3D Sensing and Sensor Fusion
`http://cg.elte.hu/~sensing`

Dmitry Chetverikov, Iván Eichhardt
csetverikov@sztaki.hu, eiiraai@inf.elte.hu

Eötvös Loránd University
Faculty of Informatics

2021 fall (2021-2022-1)

Introduction

Conventional filtering
- Convolution
- Linear smoothing filters
- Median filter

Multilateral filters
- Bilateral filters
- Joint filtering

Upsampling
- Upsampling using guided filtering

Convolution: Continuous formulation

Given two functions f, w their **convolution** is can be defined as

$$(f * w)(x) \doteq \int_{-\infty}^{\infty} f(x-\tau)w(\tau)d\tau.$$

The convolution defines a third function expressing how the shape of one is modified by the other.

Note that

- in practice, we have discretely sampled data: $\int \Rightarrow \sum$
- use the continuous repr. when *e.g.* computing derivatives!

Convolution: Discrete image space

Linear combination of input pixels: **convolution** of image f with mask w

$$g(x,y) = (f * w)(x,y) \doteq \sum_{\substack{(x',y')\in W \\ (x-x',y-y')\in F}} f(x-x',y-y') \cdot w(x',y')$$

- W is set of positions in window, F in image
- w is often called a **kernel** or a **mask** of weights

Basic properties of convolution

1. Commutative: $w * v = v * w$ (order is arbitrary)
2. Associative: $(f * w) * v = f * (w * v)$
3. Distributive: $(f + g) * w = f * w + g * w$
4. Homogeneous: $(\alpha f) * w = \alpha (f * w)$ for any constant α

- f and g are images, w and v masks
- $w * v$: mask w is treated as image and convolved with v
  - result is a larger mask
  - associativity can be used to speed up filtering

Types of noise

- **Additive picture-independent** (white) noise:
$$g(x,y) = f(x,y) + v(x,y)$$
  - $f(x,y)$ is input, $g(x,y)$ output image, $v(x,y)$ noise
  - typical channel (transmission) noise
- **Uncorrelated multiplicative noise**:
$$g(x,y) = f(x,y) \cdot v(x,y)$$
  - amplitude modulation (variation)
  - typical for TV raster lines
- **Quantisation noise** (error):
$$v_{noise}(x,y) = g_{quantised}(x,y) - f_{original}(x,y)$$
- **Salt-and-pepper, or peak noise**: Pointwise, uncorrelated random noise

Mean filter

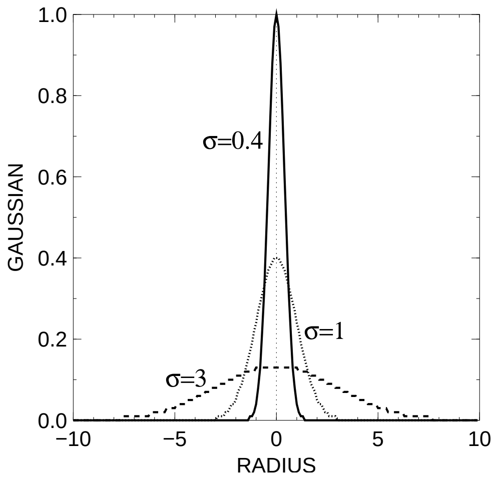
- Spatial averaging (smoothing) filter
- Non-negative weights that sum to 1
$$0 \leq w_{mean}(x,y) \leq 1, \quad \sum_{x,y} w_{mean}(x,y) = 1$$
  - in practice, use integer weights, then normalise
- Weights do not grow with distance from filter center:
$$w_{mean}(x_1,y_1) \leq w_{mean}(x_2,y_2), \quad \text{if } x_1^2 + y_1^2 > x_2^2 + y_2^2$$
- Box filter: mean filter with unit weights

Gaussian filter 1/2

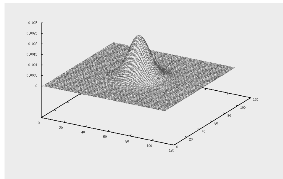
$$w_G(x,y) = \frac{1}{\sum\limits_{(x,y)\in W} e^{-\frac{r^2(x,y)}{2\sigma^2}}} e^{-\frac{r^2(x,y)}{2\sigma^2}}$$

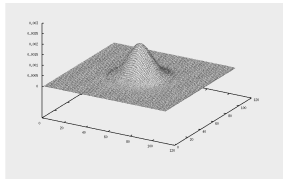
- Weights provided by 2D Gaussian (normal) distribution function.
- $r^2(x,y) = x^2 + y^2$ is squared distance from mask center
  - does not depenge on angle, on r only
  - bell-like, rotation-symmetric shape
- Parameter σ controls size of filter
  - larger $\sigma \Rightarrow$ larger filter and stronger smoothing

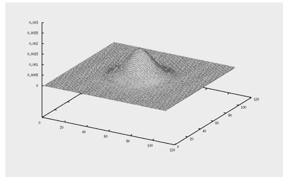
Shape of Gaussian filter for growing σ: 2D



Shape of Gaussian filter for growing σ: 3D


$\sigma = 9$


$\sigma = 10$


$\sigma = 11$

Gaussian filter 2/2

$$w_G(x,y) = \frac{1}{\sum\limits_{(x,y)\in W} e^{-\frac{x^2+y^2}{2\sigma^2}}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- When discretised, $w_G(r)$ is cut at $r_{max} = k\sigma$.
  - typically, $k = 2.5$
  - includes most of bell volume
- Gaussian filter is separable:

  $$w_G(x,y) = w_G(x) \cdot w_G(y) \Leftarrow \exp(a+b) = \exp(a) \cdot \exp(b)$$

  - fast implementation: two 1D filters instead of one 2D filter
  - $O\left((2r_{max})^2\right)$ ops in 2D, $O(2 \cdot 2r_{max})$ ops in 1D

Use of smoothing

- **Noise filtering**
  - box filter reduces zero-mean white noise as positive and negative values nullify each other
  - large filter size $\Rightarrow$ greater noise reduction
- Removing fine details
- **Subsampling**: going to lower resolution
  - average, then decimate (discard rows/columns)
- Obtaining **scale-space** representation of image
  - sequence of Gaussian-filtered images for growing σ
  - image analysis at varying degree of detail

Nonlinear median filter

- Median filter outputs median of greyvalues in window:
  - sort (rank) the pixels by greyvalue
  - select value which is in centre (middle) of sorted sequence
  - normally, window size is odd: 3×3, 5×5, etc.
- Example:
  - nine greyvalues in 3×3 window are

$$(1, 1, 3, 2, 5, 4, 4, 12, 11)$$

  - the ordered sequence is

$$(1, 1, 2, 3, \mathbf{4}, 4, 5, 11, 12)$$

  - median value is **4**

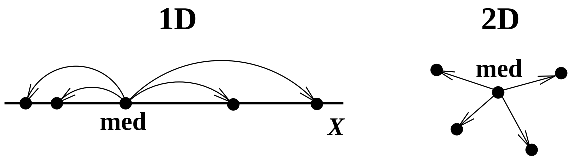
Properties of median 1/2

- Calculating the median is *non-linear* operation: For two sequences P and Q,

  $Med(\alpha P) = \alpha Med(P)$ but $Med(P + Q) \neq Med(P) + Med(Q)$
- Selecting the median can be viewed as *voting procedure*
  - during sorting, each pixels votes for a grayvalue
  - median is selected from majority, from the 'middle'
  - extremal values are rejected as not belonging to majority

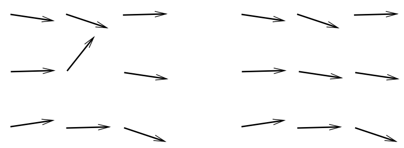
Properties of median 2/2

- Median is a *robust statistic*
  - outliers do not bias result
  - the *breakdown point* is when outliers form 50% or more
- Consider numbers as points on X. Sum of distances from median to other points is minimal for any 1D point set
  - in other words, median is the *innermost* point of set
  - this property is equivalent to definition of median
  - used to extend median to higher dimensions, **vectors**



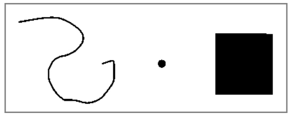
Properties of median filter

- Removes isolated noise pixels
- Does not blur image, but rounds off corners
- Removes thin lines when *filtersize* $> 2 \times$ *linewidth*
  - background pixels form majority
- Number of operations required
  - direct implementation: $O(N \cdot N_W \cdot \log N_W)$
  - run filter implementation: $O(N \cdot \log N_W)$
- Vector median filter enhances vector fields
  - removes vectors incompatible with surrounding vectors

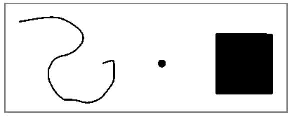


before filtering after filtering

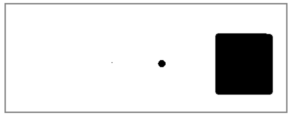
Comparing median and box filters for bilevel image



input image



median 3×3



median 7×7

median 17×17

box 5×5

box 9×9

The underlying optimisation problems

The mean is the solution to the following least-squares problem.

$$\mathrm{mean}(x_1, \ldots, x_n) = \arg\min_y \sum_{i=1}^{n} |y - x_i|^2 = \frac{1}{n} \sum_{i=1}^{n} x_i$$

While the mean and weighted mean (e.g., gaussian) have closed-form form solution, the median does not.

$$\mathrm{median}(x_1, \ldots, x_n) = \arg\min_y \sum_{i=1}^{n} |y - x_i| = ?$$

If given 1D data, finding the middle element(s) is a good approach. In other cases, iterative approaches can approx. the median.

Recap: Gaussian filter

Given the Gaussian filter $w_G(x, y) = w_g(\mathbf{v})$, smoothing the image is defined as

$$\hat{f}(\mathbf{u}) = \frac{1}{k_{\mathbf{u}}} \sum_{\mathbf{v} \in \Omega} f(\mathbf{u} - \mathbf{v}) w_G(\mathbf{v}).$$

This filter blurs everything 'without discrimination'.
To preserve meaningful structure → use Bilateral filter.

Bilateral (B) filter (1/2)

Bilateral filter adds an additional weighting factor over Gaussian:

$$w_B^{\mathbf{u}}(\mathbf{v}) = p\left(\|f(\mathbf{u}) - f(\mathbf{u} - \mathbf{v})\|\right) w_G(\mathbf{v}).$$

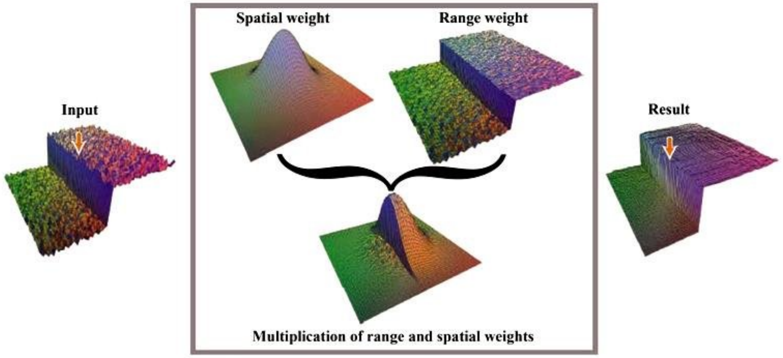
The use of weighting function p:

- dissimilar pixels get lower weights
- preserves strong edges
- smooths other regions

Applying the filter is as follows:

$$\hat{f}_B(\mathbf{u}) = \frac{1}{k_{\mathbf{u}}} \sum_{\mathbf{v} \in \Omega} f(\mathbf{u} - \mathbf{v}) w_B^{\mathbf{u}}(\mathbf{v}).$$

Bilateral (B) filter (2/2)



Bilateral filter weights at the central pixel

Bilateral Median (BM) filter

Bilateral Median filter: solve the weighted median problem:

$$\hat{f}(\mathbf{u}) = \arg\min_{y} \sum_{\mathbf{v}\in\Omega} w_B^{\mathbf{u}}(\mathbf{v}) \left| y - f(\mathbf{u}-\mathbf{v}) \right| .$$

The use of the median enables even sharper regions.

Fast implementations exist.[1,2,3]

[1] http://nomis80.org/ctmf.pdf: `http://nomis80.org/ctmf.pdf`

[2] `http://www.shellandslate.com/fastmedian.html`

[3] 100+ Times Faster Weighted Median Filter: `http://www.cse.cuhk.edu.hk/~leojia/projects/fastwmedian/index.htm`

The use of joint filtering

'Joint'/'Cross'/'Guided' filtering:

- considering various modalities to form **weights**
- filtering carried out by **sampling** a different modality

E.g., the case of RGB-D: weights formed using *colors* (image 'f'), filtering happens by modifying *depth* (image 'g').

Example: Joint Bilateral (JB) filtering (1/2)

Weights are based on color similarity in 'f':

$$w_{JB,f}^{\mathbf{u}}(\mathbf{v}) = p\left(\|f(\mathbf{u}) - f(\mathbf{u}-\mathbf{v})\|\right) w_G(\mathbf{v}).$$

Filtering is performed by sampling 'g', based on weights $w_{JB,f}^{\mathbf{u}}$:

$$\hat{g}_{JB,f}(\mathbf{u}) = \frac{1}{k_{\mathbf{u}}} \sum_{\mathbf{v}\in\Omega} g(\mathbf{u}-\mathbf{v}) w_{JB,f}^{\mathbf{u}}(\mathbf{v}).$$

It is also easy to derive the Joint Bilateral Median (JBM) filter.

Example: Joint Bilateral (JB) filtering (2/2)

Joint Bilateral   Joint Bilateral Median

Applications of Joint filtering (1/2)

- In general: Denoising, Edge-preserving filtering, *etc.*
- Fusing **flash / no-flash** images.
- **IR-Color fusion**, to enhance IR.
- **Depth-Color fusion**, to enghance Depth.
- **Video** enhancement (temporal).
- HDR compression.
- (Single image) fog removal.
- *etc.*

Applications of Joint filtering (2/2)

Joint filtering of no-flash/flash images.

g f $\hat{g}_{JB,f}$

Upsampling

Guided upsampling through specialized joint filtering:

- a *low-resultion* input (*e.g.*, ToF-depth) is to be upsampled
- guided by a *high-resolution* image overlapping the same view

Straightforward approach – use modified sampling:

- Transforming a high-res integer coordinate pair $\mathbf{q}$ to the *nearest* low-res integer pixel coordinate pair $\mathbf{q}_{\downarrow}$.
- Apply during joint filtering (i.e., '*JBU*' or '*JBMU*'), *e.g.*,

$$\hat{g}_{JBU,f}(\mathbf{u}) = \frac{1}{k_{\mathbf{u}}} \sum_{\mathbf{v}\in\Omega} g\left([\mathbf{u}-\mathbf{v}]_{\downarrow}\right) w^{\mathbf{u}}_{JB,f}(\mathbf{v}).$$

Another approach is *Iteratively upsampling*, taking powers-of-two steps to always double the resolution the apply JB or JBM.

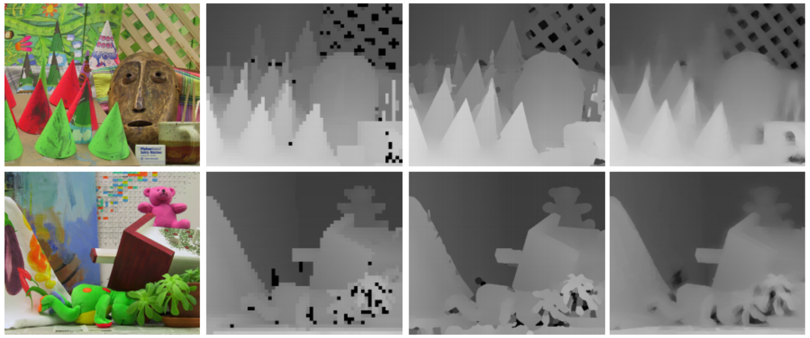
Iterative upsampling

Algorithm 1 Algorithm for upsampling a depth image $\mathtt{D}$ using a guidance image $\tilde{\mathtt{I}}$.

function $\text{UPSAMPLE}(@\text{FILTER}, \mathtt{D}, \tilde{\mathtt{I}}, params)$
  $uf \leftarrow \lfloor \log_2 (\text{size}(\tilde{\mathtt{I}}) / \text{size}(\mathtt{D})) \rfloor$ ▷ upsample factor
  $\hat{\mathtt{D}} \leftarrow \mathtt{D}$
  for $i \leftarrow 1$ to $(uf - 1)$ **do**
    $\hat{\mathtt{D}} \leftarrow \text{resize}(\hat{\mathtt{D}}, 2 \cdot \text{size}(\hat{\mathtt{D}}))$
    $\tilde{\mathtt{I}}_{lo} \leftarrow \text{resize}(\tilde{\mathtt{I}}, \text{size}(\hat{\mathtt{D}}))$
    $\hat{\mathtt{D}} \leftarrow @\text{FILTER}(\hat{\mathtt{D}}, \tilde{\mathtt{I}}_{lo}, params)$
  end for
  $\hat{\mathtt{D}} \leftarrow \text{resize}(\hat{\mathtt{D}}, \text{size}(\tilde{\mathtt{I}}))$
  $\hat{\mathtt{D}} \leftarrow @\text{FILTER}(\hat{\mathtt{D}}, \tilde{\mathtt{I}}, params)$
  return $\hat{\mathtt{D}}$
end function

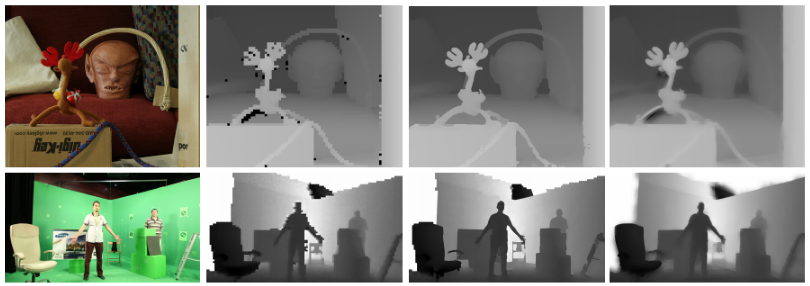
@FILTER is a joint filter, juch as JB or JBM.

Examples of guided depth upsampling (1/2)



input colour image | resized depth | depths upsampled with Algorithms 2 (left) and 3 (right)

Examples of guided depth upsampling (2/2)



input colour image | resized depth | depths upsampled with Algorithms 2 (left) and 3 (right)

Readings

- Image-guided ToF depth upsampling: a survey
- Wikipedia: Bilateral filter
- OpenCV: Bilateral filter, Adaptive Bilateral filter