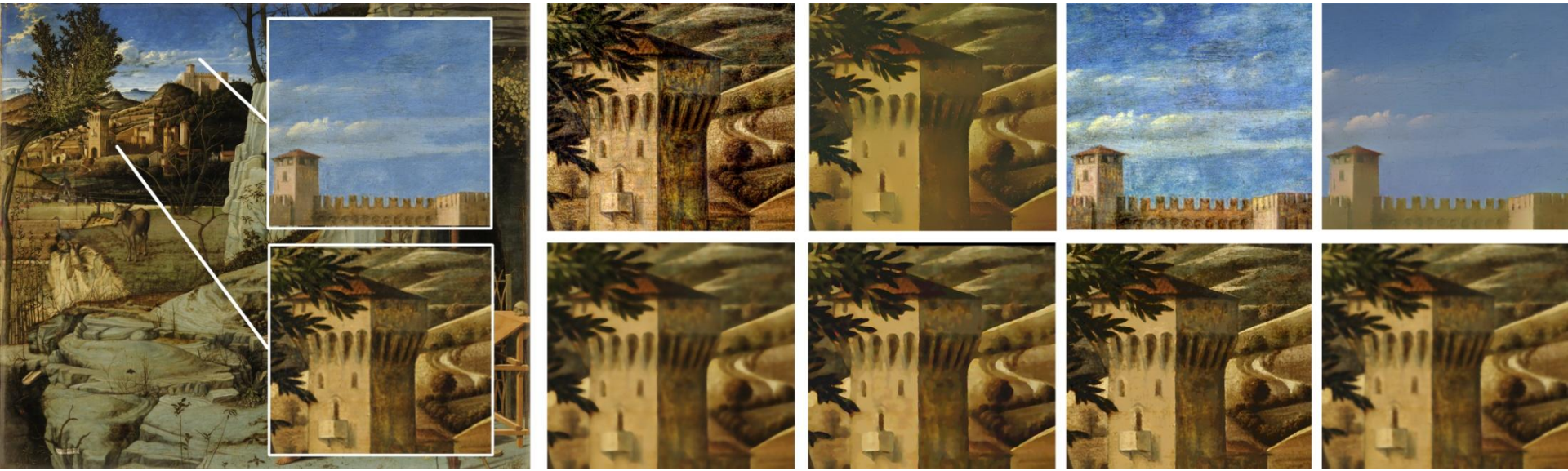
Interactive Non-Linear Image Operations on Gigapixel Images



Markus Hadwiger, Ronell Sicat, Johanna Beyer

King Abdullah University of Science and Technology

King Abdullah University of Science and Technology

KAUST

Display-Aware Image Operations

Display-Aware Image Operations

- Require multi-resolution representation, e.g., mipmaps
  - Enables choosing desired resolution
  - Anti-aliasing via pre-filtering
- Problem when image is modified by non-linear operation
  - Apply to coarse resolution directly: incorrect
  - Apply to original image: rebuild multi-resolution structure after every modification
- New type of multi-resolution pyramid: *Sparse PDF Maps*
  - Each pixel is represented by the probability density function (pdf) of its footprint in the original image
  - Enables accurate computation of many non-linear operators from one data structure

Image Pyramids

- Dyadic image pyramids (each level half of previous in each axis)
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression

level 0

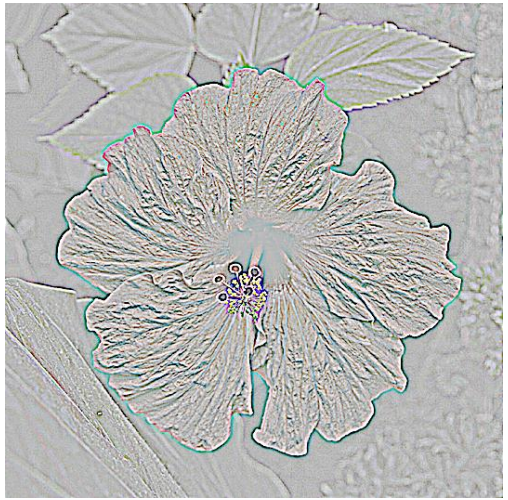
level 1

level 2

level 3

Image Pyramids

- Dyadic image pyramids (each level half of previous in each axis)
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression



level 0

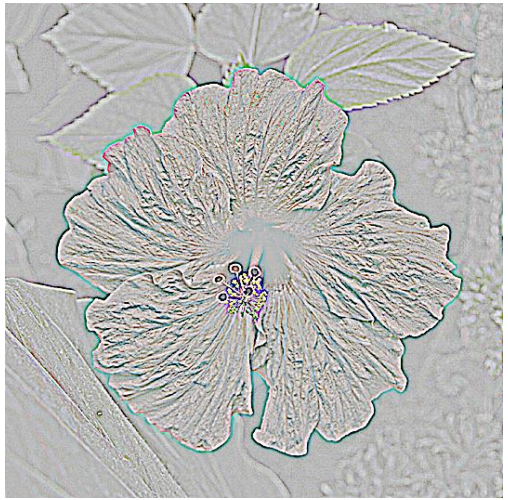
level 1

level 2

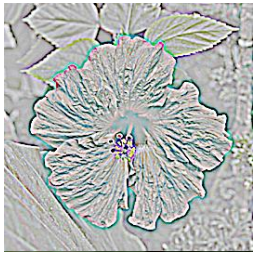
level 3

Image Pyramids

- Dyadic image pyramids (each level half of previous in each axis)
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression



level 0



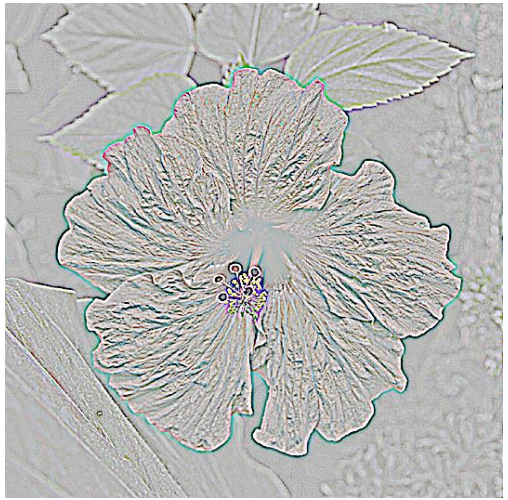
level 1

level 2

level 3

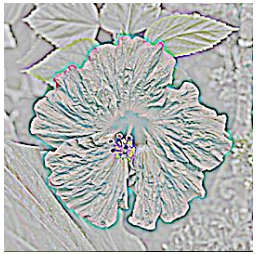
Image Pyramids

- Dyadic image pyramids (each level half of previous in each axis)
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression
  - Sparse pdf maps [Hadwiger et al. 2012]



level 0

Laplacian pyramid



level 1

level 2

level 3

Image Pyramids

- Dyadic image pyramids (each level half of previous in each axis)
  - Mipmaps [Williams 1983]: texture mapping (standard on GPUs)
  - Gaussian/Laplacian pyramids [Burt and Adelson 1983]: image processing/compression
  - Sparse pdf maps [Hadwiger et al. 2012]

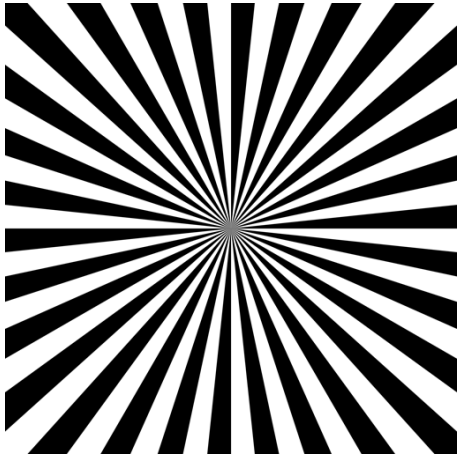
level 0

level 1

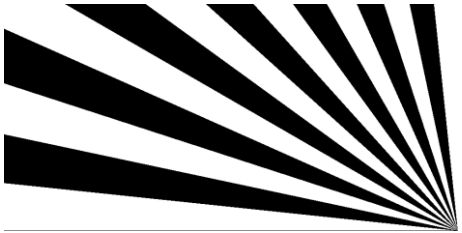
level 2

level 3

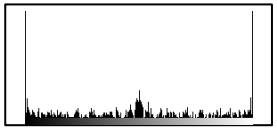
Anti-Aliasing in Image Pyramids

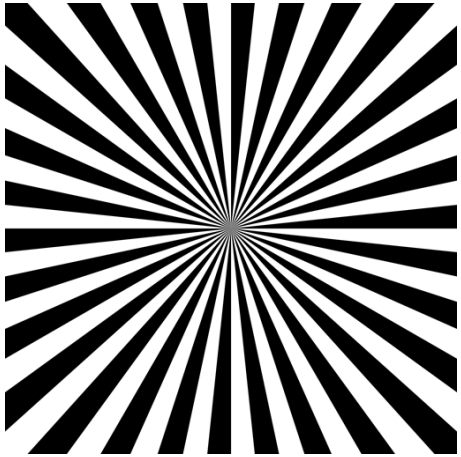


level 0

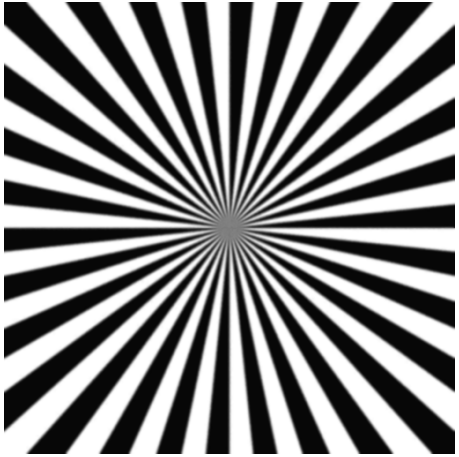


Anti-Aliasing in Image Pyramids

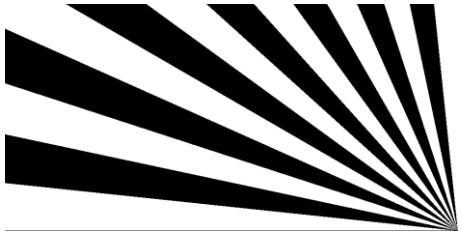




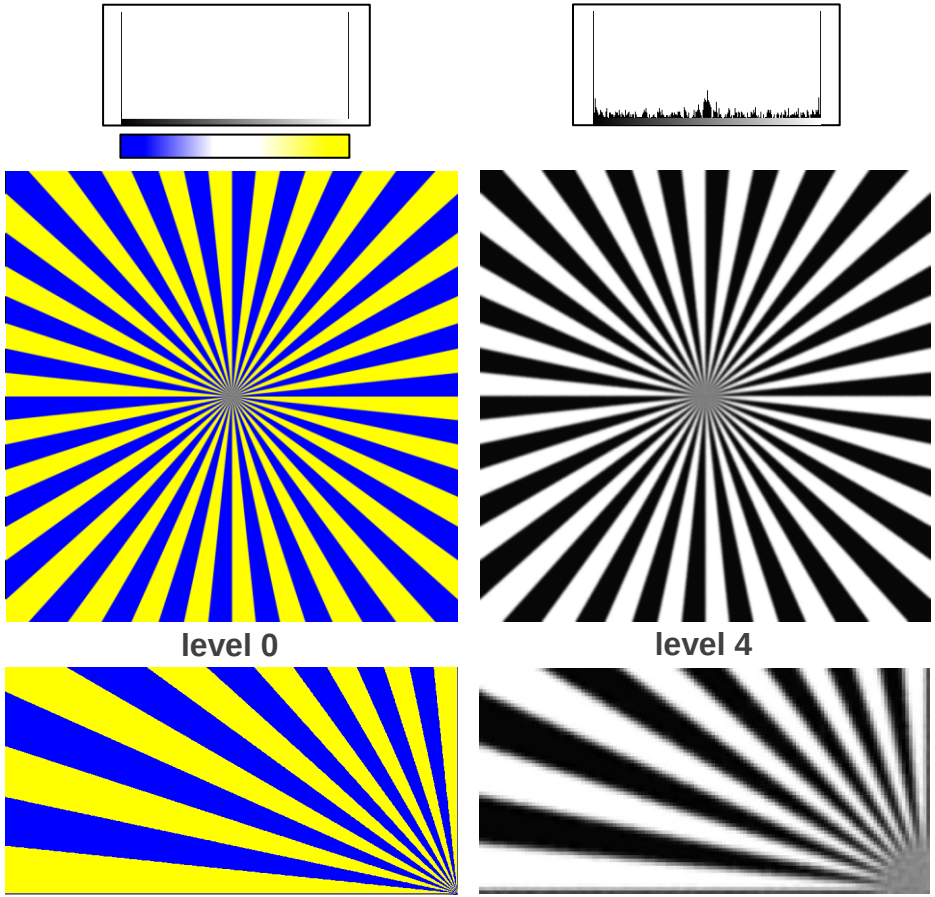
level 0



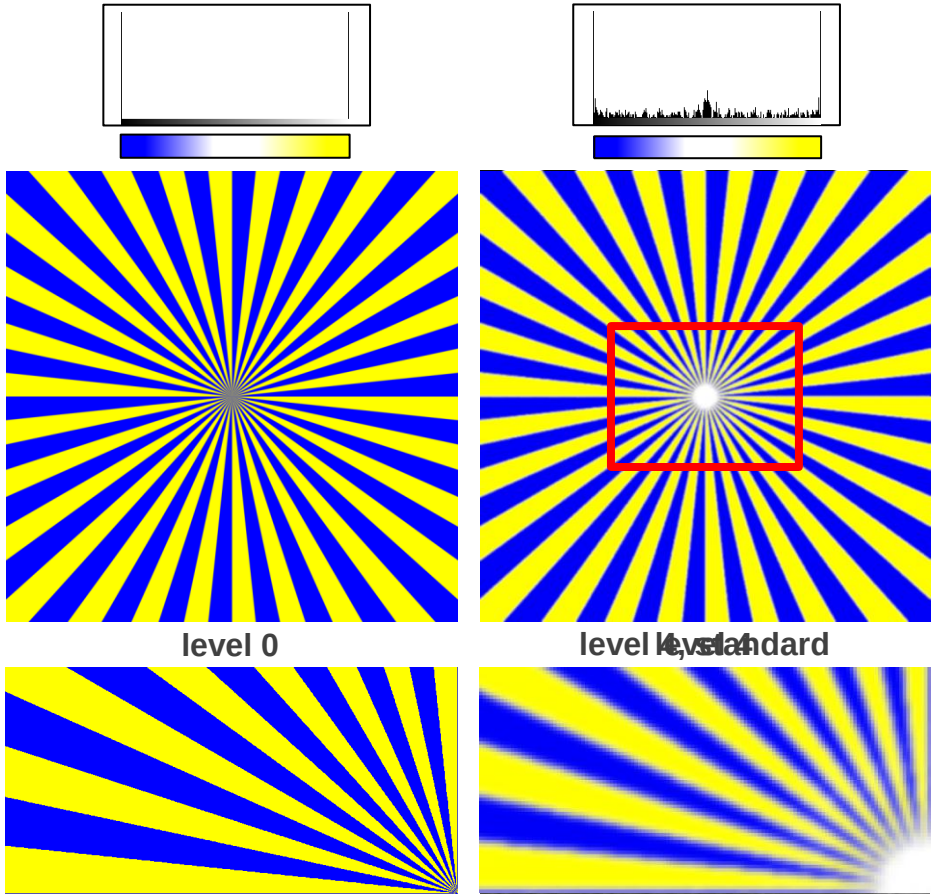
level 4



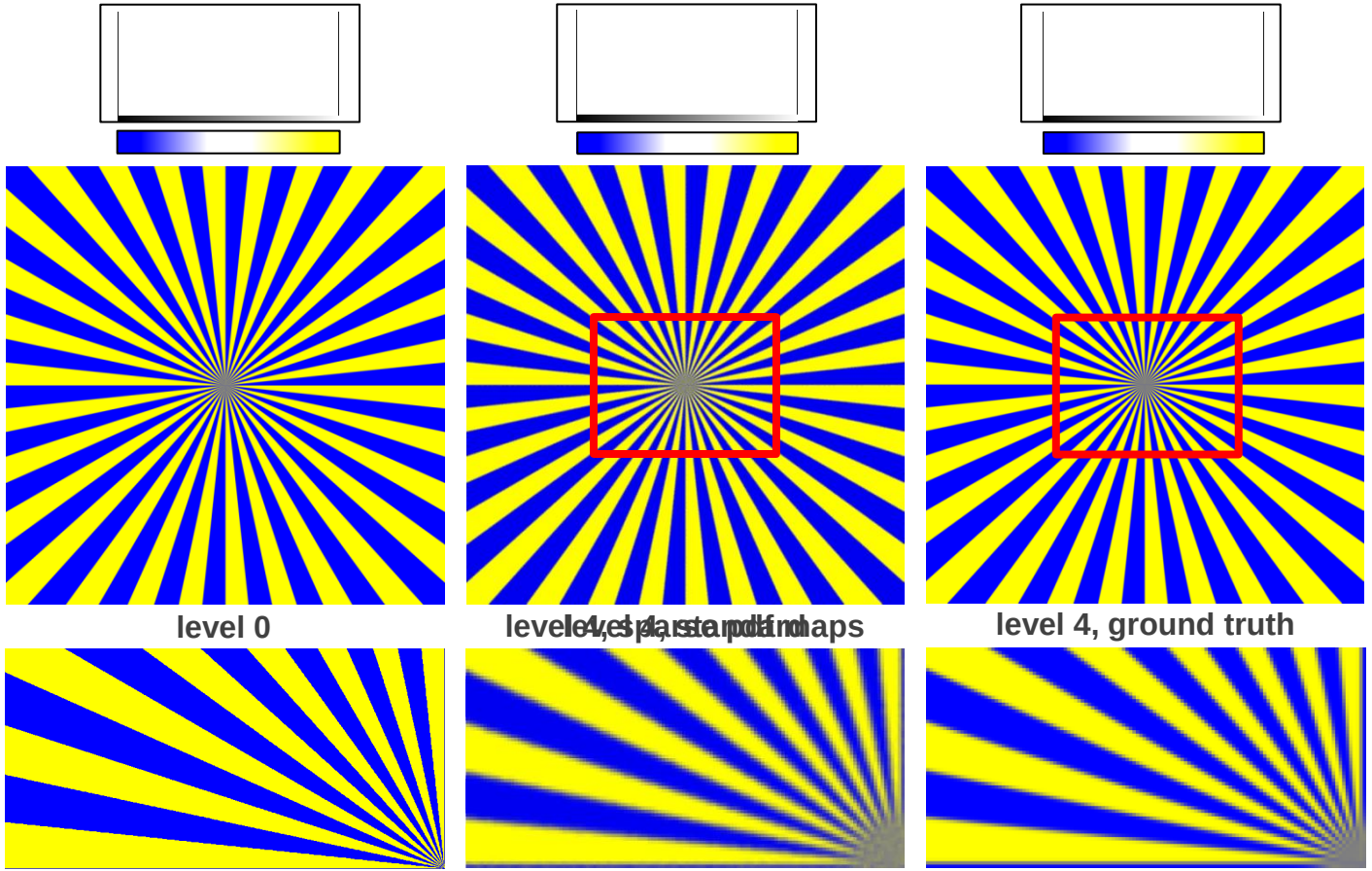
Anti-Aliasing in Image Pyramids



Anti-Aliasing in Image Pyramids

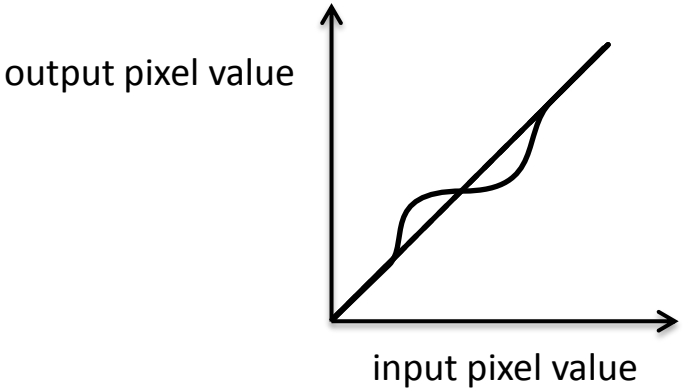


Anti-Aliasing in Image Pyramids



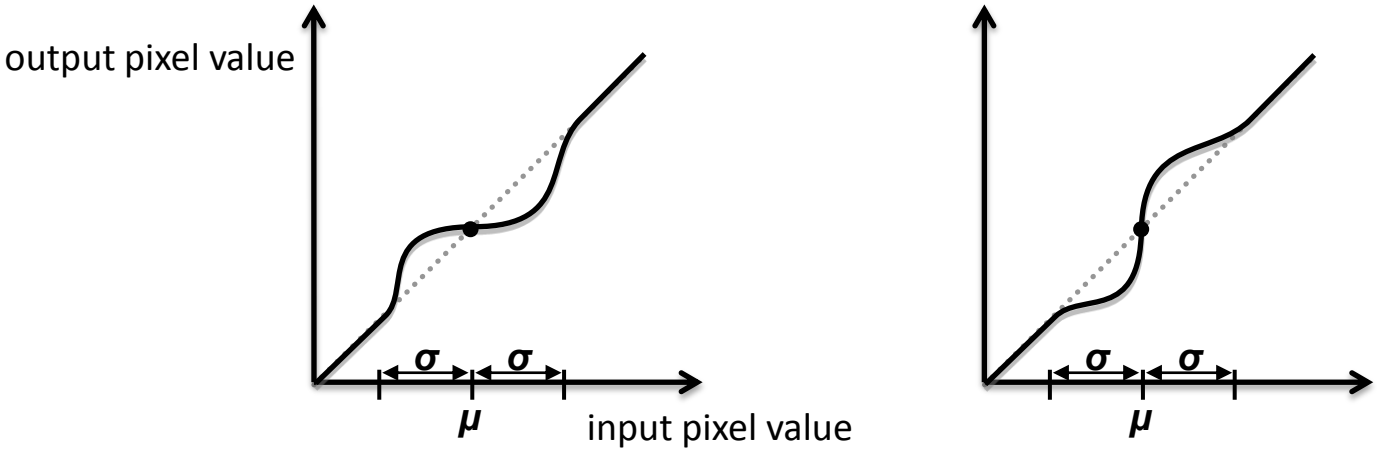
Non-Linear Image Operators

- Apply non-linear operation to each pixel
  - Color map or non-linear contrast adjustment
  - Bilateral filtering: range weight
  - Smoothed local histogram filtering [Kass and Solomon 2010]
  - Local Laplacian filtering [Paris et al. 2011]: point-wise, non-linear re-mapping functions

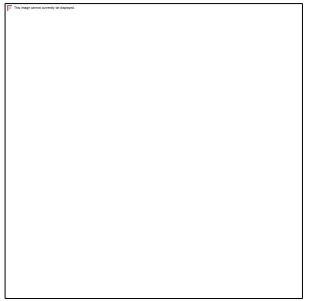


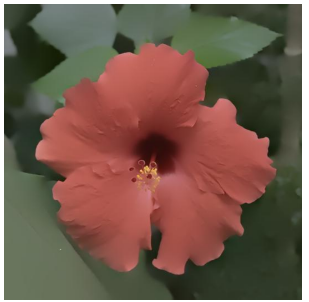
Local Laplacian Filtering [Paris et al. 2011]

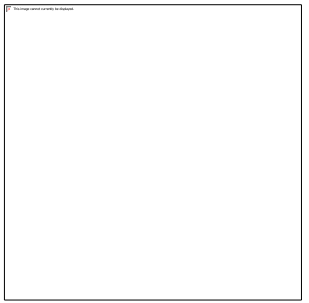
- Compute Laplacian pyramid coefficient
  - Adjust local contrast via point-wise non-linearity; then downsample



- Same as local color mapping, then downsampling
  - Cannot apply the re-mapping function to the downsampled image!
  - Need to compute either ground truth (pyramid!) or proper “anti-aliasing”

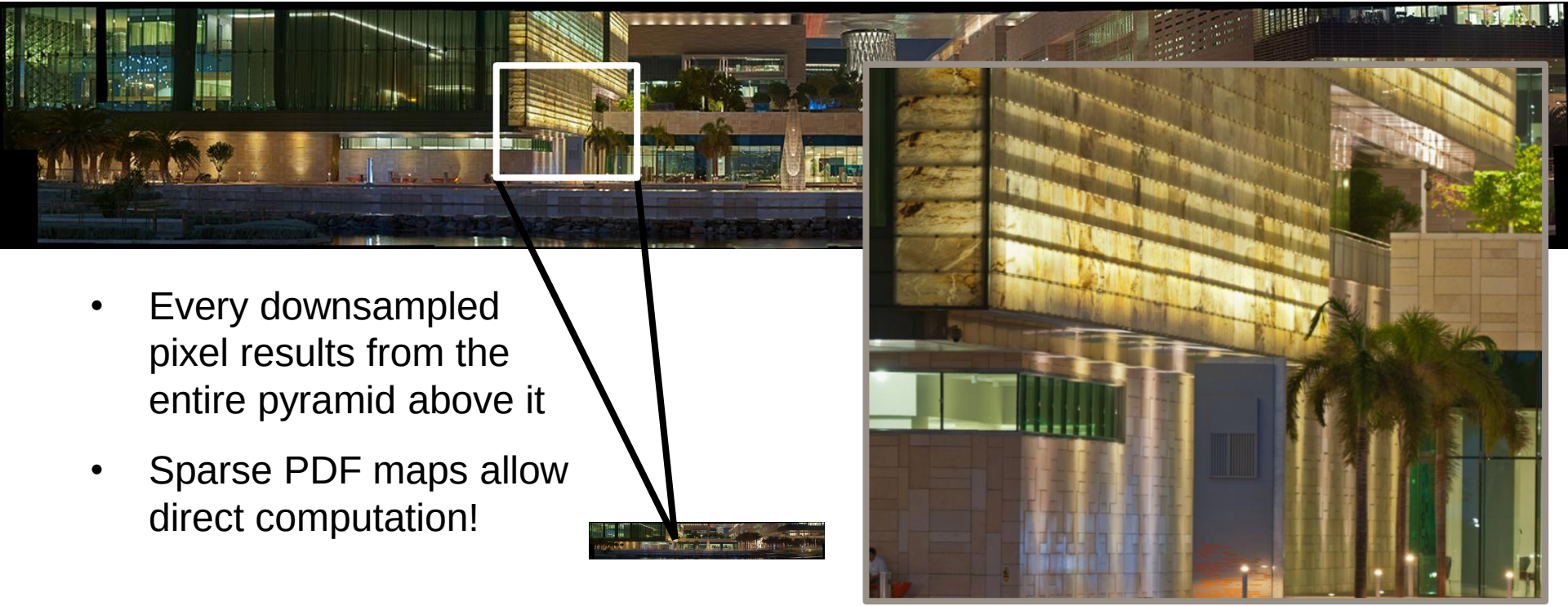






Local Laplacian Filtering: Scalability

- Night Scene Panorama: 47,908 x 7,531 pixels (361 Mpixels)

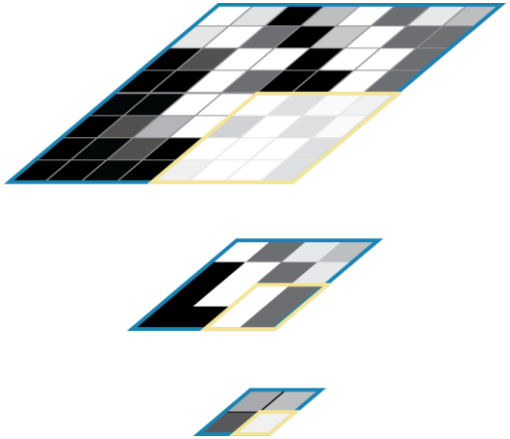


- Every downsampled pixel results from the entire pyramid above it
- Sparse PDF maps allow direct computation!

Sparse PDF Maps: Concept

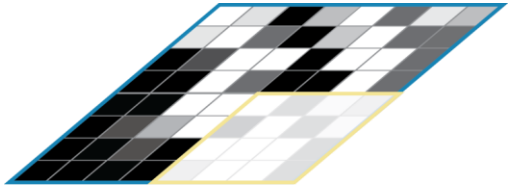
Sparse PDF Maps

- Represent distribution of pixel values in footprint in original image



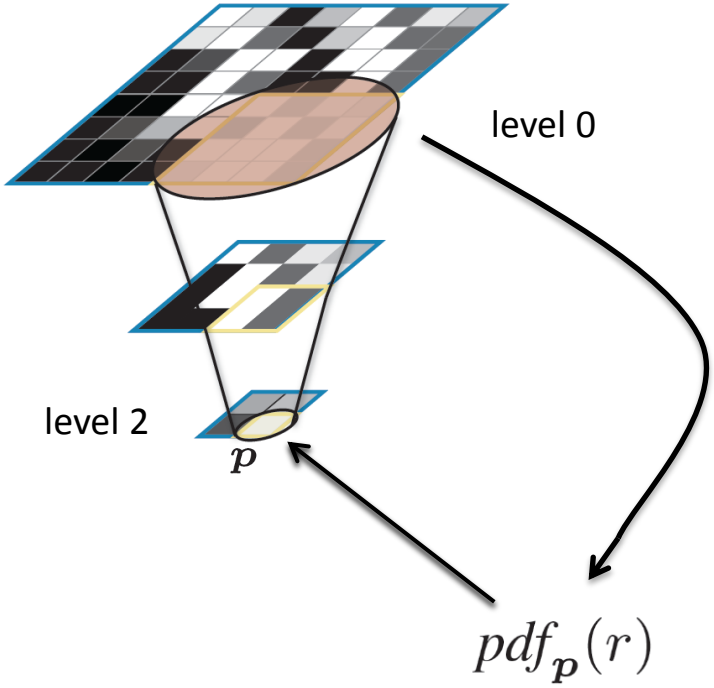
Sparse PDF Maps

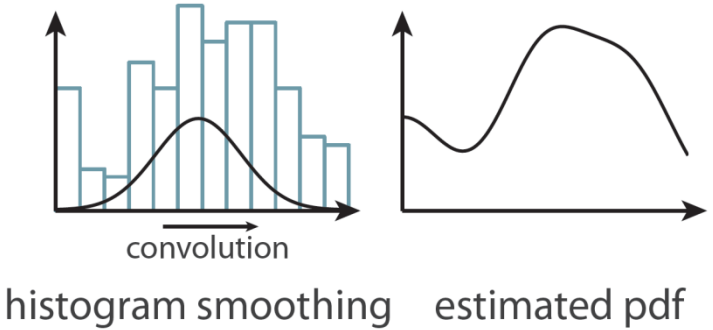
- Represent distribution of pixel values in footprint in original image



Sparse PDF Maps

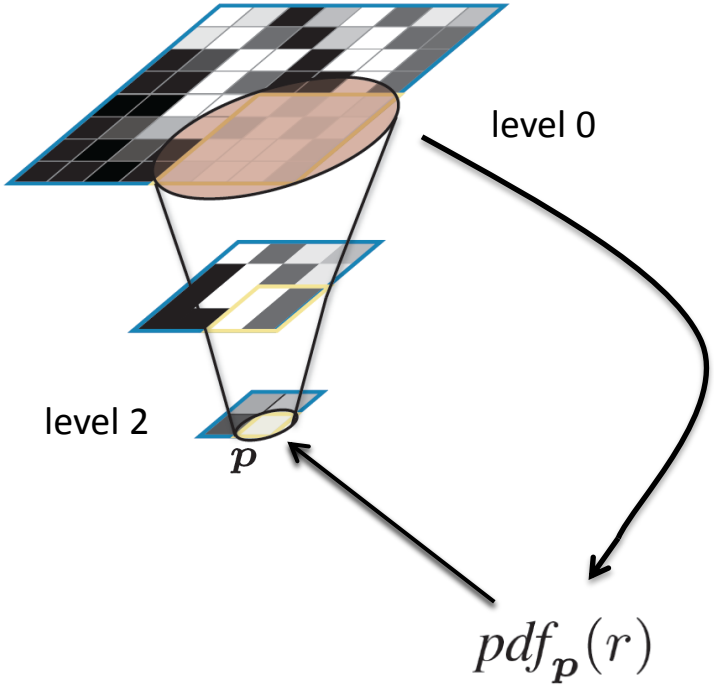
- Represent distribution of pixel values in footprint in original image

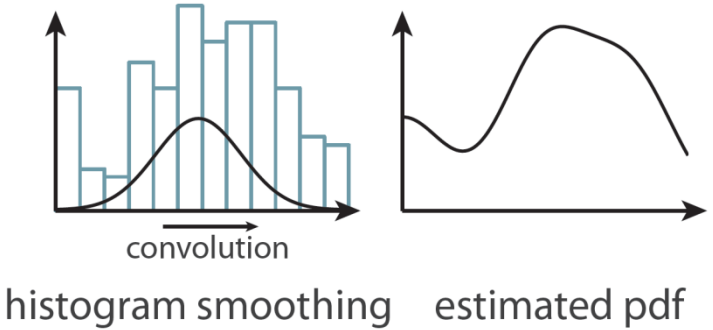




Sparse PDF Maps

- Represent distribution of pixel values in footprint in original image





Sparse PDF Maps

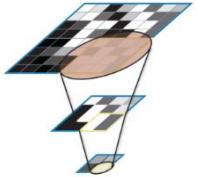
- Represent distribution of pixel values in footprint in original image

- Apply non-linear operation

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r)\, pdf_{\boldsymbol{p}}(r)\, dr$$

Example 1: Downsampled Image

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r) pdf_{\boldsymbol{p}}(r)\, dr$$

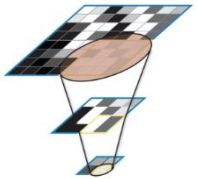


$$t_{\boldsymbol{p}}(r) = r$$

$$w_{\boldsymbol{p}} = 1$$

Example 2: Color Mapping

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r) pdf_{\boldsymbol{p}}(r)\, dr$$

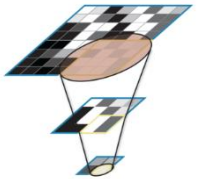


$t_{\boldsymbol{p}}(r) =$ color map

$w_{\boldsymbol{p}} = 1$

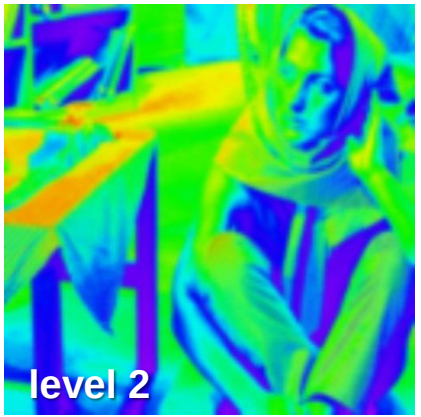
Example 2: Color Mapping

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r) pdf_{\boldsymbol{p}}(r)\, dr$$

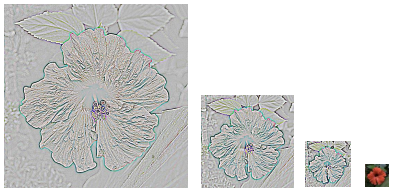


$t_{\boldsymbol{p}}(r) =$ color map

$w_{\boldsymbol{p}} = 1$



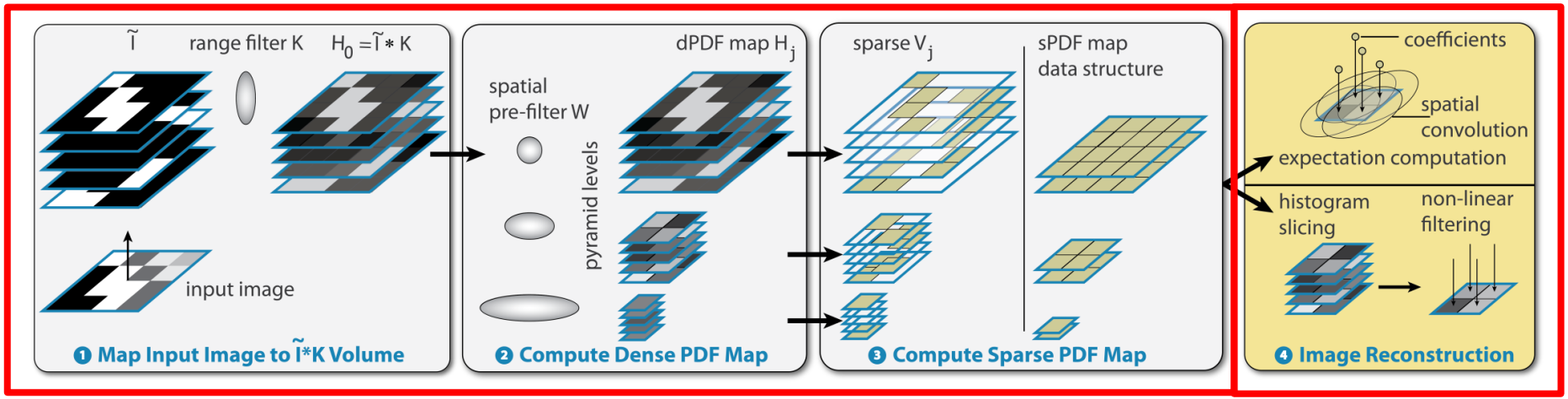
and: bilateral filtering, local Laplacian filtering in linear time, …



Interactive Gigapixel Filtering

Sparse PDF Map Computation

Pipeline

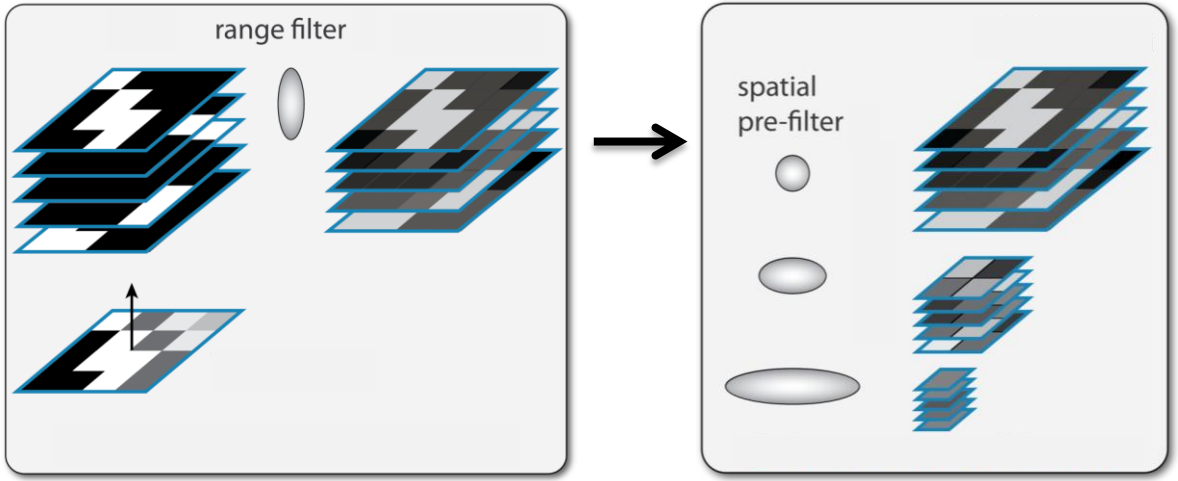


pre-computation

run time

Step 1:
Dense PDF Map

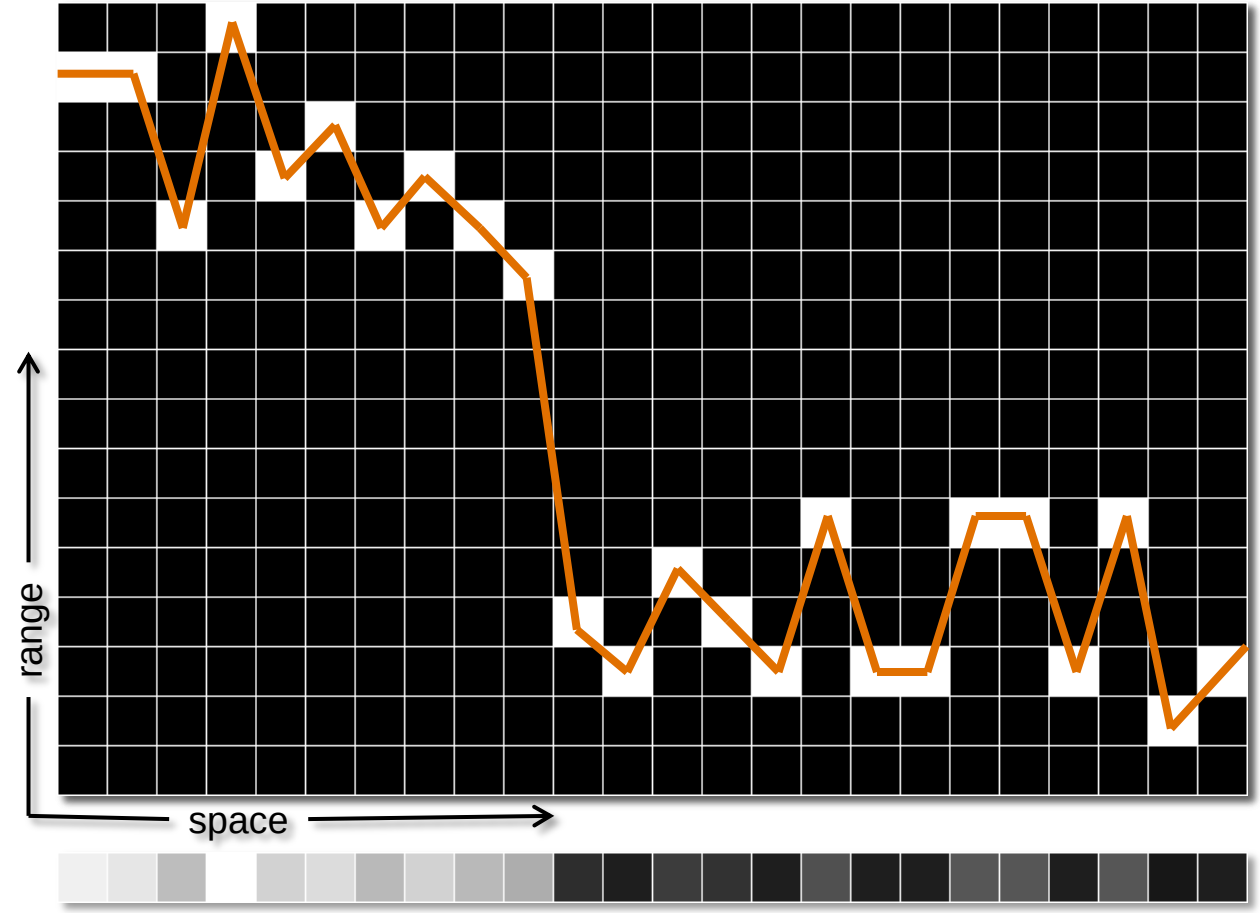
Dense PDF Map Computation



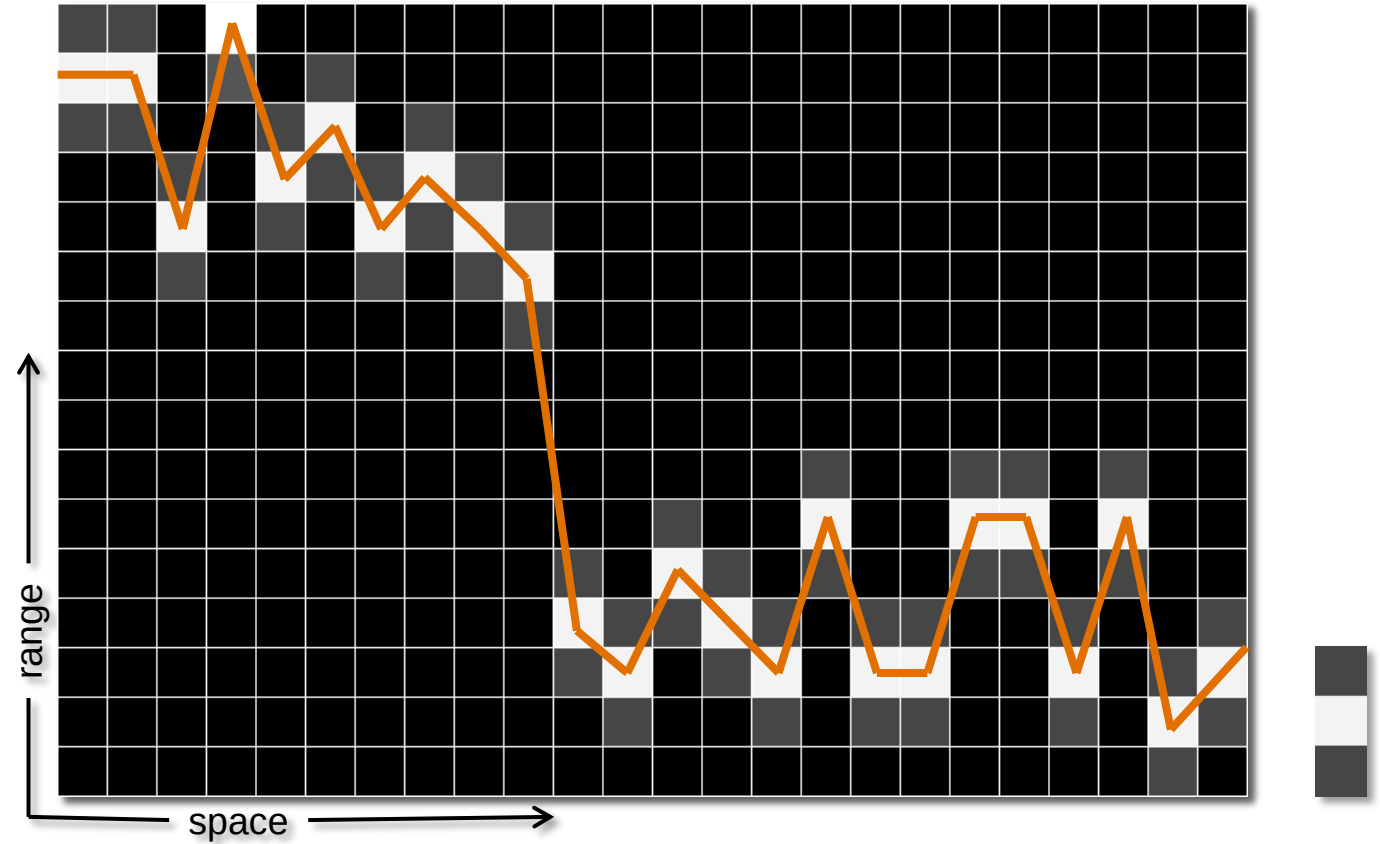
H

H is similar to a pyramid of bilateral grids [Chen et al. 2007]

Space x Range Domain

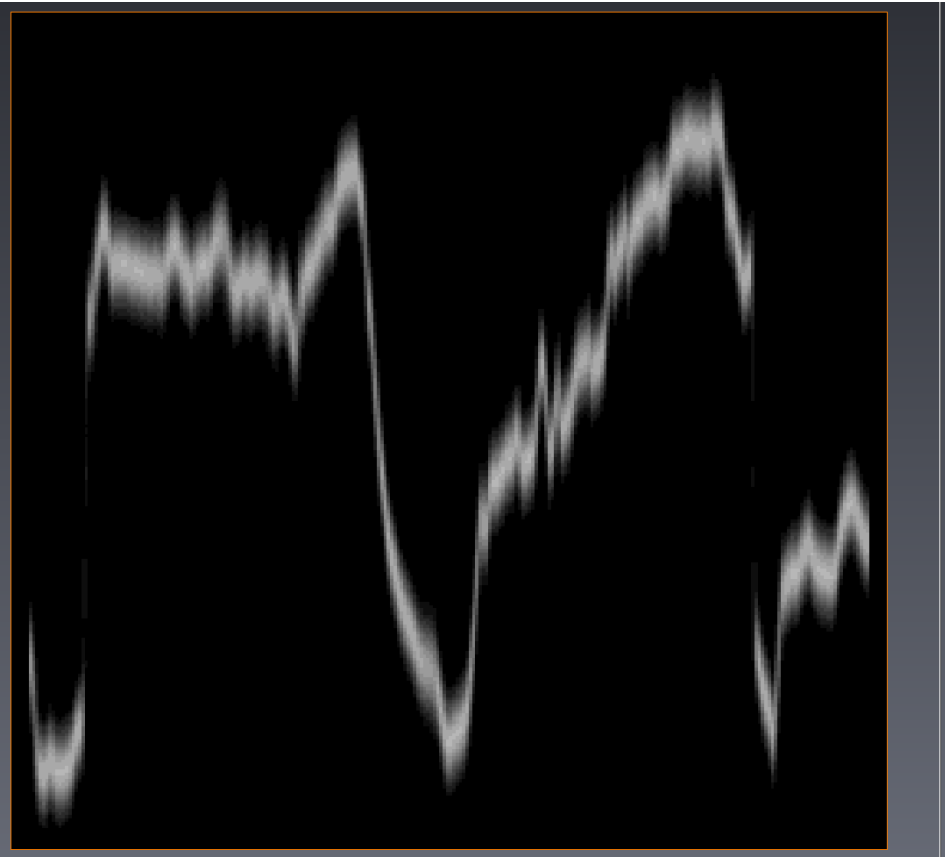


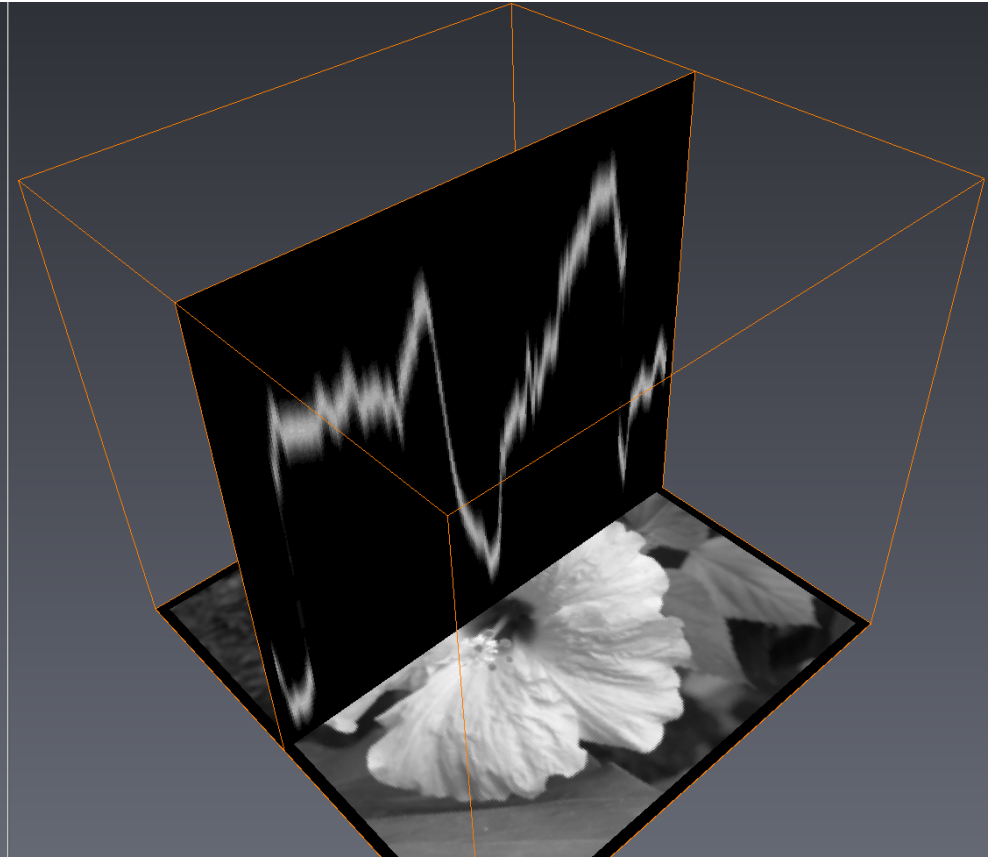
Range Smoothing



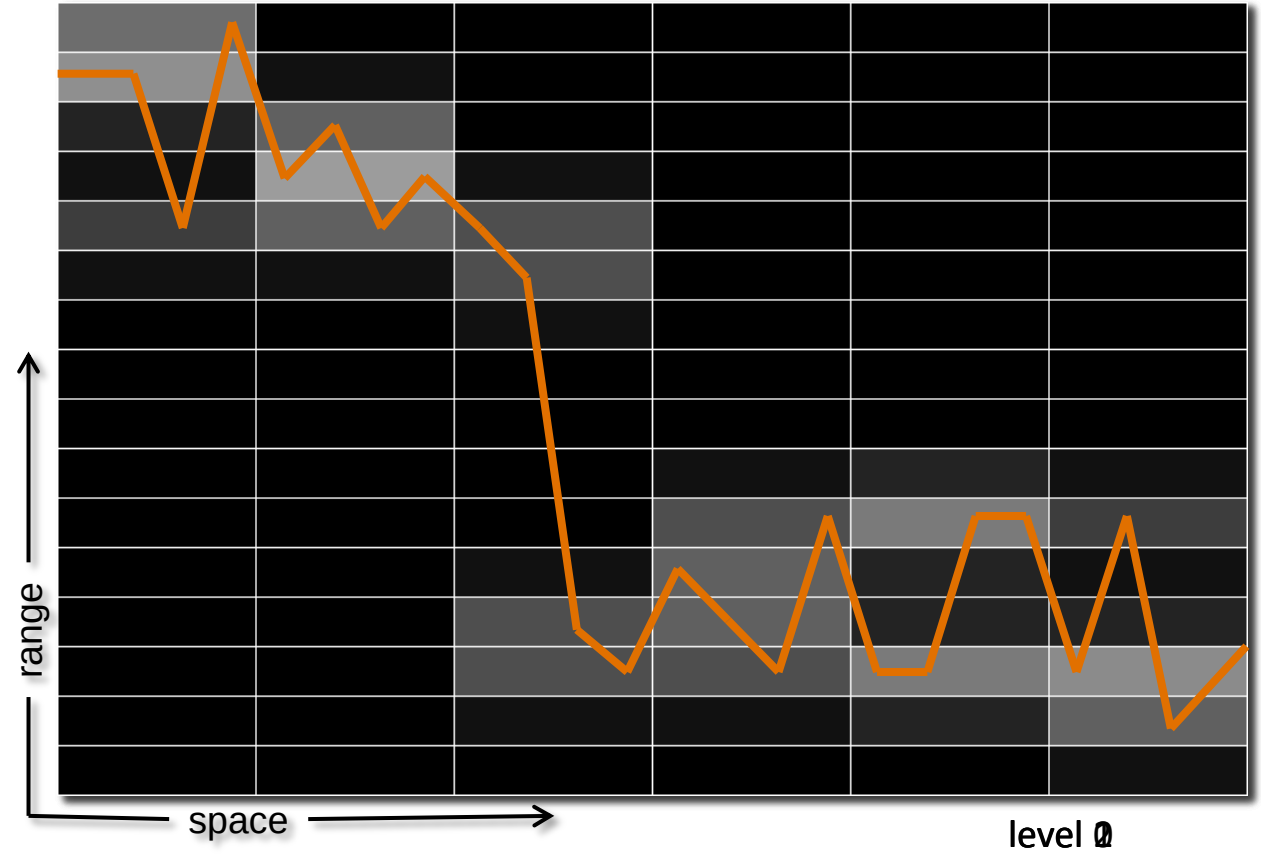
range smoothing as in smoothed local histograms [Kass and Solomon 2010]

Range Smoothing



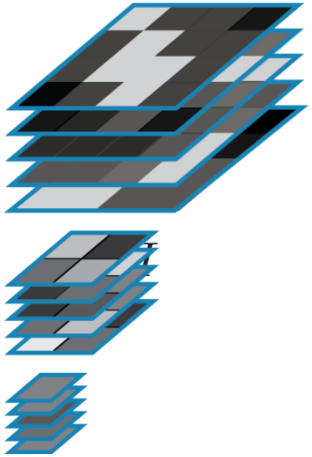


Dense PDF Map Generation



Step 2:
Sparse PDF Map

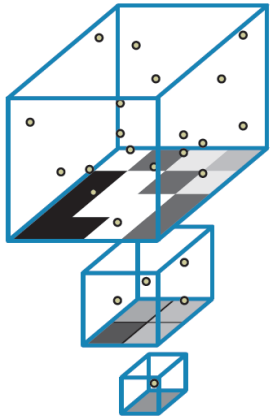
Sparse Representation



H

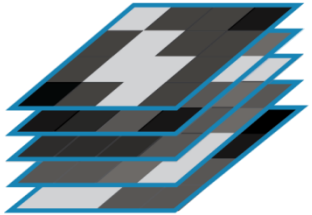
$$H \approx V * (W \otimes K)$$

$$G_{\boldsymbol{\sigma}_s} \otimes G_{\sigma_r}$$



V

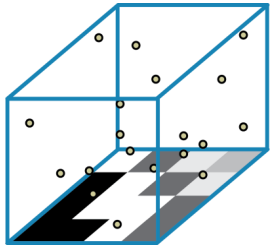
Sparse Representation



H

$$H \approx V * (W \otimes K)$$

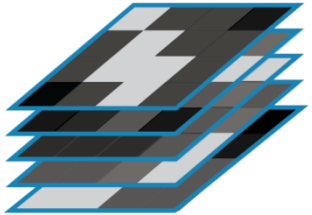
$$G_{\boldsymbol{\sigma}_s} \otimes G_{\sigma_r}$$



V

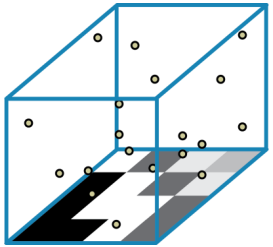
Compute V via Matching Pursuit [Mallat and Zhang 1993]

Sparse Representation



H

$$H \approx V * (W \otimes K)$$

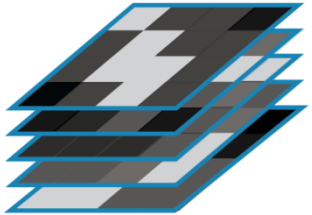


V

Compute V via Matching Pursuit [Mallat and Zhang 1993]

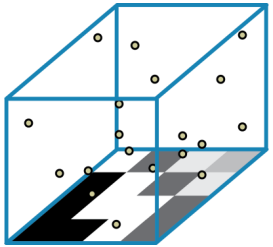
Sparse volume $V(\boldsymbol{p}_n, r_n) = c_n$ with $c_n \neq 0$

Sparse Representation



H

$$H \approx V * (W \otimes K)$$



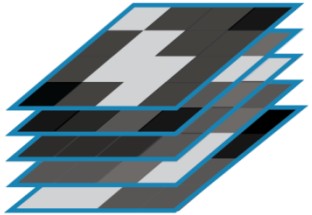
V

Compute V via Matching Pursuit [Mallat and Zhang 1993]

Sparse volume $V(\boldsymbol{p}_n, r_n) = c_n$ with $c_n \neq 0$

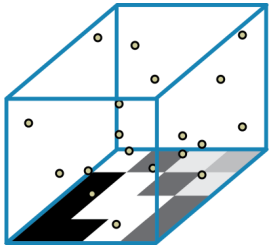
sPDF-map coefficients $(\boldsymbol{p}_n, r_n, c_n)$

Sparse Representation



H

$$H \approx V * (W \otimes K)$$



V

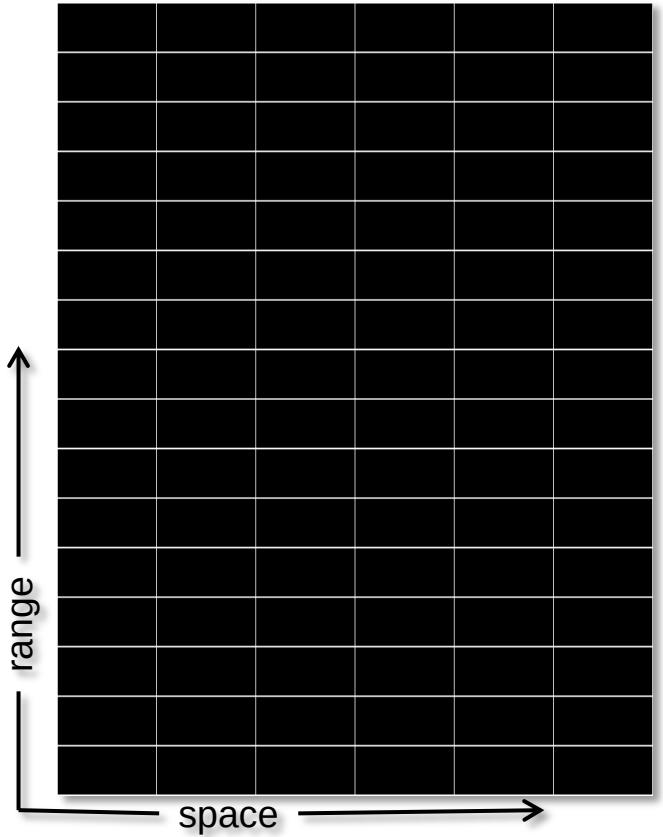
Compute V via Matching Pursuit [Mallat and Zhang 1993]

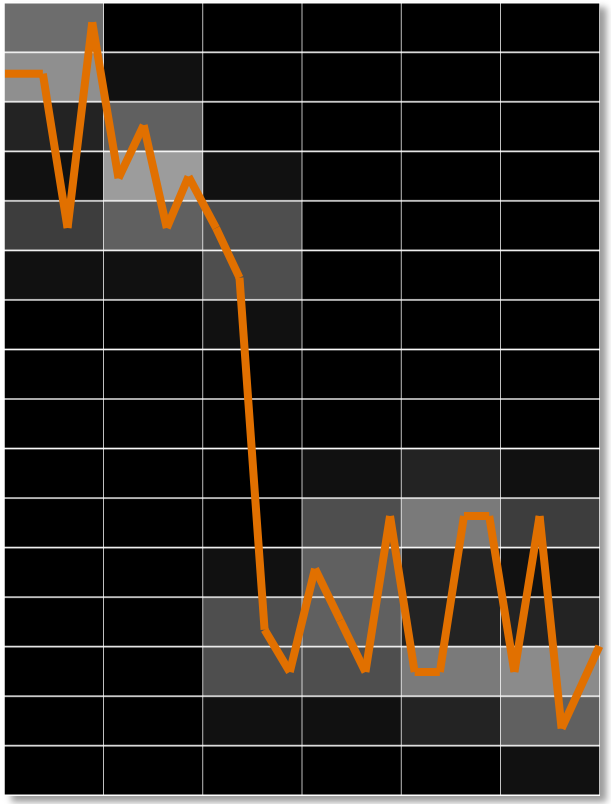
Sparse volume $V(\boldsymbol{p}_n, r_n) = c_n$ with $c_n \neq 0$

sPDF-map coefficients $(\boldsymbol{p}_n, r_n, c_n)$

→ sPDF-maps data structure

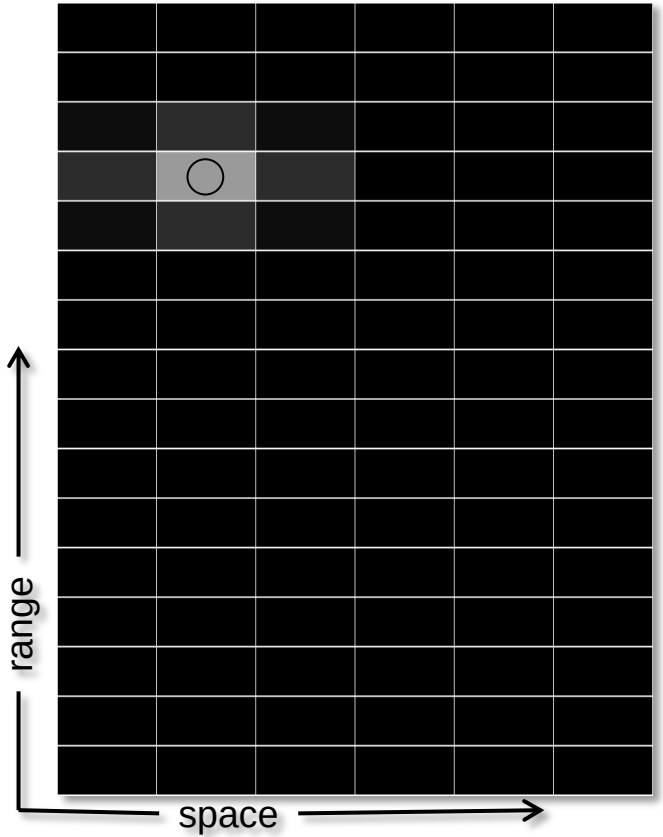
Greedy Approximation

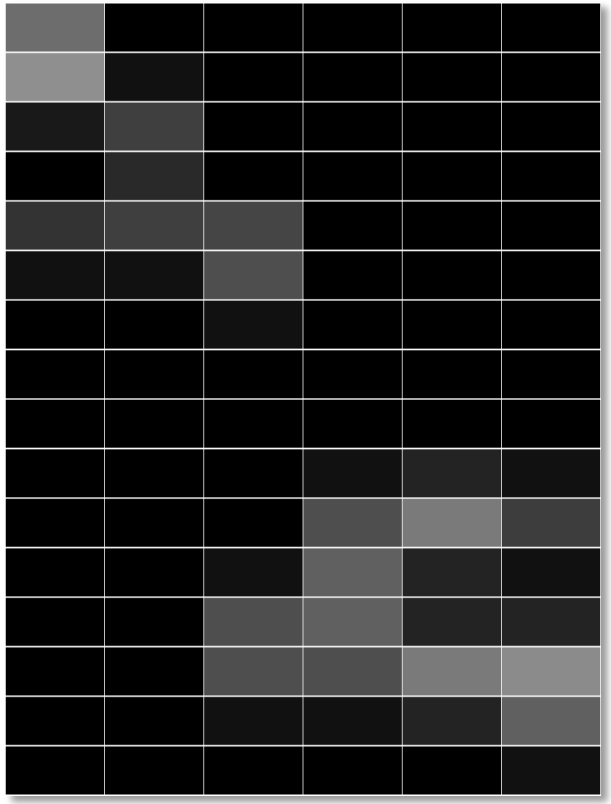




error volume

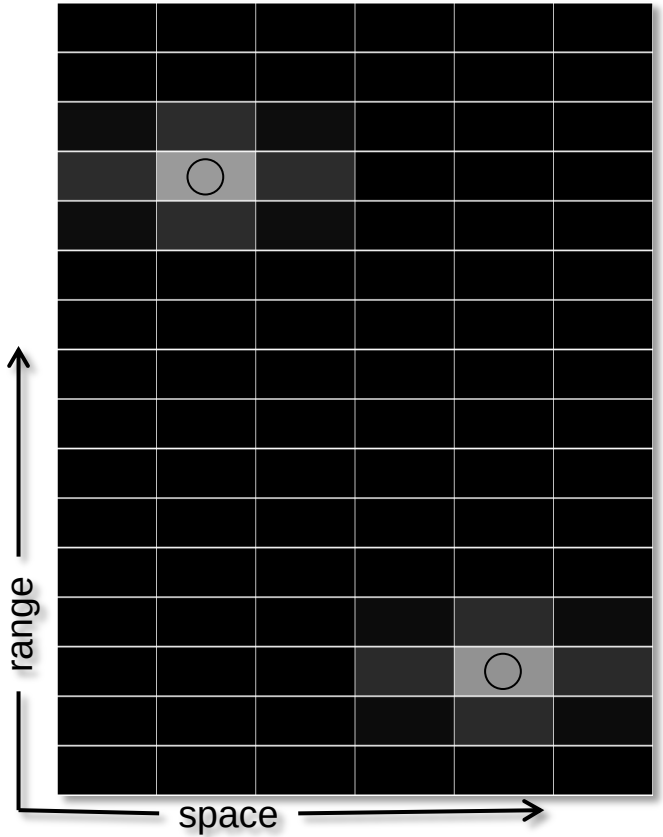
Greedy Approximation

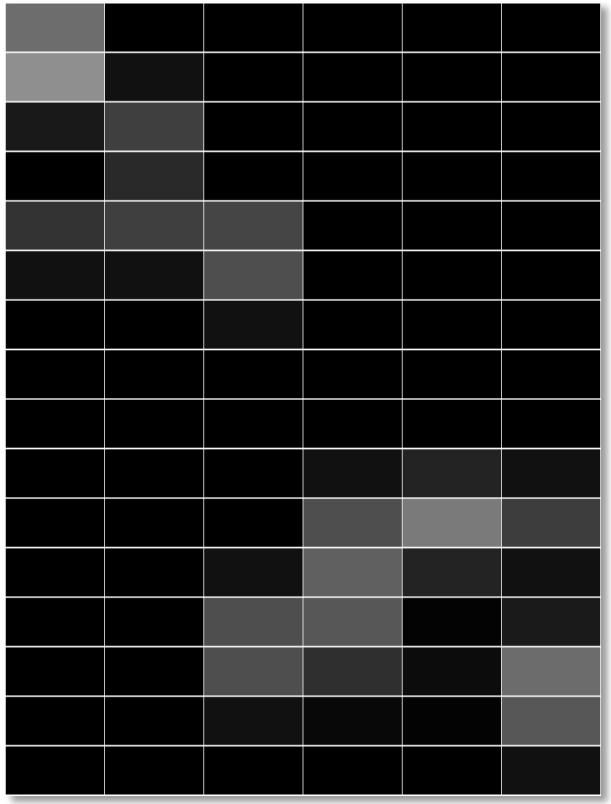




error volume

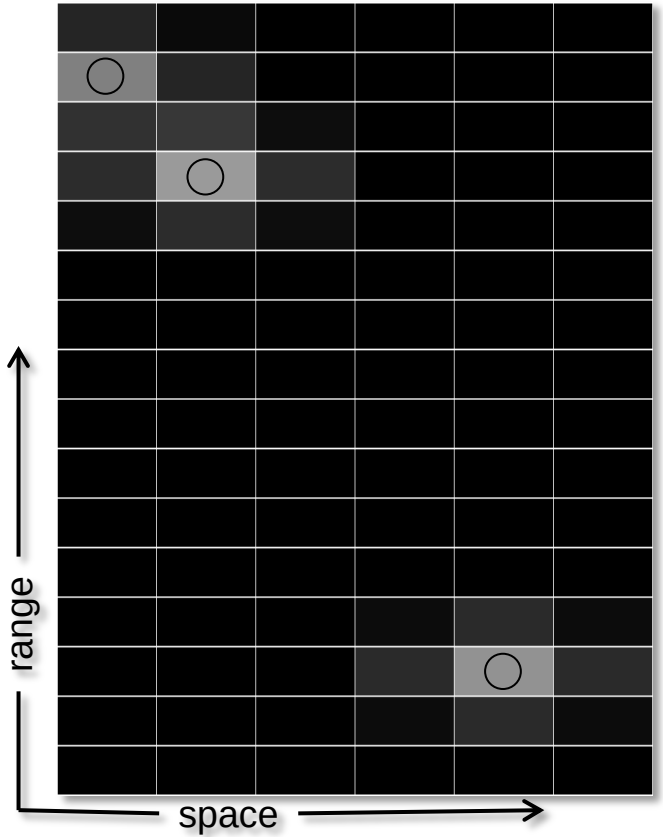
Greedy Approximation

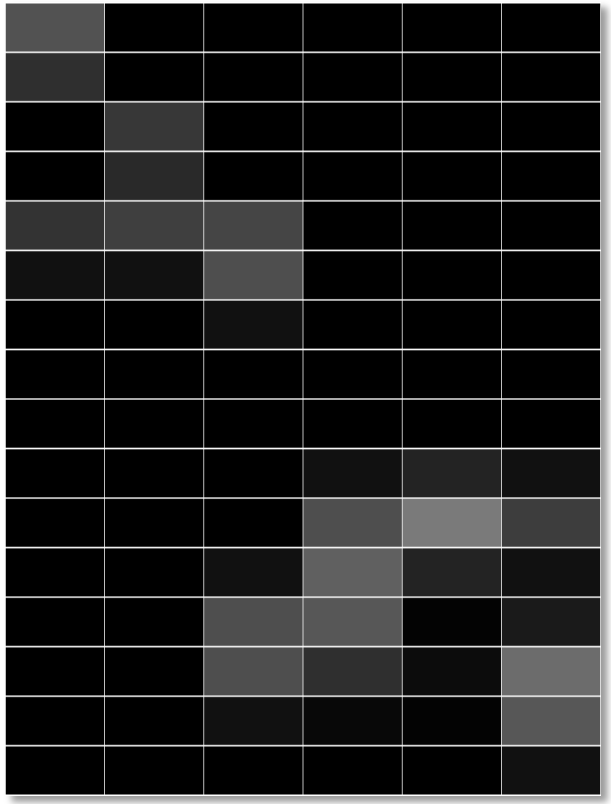




error volume

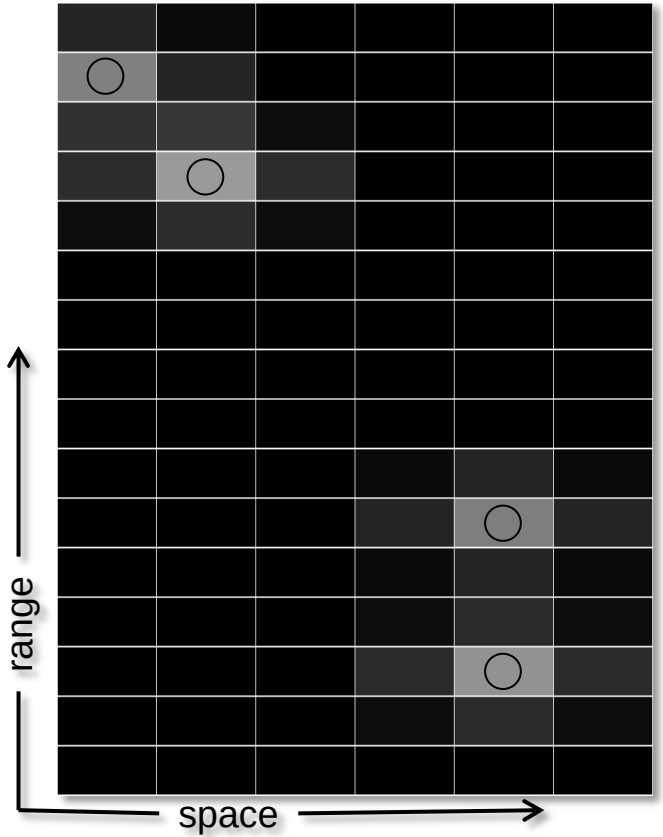
Greedy Approximation

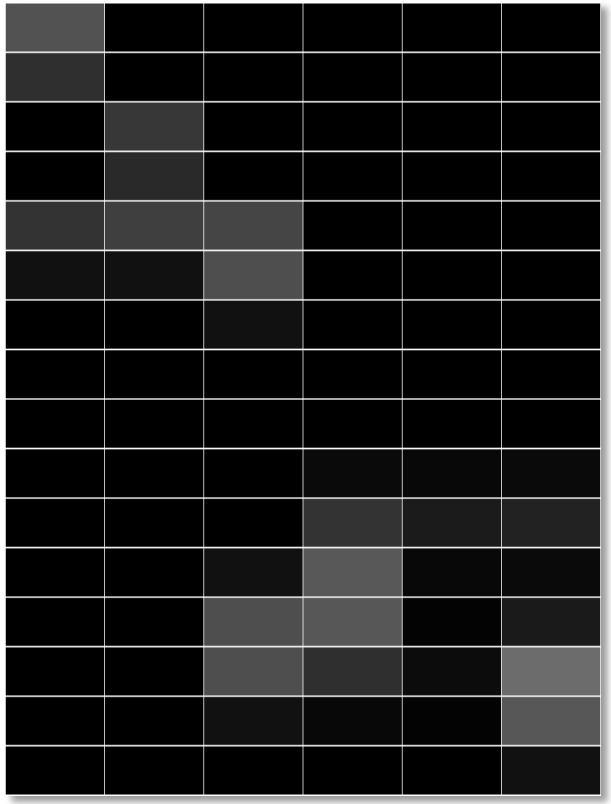




error volume

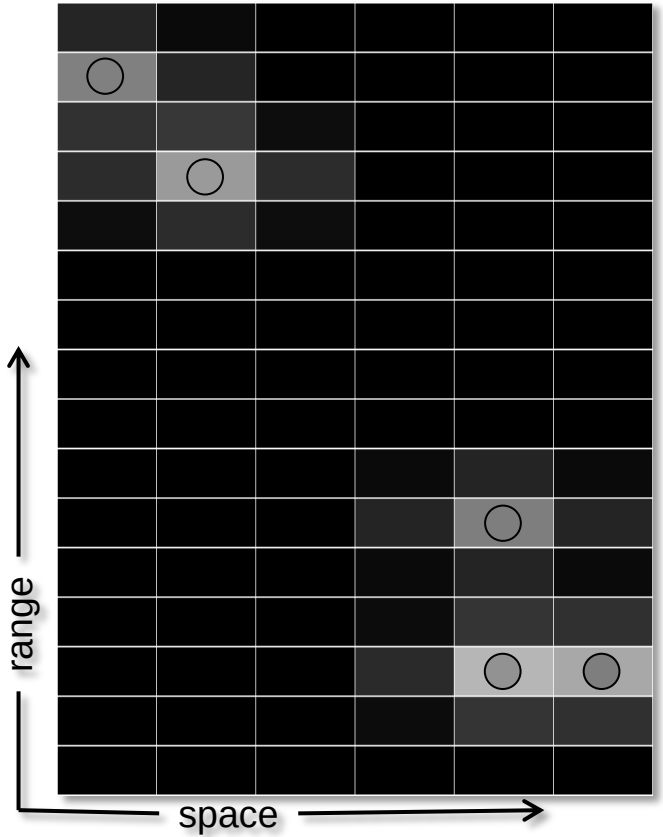
Greedy Approximation

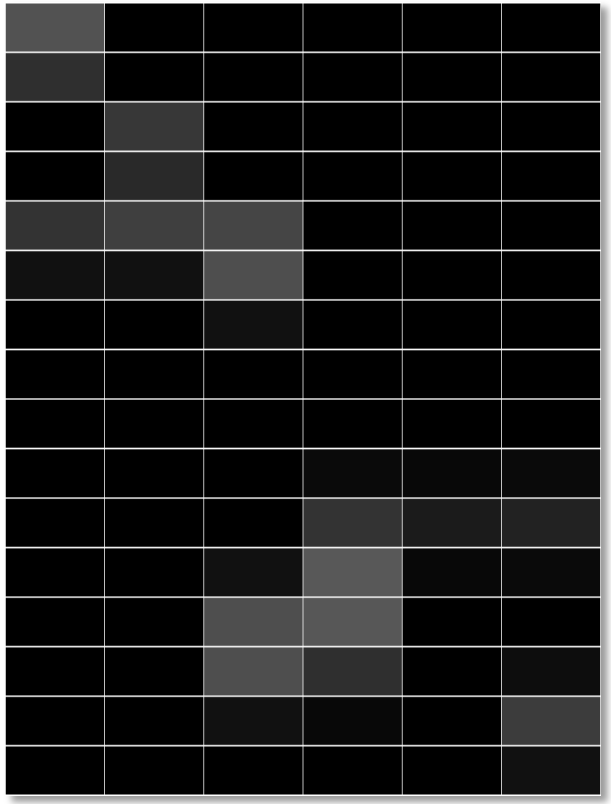




error volume

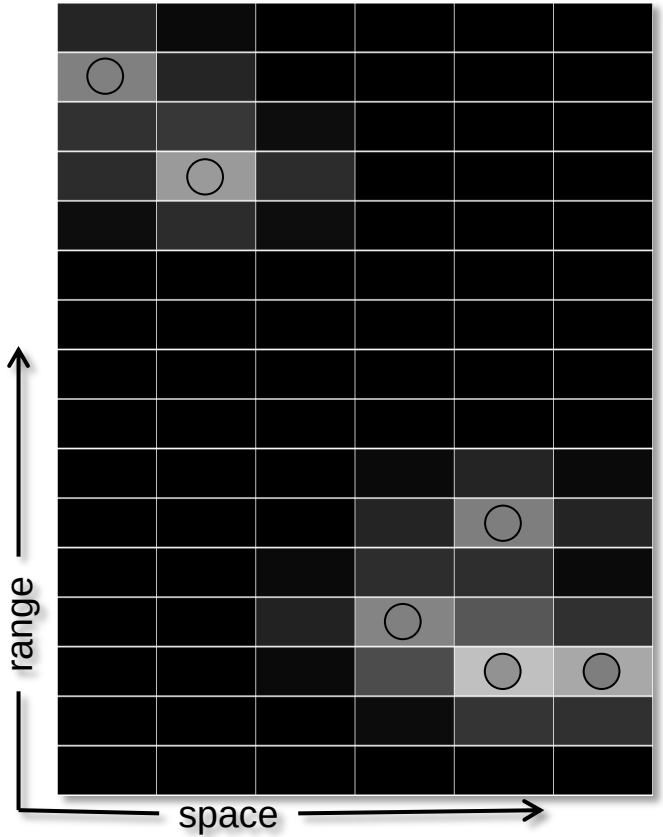
Greedy Approximation

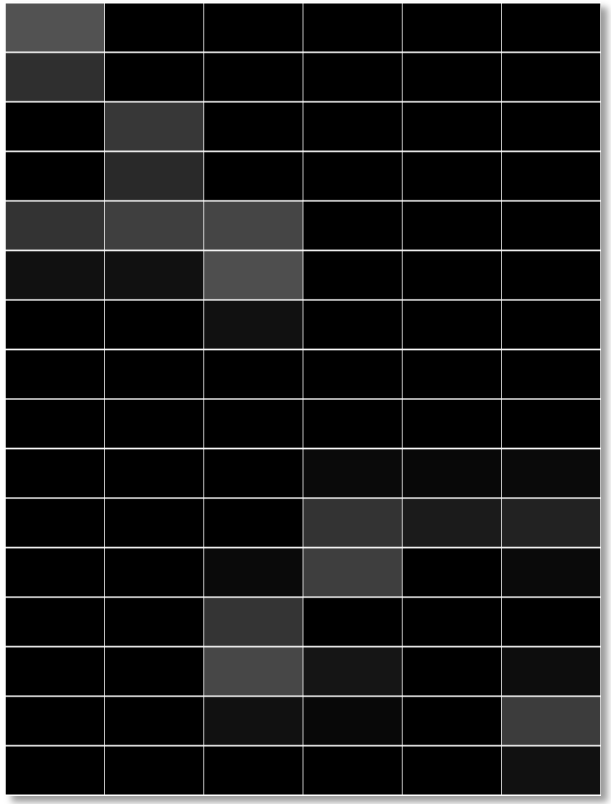




error volume

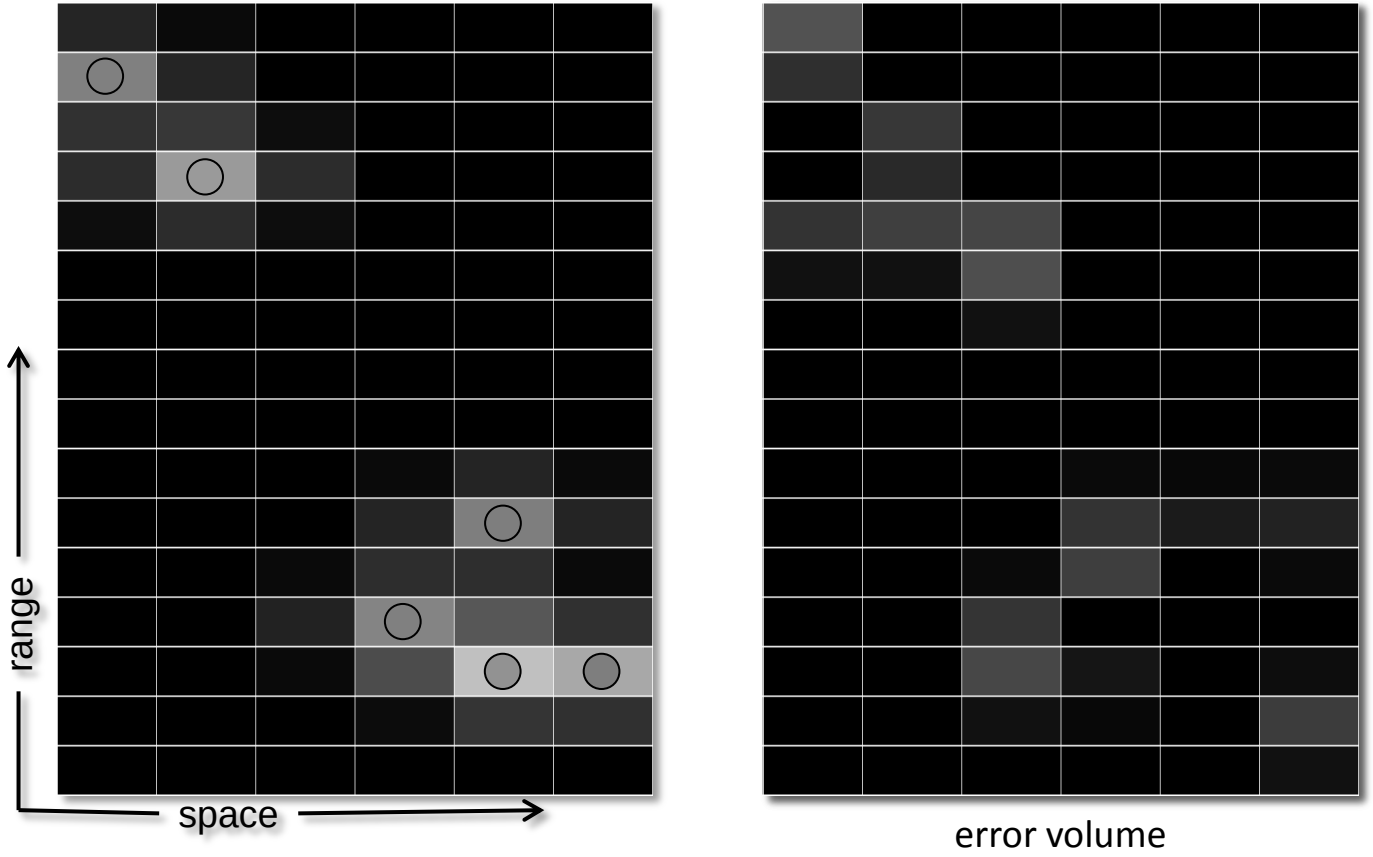
Greedy Approximation





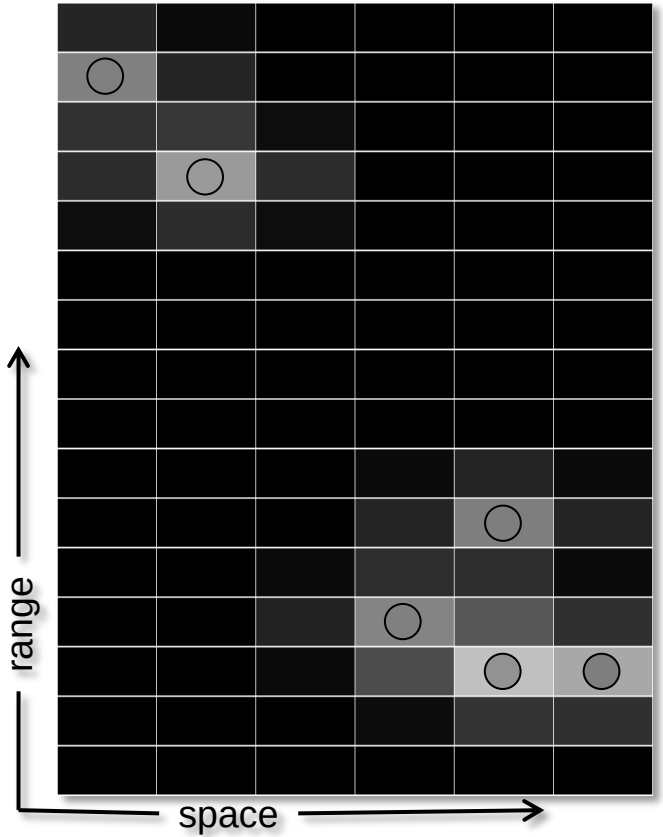
error volume

Greedy Approximation

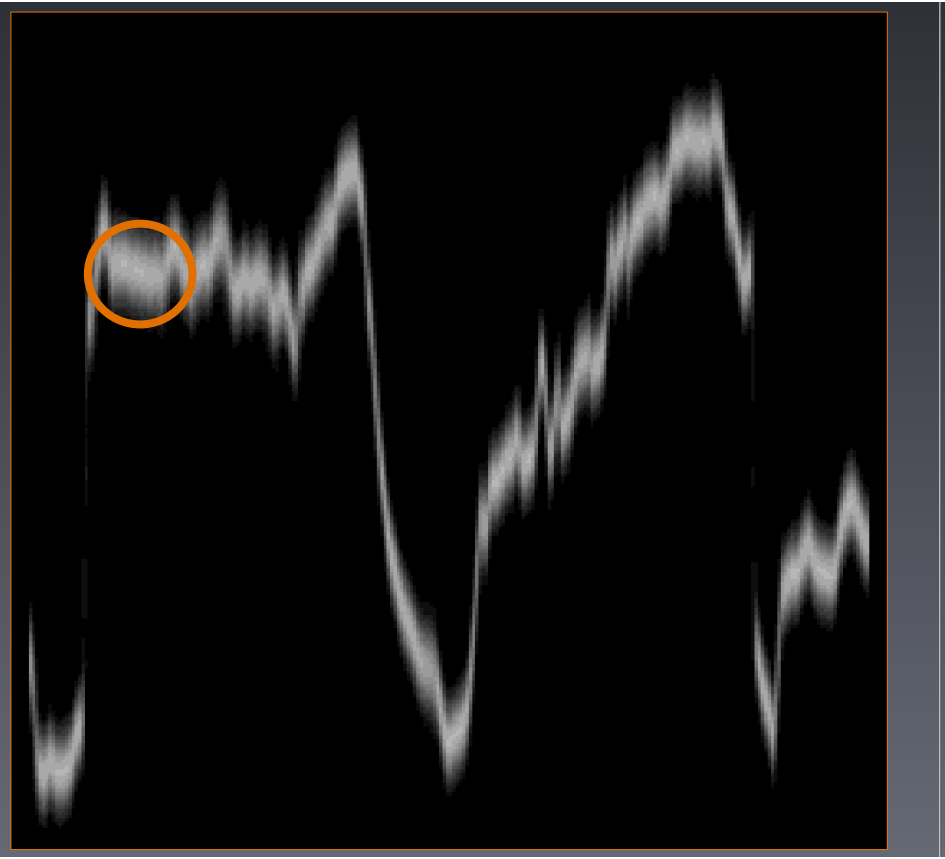


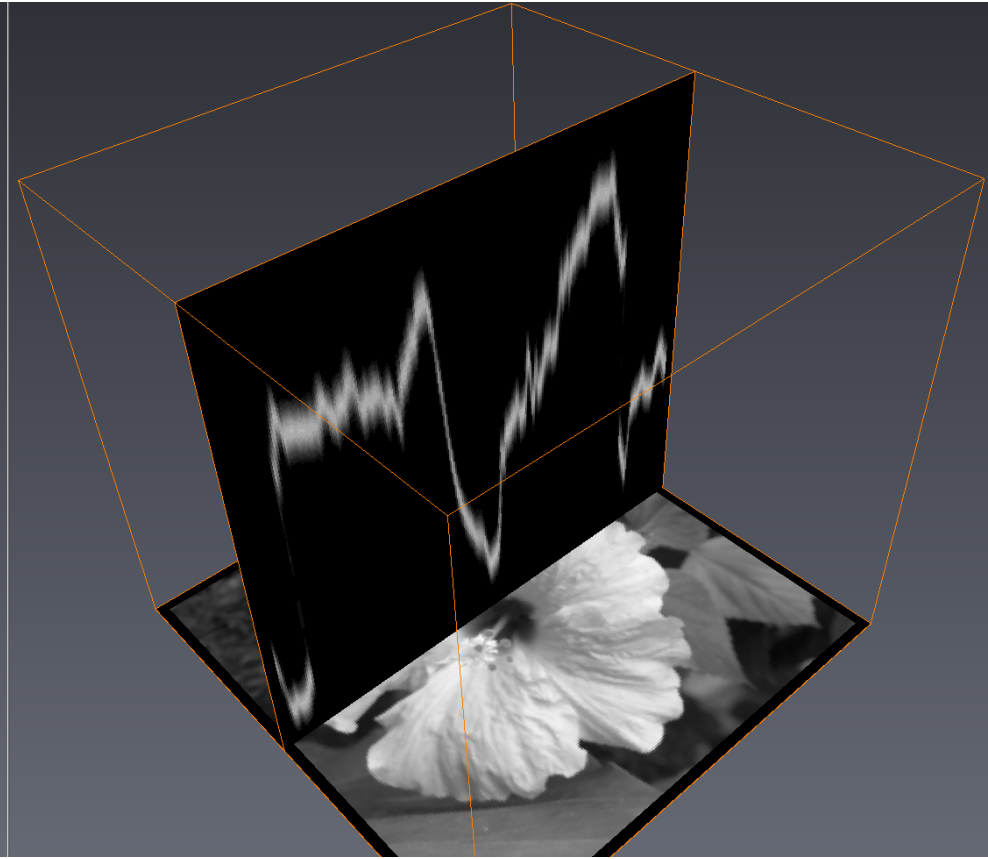
error volume

Greedy Approximation

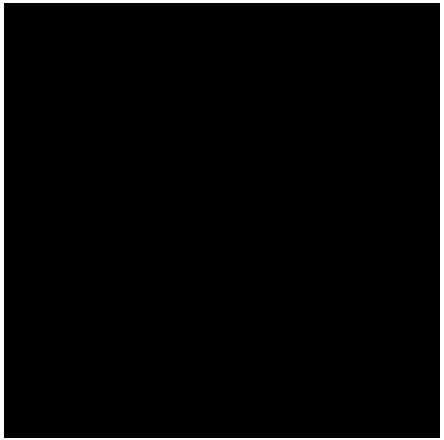


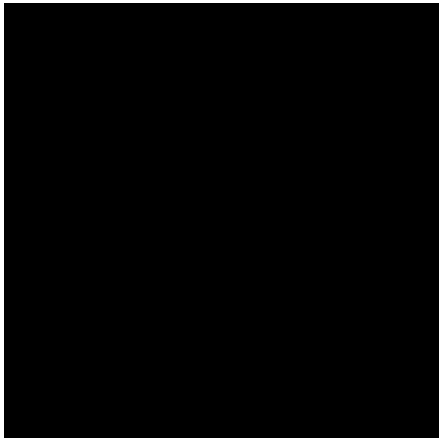
Spatial and Range Coherence

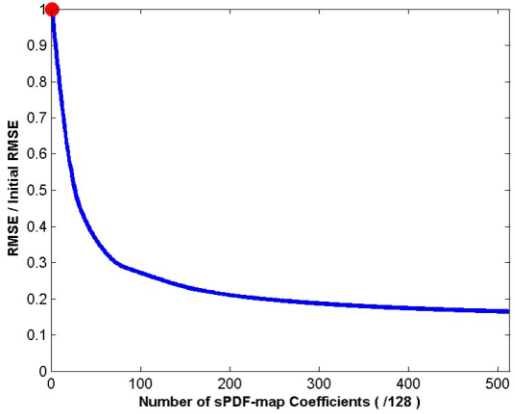


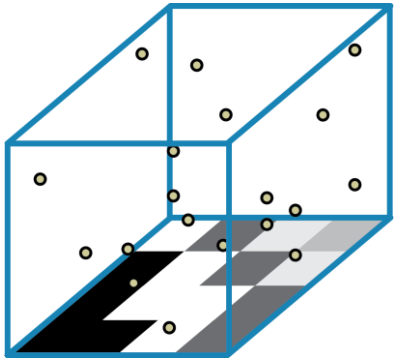


Greedy Approximation



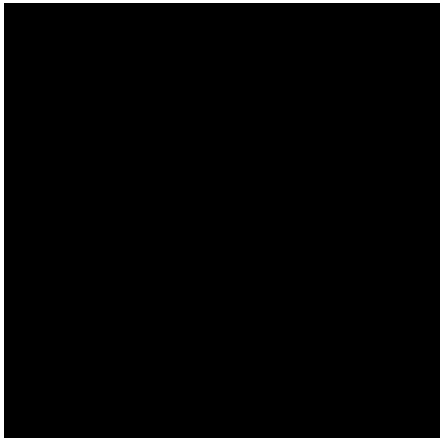


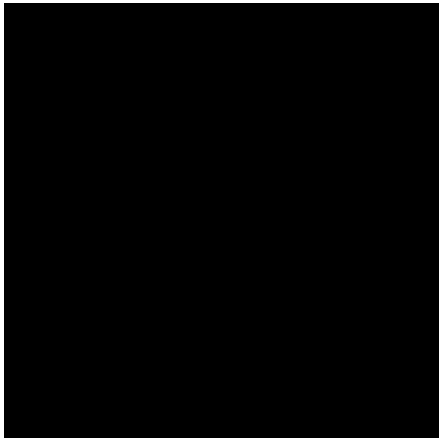


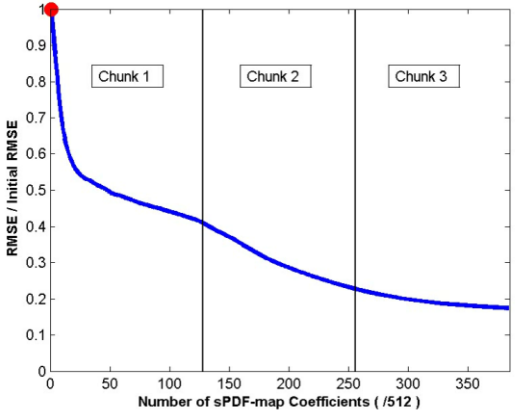


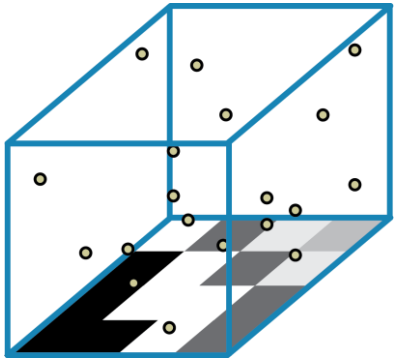
- Spatial filter W : 5 x 5
- 1 coefficient chunk
  (# coefficients == # pixels)

Greedy Approximation



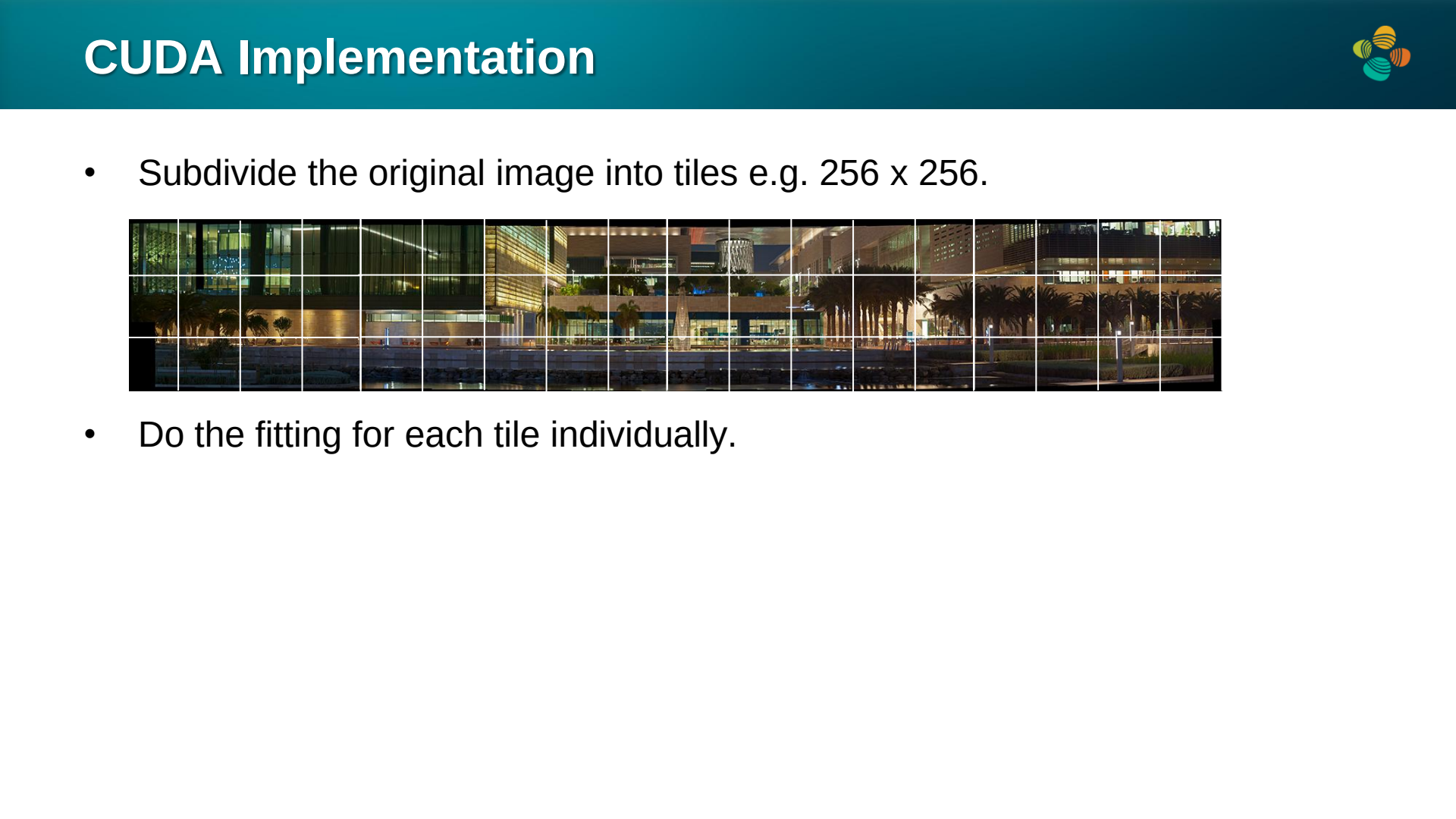






- Spatial filter W : 3 x 3
- 1-3 coefficient chunks
  (# coefficients == 1-3 * # pixels)

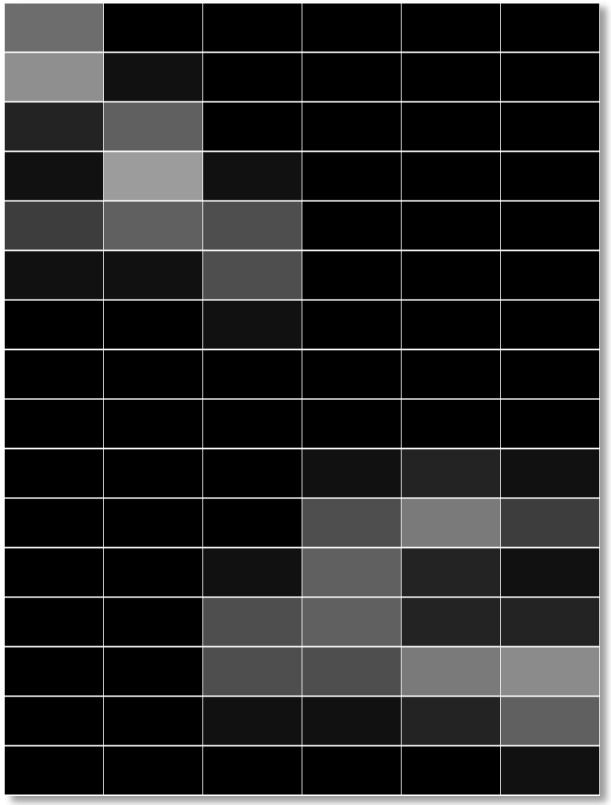
CUDA Implementation



- Subdivide the original image into tiles e.g. 256 x 256.

- Do the fitting for each tile individually.

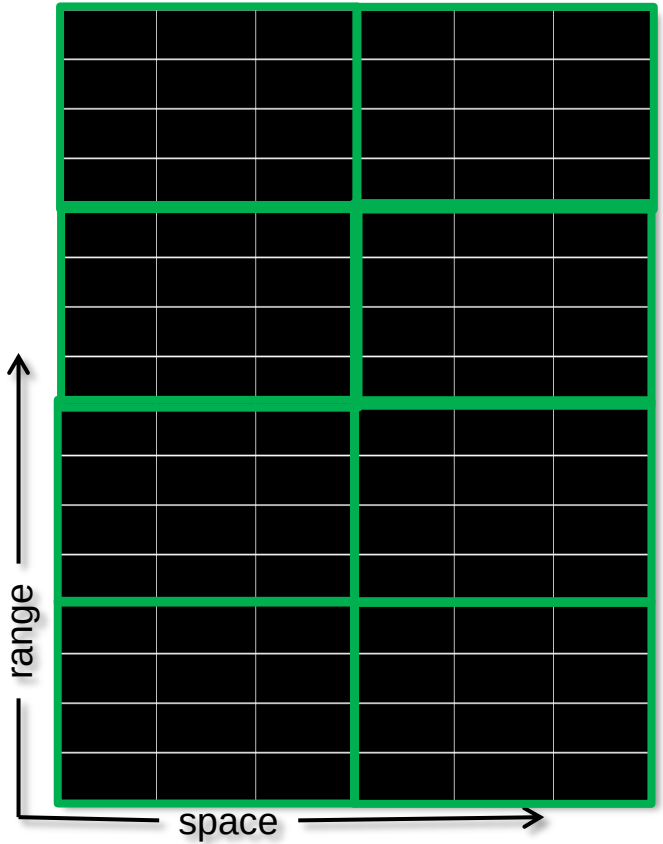
CUDA Implementation

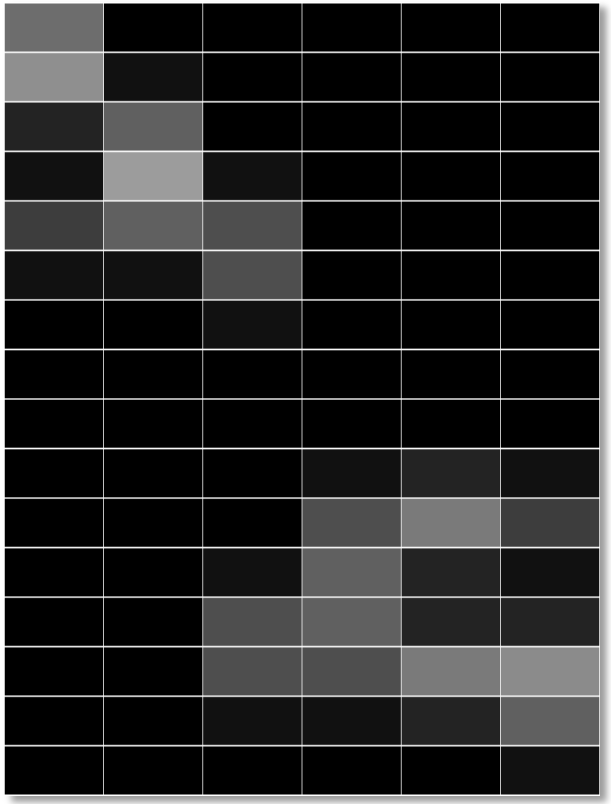


error volume

1. Copy error volume to global memory.

CUDA Implementation

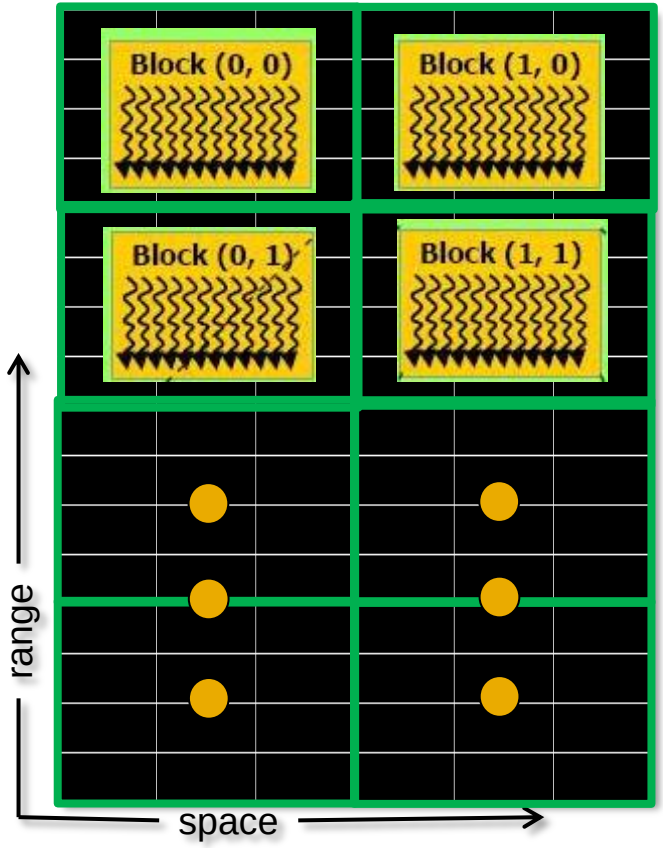


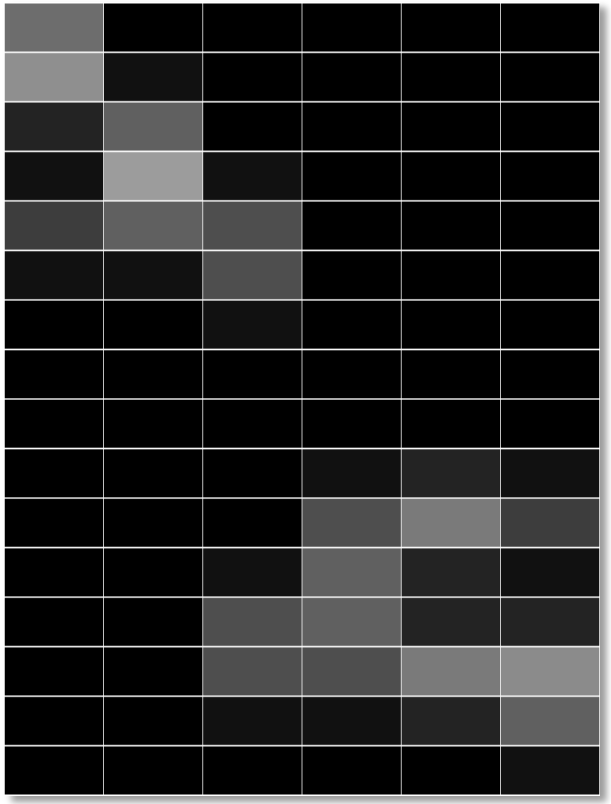


error volume

2. Subdivide the space X range domain into sub-regions e.g. 8 x 8 x 16 regions.

CUDA Implementation

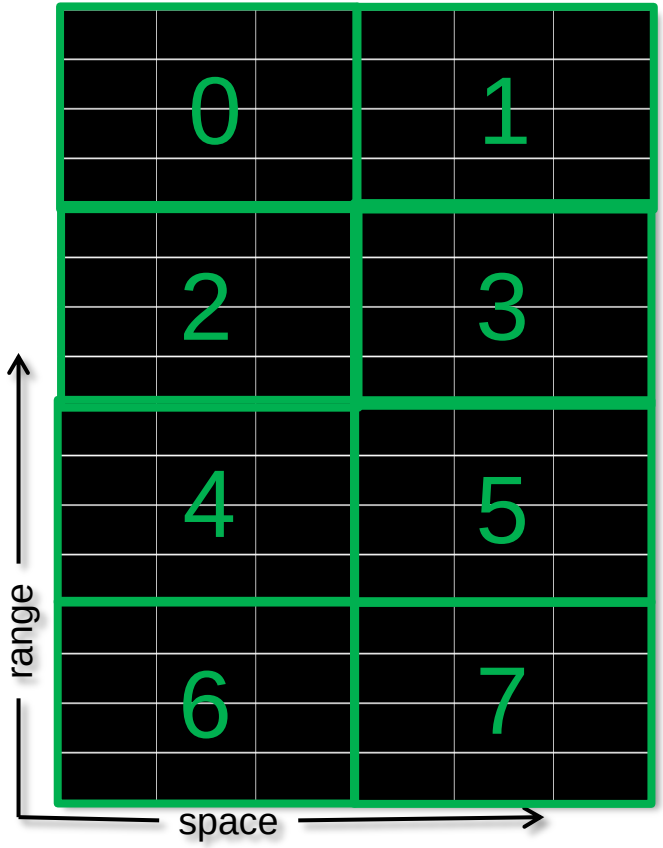


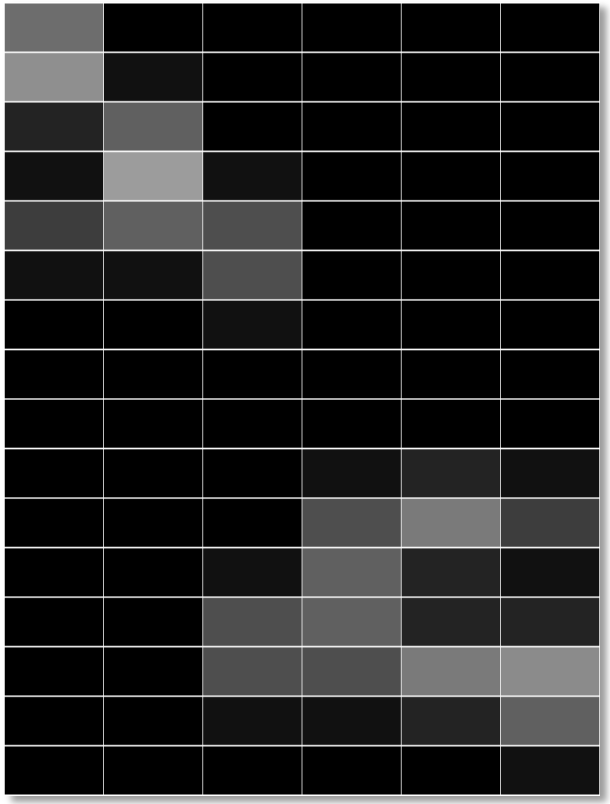


error volume

3. Compute the coefficient of all basis functions and find the maximum for each sub-region.

CUDA Implementation



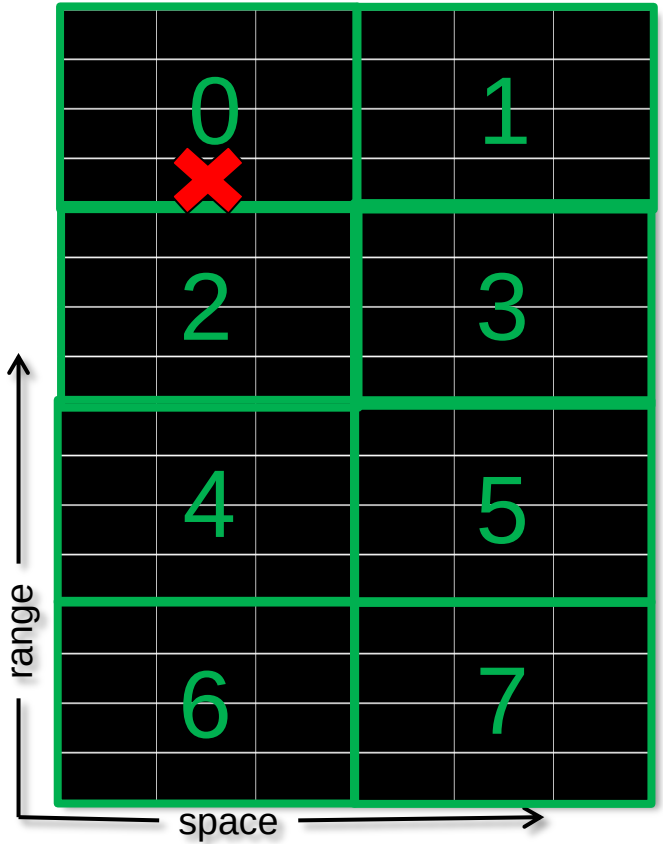


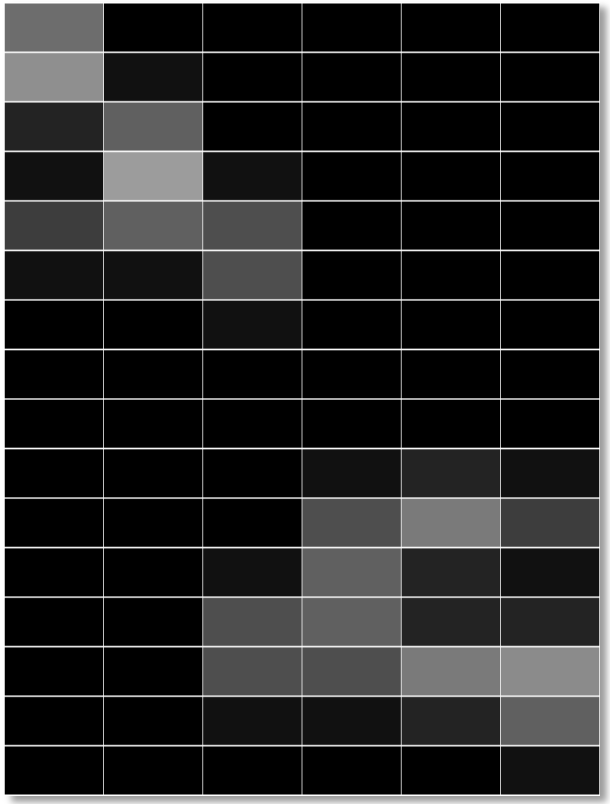
error volume

max 0	max 1
max 2	max 3
max 4	max 5
max 6	max 7

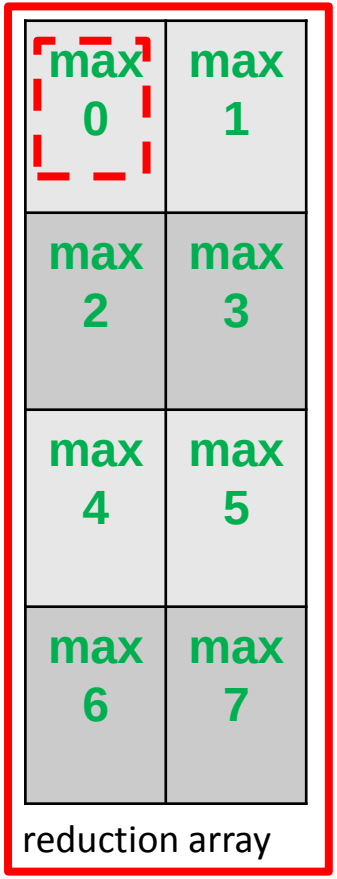
reduction array

CUDA Implementation



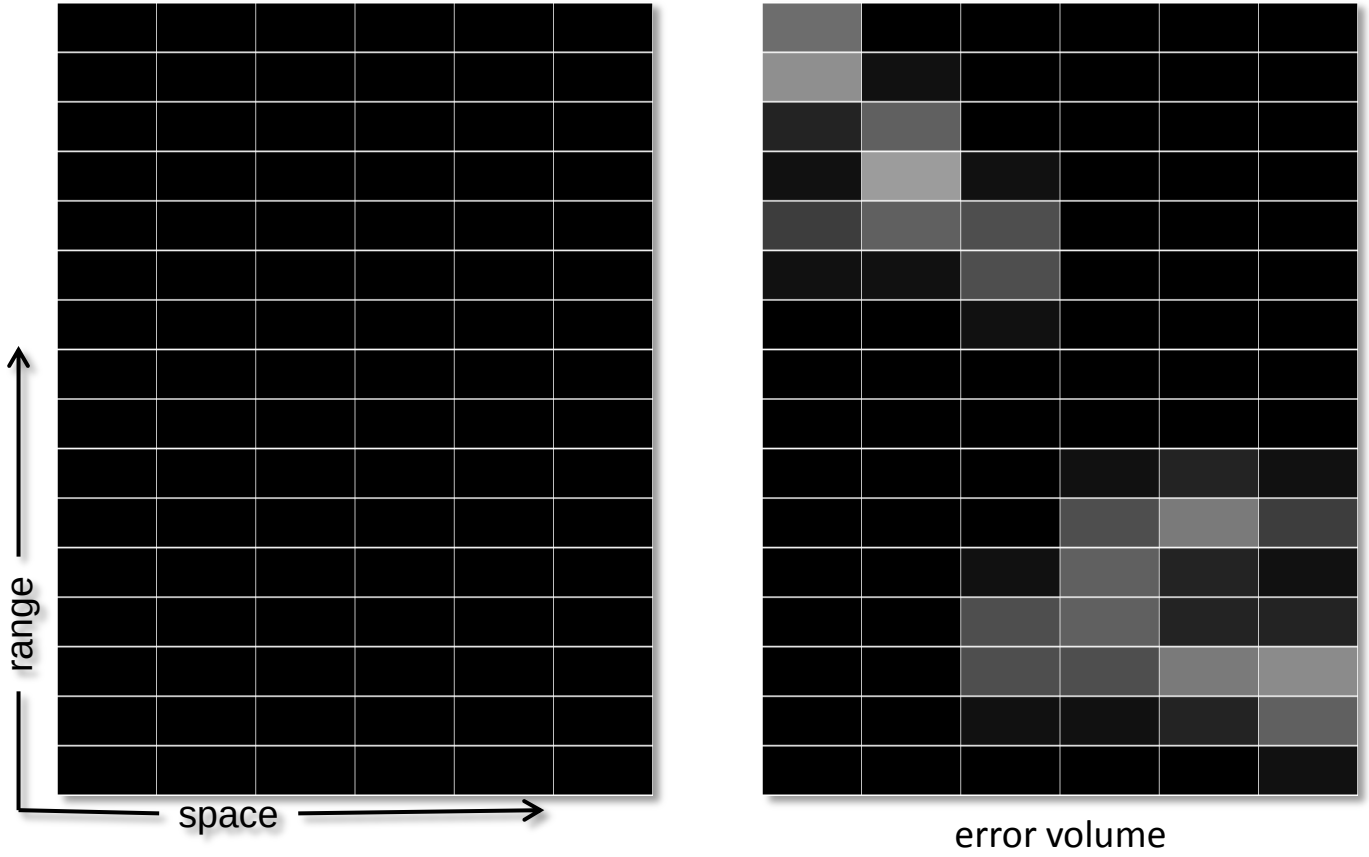


error volume



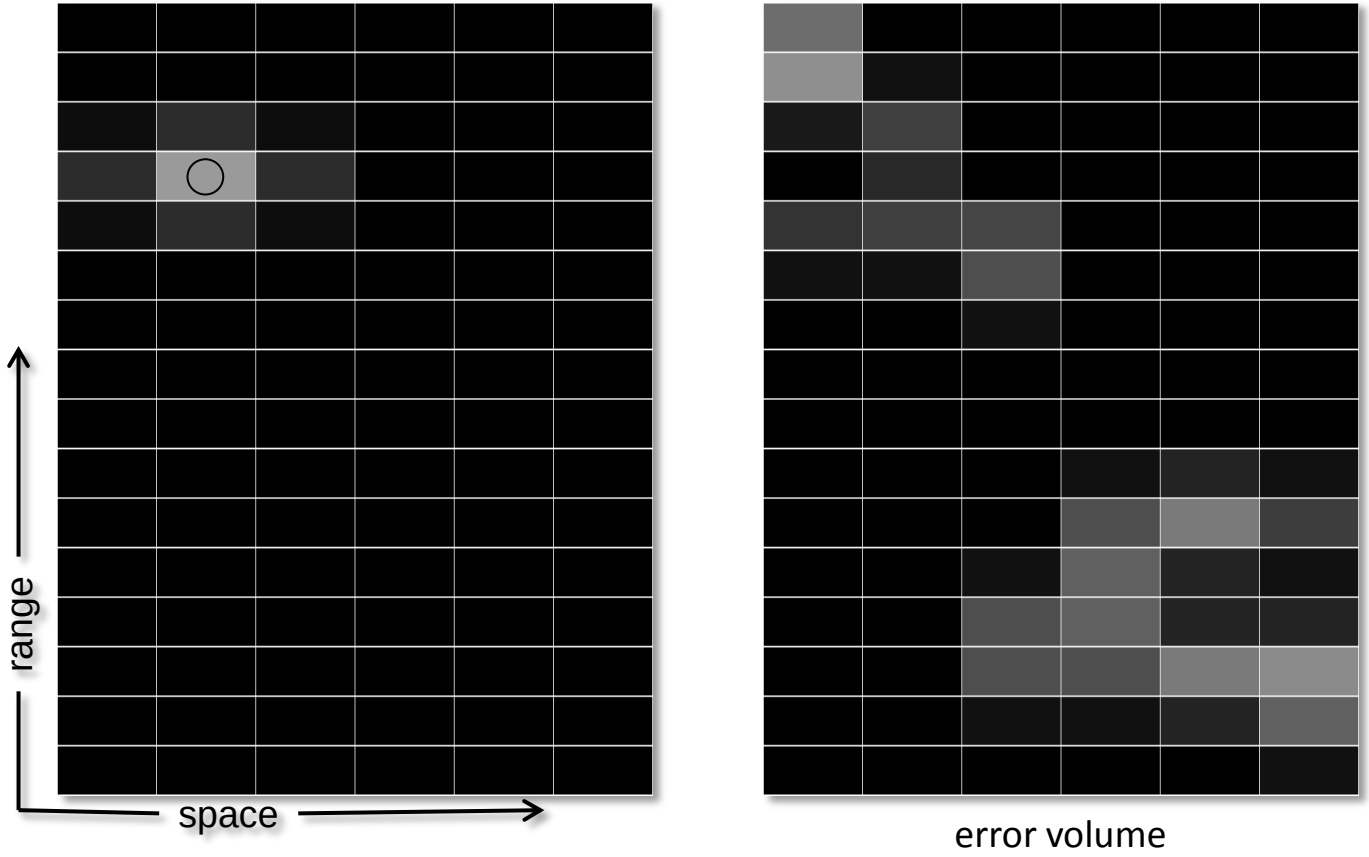
reduction array

CUDA Implementation



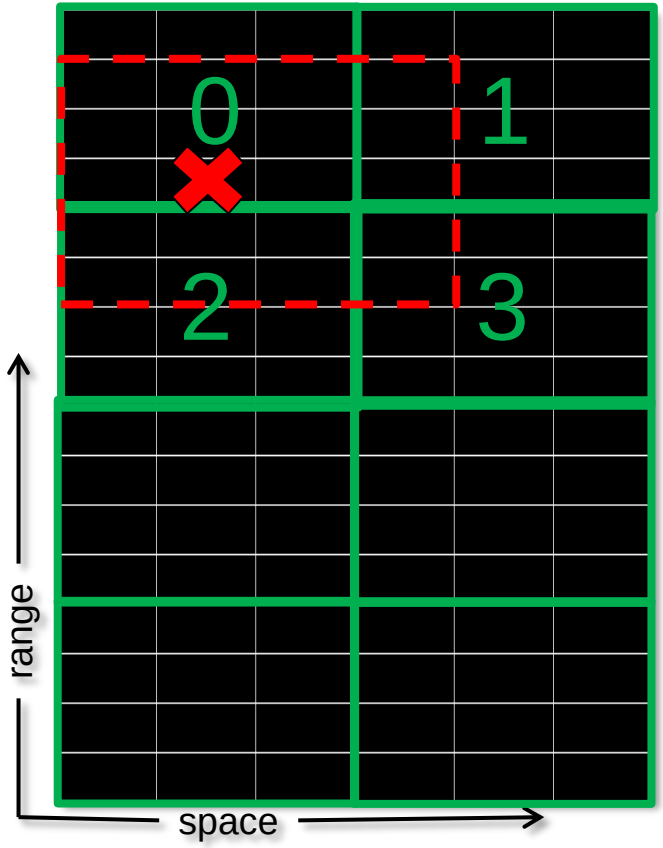
0 coefficients

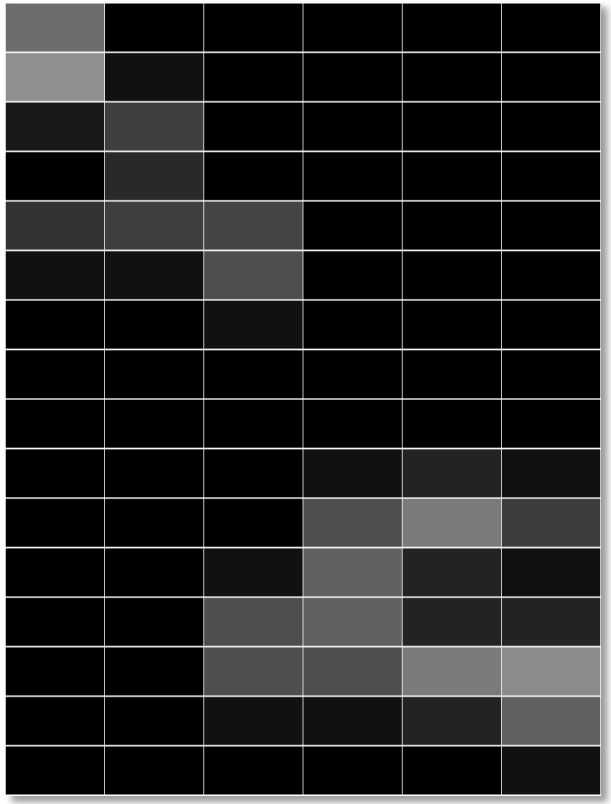
Greedy Approximation



1 coefficient

CUDA Implementation





error volume

max 0	max 1
max 2	max 3
max 4	max 5
max 6	max 7

reduction array

CPU (Intel Xeon X5675) vs GPU (GTX 680)

error volume # bins in range	CPU (single thread)	CPU (24 threads)	GPU	RMSE H vs V * (WxK)
256 bins	3,783 seconds	302 seconds	68 seconds	0.0017
16 bins	519 seconds	50 seconds	27 seconds	0.0019

- results are for fitting on a 256 x 256 histogram volume tile with 1 coefficient chunk using a 5 x 5 x 33 Gaussian basis function.
- scales linearly – fitting time is more or less constant for each tile.

Expected Value Image Comparisons

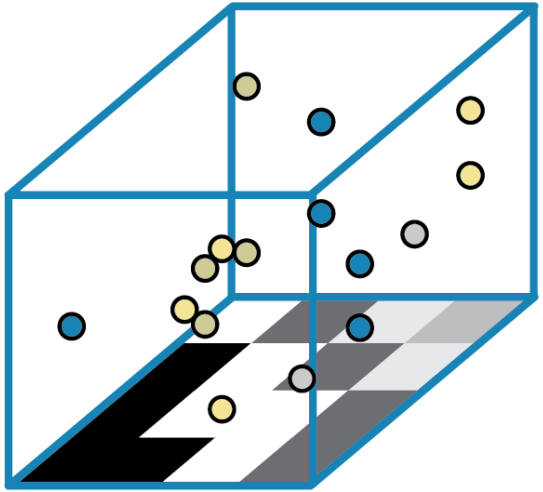
dPDF
(histogram volume)

sPDF
(256 bin residue)

sPDF
(16 bin residue)

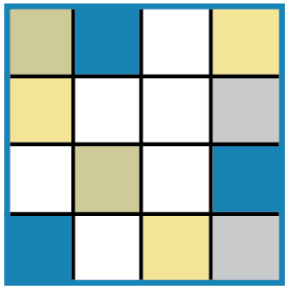
sPDF-Maps Data Structure

sPDF-Maps Data Structure



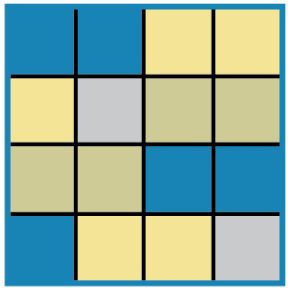
conceptual

$$V(\boldsymbol{p}_n, r_n) = c_n$$



index image

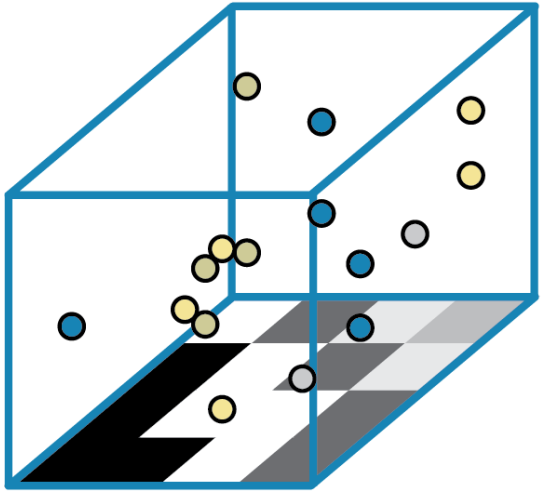
$$(index, count)_{\boldsymbol{p}}$$



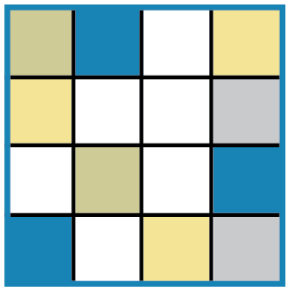
coefficient image

$$(r_n, c_n)$$

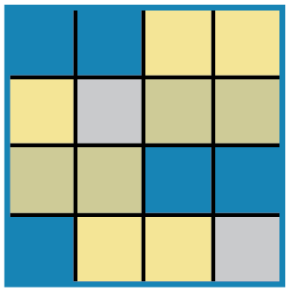
sPDF-Maps Data Structure



conceptual



index image



coefficient image

$$V(\boldsymbol{p}_n, r_n) = c_n$$

$$(index, count)_{\boldsymbol{p}}$$

$$(r_n, c_n)$$

Display-Aware Gigapixel Image Processing

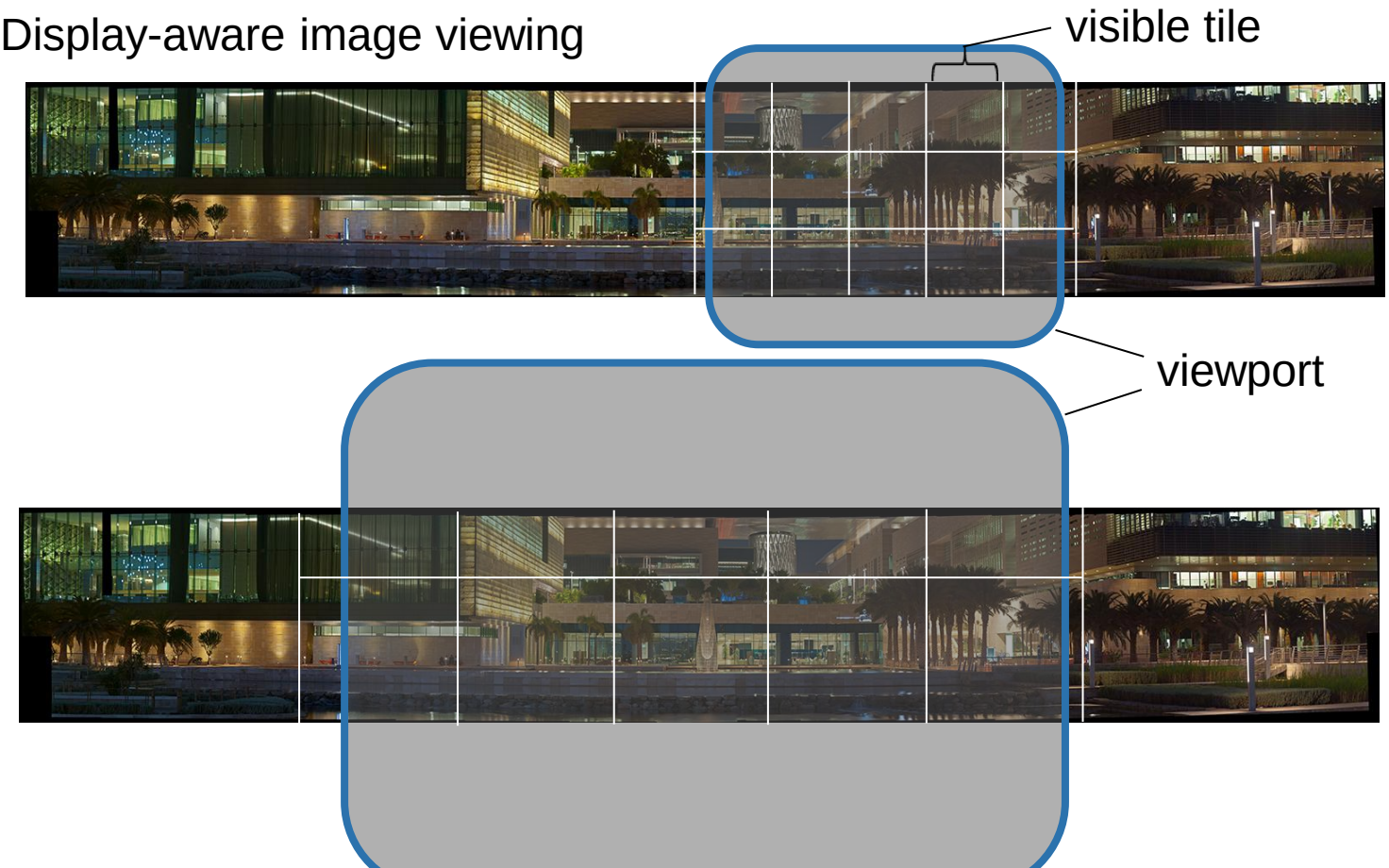
Gigapixel Image Processing

- Out-of-Core Processing
  - Divide data into smaller tiles, process each tile independently
  - Reduce memory footprint
  - Scalable approach

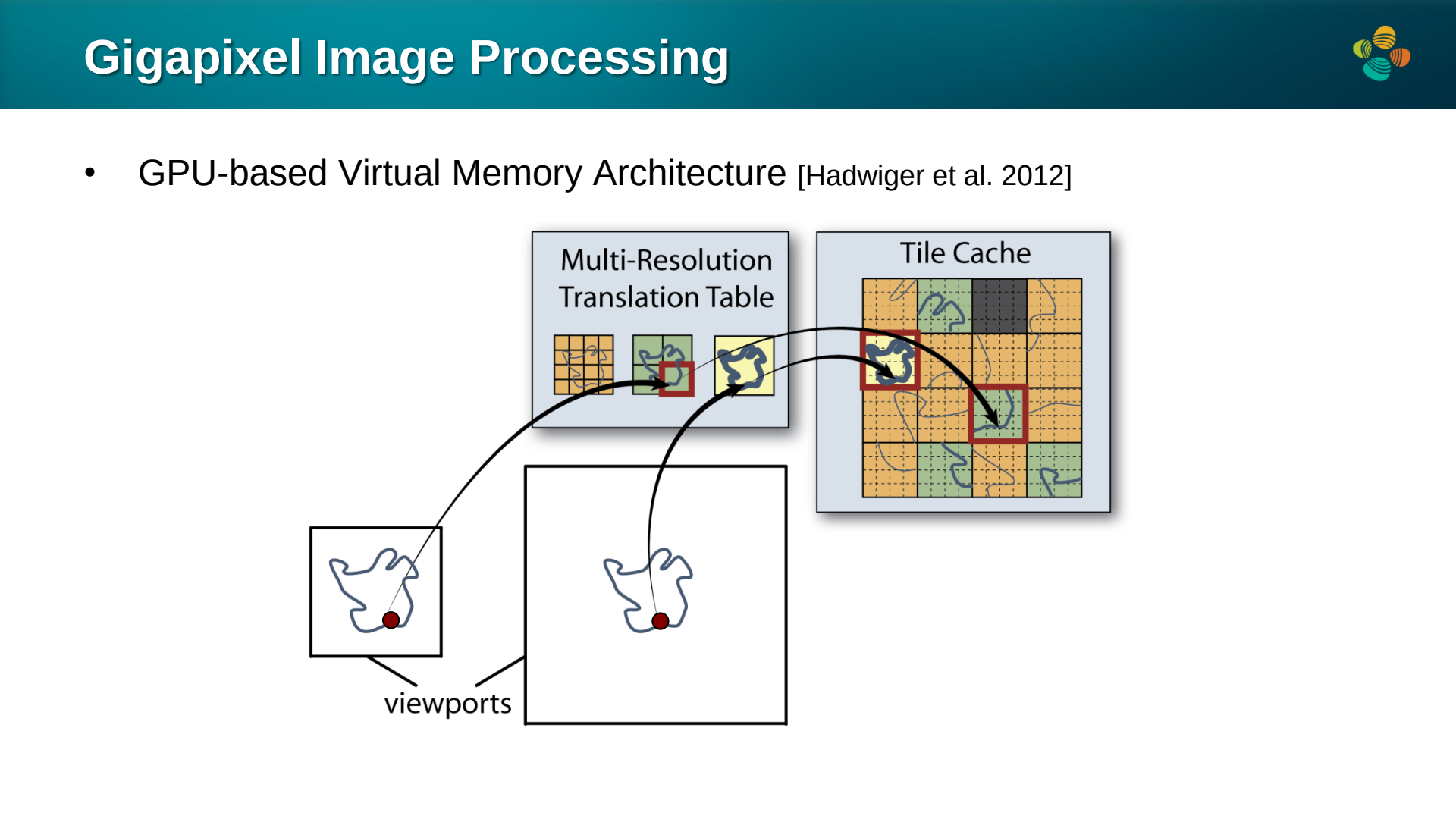
- Our processing strategy
  - Pre-processor creates image sub-tiles (e.g., 256x256)
  - Image operations are performed only on requested sub-tiles (display-aware)
  - Rendering based on tiled data, using a GPU-based virtual memory approach

Gigapixel Image Processing

- Display-aware image viewing



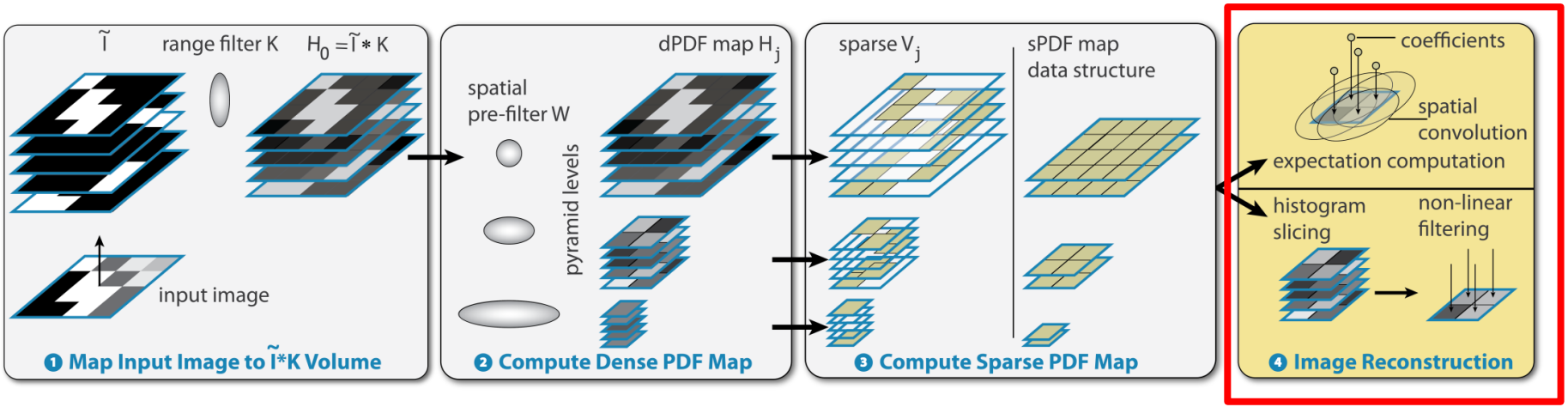
Gigapixel Image Processing



- GPU-based Virtual Memory Architecture [Hadwiger et al. 2012]

Image Reconstruction

Pipeline



pre-computation

run time

Image Reconstruction

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r) pdf_{\boldsymbol{p}}(r)\, dr$$

- Key idea # 1

  Use $V * (W \otimes K)$ instead of $pdf_{\boldsymbol{p}}(r)$

Image Reconstruction

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r)\, pdf_{\boldsymbol{p}}(r)\, dr$$

- Key idea # 1

  Use $V * (W \otimes K)$ instead of $pdf_{\boldsymbol{p}}(r)$

Image Reconstruction

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r)\left(V * (W \otimes K)\right) dr$$

- Key idea # 1

  Use $V * (W \otimes K)$ instead of $pdf_{\boldsymbol{p}}(r)$

Image Reconstruction

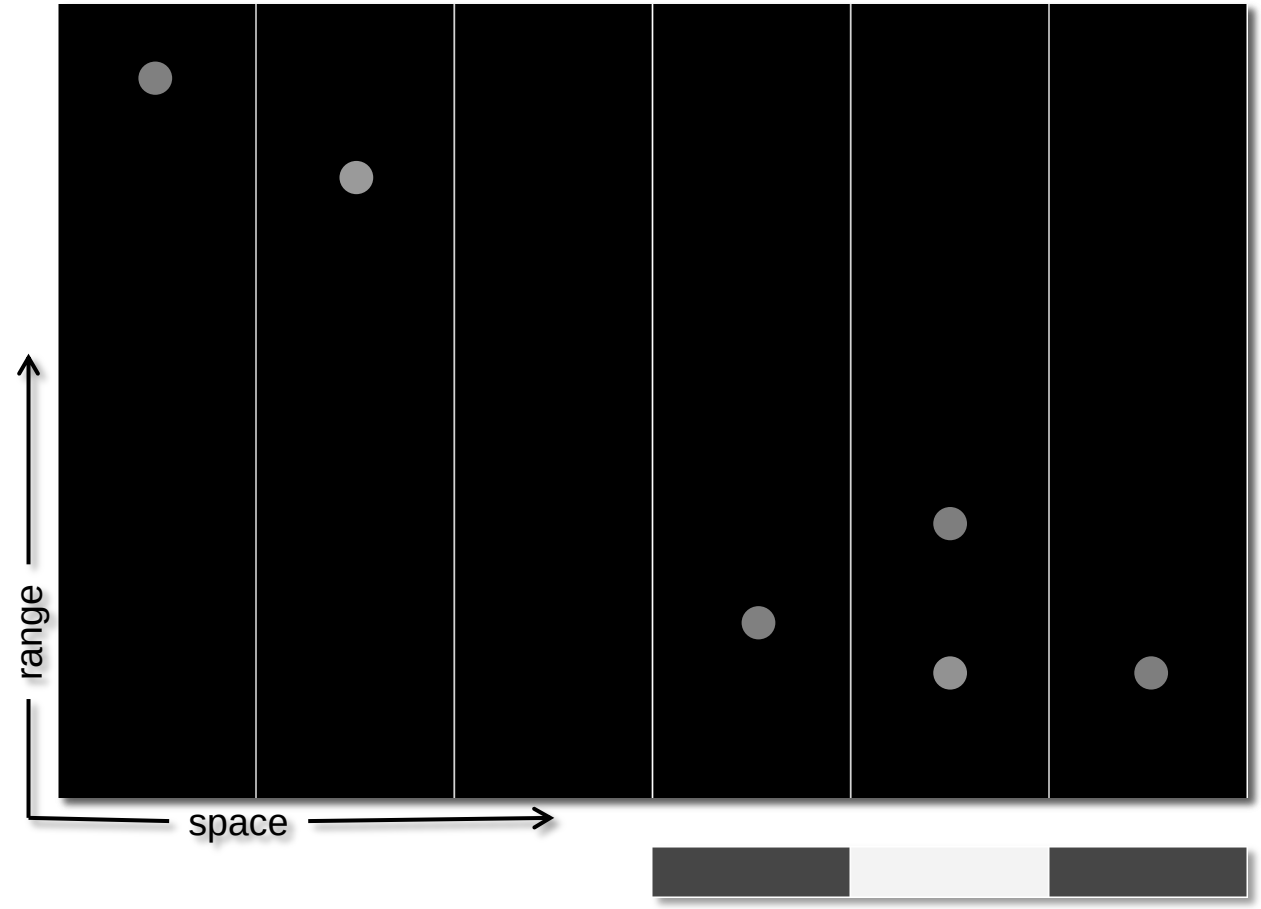


Image Reconstruction

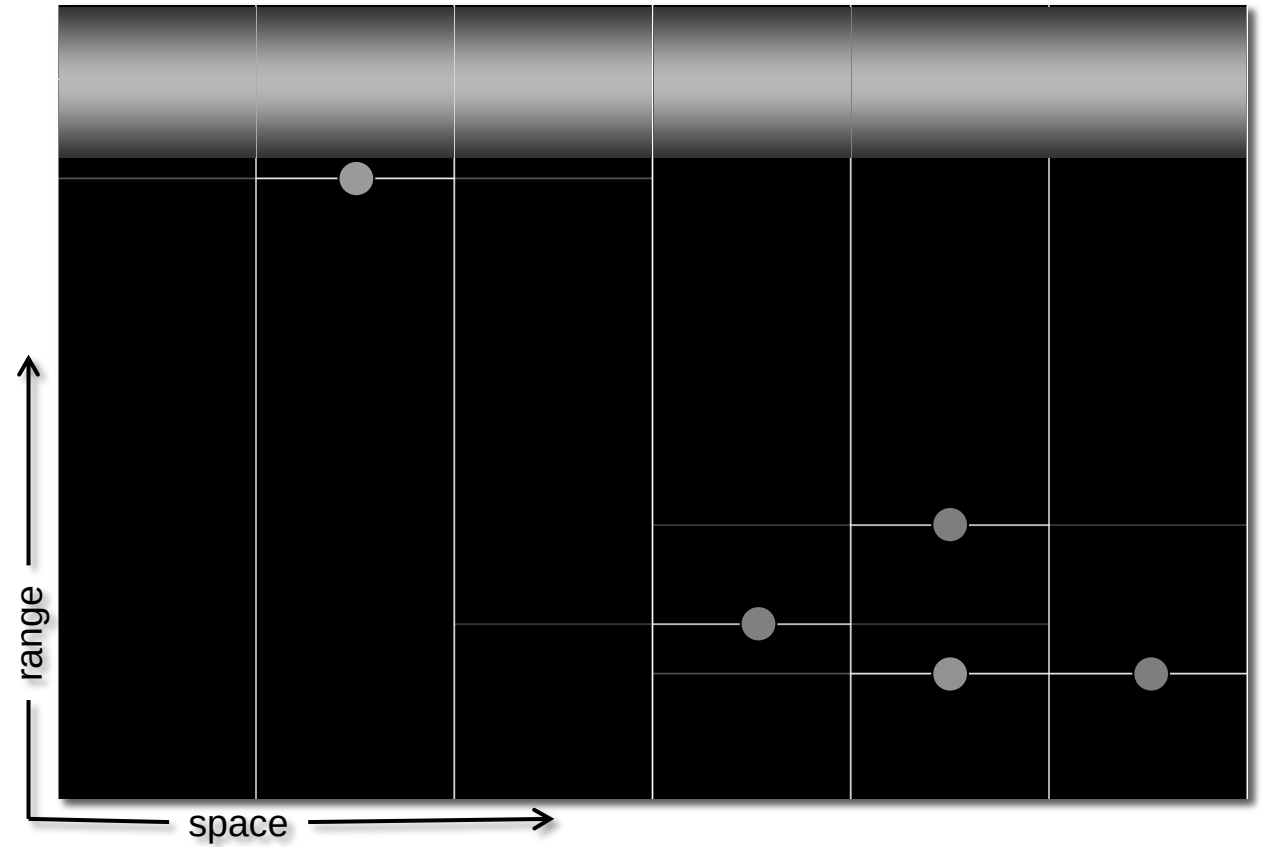
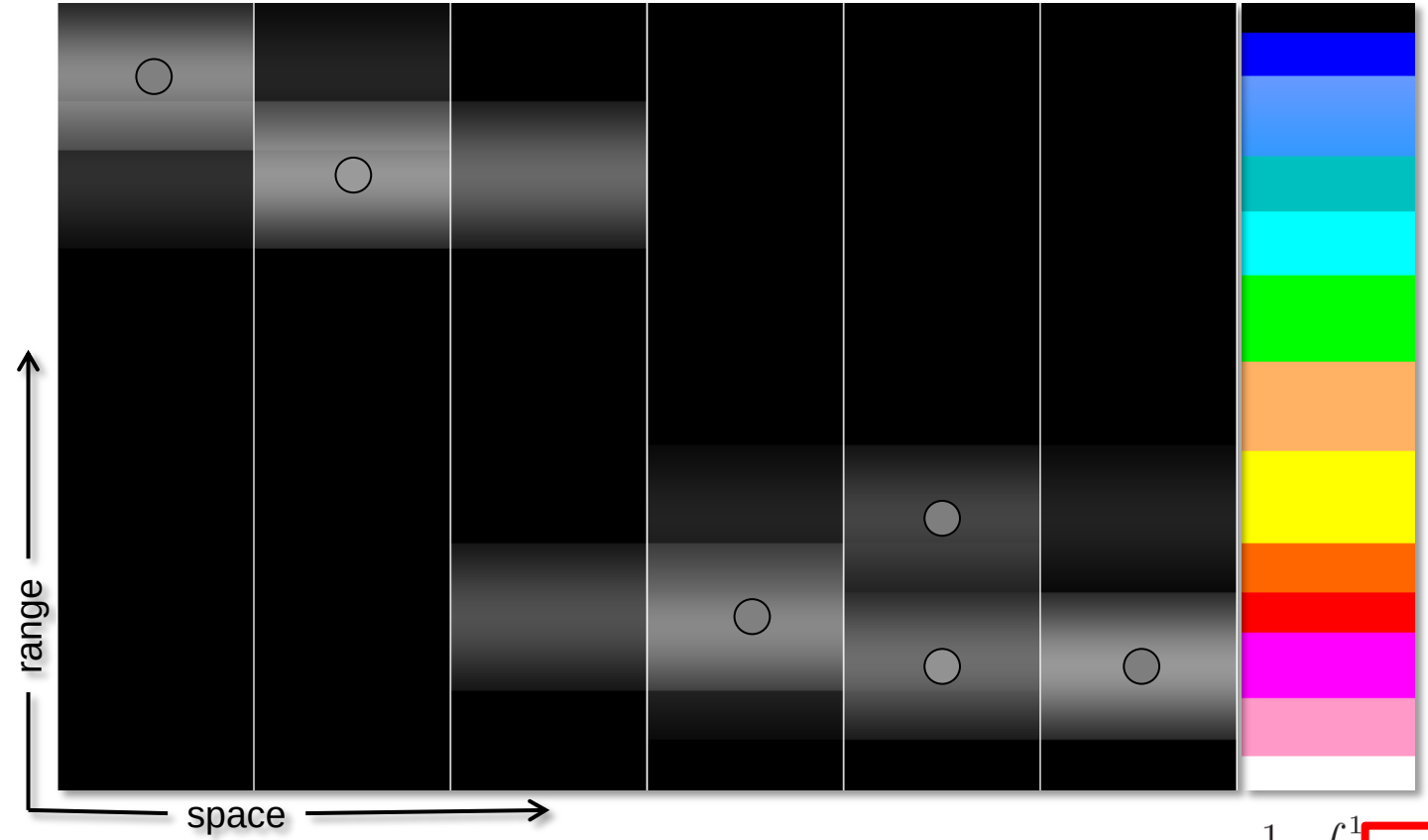


Image Reconstruction



$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r)\, pdf_{\boldsymbol{p}}(r)\, dr$$

Image Reconstruction

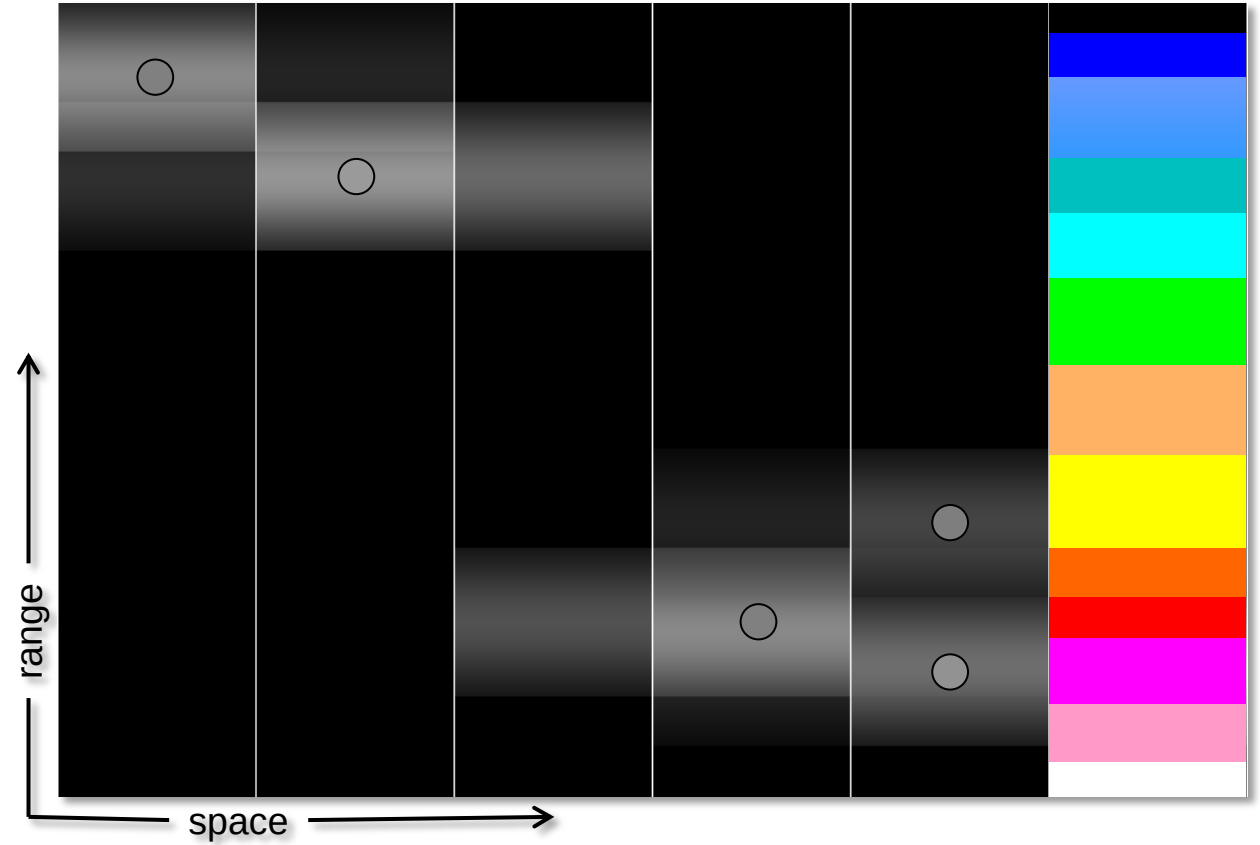


Image Reconstruction

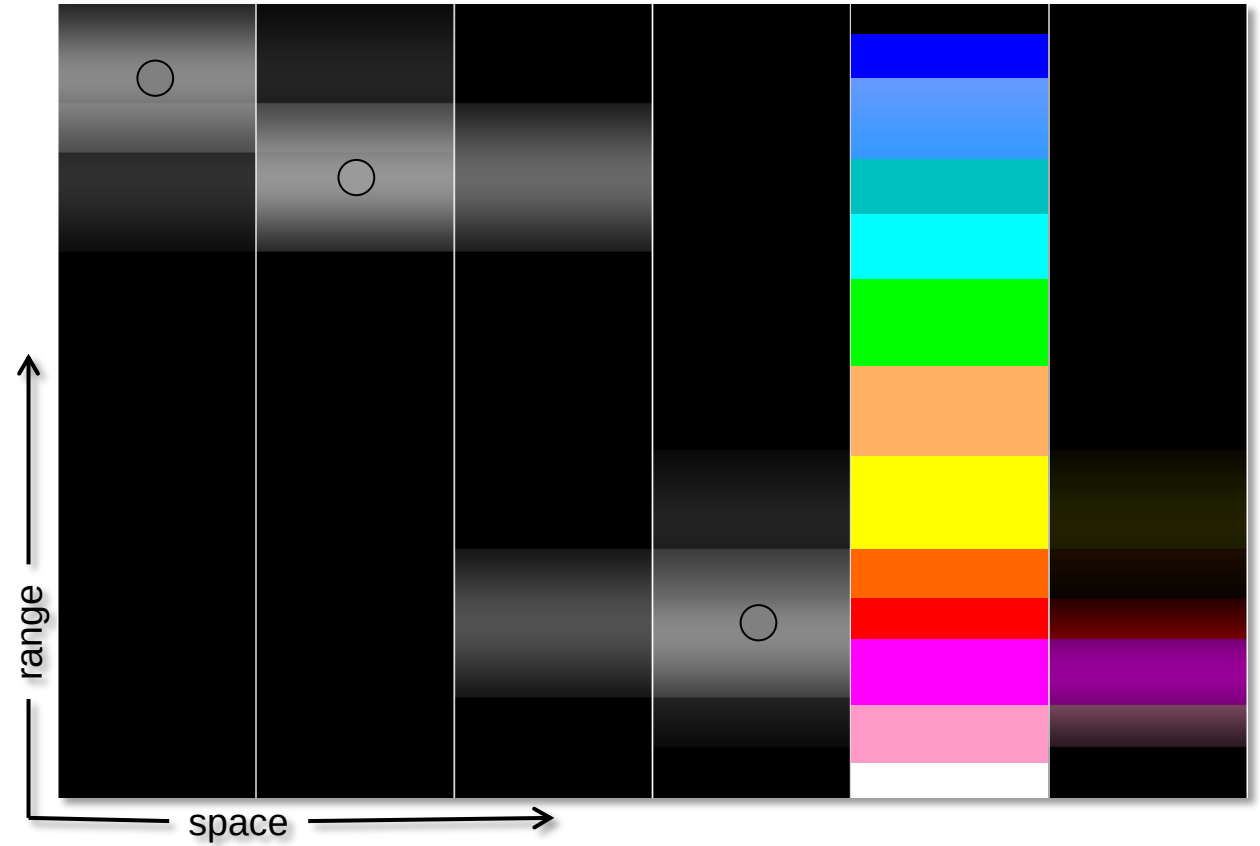


Image Reconstruction

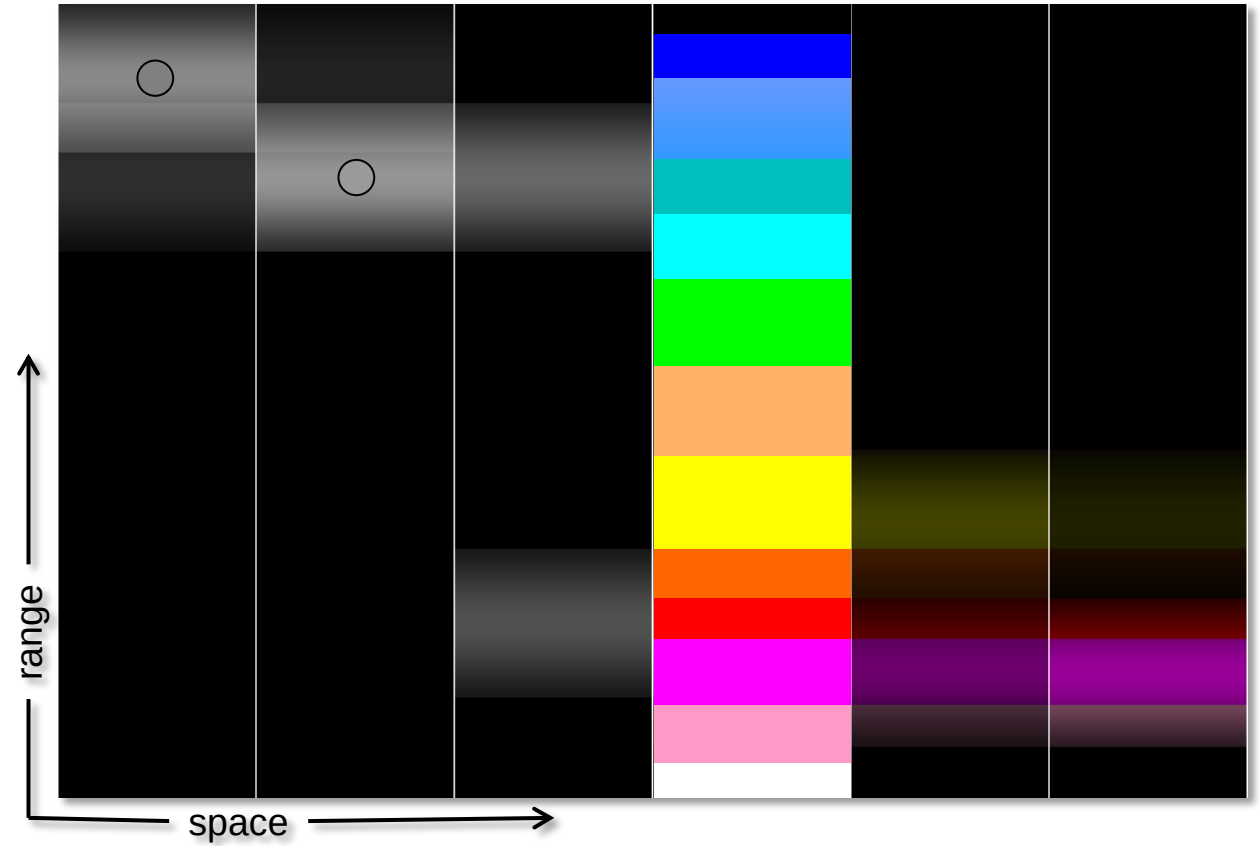


Image Reconstruction

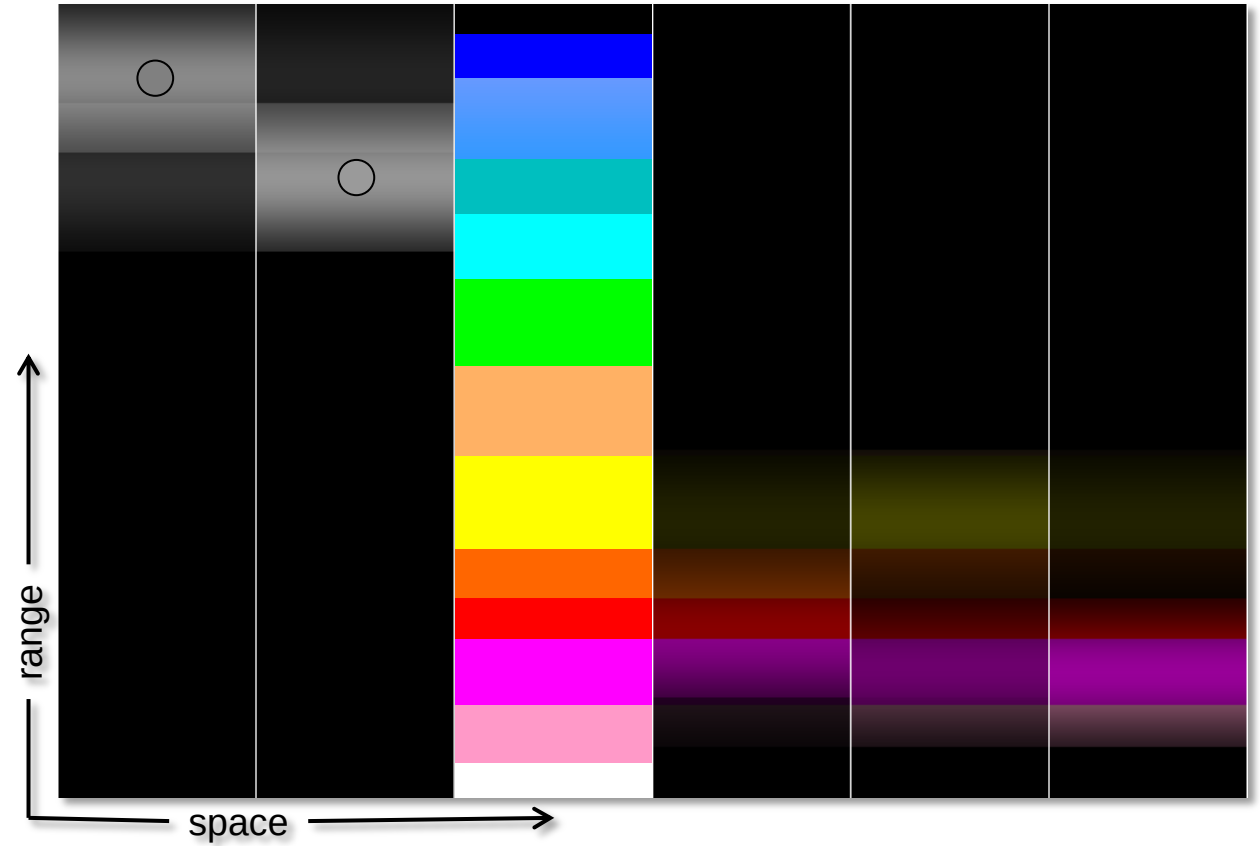


Image Reconstruction

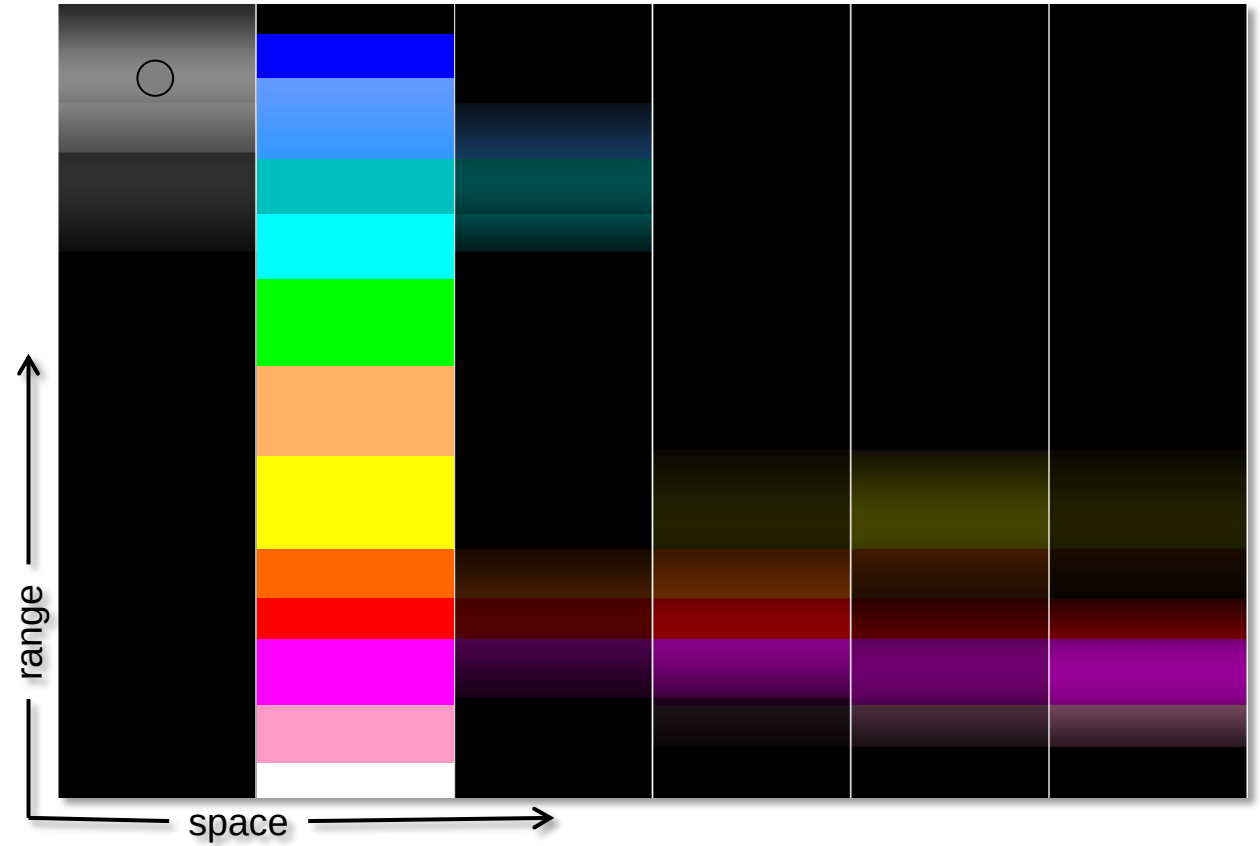


Image Reconstruction

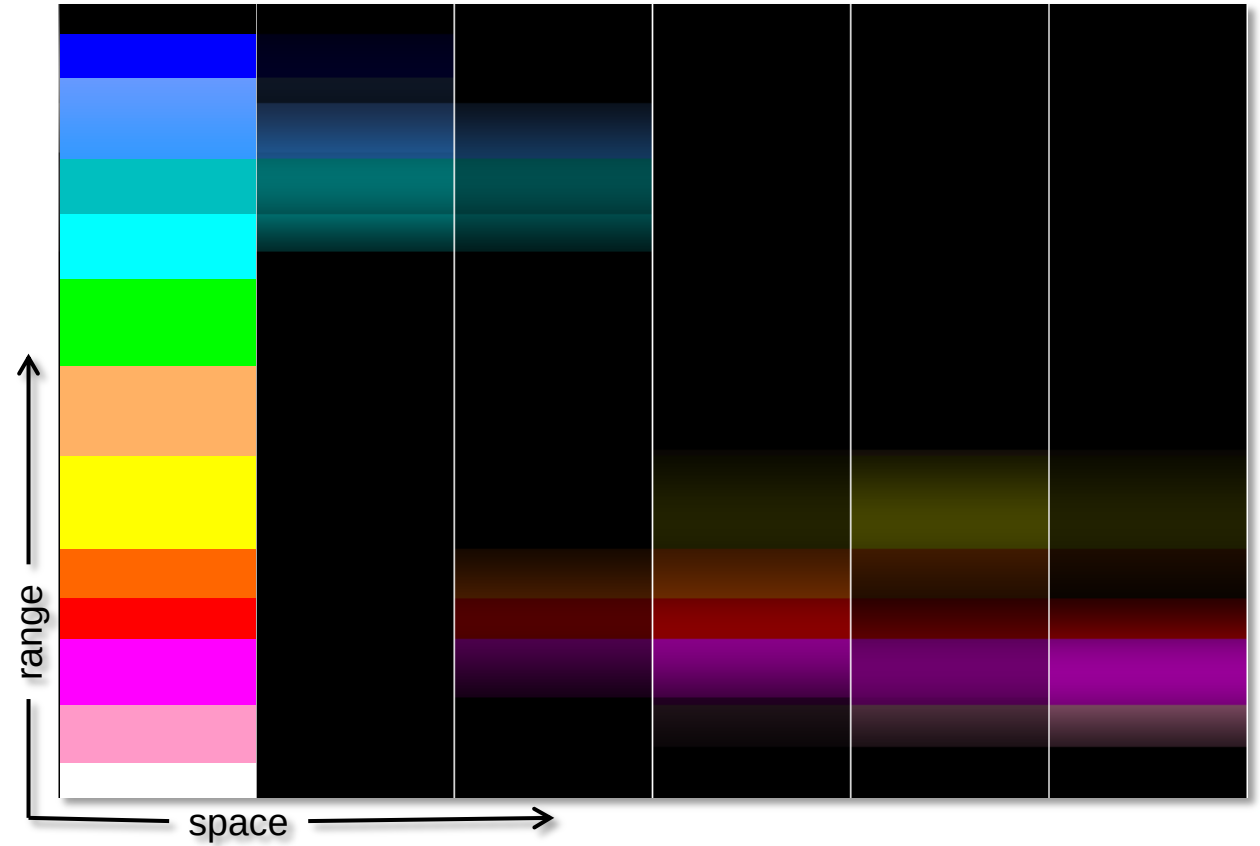


Image Reconstruction

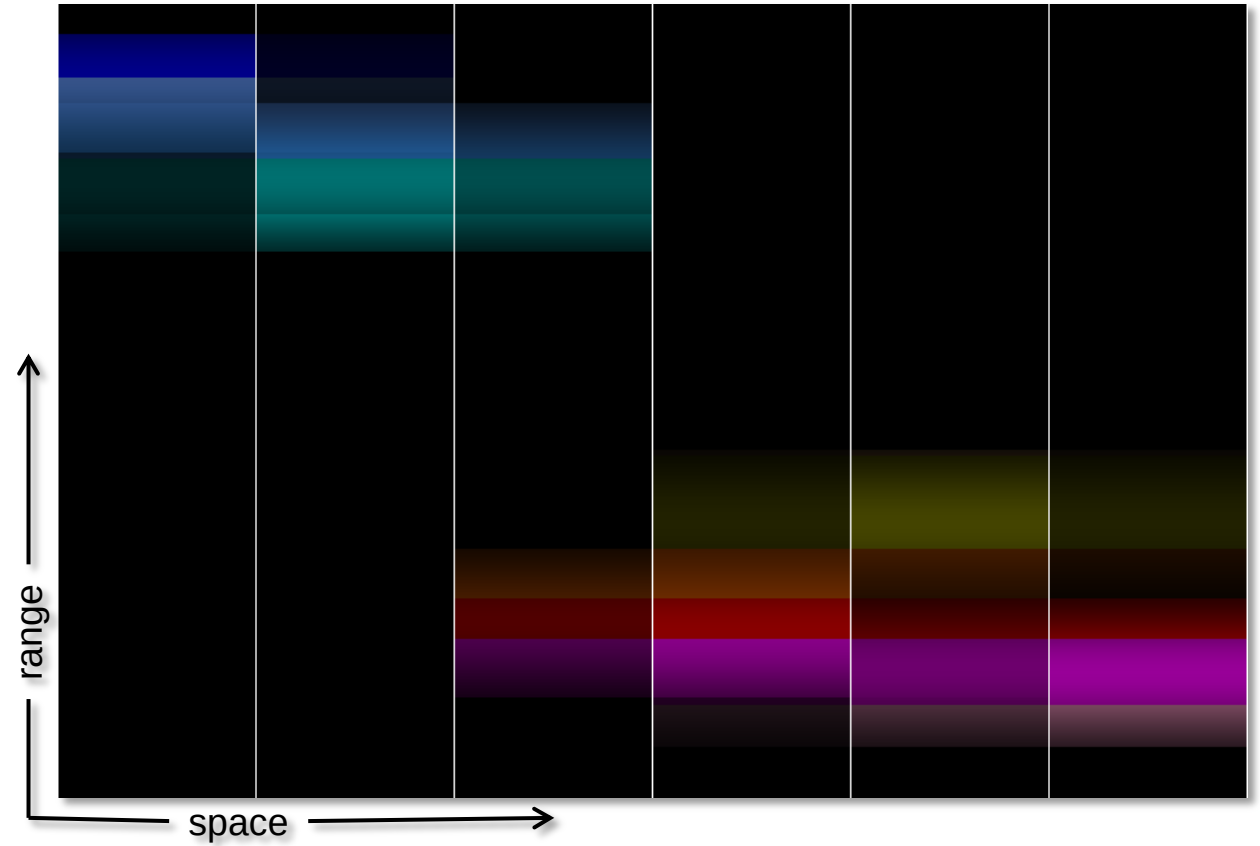


Image Reconstruction

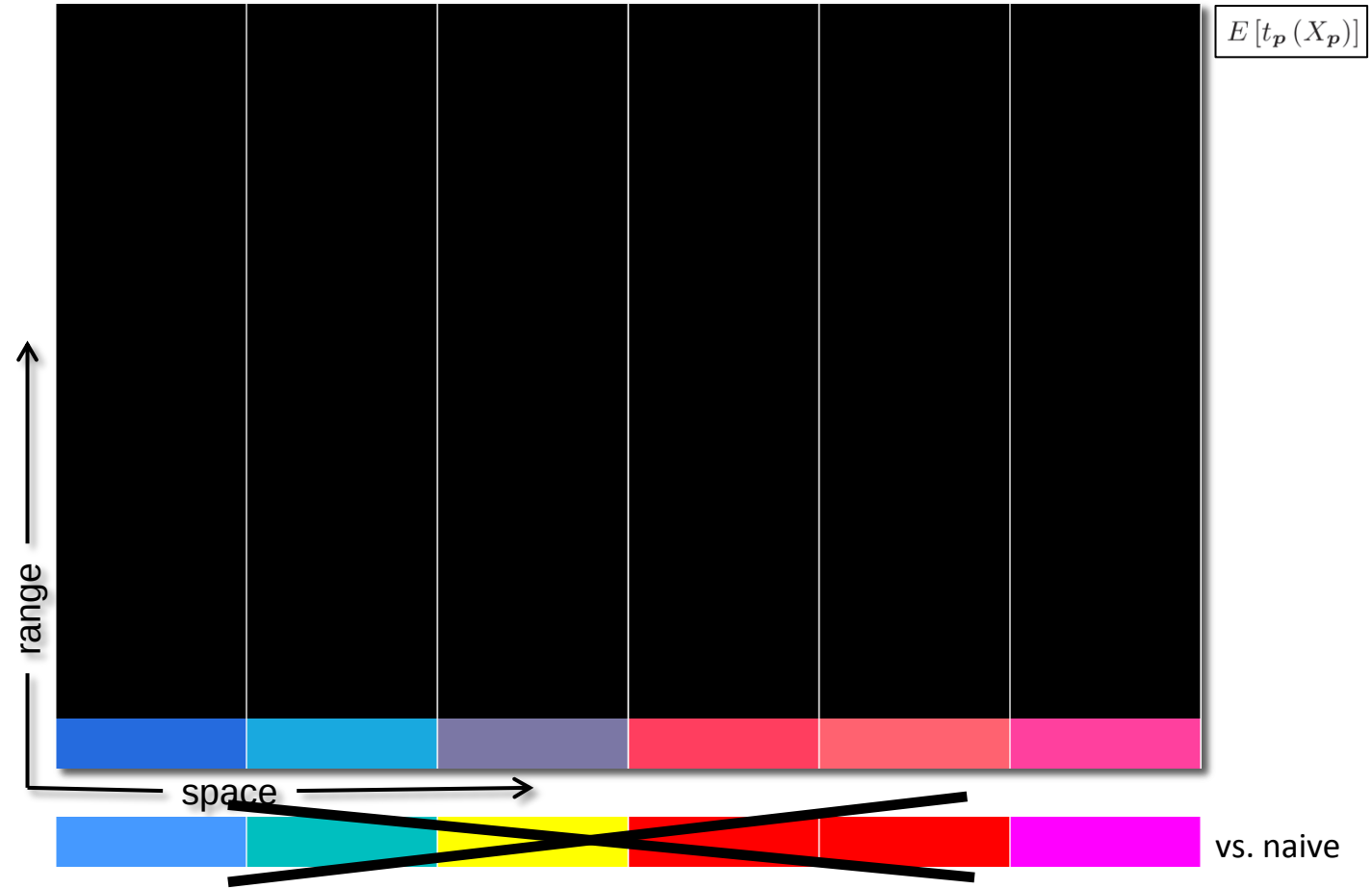


Image Reconstruction

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 t_{\boldsymbol{p}}(r)\left(V * (W \otimes K)\right) dr$$

- Key idea # 1

  Use $V * (W \otimes K)$ instead of $pdf_{\boldsymbol{p}}(r)$

- Key idea # 2

  Pre-convolve $t_{\boldsymbol{p}}(r)$ with K : $\tilde{t}_{\boldsymbol{p}} = t_{\boldsymbol{p}} * K$

Image Reconstruction

$$E\left[t_{\boldsymbol{p}}\left(X_{\boldsymbol{p}}\right)\right] = \frac{1}{w_{\boldsymbol{p}}} \int_0^1 \tilde{t}_{\boldsymbol{p}}(r)\left(V * W\right)\, dr$$

- Key idea # 1

  Use $V * (W \otimes K)$ instead of $pdf_{\boldsymbol{p}}(r)$

- Key idea # 2

  Pre-convolve $t_{\boldsymbol{p}}(r)$ with K : $\tilde{t}_{\boldsymbol{p}} = t_{\boldsymbol{p}} * K$

Image Reconstruction

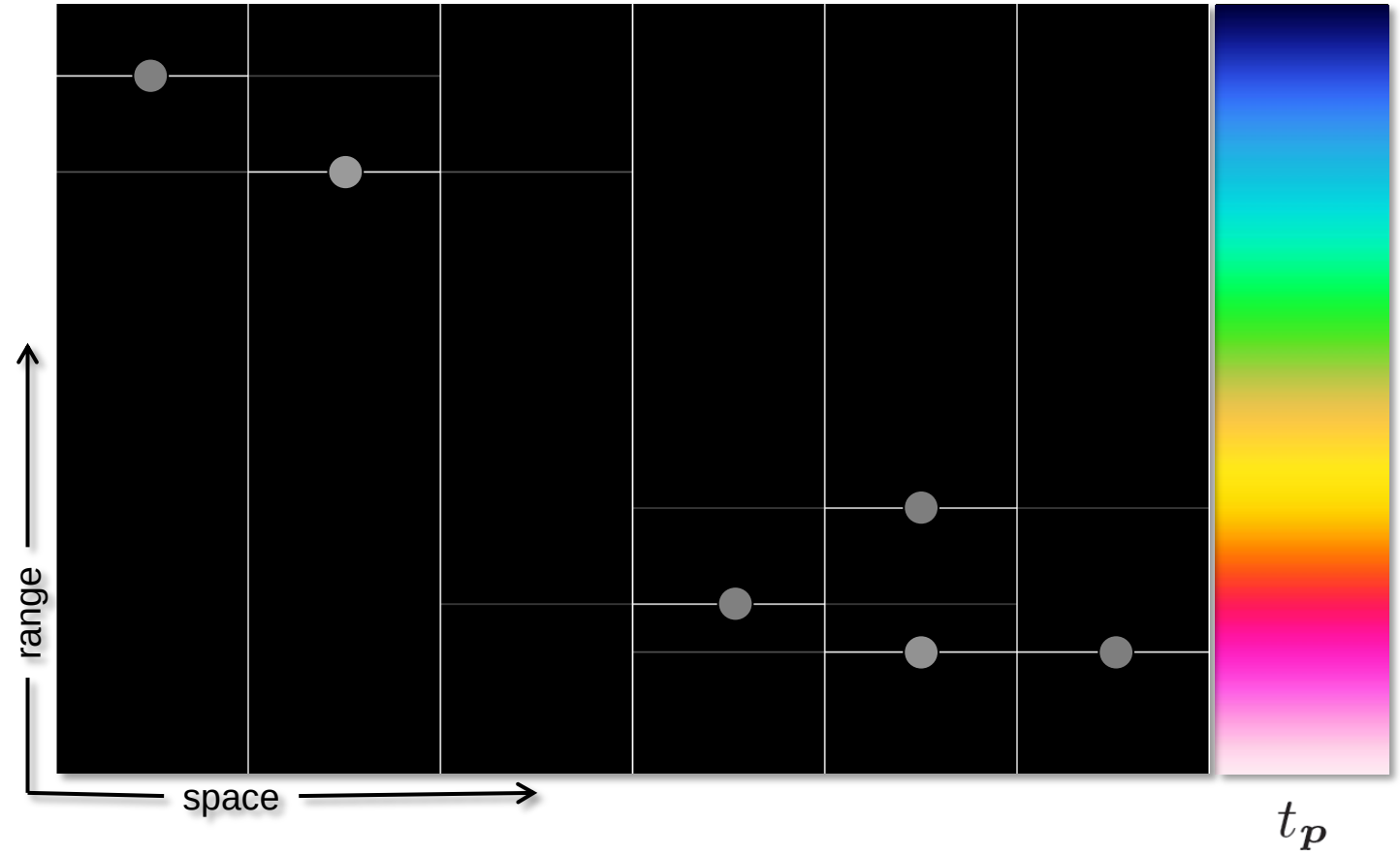


Image Reconstruction

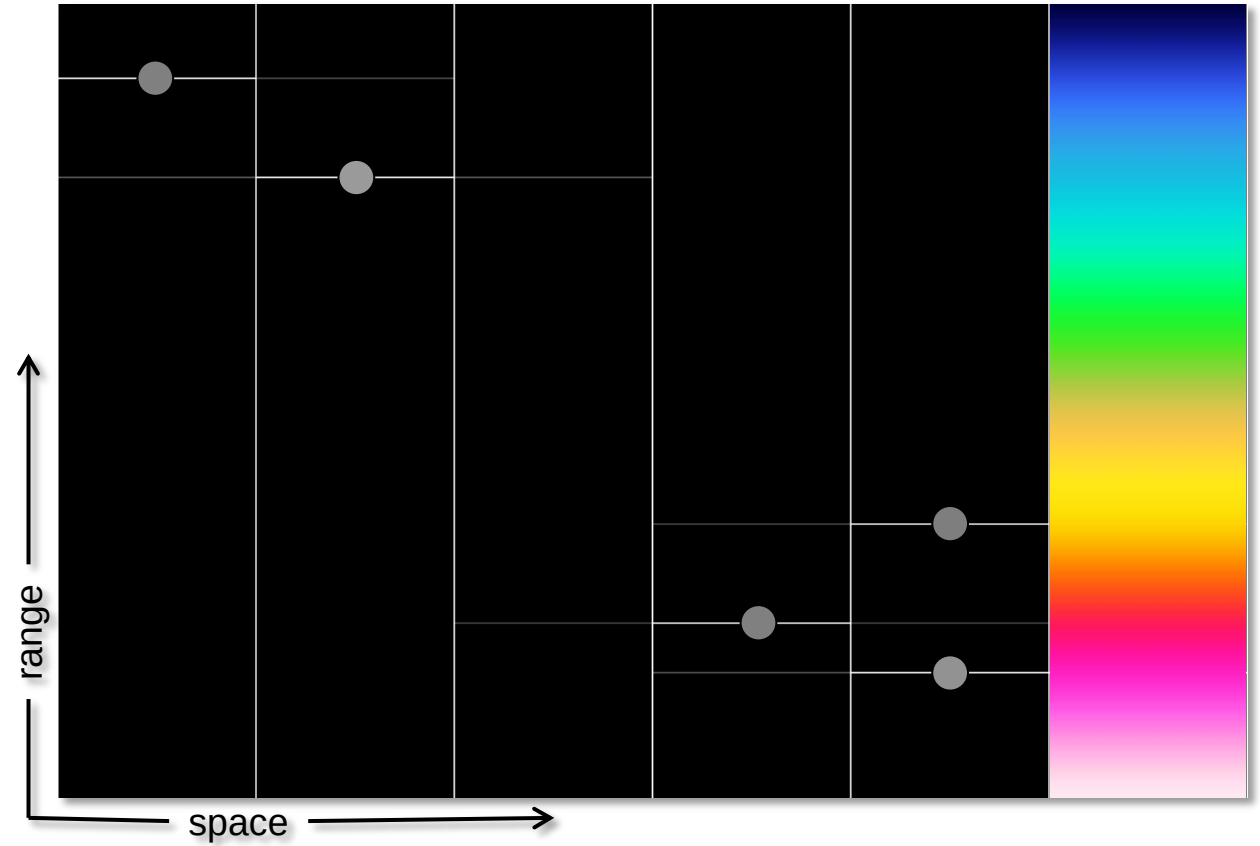


Image Reconstruction

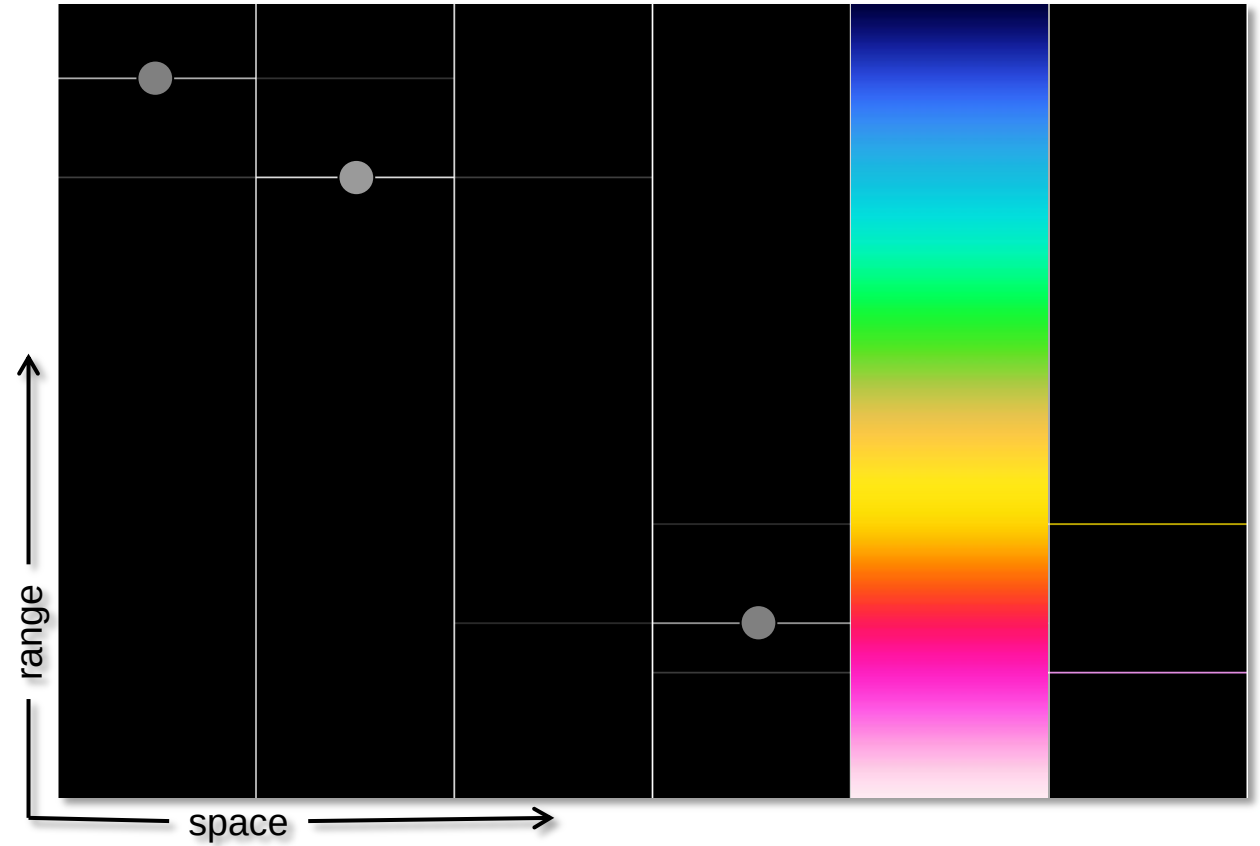


Image Reconstruction

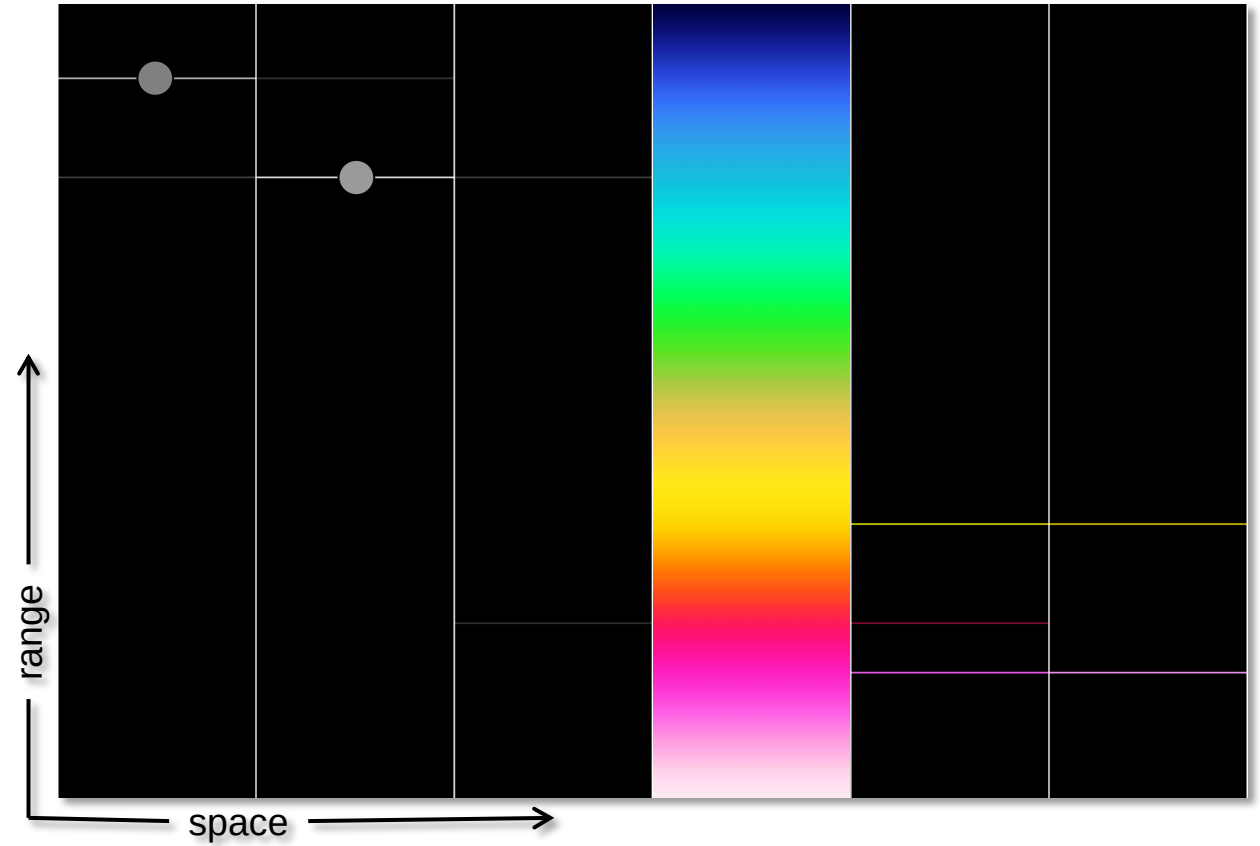


Image Reconstruction

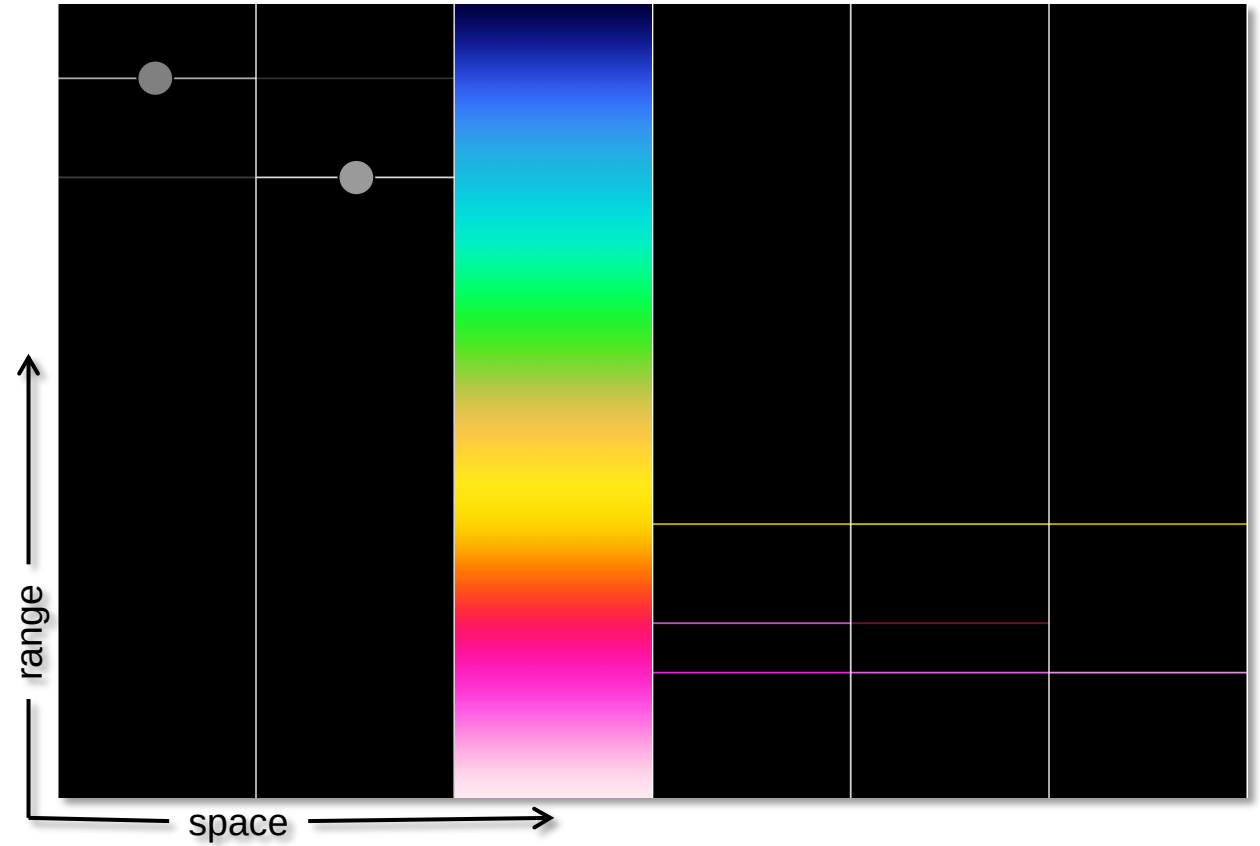


Image Reconstruction

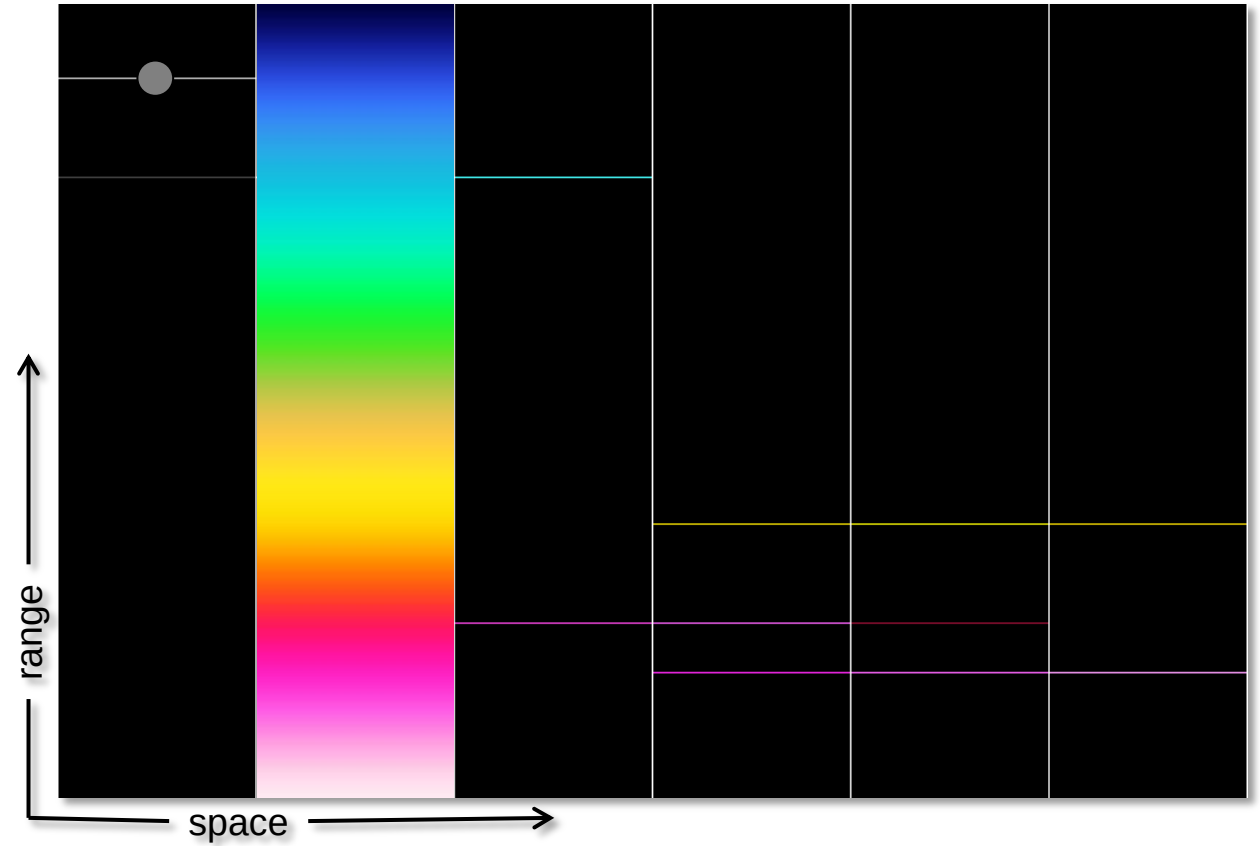


Image Reconstruction

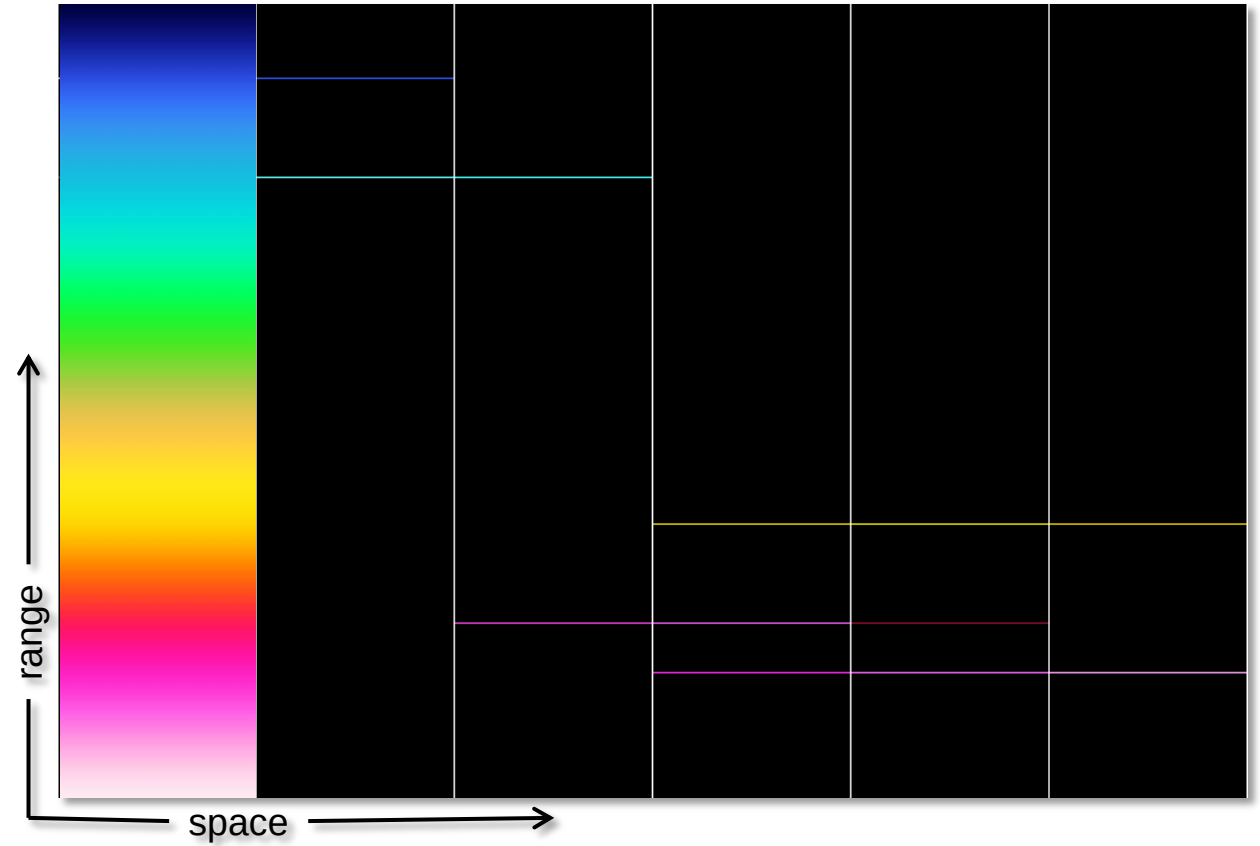


Image Reconstruction

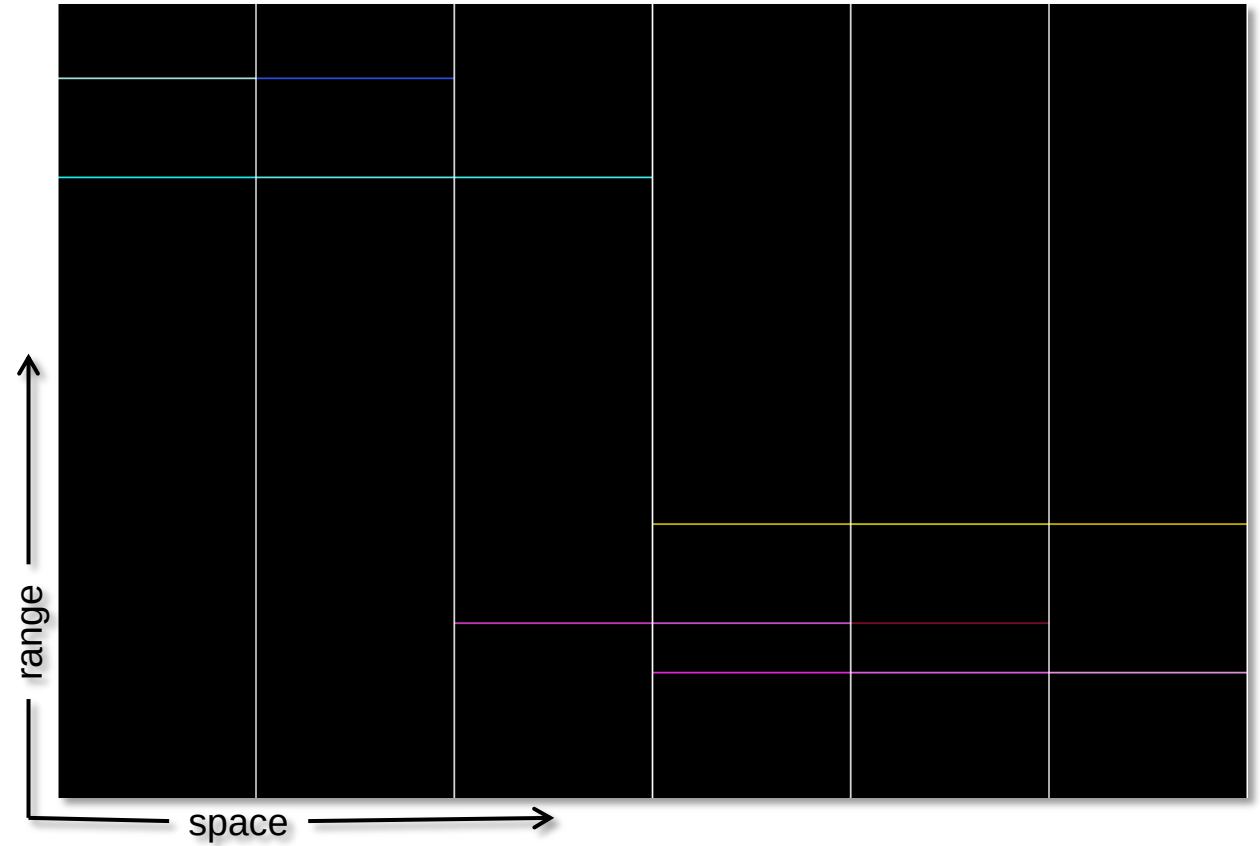


Image Reconstruction

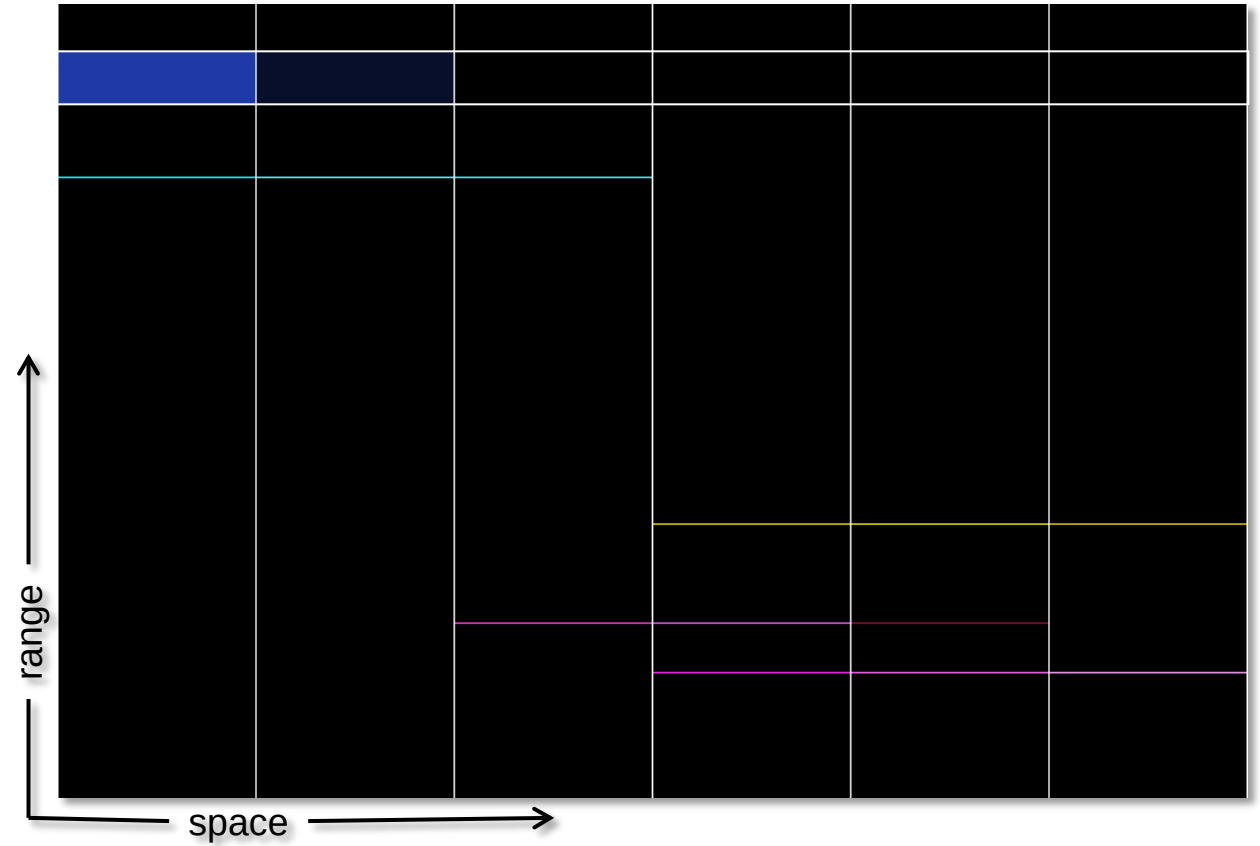


Image Reconstruction

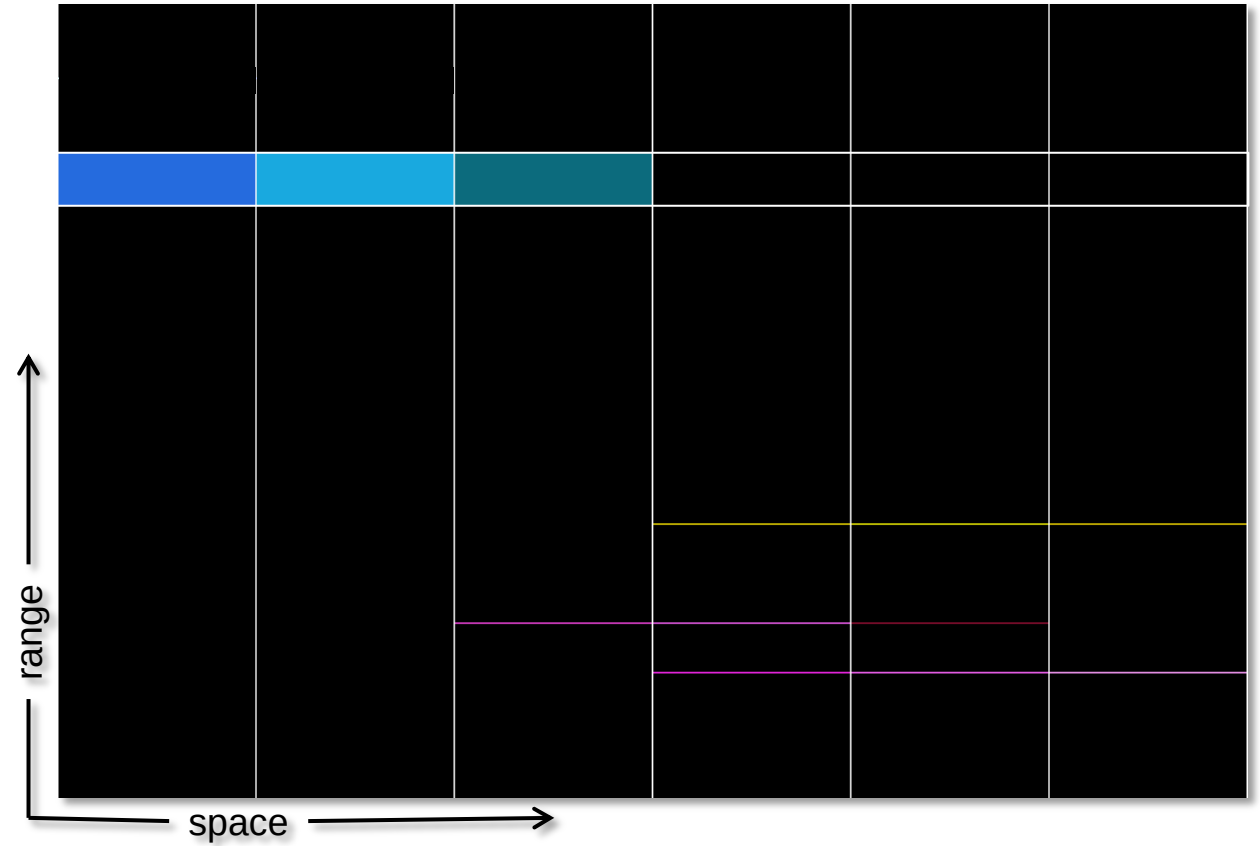


Image Reconstruction

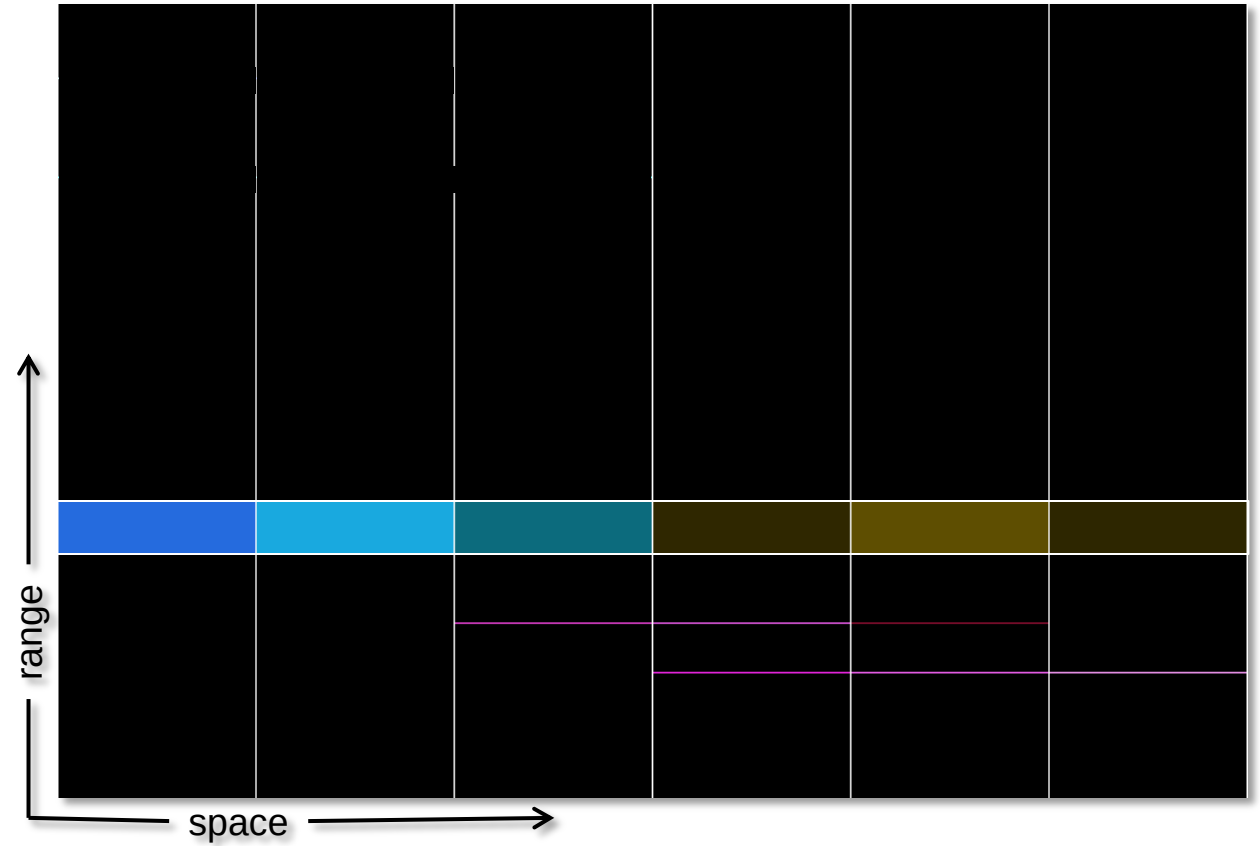


Image Reconstruction

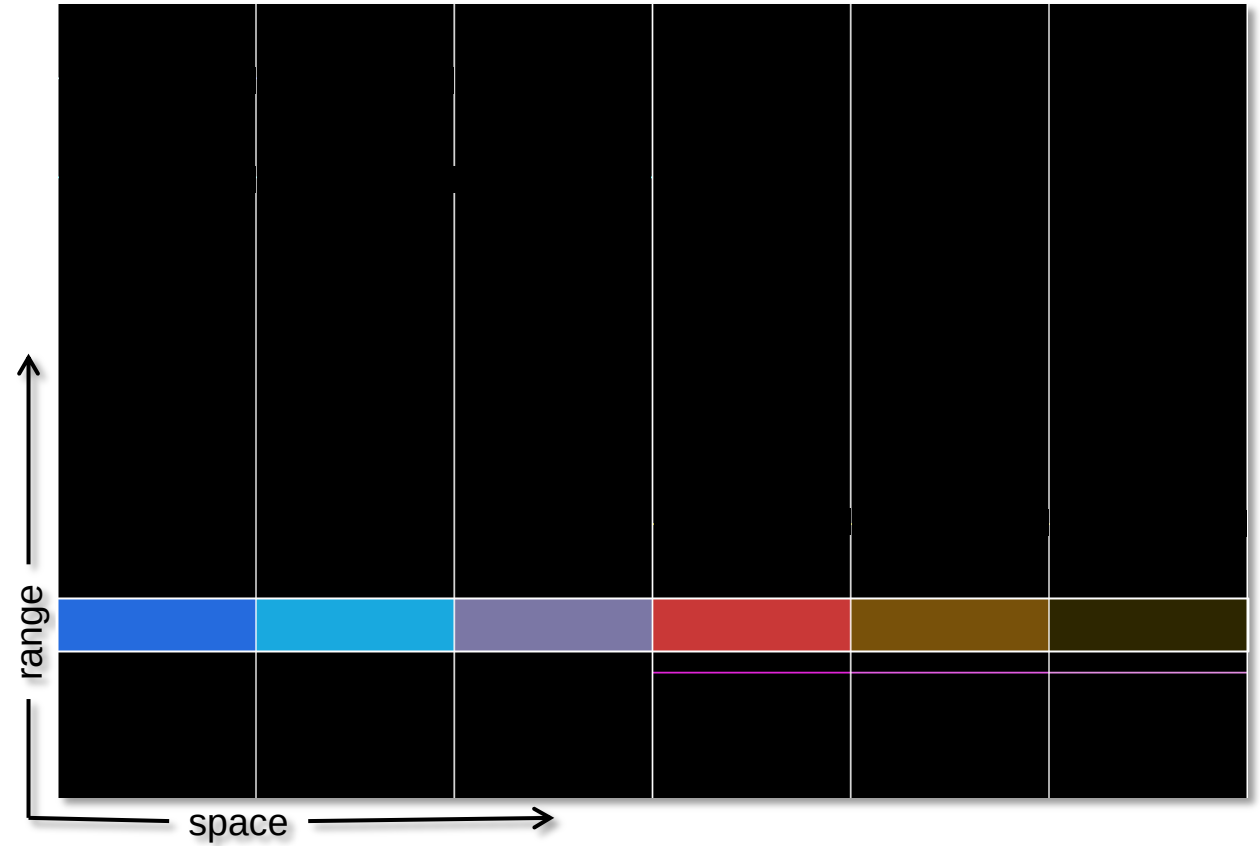


Image Reconstruction

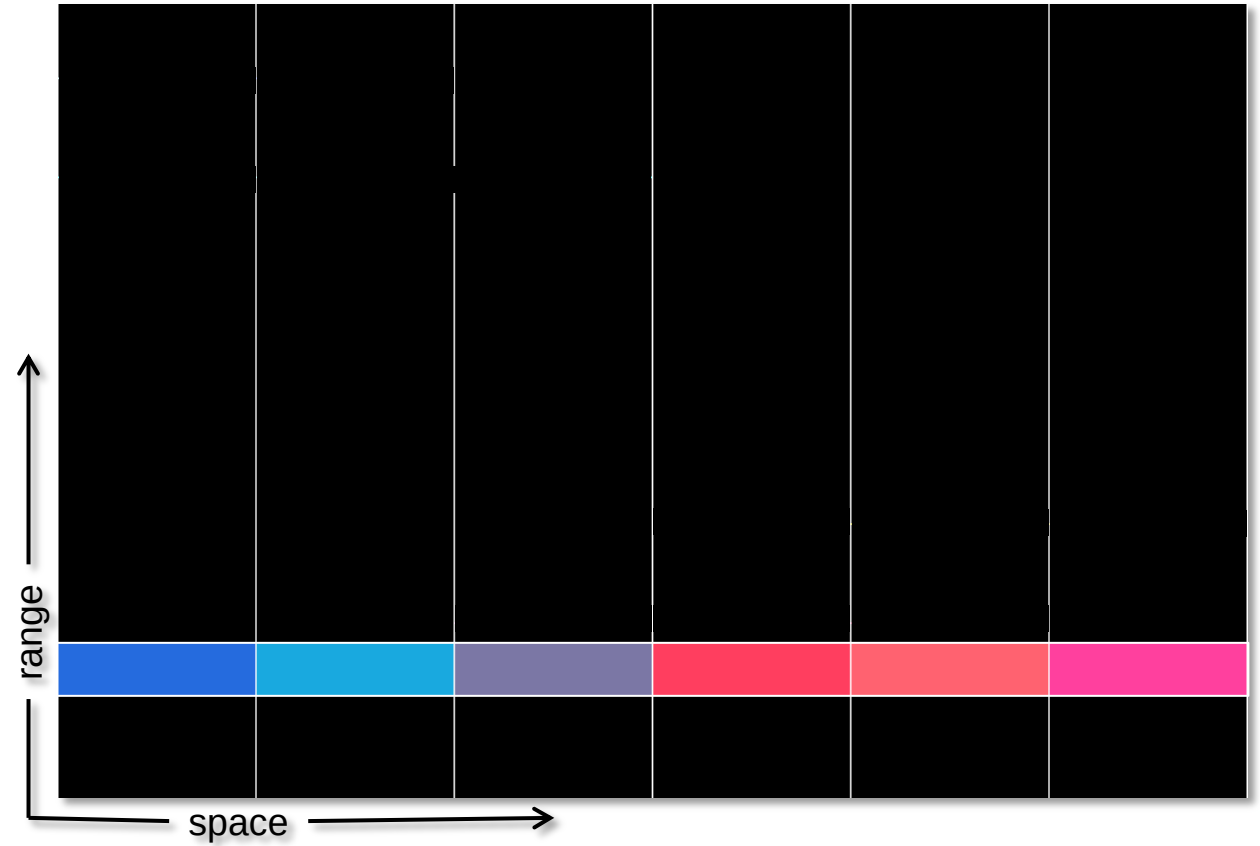
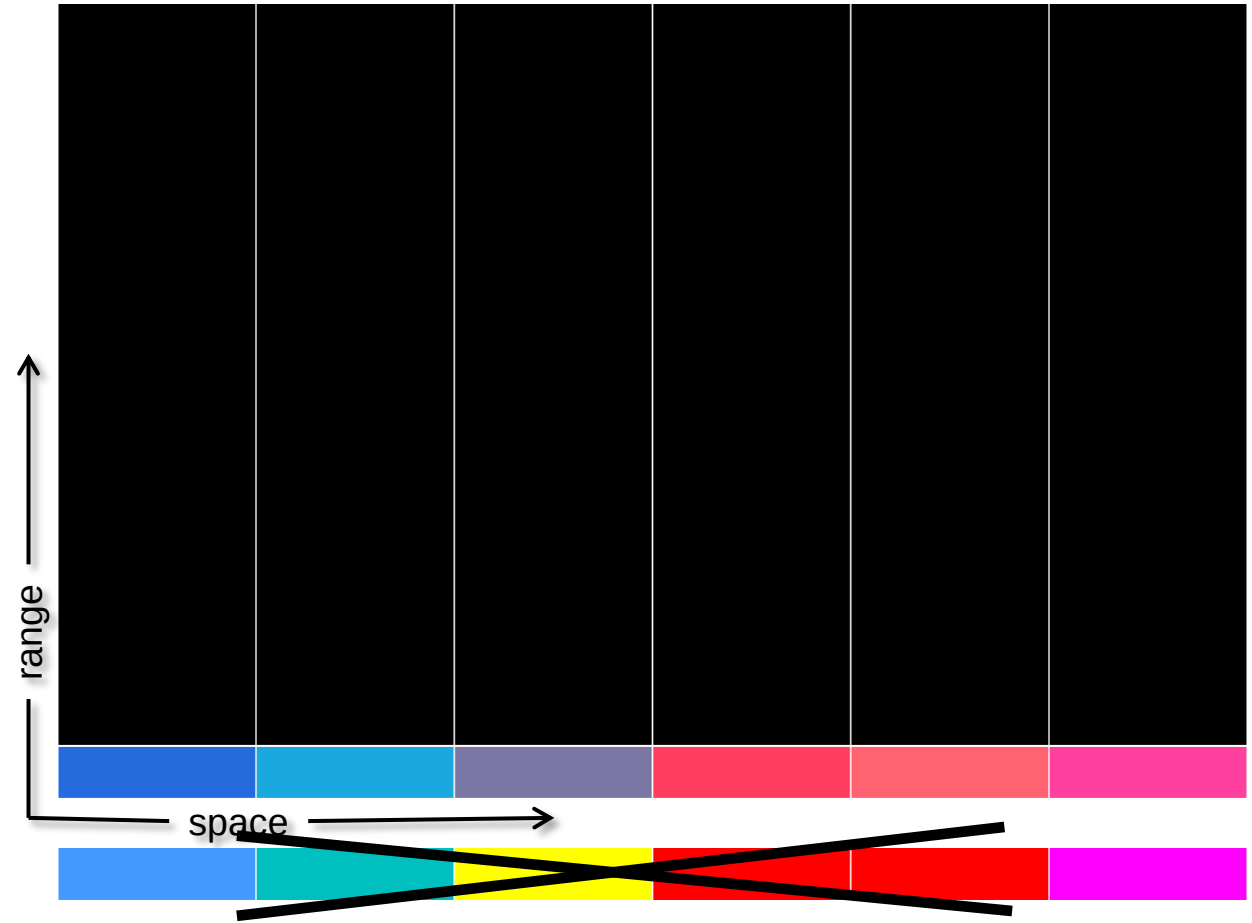


Image Reconstruction



vs. naive

Image Reconstruction - CUDA Implementation

- 1 thread per pixel
- Cuda kernel:
  - collect coefficients in pixel neighborhood, and calculate their spatial weight
  - for each coefficient in neighborhood
    - apply pre-convolved color map
    - sum up
  - store in surface

Image Reconstruction - Optimization

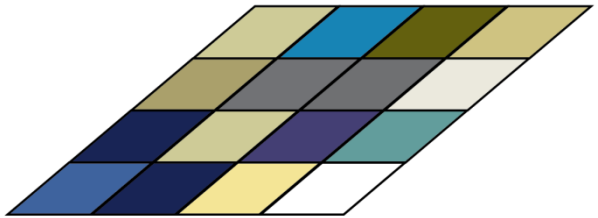
$$E\left[t(X_{\boldsymbol{p}})\right] = \frac{1}{w_{\boldsymbol{p}}} \sum W \sum \tilde{t}_{\boldsymbol{p}}(r)\, V$$

- Optimization for color mapping with global function $t_{\boldsymbol{p}}(r)$:

  no need to apply spatial convolution to each coefficient individually

Color Mapping

1. Pre-convolve color map (with range kernel)
2. For each pixel go over its coefficients
    a. Apply color map to coefficient and sum up (not spatially convolved yet!)
3. One spatial convolution in the end



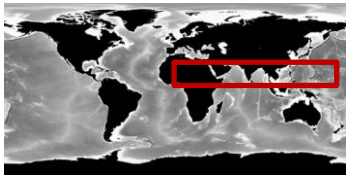
Color Mapping - CUDA Implementation

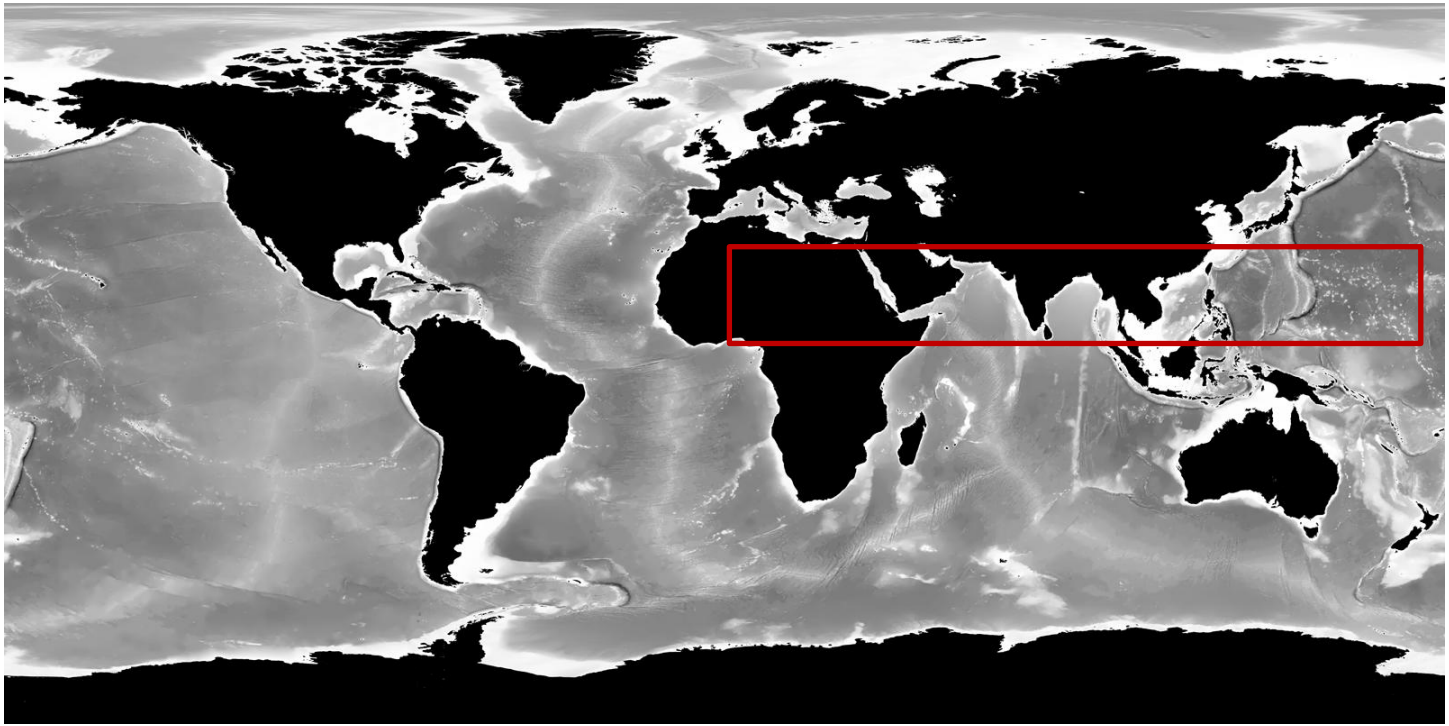
- 1 thread per pixel
- 1st pass:
  - loop over coefficients of current pixel
    - apply and sum up pre-convolved color map
  - write resulting color image into a surface
- 2nd pass:
  - spatial convolution on result of 1st pass

Results

Color Mapping Gigapixel Images

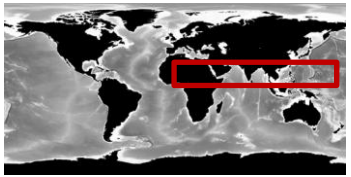
- NASA Blue Marble bathymetry
  - 21,601 x 10,801 pixels (233 Mpixels)





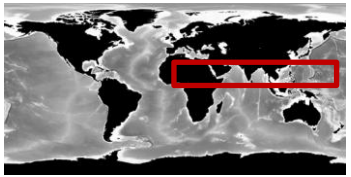
Color Mapping Gigapixel Images

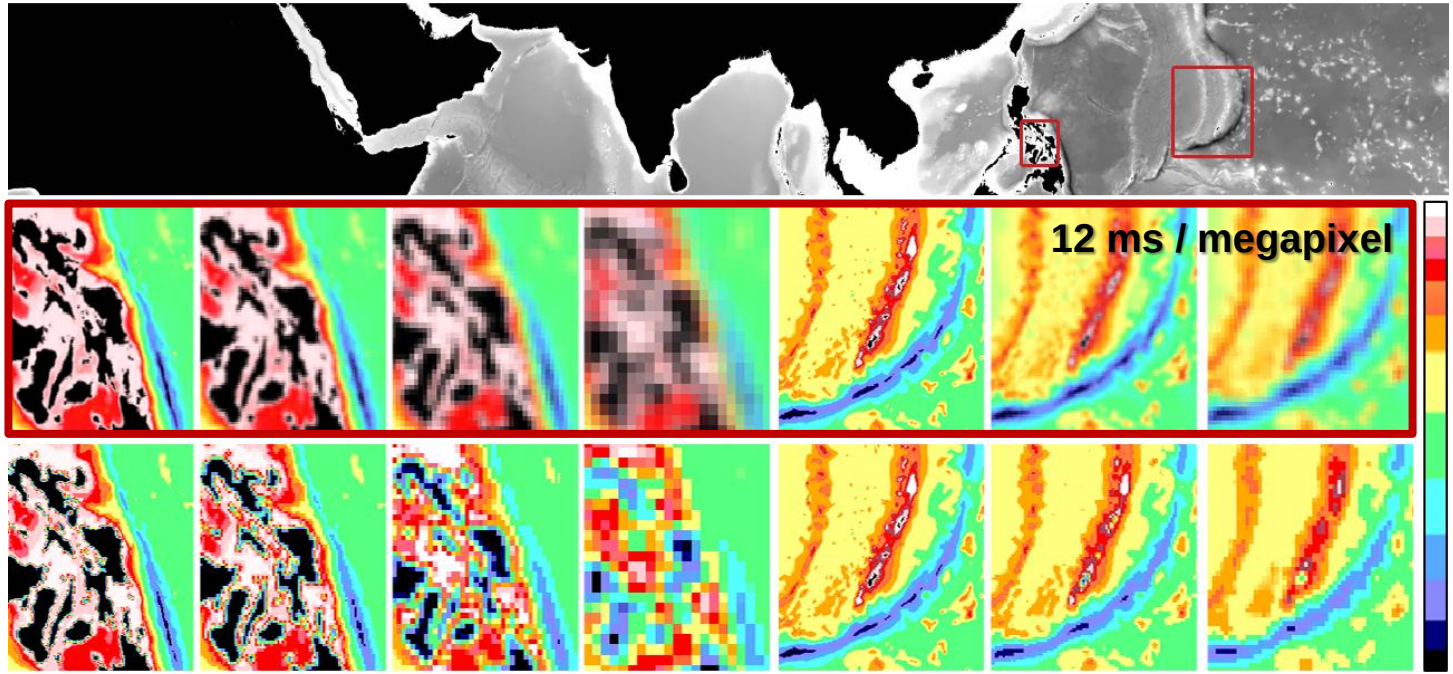
- NASA Blue Marble bathymetry
  - 21,601 x 10,801 pixels (233 Mpixels)



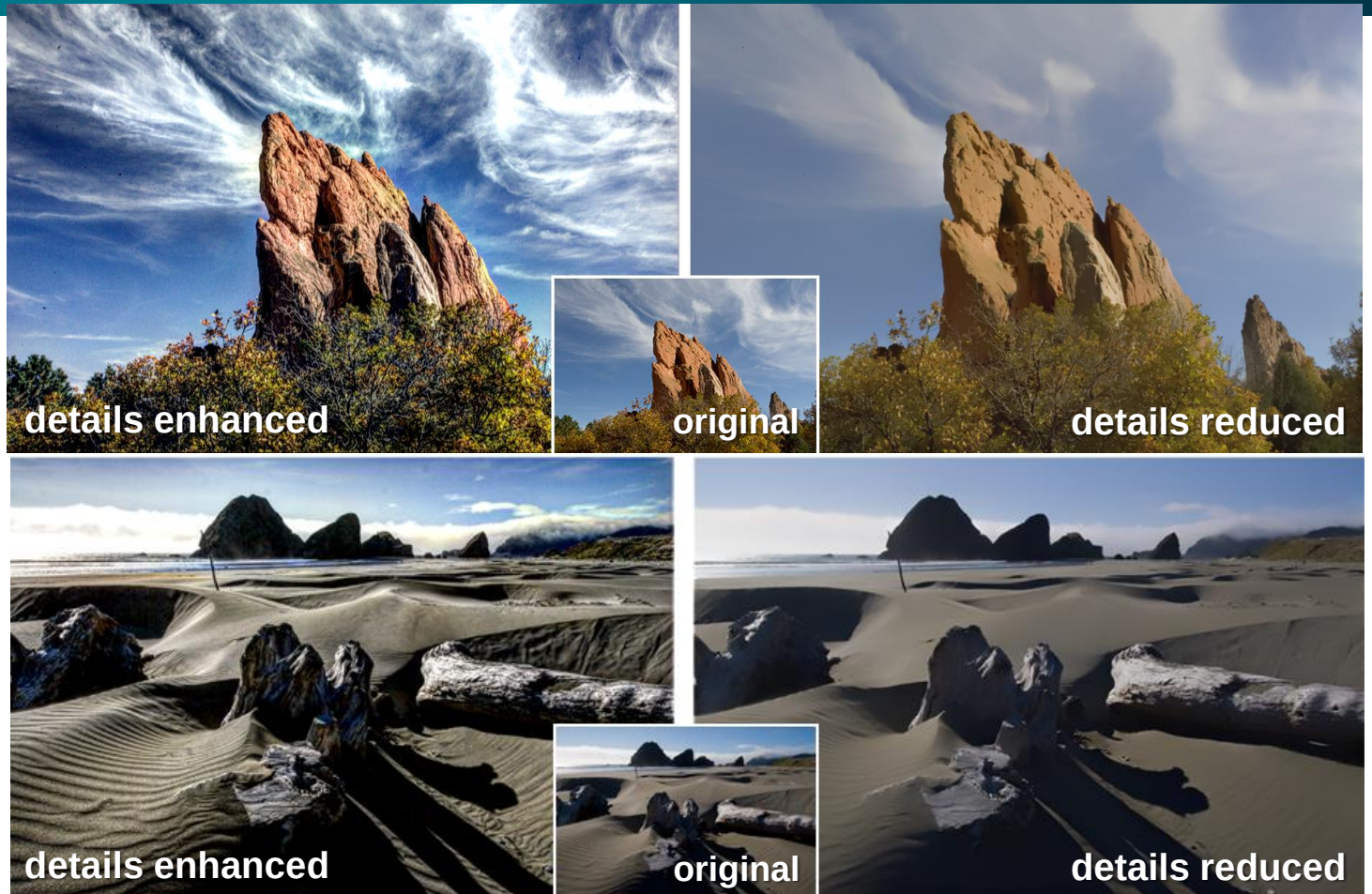
Color Mapping Gigapixel Images

- NASA Blue Marble bathymetry
  - 21,601 x 10,801 pixels (233 Mpixels)





Fast Local Laplacian Filtering



Gigapixel Local Laplacian Filtering

- Night Scene Panorama: 47,908 x 7,531 pixels (361 Mpixels)

original

details reduced

details enhanced
300 ms / megapixel

Summary

- Flexible new image pyramid representation
  - Consistent, sparse representation of pixel footprint pdfs
- Unified evaluation of many important non-linear image operations (computable from 1D pdfs)
  - Local Laplacian filtering for gigapixel images
- Efficient CUDA implementation
- Pre-computation costly, but only performed once
- Run time storage and computation similar to standard pyramids

Thank You for Your Attention!

http://gmsv.kaust.edu.sa

[Hadwiger et al. 2012]
“Sparse PDF Maps for Non-Linear Multi-Resolution Image Operations”, Siggraph Asia 2012