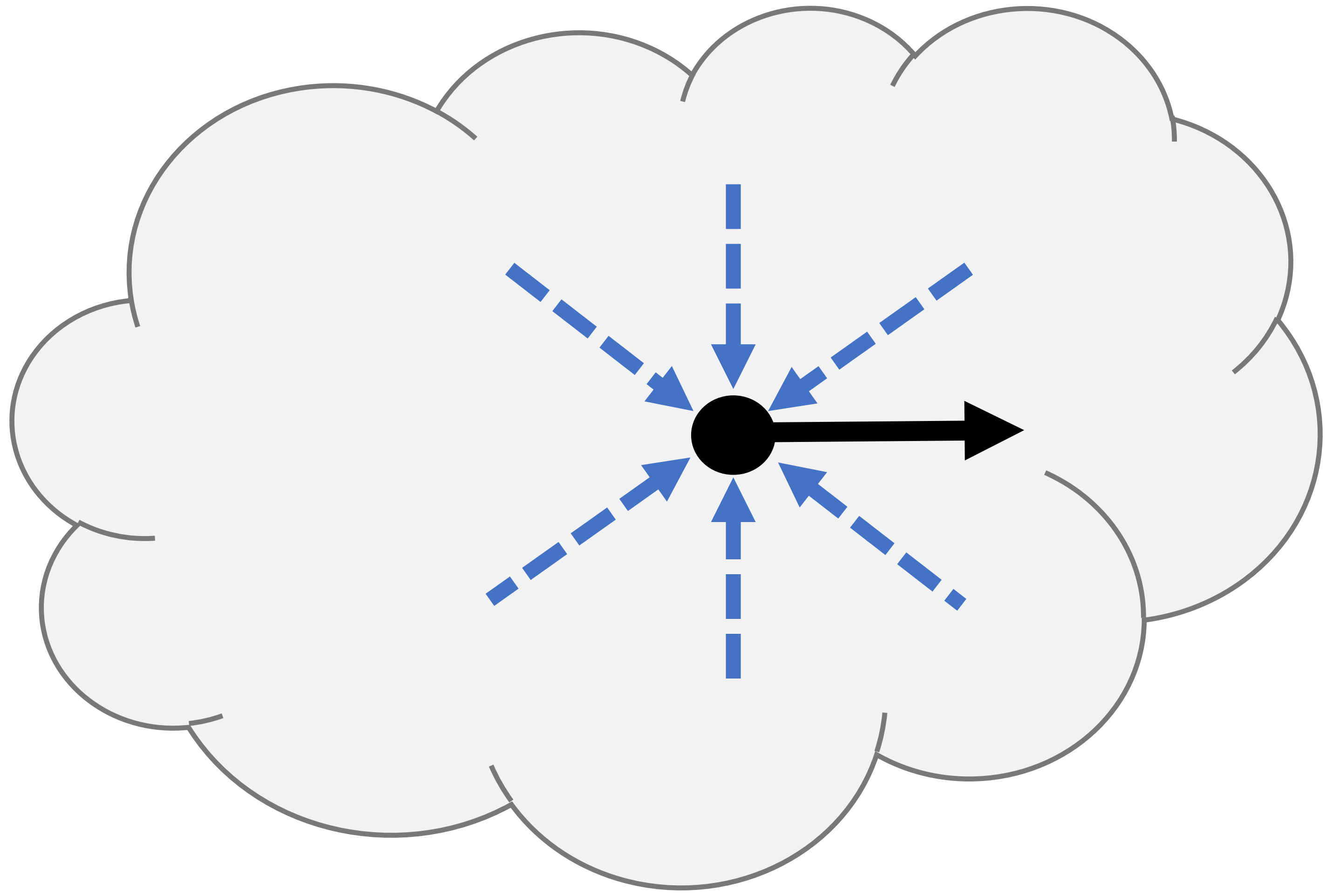
RTE: $L = K_T K_c L + Q$

(x omitted for notational simplicity)

$$(KcL)(\boldsymbol{\omega}) = \sigma_s \int_{\mathbb{S}^2} \overbrace{f_p(\boldsymbol{\omega}_i, \boldsymbol{\omega})\, L(\boldsymbol{\omega}_i)}^{f(\boldsymbol{\omega}_i)} \mathrm{d}\boldsymbol{\omega}_i$$

Scattering coefficient

Phase function



$$\partial_\pi \int_{\mathbb{S}^2} f(\boldsymbol{\omega}_i) \mathrm{d}\boldsymbol{\omega}_i = \textbf{?}$$

Requires differentiating a spherical integral

DIFFERENTIATING THE COLLISION OPERATOR

$$(KcL)(\boldsymbol{\omega}) = \sigma_s \int_{\mathbb{S}^2} \overbrace{f_p(\boldsymbol{\omega_i}, \boldsymbol{\omega})\, L(\boldsymbol{\omega_i})}^{f(\boldsymbol{\omega_i})} \mathrm{d}\boldsymbol{\omega_i}$$

$$\partial_\pi \int_{\mathbb{S}^2} f(\boldsymbol{\omega_i})\mathrm{d}\boldsymbol{\omega_i} = \underbrace{\int_{\mathbb{S}^2} \partial_\pi f(\boldsymbol{\omega_i})\mathrm{d}\boldsymbol{\omega_i}}_{\text{Interior integral}} + \underbrace{\int_{\partial\mathbb{S}^2} \left\langle \boldsymbol{n}, \frac{\partial \boldsymbol{\omega_i}}{\partial \pi} \right\rangle \Delta f(\boldsymbol{\omega_i})\mathrm{d}\boldsymbol{\omega_i}}_{\text{Boundary integral}}$$

By applying Reynolds transport theorem

(largely identical to the differentiation of the rendering equation)

OTHER TERMS IN THE RTE

$$L = K_T K_c L + Q$$

Transport operator (can be differentiated using Leibniz's rule)

$$(K_T K_c L)(x, \omega) = \int_0^D T(x', x)\,(K_c L)(x', \omega)\,d\tau$$

Transmittance

Source

$$Q = T(\boldsymbol{x_0}, \boldsymbol{x}) L_s(\boldsymbol{x_0}, \boldsymbol{\omega})$$

$$
\begin{aligned}
\dot{L}(\boldsymbol{x}, \boldsymbol{\omega}) = \int_0^D T(\boldsymbol{x}', \boldsymbol{x}) \Big[& \sigma_s(\boldsymbol{x}')\, \dot{L}^{\mathrm{ins}}(\boldsymbol{x}', \boldsymbol{\omega}) + \big(\dot{\sigma}_s(\boldsymbol{x}') - \Sigma_t(\boldsymbol{x}, \boldsymbol{\omega}, \tau)\, \sigma_s(\boldsymbol{x}')\big)\, L^{\mathrm{ins}}(\boldsymbol{x}', \boldsymbol{\omega}) \Big]\, \mathrm{d}\tau \\
+ T(\boldsymbol{x}_0, \boldsymbol{x}) \Big[& -\big(\Sigma_t(\boldsymbol{x}, \boldsymbol{\omega}, D) + \dot{D}\, \sigma_t(\boldsymbol{x}_0)\big)\, L_s(\boldsymbol{x}_0, \boldsymbol{\omega}) + \dot{L}_s(\boldsymbol{x}_0, \boldsymbol{\omega}) + \dot{D}\, \sigma_s(\boldsymbol{x}_0)\, L^{\mathrm{ins}}(\boldsymbol{x}_0, \boldsymbol{\omega}) \Big],
\end{aligned}
$$

where Σ_t is defined in Eq. (17), $\dot{L}^{\mathrm{ins}}$ follows Eq. (22), and $\dot{L}_s = \dot{L}_s^{\mathrm{r}} + \dot{L}_s^{\mathrm{e}}$ with $\dot{L}_s^{\mathrm{r}}$ given by Eq. (29)

This is Eq. (32) of the work by Zhang et al. [2019]

DIFFERENTIAL RTE, OPERATOR FORM

$$L = K_T K_c L + Q$$

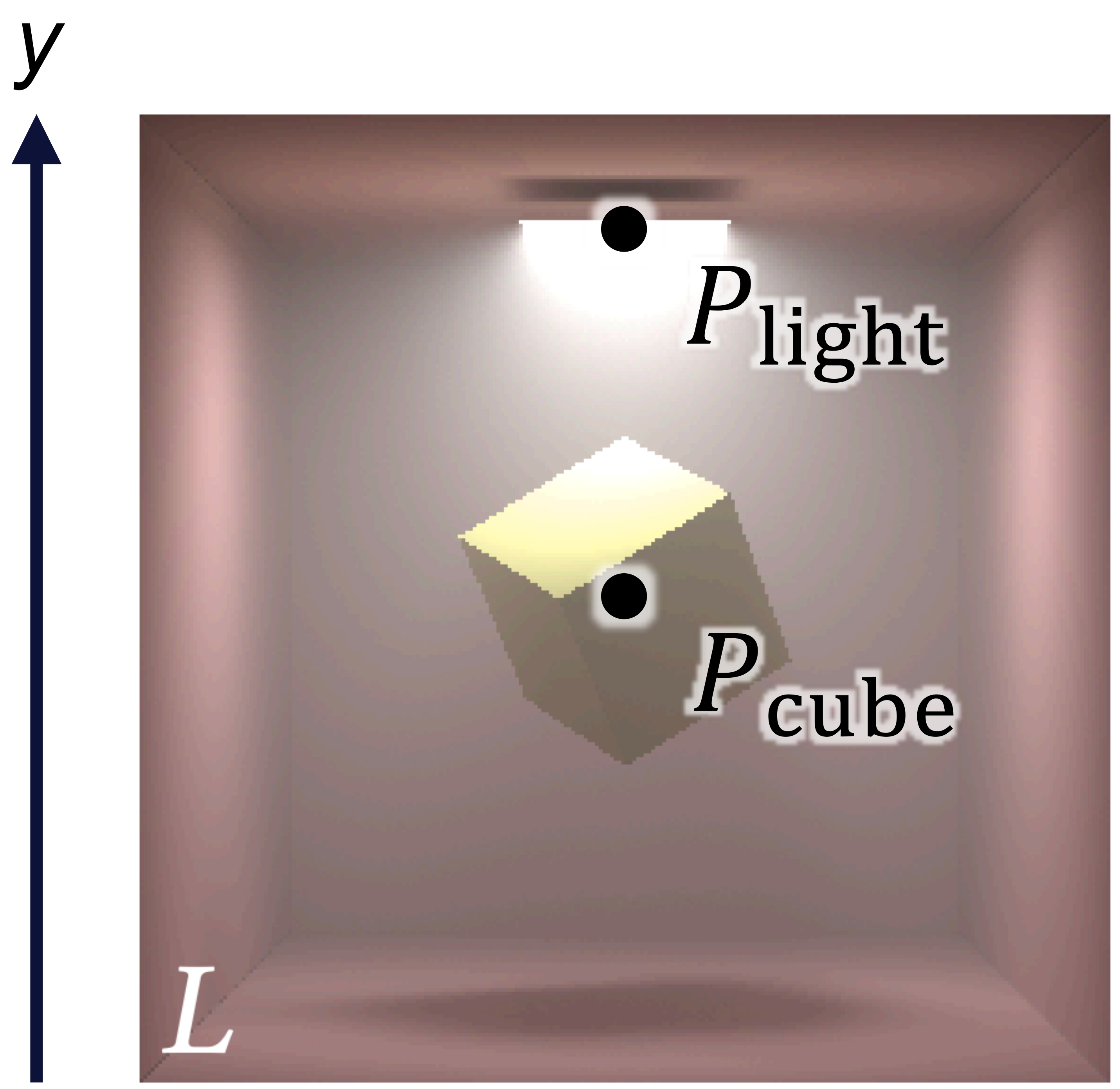
$$\partial_\pi L = \partial_\pi (K_T K_c L) + \partial_\pi Q$$

Captures the boundary integrals

$$\begin{pmatrix} \partial_\pi L \\ L \end{pmatrix} = \begin{pmatrix} K_T K_c & K_* \\ 0 & K_T K_c \end{pmatrix} \begin{pmatrix} \partial_\pi L \\ L \end{pmatrix} + \begin{pmatrix} \partial_\pi Q \\ Q \end{pmatrix}$$

Differential radiative transfer equation

SIGNIFICANCE OF THE BOUNDARY INTEGRAL



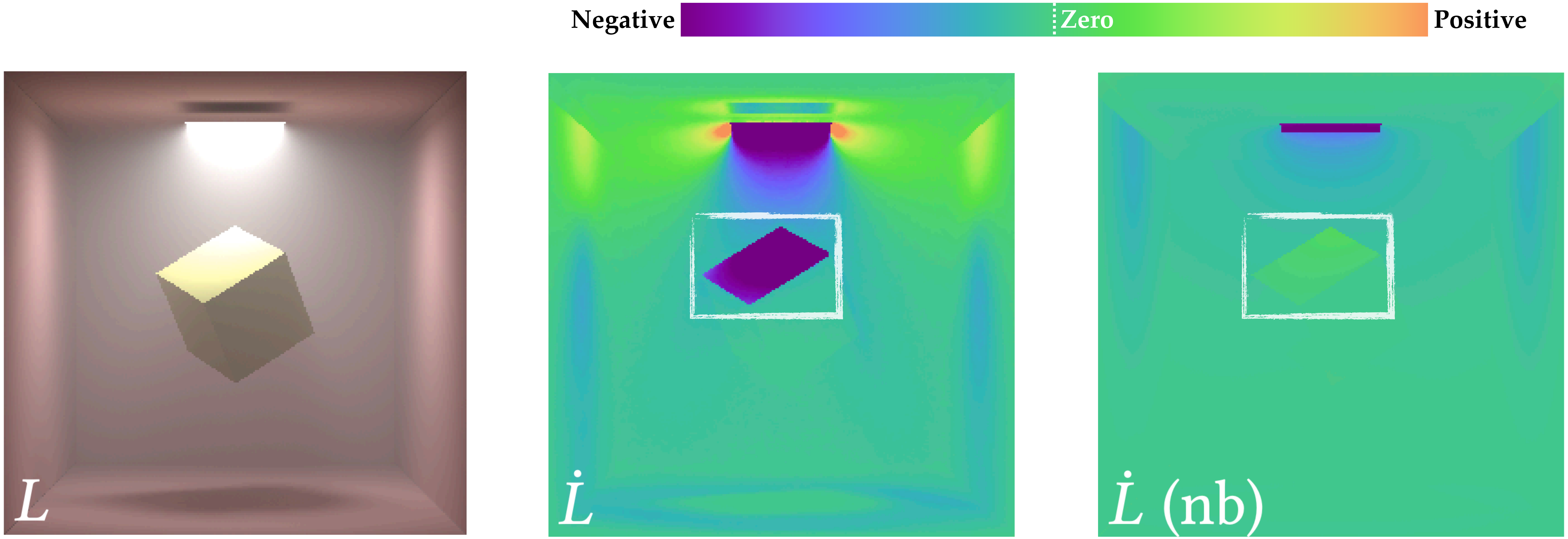
Original image

$$\mathbf{P}_{\text{light}}(\pi) = \boxed{\mathbf{P}_0} + \begin{pmatrix} 0 \\ \pi \\ 0 \end{pmatrix}$$

$$\mathbf{P}_{\text{cube}}(\pi) = \boxed{\mathbf{P}_1} + \begin{pmatrix} 0 \\ \pi \\ 0 \end{pmatrix}$$

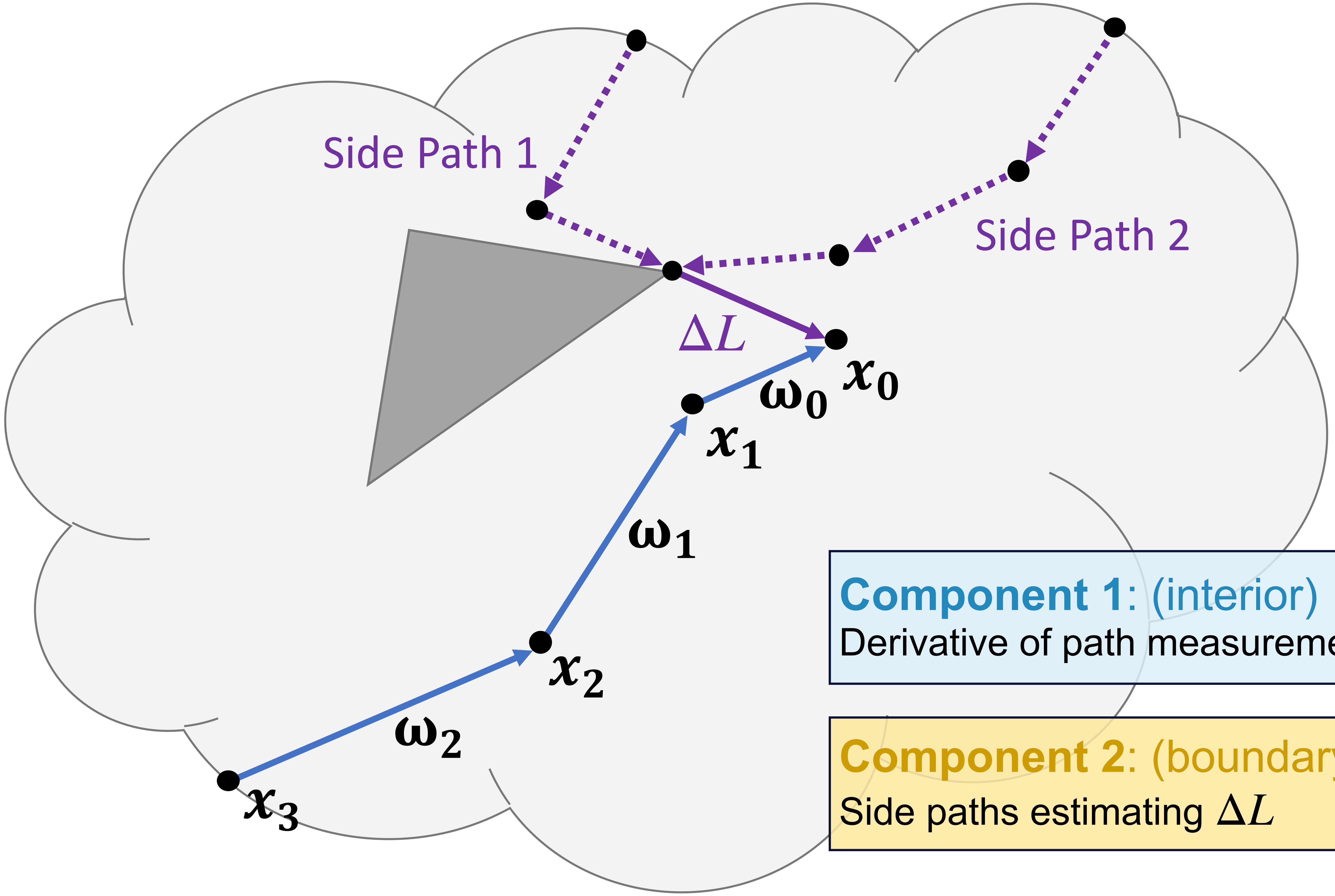
Constant
initial positions

SIGNIFICANCE OF THE BOUNDARY INTEGRAL



Original image | Derivative image | Derivative image (w/o boundary integral)

DIFFERENTIABLE VOLUMETRIC PATH TRACING



Component 1: (interior)
Derivative of path measurement contribution

Component 2: (boundary)
Side paths estimating ΔL

INVERSE-RENDERING RESULTS

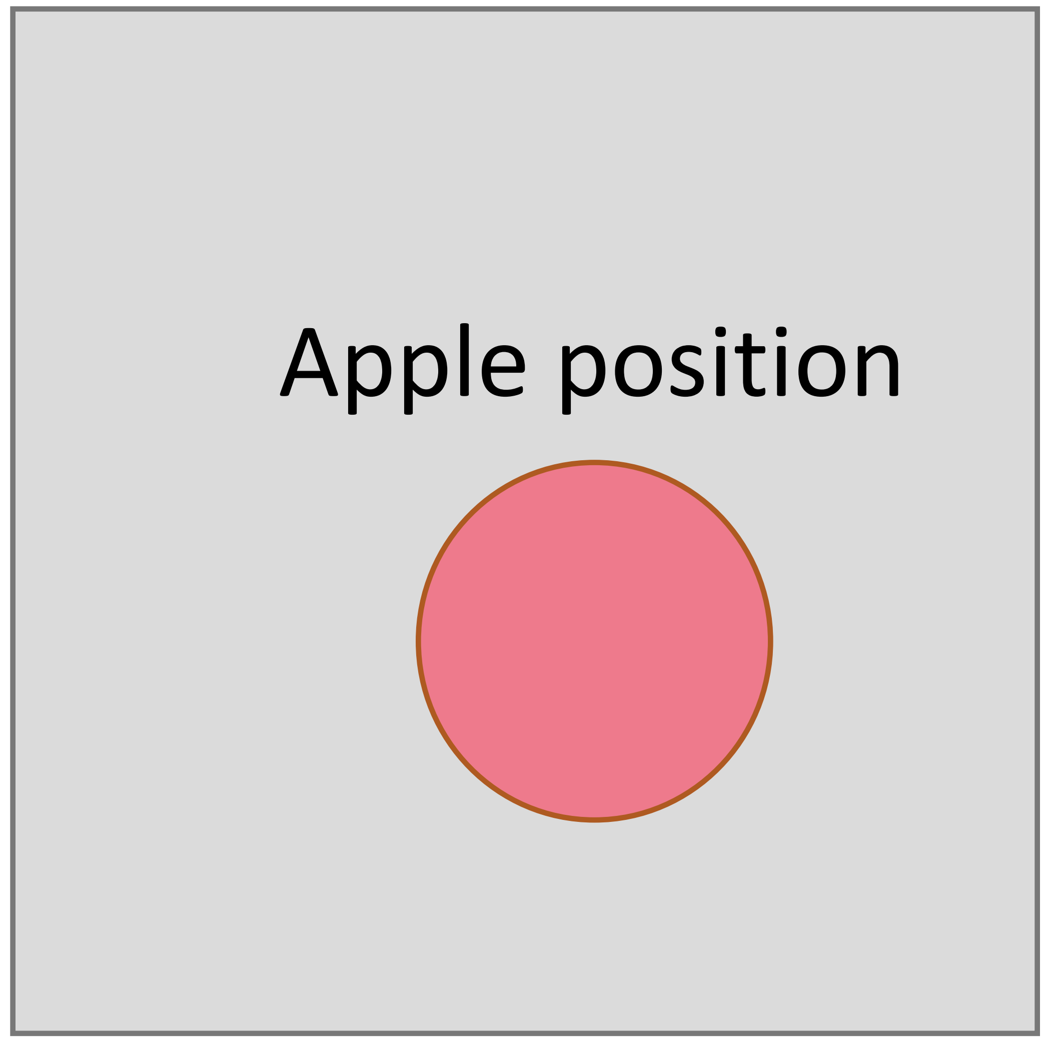
- Scene configurations
  - Participating media
  - Changing geometry

- Optimization
  - Using only image loss (L2)

INVERSE-RENDERING RESULTS

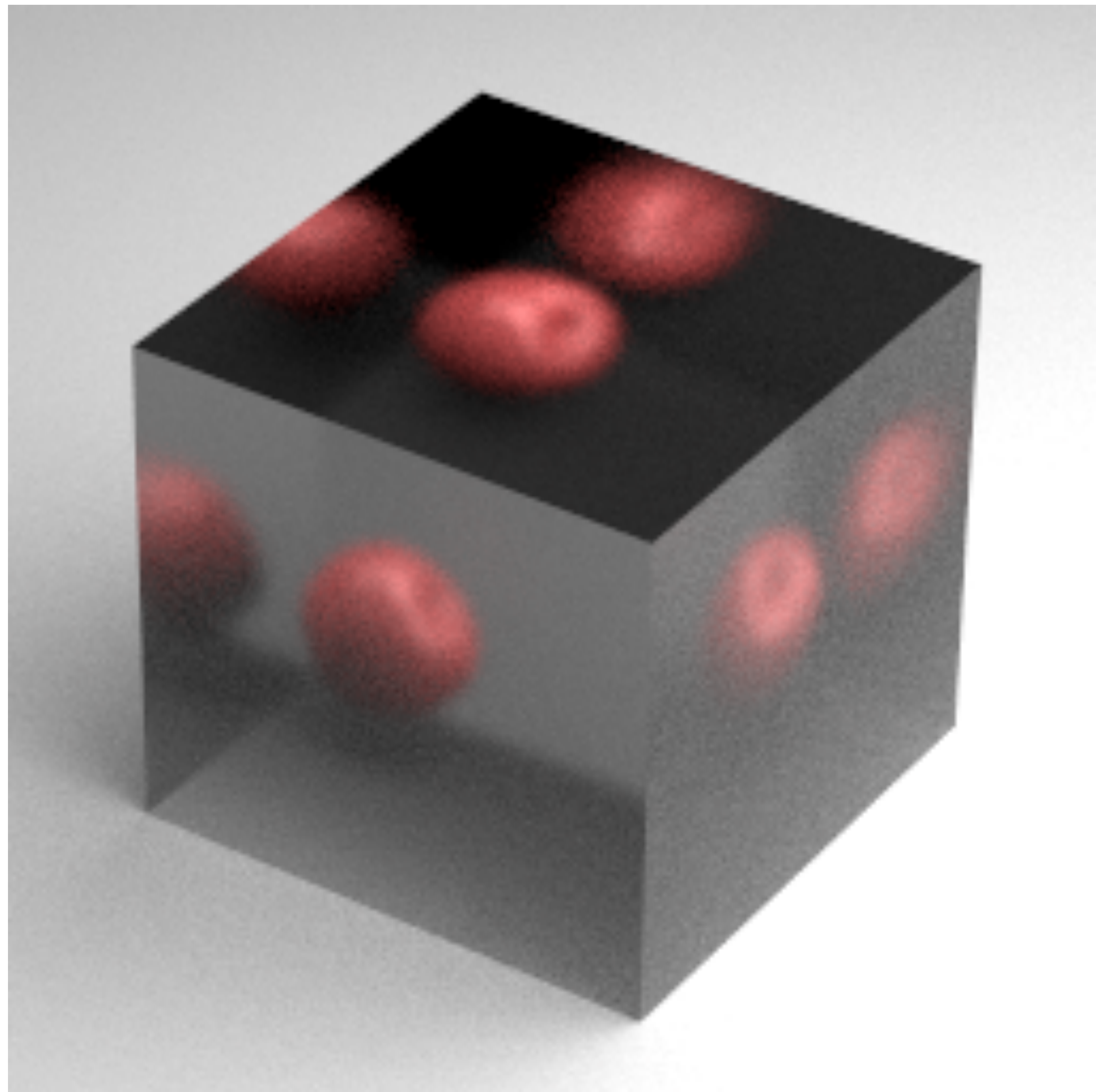
Apple in a box

Parameters

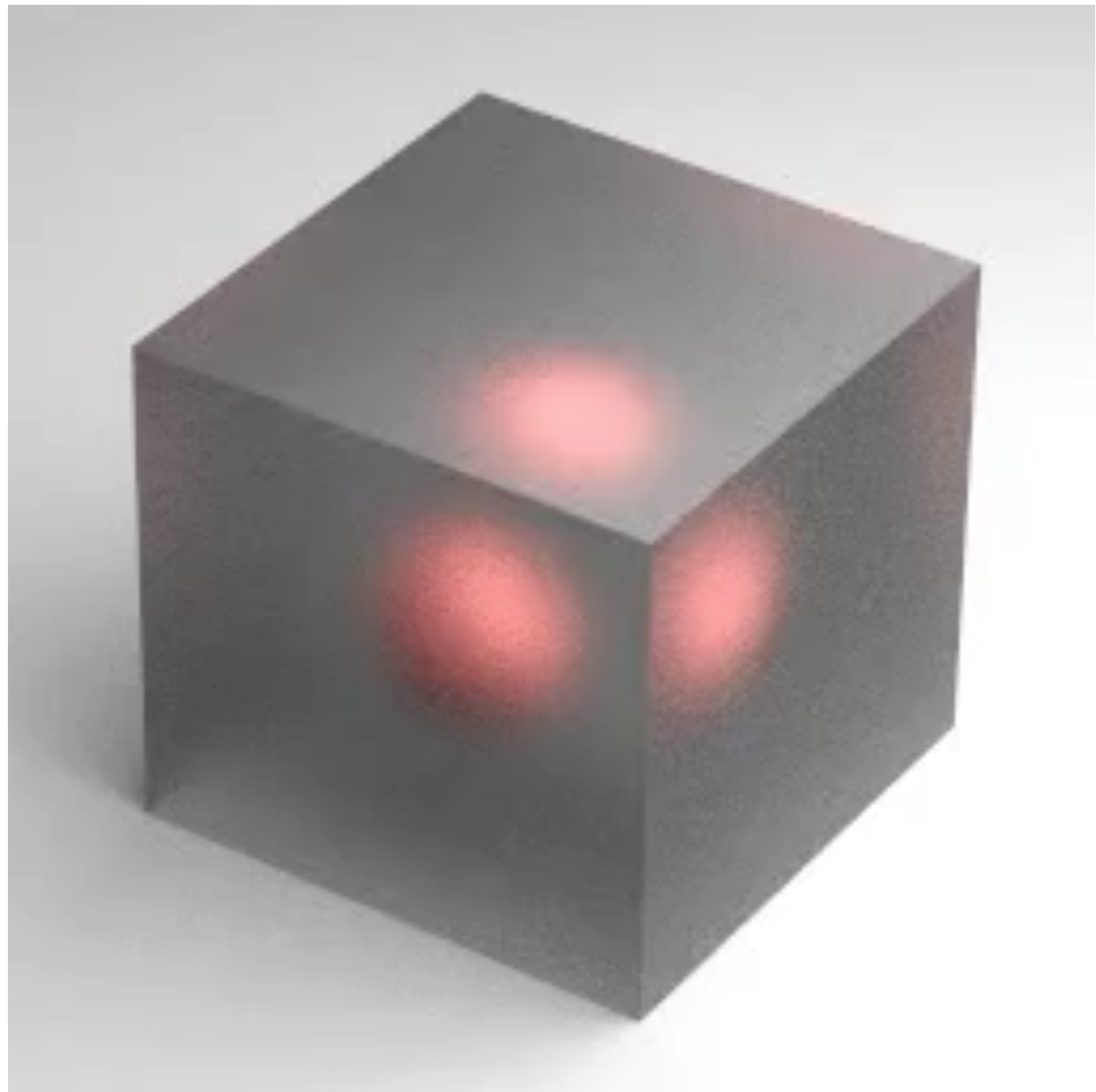


Cube roughness

Target

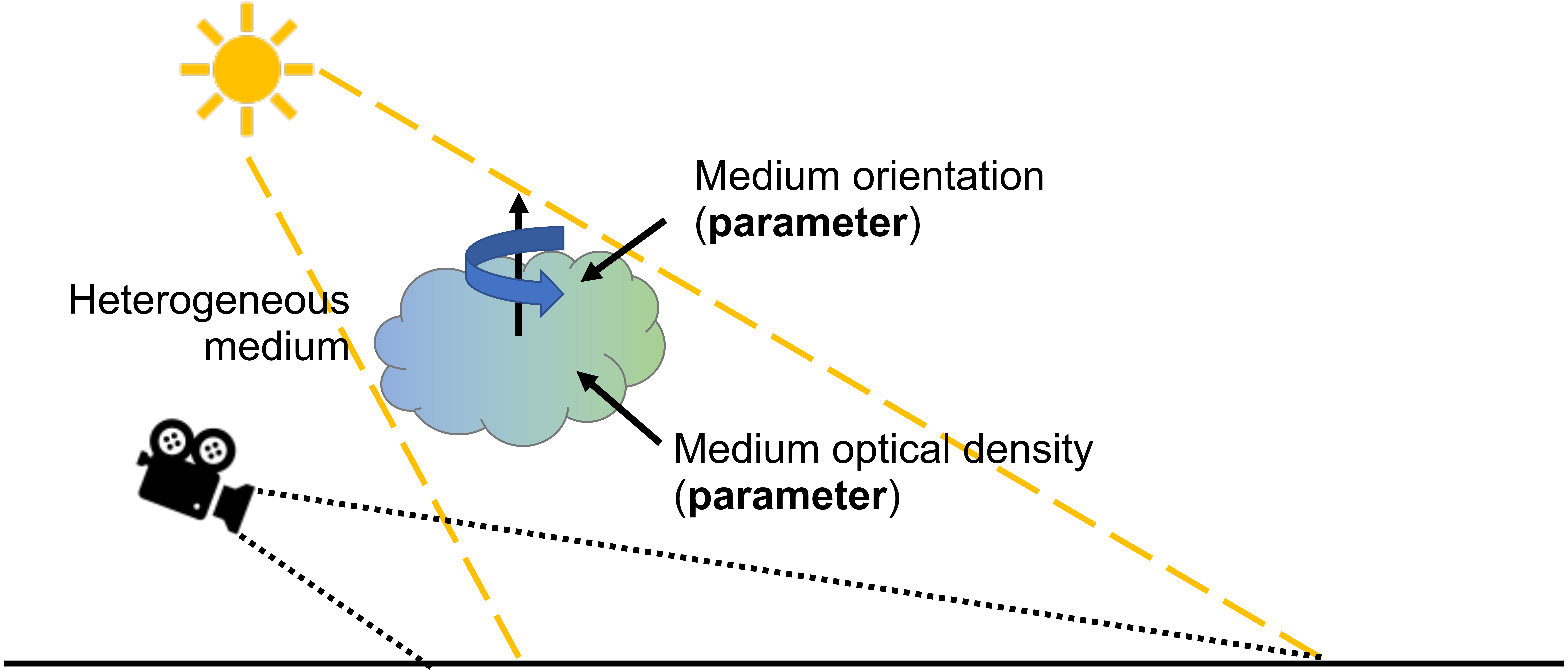


Optimization process



INVERSE-RENDERING RESULTS

Non-line-of-sight inverse rendering



INVERSE-RENDERING RESULTS

Non-line-of-sight inverse rendering

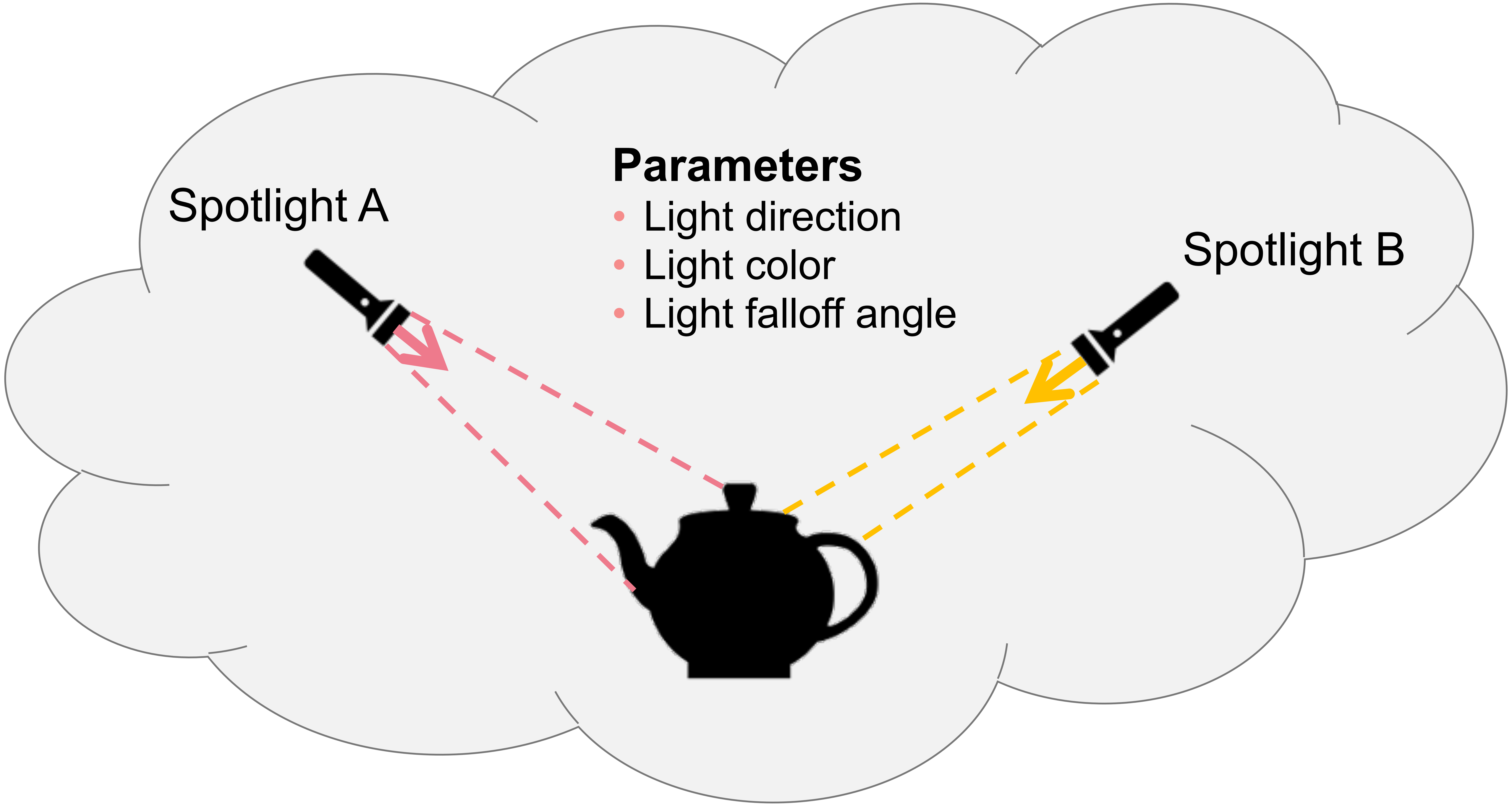
Target

Optimization process

Different view

INVERSE-RENDERING RESULTS

Design-inspired inverse rendering

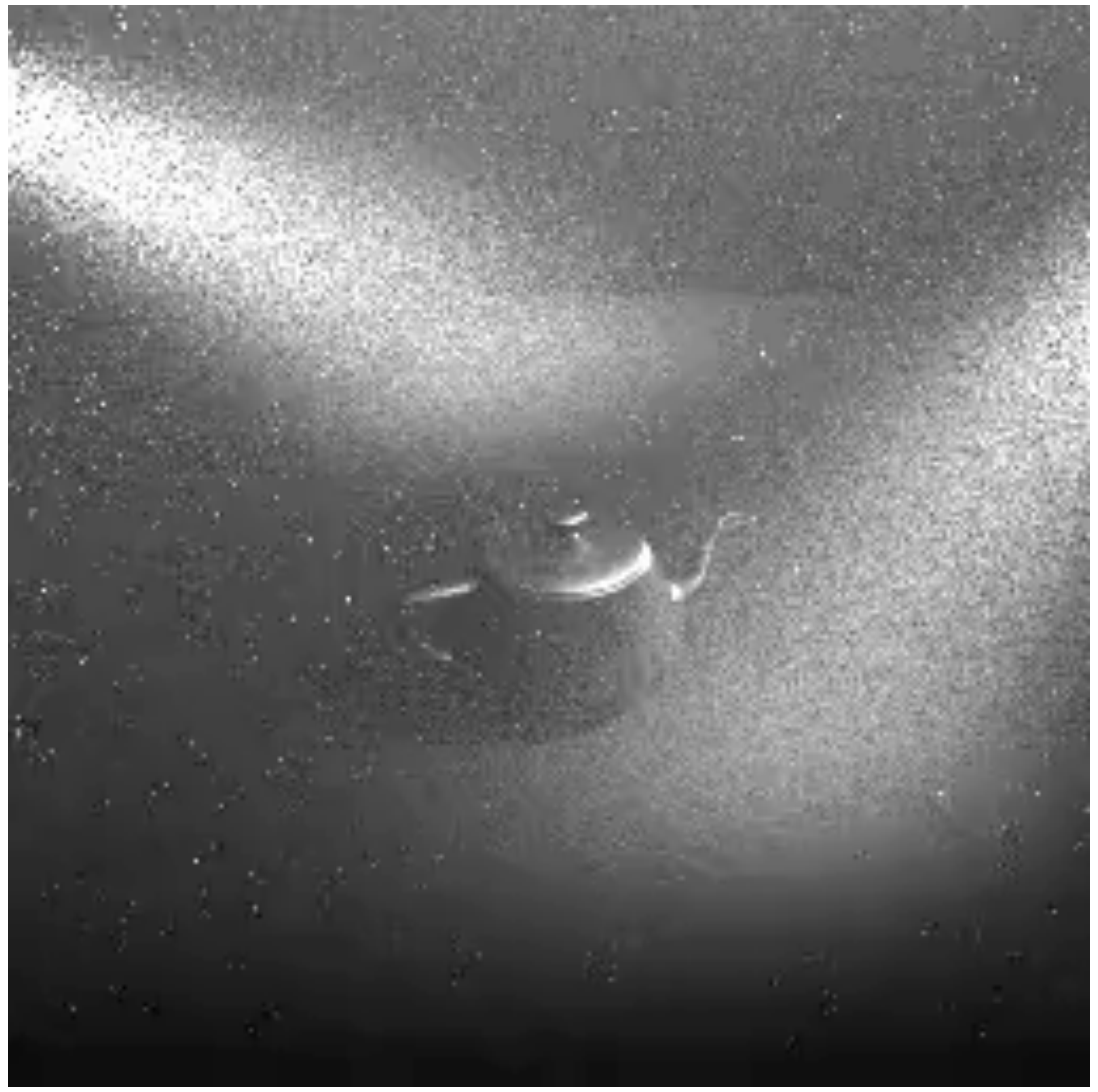


INVERSE-RENDERING RESULTS

Design-inspired inverse rendering

Target

Optimization process



CHALLENGES

Rendering equation

$$L(\boldsymbol{\omega}_\mathrm{o}) = \int_{\mathbb{S}^2} \overbrace{L_\mathrm{i}(\boldsymbol{\omega}_\mathrm{i}) f_s(\boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o})}^{f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i})} \, \mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i}) + L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

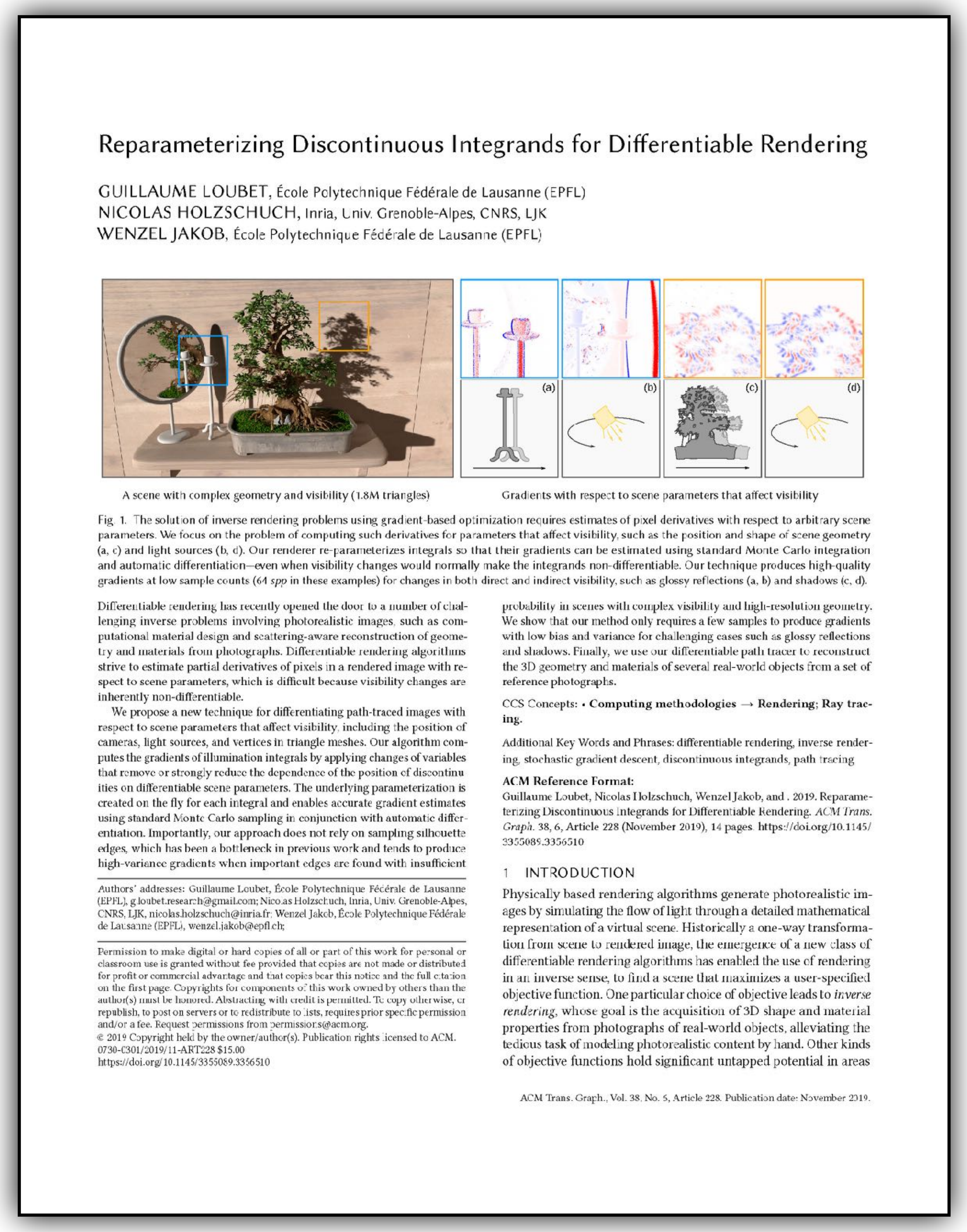
Differential rendering equation

$$\frac{\mathrm{d}}{\mathrm{d}\pi} L(\boldsymbol{\omega}_\mathrm{o}) = \underbrace{\int_{\mathbb{S}^2} \frac{\mathrm{d}}{\mathrm{d}\pi} f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i}) \, \mathrm{d}\sigma(\boldsymbol{\omega}_\mathrm{i})}_{\text{Interior integral}} + \underbrace{\int_{\partial\mathbb{S}^2} V_{\partial\mathbb{S}^2}(\boldsymbol{\omega}_\mathrm{i}) \, \Delta f_\mathrm{RE}(\boldsymbol{\omega}_\mathrm{i}) \, \mathrm{d}\ell(\boldsymbol{\omega}_\mathrm{i})}_{\text{Boundary integral}} + \frac{\mathrm{d}}{\mathrm{d}\pi} L_\mathrm{e}(\boldsymbol{\omega}_\mathrm{o})$$

- Complex scenes
  - Discontinuity points (e.g., $\partial\mathbb{S}^2$) can be expensive to detect

- Scaling out to millions of parameters

ANOTHER WAY OF DEALING WITH EDGES



Reparameterizing Discontinuous Integrands for Differentiable Rendering

GUILLAUME LOUBET, École Polytechnique Fédérale de Lausanne (EPFL)
NICOLAS HOLZSCHUCH, Inria, Univ. Grenoble-Alpes, CNRS, LJK
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL)

(a) (b) (c) (d)

A scene with complex geometry and visibility (1.8M triangles)

Gradients with respect to scene parameters that affect visibility

Fig. 1. The solution of inverse rendering problems using gradient-based optimization requires estimates of pixel derivatives with respect to arbitrary scene parameters. We focus on the problem of computing such derivatives for parameters that affect visibility, such as the position and shape of scene geometry (a, c) and light sources (b, d). Our renderer re-parameterizes integrals so that their gradients can be estimated using standard Monte Carlo integration and automatic differentiation—even when visibility changes would normally make the integrands non-differentiable. Our technique produces high-quality gradients at low sample counts (64 *spp* in these examples) for changes in both direct and indirect visibility, such as glossy reflections (a, b) and shadows (c, d).

Differentiable rendering has recently opened the door to a number of challenging inverse problems involving photorealistic images, such as computational material design and scattering-aware reconstruction of geometry and materials from photographs. Differentiable rendering algorithms strive to estimate partial derivatives of pixels in a rendered image with respect to scene parameters, which is difficult because visibility changes are inherently non-differentiable.

We propose a new technique for differentiating path-traced images with respect to scene parameters that affect visibility, including the position of cameras, light sources, and vertices in triangle meshes. Our algorithm computes the gradients of illumination integrals by applying changes of variables that remove or strongly reduce the dependence of the position of discontinuities on differentiable scene parameters. The underlying parameterization is created on the fly for each integral and enables accurate gradient estimates using standard Monte Carlo sampling in conjunction with automatic differentiation. Importantly, our approach does not rely on sampling silhouette edges, which has been a bottleneck in previous work and tends to produce high-variance gradients when important edges are found with insufficient probability in scenes with complex visibility and high-resolution geometry. We show that our method only requires a few samples to produce gradients with low bias and variance for challenging cases such as glossy reflections and shadows. Finally, we use our differentiable path tracer to reconstruct the 3D geometry and materials of several real-world objects from a set of reference photographs.

CCS Concepts: • **Computing methodologies → Rendering; Ray tracing.**

Additional Key Words and Phrases: differentiable rendering, inverse rendering, stochastic gradient descent, discontinuous integrands, path tracing

ACM Reference Format:
Guillaume Loubet, Nicolas Holzschuch, Wenzel Jakob, and . 2019. Reparameterizing Discontinuous Integrands for Differentiable Rendering. *ACM Trans. Graph.* 38, 6, Article 228 (November 2019), 14 pages. https://doi.org/10.1145/3355089.3356510

Authors' addresses: Guillaume Loubet, École Polytechnique Fédérale de Lausanne (EPFL), g.loubet.research@gmail.com; Nicolas Holzschuch, Inria, Univ. Grenoble-Alpes, CNRS, LJK, nicolas.holzschuch@inria.fr; Wenzel Jakob, École Polytechnique Fédérale de Lausanne (EPFL), wenzel.jakob@epfl.ch;

Permission to make digital or hard copies of all or part 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 components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0730-0301/2019/11-ART228 $15.00
https://doi.org/10.1145/3355089.3356510

1 INTRODUCTION

Physically based rendering algorithms generate photorealistic images by simulating the flow of light through a detailed mathematical representation of a virtual scene. Historically a one-way transformation from scene to rendered image, the emergence of a new class of differentiable rendering algorithms has enabled the use of rendering in an inverse sense, to find a scene that maximizes a user-specified objective function. One particular choice of objective leads to *inverse rendering*, whose goal is the acquisition of 3D shape and material properties from photographs of real-world objects, alleviating the tedious task of modeling photorealistic content by hand. Other kinds of objective functions hold significant untapped potential in areas

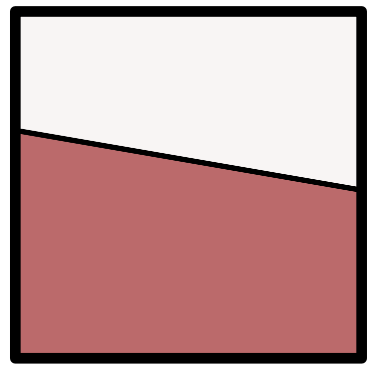
ACM Trans. Graph., Vol. 38, No. 6, Article 228. Publication date: November 2019.

Reparameterizing Discontinuous Integrals for Differentiable Rendering

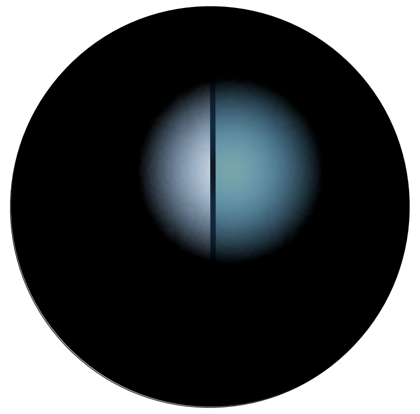
Guillaume Loubet, Nicolas Holzschuch, Wenzel Jakob

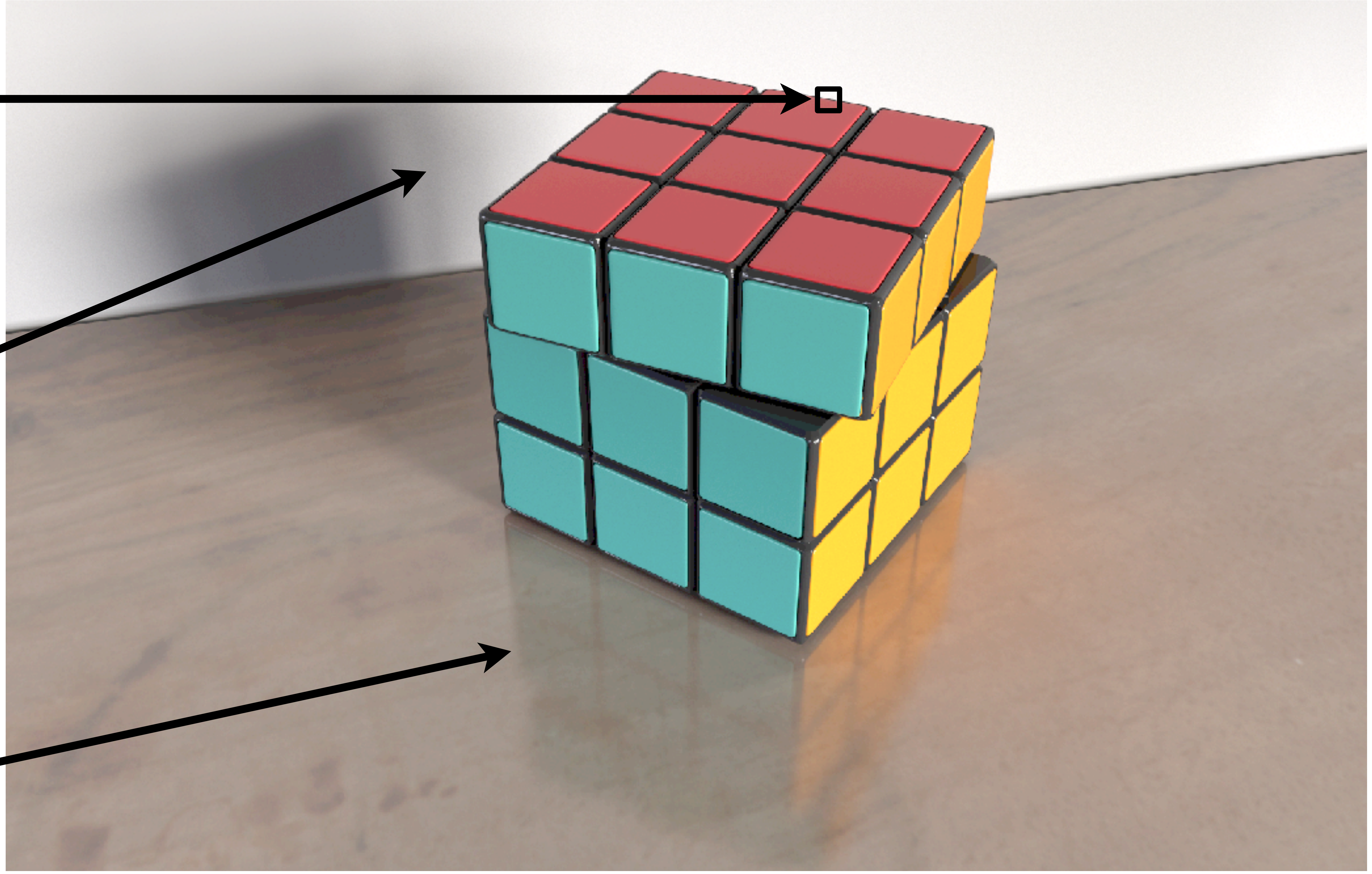
SIGGRAPH Asia 2019

MOVING DISCONTINUITIES

Pixel integrals $\iint$  $\mathrm{d}x\,\mathrm{d}y$

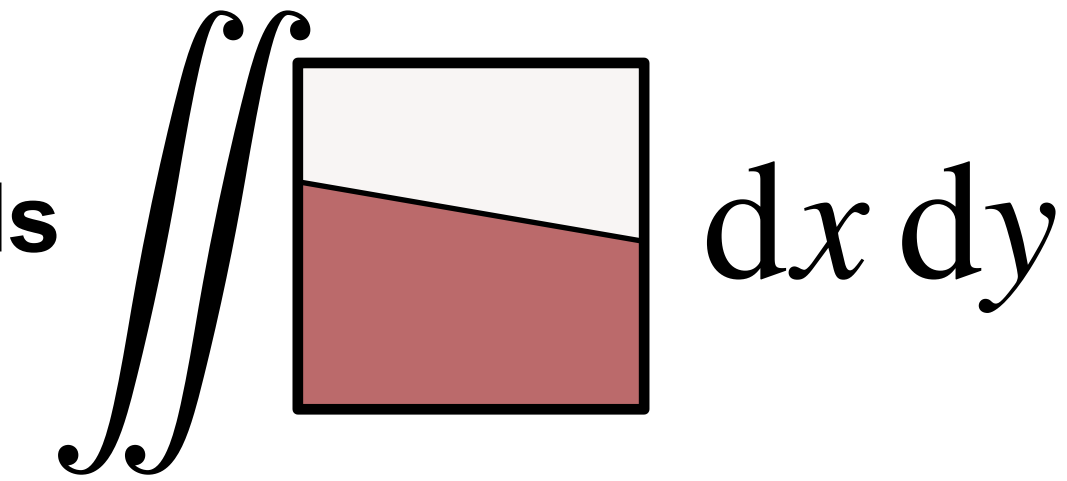
Light integrals $\int$  $\mathrm{d}\omega$

BSDF integrals $\int$  $\mathrm{d}\omega$

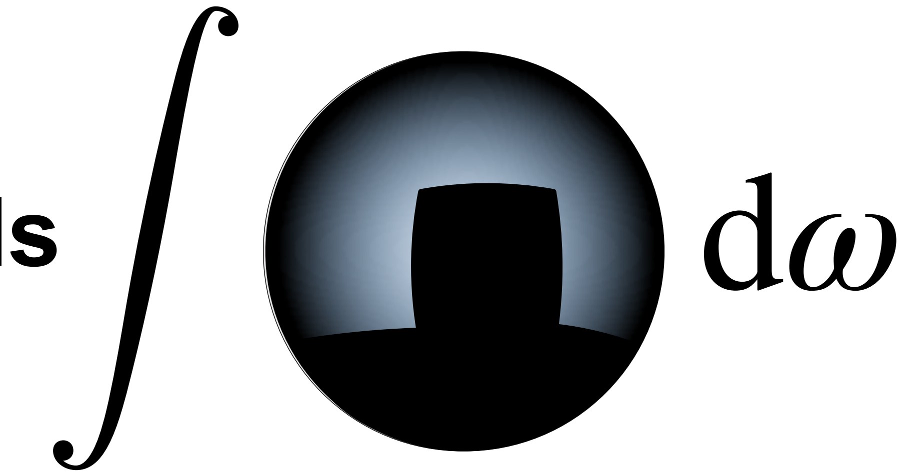


MOVING DISCONTINUITIES

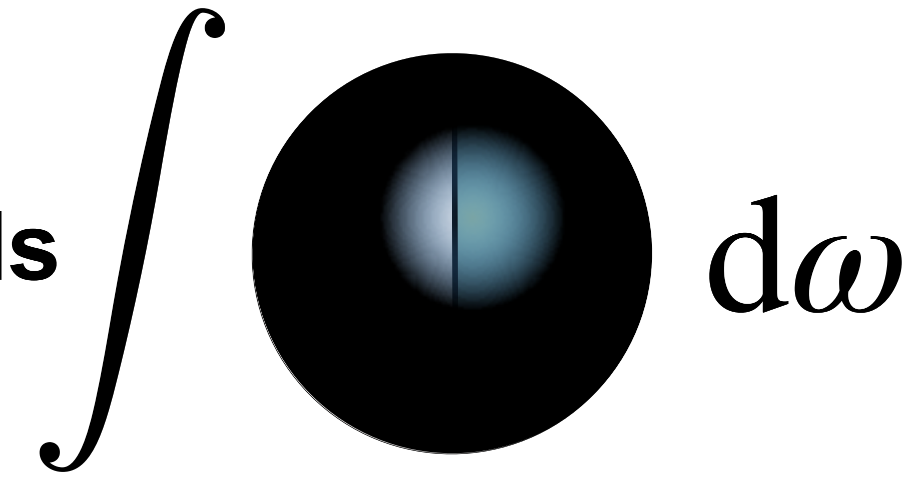
Pixel integrals

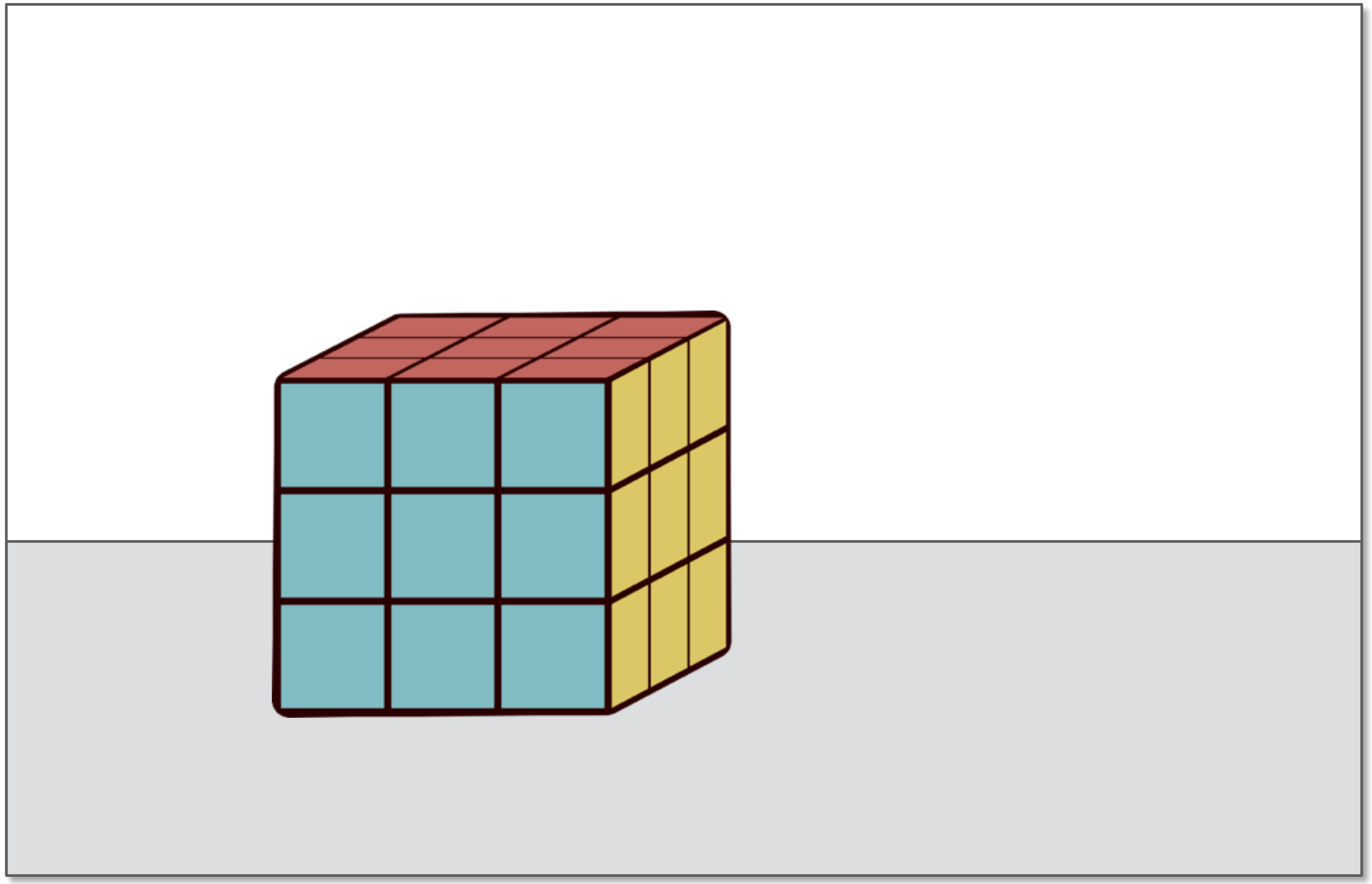


Light integrals



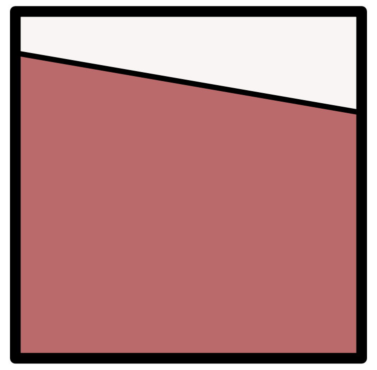
BSDF integrals



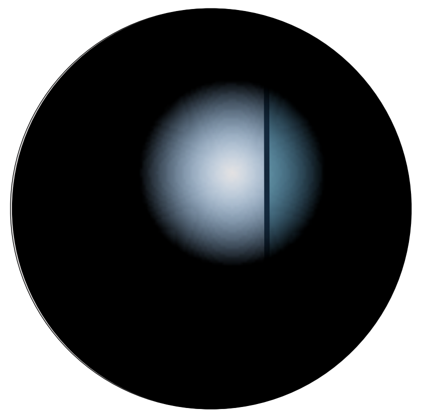


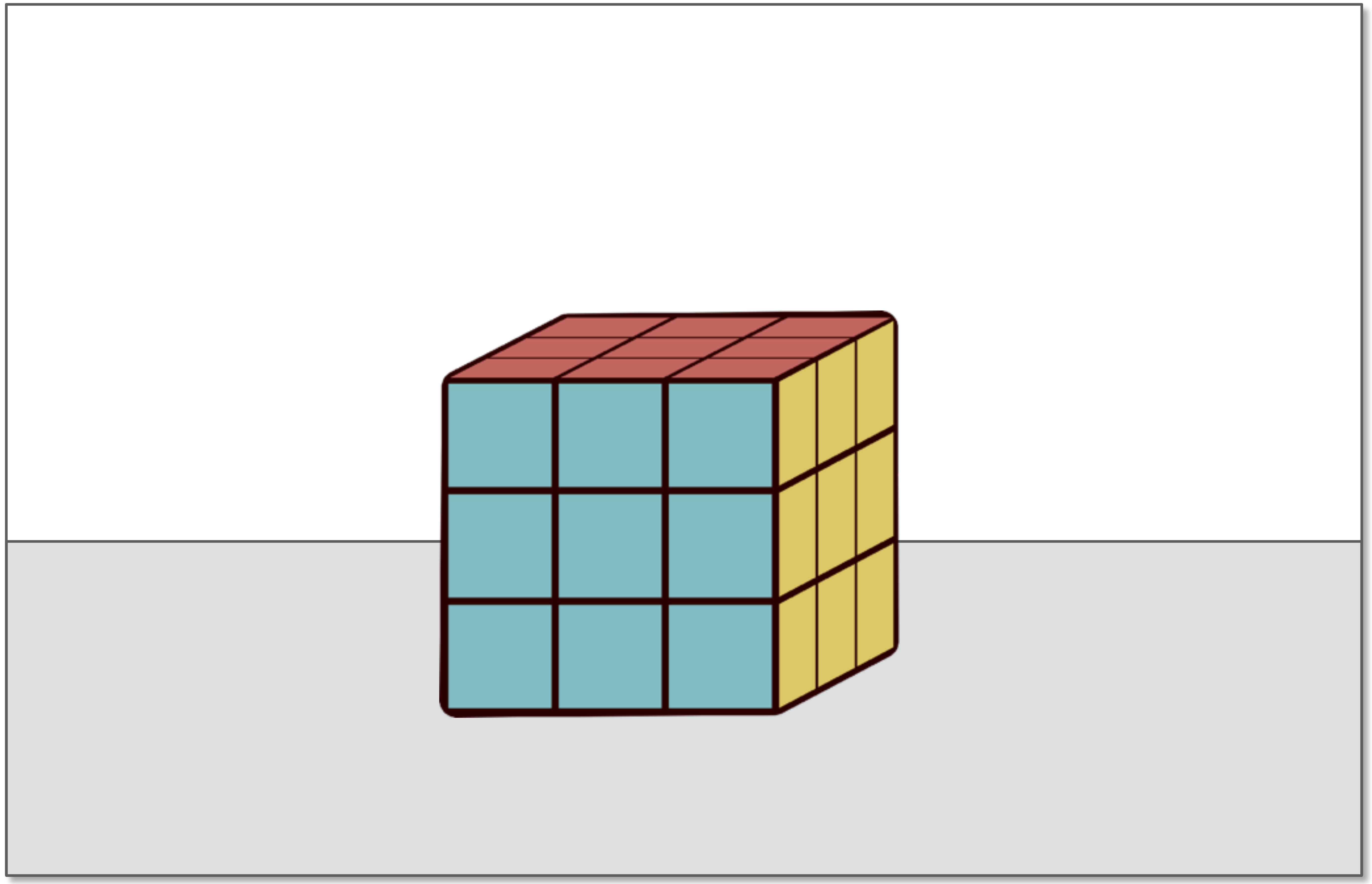
Scene parameter x_i

MOVING DISCONTINUITIES

Pixel integrals $\iint$  $\mathrm{d}x\,\mathrm{d}y$

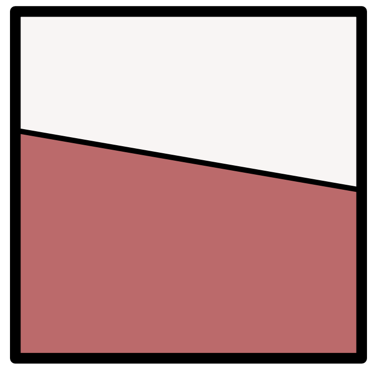
Light integrals $\int$  $\mathrm{d}\omega$

BSDF integrals $\int$  $\mathrm{d}\omega$

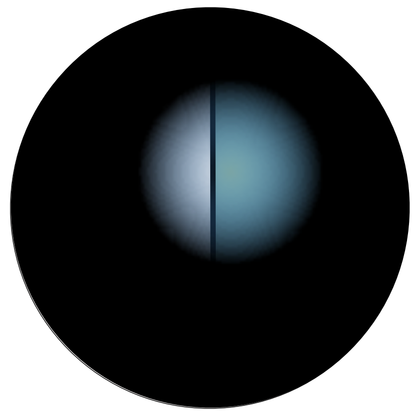


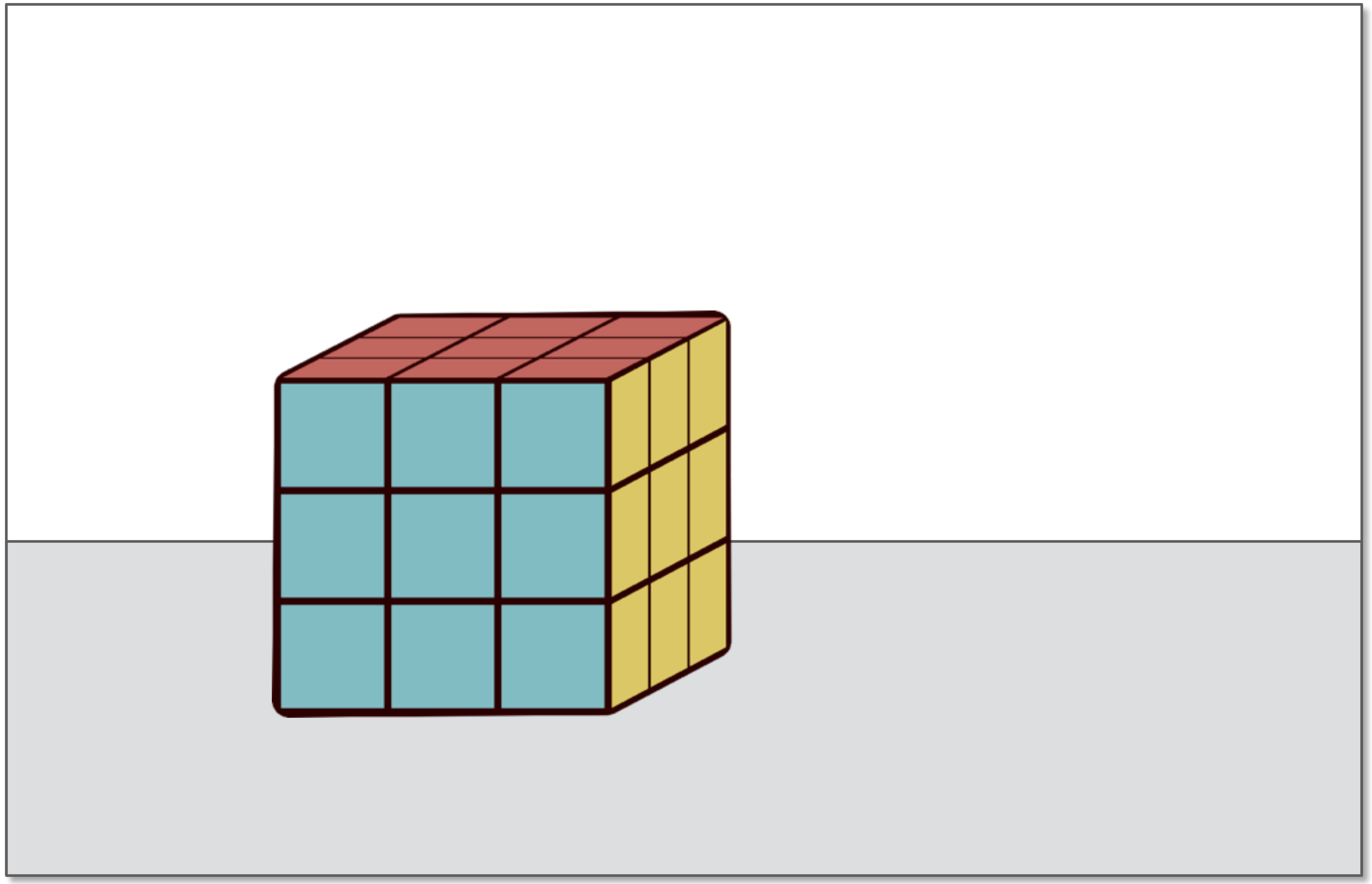
Scene parameter x_i 

MOVING DISCONTINUITIES

Pixel integrals $\iint$  $\mathrm{d}x\,\mathrm{d}y$

Light integrals $\int$  $\mathrm{d}\omega$

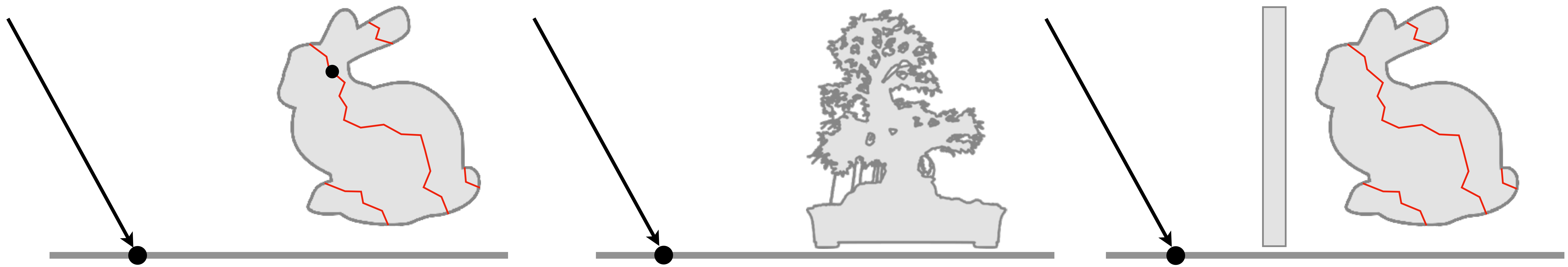
BSDF integrals $\int$  $\mathrm{d}\omega$



Scene parameter x_i

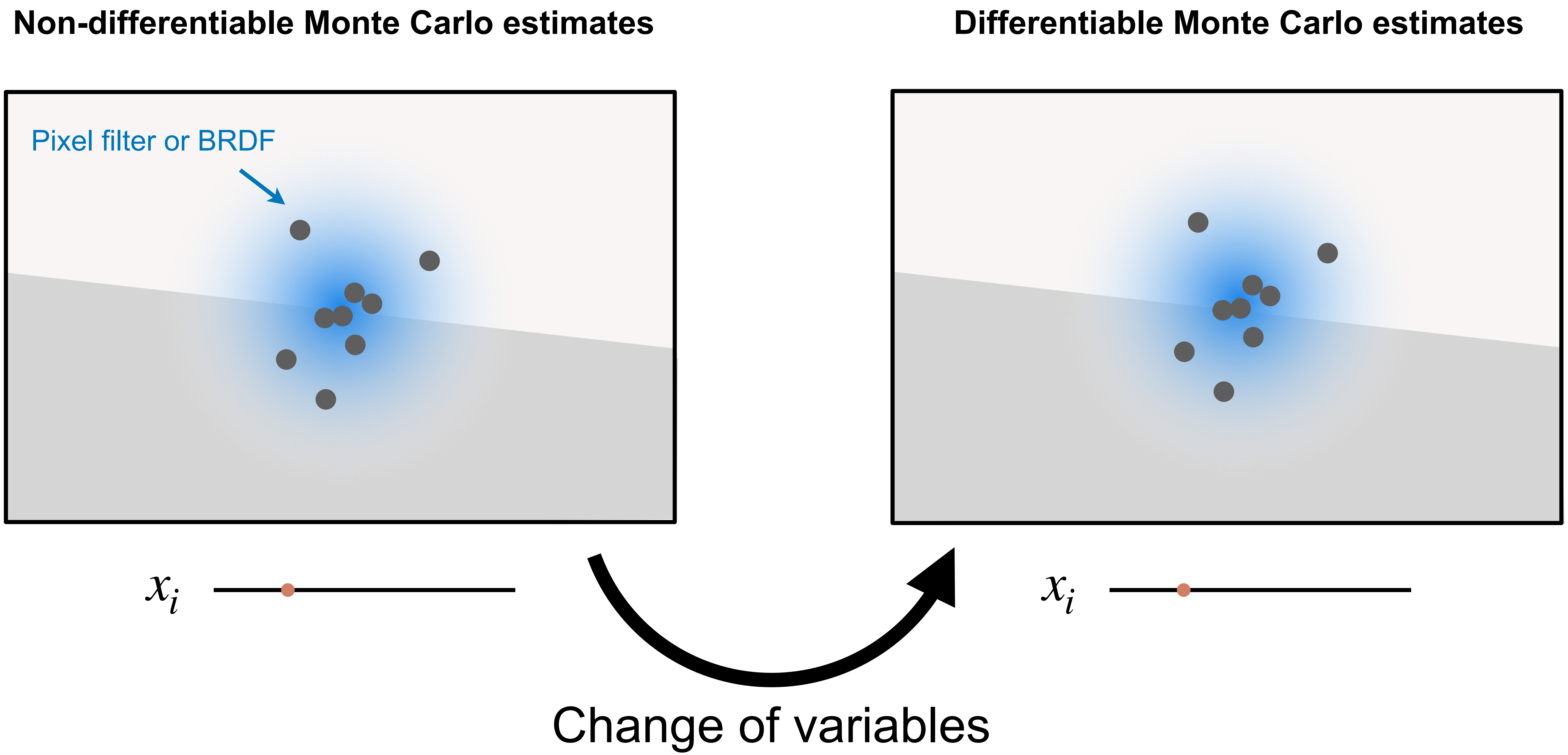
Cannot differentiate standard Monte Carlo estimates

EDGE SAMPLING

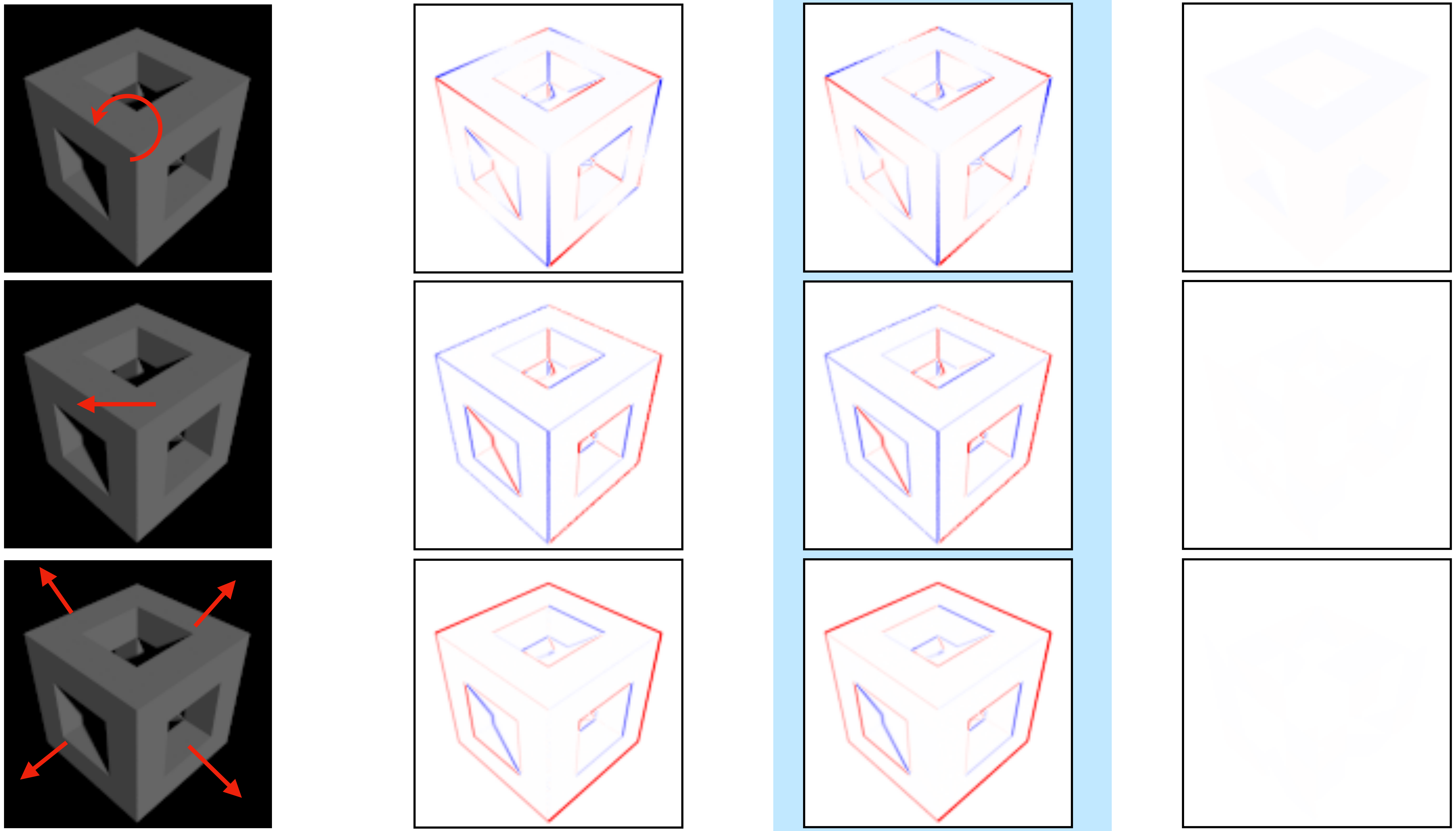


We currently don't have good acceleration data structures for this operation.

REPARAMETERIZE INTEGRALS?



RESULTS

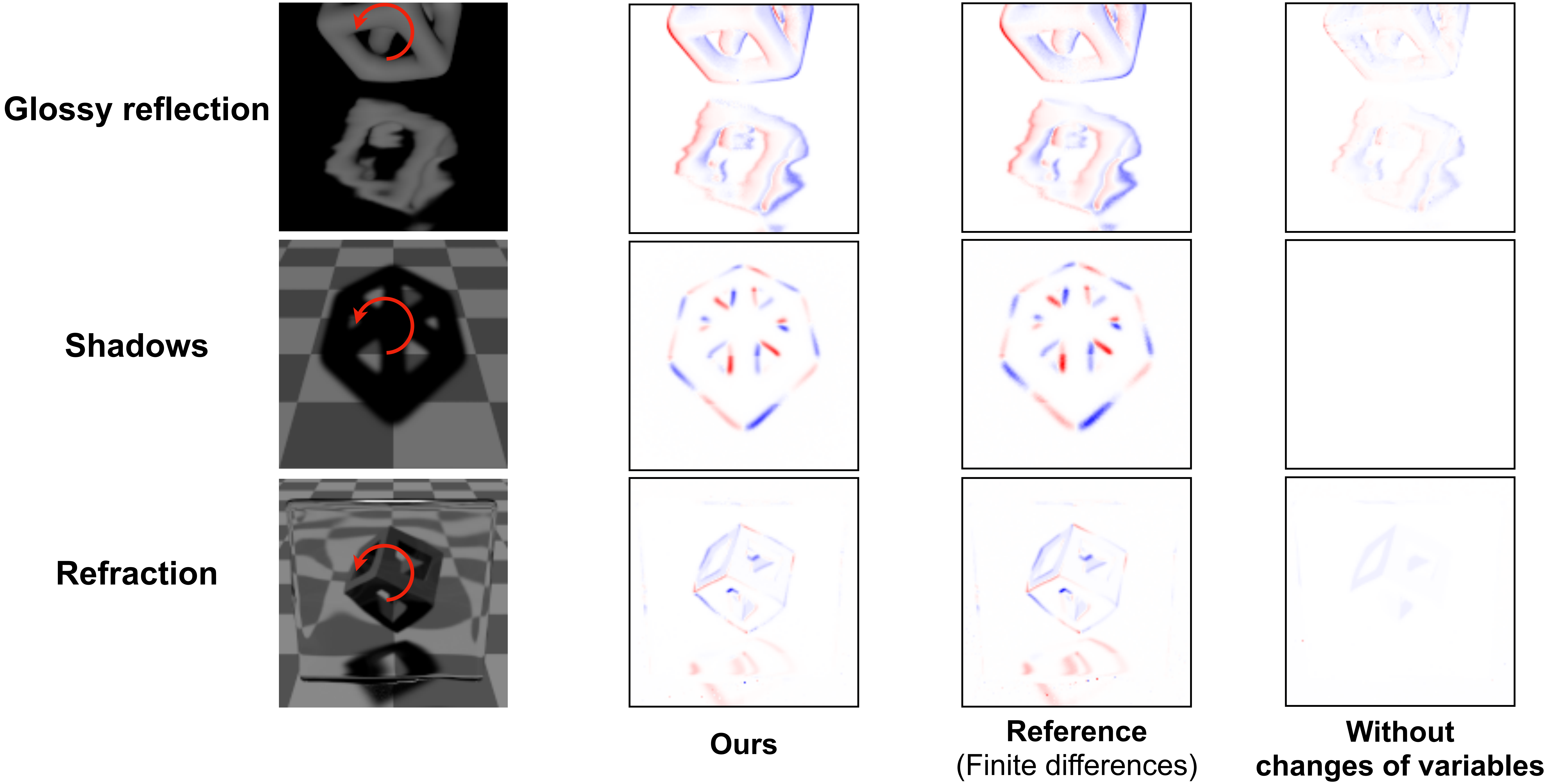


Ours

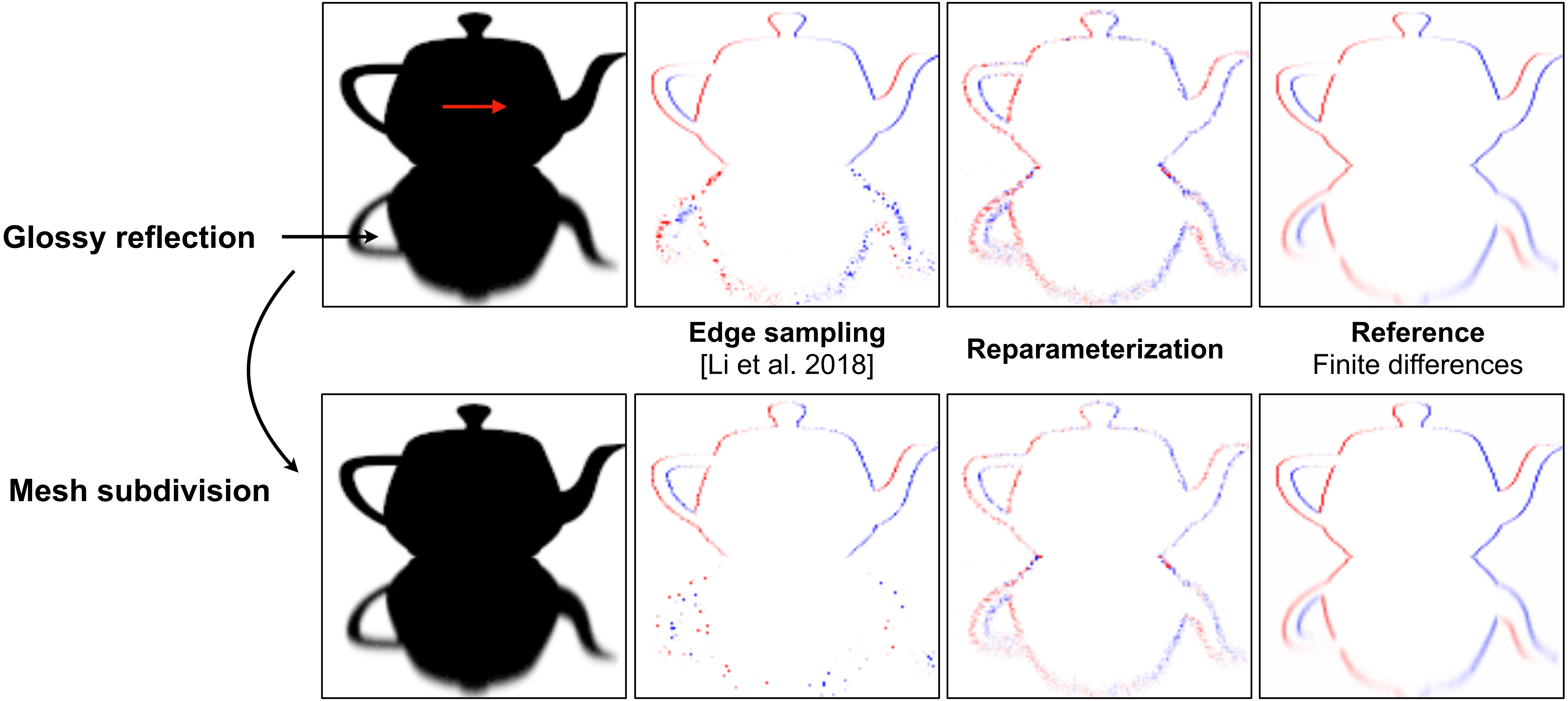
Reference
(Finite differences)

Without
changes of variables

RESULTS


Glossy reflection
Shadows
Refraction
Ours
Reference
(Finite differences)
Without
changes of variables

RESULTS


Glossy reflection
Mesh subdivision
Edge sampling
[Li et al. 2018]
Reparameterization
Reference
Finite differences

FAST INTEGRATION

Dealing with discontinuities is not enough.

Want to propagate derivative information through complex simulations with **millions** of differentiable parameters.

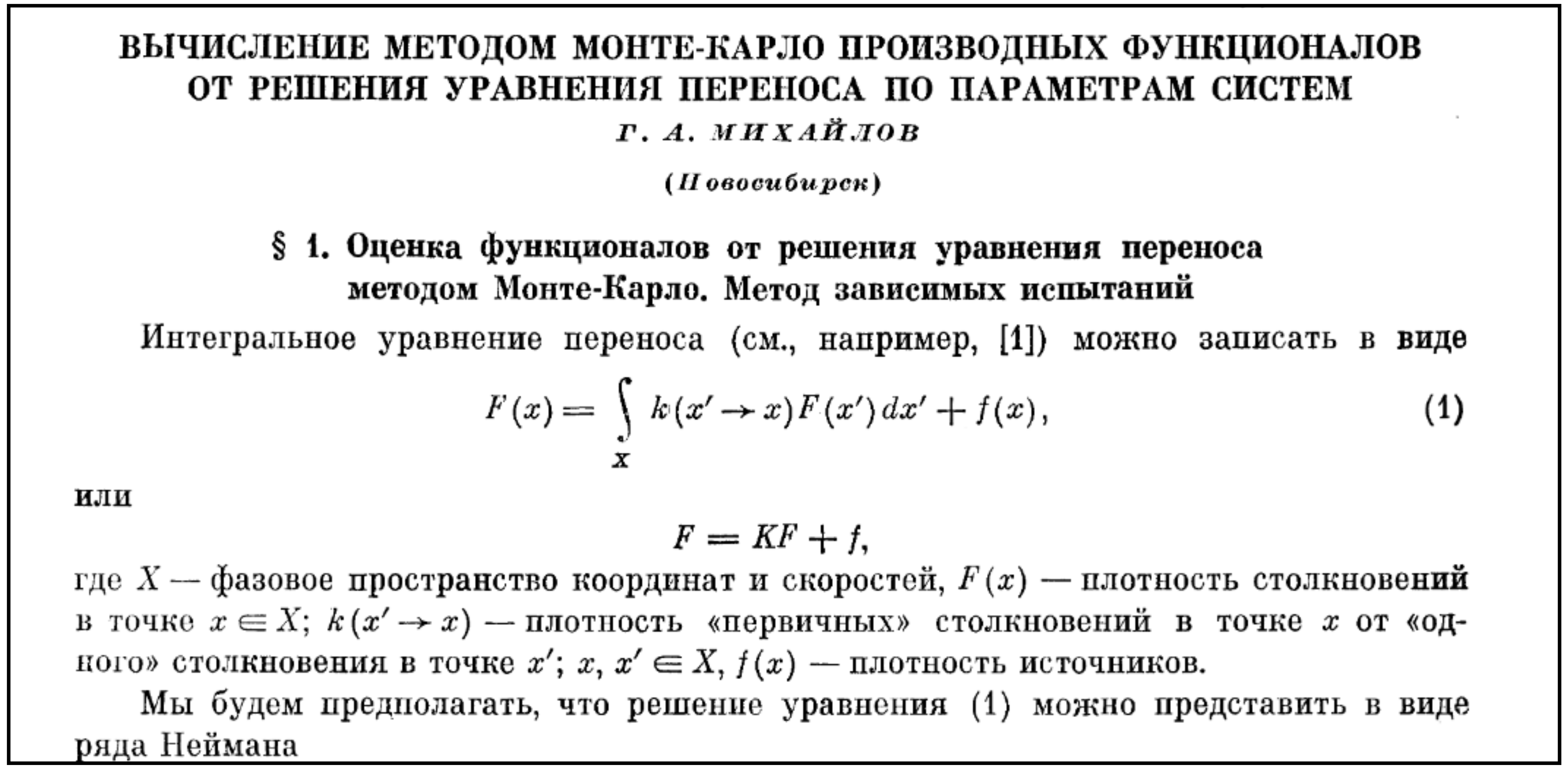
DIFFERENTIAL MONTE CARLO

“Monte-Carlo calculation of derivatives of functionals from the solution of the transfer equation according to the parameters of the system”

G. A. Mikhailov, Novosibirsk, **July 1966**

“Monte Carlo Analysis of Reactivity Coefficients in Fast Reactors, General Theory and Applications”

L.B. Miller, Argonne Natl. Laboratory, **March 1967**



ВЫЧИСЛЕНИЕ МЕТОДОМ МОНТЕ-КАРЛО ПРОИЗВОДНЫХ ФУНКЦИОНАЛОВ ОТ РЕШЕНИЯ УРАВНЕНИЯ ПЕРЕНОСА ПО ПАРАМЕТРАМ СИСТЕМ

Г. А. МИХАЙЛОВ

(Новосибирск)

§ 1. Оценка функционалов от решения уравнения переноса методом Монте-Карло. Метод зависимых испытаний

Интегральное уравнение переноса (см., например, [1]) можно записать в виде

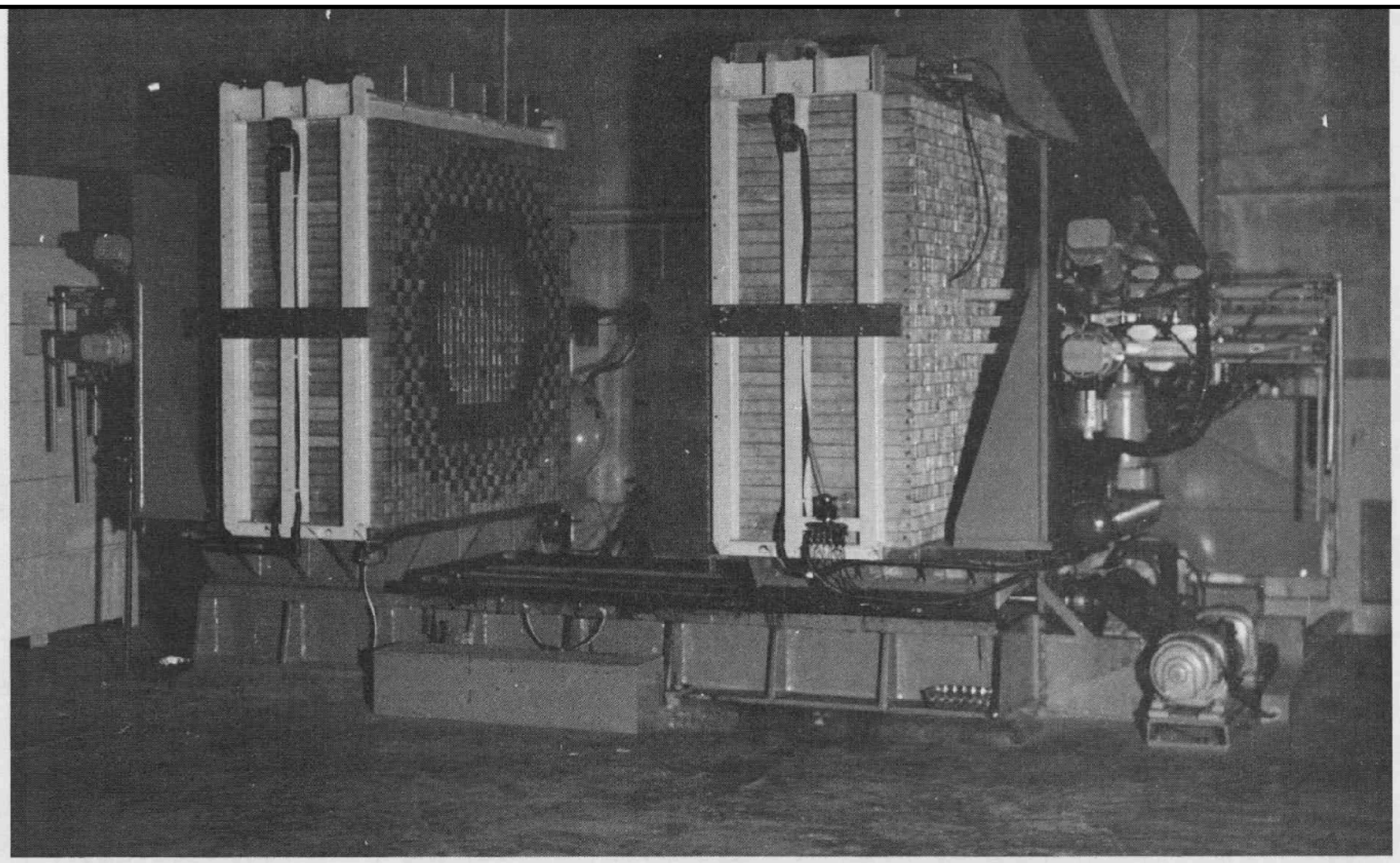
$$F(x) = \int_X k(x' \to x) F(x') dx' + f(x), \qquad (1)$$

или

$$F = KF + f,$$

где X — фазовое пространство координат и скоростей, $F(x)$ — плотность столкновений в точке $x \in X$; $k(x' \to x)$ — плотность «первичных» столкновений в точке x от «одного» столкновения в точке x'; $x, x' \in X$, $f(x)$ — плотность источников.

Мы будем предполагать, что решение уравнения (1) можно представить в виде ряда Неймана



103-298

Fig. 2. ZPR-3 Critical Facility

DIFFERENTIATING THE RENDERING EQN

TL;DR

Differential radiance is “emitted” by scene objects with differentiable parameters

Differential radiance “scatters” like normal radiance


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$
$$\frac{\partial z}{\partial y}$$


Gradients
Derivative wrt. parameters
Derivative wrt. objective
Dot product (discrete)


Gradients
Derivative wrt. parameters
Derivative wrt. objective
Dot product (discrete)

WHAT'S WRONG WITH THIS?

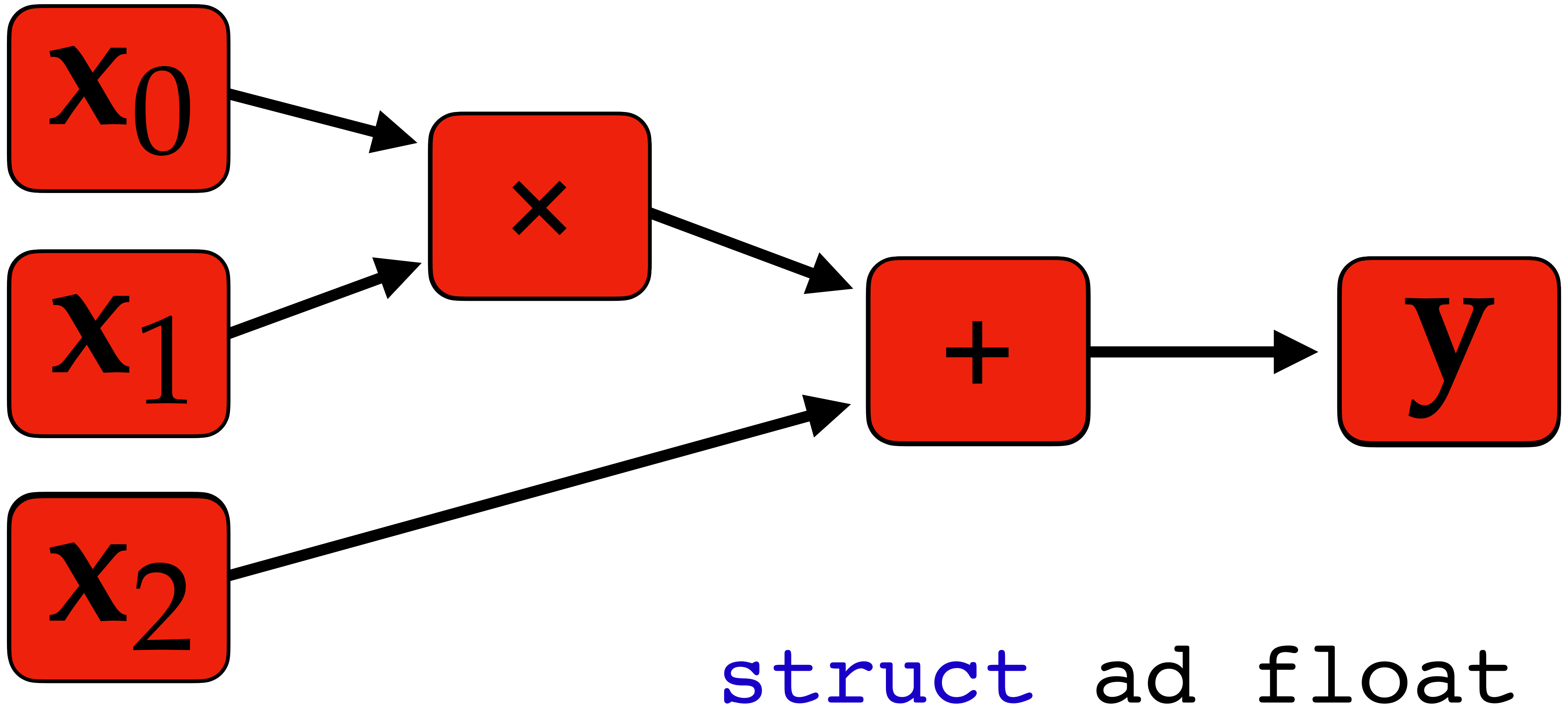
1MPix rendering &
1M parameters:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{1000000 \times 1000000}$$

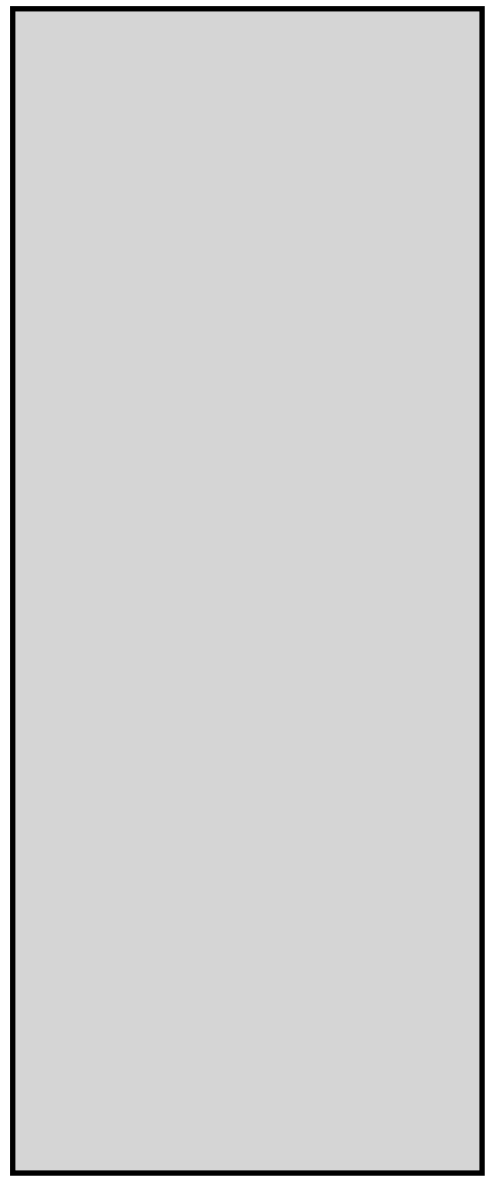
(~3.6 TiB)

Forward mode

$$\mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 + \mathbf{x}_2$$



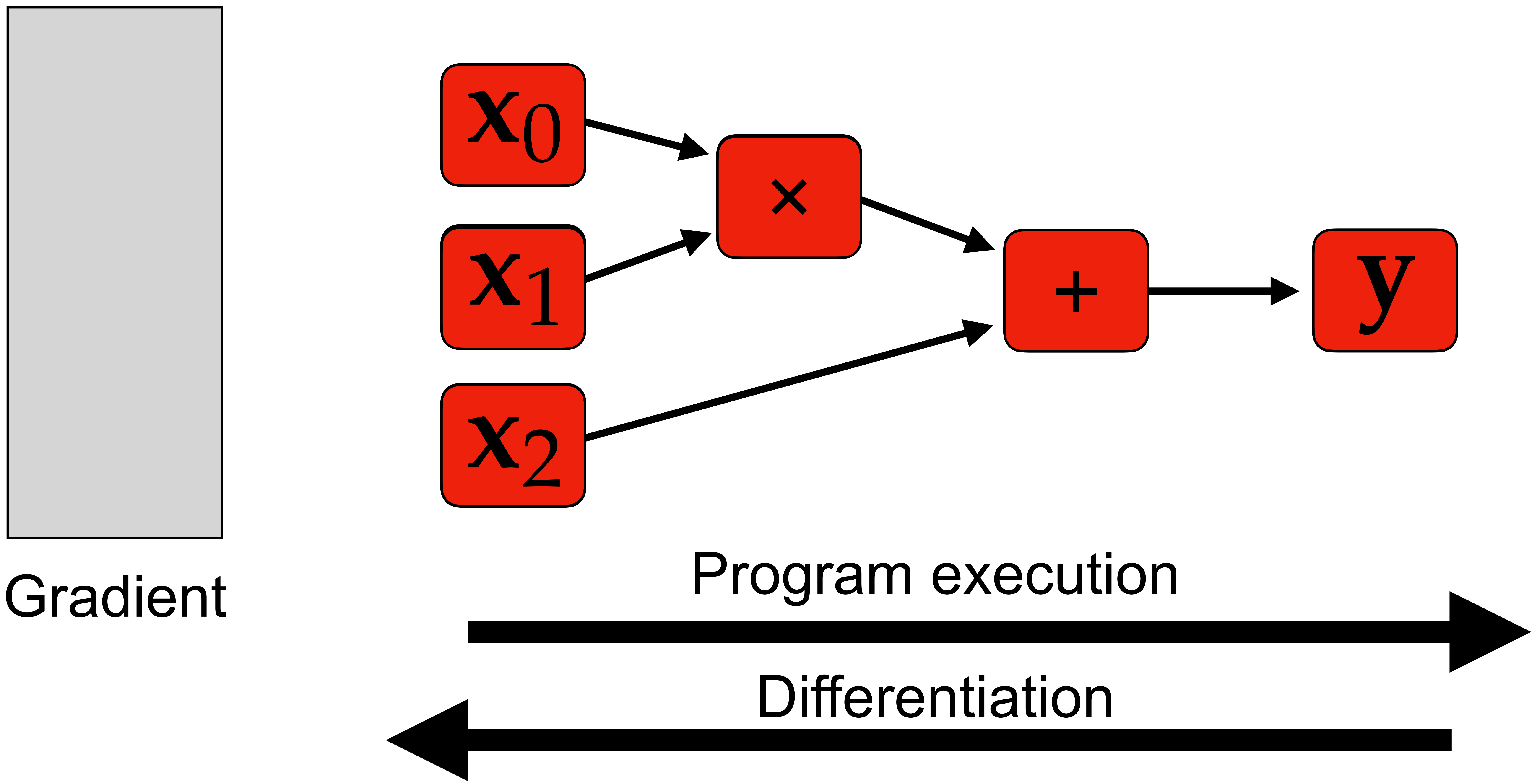

```
struct ad_float {
    float value;
    float derivative;
};
```



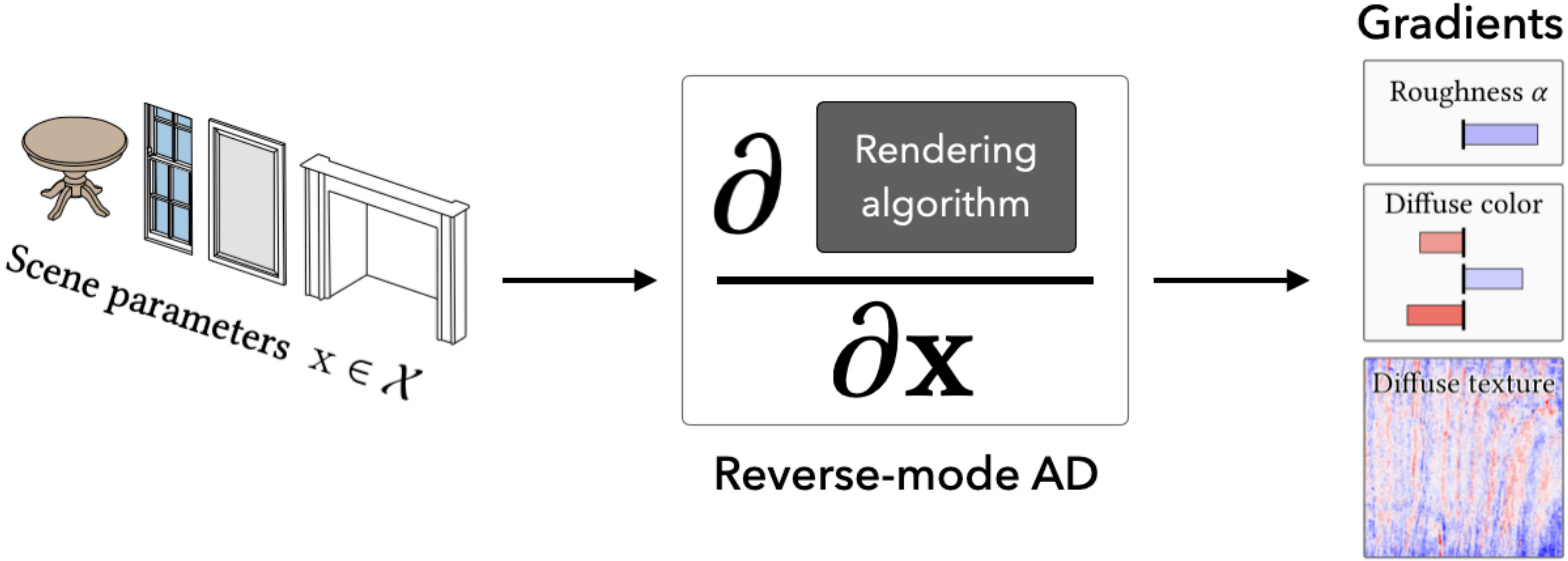
Gradient

Reverse mode

$$\mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 + \mathbf{x}_2$$

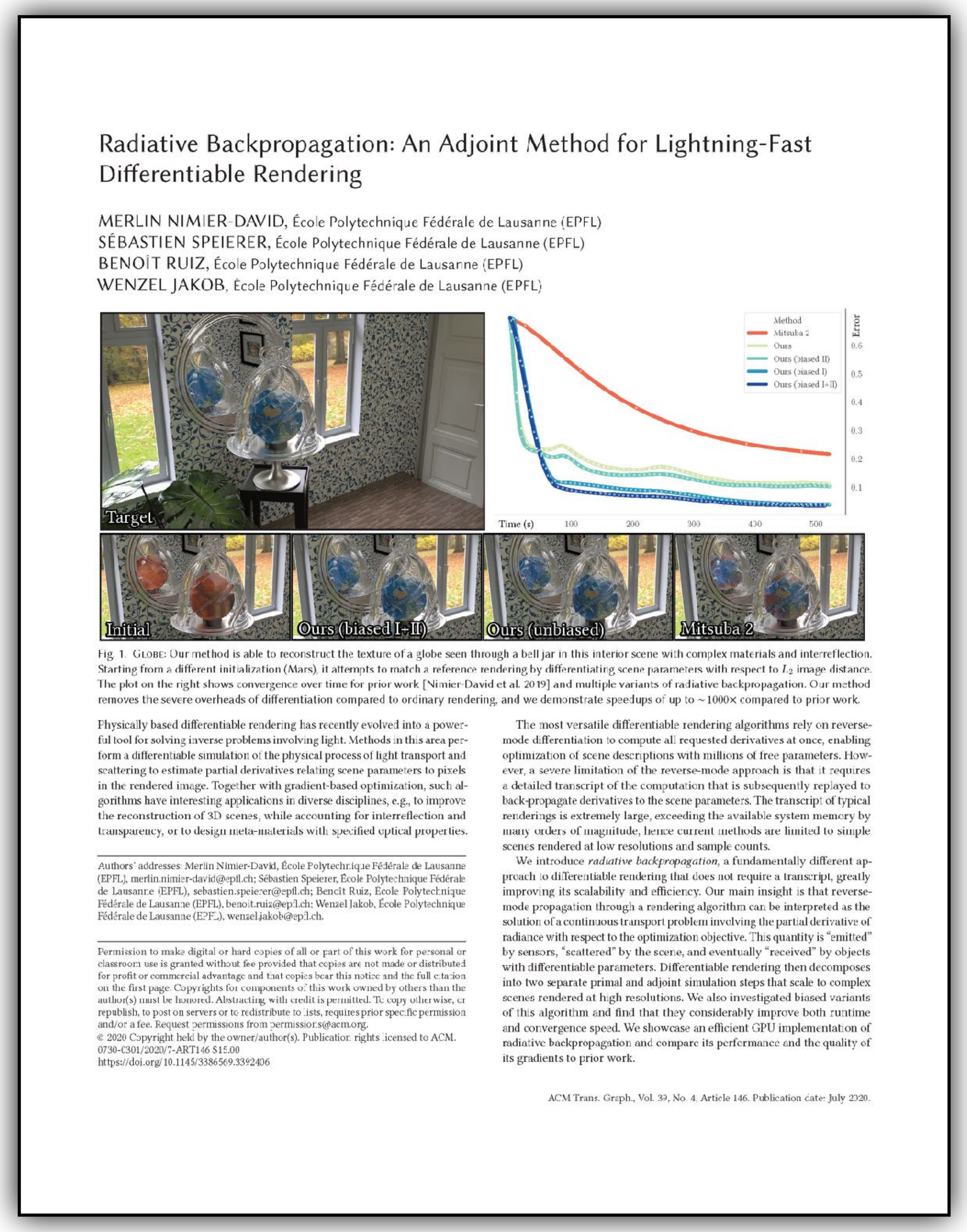


Autodiff-based differentiable rendering



Autodiff-based differentiable rendering

RADIATIVE BACKPROPAGATION



Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering

MERLIN NIMIER-DAVID, École Polytechnique Fédérale de Lausanne (EPFL)
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL)
BENOIT RUIZ, École Polytechnique Fédérale de Lausanne (EPFL)
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL)

Fig. 1. GLOBE: Our method is able to reconstruct the texture of a globe seen through a bell jar in this interior scene with complex materials and interreflection. Starting from a different initialization (Mars), it attempts to match a reference rendering by differentiating scene parameters with respect to L_2 image distance. The plot on the right shows convergence over time for prior work [Nimier-David et al. 2019] and multiple variants of radiative backpropagation. Our method removes the severe overheads of differentiation compared to ordinary rendering, and we demonstrate speedups of up to ~1000× compared to prior work.

Physically based differentiable rendering has recently evolved into a powerful tool for solving inverse problems involving light. Methods in this area perform a differentiable simulation of the physical process of light transport and scattering to estimate partial derivatives relating scene parameters to pixels in the rendered image. Together with gradient-based optimization, such algorithms have interesting applications in diverse disciplines, e.g., to improve the reconstruction of 3D scenes, while accounting for interreflection and transparency, or to design meta-materials with specified optical properties.

The most versatile differentiable rendering algorithms rely on reverse-mode differentiation to compute all requested derivatives at once, enabling optimization of scene descriptions with millions of free parameters. However, a severe limitation of the reverse-mode approach is that it requires a detailed transcript of the computation that is subsequently replayed to back-propagate derivatives to the scene parameters. The transcript of typical renderings is extremely large, exceeding the available system memory by many orders of magnitude, hence current methods are limited to simple scenes rendered at low resolutions and sample counts.

We introduce *radiative backpropagation*, a fundamentally different approach to differentiable rendering that does not require a transcript, greatly improving its scalability and efficiency. Our main insight is that reverse-mode propagation through a rendering algorithm can be interpreted as the solution of a continuous transport problem involving the partial derivative of radiance with respect to the optimization objective. This quantity is "emitted" by sensors, "scattered" by the scene, and eventually "received" by objects with differentiable parameters. Differentiable rendering then decomposes into two separate primal and adjoint simulation steps that scale to complex scenes rendered at high resolutions. We also investigated biased variants of this algorithm and find that they considerably improve both runtime and convergence speed. We showcase an efficient GPU implementation of radiative backpropagation and compare its performance and the quality of its gradients to prior work.

Authors' addresses: Merlin Nimier-David, École Polytechnique Fédérale de Lausanne (EPFL), merlin.nimier-david@epfl.ch; Sébastien Speierer, École Polytechnique Fédérale de Lausanne (EPFL), sebastien.speierer@epfl.ch; Benoît Ruiz, École Polytechnique Fédérale de Lausanne (EPFL), benoit.ruiz@epfl.ch; Wenzel Jakob, École Polytechnique Fédérale de Lausanne (EPFL), wenzel.jakob@epfl.ch.

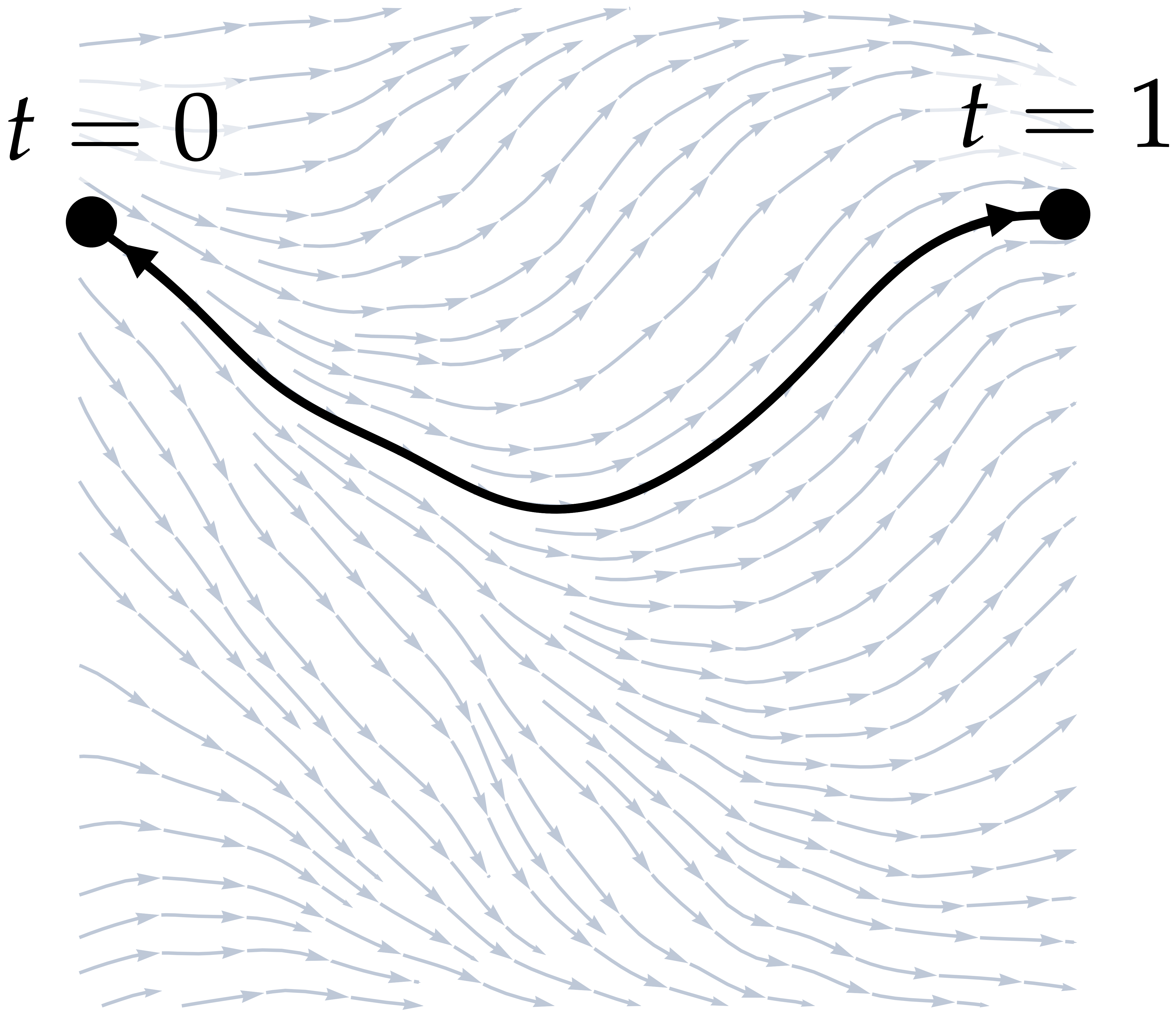
0730-0301/2020/7-ART146 $15.00
https://doi.org/10.1145/3386569.3392406

Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering

Merlin Nimier-David, Sébastien Speierer, Benoit Ruîz, Wenzel Jakob

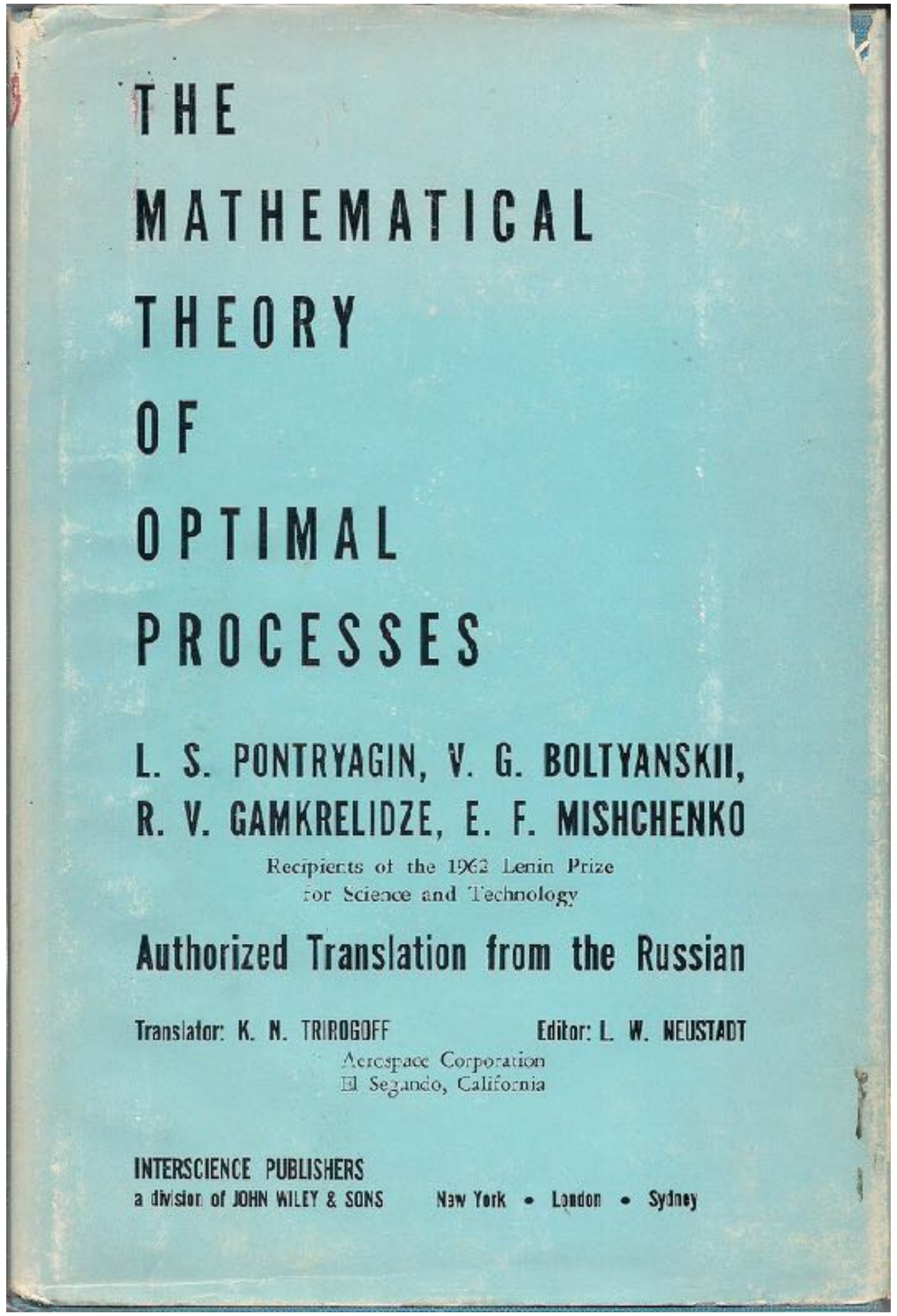
SIGGRAPH 2020

MOTIVATION: ADJOINT SENSITIVITY METHOD

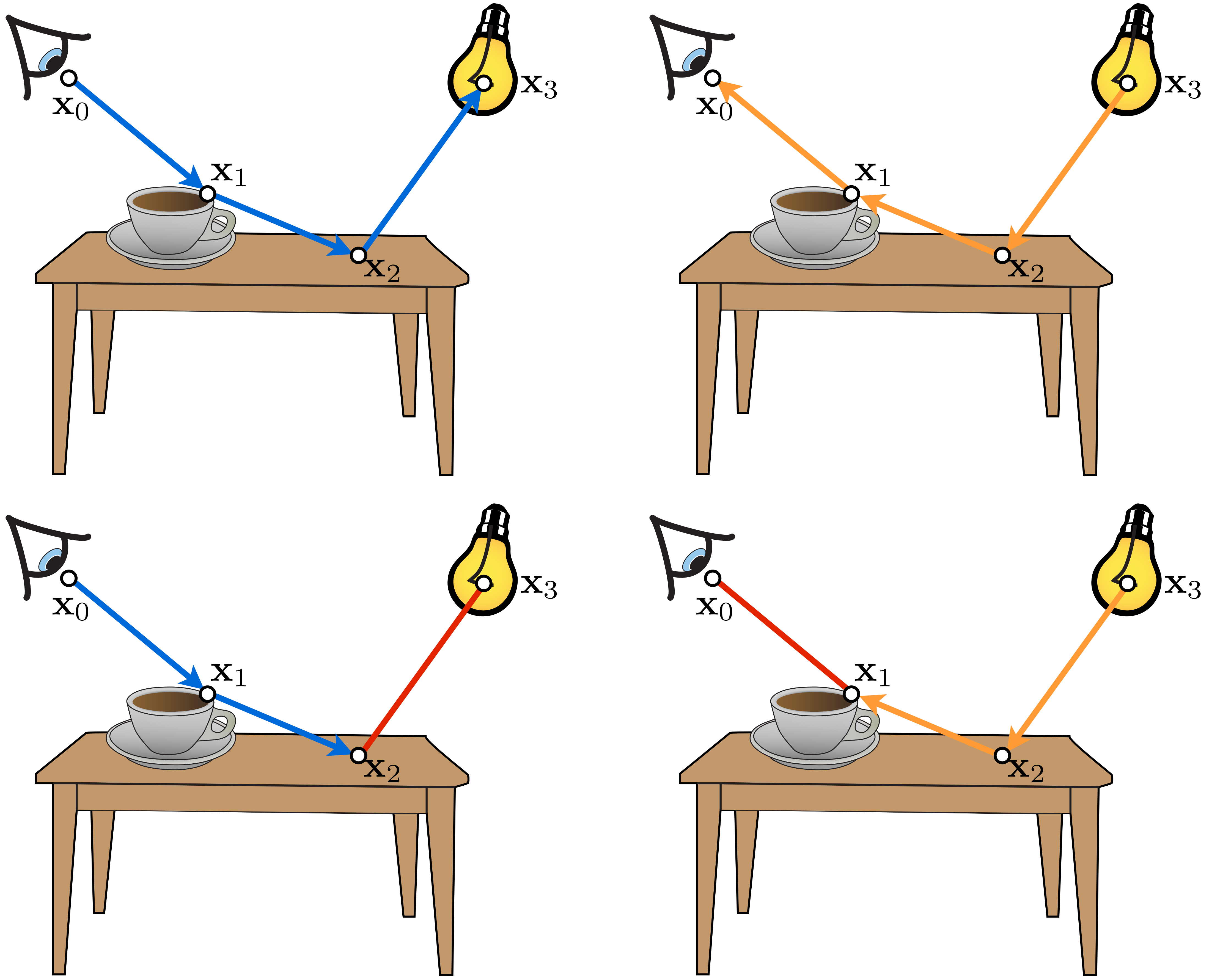


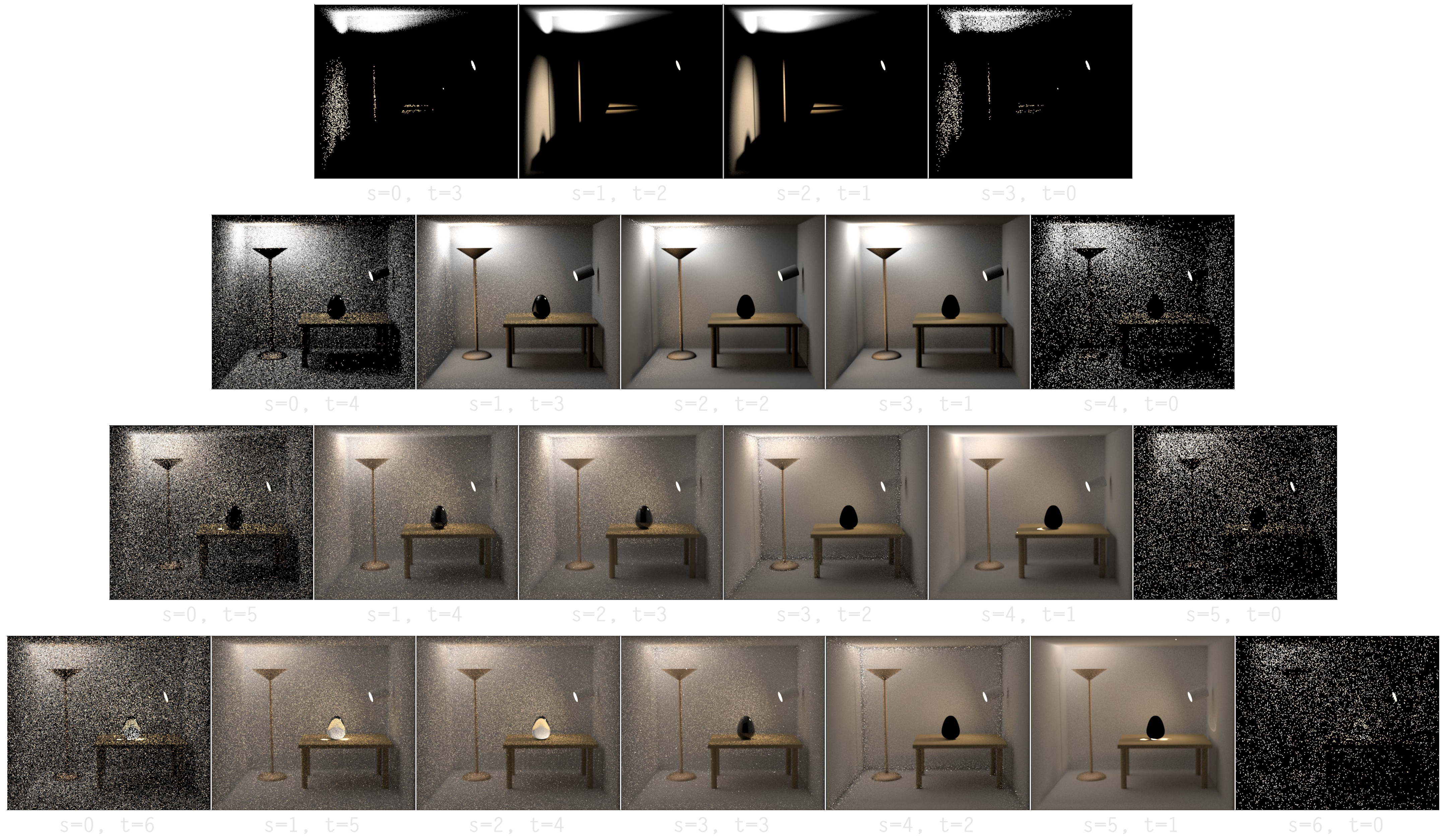
For problems with a time dimension (ODEs, ..)

Pontryagin et al.
1962



“ADJOINT” – THAT SOUNDS FAMILIAR!





Bidirectional Estimators for Light Transport

Veach & Guibas, **1994**

$$\langle O\mathbf{a}, \mathbf{b} \rangle = \langle \mathbf{a}, O\mathbf{b} \rangle$$

(Underlying principle: self-adjoint operators)

Derivatives projected into the scene

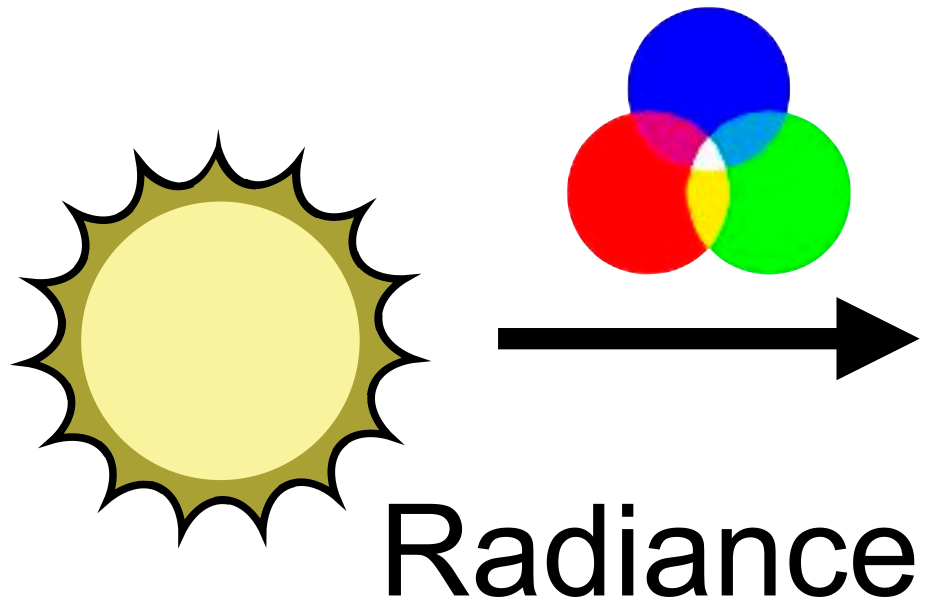

Gradients
Deriv. from objects
Deriv. from sensor
Product integral

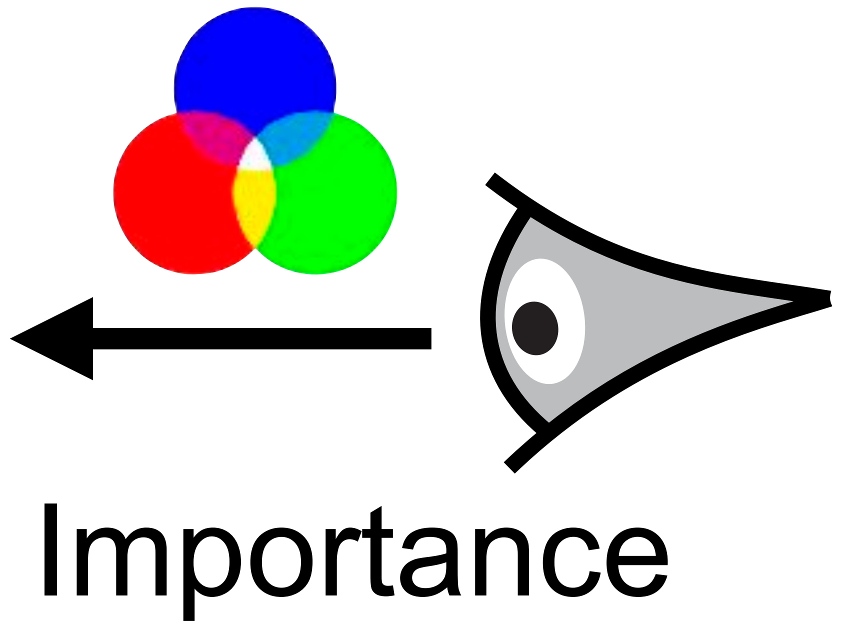

$f_s : (\omega_i, \omega_o, \underbrace{x_1, x_2, \ldots}_{\text{parameters}}) \mapsto \mathbb{R}$

ANOTHER PERSPECTIVE

Normal rendering

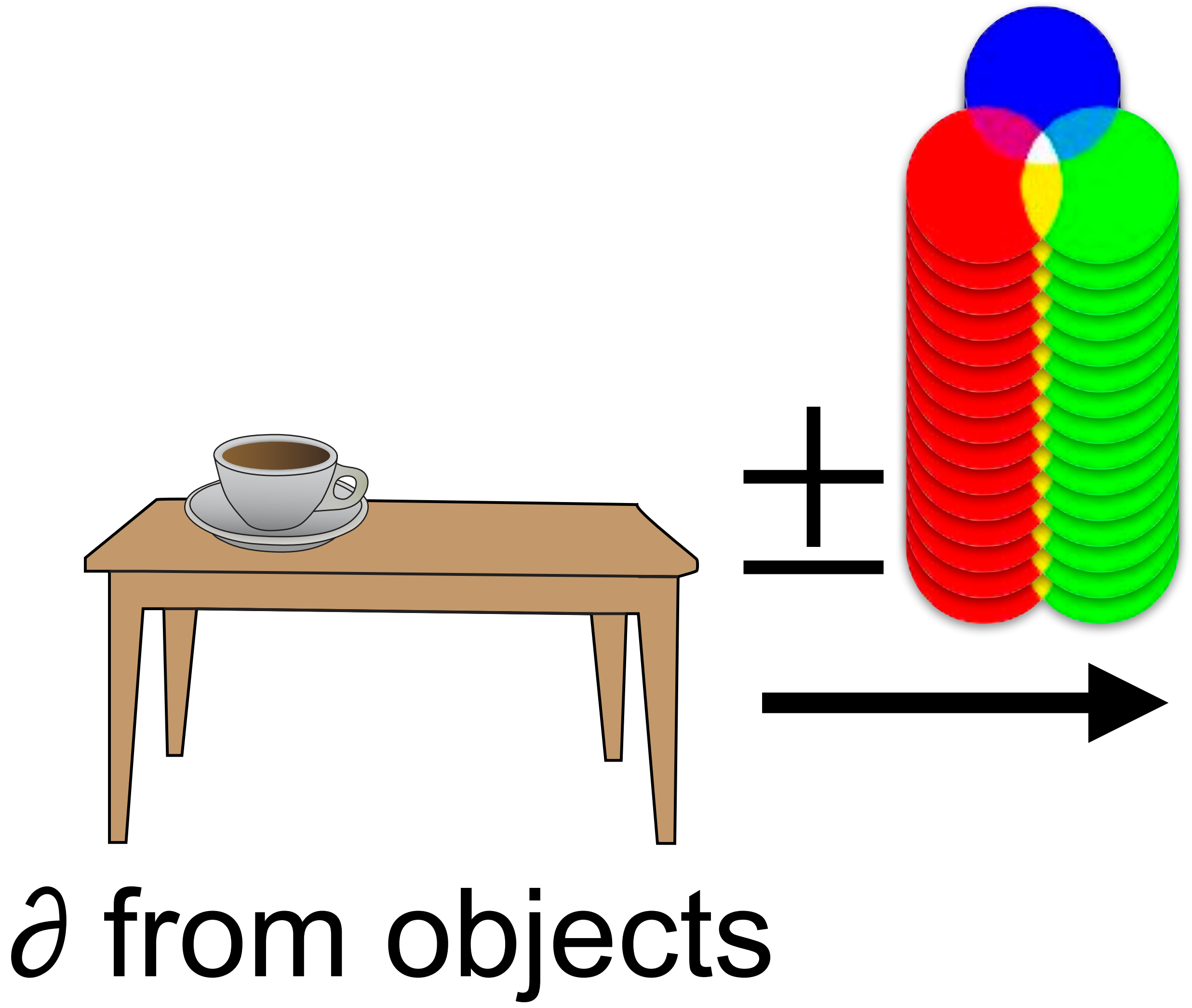
- Transporting from sensor/light may yield lower variance.

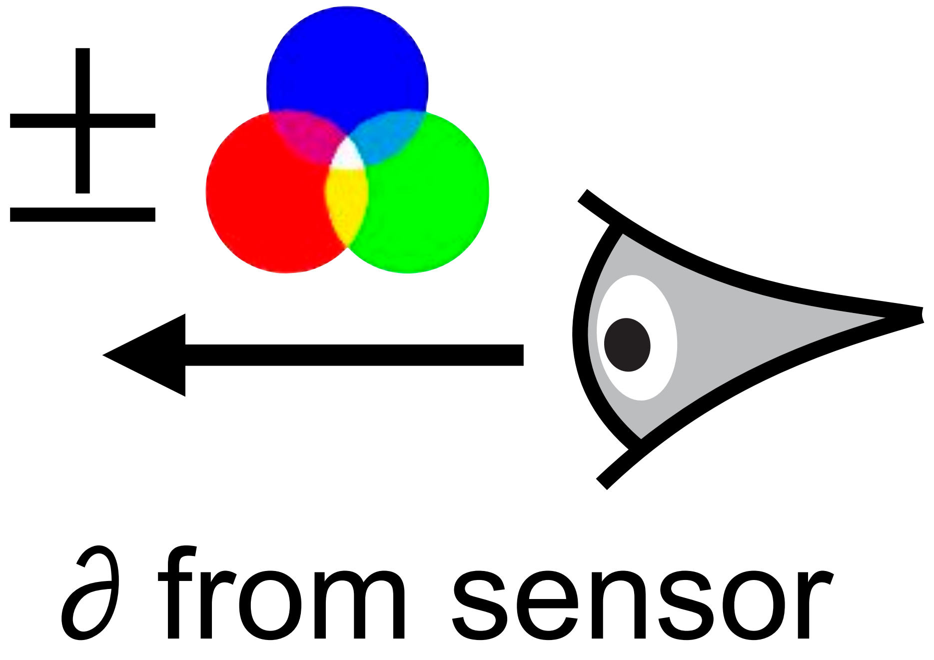




Differentiable rendering

- Transporting from objects is **completely impractical**.





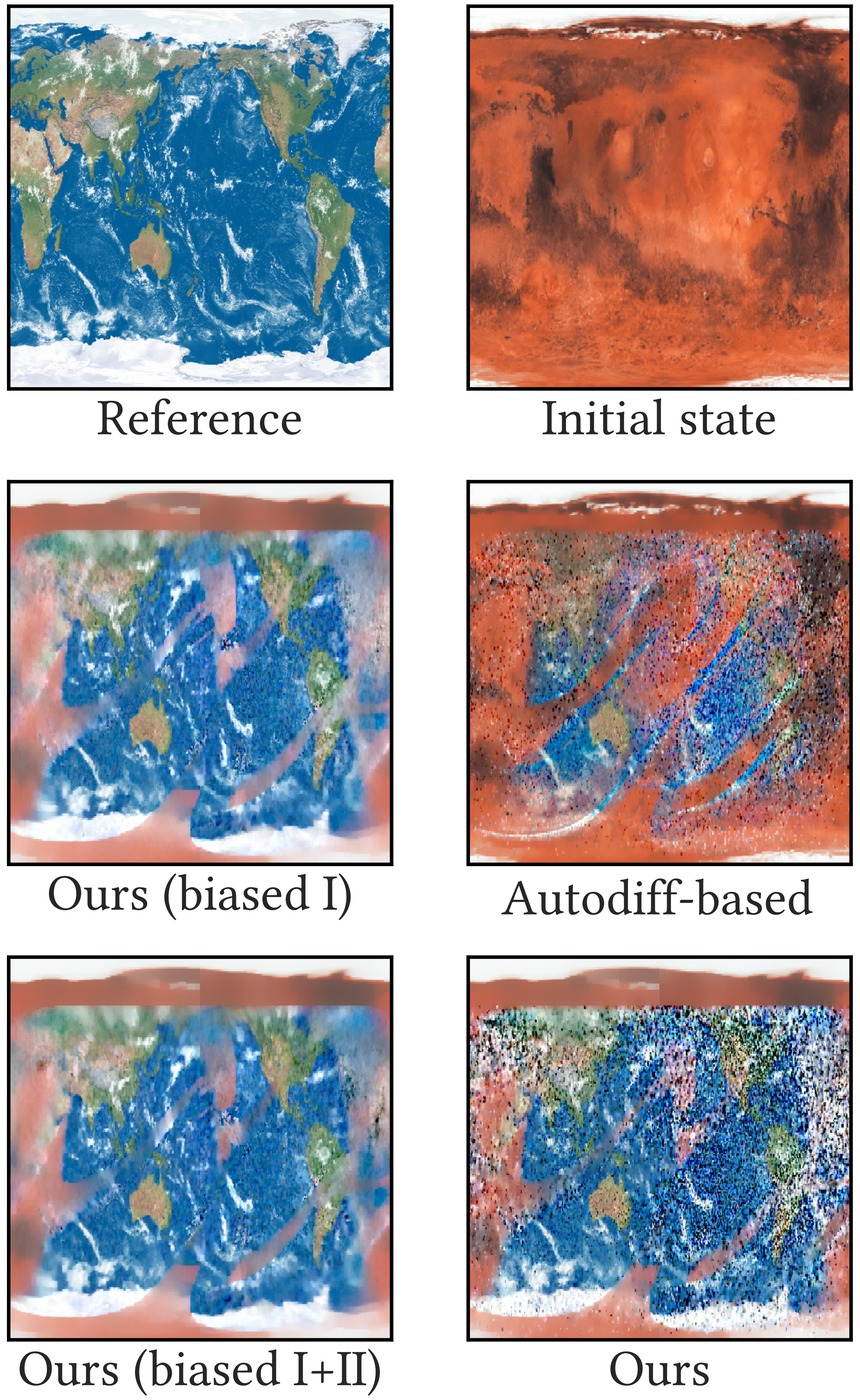
Surface texture optimization

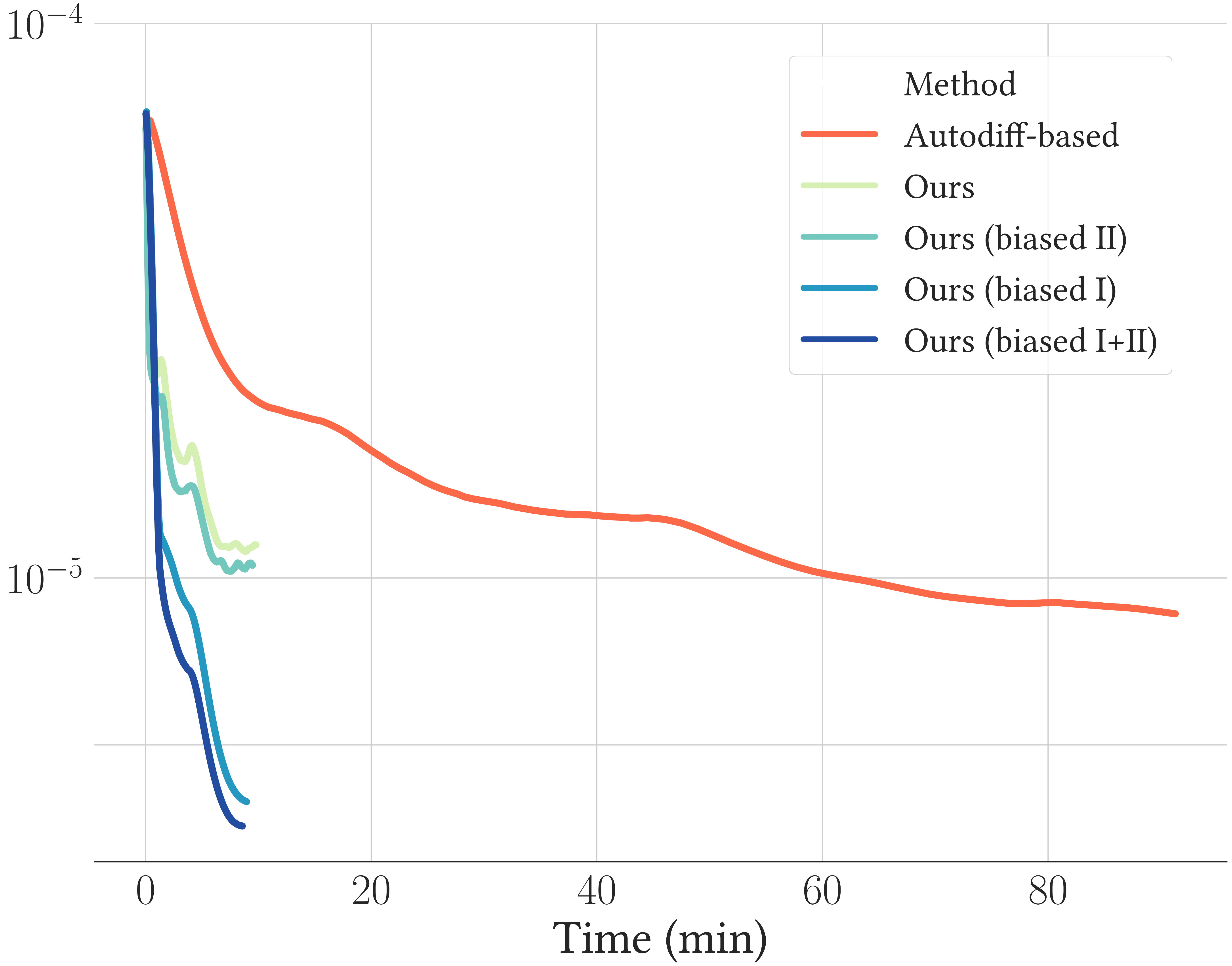
Initial state

Target state

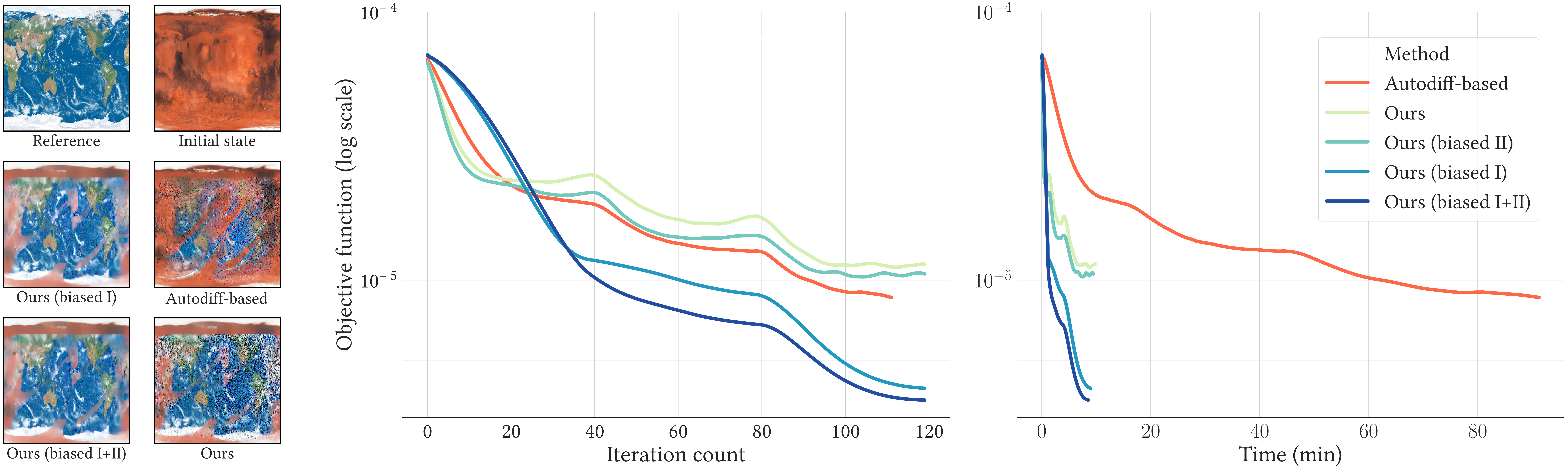

Optimized texture
Target

Surface BSDF optimization

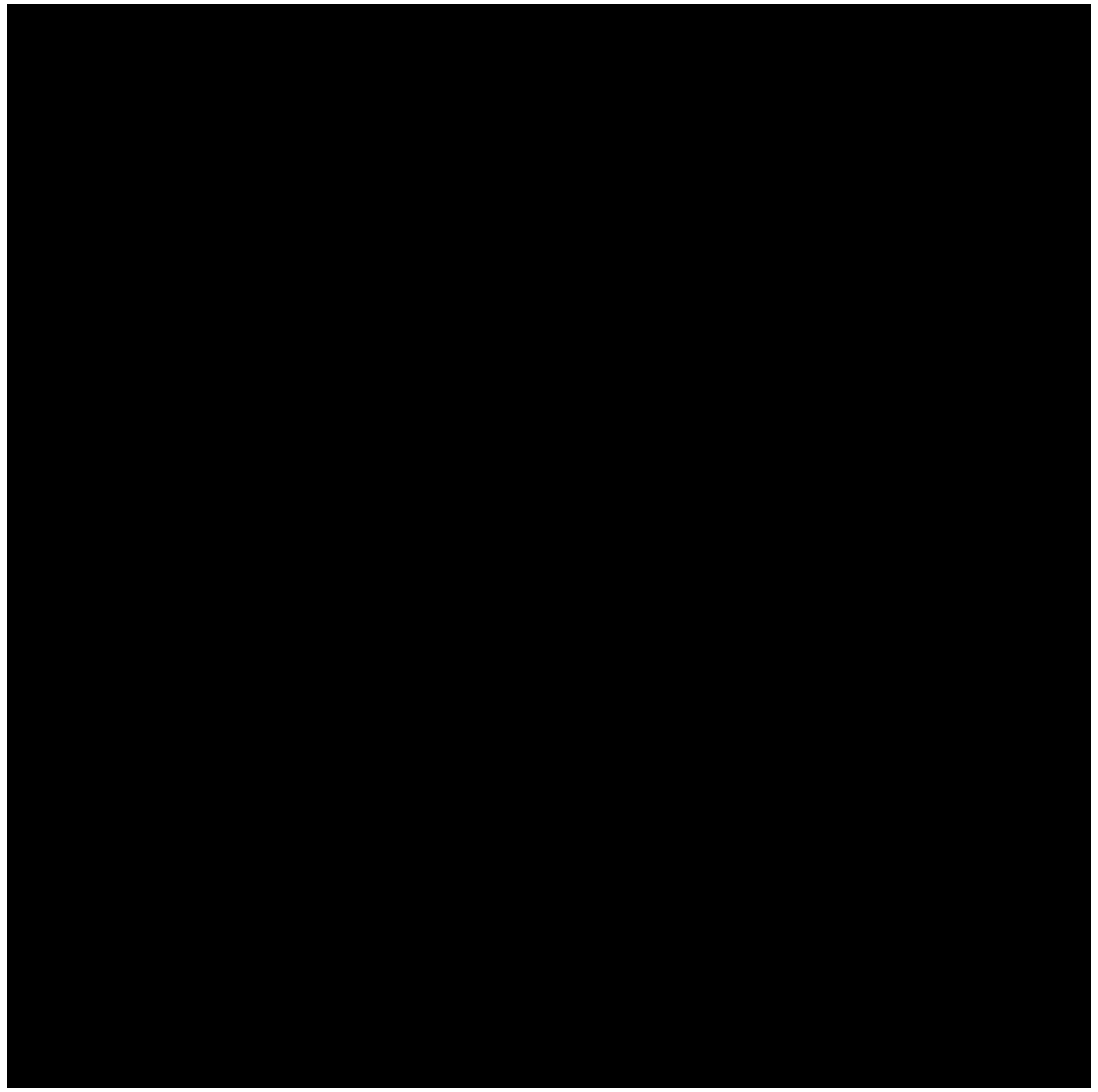




Surface BSDF optimization

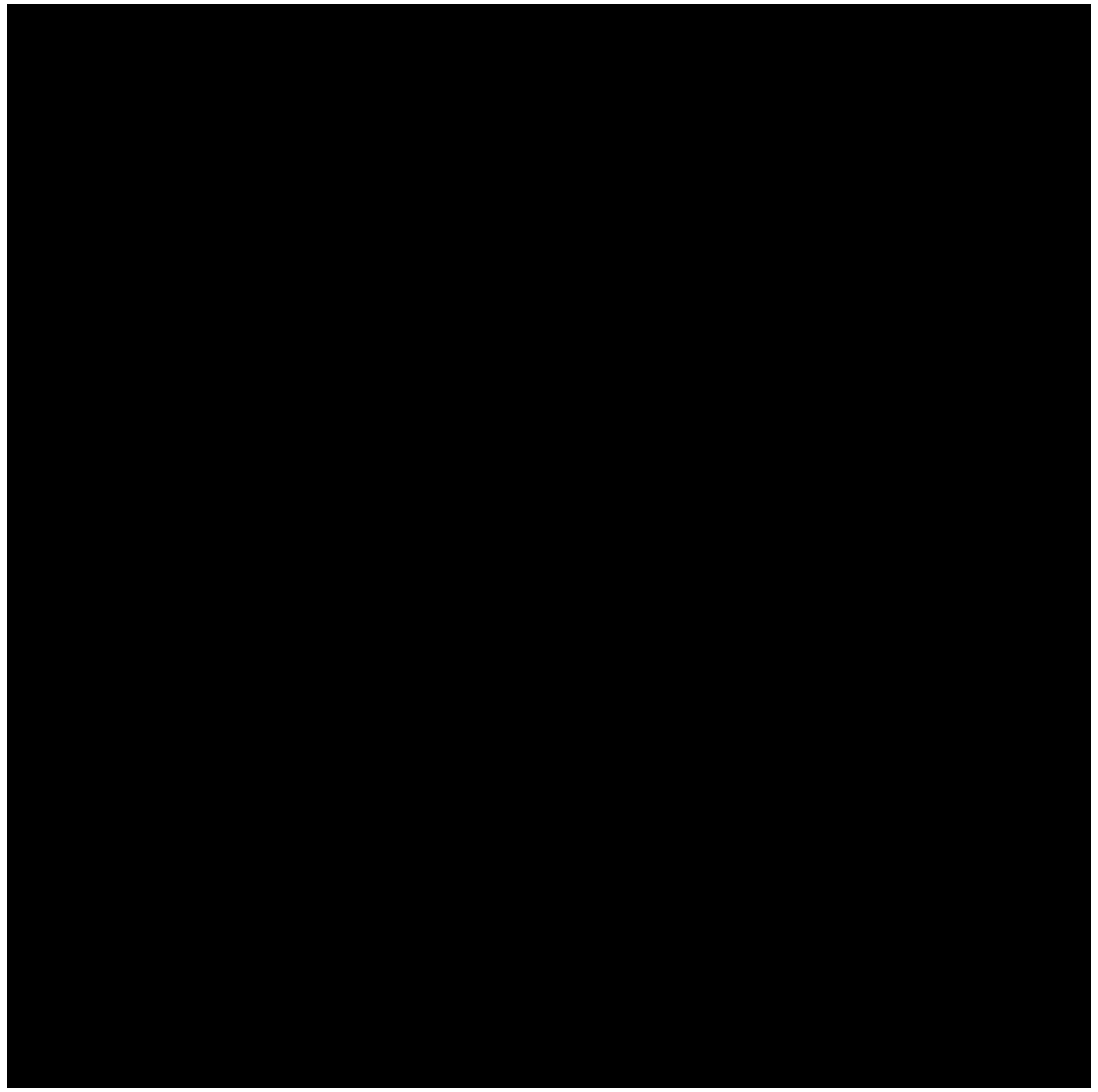


Volume density optimization



Mitsuba 2 (AD-based)

Radiative Backprop.
(biased I + II)



Target

Volume density optimization


Reference

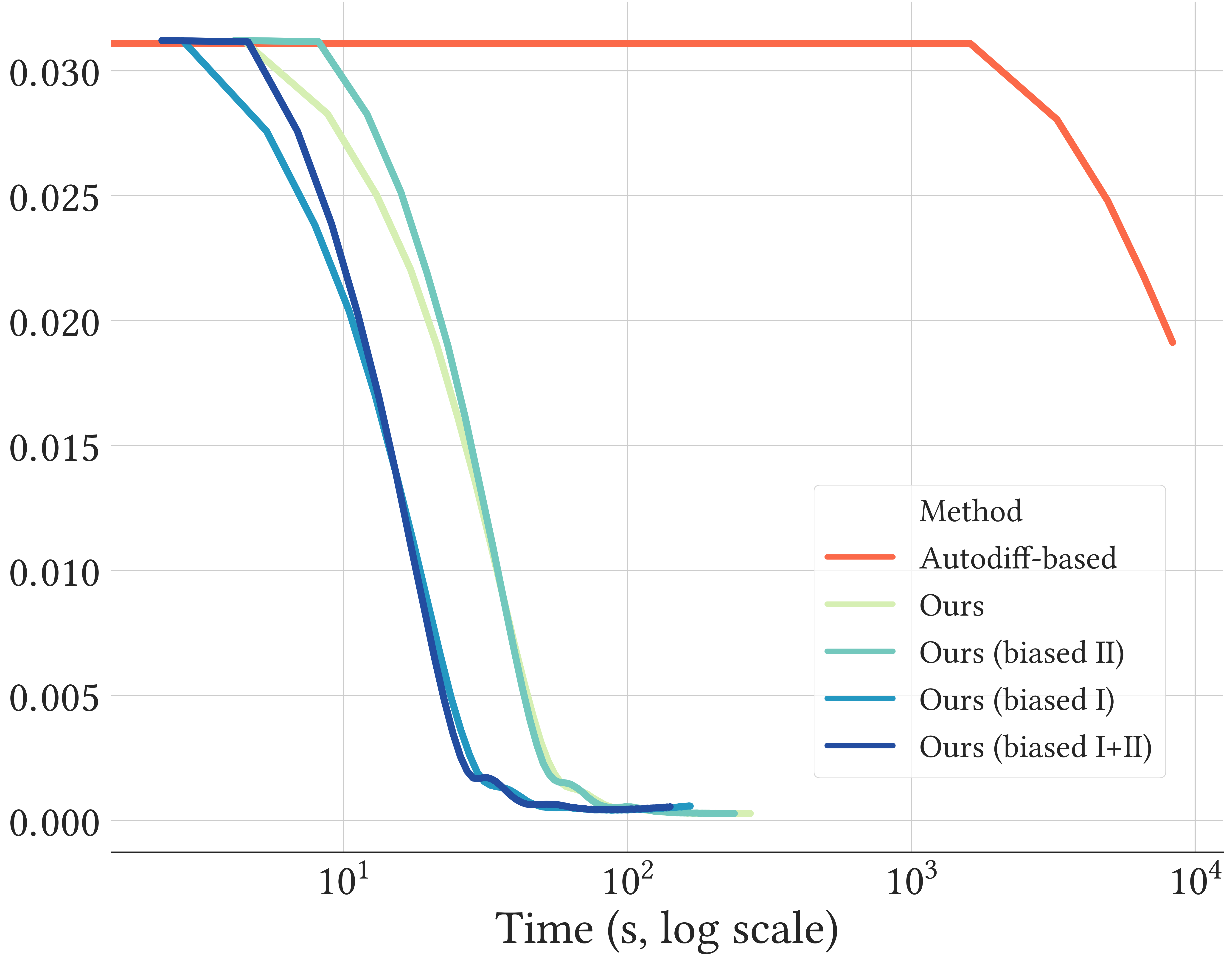

Initial state


Ours (biased I)


Autodiff-based

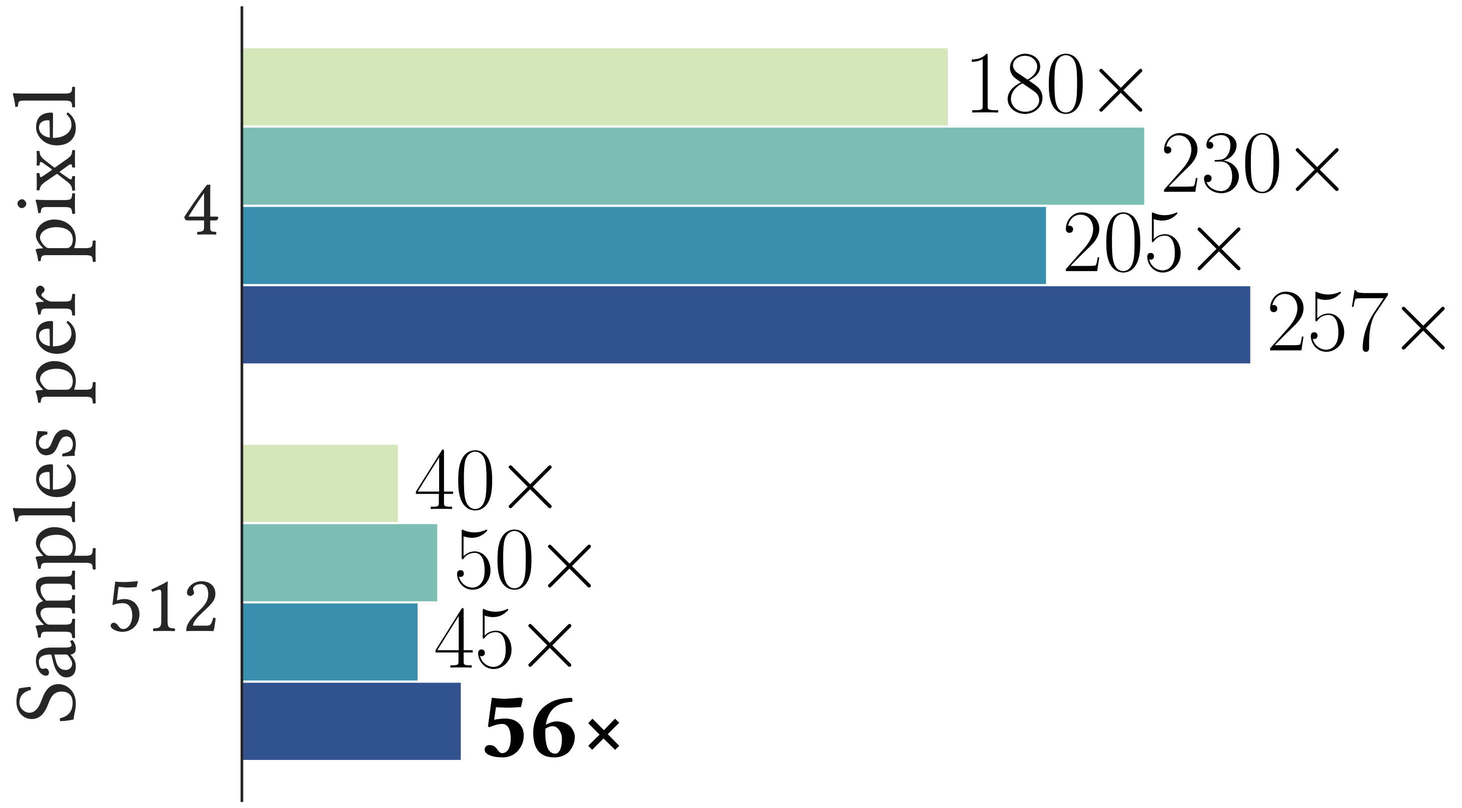

Ours (biased I+II)


Ours

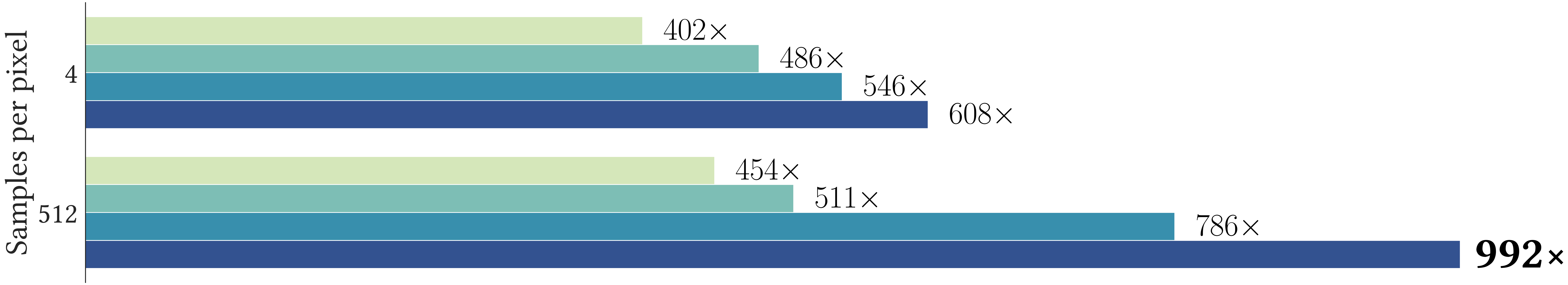


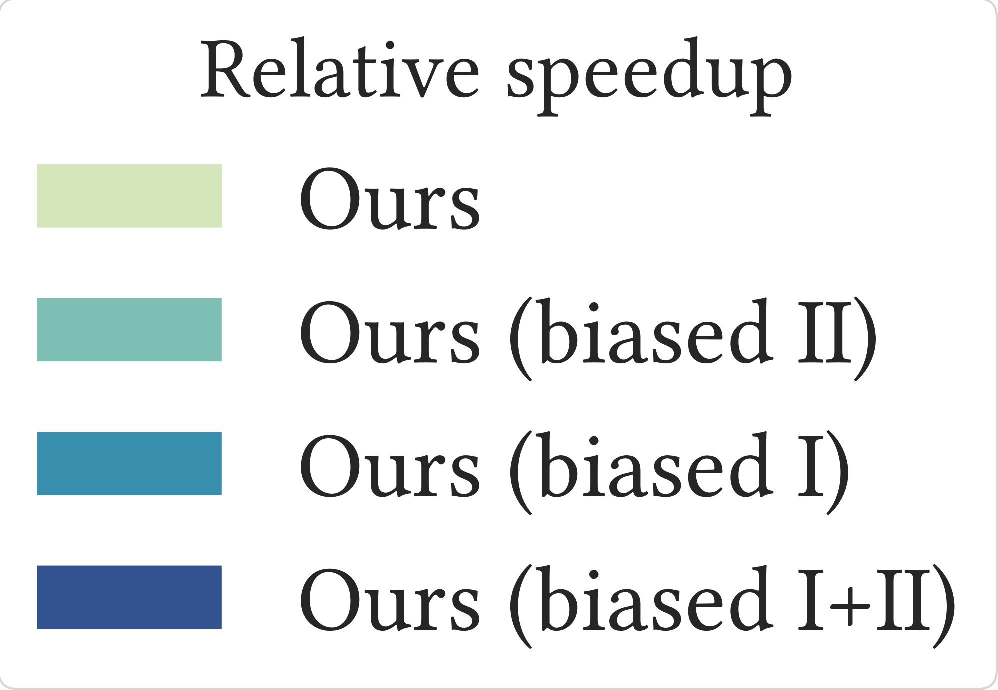
Relative speedups *vs* autodiff-based

Surface texture optimization



Volume density optimization

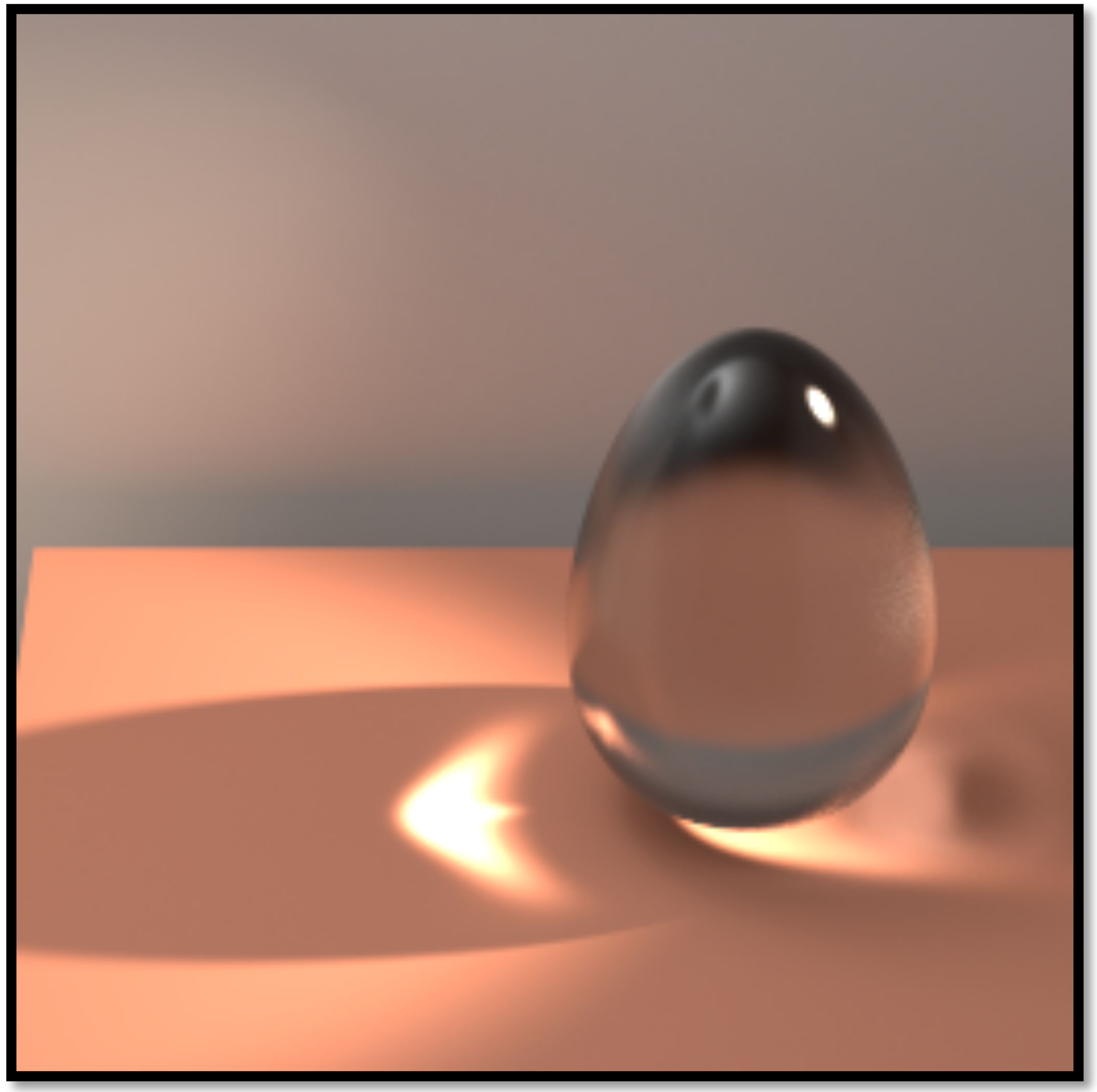




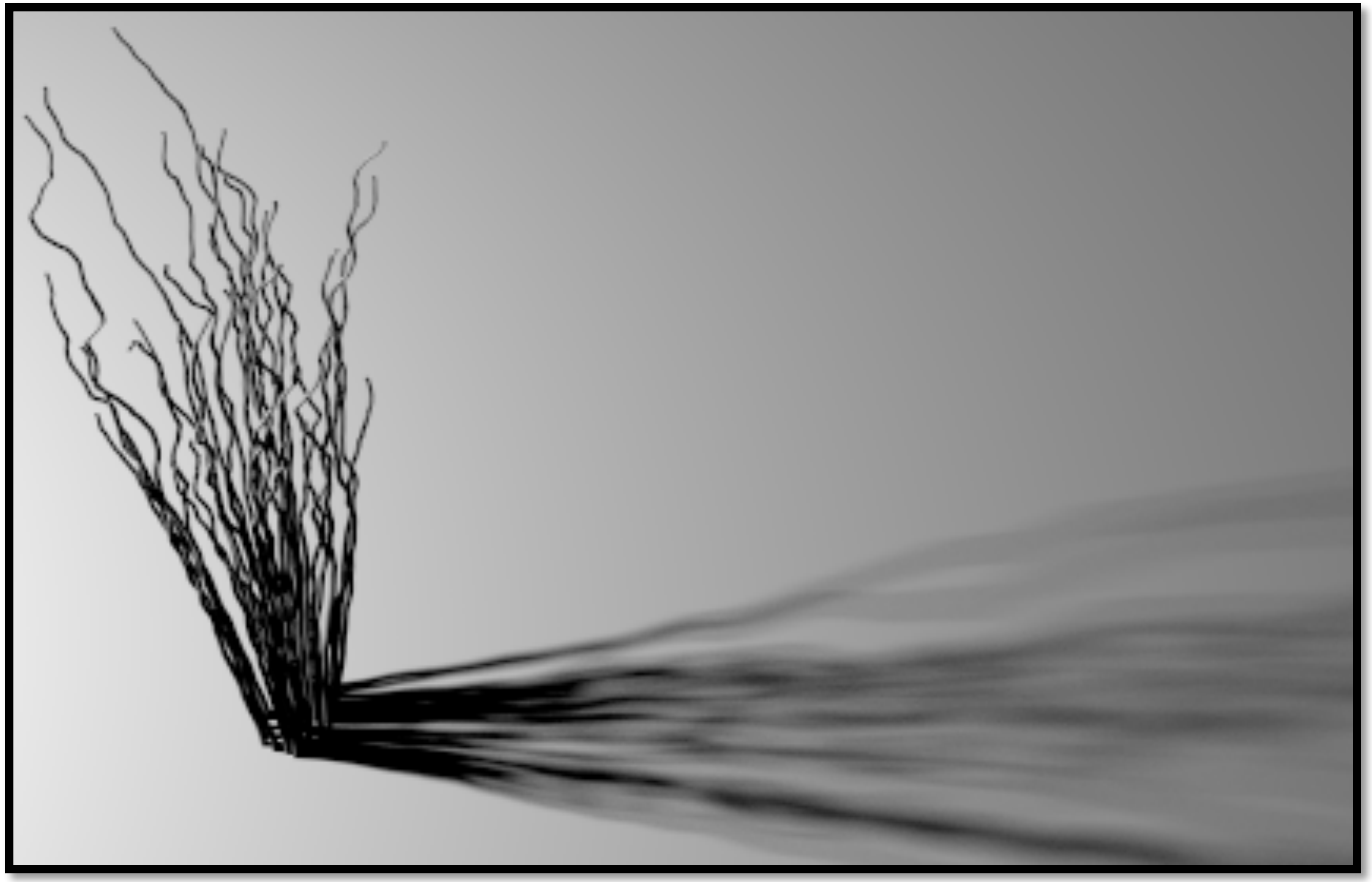
TL;DR

- Radiative Backpropagation is **"just" another kind** of light transport simulation with weird sensors and emitters.
  - Orders of magnitude faster (up to ~1000× in our experiments)
  - Lifts memory limitations entirely
  - Only need to differentiate BSDFs etc. ("easy")
  - Can build on decades of research targeting such problems!

CHALLENGES REMAIN



Complex light transport



Complex geometry & motion

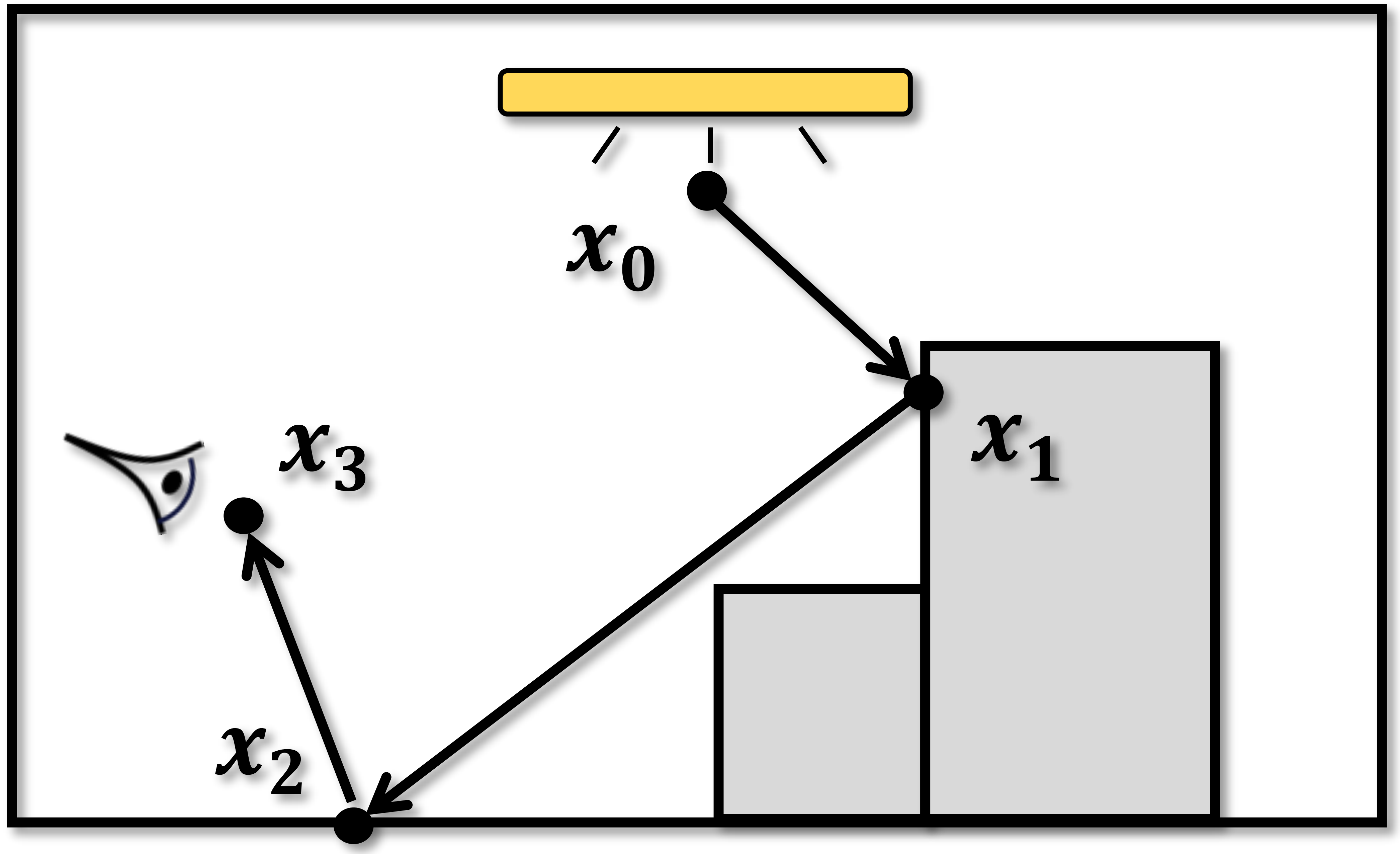
PATH INTEGRAL FOR FORWARD RENDERING

Measurement contribution function

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

Path space

Area-product measure



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

- Introduced by Veach [1997]
- Foundation of sophisticated Monte Carlo algorithms (e.g., BDPT, MCMC rendering)

Can we have something similar for **differentiable rendering**?

PATH-SPACE DIFFERENTIABLE RENDERING

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

Original | Derivative with respect to sun location

Fig. 1. We introduce **path-space differentiable rendering**, a new theoretical framework to estimate derivatives of radiometric measurements with respect to arbitrary scene parameters (e.g., material properties and object geometries). By directly differentiating full path integrals, we derive the *differential path integral* framework, enabling the design of new unbiased Monte Carlo methods capable of efficiently estimating derivatives in virtual scenes with complex geometry and light transport effects. This example shows a dinning room scene lit by the sun from outside the window. On the right, we show the corresponding derivative image with respect to the vertical location of the sun. (Please use Adobe Acrobat to view the teaser images to see them animated.)

Physics-based differentiable rendering, the estimation of derivatives of radiometric measures with respect to arbitrary scene parameters, has a diverse array of applications from solving analysis-by-synthesis problems to training machine learning pipelines incorporating forward rendering processes. Unfortunately, general-purpose differentiable rendering remains challenging due to the lack of efficient estimators as well as the need to identify and handle complex discontinuities such as visibility boundaries.

In this paper, we show how path integrals can be differentiated with respect to arbitrary differentiable changes of a scene. We provide a detailed theoretical analysis of this process and establish new differentiable rendering formulations based on the resulting differential path integrals. Our path-space differentiable rendering formulation allows the design of new Monte Carlo estimators that offer significantly better efficiency than state of the art methods in handling complex geometric discontinuities and light transport phenomena such as caustics.

We validate our method by comparing our derivative estimates to those generated using the finite-difference method. To demonstrate the effectiveness of our technique, we compare inverse-rendering performance with a few state-of-the-art differentiable rendering methods.

CCS Concepts: • **Computing methodologies → Rendering.**

Additional Key Words and Phrases: Differentiable rendering, path integral, Monte Carlo rendering

ACM Reference Format:
Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. 2020. Path-Space Differentiable Rendering. *ACM Trans. Graph.* 39, 4, Article 143 (July 2020), 19 pages. https://doi.org/10.1145/3386569.3392383

Authors' addresses: Cheng Zhang, University of California, Irvine, chengz20@uci.edu; Bailey Miller, Carnegie Mellon University, baileymark.miller@gmail.com; Kai Yan, University of California, Irvine, kyan8@uci.edu; Ioannis Gkioulekas, Carnegie Mellon University, igkioule@andrew.cmu.edu; Shuang Zhao, University of California, Irvine, shz@ics.uci.edu.

Permission to make digital or hard copies of all or part 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 components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0730-0301/2020/7-ART143 $15.00
https://doi.org/10.1145/3386569.3392383

1 INTRODUCTION

Physics-based light transport simulation, a core research topic in computer graphics since the field's inception, focus on numerically estimating radiometric sensor responses in fully specified virtual scenes. Previous research efforts have led to mature *forward rendering* algorithms that can efficiently and accurately simulate light transport in virtual environments with high complexities.

Differentiable rendering computes the derivatives of radiometric measurements with respect to differential changes of such environments. These techniques can enable, for example: (i) *gradient-based optimization* when solving *inverse-rendering* problems; and (ii) efficient integration of physics-based light transport simulation in *machine learning* and *probabilistic inference* pipelines.

ACM Trans. Graph., Vol. 39, No. 4, Article 143. Publication date: July 2020.

Path-Space Differentiable Rendering

Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, Shuang Zhao

SIGGRAPH 2020

DIFFERENTIAL PATH INTEGRAL

Path integral

$$I = \int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}}) \quad \Rightarrow \quad \frac{\mathrm{d}I}{\mathrm{d}\pi} = \ ?$$

DIFFERENTIAL PATH INTEGRAL

SIGGRAPH THINK BEYOND
2020 S2020.SIGGRAPH.ORG

Path integral

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

Differential path integral (for meshes)

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

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

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

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

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

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

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

and

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

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,

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

and

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

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

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

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

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

Full derivation in the paper [Zhang et al. 2020]

DIFFERENTIAL PATH INTEGRAL

Path integral

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

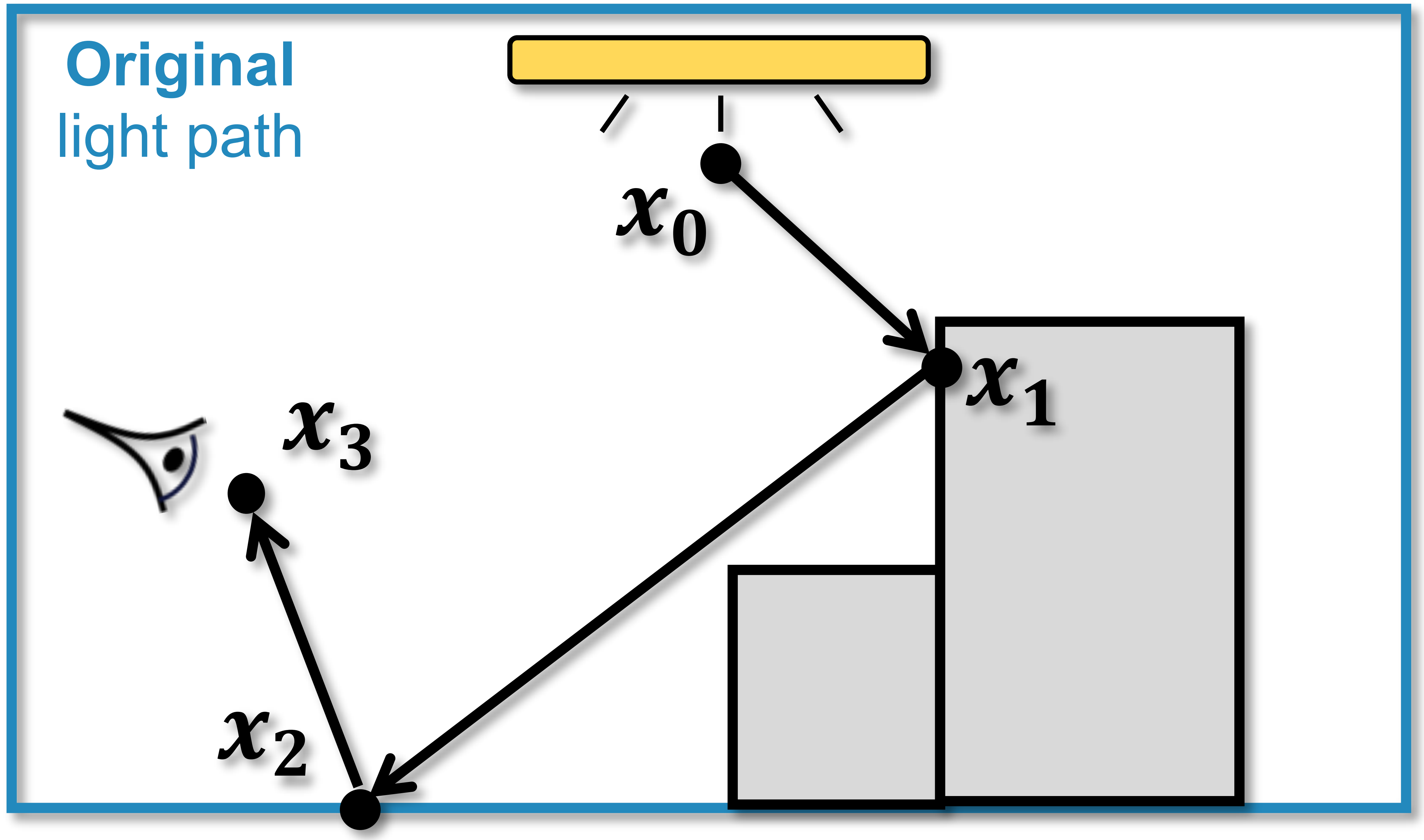
Path space

Differential path integral (for meshes)

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

Path space

Interior integral



DIFFERENTIAL PATH INTEGRAL

Path integral

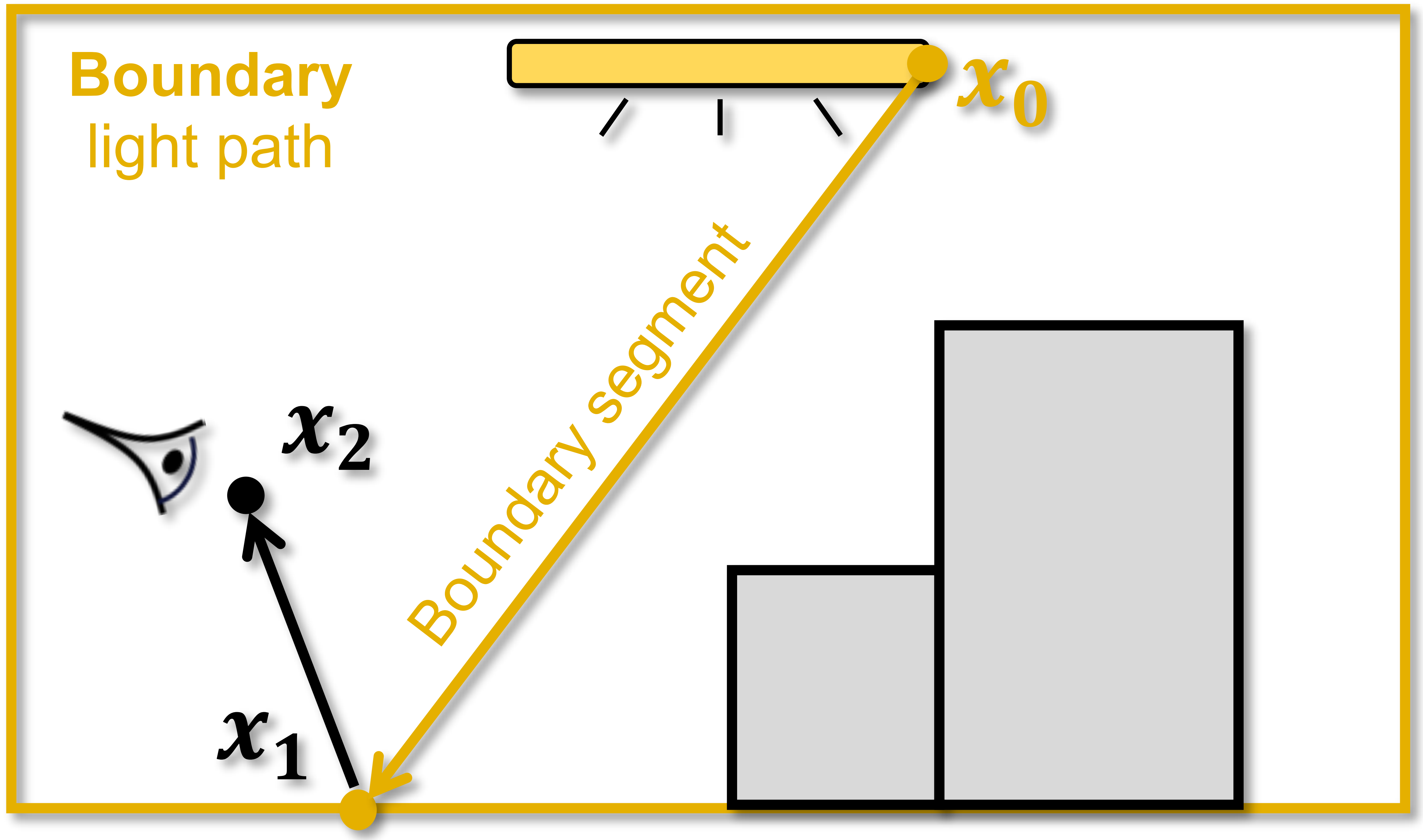
$$I = \int_{\Omega} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}})$$

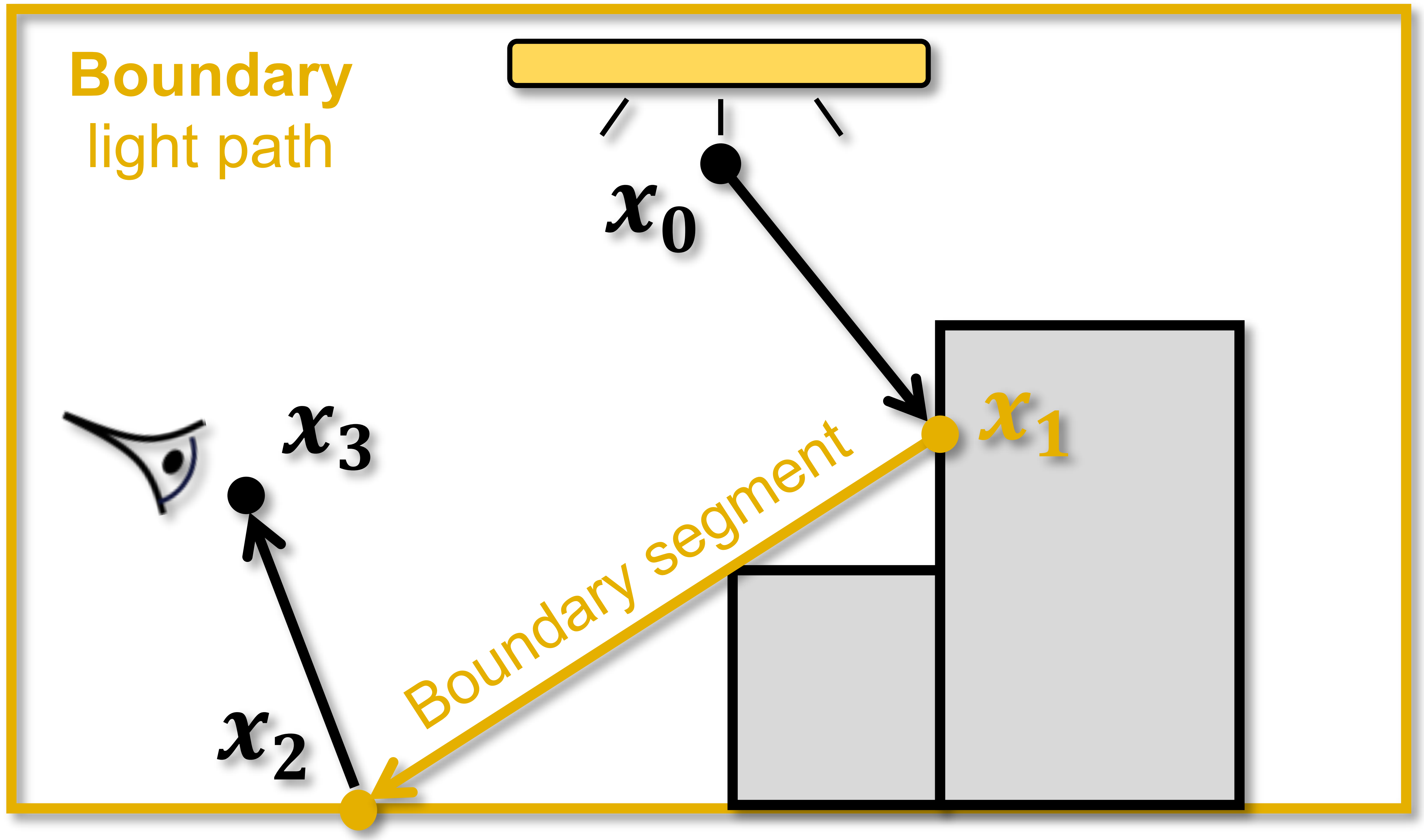
Differential path integral (for meshes)

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\Omega} \frac{\mathrm{d}}{\mathrm{d}\pi} f(\bar{\boldsymbol{x}})\,\mathrm{d}\mu(\bar{\boldsymbol{x}}) + \underbrace{\int_{\partial\Omega} g(\bar{\boldsymbol{x}})\,\mathrm{d}\mu'(\bar{\boldsymbol{x}})}_{\text{Boundary integral}}$$

Boundary path space

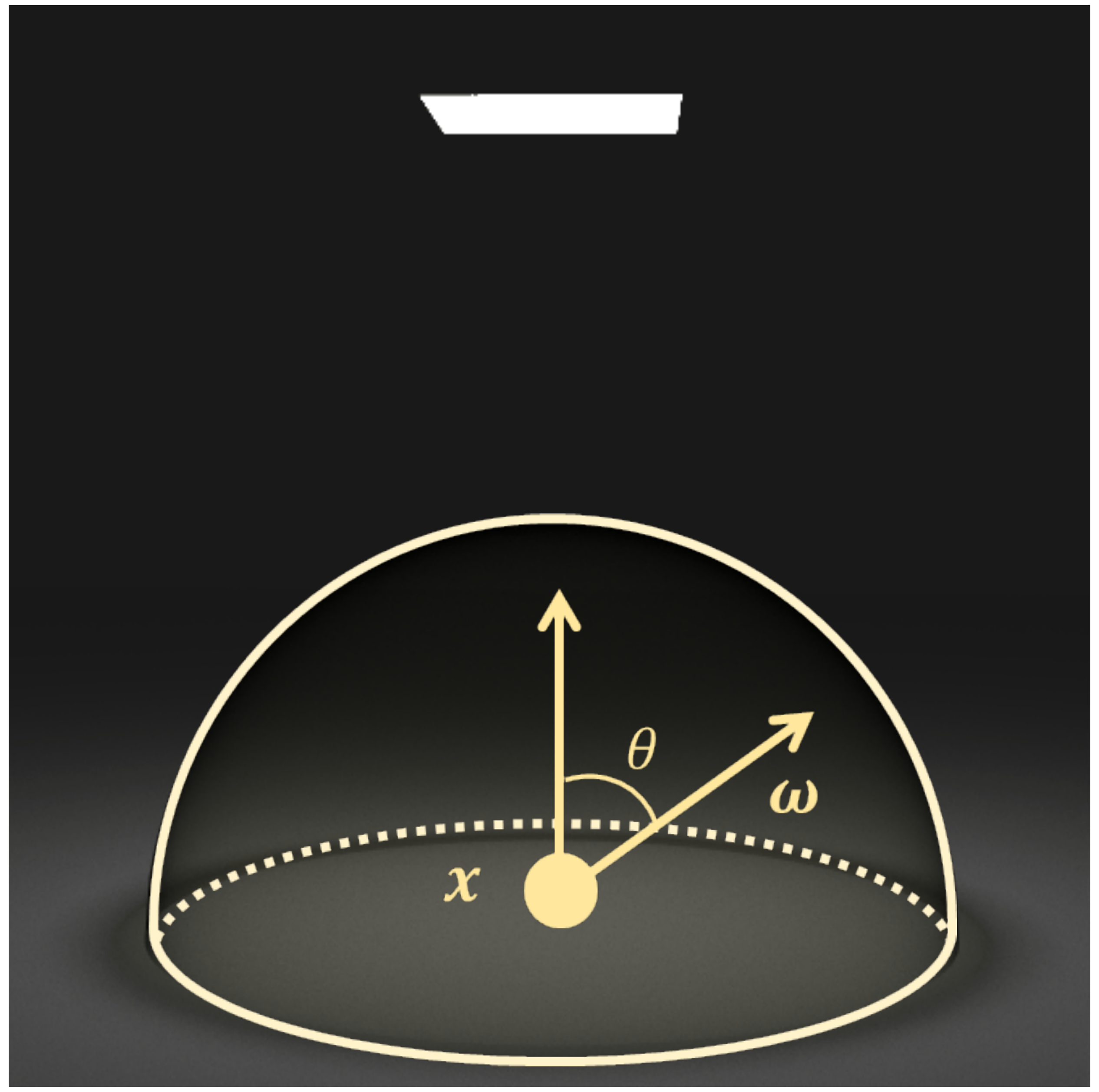
Boundary light path: same as original light path except having exactly one boundary segment



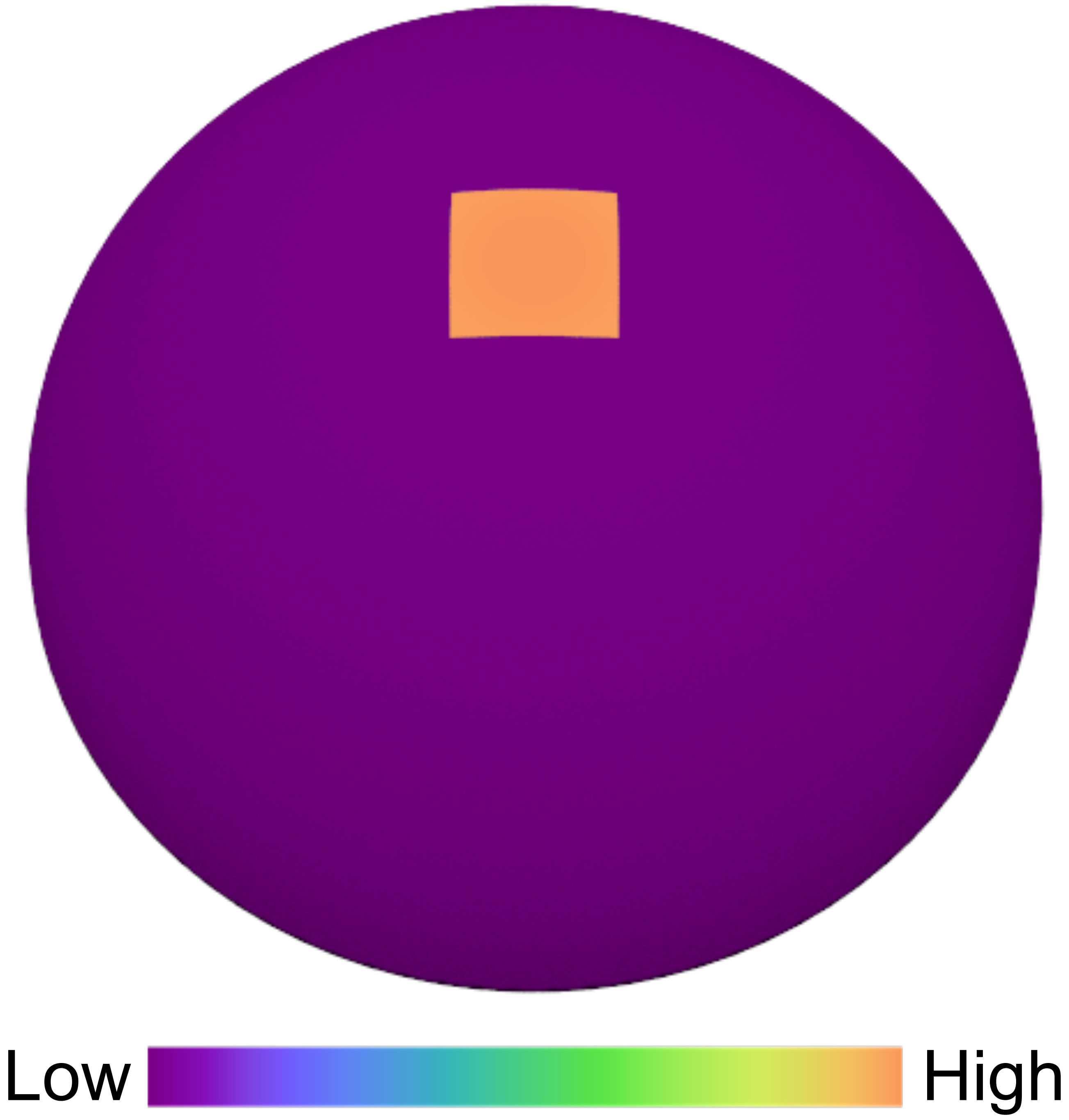


RECAP: DIFFERENTIAL IRRADIANCE

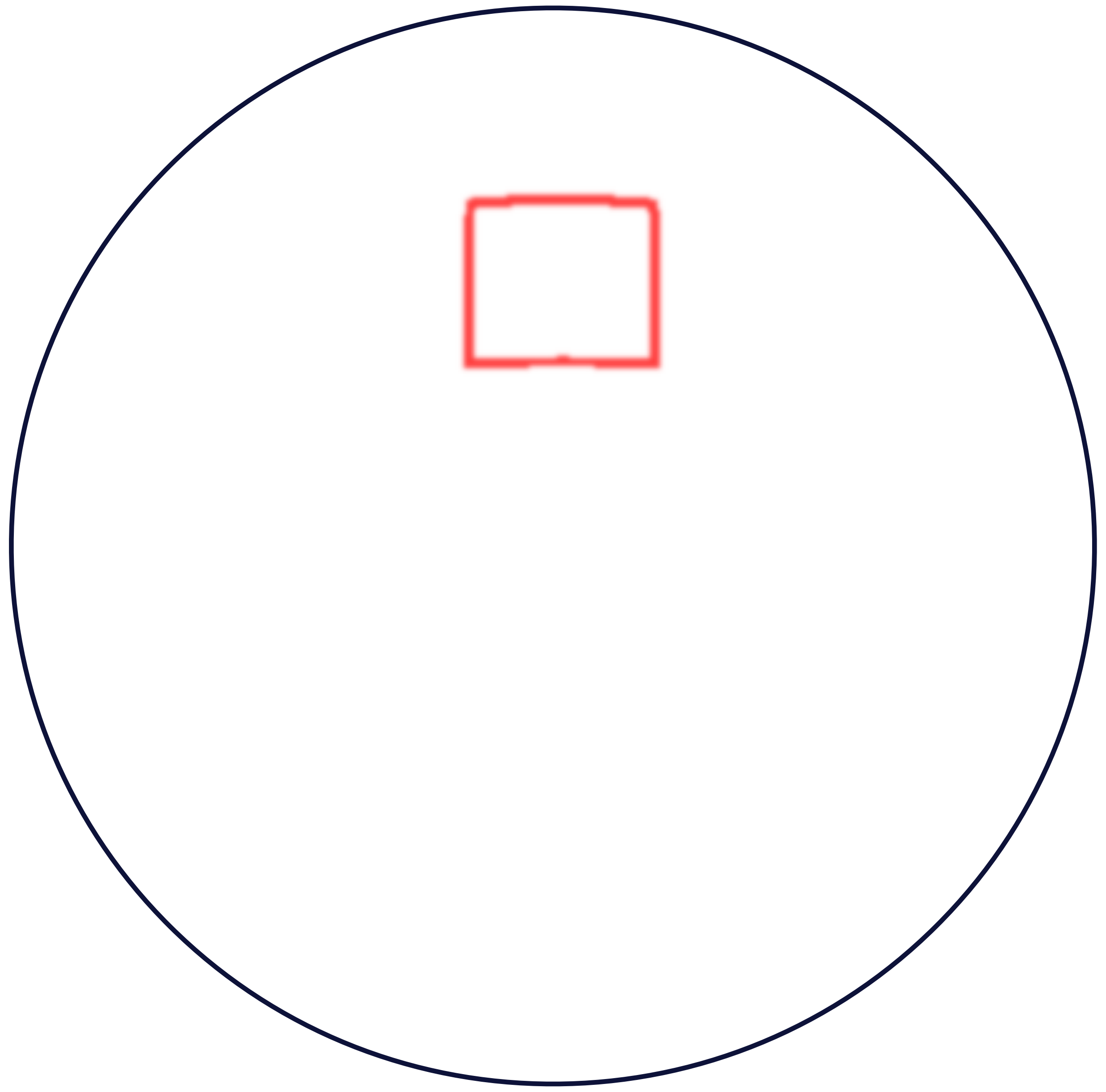
π: emitter size



$f_E(\boldsymbol{\omega})$



$\partial\mathbb{H}^2$

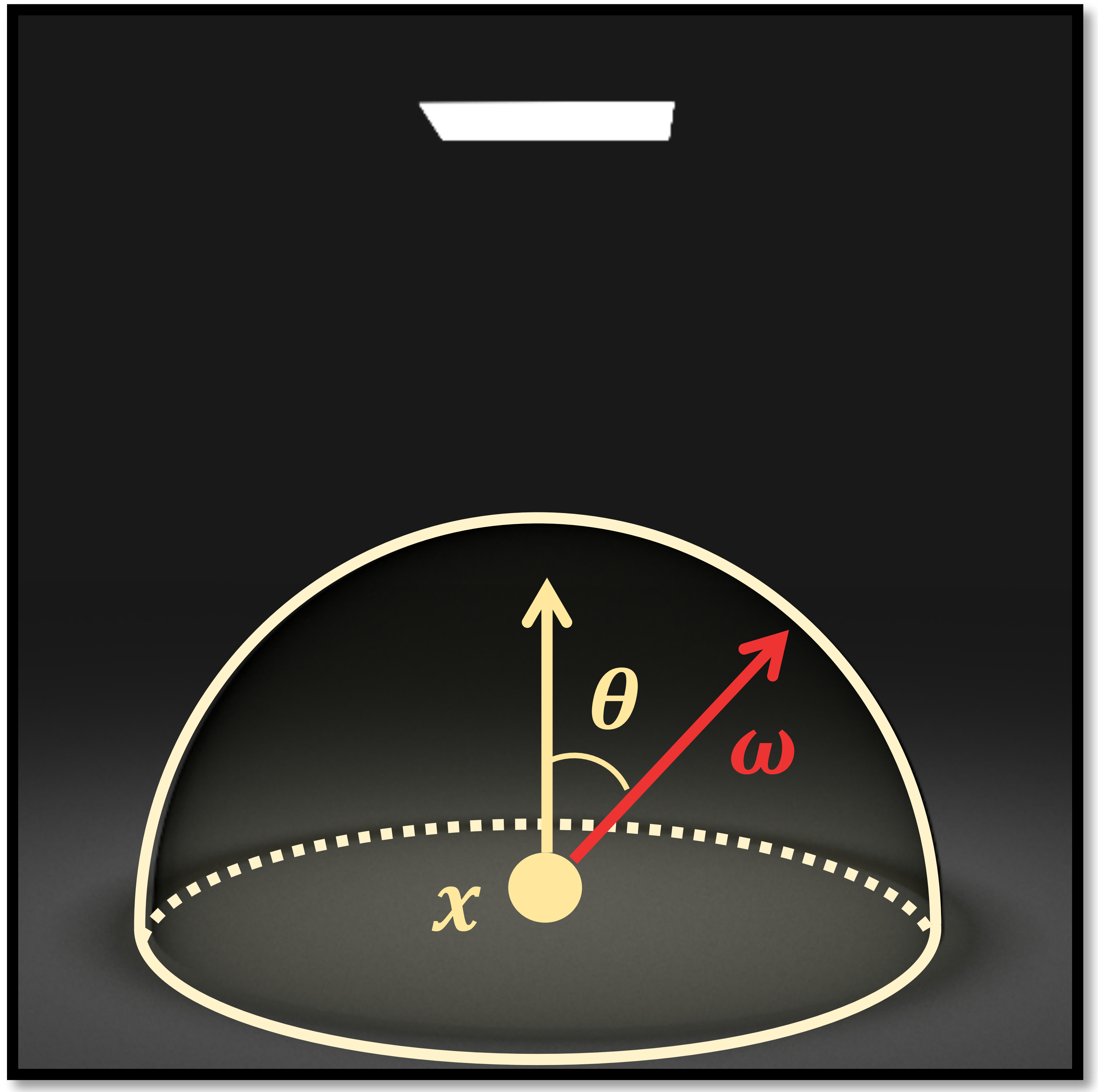


Interior integral $= 0$ Boundary integral

$$E = \int_{\mathbb{H}^2} \overbrace{L_\mathrm{i}(\boldsymbol{\omega}) \cos\theta}^{f_E(\boldsymbol{\omega})} \, \mathrm{d}\sigma(\boldsymbol{\omega}) \xrightarrow{\text{Reynolds}} \frac{\mathrm{d}E}{\mathrm{d}\pi} = \int_{\mathbb{H}^2} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\boldsymbol{\omega}) \, \mathrm{d}\sigma(\boldsymbol{\omega}) + \int_{\partial\mathbb{H}^2} V_{\partial\mathbb{H}^2}(\boldsymbol{\omega}) \, \Delta f_E(\boldsymbol{\omega}) \, \mathrm{d}\ell(\boldsymbol{\omega})$$

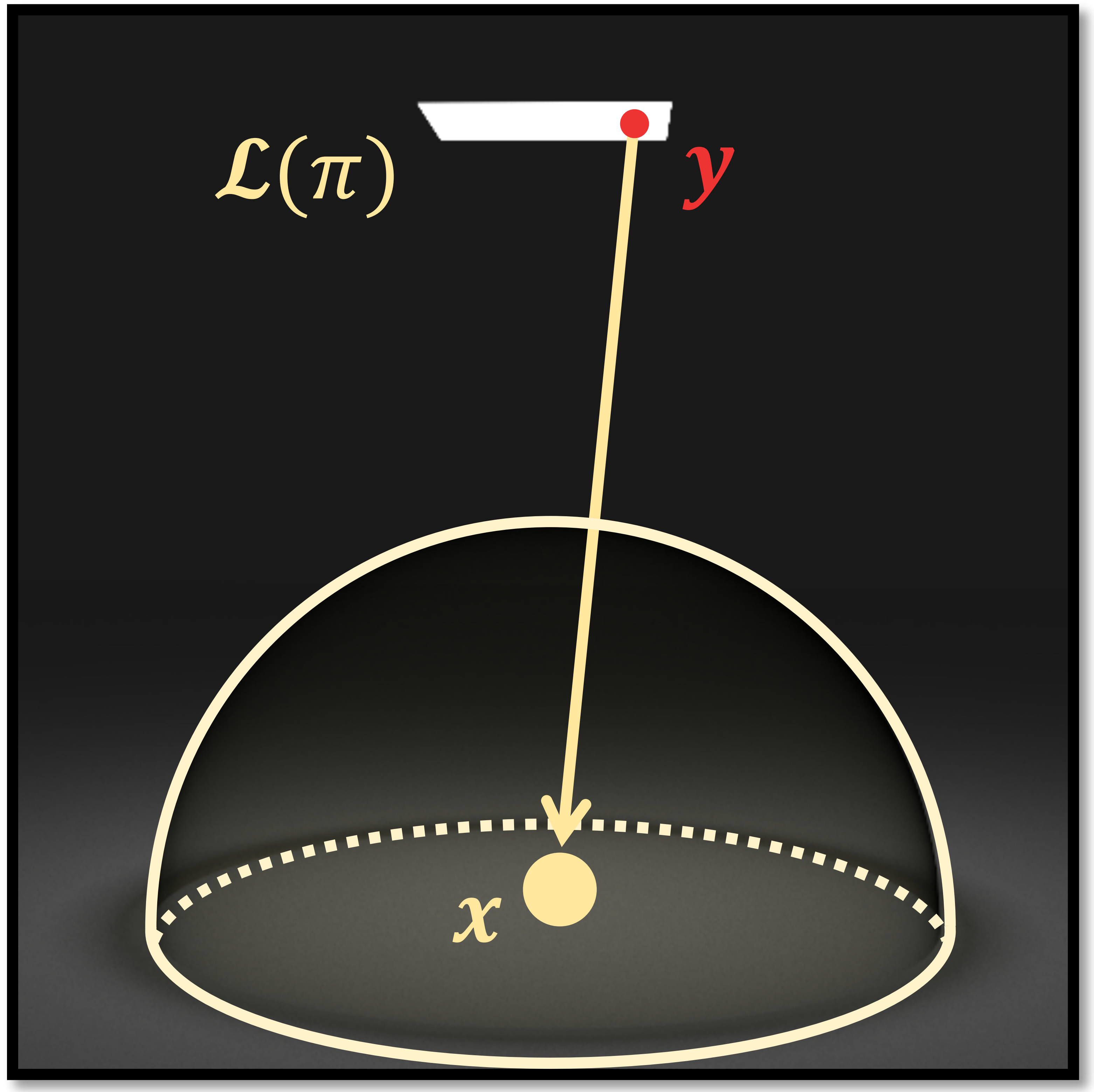
CHANGE OF VARIABLE

Spherical integral



$$E = \int_{\mathbb{H}^2} L_i(\boldsymbol{\omega}) \cos\theta \, \mathrm{d}\sigma(\boldsymbol{\omega})$$

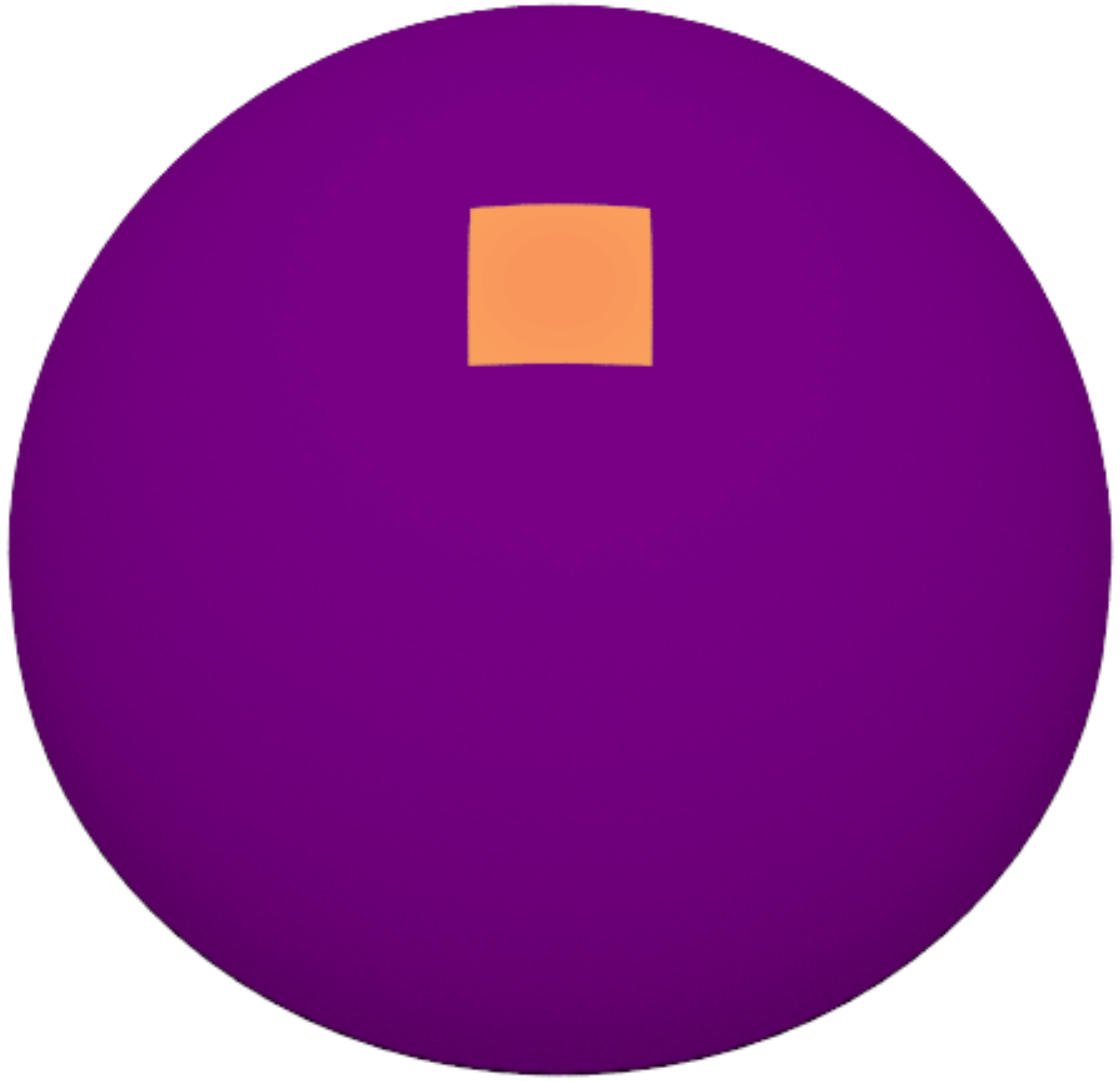
Surface integral



$$E = \int_{\mathscr{L}(\pi)} L_e(\mathbf{y} \to \mathbf{x}) \, G(\mathbf{x} \leftrightarrow \mathbf{y}) \, \mathrm{d}A(\mathbf{y})$$

CHANGE OF VARIABLE

Spherical integral

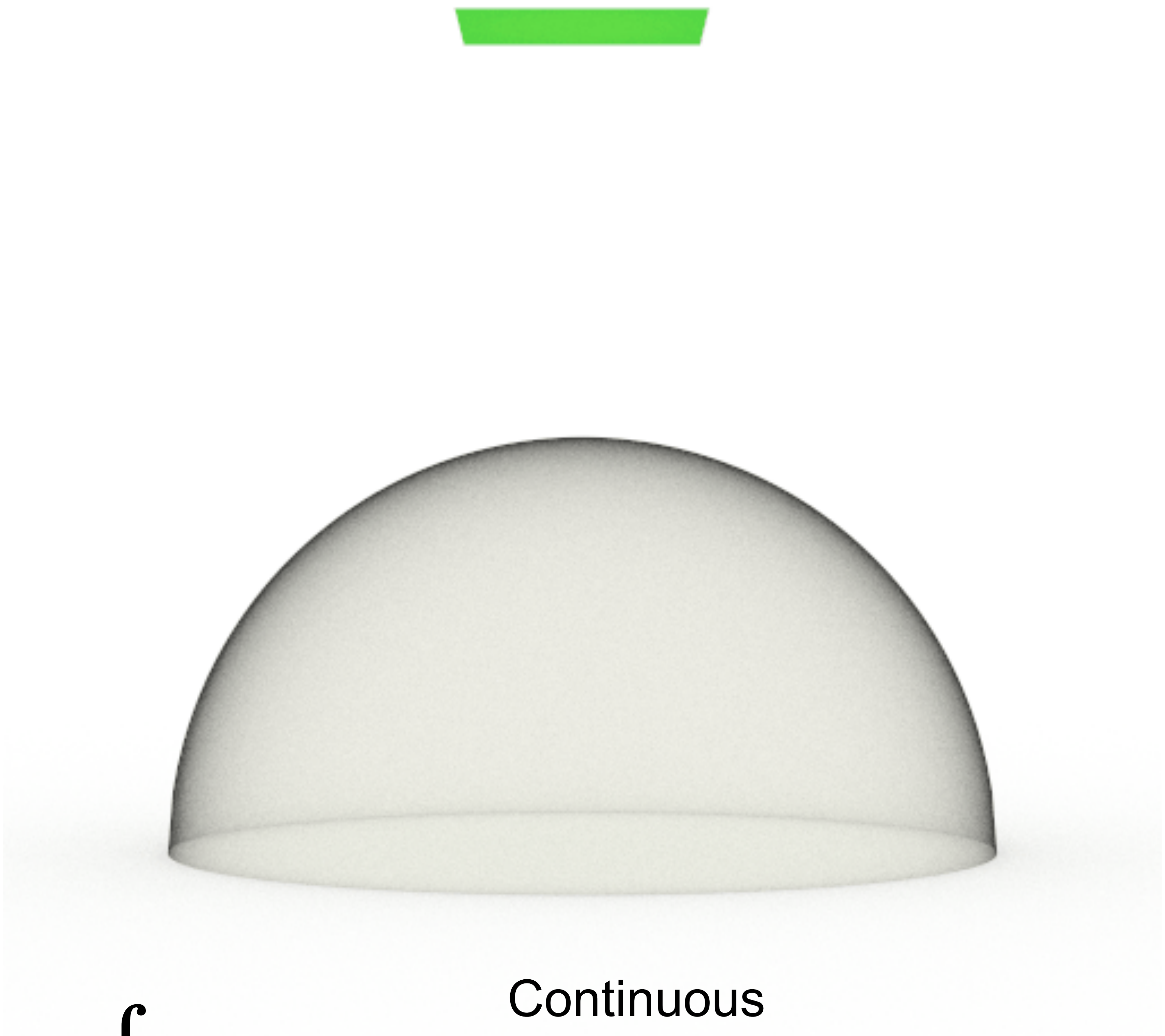


Discontinuous

$$E = \int_{\mathbb{H}^2} \boxed{L_\mathrm{i}(\boldsymbol{\omega}) \, \cos\theta} \, \mathrm{d}\sigma(\boldsymbol{\omega})$$

Constant domain

Surface integral



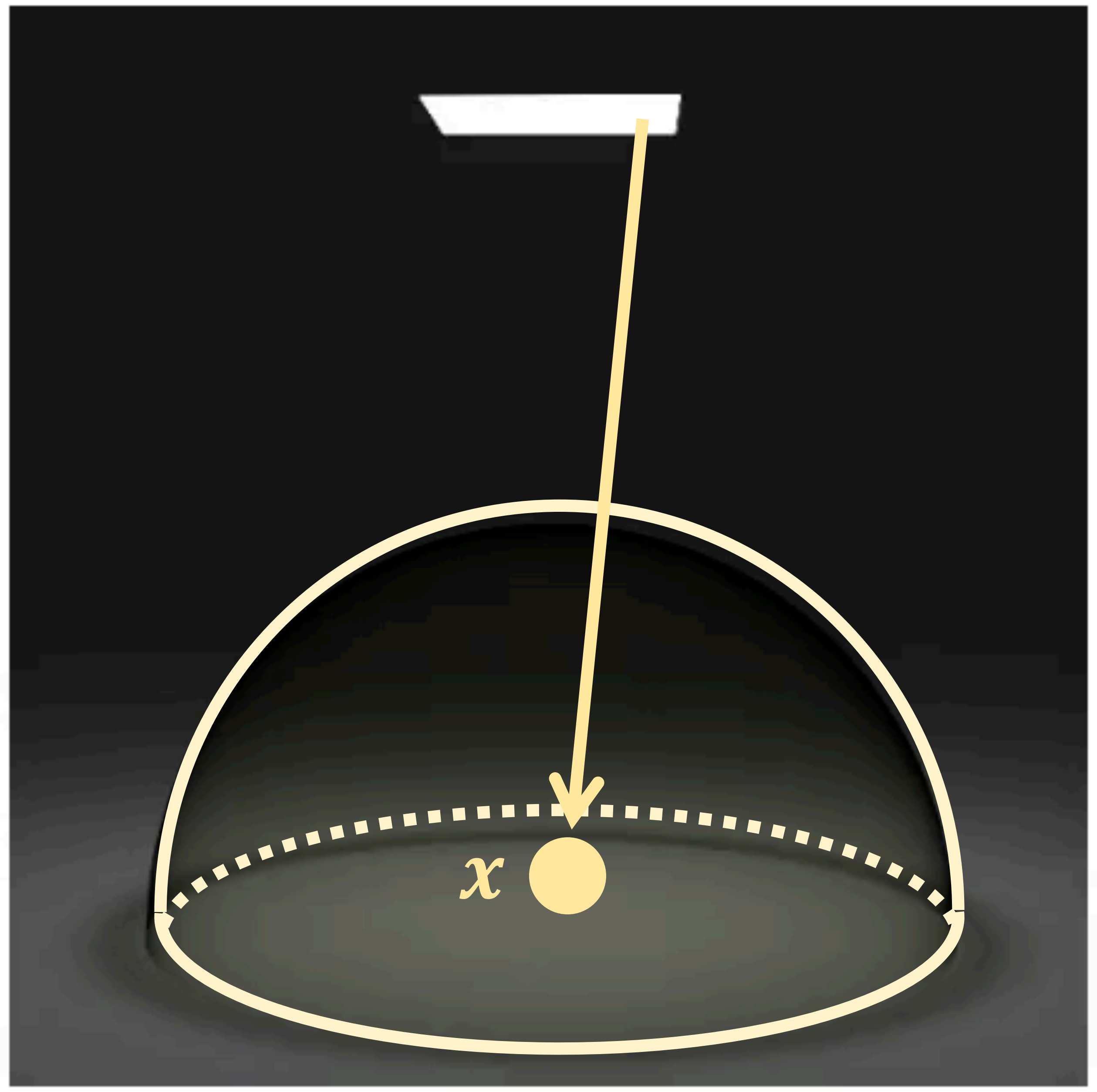
Continuous

$$E = \int_{\mathscr{L}(\pi)} \boxed{L_\mathrm{e}(\mathbf{y} \to \mathbf{x}) \, G(\mathbf{x} \leftrightarrow \mathbf{y})} \, \mathrm{d}A(\mathbf{y})$$

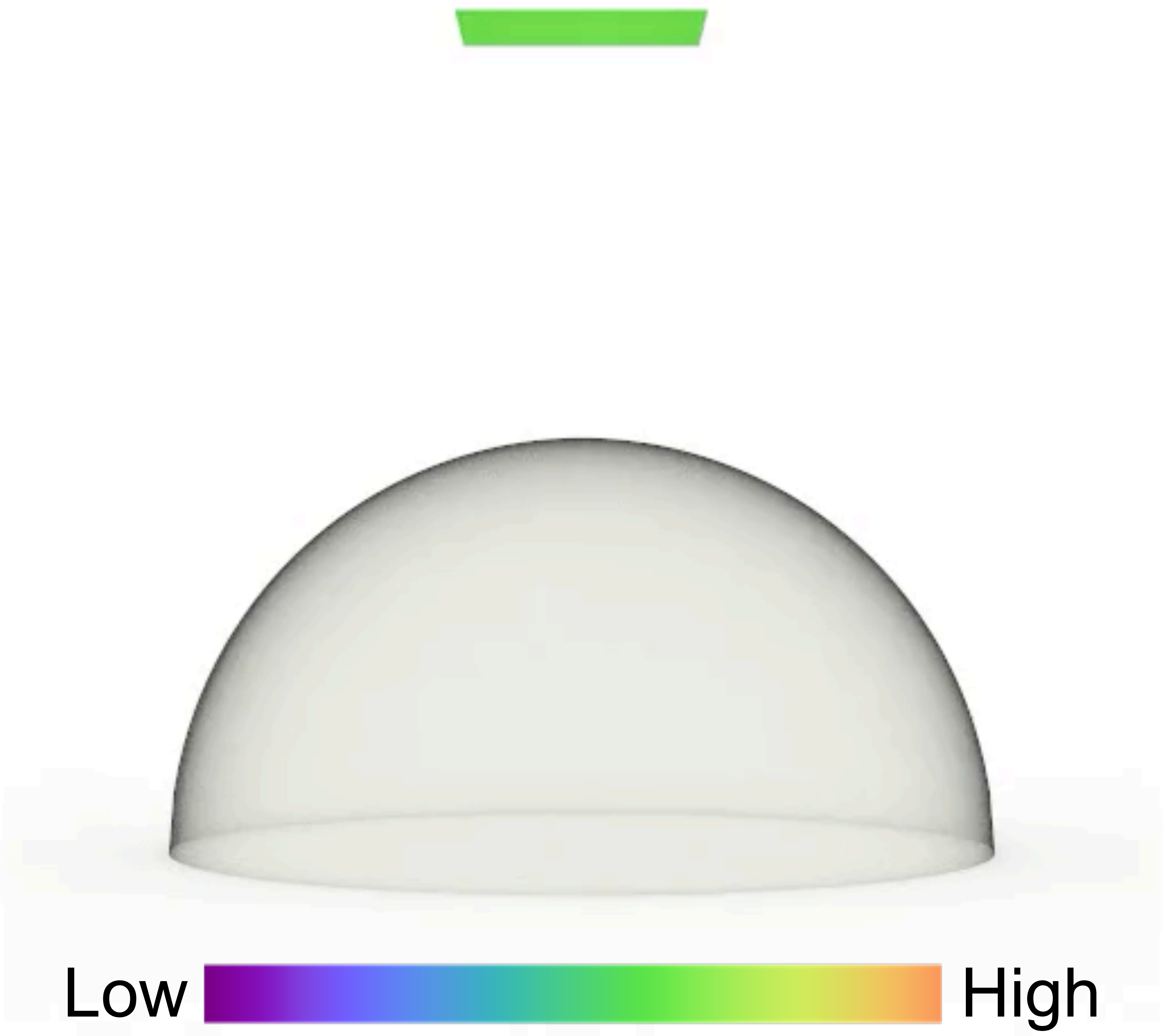
Evolving domain

DIFFERENTIAL IRRADIANCE

π: emitter size



$f_E(\mathbf{y})$



$\partial\mathscr{L}(\pi)$

when $\mathscr{L}(\pi)$ is flat

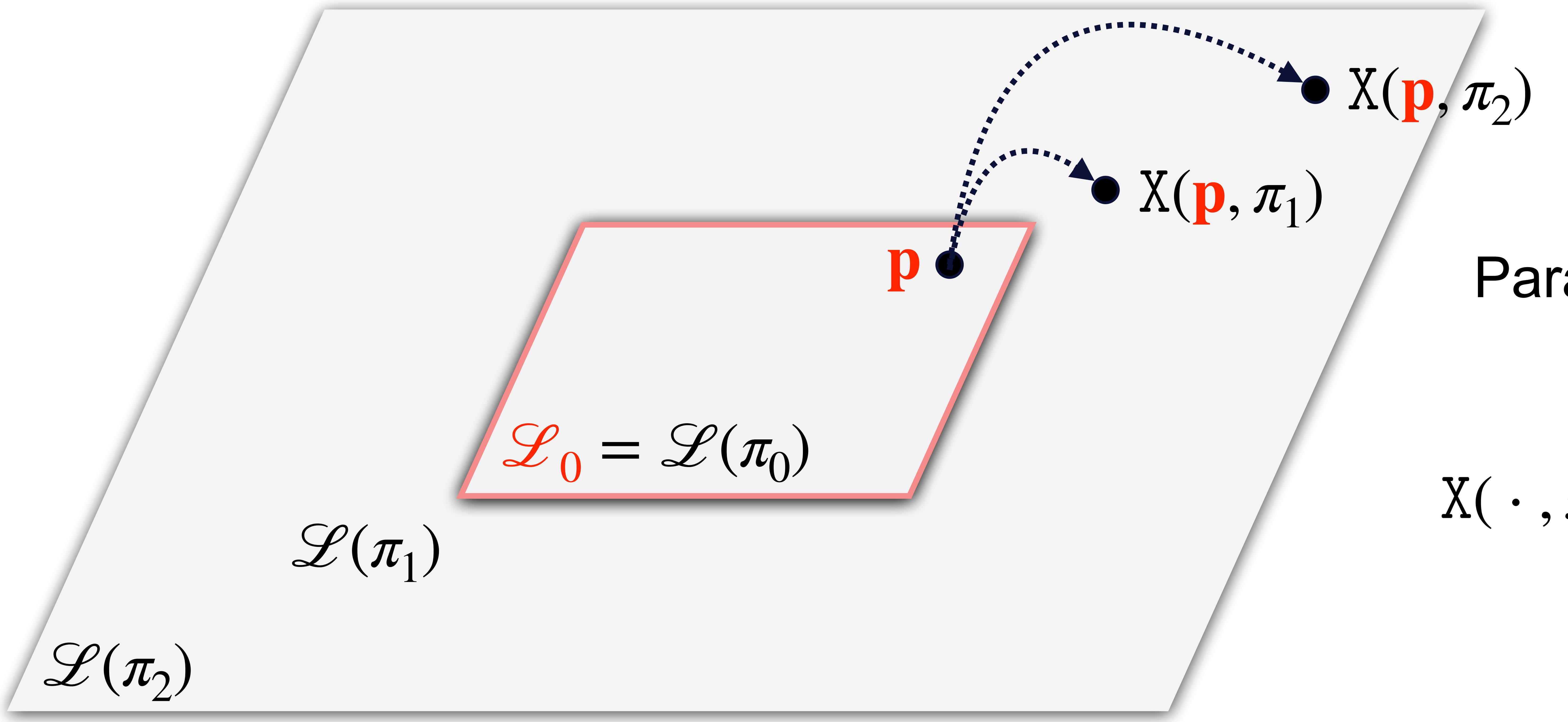
Interior integral

Boundary integral $\neq 0$

$$E = \int_{\mathscr{L}(\pi)} \overbrace{L_e(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y})}^{f_E(\mathbf{y})}\, \mathrm{d}A(\mathbf{y}) \Rightarrow \frac{\mathrm{d}E}{\mathrm{d}\pi} = \boxed{\int_{\mathscr{L}(\pi)} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\mathbf{y})\, \mathrm{d}A(\mathbf{y})} + \boxed{\int_{\partial\mathscr{L}(\pi)} V_{\partial\mathscr{L}(\pi)}(\mathbf{y})\, \Delta f_E(\mathbf{y})\, \mathrm{d}\ell(\mathbf{y})}$$

REPARAMETERIZATION

Before reparameterization $E = \int_{\mathscr{L}(\pi)} L_{\mathrm{e}}(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y})\, \mathrm{d}A(\mathbf{y})$



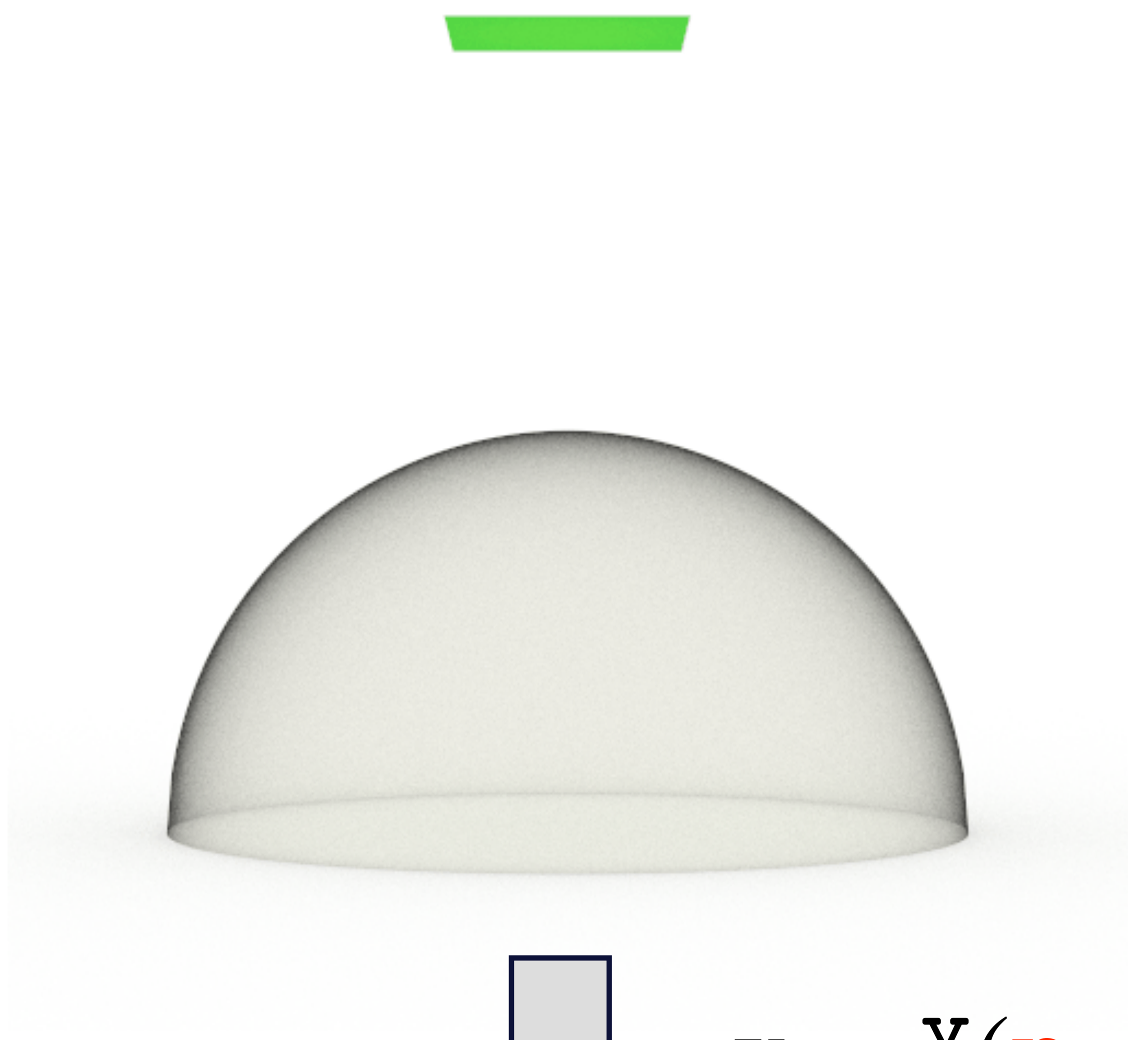
Parameterize $\mathscr{L}(\pi)$ using fixed $\mathscr{L}_0$:

$$\mathbf{y} = \mathbb{X}(\mathbf{p}, \pi)$$

$\mathbb{X}(\,\cdot\,, \pi)$ is **one-to-one** and **continuous**

Reparameterization with $\mathbf{y} = \mathbb{X}(\mathbf{p}, \pi)$ $E = \int_{\mathscr{L}_0} L_{\mathrm{e}}(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y}) \left| \frac{\mathrm{d}A(\mathbf{y})}{\mathrm{d}A(\mathbf{p})} \right| \mathrm{d}A(\mathbf{p})$

REPARAMETERIZATION



$\mathbf{y} = \mathrm{X}(\mathbf{p}, \pi)$

$$E = \int_{\mathscr{L}(\pi)} \overbrace{L_\mathrm{e}(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y})}^{f_E(\mathbf{y})}\, \mathrm{d}A(\mathbf{y})$$

Interior integral = 0 | Boundary integral ≠ 0

$$\frac{\mathrm{d}E}{\mathrm{d}\pi} = \int_{\mathscr{L}(\pi)} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\mathbf{y})\, \mathrm{d}A(\mathbf{y}) + \int_{\partial\mathscr{L}(\pi)} V_{\partial\mathscr{L}(\pi)}(\mathbf{y})\, \Delta f_E(\mathbf{y})\, \mathrm{d}\ell(\mathbf{y})$$

π-dependent

$$E = \int_{\mathscr{L}_0} \overbrace{L_\mathrm{e}(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y}) \left| \frac{\mathrm{d}A(\mathbf{y})}{\mathrm{d}A(\mathbf{p})} \right|}^{f_E(\mathbf{p})}\, \mathrm{d}A(\mathbf{p})$$

Interior integral ≠ 0 | Boundary integral = 0

$$\frac{\mathrm{d}E}{\mathrm{d}\pi} = \int_{\mathscr{L}_0} \frac{\mathrm{d}f_E}{\mathrm{d}\pi}(\mathbf{p})\, \mathrm{d}A(\mathbf{p}) + \int_{\partial\mathscr{L}_0} V_{\partial\mathscr{L}_0}(\mathbf{p})\, \Delta f_E(\mathbf{p})\, \mathrm{d}\ell(\mathbf{p})$$

REPARAMETERIZATION

Reparameterization for irradiance

$$E = \int_{\mathscr{L}(\pi)} L_e(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y})\, \mathrm{d}A(\mathbf{y}) \xrightarrow{\mathbf{y} = \mathbb{X}(\mathbf{p}, \pi)} E = \int_{\mathscr{L}_0} L_e(\mathbf{y} \to \mathbf{x})\, G(\mathbf{x} \leftrightarrow \mathbf{y}) \left| \frac{\mathrm{d}A(\mathbf{y})}{\mathrm{d}A(\mathbf{p})} \right| \mathrm{d}A(\mathbf{p})$$

fixed surface

Reparameterization for path integral

Spatial form

$$I = \int_{\mathbf{\Omega}(\pi)} f(\bar{\mathbf{x}})\, \mathrm{d}\mu(\bar{\mathbf{x}}) \xrightarrow{\bar{\mathbf{x}} = \mathbb{X}(\bar{\mathbf{p}}, \pi)} I = \int_{\mathbf{\Omega}_0} f(\bar{\mathbf{x}}) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| \mathrm{d}\mu(\bar{\mathbf{p}})$$

Material form

$$\left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| = \prod_i \left| \frac{\mathrm{d}A(\mathbf{x}_i)}{\mathrm{d}A(\mathbf{p}_i)} \right|$$

fixed path space

SIGGRAPH 2020 THINK BEYOND S2020.SIGGRAPH.ORG

Path integrals

Spatial form
$$I = \int_{\Omega(\pi)} f(\bar{\mathbf{x}})\,\mathrm{d}\mu(\bar{\mathbf{x}}) \xrightarrow{\bar{\mathbf{x}} = \mathbb{X}(\bar{\mathbf{p}}, \pi)} I = \int_{\Omega_0} f(\bar{\mathbf{x}}) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| \mathrm{d}\mu(\bar{\mathbf{p}})$$
Material form

Differential path integrals

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

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\Omega_0} \frac{\mathrm{d}}{\mathrm{d}\pi} \left(f(\bar{\mathbf{x}}) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| \right) \mathrm{d}\mu(\bar{\mathbf{p}}) + \int_{\partial\Omega_0} g(\bar{\mathbf{p}})\,\mathrm{d}\mu'(\bar{\mathbf{p}})$$

Pro: no global parameterization required
Con: more types of discontinuities

Pro: fewer types of discontinuities
Con: requires global parameterization $\mathbb{X}$

DIFFERENTIAL PATH INTEGRAL

Spatial form

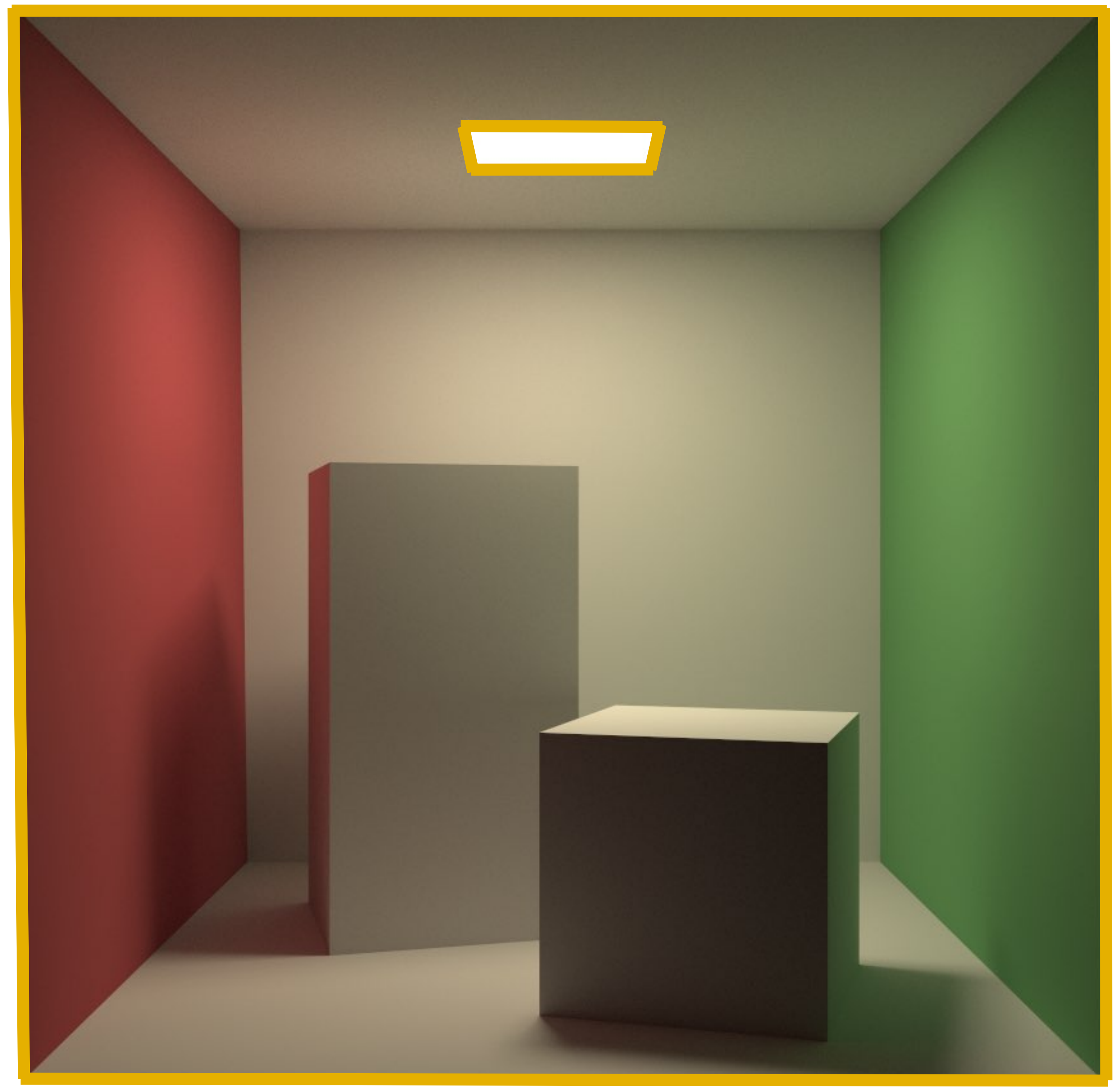
Differential path integrals

Material form

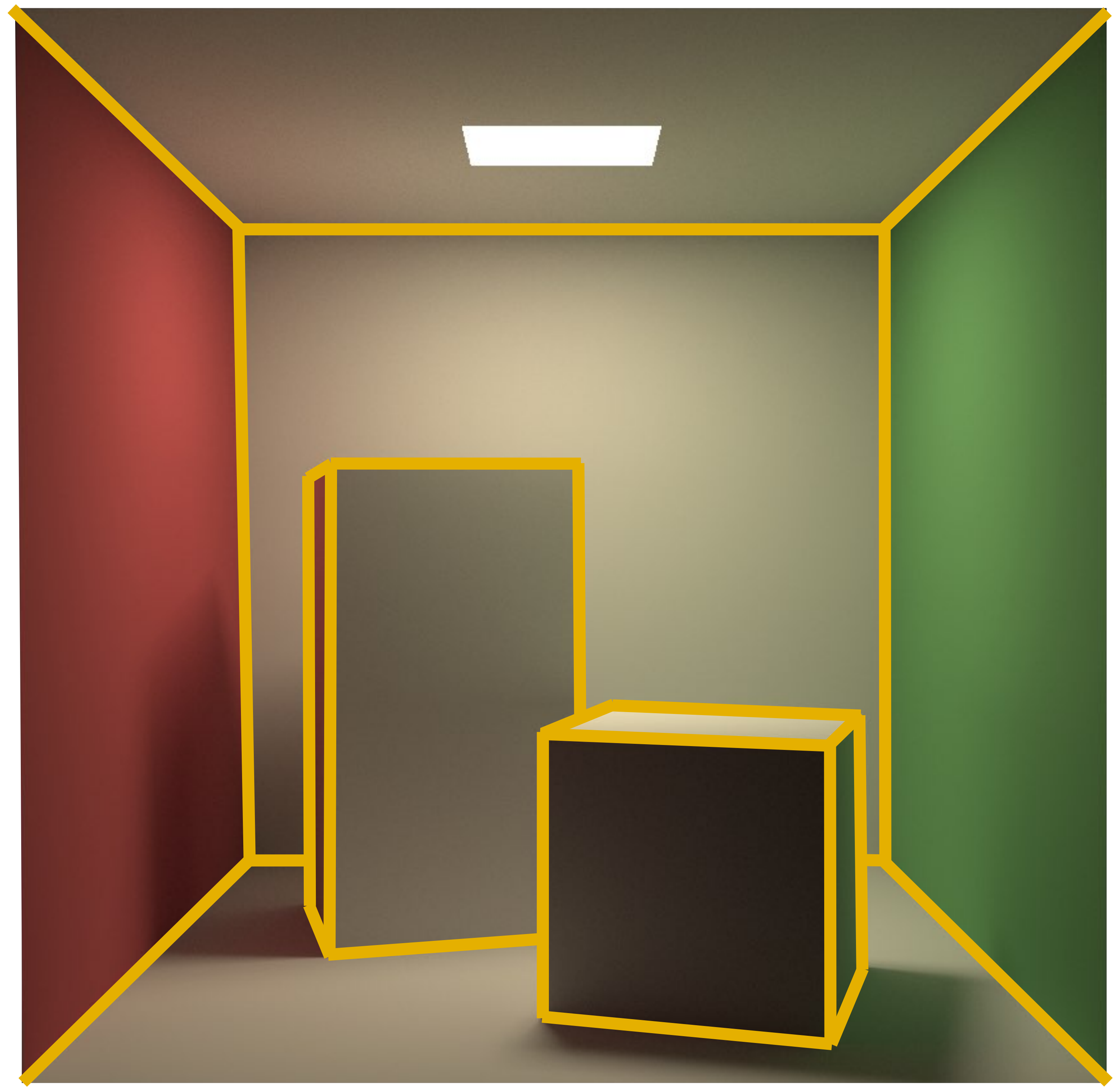
$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\Omega(\pi)} \frac{\mathrm{d}f}{\mathrm{d}\pi}(\bar{\mathbf{x}})\,\mathrm{d}\mu(\bar{\mathbf{x}}) + \int_{\partial\Omega(\pi)} g(\bar{\mathbf{x}})\,\mathrm{d}\mu'(\bar{\mathbf{x}})$$

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\Omega_0} \frac{\mathrm{d}}{\mathrm{d}\pi}\left(f(\bar{\mathbf{x}}) \left|\frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})}\right|\right)\mathrm{d}\mu(\bar{\mathbf{p}}) + \int_{\partial\Omega_0} g(\bar{\mathbf{p}})\,\mathrm{d}\mu'(\bar{\mathbf{p}})$$

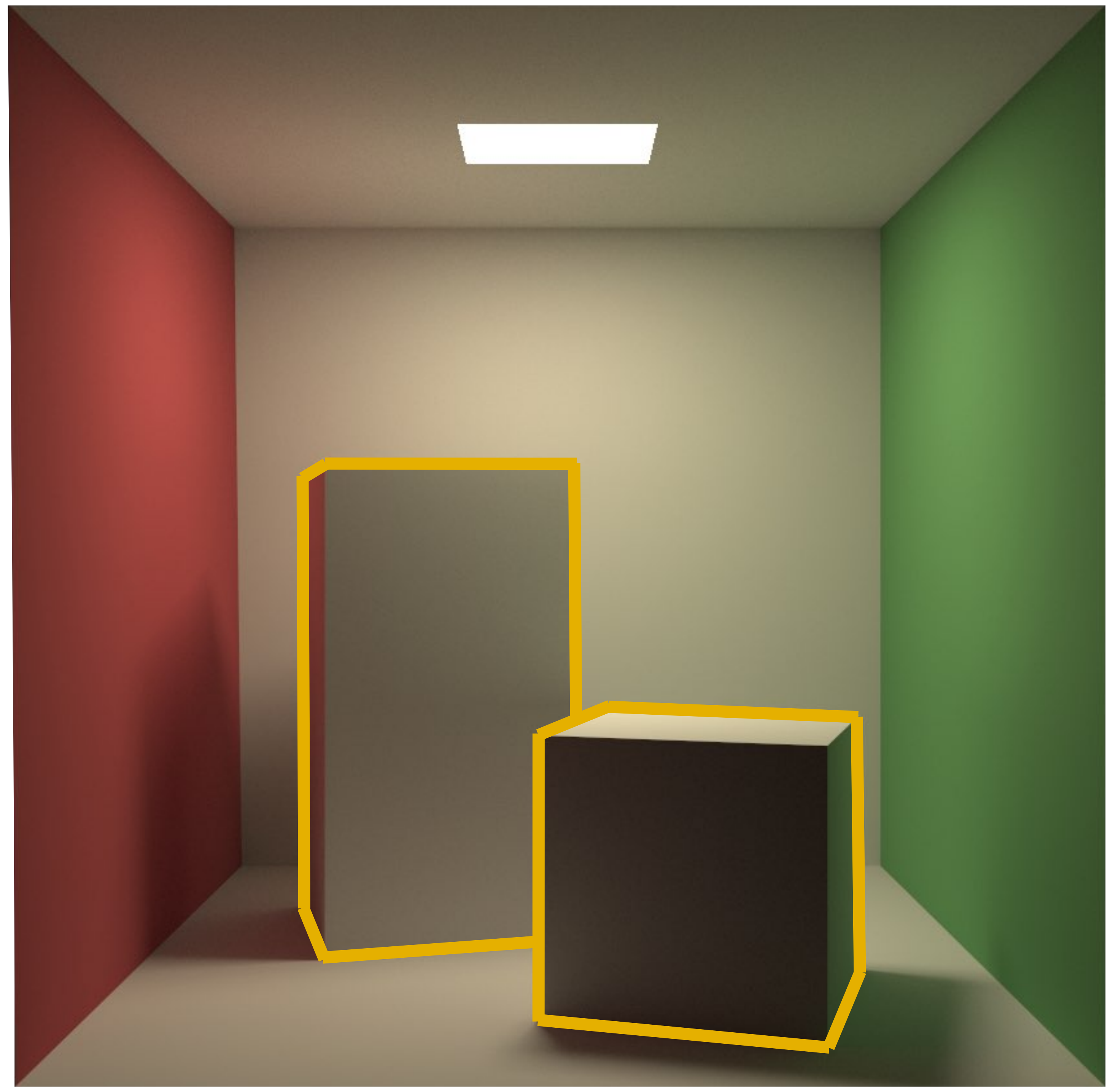
Boundary edges



Sharp edges



Silhouette edges

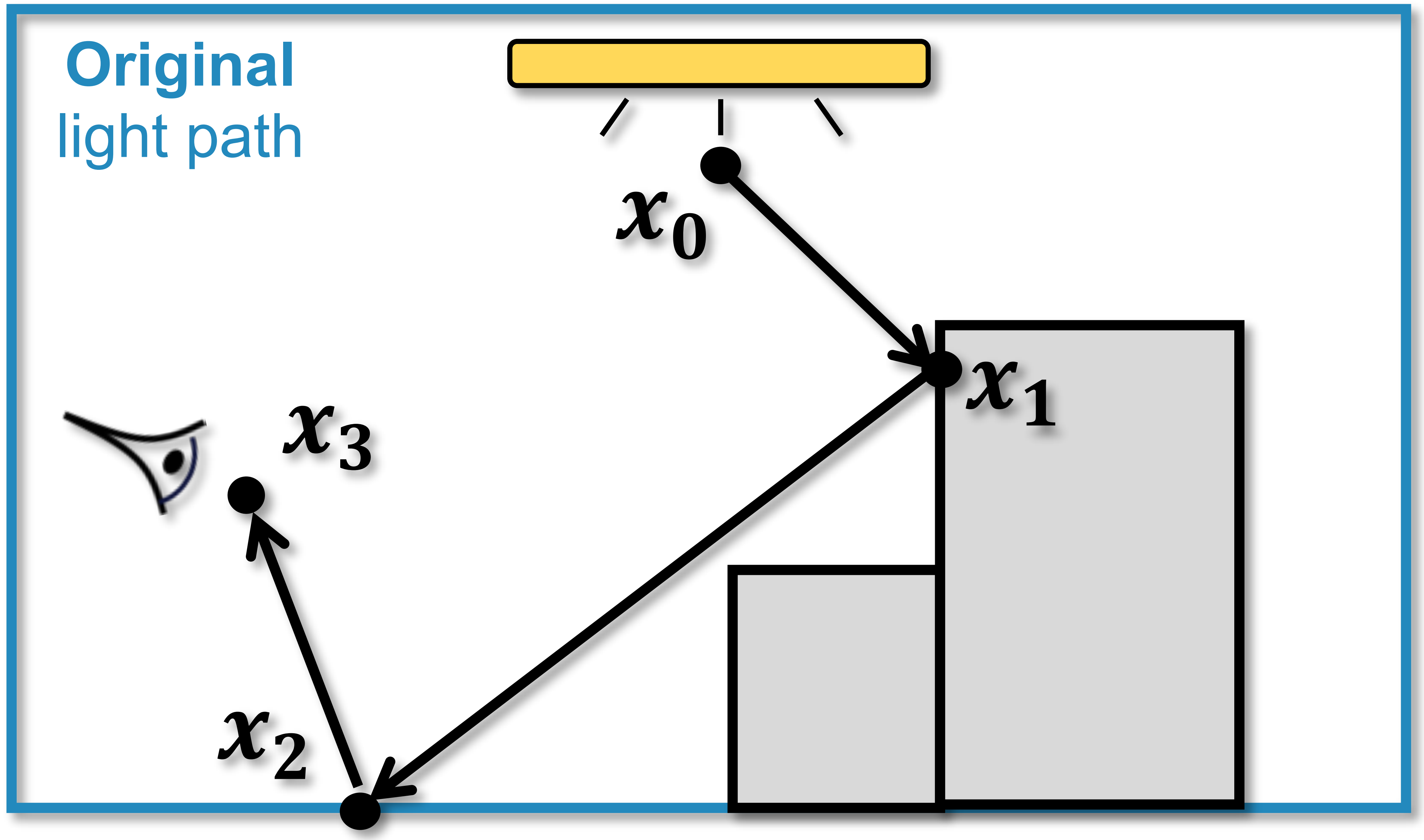


PATH-SPACE MONTE CARLO ESTIMATION

(material-form)
Differential path integral

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \underbrace{\int_{\Omega_0} \frac{\mathrm{d}}{\mathrm{d}\pi}\left(f(\bar{\mathbf{x}}) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| \right) \mathrm{d}\mu(\bar{\mathbf{p}})}_{\text{Interior integral}} + \underbrace{\int_{\partial\Omega_0} g(\bar{\mathbf{p}})\, \mathrm{d}\mu'(\bar{\mathbf{p}})}_{\text{Boundary integral}}$$

Estimated **separately**



Can be estimated using identical path sampling strategies as forward rendering

- Unidirectional path tracing
- Bidirectional path tracing
- …

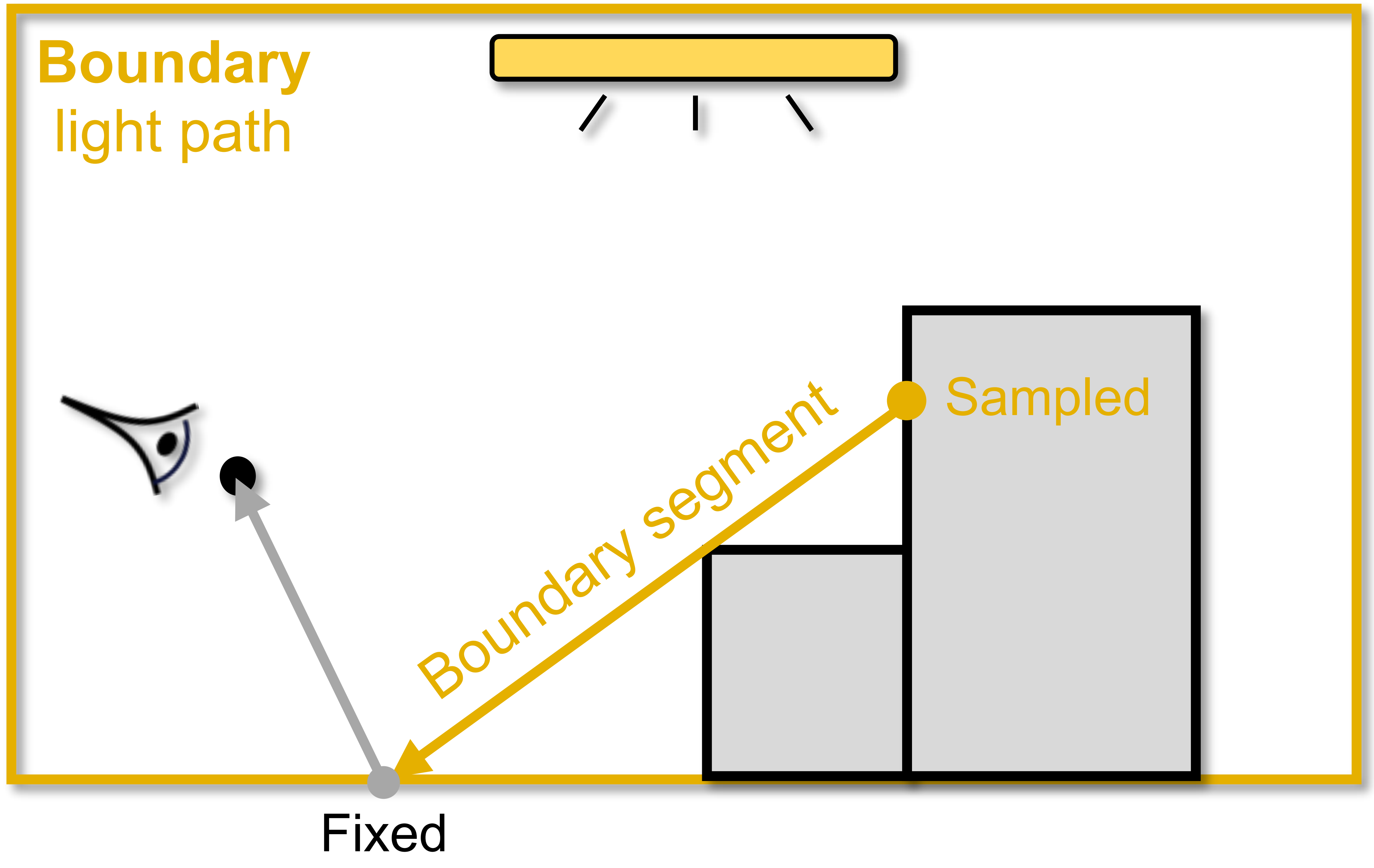
ESTIMATION OF BOUNDARY INTEGRALS

(material-form)
Differential path integral

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\Omega_0} \frac{\mathrm{d}}{\mathrm{d}\pi}\left(f(\bar{\mathbf{x}}) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| \right) \mathrm{d}\mu(\bar{\mathbf{p}}) + \underbrace{\int_{\partial\Omega_0} g(\bar{\mathbf{p}})\, \mathrm{d}\mu'(\bar{\mathbf{p}})}_{\text{Boundary integral}}$$

Unidirectional sampling:

- Construct the boundary path from the eye
- Draw the boundary segment by fixing one endpoint and sampling the other
- Problems
  – Requires expensive silhouette detection



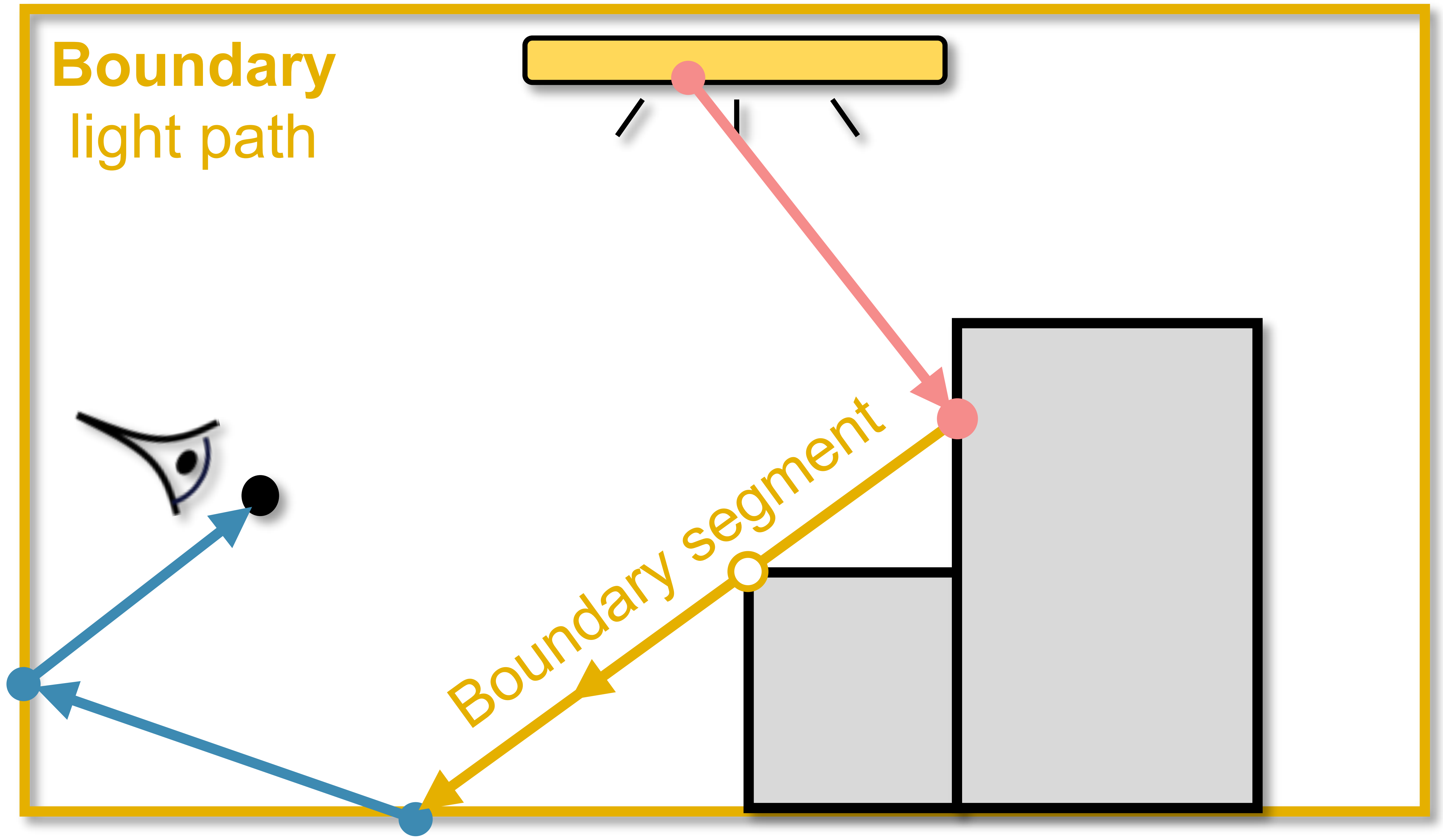
ESTIMATION OF BOUNDARY INTEGRALS

(material-form)
Differential path integral

$$\frac{\mathrm{d}I}{\mathrm{d}\pi} = \int_{\Omega_0} \frac{\mathrm{d}}{\mathrm{d}\pi}\left(f(\bar{\mathbf{x}}) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right| \right) \mathrm{d}\mu(\bar{\mathbf{p}}) + \underbrace{\int_{\partial\Omega_0} g(\bar{\mathbf{p}})\, \mathrm{d}\mu'(\bar{\mathbf{p}})}_{\text{Boundary integral}}$$

Multi-directional sampling:

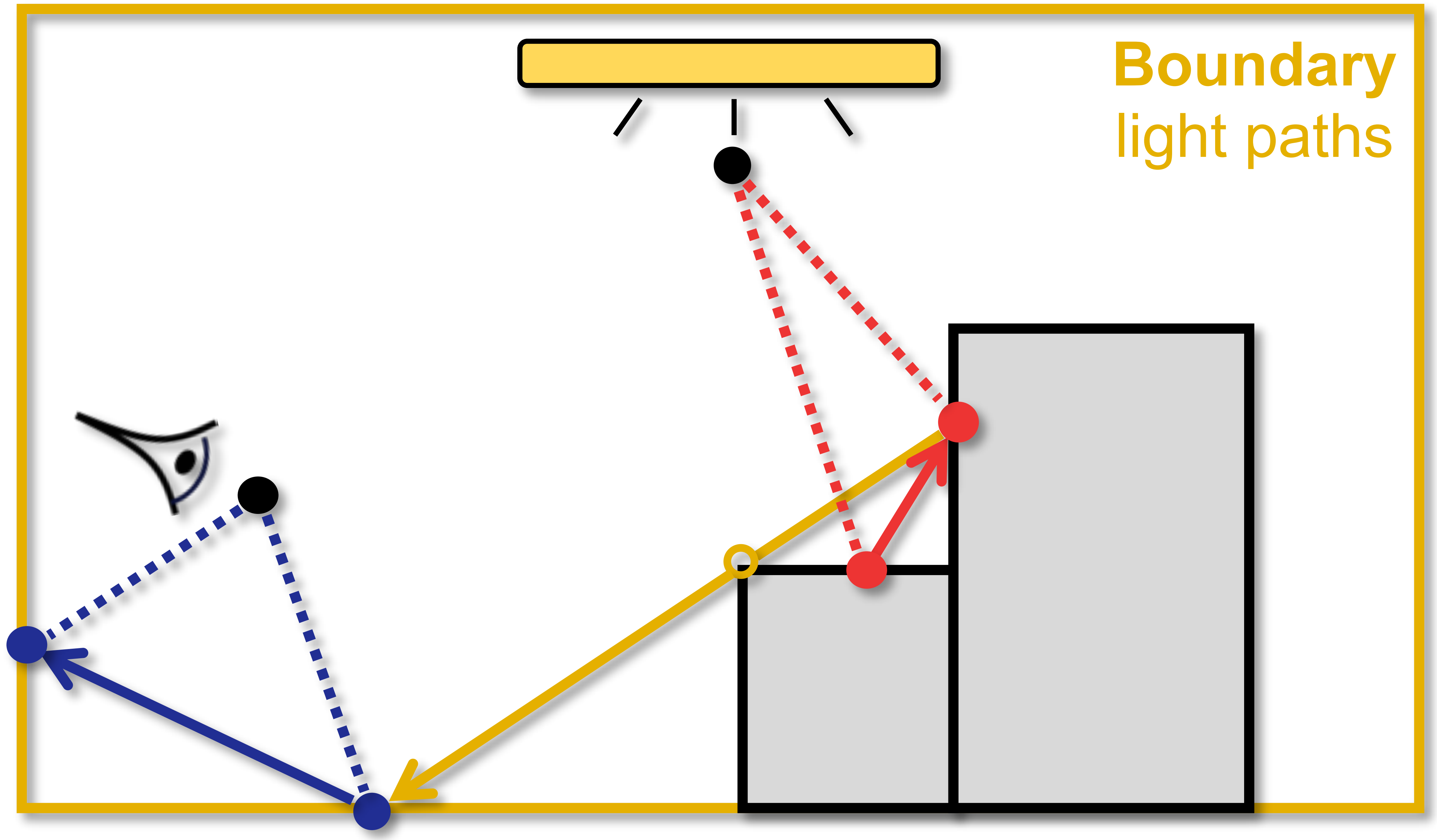
- Construct the boundary segment from the middle
- Construct source and sensor subpaths
- To further improve efficiency
  - Next-event estimation
  - Importance sampling boundary segments



TWO PATH-SPACE ESTIMATORS

Unidirectional estimator

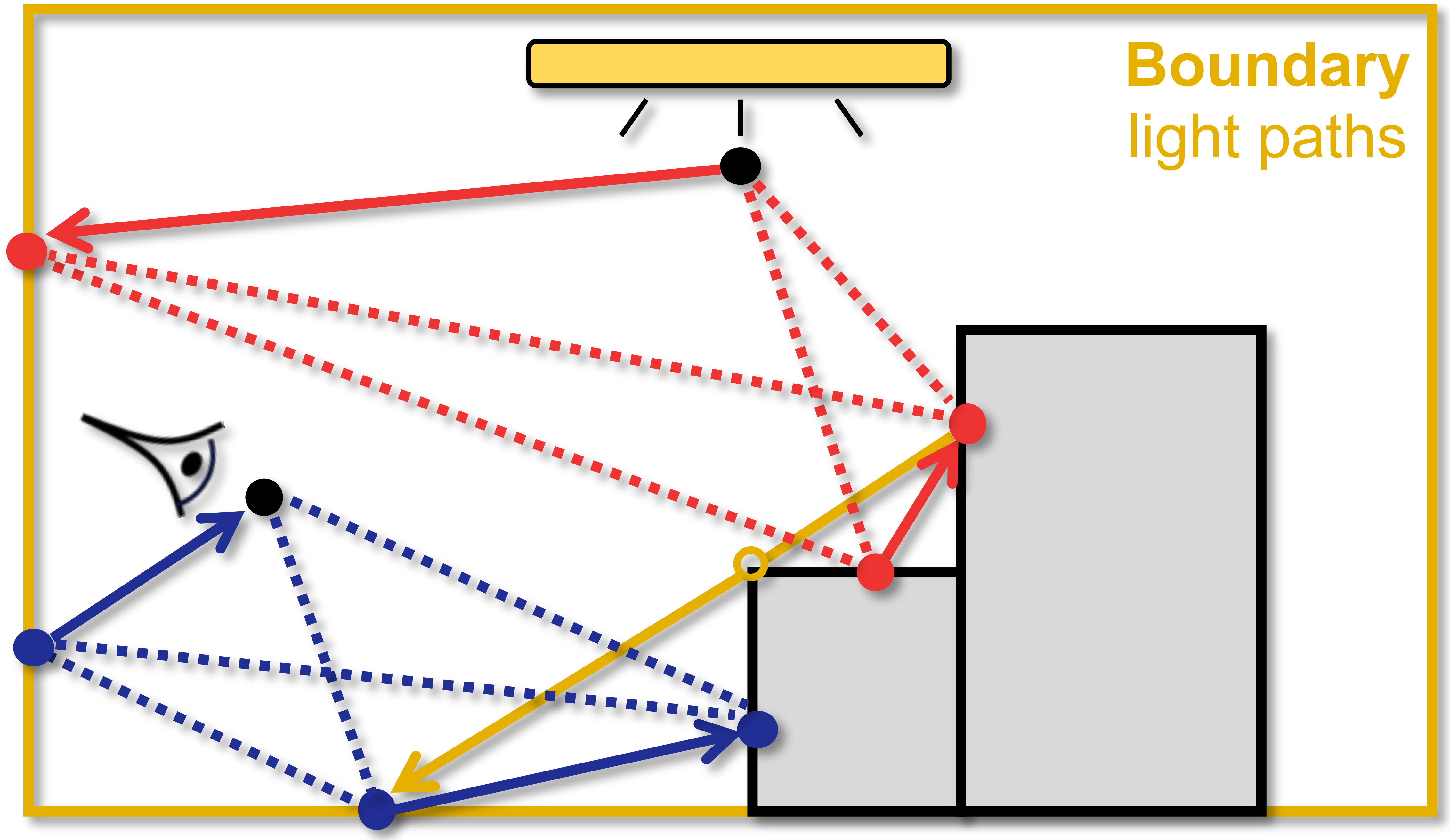
- Interior: **unidirectional** path tracing
- Boundary: **unidirectional** sampling of subpaths



Unidirectional path tracing + NEE

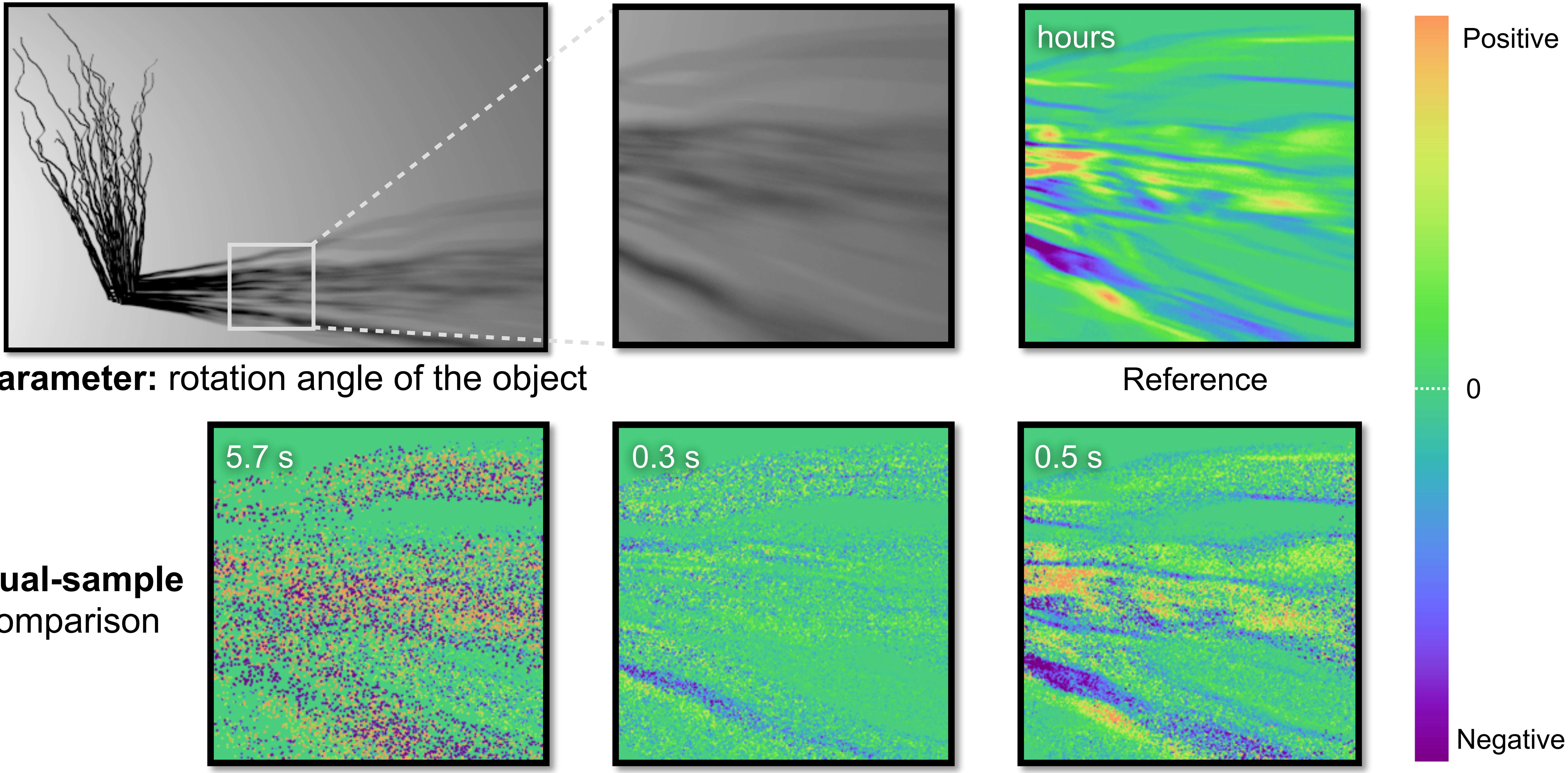
Bidirectional estimator

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



Bidirectional path tracing

RESULT: COMPLEX GEOMETRY & MOTION



Parameter: rotation angle of the object

Reference

Equal-sample comparison

Path tracing w/ edge sampling [Li et al. 2018, Zhang et al. 2019]

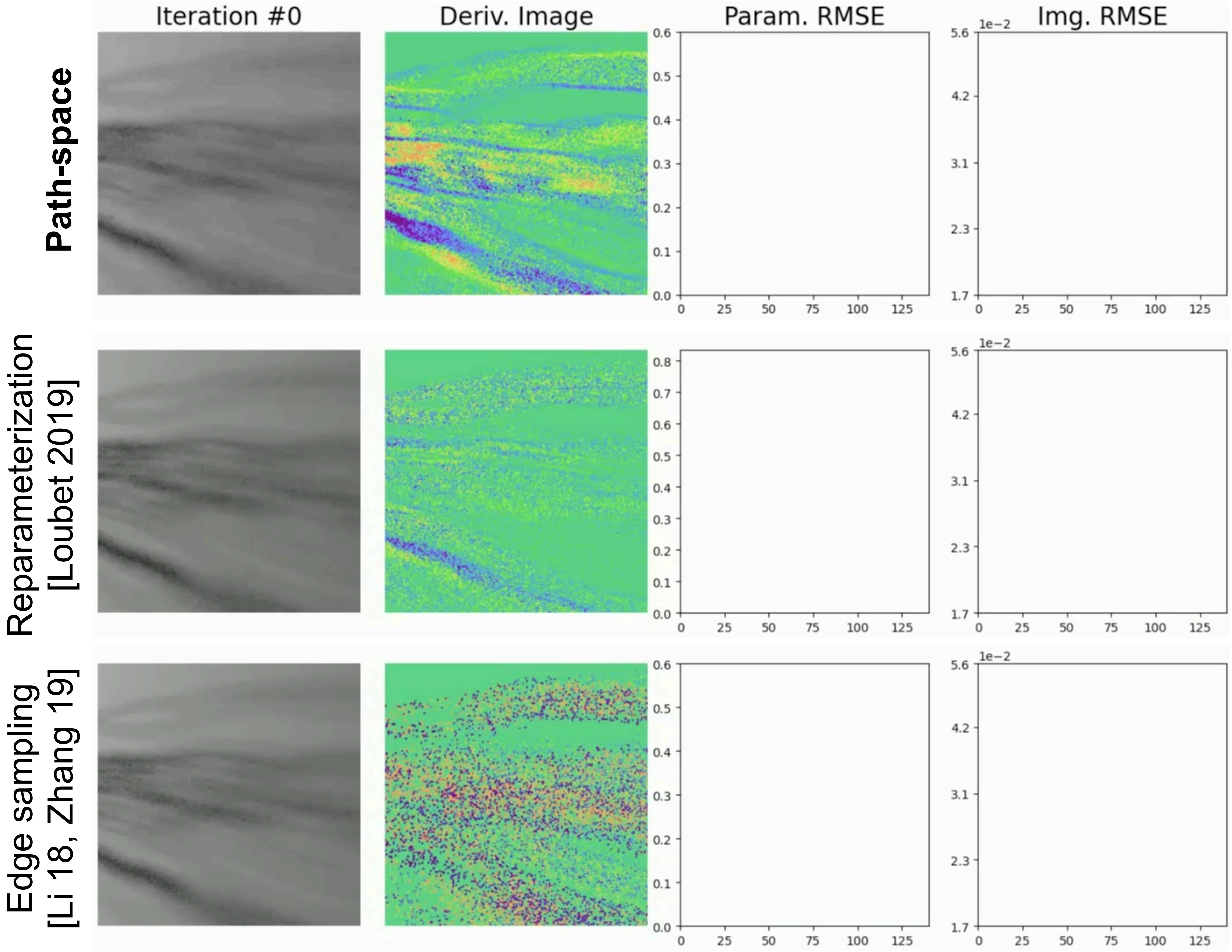
Reparameterization [Loubet et al. 2019]

Path-space, unidir. [Zhang et al. 2020]

RESULT: COMPLEX LIGHT-TRANSPORT EFFECT

Target image

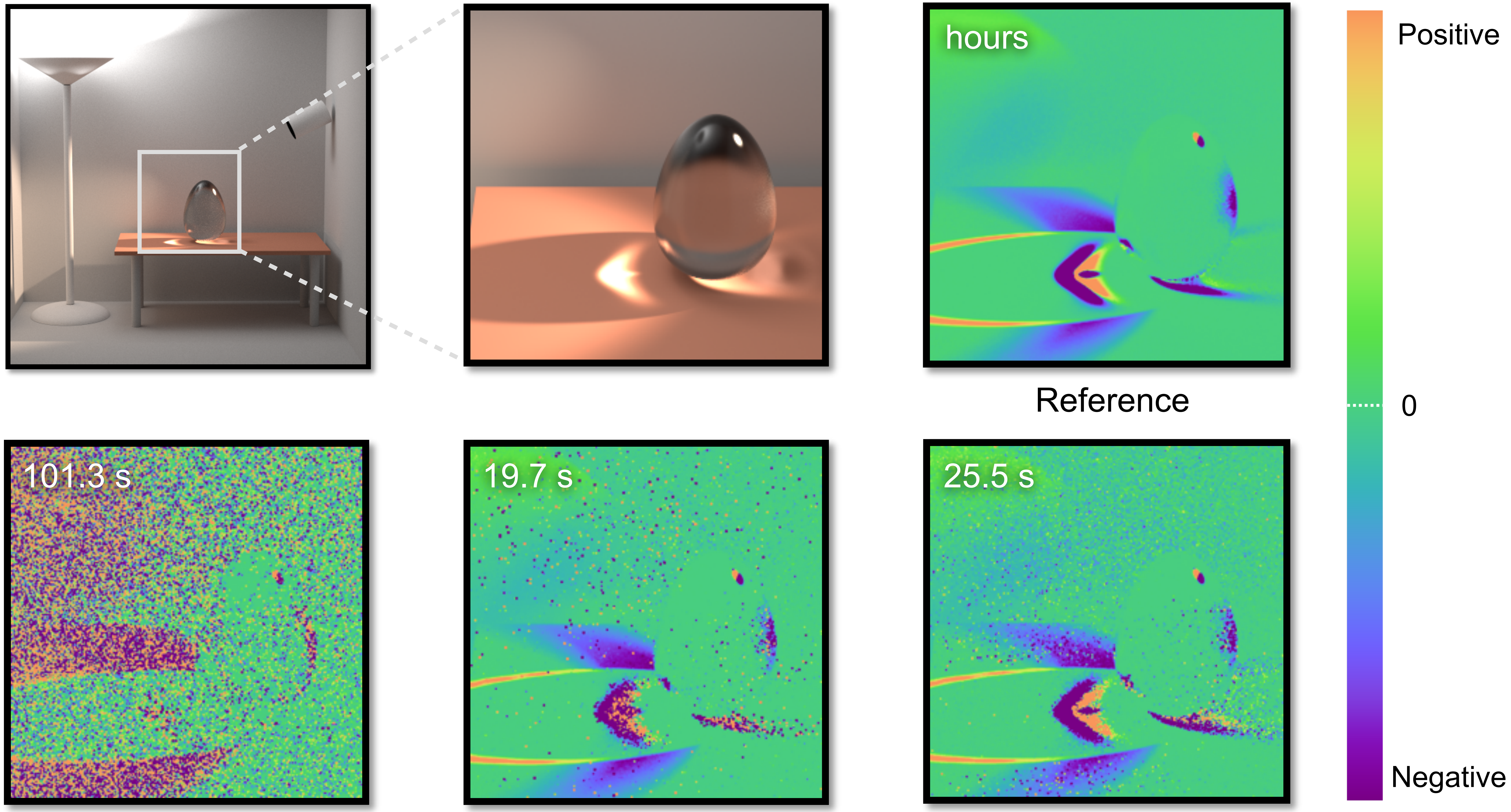
- Optimizing
  - Object rotation angle
- **Equal-sample** per iteration
- **Identical** optimization settings
  - Learning rate (Adam)
  - Initializations



RESULT: COMPLEX LIGHT-TRANSPORT EFFECT

Parameter: vertical position of the spot light

Equal-sample comparison



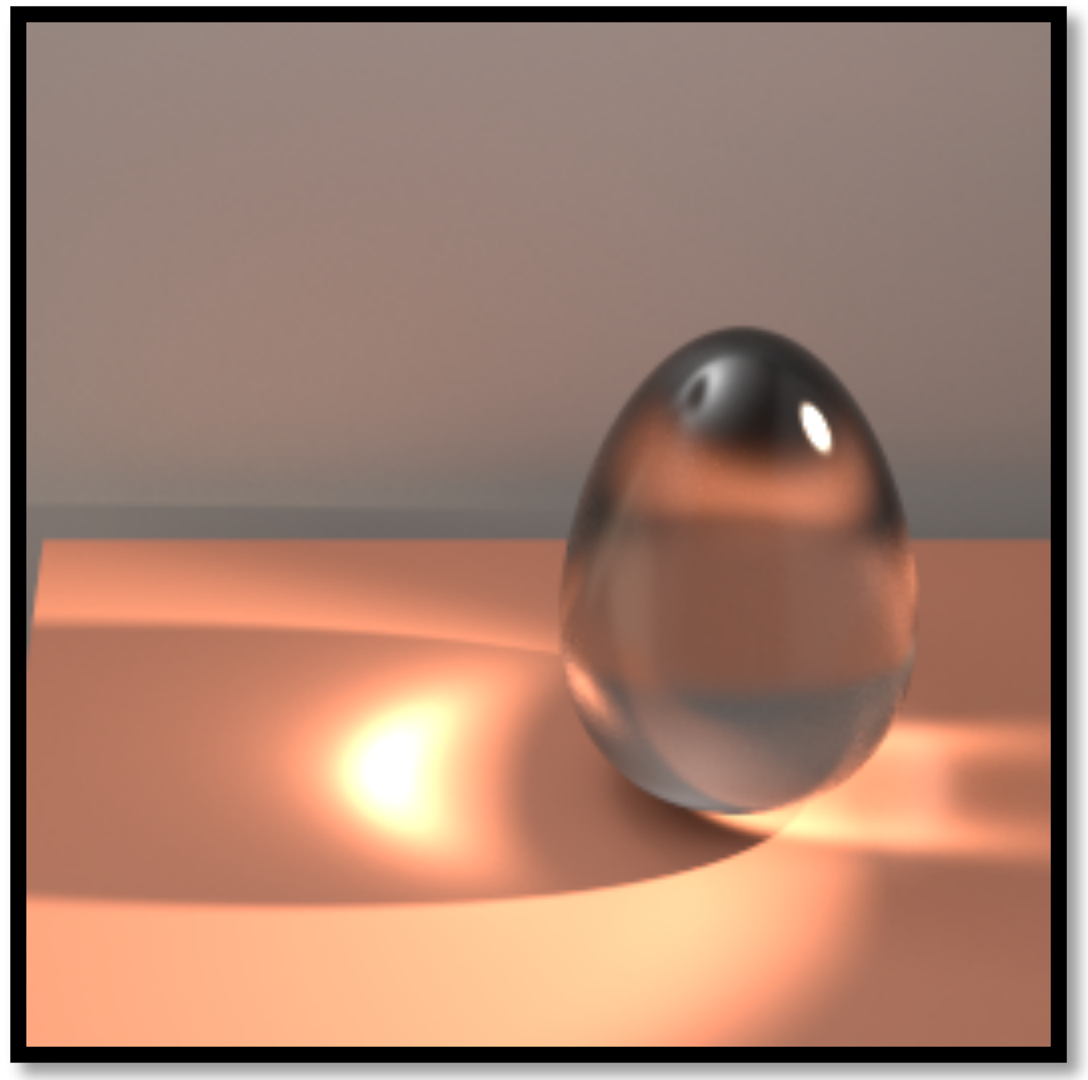
Path tracing w/ edge sampling [Li et al. 2018, Zhang et al. 2019]

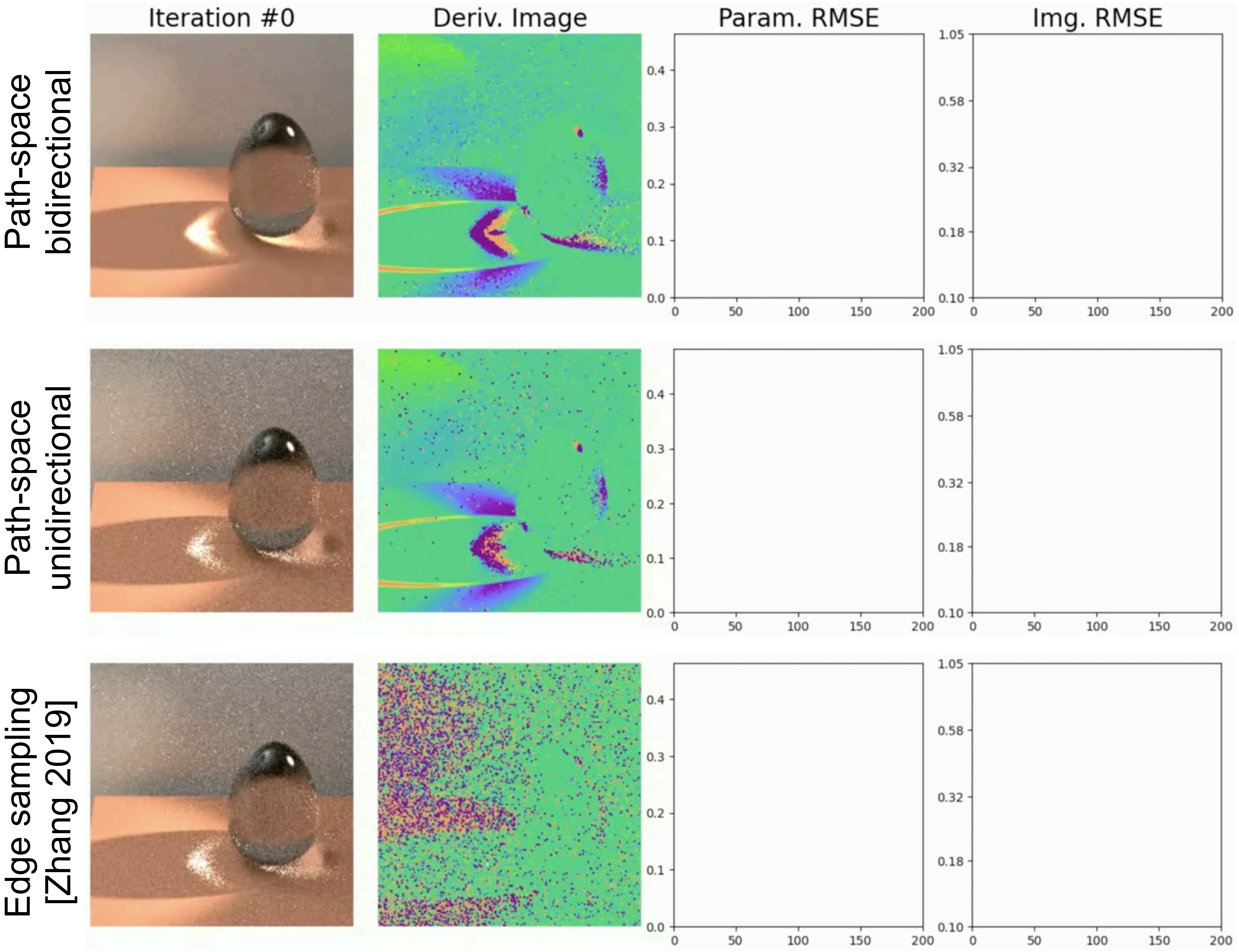
Path-space, unidirectional [Zhang et al. 2020]

Path-space, bidirectional [Zhang et al. 2020]

RESULT: COMPLEX LIGHT-TRANSPORT EFFECT

Target image

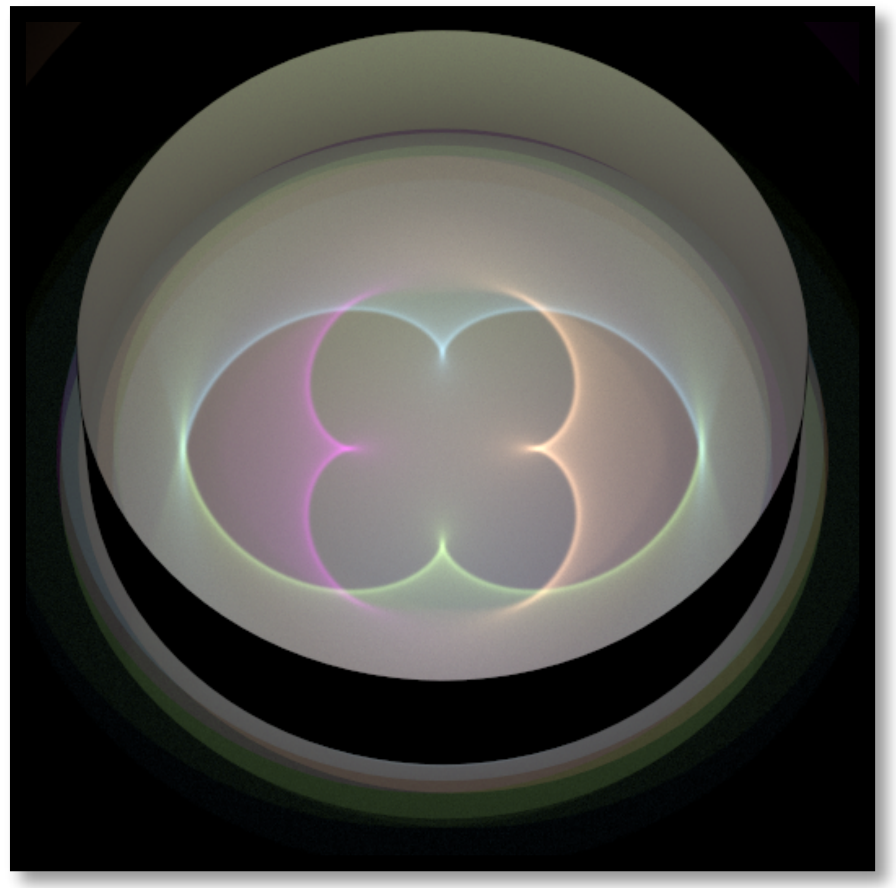




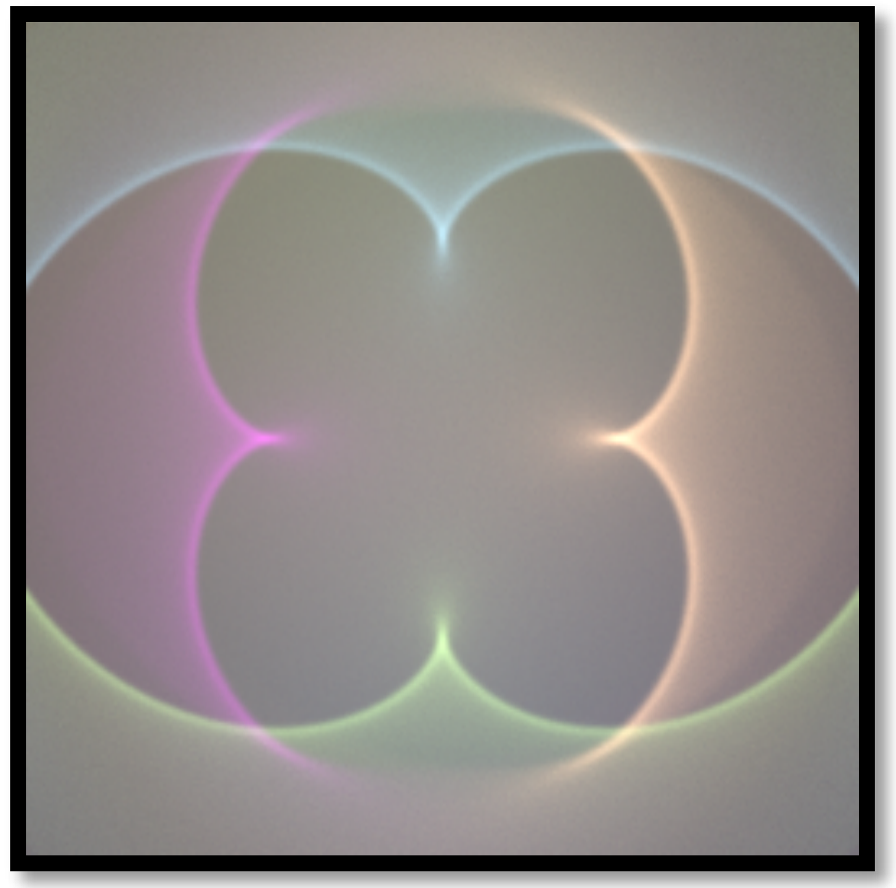
- Optimizing
  - Glass IOR
  - Spotlight position
- **Equal-time** per iteration
- **Identical** optimization settings
  - Learning rate (Adam)
  - Initializations

RESULT: COMPLEX LIGHT-TRANSPORT EFFECT

Config.

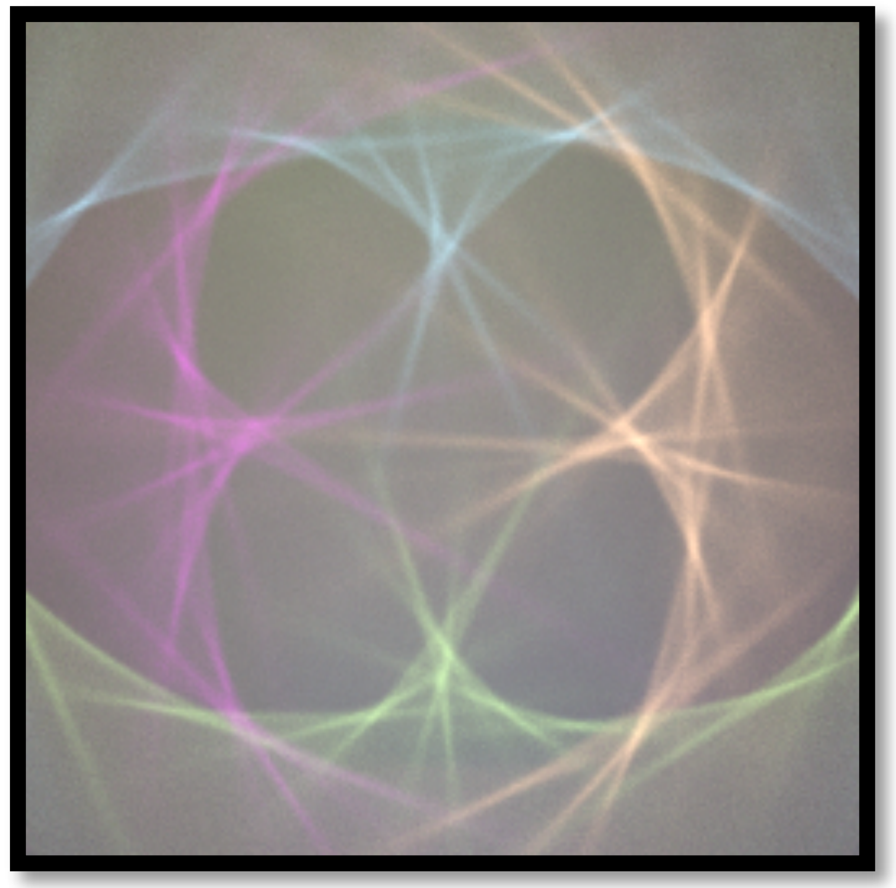


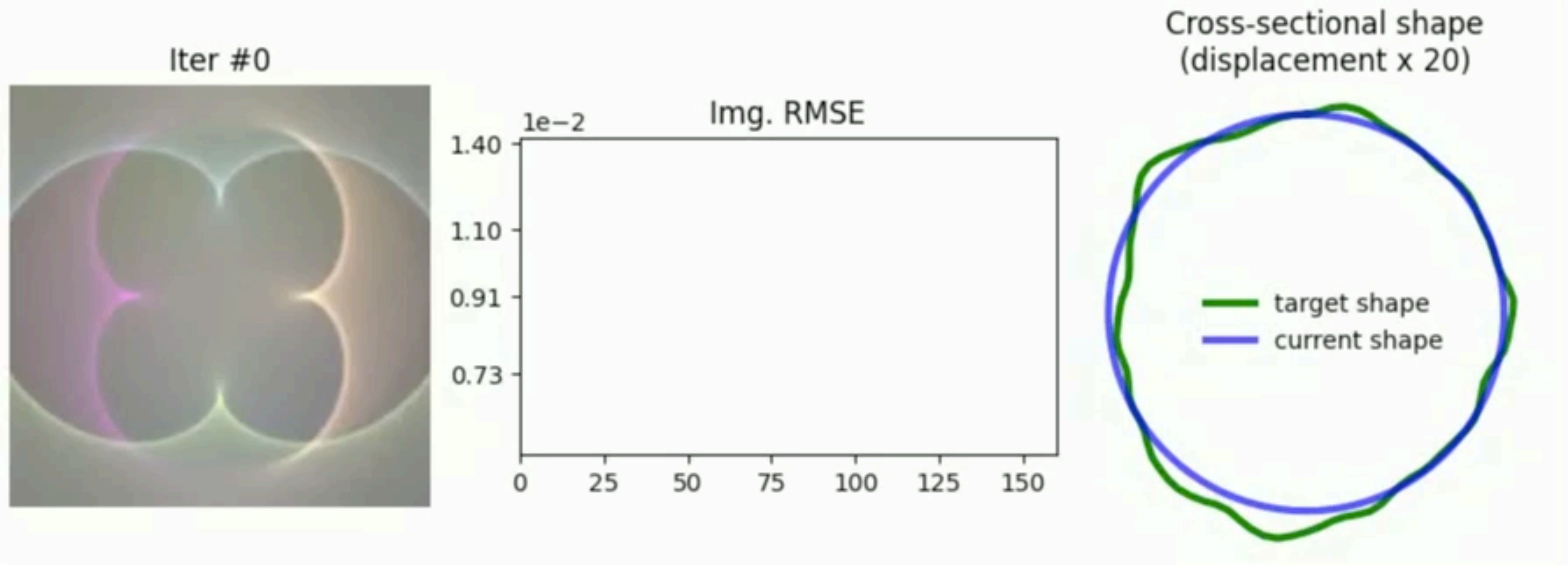
Initial



- Scene configuration:
  - A glossy ring lit by four colored light sources
- Optimizing:
  - **Cross-sectional shape** of the ring

Target



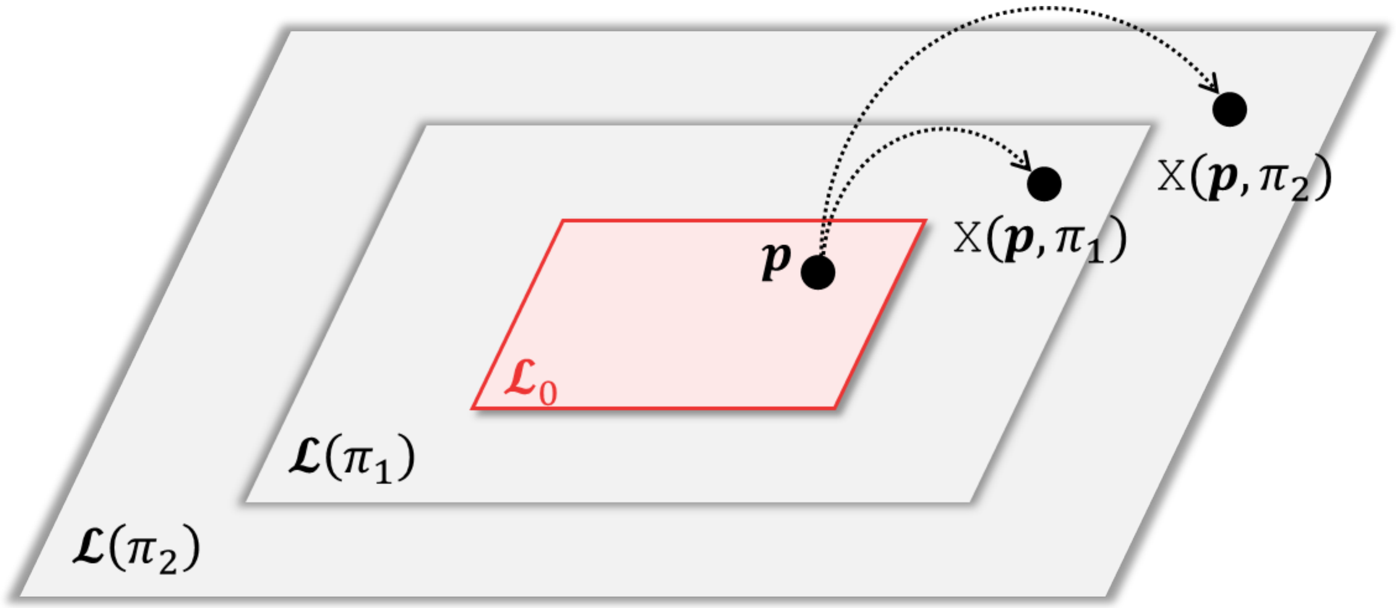


SUMMARY

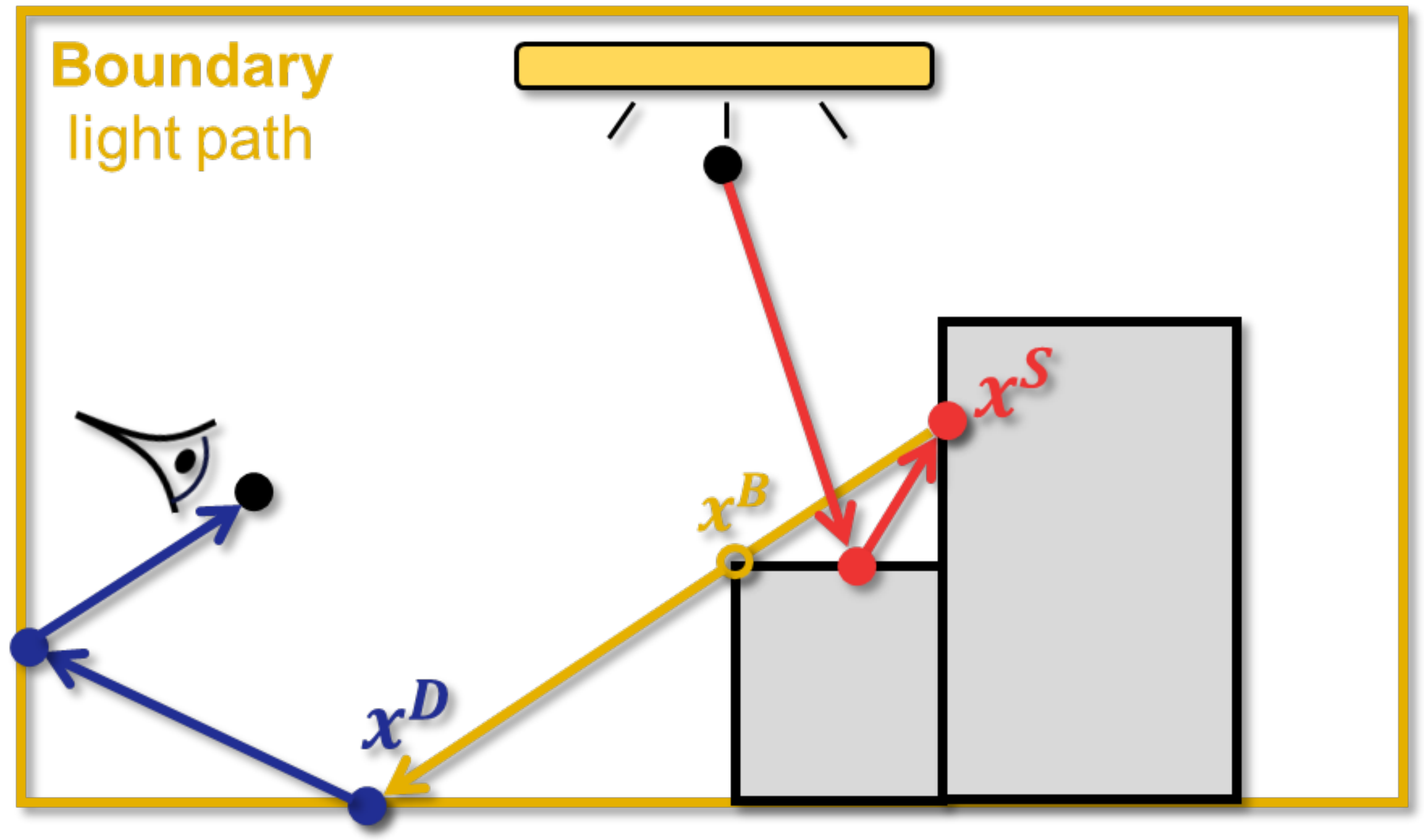
- **Differential path integral**
  - Separated interior and boundary components

$$\frac{\mathrm{d}}{\mathrm{d}\pi}\int_{\Omega} f\,\mathrm{d}\mu = \underbrace{\int_{\Omega}\frac{\mathrm{d}f}{\mathrm{d}\pi}\,\mathrm{d}\mu}_{\text{Interior}} + \underbrace{\int_{\partial\Omega} g\,\mathrm{d}\mu'}_{\text{Boundary}}$$

- **Reparameterization**
  - Only need to consider silhouette edges



- **Unbiased Monte Carlo methods**
  - Unidirectional and bidirectional algorithms
  - No silhouette detection is needed

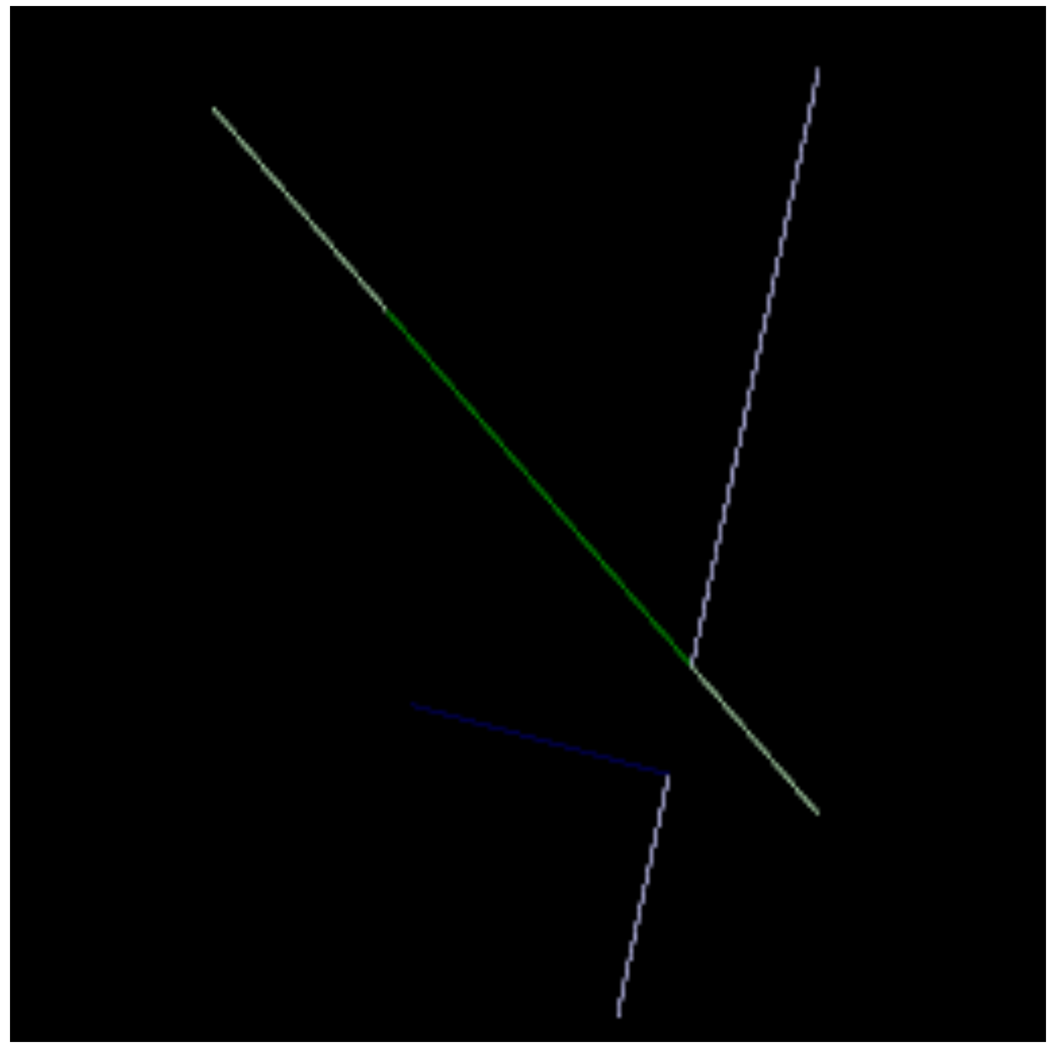


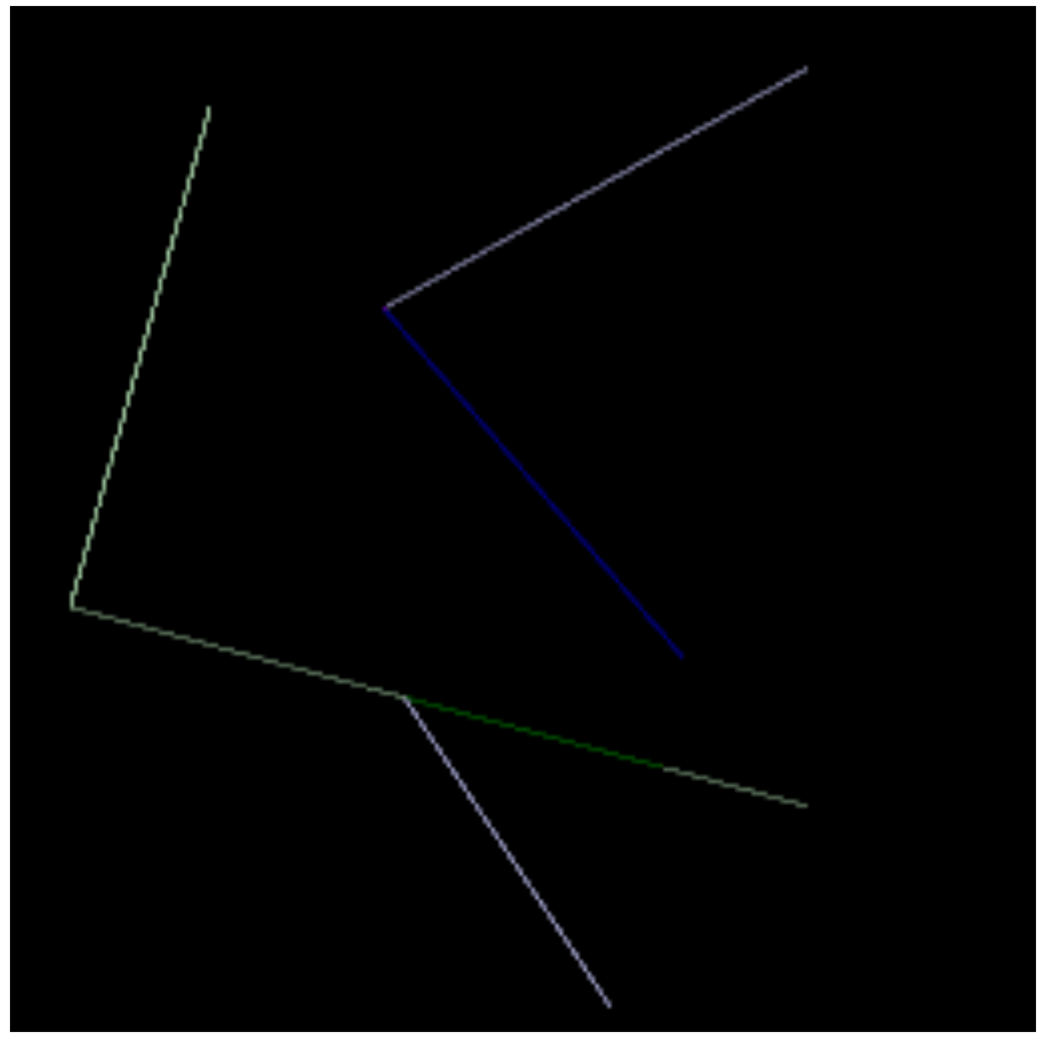
IMPLEMENTATION DETAILS

CODE WALKTHROUGH OF A 2D RENDERER

- C++11, single thread, ~300 lines
- render images of 2D triangles, their screen coordinates derivatives, and compute gradients w.r.t. vertex positions & color

"talk is cheap, show me the code!"





https://github.com/BachiLi/diffrender_tutorials

DATA STRUCTURES: 2D & 3D VECTORS

```
template <typename T>
struct Vec2 {
    T x, y;
    Vec2(T x = 0, T y = 0) : x(x), y(y) {}
};
template <typename T>
struct Vec3 {
    T x, y, z;
    Vec3(T x = 0, T y = 0, T z = 0) : x(x), y(y), z(z) {}
};
using Vec2f = Vec2<Real>;
using Vec3i = Vec3<int>;
using Vec3f = Vec3<Real>;

// some basic vector operations
// …
```

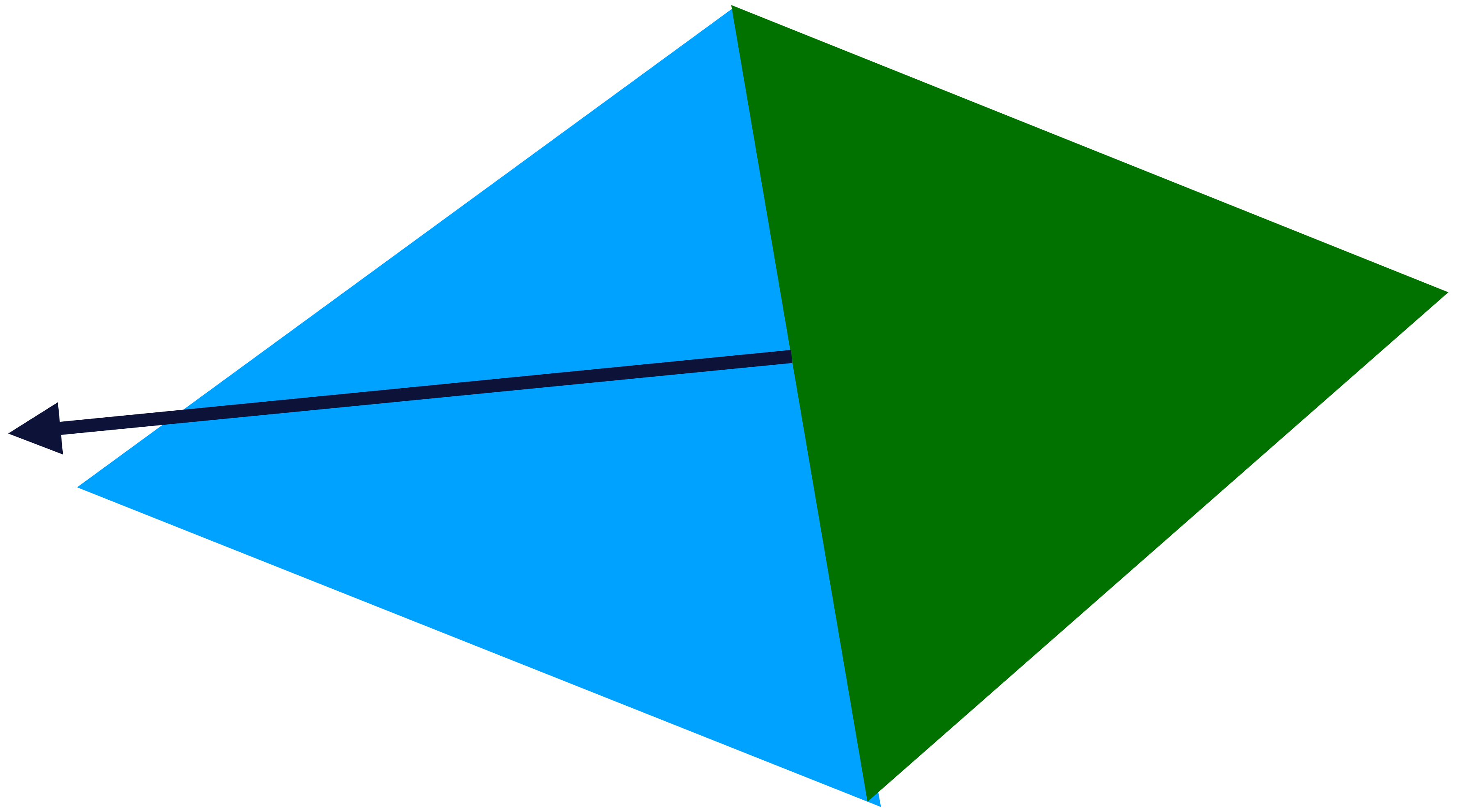
DATA STRUCTURES: TRIANGLE MESHES

```
struct TriangleMesh {
    vector<Vec2f> vertices;
    vector<Vec3i> indices;
    vector<Vec3f> colors; // defined for each face
};

// stores the gradients w.r.t. a triangle mesh
struct DTriangleMesh {
    DTriangleMesh(int num_vertices, int num_colors) {
        vertices.resize(num_vertices, Vec2f{0, 0});
        colors.resize(num_colors, Vec3f{0, 0, 0});
    }

    vector<Vec2f> vertices;
    vector<Vec3f> colors;
};
```

DATA STRUCTURES: EDGES

```
struct Edge {
    int v0, v1; // vertex ID, v0 < v1

    Edge(int v0, int v1) : v0(min(v0, v1)), v1(max(v0, v1)) {}

    // for sorting edges
    bool operator<(const Edge &e) const {
        return this->v0 != e.v0 ? this->v0 < e.v0 :
                                  this->v1 < e.v1;
    }
};
```

need to avoid double count this edge



DATA STRUCTURES: EDGE SAMPLER

```
// for sampling edges with inverse transform sampling
struct Sampler {
    vector<Real> pmf, cdf;
};


// binary search for inverting the CDF in the sampler
int sample(const Sampler &sampler, const Real u) {
    auto cdf = sampler.cdf;
    return clamp<int>(upper_bound(
        cdf.begin(), cdf.end(), u) - cdf.begin() - 1,
        0, cdf.size() - 2);
}
```

DATA STRUCTURES: 2D IMAGE

```
struct Img {
    Img(int width, int height,
        const Vec3f &val = Vec3f{0, 0, 0}) :
            width(width), height(height) {
        color.resize(width * height, val);
    }

    vector<Vec3f> color;
    int width;
    int height;
};
```

SETUP: THE MAIN() FUNCTION

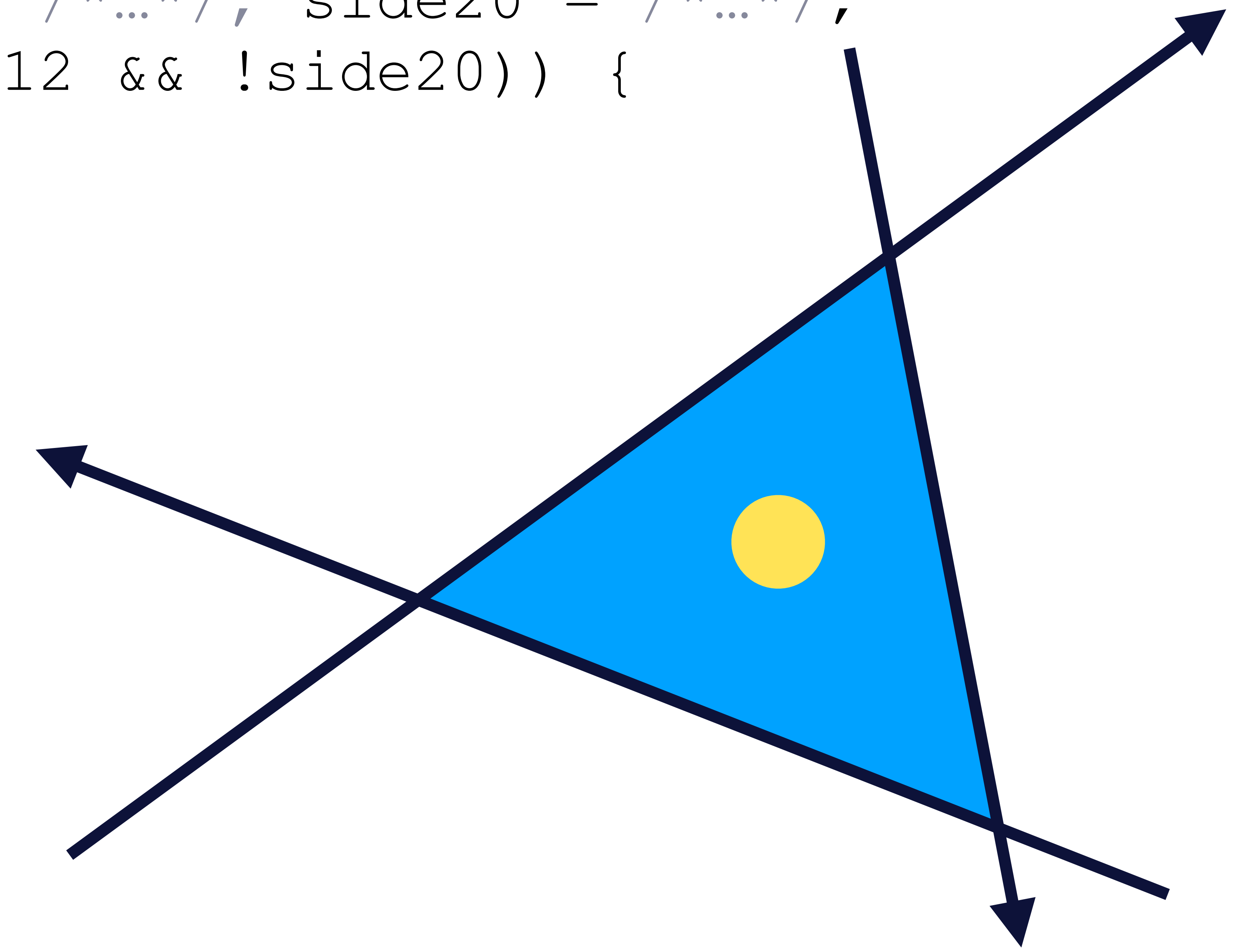
```
int main(int argc, char *argv[]) {
    TriangleMesh mesh{
        {{50.0, 25.0}, {200.0, 200.0}, {15.0, 150.0}, // vertices
         {200.0, 15.0}, {150.0, 250.0}, {50.0, 100.0}},
        {{0, 1, 2}, {3, 4, 5}}, // indices
        {{0.3, 0.5, 0.3}, {0.3, 0.3, 0.5}} // color
    };
    Img img(256, 256);
    mt19937 rng(1234);
    render(/*…*/);
    save_img(img, "render.ppm");
    // compute derivatives
    // adjoint is the gradient of some scalar loss w.r.t. image
    Img adjoint(img.width, img.height, Vec3f{1, 1, 1});
    Img dx(img.width, img.height), dy(img.width, img.height);
    DTriangleMesh d_mesh(mesh.vertices.size(), mesh.colors.size());
    d_render(/*…*/);
    // save derivative images
    // …
    return 0;
}
```

RENDER

```
void render(/*…*/) {
    // …
    for (int y = 0; y < img.height; y++) { // for each pixel
        for (int x = 0; x < img.width; x++) {
            for (int dy = 0; dy < sqrt_num_samples; dy++) { // for each subpixel
                for (int dx = 0; dx < sqrt_num_samples; dx++) {
                    // …
                    auto color = raytrace(mesh, screen_pos);
                    img.color[y * img.width + x] += color / samples_per_pixel;
                }
            }
        }
    }
}
```

“RAY TRACE”

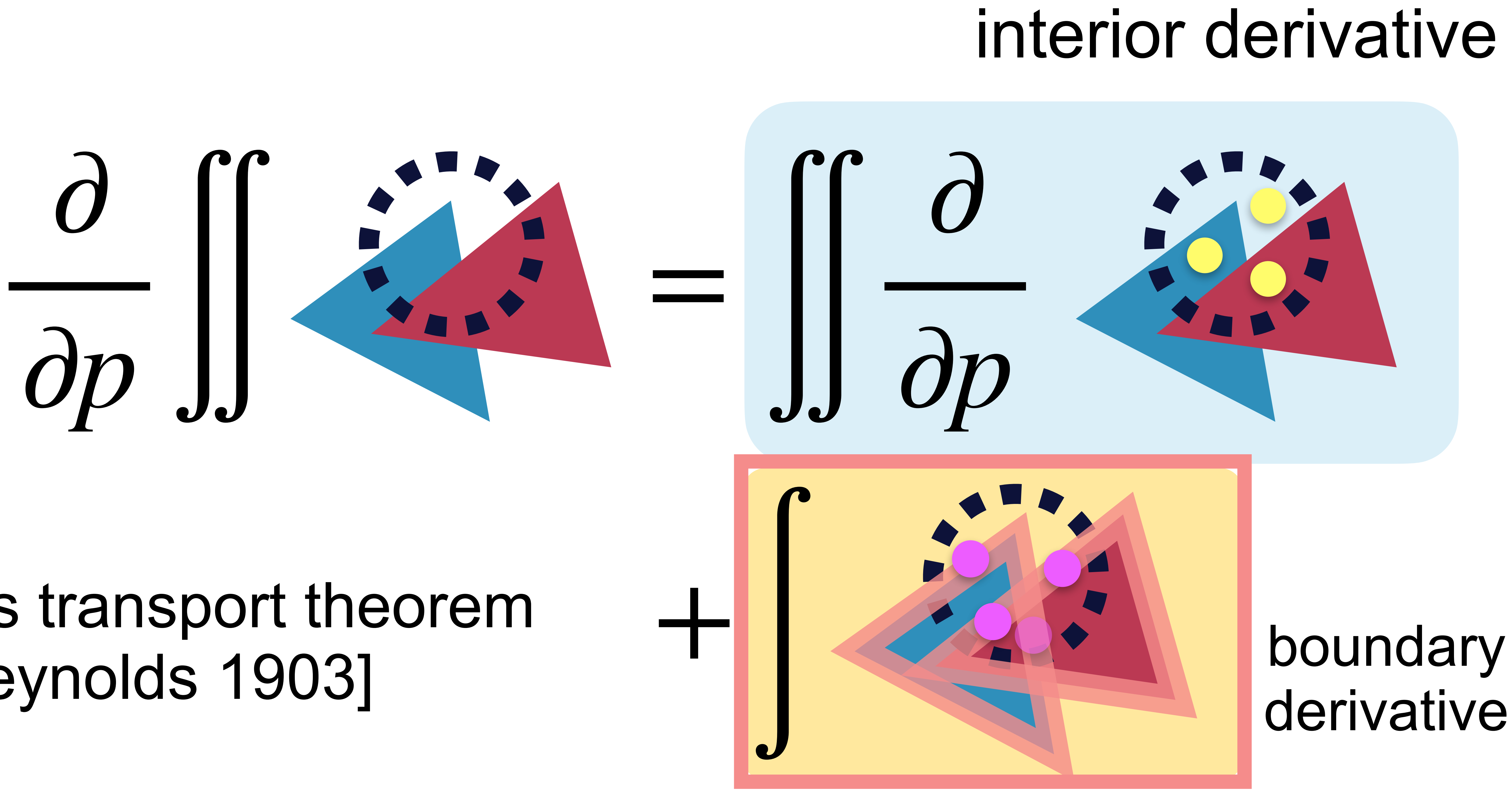
```
Vec3f raytrace(const TriangleMesh &mesh, const Vec2f &screen_pos, int *hit_index = nullptr) {
    // loop over all triangles in the mesh, return the first one that hits
    for (int i = 0; i < (int)mesh.indices.size(); i++) {
        // retrieve the three vertices of a triangle
        auto index = mesh.indices[i];
        auto v0 = mesh.vertices[index.x], v1 = mesh.vertices[index.y],
             v2 = mesh.vertices[index.z];
        // form three half-planes: v1-v0, v2-v1, v0-v2
        // if a point is on the same side of all three half-planes, it's inside the triangle
        auto n01 = normal(v1 - v0), n12 = normal(v2 - v1), n20 = normal(v0 - v2);
        auto side01 = dot(screen_pos - v0, n01) > 0, side12 = /*…*/, side20 = /*…*/;
        if ((side01 && side12 && side20) || (!side01 && !side12 && !side20)) {
            if (hit_index != nullptr) {
                *hit_index = i;
            }
            return mesh.colors[i];
        }
    }
    // return background
    // …
}
```



RENDERING: DONE!

DIFFERENTIABLE RENDER

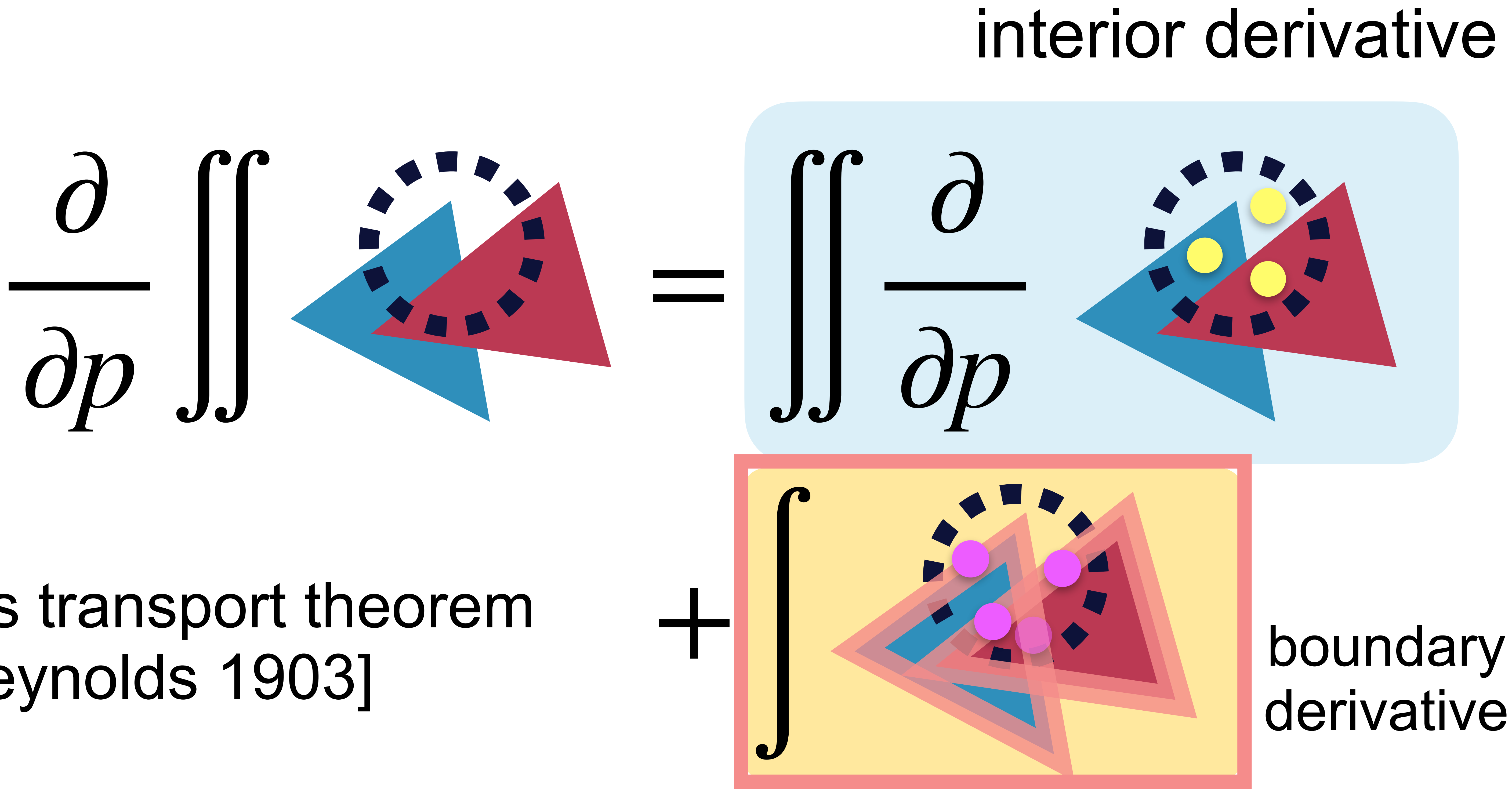

```
void d_render(/*…*/) {
    compute_interior_derivatives(/*…*/);
    auto edges = collect_edges(mesh);
    auto edge_sampler = build_edge_sampler(mesh, edges);
    compute_edge_derivatives(/*…*/);
}
```



Reynolds transport theorem
[Reynolds 1903]

DIFFERENTIABLE RENDER

```
void d_render(/*…*/) {
    compute_interior_derivatives(/*…*/);
    auto edges = collect_edges(mesh);
    auto edge_sampler = build_edge_sampler(mesh, edges);
    compute_edge_derivatives(/*…*/);
}
```



Reynolds transport theorem
[Reynolds 1903]

INTERIOR DERIVATIVES

```
void compute_interior_derivatives(/*…*/) {
    // …
    for (int y = 0; y < adjoint.height; y++) { // for each pixel
        for (int x = 0; x < adjoint.width; x++) {
            for (int dy = 0; dy < sqrt_num_samples; dy++) { // for each subpixel
                for (int dx = 0; dx < sqrt_num_samples; dx++) {
                    // …
                    int hit_index = -1;
                    raytrace(mesh, screen_pos, &hit_index);
                    if (hit_index != -1) {
                        // scatter to the gradient buffer
                        d_colors[hit_index] +=
                            adjoint.color[y * adjoint.width + x] / samples_per_pixel;
                    }
                }
            }
        }
    }
}
```

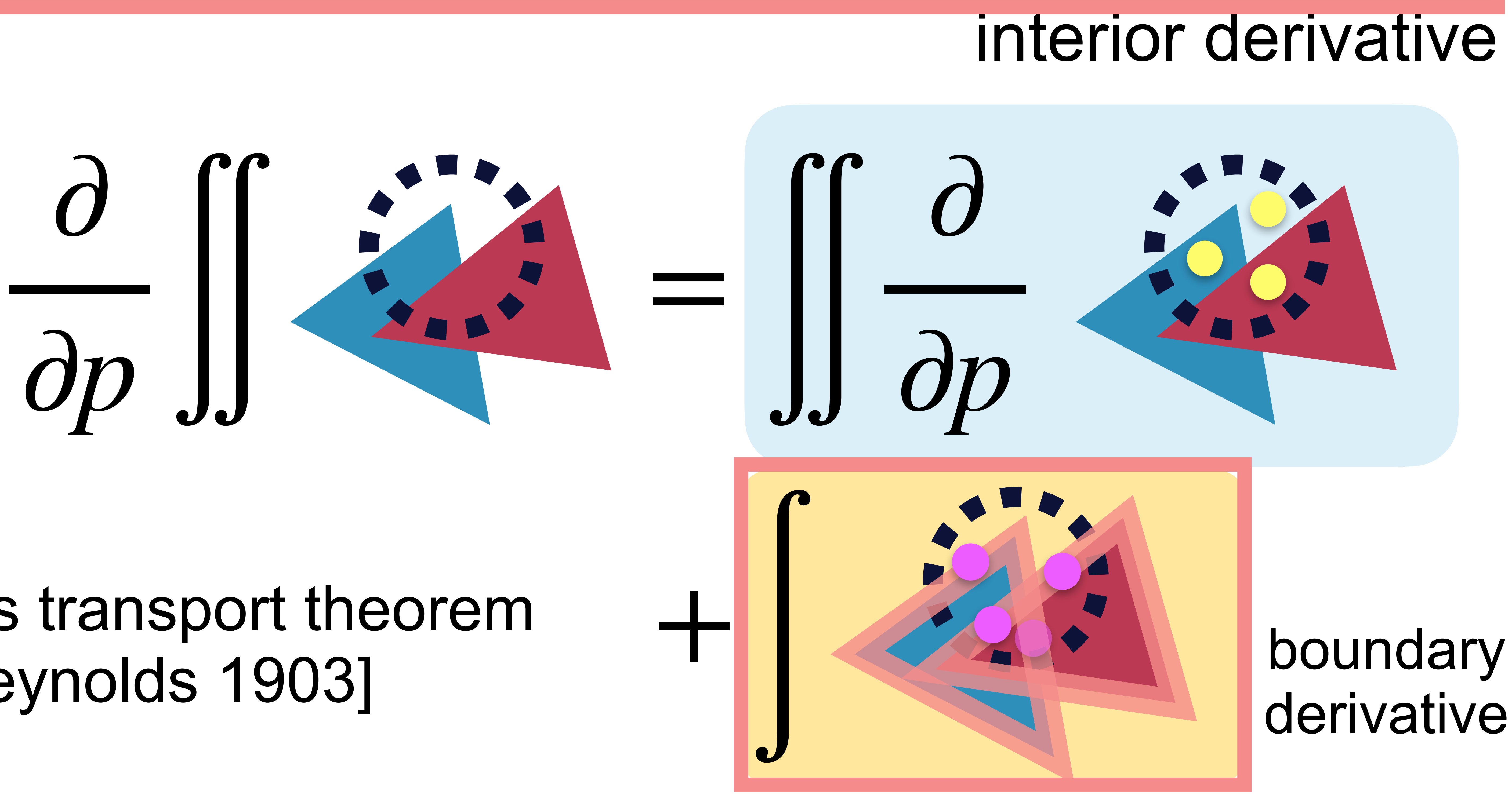
INTERIOR DERIVATIVES

```
void compute_interior_derivatives(/*…*/) {
    // …
    for (int y = 0; y < adjoint.height; y++) { // for each pixel
        for (int x = 0; x < adjoint.width; x++) {
            for (int dy = 0; dy < sqrt_num_samples; dy++) { // for each subpixel
                for (int dx = 0; dx < sqrt_num_samples; dx++) {
                    // …
                    int hit_index = -1;
                    raytrace(mesh, screen_pos, &hit_index);
                    if (hit_index != -1) {
                        // scatter to the gradient buffer
                        d_colors[hit_index] +=
                            adjoint.color[y * adjoint.width + x] / samples_per_pixel;
                    }
                }
            }
        }
    }
}
```

- automatic differentiation of a standard renderer
- can be replaced by radiative backpropagation

DIFFERENTIABLE RENDER

```
void d_render(/*…*/) {
    compute_interior_derivatives(/*…*/);
    auto edges = collect_edges(mesh);
    auto edge_sampler = build_edge_sampler(mesh, edges);
    compute_edge_derivatives(/*…*/);
}
```



Reynolds transport theorem
[Reynolds 1903]

COLLECT EDGES

```
vector<Edge> collect_edges(const TriangleMesh &mesh) {
    set<Edge> edges;
    for (auto index : mesh.indices) {
        edges.insert(Edge(index.x, index.y));
        edges.insert(Edge(index.y, index.z));
        edges.insert(Edge(index.z, index.x));
    }
    return vector<Edge>(edges.begin(), edges.end());
}
```

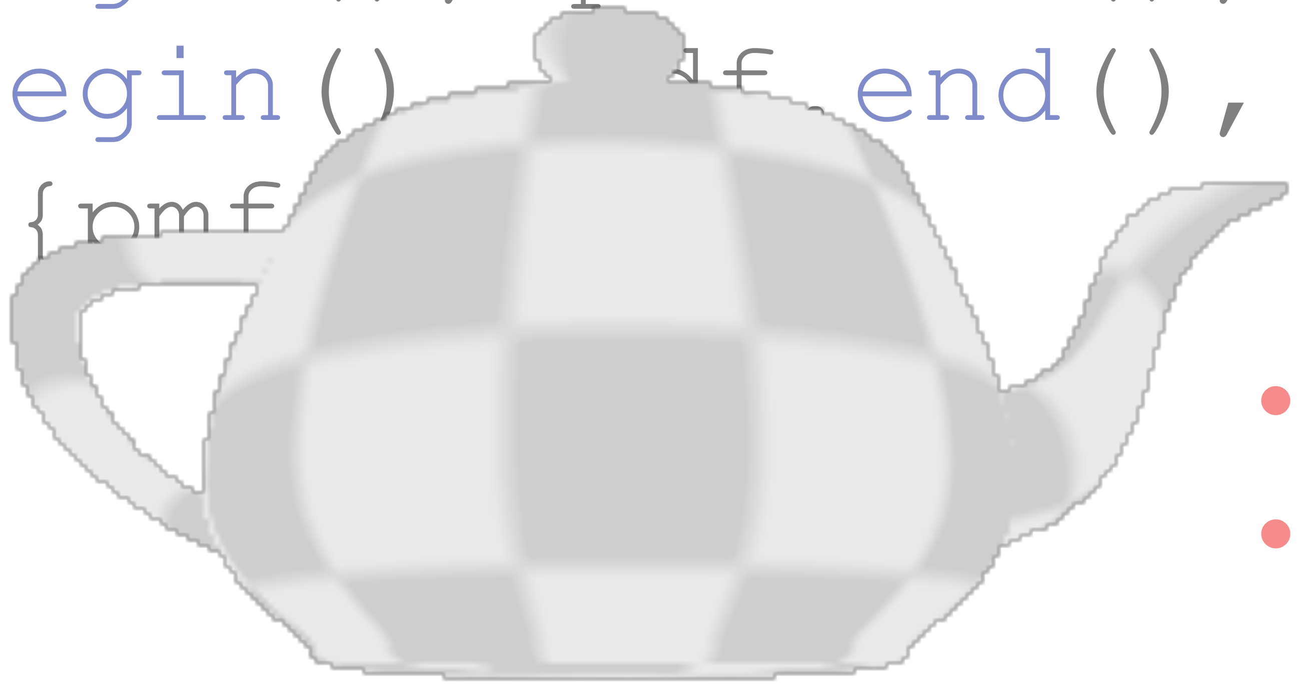
- can be parallelized using parallel sorting + parallel stream compaction

BUILD EDGE SAMPLER


```
// build a discrete CDF using edge length
Sampler build_edge_sampler(const TriangleMesh &mesh,
                           const vector<Edge> &edges) {
    vector<Real> pmf, cdf;
    pmf.reserve(edges.size()); cdf.reserve(edges.size() + 1);
    cdf.push_back(0);
    for (auto edge : edges) {
        auto v0 = mesh.vertices[edge.v0], v1 = mesh.vertices[edge.v1];
        pmf.push_back(length(v1 - v0));
        cdf.push_back(pmf.back() + cdf.back());
    }
    auto length_sum = cdf.back(); // normalize pmf/cdf
    for_each(pmf.begin(), pmf.end(), [&](Real &p) {p /= length_sum;});
    for_each(cdf.begin(), cdf.end(), [&](Real &p) {p /= length_sum;});
    return Sampler{pmf, cdf};
}
```

BUILD EDGE SAMPLER

```
// build a discrete CDF using edge length
Sampler build_edge_sampler(const TriangleMesh &mesh,
                           const vector<Edge> &edges) {
    vector<Real> pmf, cdf;
    pmf.reserve(edges.size()); cdf.reserve(edges.size() + 1);
    cdf.push_back(0);
    for (auto edge : edges) {
        auto v0 = mesh.vertices[edge.v0], v1 = mesh.vertices[edge.v1];
        pmf.push_back(length(v1 - v0));
        cdf.push_back(pmf.back() + cdf.back());
    }
    auto length_sum = cdf.back(); // normalize pmf/cdf
    for_each(pmf.begin(), pmf.end(), [&](Real &p) {p /= length_sum;});
    for_each(cdf.begin(), cdf.end(), [&](Real &p) {p /= length_sum;});
    return Sampler{pmf
}
```



- can exclude non-silhouette edges here
- can be parallelized using parallel scans

EDGE SAMPLING!

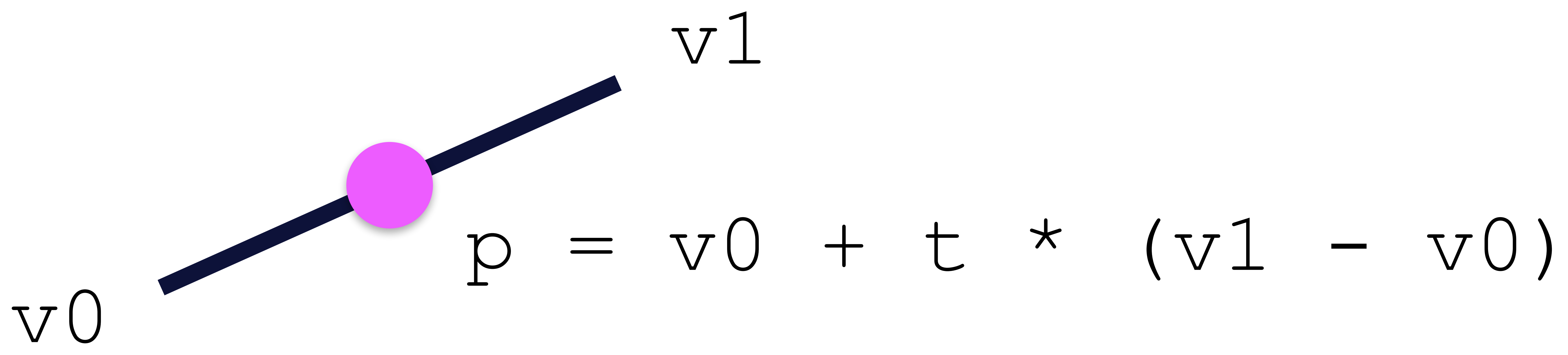
```
void compute_edge_derivatives(/*…*/) {
    for (int i = 0; i < num_edge_samples; i++) {
        // pick an edge
        // …
        // pick a point p on the edge
        // …
        // compute the colors at the two sides of p
        // …
        // compute the weights using the PDF and adjoint image
        // …
        // compute the derivatives using the Reynolds transport theorem
        // …
    }
}
```

EDGE SAMPLING — PICKING EDGES

```
void compute_edge_derivatives(/*…*/) {
    for (int i = 0; i < num_edge_samples; i++) {
        // pick an edge
        // …
        // pick a point p on the edge
        // …
        // compute the colors at the two sides of p
        // …
        // compute the weights using the PDF and adjoint image
        // …
        // compute the derivatives using the Reynolds transport theorem
        // …
    }
}
```

EDGE SAMPLING — PICKING EDGES

```
// pick an edge
auto edge_id = sample(edge_sampler, uni_dist(rng));
auto edge = edges[edge_id];
// pick a point p on the edge
auto v0 = mesh.vertices[edge.v0], v1 = mesh.vertices[edge.v1];
auto t = uni_dist(rng);
auto p = v0 + t * (v1 - v0);
auto xi = (int)p.x; auto yi = (int)p.y; // integer coordinates
if (xi < 0 || yi < 0 || xi >= adjoint.width || yi >= adjoint.height) continue;
```

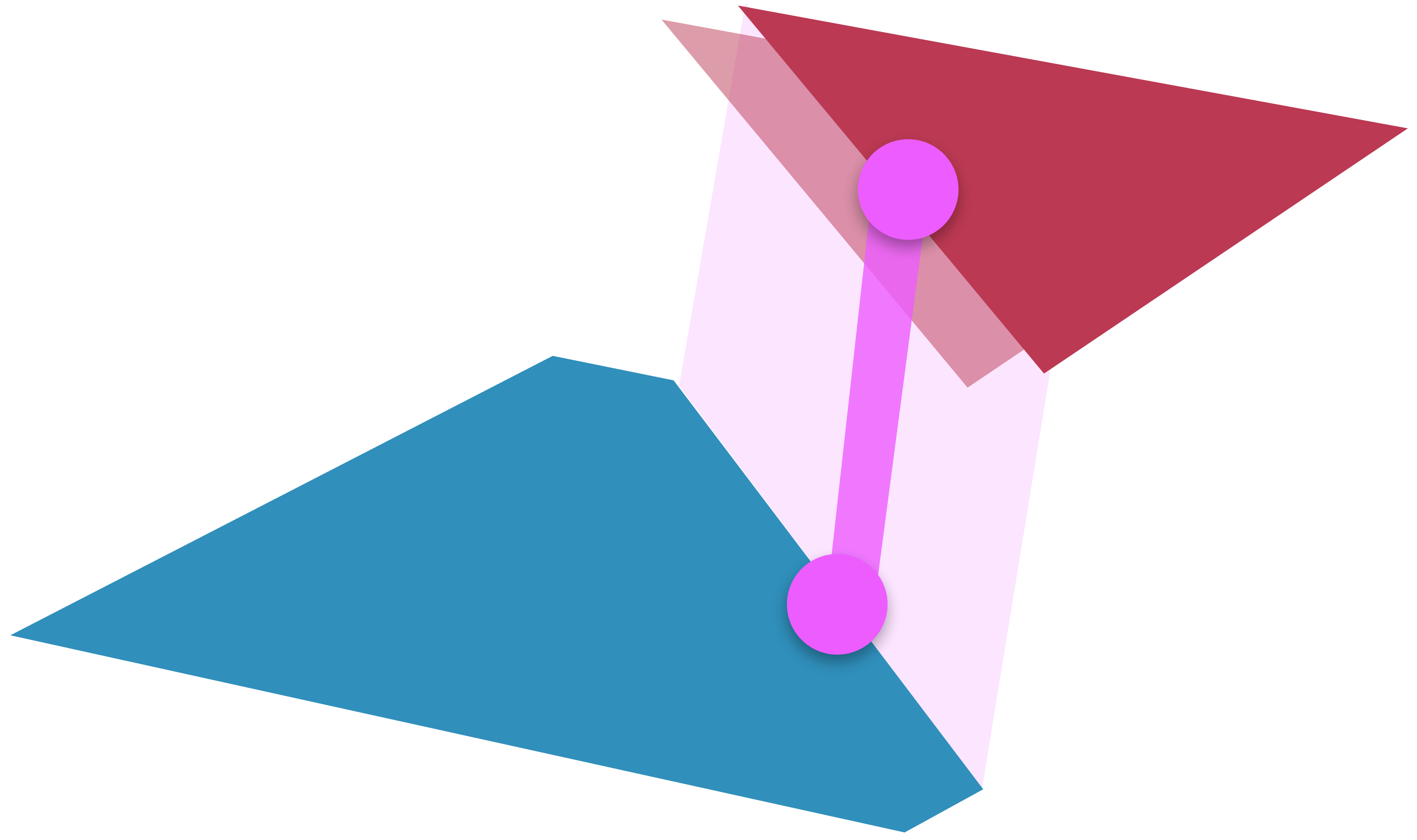


EDGE SAMPLING — COMPUTE DIFFERENCES

```
void compute_edge_derivatives(/*…*/) {
    for (int i = 0; i < num_edge_samples; i++) {
        // pick an edge
        // …
        // pick a point p on the edge
        // …
        // compute the colors at the two sides of p
        // …
        // compute the weights using the PDF and adjoint image
        // …
        // compute the derivatives using the Reynolds transport theorem
        // …
    }
}
```

EDGE SAMPLING — COMPUTE DIFFERENCES

```
// …
// compute the colors at the two sides of the selected edge
auto n = normal((v1 - v0) / length(v1 - v0));
auto color_in = raytrace(mesh, p - 1e-3f * n),
     color_out = raytrace(mesh, p + 1e-3f * n);
// …
```



EDGE SAMPLING — COMPUTE WEIGHTS

```
void compute_edge_derivatives(/*…*/) {
    for (int i = 0; i < num_edge_samples; i++) {
        // pick an edge
        // …
        // pick a point p on the edge
        // …
        // sample the two sides of p
        //
        // compute the weights using the PDF and adjoint image
        // …
        // compute the derivatives using the Reynolds transport theorem
        // …
    }
}
```

EDGE SAMPLING — COMPUTE WEIGHTS

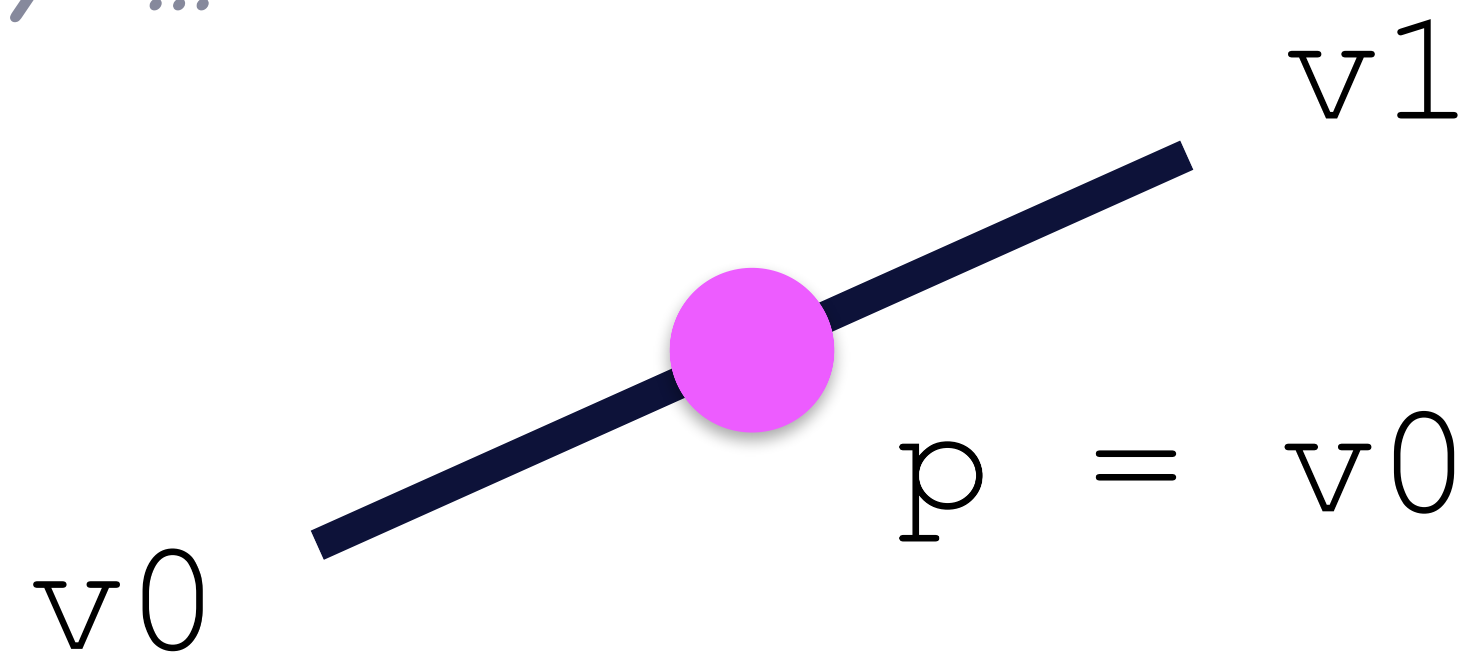
```
// …
// compute the weights using the PDF and adjoint image
auto pmf = edge_sampler.pmf[edge_id];
auto pdf = pmf / (length(v1 - v0));
auto weight = Real(1 / (pdf * Real(num_edge_samples)));
auto adj = dot(color_in - color_out,
               adjoint.color[yi * adjoint.width + xi]);
// …
```

EDGE SAMPLING — REYNOLDS TRANSPORT THEOREM

```
void compute_edge_derivatives(/*…*/) {
    for (int i = 0; i < num_edge_samples; i++) {
        // pick an edge
        // …
        // pick a point p on the edge
        // …
        // sample the two sides of p
        // …
        // compute the weights using the PDF and adjoint image
        // …
        // compute the derivatives using the Reynolds transport theorem
        //
    }
}
```

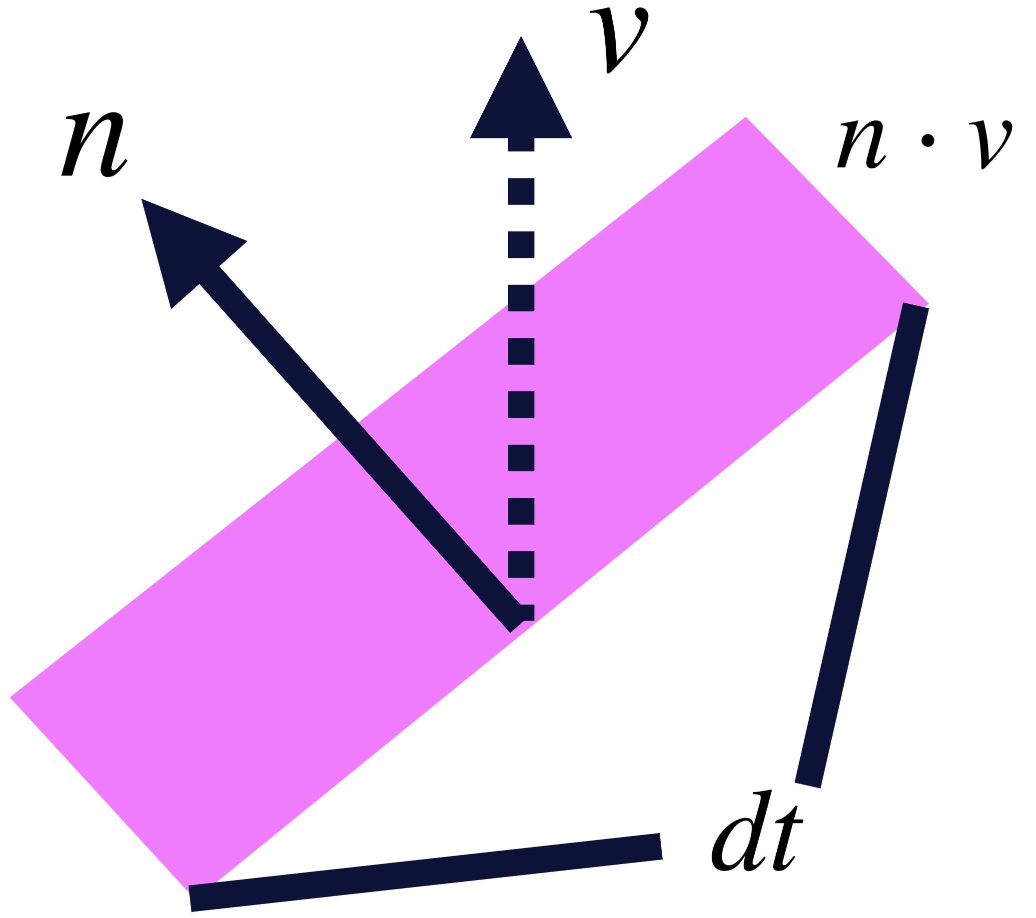
EDGE SAMPLING — REYNOLDS TRANSPORT THEOREM


```
// …
// compute the derivatives using Reynolds transport theorem
auto d_v0 = Vec2f{(1 - t) * n.x, (1 - t) * n.y} * adj * weight;
auto d_v1 = Vec2f{     t  * n.x,      t  * n.y} * adj * weight;
// screen coordinate derivatives ignore the adjoint
auto dx = -n.x * (color_in - color_out) * weight;
auto dy = -n.y * (color_in - color_out) * weight;
// scatter gradients to buffers
// …
```



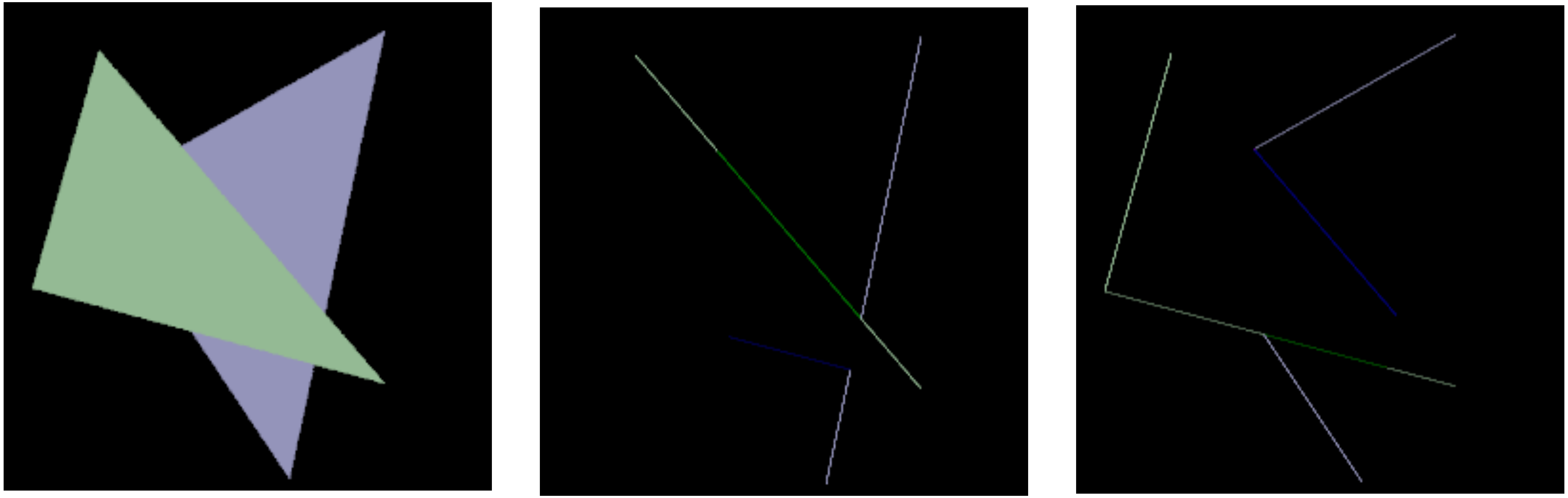
p = v0 + t * (v1 - v0)

$$v = \frac{\partial p}{\partial \text{param}}$$



$$\int \left(f_- - f_+\right)(n \cdot v)\, dt$$

EDGE SAMPLING: DONE!



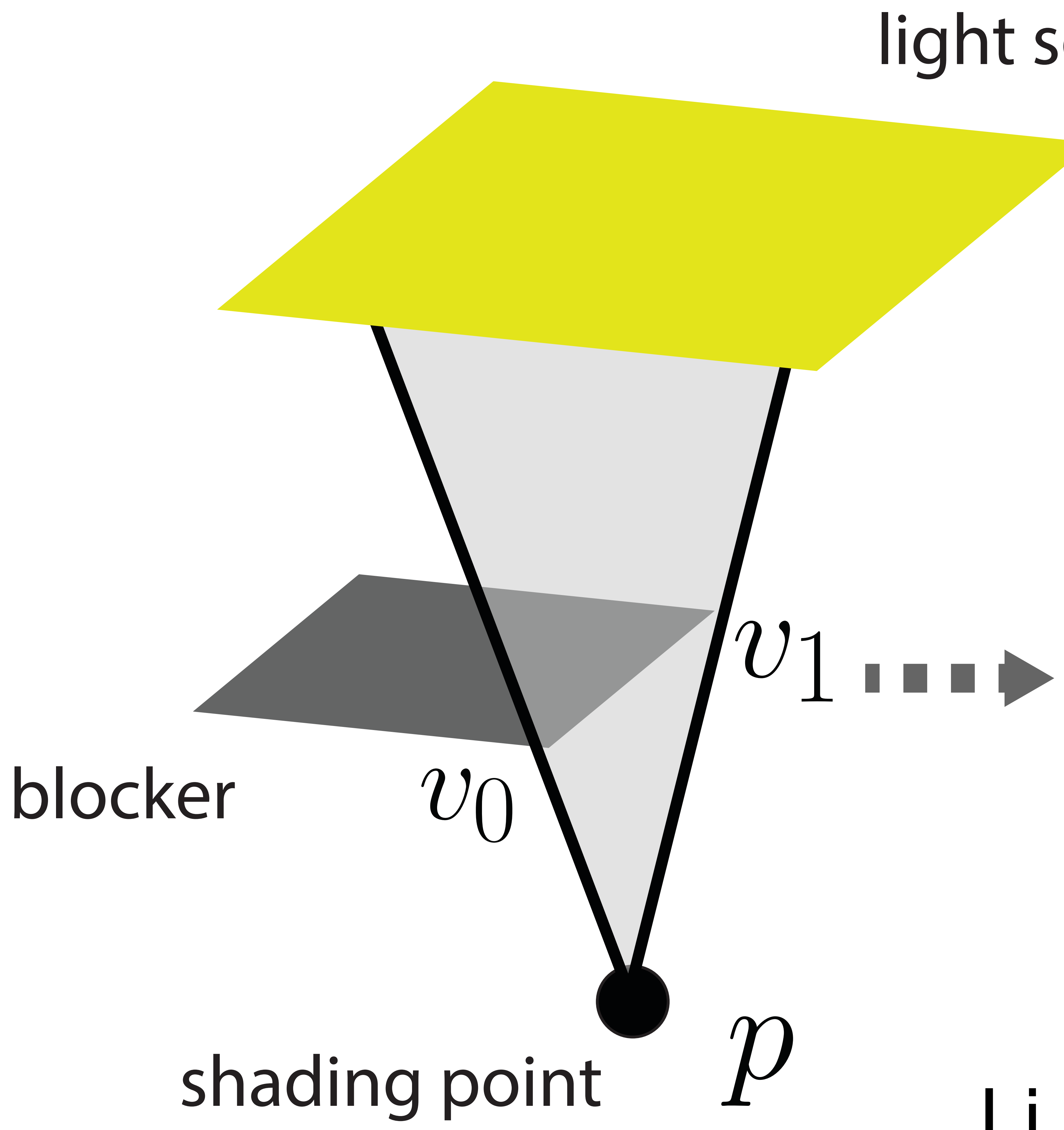
negative dx

positive dx

EXTENSION & DISCUSSIONS

- 3D ray tracing and edge sampling
- camera
- spatially-varying shading
- differentiating PDFs in interior derivatives
- stratification

3D RAY TRACING AND EDGE SAMPLING



$$\int (f_- - f_+)\,(n \cdot v)\,dt$$

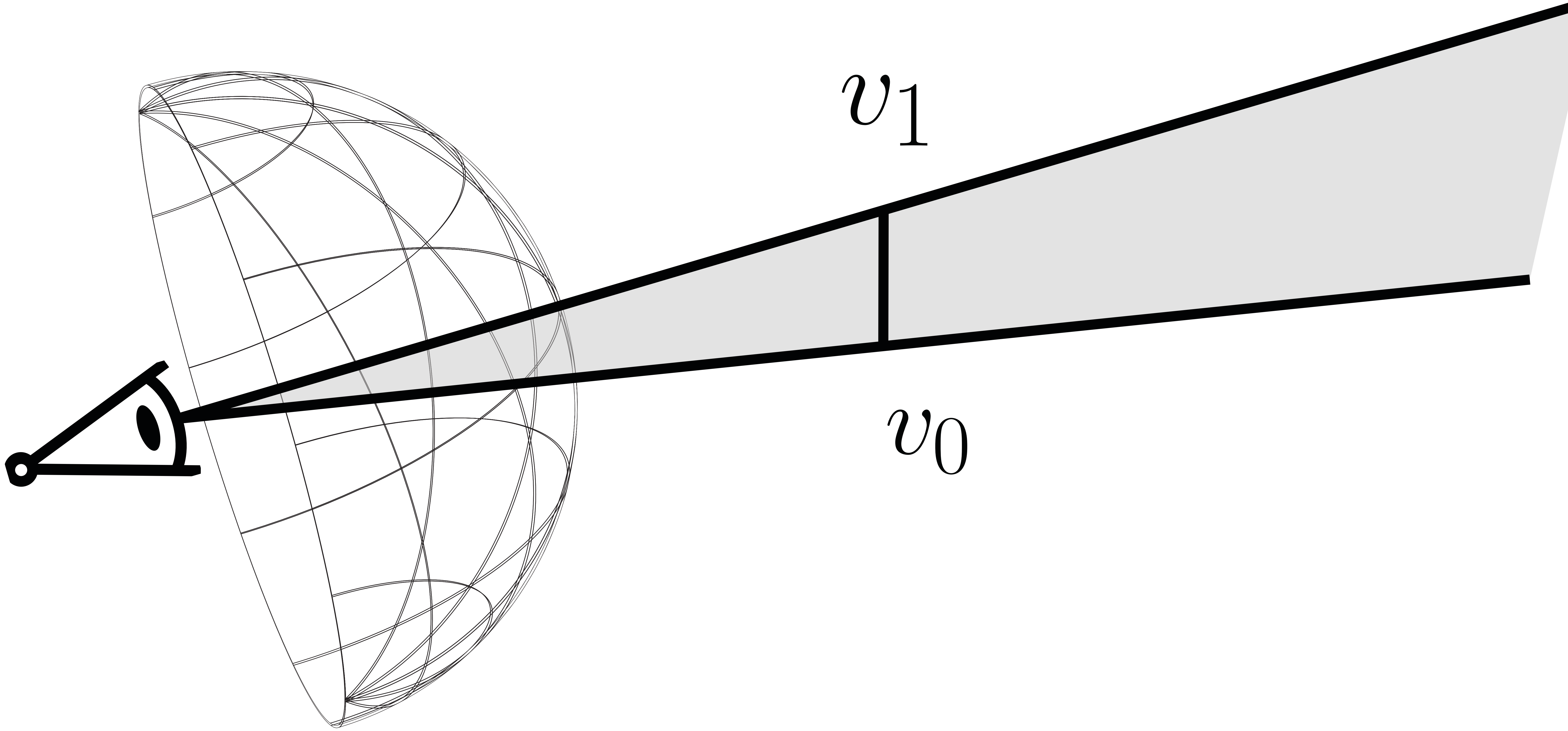
what are these?

Li et al. 2018 & Zhang et al. 2019 derived the equations

implementation at
https://github.com/BachiLi/diffrender_tutorials

CAMERA

- linear projective cameras: a preprocessing pass
- other cameras: use the 3D formula and backprop to camera parameters
  - also works for defocus blur



SPATIALLY-VARYING SHADING

- Should be handled in the interior derivatives
- Use a smooth texture reconstruction filter (e.g., trilinear interpolation)
  - mipmapping is extremely important for variance reduction

```
void compute_interior_derivatives(/*…*/) {
    // …
    for (int y = 0; y < adjoint.height; y++) { // for each pixel
        for (int x = 0; x < adjoint.width; x++) {
            for (int dy = 0; dy < sqrt_num_samples; dy++) { // for each subpixel
                for (int dx = 0; dx < sqrt_num_samples; dx++) {
                    // …
                    int hit_index = -1;
                    raytrace(mesh, screen_pos, &hit_index);
                    if (hit_index != -1) {
                        // scatter to the gradient buffer
                        d_colors[hit_index] +=
                            adjoint.color[y * adjoint.width + x] / samples_per_pixel;
                    }
                }
            }
        }
    }
}
```

DIFFERENTIATING PDFS IN INTERIOR DERIVATIVES

- PDF in importance sampling = Jacobian for reparametrization
- The Monte Carlo estimator is correct whether you backprop through the PDF or not
  - unclear which one has lower variance, pick the one that is more computationally convenient

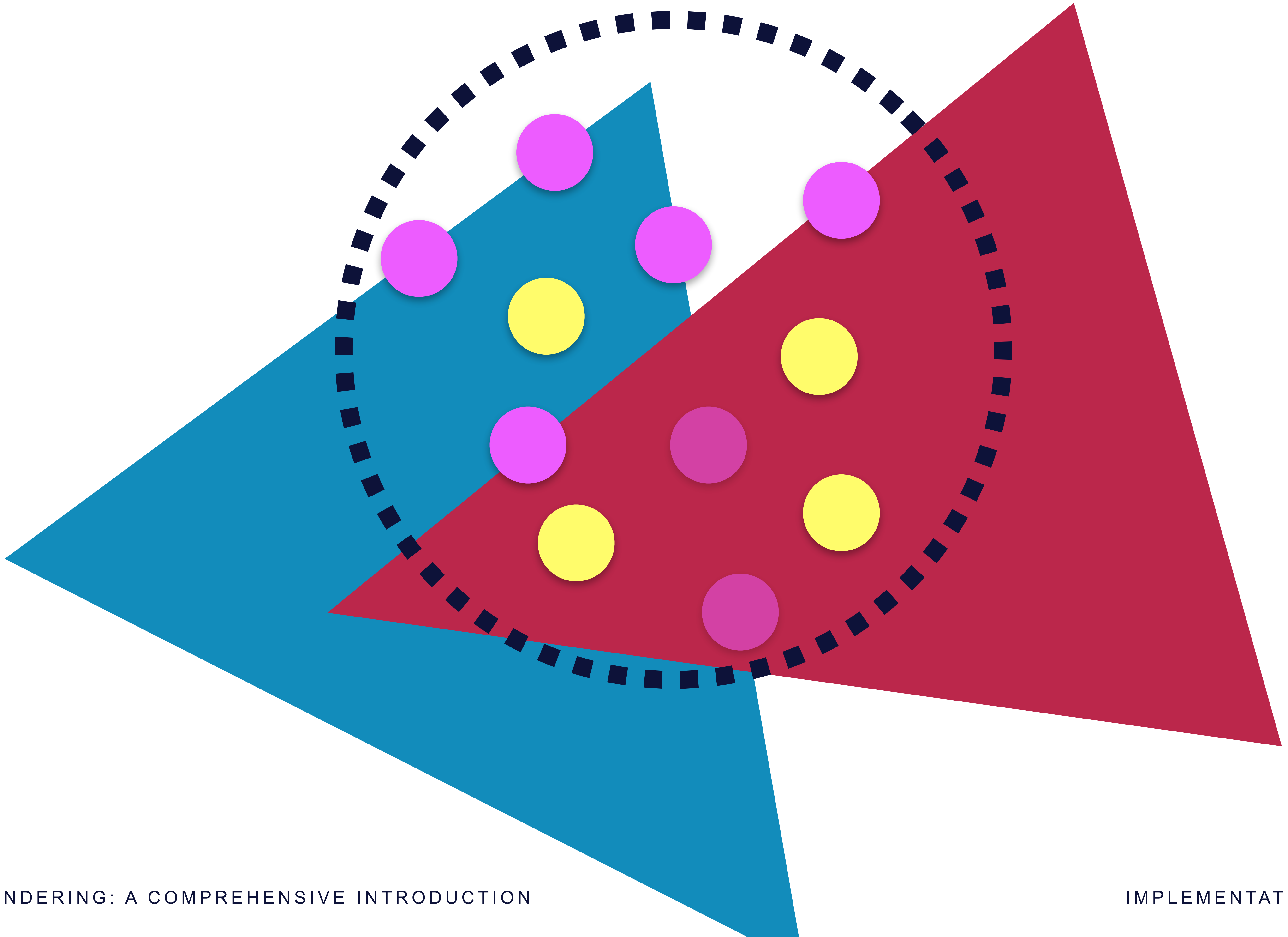
$$\int \nabla_\theta f(x;\theta)dx \qquad x = g(y)$$

$$= \int \left(\nabla_\theta f(x;\theta) \right) \Big|_{x \to g(y)} \frac{dx}{dy} dy \qquad \textit{differentiate -> reparametrize}$$

$$= \int \nabla_\theta \left(f(x;\theta) \Big|_{x \to g(y)} \frac{dx}{dy} \right) dy \qquad \textit{reparametrize -> differentiate}$$

STRATIFICATION

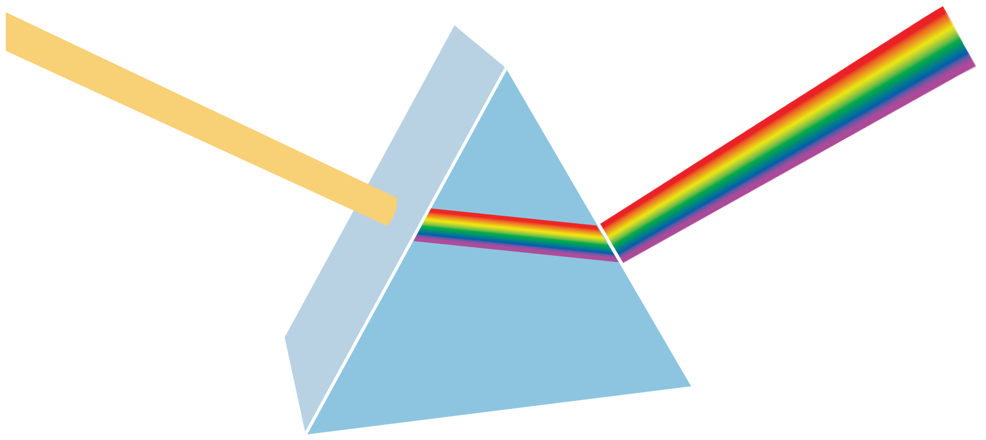
- low discrepancy samples, stratified sampling, etc can all be used
- stratifying differentiable rendering is an unsolved problem



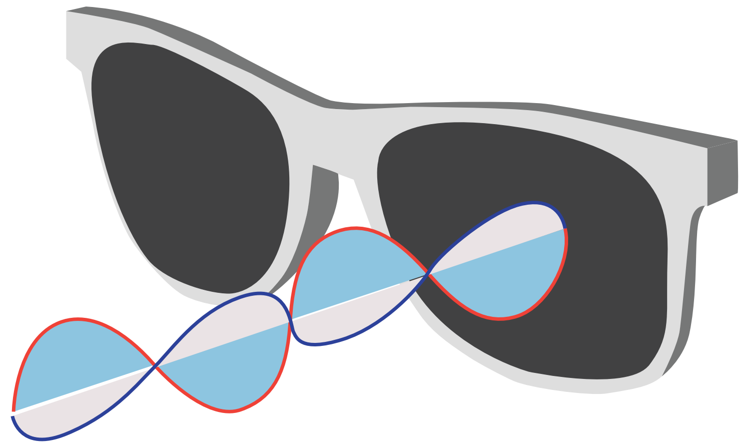
RETROSPECTIVE: MITSUBA

- **Mitsuba** is an open source physically-based rendering system
  - Common platform for rendering research (many papers at SIGGRAPH (Asia), EG, EGSR, etc., build on it)
  - ~ 120 plugins (highly modular architecture)
  - ~ 180'000 lines of C++ code

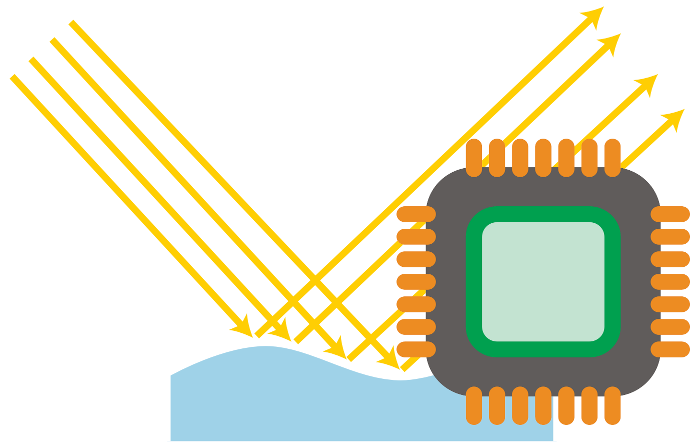
- **BUT**: did *not* provide a number of key features:



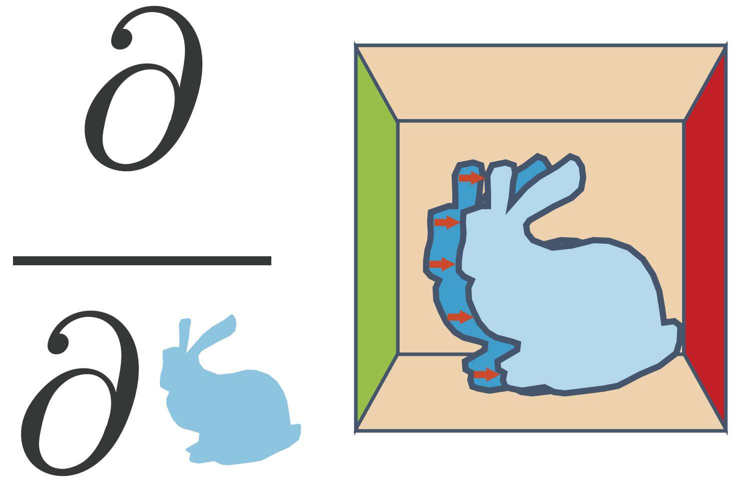
Spectral rendering



Polarization

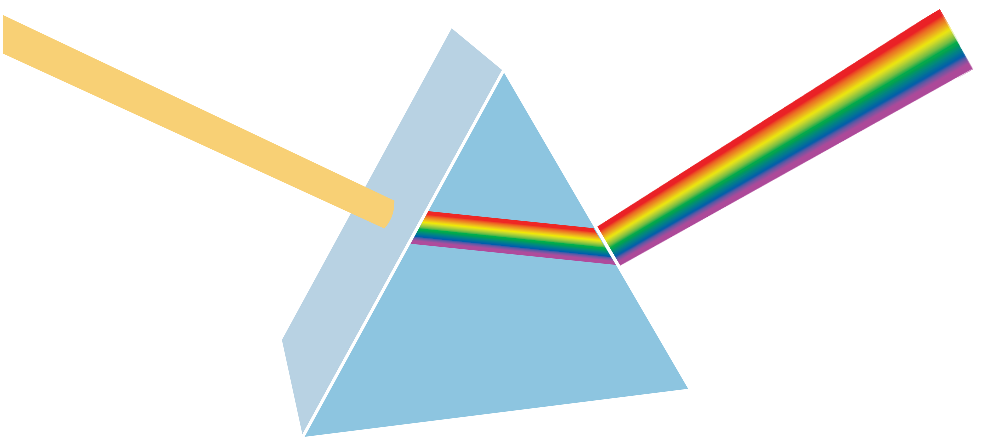
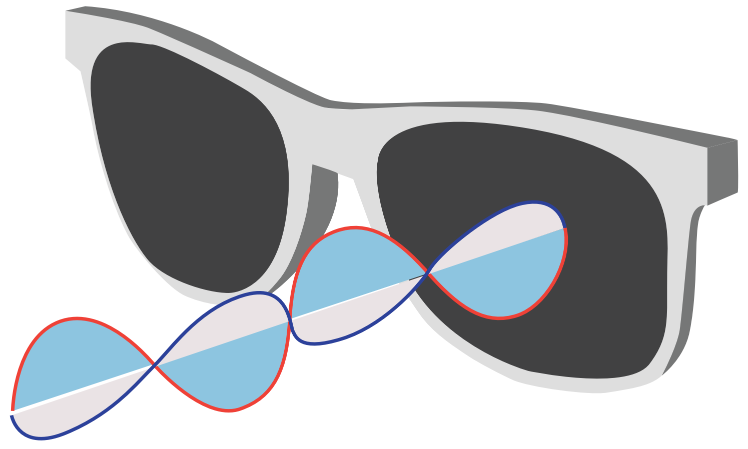
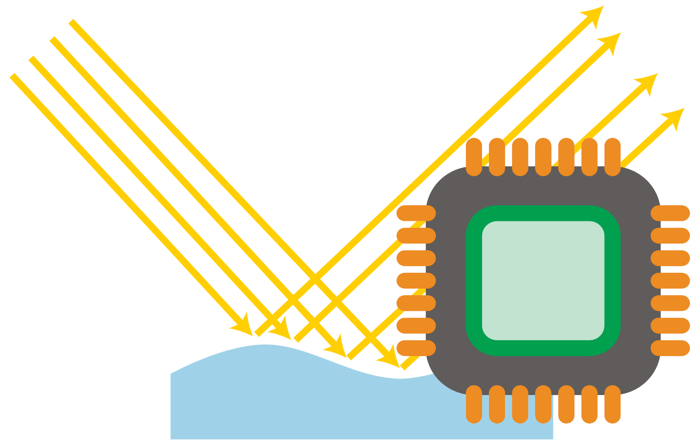
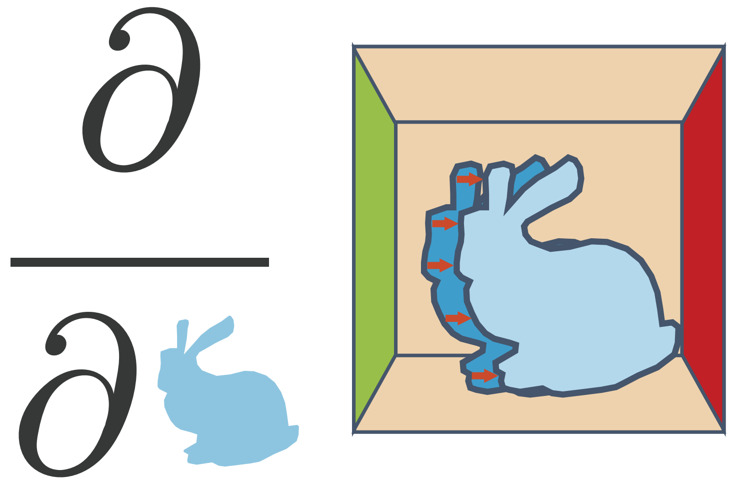


Vectorization



Differentiable rendering

WHAT TO DO?

- **Hack Mitsuba to support all of these features?**
  - Not a good idea: each change touches almost every file of the renderer.
  - Want to support various combinations as well

- **Create new programming language for developing rendering systems?**
  - Could deal with variants using automated program transformations.
  - Don’t have the manpower for such an effort.

OUR APPROACH: PROGRAM LIFTING

- **Let the type system do all the hard work**
  - Write 1 generic implementation and create variants by substituting types.

- **Key C++ features that enable this**
  - Templates
  - Variadic templates
  - Compile-time computation
  - **if constexpr (…) { }**

ENOKI

float

↓

CUDAArray<float>

↓

DiffArray<CUDAArray<float>>

↓

Spectrum<DiffArray<CUDAArray<float>>>

↓

MuellerMatrix<Spectrum<DiffArray<CUDAArray<float>>>>

MITSUBA 2 ARCHITECTURE

Generic rendering algorithms

Float type | **Spectrum type**

Enoki

Composable array types

MITSUBA 2 ARCHITECTURE

Generic plugins

Perspective sensor	Plastic BSDF	Rough conductor BSDF	Path tracing integrator	Area emitter	Environment emitter	Blackbody spectrum

Derived types & data structures
Intersection, sample record, etc

Float type
float, Packet<float>, DiffArray<...>

Spectrum type
Spectrum<Float, 4>, MuellerMatrix<...>

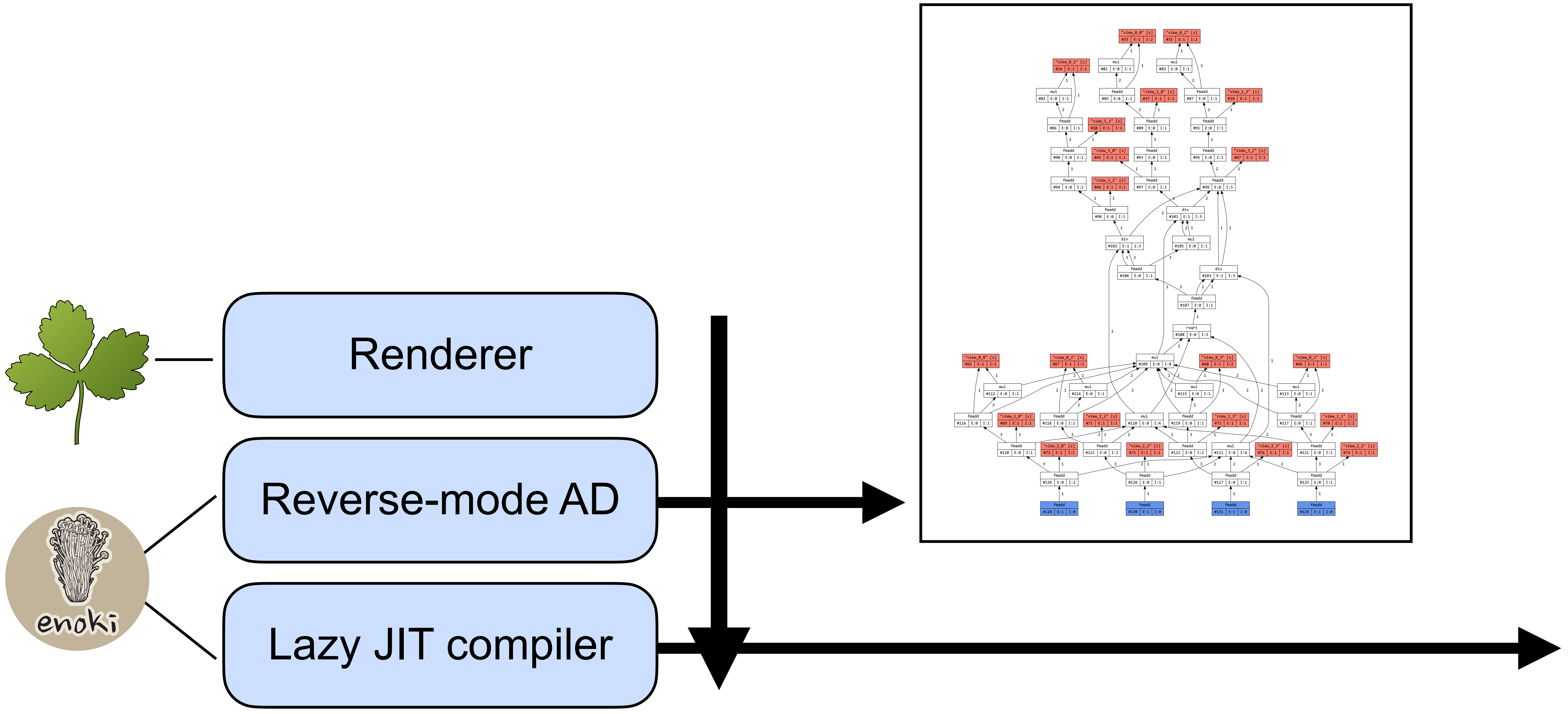
Routing layer

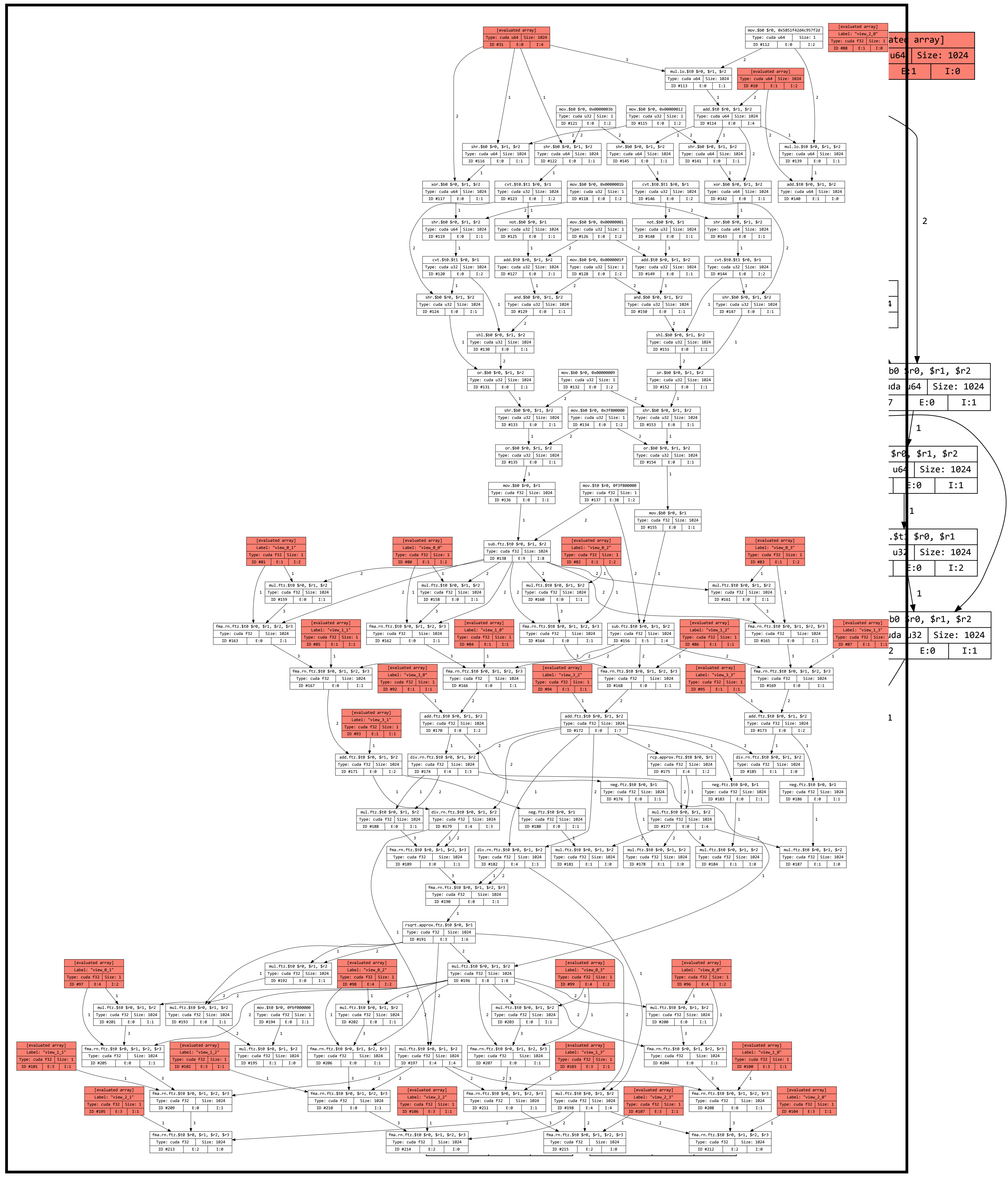
Scalar backend	Vector backends	CUDA backend	Autodiff backend

enoki

DIFFERENTIABLE RENDERING IN MITSUBA 2

```
Point2f sample = sampler->next_2d();

Ray3f ray = camera->sample_ray(sample);

SurfaceInteraction3f si = scene->ray_intersect(ray)

BSDFSample3f bsdf_sample = si.bsdf->sample(sampler.next_2d())
```

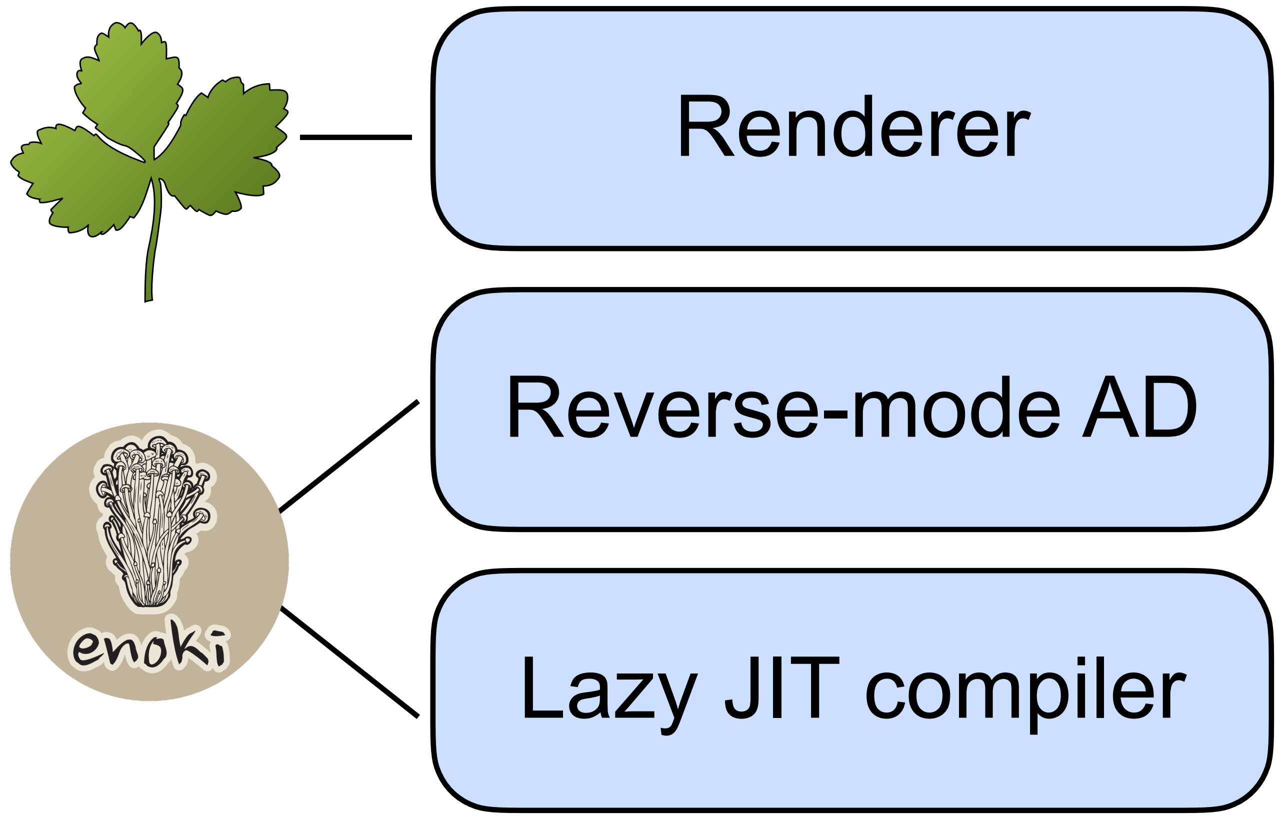




DIFFERENTIABLE RENDERING IN MITSUBA 2

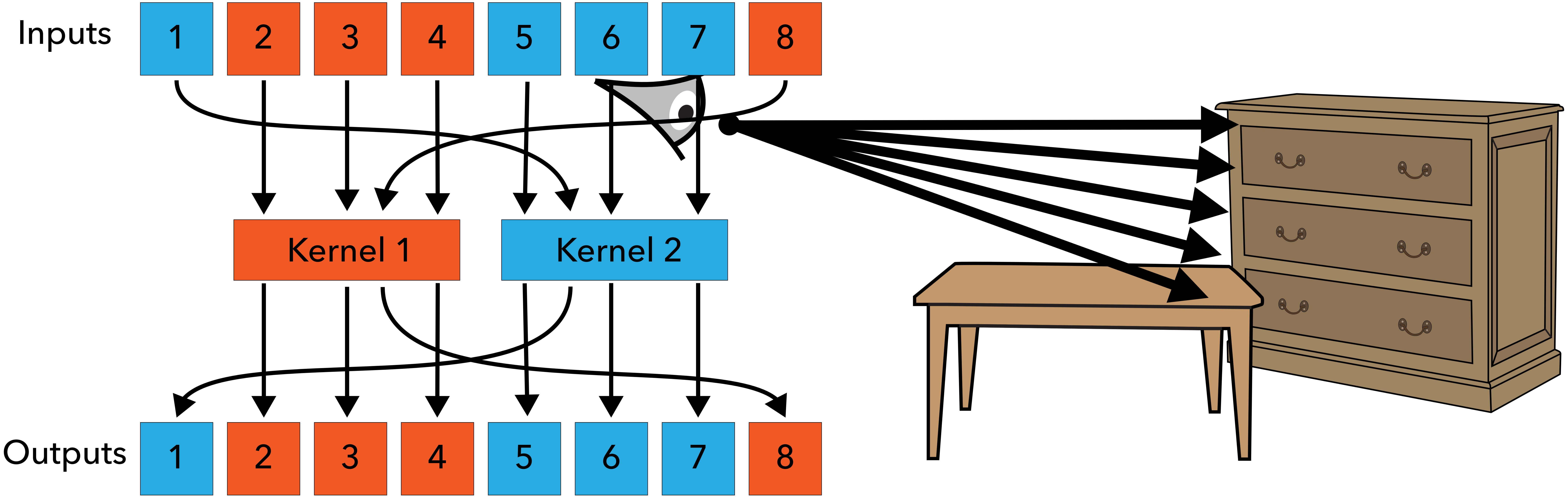
```
Point2f sample = sampler->next_2d();
Ray3f ray = camera->sample_ray(sample);
SurfaceInteraction3f si = scene->ray_intersect(ray)
BSDFSample3f bsdf_sample = si.bsdf->sample(sampler.next_2d())
```

- Compilation is *fast* (~100 *us*) (just hash table lookups + string concatenation)
- PTX (CUDA), soon: LLVM (CPU)
- Caches compiled kernels
- Can prototype rendering code in Jupyter notebooks with reasonable performance.



DIFFERENTIABLE RENDERING IN MITSUBA 2

```
Point2f sample = sampler->next_2d();

Ray3f ray = camera->sample_ray(sample);

SurfaceInteraction3f si = scene->ray_intersect(ray)

BSDFSample3f bsdf_sample = si.bsdf->sample(sampler.next_2d())
```



USAGE IN PYTHON

```python
import enoki as ek
import mitsuba
mitsuba.set_variant('gpu_autodiff_rgb')
```

```python
from mitsuba.core import Float, Thread
from mitsuba.core.xml import load_file
from mitsuba.python.util import traverse
from mitsuba.python.autodiff import render, write_bitmap, Adam
```

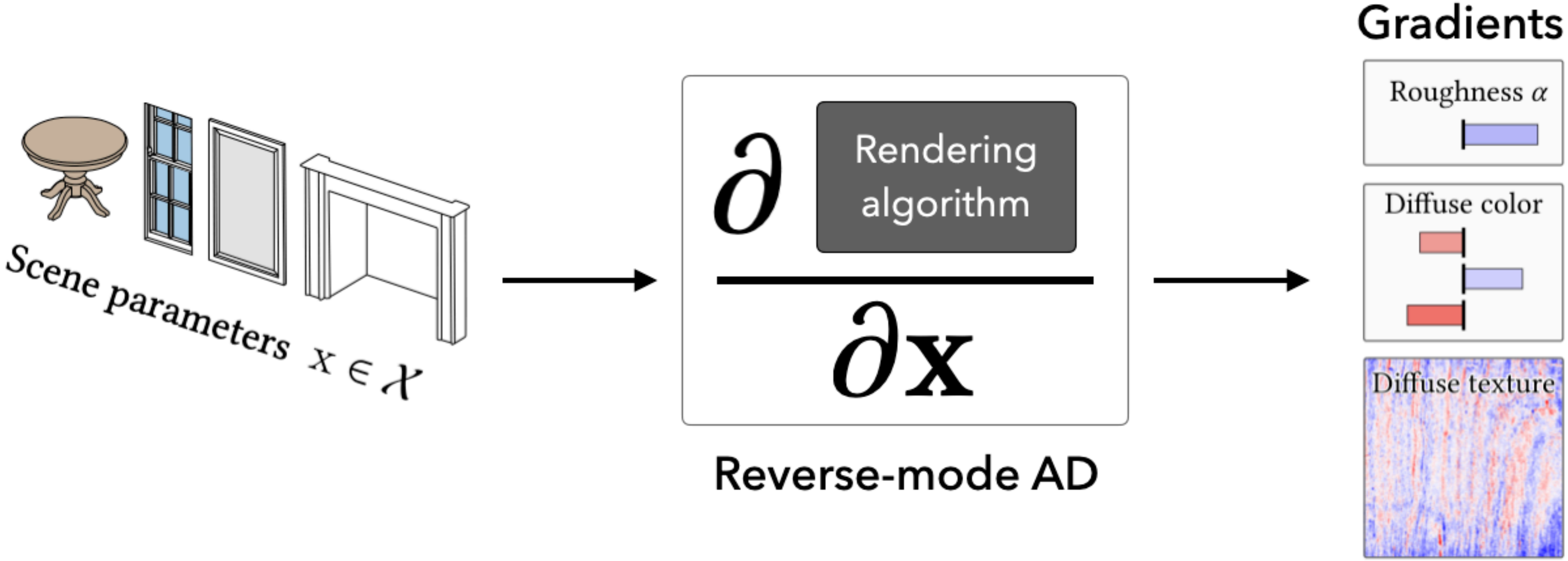
```python
# Load example scene
Thread.thread().file_resolver().append('bunny')
scene = load_file('bunny/bunny.xml')
```

```python
# Find differentiable scene parameters
params = traverse(scene)
```

USAGE IN PYTHON

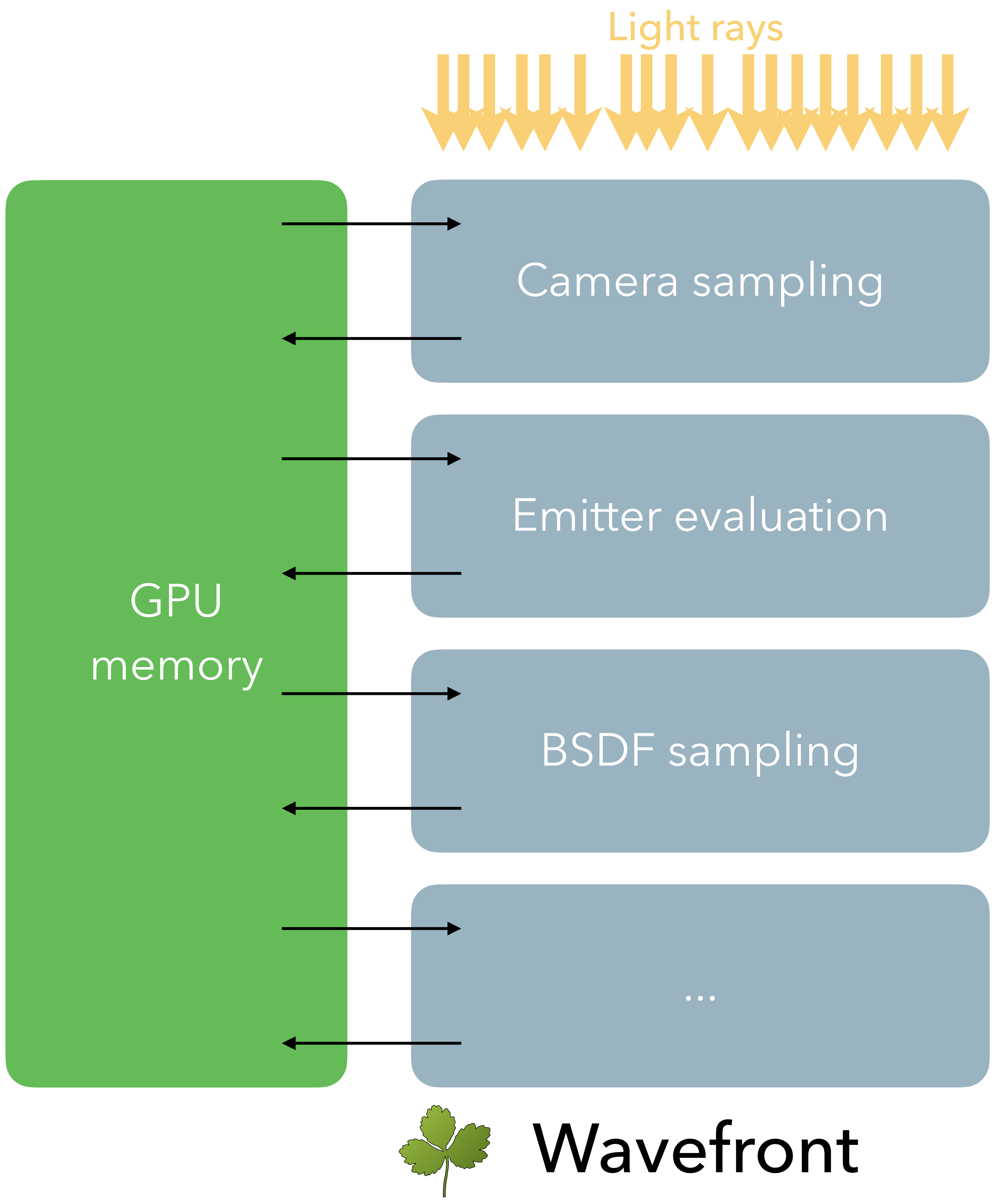
```
opt = Adam(params, lr=.02)

for it in range(100):
    image = render(scene, optimizer=opt, unbiased=True, spp=1)
    write_bitmap('out_%03i.png' % it, image, crop_size)

    ob_val = ek.hsum(ek.sqr(image - image_ref)) / len(image)

    ek.backward(ob_val)

    opt.step()
```

DIFFERENTIABLE RENDERING IN MITSUBA 2

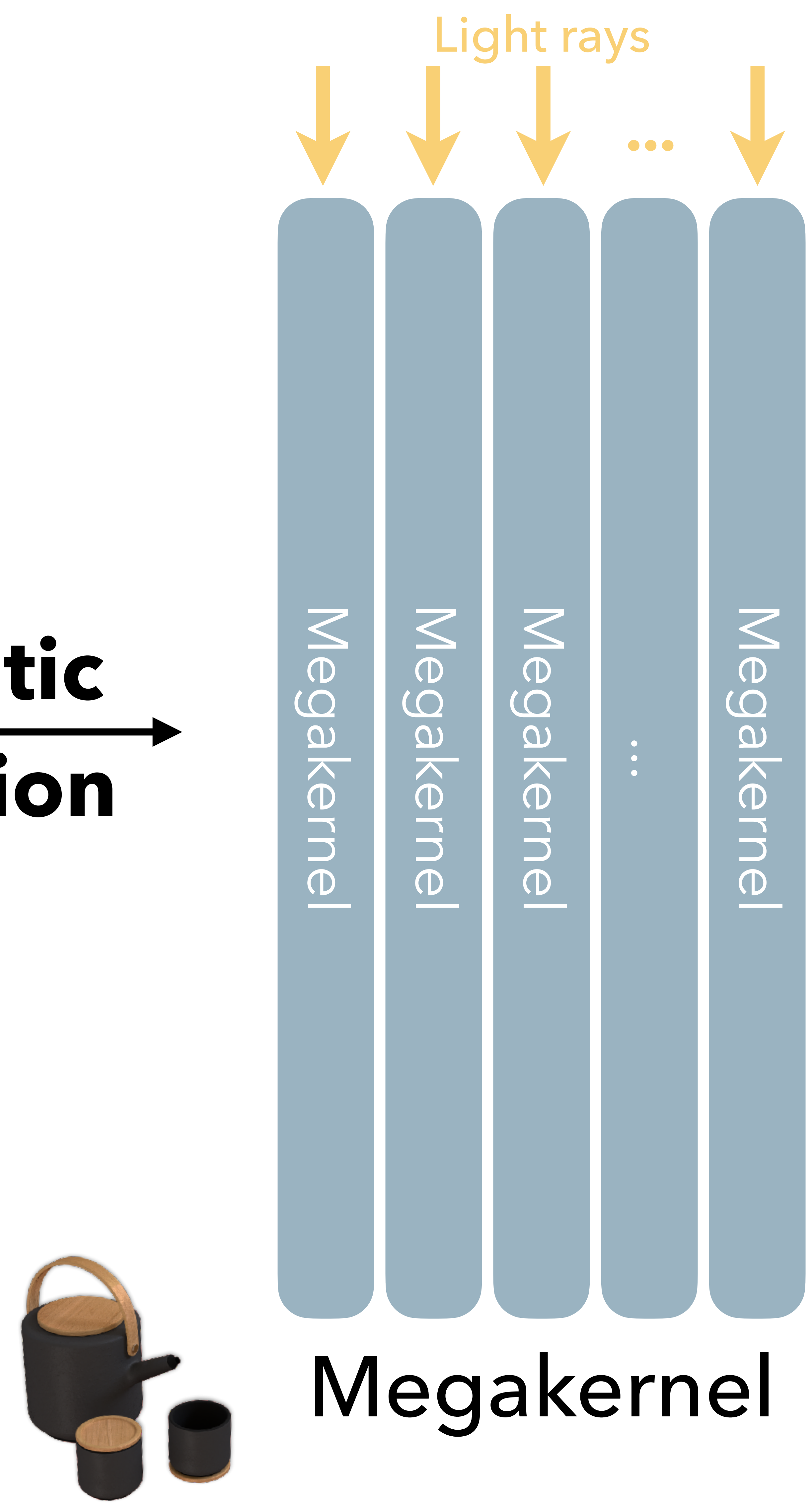


DIFFERENTIABLE RENDERING IN MITSUBA 2

WAVEFRONT VS MEGAKERNEL

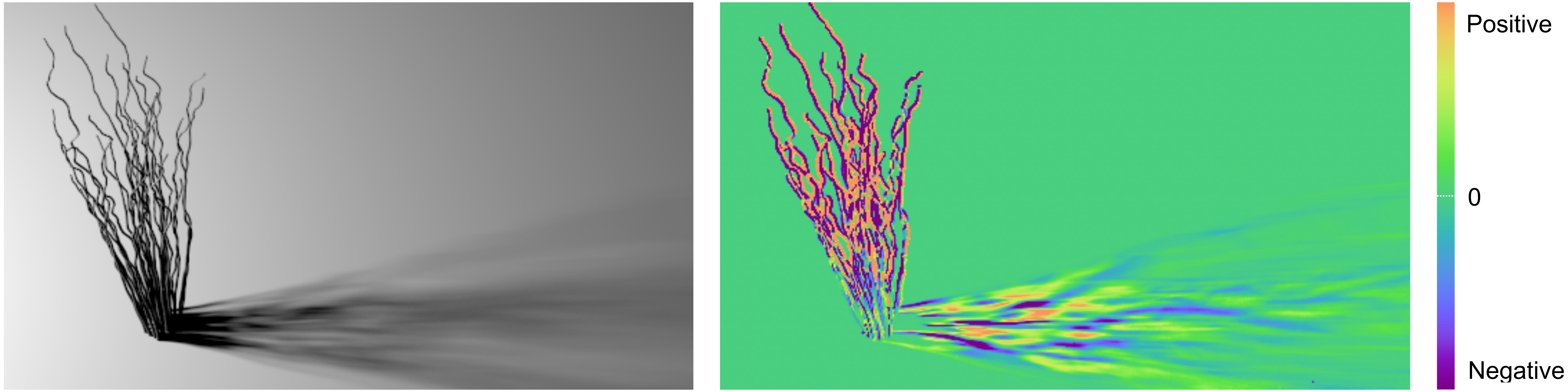


Automatic conversion



DIRECT ILLUMINATION USING PATH-SPACE FORMULATION

Parameter: rotation angle of the object



- Implementing differentiable *direct illumination* using the path-space formulation

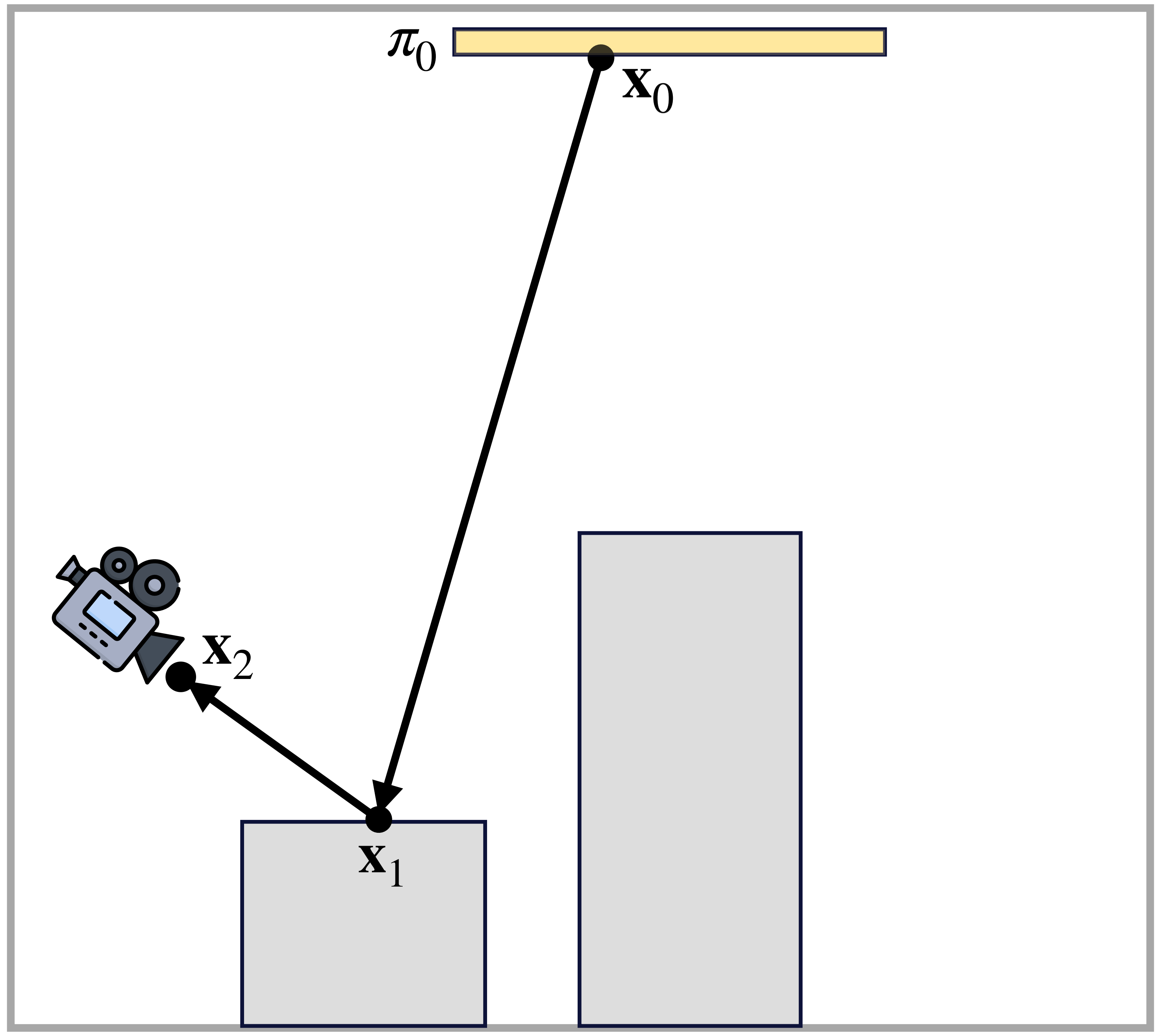
Still nontrivial with complex geometry & motion!

Spatial-form direct-illumination integral

$$I_j = \int_{\mathcal{M}(\pi)^3} \underbrace{f_j(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2)}_{= \bar{\mathbf{x}}} \, \mathrm{d}\mu(\bar{\mathbf{x}})$$

f_j : measurement contribution for pixel j, $\mathcal{M}$: scene geometry

π controls the position of the area light

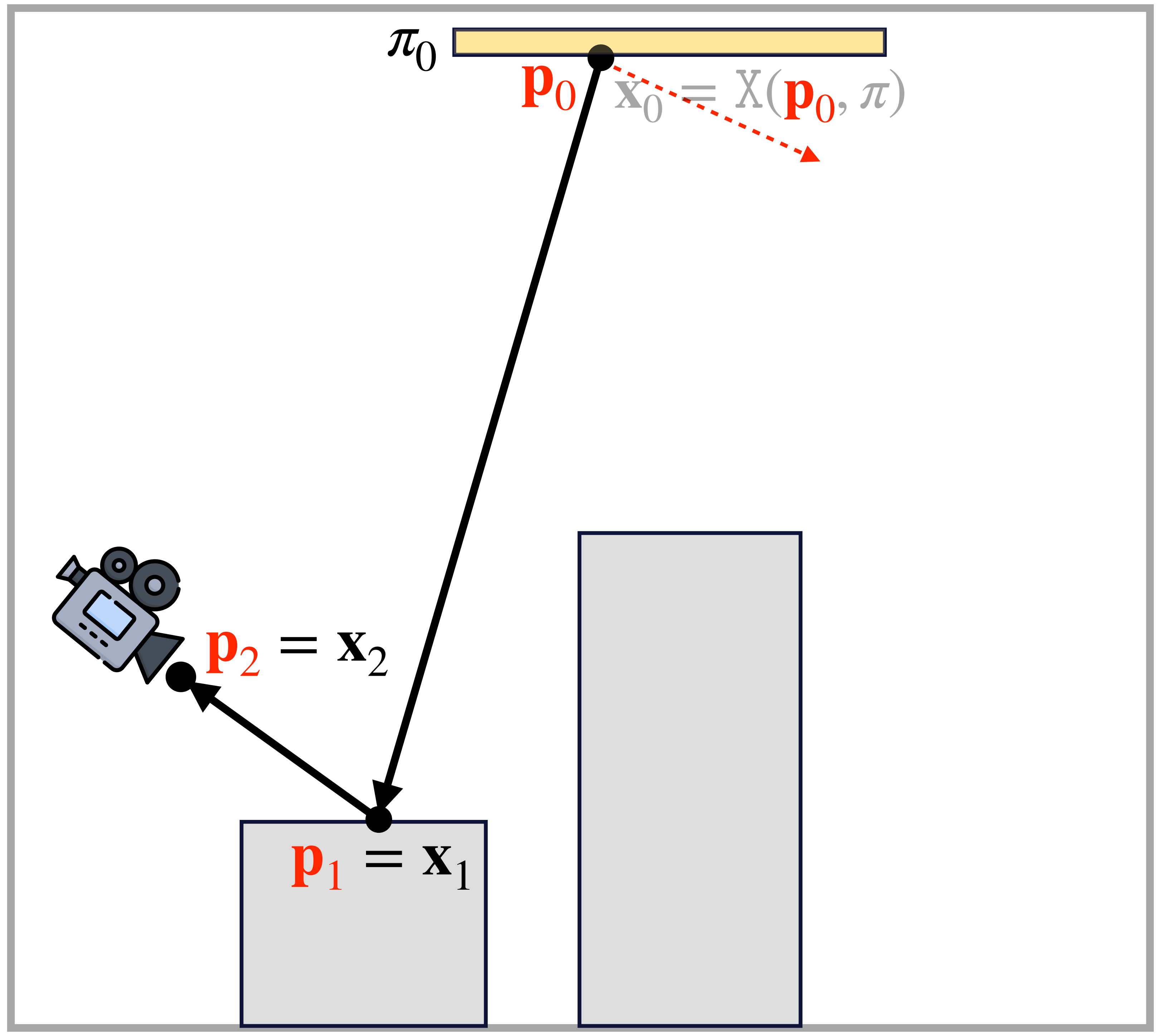


Material-form direct-illumination integral

$$I_j = \int_{\mathcal{M}_0^3} \underbrace{f_j(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right|}_{= f_j(\bar{\mathbf{p}})} \mathrm{d}\mu(\bar{\mathbf{p}})$$

Change of variable: $\mathbf{x}_i = \mathtt{X}(\mathbf{p}_i, \pi)$ with $\mathcal{M}_0 = \mathcal{M}(\pi_0)$

π controls the position of the area light



Material-form direct-illumination integral

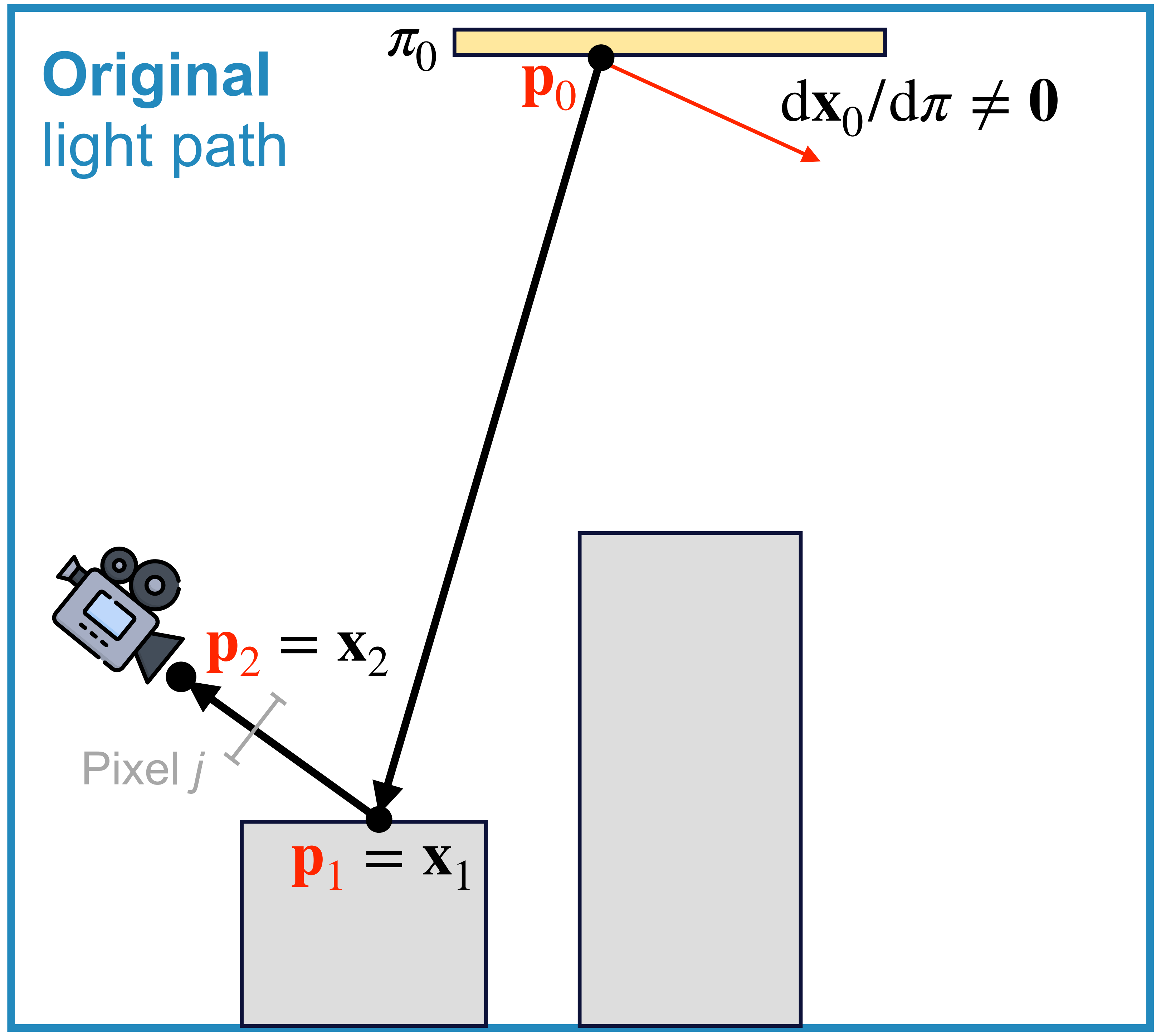
$$I_j = \int_{\mathcal{M}_0^3} \underbrace{f_j(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right|}_{= f_j(\bar{\mathbf{p}})} \mathrm{d}\mu(\bar{\mathbf{p}})$$

Change of variable: $\mathbf{x}_i = \mathrm{X}(\mathbf{p}_i, \pi)$ with $\mathcal{M}_0 = \mathcal{M}(\pi_0)$

Material-form *differential* direct-illumination integral

$$\frac{\mathrm{d}I_j}{\mathrm{d}\pi} = \underbrace{\int_{\mathcal{M}_0^3} \frac{\mathrm{d}f_j}{\mathrm{d}\pi}(\bar{\mathbf{p}})\, \mathrm{d}\mu(\bar{\mathbf{p}})}_{\text{Interior integral}} + \underbrace{\int_{\partial\mathcal{M}_0^3} g_j(\bar{\mathbf{p}})\, \mathrm{d}\mu'(\bar{\mathbf{p}})}_{\text{Boundary integral}}$$

π controls the position of the area light



- Consider the problem of estimating $\mathrm{d}I_j/\mathrm{d}\pi\,\big|_{\pi=\pi_0}$
  - $\mathbf{x}_0$ equals $\mathbf{p}_0$ in *value* but has nonzero derivative:

    $$\mathrm{d}\mathbf{x}_0/\mathrm{d}\pi = \mathrm{d}\mathrm{X}(\mathbf{p}_0, \pi)/\mathrm{d}\pi$$

  - This can affect $\mathrm{d}f_j(\bar{\mathbf{p}})/\mathrm{d}\pi$ via:
    - Emission $L_e(\mathbf{x}_0 \to \mathbf{x}_1)$
    - Geometric term $G(\mathbf{x}_0 \leftrightarrow \mathbf{x}_1)$
    - BSDF $f_s(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2)$
    - Jacobian determinant $|dA(\mathbf{x}_0)/dA(\mathbf{p}_0)|$

DIRECT ILLUMINATION USING PATH-SPACE FORMULATION

Material-form direct-illumination integral

$$I_j = \int_{\mathcal{M}_0^3} \underbrace{f_j(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right|}_{= f_j(\bar{\mathbf{p}})} \mathrm{d}\mu(\bar{\mathbf{p}})$$

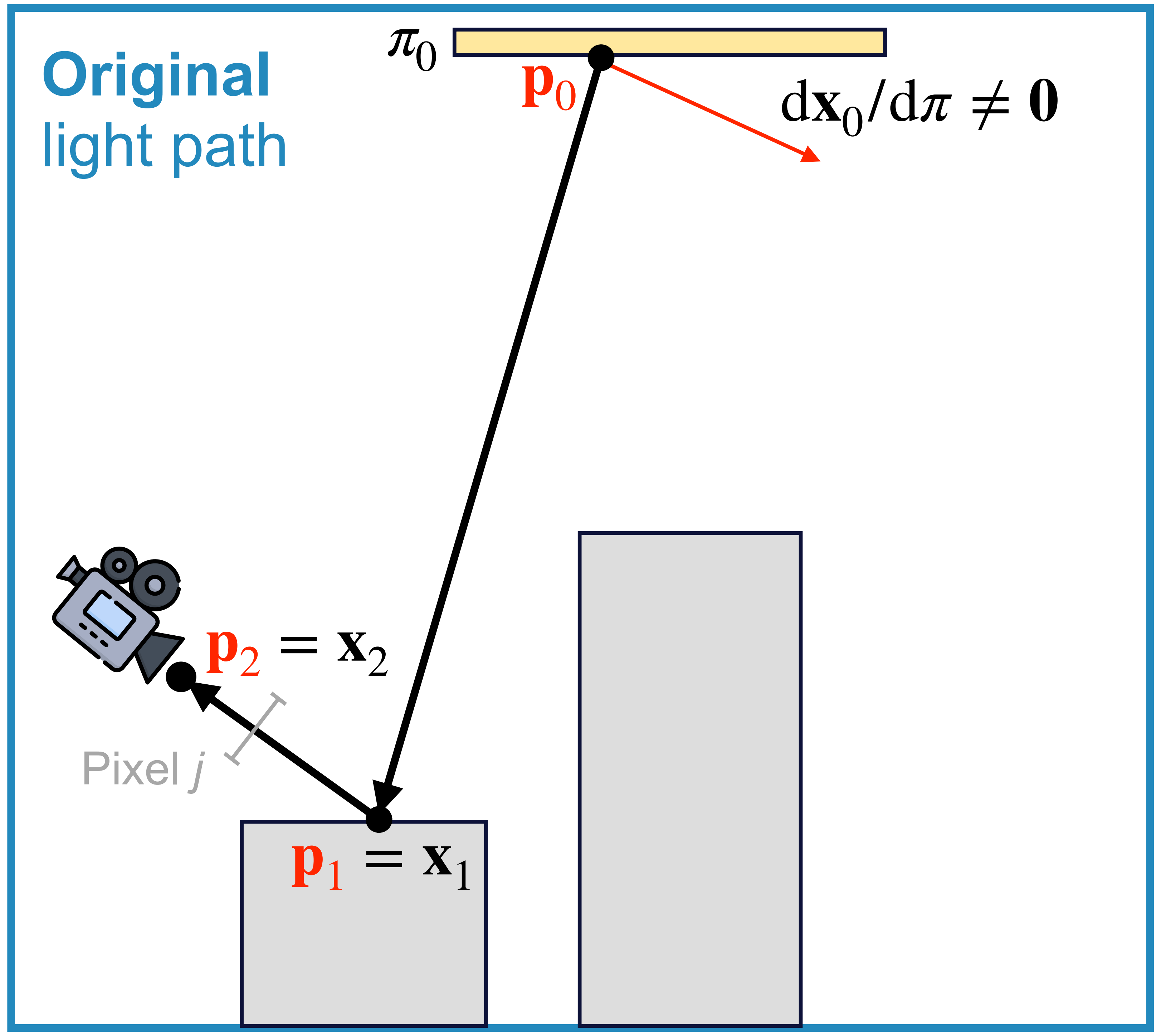
Change of variable: $\mathbf{x}_i = \mathrm{X}(\mathbf{p}_i, \pi)$ with $\mathcal{M}_0 = \mathcal{M}(\pi_0)$

Material-form *differential* direct-illumination integral

$$\frac{\mathrm{d}I_j}{\mathrm{d}\pi} = \boxed{\int_{\mathcal{M}_0^3} \frac{\mathrm{d}f_j}{\mathrm{d}\pi}(\bar{\mathbf{p}})\, \mathrm{d}\mu(\bar{\mathbf{p}})} + \int_{\partial\mathcal{M}_0^3} g_j(\bar{\mathbf{p}})\, \mathrm{d}\mu'(\bar{\mathbf{p}})$$

Interior integral

π controls the position of the area light



- Consider the problem of estimating $\mathrm{d}I_j/\mathrm{d}\pi\,\big|_{\pi=\pi_0}$
  - Interior integral estimated using *standard methods*:
    - Using scene geometry $\mathcal{M}_0 = \mathcal{M}(\pi_0)$
    - Sample camera ray (through pixel *j*) that gives $\mathbf{p}_2$, $\mathbf{p}_1$
    - Sample $\mathbf{p}_0$ using MIS (area + solid angle sampling)

How $\mathbf{p}_0$ is drawn does NOT affect
$\mathrm{d}\mathbf{x}_0/\mathrm{d}\pi = \mathrm{d}\mathrm{X}(\mathbf{p}_0, \pi)/\mathrm{d}\pi$

Material-form direct-illumination integral

$$I_j = \int_{\mathcal{M}_0^3} \underbrace{f_j(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right|}_{= f_j(\bar{\mathbf{p}})} \mathrm{d}\mu(\bar{\mathbf{p}})$$

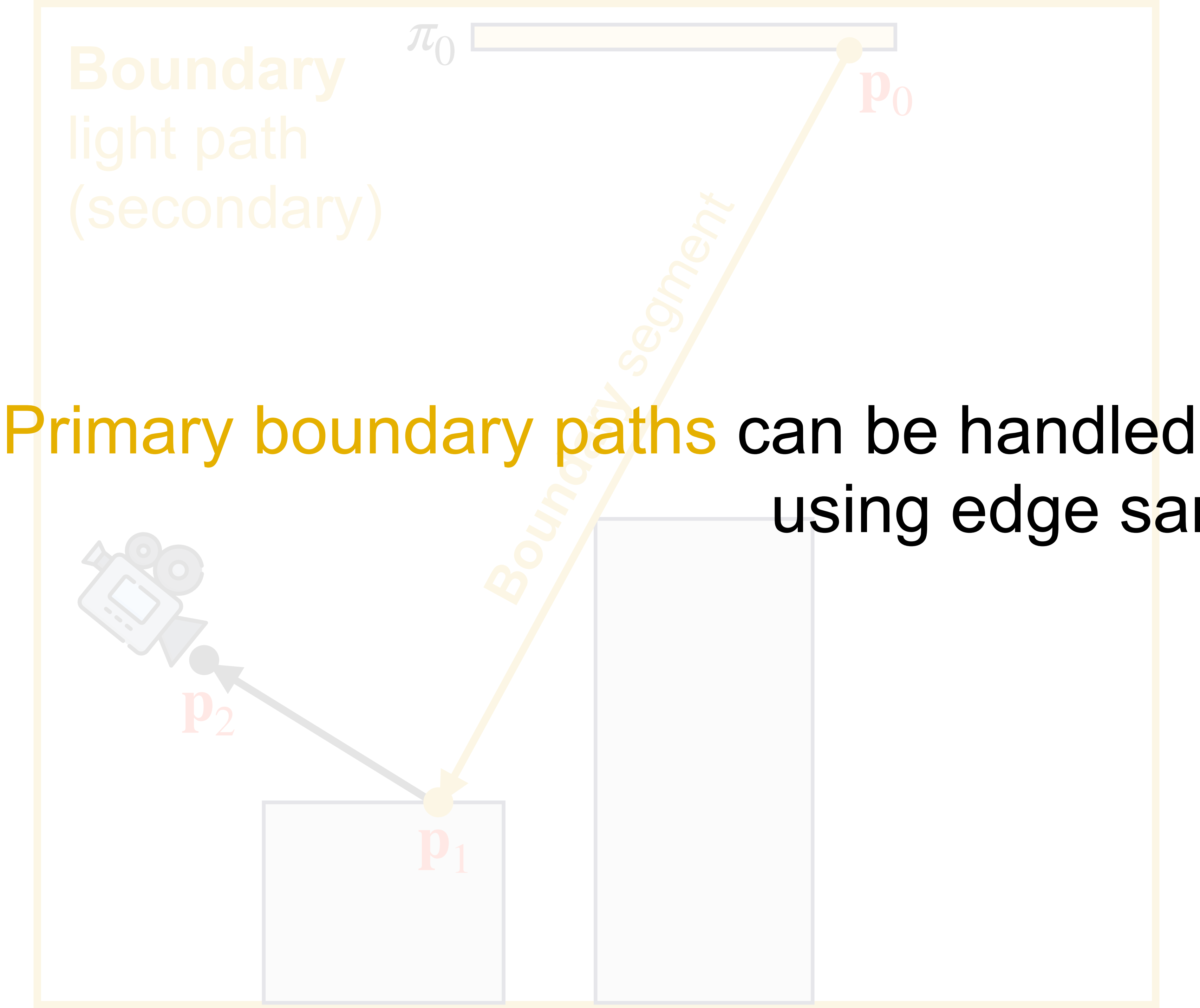
Change of variable: $\mathbf{x}_i = \mathrm{X}(\mathbf{p}_i, \pi)$ with $\mathcal{M}_0 = \mathcal{M}(\pi_0)$

Material-form *differential* direct-illumination integral

$$\frac{\mathrm{d}I_j}{\mathrm{d}\pi} = \int_{\mathcal{M}_0^3} \frac{\mathrm{d}f_j}{\mathrm{d}\pi}(\bar{\mathbf{p}})\, \mathrm{d}\mu(\bar{\mathbf{p}}) + \boxed{\int_{\partial\mathcal{M}_0^3} g_j(\bar{\mathbf{p}})\, \mathrm{d}\mu'(\bar{\mathbf{p}})}$$

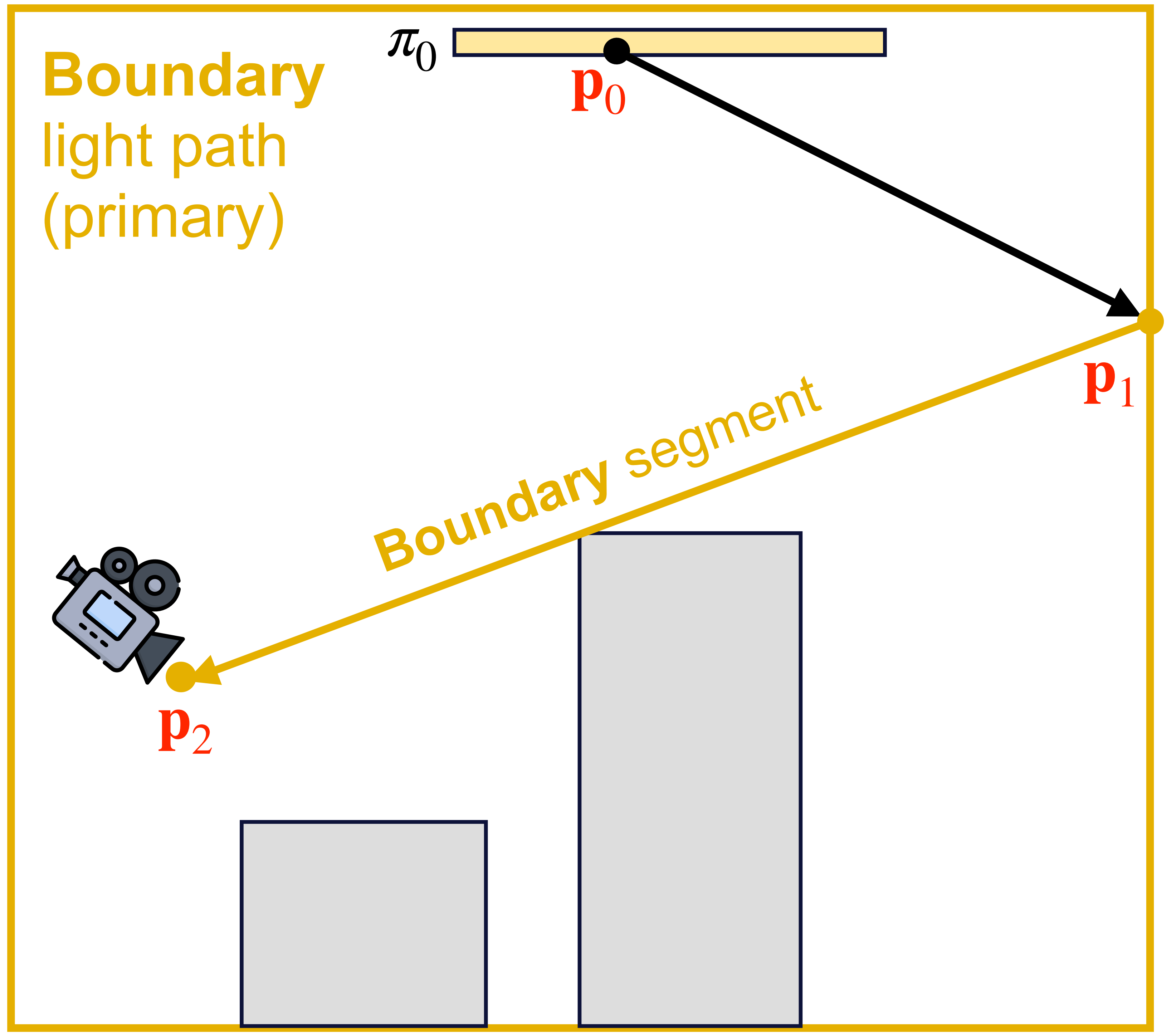
Boundary integral

π controls the position of the area light



Primary boundary paths can be handled easily using edge sampling

π controls the position of the area light



SIGGRAPH 2020 S2020.SIGGRAPH.ORG THINK BEYOND

Material-form direct-illumination integral

$$I_j = \int_{\mathcal{M}_0^3} \underbrace{f_j(\mathbf{x}_0 \to \mathbf{x}_1 \to \mathbf{x}_2) \left| \frac{\mathrm{d}\mu(\bar{\mathbf{x}})}{\mathrm{d}\mu(\bar{\mathbf{p}})} \right|}_{= f_j(\bar{\mathbf{p}})} \mathrm{d}\mu(\bar{\mathbf{p}})$$

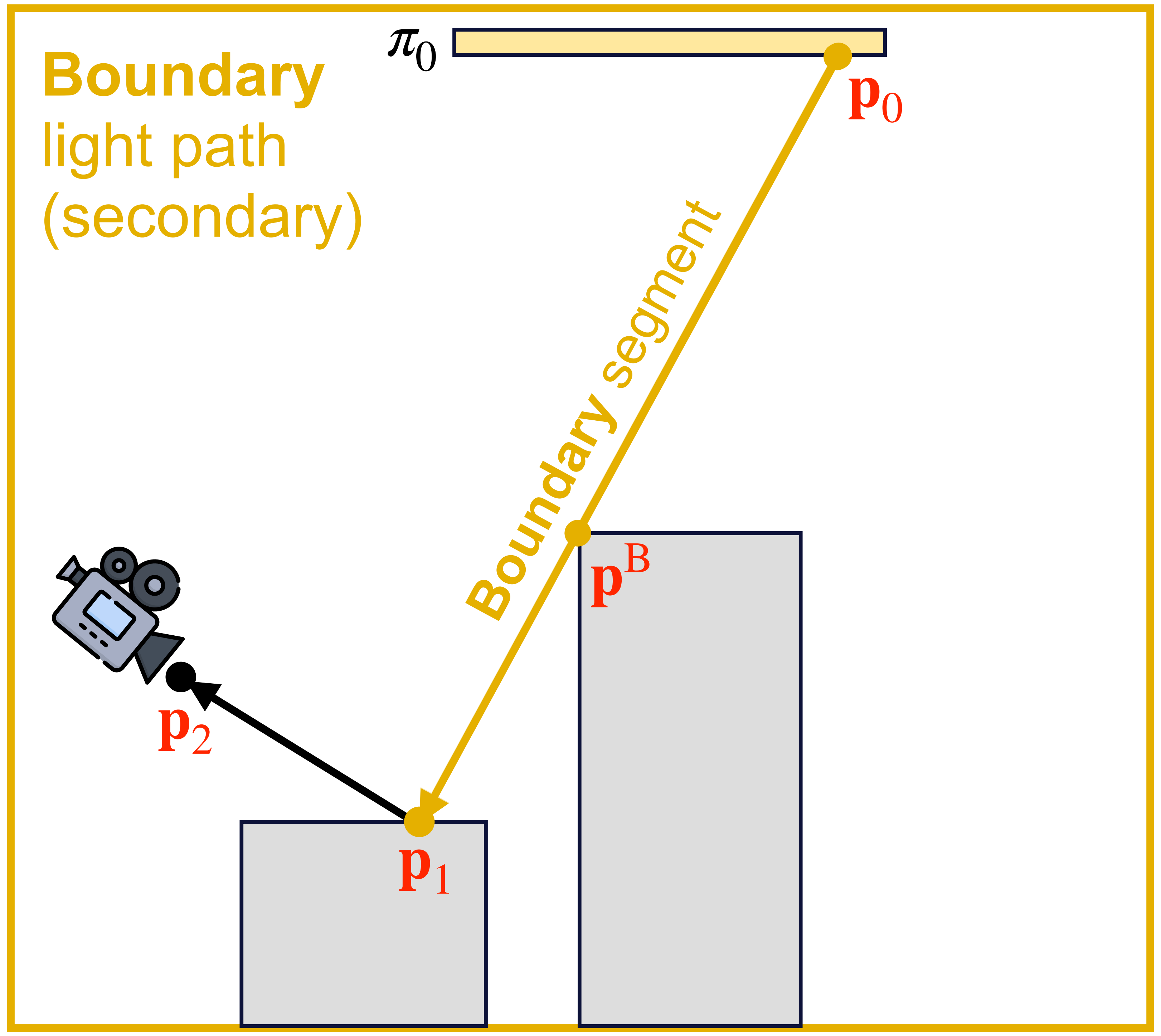
Change of variable: $\mathbf{x}_i = \mathrm{X}(\mathbf{p}_i, \pi)$ with $\mathcal{M}_0 = \mathcal{M}(\pi_0)$

Material-form *differential* direct-illumination integral

$$\frac{\mathrm{d}I_j}{\mathrm{d}\pi} = \int_{\mathcal{M}_0^3} \frac{\mathrm{d}f_j}{\mathrm{d}\pi}(\bar{\mathbf{p}})\,\mathrm{d}\mu(\bar{\mathbf{p}}) + \boxed{\int_{\partial\mathcal{M}_0^3} g_j(\bar{\mathbf{p}})\,\mathrm{d}\mu'(\bar{\mathbf{p}})}$$

Boundary integral

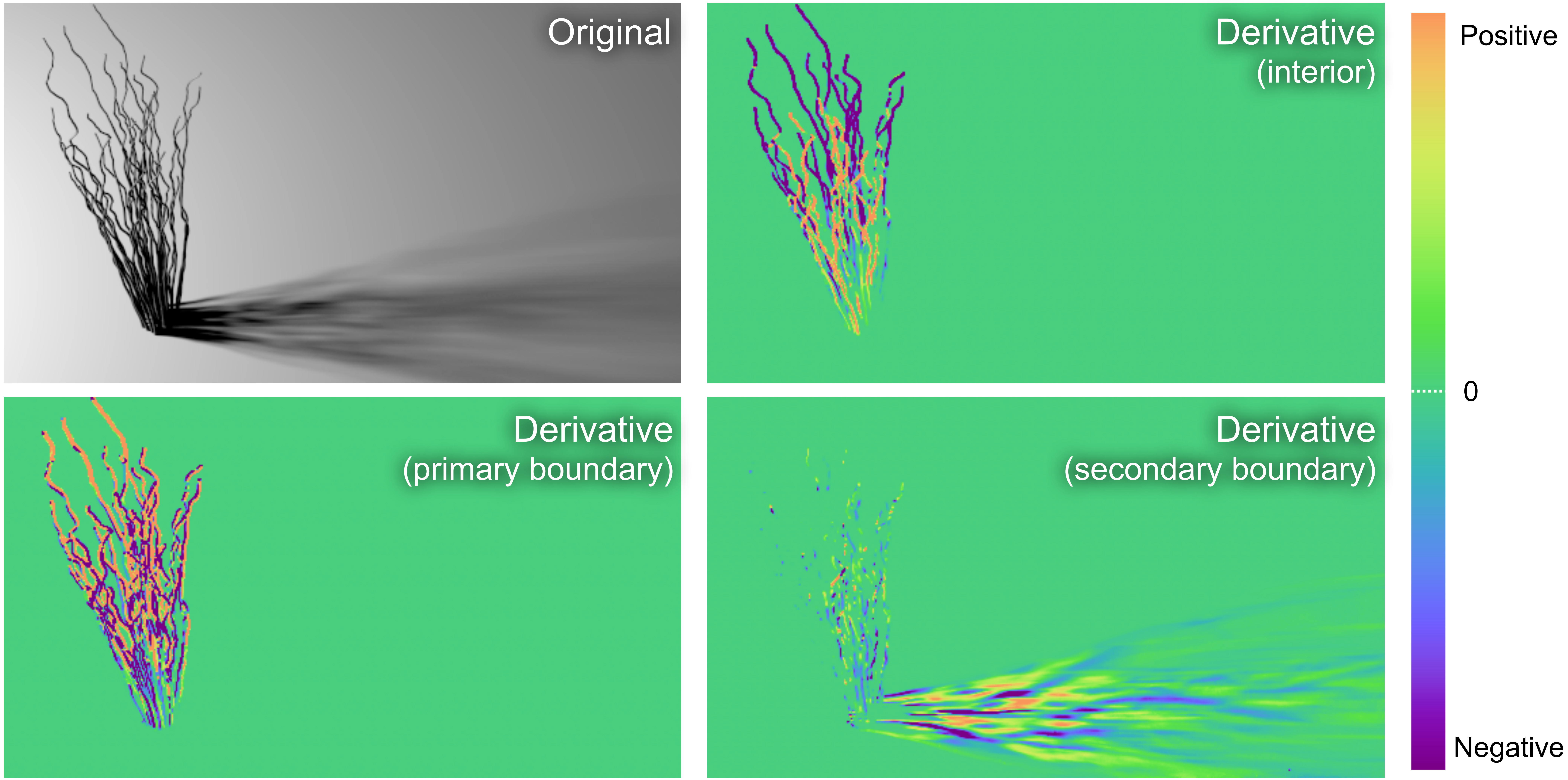
π controls the position of the area light



- Consider the problem of estimating $\mathrm{d}I_j/\mathrm{d}\pi\,\big|_{\pi=\pi_0}$
  - Secondary **boundary** light paths:
    - Determined uniquely by the boundary segment $\mathbf{p}_0\,\mathbf{p}_1$ (with pinhole camera + direct illumination)
  - The boundary segment $\mathbf{p}_0\,\mathbf{p}_1$:
    - Resides within a 3D manifold
    - For polygonal meshes: can be parameterized with $\mathbf{p}^B$ on a face edge (1D) + $\mathbf{p}_0$ on the light source (2D)
    - Should be *importance sampled* with a pdf $\propto g_j(\bar{\mathbf{p}})$
    - Sampling can be guided easily!

COMPONENT-WISE VISUALIZATIONS

Parameter: rotation angle of the object



CLOSING NOTES

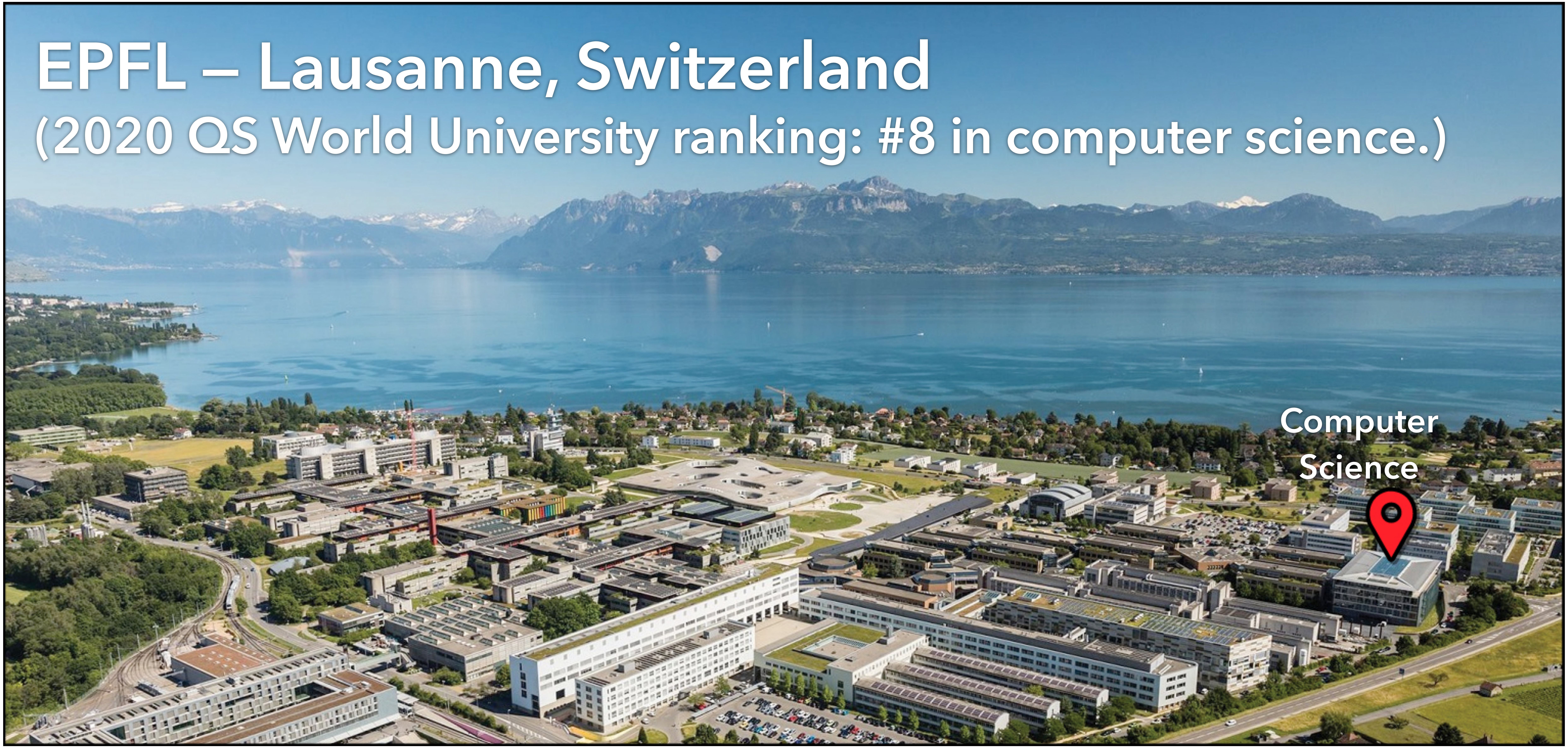
- Physics-based differentiable rendering is a rich topic, and we are just getting started

- Solving inverse-rendering problems is not only about differentiation
  - What loss to use?
  - How to avoid local minima?
  - How to handle non-differentiable things like mesh topology?
  - How to efficiently integrate physics-based rendering into machine learning pipelines?

- We look forward to future collaborations on all these topics!

PHD POSITIONS AT UC IRVINE

AVAILABLE POSITIONS (4 PHDS & 1 POSTDOC)



Apply to UCSD too!

apply to the graduate program/postdoc if you want to work on differentiable graphics!

Center for Visual Computing

UC San Diego

JACOBS SCHOOL OF ENGINEERING

faculty: Ravi Ramamoorthi, Henrik Jensen, David Kriegman, Manmohan Chandraker, Hao Su, Albert Chern, Nuno Vasconcelos, Xiaolong Wang, Thomas DeFanti, Jurgen Schulze, Zhuowen Tu, and … **me**!

THANK YOU

- Funding agencies

- Collaborators
  - Miika Aittala, Frédo Durand, Ioannis Gkioulekas, Nicolas Holzschuch, Jaakko Lehtinen, Guillaume Loubet, Bailey Miller, Merlin Nimier-David, Ravi Ramamoorthi, Delio Vicini, Lifan Wu, Kai Yan, Tizian Zeltner, Cheng Zhang, Changxi Zheng

- Course website: https://shuangz.com/courses/pbdr-course-sg20/