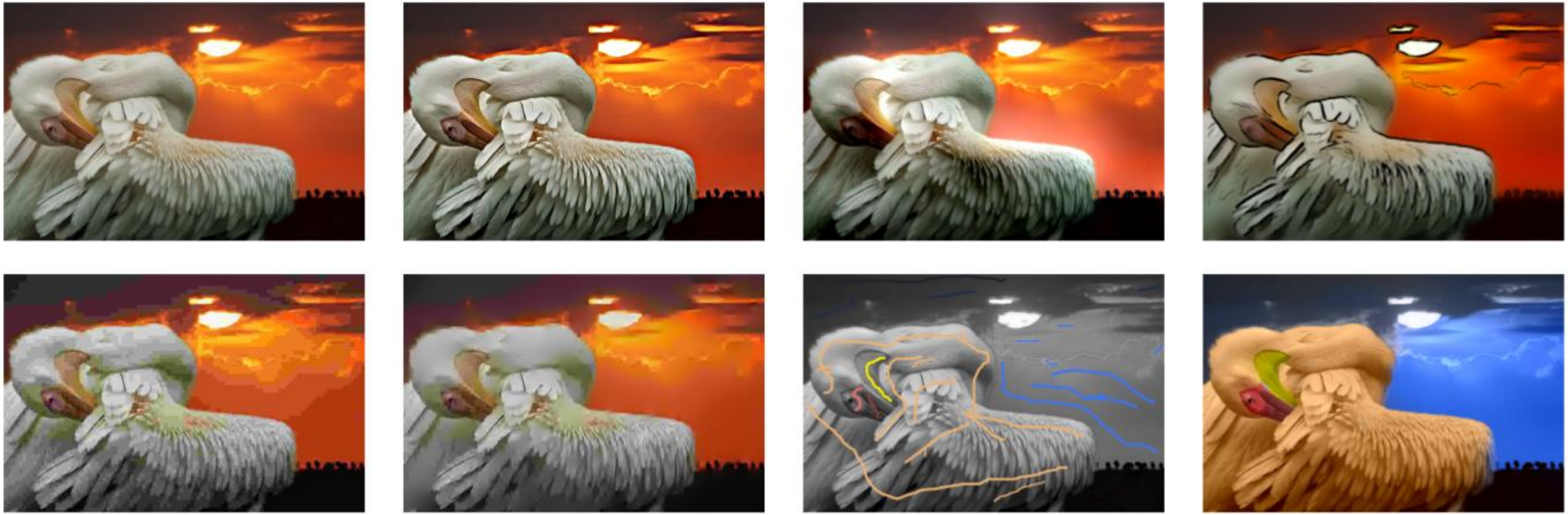
Gradient-domain image processing



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

15-463, 15-663, 15-862
Computational Photography
Fall 2019, Lecture 9

Course announcements

- Homework 2 is out.
  - Due September 27th.
  - Requires camera *and* tripod.
  - Start early! *Substantially* larger programming and imaging components than in Homework 1.
  - Generous bonus component, up to 50% extra credit.
  - No really: start early!

- Computational imaging group meeting is on Fridays, 3 - 4 pm, WEH 5421.
  - You are welcome to attend.
  - You can also join the comp-imaging mailing list for related announcements (see Piazza for link).

Overview of today’s lecture

- 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.
- Gradient 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

Someone leaked season 8 of Game of Thrones

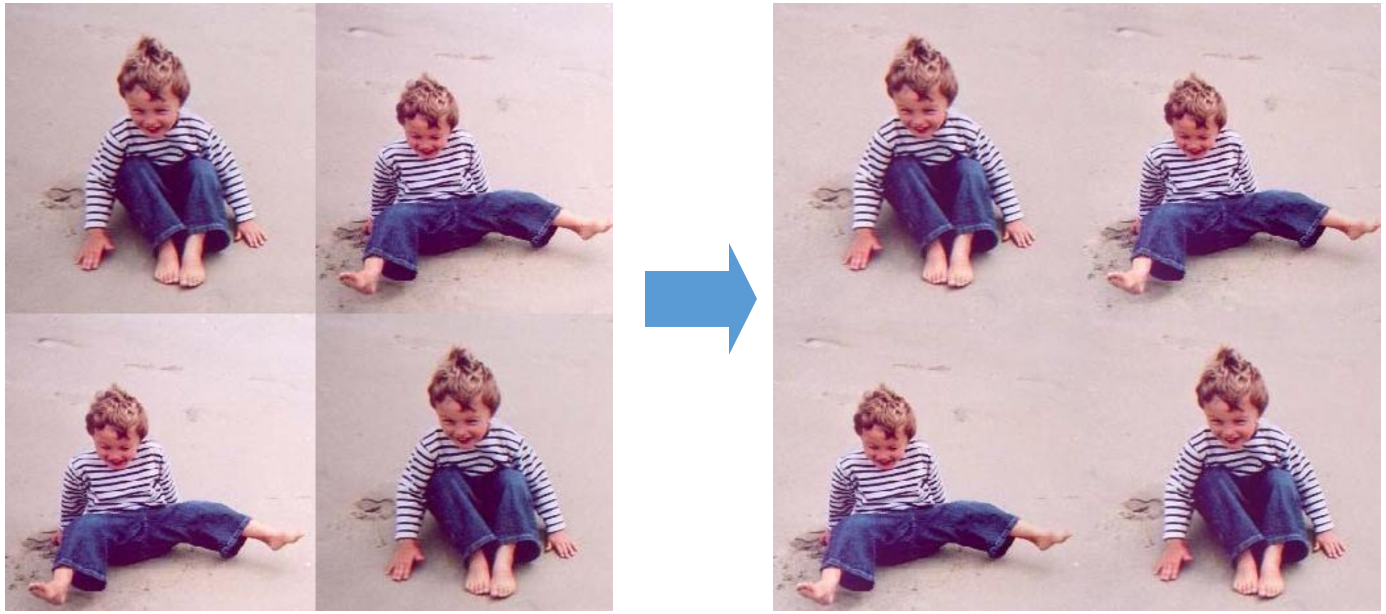
or, more likely, they made some creative use of Poisson blending

Application: Poisson blending

originals copy-paste Poisson blending

More applications

Removing Glass Reflections



Seamless Image Stitching

Yet more applications

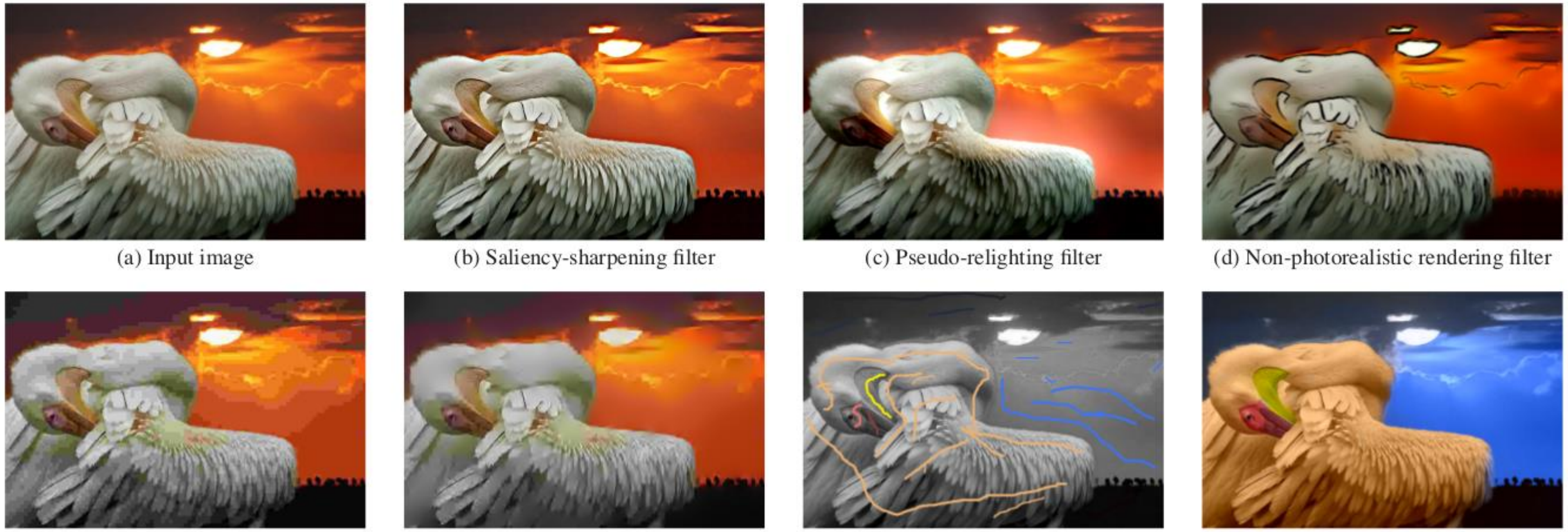
Fusing day and night photos

Tonemapping

Entire suite of image editing tools

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

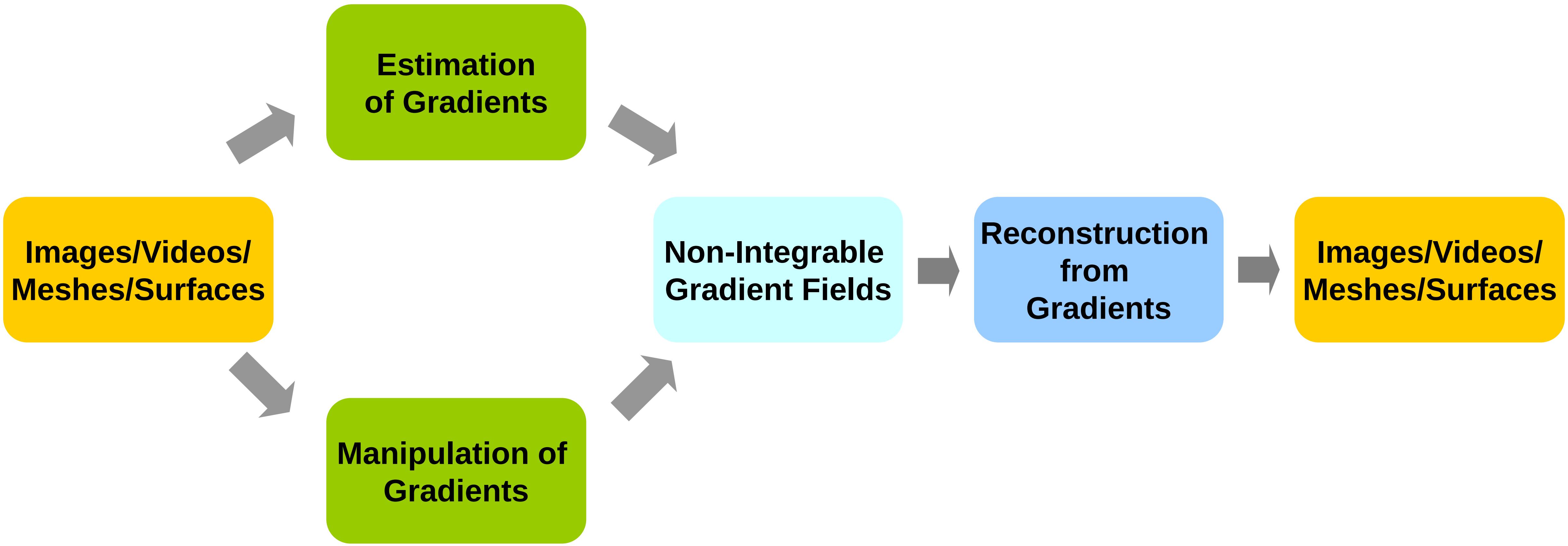
Pravin Bhat[1] C. Lawrence Zitnick[2] Michael Cohen[1,2] Brian Curless[1]
[1]University of Washington [2]Microsoft Research



(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

Figure 1: The figure shows some of the image-enhancement filters we have created using the GradientShop optimization-framework. GradientShop has been designed to allow applications to explore gradient-domain solutions for various image processing problems.

Main pipeline



Basics of images and gradients

Image representation

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

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

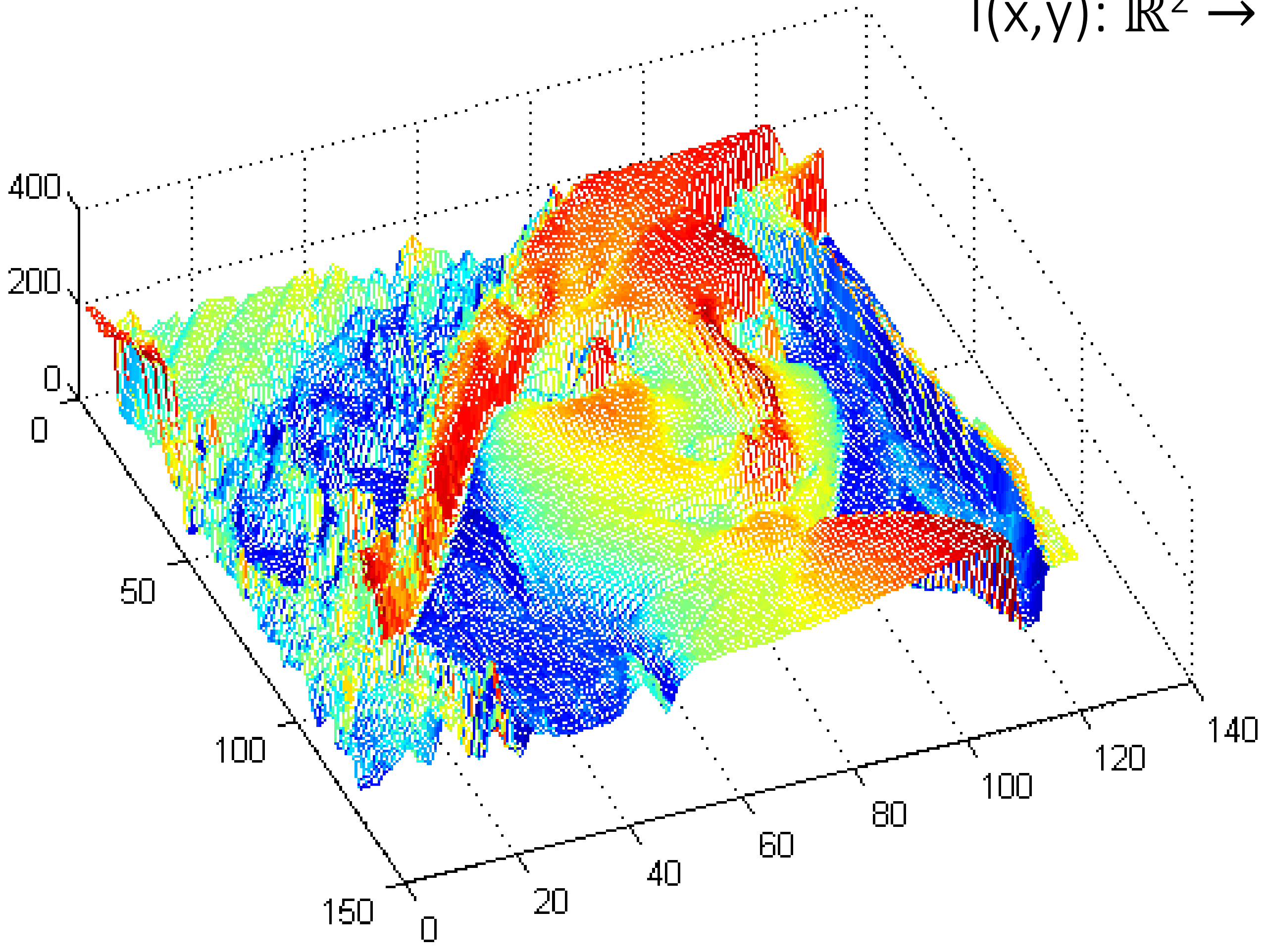


Image gradients

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

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

vector field $\nabla I = \{\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\} \; : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

Image gradients

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

scalar field $I(x, y) \; : \mathbb{R}^2 \rightarrow \mathbb{R}$ ➡ vector field $\nabla I = \{\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\} \; : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

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

Finite differences

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

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

Finite differences

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

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

How do you efficiently compute this?

Finite differences

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

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

What convolution kernel does this correspond to?

Finite differences

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

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

-1	0	1	?

1	0	-1	?

Finite differences

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

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

1D derivative filter

1	0	-1

Finite differences

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

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

For discrete signals: Remove limit and set h = 1

$$f'(x) = f(x+1) - f(x)$$

1D derivative filter

1	-1

We will be using forward differences in this lecture!

Image gradients

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

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

vector field $\nabla I = \{\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\} \ : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

- 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 core 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 $(\mathbf{u}, \mathbf{v})$, 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

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

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

Curl and divergence

Curl: vector operator showing the rate of rotation of a vector field.

$$Curl\ (\nabla I) = \nabla \times \nabla I$$

What is the dimension of this?

Divergence: vector operator showing the isotropy of a vector field.

$$Div\ (\nabla I) = \nabla \bullet \nabla I$$

What is the dimension of this?

Curl and divergence

Curl: vector operator showing the rate of rotation of a vector field.

$$Curl\ (\nabla I) = \nabla \times \nabla I$$

Another vector field (in 2D, this is parallel to a vector orthogonal to the 2D plane).

Divergence: vector operator showing the isotropy of a vector field.

$$Div\ (\nabla I) = \nabla \bullet \nabla I$$

Scalar field

How do we write these operators in terms of derivatives of I?

Curl and divergence

Curl: vector operator showing the rate of rotation of a vector field.

$$Curl\ (\nabla I) = \det \begin{vmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \\ I_x & I_y \end{vmatrix} = \frac{\partial I_y}{\partial x} - \frac{\partial I_x}{\partial y} = I_{yx} - I_{xy}$$

(here we ignore a unit vector k)

Divergence: vector operator showing the isotropy of a vector field.

$$div\ (I_x, I_y) = \frac{\partial I_x}{\partial x} + \frac{\partial I_y}{\partial y} = I_{xx} + I_{yy}$$

Property of twice-differentiable functions

Curl of the gradient field should be zero:

$$Curl\ (\nabla I) = I_{yx} - I_{xy} = 0$$

What does that mean intuitively?

Property of twice-differentiable functions

Curl of the gradient field should be zero:

$$Curl\ (\nabla I) = I_{yx} - I_{xy} = 0$$

What does that mean intuitively?

- Same result independent of order of differentiation.

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

Demonstration

Image

I_x

I_y

How do we compute this? → $\mathrm{Div}(I_{x,}\ I_y)$

$\mathrm{Curl}(I_{x,}\ I_y)$

I_{xy}

=

I_{yx}

Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

→ 1D derivative filter

1	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

→ Laplace filter

?

Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$\longrightarrow$ 1D derivative filter

1	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

$\longrightarrow$ Laplace filter

1	-2	1

Property of twice-differentiable functions

Curl of the gradient field should be zero:

$$Curl\ (\nabla I) = I_{yx} - I_{xy} = 0$$

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 $\mathbf{I}$?

?

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

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

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

such that

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

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

Only if:

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

Vector field integration

Two core 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 field 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*.

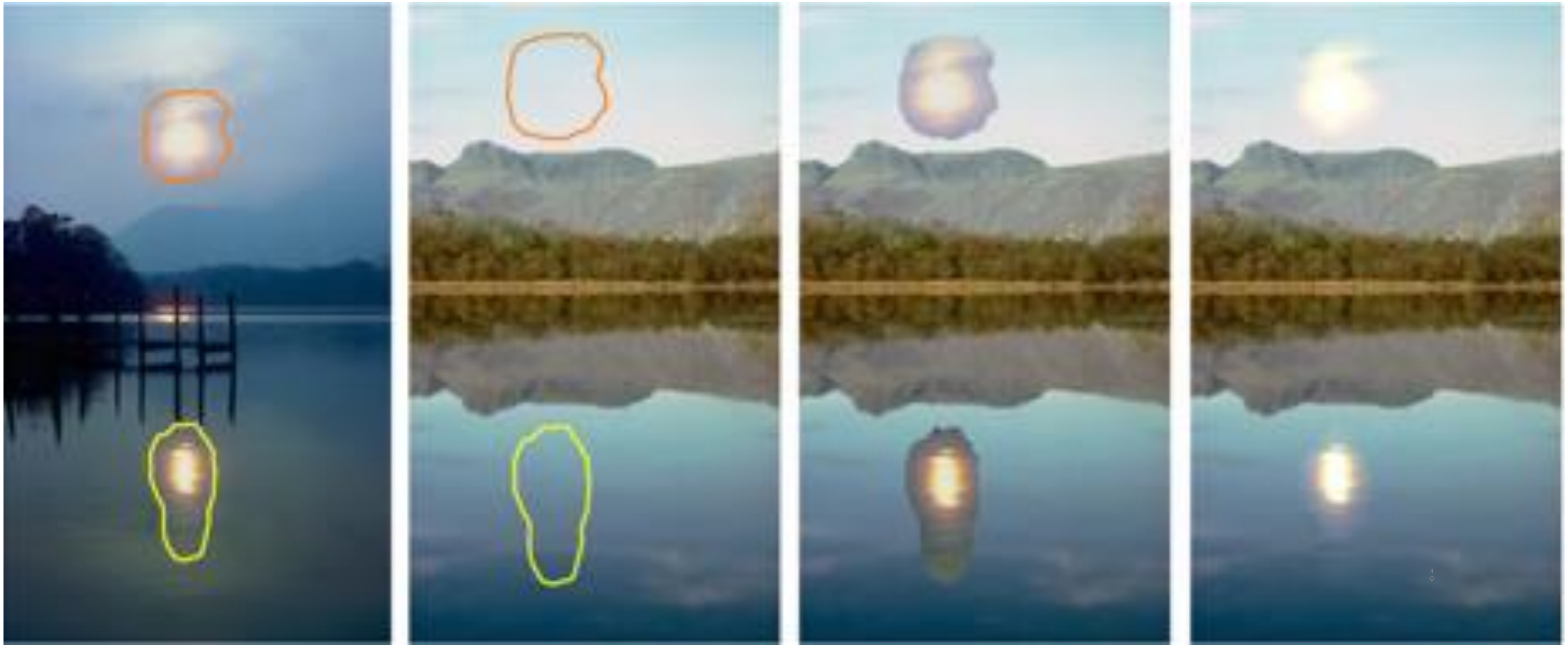
Poisson blending

Application: Poisson blending

originals copy-paste Poisson blending

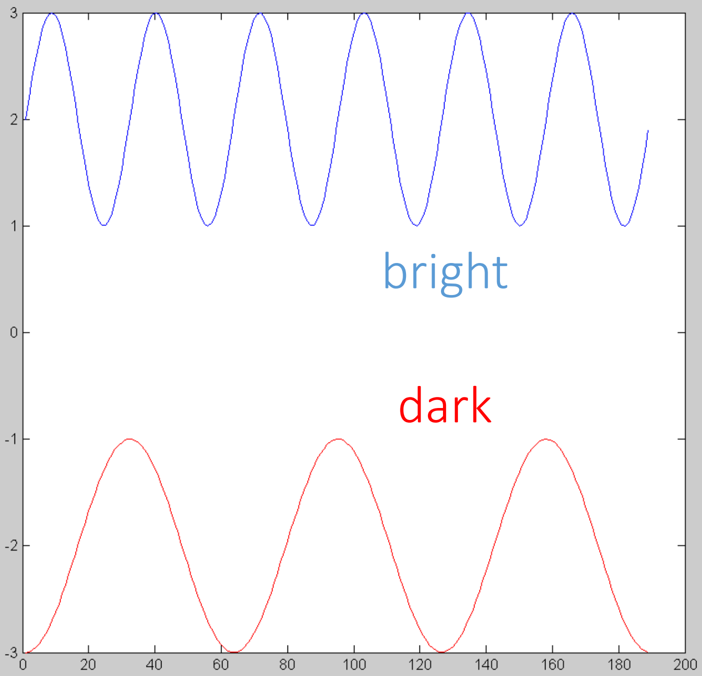
Key idea

When blending, retain the gradient information as best as possible

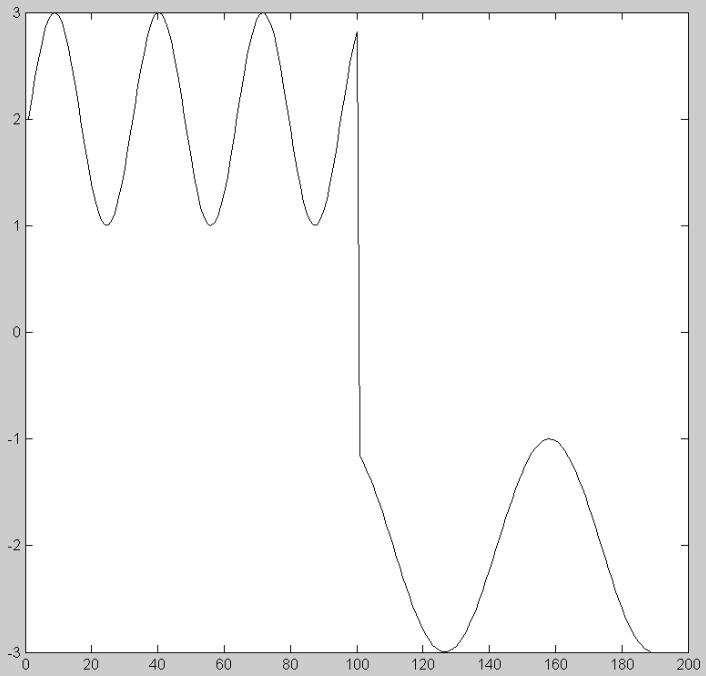


source destination copy-paste Poisson blending

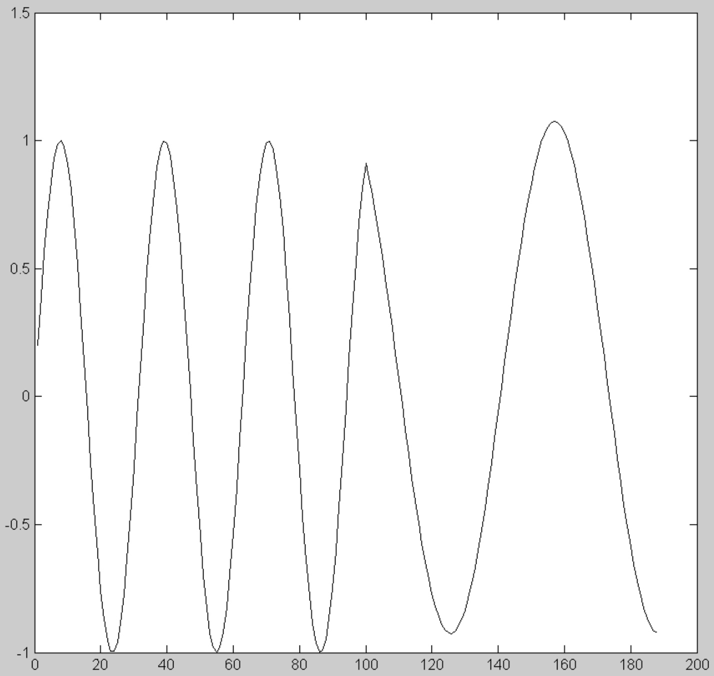
Poisson blending: 1D example



two signals



regular blending



blending derivatives

Definitions and notation

Notation

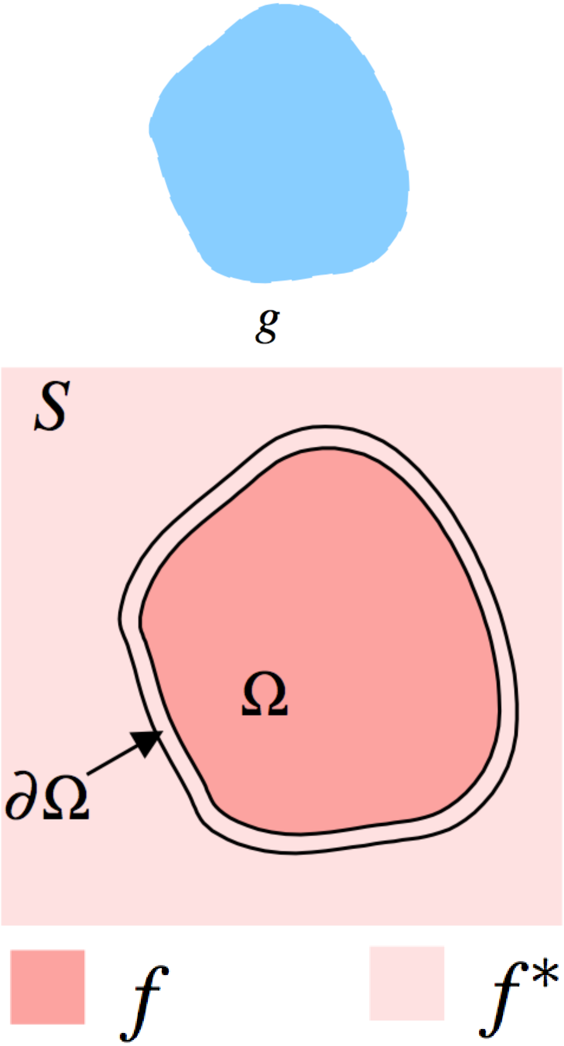
g: source function

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

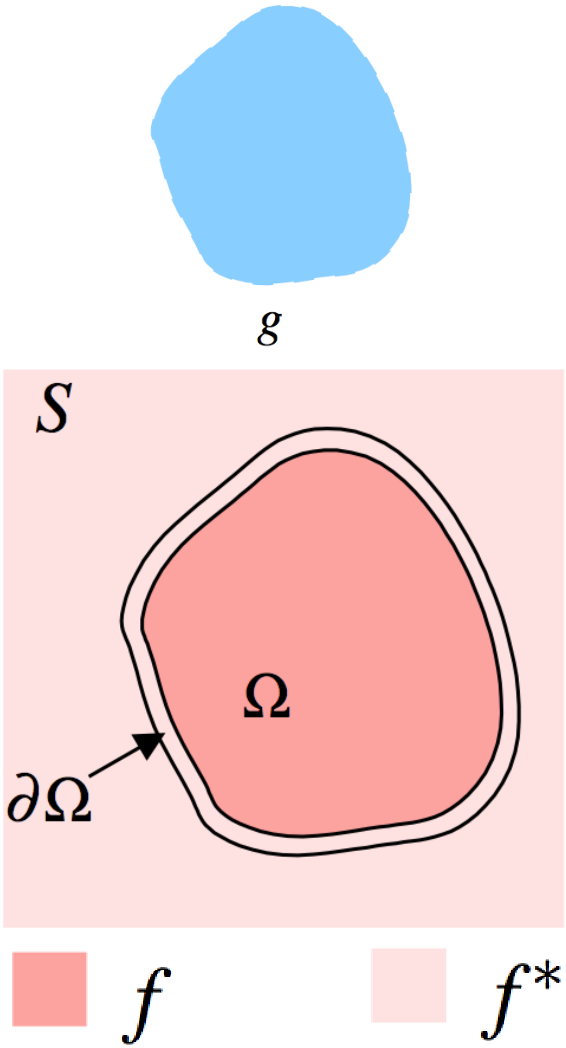
Ω: destination domain

f: interpolant function

f*: destination function

How should we determine f?

- should it look like g?
- should it look like f*?



Interpolation criterion

"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 ...

Image gradient

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

is this known?

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

Interpolation criterion

"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 gradient of g

f is equivalent to f* at the boundaries

Recall ...

Image gradient

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

Yes, since the source function g is known

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

Equivalently

This is where *Poisson* blending comes from

Poisson equation (with Dirichlet boundary conditions)

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

what does this term do?

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

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

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

$$\begin{aligned} \text{div } \mathbf{v} &= \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \\ &= \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \\ &= \Delta g \end{aligned}$$

Equivalently

Poisson equation (with Dirichlet boundary conditions)

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

Laplacian of f same as g

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

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

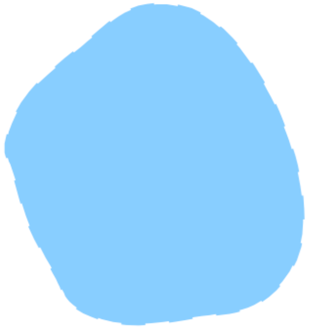
Divergence $\text{div } \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$

$$\begin{aligned} \text{div } \mathbf{v} &= \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \\ &= \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \\ &= \Delta g \end{aligned}$$

Equivalently

Poisson equation (with Dirichlet boundary conditions)

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

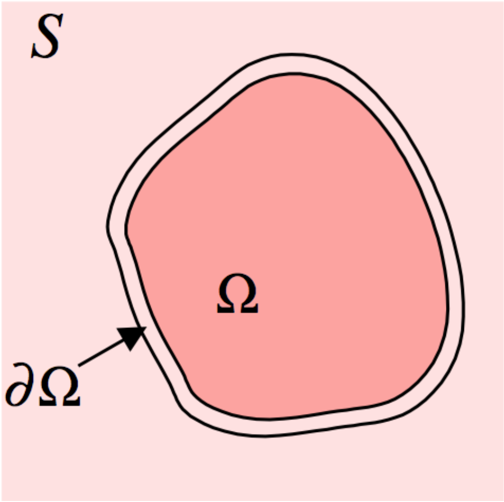


g

so make these guys ...

Δg Δf

the same



How can we do this?

Equivalently

Poisson equation (with Dirichlet boundary conditions)

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

So for each pixel p, do: $$\Delta f_p = \Delta g_p$$

Or for discrete images: $$4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$$

How did we compute the Laplacian?

Equivalently

Poisson equation (with Dirichlet boundary conditions)

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

So for each pixel p, do:

$$\Delta f_p = \Delta g_p$$

Or for discrete images:

$$4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$$

Recall...

Laplace filter

0	1	0
1	-4	1
0	1	0

What's known and what's unknown?

Equivalently

Poisson equation (with Dirichlet boundary conditions)

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

So for each pixel p, do: $\Delta f_p = \Delta g_p$

Or for discrete images: $4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$

Recall...

Laplace filter

0	1	0
1	-4	1
0	1	0

f is unknown except at the boundary

g and its Laplacian are known

We can rewrite this as

linear equation of N variables

$$4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$$

one for each pixel in destination

In vector form:

$$[0 \cdots -1 \cdots -1 \quad 4 \quad -1 \cdots -1 \cdots 0] \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} \Delta g_1 \\ \vdots \\ \Delta g_{q_1} \\ \vdots \\ \Delta g_{q_2} \\ \Delta g_p \\ \Delta g_{q_3} \\ \vdots \\ \Delta g_{q_4} \\ \vdots \\ \Delta g_N \end{bmatrix}$$

What is this? ↗

(each pixel adds another 'sparse' row here)

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:

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

Expand the error:

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

Minimize the error:

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

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

Solving the linear system

Convert the system to a linear least-squares problem:

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

Expand the error:

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

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 <u>never</u> want to compute the inverse of a matrix.

In Matlab:

```
f = A \ b
```

Integration procedures

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

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

- 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 gradient of g

f is equivalent to f* at the boundaries

Recall ...

Image gradient

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

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 gradient of g

f is equivalent to f* at the boundaries

Recall ...

Image gradient

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

And for discrete images:

$\frac{\partial}{\partial x} \approx$

1	-1

$\frac{\partial}{\partial y} \approx$

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 ...

Image gradient

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

And for discrete images:

$$\frac{\partial}{\partial x} \approx \begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$\frac{\partial}{\partial y} \approx \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

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:

$$\frac{\partial}{\partial x} \approx \begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$\frac{\partial}{\partial y} \approx \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Let's look again at our optimization 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:

$$\frac{\partial}{\partial x} \approx \begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$\frac{\partial}{\partial y} \approx \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

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:

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

What is this vector?

What is this matrix?

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:

$$\frac{\partial E}{\partial f} = 0 \Rightarrow \underbrace{G^T G}_{A} f = \overbrace{G^T v}^{b}$$

It is equal to the vector b we derived previously!

It is equal to the Laplacian matrix A we derived previously!

Reminder from variational case

Poisson equation (with Dirichlet boundary conditions)

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

So for each pixel p, do:

$$\Delta f_p = \Delta g_p$$

Or for discrete images:

$$4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$$

Recall...

Laplace filter

0	1	0
1	-4	1
0	1	0

What's known and what's unknown?

Reminder from variational case

linear equation of N variables

$$4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$$

one for each pixel in destination

In vector form:

$$[0 \cdots -1 \cdots -1 \quad 4 \quad -1 \cdots -1 \cdots 0] \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} \Delta g_1 \\ \vdots \\ \Delta g_{q_1} \\ \vdots \\ \Delta g_{q_2} \\ \Delta g_p \\ \Delta g_{q_3} \\ \vdots \\ \Delta g_{q_4} \\ \vdots \\ \Delta g_N \end{bmatrix}$$

(each pixel adds another 'sparse' row here)

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:

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

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

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

Gradient descent

Given the loss function:

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

Minimize by iteratively computing:

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

Is this cheaper than the pseudo-inverse approach?

Gradient descent

Given the loss function:

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

Minimize by iteratively computing:

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

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:

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

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:

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

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:

$$f^{i+1} = f^i + \eta^i d^i, \quad r^i = b - Af^i, \qquad \text{for } i = 0, 1, \ldots, N$$

$$d^{i+1} = r^{i+1} + \beta^{i+1} d^i,$$

$$\beta^{i+1} = \frac{\left(r^{i+1}\right)^T r^{i+1}}{(r^i)^T r^i} \qquad \eta^i = \frac{\left(d^i\right)^T r^i}{(d^i)^T A d^i}$$

- Smarter way for selecting update directions
- Everything can still be done using convolutions

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 typically use a *multi-grid* 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 NxN.

We can rewrite this as

linear equation of N variables

$$4f_p - \sum_{q \in N_p} f_q = 4g_p - \sum_{q \in N_p} g_q$$

one for each pixel in destination

In vector form:

$$[0 \cdots -1 \cdots -1 \quad 4 \quad -1 \cdots -1 \cdots 0] \begin{bmatrix} f_1 \\ \vdots \\ f_{q_1} \\ \vdots \\ f_{q_2} \\ f_p \\ f_{q_3} \\ \vdots \\ f_{q_4} \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} \Delta g_1 \\ \vdots \\ \Delta g_{q_1} \\ \vdots \\ \Delta g_{q_2} \\ \Delta g_p \\ \Delta g_{q_3} \\ \vdots \\ \Delta g_{q_4} \\ \vdots \\ \Delta g_N \end{bmatrix}$$

What is this? ↗

(each pixel adds another 'sparse' row here)

⇒ 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)

Note: Handling (Dirichlet) boundary conditions

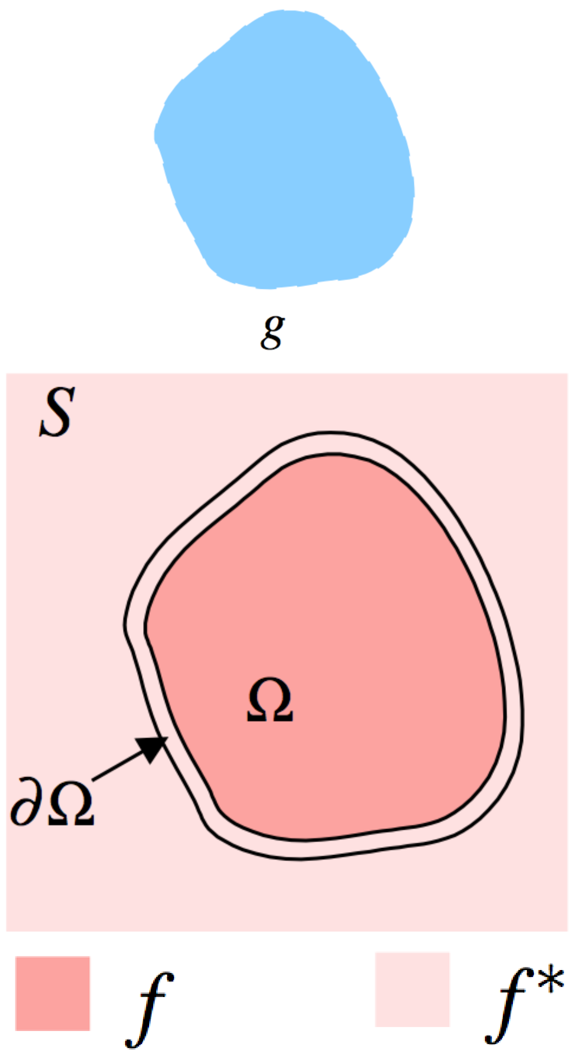
- Form a mask M that is 0 for pixels that should *not* be updated (pixels on S-Ω and $\partial\Omega$) and 1 otherwise.

- 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:

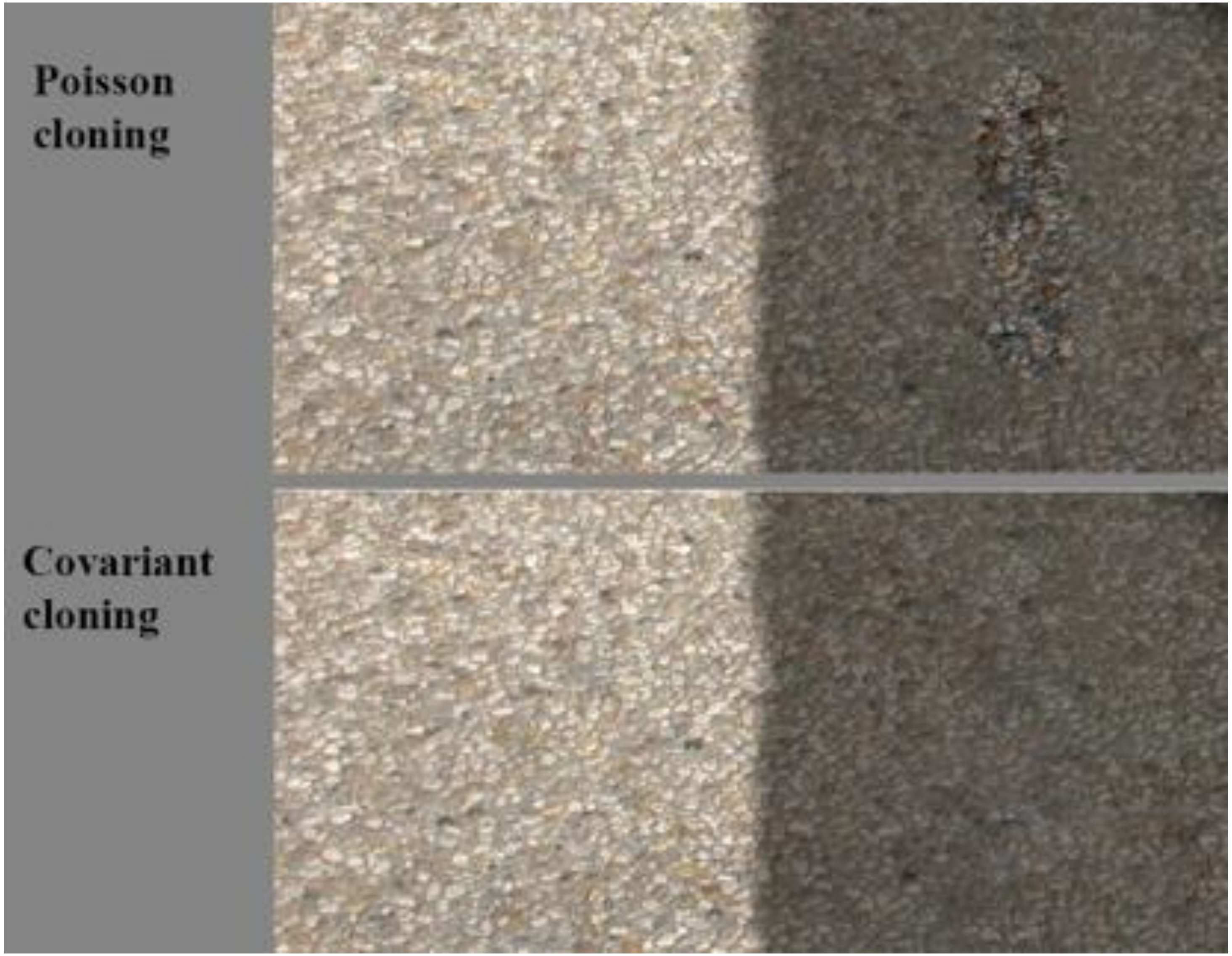
$$f^{i+1} = f^i + M\eta^i r^i \qquad \text{(gradient descent)}$$

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



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

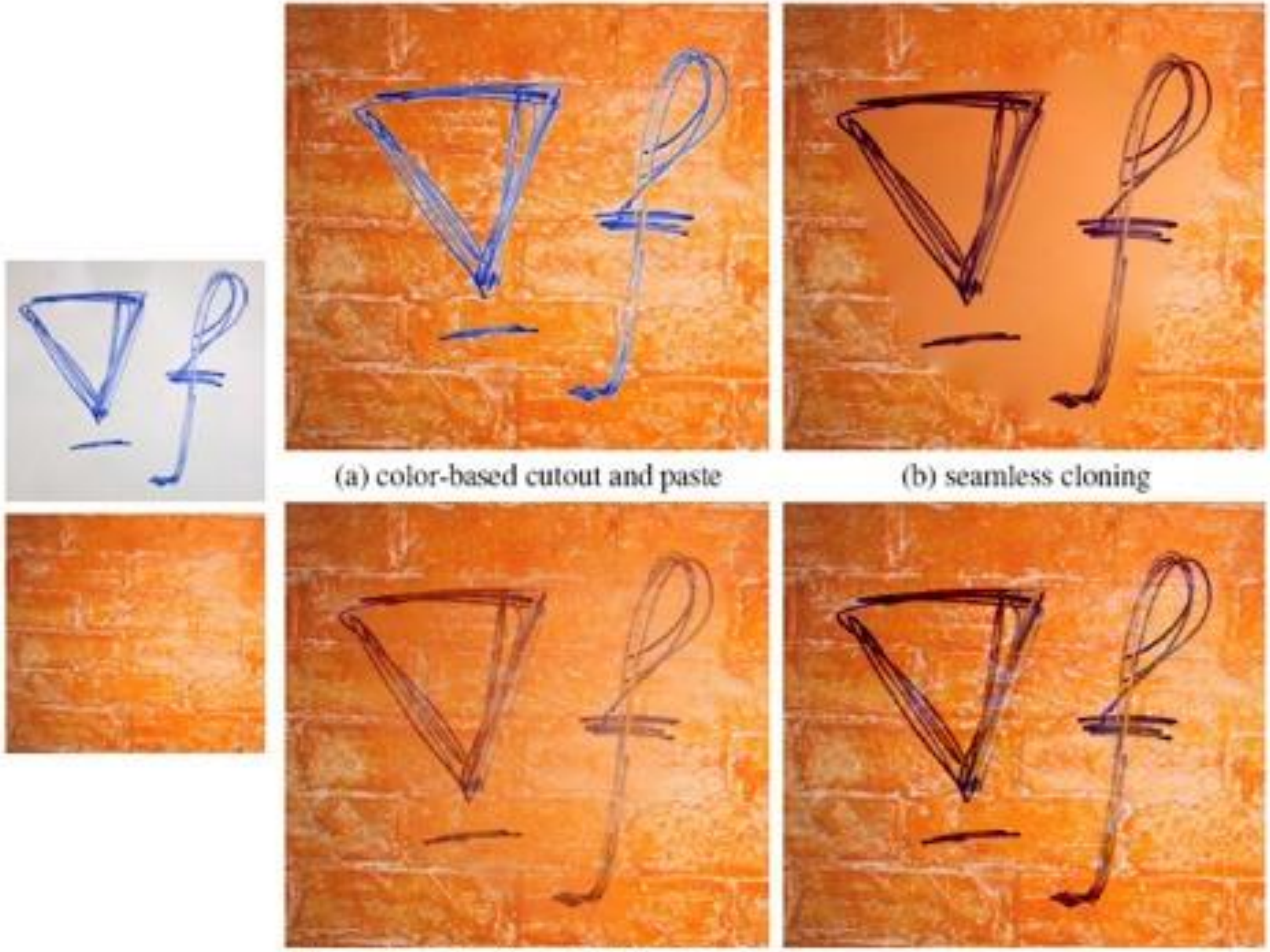
Poisson blending

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

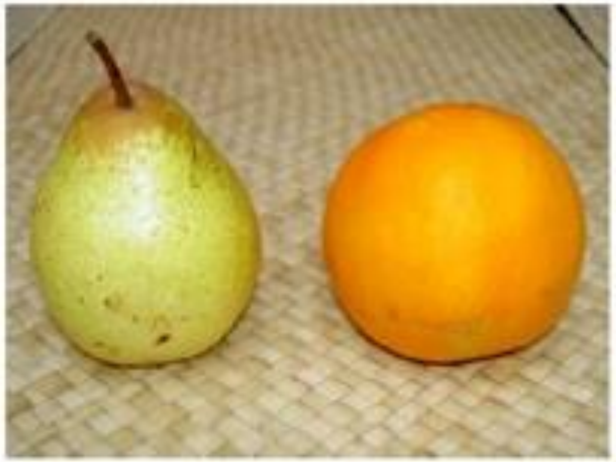
How would you do this with Poisson blending?

Concealment

How would you do this with Poisson blending?

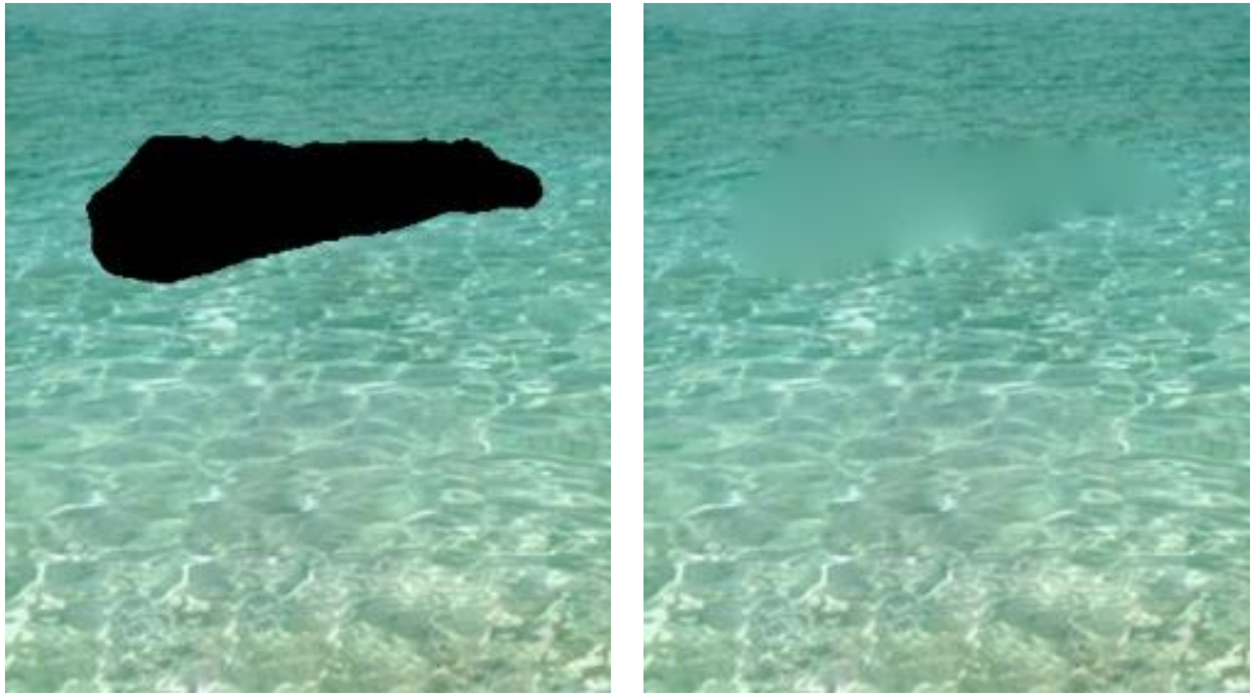
- Insert a copy of the background.

Texture swapping



Special case: membrane interpolation

How would you do this?



Special case: membrane interpolation

How would you do this?

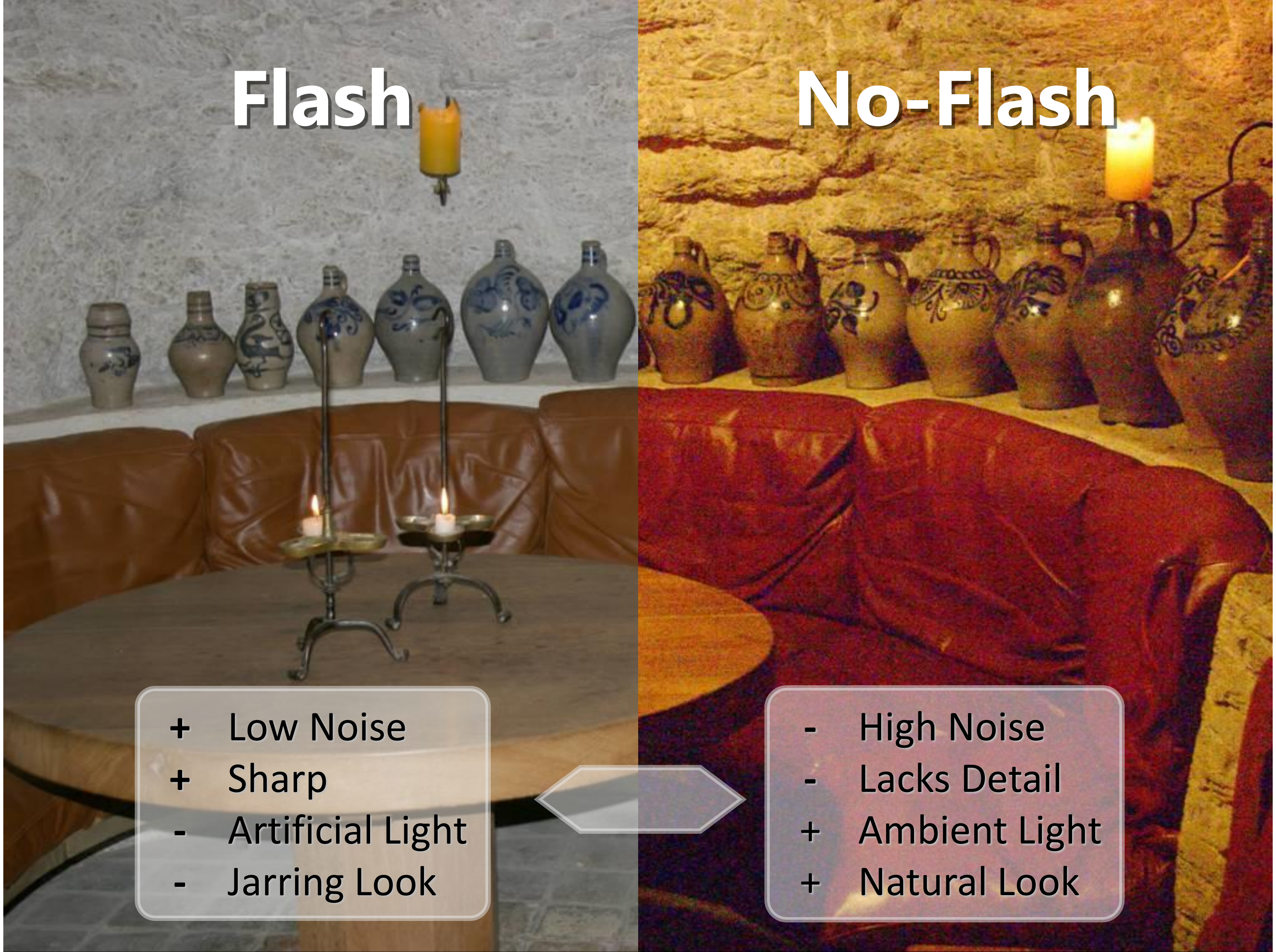
Poisson problem

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

Laplacian problem

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

Flash/no-flash photography


Flash
No-Flash
+ Low Noise
+ Sharp
- Artificial Light
- Jarring Look
- High Noise
- Lacks Detail
+ Ambient Light
+ Natural Look


Denoising Result

• Sh


Denoising Result

Key idea

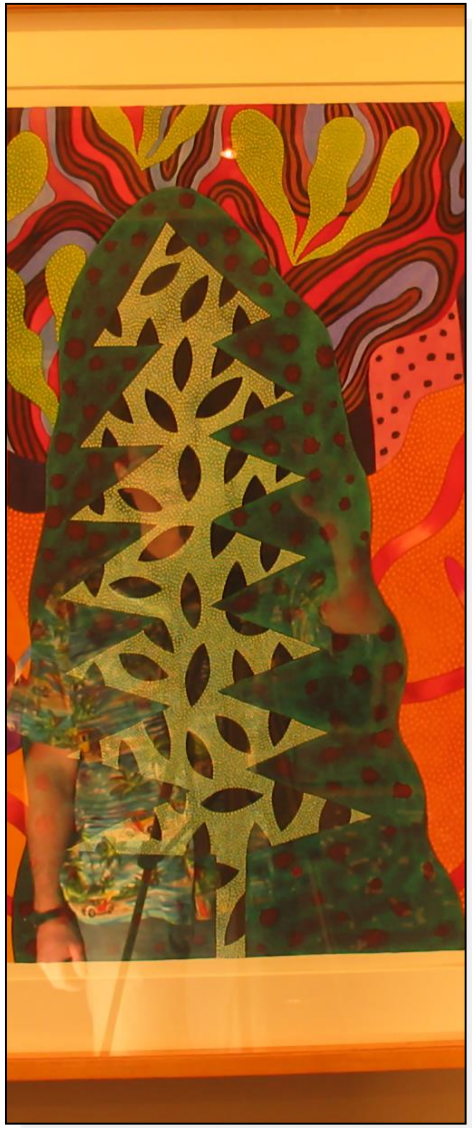
Denoise the no-flash image while maintaining the edge structure of the flash image

- How would you do this using the image editing techniques we’ve learned about?

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

Removing self-reflections and hot-spots

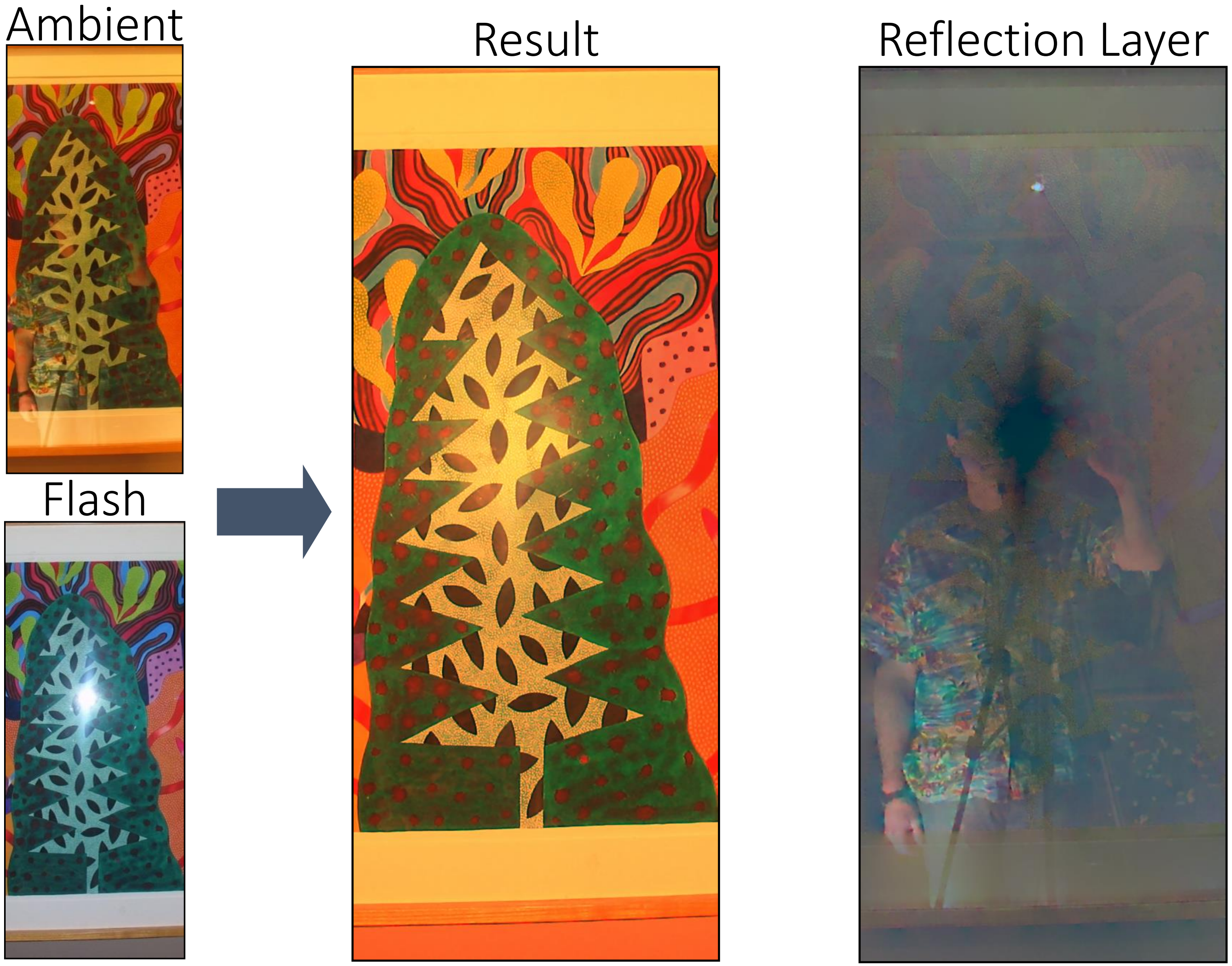
Ambient



Flash

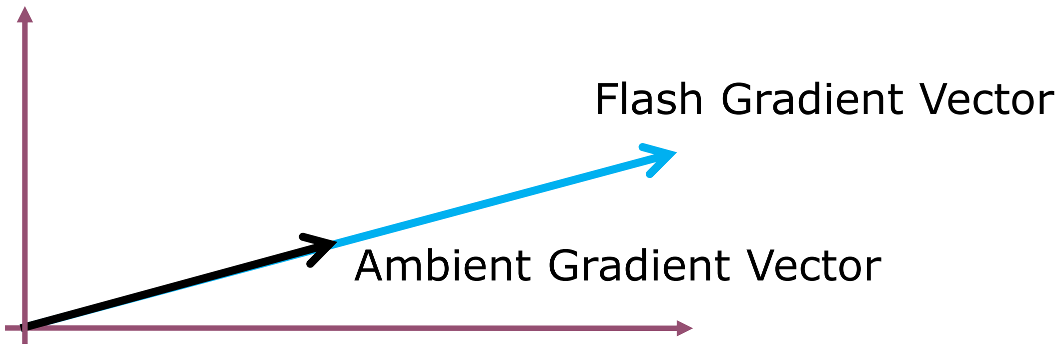
Removing self-reflections and hot-spots

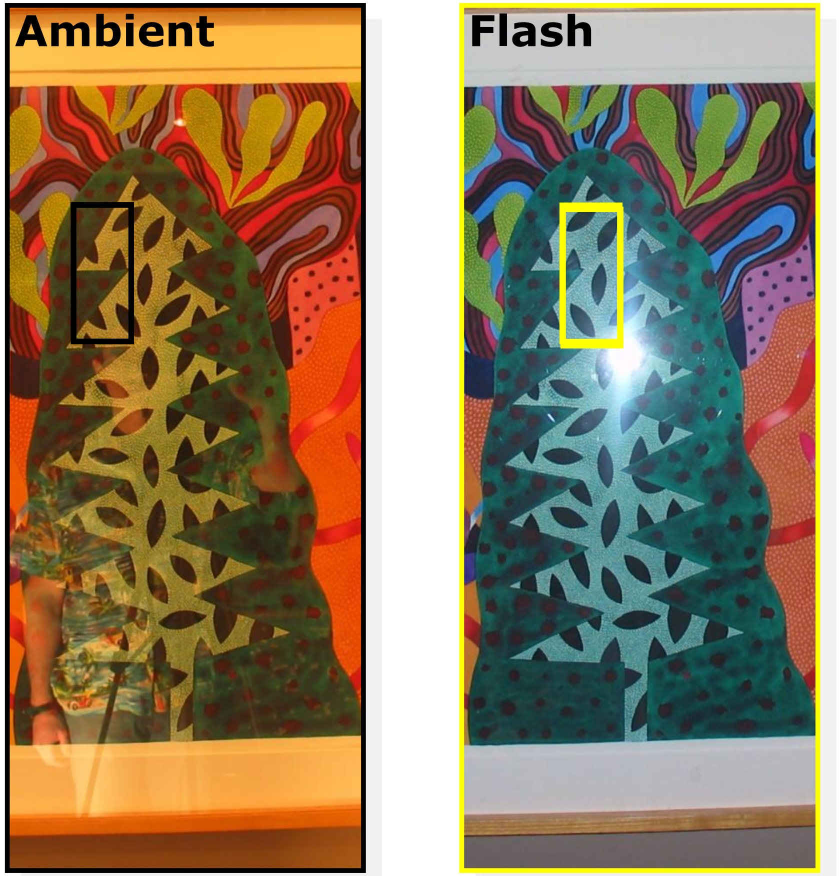
Removing self-reflections and hot-spots

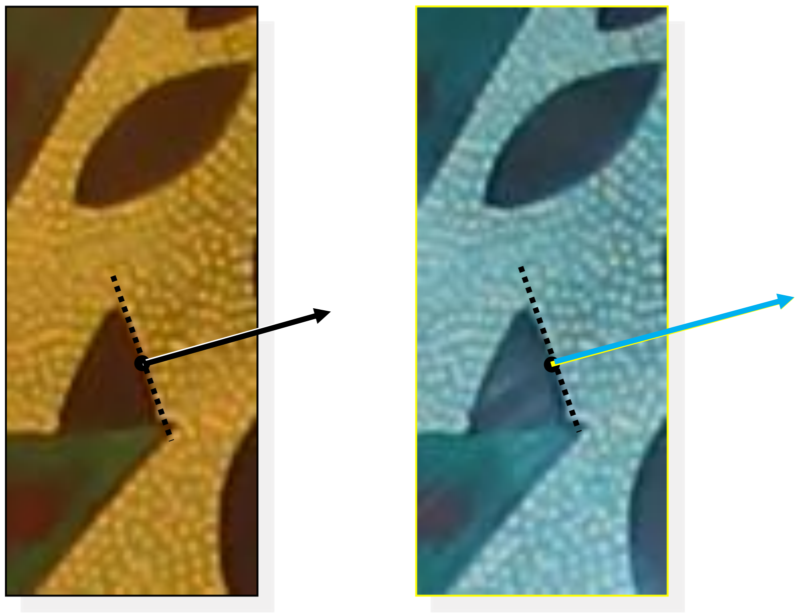


Idea: look at how gradients are affected

Same gradient vector direction



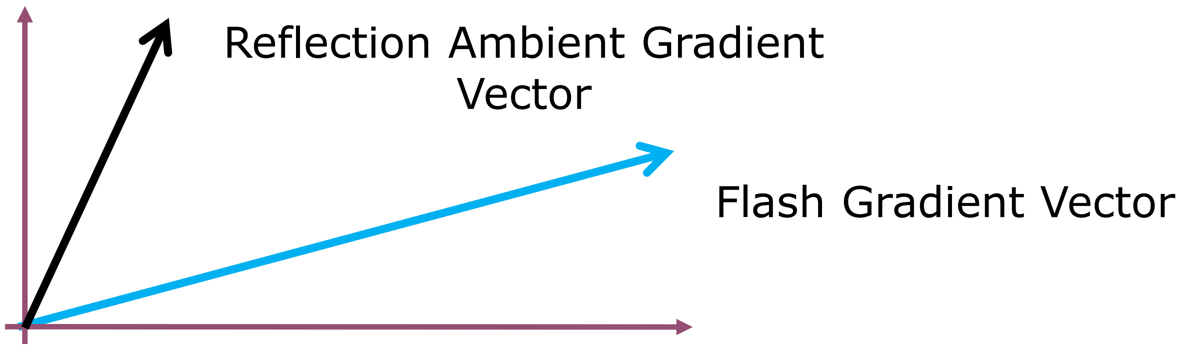


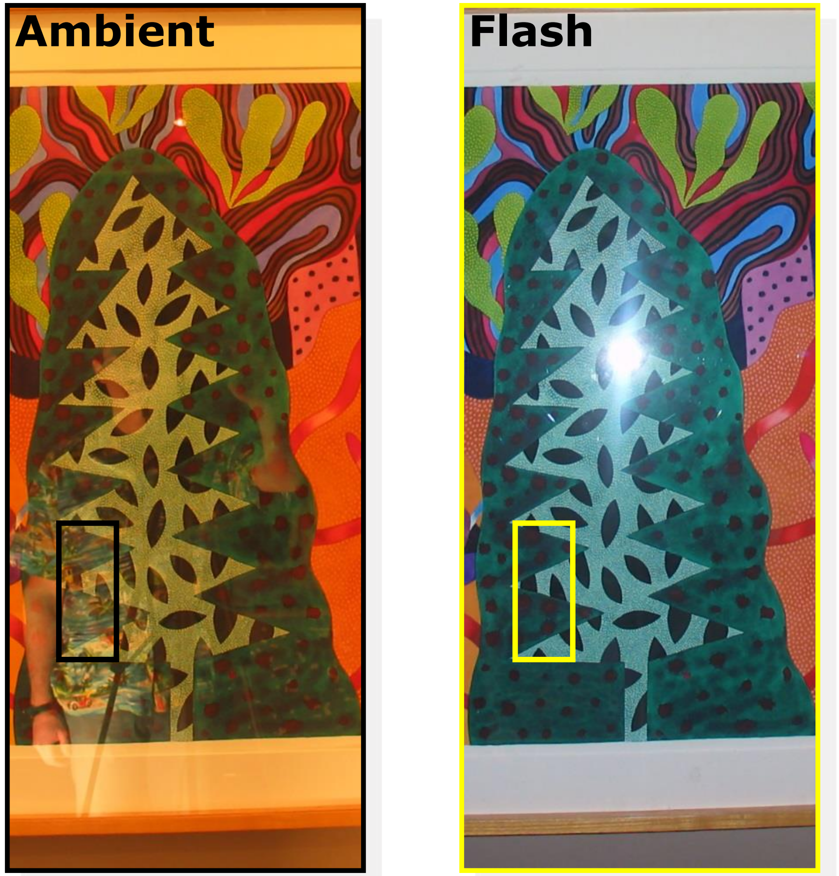


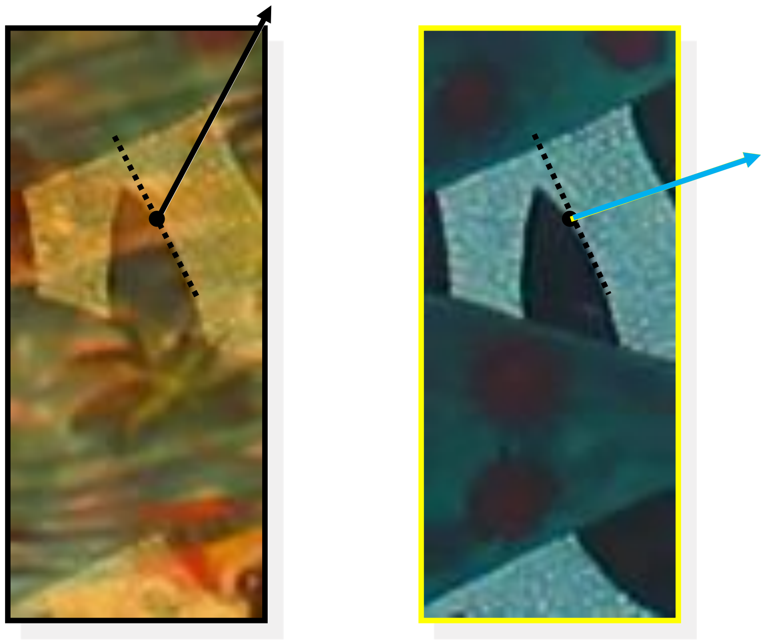
No reflections

Idea: look at how gradients are affected

Different gradient vector direction

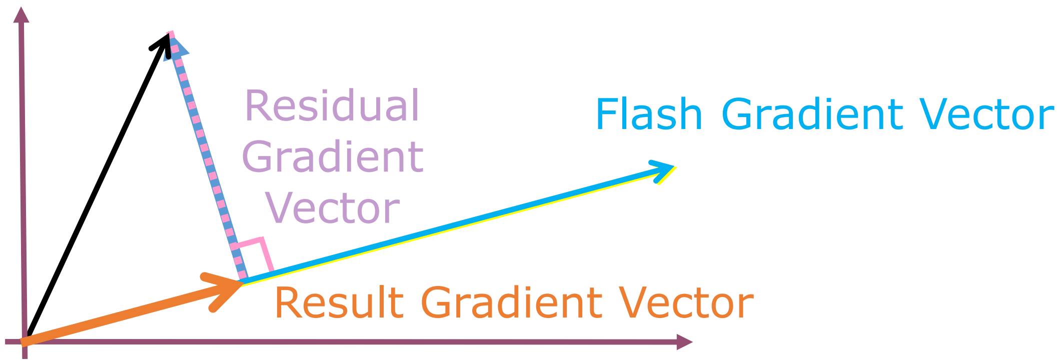


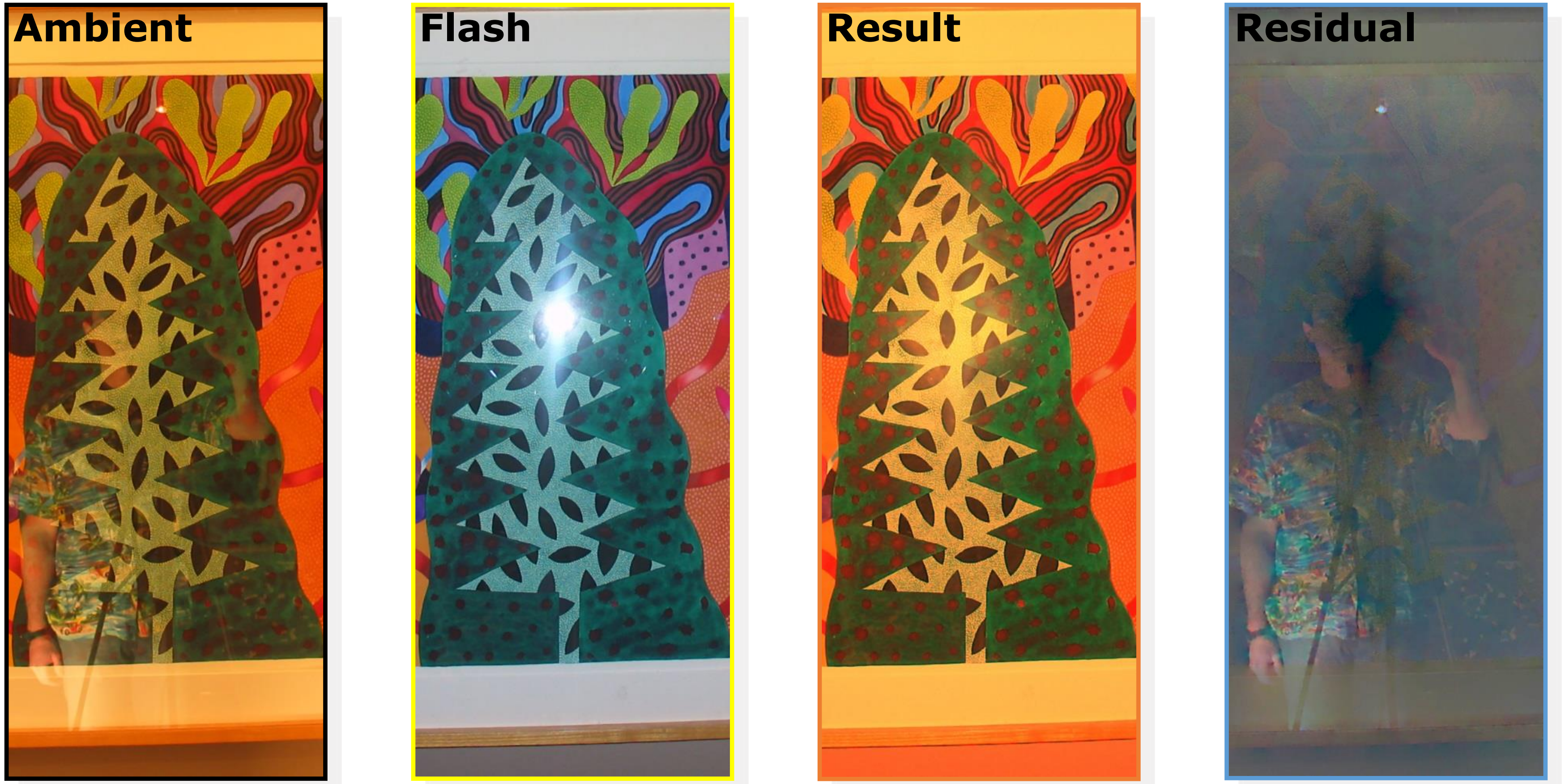




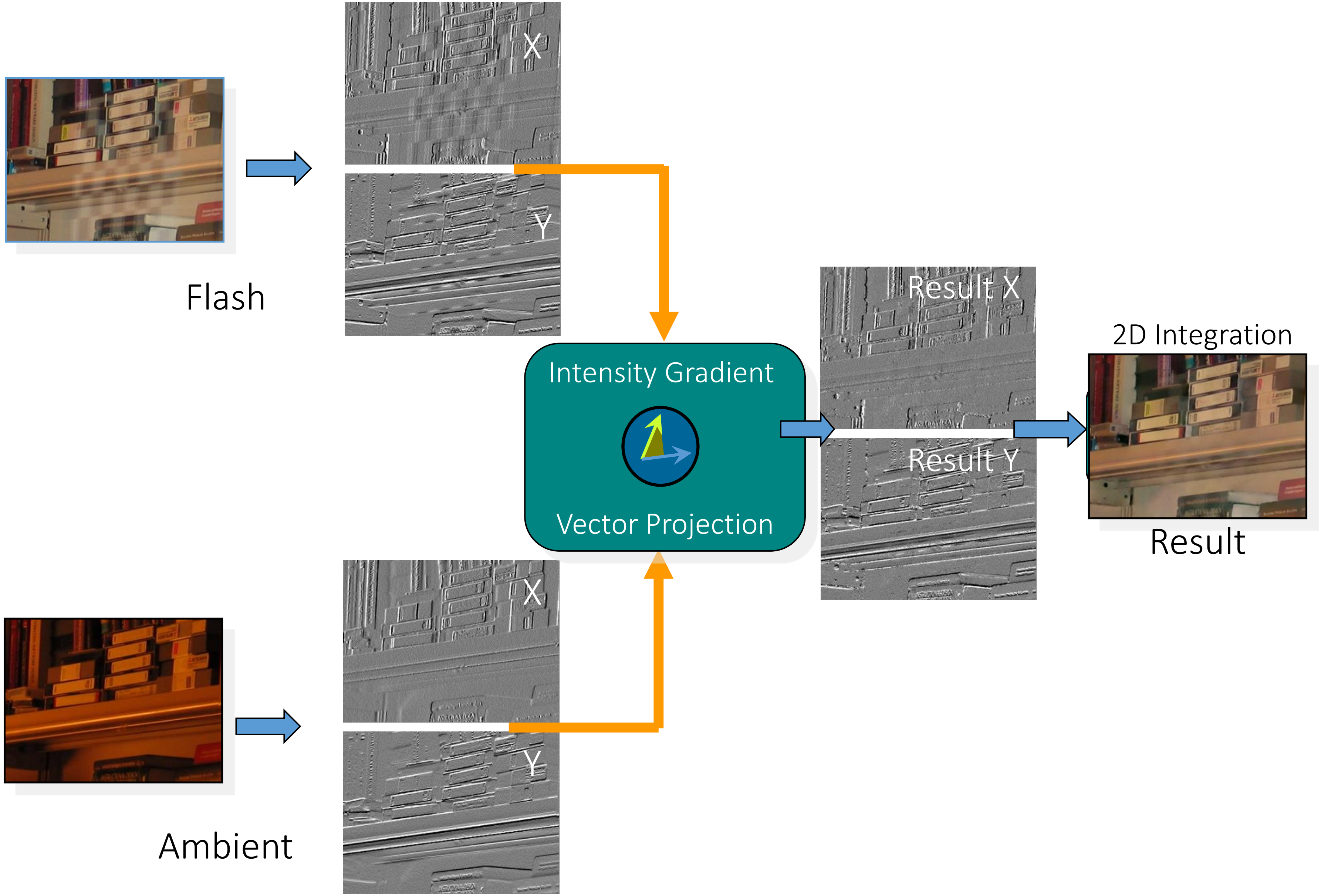
With reflections

Gradient projections



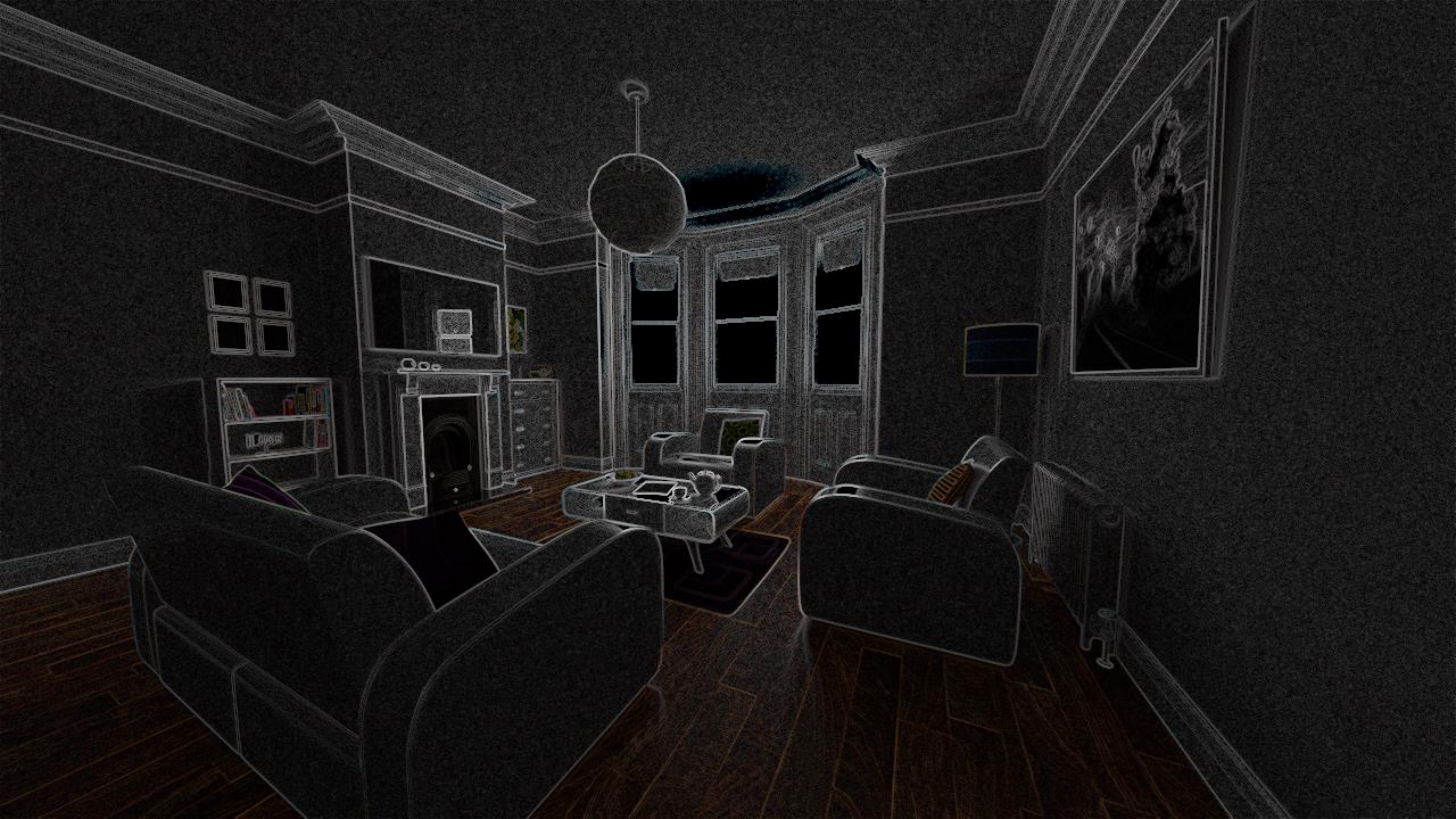


Flash/no-flash with gradient-domain processing



Gradient-domain rendering


Love

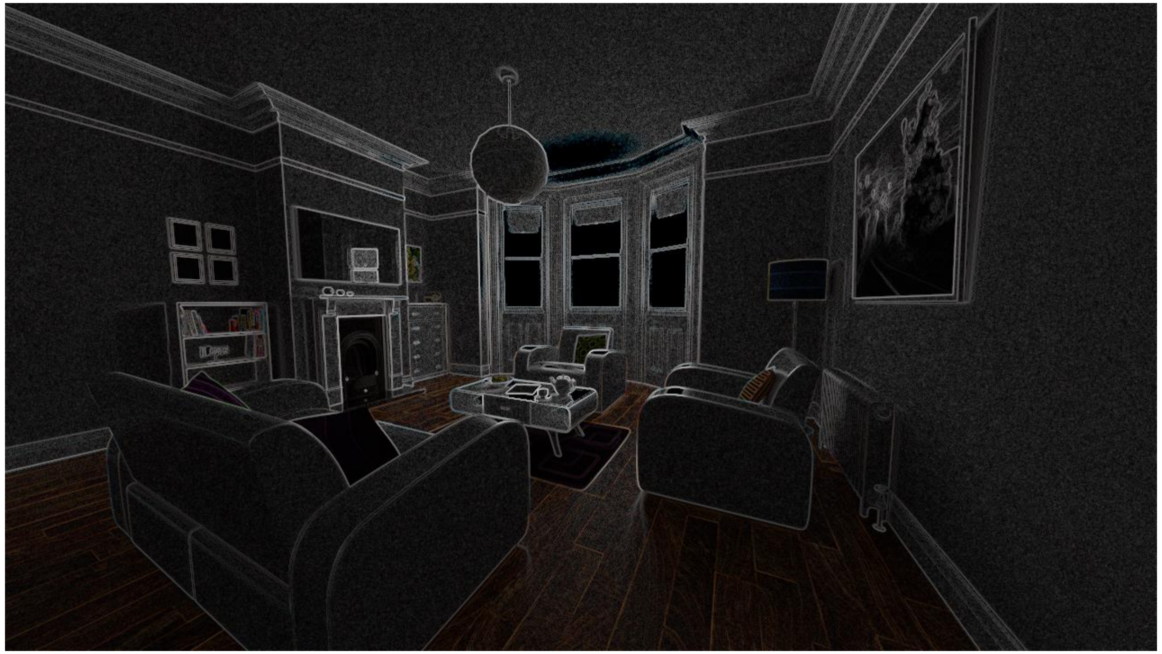


Primal domain

Gradient domain

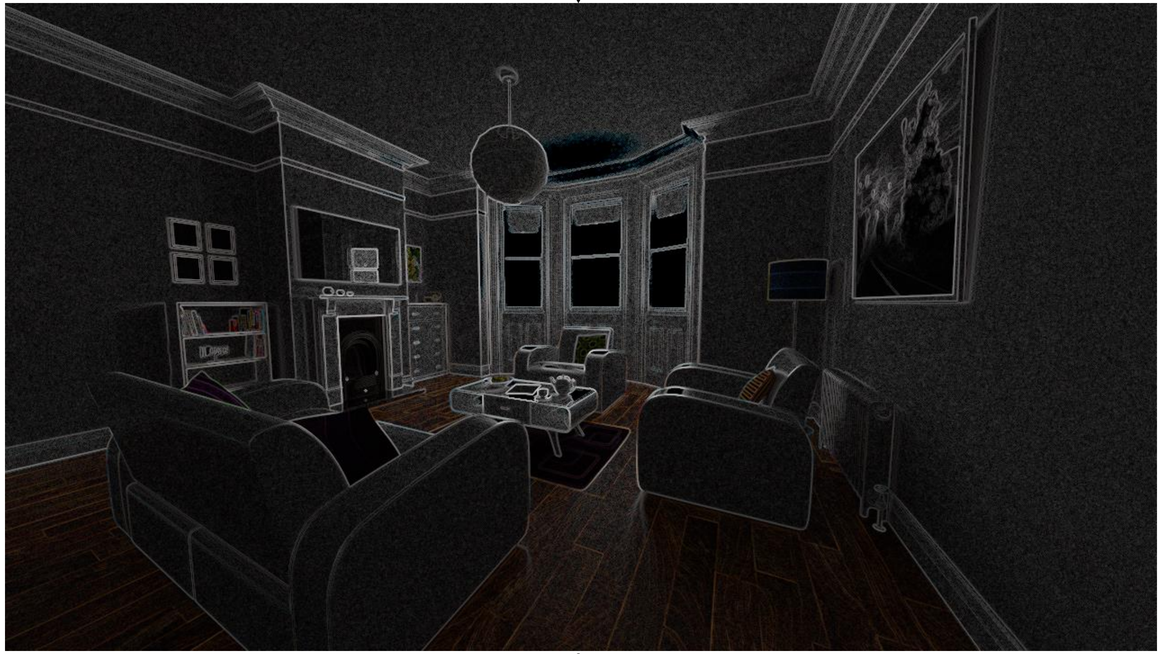

gradients of
natural images
are *sparse*
(close to zero
in most places)
Primal domain
Gradient domain

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)

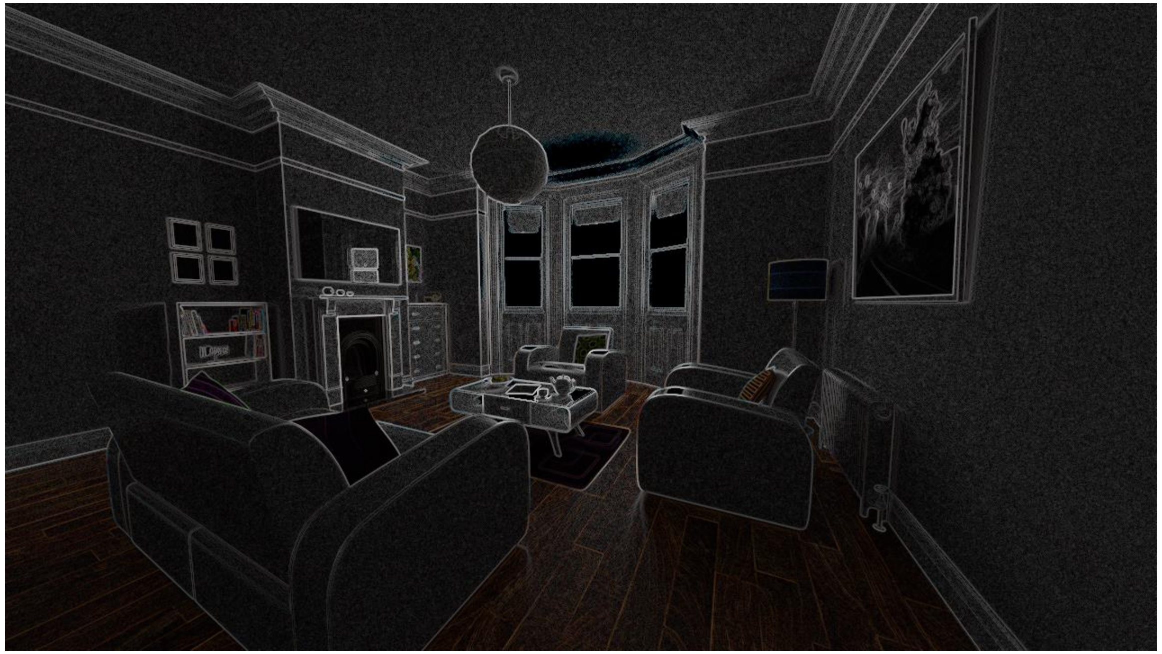


integration (e.g., Poisson reconstruction)

Rendering

Primal-domain rendering: simulate intensities directly

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

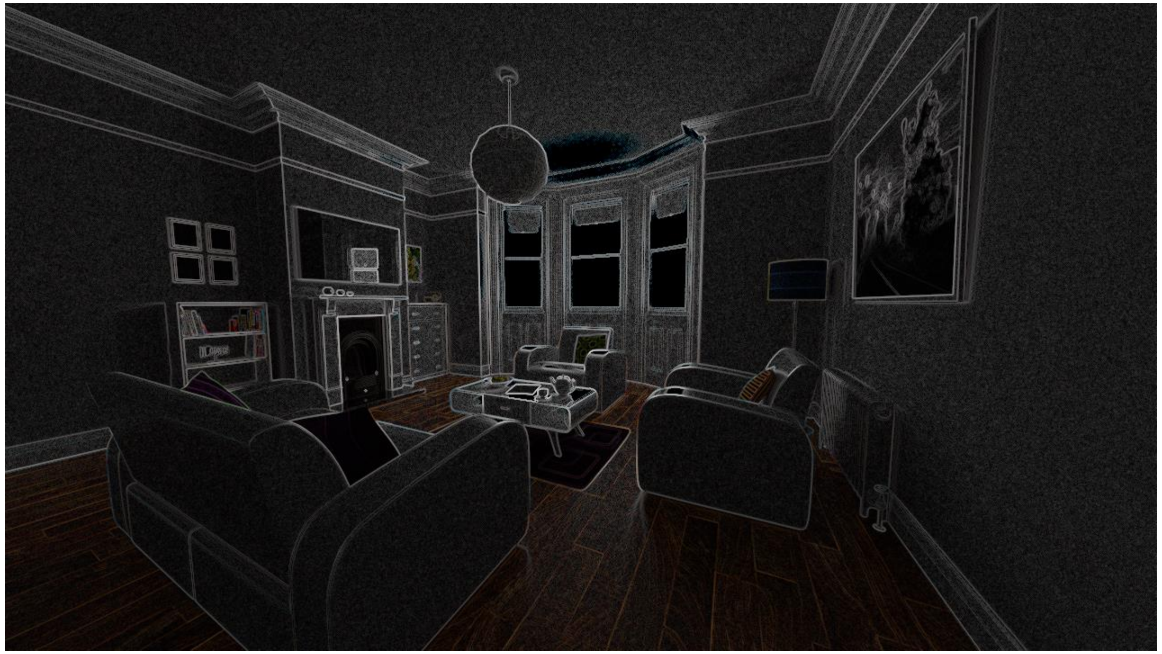


Why would gradient-domain rendering make sense?

Rendering

Primal-domain rendering: simulate intensities directly

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



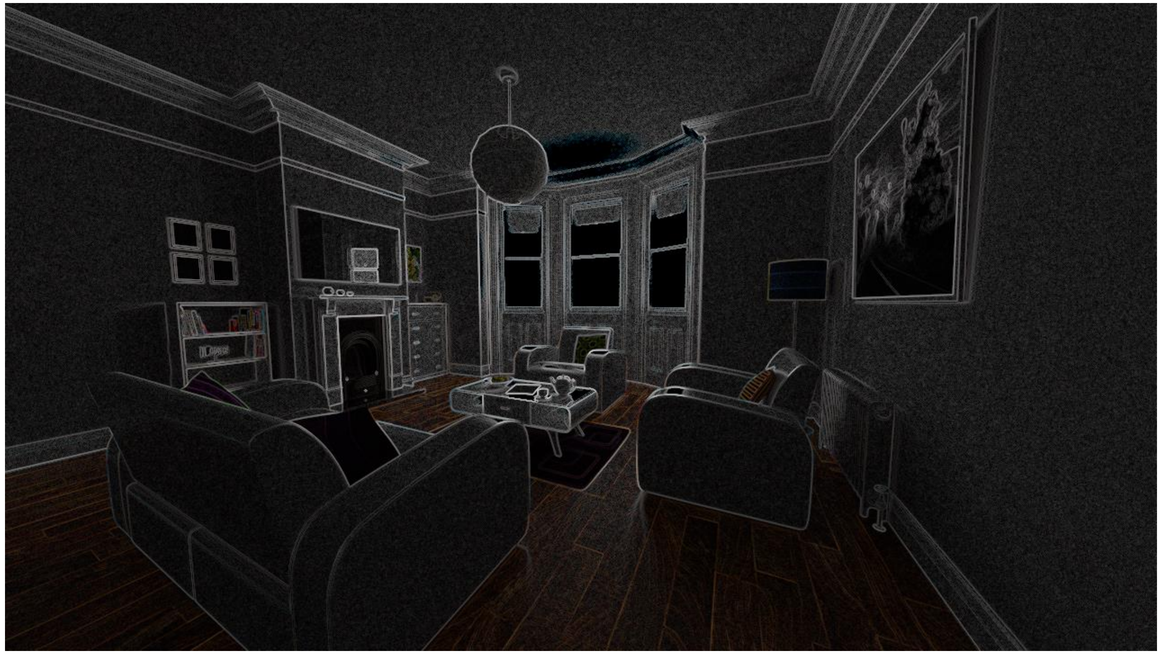
Why would gradient-domain rendering make sense?

- 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



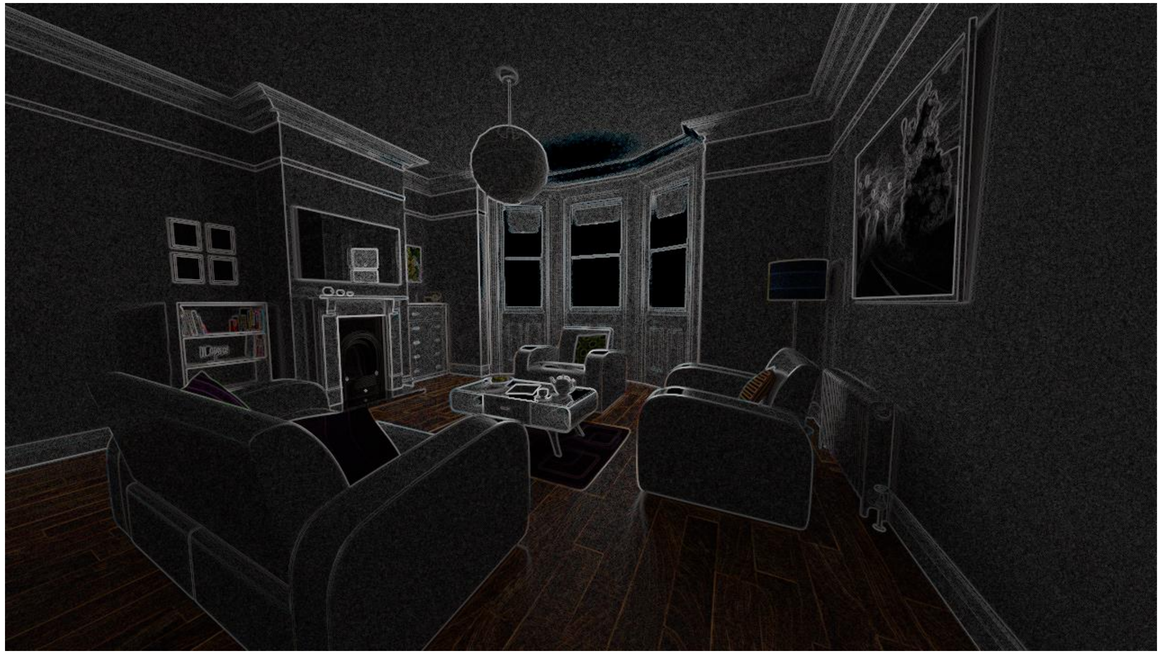
Why would gradient-domain rendering make sense? Why not all?

- 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



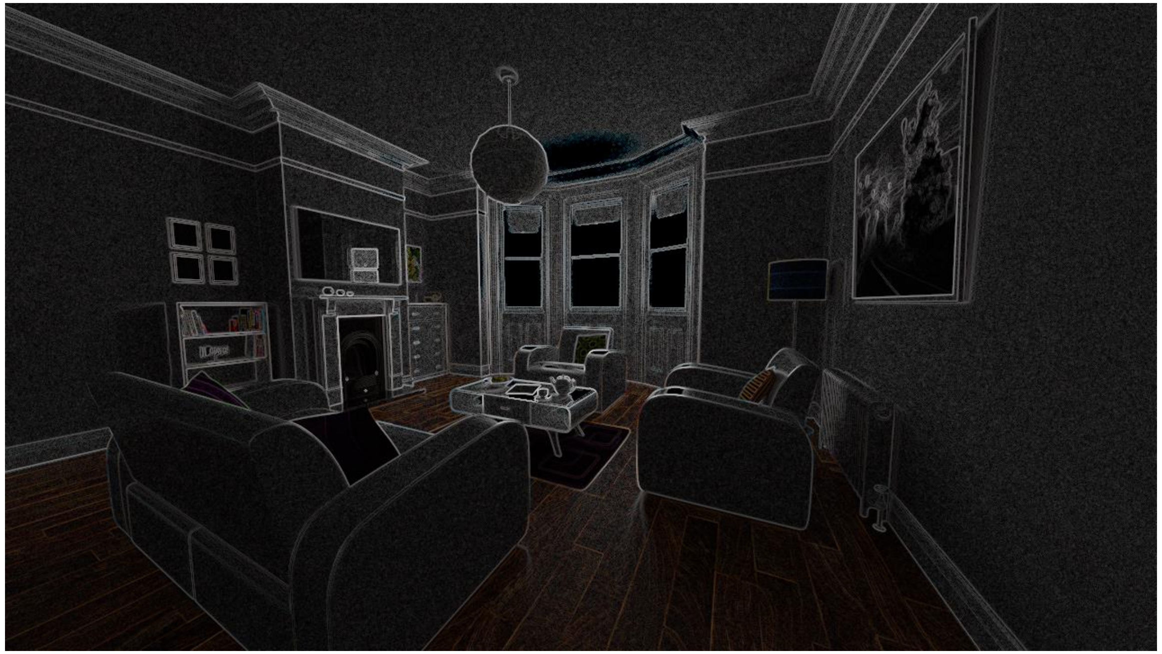
You still need to render a few sparse pixels (roughly one per “flat” region in the image) in primal domain, to use as boundary conditions when doing Poisson reconstruction.

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

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 when doing Poisson reconstruction.

- 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

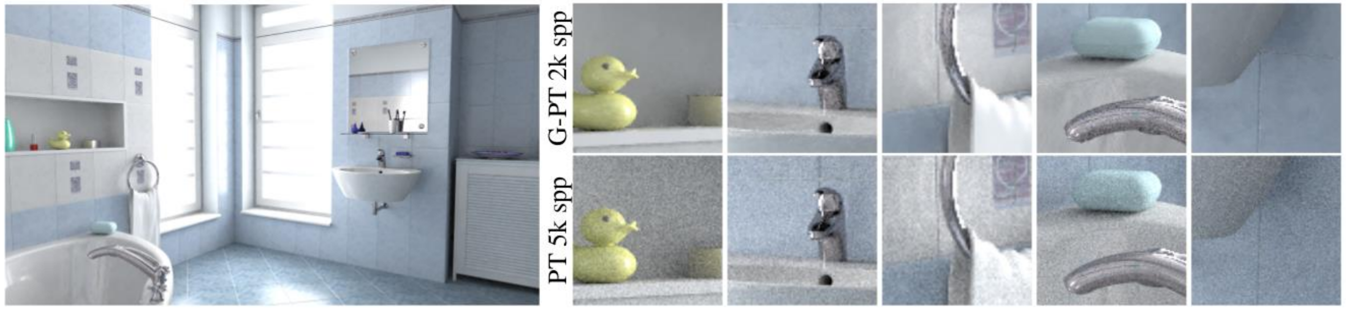


Figure 1: *Comparing gradient-domain path tracing (G-PT, L_1 reconstruction) to path tracing at equal rendering time (2 hours). In this time, G-PT draws about 2,000 samples per pixel and the path tracer about 5,000. G-PT consistently outperforms path tracing, with the rare exception of some highly specular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

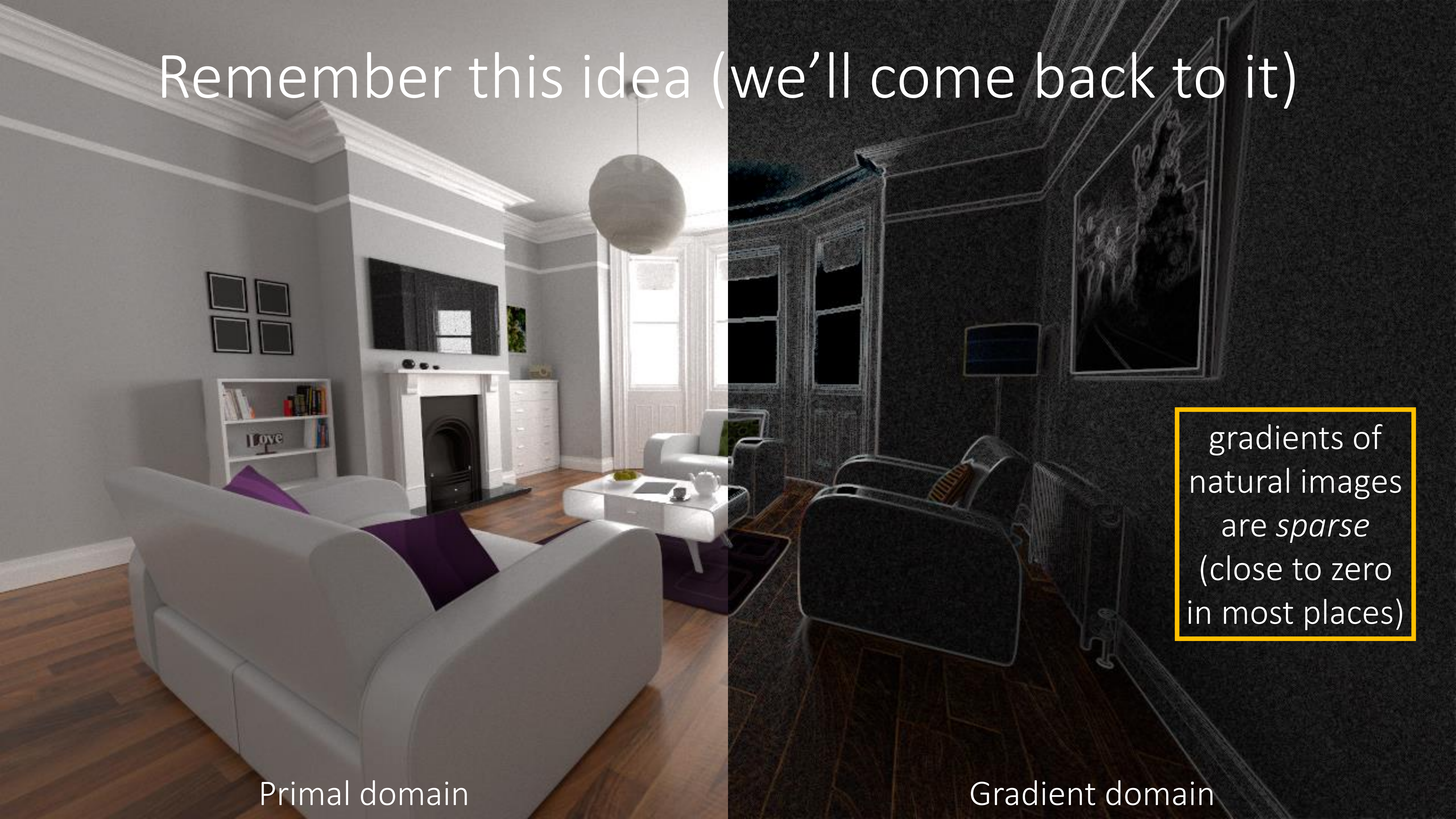
A lot of papers since SIGGRAPH 2013 (first introduction of gradient-domain rendering) that are looking to extend basically all primal-domain rendering algorithms to the gradient domain.

Does it help?

Gradient-domain path tracing (2 minutes)

Primal-domain path tracing (2 minutes)

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



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?

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?

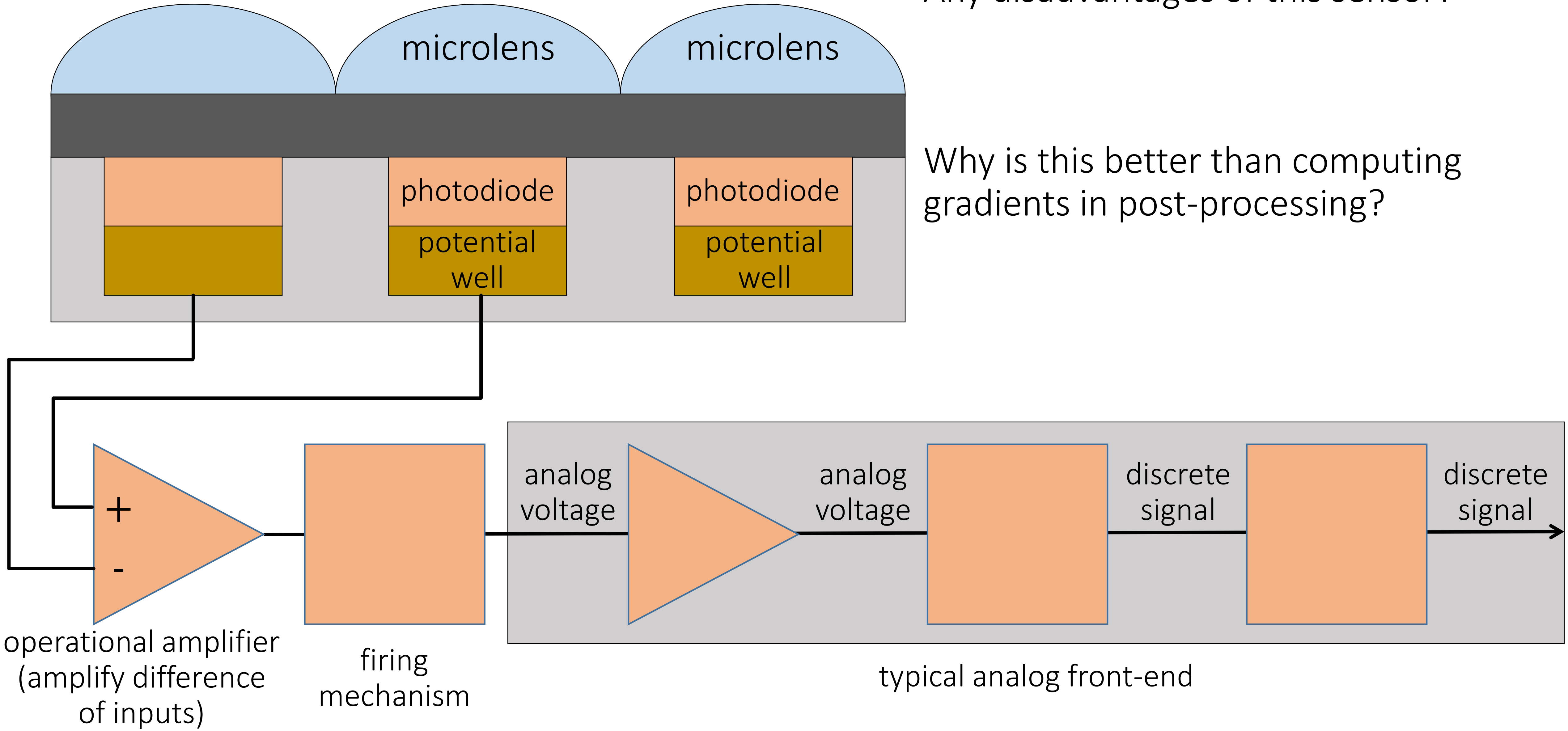
Change the sensor

Can you think how?

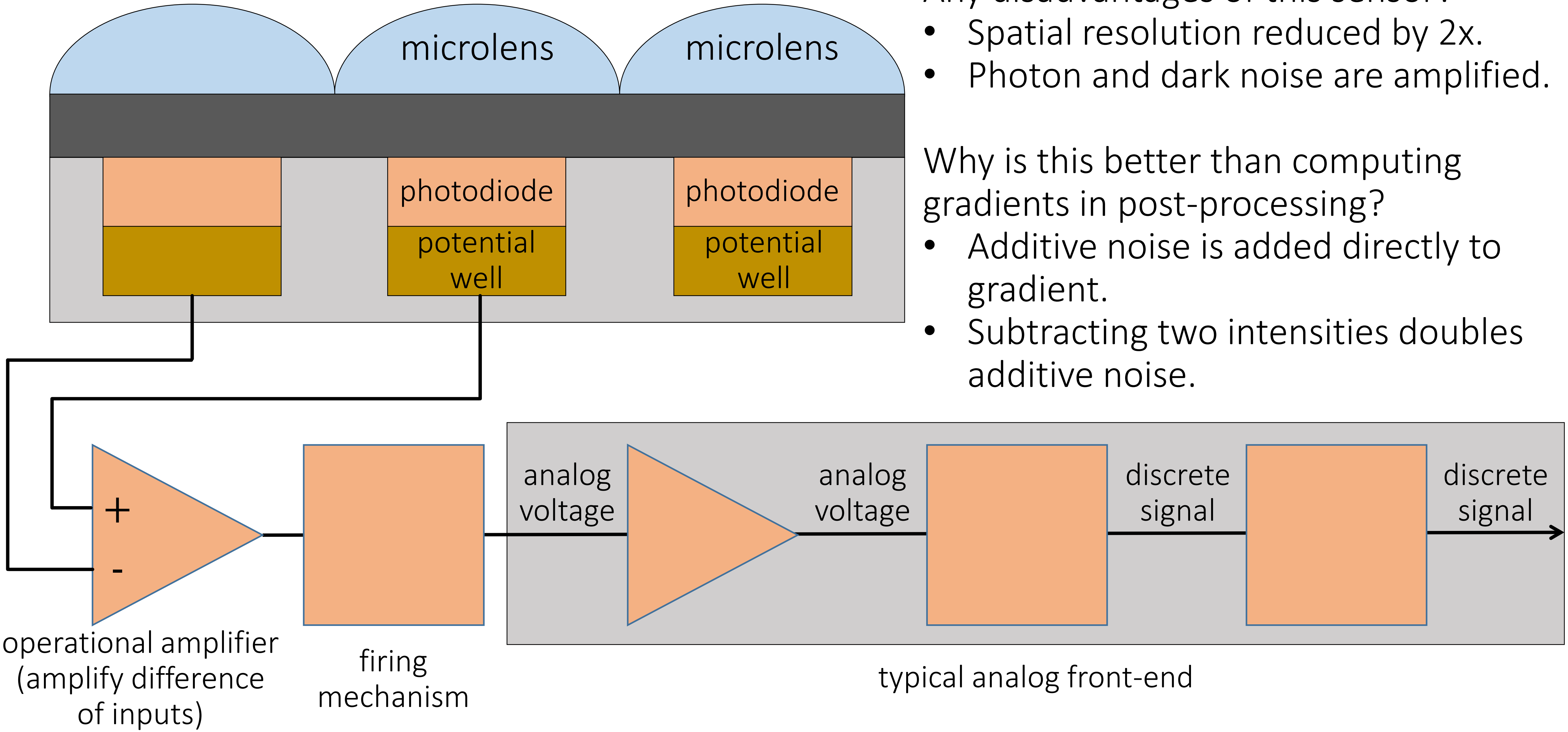
Change the sensor

Any disadvantages of this sensor?

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



Change the sensor



Any disadvantages of this sensor?

- Spatial resolution reduced by 2x.
- Photon and dark noise are amplified.

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

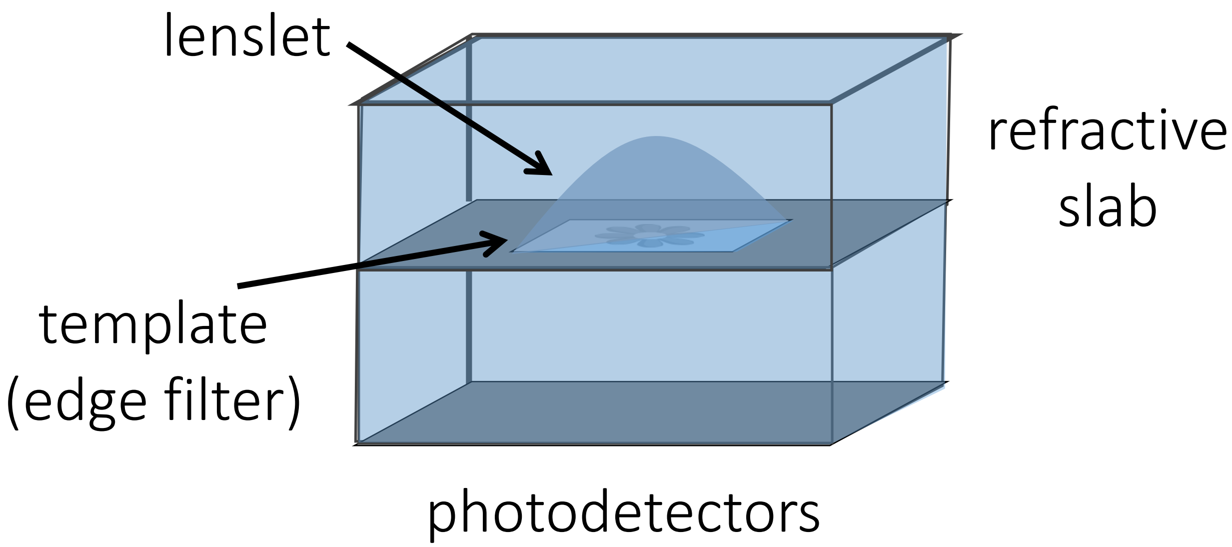
- Additive noise is added directly to gradient.
- Subtracting two intensities doubles additive noise.

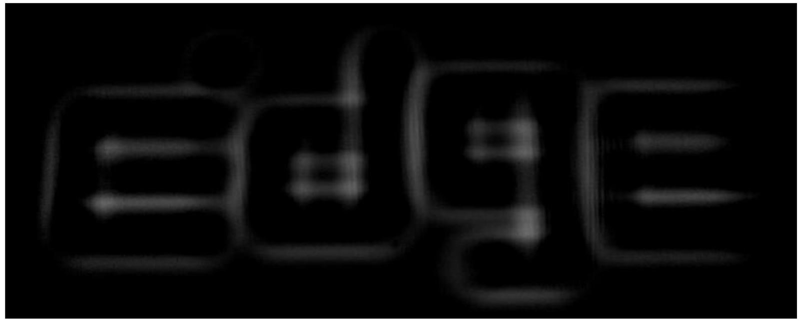
Change the optics

Can you think how?

Change the optics

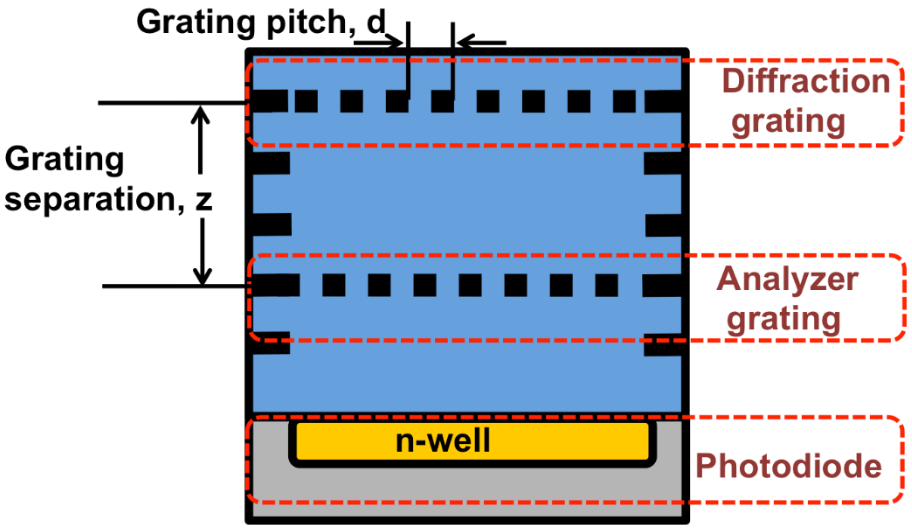
Optical filtering

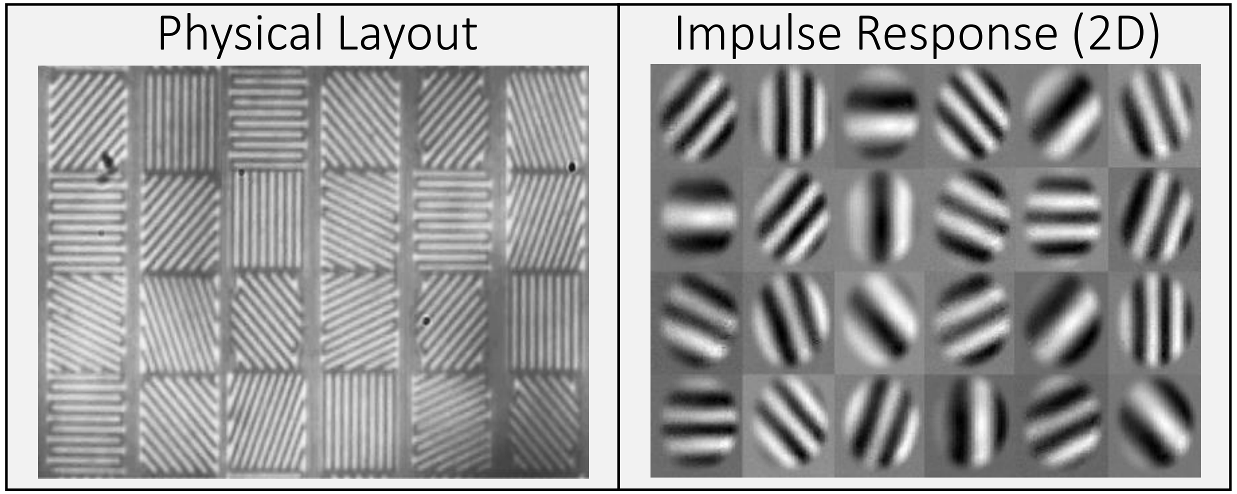




resulting image

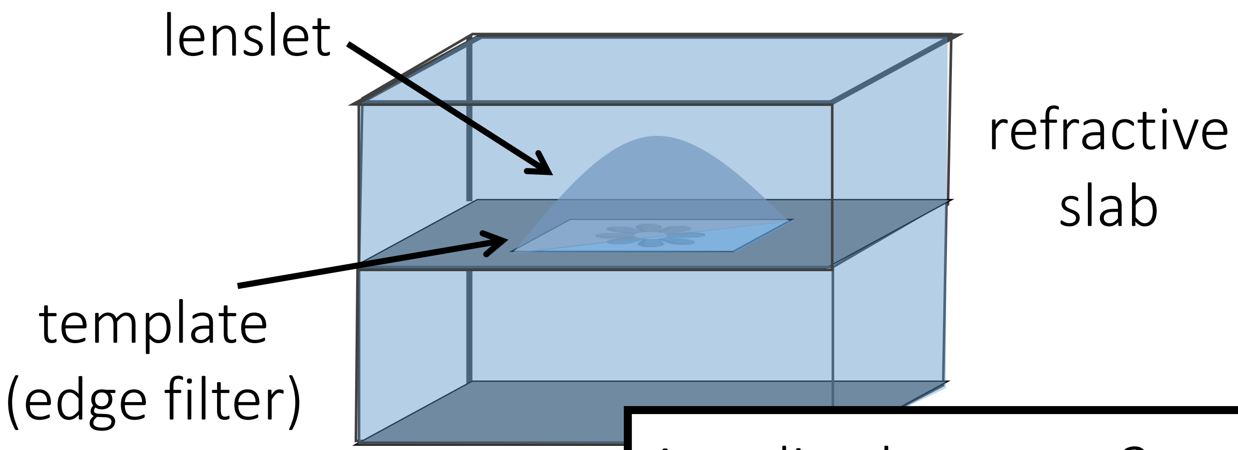
Angle-sensitive pixels





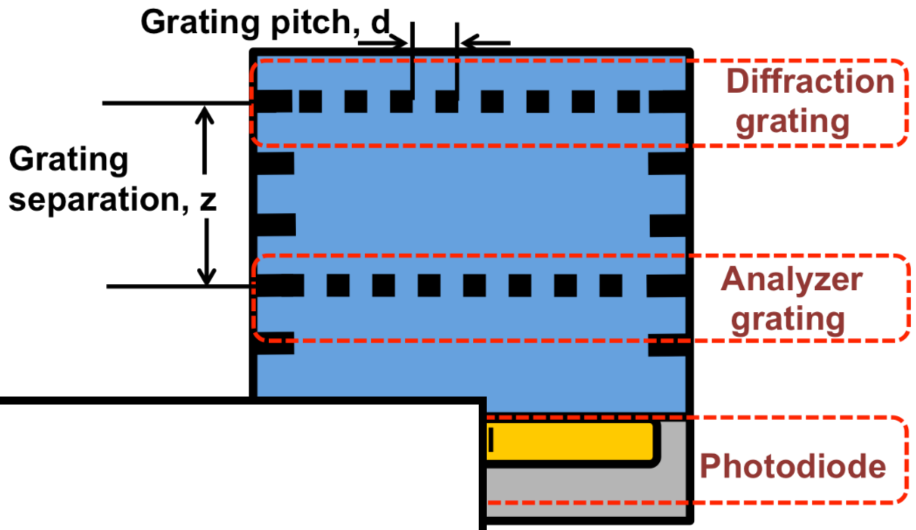
Change the optics

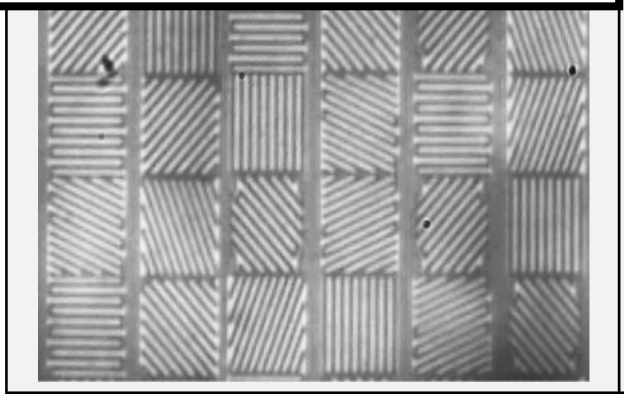
Optical filtering

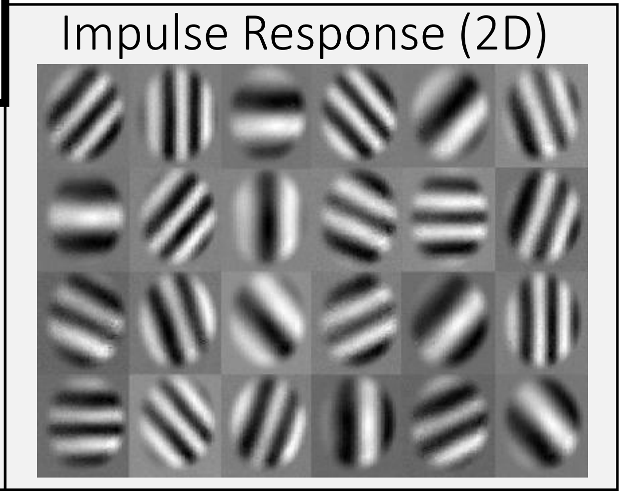


resulting image

Angle-sensitive pixels



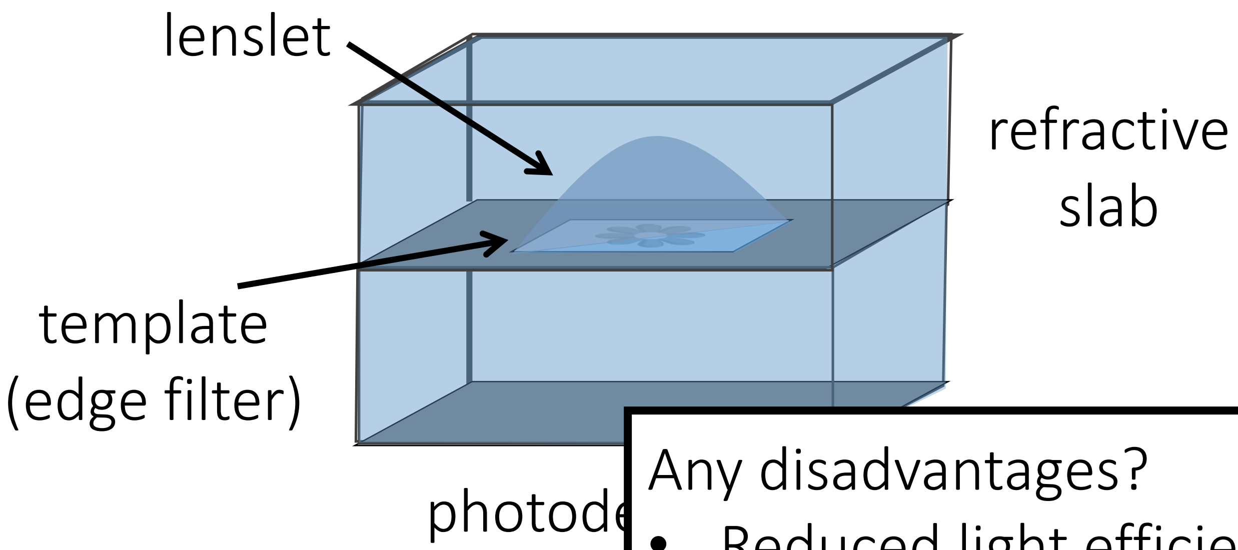


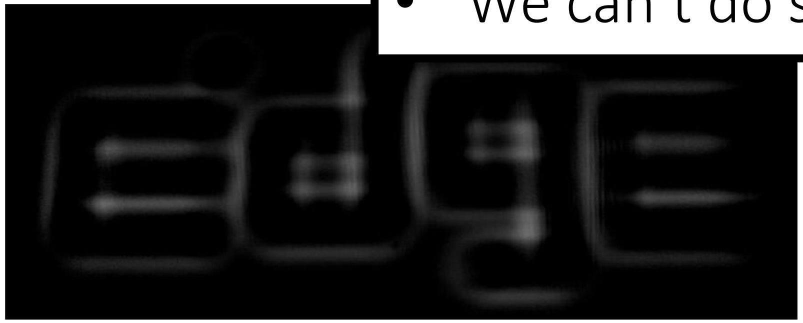


Any disadvantages?

Change the optics

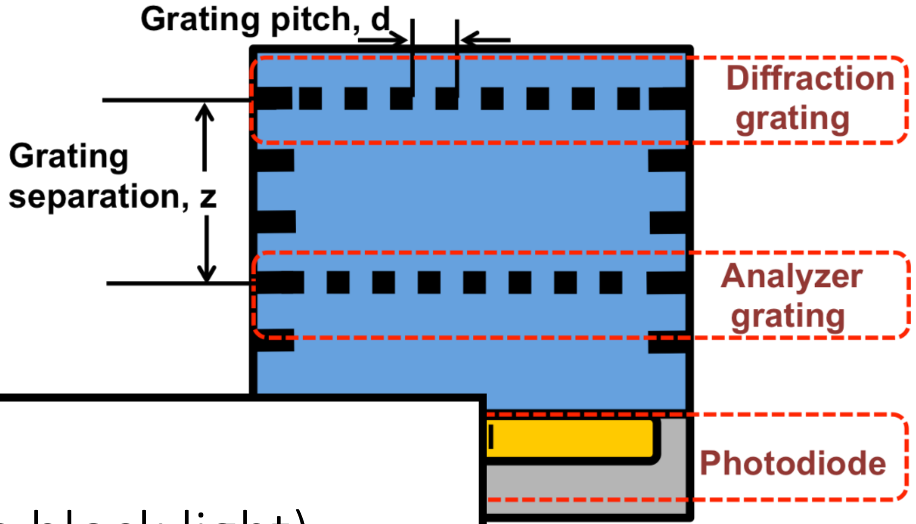
Optical filtering

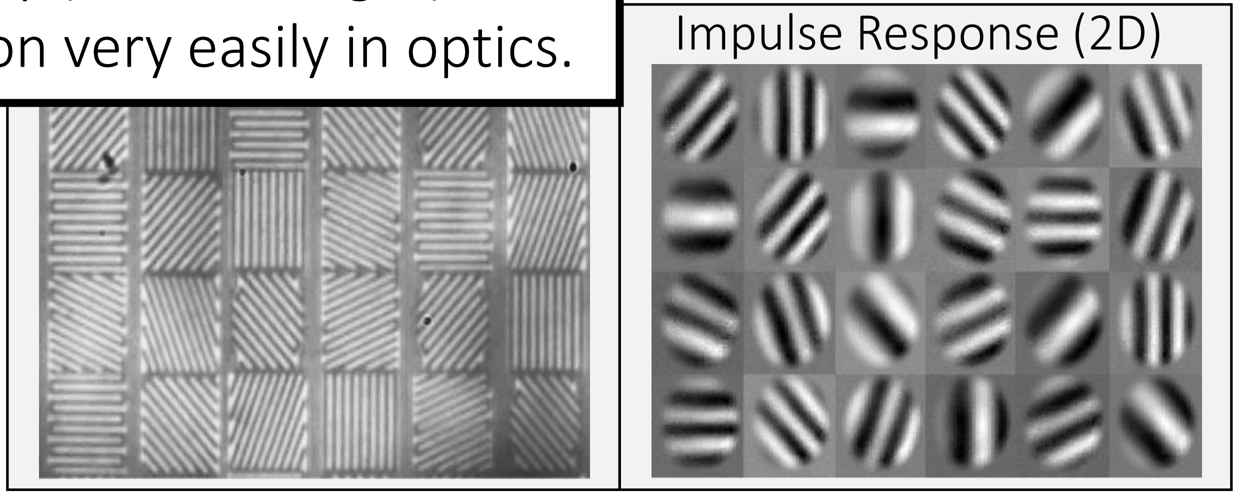




resulting image

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?

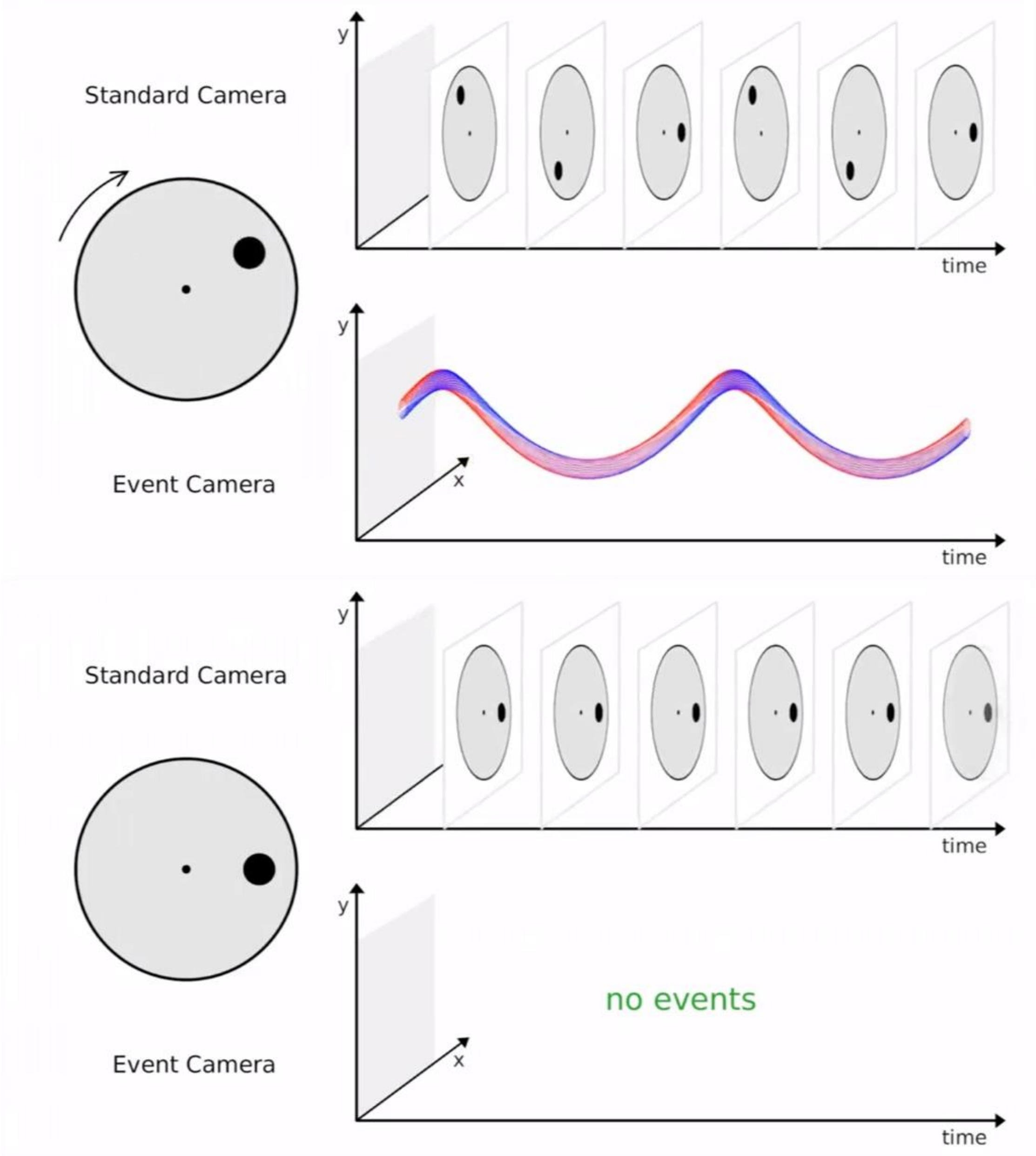
- You need to use a Poisson solver to reconstruct the image from the measured gradients.

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

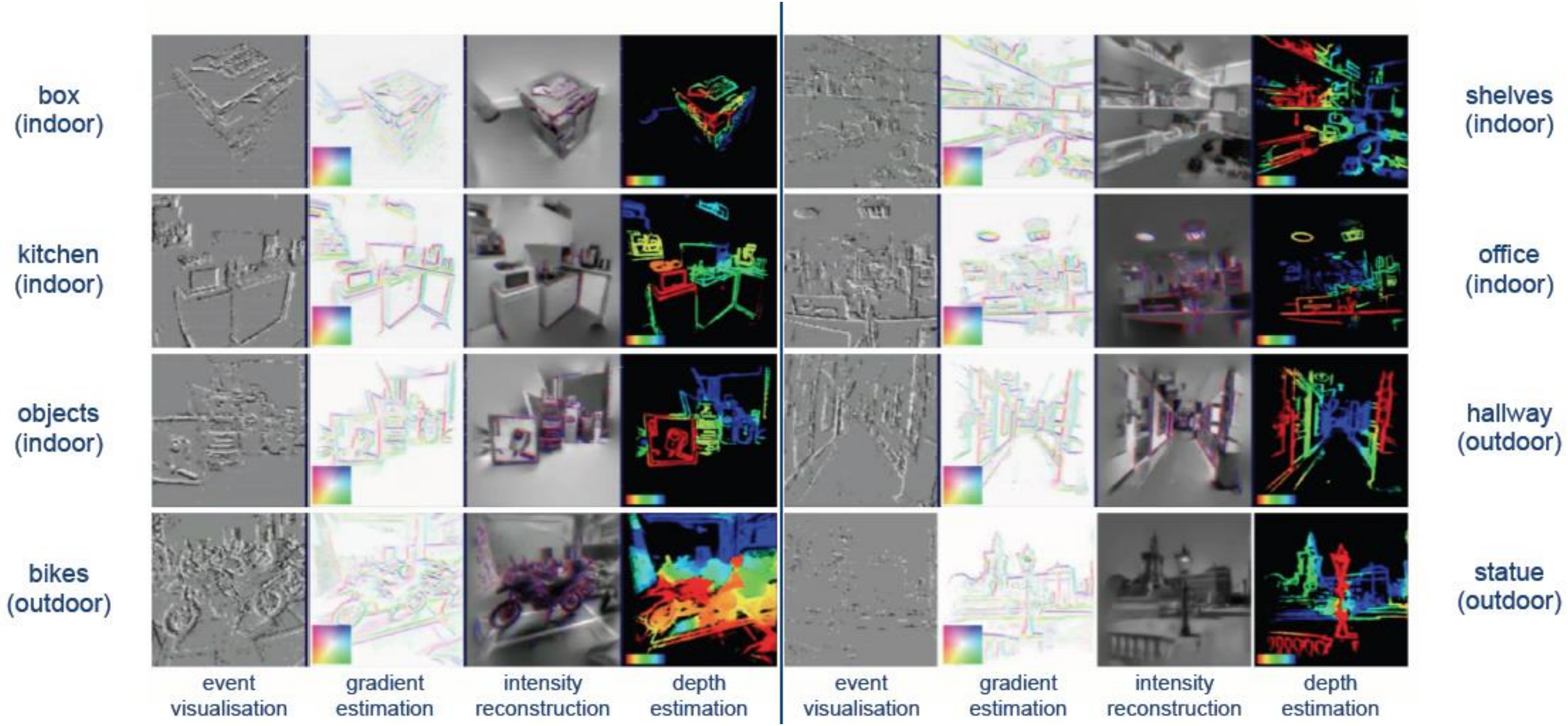
High-speed output on a quadcopter:

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.
  A paper discussing both Poisson solvers and projection-based methods for integration in a unified way, along with suggesting various generalizations.
- Szeliski, "Locally adapted hierarchical basis preconditioning," SIGGRAPH 2006.
  A standard reference on multi-grid and preconditioning techniques for accelerating the Poisson solver.
- 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 conjugate gradient solvers for large linear systems.
- Briggs et al., "A multigrid tutorial," SIAM 2000.
  A nice 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, including tone-mapping, colorization, converting to grayscale, edge enhancement, image abstraction and non-photorealistic rendering.
- 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*.
- 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!).