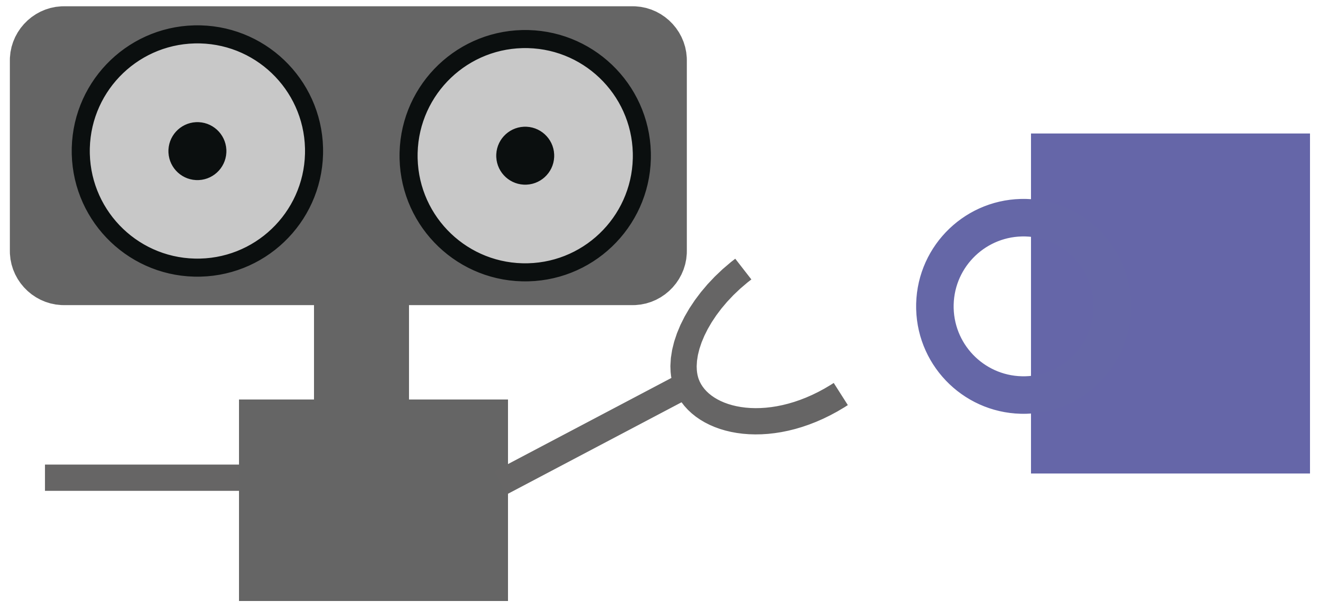
Differentiable Visual Computing

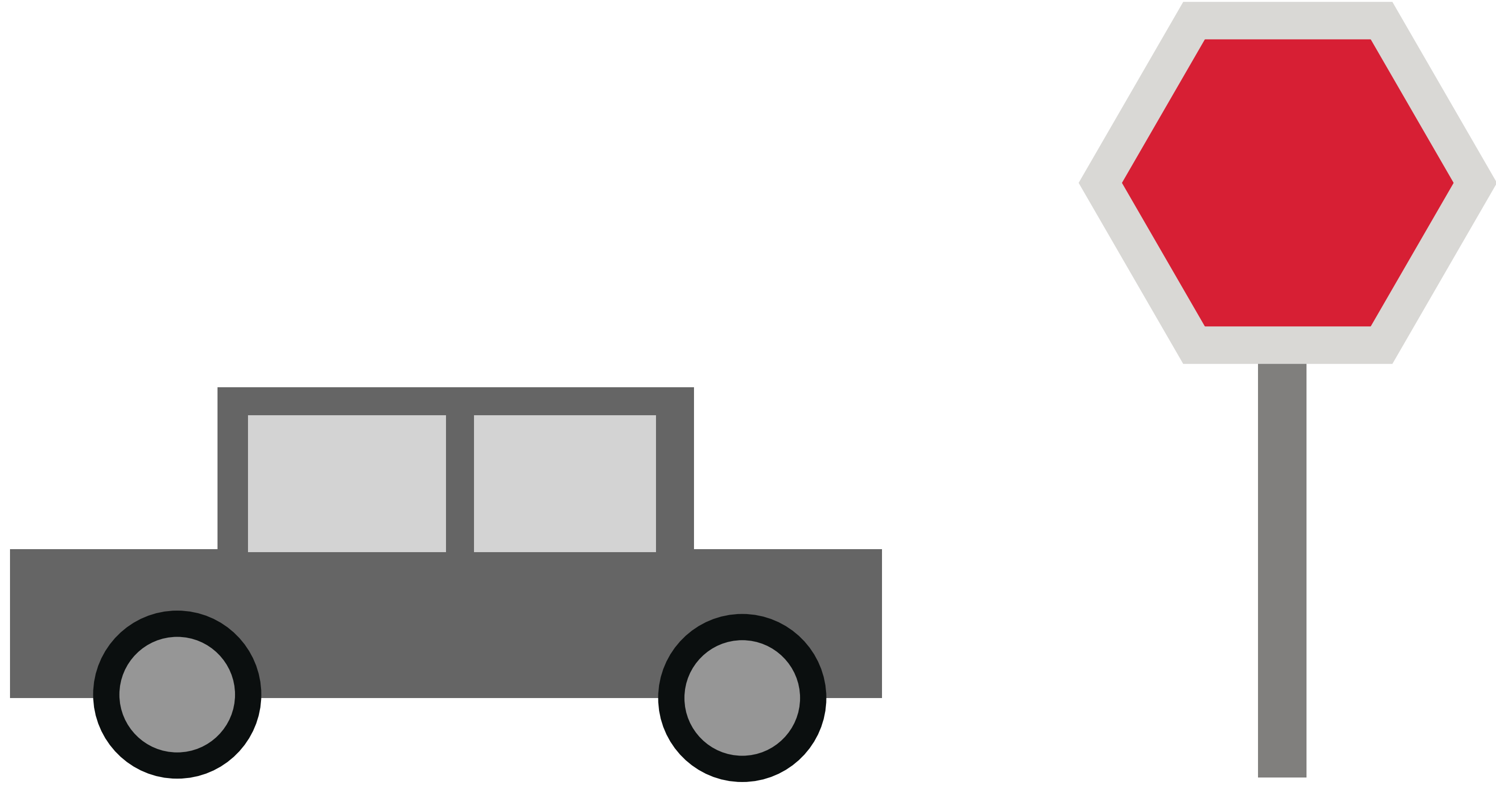
Tzu-Mao Li (李子懋)

UCSD

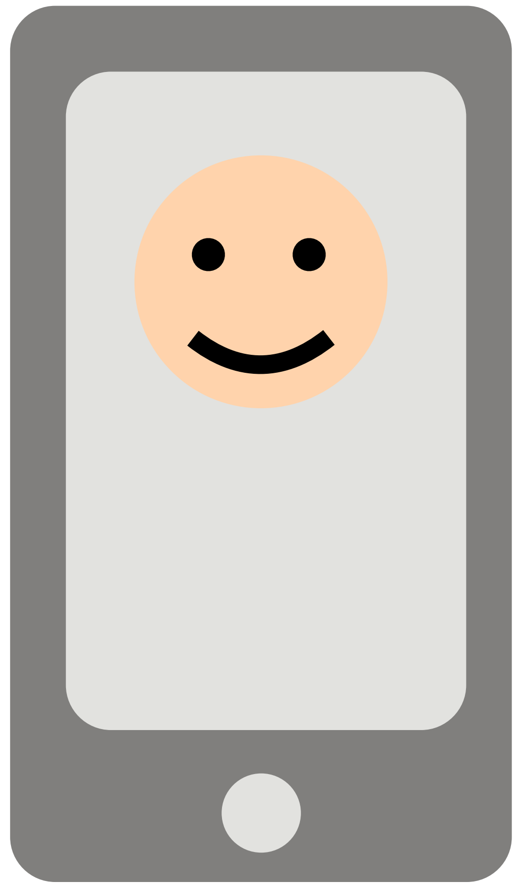
The world is visual



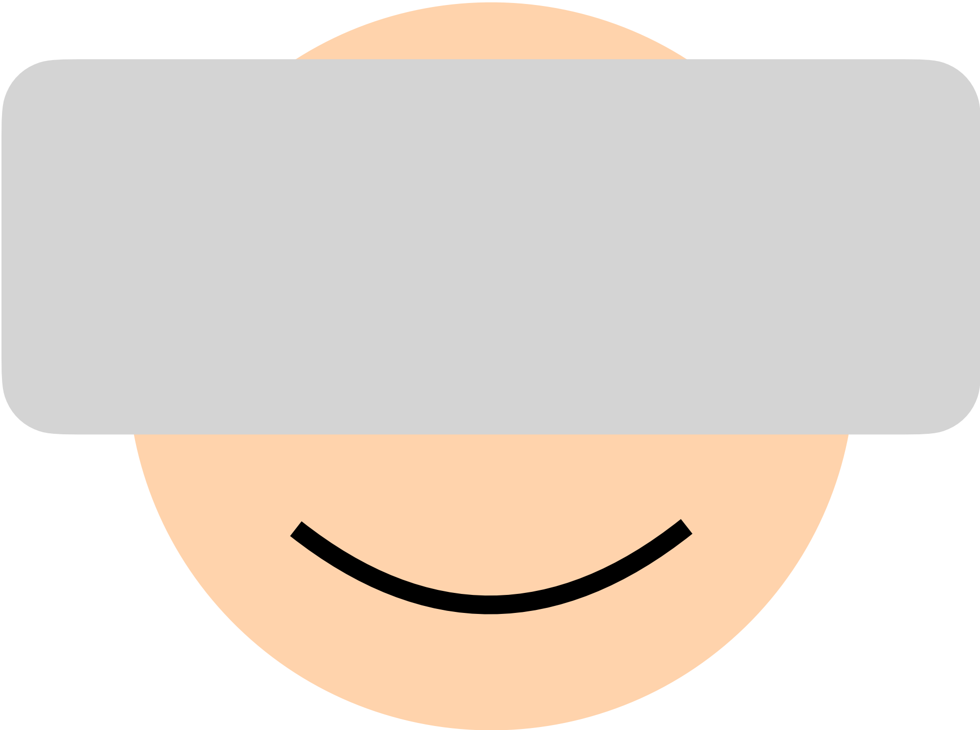
robots



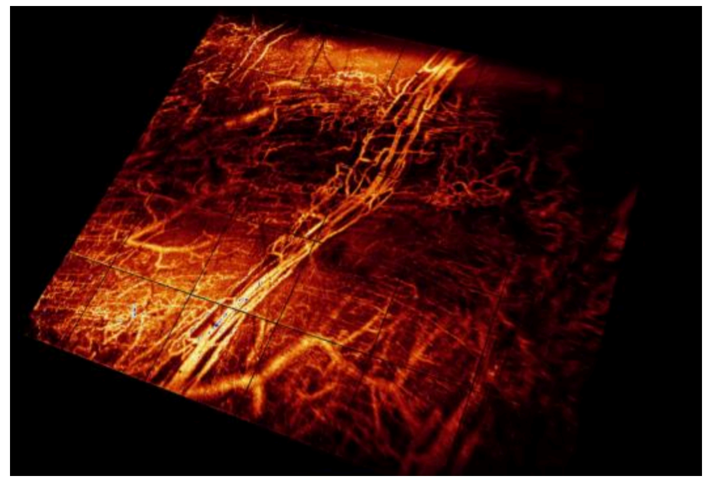
autonomous driving



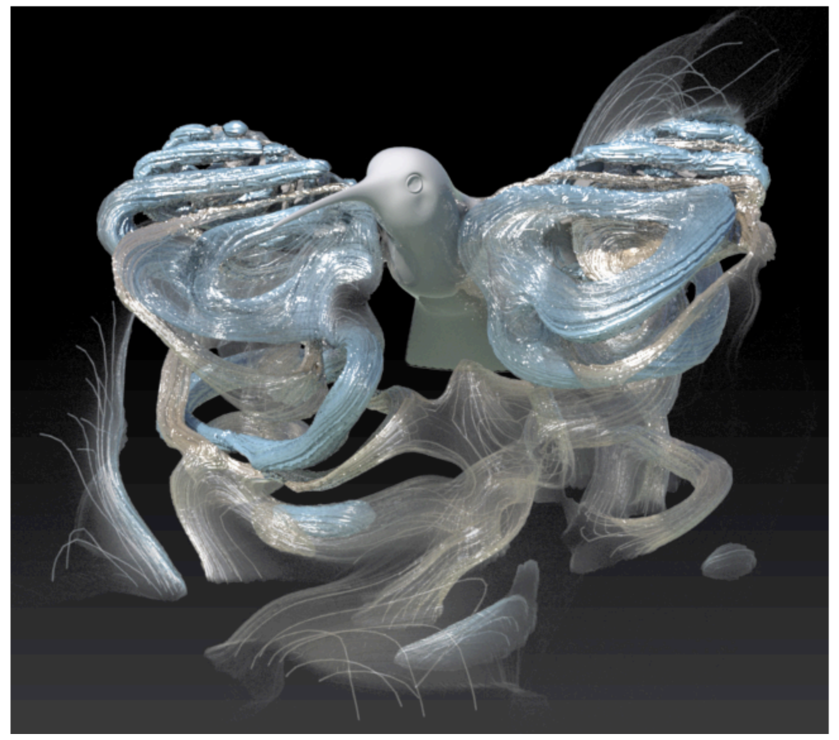
camera



virtual reality

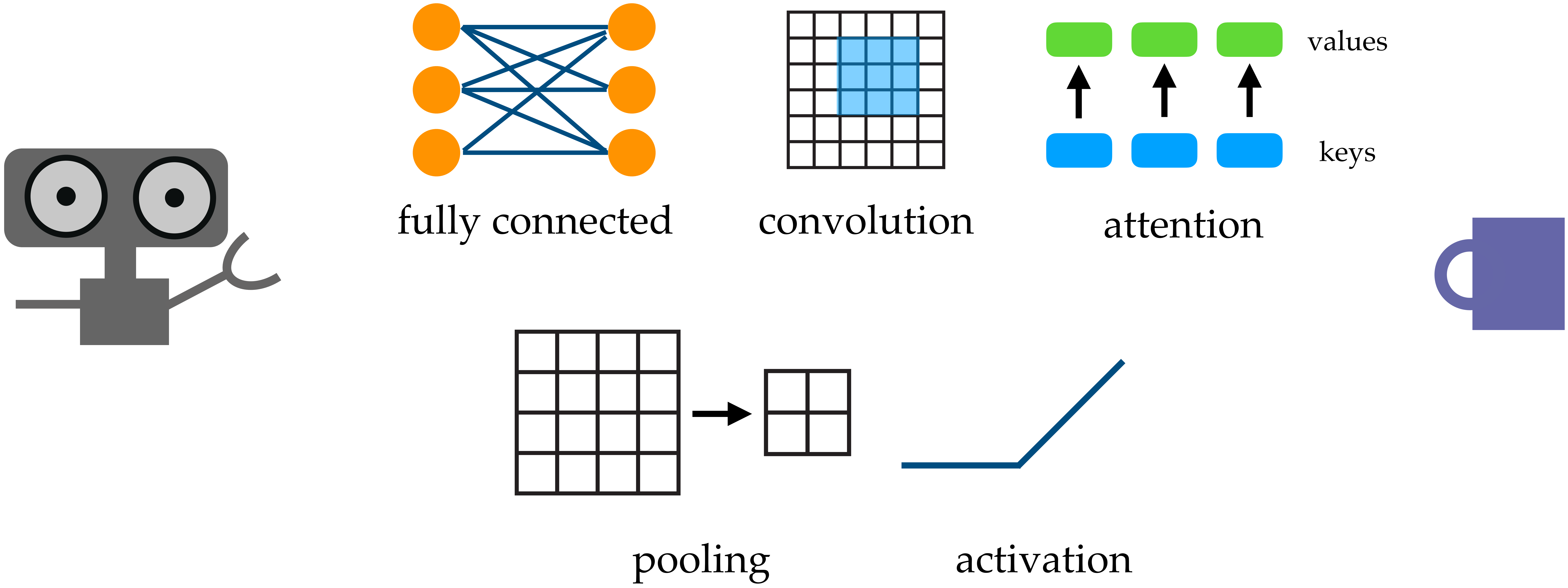


medical imaging



scientific visualization

Neural networks: powerful visual processors



Beyond neural networks

- embed prior knowledge e.g., 3D reasoning

Beyond neural networks

- embed prior knowledge e.g., 3D reasoning
- enable speed

deeplab-res101-v2 on full HD: 2.6 TFLOPs, 40GB features

Chen et al.

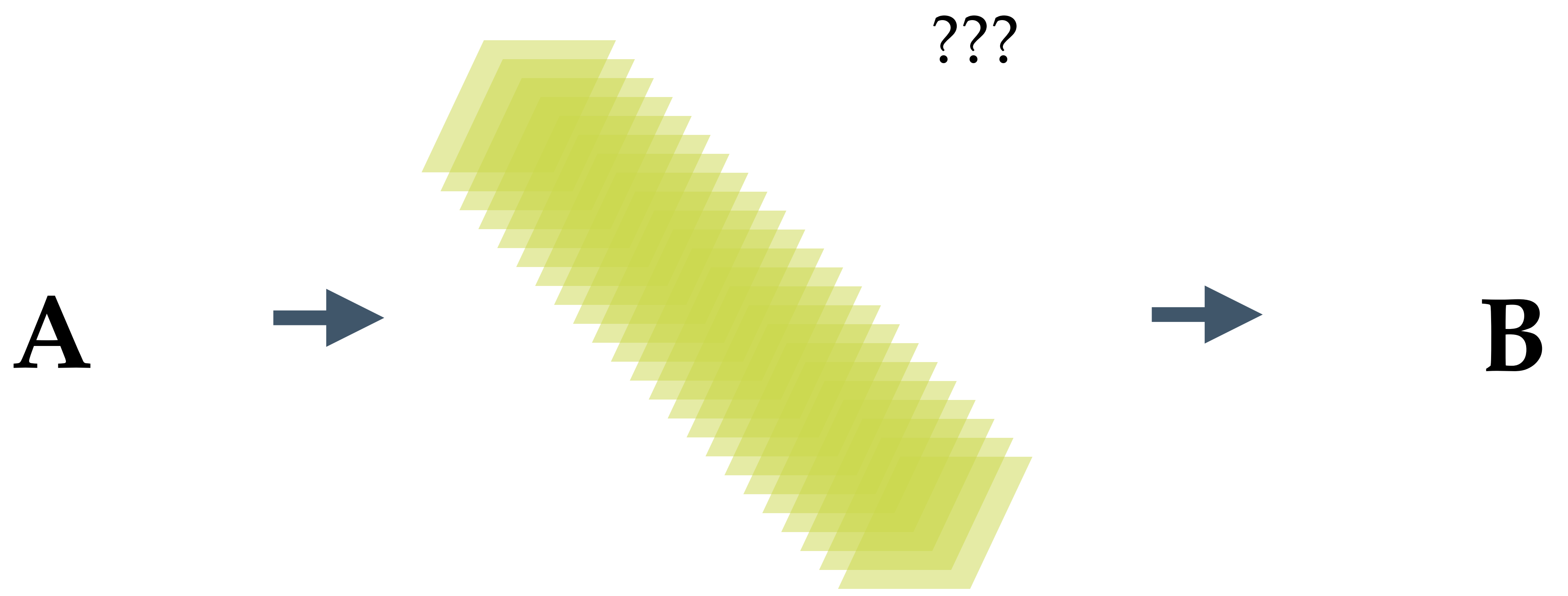
https://github.com/albanie/convnet-burden

RTX 2080 Ti
theoretical peak perf:
6.7 TFLOP/s

OpenAI recently published GPT-3, the largest language model ever trained. GPT-3 has 175 billion parameters and would require **355 years** and $4,600,000 to train - even with the lowest priced GPU cloud on the market.
Jun 3, 2020

Beyond neural networks

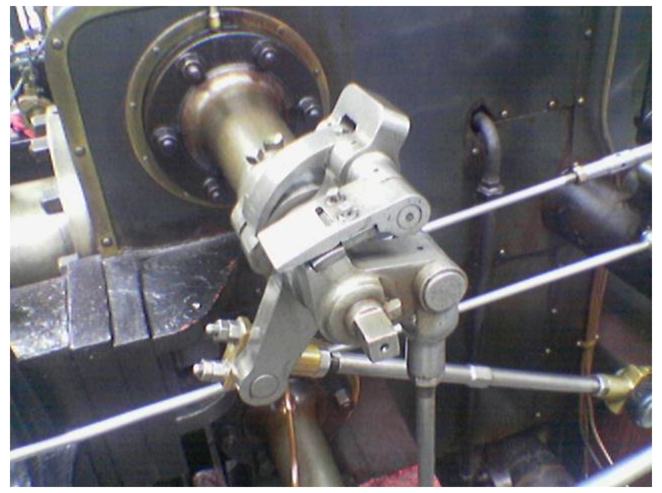
- embed prior knowledge e.g., 3D reasoning
- enable speed
- debug & control



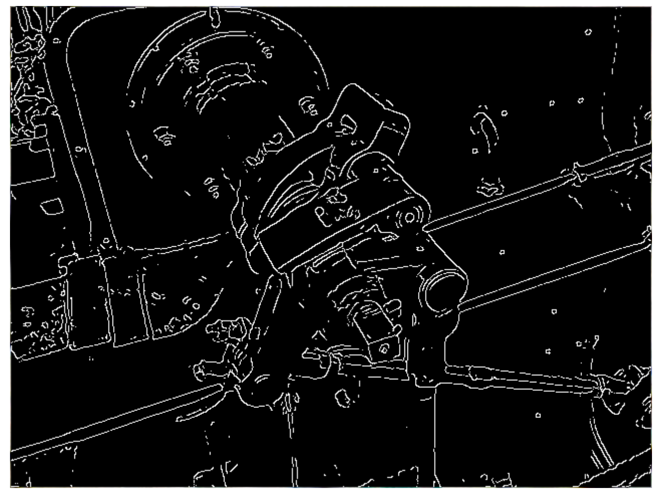
Classical methods: explicit modeling

3D scene

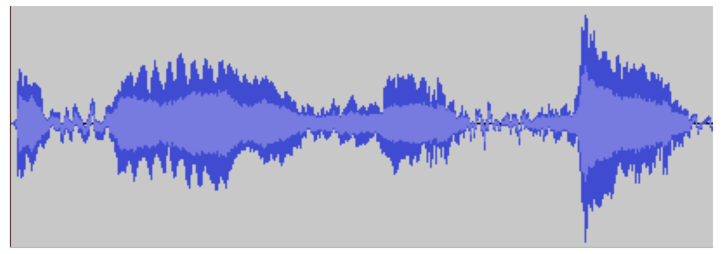
image



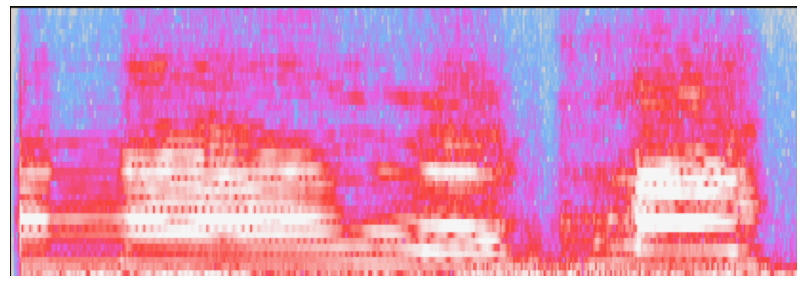
image



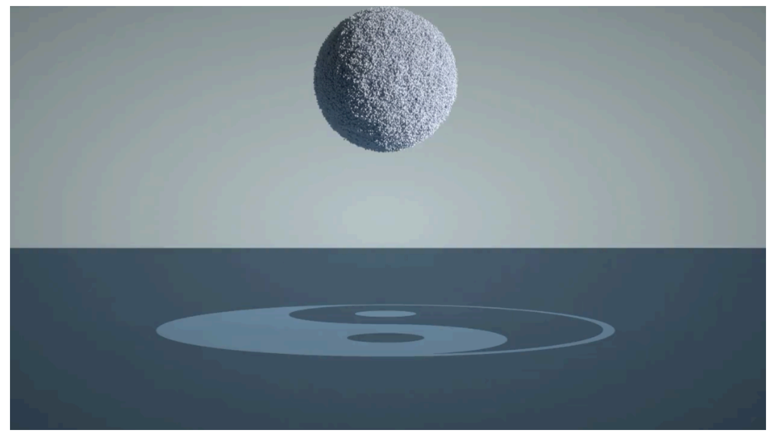
edges



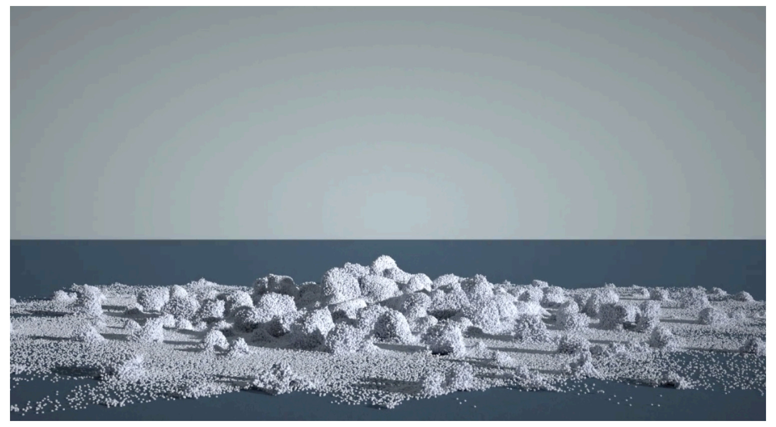
sound



spectrogram



initial



simulated

https://en.wikipedia.org/wiki/Canny_edge_detector

Classical methods: explicit modeling

✓ embedded knowledge
✓ tailored to the application
✓ debug and control

https://en.wikipedia.org/wiki/Canny_edge_detector

Classical methods: explicit modeling

✓ embedded knowledge
✓ tailored to the application
✓ debug and control

✗ do not apply as broadly
✗ do not learn from data

sound spectrogram initial simulated

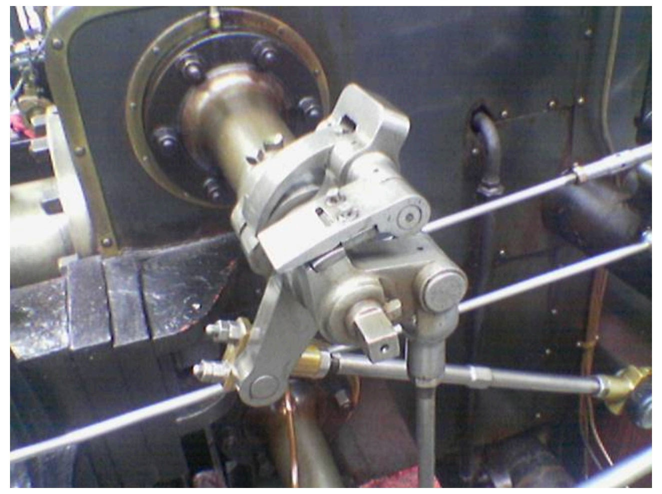
https://en.wikipedia.org/wiki/Canny_edge_detector

Key to connect classical methods and deep learning: derivatives

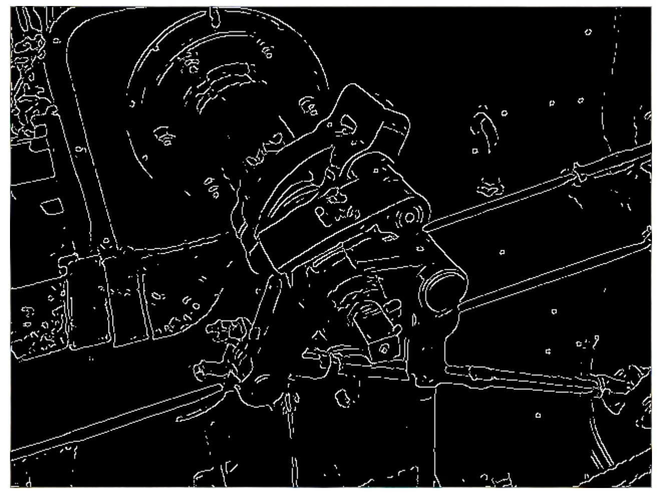
derivatives enable intelligent decisions

3D scene

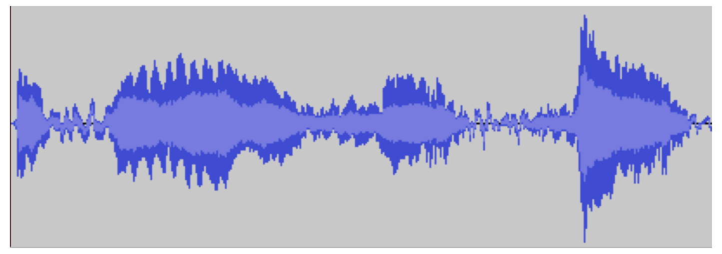
image



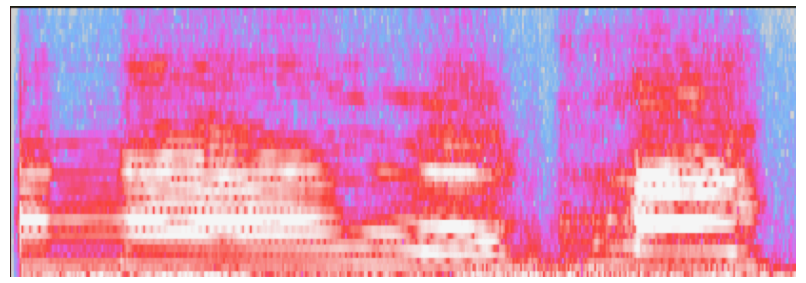
image



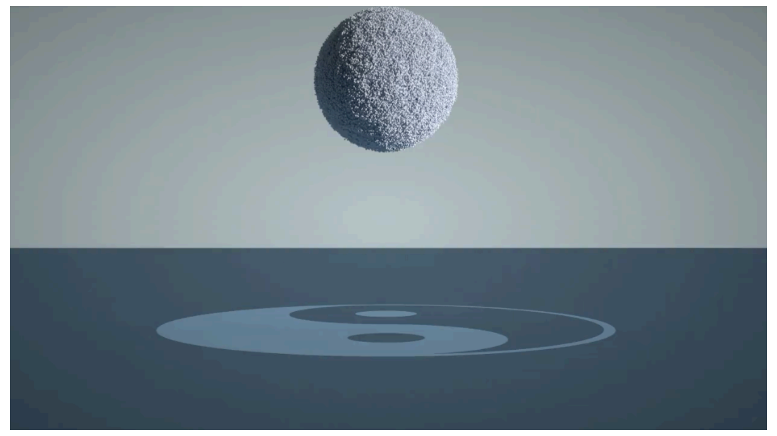
edges



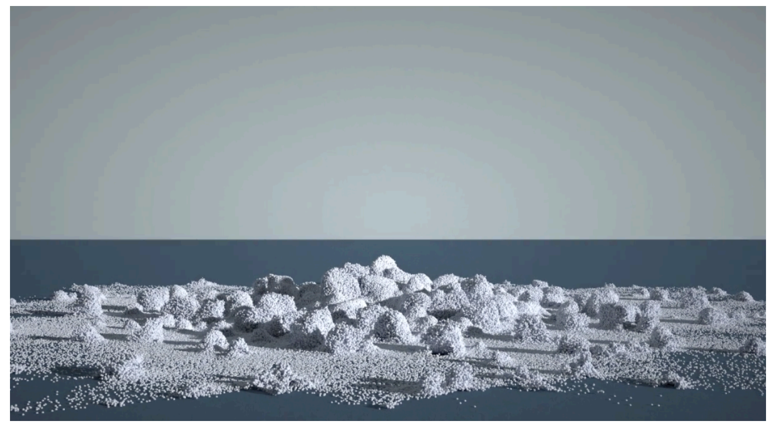
sound



spectrogram



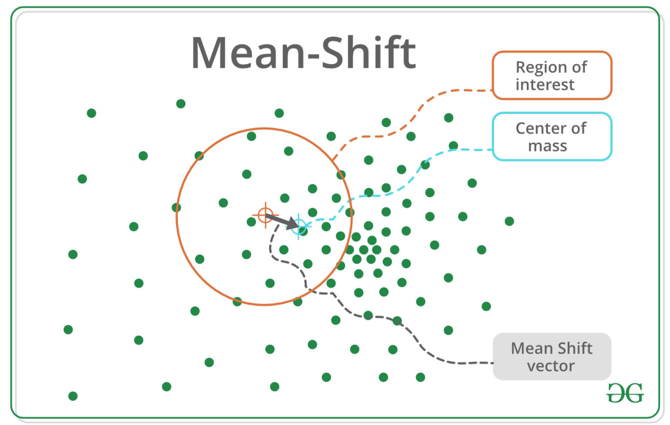
initial



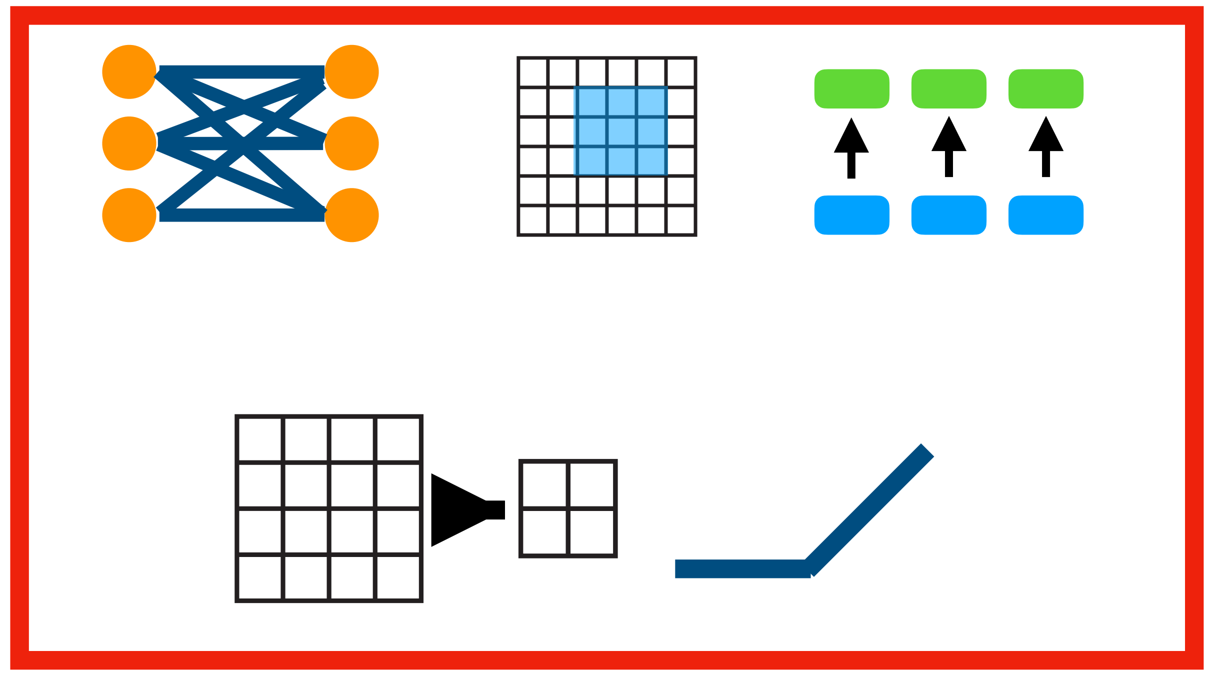
simulated

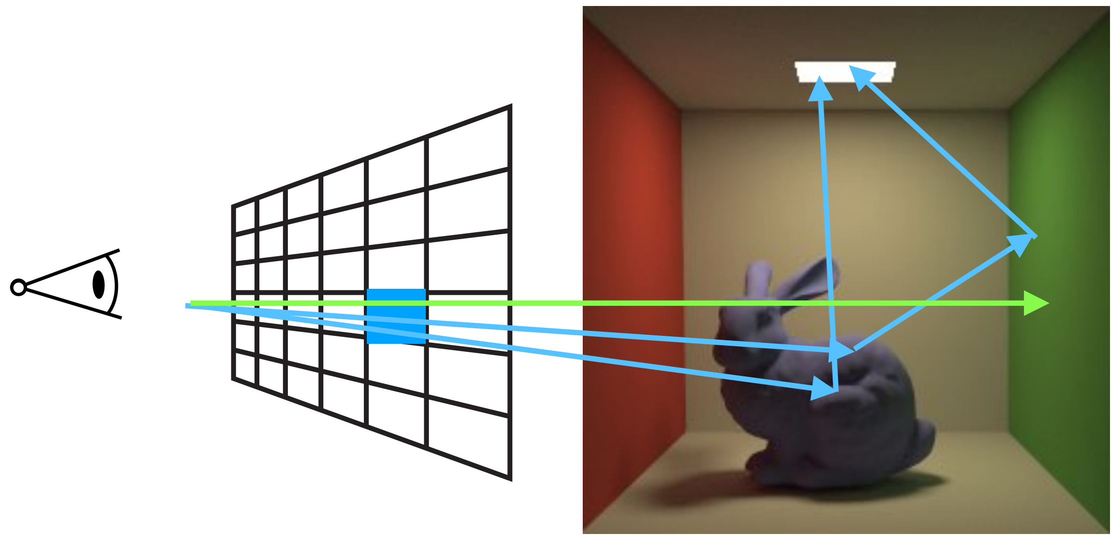
https://en.wikipedia.org/wiki/Canny_edge_detector

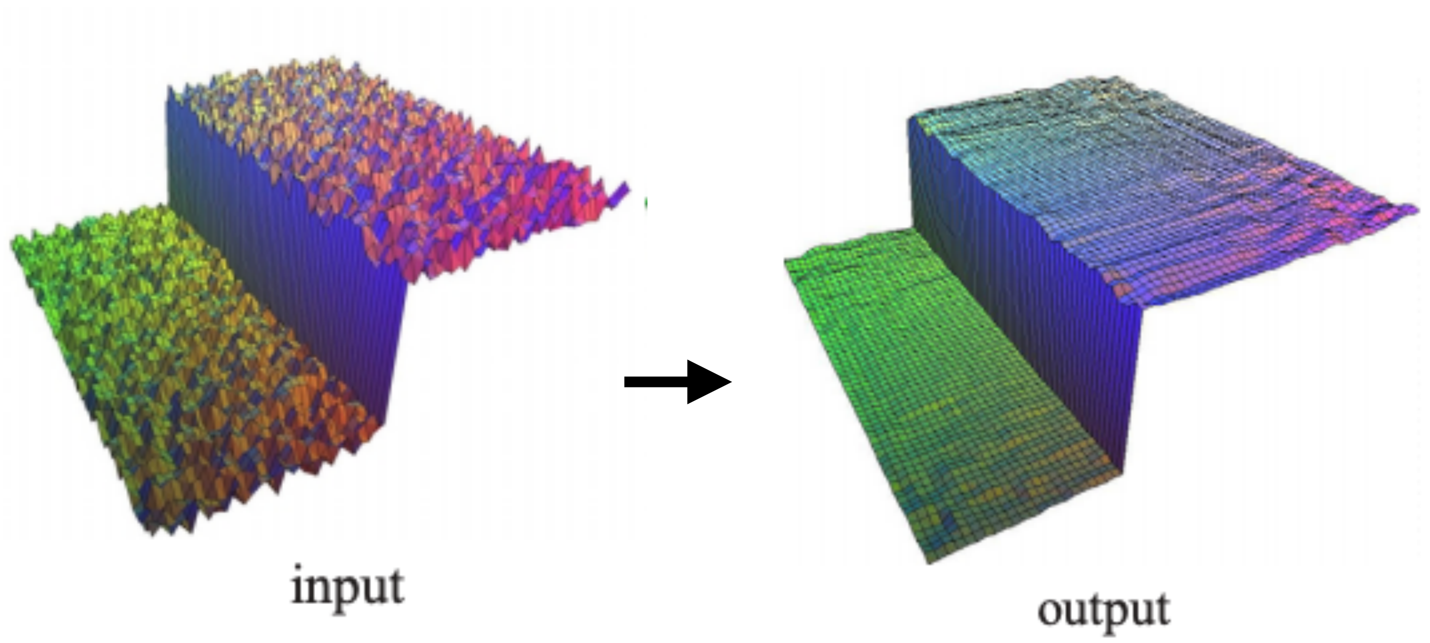
There is a huge world outside of the existing deep learning toolbox



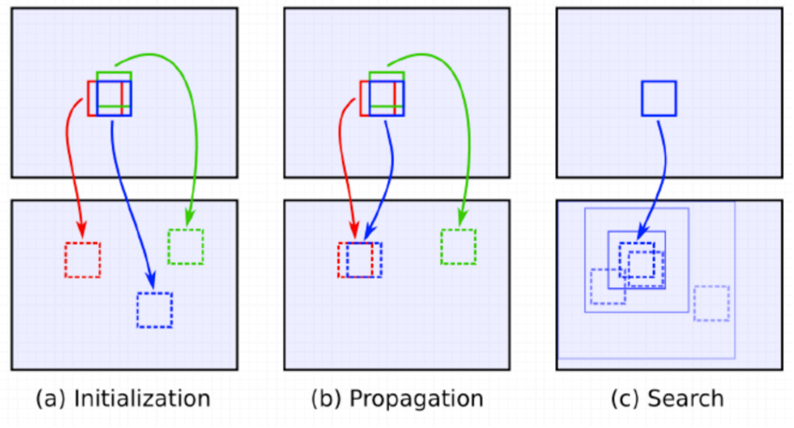
mean-shift clustering


deep learning framework toolbox

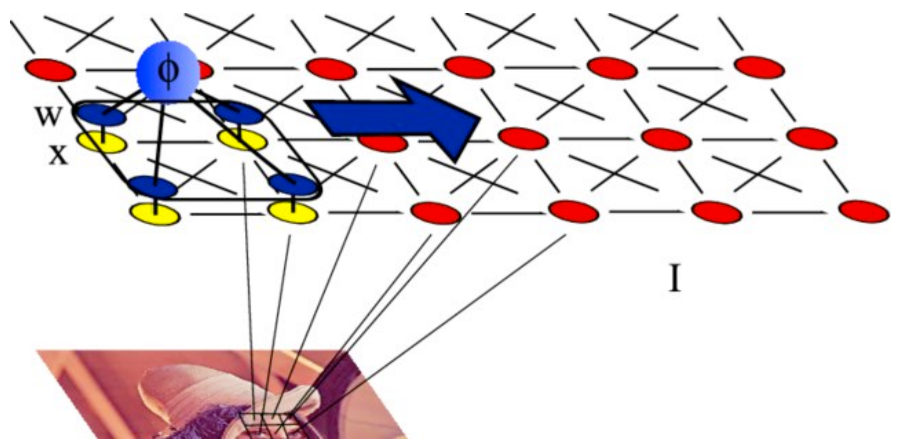

ray tracing / rasterization



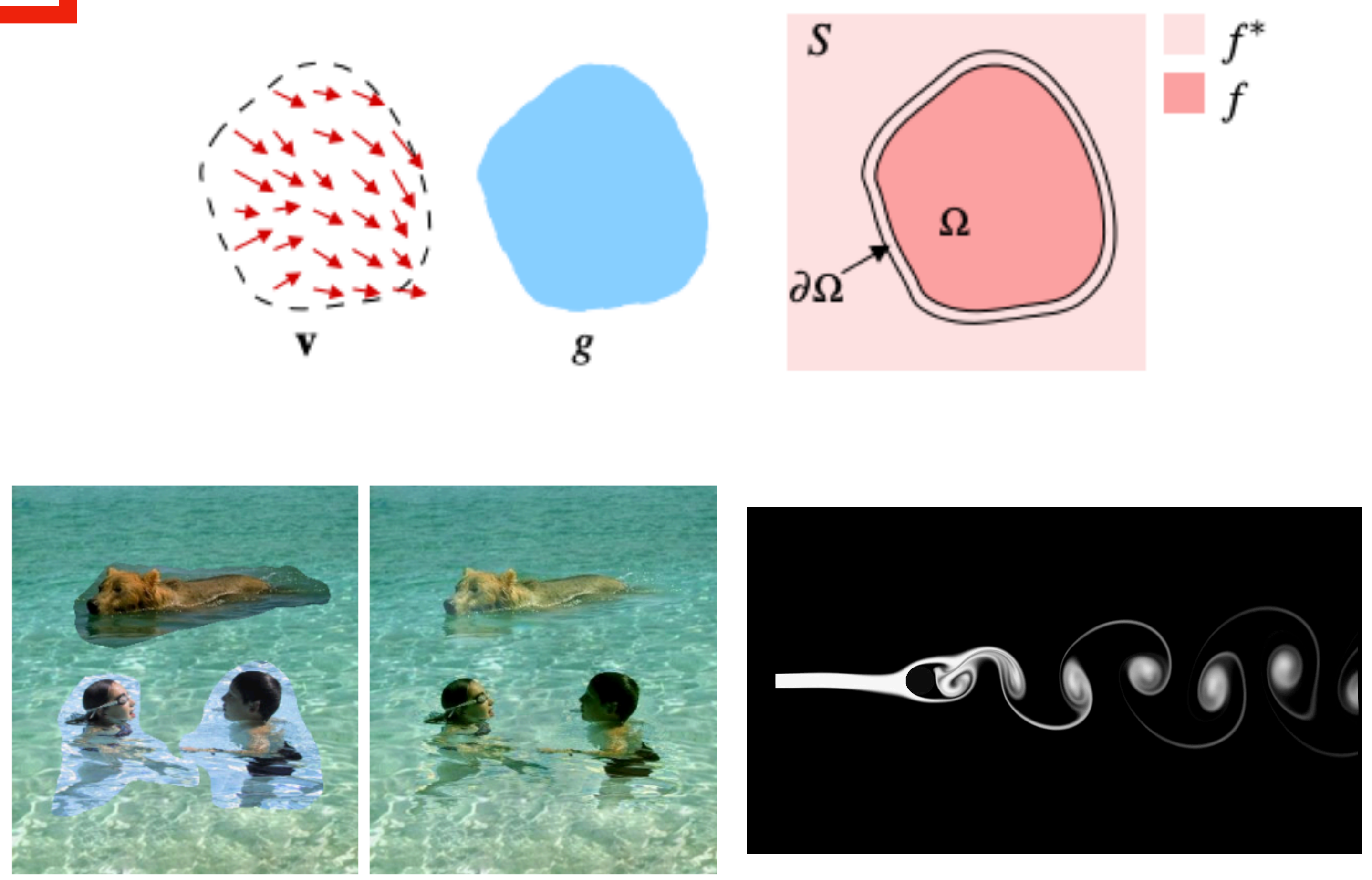
edge-aware filtering



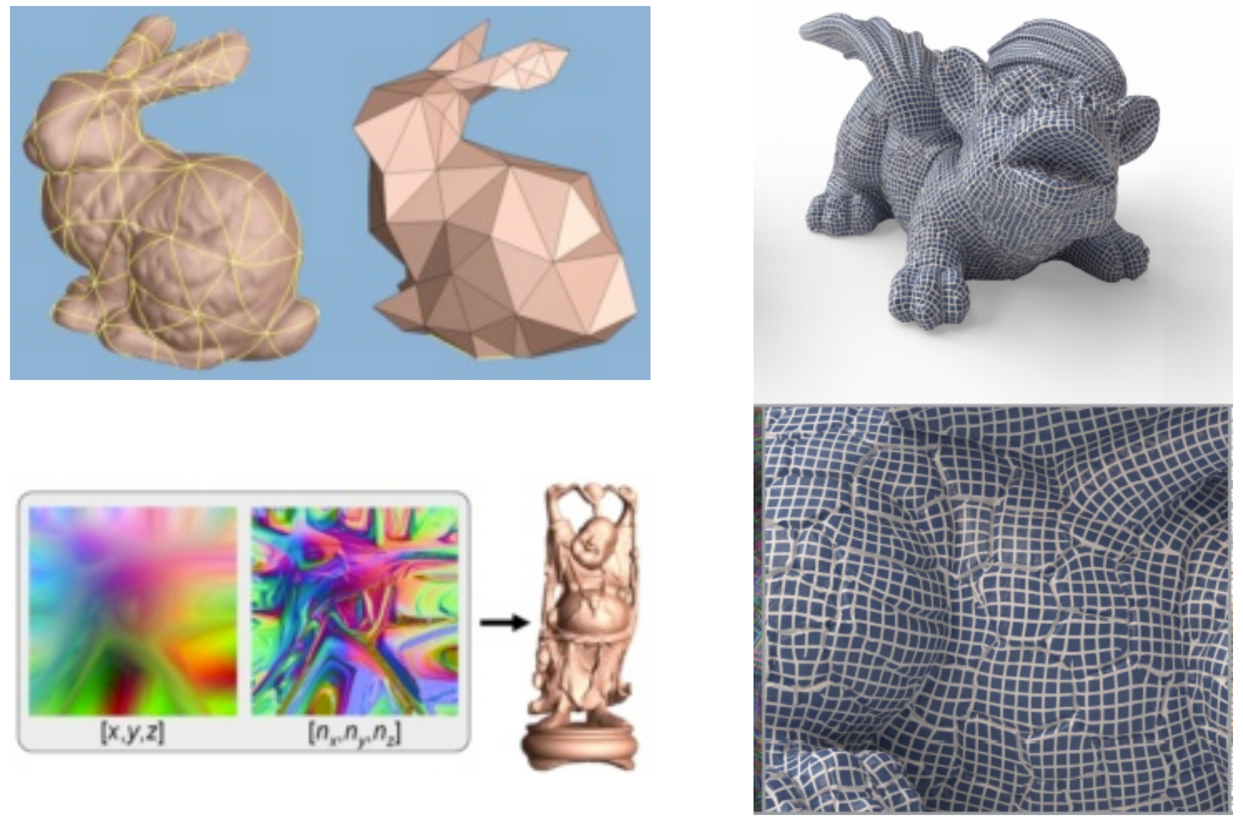
patchmatch



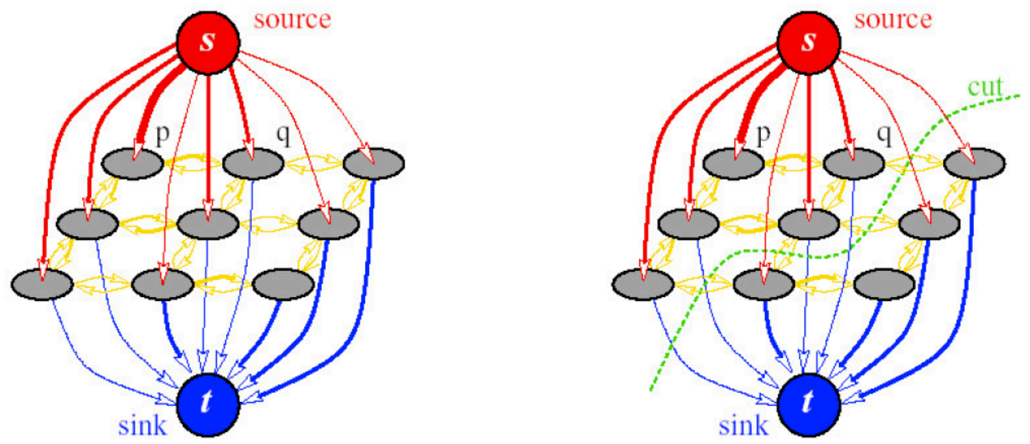
Markov random field



PDE solvers



meshing



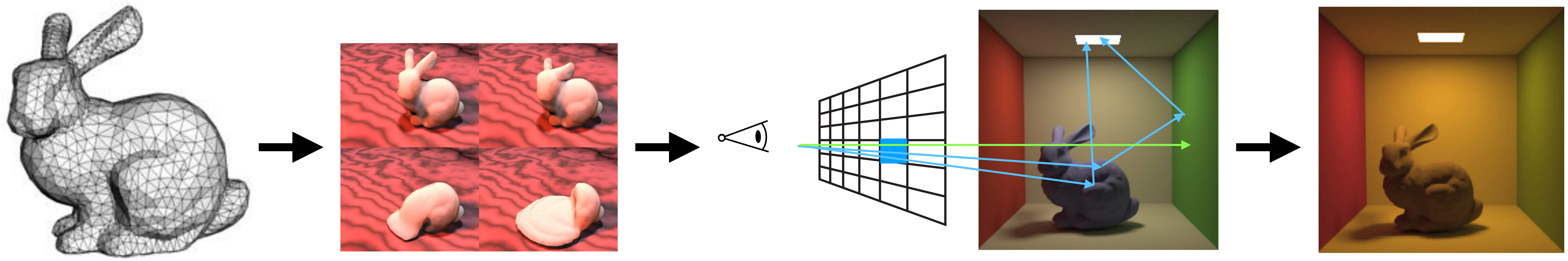
graph cut

Differentiable graphics

connects classical graphics algorithms with modern data-driven methods

Differentiable graphics

connects classical graphics algorithms with
modern data-driven methods



geometry modeling

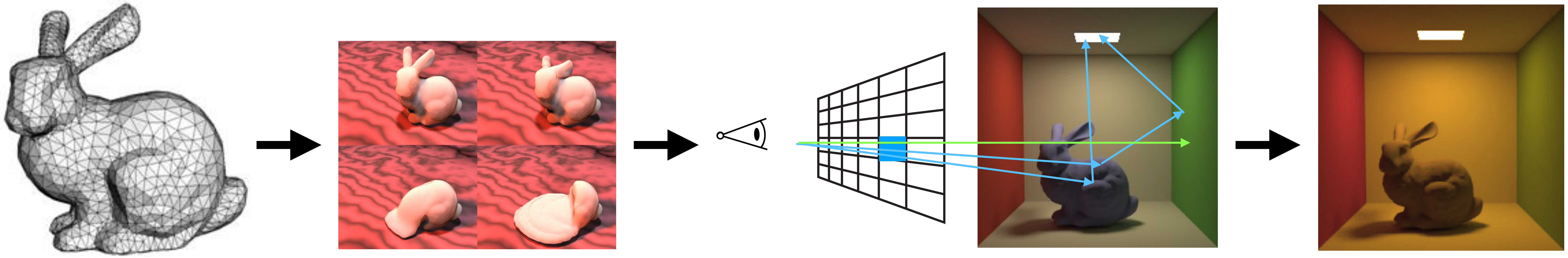
animation/
physical simulation

rendering

image processing

Differentiable graphics

connects classical graphics algorithms with
modern data-driven methods **through derivatives**



differentiable
geometry modeling

differentiable
animation/
physical simulation

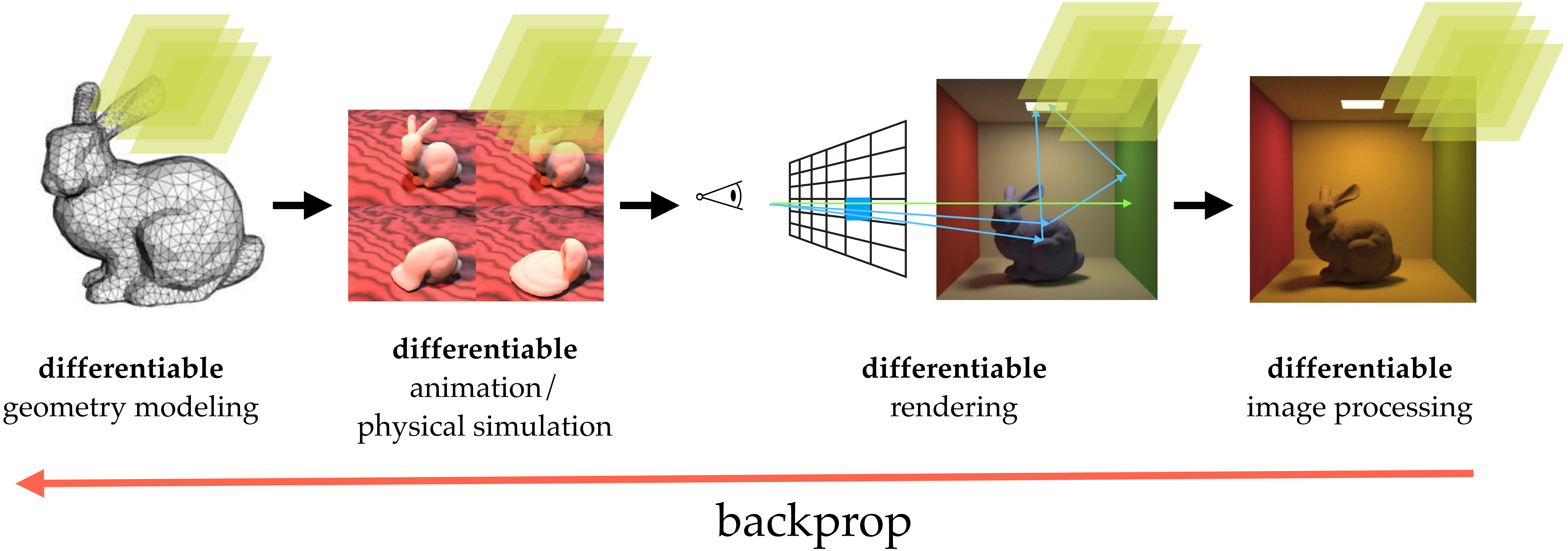
differentiable
rendering

differentiable
image processing

backprop

Differentiable graphics

connects classical graphics algorithms with
modern data-driven methods **through derivatives**



Differentiating graphics algorithms is hard

- beyond convolution
- discontinuities
- tedious derivation

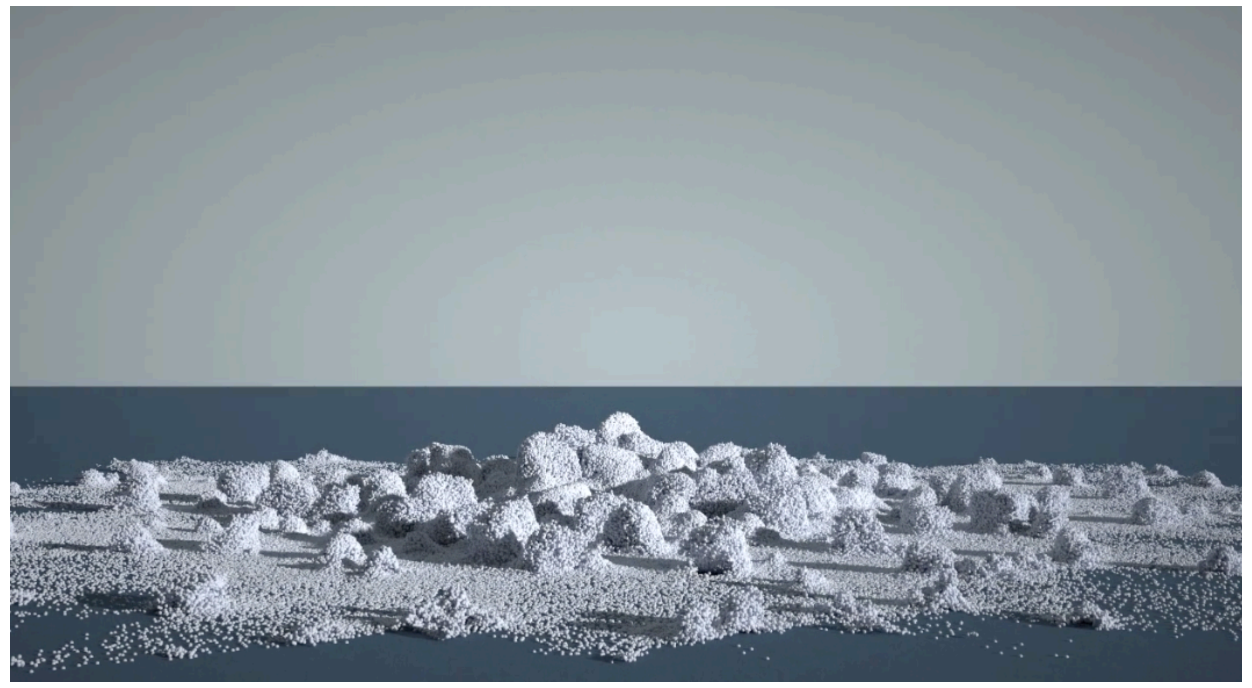
$$\nabla_\theta \int_{A(\theta)} g(x)dx = \int_{A(\theta)} \nabla_\theta g(x)dx + \int_{\partial A(\theta)} \left(\nabla_\theta p(\theta) \cdot n\right) g(p)dp$$

$$\frac{\partial \mathbf{x}}{\partial t} = \frac{\partial E}{\partial \mathbf{p}}$$

$$\frac{\partial \mathbf{p}}{\partial t} = -\frac{\partial E}{\partial \mathbf{x}} \qquad \frac{\partial \mathbf{t}'}{\partial \mathbf{c}_i} = \frac{-1}{\frac{\partial f}{\partial \mathbf{t}'}} \frac{\partial f}{\partial \mathbf{c}_i}$$

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

`tf.conv2d` v.s.

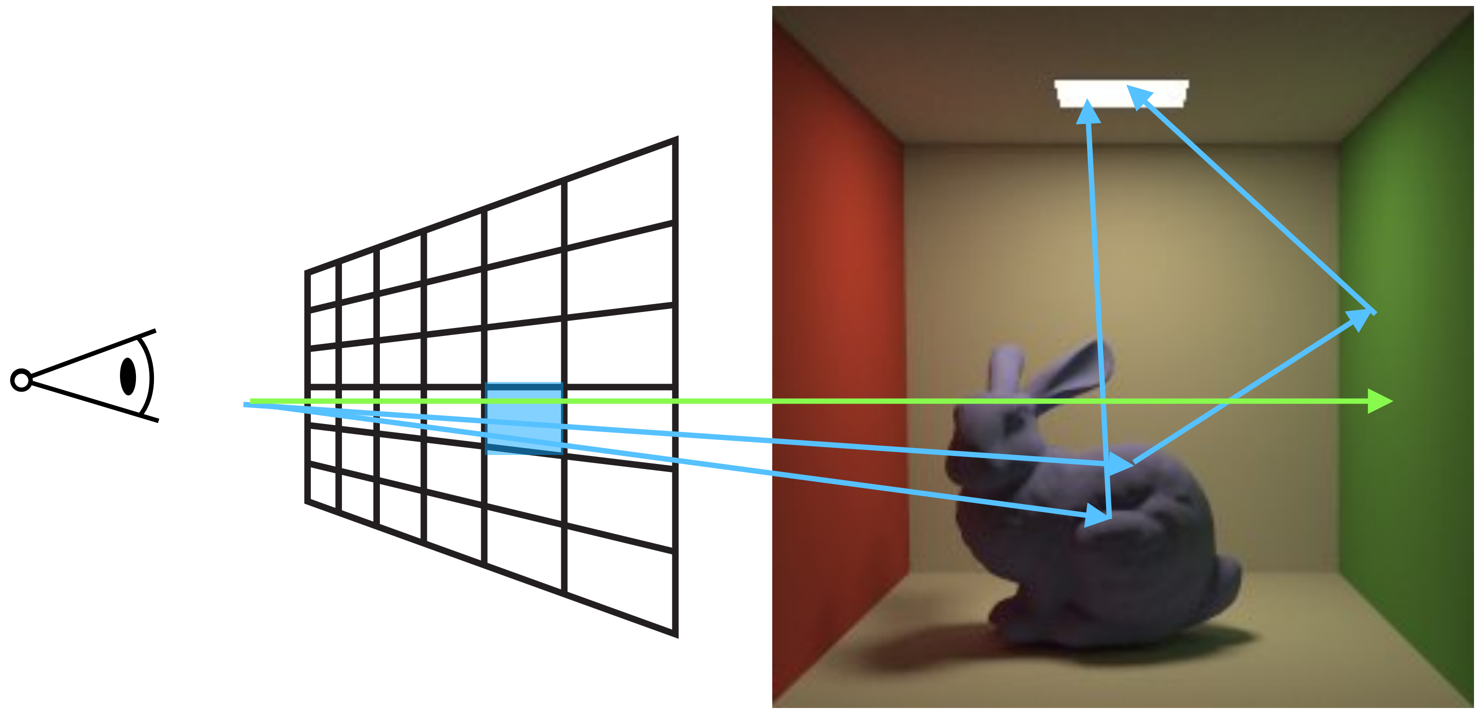


Challenges of both algorithms and systems

beyond traditional automatic differentiation

- need differentiable algorithms
- need differentiating compilers

∇

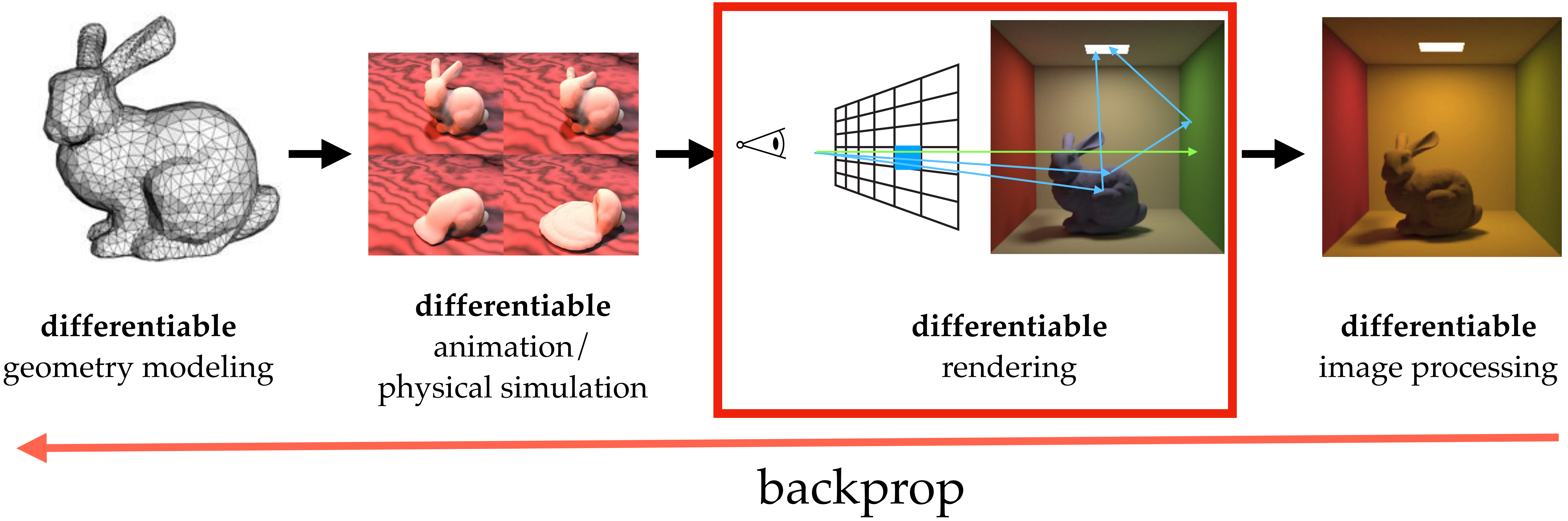


∇

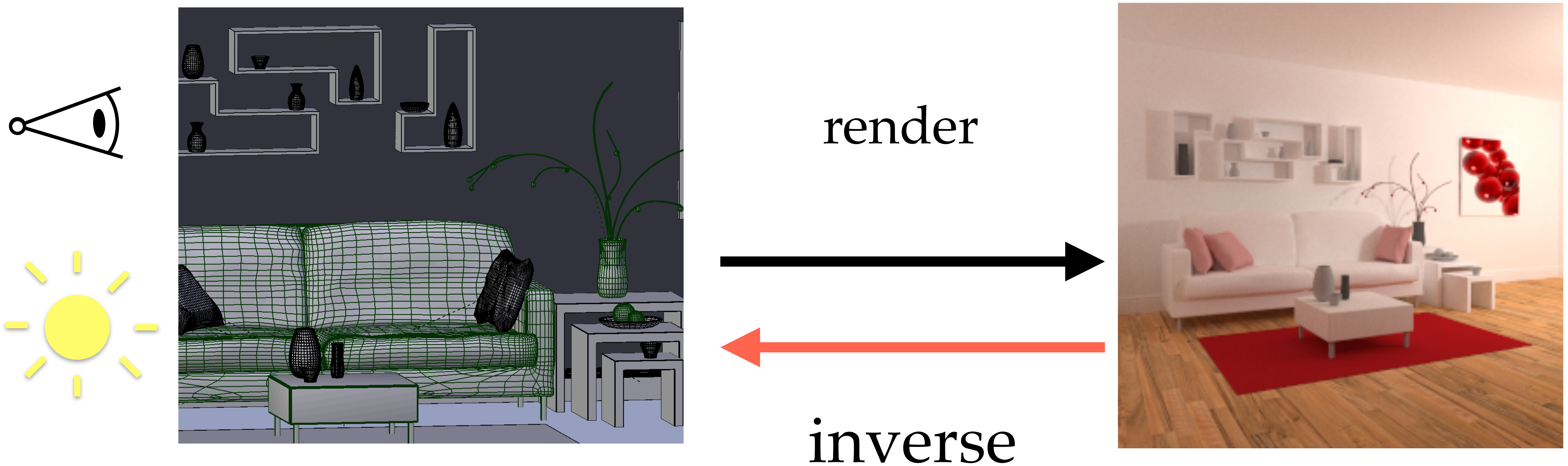
```
auto scatter_contrib = Vector3{0, 0, 0};
auto scatter_bsdf = Vector3{0, 0, 0};
if (bsdf_isect.valid()) {
    const auto &bsdf_shape = scene.shapes[bsdf_isect.shape_id];
    auto dir = bsdf_point.position - p;
    auto dist_sq = length_squared(dir);
    auto wo = dir / sqrt(dist_sq);
    auto pdf_bsdf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

Differentiable graphics

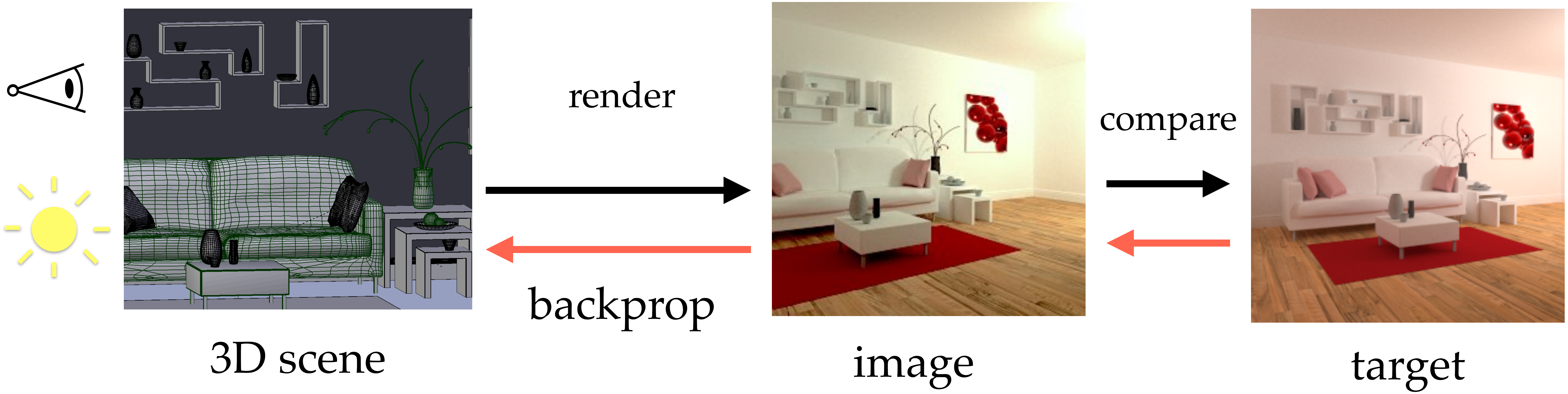
connects classical graphics algorithms with
modern data-driven methods **through derivatives**



Motivation 1: inverse rendering



Use gradients to update 3D scene



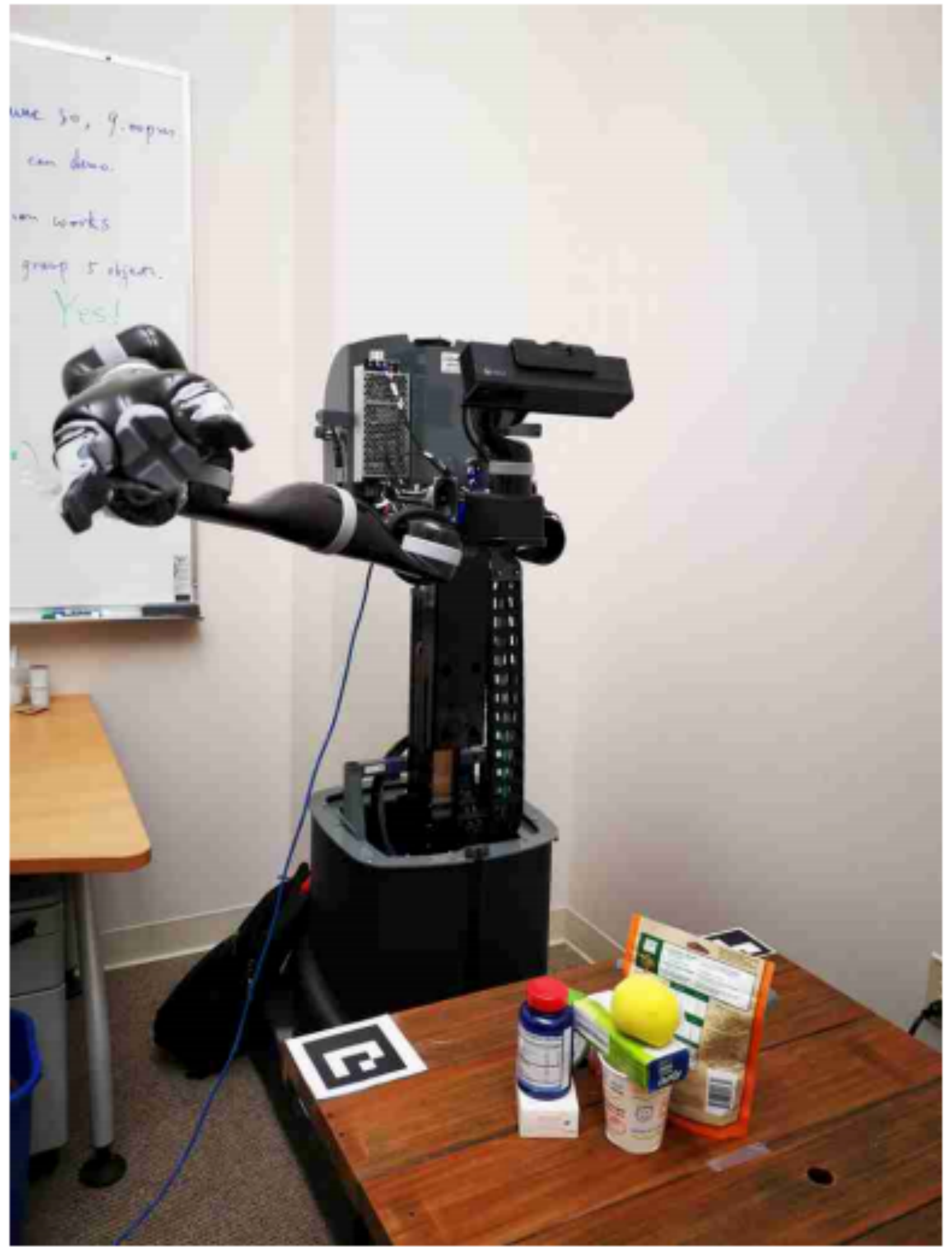
camera
geometry
material
light

Motivation 2: deep learning

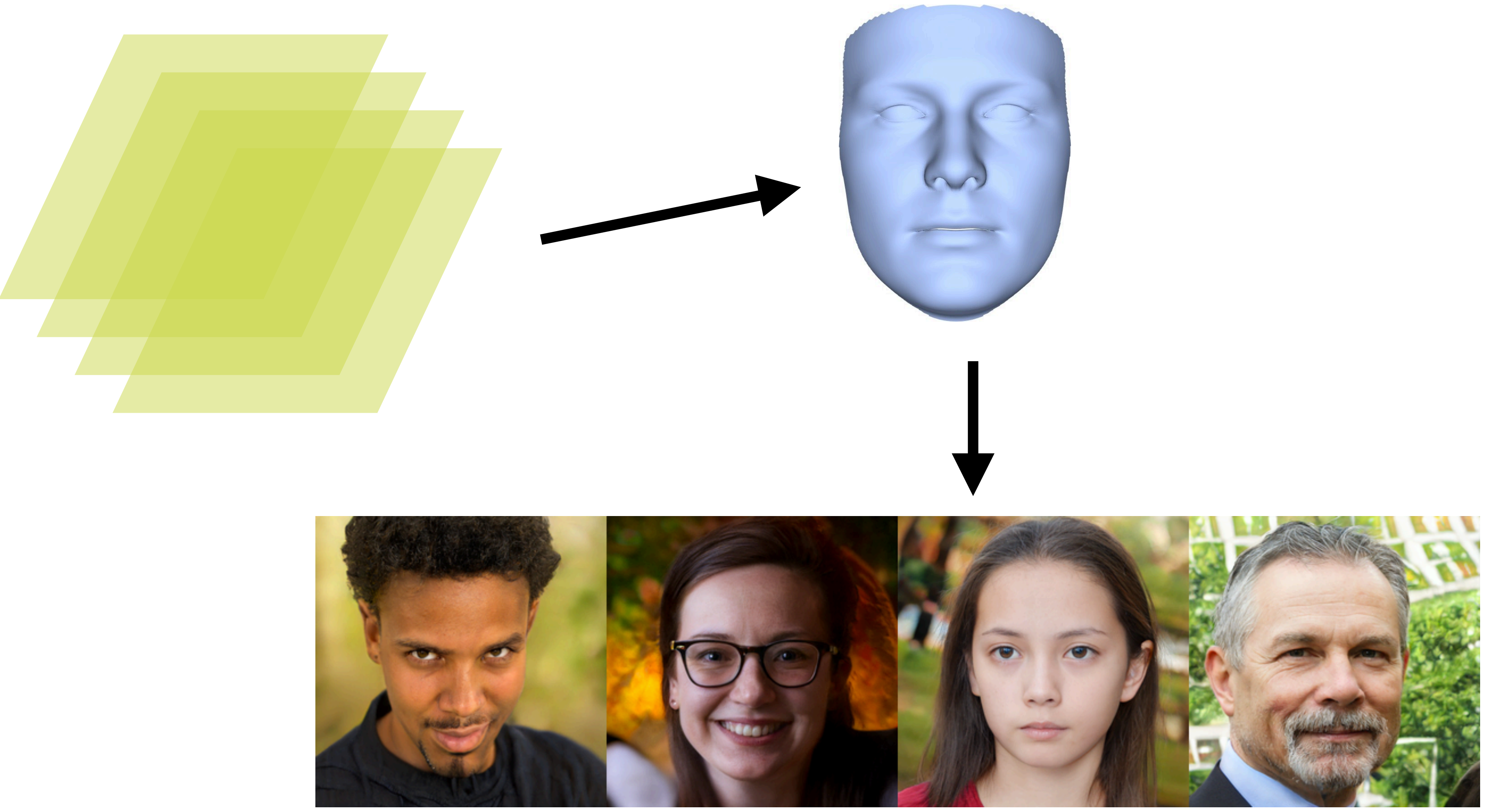
adversarial robustness, self-supervised learning, etc

Many applications

virtual conferencing



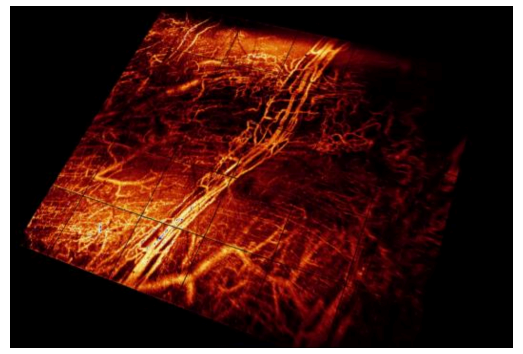
robotics



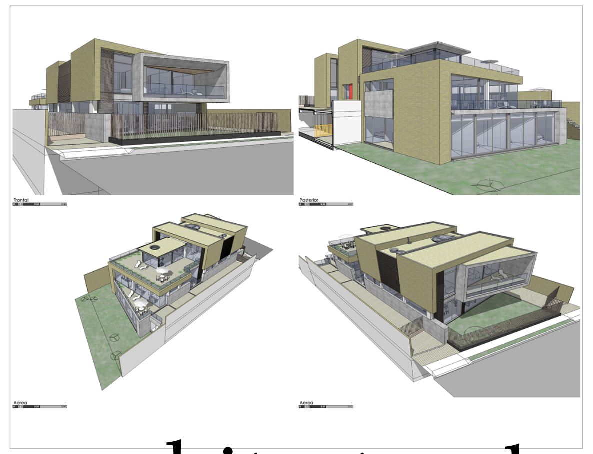
3D content creation

visual effects

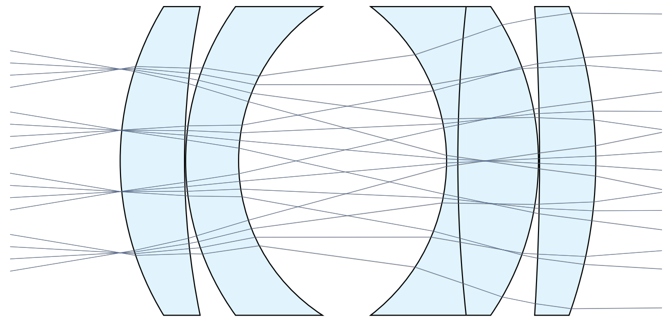
fabrication



biomedical optics

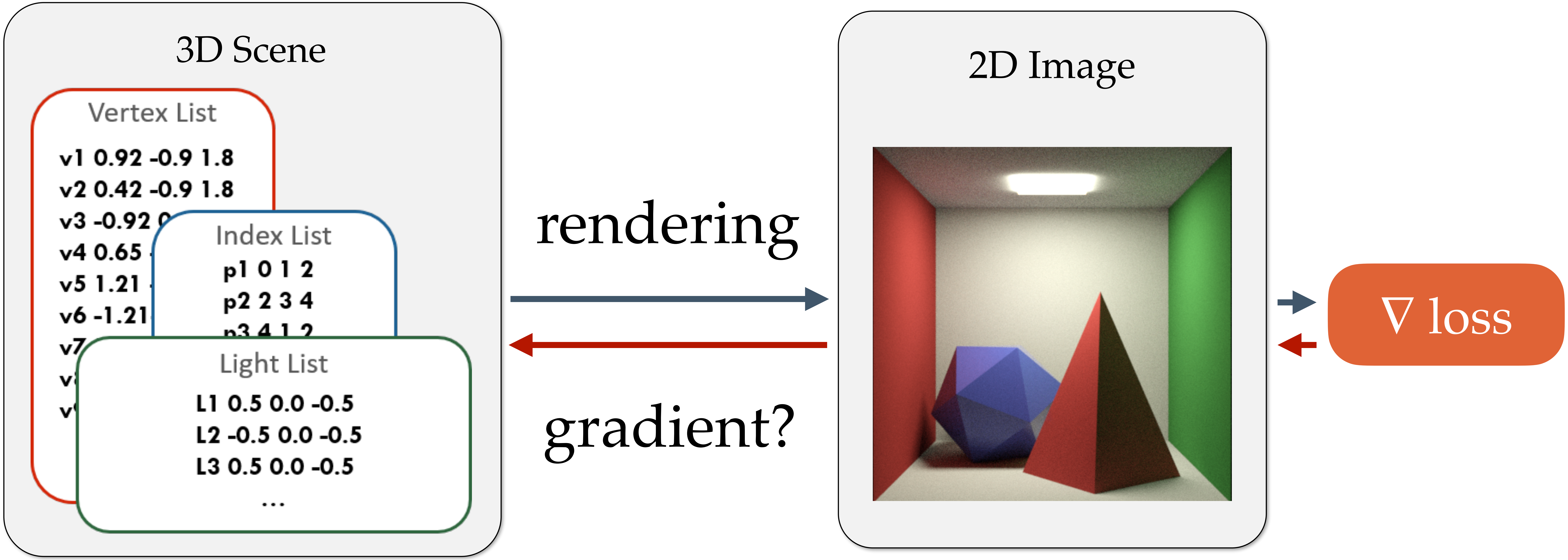


architectural design



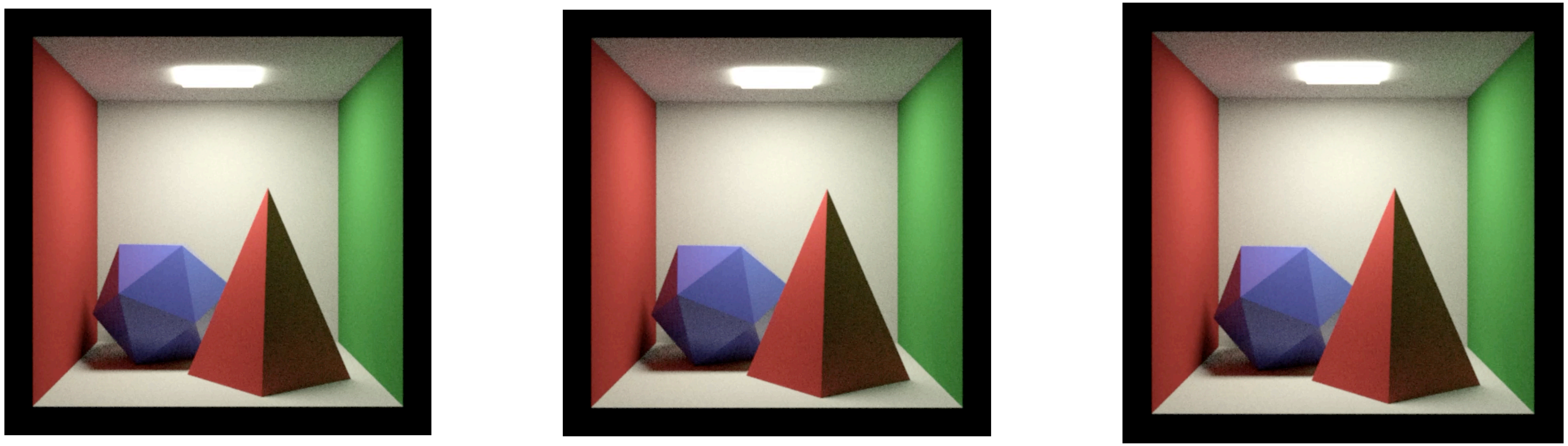
computational imaging

Task: compute the rendering gradient



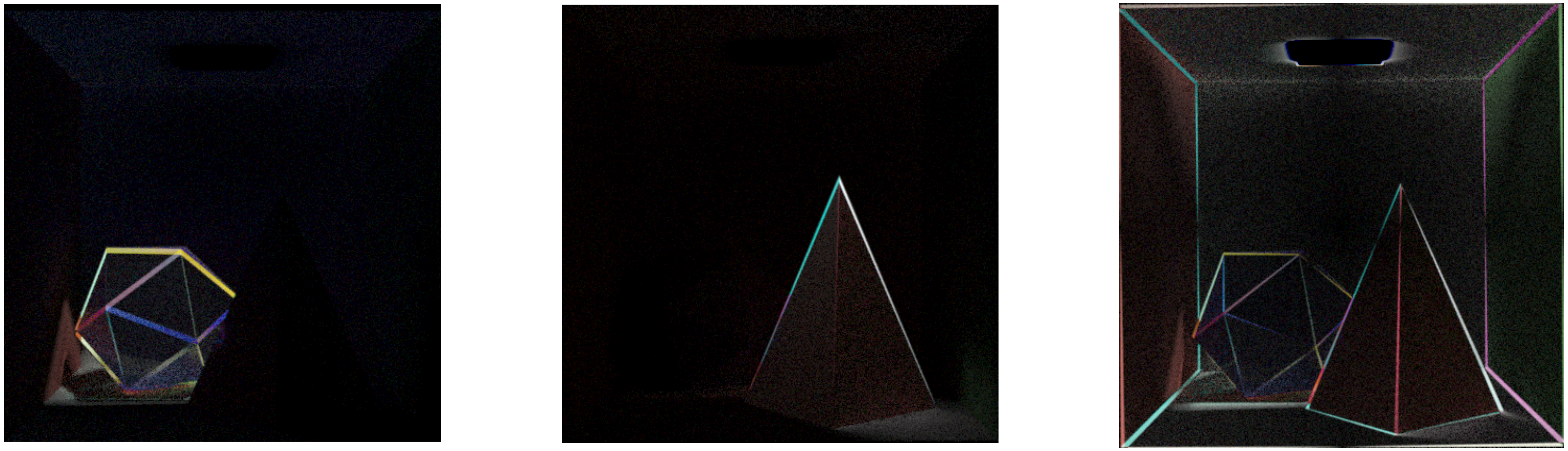
Task: compute the rendering gradient

image $\boldsymbol{I}$



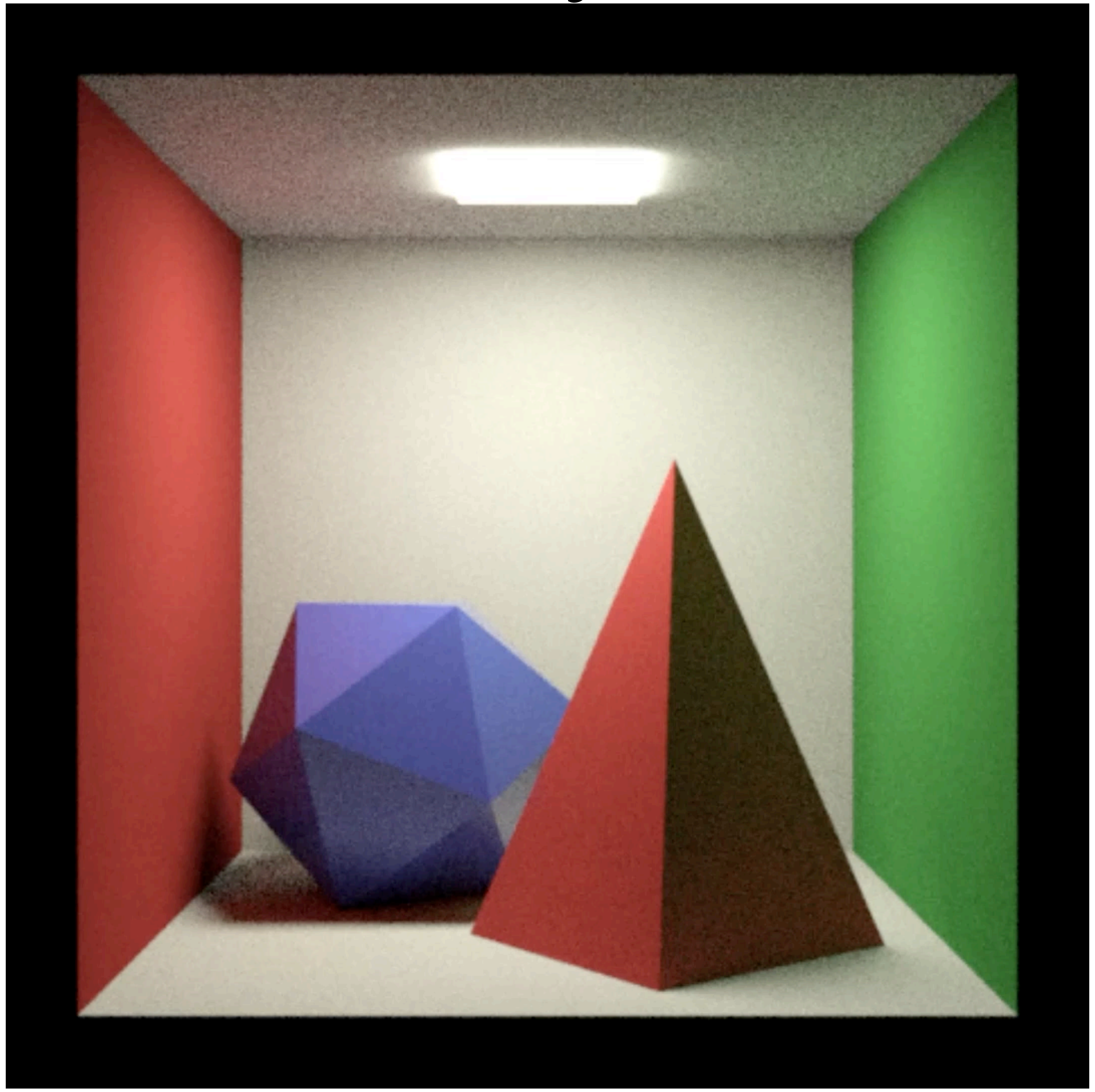
θ: object translation   θ: vertex position   θ: camera rotation

gradient $\frac{\partial \boldsymbol{I}}{\partial \theta}$



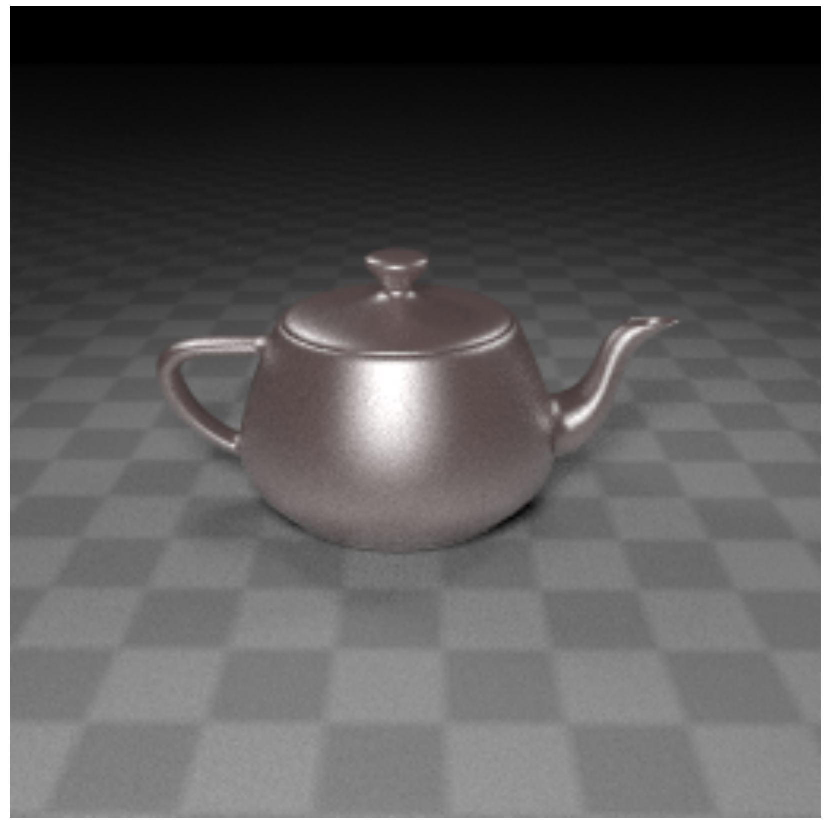
Challenge: visibility

both camera visibility and shadow matter

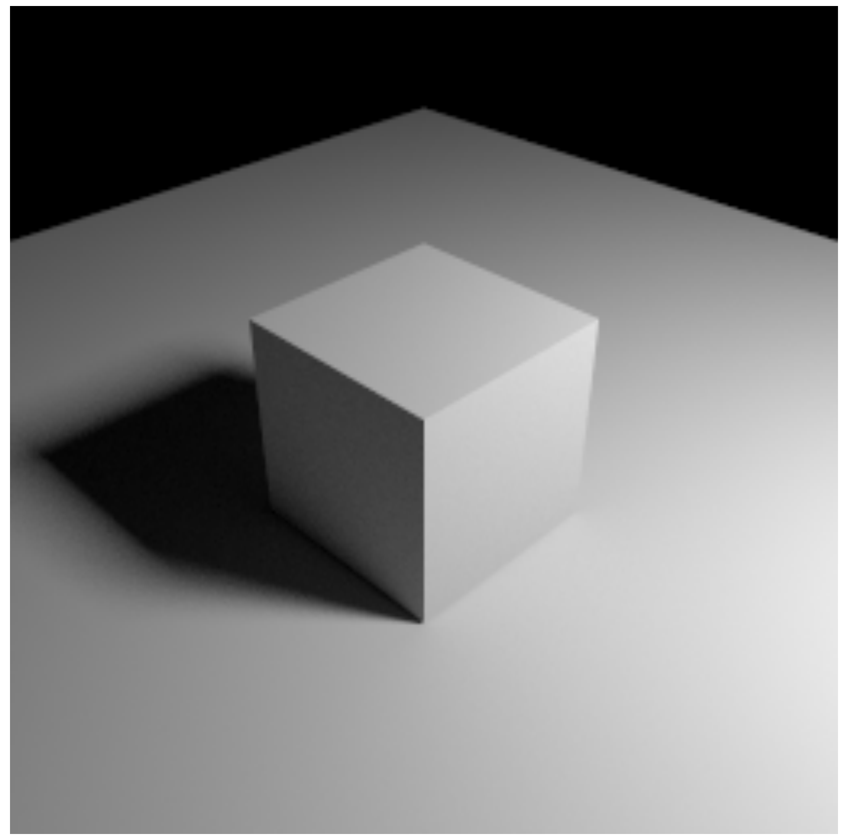


Goal: correct, general, differentiable, and physically-based rendering

camera visibility +



glossy reflection

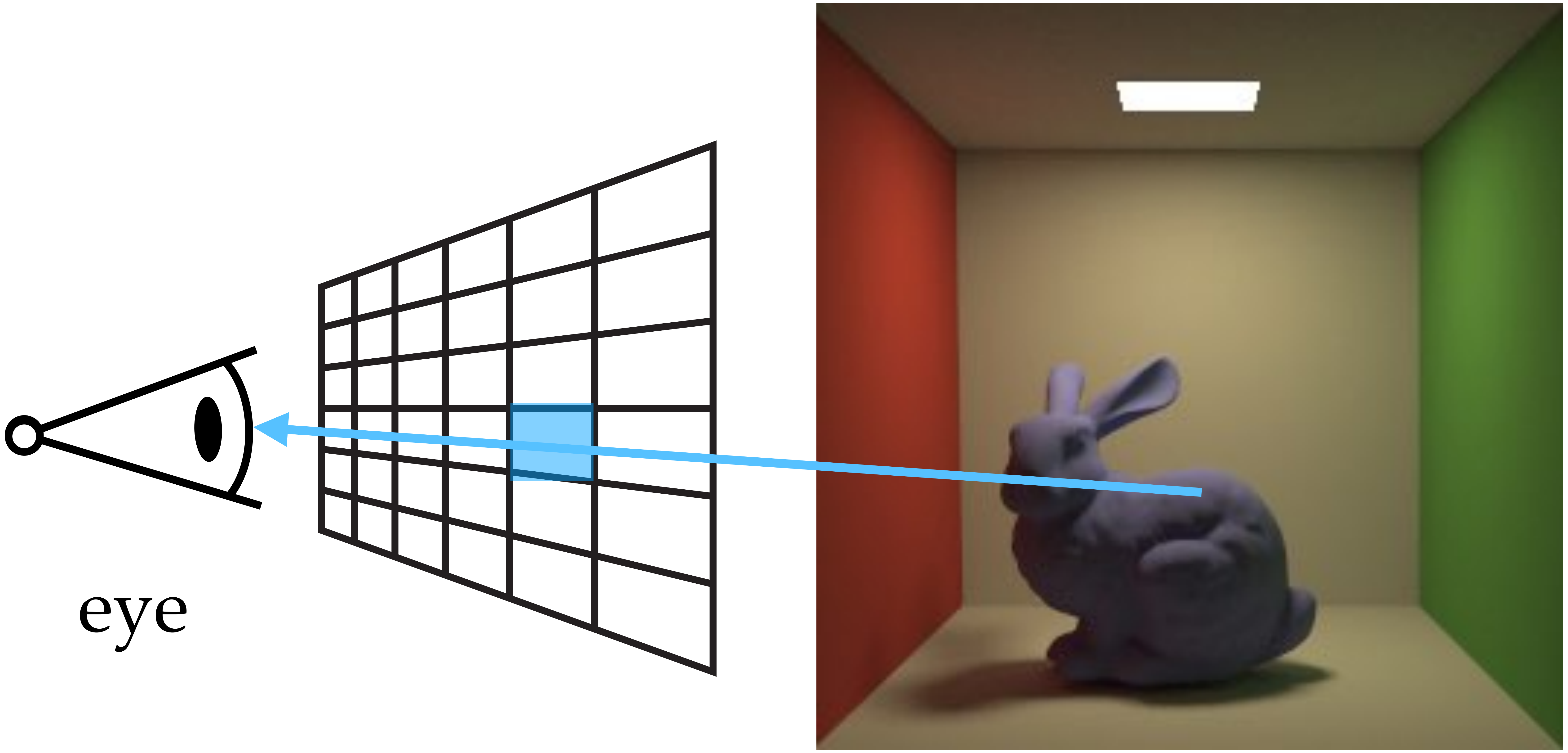


shadow

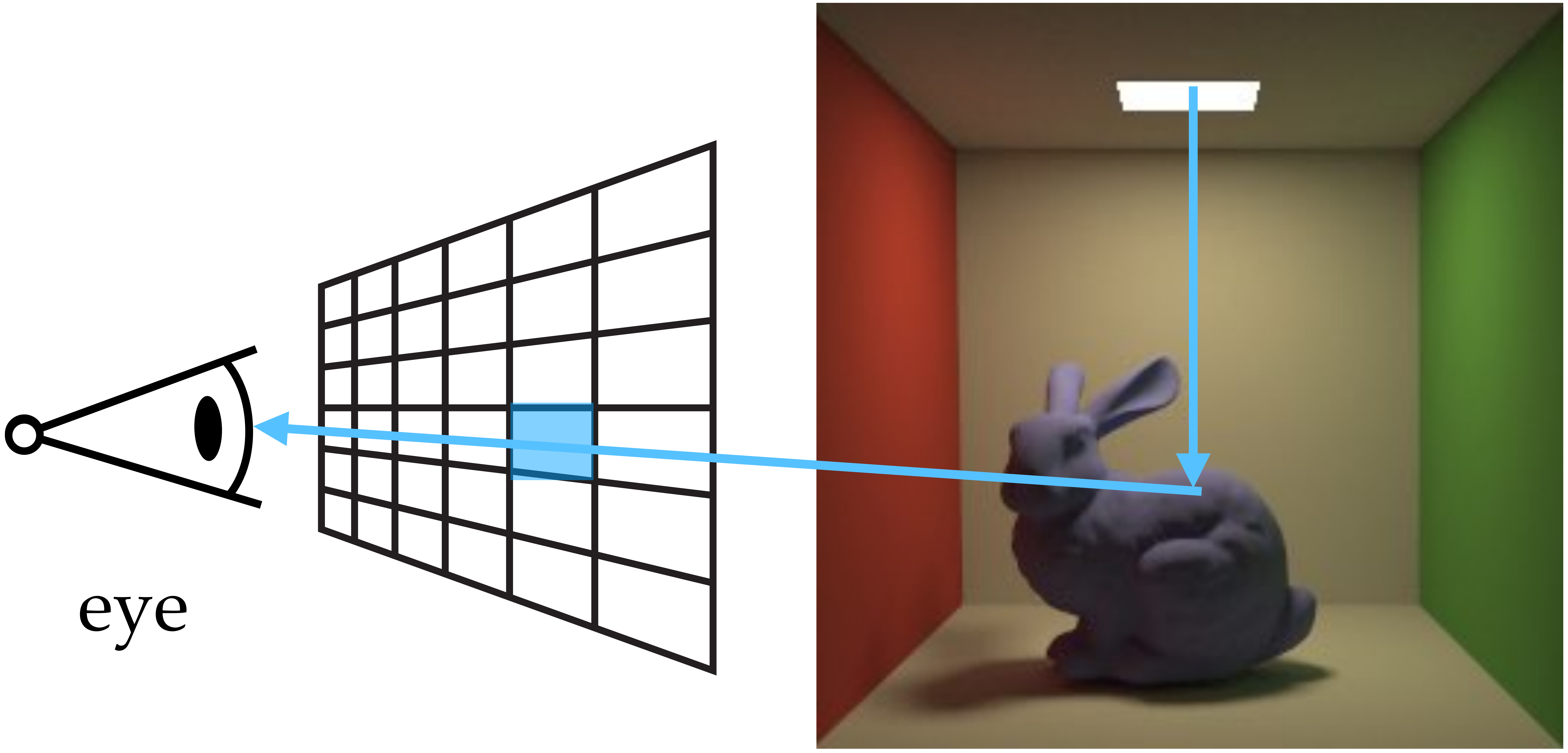
mirror reflection

global illumination

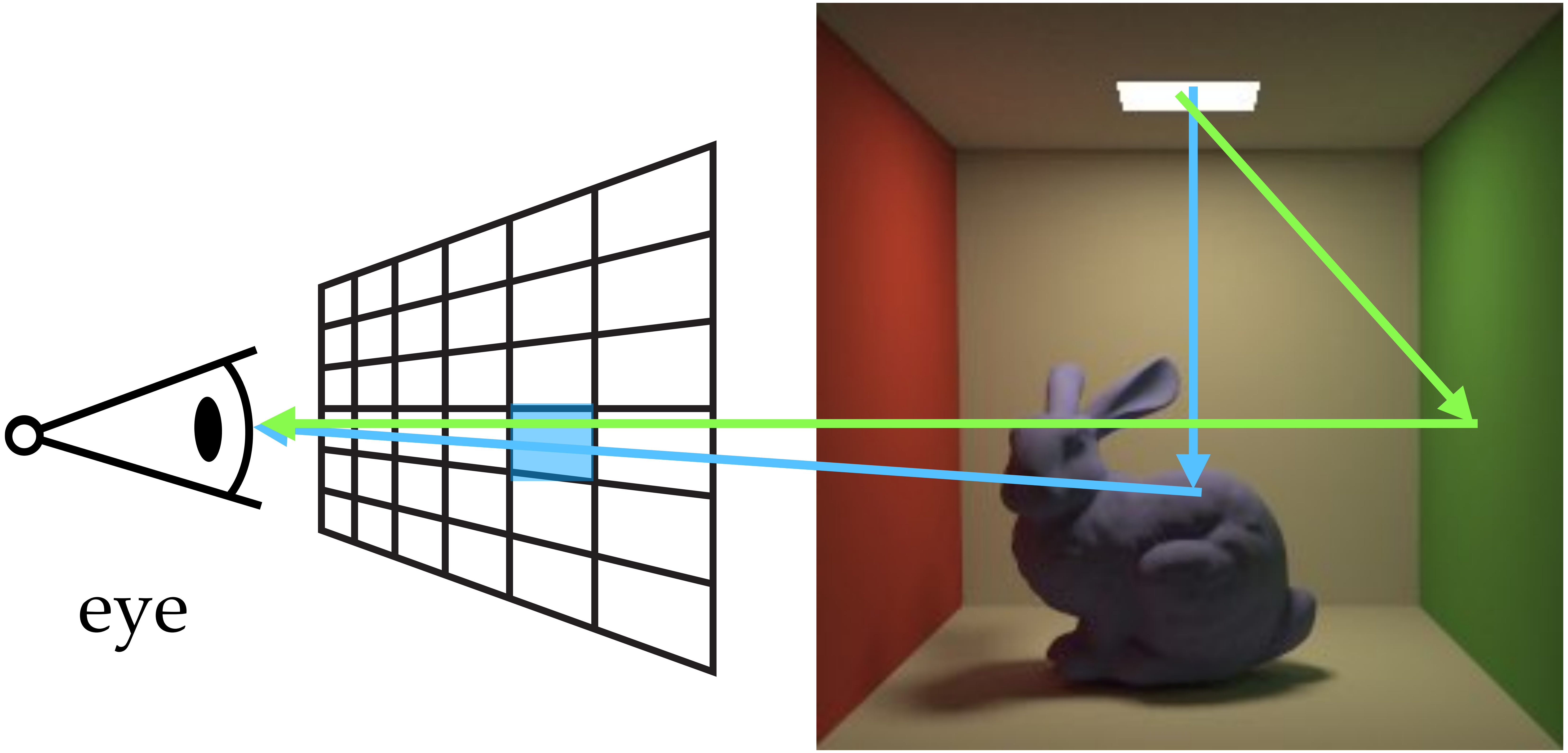
Renderers sample light paths



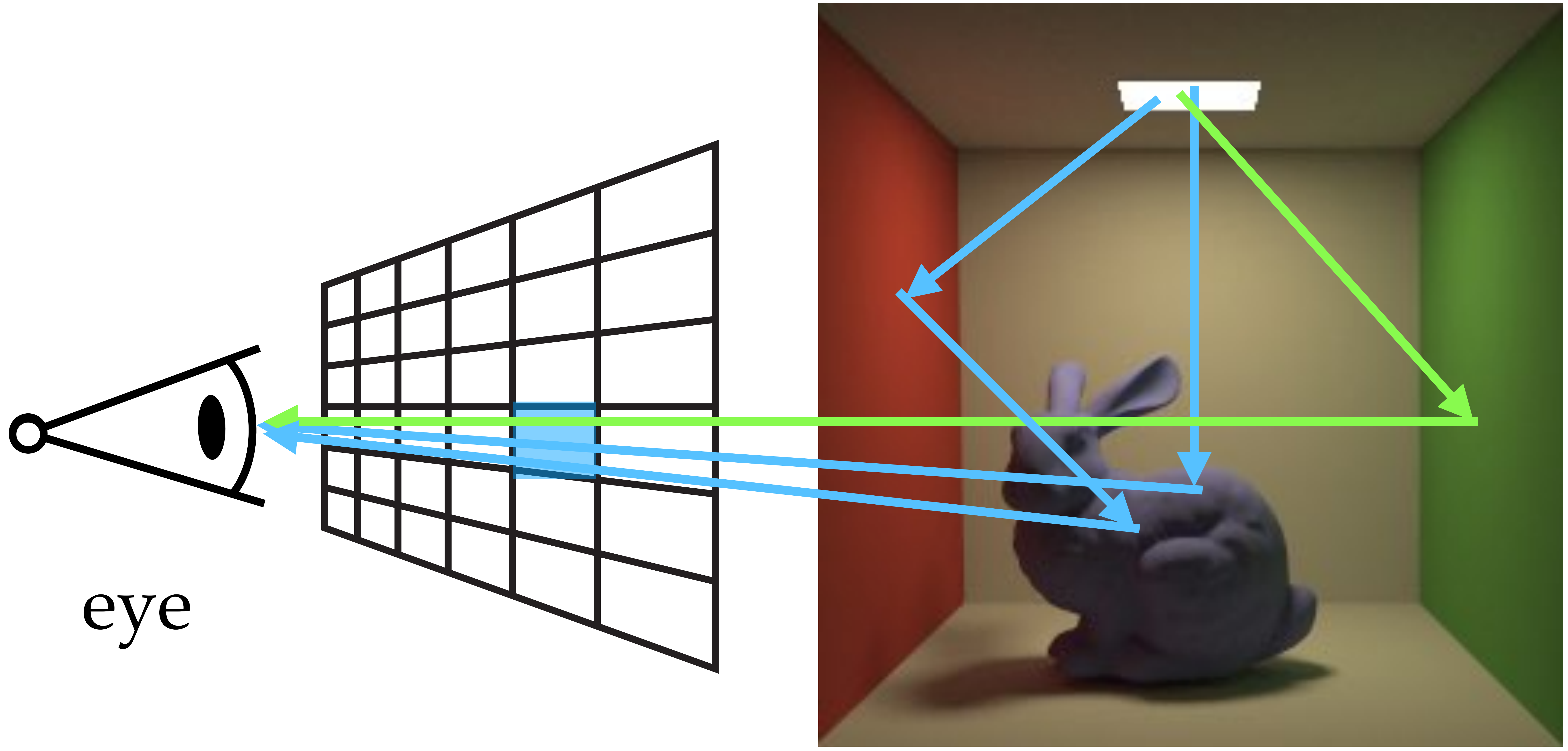
Renderers sample light paths



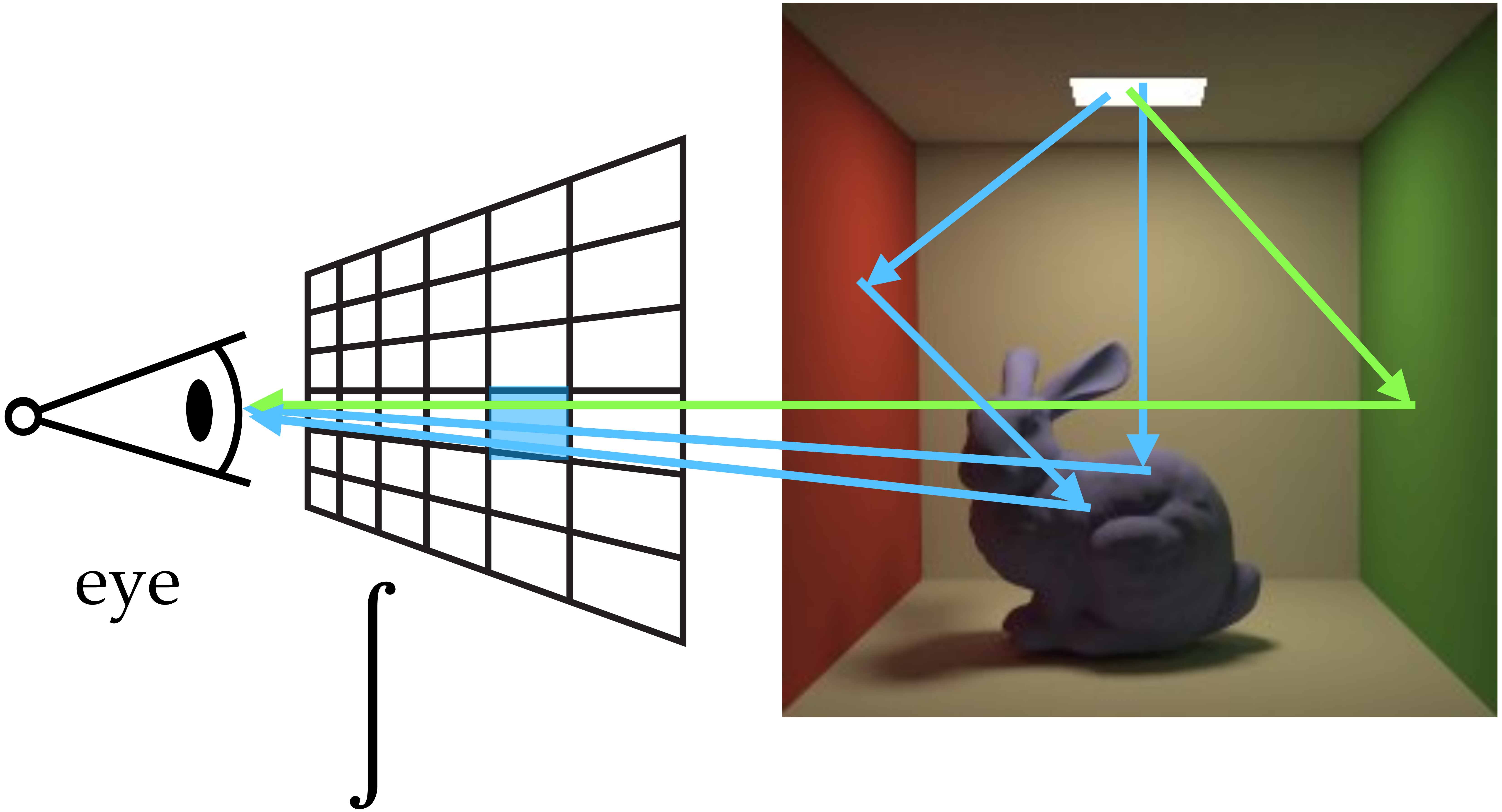
Renderers sample light paths



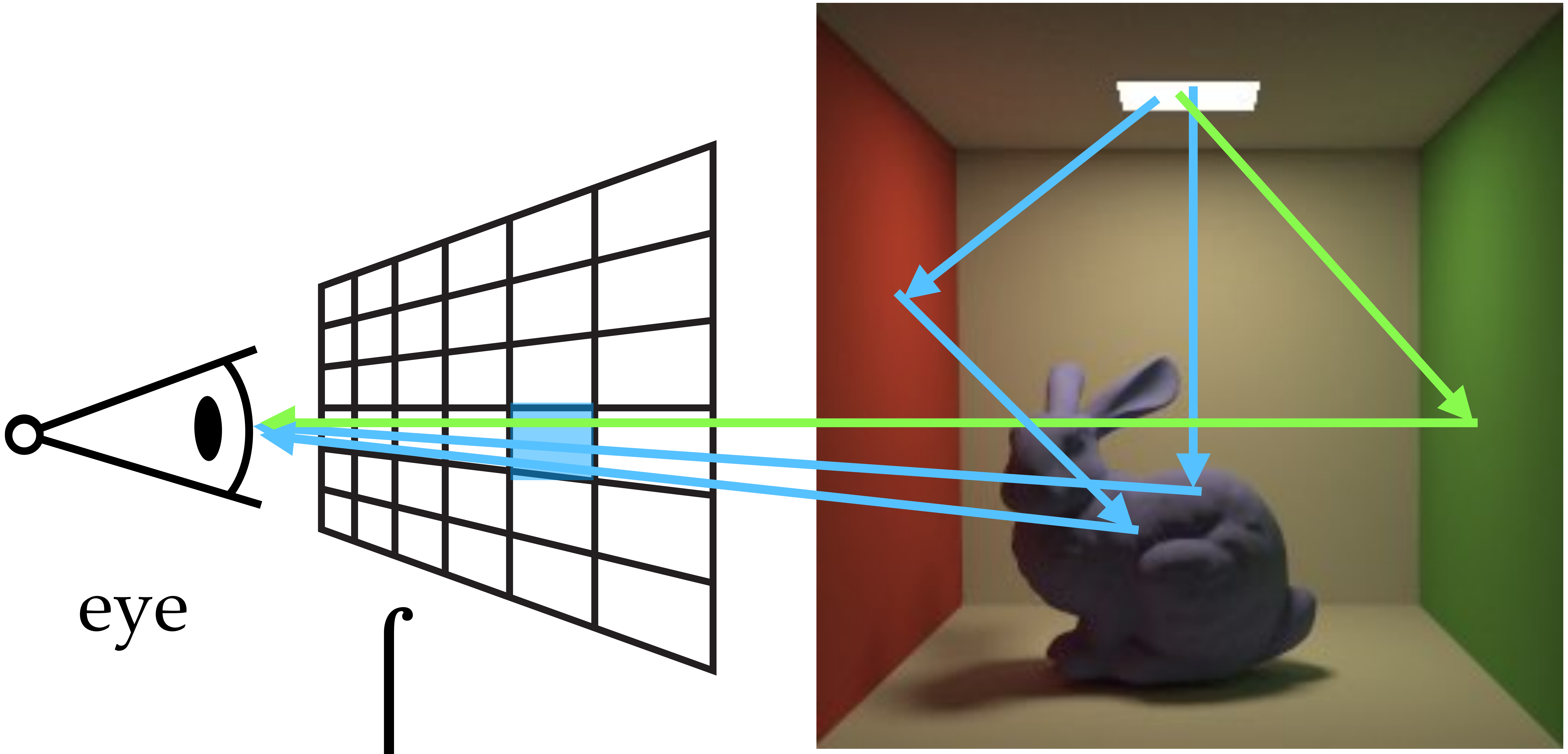
Renderers sample light paths



Renderers sample light paths



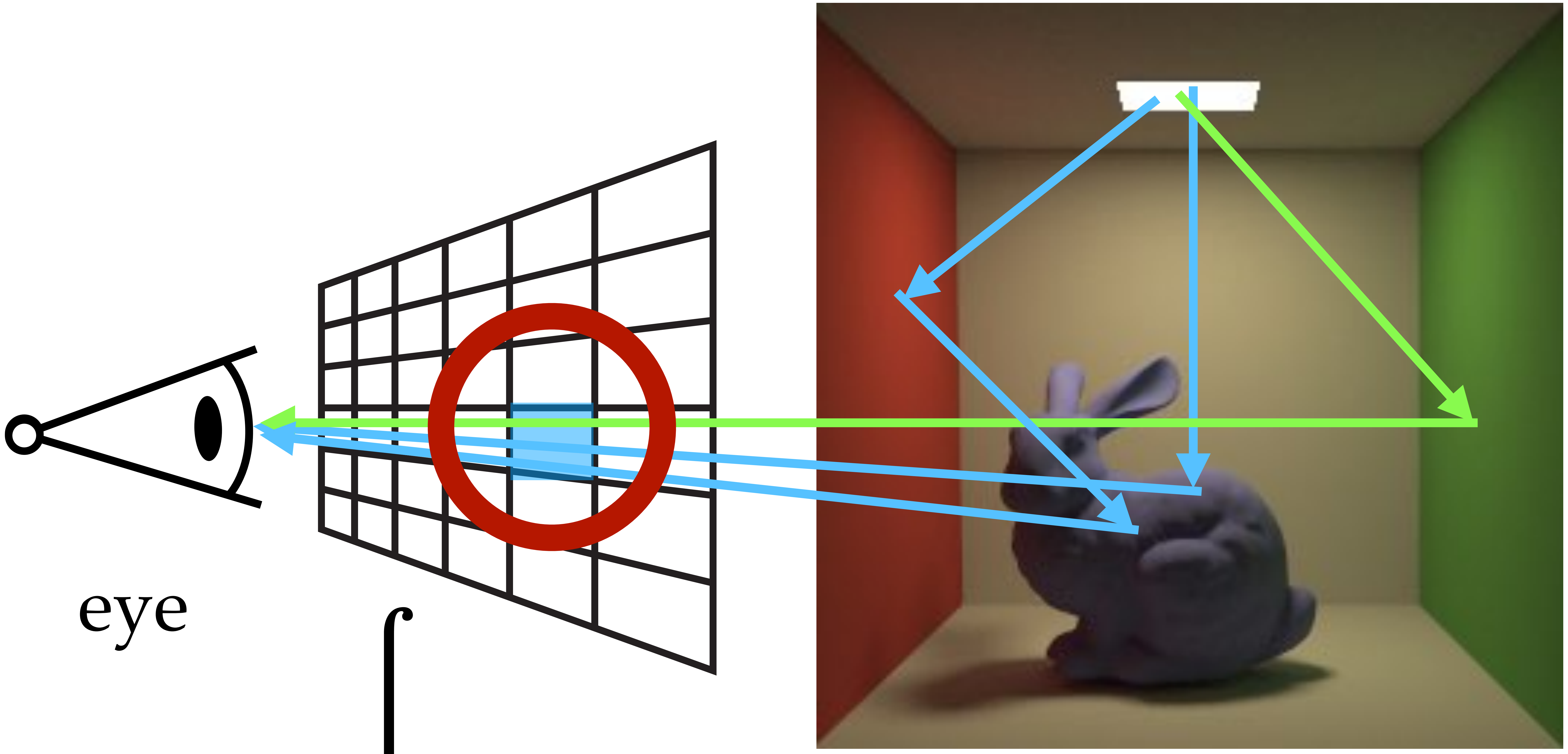
Renderers sample light paths



$\int$ key observation: the **integrand** is discontinuous, but the **integral** is differentiable

***Li**, Aittala, Durand, Lehtinen, 2018*

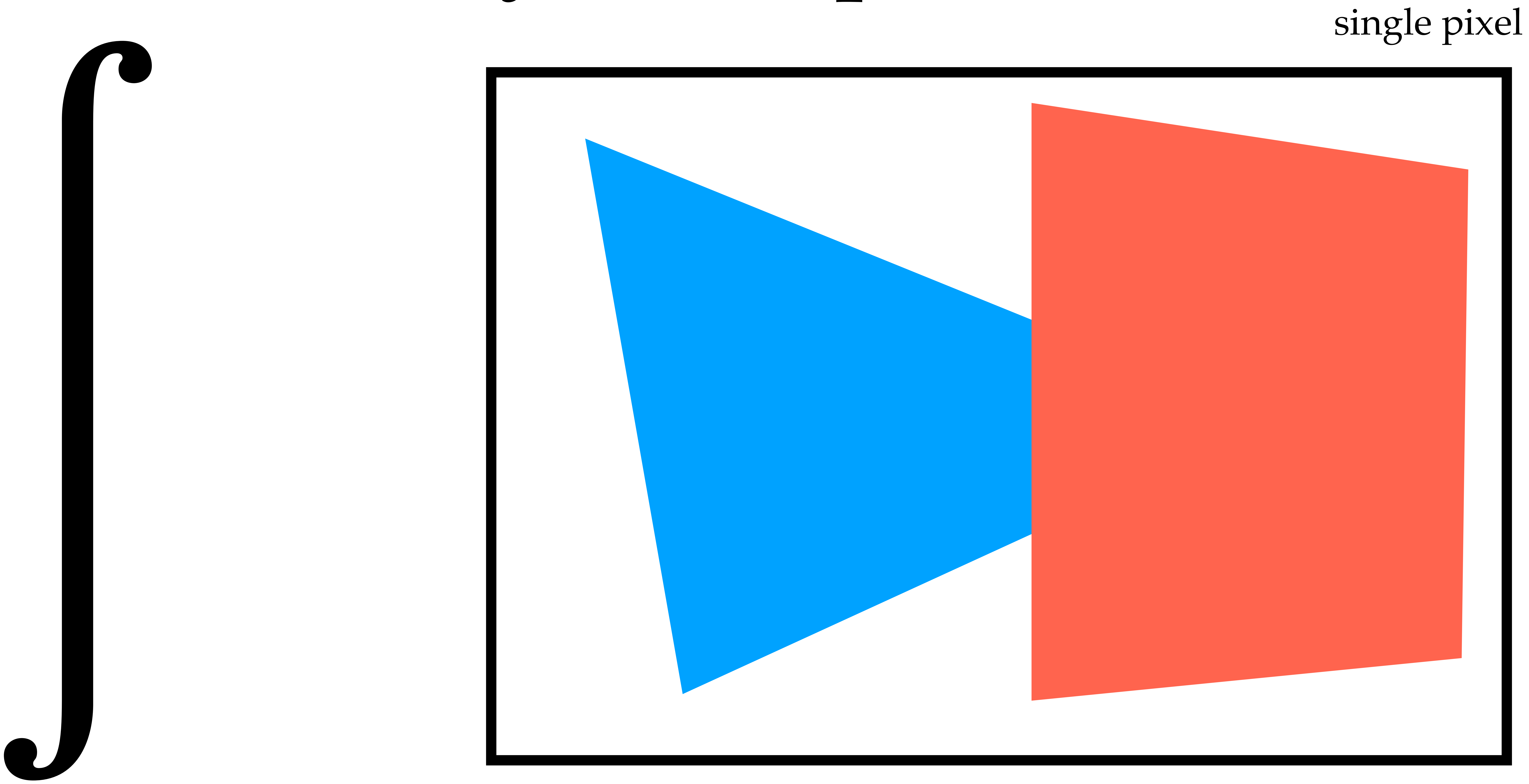
Renderers sample light paths



$\int$ key observation: the **integrand** is discontinuous, but the **integral** is differentiable

__Li__, Aittala, Durand, Lehtinen, 2018

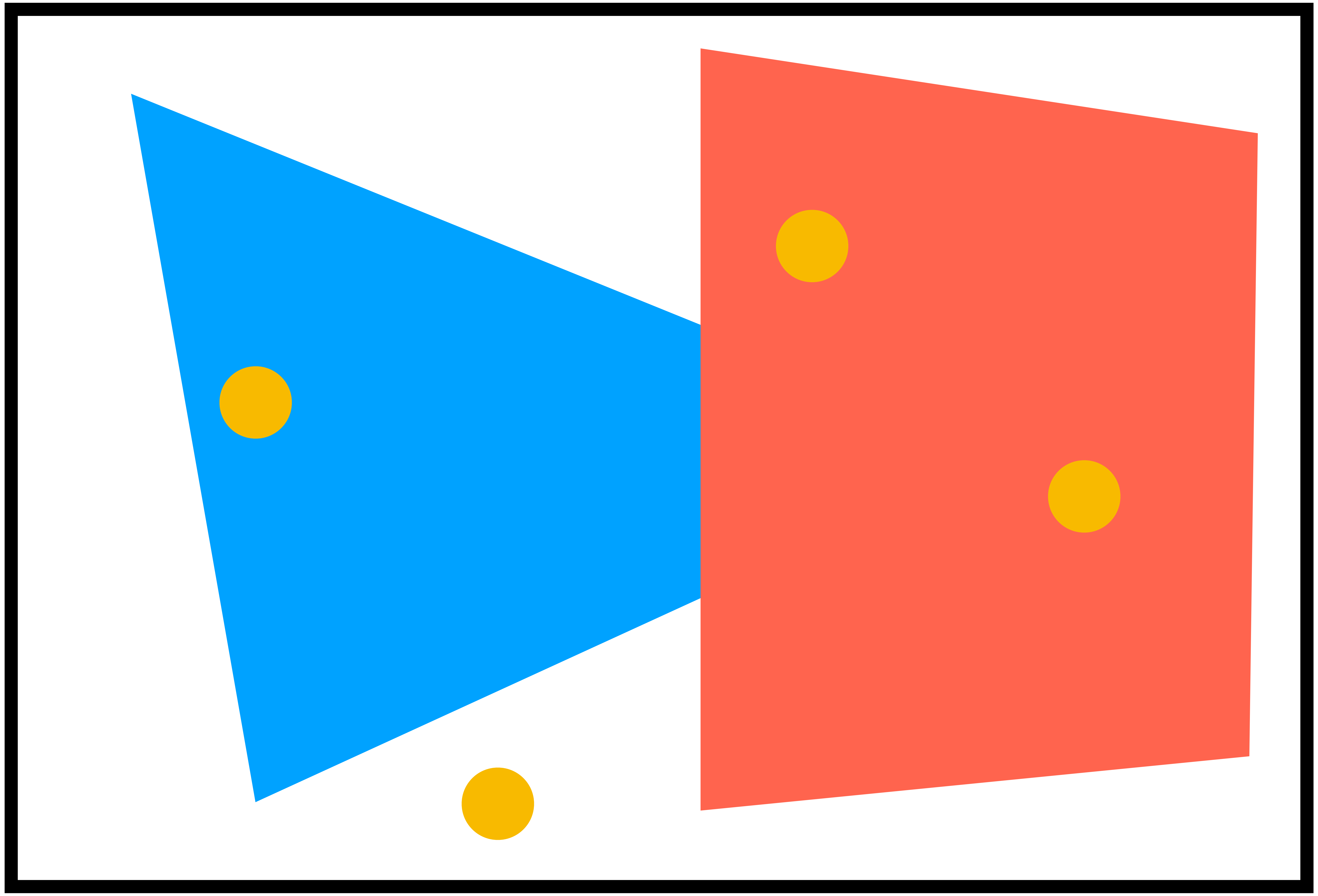
Toy example



rendering = sample integral

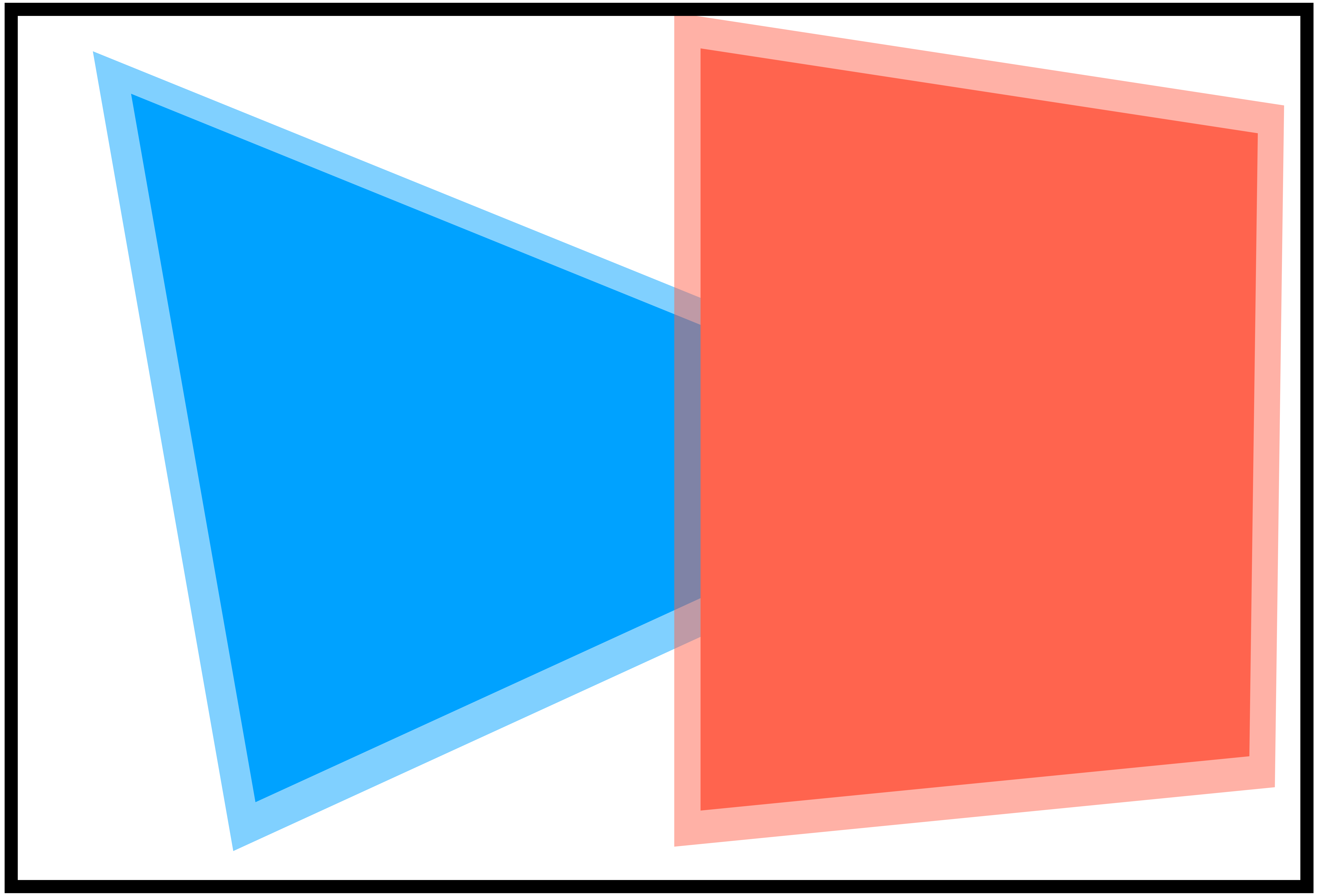
$$\Sigma$$

single pixel



Differential of shape parameters = boundary changes

$$\nabla \int$$

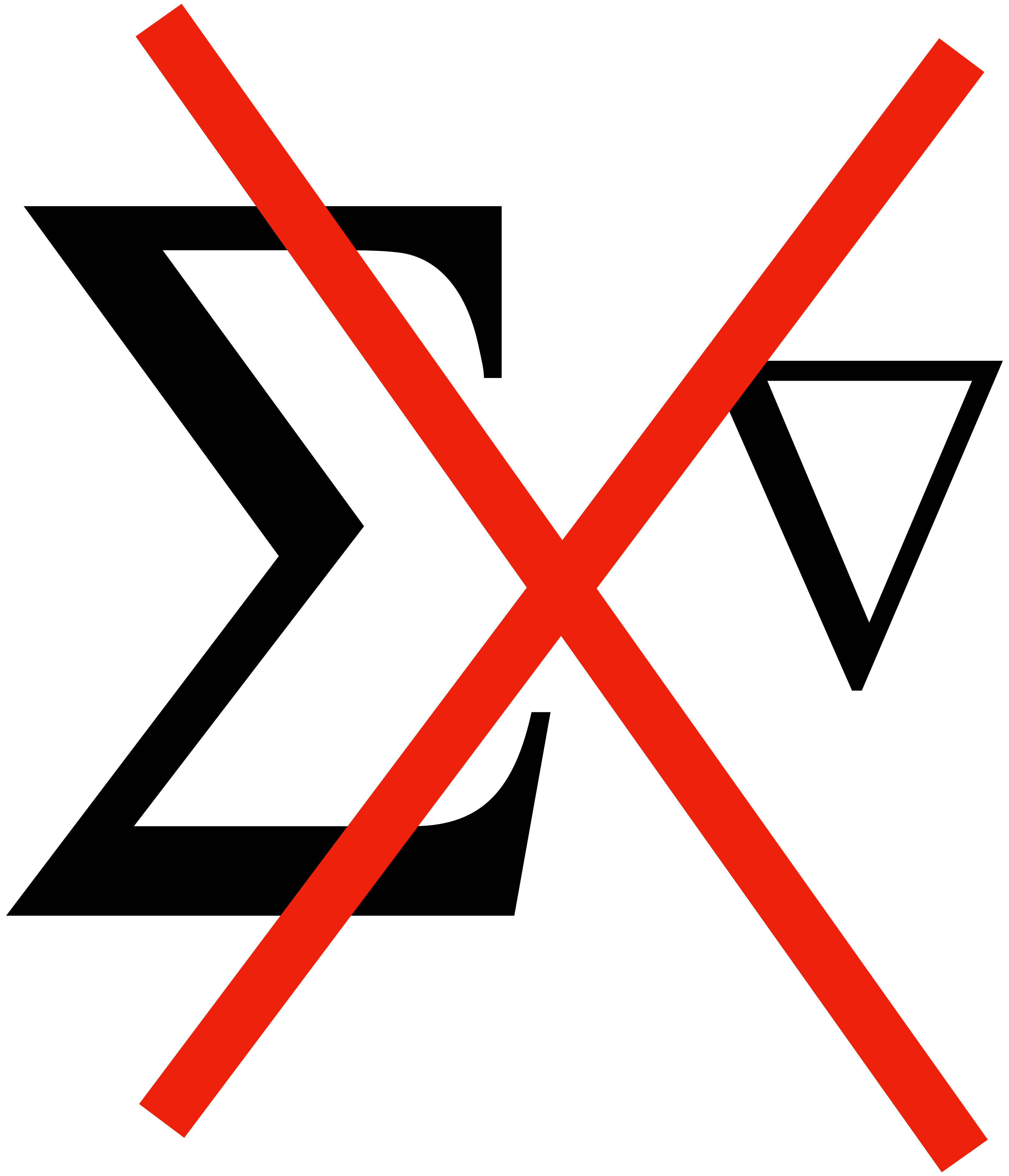


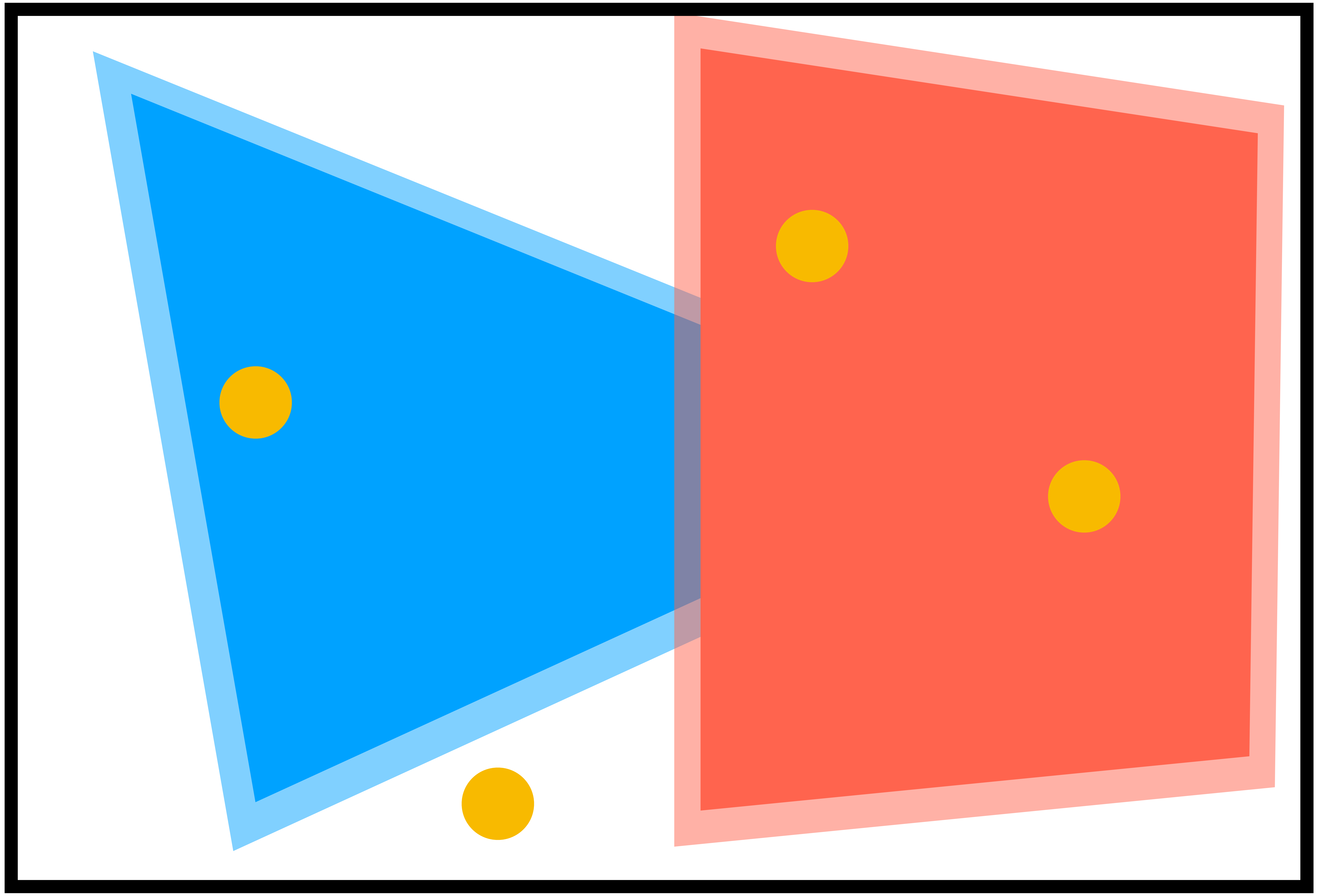
=the change of pixel color after we scale the shapes

Area sampling misses the boundaries

automatic differentiation can't compute correct gradients!

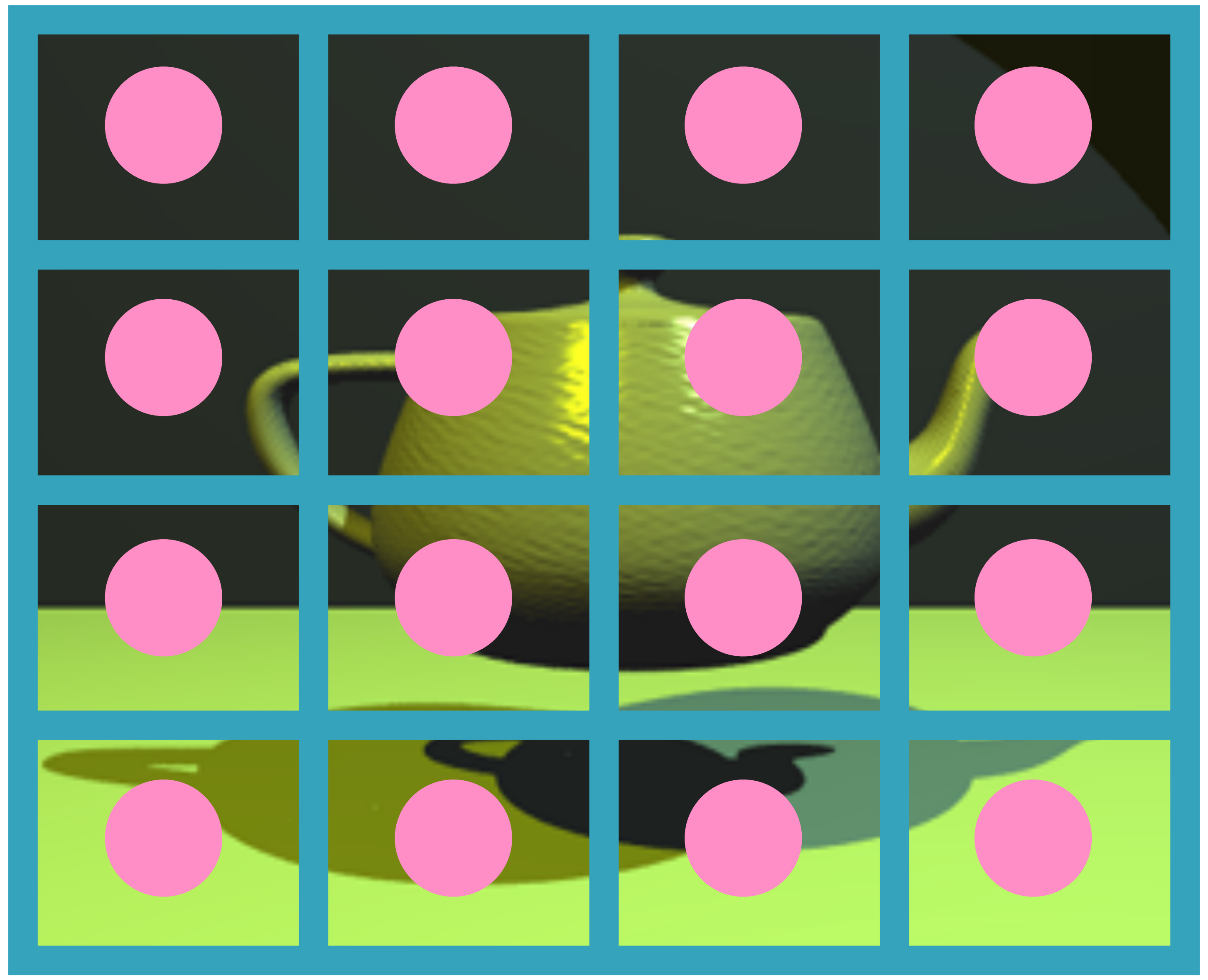
single pixel

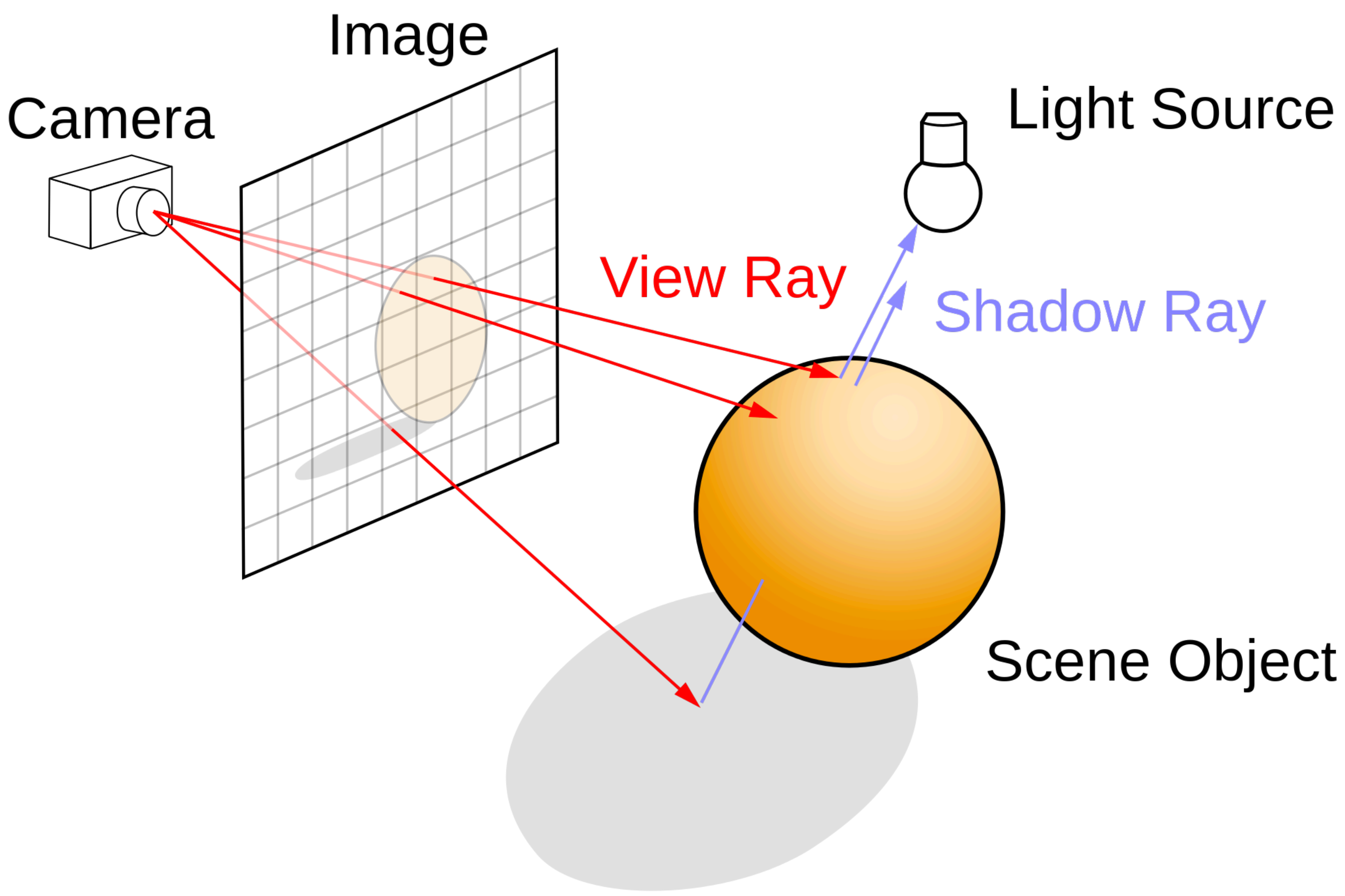




Existing techniques are all based on area sampling

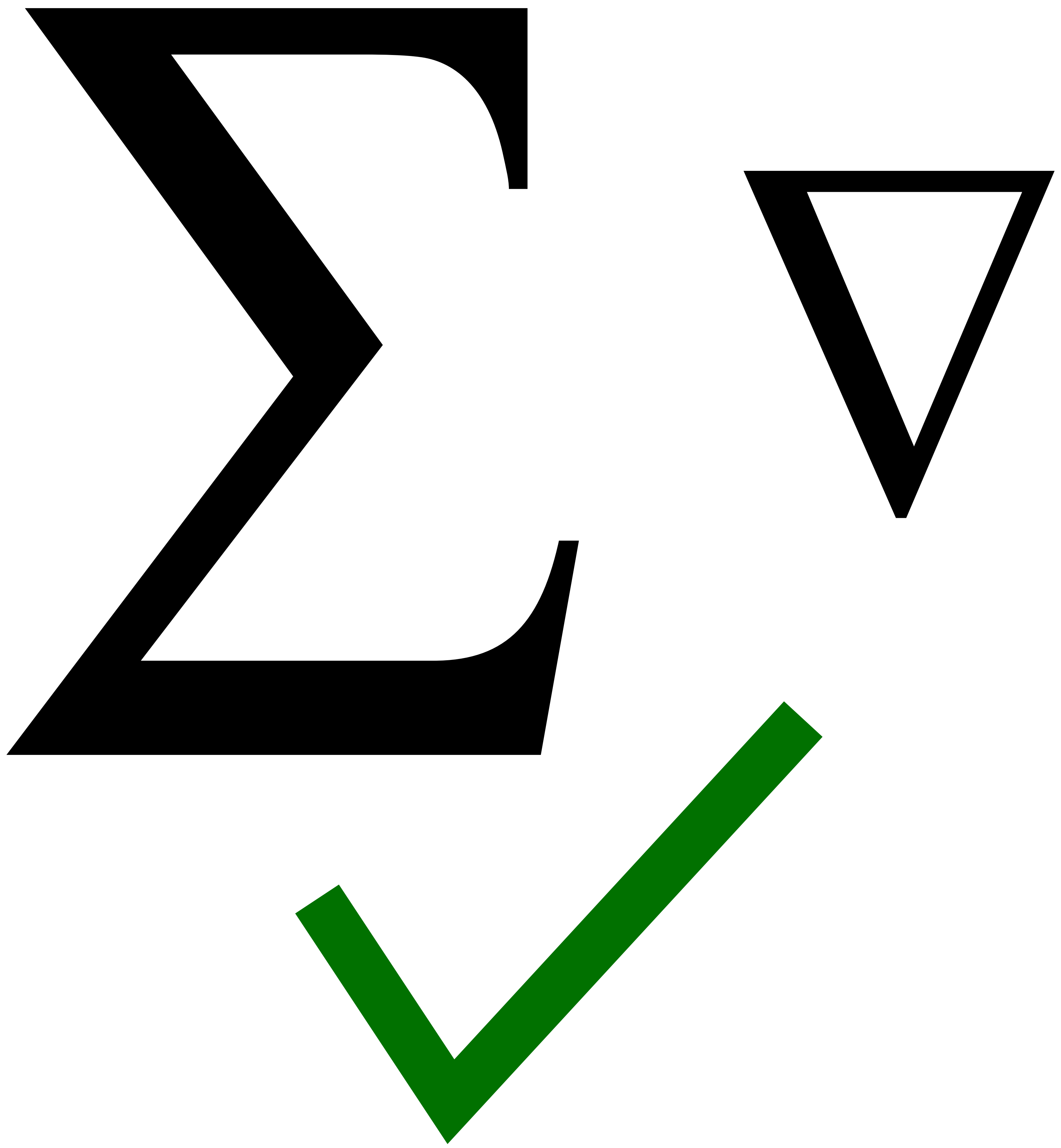
need a fundamentally different method

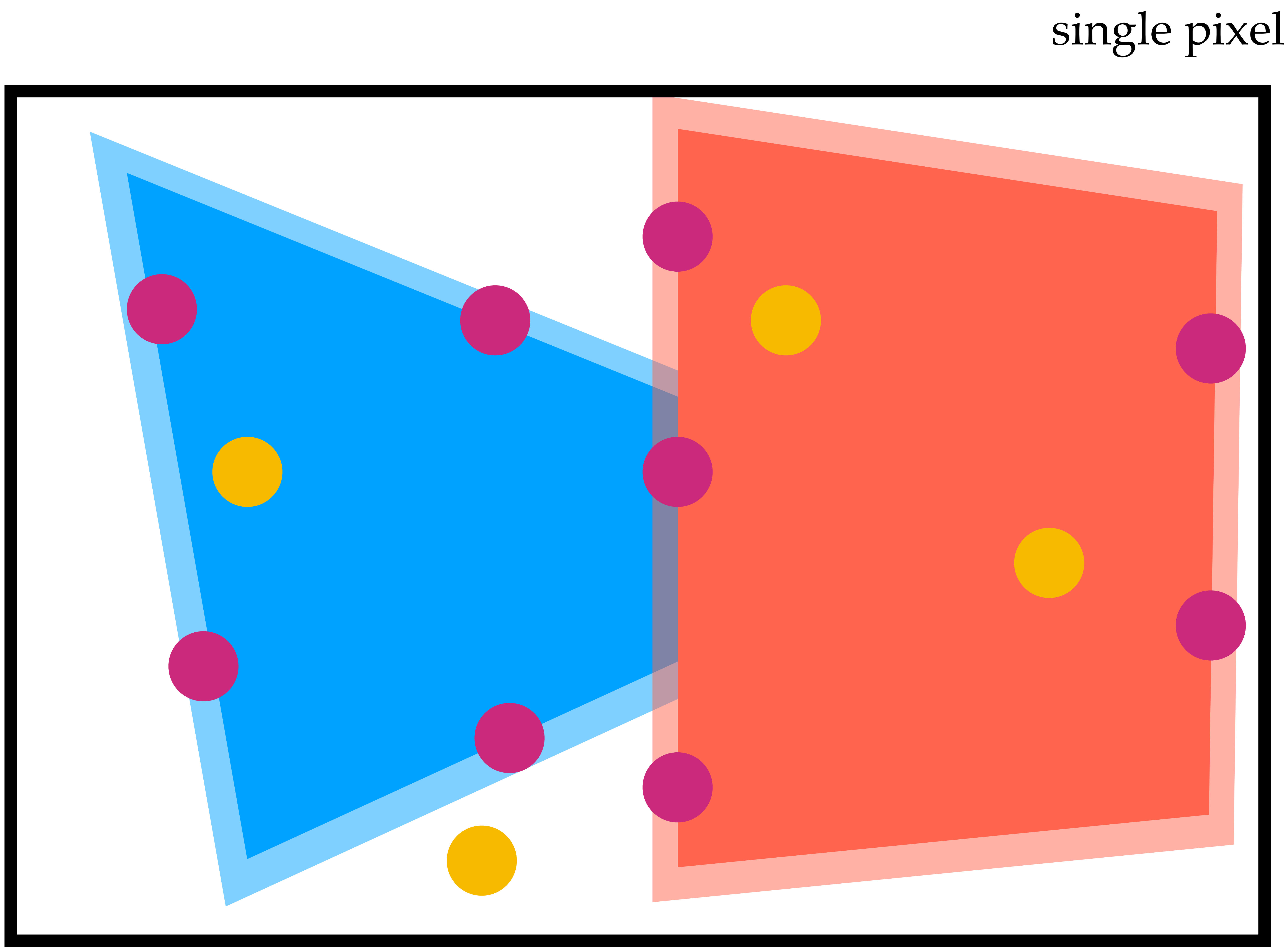




https://en.wikipedia.org/wiki/Utah_teapot#/media/File:Utah_teapot.png
https://en.wikipedia.org/wiki/Ray_tracing_(graphics)#/media/File:Ray_trace_diagram.svg

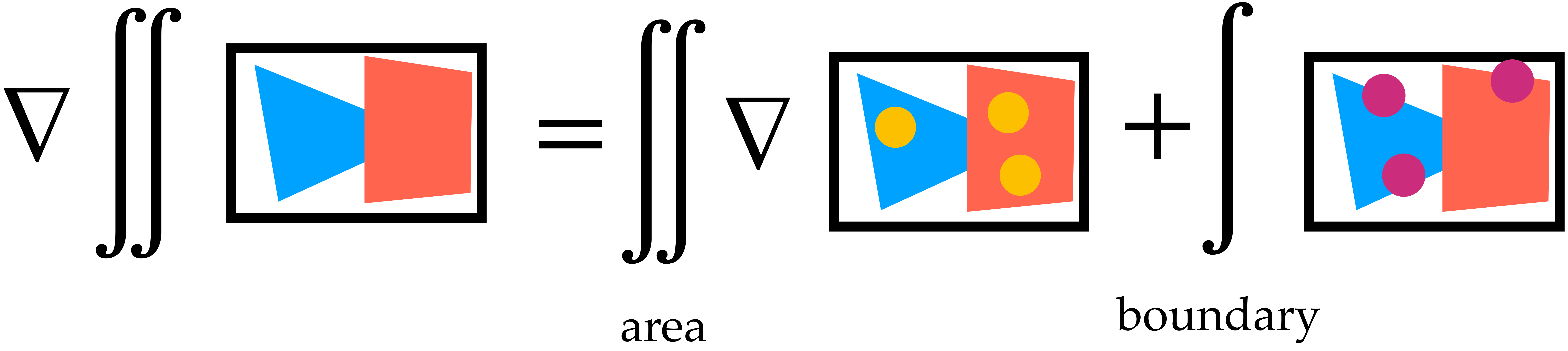
Our key idea: integrate over the boundaries





__Li__, Aittala, Durand, Lehtinen, 2018

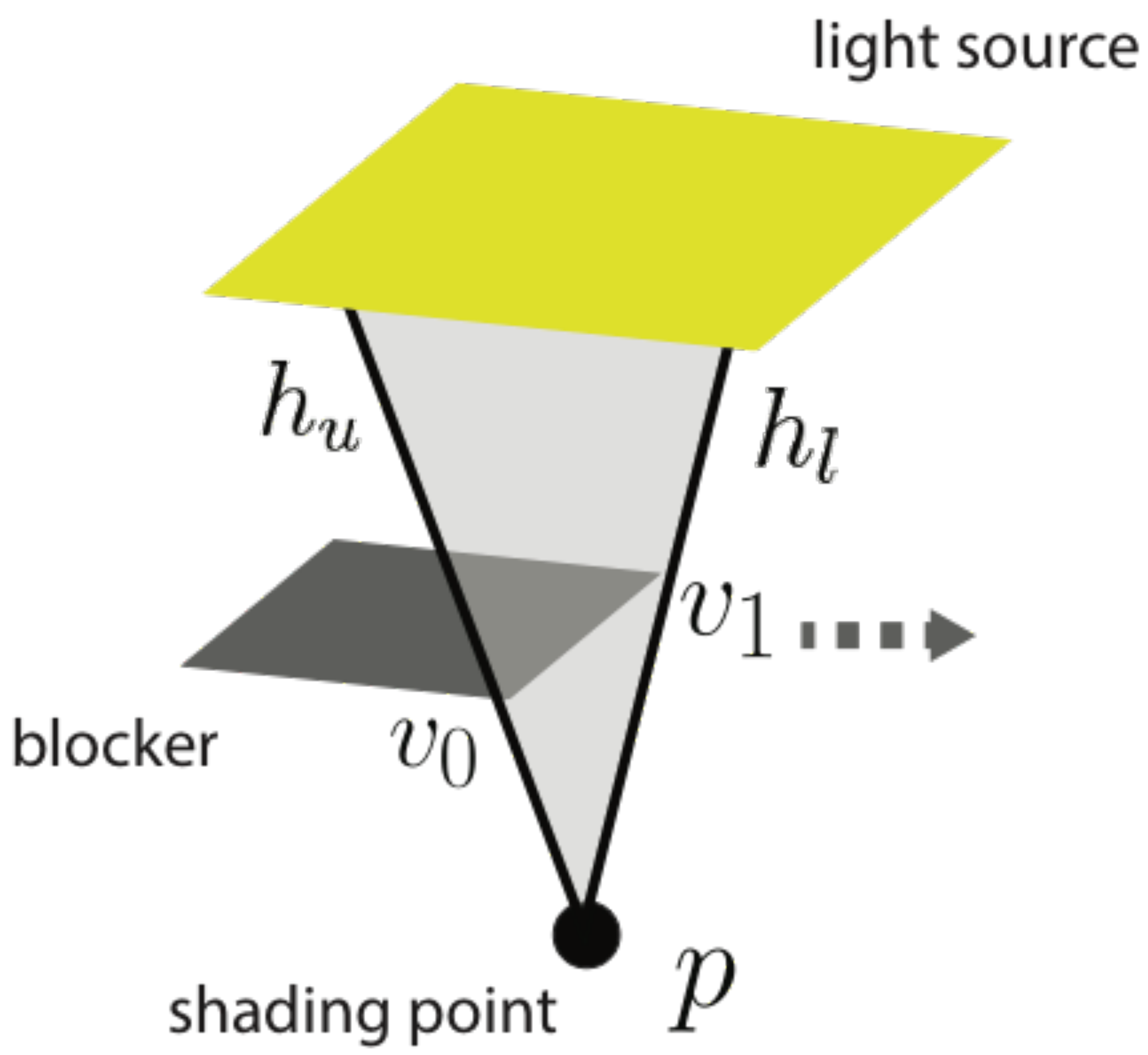
Mathematically

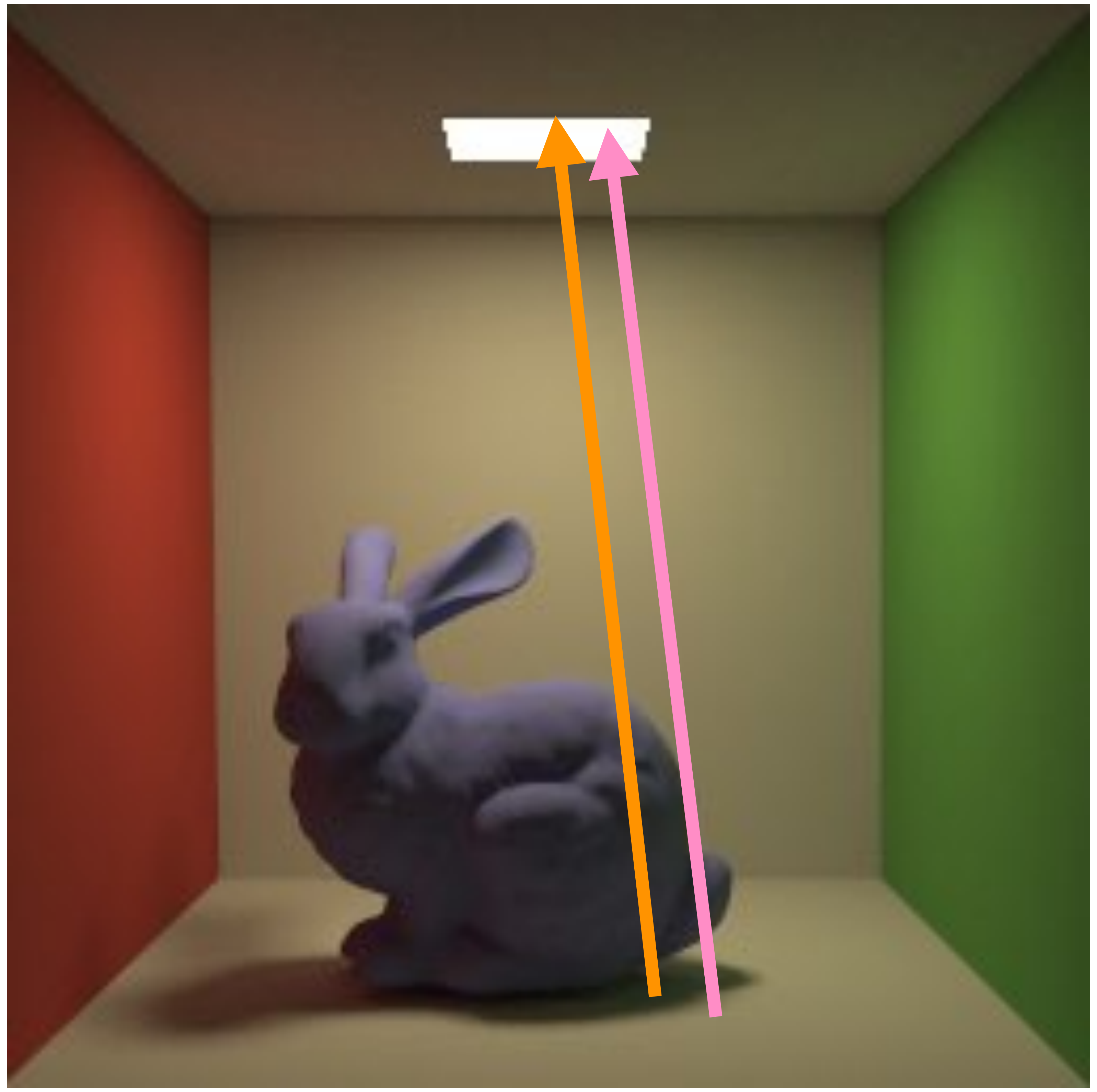


derived through Dirac delta or Reynolds transport theorem

Generalize to shadow & interreflection

integrating a different domain





Challenge: scalability

sampling triangle edges requires different acceleration data structures

0.15M triangles

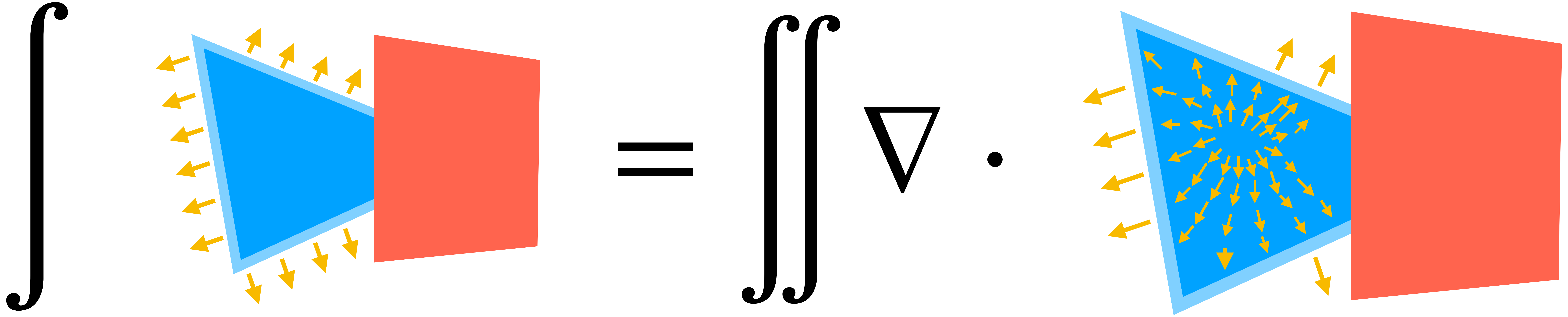
0.8M triangles

4.8M triangles

https://casual-effects.com/data/
https://benedikt-bitterli.me/resources/

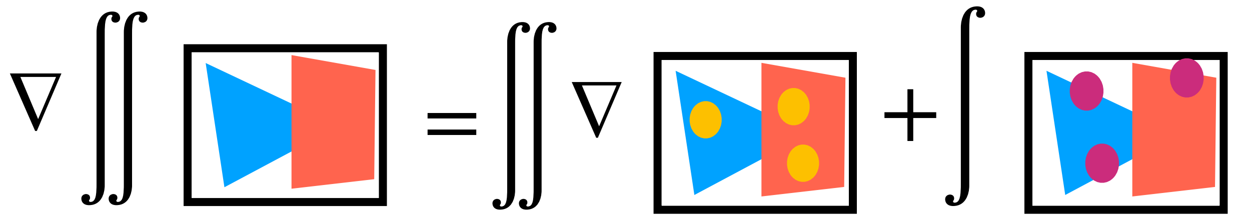
We can convert boundary integrals back to area integrals (divergence theorem)

we can then reuse data structures from traditional rendering



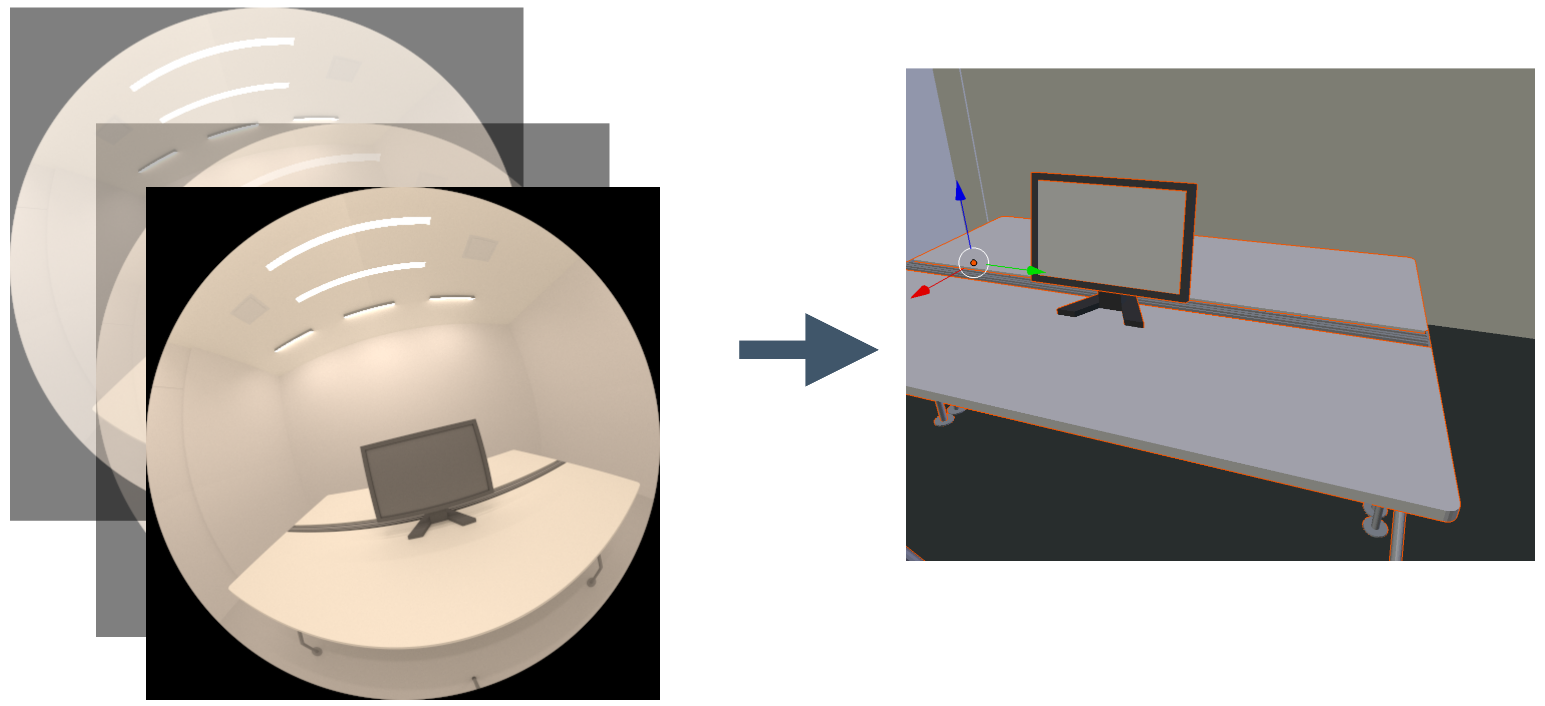
boundary

area



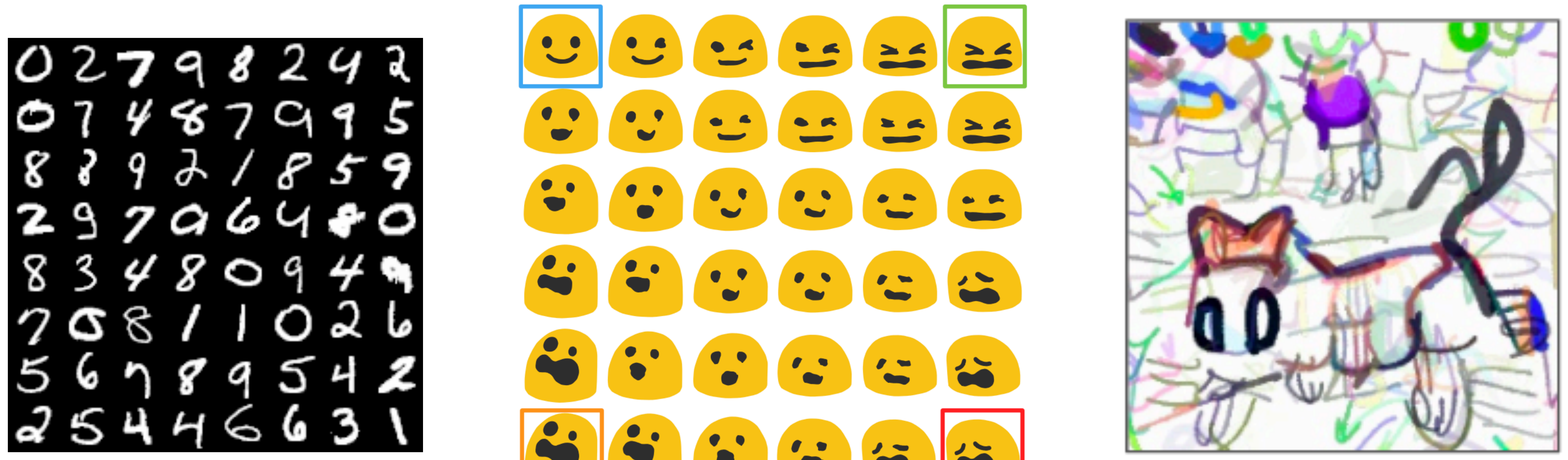
*Bangaru, **Li**, Durand, 2020*

Applications



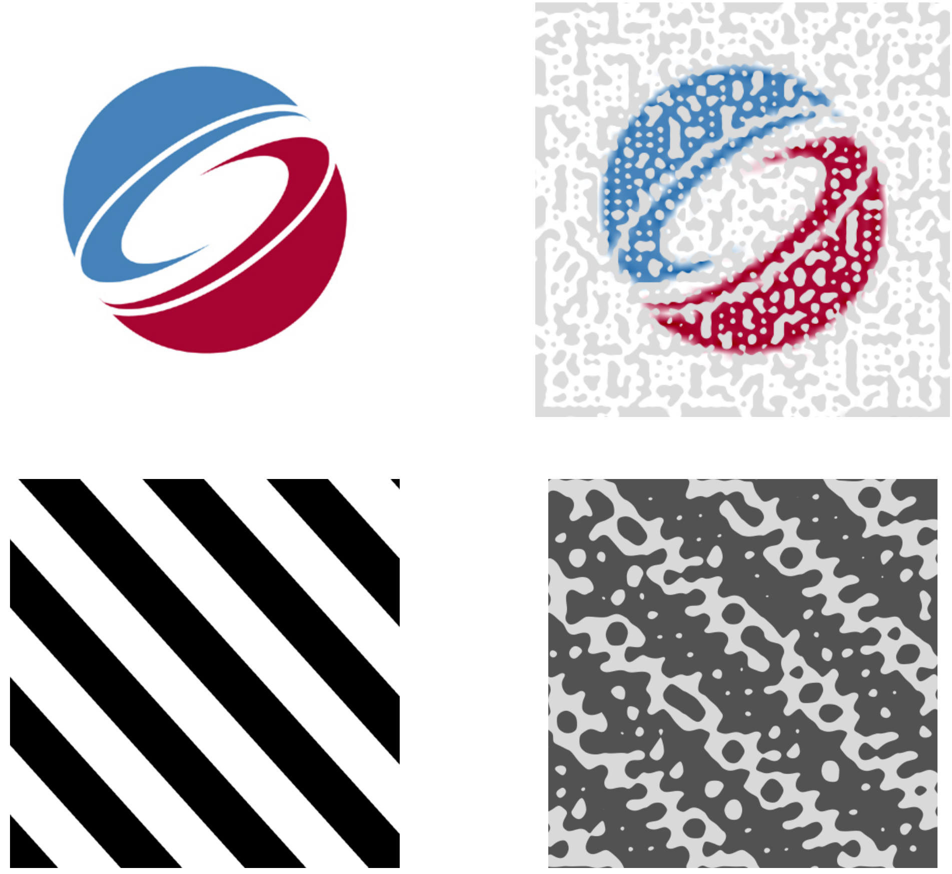
inverse rendering

analyzing vision systems



"A drawing of a cat".

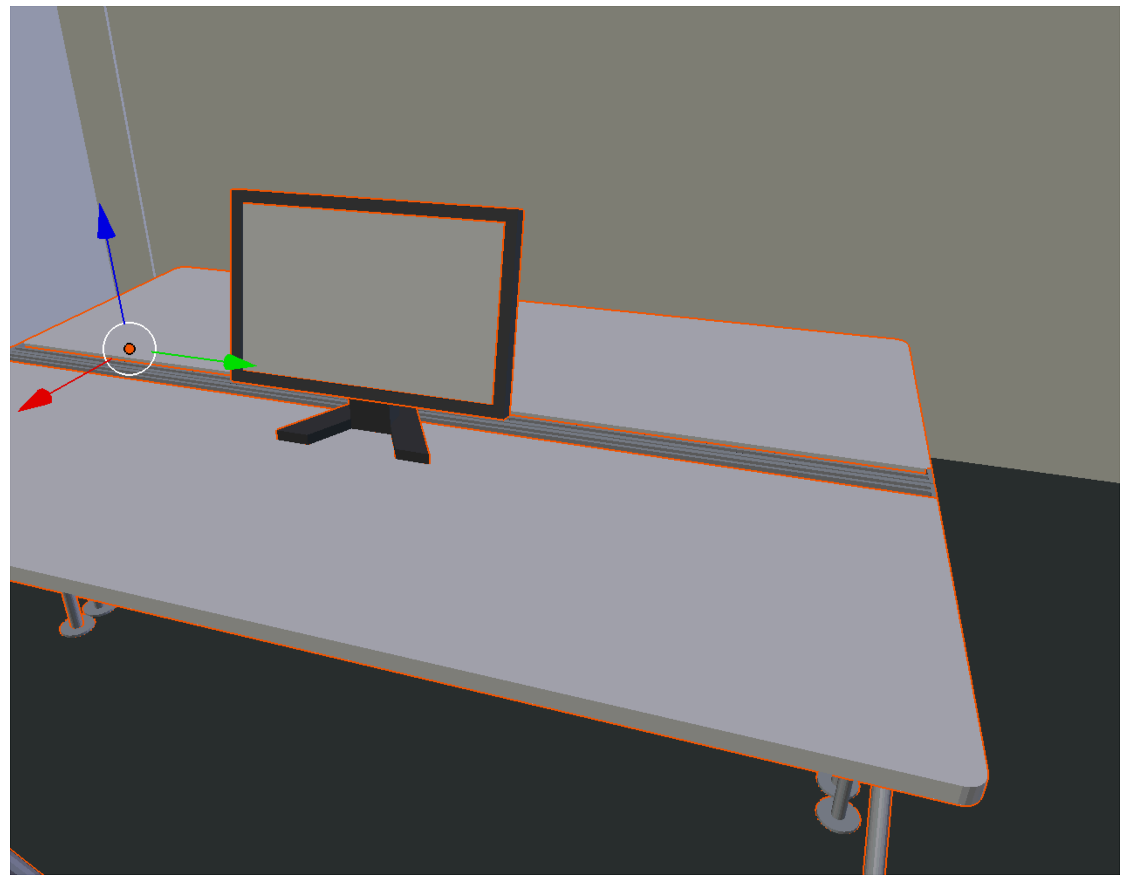
bridging vector and raster graphics



inverse shader design

Inverse rendering

differentiable rendering unifies 3D reconstruction



- geometry
- material
- light
- camera

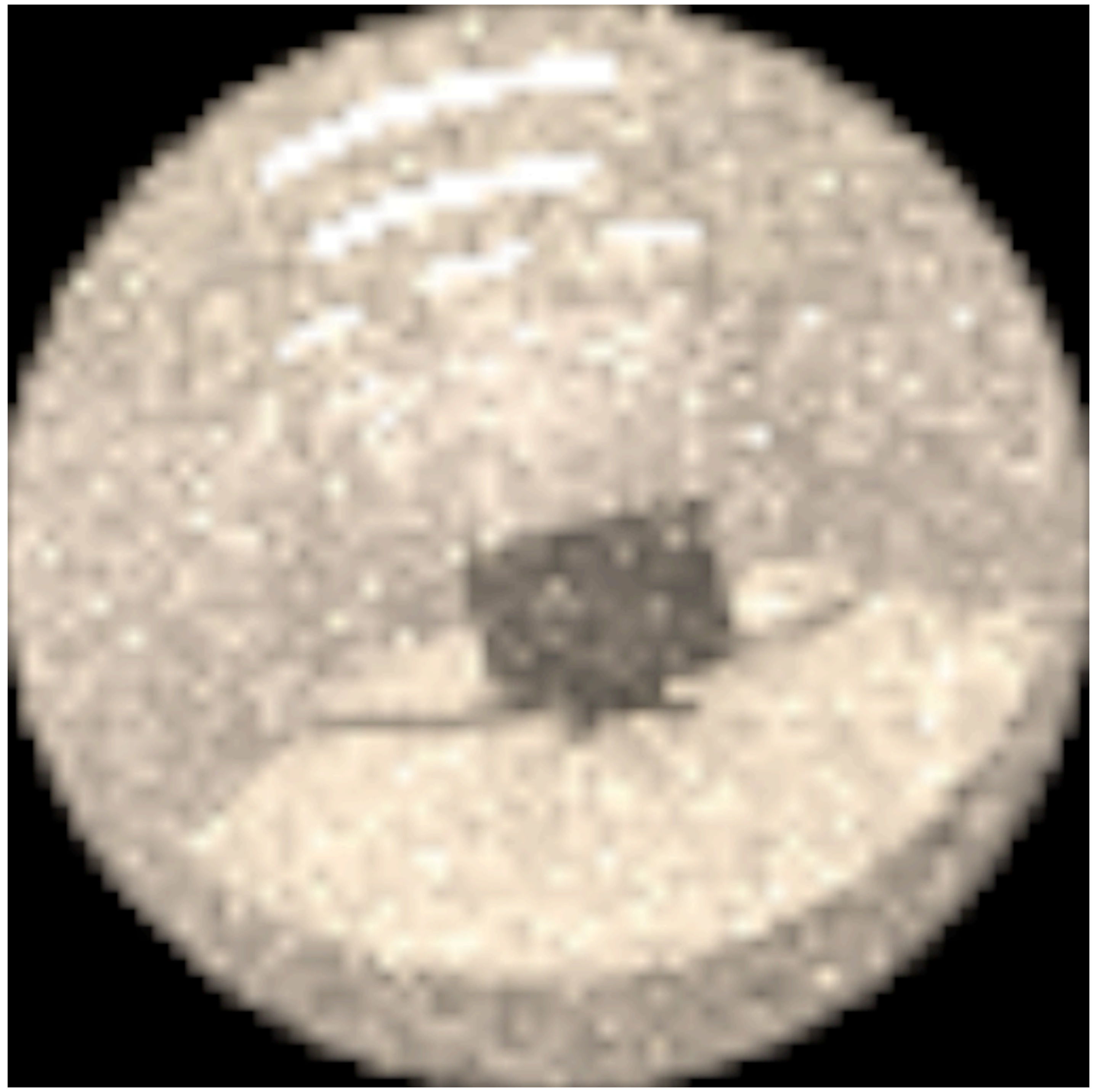
Inverse rendering with real photos

optimize camera pose, light emission & materials

initial guess   target   reconstructed

Inverse rendering with real photos

optimize camera pose, light emission & materials

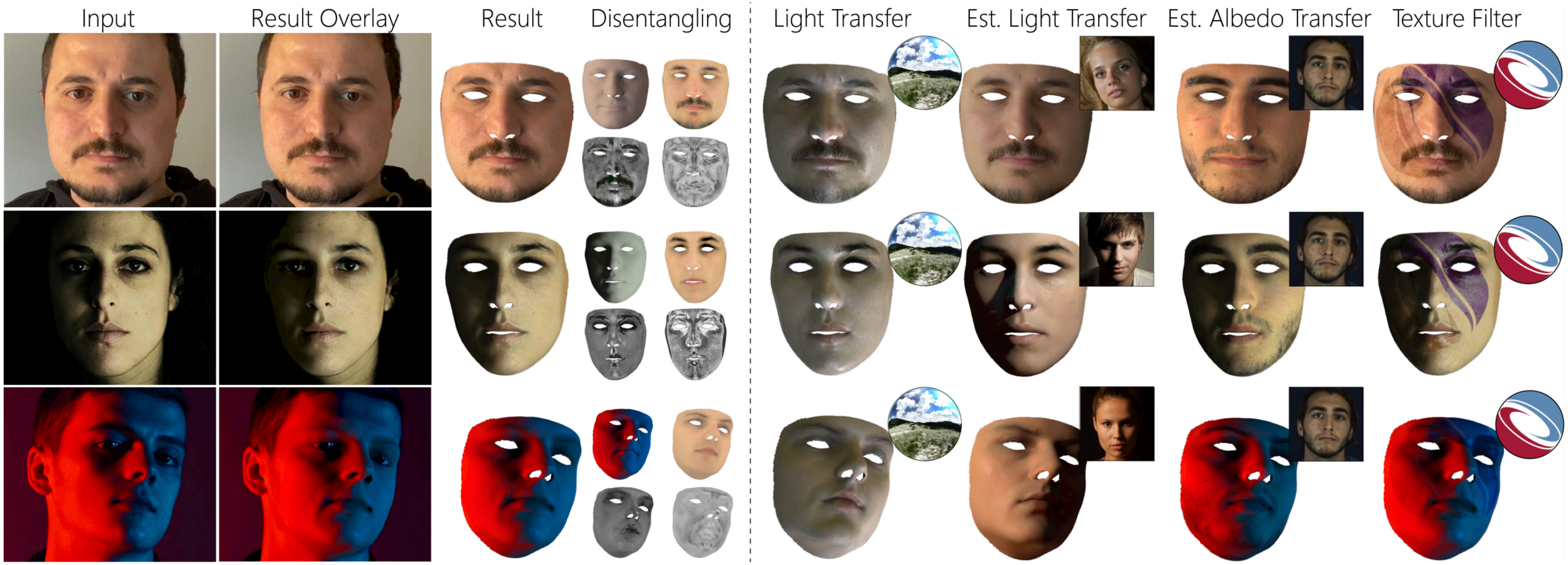


optimization video

target

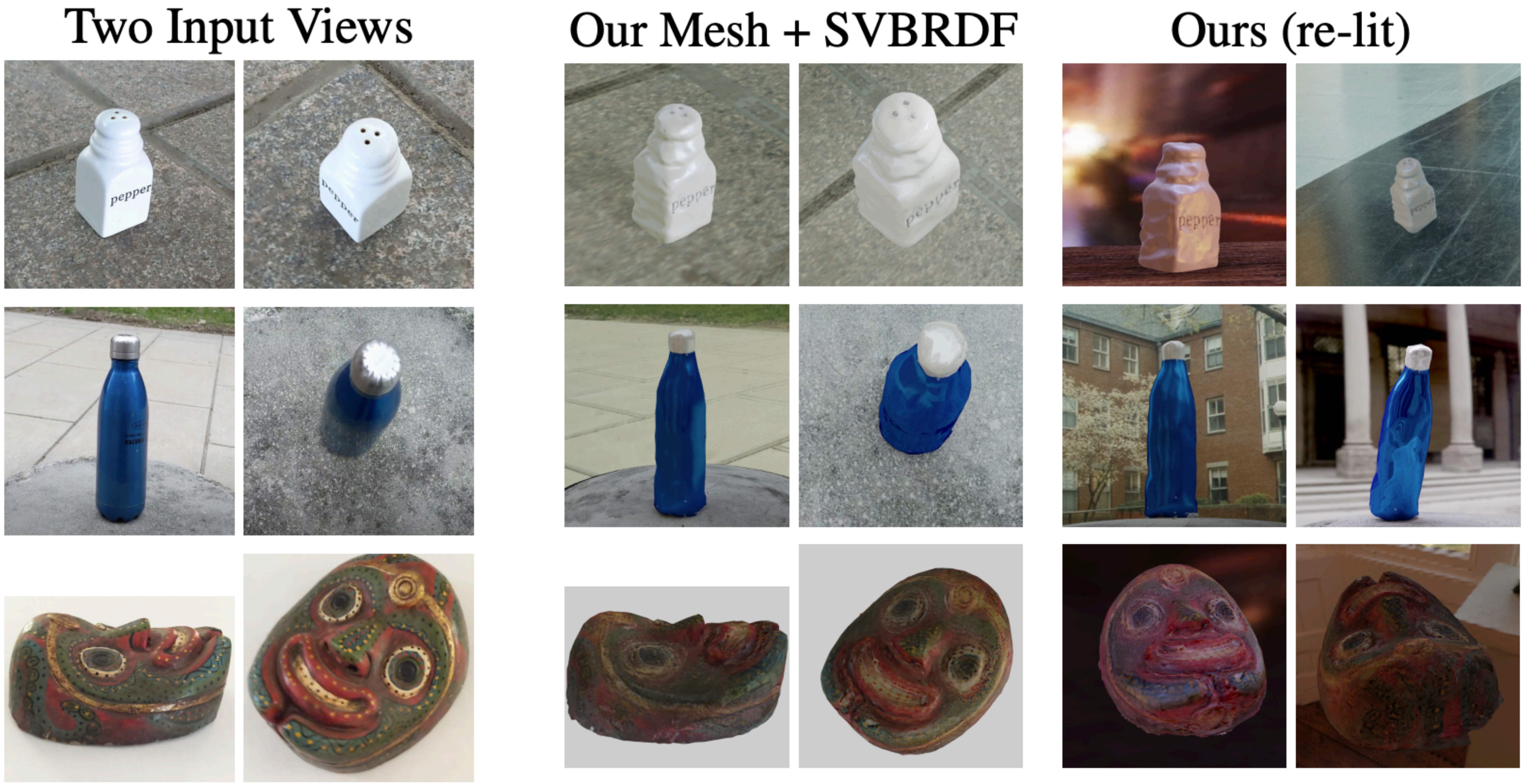
Face reconstruction

from Dib et al., using our differentiable renderer (not my work!)



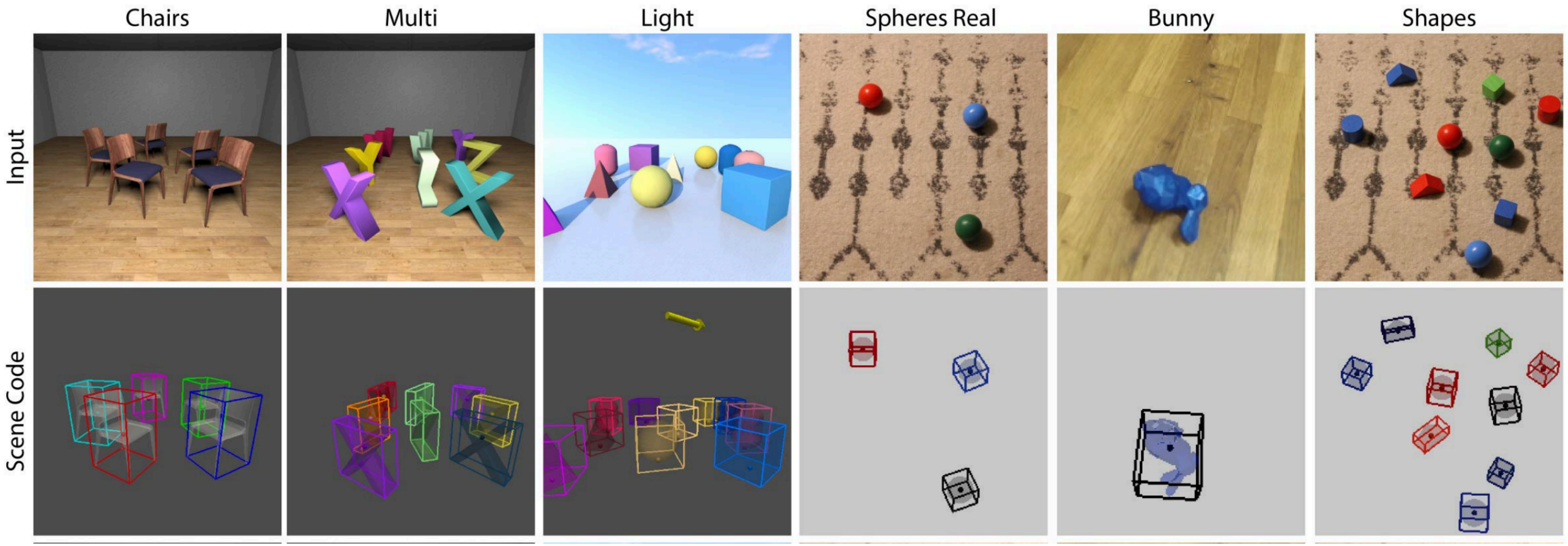
Multi-view 3D reconstruction

from Goel et al., using our differentiable renderer (not my work!)

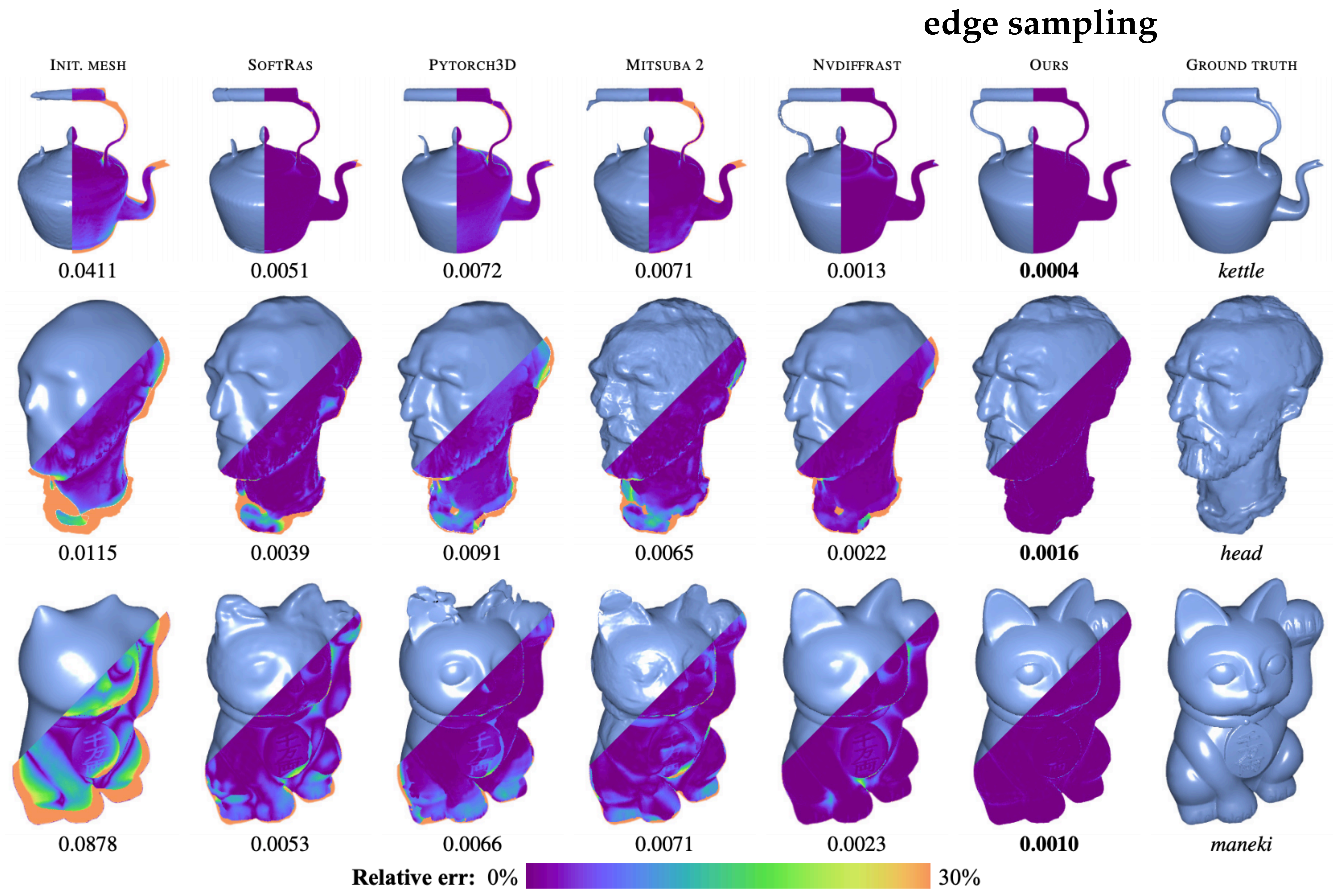


Learning to generate 3D scenes

from Griffiths et al., using our differentiable renderer (not my work!)

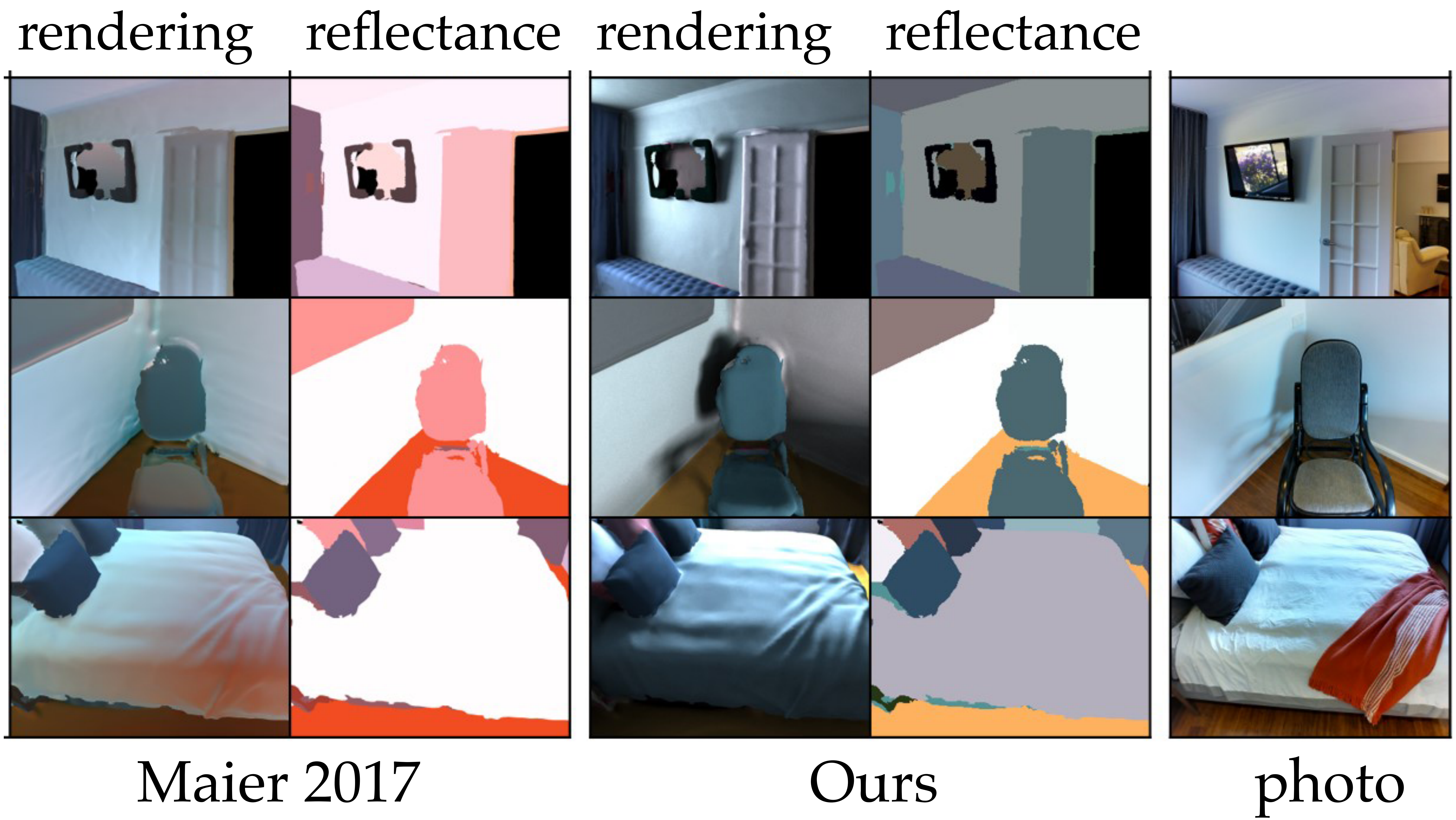


Having correct gradients matters



from Luan et al. 2021
(not my work!)

Better lighting model = better reconstruction



*Azinović, **Li**, Kaplanyan, Nießner, 2019*

Augmented reality application

3D reconstruction is useful for editing images

real photo

edited

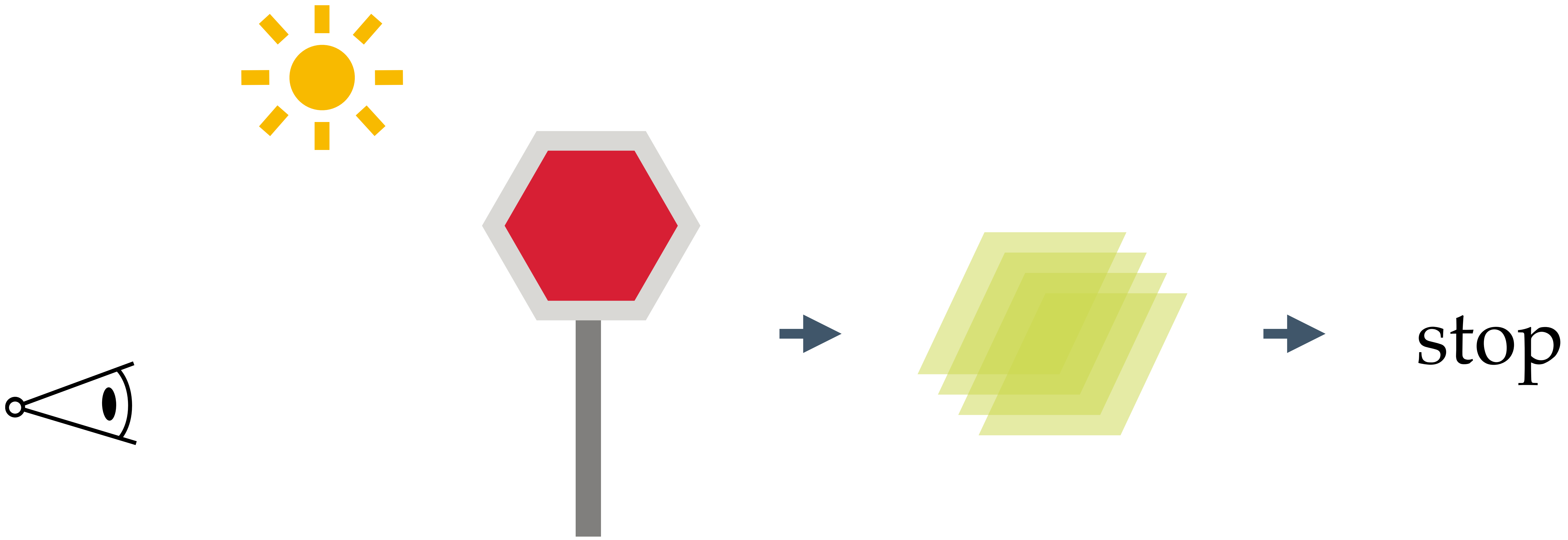
*Azinović, **Li**, Kaplanyan, Nießner, 2019*

Analyzing the behavior of vision systems

differentiable rendering allows us to ask 3D questions

Analyzing the behavior of vision systems

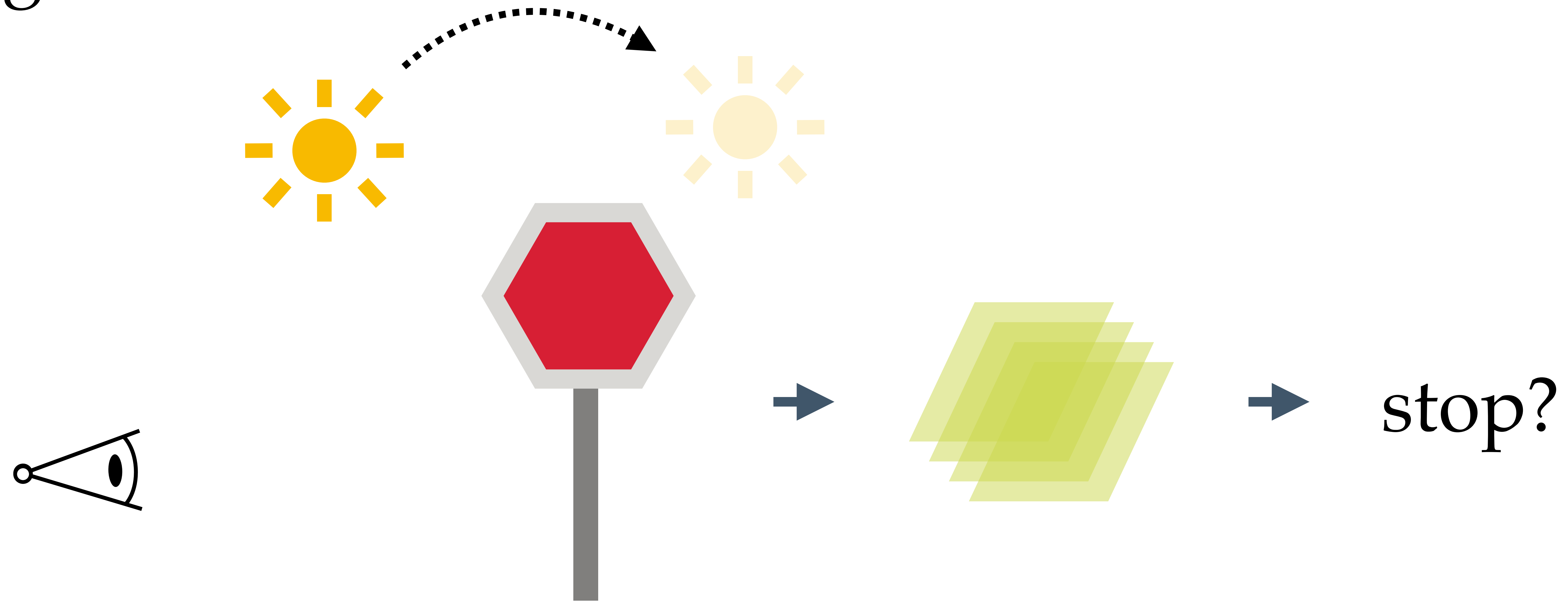
differentiable rendering allows us to ask 3D questions



Analyzing the behavior of vision systems

differentiable rendering allows us to ask 3D questions

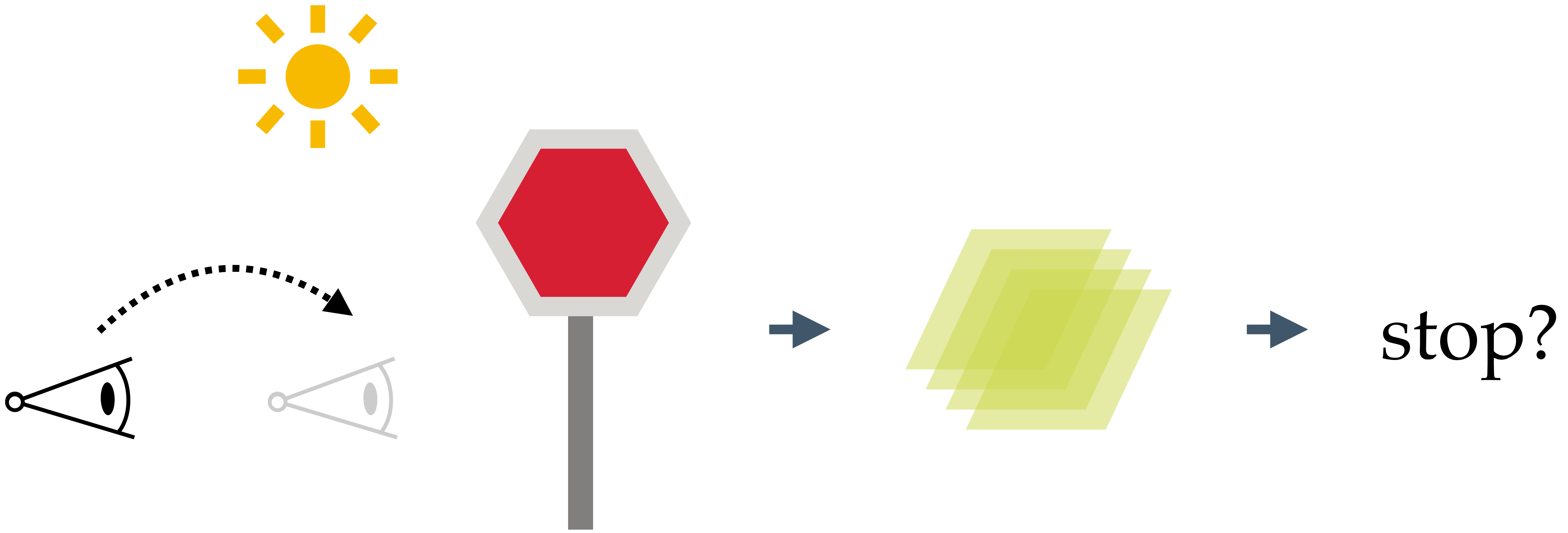
- lighting



Analyzing the behavior of vision systems

differentiable rendering allows us to ask 3D questions

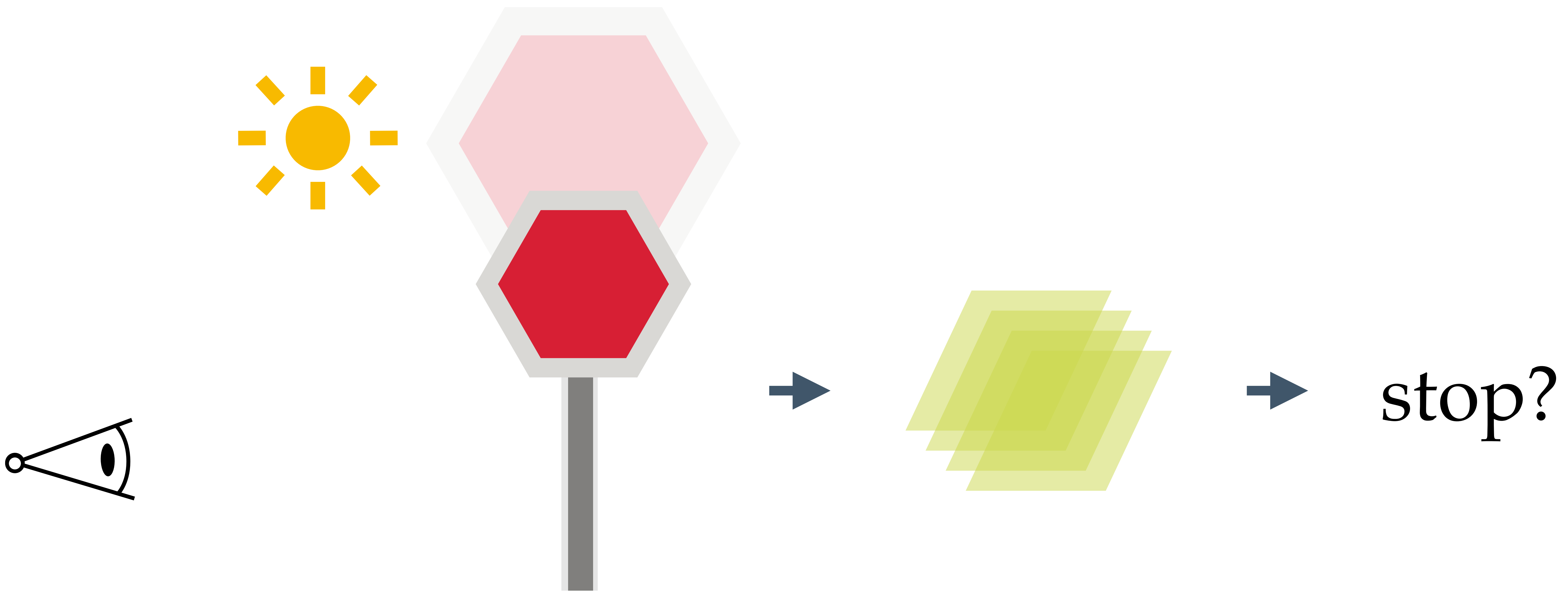
- lighting
- view



Analyzing the behavior of vision systems

differentiable rendering allows us to ask 3D questions

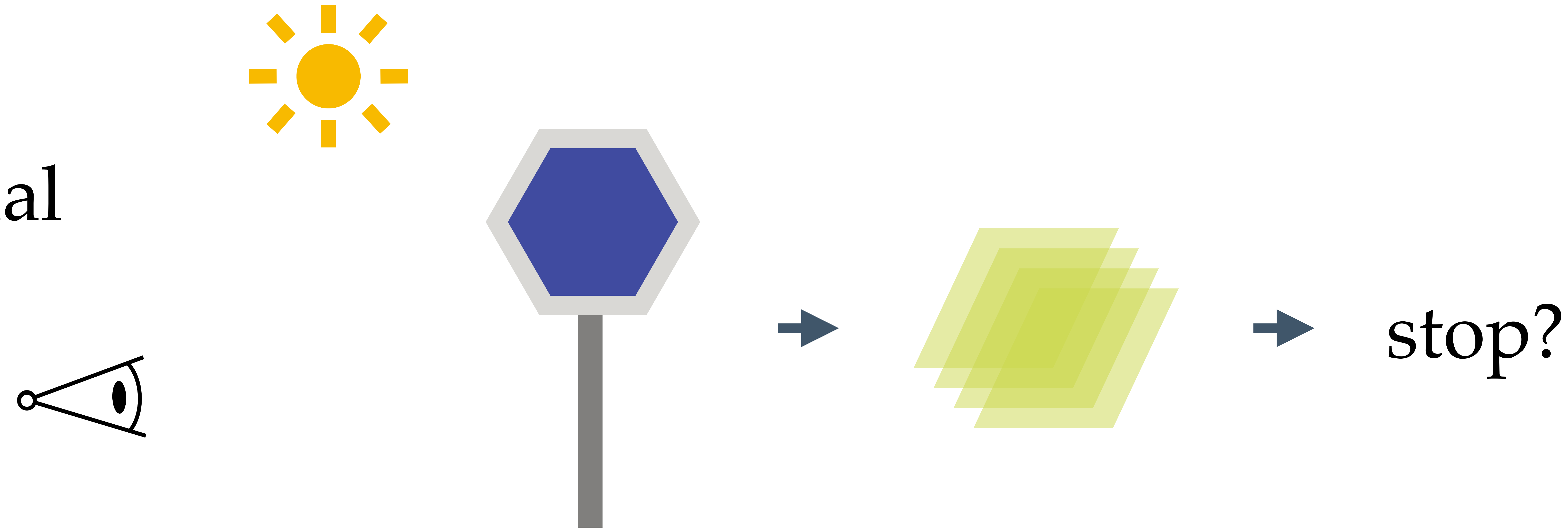
- lighting
- view
- shape



Analyzing the behavior of vision systems

differentiable rendering allows us to ask 3D questions

- lighting
- view
- shape
- material



3D adversarial examples

optimize for vertex position, camera pose,
light intensity, position

VGG 16:
53% street sign
6.7% handrail

3D adversarial examples

optimize for vertex position, camera pose,
light intensity, position

VGG 16:
53% street sign
6.7% handrail

5 iterations:
26.8% handrail
20.2% street sign

3D adversarial examples

optimize for vertex position, camera pose,
light intensity, position

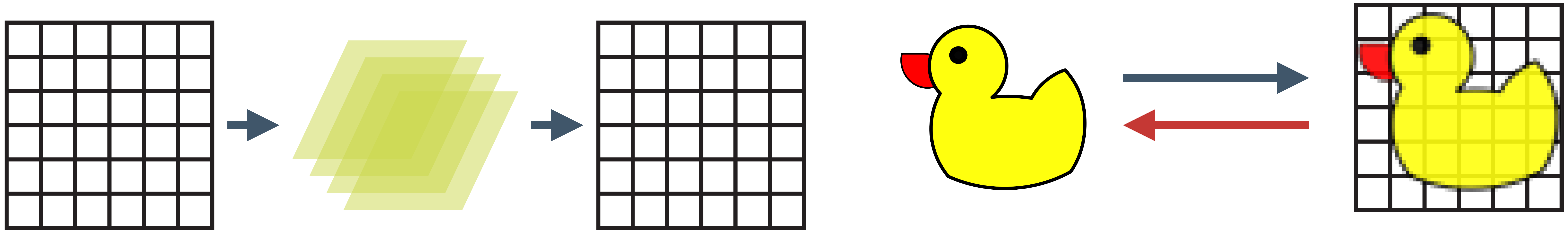
VGG 16:
53% street sign
6.7% handrail

5 iterations:
26.8% handrail
20.2% street sign

25 iterations:
23.3% handrail
3.4% street sign

Bridging vector and raster graphics

differentiable rendering enables
neural network operations on vector graphics



__Li__, Lukáč, Gharbi, Ragan-Kelley, 2020

Bridging vector and raster graphics

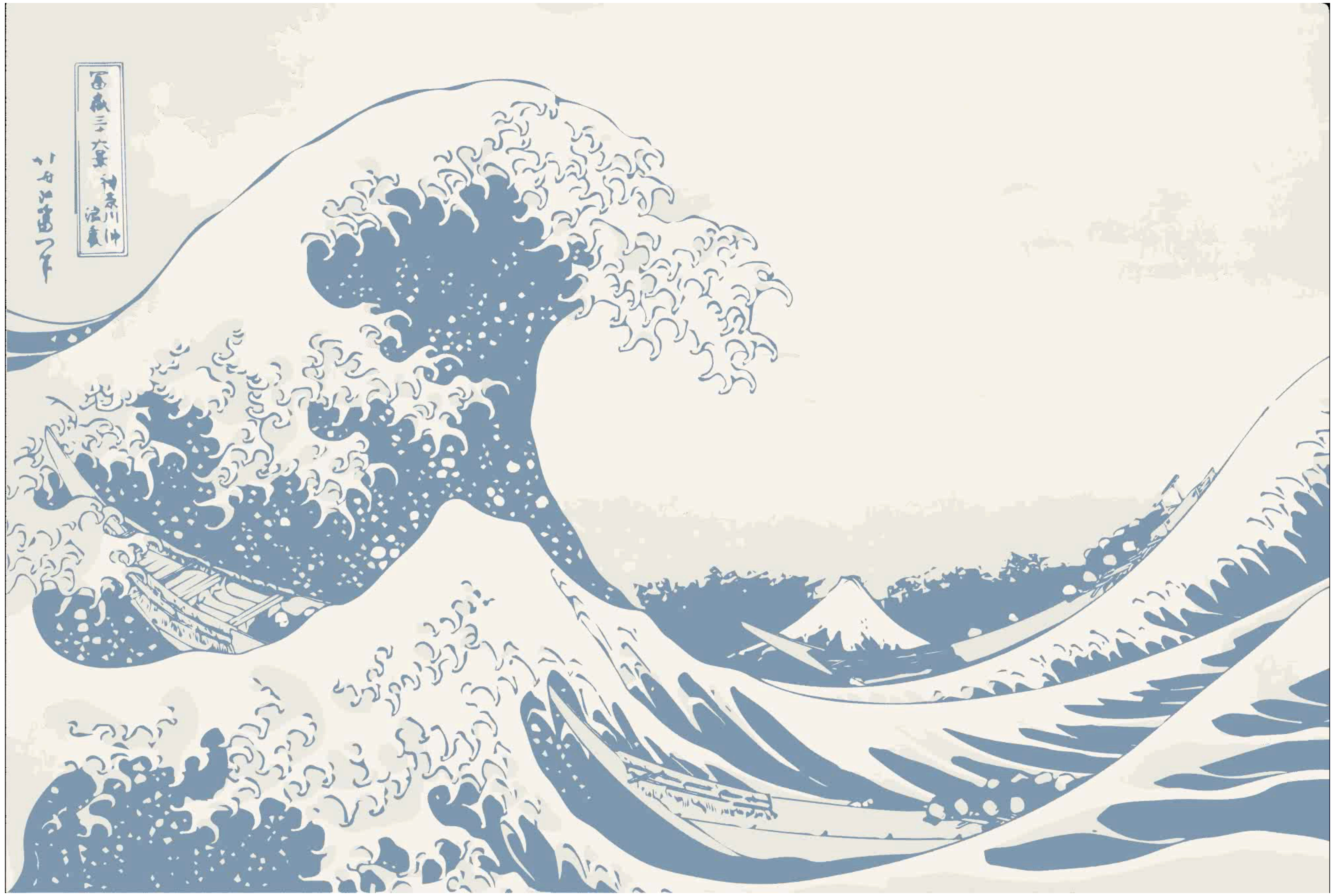
painterly rendering

ours

target

Bridging vector and raster graphics

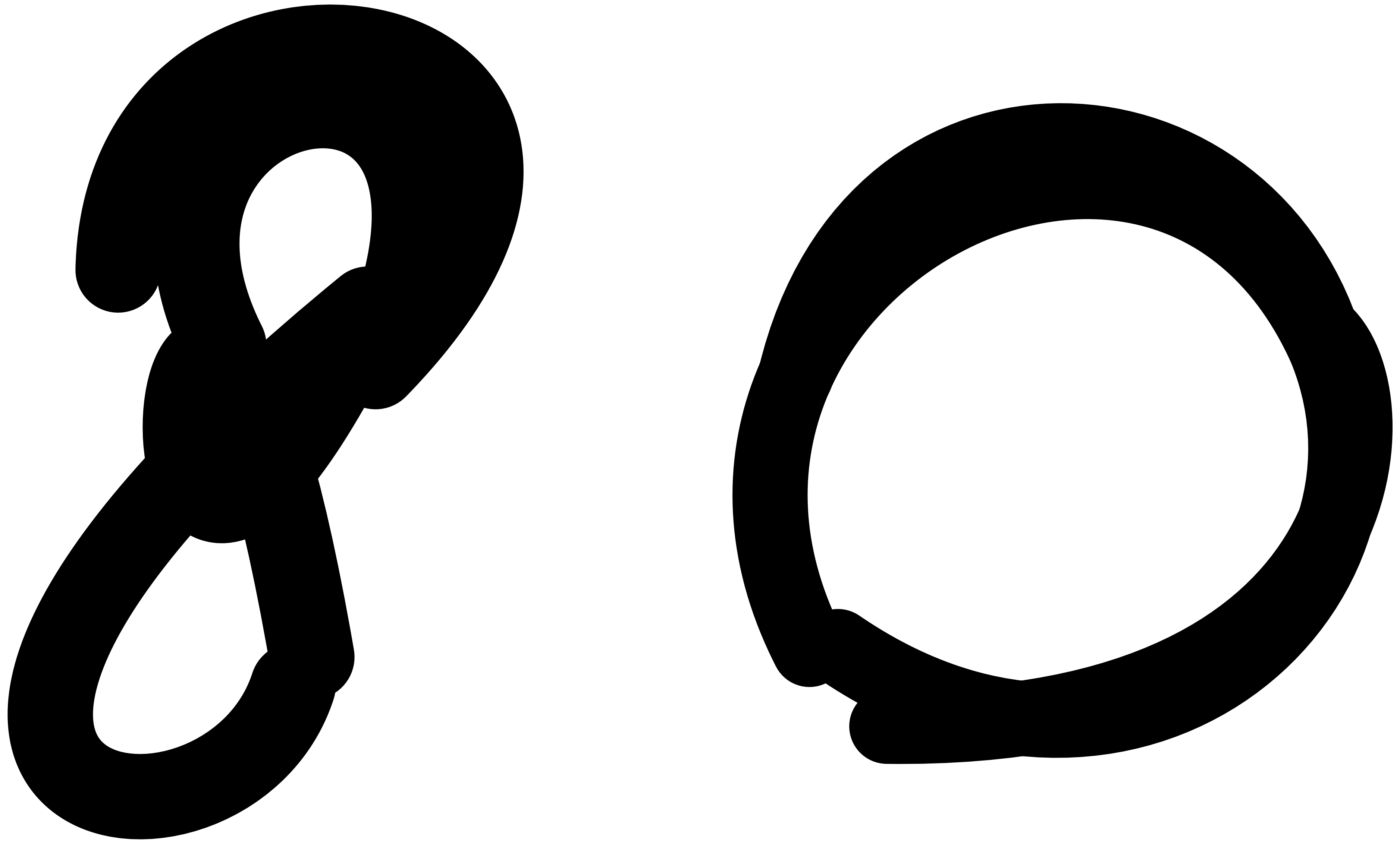
image processing for vector graphics



Bridging vector and raster graphics

unsupervised learning for vector graphics

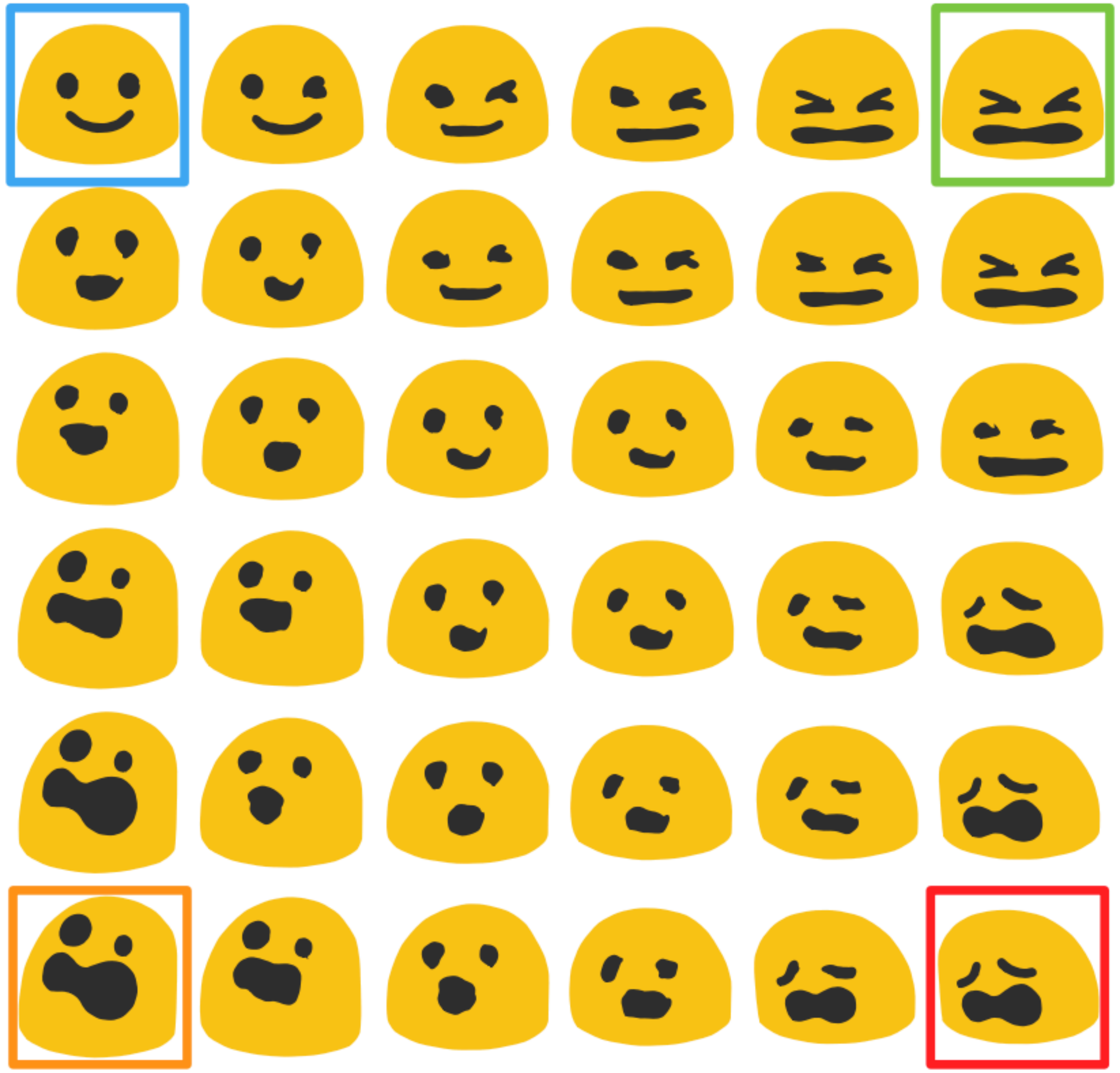
raster training data
(MNIST)



our vector output

Bridging vector and raster graphics

unsupervised learning for vector graphics



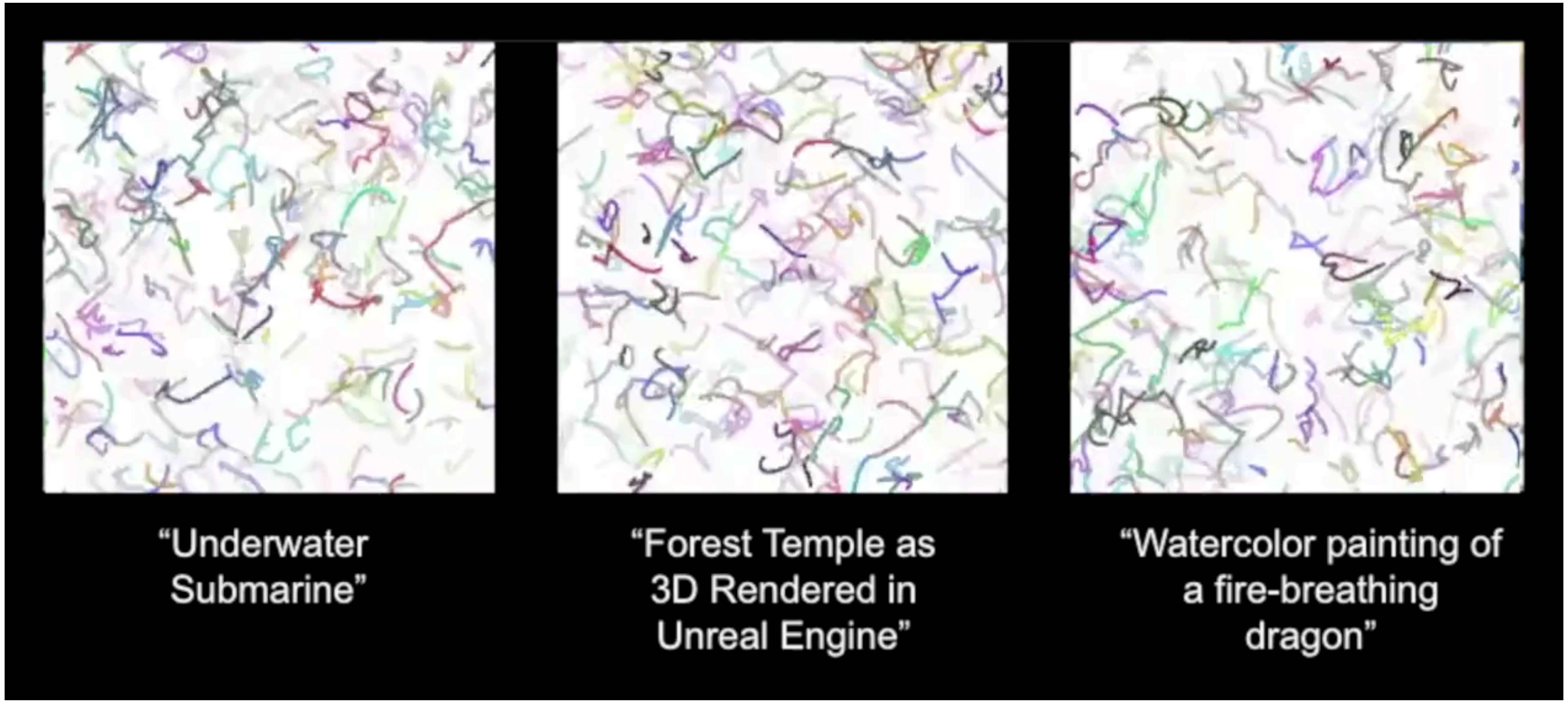
from Reddy et al.,
interpolating emojis using our renderer

(not my work!)

CLIPDraw [Frans et al.]

(not my work!)

- text to drawing using CLIP & differentiable rasterization using our renderer

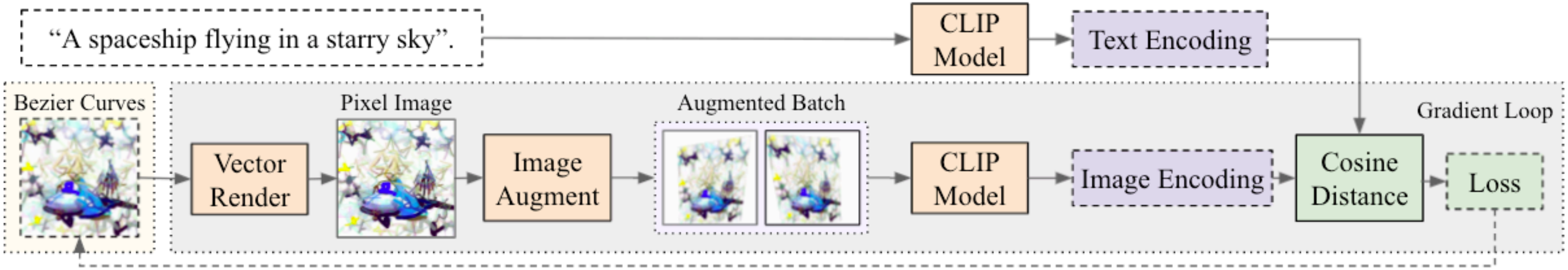


https://kvfrans.com/clipdraw-exploring-text-to-drawing-synthesis/

CLIPDraw [Frans et al.]

(not my work!)

- text to drawing using CLIP & differentiable rasterization using our renderer

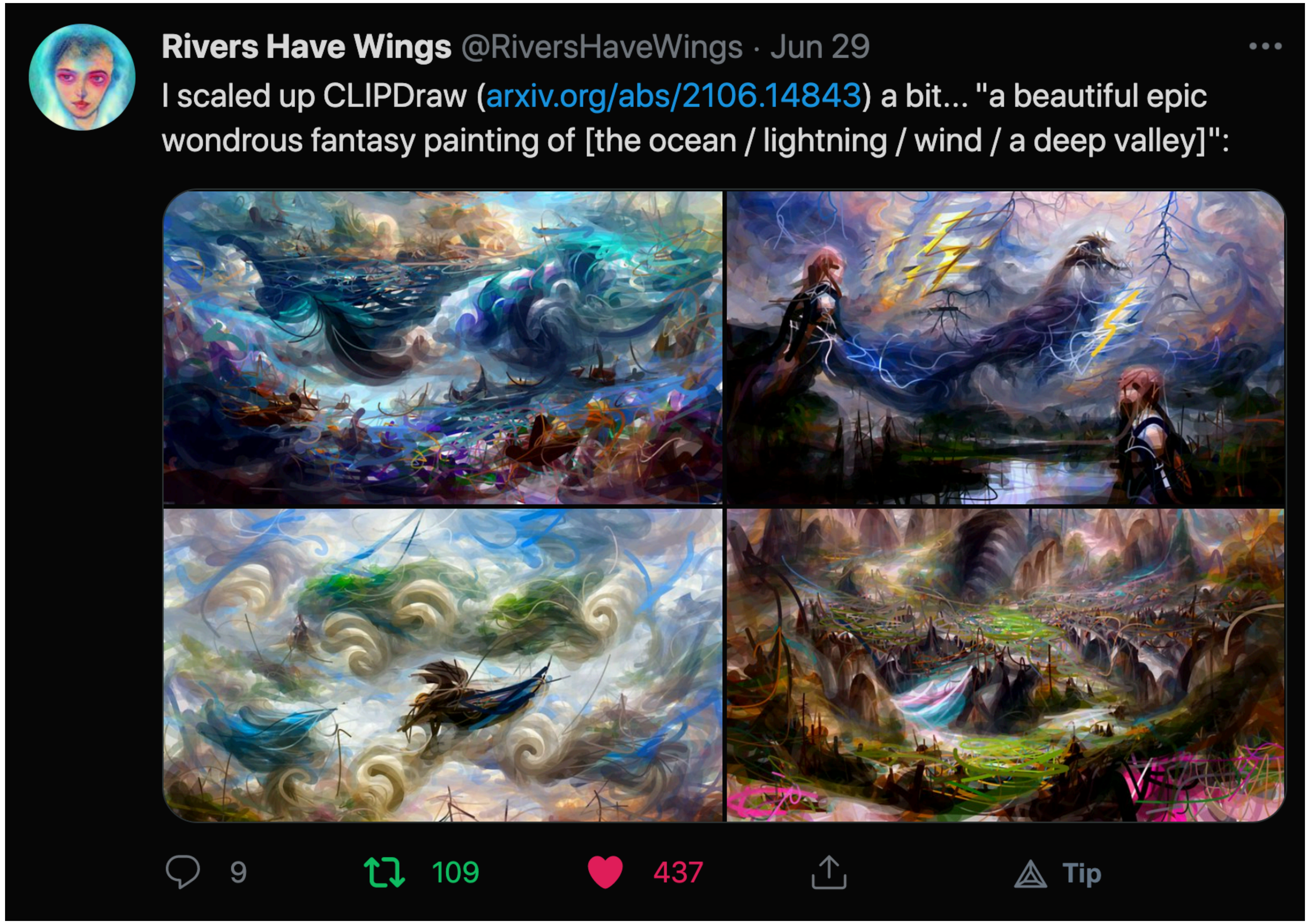


https://kvfrans.com/clipdraw-exploring-text-to-drawing-synthesis/

CLIPDraw [Frans et al.]

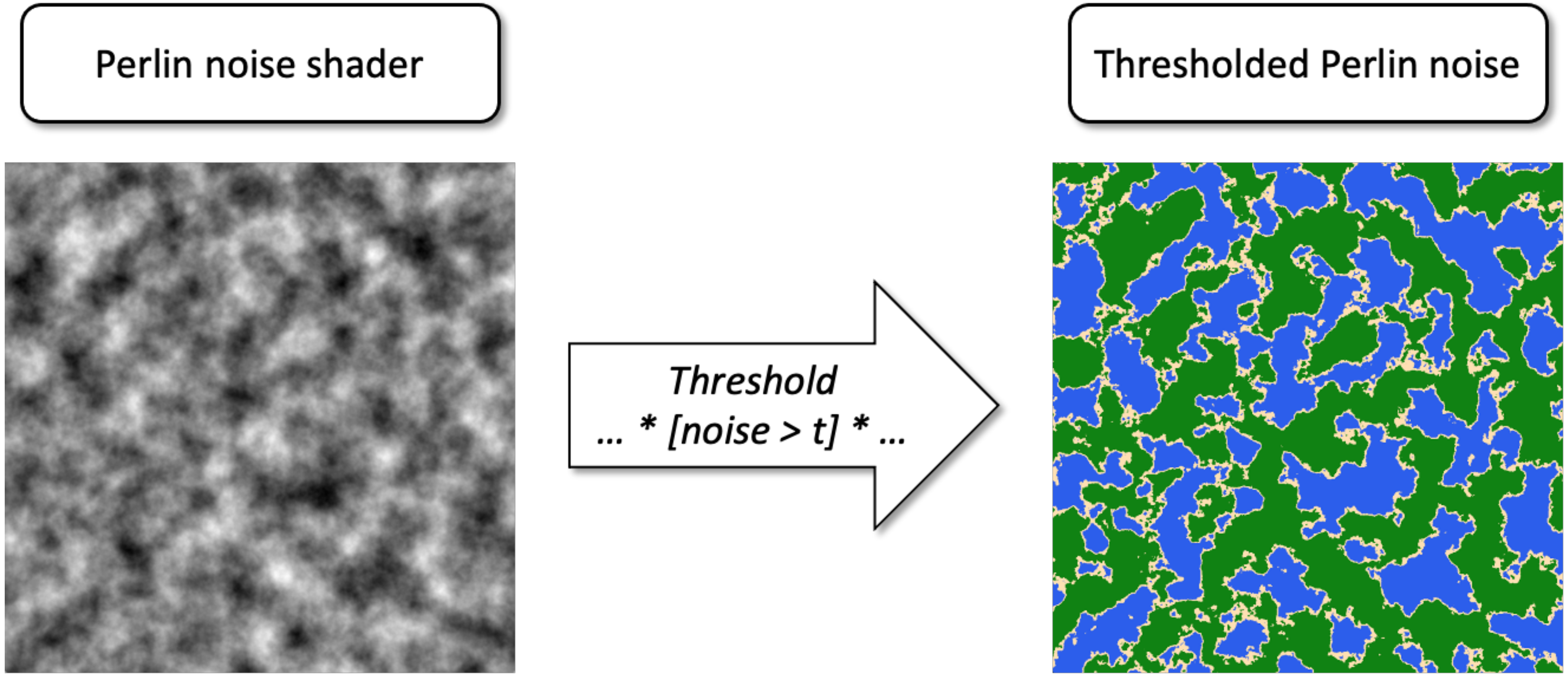
(not my work!)

- text to drawing using CLIP & differentiable rasterization using our renderer



Inverse shader design

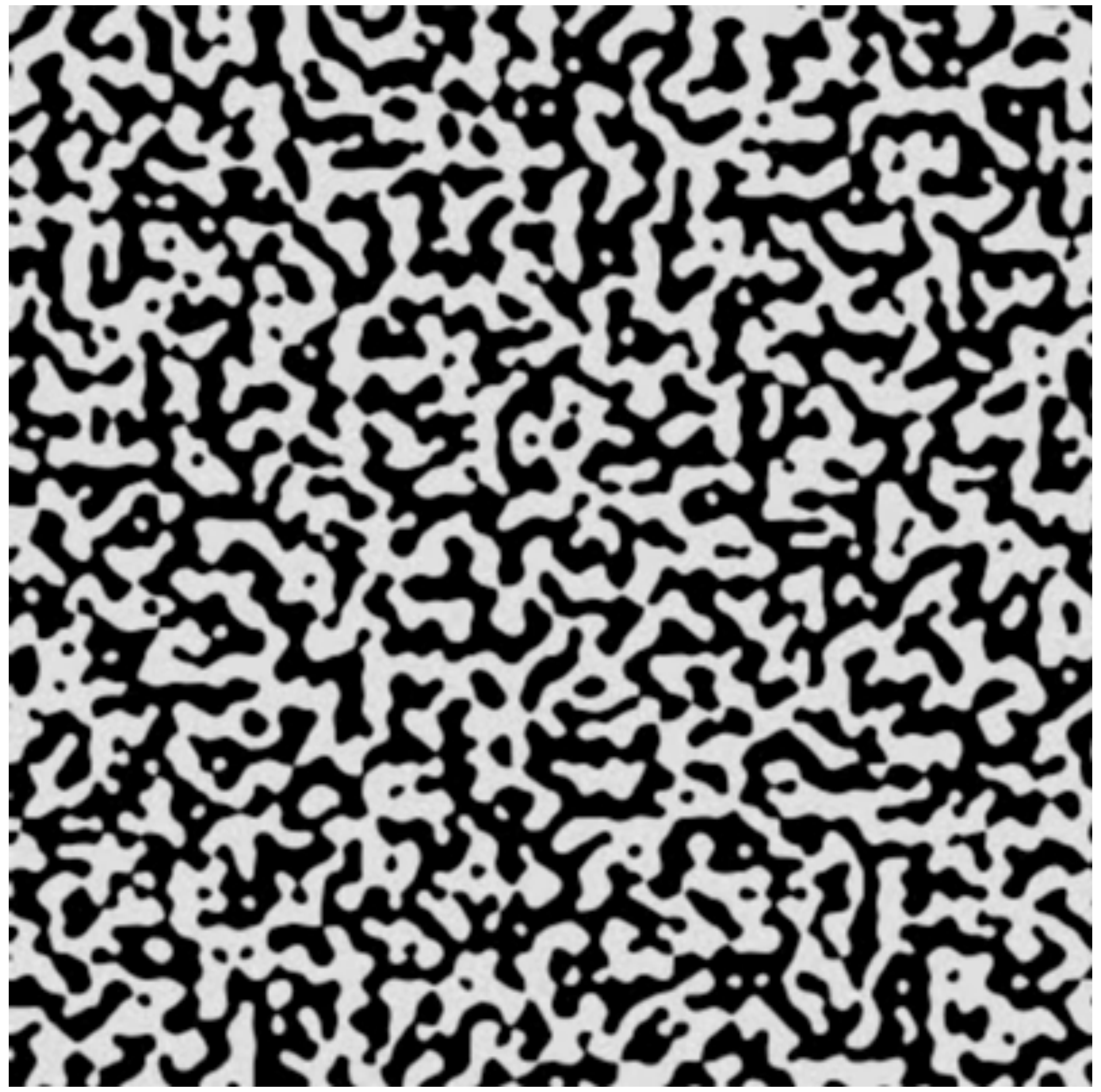
- thresholding Perlin noise leads to discontinuities



*Bangaru, Michel, Mu, Bernstein, **Li**, Ragan-Kelley, 2021*

Inverse shader design

target image



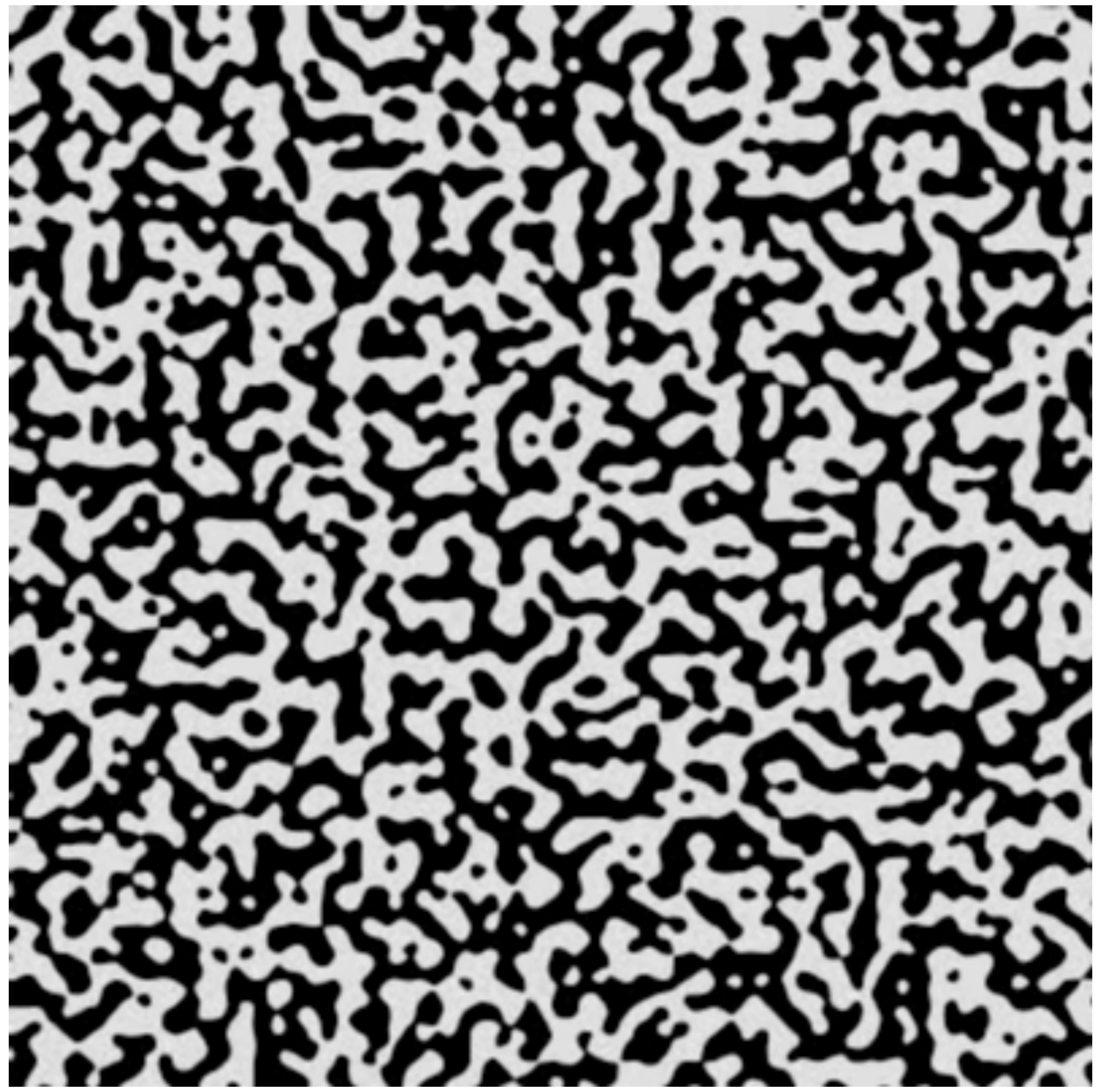
our shader optimization

Bangaru, Michel, Mu, Bernstein, ***Li****, Ragan-Kelley, 2021*

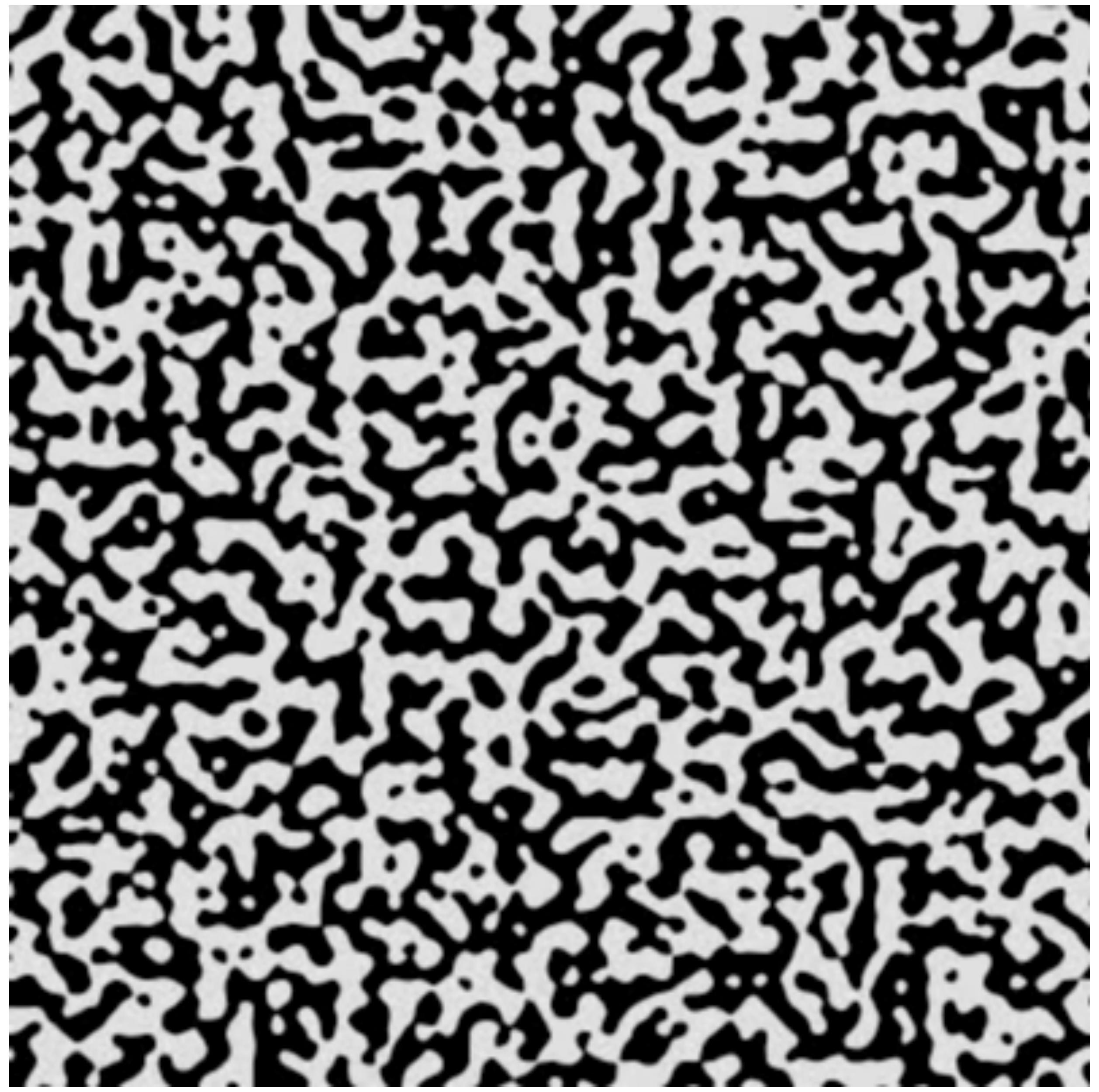
Inverse shader design

- ignoring discontinuities lead to worse/incorrect results

target image



our shader optimization

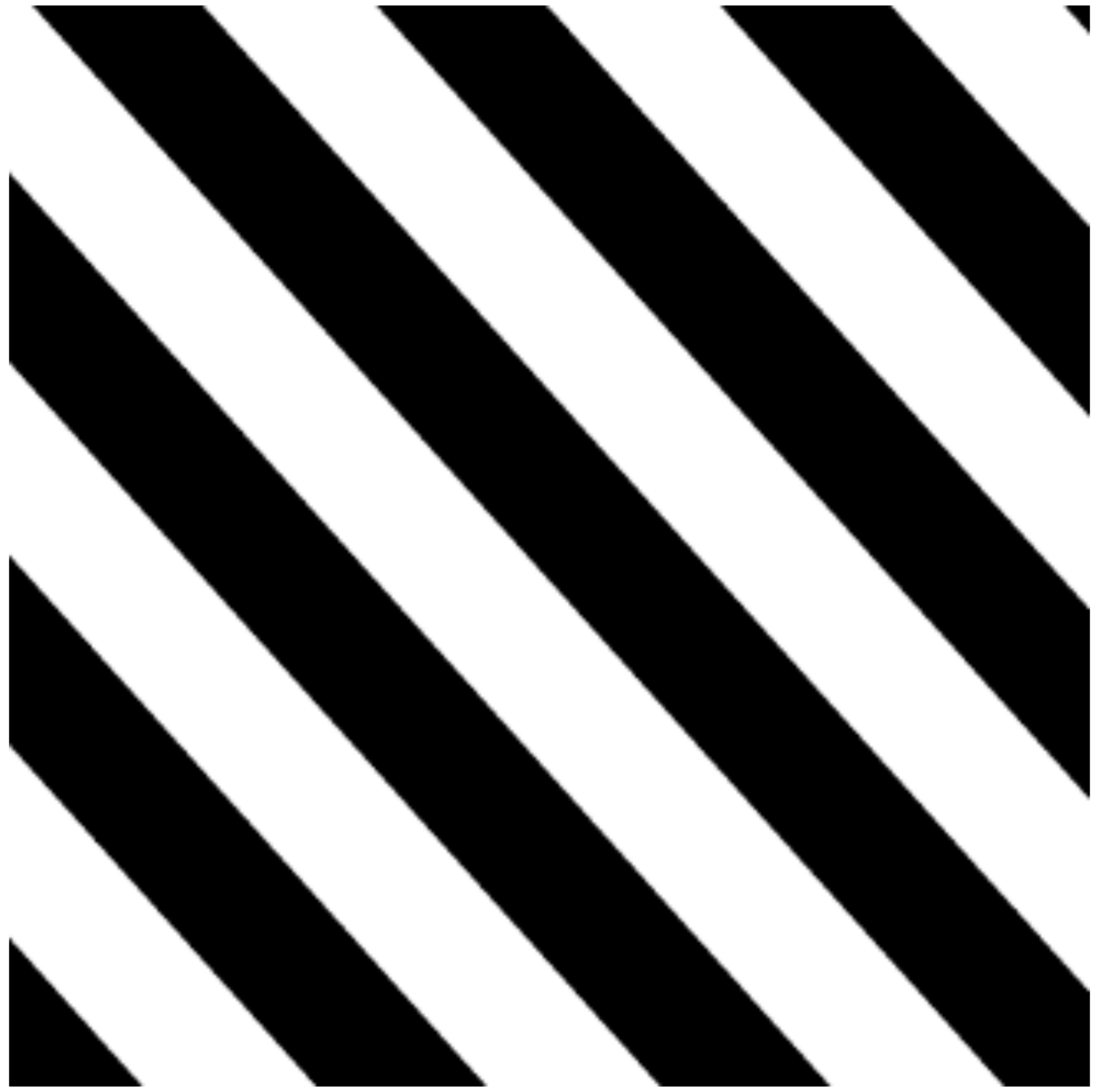


naive autodiff

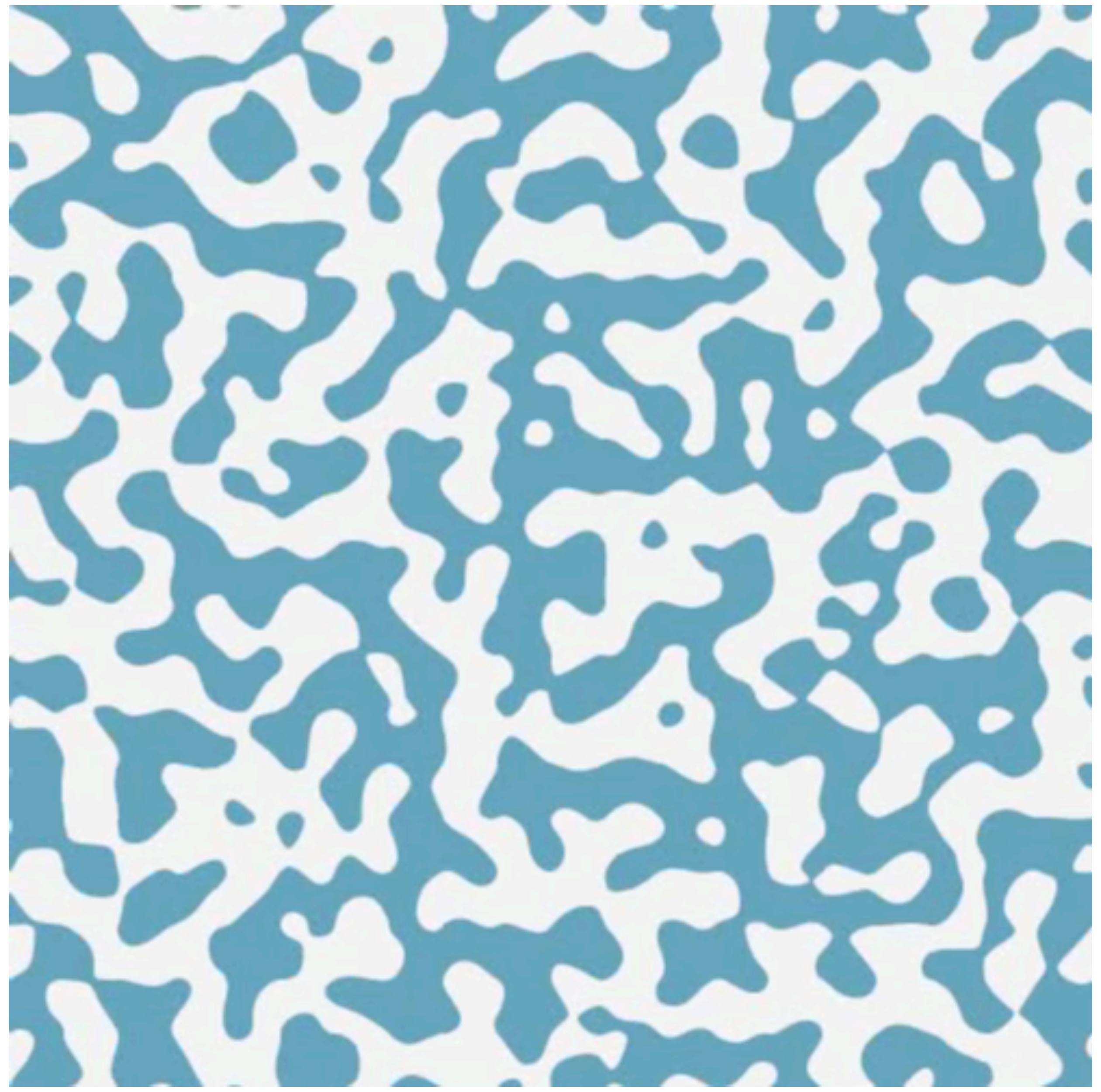
*Bangaru, Michel, Mu, Bernstein, **Li**, Ragan-Kelley, 2021*

Inverse shader design

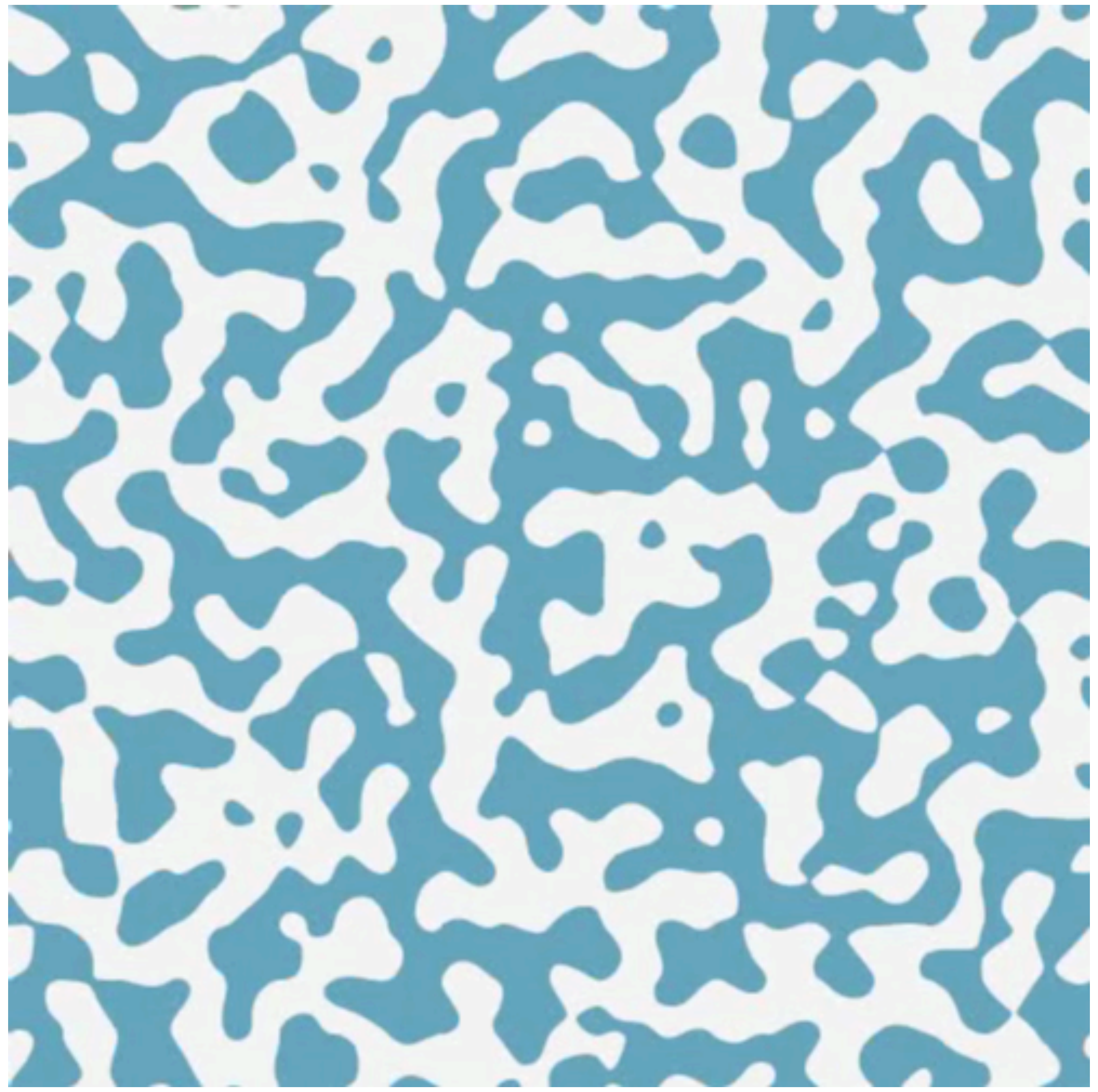
- ignoring discontinuities lead to worse/incorrect results



target image



our shader optimization

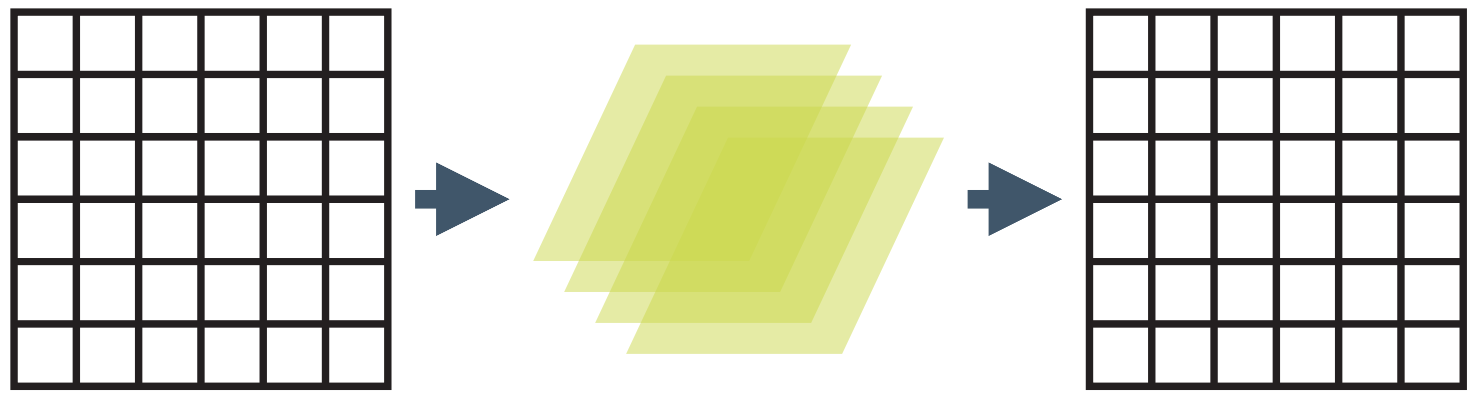


naive autodiff

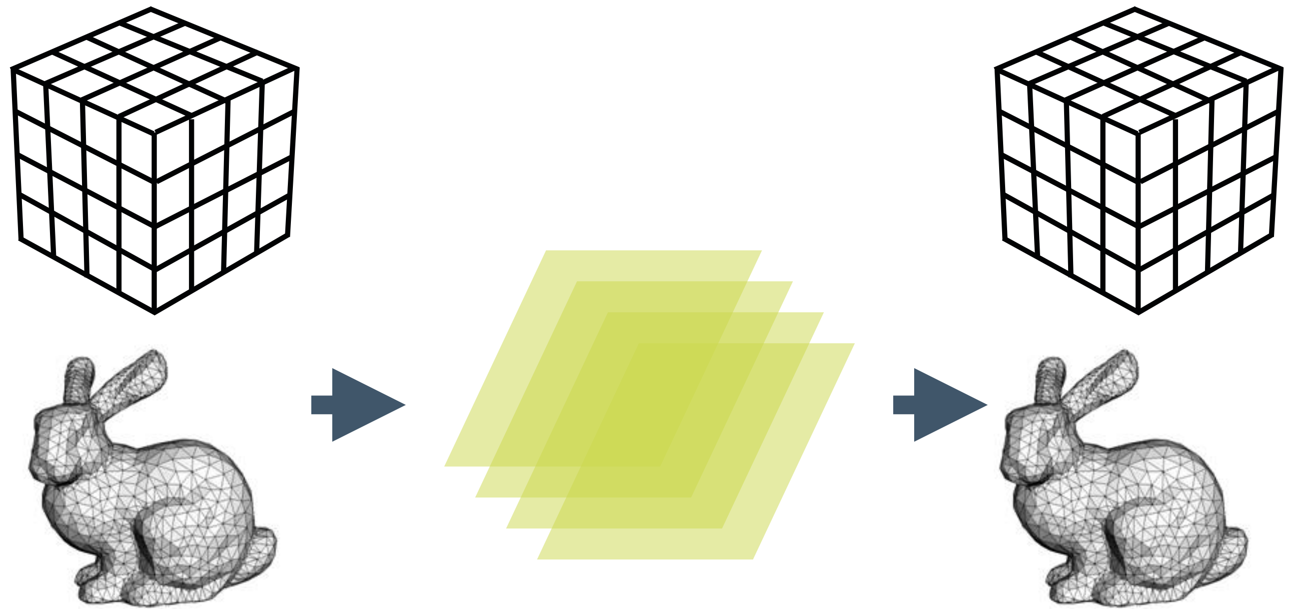
Bangaru, Michel*, Mu, Bernstein, **Li**, Ragan-Kelley, 2021*

Recap

differentiable rendering connects 2D and 3D (and vector / raster)



2D 2D



3D 3D

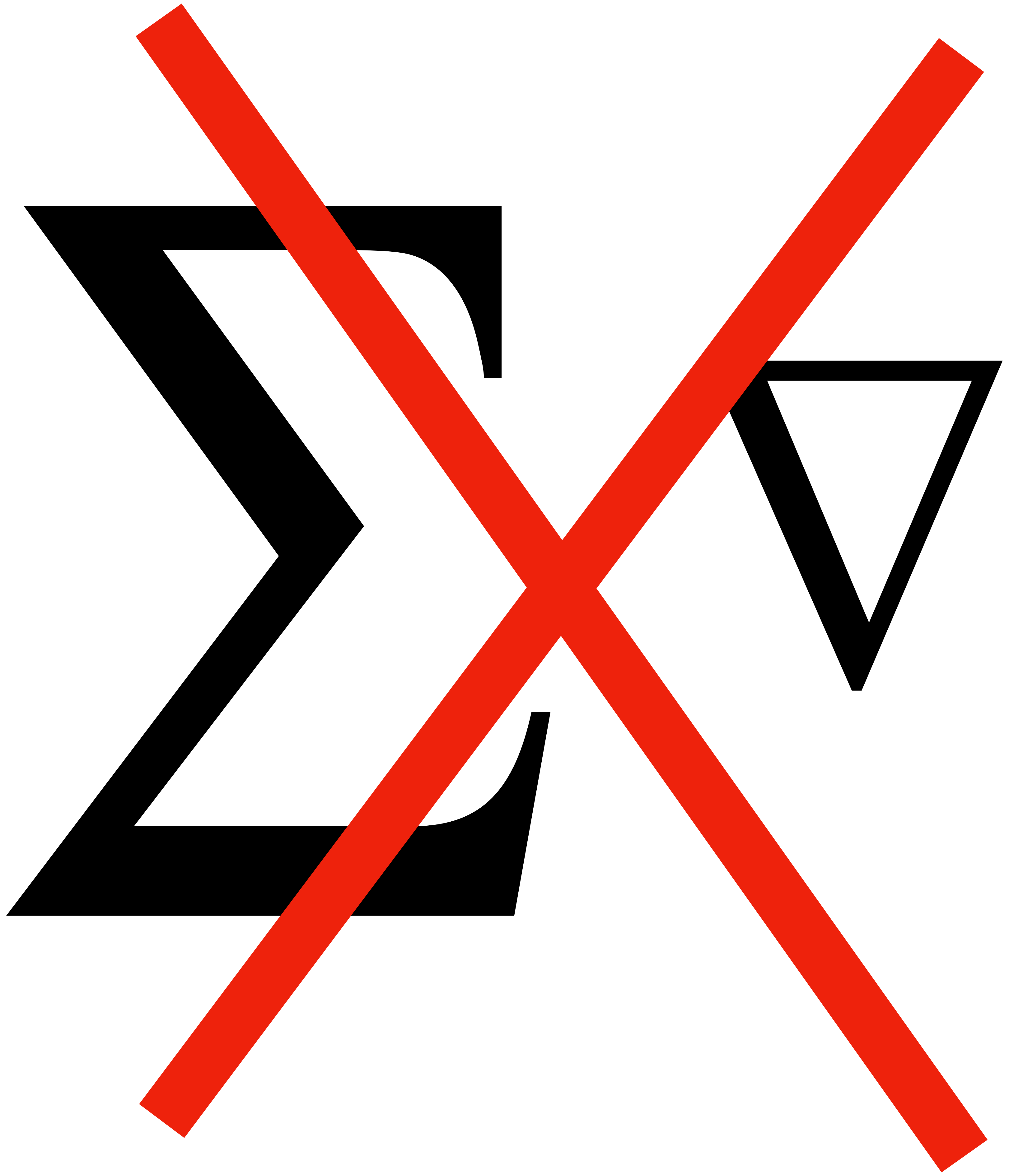
3D

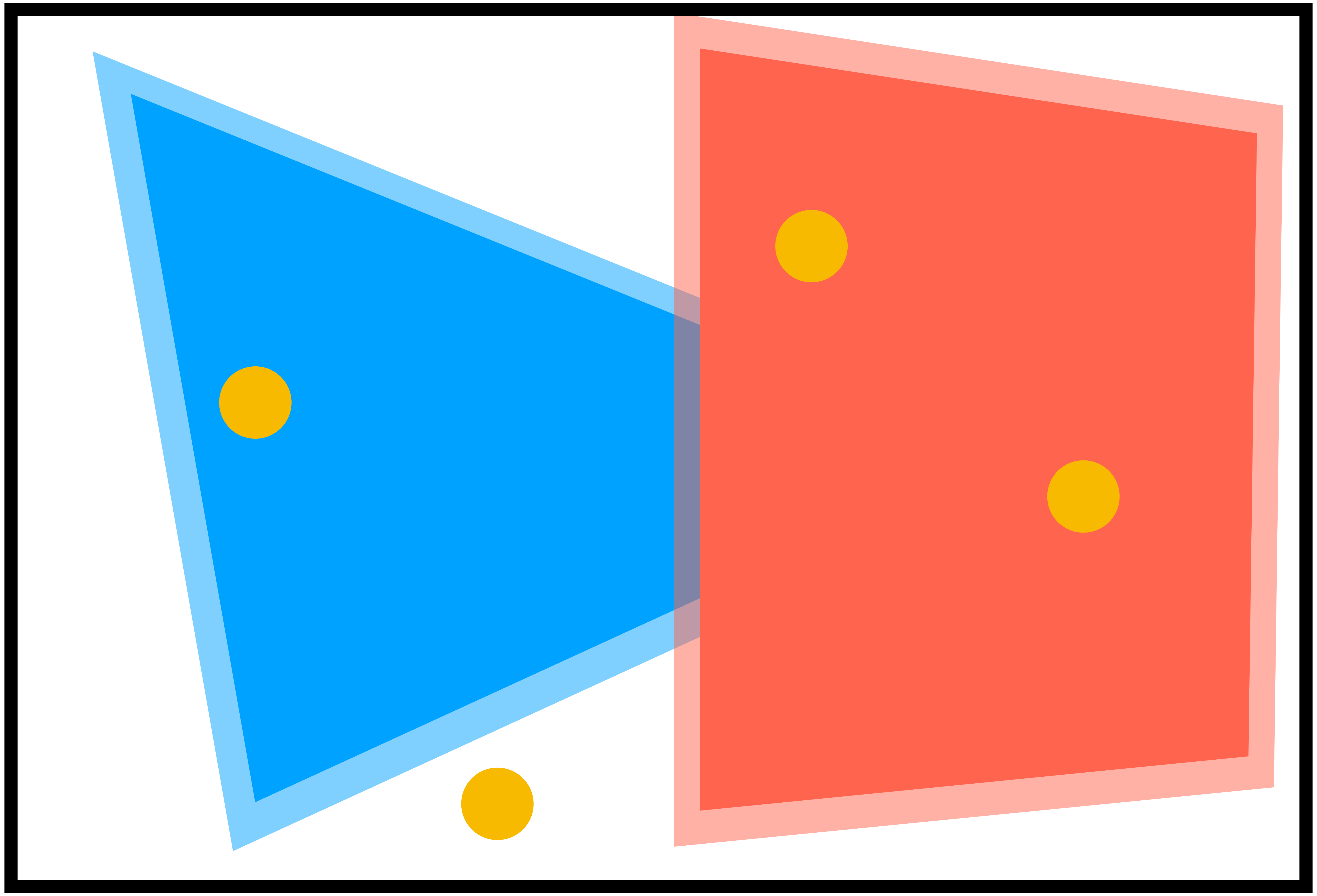
differentiable rendering

2D

Recap

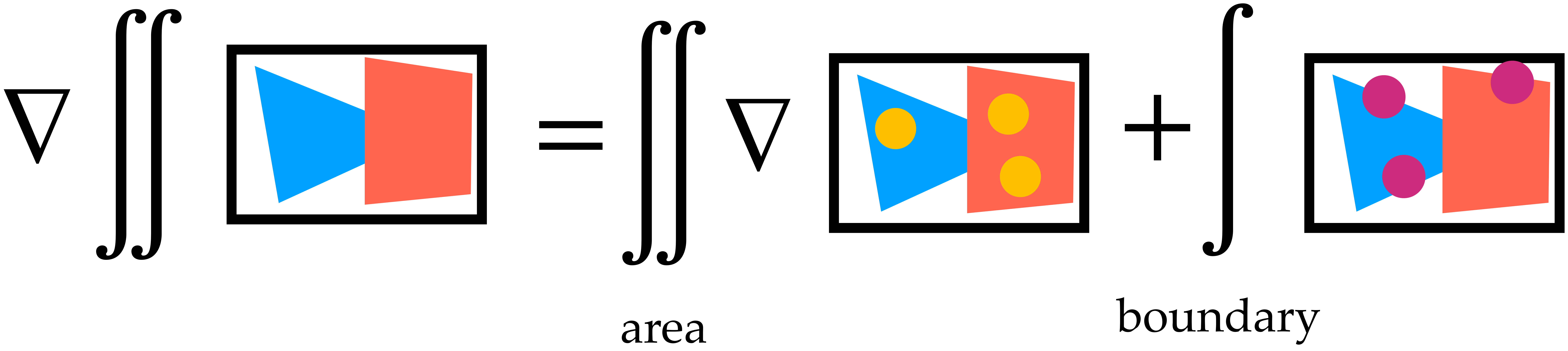
automatically differentiating a renderer computes incorrect results!



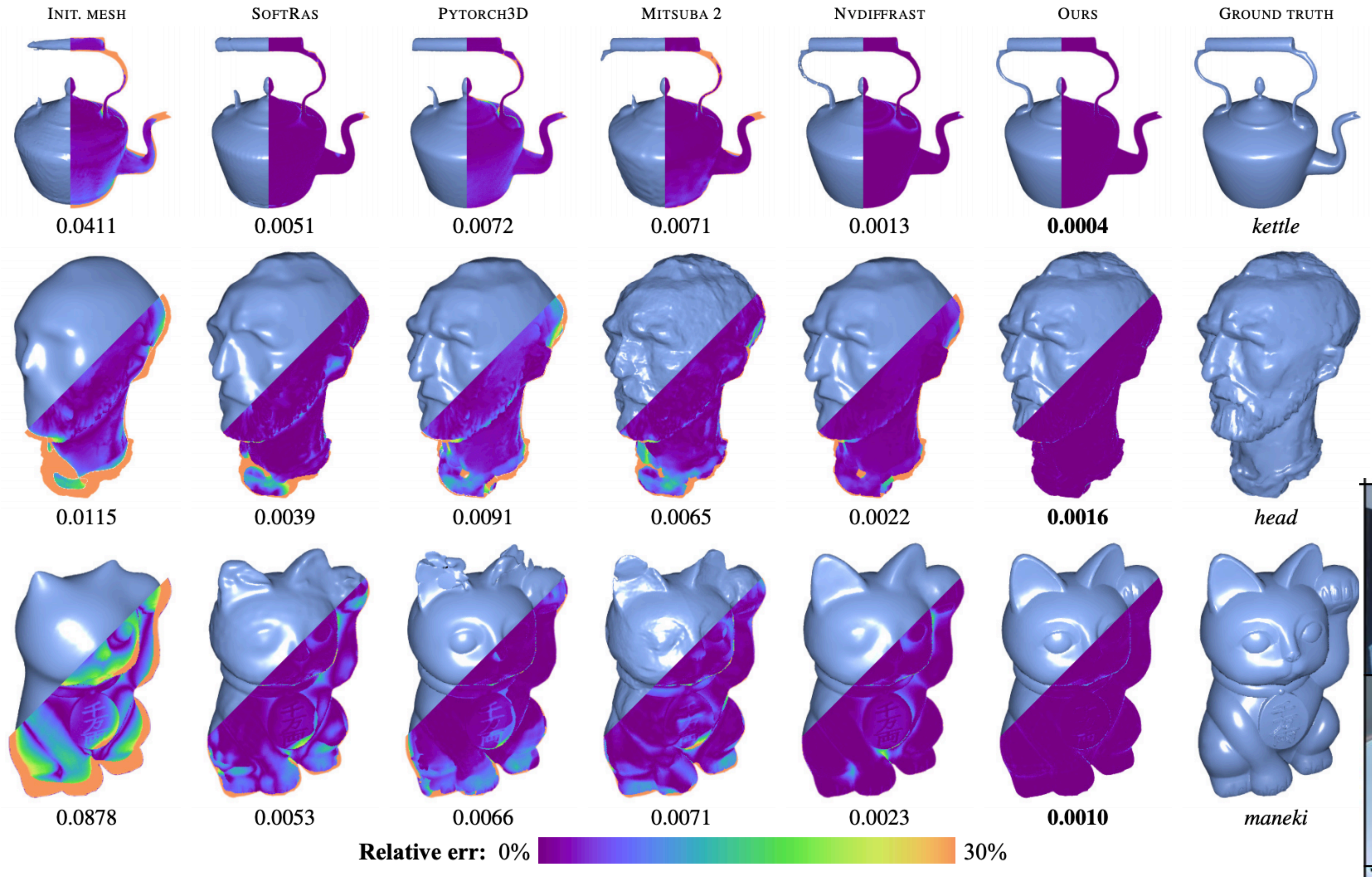


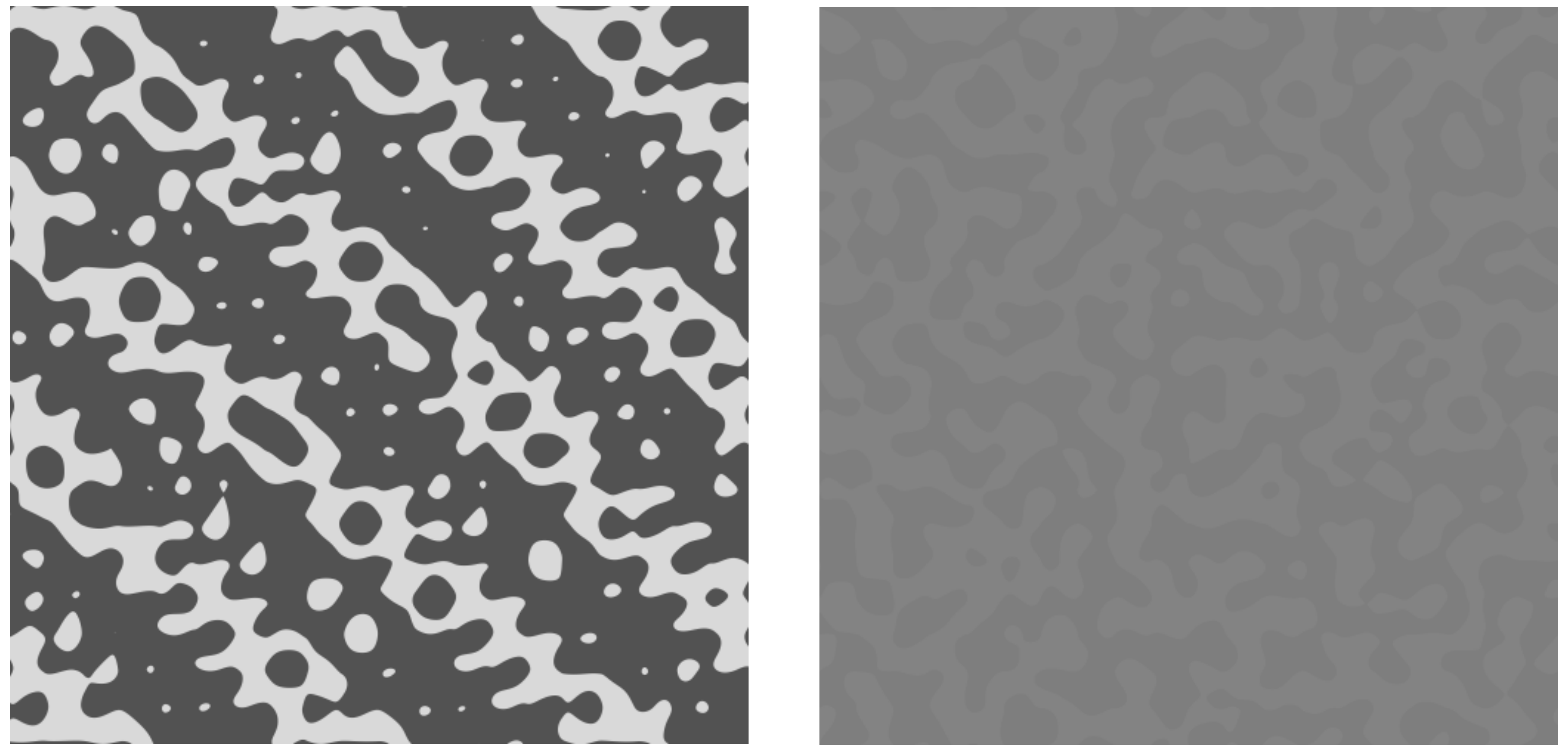
Recap

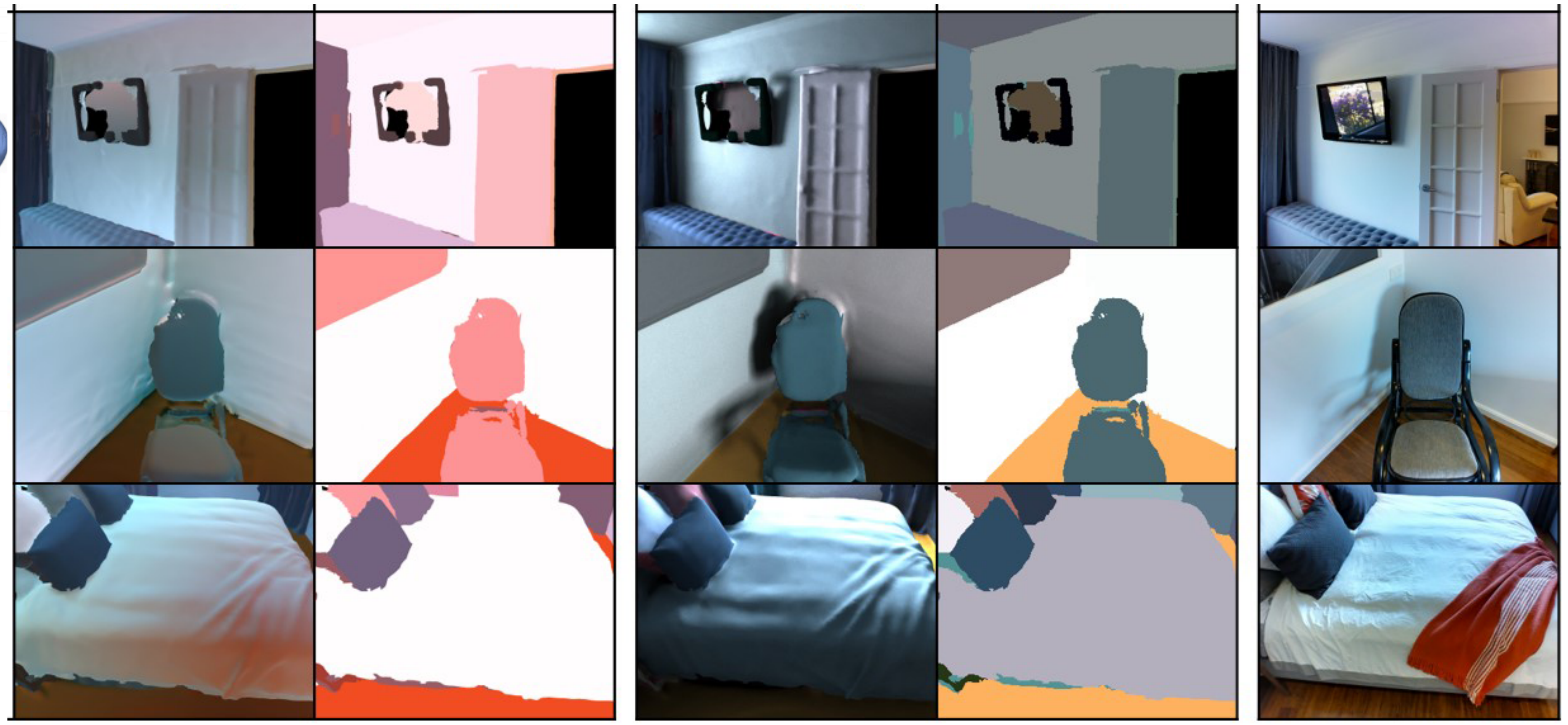
- what autodiff is missing:
  the domain knowledge that rendering equation is an integral
- we derive the correct derivatives from first principles



Correct gradients & lighting matters



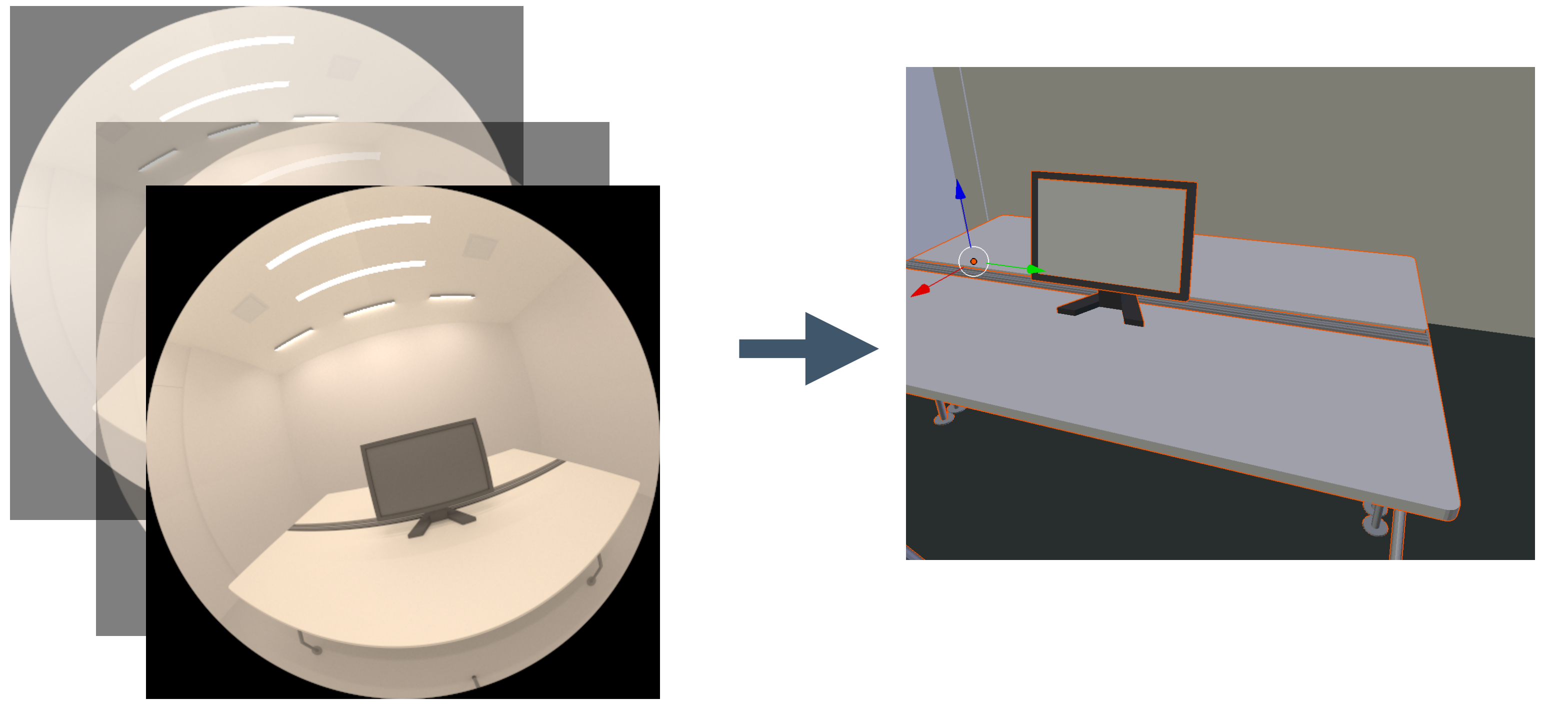




Maier 2017 Ours photo

Beyond Neural Networks

differentiable rendering enables 3D & vector reasoning in optimization and training

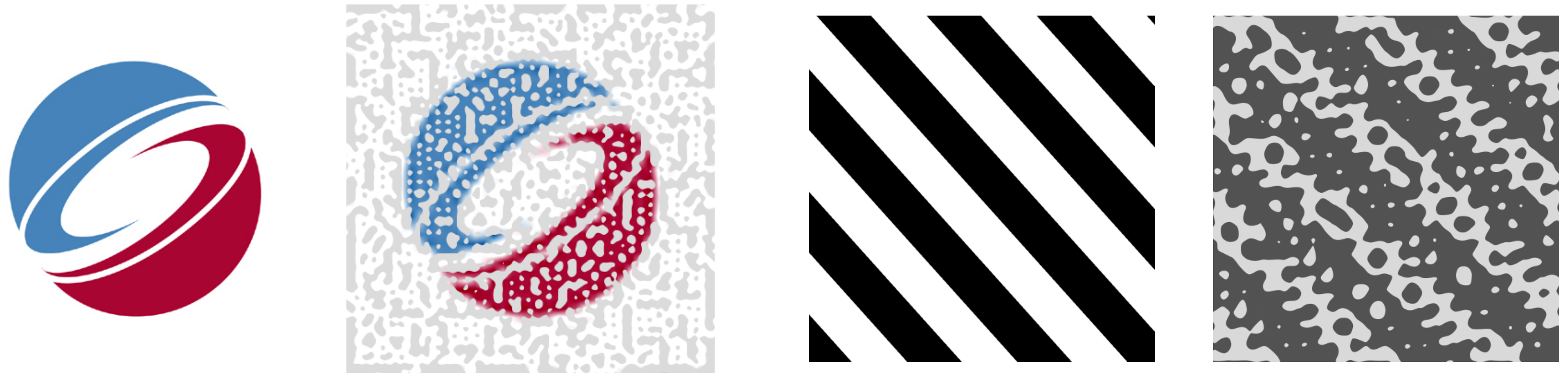


inverse rendering

analyzing vision systems

“A drawing of a cat”.

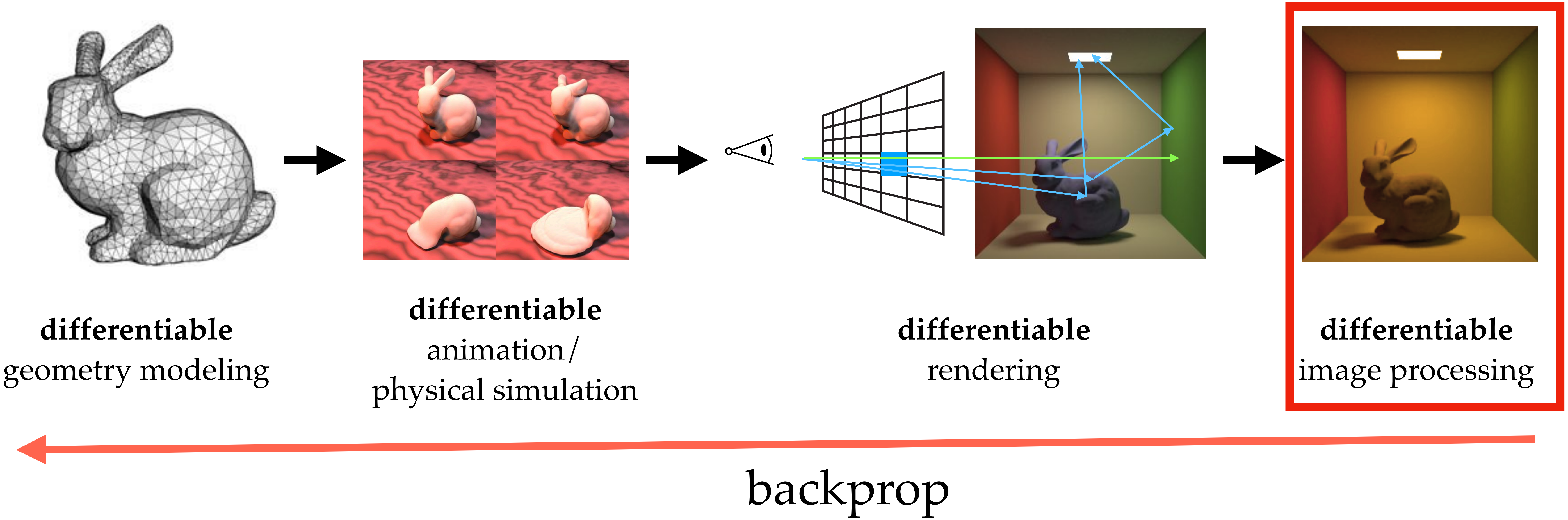
bridging vector and raster graphics



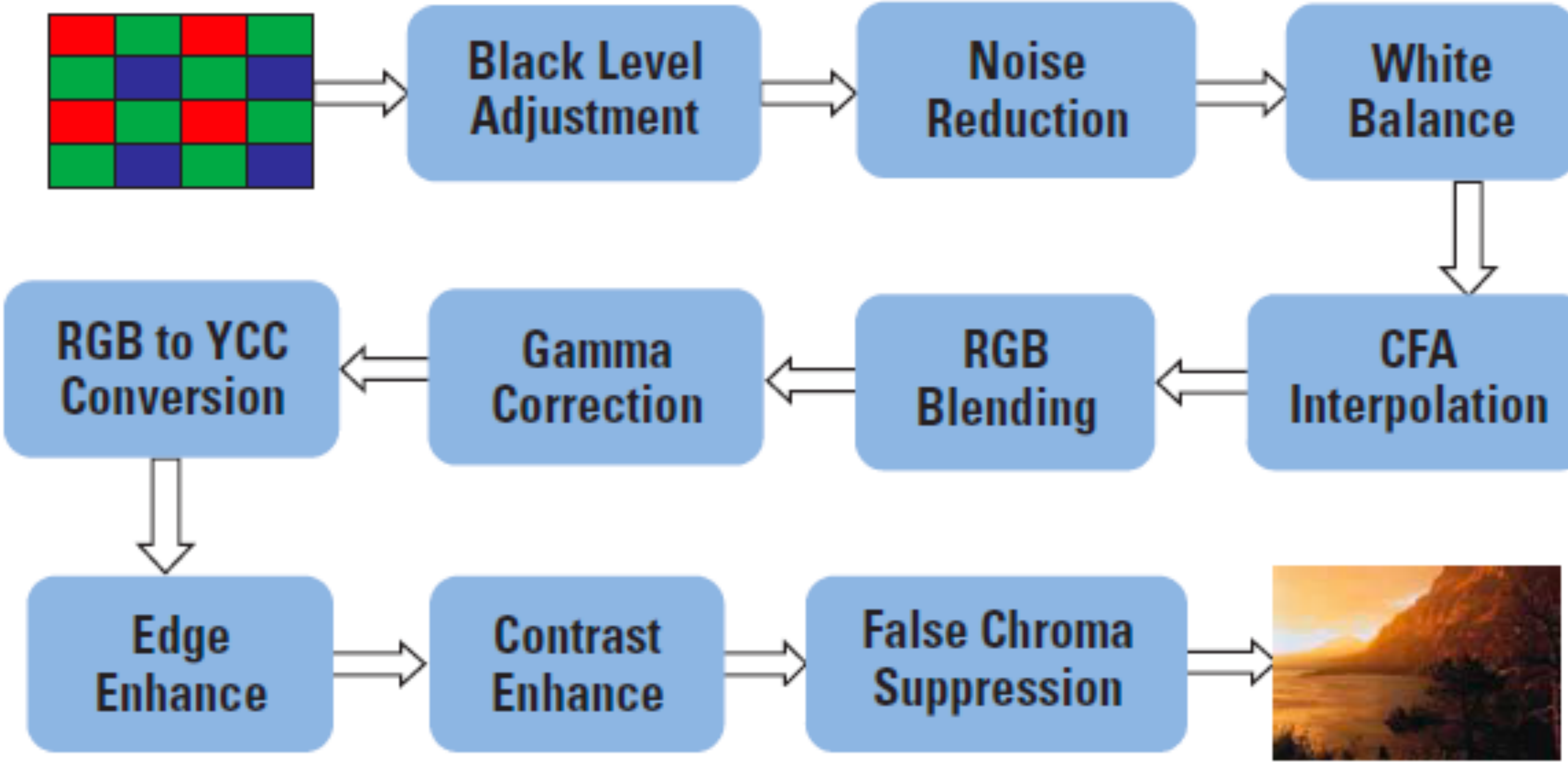
inverse shader design

Differentiable graphics

connects classical graphics algorithms with
modern data-driven methods **through derivatives**



Modern cameras have a complex pipeline



https://iamard.blogspot.com/2014/01/isp-image-processing-pipeline-overview.html

Convolutional neural networks: powerful but expensive

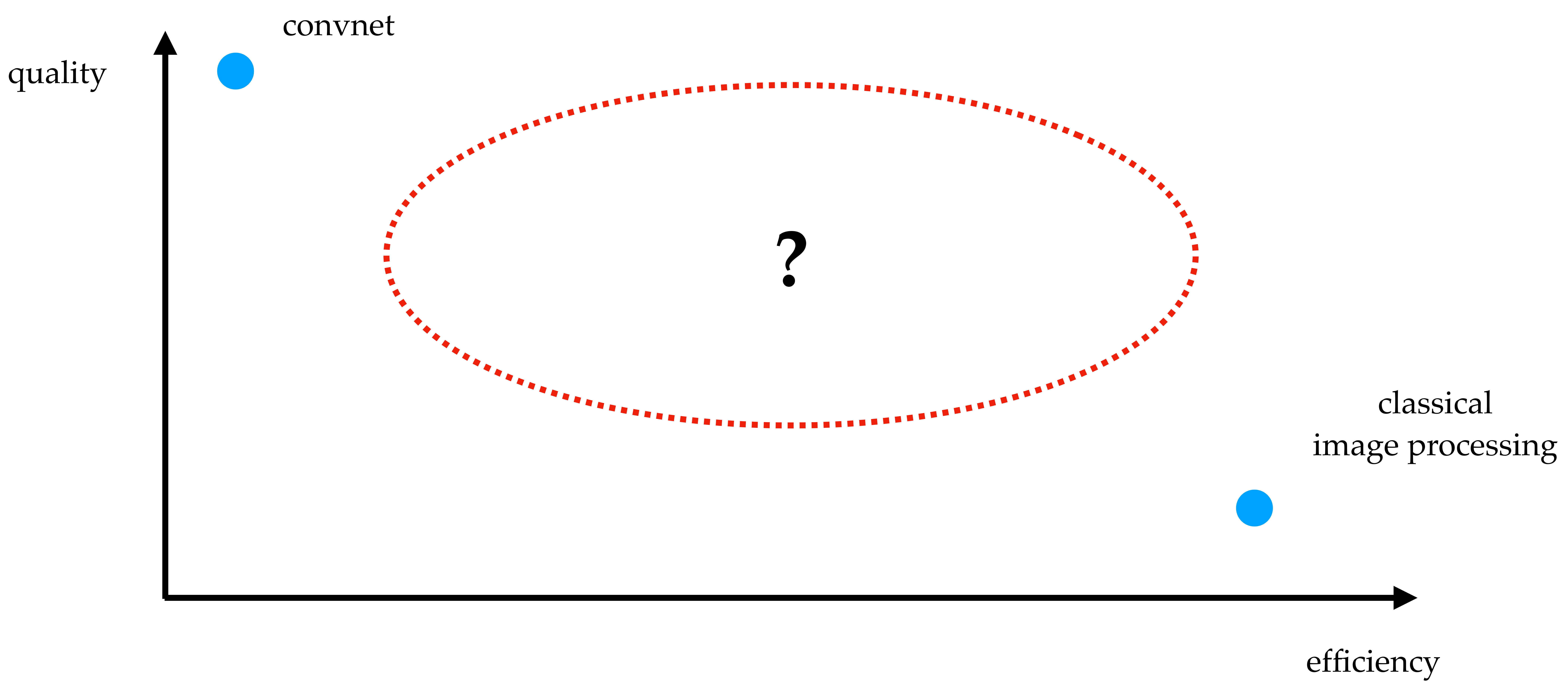
designing our own algorithms enables speed & control

deeplab-res101-v2 on full HD: 2.6 TFLOPs, 40GB features

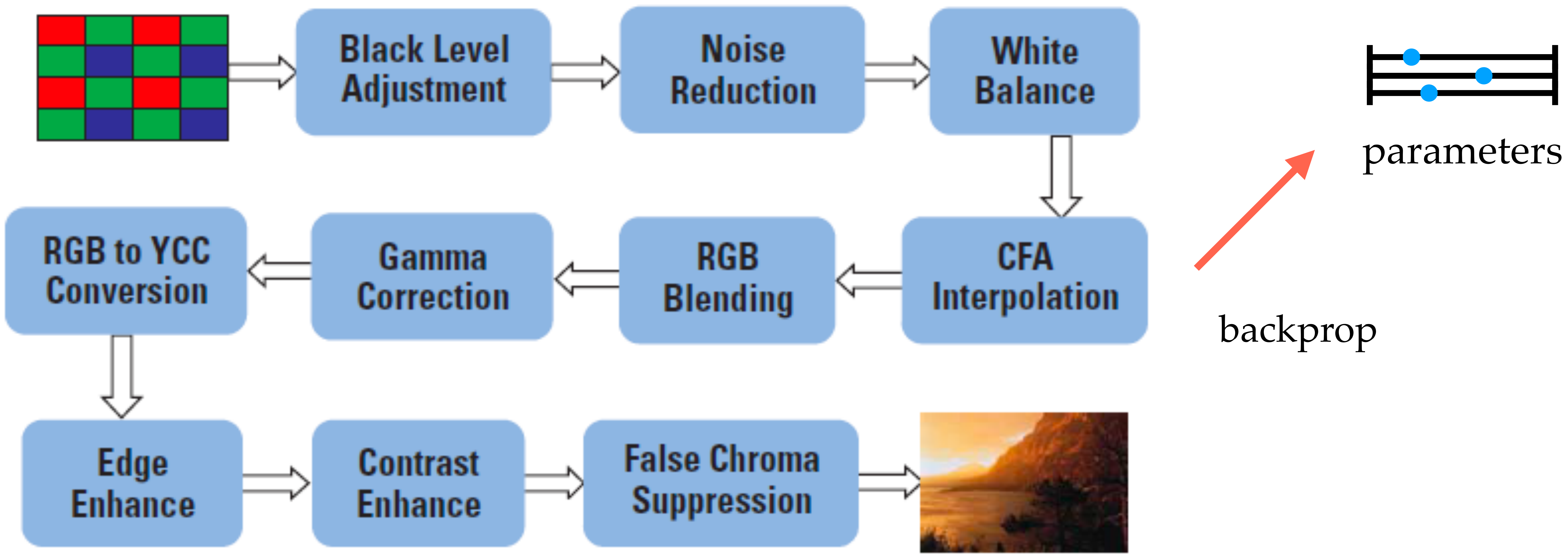
Chen et al.

Google Pixel 4 XL
theoretical peak perf:
954.7 GFLOP/s

The missing Pareto frontier of quality & efficiency trade-offs



Idea: treating traditional image processing building blocks as neural networks



Domain-specific building blocks are more efficient than generic convolutional layers

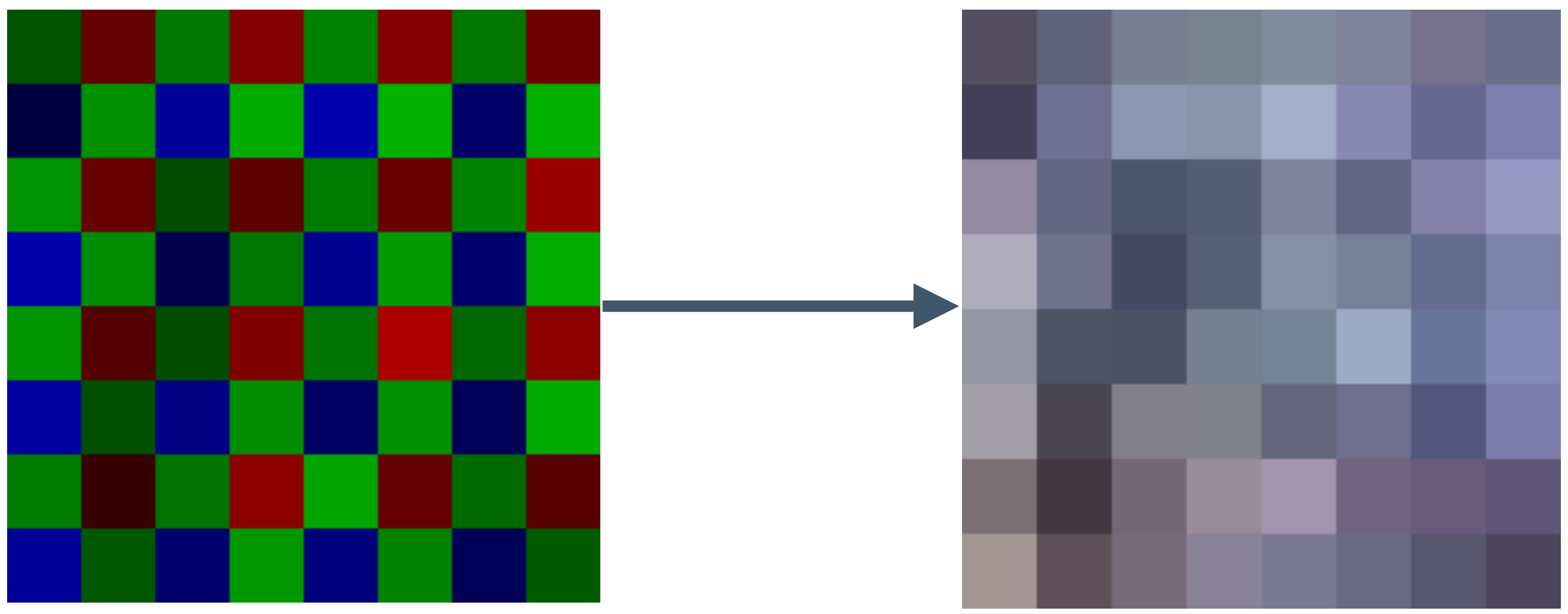
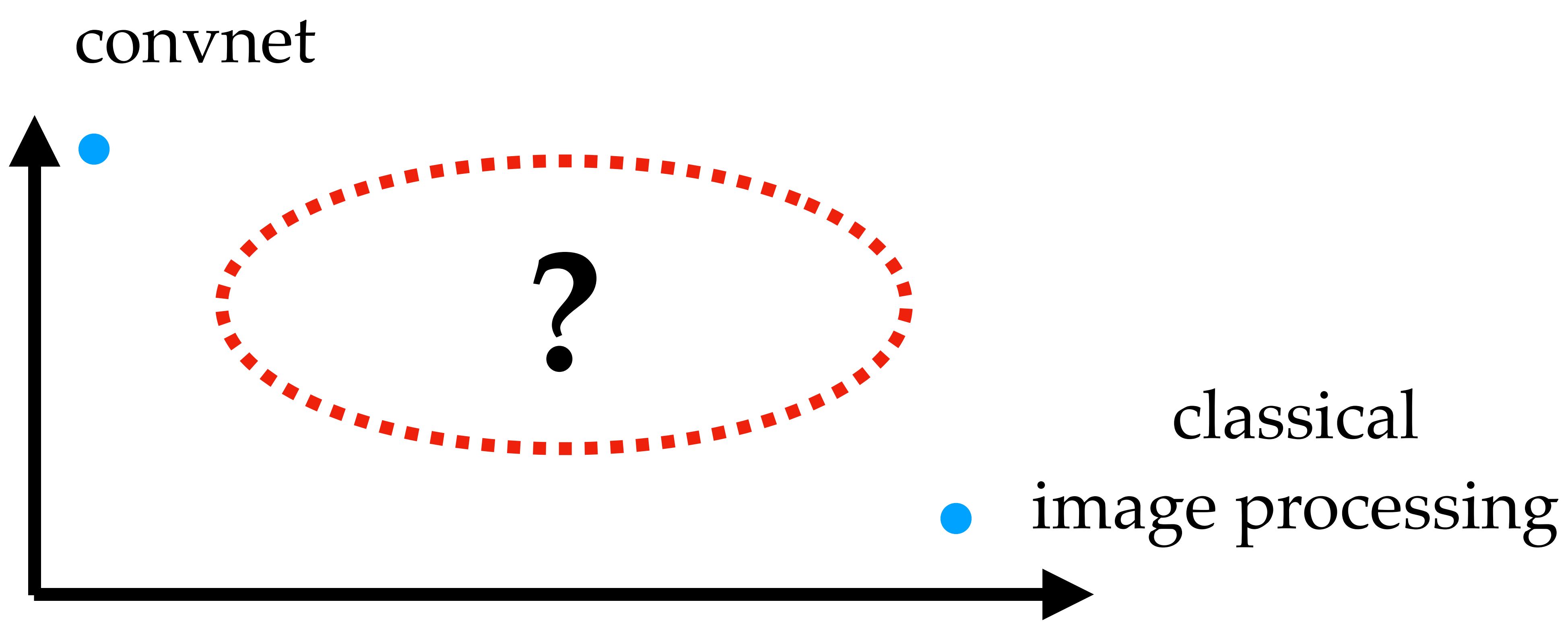
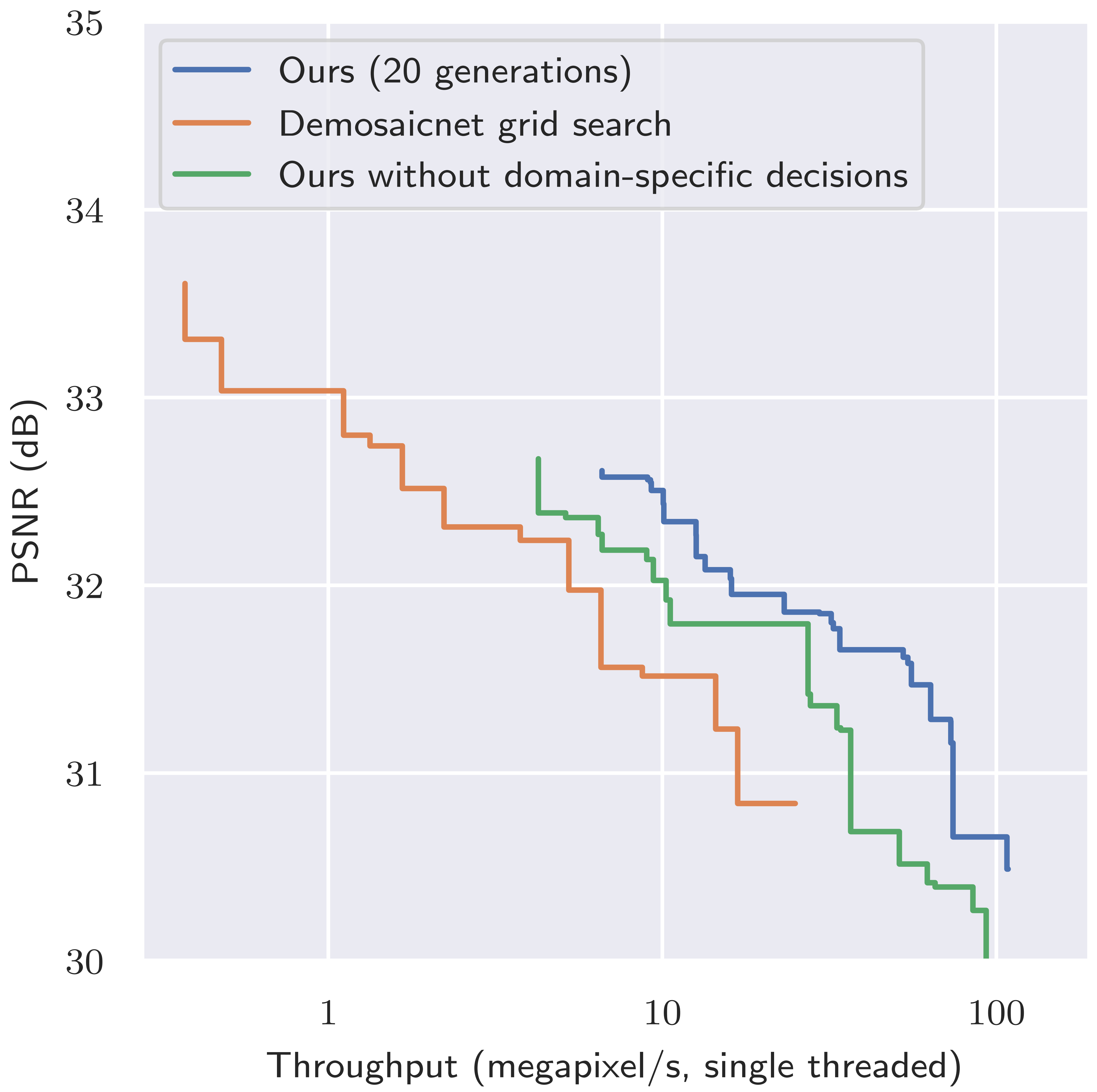


image demosaicing





work in submission

Challenge: deep learning frameworks are limited

new algorithms require new operators

2D convolution
`tf.conv2d`

warping images
`tfa.image.dense_image_warp`

downsample while taking maximum
`tf.max_pool2d`

Challenge: deep learning frameworks are limited

new algorithms require new operators

2D convolution
`tf.conv2d`

warping images
`tfa.image.dense_image_warp`

downsample while taking maximum
`tf.max_pool2d`

4D convolution?

spatially-varying convolution?

Lanzcos interpolation?

different boundary conditions?

progressive mesh?

ray tracing?

skinning?

material point method?

Deep learning frameworks are limited

new

2D
tf.

warping images
`tfa.image.dense_image_warp`

downsample while taking maximum
`tf.max_pool2d`

lution?

on?

litions?

progressive mesh?

ray tracing?

skinning?

material point method?

Deep learning frameworks are limited

Deep learning frameworks are limited

Machine Learning Systems are Stuck in a Rut

Paul Barham
Google Brain

Michael Isard
Google Brain

Abstract

In this paper we argue that systems for numerical computing are stuck in a local basin of performance and programmability. Systems researchers are doing an excellent job improving the performance of 5-year-old benchmarks, but gradually making it harder to explore innovative machine learning research ideas.

We build compilers for exploring differentiable programming models

CUDA 308 lines

430 ms (1M pix)
2270 ms (4M pix)

```python
xx = Variable(th.arange(0, w).cuda().view(1, -1).repeat(h, 1))
yy = Variable(th.arange(0, h).cuda().view(-1, 1).repeat(1, w))
gx = ((xx+0.5)/w) * gw
gy = ((yy+0.5)/h) * gh
gz = th.clamp(guide, 0.0, 1.0)*gd
fx = th.clamp(th.floor(gx - 0.5), min=0)
fy = th.clamp(th.floor(gy - 0.5), min=0)
fz = th.clamp(th.floor(gz - 0.5), min=0)
wx = gx - 0.5 - fx
wy = gy - 0.5 - fy
wx = wx.unsqueeze(0).unsqueeze(0)
wy = wy.unsqueeze(0).unsqueeze(0)
wz = th.abs(gz-0.5 - fz)
wz = wz.unsqueeze(1)
fx = fx.long().unsqueeze(0).unsqueeze(0)
fy = fy.long().unsqueeze(0).unsqueeze(0)
fz = fz.long()
cx = th.clamp(fx+1, max=gw-1);
cy = th.clamp(fy+1, max=gh-1);
cz = th.clamp(fz+1, max=gd-1)
fz = fz.view(bs, 1, h, w)
cz = cz.view(bs, 1, h, w)
batch_idx = th.arange(bs).view(bs, 1, 1, 1).long().cuda()
out = []
co = c // (ci+1)
for c_ in range(co):
  c_idx = th.arange((ci+1)*c_, (ci+1)*(c_+1)).view(\
            1, ci+1, 1, 1).long().cuda()
  a = grid[batch_idx, c_idx, fz, fy, fx]*(1-wx)*(1-wy)*(1-wz) + \
           grid[batch_idx, c_idx, cz, fy, fx]*(1-wx)*(1-wy)*(  wz) + \
           grid[batch_idx, c_idx, fz, cy, fx]*(1-wx)*(  wy)*(1-wz) + \
           grid[batch_idx, c_idx, cz, cy, fx]*(1-wx)*(  wy)*(  wz) + \
           grid[batch_idx, c_idx, fz, fy, cx]*(  wx)*(1-wy)*(1-wz) + \
           grid[batch_idx, c_idx, cz, fy, cx]*(  wx)*(1-wy)*(  wz) + \
           grid[batch_idx, c_idx, fz, cy, cx]*(  wx)*(  wy)*(1-wz) + \
           grid[batch_idx, c_idx, cz, cy, cx]*(  wx)*(  wy)*(  wz)
  o = th.sum(a[:, :-1, ...]*input, 1) + a[:, -1, ...]
  out.append(o.unsqueeze(1))
out = th.cat(out, 1)

out.backward(adjoints)
d_input = input.grad
d_grid = grid.grad
d_guide = guide.grad
```

PyTorch 42 lines

1440 ms (1M pix)
out of memory (4M pix)

```cpp
// Slice an affine matrix from the grid and
// transform the color
Expr gx = cast<float>(x)/sigma_s;
Expr gy = cast<float>(y)/sigma_s;
Expr gz =
  clamp(guide(x,y,n),0.f,1.f)*grid.channels();
Expr fx = cast<int>(gx);
Expr fy = cast<int>(gy);
Expr fz = cast<int>(gz);
Expr wx = gx-fx, wy = gy-fy, wz = gz-fz;
Expr tent =
  abs(rt.x-wx)*abs(rt.y-wy)*abs(rt.z-wz);
RDom rt(0,2,0,2,0,2);
Func affine;
affine(x,y,c,n) +=
  grid(fx+rt.x,fy+rt.y,fz+rt.z,c,n)*tent;
Func output;
Expr nci = input.channels();
RDom r(0, nci);
output(x,y,co,n) = affine(x,y,co*(nci+1)+nci,n);
output(x,y,co,n) +=
  affine(x,y,co*(nci+1)+r,n) * in(x,y,r,n);

// Propagate the gradients to inputs
auto d = propagate_adjoints(output, adjoints);
Func d_in = d(in);
Func d_guide = d(guide);
Func d_grid = d(grid);
```

Halide (ours) 24 lines

64 ms (1M pix)
165 ms (4M pix)

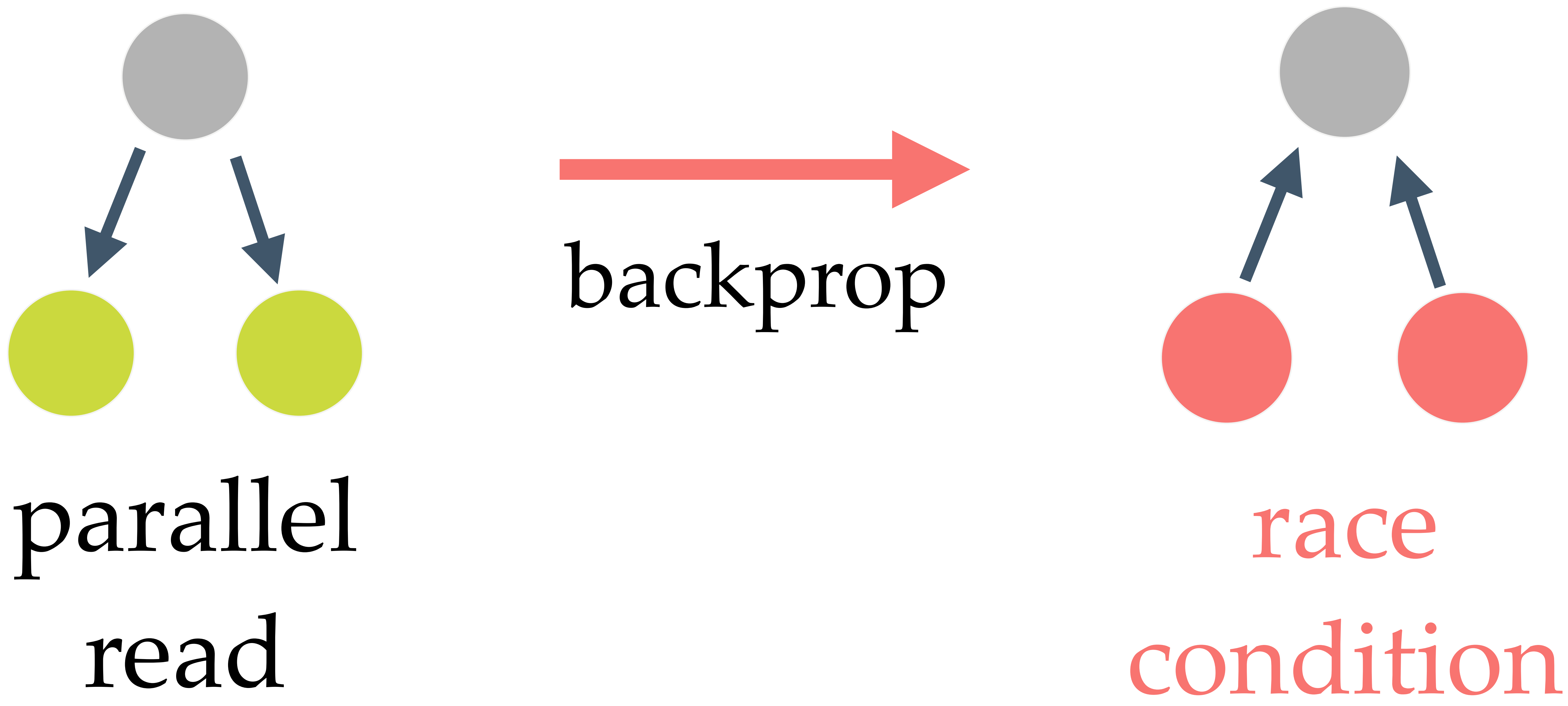
bilateral learning
[Gharbi 2017]

red region: gradient code

Li, Gharbi, Adams, Durand, Ragan-Kelley, 2018

Generate correct and efficient parallel derivative code is hard

- backpropagation *inverts* dependencies



Generate correct and efficient parallel derivative code is hard

- backpropagation *inverts* dependencies
- optimized code is harder to analyze

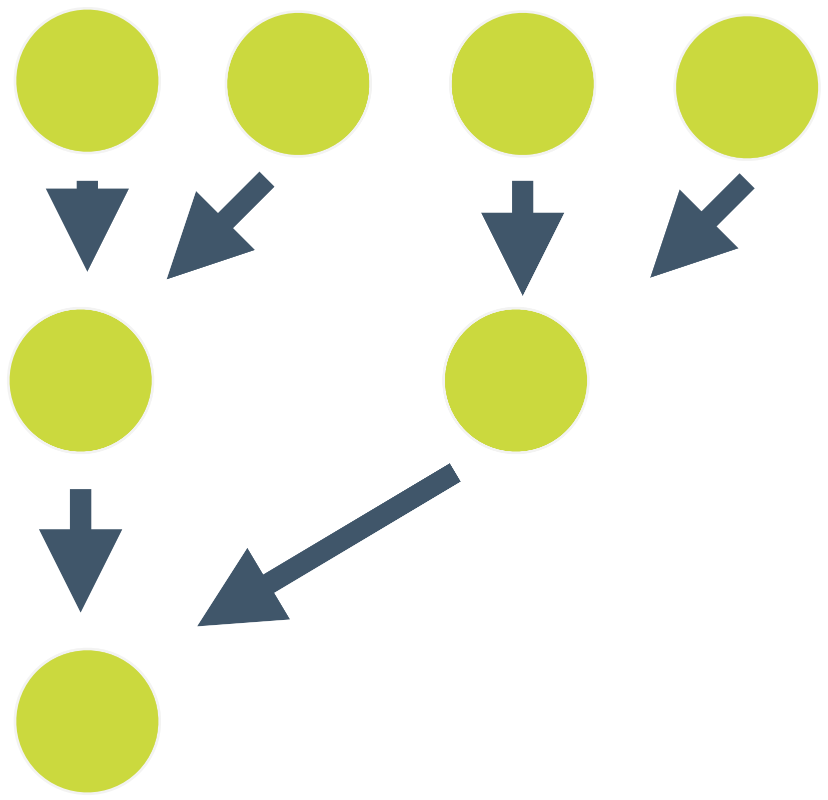
$$y \leftarrow \sum x_i \qquad\qquad dx_i \leftarrow dy$$

Generate correct and efficient parallel derivative code is hard

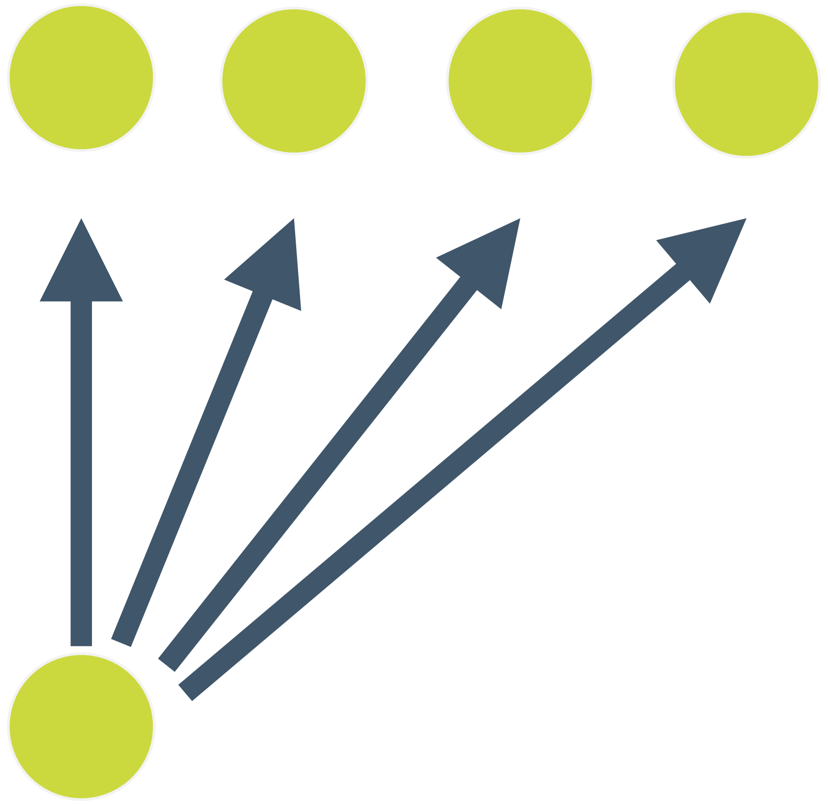
- backpropagation *inverts* dependencies

- optimized code is harder to analyze

$$y \leftarrow \sum x_i$$



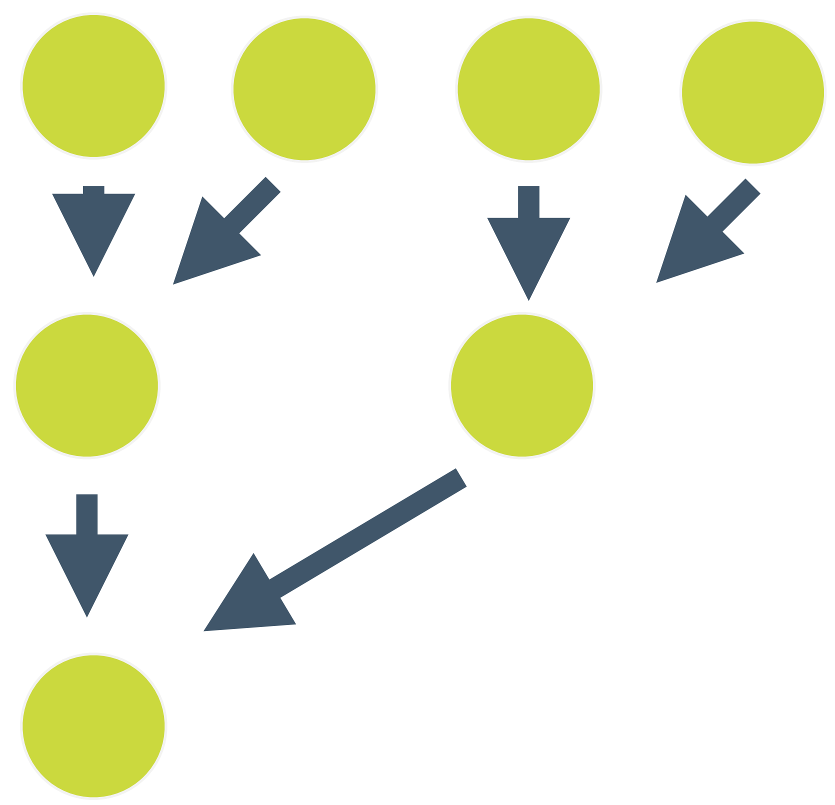
$$dx_i \leftarrow dy$$



Key idea: differentiate algorithm, not implementation

$$y = \sum x_i$$

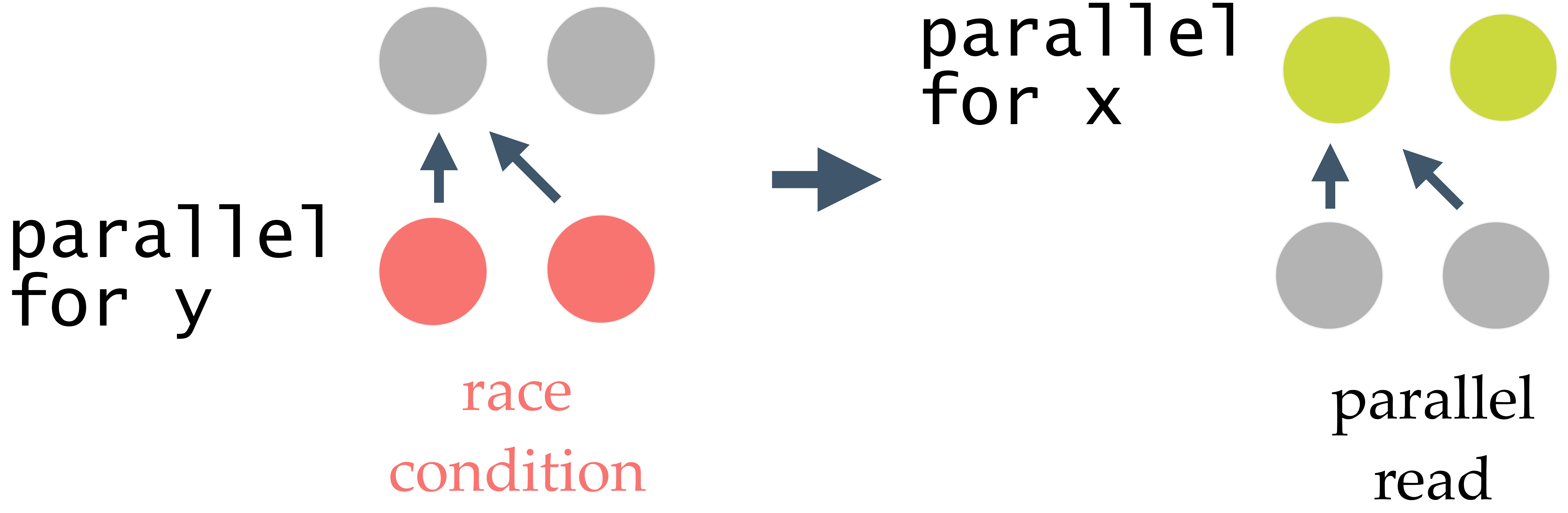
high-level, domain-specific algorithm



low-level implementation

Key idea: differentiate algorithm, not implementation

- avoid race condition by transforming loops

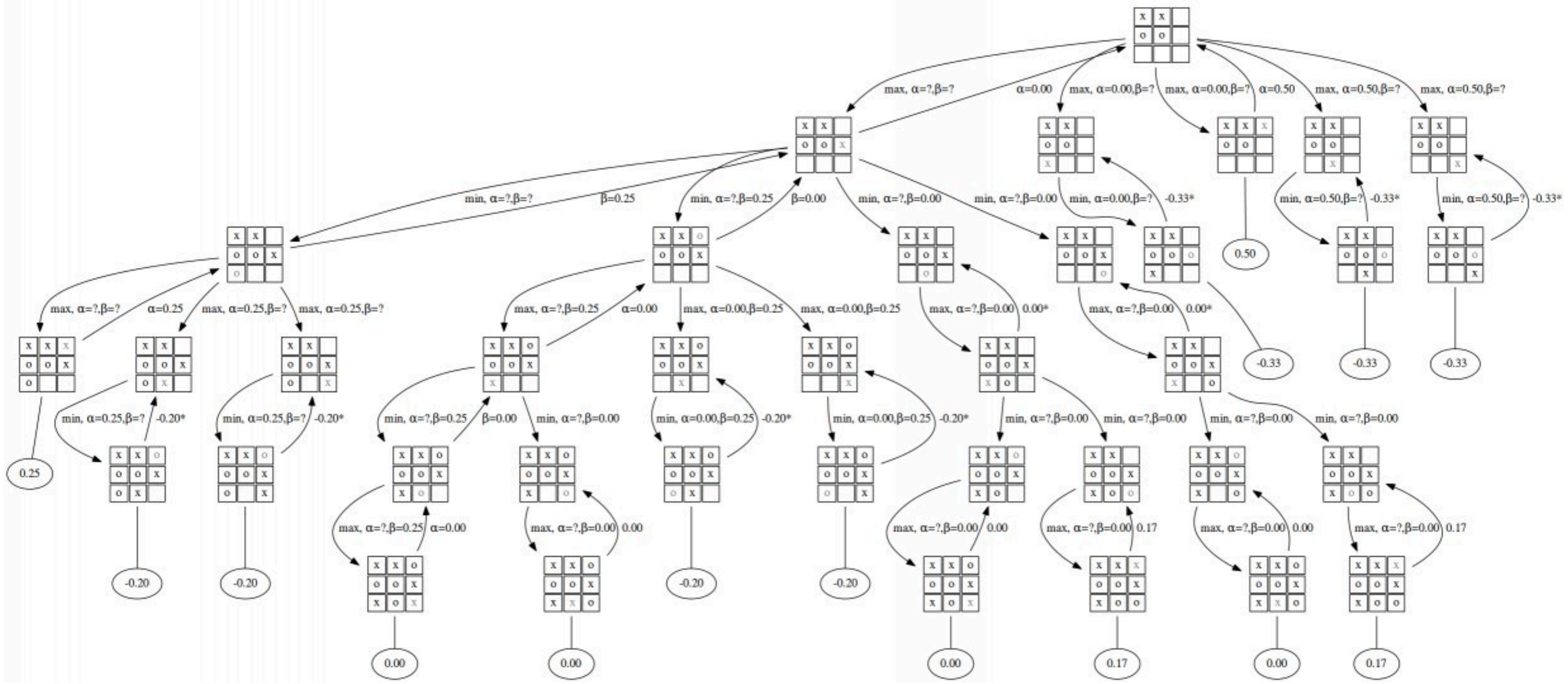


Key idea: differentiate algorithm, not implementation

- avoid race condition by transforming loops
- implementation can be defined by users or searched automatically (c.f. autoscheduling/program synthesis)

$$y = \sum x_i$$

algorithm



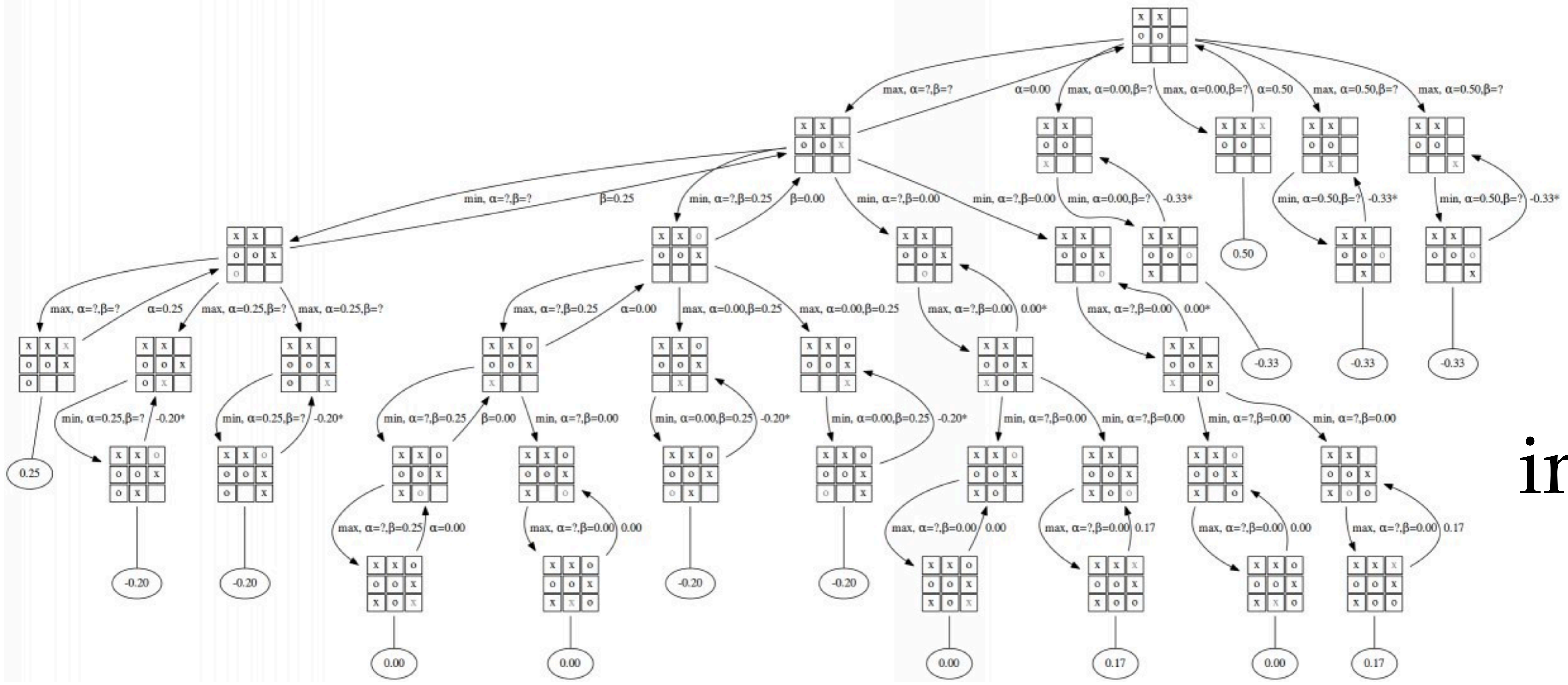
search for implementation

Automatically searching for implementations

key idea: use a lightweight neural net to predict the performance

$$y = \sum x_i$$

algorithm



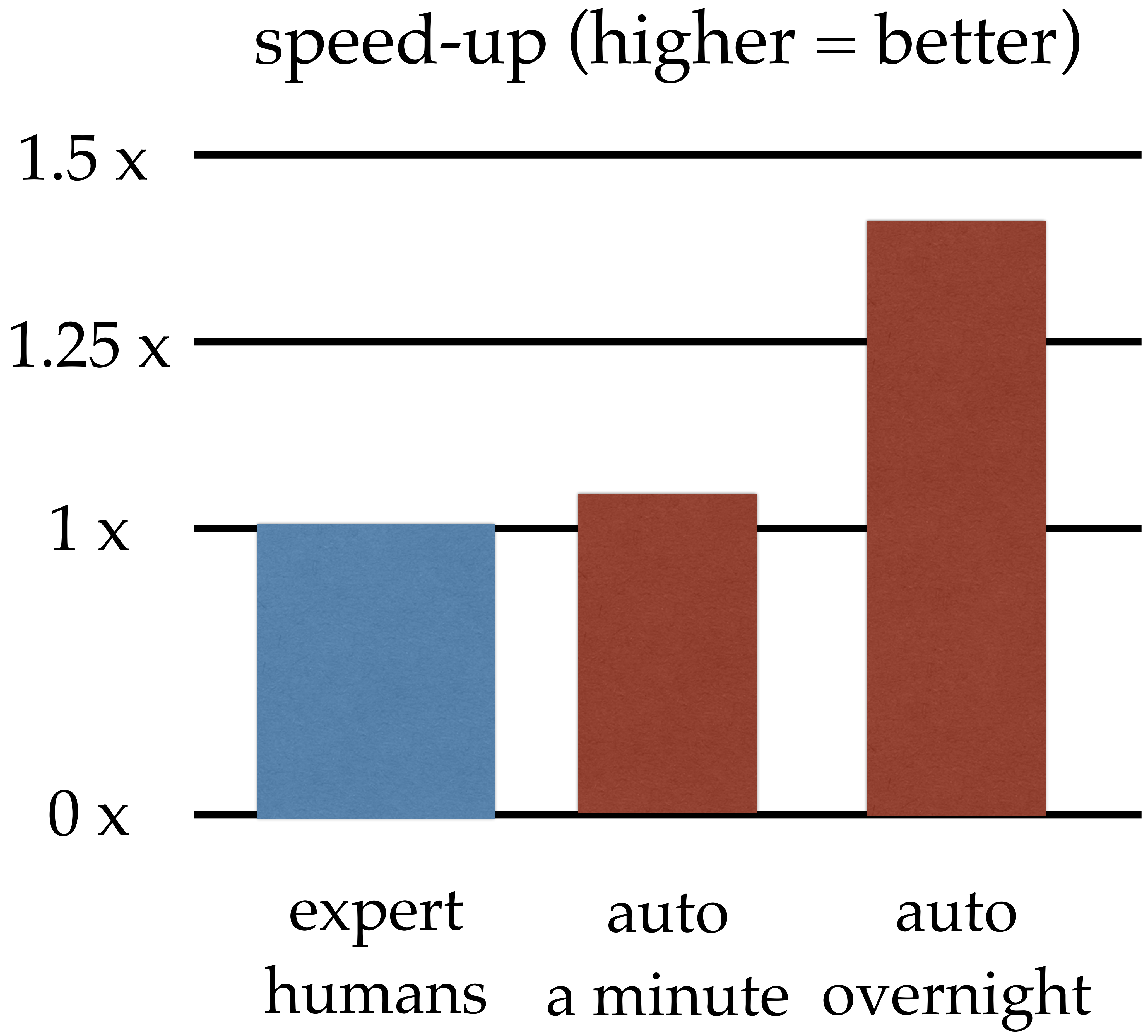
implementation

Adams, Ma, Anderson, Baghdadi, **Li**, Gharbi, …, 2019

Anderson, Adams, Ma, **Li**, Ragan-Kelley, 2021

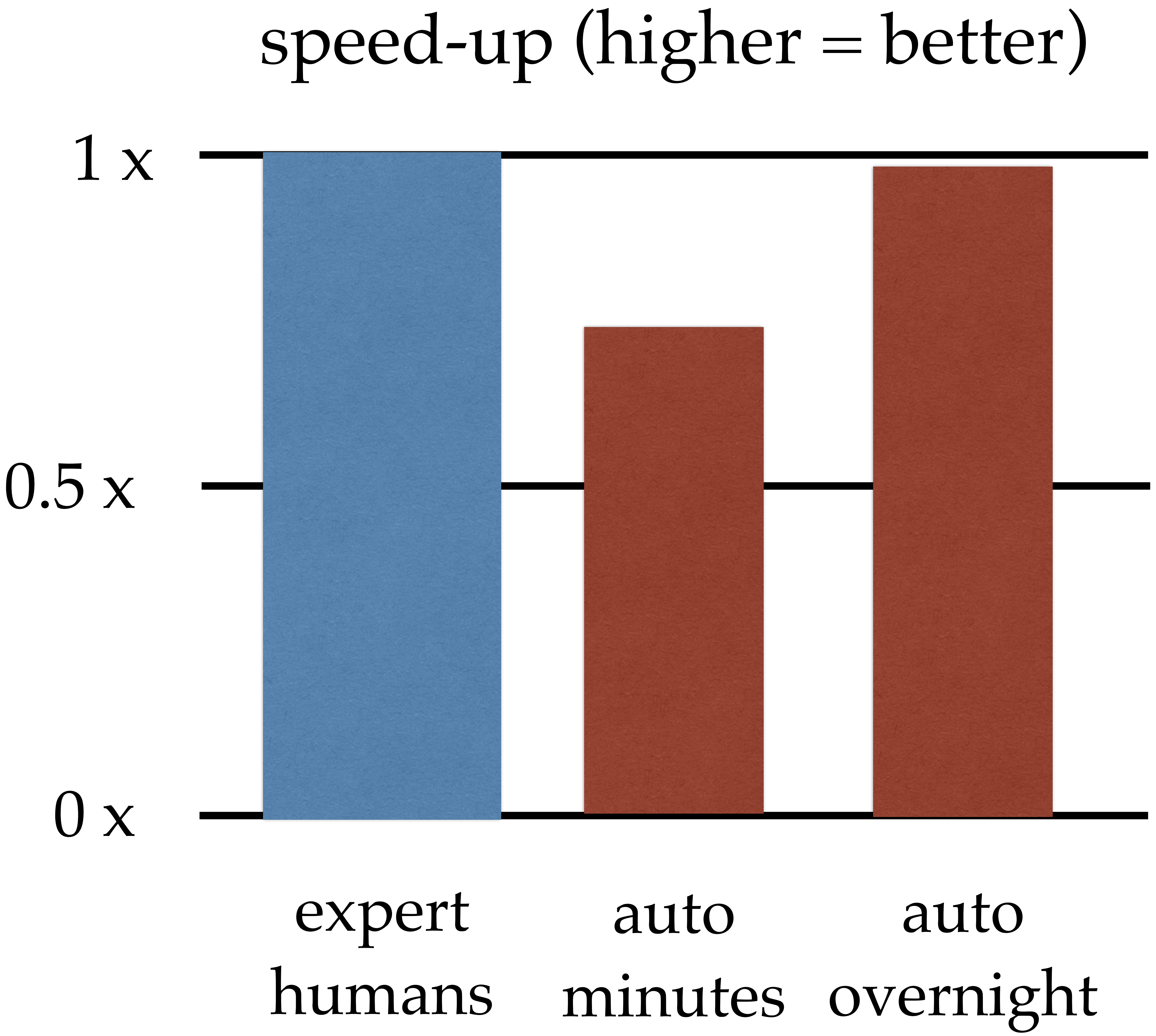
Benchmark: 16 imaging / ML pipelines

on a CPU



e.g. local laplacian filter, resnet 50
matmul, non-local means

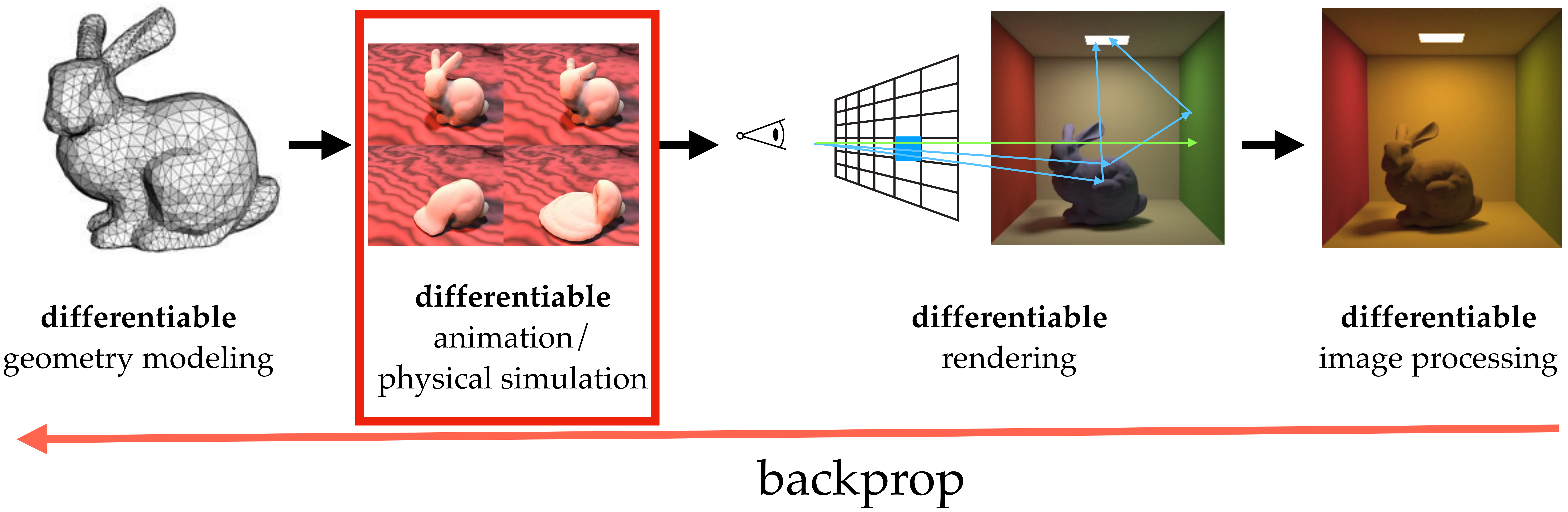
Also works for GPUs



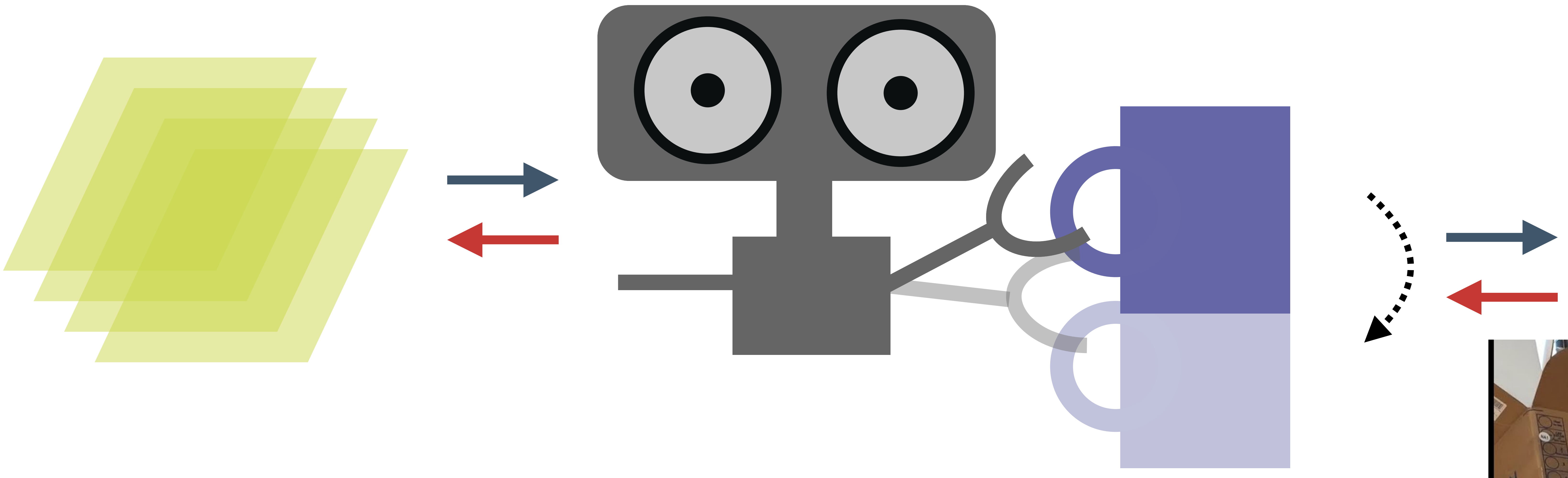
18 pipelines

Differentiable graphics

connects classical graphics algorithms with
modern data-driven methods **through derivatives**



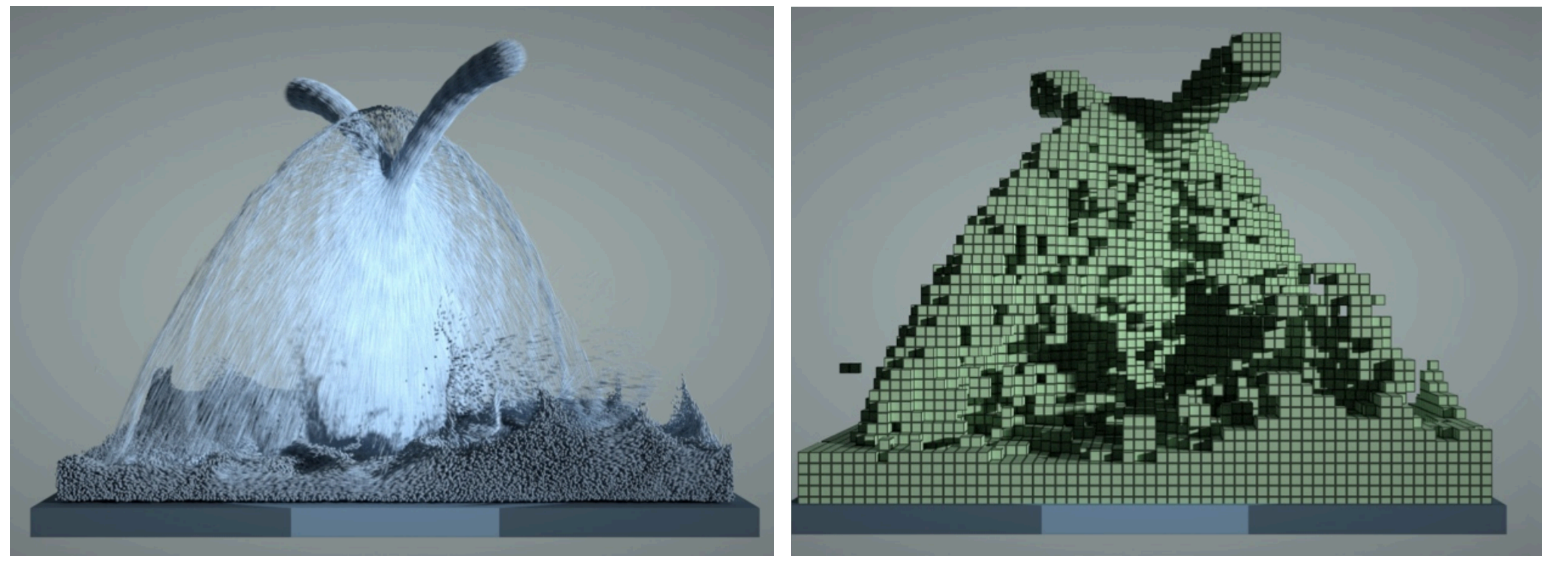
Differentiating physics enables intelligent decision



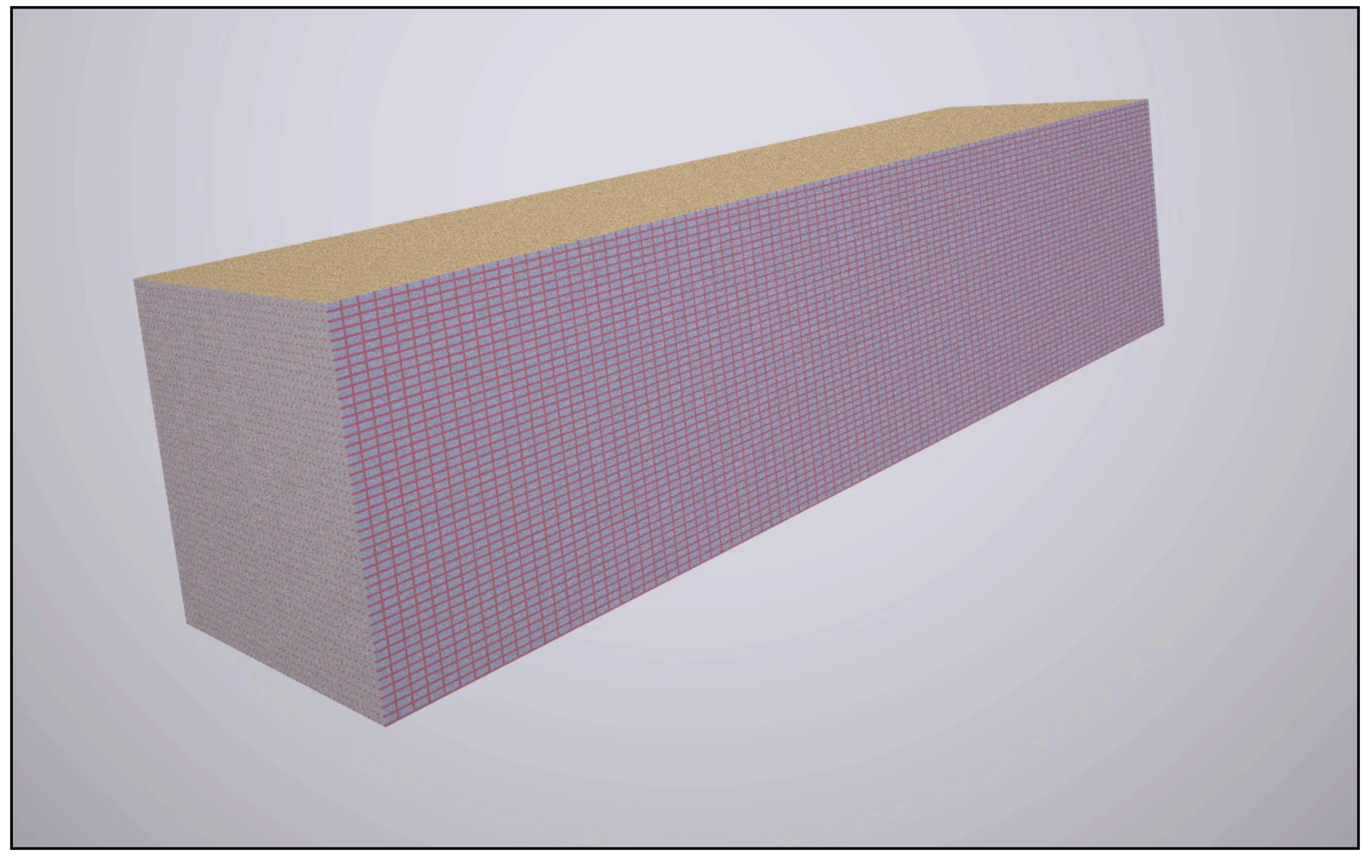
(not my work!)

Challenges: 3D scalability & discontinuities

large-scale physical simulation requires sparse data structure

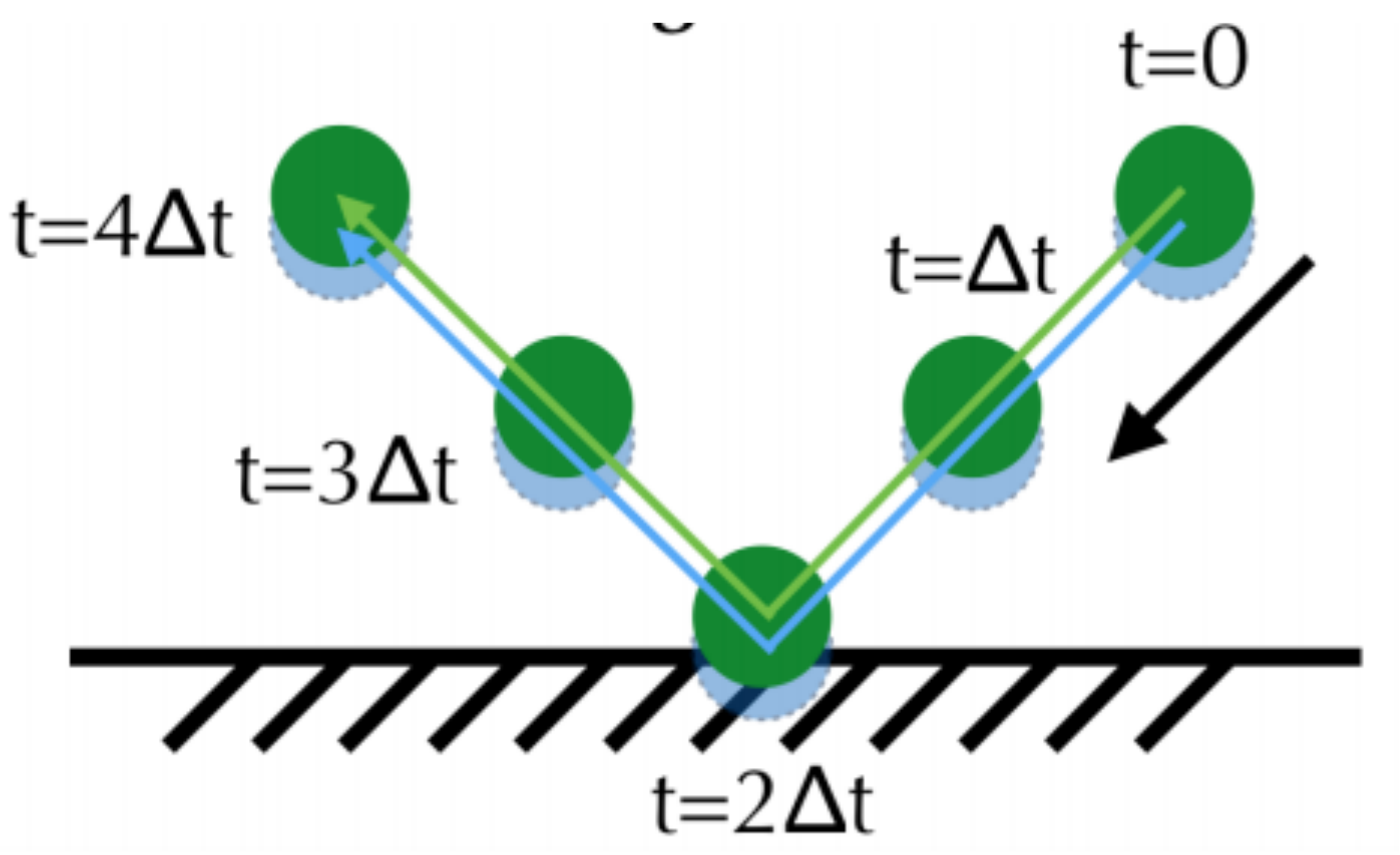


512x512x512



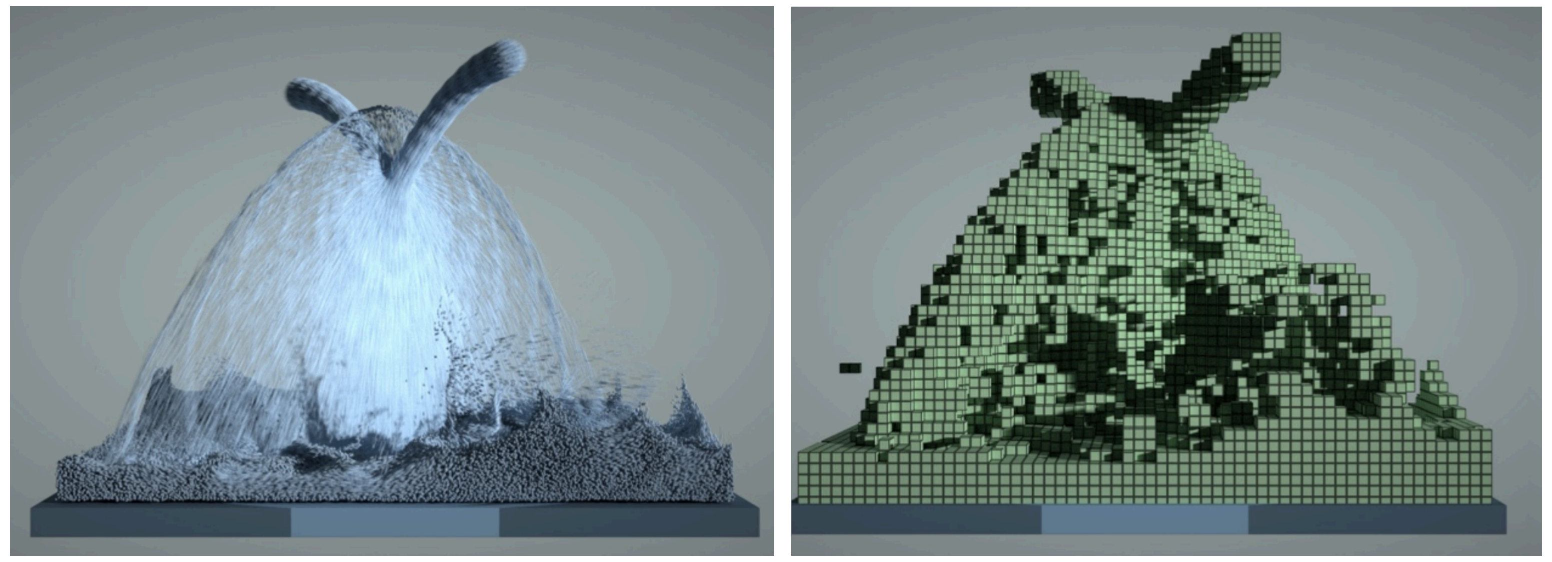
3000x2400x1600

physical phenomenon such as contact leads to discontinuities

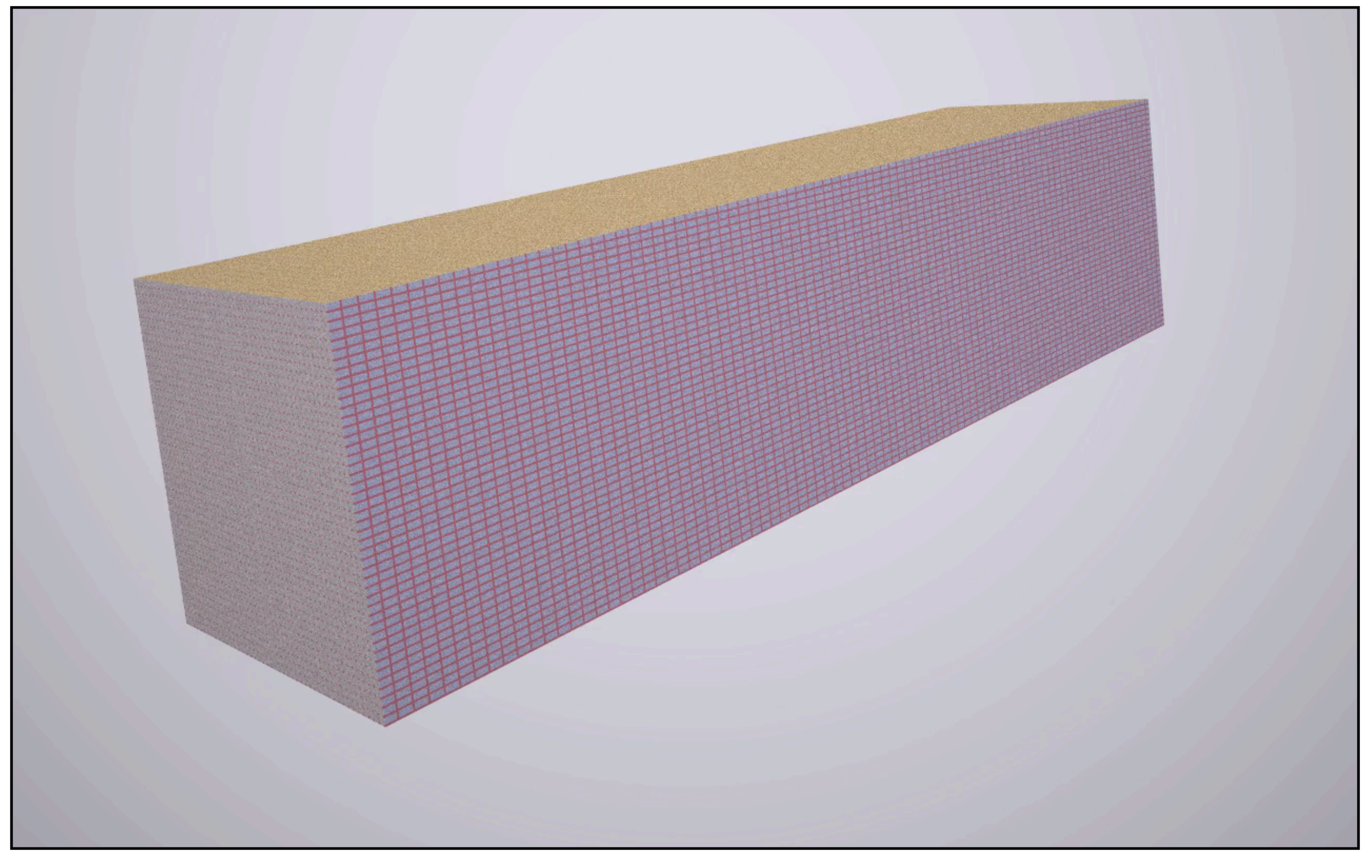


Challenges: 3D scalability & discontinuities

large-scale physical simulation requires sparse data structure



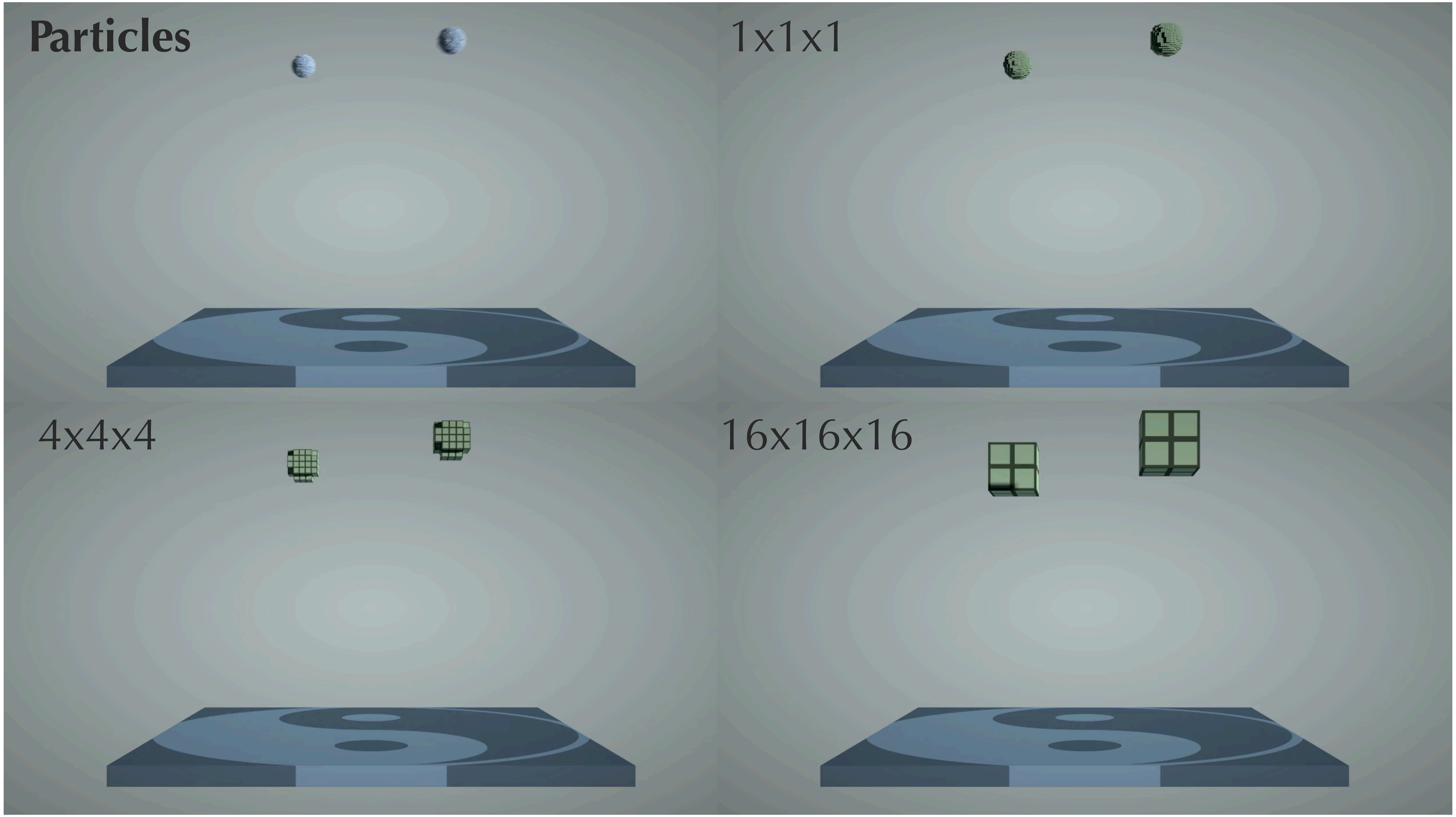
512x512x512



3000x2400x1600

physical phenomenon such as contact leads to discontinuities

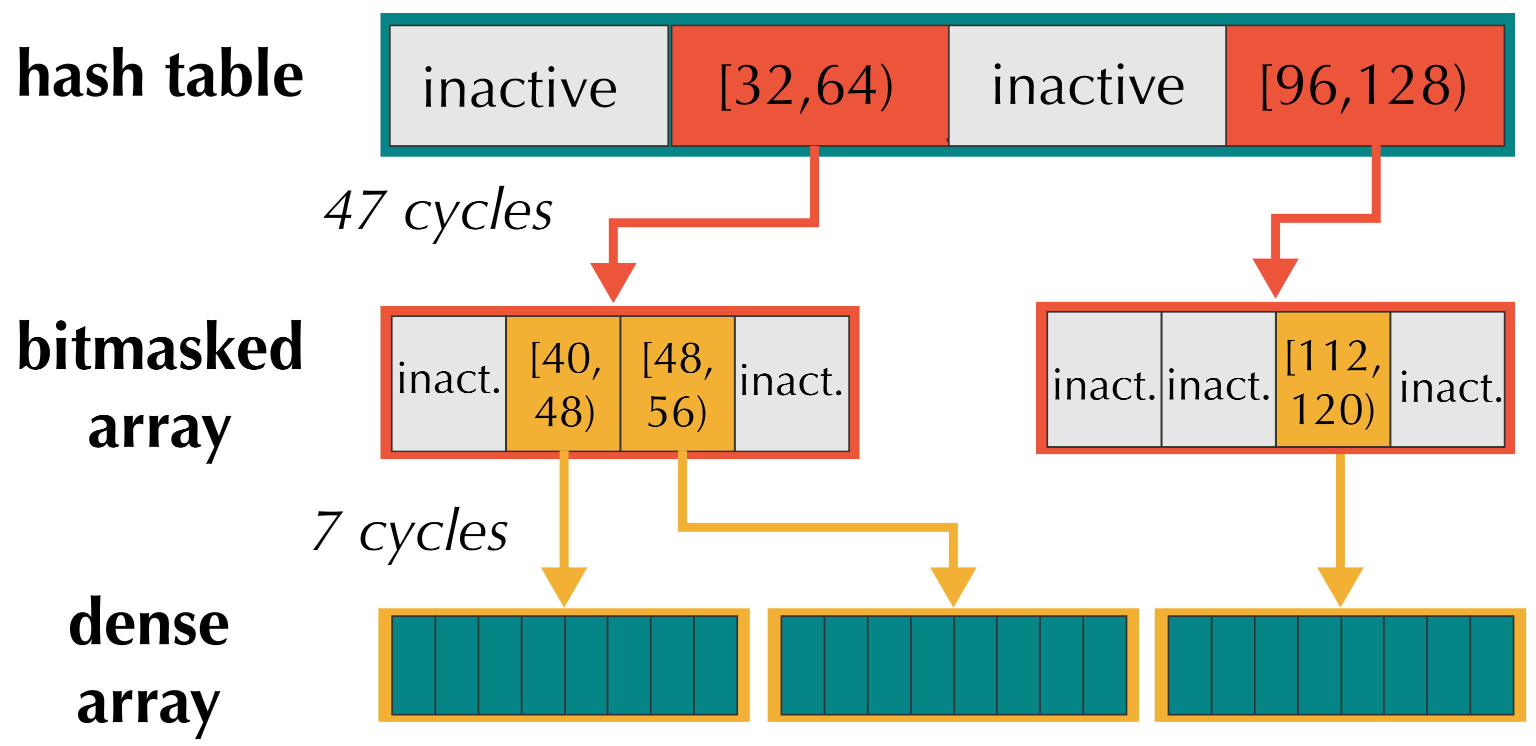
Hierarchical sparse array efficiently models sparsity



Taichi: a differentiating compiler for hierarchical sparse arrays

- decouple sparse data structure and access
  - compiler outputs optimized code

```
f.layout =
```




```
f[x, y, z]
```

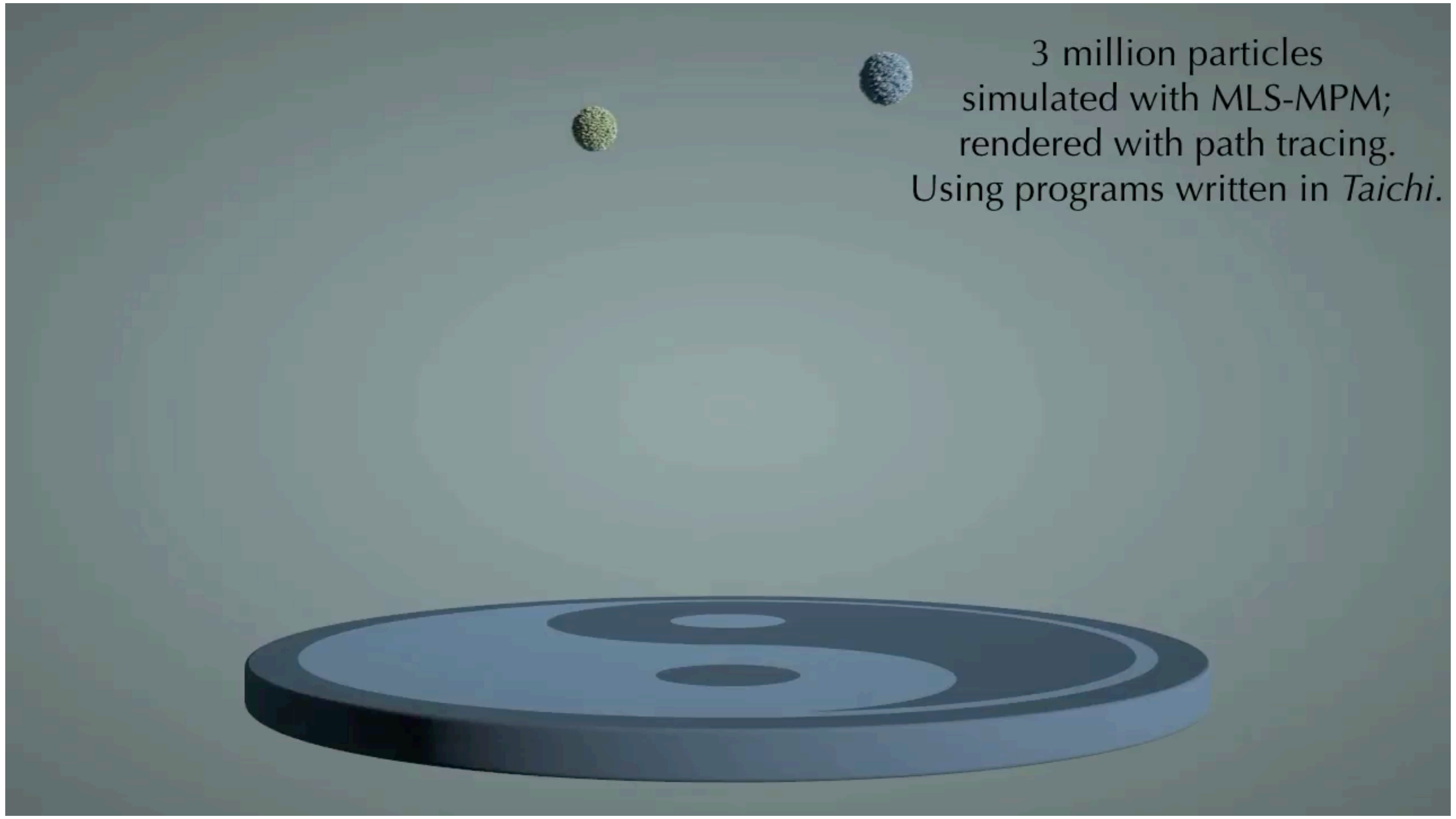
algorithm:
access like dense array

implementation:
specify layout separately

*Hu, **Li**, Anderson, Ragan-Kelley, Durand 2019*
*Hu, Anderson, **Li**, Sun, Carr, Ragan-Kelley, Durand 2020*

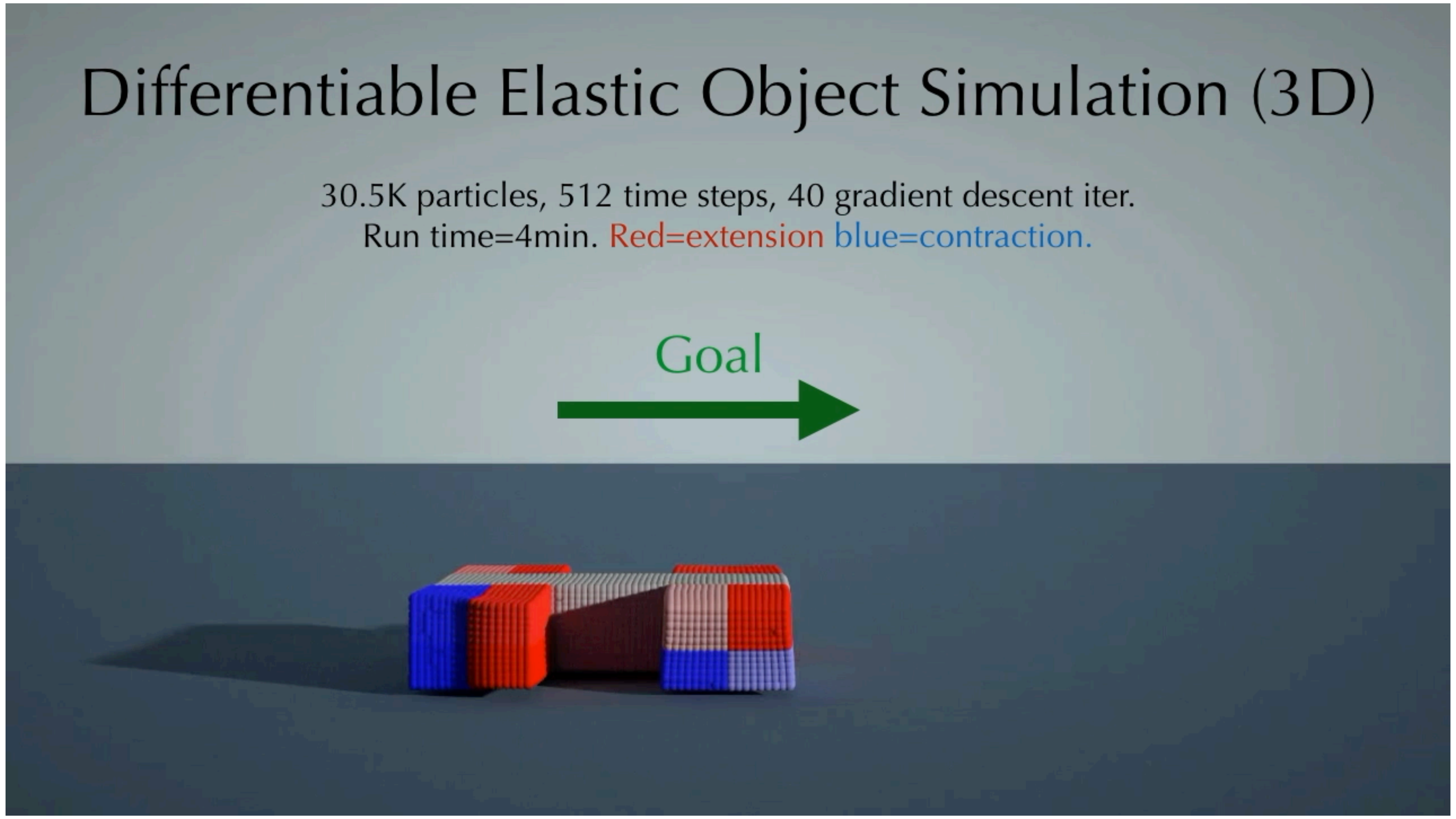
Taichi makes simulation code shorter and faster

- GPU variant of material point method [Gao 2018]
  - code 13x shorter, 1.2x faster



First step towards large-scale model-based RL

propagate gradients through simulation to robot controllers



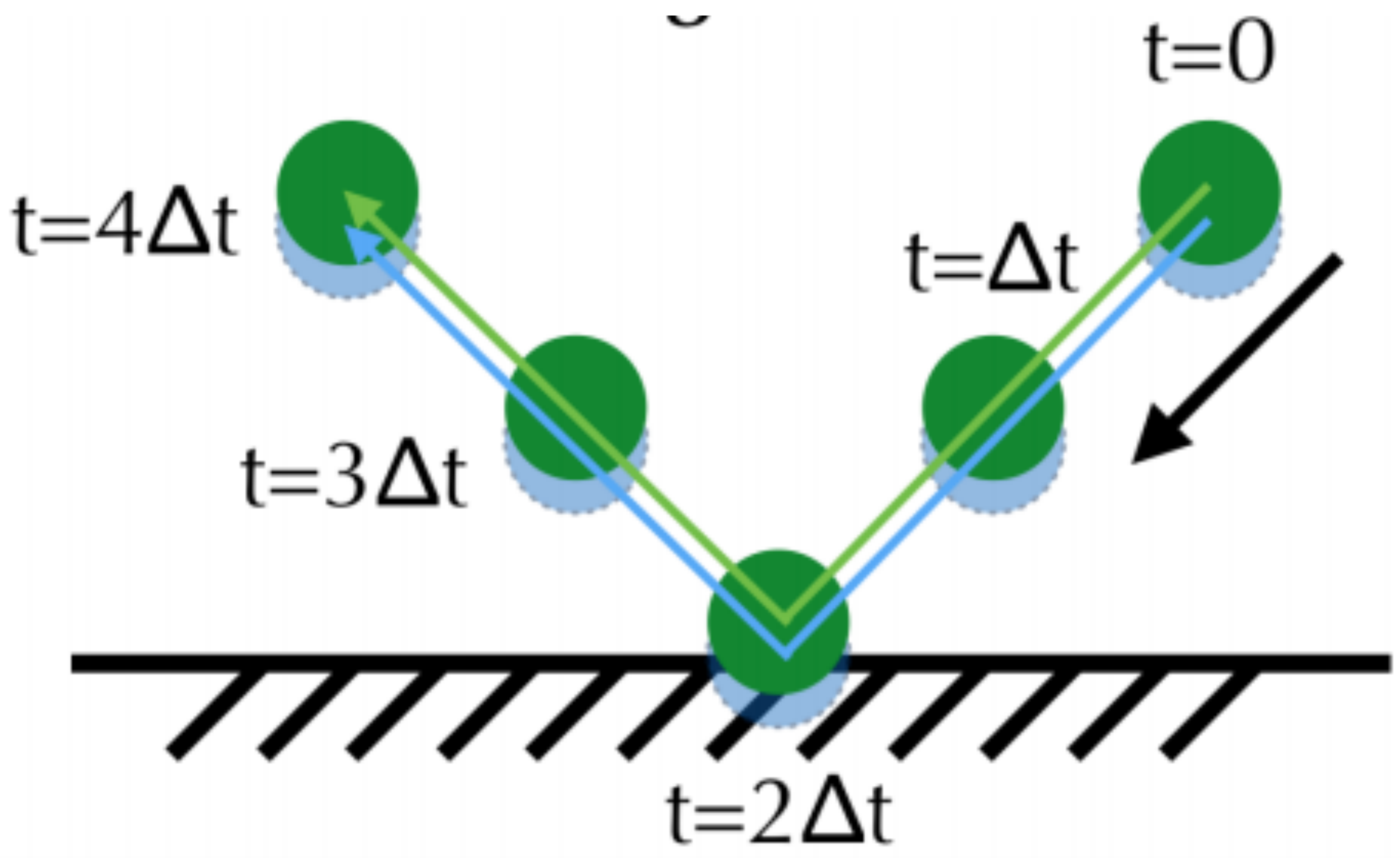
Challenges: 3D scalability & discontinuities

large-scale physical simulation
requires sparse data structure

512x512x512

3000x2400x1600

physical phenomenon such as
contact leads to discontinuities

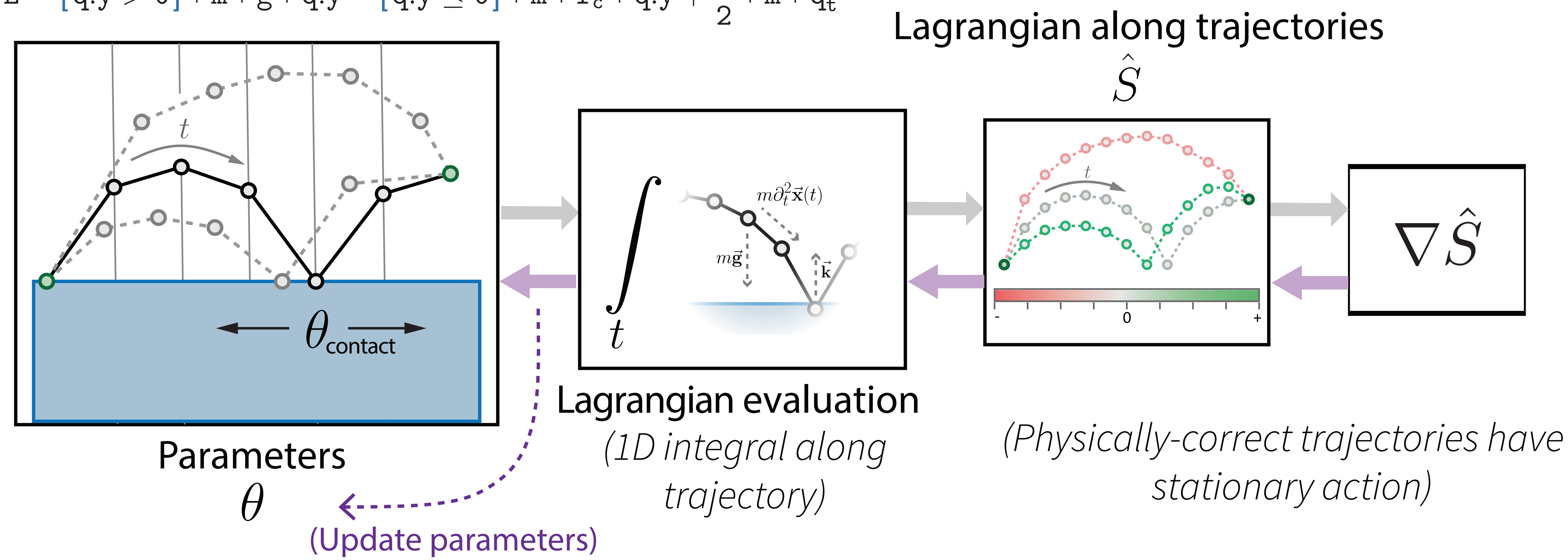


Many physics problems can be modeled as integrals

just like rendering!

$$\mathtt{S}(\theta) = \int_{\mathtt{t}=\mathtt{t}_0}^{\mathtt{t}_1} \mathtt{L}(\mathtt{q}, \mathtt{q}_\mathtt{t}, \mathtt{t})$$

$$\mathtt{L} = [\mathtt{q.y} > 0] * \mathtt{m} * \mathtt{g} * \mathtt{q.y} - [\mathtt{q.y} \leq 0] * \mathtt{m} * \mathtt{f}_\mathtt{c} * \mathtt{q.y} + \frac{1}{2} * \mathtt{m} * \mathtt{q}_\mathtt{t}^2$$



Need a language for describing and differentiating integrals

- if statements = discontinuities

$$\frac{\partial}{\partial p}\int_{x=0}^{x=1}$$

```
def f(x, p):
  if g(x, p) > 0:
    return x * x
  else:
    return x
```

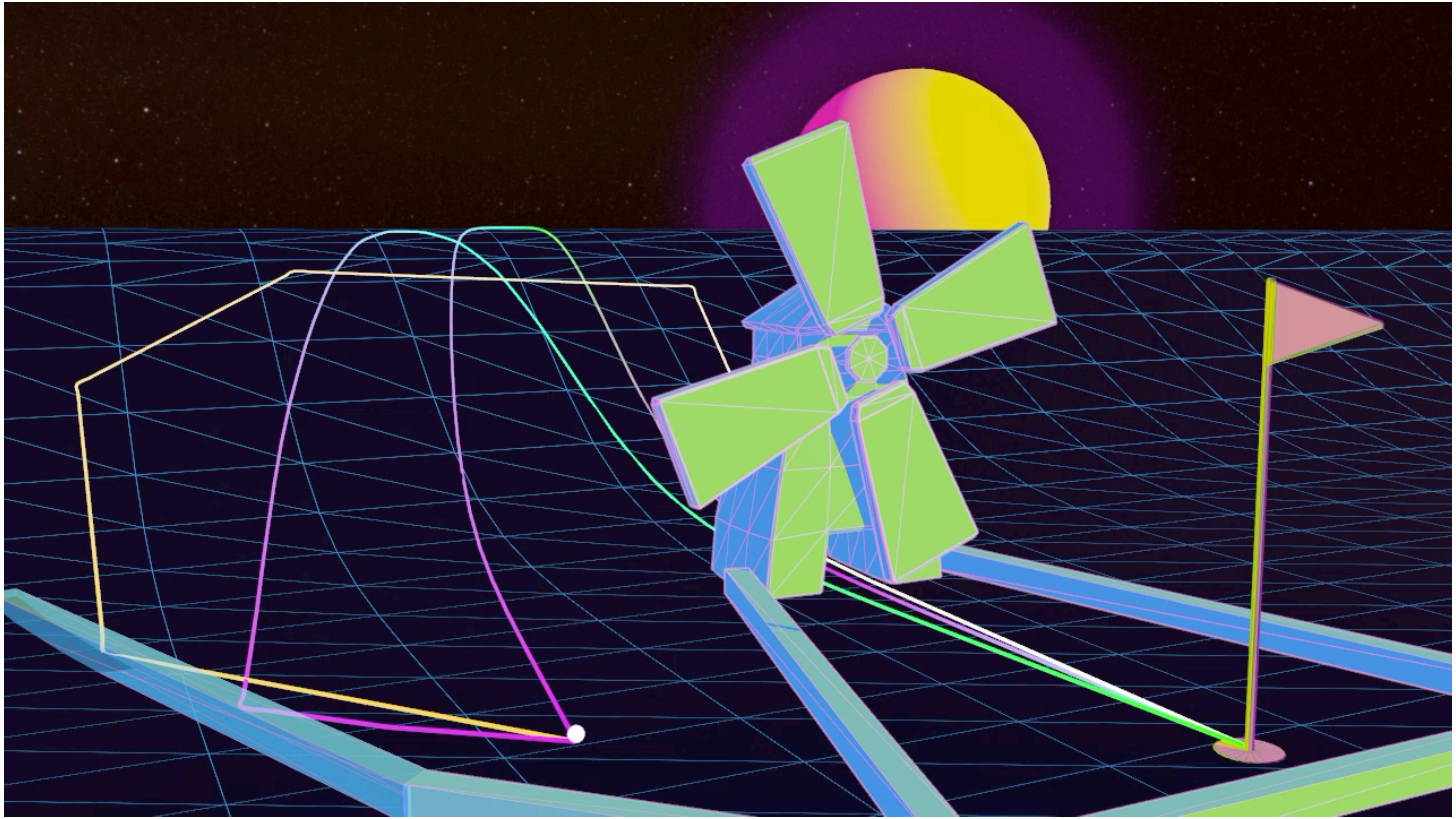
$$\int \ldots \rightarrow \sum \ldots$$

algorithm:
integrals & derivatives

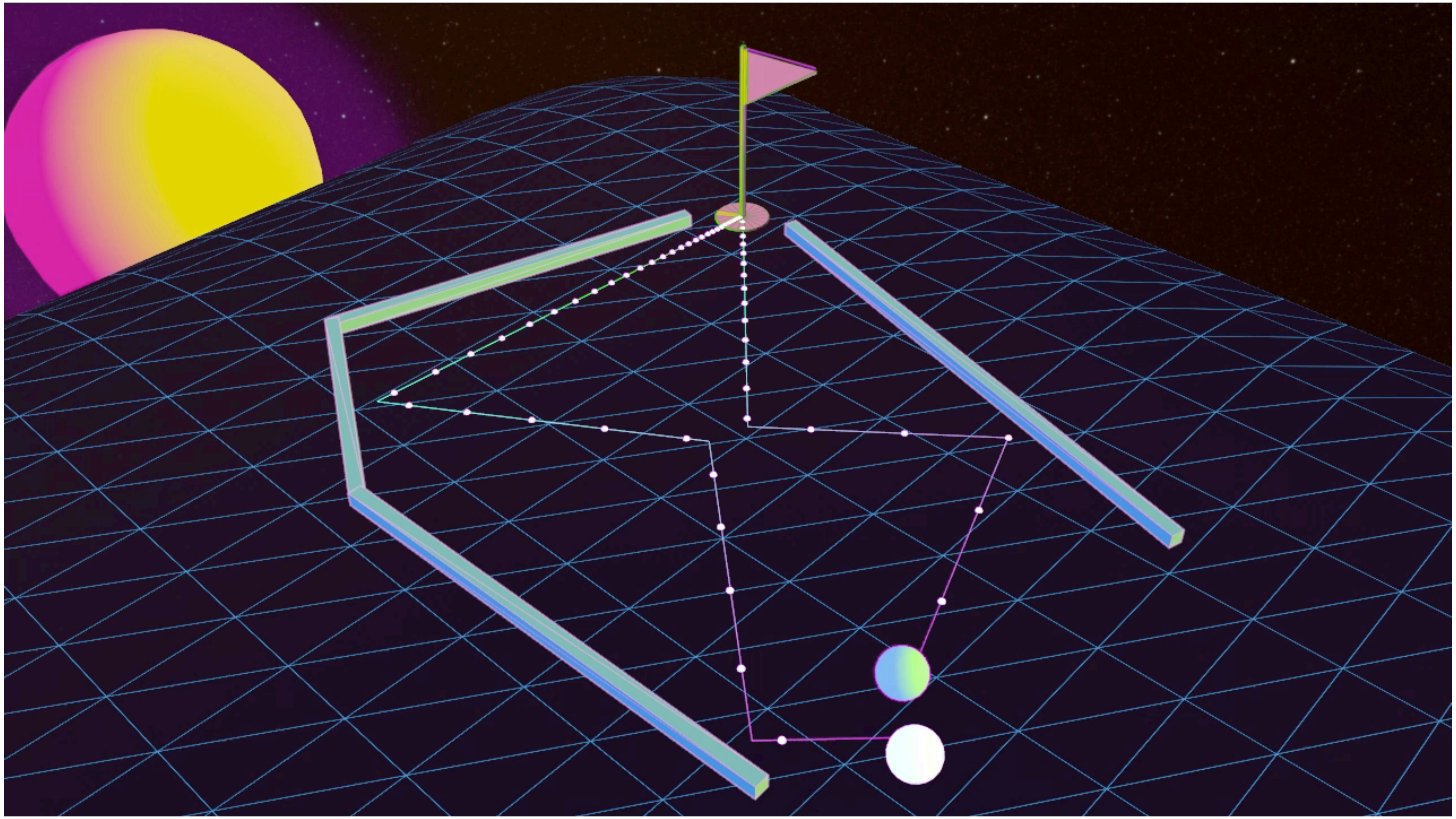
implementation:
integral discretization

*Bangaru, Michel, Mu, Bernstein, **Li**, Ragan-Kelley, 2021*

Application: animation design / motion planning

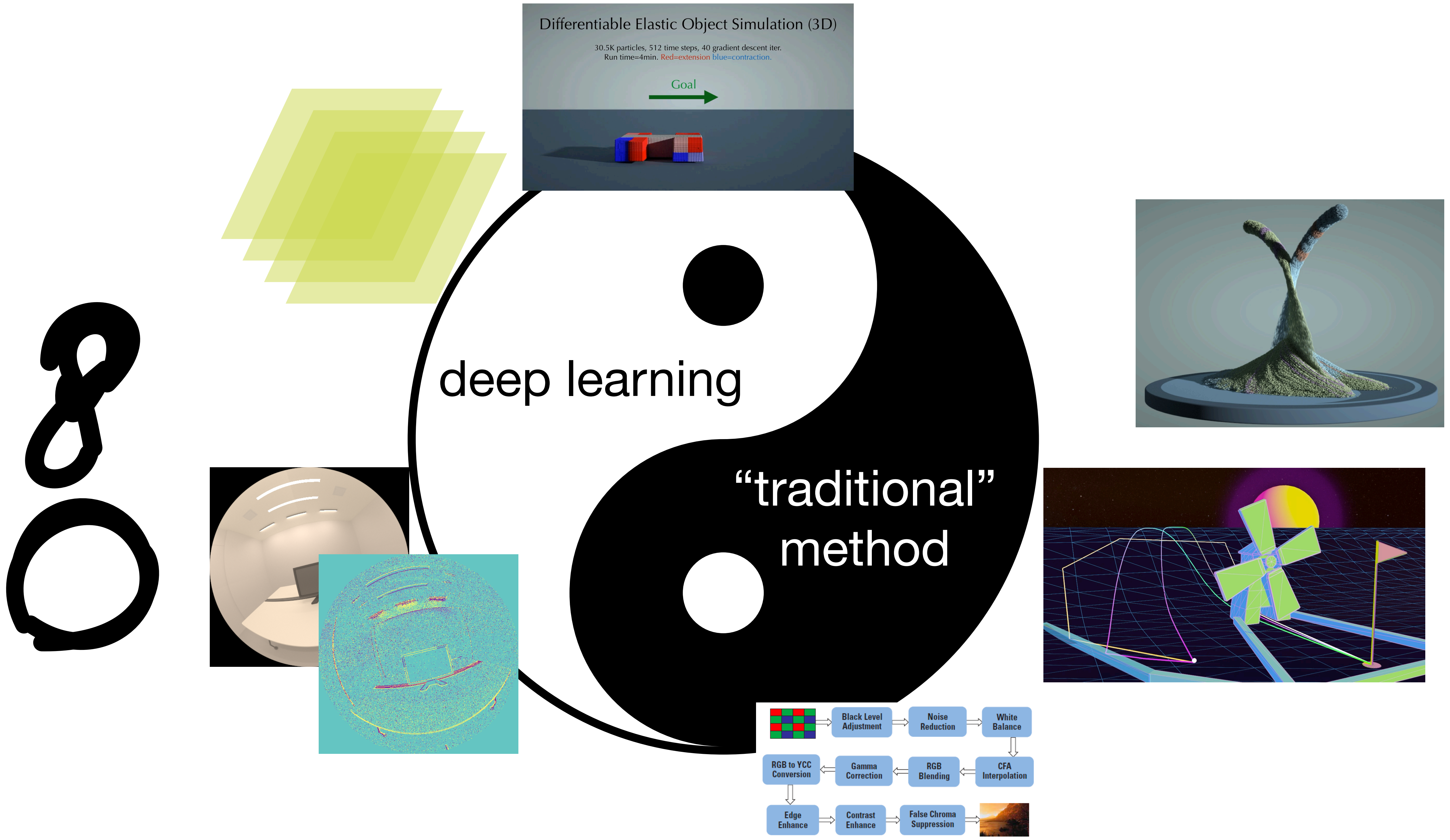


Application: animation design / motion planning



Vision

- differentiable programs bridge deep learning and traditional methods
- our work: extract domain knowledge encoded in graphics algorithms
  - tools: differential calculus and PL/compiler



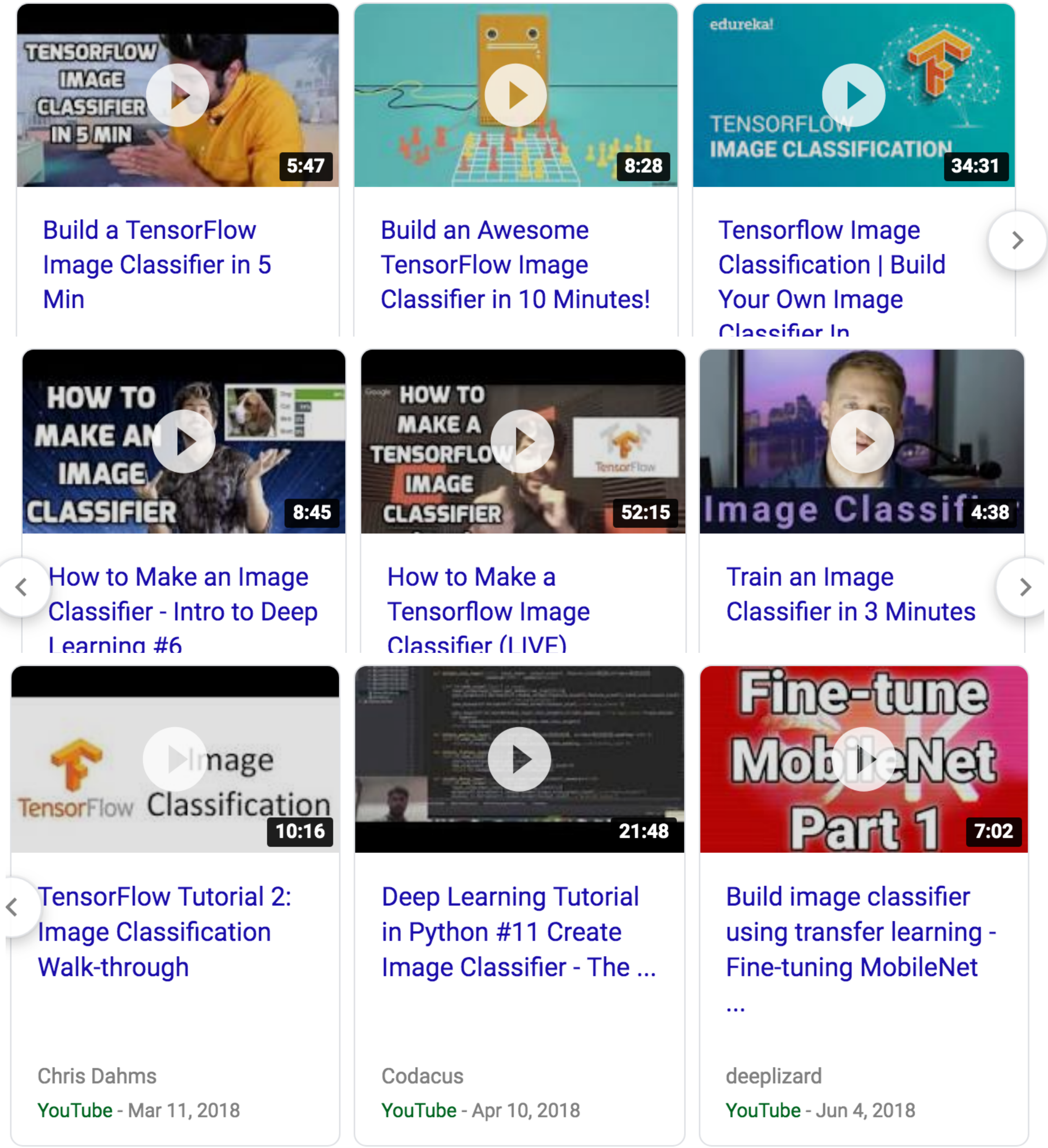
Why was deep learning successful?

Before

required expertise:

- program differentiation
- numerical computing
- GPU hacking
- optimization

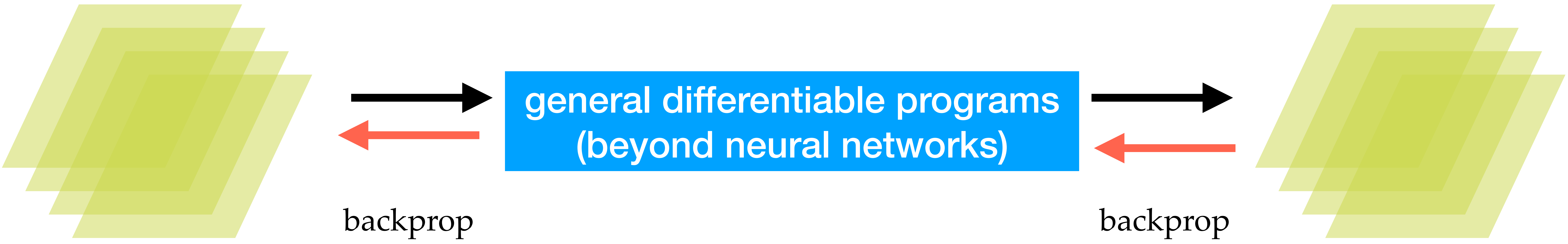
After



slide courtesy of Caroline Lemieux

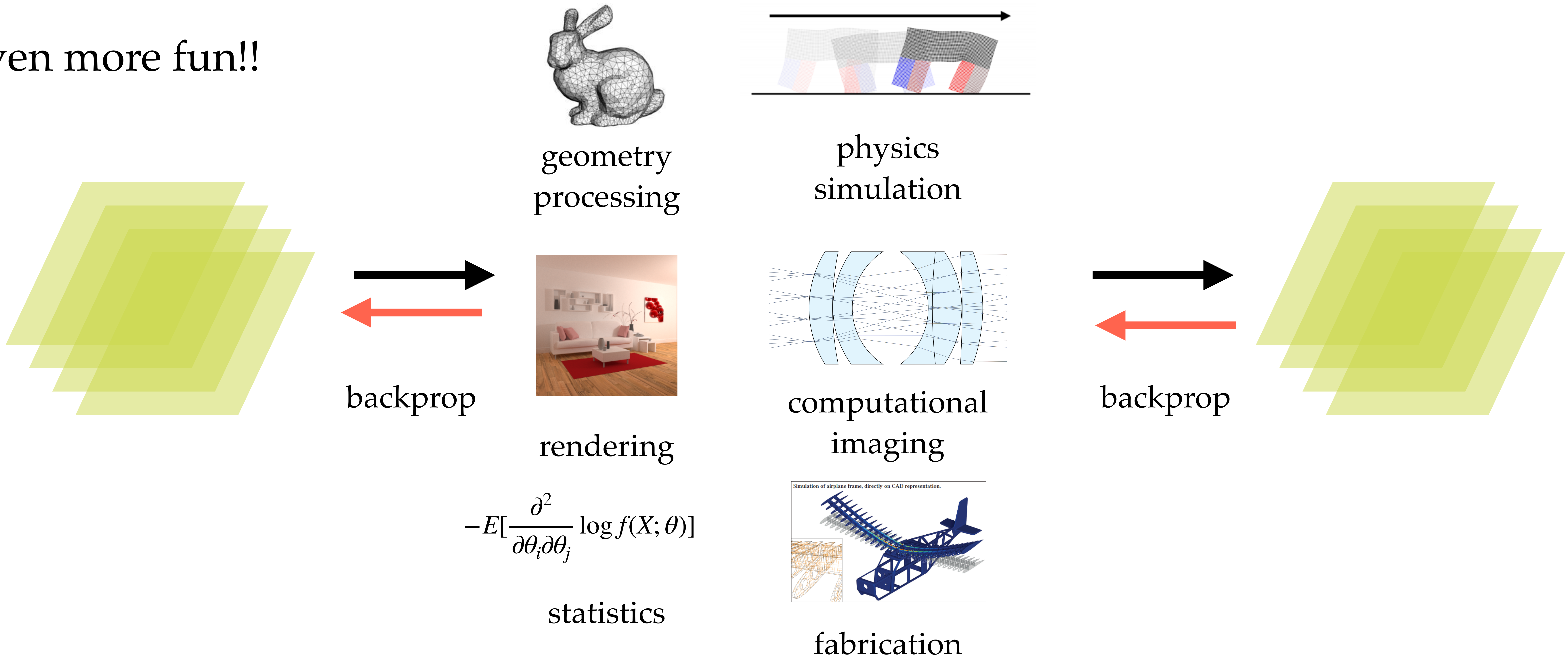
Differentiable programming: next-gen deep learning

- interpretable, controllable, generalizable
  - and fun!



Differentiable visual computing: next-gen graphics / vision

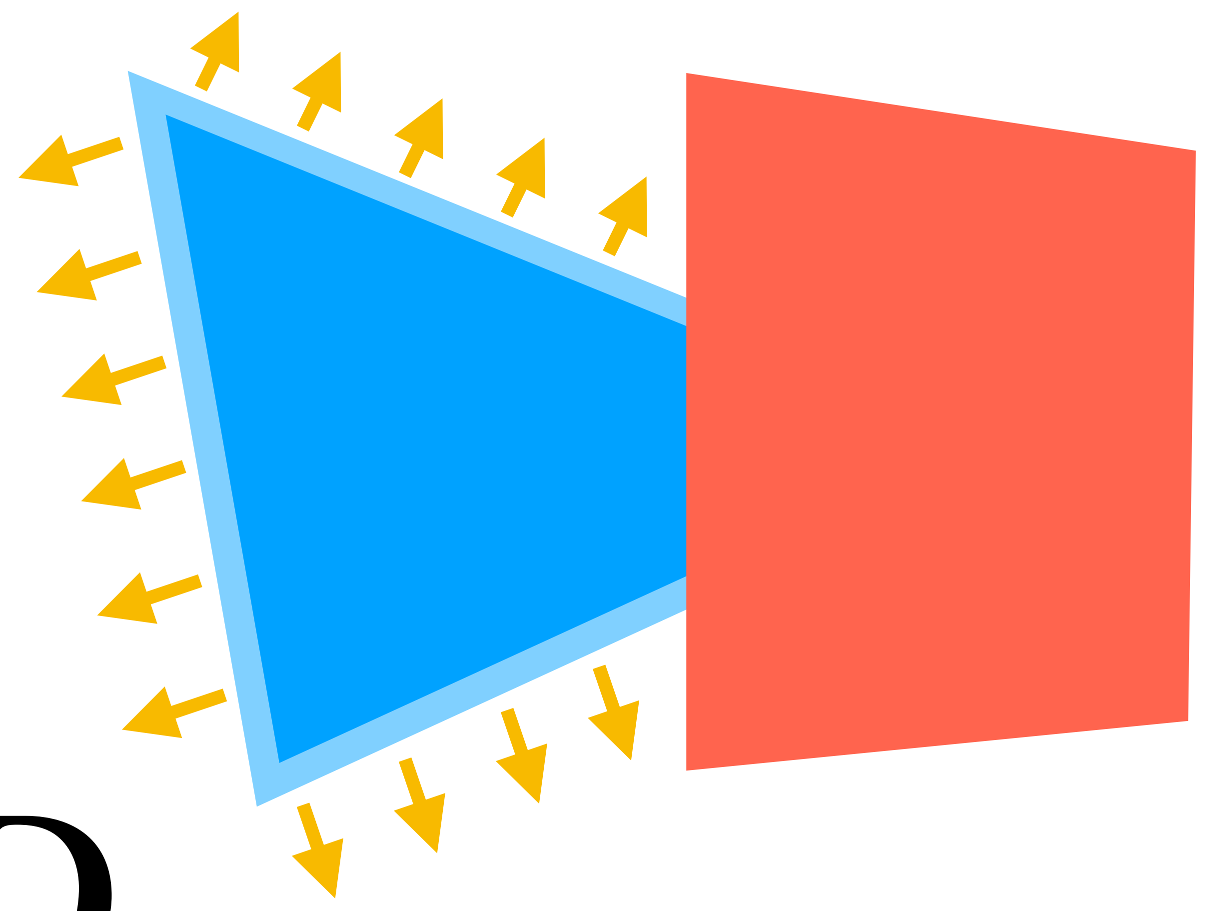
- brings physical inductive bias to graphics / vision systems
  - even more fun!!



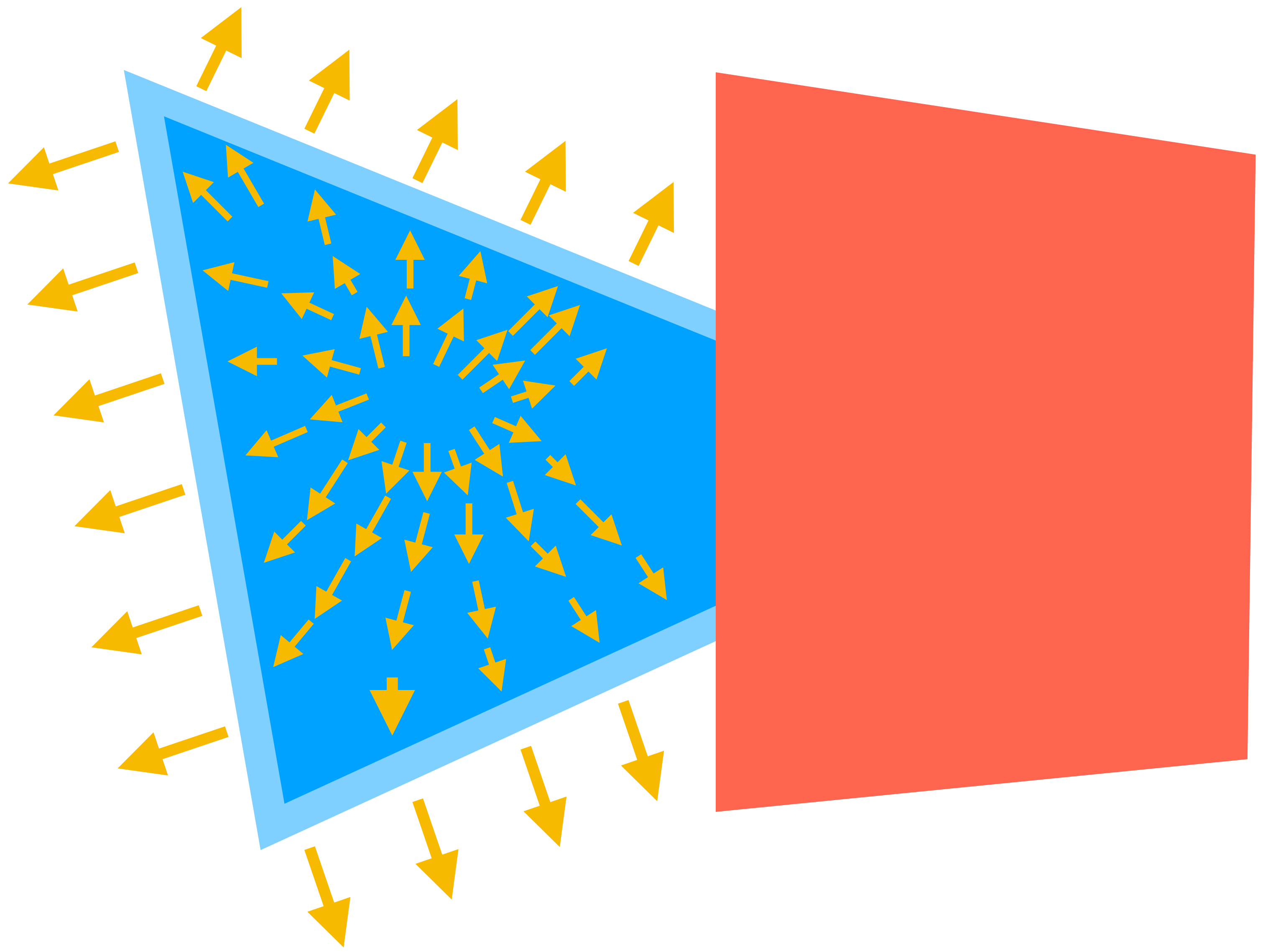
We need more theorems / algorithms

- for deriving & connecting different differential quantities
- for analyzing pros / cons of different estimators

$$\int_{\partial D}$$



$$= \iint_D \nabla \cdot$$



We need better programming languages

- for describing programs that involve integrals and differentiation

$$\frac{\partial}{\partial p}\int_{x=0}^{x=1}$$

```
def f(x, p):
  if g(x, p) > 0:
    return x * x
  else:
    return x
```

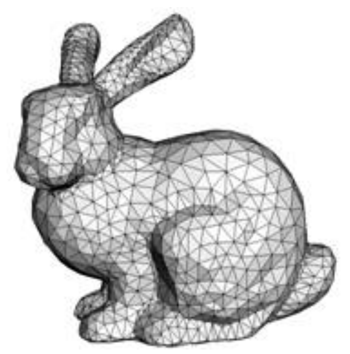
We need better compilers

- for efficiently mapping programs to hardware

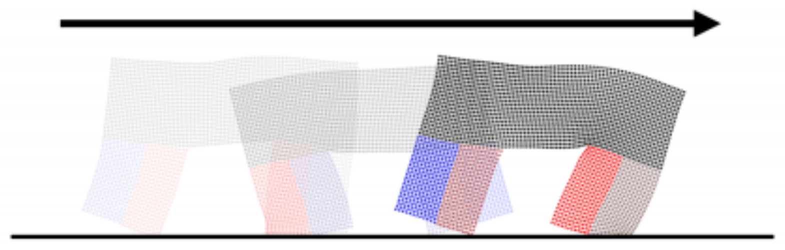
$$\frac{\partial}{\partial p}\int_{x=0}^{x=1}$$

```
def f(x, p):
  if g(x, p) > 0:
    return x * x
  else:
    return x
```

We need our Tensorflow / PyTorch for differentiable visual computing

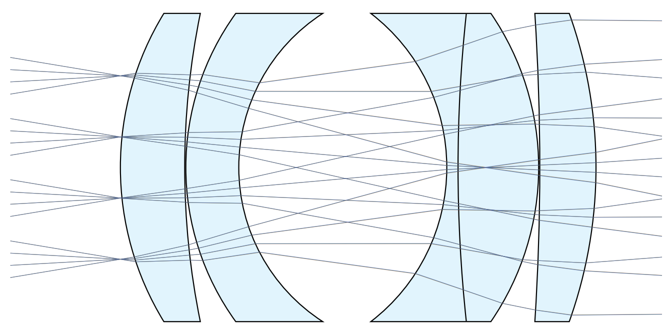


geometry processing



physics simulation

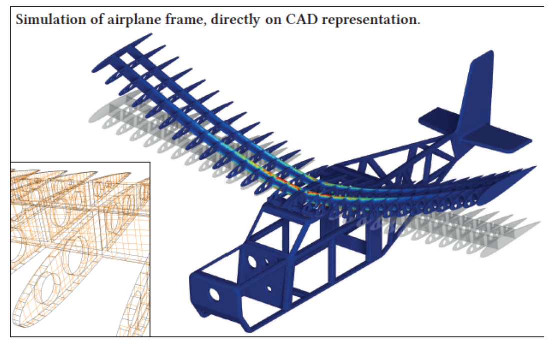
rendering



computational imaging

$$-E[\frac{\partial^2}{\partial\theta_i\partial\theta_j}\log f(X;\theta)]$$

statistics



fabrication

v.s.