Gradient-domain image processing
Course announcements

• Homework assignment 3 is out.
  - Due October 19th.
  - Generous bonus components.
  - Homework assignment submissions have now moved to Gradescope.

• Should we have office hours on Friday?
  - Vote on Piazza!

• Keep an eye out for final project announcement tonight.
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.
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

(a) Input image  (b) Saliency-sharpening filter  (c) Pseudo-relighting filter  (d) Non-photorealistic rendering filter

(e) Compressed input-image  (f) De-blocking filter  (g) User input for colorization  (h) Colorization filter
Main pipeline

Original Images → Estimation of Gradients → Manipulation of Gradients → Edited Gradient Fields → Integration of Gradient Fields → Edited Images
Basics of gradients and fields
Some vector calculus definitions in 2D

Scalar field: a function assigning a \textit{scalar} to every point in space.

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

Vector field: a function assigning a \textit{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.

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

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.

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

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.

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

Curl: cross product of nabla with a vector field.

\[ \nabla \times \begin{bmatrix} u(x, y) & v(x, y) \end{bmatrix} = \left( \frac{\partial v}{\partial x}(x, y) - \frac{\partial u}{\partial y}(x, y) \right) \hat{k} \]
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 \( \nabla \) with itself!
Simplified notation

Nabla (or del): vector differential operator.

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

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

\[ \nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \]

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.

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

Think of this as a 2D vector.

This is a vector field.

This is a scalar field.
Simplified notation

Curl of the gradient:

\[ \nabla \times \nabla I = I_{yx} - I_{xy} \]

Divergence of the gradient:

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

Laplacian: scalar differential operator.

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

Inner product of del with itself!
Image representation

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

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

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

scalar field $I(x, y): \mathbb{R}^2 \to \mathbb{R}$ $\rightarrow$ vector field $\nabla I(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) \\ \frac{\partial I}{\partial y}(x, y) \end{bmatrix}$
Image gradients

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

Scalar field \( I(x, y): \mathbb{R}^2 \to \mathbb{R} \) \( \xrightarrow{\text{ } \to \text{ }} \) Vector field \( \nabla I(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) \\ \frac{\partial I}{\partial y}(x, y) \end{bmatrix} \)

- 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 \( h = 1 \).

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

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

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:

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

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:

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

Very important: make sure to use consistent derivative and Laplacian filters!
Image gradients

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

\[ I(x, y) : \mathbb{R}^2 \rightarrow \mathbb{R} \quad \Rightarrow \quad \nabla I(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) \\ \frac{\partial I}{\partial y}(x, y) \end{bmatrix} \]

- 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 \(I\)?

\[
\begin{align*}
I(x, y) : \mathbb{R}^2 &\rightarrow \mathbb{R} \\
\frac{\partial I}{\partial x}(x, y) &\equiv u(x, y) \\
\frac{\partial I}{\partial y}(x, y) &\equiv v(x, y)
\end{align*}
\]
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

\[ \nabla \times \nabla I = \nabla \times (I_x \times I_y) = \Delta I \]

image \( I \)  \( \rightarrow \)  \( I_x \)  \( \rightarrow \)  \( I_y \)  \( \rightarrow \)  \( \Delta I \)  \( \nabla \times \nabla I \)  \( \rightarrow \)  \( I_{xy} \)  \( \rightarrow \)  \( I_{yx} \)
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 \(I\)?

\[
\begin{align*}
I(x, y) & : \mathbb{R}^2 \to \mathbb{R} \\
\frac{\partial I}{\partial x}(x, y) & = u(x, y) \\
\frac{\partial I}{\partial y}(x, y) & = v(x, y)
\end{align*}
\]

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:
Vector field integration

Two fundamental questions:

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

- How can integration of a vector field be performed?
Different types of integration problems

- Reconstructing height fields from gradients
  Applications: shape from shading, photometric stereo

- Manipulating image gradients
  Applications: tonemapping, image editing, matting, fusion, mosaics

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

Key challenge: Most vector fields in applications are not integrable.
- Integration must be done approximately.
A prototypical integration problem: Poisson blending
Application: Poisson blending

originals  copy-paste  Poisson blending
Key idea

When blending, retain the gradient information as best as possible.

source  destination  copy-paste  Poisson blending
Definitions and notation

**Notation**

- $g$: source function
- $S$: destination
- $\Omega$: 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 \( f^* \)?
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 $f^*$. 

Definitions and notation

**Notation**

$g$: source function

$S$: destination

$\Omega$: destination domain

$f$: interpolant function

$f^*$: destination function
Least-squares integration and the Poisson problem
“Variational” means optimization where the unknown is an entire function.

Recall ...

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?

Nabla operator definition

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

is this known?

\[
\mathbf{v} = (u, v)
\]
“Variational” means optimization where the unknown is an entire function.

**Variational problem**

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

gradient of \( f \) looks like vector field \( \mathbf{v} \)

\( f \) is equivalent to \( f^* \)
at the boundaries

Recall ...

**Nabla operator definition**

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

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 solution of 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:

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

The stationary point of the variational loss is the solution to the solution of 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 \( \text{div } \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \)

Input vector field:

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

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

Poisson equation (with Dirichlet boundary conditions)

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

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?

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

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

Poisson equation (with Dirichlet boundary conditions)

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

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

So make these ...

\[ \Delta g \quad \Delta f \quad \text{equal} \]

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 solution of 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)

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

Recall ...

**Laplacian filter**

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

So for each pixel, do:

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

Or for discrete images:

**partial-x derivative filter**

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

**partial-y derivative filter**

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

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

\[
\begin{align*}
-4f(x, y) &+ f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) \\
&= u(x + 1, y) - u(x, y) + v(x, y + 1) - v(x, y)
\end{align*}
\]

Or for discrete images:

\[(\Delta f)(x, y) = (\nabla \cdot \mathbf{v})(x, y)\]
Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

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

Recall ...

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

- Partial-x derivative filter
  
  \[
  \begin{array}{c}
  1 \\
  -1 \\
  \end{array}
  \]

- Partial-y derivative filter
  
  \[
  \begin{array}{c}
  1 \\
  -1 \\
  \end{array}
  \]

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

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

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

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

In vector form:

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

\(A\) \(f\) \(b\)
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:

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

What is this? $A$

What are the sizes of these? $f$ and $b$
We can rewrite this as

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

In vector form:

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

We call this the \textit{Laplacian matrix}.
Poisson equation (with Dirichlet boundary conditions)

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

After discretization, equivalent to:

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

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:

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

Expand the error:

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

Minimize the error:

Set derivative to 0

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

Solve for \( f \)

\[ f = (A^\top A)^{-1}A^\top b \]
Solving the linear system

Convert the system to a linear least-squares problem:

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

Expand the error:

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

Minimize the error:

Set derivative to 0

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

Solve for \( x \)

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

In Matlab:

\[ f = A \backslash b \]

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

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

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

After discretization, equivalent to:

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

**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)
Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

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

After discretization, equivalent to:

\[
\begin{bmatrix}
0 & \cdots & 1 & 0 & 1 & -1 & \cdots & 0
\end{bmatrix} \cdot \begin{bmatrix}
f_1 \\
f_{q_1} \\
f_{q_2} \\
f_{q_3} \\
f_{p} \\
f_{q_4} \\
f_{p}
\end{bmatrix} = \begin{bmatrix}
(\nabla \cdot \mathbf{v})_1 \\
(\nabla \cdot \mathbf{v})_{q_1} \\
(\nabla \cdot \mathbf{v})_{q_2} \\
(\nabla \cdot \mathbf{v})_{q_3} \\
(\nabla \cdot \mathbf{v})_{p} \\
\vdots \\
\vdots \\
(\nabla \cdot \mathbf{v})_{p}
\end{bmatrix}
\]

Matrix is \( P \times P \rightarrow \) billions of entries

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)
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.
  + Quadtree decompositions.

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

Variational problem

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

- gradient of $f$ looks like vector field $\mathbf{v}$
- $f$ is equivalent to $f^*$ at the boundaries

Recall ...

Nabla operator definition

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

Input vector field:

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

**Variational problem**

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

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

- **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:

- 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.

Discrete 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.

Discrete problem

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

Recall ...

**Image gradient**

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

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:

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

... 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:

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

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

Given the loss function:

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

... we compute its derivative:

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

... and we set that to zero:

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

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:

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

... and we set that to zero:

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

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:

\[
\begin{bmatrix}
0 & \cdots & 1 & \cdots & 1 & \cdots & -4 & \cdots & 1 & \cdots & -1 & \cdots & 0
\end{bmatrix}.
\]

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:

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

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

Given the loss function:

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

… we compute its derivative:

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

We call this term the residual

... and then we iteratively compute a solution:

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

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

Make derivative of loss function 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:

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

for $i = 0, 1, ..., N$

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

Given the loss function:

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

Minimize by iteratively computing:

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

for i = 0, 1, ..., N

Is this cheaper than the pseudo-inverse approach?

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

Given the loss function:

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

Minimize by iteratively computing:

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

for i = 0, 1, ..., N

Is this cheaper than the pseudo-inverse approach?

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

Given the loss function:

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

Minimize by iteratively computing:

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

Is this cheaper than the pseudo-inverse approach?

- We never need to compute \( A \), only its products with vectors \( f, r \).
- Vectors \( f, r \) are images.
- Because \( A \) is the Laplacian matrix, these matrix-vector products can be efficiently computed using convolutions with the Laplacian kernel.
In practice: conjugate gradient descent

Given the loss function:

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

Minimize by iteratively computing:

\[ f^{i+1} = f^i + \eta_i d^i, \quad r^i = b - Af^i, \quad \text{for } i = 0, 1, \ldots, N \]

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

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

- Smarter way for selecting update directions
- Everything can still be done using convolutions
In practice: conjugate gradient descent

Given the loss function:

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

Minimize by iteratively computing:

\[ f^{i+1} = f^i + \eta_i d^i, \quad r^i = b - Af^i, \quad \text{for } i = 0, 1, ..., N \]

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

\[ \beta^{i+1} = \frac{(r^{i+1})^T r^{i+1}}{(r^i)^T r^i}, \quad \eta^i = \frac{(d^i)^T r^i}{(d^i)^T Ad^i} \]

- Smarter way for selecting update directions
- Everything can still be done using convolutions
- See assignment 3 for version with only one convolution.
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)**

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

After discretization, equivalent to:

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

**Linear system of equations:**

\[ Af = b \]

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 \textit{numerically stable}.

What \textit{preconditioner} \( P \) should we use?
Note: preconditioning

We are solving this linear system:

\[ Af = b \]

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

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

When is it preferable to solve this alternative linear system?

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

What preconditioner \( P \) should we use?

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

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

We are solving this linear system:

\[ Af = b \]

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

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

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.
Discretization of the Poisson equation

Poisson equation (with Dirichlet boundary conditions)

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

After discretization, equivalent to:

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

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

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)
Note: handling (Dirichlet) boundary conditions

• Form a mask $M$ that is 0 for pixels that should not be updated (pixels on $S\cdot\Omega$ 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:

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

$$f^{i+1} = f^i + M\eta^i d^i$$  \hspace{1cm} \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
Blending objects with holes

(a) color-based cutout and paste
(b) seamless cloning
(c) seamless cloning and destination averaged
(d) mixed seamless cloning
Editing
Concealment

How would you do this with Poisson blending?
Concealment

How would you do this with Poisson blending?
- Insert a copy of the background.
Texture swapping
Special case: membrane interpolation

How would you do this?
Special case: membrane interpolation

How would you do this?

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

Laplacian problem
\[
\min_f \iint\limits_\Omega |\nabla f|^2 \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}
\]
Entire suite of image editing tools

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

Pravin Bhat$^1$ C. Lawrence Zitnick$^2$ Michael Cohen$^{1,2}$ Brian Curless$^1$
$^1$University of Washington $^2$Microsoft Research

(a) Input image  (b) Saliency-sharpening filter  (c) Pseudo-relighting filter  (d) Non-photorealistic rendering filter

(e) Compressed input-image  (f) De-blocking filter  (g) User input for colorization  (h) Colorization filter
Flash/no-flash photography
**Flash**

+ Low Noise
+ Sharp
- Artificial Light
- Jarring Look

**No-Flash**

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

Ambient

Flash

Face

Hands

Tripod
Removing self-reflections and hot-spots

Ambient

Flash

Result

Reflection Layer
Idea: look at how gradients are affected

Same gradient vector direction

Flash Gradient Vector

Ambient Gradient Vector

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

Residual Gradient Vector
Flash Gradient Vector
Result Gradient Vector
Flash/no-flash with gradient-domain processing

Flash

Ambient

Intensity Gradient

Vector Projection

Result X

Result Y

2D Integration

Result
Gradient-domain rendering
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.
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.

Why not all?

Primal-domain rendering: simulate intensities directly

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

Gradient-Domain Metropolis Light Transport

Jaakko Lehtinen\textsuperscript{1,2}, Tero Karras\textsuperscript{3}, Samuli Laine\textsuperscript{1}, Miika Aittala\textsuperscript{2,1}, Frédéric Durand\textsuperscript{3}, Timo Aila\textsuperscript{1}

\textsuperscript{1}NVIDIA Research \hspace{1cm} \textsuperscript{2}Aalto University \hspace{1cm} \textsuperscript{3}MIT CSAIL

Figure 1: We compute image gradients $I_x$, $I_y$ and a coarse image $I^0$ 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\textsuperscript{1}, Marco Manzi\textsuperscript{2}, Miika Aittala\textsuperscript{1}, Jaakko Lehtinen\textsuperscript{1,3}, Frédéric Durand\textsuperscript{3}, Matthias Zwicker\textsuperscript{2}

\textsuperscript{1}Aalto University \hspace{1cm} \textsuperscript{2}University of Bern \hspace{1cm} \textsuperscript{3}NVIDIA \hspace{1cm} \textsuperscript{4}MIT CSAIL

Figure 1: Comparing gradient-domain path tracing (G-PT, $L_3$ 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.
Does it help?
Gradient-domain path tracing (2 minutes)
Remember this idea (we’ll come back to it)

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

- analog voltage
- analog voltage
- discrete signal
- discrete signal
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 reduced by 2x. Why is this better than computing gradients in post-processing?
• When using the sensor, additive noise is added directly to the analog gradient.
• When subtracting two images in post-processing, we double additive noise.
What about Poisson noise?
• Poisson noise is doubled in both cases.
Change the optics

Can you think how?
Change the optics

Optical filtering

Angle-sensitive pixels

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

Physical Layout

Impulse Response (2D)

resulting image
Change the optics

Optical filtering

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

Resulting image

Angle-sensitive pixels

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

Impulse Response (2D)

Any disadvantages?
Change the optics

Any disadvantages?
• Reduced light efficiency (we block light).
• We can’t do subtraction very easily in optics.
One of my favorite papers

Why I want a Gradient Camera

Jack Tumblin
Northwestern University
ejt@cs.northwestern.edu

Amit Agrawal
University of Maryland
aagrawal@umd.edu

Ramesh Raskar
MERL
raskar@merl.com

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

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

How would you build a gradient camera?
• Change the sensor.
• Change the optics.
We can also compute *temporal* gradients

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

Concept figure for event-based camera: 
https://www.youtube.com/watch?v=kPCZESVfHoQ
High-speed output on a quadcopter: 
https://www.youtube.com/watch?v=LauQ6LWTkxM
Simulator: 
http://rpg.ifi.uzh.ch/esim
Slowly becoming popular in robotics and vision
Basic reading:

- Szeliski textbook, Sections 3.13, 3.5.5, 9.3.4, 10.4.3.
  The original Poisson image editing paper.
  A great resource (entire course!) for gradient-domain image processing.
  A paper on photography with flash and no-flash pairs, using gradient-domain image processing.

Additional reading:

  A great resource (entire course!) for gradient-domain image processing.
  A great resource (entire course!) for gradient-domain image processing.
  A paper on photography with flash and no-flash pairs, using gradient-domain image processing.

References

162