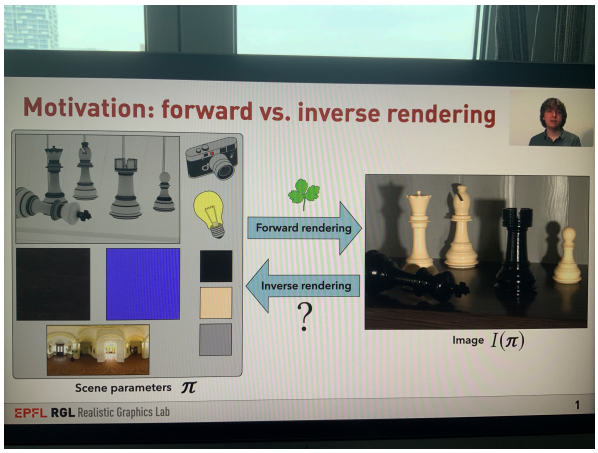
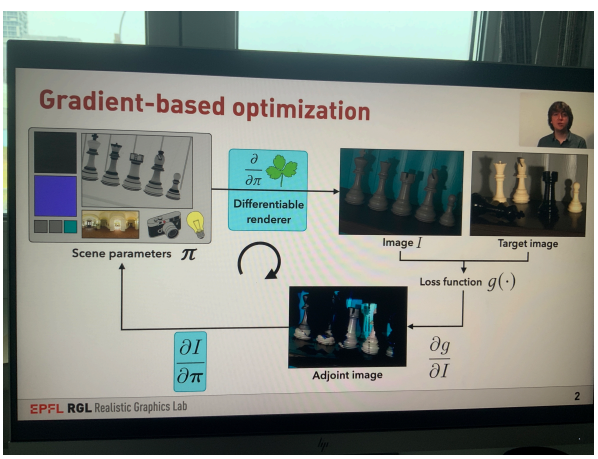


1. Initial guess for params to render
2. Use loss function $g(x)$ to compared against reference we want to achieve
   - Use gradient descent to update the inputs

1. Need: Gradient of the loss $\frac{\partial g}{\partial I}$ respect to all pixels
   (how much pixels are effected by the change of the loss)

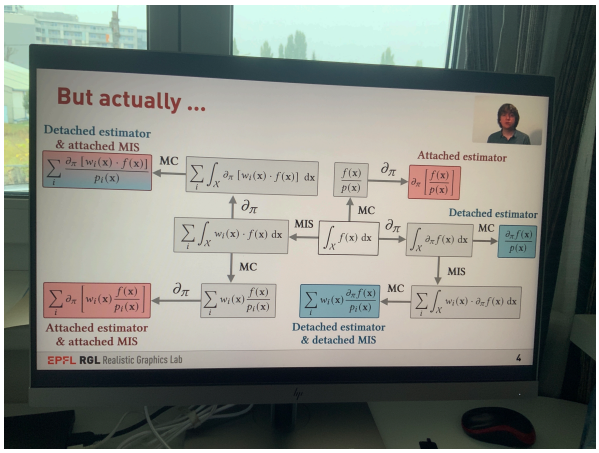
2. Need: Gradient of the image pixels $\frac{\partial I}{\partial \pi}$
   w/ respect to scene params



Main challenge:
realizing the 2. step and it
req. we can differentiate w/ the
complete rendering process

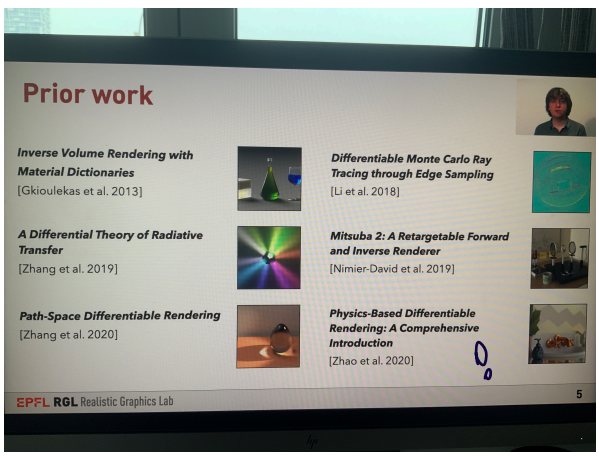
One assumes: this is easy (i.e. $\frac{\partial}{\partial \pi}$)
because we have great tools
to automatically differentiate

But: differentiating rendering code requires more specifics
We have integrals that we solve w/: Monte Carlo, MIS



By adding differentiation:
we get a large # of
← approaches, that all have different
trade offs.

Paper tries to understand space of
possibilities.

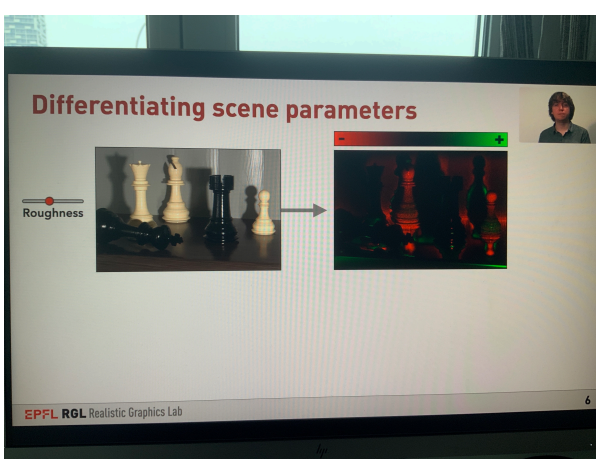


Refs

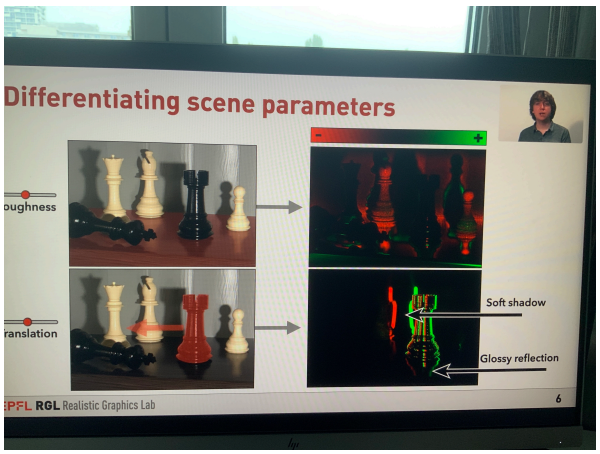
To get the intuition behind the gradient we want to compute
we can look at gradient imgs.

Gradient image shows:
- it encodes img. pixels change due to infinitisimal change of the
single input param

e.g. surface roughness:



e.g. translation: causes a global change, due to shadows and reflection



Gradient images

⊕ Visualization

⊖ But not really useful for optimization task

Because: Result of forward mode differentiation

We track only single input param all the way to the image pixels.

Arguably: If gradient image needed: something simpler finite differences

But we care many scene params at once $(\pi_1, \dots, \pi_n)$
e.g. high res. textures or mesh vertices

Running forward diff. once for each input wouldn't scale

Instead:

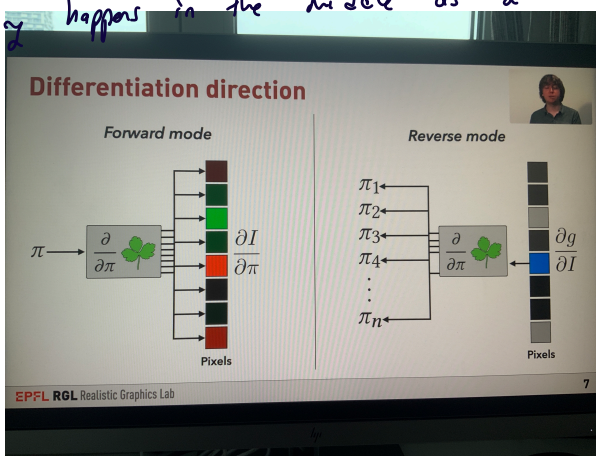
① Use gradient descent of the loss respect to each pixel

② Use reverse mode differentiation to work our way backwards for scene params

(For neural networks = Back propogation)

⊕ key for scalability

⊖ usually requires storing everything happens in the middle as a computation graph



But for rendering: the computation graphs becomes huge
- memory

Ludily: Memory issue addressed:

→ replaces standard reverse mode diff w/ adjoint rendering pass

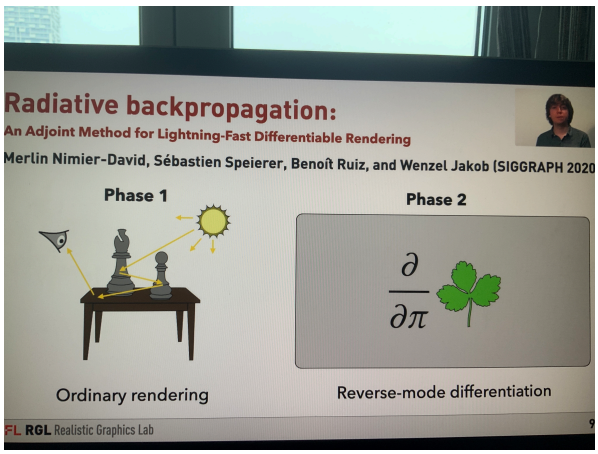
→ makes diff. light transform more scalable

- ignores discontinuities everywhere in the integrals we solve, mostly: visibility

problematic wherever position depends on some diff. scene param

Intuitively: if we estimate the integral using the discrete monte carlo sampling, the $f(x)$ at the points will be non diffable due to the geometry change

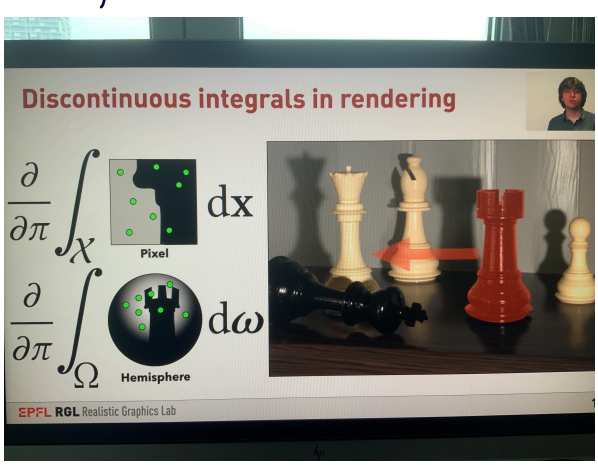
⇒ computed gradients will be wrong

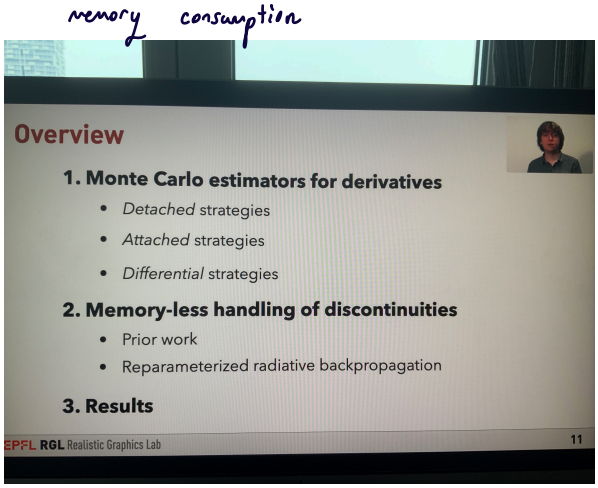


2. contribution from this paper:

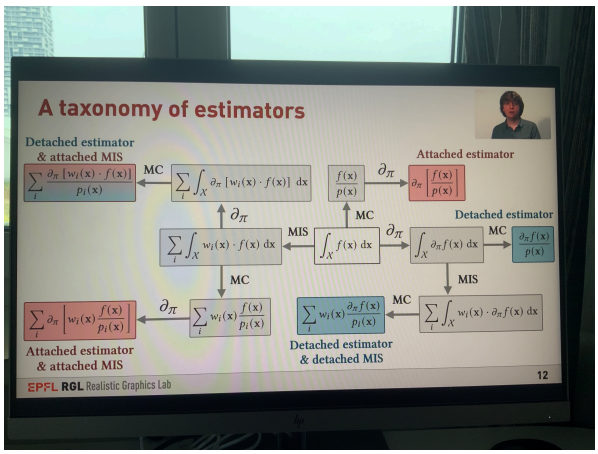
- extension to radiative backpropagation to support this case

→ this extension scale well w/o large memory consumption





1. How to use monte carlo to estimate derivatives of integrals
(initially assume: discontinuities not problematic)



$$\int_X f(x, \pi)\, dx$$

x: integration variable, π: extra param

Two things necessary:
- Differentiate respect to π
- Solve integral w/ Monte Carlo

} they do not commute

⇙ two alterings operations

Either:

1) differentiate the integral
2) apply monte carlo to solve

Doesn't matter: that the integrand happens to be a derivative we get a standard expression of estimator

Analogous to (alternative):
⇒ detached estimator
As it produces STATIC samples

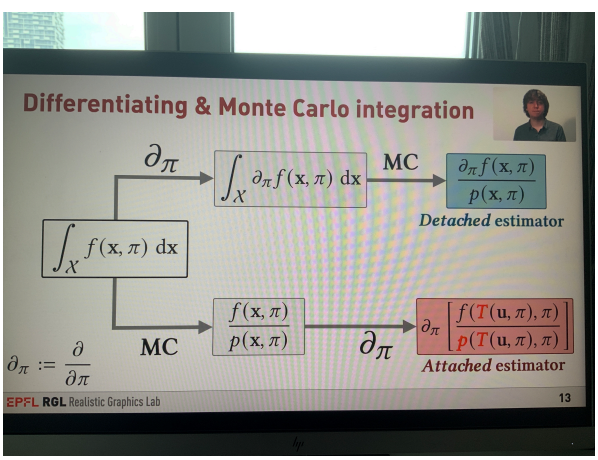
Alternatively:

1) apply monte carlo to solve
2) differentiate the estimator

It also what happens when auto diff-in existing code
→ More involved:
also differentiates MC process itself
Concretely:
- sampling routine that takes env. numbers and
- division by the prob. density

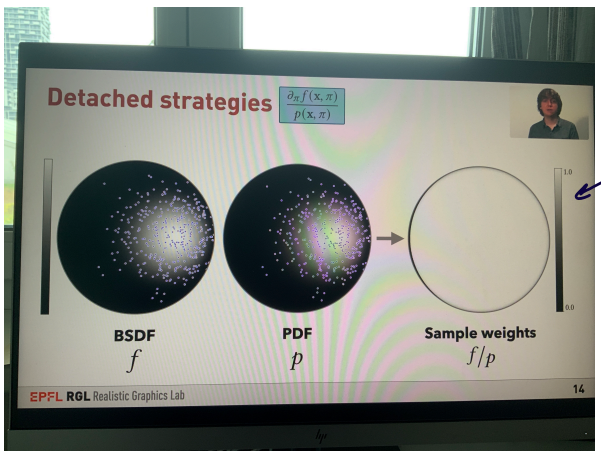
both depends on diff. params too

⇒ Generated samples will move w/ infinitisimal changes of inputs
⇒ attached estimator



Detached Strategies

sampling from a glossy microfacet BSDF (plot $f(x, \pi)$ over hemisphere



MC samples drawn from those have sample weights almost constant

$\Rightarrow$ sufficiently few samples to perform the integration

However: things change when we look at the BSDF diff. based on its surface roughness param:

Function has very different shapes w/ positive and neg. regions

Using ordinary sampling density

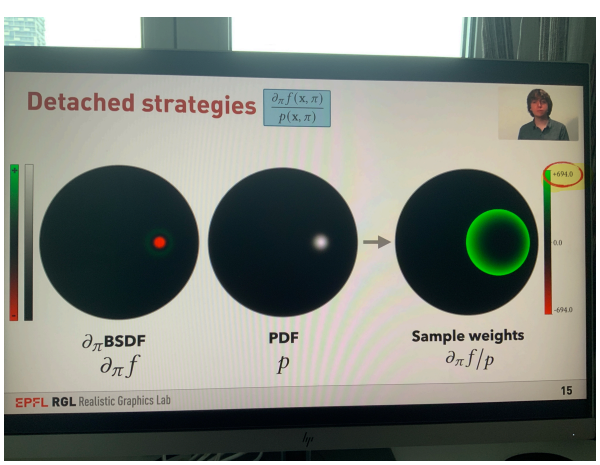
$\Rightarrow$ clearly not suited anymore

$\ominus$ undersamples some regions of BSDF derivatives

$\Rightarrow$ high sample rates

$\Rightarrow$ high variance

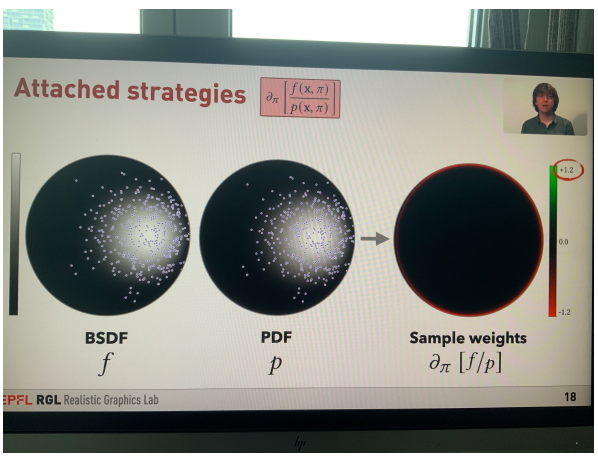
$\Rightarrow$ more drastic w/ more specular surfaces

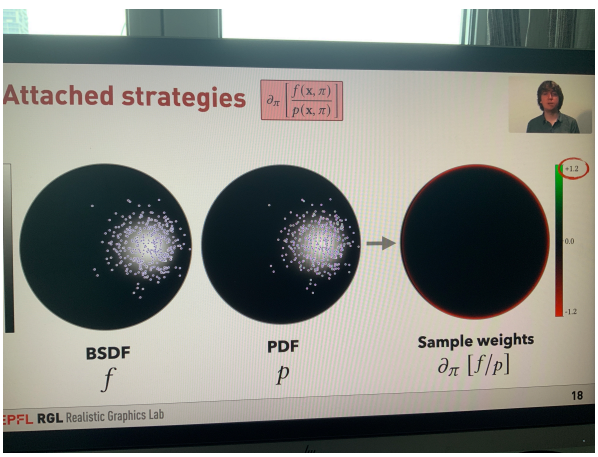


Attached Strategies:

Same example as above, we now apply the differentiation to the sample rates at the end

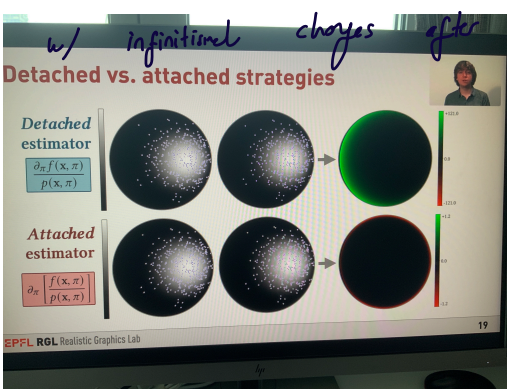
⊕ weights stod much more constant
⇒ good sampling strategy for the ordinary integrand is usually still reasonable for the differentiated case

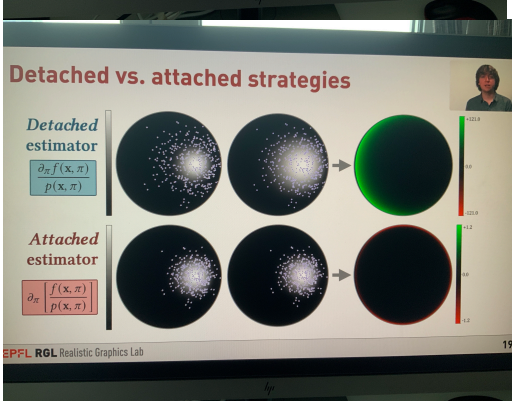




More specular

This is exactly because samples now follow changes of the BSDF w/ infinitisimal changes after roughness





Detached:

BSDF changes in the same differential way, but the sample pos. still won't adapt

Drawbacks of Attached strategy:

In rendering apps product integrals are important, e.g.

BSDF · Incident radiance // computing reflectance at a point

f_r L_i

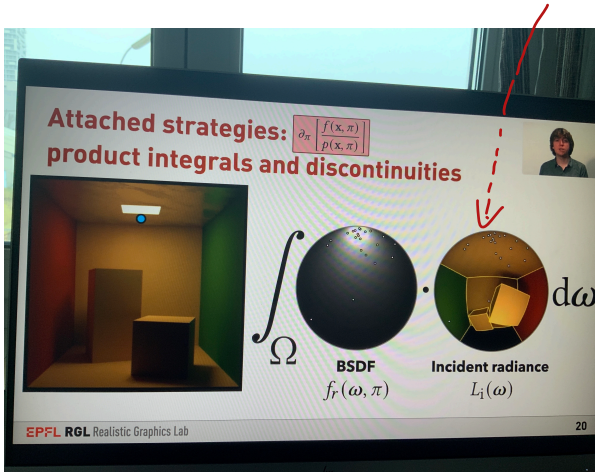
Attached BSDF sampling:

- More w/ changes of the surface roughness => ⊕ BSDF coeff.

⊖ But: Additional complication for the 2nd term (i.e. L_i)

- Variance might be high, because moment introduces directional derivatives of the L_i term
- Samples cross these discontinuities caused by the scene geometry

=> naively apply attached estimators to the problem gives incorrect results



Summary:

Attached strategies can outperform detached strategies, but they come w/ additional problems (can be avoided, later)

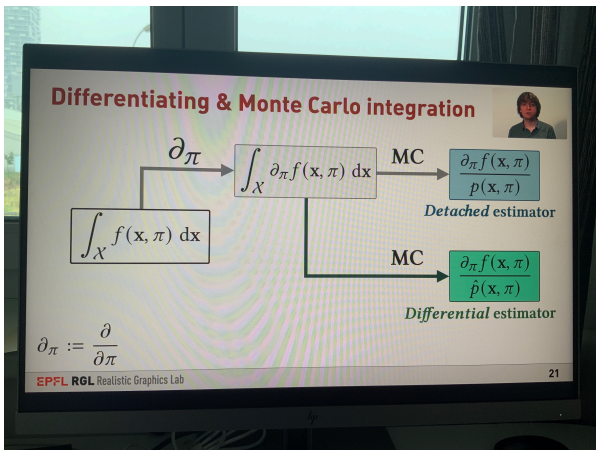
Detached:

⊖ Potential mismatch of function derivatives and pdf

⊕ Conceptually much simpler

Part of the issue:

- However that we are so fast .. existing BSDF sampling technique using ordinary rendering
- For differentiable rendering:
  - Nothing prevents us from doing detached sampling w/ different density that is better suited for estimating gradients
    → Differential estimator



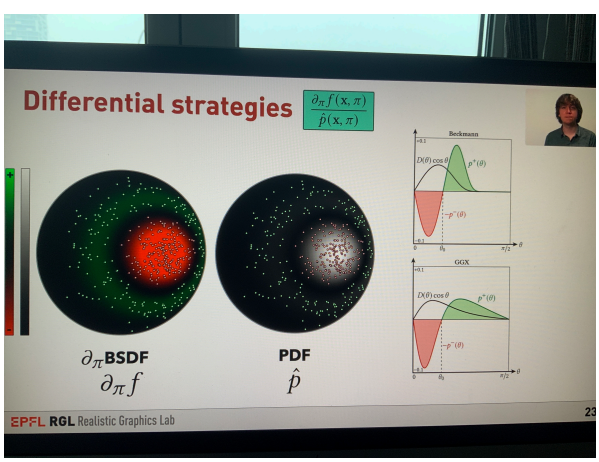
Example: Glossy BSDF

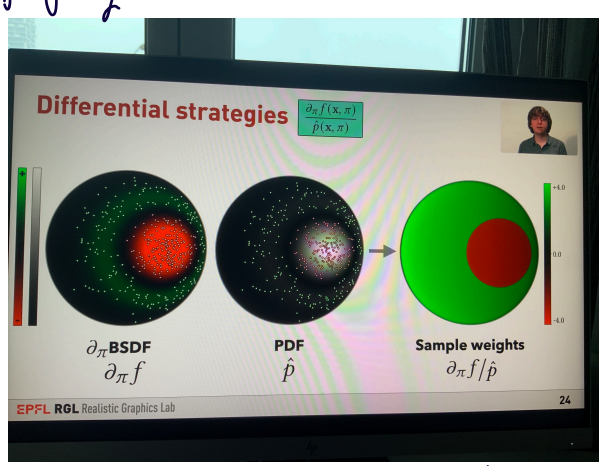
Derivatives are very different from the ordinary sampling density

It would be much better to use density that is roughly proportional

Paper shows:

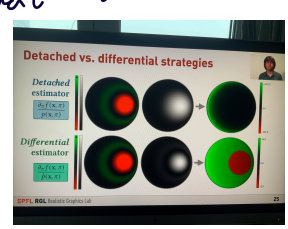
how to sample from such a distribution for the class microfacet models:
(specifically Beckmann, GGX)

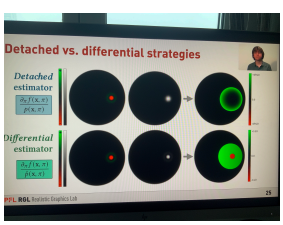


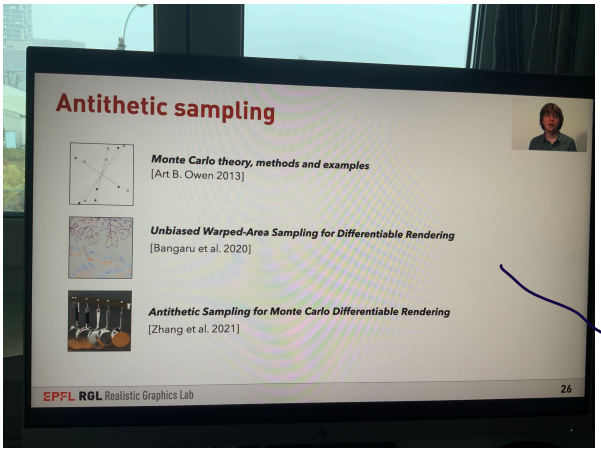


still produces signed sampled weights but they are almost constant :)

(+) improvement vs. detached case across all roughness levels



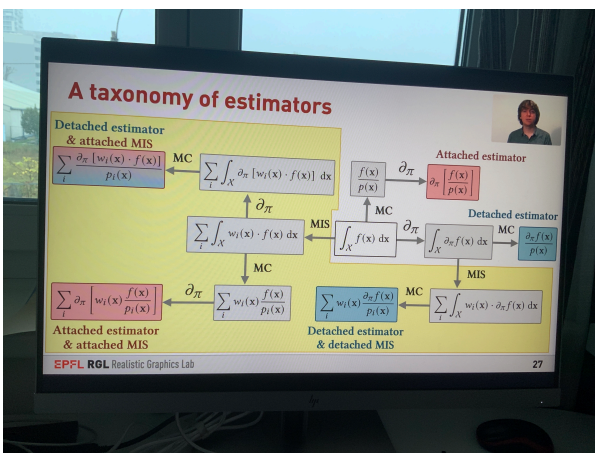


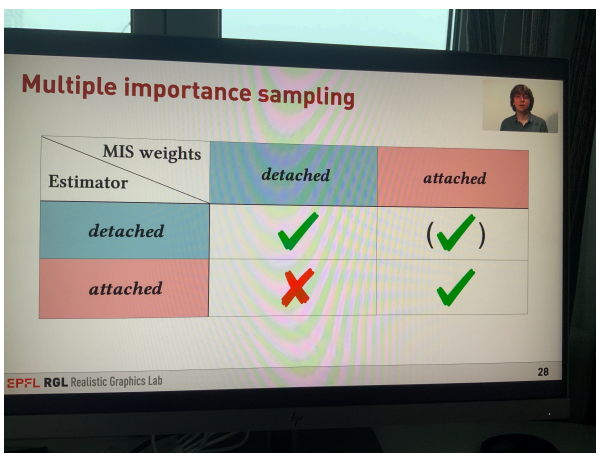


For getting low variance of signed integrals:
- it isn't sufficient to sample proportionally

→ Address this: Antithetic sampling
- correlate samples between pos. and neg. regions to accomplish this

→ introduces for diff. rendering

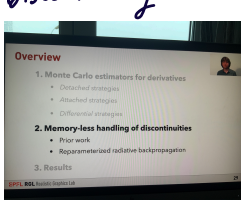




MIS: increases # of estimator
- essentially: apply attaching vs. detaching to the MIS weights

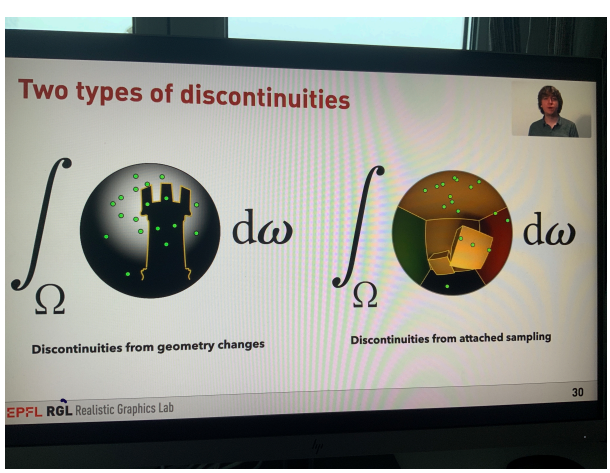
→ More in paper

Discontinuity Issue:

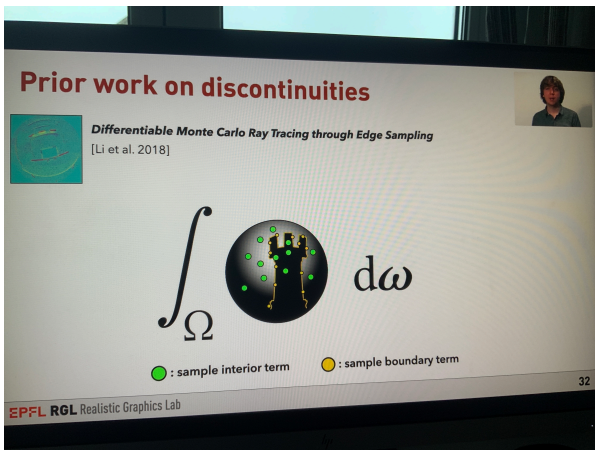


2 Situations:
- sample values are non diffable between movements

(1) either changes in geometry or,
(2) moving samples moved by attached strategies



(1)



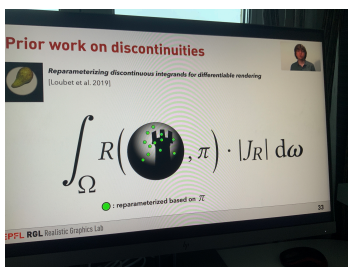
Initial solution: Edge sampling

- Discontinuity handled separately:
  → by placing samples on the edges
- ⊖ tricky for complex geometry

Alternative:

- Removing discontinuities before differentiating

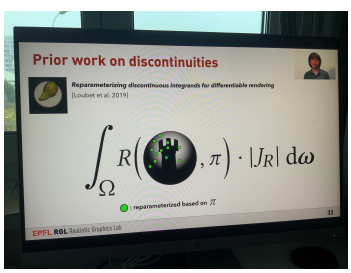
Achieved: Change of vars. (more suitable coord. system, Jacobian term necessary)



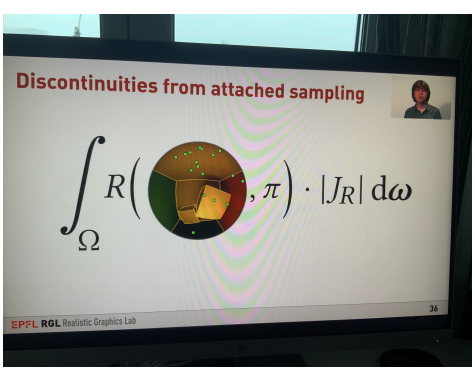
Result interpretation:

- placing samples have additional dependency on point
  ⇒ they follow discontinuities

⊖ only approx, produces bias
→ improvements: Bangaru et al. 2020



(2) No prior work



Problem turned around:

- Samples needs to stop moving (vs. detached case)

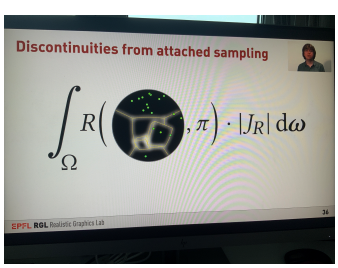
Propose:

- Change of vars

Q - Careful:

- cancelling all movement produces static samples (like detached case)

⊖ lose benefit of attaching samples Q



Instead: Proposed method slows down samples gradually as they approach the edges

⇒ moving samples, so long they don't interfere w/ discontinuities

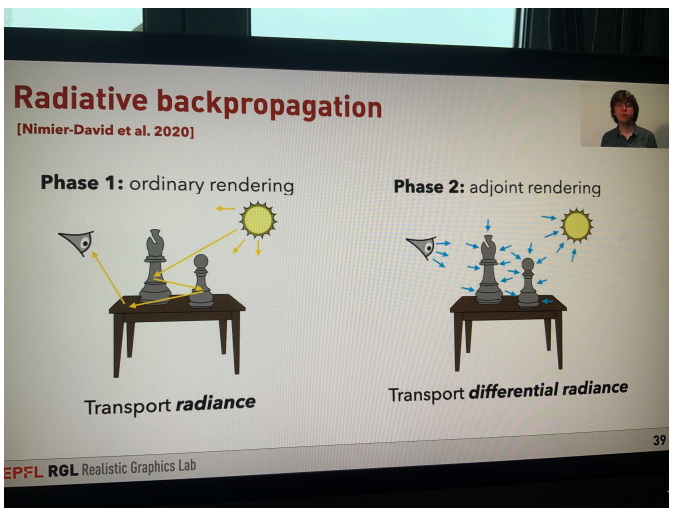
↑ Techniques to address disc. issue, but still large computation graphs

⇒ Combine shapes of vars. approach w/ radiative backpropagation + side steps mem. issue

Graph replaced w/ adjoint rendering pass but involves a new unit called differentiable radiance

⇒ behaves similarly to ordinary radiance used in rendering

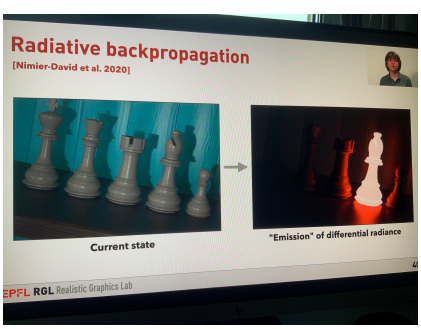
→ not emitted from light sources, instead wherever scene geom changes as a result of differentiation

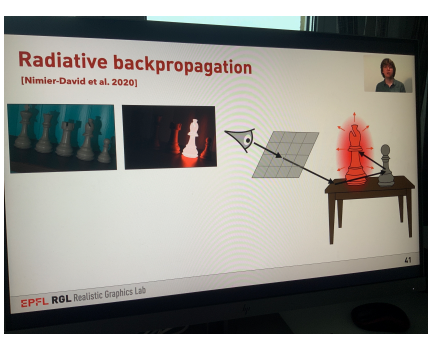


e.g. differentiating respect to color of the bishop piece means now new kind of radiation of this framework that scatters in to the scene

⇒ One can adapt existing rendering techniques like in a path tracer:
- Trace rays start from cam., through the scene until we reach something emissive

⇒ this resembles forward mode differentiation that produces gradient image

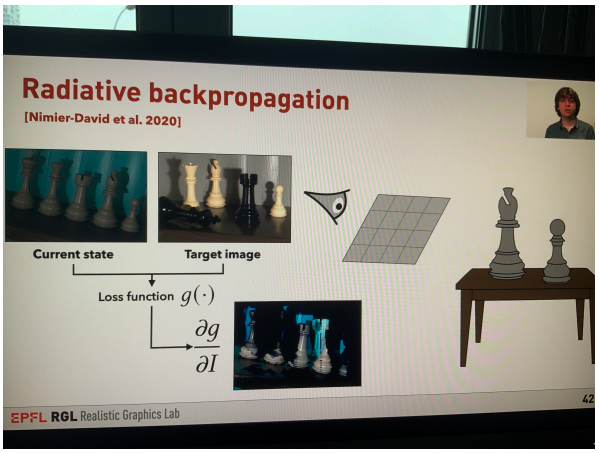




Key observation:

- like ordinary rendering this simulation is still symmetric between emission and measurement

⇒ turns the problem around



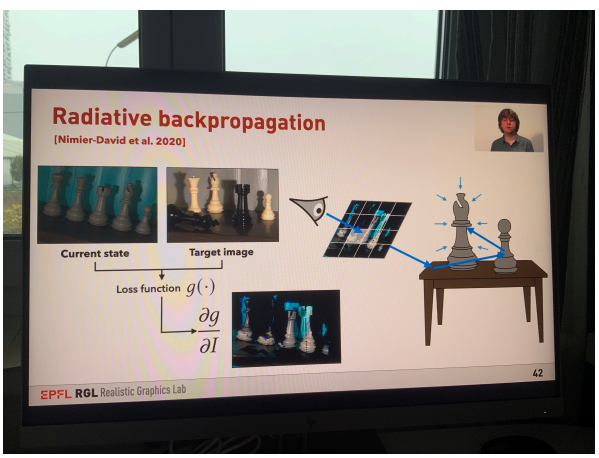
1. Render the current state forward normally
2. Backpropagate through the loss function
   - resulting image encodes how pixels should change to better satisfy the loss is emitted from the camera side

For each pixel:

trace gradients through the scene

bounce around until they are absorbed

scene params staying tracked w/

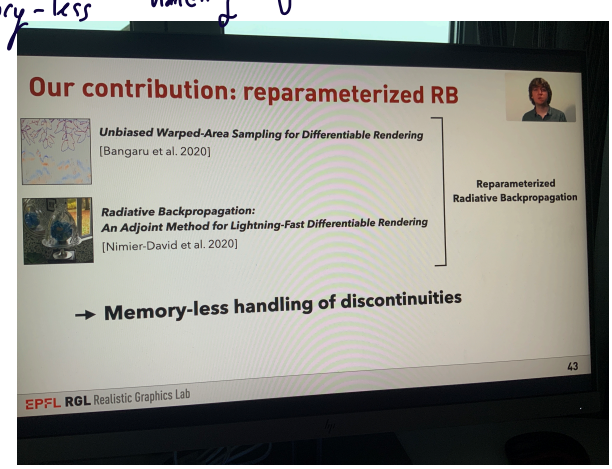


In other words:

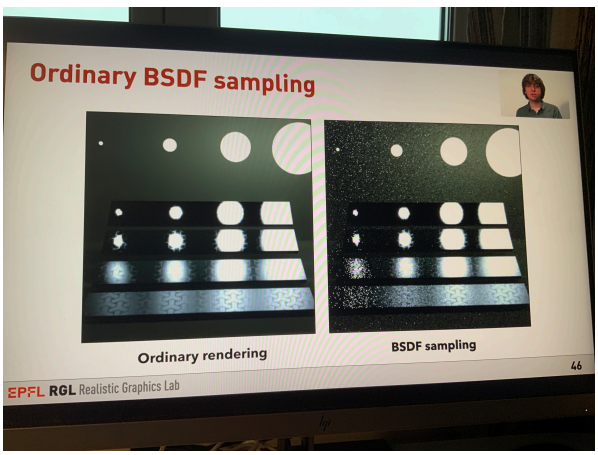
instead of backpropagating huge comp. worth do another form of rendering w/ a new unit that distributes gradients directly to all scene params at once

⇒ Paper shows: how to combine radiative backpropagation w/ changes of vars. approach

⇒ memory-less handling of discontinuities

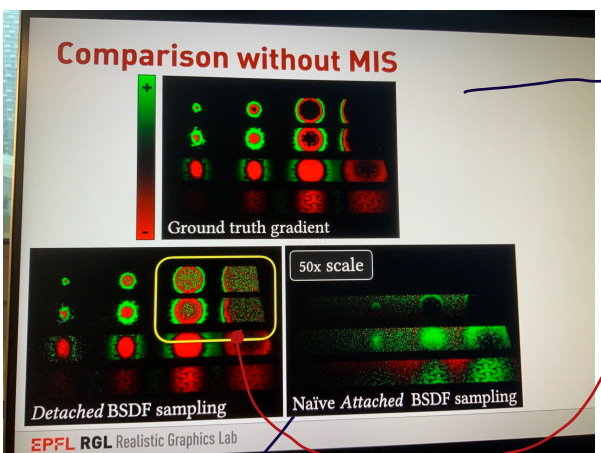


Results:



BSDF sampling:

⊕ w/ high specular surfaces and large light sources *



→ gradients after diff. roughness param used across all surfaces

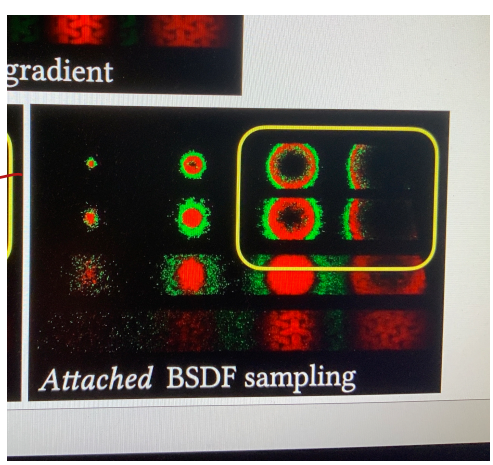
→ we can use same setup to judge the effectiveness of the gradient estimators at various roughnesses

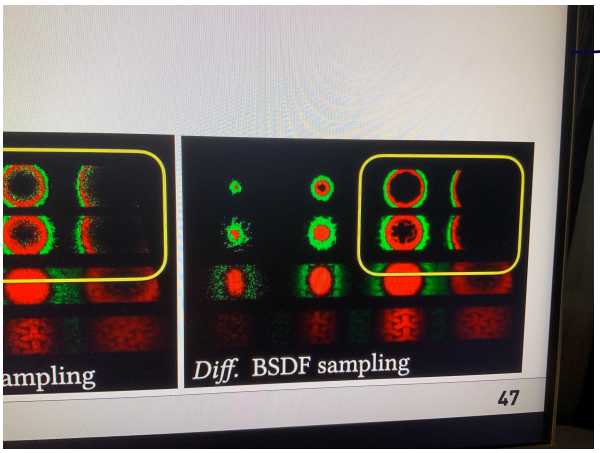
→ Because of the mismatch BSDF derivatives and the ordinary sampling pdf detached sampling works poorly in the cases it usually performs well *

w/o taking care of the additional disc. introduced by it

⇒ results wrong

w/ fix: correct gradients and vs. detached case reduction of variance





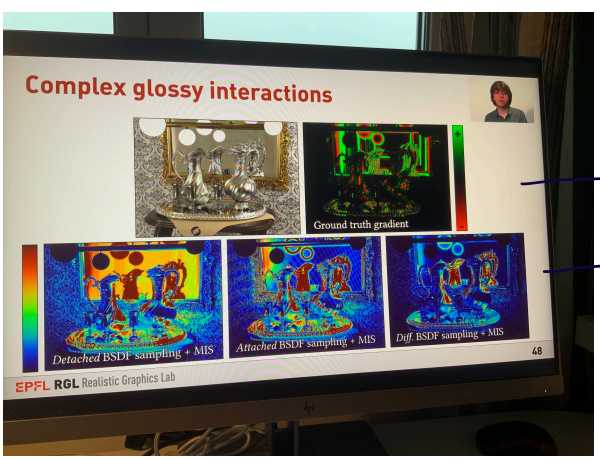
→ applying diffable estimator samples the derivs. BSDF directly

specific task: differentiating the roughness param. Gut very effective nice gradients in equal time

For completeness: → (light surface sampling!)

one can combine emitter sampling using MIS

⇒ reduces variance in those other regions where we expect light surface sampling to help



complex scene:

varying roughness and many glossy reflections

→ again differentiate under roughness param

→ visualized pixel-wise σ of the estimators

detached: ↑ variance

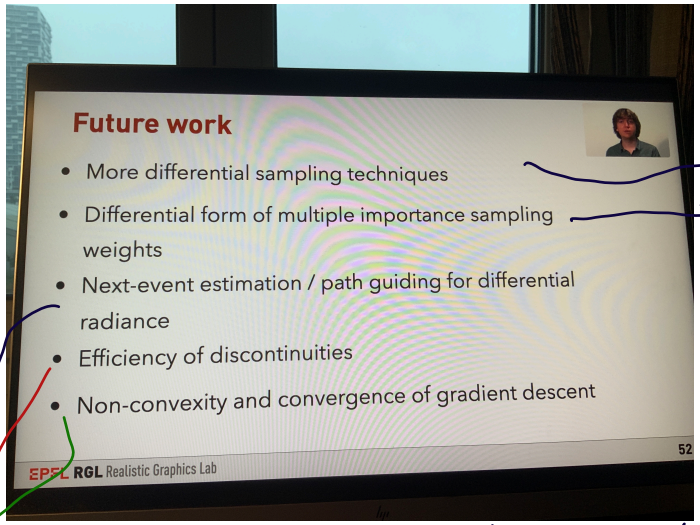
attached: better vs. detached, especially highly reflective regions (e.g. mirror)

diff.: Best overall, most variance remaining on interreflections we don't sample

Additional examples see paper/video:

2 optimization tasks w/ synthetic data:

1. estimate pos. of cam. and light source
2. Shape reconstruction: geometry of the rock



→ a lot of promising sampling techniques specifically designed w/ differentiation in mind

→ all MIS comb. in the paper still use ordinary weights based on pdfs
But, these don't say much about the variance of the differentiated estimators
So working on a truly differential form of MIS could be interesting

→ Algos still close to brute force path tracer, one could add NEE or path guiding that targets differentiates gradients specifically

→ Changes of vars. main bottleneck

→ Focus: correct gradients which is important ingredient in inverse rendering but often not enough

⇒ optimization objectives can be highly non convex
need to learn more how scene parameterization effects convergence behavior