Bauhaus-Universität Weimar

Image Analysis and Object Recognition

Exercise 5

SIFT
Scale Invariant Feature Transform

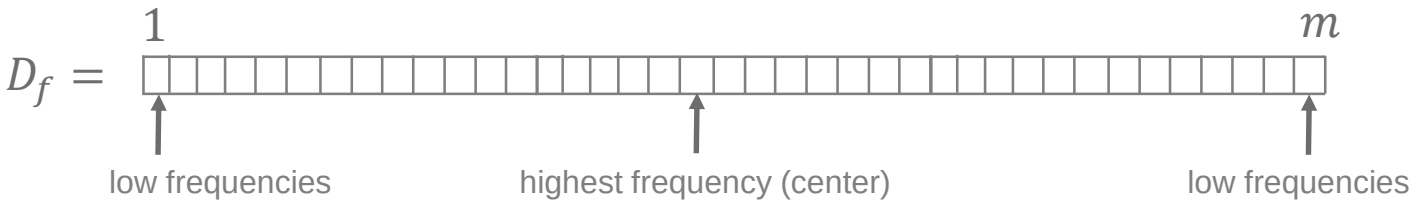
SS 2014

(Course notes for internal use only!)

Comments Ex 4

Generalization of descriptor by using $n = 24$ elements

- Number of elements m in $D = [(x_1 + jy_1), \ldots, (x_m + jy_m)]^T$
- Fourier transform $D_f = fft(D)$

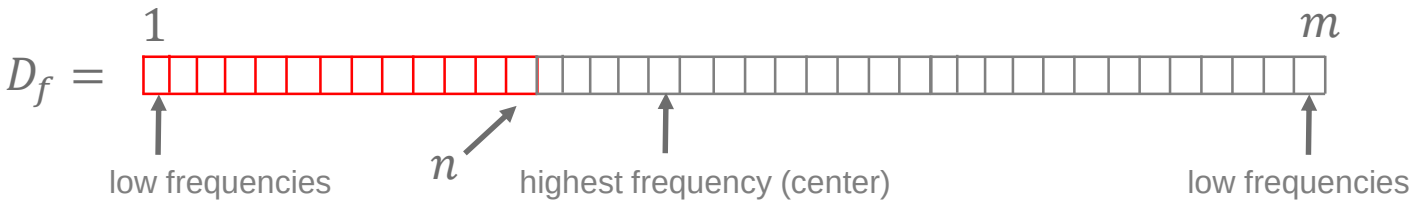


→ **Symmetric vector**

Comments Ex 4

Generalization of descriptor by using $n = 24$ elements

- Number of elements m in $D = [(x_1 + jy_1), \dots, (x_m + jy_m)]^T$
- Fourier transform $D_f = fft(D)$



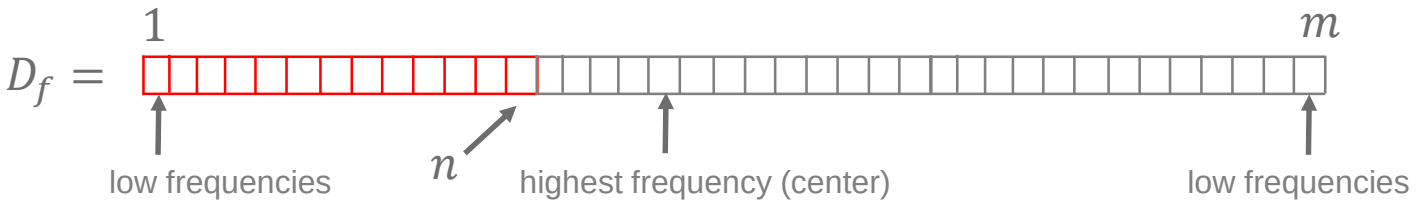
→ **Symmetric vector**

→ Extract first n values → generalization

Comments Ex 4

Generalization of descriptor by using $n = 24$ elements

- Number of elements m in $D = [(x_1 + jy_1), \dots, (x_m + jy_m)]^T$
- Fourier transform $D_f = fft(D)$



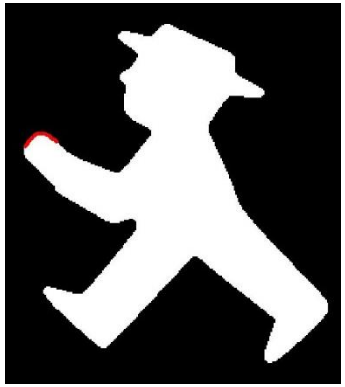
→ **Symmetric vector**

→ Extract first n values → generalization

- Fourier transform $D_f = fft(D, n)$ → **only transforms n elements of D!**

$n = 842$

$n = 421$



$n = 24$

Bauhaus-Universität Weimar

Exercise 5

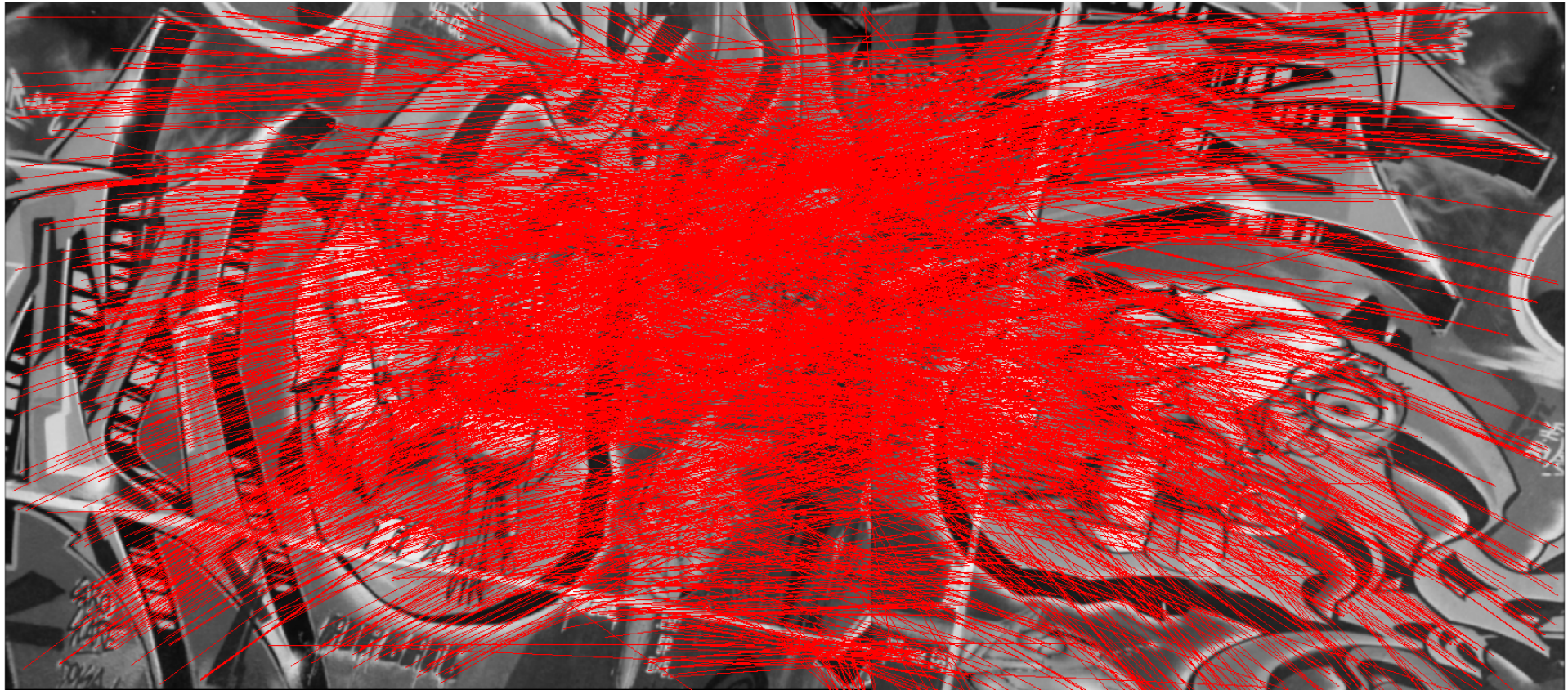
SIFT – Scale Invariant Feature Transform

Motivation

- Reliable matching between different views of an object or scene
  - Recognition, Motion tracking, Multiview geometry,…

Motivation

- Reliable matching between different views of an object or scene
  - Recognition, Motion tracking, Multiview geometry,…



Desired Specifications

- Many points
- Repeatable Extraction
- Scale and orientation invariant
- Fast to extract
- Not only positions in image and scale
  → Descriptor vector for each point
- Fast and easy to match

SIFT

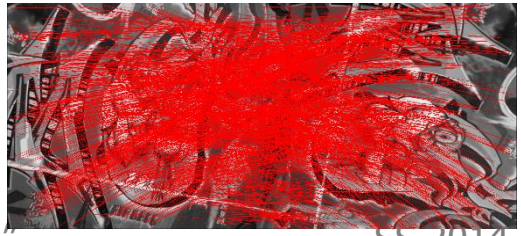
- Distinctive image features from scale-invariant keypoints. David G. Lowe, International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

→ Finds keypoints
→ Generates describing image features for them
→ Invariant to image scaling and rotation
→ Partially invariant to changes in illumination and camera viewpoints
→ Many keypoints are found in typical images
→ Highly distinctive

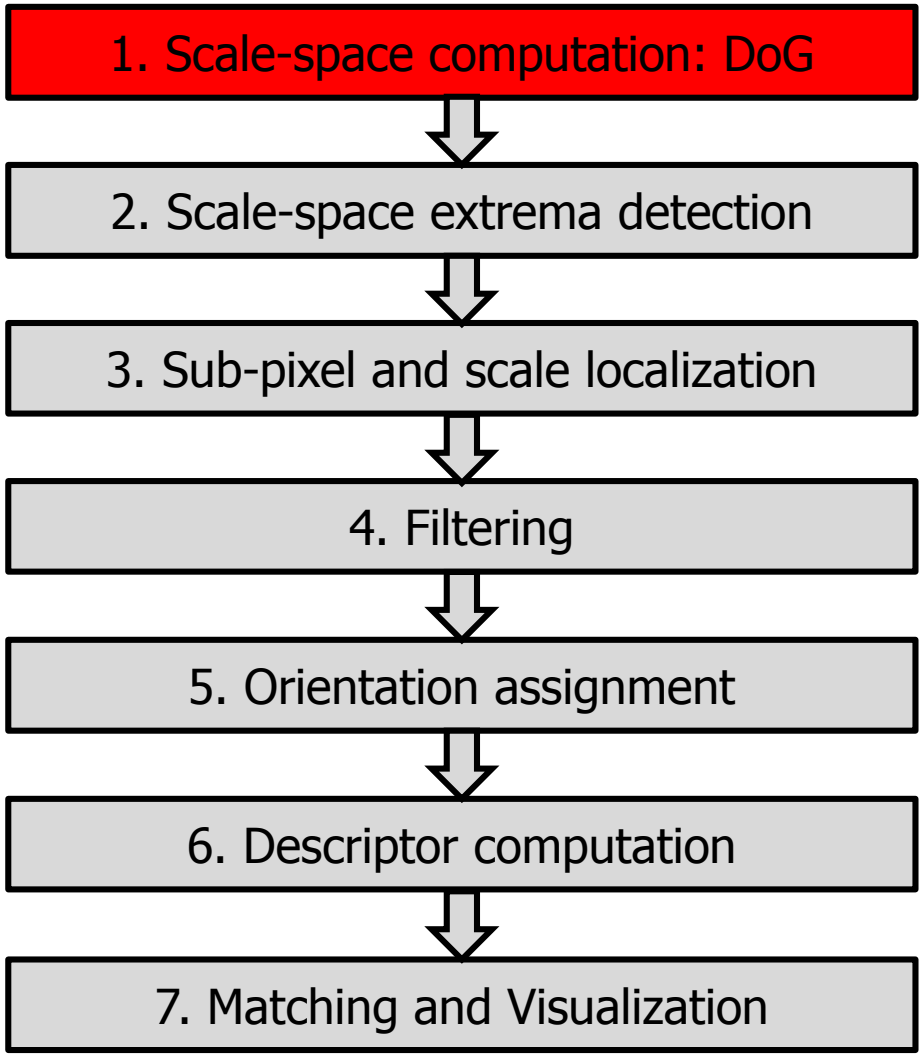
Algorithm Outline

2 Input images

1. Scale-space computation: DoG
2. Scale-space extrema detection
3. Sub-pixel and scale localization
4. Filtering
5. Orientation assignment
6. Descriptor computation
7. Matching and Visualization



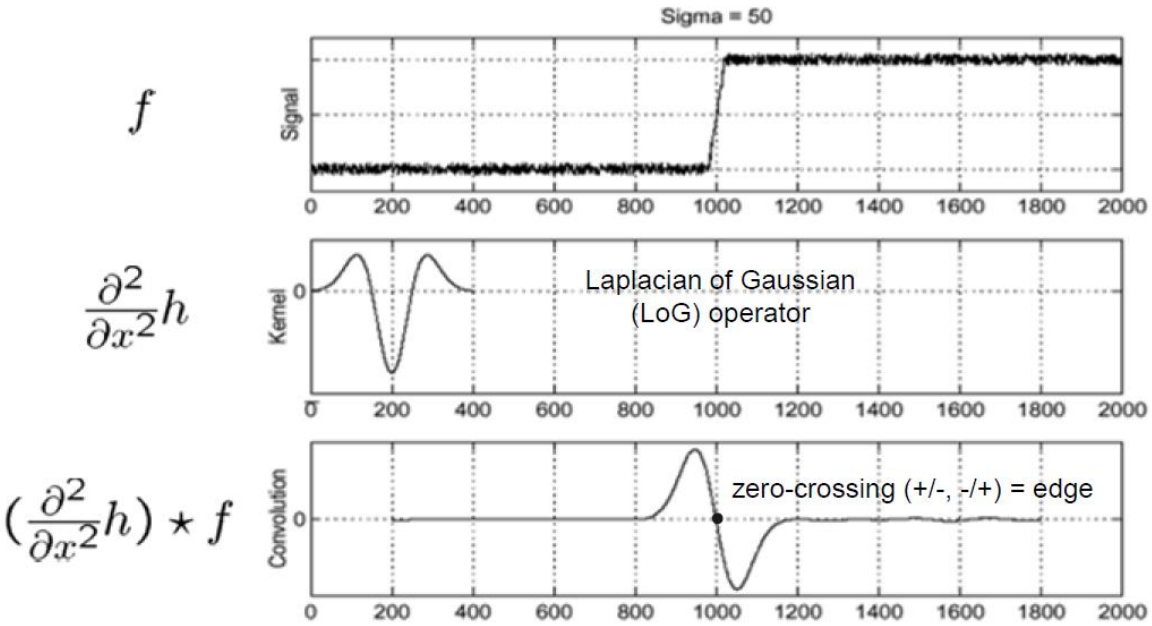
Algorithm Outline



Scale-Space

- Laplacian of Gaussian filtering (LoG)
  - Filtering a signal with second derivatives of Gaussian function
  - Highlights regions of rapid intensity change → zero crossings

Consider: $\frac{\partial^2}{\partial x^2}(h \star f)$



Scale-Space

LoG-approximation using Difference of Gaussians (DoG):

- Image blurring using Gaussian filtering:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

- Difference of Gaussians (DoG):

$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned}$$

- Easy and efficient to compute

Scale-Space: Example

- Input image

Scale-Space: Example

Blurred image

DoG-Image

$\sigma = 2.0$

Scale-Space: Example

Blurred image

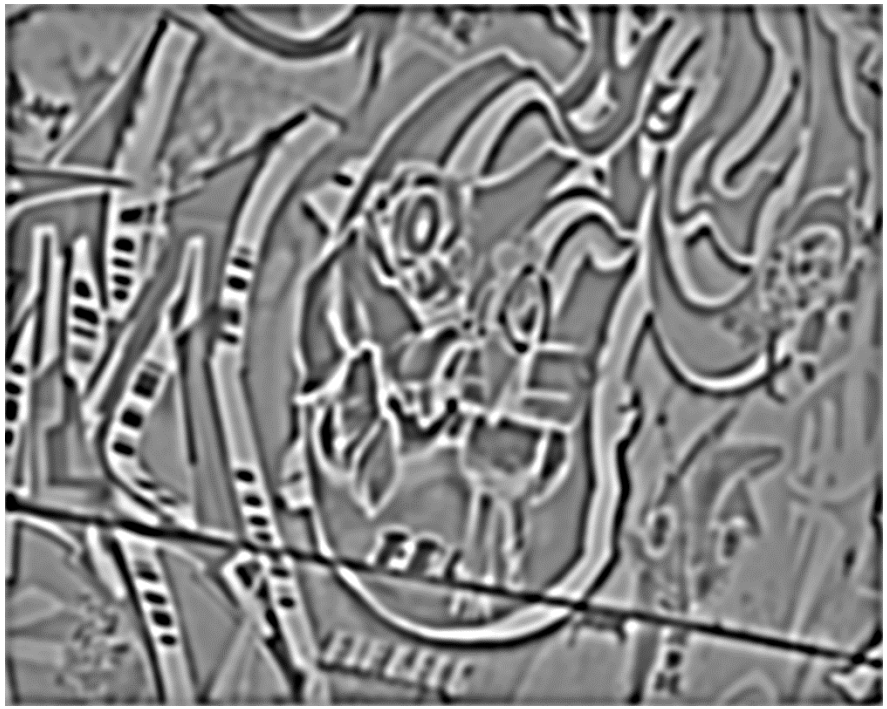
DoG-Image

$\sigma = 4.0$

Scale-Space: Example

Blurred image

DoG-Image

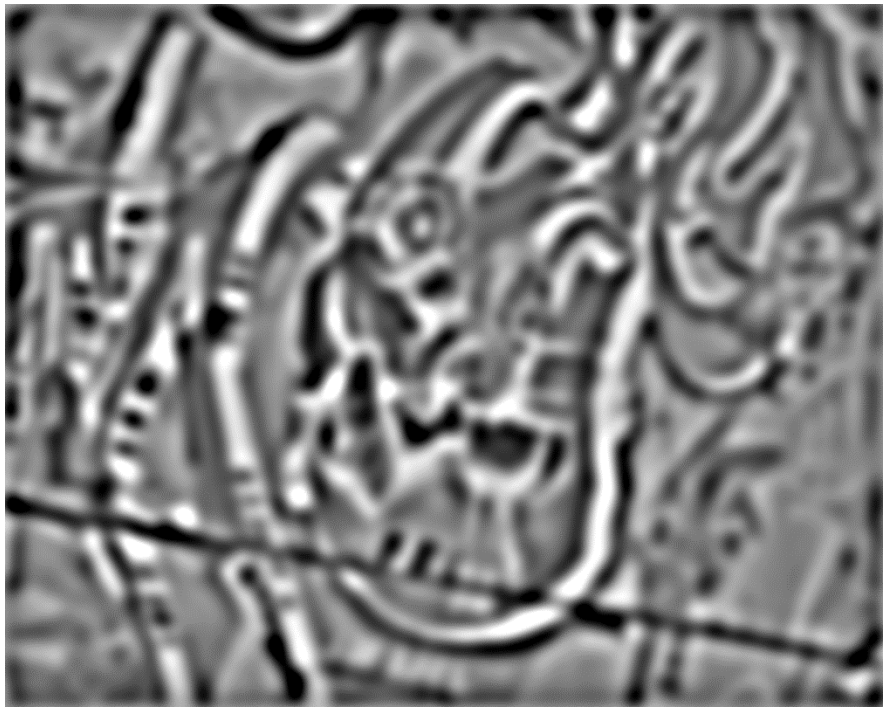


$\sigma = 8.0$

Scale-Space: Example

Blurred image

DoG-Image

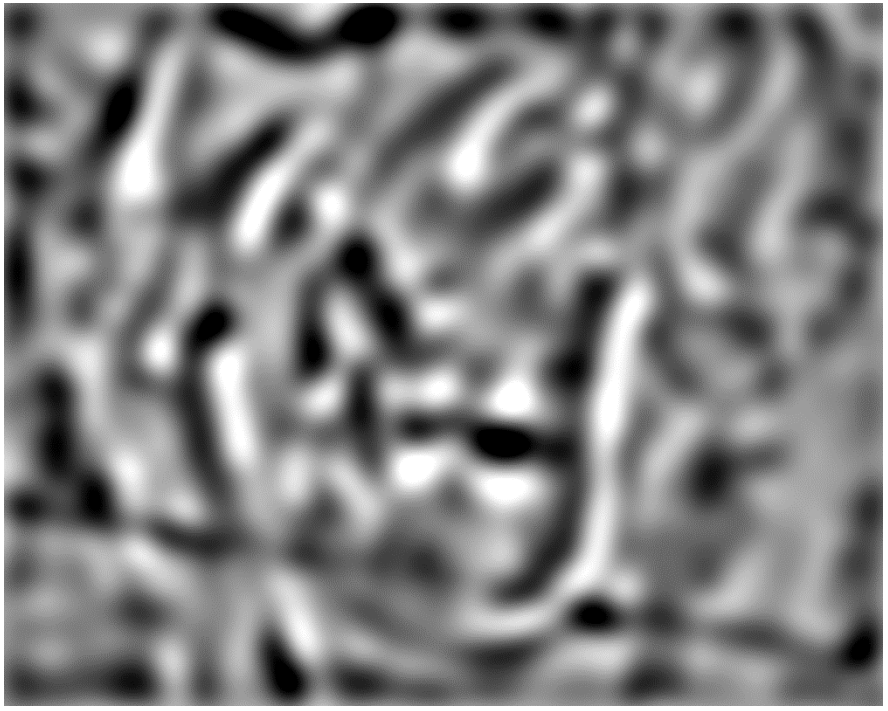


$\sigma = 16.0$

Scale-Space: Example

Blurred image

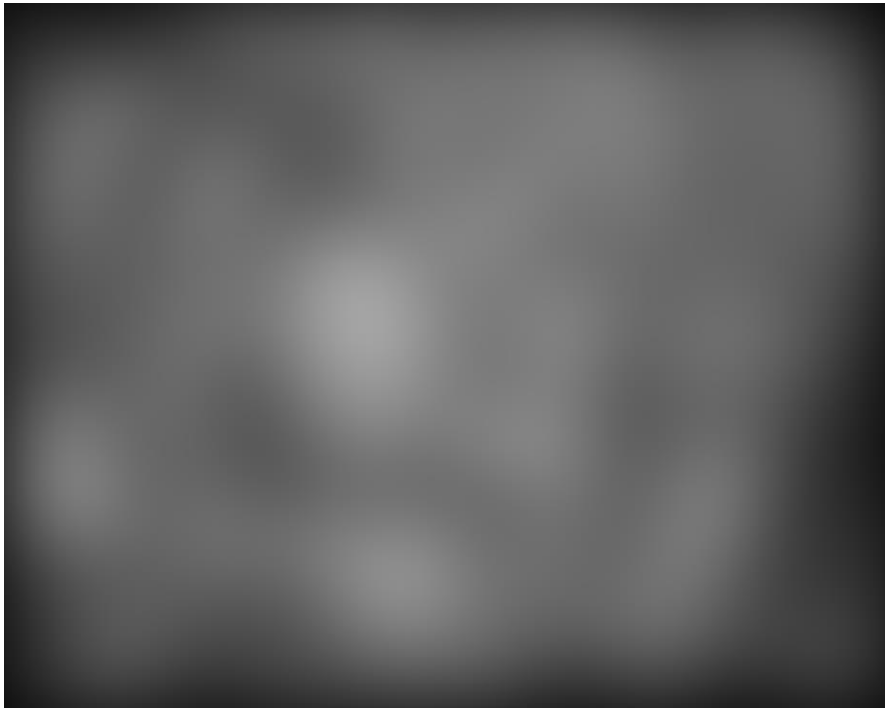
DoG-Image



$\sigma = 32.0$

Scale-Space: Example

Blurred image

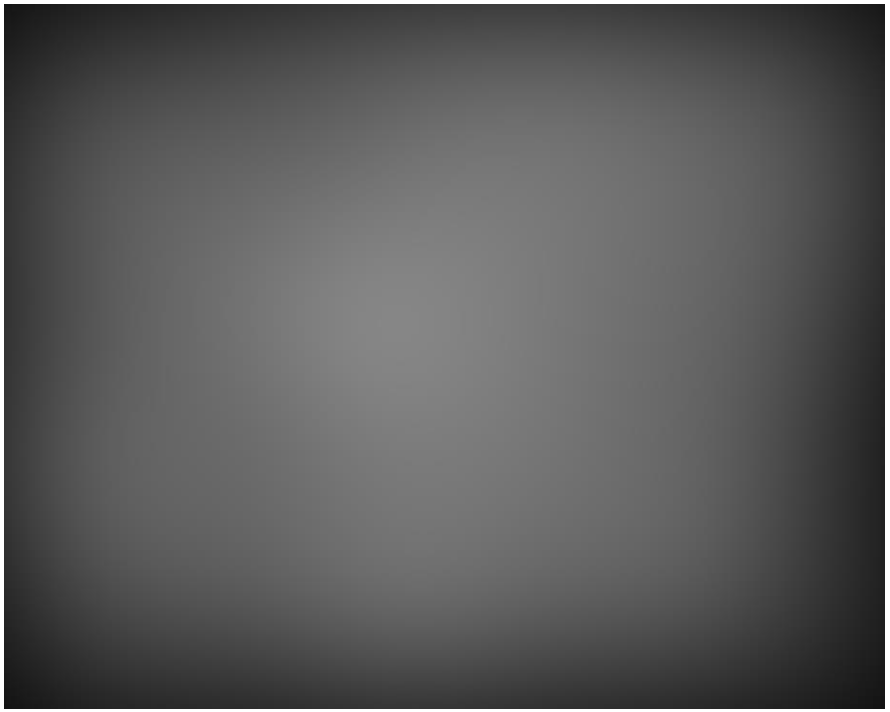


DoG-Image

$\sigma = 64.0$

Scale-Space: Example

Blurred image

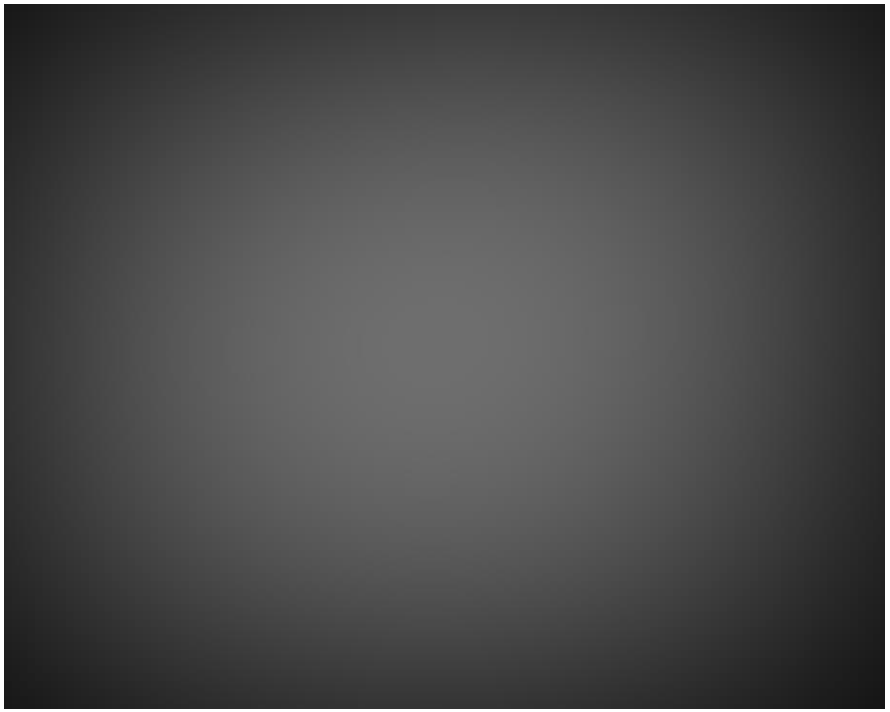


DoG-Image

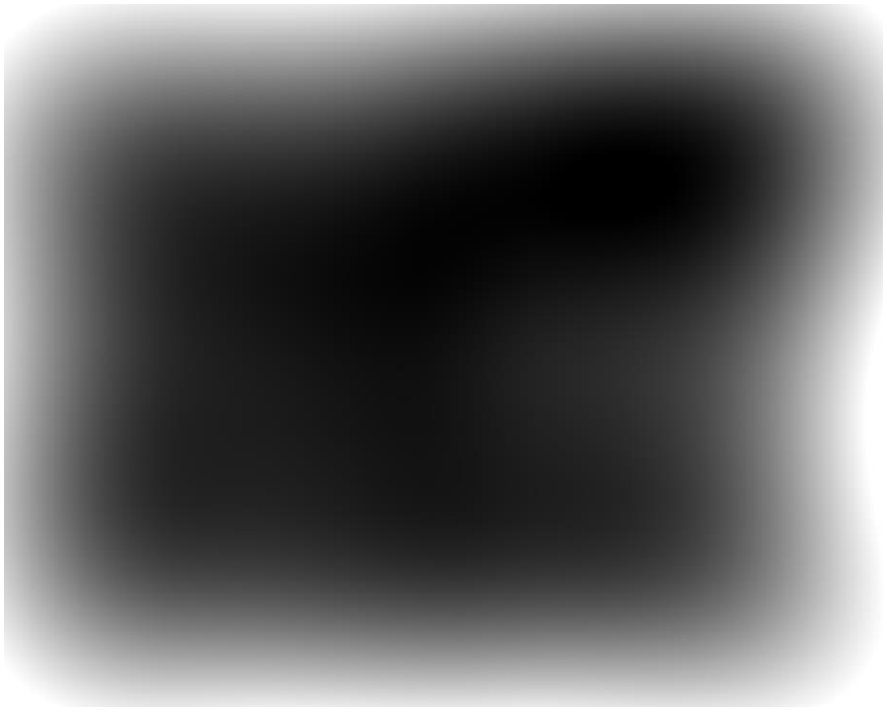
$\sigma = 128.0$

Scale-Space: Example

Blurred image



DoG-Image

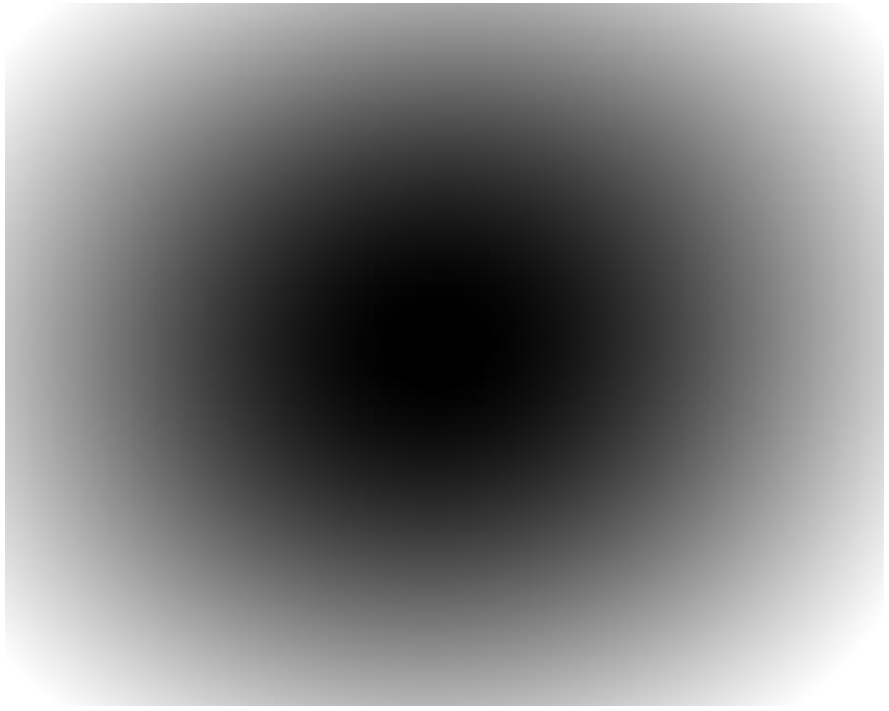


$\sigma = 256.0$

Scale-Space: Example

Blurred image

DoG-Image



$\sigma = 512.0$

Scale-Space: Prior Smoothing

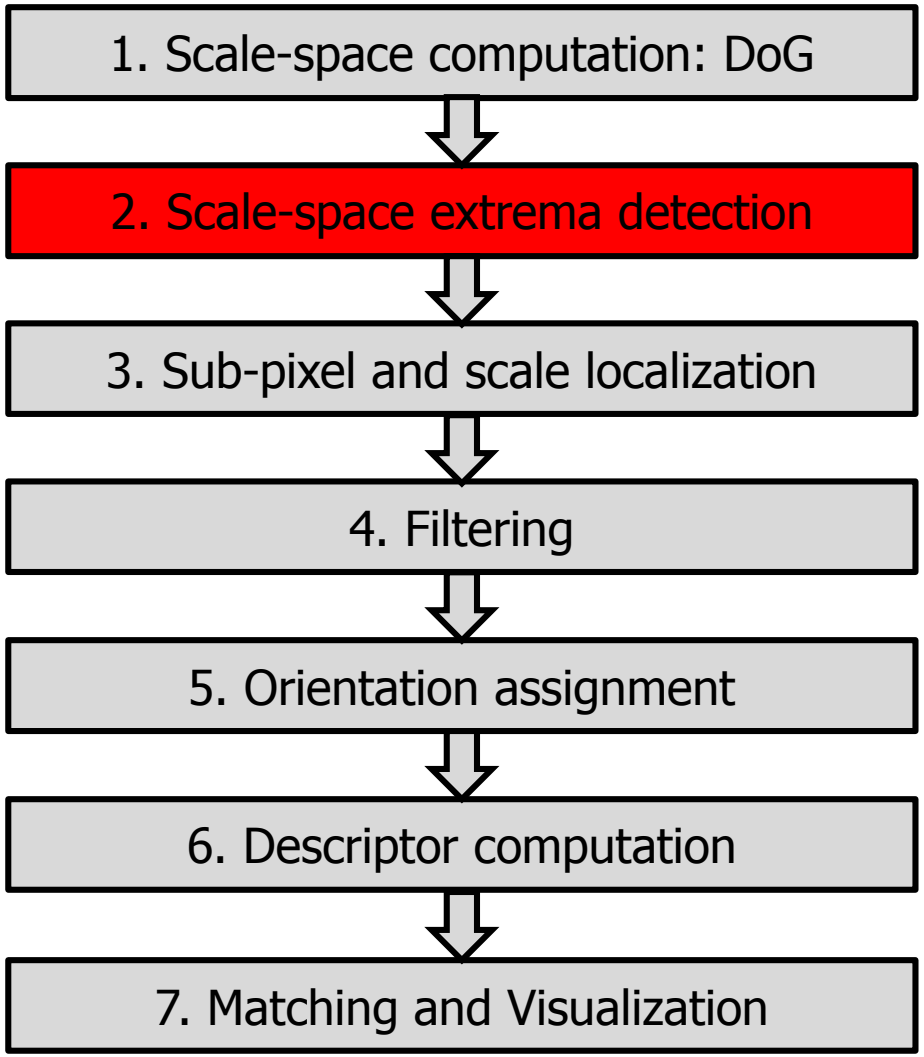
Prior Gaussian smoothing of the input image

- Increasing σ also increases **robustness** and **repeatability**
- $\sigma = 1.6$ has shown to be good (found empirically)

- Smoothing discards highest frequencies

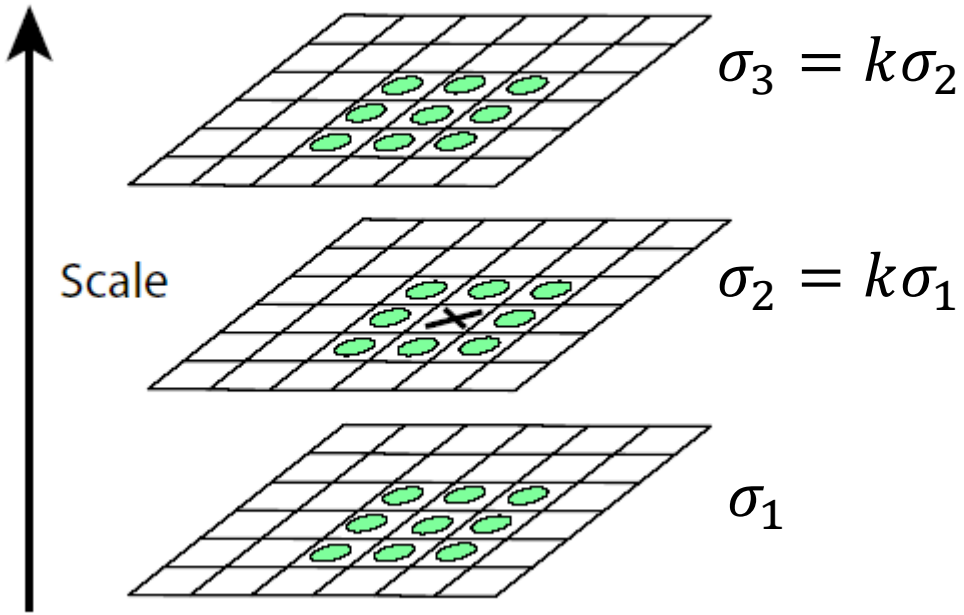
→Doubling the image initially increases number keypoints

Algorithm Outline



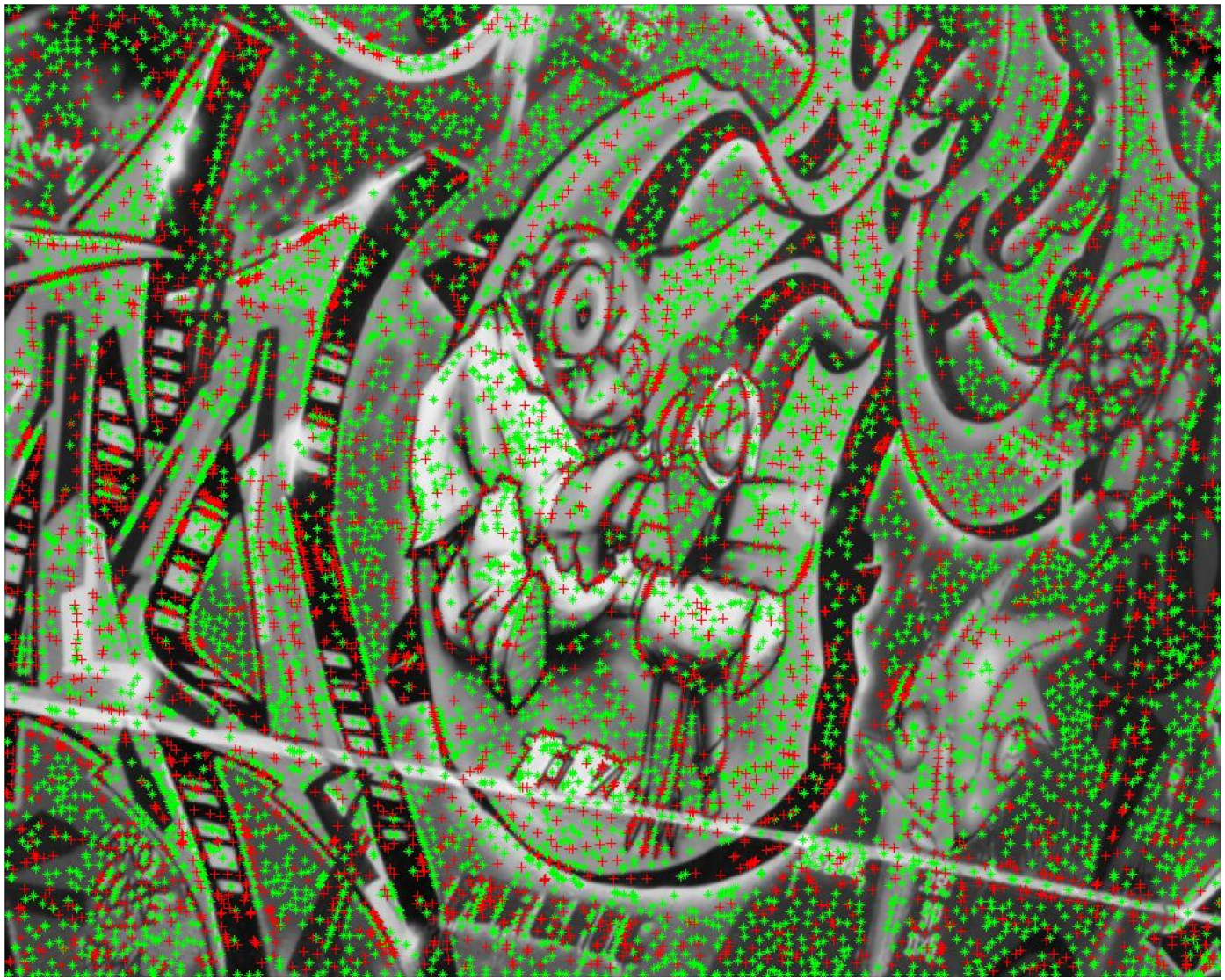
Extrema Detection

- Find local Maxima and Minima in DoG-Pyramid
  → Check local neighborhood



- 8 Pixels in scale σ_2
- 9 Pixels in scale σ_1
- 9 Pixels in scale σ_3

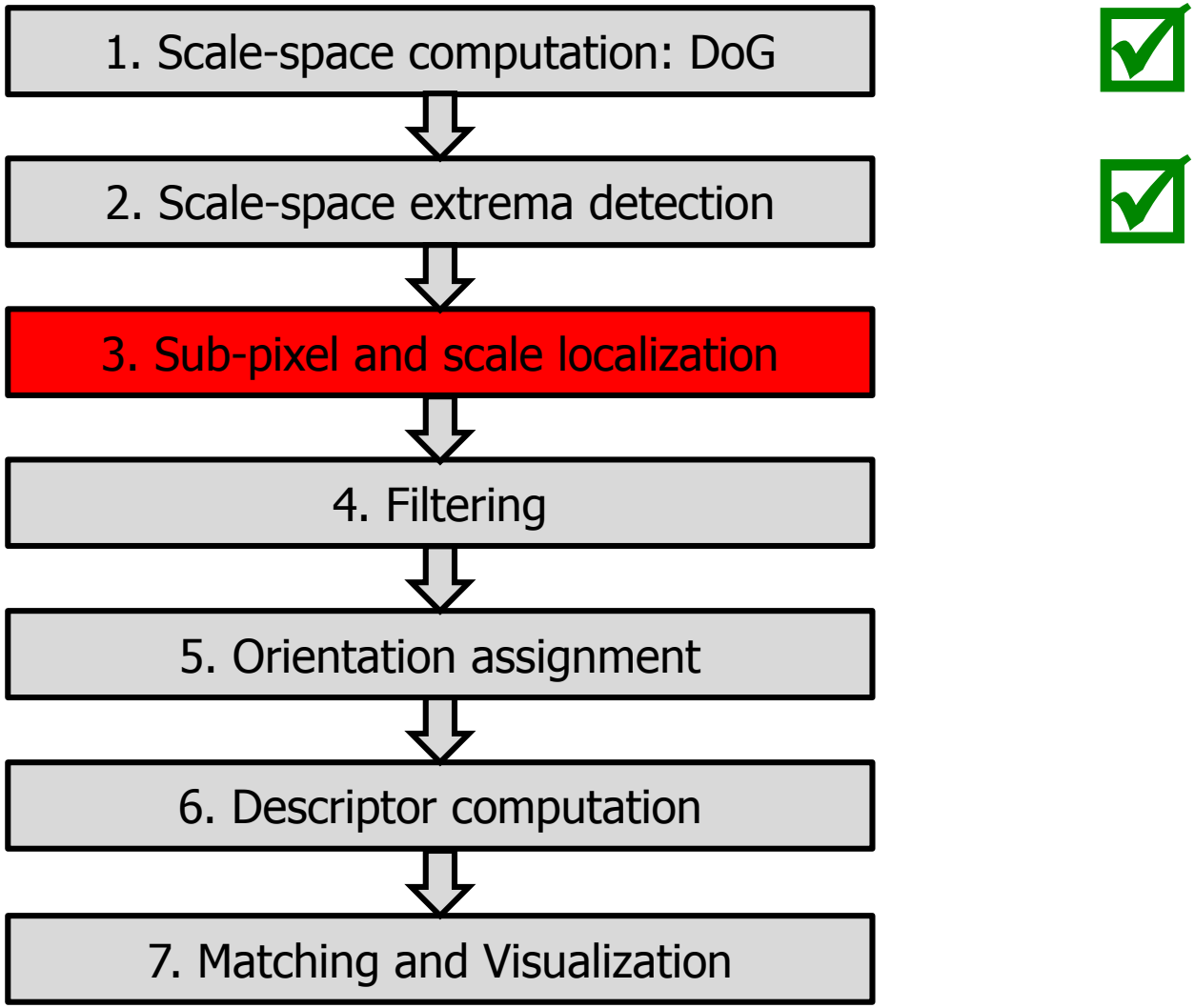
Extrema Detection Result



Red: minima

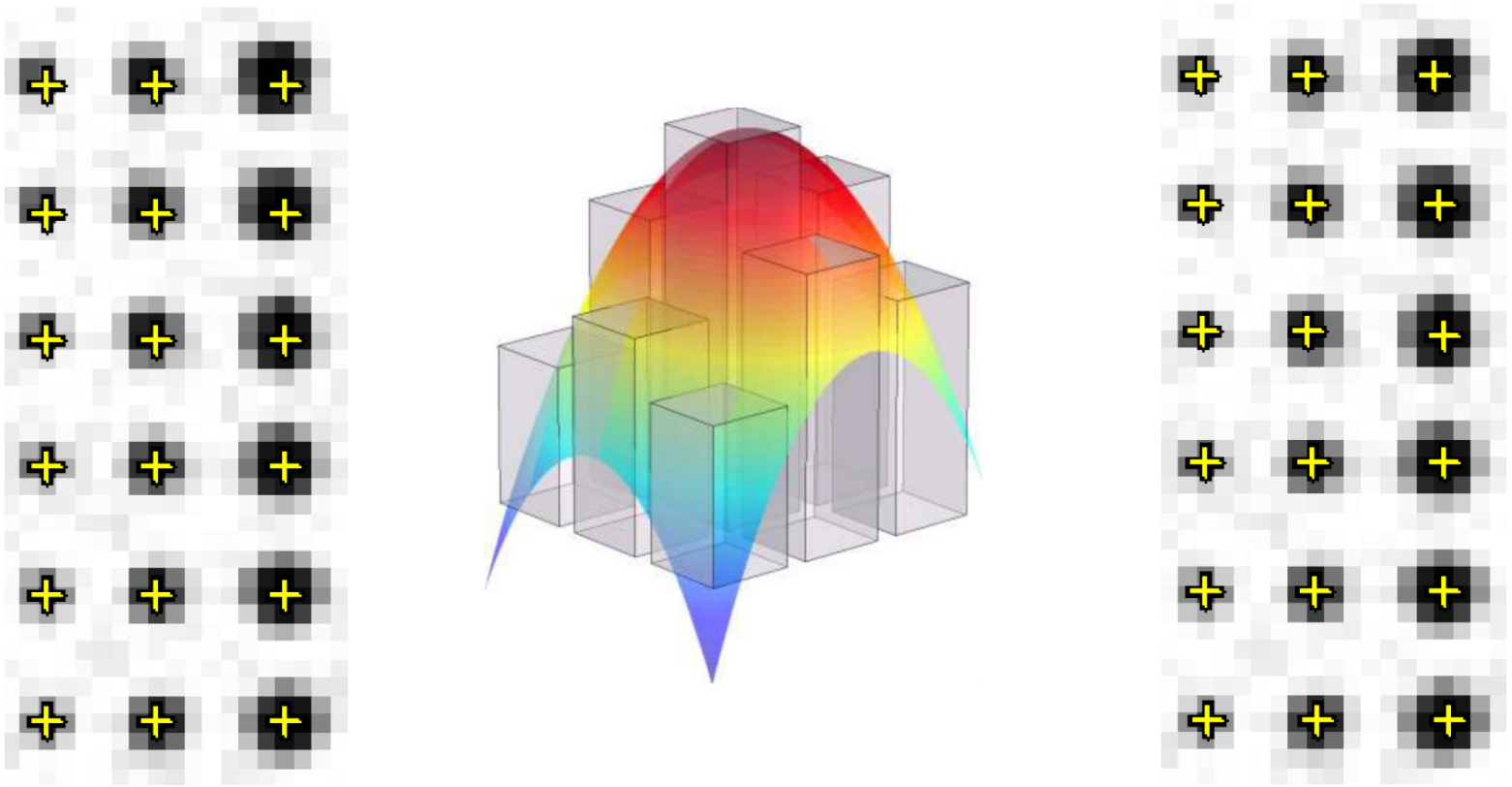
Green: maxima

Algorithm Outline



Localization

- Aim: Find the subpixel position of keypoint candidates in position and scale (**3D**)
- Example for 2D: Paraboloid fitting



Localization

- Fit 3D quadratic functions to local keypoint candidates

- Taylor expansion with keypoint candidate X as origin

$$D(X) = D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X, \text{ where } X = (x, y, \sigma)$$

- Computation of derivatives: simple pixel differences

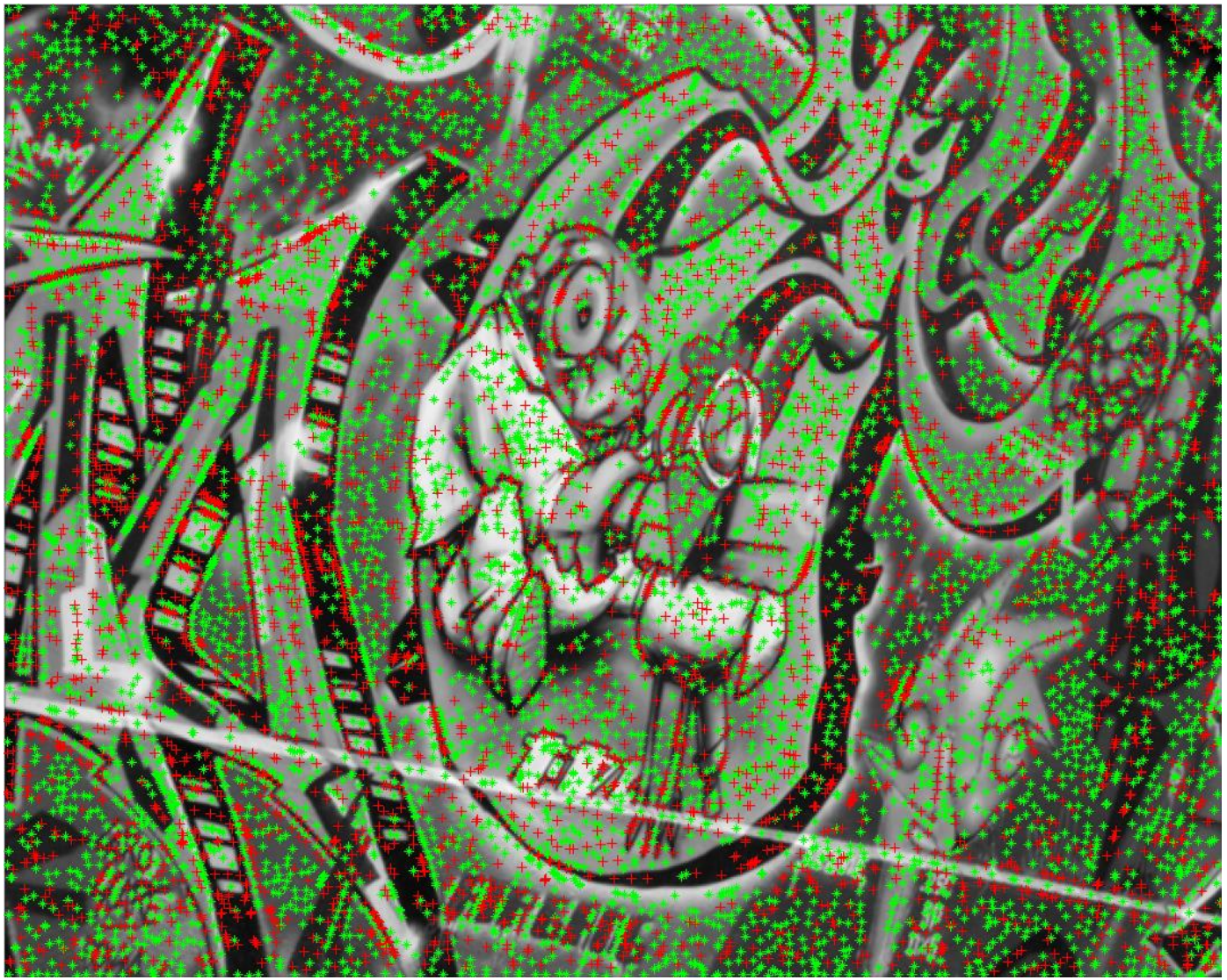
- Set first derivative wrt. X to zero

$$0 = \frac{\partial D}{\partial X} + \frac{\partial^2 D}{\partial X^2} \hat{X}$$

- Solve corresponding linear 3×3 system

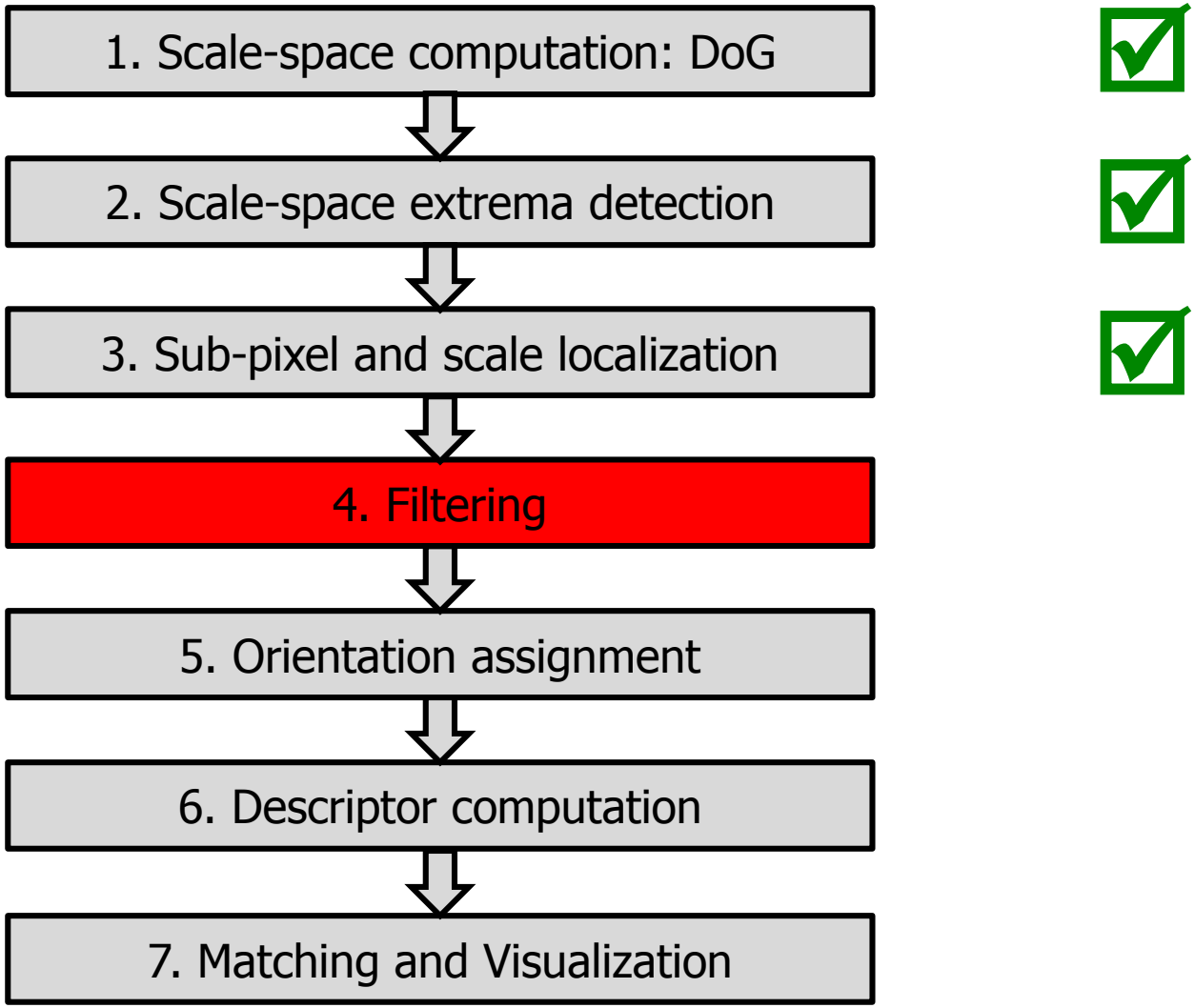
$$\hat{X} = -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X}$$

Localization: Initial Points



Localization: Estimated Locations

Algorithm Outline



Filtering

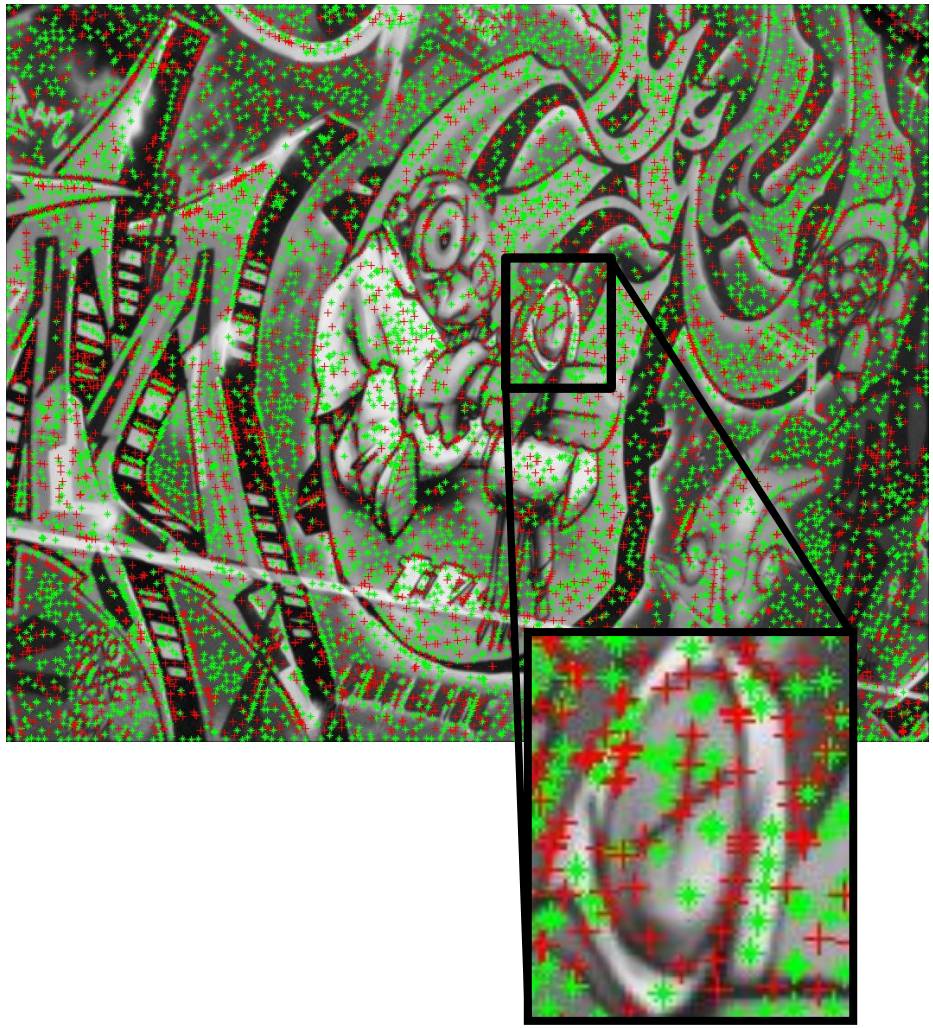
- Filter low contrast points, if

$$\left|D\left(\hat{X}\right)\right| = \left|D + \frac{1}{2}\frac{\partial D^T}{\partial X}\hat{X}\right| < 0.03$$

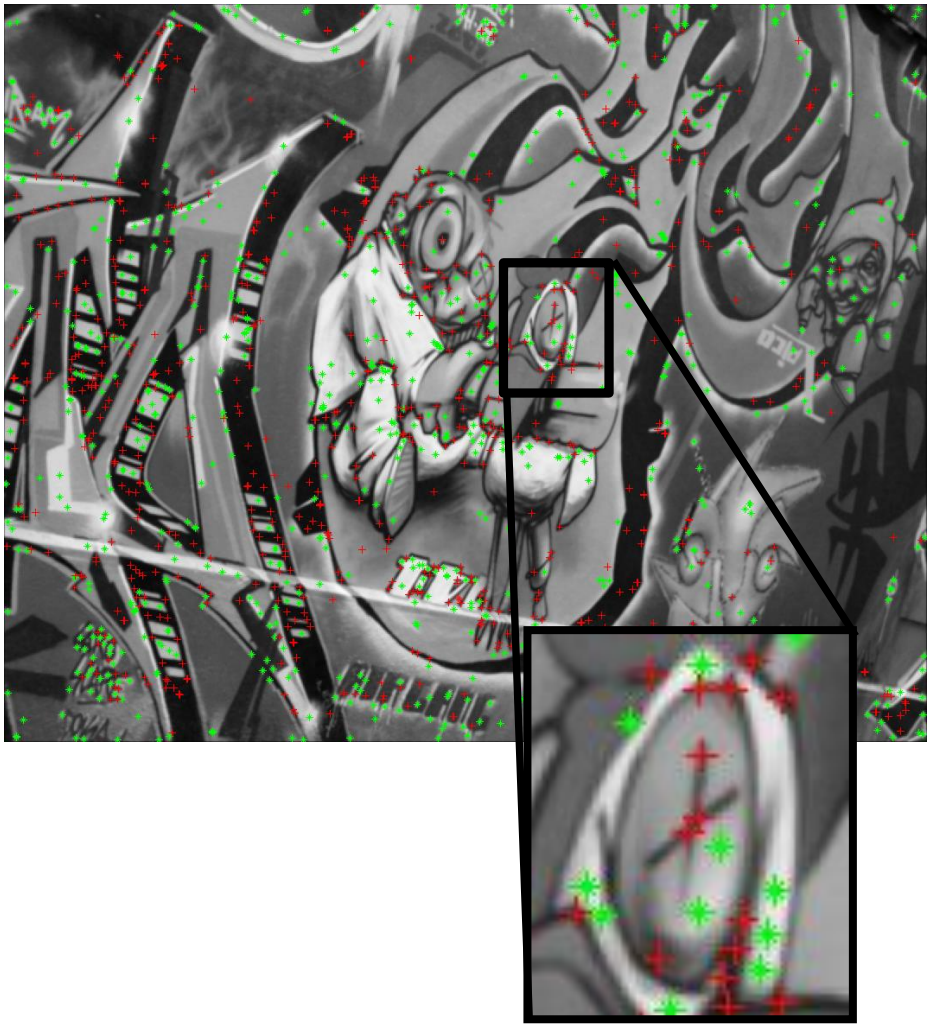
- More stability: Eliminate edge-responses
  - One large principal curvature across the edge but a small one in perpendicular direction
  - Principal curvatures from Hessian: $H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$
  - If $\frac{Trace(H)}{Det(T)} > \frac{(r+1)^2}{r}$ → throw point away (e.g. with $r = 10$)

Filtering Results

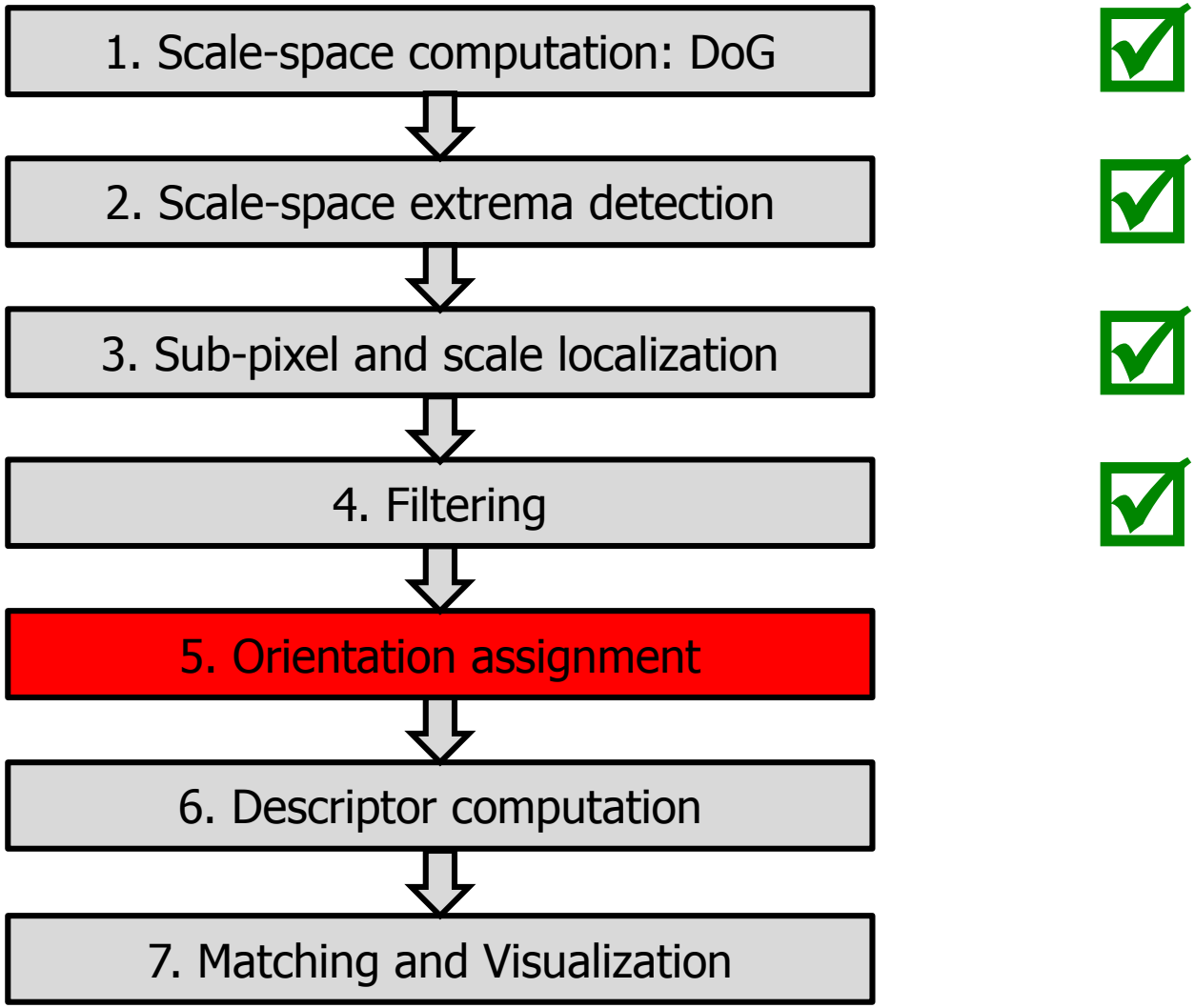
Localized Points



Filtered Points



Algorithm Outline



Assign Gradient Orientations

- **Idea**: achieving **orientation invariance** by computing descriptors relative to keypoints main orientation

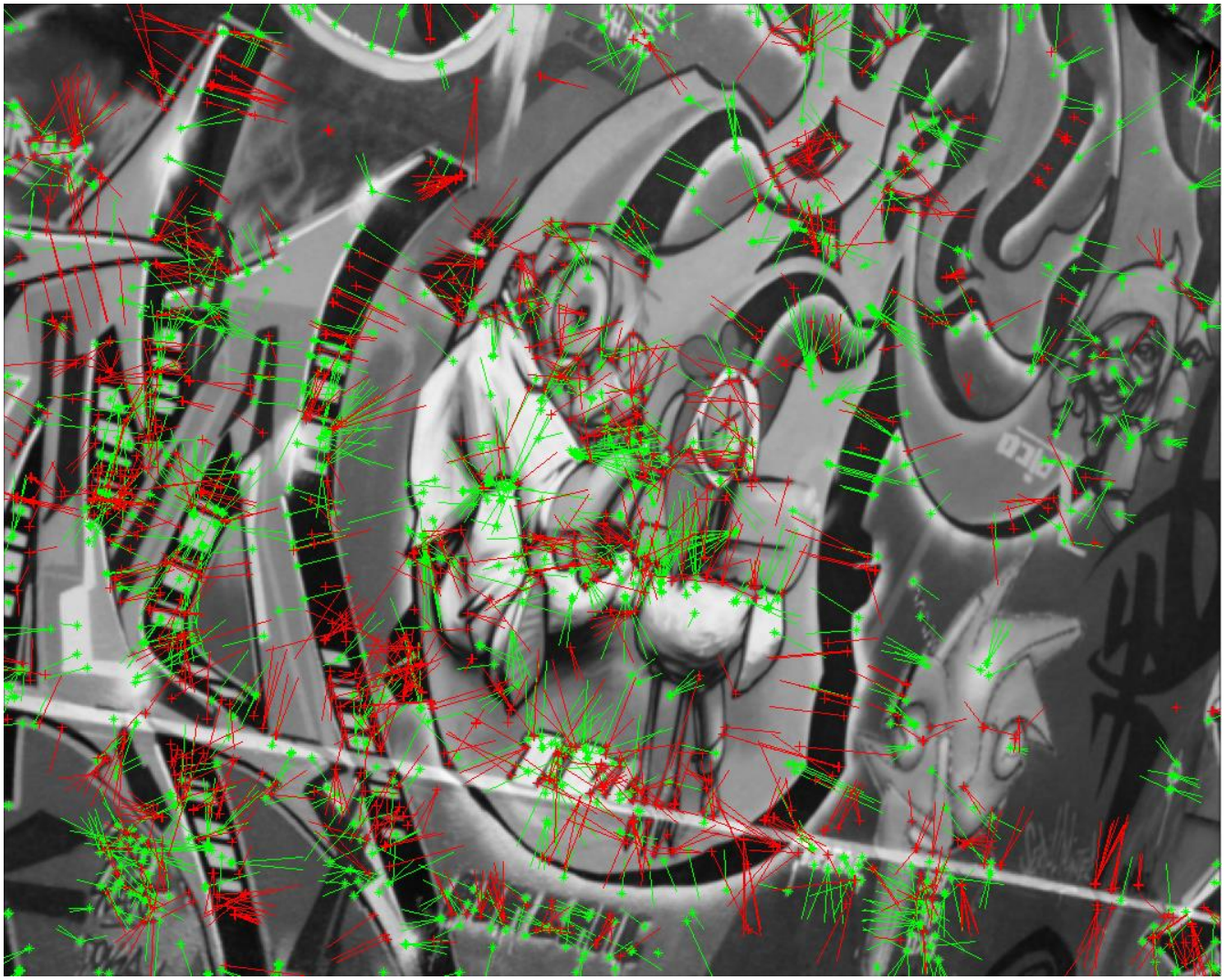
- Computation of Gradient orientations θ and magnitudes m for all scales and pixels using Gaussian filtered images L

$$m(x,y) = \sqrt{\big(L(x+1,y) - L(x-1,y)\big)^2 + \big(L(x,y+1) - L(x,y-1)\big)^2}$$

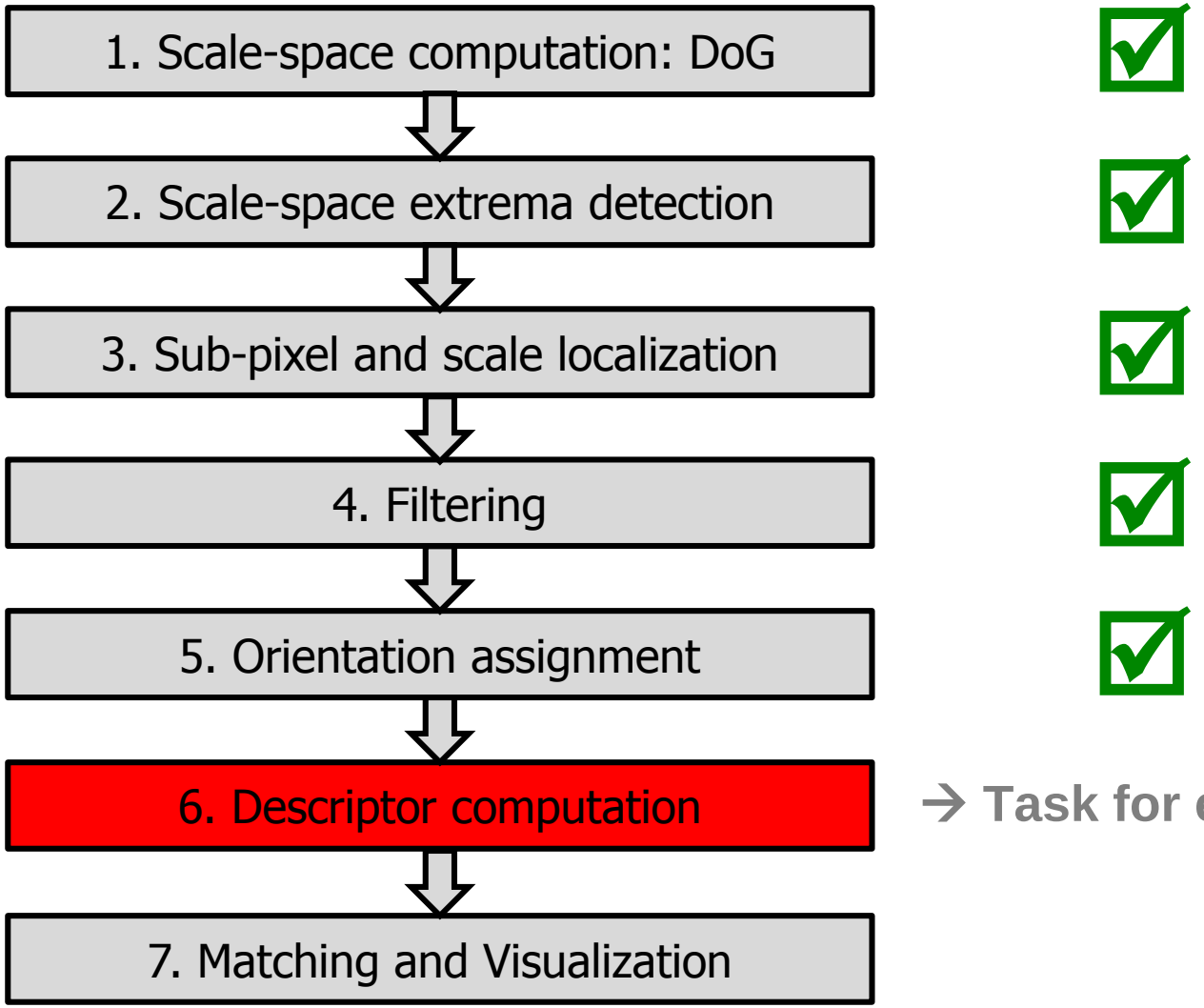
$$\theta(x,y) = atan\Big(\big(L(x,y+1) - L(x,y-1)\big)/\big(L(x+1,y) - L(x-1,y)\big)\Big)$$

- Use this information to assign main orientation of local neighborhood to each keypoint

Orientation Assignment: Result



Algorithm Outline



Descriptor Computation: Overview

- **Inputs**
  - Gradient magnitude and angle images for each scale
  - Keypoint positions + their main local orientation

- **Outputs**
  - 128-element descriptor vector for each keypoint

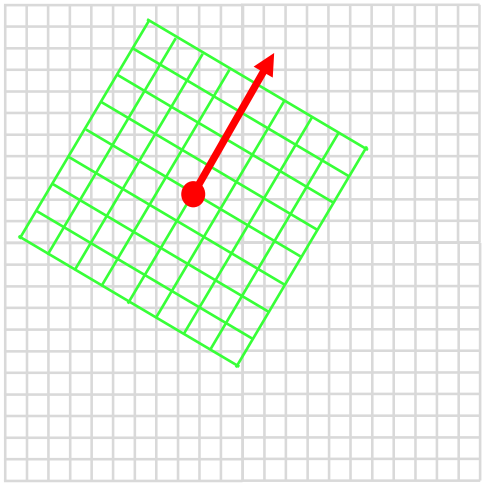
  → This has to be computed!

  - 5-element vector for each keypoint: $[y, x, \sigma, \theta_m, m_m]$, where
    - (y, x): subpixel image coordinates
    - σ: scale
    - θ_m: main gradient orientation
    - m_m: main gradient magnitude

  → These values are already available, rearrange them (for visualization)

Descriptor Computation: Overview

Keypoint and gradient direction

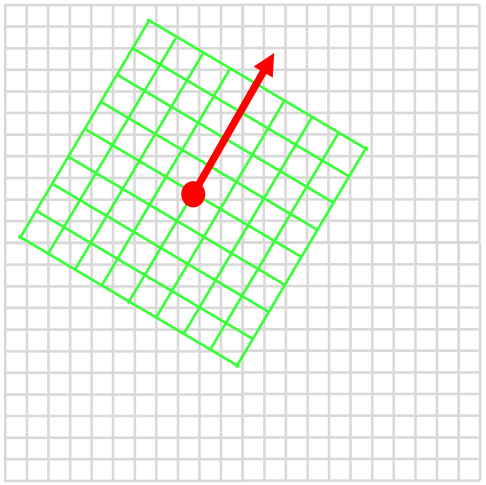


Main idea: Use gradient magnitude and orientation of local neighborhood to compute **scale and rotation invariant** descriptor

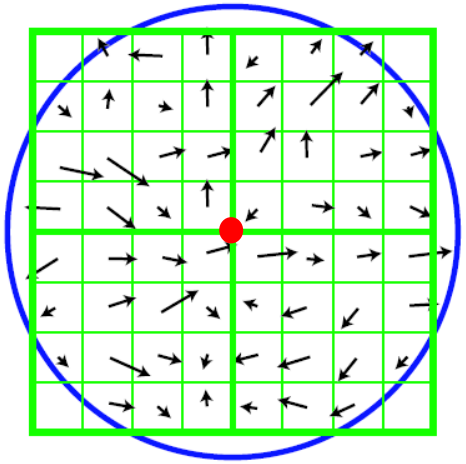
- Rotate sampling window by assigned main gradient direction

Descriptor Computation: Overview

Keypoint and gradient direction



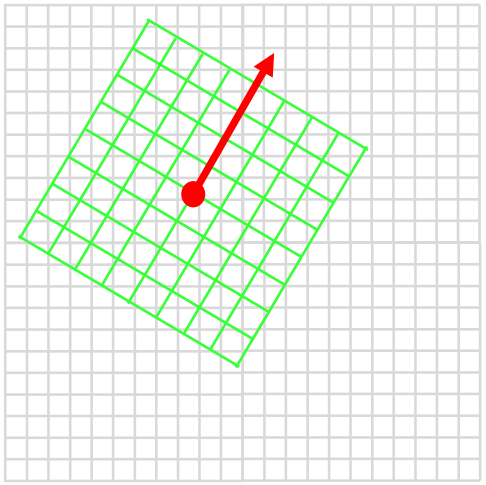
Sampled gradients



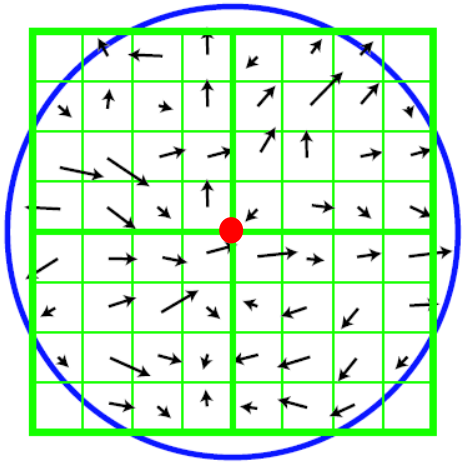
- Rotate sampling window by assigned main gradient direction
- Sample gradient magnitudes and orientations and weight magnitudes using a Gaussian function (blue circle)

Descriptor Computation: Overview

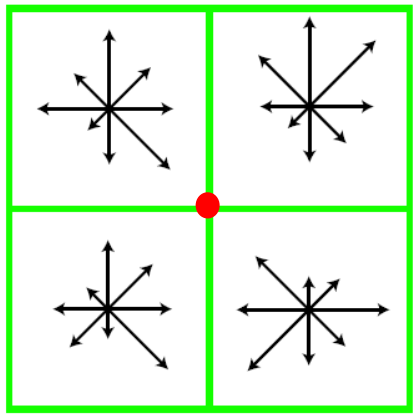
Keypoint and gradient direction



Sampled gradients



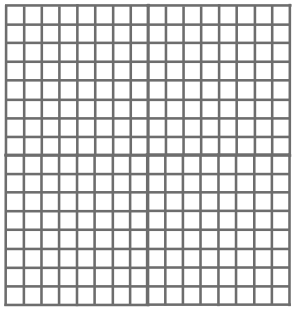
Descriptor

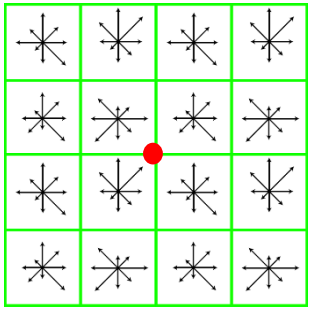


- Rotate sampling window by assigned main gradient direction
- Sample gradient magnitudes and orientations and weight magnitudes using a Gaussian function (blue circle)
- Build descriptor from orientation histograms of gradient magnitudes

Descriptor Computation: Some Facts

1) Window size w for descriptor: 16×16 px





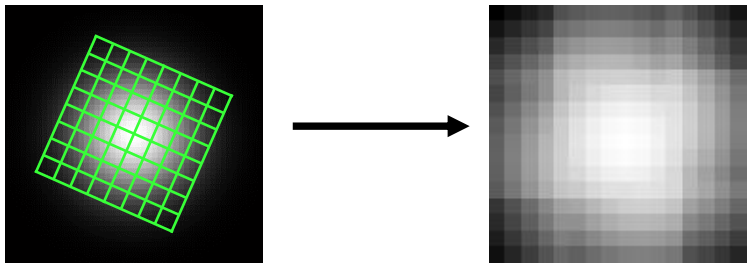
→ 16 histograms

→ Each histogram is build using 4×4 px

2) Gaussian weighting window: $\sigma = 0.5 \cdot w = 8$ → 49×49 filter

→ Filter has to be resampled to size of local window (16×16)

→ Resampling needed wrt position of keypoint



3) Use Gradient magnitudes and orientations from the nearest scale σ of keypoints

Descriptor Computation: Algorithm

- compute the 49×49 Gaussian filter ($\sigma = 8$)
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays x_c, y_c with local image coordinates for a 16×16 neighborhood
    - Rotate local coordinates x_c and y_c using main orientation θ_m
    - Determine nearest scale σ_n
    - Sample Gaussian window, magnitude and angle image at σ_n → G_s, M_s, θ_s
    - Compute weighted gradient magnitude $M_w = G_s .* M_s$
    - Rotate magnitude angles of θ_s by $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers $360/8 = 45$ degrees)
    - Build descriptor vector → rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

Descriptor Computation: Algorithm

- compute the 49×49 Gaussian filter ($\sigma = 8$)
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays x_c, y_c with local image coordinates for a 16×16 neighborhood
    - Rotate local coordinates x_c and y_c using main orientation θ_m
    - Determine nearest scale σ_n
    - Sample Gaussian window, magnitude and angle image at σ_n → G_s, M_s, θ_s
    - Compute weighted gradient magnitude $M_w = G_s .* M_s$
    - Rotate magnitude angles of θ_s by $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers $360/8 = 45$ degrees)
    - Build descriptor vector → rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

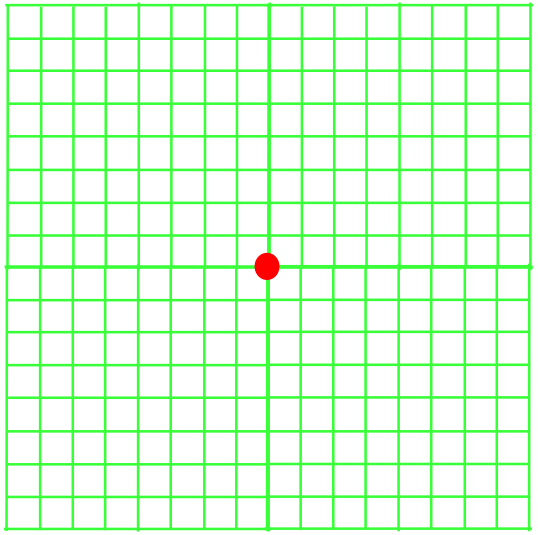
Descriptor Computation 1/5

Local coordinate arrays:

$$x_c = repmat([-\mathbf{7.5}:\mathbf{7.5}], [16,1]);$$

$$y_c = x_c';$$

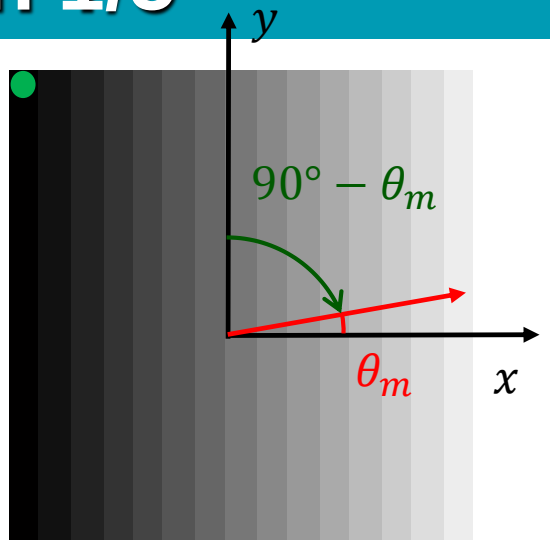
- Center of descriptor window lays on edge → floating values



Descriptor Computation 1/5

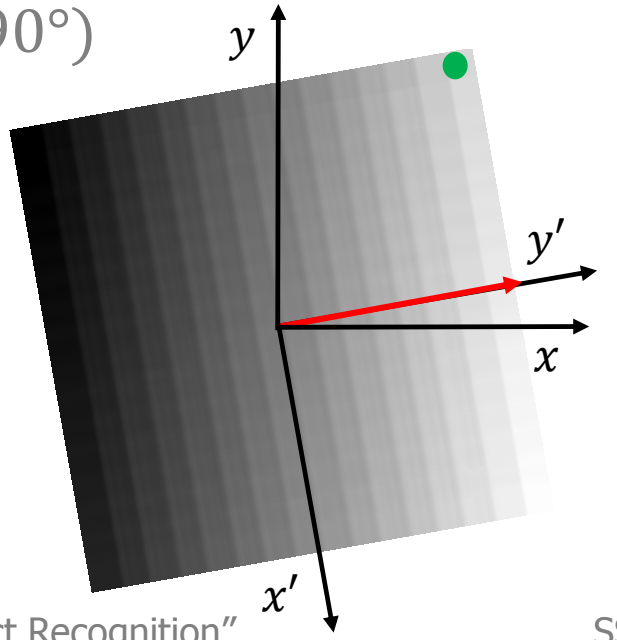
Rotation Example:

- Rotation of local coordinates x_c when $\theta_m = 10°$ is given



→ Clockwise rotation by $90° - \theta_m$ is desired: $\theta' = -(90° - \theta_m) = (\theta_m - 90°)$

$$x_c{}' = \sin(\theta')\, y_c + \cos(\theta')\, x_c$$
$$y_c{}' = \cos(\theta')\, y_c - \sin(\theta')\, x_c$$

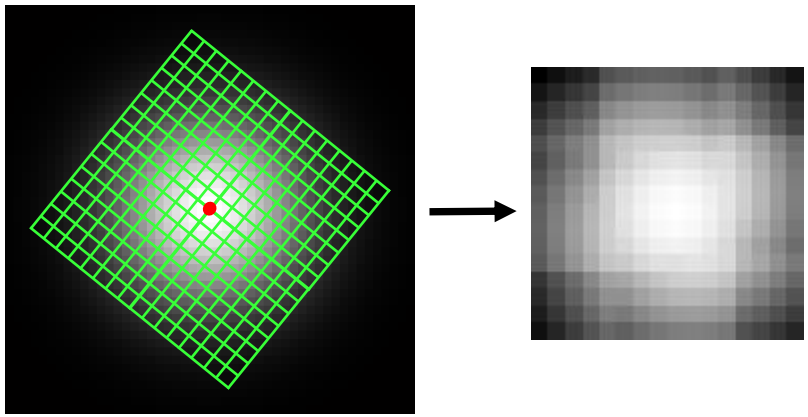


Descriptor Computation: Algorithm

- compute the 49×49 Gaussian filter ($\sigma = 8$)
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays x_c, y_c with local image coordinates for a 16×16 neighborhood
    - Rotate local coordinates x_c and y_c using main orientation θ_m
    - Determine nearest scale σ_n
    - Sample Gaussian window, magnitude and angle image at σ_n → G_s, M_s, θ_s
    - Compute weighted gradient magnitude $M_w = G_s .* M_s$
    - Rotate magnitude angles of θ_s by $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers $360/8 = 45$ degrees)
    - Build descriptor vector → rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

Descriptor Computation 2/5

- **Nearest scale** σ_n: Already available
- **Sampling** of Gaussian Window using x_c' and y_c'
  - Gaussian Window G (49×49)
    - → Translate x_c' and y_c' to center of G
    - → here: $\bar{x}_c' = round(x_c' + 49/2)$;
    - → Sample new values G_s
      - → Use $\bar{x}_c'$and $\bar{y}_c'$ as indices
      - → Use function *sub2ind*



- **Sampling** of magnitude and angle images using x_c' and y_c'
  - Same concept as for Gaussian window
  - Gradient direction and magnitude images according to σ_n
  - Shift local rotated coordinates $\bar{x}_c'$and $\bar{y}_c'$to keypoint position in image → Resampling yields M_s, θ_s

Descriptor Computation: Algorithm

- compute the 49×49 Gaussian filter ($\sigma = 8$)
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays x_c, y_c with local image coordinates for a 16×16 neighborhood
    - Rotate local coordinates x_c and y_c using main orientation θ_m
    - Determine nearest scale σ_n
    - Sample Gaussian window, magnitude and angle image at σ_n → G_s, M_s, θ_s
    - Compute weighted gradient magnitude $M_w = G_s .* M_s$
    - Rotate magnitude angles of θ_s by $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers $360/8 = 45$ degrees)
    - Build descriptor vector → rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

Descriptor Computation 3/5

- **Weighted gradient magnitude**: $M_w = G_s .* M_s$

- **Rotate sampled local gradient directions**
  - Local main orientation θ_m already available

$$\theta_{s,rot} = \theta_s - \theta_m$$

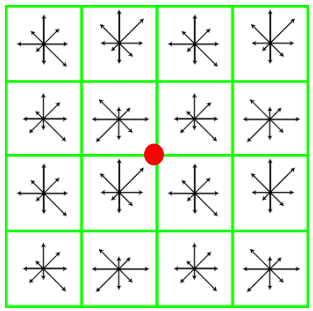
Descriptor Computation: Algorithm

- compute the 49×49 Gaussian filter ($\sigma = 8$)
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays x_c, y_c with local image coordinates for a 16×16 neighborhood
    - Rotate local coordinates x_c and y_c using main orientation θ_m
    - Determine nearest scale σ_n
    - Sample Gaussian window, magnitude and angle image at σ_n → G_s, M_s, θ_s
    - Compute weighted gradient magnitude $M_w = G_s .* M_s$
    - Rotate magnitude angles of θ_s by $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers $360/8 = 45$ degrees)
    - Build descriptor vector → rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

Descriptor Computation 4/5

Histogram building

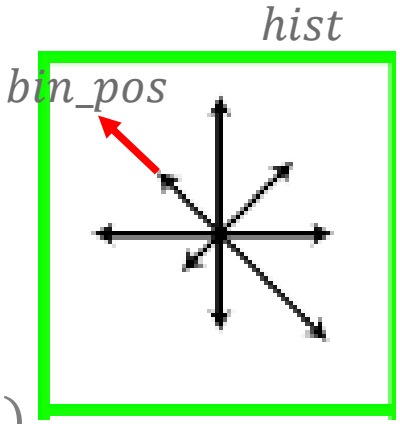
- 16 Histograms of gradient directions
  - → One for each 4×4 px cell of $\theta_{s,rot}$
  - → 45° binsize → 8 bins



- **Use weighted gradient magnitudes** instead of adding 1
  - Usually

$$hist(bin\_pos) = hist(bin\_pos) + 1;$$

    - ↖ has length of 1



  - Here

$$hist(bin\_pos) = hist(bin\_pos) + M_w(row, col)$$

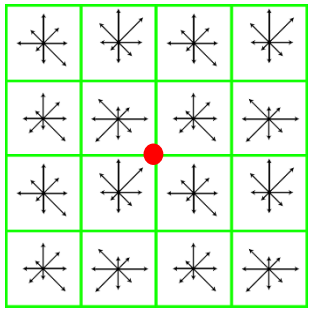
    - ↖ has length of corresponding weighted gradient magnitude $M_w(row, col)$

Descriptor Computation: Algorithm

- compute the 49×49 Gaussian filter ($\sigma = 8$)
- **for** all scales
  - **for** all points in current scale
    - Initialize arrays x_c, y_c with local image coordinates for a 16×16 neighborhood
    - Rotate local coordinates x_c and y_c using main orientation θ_m
    - Determine nearest scale σ_n
    - Sample Gaussian window, magnitude and angle image at σ_n → G_s, M_s, θ_s
    - Compute weighted gradient magnitude $M_w = G_s .* M_s$
    - Rotate magnitude angles of θ_s by $\theta_{s,rot} = \theta_s - \theta_m$
    - Build the 16 histograms with 8 bins (each bin covers $360/8 = 45$ degrees)
    - Build descriptor vector → rearrange values of Histograms (128 bins) to a vector
    - Normalize the vector
  - **end**
- **end**

Descriptor Computation 5/5

- Result of histogram building:
  - 16 histograms with 8 bins → $16 \times 8 = 128$ values



- **Build** a **descriptor** vector d
  - Rearrange values for each keypoint to one vector d of 128 elements
- **Normalize** the descriptor to unit length

$$d = \frac{d}{\|d\|} = \frac{d}{norm(d)}$$

  → Reduces effects of illumination change
- **Crop** all values $d > 0.2$ → set these values to 0.2

  → Reduces non-linear illumination changes
- **Normalize** d again → **finished**

Matlab Code Structure: Main Function

```
% Main function: TestSIFTFeatures
% Here the whole processing chain is done
% Used functions:
%   - CalcSIFTFeatures to do steps 1-7 for two input images
%   - MatchSIFTFeatures to do step 9
%   - VisualizeMatches to do step 10
function TestSIFTFeatures

    % select and import image data
    [file1, path1, image1, img_size1] = read_image('Select file 1 for processing');
    image1 = mat2gray( image1 );
    [file2, path2, image2, img_size2] = read_image('Select file 2 for processing');
    image2 = mat2gray( image2 );

    % calculate SIFT features
    plot = 1;   % 0 --> no plotting of intermediate results; 1 --> plotting activated
    pre_smooth = 1; % 0 --> no prior smoothing; 1 --> prior smoothing (more robustness)
    thresh_contrast = 0.03;%; % the higher, the less points - default: 0.03
    thresh_edges = 6; %the lower, the less points - default: 10

    [Keypoints1, Locations1, image1] = CalcSIFTFeatures(image1, img_size1, plot, pre_smooth, thresh_contrast, thresh_edges);
    [Keypoints2, Locations2, image2] = CalcSIFTFeatures(image2, img_size2, plot, pre_smooth, thresh_contrast, thresh_edges);

    if ~length(Keypoints1(:,1)) | ~length(Keypoints2(:,1))
        return;
    end
    % match points
    thresh_dist = 0.2; % all matches [0,...,sqrt(128)] gt thres_dist will be dismissed
    [matches] = MatchSIFTFeatures(Keypoints1, Keypoints2, thresh_dist);

    % visualize result
    VisualizeMatches(image1, image2, matches, Locations1, Locations2)
end
```

atures

```
function [Keypoints, Locations, Image] = CalcSIFTFeatures(image, img_size, plot, pre_smooth, thresh_contrast, thresh_edges)


    %-------------------------------------------------------------------
    % Calculate scale space and differences of Gaussians
    % caution: pree-smoothing leads to an doubled image size (nn-sampling)
    [DoG_pyramid, Sigmas, Image_pyramid, img_size] = CalcDoGPyramid(n_int, image, sigma_start, sigma_stop, plot, pre_smooth);

    %-------------------------------------------------------------------
    % find initial local maxima and minima in DoG Images
    [Maxima, Minima] = FindLocalExtrema(DoG_pyramid, Sigmas);

    %-------------------------------------------------------------------
    % refine the results

    % at first just do shifting of extrema, if position tells us to do so
    shifting = 1; % therefore, just set shifting to 1
    % this avoids the refinement and just shifts yx positions if necessary
    Maxima = LocalizeContrastEdges(Maxima, DoG_pyramid, Sigmas, thresh_contrast, thresh_edges, shifting);
    Minima = LocalizeContrastEdges(Minima, DoG_pyramid, Sigmas, thresh_contrast, thresh_edges, shifting);

    % now, we can do refinement with new positions (contrast and edge-iness)
    shifting = 0; % no shifting, just filtering out due to contrastt and edge-iness
    Maxima = LocalizeContrastEdges(Maxima, DoG_pyramid, Sigmas, thresh_contrast, thresh_edges, shifting);
    Minima = LocalizeContrastEdges(Minima, DoG_pyramid, Sigmas, thresh_contrast, thresh_edges, shifting);

     %-------------------------------------------------------------------
     % orientation assignment: compute orientations for each scale, where
     % extrema occur
     [Magnitude_Pyramid, Angle_Pyramid] = ComputeOrientations(Image_pyramid, Maxima, Minima);

    %-------------------------------------------------------------------
    % assign orientations
    [Maxima] = AssignOrientations(Magnitude_Pyramid, Angle_Pyramid, Maxima, Sigmas);
    [Minima] = AssignOrientations(Magnitude_Pyramid, Angle_Pyramid, Minima, Sigmas);

    % plot results
    if plot
        title = 'Refined maxima (red) and Minima (green) with orientation and magnitude.';
        PlotExtrema(Maxima, Minima, Image_pyramid{1}, title);
    end
    %-------------------------------------------------------------------

    % We are not longer interested whether the points are max or mins
    % --> concatenation to one array each scale
    for s = 1:length(Maxima)
       Extrema{s} = [Maxima{s}; Minima{s}];
    end

    % descriptor computation
    [Keypoints, Locations] = ComputeDescriptor(Magnitude_Pyramid, Angle_Pyramid, Extrema, Sigmas, img_size);

end
```

Task: Implement Function *ComputeDescriptor*

```
% descriptor computation
[Keypoints, Locations] = ComputeDescriptor(Magnitude_Pyramid, Angle_Pyramid, Extrema, Sigmas, img_size);
```

Provided:

```
%-------------------------------------------------------------------------
% Computation of the final descriptor
% Inputs:
%   - Magnitude_Pyramid: Cell with length(Magnitude_Pyramid) = length(Maxima)
%       --> in each level for each pixel the local gradient magnitudes are stored
%   - Angle_Pyramid: Cell with length(Angle_Pyramid) = length(Maxima)
%       --> in each level for each pixel the local gradient orientations are stored
%   - Maxima: Cell with maxima in each scale step-->array of size (n_points, 5)
%             [y,x,sigma,orientation,magnitude]
%   - Minima: see maxima
%   - Sigmas: vector of size (n_scales) with corresponding values sigma for
%             each scale level of DoG-pyramid
%   - img_size: size of the image [y,x]
% Outputs:
%   - Keypoints: Array with keypoint information for the image
%                Dimensions: [n_keypoints, 128]
%   - Locations: Coordinates of keypoints for the image; in
%                each row of that array (size [n_points, 5]) we have:
%                [y,x,sigma,orientation,magnitude]
% The Keypoits- and the Locations-array will have the same number of rows
% and a row in Keypoits(i) corresponds to a row Locations(i)!
% After computation there will be not longer a separation of minima and
% maxima!
function [Keypoints, Locations] = ComputeDescriptor(Magnitude_Pyramid, Angle_Pyramid, Extrema, Sigmas, img_size)

    Implement desciptor computation here!


end
```


ComputeDescriptor: Inputs

```
% descriptor computation
[Keypoints, Locations] = ComputeDescriptor(Magnitude_Pyramid, Angle_Pyramid, Extrema, Sigmas, img_size);
```

- ***img_size****:* 2-element vector with image size (row,col)
- ***Sigmas***: $1 \times n_s$ vector with σ for each scale of the pyramid
  - → n_s = number of scale space steps
- ***Magnitude_Pyramid***: $1 \times n_s$ cell with gradient magnitudes
  - → *Magnitude_Pyramid{1}* contains an array of *img_size* with gradient magnitudes in scale *Sigmas(1)*
- ***Angle_Pyramid***: $1 \times n_s$ cell with gradient orientations
- ***Extrema***: $1 \times n_s$ cell with keypoints for each scale

ComputeDescriptor: Inputs

```
% descriptor computation
[Keypoints, Locations] = ComputeDescriptor(Magnitude_Pyramid, Angle_Pyramid, Extrema, Sigmas, img_size);
```

Extrema: $1 \times n_s$ cell with keypoints for each scale

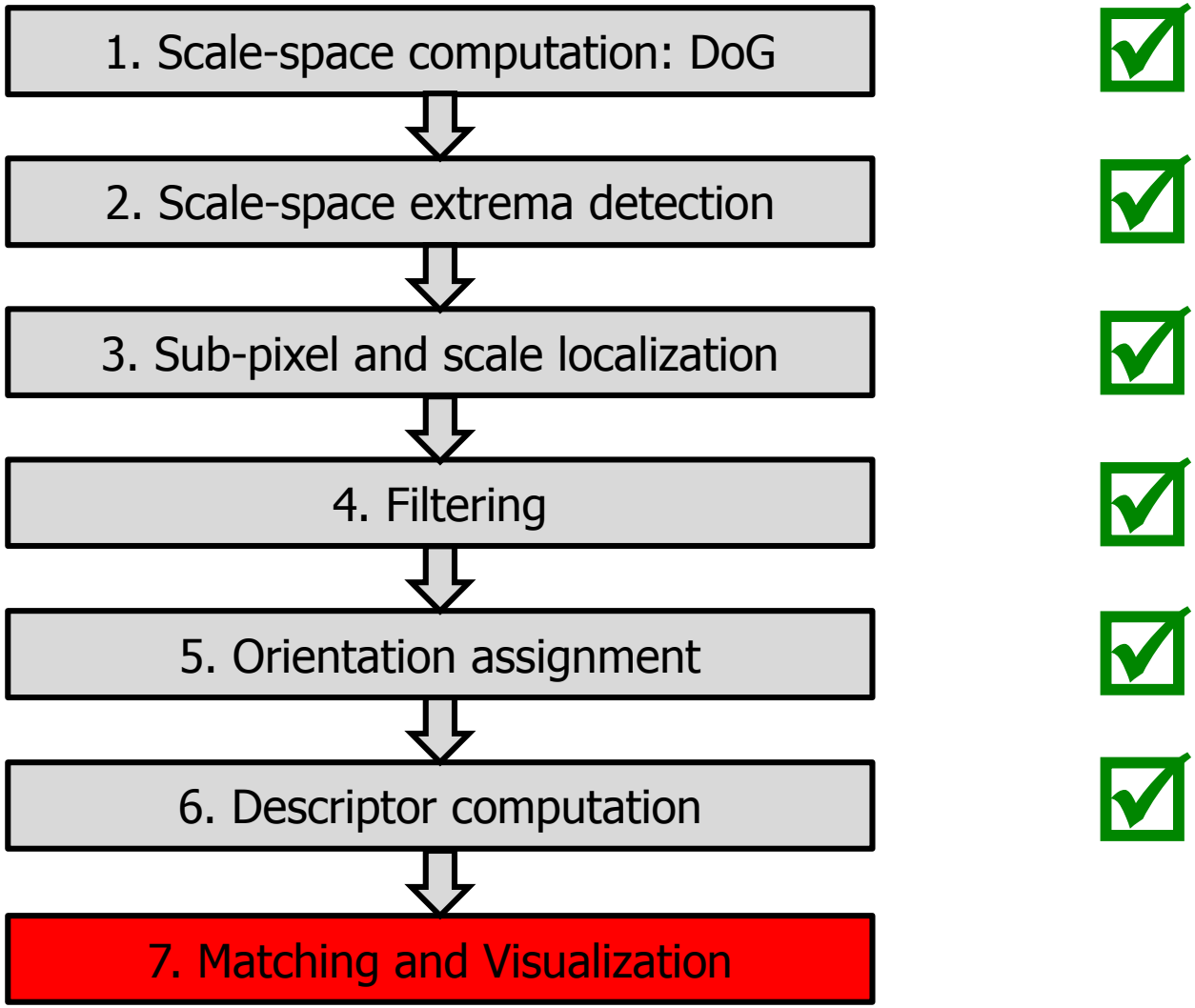
- Get extrema for scale σ: *ex = Extrema{σ};*
  - → *ex* is an array of size $n \times 5$, where n is the number of extrema
  - → If there are no extrema in scale σ, *ex* will be an empty array
  - → Otherwise, each row r of *ex* obtains the following information:
  - → $ex(r,:) = [y, x, \sigma, \theta_m, m_m]$
    - → y, x: subpixel position of keypoint (y → rows, x → columns)
    - → σ: estimated scale of keypoint *r*
    - → θ_m: Main gradient direction
    - → m_m: Main gradient magnitude (not needed in computations, just for visualization)

ComputeDescriptor: Outputs (Your Task)

```
% descriptor computation
[Keypoints, Locations] = ComputeDescriptor(Magnitude_Pyramid, Angle_Pyramid, Extrema, Sigmas, img_size);
```

- ***Keypoints***: array of size $n_p \times 128$ with the descriptor vectors
  - → n_p is the overall number of keypoints (scale-independent!)
  - → Each row represents the descriptor vector of one keypoint
- ***Locations***: array of size $n_p \times 5$ with corresponding information for each keypoint
  - → Store $ex(r,:) = [y, x, \sigma, \theta_m, m_m]$ here (also scale-independent)
  - → Rearrange input values of *Extrema*
- Order of Keypoints and Locations has to be the same!

Algorithm Outline



Matching and Visualization

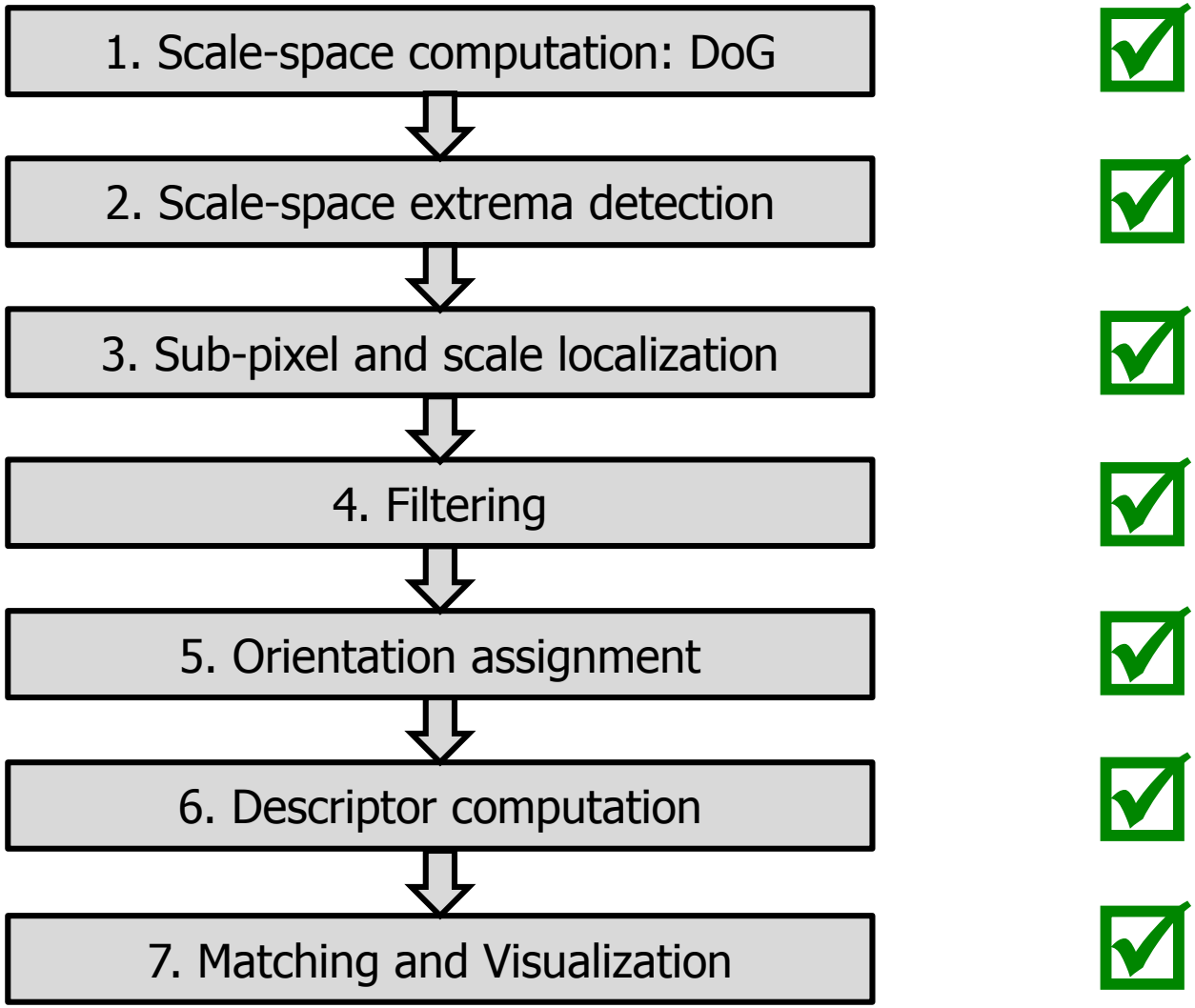
Comparison of all $i = 1, \dots, n_p$ descriptors from image 1 and 2 ($descr_i^1$ and $descr_i^2$)

- Euclidean norm for distance measurement

$$d_i = \sqrt{\sum_{j=0,\dots,128} \left(descr_{i,j}^1 - descr_{i,j}^2\right)^2}$$
$$= norm(descr_{i,j}^1 - descr_{i,j}^2)$$

- Eliminate double matches

Algorithm Outline



Example

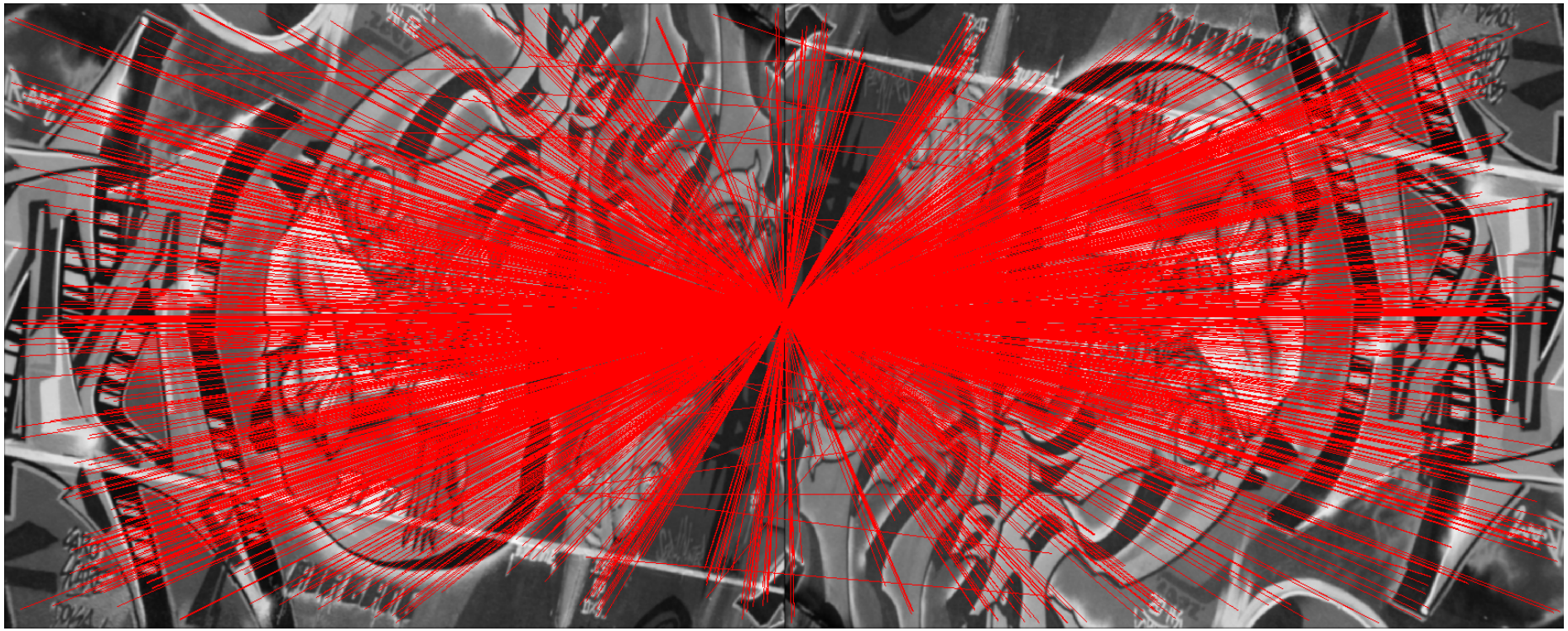
- Input Images

Input1.jpg

Input2.jpg

Example

- Desired Result



- Tipp: Use subimages for testing (runtime)

Provided Data (See CV-Page)

- **Matlab** source: *TestSIFTFeatures.m*
- **Octave** source: *TestSIFTFeatures_octave.m*
- Full-size images: *Input1.jpg, Input2.jpg*

- Small-size images: *Input1_s.jpg, Input2_s.jpg*

Thank you!