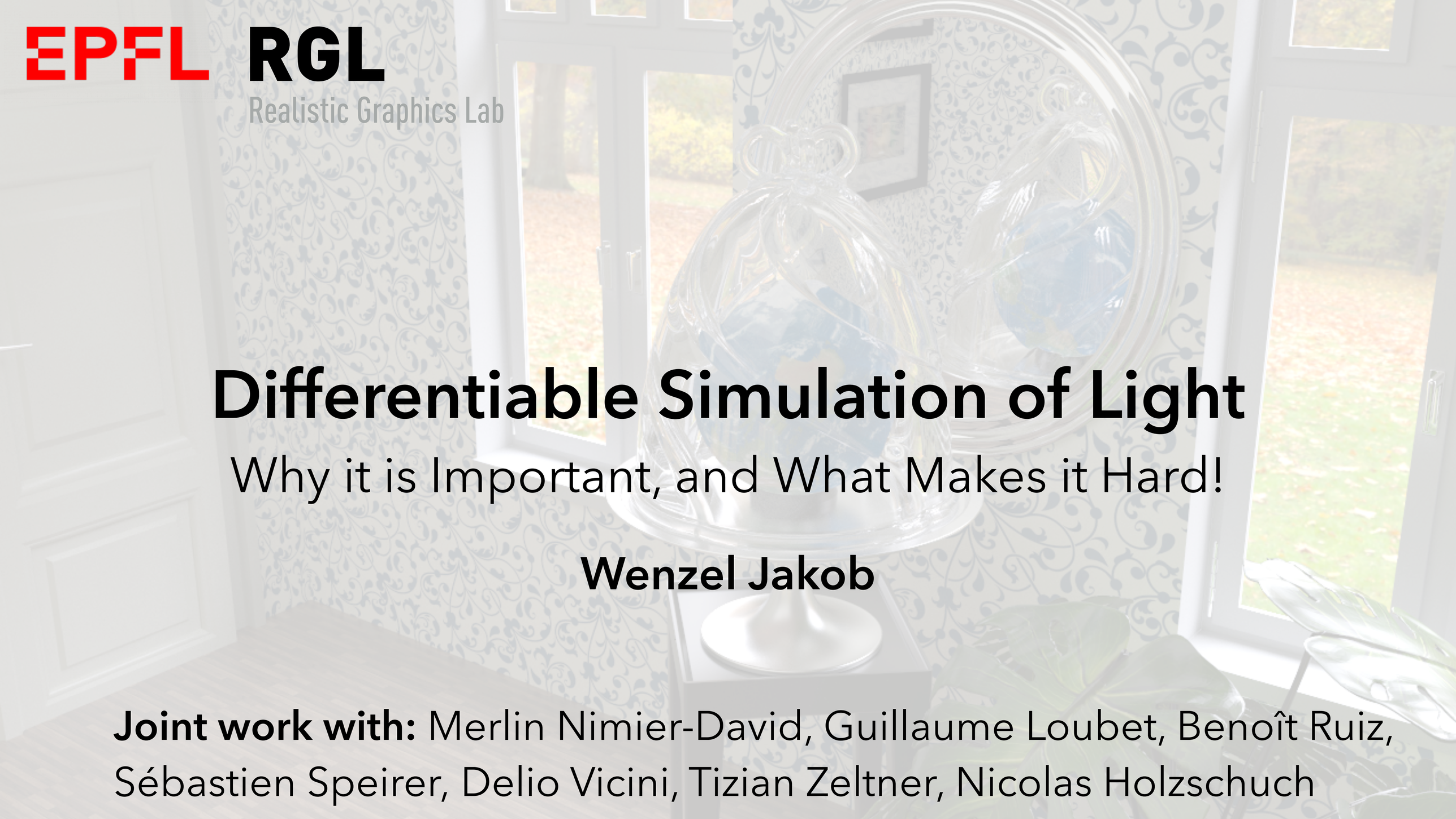


EPFL RGL
Realistic Graphics Lab

Differentiable Simulation of Light

Why it is Important, and What Makes it Hard!

Wenzel Jakob

Joint work with: Merlin Nimier-David, Guillaume Loubet, Benoît Ruiz, Sébastien Speirer, Delio Vicini, Tizian Zeltner, Nicolas Holzschuch

Agenda for today

Inverse rendering

Example problem

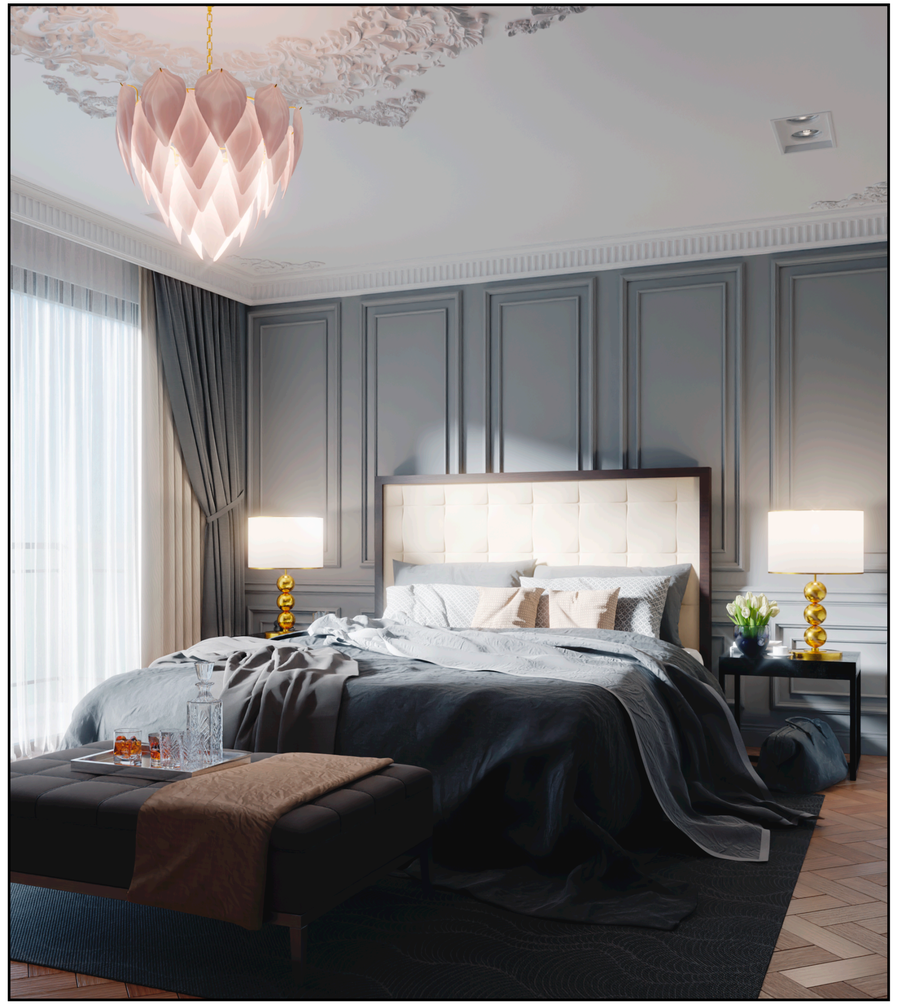
Differentiable rendering

Challenges

1. How to do this at all?
2. Efficiency
3. Discontinuities
4. Robustness

Forward vs. Inverse Rendering

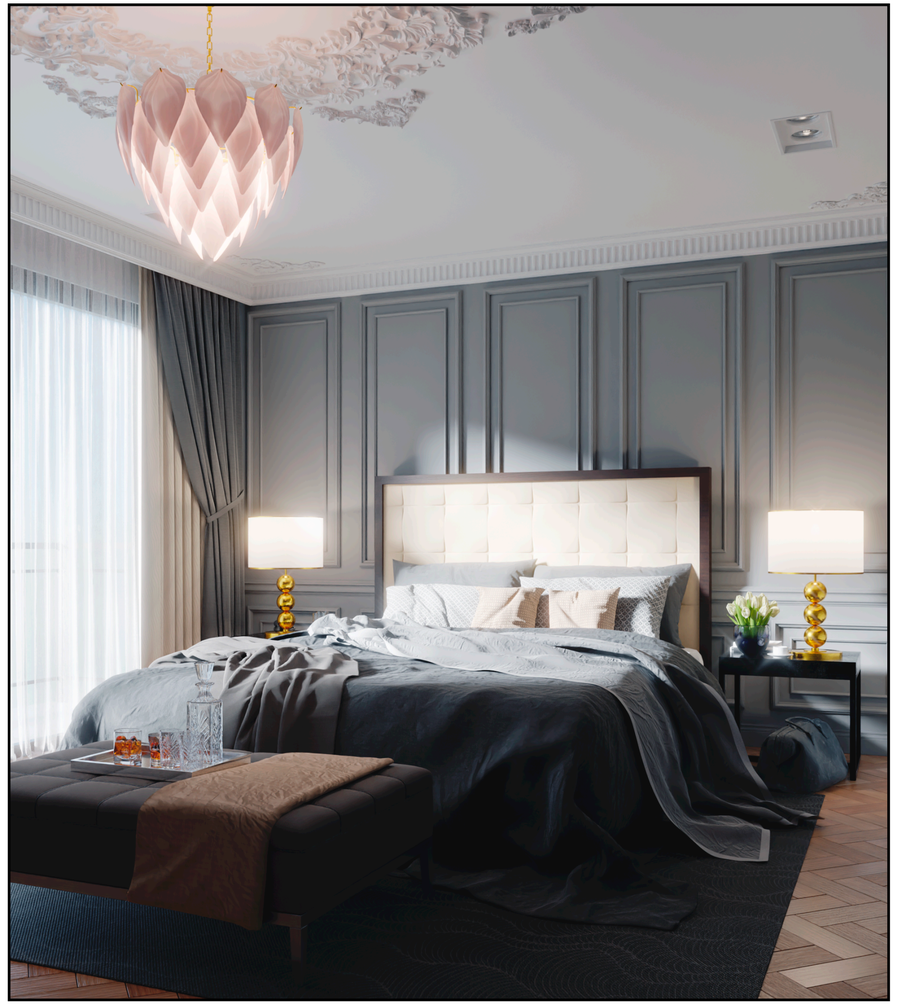
Geometry, materials, emitters, ...



Scene: "bed classic" from jiraniano

Forward vs. Inverse Rendering

Rendering



Geometry, materials, emitters, ...

Scene: "bed classic" from jiraniano

Forward vs. Inverse Rendering

Rendering

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

Geometry, materials, emitters, ...

Scene: "bed classic" from jiraniano

Forward vs. Inverse Rendering

Rendering

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

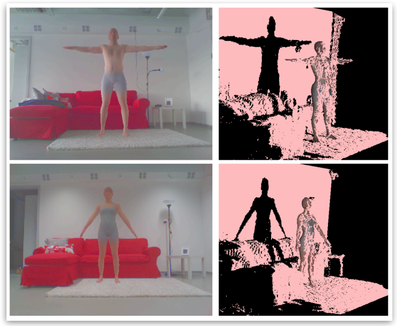
Inverse rendering

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

Geometry, materials, emitters, ...

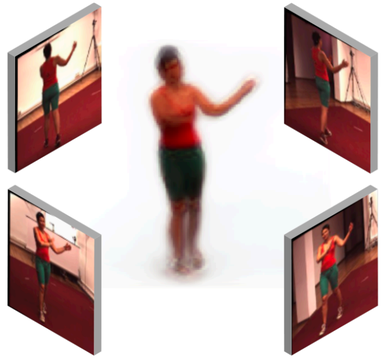
Scene: "bed classic" from jiraniano

Inverse rendering in computer vision

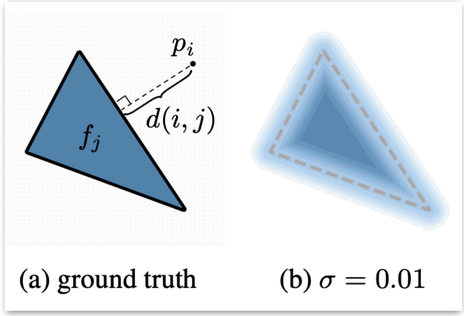


OpenDR: an Approximate Differentiable Renderer
[Loper et al. 2014]

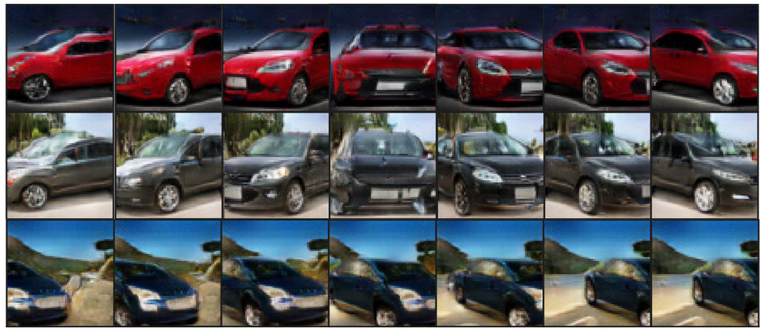
Neural 3D Mesh Renderer
[Kato et al. 2017]



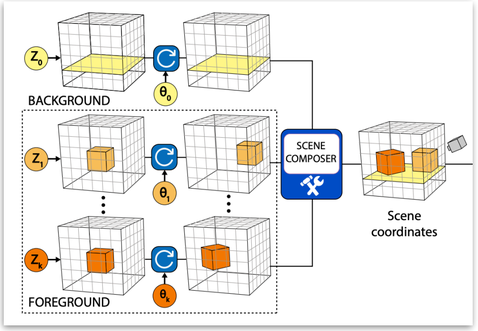
Unsupervised Geometry-Aware Representation for 3D Human Pose Estimation
[Rhodin et al., 2016]



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 of today's talk

- **Inverse rendering for realistic** $f(\mathbf{x})$

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

Focus of today's talk

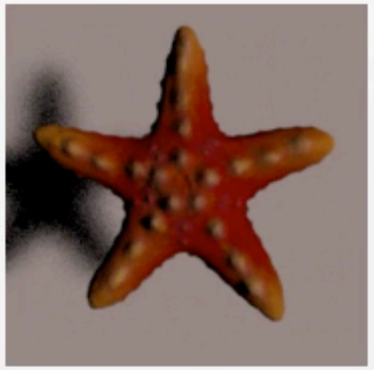
- **Inverse rendering for realistic** $f(\mathbf{x})$

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

- **No neural networks.**

  Shouldn't need them, we understand the underlying equations.
  (Of course you can still put neural networks around the renderer)

Shape & material reconstruction



Target

Target

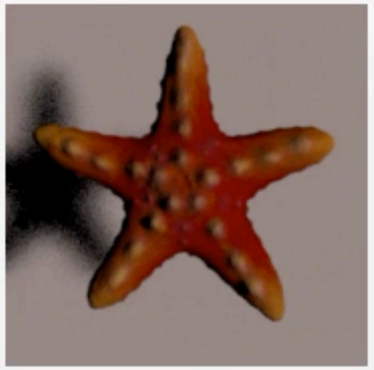
Target

Target

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

Shape & material reconstruction

Input scene



Target

Input scene

Target

Input scene

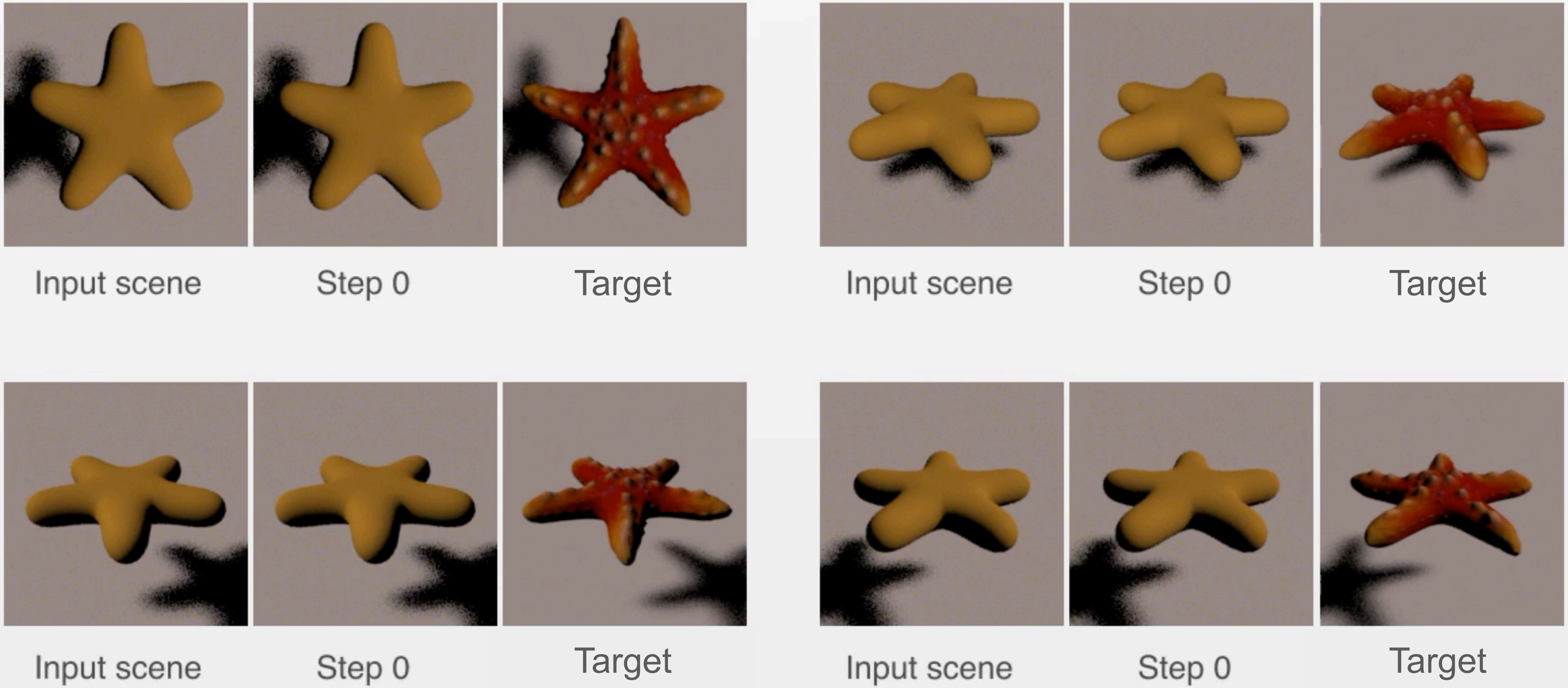
Target

Input scene

Target

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

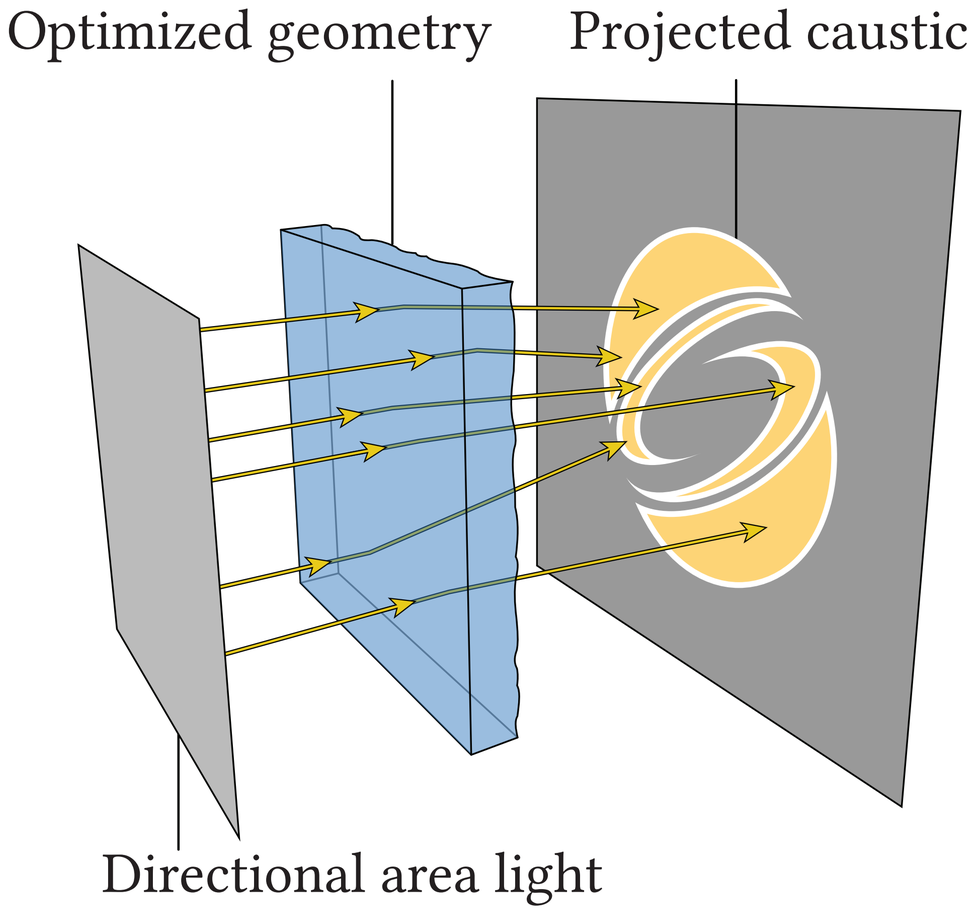
Shape & material reconstruction



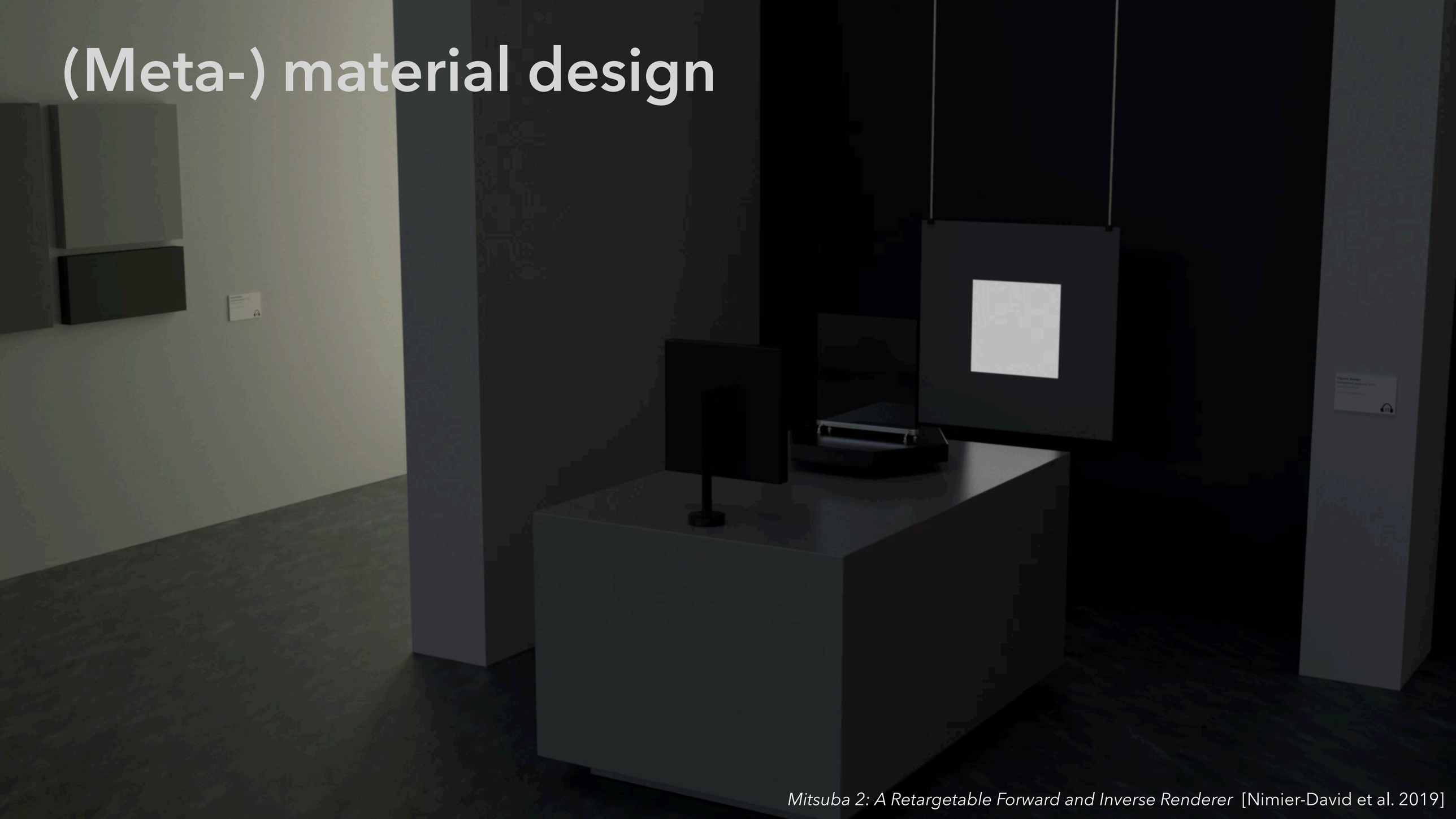
Reparameterizing discontinuous integrands for differentiable rendering [Loubet et al. 2019]

Application: caustic design

Schwartzburg et al. 2014

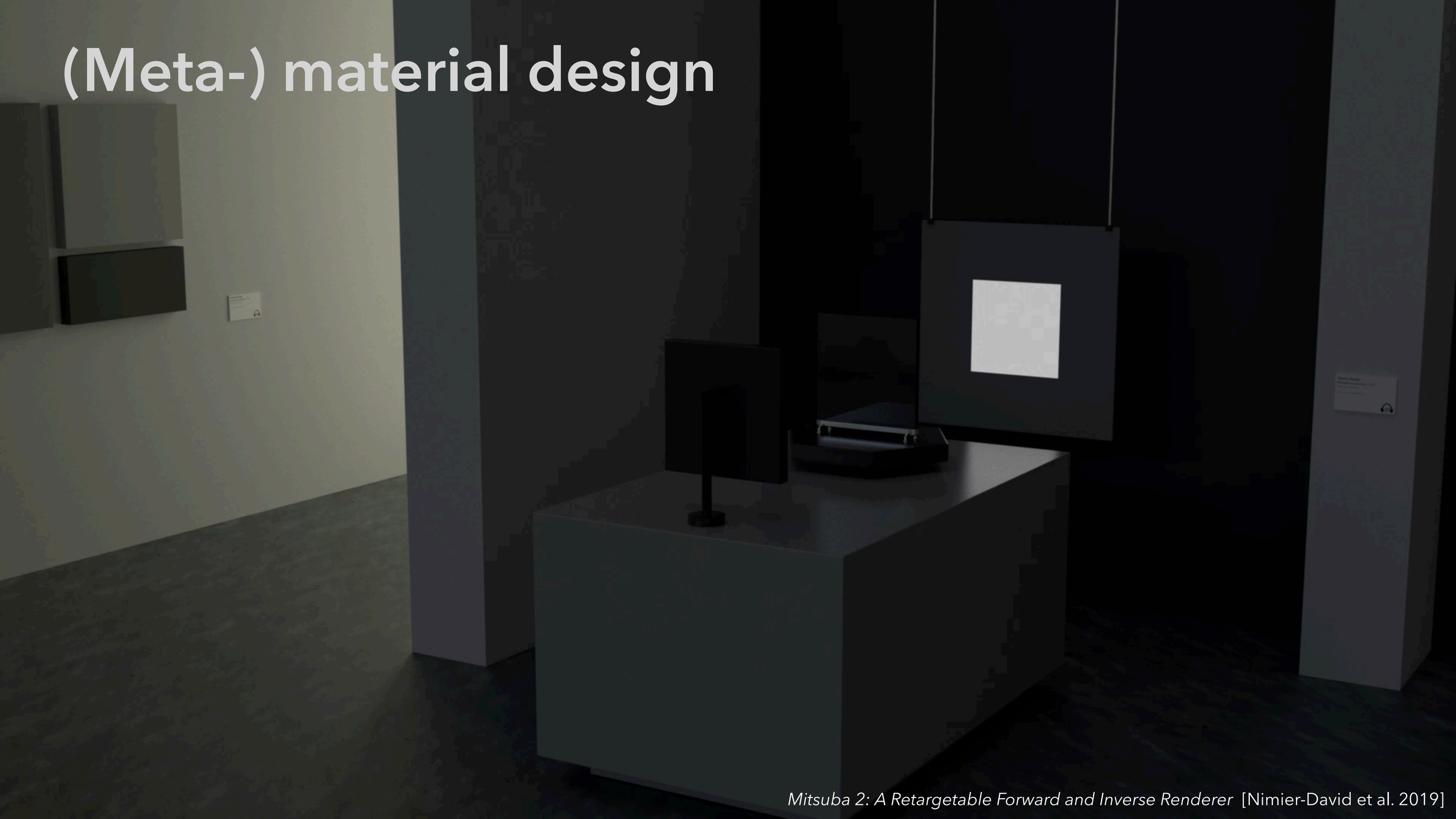


(Meta-) material design



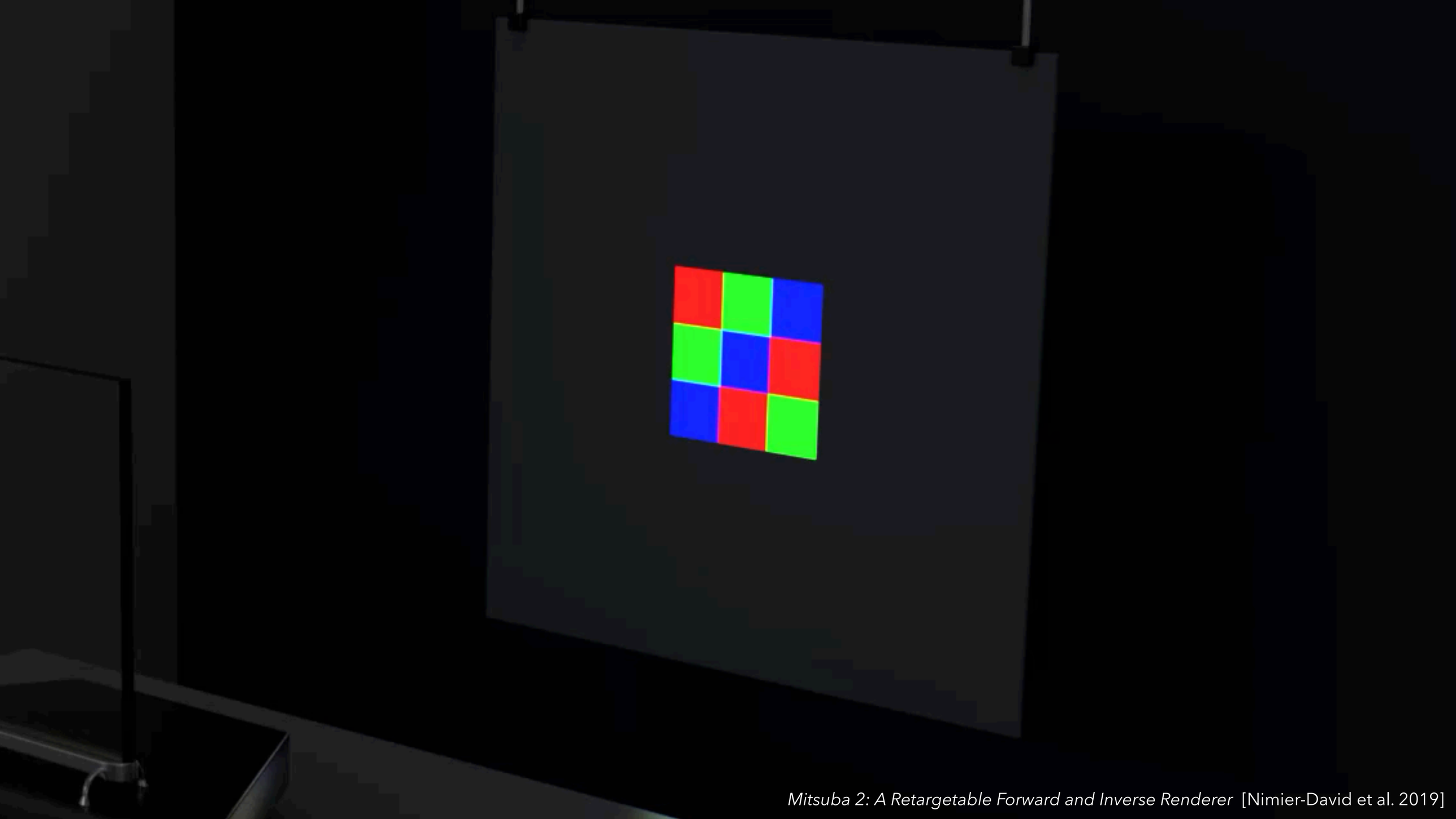
Mitsuba 2: A Retargetable Forward and Inverse Renderer [Nimier-David et al. 2019]

(Meta-) material design



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

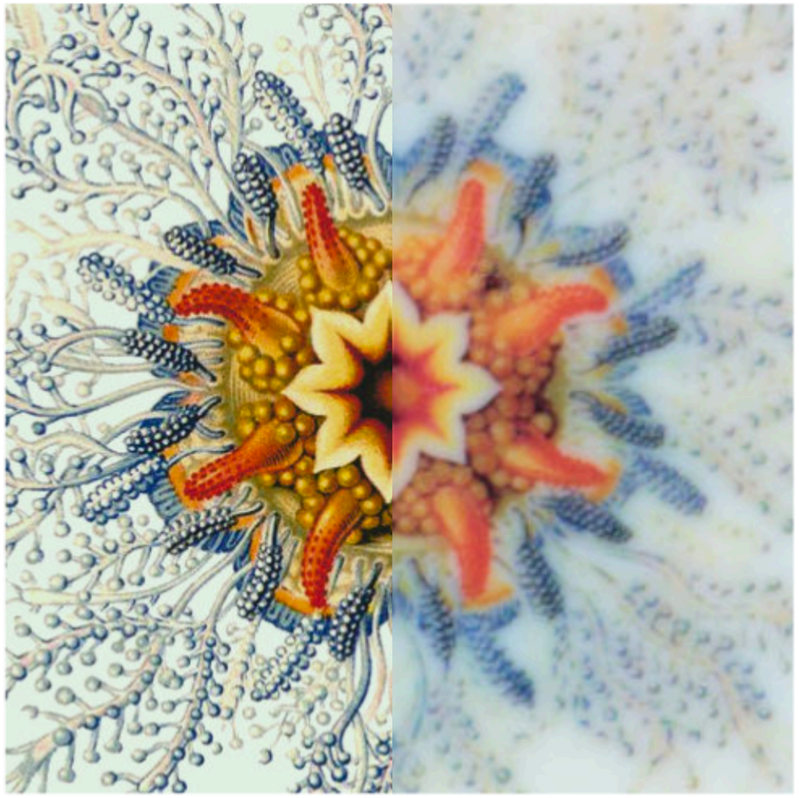
Mitsuba 2: A Retargetable Forward and Inverse Renderer [Nimier-David et al. 2019]



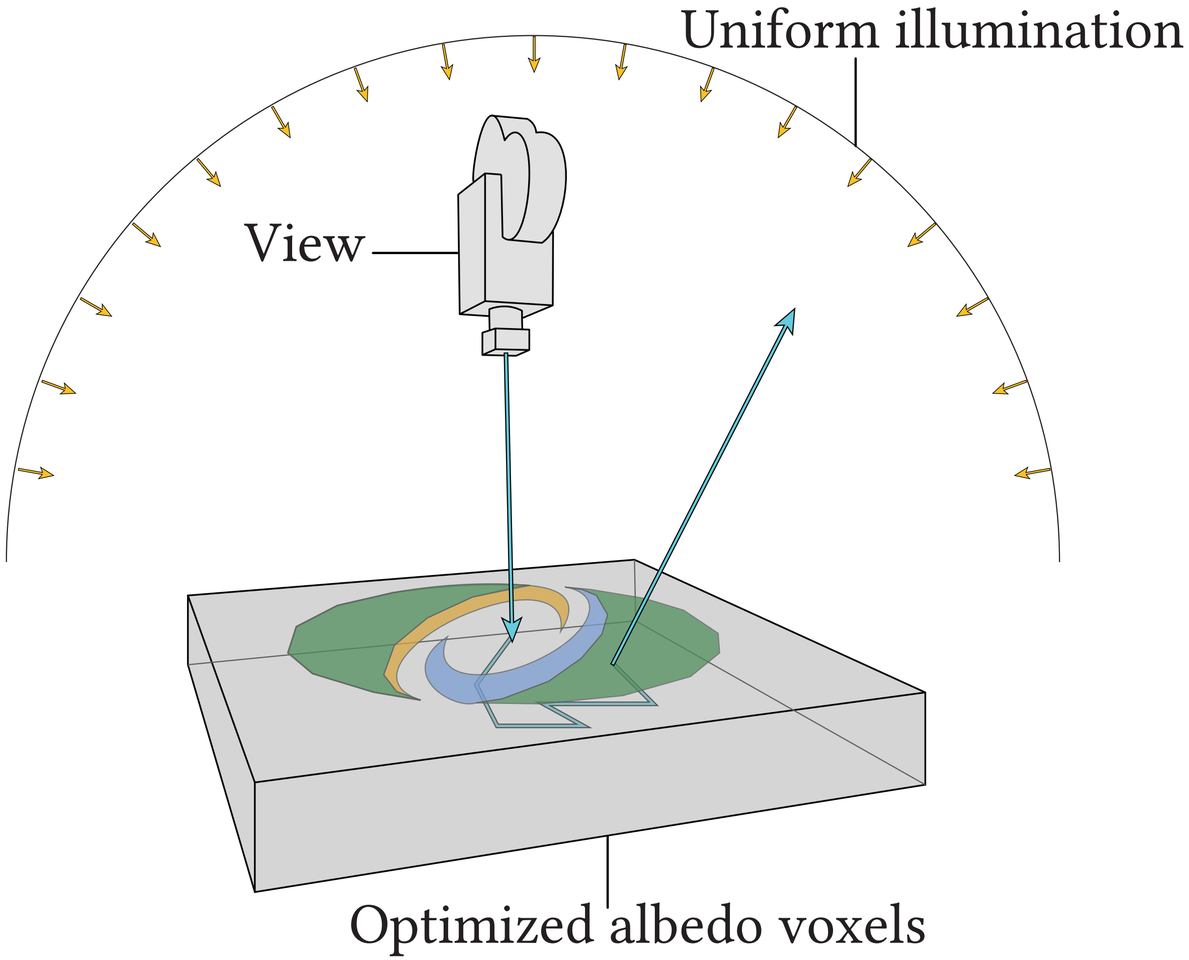
Mitsuba 2: A Retargetable Forward and Inverse Renderer [Nimier-David et al. 2019]

Fabrication: color optimization for 3D printing

Elek et al. 2017



Target Naive print



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

Reference: diffuse surface texture

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

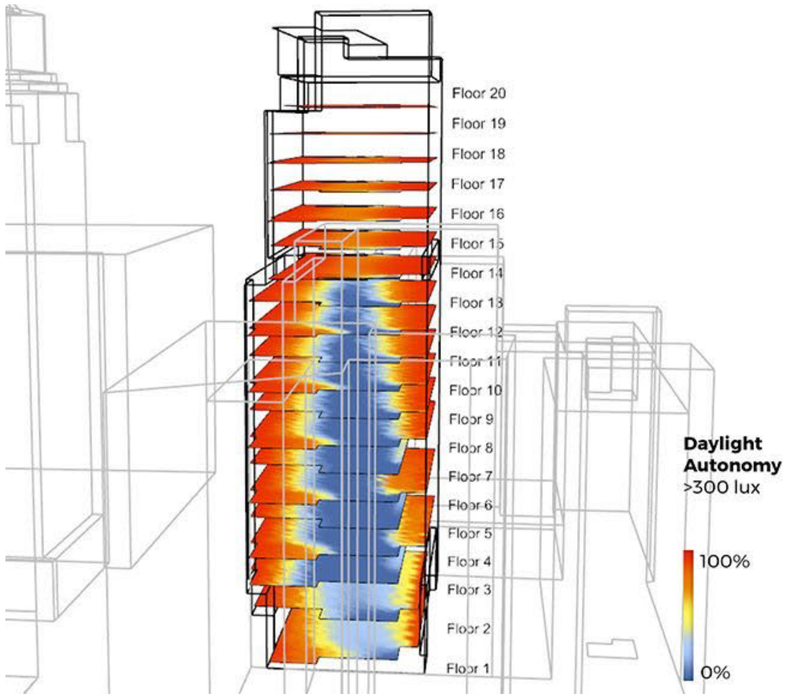
Reference: diffuse surface texture

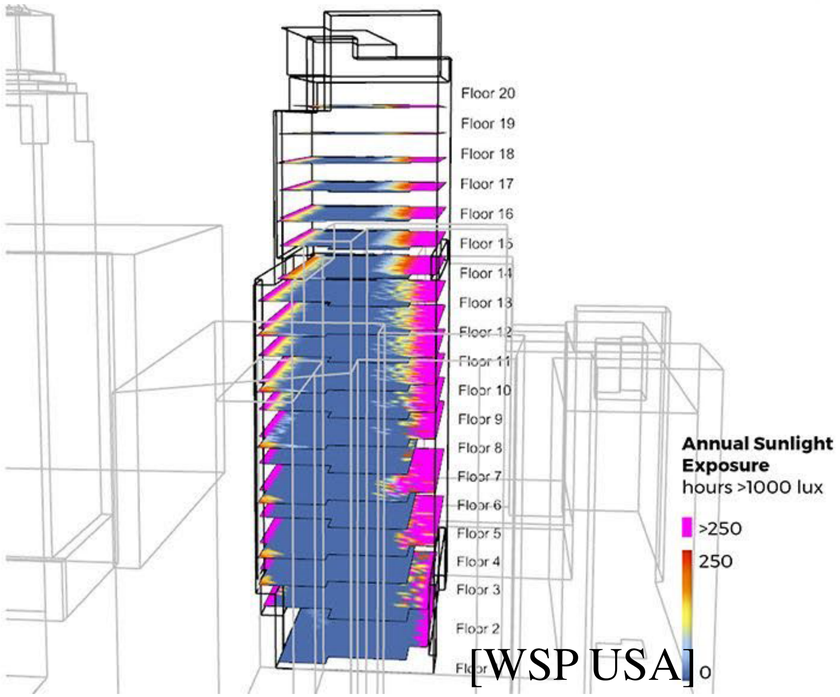
Beyond computer graphics: a world of applications

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

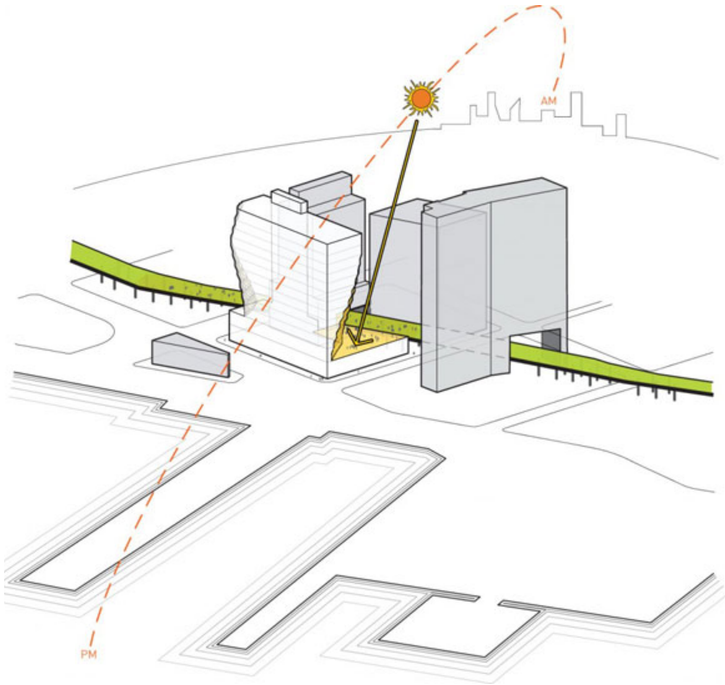
Beyond computer graphics: a world of applications

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]

Agenda for today

Inverse rendering

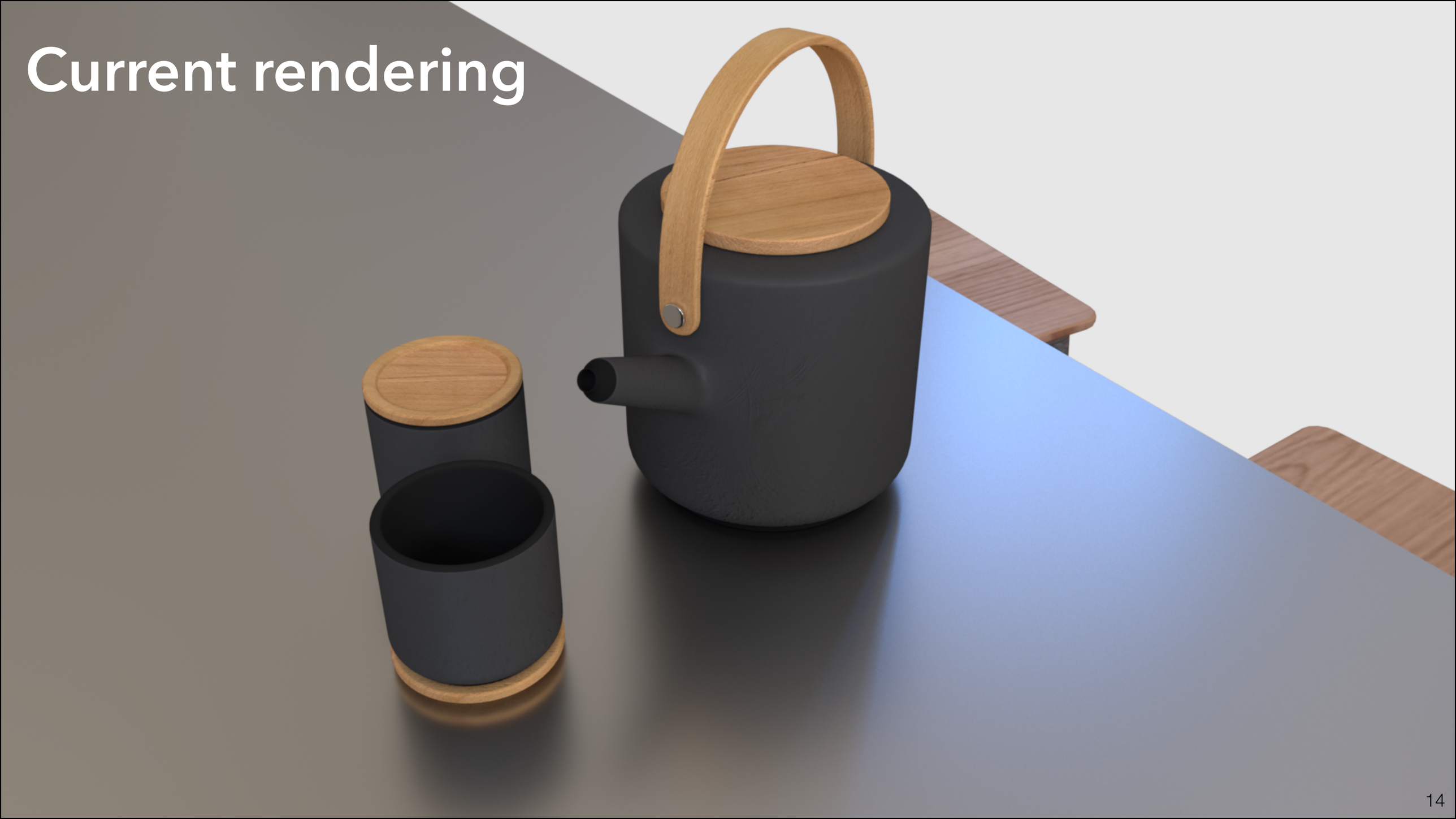
Example problem

Differentiable rendering

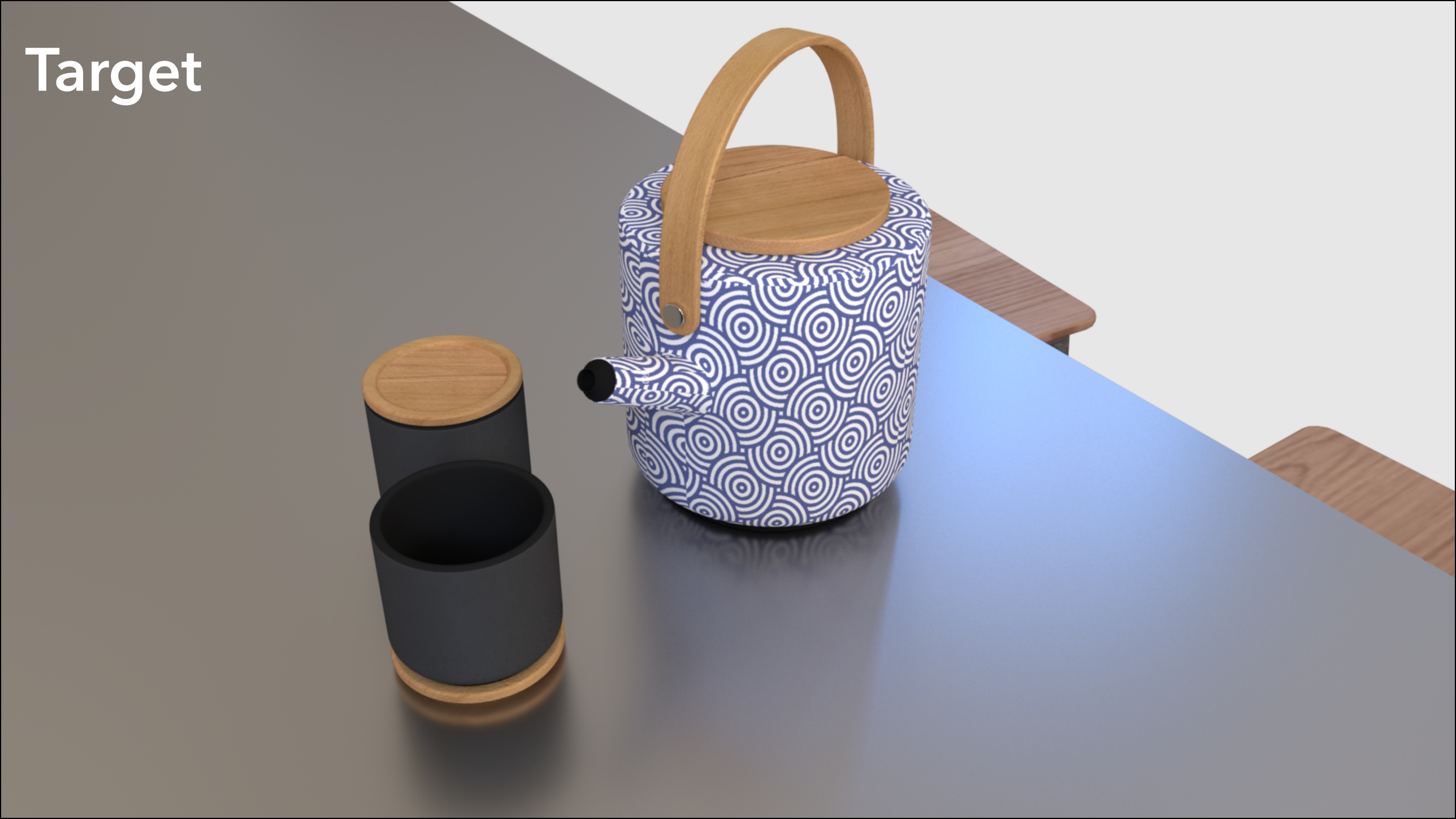
Challenges

1. How to do this at all?
2. Efficiency
3. Discontinuities
4. Robustness

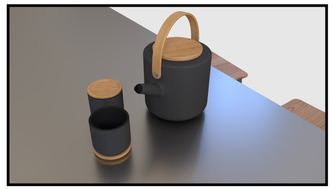
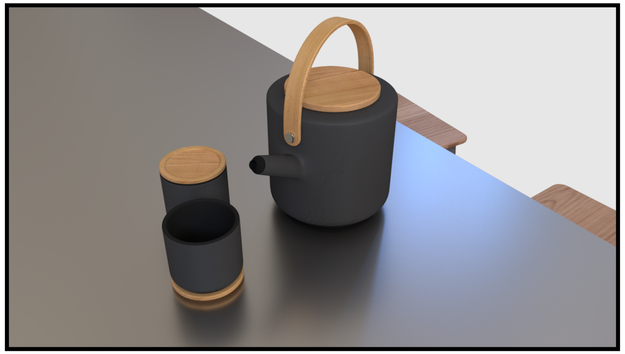
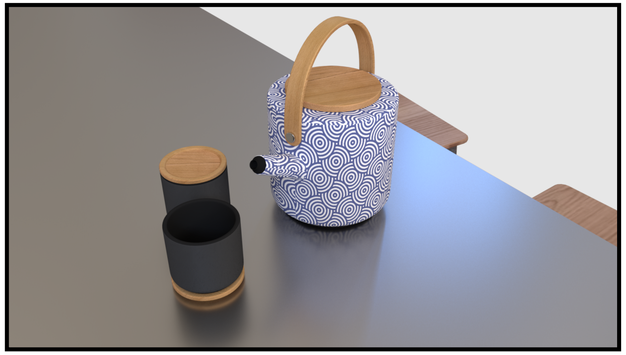
Current rendering



Target



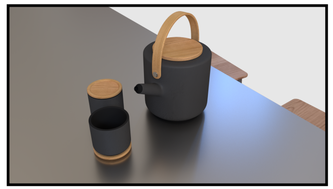
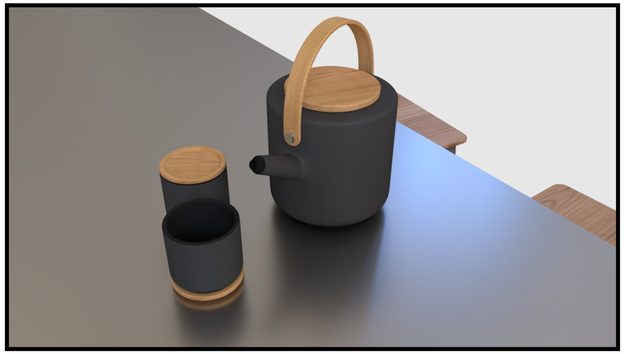
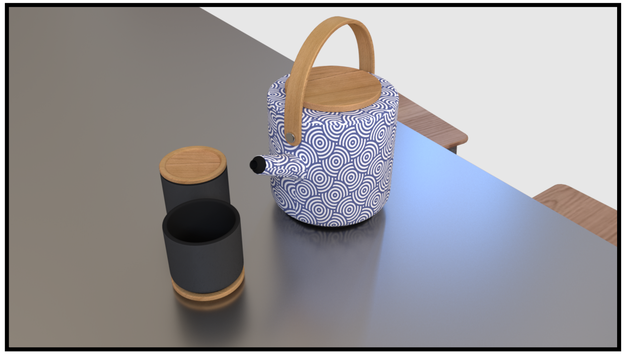
Objective function (a.k.a. “loss”)

$$g\left(\text{  }\right) = \left\| \text{  } - \text{  } \right\|^2$$

Rendering

Target

Objective function (a.k.a. "loss")

$$g\left(\text{  }\right) = \left\| \text{  } - \text{  } \right\|^2$$

Rendering

Target

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

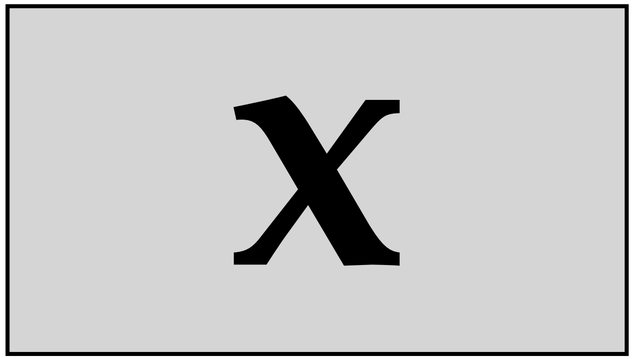
Scene parameters

Objective

Rendering algorithm

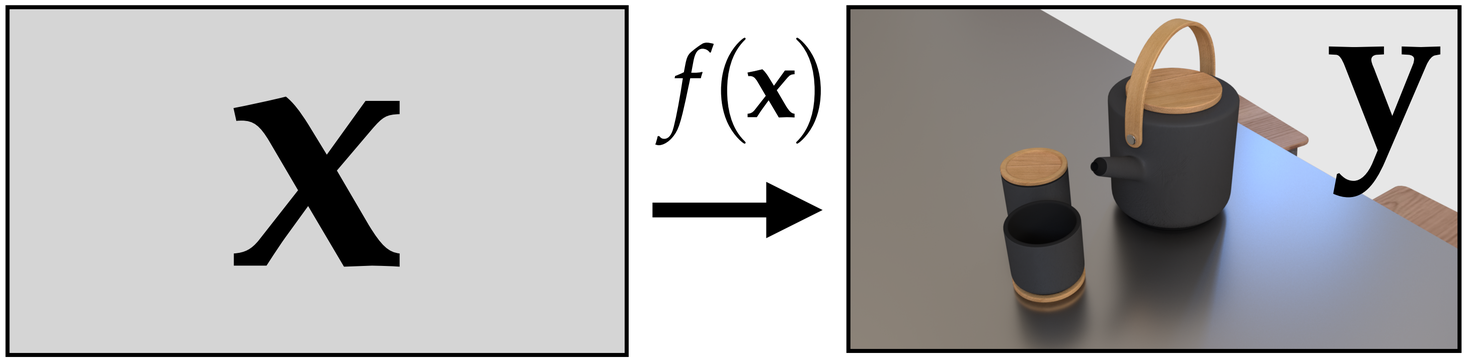
Gradient-based optimization

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



Gradient-based optimization

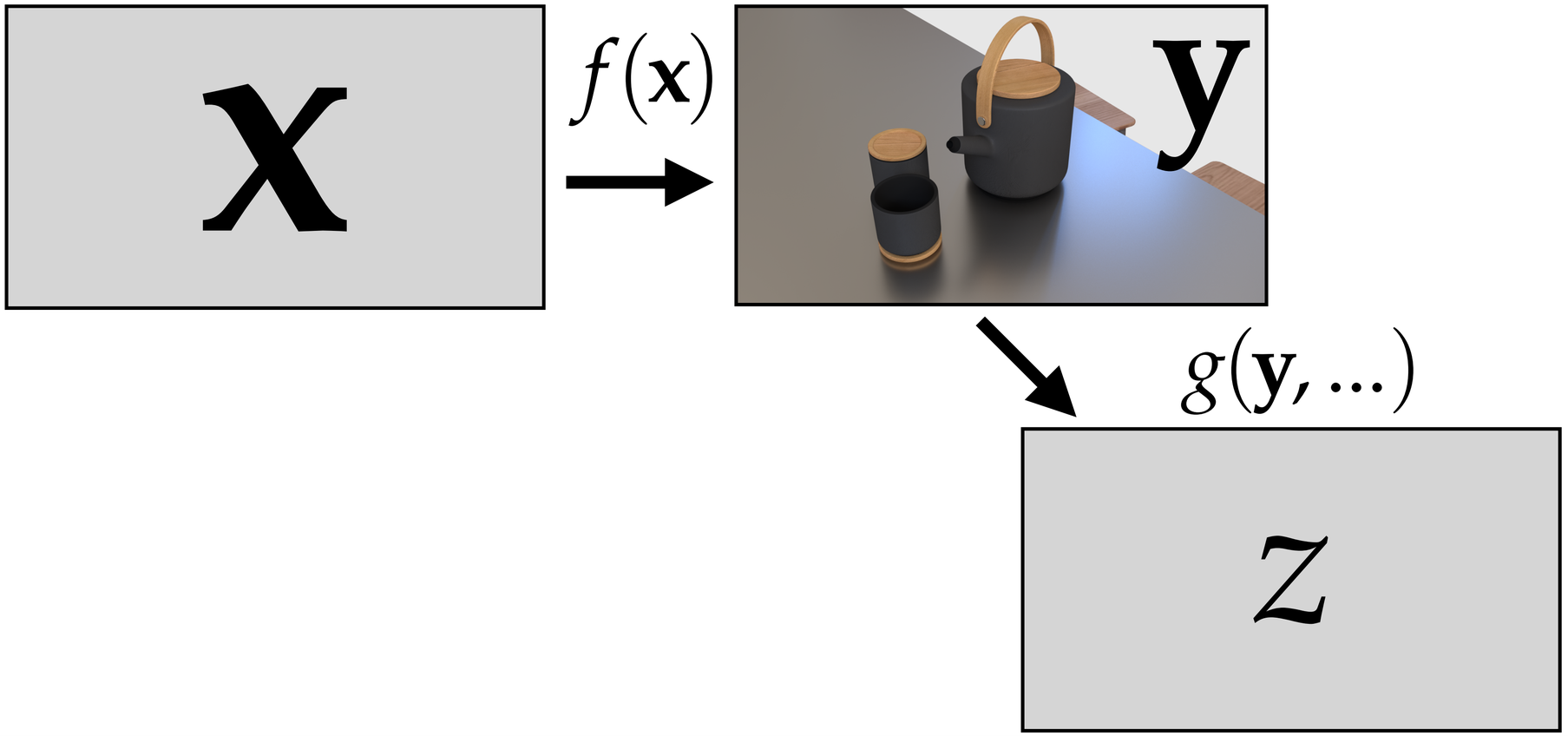
The problem: $\underset{\mathbf{x}\in\mathcal{X}}{\text{minimize}}\; g(f(\mathbf{x}))$



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

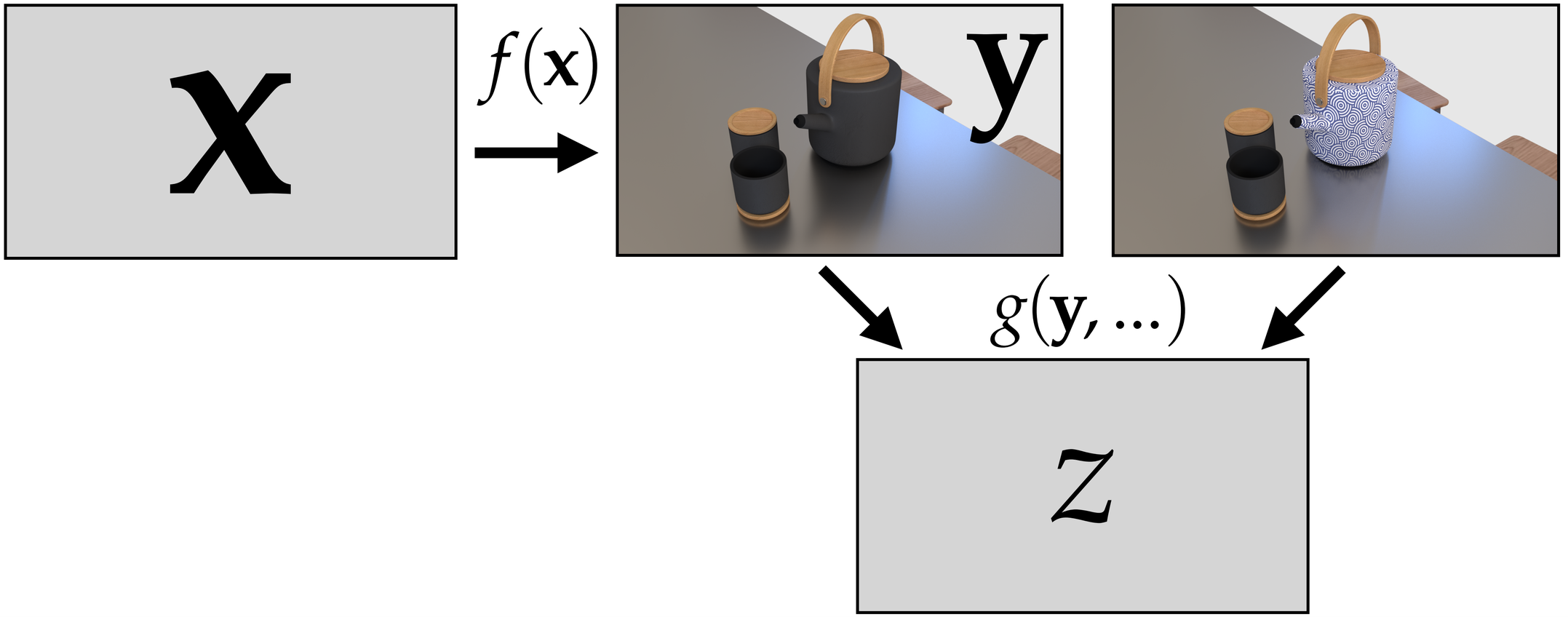
Gradient-based optimization

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



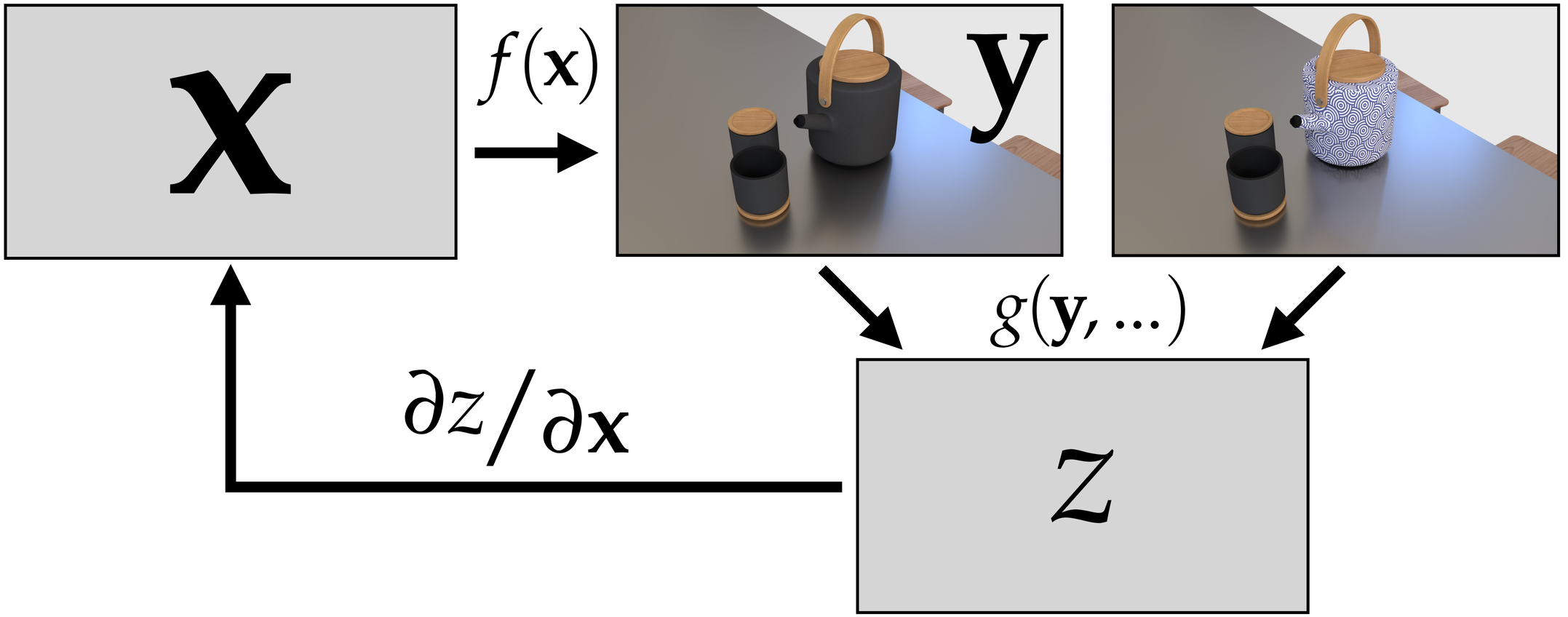
Gradient-based optimization

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

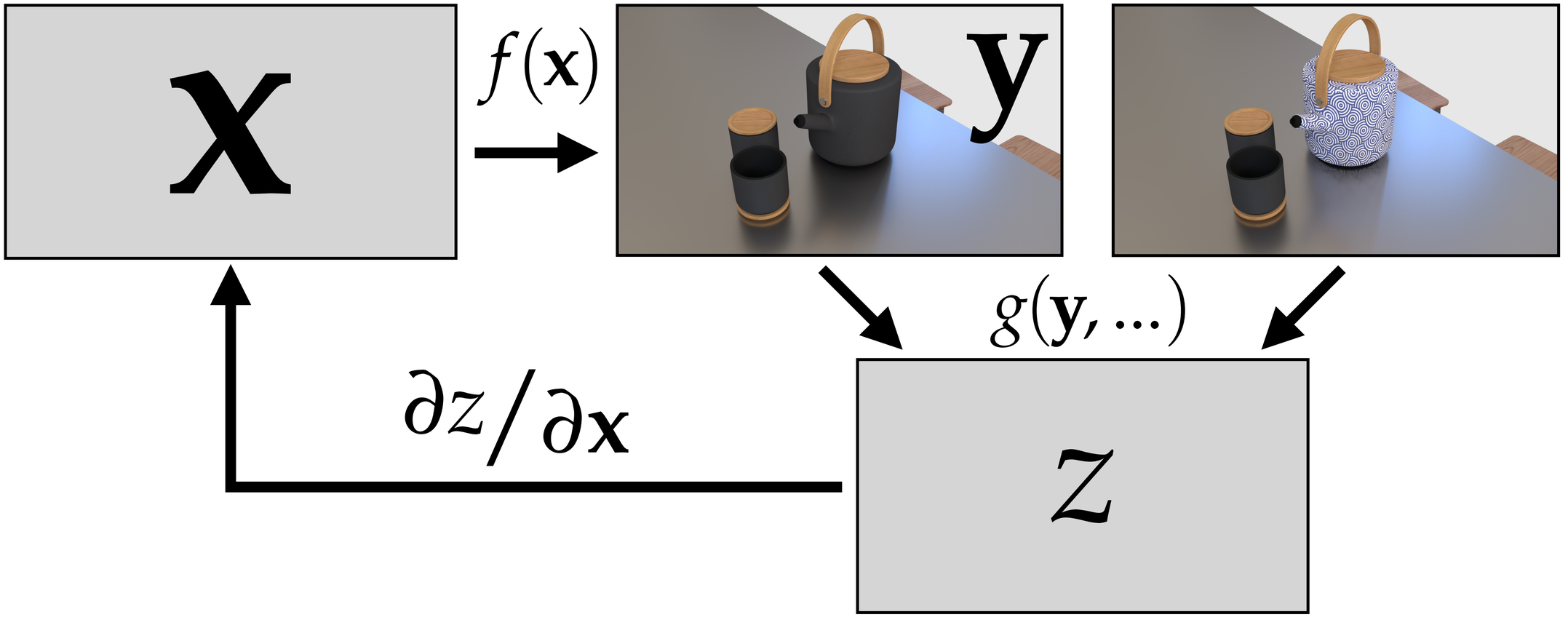


Gradient-based optimization

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



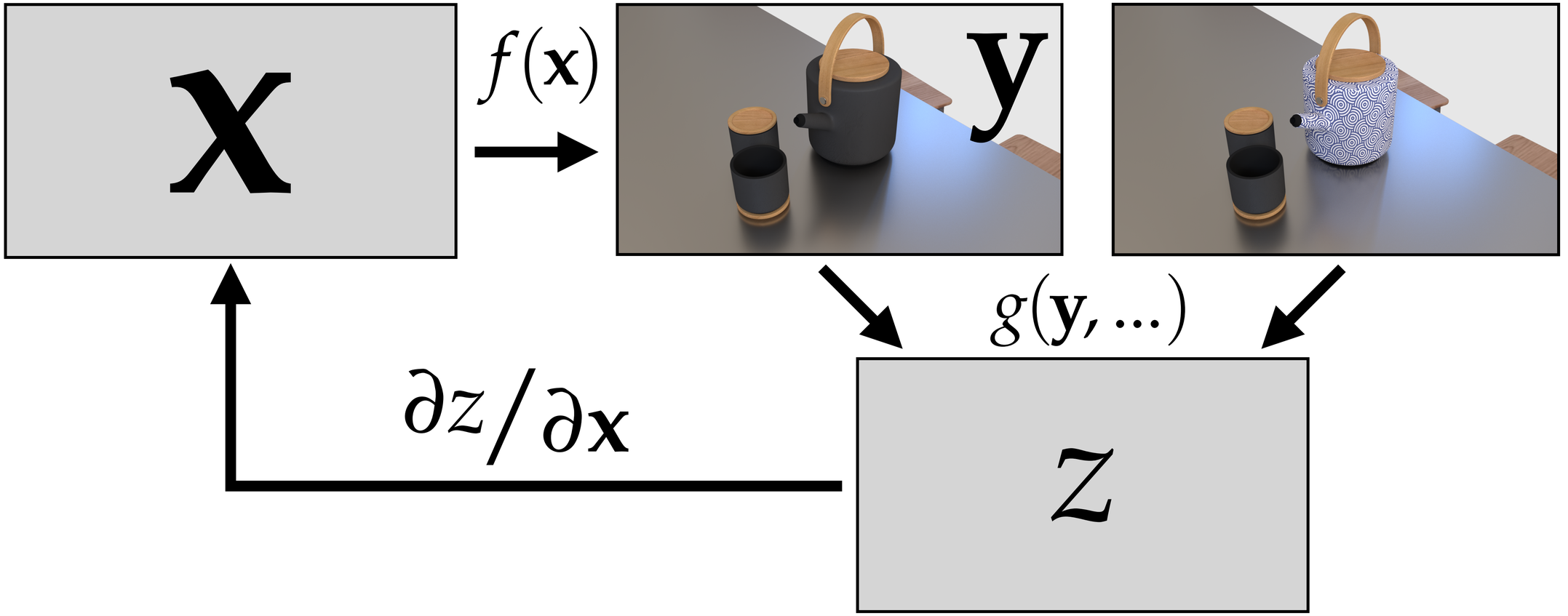
Gradient-based optimization



Gradient-based optimization

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

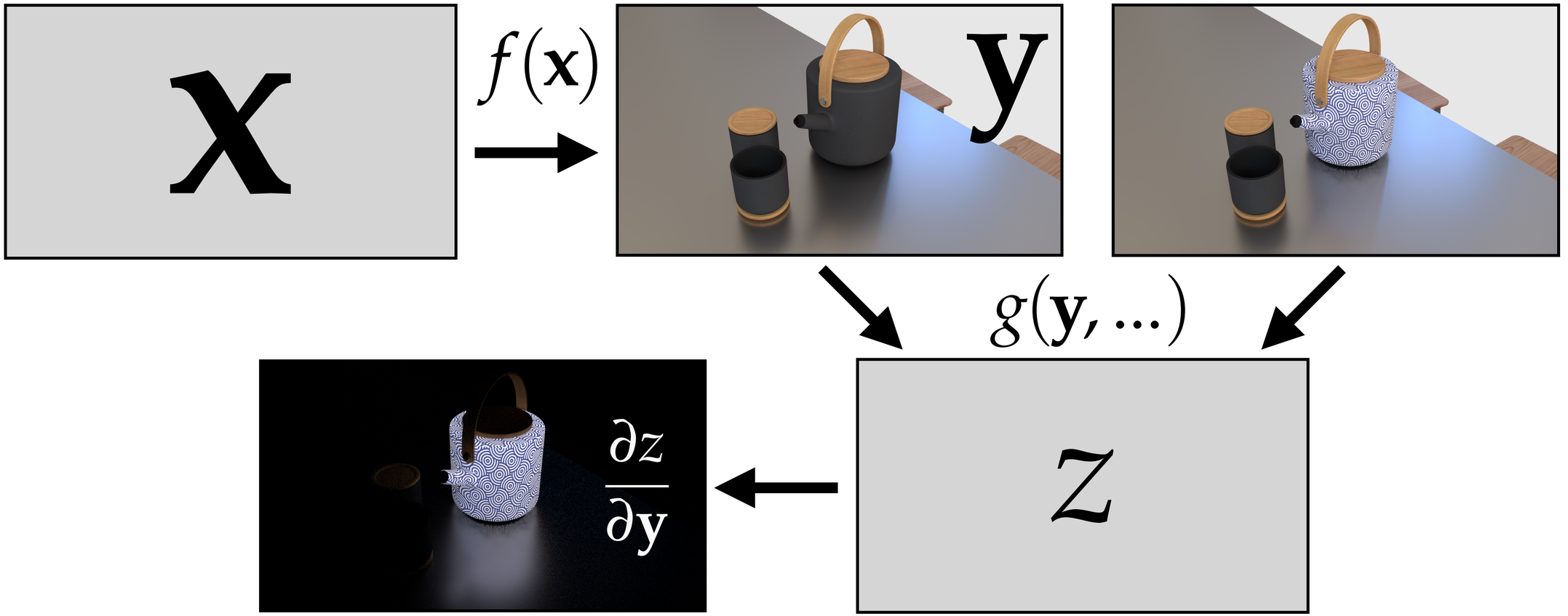
CHAIN RULE



Gradient-based optimization

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

CHAIN RULE

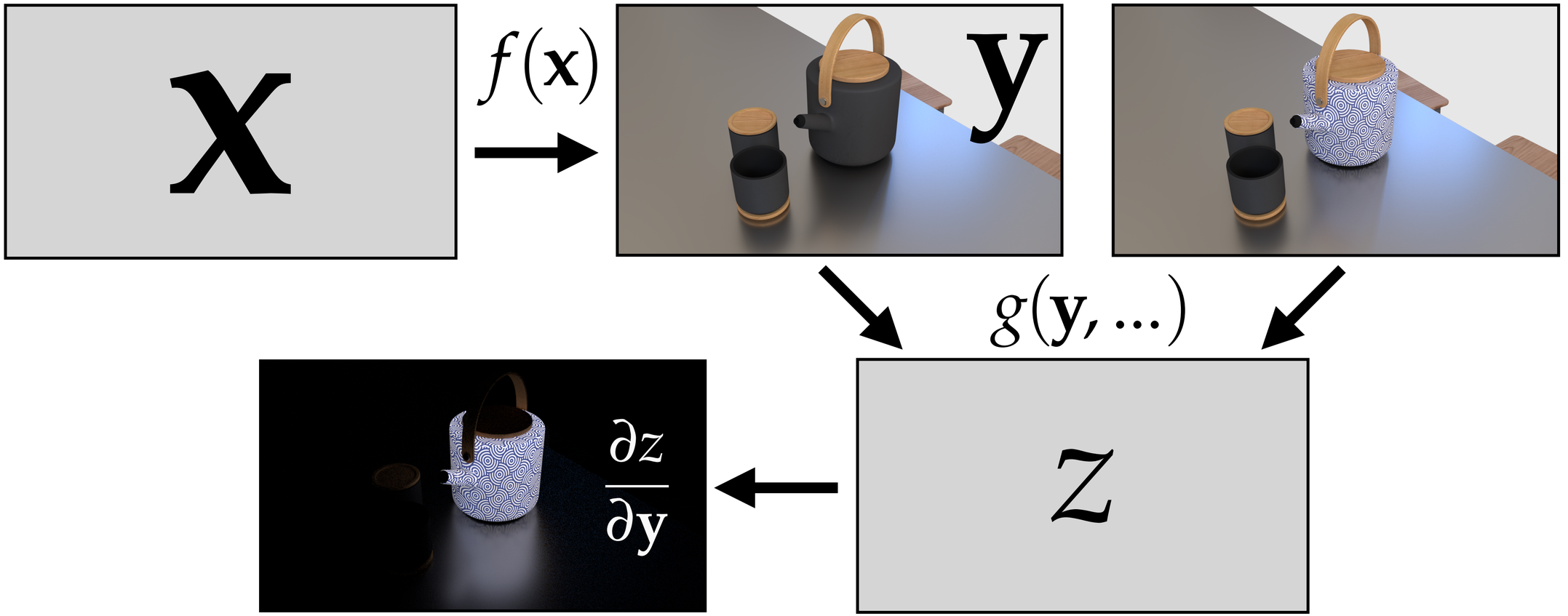


Gradient-based optimization

Vector

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

CHAIN RULE



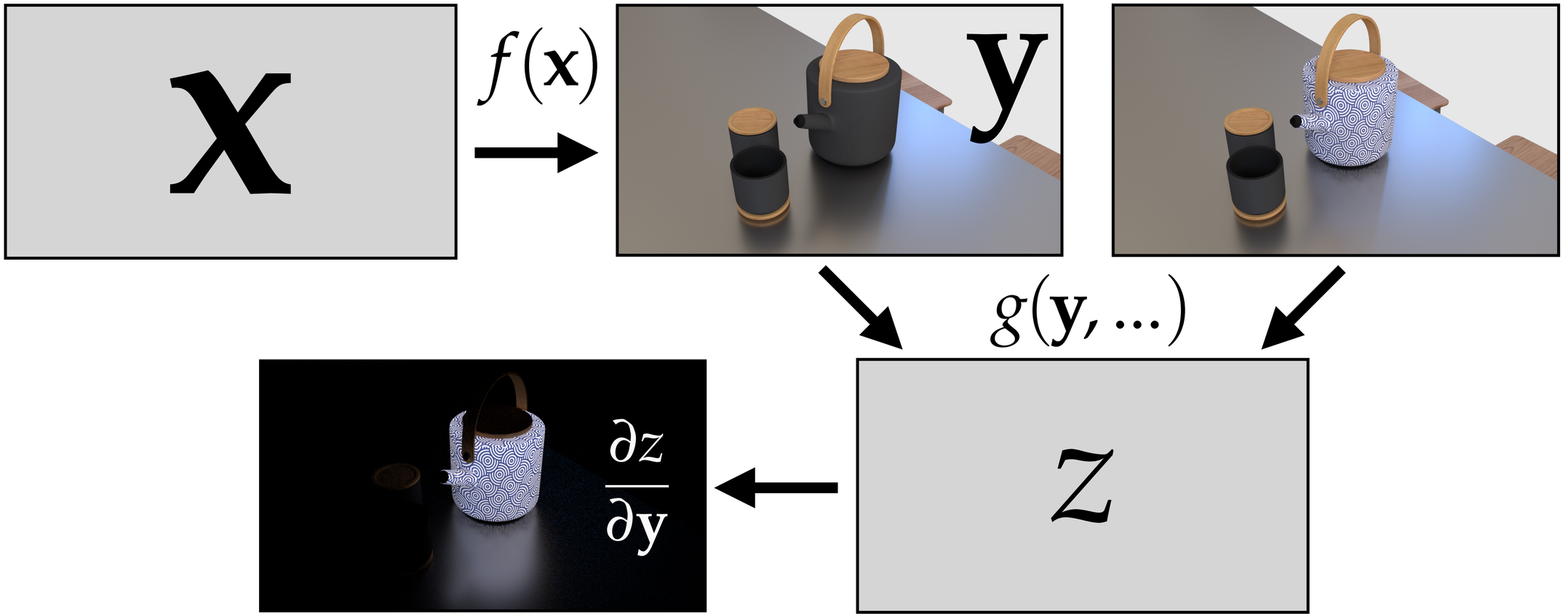
Gradient-based optimization

Vector

Matrix

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

CHAIN RULE



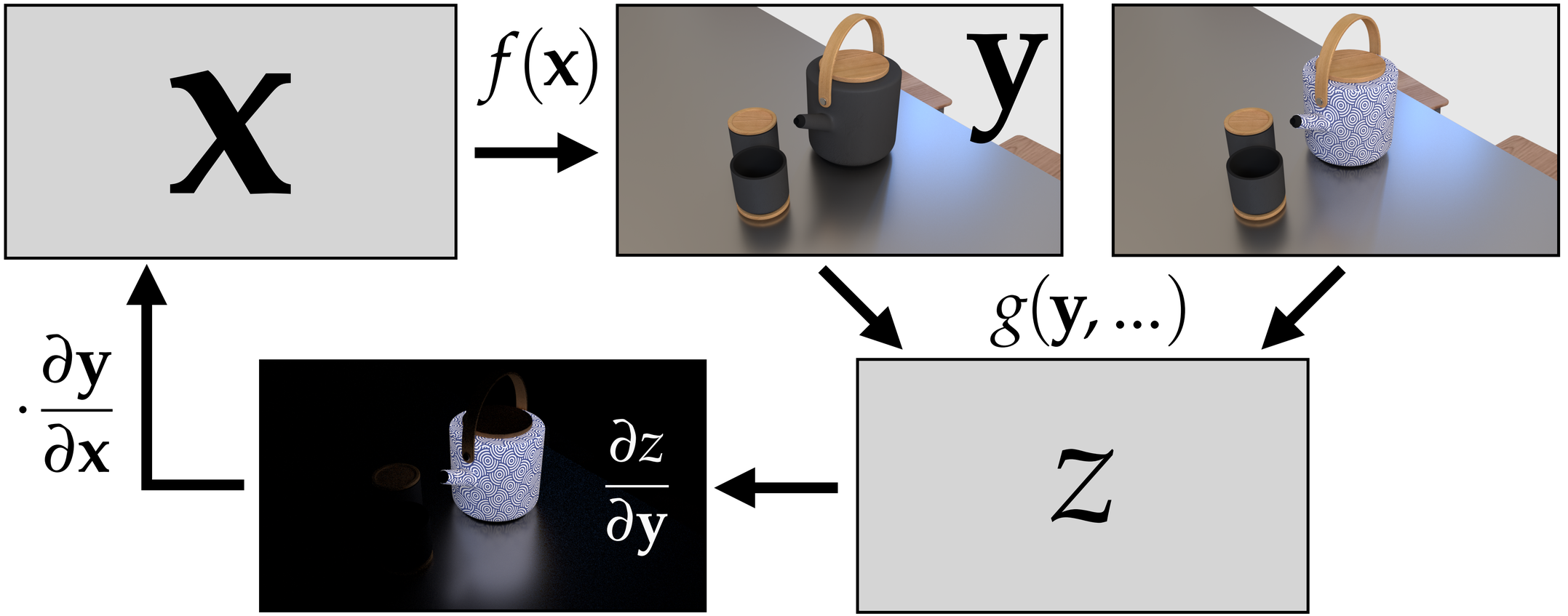
Gradient-based optimization

Vector

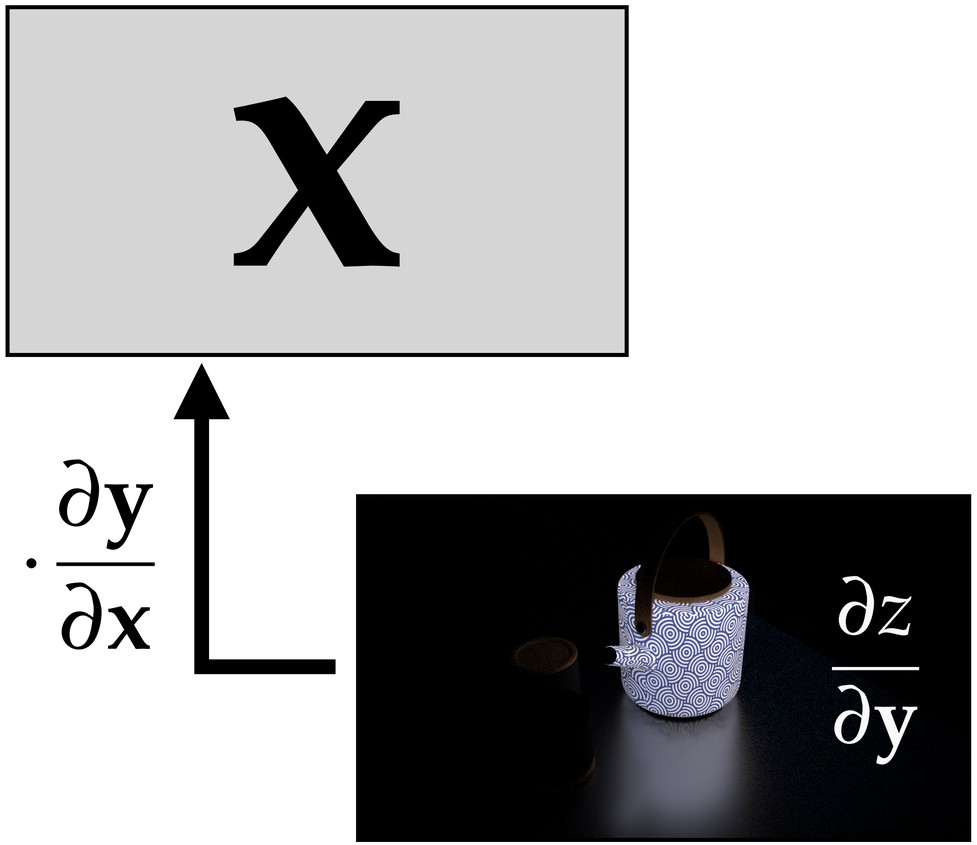
Matrix

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

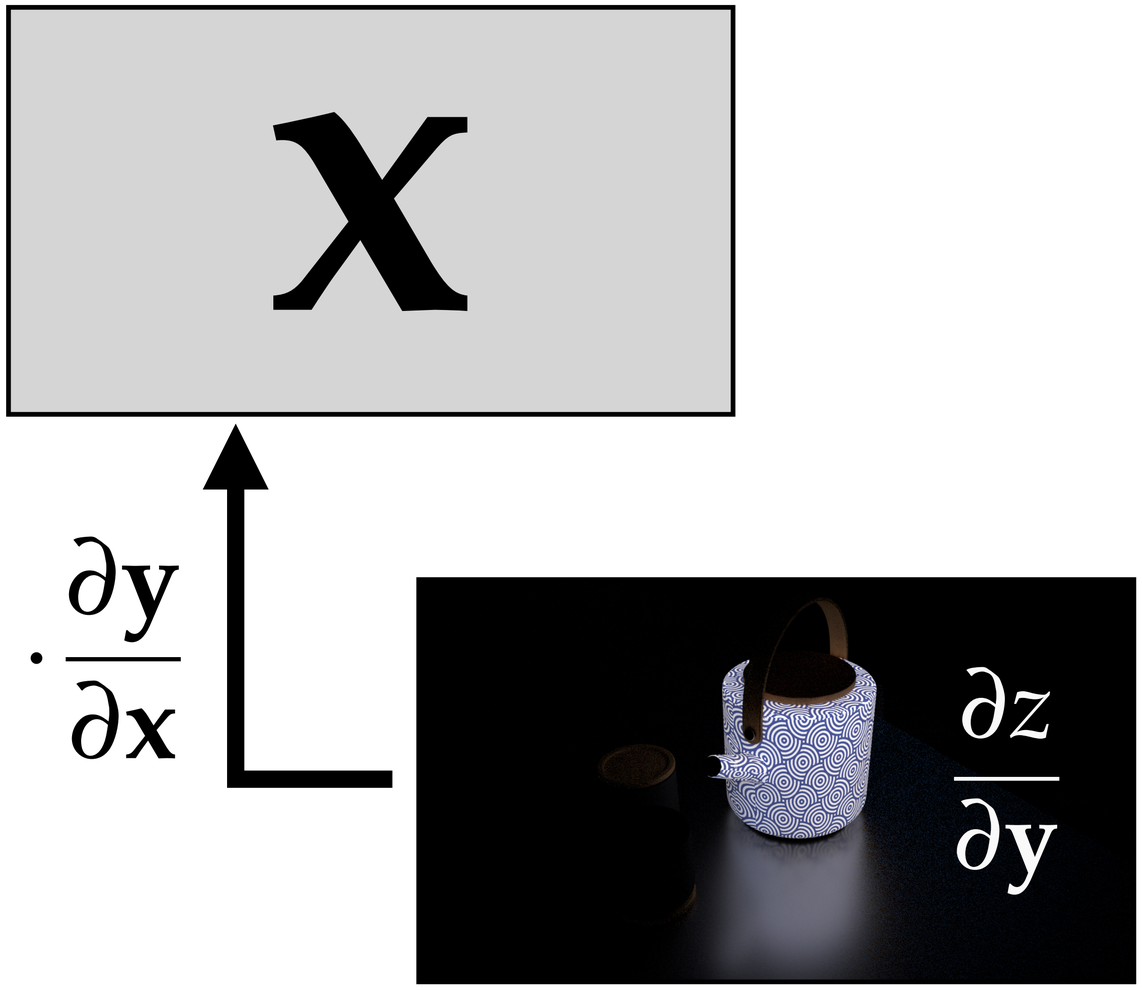
CHAIN RULE

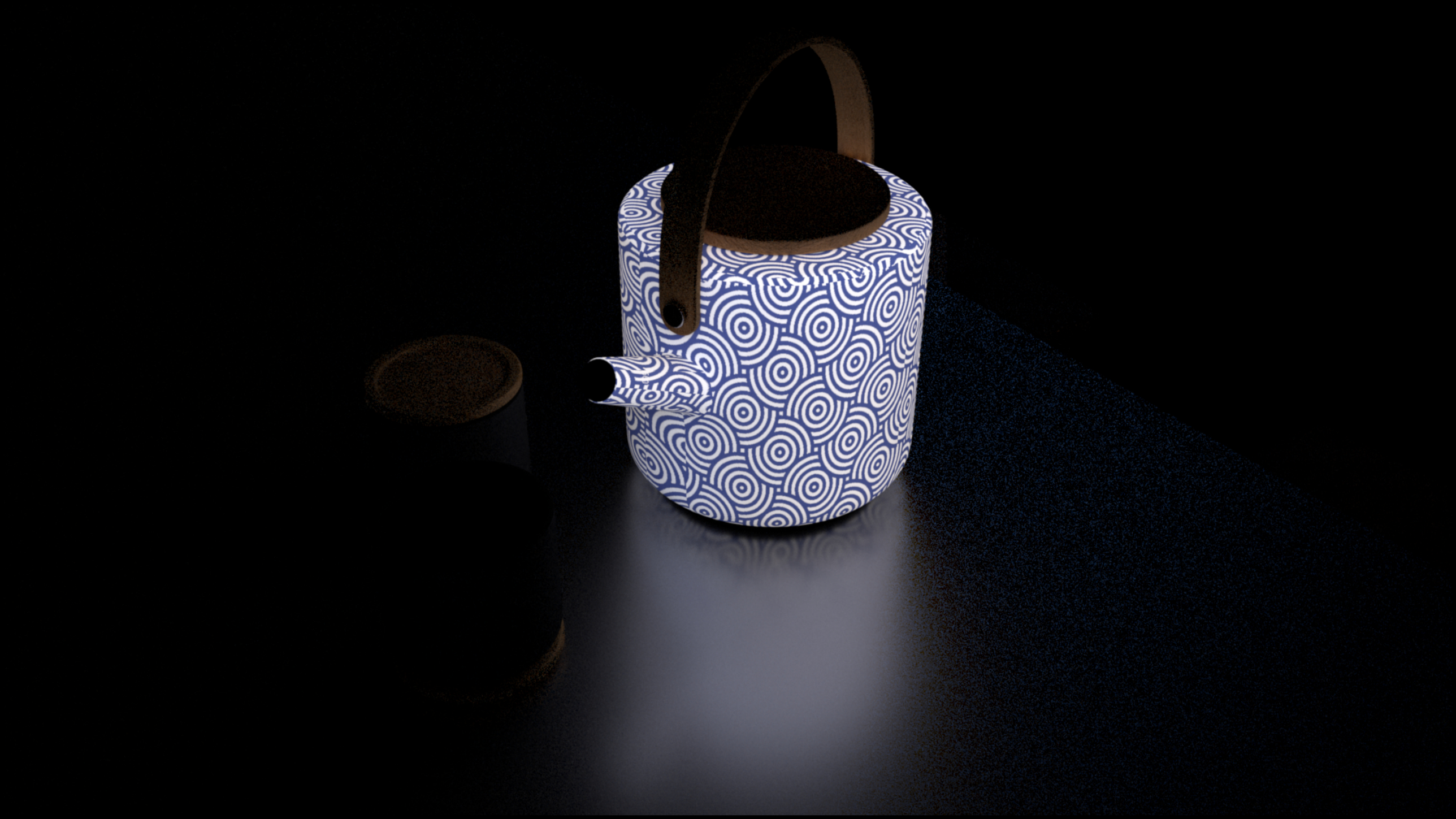


Gradient-based optimization

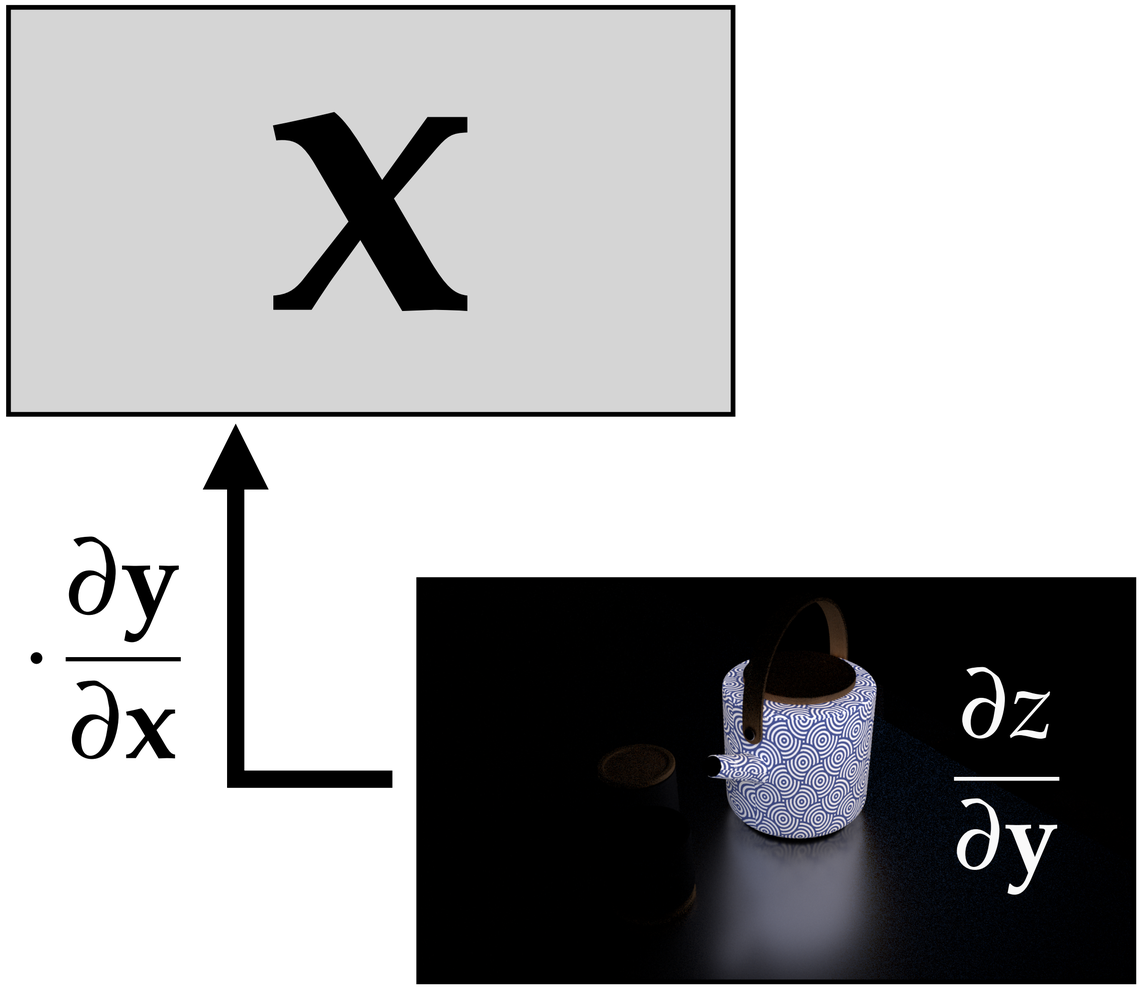


Gradient-based optimization



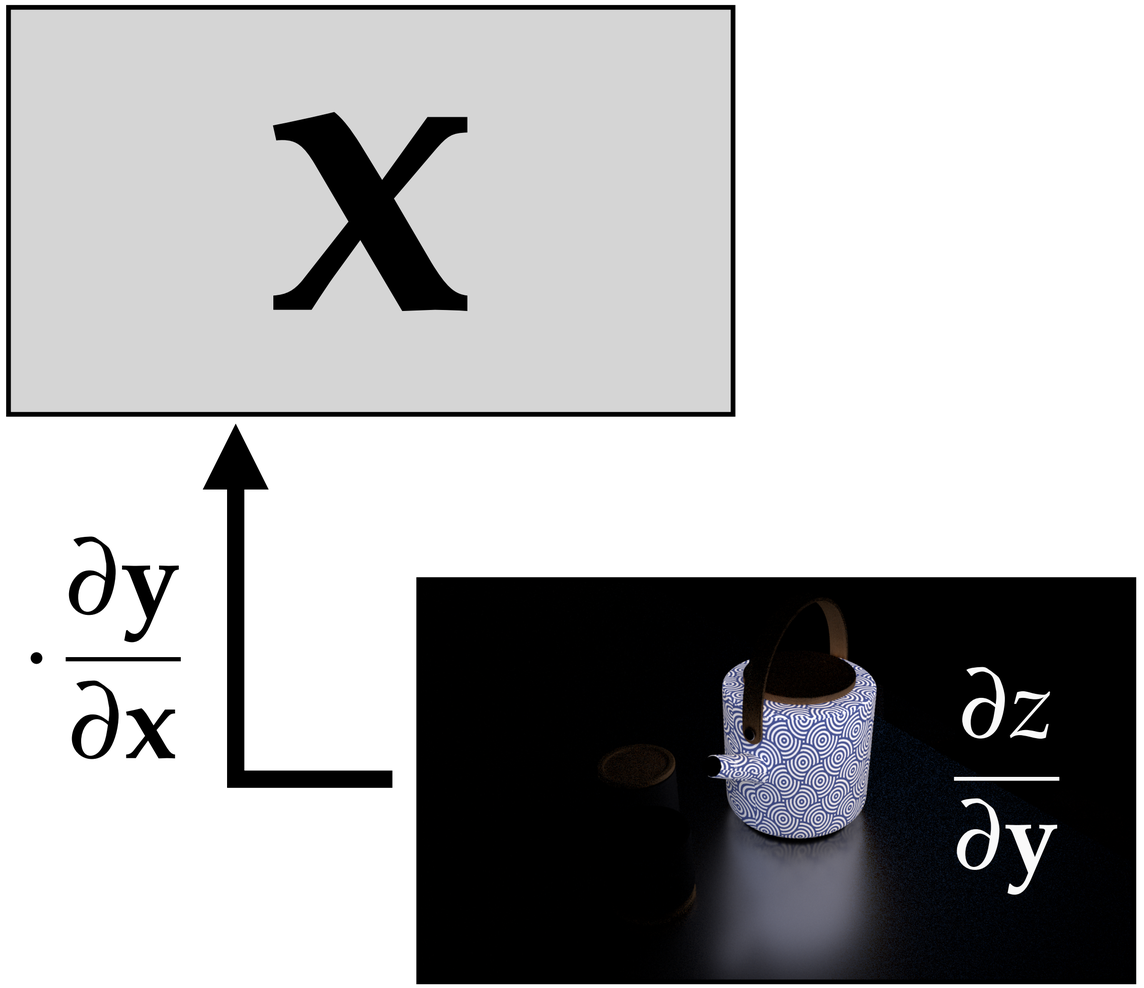


Gradient-based optimization

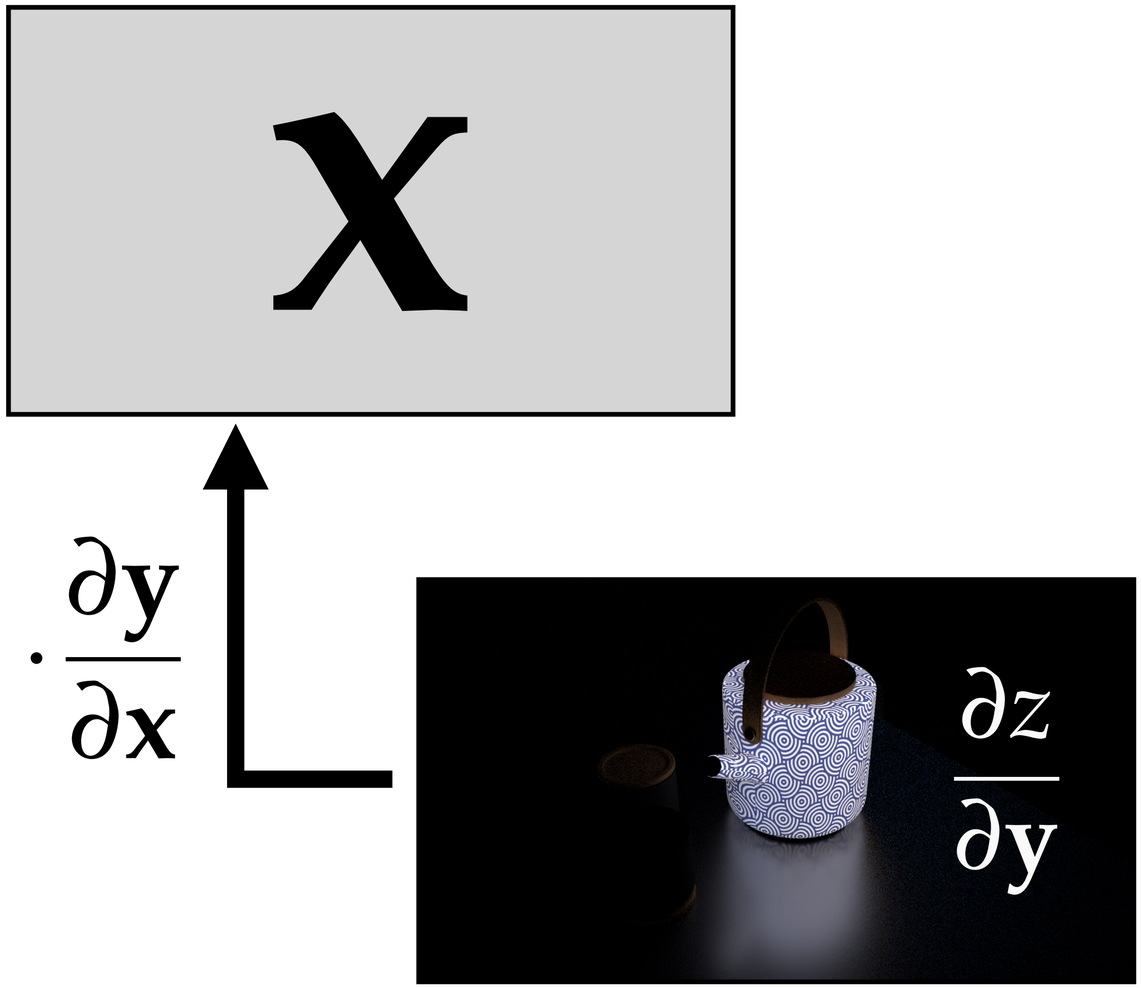


Gradient-based optimization

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



Gradient-based optimization



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

Challenges

1. How to do this (at all?)
2. How to do it efficiently?
3. How to deal with edges?
4. How to make it robust?

How to do this (at all?)

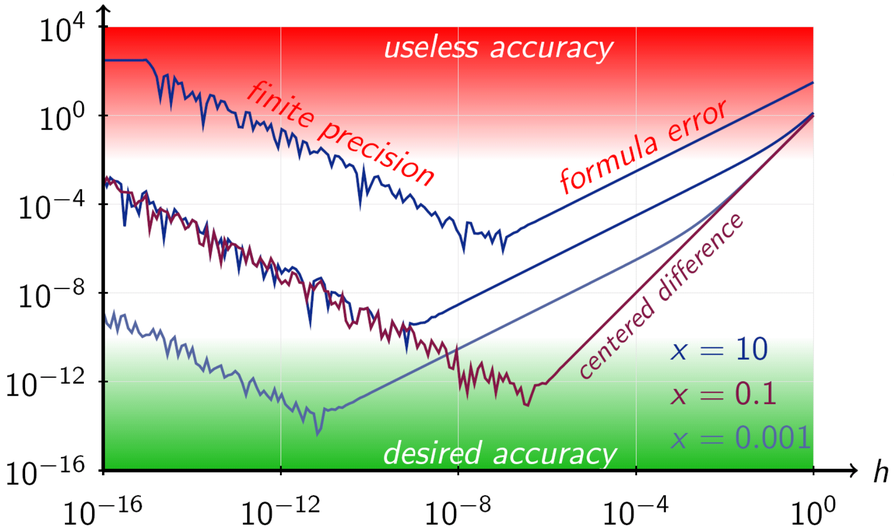
Use finite differences!

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

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]

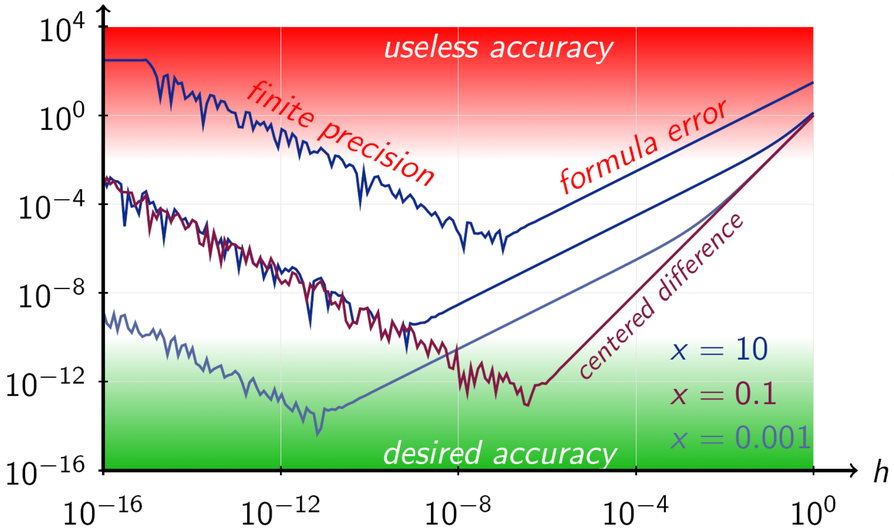
Potential problems:

- Bad approximation (big ε), rounding error (small ε)

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]

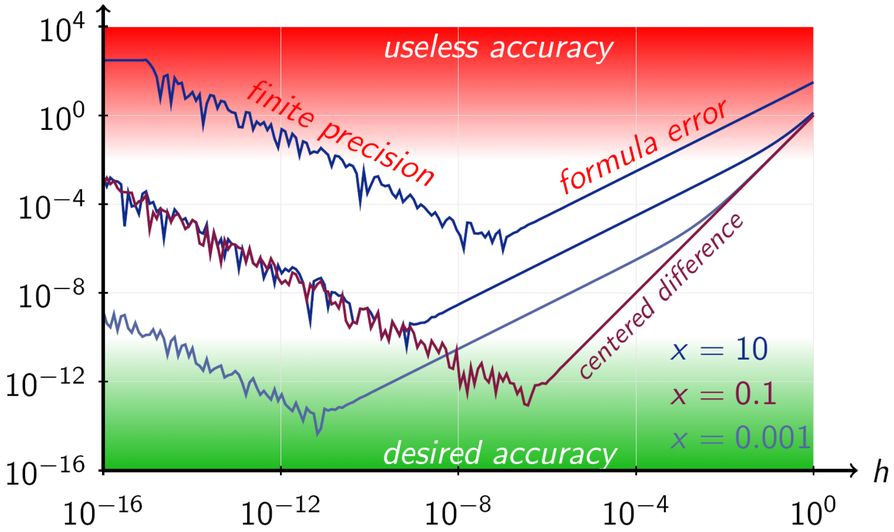
Potential problems:

- Bad approximation (big ε), rounding error (small ε)
- Need to correlate Monte Carlo samples

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.

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**

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

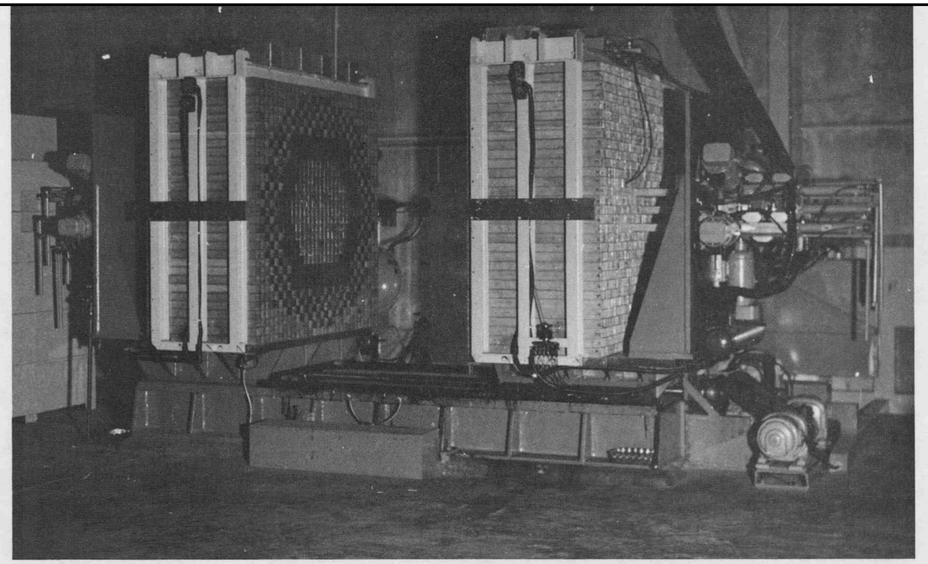
$$F = KF + f,$$

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

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

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

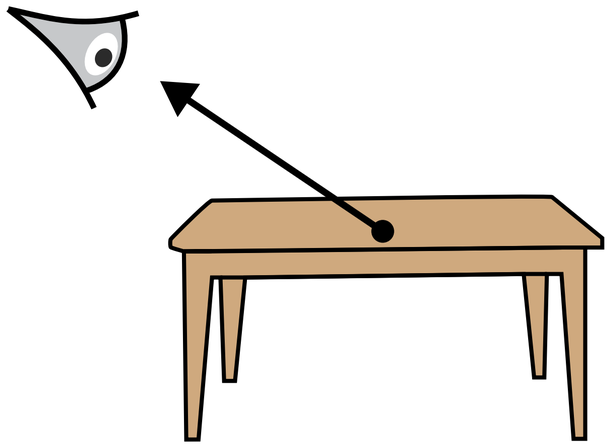


103-298

Fig. 2. ZPR-3 Critical Facility

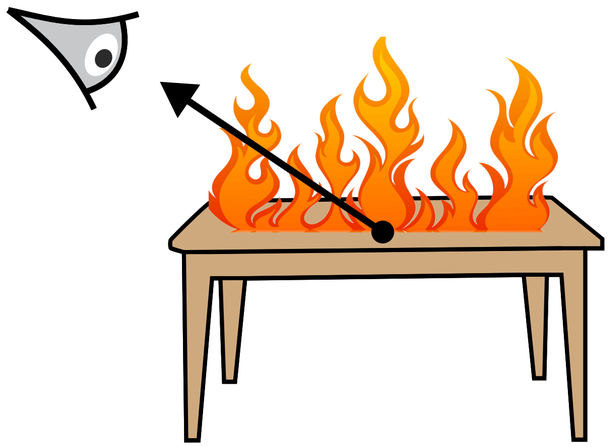
Let's differentiate the rendering equation

$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_{S^2} L_i(\mathbf{x}, \omega') \, f_s(\mathbf{x}, \omega, \omega') \, \cos\theta \, \mathrm{d}\omega'$$



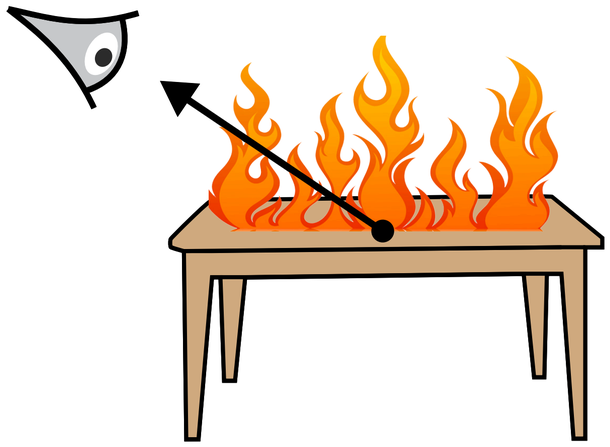
Let’s differentiate the rendering equation

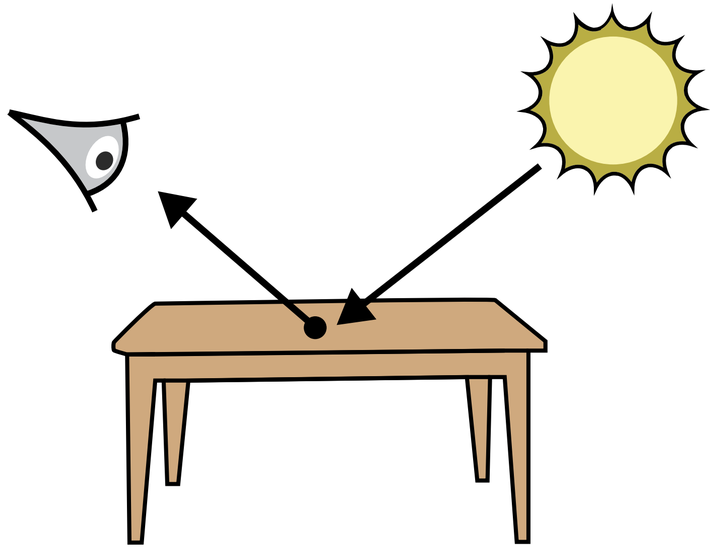
$$L_o(\mathbf{x},\omega) = L_e(\mathbf{x},\omega) + \int_{S^2} L_i(\mathbf{x},\omega')\, f_s(\mathbf{x},\omega,\omega')\, \cos\theta \, \mathrm{d}\omega'$$



Let's differentiate the rendering equation

$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_{S^2} L_i(\mathbf{x}, \omega') \, f_s(\mathbf{x}, \omega, \omega') \, \cos\theta \, \mathrm{d}\omega'$$





Let's differentiate the rendering equation

$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_{S^2} L_i(\mathbf{x}, \omega') \, f_s(\mathbf{x}, \omega, \omega') \, \cos\theta \, \mathrm{d}\omega'$$

$\partial_{\mathbf{x}}$ derivative wrt. scene parameters

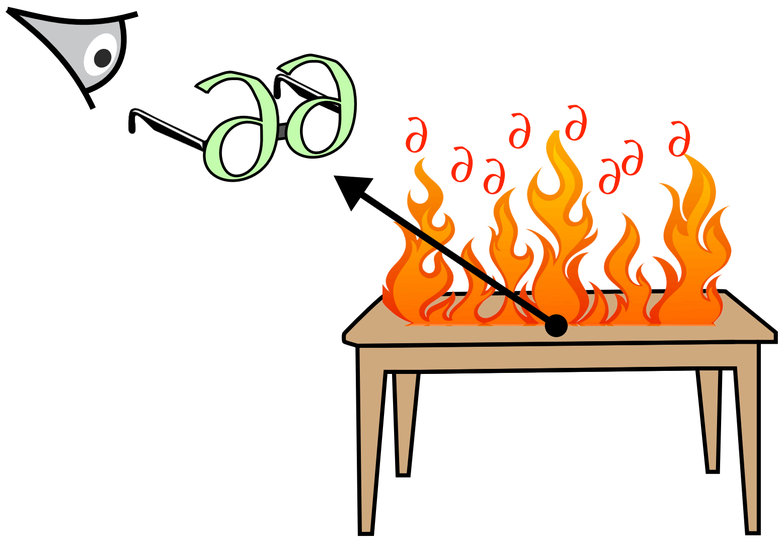
Let's differentiate the rendering equation

$$\partial_{\mathbf{x}} L_o(\mathbf{x}, \omega) = \partial_{\mathbf{x}} L_e(\mathbf{x}, \omega) + \int_{S^2} L_i(\mathbf{x}, \omega') \, f_s(\mathbf{x}, \omega, \omega') \, \cos\theta \, \mathrm{d}\omega'$$

$\partial_{\mathbf{x}}$ derivative wrt. scene parameters

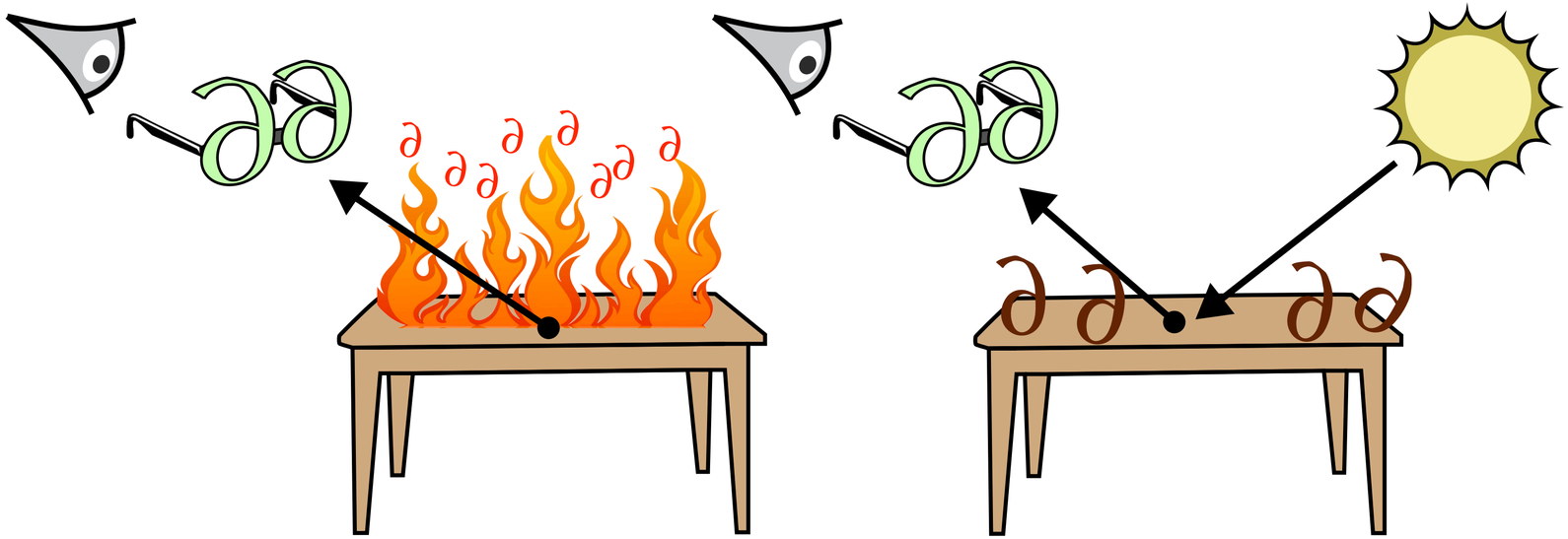
Let’s differentiate the rendering equation

$$\partial_{\mathbf{x}} L_o(\mathbf{x},\omega) = \partial_{\mathbf{x}} L_e(\mathbf{x},\omega) + \int_{S^2} L_i(\mathbf{x},\omega')\, f_s(\mathbf{x},\omega,\omega')\, \cos\theta \, \mathrm{d}\omega'$$



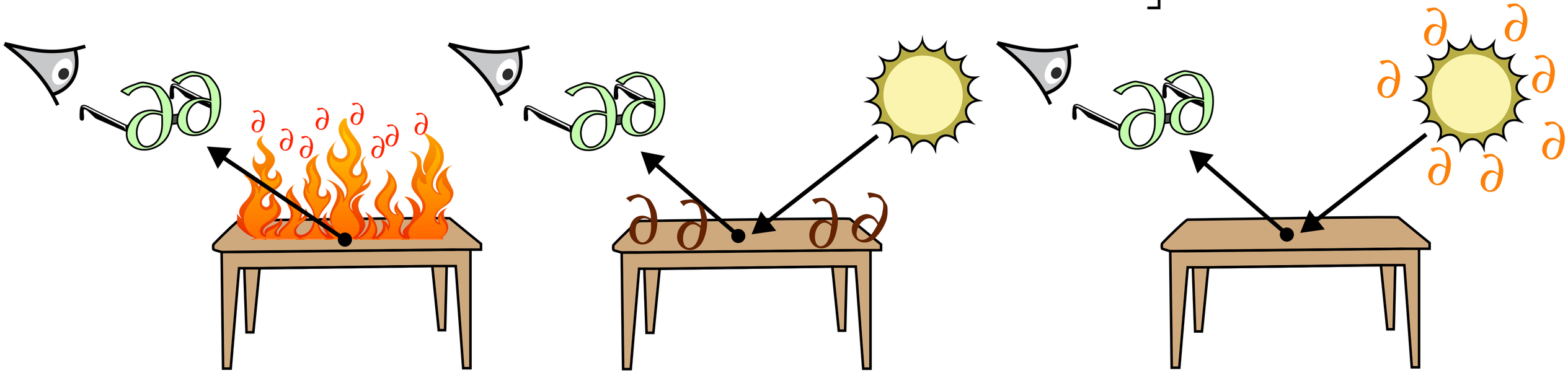
Let’s differentiate the rendering equation

$$\partial_{\mathbf{x}} L_o(\mathbf{x},\omega) = \partial_{\mathbf{x}} L_e(\mathbf{x},\omega) + \int_{S^2} \Big[L_i(\mathbf{x},\omega') \, \partial_{\mathbf{x}} f_s(\mathbf{x},\omega,\omega')$$



Let's differentiate the rendering equation

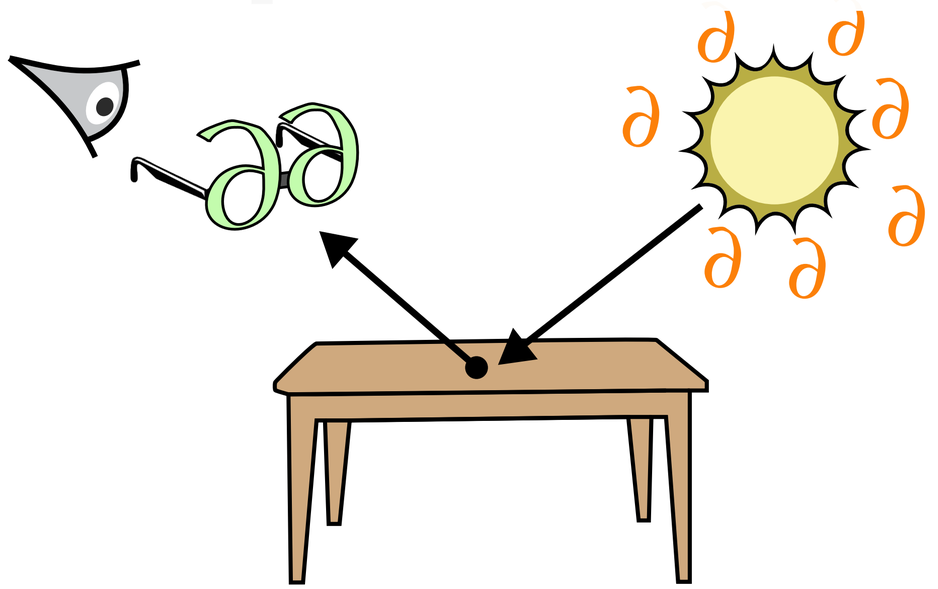
$$\partial_{\mathbf{x}} L_o(\mathbf{x},\omega) = \partial_{\mathbf{x}} L_e(\mathbf{x},\omega) + \int_{S^2} \Big[L_i(\mathbf{x},\omega')\, \partial_{\mathbf{x}} f_s(\mathbf{x},\omega,\omega') + \partial_{\mathbf{x}} L_i(\mathbf{x},\omega')\, f_s(\mathbf{x},\omega,\omega') \Big] \cos\theta \, \mathrm{d}\omega'$$



Let’s differentiate the rendering equation

TL;DR

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



Let’s differentiate the rendering equation

$$\partial_x L_o(\mathbf{x},\omega) = \partial_x L_e(\mathbf{x},\omega) + \partial_x \int_{\Omega} L_i(\mathbf{x},\omega')\, f_s(\mathbf{x},\omega,\omega') \cos\theta \, d\omega'$$

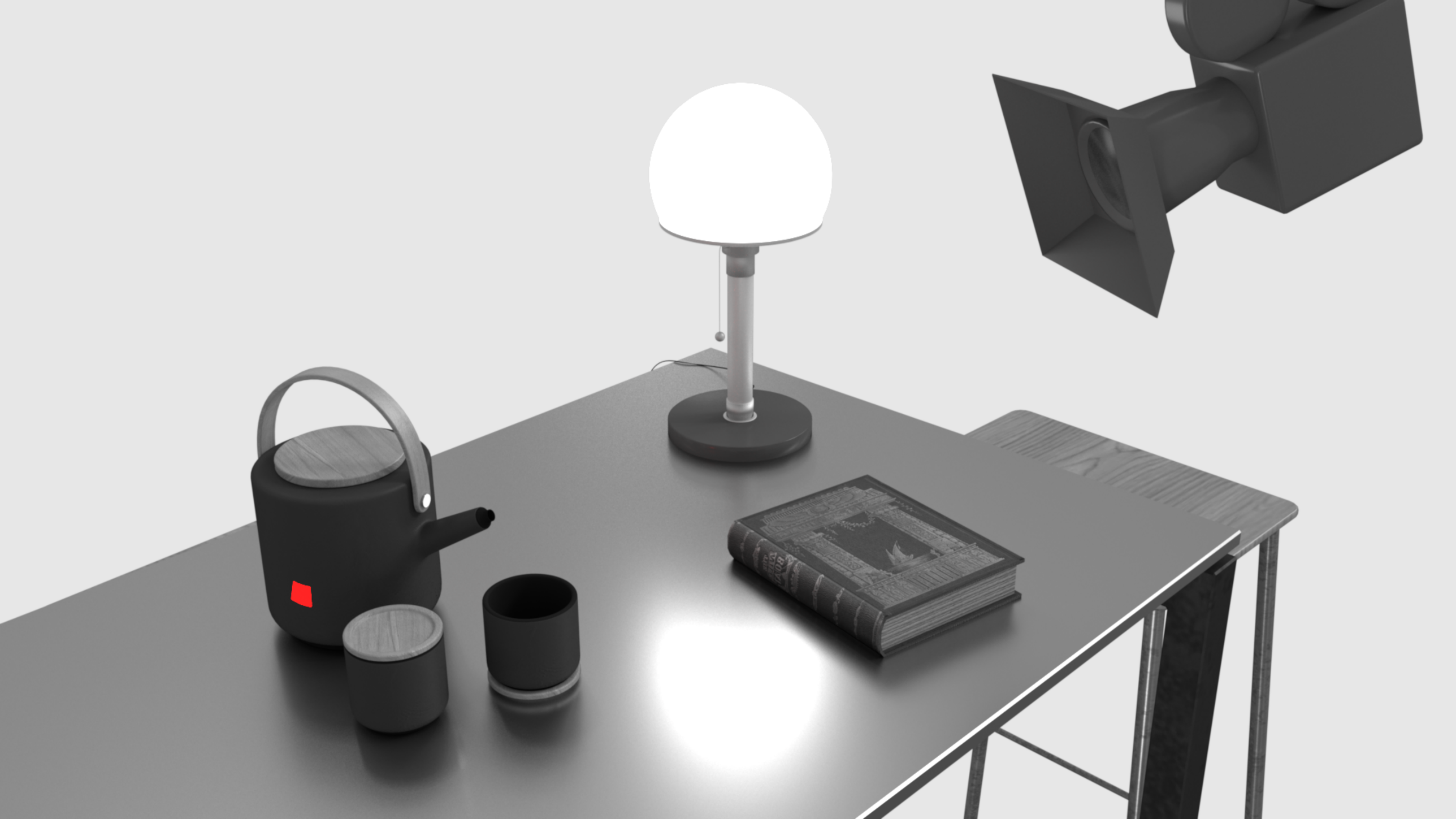
$$+ \int_{\Omega} \partial_x \left[L_i(\mathbf{x},\omega')\, f_s(\mathbf{x},\omega,\omega') \cos\theta \right] d\omega'$$

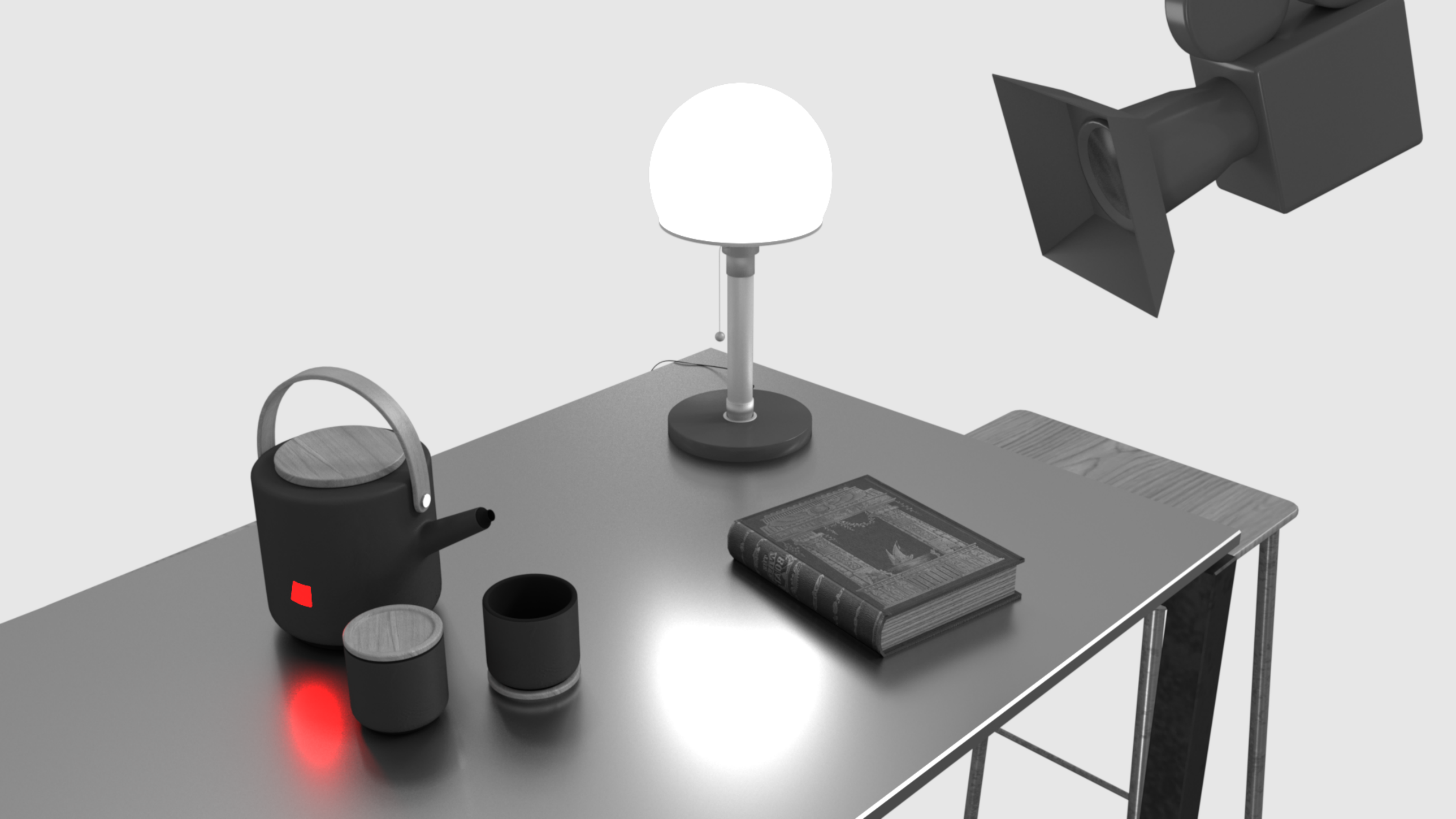
$$+ \partial_x \left[L_i(\mathbf{x},\omega')\, f_s(\mathbf{x},\omega,\omega') \cos\theta \, d\omega' \right]$$

TL;DR

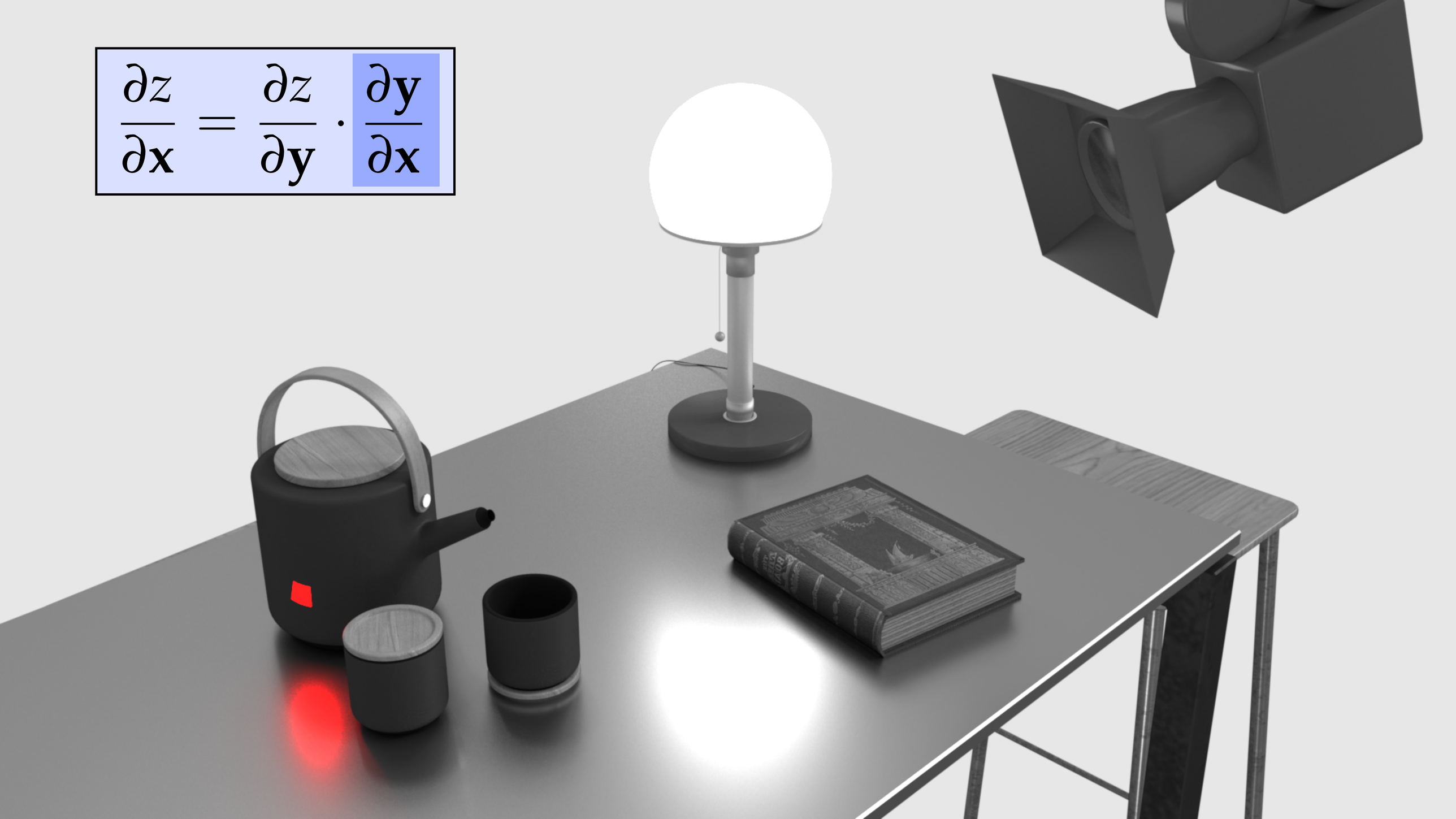
Differential radiance is “emitted” by scene objects with differentiable parameters

Differential radiance “scatters” like normal radiance

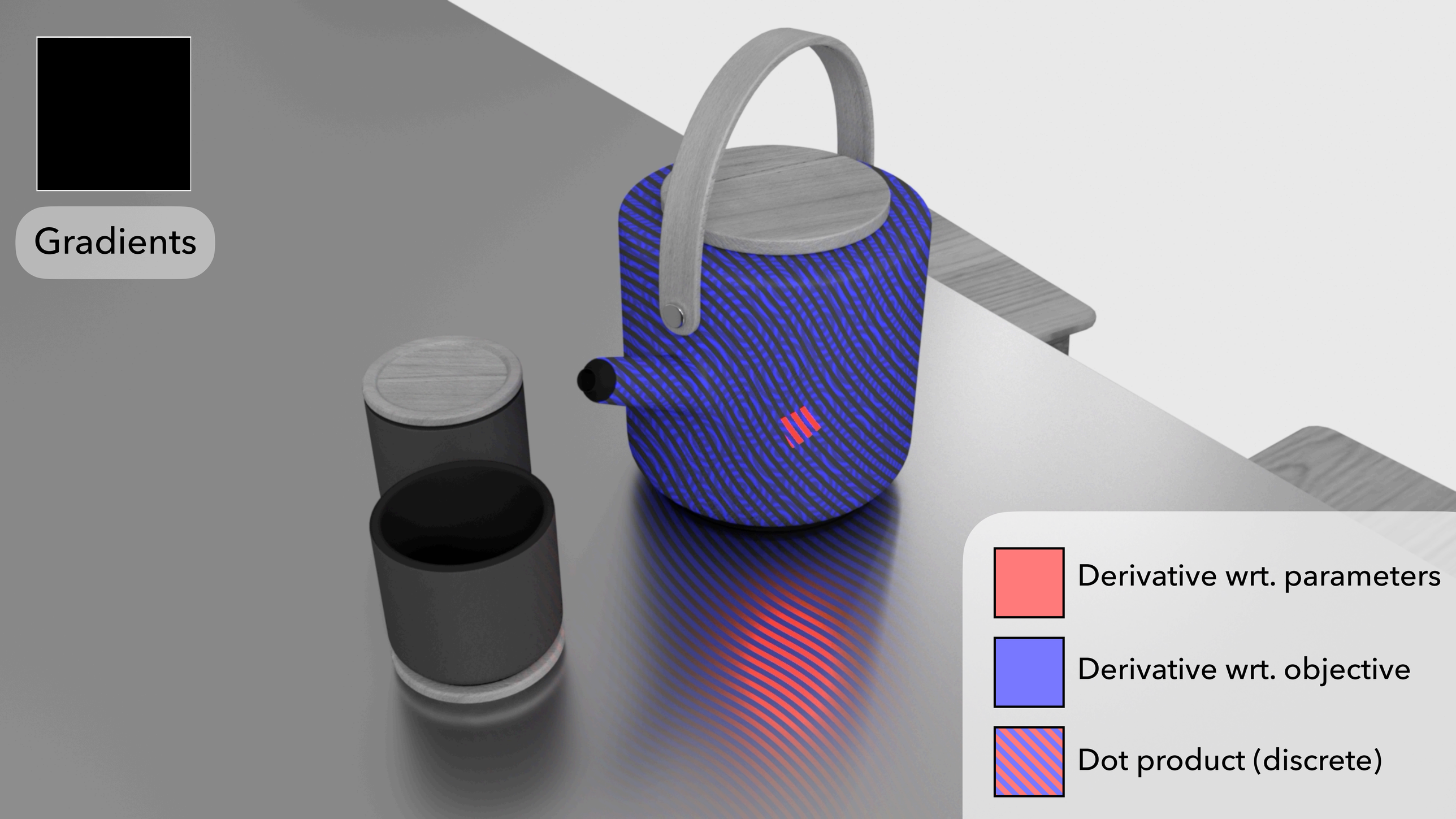


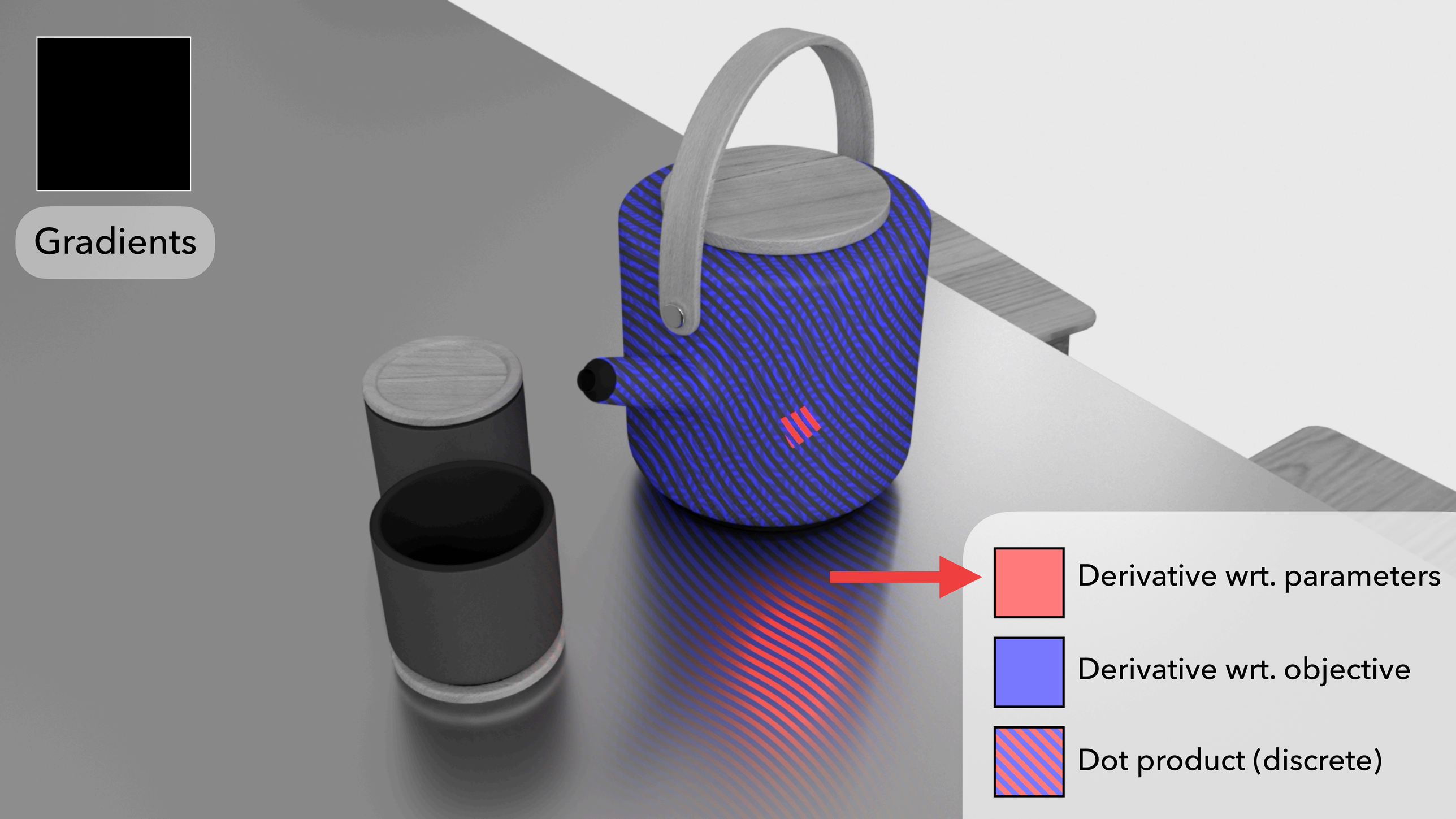


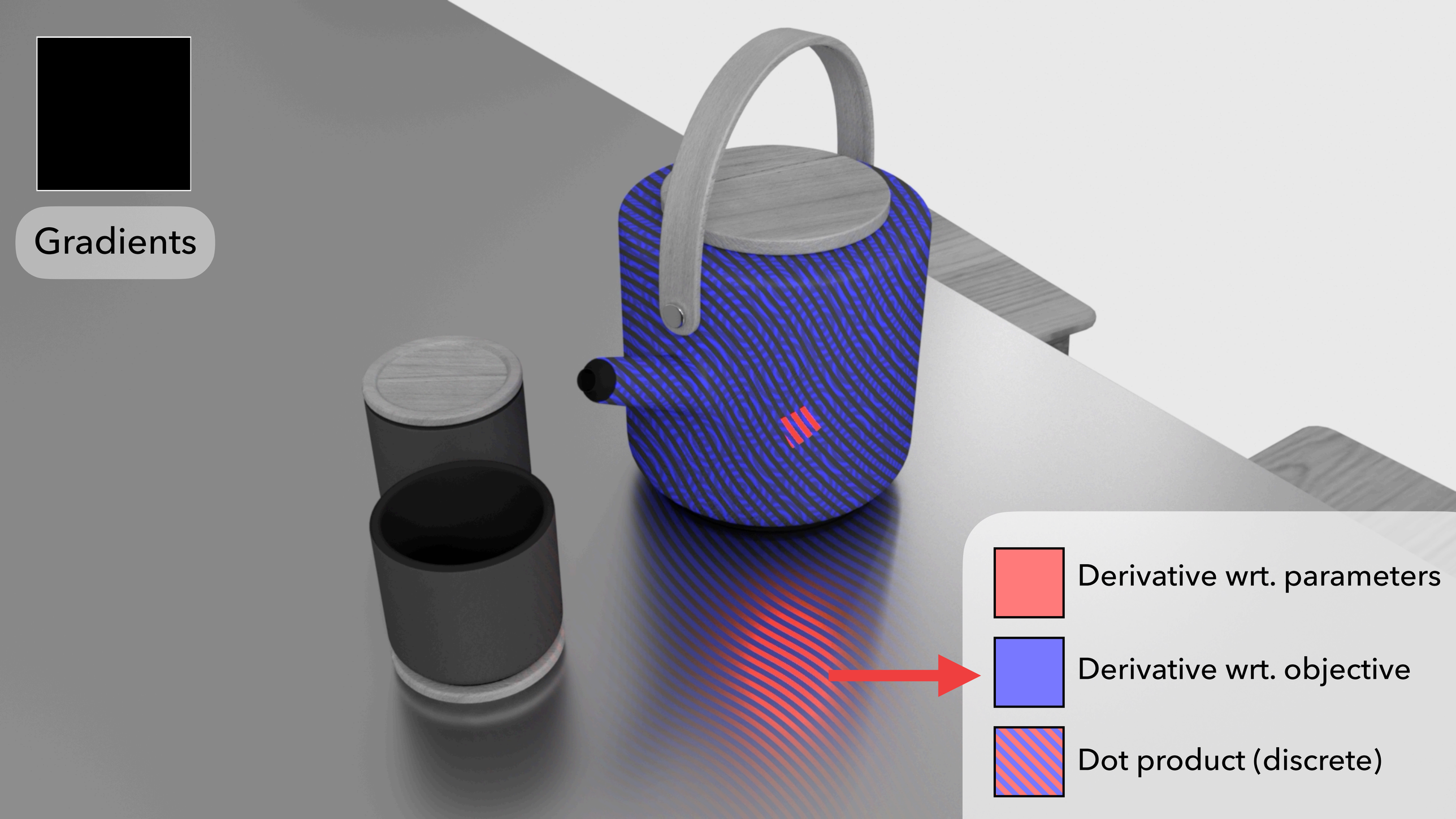
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

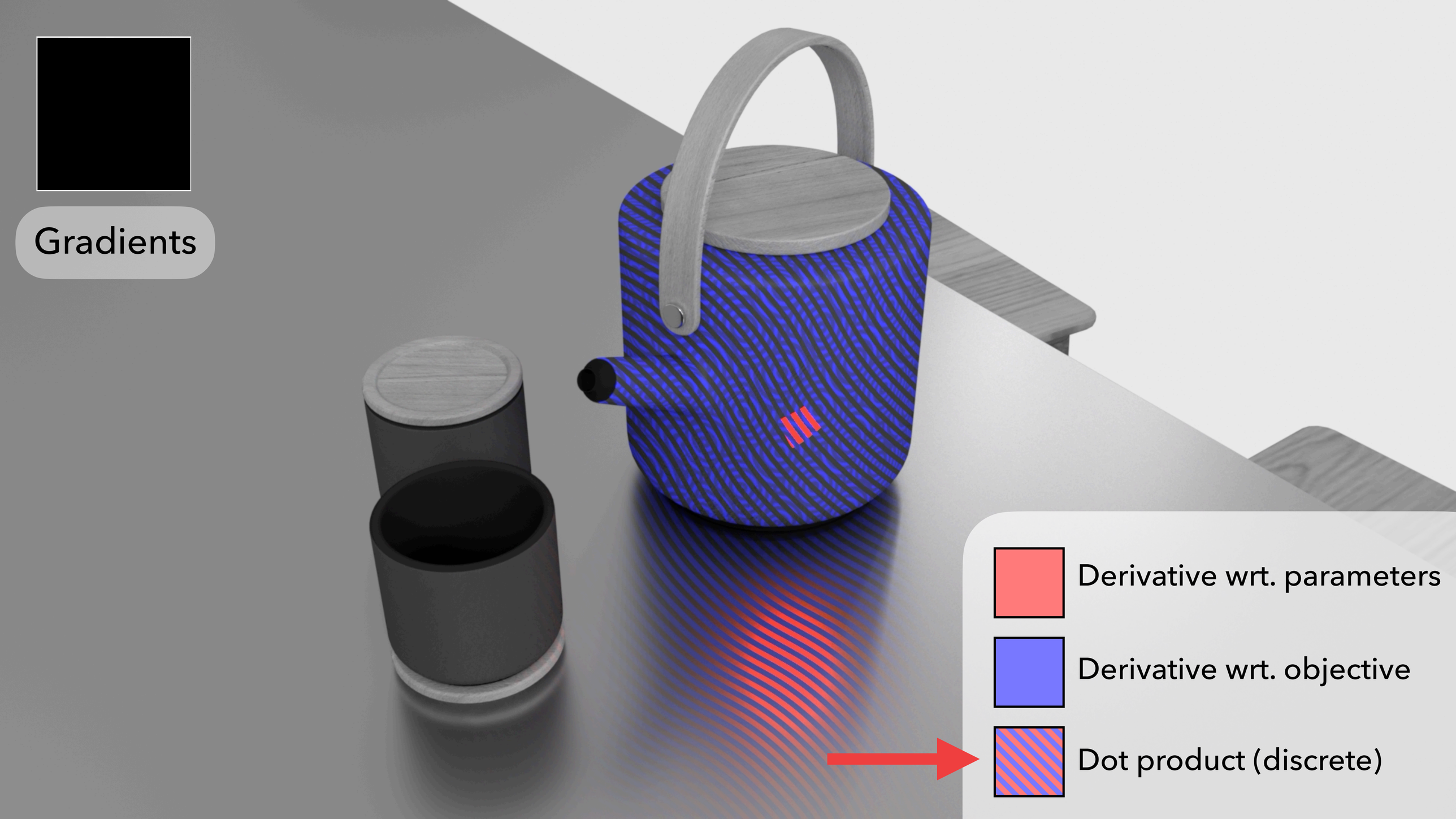


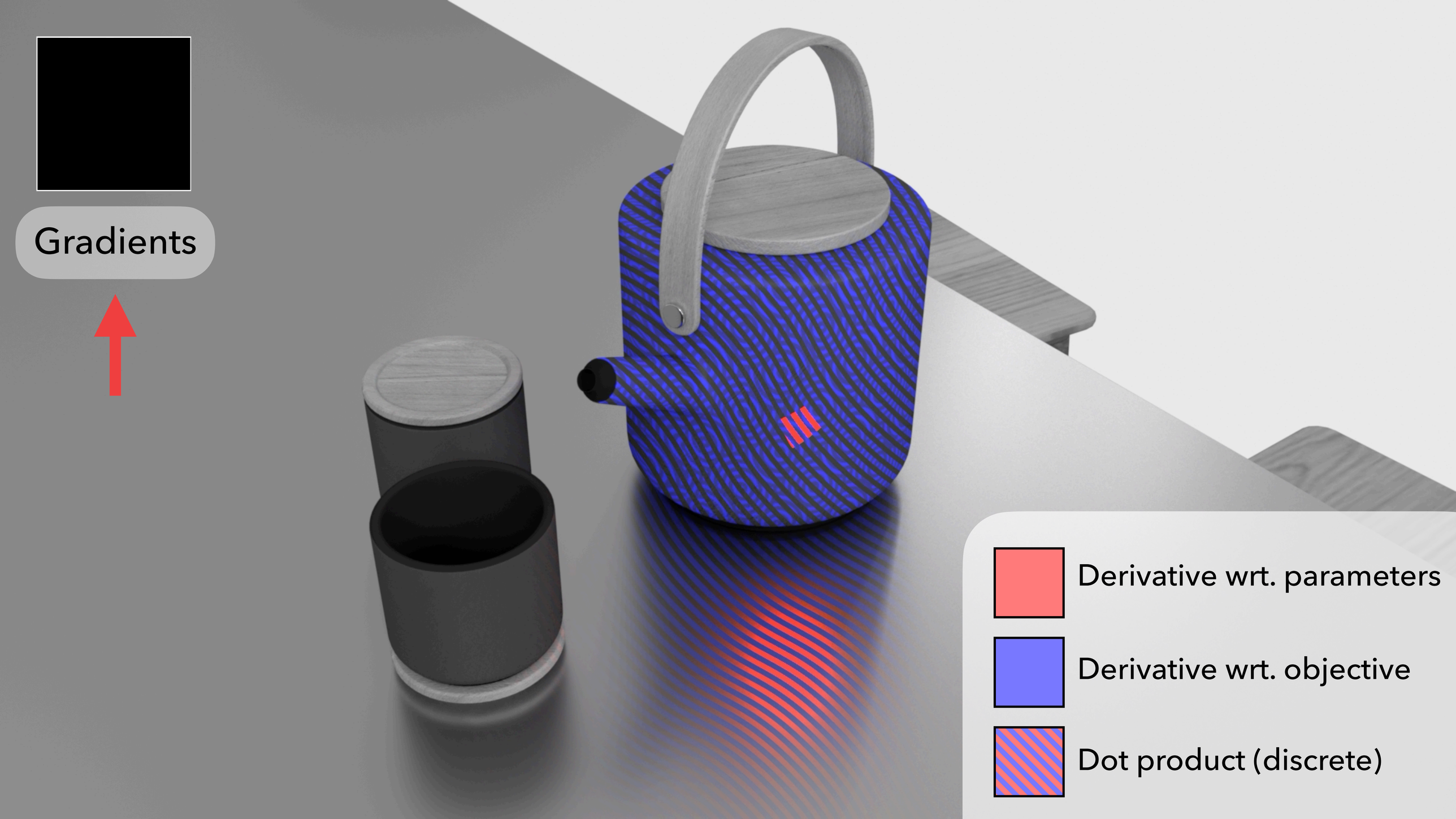
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$


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


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


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


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

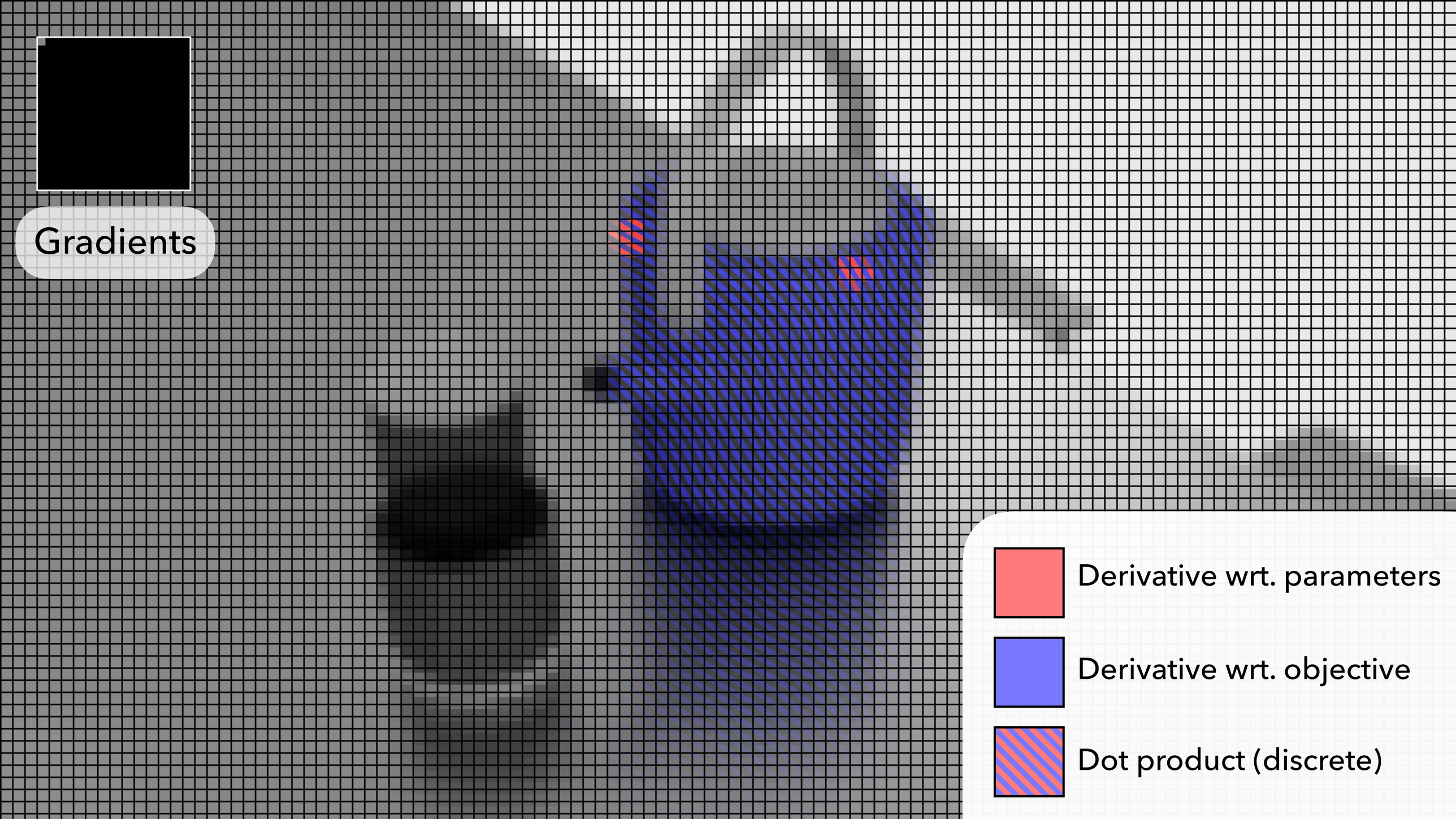

Gradients
Derivative wrt. parameters
Derivative wrt. objective
Dot product (discrete)

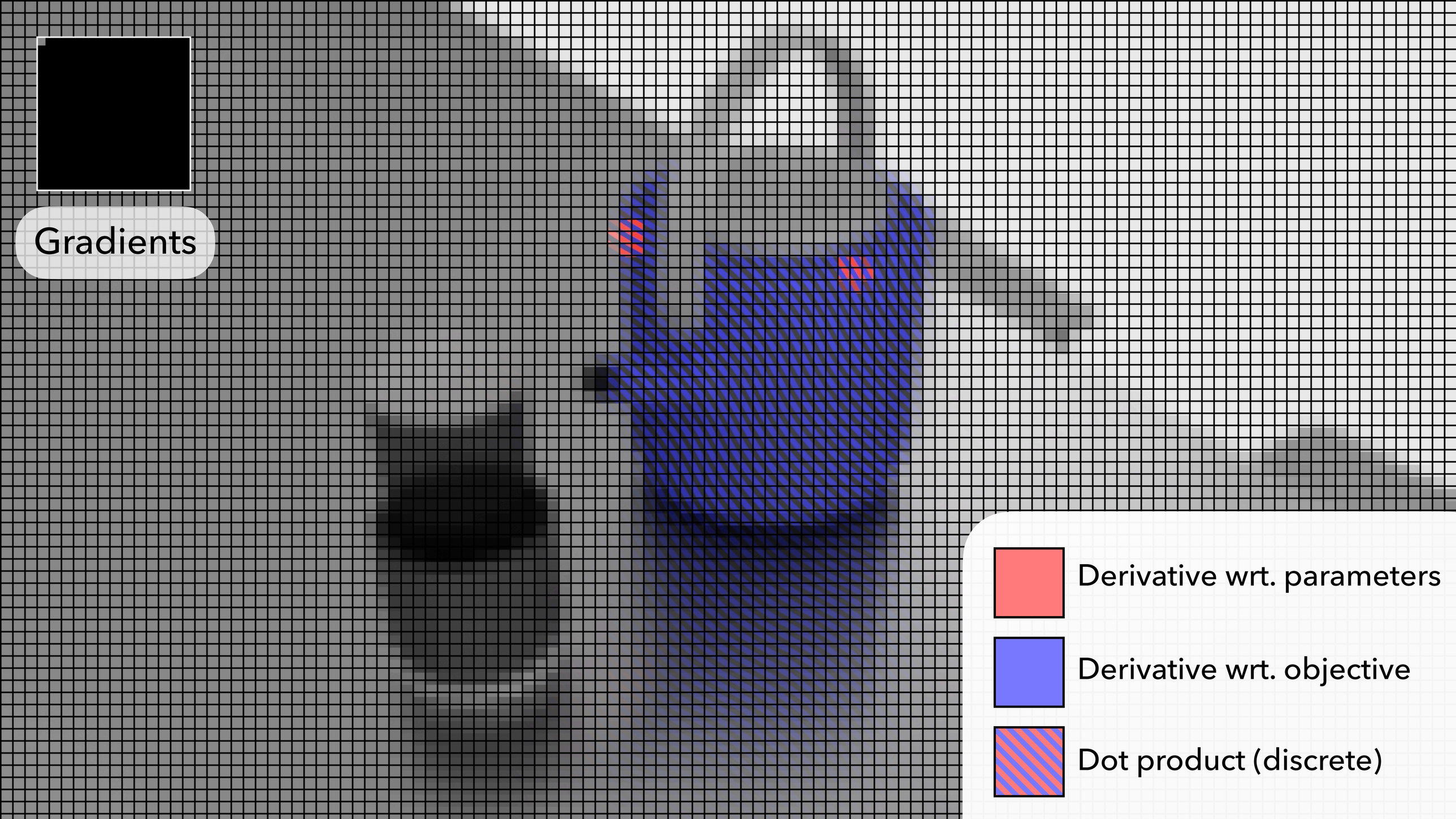
Gradients

Derivative wrt. parameters

Derivative wrt. objective

Dot product (discrete)


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


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

What is wrong with this approach?

1MPix rendering &
1M parameters:

What is wrong with this approach?

1MPix rendering &
1M parameters:

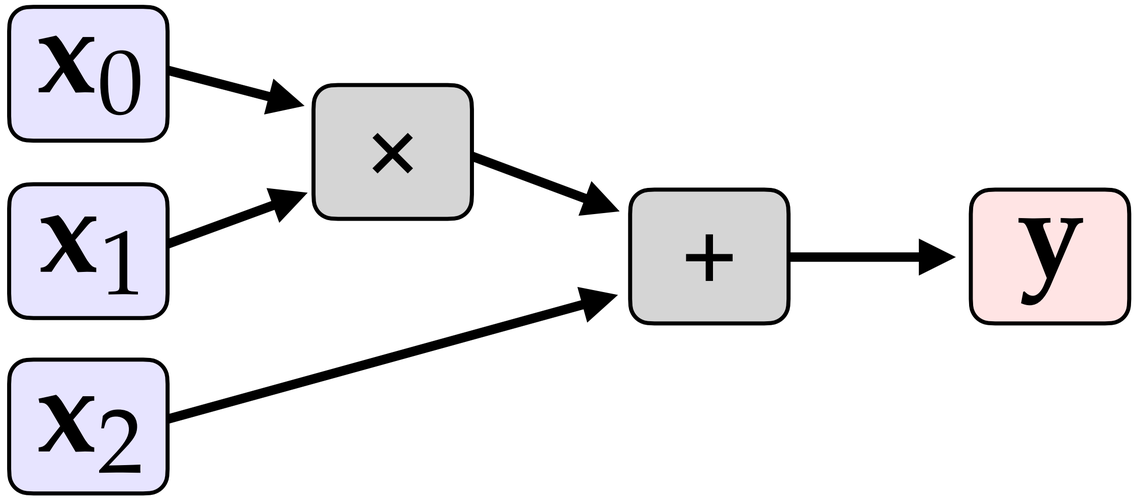
$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{1000000 \times 1000000}$$

(~3.6 TiB)

Directionality of differentiation

Forward mode

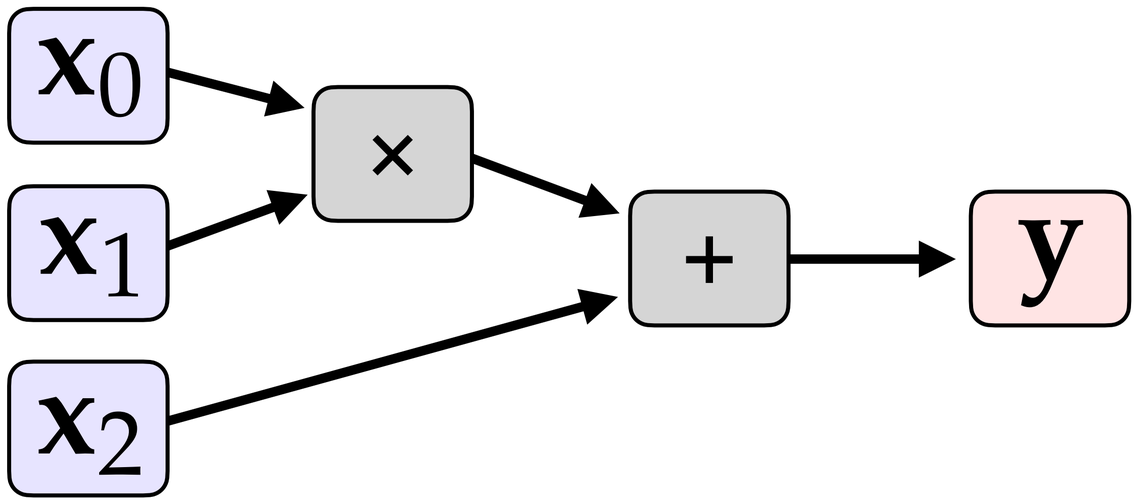
$$\mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 + \mathbf{x}_2$$

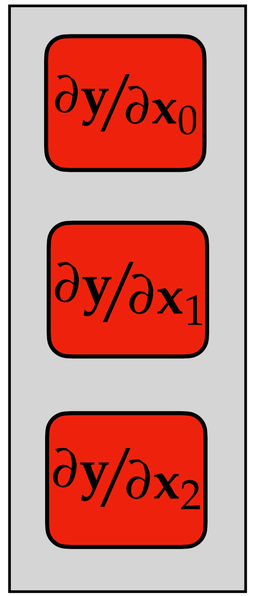


Directionality of differentiation

Forward mode

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



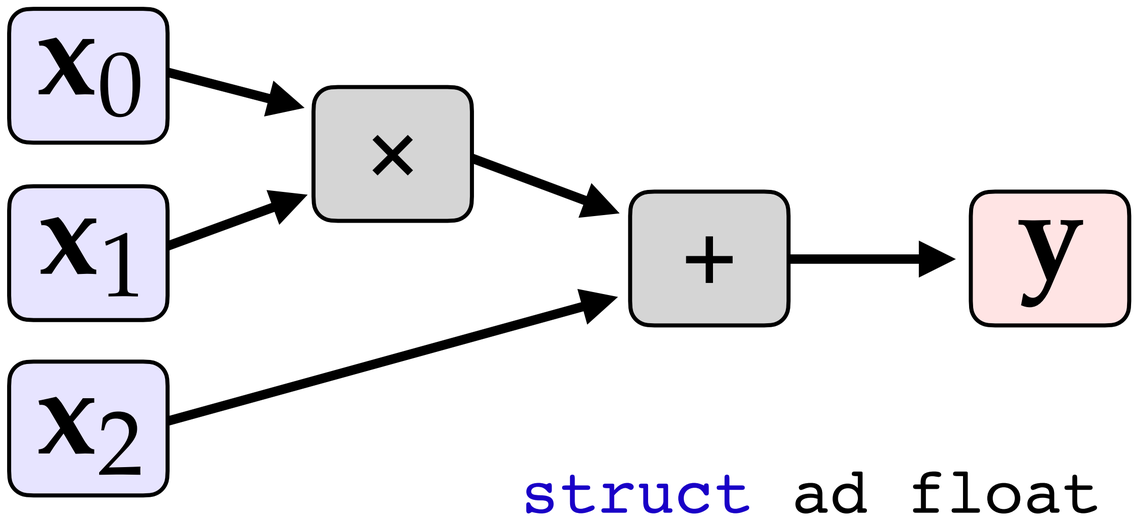


Gradient

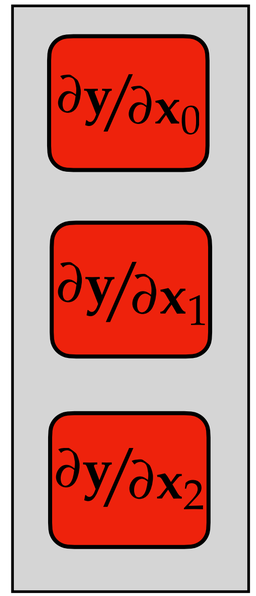
Directionality of differentiation

Forward mode

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



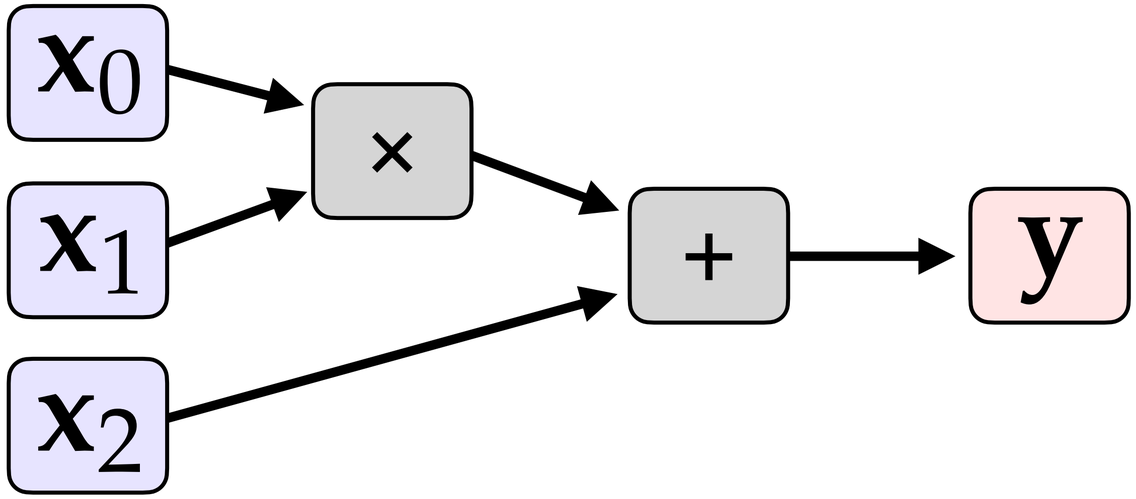

```
struct ad_float {
    float value;
    float derivative;
};
```



Gradient

Directionality of differentiation

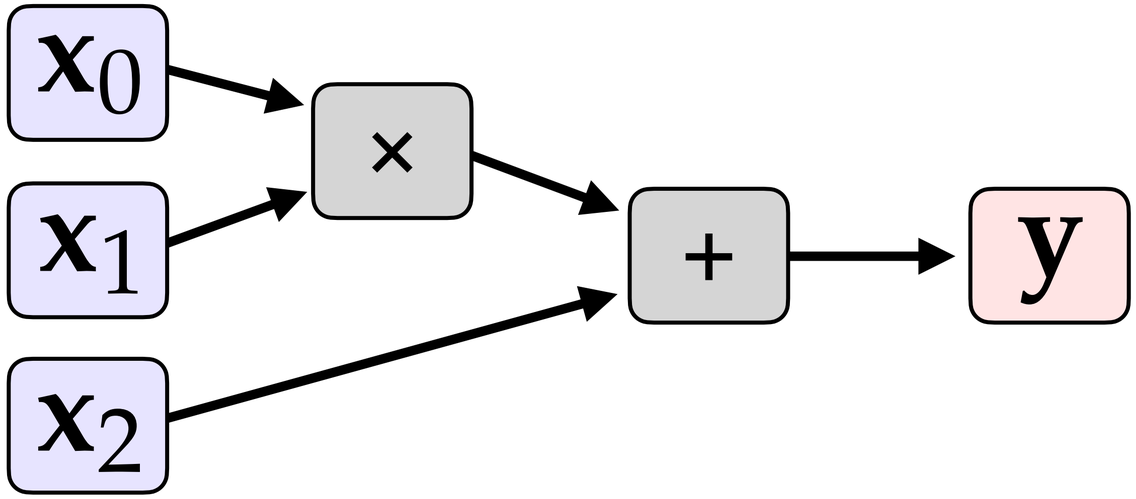
$$\mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 + \mathbf{x}_2$$



Directionality of differentiation

Reverse mode

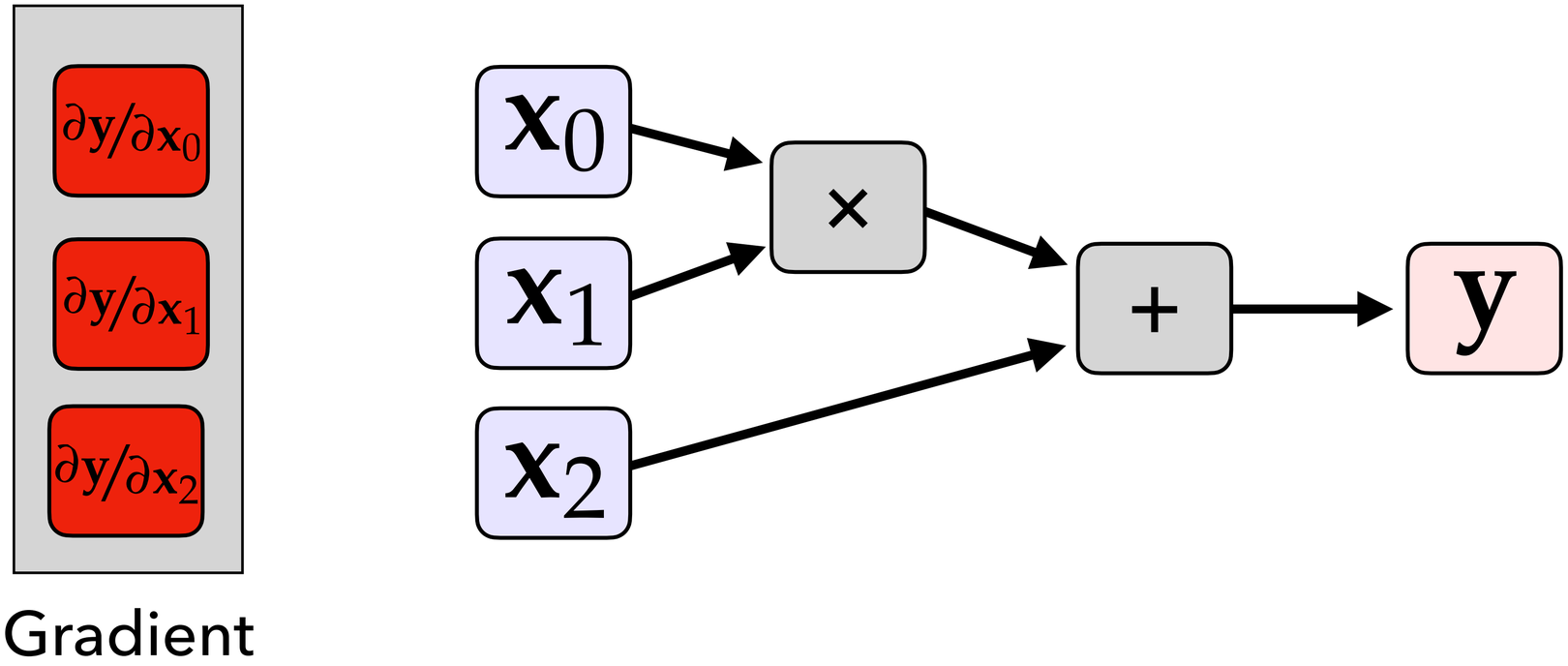
$$\mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 + \mathbf{x}_2$$



Directionality of differentiation

Reverse mode

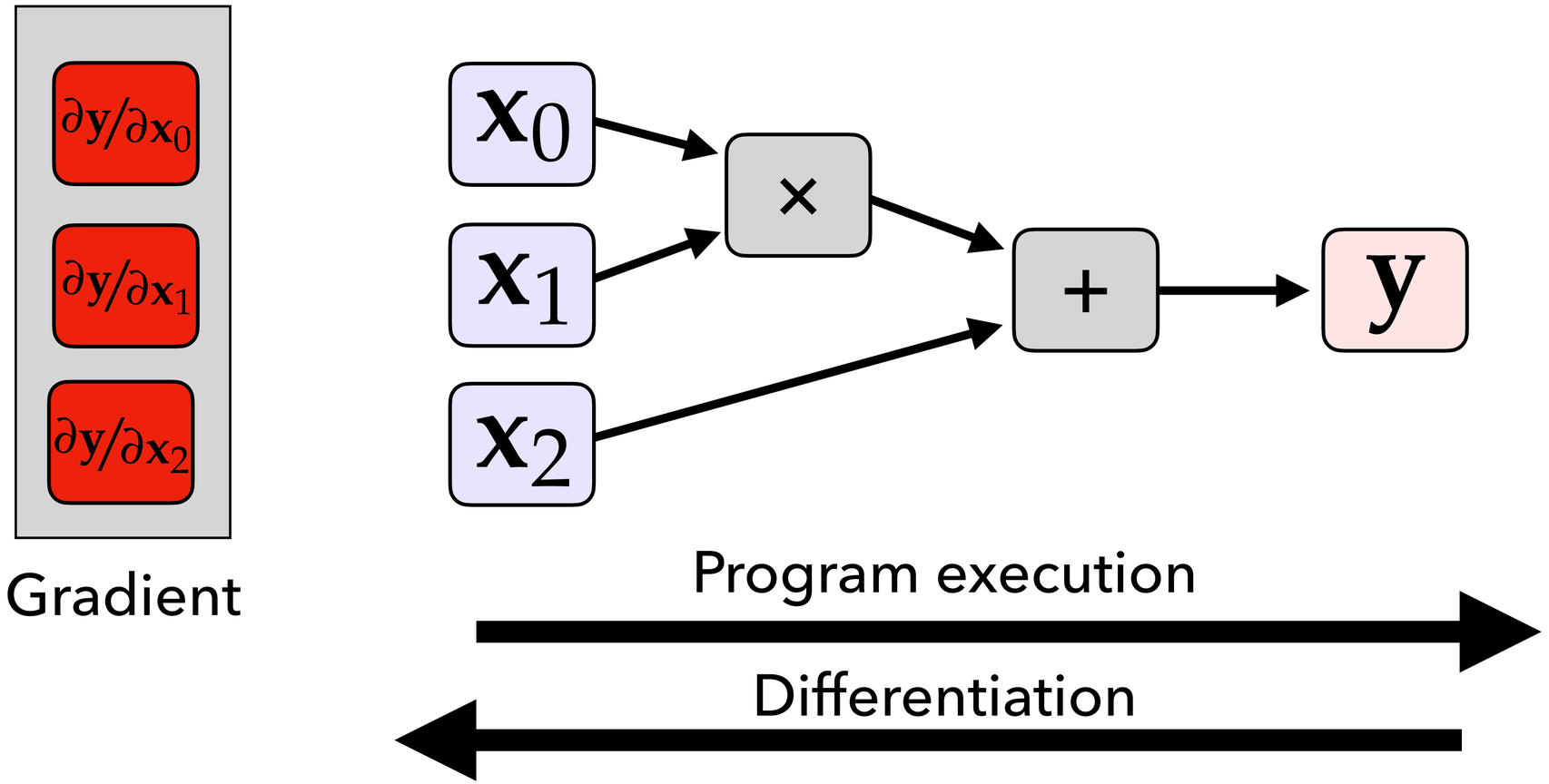
$$\mathbf{y} = \mathbf{x}_0 \cdot \mathbf{x}_1 + \mathbf{x}_2$$



Directionality of differentiation

Reverse mode

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



Differentiable rendering in Mitsuba 2

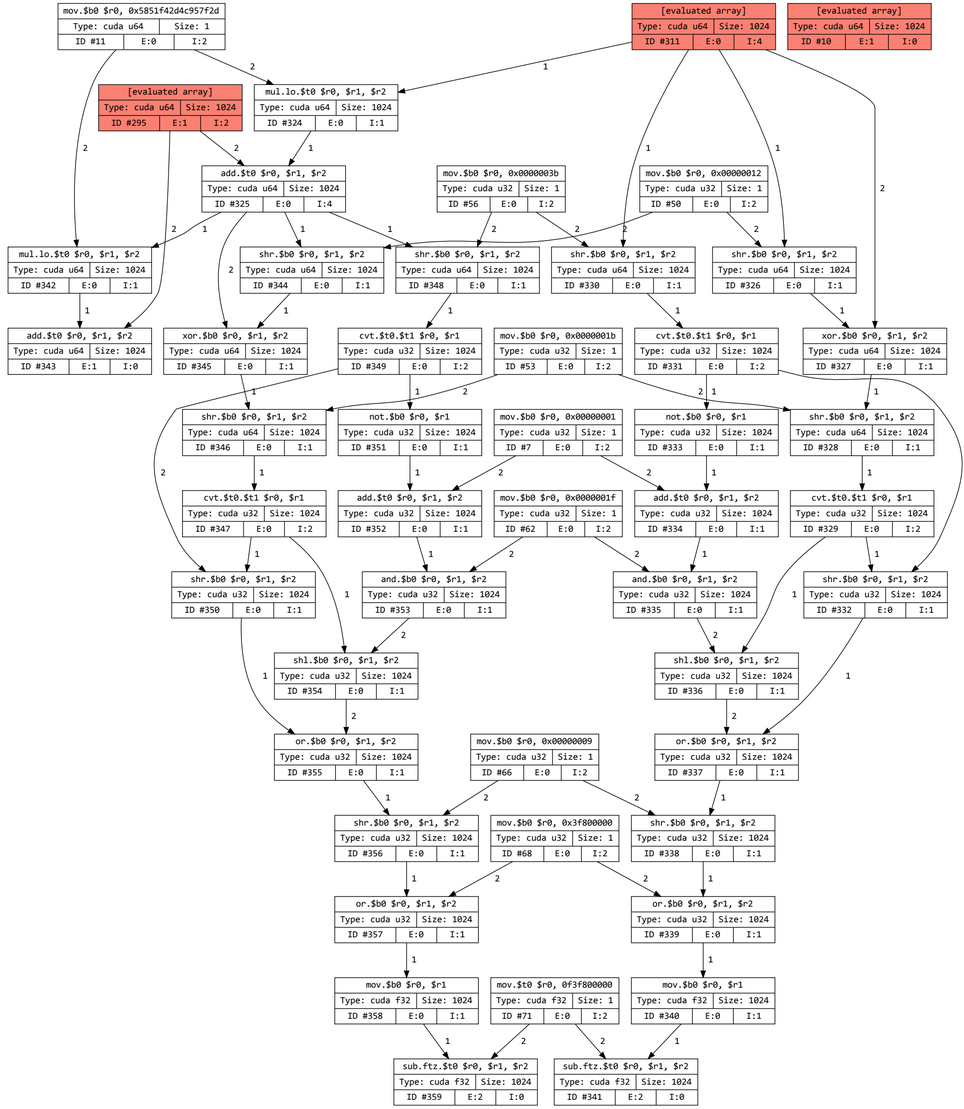
```
Point2f sample = sampler->next_2d();

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())
```

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())
```

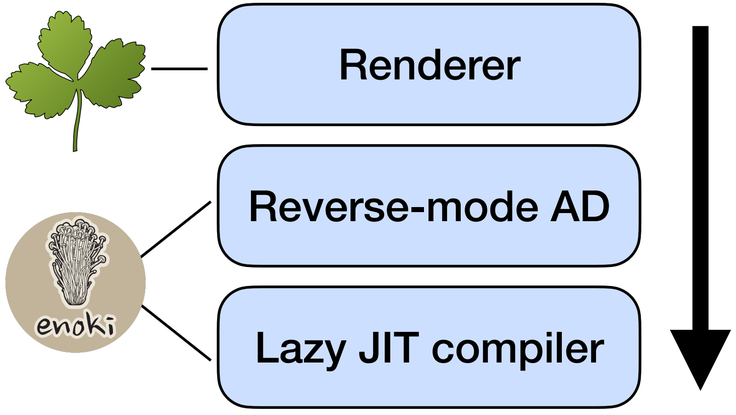


Differentiable rendering in Mitsuba 2

```
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())
```

Differentiable rendering in Mitsuba 2

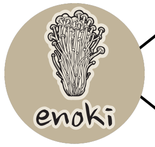
```
Point2f sample = sampler->next_2d();

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

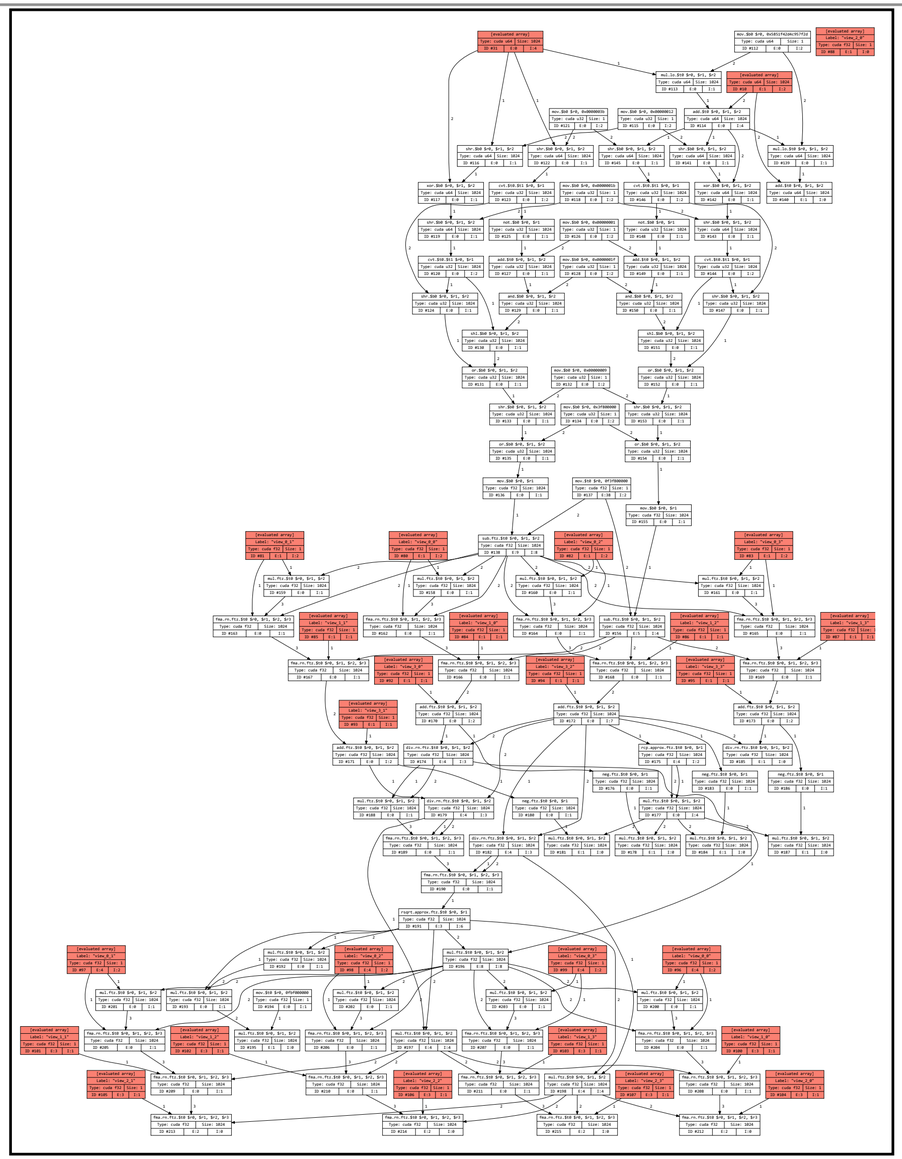
```
Point2f sample = sampler->next_2d();

Ray3f ray = camera->sample_ray(sample);

SurfaceInteraction3f si = scene->ray_intersect(ray)

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

Renderer



Reverse-mode AD

Lazy JIT compiler

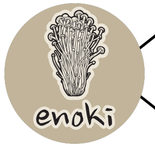


Differentiable rendering in Mitsuba 2

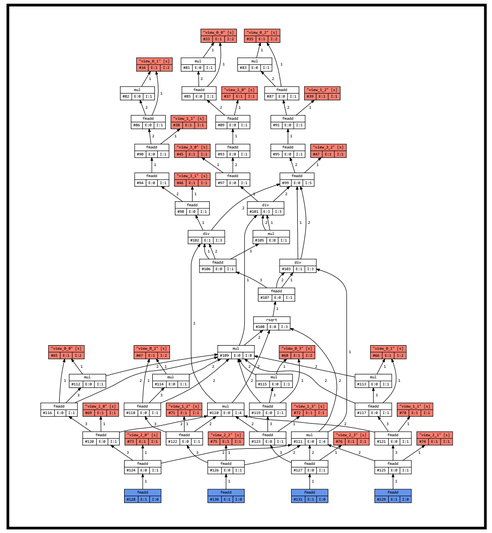
```
Point2f sample = sampler->next_2d();

Ray3f ray = camera->sample_ray(sample);

SurfaceInteraction3f si = scene->ray_intersect(ray)

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

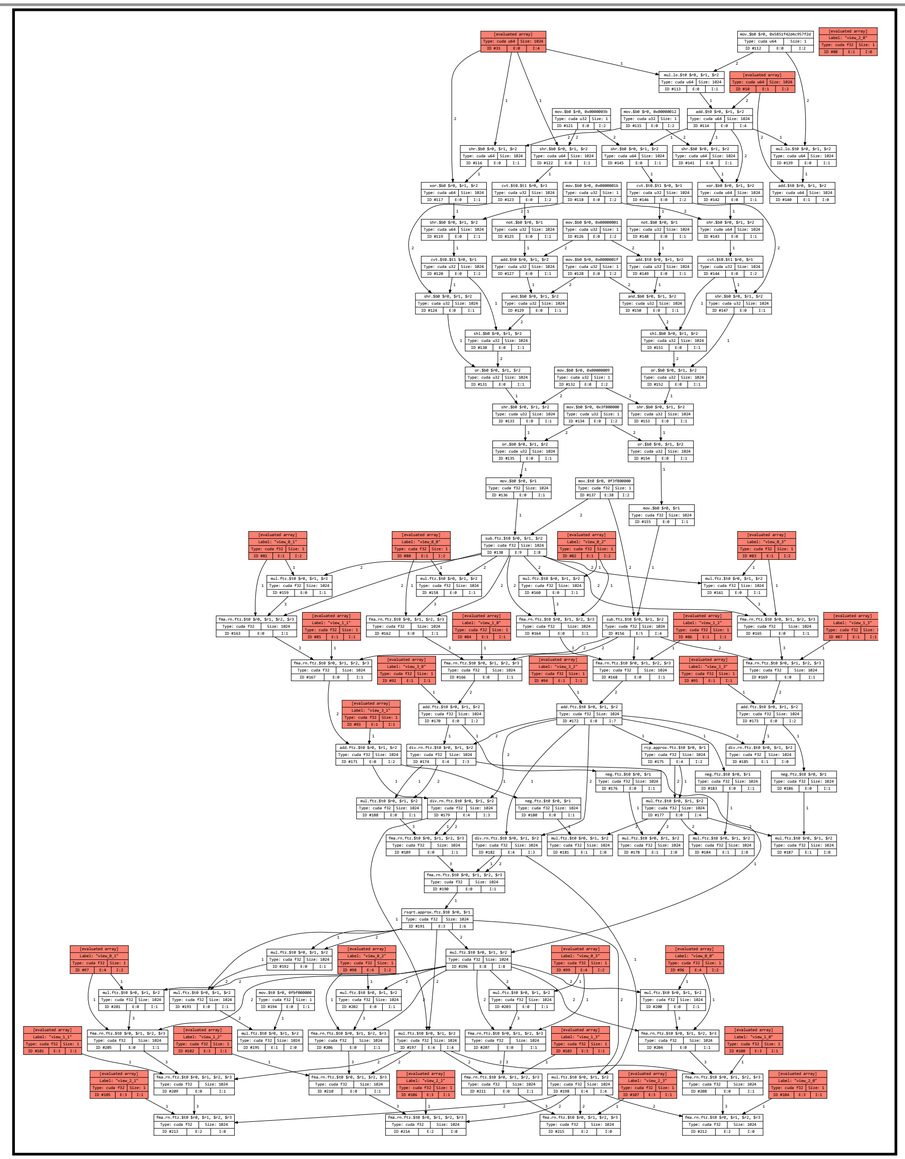
Renderer

Reverse-mode AD



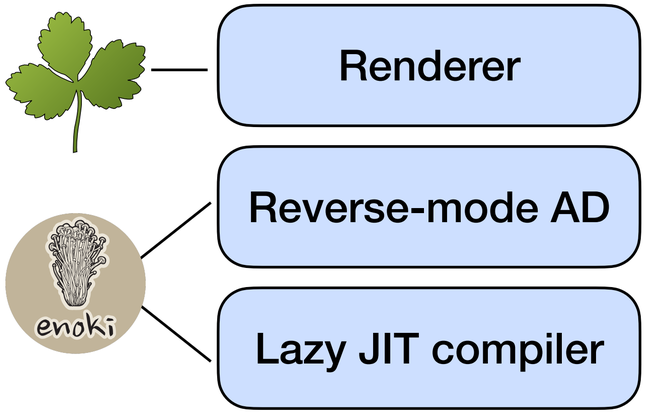
Lazy JIT compiler





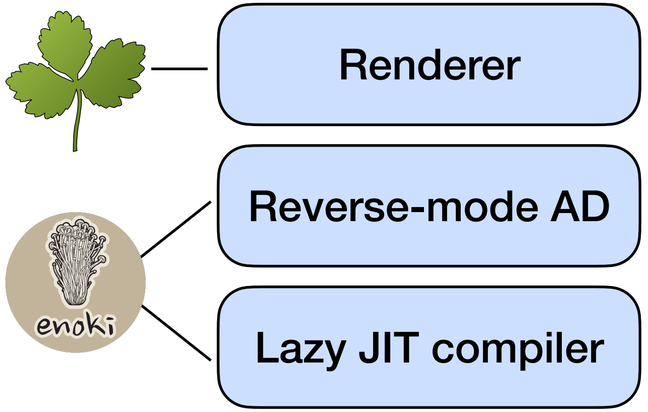
Differentiable rendering in Mitsuba 2

```
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())
```



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())
```

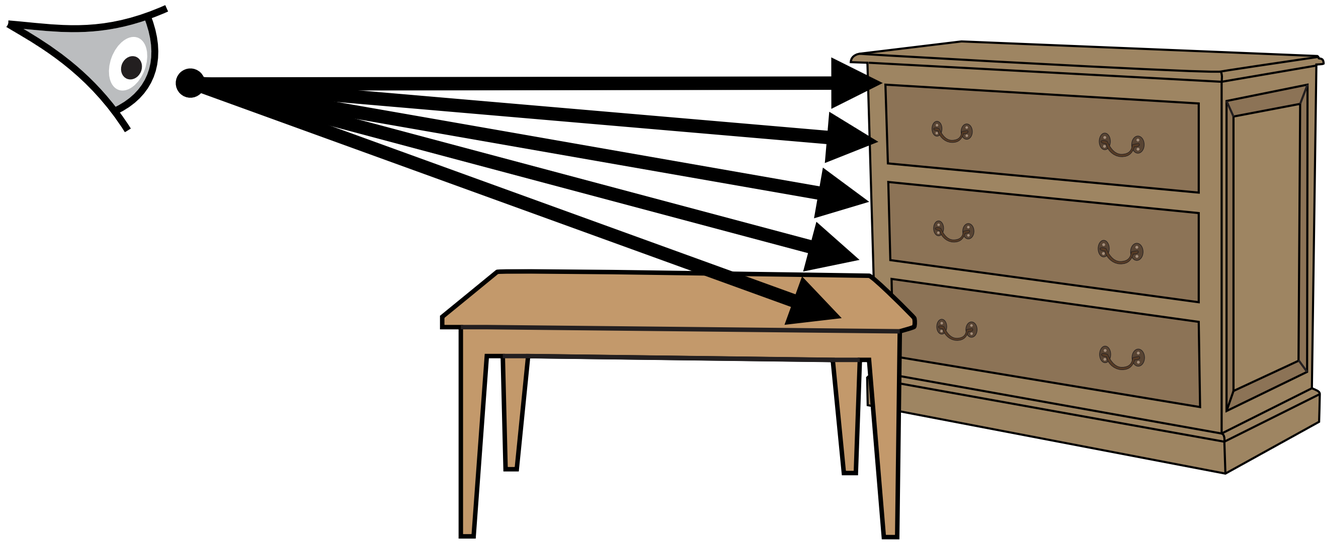
- Compilation is *fast* (~100 *us*) (just hash table lookups + string concatenation)
- PTX (CUDA), soon: LLVM (CPU)
- Caches compiled kernels
- Can prototype rendering code in Jupyter notebooks with reasonable performance.

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())
```

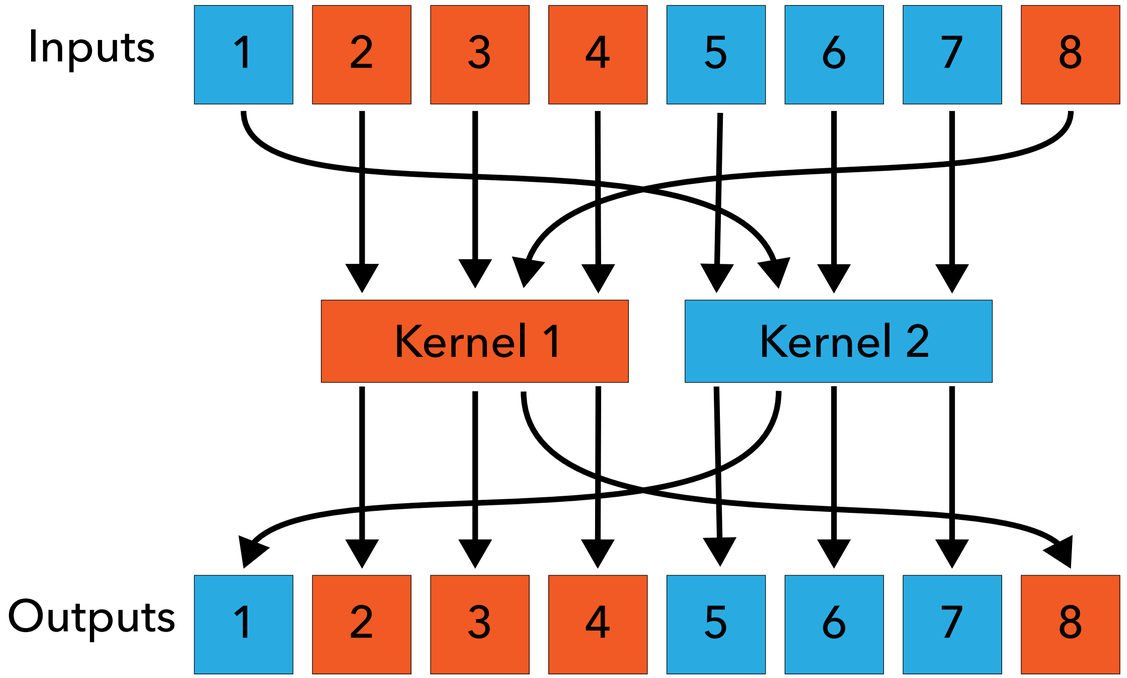
Differentiable rendering in Mitsuba 2

```
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())
```

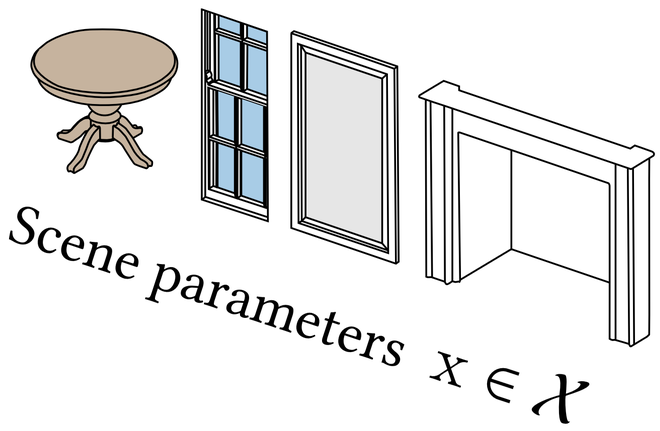


Differentiable rendering in Mitsuba 2

```
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())
```



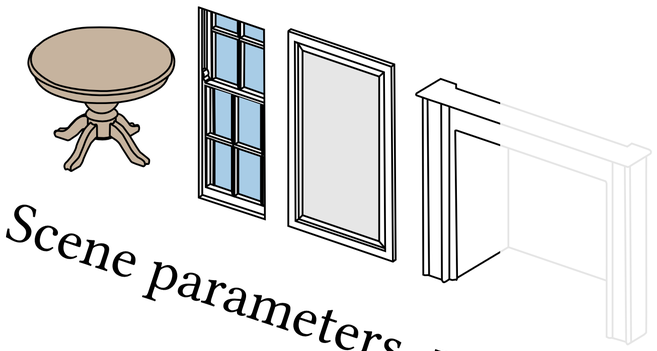
Autodiff-based differentiable rendering



Scene parameters $x \in \mathcal{X}$

Rendering algorithm

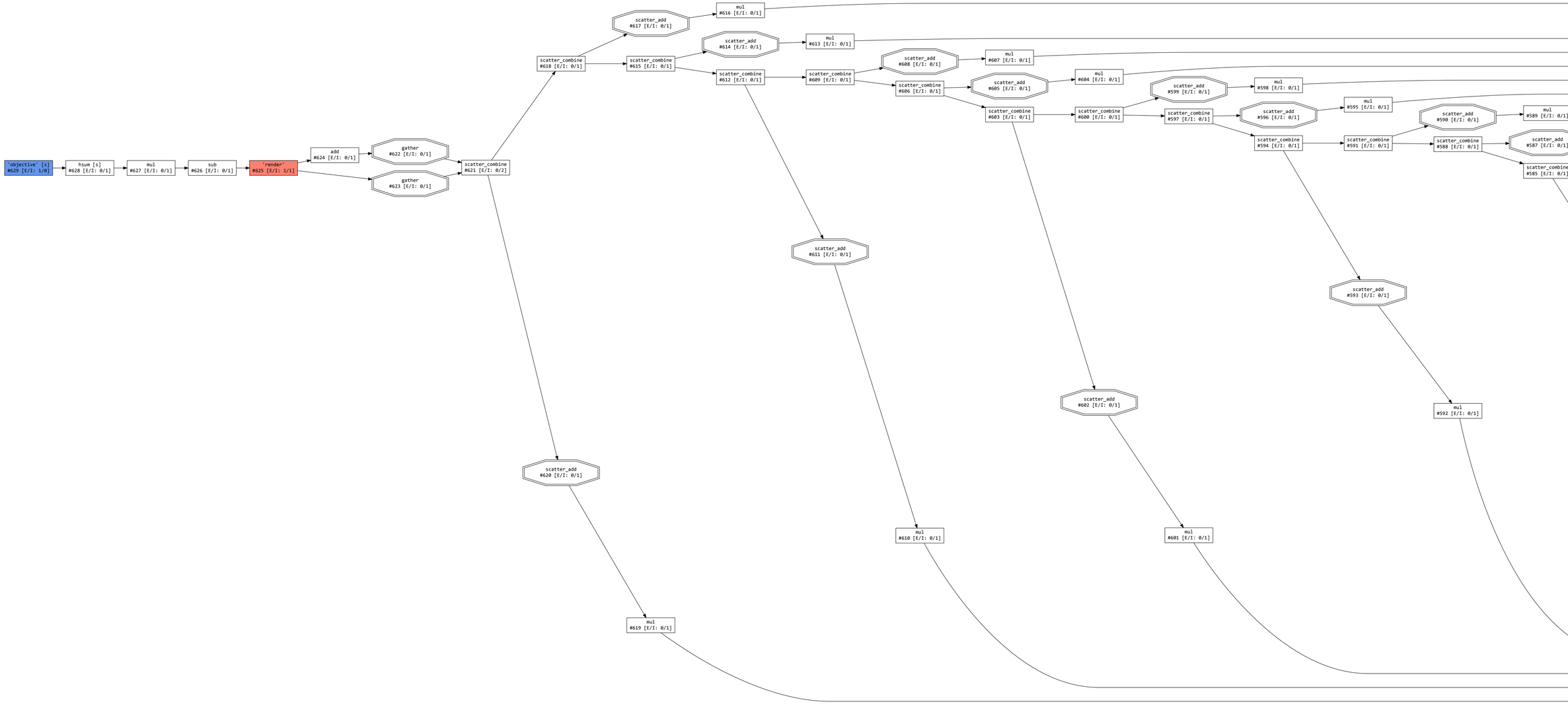
Autodiff-based differentiable rendering



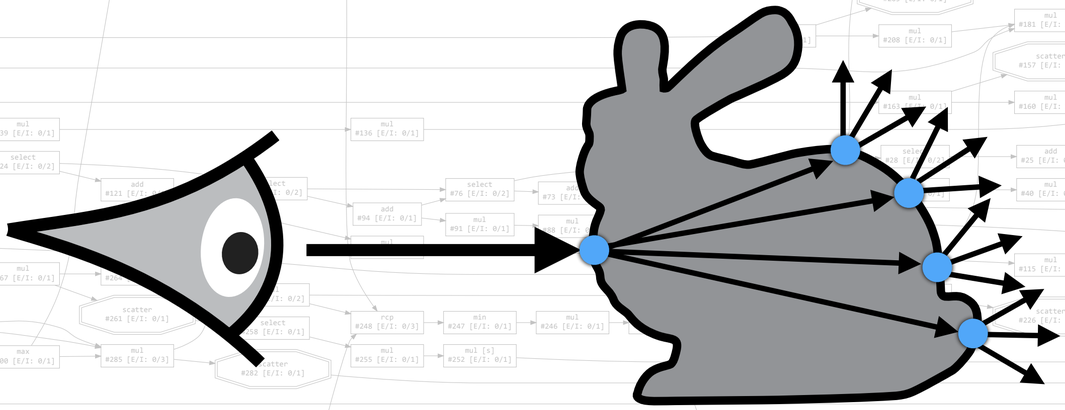
Scene parameters $x \in \mathcal{X}$

Vector explosion by *upklyak* at Freepik

Differentiable rendering using reverse-mode AD



Differentiable rendering using reverse-mode AD



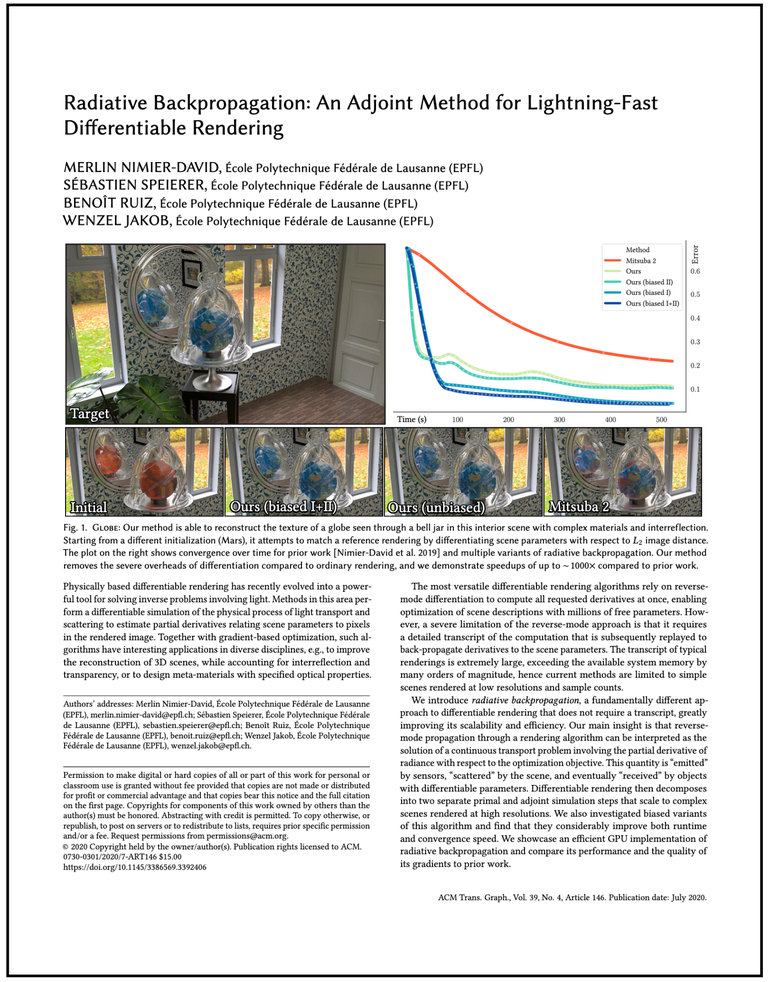
Kind of depressing ..

- **Forward mode**
  Too slow for interesting problems with many parameters.

- **Reverse mode**
  Computation graph explodes even having spent a considerable amount of time on optimizations (JIT, graph simplification, etc.)

- **What now?**

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)
BENOÎT 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

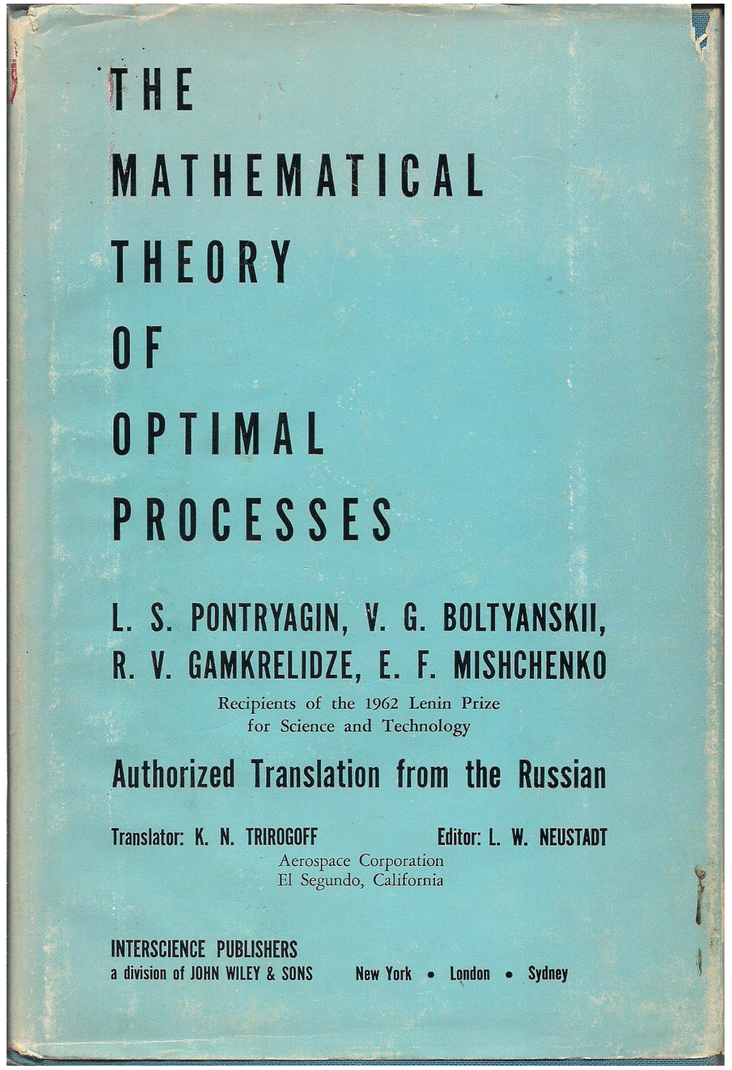
SIGGRAPH 2020

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

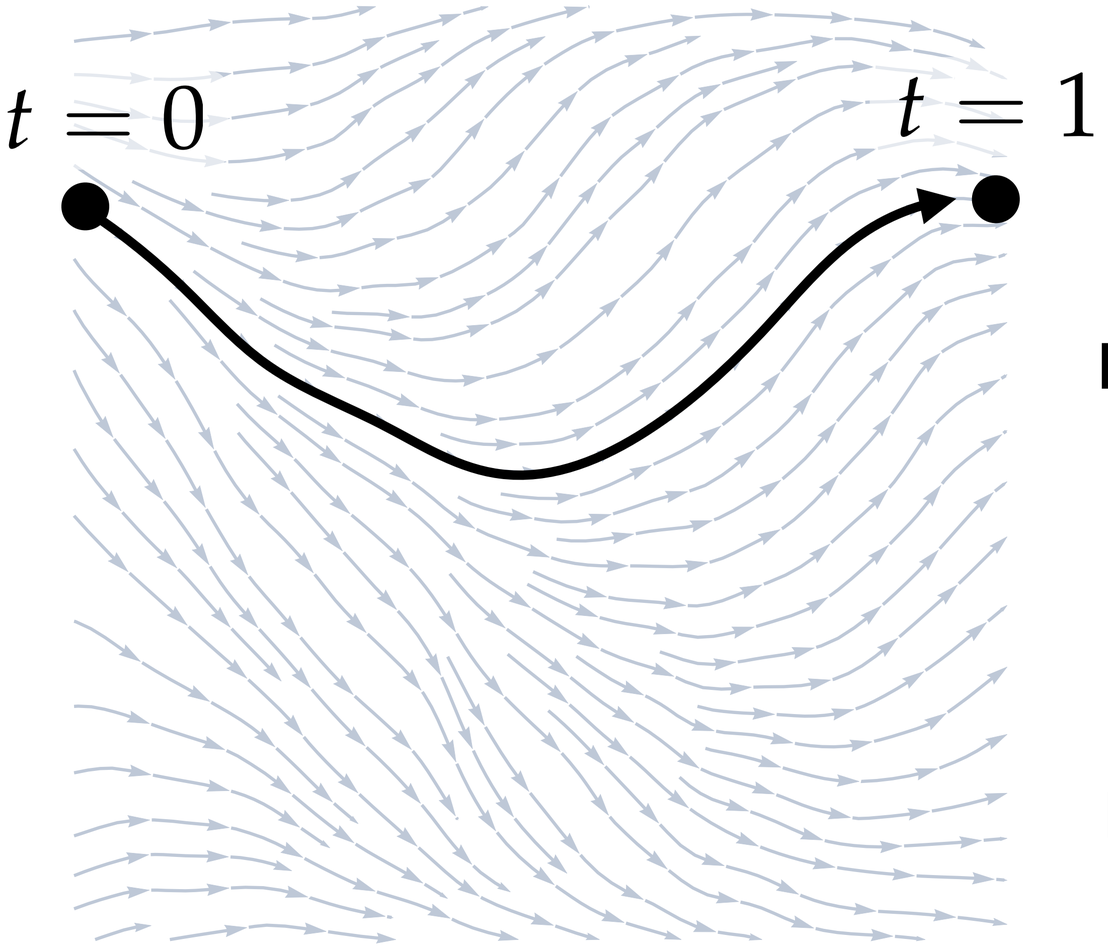
Motivation: Adjoint Sensitivity Method

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

Pontryagin et al.
1962

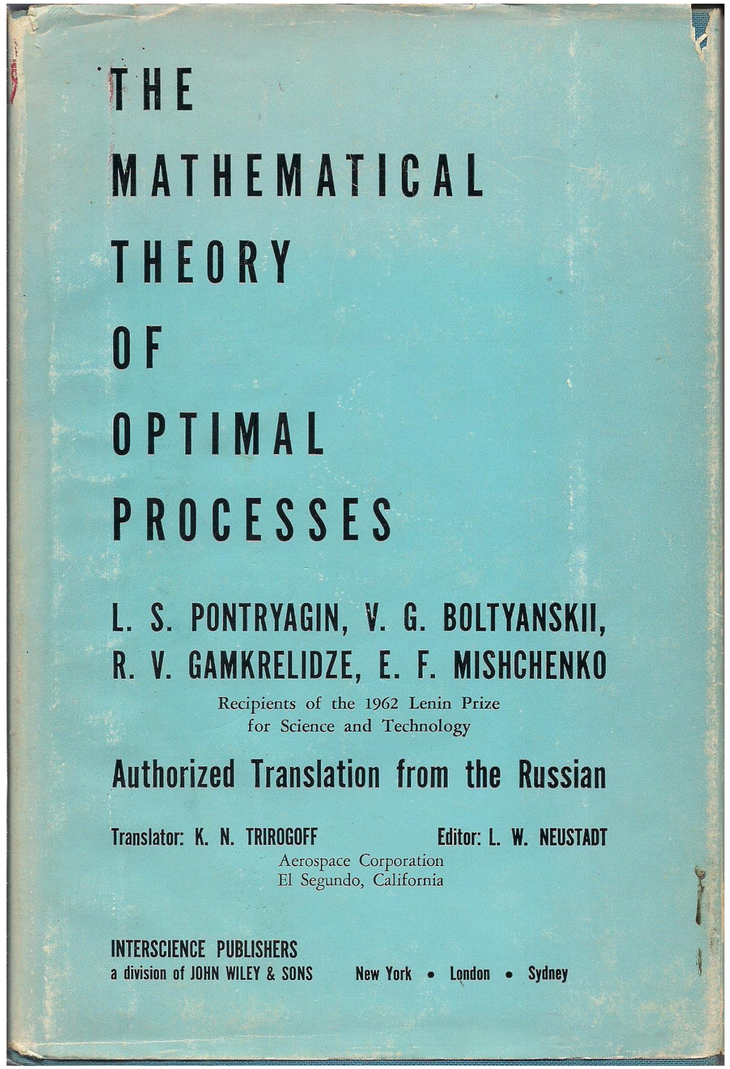


Motivation: Adjoint Sensitivity Method

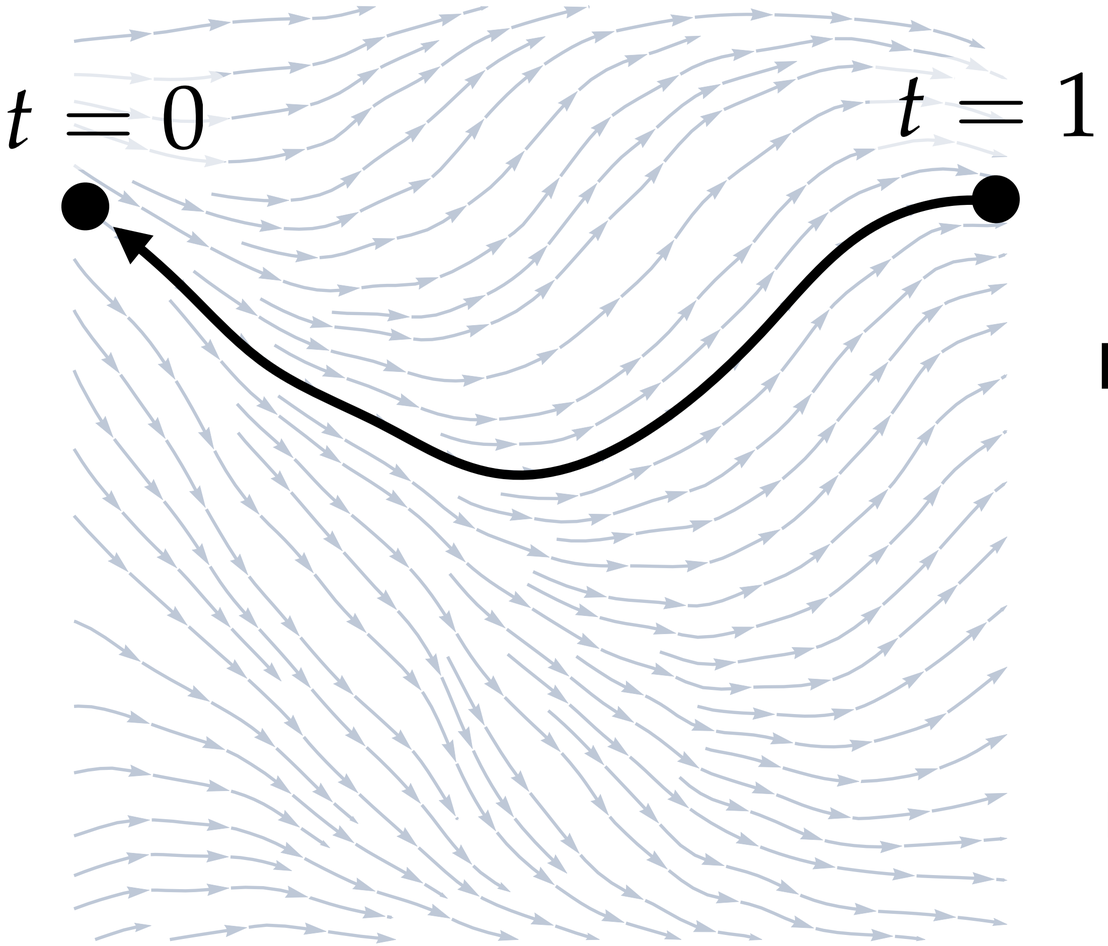


For problems with a time dimension (ODEs, ..)

Pontryagin et al.
1962

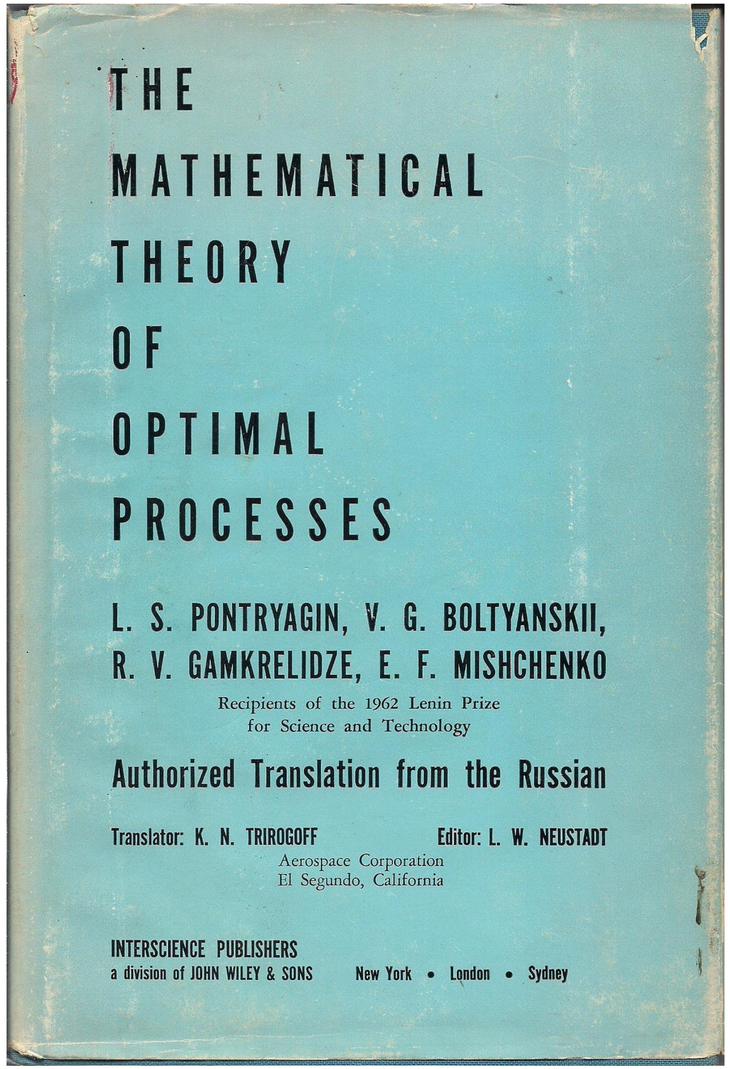


Motivation: Adjoint Sensitivity Method

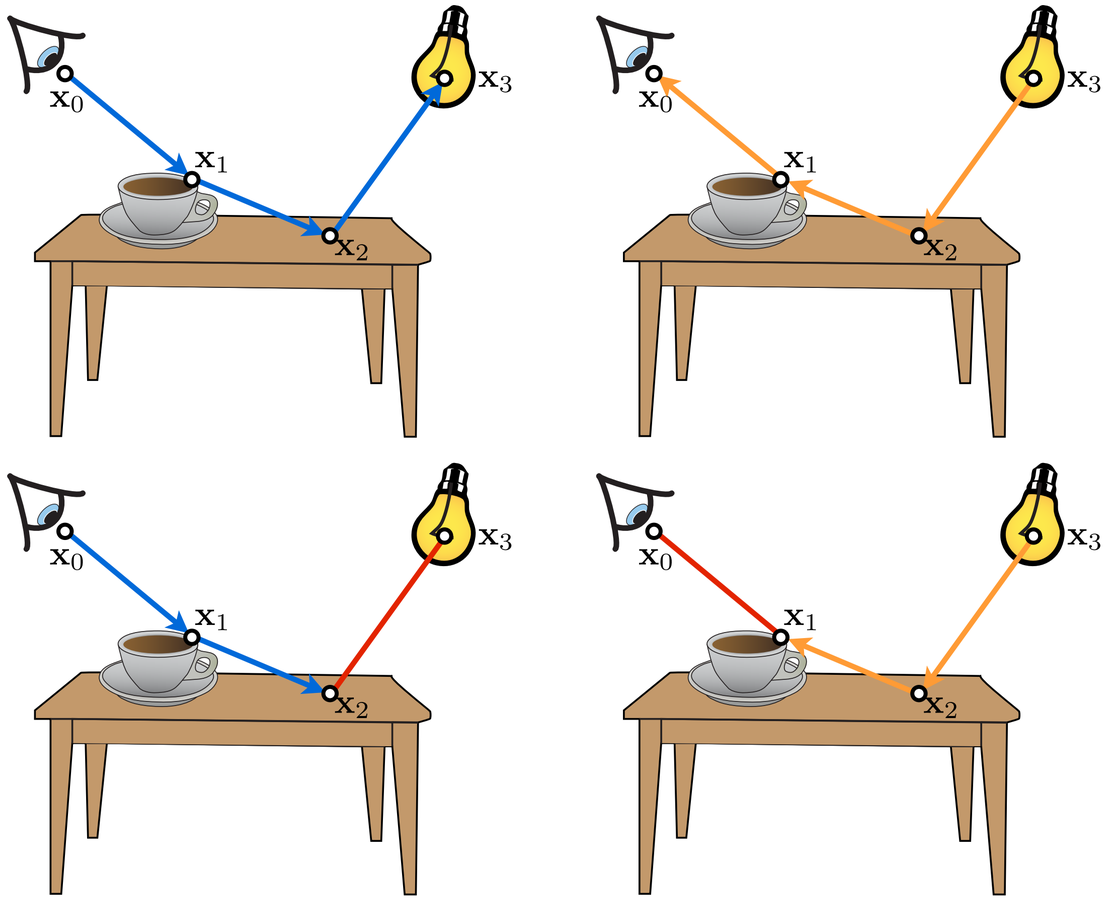


For problems with a time dimension (ODEs, ..)

Pontryagin et al.
1962



“Adjoint” – that sounds familiar!

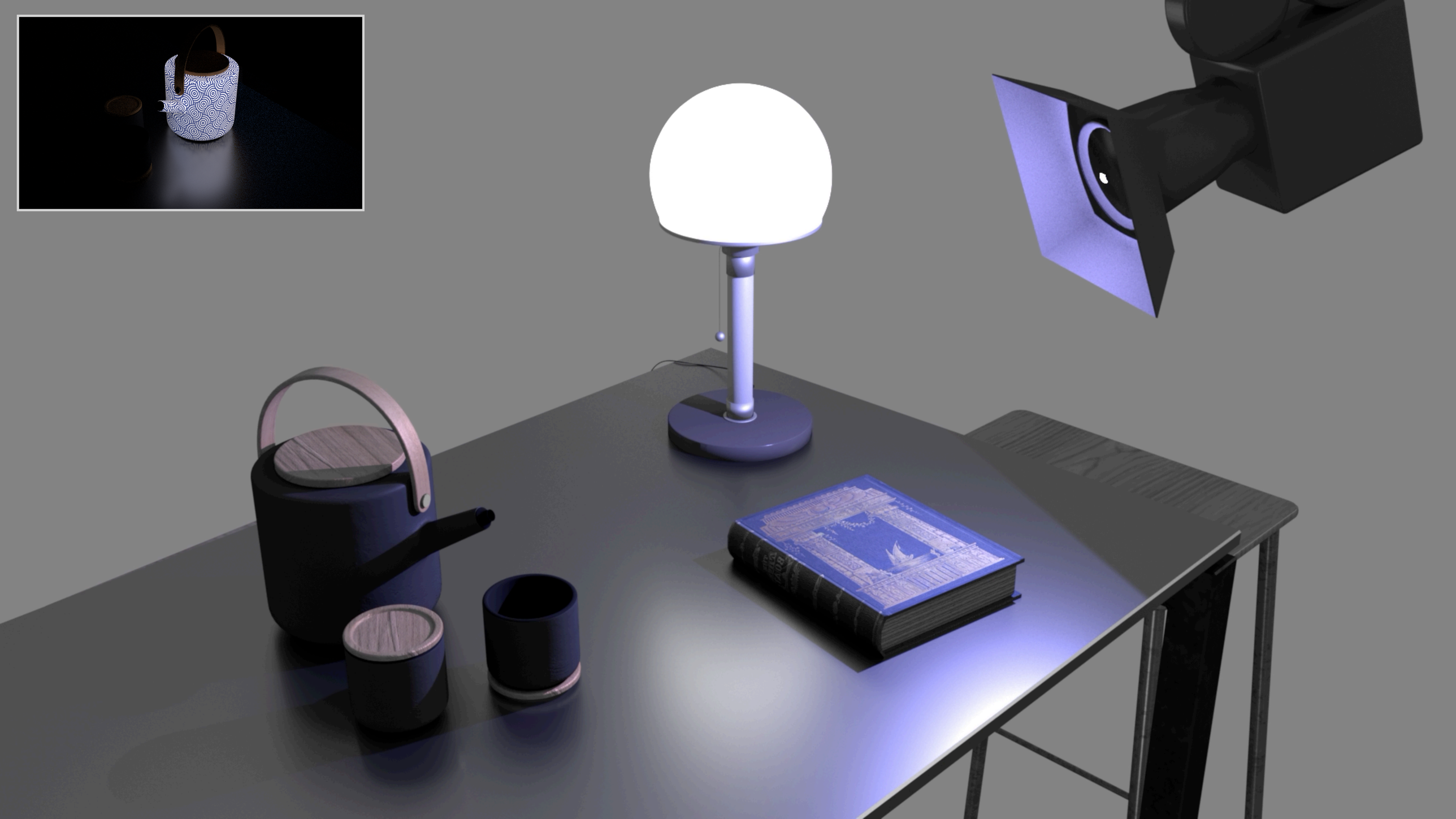


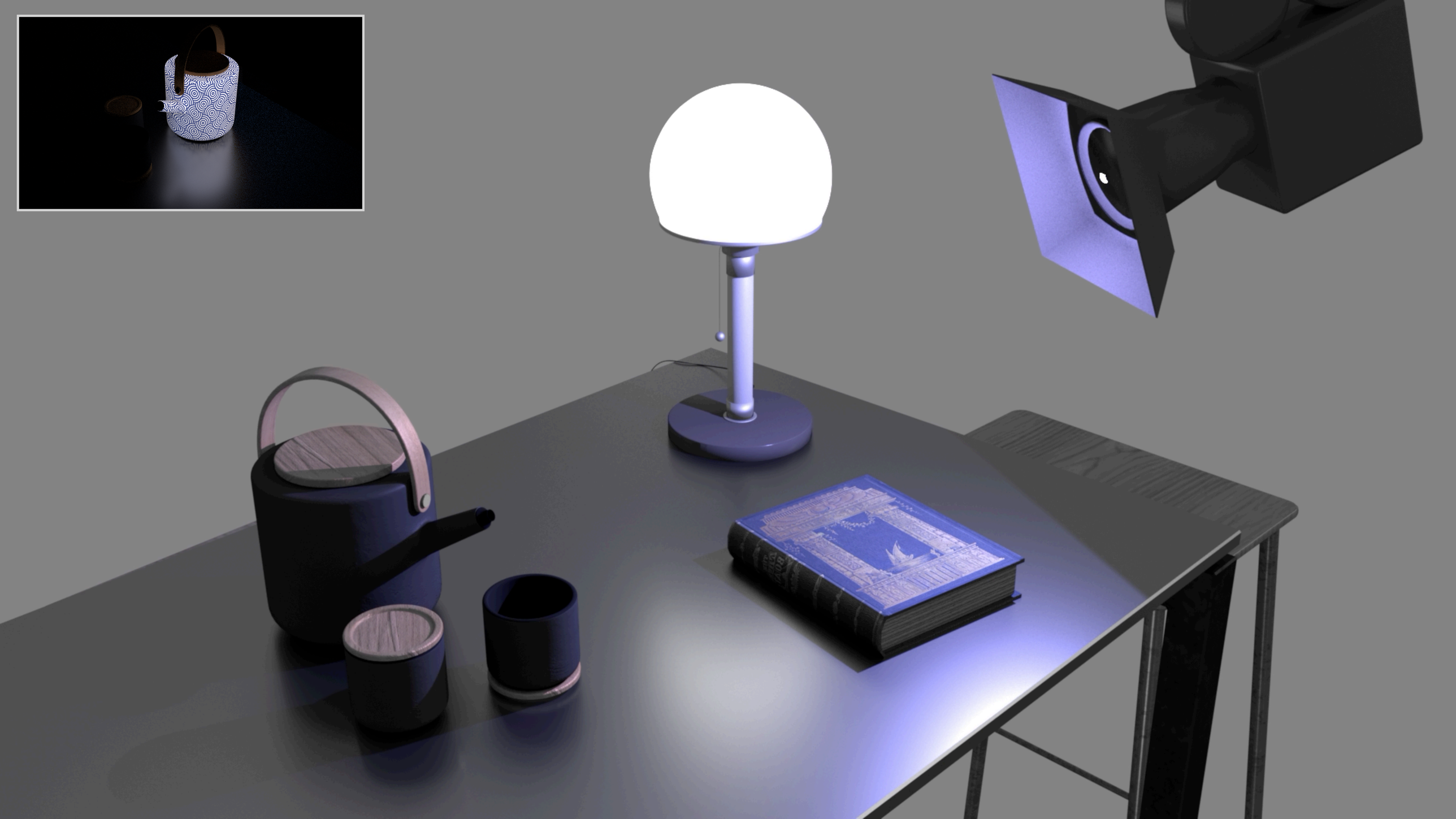
Bidirectional Estimators for Light Transport

Veach & Guibas, **1994**

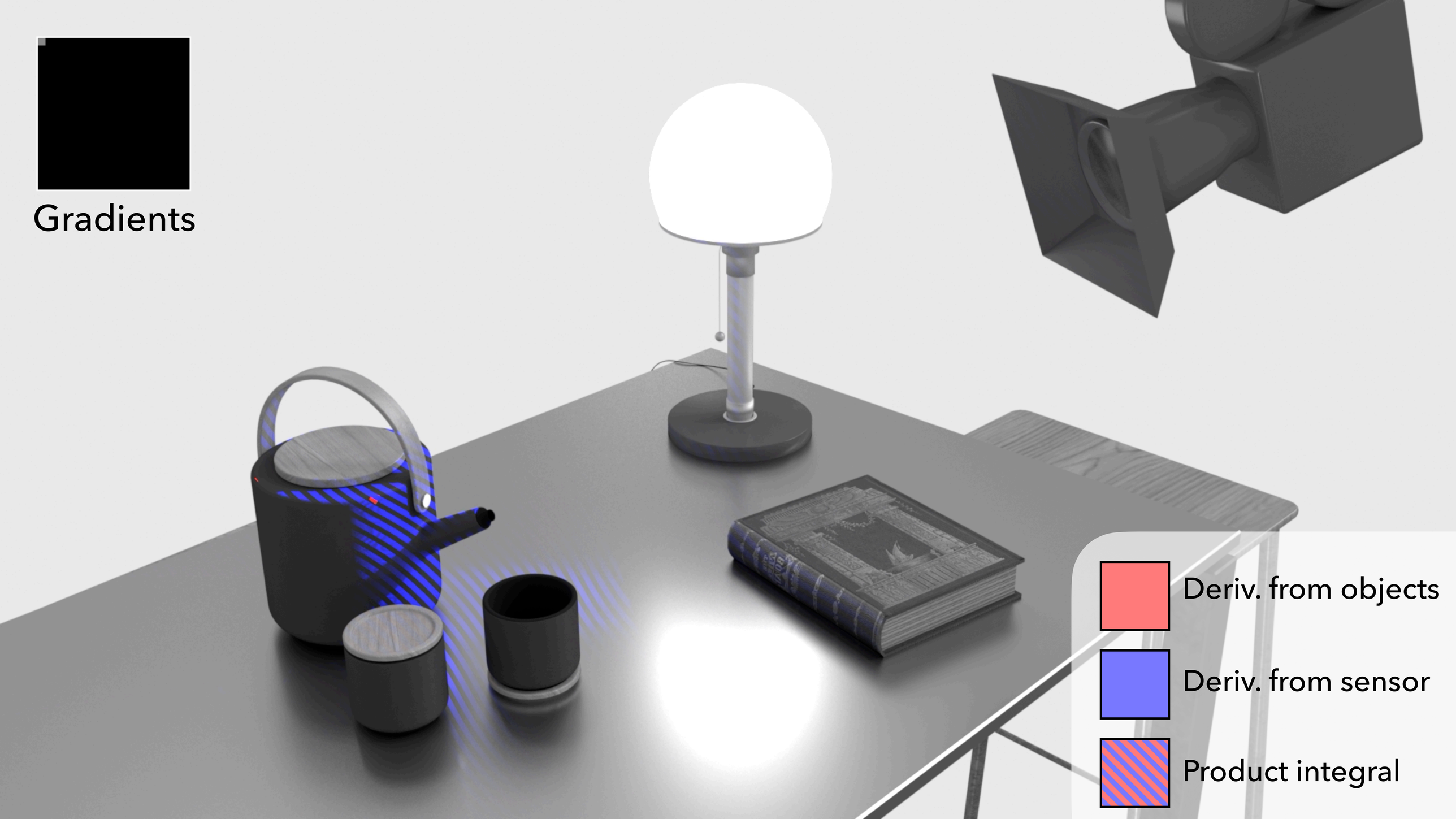
$$\langle O\mathbf{a}, \mathbf{b} \rangle = \langle \mathbf{a}, O\mathbf{b} \rangle$$

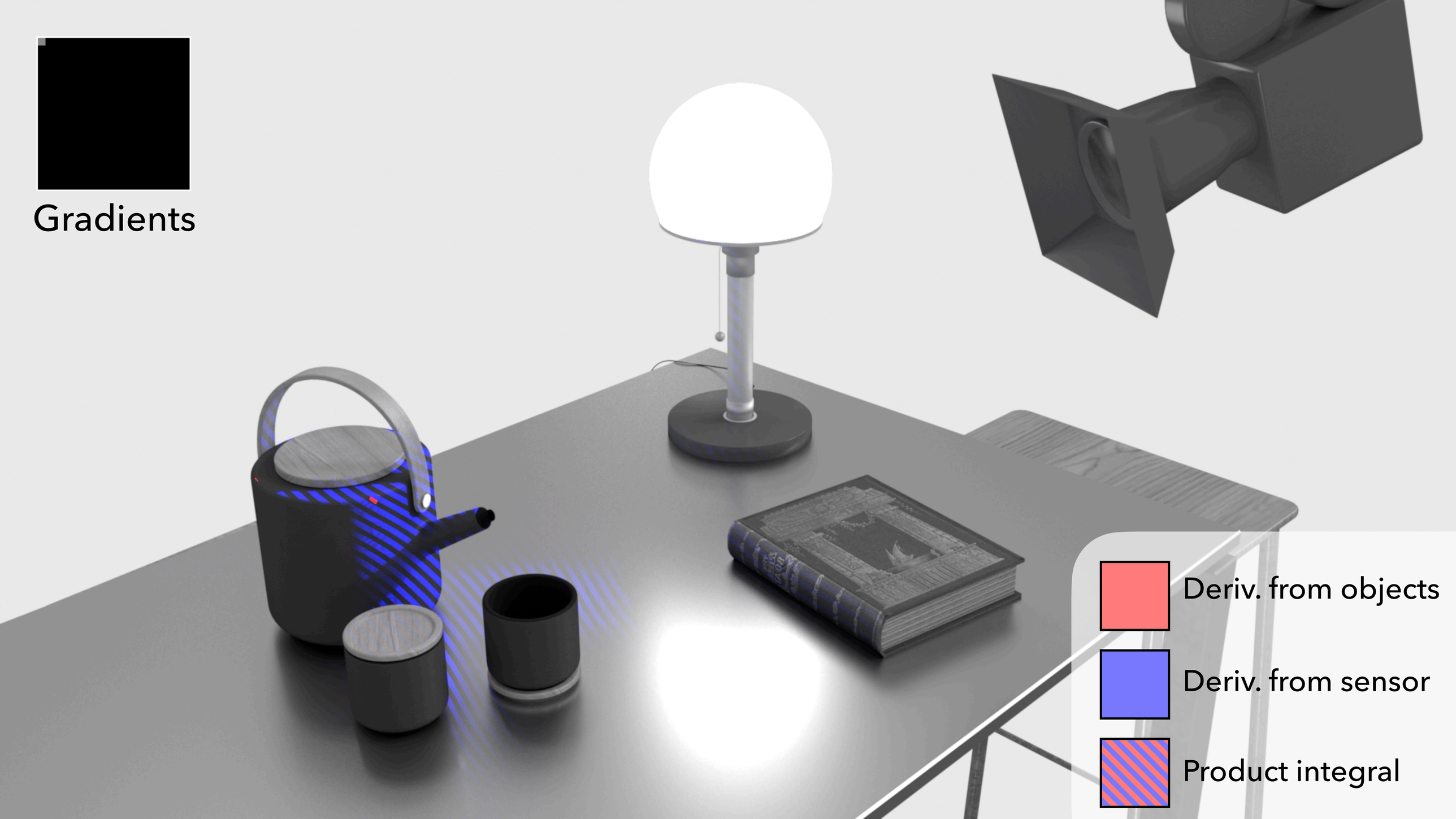
(Underlying principle: self-adjoint operators)

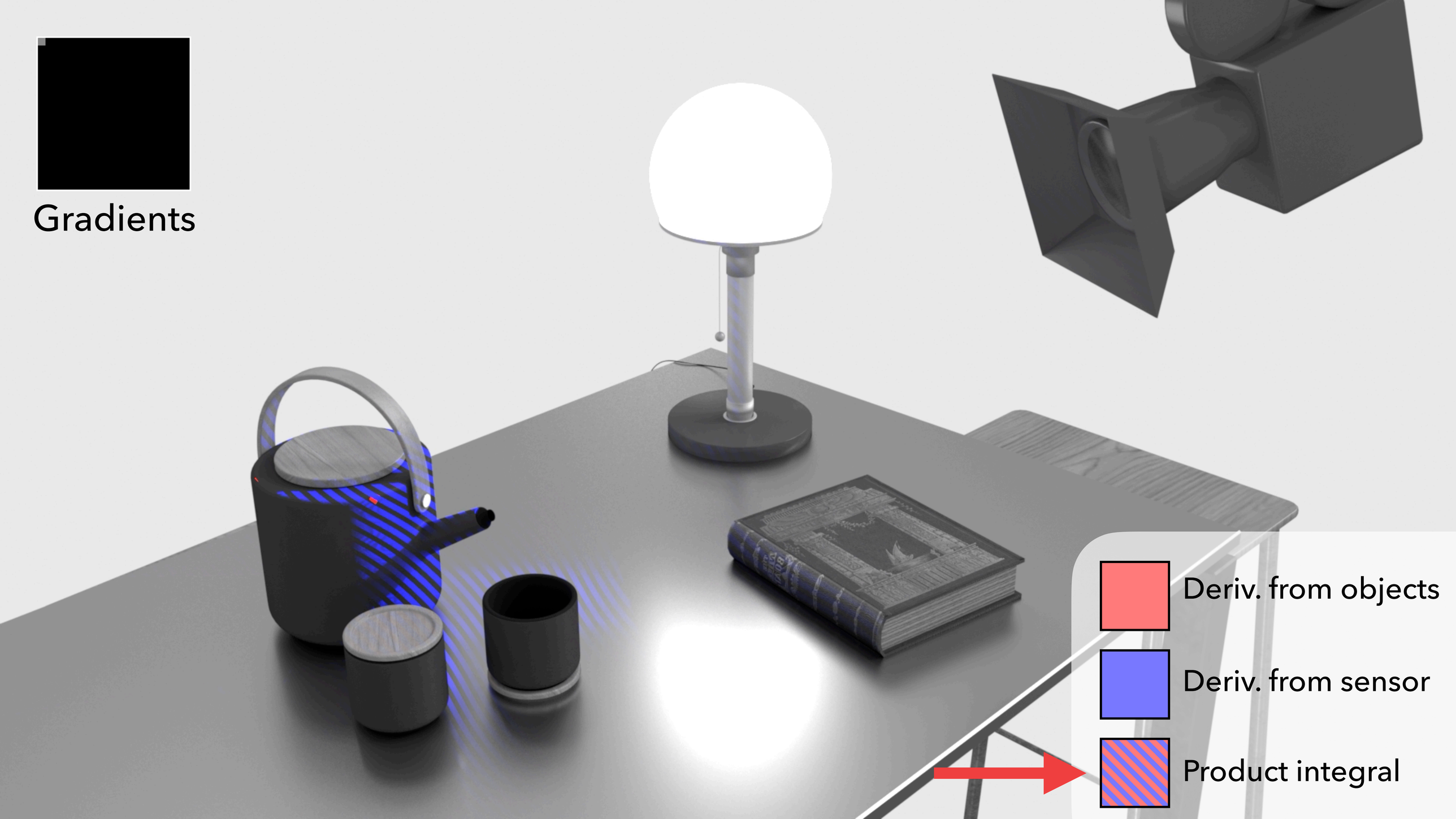


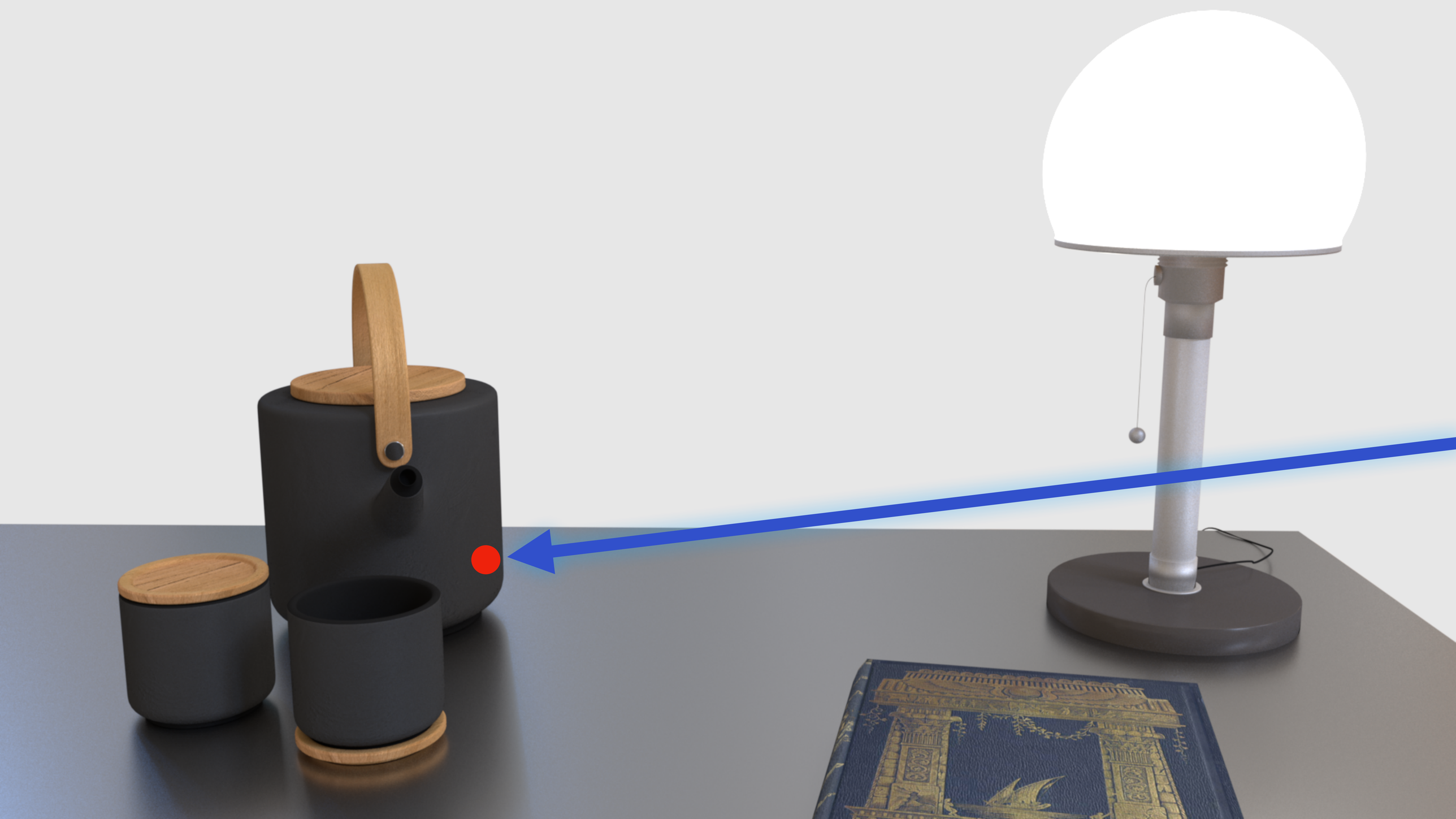


Derivatives projected into the scene

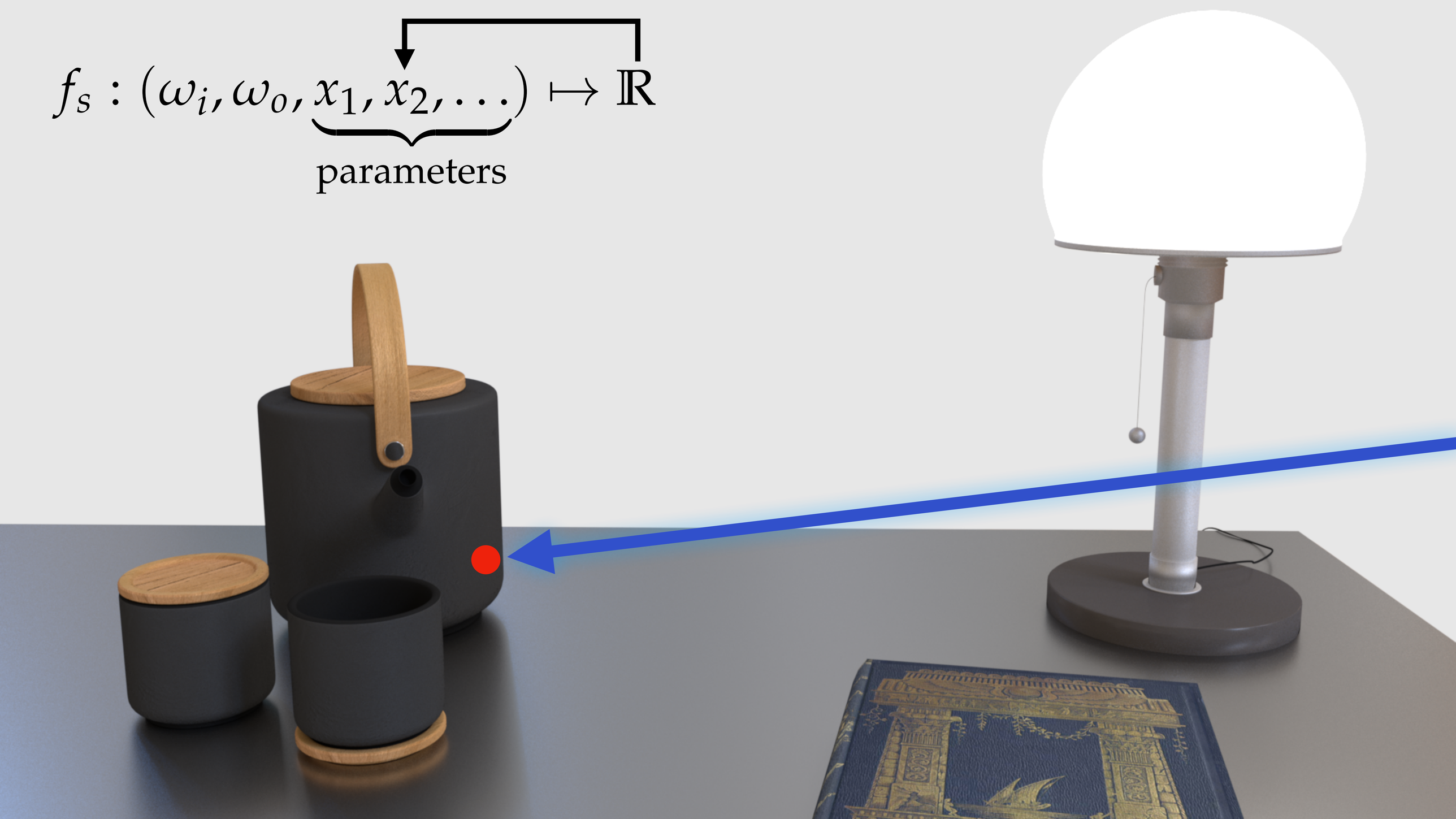

Gradients
Deriv. from objects
Deriv. from sensor
Product integral

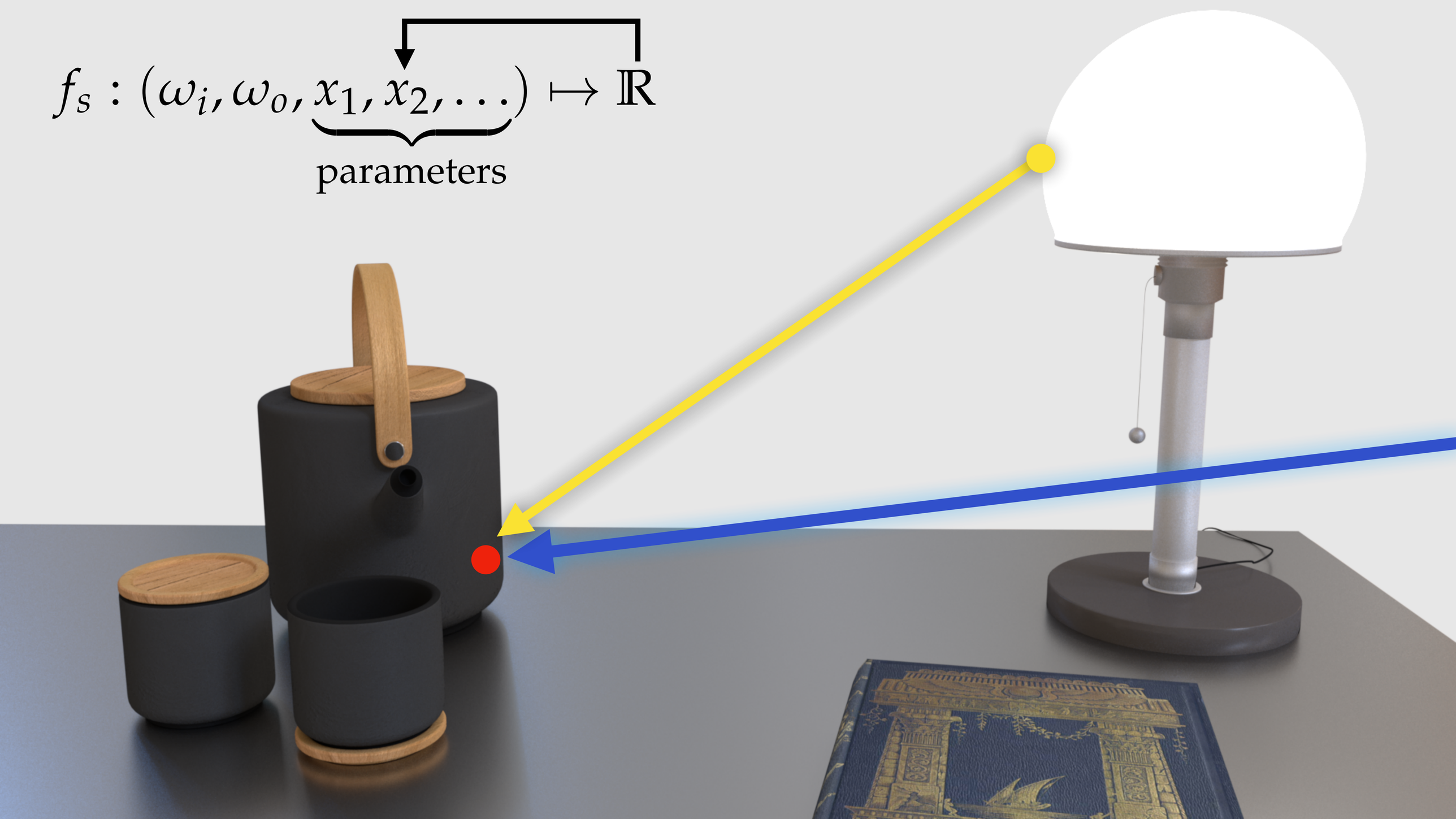

Gradients
Deriv. from objects
Deriv. from sensor
Product integral


Gradients
Deriv. from objects
Deriv. from sensor
Product integral



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




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

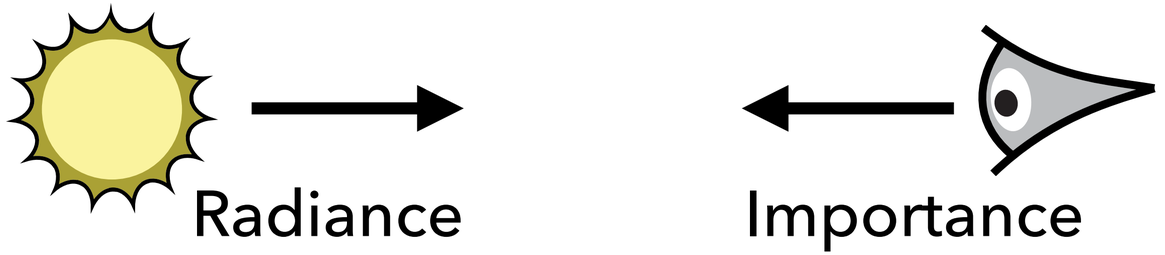
Another perspective

“Normal” rendering

Another perspective

“Normal” rendering

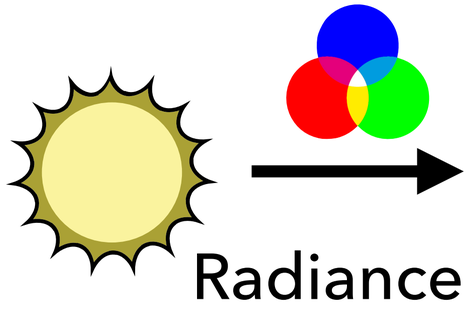
- Transporting from sensor/light may yield lower variance.

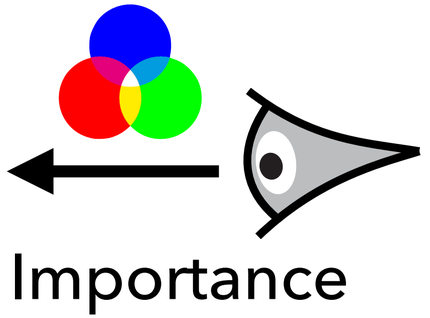


Another perspective

“Normal” rendering

- Transporting from sensor/light may yield lower variance.

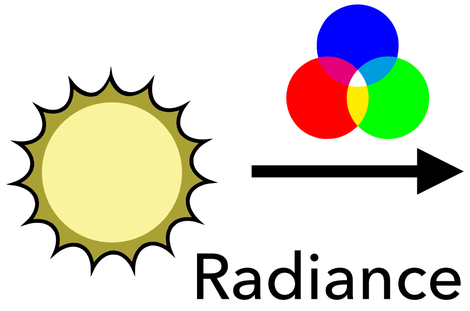




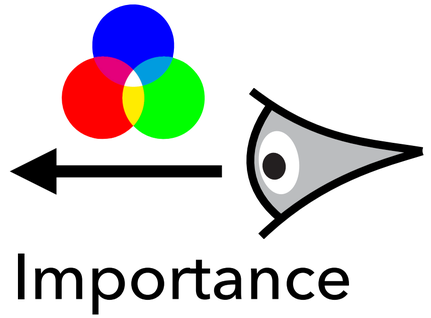
Another perspective

“Normal” rendering

- Transporting from sensor/light may yield lower variance.

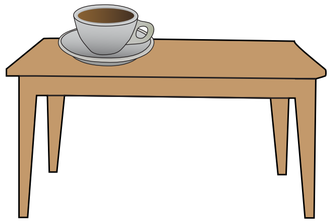


Radiance

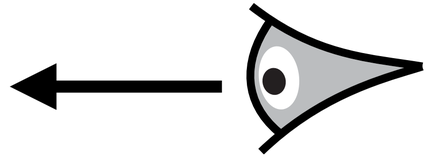


Importance

Differentiable rendering



∂ from objects

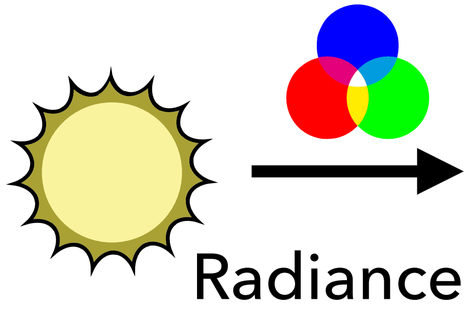


∂ from sensor

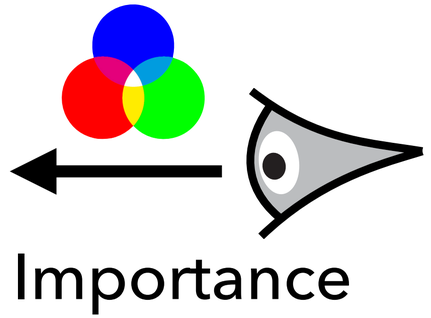
Another perspective

“Normal” rendering

- Transporting from sensor/light may yield lower variance.

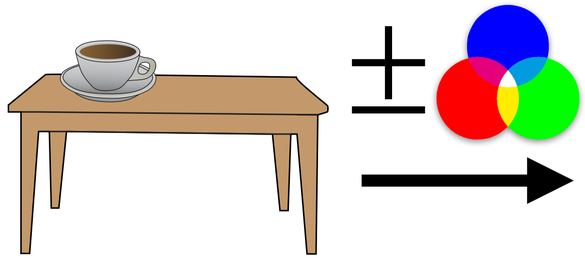


Radiance

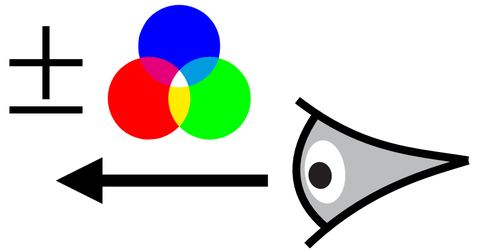


Importance

Differentiable rendering



∂ from objects

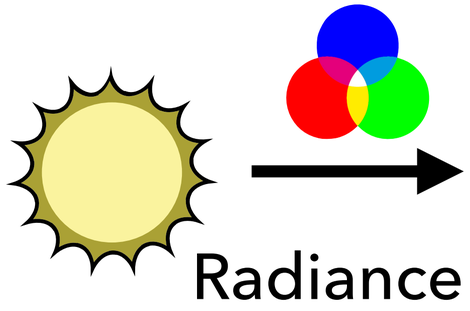


∂ from sensor

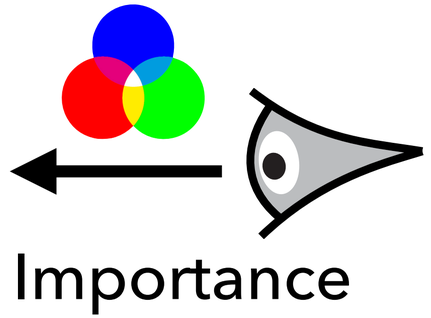
Another perspective

“Normal” rendering

- Transporting from sensor/light may yield lower variance.



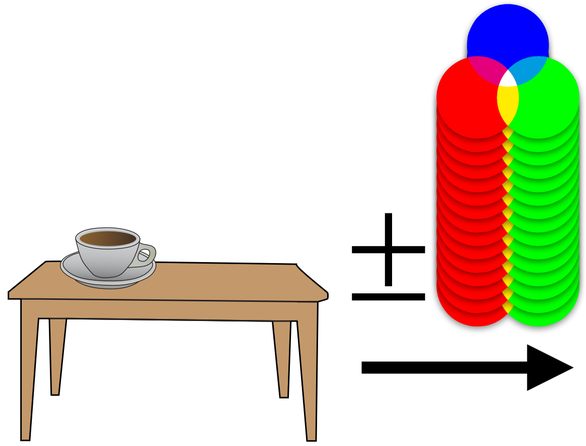
Radiance



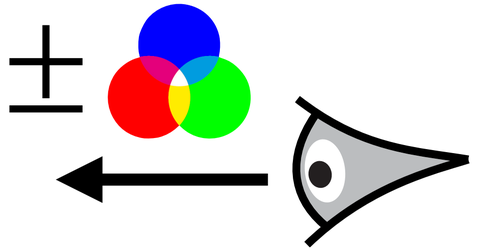
Importance

Differentiable rendering

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



∂ from objects



∂ from sensor

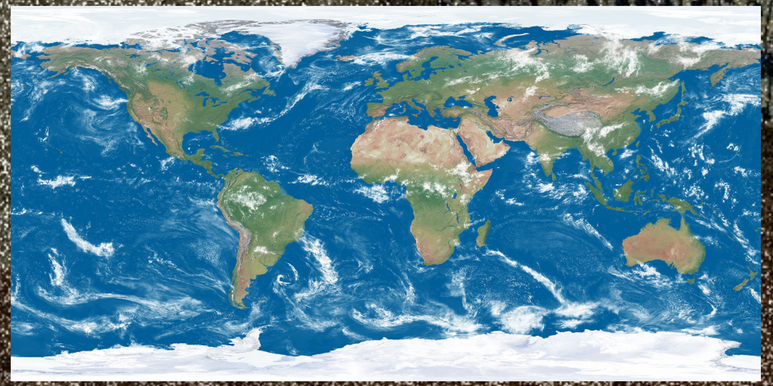
Surface BSDF optimization

Initial scene

Target (only given as image)

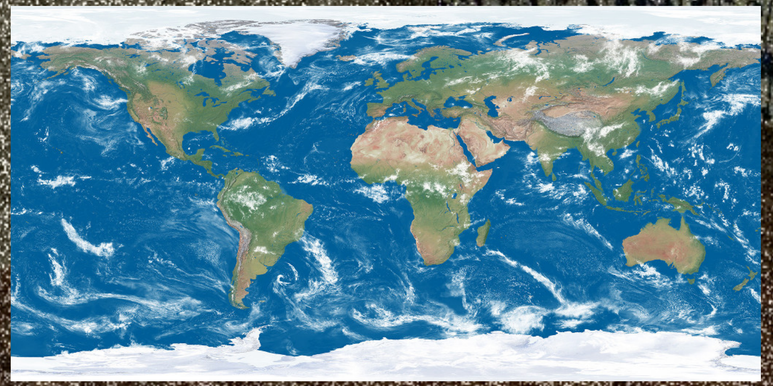
Optimized texture

Target

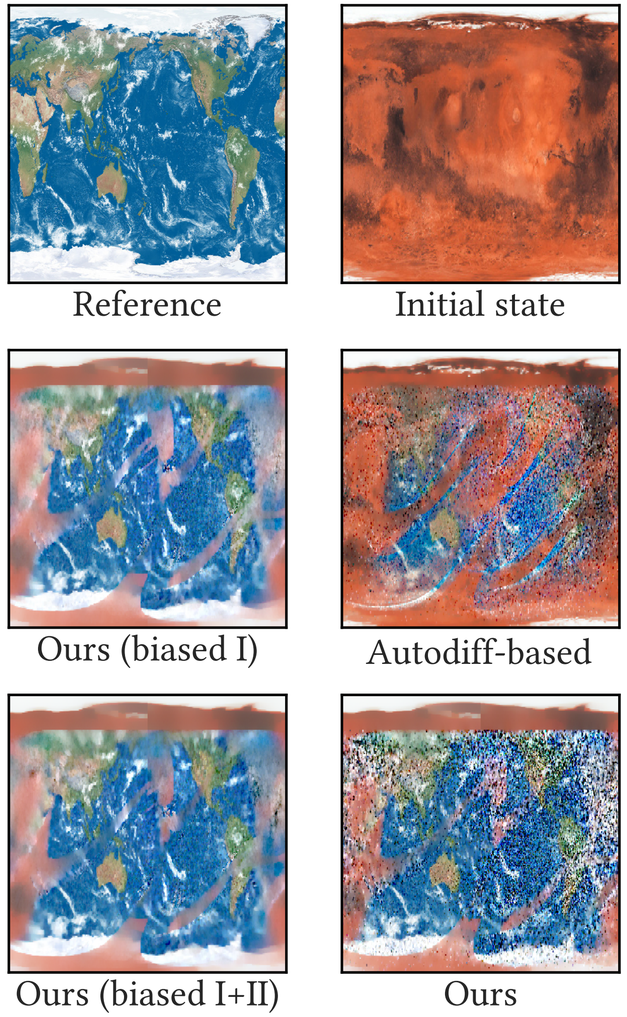


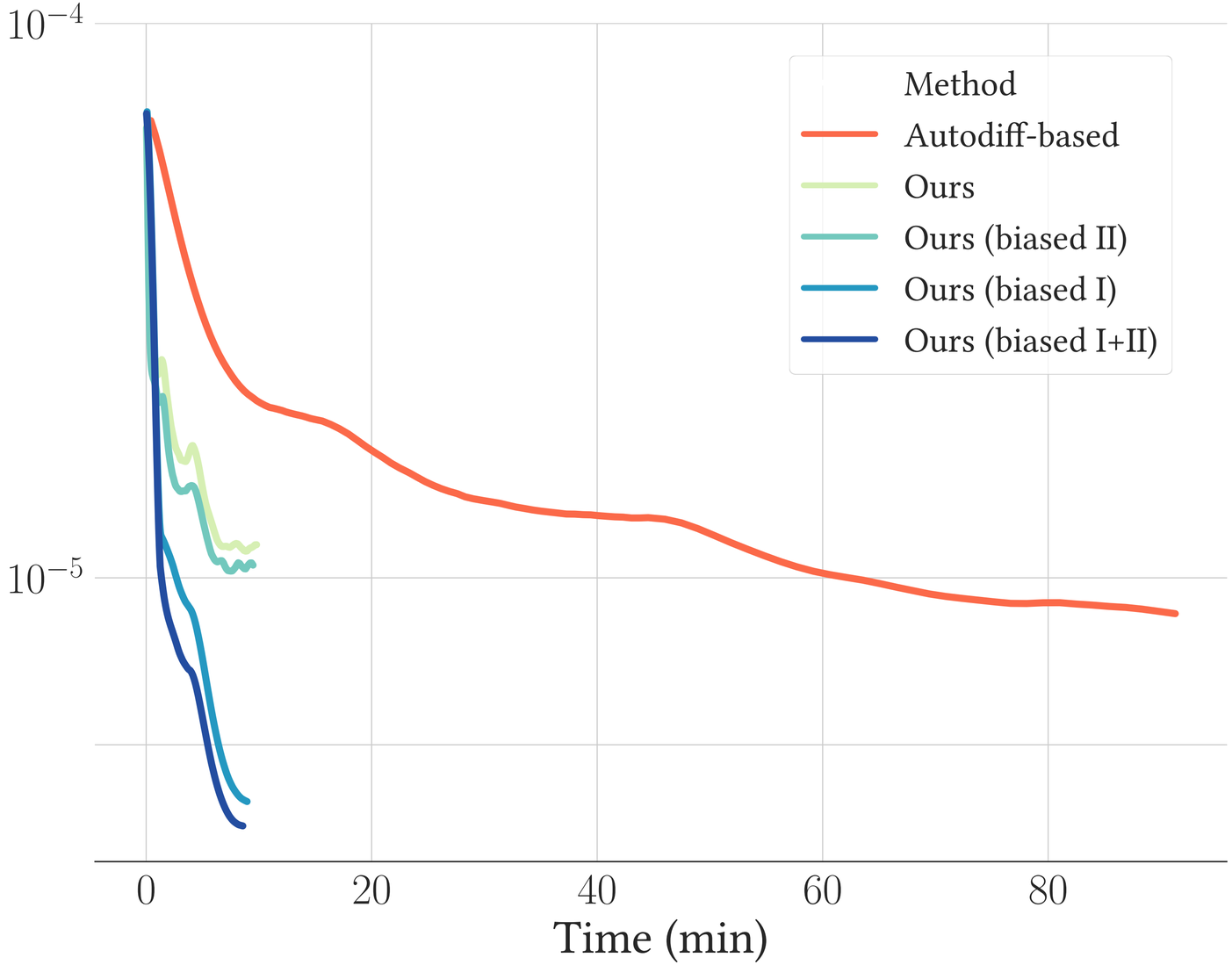
Optimized texture

Target

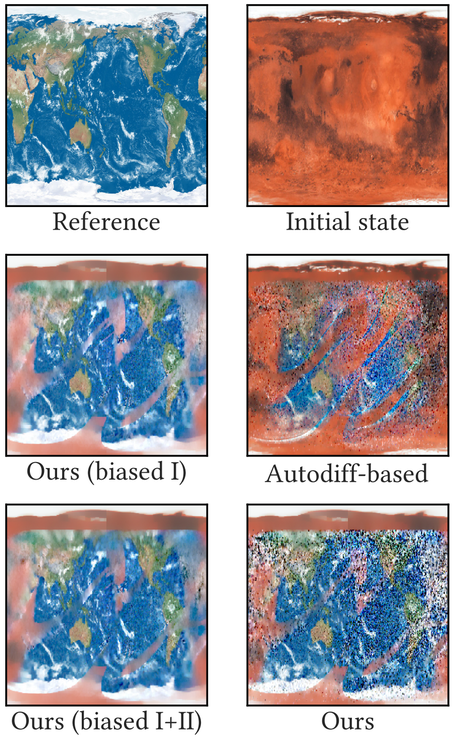


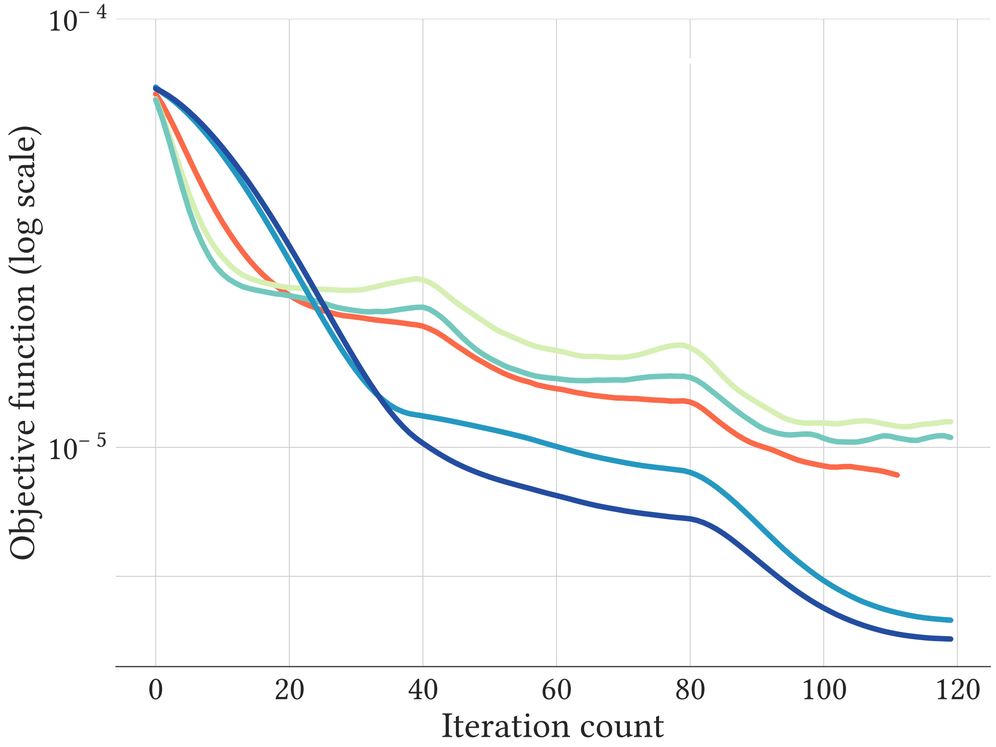
Surface BSDF optimization

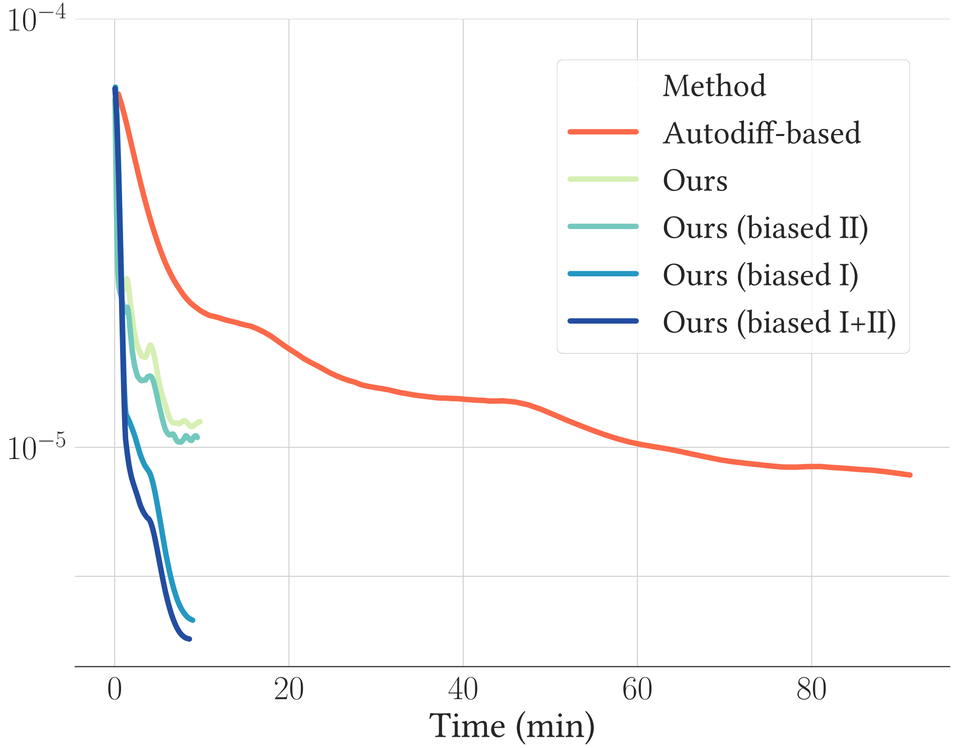




Surface BSDF optimization







Volume density optimization

Mitsuba 2 (AD-based)

Radiative Backprop.
(biased I + II)

Target

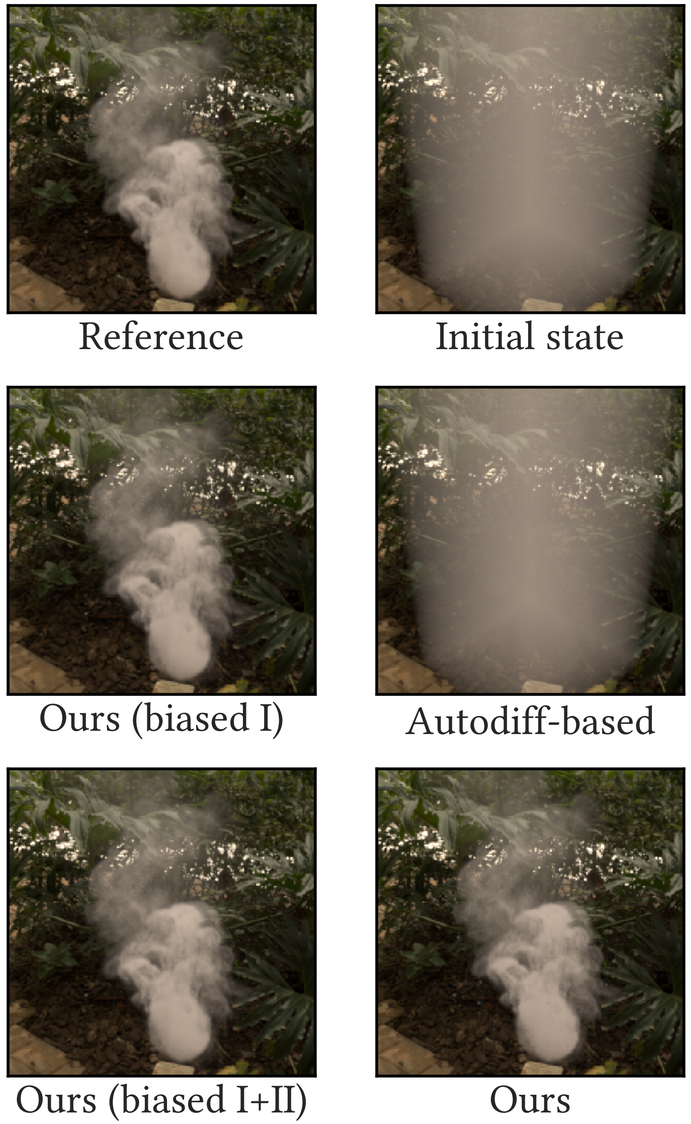
Volume density optimization

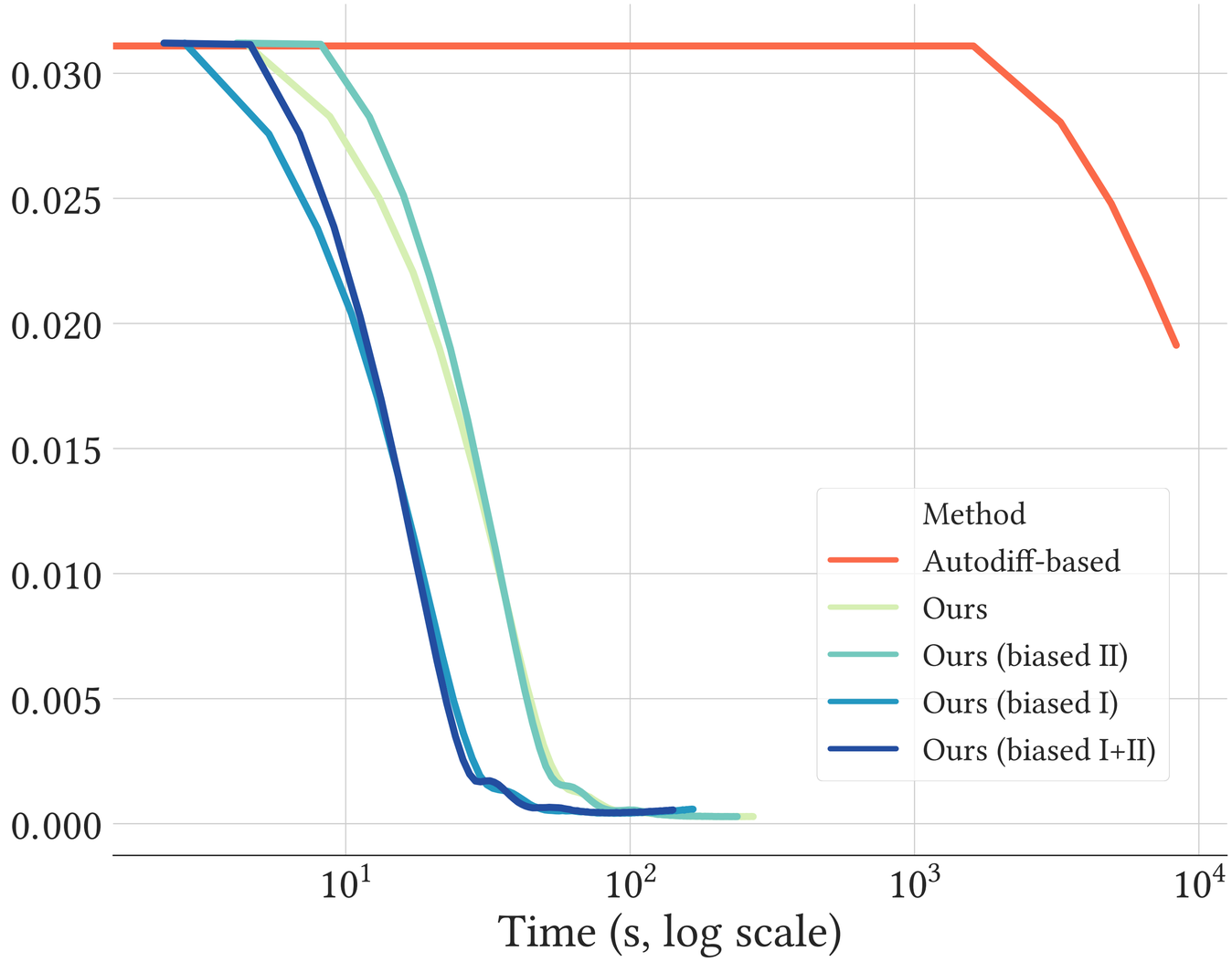
Mitsuba 2 (AD-based)

Radiative Backprop.
(biased I + II)

Target

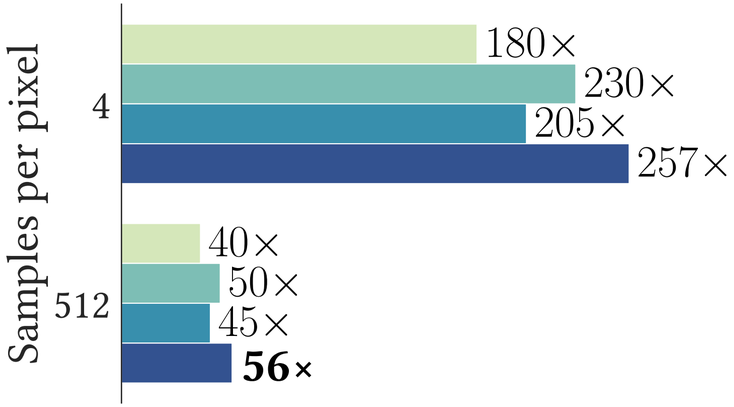
Volume density optimization



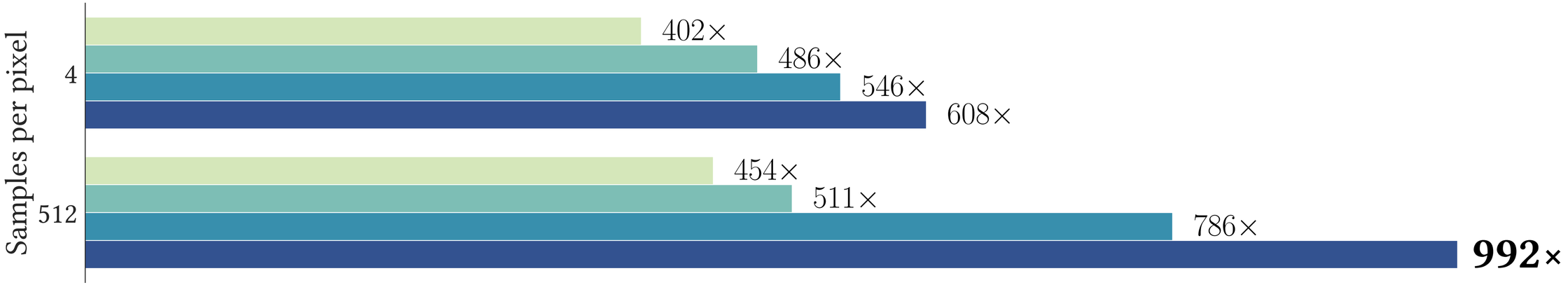


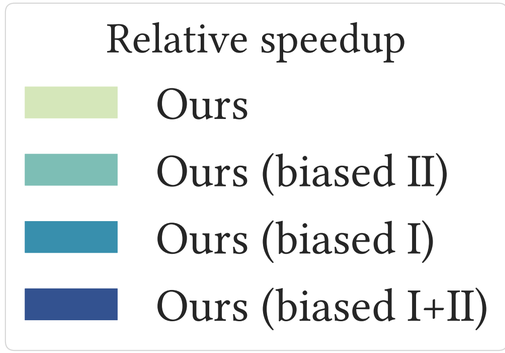
Relative speedups *vs* autodiff-based

Surface texture optimization



Volume density optimization

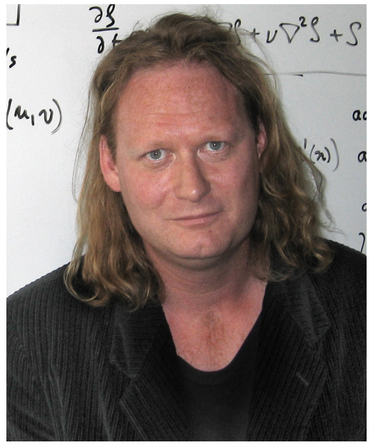




TL;DR

- 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!

ArXiv paper by Jos Stam



Computing Light Transport Gradients using the Adjoint Method

Jos Stam, Graphics Researcher, NVIDIA.

03/10/2019 - present

Abstract

This paper proposes a new equation from continuous adjoint theory to compute the gradient of quantities governed by the Transport Theory of light. Unlike discrete gradients ala ***autograd***[1], which work at the code level, we first formulate the continuous theory and then discretize it. The key insight of this paper is that computing gradients in Transport Theory is akin to computing the importance, a quantity adjoint to radiance that satisfies an adjoint equation. Importance tells us where to look for light that matters. This is one of the key insights of this paper. In fact, this mathematical journey started from a whimsical thought that these adjoints might be related. Computing gradients is therefore no more complicated than computing the importance field. This insight and the following paper hopefully will shed some light on this complicated problem and ease the implementations of gradient computations in existing path tracers.

1. Introduction

In this paper we present a general framework for computing gradients in the context of light propagation. Gradients are of central importance in the fields of machine learning, computer vision and computer graphics. Often, we need to invert a simulation like a rendering to recover *hidden* control parameters. For smooth problems the gradient is a key instrument in methods such as gradient descent or quasi-Newton iteration. This paper is concerned with computing the gradient of a solution to a transport equation. In order to achieve this goal, we derive a continuous adjoint equation for the gradient of the radiance. This equation is a generalization of the usual ***backpropagation*** algorithm popular in deep learning. We show that the adjoint equation for the gradient is almost identical to the adjoint equation for the ***importance*** in transport theory. The only difference is a different source term that is equal to the initial gradient of the cost function with respect to the radiance field.

The method of computation is akin to a bi-directional Monte Carlo solution using radiance and importance. First the transport equation is solved for the radiance ***forward*** from the light sources to the receivers (camera/eye). Then the adjoint transport equation is solved ***backwards*** from the receiver for the adjoint of the gradient of radiance similarly to the importance. As the propagation progresses backwards, we update the gradients of the cost function with respect to the controls acting at that point in the path. The reader familiar with backpropagation in deep learning will appreciate the analogy with

[1] Autograd is just one of many packages out there that computes differentials at the code level. See https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html for more details and [2] for an excellent introduction to Automatic Differentiation.

Computing Light Transport Gradients using the Adjoint Method

arXiv:2006.15059

Agenda for today

Inverse rendering

Differentiable rendering

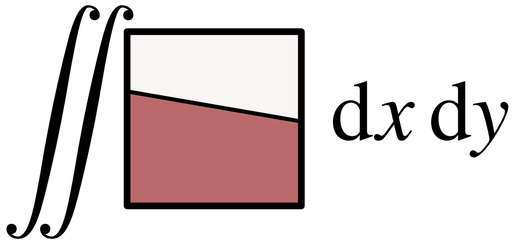
Example problem

Challenges

1. How to do this at all?
2. Efficiency
3. **Discontinuities**
4. Robustness

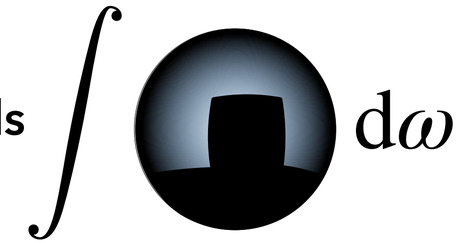
Differentiating Monte Carlo Estimates

Pixel integrals



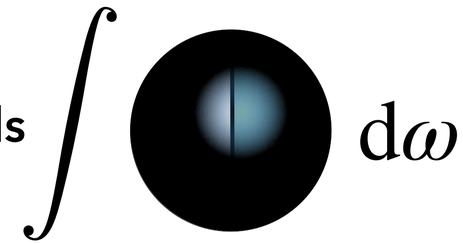
$$\mathrm{d}x\,\mathrm{d}y$$

Light integrals

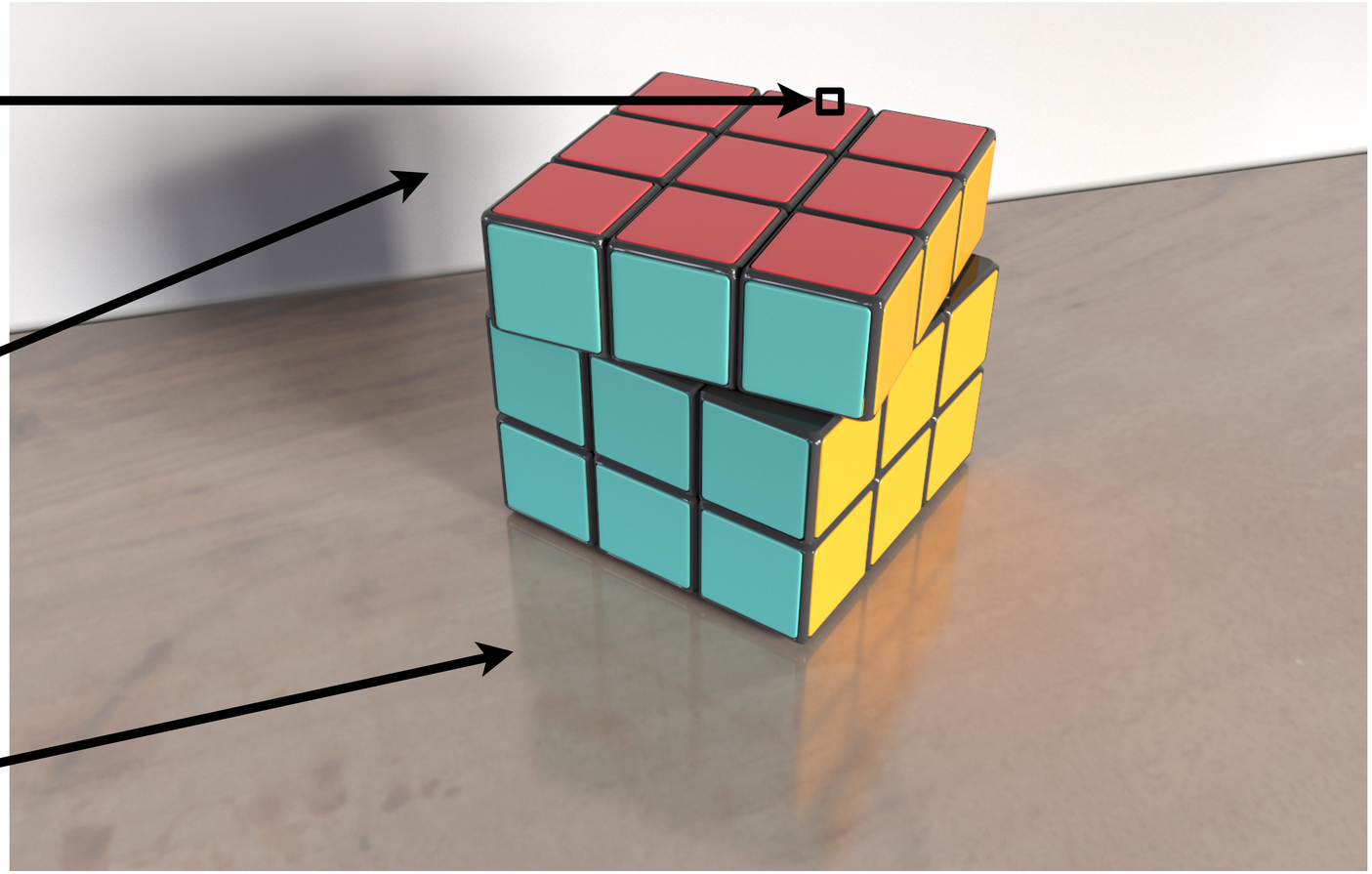


$$\mathrm{d}\omega$$

BSDF integrals



$$\mathrm{d}\omega$$

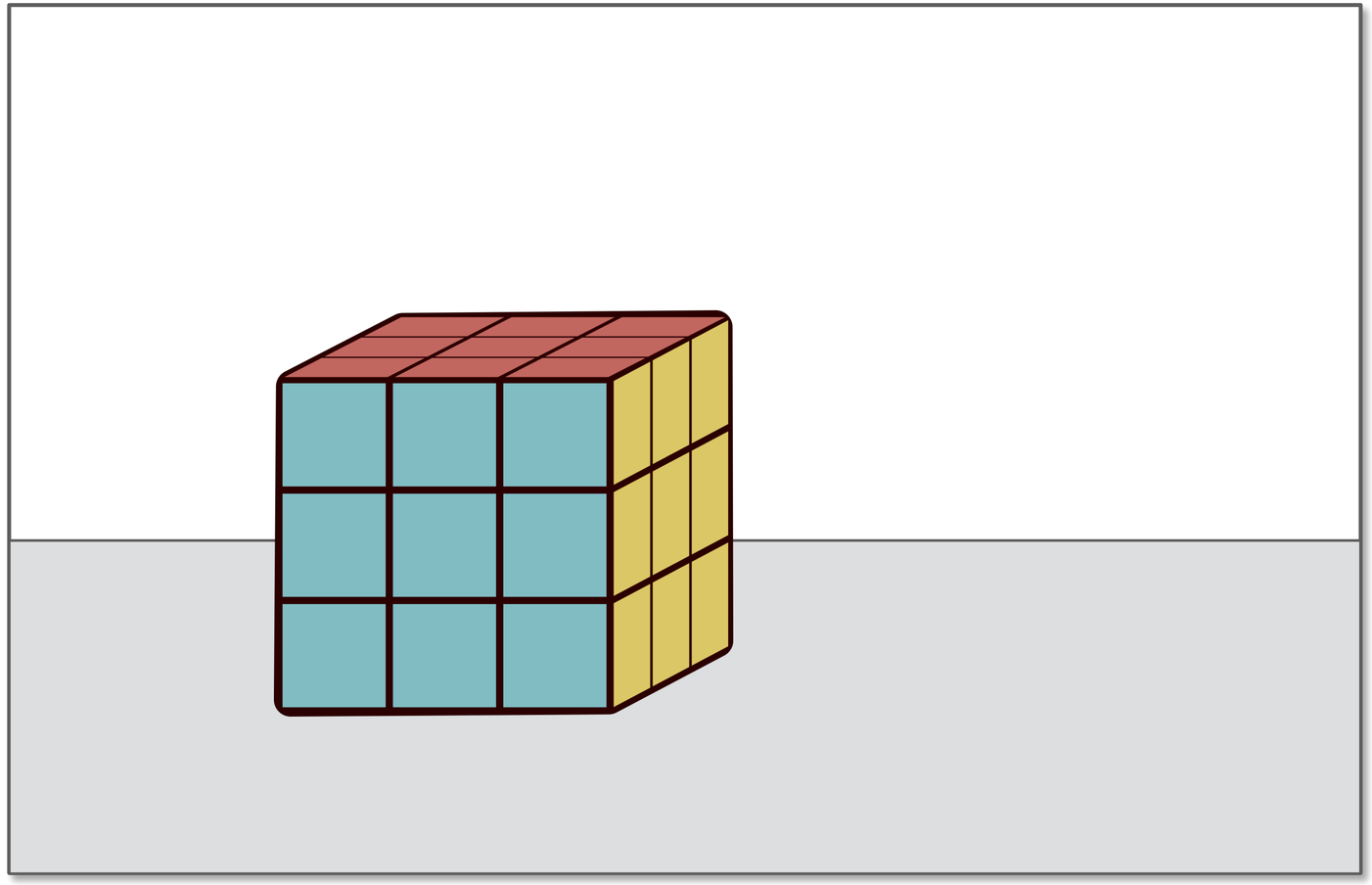


Differentiating Monte Carlo Estimates

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

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

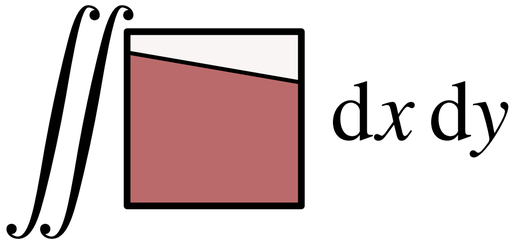
BSDF integrals $\int \mathrm{d}\omega$



Scene parameter x_i

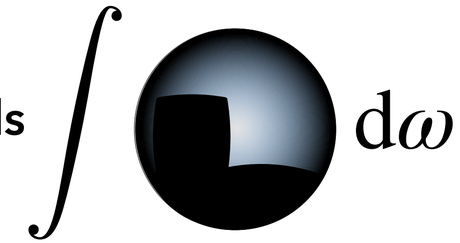
Differentiating Monte Carlo Estimates

Pixel integrals



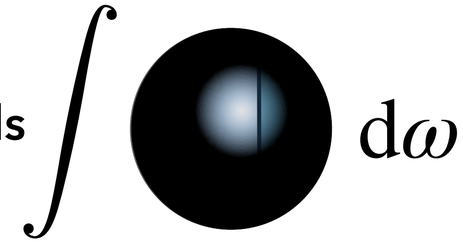
$$\mathrm{d}x\,\mathrm{d}y$$

Light integrals

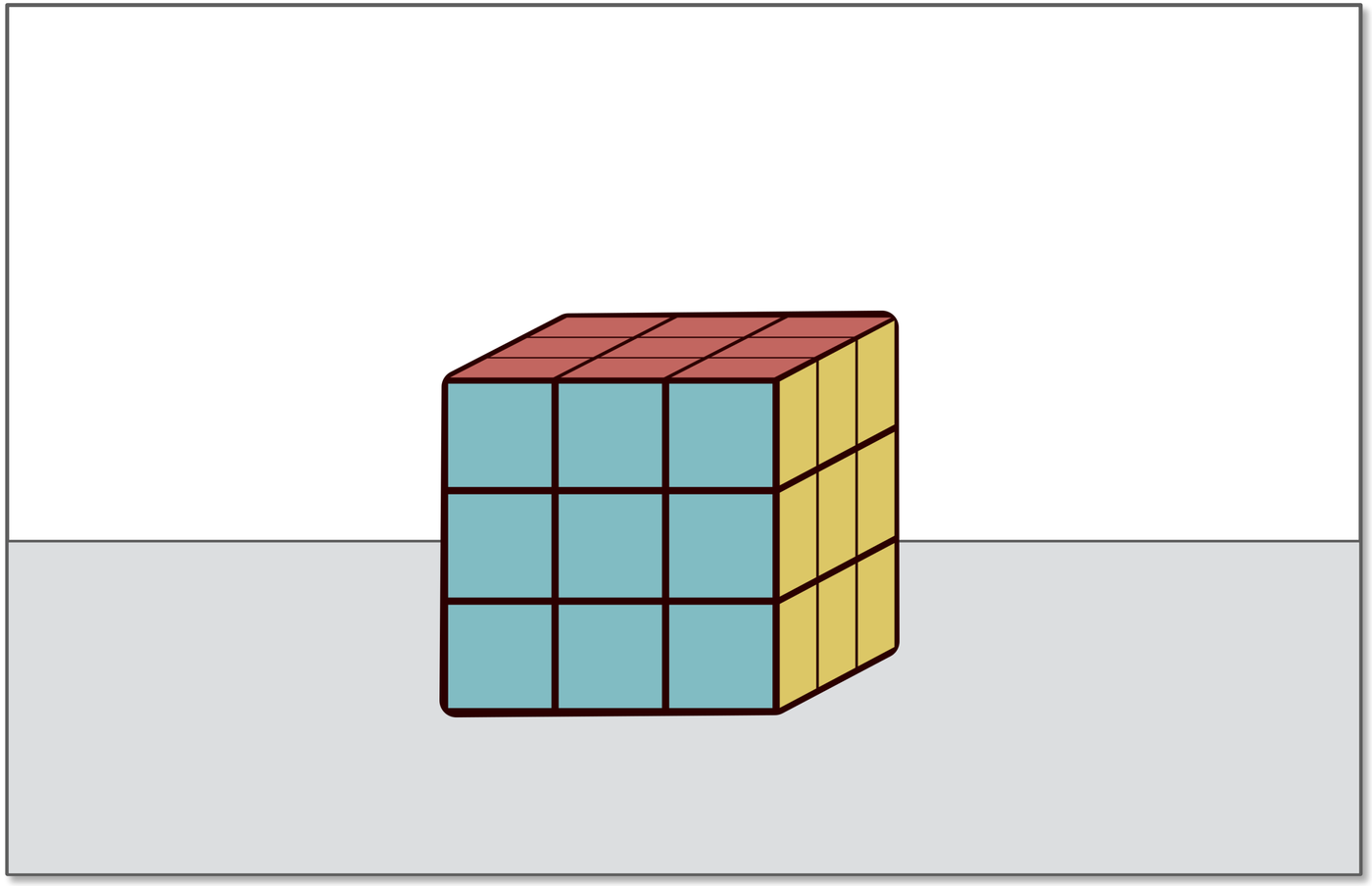


$$\mathrm{d}\omega$$

BSDF integrals



$$\mathrm{d}\omega$$



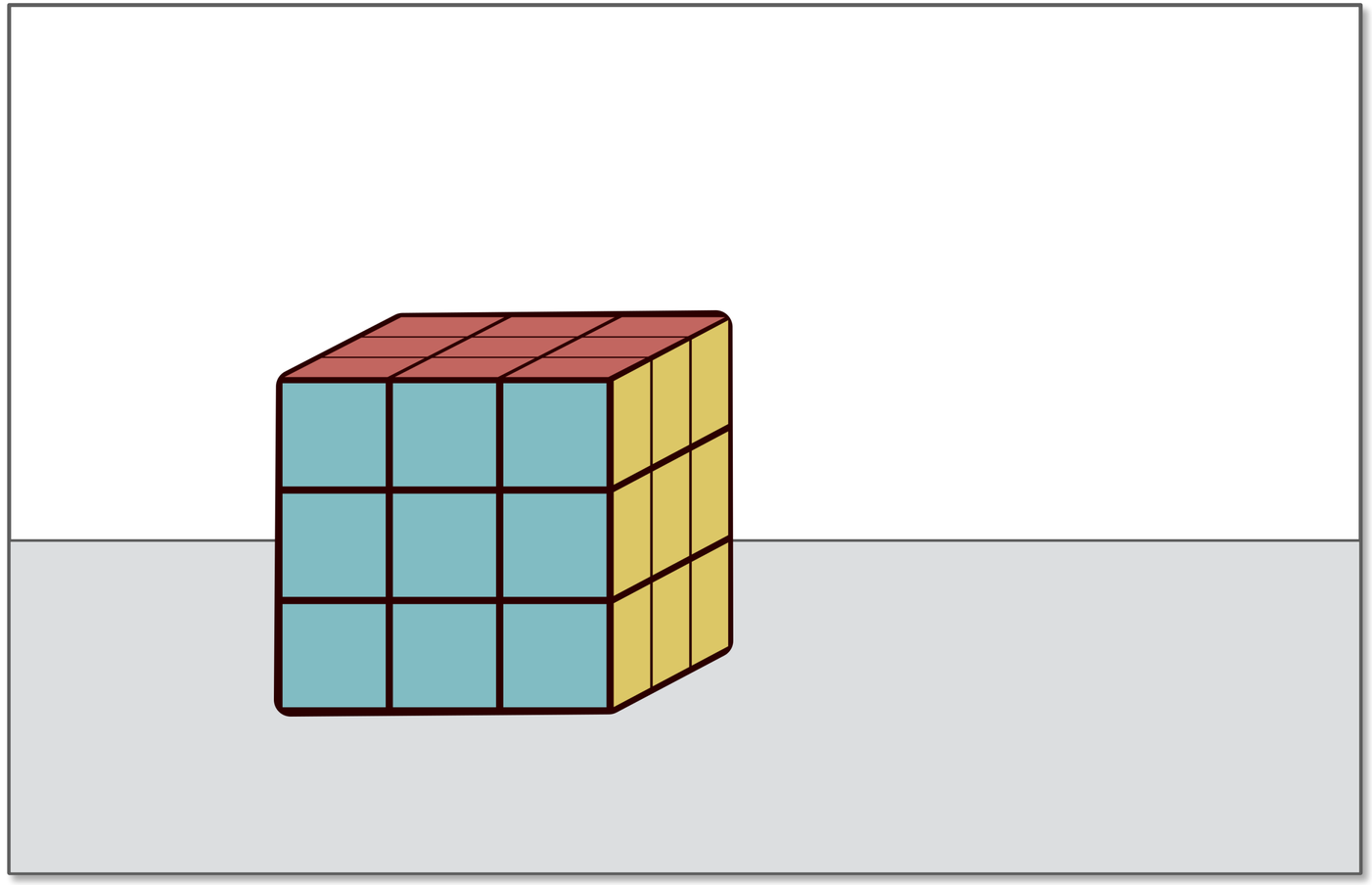
Scene parameter x_i

Differentiating Monte Carlo Estimates

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

The problem

$$\frac{\partial}{\partial x} \int f(x,y)\,\mathrm{d}y = \int \frac{\partial}{\partial x} f(x,y)\,\mathrm{d}y$$

The problem

$$\frac{\partial}{\partial x} \int f(x,y)\,\mathrm{d}y = \int \frac{\partial}{\partial x} f(x,y)\,\mathrm{d}y$$

The problem

$$\frac{\partial}{\partial x} \int f(x,y)\,\mathrm{d}y \neq \int \frac{\partial}{\partial x} f(x,y)\,\mathrm{d}y$$

The problem

$$\frac{\partial}{\partial x} \int f(x,y)\,\mathrm{d}y \neq \int \frac{\partial}{\partial x} f(x,y)\,\mathrm{d}y$$

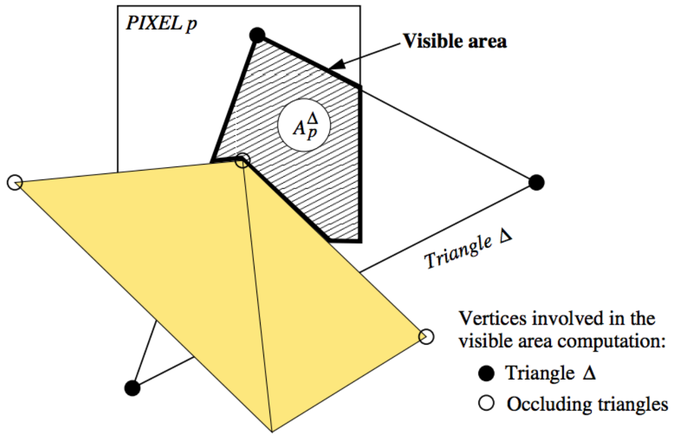
What we want.

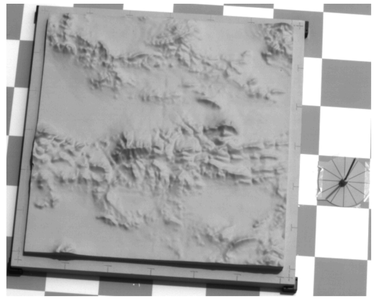
Differentiated simulation.

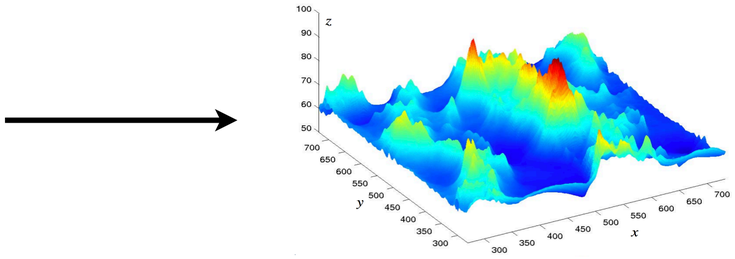
Discontinuities in differentiable renderers

- **Analytic pixel occupancy (NASA)**

[Smelyansky et al. 2002] [Jalobeanu et al. 2004]





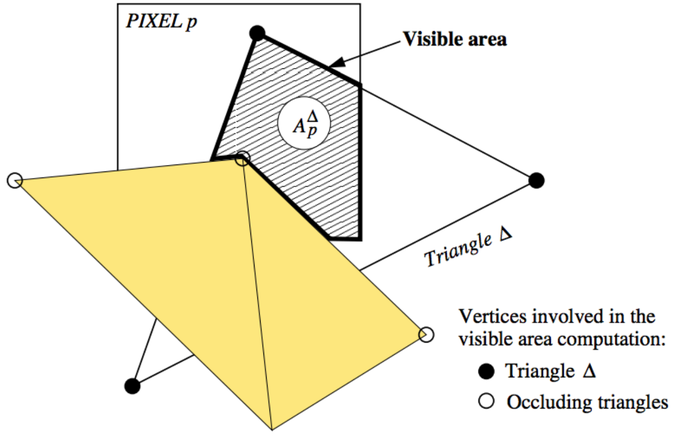


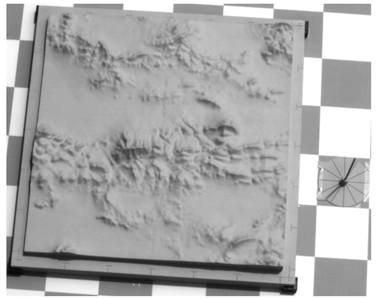
Discontinuities in differentiable renderers

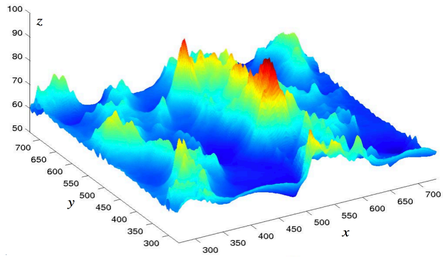
- **Analytic pixel occupancy (NASA)**

[Smelyansky et al. 2002]

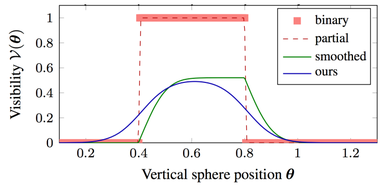
[Jalobeanu et al. 2004]



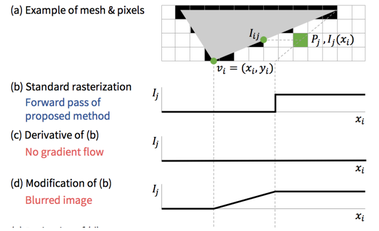




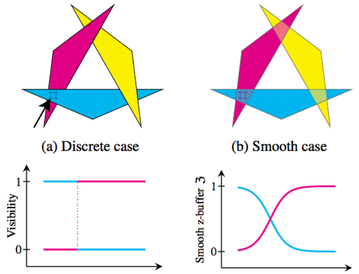
Approximations using blur (direct visibility)



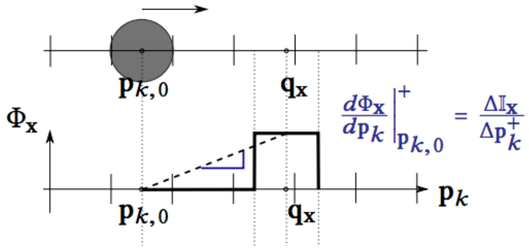
[Rhodin et al. 2015]



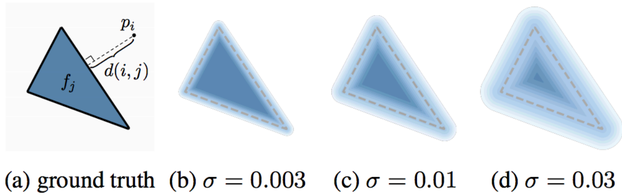
[Kato et al. 2018]



[Petersen et al. 2019]

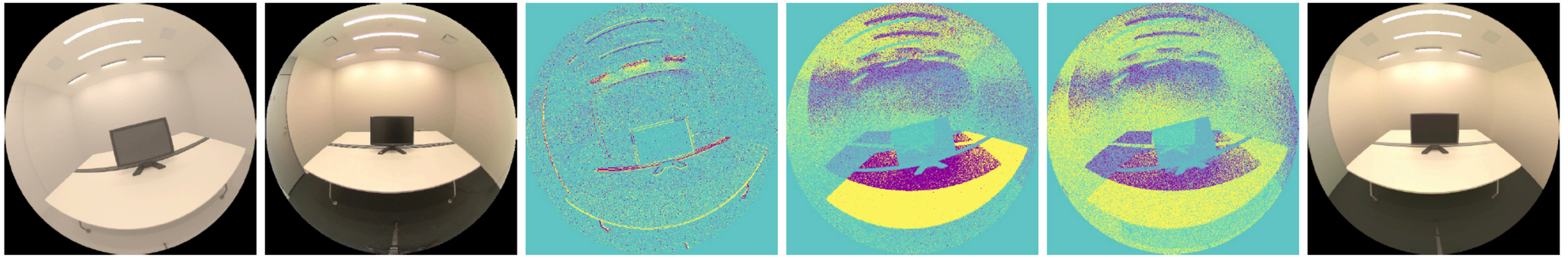


[Yifan et al. 2019]

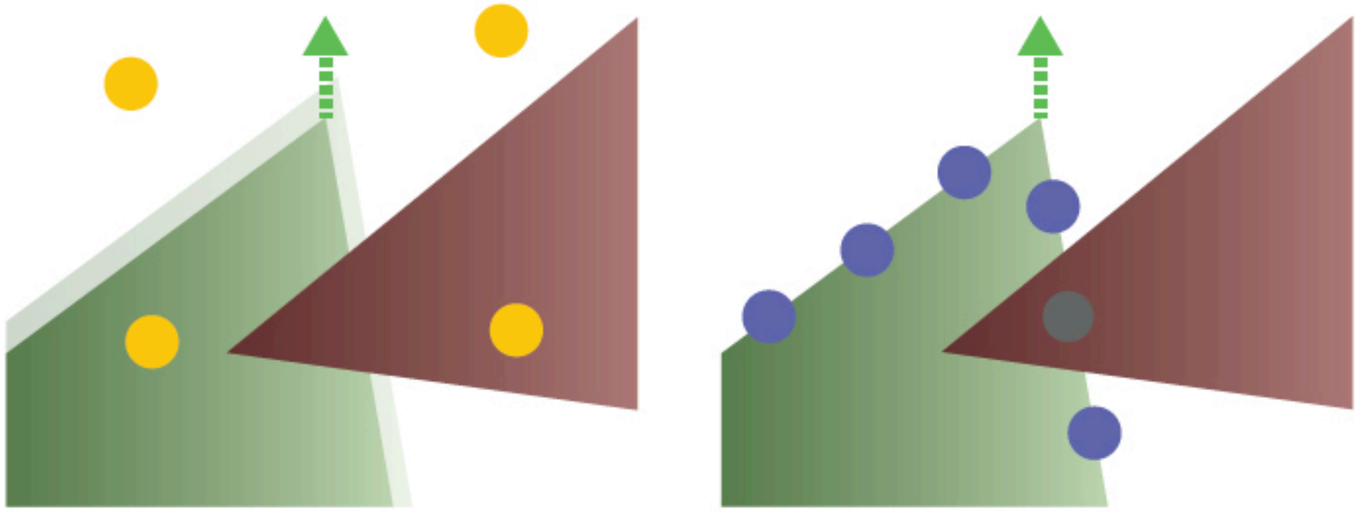


[Liu et al. 2019]

Differentiable Monte Carlo Ray Tracing Through Edge Sampling



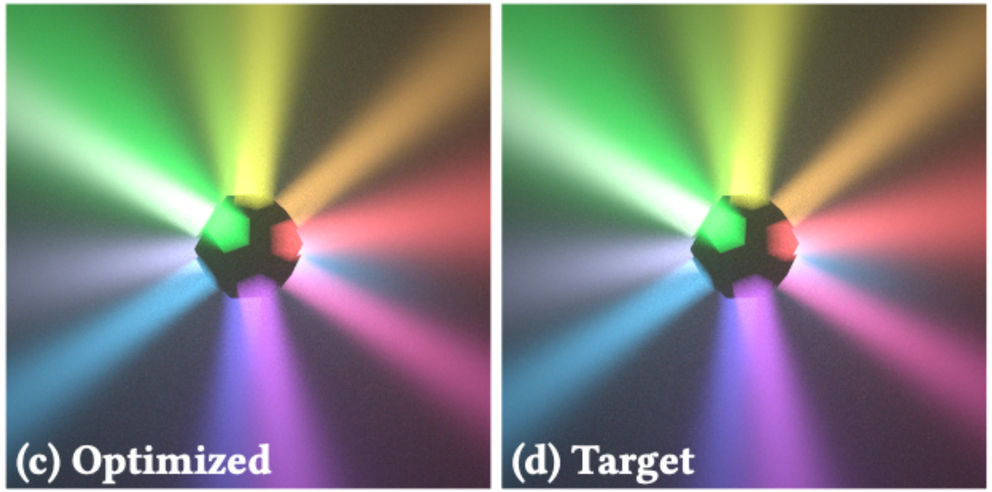
(a) initial guess (b) real photograph (c) camera gradient (per-pixel contribution) (d) table albedo gradient (per-pixel contribution) (e) light gradient (per-pixel contribution) (f) our fitted result



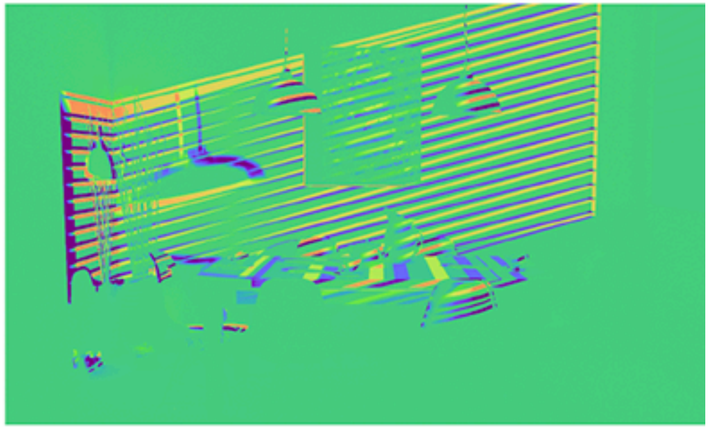
Tzu-Mao Li, Miika Aittala,
Frédo Durand, Jaakko Lehtinen

SIGGRAPH Asia 2018

Differentiable Monte Carlo Ray Tracing Through Edge Sampling




Original


Derivative with respect to sun location

A Differential Theory of Radiative Transfer

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

SIGGRAPH 2019

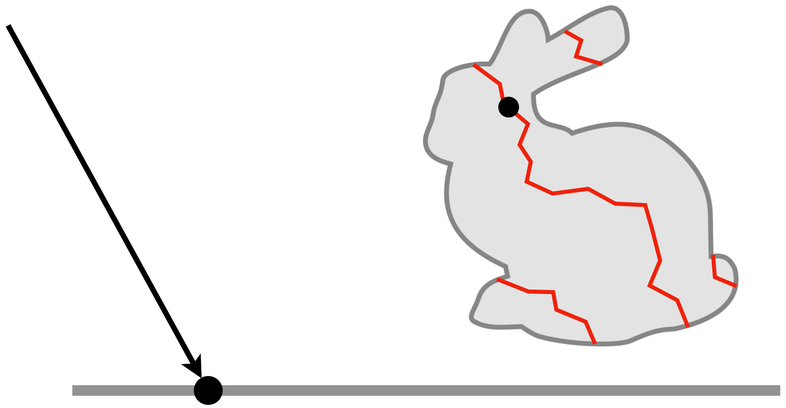
Path-Space Differentiable Rendering

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

SIGGRAPH 2020

Edge sampling

Edge sampling

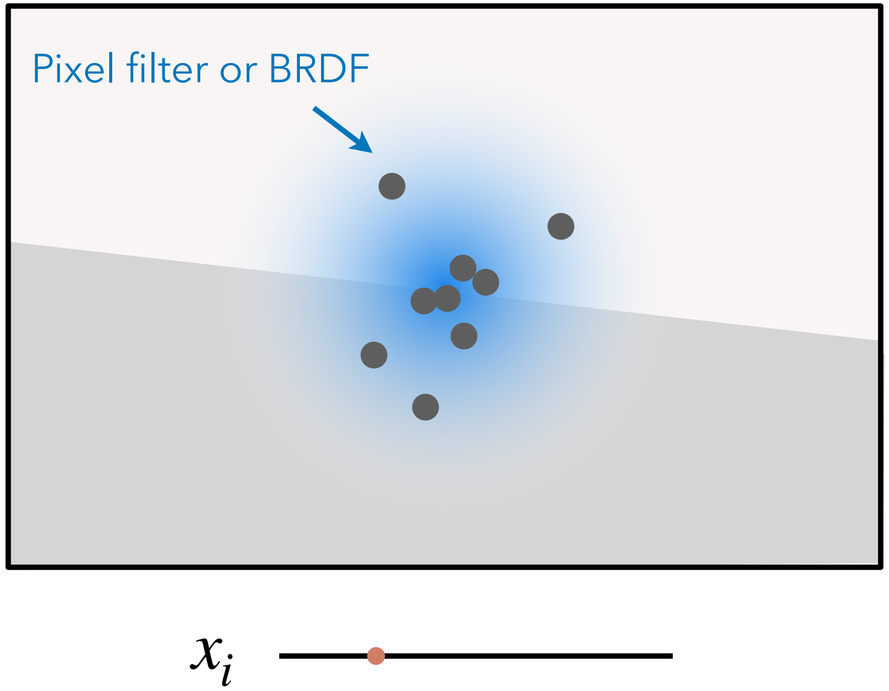


Edge sampling

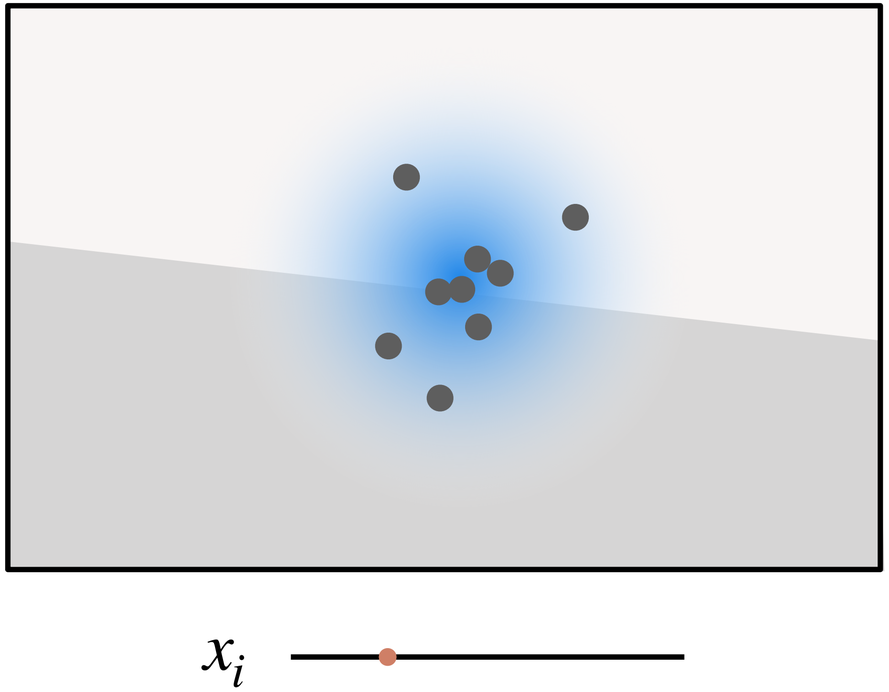
We currently don’t have good acceleration data structures for this operation.

Our approach: reparameterizing integrals

Non-differentiable Monte Carlo estimates

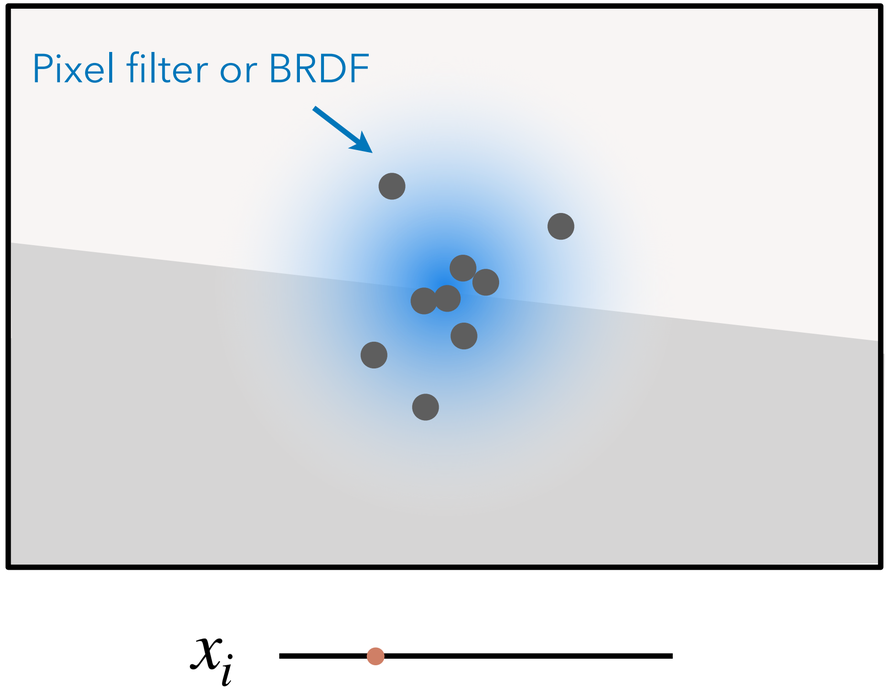


Differentiable Monte Carlo estimates

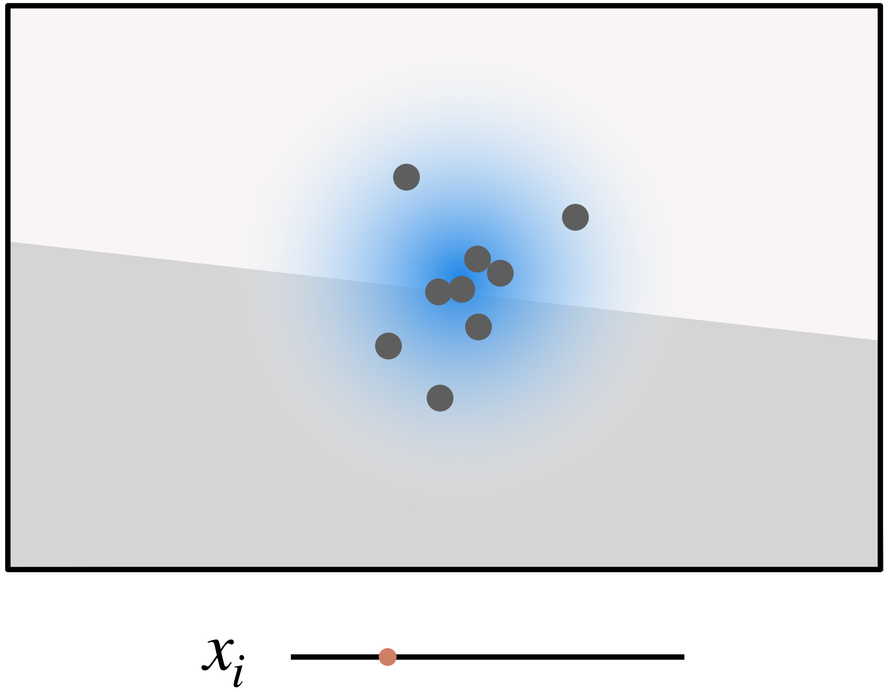


Our approach: reparameterizing integrals

Non-differentiable Monte Carlo estimates

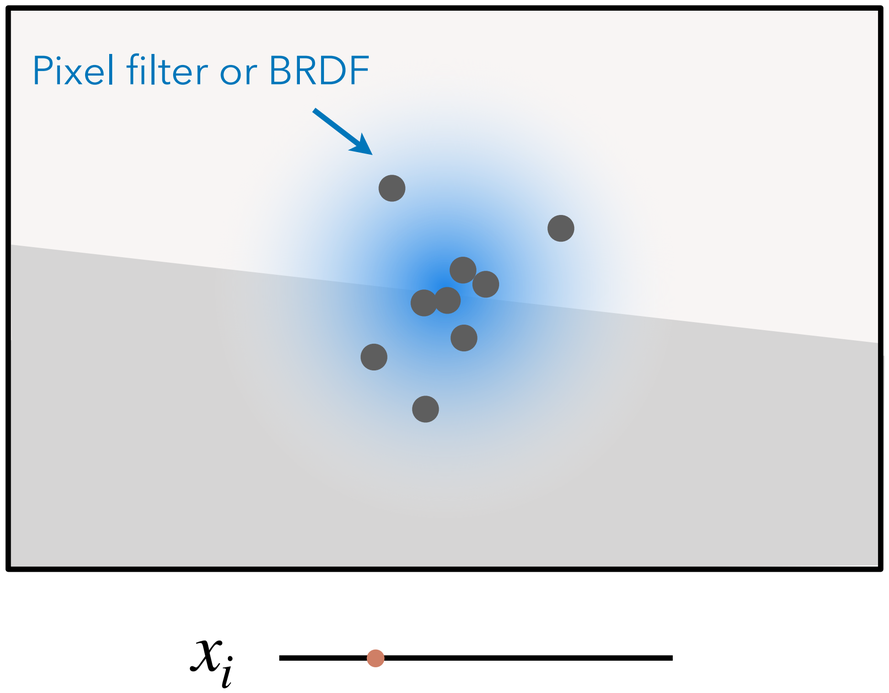


Differentiable Monte Carlo estimates

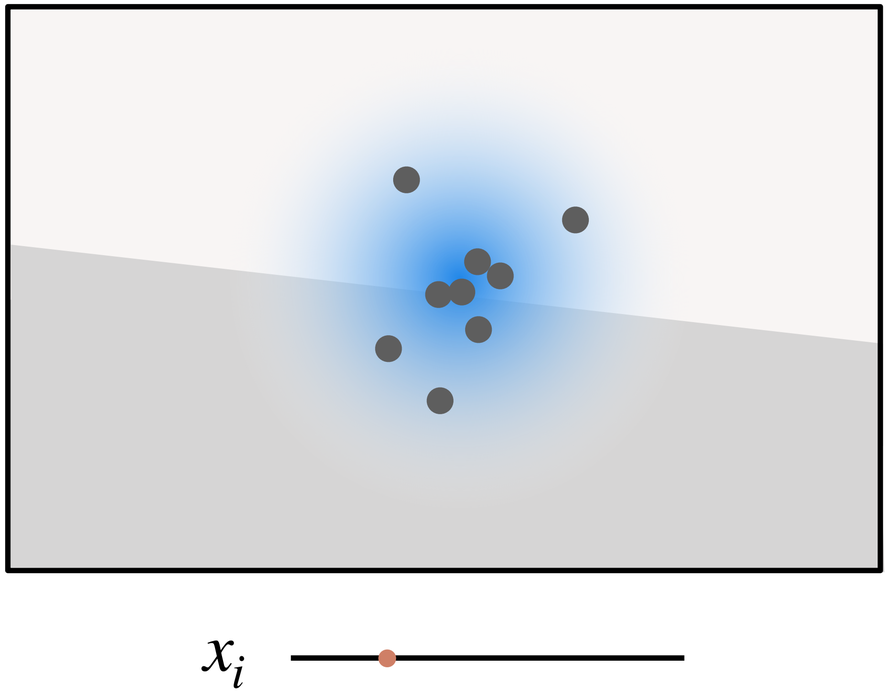


Our approach: reparameterizing integrals

Non-differentiable Monte Carlo estimates



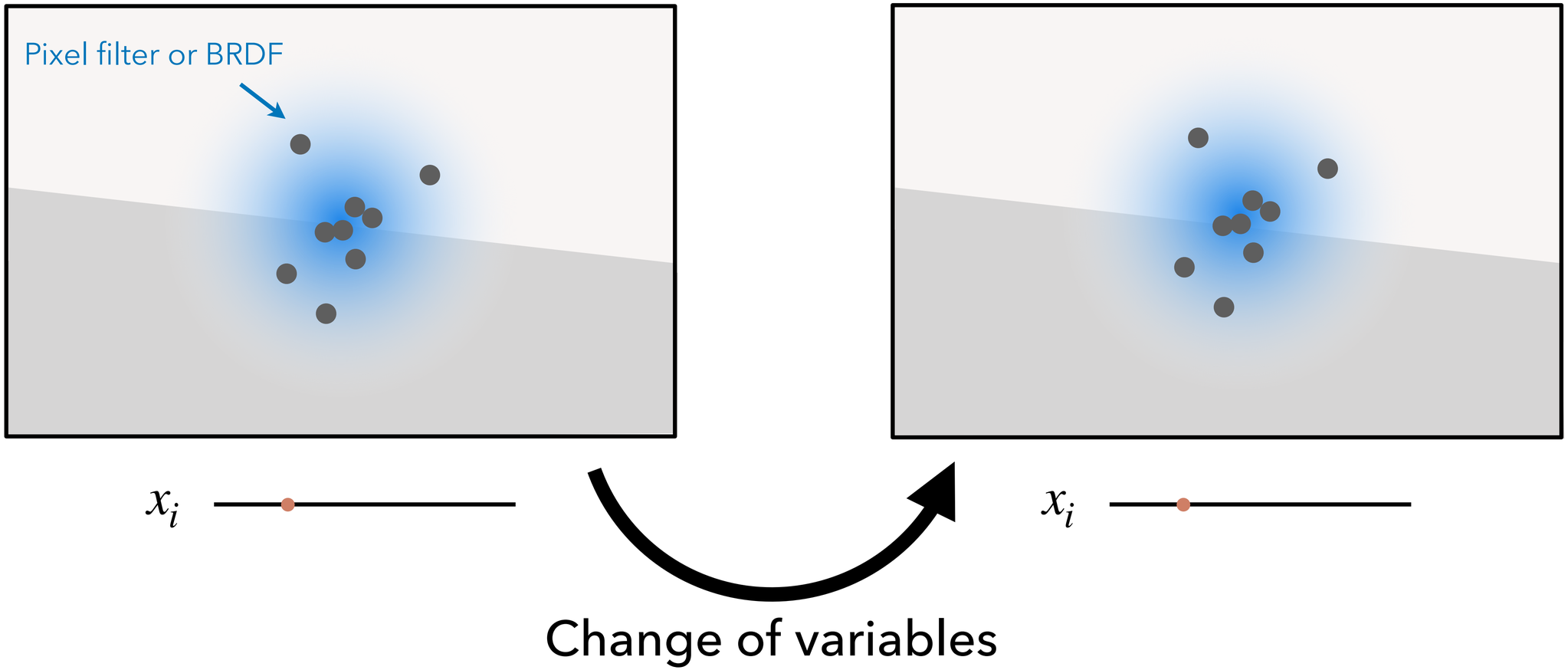
Differentiable Monte Carlo estimates



Our approach: reparameterizing integrals

Non-differentiable Monte Carlo estimates

Differentiable Monte Carlo estimates



More details

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 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

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

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;

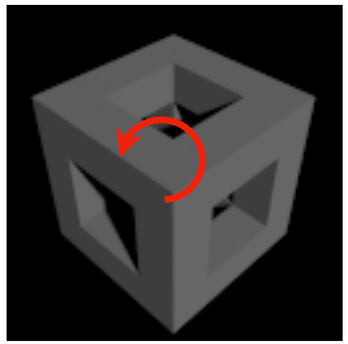
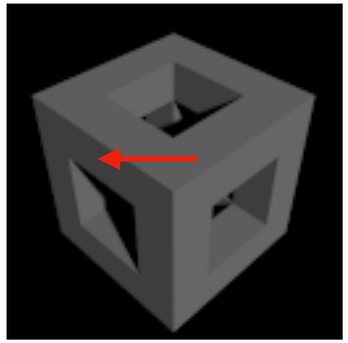
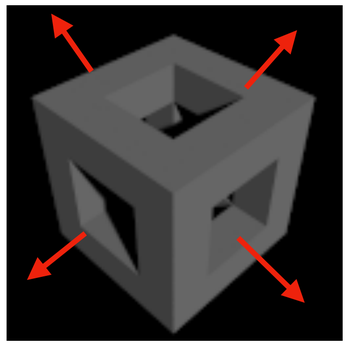
0730-0301/2019/11-ART228 $15.00
https://doi.org/10.1145/3355089.3356510

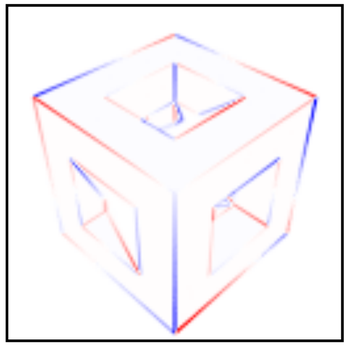
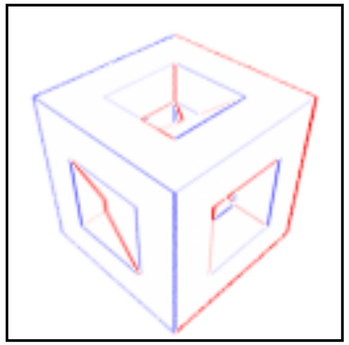
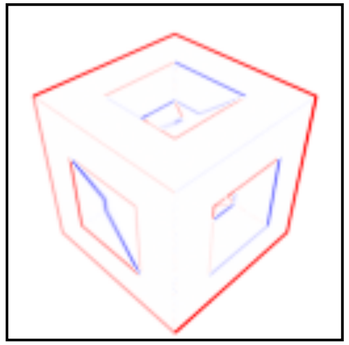
Reparameterizing Discontinuous Integrals for Differentiable Rendering

Guillaume Loubet, Nicolas Holzschuch, Wenzel Jakob

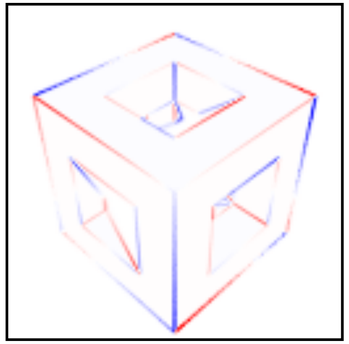
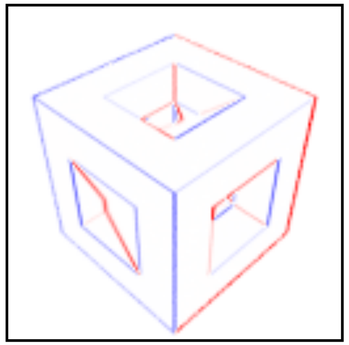
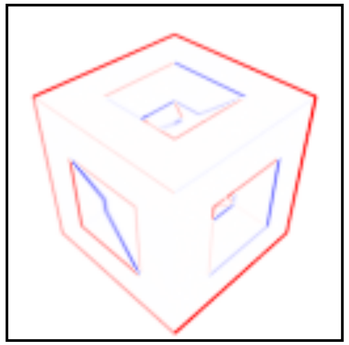
SIGGRAPH Asia 2019

Results: comparison to reference gradient images

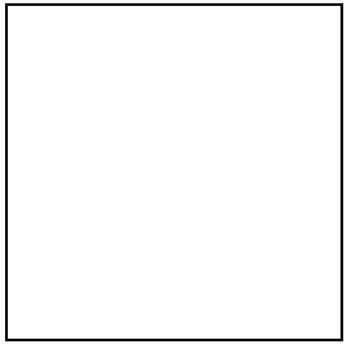
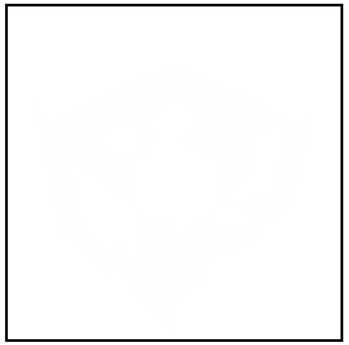




Ours

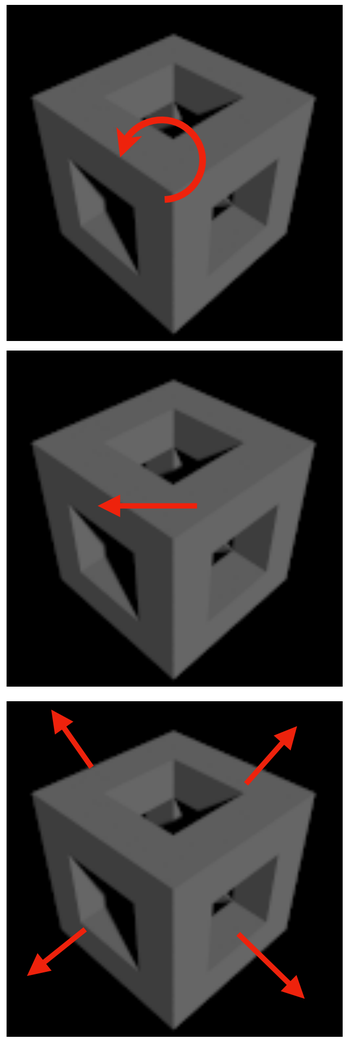




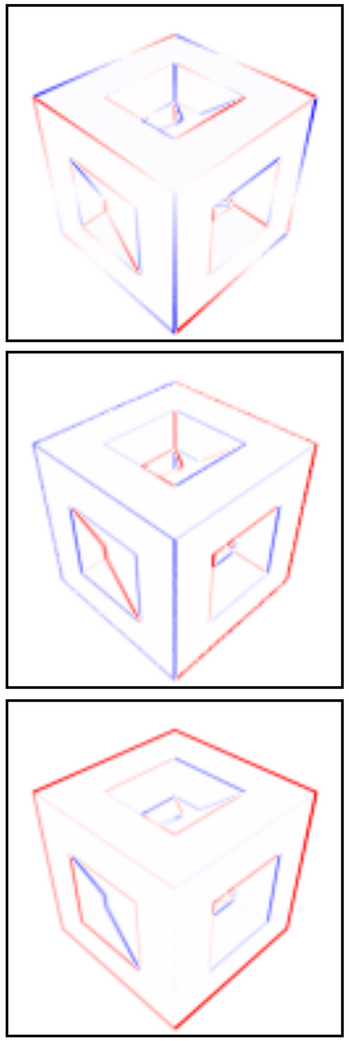
Reference
(Finite differences)

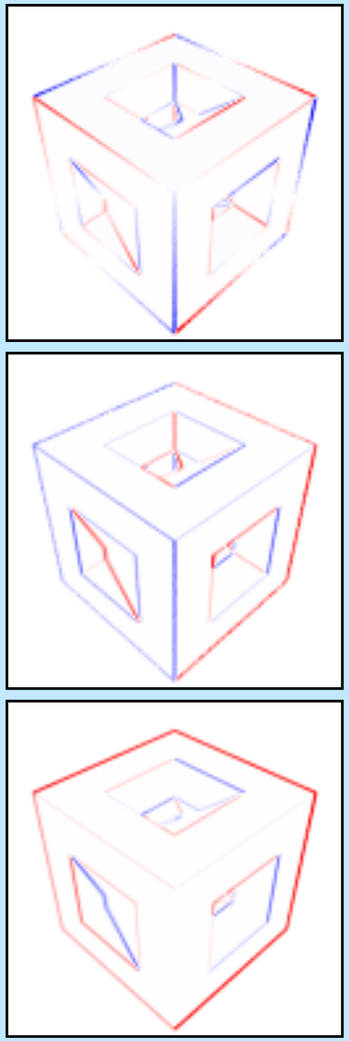
Without changes of variables

Results: comparison to reference gradient images

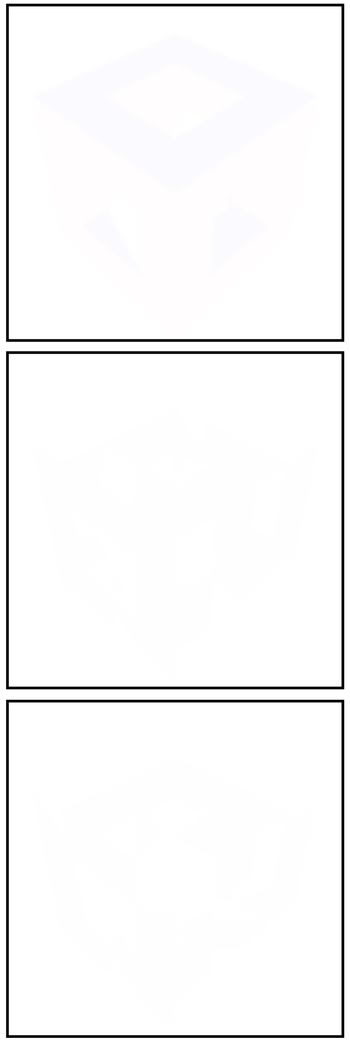




Ours

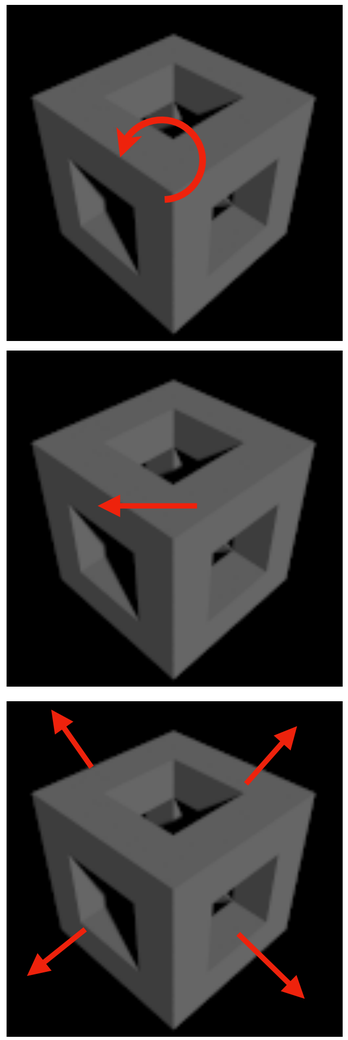


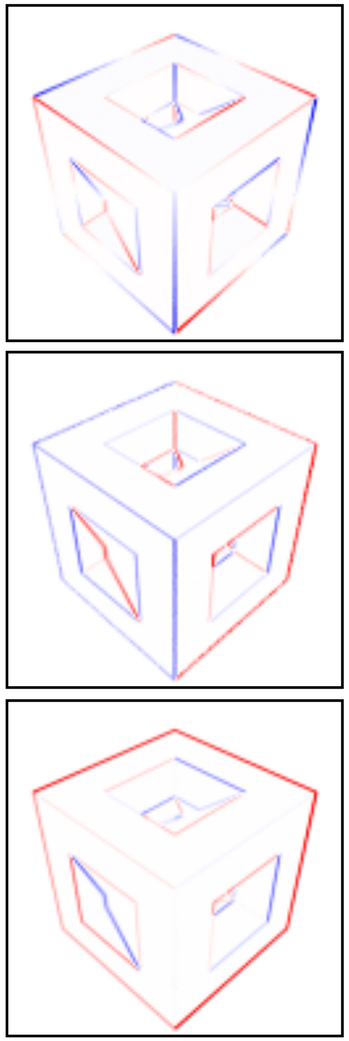
Reference
(Finite differences)



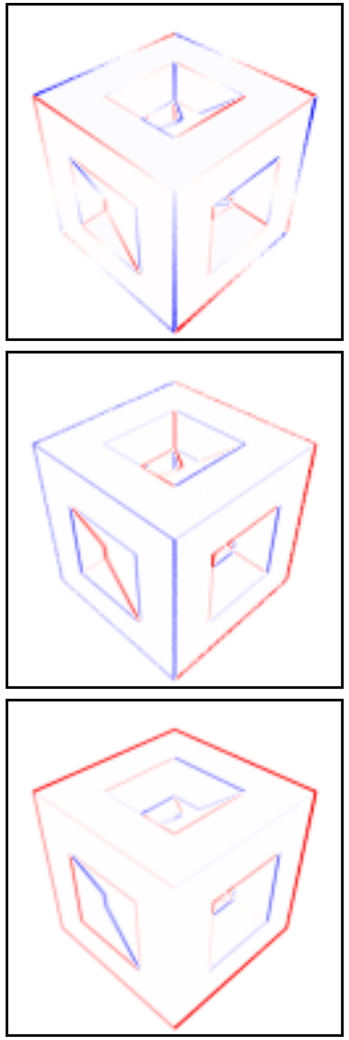
Without changes of variables

Results: comparison to reference gradient images

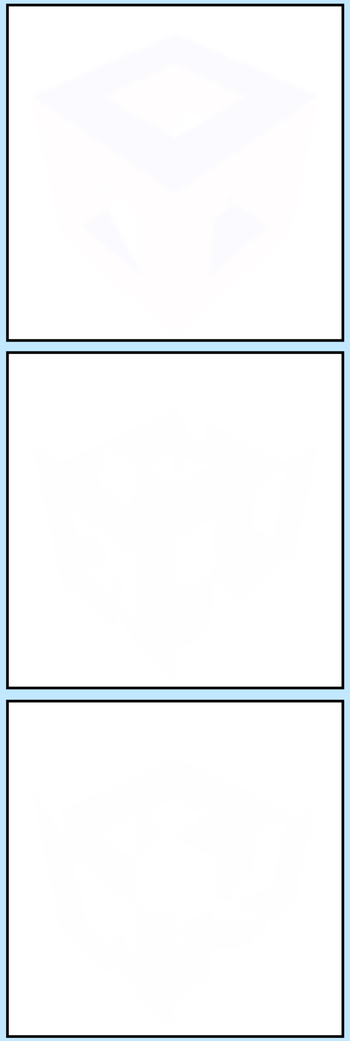




Ours

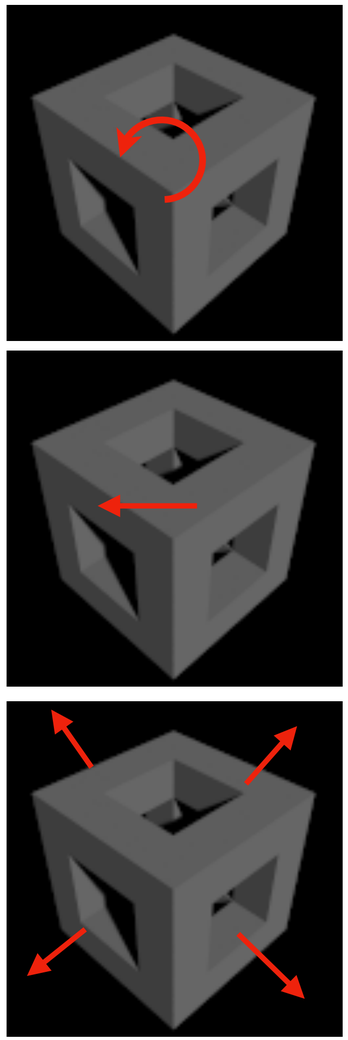


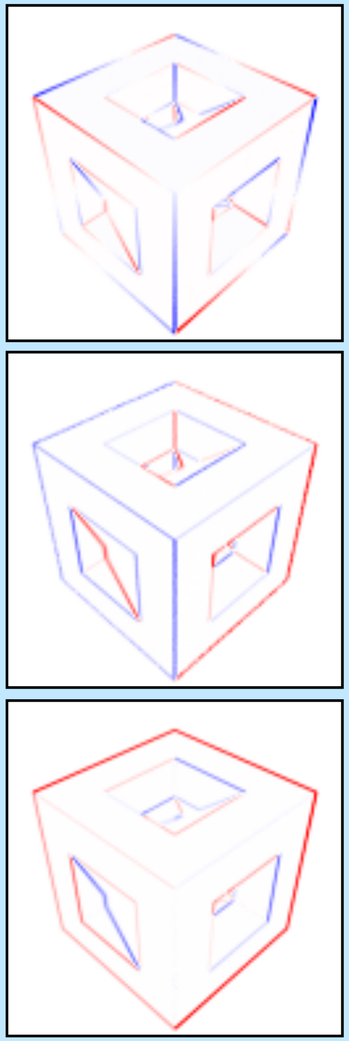
Reference
(Finite differences)



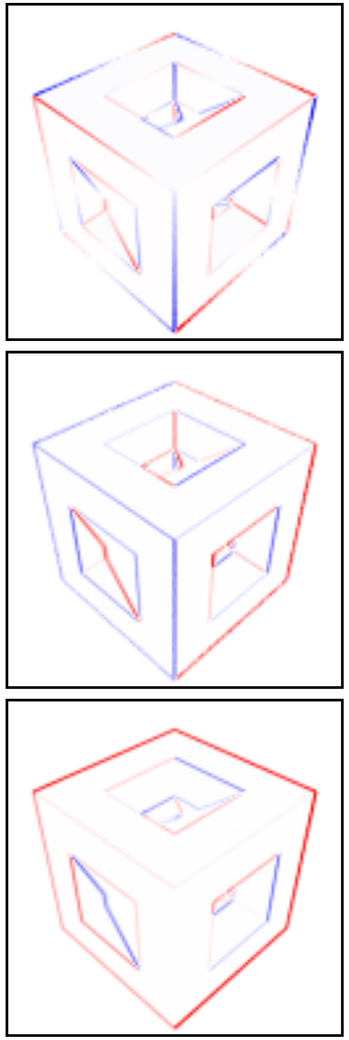
Without changes of variables

Results: comparison to reference gradient images

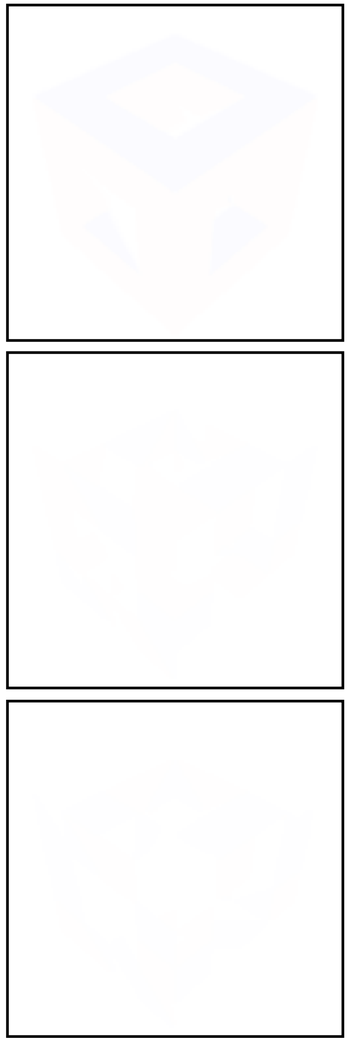




Ours



Reference
(Finite differences)



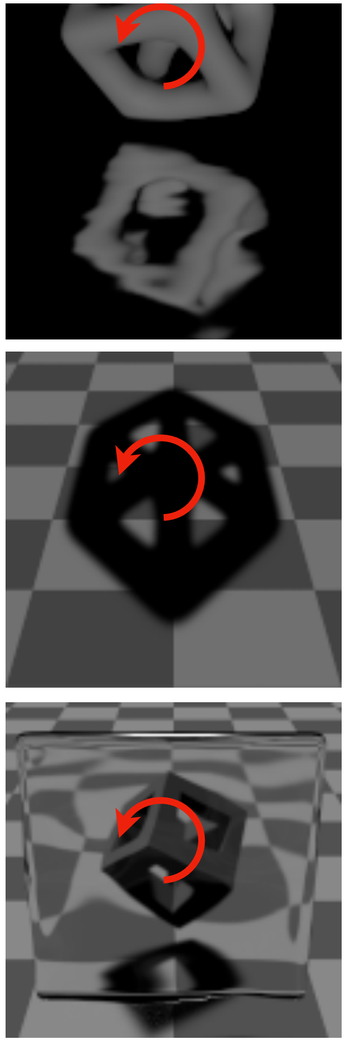
Without changes of variables

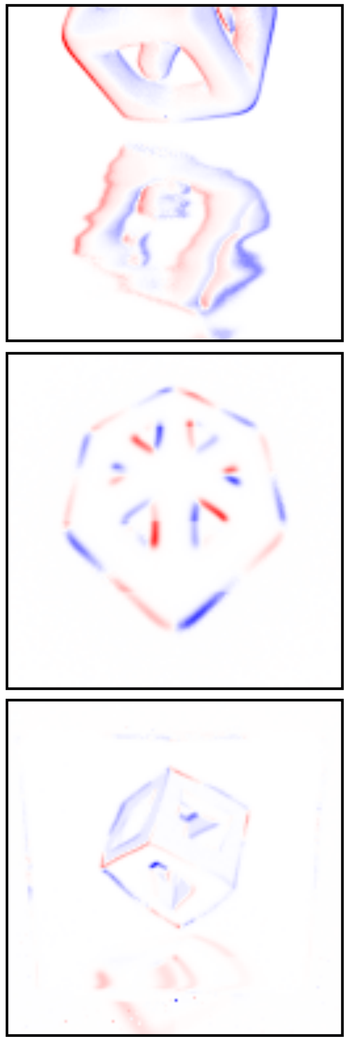
Results: comparison to reference gradient images

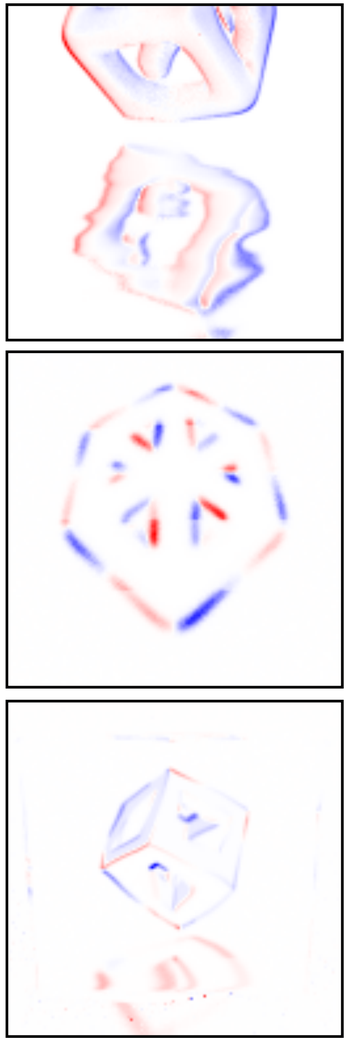
Glossy reflection

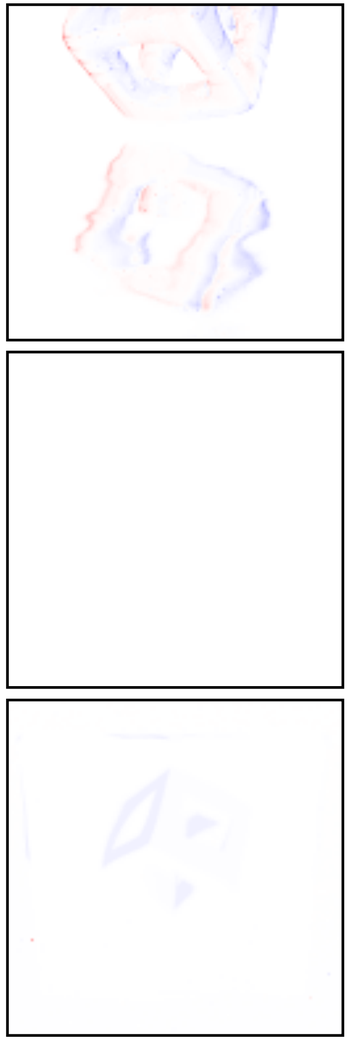
Shadows

Refraction









Ours

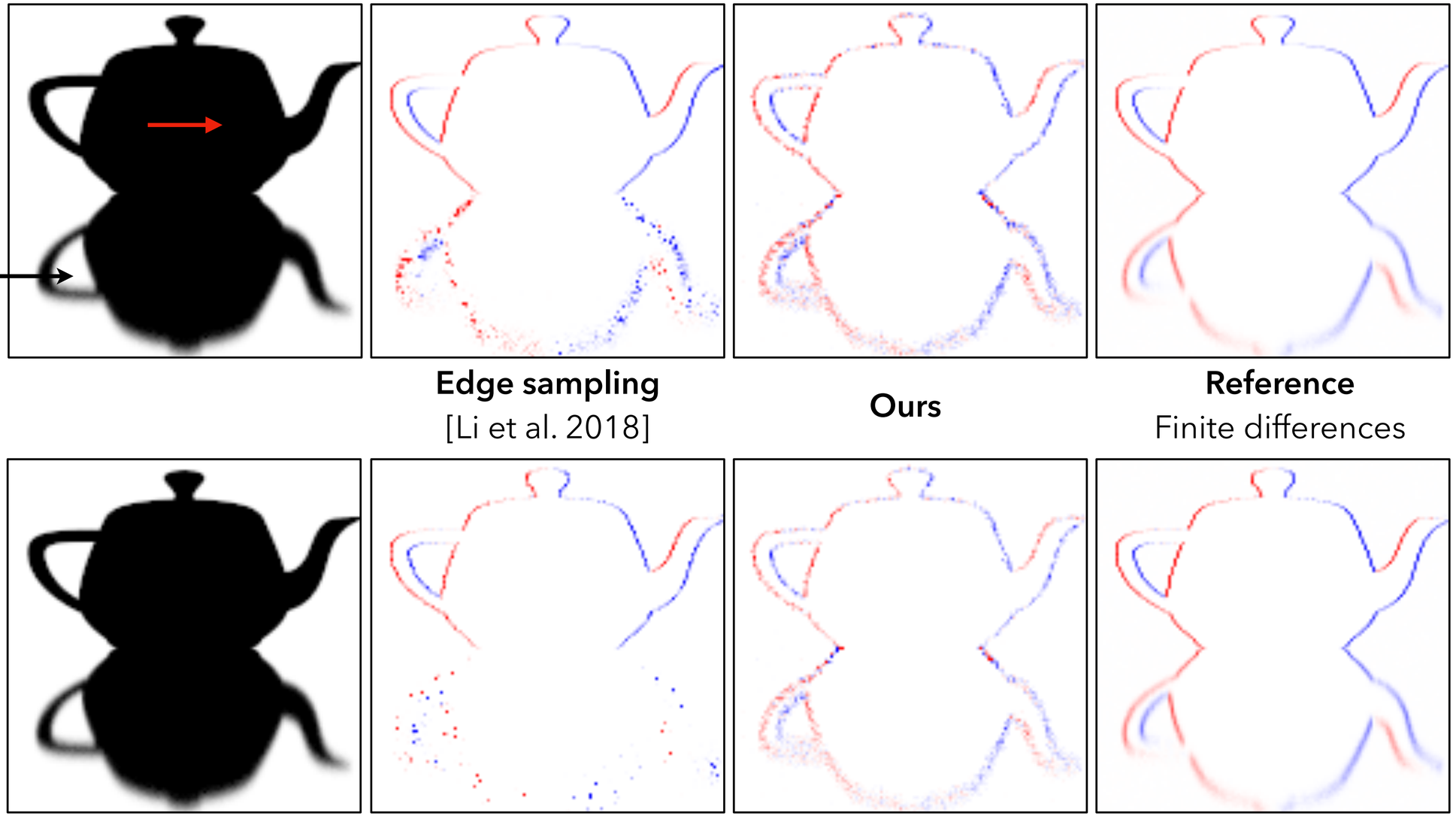
Reference
(Finite differences)

Without
changes of variables

Results: comparison to edge sampling

Glossy reflection

Mesh subdivision



Edge sampling
[Li et al. 2018]

Ours

Reference
Finite differences

Agenda for today

Inverse rendering

Example problem

Differentiable rendering

Challenges

1. How to do this at all?
2. Efficiency
3. Discontinuities
4. **Robustness**

Robustness

Major contrast to neural networks:

- Neural networks use random initialization! High-dimensional layout of local minima + noisy SGD tends to find good solutions.
- **Not true for differentiable rendering.**
  (would be nice to start from random/empty initialization)

Robustness

Major contrast to neural networks:

- Neural networks use random initialization! High-dimensional layout of local minima + noisy SGD tends to find good solutions.
- **Not true for differentiable rendering.**
  (would be nice to start from random/empty initialization)

Robustness

Major contrast to neural networks:

- Neural networks use random initialization! High-dimensional layout of local minima + noisy SGD tends to find good solutions.
- **Not true for differentiable rendering.**
  (would be nice to start from random/empty initialization)

- Optimization objective: what to choose?

Robustness

Major contrast to neural networks:

- Neural networks use random initialization! High-dimensional layout of local minima + noisy SGD tends to find good solutions.
- **Not true for differentiable rendering.**
  (would be nice to start from random/empty initialization)

- Optimization objective: what to choose?
- Which scene representations promote convexity?

Robustness

Major contrast to neural networks:

- Neural networks use random initialization! High-dimensional layout of local minima + noisy SGD tends to find good solutions.
- **Not true for differentiable rendering.**
  (would be nice to start from random/empty initialization)

- Optimization objective: what to choose?
- Which scene representations promote convexity?
- Tweaking multiple knobs at during optimization seems hacky..

Robustness

Major contrast to neural networks:

- Neural networks use random initialization! High-dimensional layout of local minima + noisy SGD tends to find good solutions.
- **Not true for differentiable rendering.**
  (would be nice to start from random/empty initialization)

- Optimization objective: what to choose?
- Which scene representations promote convexity?
- Tweaking multiple knobs at during optimization seems hacky..

Exciting times!!

EPFL **RGL**

Realistic Graphics Lab

Differentiable Simulation of Light

Why it is Important, and What Makes it Hard!

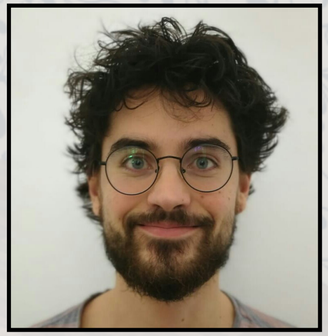
Model credits:

- Olesya Jakob
- Benedikt Bitterli's scene repository
- HDRI Haven
- BlenderArtists/TurboSquid users

  Warcos, blendswapisweird, Master2main and Young_Wizard


Merlin Nimier-David


Guillaume Loubet


Benoît Ruiz


Sébastien Speierer


Delio Vicini


Tizian Zeltner


Nicolas Holzschuch

My lab is hiring (PhDs + Postdocs)