Multiple Exposure Photo by Christoffer Relander

BBM444

FUNDAMENTALS OF COMPUTATIONAL PHOTOGRAPHY

Lecture #06 – Gradient-Domain Image Processing

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Erkut Erdem // Hacettepe University // Spring 2024

Today's Lecture

- Gradient-domain image processing
- Basics on images and gradients
- Integrable vector fields
- Poisson blending
- Flash/no-flash photography
- Gradient-domain rendering and cameras

Disclaimer: The material and slides for this lecture were borrowed from

—Ioannis Gkioulekas' 15-463/15-663/15-862 "Computational Photography" class

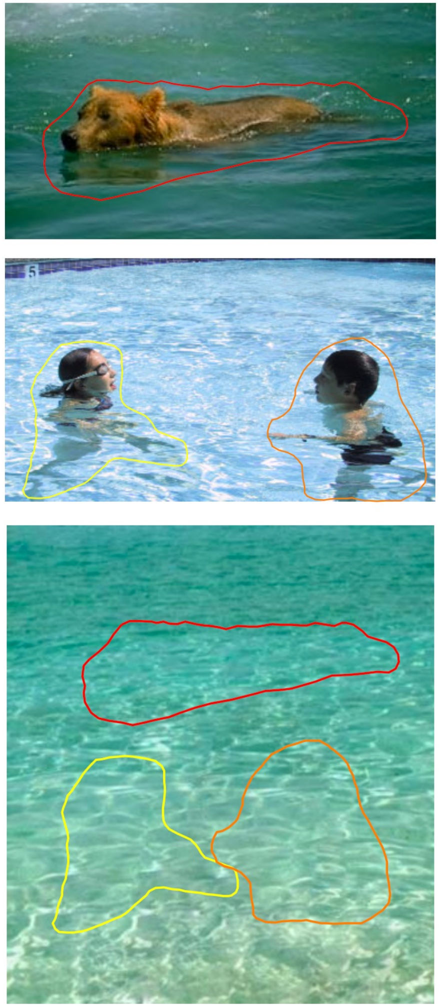
—Amit Agrawal's slides on "Gradient-Domain Based Flash/No-flash Photography"

—Adrien Gruson's slides on "Gradient-Domain Rendering"

—Davide Scaramuzza's tutorial on "Event-based Cameras"

Gradient-domain image processing

Application: Poisson blending



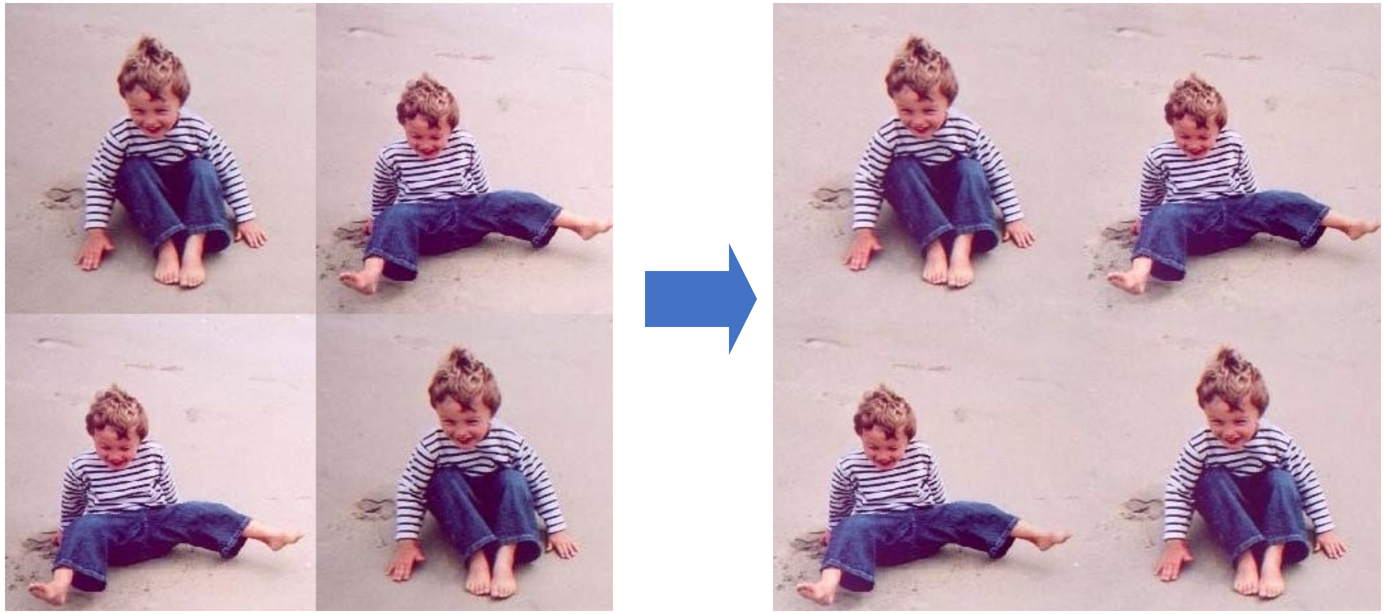
originals

copy-paste

Poisson blending

More applications

Removing Glass Reflections



Seamless Image Stitching

Yet more applications

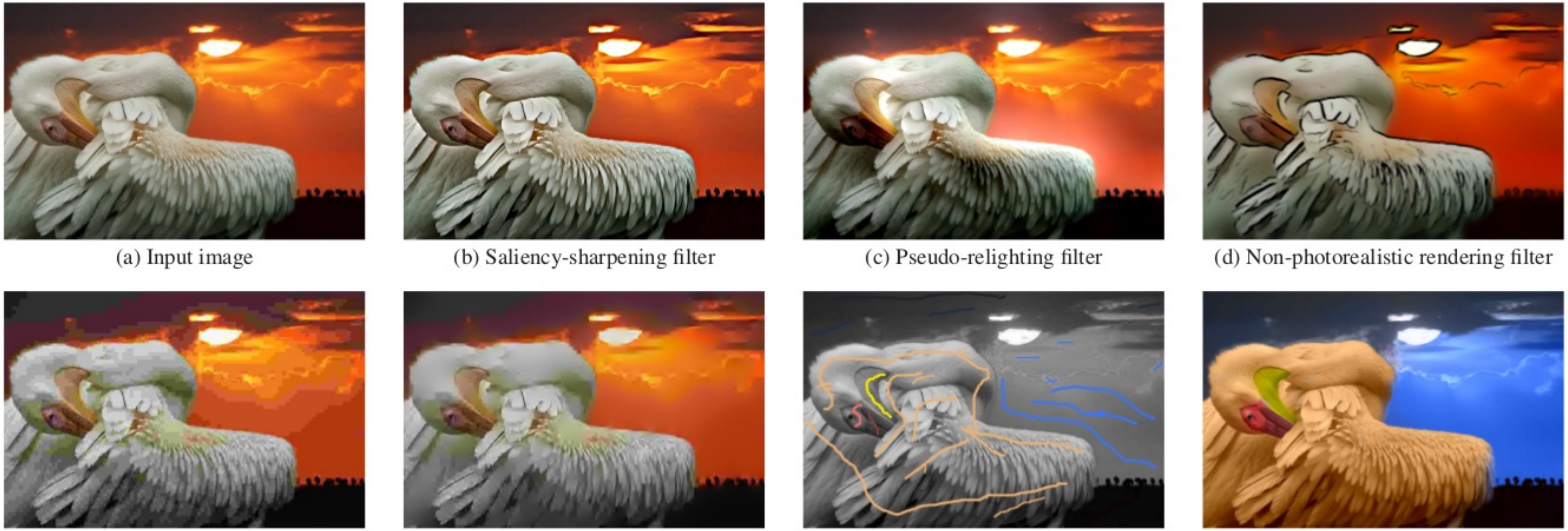
Fusing day and night photos

Tonemapping

Entire suite of image editing tools

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

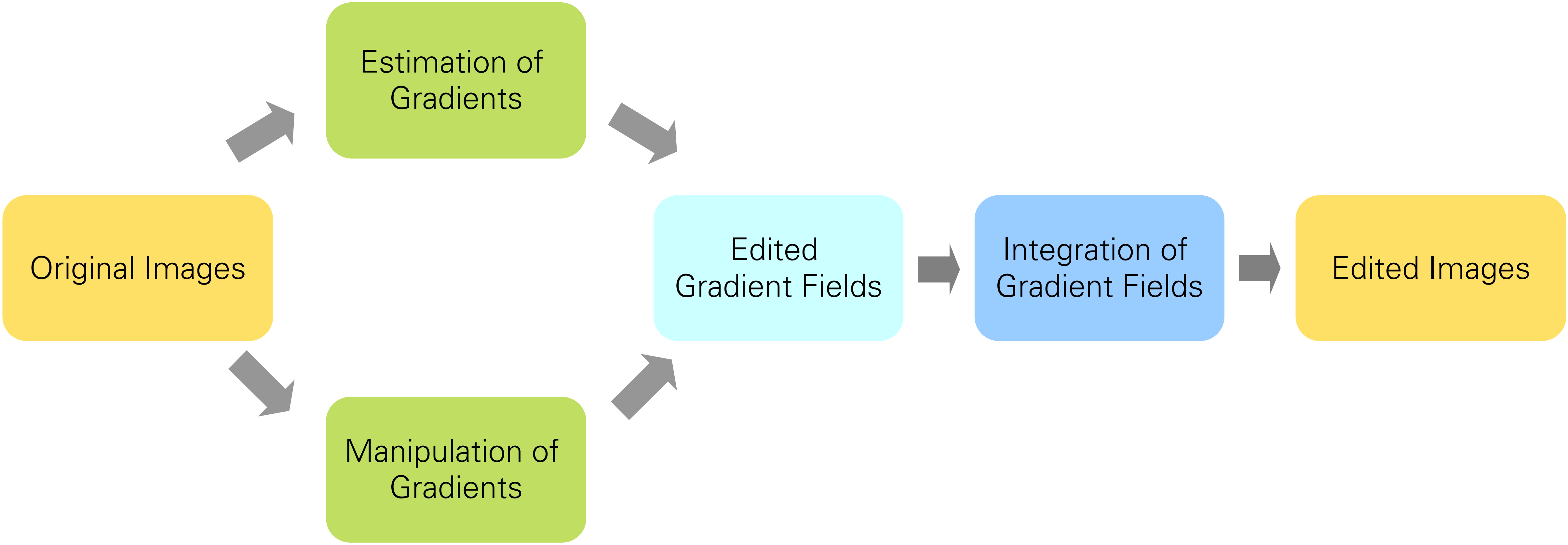
Pravin Bhat[1] C. Lawrence Zitnick[2] Michael Cohen[1,2] Brian Curless[1]
[1]University of Washington [2]Microsoft Research



(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



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.

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

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?
- 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.

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

Think of this as a 2D vector.

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

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

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 <u>vector</u> 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 <u>scalar</u> 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 <u>vector</u> field.

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.

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 <u>vector</u> field.

Divergence: inner product of nabla with a vector field.

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

This is a <u>scalar</u> field.

Curl: cross product of nabla with a vector field.

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

This is a <u>vector</u> field.

This is a <u>scalar</u> 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)

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

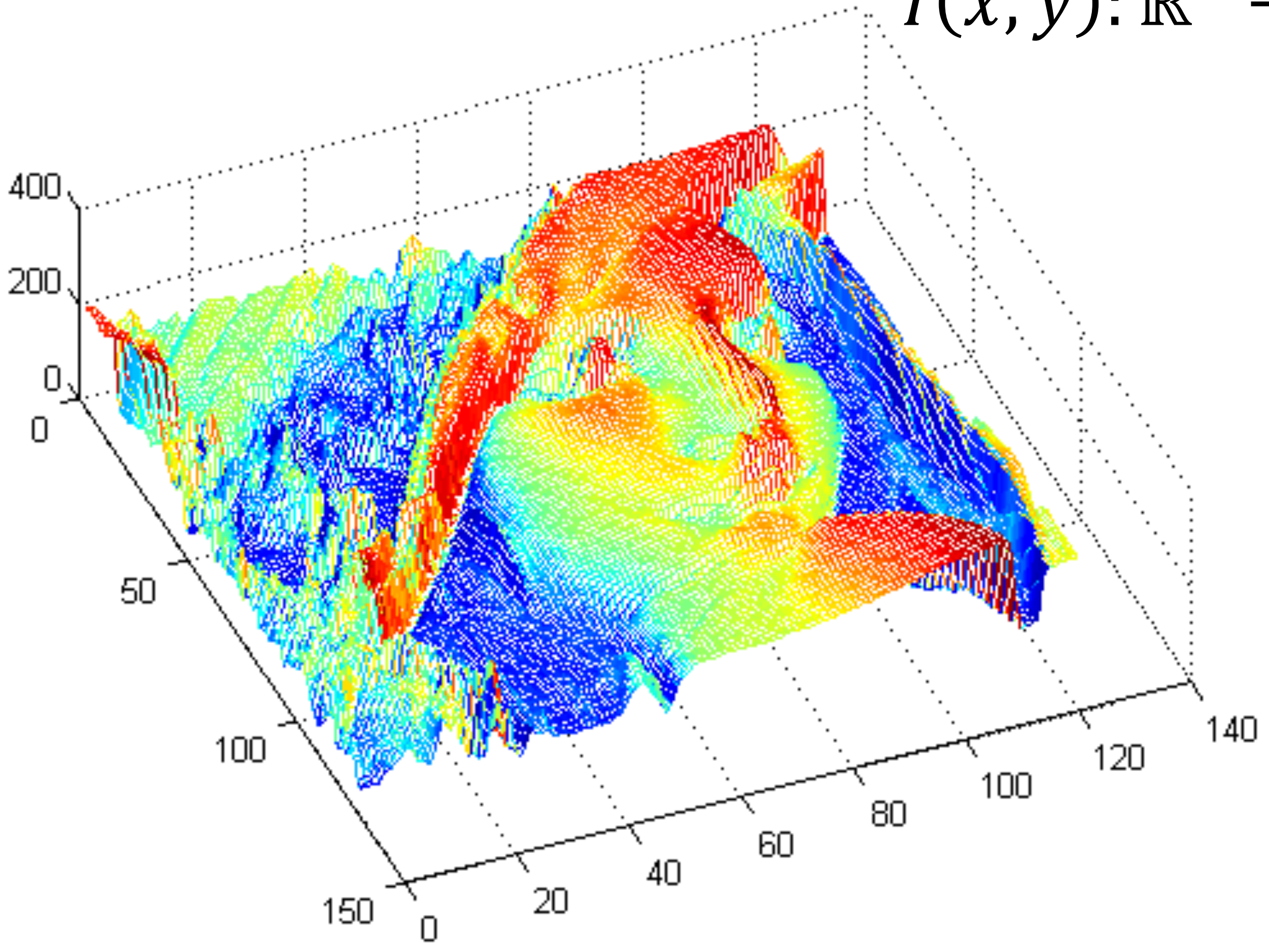


Image gradients

Convert the scalar field into a vector field through differentiation.

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

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:

$$\underbrace{\begin{bmatrix} 1 & -1 \end{bmatrix} * \begin{bmatrix} 1 & -1 \end{bmatrix}} + \underbrace{\begin{bmatrix} 1 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 \\ -1 \end{bmatrix}} = \quad ?$$

What is this? What is this?

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

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

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:

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

Laplacian filter

0	1	0
1	-4	1
0	1	0

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

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

- Image gradients are very informative!

Application - Seam Carving

[Shai & Avidan, SIGGRAPH 2007]

Slide credit: K. Grauman

Application - Seam Carving

Content-aware resizing

Traditional resizing

[Shai & Avidan, SIGGRAPH 2007]

Slide credit: K. Grauman

Application - Seam Carving

Seam Carving: Main idea

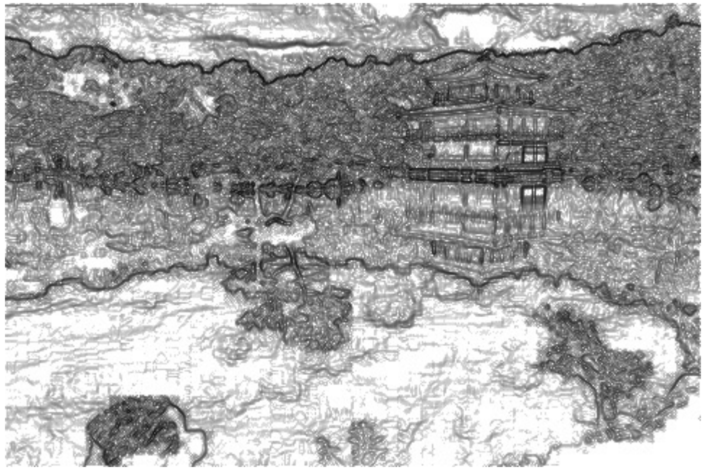
Content-aware resizing

Intuition:

- Preserve the most "interesting/important" content
  → Prefer to remove pixels with low gradient energy
- To reduce or increase size in one dimension, remove irregularly shaped "seams"
  → Optimal solution via dynamic programming.

Slide credit: K. Grauman

Seam Carving: Main idea

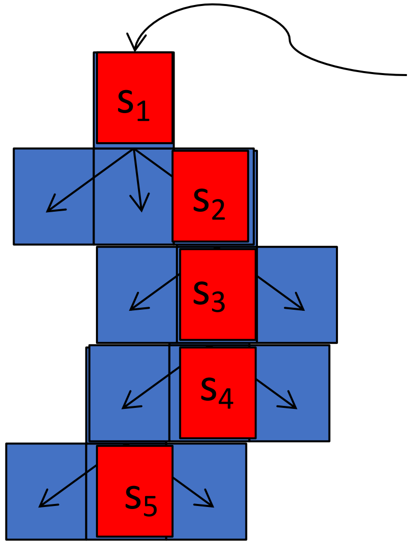


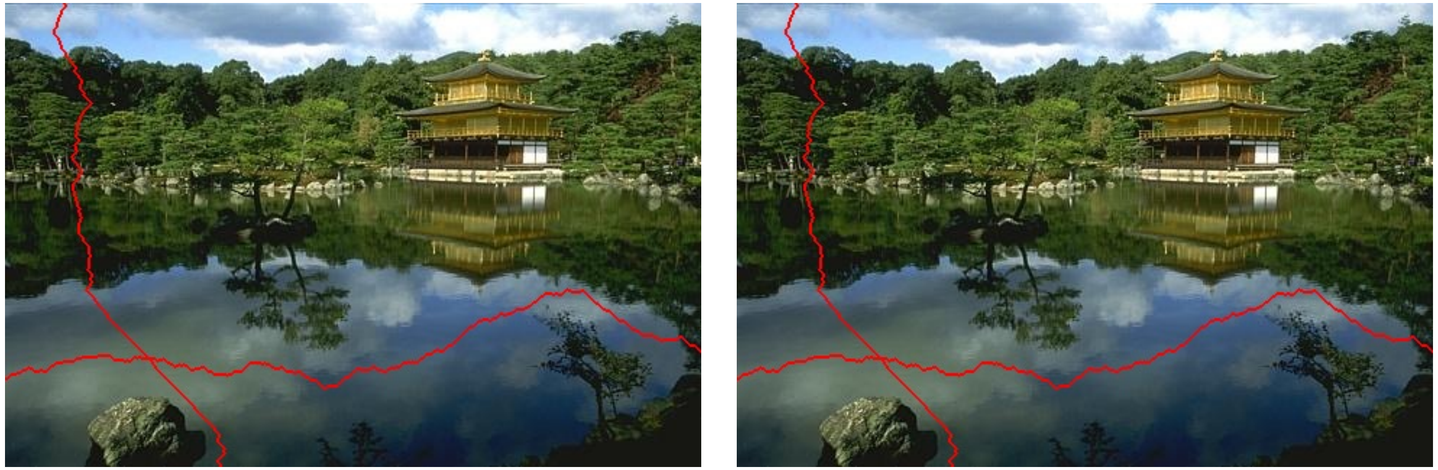
$$Energy(f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

- Want to remove seams where they won't be very noticeable:
- Measure "energy" as gradient magnitude
- Choose seam based on **minimum total energy path** across image, subject to 8-connectedness.

Slide credit: K. Grauman

Seam Carving: Algorithm





$$Energy(f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

- Let a vertical seam **s** consist of h positions that form an 8-connected path.
- Let the cost of a seam be: $Cost(\mathbf{s}) = \sum_{i=1}^{h} Energy(f(s_i))$
- Optimal seam minimizes this cost.
- Compute it efficiently with dynamic programming: $\mathbf{s}^* = \min_{\mathbf{s}} Cost(\mathbf{s})$

Slide credit: K. Grauman

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

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?
- Can we go in the opposite direction, from gradients to images?

Vector field integration

Two fundamental questions:

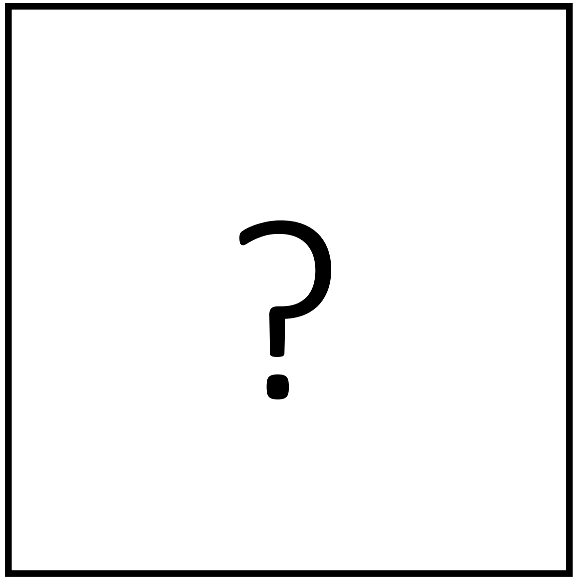
- When is integration of a vector field possible?

- 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 I?



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

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 should be 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:

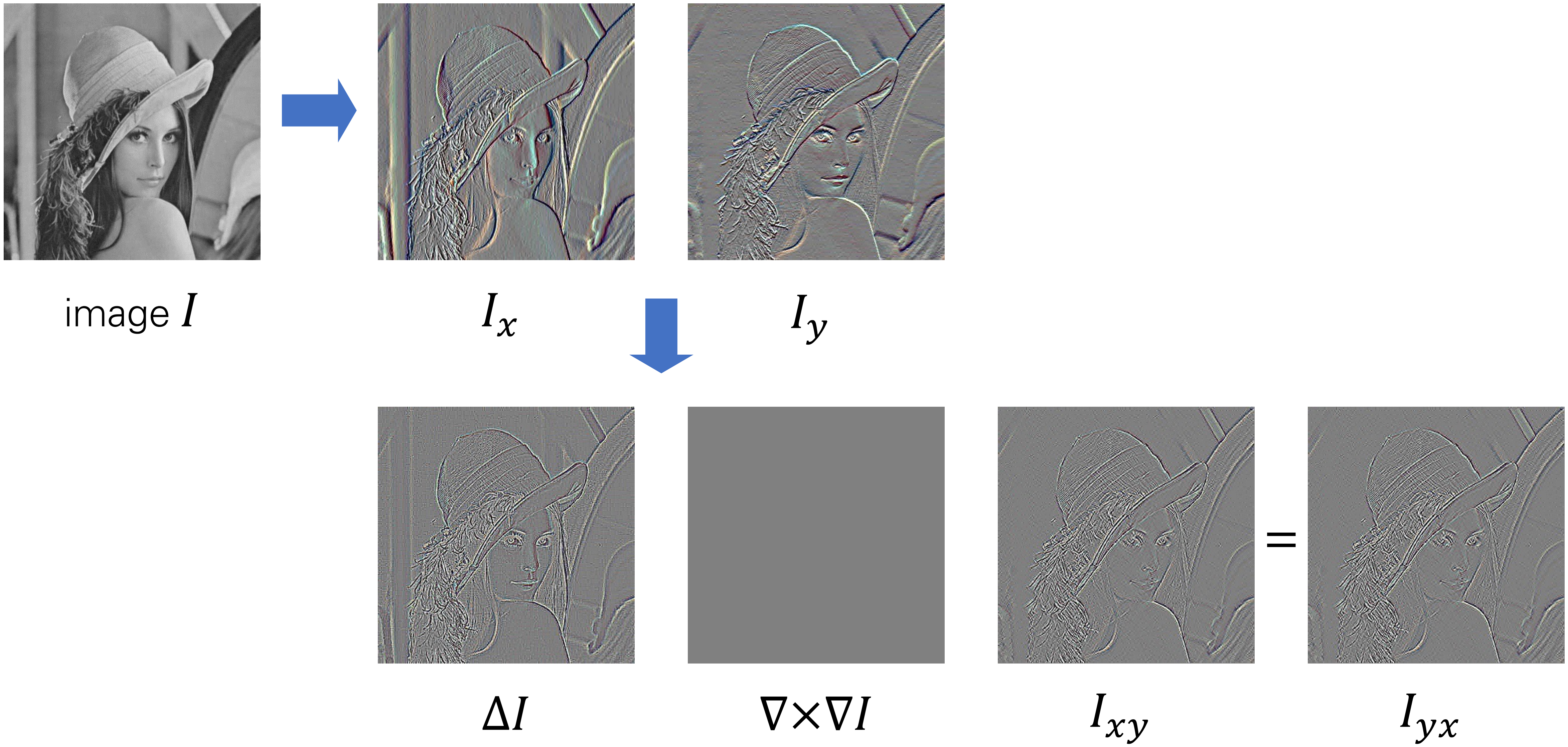
$$\nabla \times \nabla I = I_{yx} - I_{xy} = 0$$

What does that mean intuitively?

- Same result independent of order of differentiation.

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

Demonstration



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 (u, v), can we always integrate it into a scalar field I?

?

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

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

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

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

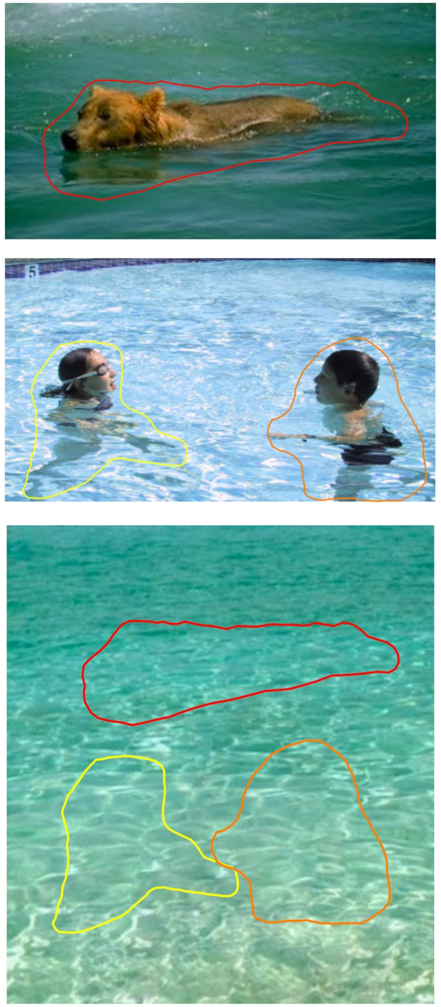
- Manipulation of 3D gradients
  Applications: mesh editing, video operations

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

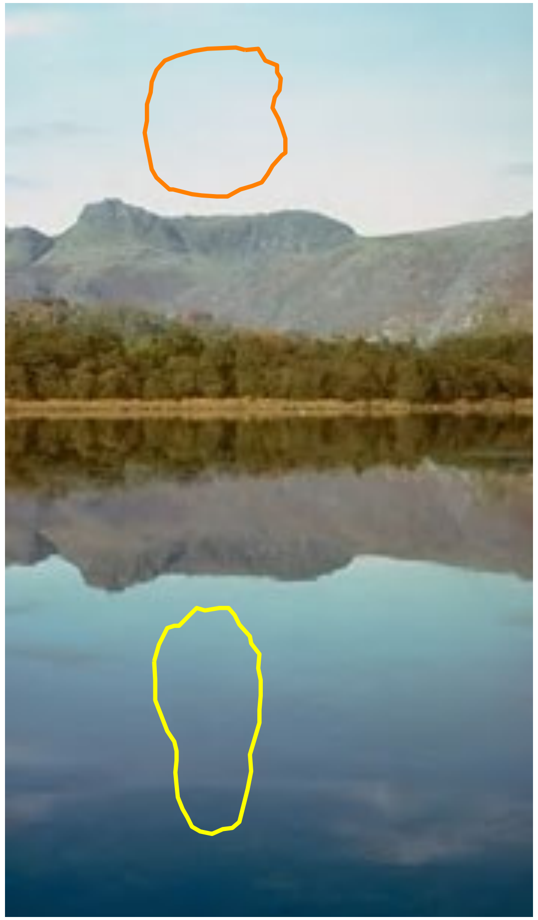
copy-paste

Poisson blending

Key idea

When blending, retain the <u>gradient</u> information as best as possible

source



destination

copy-paste

Poisson blending

Definitions and notation

Notation

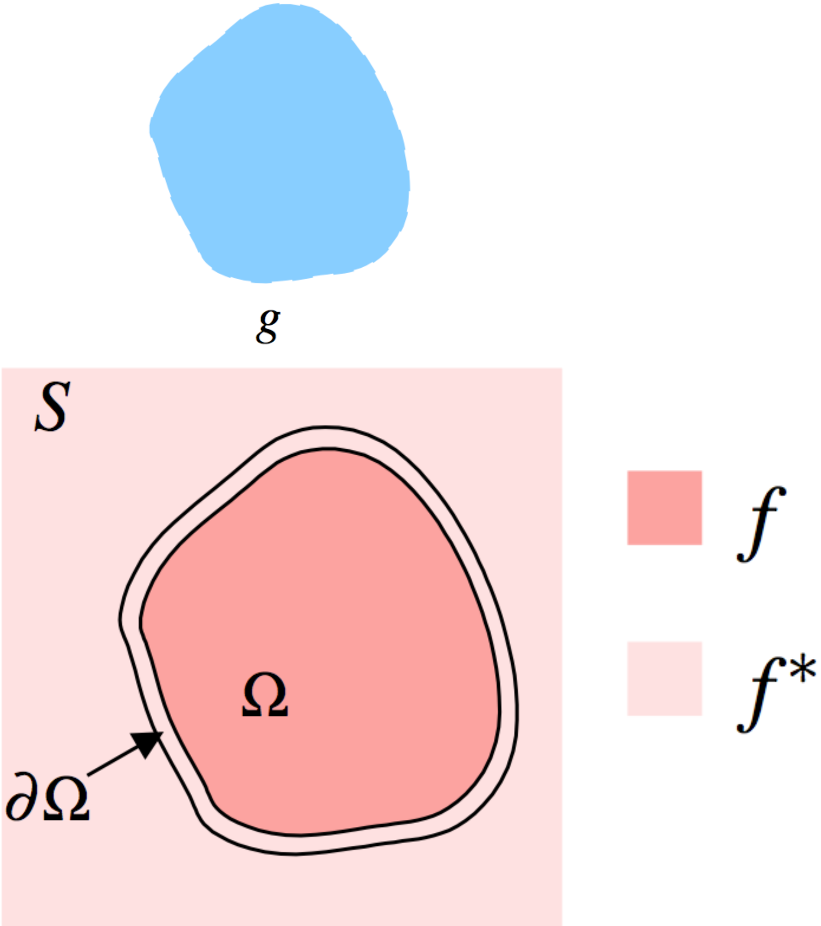
g: source function

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

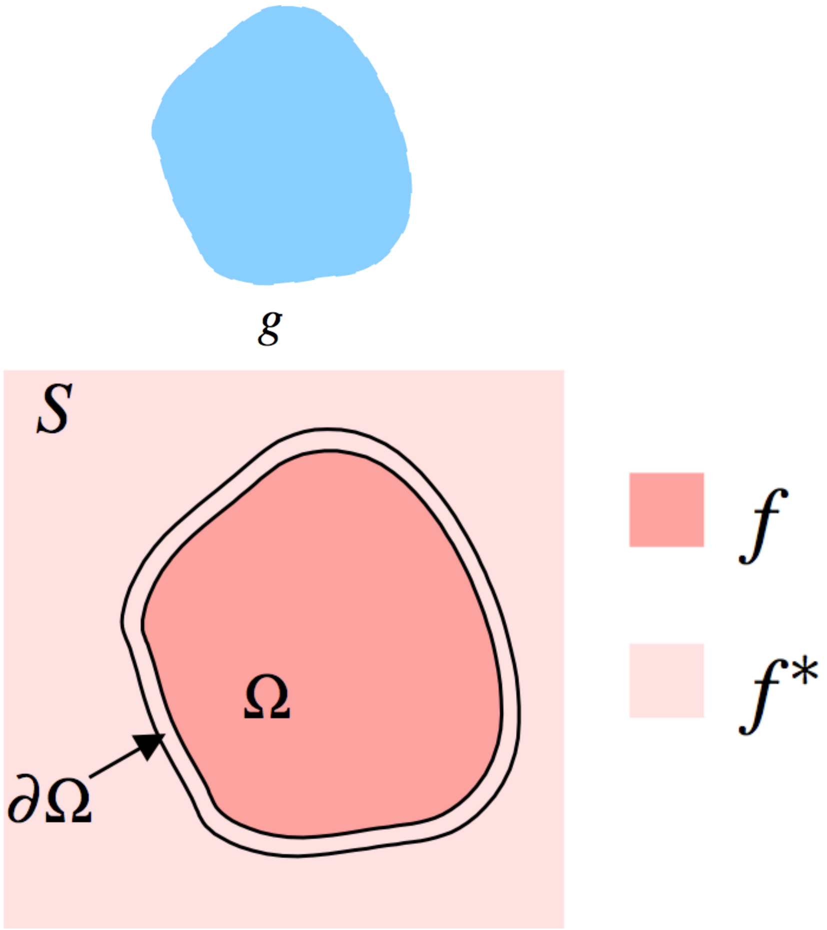
Ω: destination domain

f: 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

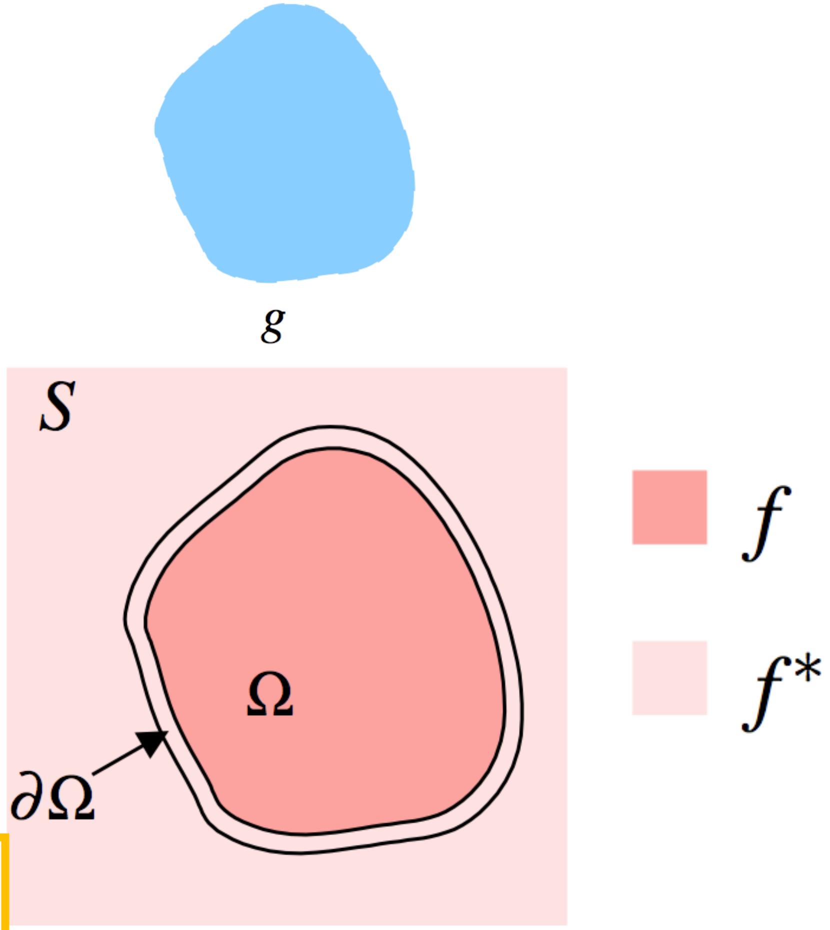
Ω: destination domain

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)

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

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

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)

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

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

Input vector field:

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

In the Poisson blending example...

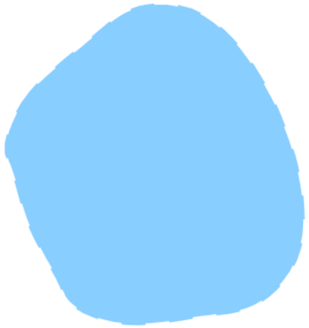
The stationary point of the variational loss is the solution to the:

Poisson equation (with Dirichlet boundary conditions)

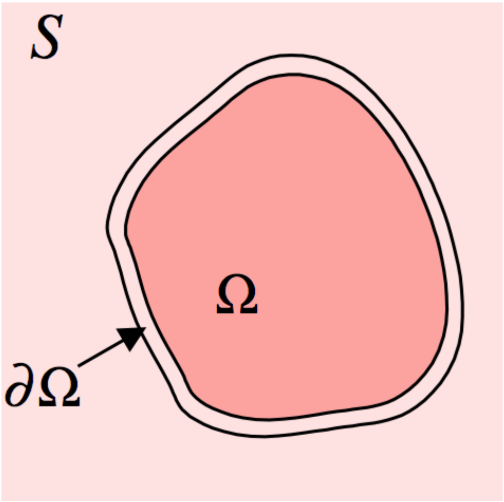
$$\Delta f = \text{div}\ \mathbf{v} \quad \text{over} \quad \Omega, \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

Find f such that:

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



g



What does the input vector field equal in Poisson blending?

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

In the Poisson blending example…

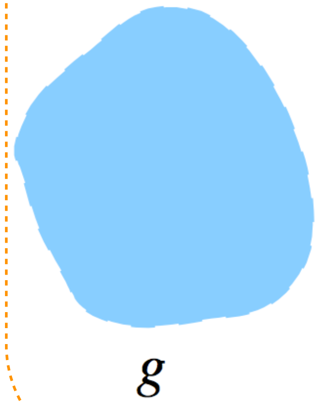
The stationary point of the variational loss is the solution to the:

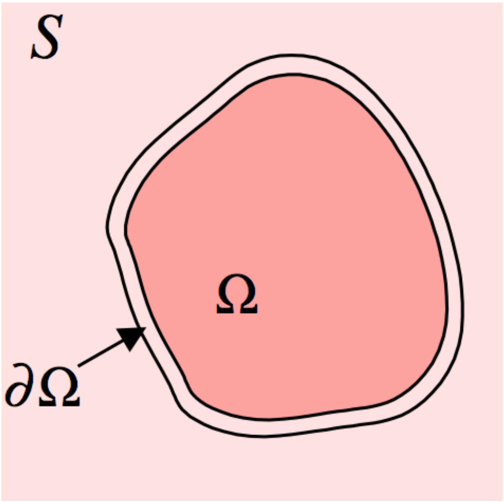
Poisson equation (with Dirichlet boundary conditions)

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

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…

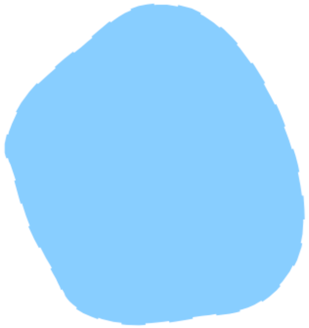
The stationary point of the variational loss is the solution to the:

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 $\partial\Omega$.

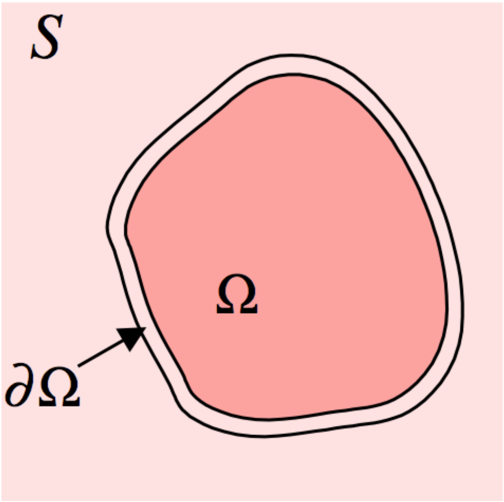


g

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)

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

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:

$$-4f(x,y) + f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) = u(x+1,y) - u(x,y) + v(x,y+1) - v(x,y)$$

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):

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

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 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? →

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

what are the sizes of these?

We can rewrite this as

linear equation of P variables

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

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:

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

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:

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

Expand the error:

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

Minimize the error:

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

Solve for x $f = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\boldsymbol{b}$ ⟵ Note: You almost never want to compute the inverse of a matrix.

In Matlab:

```
f = A \ b
```

Discrete 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:

$$Af = b$$

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 <u>very</u> 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

1	-1

partial-y derivative filter

1
-1

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:

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

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:

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

$G^T v$ → It is equal to the vector b we derived previously!

$G^T G$ → It is equal to the Laplacian matrix A we derived previously!

Reminder from variational case

Poisson equation (with Dirichlet boundary conditions)

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

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 <u>exactly</u> 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:

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

Is this cheaper than the pseudo-inverse approach?

- We never need to compute A, only its products with vectors $\mathbf{f}$, $\mathbf{r}$.

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:

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

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:

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

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

- 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 f^0 matter?

- It doesn’t matter in terms of what final 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 f (e.g., 4x4).
  - Use the solution to initialize gradient descent for a higher resolution f (e.g., 8x8).

  …

  - Use the solution to initialize gradient descent for an 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:

$$Af = b$$

Preconditioning can be incorporated in the conjugate gradient descent algorithm.

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.

Is this effective for Poisson solvers?

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

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

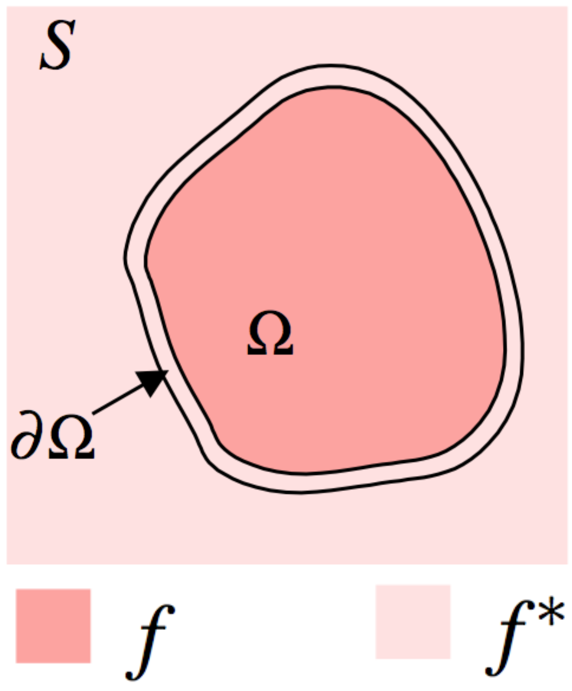
- Form a mask B that is 0 for pixels that should not be updated (pixels on S-Ω and $\partial\Omega$) and 1 otherwise.

- 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:

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

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



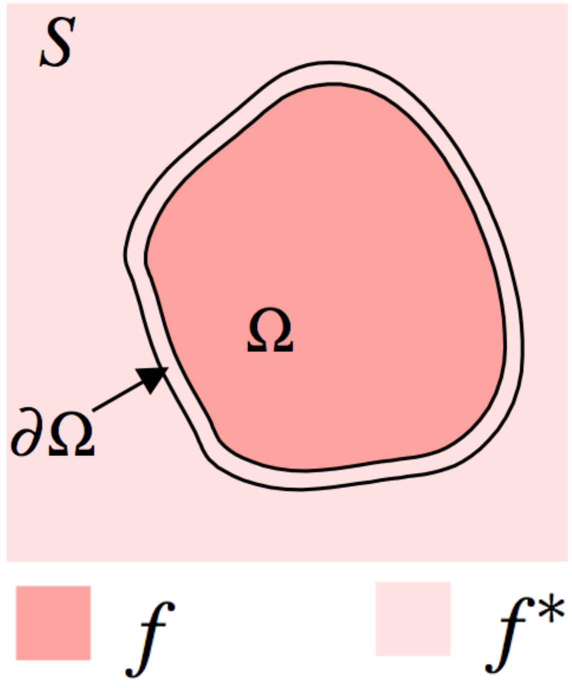
Note: handling (Dirichlet) boundary conditions

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

- 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:

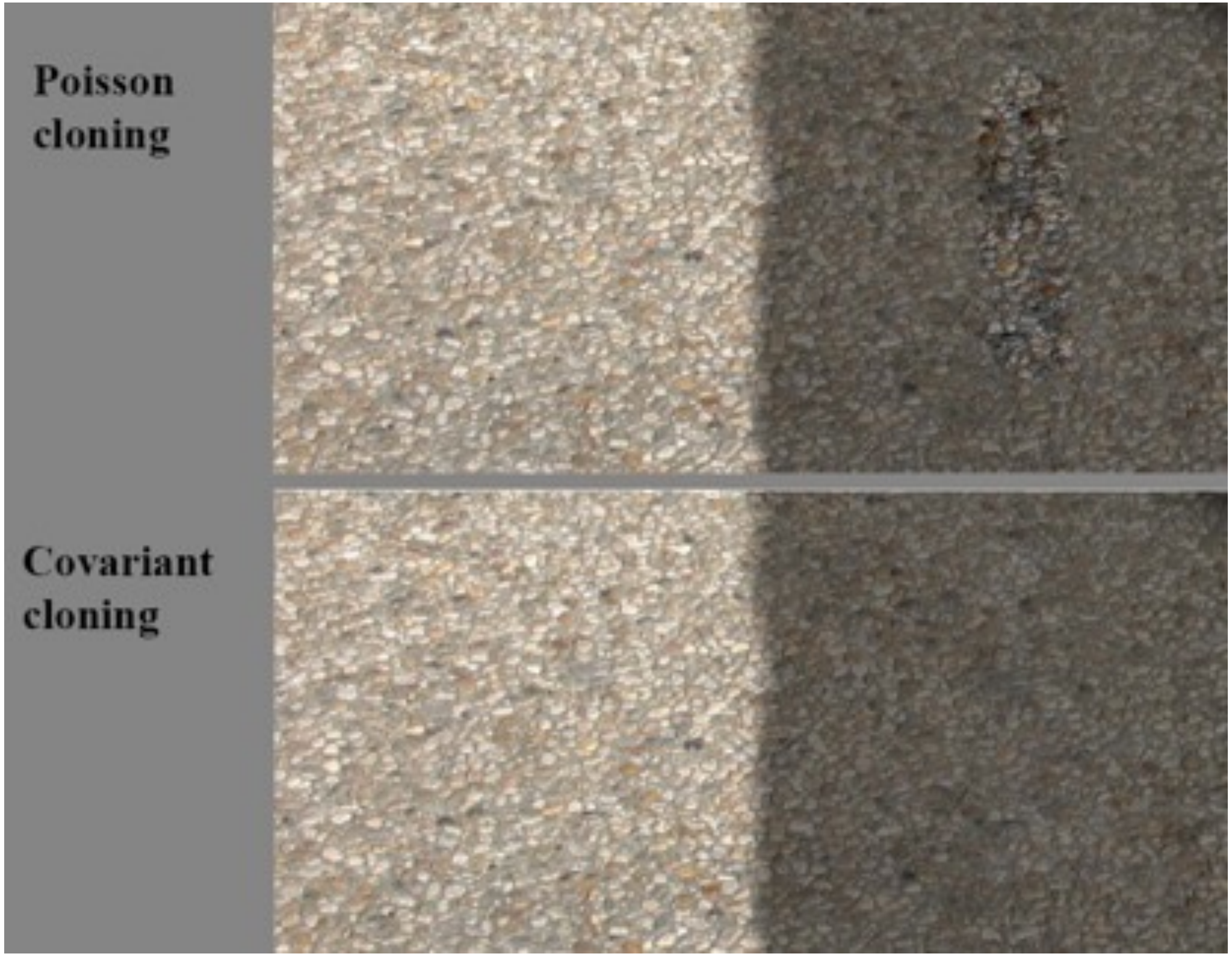
$$f^{i+1} = f^i + B\eta^i r^i \qquad \text{(gradient descent)}$$

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



Poisson image editing examples

Photoshop's "healing brush"



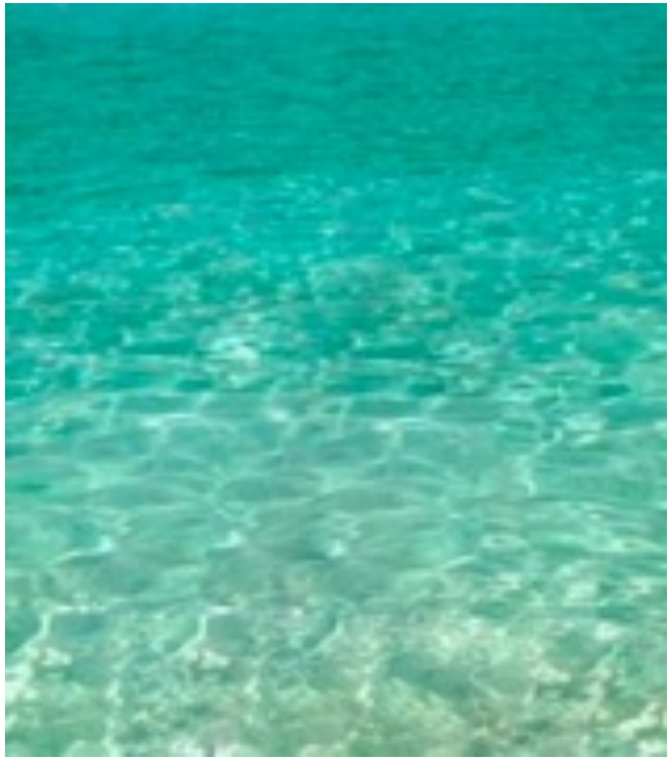
Slightly more advanced version of what we covered here:

- Uses higher-order derivatives

Contrast problem

Loss of contrast when pasting from dark to bright:

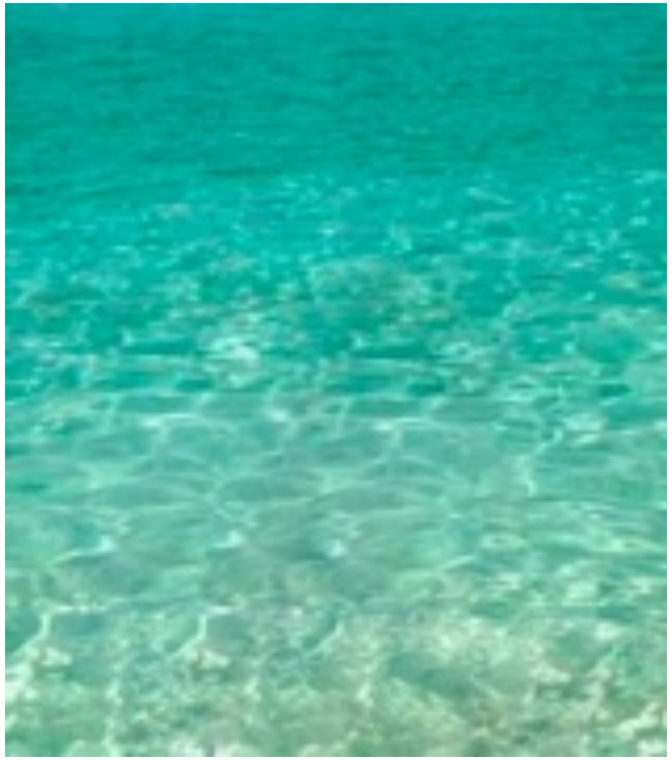
- Contrast is a multiplicative property.
- With Poisson blending we are matching linear differences.



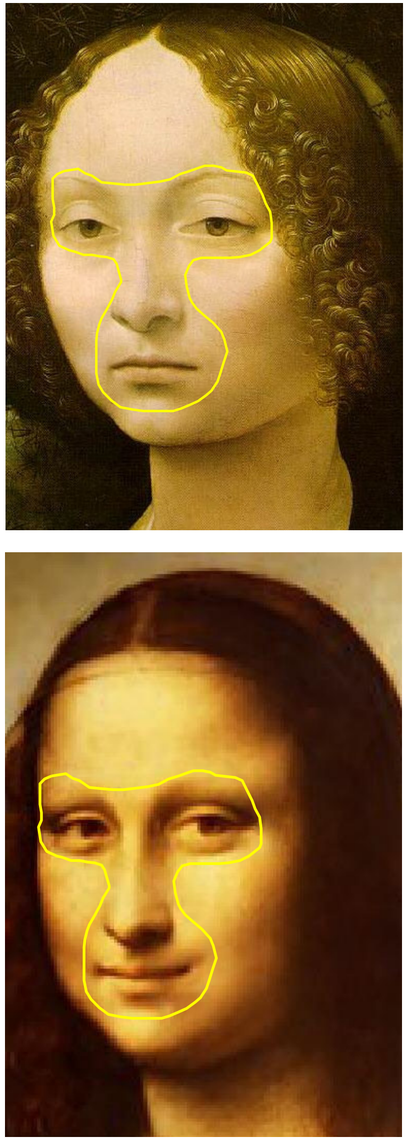
Contrast problem

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.

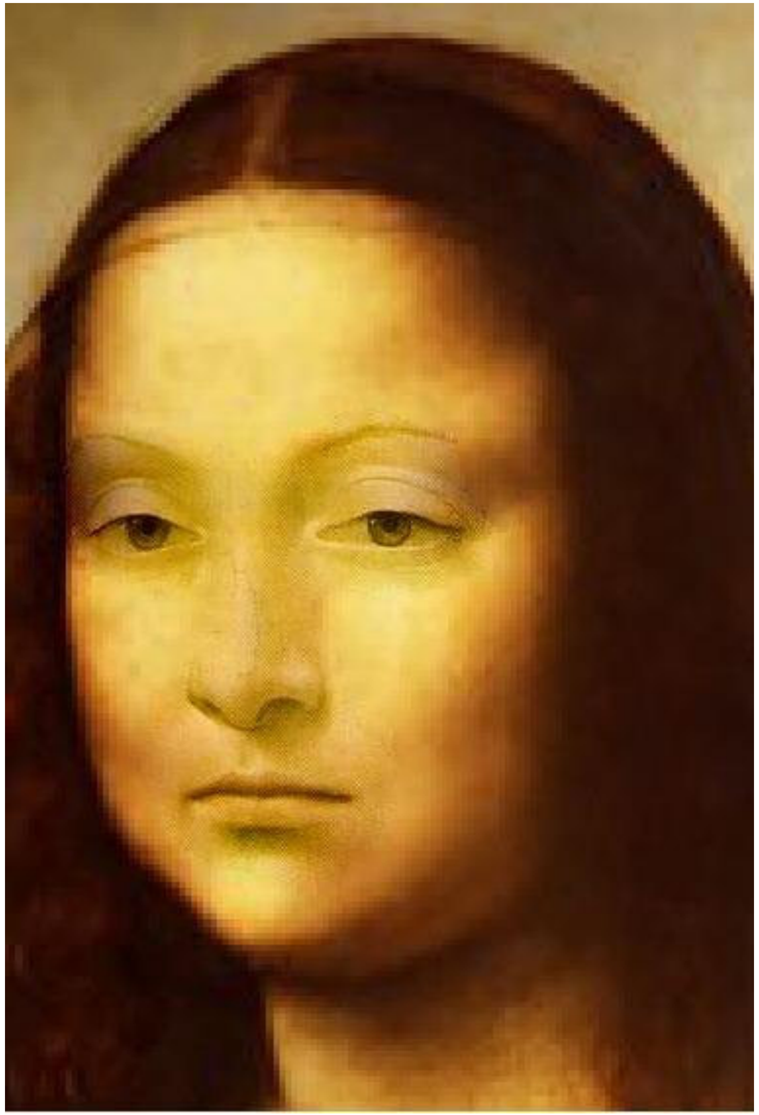


More blending



originals

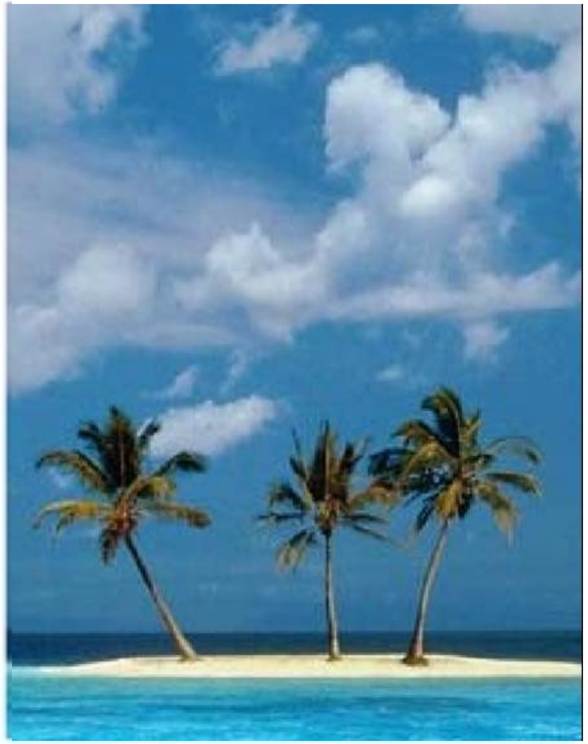
copy-paste



Poisson blending

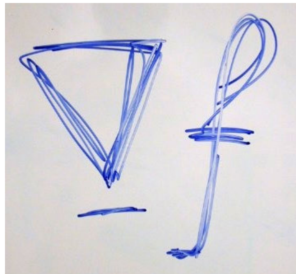
Blending transparent objects

source



destination

Blending objects with holes



color-based cutout and paste

seamless cloning

seamless cloning and destination averaged

mixed seamless cloning

Editing

Concealment

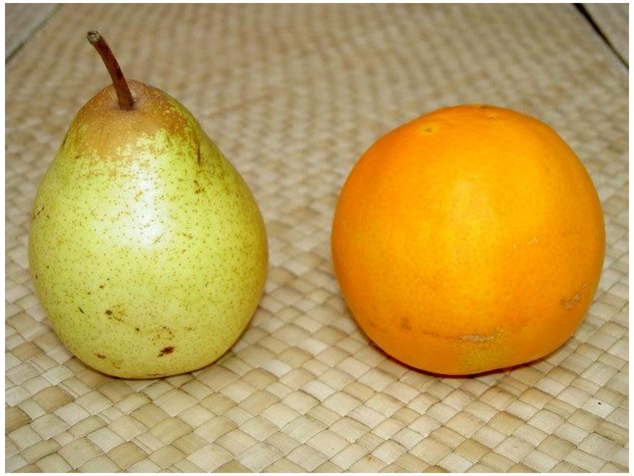
How would you do this with Poisson blending?

Concealment

How would you do this with Poisson blending?

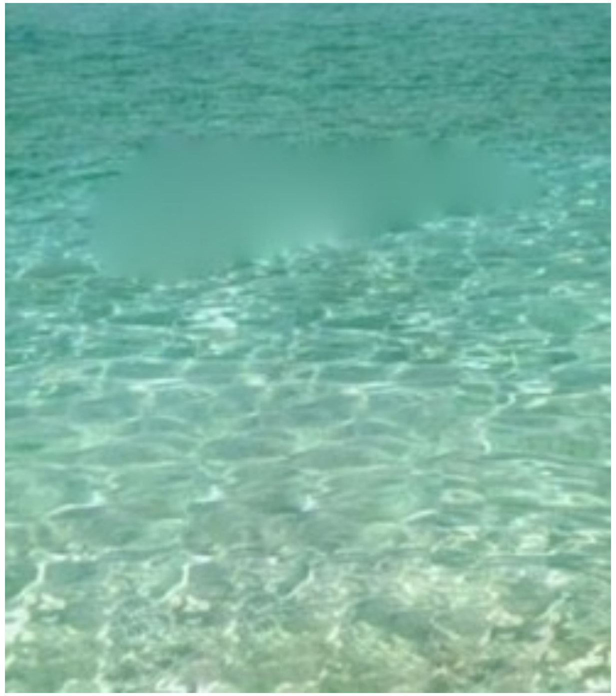
- Insert a copy of the background.

Texture swapping



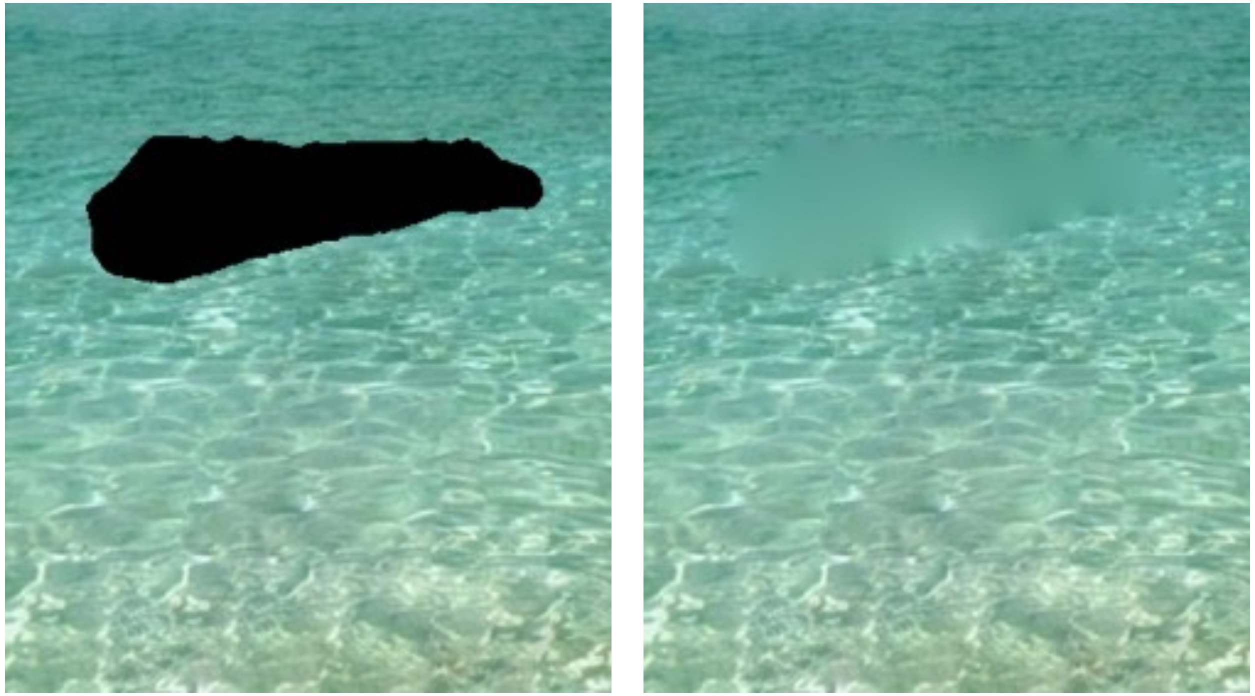
Special case: membrane interpolation

How would you do this?



Special case: membrane interpolation

How would you do this?



Poisson problem

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

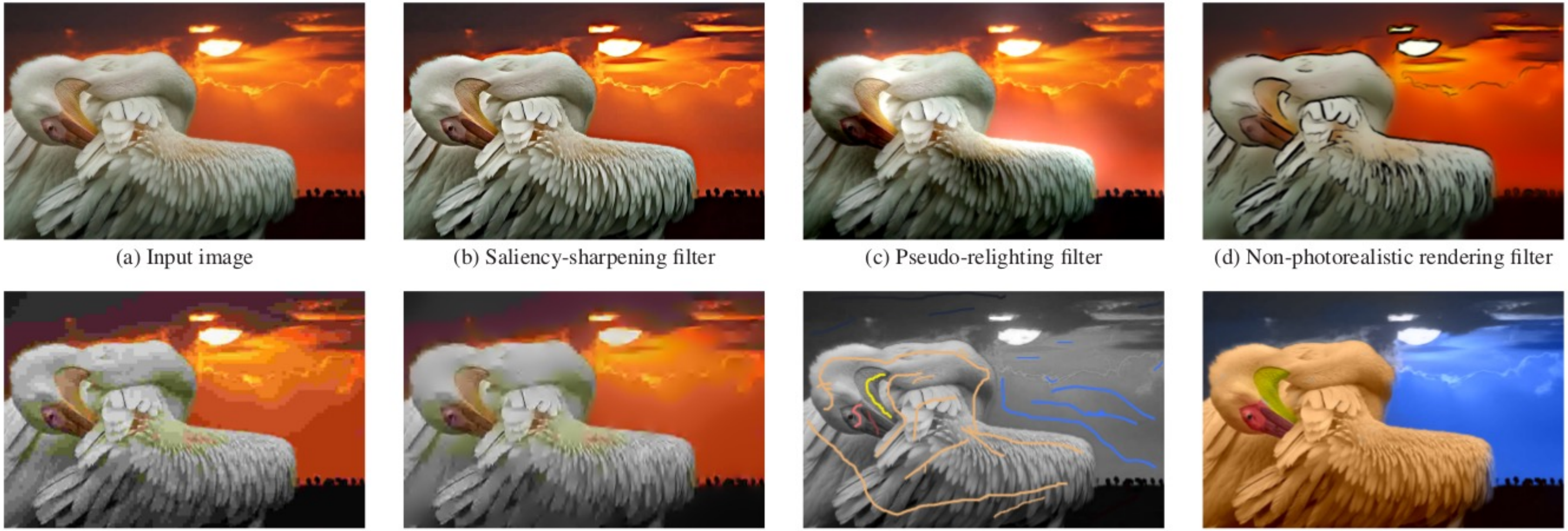
Laplacian problem

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

Entire suite of image editing tools

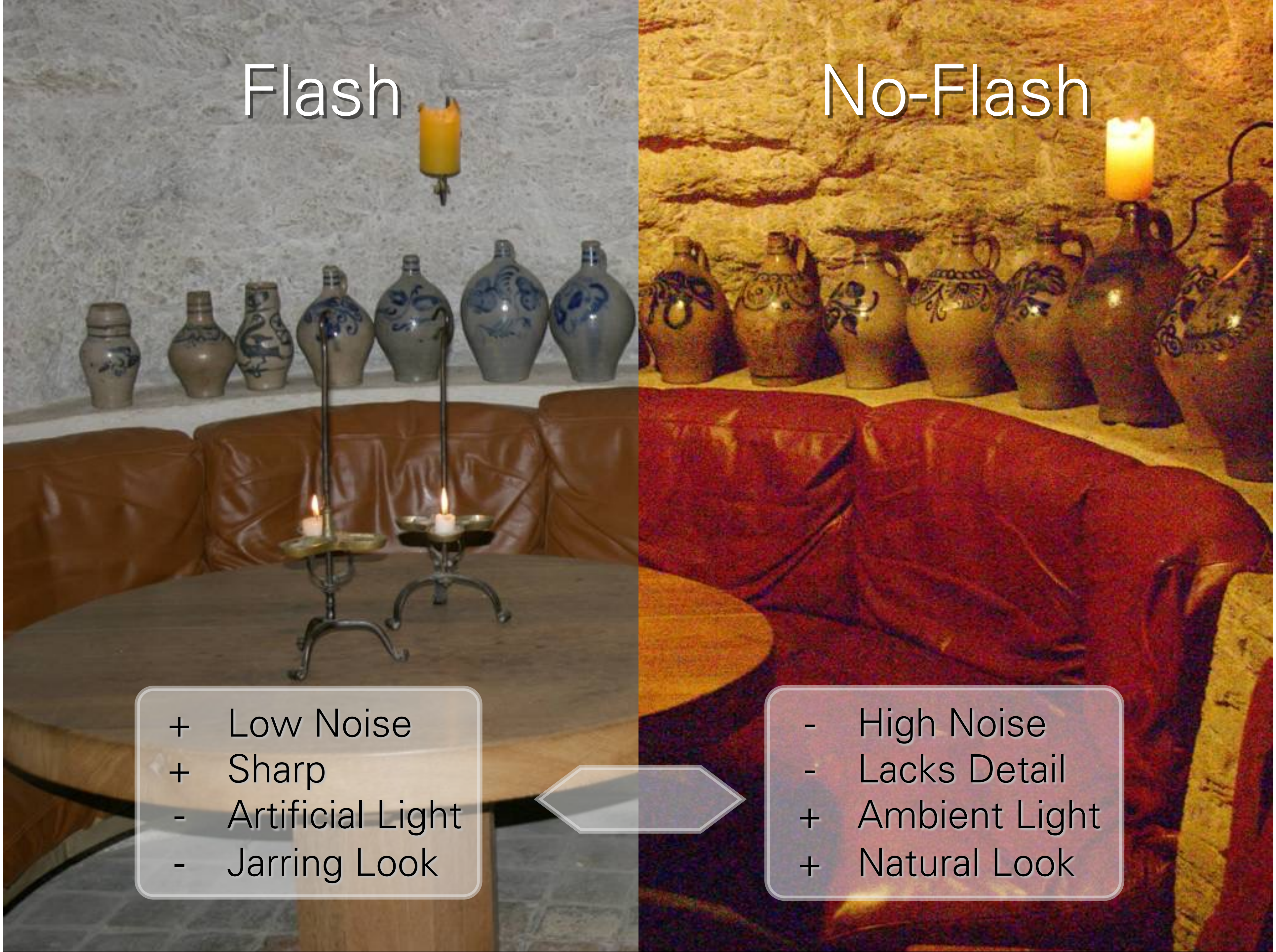
GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering

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
No-Flash
+ Low Noise
+ Sharp
- Artificial Light
- Jarring Look
- 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?

Photography Artifacts: Flash Hotspot

Ambient

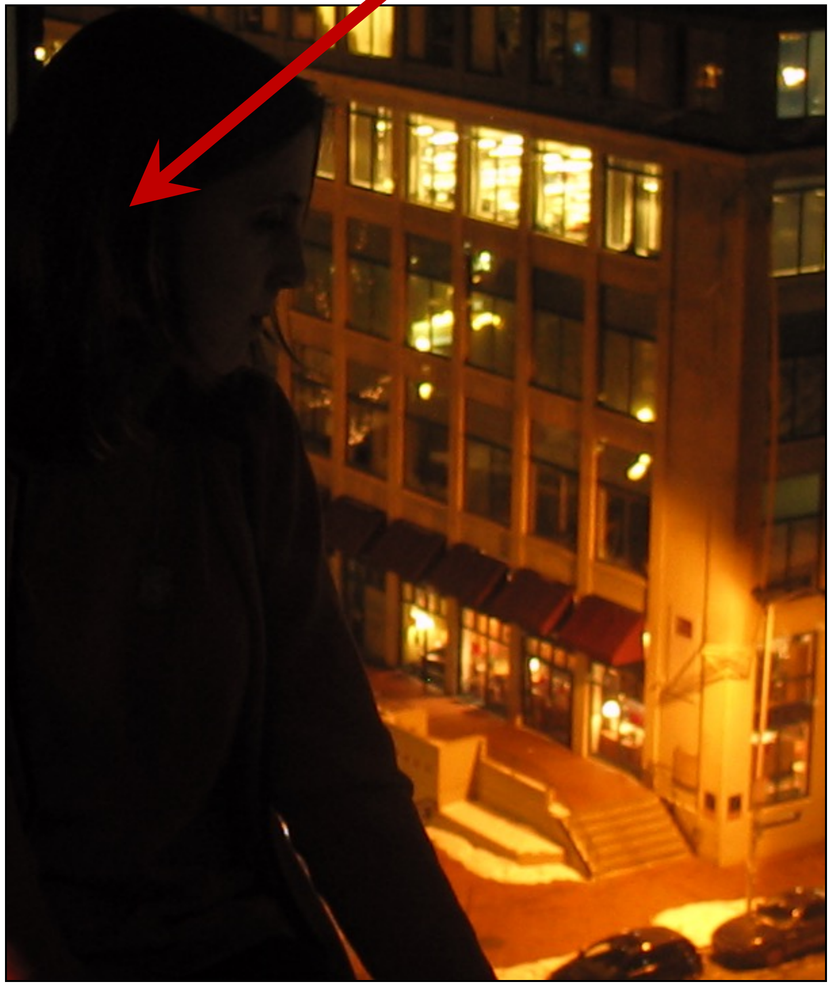
Flash

Flash Hotspot

Reflections due to Flash

Underexposed

Reflections



Ambient

Flash

Distance Dependance

Flash

Distant people underexposed

Removing self-reflections and hot-spots


Ambient

Flash

Removing self-reflections and hot-spots

Removing self-reflections and hot-spots

Ambient

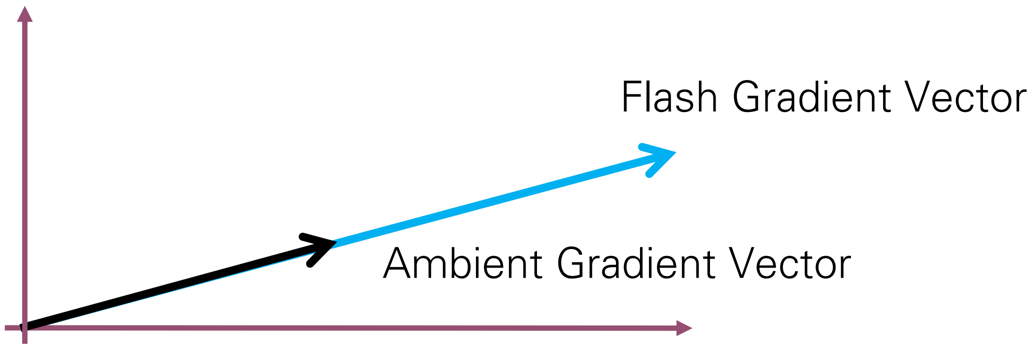
Flash

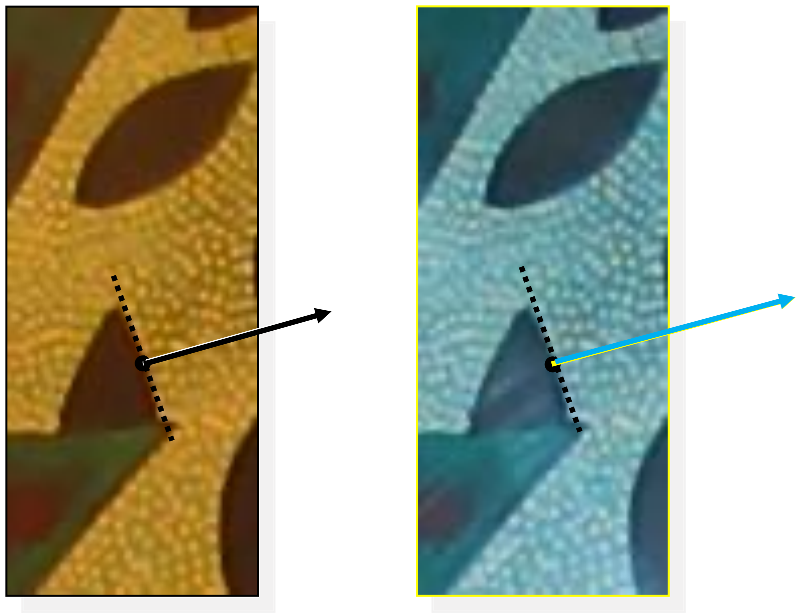
Result

Reflection Layer

Idea: look at how gradients are affected

Same gradient vector direction

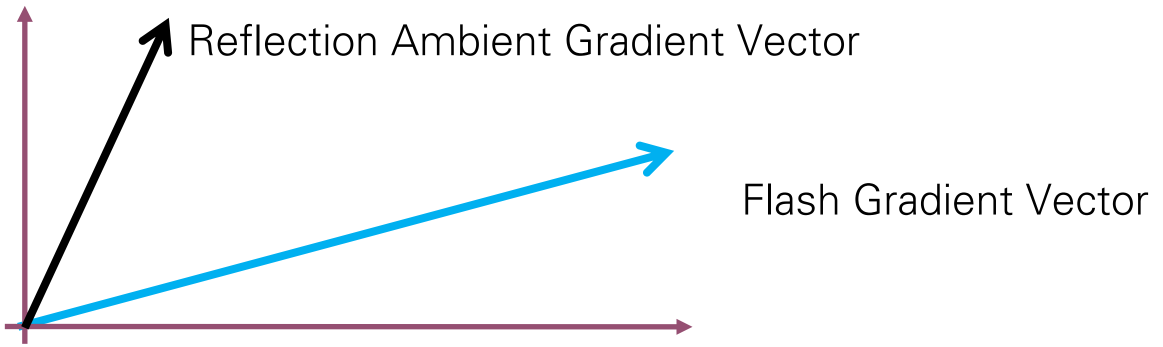


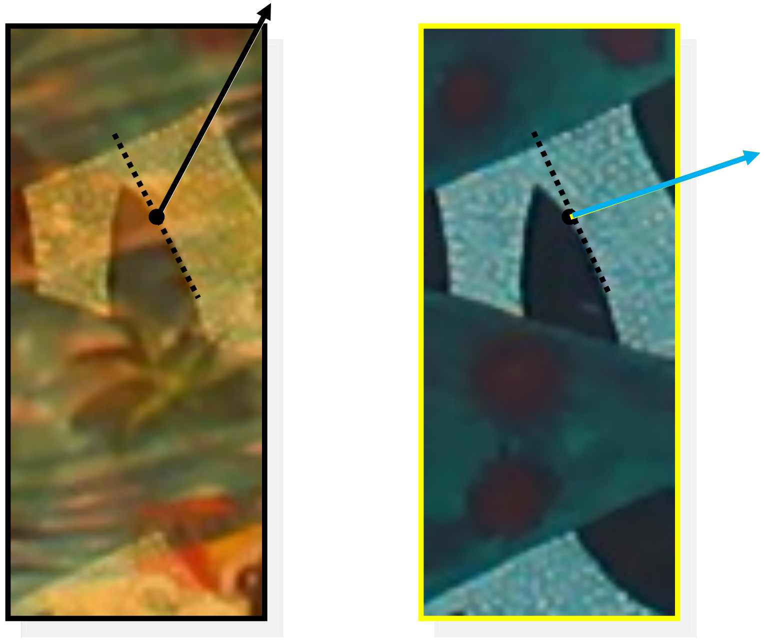


No reflections

Idea: look at how gradients are affected

Different gradient vector direction

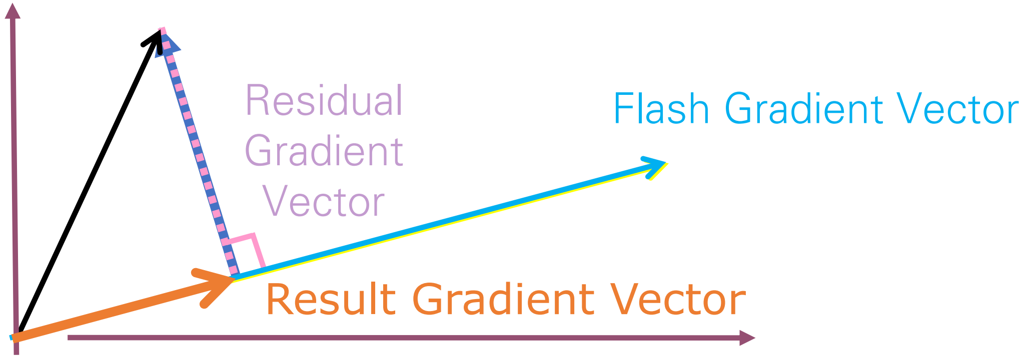




With reflections

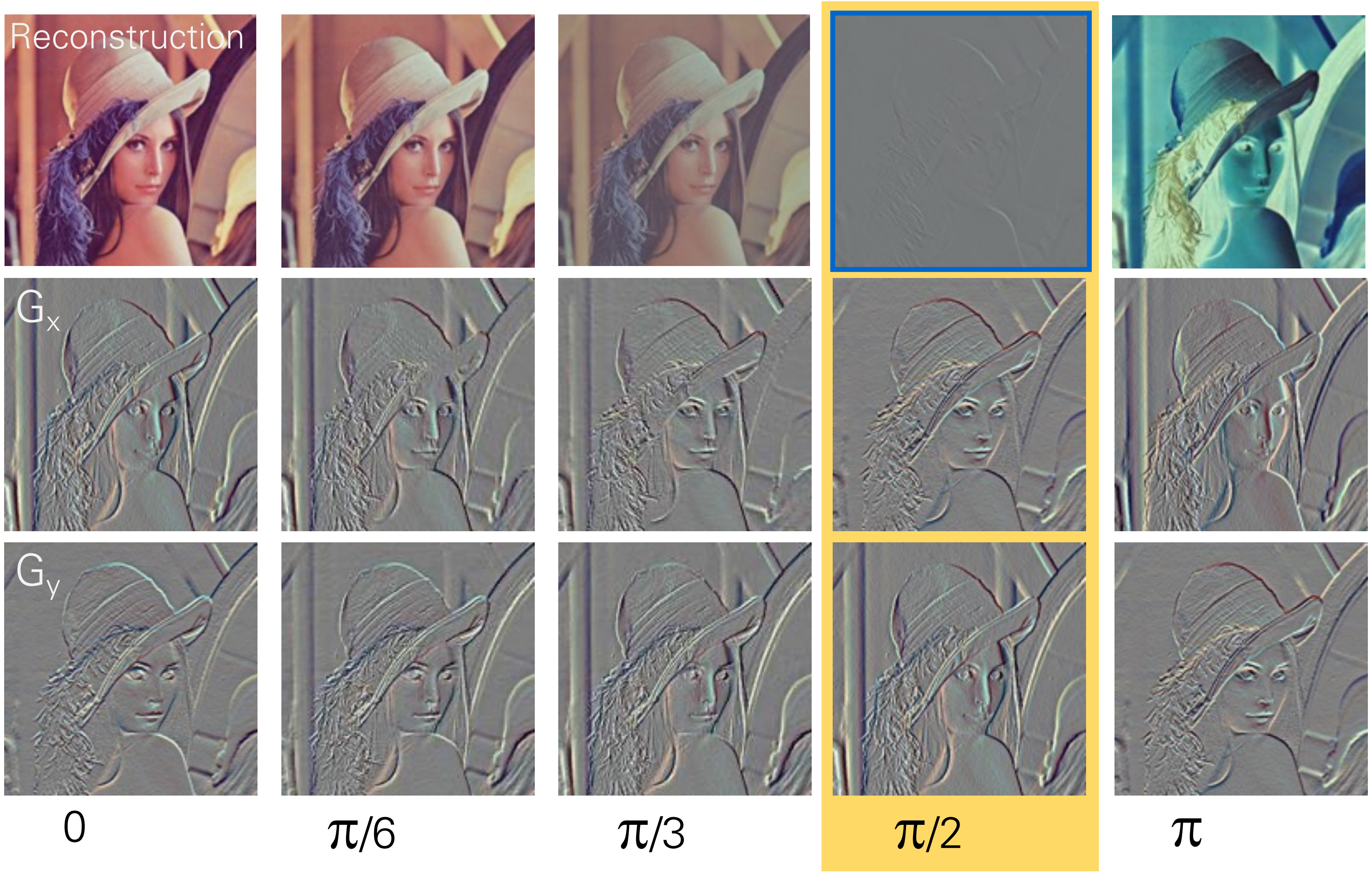
Gradient projections

- Image gradients in flash and ambient images should be aligned.
- Ambient gradient direction is refined by projecting onto the flash gradient.
- "Result" image is formed by 2D integration of the refined gradient.
- Residual gradients after projection create the "reflection layer".
- Gradient projection splits an image into reflection-free and reflection layers.



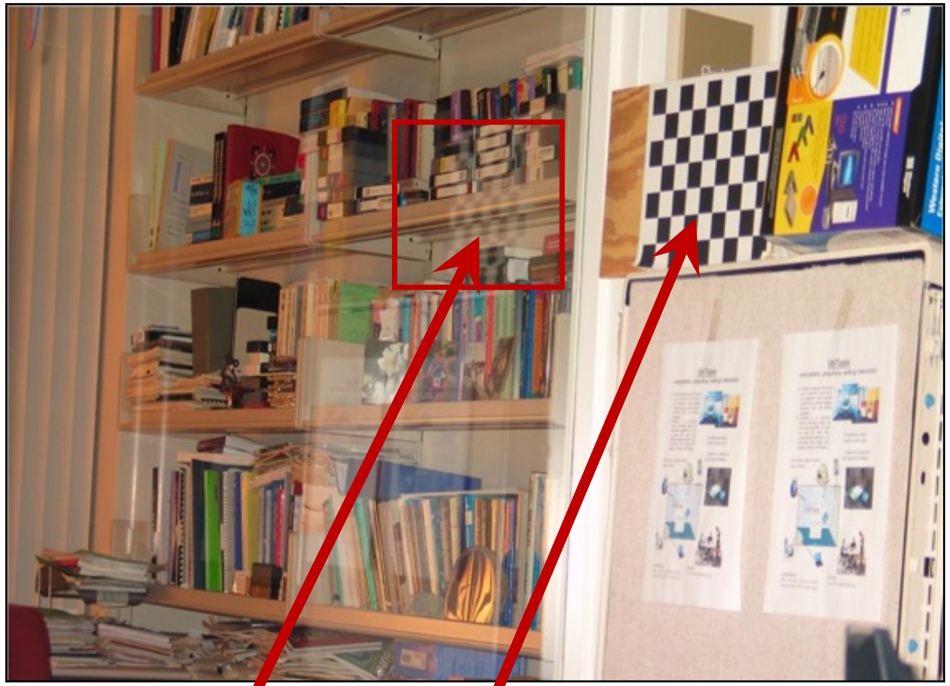
Why projections?

- Projection ensures the gradient direction is preserved, even with a new magnitude.
- Orthogonal gradients holds minimal visual information.
- Rotating gradients by 90° yields zero divergence.
- 90° rotation results in no image detail.
- 180° rotation creates a negative image.

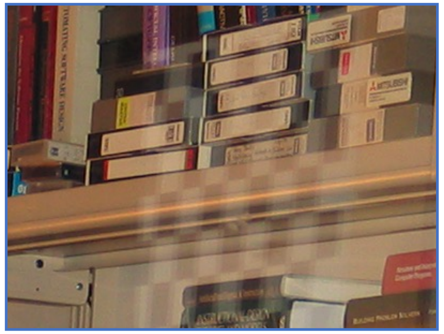


Flash/no-flash with gradient-domain processing

Flash



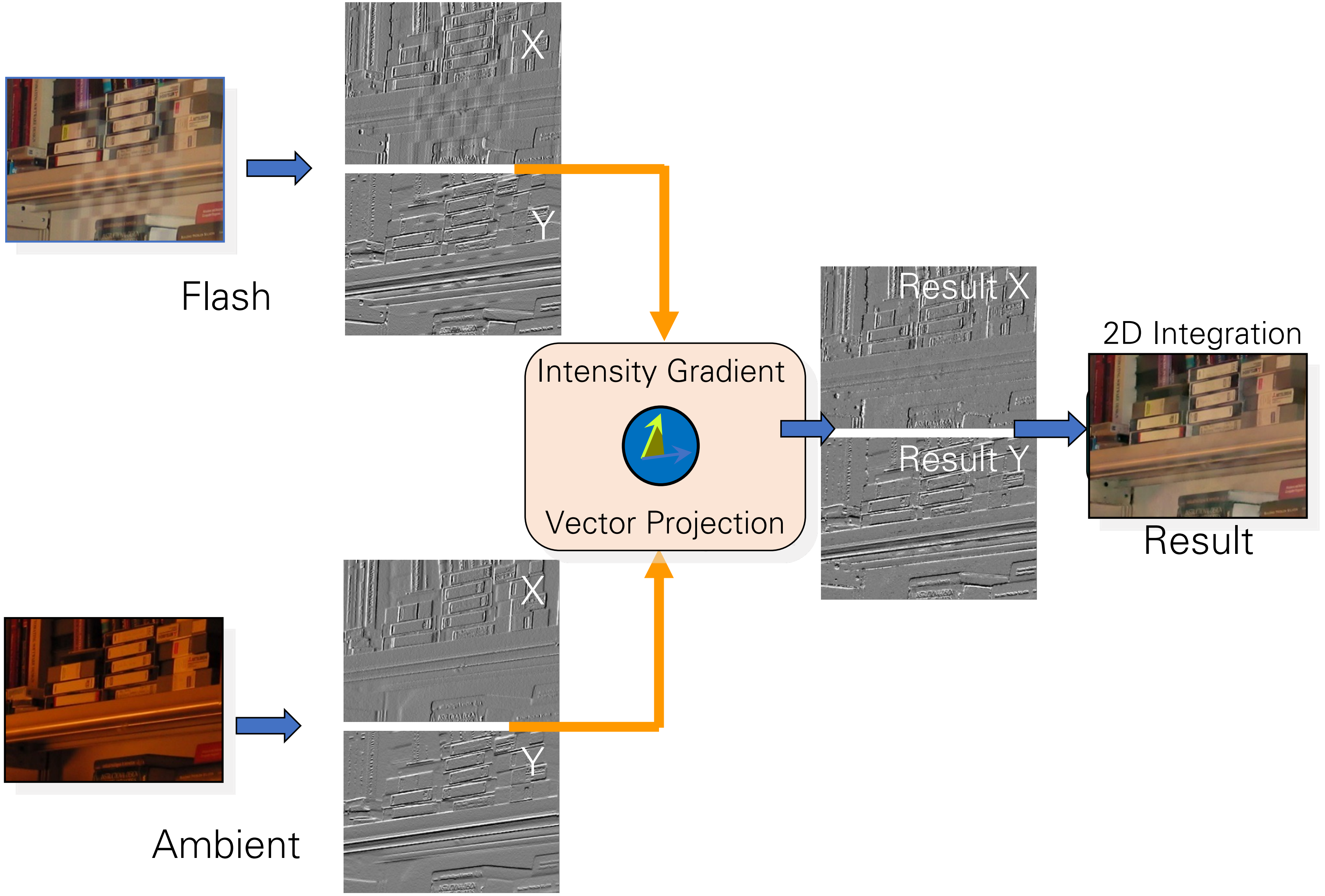
Ambient



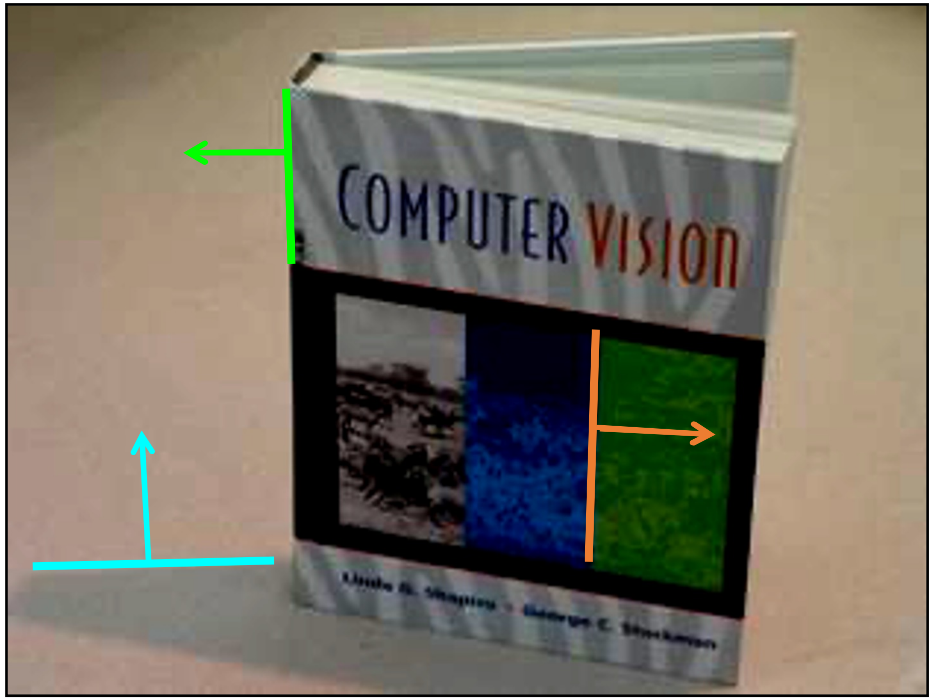
Checkerboard outside glass window

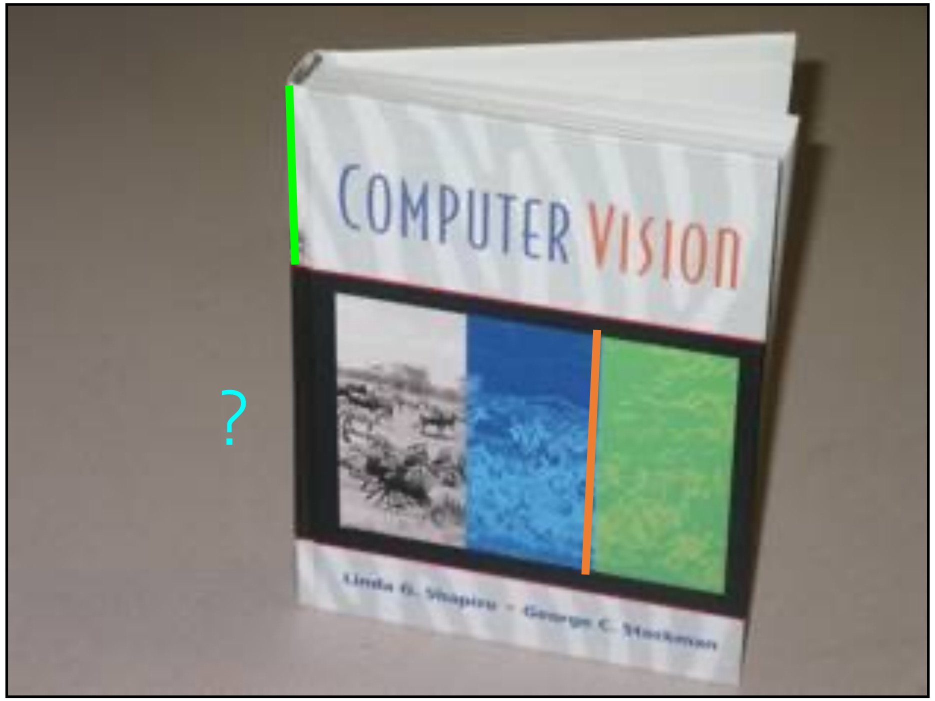
Reflections on glass window

Flash/no-flash with gradient-domain processing



Invariance of Gradient Vectors Orientation (Gradient Orientation Coherency)



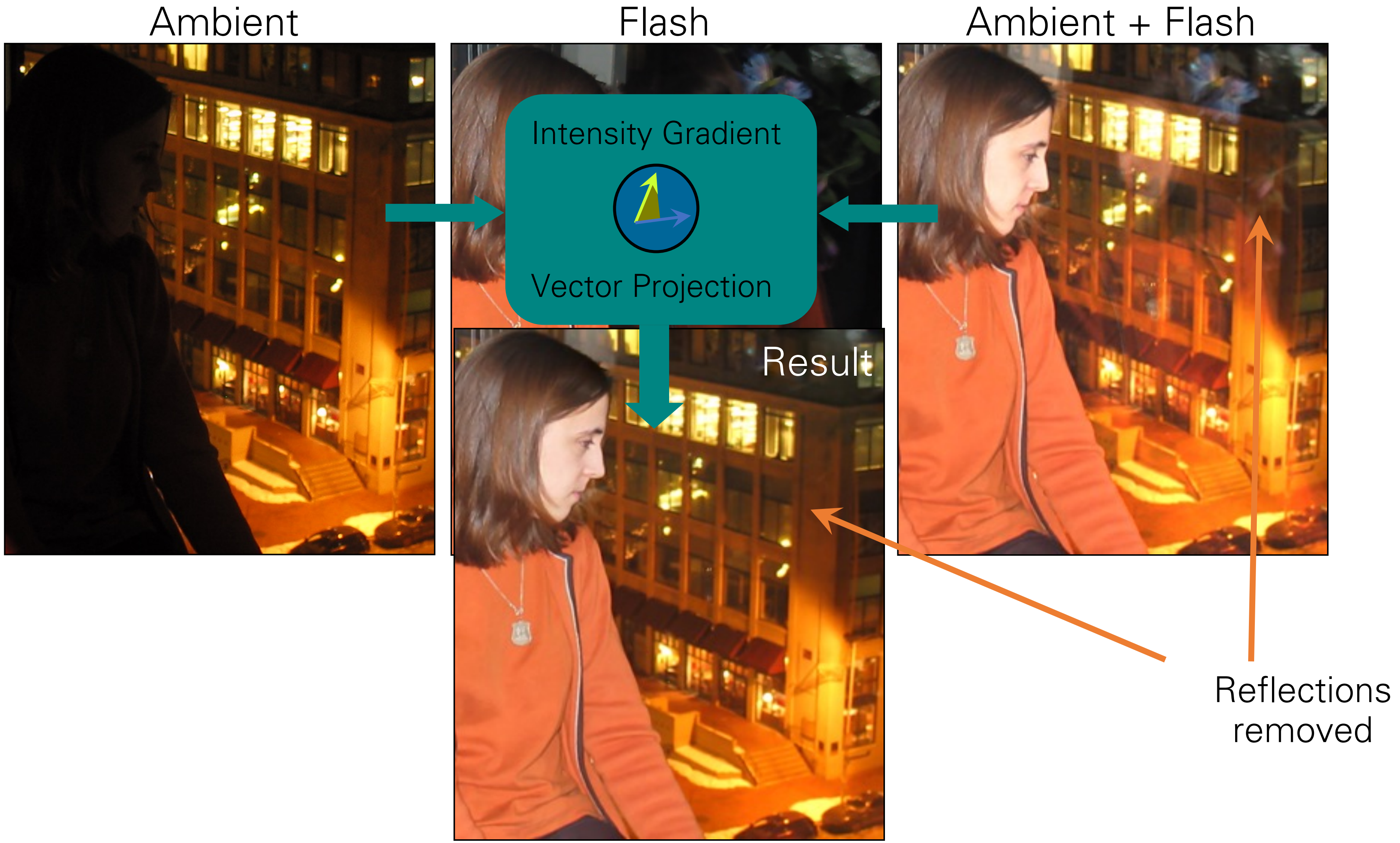


✓ Reflectance Edge

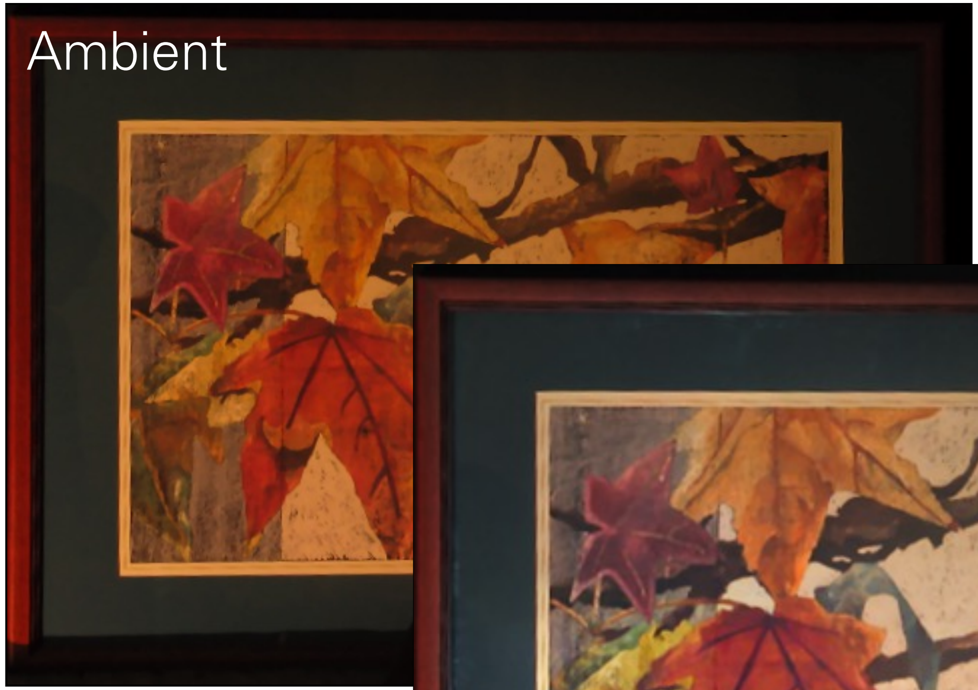
• ↑↓ Geometric Edge

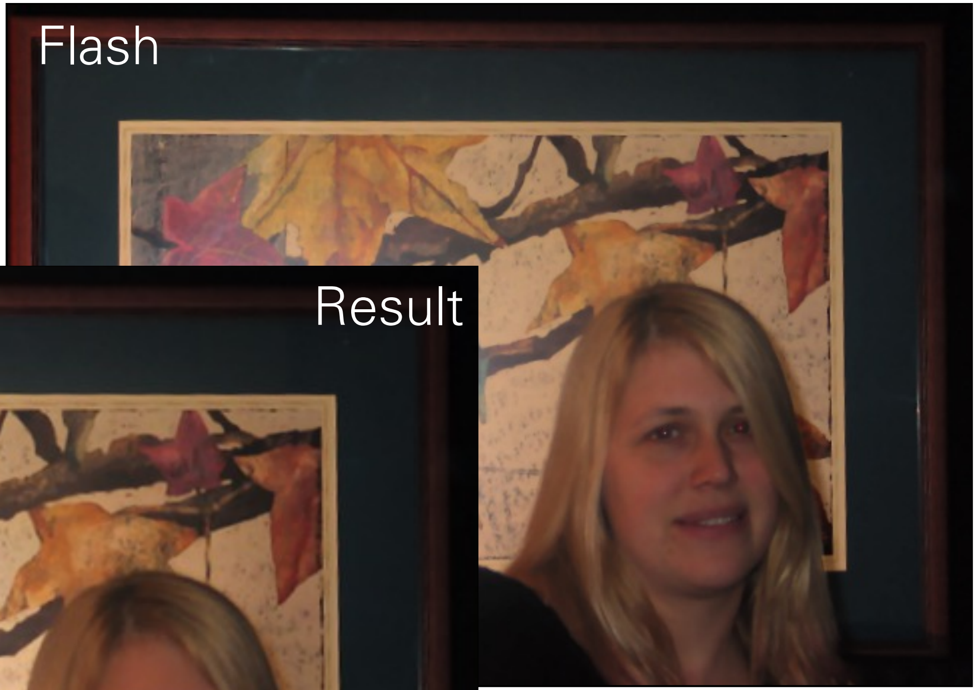
✕ Illumination Edge

Removing Reflections due to Flash



Removing Flash Hotspot

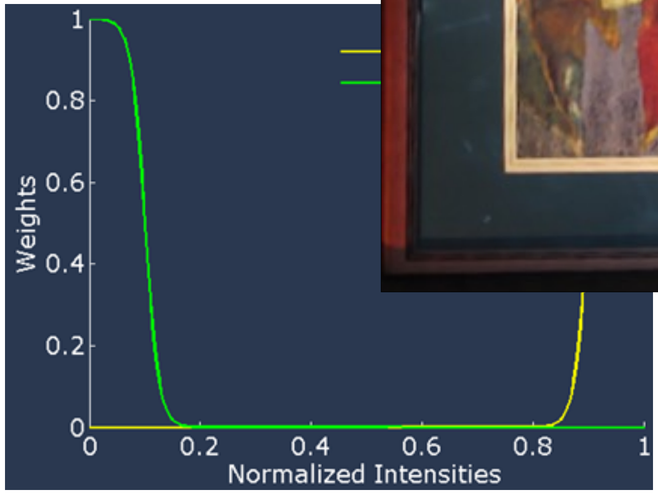




ights W_s

and ambient image

d Gradient Coherency



Depth Compensation

Ambient

Flash

Result

Distant Persons

Scale flash gradients using the ratio of flash and ambient images

$$\frac{Flash}{Ambient} = \frac{\rho \cos\theta}{(Ambient^{*}) \times distance^2} \propto \frac{1}{distance^2}$$

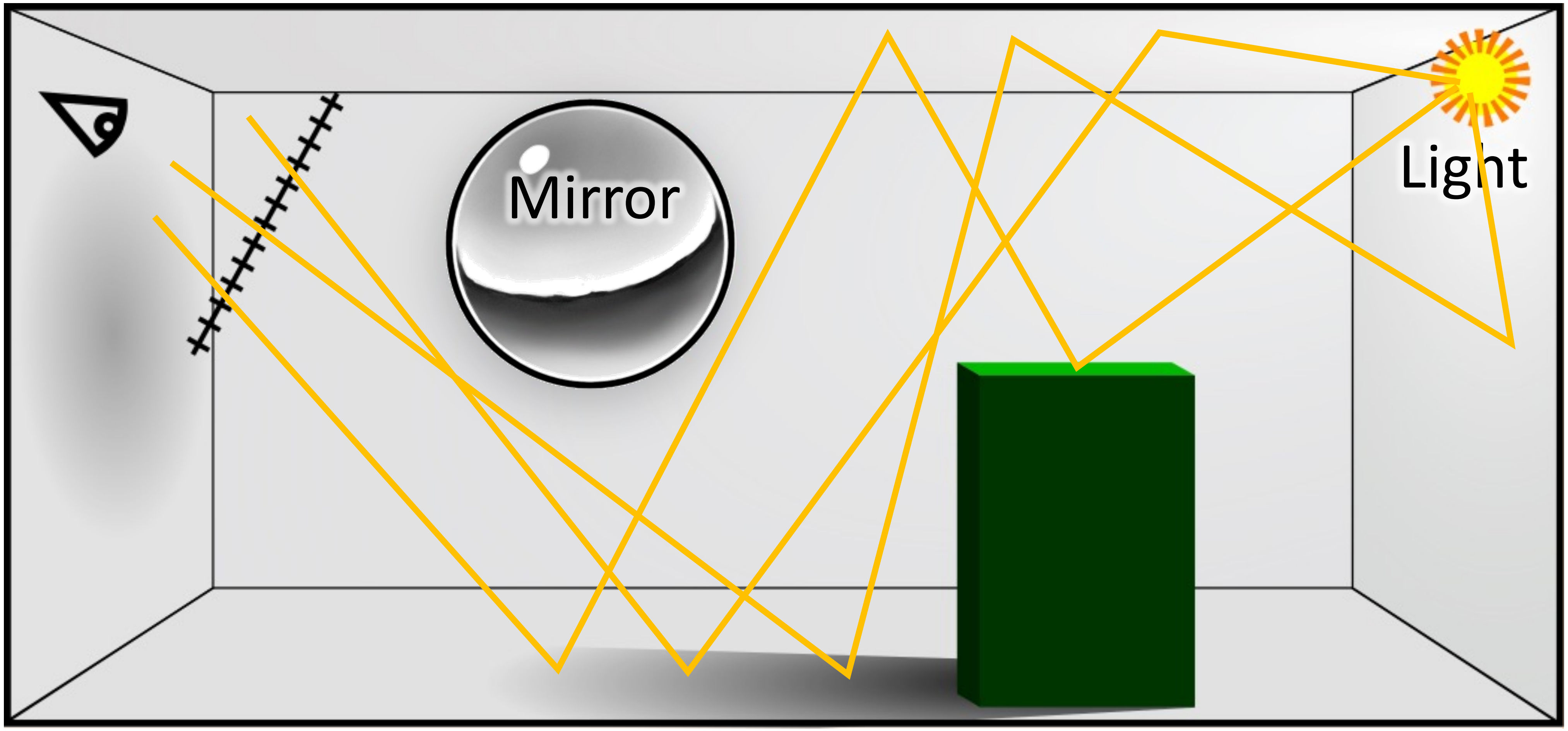
Limitations

- Difficult Scenarios
  - Dynamic scenes
  - Co-located artifacts
  - Strong ambient illumination edges

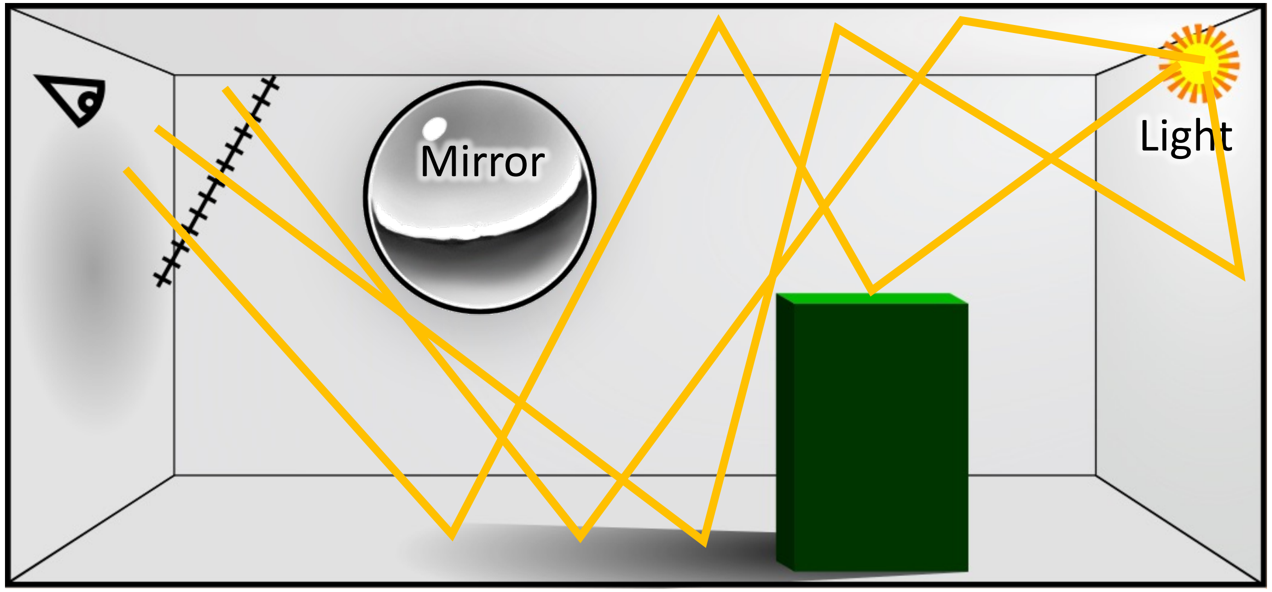
- Issues
  - Lack of reliable gradients
    - Homogeneous or dark regions
  - Color coherency

Gradient-domain rendering

Rendering Equation



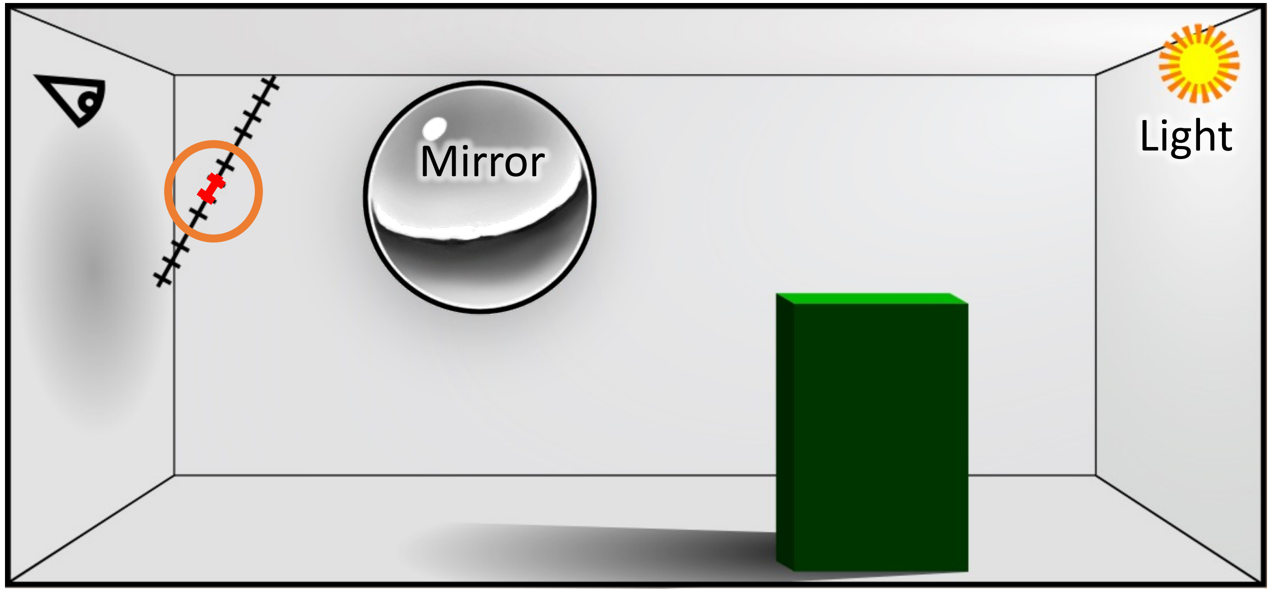
Rendering Equation



$$I_j = \int_{\Omega} f_j(\bar{x})\, \mathrm{d}\mu\,(\bar{x})$$

[Veach 98]

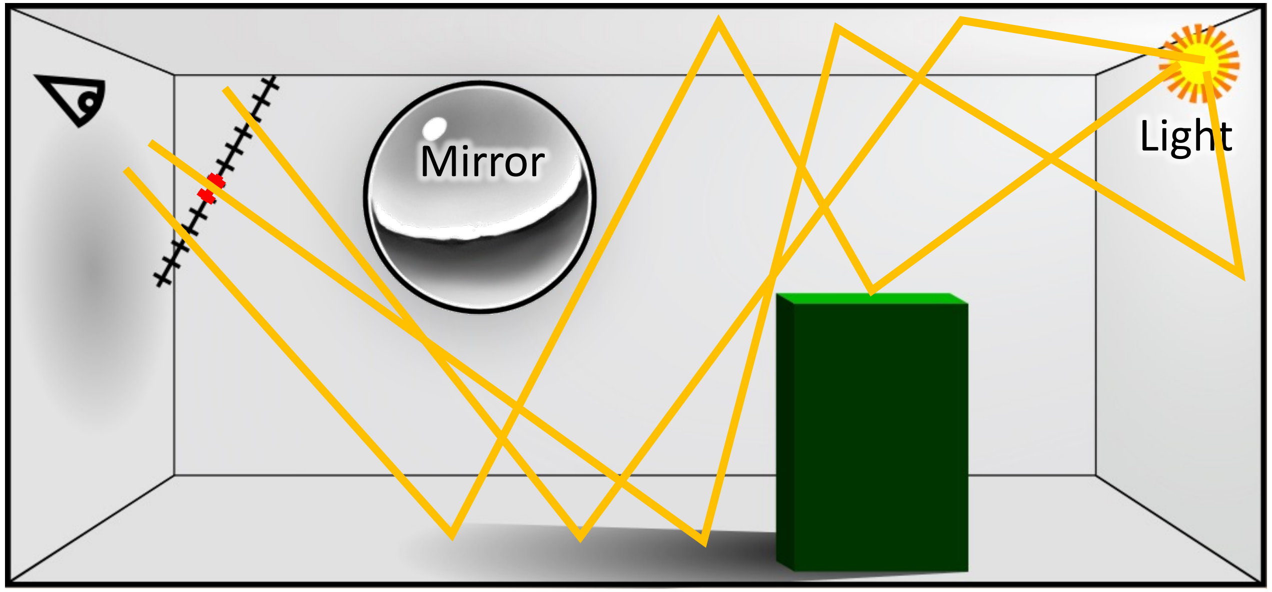
Rendering Equation



$$I_j = \int_{\Omega} f_j(\bar{x})\, \mathrm{d}\mu(\bar{x})$$

[Veach 98]

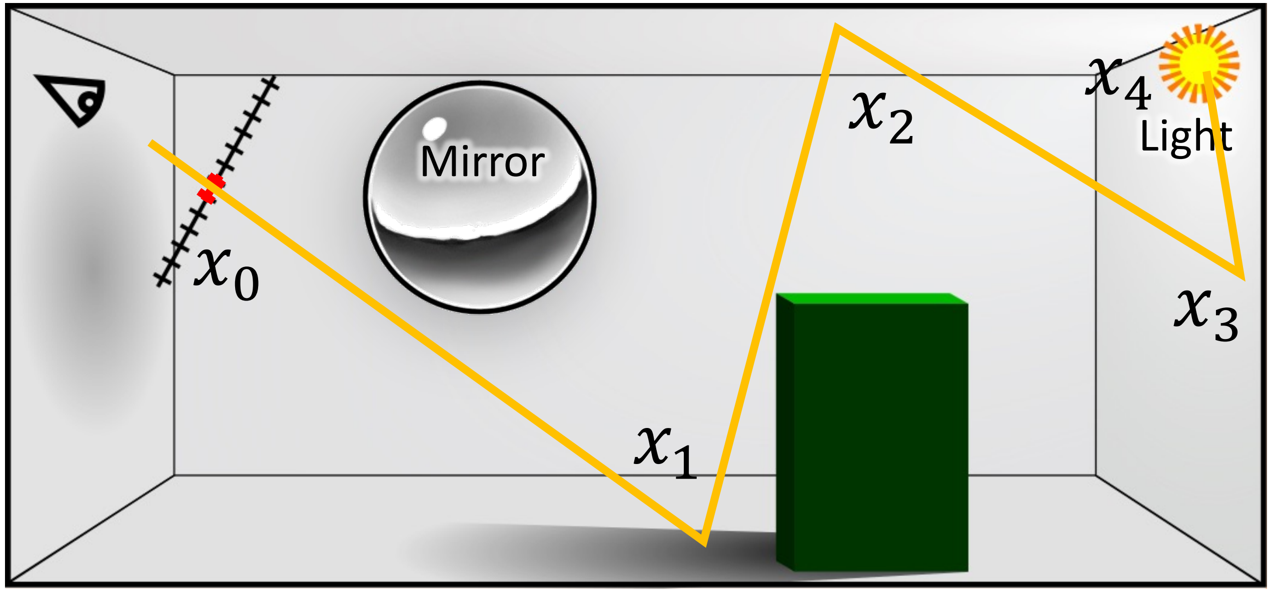
Rendering Equation



$$I_j = \int_{\Omega} f_j(\bar{x})\, \mathrm{d}\mu(\bar{x})$$

[Veach 98]

Rendering Equation



$$\bar{x} = x_0 x_1 x_2 x_3 x_4$$

$$I_j = \int_{\Omega} f_j(\bar{x}) \, \mathrm{d}\mu(\bar{x})$$

$f_j(\bar{x})$ = (Materials) x (Geometries)

x Emitted Lum. x Pixel filtering

[Veach 98]

Rendering Equation

$$I_j = \int_{\Omega} f_j(\bar{x}) \, \mathrm{d}\mu \, (\bar{x})$$

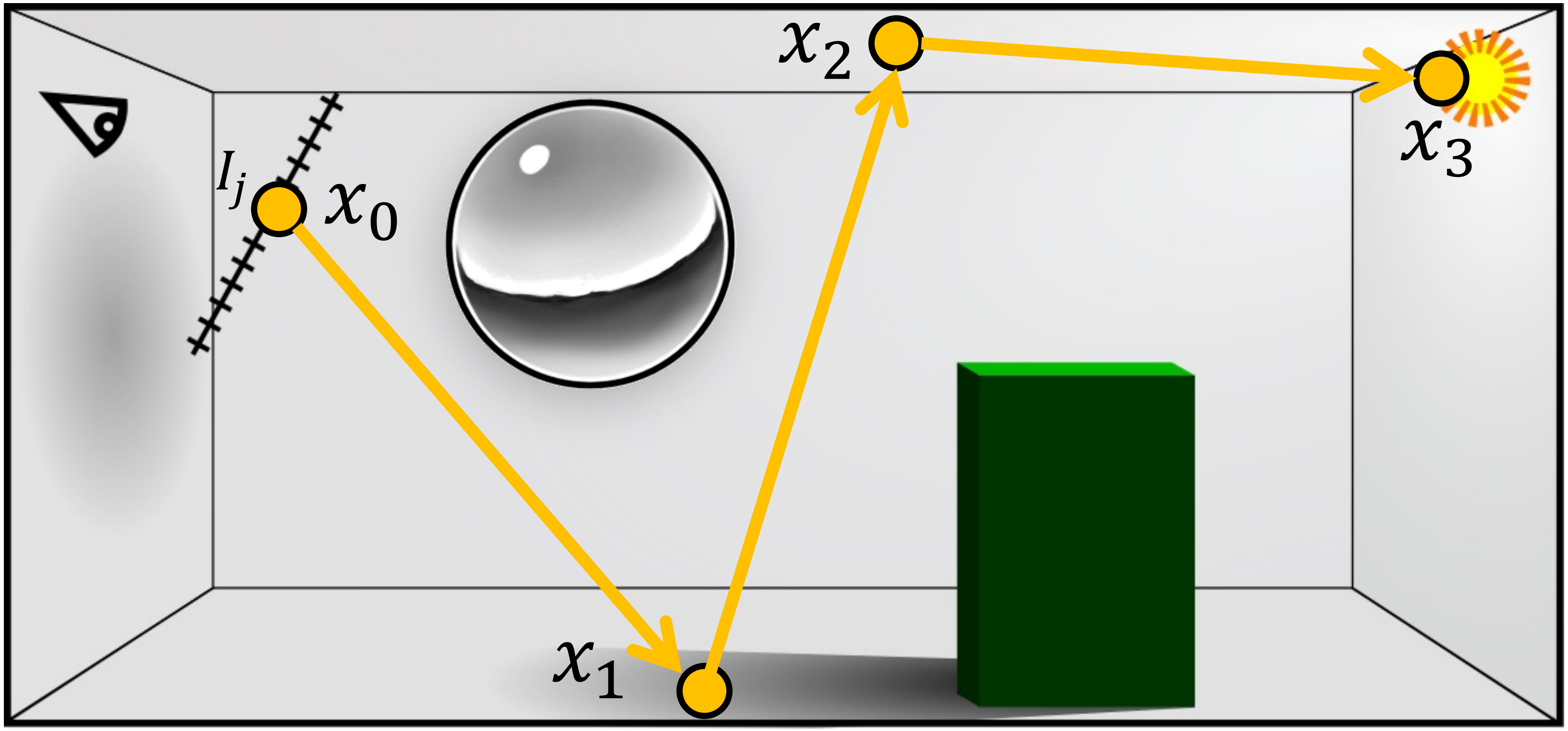
→

Monte Carlo estimator

$$I_j \approx \frac{1}{N} \sum_{k=1}^{N} \frac{f_j(\bar{x}_k)}{p(\overline{x_k})}$$

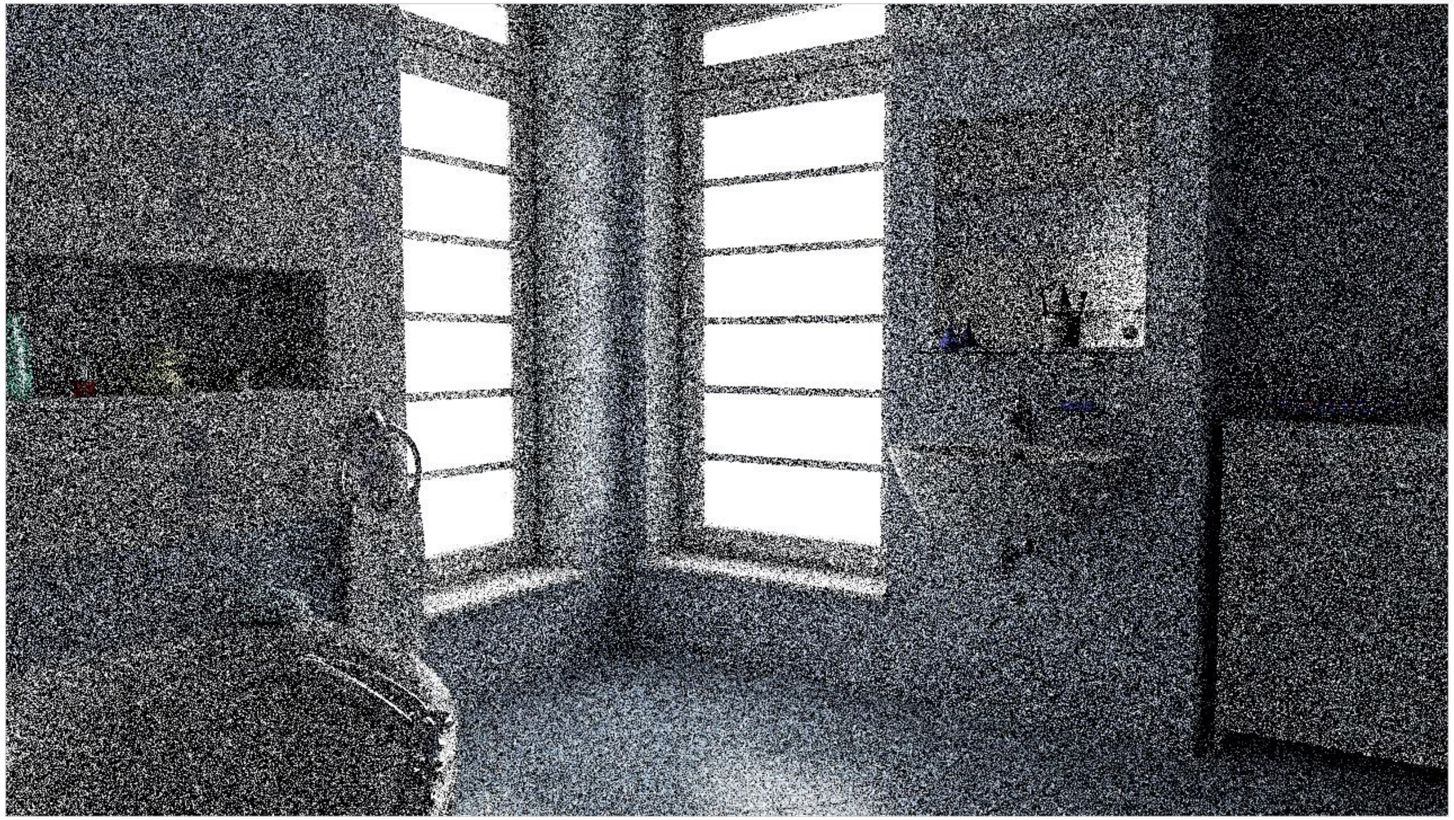
$p(\overline{x_k})$ is the probability density to sample $\overline{x_k}$

Path Tracing



Motivation

error / 2 = samples * 4



15min 30min 45min 1h

Motivation

Observation

- Noise mostly proportional to signal magnitude

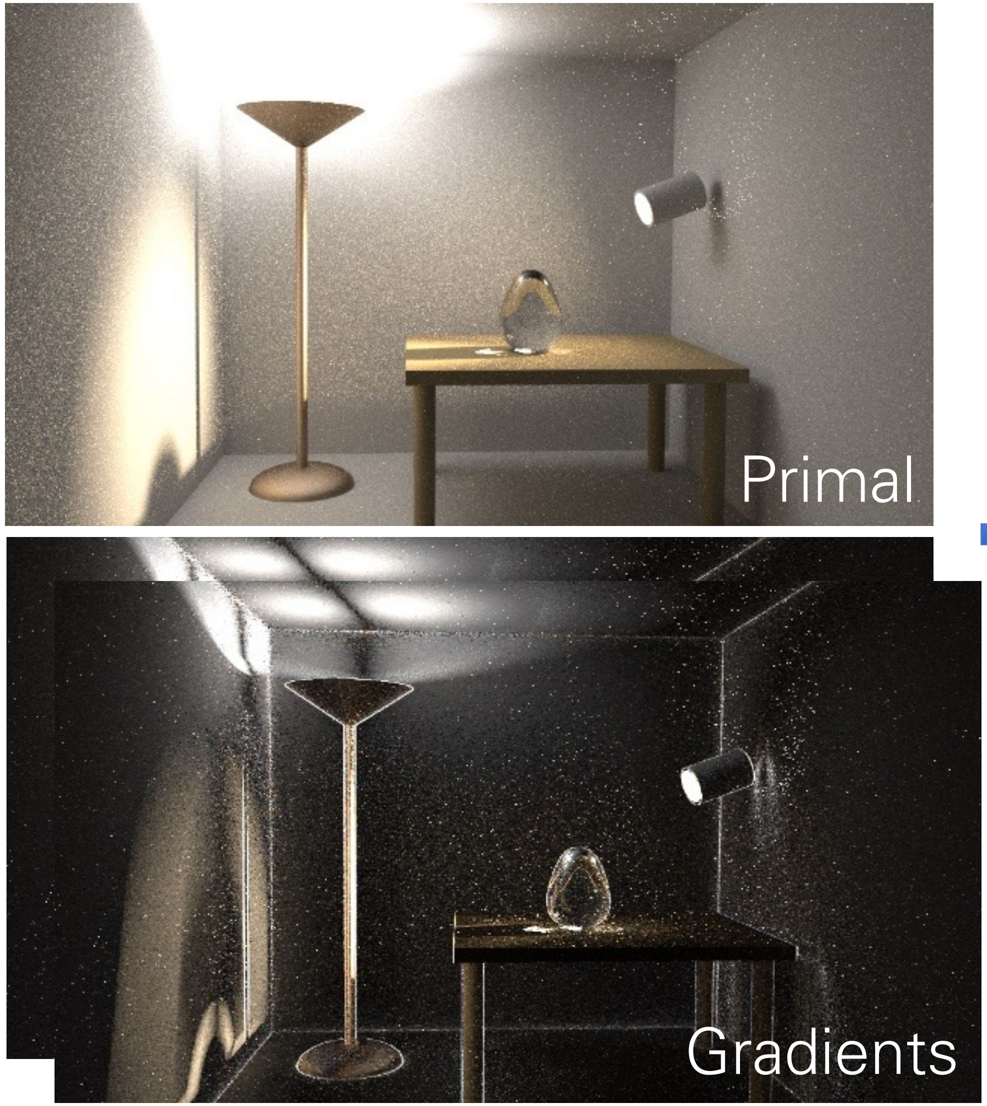
Idea

- Noise reduction by sampling **sparse** signal representation
  - Sparse: signal magnitude low, except in small regions
  - Wavelets, edge filters, **gradients,** etc.
  - Theoretical justification: Kettunen et al. SIGGRAPH 2015

The Basic Algorithm

1. Perform standard Monte Carlo rendering to obtain primal image
2. Sample gradients: horizontal and vertical
3. Reconstruct image from primal and gradients

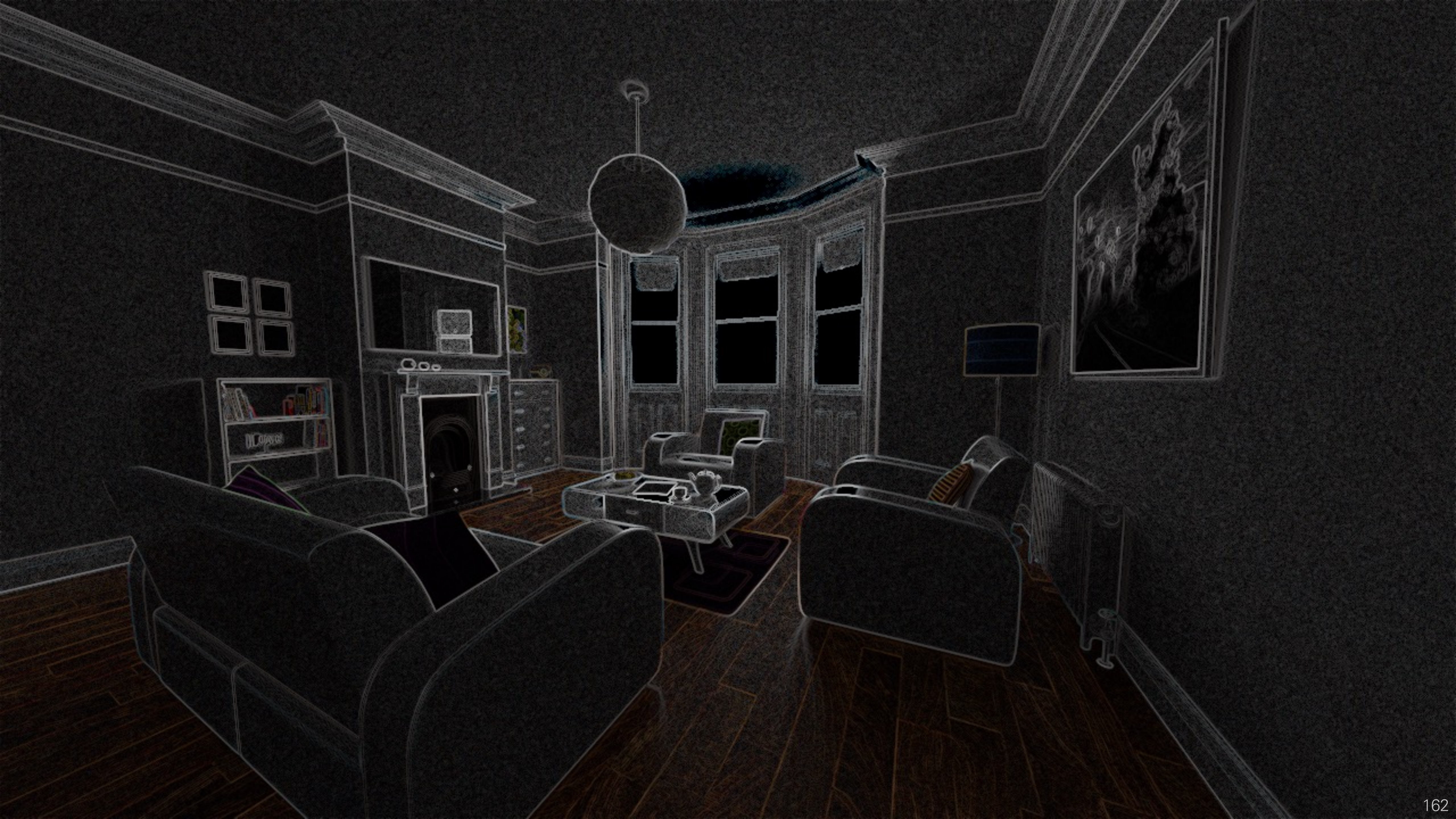
Image Reconstruction



Reconstructed image

Fusing **gradients** and **primal** information inside one image


Love

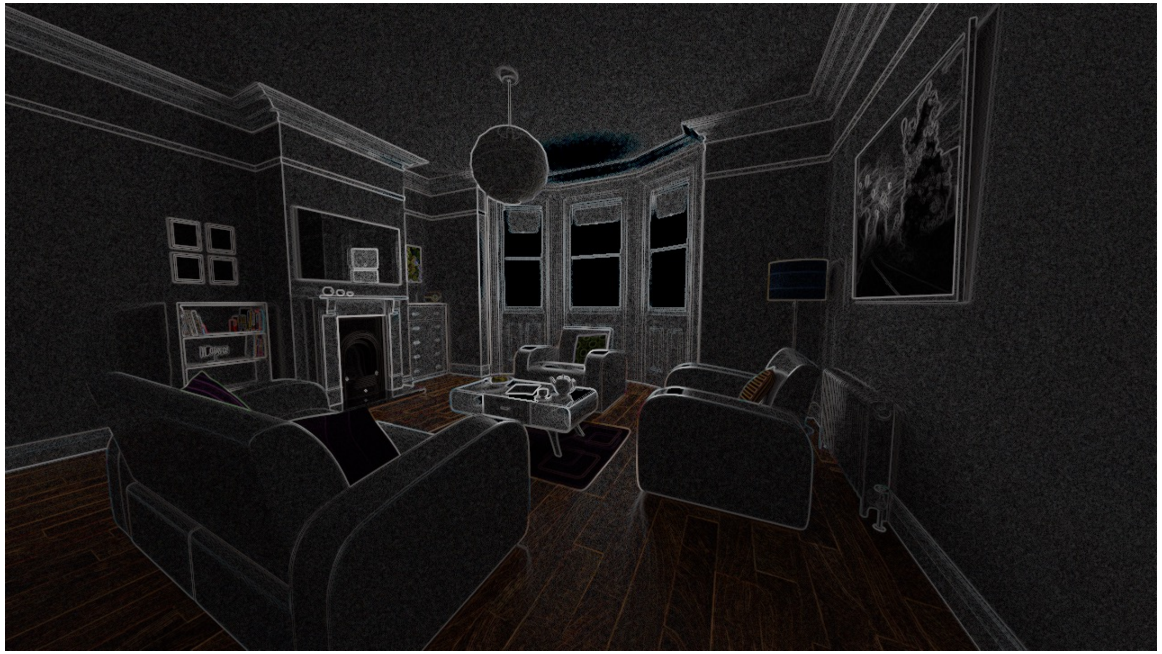


Primal domain

Gradient domain

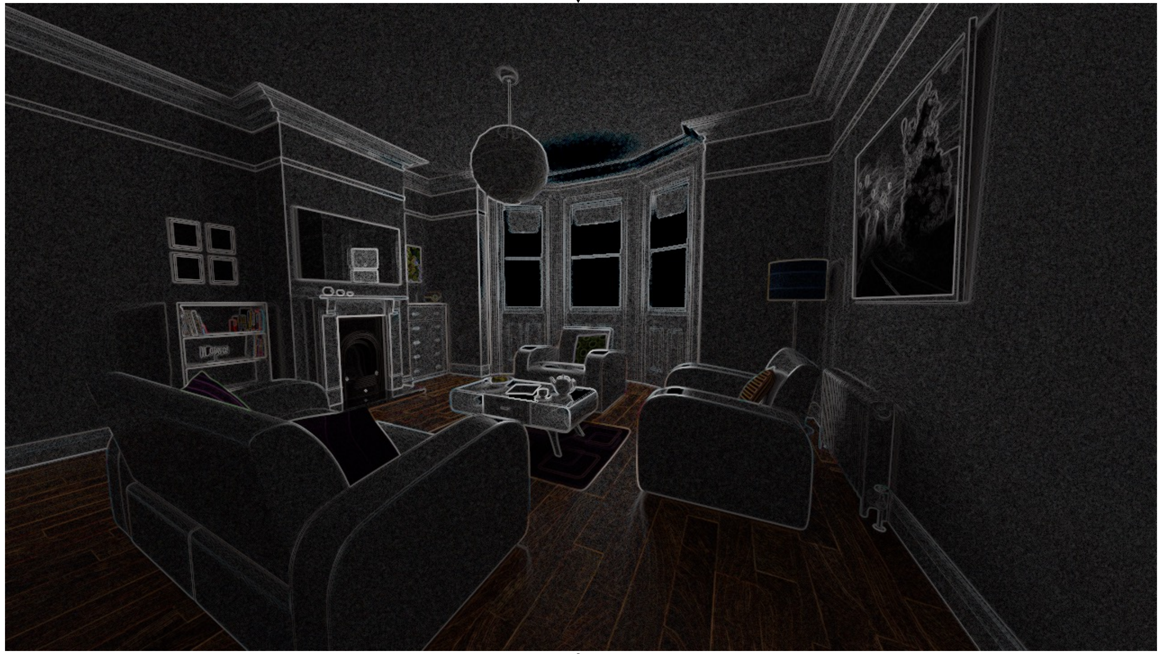

gradients of
natural images
are sparse
(close to zero
in most places)
Primal domain
Gradient domain

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)

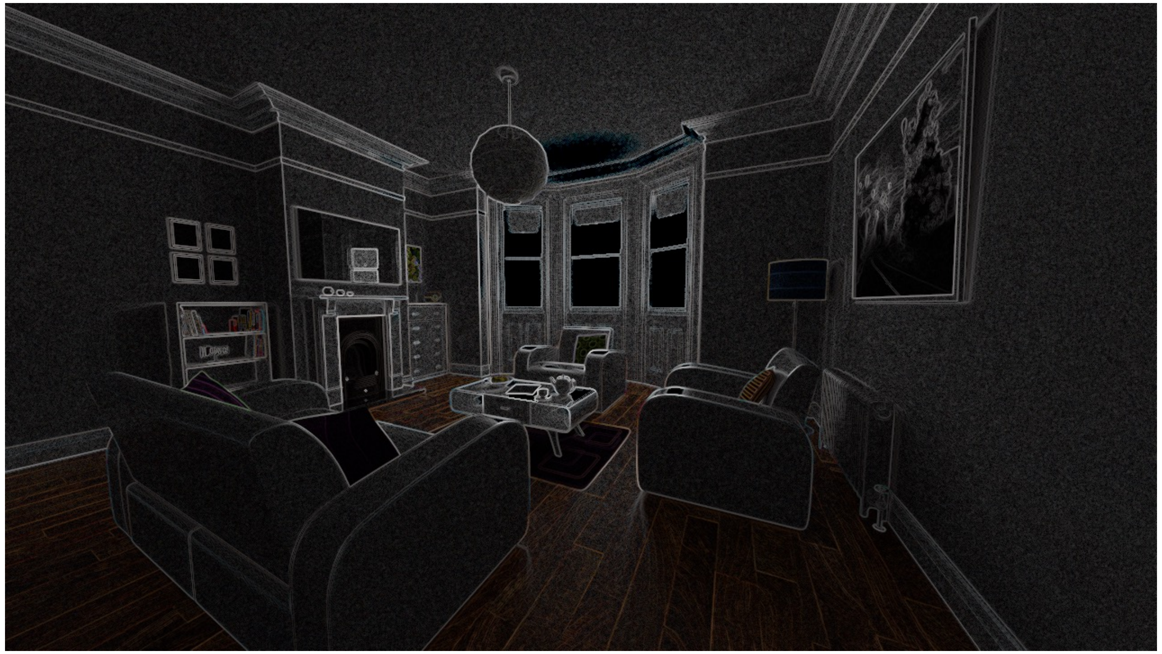


integration (e.g., Poisson solver)

Rendering

Primal-domain rendering: simulate intensities directly

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

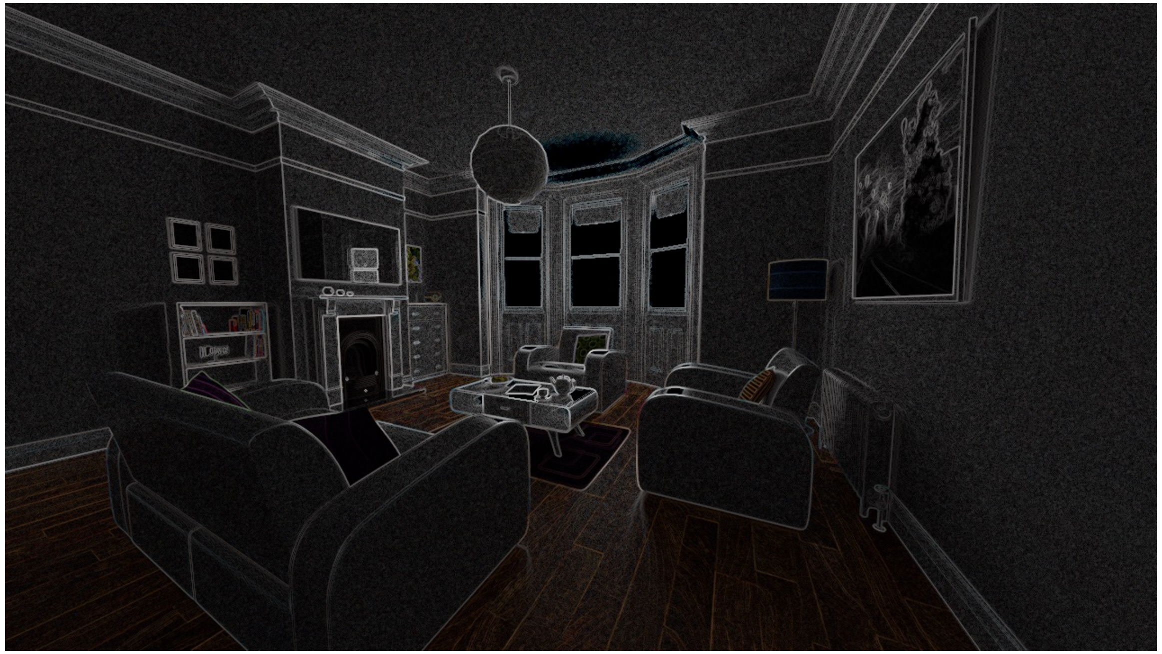


Why would gradient-domain rendering make sense?

Rendering

Primal-domain rendering: simulate intensities directly

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



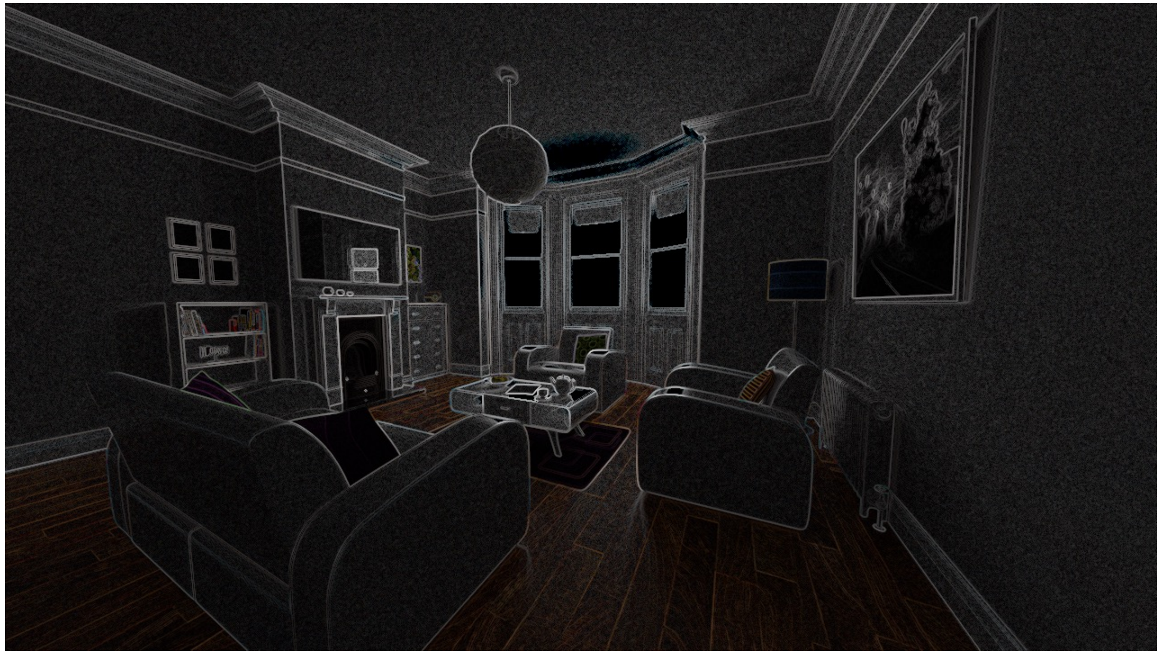
Why would gradient-domain rendering make sense?

- 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?

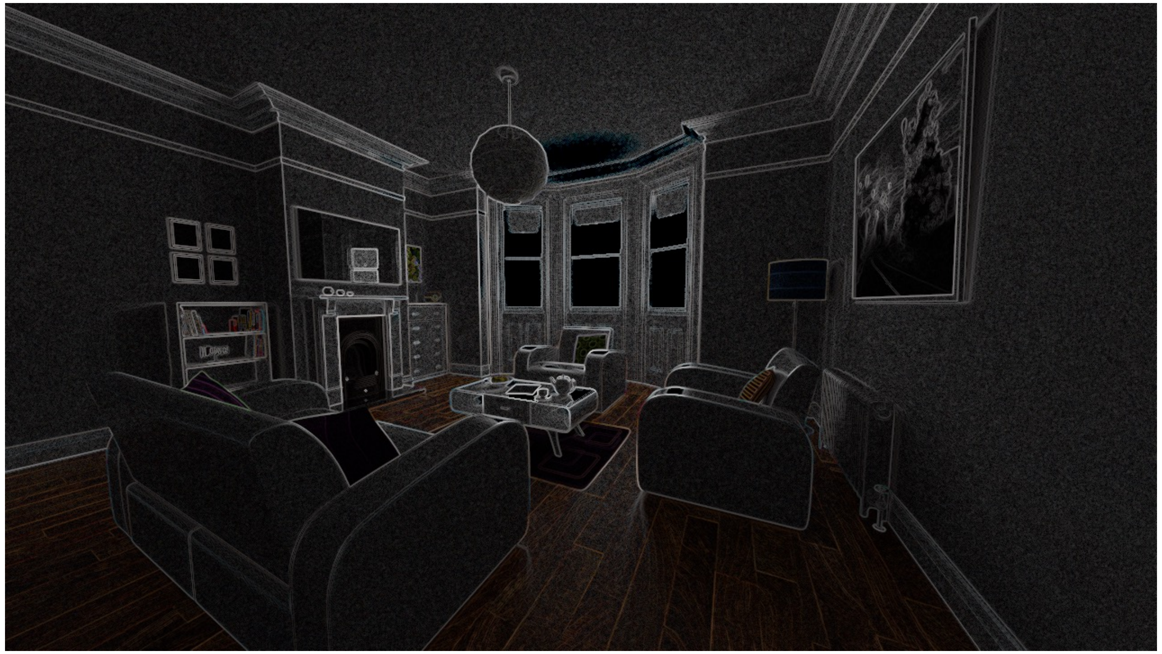
Why not all?

- 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

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

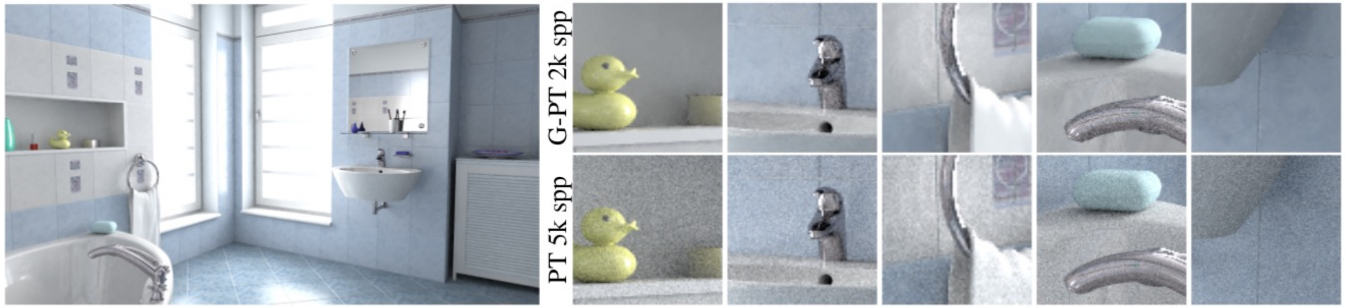


Figure 1: *Comparing gradient-domain path tracing (G-PT, L_1 reconstruction) to path tracing at equal rendering time (2 hours). In this time, G-PT draws about 2,000 samples per pixel and the path tracer about 5,000. G-PT consistently outperforms path tracing, with the rare exception of some highly specular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

Modern Gradient-Domain Rendering

Deep Convolutional Reconstruction For Gradient-Domain Rendering

MARKUS KETTUNEN, Aalto University
ERIK HÄRKÖNEN, Aalto University
JAAKKO LEHTINEN, Aalto University and Nvidia

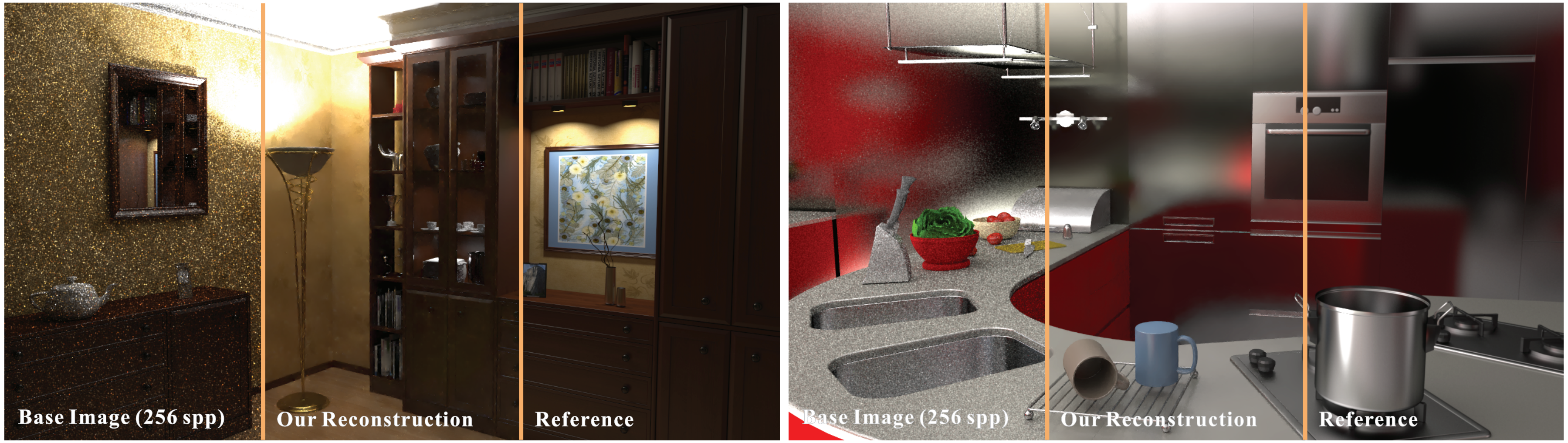
Input (Ours) | NFOR [Bitterli et al. 2016] | KPCN [Bako et al. 2017] | NGPT (Ours) | Ground Truth

https://github.com/mkettune/ngpt

Modern Gradient-Domain Rendering

GradNet: Unsupervised Deep Screened Poisson Reconstruction for Gradient-Domain Rendering

JIE GUO*, State Key Lab for Novel Software Technology, Nanjing University
MENGTIAN LI*, State Key Lab for Novel Software Technology, Nanjing University
QUEWEI LI, State Key Lab for Novel Software Technology, Nanjing University
YUTING QIANG, State Key Lab for Novel Software Technology, Nanjing University
BINGYANG HU, State Key Lab for Novel Software Technology, Nanjing University
YANWEN GUO†, State Key Lab for Novel Software Technology, Nanjing University
LING-QI YAN†, University of California, Santa Barbara



https://github.com/iRedBean/Deep-Poisson-Reconstruction

Gradient cameras

Gradient camera

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?

Gradient camera

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?

Gradient camera

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?

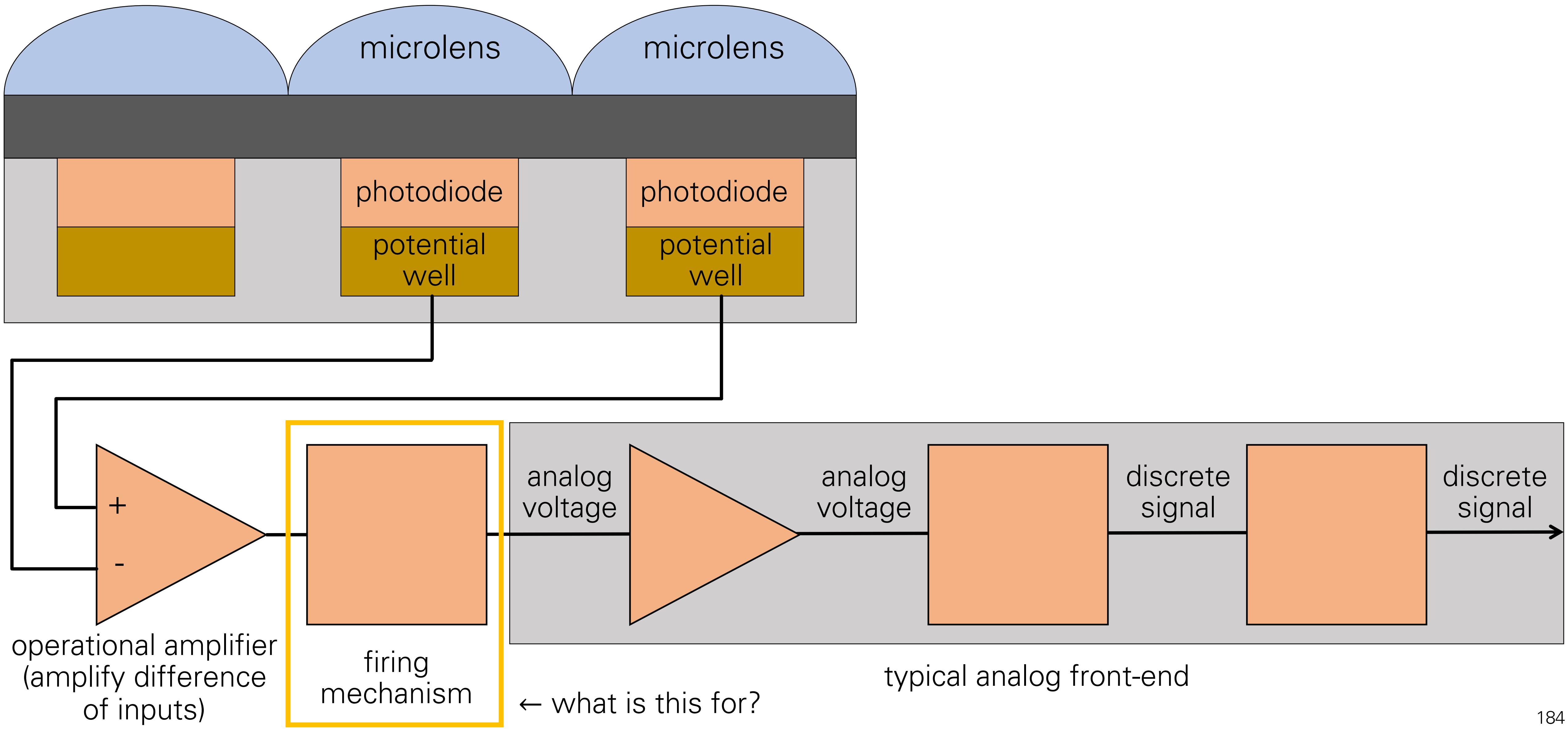
- You need to use a Poisson solver to reconstruct the image from the measured gradients.

How would you build a gradient camera?

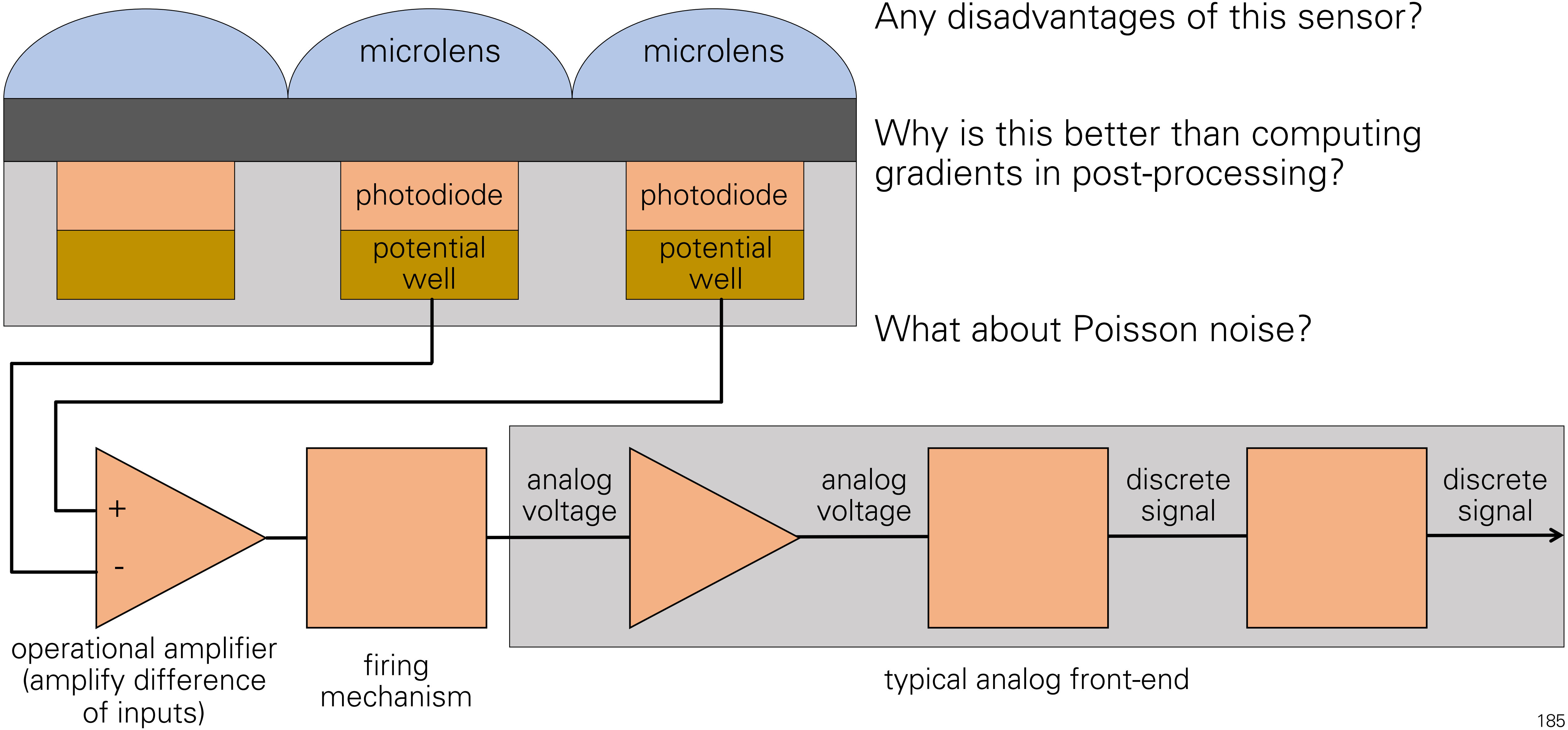
Change the sensor

Can you think how?

Change the sensor



Change the sensor

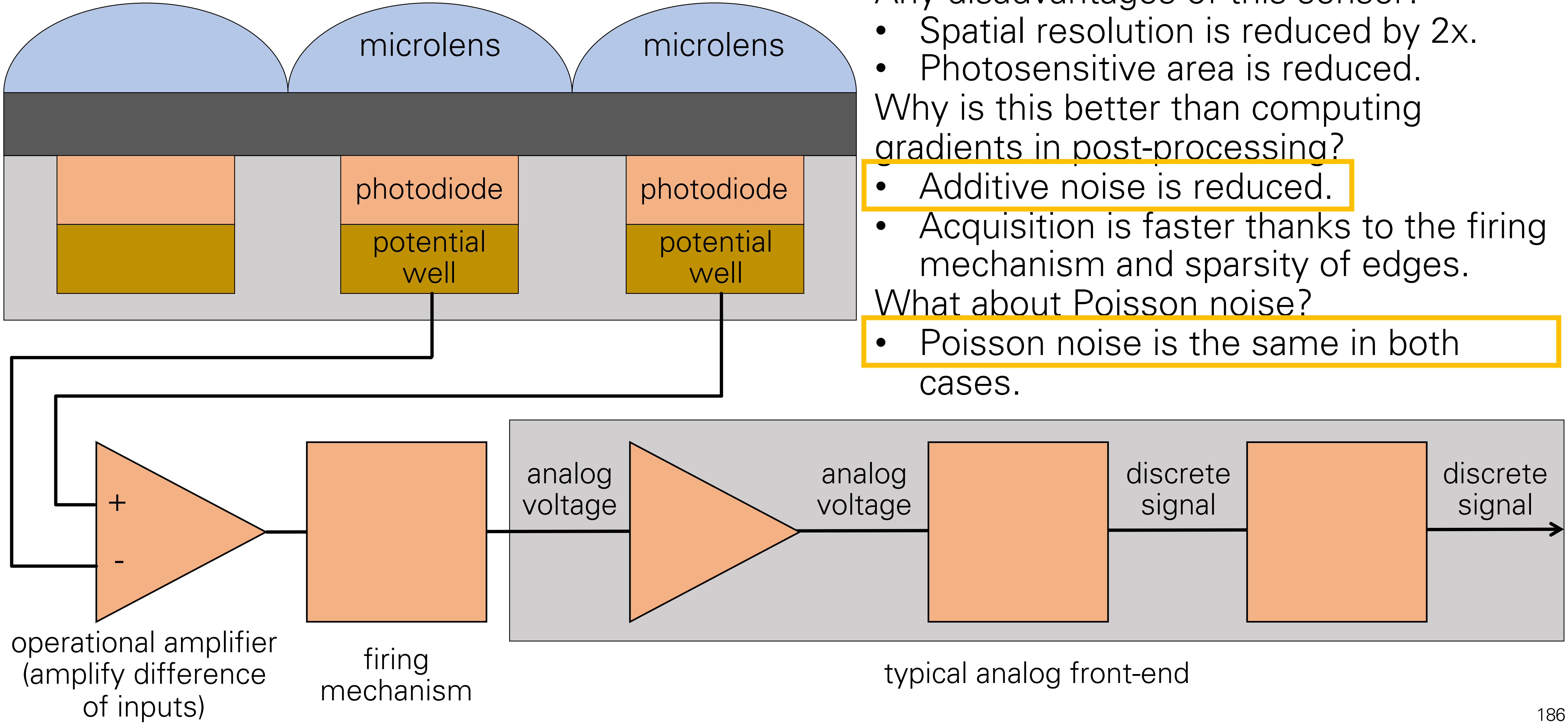


Any disadvantages of this sensor?

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?

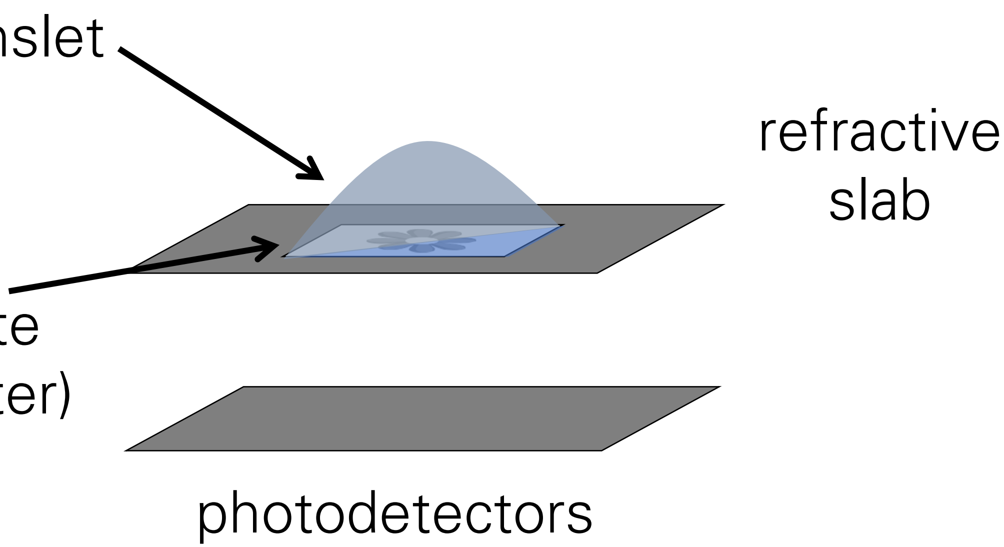
- Poisson noise is the same in both cases.

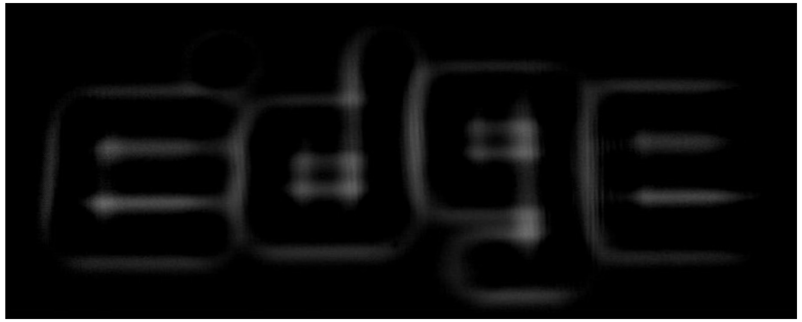
Change the optics

Can you think how?

Change the optics

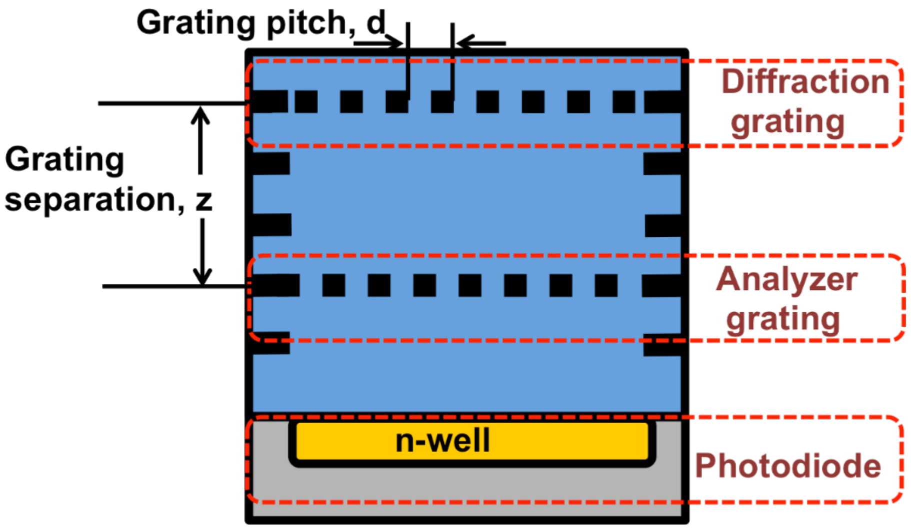
Optical filtering



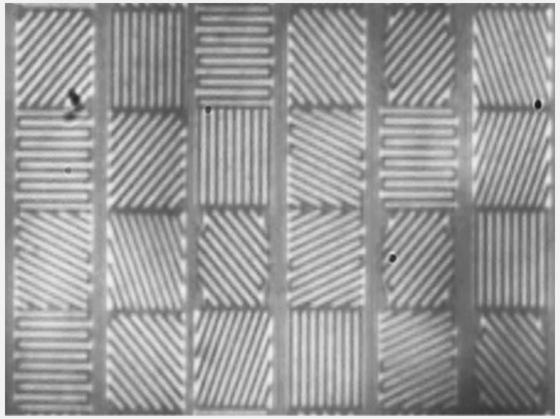


resulting image

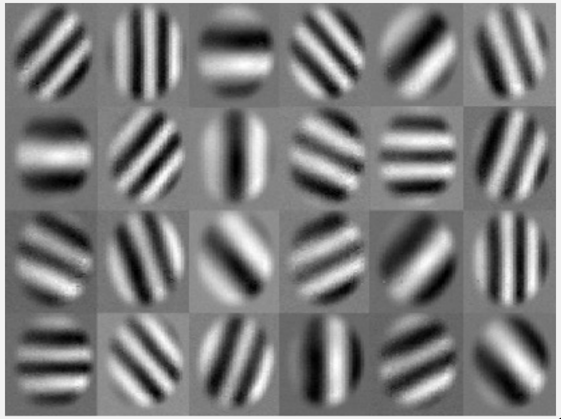
Angle-sensitive pixels



Physical Layout

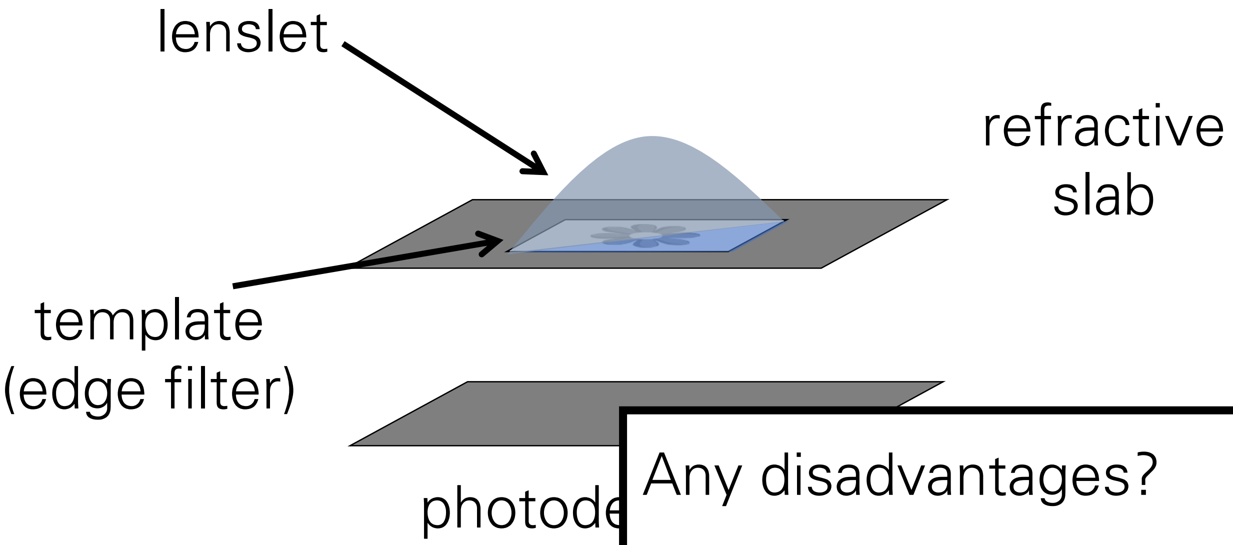


Impulse Response (2D)



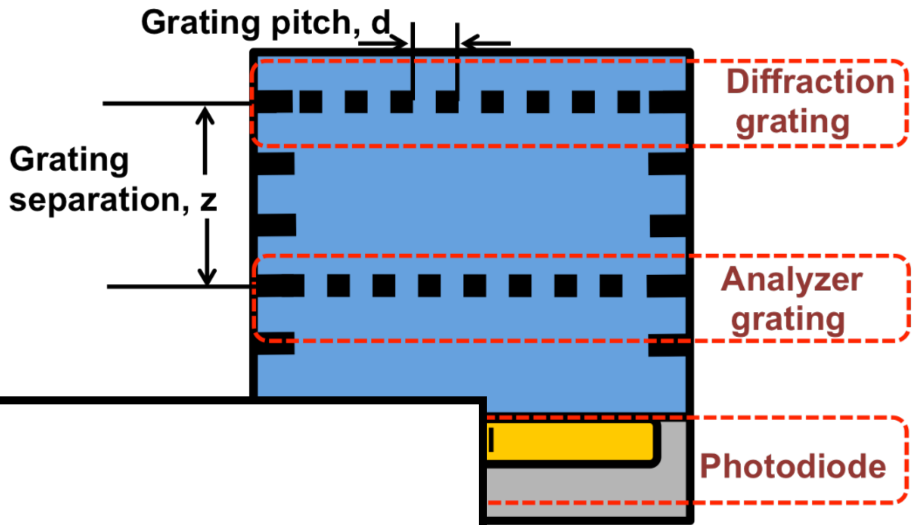
Change the optics

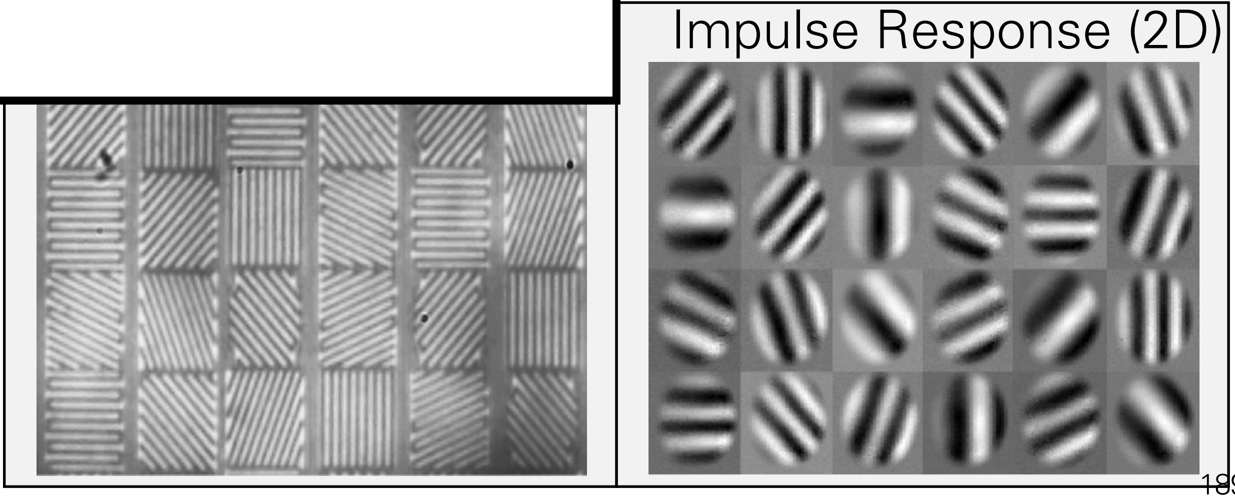
Optical filtering



resulting image

Angle-sensitive pixels





Any disadvantages?

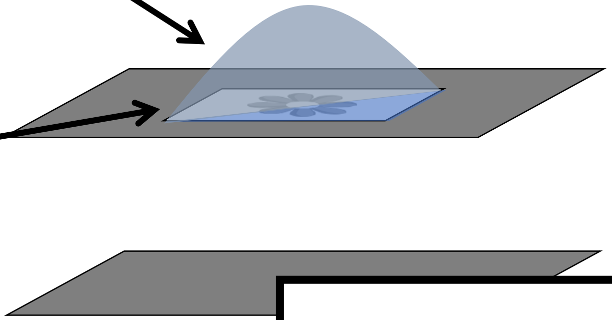
Change the optics

Optical filtering

lenslet

refractive slab

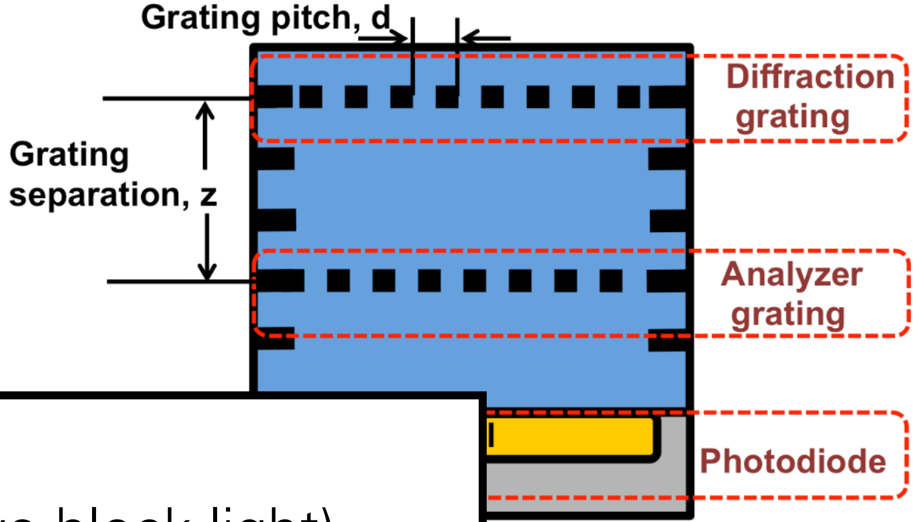
template (edge filter)

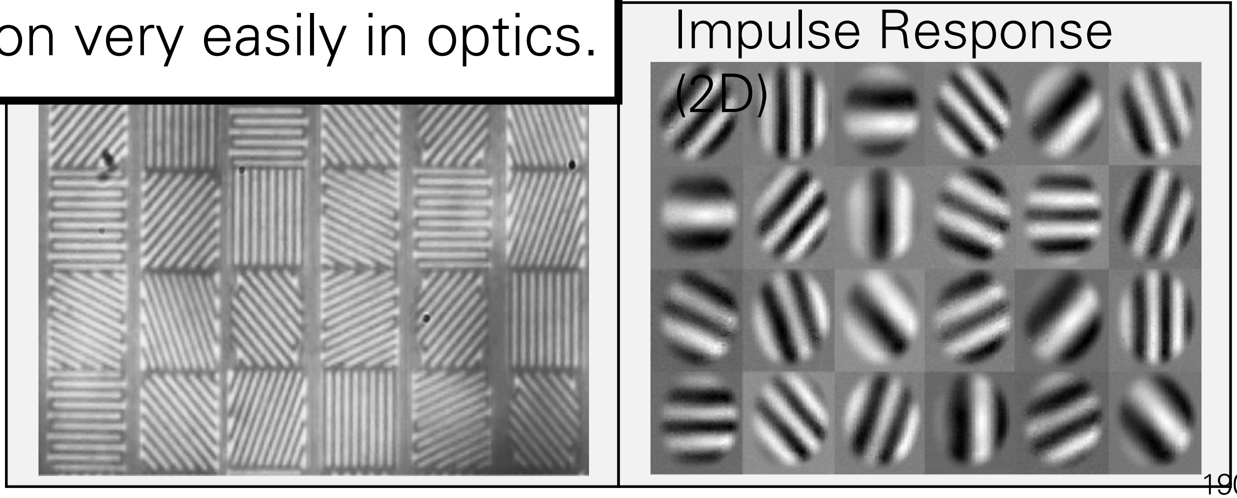


photode

resulting image

Angle-sensitive pixels





Any disadvantages?

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

Gradient camera

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?

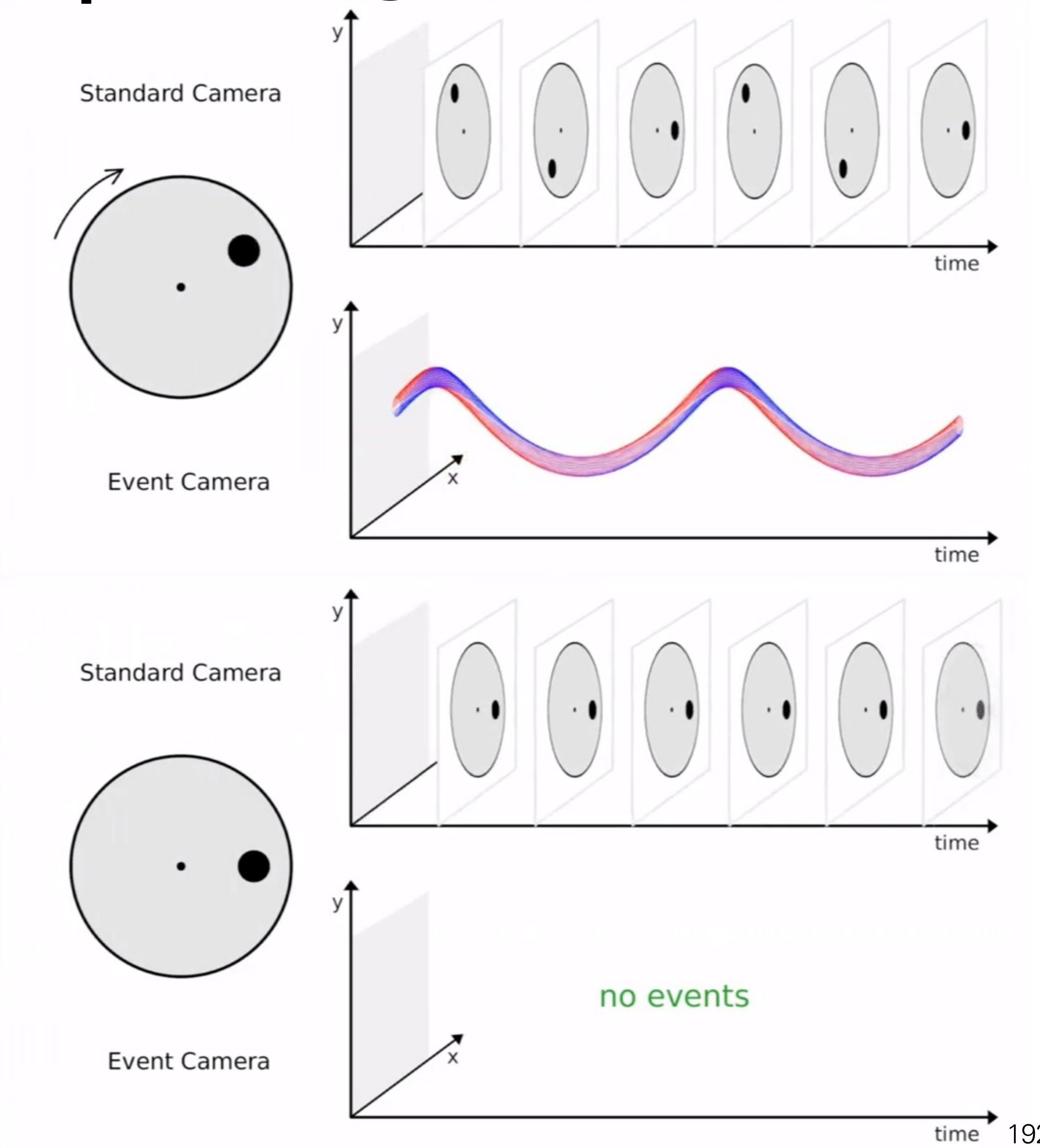
- You need to use a Poisson solver to reconstruct the image from the measured gradients.

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

Open Challenges in Computer Vision

- The past 60 years of research have been devoted to frame-based cameras.

  ...but they are not good enough!

Latency & Motion blur

Dynamic Range

- **Event cameras** do not suffer from these problems!

What is an event camera?

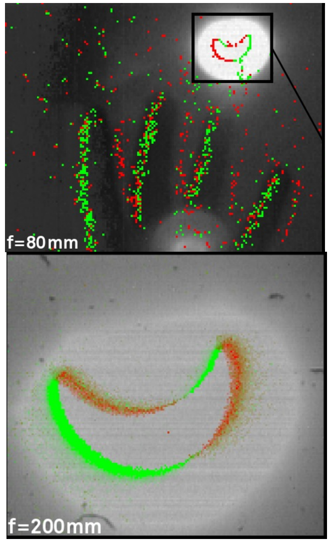
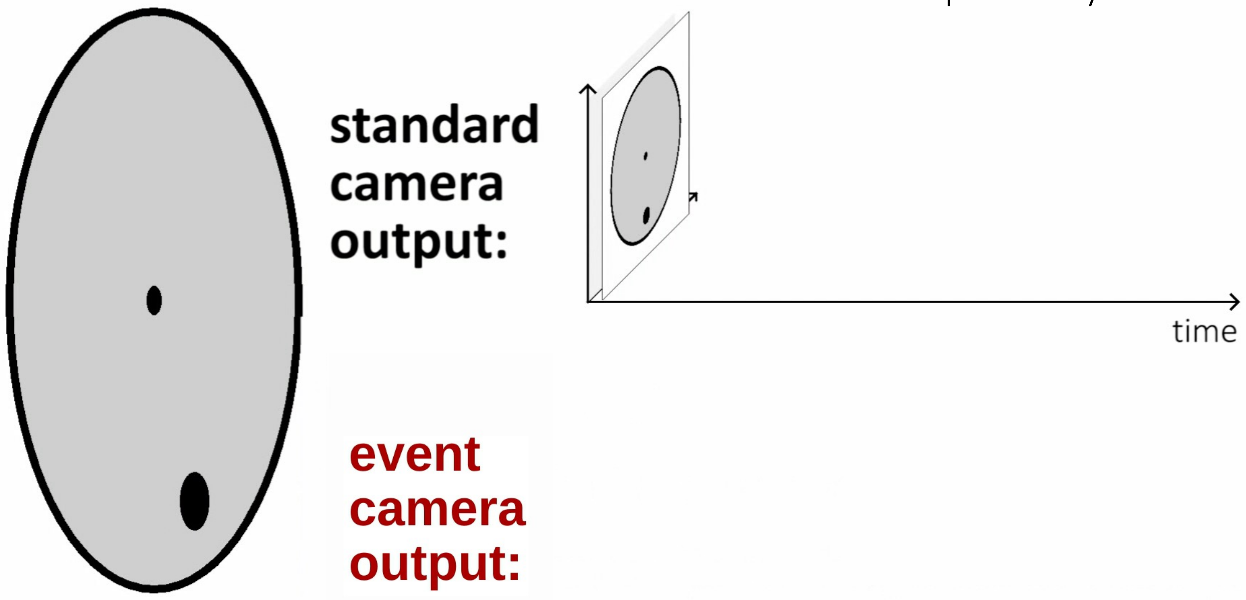


Image of the solar eclipse captured by a DVS

- Novel sensor that measures only **motion in the scene**
- **First commercialized in 2008** by T. Delbruck (UZH&ETH) under the name of Dynamic Vision Sensor (DVS)
- **Low-latency** (~ 1 μs)
- **No motion blur**
- **High dynamic range** (140 dB instead of 60 dB)
- **Ultra-low power** (mean: 1mW vs 1W)

Traditional vision algorithms cannot be used because:

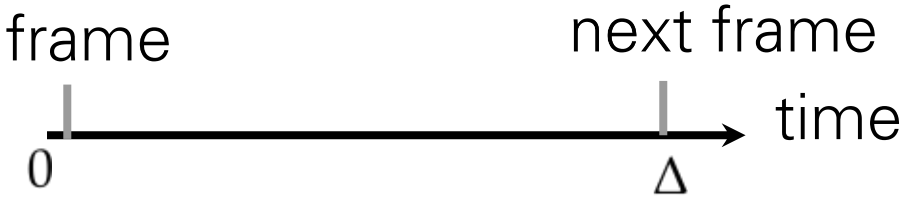
- **Asynchronous** pixels
- **No intensity information** (only binary intensity changes)



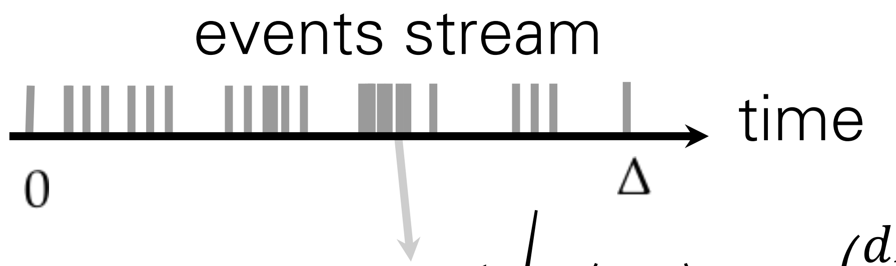
Lichtsteiner, Posch, Delbruck, A 128x128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor, 2008

Camera vs Event Camera

- A **traditional camera** outputs frames at **fixed time intervals**:



- By contrast, a **DVS** outputs **asynchronous events** at **microsecond resolution**. An event is generated each time a single pixel detects an intensity changes value

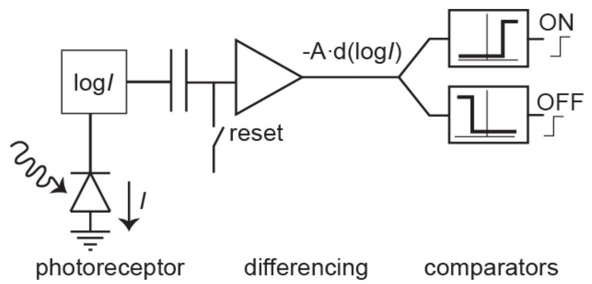


event: $\left\langle t, \langle x, y \rangle, sign\left(\frac{dI(x,y)}{dt}\right) \right\rangle$

- t: Timestamp (μs)
- $\langle x, y \rangle$: Pixel coordinates
- $sign\left(\frac{dI(x,y)}{dt}\right)$: Event polarity (or sign) (-1 or 1): increase or decrease of brightness

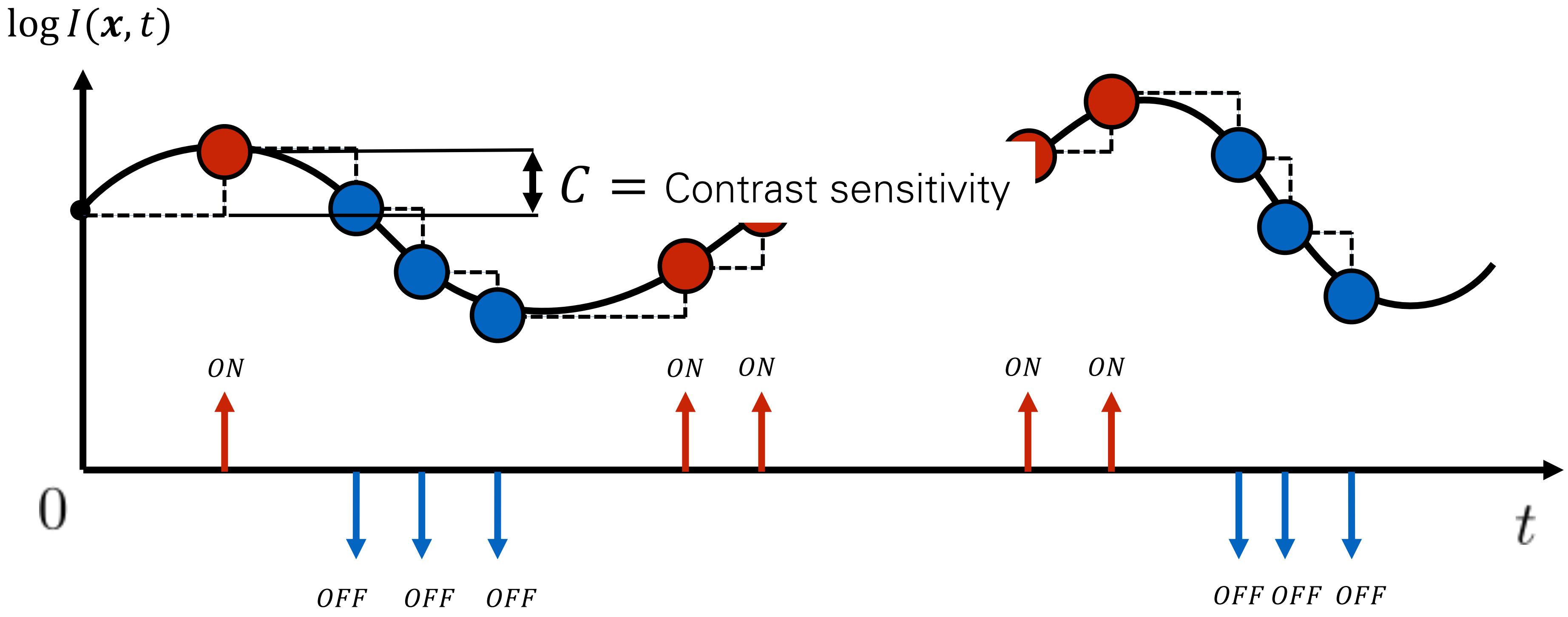
Lichtsteiner, Posch, Delbruck, A 128x128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor, 2008

Generative Event Model



Consider the intensity at a **single pixel**…

$$\pm C = \log I(\boldsymbol{x}, t) - \log I(\boldsymbol{x}, t - \Delta t)$$

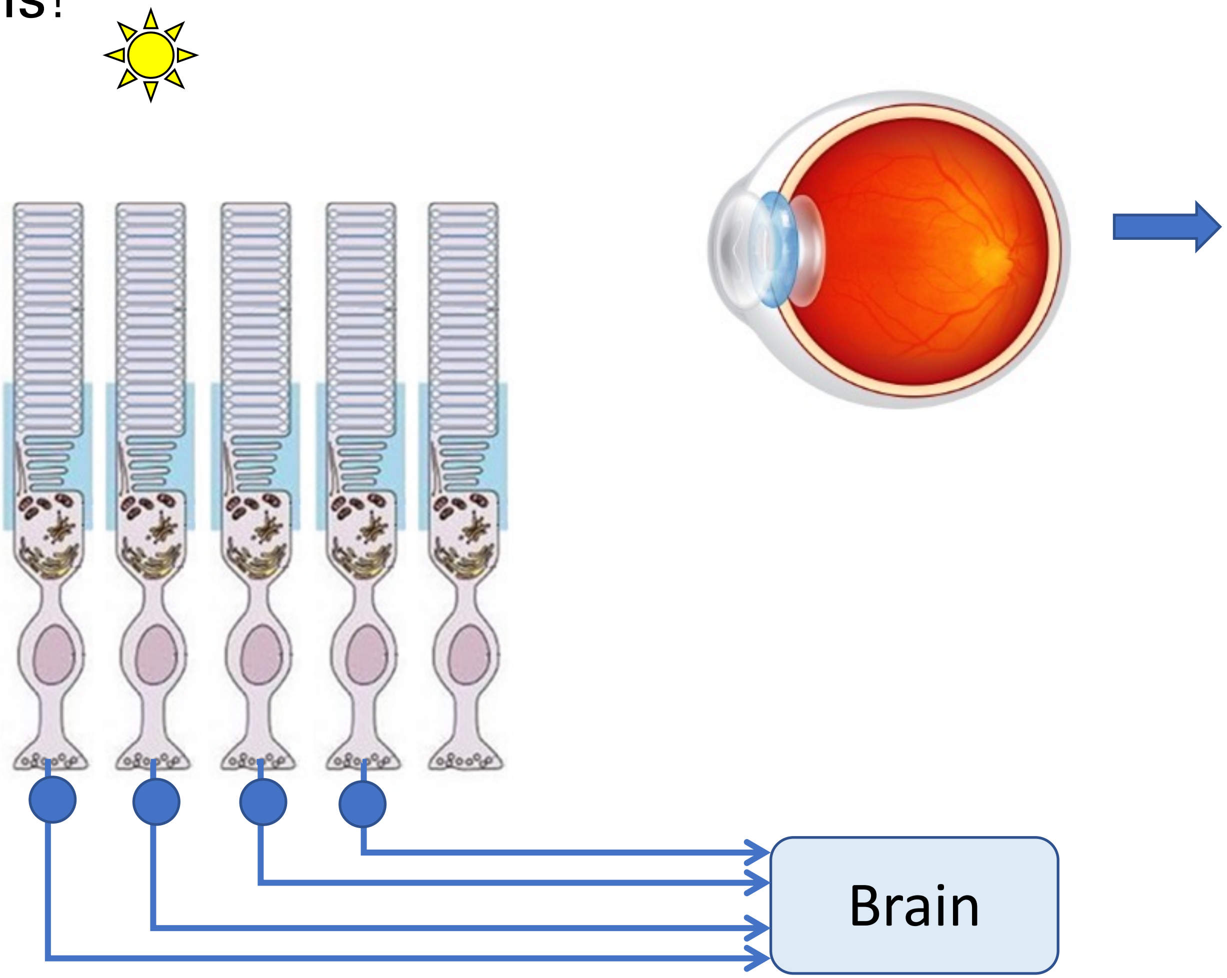


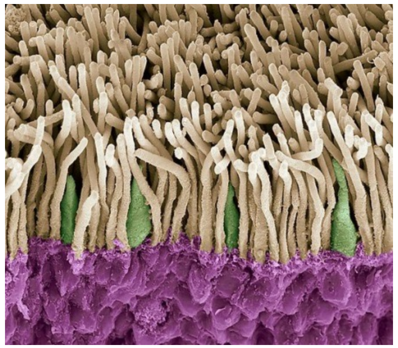
Events are triggered **asynchronously**

Lichtsteiner, Posch, Delbruck, A 128x128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor, 2008

Event cameras are inspired by the Human Eye

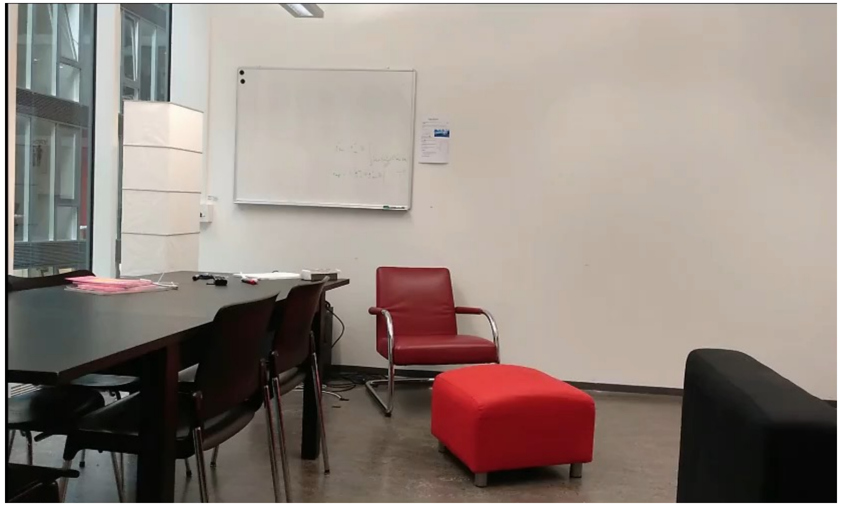
Human retina:
- 130 million **photoreceptors**
- But only 2 million **axons**!



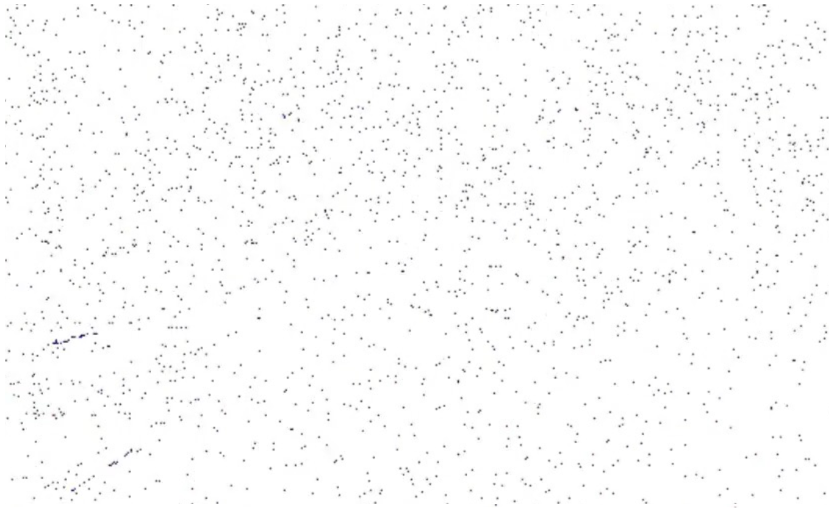


Event Camera Output with No Motion

Standard Camera



Event Camera (ON, OFF events)

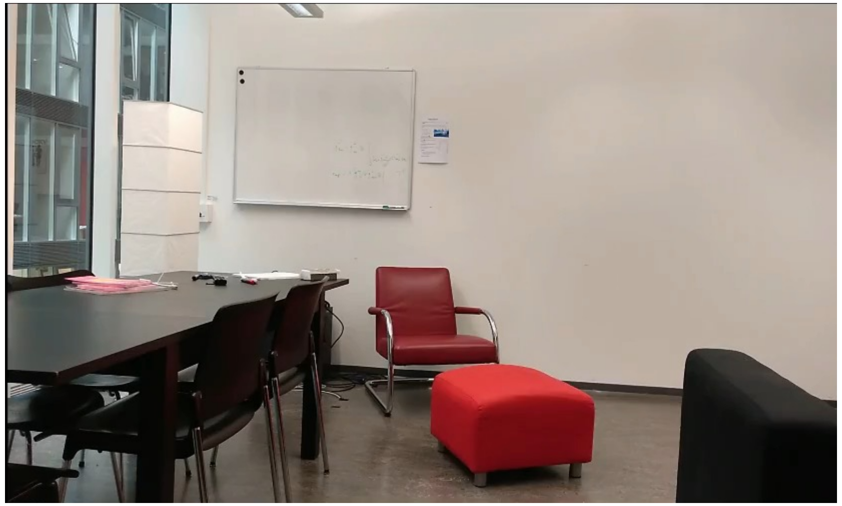


ΔT = 40 ms

Without motion, only background noise is output

Event Camera Output with Relative Motion

Standard Camera

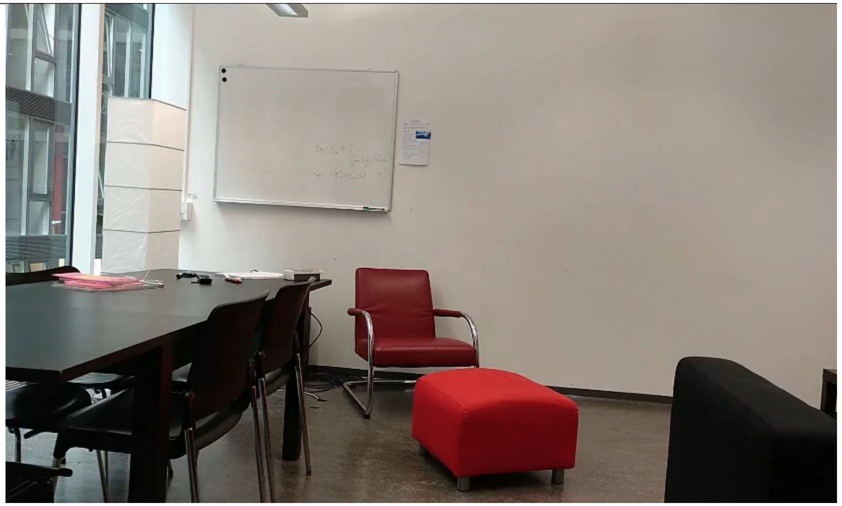


Event Camera (ON, OFF events)

$\Delta T = 10$ ms

Event Camera Output with Relative Motion

Standard Camera

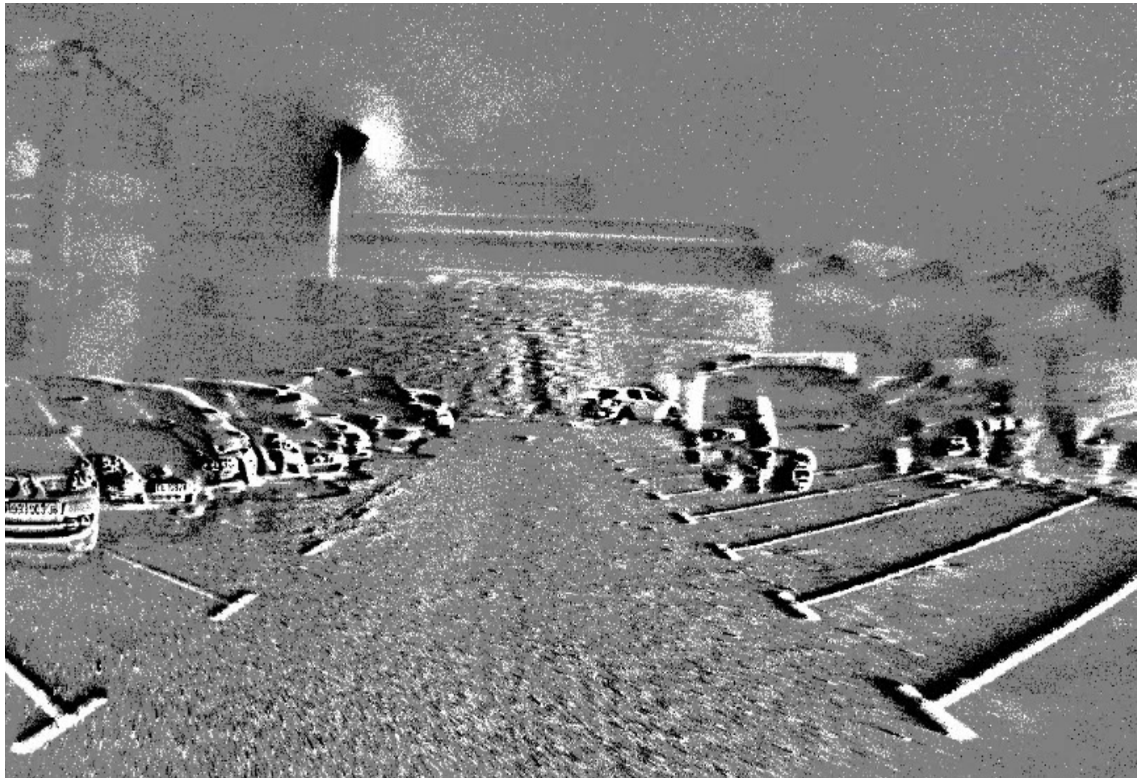


Event Camera (ON, OFF events)

ΔT = 40 ms

Low-light Sensitivity (night drive)

GoPro Hero 6

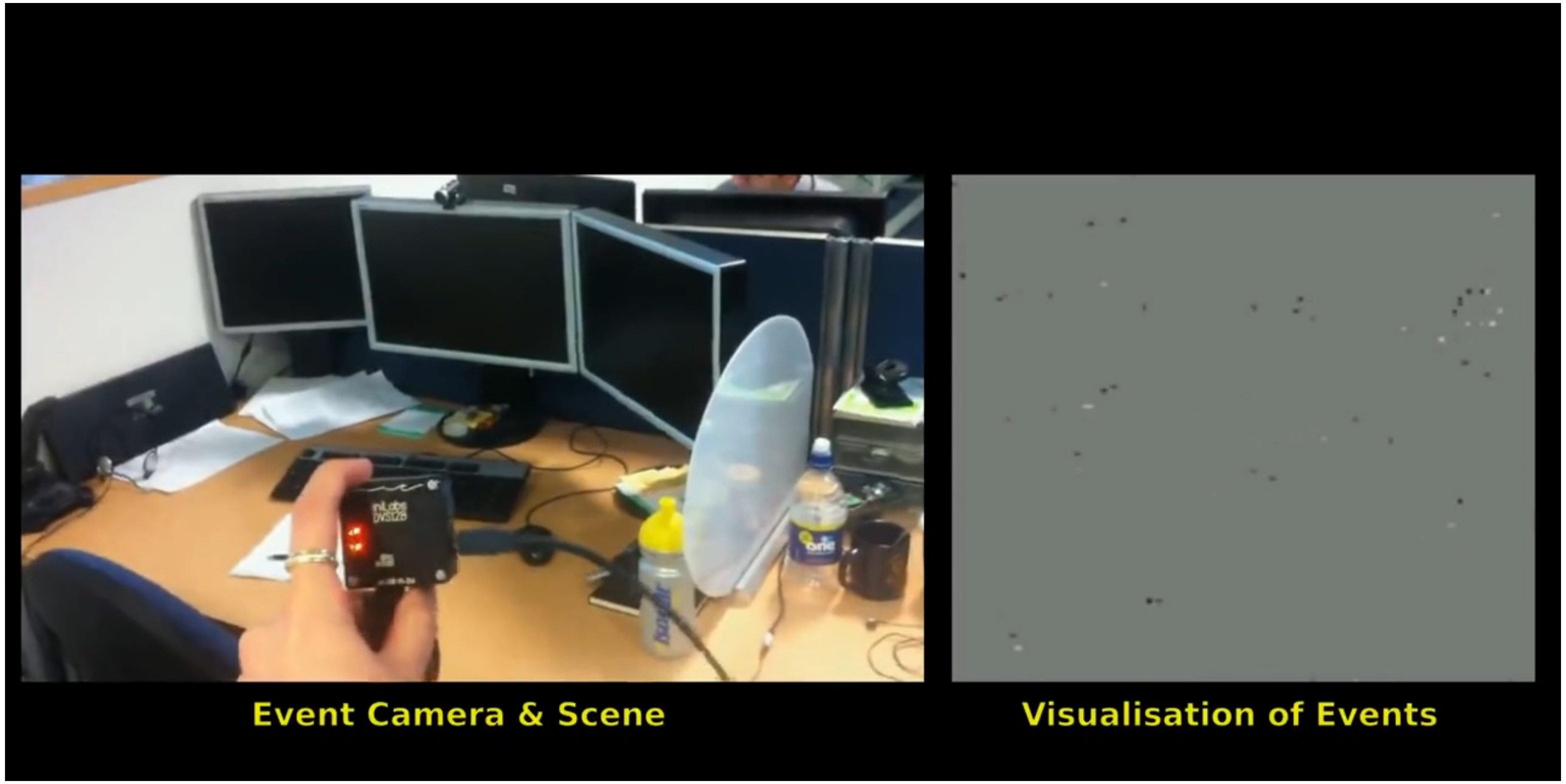


Event Camera by Prophesee
White = Positive events
Black = Negative events

Video credit: Prophesee - https://www.prophesee.ai

Image Reconstruction from Events

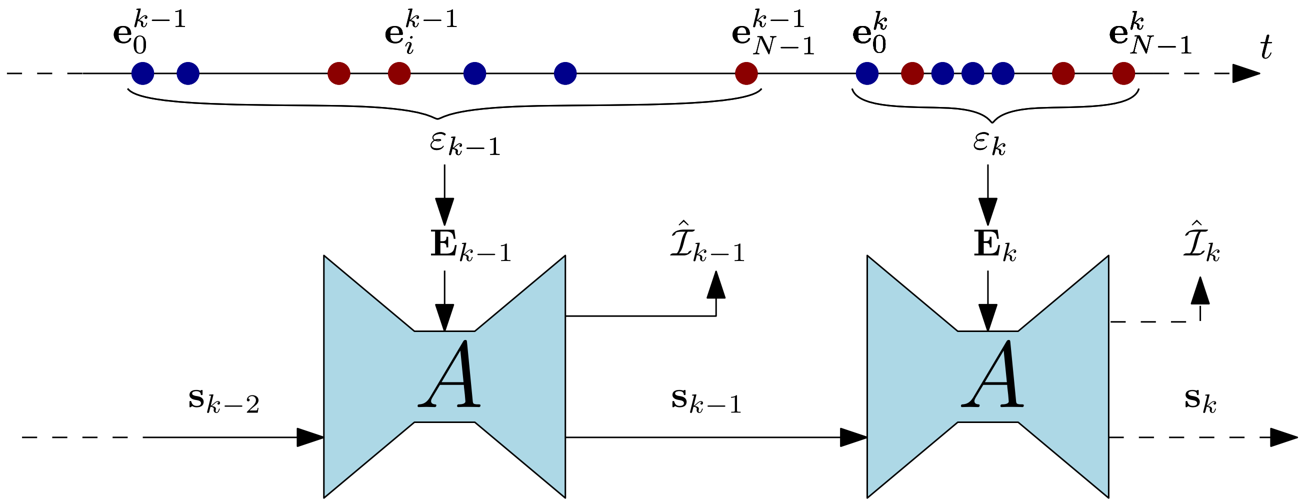
- Probabilistic simultaneous, gradient & rotation estimation from $C = -\nabla L \cdot \mathbf{u}$
- Obtain intensity from gradients via Poisson reconstruction
- The reconstructed image has super-resolution and high dynamic range (HDR)
- In real time on a GPU

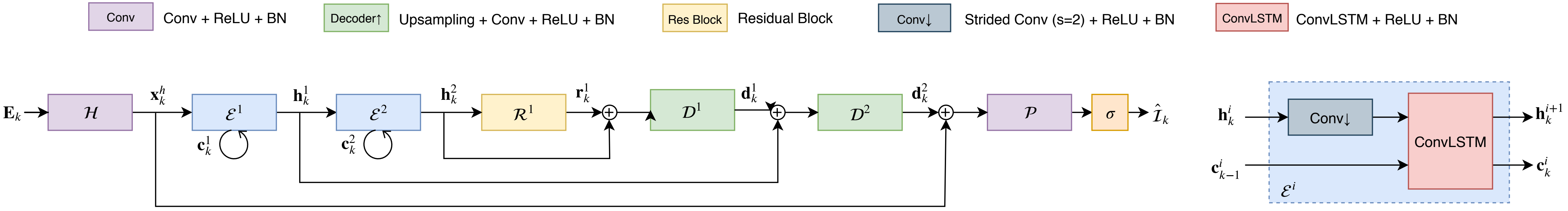


Kim et al., Simultaneous Mosaicing and Tracking with an Event Camera, BMVC'14

Image Reconstruction from Events – E2VID

- A fully convolutional, UNet-like architecture composed of recurrent encoder layers, followed by residual blocks and decoder layers, with skip connections between symmetric layers.

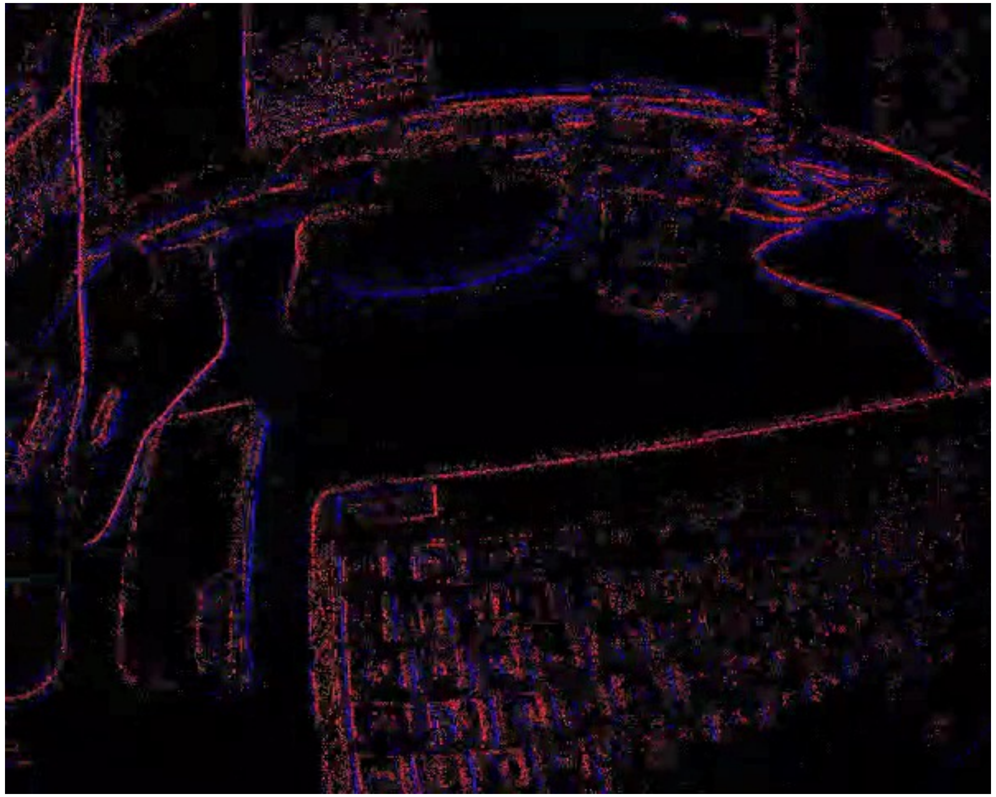




Rebecq et al., "Events-to-Video: Bringing Modern Computer Vision to Event Cameras", CVPR19.
Rebecq et al., "High Speed and High Dynamic Range Video with an Event Camera", PAMI, 2019.

Image Reconstruction from Events – E2VID

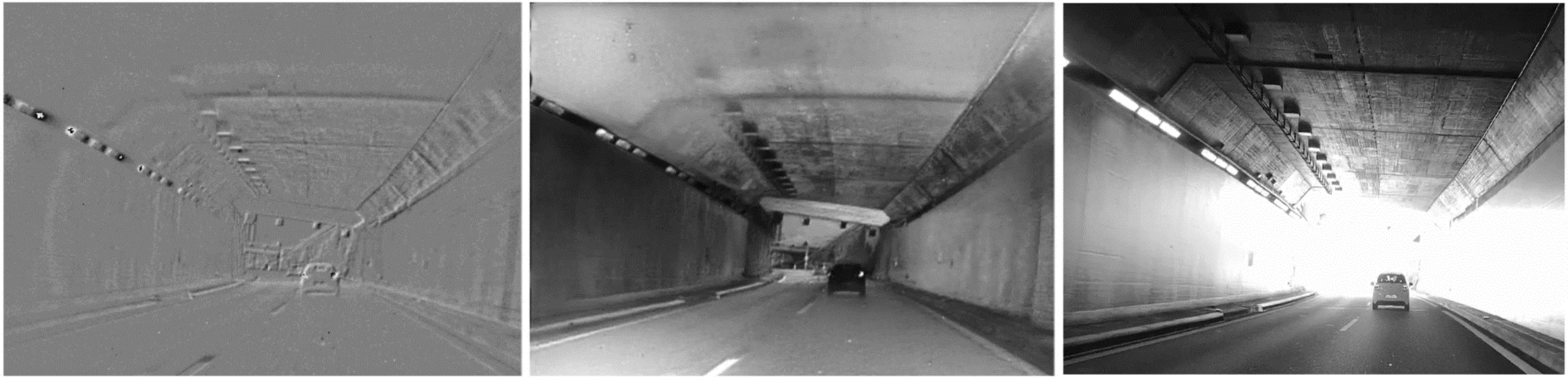
Events



Reconstructed image from events (Samsung DVS)

Rebecq et al., "Events-to-Video: Bringing Modern Computer Vision to Event Cameras", CVPR19.
Rebecq et al., "High Speed and High Dynamic Range Video with an Event Camera", PAMI, 2019.

HDR Video: Driving out of a tunnel



Events **Our reconstruction** **Phone camera**

Rebecq et al., "Events-to-Video: Bringing Modern Computer Vision to Event Cameras", CVPR19.
Rebecq et al., "High Speed and High Dynamic Range Video with an Event Camera", PAMI, 2019.

HDR Video: Night Drive

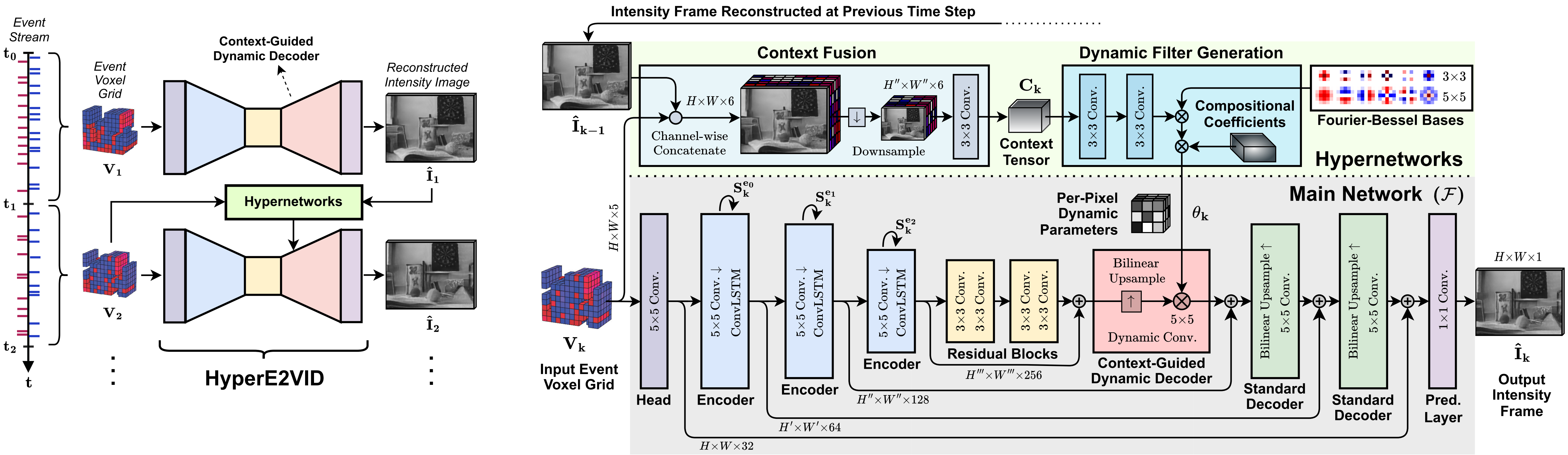
Our reconstruction from events

GoPro Hero 6

Rebecq et al., "Events-to-Video: Bringing Modern Computer Vision to Event Cameras", CVPR19.
Rebecq et al., "High Speed and High Dynamic Range Video with an Event Camera", PAMI, 2019.

Image Reconstruction from Events - HyperE2VID

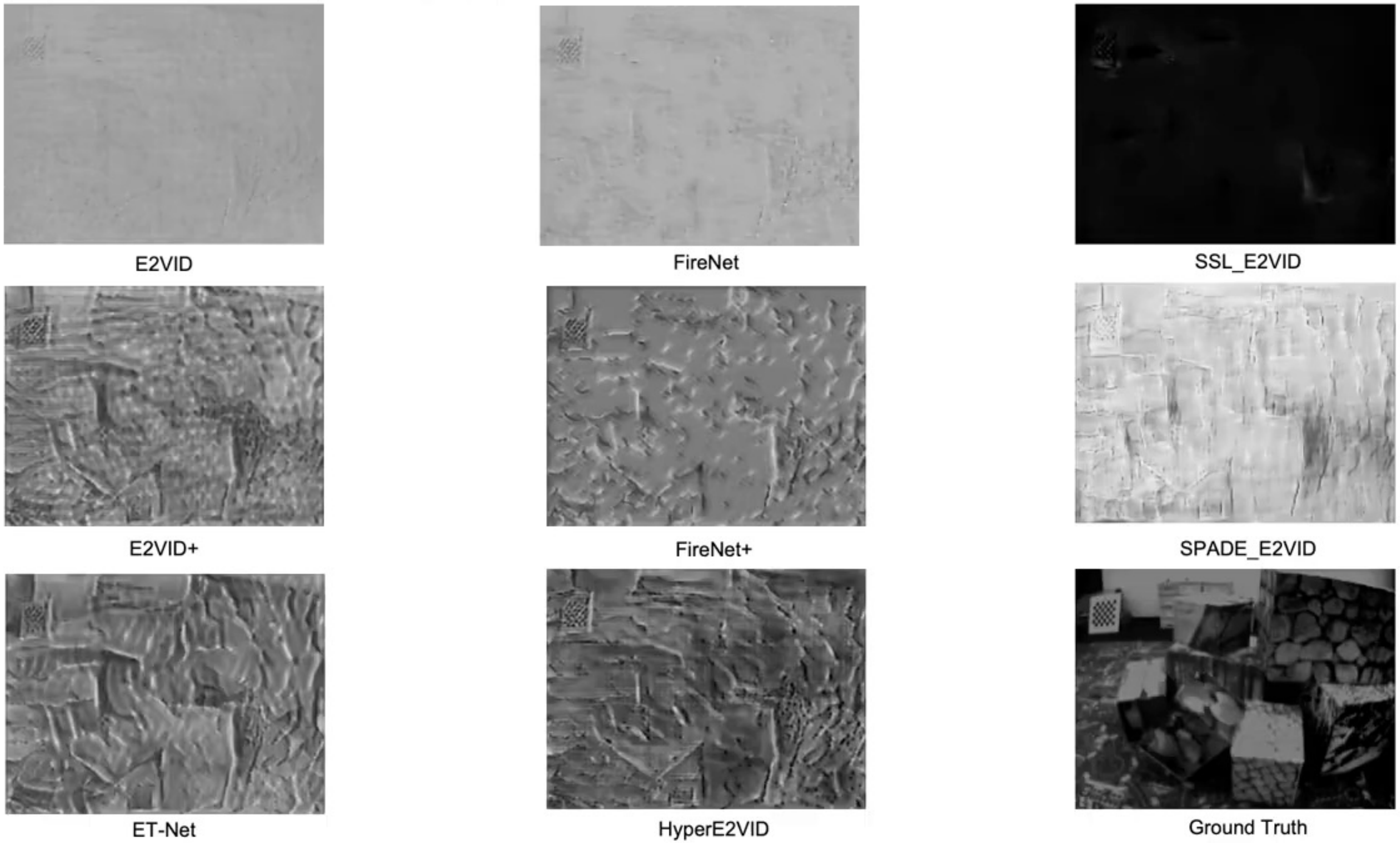
- A dynamic network architecture for the task of video reconstruction from events, where existing static architectures are extended with hypernetworks, dynamic convolutional layers, and a context fusion block.



Ercan et al., "HyperE2VID: Improving Event-Based Video Reconstruction via Hypernetworks", IEEE TIP, 2024.

Image Reconstruction from Events - HyperE2VID

boxes_6dof sequence from ECD dataset

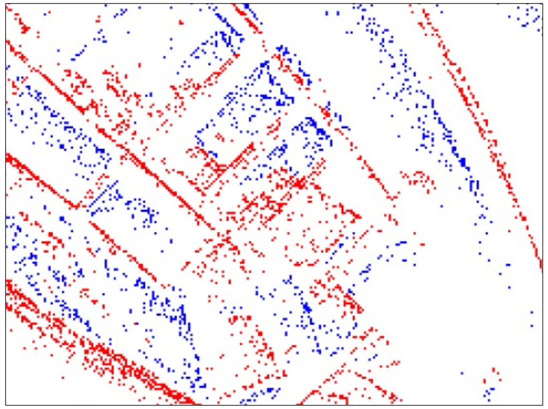


Ercan et al., “HyperE2VID: Improving Event-Based Video Reconstruction via Hypernetworks”, IEEE TIP, 2024.

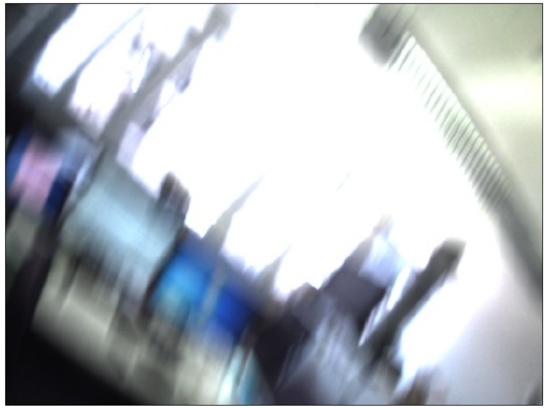
What if we combined the complementary advantages of event and standard cameras?

Why combining them?

	< 10 years research	> 60 years of research!



Event Camera

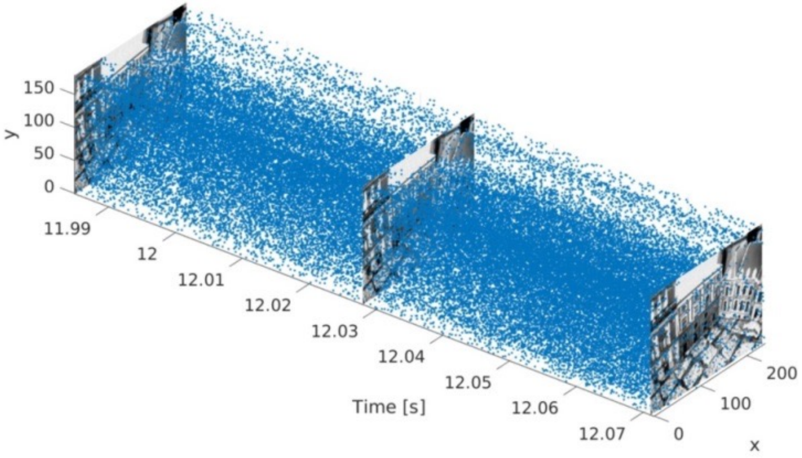


Standard Camera

	Event Camera	Standard Camera
Update rate	High (asynchronous): 1 MHz	Low (synchronous)
Dynamic Range	High (140 dB)	Low (60 dB)
Motion Blur	No	Yes
Static motion	No (event camera is a high pass filter)	Yes
Absolute intensity	No (reconstructable up to a constant)	Yes

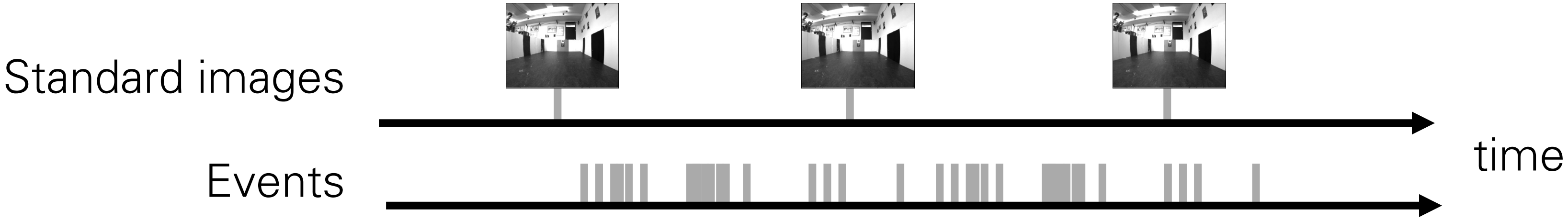
DAVIS sensor: Events + Images + IMU

- Combines an event and a standard camera in the same pixel array (→ the same pixel can both trigger events and integrate light intensity).
- It also has an IMU



Spatio-temporal visualization
of the output of a DAVIS sensor

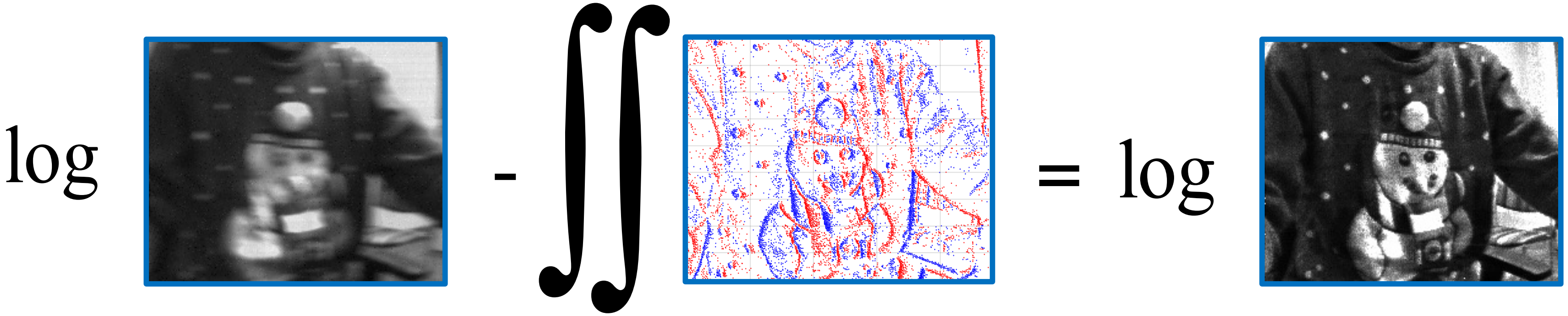
Temporal aggregation of events
overlaid on a DAVIS frame



Brandli et al. A 240x180 130dB 3us latency global shutter spatiotemporal vision sensor. IEEE JSSC, 2014

Deblurring a blurry video

- A **blurry image** can be regarded as the **integral of a sequence of latent images** during the exposure time, while the **events** indicate the **changes between the latent images.**
- **Finding:** sharp image obtained by subtracting the double integral of event from input image



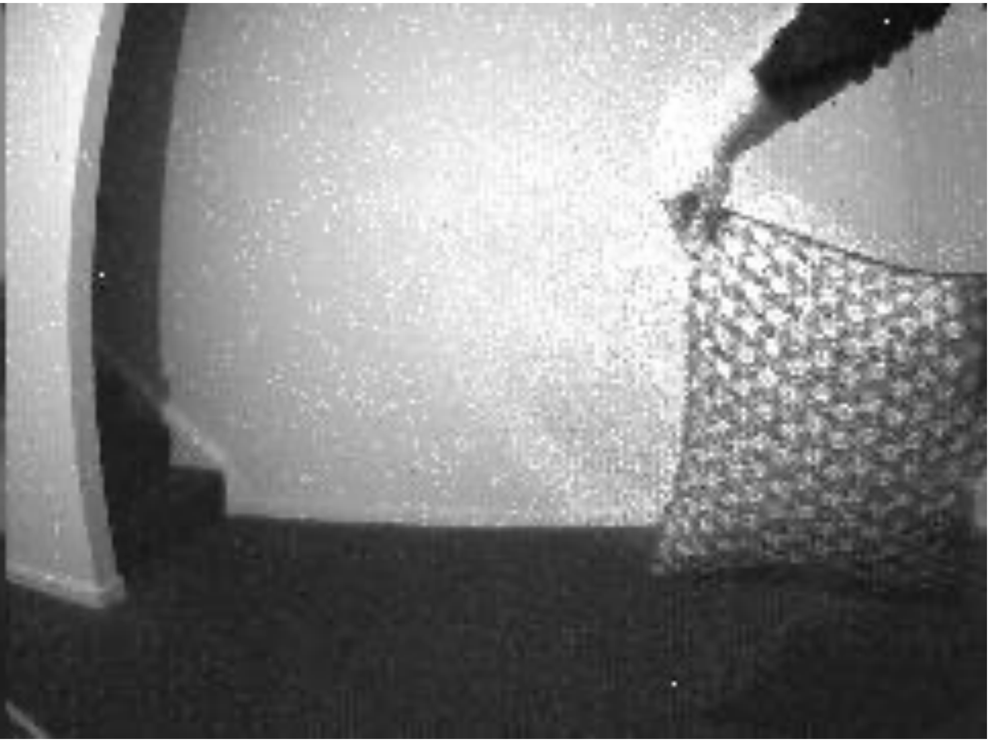
Input blur image   Input events   Output sharp image

Pan et al., Bringing a Blurry Frame Alive at High Frame-Rate with an Event Camera, CVPR 2019

Deblurring a blurry video

- A **blurry image** can be regarded as the **integral of a sequence of latent images** during the exposure time, while the **events** indicate the **changes between the latent images**.
- **Finding:** sharp image obtained by subtracting the double integral of event from input image

Input blur image

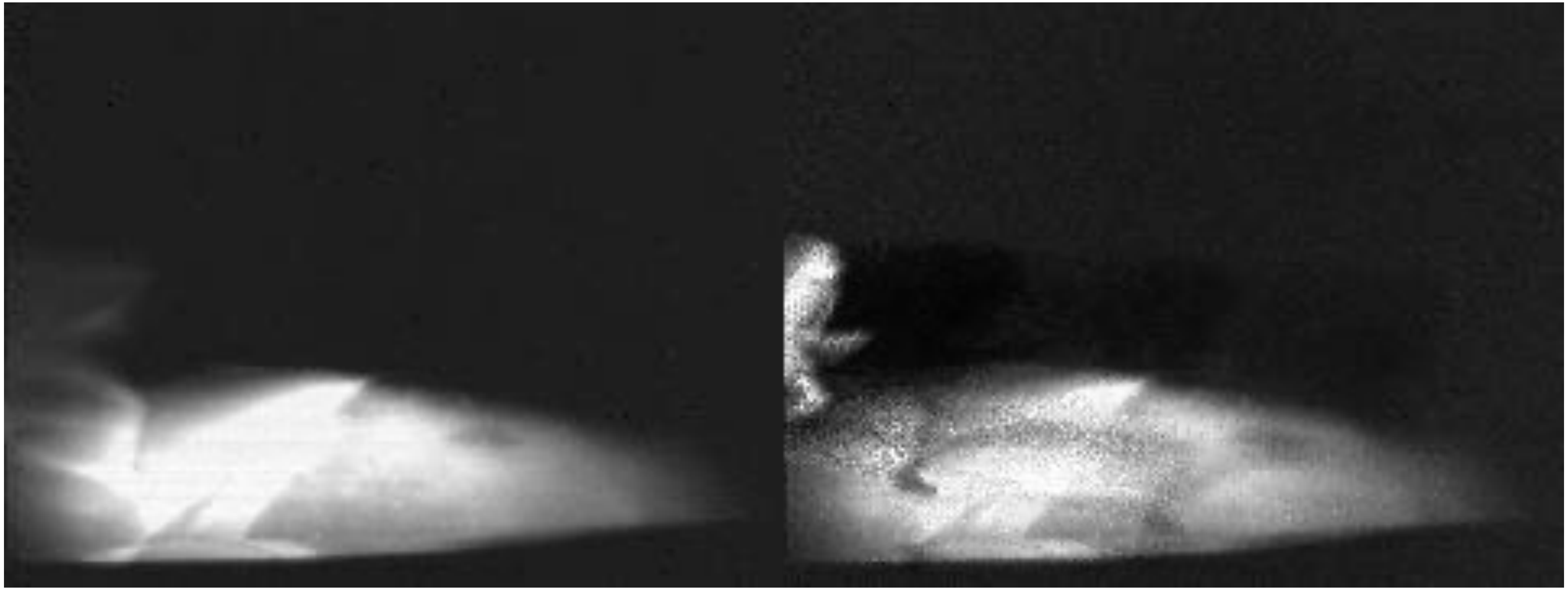


Output sharp video

Pan et al., Bringing a Blurry Frame Alive at High Frame-Rate with an Event Camera, CVPR 2019

Deblurring a blurry video

- A **blurry image** can be regarded as the **integral of a sequence of latent images** during the exposure time, while the **events** indicate the **changes between the latent images**.
- **Finding:** sharp image obtained by subtracting the double integral of event from input image

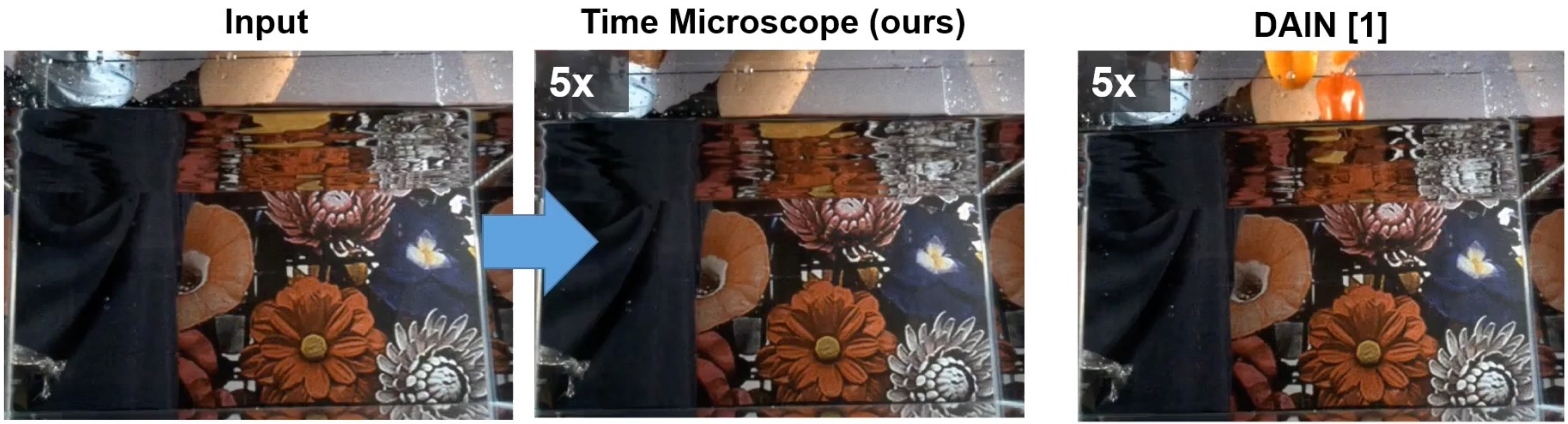


Input blur image　　　Output sharp video

Pan et al., Bringing a Blurry Frame Alive at High Frame-Rate with an Event Camera, CVPR 2019

Video Frame Interpolation

- Video frame interpolation methods aims at **generating intermediate frames** by **inferring object motions** in the image from **consecutive keyframes**.
- Motion is generally modelled with first-order approximations like **optical flow.**
  - This choice restricts the types of motions, leading to errors in highly dynamic scenarios.
- **Event cameras** provides **auxiliary visual information** in the blind-time **between frames**.



Tulyakov et al., Time Lens++: Event-based Frame Interpolation with Parametric Non-linear Flow and Multi-scale Fusion, CVPR 2022

Next Lecture:
Focal Stacks and Lightfields