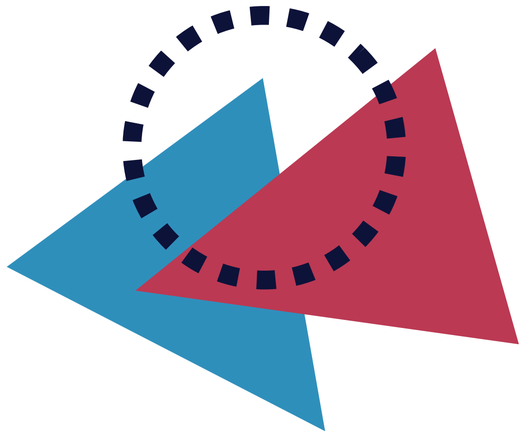
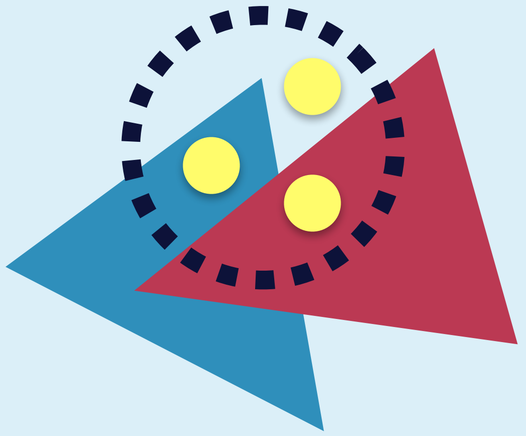
Differentiable Rendering 3

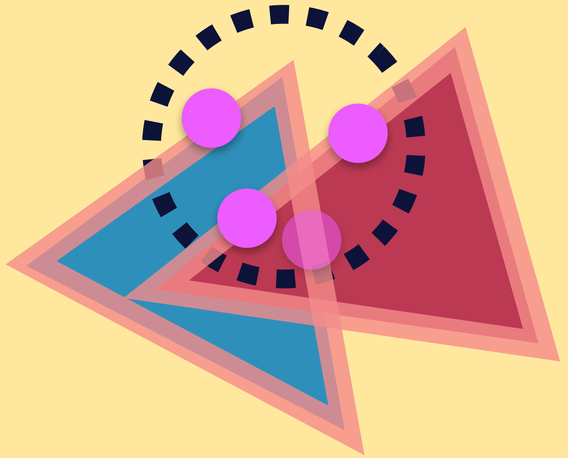
UCSD CSE 272
Advanced Image Synthesis

Tzu-Mao Li

Today: interior derivatives

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

interior derivative

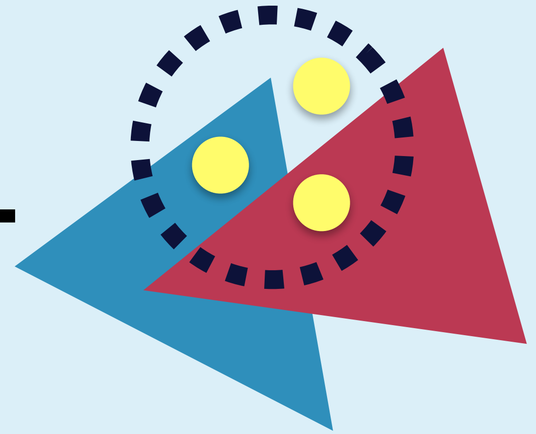
$$+ \int$$ 

boundary derivative

Reynolds transport theorem
[Reynolds 1903]

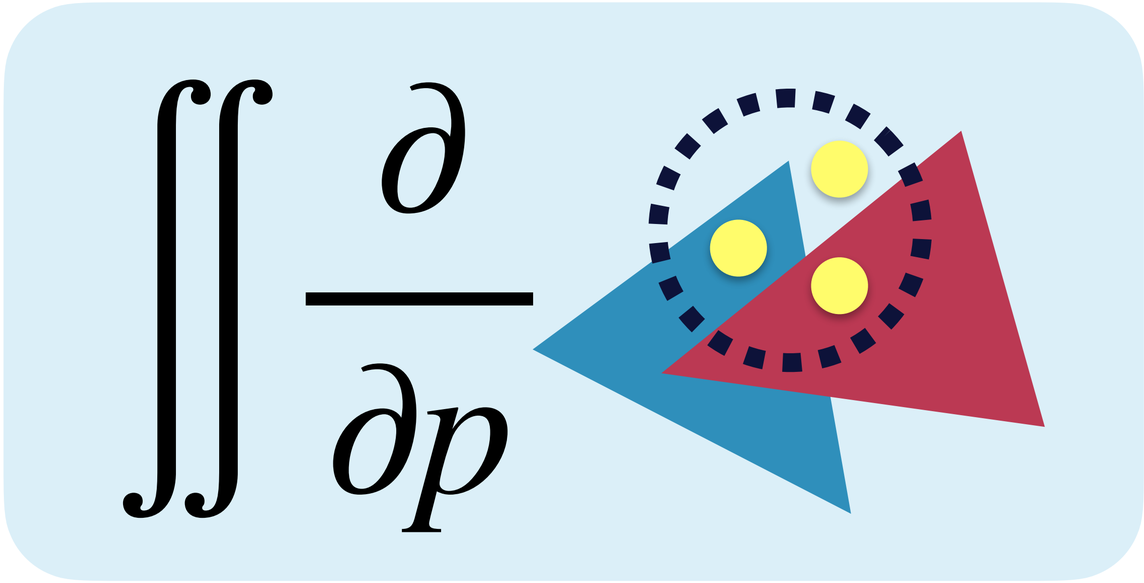
Two challenges of computing the interior derivatives

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



Two challenges of computing the interior derivatives

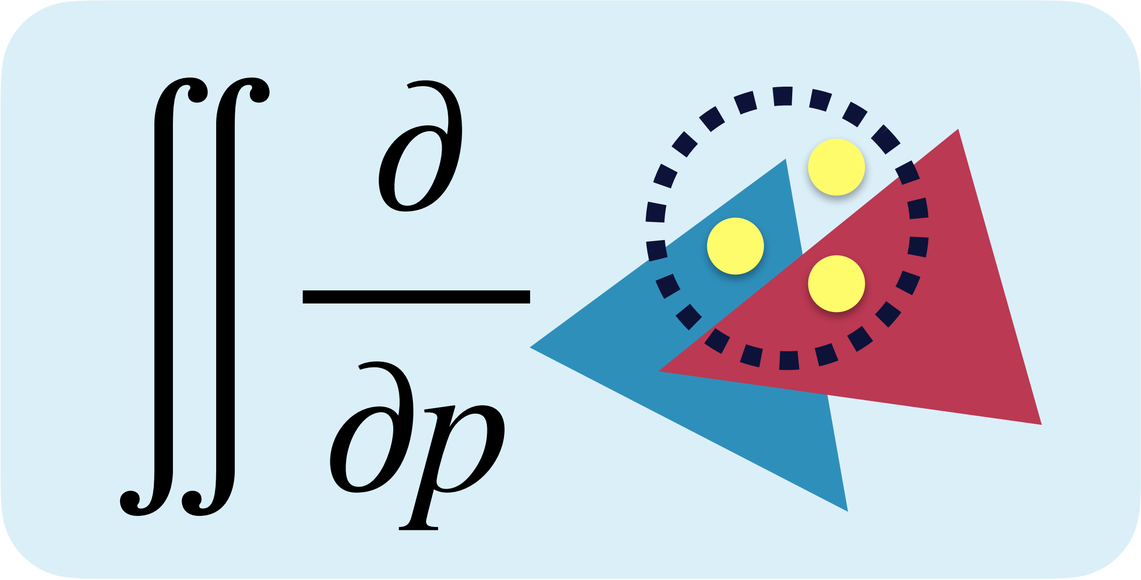
- importance sampling of forward rendering
  may not be a good strategy for differentiable rendering



Two challenges of computing the interior derivatives

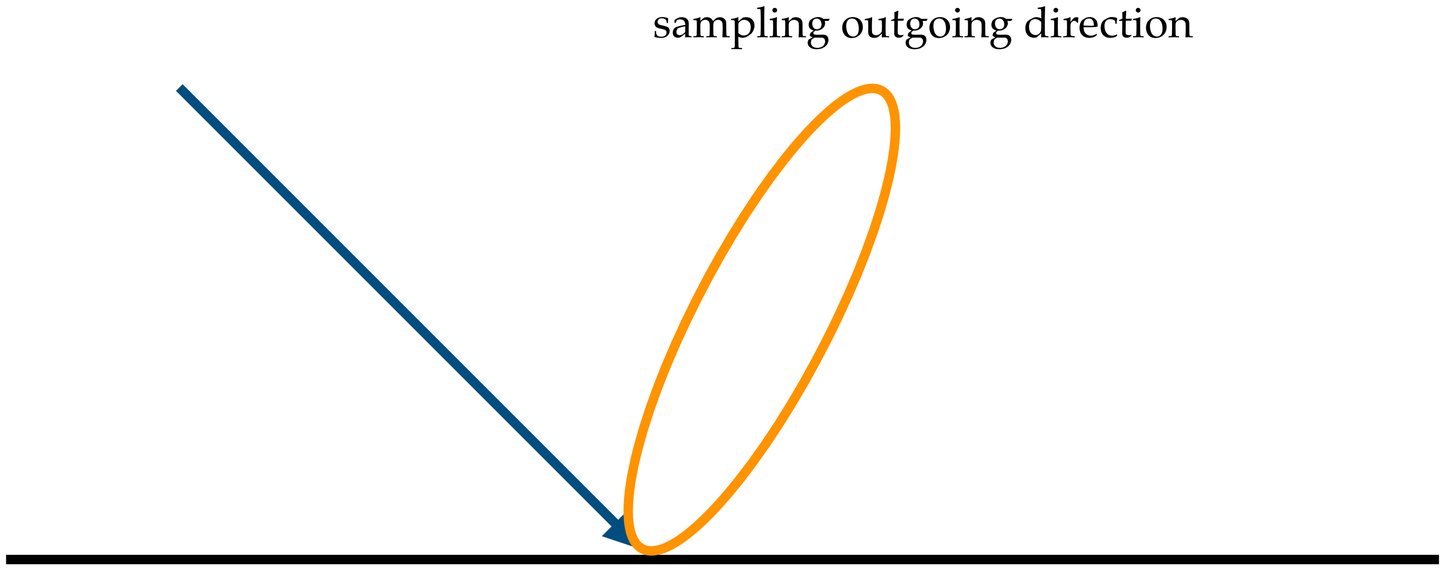
- importance sampling of forward rendering
  may not be a good strategy for differentiable rendering

- reverse-mode autodiff can lead to
  high memory consumption



Importance sampling in differentiable rendering

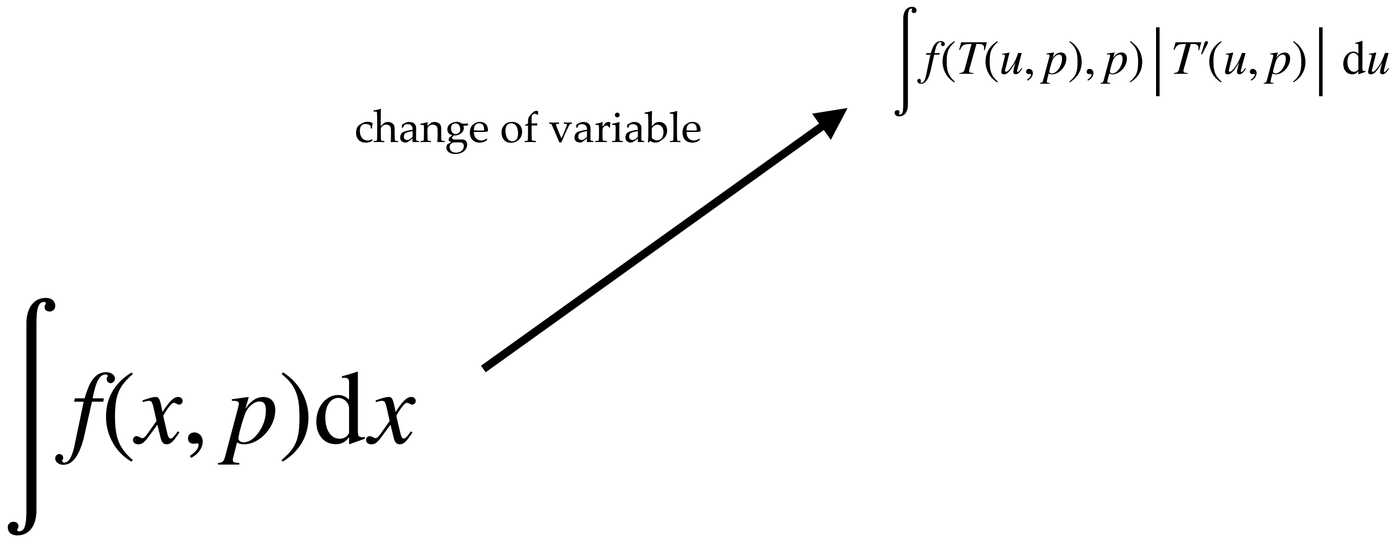
quiz: how did we do it in a forward path tracer? how should we do it in differentiable rendering?



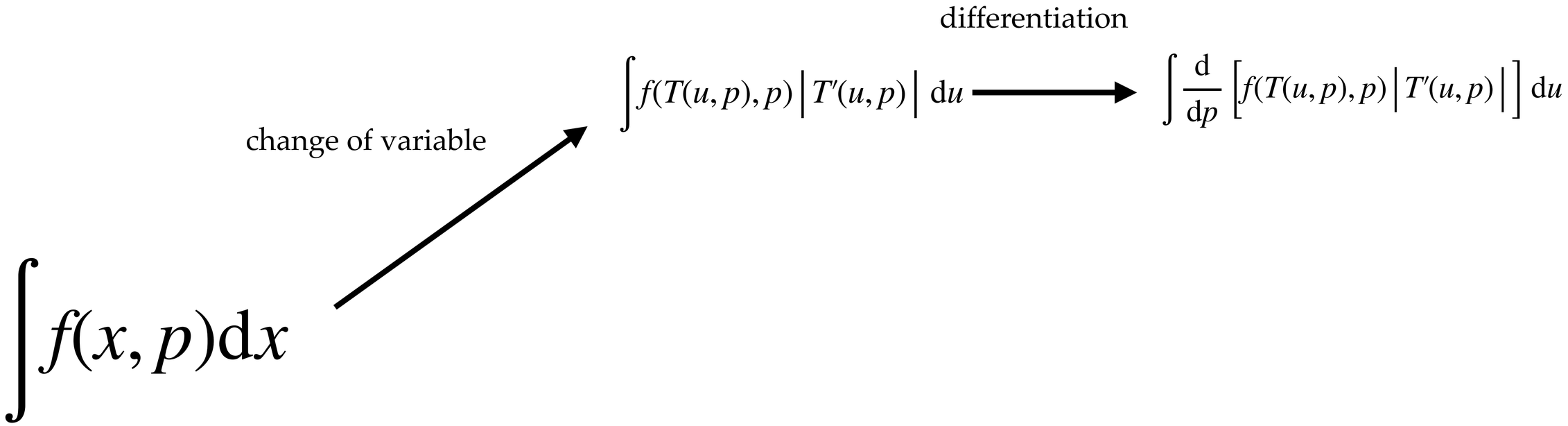
Change of variable vs differentiation

$$\int f(x, p)\mathrm{d}x$$

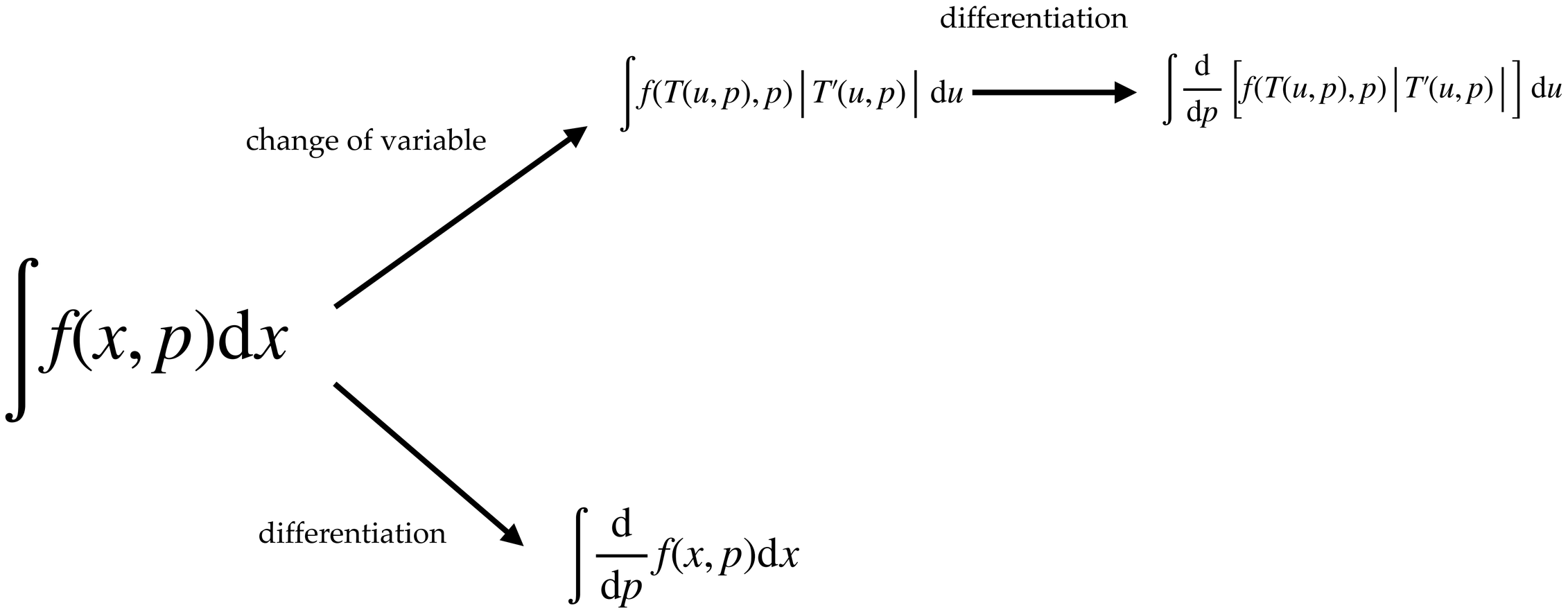
Change of variable vs differentiation



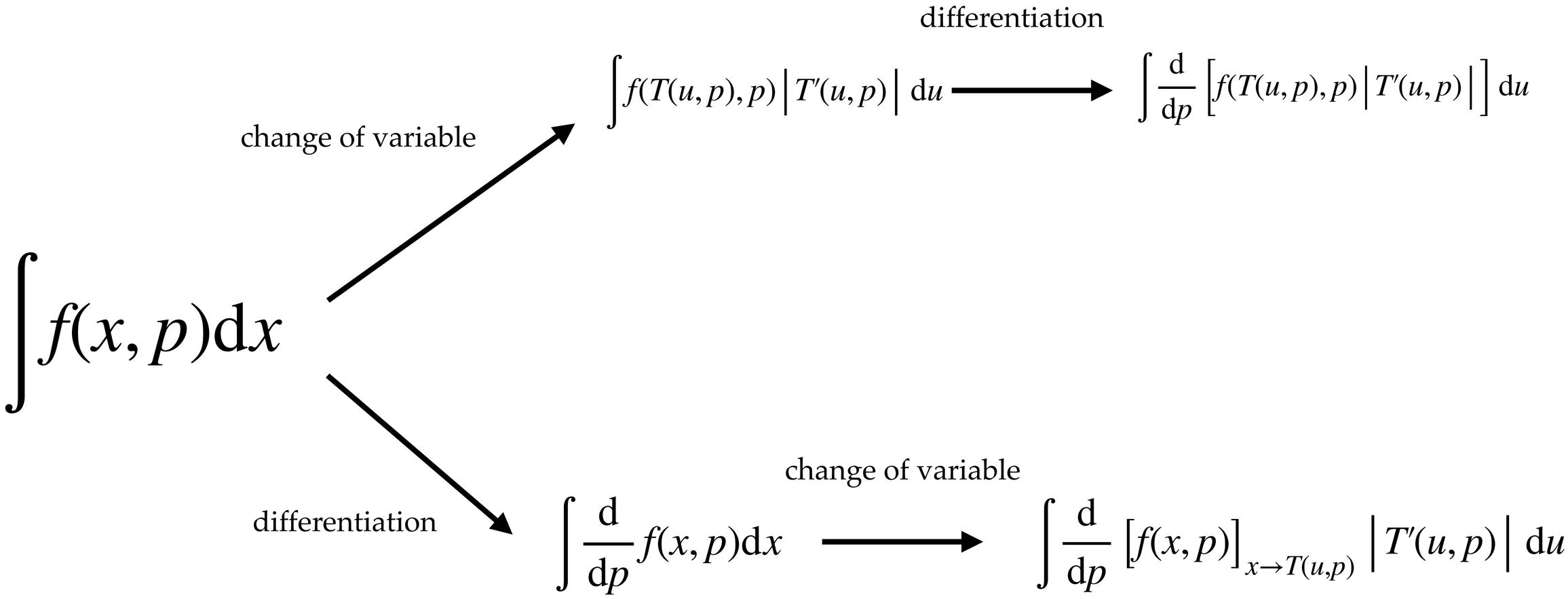
Change of variable vs differentiation



Change of variable vs differentiation



Change of variable vs differentiation



Change of variable vs differentiation

$$\int f(x,p)\mathrm{d}x$$

change of variable

$$\int f(T(u,p),p)\left|T'(u,p)\right|\mathrm{d}u$$

differentiation

$$\int \frac{\mathrm{d}}{\mathrm{d}p}\left[f(T(u,p),p)\left|T'(u,p)\right|\right]\mathrm{d}u$$

differentiation

$$\int \frac{\mathrm{d}}{\mathrm{d}p} f(x,p)\mathrm{d}x$$

change of variable

$$\int \frac{\mathrm{d}}{\mathrm{d}p}\left[f(x,p)\right]_{x\to T(u,p)}\left|T'(u,p)\right|\mathrm{d}u$$

what are the differences?

There are many different ways to evaluate interior derivatives!

- open questions
  - when should we use which one? how do we choose "T"?
  - how to do multiple importance sampling?

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(x,p) \right]_{x \to T(u,p)} \left| T'(u,p) \right| \, \mathrm{d}u$$

differentiate-then-reparametrize
(aka "detached")

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

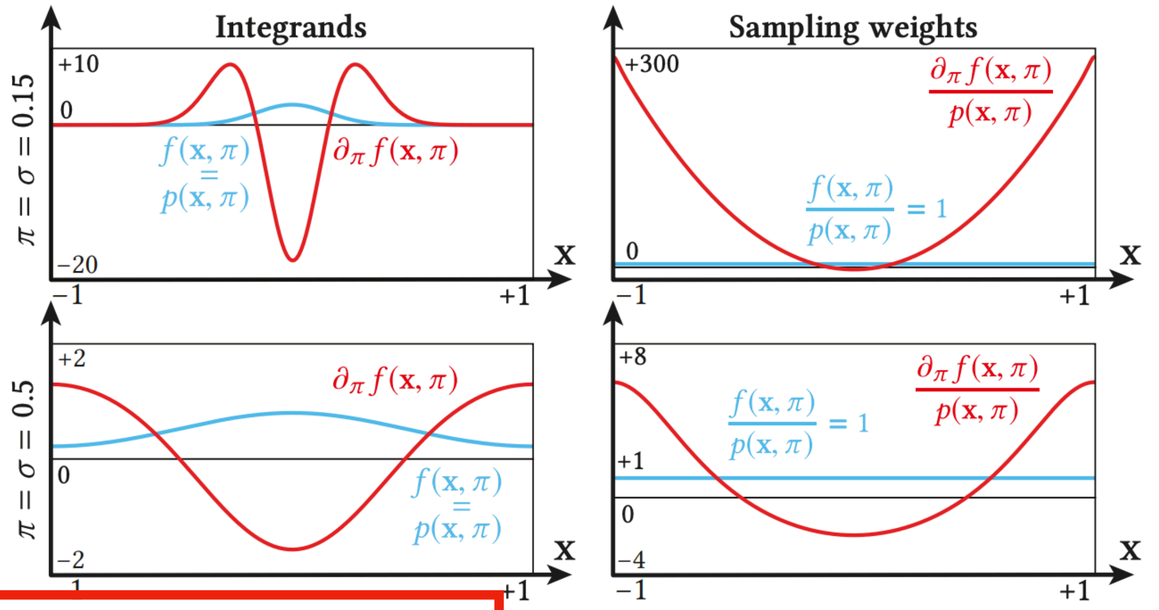
reparametrize-then-differentiate
(aka "attached")

$$\int \left[\frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \right]_{u \to \hat{T}(v,p)} \mathrm{d}v$$

(no one has done this yet)

reparametrize-then-differentiate-then-reparametrize

Differentiate-then-reparametrize: reusing T from forward rendering can be bad



sharp Gaussian

flat Gaussian

figure from Zeltner et al.

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(x,p) \right]_{x \to T(u,p)} \left| T'(u,p) \right| \mathrm{d}u$$

differentiate-then-reparametrize
(aka "detached")

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

reparametrize-then-differentiate
(aka "attached")

Reparametrize-then-differentiate: not perfect either

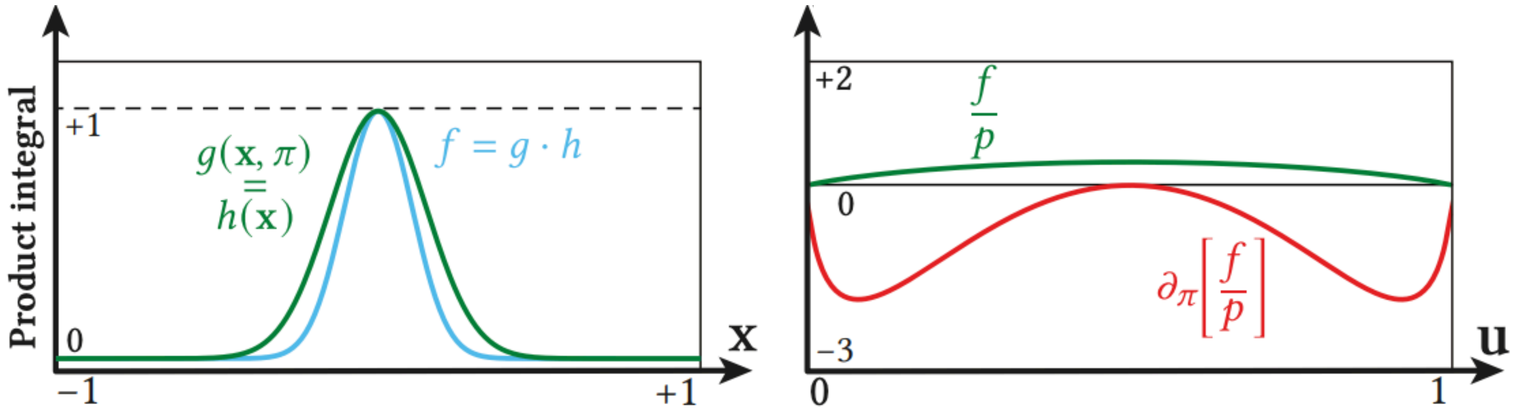


figure from Zeltner et al.

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(x,p) \right]_{x \to T(u,p)} \left| T'(u,p) \right| \, \mathrm{d}u$$

differentiate-then-reparametrize
(aka “detached”)

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

reparametrize-then-differentiate
(aka “attached”)

Reparametrize-then-differentiate: need to deal with discontinuities!

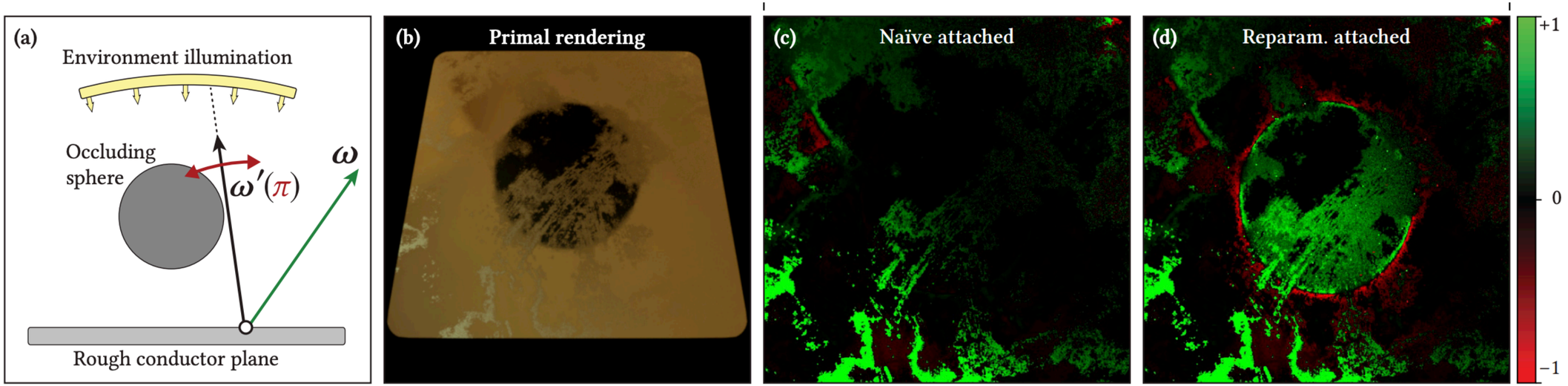


figure from Zeltner et al.

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(x,p) \right]_{x \to T(u,p)} \left| T'(u,p) \right| \, \mathrm{d}u$$

differentiate-then-reparametrize
(aka "detached")

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

reparametrize-then-differentiate
(aka "attached")

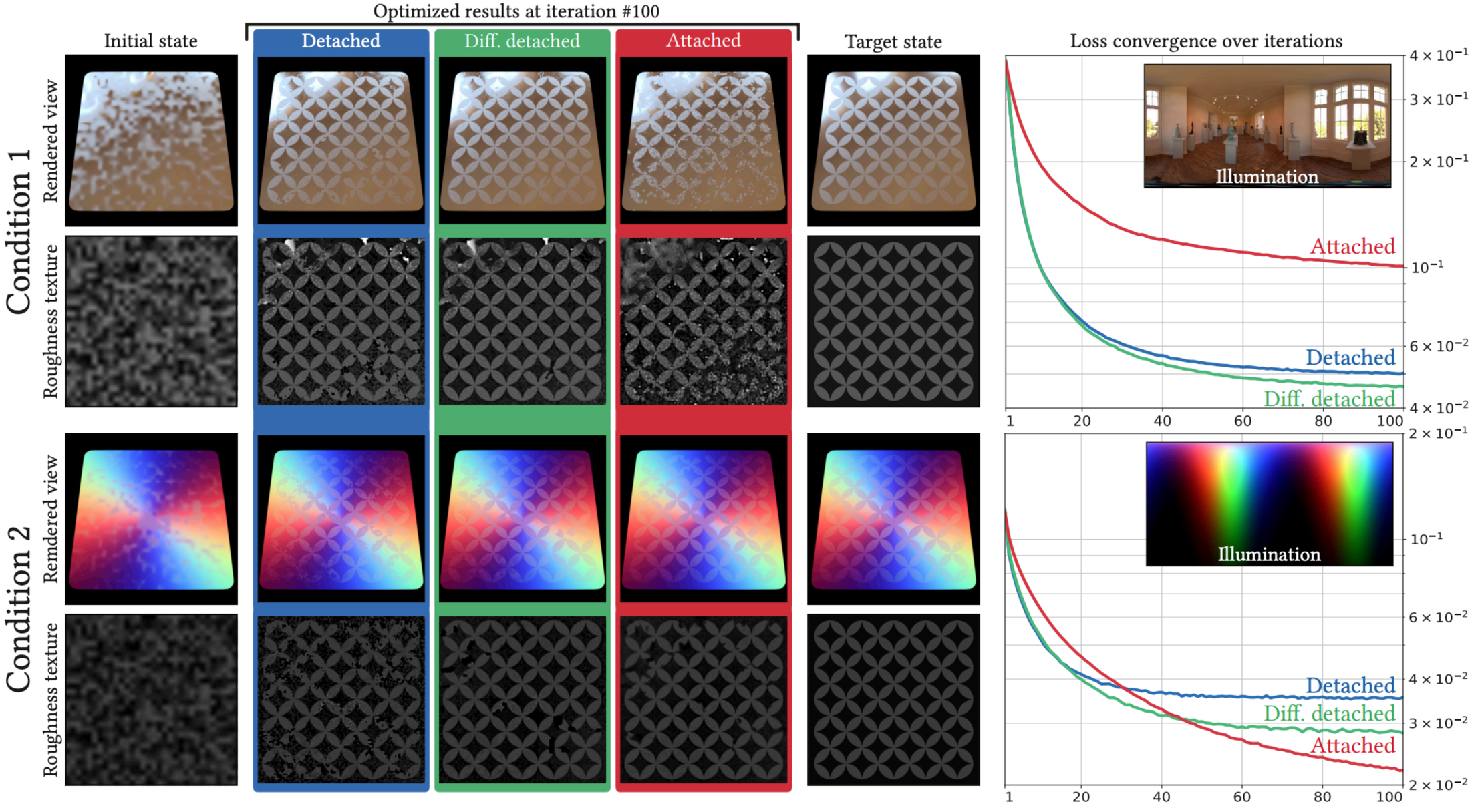
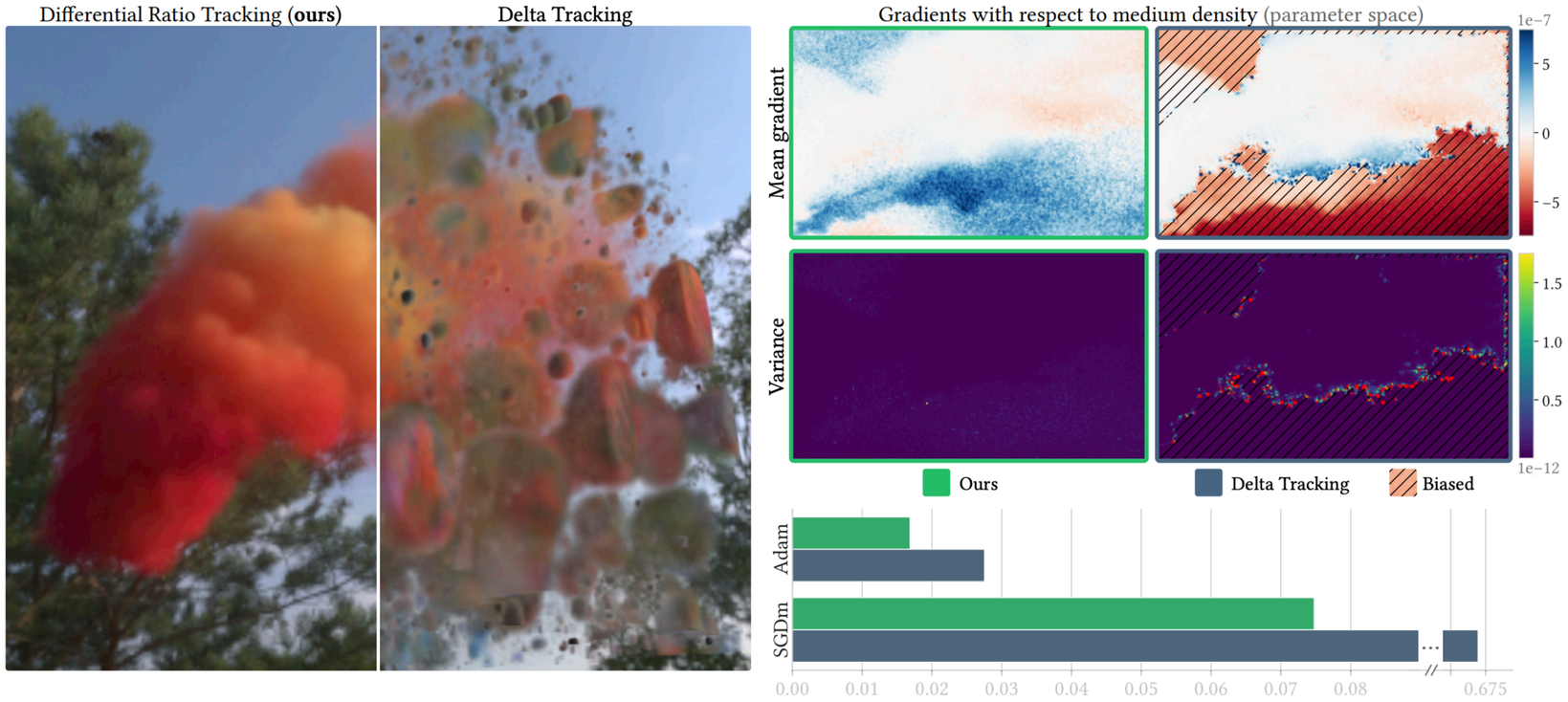


figure from Zeltner et al.

Transmittance sampling in differentiable rendering

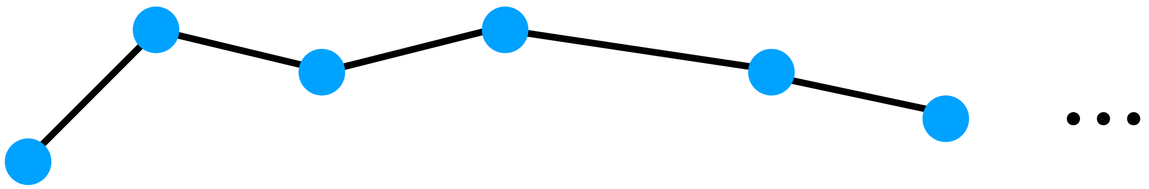
Unbiased Inverse Volume Rendering with Differential Trackers

MERLIN NIMIER-DAVID, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
THOMAS MÜLLER, NVIDIA, Switzerland
ALEXANDER KELLER, NVIDIA, Germany
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland



Memory consumption

- reverse-mode automatic differentiation requires us to memorize the whole light path
  - same reason why training deep networks needs lots of GPU memory!



```
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
```

Two solutions for memory consumption

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(x,p) \right]_{x \to T(u,p)} \left| T'(u,p) \right| \, \mathrm{d}u$$

differentiate-then-reparametrize

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

reparametrize-then-differentiate

Rendering equation: products of BRDFs

$$L = f_0 f_1 f_2 \cdots L_e$$

Rendering equation: products of BRDFs

$$L = f_0 f_1 f_2 \cdots L_e$$

product rule

$$\nabla L = \nabla f_0 f_1 f_2 \cdots L_e + f_0 \nabla f_1 f_2 \cdots L_e + f_0 f_1 \nabla f_2 \cdots L_e \cdots$$

naive evaluation: O(n^2)

Rendering equation: products of BRDFs

$$L = f_0 f_1 f_2 \cdots L_e$$

product rule

$$\nabla L = \nabla f_0 f_1 f_2 \cdots L_e + f_0 \nabla f_1 f_2 \cdots L_e + f_0 f_1 \nabla f_2 \cdots L_e \cdots$$

reverse-mode = caching shared terms (need to store F_0 to F_n)

time complexity: O(n)
memory complexity: O(n)

$$F_0 = f_1 f_2 \cdots L_e \qquad F_1 = f_0 f_2 \cdots L_e \qquad \nabla L = \nabla f_0 F_0 + \nabla f_1 F_1 + \nabla f_2 F_2 \cdots$$

Rendering equation: products of BRDFs

$$L = f_0 f_1 f_2 \cdots L_e$$

product rule

$$\nabla L = \nabla f_0 f_1 f_2 \cdots L_e + f_0 \nabla f_1 f_2 \cdots L_e + f_0 f_1 \nabla f_2 \cdots L_e \cdots$$

$$= \nabla f_0 \frac{L}{f_0} + \nabla f_1 \frac{L}{f_1} + \nabla f_2 \frac{L}{f_2} \cdots$$

time complexity: O(n)
memory complexity: O(1)

instead of caching: reconstruct shared terms from L

Rendering equation: products of BRDFs

$$L = f_0 f_1 f_2 \cdots L_e$$

product rule

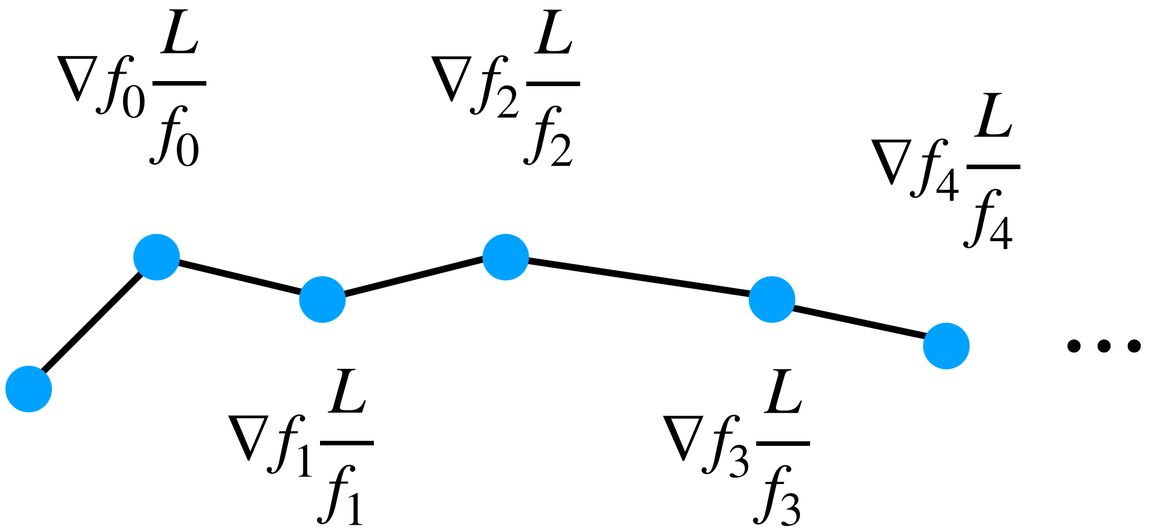
$$\nabla L = \nabla f_0 f_1 f_2 \cdots L_e + f_0 \nabla f_1 f_2 \cdots L_e + f_0 f_1 \nabla f_2 \cdots L_e \cdots$$

$$= \nabla f_0 \frac{L}{f_0} + \nabla f_1 \frac{L}{f_1} + \nabla f_2 \frac{L}{f_2} \cdots$$

time complexity: O(n)
memory complexity: O(1)

instead of caching: reconstruct shared terms from L

Efficient derivative in constant memory



$$\nabla L = \nabla f_0 \frac{L}{f_0} + \nabla f_1 \frac{L}{f_1} + \nabla f_2 \frac{L}{f_2} \cdots$$

Two solutions for memory consumption

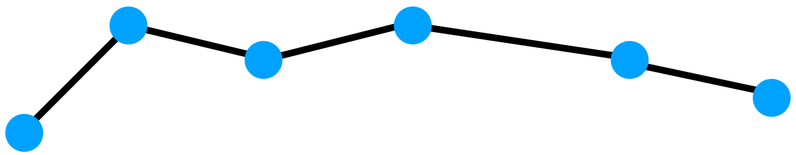
$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(x,p) \right]_{x \to T(u,p)} \left| T'(u,p) \right| \, \mathrm{d}u$$

differentiate-then-reparametrize

$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

reparametrize-then-differentiate

Reparametrize-then-differentiate requires us to differentiate through sampling



$$\int \frac{\mathrm{d}}{\mathrm{d}p} \left[f(T(u,p),p) \left| T'(u,p) \right| \right] \mathrm{d}u$$

$$\omega' = \text{sample\_bsdf}(\omega, u, p)$$

Review: reverse-mode autodiff

```
f(x, y):
    a = g(x, y)
    b = h(a)
    return b
```

```
Df(x, y, dL/db):
    a = g(x, y)
    b = h(a)
    dL/da = Dh(a, dL/db)
    dL/dx, dL/dy = Dg(x, y, dL/da)
    return b, dL/dx, dL/dy
```

Review: reverse-mode autodiff

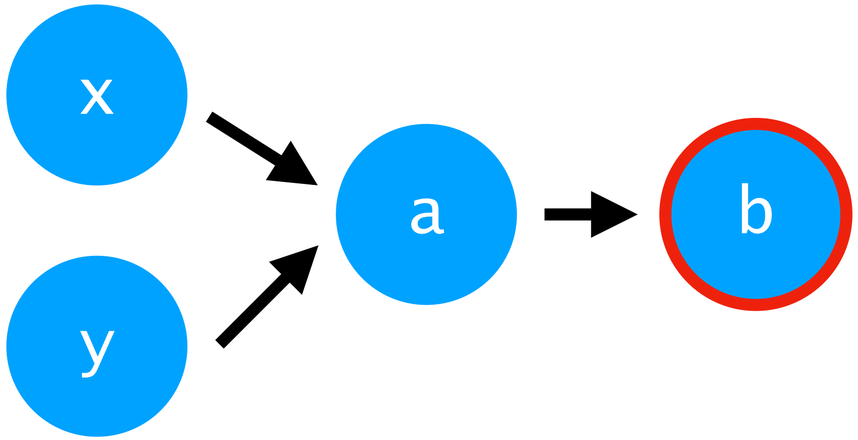
```
f(x, y):
    a = g(x, y)
    b = h(a)
    return b
```

Df(x, y, $\frac{\mathrm{d}L}{\mathrm{d}b}$):

```
    a = g(x, y)
    b = h(a)
```

computes the gradient in one pass

for each node, propagate the differentials



Review: reverse-mode autodiff

```
f(x, y):
    a = g(x, y)
    b = h(a)
    return b
```

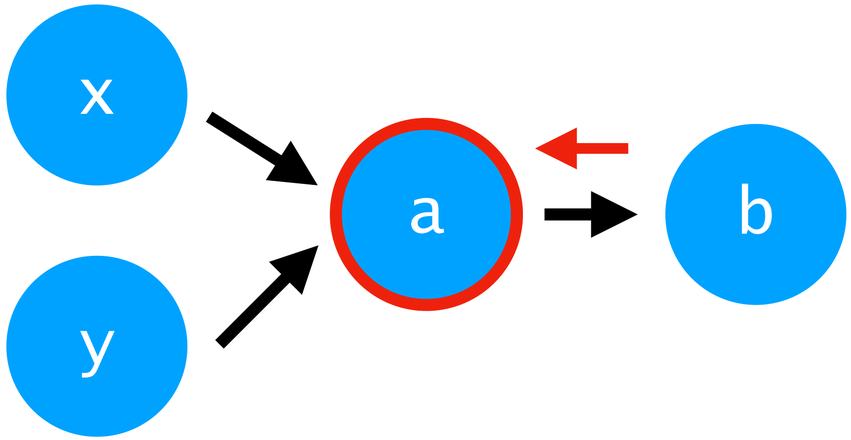
Df(x, y, $\frac{dL}{db}$):

```
    a = g(x, y)
    b = h(a)
```

$\frac{dL}{da}$ = Dh(a, $\frac{dL}{db}$)

computes the gradient in one pass

for each node, propagate the differentials



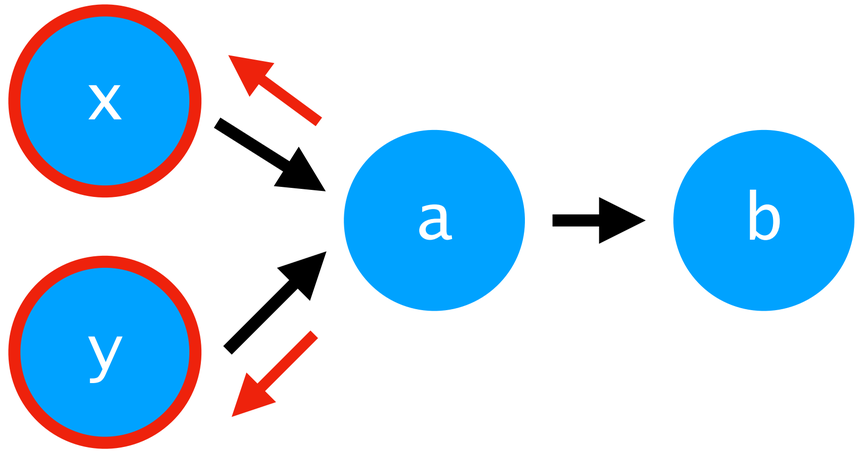
Review: reverse-mode autodiff

```
f(x, y):
    a = g(x, y)
    b = h(a)
    return b
```

```
Df(x, y, dL/db):
    a = g(x, y)
    b = h(a)
    dL/da = Dh(a, dL/db)
    dL/dx, dL/dy = Dg(x, y, dL/da)
    return b, dL/dx, dL/dy
```

computes the gradient in one pass

for each node, propagate the differentials



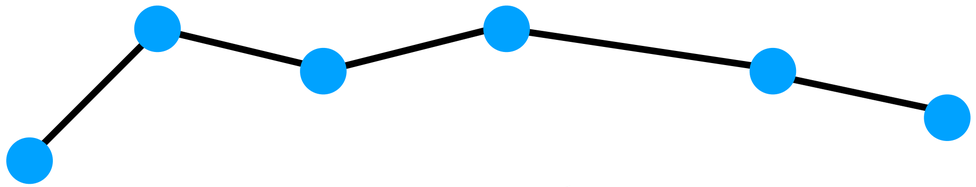
We want to apply reverse mode in a forward fashion

- so that we don't need to remember the forward pass

```
L = 0
throughput = 0
ω_0 = camera_ray
for i in range(1, max_depth):
  ω_i = sample_bsdf(ω_{i-1}, u, p)
  throughput *= bsdf(ω_i, ω_{i-1}, p)
  L += throughput * emission
return L
```

→

```
for i in range(1, max_depth):
  _, _, dL/dp = Dsample_bsdf(ω_{i-1}, u, p, dL/dω_i)
  ∇L += dL/dp
  dL/dp = Dbsdf(ω_i, ω_{i-1}, p, dL/dbsdf)
  ∇L += dL/dp
return ∇L
```



$$\frac{\mathrm{d}L}{\mathrm{dbsdf}} = \nabla f_0 \frac{L}{f_0}$$

$$\frac{\mathrm{d}L}{\mathrm{d}\omega'} = ?$$

We want to apply reverse mode in a forward fashion

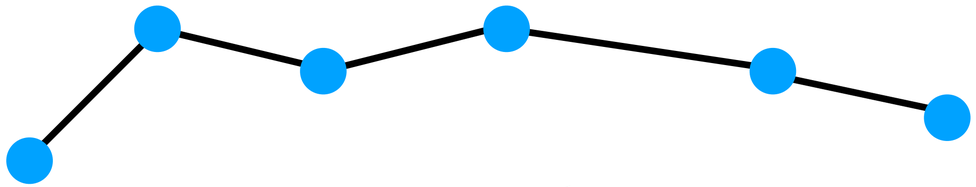
- so that we don't need to remember the forward pass

```
L = 0
throughput = 0
ω_0 = camera_ray
for i in range(1, max_depth):
  ω_i = sample_bsdf(ω_{i-1}, u, p)
  throughput *= bsdf(ω_i, ω_{i-1}, p)
  L += throughput * emission
return L
```

→

```
for i in range(1, max_depth):
  _, _, dL/dp = Dsample_bsdf(ω_{i-1}, u, p, dL/dω_i)
  ∇L += dL/dp
  dL/dp = Dbsdf(ω_i, ω_{i-1}, p, dL/dbsdf)
  ∇L += dL/dp
return ∇L
```

compute using previous solution

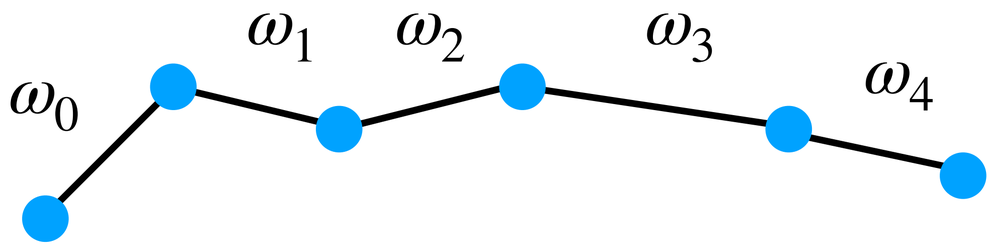


$$\frac{\mathrm{d}L}{\mathrm{dbsdf}} = \nabla f_0 \frac{L}{f_0}$$

$$\frac{\mathrm{d}L}{\mathrm{d}\omega'} = ?$$

First step: compute $\frac{\mathrm{d}L}{\mathrm{d}\omega_0}$

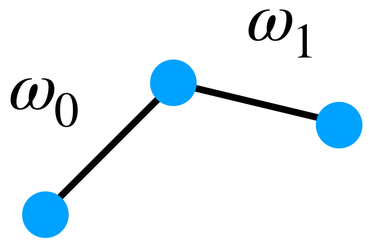
using forward mode on the path tracing loop




```
L = 0
throughput = 0
ω_0 = camera_ray
for i in range(1, max_depth):
  ω_i = sample_bsdf(ω_{i-1}, u, p)
  throughput *= bsdf(ω_i, ω_{i-1}, p)
  L += throughput * emission
return L
```

```
for i in range(1, max_depth):
  _, _, dL/dp = Dsample_bsdf(ω_{i-1}, u, p, dL/dω_i)
  ∇L += dL/dp
  dL/dp = Dbsdf(ω_i, ω_{i-1}, p, dL/dbsdf)
  ∇L += dL/dp
return ∇L
```

We can compute the subsequent $\frac{dL}{d\omega_i}$ using chain rule & matrix inversion



$$\frac{dL}{d\omega_{i-1}} = \frac{d\omega_i}{d\omega_{i-1}} \frac{dL}{d\omega_i}$$

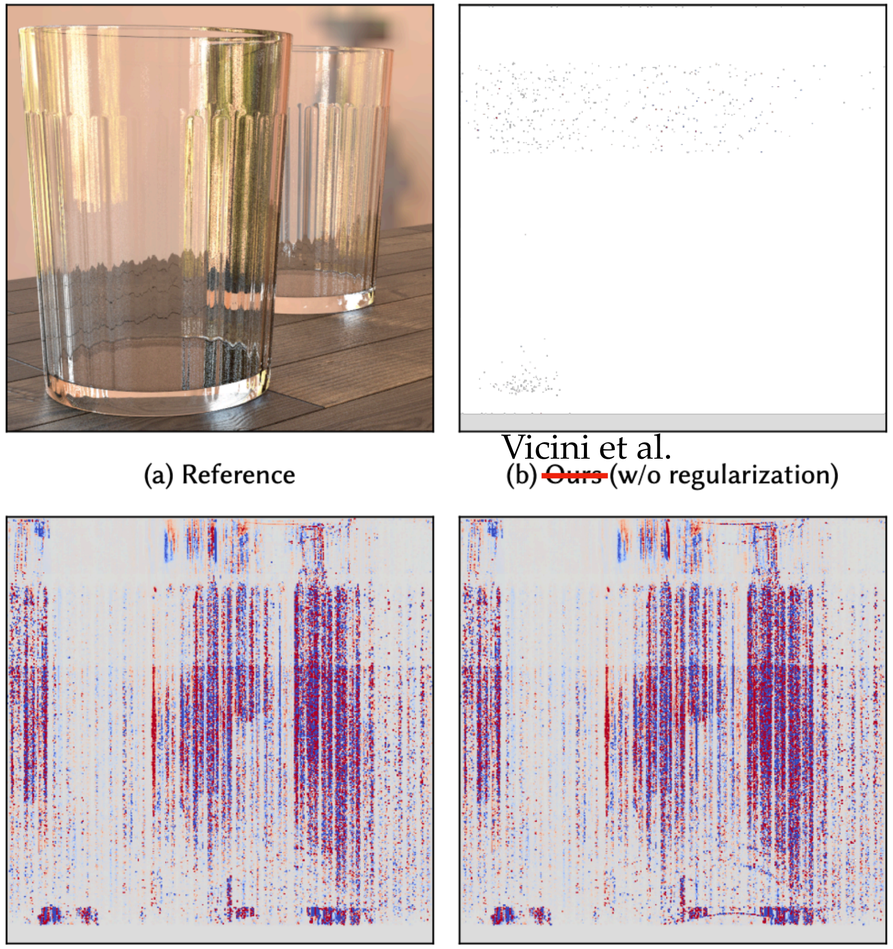
$$\left(\frac{d\omega_i}{d\omega_{i-1}}\right)^{-1} \frac{dL}{d\omega_{i-1}} = \frac{dL}{d\omega_i}$$

get $\frac{d\omega_i}{d\omega_{i-1}}$ by differentiating sample_bsdf

```
L = 0
throughput = 0
ω_0 = camera_ray
for i in range(1, max_depth):
  ω_i = sample_bsdf(ω_{i-1}, u, p)
  throughput *= bsdf(ω_i, ω_{i-1}, p)
  L += throughput * emission
return L
```

```
for i in range(1, max_depth):
  _, _, dL/dp = Dsample_bsdf(ω_{i-1}, u, p, dL/dω_i)
  ∇L += dL/dp
  dL/dp = Dbsdf(ω_i, ω_{i-1}, p, dL/dbsdf)
  ∇L += dL/dp
return ∇L
```

Detail: add noise to $\frac{\mathrm{d}\omega_i}{\mathrm{d}\omega_{i-1}}$ to avoid non-invertibility



(a) Reference

(b) ~~Ours~~ (w/o regularization)

Vicini et al.

(c) ~~Ours~~

Vicini et al.

(d) Conv. AD

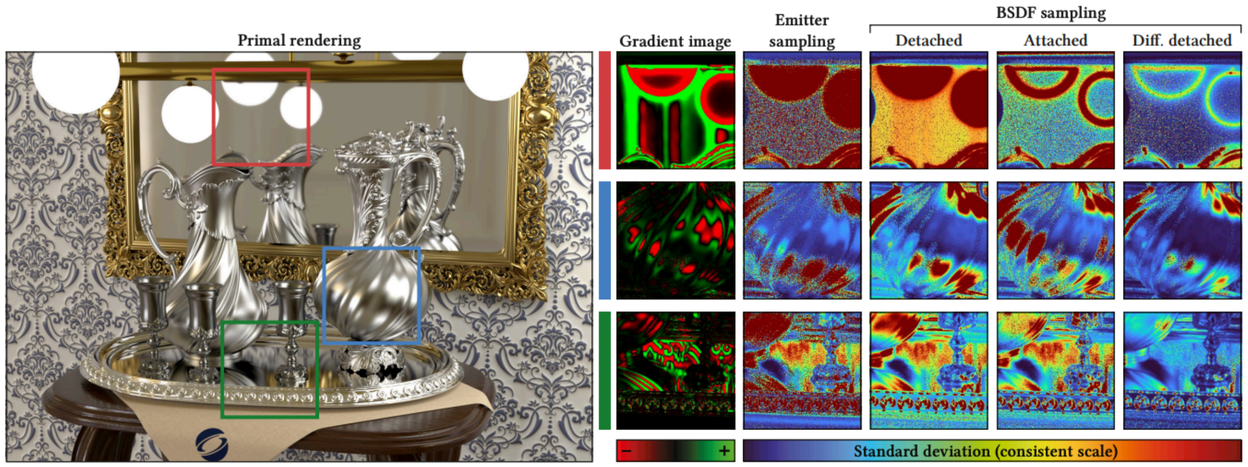
Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time

DELIO VICINI, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Further reading

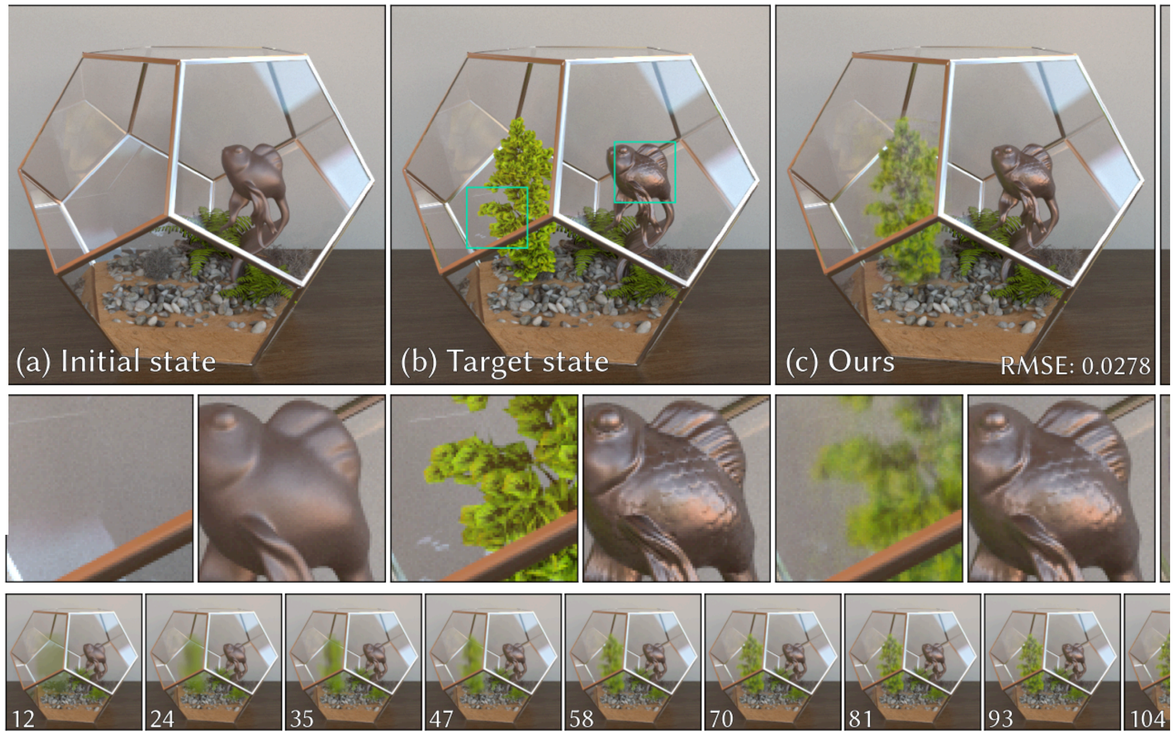
Monte Carlo Estimators for Differential Light Transport

TIZIAN ZELTNER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
ILIYAN GEORGIEV, Autodesk, United Kingdom
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland



Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time

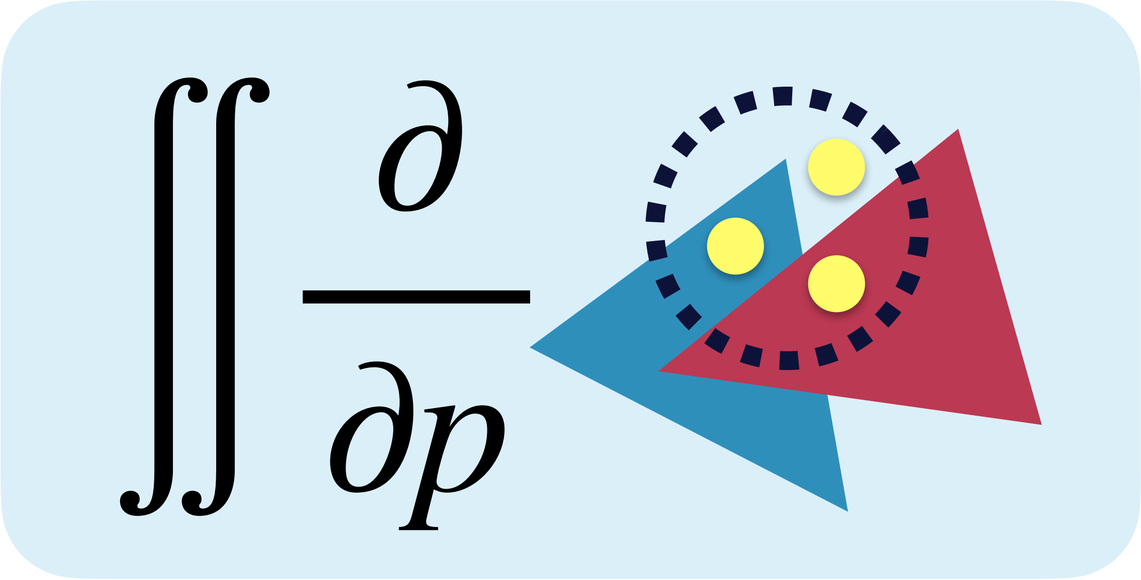
DELIO VICINI, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland



Two challenges of computing the interior derivatives

- importance sampling of forward rendering
  may not be a good strategy for differentiable rendering

- reverse-mode autodiff can lead to
  high memory consumption



Next time: stratification

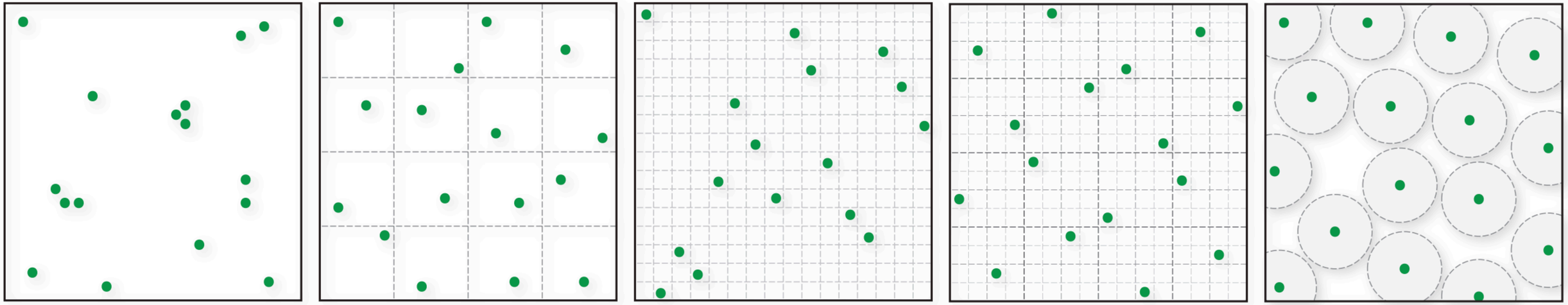


figure courtesy of Wojciech Jarosz

https://cs.dartmouth.edu/~wjarosz/publications/subr16fourier-slides-2-patterns.pdf