Gradient-Domain Processing for Large Images

Misha Kazhdan and Hugues Hoppe
Johns Hopkins University Microsoft Research

Outline

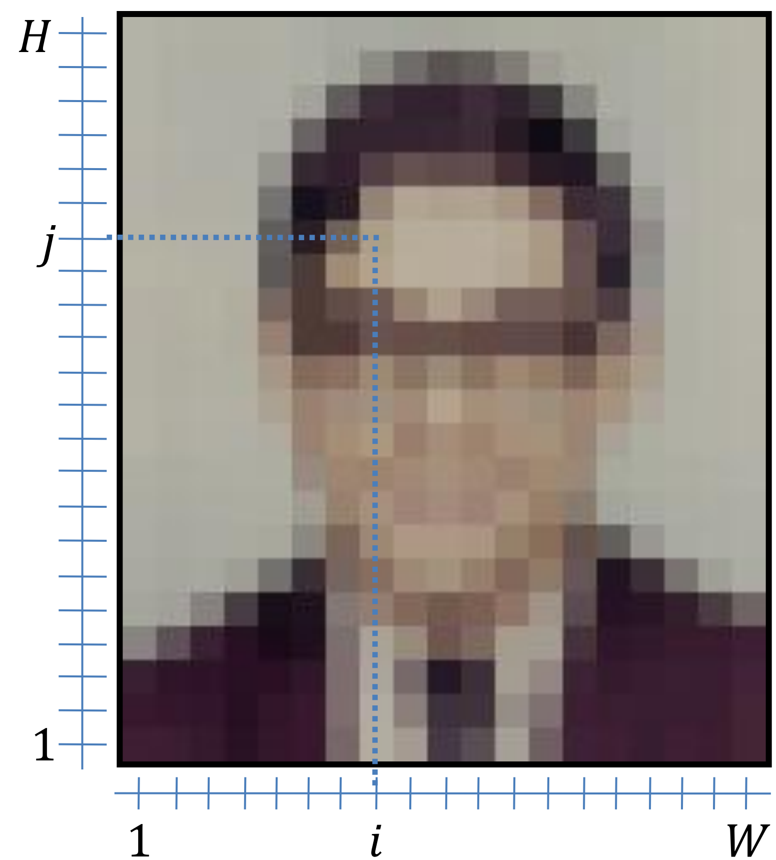
- Motivation
  - Image Stitching
  - LDR Compression
- What’s the problem?
- What’s the big problem?
- The Big Picture

Motivation

We think of an image as a 2D array of values, with a color associated to each of the $W \times H$ pixels.

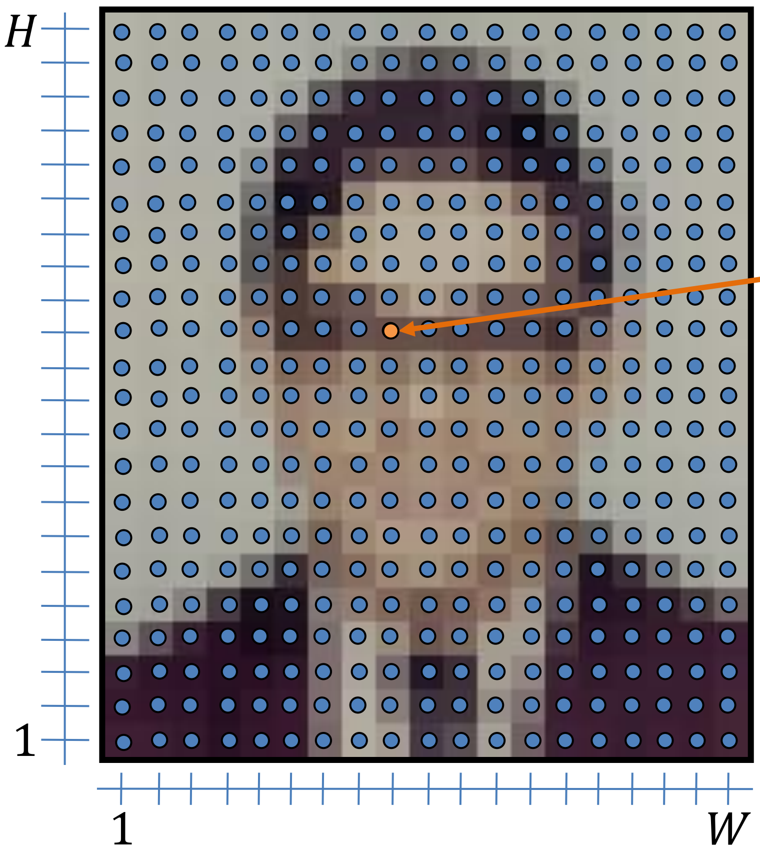
This may not be how our eyes process visual information.

$\Rightarrow$ It might not be the best representation for image processing.



Gradient-Domain Image Processing

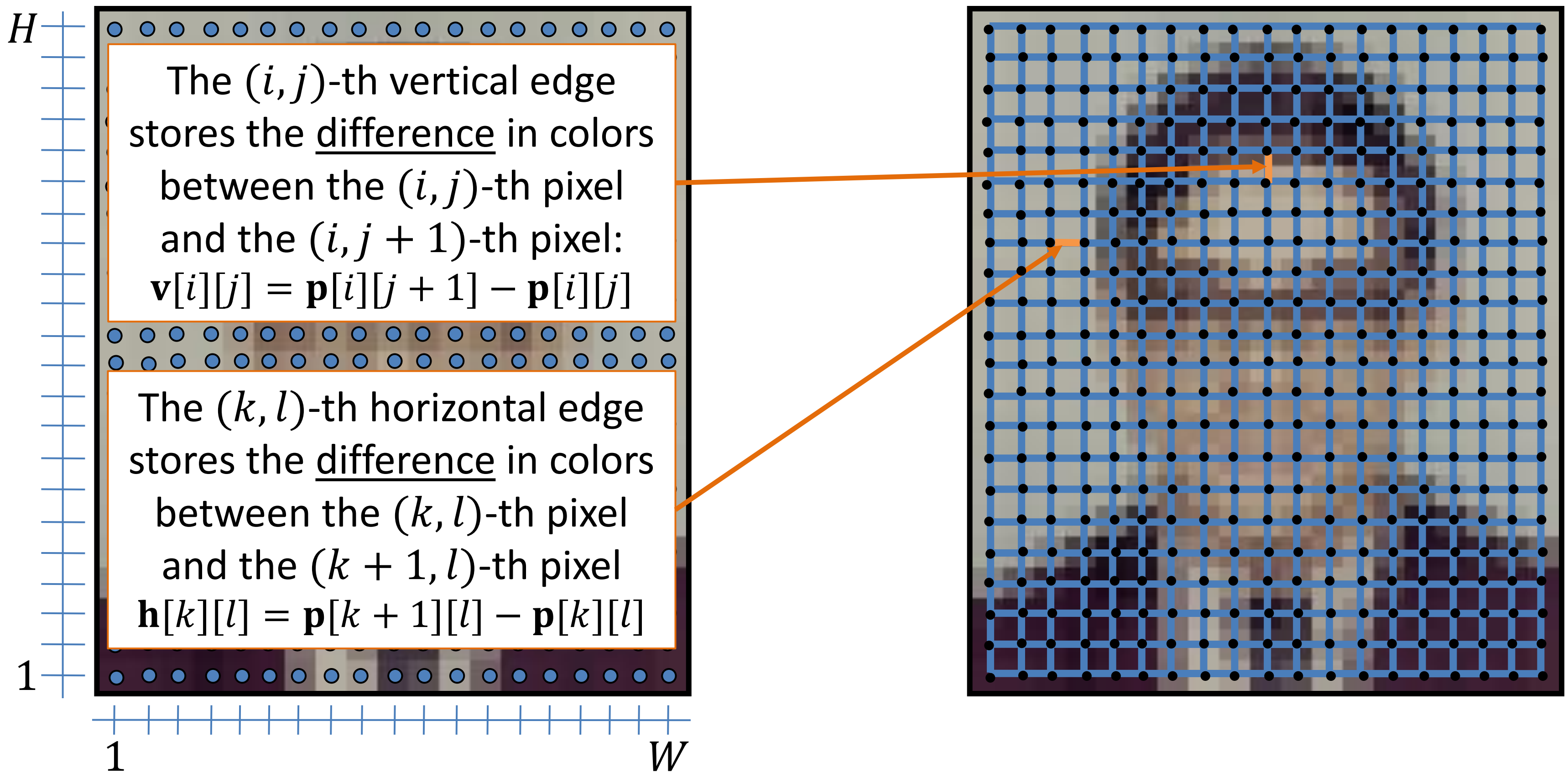
Rather than representing an image as a disjoint set of pixel values…



The (i, j)-th entry stores the value of the (i, j)-th pixel

Gradient-Domain Image Processing

… represent images by the set of horizontal and vertical pixel **differences**.

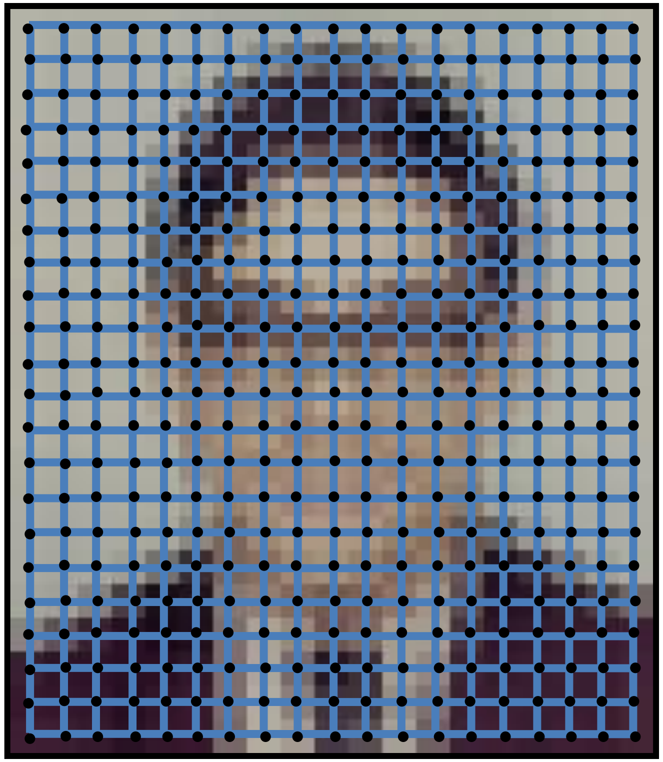


Gradient-Domain Image Processing

… represent images by 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



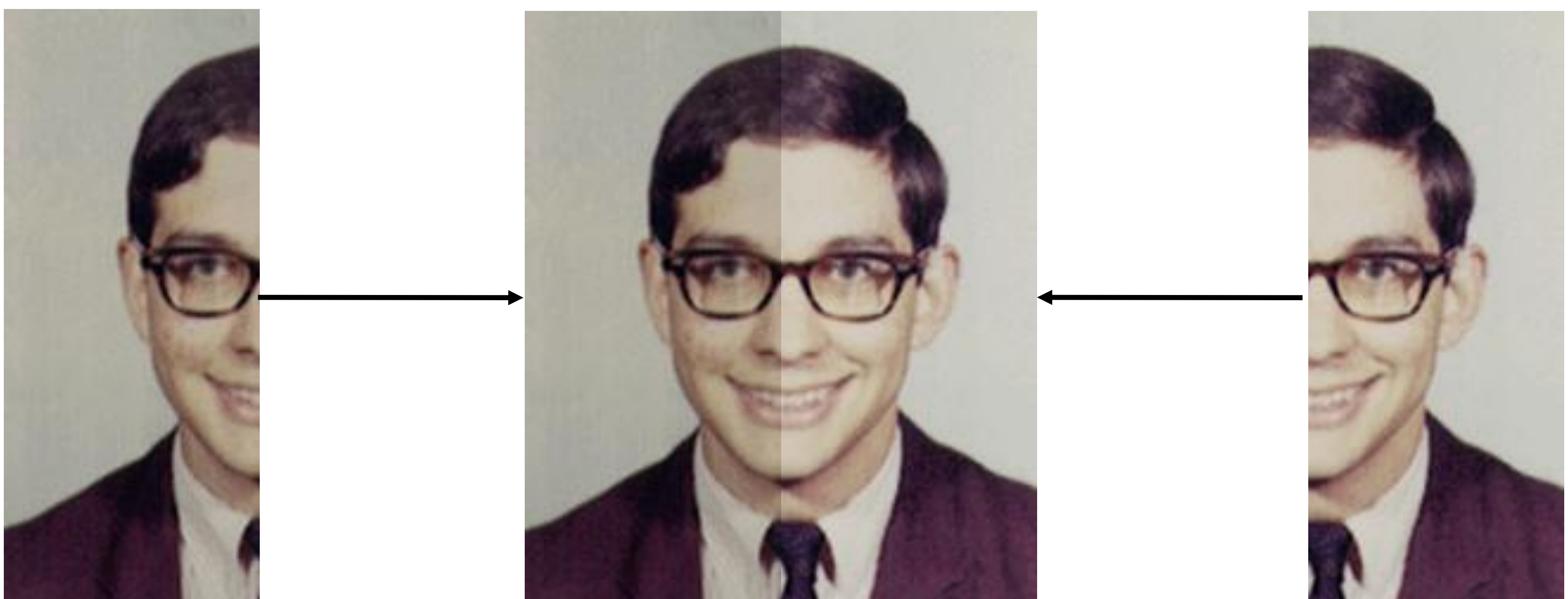
Gradient-Domain Image Processing

Many image processing techniques are easier when formulated 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]

Applications: Image Stitching

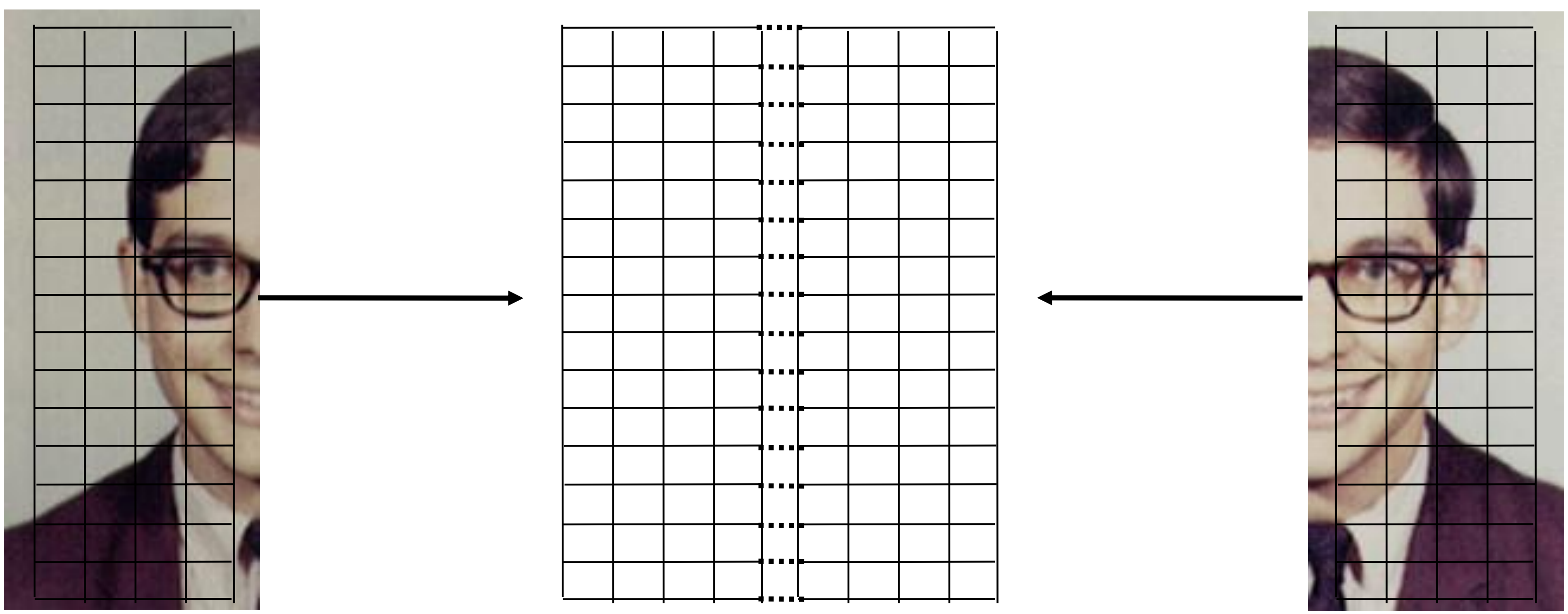
Stitching together images by compositing pixel values, we get a discontinuity across the seam due to the different camera settings.



Applications: Image Stitching

Combine the image data by merging gradients from the two images.

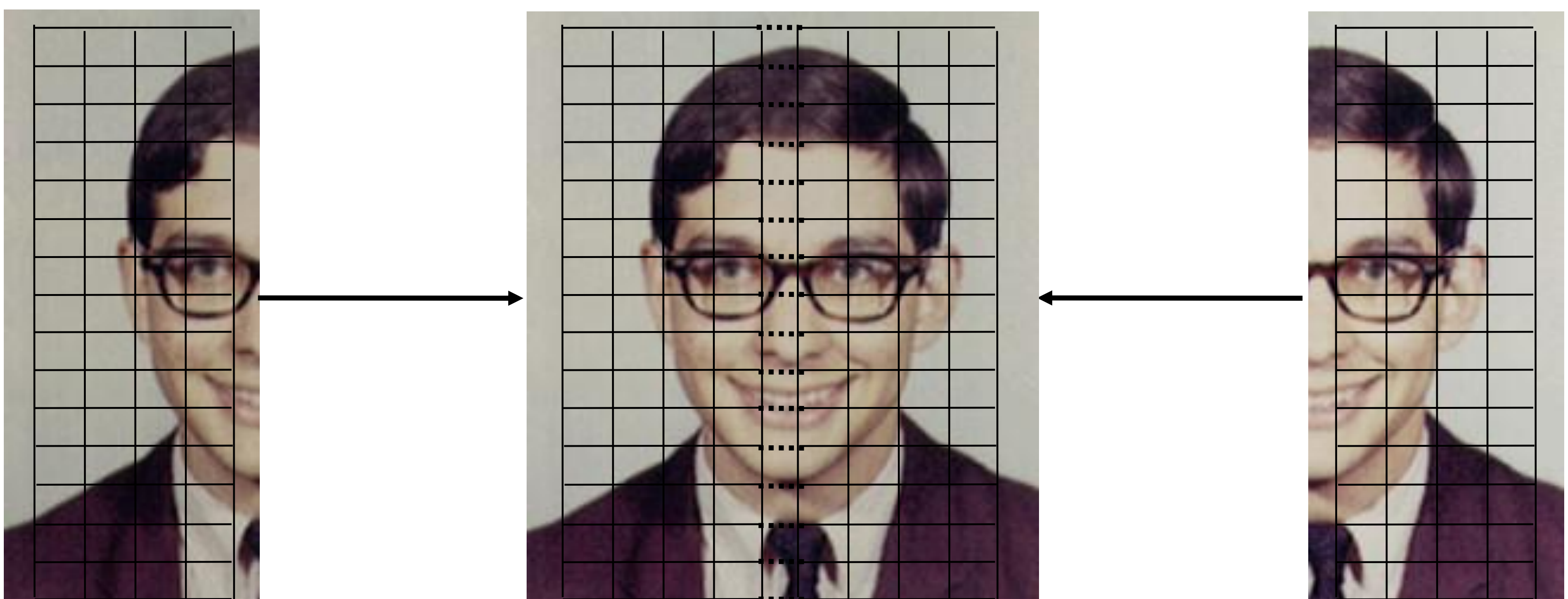
For the transition to be smooth, set the values along the missing edges to zero.



Applications: Image Stitching

Combine the image data by merging gradients from the two images.

For the transition to be smooth, set the values along the missing edges to zero.



Applications: HDR* Compression

Images may be represented with a high dynamic range (i.e. more than 8-bits per channel).

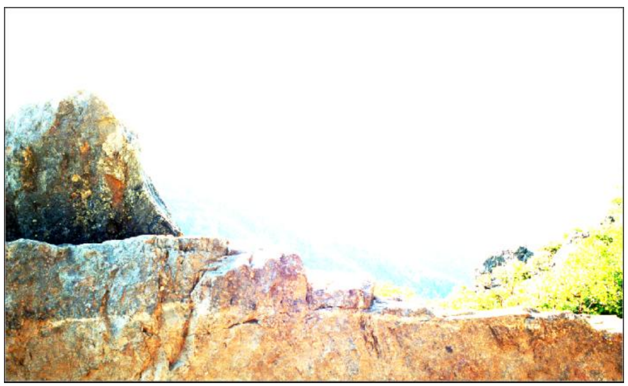
- Improved camera sensors
- Combining data from multiple exposures (bracketing)
- Virtual image generation

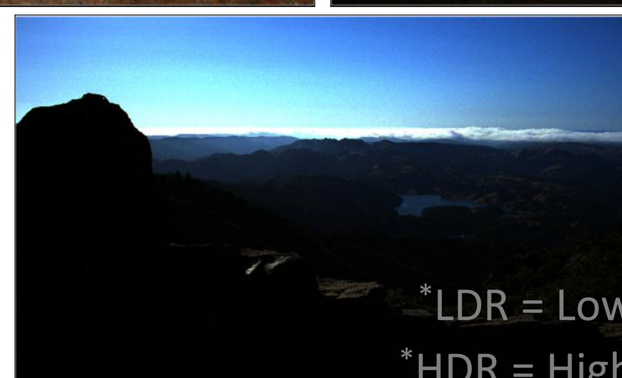
*HDR = High Dynamic Range

Applications: HDR[*] Compression

Images may be represented with a high dynamic range (i.e. more than 8-bits per channel).

How to visualize with an LDR[*] image/display?





[*]LDR = Low Dynamic Range

[*]HDR = High Dynamic Range

Applications: HDR Compression

<u>Observation</u>:

Since our eye is sensitive to image boundaries:

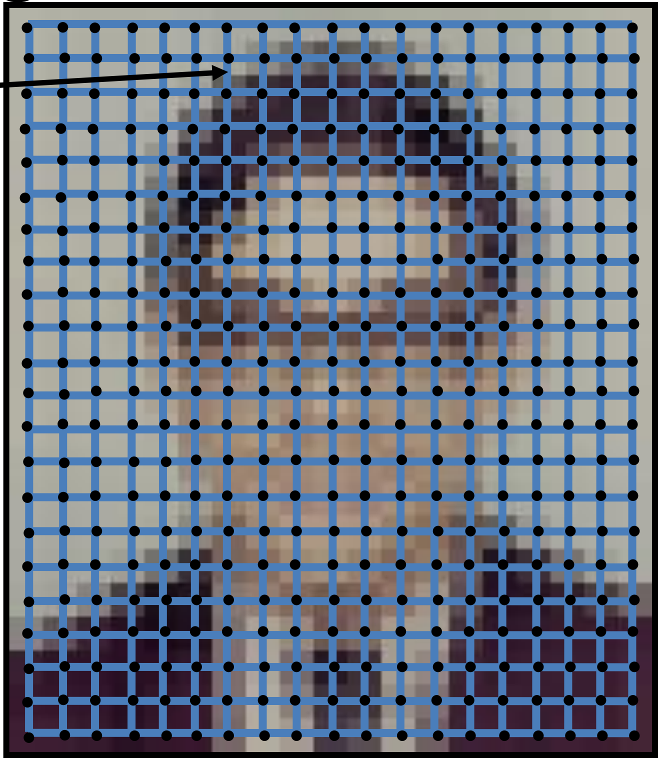
- $\Rightarrow$ The LDR image should preserve the boundaries
- $\Rightarrow$ The transition need not be as abrupt

Applications: HDR Compression

<u>Approach</u>:

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

Where gradients are large make them a little smaller,



This preserves the image boundaries, but reduces the dynamic range.

Applications: HDR Compression

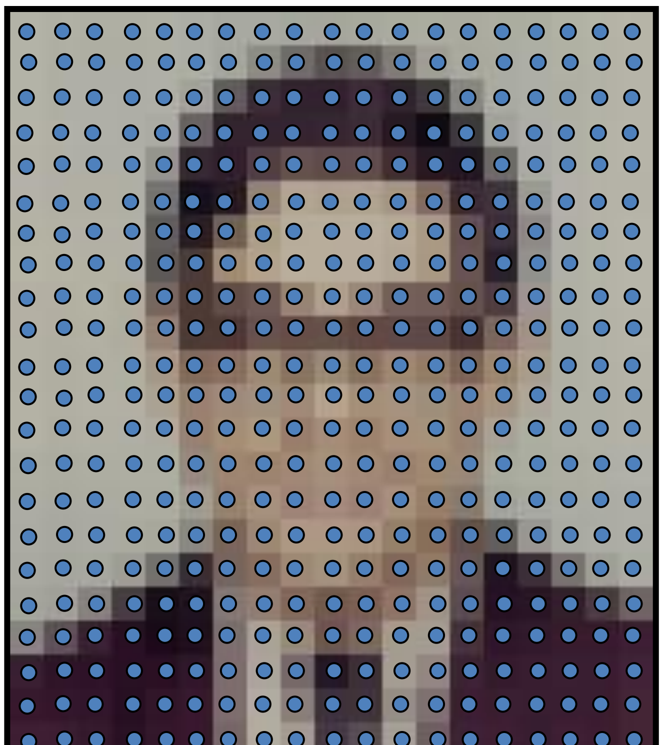
This gives a single (virtual) LDR image capturing the information at the different exposures, while still representing edges.

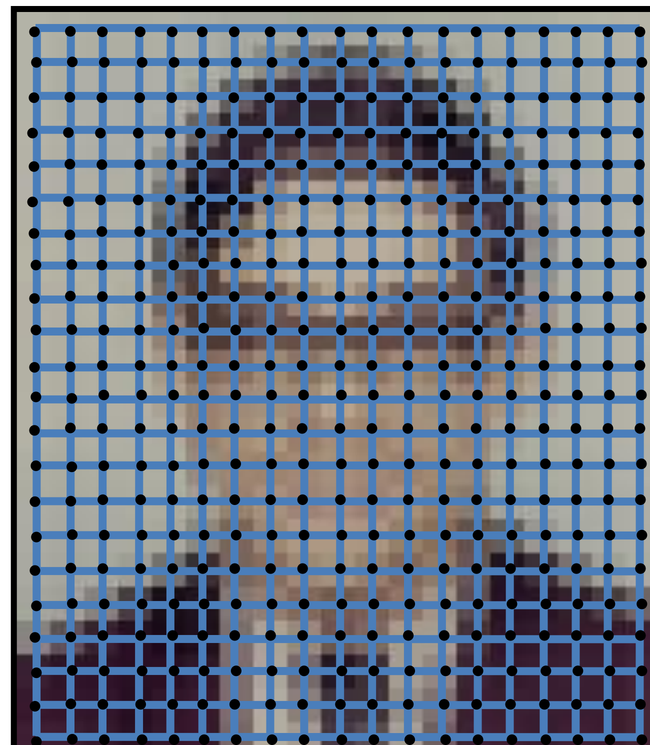
Outline

- Motivation
- What’s the problem?
  - Posing the problem
  - Solving the problem
- What’s the big problem?
- The Big Picture

The Problem

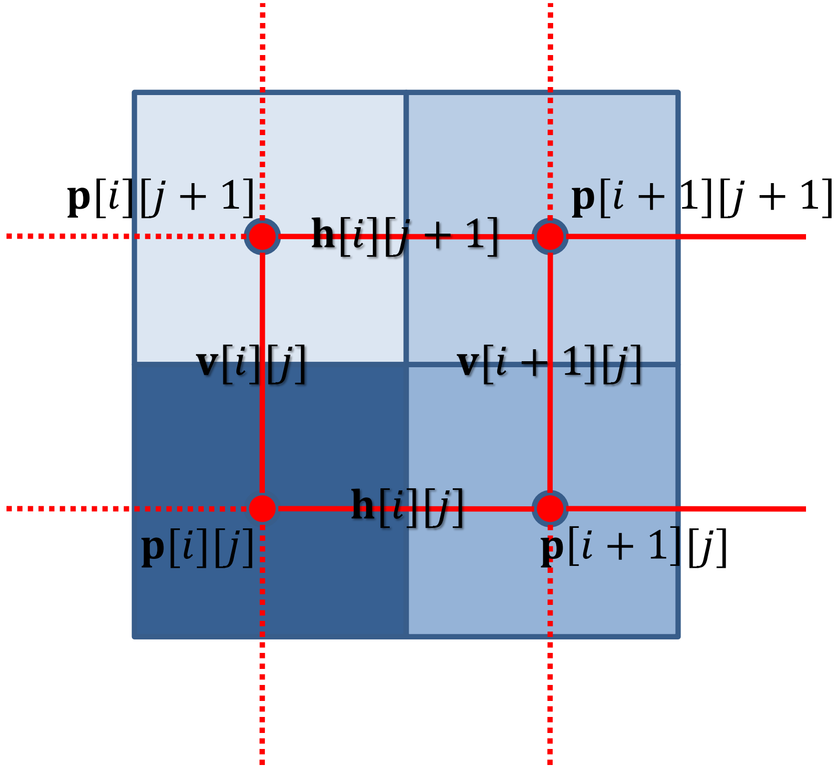
We need to transition from gradient-domain-space representations back to color-space.





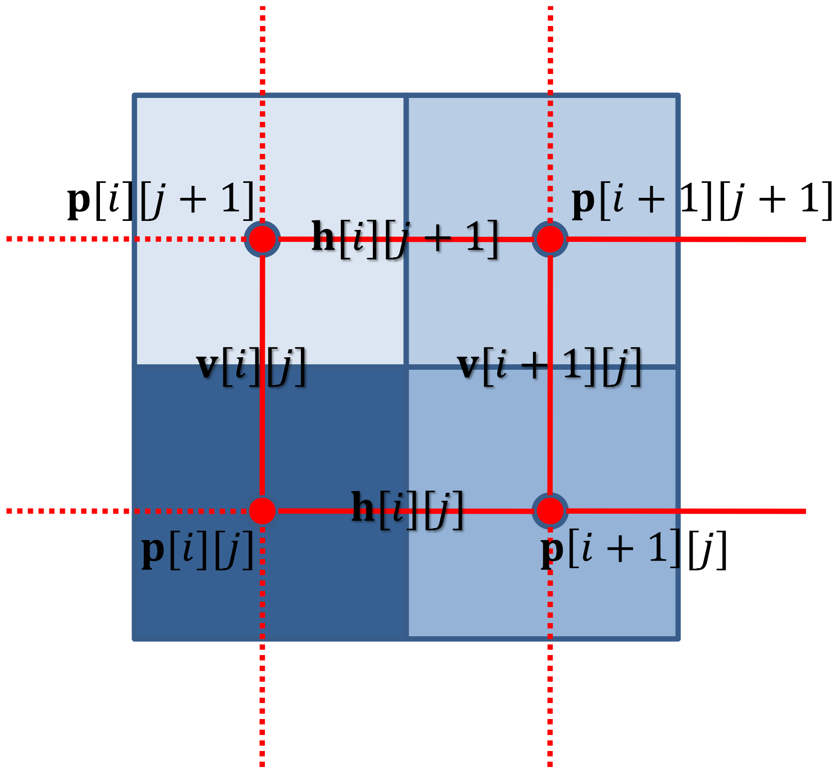
The Problem with the Problem

Finding colors whose differences conform to the gradients may not be possible.



The Problem with the Problem

Given the color at pixel $\mathbf{p}[i][j]$ and the target gradients, what is the color at $\mathbf{p}[i+1][j+1]$?

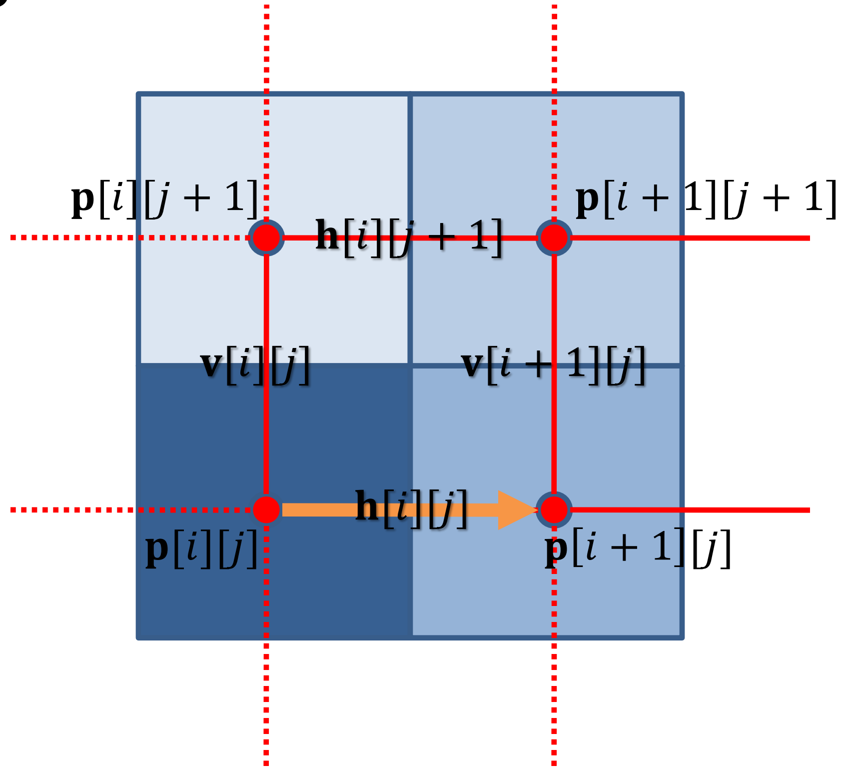


The Problem with the Problem

Given the color at pixel $\mathbf{p}[i][j]$ and the target gradients, what is the color at $\mathbf{p}[i+1][j+1]$?

To fit the gradients, the neighbor's value is:

$$\mathbf{p}[i+1][j] = \mathbf{p}[i][j] + \mathbf{h}[i][j]$$

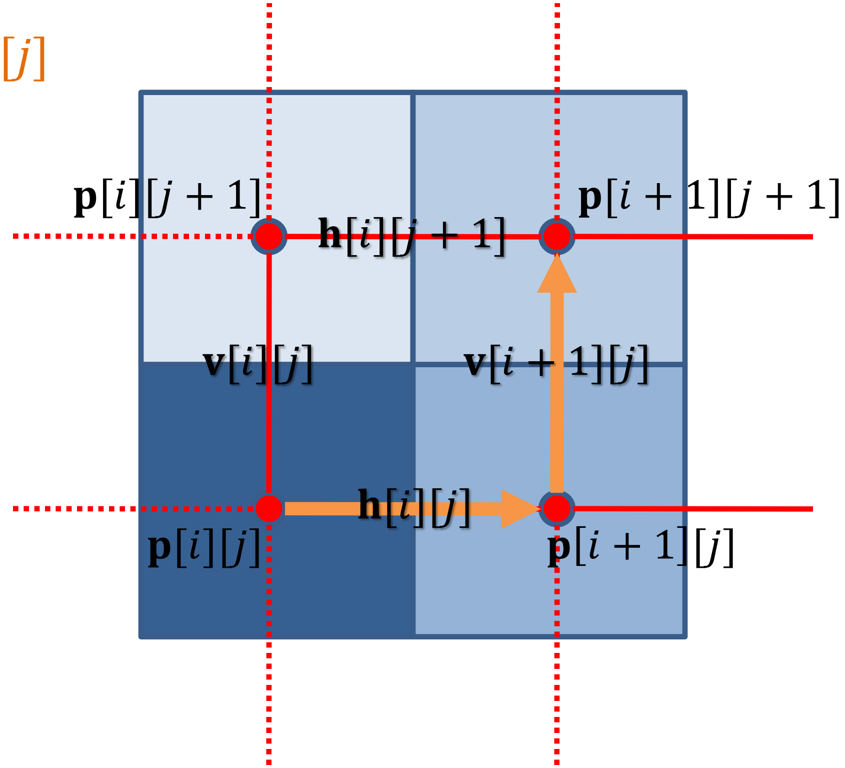


The Problem with the Problem

Given the color at pixel $\mathbf{p}[i][j]$ and the target gradients, what is the color at $\mathbf{p}[i+1][j+1]$?

… and the neighbor's neighbor's value is:

$$\mathbf{p}[i+1][j+1] = \mathbf{p}[i][j] + \mathbf{h}[i][j] + \mathbf{v}[i+1][j]$$



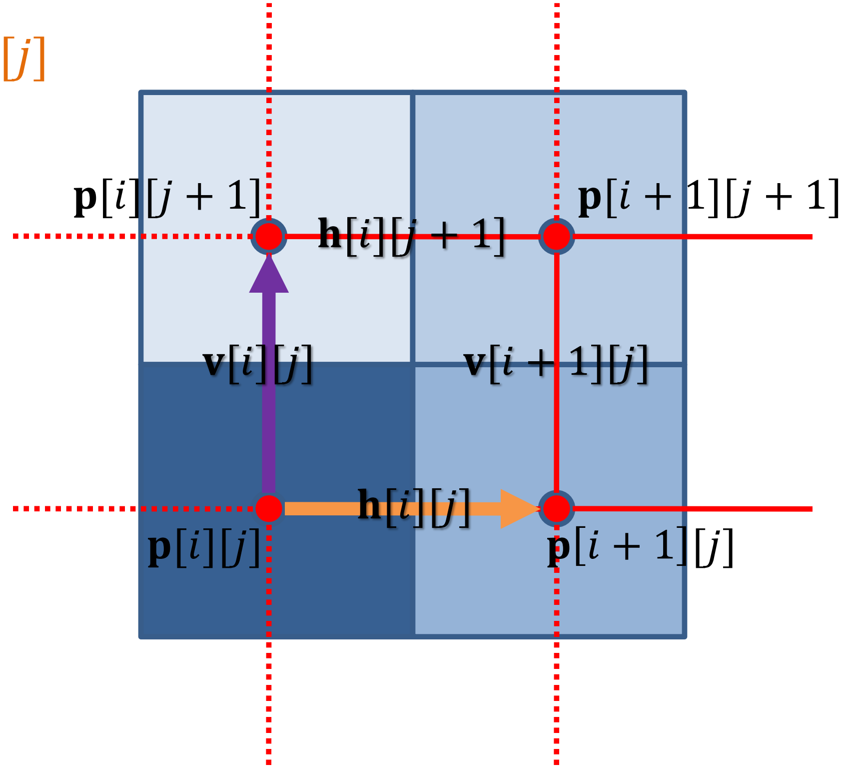
The Problem with the Problem

Given the color at pixel $\mathbf{p}[i][j]$ and the target gradients, what is the color at $\mathbf{p}[i+1][j+1]$?

Alternatively:

$\mathbf{p}[i+1][j+1] = \mathbf{p}[i][j] + \mathbf{h}[i][j] + \mathbf{v}[i+1][j]$

$\mathbf{p}[i][j+1] = \mathbf{p}[i][j] + \mathbf{v}[i][j]$

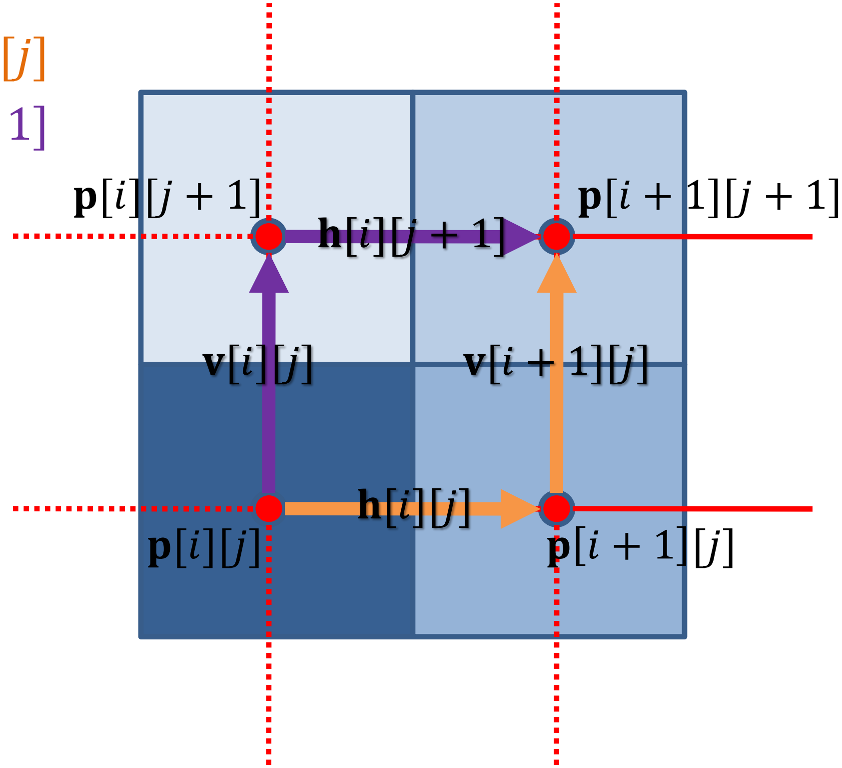


The Problem with the Problem

Given the color at pixel $\mathbf{p}[i][j]$ and the target gradients, what is the color at $\mathbf{p}[i+1][j+1]$?

Alternatively:

$$\mathbf{p}[i+1][j+1] = \mathbf{p}[i][j] + \mathbf{h}[i][j] + \mathbf{v}[i+1][j]$$
$$\mathbf{p}[i+1][j+1] = \mathbf{p}[i][j] + \mathbf{v}[i][j] + \mathbf{h}[i][j+1]$$



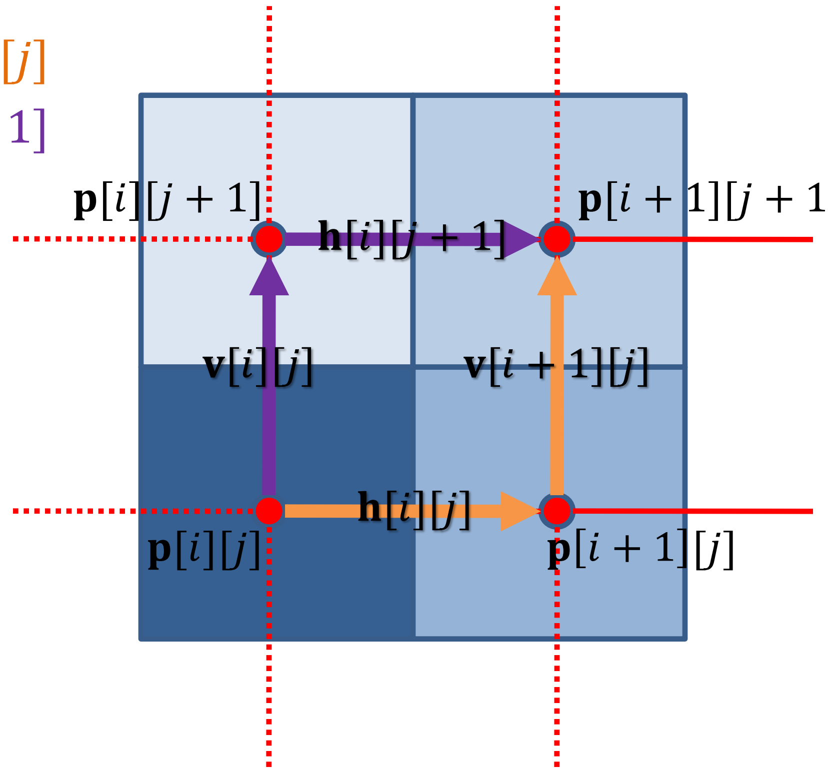
The Problem with the Problem

Given the color at pixel $\mathbf{p}[i][j]$ and the target gradients, what is the color at $\mathbf{p}[i+1][j+1]$?

Alternatively:

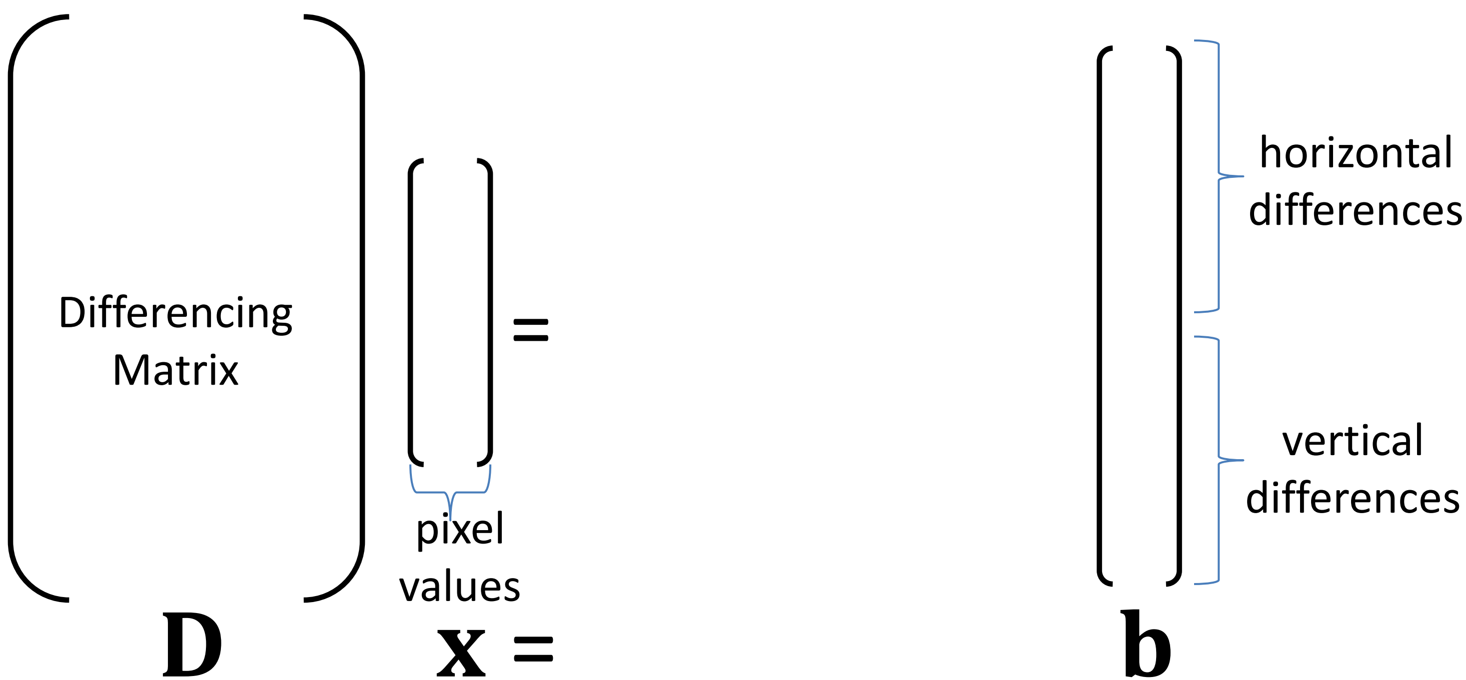
$$\mathbf{p}[i+1][j+1] = \mathbf{p}[i][j] + \mathbf{h}[i][j] + \mathbf{v}[i+1][j]$$
$$\mathbf{p}[i+1][j+1] = \mathbf{p}[i][j] + \mathbf{v}[i][j] + \mathbf{h}[i][j+1]$$

Without restricting the gradients, the two do not have to match (i.e. the gradient field may not be *curl-free*).



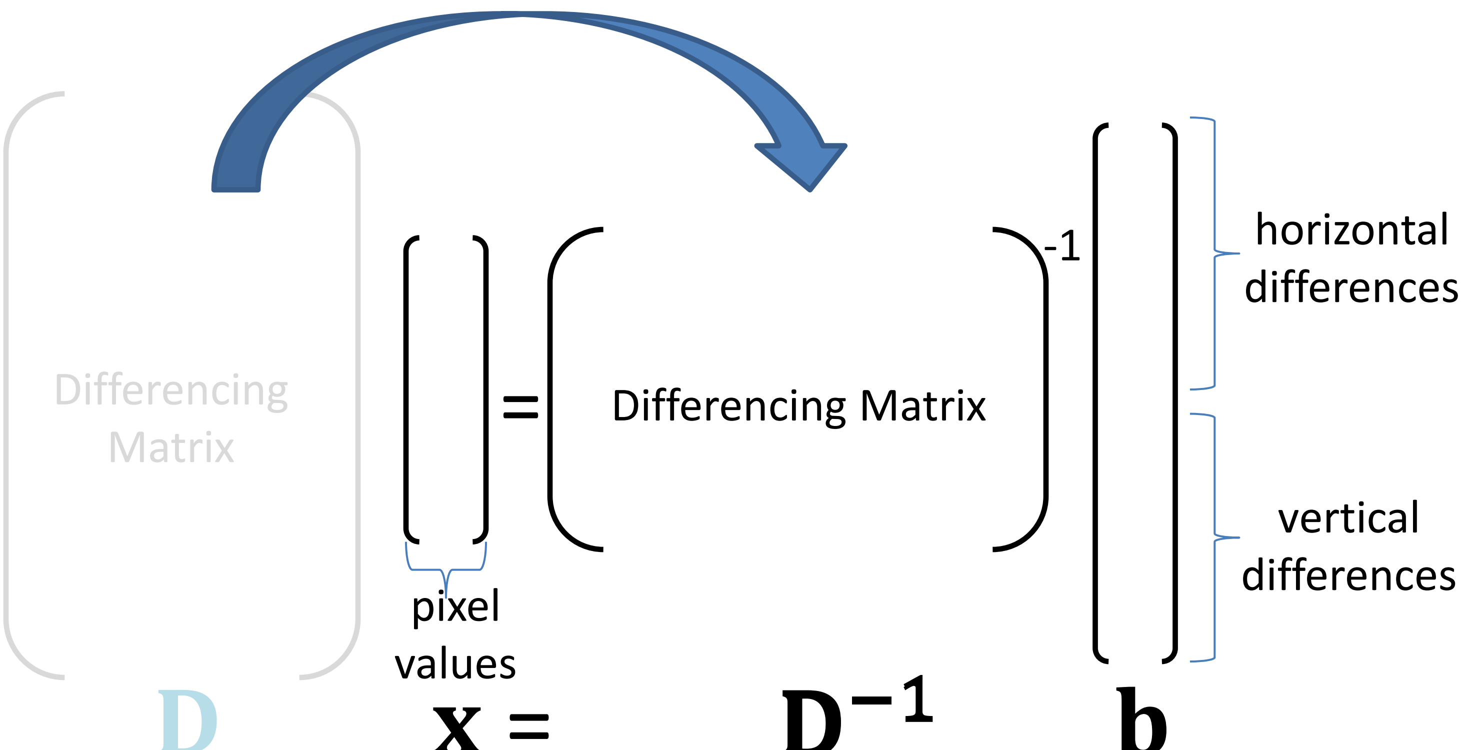
The Problem with the Problem

Consider the system of equations defining the gradients in terms of pixel values…



The Problem with the Problem

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

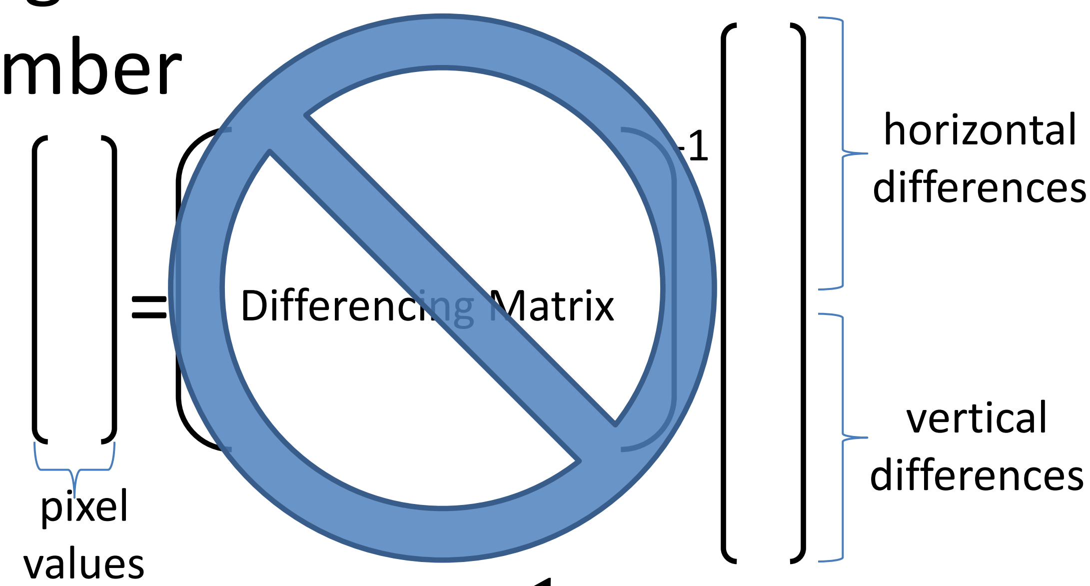


$$\mathbf{x} = \mathbf{D}^{-1}\,\mathbf{b}$$

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.



$$\mathbf{x} = \mathbf{D}^{-1} \mathbf{b}$$

What is the Problem

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

~~find $\mathbf{x}$ satisfying $\|\mathbf{Dx} - \mathbf{b}\|^2 = 0$~~

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

find $\mathbf{x}$ minimizing $\|\mathbf{Dx} - \mathbf{b}\|^2$

What is the Problem

<u>The Normal Equation</u>:

Solving the minimization problem can be done by multiplying both sides by $\mathbf{D}^{\top}$.

$$\begin{pmatrix} \\ \text{Differencing} \\ \text{Matrix} \\ \\ \end{pmatrix} \underbrace{\begin{bmatrix} \\ \\ \end{bmatrix}}_{\text{pixel values}} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{horizontal differences} \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} \text{vertical differences} \end{matrix}$$

$$\mathbf{D} \quad \mathbf{x} = \quad \mathbf{b}$$

What is the Problem

<u>The Normal Equation</u>:

Solving the minimization problem can be done by multiplying both sides by $\mathbf{D}^\top$.

$$\underbrace{\begin{pmatrix} \text{Differencing Matrix} \\ \text{Transpose} \end{pmatrix}}_{\mathbf{D}^\top} \underbrace{\begin{pmatrix} \text{Differencing} \\ \text{Matrix} \end{pmatrix}}_{\mathbf{D}} \underbrace{\begin{bmatrix} \\ \end{bmatrix}}_{\substack{\text{pixel} \\ \text{values} \\ \mathbf{x}}} = \underbrace{\begin{pmatrix} \text{Differencing Matrix} \\ \text{Transpose} \end{pmatrix}}_{\mathbf{D}^\top} \underbrace{\begin{bmatrix} \\ \end{bmatrix}}_{\mathbf{b}} \begin{array}{l} \left.\vphantom{\begin{matrix}a\\a\end{matrix}}\right\} \text{horizontal differences} \\ \left.\vphantom{\begin{matrix}a\\a\end{matrix}}\right\} \text{vertical differences} \end{array}$$

$$\mathbf{D}^\top \mathbf{D} \mathbf{x} = \mathbf{D}^\top \mathbf{b}$$

What is the Problem

<u>The Normal Equation</u>:

Solving the minimization problem can be done by multiplying both sides by $\mathbf{D}^{\top}$.

This turns the matrix into a square one.

$$\begin{pmatrix} \text{Differencing Matrix} \\ \text{Transpose} \\ \text{x} \\ \text{Differencing Matrix} \end{pmatrix} \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{\text{pixel values}} = \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{\text{transformed constraints}}$$

$$\mathbf{D}^{\top} \cdot \mathbf{D} \quad \mathbf{x} = \mathbf{D}^{\top}\mathbf{b}$$

What is the Problem

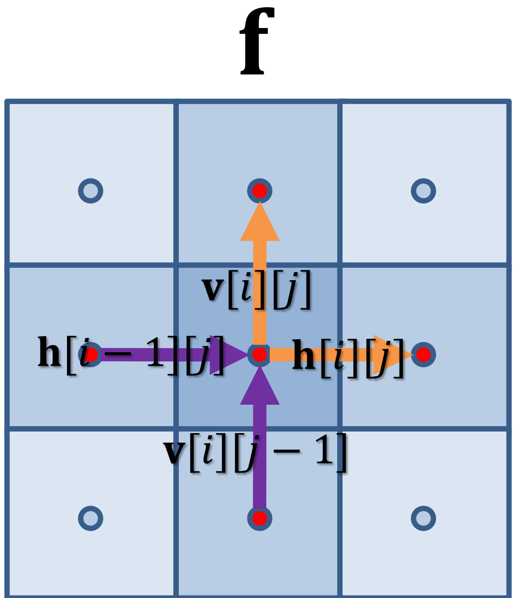
We need to solve a linear system of the form $\mathbf{Lx} = \mathbf{f}$ for the unknown pixel values $\mathbf{x}$

What is the Problem

We need to solve a linear system of the form $\mathbf{L}\mathbf{x} = \mathbf{f}$ for the unknown pixel values $\mathbf{x}$

$\mathbf{f} = \mathbf{D}^\top \mathbf{b}$ is the *divergence* of the constraints:

$$\mathbf{f}[i][j] = (\mathbf{v}[i][j] - \mathbf{v}[i][j-1]\,) + (\mathbf{h}[i][j] - \mathbf{h}[i-1][j])$$



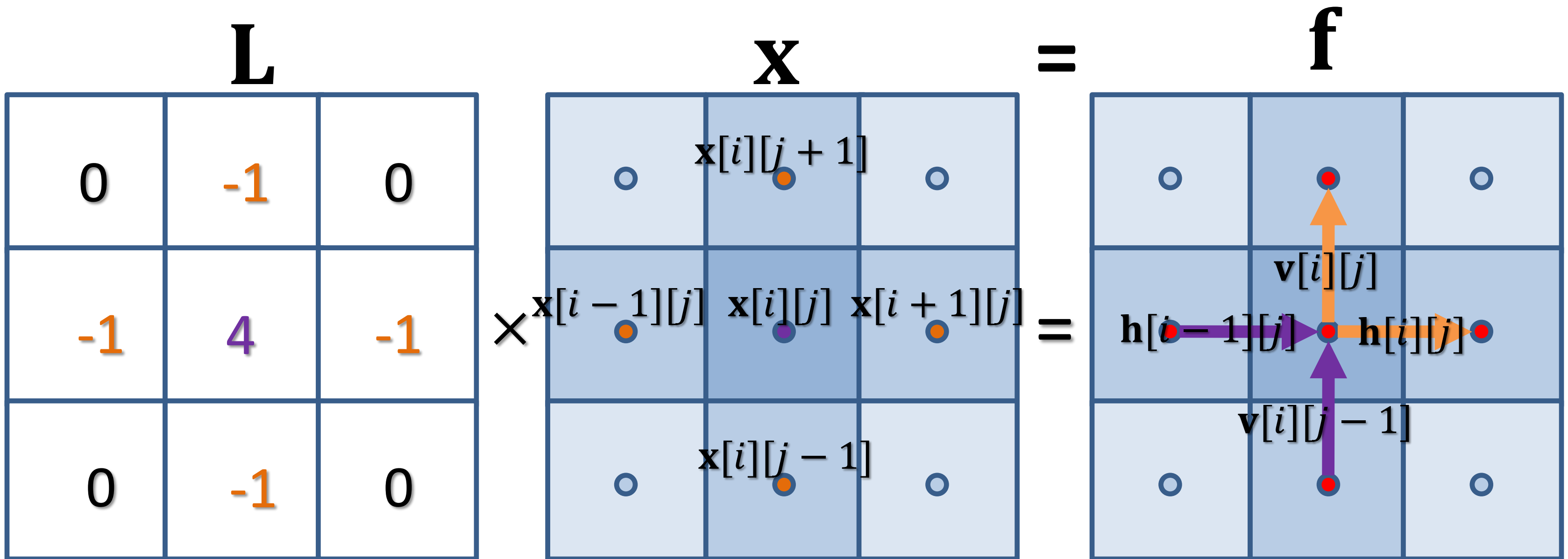
What is the Problem

We need to solve a linear system of the form $\mathbf{Lx} = \mathbf{f}$ for the unknown pixel values $\mathbf{x}$

$\mathbf{f} = \mathbf{D}^\top \mathbf{b}$ is the *divergence* of the constraints:

$\mathbf{Lx} = (\mathbf{D}^\top \mathbf{D})\mathrm{x}$ is the *Laplacian* of $\mathbf{x}$

$$(\mathbf{Lx})[i][j] \; = \; 4\mathbf{x}[i][j] - \mathbf{x}[i][j+1] - \mathbf{x}[i+1][j] - \mathbf{x}[i-1][j] - \mathbf{x}[i][j-1]$$



Solving the Problem

<u>Gauss Seidel Iteration</u>:

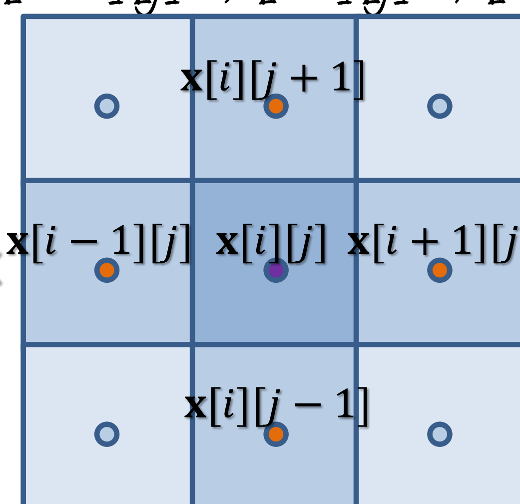
For each pixel (i,j)

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

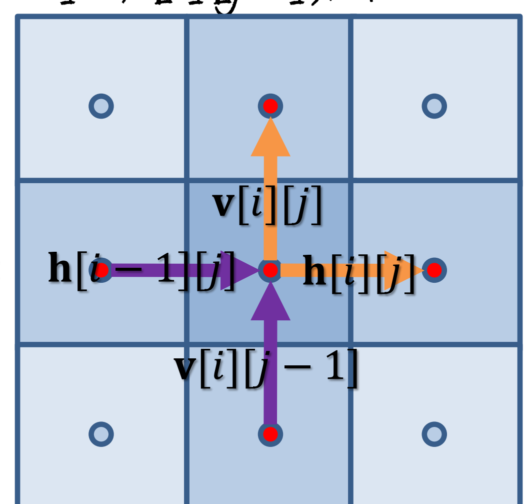
$$4x[i][j]-x[i+1][j]-x[i-1][j]-x[i][j+1]-x[i][j-1]=f[i][j]$$
$$\Downarrow$$
$$x[i][j]\leftarrow(f[i][j]+x[i+1][j]+x[i-1][j]+x[i][j+1]+x[i][j-1])/4$$

0	-1	0
-1	4	-1
0	-1	0

$\times$



$=$



Solving the Problem

Gauss Seidel Iteration:

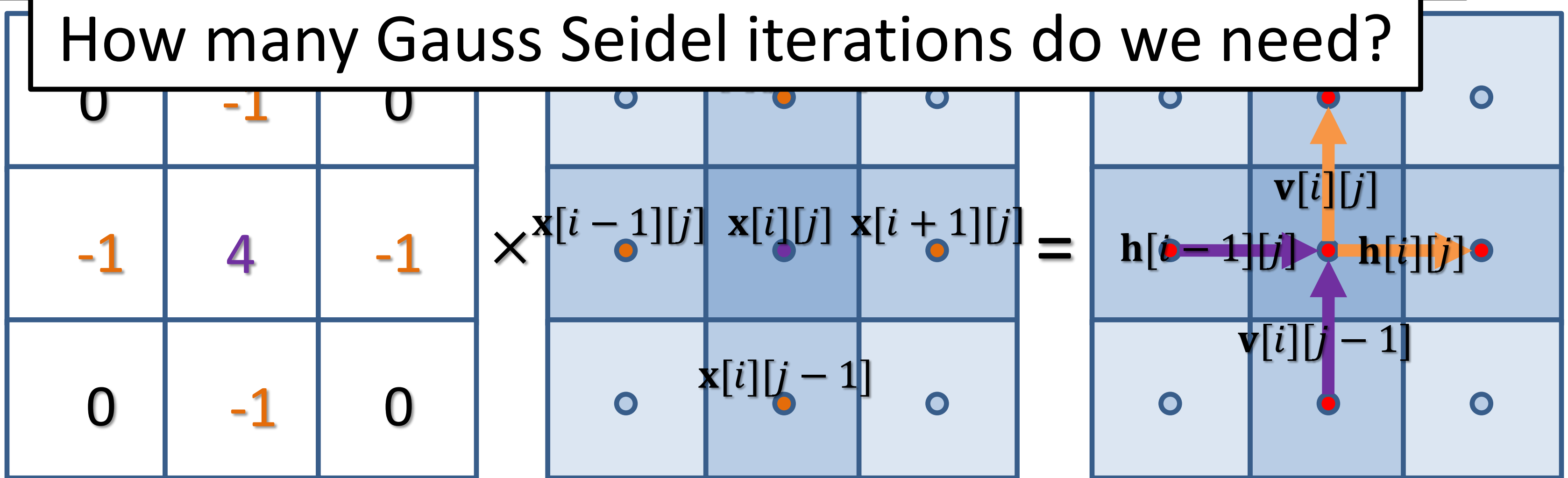
For each pixel (i,j)

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

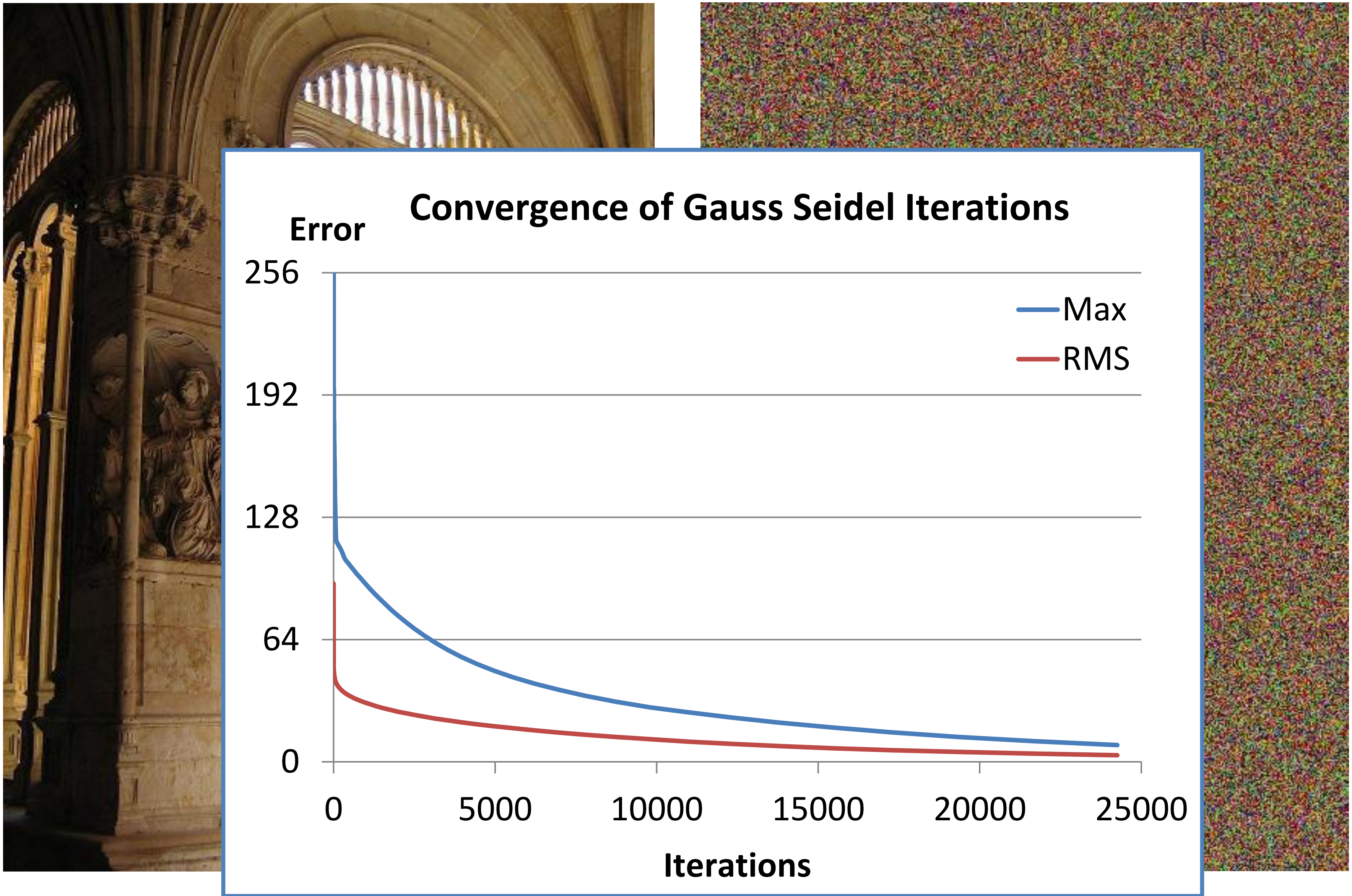
$$4x[i][j]-x[i+1][j]-x[i-1][j]-x[i][j+1]-x[i][j-1]=f[i][j]$$

Similar to the way we solved the radiosity equation

How many Gauss Seidel iterations do we need?



Too Many!

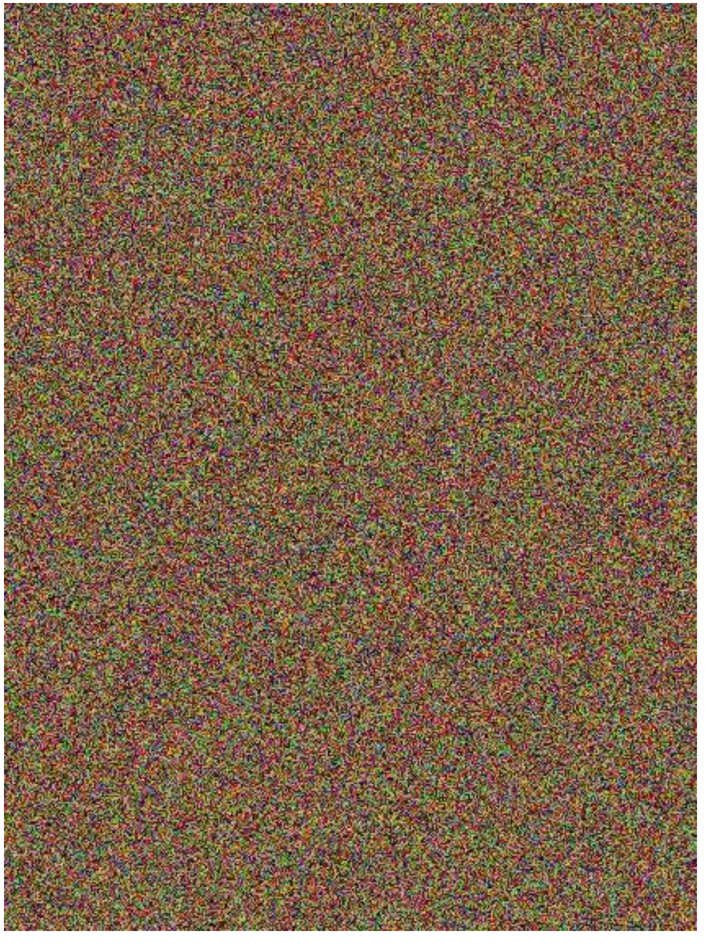

Convergence of Gauss Seidel Iterations
Error
256
192
128
64
0
Max
RMS
0
5000
10000
15000
20000
25000
Iterations

Too Many Gauss Seidel Iterations?

Analyzing the convergence of the Gauss-Seidel:

- ✓ fine details appear quickly
- ✗ low-resolution component resolves slowly

⇒ Solve for the low-resolution component <u>efficiently</u> using a down-sampled version of the image.



Solving the Problem

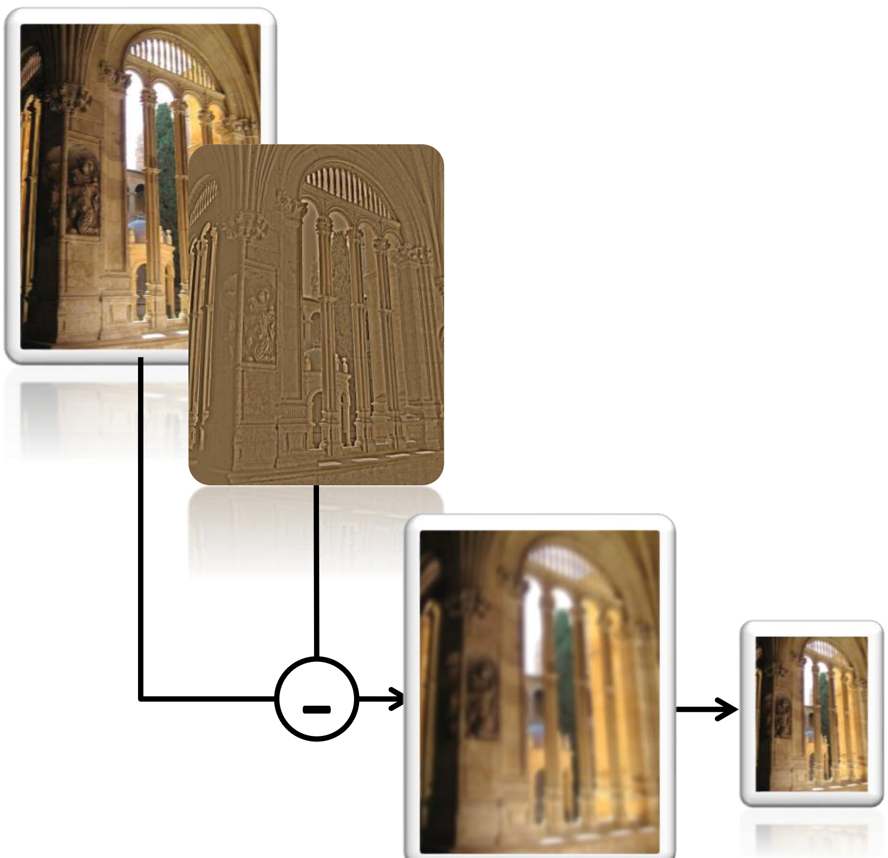
Multi-grid Solvers

1. Perform a few G.S. iterations at the high-resolution

Solving the Problem

Multi-grid Solvers

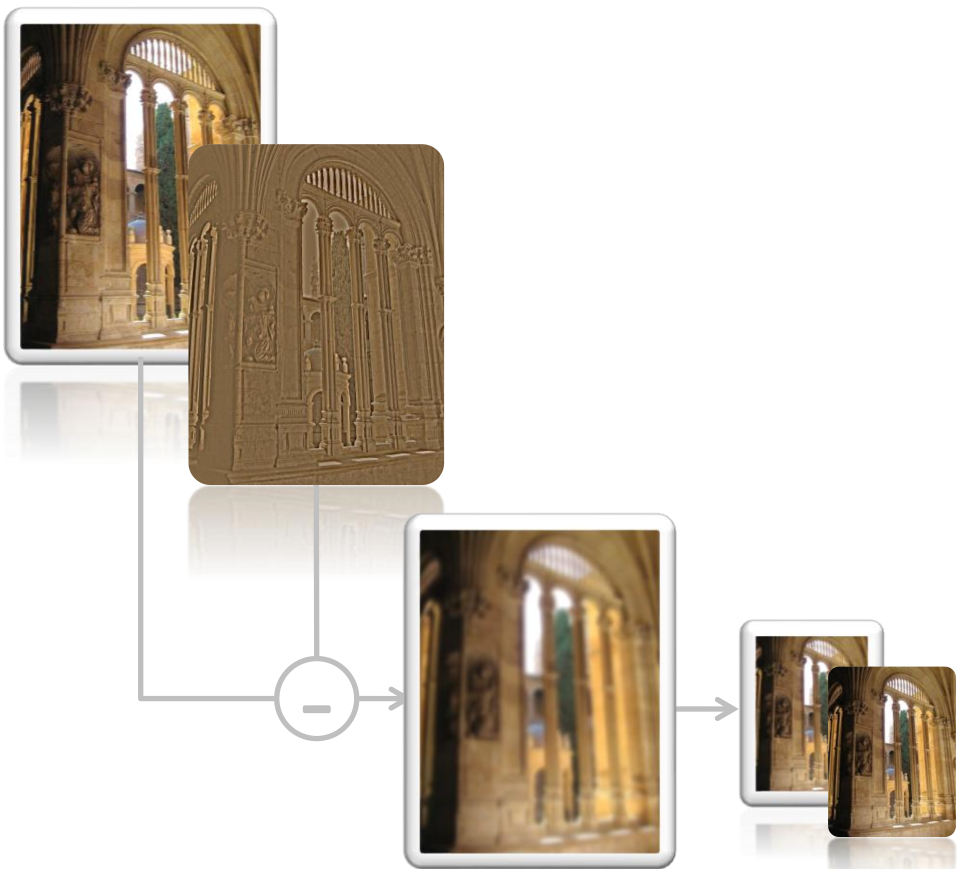
2. Compute the residual and down-sample



Solving the Problem

Multi-grid Solvers

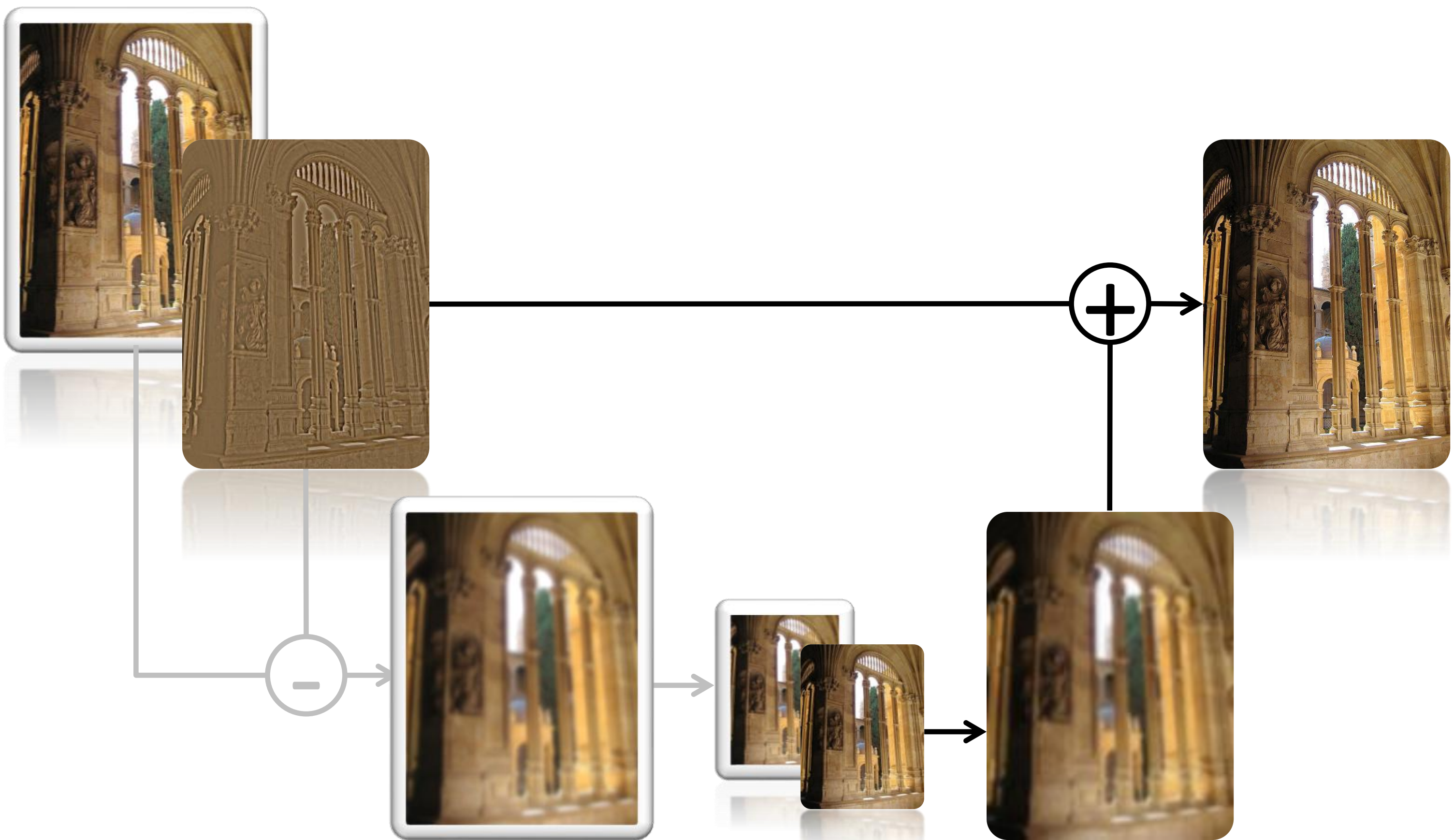
3. Solve the low-resolution, down-sampled residual



Solving the Problem

Multi-grid Solvers

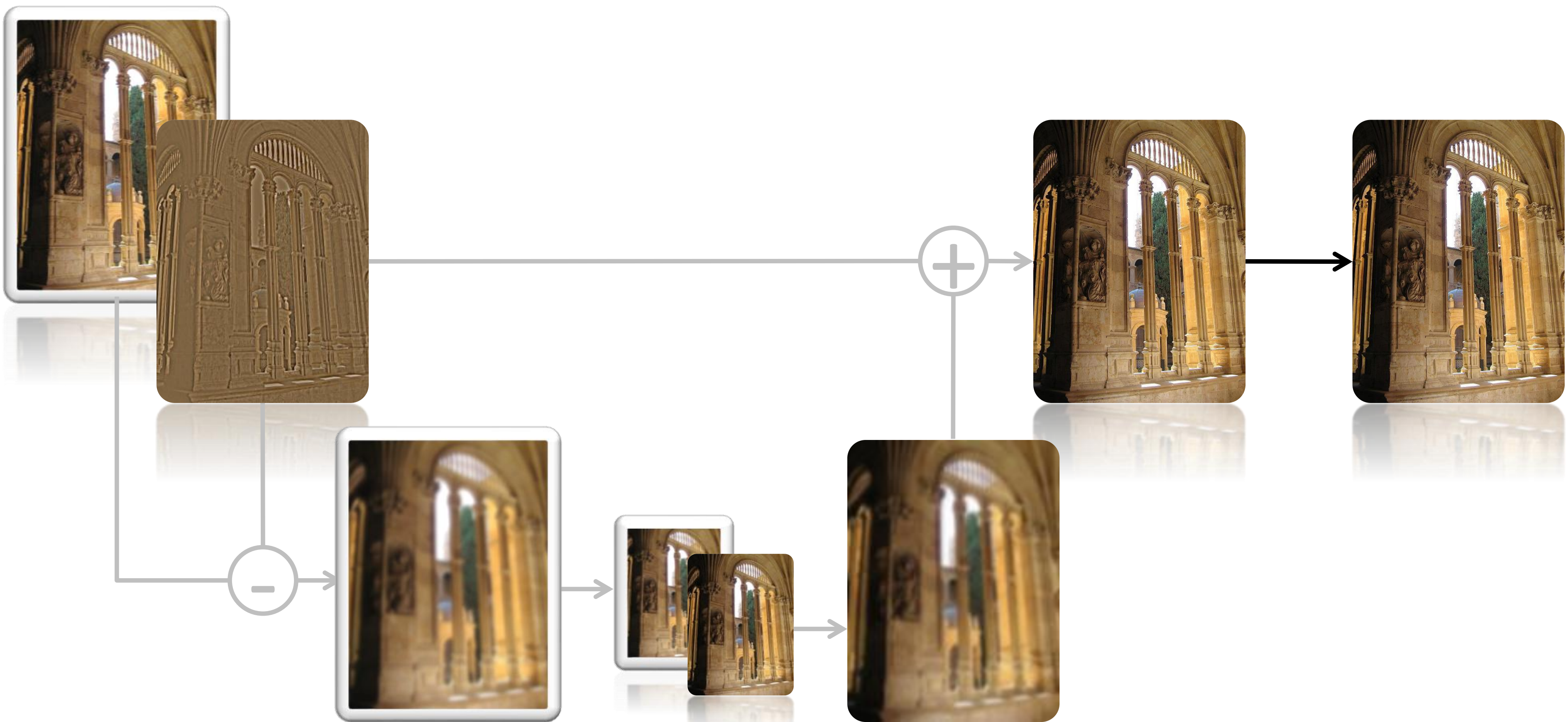
4. Up-sample and add to the estimated high-resolution solution



Solving the Problem

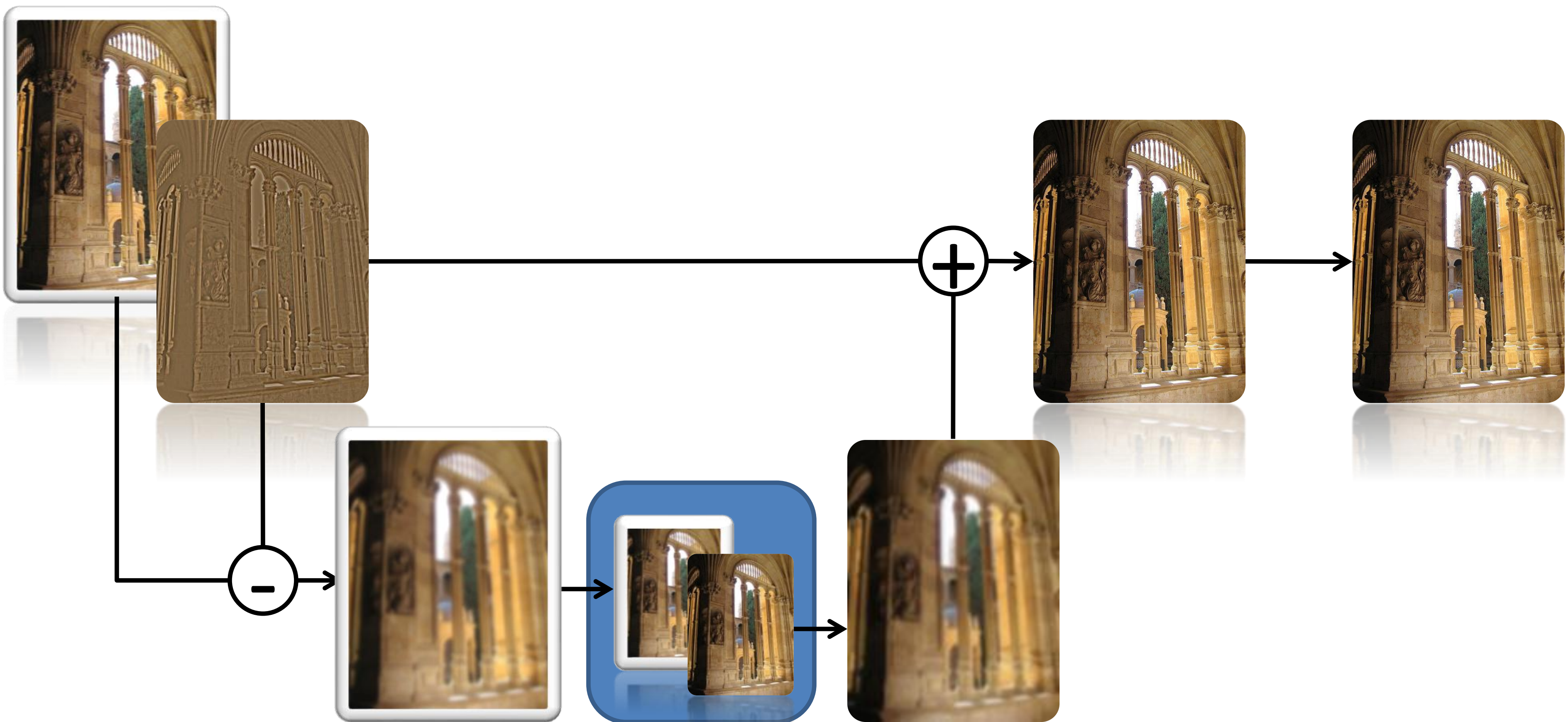
Multi-grid Solvers

5. Perform some more G.S. iterations at the high-resolution



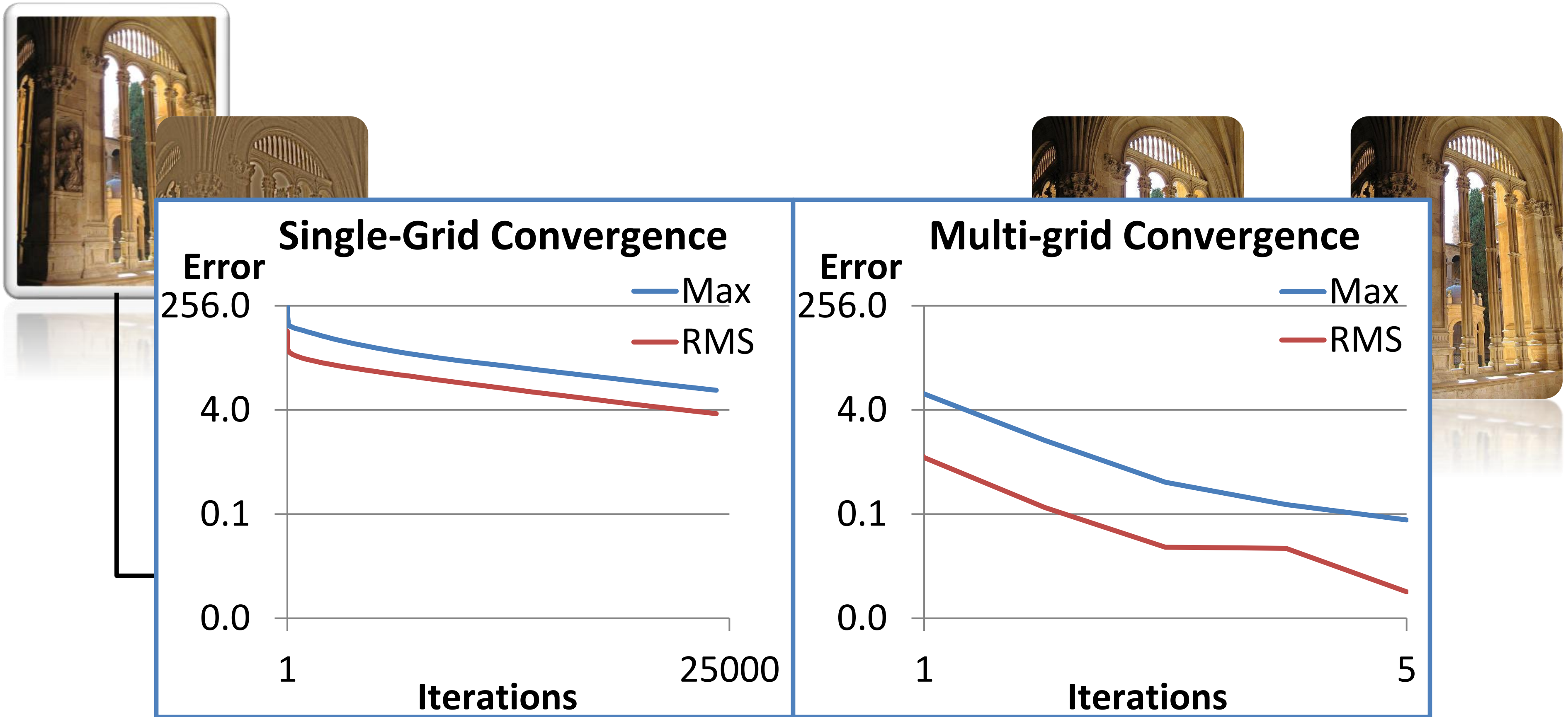
Solving the Problem

To solve the lower-resolution problem, the process can be recursed.



Solving the Problem

To solve the lower-resolution problem, the process can be recursed.



Outline

Motivation

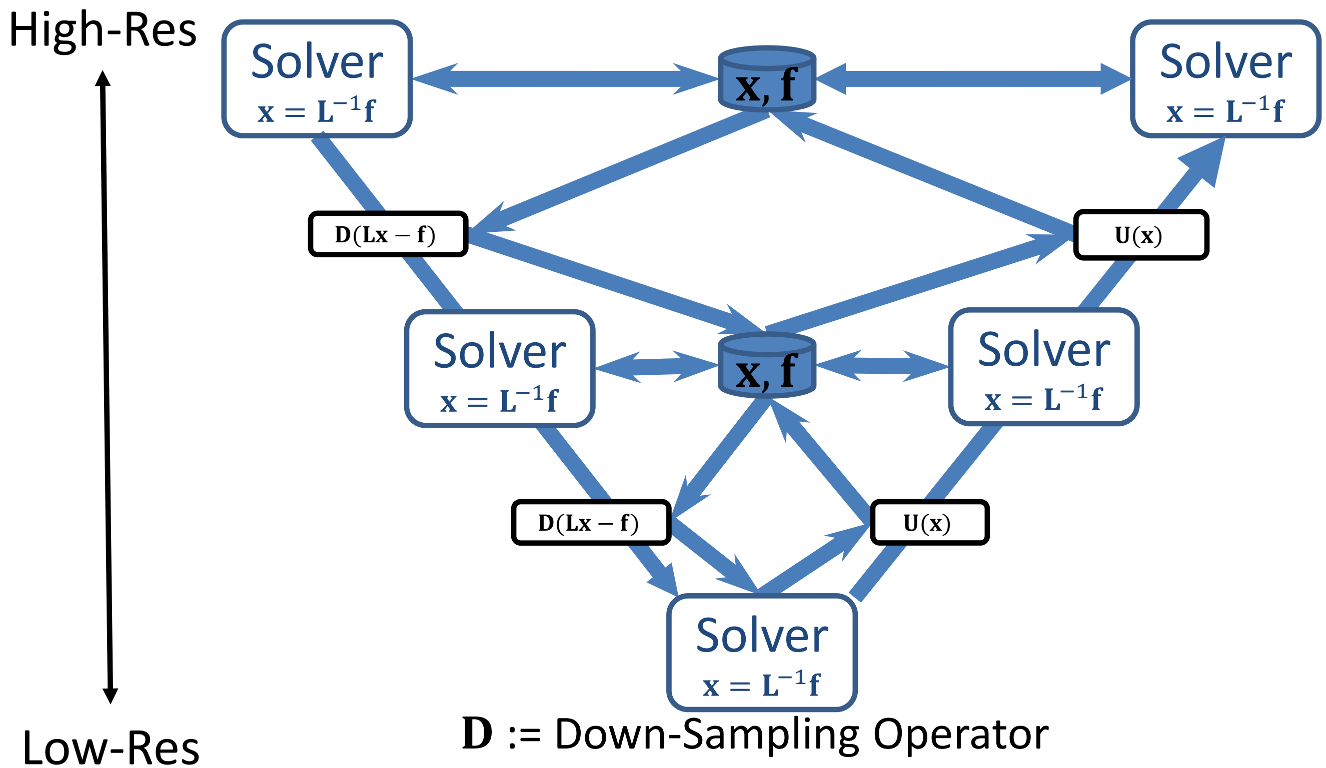
What’s the problem?

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.



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

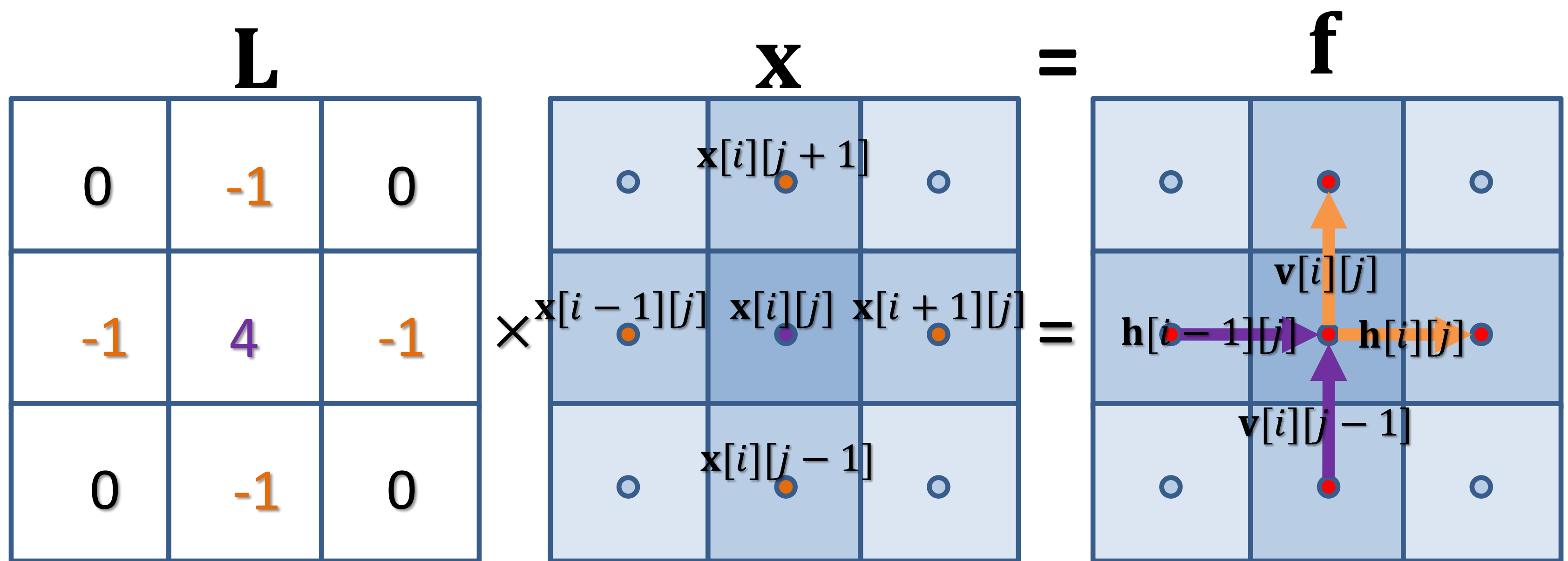
$\mathbf{U}$:= Up-Sampling Operator

The Big Problem

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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:

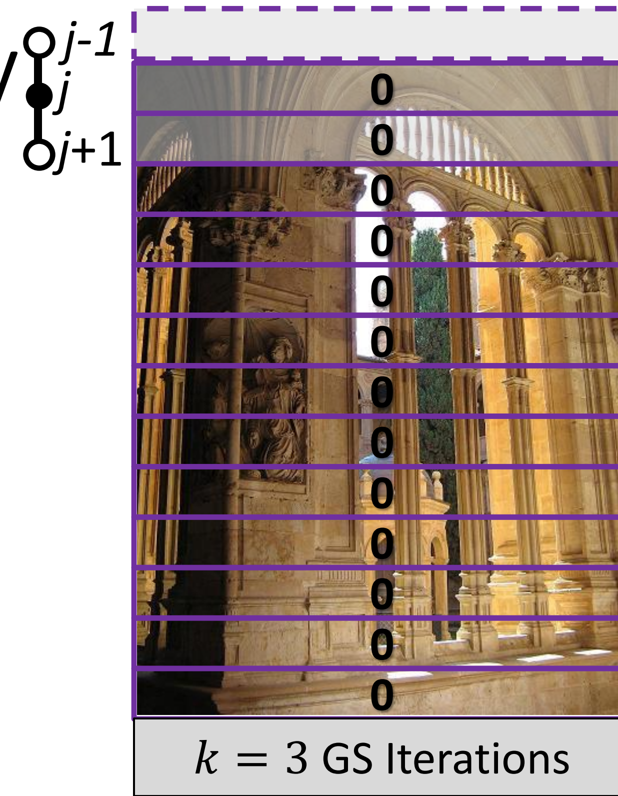


Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.

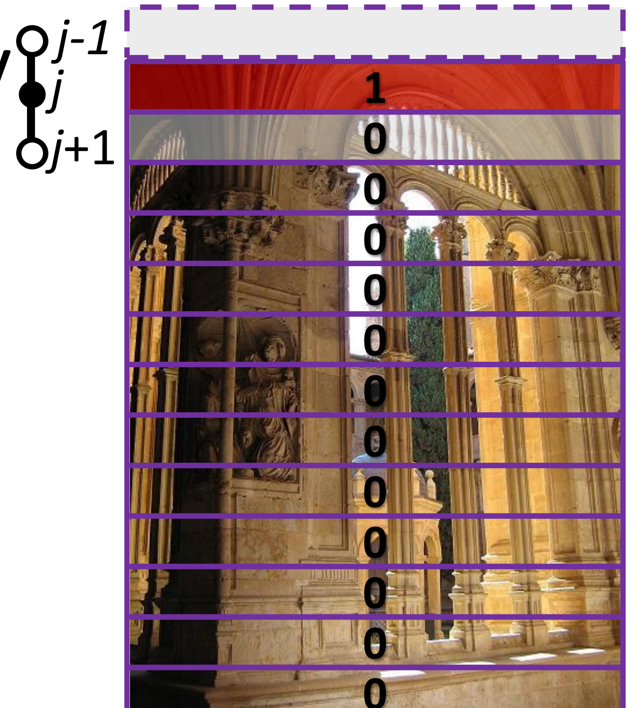


Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



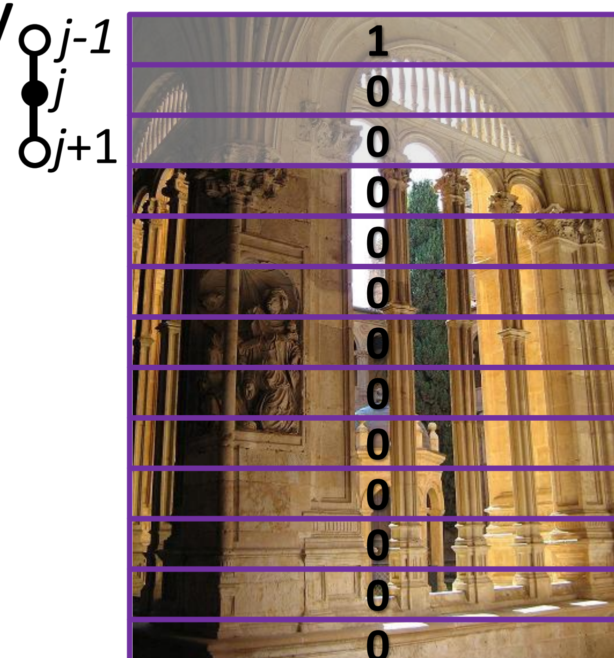
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



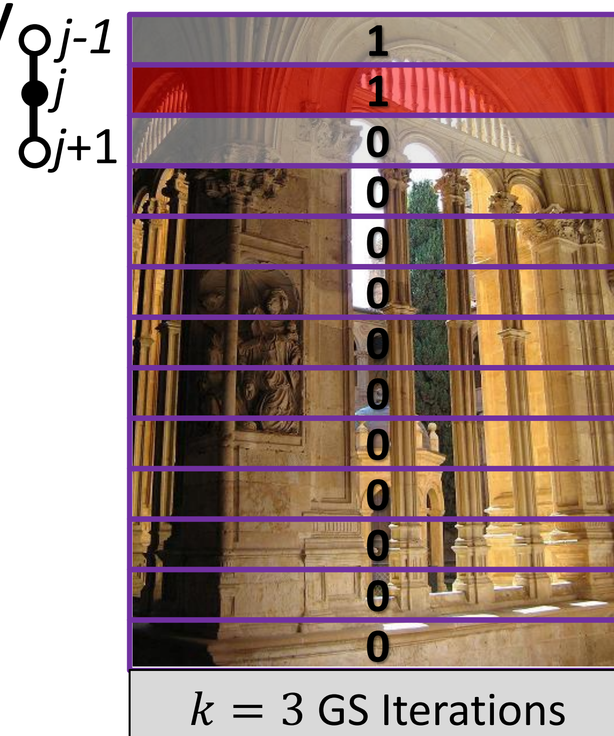
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.

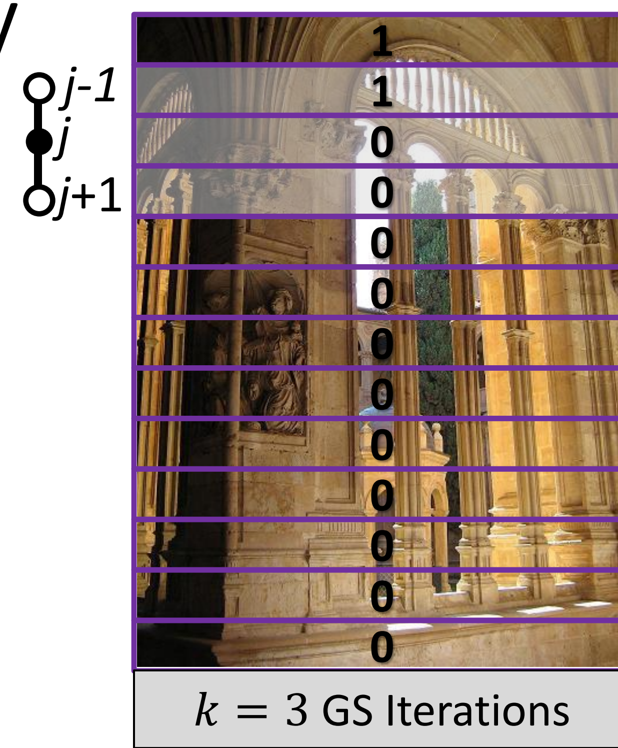


Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.

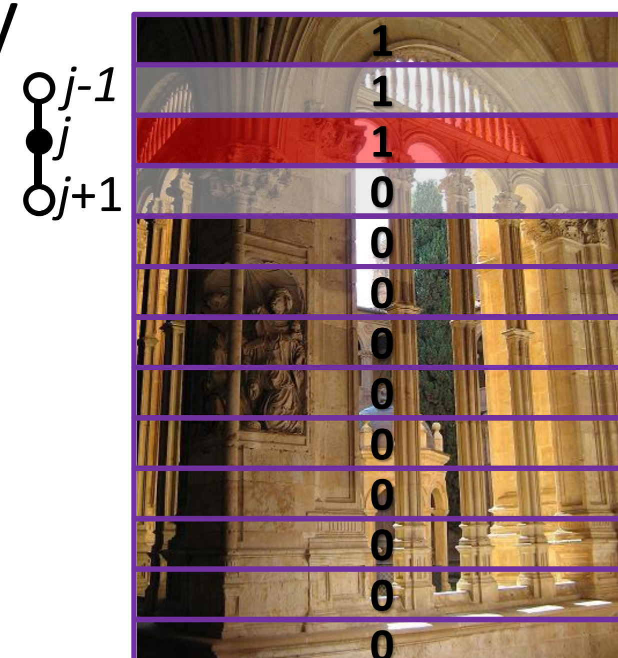


Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



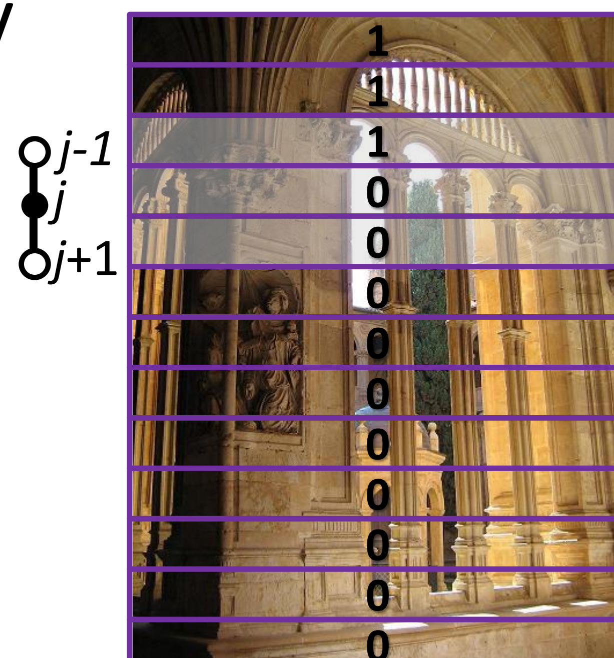
k = 3 GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



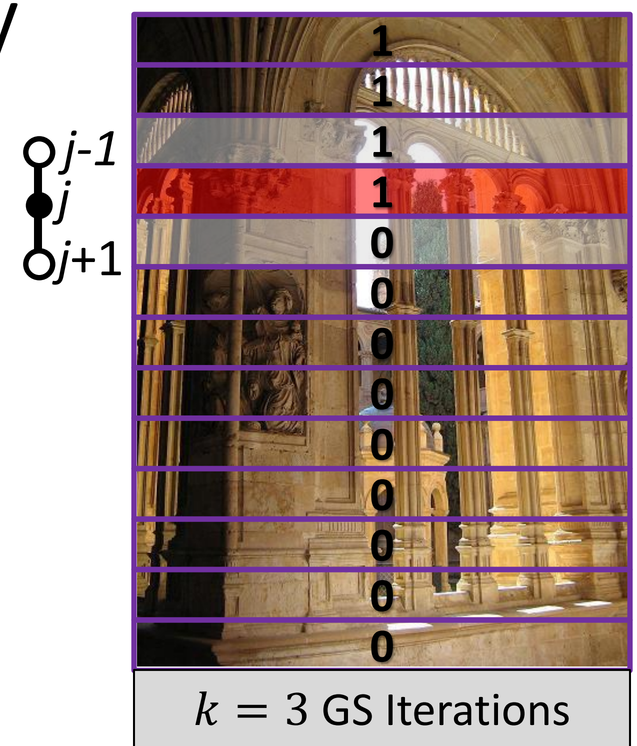
k = 3 GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.

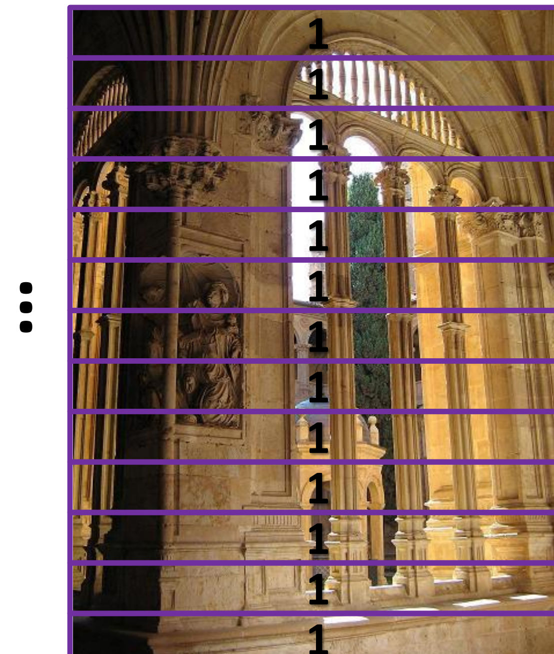


Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



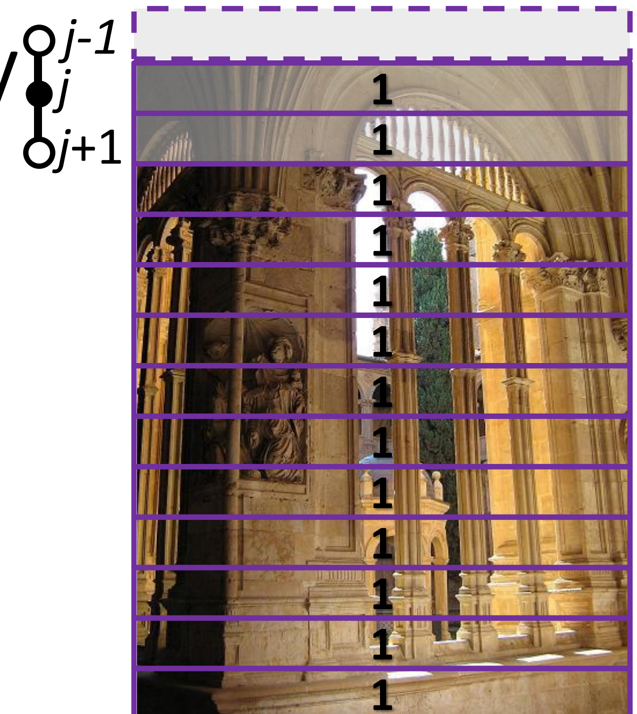
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



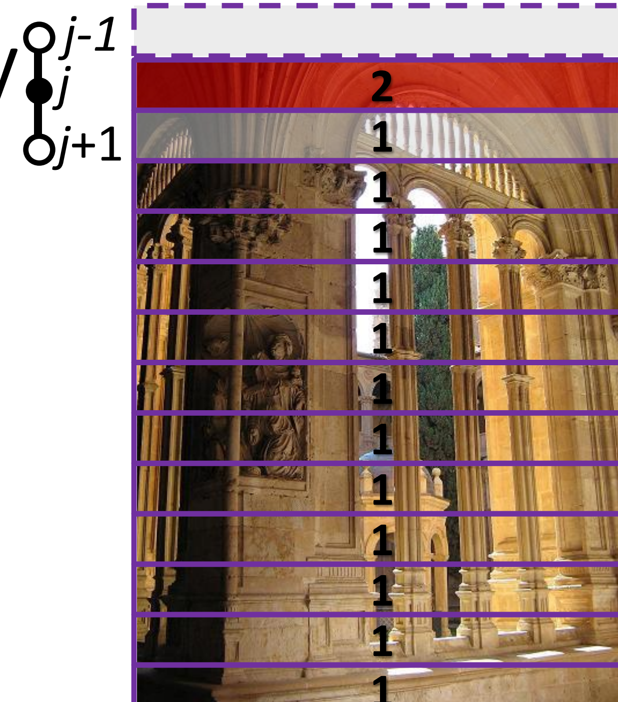
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



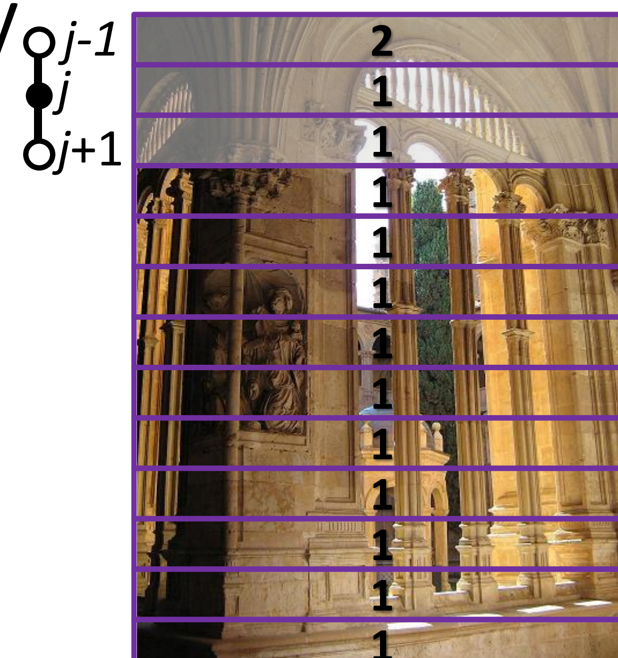
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.



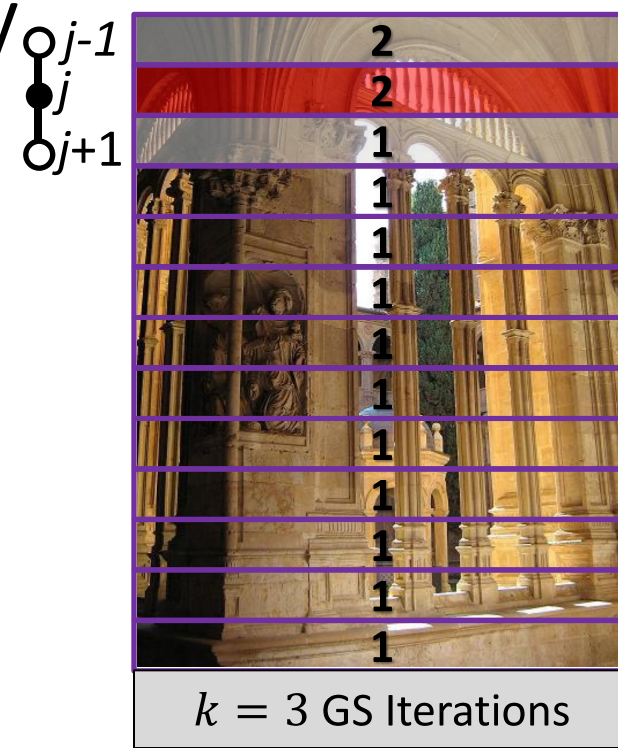
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.

⋮

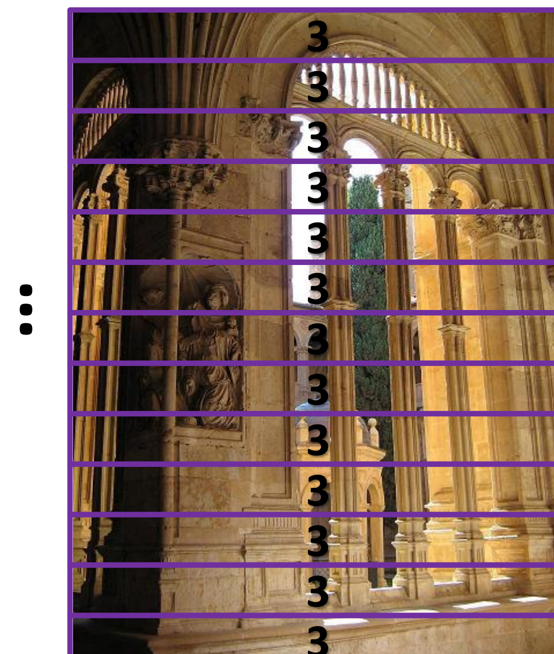
$k = 3$ GS Iterations

Streaming Multiple GS Iterations

$$x[i][j] \leftarrow (f[i][j] + x[i+1][j] + x[i-1][j] + x[i][j+1] + x[i][j-1])/4$$

✓ 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.

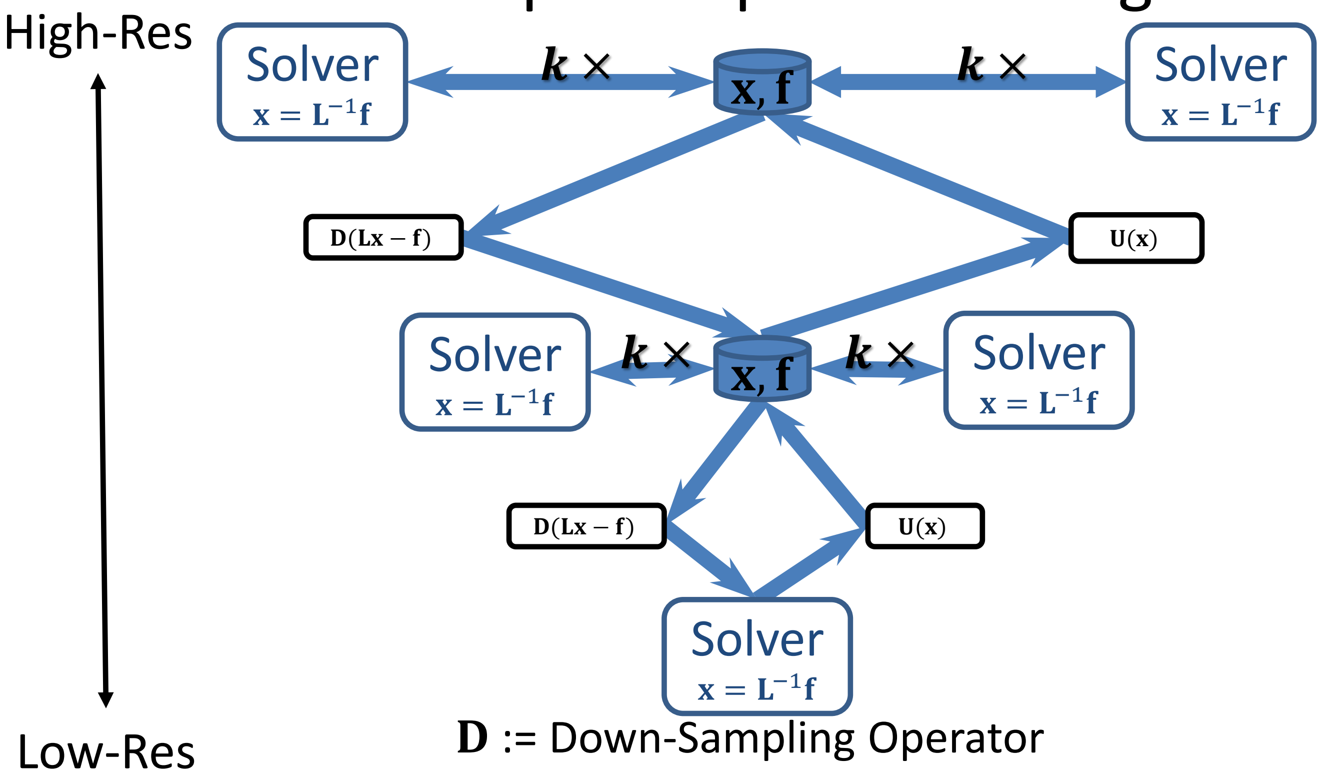


$k = 3$ GS Iterations

The Big Problem

Since we update all the pixels before starting the next Gauss-Seidel iteration:

- ✖ k GS iterations require k passes through the data.

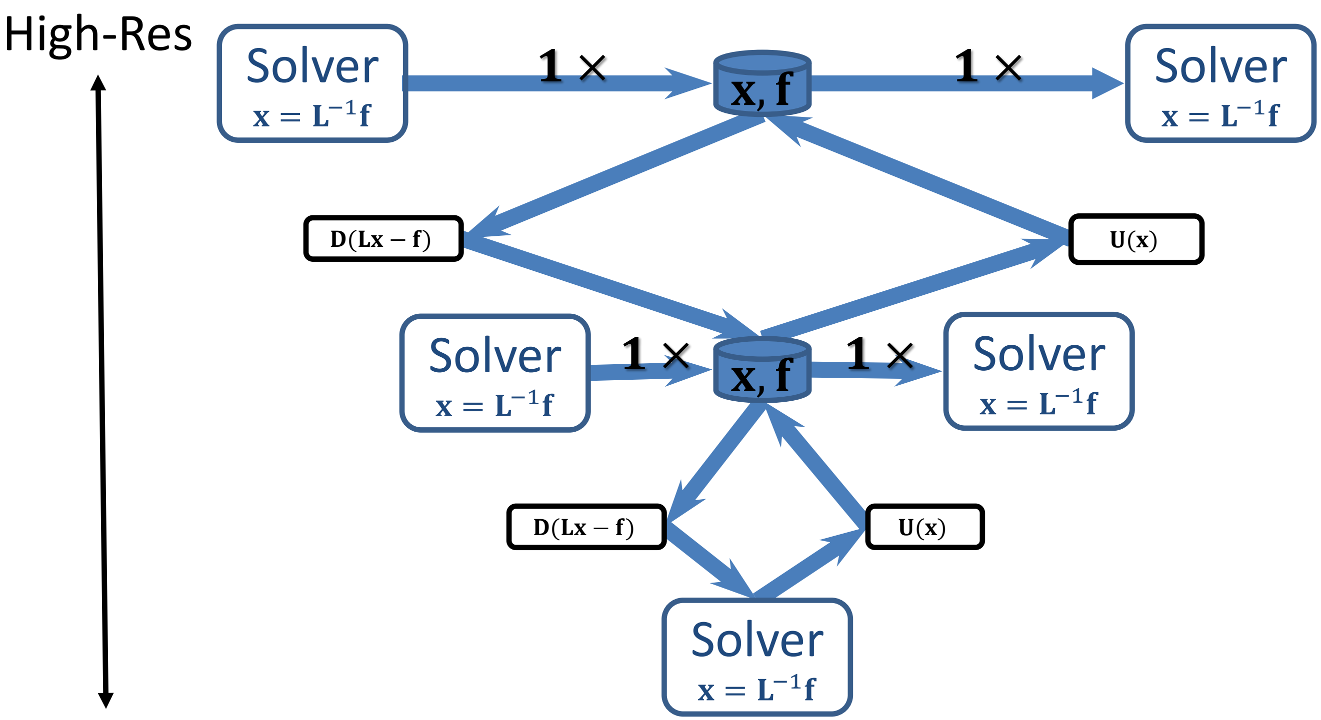


$\mathbf{D}$:= Down-Sampling Operator

$\mathbf{U}$:= Up-Sampling Operator

Streaming Multiple GS Iterations

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



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

$\mathbf{U}$:= Up-Sampling Operator

Streaming Multiple GS Iterations

To perform k GS iterations in one pass:

- Keep a window of $k+2$ rows in memory while sweeping through the pixels.

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

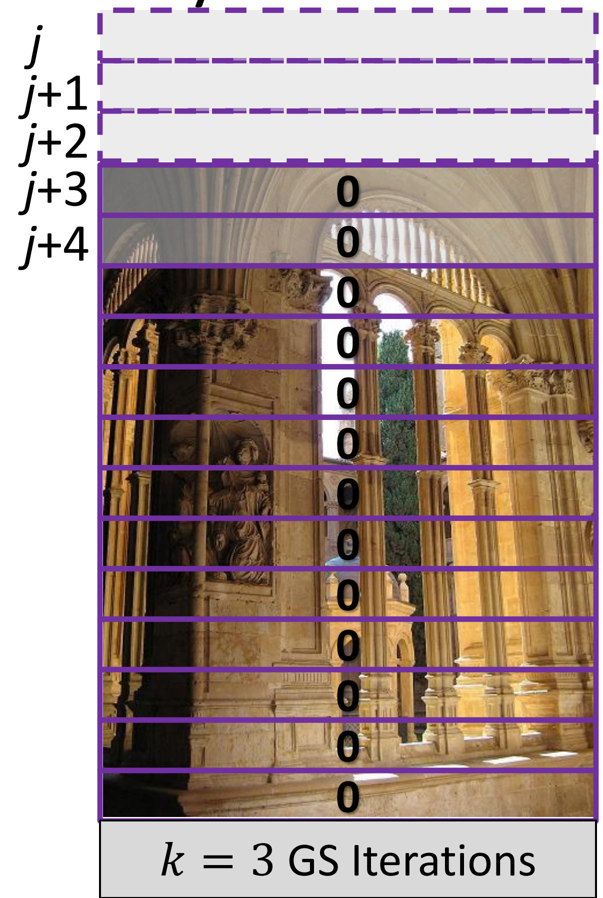
Do

1. For($i=k$; $i>0$; i--)

   Update pixels in in row $j+i$

2. Increment j

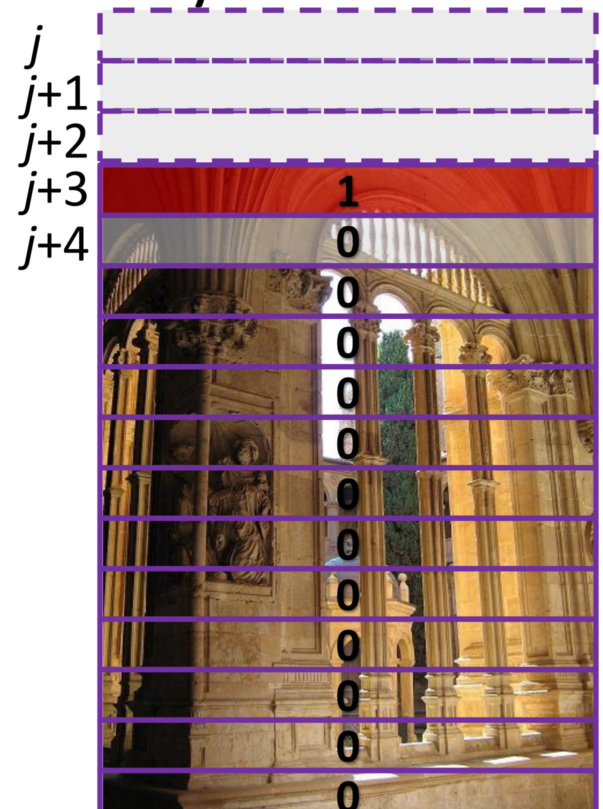
While j<height



Streaming Multiple GS Iterations

To perform k GS iterations in one pass:

– Keep a window of $k+2$ rows in memory while sweeping through the pixels.



$k = 3$ GS Iterations

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

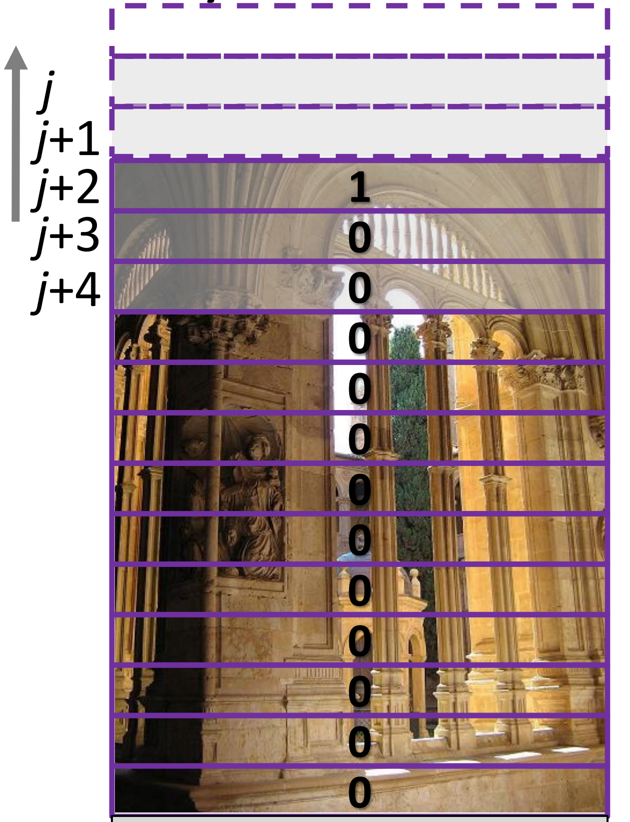
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

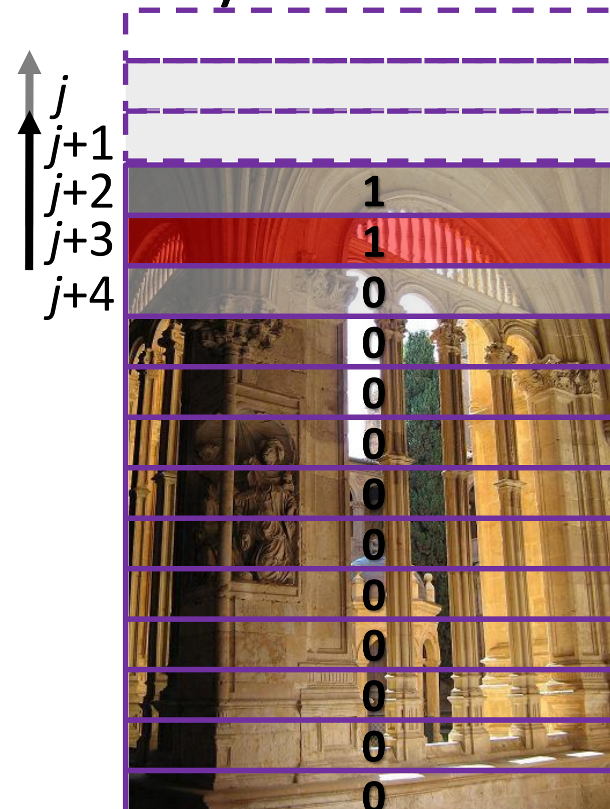
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

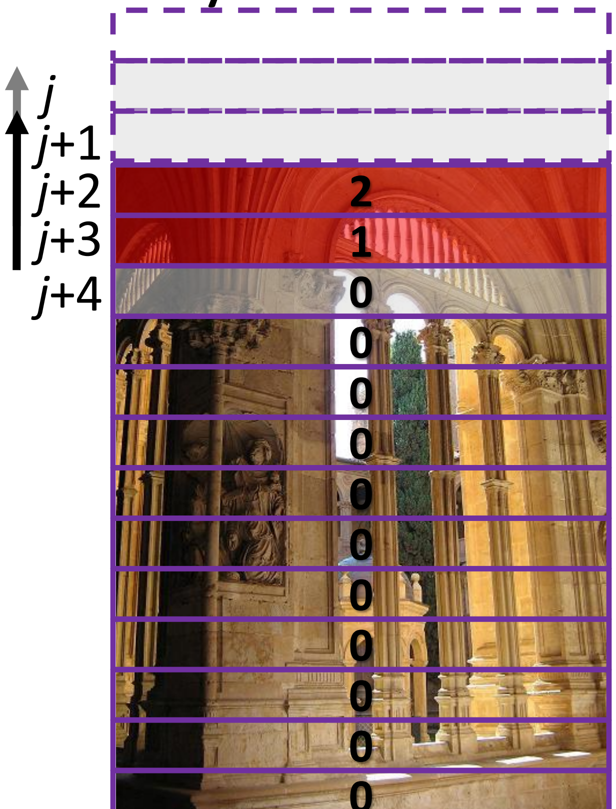
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

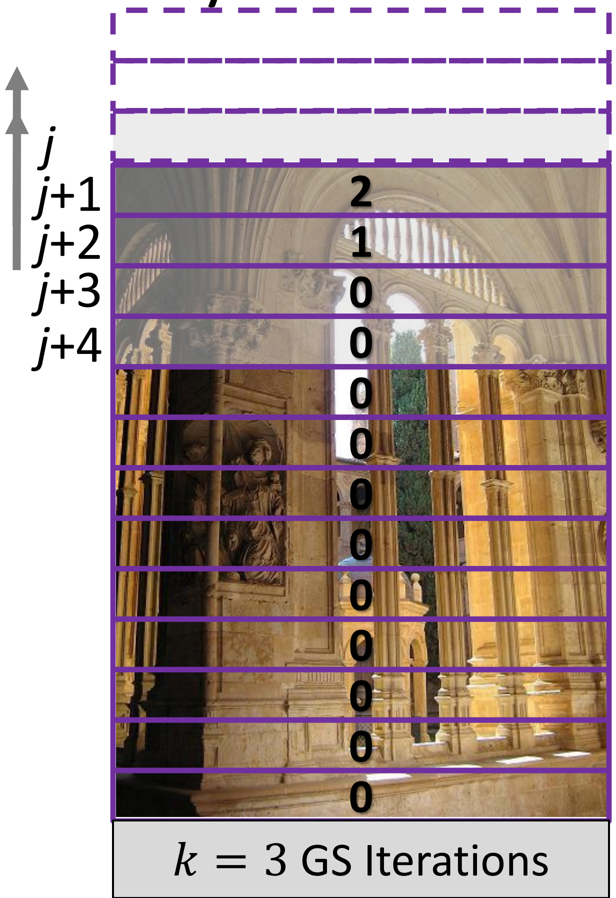
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

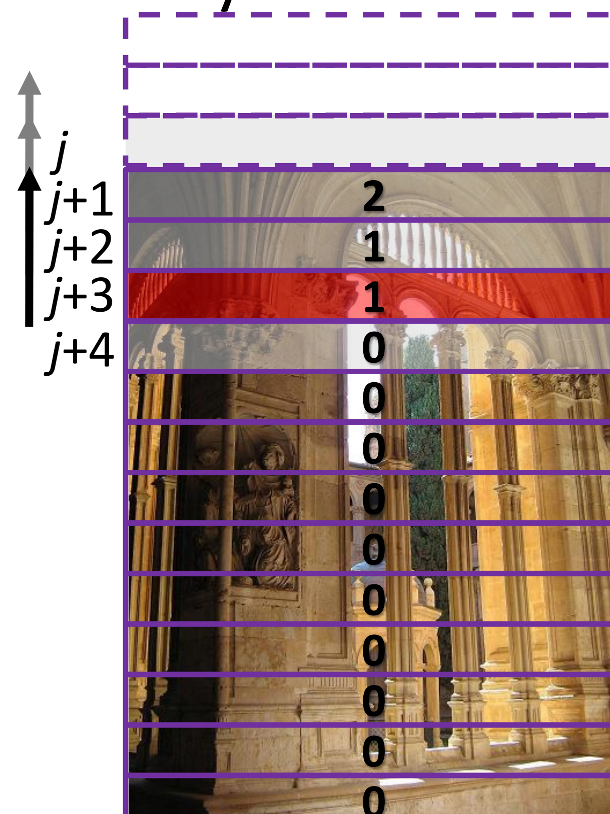
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

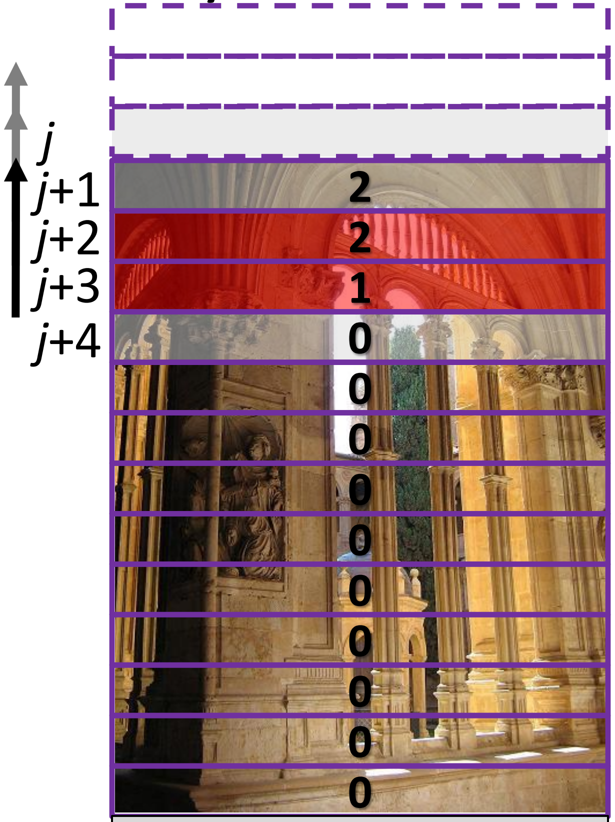
Do

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:

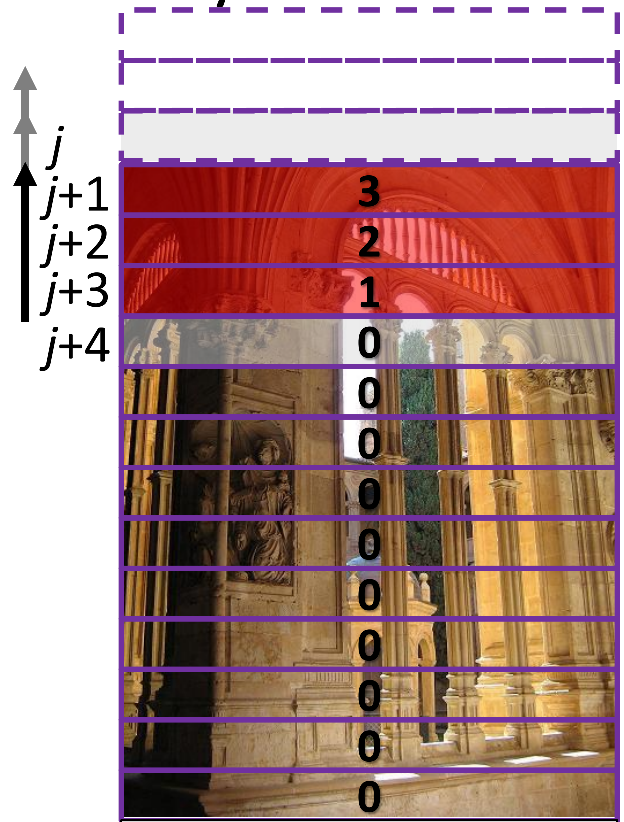
– Keep a window of $k + 2$ rows in memory while sweeping through the pixels.

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

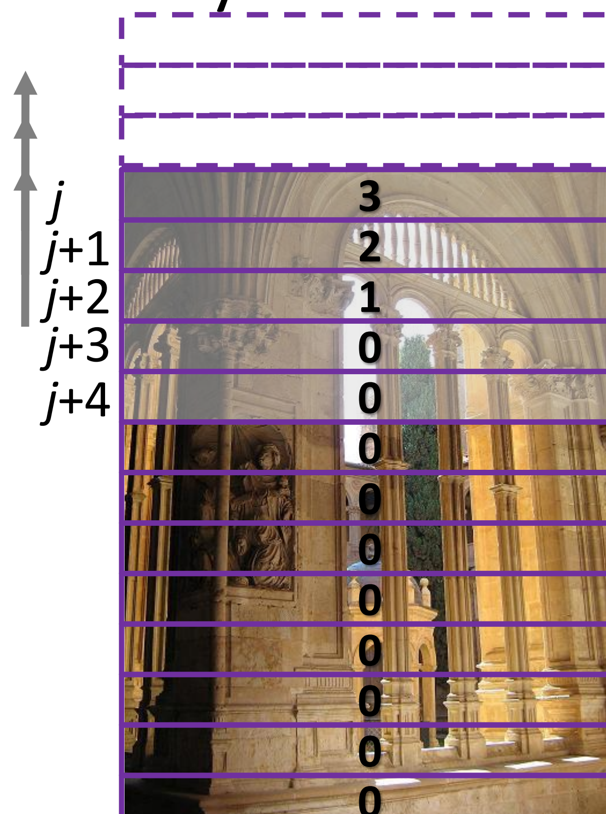
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

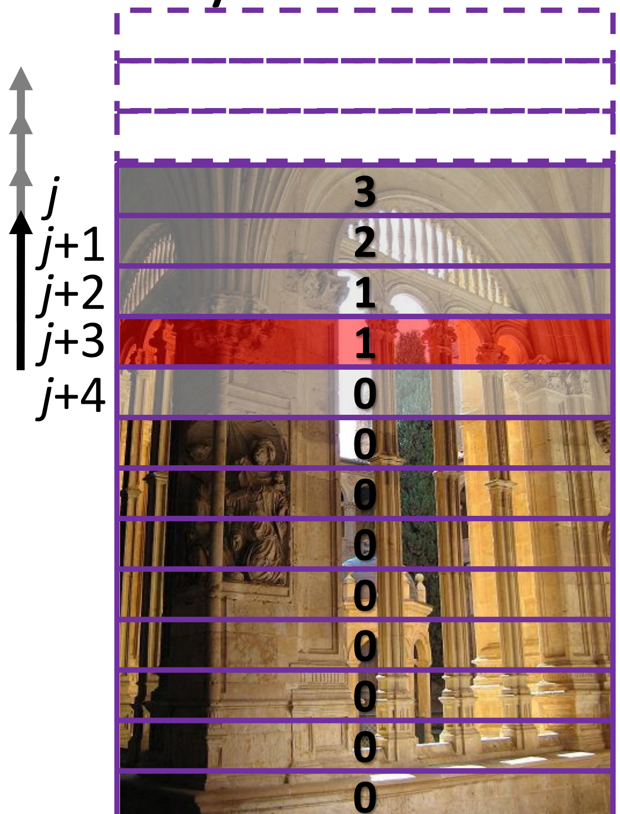
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

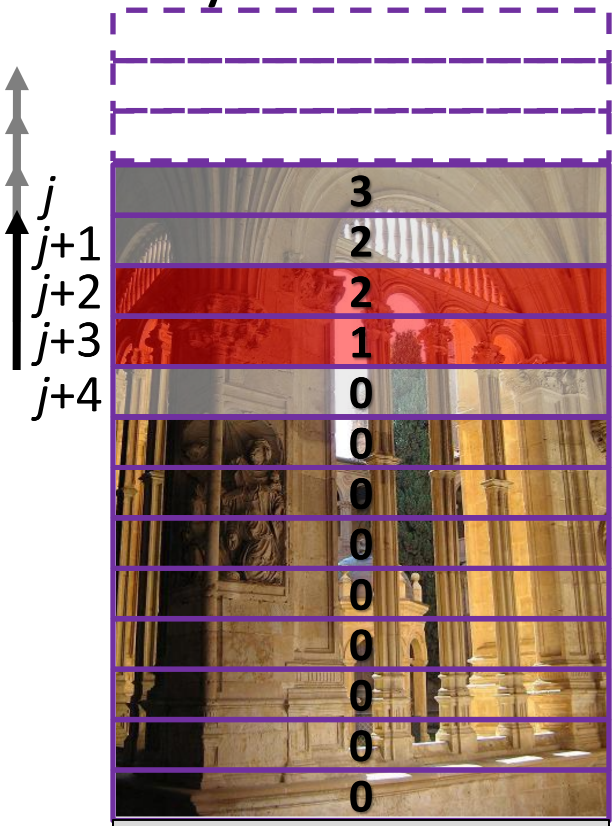
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

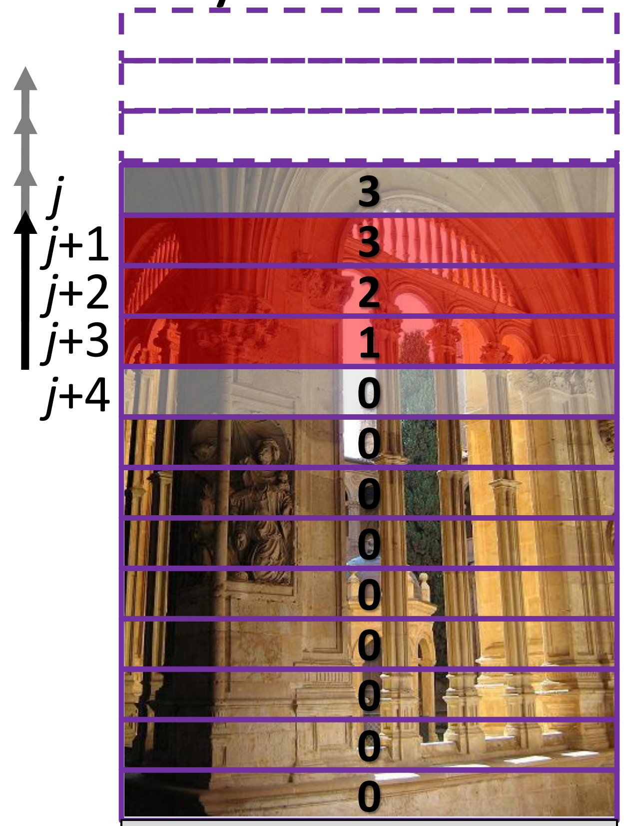
Do

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 while sweeping through the pixels.

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

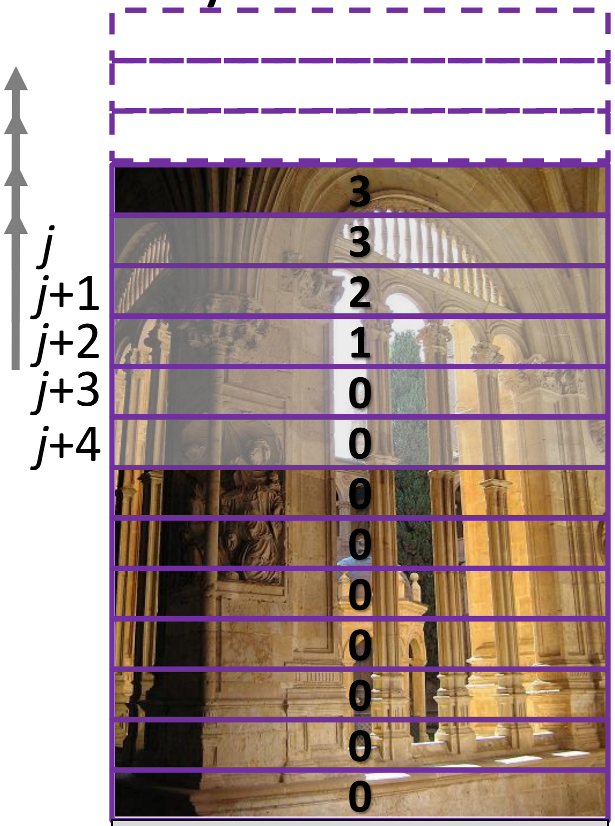
Do

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:

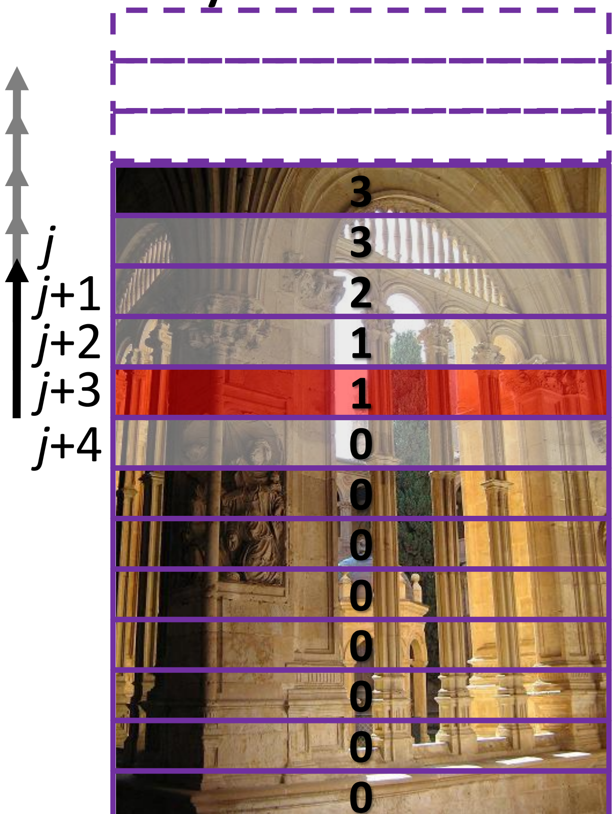
– Keep a window of $k + 2$ rows in memory while sweeping through the pixels.

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:

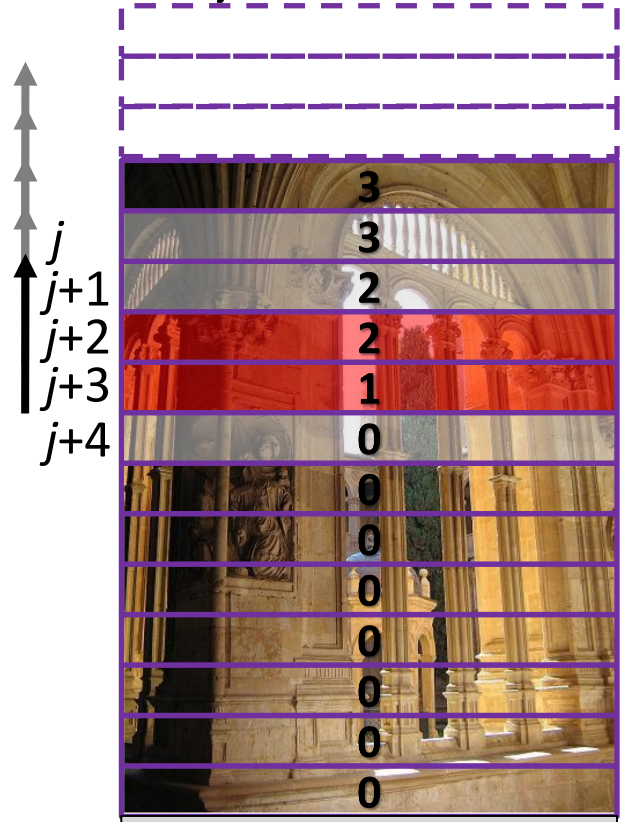
– Keep a window of $k + 2$ rows in memory while sweeping through the pixels.

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:

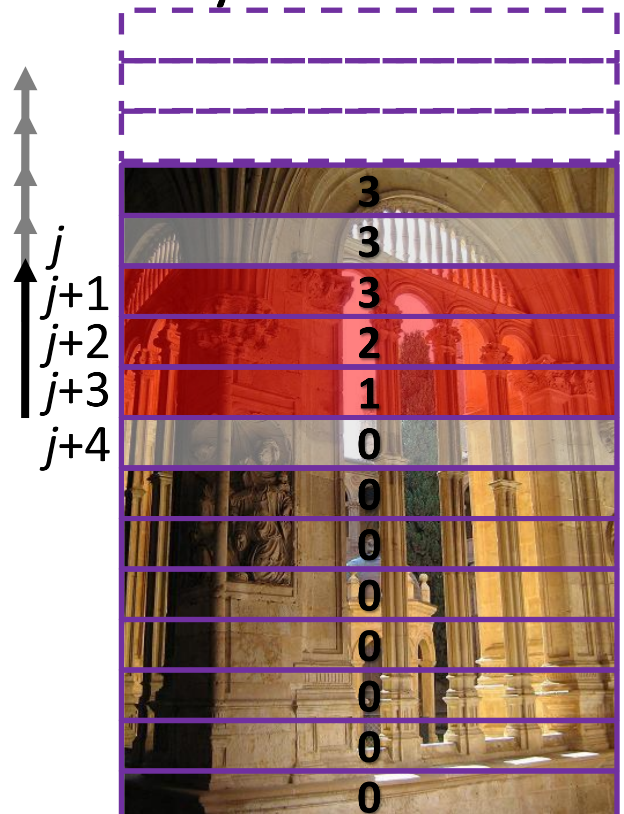
– Keep a window of $k + 2$ rows in memory while sweeping through the pixels.

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

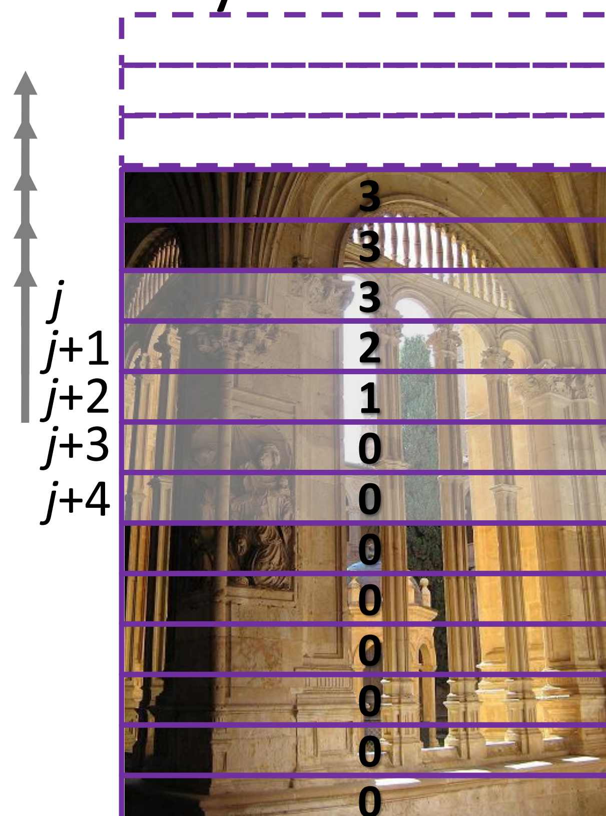
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

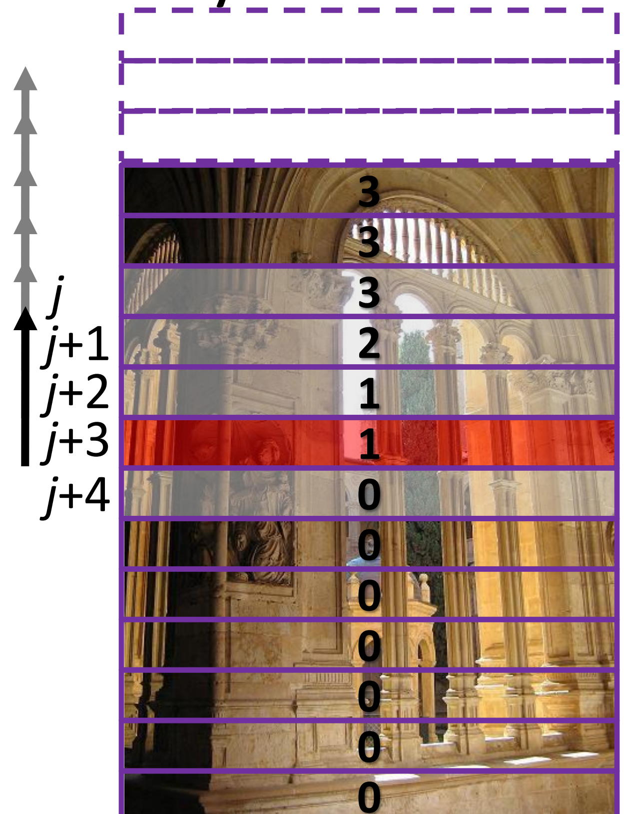
Do

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 while sweeping through the pixels.

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

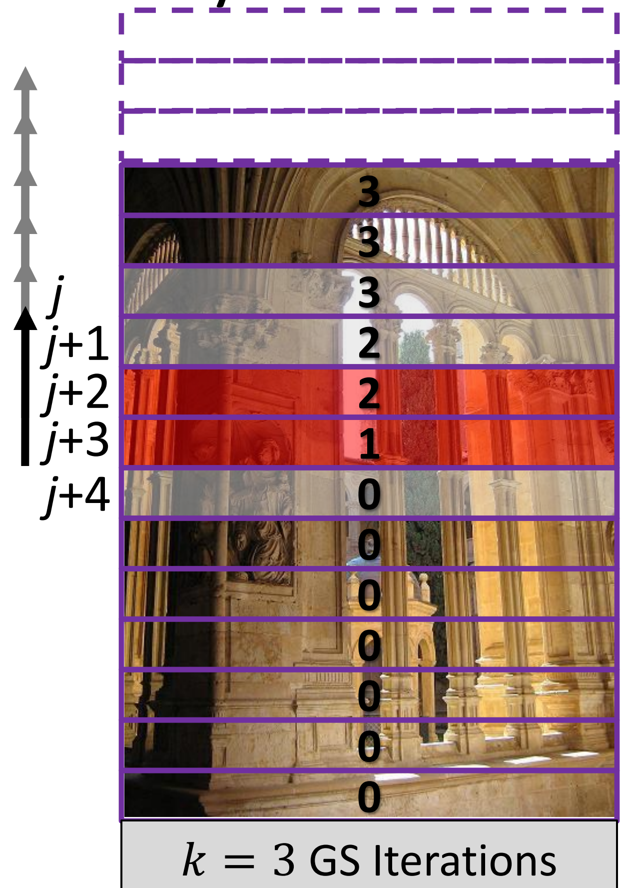
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

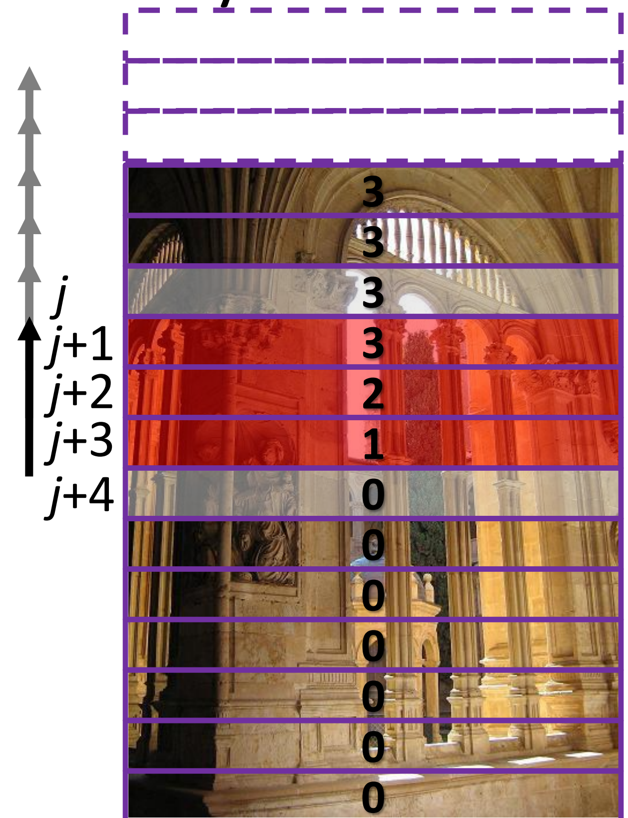
Do

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 while sweeping through the pixels.

Initialize start of the window at row j=-k

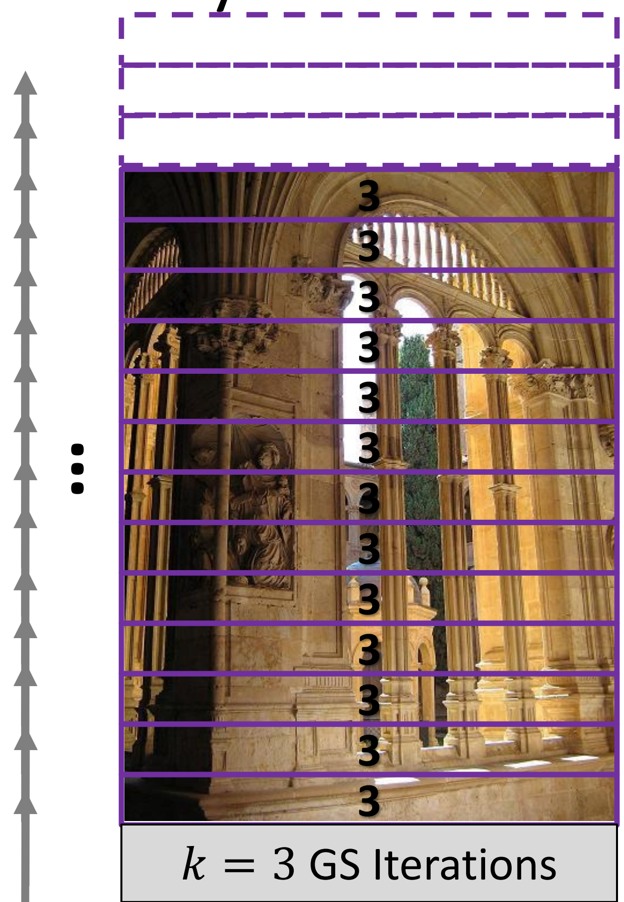
Do

1. For(i=k ; i > 0 ; i--)

   Update pixels in in row j+i

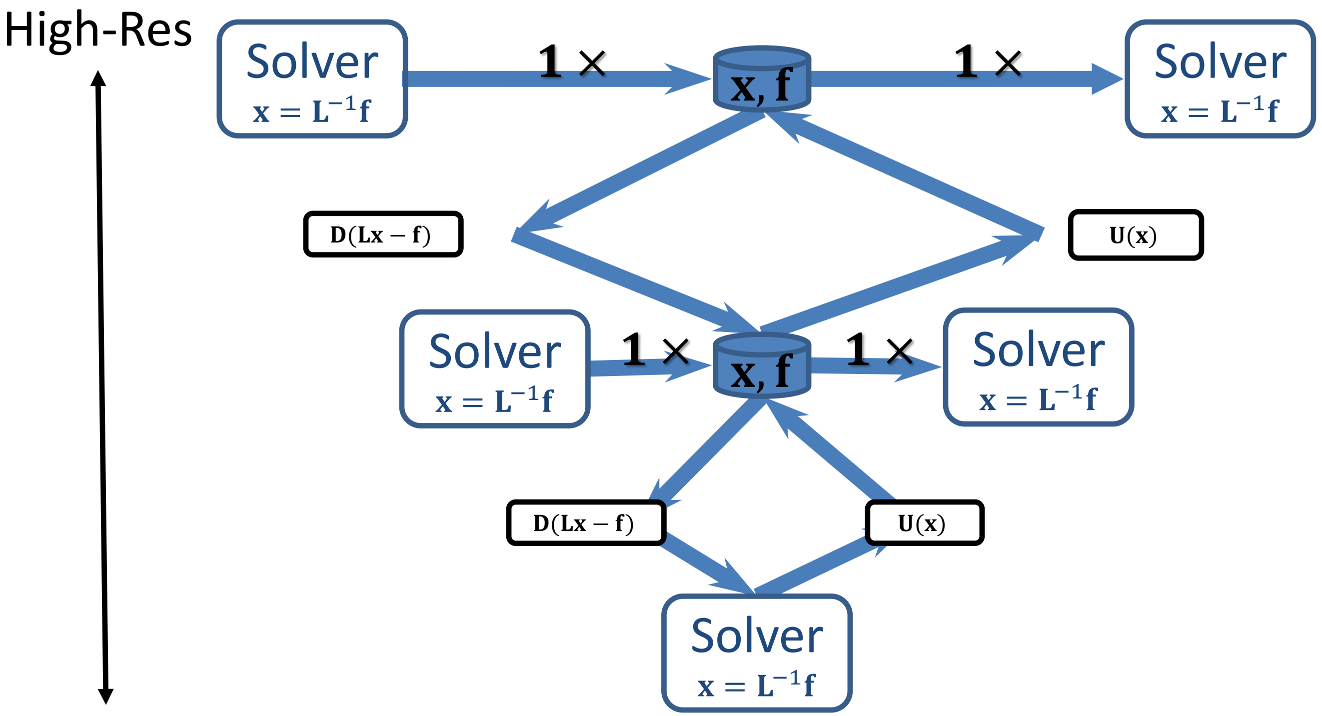
2. Increment j

While j<height



Streaming Multiple GS Iterations

With more careful scheduling, can stream directly between levels.

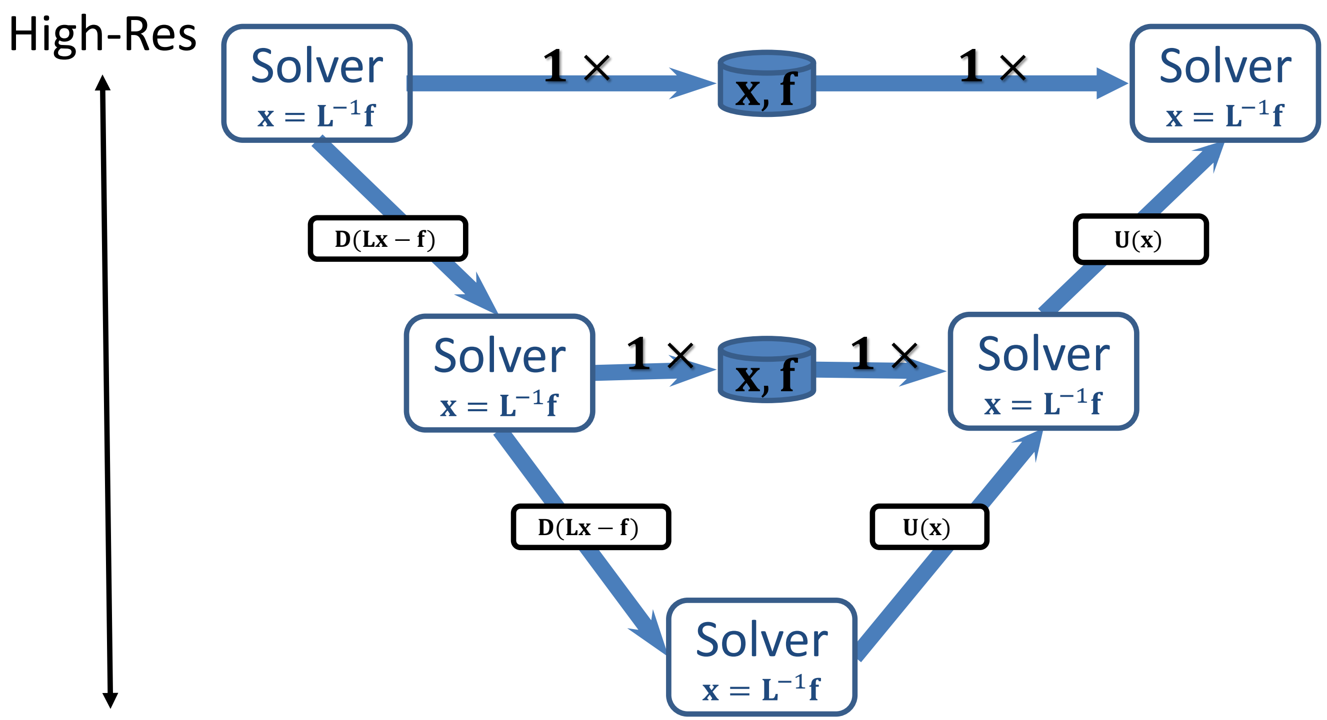


$\mathbf{D}$:= Down-Sampling Operator

$\mathbf{U}$:= Up-Sampling Operator

Streaming Multiple GS Iterations

With more careful scheduling, can stream directly between levels.

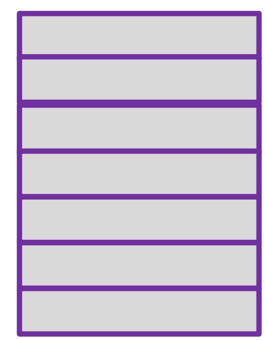


$\mathbf{D}$:= Down-Sampling Operator
$\mathbf{U}$:= Up-Sampling Operator

Streaming Multiple GS Iterations

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

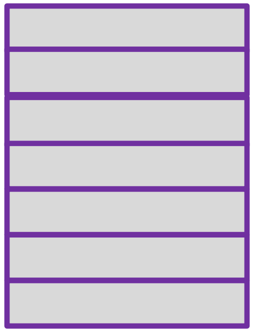
- Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start processing there as well.



Streaming Multiple GS Iterations

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

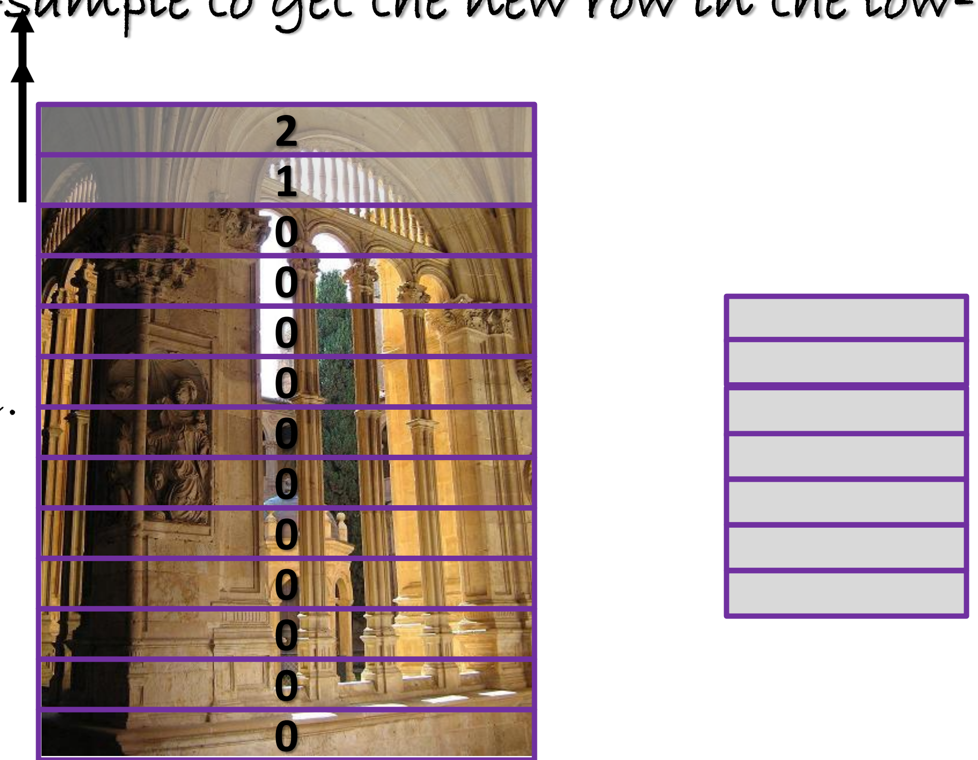
- **Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.**
- Once we add a new row to the low-resolution system, we can start processing there as well.



Streaming Multiple GS Iterations

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

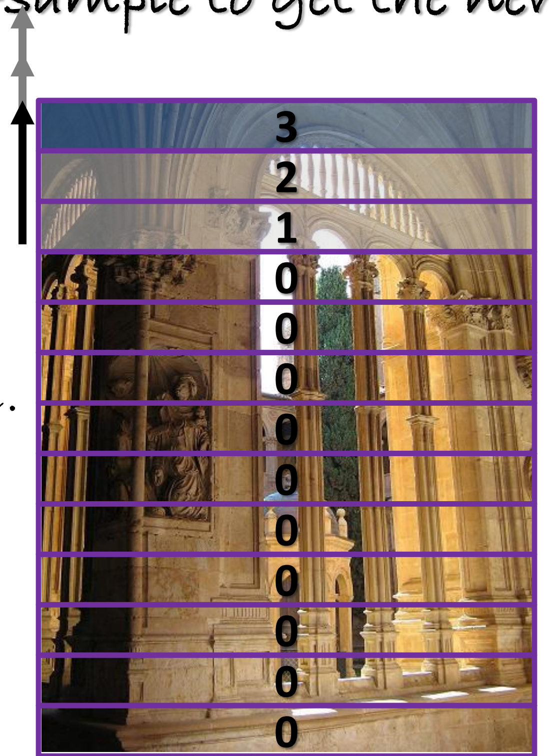
- **Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.**
- Once we add a new row to the low-resolution system, we can start processing there as well.

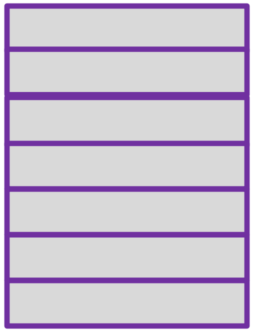


Streaming Multiple GS Iterations

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

- **Once we have processed two high-resolution rows, we can get the residual and 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 processing there as well.

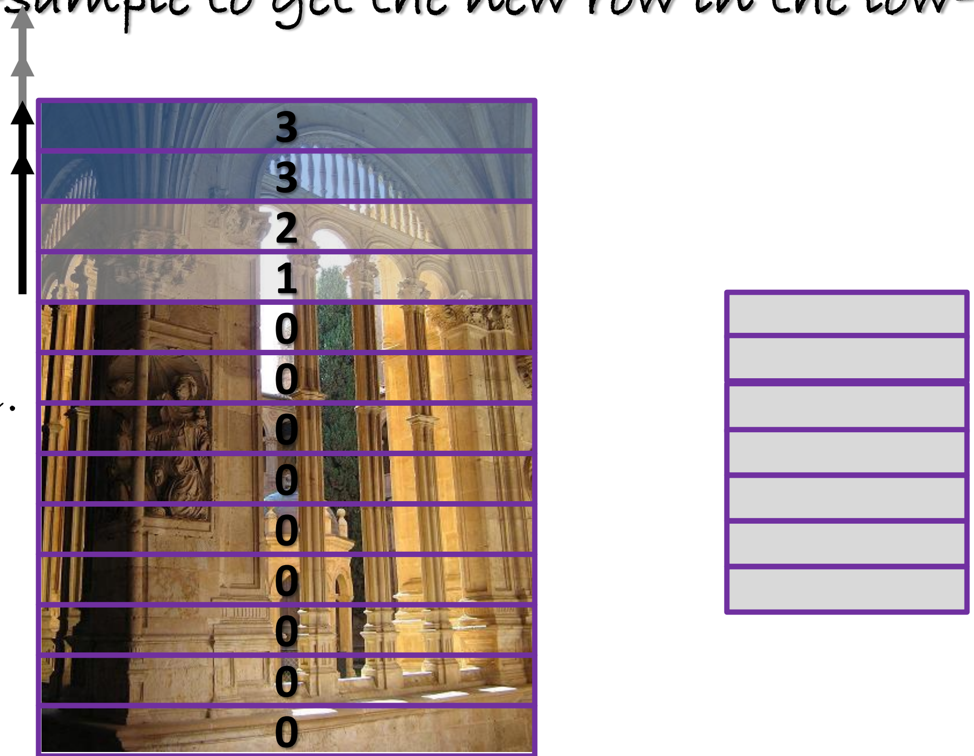




Streaming Multiple GS Iterations

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

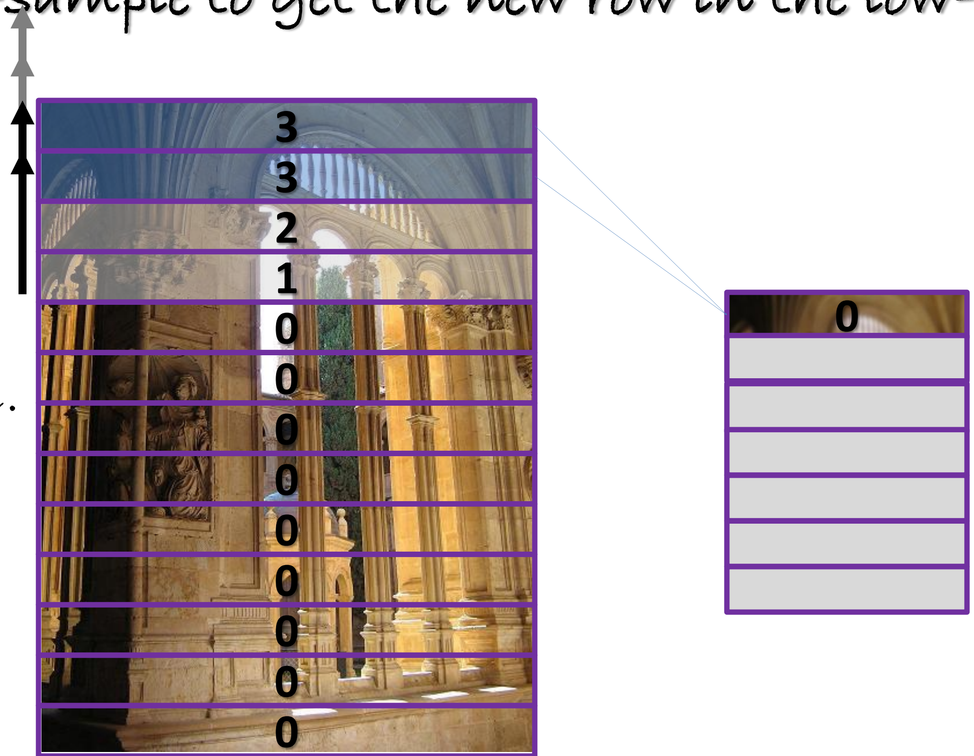
- Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start processing there as well.



Streaming Multiple GS Iterations

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

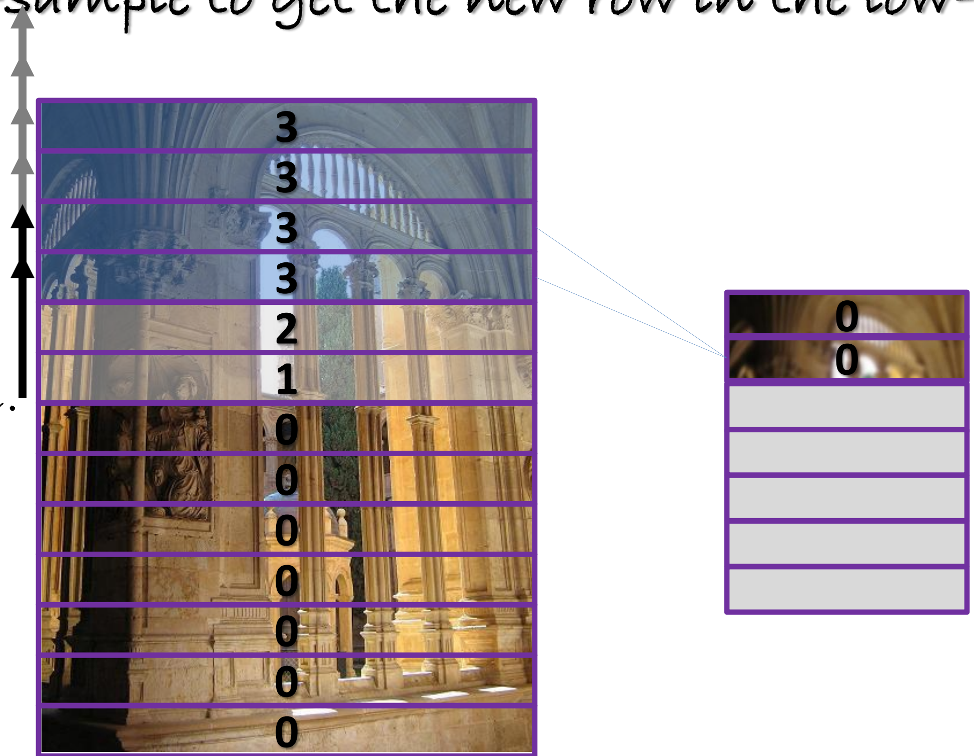
- *Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.*
- *Once we add a new row to the low-resolution system, we can start processing there as well.*



Streaming Multiple GS Iterations

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

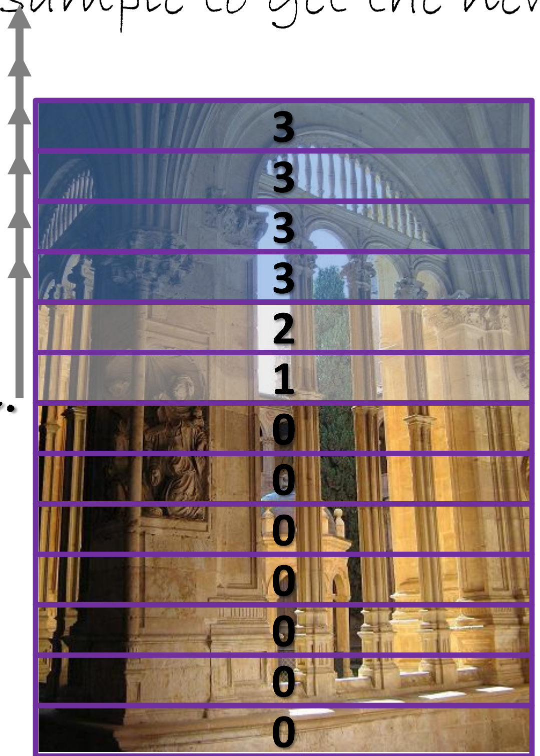
- Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.
- Once we add a new row to the low-resolution system, we can start processing there as well.



Streaming Multiple GS Iterations

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

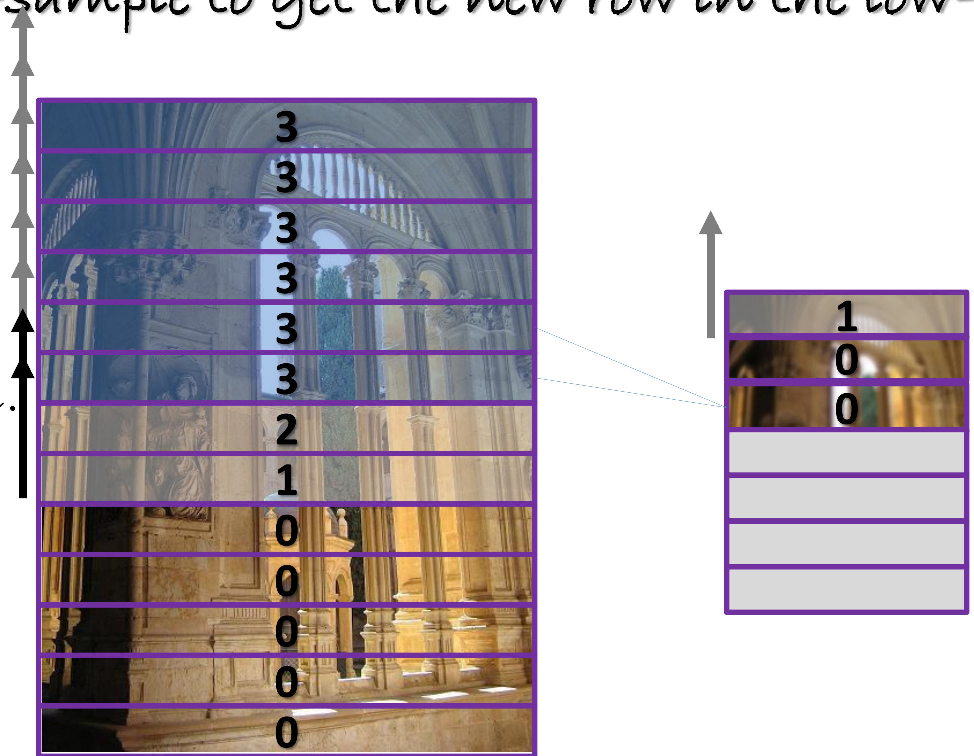
- Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.
- **Once we add a new row to the low-resolution system, we can start processing there as well.**



Streaming Multiple GS Iterations

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

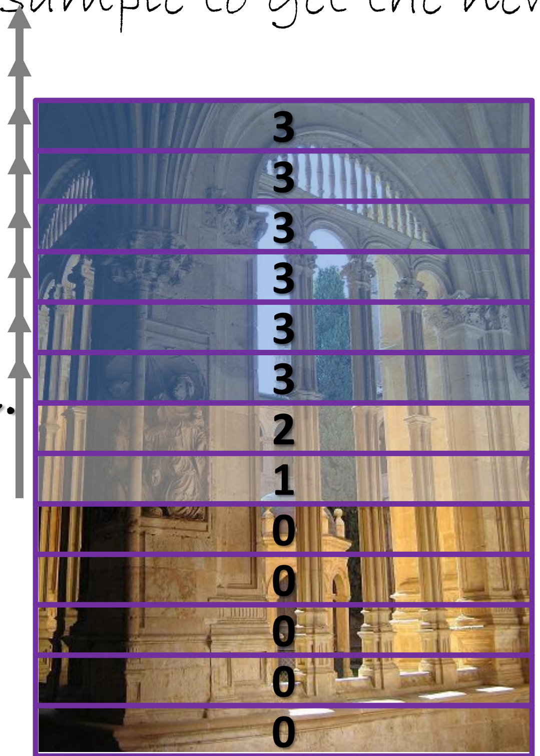
- ***Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.***
- Once we add a new row to the low-resolution system, we can start processing there as well.

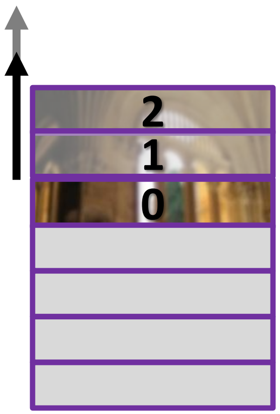


Streaming Multiple GS Iterations

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

- Once we have processed two high-resolution rows, we can get the residual and 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 processing there as well.**

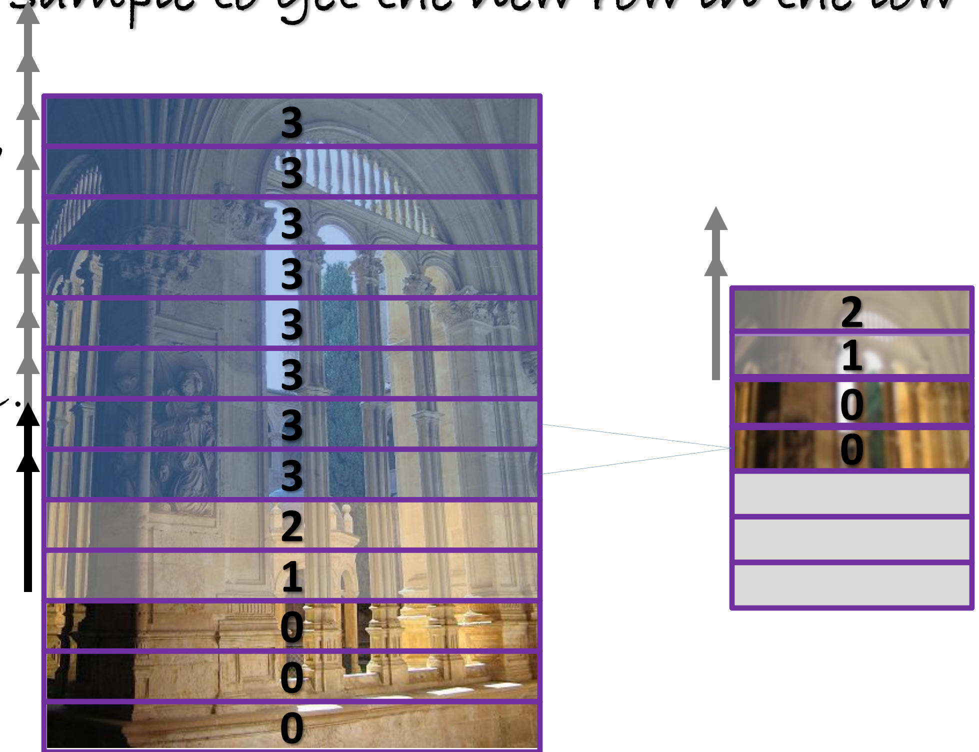




Streaming Multiple GS Iterations

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

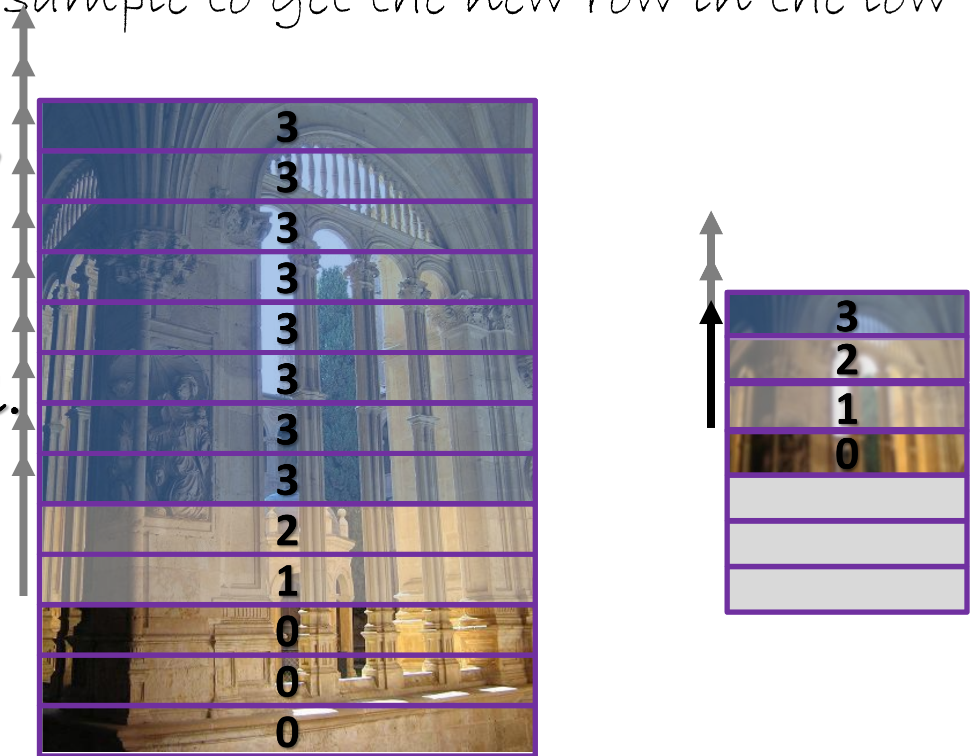
- *Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.*
- Once we add a new row to the low-resolution system, we can start processing there as well.



Streaming Multiple GS Iterations

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

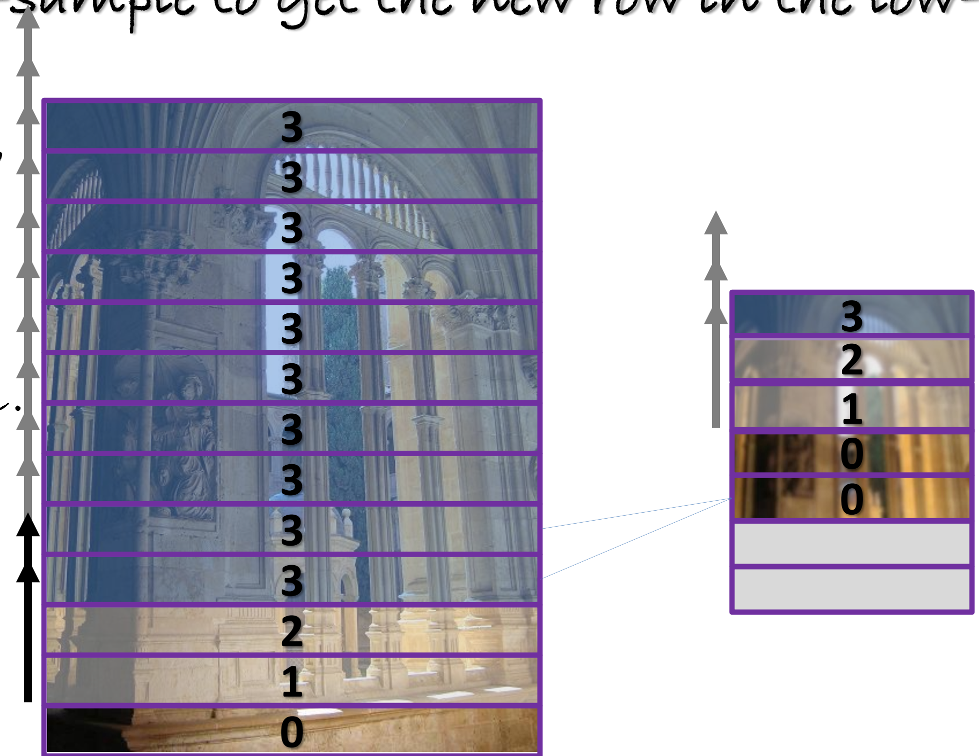
- Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.
- **Once we add a new row to the low-resolution system, we can start processing there as well.**



Streaming Multiple GS Iterations

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

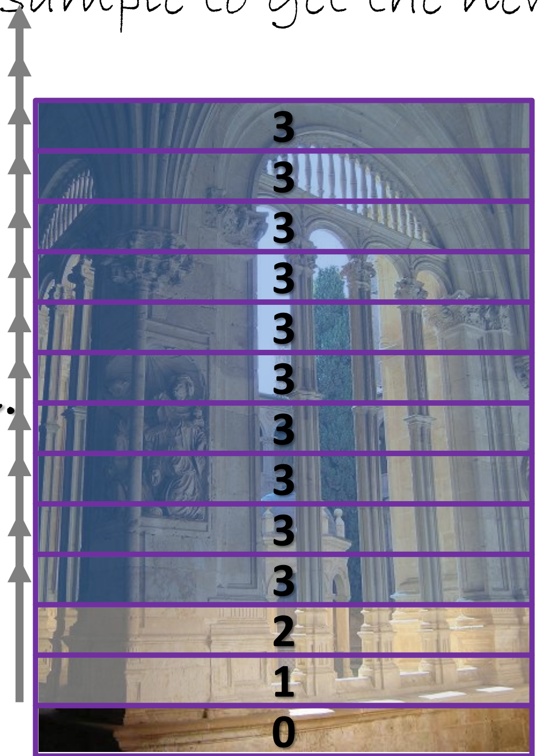
- *Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.*
- *Once we add a new row to the low-resolution system, we can start processing there as well.*

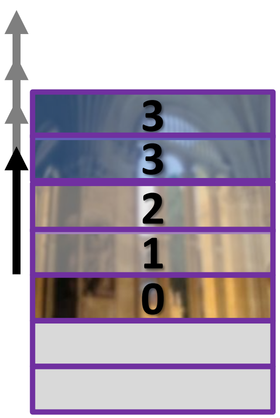


Streaming Multiple GS Iterations

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

- Once we have processed two high-resolution rows, we can get the residual and 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 processing there as well.**

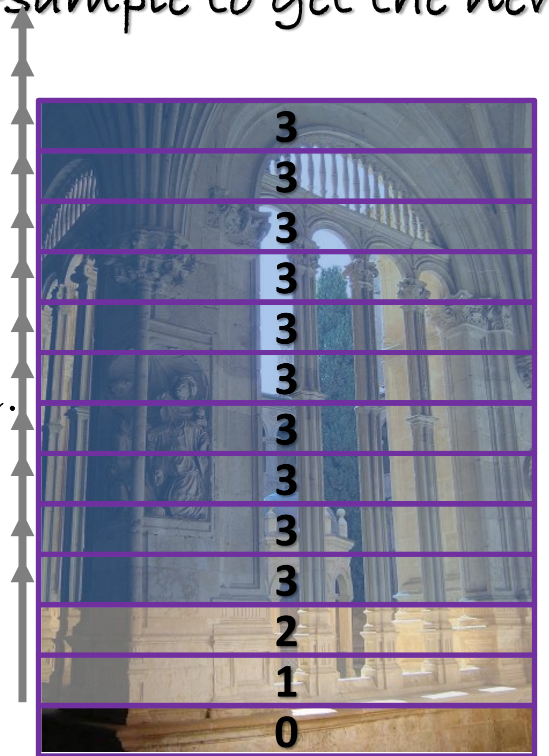


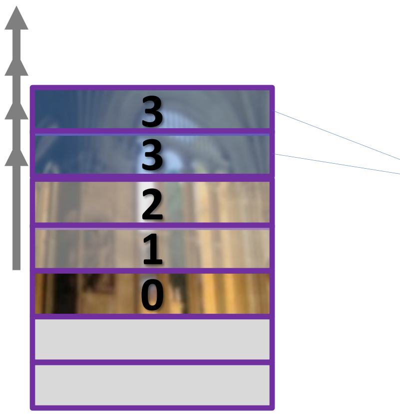


Streaming Multiple GS Iterations

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

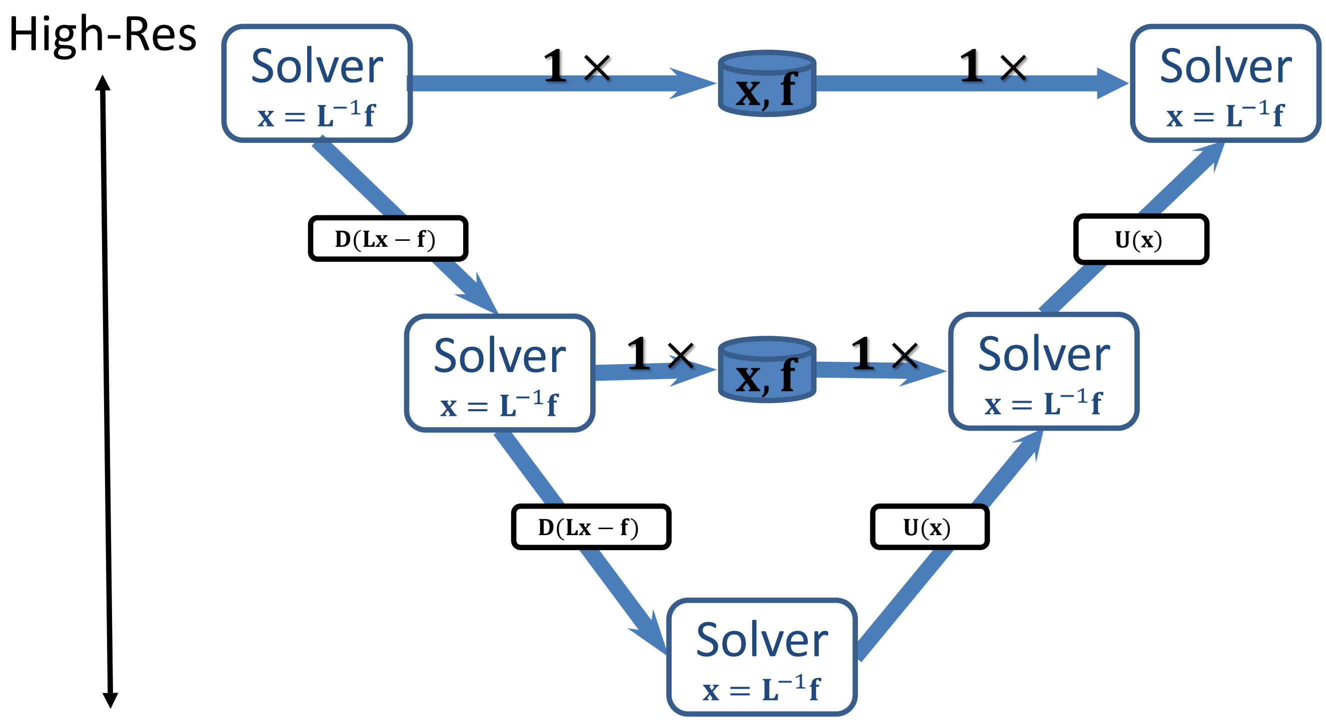
- *Once we have processed two high-resolution rows, we can get the residual and down-sample to get the new row in the low-resolution system.*
- Once we add a new row to the low-resolution system, we can start processing there as well.





Processing Large Images

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



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

$\mathbf{U}$:= Up-Sampling Operator

Outline

Motivation

What's the problem?

What's the big problem?

The Big Picture

How Big is Big?

<u>St James</u>:

Stitched from 643 photographs

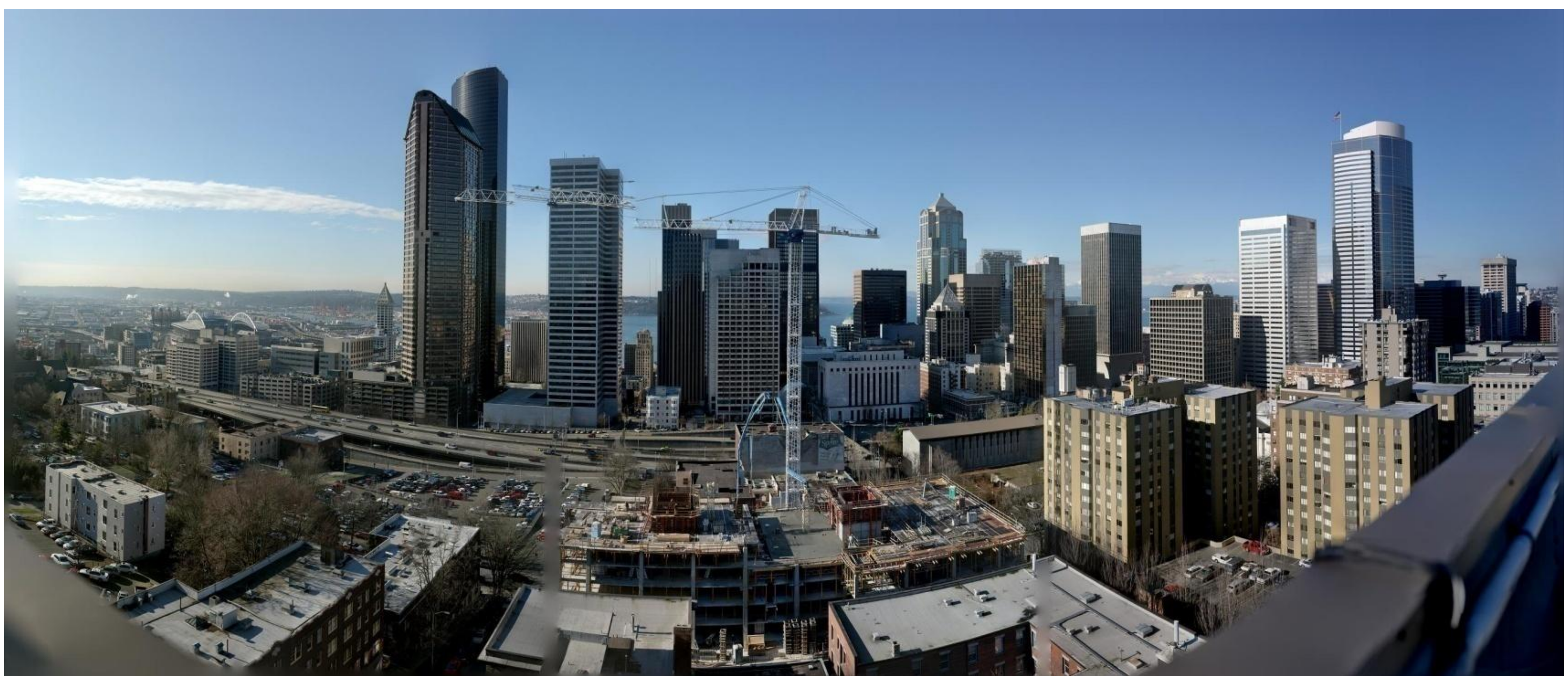
Contains 3.3 billion (88,309 x 37,842) pixels

How Big is Big?

<u>St James</u>:

Stitched from 643 photographs

Contains 3.3 billion (88,309 x 37,842) pixels

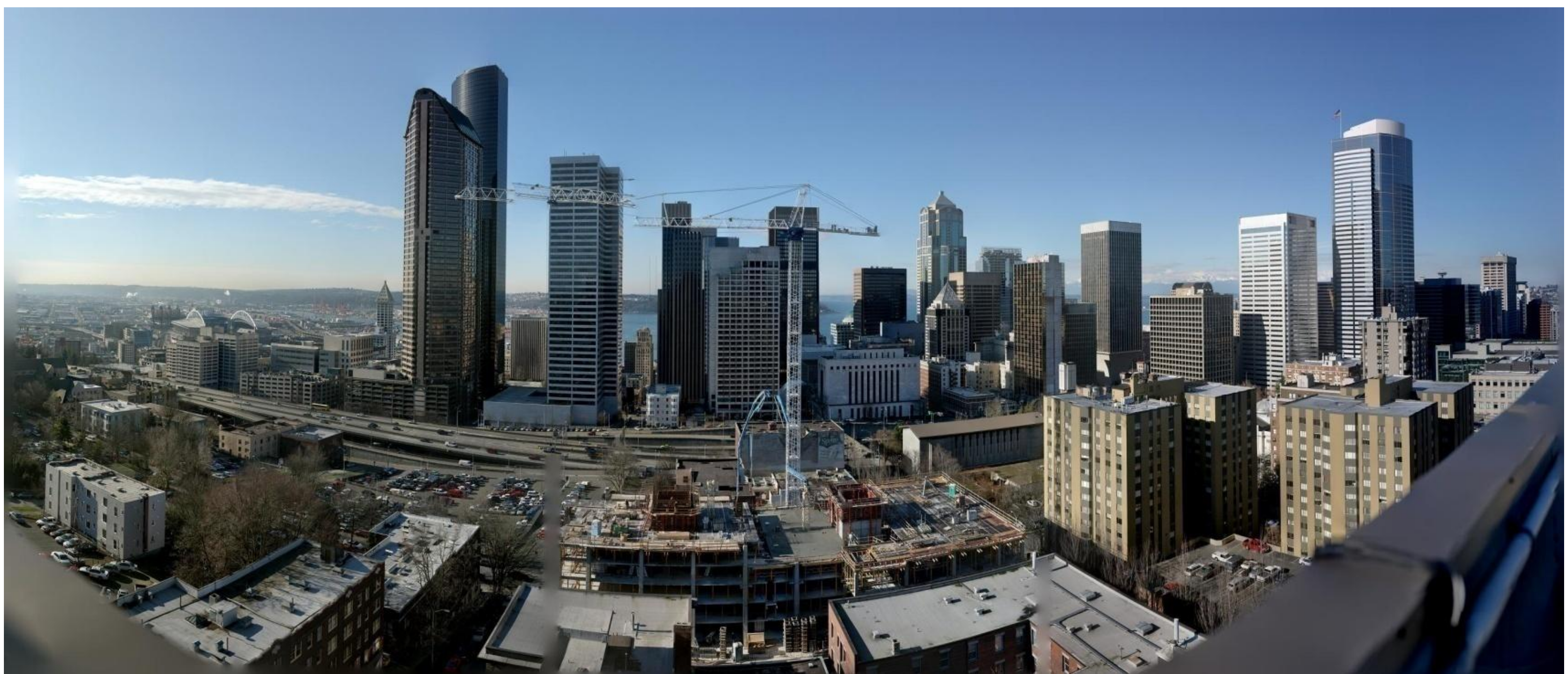


How Big is Big?

St James:

Peak Memory Usage: 408 MB

Solver Time: 1:27:50

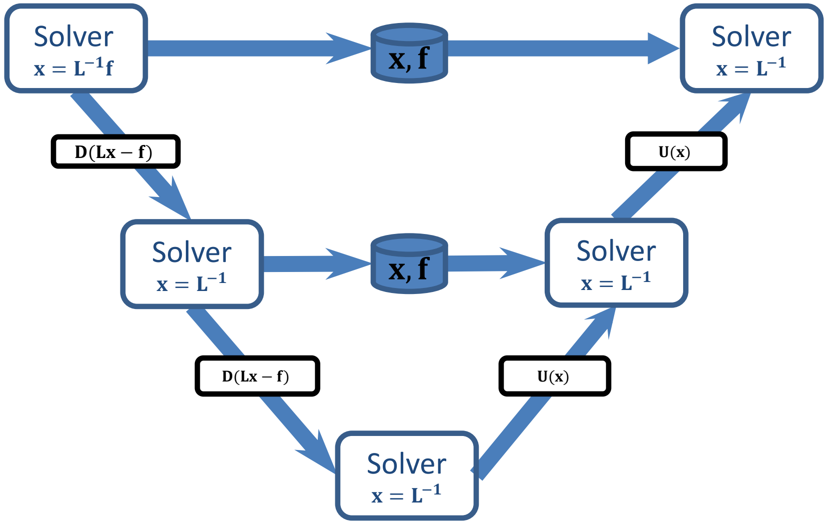


How Big is Big?

Are these good numbers?

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

How Big is Big?



Are these good numbers?

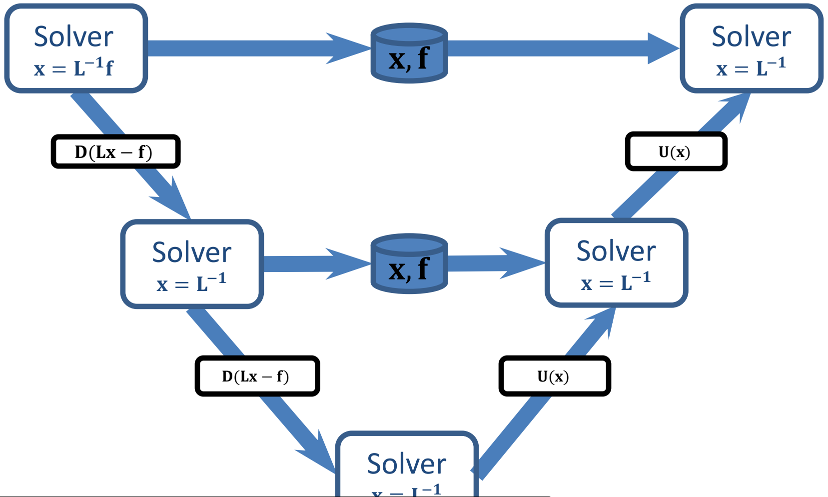
Peak Memory Usage: 408 MB

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 $\mathbf{x}$ and $\mathbf{f}$: 53 GB
- Up-Sampling
  - Read in $\mathbf{x}$ and $\mathbf{f}$: 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

Cons n):

- Do
  -
  -
- Up
  - Read in $\mathbf{x}$ and $\mathbf{f}$: 53 GB
  - Write out the image: 20 GB

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

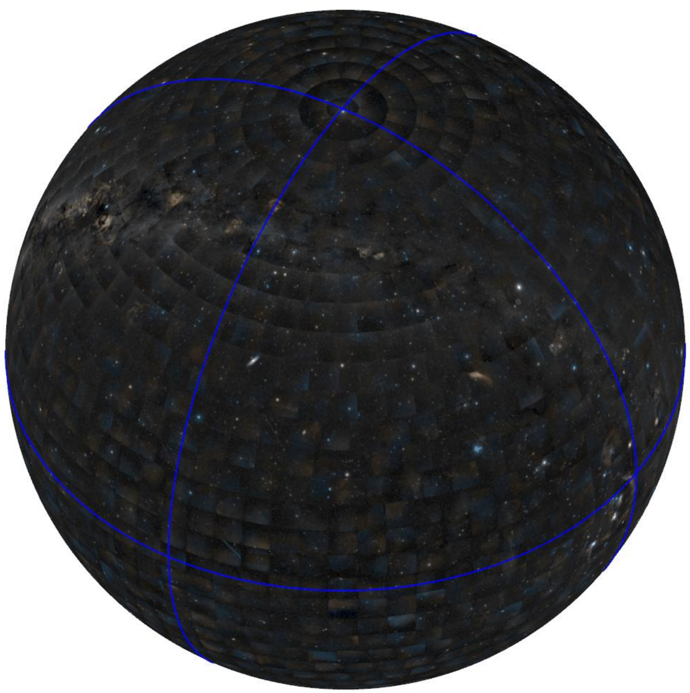
Assuming 30-35 MB/s reads and writes, I/O alone would take:
1:19:02 -- 1:32:13

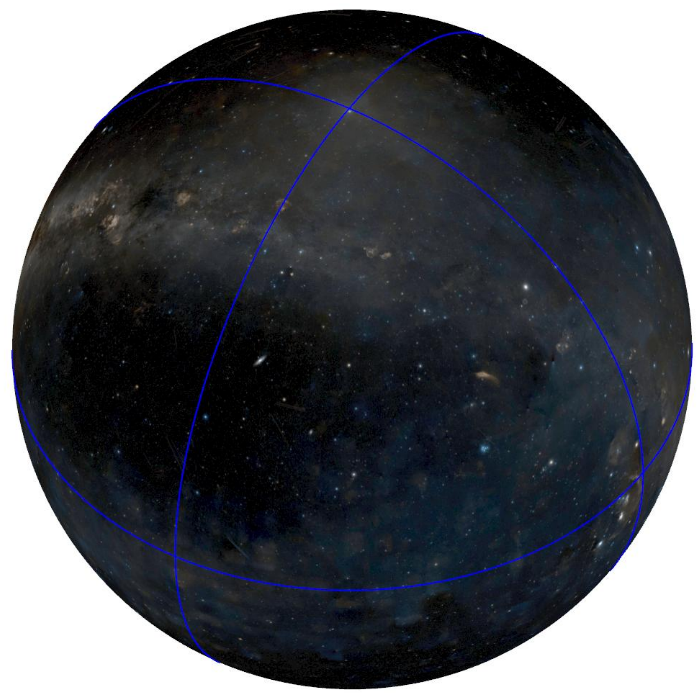
How Big is Big?

WWT:

Stitched from 1790, 529 MP photographs

Comprises a 1 terapixel image



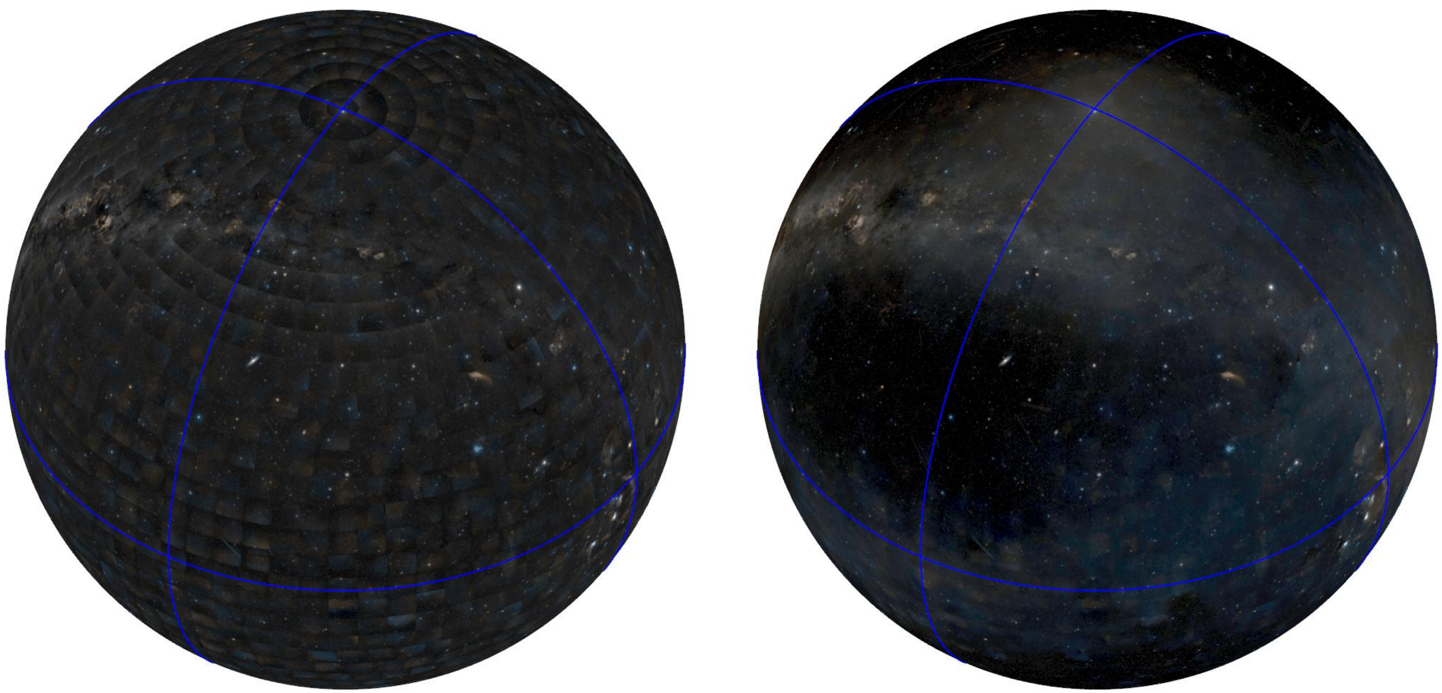


How Big is Big?

WWT:

Solver Time: 9 hours

Distributed across a 16-node cluster



How Big is Big?

WWT: