Gradient-domain image processing

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

• Homework assignment 3 is out.
  - Due October 17th.
  - Generous bonus components.

• Extra reading group this Friday on color.
Overview of today’s lecture

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

Many of these slides were adapted from:

- Kris Kitani (15-463, Fall 2016).
- Fredo Durand (MIT).
- James Hays (Georgia Tech).
- Amit Agrawal (MERL).
- Jaakko Lehtinen (Aalto University).
Gradient-domain image processing
Application: Poisson blending

originals  copy-paste  Poisson blending
More applications

Removing Glass Reflections

Seamless Image Stitching
Yet more applications

Fusing day and night photos

Tonemapping
Entire suite of image editing tools

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

Pravin Bhat\textsuperscript{1} C. Lawrence Zitnick\textsuperscript{2} Michael Cohen\textsuperscript{1,2} Brian Curless\textsuperscript{1}

\textsuperscript{1}University of Washington \textsuperscript{2}Microsoft Research
Main pipeline

1. Original Images
2. Estimation of Gradients
3. Manipulation of Gradients
4. Edited Gradient Fields
5. Integration of Gradient Fields
6. Edited Images
Basics of gradients and fields
Some vector calculus definitions in 2D

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

\[ I(x, y): \mathbb{R}^2 \to \mathbb{R} \]

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

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

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.

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

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.

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

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

\[ \nabla I(x, y) = ? \]

Divergence: inner product of nabla with a vector field.

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

Curl: cross product of nabla with a vector field.

\[ \nabla \times [u(x, y) \ v(x, y)] = ? \]
Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

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

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

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

Divergence: inner product of nabla with a vector field.

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

Curl: cross product of nabla with a vector field.

\[
\nabla \times [u(x, y), v(x, y)] = \left( \frac{\partial v}{\partial x}(x, y) - \frac{\partial u}{\partial y}(x, y) \right) \hat{k}
\]
Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

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

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

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

Divergence: inner product of nabla with a vector field.

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

Curl: cross product of nabla with a vector field.

\[ \nabla \times [u(x, y) \ v(x, y)] = \left( \frac{\partial v}{\partial x}(x, y) - \frac{\partial u}{\partial y}(x, y) \right) \hat{k} \]
Some vector calculus definitions in 2D

Nabla (or del): vector differential operator.

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

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

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

Divergence: inner product of nabla with a vector field.

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

Curl: cross product of nabla with a vector field.

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

\[ \nabla = [ x \ y ] \]

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

\[ \nabla I = [ I_x \ I_y ] \]

Divergence: inner product of nabla with a vector field.

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

Curl: cross product of nabla with a vector field.

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

\[ I(x, y): \mathbb{R}^2 \to \mathbb{R} \]
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 $\nabla I(x, y) = \left[ \frac{\partial I}{\partial x}(x, y) \quad \frac{\partial I}{\partial y}(x, y) \right]$
Convert the *scalar* field into a *vector* field through differentiation.

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

**Vector field** $\nabla I(x, y) = \left[ \frac{\partial I}{\partial x}(x, y) \quad \frac{\partial I}{\partial y}(x, y) \right]$
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 \( h = 1 \).

\[
\frac{\partial I}{\partial x}(x, y) = I(x + 1, y) - I(x, y)
\]
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 \( h = 1 \).

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

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 \( h = 1 \).

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

Similarly for partial-\( y \) derivative.

\[
\frac{\partial I}{\partial y}(x, y) = I(x, y + h) - I(x, y)
\]
Discrete Laplacian

How do we compute the image Laplacian?

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

\[
\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} \ast \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix} \ast \begin{pmatrix} 1 \\ -1 \end{pmatrix} = ?
\]

What is this? What is this?
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:

\[
\begin{bmatrix} 1 & -1 \end{bmatrix} \ast \begin{bmatrix} 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & -1 \end{bmatrix} \ast \begin{bmatrix} 1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}
\]

\[ \frac{\partial^2 I}{\partial x^2}(x, y) \]

\[ \frac{\partial^2 I}{\partial y^2}(x, y) \]
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:

\[
\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \ast \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \ast \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}
\]

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

\[ \nabla \cdot ( \nabla (f)) = \Delta (f) \]

- gradient operator
- divergence operator
- Laplacian operator

Typically requires implementing derivatives in various differential operators differently.
Image gradients

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

Scalar field $I(x, y): \mathbb{R}^2 \rightarrow \mathbb{R}$ \quad \longrightarrow \quad 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 \(I\)?

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

such that

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

\[ \frac{\partial I}{\partial y}(x, y) = v(x, y) \]
Property of twice-differentiable functions

Curl of the gradient field equals zero:

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

What does that mean intuitively?
Property of twice-differentiable functions

Curl of the gradient field should be zero:

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

What does that mean intuitively?
• Same result independent of order of differentiation.

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

image $I$

$I_x$  $I_y$

$\Delta I$  $\nabla \times \nabla I$  $I_{xy}$  $I_{yx}$
Property of twice-differentiable functions

Curl of the gradient field should be zero:

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

What does that mean intuitively?

- Same result independent of order of differentiation.

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

Can you use this property to derive an integrability condition?
Integrable fields

Given an arbitrary vector field \((u, v)\), can we always integrate it into a scalar field \(I\) such that

\[
\begin{align*}
\frac{\partial I}{\partial x}(x, y) &= u(x, y) \\
\frac{\partial I}{\partial y}(x, y) &= v(x, y)
\end{align*}
\]

such that

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

Only if:

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

\[
u(x, y): \mathbb{R}^2 \rightarrow \mathbb{R}
\]
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
Key idea

When blending, retain the gradient information as best as possible

source | destination | copy-paste | Poisson blending
Definitions and notation

Notation

\( g \): source function

\( S \): destination

\( \Omega \): destination domain

\( f \): interpolant function

\( f^* \): destination function

Which one is the unknown?
Definitions and notation

**Notation**

\( g \): source function

\( S \): destination

\( \Omega \): 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
\( \Omega \): destination domain
\( f \): interpolant function
\( f^* \): destination function

Find \( f \) such that:
- \( \nabla f = \nabla g \) inside \( \Omega \).
- \( f = f^* \) at the boundary \( \partial \Omega \).

Poisson blending: integrate vector field \( \nabla g \) with Dirichlet boundary conditions \( f^* \).
Least-squares integration and the Poisson problem
“Variational” means optimization where the unknown is an entire function.

Variational problem

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

Recall ...

Nabla operator definition

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

is this known?

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

$f$ is equivalent to $f^*$ at the boundaries

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

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

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

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

**Poisson equation (with Dirichlet boundary conditions)**

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

What does this term do?

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:

\[ \mathbf{v} = (u, v) \]

This can be derived using the *Euler-Lagrange equation*. 
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 \(\mathbf{v}\)

Recall ...

**Laplacian** \(\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\)

**Divergence** \(\text{div } \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\)

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

Input vector field:

\[
\mathbf{v} = (u, v)
\]
In the Poisson blending example...

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

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

Find \( f \) such that:
- \( \nabla f = \nabla g \) inside \( \Omega \).
- \( f = f^* \) at the boundary \( \partial \Omega \).

What does the input vector field equal in Poisson blending?

\[
\mathbf{v} = (u, v) = \]

\( g \)
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 \( \Omega \).
- \( f = f^* \) 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)

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

Find \( f \) such that:
- \( \nabla f = \nabla g \) inside \( \Omega \).
- \( f = f^* \) at the boundary \( \partial \Omega \).

so make these ...

\[ \Delta g \quad \Delta f \]

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)

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

Recall ...

Laplacian filter

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0 \\
\end{array}
\]

So for each pixel, do:

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

Or for discrete images:

partial-x derivative filter

\[
\begin{array}{c}
1 \\
-1 \\
\end{array}
\]

partial-y derivative filter

\[
\begin{array}{c}
1 \\
-1 \\
\end{array}
\]
Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

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

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)

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

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

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

Or for discrete images (more compact notation):

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

Recall ...

Laplacian filter

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Partial-x derivative filter

| 1 | -1 |

Partial-y derivative filter

| 1 | -1 |
We can rewrite this as

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

In vector form:

\[
\begin{bmatrix}
0 & \cdots & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \cdots & 0
\end{bmatrix} \cdot
\begin{bmatrix}
f_1 \\
f_q_1 \\
\vdots \\
f_q_2 \\
f_p \\
f_q_3 \\
\vdots \\
f_q_4 \\
\vdots \\
f_P
\end{bmatrix} =
\begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
(\nabla \cdot \mathbf{v})_{q_1} \\
\vdots \\
(\nabla \cdot \mathbf{v})_{q_2} \\
(\nabla \cdot \mathbf{v})_p \\
(\nabla \cdot \mathbf{v})_{q_3} \\
\vdots \\
(\nabla \cdot \mathbf{v})_{q_4} \\
\vdots \\
(\nabla \cdot \mathbf{v})_P
\end{bmatrix}
\]

\(A\) \(f\) \(b\)
We can rewrite this as

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

one for each pixel \( p = 1, \ldots, P \)

In vector form:

\[
\begin{bmatrix}
0 & \cdots & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
f_1 \\
\vdots \\
f_{q_1} \\
f_p \\
\vdots \\
f_{q_4} \\
f_{P}
\end{bmatrix}
=
\begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
\vdots \\
(\nabla \cdot \mathbf{v})_{q_1} \\
\vdots \\
(\nabla \cdot \mathbf{v})_p \\
\vdots \\
(\nabla \cdot \mathbf{v})_{q_4} \\
\vdots \\
(\nabla \cdot \mathbf{v})_{P}
\end{bmatrix}
\]

what is this?

A

what are the sizes of these?

\[ A \]

\[ f \]

\[ b \]
We can rewrite this as

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

one for each pixel $p = 1, \ldots, P$

In vector form:

$$A f = b$$

where

\[
\begin{bmatrix}
0 & \cdots & 1 & \cdots & 1 & -4 & 1 & \cdots & 1 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
f_1 \\
f_{q_1} \\
f_{q_2} \\
f_p \\
f_{q_3} \\
f_{q_4} \\
f_2 \\
f_P
\end{bmatrix}
= \begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
(\nabla \cdot \mathbf{v})_{q_1} \\
(\nabla \cdot \mathbf{v})_{q_2} \\
(\nabla \cdot \mathbf{v})_p \\
(\nabla \cdot \mathbf{v})_{q_3} \\
(\nabla \cdot \mathbf{v})_{q_4} \\
(\nabla \cdot \mathbf{v})_2 \\
(\nabla \cdot \mathbf{v})_P
\end{bmatrix}
\]

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:

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

This requires ordering pixels in column-major order.

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

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

Poisson equation (with Dirichlet boundary conditions)

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

After discretization, equivalent to:

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

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:

\[ E_{\text{LLS}} = \| A f - b \|^2 \]

Expand the error:

\[ E_{\text{LLS}} = f^T (A^T A) f - 2 f^T (A^T b) + \| b \|^2 \]

Minimize the error:

Set derivative to 0

\[ (A^T A) f = A^T b \]

Solve for \( x \)

\[ f = (A^T A)^{-1} A^T b \]

In Matlab:

\[ f = A \backslash b \]

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

Poisson equation (with Dirichlet boundary conditions)

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

After discretization, equivalent to:

\[
\begin{bmatrix}
D & I & 0 & 0 & 0 & \ldots & 0 \\
I & D & I & 0 & 0 & \ldots & 0 \\
0 & I & D & I & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \ldots & 0 & I & D & I & 0 \\
0 & \ldots & \ldots & 0 & I & D & I \\
0 & \ldots & \ldots & \ldots & 0 & I & D
\end{bmatrix}
\begin{bmatrix}
f_1 \\ f_{d_1} \\ \vdots \\ f_{d_q} \\ f_p \\ \vdots \\ f_{d_q} \\ f_p \\ \vdots \\ f_{d_q} \\ f_p \\ \vdots \\ f_{d_q} \\ f_p
\end{bmatrix} =
\begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\ (\nabla \cdot \mathbf{v})_{d_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{d_q} \\ (\nabla \cdot \mathbf{v})_p \\ (\nabla \cdot \mathbf{v})_{d_1} \\ \vdots \\ (\nabla \cdot \mathbf{v})_{d_q} \\ (\nabla \cdot \mathbf{v})_p
\end{bmatrix}
\]

Linear system of equations:

\[ A \mathbf{f} = \mathbf{b} \]

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:

\[
\begin{bmatrix}
D & I & 0 & 0 & 0 & \ldots & 0 \\
I & D & I & 0 & 0 & \ldots & 0 \\
0 & I & D & I & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \ldots & 0 & I & D & I & 0 \\
0 & \ldots & \ldots & 0 & I & D & I \\
0 & \ldots & \ldots & \ldots & 0 & I & D \\
\end{bmatrix}
\begin{bmatrix}
f_1 \\
\vdots \\
f_{d_1} \\
\vdots \\
f_{d_2} \\
\vdots \\
f_p \\
\vdots \\
f_{q_3} \\
\vdots \\
f_{q_4} \\
\vdots \\
f_P \\
\end{bmatrix}
= \begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
\vdots \\
(\nabla \cdot \mathbf{v})_{d_1} \\
\vdots \\
(\nabla \cdot \mathbf{v})_{d_2} \\
\vdots \\
(\nabla \cdot \mathbf{v})_p \\
\vdots \\
(\nabla \cdot \mathbf{v})_{q_3} \\
\vdots \\
(\nabla \cdot \mathbf{v})_{q_4} \\
\vdots \\
(\nabla \cdot \mathbf{v})_P \\
\end{bmatrix}
\]

Linear system of equations:

\[
Af = b
\]

Matrix is \( P \times P \to \text{billions of entries} \)

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

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

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

• Alternative solvers: projection procedures.
  We will discuss one of these when we cover photometric stereo.
A more efficient Poisson solver
Let’s look again at our optimization problem

Variational problem

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

\[\text{gradient of } f \text{ looks like vector field } \mathbf{v} \quad \text{f is equivalent to } f^* \text{ at the boundaries}\]

Recall ...

\[\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x}, & \frac{\partial f}{\partial y} \end{bmatrix}\]

Input vector field:

\[\mathbf{v} = (u, v)\]
Let’s look again at our optimization problem

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

\[\text{gradient of } f \text{ looks like vector field } \mathbf{v}\]

\[\text{f is equivalent to } f^* \text{ at the boundaries}\]

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**

\[
\begin{bmatrix}
1 \\
-1
\end{bmatrix}
\]

**partial-y derivative filter**

\[
\begin{bmatrix}
1 \\
-1
\end{bmatrix}
\]
Let’s look again at our optimization problem

We can use the gradient approximation to discretize the variational problem

\[
\min_f \|Gf - v\|^2
\]

What are G, f, and v?

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:

\[
\begin{array}{c}
\text{partial-x derivative filter} \\
\begin{pmatrix} 1 & -1 \end{pmatrix}
\end{array}
\]

\[
\begin{array}{c}
\text{partial-y derivative filter} \\
\begin{pmatrix} 1 \\ -1 \end{pmatrix}
\end{array}
\]
Let’s look again at our optimization problem

We can use the gradient approximation to discretize the variational problem

\[ \min_f \| Gf - v \|^2 \]

matrix \( G \) formed by stacking together discrete gradients

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

\[ \begin{bmatrix} 1 & -1 \end{bmatrix} \]

partial-\( y \) derivative filter

\[ \begin{bmatrix} 1 \\ -1 \end{bmatrix} \]
Let’s look again at our optimization problem

We can use the gradient approximation to discretize the variational problem

\[
\min_{f} \| Gf - v \|^2
\]

Recall ...

Image gradient

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

And for discrete images:

partial-x derivative filter

\[
\begin{bmatrix} 1 & -1 \end{bmatrix}
\]

partial-y derivative filter

\[
\begin{bmatrix} 1 \\ -1 \end{bmatrix}
\]
Approach 1: Compute stationary points

Given the loss function:

\[ E(f) = \|Gf - \nu\|^2 \]

... we compute its derivative:

\[ \frac{\partial E}{\partial f} = ? \]
Approach 1: Compute stationary points

Given the loss function:

\[ E(f) = \|Gf - \nu\|^2 \]

... we compute its derivative:

\[ \frac{\partial E}{\partial f} = G^T G f - G^T \nu \]

... and we do what with it?
Approach 1: Compute stationary points

Given the loss function:

\[ E(f) = \|Gf - \nu\|^2 \]

... we compute its derivative:

\[ \frac{\partial E}{\partial f} = G^T G f - G^T \nu \]

... and we set that to zero:

\[ \frac{\partial E}{\partial f} = 0 \implies G^T G f = G^T \nu \]

What is this vector?

What is this matrix?
Approach 1: Compute stationary points

Given the loss function:

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

... we compute its derivative:

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

... and we set that to zero:

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

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)

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

After discretization, equivalent to:

\[
\begin{bmatrix}
D & I & 0 & 0 & 0 & \cdots & 0 \\
I & D & I & 0 & 0 & \cdots & 0 \\
0 & I & D & I & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & I & D & I & 0 \\
0 & \cdots & \cdots & 0 & I & D & I \\
0 & \cdots & \cdots & \cdots & 0 & I & D \\
\end{bmatrix}
\begin{bmatrix}
f_1 \\
f_{d_1} \\
f_{d_2} \\
f_p \\
f_{d_3} \\
f_{d_4} \\
f_p \\
\end{bmatrix}
= \begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
(\nabla \cdot \mathbf{v})_{d_1} \\
(\nabla \cdot \mathbf{v})_{d_2} \\
(\nabla \cdot \mathbf{v})_p \\
(\nabla \cdot \mathbf{v})_{d_3} \\
(\nabla \cdot \mathbf{v})_{d_4} \\
(\nabla \cdot \mathbf{v})_p \\
\end{bmatrix}
\]

Linear system of equations:

\[ Af = b \]

Same system as:

\[ G^T G f = G^T \mathbf{v} \]

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:

\[ E(f) = \|Gf - \nu\|^2 \]

... we compute its derivative:

\[ \frac{\partial E}{\partial f} = G^T Gf - G^T \nu \]

... and we set that to zero:

\[ \frac{\partial E}{\partial f} = 0 \Rightarrow G^T Gf = G^T \nu \]

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:

\[ \frac{\partial E}{\partial f} = G^T G f - G^T v = A f - b \equiv -r \]

We call this term the residual.
Approach 2: Use gradient descent

Given the loss function:

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

... we compute its derivative:

\[ \frac{\partial E}{\partial f} = G^T G f - G^T v = A f - b \equiv -r \]

... and then we iteratively compute a solution:

\[ f^{i+1} = f^i + \eta^i r^i \]

for \( i = 0, 1, ..., N \), where \( \eta^i \) are positive step sizes.
Selecting optimal step sizes

Make derivative of loss function \textit{with respect to} \( \eta^i \) equal to zero:

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

\[
E(f^{i+1}) = \| G(f^i + \eta^i r^i) - v \|^2
\]
Selecting optimal step sizes

Make derivative of loss function \textit{with respect to} \( \eta^i \) equal to zero:

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

\[
E(f^{i+1}) = \| G(f^i + \eta^i r^i) - v \|^2
\]

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

Given the loss function:

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

Minimize by iteratively computing:

$$r_i = b - Af_i, \quad \eta_i = \frac{(r_i)^T r_i}{(r_i)^T A r_i}, \quad f^{i+1} = f^i + \eta_i r_i, \quad i = 0, \ldots, N$$

Is this cheaper than the pseudo-inverse approach?
Gradient descent

Given the loss function:

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

Minimize by iteratively computing:

\[ r^i = b - Af^i, \quad \eta^i = \frac{(r^i)^T r^i}{(r^i)^T Ar^i}, \quad f^{i+1} = f^i + \eta^i r^i, \quad i = 0, ..., N \]

Is this cheaper than the pseudo-inverse approach?

• We never need to compute \( A \), only its products with vectors \( f, r \).
Gradient descent

Given the loss function:

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

Minimize by iteratively computing:

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

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:

\[ E(f) = \|G f - v\|^2 \]

Minimize by iteratively computing:

\[ r^i = b - A f^i, \quad \eta^i = \frac{(r^i)_T r^i}{(r^i)_T A r^i}, \quad f^{i+1} = f^i + \eta^i r^i, \quad i = 0, \ldots, N \]

Is this cheaper than the pseudo-inverse approach?

- 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:

\[ d^{i+1} = r^i + \beta^i d^i, \quad \eta^i = \frac{(r^i)^T r^i}{(d^i)^T A d^i}, \quad f^{i+1} = f^i + \eta^i d^i, \quad i = 0, \ldots, N \]

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

Does the initialization $f^0$ 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 $f^0$ matter?

- It doesn’t matter in terms of what final $f$ we converge to, because the loss function is convex.

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

- 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 $P \times P$.
- 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)

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

After discretization, equivalent to:

$$\begin{bmatrix}
D & I & 0 & 0 & 0 & \cdots & 0 \\
I & D & I & 0 & 0 & \cdots & 0 \\
0 & I & D & I & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & I & D & I & 0 \\
0 & \cdots & \cdots & 0 & I & D & I \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
\end{bmatrix}
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
f_5 \\
f_6 \\
f_7 \\
f_8 \\
f_9 \\
f_{10} \\
f_{11} \\
f_{12} \\
f_{13} \\
f_{14} \\
f_{15} \\
f_{16} \\
f_{17} \\
f_{18} \\
f_{19} \\
f_{20} \\
\end{bmatrix}
= \begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
(\nabla \cdot \mathbf{v})_2 \\
(\nabla \cdot \mathbf{v})_3 \\
(\nabla \cdot \mathbf{v})_4 \\
(\nabla \cdot \mathbf{v})_5 \\
(\nabla \cdot \mathbf{v})_6 \\
(\nabla \cdot \mathbf{v})_7 \\
(\nabla \cdot \mathbf{v})_8 \\
(\nabla \cdot \mathbf{v})_9 \\
(\nabla \cdot \mathbf{v})_{10} \\
(\nabla \cdot \mathbf{v})_{11} \\
(\nabla \cdot \mathbf{v})_{12} \\
(\nabla \cdot \mathbf{v})_{13} \\
(\nabla \cdot \mathbf{v})_{14} \\
(\nabla \cdot \mathbf{v})_{15} \\
(\nabla \cdot \mathbf{v})_{16} \\
(\nabla \cdot \mathbf{v})_{17} \\
(\nabla \cdot \mathbf{v})_{18} \\
(\nabla \cdot \mathbf{v})_{19} \\
(\nabla \cdot \mathbf{v})_{20}
\end{bmatrix}$$

Linear system of equations:

$$A f = b$$

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 \textit{preconditioner} $P$ should we use?
We are solving this linear system:

$$Af = b$$

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

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

When is it preferable to solve this alternative linear system?

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

What *preconditioner* $P$ should we use?

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

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

We are solving this linear system:

\[ Af = b \]

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

\[ P^{-1}Af = P^{-1}b \]

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)

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

After discretization, equivalent to:

Linear system of equations:

$$Af = b$$

Matrix is $P \times P \rightarrow \text{billions of entries}$

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 $S-\Omega$ and $\partial \Omega$) and 1 otherwise.

- Use convolution to perform Laplacian filtering over the entire image.

- Use (conjugate) gradient descent rules to only update pixels for which the mask is 1. Equivalently, change the update rules to:

  \[ f^{i+1} = f^i + B\eta^i r^i \] (gradient descent)

  \[ f^{i+1} = f^i + B\eta^i d^i \] (conjugate gradient descent)
Note: handling (Dirichlet) boundary conditions

- Form a mask $B$ that is 0 for pixels that should not be updated (pixels on $S-\Omega$ 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:

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

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

In practice, masking is also required at other steps of (conjugate) gradient descent, to deal with invalid boundaries (e.g., from convolutions). See homework assignment 3.
Poisson image editing examples
Photoshop’s “healing brush”

- Uses higher-order derivatives

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
Blending objects with holes
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\textsuperscript{1} C. Lawrence Zitnick\textsuperscript{2} Michael Cohen\textsuperscript{1,2} Brian Curless\textsuperscript{1}
\textsuperscript{1}University of Washington \textsuperscript{2}Microsoft Research
Flash/no-flash photography
Flash

+ Low Noise
+ Sharp
- Artificial Light
- Jarring Look

No-Flash

- High Noise
- Lacks Detail
+ Ambient Light
+ Natural Look
No-Flash
Denoising Result
Key idea

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

Can we do similar flash/no-flash fusion tasks with gradient-domain processing?
Removing self-reflections and hot-spots
Removing self-reflections and hot-spots
Removing self-reflections and hot-spots
Idea: look at how gradients are affected

Same gradient vector direction

Flash Gradient Vector

Ambient Gradient Vector

Ambient

Flash

No reflections
Idea: look at how gradients are affected

Different gradient vector direction

Reflection Ambient Gradient Vector

Flash Gradient Vector

With reflections
Gradient projections

Ambient  Flash  Result  Residual
Flash/no-flash with gradient-domain processing
Gradient-domain rendering
Primal domain

Gradient domain
gradients of natural images are sparse (close to zero in most places)
Can I go from one image to the other?
Can I go from one image to the other?

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

integration (e.g., Poisson solver)
Rendering

Primal-domain rendering: simulate intensities directly

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

Why would gradient-domain rendering make sense?
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

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

Gradients of natural images are sparse (close to zero in most places)
Gradient cameras
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?

Gradients of natural images are sparse (close to zero in most places)
One of my favorite papers

Why I want a Gradient Camera

Jack Tumblin  
Northwestern University  
jet@cs.northwestern.edu

Amit Agrawal  
University of Maryland  
aagrawal@umd.edu

Ramesh Raskar  
MERL  
raskar@merl.com

Why would you want a gradient camera?
• Much faster frame rate, as you only read out very few pixels (where gradient is significant).
• Much higher dynamic range, if also combined with logarithmic gradients.

Can you directly display the measurements of such a camera?

How would you build a gradient camera?
One of my favorite papers

Why I want a Gradient Camera

Jack Tumblin
Northwestern University
jet@cs.northwestern.edu

Amit Agrawal
University of Maryland
aagrawal@umd.edu

Ramesh Raskar
MERL
raskar@merl.com

Why would you want a gradient camera?
• Much faster frame rate, as you only read out very few pixels (where gradient is significant).
• Much higher dynamic range, if also combined with logarithmic gradients.

Can you directly display the measurements of such a camera?
• You need to use a Poisson solver to reconstruct the image from the measured gradients.

How would you build a gradient camera?
Change the sensor

Can you think how?
Change the sensor

operational amplifier (amplify difference of inputs)

firing mechanism

what is this for?

typical analog front-end

discrete signal

analog voltage

analog voltage

+ -
Change the sensor

Any disadvantages of this sensor?

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

What about Poisson noise?

operational amplifier (amplify difference of inputs)

firing mechanism

analog voltage

typical analog front-end

discrete signal

discrete signal

Change the sensor

- photodiode
- microlens
- potential well

Analog voltage

Typical analog front-end

Discrete signal

155
Any disadvantages of this sensor?

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

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

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

What about Poisson noise?

- Poisson noise is the same in both cases.
Noise considerations

\[ L_1 \]

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

\[ L_2 \]

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

\[
\sigma(I_1 - I_2)^2 = ?
\]

Digital subtraction in post-processing

\[
L_1 \approx \text{Poisson}(t \cdot (a \cdot \Phi_1 + D))
\]

\[
L_2 \approx \text{Poisson}(t \cdot (a \cdot \Phi_2 + D))
\]

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

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

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

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

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

\[
\sigma(I)^2 = ?
\]

Analog subtraction on sensor

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

\[
I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}}
\]
Noise considerations

Digital subtraction in post-processing

Analog subtraction on sensor

\[ I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]
\[ I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]

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

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

Digital subtraction in post-processing

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

which variance is better?

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

Digital subtraction in post-processing

Analog subtraction on sensor

\[ I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]
\[ I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]

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

Terms related to Poisson noise are the same, additive noise is reduced if opamp is well-designed.

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

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

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

Digital subtraction in post-processing

\[ I_1 = L_1 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]
\[ I_2 = L_2 \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]

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

what is the distribution of the difference \( L_1 - L_2 \)?

Analog subtraction on sensor

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

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

\[ L_1 \]

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

\[ L_2 \]

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

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

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

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

Digital subtraction in post-processing

Analog subtraction on sensor

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

\[ I = (L_1 - L_2) \cdot g + n_{\text{opamp}} \cdot g + n_{\text{read}} \cdot g + n_{\text{ADC}} \]
Change the optics

Can you think how?
Change the optics

Optical filtering

- lenslet
- refractive slab
- template (edge filter)
- photodetectors

Resulting image

Angle-sensitive pixels

- Grating pitch, d
- Grating separation, z
- Diffraction grating
- Analyzer grating
- n-well
- Photodiode

Physical Layout

Impulse Response (2D)
Change the optics

Optical filtering

- lenslet
- refractive slab
- template (edge filter)

Angle-sensitive pixels

- Grating pitch, d
- Grating separation, z
- Diffraction grating
- Analyzer grating
- Photodiode

Any disadvantages?

resulting image

Impulse Response (2D)
Change the optics

Optical filtering

- lenslet
- template (edge filter)
- photodetectors

refractive slab

resulting image

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

Angle-sensitive pixels

- Grating pitch, \( d \)
- Grating separation, \( z \)

Impulse Response (2D)
One of my favorite papers

Why I want a Gradient Camera

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack Tumblin</td>
<td>Northwestern University</td>
<td><a href="mailto:jet@cs.northwestern.edu">jet@cs.northwestern.edu</a></td>
</tr>
<tr>
<td>Amit Agrawal</td>
<td>University of Maryland</td>
<td><a href="mailto:aagrawal@umd.edu">aagrawal@umd.edu</a></td>
</tr>
<tr>
<td>Ramesh Raskar</td>
<td>MERL</td>
<td><a href="mailto:raskar@merl.com">raskar@merl.com</a></td>
</tr>
</tbody>
</table>

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
Basic reading:
- Szeliski textbook, Sections 3.13, 3.5.5, 9.3.4, 10.4.3.

Additional reading:
- Hua et al., “Light transport simulation in the gradient domain,” SIGGRAPH Asia 2018 course.

References