⊕ ⊖ Computational ⊗ ⊘ Photography

Bilateral Filtering+

Jongmin Baek

CS 478 Lecture
Feb 1, 2012

Announcements

- Assignment 1 grading
  - Are you signed up?
- Assignment 2
  - Due 2/8
- Term project proposal
  - Due 2/13
  - *Must* have had project conference. Sign up.

Overview

- Bilateral filtering
  - Theory and Applications
- Generalizations
- Other edge-aware filters

Bilateral Filtering

- A very popular "edge-aware" filter
- Blurs a signal without destroying structure

Blurring 101

- For each pixel v, mix it with its neighbors.
- Typically a convolution with a kernel f:

$$v'(x_1, x_2) = \underset{\text{SUM}}{\sum_{y1,y2}} \overset{\text{NEIGHBOR}}{v(y_1, y_2)} \underset{\text{WEIGHT}}{f(y_1-x_1, y_2-x_2)}.$$

- Kernel is typically normalized (sum to one)

Box Filter

$$v'(x_1, x_2) = \sum_{y_1,y_2} v(y_1, y_2)\, f(y_1-x_1, y_2-x_2).$$

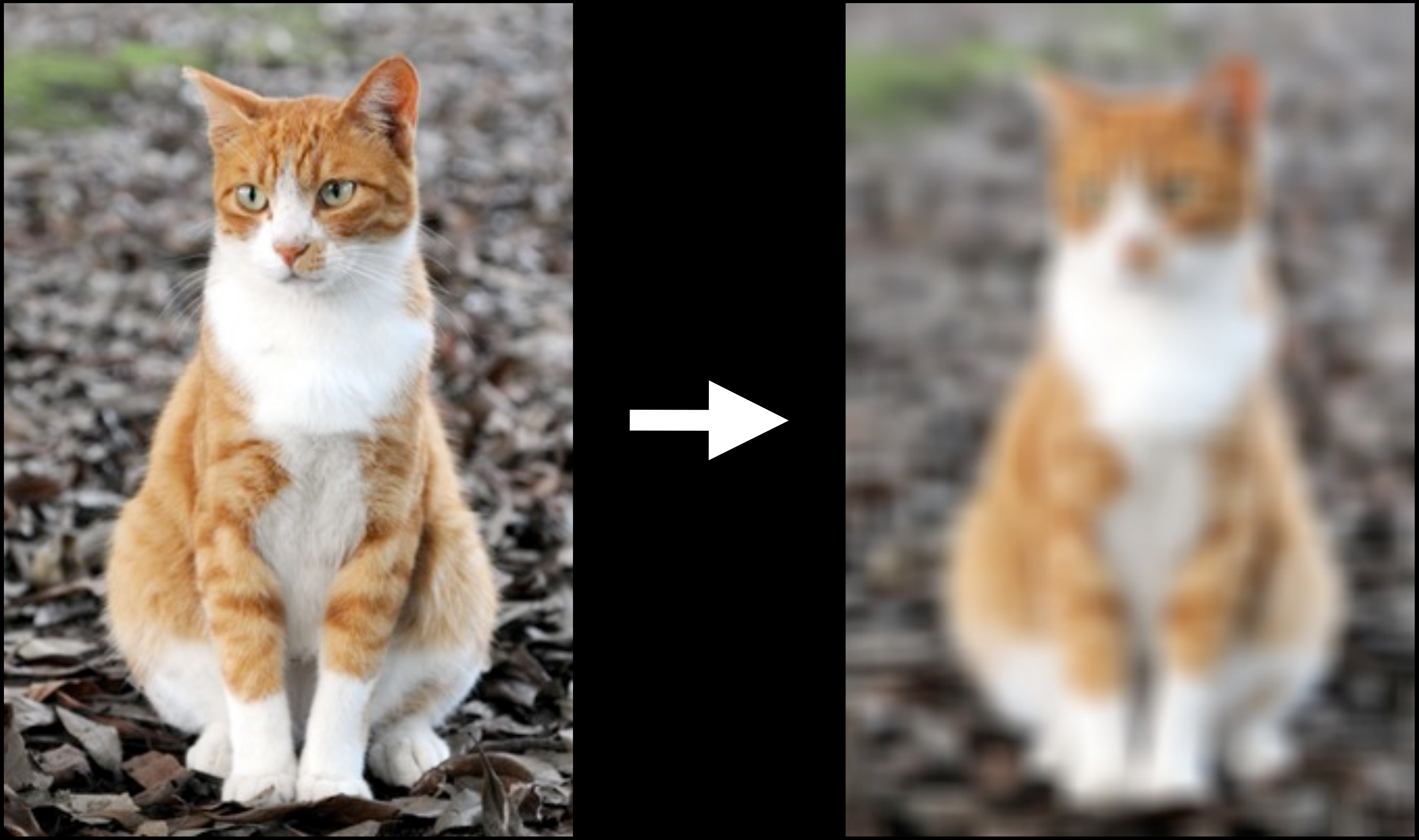
- Box filter of size w x h

$$f(a, b) = \begin{cases} 1/(wh), & \text{if } |a| \leq w/2 \text{ and } |b| \leq h/2, \\ 0, & \text{otherwise.} \end{cases}$$

Box Filter

$$v'(x_1, x_2) = \sum_{y_1,y_2} v(y_1, y_2)\, f(y_1-x_1, y_2-x_2).$$

- Box filter of size w x h



Gaussian Filter

$v'(x_1, x_2) = \sum_{y1,y2} v(y_1, y_2)\ f(y_1-x_1, y_2-x_2).$

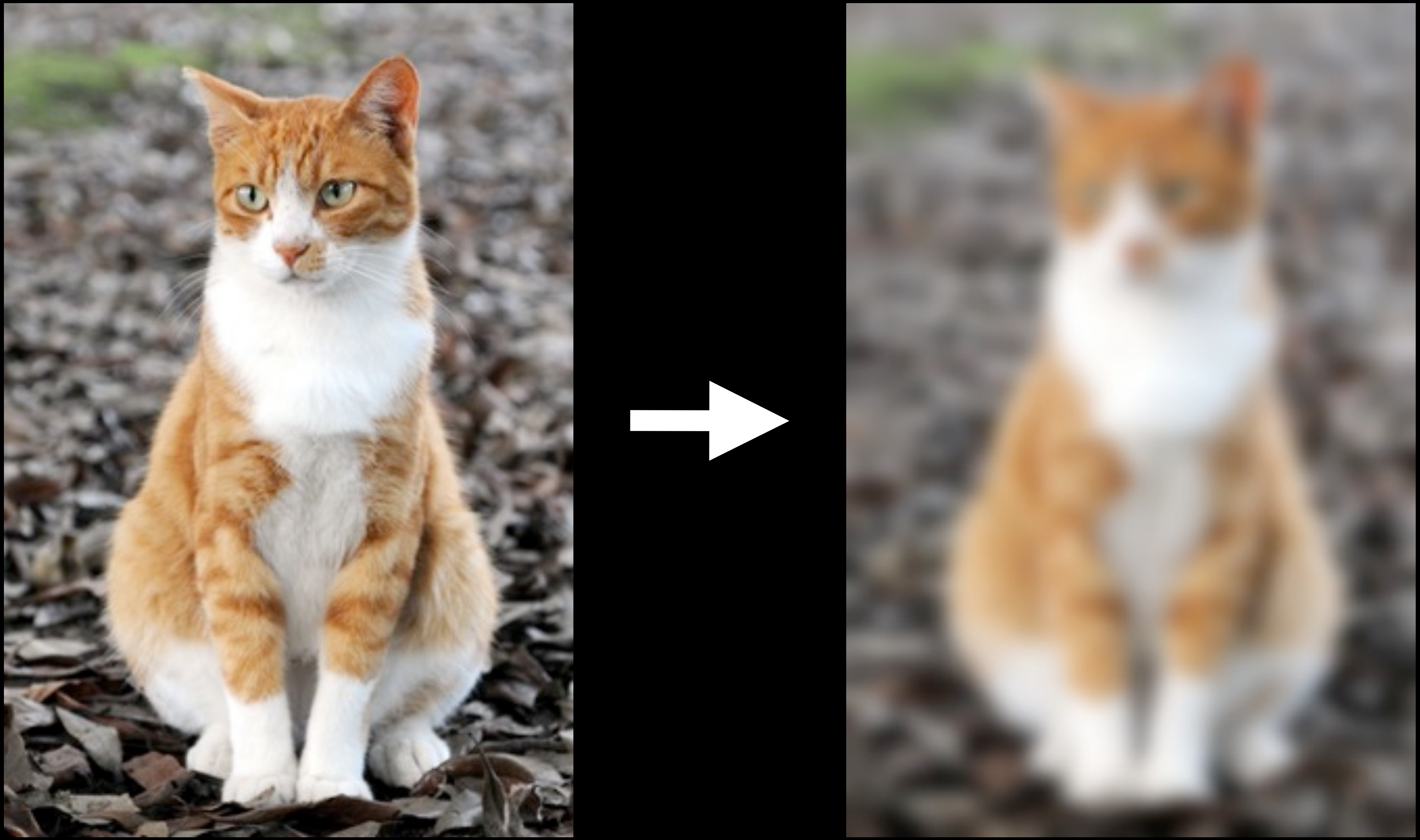
- Gaussian of stdev 5

$$f(a, b) = \exp(- [a^2+b^2] / 10) / (50\pi)^{0.5}$$

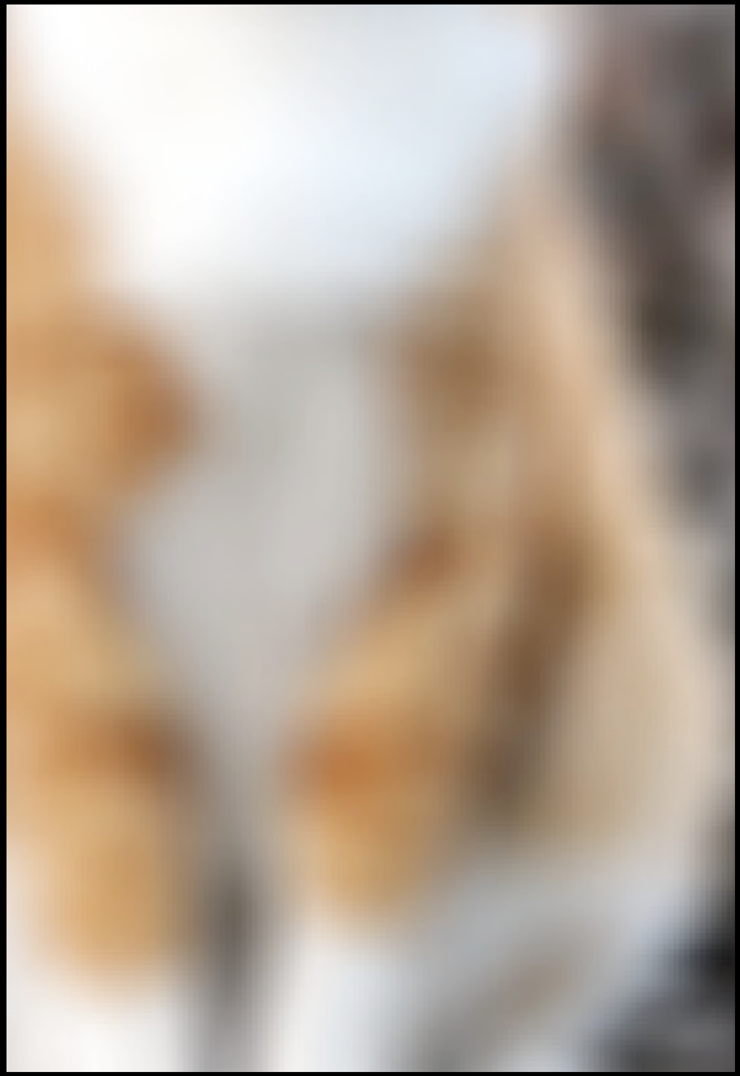
Gaussian Filter

$v'(x_1, x_2) = \sum_{y_1,y_2} v(y_1, y_2)\, f(y_1-x_1, y_2-x_2).$

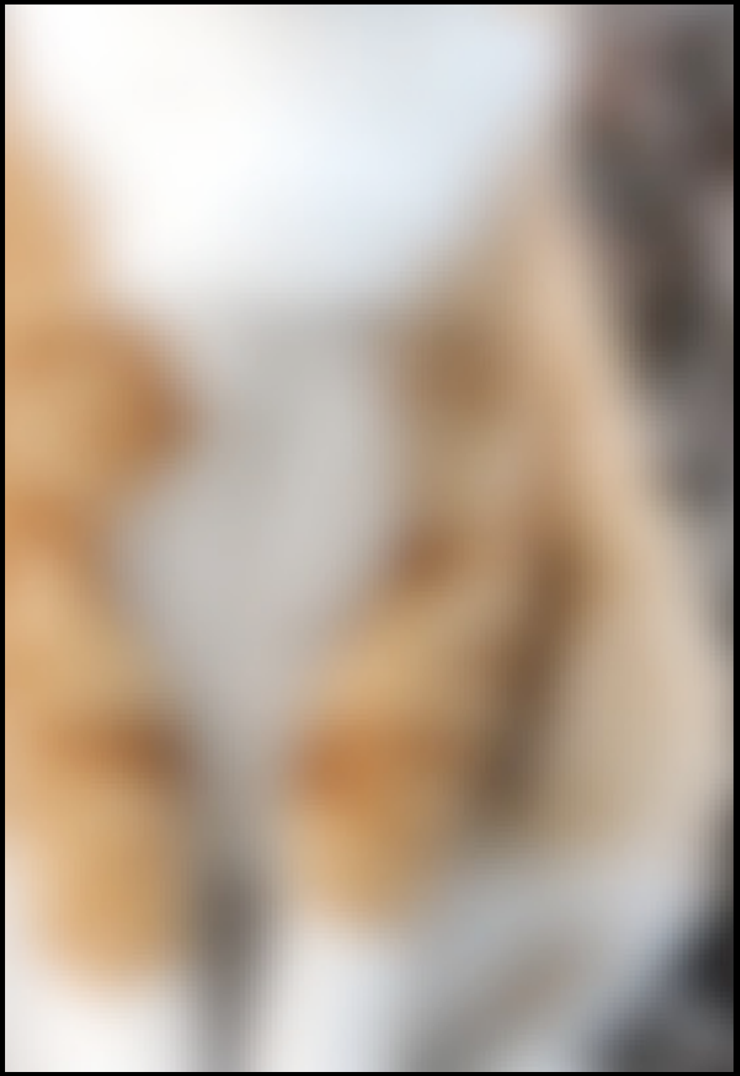
- Gaussian of stdev 5



Box vs. Gaussian

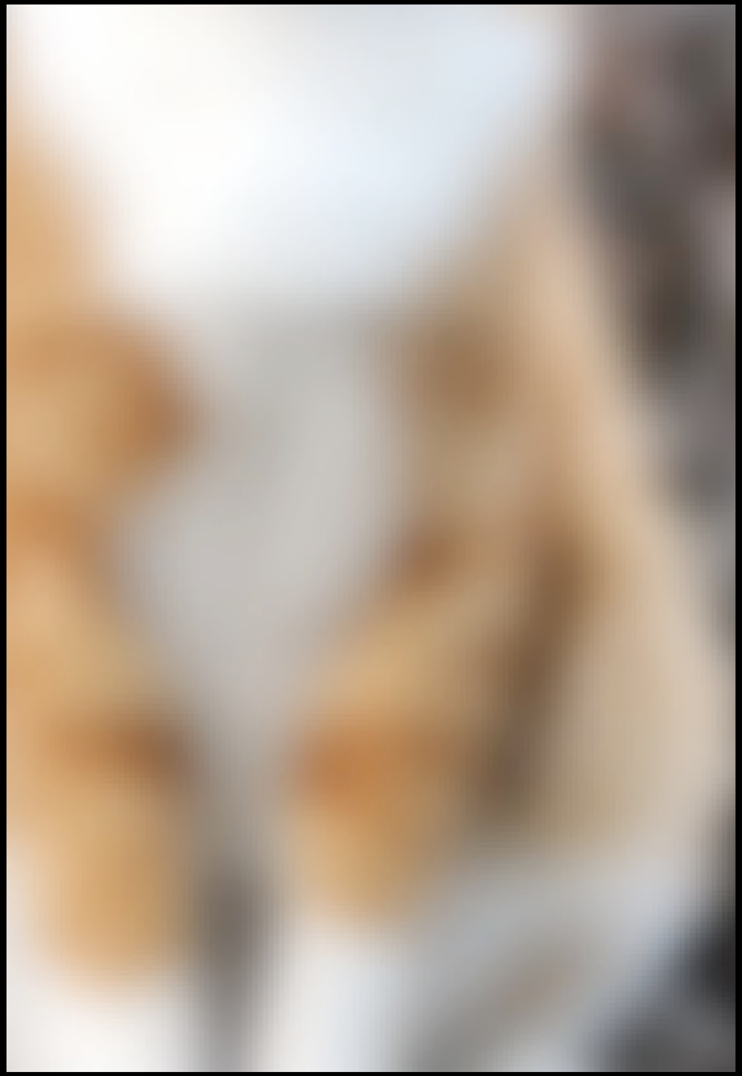


Box



Gaussian

Box vs. Gaussian

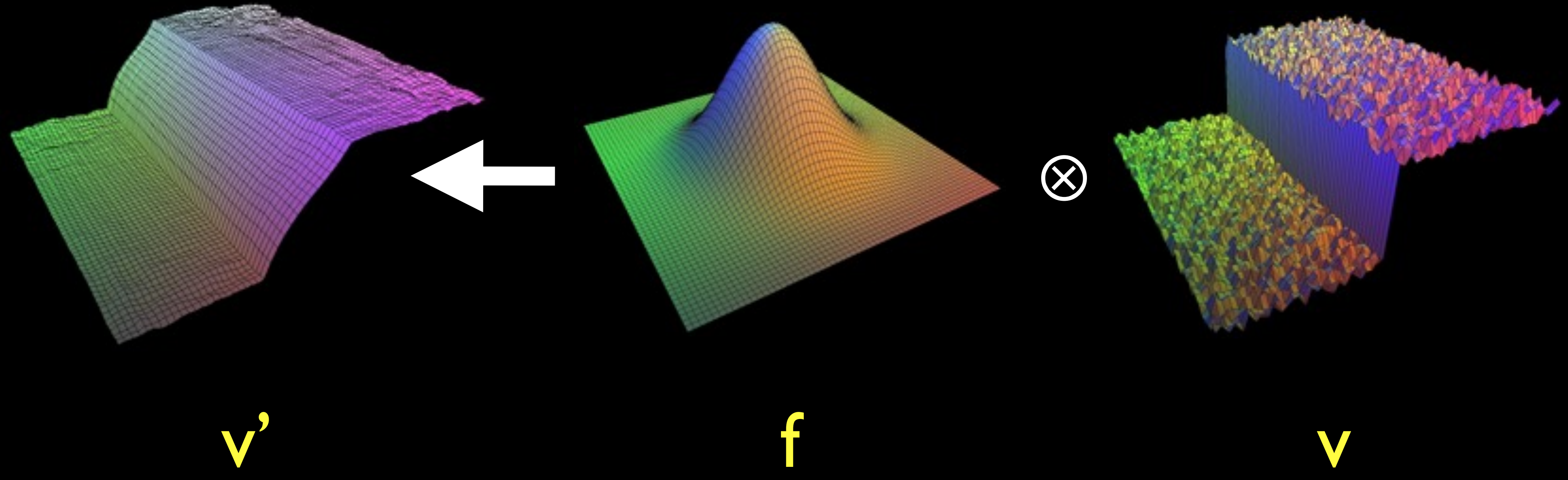


Gaussian

Box

Gaussian on Edges

- Averaging with neighbors.
  - Weights decay as *spatial* distance grows.
  - $v'(x) = \sum_y v(y)\, f(y\text{-}x)$.



Gaussian on Edges

- Why do we average with neighbors?
    - Trying to get a *better* estimate of local radiance
    - If so, why not average with neighbors that are more likely to have similar radiance?

Bilateral filtering

- Averaging with neighbors.
  - Weights decay as *spatial* distance grows.
  - Weights decay as *color* distance grows.

$$v'(x) = \underline{\sum_y} v(y)\, f(y-x)\, g(v(y)-v(x))$$

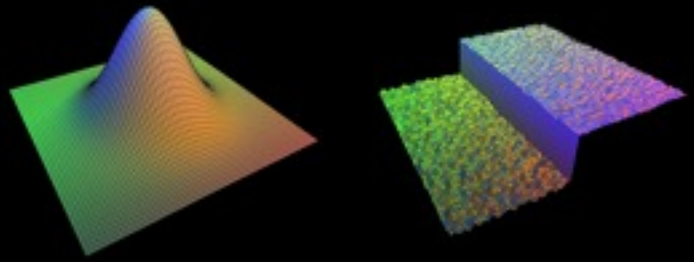
$$\sum_y f(y-x)\, g(v(y)-v(x)) = K(x)$$

weight on spatial distance

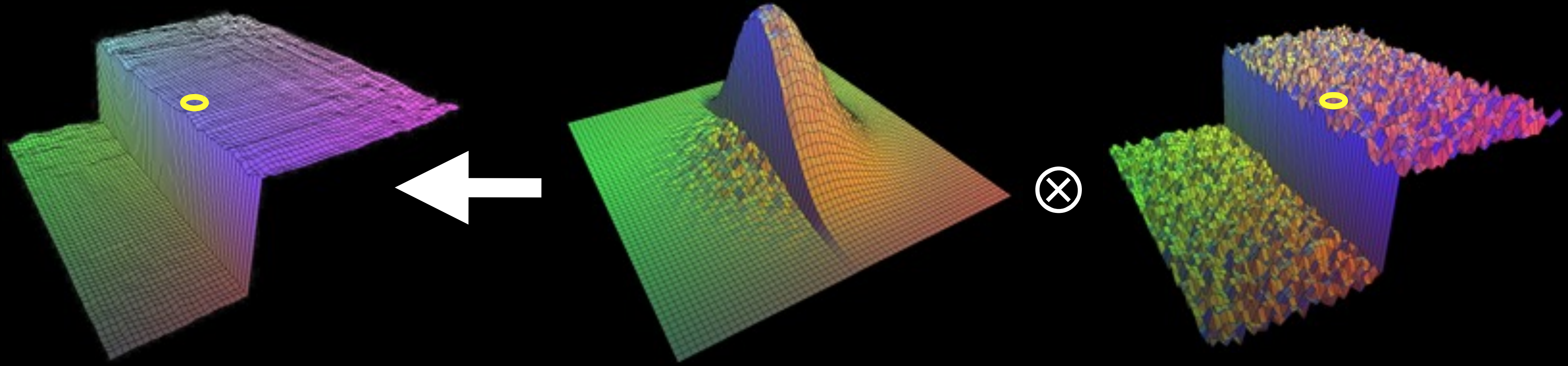
weight on color distance

Bilateral on Edges

- Averaging with neighbors.
  - $v'(x) = \sum'_y v(y)\, f(y\text{-}x)\, g(v(y)\text{-}v(x))$



(at a particular x)



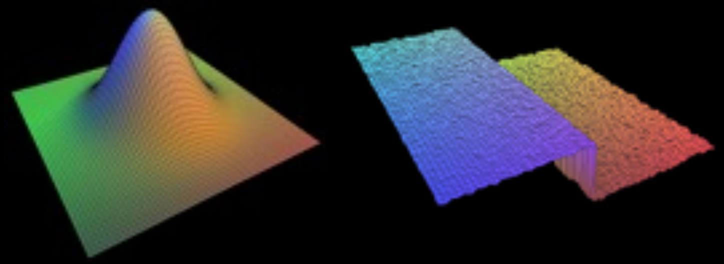
v’

f (at a particular x)

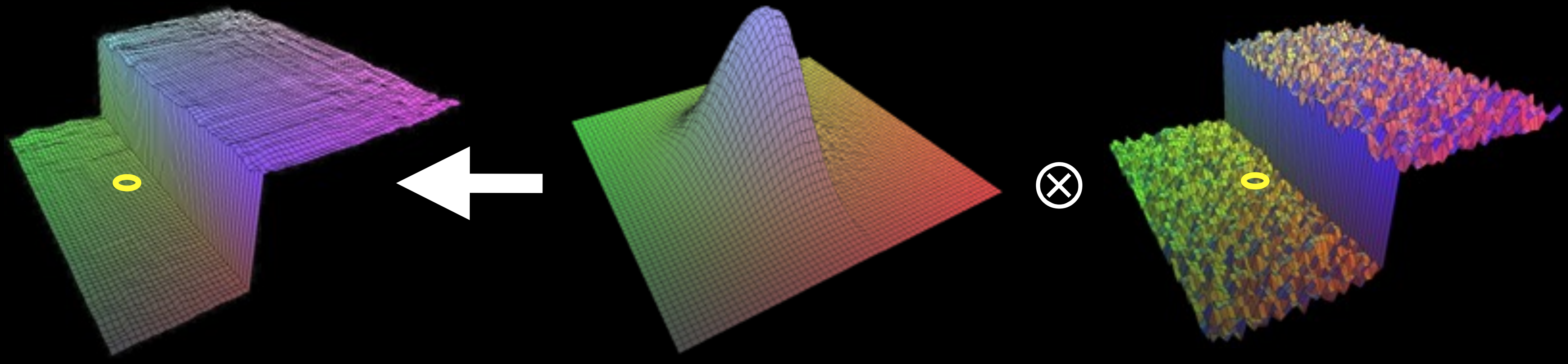
v

Bilateral on Edges

- Averaging with neighbors.
  - $v'(x) = \sum'_y v(y)\ f(y\text{-}x)\ g(v(y)\text{-}v(x))$



(at a particular x)



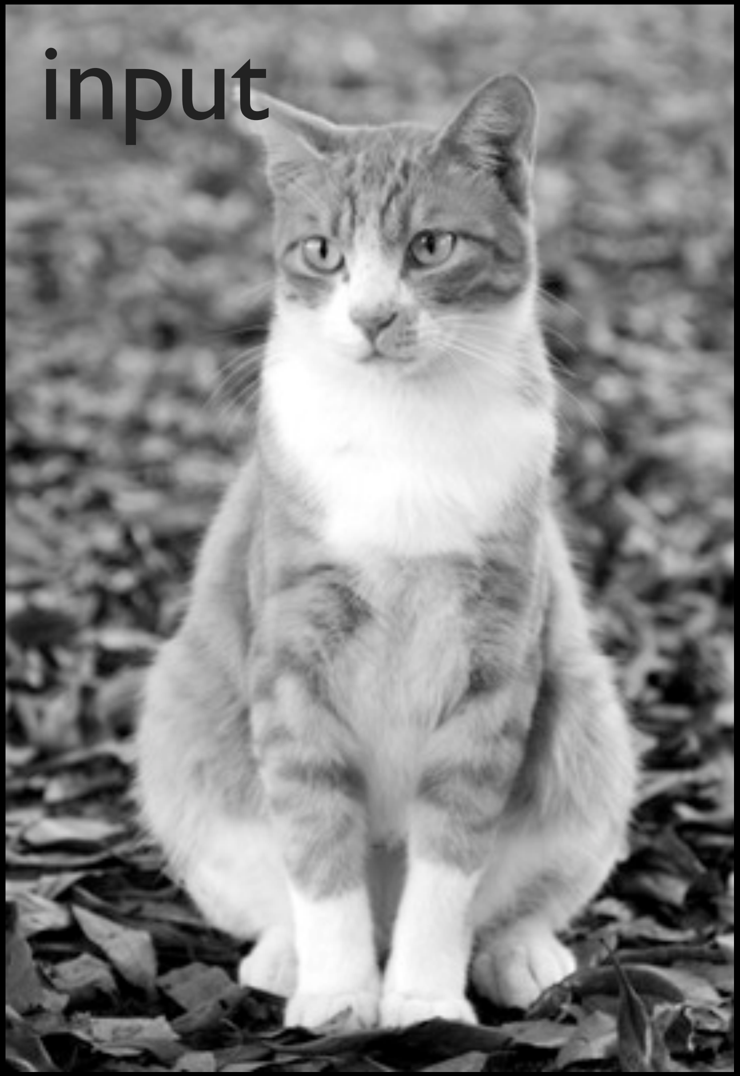
v'

f (at a particular x)

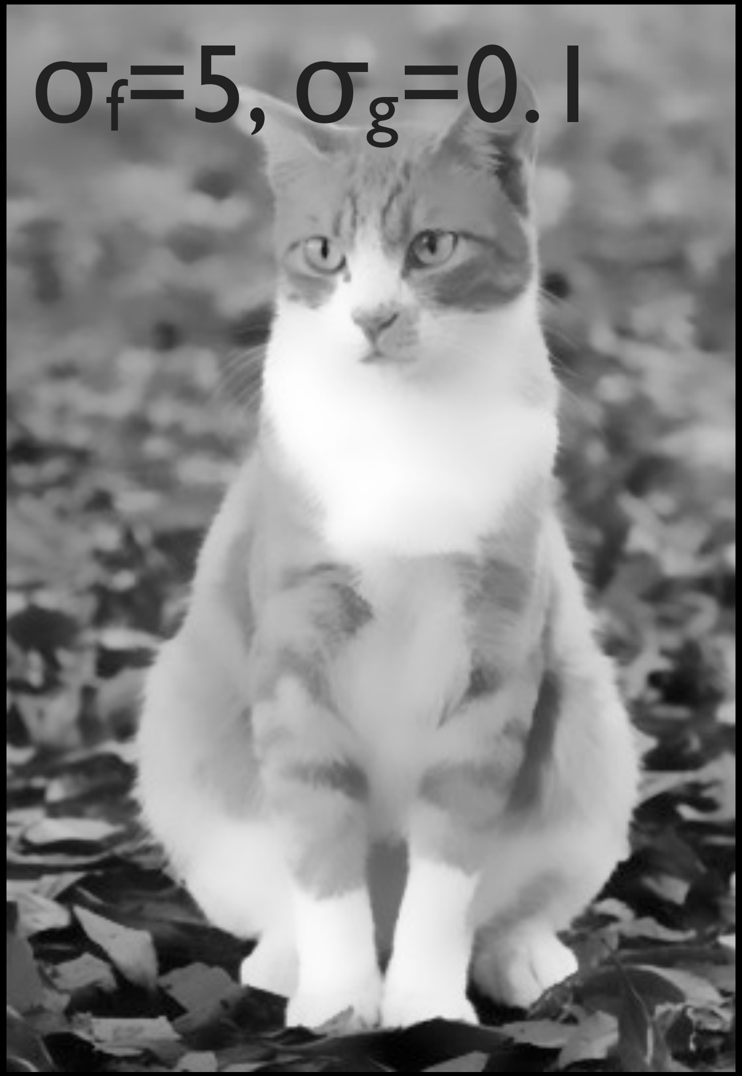
v

Bilateral Examples

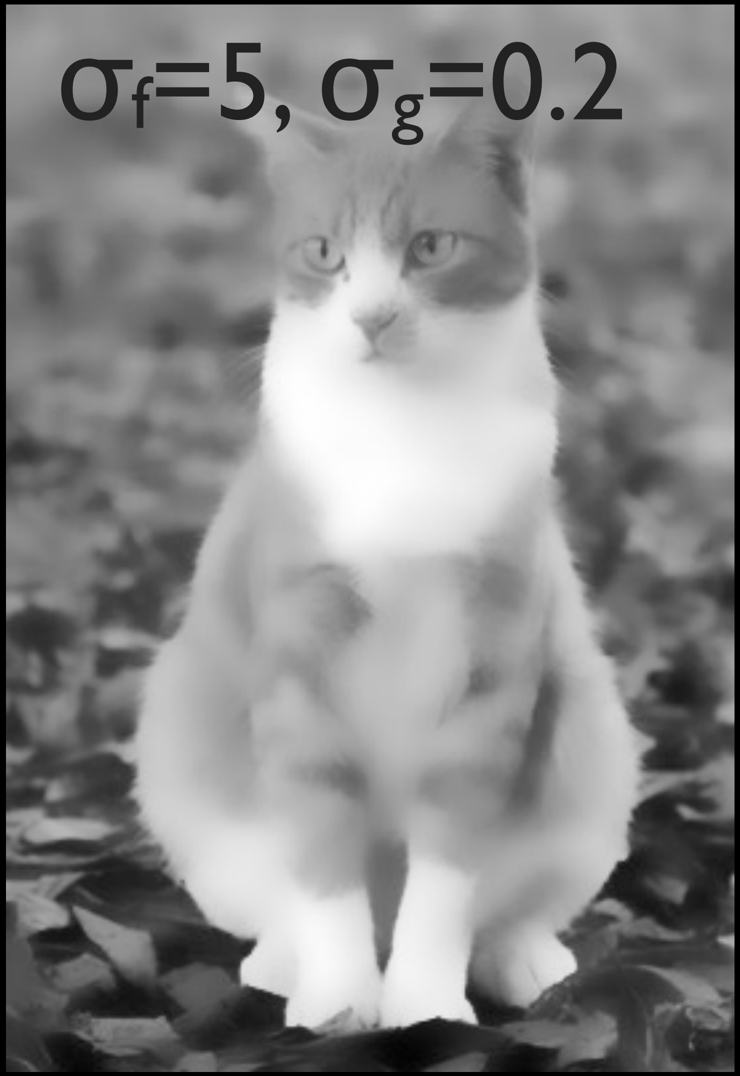
- $v'(x) = \sum'_y v(y)\, f(y-x)\, g(v(y)-v(x))$
  - The stdevs of f and g control filter strength.



input



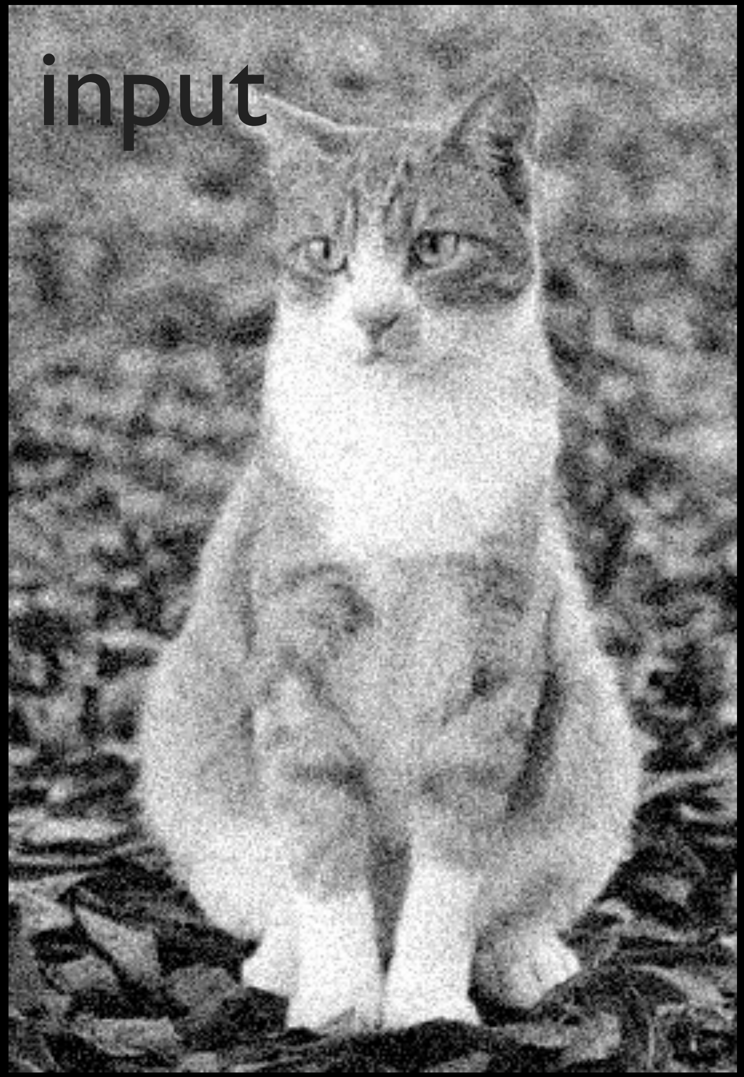
$\sigma_f=5, \sigma_g=0.1$

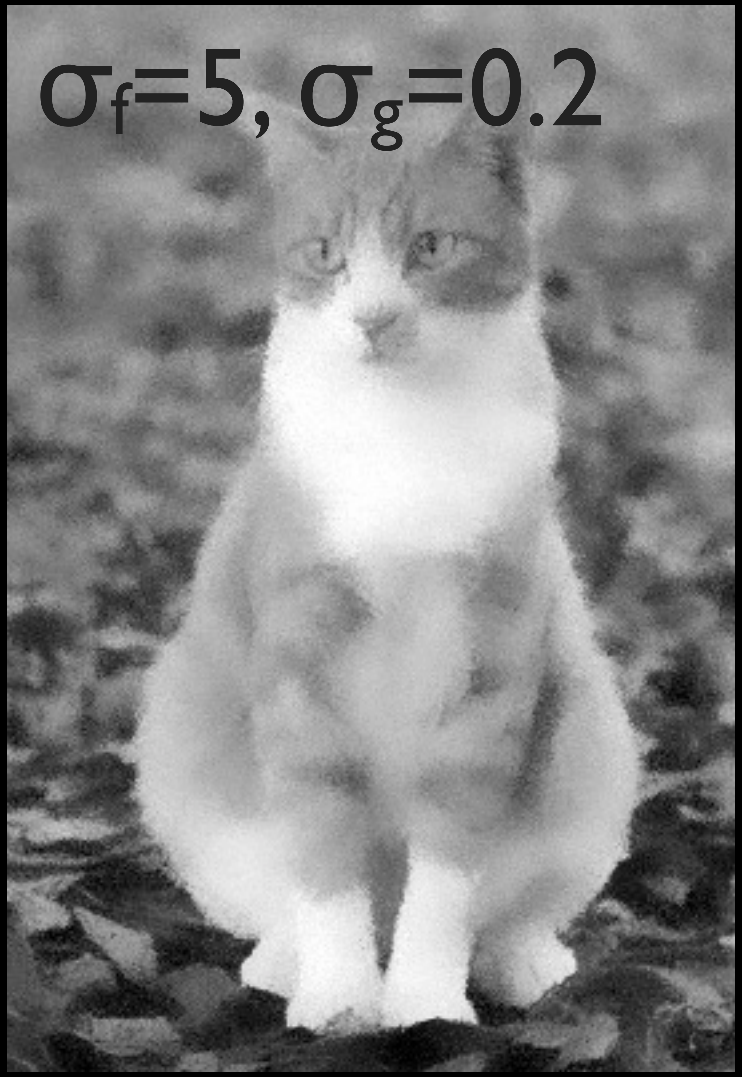


$\sigma_f=5, \sigma_g=0.2$

Applications

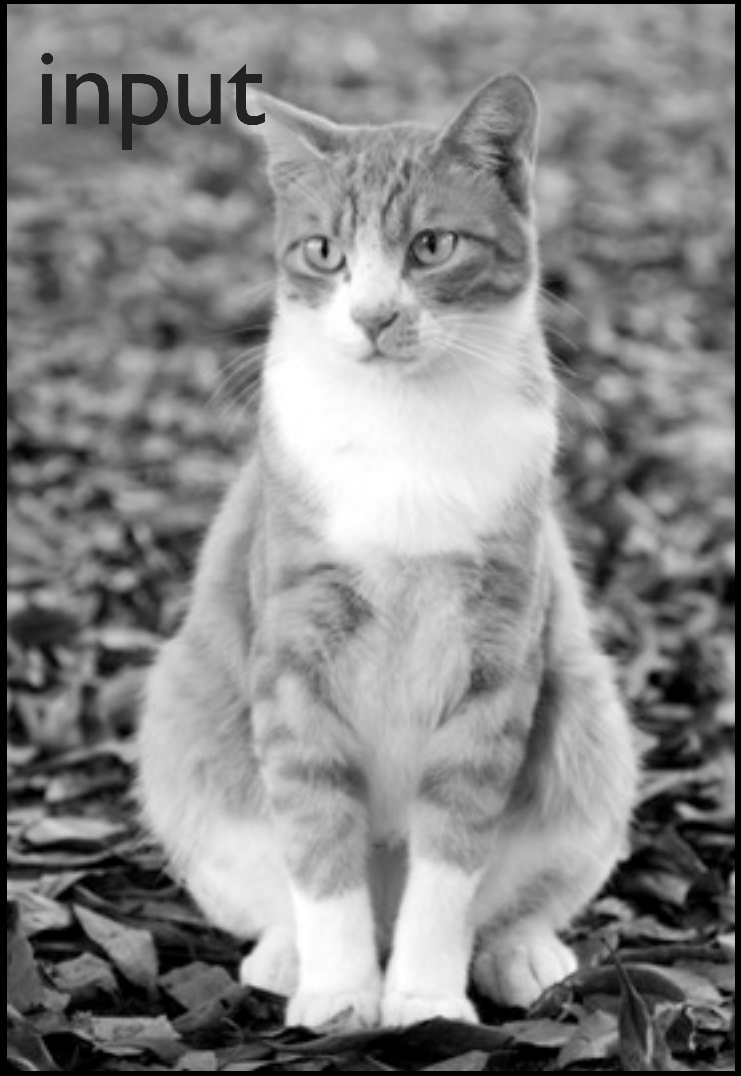
Light Denoising

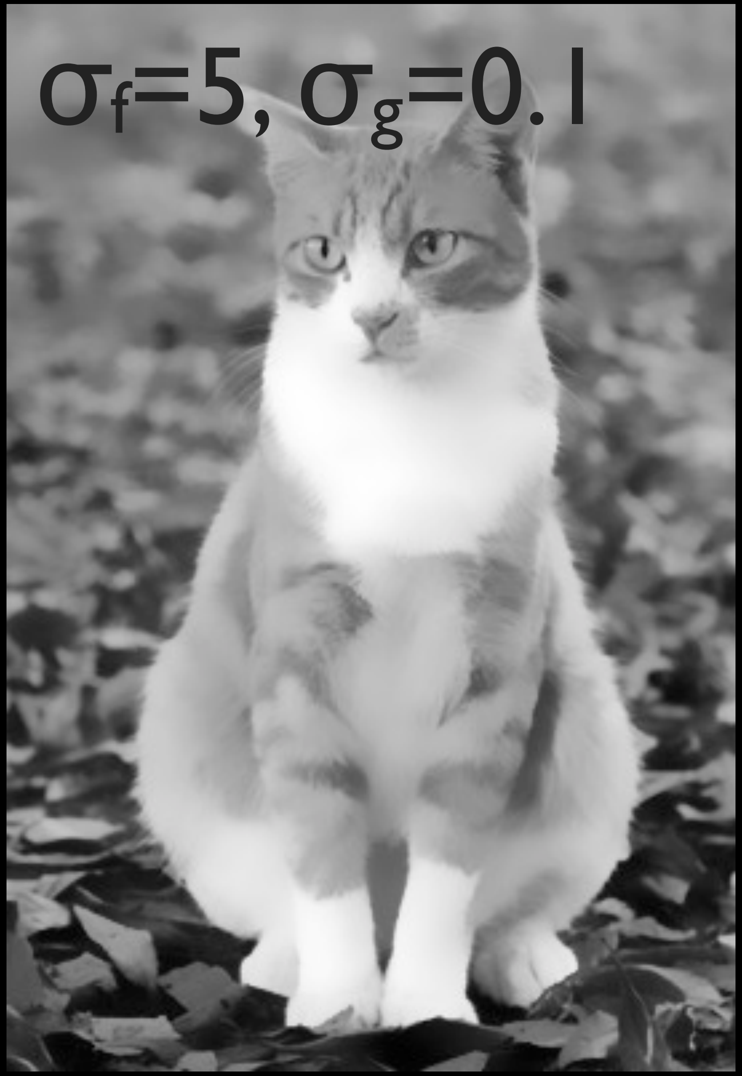




Applications

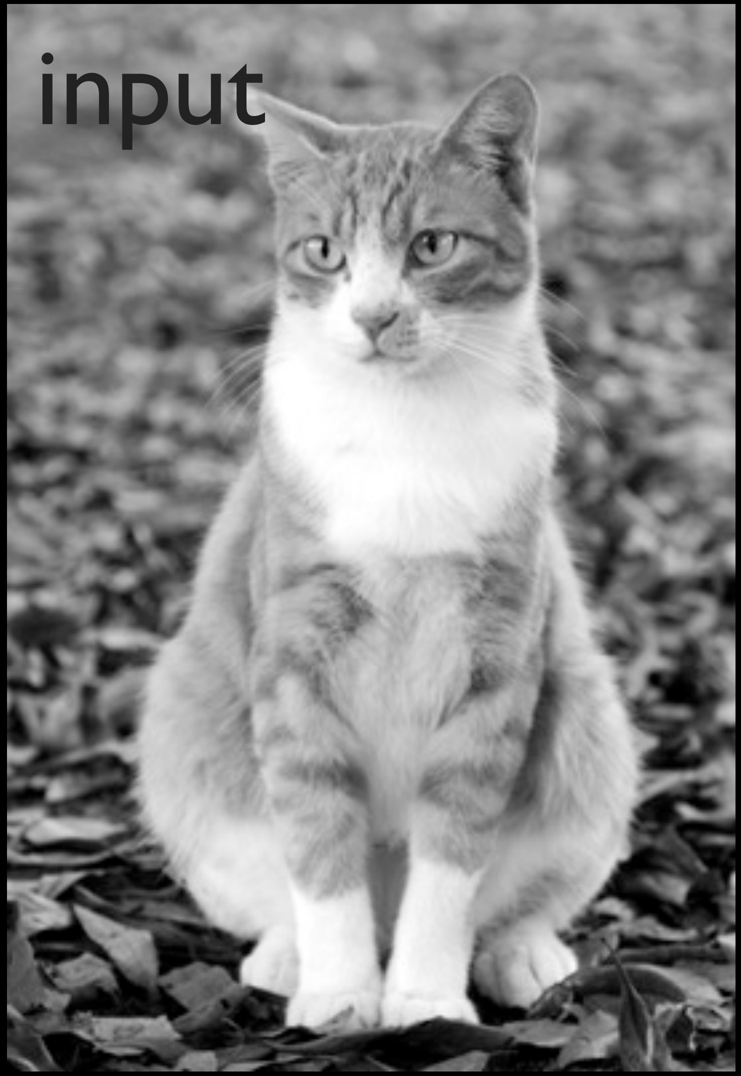
Non-Photorealistic Rendering



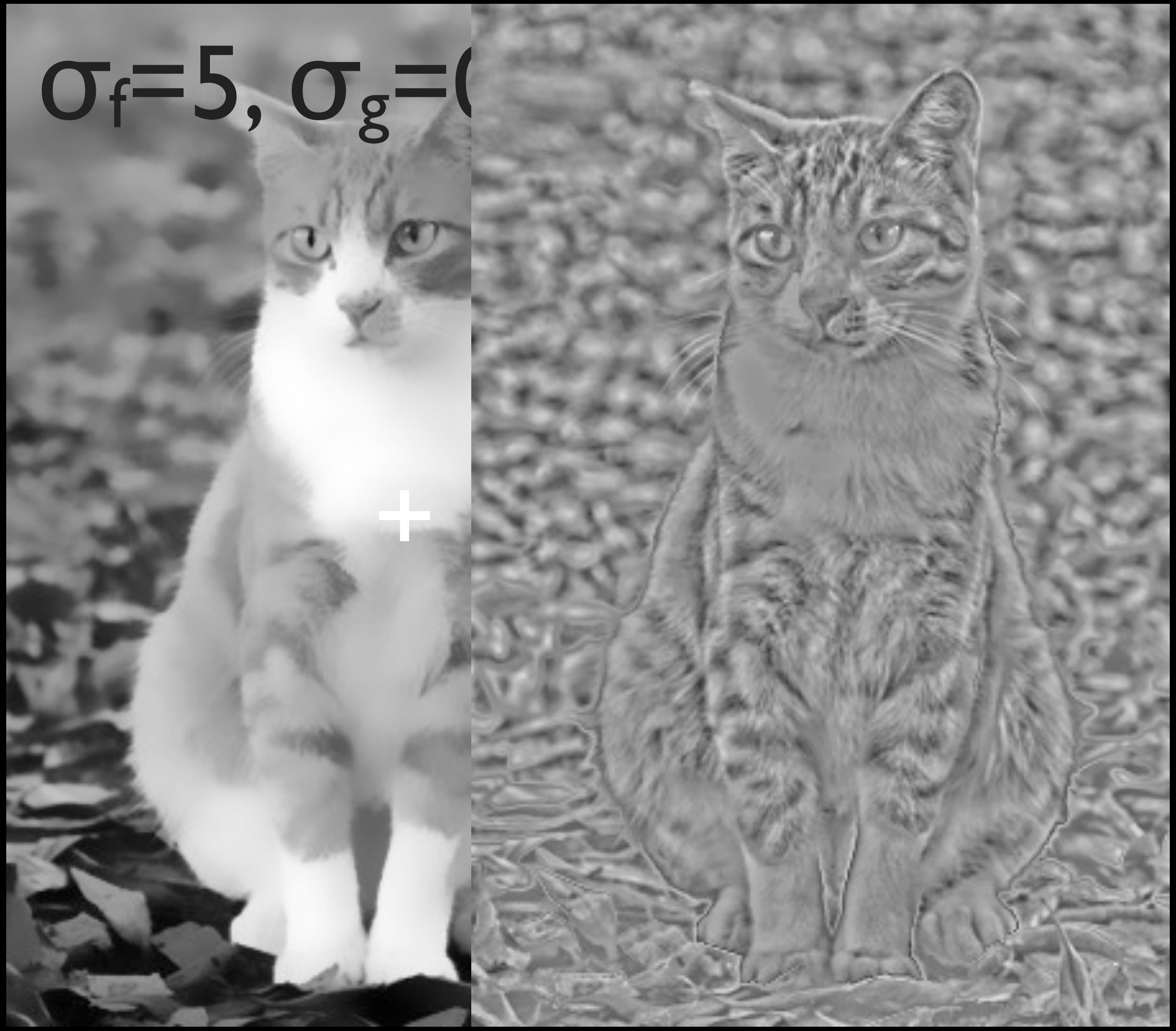


Applications

Detail Enhancement



=

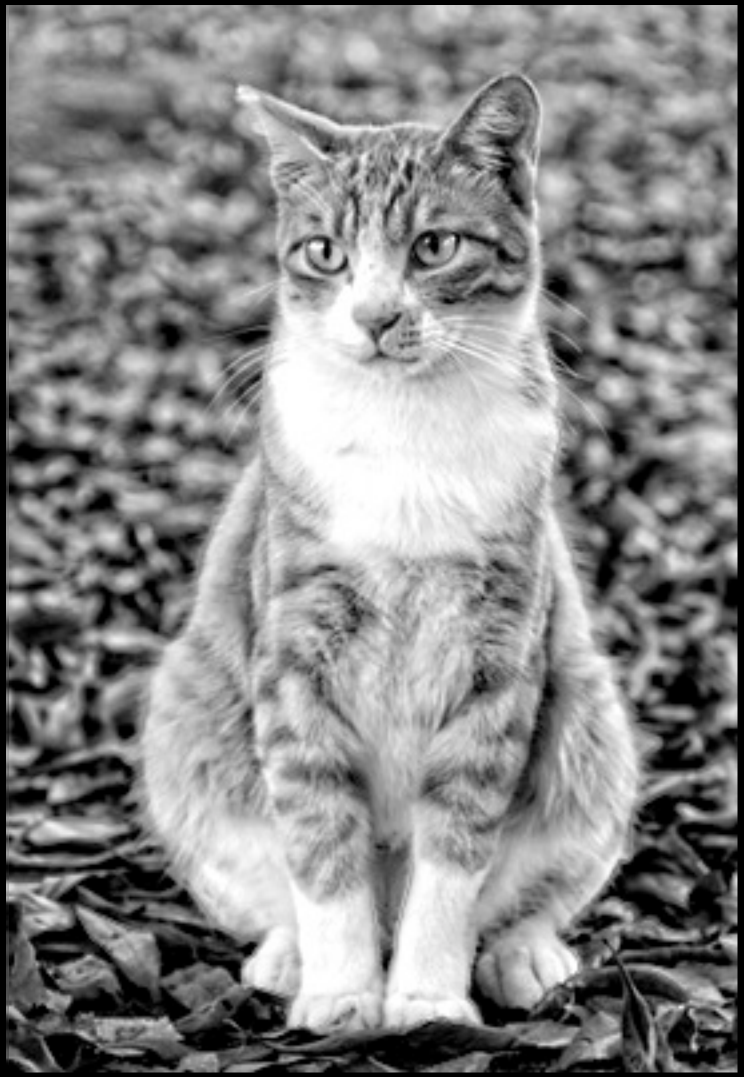


input minus detail

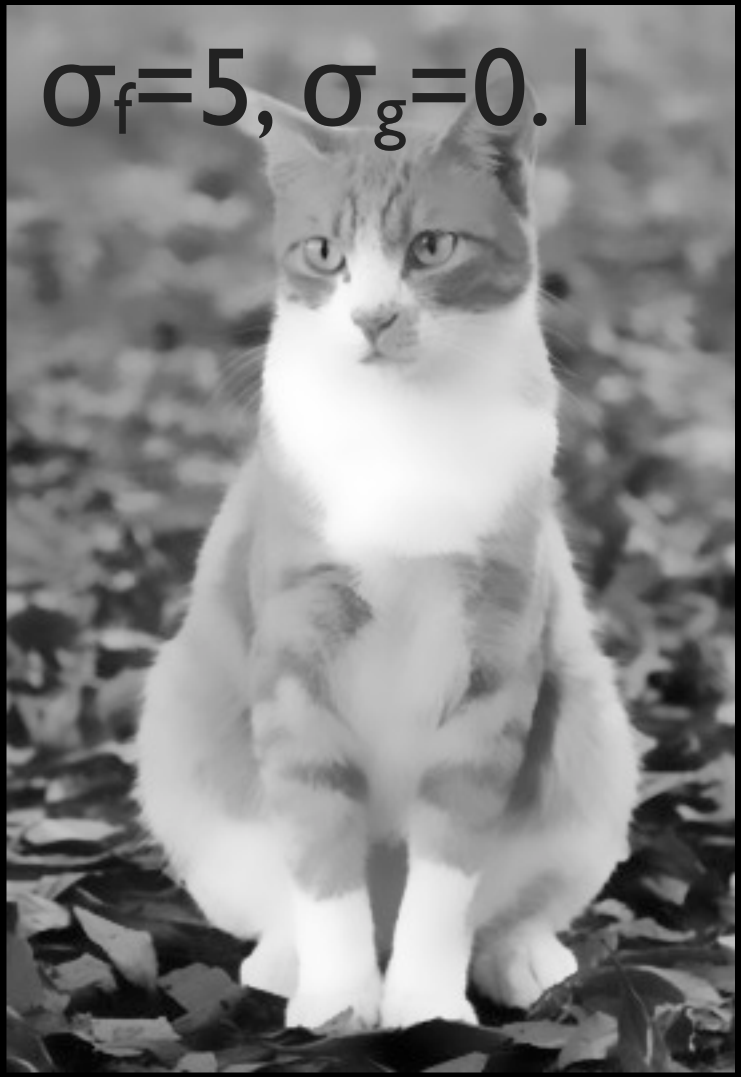
detail x 3

Applications

Detail Enhancement



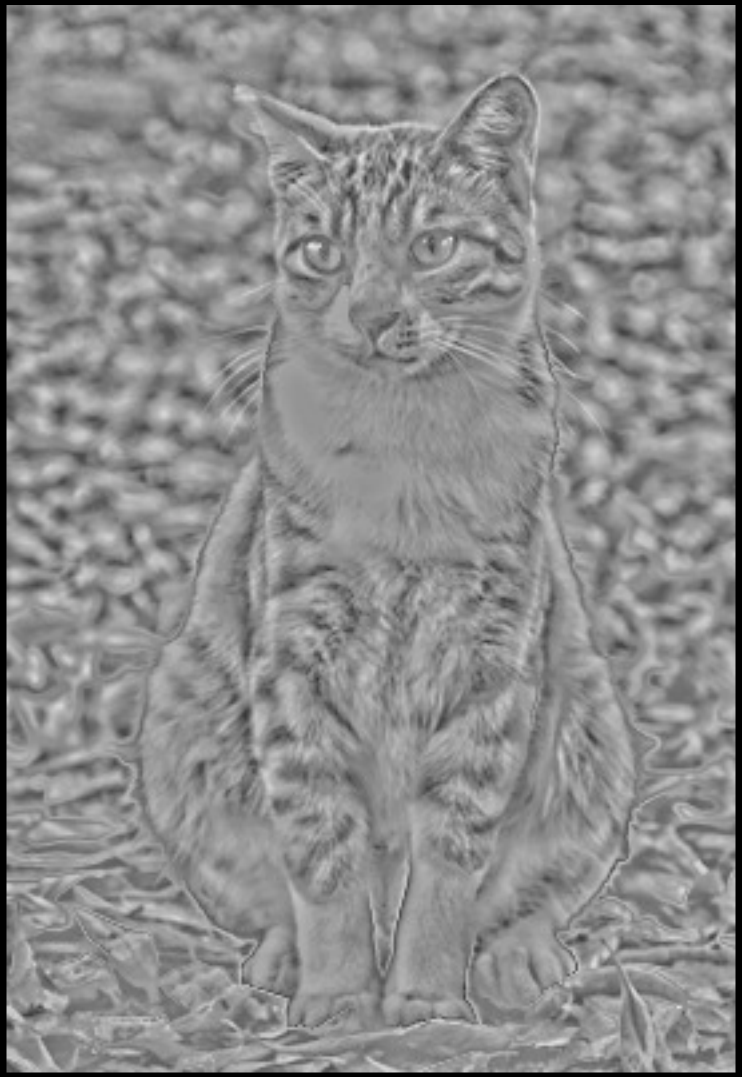
=



σ_f=5, σ_g=0.1

input minus detail

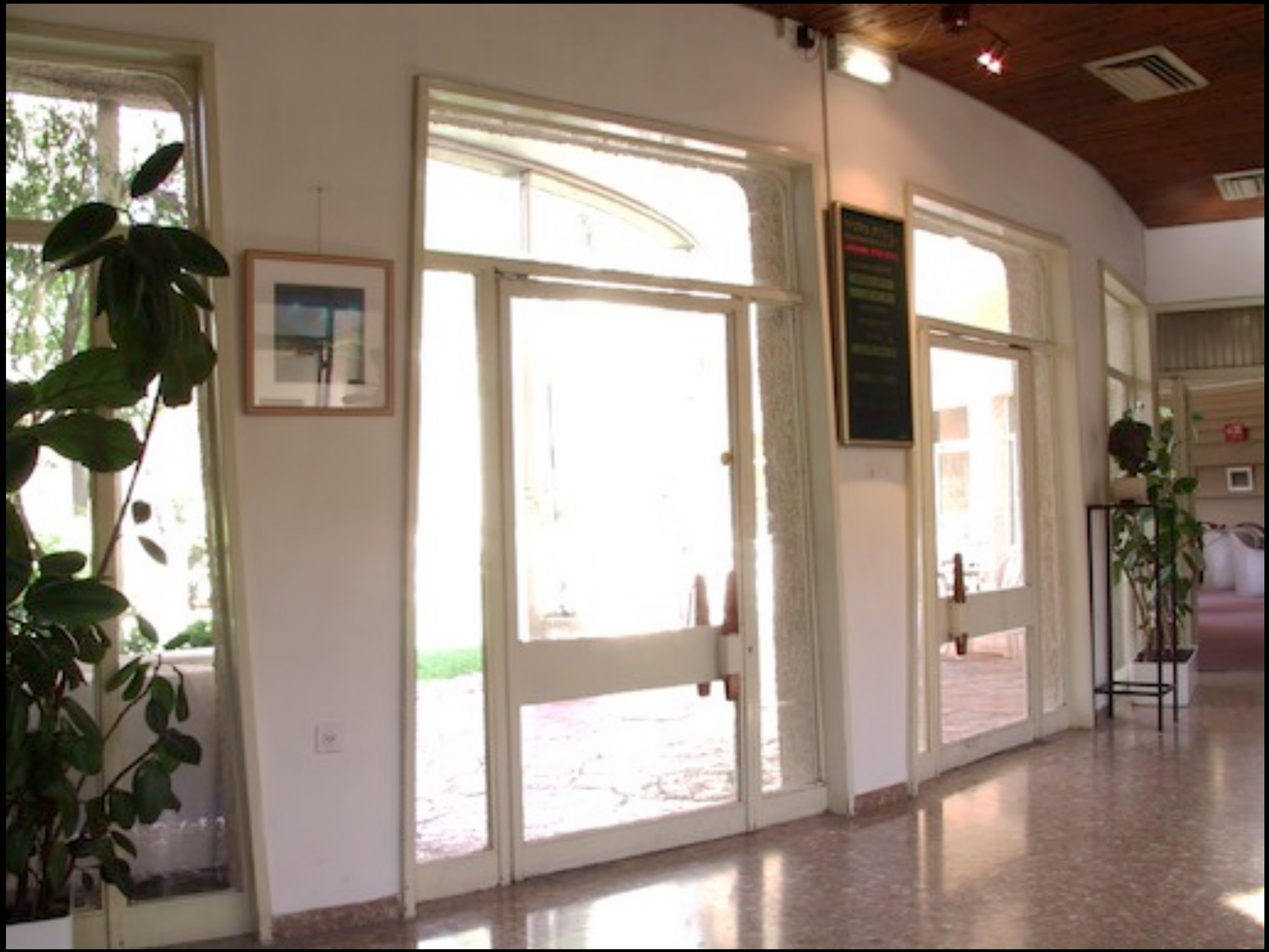
+



detail x 3

Applications

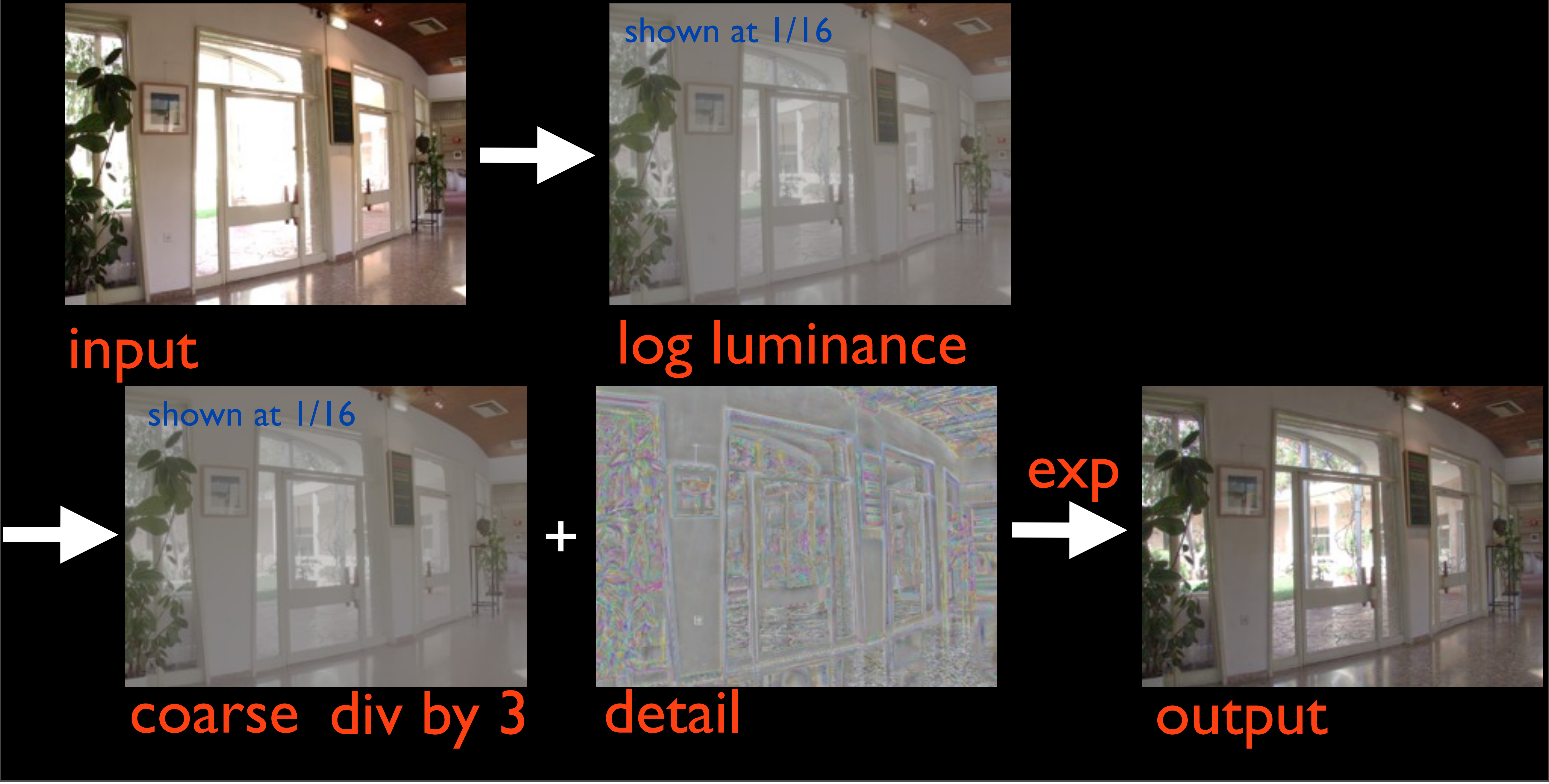
HDR Tone Mapping



gamma 0.6

Applications

HDR Tone Mapping



shown at 1/16

input

log luminance

shown at 1/16

coarse div by 3

+

detail

exp

output

Caveats

- Current formulation:

NEIGHBOR WEIGHT$_2$

$$v'(x) = \sum'_y v(y)\, f(y-x)\, g(v(y)-v(x))$$

NORMALIZED SUM WEIGHT$_1$

Naive implementation: $O(N^2)$

Truncate f: $O(N \sigma_f^2)$

Compare to regular gaussian: $O(N \sigma_f)$ or $O(N \log N)$

separable using FFT

Normalization

- Current formulation:

NEIGHBOR WEIGHT$_2$

$$v'(x) = \sum'_y v(y)\, f(y-x)\, g(v(y)-v(x))$$

NORMALIZED SUM WEIGHT$_1$

Filter an image whose pixels are all 1

using the same weights.

The resulting image = K(x)

Normalization

- Current formulation:

$$v'(x) = \underset{\text{NORMALIZED SUM}}{\sum'_y} \overset{\text{NEIGHBOR}}{v(y)} \underset{\text{WEIGHT}_1}{f(y-x)} \overset{\text{WEIGHT}_2}{g(v(y)-v(x))}$$

Equivalently, add a homogeneous channel to p.
De-homogenize later.

e.g. if p=(r,g,b), filter (r,g,b,1) instead.

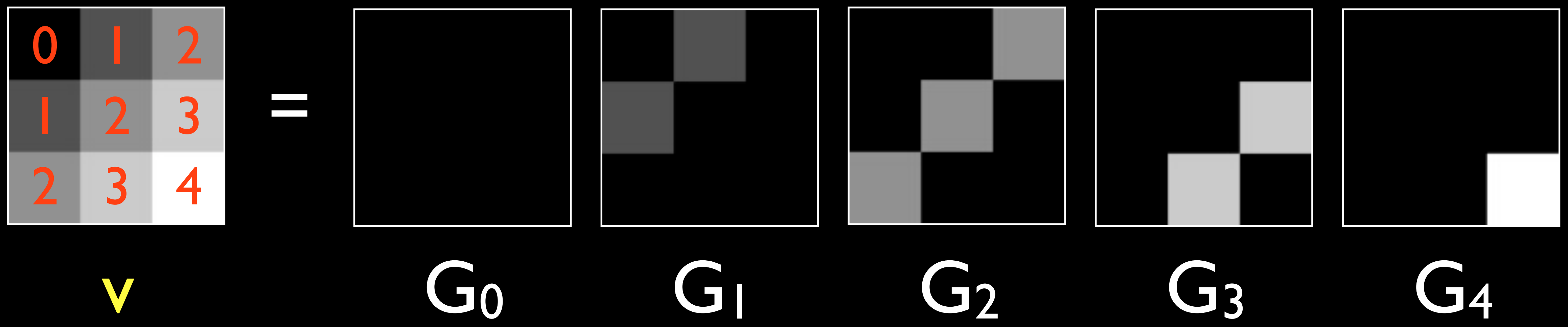
If the result is (r',g',b',k), compute (r'/k,g'/k,b'/k).

Acceleration #1

(Porikli, CVPR 2008)

$$v'(x) = \sum'_{y} v(y)\, f(x-y)\, g(v(x)-v(y))$$

- Partition pixels by value: G_0, G_1, ..., G_{255}.



0	1	2
1	2	3
2	3	4

Acceleration #1

(Porikli, CVPR 2008)

$$v'(x) = \sum'_y v(y)\ f(x-y)\ g(v(x)-v(y))$$

- Partition pixels by value: $G_0, G_1, \ldots, G_{255}$.
- Then, contribution from pixels in G_i

  $v'(x) = \sum_i \underline{\sum_{y \in G_i} v(y)\ f(x-y)\ g(v(x)-v(y))}$
- $= \sum_i \sum_{y \in G_i} \underline{i}\ f(x-y)\ \underline{g(v(x)-i)}$ independent of y
- $= \sum_i \underline{[\ \sum_{y \in G_i} f(x-y)\ ]}\ i\ g(v(x)-i)$

  gaussian blur of mask

Acceleration #1

(Porikli, CVPR 2008)

- $v'(x) = \sum_i [\underbrace{\sum_{y \in G_i} f(x-y)]}_{\text{gaussian blur of mask}} \; i \; \underbrace{g(v(x)-i)}_{\text{weight}}$
- For each pixel, do a weighted sum of the blurred masks.
- Runtime: O(256 N log N)

Not impressive?

Acceleration #1

(Porikli, CVPR 2008)

- $v'(x) = \sum_i \underbrace{\left[\sum_{y \in G_i} f(x-y) \right]}_{\text{box filter of mask}} i \, \underbrace{g(v(x)-i)}_{\text{weight}}$

  Box filter is O(1) amortized

- For each pixel, do a weighted sum of the blurred masks.
- Runtime: O(256 N)

Acceleration #1

(Porikli, CVPR 2008)

- $v'(x) = \sum_i \underbrace{[\ \sum_{y \in G_i} f(x-y)\ ]}_{\text{box filter of mask}}\ i\ \underbrace{g(v(x)-i)}_{\text{weight}}$

  Box filter is O(1) amortized

- For each pixel, do a weighted sum of the blurred masks.

- Runtime: O(32 N)

  Using fewer groups G_i

Acceleration #2

(Durand and Dorsey, SIGGRAPH 2002)

$$v'(x) = \sum'_y v(y)\, f(x-y)\, g(v(x)-v(y))$$

- Define $v_i(y) = v(y)\, g(i - v(y))$
- Apply gaussian blur to v_i to get w_i.
- Then,
  $v'(x) = \sum'_y f(x-y)\, v(y)\, g(v(x)-v(y))$
- $\qquad = \sum'_y f(x-y)\, v_{v(x)}(y)$
- $\qquad = w_{v(x)}(x)$

Acceleration #2

(Durand and Dorsey, SIGGRAPH 2002)

$v'(x) = w_{v(x)}(x)$ where

$w_i = f \otimes v_i$

- Need to compute each w_i.
  - 256 gaussian blurs..., one for each $i \in [0,255]$
- In practice, can sample i to be of fewer values.
  - O(32 N log N)

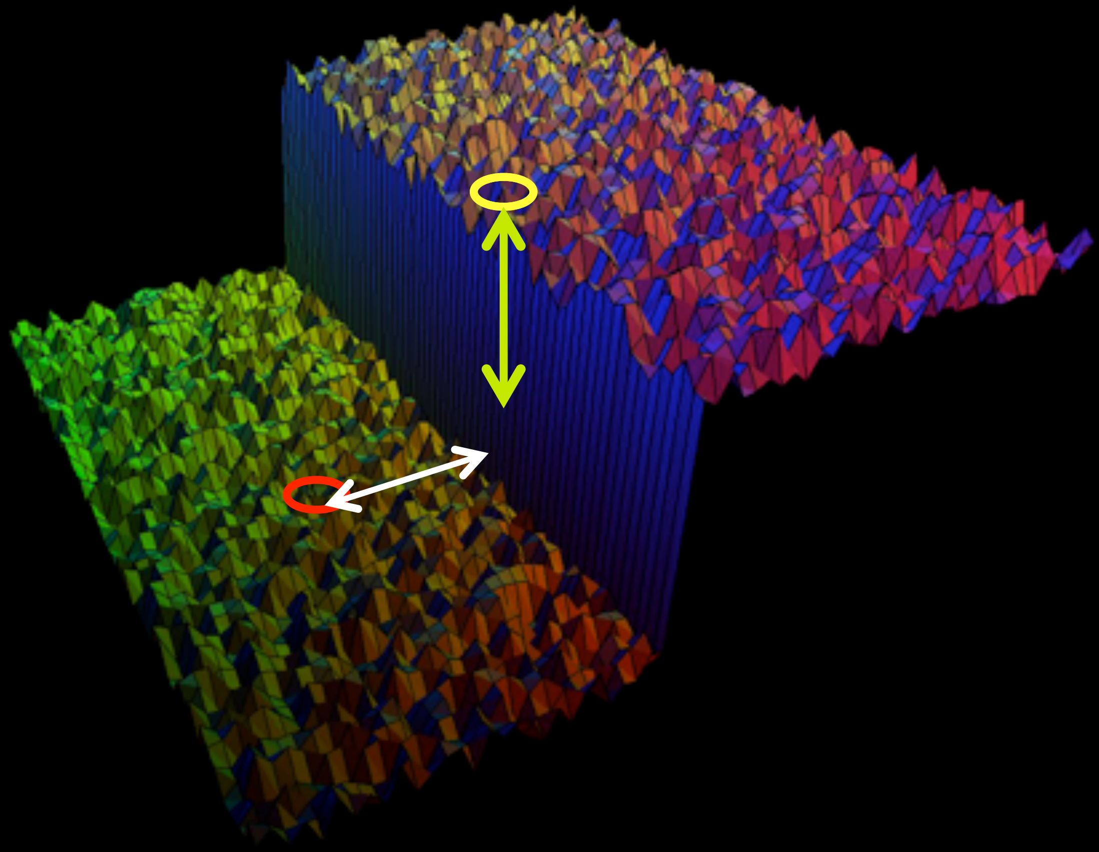
Acceleration #1,2

- Downsides?
    - We're grouping pixels by intensity. This works for grayscale image (d=1)
    - Runtime exponential in d.
        - The set of possible intensity vectors grow fast!

Let’s Generalize

$$v'(x) = \sum'_y v(y)\, f(x-y)\, g(v(x)-v(y))$$

The weight is a 3D distance.

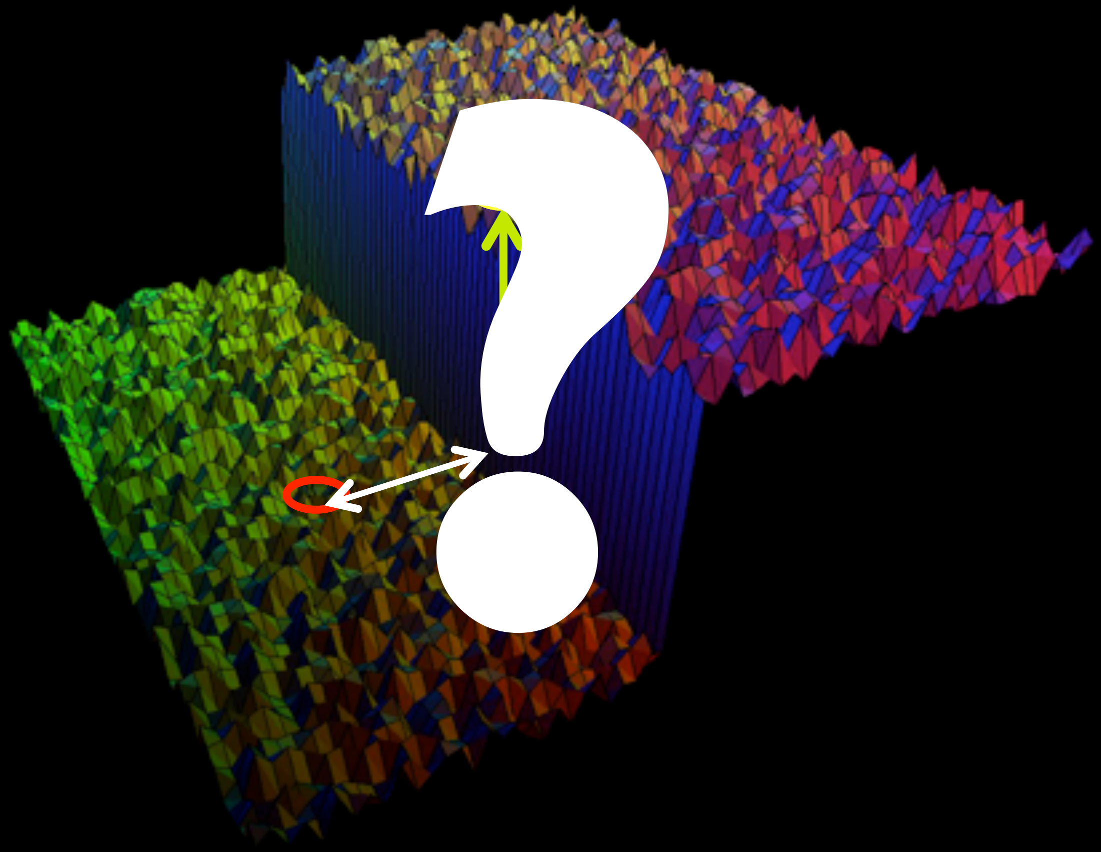


Let’s Generalize

$$v'(x) = \sum'_{y} v(y)\, f(\ \underline{p(x)}\ -\ \underline{p(y)}\ )$$

The weight is a 3D distance.

positions in
some arbitrary
space



Examples

$$v'(x) = \sum{}'_y\, v(y)\; f(\, \underline{\;p(x)\;} - \underline{\;p(y)\;}\,)$$

positions in some arbitrary space

- Grayscale bilateral:
  - $v(x,y)=\{I_{x,y}\}$
  - $p(x,y)=\{x, y, I_{x,y}\}$
- Color bilateral
  - $v(x,y)=\{R_{x,y}, G_{x,y}, B_{x,y}\}$
  - $p(x,y)=\{x, y, R_{x,y}, G_{x,y}, B_{x,y}\}$

Joint Bilateral Filtering

$$v'(x) = \sum'_y v(y)\ f(\underline{\ p(x)\ } - \underline{\ p(y)\ })$$

positions in some arbitrary space

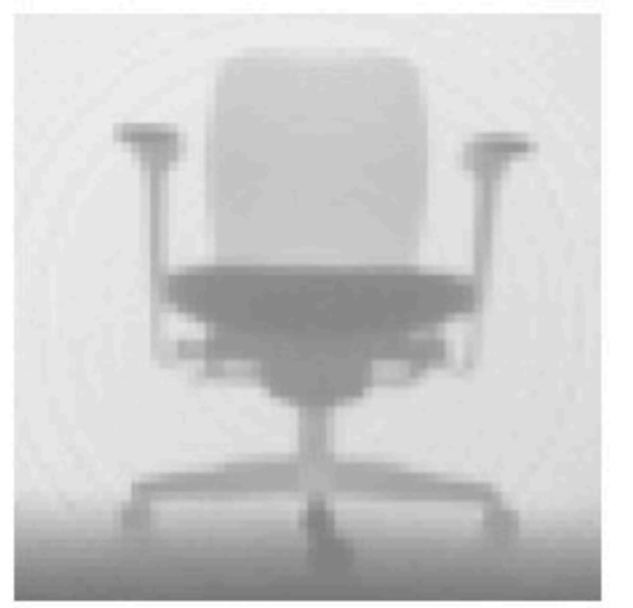
- There is no reason for which v and p should use the same RGB values.
  - $v(x,y)=\{R^1_{x,y}, G^1_{x,y}, B^1_{x,y}\}$
  - $p(x,y)=\{x, y, R^2_{x,y}, G^2_{x,y}, B^2_{x,y}\}$
- Blur an image while respecting edges in another image!

More Applications

Sensor Fusion

Scene Image

p



Coarse Depthmap

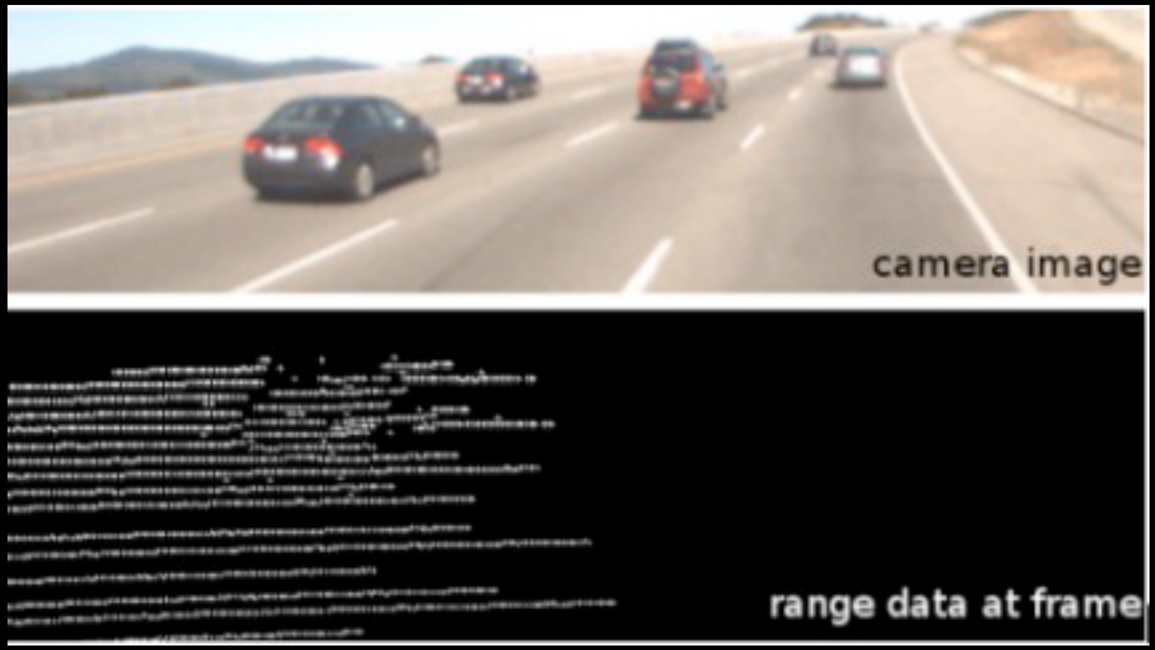
v

More Applications

Sensor Fusion

Scene Image

p



Sparse Depthmap

v

More Applications

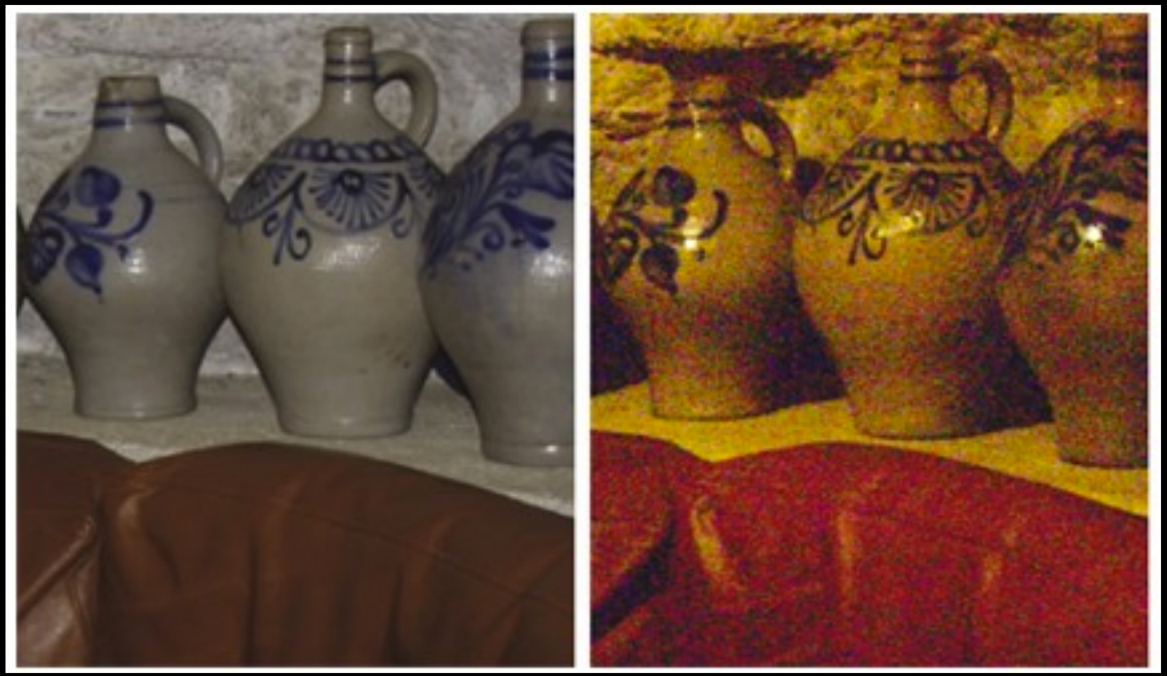
Selection Propagation

http://www.youtube.com/watch?v=e7kLRIlwHPc&t=3m36s

p : input image
v : map of (sparse) user strokes

More Applications

Flash-No-Flash Denoising

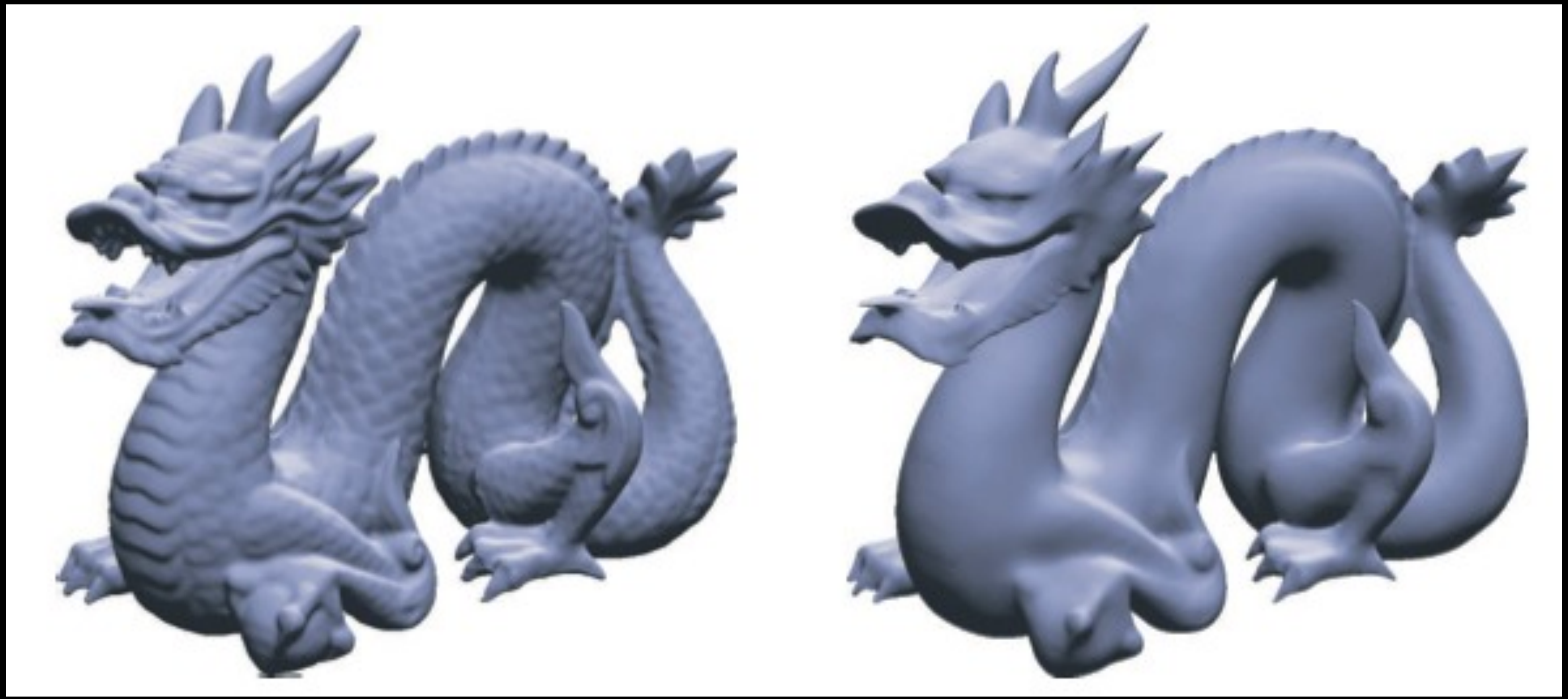


Flash Image p

No-Flash Image v

More Applications

Mesh Smoothing



Input
Mesh
v

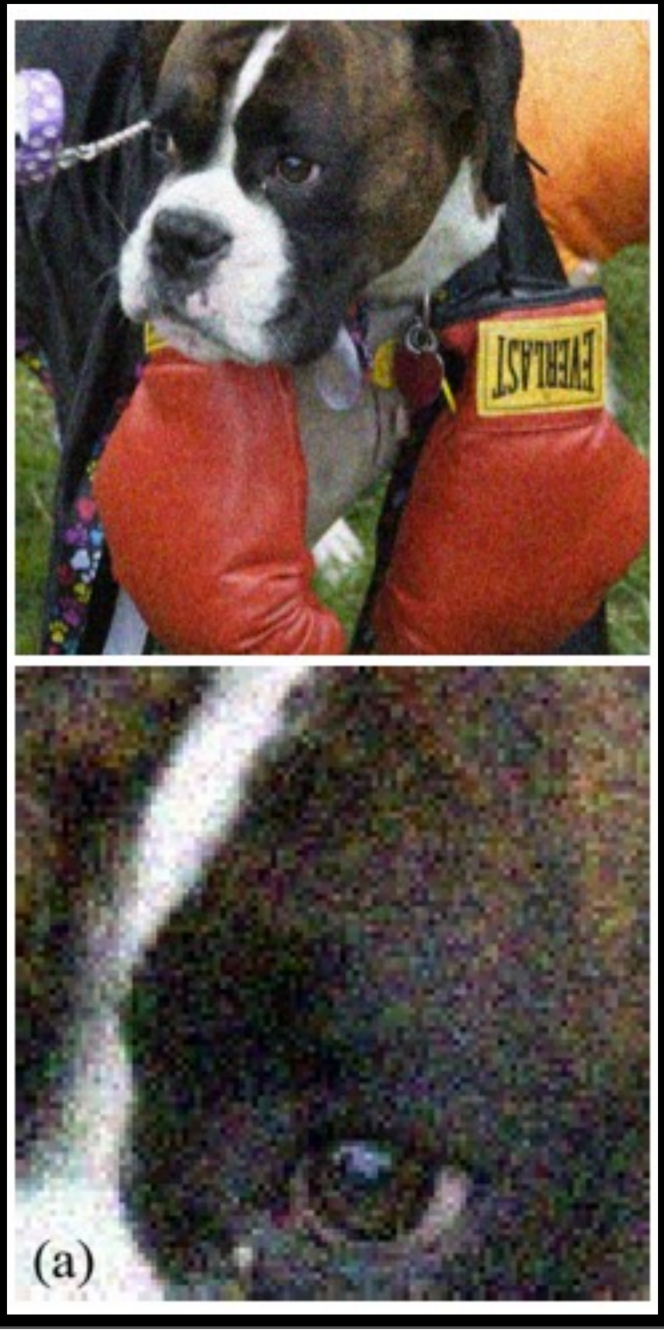
Output
Mesh
v'

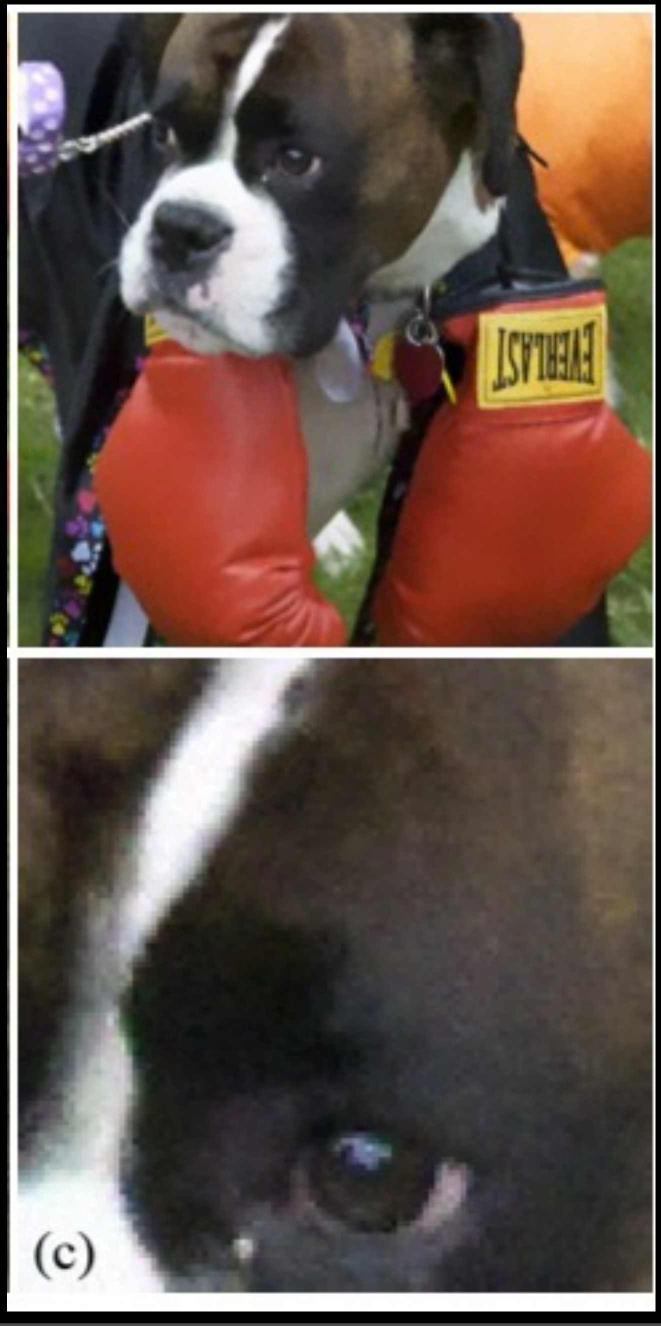
p is a local
descriptor of
each vertex

More Applications

Non-Local Means Denoising

Input
v





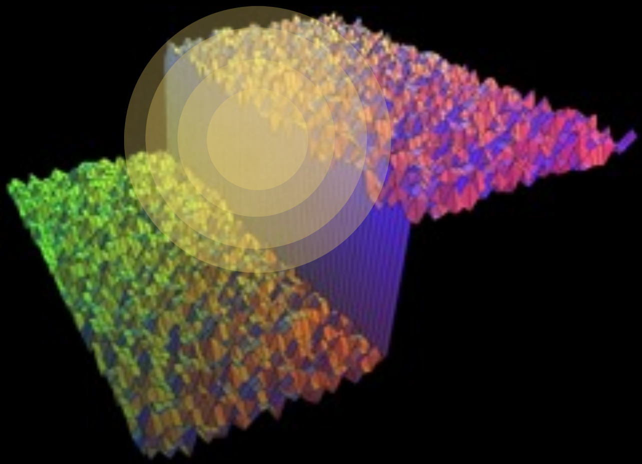
Output
v’

p is a local descriptor of the patch around each pixel

Acceleration #3 and on

$$v'(x) = \sum'_y v(y)\, f(p(x) - p(y))$$

- Let's think about this in a different way.
  - We have a high dimensional signal v that lives in the space of p.



This is a linear filter in this space!

$$v(p^{-1}(X)) = \sum'_Y v(p^{-1}(Y))\, f(X - Y)$$

Take $v \cdot p^{-1}$ and do a gaussian blur!

High-Dimensional Gauss Transform

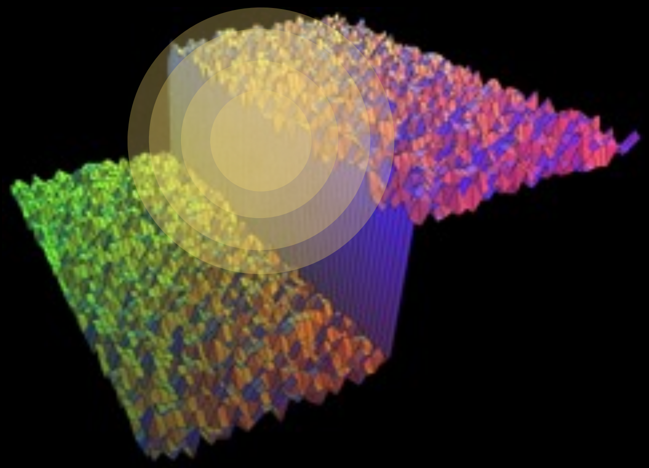
$$v'(x) = \sum'_y v(y)\, f(p(x) - p(y))$$

↓

$$\hat{v}_i = \sum_j e^{-|p_i - p_j|^2/2} v_j$$

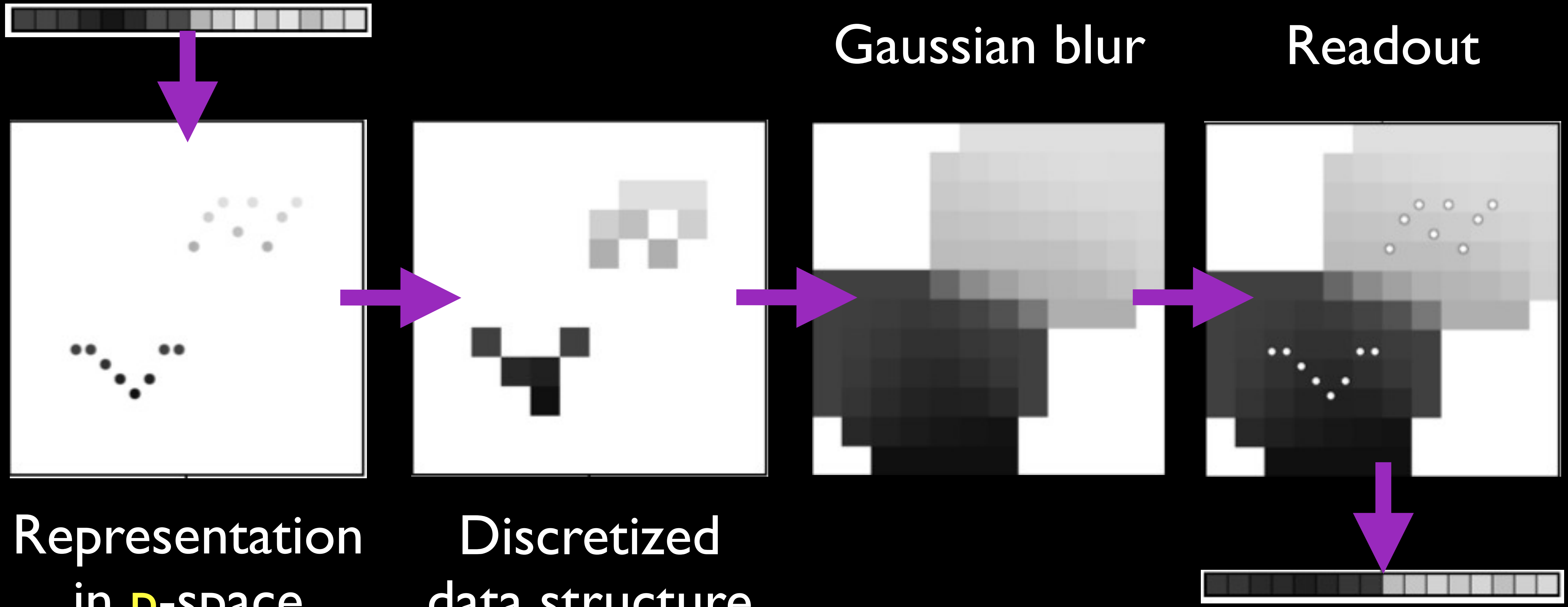
High-Dimensional Gauss Transform

- Take a high-dimensional signal.
- Put it into a data structure.
- Perform a Gaussian blur really fast.
- Read out its values.



High-Dimensional Gauss Transform

Input v



Gaussian blur

Readout

Representation
in p-space

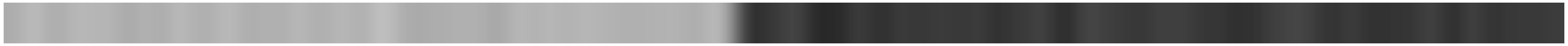
Discretized
data structure

Output v'

What data structure to use?

Explicitly represent position–space

- Consider a bilateral filter of this 1D grayscale signal

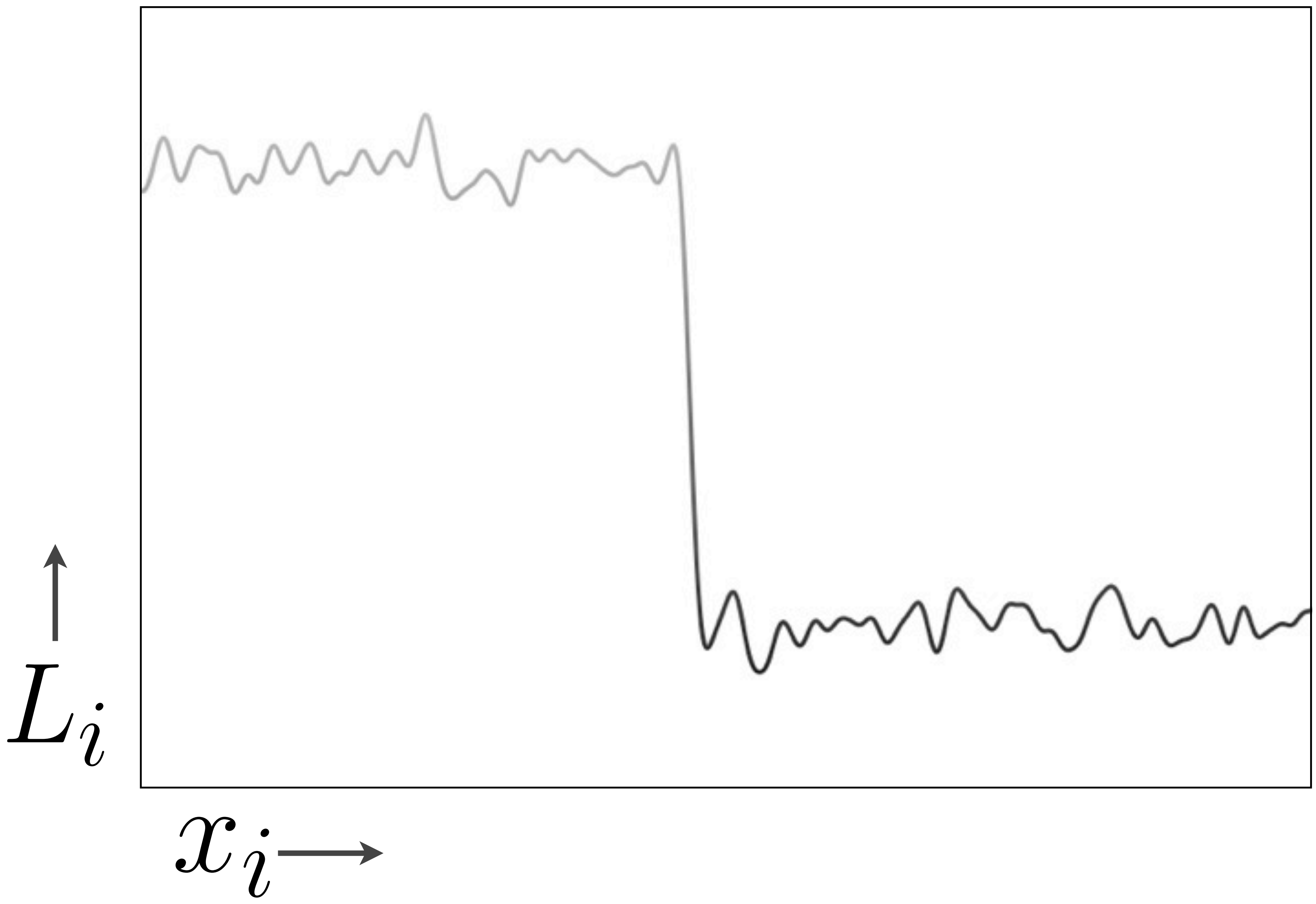


$$p_i = [x_i \ L_i] \qquad v_i = [L_i \ 1]$$

Slides stolen from Andrew Adams

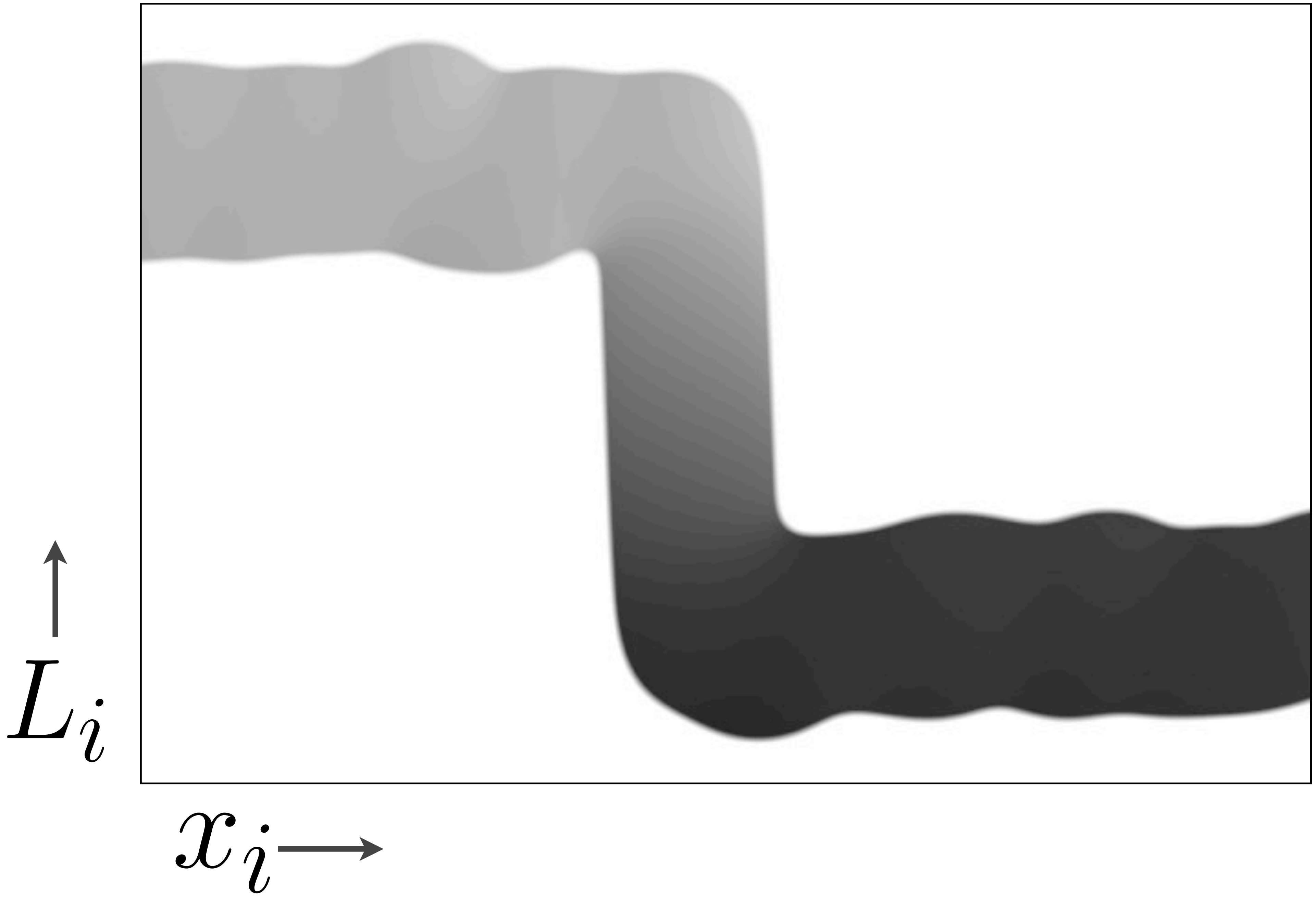
Splat -> Blur -> Slice

- Embed the signal in position-space



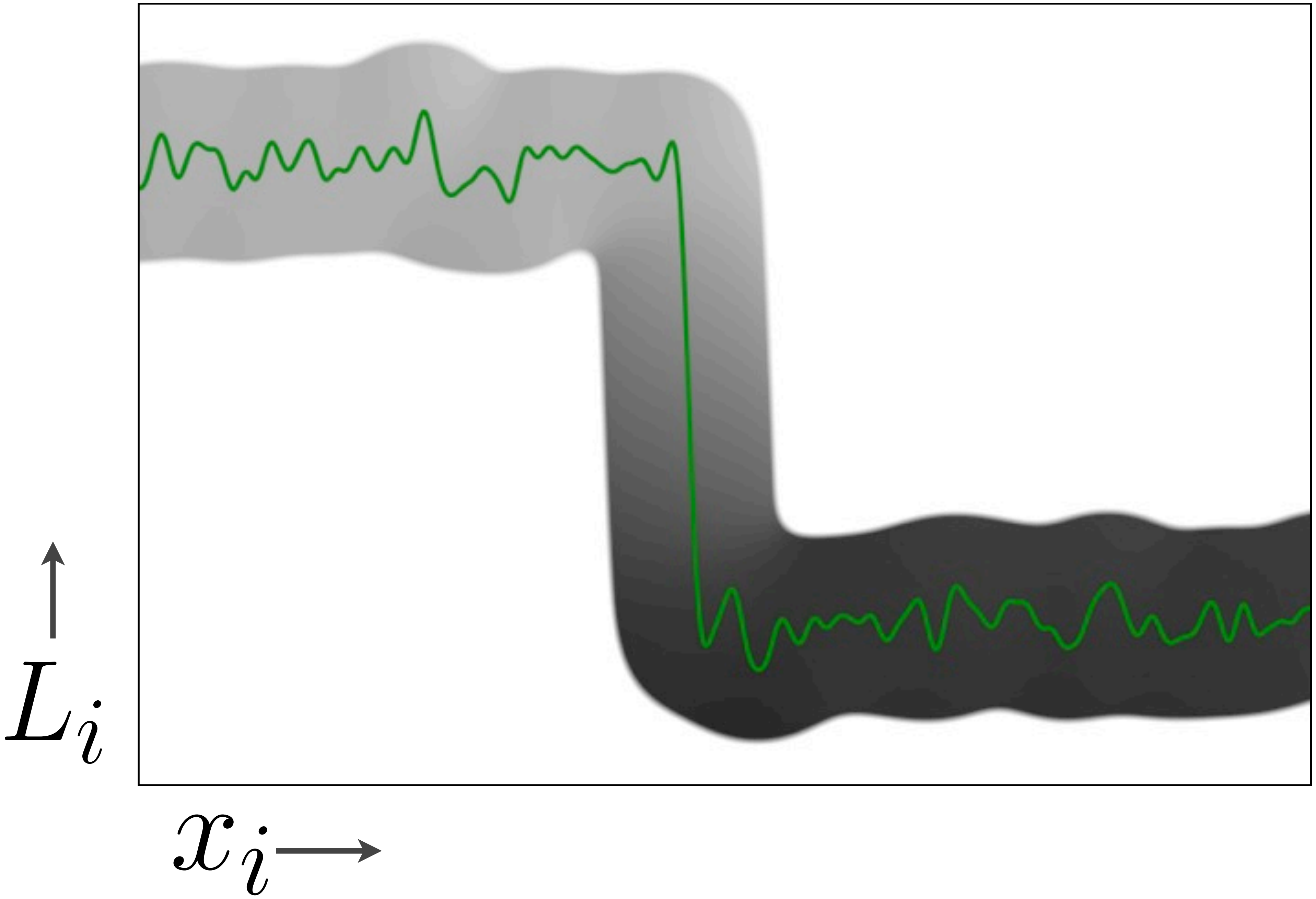
Splat -> Blur -> Slice

- Perform a Gaussian blur in that space



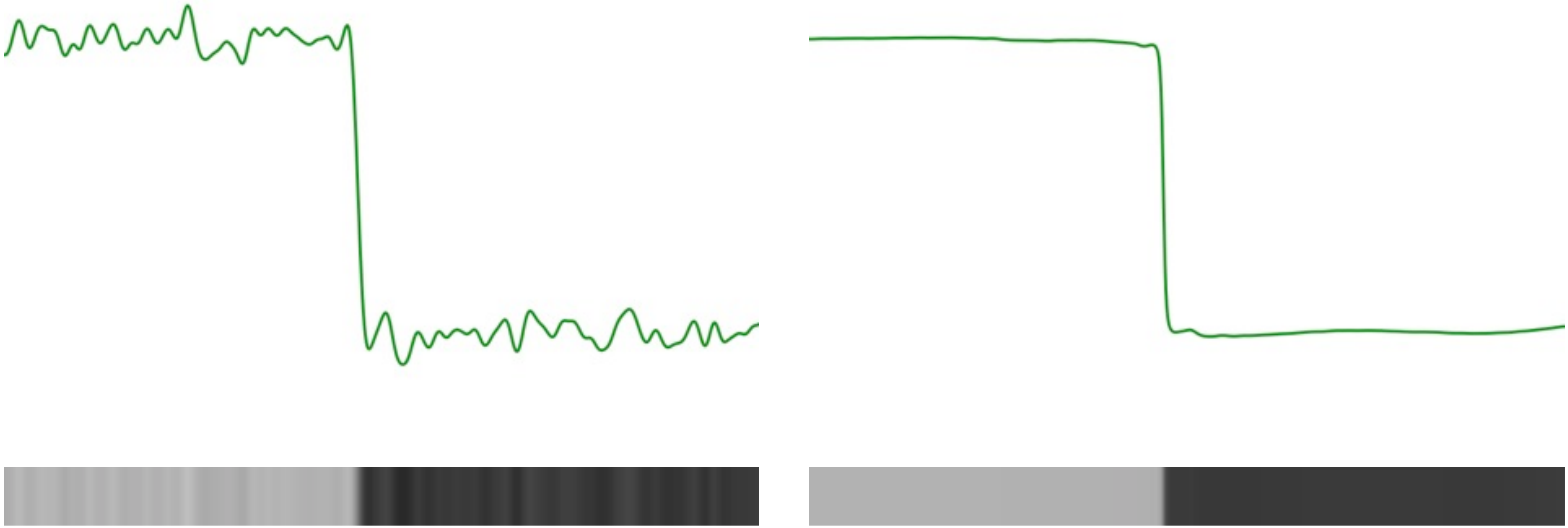
Splat -> Blur -> Slice

- Sample the space at positions p_i



The Result

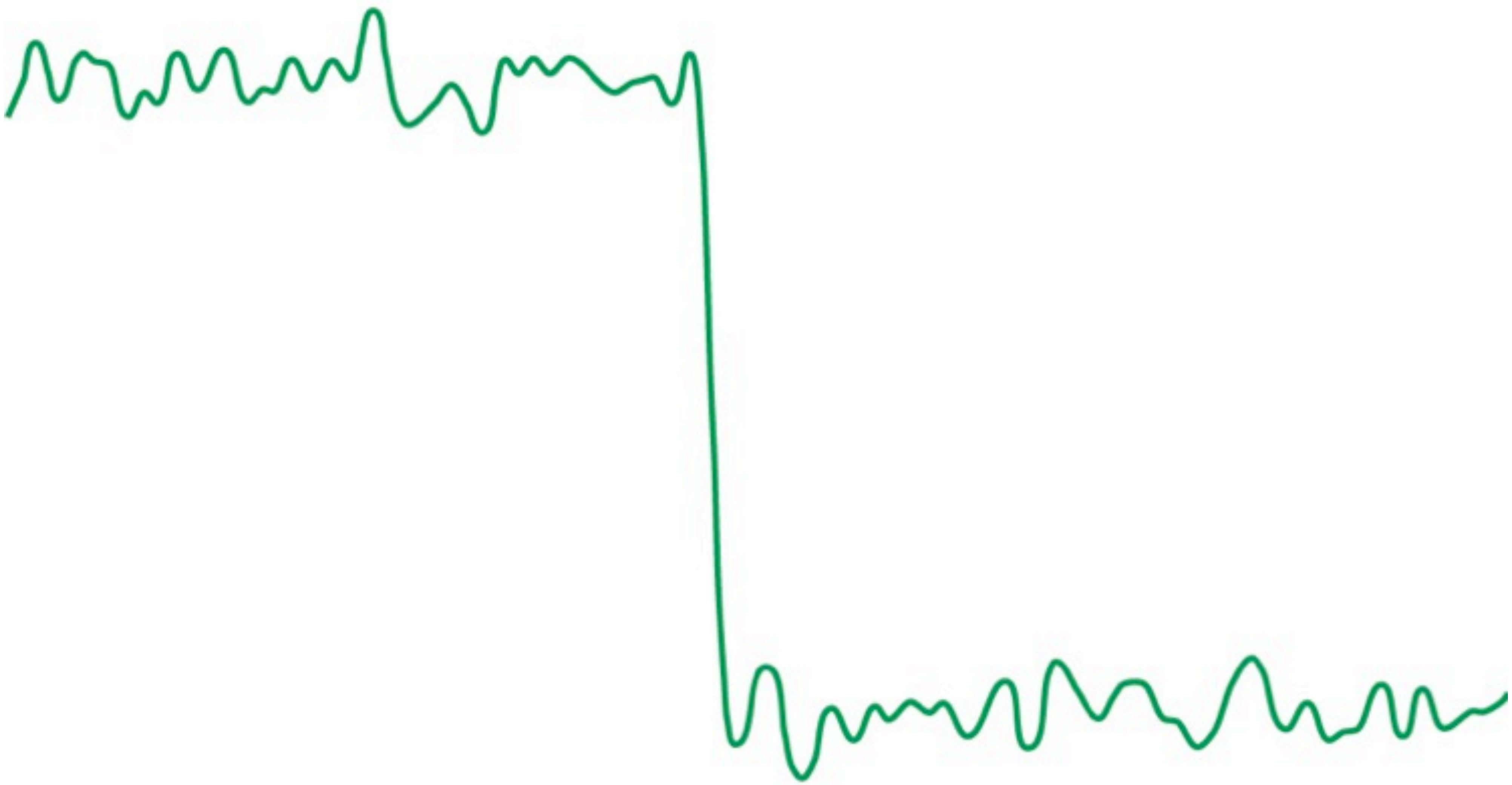
- We’ve smoothed the data without losing the edge



Input

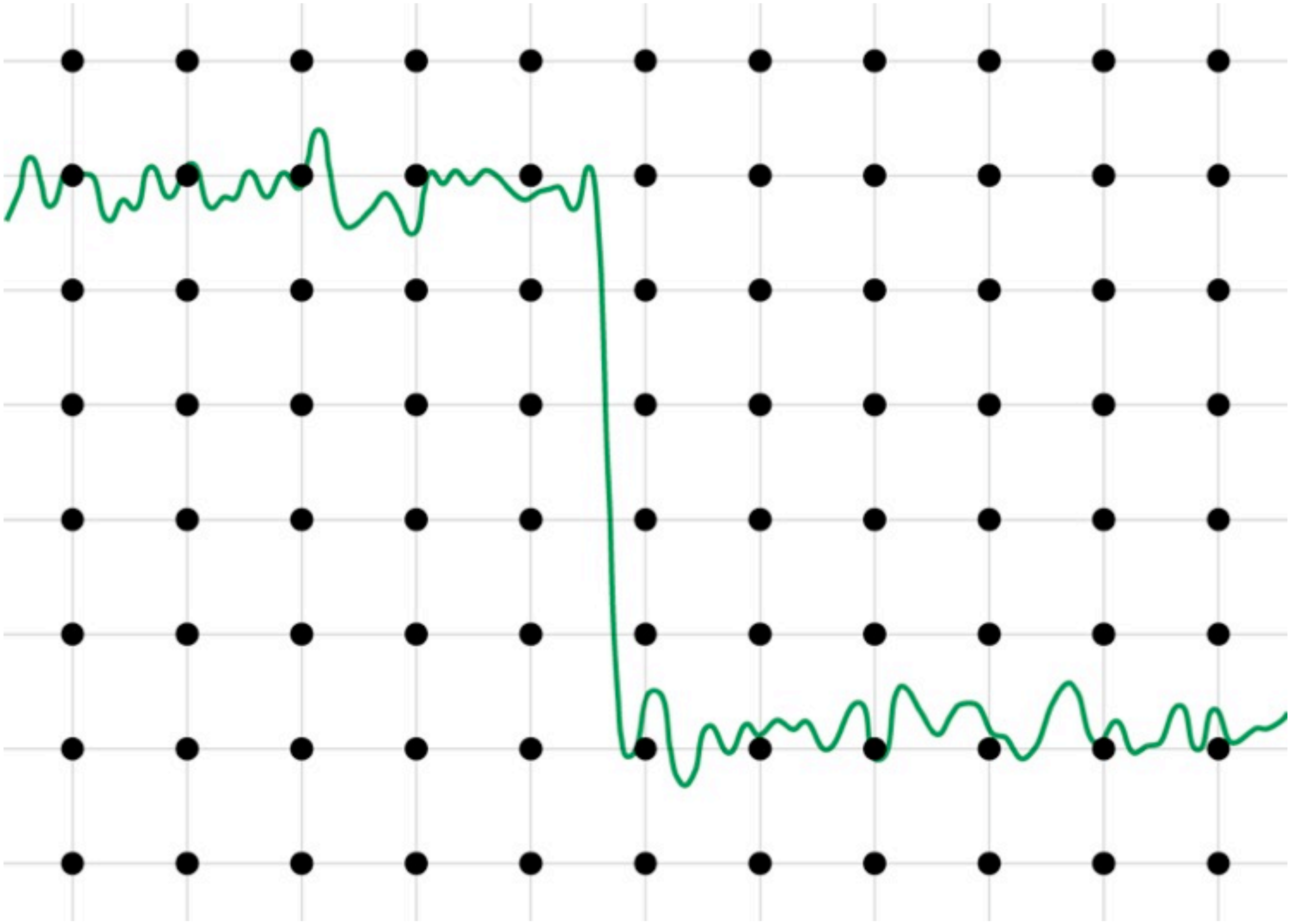
Output

How do we represent the space?

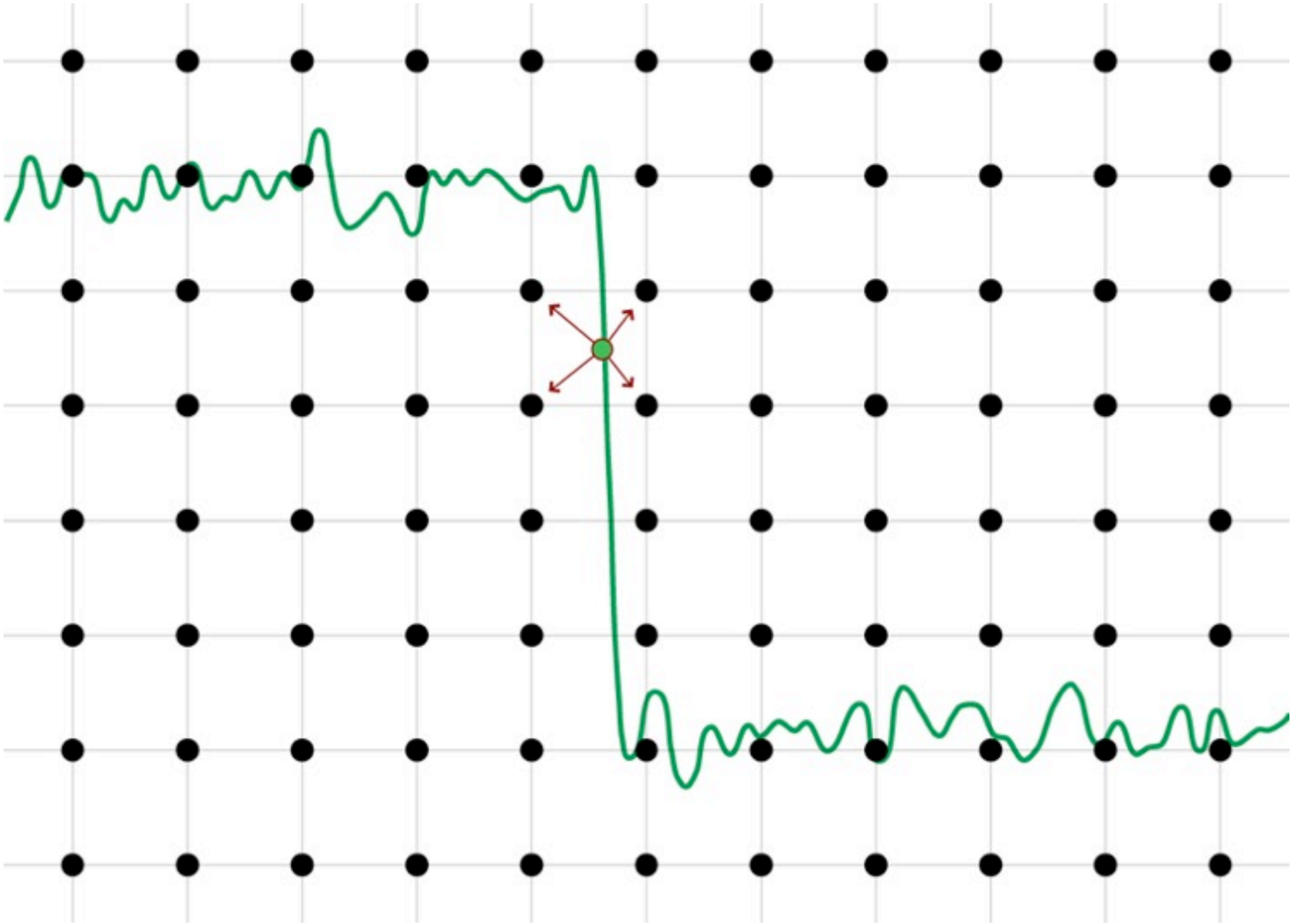


With a grid (**Acceleration #3**)

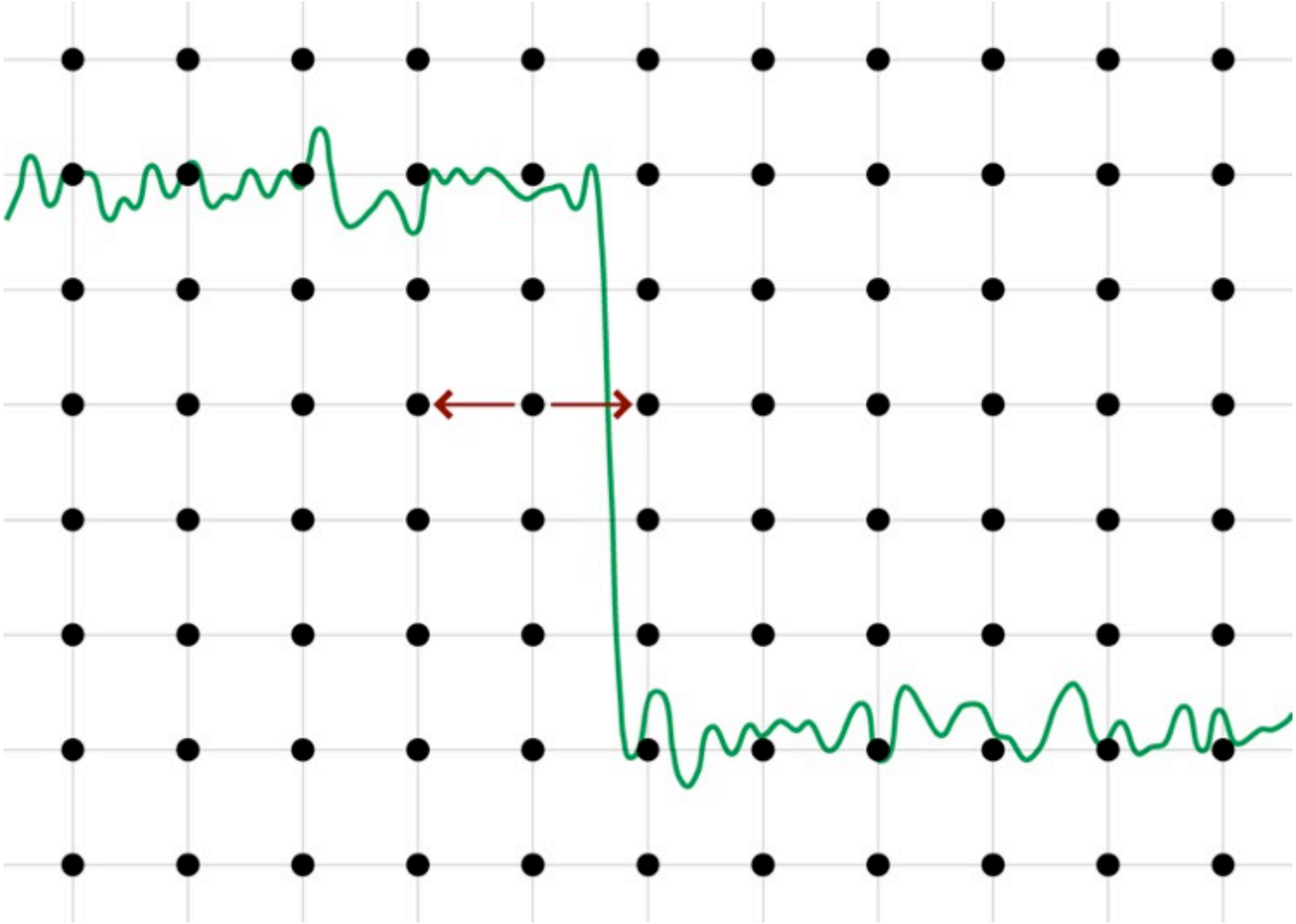
[Paris and Durand, 2006]



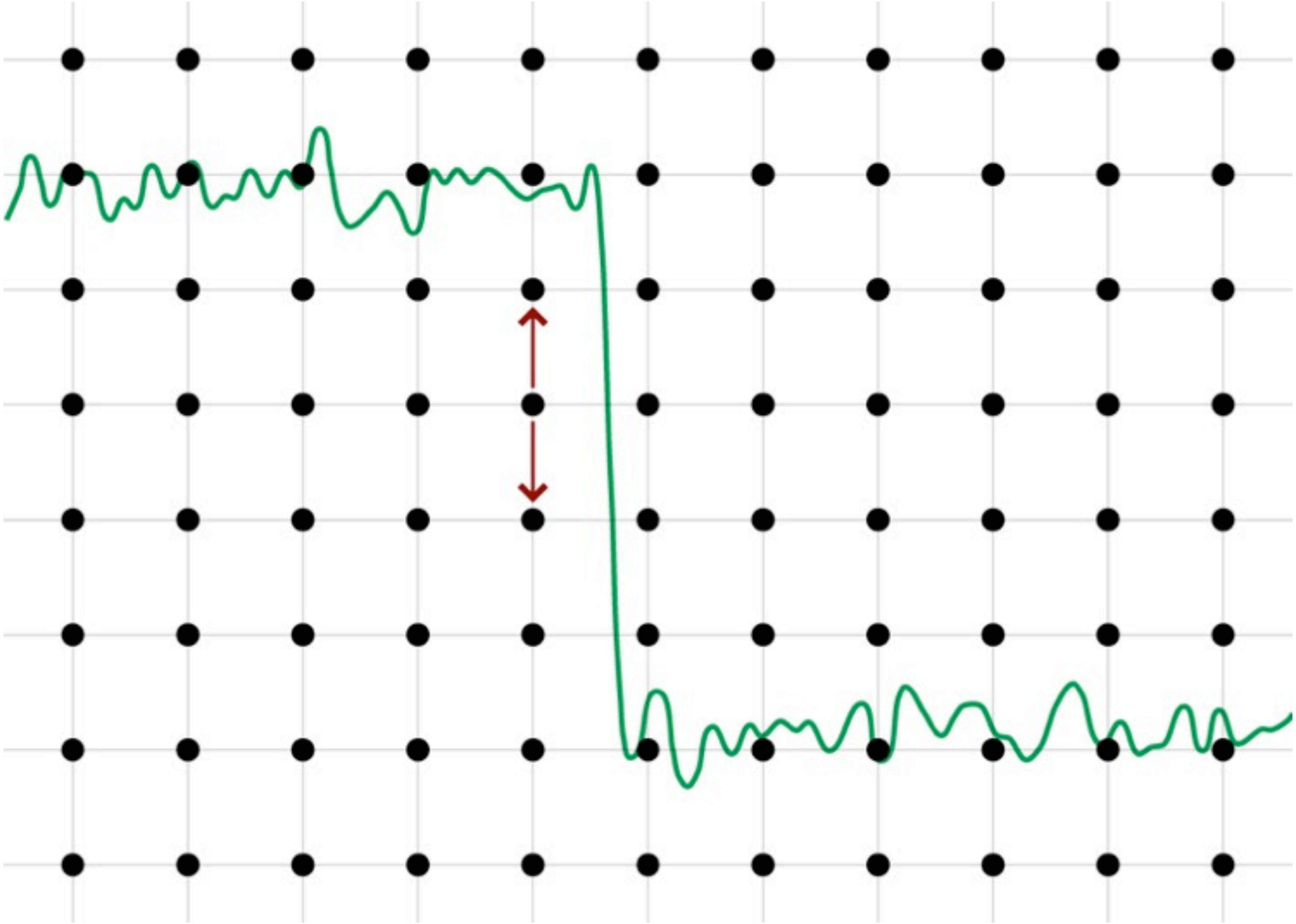
With a grid: Splat



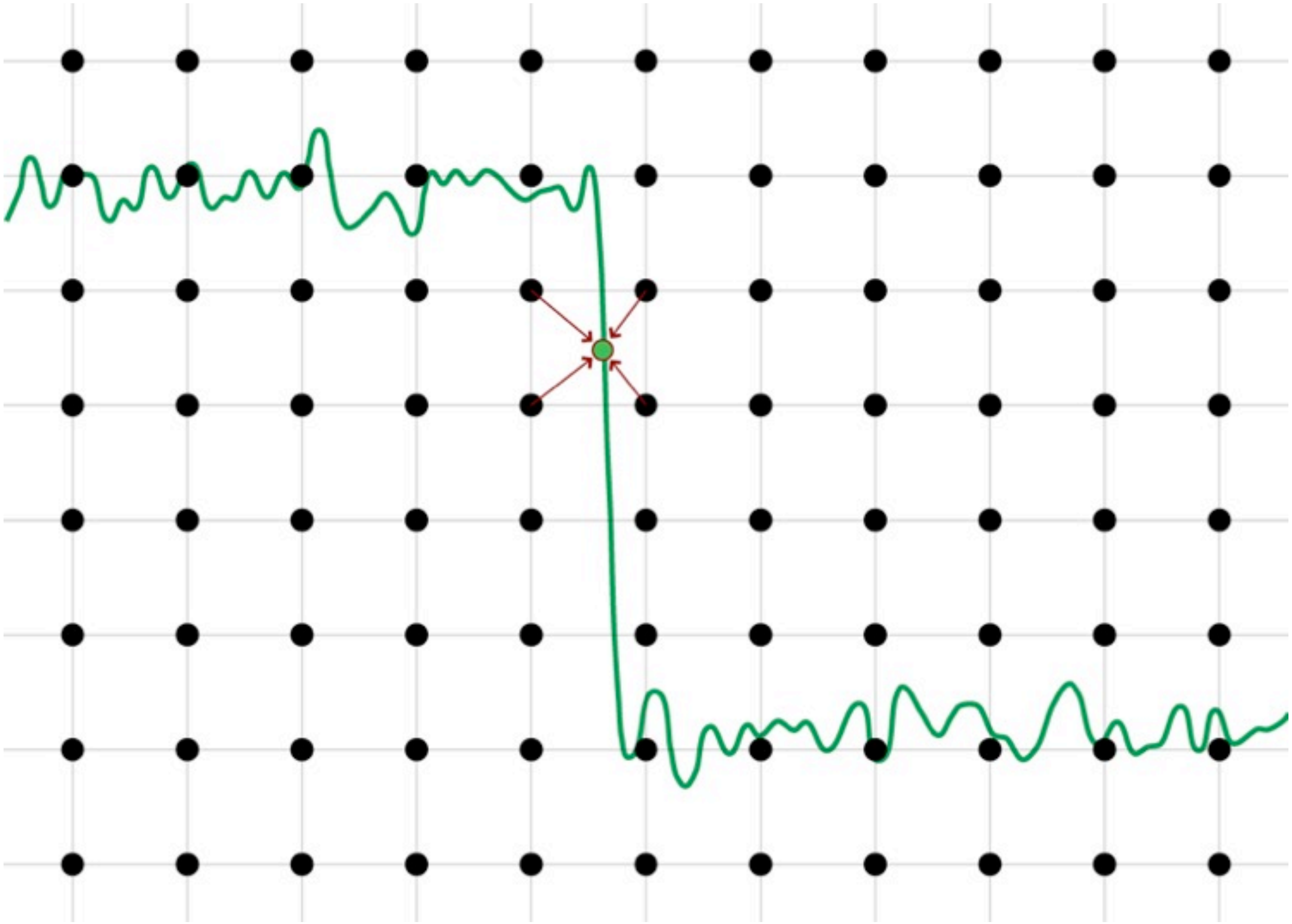
With a grid: Blur



With a grid: Blur

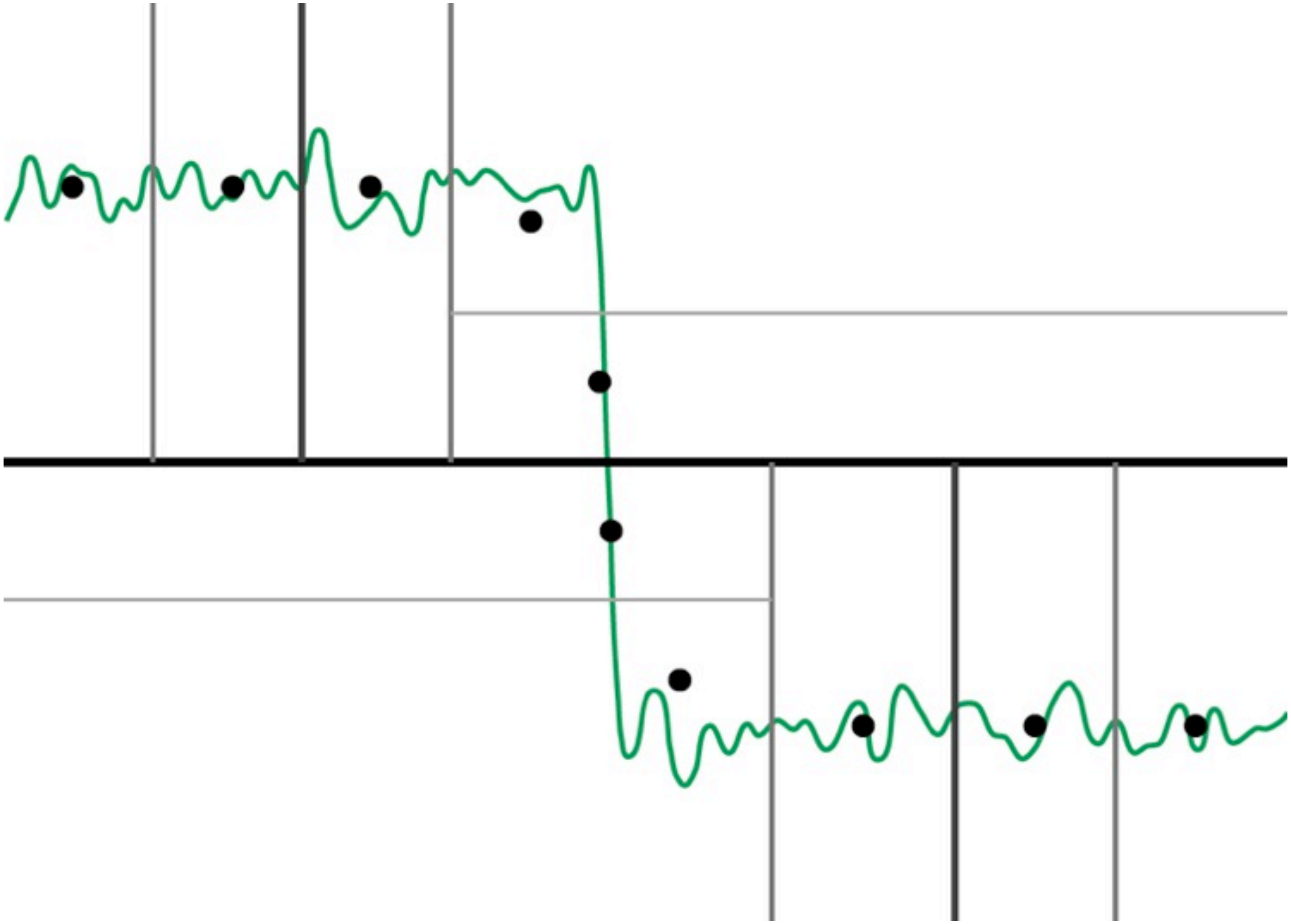


With a grid: Slice



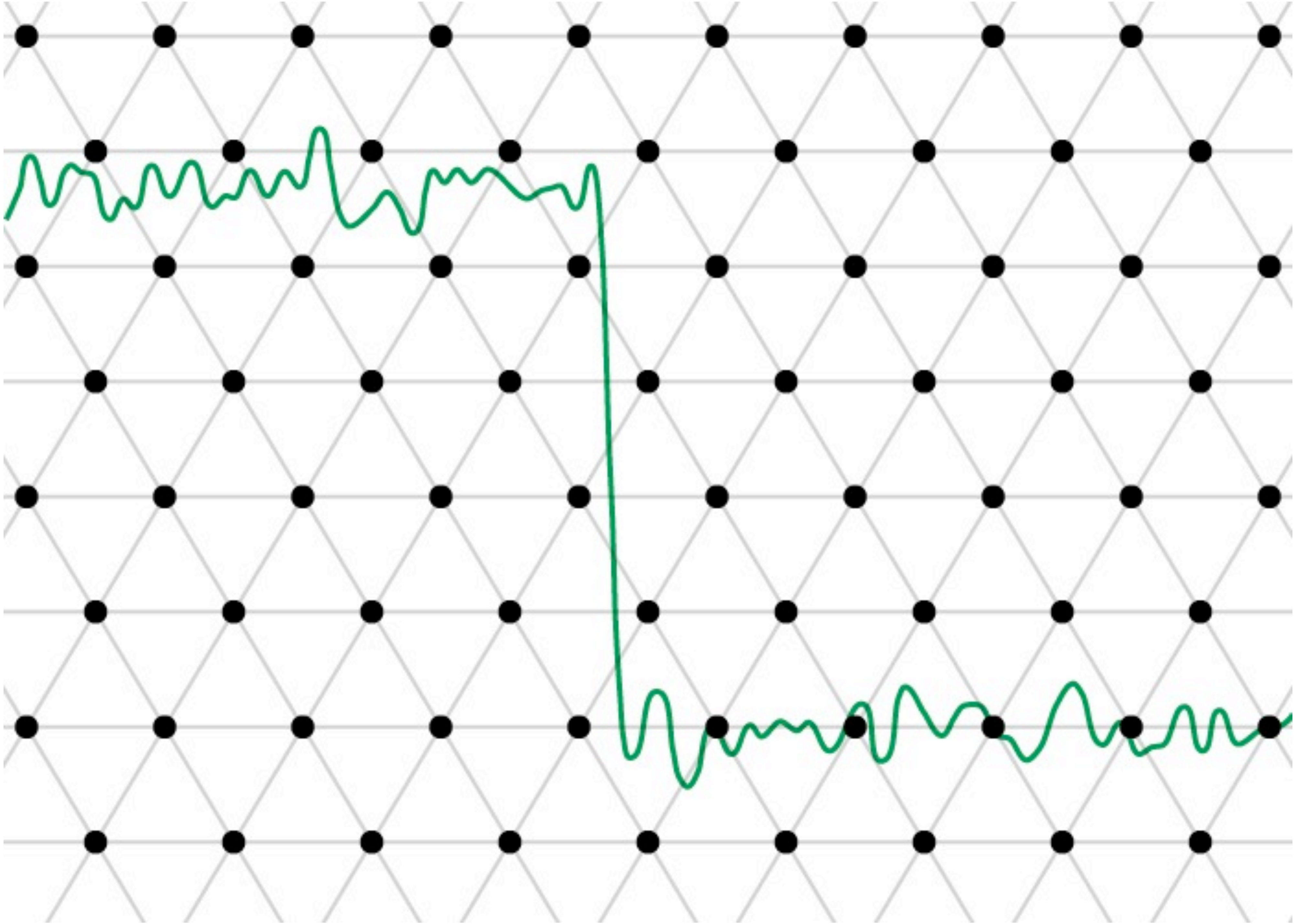
With a kd-tree (**Acceleration #4**)

[Adams, Gelfand, Dolson, Levoy, SIGGRAPH 2009]

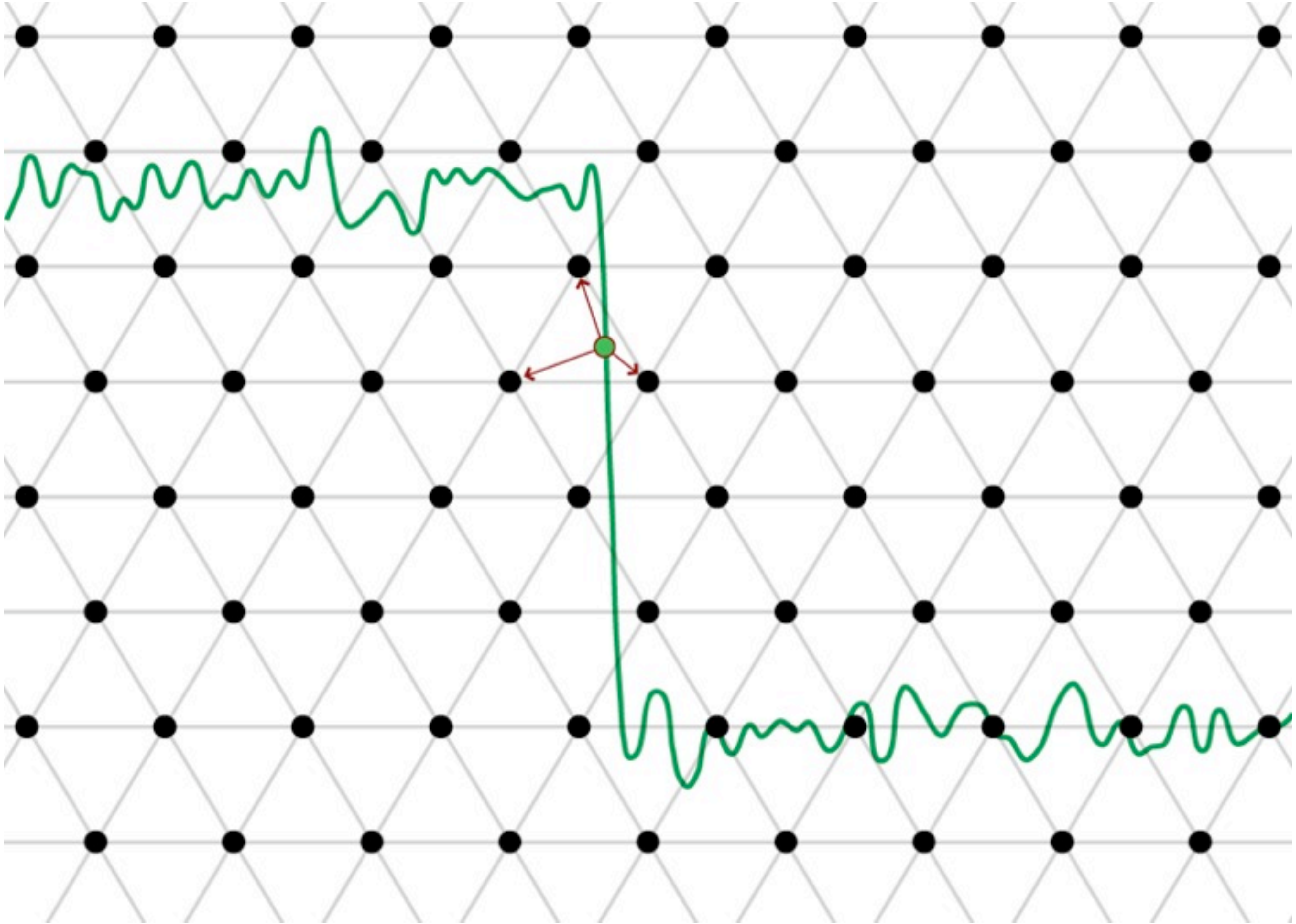


With a lattice (**Acceleration #5**)

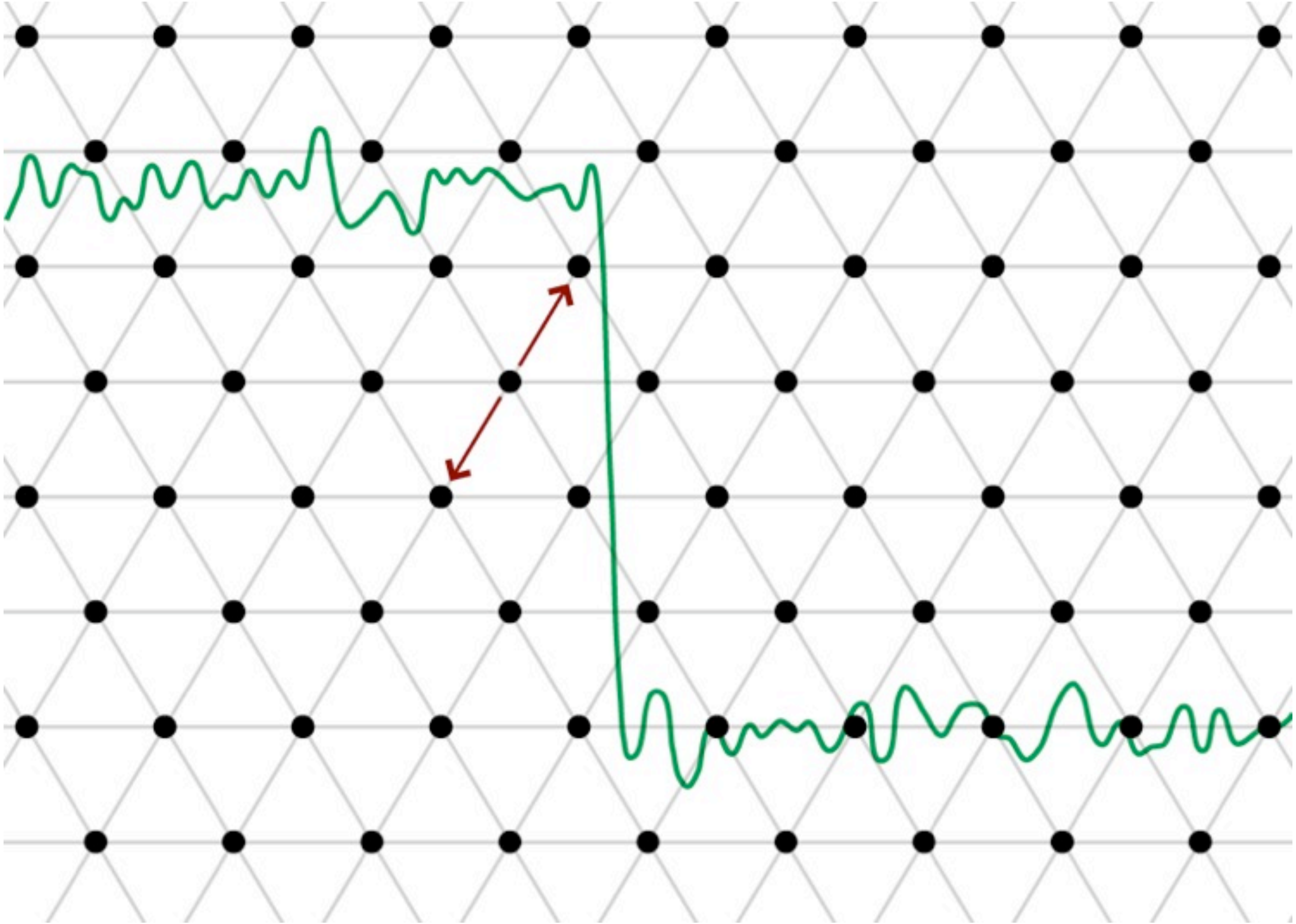
[Adams, Baek, Davis, EUROGRAPHICS 2010]



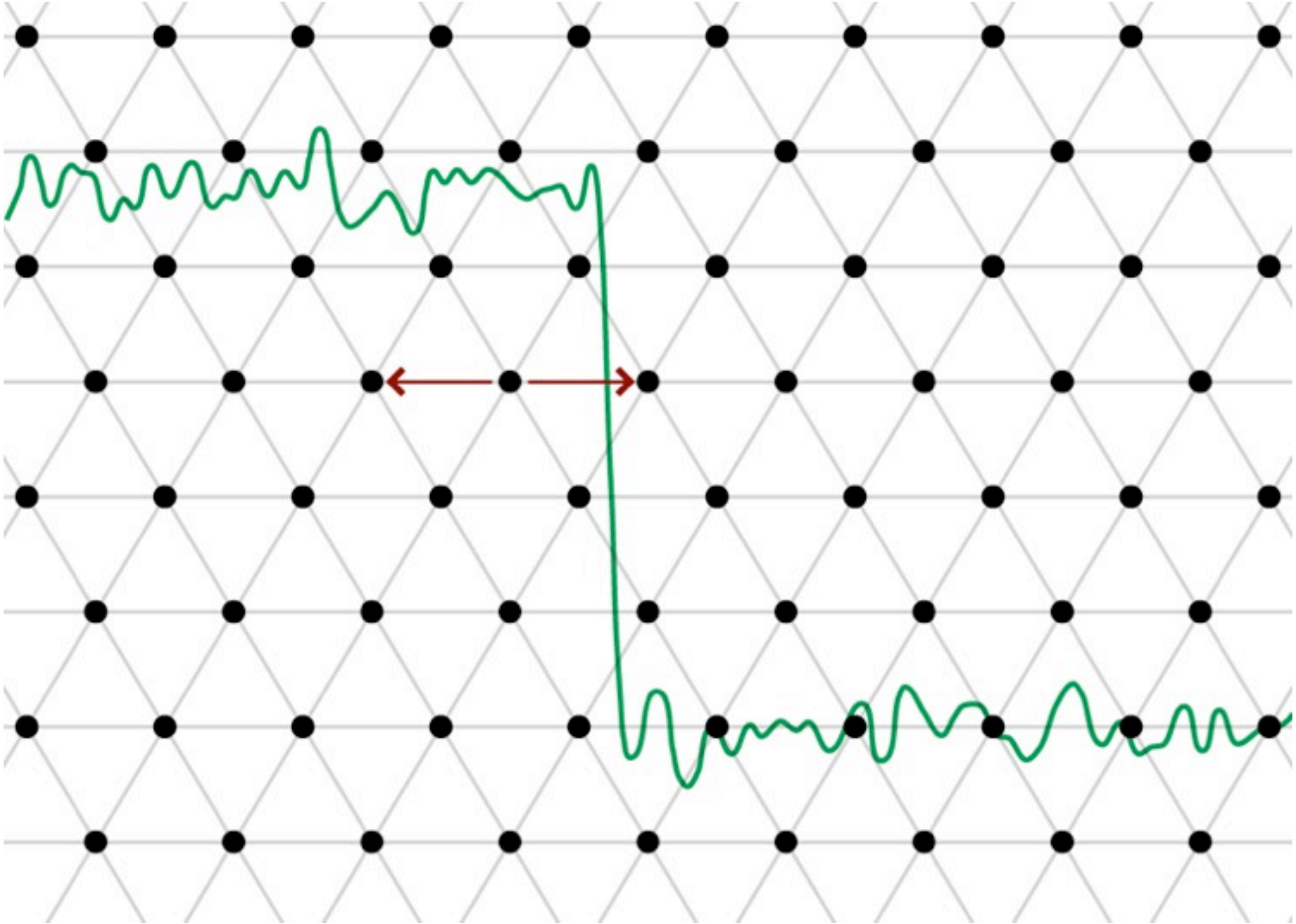
With a lattice: Splat



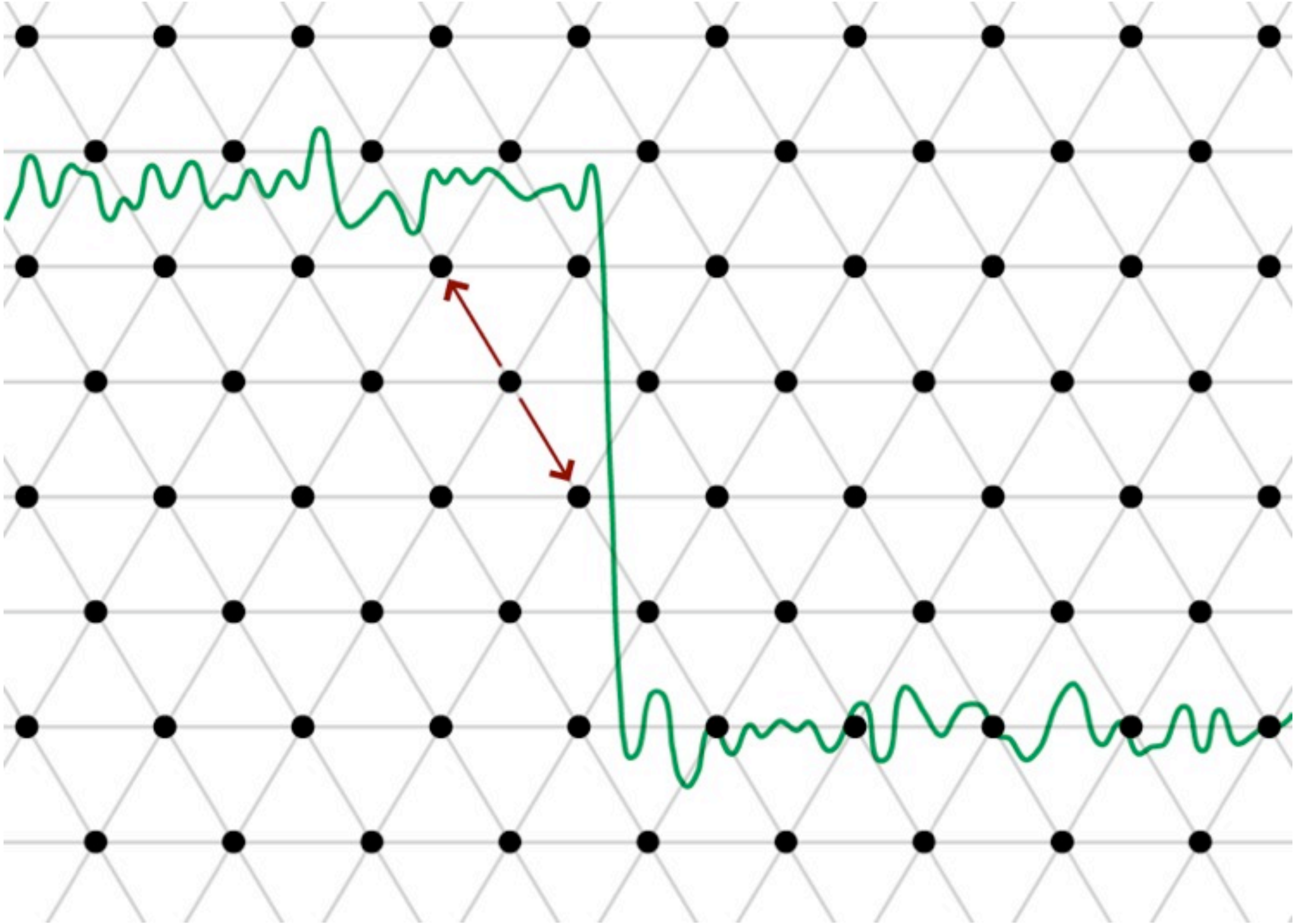
With a lattice: Blur



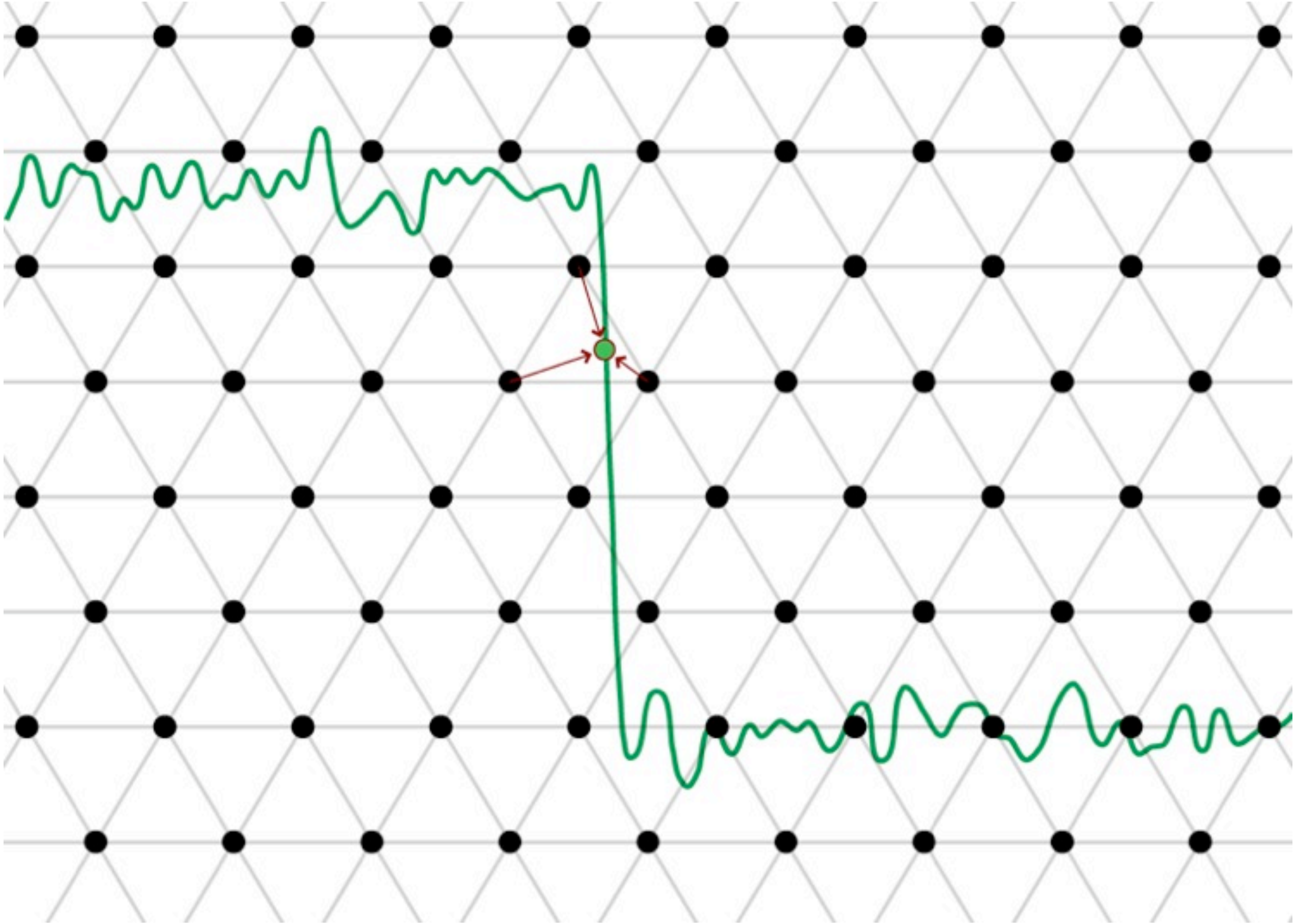
With a lattice: Blur



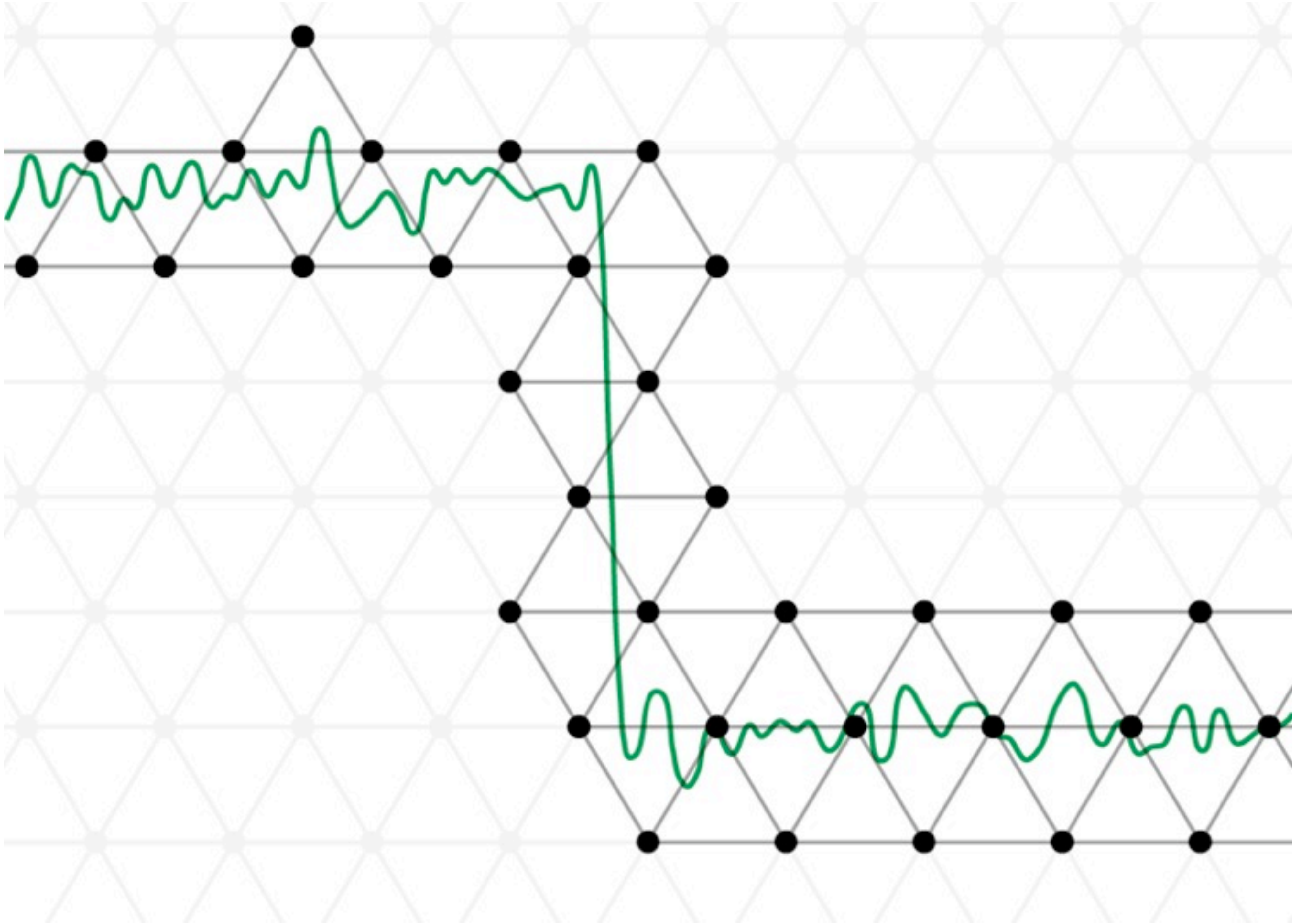
With a lattice: Blur



With a lattice: Slice

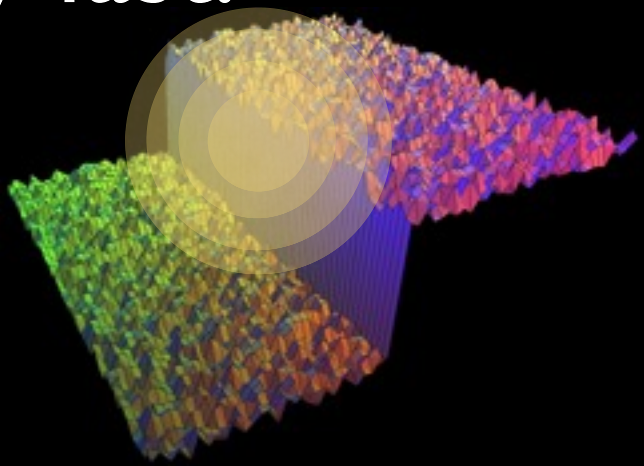


With a lattice



Recap

- Take a bilateral filter problem.
- Rewrite as a high-dimensional signal.
- Put it into a data structure.
- Perform a Gaussian blur really fast.
- Read out its values.



Comparisons

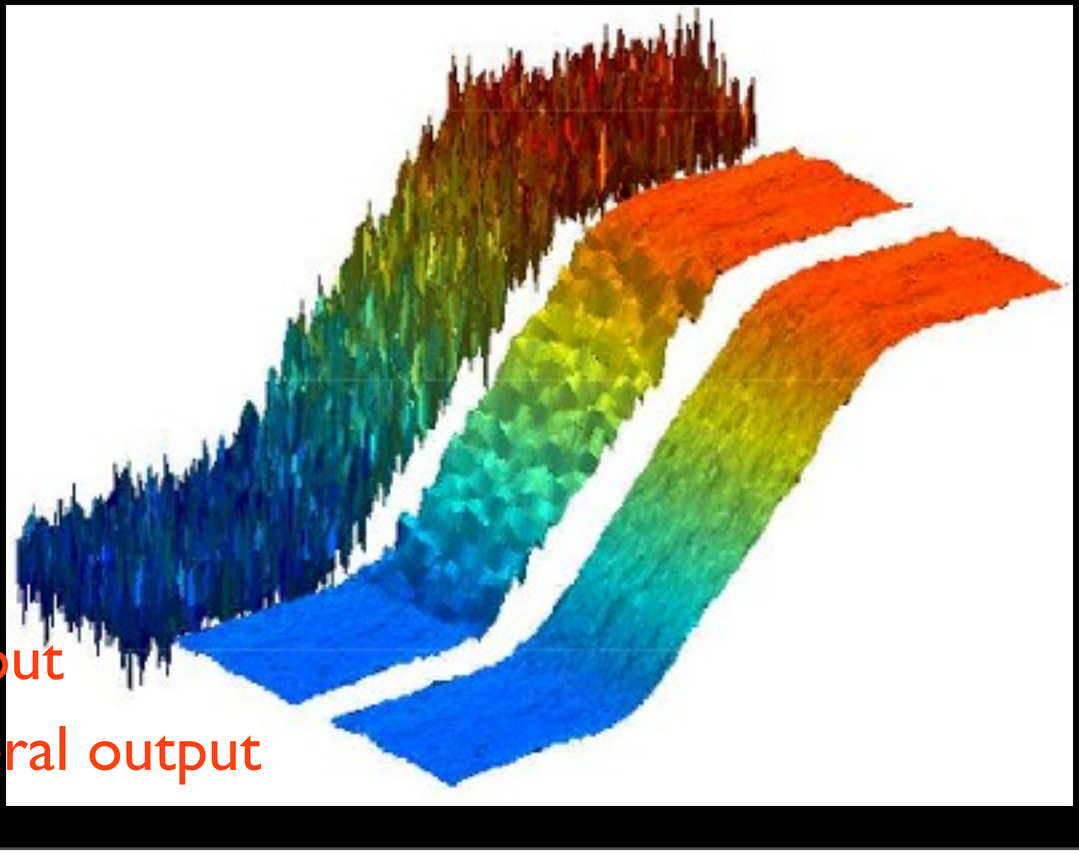
Method	Runtime	d>1?	Can handle sparse data?	Joint Bilateral Filter?
Porikli ’08	$O(N \log N)$	No	No	No
Dorsey ’02	$O(N \log N)$	No	No	No
Grid	$O(2^d N)$	Yes	Poorly	Yes
KD-tree	$O(d N \log N)$	Yes	Yes	Yes
Lattice	$O(d^2 N)$	Yes	Yes	Yes

Other Filters

- TONS of other edge-aware filters
  - A paper or two at every SIGGRAPH

Trilateral Filter

- Bilateral filter penalizes deviation from pixel value
  - e.g. $p(y)\ f(p(y) - p(x))$
- Penalize deviation from the tangent at $p(x)$
  - e.g. $(p(y) - \partial p(x)(y-x))\ f(p(y) - p(x) - \partial p(x)(y-x))$
- Intuition:
  - Bilateral = piecewise flat
  - Trilateral = piecewise linear
- Theoretically better, but slower.



Noisy input

Bilateral output

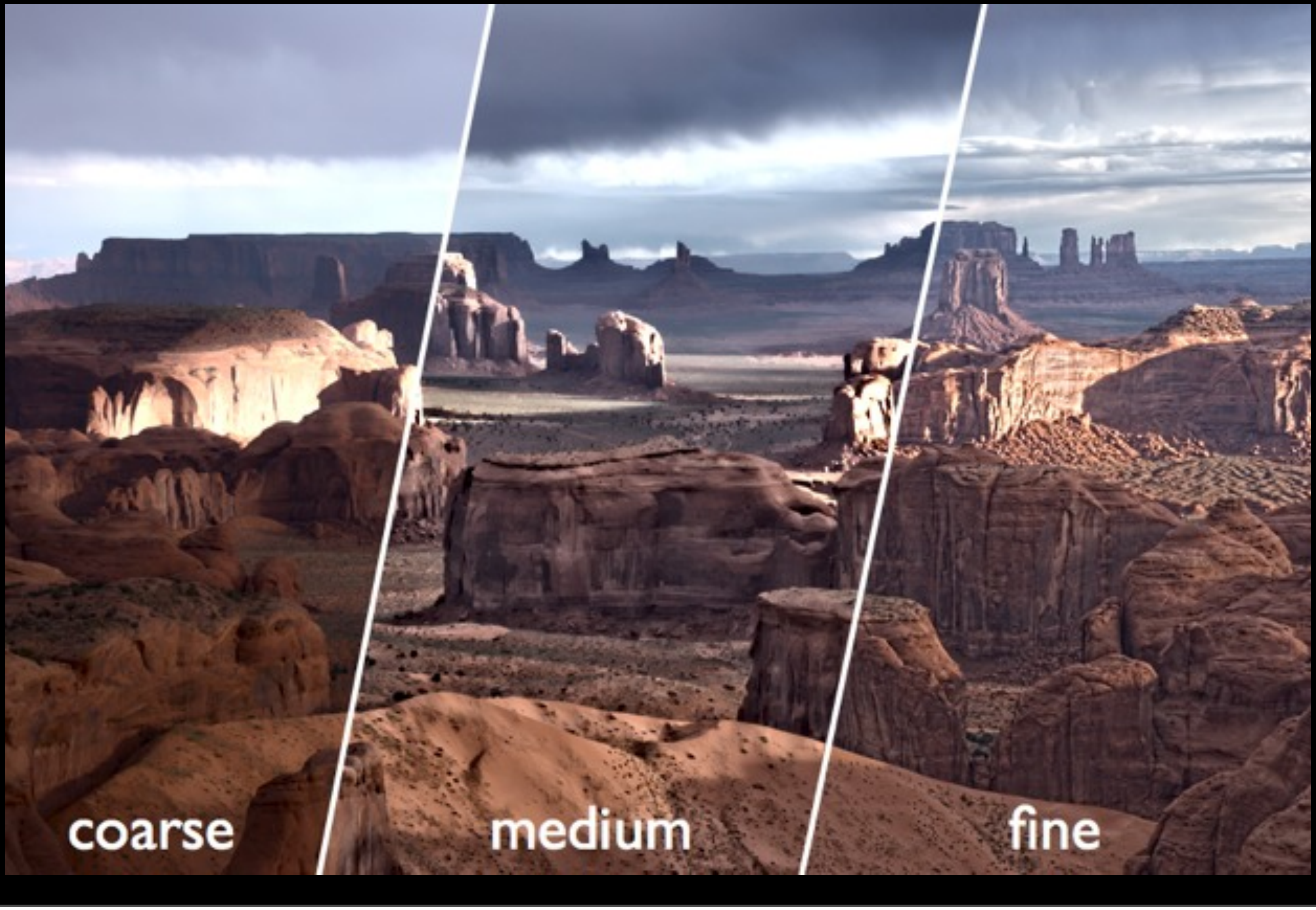
Trilateral output

Weighted Least-Squares Filter

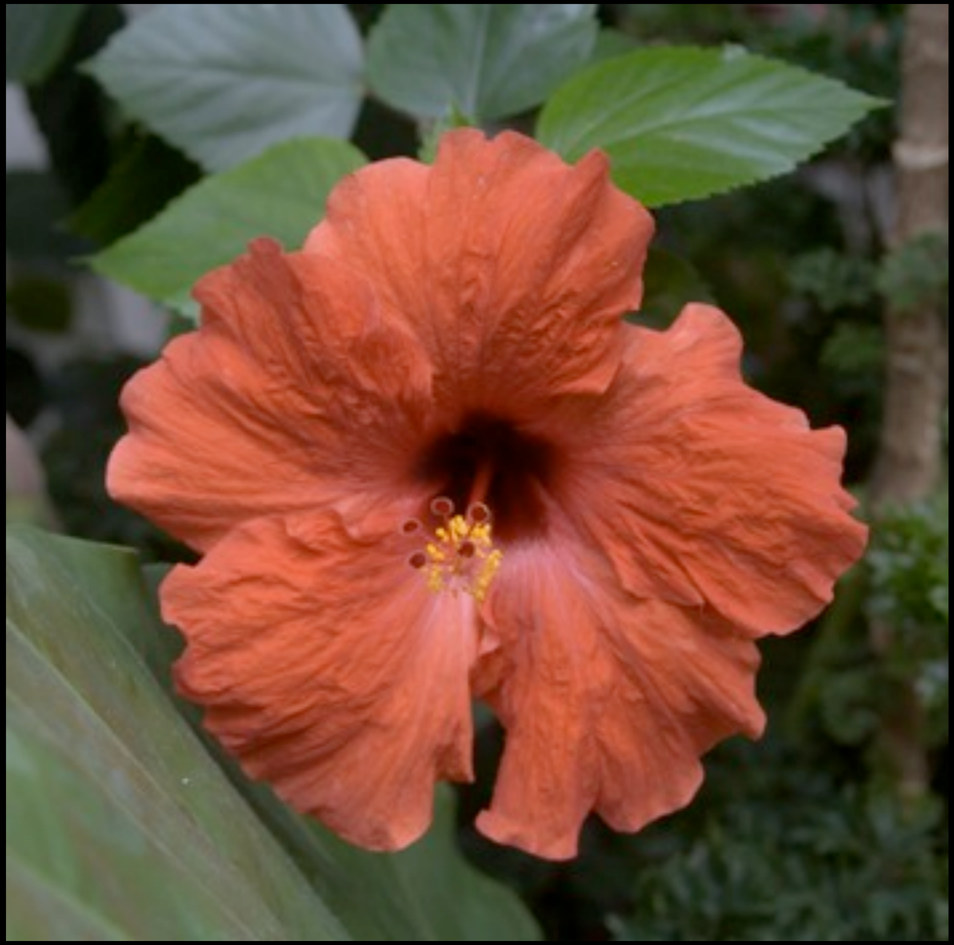
- Express smoothing as an optimization
  - Given image $v(x)$, find $v'(x)$ that minimizes:
    - $\underbrace{\lambda_1 \sum_x [v'(x) - v(x)]^2}_{\text{data term}} + \underbrace{\lambda_2 \sum_x w_x [\partial v' / \partial x(x)]^2}_{\text{smoothness term}}$
    - v' should be similar to input, but should not have high gradients where v does not.

Weighted Least-Squares Filter

- By choosing w_x wisely, one can selectively suppress edges at different scale. (Similar to σ_f, σ_g in bilateral)



Aside: Image Pyramid



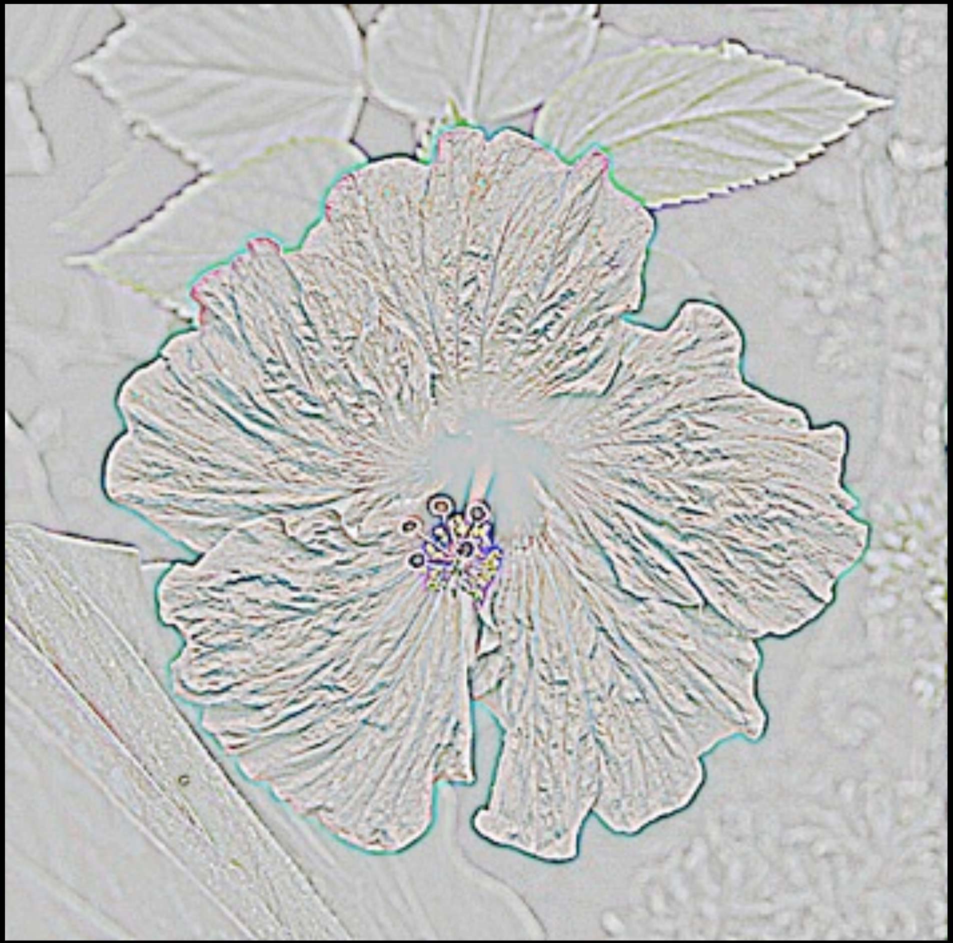
level 0

level 1

level 2

level 3
(residual)

Aside: Image Pyramid



level 0

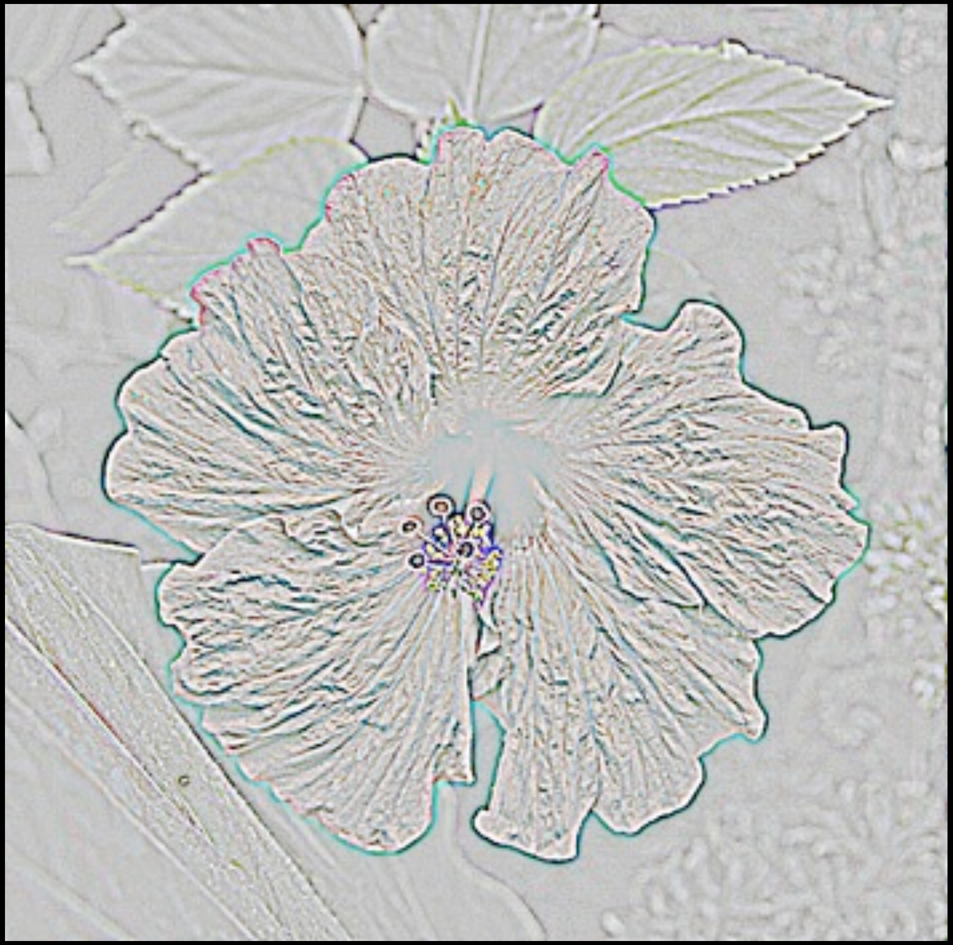
level 1

level 2

level 3
(residual)

Each level contains certain frequency details.

Aside: Image Pyramid



level 0

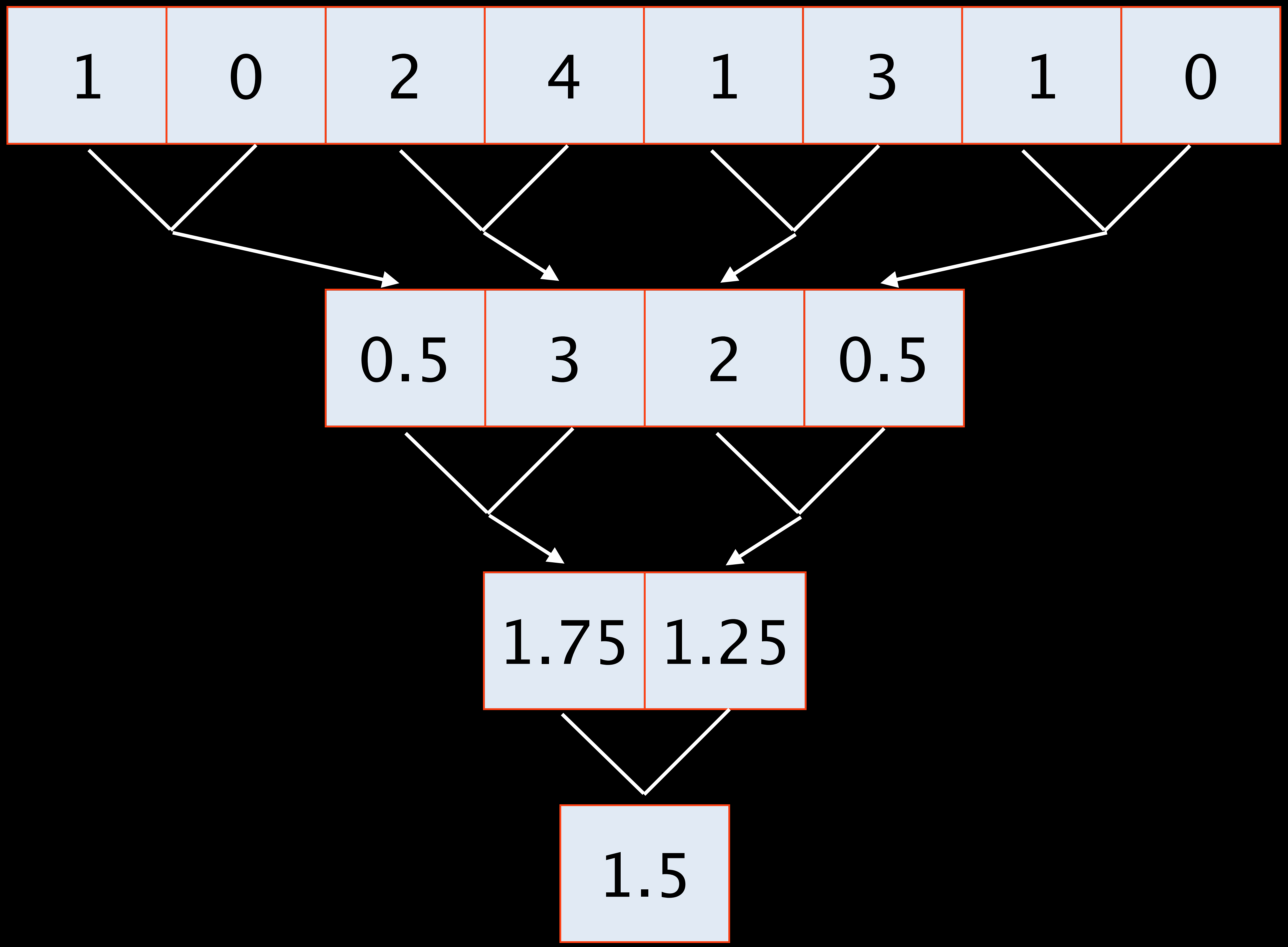
level 1

level 2

level 3
(residual)

Question: How to downsample / upsample?

Wavelet Transform



1	0	2	4	1	3	1	0

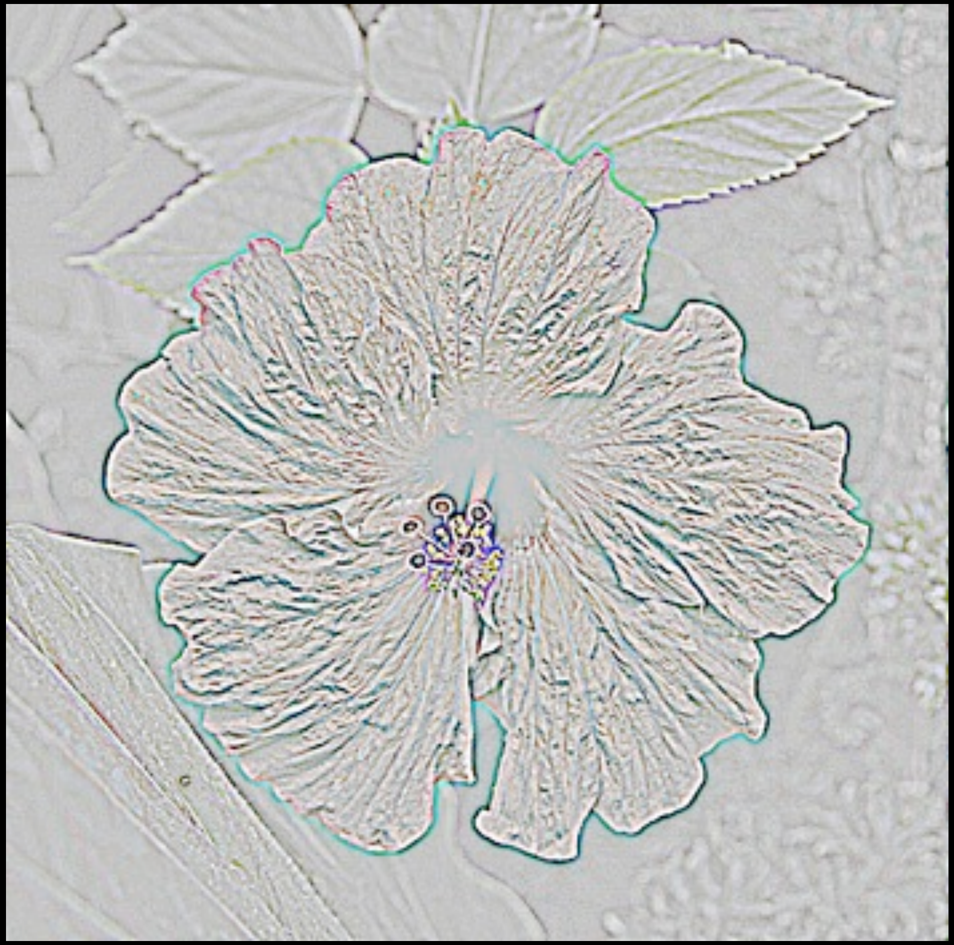
0.5	3	2	0.5

1.75	1.25

1.5

Average pairs. Duplicate to upsample

Laplacian Pyramid



level 0

level 1

level 2

level 3
(residual)

To downsample, Gaussian blur and subsample.
To upsample, insert zeros and blur.

Image Pyramid for Detail Magnification

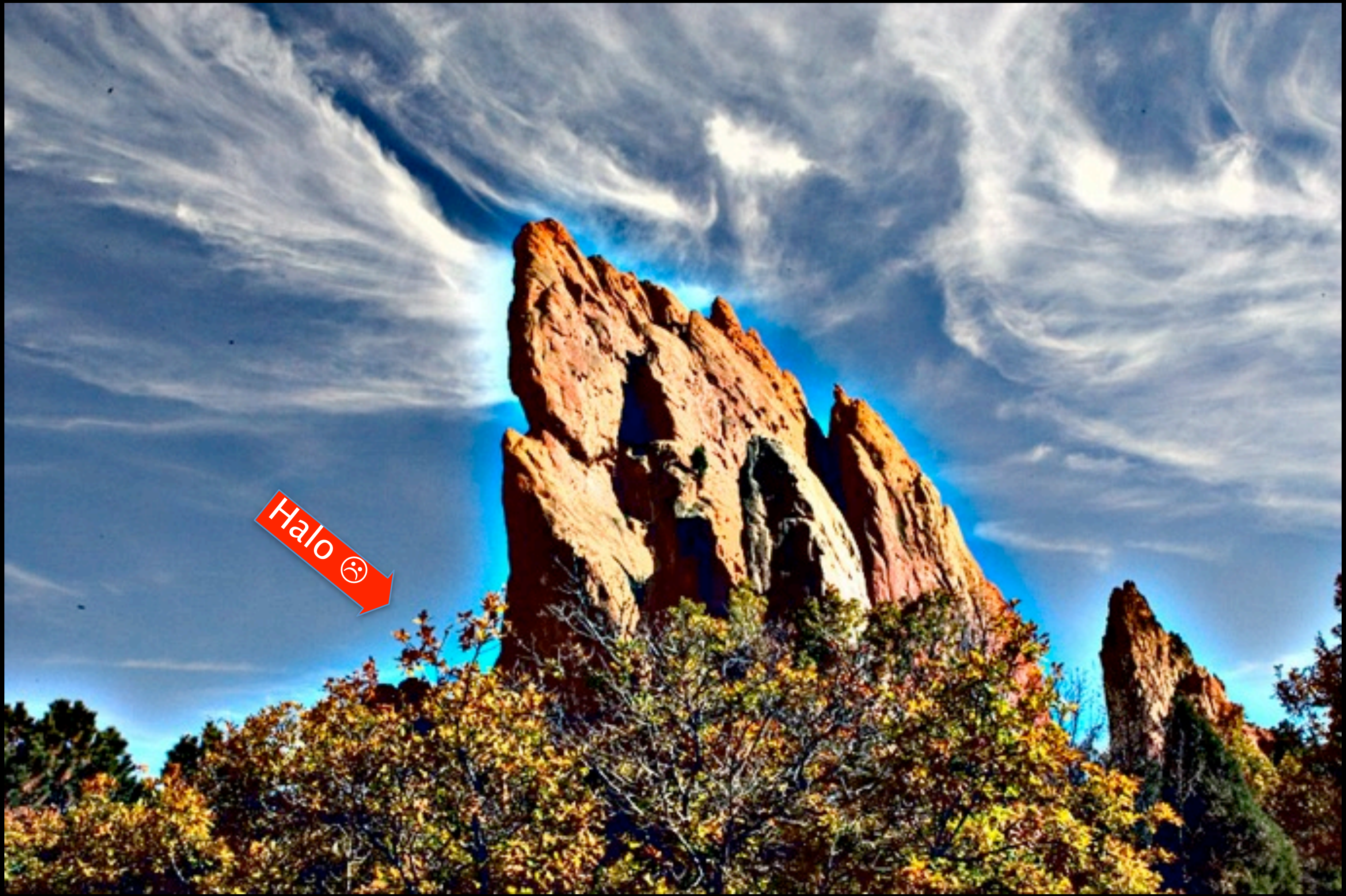


Image Pyramid for Detail Magnification

- Unsuitable for filtering?
  - Details of different "scale" or "frequency" are not nicely separated into different levels.
    - Almost, but not quite.

Edge-Avoiding Wavelets

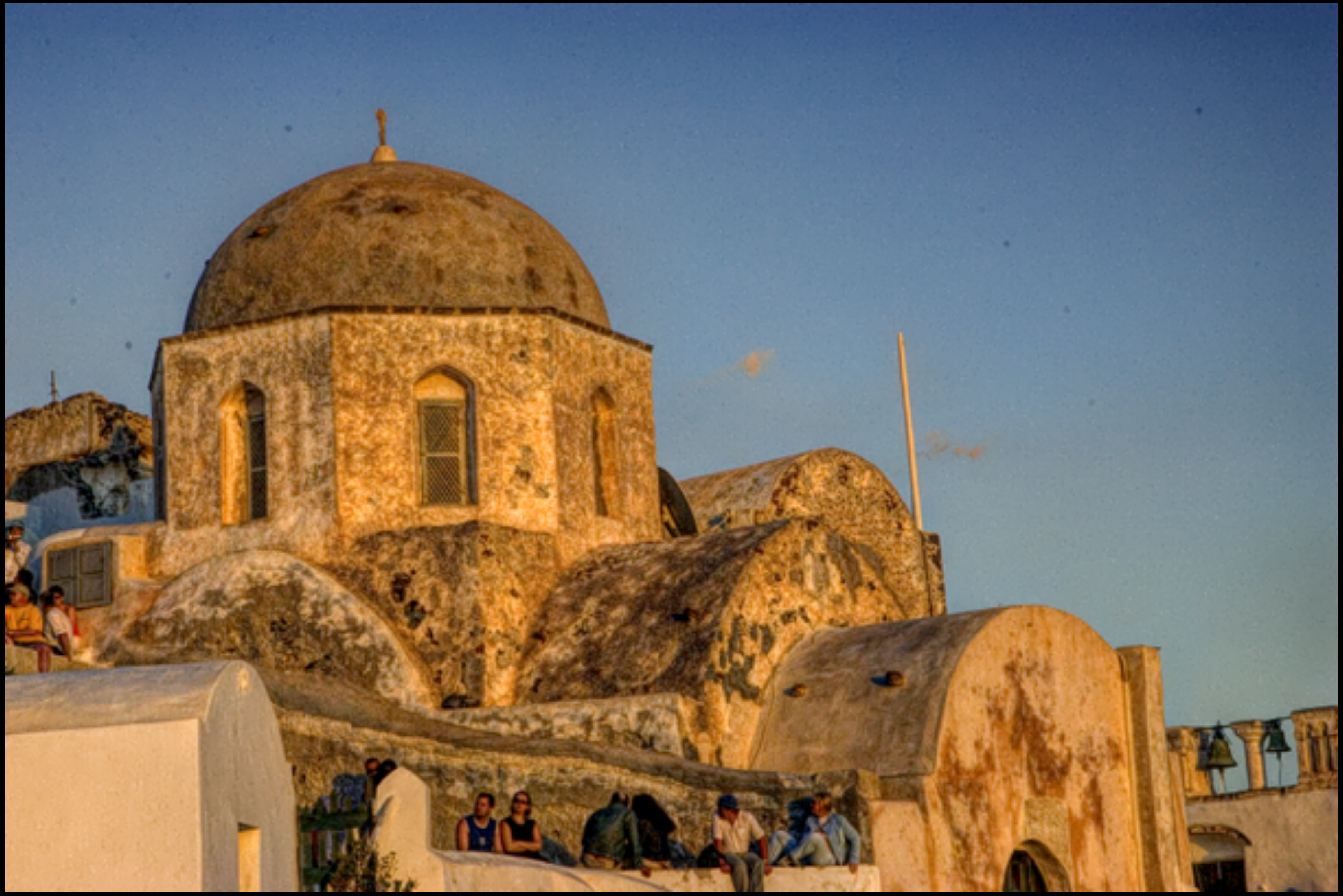
- Modify wavelet transform.
  - Instead of using the simple coefficients, make the coefficients depend on edge strength.

Edge-Avoiding Wavelets

Local Laplacian Filter

- Use regular laplacian pyramid.
- Generate a new laplacian pyramid by filtering the coefficients in a clever way.

Local Laplacian Filter



Summary

- Edge-aware image processing is a popular topic.
  - Bilateral filter
    - Many acceleration schemes
    - Many generalizations
    - Many applications
  - Other filters.