Gradient-Domain Processing for Large Images

Misha Kazhdan and Hugues Hoppe
Johns Hopkins University Microsoft Research
Outline

• Motivation
  – Image Stitching
  – LDR Compression

• What’s the problem?
• What’s the big problem?
• The Big Picture
Motivation

We tend to think of an image as a 2D array of values, with a color associated to each of the $W \times H$ pixels. Though this is an easy way to represent images, it is not how our eyes process visual information. So it might not be the best representation for image processing.
Gradient-Domain Image Processing

Many image processing techniques become easier if the image is represented in the gradient-domain.

- Removing Lighting Effects [Horn ’74, Weiss ’01]
- HDR Compression [Fattal ’02]
- Image Compositing [Perez ’03, Agarwala ’04, Jia ’06]
- Image Stitching [Levin ’04, Agarwala ’07]
- Shadow/Reflection Removal [Finlayson ’02, Agrawal ’05]
Gradient-Domain Image Processing

Rather than representing an image as a disjoint set of pixel values...

\[
\begin{align*}
W & \quad 1 \\
H & \quad 1 \\
\end{align*}
\]

The \((i, j)\)-th entry stores the value of the \((i, j)\)-th pixel.
Gradient-Domain Image Processing

... Images are represented as the set of horizontal and vertical pixel differences.

The \((i, j)\)-th vertical edge stores the difference in colors between the \((i, j)\)-th pixel and the \((i, j + 1)\)-th pixel:

\[ v[i][j] = p[i][j + 1] - p[i][j] \]

The \((k, l)\)-th horizontal edge stores the difference in colors between the \((k, l)\)-th pixel and the \((k + 1, l)\)-th pixel:

\[ h[k][l] = p[k + 1][l] - p[k][l] \]
Gradient-Domain Image Processing

... Images are represented as the set of horizontal and vertical pixel differences. This conforms to our eye’s sensitivity to boundaries:

• In smooth regions the edge-values are small
• At image boundaries the edge-values are large
Applications: Image Stitching

Stitching together different images by merging the pixel values, we get a discontinuity across the seam due to the different camera settings.
Applications: Image Stitching

We can combine the image data by merging the gradients from the two images. Since we want the transition to be smooth, we set the values along the missing edges to zero.
Applications: Image Stitching

We can combine the image data by merging the gradients from the two images. Since we want the transition to be smooth, we set the values along the missing edges to zero.
Applications: HDR Compression

We can now obtain images with a high dynamic range (i.e. more than 8-bits per channel).

- Improved camera sensors
- Combining data from multiple exposures (bracketing)
- Virtual image generation
Applications: HDR Compression

With a single HDR photograph, we can image the scene at many different exposures.

Can we get the exposure info in one LDR image?
Applications: HDR Compression

Observation:
Since our eye is sensitive to image boundaries, we want a single LDR image that captures all of the boundary information.
Applications: HDR Compression

Approach:
Compute the gradient-domain representation and modulate the gradient magnitudes:

- Where gradients are large, make them a little smaller,

This still preserves the image boundaries, but reduces the dynamic range.
Applications: HDR Compression

This gives a single (virtual) LDR image capturing the information at the different exposures simultaneously.
Outline

• Motivation
• What’s the problem?
  – Posing the problem
  – Solving the problem
• What’s the big problem?
• The Big Picture
What’s the Problem?

For this to be useful, we need a way to transition from gradient-domain-space representations to RGB-space representations.
What’s the Problem?

Finding the set of RGB values whose differences conform to the desired gradient constraints is not possible!!!
What’s the Problem?

Suppose we have figured out the value at pixel $p[i][j]$ and we know all the gradient constraints.
What’s the Problem?

Suppose we have figured out the value at pixel \( p[i][j] \) and we know all the gradient constraints. If pixel values conform to the desired gradients, the neighbor’s value is:

- \( p[i+1][j] = p[i][j] + h[i][j] \)
What's the Problem?

Suppose we have figured out the value at pixel $p[i][j]$ and we know all the gradient constraints. And then the neighbor's neighbor's value is:

- $p[i+1][j+1] = p[i][j] + h[i][j] + v[i+1][j]$
What’s the Problem?

Suppose we have figured out the value at pixel $p[i][j]$ and we know all the gradient constraints. But there are two ways to define the value at pixel $p[i+1][j+1]$: 

- $p[i+1][j+1] = p[i][j] + h[i][j] + v[i+1][j]$  
- $p[i][j+1] = p[i][j] + v[i][j]$
What’s the Problem?

Suppose we have figured out the value at pixel \( p[i][j] \) and we know all the gradient constraints. But there are two ways to define the value at pixel \( p[i + 1][j + 1] \):

- \( p[i + 1][j + 1] = p[i][j] + h[i][j] + v[i + 1][j] \)
- \( p[i + 1][j + 1] = p[i][j] + v[i][j] + h[i][j + 1] \)
What’s the Problem?

Suppose we have figured out the value at pixel $p[i][j]$ and we know all the gradient constraints. But there are two ways to define the value at pixel $p[i+1][j+1]$:

- $p[i+1][j+1] = p[i][j] + h[i][j] + v[i+1][j]$
- $p[i+1][j+1] = p[i][j] + v[i][j] + h[i][j+1]$

Since we didn’t restrict the definition of the gradients, there is not guarantee that the two definitions are equal.
What’s the Problem?

We can think of this as a system of equations defining the gradients in terms of pixel values:

\[
\begin{align*}
D & \quad \text{Differencing Matrix} \\
\begin{pmatrix} x \end{pmatrix} & \quad \text{pixel values} \\
\begin{pmatrix} b \end{pmatrix} & \quad \text{horizontal differences} \\
& \quad \text{vertical differences}
\end{align*}
\]
What’s the Problem?

To solve for the pixel values from the gradient constraints, we want to invert the matrix.

$$D x = b$$

$$x = D^{-1} b$$
What’s not the Problem

To solve for the pixel values from the gradient constraints, we want to invert the matrix.

Since the number of gradient constraints is larger than the number of pixels, the system is over-constrained.
What is the Problem

✗ We can’t solve for pixel values whose differences match the specified gradients:

\[
\text{find } x \text{ satisfying } \|Dx - b\| = 0
\]

✓ We can solve for pixel values whose differences are closest to the specified gradients:

\[
\text{find } x \text{ minimizing } \|Dx - b\|
\]
What is the Problem

The Normal Equation:
Solving the minimization problem can be done by pre-multiplying both sides of the equation by $D^T$ to turn the matrix into a square one:

\[
\begin{pmatrix}
\text{Differencing Matrix}
\end{pmatrix}
\begin{pmatrix}
\text{pixel values}
\end{pmatrix}
= 
\begin{pmatrix}
\text{horizontal differences} \\
\text{vertical differences}
\end{pmatrix}
\]
What is the Problem

The Normal Equation:

Solving the minimization problem can be done by pre-multiplying both sides of the equation by $D^T$ to turn the matrix into a square one:

$$D^T D x = D^T b$$

- $D^T$: Differencing Matrix Transpose
- $D$: Differencing Matrix
- $x$: pixel values
- $D^T$: Differencing Matrix Transpose
- $b$: horizontal and vertical differences
What is the Problem

The Normal Equation:
Solving the minimization problem can be done by pre-multiplying both sides of the equation by $D^\top$ to turn the matrix into a square one:

\[ D^\top \cdot D \cdot x = D^\top \cdot b \]
What is the Problem

We end up solving a linear system of the form $Lx = f$ for the unknown pixel values $x$
What is the Problem

We end up solving a linear system of the form
\[ Lx = f \]
for the unknown pixel values \( x \)

\[ -Lx = (D^\top D)x \]

is the Laplacian of \( x \)

\[ (Lx)[i][j] = 4x[i][j] - x[i][j + 1] - x[i + 1][j] - x[i - 1][j] - x[i][j - 1] \]
What is the Problem

We end up solving a linear system of the form
\( Lx = f \) for the unknown pixel values \( x \)

\(-Lx = (D^\top D)x \) is the Laplacian of \( x \)

\(-f = D^\top b \) is the divergence of the constraints:

\( f[i][j] = (v[i][j] - v[i][j-1]) + (h[i][j] - h[i-1][j]) \)
What is the Problem

We end up solving a linear system of the form

\[ Lx = f \]

for the unknown pixel values \( x \)

\[-Lx = (D^T D)x \] is the Laplacian of \( x \)

\[-f = D^T b \] is the divergence of the constraints
# Solving the Problem

## Gauss Seidel Iteration:

For each pixel \((i,j)\)

1. Assume the rest of the pixel values are correct
2. Update \(x[i,j]\) so that the equation is satisfied:

\[
4x[i,j]-x[i+1,j]-x[i-1,j]-x[i,j+1]-x[i,j-1] = f[i,j]
\]

\[\Downarrow\]

\[
x[i,j] \leftarrow \frac{(f[i,j] + x[i+1,j] + x[i-1,j] + x[i,j+1] + x[i,j-1])}{4}
\]

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

Gauss Seidel Iteration:

For each pixel \((i,j)\)

1. Assume the rest of the pixel values are correct
2. Update \(x[i][j]\) so that the equation is satisfied:

\[
4x[i][j] - x[i+1][j] - x[i-1][j] - x[i][j+1] - x[i][j-1] = f[i][j]
\]

Similar to the way we solved the radiosity equation

So how many Gauss Seidel iterations must we perform in order to get to the right answer?
Too Many!

**Convergence of Gauss Seidel Iterations**

- **Error** vs **Iterations**

  - **Max**
  - **RMS**
Too Many Gauss Seidel Iterations?

Analyzing the convergence of the Gauss-Seidel:

✓ fine details appear quickly

✗ low-resolution details resolve slowly

⇒ Instead of solving for the low-resolution component using a high-resolution image, down-sample and solve for the low-res more efficiently.
Solving the Problem

Multi-grid Solvers

1. Perform a few solver iterations on the high-res image
2. Compute the residual and down-sample
3. Solve the low-res, down-sampled residual
4. Up-sample the low-res solution and add to the high-res
5. Perform a few more solver iterations on the high-res image
Solving the Problem

Multi-grid Solvers

1. Perform a few solver iterations on the high-res image
Solving the Problem

Multi-grid Solvers

2. Compute the residual and down-sample
Solving the Problem

Multi-grid Solvers

3. Solve the low-res, down-sampled residual
Solving the Problem

Multi-grid Solvers

4. Up-sample the low-res solution and add to the high-res
Solving the Problem

Multi-grid Solvers

5. Perform a few more solver iterations on the high-res image
Solving the Problem

To solve the lower-resolution problem, the process is repeated recursively.
Solving the Problem

To solve the lower-resolution problem, the process is repeated recursively.
Outline

• Motivation
• What’s the problem?
• What’s the big problem?
• The Big Picture
The Big Problem

When the image is too large to fit into memory, we need to stream it in from disk.

\[ D := \text{Down-Sampling Operator} \]
\[ U := \text{Up-Sampling Operator} \]
The Big Problem

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ Only a small part of the image needs be memory-resident at any time:

\[ L \leftarrow \left( f[i][j] + x[i][j+1] + x[i][j-1] + x[i+1][j] + x[i-1][j+1] + x[i-1][j-1] \right) / 4 \]
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.
⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.
⇒ We can do the updates by only storing 3 image rows in memory at a time.

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.

⇒ We can do the updates by only storing 3 image rows in memory at a time.
Streaming Multiple GS Iterations

✓ Pixel updates only require knowledge of pixel values in the 1-ring neighborhood.
⇒ We can do the updates by only storing 3 image rows in memory at a time.
The Big Problem

- We have to update all the image pixels before starting the next Gauss-Seidel iteration.

⇒ $k$ GS iterations require $k$ passes through the data.

$D :=$ Down-Sampling Operator
$U :=$ Up-Sampling Operator
Streaming Multiple GS Iterations

With careful scheduling, we can perform multiple GS iterations in a single pass through the data.

\[ D := \text{Down-Sampling Operator} \]
\[ U := \text{Up-Sampling Operator} \]
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   
   Update pixels in row $j+i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

– Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For ($i = k$; $i > 0$; $i -$)
   
   Update pixels in in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

– Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For ($i = k; i > 0; i--$)
   Update pixels in in row $j + i$

2. Increment $j$

While $j < height$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For ($i = k; i > 0; i--$)
   
   Update pixels in in row $j + i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform \( k \) GS iterations in one pass:

- Keep a window of \( k + 2 \) rows in memory as we sweep through the data.

Initialize start of the window at row \( j = -k \)

Do

1. For( \( i = k; i > 0; i-- \) )
   - Update pixels in row \( j + i \)

2. Increment \( j \)

While \( j < \text{height} \)
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For $i = k; i > 0; i--$
   - Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

  Initialize start of the window at row $j = -k$

  Do

  1. For($i = k; i > 0; i--$)
     
     update pixels in in row $j+i$

  2. Increment $j$

  While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For ($i = k; i > 0; i--$)  
    Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   Update pixels in in row $j + i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   
   Update pixels in in row $j + i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For ($i = k; i > 0; i--$)
   
   Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform \( k \) GS iterations in one pass:

- Keep a window of \( k + 2 \) rows in memory as we sweep through the data.

Initialize start of the window at row \( j = -k \)

Do

1. For( \( i = k; i > 0; i-- \) )
   - Update pixels in in row \( j + i \)

2. Increment \( j \)

While \( j < \text{height} \)

\( k = 3 \) GS Iterations
Streaming Multiple GS Iterations

To perform \( k \) GS iterations in one pass:

- Keep a window of \( k + 2 \) rows in memory as we sweep through the data.

Initialize start of the window at row \( j = -k \)

Do

1. For\( (i = k; i > 0; i--) \)
   Update pixels in in row \( j + i \)

2. Increment \( j \)

While \( j < \text{height} \)
Streaming Multiple GS Iterations

To perform \( k \) GS iterations in one pass:

- Keep a window of \( k + 2 \) rows in memory as we sweep through the data.

Initialize start of the window at row \( j = -k \)

Do

1. For( \( i = k; i > 0; i-- \) )
   Update pixels in in row \( j + i \)

2. Increment \( j \)

While \( j < \text{height} \)
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For ($i = k; i > 0; i--$)
   Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   - Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   
   Update pixels in in row $j + i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   Update pixels in in row $j + i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For $(i = k; i > 0; i--)$
   
   Update pixels in row $j + i$

2. Increment $j$

While $j < \text{height}$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j = -k$

Do

1. For($i = k; i > 0; i--$)
   Update pixels in row $j+i$

2. Increment $j$

While $j < \text{height}$
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j=-k$

Do

1. For($i=k; i>0; i--$) Update pixels in in row $j+i$

2. Increment $j$

While $j<height$

$k = 3$ GS Iterations
Streaming Multiple GS Iterations

To perform $k$ GS iterations in one pass:

- Keep a window of $k + 2$ rows in memory as we sweep through the data.

Initialize start of the window at row $j=-k$

Do

1. For ($i=k; i > 0; i--$)
   Update pixels in in row $j+i$

2. Increment $j$

While $j<height$
Streaming Multiple GS Iterations

With some more careful scheduling, we can also stream directly between levels.

$D := \text{Down-Sampling Operator}$

$U := \text{Up-Sampling Operator}$
Streaming Multiple GS Iterations

With some more careful scheduling, we can also stream directly between levels.

\[ D := \text{Down-Sampling Operator} \]
\[ U := \text{Up-Sampling Operator} \]
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

• Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.

• Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- **Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.**
- **Once we add a new row to the low-resolution system, we can start solving there as well.**
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.

- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.
- **Once we add a new row to the low-resolution system, we can start solving there as well.**
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

• Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.

• **Once we add a new row to the low-resolution system, we can start solving there as well.**
Streaming Multiple GS Iterations

When down-sampling, we can start solving the low-res problem before the high-res completes.

- Each time we solve two high-resolution rows, we down-sample to get the new row in the low resolution system.
- Once we add a new row to the low-resolution system, we can start solving there as well.
Processing Large Images

Carefully scheduling computation, we perform a multi-grid cycle in just two streaming passes.

\[ D := \text{Down-Sampling Operator} \]
\[ U := \text{Up-Sampling Operator} \]
Outline

• Motivation
• What’s the problem?
• What’s the big problem?
• The Big Picture
How Big is Big?

St James:

• Stitched from 643 photographs
• Contains 3.3 billion (88,309 x 37,842) pixels
How Big is Big?

- Stitched from 643 photographs
- Contains 3.3 billion (88,309 x 37,842) pixels
How Big is Big?

St James:

• Solver Time: 1:27:50
• Peak Memory Usage: 408 MB
How Big is Big?

Are these good numbers?

- Peak Memory Usage: 408 MB
- Solver Time: 1:27:50
How Big is Big?

Are these good numbers?

- Peak Memory Usage: 408 MB
- Solver Time: 1:27:50

Consider the disk I/O (assuming 16-bit precision):

- Down-Sampling
  - Read in the gradient constraints: 40 GB
  - Write out $x$ and $f$: 53 GB

- Up-Sampling
  - Read in $x$ and $f$: 53 GB
  - Write out the image: 20 GB
How Big is Big?

Are these good numbers?

- Peak Memory Usage: 408 MB
- Solver Time: 1:27:50

Consider the disk I/O (assuming 16-bit precision):

- Down-Sampling
  - Read in the gradient constraints: 40 GB
  - Write out $x$ and $f$: 53 GB

- Up-Sampling
  - Read in $x$ and $f$: 53 GB
  - Write out the image: 20 GB

If this were in-core, we would have needed 93 GB of storage.

Assuming 30-35 MB/s reads and writes, I/O alone would take:

1:19:02 -- 1:32:13
How Big is Big?

**WWT:**
- Stitched from 1790 photographs
- Comprises a 1 terapixel image
How Big is Big?

WWT:

- Solver Time: 9 hours
- Distributed across a 16-node cluster
How Big is Big?

WWT:

• Solver Time: 9 hours
• Distributed across a 16-node cluster