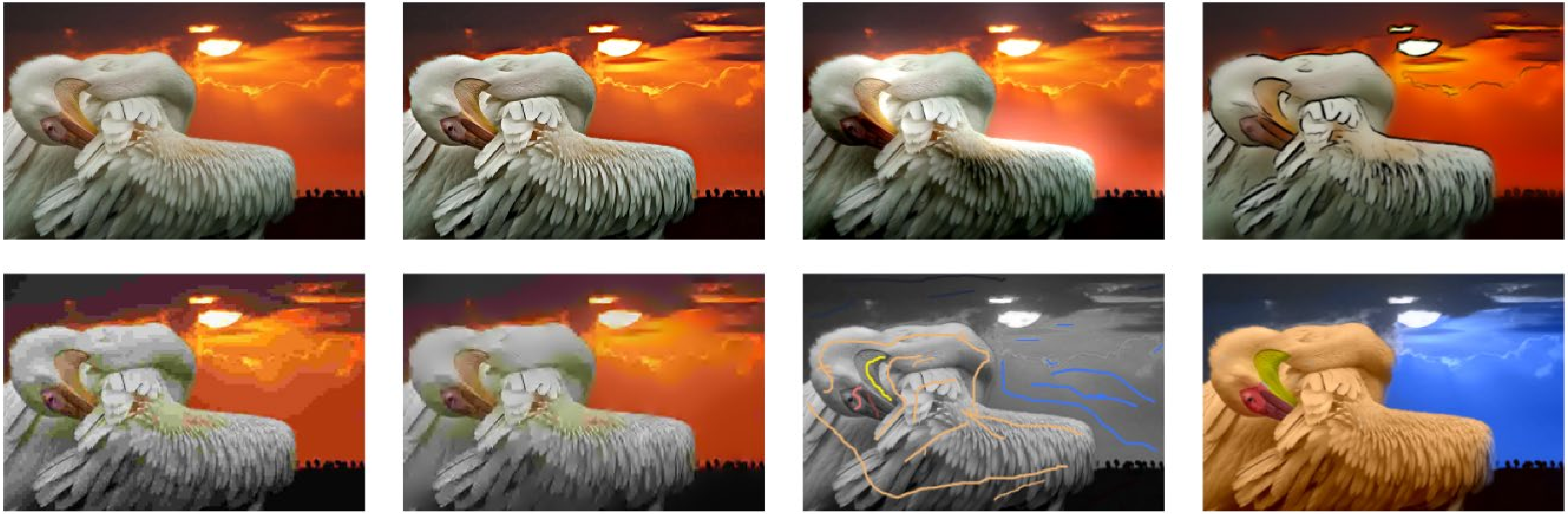
Gradient-domain image processing



15-463, 15-663, 15-862
Computational Photography
Fall 2021, Lecture 10

http://graphics.cs.cmu.edu/courses/15-463

Course announcements

- Homework assignment 3 is out.
  - Due October 18th.
  - Generous bonus components.

Overview of today’s lecture

- Leftover from bilateral filtering.
- Gradient-domain image processing.
- Basics on images and gradients.
- Integrable vector fields.
- Poisson blending.
- A more efficient Poisson solver.
- Poisson image editing examples.
- Flash/no-flash photography.
- Gradient-domain rendering and cameras.

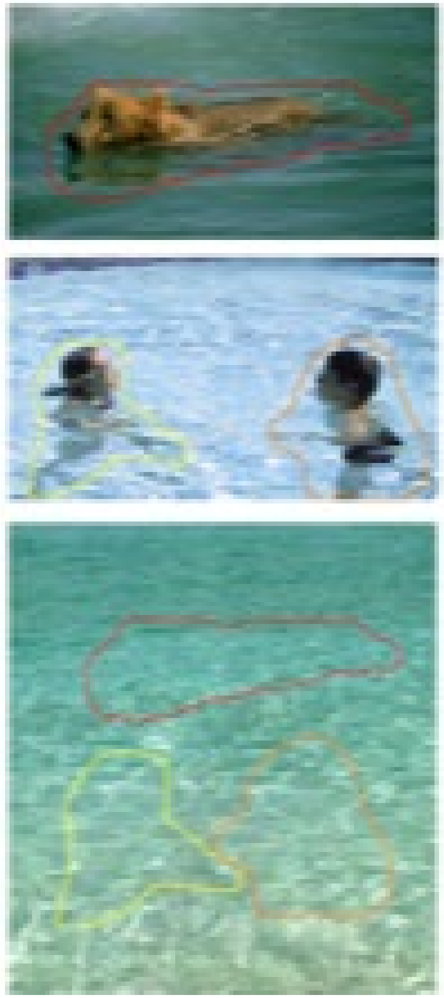
Slide credits

Many of these slides were adapted from:

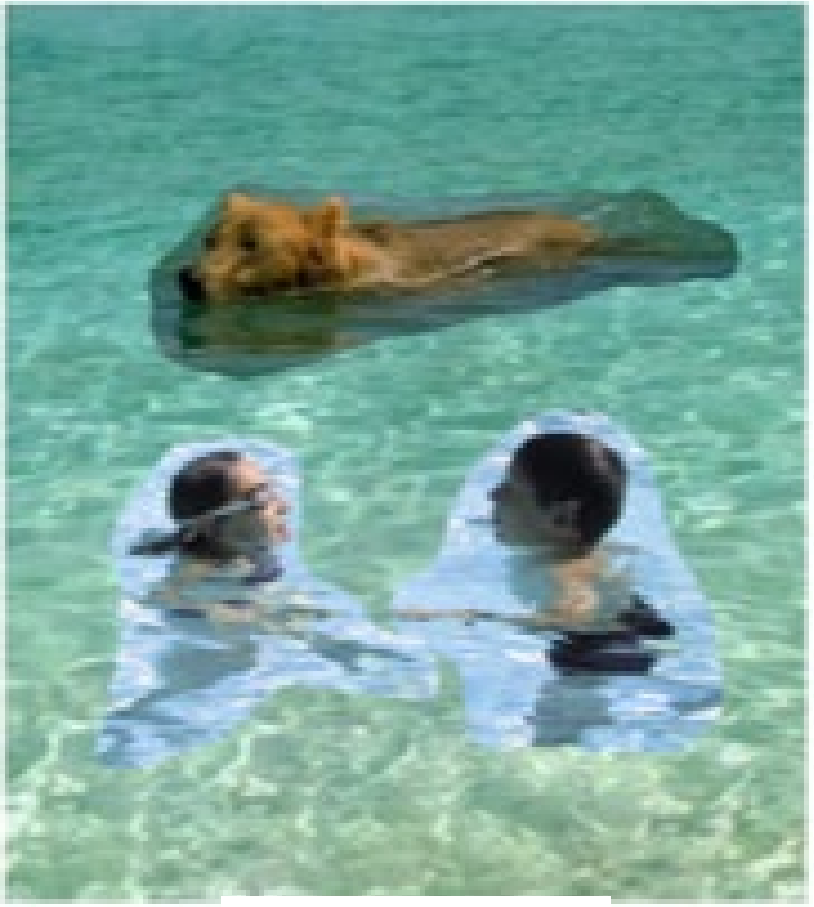
- Kris Kitani (15-463, Fall 2016).
- Fredo Durand (MIT).
- James Hays (Georgia Tech).
- Amit Agrawal (MERL).
- Jaakko Lehtinen (Aalto University).

Gradient-domain image processing

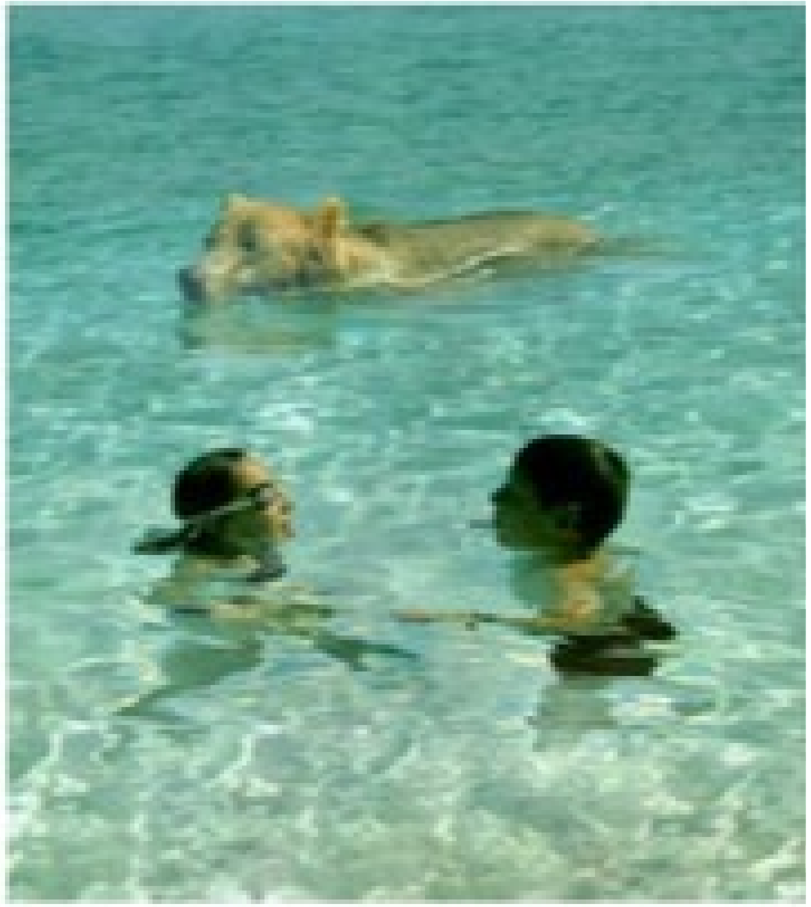
Application: Poisson blending



originals

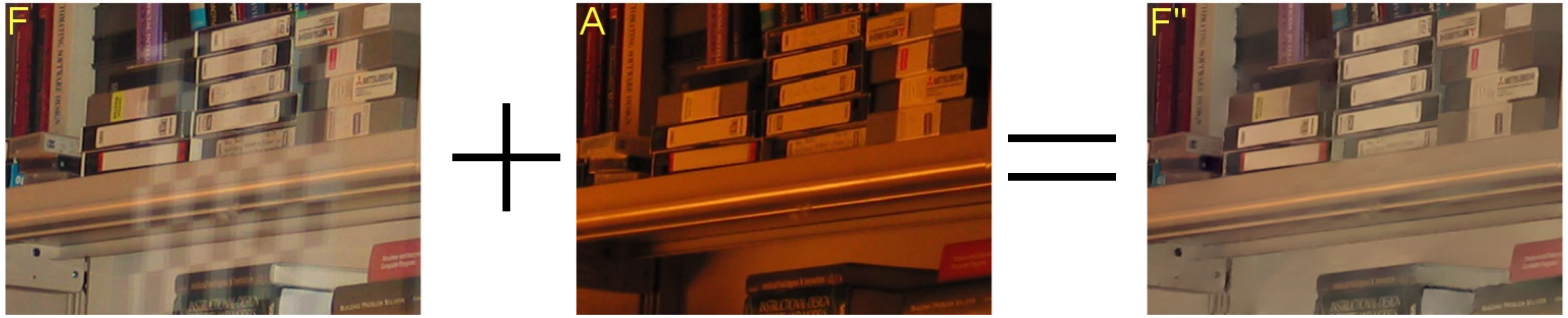


copy-paste

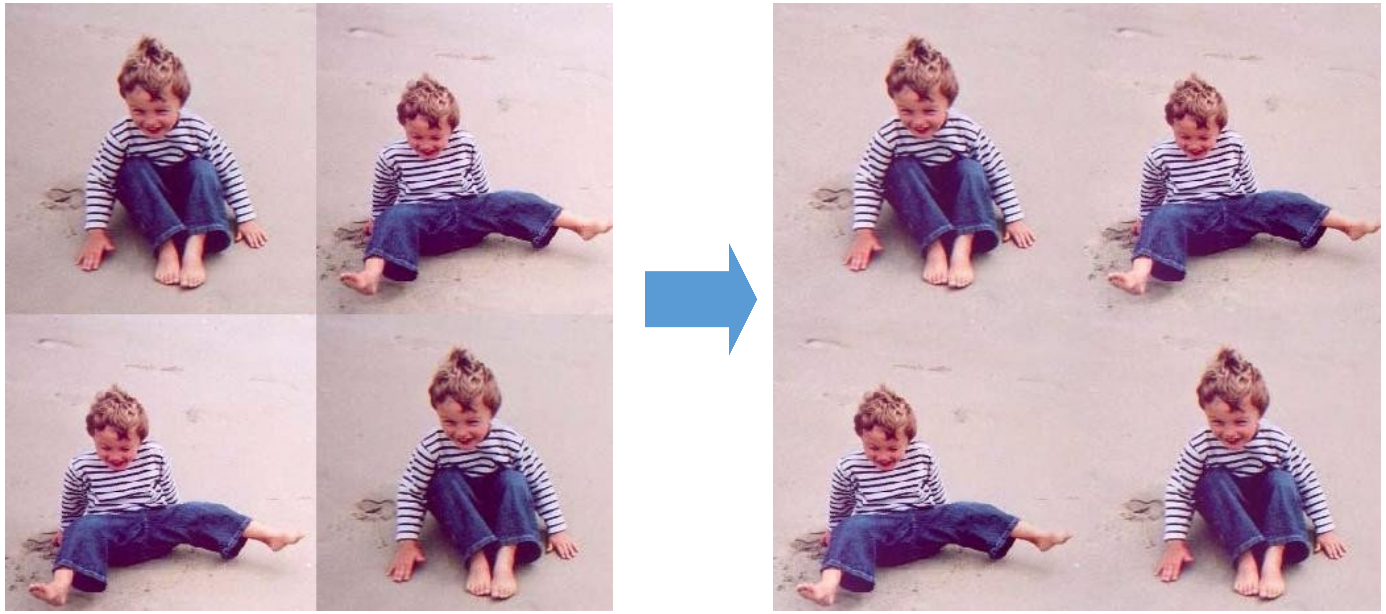


Poisson blending

More applications

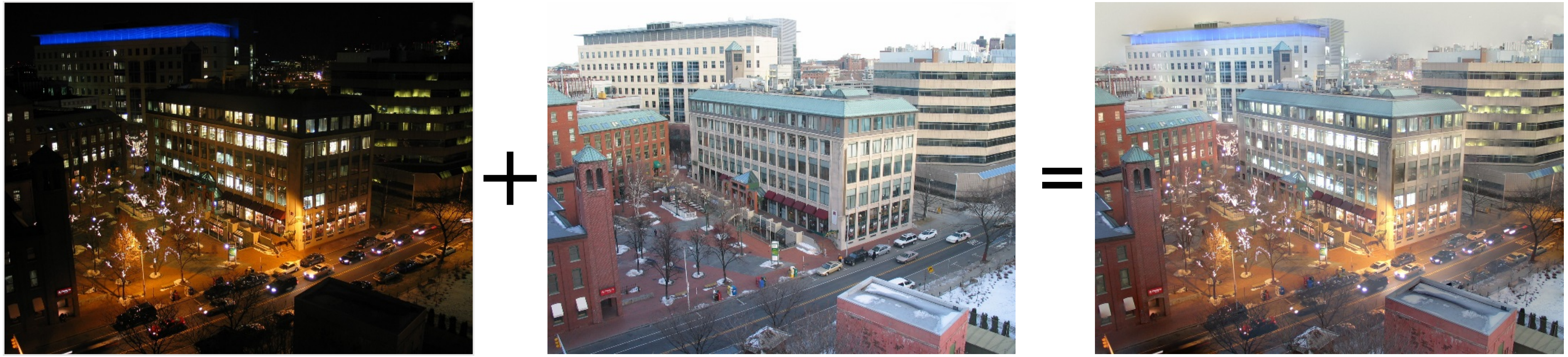


Removing Glass Reflections

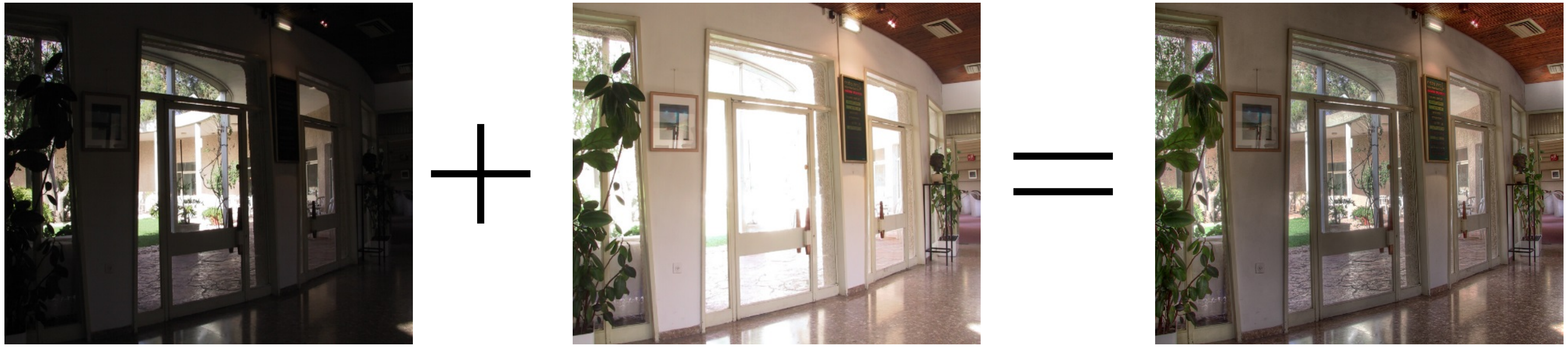


Seamless Image Stitching

Yet more applications

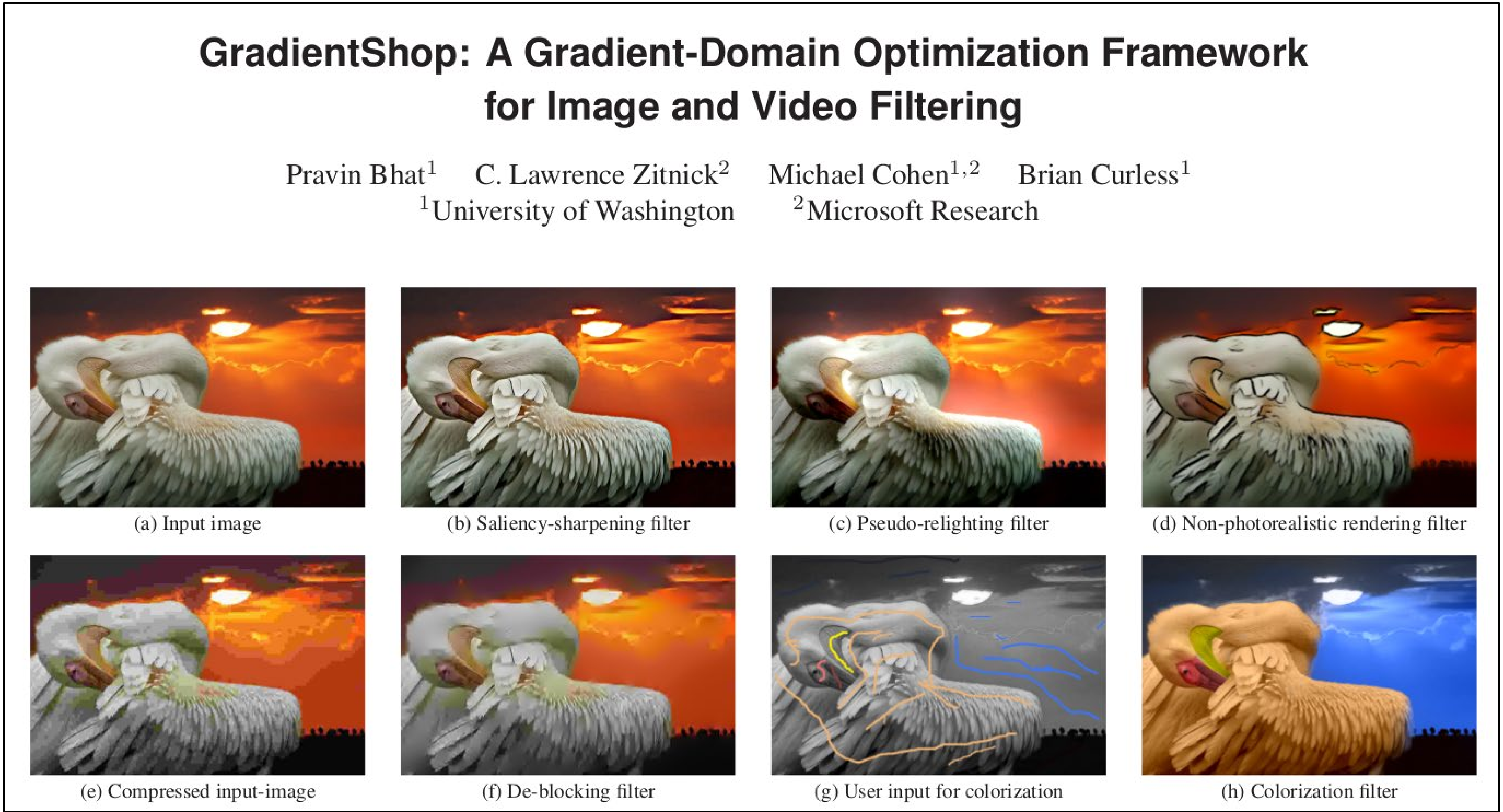


Fusing day and night photos



Tonemapping

Entire suite of image editing tools



GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering

Pravin Bhat[1] C. Lawrence Zitnick[2] Michael Cohen[1,2] Brian Curless[1]
[1]University of Washington [2]Microsoft Research

(a) Input image

(b) Saliency-sharpening filter

(c) Pseudo-relighting filter

(d) Non-photorealistic rendering filter

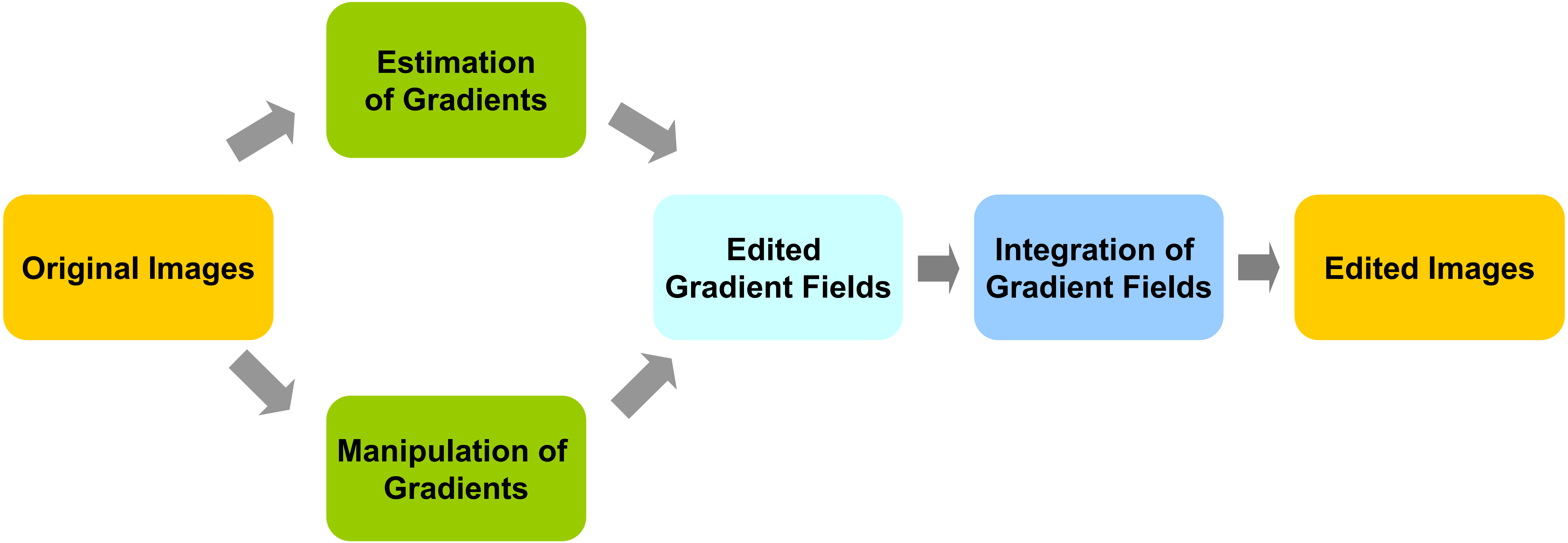
(e) Compressed input-image

(f) De-blocking filter

(g) User input for colorization

(h) Colorization filter

Main pipeline



Original Images

Estimation of Gradients

Manipulation of Gradients

Edited Gradient Fields

Integration of Gradient Fields

Edited Images

Basics of gradients and fields

Some vector calculus definitions in 2D

Scalar field: a function assigning a scalar to every point in space.

$$I(x, y): \mathbb{R}^2 \rightarrow \mathbb{R}$$

Vector field: a function assigning a vector to every point in space.

$$[u(x, y) \quad v(x, y)]: \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

Can you think of examples of scalar fields and vector fields?

Some vector calculus definitions in 2D

Scalar field: a function assigning a scalar to every point in space.

$$I(x, y) \colon \mathbb{R}^2 \to \mathbb{R}$$

Vector field: a function assigning a vector to every point in space.

$$[u(x, y) \quad v(x, y)] \colon \mathbb{R}^2 \to \mathbb{R}^2$$

Can you think of examples of scalar fields and vector fields?

- A grayscale image is a scalar field.
- A two-channel image is a vector field.
- A three-channel (e.g., RGB) image is also a vector field, but of higher-dimensional range than what we will consider here.

Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}$$

Think of this as
a 2D vector.

Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}$$

Think of this as a 2D vector.

Gradient (grad): product of nabla with a scalar field.

$$\nabla I(x, y) \ = ?$$

Divergence: inner product of nabla with a vector field.

$$\nabla \cdot [u(x, y) \quad v(x, y)] = ?$$

Curl: cross product of nabla with a vector field.

$$\nabla \times [u(x, y) \quad v(x, y)] = ?$$

Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}$$

Think of this as a 2D vector.

Gradient (grad): product of nabla with a scalar field.

$$\nabla I(x,y) \ = \begin{bmatrix} \frac{\partial I}{\partial x}(x,y) & \frac{\partial I}{\partial y}(x,y) \end{bmatrix}$$

What is the dimension of this?

Divergence: inner product of nabla with a vector field.

$$\nabla \cdot [u(x,y) \quad v(x,y)] = \frac{\partial u}{\partial x}(x,y) + \frac{\partial v}{\partial y}(x,y)$$

What is the dimension of this?

Curl: cross product of nabla with a vector field.

$$\nabla \times [u(x,y) \quad v(x,y)] = \left(\frac{\partial v}{\partial x}(x,y) - \frac{\partial u}{\partial y}(x,y) \right) \hat{k}$$

What is the dimension of this?

Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}$$

Think of this as a 2D vector.

Gradient (grad): product of nabla with a scalar field.

$$\nabla I(x, y) \ = \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) & \frac{\partial I}{\partial y}(x, y) \end{bmatrix}$$

This is a vector field.

Divergence: inner product of nabla with a vector field.

$$\nabla \cdot [u(x, y) \quad v(x, y)] = \frac{\partial u}{\partial x}(x, y) + \frac{\partial v}{\partial y}(x, y)$$

This is a scalar field.

Curl: cross product of nabla with a vector field.

$$\nabla \times [u(x, y) \quad v(x, y)] = \left(\frac{\partial v}{\partial x}(x, y) - \frac{\partial u}{\partial y}(x, y) \right) \hat{k}$$

This is a vector field.

Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

$$\nabla = \left[\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y}\right]$$

Think of this as a 2D vector.

Gradient (grad): product of nabla with a scalar field.

$$\nabla I(x,y) \ = \left[\frac{\partial I}{\partial x}(x,y) \quad \frac{\partial I}{\partial y}(x,y)\right]$$

This is a vector field.

Divergence: inner product of nabla with a vector field.

$$\nabla \cdot [u(x,y) \quad v(x,y)] = \frac{\partial u}{\partial x}(x,y) + \frac{\partial v}{\partial y}(x,y)$$

This is a scalar field.

Curl: cross product of nabla with a vector field.

$$\boxed{\nabla \times [u(x,y) \quad v(x,y)] = \left(\frac{\partial v}{\partial x}(x,y) - \frac{\partial u}{\partial y}(x,y)\right)} \hat{k}$$

This is a vector field.

This is a scalar field.

Combinations

Curl of the gradient:

$$\nabla \times \nabla I(x, y) = ?$$

Divergence of the gradient:

$$\nabla \cdot \nabla I(x, y) = ?$$

Combinations

Curl of the gradient:

$$\nabla \times \nabla I(x,y) = \frac{\partial^2}{\partial y \partial x} I(x,y) - \frac{\partial^2}{\partial x \partial y} I(x,y)$$

Divergence of the gradient:

$$\nabla \cdot \nabla I(x,y) = \frac{\partial^2}{\partial x^2} I(x,y) + \frac{\partial^2}{\partial y^2} I(x,y) \equiv \Delta I(x,y)$$

Laplacian: scalar differential operator.

$$\Delta \equiv \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Inner product of
del with itself!

Simplified notation

Nabla (or del): vector differential operator.

$$\nabla = [\quad x \qquad y]$$

Think of this as
a 2D vector.

Gradient (grad): product of nabla with a scalar field.

$$\nabla I \; = [I_x \quad I_y]$$

This is a
vector field.

Divergence: inner product of nabla with a vector field.

$$\nabla \cdot [u \quad v] = u_x + v_y$$

This is a
scalar field.

Curl: cross product of nabla with a vector field.

$$\nabla \times [u \quad v] = \left(v_x - u_y\right)\hat{k}$$

This is a vector field.

This is a scalar field.

Simplified notation

Curl of the gradient:

$$\nabla \times \nabla I = I_{yx} - I_{xy}$$

Divergence of the gradient:

$$\nabla \cdot \nabla I = I_{xx} + I_{yy} \equiv \Delta I$$

Laplacian: scalar differential operator.

$$\Delta \equiv \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Inner product of del with itself!

Image representation

We can treat grayscale images as scalar fields (i.e.. two dimensional functions)

$$I(x, y): \mathbb{R}^2 \rightarrow \mathbb{R}$$

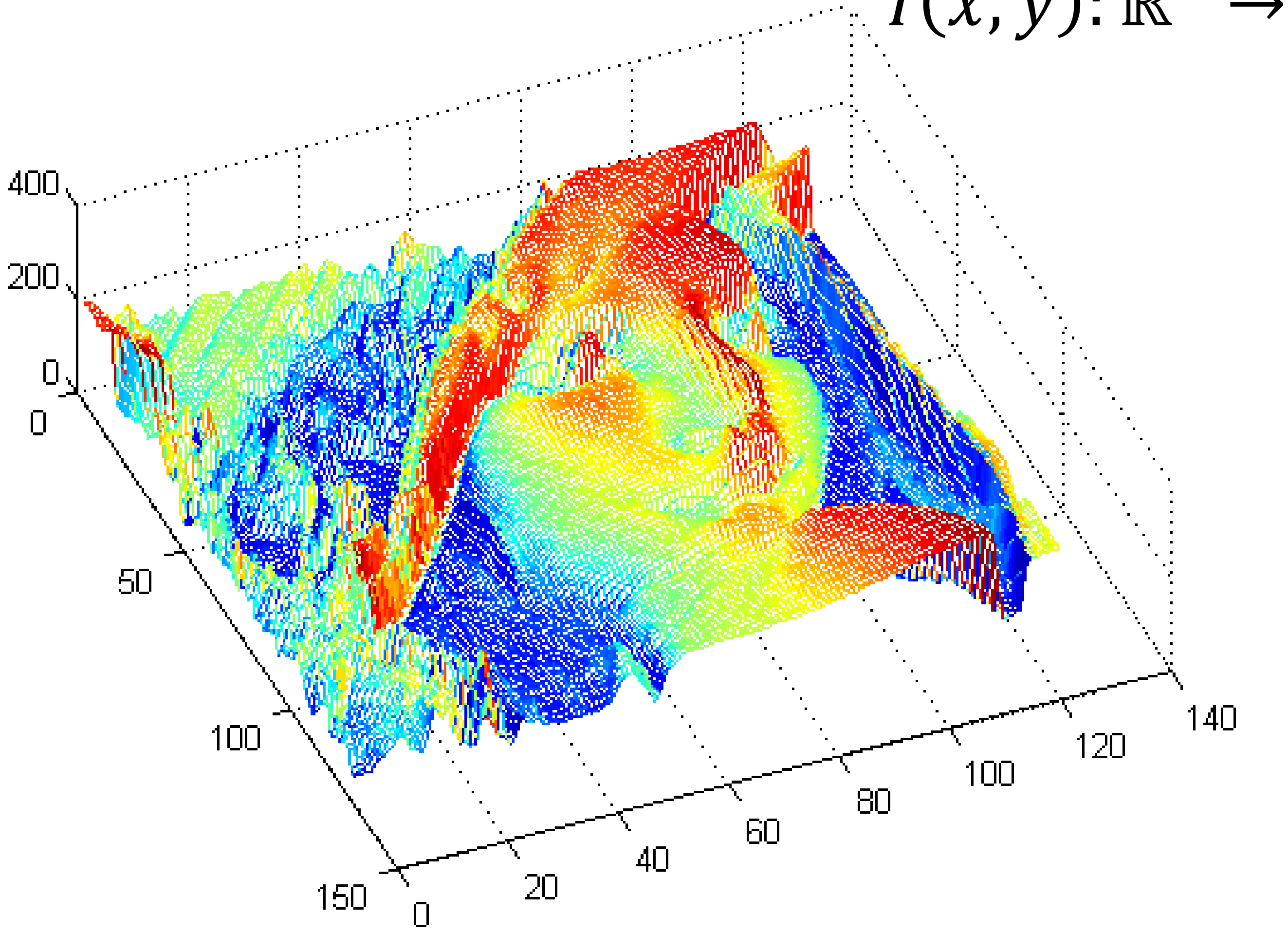
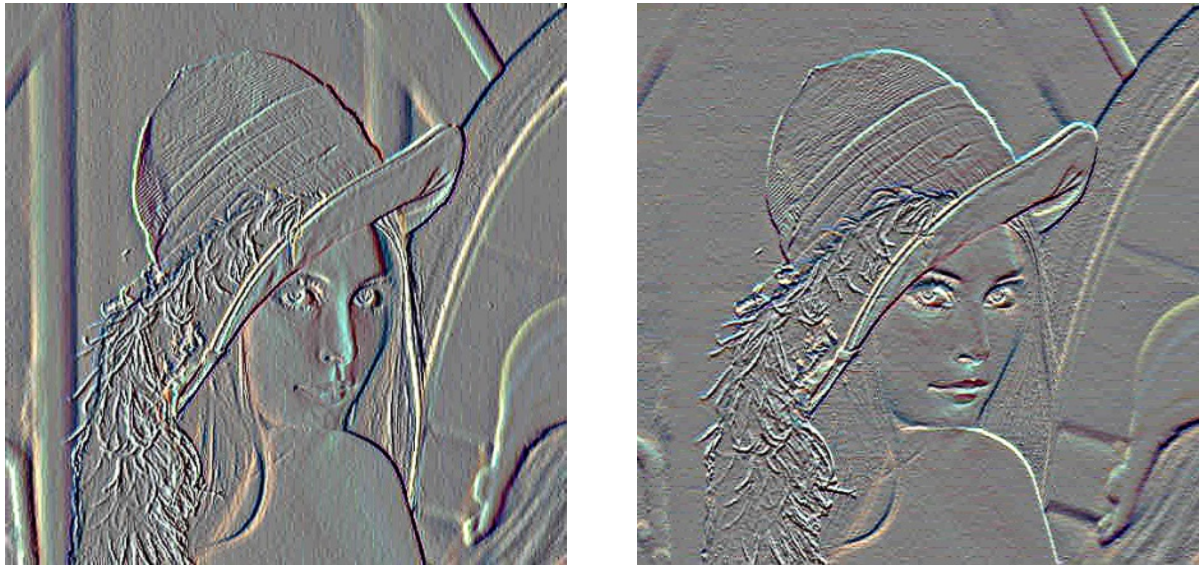


Image gradients

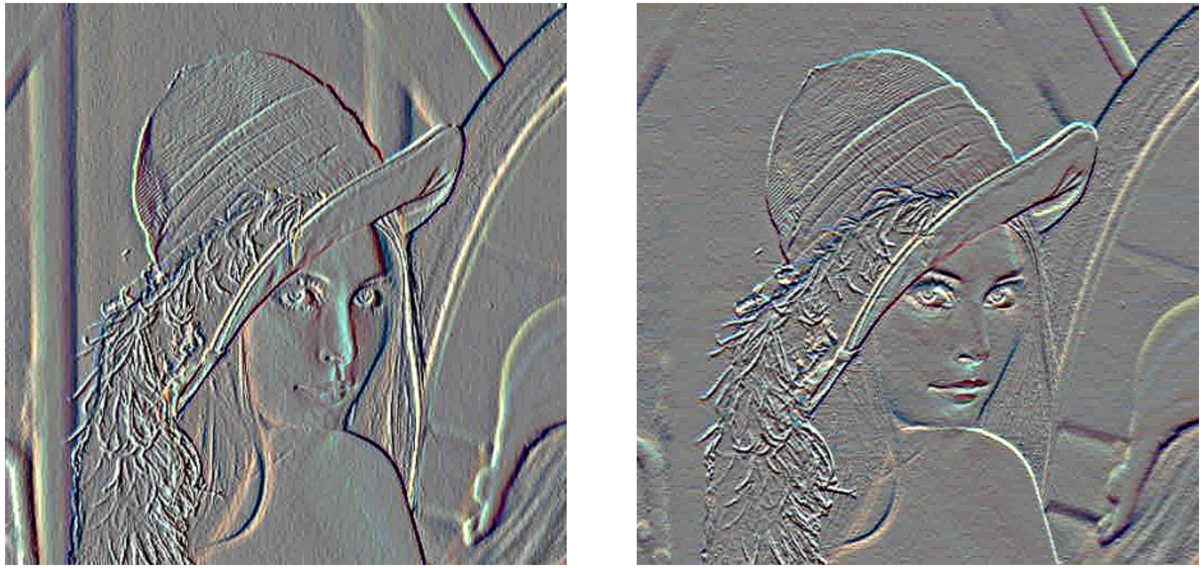
Convert the *scalar* field into a *vector* field through differentiation.



scalar field $I(x,y)\colon \mathbb{R}^2 \rightarrow \mathbb{R}$ ➡ vector field $\nabla I(x,y) = \left[\frac{\partial I}{\partial x}(x,y) \quad \frac{\partial I}{\partial y}(x,y)\right]$

Image gradients

Convert the *scalar* field into a *vector* field through differentiation.



scalar field $I(x,y): \mathbb{R}^2 \to \mathbb{R}$

vector field $\nabla I(x,y) = \left[\frac{\partial I}{\partial x}(x,y) \quad \frac{\partial I}{\partial y}(x,y)\right]$

- How do we do this differentiation in real *discrete* images?

Finite differences

High-school reminder: definition of a derivative using forward difference.

$$\frac{\partial I}{\partial x}(x,y) = \lim_{h\to 0} \frac{I(x+h,y) - I(x,y)}{h}$$

For discrete scalar fields: remove limit and set h = 1.

$$\frac{\partial I}{\partial x}(x,y) = I(x+1,y) - I(x,y)$$

What convolution kernel does this correspond to?

Finite differences

High-school reminder: definition of a derivative using forward difference.

$$\frac{\partial I}{\partial x}(x, y) = \lim_{h \to 0} \frac{I(x + h, y) - I(x, y)}{h}$$

For discrete scalar fields: remove limit and set h = 1.

$$\frac{\partial I}{\partial x}(x, y) = I(x + 1, y) - I(x, y)$$

-1	1

?

1	-1

?

Finite differences

High-school reminder: definition of a derivative using forward difference.

$$\frac{\partial I}{\partial x}(x,y) = \lim_{h\to 0} \frac{I(x+h,y) - I(x,y)}{h}$$

For discrete scalar fields: remove limit and set h = 1.

$$\frac{\partial I}{\partial x}(x,y) = I(x+1,y) - I(x,y)$$

partial-x derivative filter

1	-1

Note: common to use central difference, but we will *not* use it in this lecture.

$$\frac{\partial I}{\partial x}(x,y) = \frac{I(x+1,y) - I(x-1,y)}{2}$$

Finite differences

High-school reminder: definition of a derivative using forward difference.

$$\frac{\partial I}{\partial x}(x,y) = \lim_{h \to 0} \frac{I(x+h,y) - I(x,y)}{h}$$

For discrete scalar fields: remove limit and set h = 1.

$$\frac{\partial I}{\partial x}(x,y) = I(x+1,y) - I(x,y)$$

partial-x derivative filter

1	-1

Similarly for partial-y derivative.

partial-y derivative filter

$$\frac{\partial I}{\partial y}(x,y) = I(x,y+h) - I(x,y)$$

1
-1

Discrete Laplacian

How do we compute the image Laplacian?

$$\Delta I(x, y) = \frac{\partial^2 I}{\partial x^2}(x, y) + \frac{\partial^2 I}{\partial y^2}(x, y)$$

Discrete Laplacian

How do we compute the image Laplacian?

$$\Delta I(x, y) = \frac{\partial^2 I}{\partial x^2}(x, y) + \frac{\partial^2 I}{\partial y^2}(x, y)$$

Use multiple applications of the discrete derivative filters:

$$\underbrace{\begin{array}{|c|c|}\hline 1 & -1 \\ \hline\end{array} * \begin{array}{|c|c|}\hline 1 & -1 \\ \hline\end{array}}_{\text{What is this?}} + \underbrace{\begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline\end{array} * \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline\end{array}}_{\text{What is this?}} \quad = \qquad ?$$

Discrete Laplacian

How do we compute the image Laplacian?

$$\Delta I(x,y) = \frac{\partial^2 I}{\partial x^2}(x,y) + \frac{\partial^2 I}{\partial y^2}(x,y)$$

Use multiple applications of the discrete derivative filters:

Laplacian filter

$$\underbrace{\begin{array}{|c|c|}\hline 1 & -1 \\ \hline \end{array} * \begin{array}{|c|c|}\hline 1 & -1 \\ \hline \end{array}}_{\frac{\partial^2 I}{\partial x^2}(x,y)} + \underbrace{\begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array} * \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array}}_{\frac{\partial^2 I}{\partial y^2}(x,y)} = \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

Discrete Laplacian

Very important to:
- use consistent derivative and Laplacian filters.
- account for boundary shifting and padding from convolution.

How do we compute the image Laplacian?

$$\Delta I(x,y) = \frac{\partial^2 I}{\partial x^2}(x,y) + \frac{\partial^2 I}{\partial y^2}(x,y)$$

Use multiple applications of the discrete derivative filters:

Laplacian filter

$$\underbrace{\begin{array}{|c|c|}\hline 1 & -1 \\ \hline\end{array} * \begin{array}{|c|c|}\hline 1 & -1 \\ \hline\end{array}}_{\frac{\partial^2 I}{\partial x^2}(x,y)} + \underbrace{\begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline\end{array} * \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline\end{array}}_{\frac{\partial^2 I}{\partial y^2}(x,y)} = \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline\end{array}$$

Warning!

Very important for the techniques discussed in this lecture to:

- use consistent derivative and Laplacian filters.
- account for boundary shifting and padding from convolution.

A correct implementation of differential operators should pass the following test:

Equality holds at all pixels except boundary
(first and last row, first and last column).

$\nabla \cdot (\nabla($ $))$ $=$ $\Delta($ $)$

gradient operator

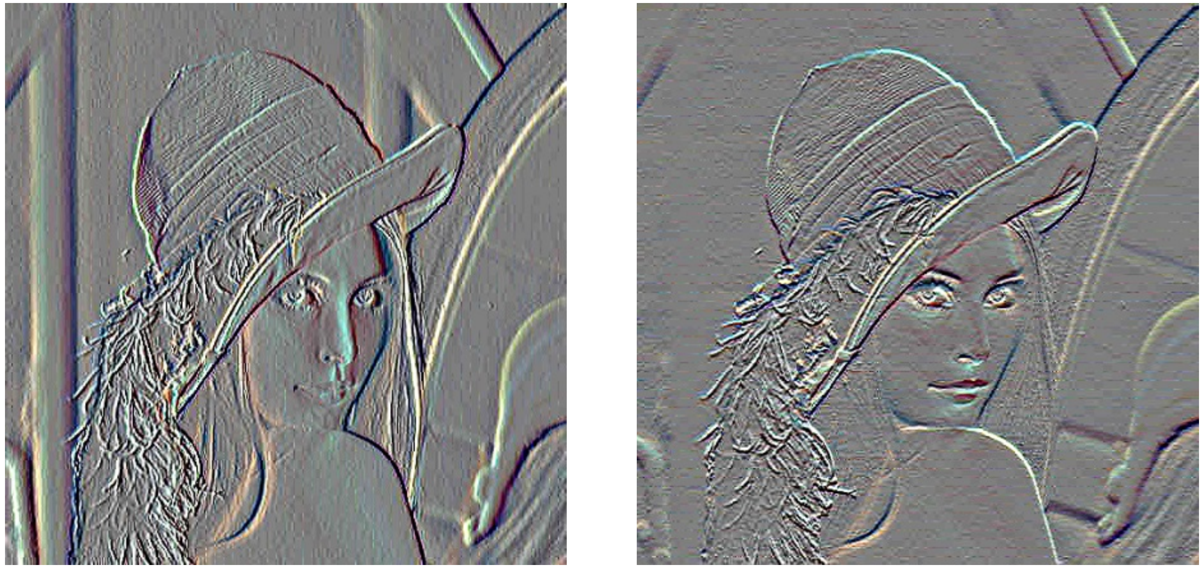
Laplacian operator

divergence operator

Typically requires implementing derivatives in various differential operators differently.

Image gradients

Convert the *scalar* field into a *vector* field through differentiation.



scalar field $I(x,y): \mathbb{R}^2 \to \mathbb{R}$ ➡ vector field $\nabla I(x,y) = \left[\frac{\partial I}{\partial x}(x,y) \quad \frac{\partial I}{\partial y}(x,y) \right]$

- How do we do this differentiation in real *discrete* images?
- Can we go in the opposite direction, from gradients to images?

Vector field integration

Two fundamental questions:

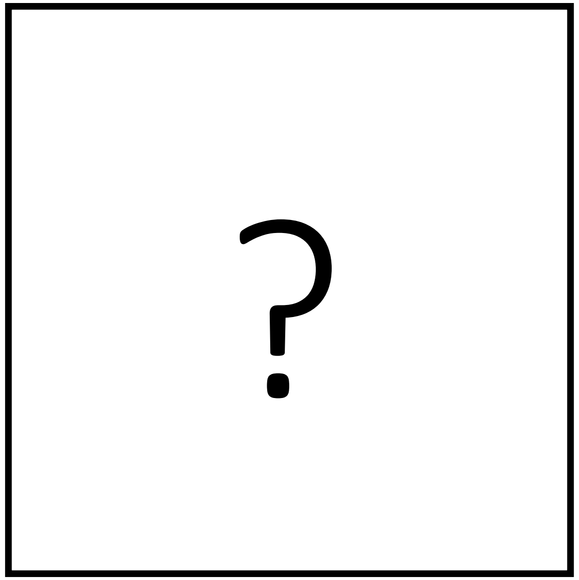
- When is integration of a vector field possible?

- How can integration of a vector field be performed?

Integrable vector fields

Integrable fields

Given an arbitrary vector field (u, v), can we always integrate it into a scalar field $\mathbf{I}$?



$I(x, y)\colon\ \mathbb{R}^2 \to \mathbb{R}$

$u(x, y)\colon\ \mathbb{R}^2 \to \mathbb{R}$

$v(x, y)\colon\ \mathbb{R}^2 \to \mathbb{R}$

such that

$$\frac{\partial I}{\partial x}(x, y) = u(x, y)$$

$$\frac{\partial I}{\partial y}(x, y) = v(x, y)$$

Property of twice-differentiable functions

Curl of the gradient field equals zero:

$$\nabla \times \nabla I = I_{yx} - I_{xy} = 0$$

What does that mean intuitively?

Property of twice-differentiable functions

Curl of the gradient field should be zero:

$$\nabla \times \nabla I = I_{yx} - I_{xy} = 0$$

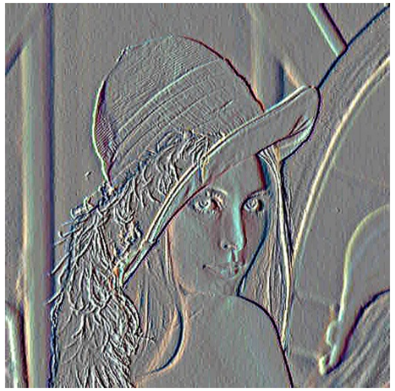
What does that mean intuitively?

- Same result independent of order of differentiation.

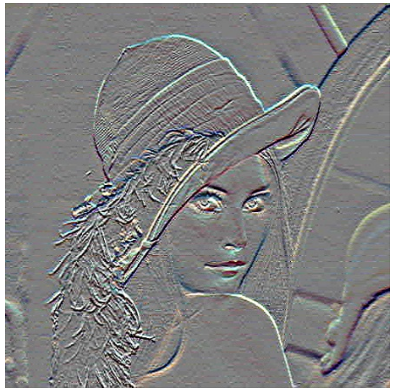
$$I_{yx} = I_{xy}$$

Demonstration

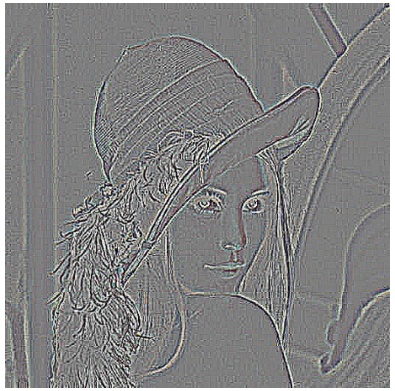
image I



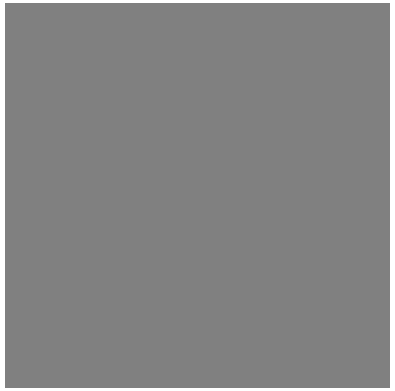
I_x



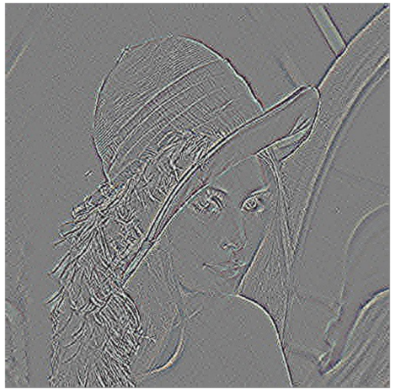
I_y



ΔI

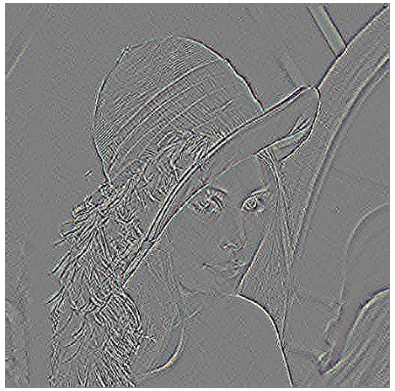


$\nabla \times \nabla I$



I_{xy}

=



I_{yx}

Property of twice-differentiable functions

Curl of the gradient field should be zero:

$$\nabla \times \nabla I = I_{yx} - I_{xy} = 0$$

What does that mean intuitively?

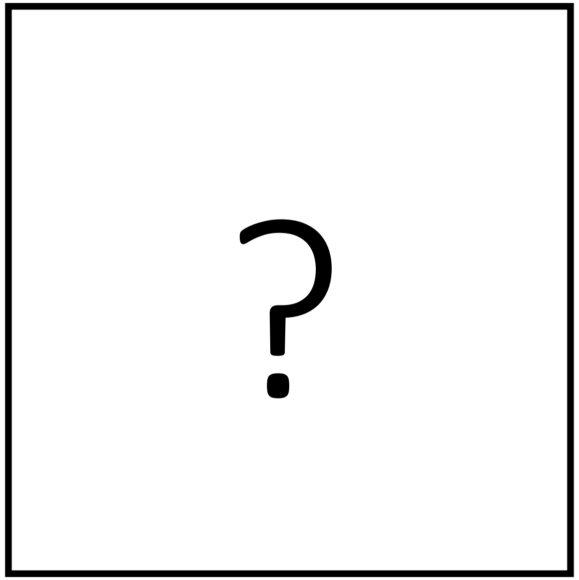
- Same result independent of order of differentiation.

$$I_{yx} = I_{xy}$$

Can you use this property to derive an integrability condition?

Integrable fields

Given an arbitrary vector field $(\mathbf{u}, \mathbf{v})$, can we always integrate it into a scalar field $\mathbf{I}$?



$I(x, y)\colon\ \mathbb{R}^2 \rightarrow \mathbb{R}$

$u(x, y)\colon\ \mathbb{R}^2 \rightarrow \mathbb{R}$

$v(x, y)\colon\ \mathbb{R}^2 \rightarrow \mathbb{R}$

such that
$$\frac{\partial I}{\partial x}(x, y) = u(x, y)$$
$$\frac{\partial I}{\partial y}(x, y) = v(x, y)$$

Only if:

$$\nabla \times \begin{bmatrix} u(x, y) \\ v(x, y) \end{bmatrix} = 0 \Rightarrow \frac{\partial u}{\partial y}(x, y) = \frac{\partial v}{\partial x}(x, y)$$

Vector field integration

Two fundamental questions:

- When is integration of a vector field possible?
  - Use curl to check for equality of mixed partial second derivatives.

- How can integration of a vector field be performed?

Different types of integration problems

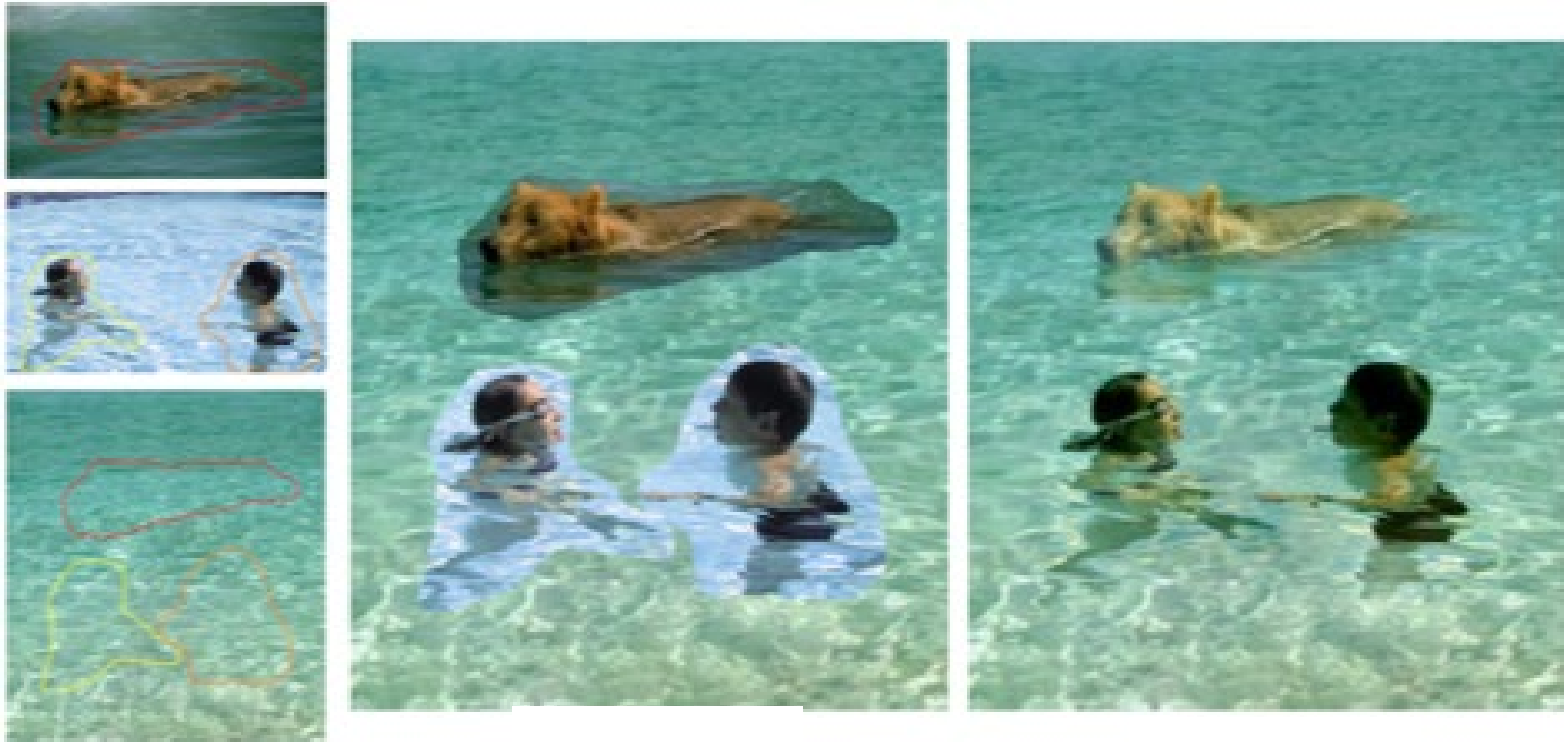
- Reconstructing height fields from gradients
  Applications: shape from shading, photometric stereo
- Manipulating image gradients
  Applications: tonemapping, image editing, matting, fusion, mosaics
- Manipulation of 3D gradients
  Applications: mesh editing, video operations

Key challenge: Most vector fields in applications are not integrable.

- Integration must be done *approximately*.

A prototypical integration problem: Poisson blending

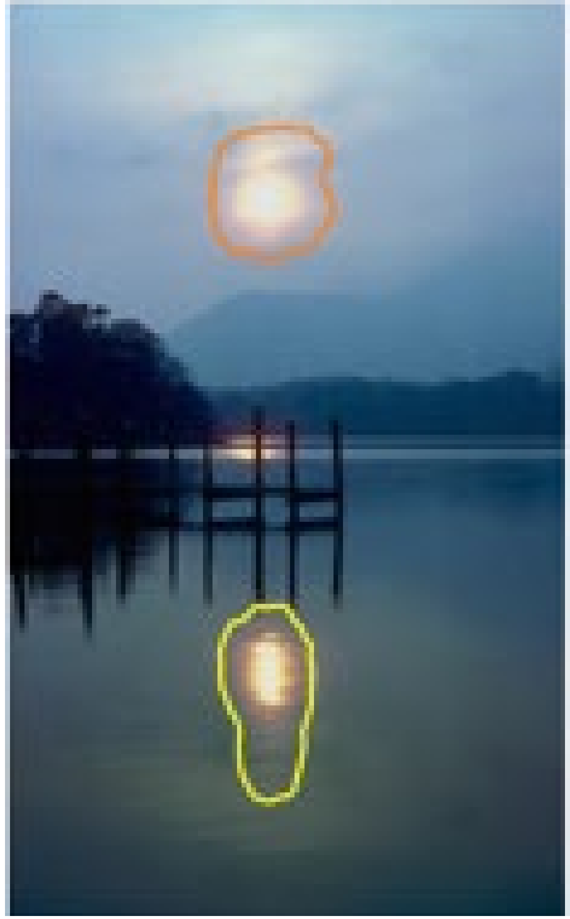
Application: Poisson blending



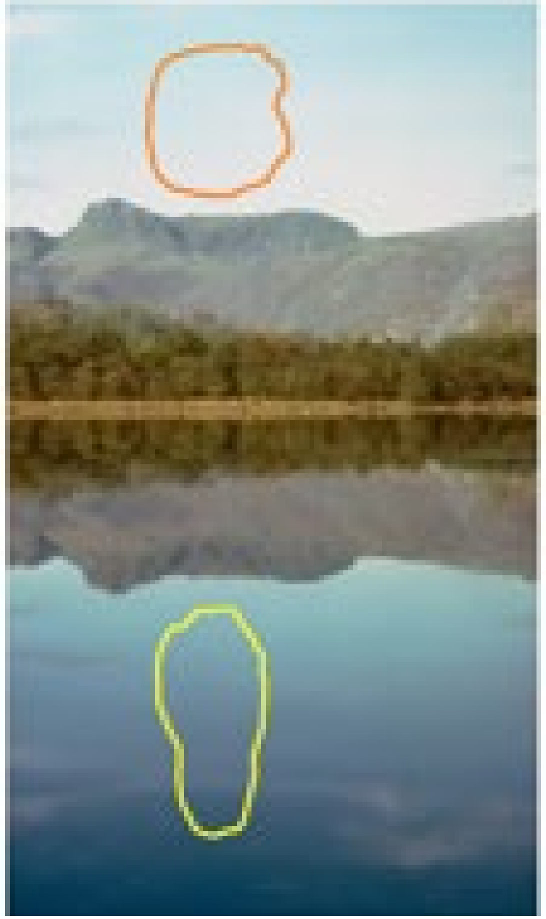
originals copy-paste Poisson blending

Key idea

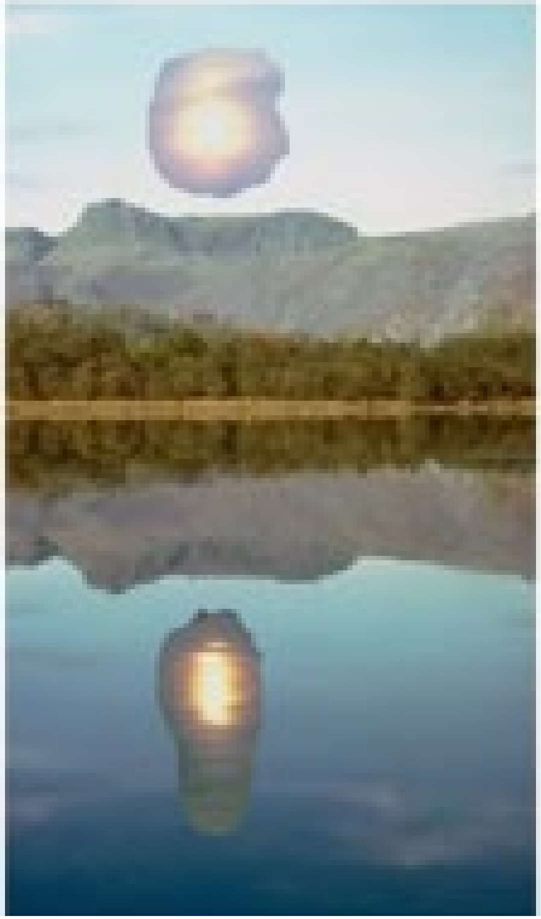
When blending, retain the gradient information as best as possible



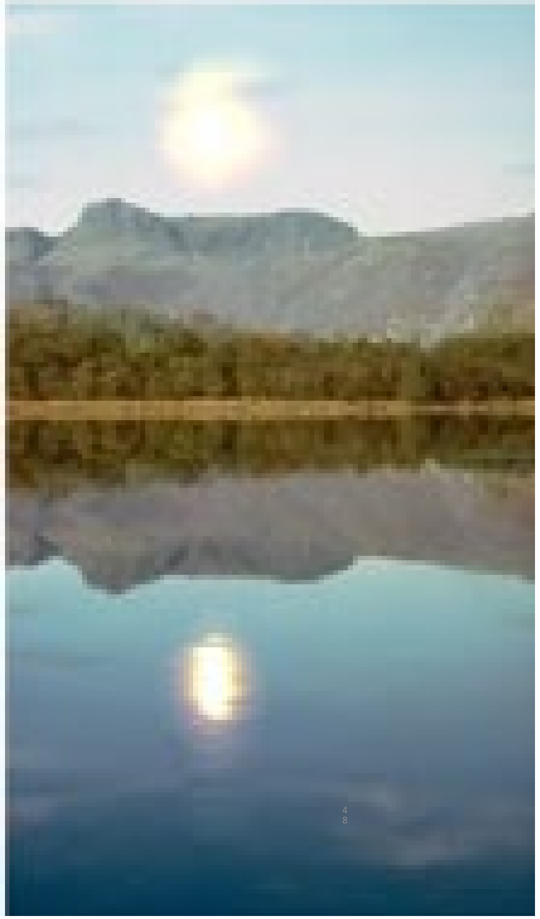
source



destination



copy-paste



Poisson blending

Definitions and notation

Notation

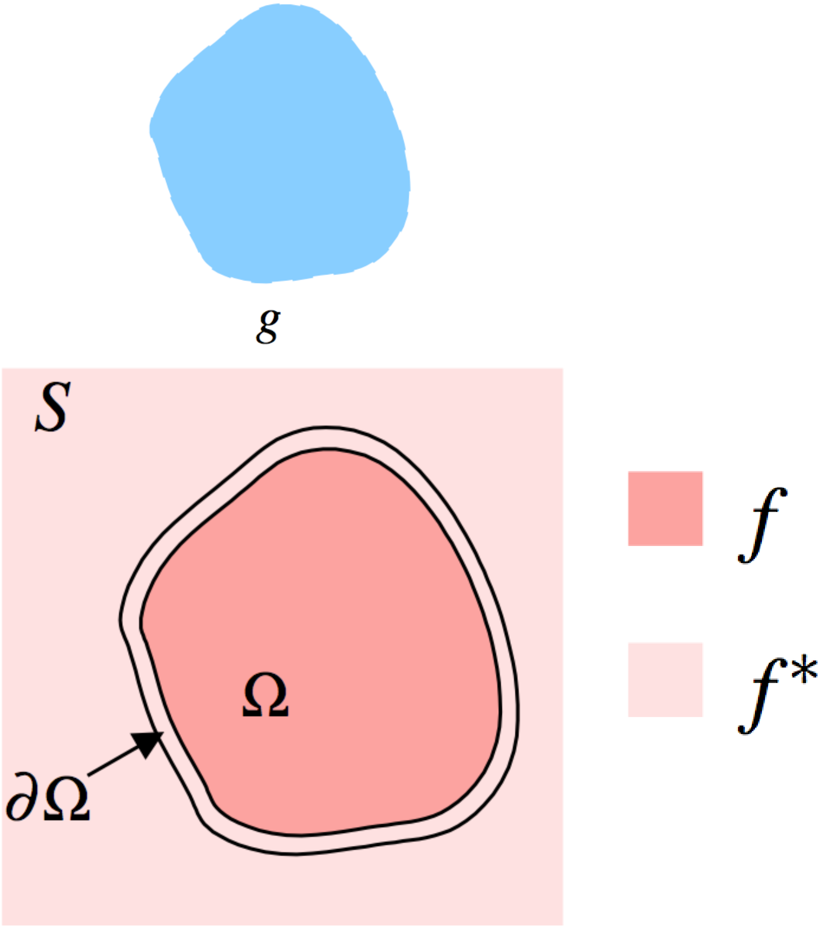
g: source function

S: destination

Ω: destination domain

f: interpolant function

f^*: destination function



Which one is the unknown?

Definitions and notation

Notation

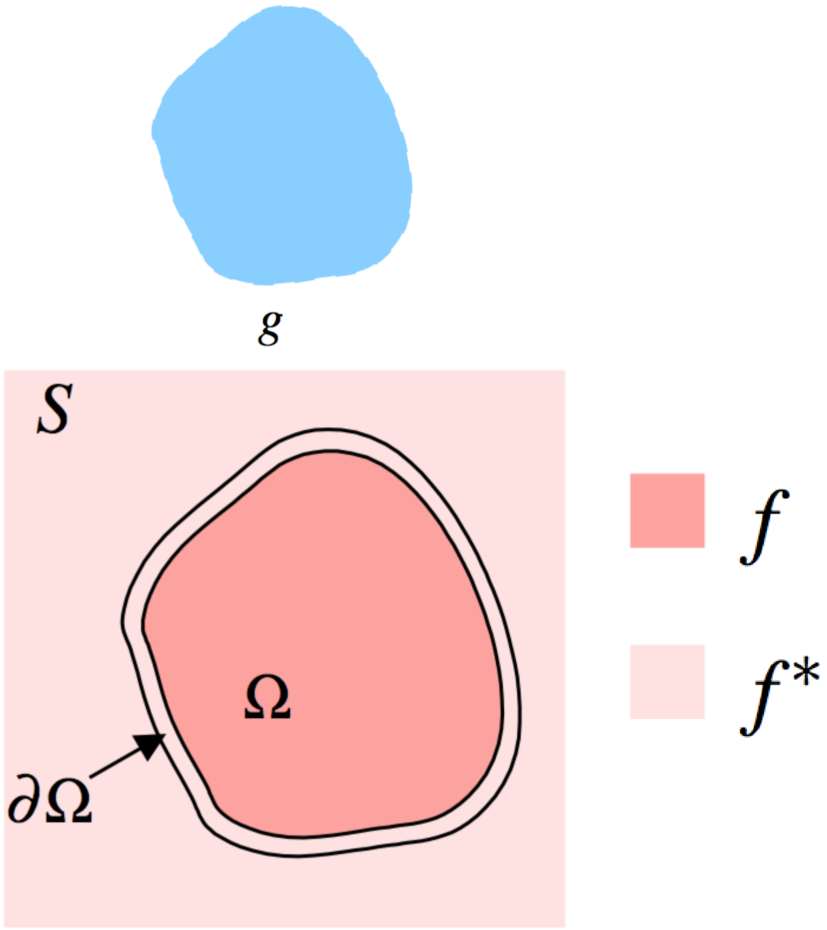
g: source function

S: destination

Ω: destination domain

$\boxed{f\text{: interpolant function}}$

f^*: destination function



How should we determine f?

- Should it be similar to g?
- Should it be similar to f^*?

Definitions and notation

Notation

g: source function

S: destination

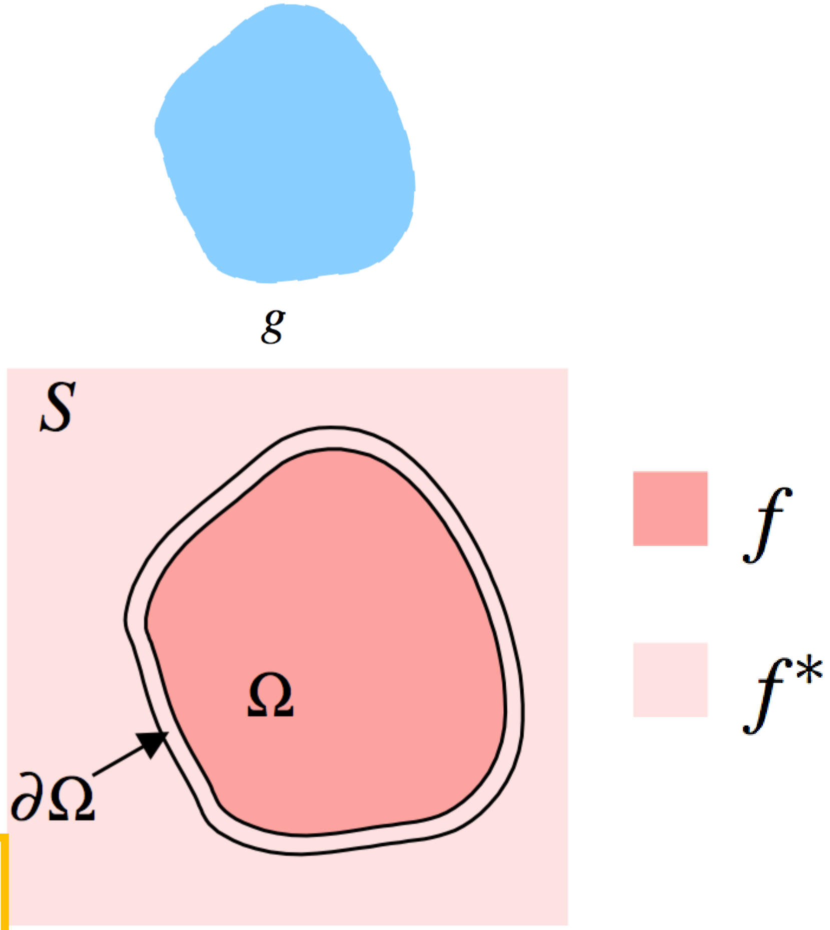
Ω: destination domain

f: interpolant function

f^*: destination function

Find f such that:

- $\nabla f = \nabla g$ inside Ω.
- $f = f^*$ at the boundary $\partial\Omega$.



Poisson blending: integrate vector field ∇g with Dirichlet boundary conditions f^*.

Least-squares integration and the Poisson problem

Least-squares integration

"Variational" means optimization where the unknown is an entire function

Variational problem

$$\min_{f} \iint_{\Omega} |\nabla f - \mathbf{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

what does this term do?

what does this term do?

Recall ...

Nabla operator definition

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

is this known?

$$\mathbf{v} = (u, v)$$

Least-squares integration

"Variational" means optimization where the unknown is an entire function

Variational problem

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

gradient of f looks like vector field **v**

f is equivalent to f* at the boundaries

Why do we need boundary conditions for least-squares integration?

Recall ...

Nabla operator definition

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Yes, this is the vector field we are integrating

$$\mathbf{v} = (u, v)$$

Equivalently

The *stationary point* of the variational loss is the solution to the:

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \mathrm{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

what does this term do?

This can be derived using the *Euler-Lagrange equation*.

Recall ...

Laplacian $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

Divergence $\mathrm{div}\ \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$

Input vector field:

$$\mathbf{v} = (u, v)$$

Equivalently

The *stationary point* of the variational loss is the solution to the:

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \operatorname{div} \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Laplacian of f same as divergence of vector field **v**

This can be derived using the *Euler-Lagrange equation*.

Recall ...

Laplacian $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

Divergence $\operatorname{div} \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$

Input vector field:

$$\mathbf{v} = (u, v)$$

In the Poisson blending example…

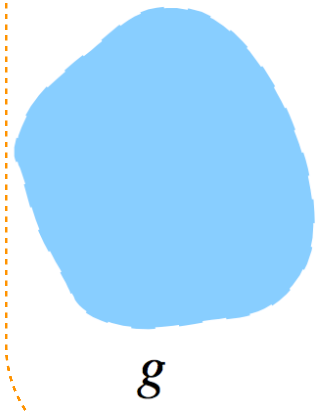
The *stationary point* of the variational loss is the solution to the:

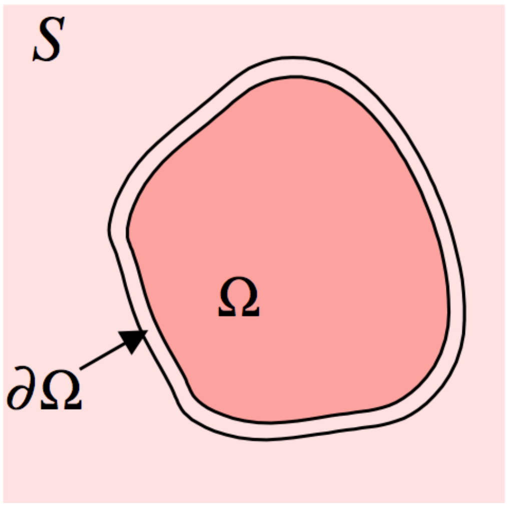
Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \mathrm{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Find f such that:

- $\nabla f = \nabla g$ inside Ω.
- $f = f^*$ at the boundary $\partial\Omega$.





What does the input vector field equal in Poisson blending?

$$\mathbf{v} = (u, v) =$$

In the Poisson blending example…

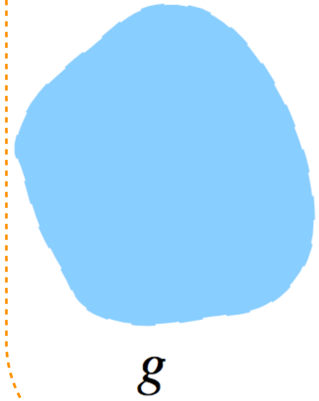
The *stationary point* of the variational loss is the solution to the:

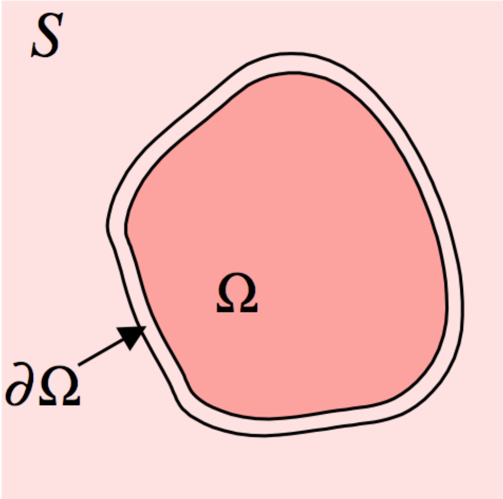
Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div } \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Find f such that:

- $\nabla f = \nabla g$ inside Ω.
- $f = f^*$ at the boundary $\partial\Omega$.





What does the input vector field equal in Poisson blending?

$$\mathbf{v} = (u, v) = \nabla g$$

What does the divergence of the input vector field equal in Poisson blending?

$$\text{div } \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} =$$

In the Poisson blending example…

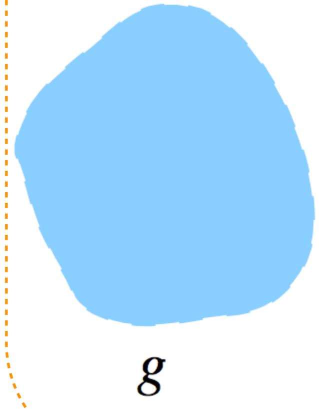
The *stationary point* of the variational loss is the solution to the:

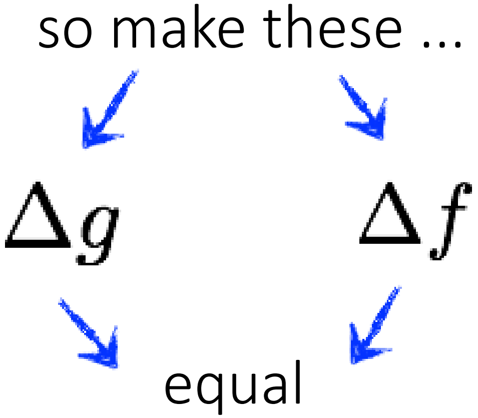
Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div } \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Find f such that:

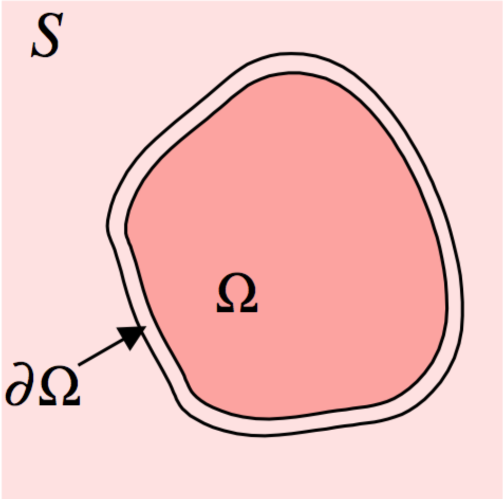
- $\nabla f = \nabla g$ inside Ω.
- $f = f^*$ at the boundary $\partial\Omega$.




so make these ...

Δg Δf

equal



What does the input vector field equal in Poisson blending?

$$\mathbf{v} = (u, v) = \nabla g$$

What does the divergence of the input vector field equal in Poisson blending?

$$\text{div } \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \Delta g$$

Equivalently

The *stationary point* of the variational loss is the solution to the:

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

How do we solve the Poisson equation?

Recall ...

Laplacian $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

Divergence $\text{div}\ \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$

Input vector field:

$$\mathbf{v} = (u, v)$$

Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Recall ...

Laplacian filter

0	1	0
1	-4	1
0	1	0

partial-x derivative filter

1	-1

partial-y derivative filter

1
-1

So for each pixel, do:

$$(\Delta f)(x, y) = (\nabla \cdot \mathbf{v})(x, y)$$

Or for discrete images:

Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \operatorname{div} \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Recall ...

Laplacian filter

0	1	0
1	-4	1
0	1	0

partial-x derivative filter

1	-1

partial-y derivative filter

1
-1

So for each pixel, do:

$$(\Delta f)(x, y) = (\nabla \cdot \mathbf{v})(x, y)$$

Or for discrete images:

$$\begin{aligned} -4f(x,y) + f(x+1,y) + f(x-1,y) \\ +f(x,y+1) + f(x,y-1) \\ = u(x+1,y) - u(x,y) + v(x,y+1) \\ - v(x,y) \end{aligned}$$

Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Recall ...

Laplacian filter

0	1	0
1	-4	1
0	1	0

partial-x derivative filter

1	-1

partial-y derivative filter

1
-1

So for each pixel, do (more compact notation):

$$(\Delta f)_p = (\nabla \cdot \mathbf{v})_p$$

Or for discrete images (more compact notation):

$$-4f_p + \sum_{q \in N_p} f_q = (u_x)_p + \left(v_y\right)_p$$

We can rewrite this as

linear equation
of P variables

$$-4f_p + \sum\nolimits_{q\in N_p} f_q = (u_x)_p + \left(v_y\right)_p$$

one for each
pixel p = 1, …, P

In vector form:

(each pixel adds another ‘sparse’ row here)

$$\underbrace{\begin{bmatrix} & & & & & \vdots & & & & & \\ 0 & \cdots & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \cdots & 0 \\ & & & & & \vdots & & & & & \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix}}_{f} = \underbrace{\begin{bmatrix} (\nabla\cdot\mathbf{v})_1 \\ \vdots \\ (\nabla\cdot\mathbf{v})_{q_1} \\ \vdots \\ (\nabla\cdot\mathbf{v})_{q_2} \\ (\nabla\cdot\mathbf{v})_p \\ (\nabla\cdot\mathbf{v})_{q_3} \\ \vdots \\ (\nabla\cdot\mathbf{v})_{q_4} \\ \vdots \\ (\nabla\cdot\mathbf{v})_P \end{bmatrix}}_{b}$$

We can rewrite this as

linear equation of P variables

$$-4f_p + \sum_{q \in N_p} f_q = (u_x)_p + \left(v_y\right)_p$$

one for each pixel p = 1, …, P

In vector form:

(each pixel adds another ‘sparse’ row here)

what is this? →

$$\underbrace{\begin{bmatrix} \vdots \\ \boxed{0 \quad \cdots \quad 1 \quad \cdots \quad 1 \quad -4 \quad 1 \quad \cdots \quad 1 \quad \cdots \quad 0} \\ \vdots \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix}}_{f} = \underbrace{\begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}}_{b}$$

what are the sizes of these?

We can rewrite this as

linear equation of P variables

$$-4f_p + \sum_{q \in N_p} f_q = (u_x)_p + \left(v_y\right)_p$$

one for each pixel p = 1, …, P

In vector form:

(each pixel adds another ‘sparse’ row here)

$$\underbrace{\begin{bmatrix} & & & & & \vdots & & & & & \\ 0 & \cdots & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \cdots & 0 \\ & & & & & \vdots & & & & & \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix}}_{f} = \underbrace{\begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}}_{b}$$

We call this the *Laplacian matrix*

Laplacian matrix

For a $m \times n$ image, we can re-organize this matrix into *block tridiagonal form* as:

$$A_{mn \times mn} = \begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & \boxed{I} & \boxed{D} \end{bmatrix}$$

This requires ordering pixels in column-major order.

$I_{m \times m}$ is the $m \times m$ identity matrix

$$D_{m \times m} = \begin{bmatrix} -4 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -4 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -4 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -4 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -4 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -4 \end{bmatrix}$$

Discrete Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

After discretization, equivalent to:

$$\begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix} = \begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}$$

⇨

Linear system of equations:

$$Af = b$$

How would you solve this?

WARNING: requires special treatment at the borders (target boundary values are same as source)

Solving the linear system

Convert the system to a linear least-squares problem:

$$E_{\text{LLS}} = \|\mathbf{A}\,\boldsymbol{f} - \boldsymbol{b}\|^2$$

Expand the error:

$$E_{\text{LLS}} = \boldsymbol{f}^\top(\mathbf{A}^\top\mathbf{A})\boldsymbol{f} - 2\boldsymbol{f}^\top(\mathbf{A}^\top\boldsymbol{b}) + \|\boldsymbol{b}\|^2$$

Minimize the error:

Set derivative to 0 $(\mathbf{A}^\top\mathbf{A})\boldsymbol{f} = \mathbf{A}^\top\boldsymbol{b}$

Solve for x $\boldsymbol{f} = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\boldsymbol{b}$ ⟵

Note: You almost never want to compute the inverse of a matrix.

In Matlab:

```
f = A \ b
```

Discrete the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \operatorname{div} \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

After discretization, equivalent to:

$$\begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix} = \begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}$$

Linear system of equations:

$$Af = b$$

What is the size of this matrix?

WARNING: requires special treatment at the borders (target boundary values are same as source)

Discrete Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

After discretization, equivalent to:

$$\begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix} = \begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}$$

Linear system of equations:

$$Af = b$$

Matrix is $P \times P$ → billions of entries

WARNING: requires special treatment at the borders (target boundary values are same as source)

Integration procedures

- Poisson solver (i.e., least squares integration)
  + Generally applicable.
  - Matrices A can become very large.

- Acceleration techniques:
  + (Conjugate) gradient descent solvers.
  + Multi-grid approaches.
  + Pre-conditioning.
  …

- Alternative solvers: projection procedures.
  We will discuss one of these when we cover photometric stereo.

A more efficient Poisson solver

Let’s look again at our optimization problem

Variational problem

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

gradient of f looks like vector field **v**

f is equivalent to f* at the boundaries

Input vector field:

$$\mathbf{v} = (u, v)$$

Recall ...

Nabla operator definition

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Let’s look again at our optimization problem

Variational problem

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

gradient of f looks like vector field **v**

f is equivalent to f* at the boundaries

Input vector field:

$$\mathbf{v} = (u, v)$$

Recall ...

Nabla operator definition

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

And for discrete images:

partial-x derivative filter

1	-1

partial-y derivative filter

1
-1

Let’s look again at our optimization problem

We can use the gradient approximation to discretize the variational problem

Discrete problem

What are G, f, and v?

$$\min_f \|Gf - v\|^2$$

We will ignore the boundary conditions for now.

Recall ...

Nabla operator definition

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

And for discrete images:

partial-x derivative filter

1	-1

partial-y derivative filter

1
-1

Let’s look again at our optimization problem

We can use the gradient approximation to discretize the variational problem

Discrete problem

matrix G formed by stacking together discrete gradients

$$\min_{f} \|Gf - v\|^2$$

vectorized version of the unknown image

vectorized version of the target gradient field

We will ignore the boundary conditions for now.

Recall ...

Image gradient

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

And for discrete images:

partial-x derivative filter

1	-1

partial-y derivative filter

1
-1

Let’s look again at our optimization problem

We can use the gradient approximation to discretize the variational problem

Discrete problem

matrix G formed by stacking together discrete gradients

$$\min_{f} \|Gf - v\|^2$$

vectorized version of the unknown image

vectorized version of the target gradient field

How do we solve this optimization problem?

Recall ...

Image gradient

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

And for discrete images:

partial-x derivative filter

$$\begin{array}{|c|c|}\hline 1 & -1 \\ \hline\end{array}$$

partial-y derivative filter

$$\begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline\end{array}$$

Approach 1: Compute stationary points

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} =?$$

Approach 1: Compute stationary points

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} = G^T G f - G^T v$$

… and we do what with it?

Approach 1: Compute stationary points

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} = G^T G f - G^T v$$

… and we set that to zero:

$$\frac{\partial E}{\partial f} = 0 \Rightarrow \underbrace{G^T G}_{\text{What is this matrix?}} f = \overbrace{G^T v}^{\text{What is this vector?}}$$

Approach 1: Compute stationary points

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} = G^T G f - G^T v$$

… and we set that to zero:

$$\frac{\partial E}{\partial f} = 0 \Rightarrow \underbrace{G^T G}_{\text{It is equal to the Laplacian matrix A we derived previously!}} f = \overbrace{G^T v}^{\text{It is equal to the vector b we derived previously!}}$$

Reminder from variational case

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div } \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

After discretization, equivalent to:

$$\begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix} = \begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}$$

Linear system of equations:

$$Af = b$$

Same system as:

$$G^T G f = G^T v$$

We arrive at the same system, no matter whether we discretize the continuous Poisson equation or the variational optimization problem.

Approach 1: Compute stationary points

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} = G^T G f - G^T v$$

… and we set that to zero:

$$\frac{\partial E}{\partial f} = 0 \Rightarrow G^T G f = G^T v$$

Solving this is exactly as expensive as what we had before.

Approach 2: Use gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} = G^T G f - G^T v = Af - b \equiv -r$$

We call this term the *residual*

Approach 2: Use gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

… we compute its derivative:

$$\frac{\partial E}{\partial f} = G^T G f - G^T v = Af - b \equiv -r$$

We call this term the *residual*

… and then we *iteratively* compute a solution:

$$f^{i+1} = f^i + \eta^i r^i$$

for $\mathrm{i} = 0, 1, \ldots, \mathrm{N}$, where η^i are positive *step sizes*

Selecting optimal step sizes

Make derivative of loss function *with respect to* η^i equal to zero:

$$E(f) = \|Gf - v\|^2$$

$$E\left(f^{i+1}\right) = \left\|G\left(f^i + \eta^i r^i\right) - v\right\|^2$$

Selecting optimal step sizes

Make derivative of loss function *with respect to* η^{i} equal to zero:

$$E(f) = \|Gf - v\|^2$$

$$E\left(f^{i+1}\right) = \left\|G\left(f^{i} + \eta^{i} r^{i}\right) - v\right\|^2$$

$$\frac{\partial E\left(f^{i+1}\right)}{\partial \eta^{i}} = \left[b - A\left(f^{i} + \eta^{i} r^{i}\right)\right]^T r^{i} = 0 \Rightarrow \eta^{i} = \frac{\left(r^{i}\right)^T r^{i}}{(r^{i})^T A r^{i}}$$

Gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

Minimize by iteratively computing:

$$r^i = b - Af^i, \quad \eta^i = \frac{\left(r^i\right)^T r^i}{\left(r^i\right)^T A r^i}, \quad f^{i+1} = f^i + \eta^i r^i, \quad i = 0, \ldots, N$$

Is this cheaper than the pseudo-inverse approach?

Gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

Minimize by iteratively computing:

$$r^i = b - \boxed{Af^i}, \quad \eta^i = \frac{\left(r^i\right)^T r^i}{\left(r^i\right)^T \boxed{Ar^i}}, \quad f^{i+1} = f^i + \eta^i r^i, \quad i = 0, \ldots, N$$

Is this cheaper than the pseudo-inverse approach?

- We never need to compute A, only its products with vectors f, r.

Gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

Minimize by iteratively computing:

$$r^i = b - \boxed{Af^i}, \quad \eta^i = \frac{\left(r^i\right)^T r^i}{\left(r^i\right)^T \boxed{Ar^i}}, \quad f^{i+1} = f^i + \eta^i r^i, \quad i = 0, \ldots, N$$

Is this cheaper than the pseudo-inverse approach?

- We never need to compute A, only its products with vectors f, r.
- Vectors f, r are images.

Gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

Minimize by iteratively computing:

$$r^i = b - \boxed{Af^i}, \quad \eta^i = \frac{\left(r^i\right)^T r^i}{\left(r^i\right)^T \boxed{Ar^i}}, \quad f^{i+1} = f^i + \eta^i r^i, \quad i = 0, \ldots, N$$

Is this cheaper than the pseudo-inverse approach?

- We never need to compute A, only its products with vectors f, r.
- Vectors f, r are images.
- Because A is the *Laplacian matrix*, these matrix-vector products can be efficiently computed using *convolutions* with the *Laplacian kernel*.

In practice: conjugate gradient descent

Given the loss function:

$$E(f) = \|Gf - v\|^2$$

Minimize by iteratively computing:

$$d^i = r^i + \beta^i d^i, \quad \eta^i = \frac{\left(r^i\right)^T r^i}{\left(d^i\right)^T \boxed{Ad^i}}, \quad f^{i+1} = f^i + \eta^i d^i, \quad i = 0, \dots, N$$

$$r^{i+1} = r^i - \eta^i \boxed{Ad^i}, \quad \beta^i = \frac{\left(r^{i+1}\right)^T r^{i+1}}{\left(r^i\right)^T r^i}$$

- Smarter way for selecting update directions
- Everything can still be done using convolutions
- Only one convolution needed per iteration

Note: initialization

Does the initialization $\mathbf{f}^0$ matter?

Note: initialization

Does the initialization $\mathbf{f}^0$ matter?

- It doesn't matter in terms of what final $\mathbf{f}$ we converge to, because the loss function is convex.

$$E(f) = \|Gf - v\|^2$$

Note: initialization

Does the initialization $\mathbf{f}^0$ matter?

- It doesn't matter in terms of what final $\mathbf{f}$ we converge to, because the loss function is convex.

$$E(f) = \|Gf - v\|^2$$

- It does matter in terms of convergence speed.
- We can use a *multi-resolution* approach:

  - Solve an initial problem for a very low-resolution f (e.g., 2x2).
  - Use the solution to initialize gradient descent for a higher resolution $\mathbf{f}$ (e.g., 4x4).
  - Use the solution to initialize gradient descent for a higher resolution $\mathbf{f}$ (e.g., 8x8).

  …

  - Use the solution to initialize gradient descent for an $\mathbf{f}$ with the original resolution PxP.
- *Multi-grid* algorithms alternative between higher and lower resolutions during the (conjugate) gradient descent iterative procedure.

Reminder from variational case

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div }\mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

After discretization, equivalent to:

$$\begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix} = \begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}$$

Linear system of equations:

$$Af = b$$

Remember that what we are doing is equivalent to solving this linear system.

Note: preconditioning

We are solving this linear system:

$$Af = b$$

For any invertible matrix P, this is equivalent to solving:

$$P^{-1}Af = P^{-1}b$$

When is it preferable to solve this alternative linear system?

Note: preconditioning

We are solving this linear system:

$$Af = b$$

For any invertible matrix P, this is equivalent to solving:

$$P^{-1}Af = P^{-1}b$$

When is it preferable to solve this alternative linear system?

- Ideally: If A is invertible, and P is the same as A, the linear system becomes trivial! But computing the inverse of A is even more expensive than solving the original linear system.
- In practice: If the matrix $P^{-1}A$ has a better condition number, or its singular values are more uniformly distributed, the linear system becomes more *numerically stable*.

What *preconditioner* P should we use?

Note: preconditioning

We are solving this linear system:

$$Af = b$$

For any invertible matrix P, this is equivalent to solving:

$$P^{-1}Af = P^{-1}b$$

When is it preferable to solve this alternative linear system?

- Ideally: If A is invertible, and P is the same as A, the linear system becomes trivial! But computing the inverse of A is even more expensive than solving the original linear system.
- In practice: If the matrix $P^{-1}A$ has a better condition number, or its singular values are more uniformly distributed, the linear system becomes more *numerically stable*.

What *preconditioner* P should we use?

- Standard preconditioners like Jacobi.
- More effective preconditioners. Active area of research.

$$P_{\text{Jacobi}} = \text{diag}(A)$$

Note: preconditioning

We are solving this linear system:

$$Af = b$$

For any invertible matrix **P**, this is equivalent to solving:

$$P^{-1}Af = P^{-1}b$$

Preconditioning can be incorporated in the conjugate gradient descent algorithm.

When is it preferable to solve this alternative linear system?

- Ideally: If **A** is invertible, and **P** is the same as **A**, the linear system becomes trivial! But computing the inverse of **A** is even more expensive than solving the original linear system.
- In practice: If the matrix **P**$^{-1}$**A** has a better condition number, or its singular values are more uniformly distributed, the linear system becomes more *numerically stable*.

What *preconditioner* **P** should we use?

- Standard preconditioners like Jacobi.
- More effective preconditioners. Active area of research.

Is this effective for Poisson solvers?

$$\boxed{P_{\text{Jacobi}} = \operatorname{diag}(A)}$$

Discrete Poisson equation

Poisson equation (with Dirichlet boundary conditions)

$$\Delta f = \text{div }\mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

After discretization, equivalent to:

$$\begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_P \end{bmatrix} = \begin{bmatrix} (\nabla \cdot \mathbf{v})_1 \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_2} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{q_3} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{q_4} \\ \vdots \\ (\nabla \cdot \mathbf{v})_P \end{bmatrix}$$

Linear system of equations:

$$Af = b$$

Matrix is $P \times P$ → billions of entries

WARNING: requires special treatment at the borders
(target boundary values are same as source)

Note: handling (Dirichlet) boundary conditions

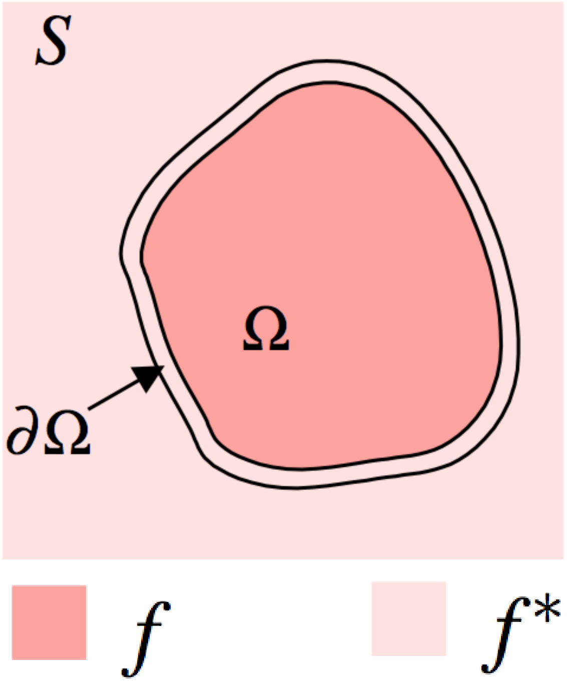
- Form a mask B that is 0 for pixels that should *not* be updated (pixels on S-Ω and $\partial\Omega$) and 1 otherwise.
- Use convolution to perform Laplacian filtering over the *entire image*.
- Use (conjugate) gradient descent rules to only update pixels for which the mask is 1. Equivalently, change the update rules to:

$$f^{i+1} = f^i + B\eta^i r^i$$

(gradient descent)

$$f^{i+1} = f^i + B\eta^i d^i$$

(conjugate gradient descent)



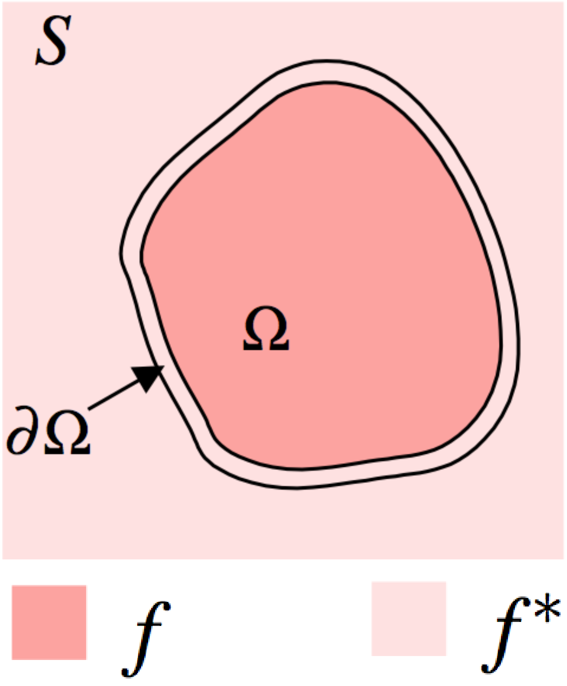
Note: handling (Dirichlet) boundary conditions

In practice, masking is also required at other steps of (conjugate) gradient descent, to deal with invalid boundaries (e.g., from convolutions). See homework assignment 3.

- Form a mask B that is 0 for pixels that should *not* be updated (pixels on S-Ω and $\partial\Omega$) and 1 otherwise.
- Use convolution to perform Laplacian filtering over the *entire image*.
- Use (conjugate) gradient descent rules to only update pixels for which the mask is 1. Equivalently, change the update rules to:

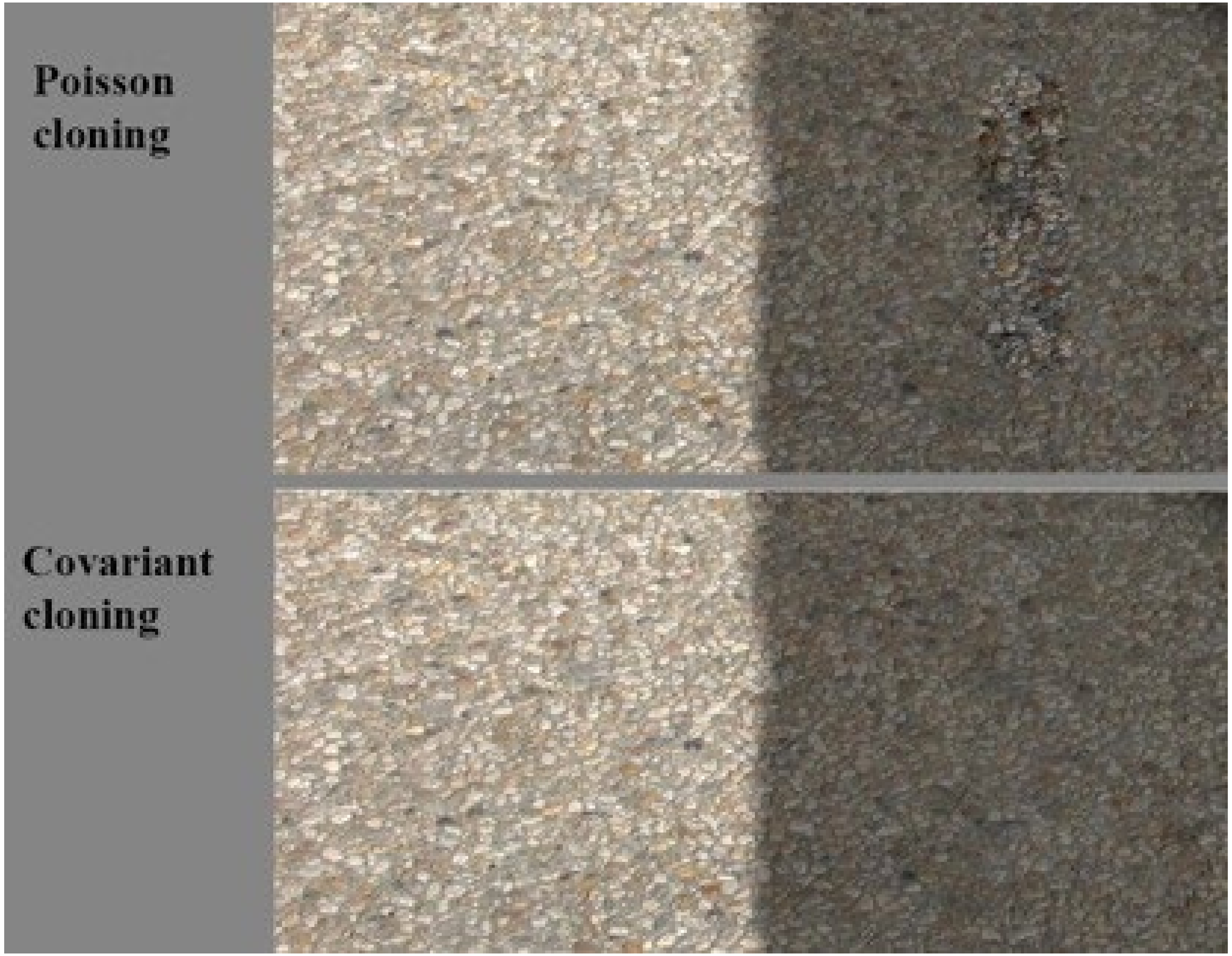
$$f^{i+1} = f^i + B\eta^i r^i \qquad \text{(gradient descent)}$$

$$f^{i+1} = f^i + B\eta^i d^i \qquad \text{(conjugate gradient descent)}$$



Poisson image editing examples

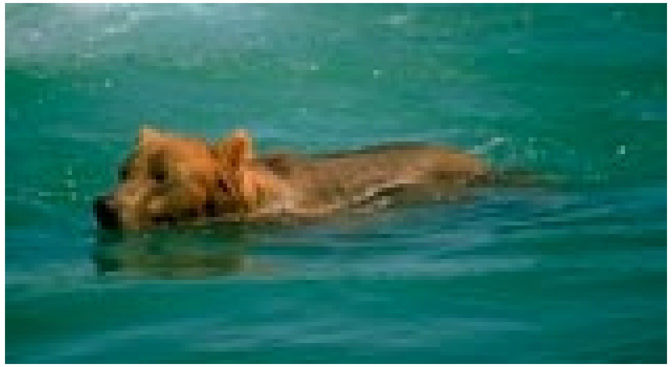
Photoshop’s “healing brush”



Slightly more advanced version of what we covered here:

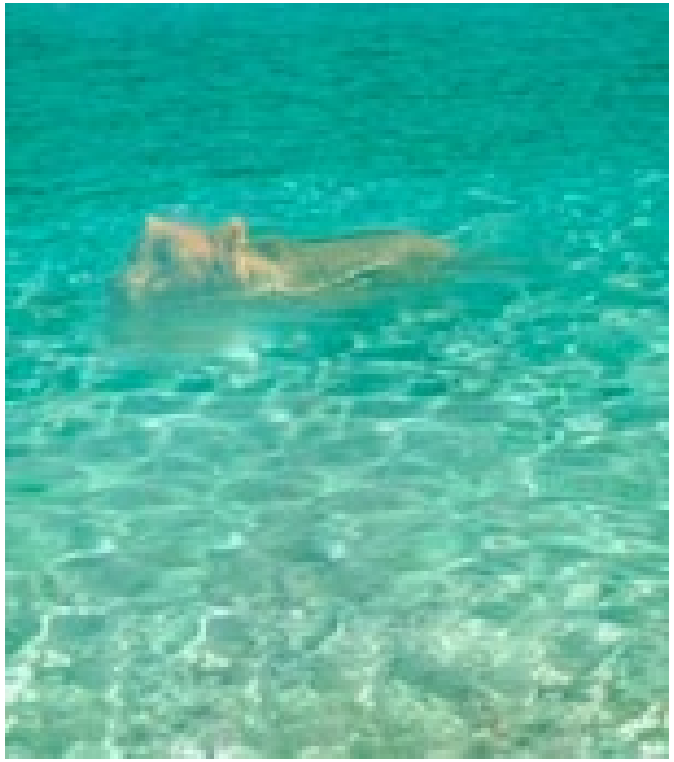
- Uses higher-order derivatives

Contrast problem

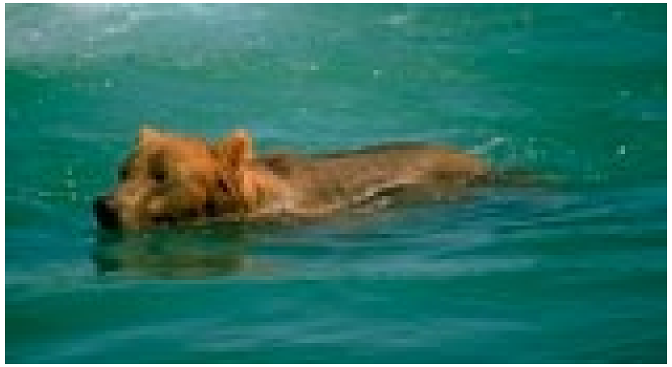


Loss of contrast when pasting from dark to bright:

- Contrast is a multiplicative property.
- With Poisson blending we are matching linear differences.



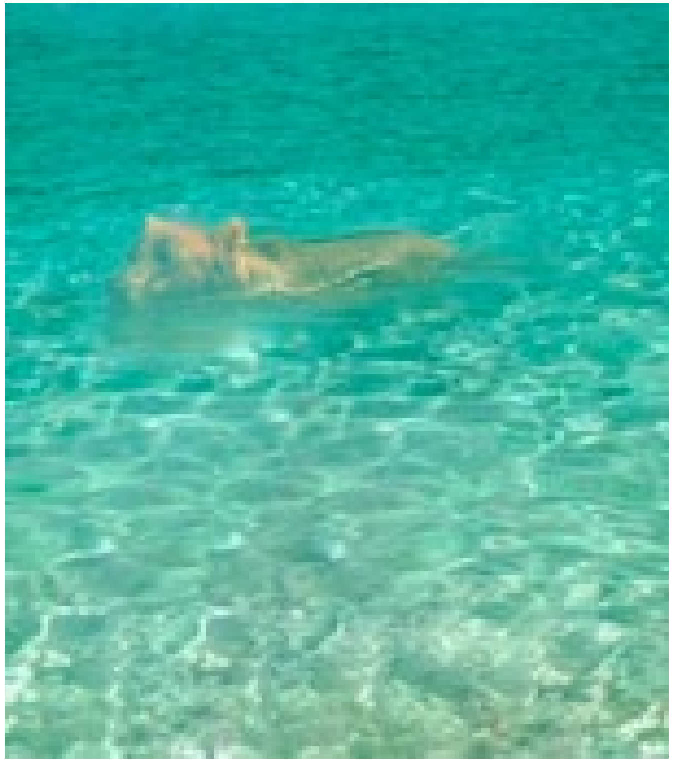
Contrast problem

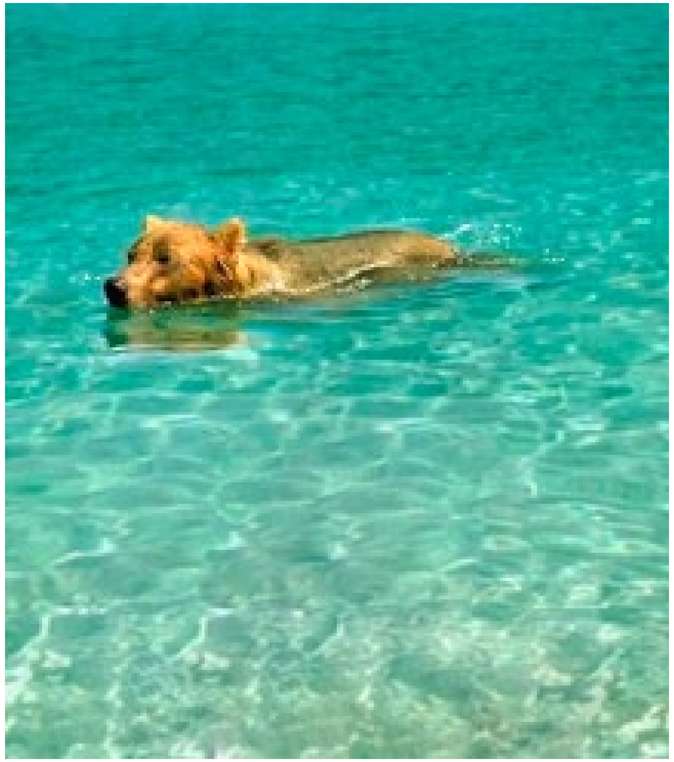


Loss of contrast when pasting from dark to bright:

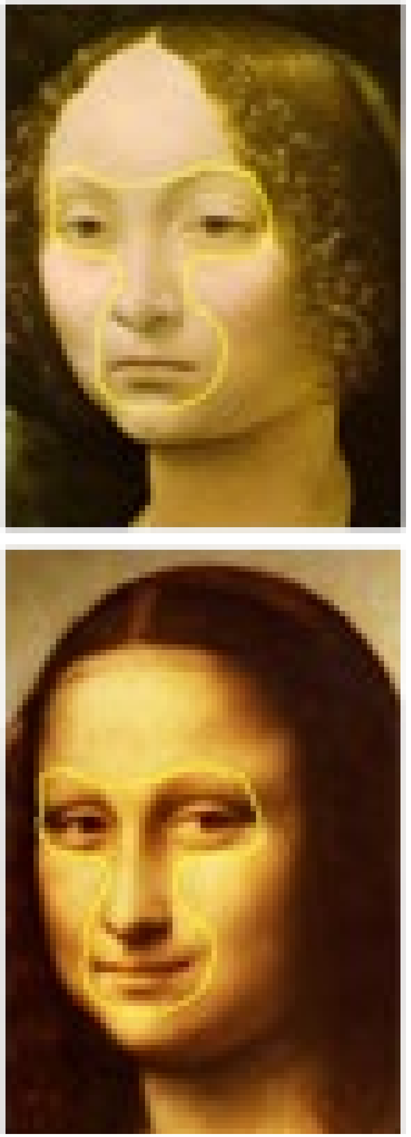
- Contrast is a multiplicative property.
- With Poisson blending we are matching linear differences.

Solution: Do blending in log-domain.

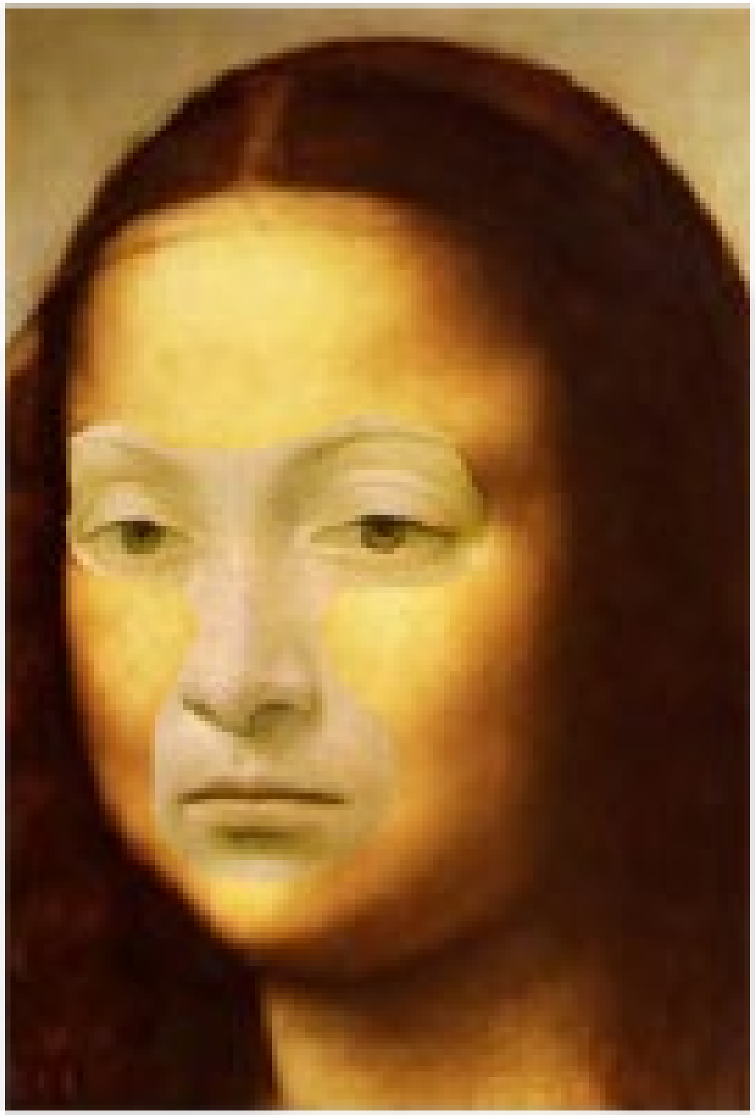




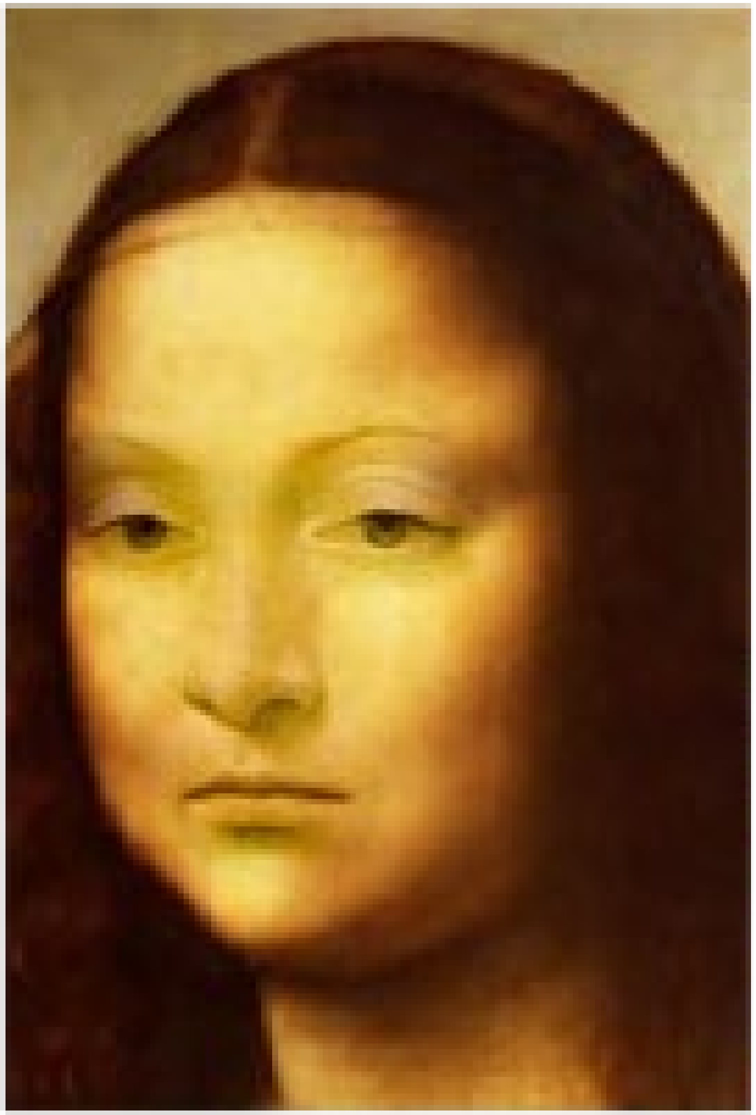
More blending



originals

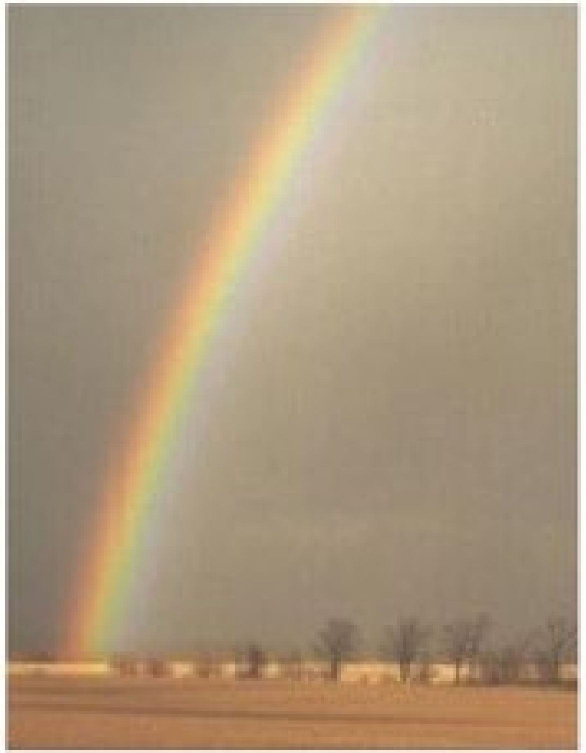


copy-paste

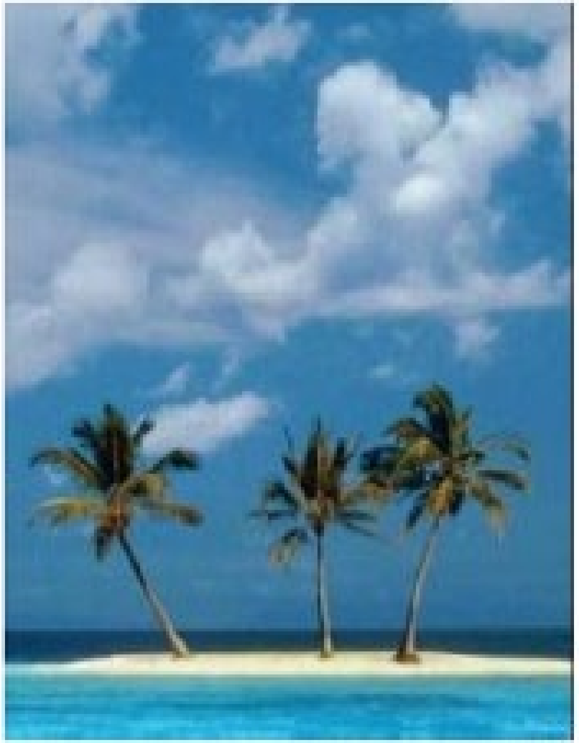


Poisson blending

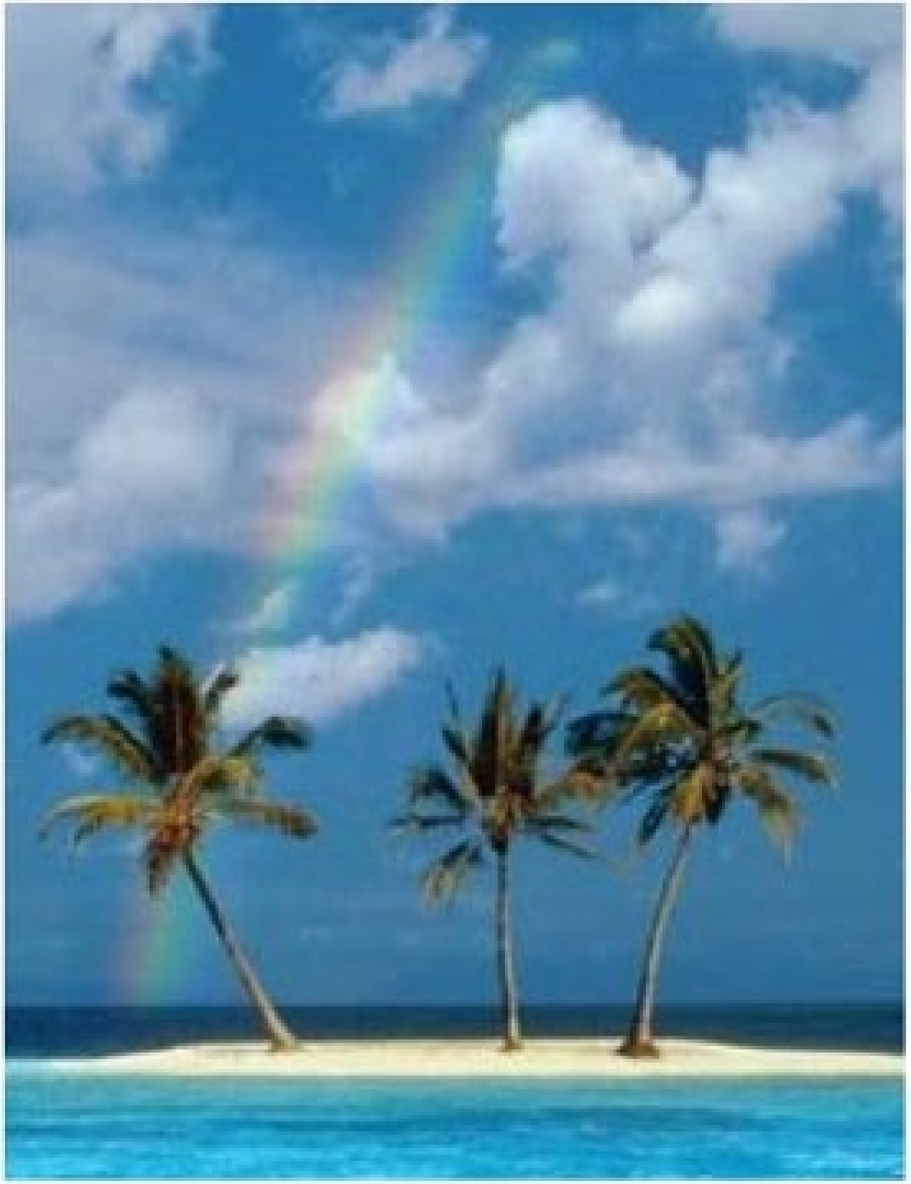
Blending transparent objects



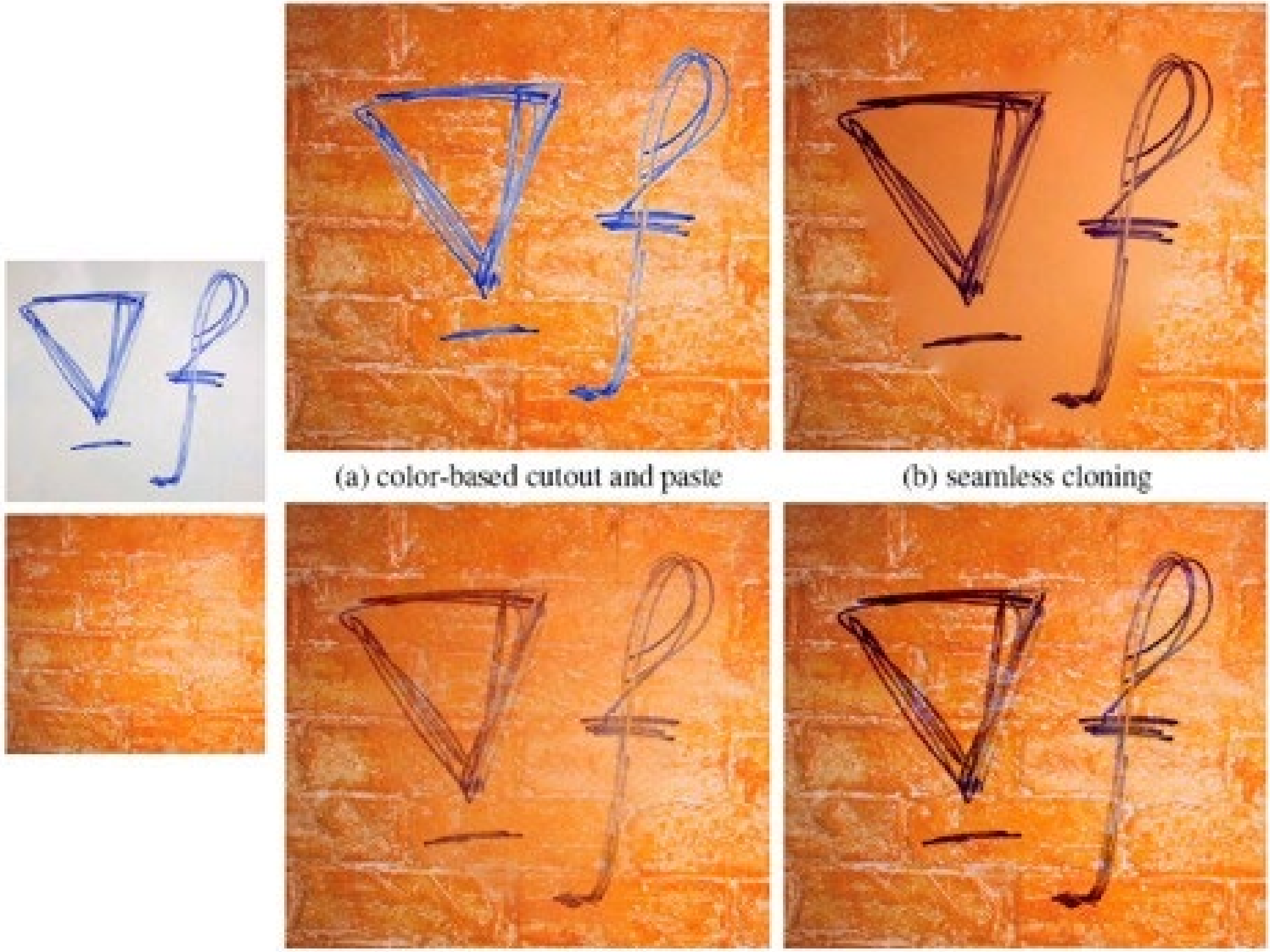
source



destination



Blending objects with holes



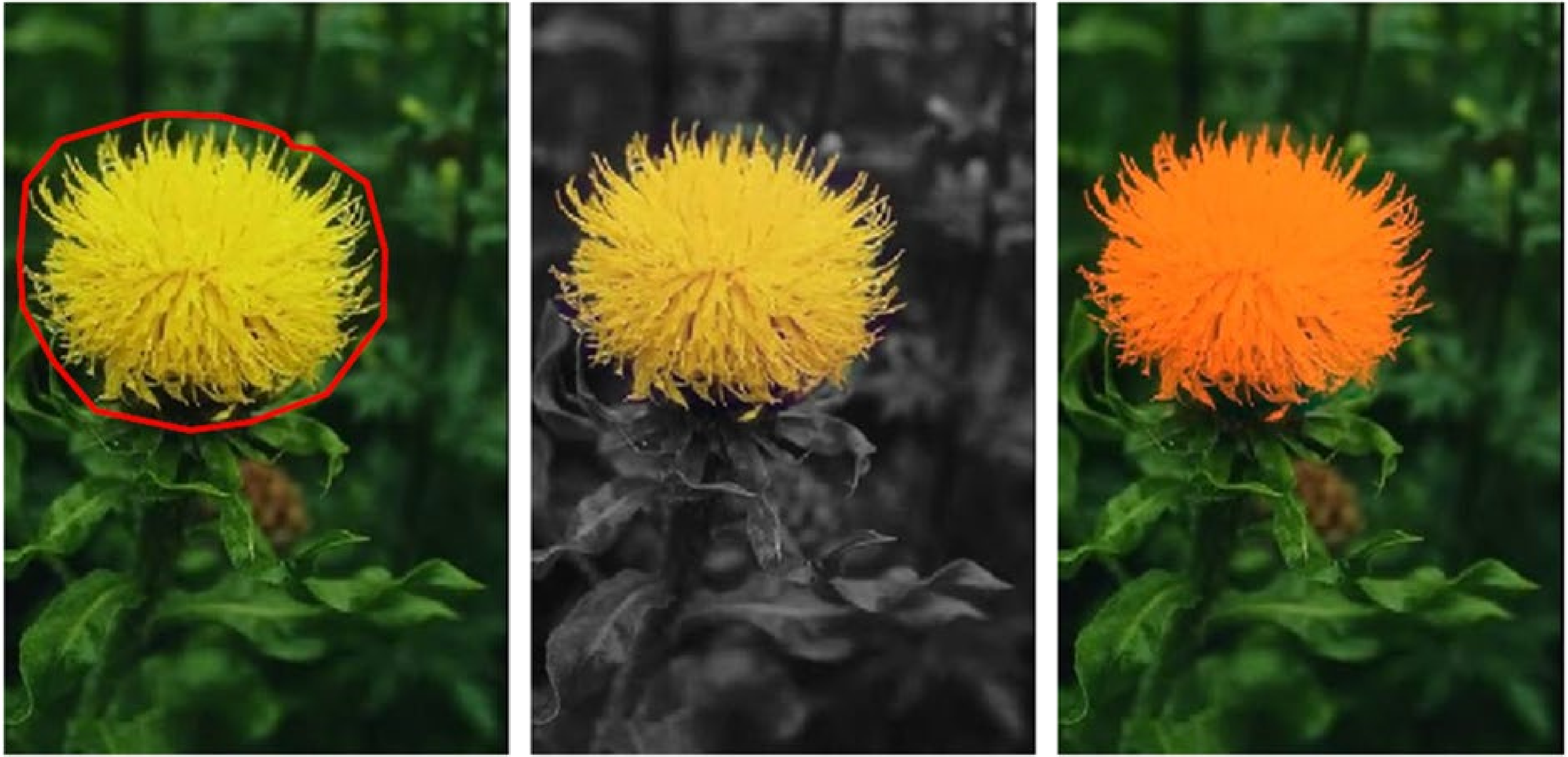
(a) color-based cutout and paste

(b) seamless cloning

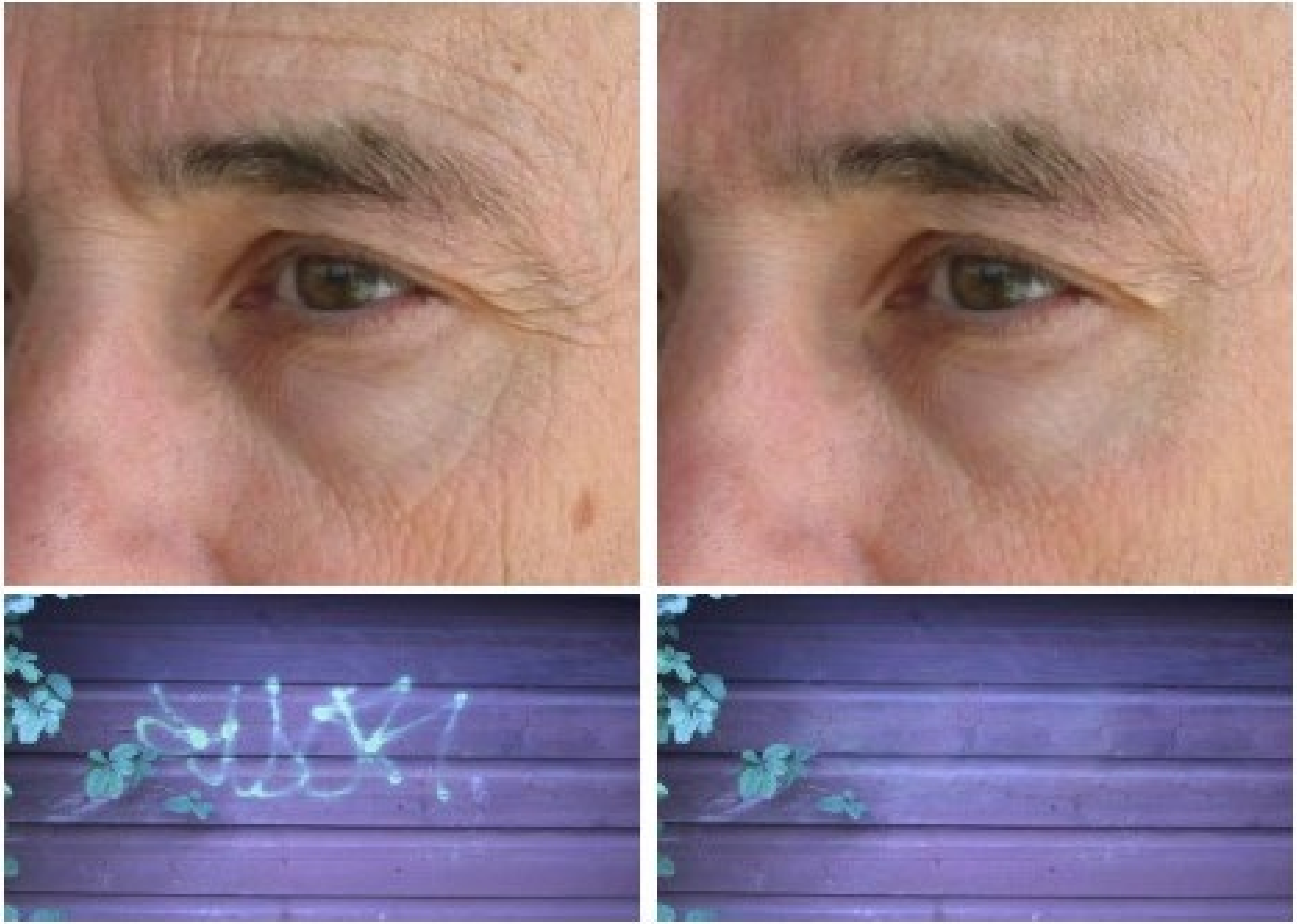
(c) seamless cloning and destination averaged

(d) mixed seamless cloning

Editing

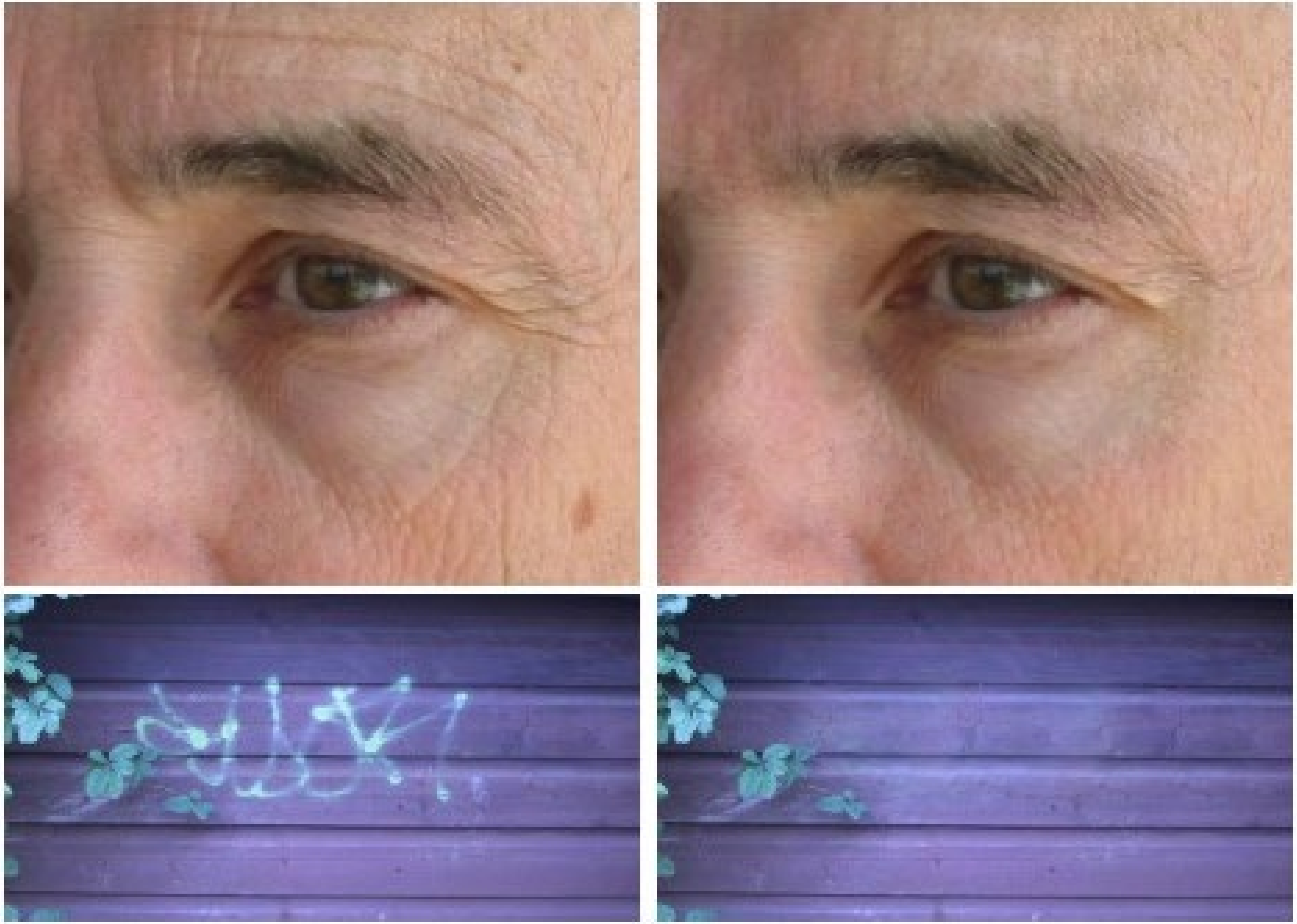


Concealment



How would you do this with Poisson blending?

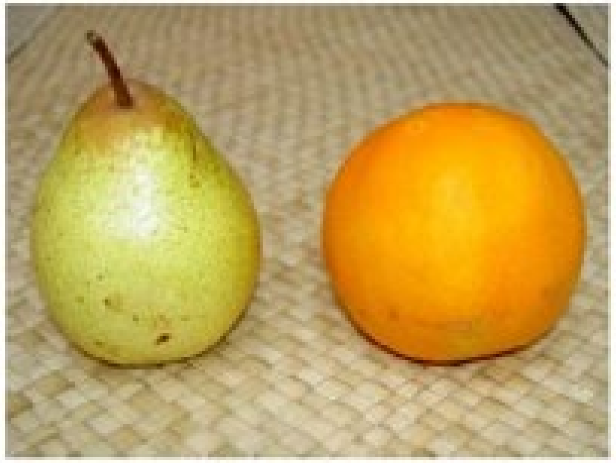
Concealment

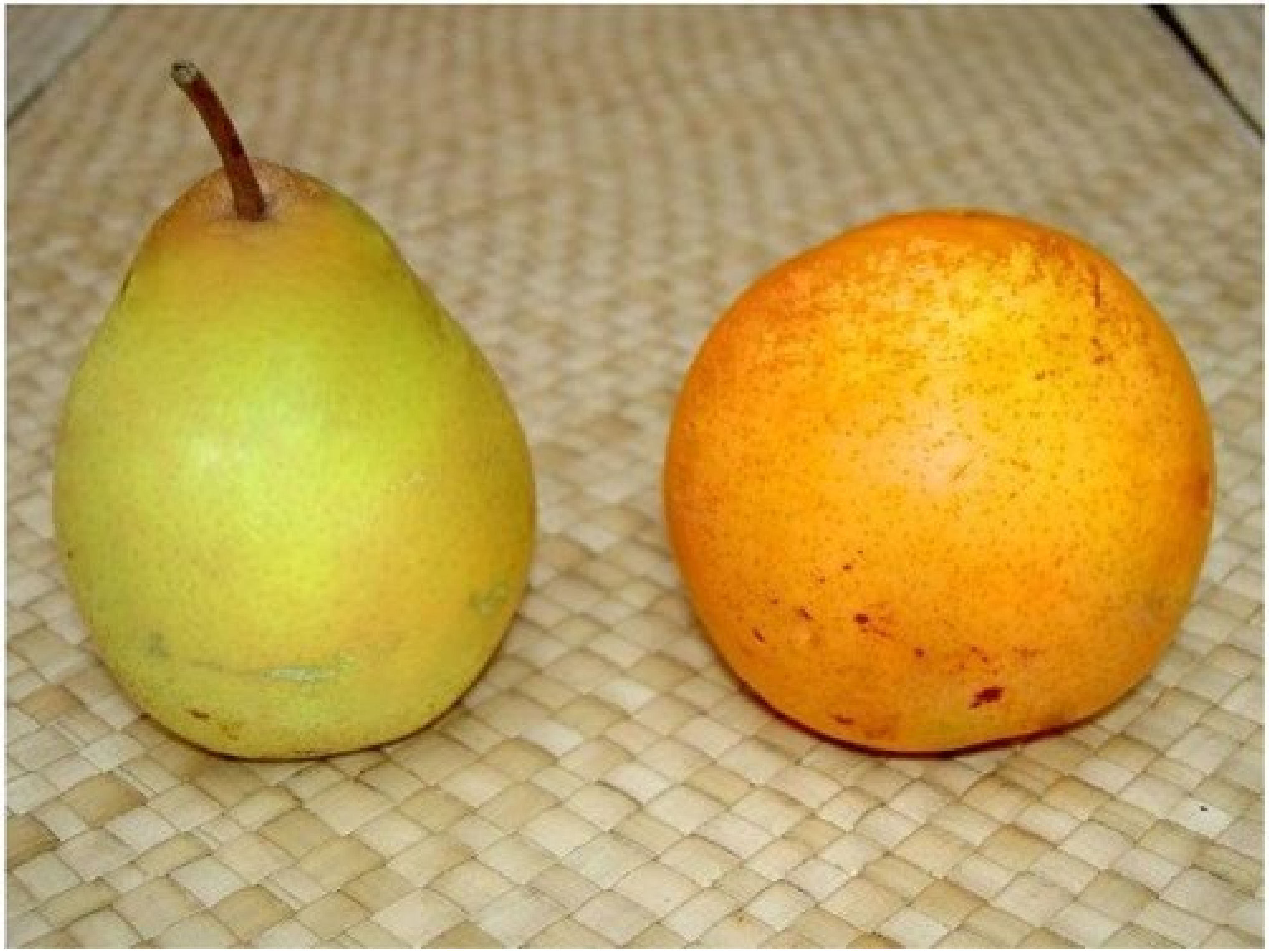


How would you do this with Poisson blending?

- Insert a copy of the background.

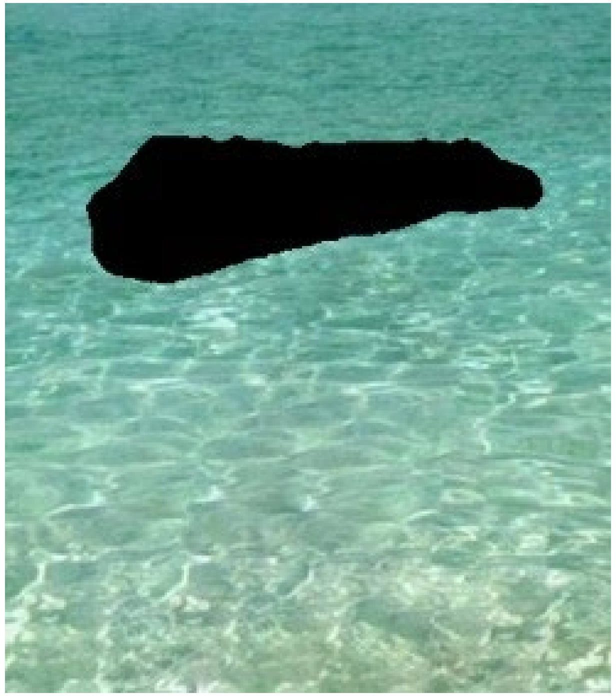
Texture swapping

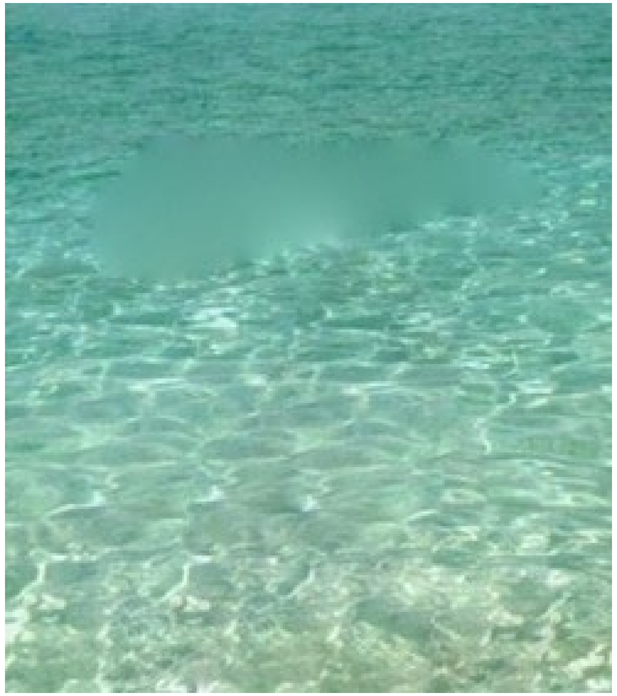




Special case: membrane interpolation

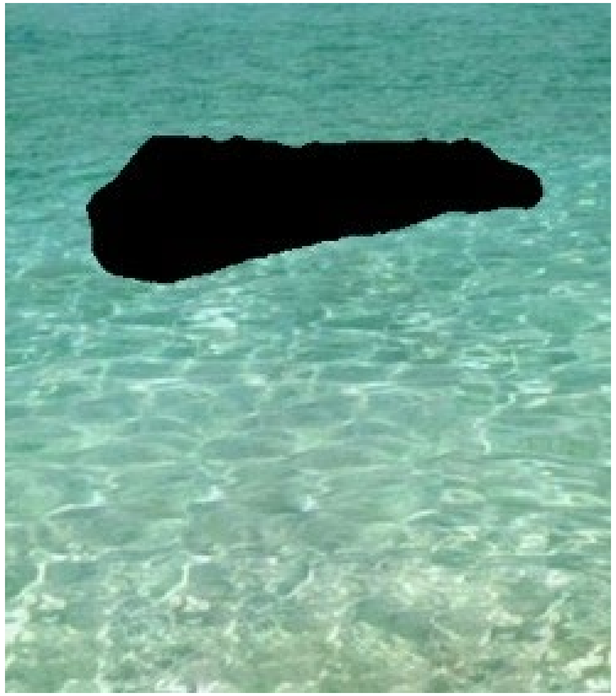
How would you do this?

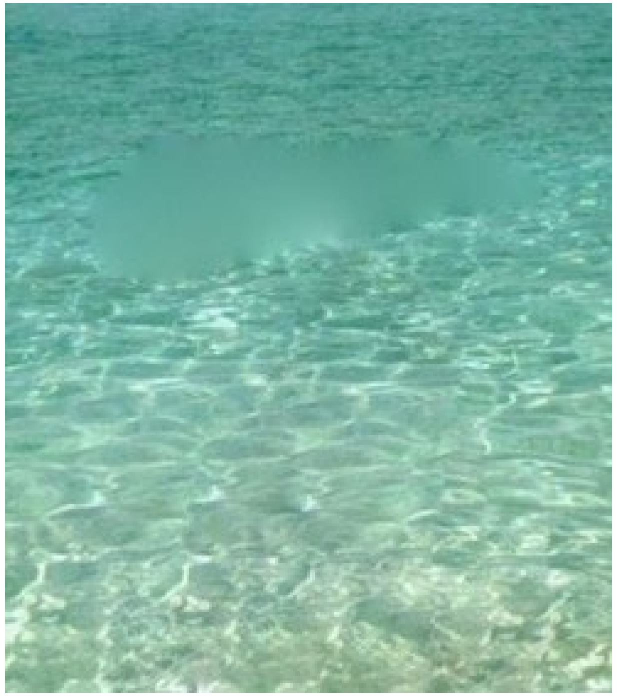




Special case: membrane interpolation

How would you do this?





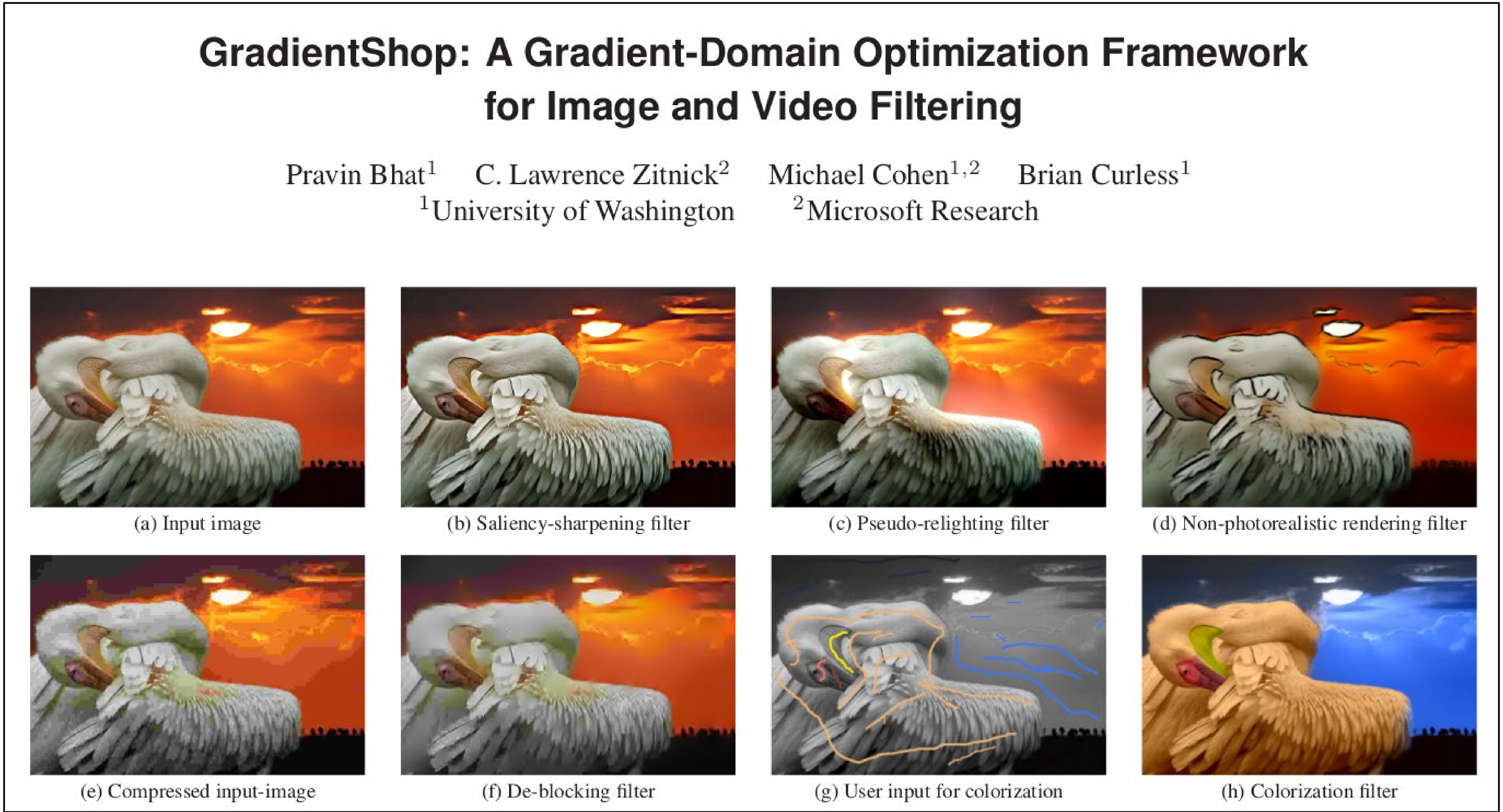
Poisson problem

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Laplacian problem

$$\min_f \iint_\Omega |\nabla f|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Entire suite of image editing tools



GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering

Pravin Bhat[1] C. Lawrence Zitnick[2] Michael Cohen[1,2] Brian Curless[1]
[1]University of Washington [2]Microsoft Research

(a) Input image

(b) Saliency-sharpening filter

(c) Pseudo-relighting filter

(d) Non-photorealistic rendering filter

(e) Compressed input-image

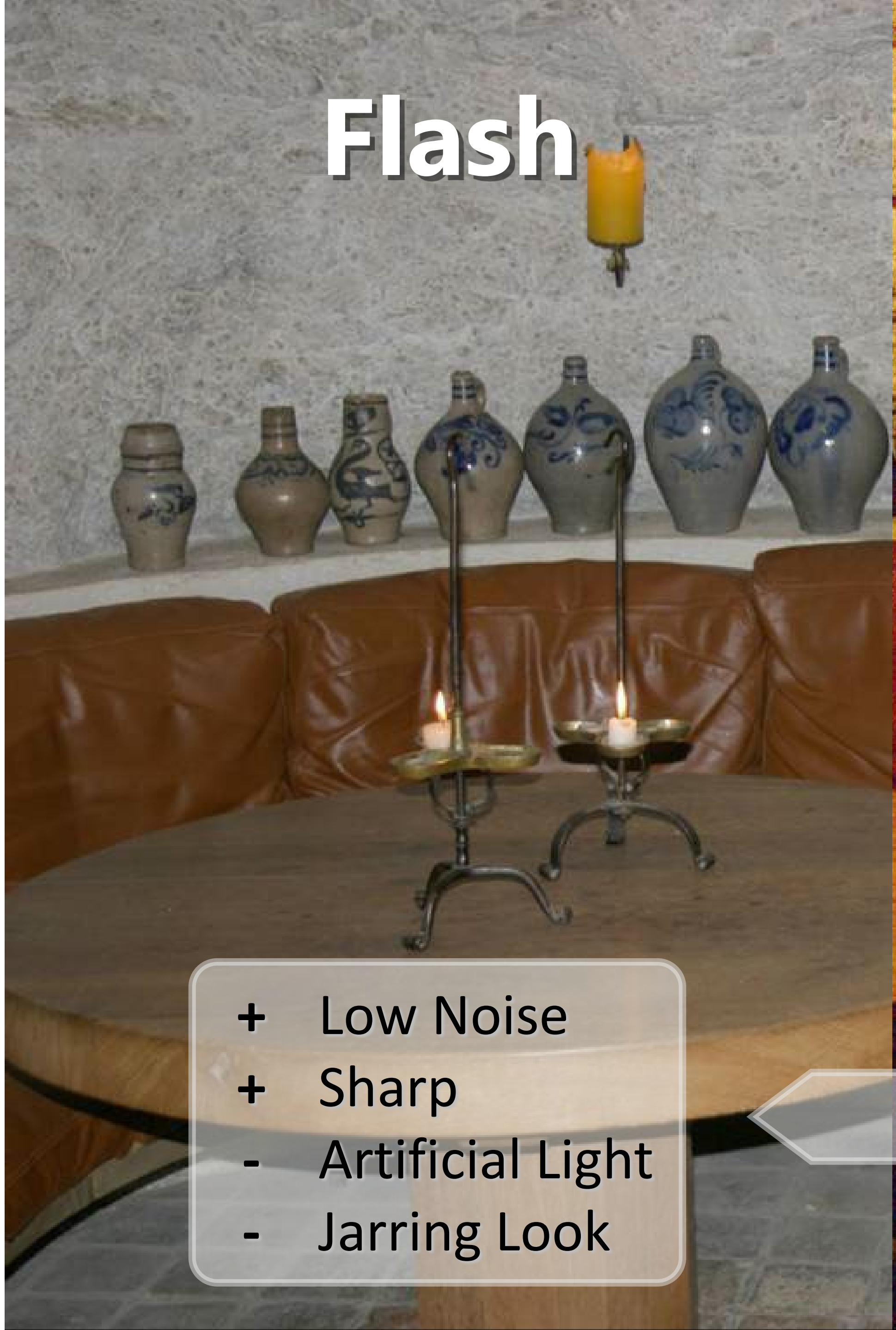
(f) De-blocking filter

(g) User input for colorization

(h) Colorization filter

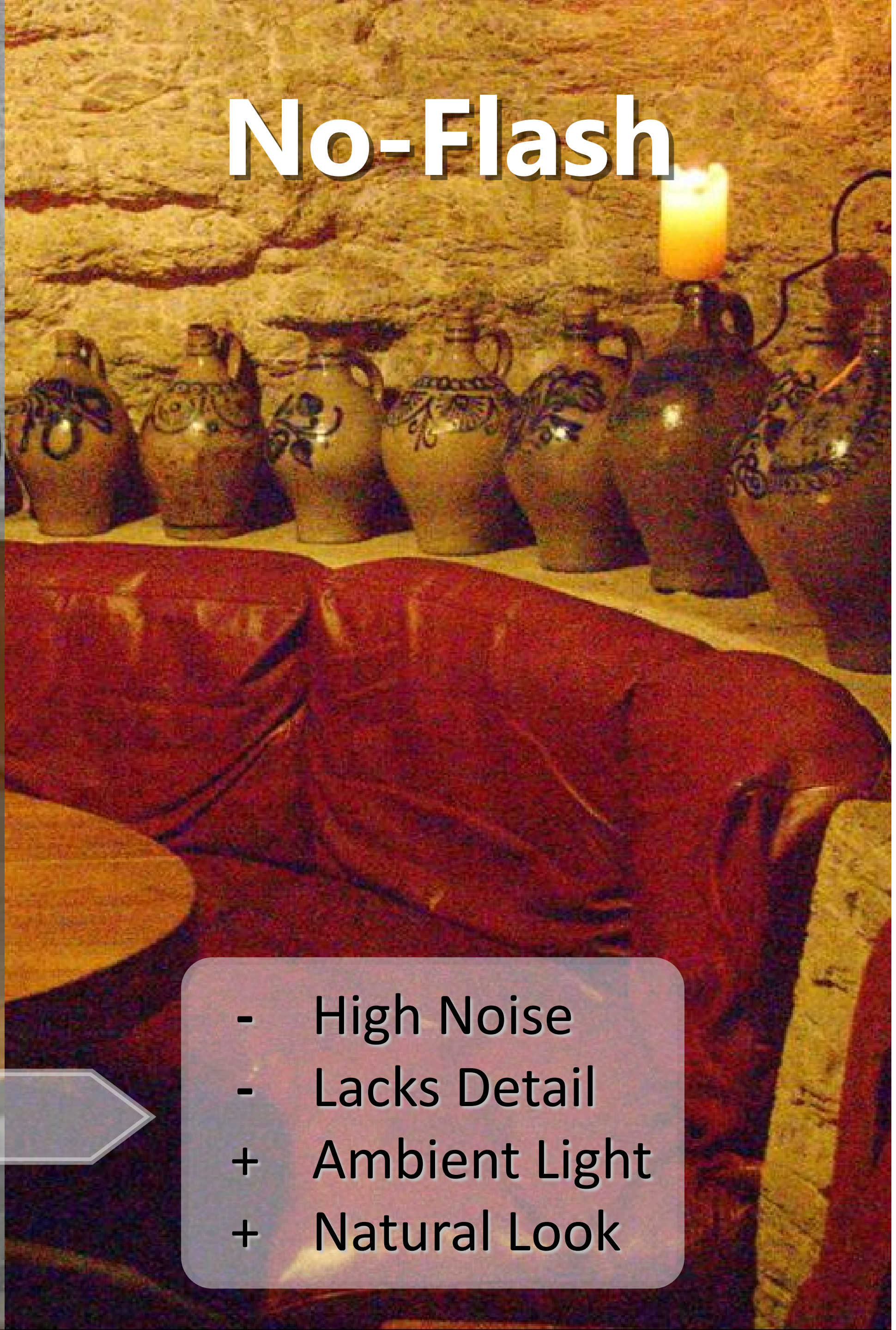
Flash/no-flash photography

Flash



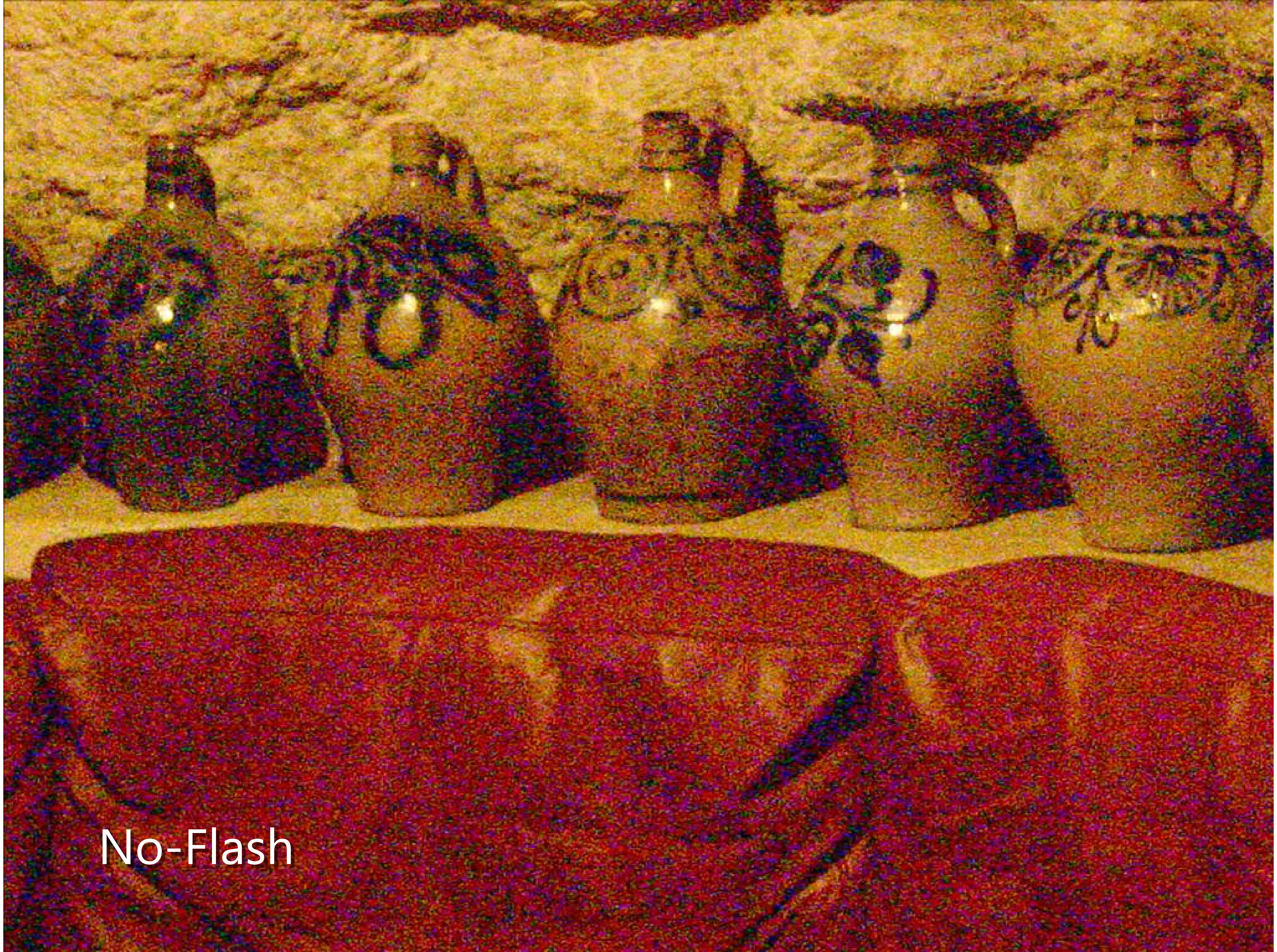
- + Low Noise
- + Sharp
- - Artificial Light
- - Jarring Look

No-Flash

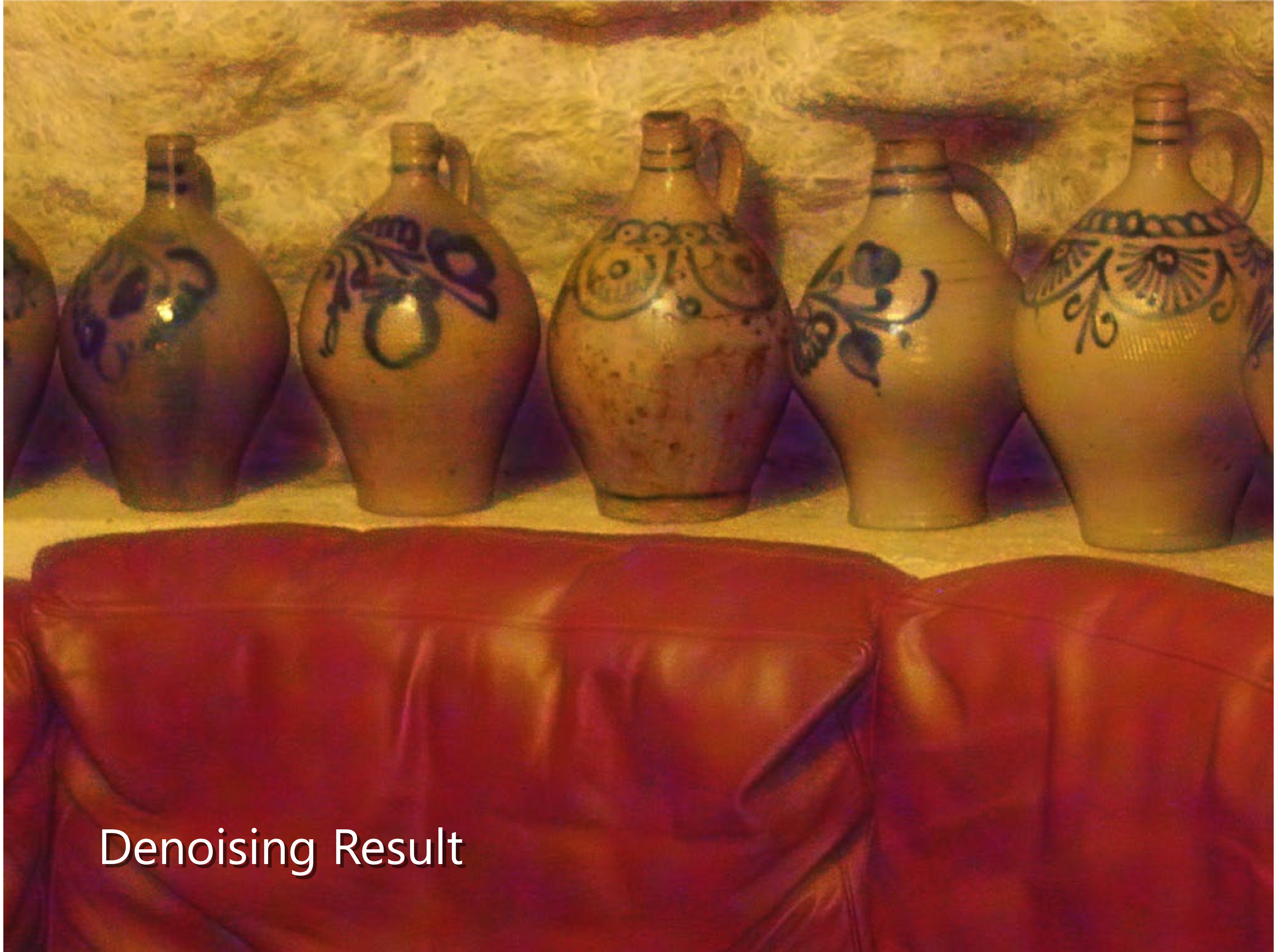


- - High Noise
- - Lacks Detail
- + Ambient Light
- + Natural Look

Denoising Result



No-Flash



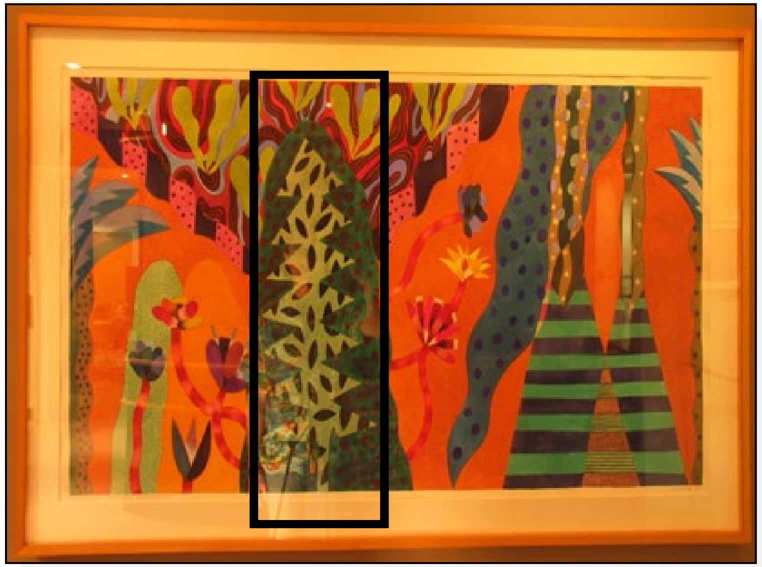
Denoising Result

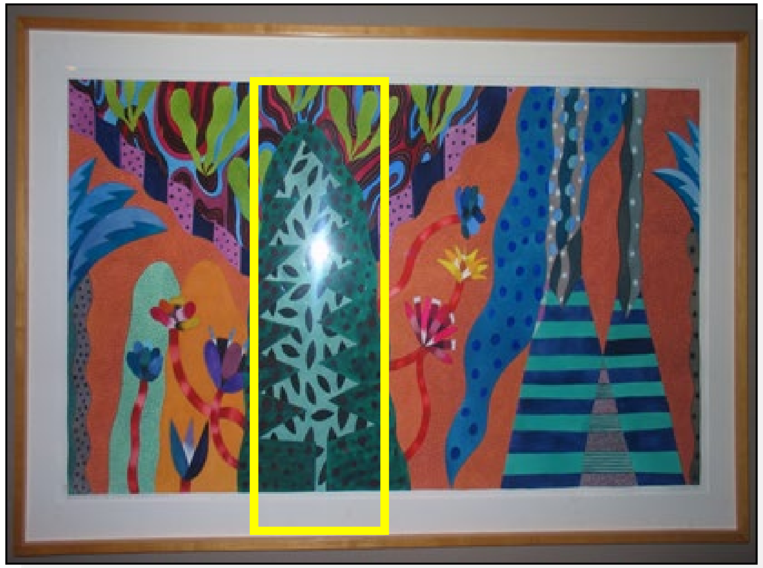
Key idea

Denoise the no-flash image while maintaining the edge structure of the flash image.

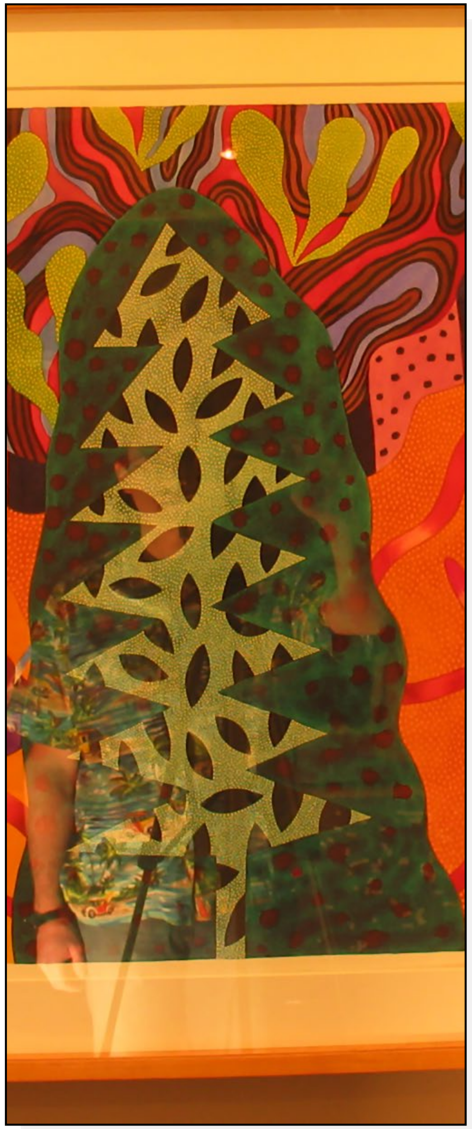
Can we do similar flash/no-flash fusion tasks with gradient-domain processing?

Removing self-reflections and hot-spots

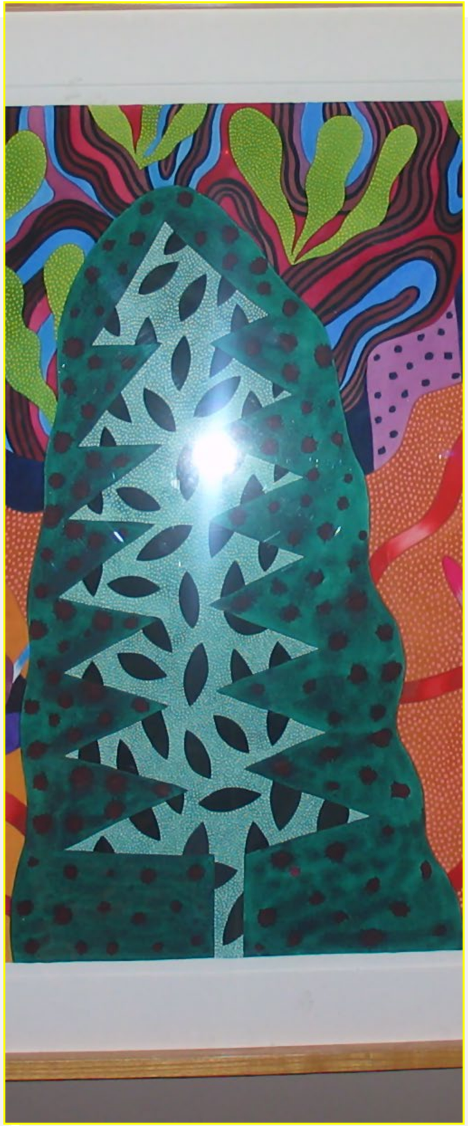




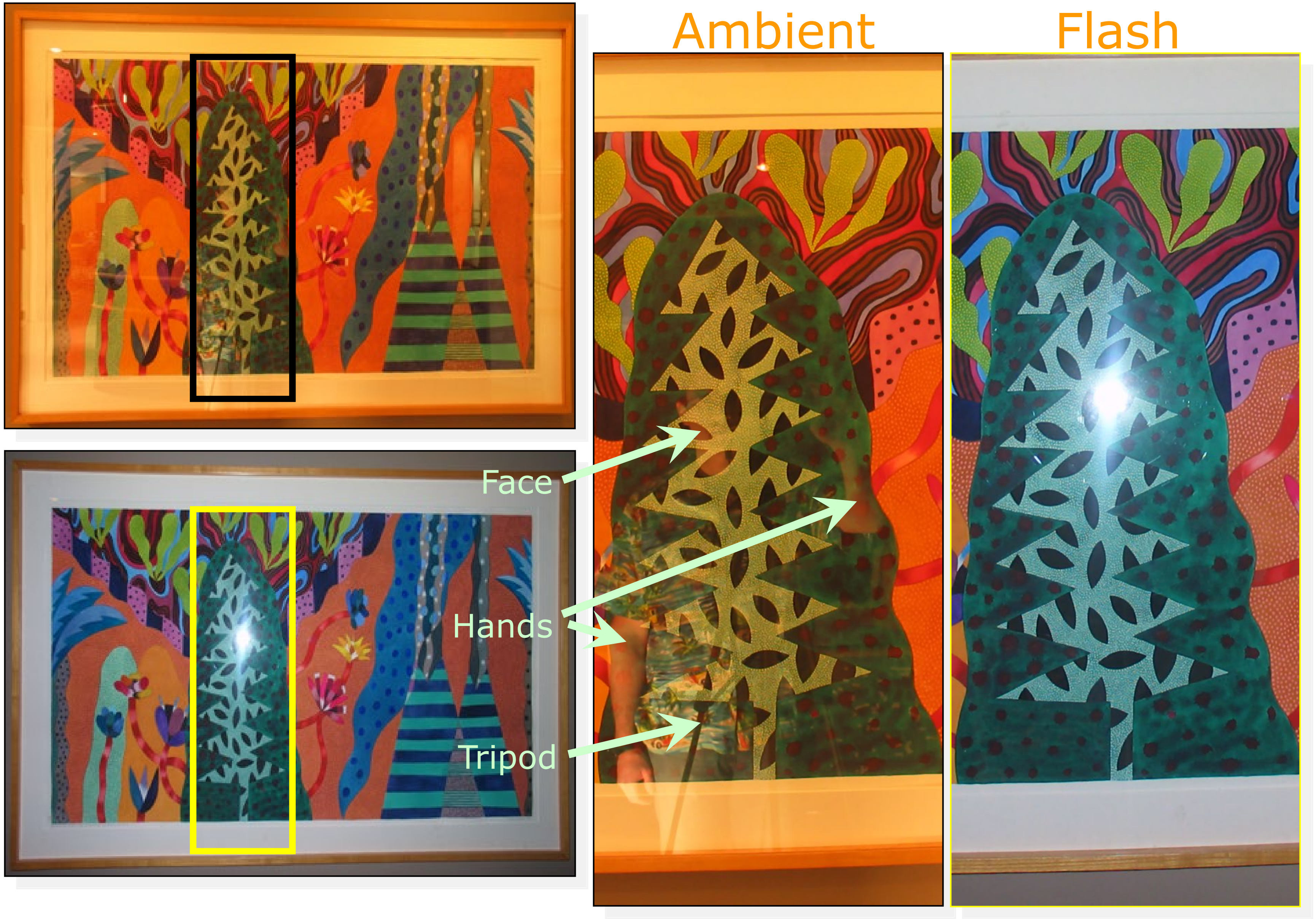
Ambient



Flash



Removing self-reflections and hot-spots



Ambient

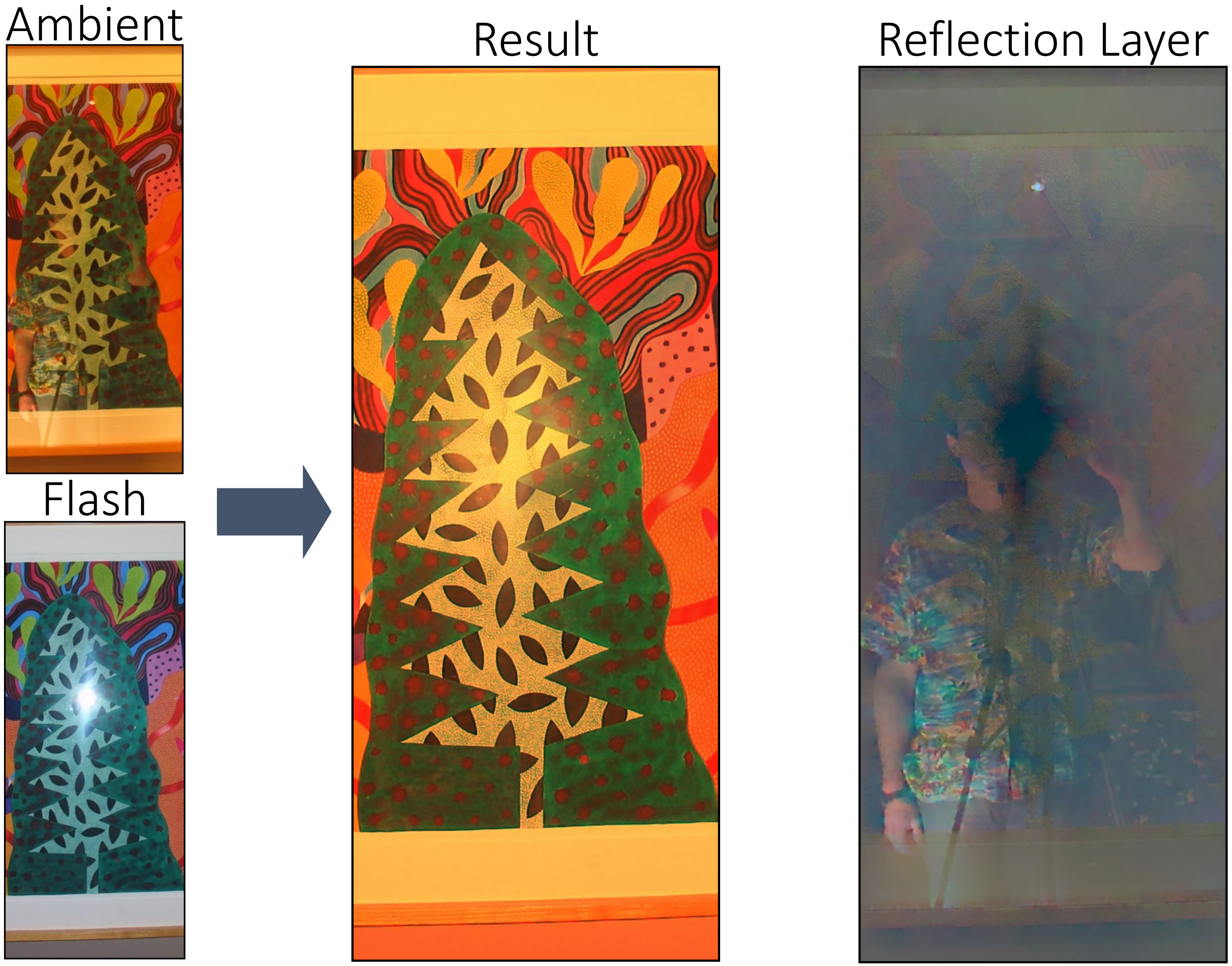
Flash

Face

Hands

Tripod

Removing self-reflections and hot-spots



Ambient

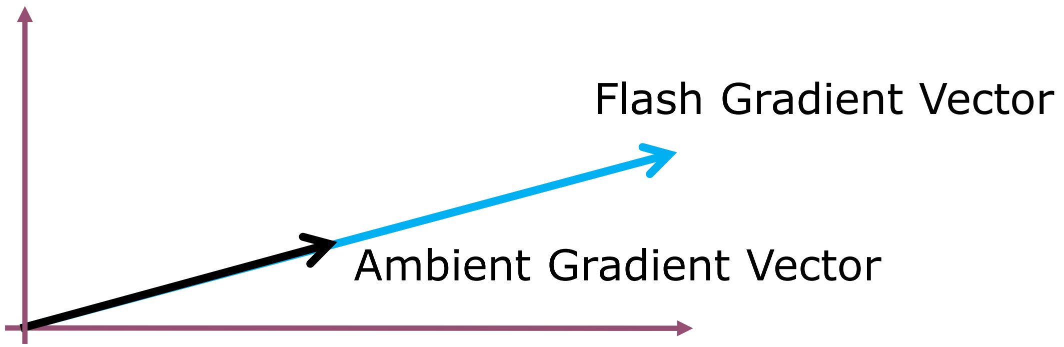
Flash

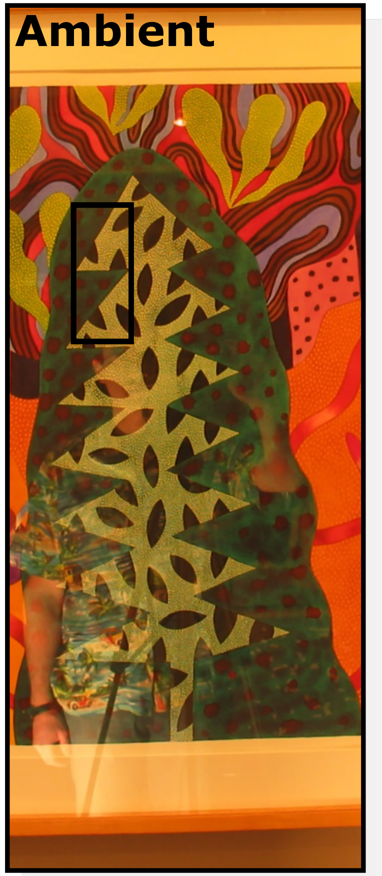
Result

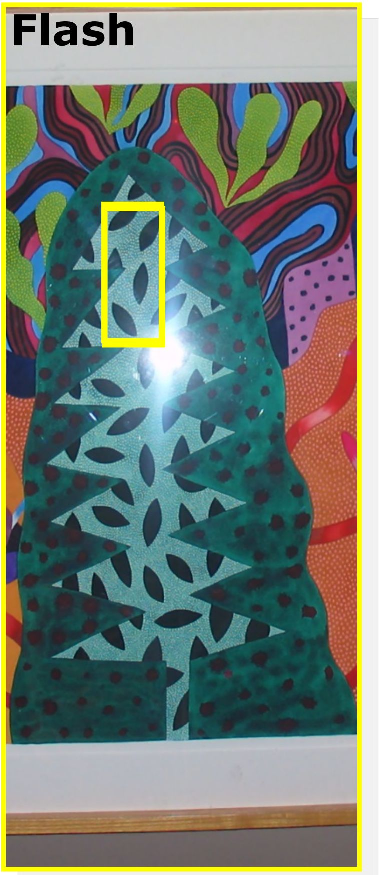
Reflection Layer

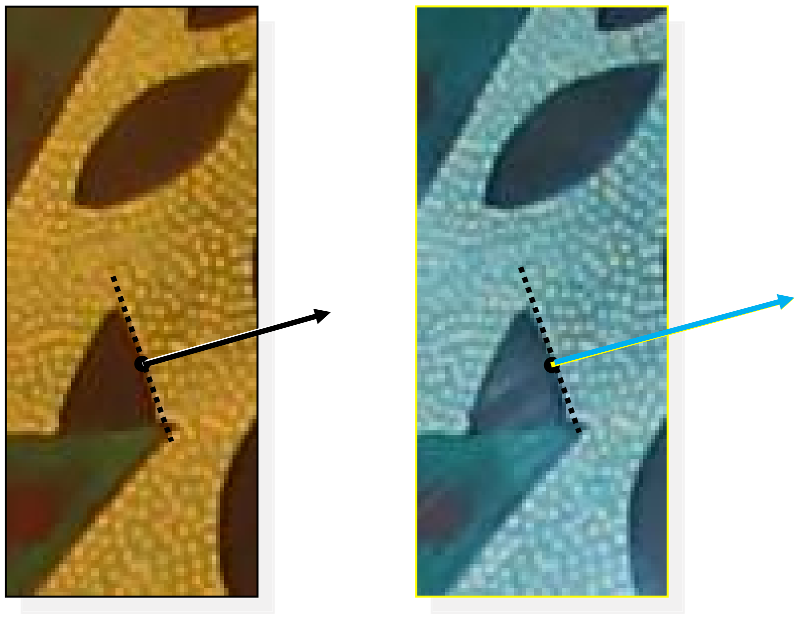
Idea: look at how gradients are affected

Same gradient vector direction





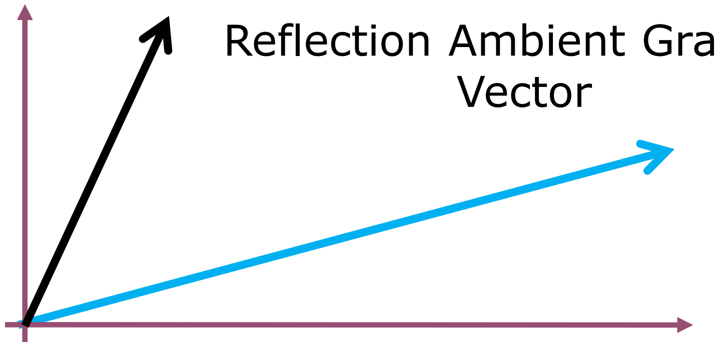


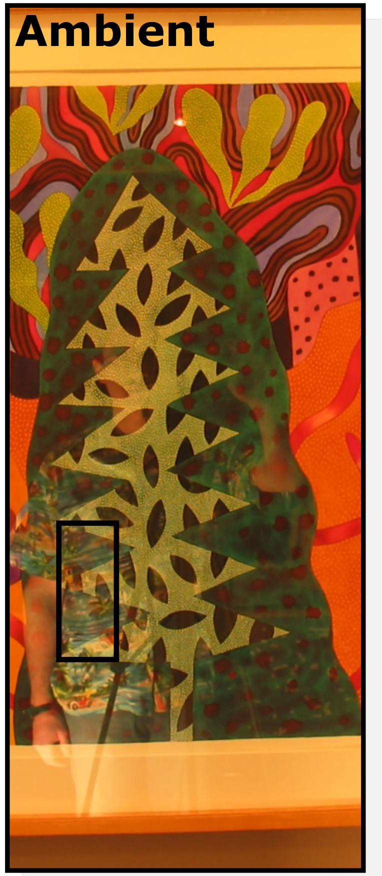


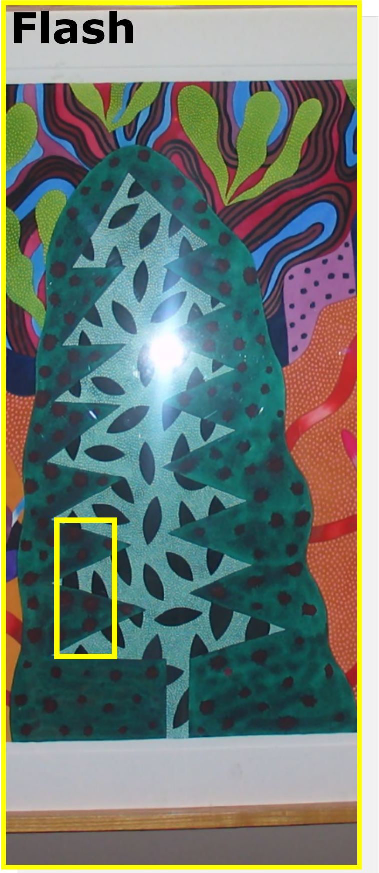
No reflections

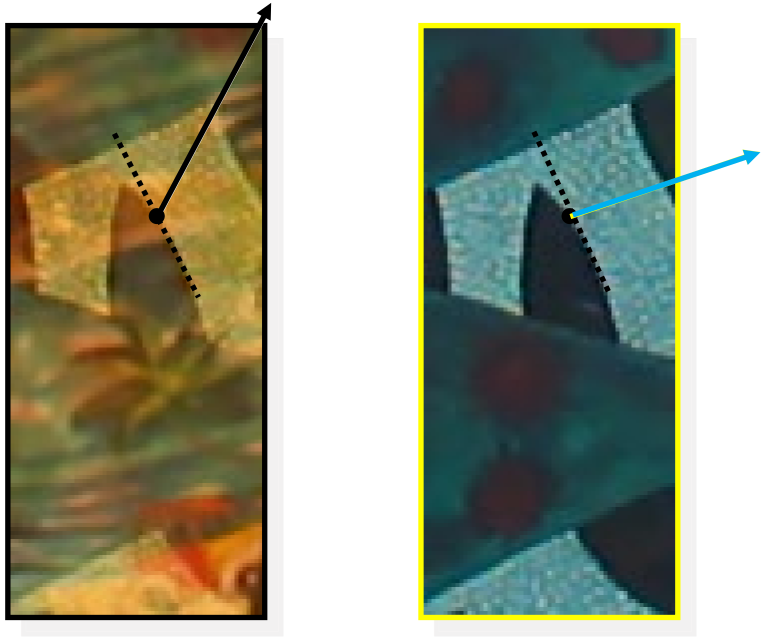
Idea: look at how gradients are affected

Different gradient vector direction



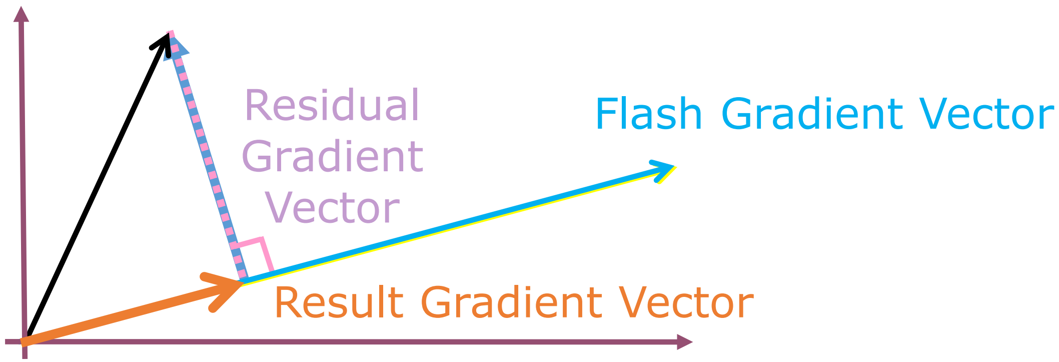


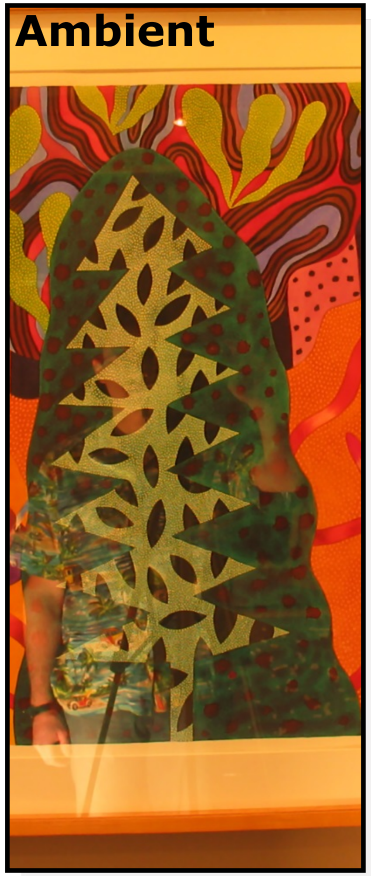


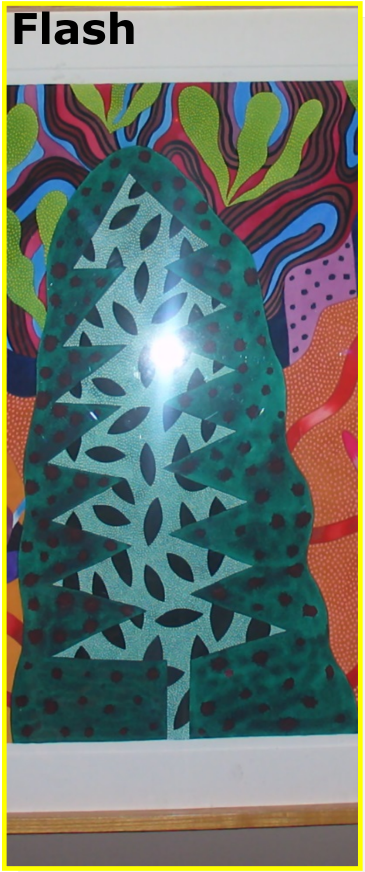


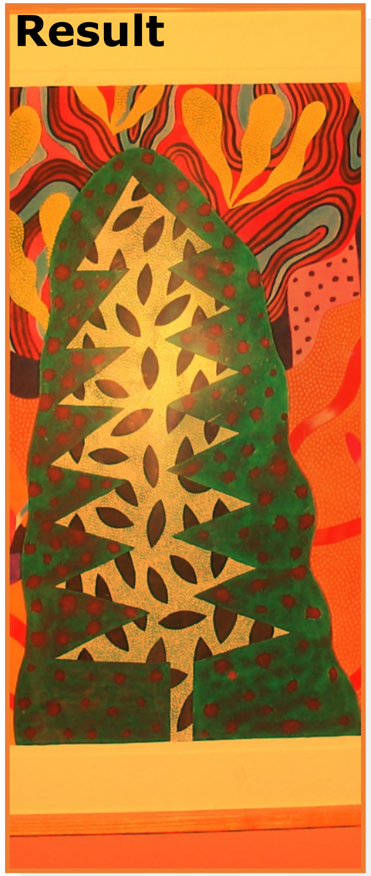
With reflections

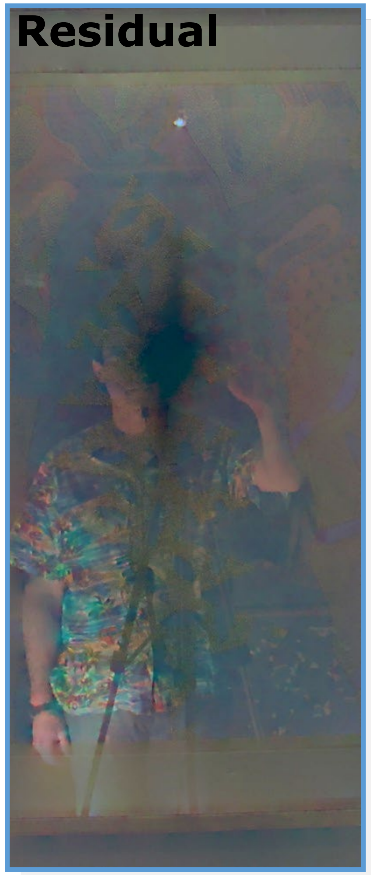
Gradient projections



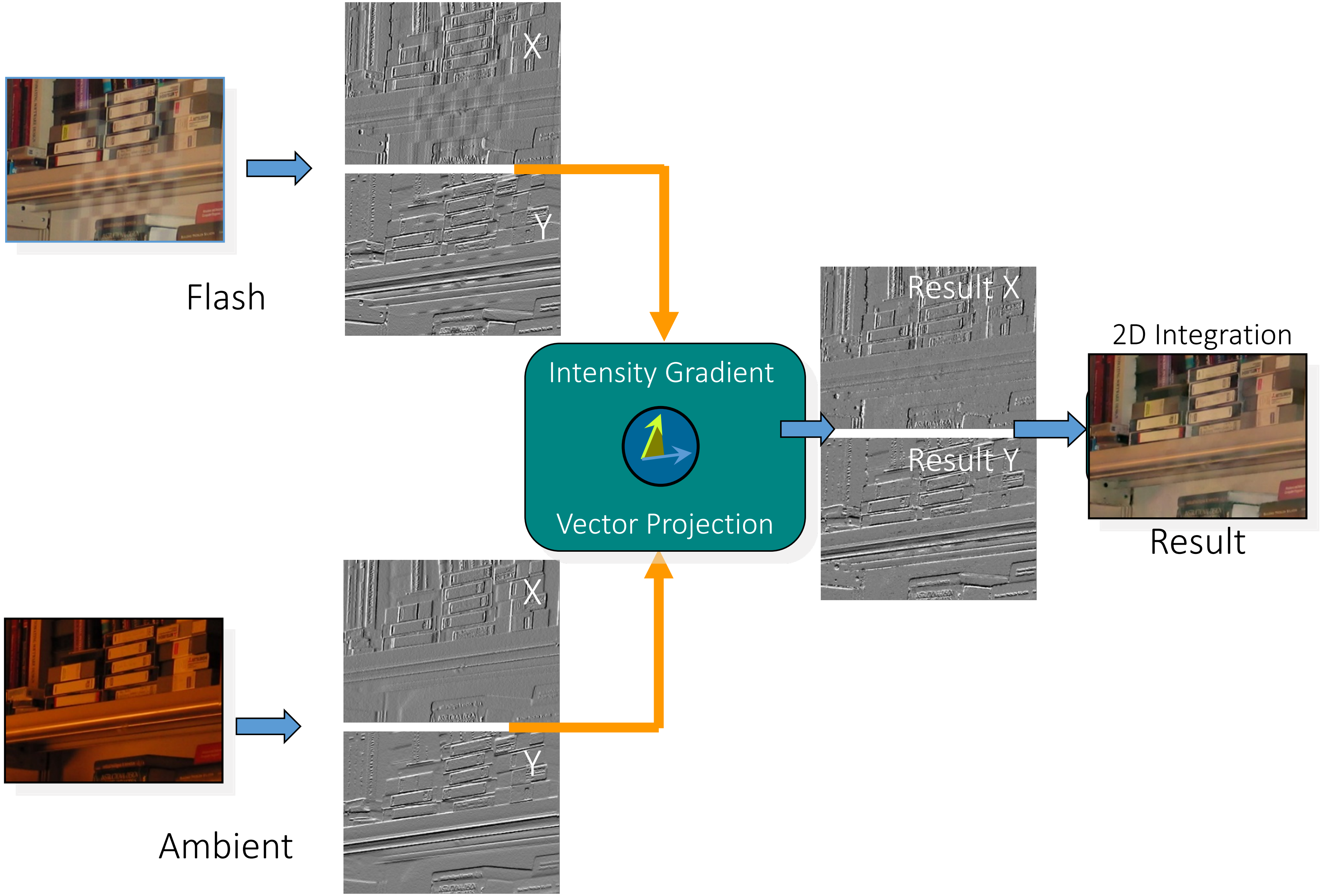






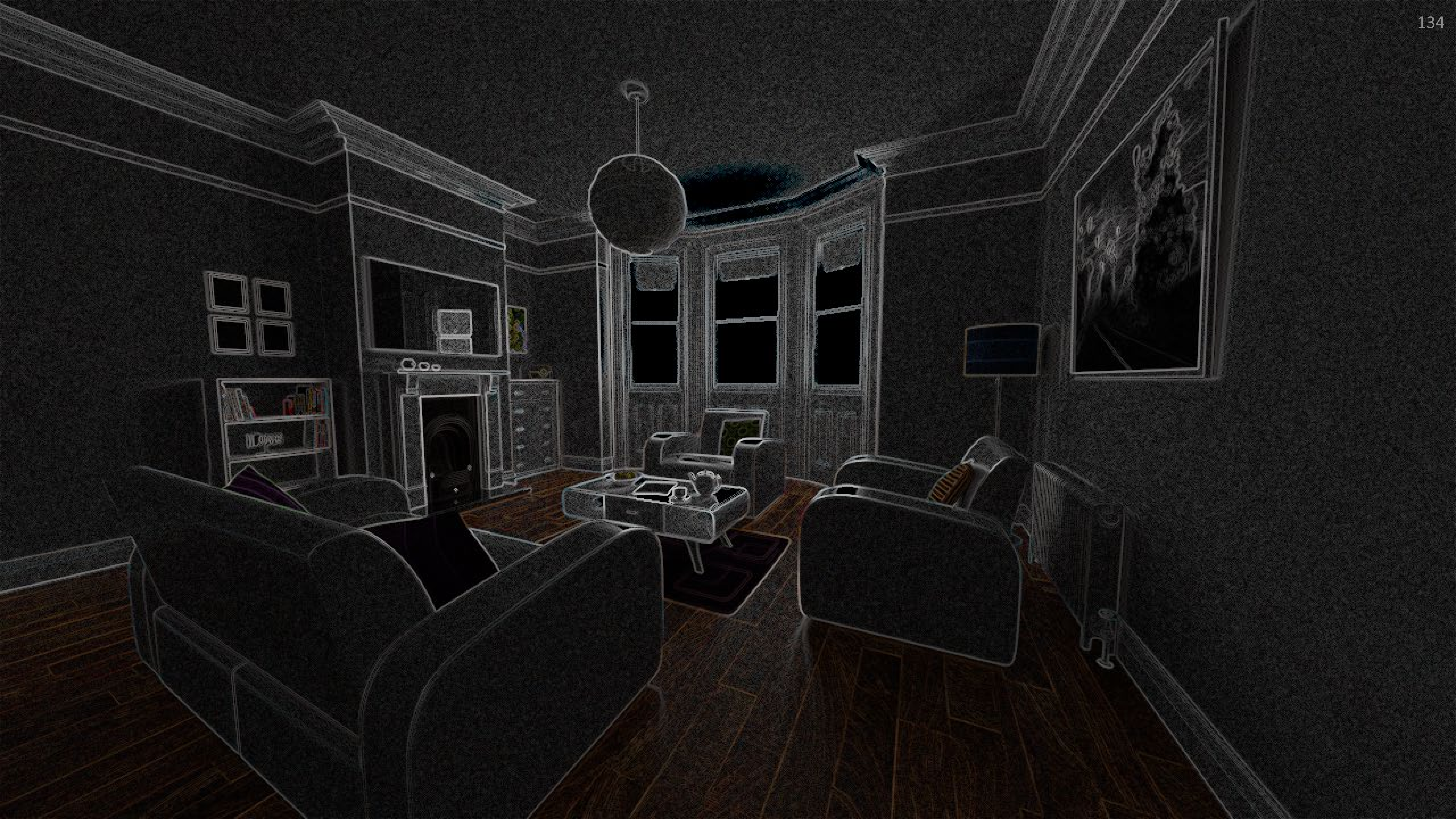


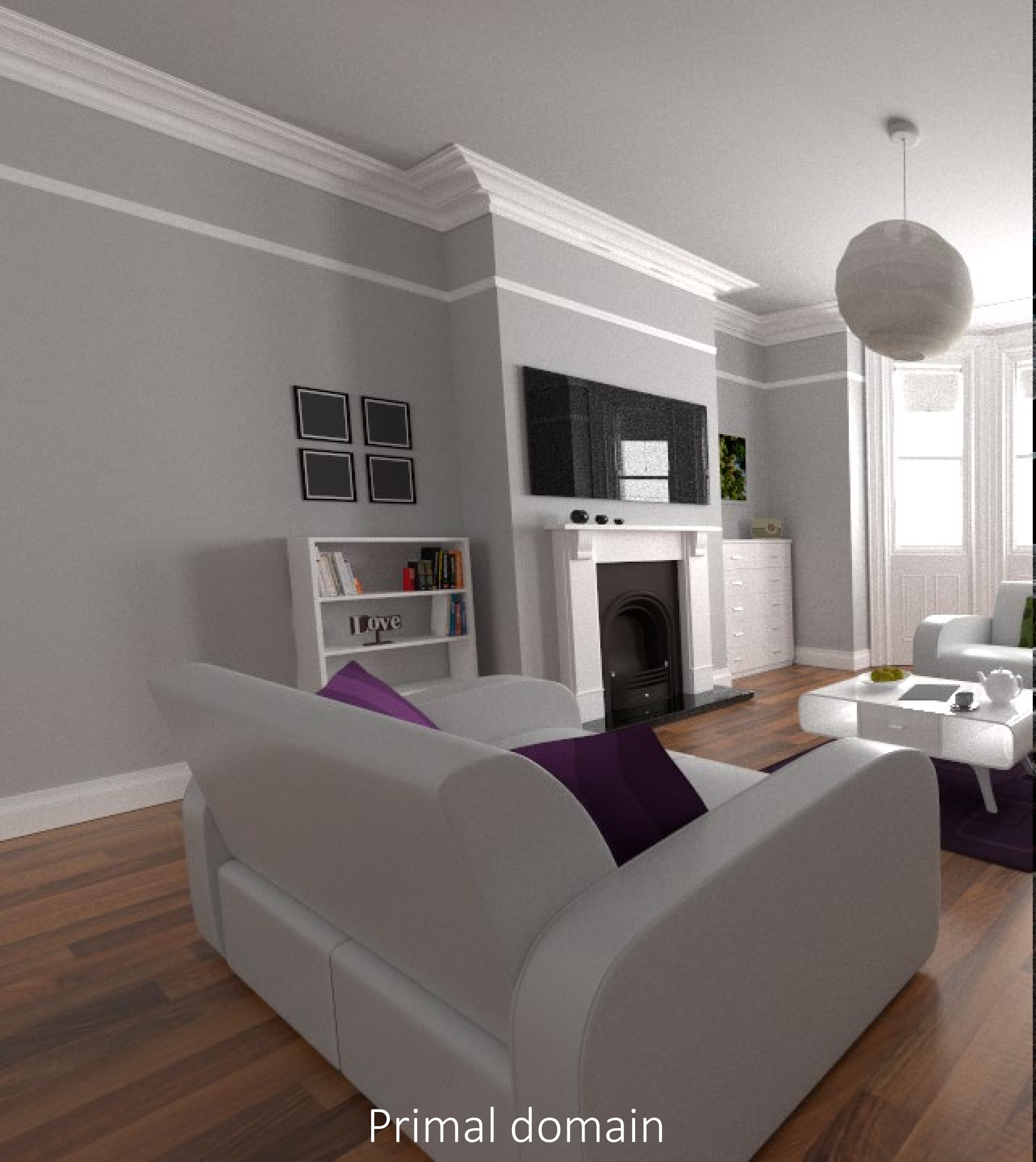
Flash/no-flash with gradient-domain processing



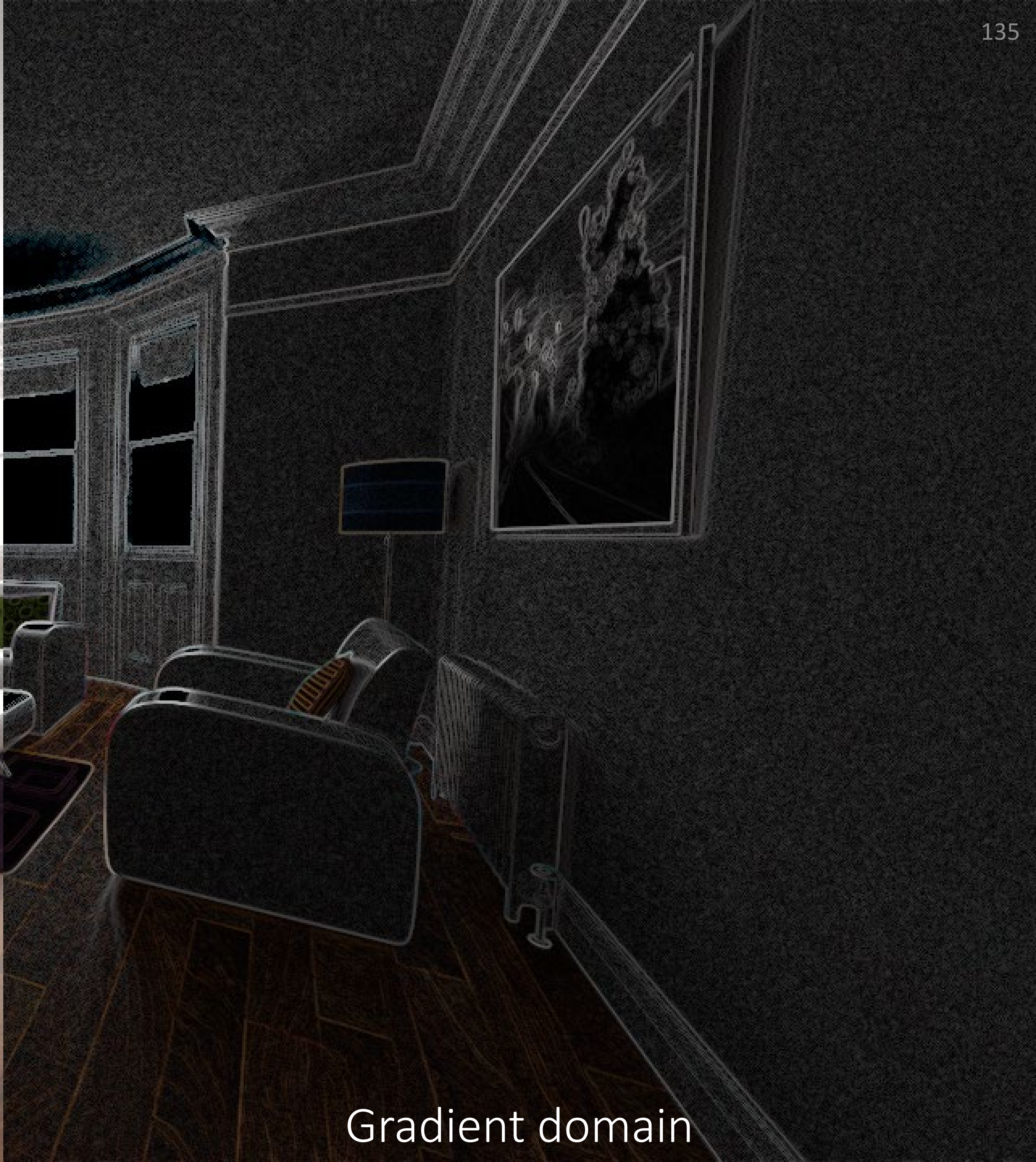
Gradient-domain rendering

Love

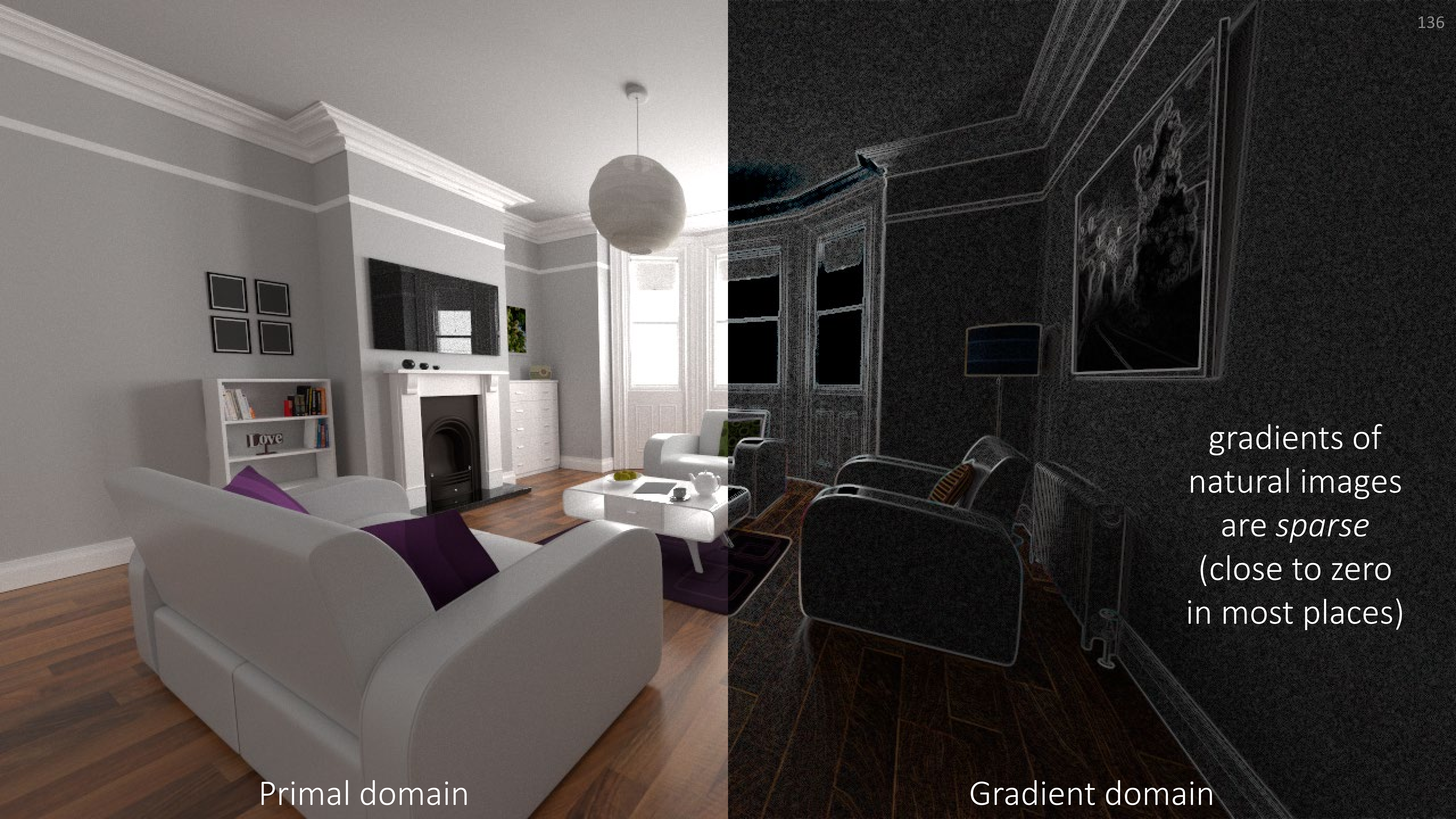




Primal domain



Gradient domain

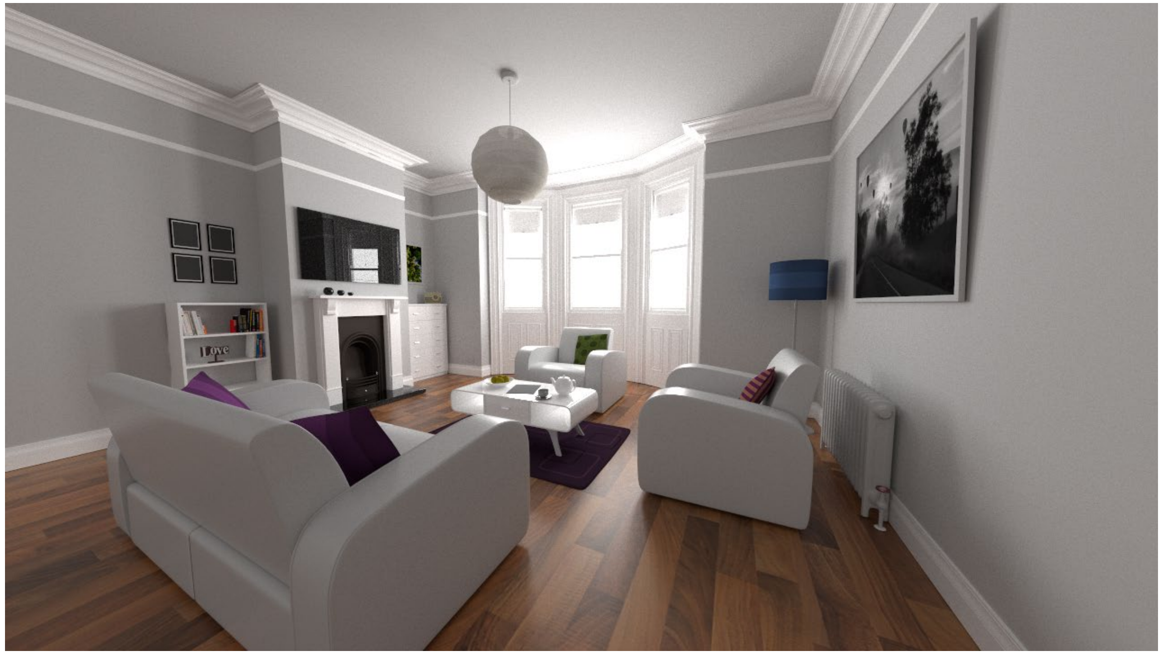


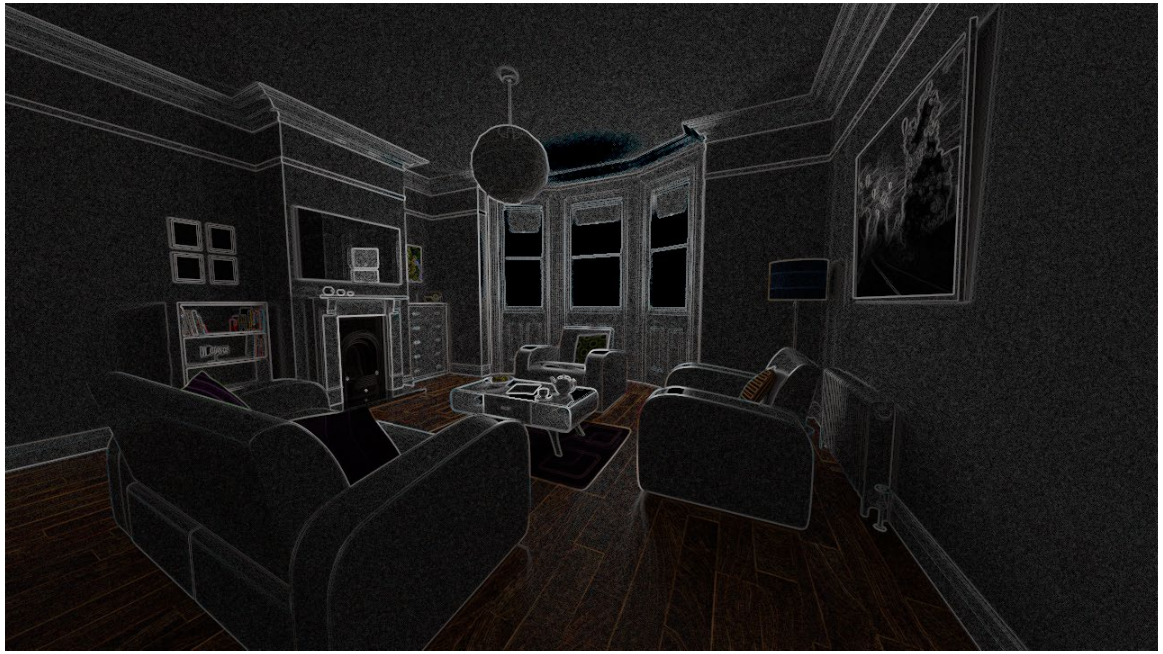
Primal domain

Gradient domain

gradients of
natural images
are *sparse*
(close to zero
in most places)

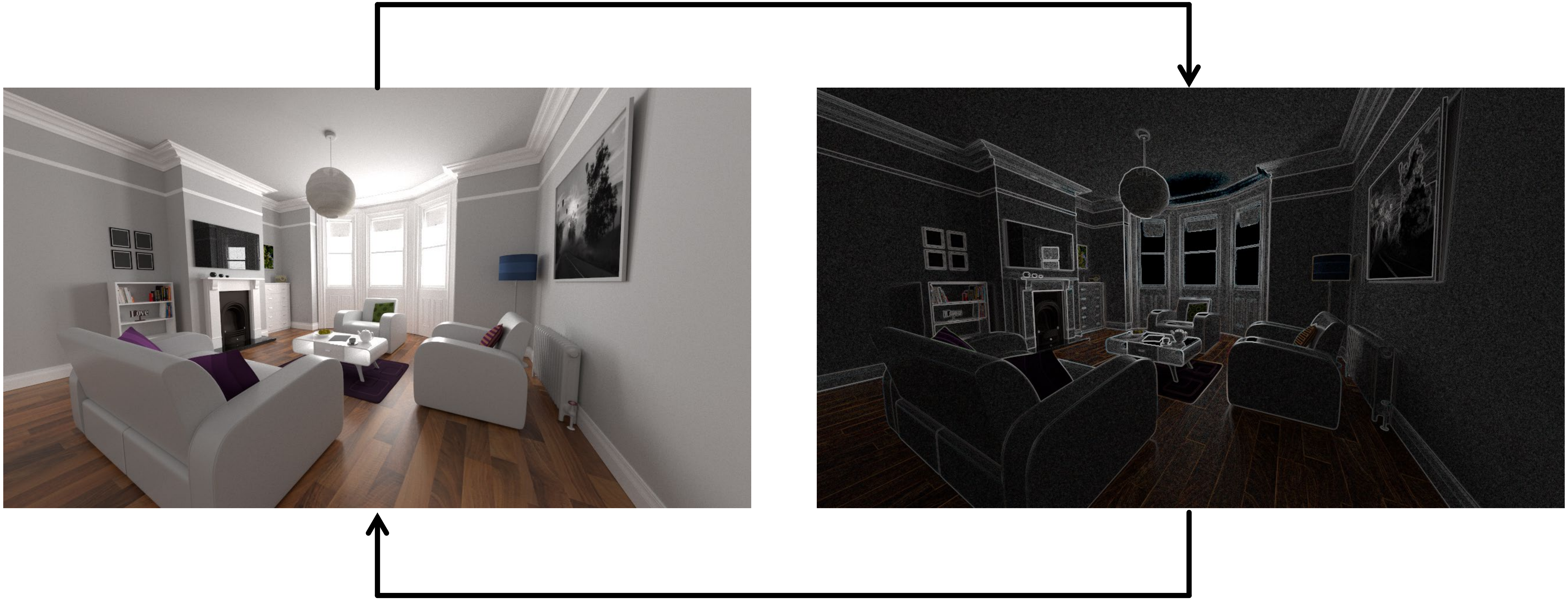
Can I go from one image to the other?





Can I go from one image to the other?

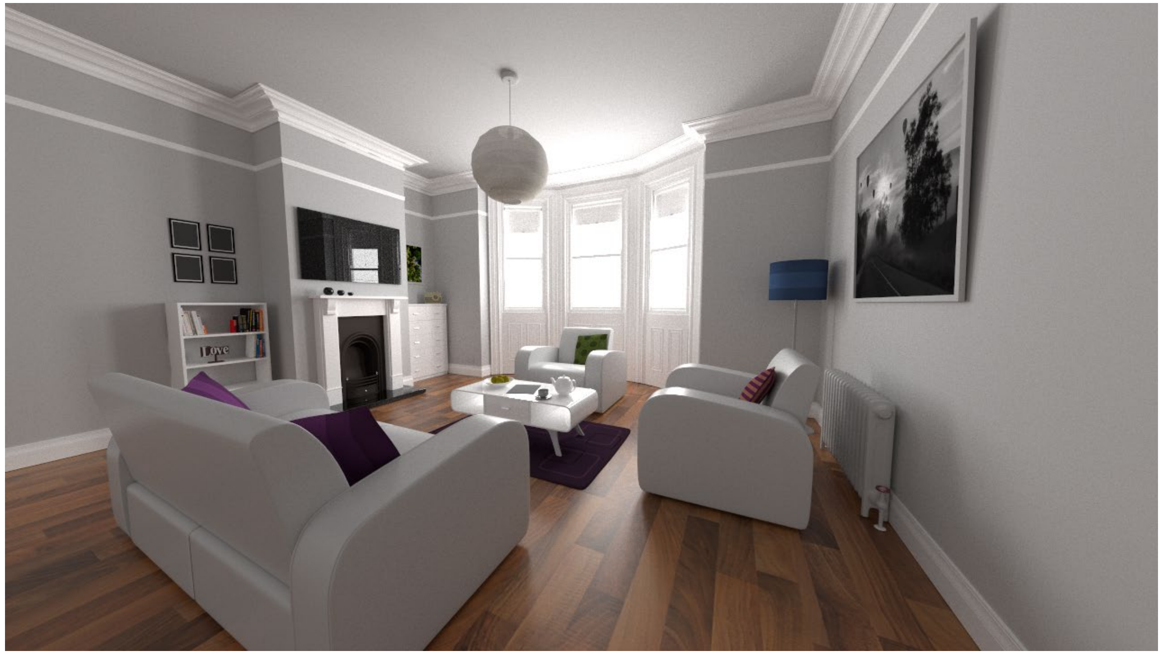
differentiation (e.g., convolution with forward-difference kernel)



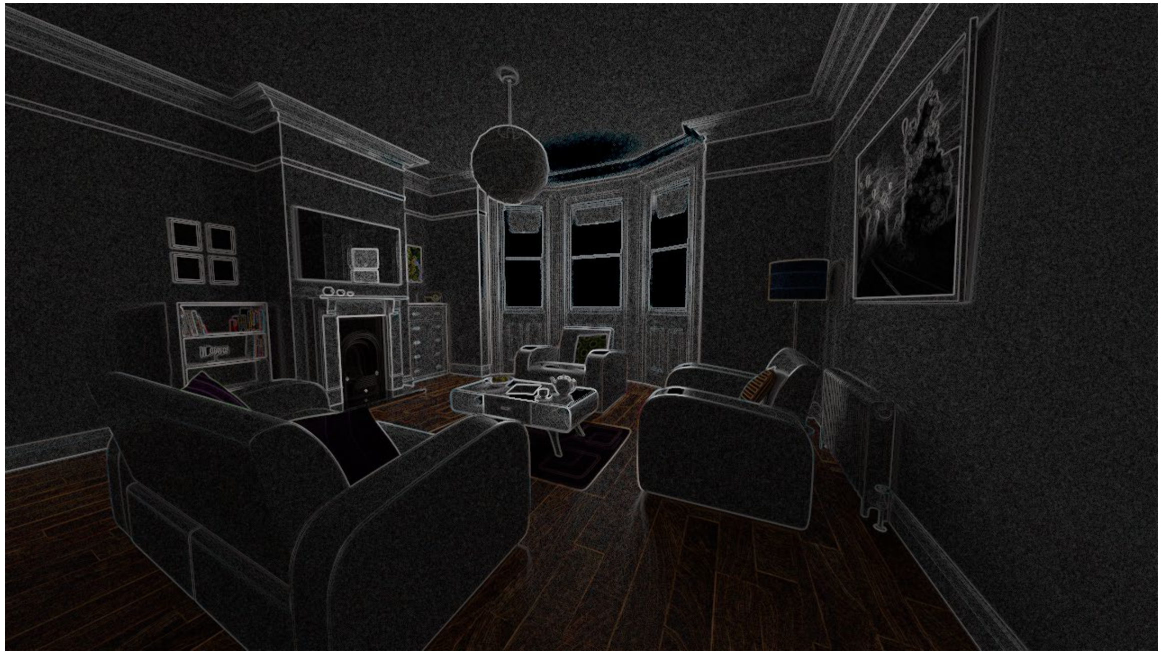
integration (e.g., Poisson solver)

Rendering

Primal-domain rendering: simulate intensities directly



Gradient-domain rendering: simulate gradients, then solve Poisson problem

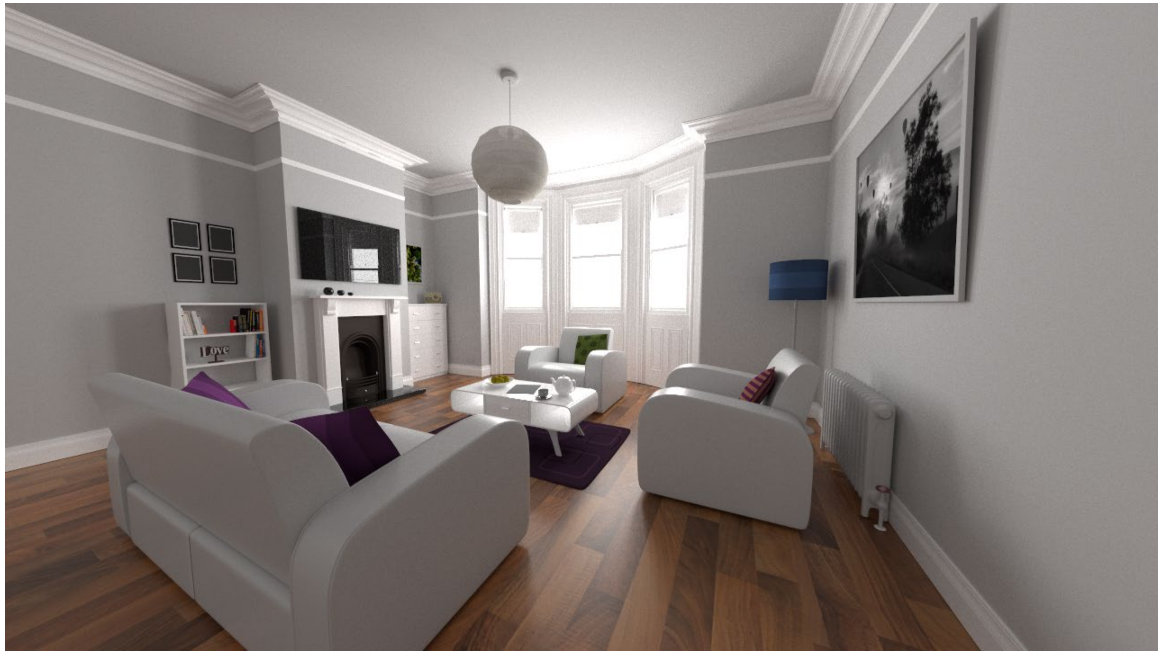


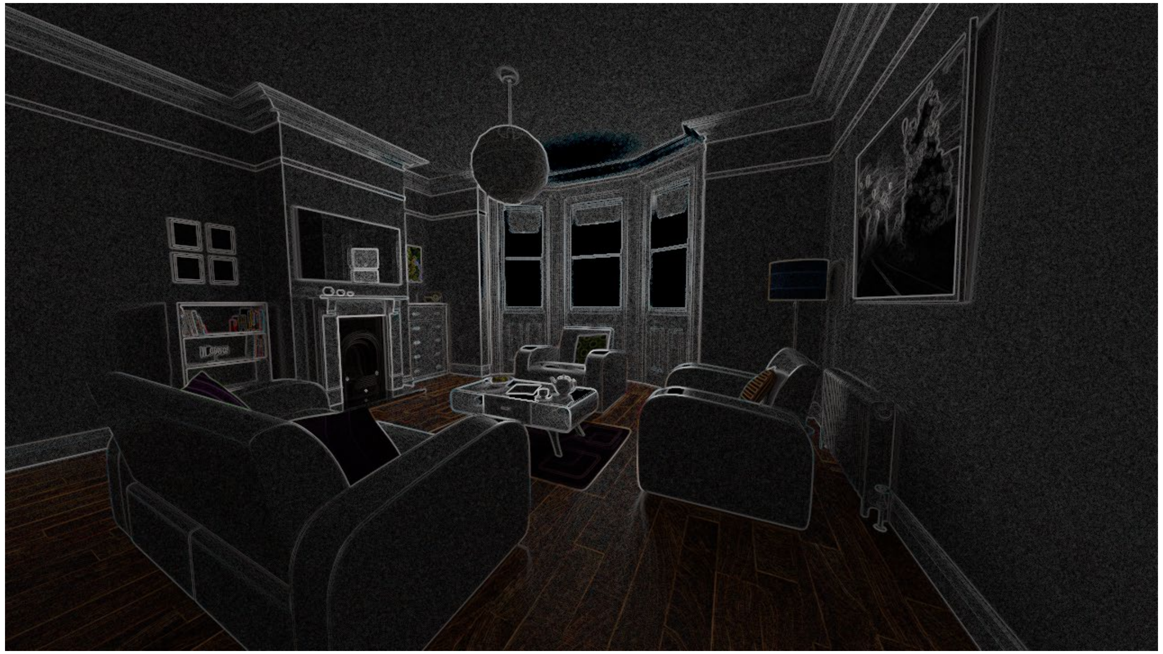
Why would gradient-domain rendering make sense?

Rendering

Primal-domain rendering: simulate intensities directly

Gradient-domain rendering: simulate gradients, then solve Poisson problem





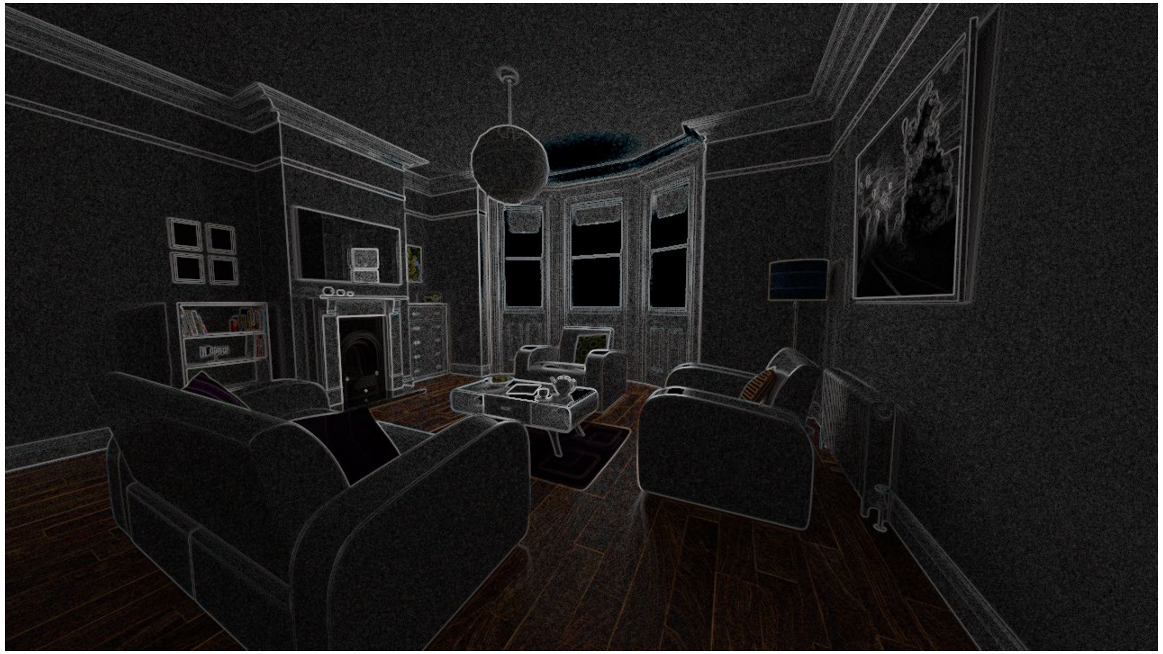
Why would gradient-domain rendering make sense?

- Since gradients are sparse, I can focus most (but not all of) my resources (i.e., ray samples) on rendering the few pixels that are non-zero in gradient space, with much lower variance.
- Poisson reconstruction performs a form of “filtering” to further reduce variance.

Rendering

Primal-domain rendering: simulate intensities directly

Gradient-domain rendering: simulate gradients, then solve Poisson problem



Why would gradient-domain rendering make sense?

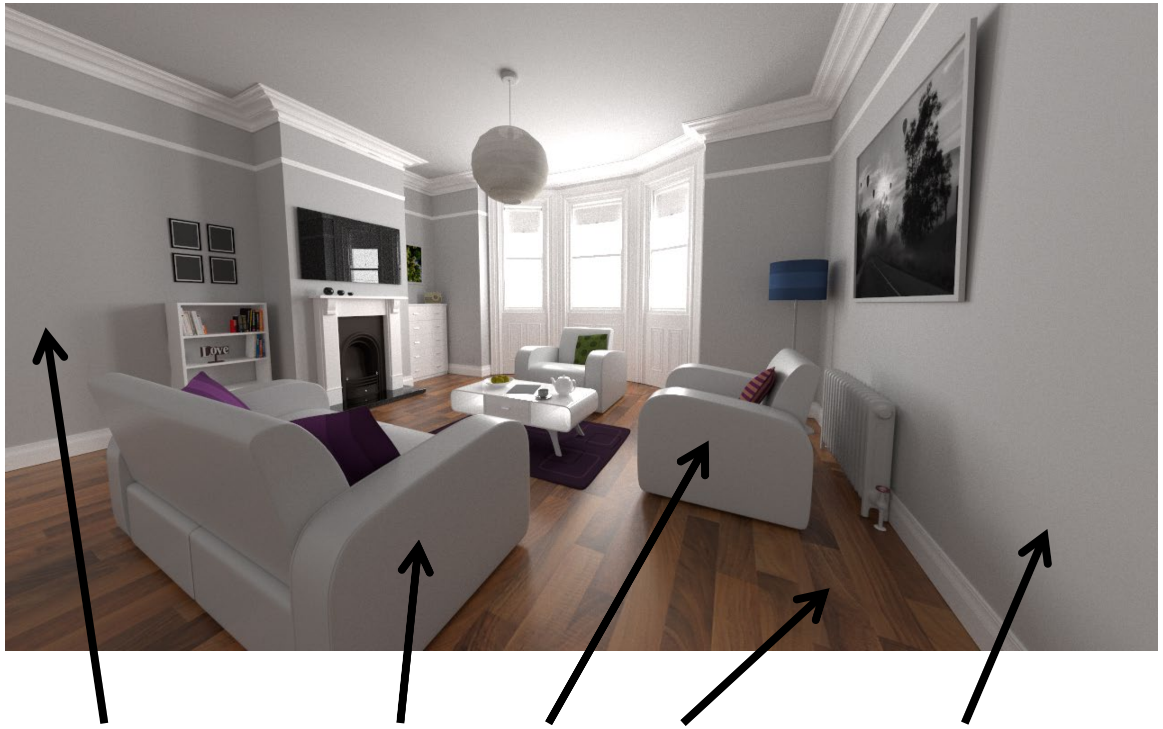
Why not all?

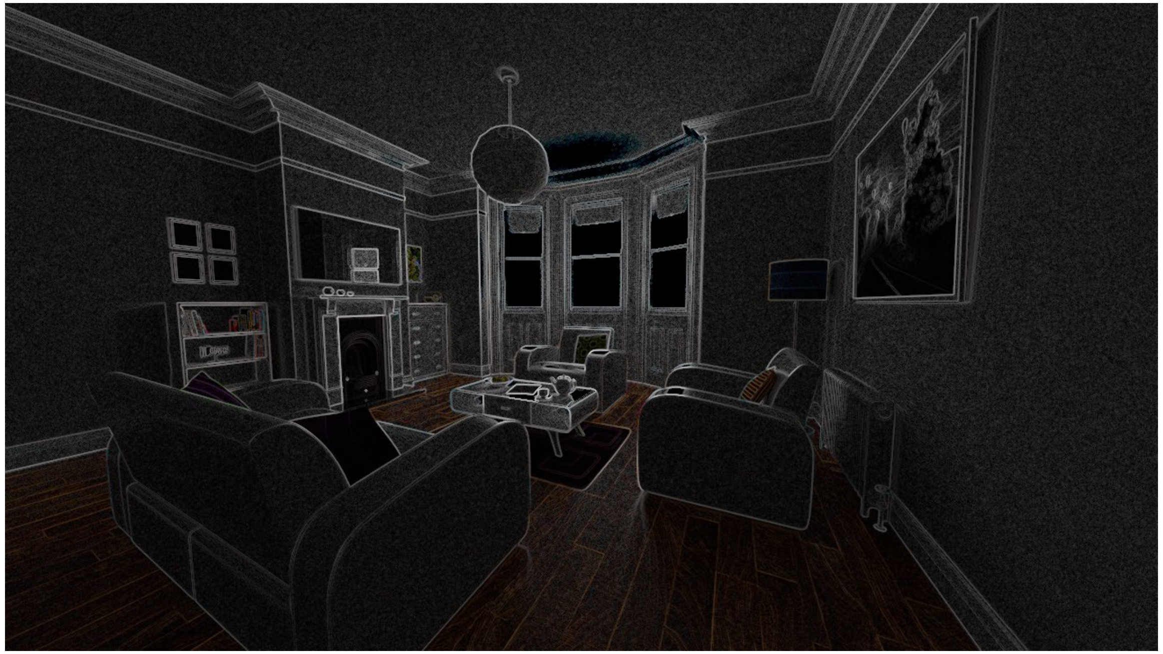
- Since gradients are sparse, I can focus most (but not all of) my resources (i.e., ray samples) on rendering the few pixels that are non-zero in gradient space, with much lower variance.
- Poisson reconstruction performs a form of “filtering” to further reduce variance.

Rendering

Primal-domain rendering: simulate intensities directly

Gradient-domain rendering: simulate gradients, then solve Poisson problem

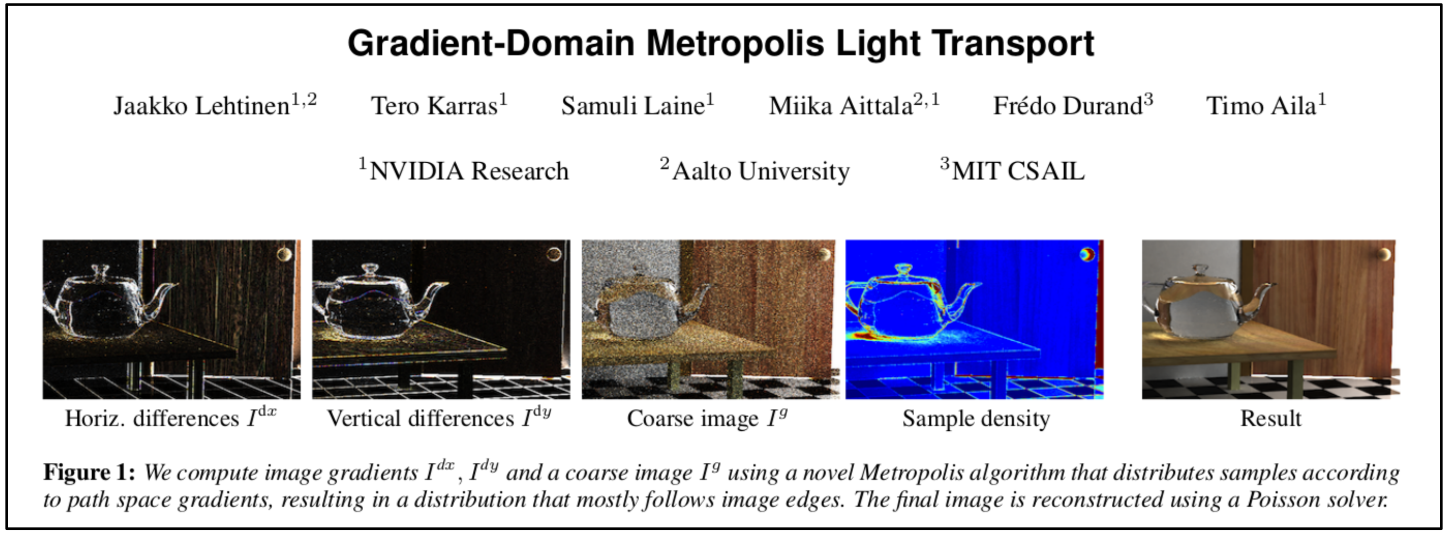




You still need to render a few sparse pixels (roughly one per “flat” region in the image) in primal domain, to use as boundary conditions in the Poisson solver.

- In practice, do image-space stratified sampling to select these pixels.

Gradient-domain rendering



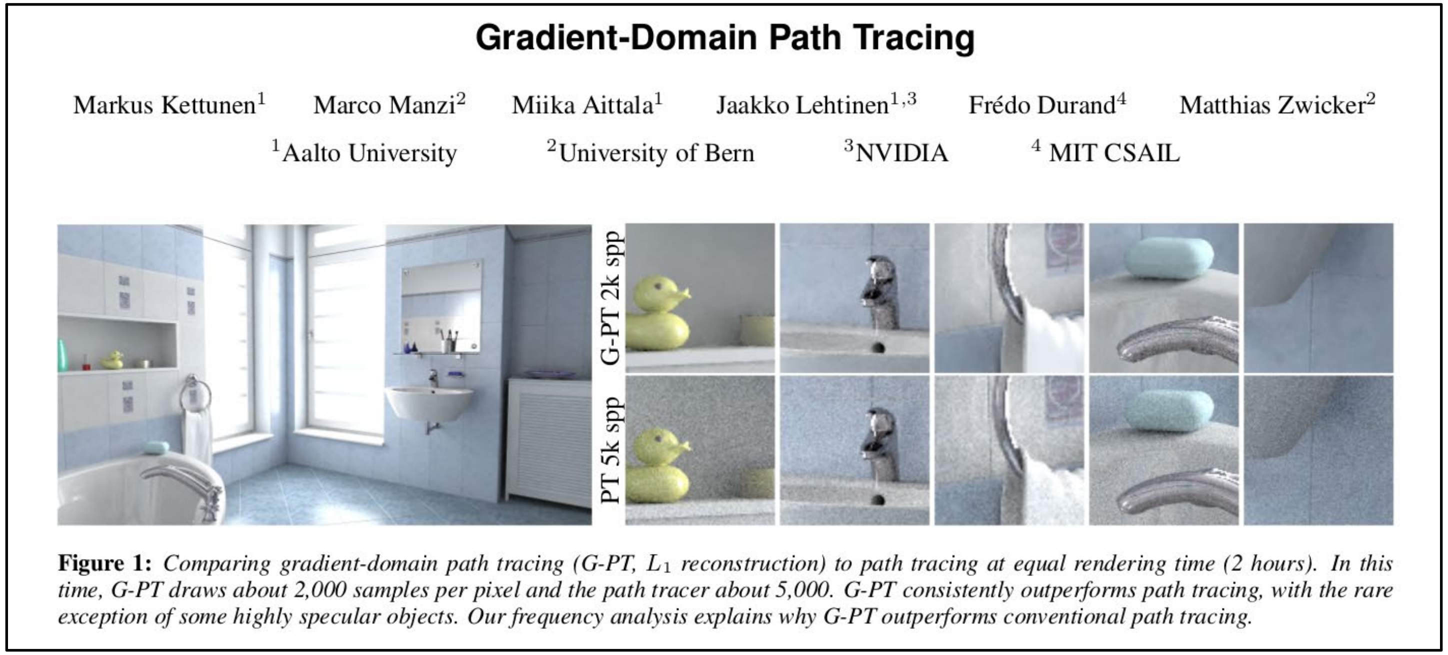
Gradient-Domain Metropolis Light Transport

Jaakko Lehtinen[1,2] Tero Karras[1] Samuli Laine[1] Miika Aittala[2,1] Frédo Durand[3] Timo Aila[1]

[1]NVIDIA Research [2]Aalto University [3]MIT CSAIL

Horiz. differences I^{dx} Vertical differences I^{dy} Coarse image I^{g} Sample density Result

Figure 1: *We compute image gradients I^{dx}, I^{dy} and a coarse image I^{g} using a novel Metropolis algorithm that distributes samples according to path space gradients, resulting in a distribution that mostly follows image edges. The final image is reconstructed using a Poisson solver.*



Gradient-Domain Path Tracing

Markus Kettunen[1] Marco Manzi[2] Miika Aittala[1] Jaakko Lehtinen[1,3] Frédo Durand[4] Matthias Zwicker[2]

[1]Aalto University [2]University of Bern [3]NVIDIA [4] MIT CSAIL

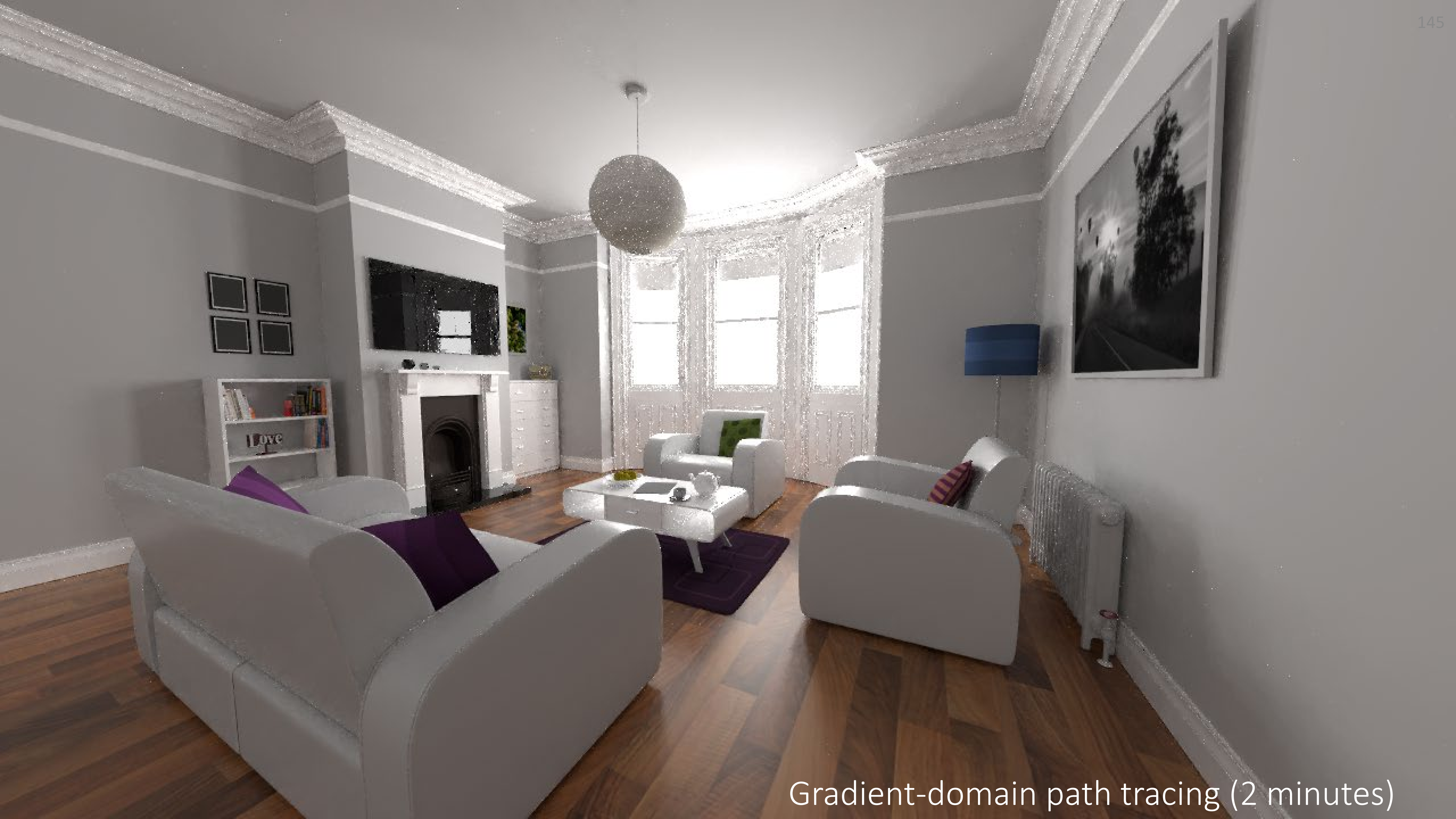
G-PT 2k spp

PT 5k spp

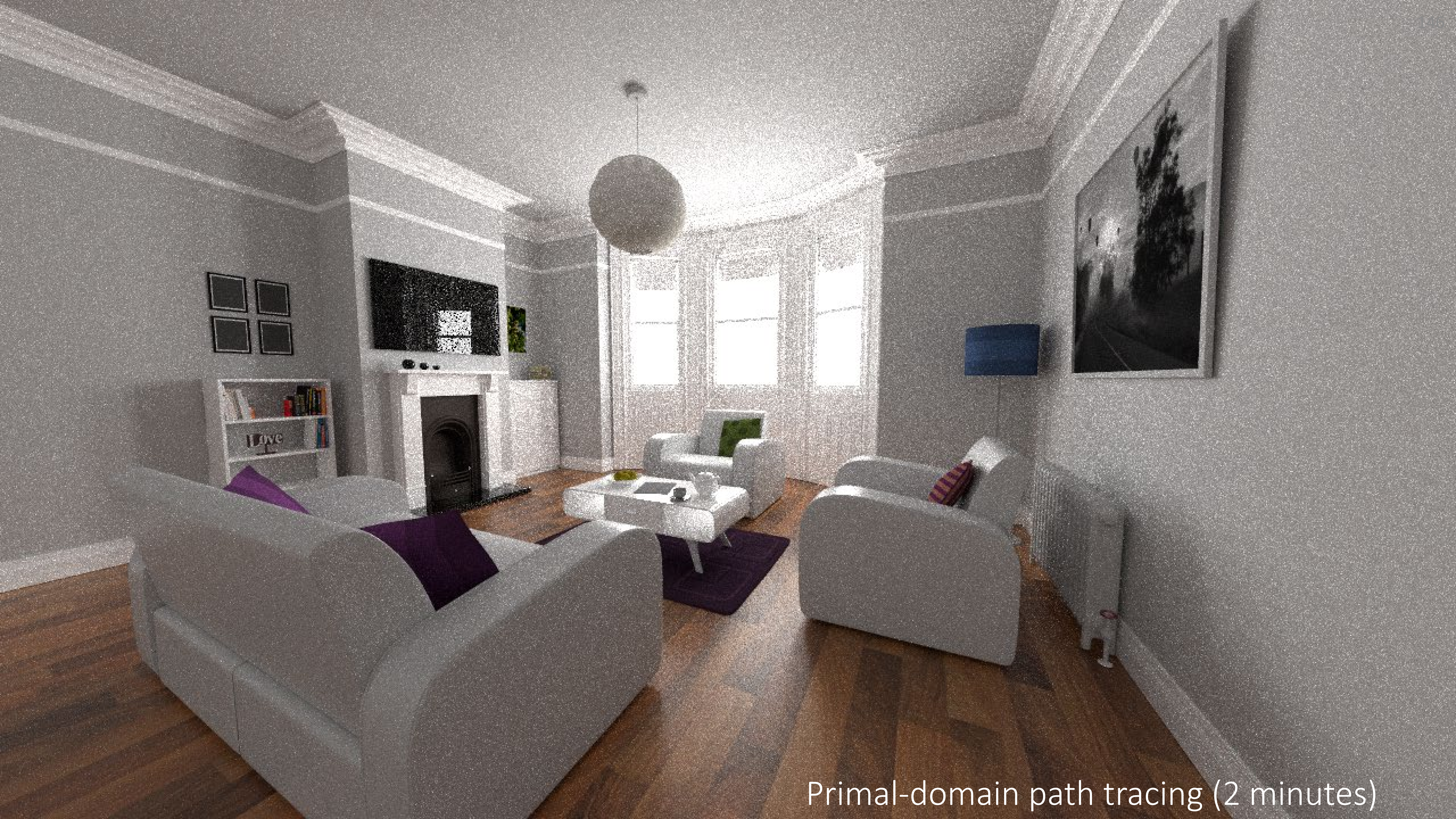
Figure 1: *Comparing gradient-domain path tracing (G-PT, L_1 reconstruction) to path tracing at equal rendering time (2 hours). In this time, G-PT draws about 2,000 samples per pixel and the path tracer about 5,000. G-PT consistently outperforms path tracing, with the rare exception of some highly specular objects. Our frequency analysis explains why G-PT outperforms conventional path tracing.*

A lot of papers since SIGGRAPH 2013 (first introduction of gradient-domain rendering) that are looking to extend basically all primal-domain rendering algorithms to the gradient domain.

Does it help?

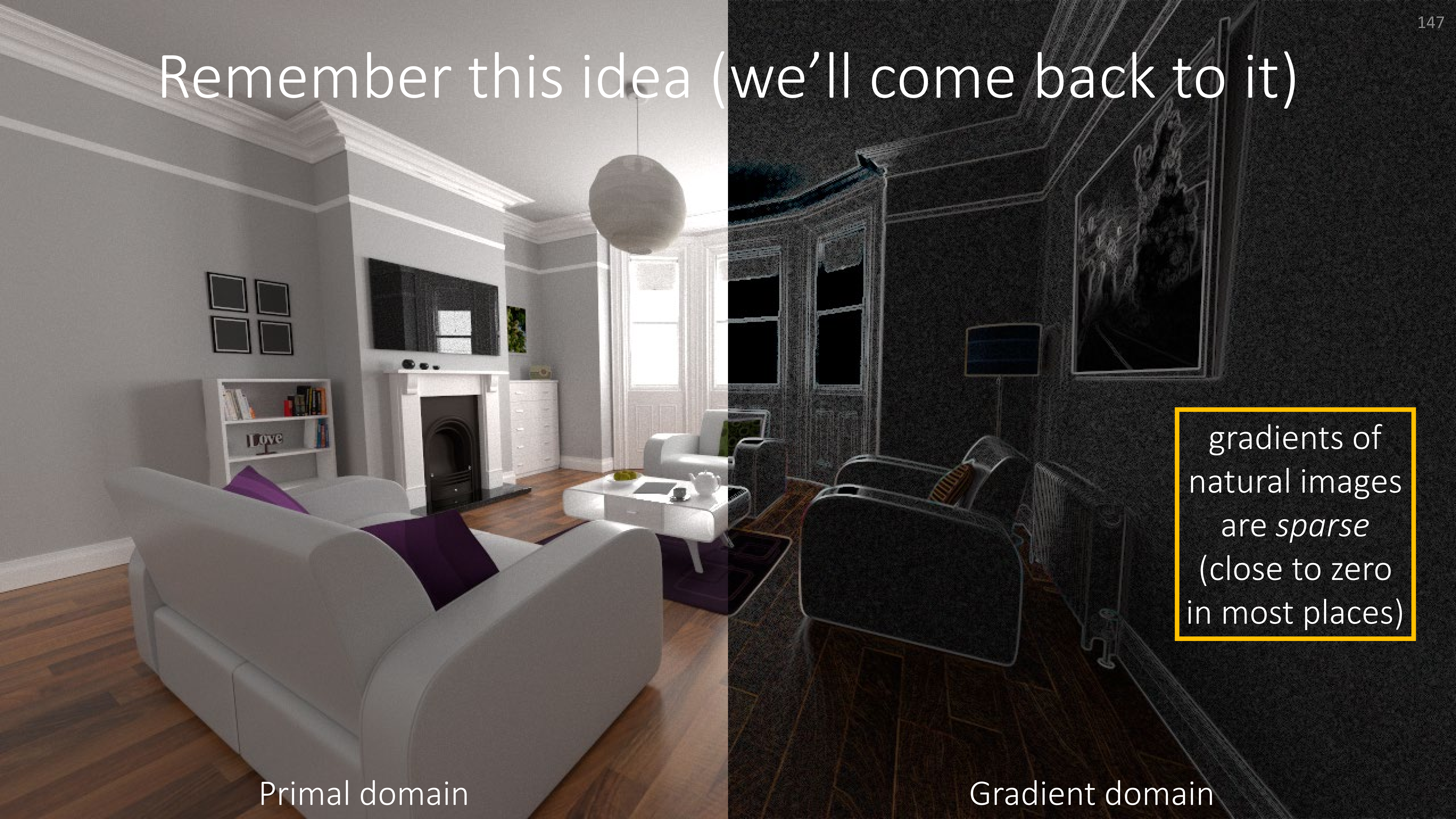


Gradient-domain path tracing (2 minutes)



Primal-domain path tracing (2 minutes)

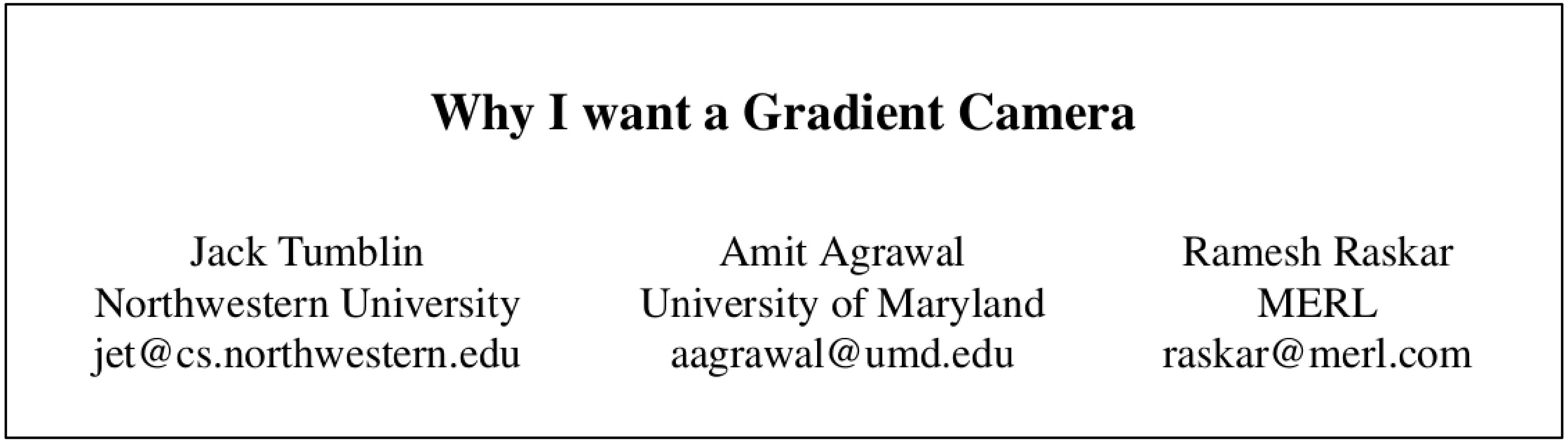
Remember this idea (we’ll come back to it)



gradients of natural images are *sparse* (close to zero in most places)

Gradient cameras

One of my favorite papers



Why I want a Gradient Camera

Jack Tumblin
Northwestern University
jet@cs.northwestern.edu

Amit Agrawal
University of Maryland
aagrawal@umd.edu

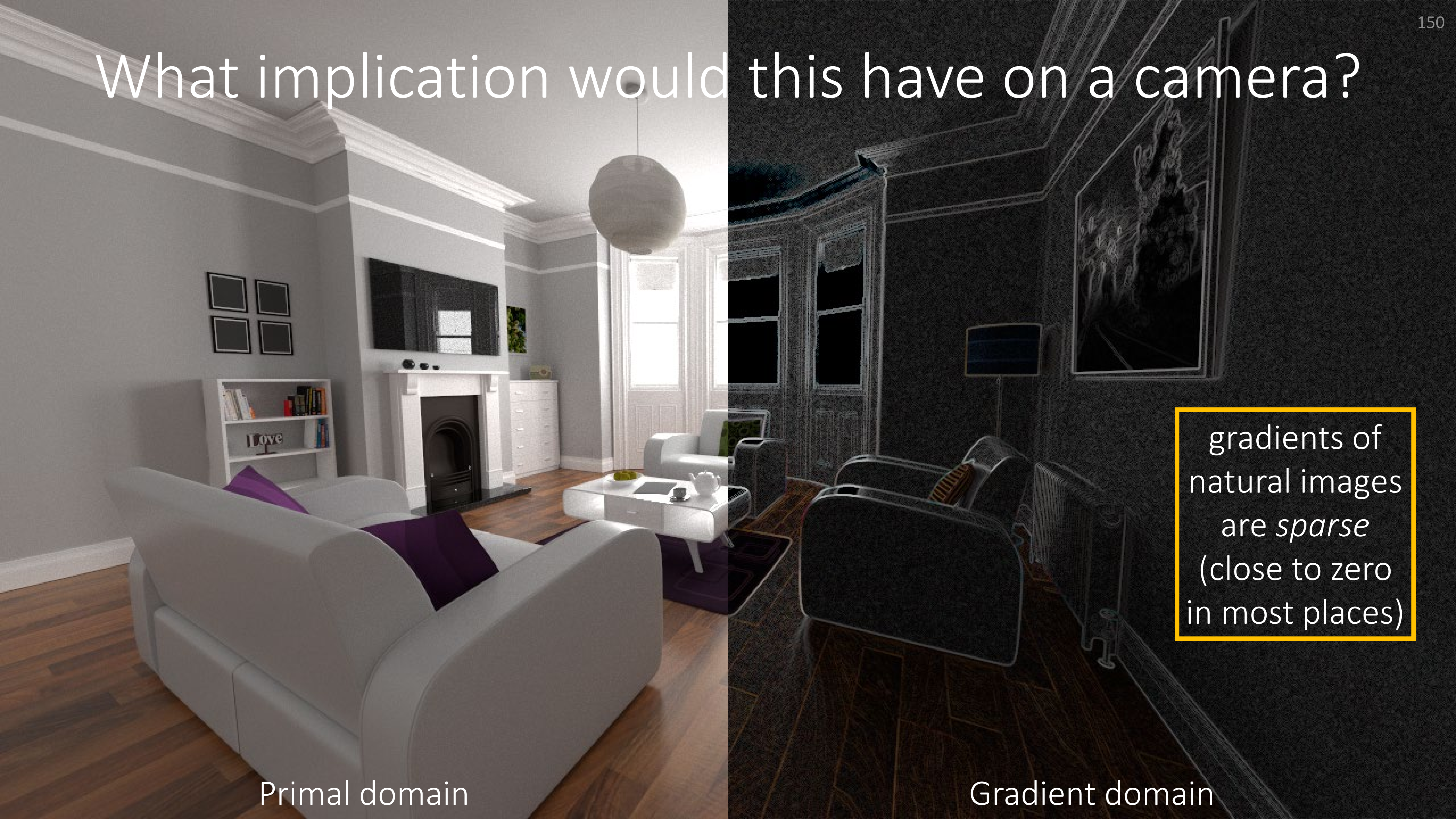
Ramesh Raskar
MERL
raskar@merl.com

Why would you want a gradient camera?

Can you directly display the measurements of such a camera?

How would you build a gradient camera?

What implication would this have on a camera?

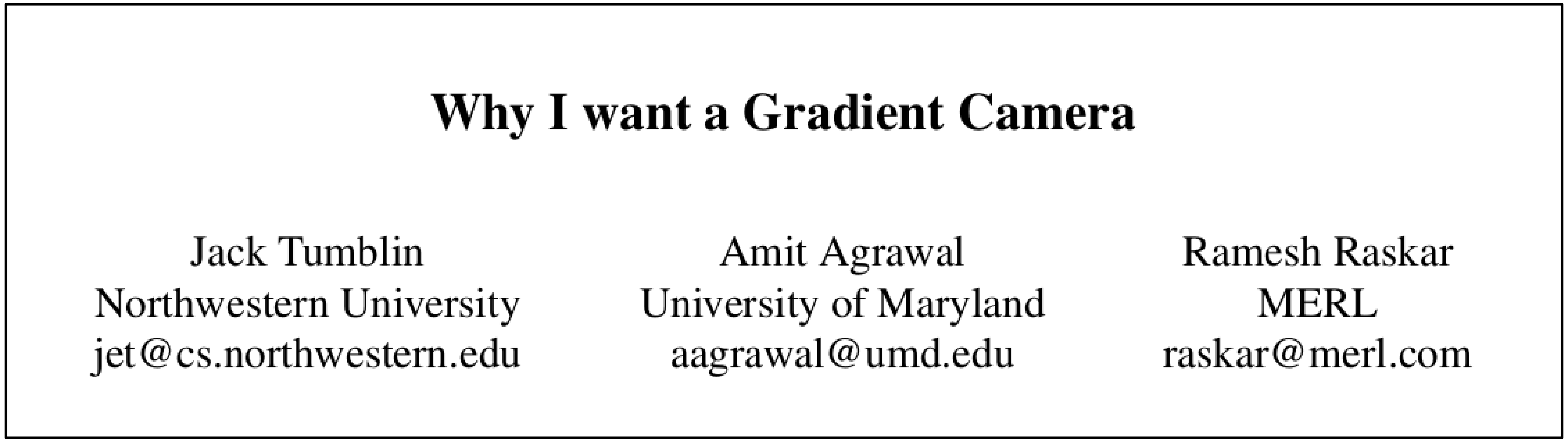


Primal domain

Gradient domain

gradients of natural images are *sparse* (close to zero in most places)

One of my favorite papers



Why I want a Gradient Camera

Jack Tumblin
Northwestern University
jet@cs.northwestern.edu

Amit Agrawal
University of Maryland
aagrawal@umd.edu

Ramesh Raskar
MERL
raskar@merl.com

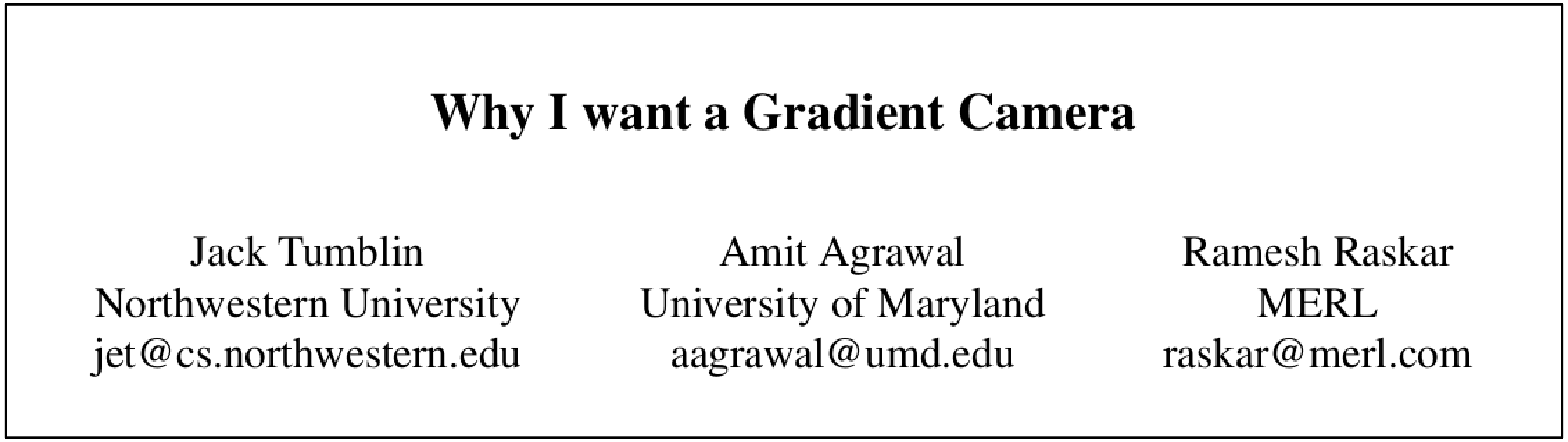
Why would you want a gradient camera?

- Much faster frame rate, as you only read out very few pixels (where gradient is significant).
- Much higher dynamic range, if also combined with logarithmic gradients.

Can you directly display the measurements of such a camera?

How would you build a gradient camera?

One of my favorite papers



Why I want a Gradient Camera

Jack Tumblin
Northwestern University
jet@cs.northwestern.edu

Amit Agrawal
University of Maryland
aagrawal@umd.edu

Ramesh Raskar
MERL
raskar@merl.com

Why would you want a gradient camera?

- Much faster frame rate, as you only read out very few pixels (where gradient is significant).
- Much higher dynamic range, if also combined with logarithmic gradients.

Can you directly display the measurements of such a camera?

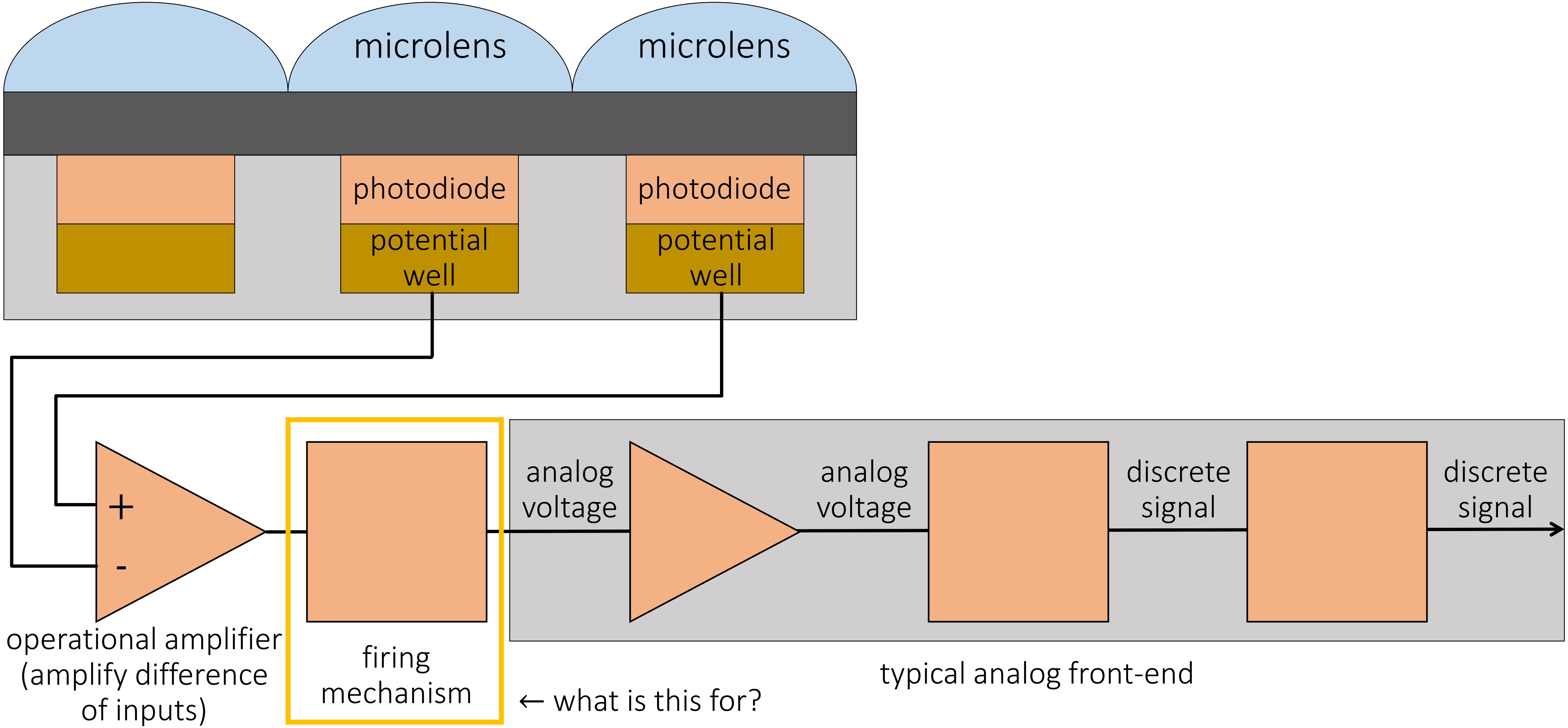
- You need to use a Poisson solver to reconstruct the image from the measured gradients.

How would you build a gradient camera?

Change the sensor

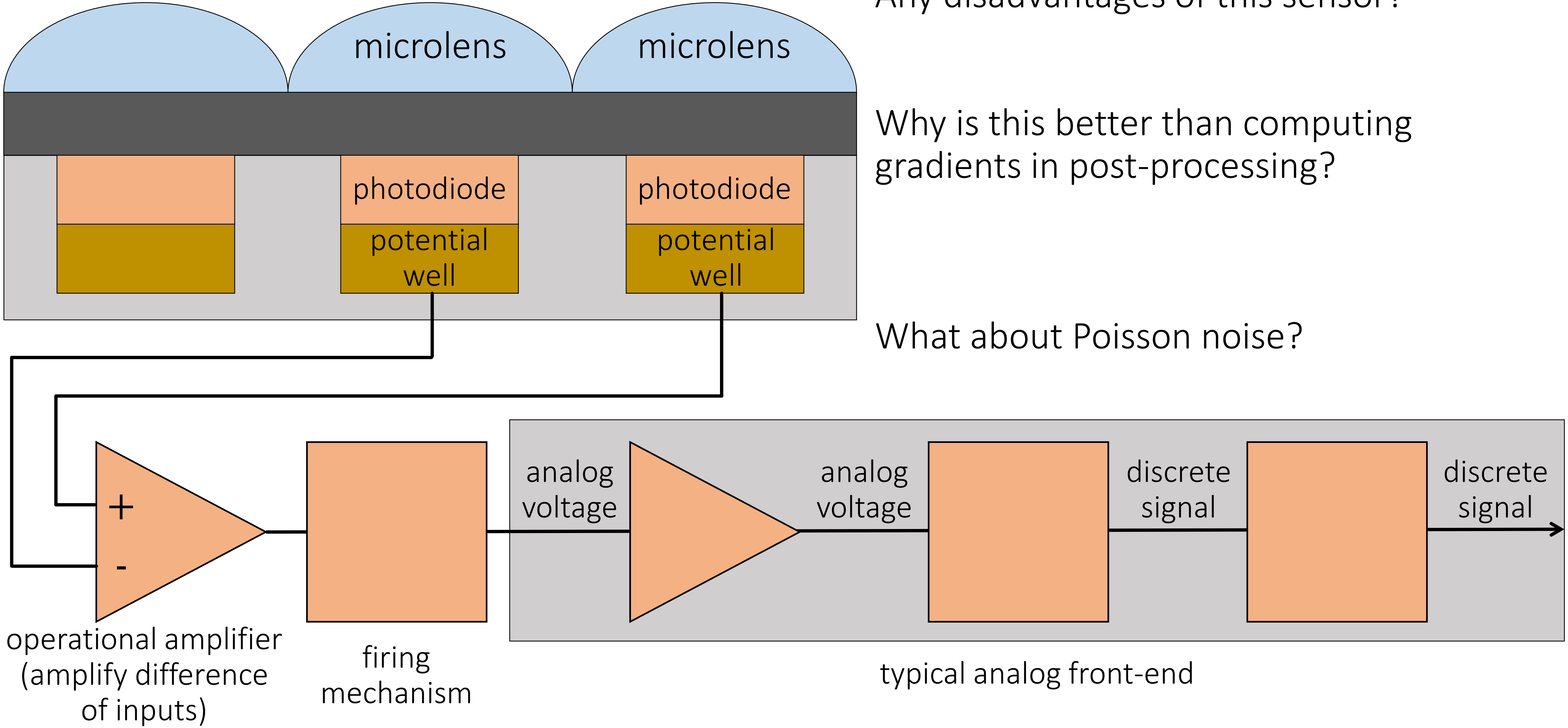
Can you think how?

Change the sensor



← what is this for?

Change the sensor

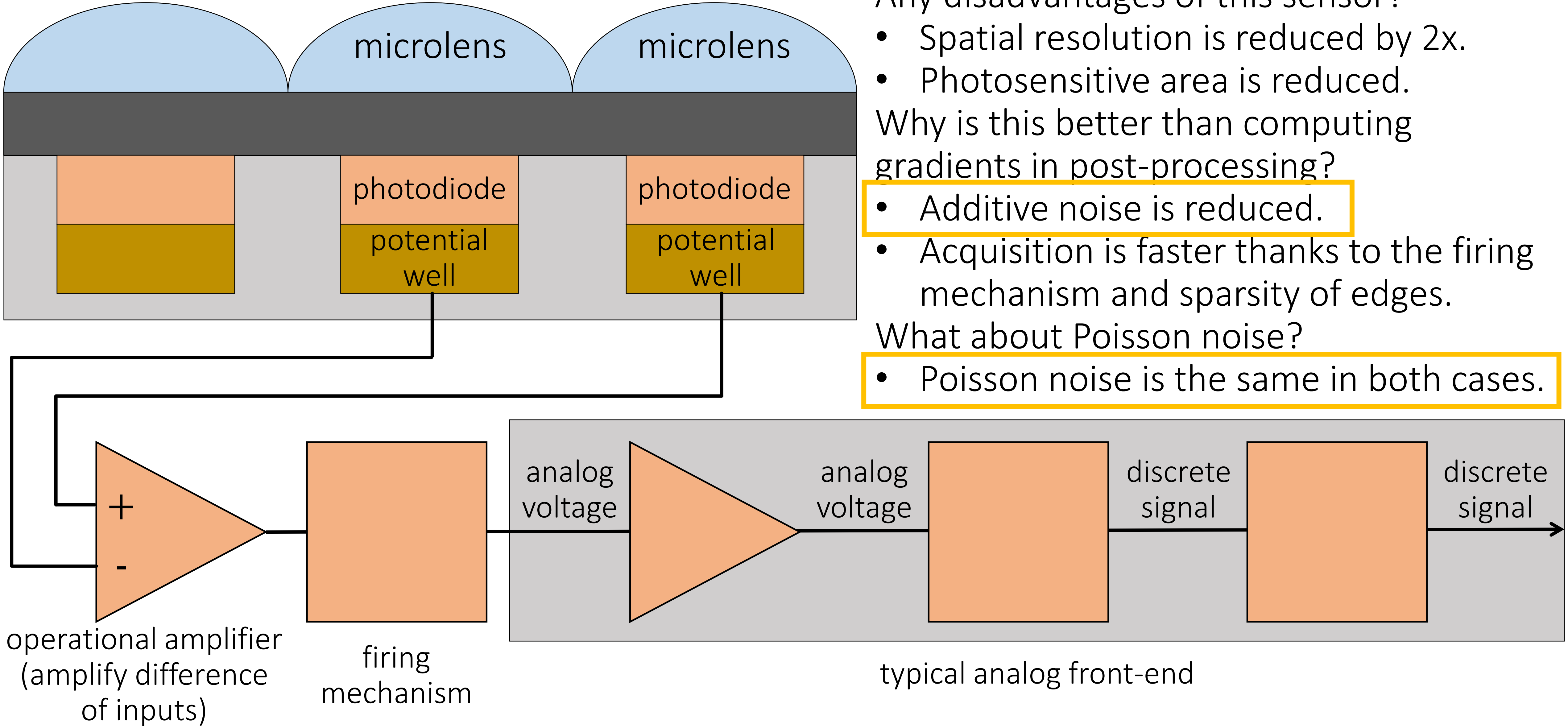


Any disadvantages of this sensor?

Why is this better than computing gradients in post-processing?

What about Poisson noise?

Change the sensor



Any disadvantages of this sensor?

- Spatial resolution is reduced by 2x.
- Photosensitive area is reduced.

Why is this better than computing gradients in post-processing?

- Additive noise is reduced.
- Acquisition is faster thanks to the firing mechanism and sparsity of edges.

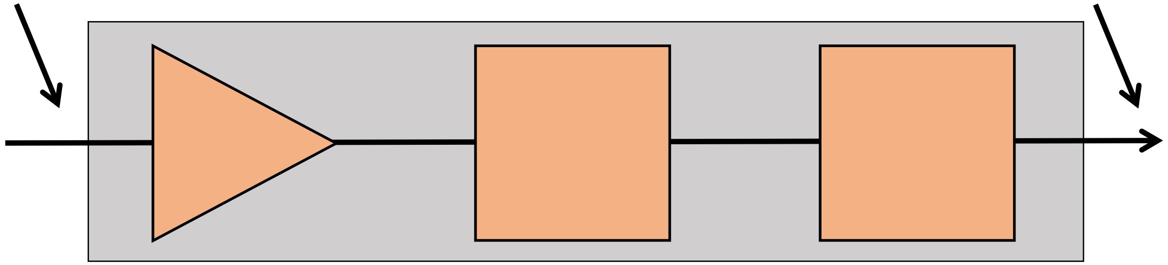
What about Poisson noise?

- Poisson noise is the same in both cases.

Noise considerations

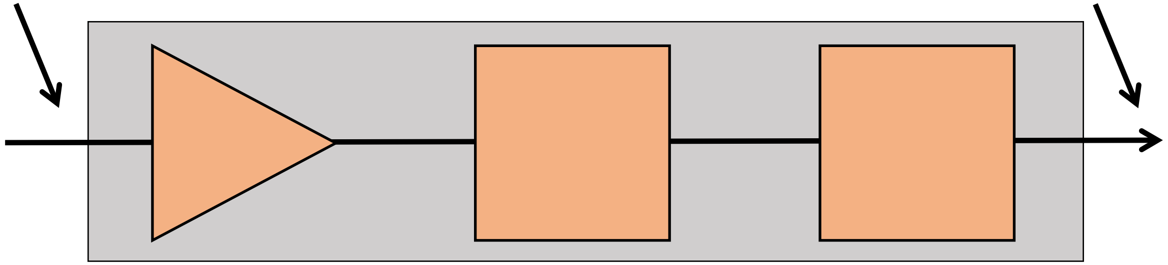
L_1

$I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



L_2

$I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



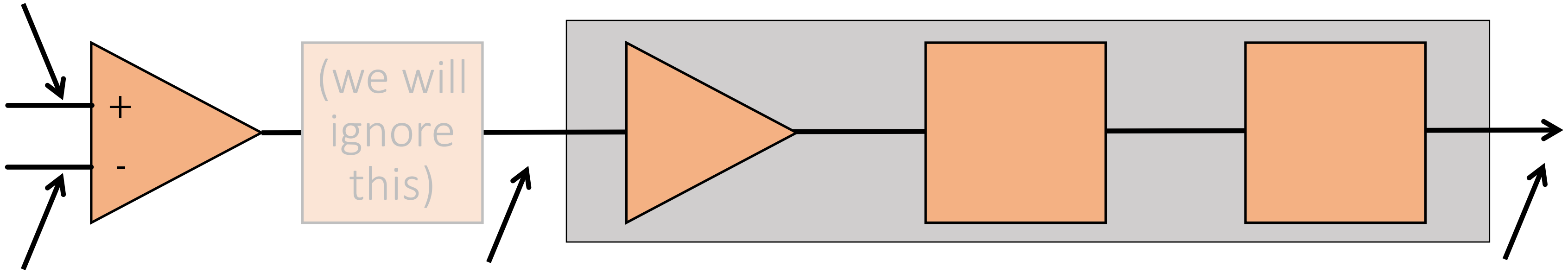
Digital subtraction in post-processing

$$\sigma(I_1 - I_2)^2 = ?$$

Analog subtraction on sensor

$$\sigma(I)^2 = ?$$

L_1



L_2

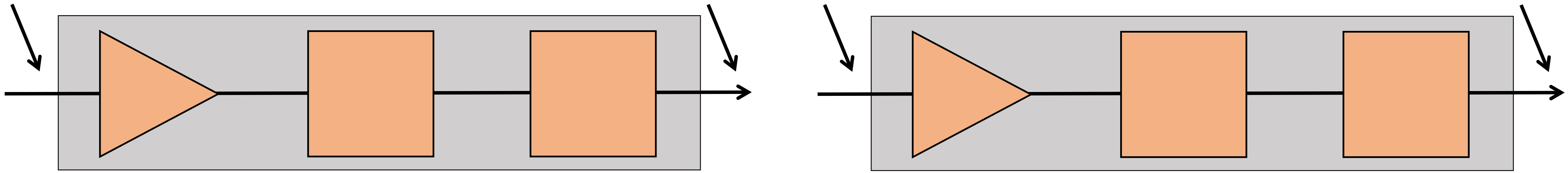
$D = (L_1 - L_2) + n_{\text{opamp}}$

$I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$

$L_1 \sim \text{Poisson}(t \cdot (a \cdot \Phi_1 + D))$
$L_2 \sim \text{Poisson}(t \cdot (a \cdot \Phi_2 + D))$
$n_{\text{opamp}} \sim \text{Normal}(0, \sigma_{\text{opamp}})$
$n_{\text{read}} \sim \text{Normal}(0, \sigma_{\text{read}})$
$n_{\text{ADC}} \sim \text{Normal}(0, \sigma_{\text{ADC}})$

Noise considerations

L_1 $\quad$ $I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$ $\quad$ L_2 $\quad$ $I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



Digital subtraction in post-processing

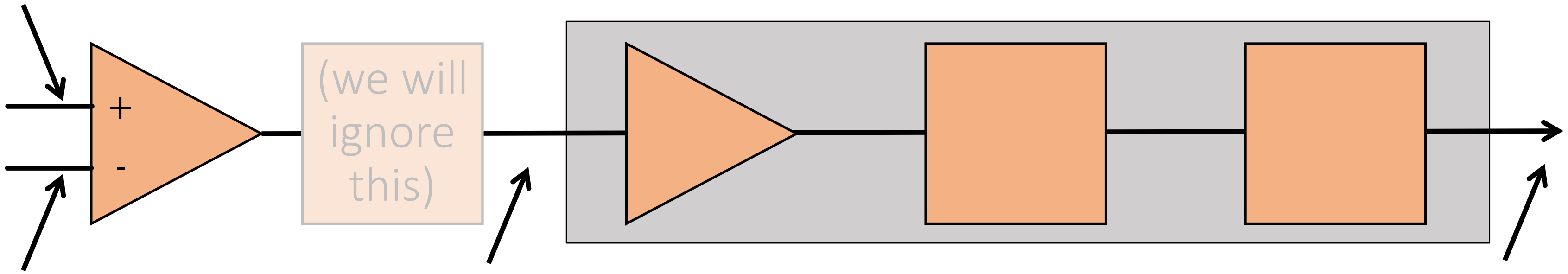
$$\sigma(I_1 - I_2)^2 = \sigma(L_1 - L_2)^2 + 2 \cdot \sigma_{\text{read}}^2 \cdot g^2 + 2 \cdot \sigma_{\text{ADC}}^2$$

which variance is better?

Analog subtraction on sensor

$$\sigma(I)^2 = \sigma(L_1 - L_2)^2 + \sigma_{\text{opamp}}^2 \cdot g^2 + \sigma_{\text{read}}^2 \cdot g^2 + \sigma_{\text{ADC}}^2$$

L_1



$L_1 \sim \text{Poisson}(t \cdot (a \cdot \Phi_1 + D))$

$L_2 \sim \text{Poisson}(t \cdot (a \cdot \Phi_2 + D))$

$n_{\text{opamp}} \sim \text{Normal}(0, \sigma_{\text{opamp}})$

$n_{\text{read}} \sim \text{Normal}(0, \sigma_{\text{read}})$

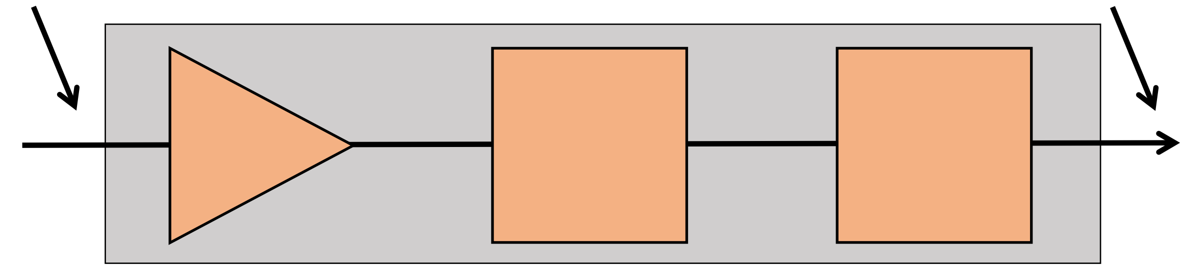
$n_{\text{ADC}} \sim \text{Normal}(0, \sigma_{\text{ADC}})$

L_2 $\quad$ $D = (L_1 - L_2) + n_{\text{opamp}}$ $\quad$ $I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$

Noise considerations

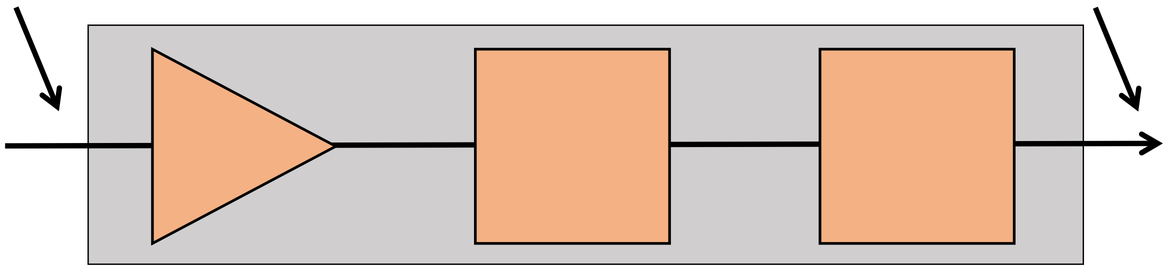
L_1

$I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



L_2

$I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



Digital subtraction in post-processing

$$\sigma(I_1 - I_2)^2 = \boxed{\sigma(L_1 - L_2)^2} + \boxed{2 \cdot \sigma_{\text{read}}^2 \cdot g^2 + 2 \cdot \sigma_{\text{ADC}}^2}$$

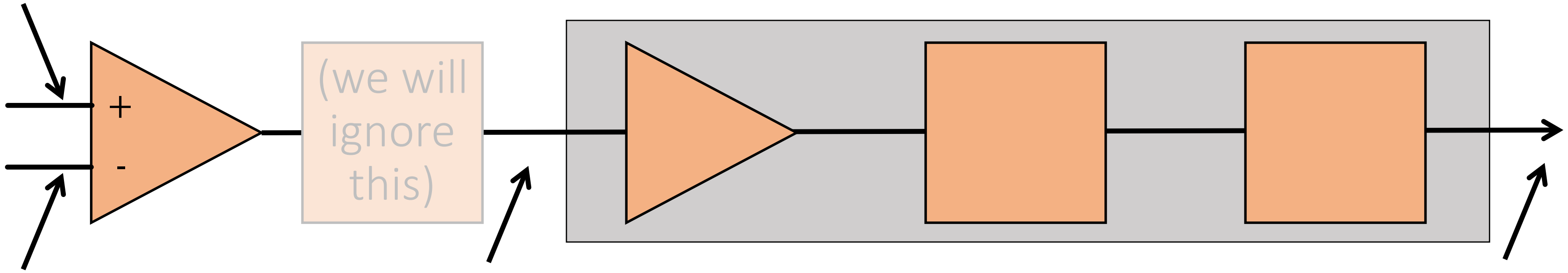
terms related to Poisson noise are the same

additive noise is reduced if opamp is well-designed

Analog subtraction on sensor

$$\sigma(I)^2 = \boxed{\sigma(L_1 - L_2)^2} + \boxed{\sigma_{\text{opamp}}^2 \cdot g^2 + \sigma_{\text{read}}^2 \cdot g^2 + \sigma_{\text{ADC}}^2}$$

L_1



L_2

$D = (L_1 - L_2) + n_{\text{opamp}}$

$I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$

$L_1 \sim \text{Poisson}(t \cdot (a \cdot \Phi_1 + D))$

$L_2 \sim \text{Poisson}(t \cdot (a \cdot \Phi_2 + D))$

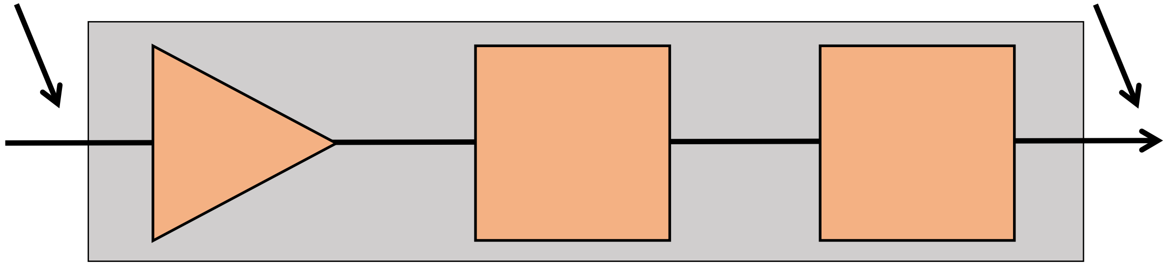
$n_{\text{opamp}} \sim \text{Normal}(0, \sigma_{\text{opamp}})$

$n_{\text{read}} \sim \text{Normal}(0, \sigma_{\text{read}})$

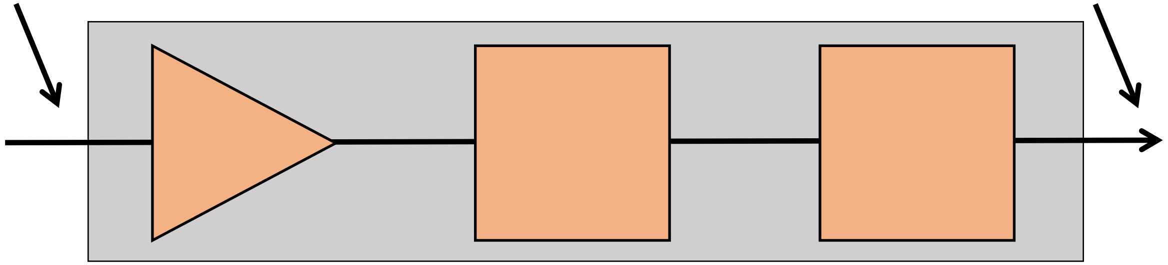
$n_{\text{ADC}} \sim \text{Normal}(0, \sigma_{\text{ADC}})$

Noise considerations

L_1 $\qquad I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



L_2 $\qquad I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



Digital subtraction in post-processing

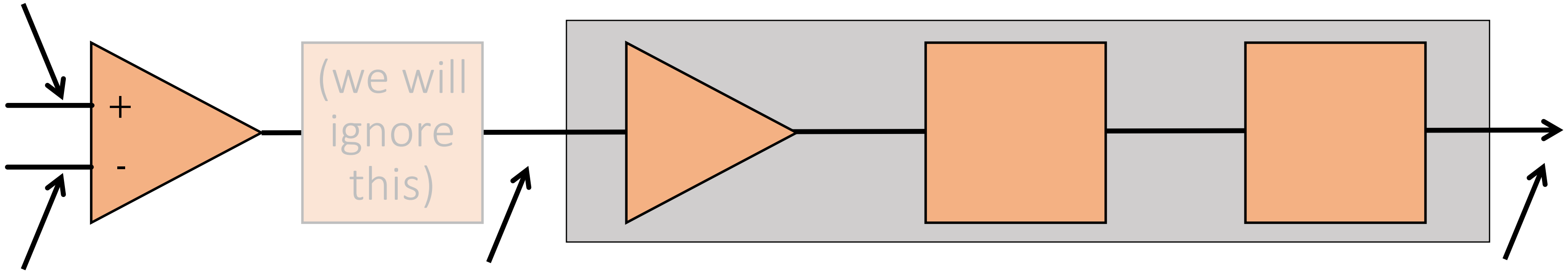
$$\sigma(I_1 - I_2)^2 = \boxed{\sigma(L_1 - L_2)^2} + 2 \cdot \sigma_{\text{read}}^2 \cdot g^2 + 2 \cdot \sigma_{\text{ADC}}^2$$

what is the distribution of the difference $L_1 - L_2$?

$$\sigma(I)^2 = \boxed{\sigma(L_1 - L_2)^2} + \sigma_{\text{opamp}}^2 \cdot g^2 + \sigma_{\text{read}}^2 \cdot g^2 + \sigma_{\text{ADC}}^2$$

Analog subtraction on sensor

L_1



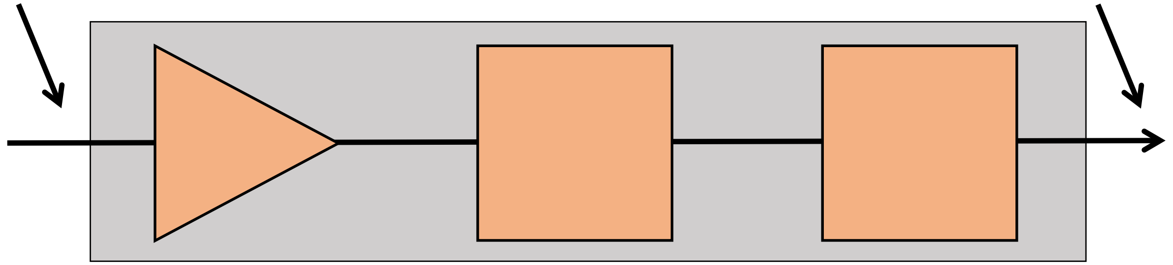
L_2 $\qquad D = (L_1 - L_2) + n_{\text{opamp}}$ $\qquad I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$

$$L_1 \sim \text{Poisson}(t \cdot (a \cdot \Phi_1 + D))$$
$$L_2 \sim \text{Poisson}(t \cdot (a \cdot \Phi_2 + D))$$
$$n_{\text{opamp}} \sim \text{Normal}(0, \sigma_{\text{opamp}})$$
$$n_{\text{read}} \sim \text{Normal}(0, \sigma_{\text{read}})$$
$$n_{\text{ADC}} \sim \text{Normal}(0, \sigma_{\text{ADC}})$$

Noise considerations

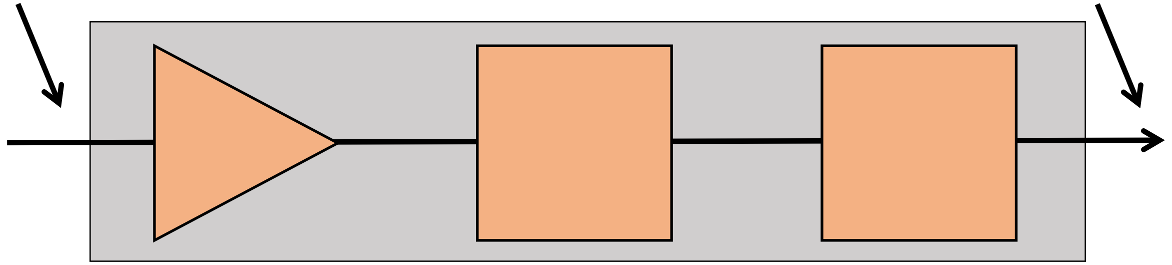
L_1

$I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



L_2

$I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$



Digital subtraction in post-processing

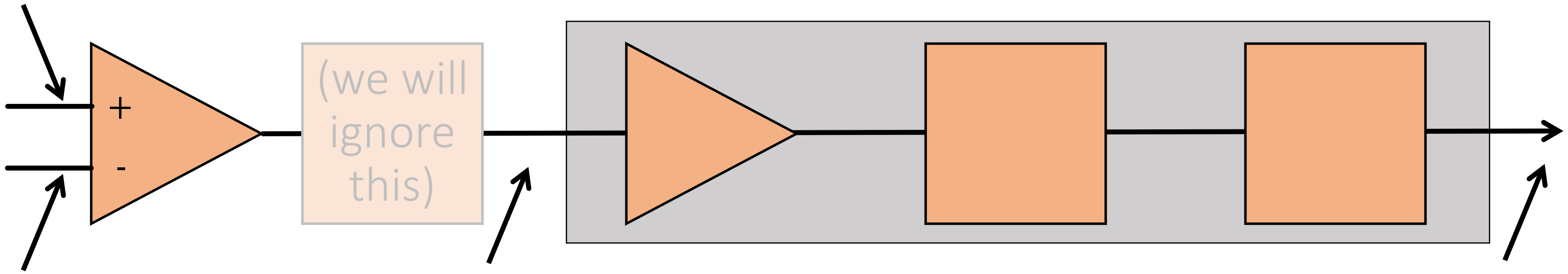
$$\sigma(I_1 - I_2)^2 = \boxed{\sigma(L_1 - L_2)^2} + 2 \cdot \sigma_{\text{read}}^2 \cdot g^2 + 2 \cdot \sigma_{\text{ADC}}^2$$

$$\boxed{L_1 - L_2 \sim \text{Skellam}\big(t \cdot a \cdot (\Phi_1 - \Phi_2), t \cdot (a \cdot (\Phi_1 + \Phi_2) + 2 \cdot D)\big)}$$

Analog subtraction on sensor

$$\sigma(I)^2 = \boxed{\sigma(L_1 - L_2)^2} + \sigma_{\text{opamp}}^2 \cdot g^2 + \sigma_{\text{read}}^2 \cdot g^2 + \sigma_{\text{ADC}}^2$$

L_1



L_2

$D = (L_1 - L_2) + n_{\text{opamp}}$

$I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}$

$L_1 \sim \text{Poisson}(t \cdot (a \cdot \Phi_1 + D))$

$L_2 \sim \text{Poisson}(t \cdot (a \cdot \Phi_2 + D))$

$n_{\text{opamp}} \sim \text{Normal}\big(0, \sigma_{\text{opamp}}\big)$

$n_{\text{read}} \sim \text{Normal}(0, \sigma_{\text{read}})$

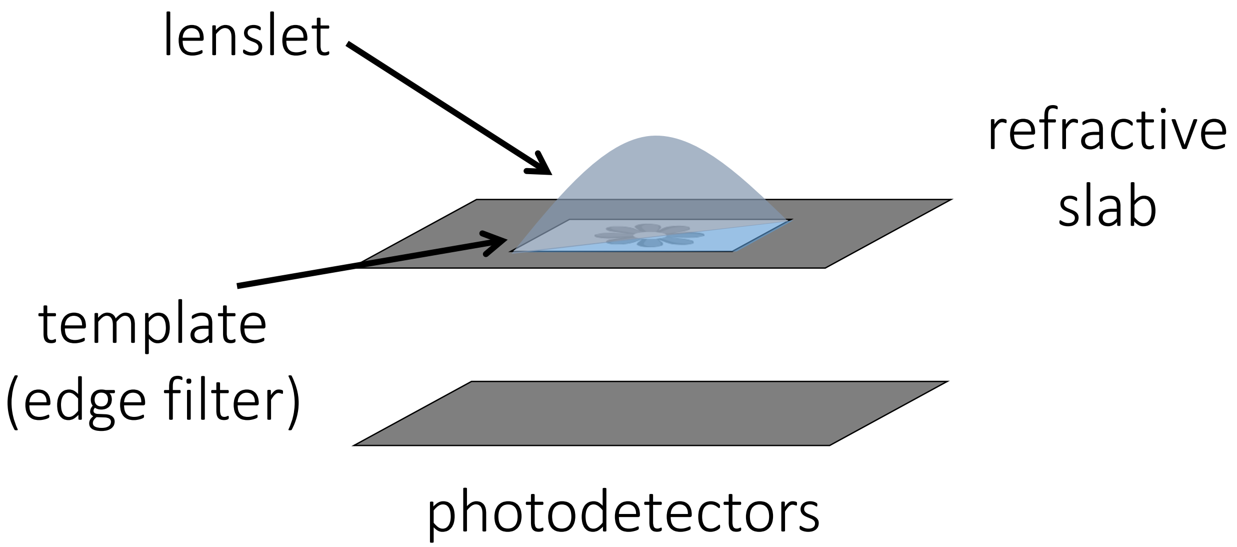
$n_{\text{ADC}} \sim \text{Normal}(0, \sigma_{\text{ADC}})$

Change the optics

Can you think how?

Change the optics

Optical filtering

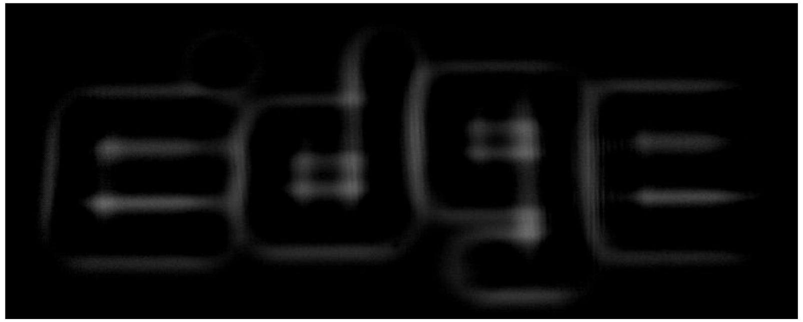


lenslet

refractive slab

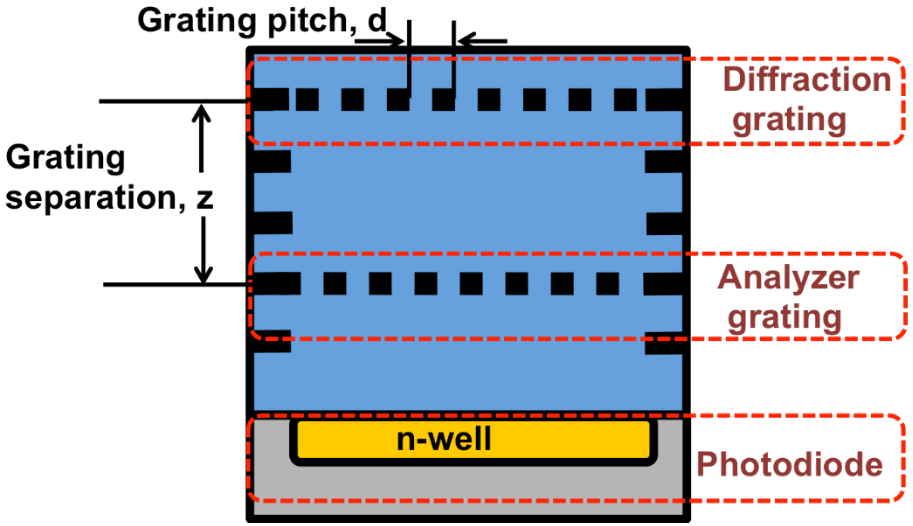
template (edge filter)

photodetectors



resulting image

Angle-sensitive pixels



Grating pitch, d

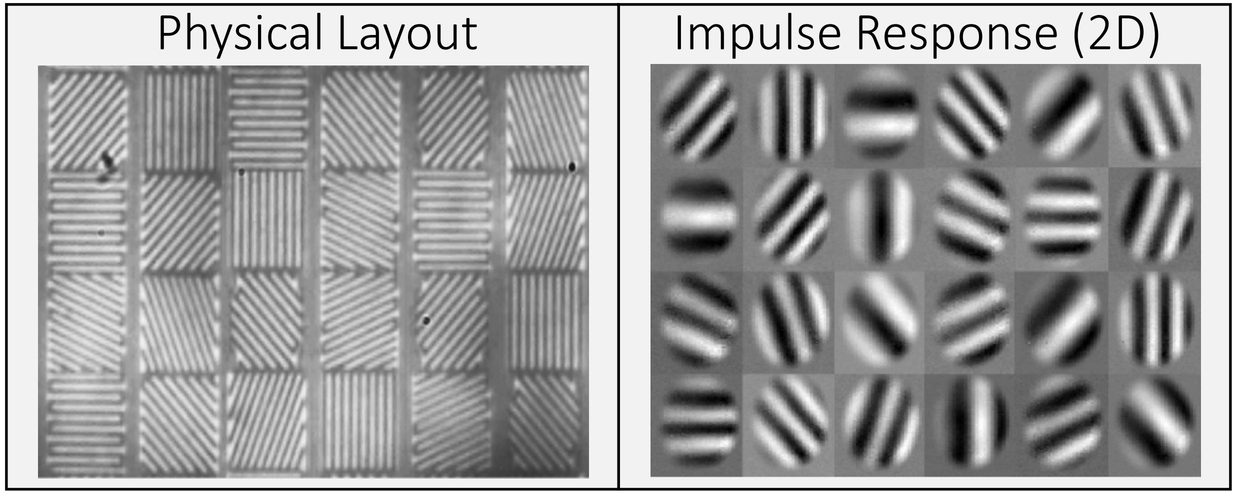
Grating separation, z

Diffraction grating

Analyzer grating

n-well

Photodiode

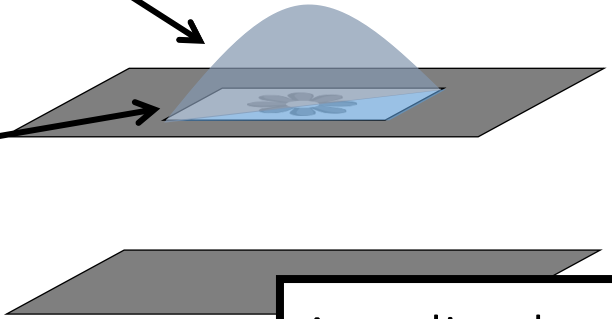


Physical Layout

Impulse Response (2D)

Change the optics

Optical filtering



lenslet

refractive slab

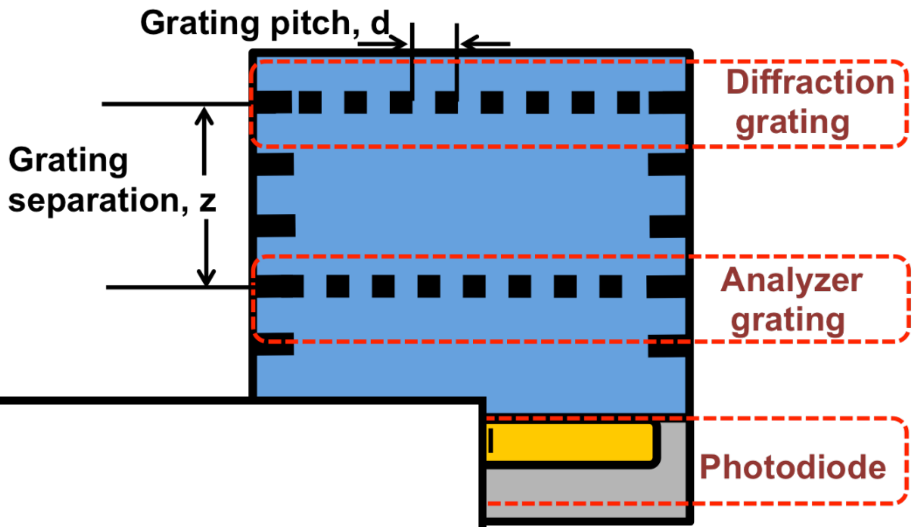
template (edge filter)

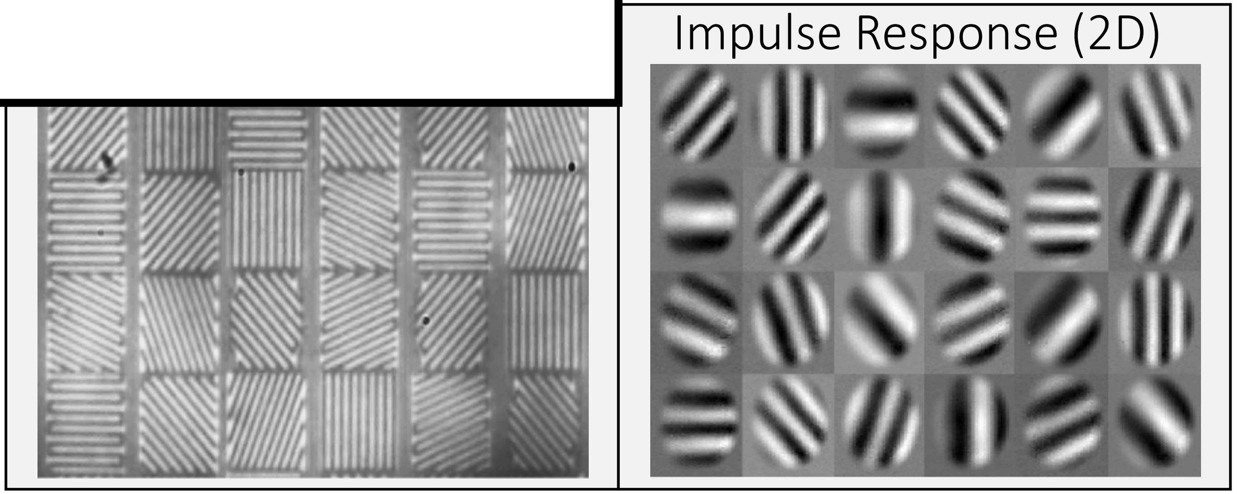
photode

resulting image

Any disadvantages?

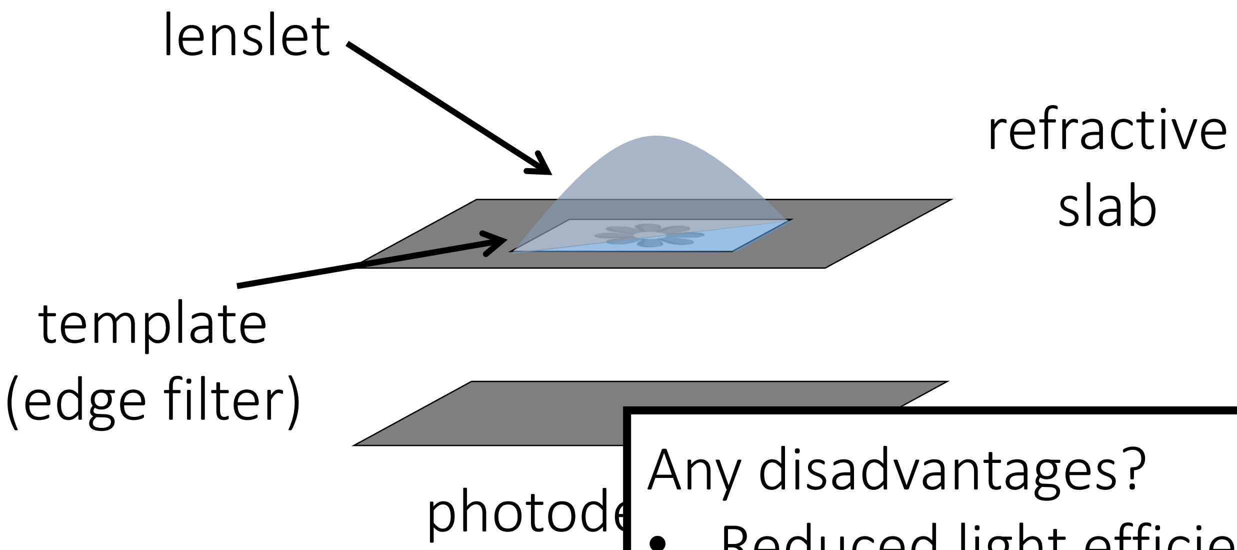
Angle-sensitive pixels

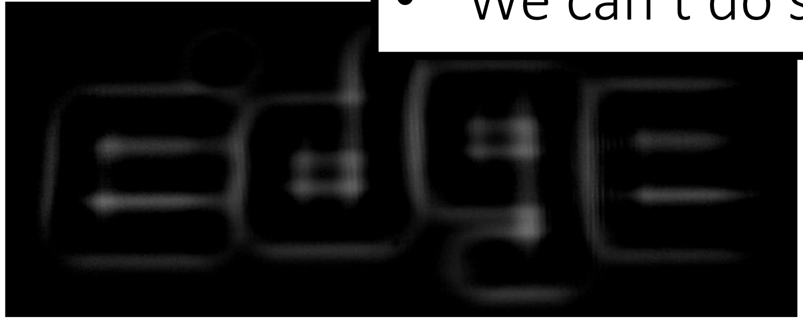




Change the optics

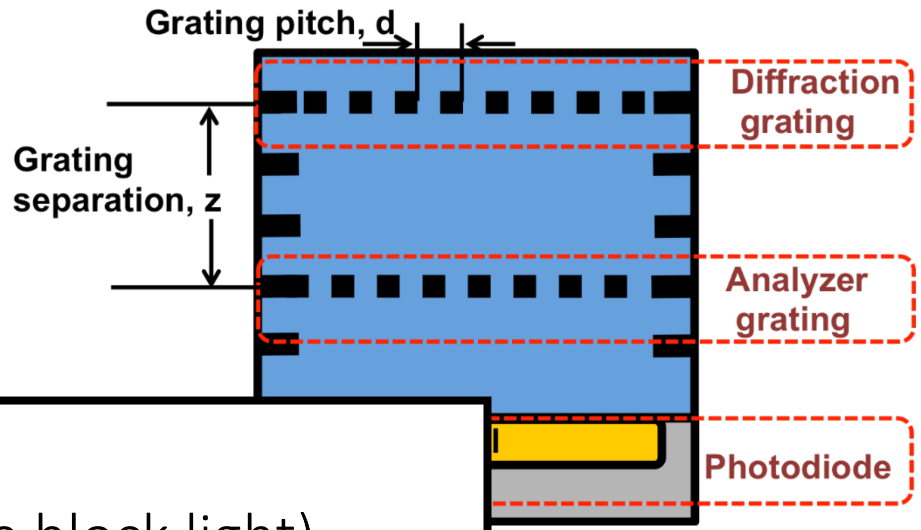
Optical filtering

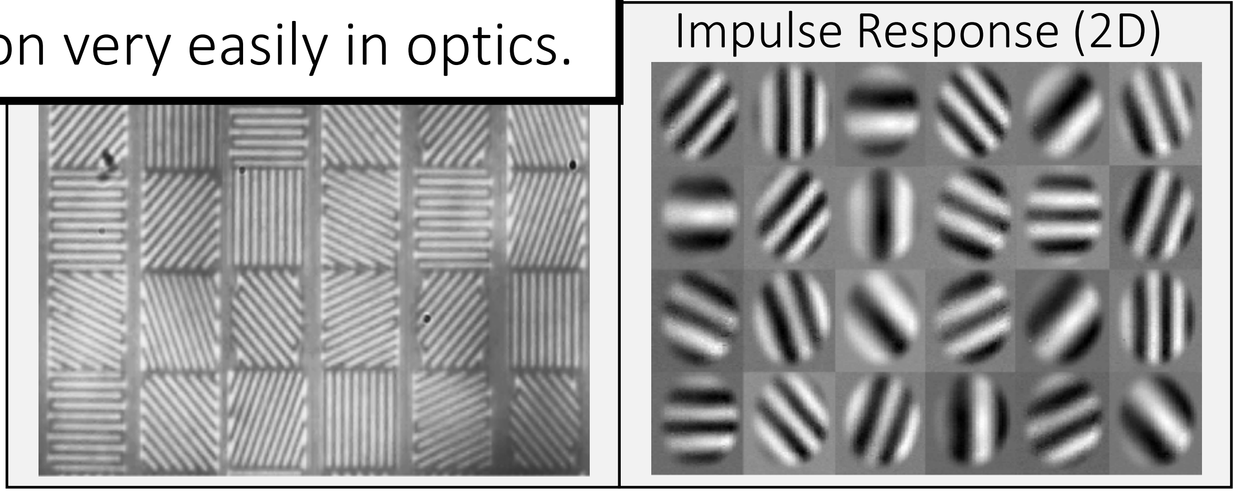




resulting image

Angle-sensitive pixels

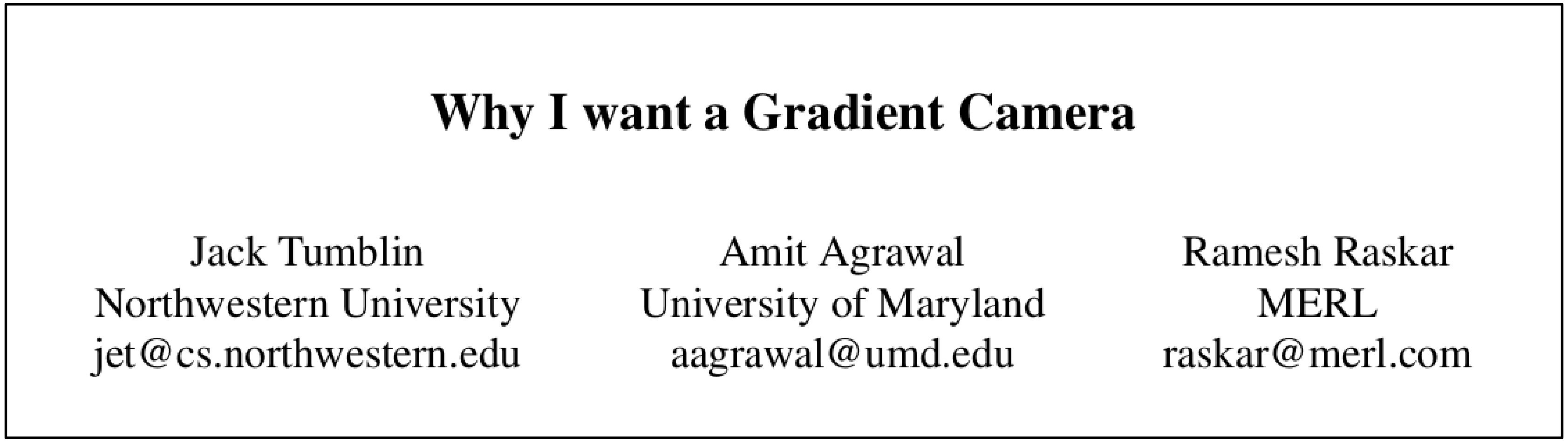




Any disadvantages?

- Reduced light efficiency (we block light).
- We can’t do subtraction very easily in optics.

One of my favorite papers



Why I want a Gradient Camera

Jack Tumblin
Northwestern University
jet@cs.northwestern.edu

Amit Agrawal
University of Maryland
aagrawal@umd.edu

Ramesh Raskar
MERL
raskar@merl.com

Why would you want a gradient camera?

- Much faster frame rate, as you only read out very few pixels (where gradient is significant).
- Much higher dynamic range, if also combined with logarithmic gradients.

Can you directly display the measurements of such a camera?

- You need to use a Poisson solver to reconstruct the image from the measured gradients.

How would you build a gradient camera?

- Change the sensor.
- Change the optics.

We can also compute *temporal* gradients

event-based cameras (a.k.a. dynamic vision sensors, or DVS)

Concept figure for event-based camera:

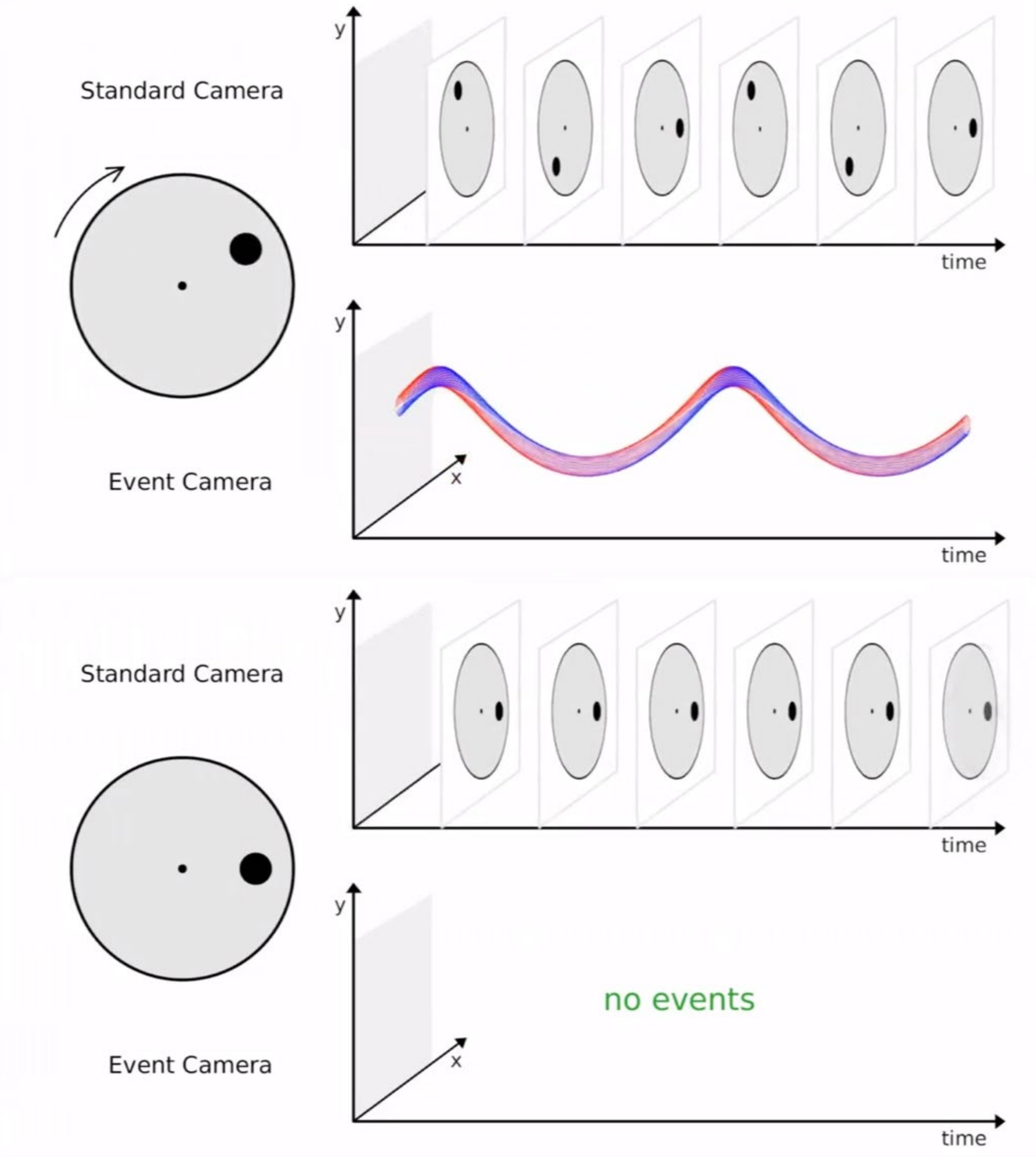
https://www.youtube.com/watch?v=kPCZESVfHoQ

High-speed output on a quadcopter:

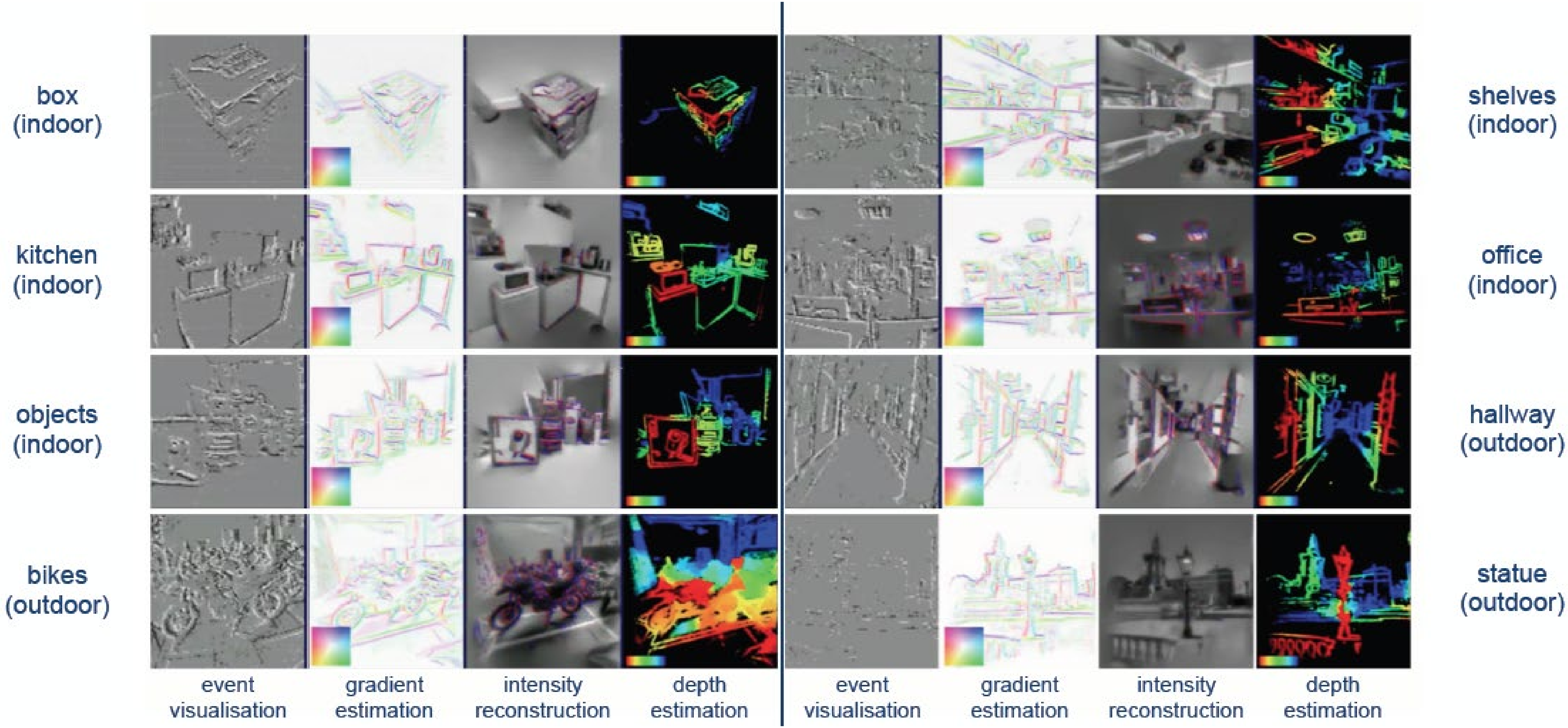
https://www.youtube.com/watch?v=LauQ6LWTkxM

Simulator:

http://rpg.ifi.uzh.ch/esim



Slowly becoming popular in robotics and vision



References

Basic reading:

- Szeliski textbook, Sections 3.13, 3.5.5, 9.3.4, 10.4.3.
- Pérez et al., “Poisson Image Editing,” SIGGRAPH 2003.
  The original Poisson image editing paper.
- Agrawal and Raskar, “Gradient Domain Manipulation Techniques in Vision and Graphics,” ICCV 2007 course, http://www.amitkagrawal.com/ICCV2007Course/
  A great resource (entire course!) for gradient-domain image processing.
- Agrawal et al., “Removing Photography Artifacts Using Gradient Projection and Flash-Exposure Sampling,” SIGGRAPH 2005.
  A paper on photography with flash and no-flash pairs, using gradient-domain image processing.

Additional reading:

- Georgiev, “Covariant Derivatives and Vision,” ECCV 2006.
  An paper from Adobe on the version of Poisson blending implemented in Photoshop’s “healing brush”.
- Elder and Goldberg, “Image editing in the contour domain”, PAMI 2001.
  One of the very first papers discussing gradient-domain image processing.
- Frankot and Chellappa, “A method for enforcing integrability in shape from shading algorithms,” PAMI 1988.
- Bhat et al., “Fourier Analysis of the 2D Screened Poisson Equation for Gradient Domain Problems,” ECCV 2008.
  A couple of papers discussing the (Fourier) basis projection approach for solving the Poisson integration problem.
- Agrawal et al., “What Is the Range of Surface Reconstructions from a Gradient Field?,” ECCV 2006.
- Quéau et al., “Normal Integration: A Survey,” JMIV 2017.
  Two papers reviewing various gradient (and surface normal) integration techniques, including Poisson solvers.
- Szeliski, “Locally adapted hierarchical basis preconditioning,” SIGGRAPH 2006.
- Krishnan and Szeliski, “Multigrid and multilevel preconditioners for computational photography,” SIGGRAPH 2011.
- Krishnan et al., “Efficient Preconditioning of Laplacian Matrices for Computer Graphics,” SIGGRAPH 2013.
  A few well-known references on multi-grid and preconditioning techniques for accelerating the Poisson solver, with a specific focus on computational photography applications..
- Shewchuk, “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain,” CMU TR 1994, http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf
  A great reference on (preconditioned) conjugate gradient solvers for large linear systems.
- Briggs et al., “A multigrid tutorial,” SIAM 2000.
  A great reference book on multi-grid approaches.
- Bhat et al., “GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering,” TOG 2010.
  A paper describing gradient-domain processing as a general image processing paradigm, which can be used for a broad set of applications beyond blending.
- Krishnan and Fergus, “Dark Flash Photography,” SIGGRAPH 2009.
  A paper proposing doing flash/no-flash photography using infrared flash lights.
- Kazhdan et al., “Poisson surface reconstruction,” SGP 2006.
- Kazhdan and Hoppe, “Screened Poisson surface reconstruction,” TOG 2013.
  Two papers discussing Poisson problems for reconstructing meshes from point clouds and normals. This is arguably the most commonly used surface reconstruction algorithm.
- Lehtinen et al., “Gradient-domain metropolis light transport,” SIGGRAPH 2013.
- Kettunen et al., “Gradient-domain path tracing,” SIGGRAPH 2015.
- Hua et al., “Light transport simulation in the gradient domain,” SIGGRAPH Asia 2018 course, http://beltegeuse.s3-website-ap-northeast-1.amazonaws.com/research/2018_GradientCourse/
  In addition to *editing* images in the gradient-domain, we can *render* them directly in the gradient-domain.
- Tumblin et al., “Why I want a gradient camera?” CVPR 2005.
  We can even directly *measure* images in the gradient domain, using so-called *gradient cameras*.
- Callenberg et al., “Snapshot difference imaging using correlation time-of-flight sensors,” SIGGRAPH Asia 2017.
  A form of camera with differential pixels.
- Koppal et al., “Toward wide-angle microvision sensors”, PAMI 2013.
  Gradient cameras using optical filtering.
- Chen et al., “ASP vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels,” CVPR 2016.
  Gradient cameras using angle-sensitive pixels.
- Kim et al., “Real-time 3D reconstruction and 6-DoF tracking with an event camera,” ECCV 2016.
  A paper on using evet-based cameras for computer vision applications in very fast frame rates (best paper award at ECCV 2016!).