Local Features

CSE 4310 – Computer Vision
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

What is a Feature?

- Any information extracted from an image.
- The number of possible features we can define is infinite.
- Any function F we can define that takes in an image as an argument and produces one or more numbers as output defines a feature.
  - Correlation with a template defines a feature.
  - Projecting to PCA space defines features.
  - More examples?

What is a Feature?

- Any information extracted from an image.
- The number of possible features we can define is infinite.
- Any function F we can define that takes in an image as an argument and produces one or more numbers as output defines a feature.
  - Correlation with a template defines a feature.
  - Projecting to PCA space defines features.
  - Average intensity.
  - Std of values in window.

Types of Features

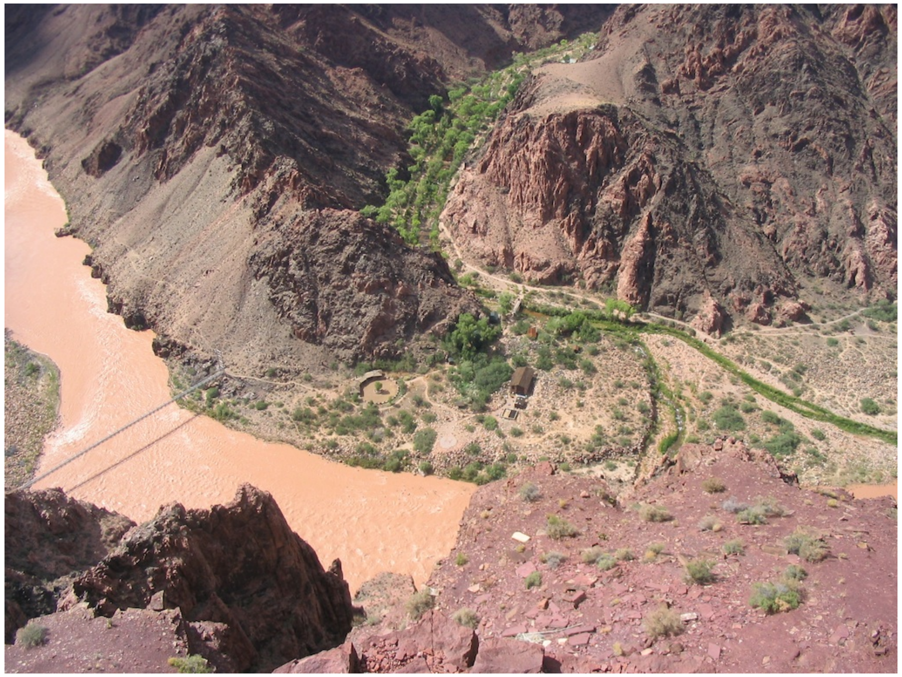
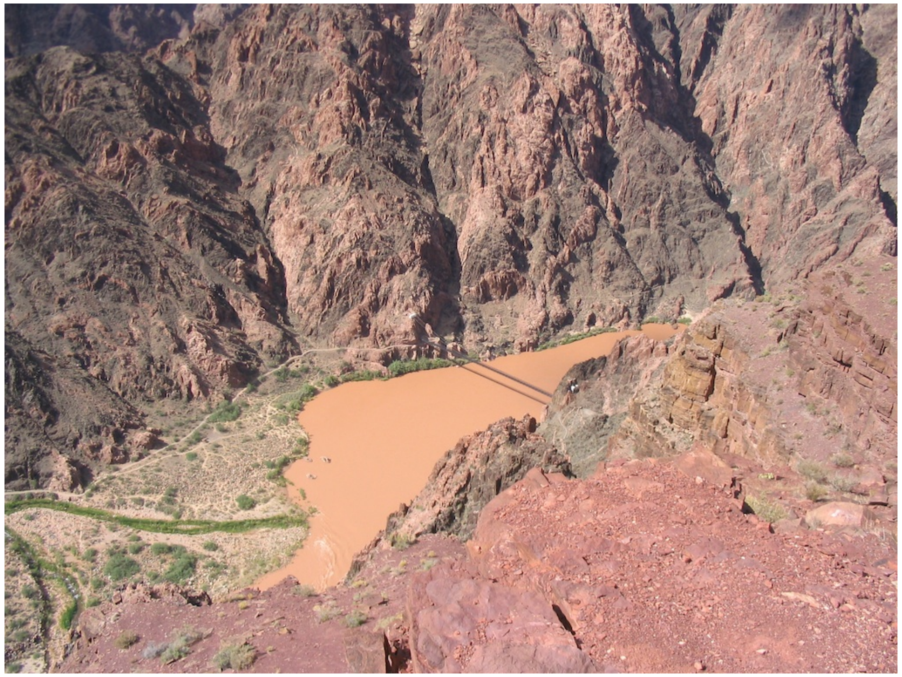
- Global features: extracted from the entire image.
  - Examples: template (the image itself), PCA projection.
- Region-based features: extracted from a subwindow of the image.
  - Any global feature can become a region-based feature by applying it to a specific image region instead of applying it to the entire image.
- Local features: describe a pixel, and the vicinity (nearby region) around a specific pixel.
  - If they describe a "vicinity", which is a region, one could argue that they are regional features.
  - The main difference is that a local feature always refers to a specific pixel location. This is not a well-defined or mathematical difference, but it will become clearer when we see examples of local features.

Uses of Features

- Features can be used for any computer vision problem.
  - Detection.
  - Recognition.
  - Tracking.
  - Stereo estimation.
- Different types of features work better for different problems, and for different assumptions about the images.
  - That is why there are many different types of features that people use.

Example: Pixel Correspondences

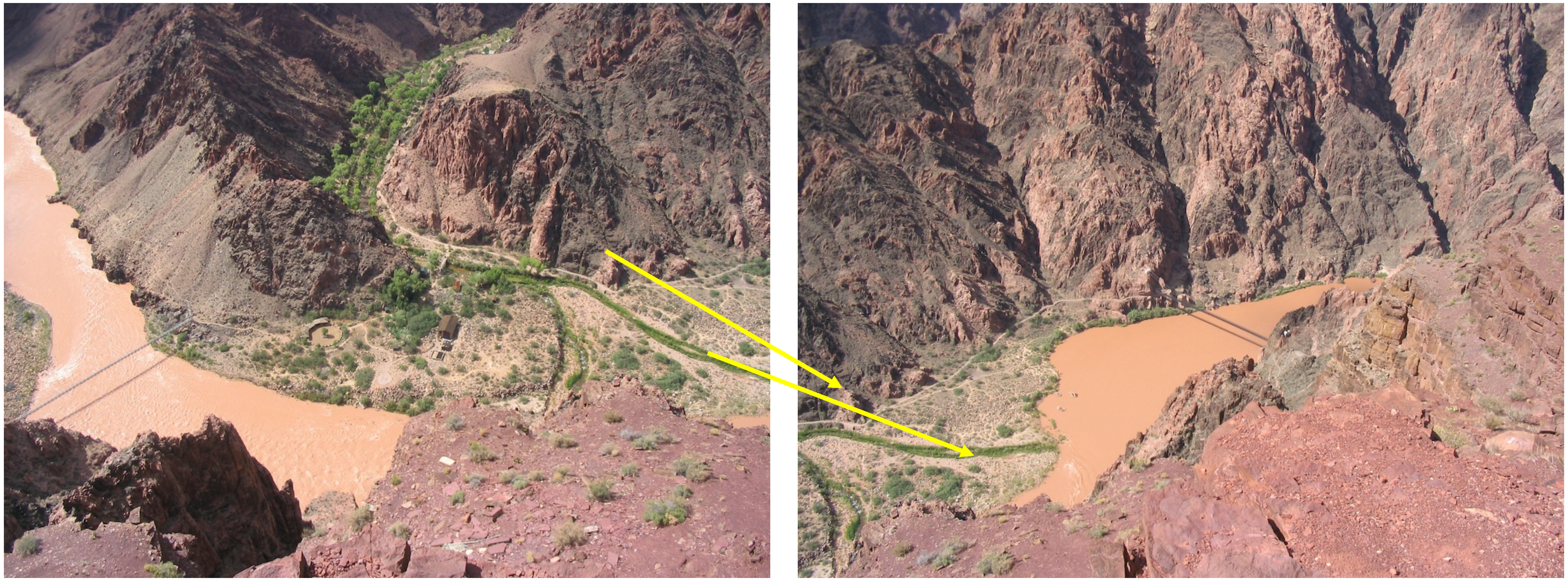
- Are there locations that are visible in both images below?
- If so, which pixels correspond to each other?
  - Even for a human, the answers are not obvious.
  - Do you see any matching regions?

Credit: images downloaded from https://www.cs.ubc.ca/~lowe/keypoints/

Example: Pixel Correspondences

- Are there locations that are visible in both images below?
- If so, which pixels correspond to each other?
  - Even for a human, the answers are not obvious.
  - Do you see any matching regions?



Credit: images downloaded from https://www.cs.ubc.ca/~lowe/keypoints/

Example: Pixel Correspondences

- Finding correspondences between two images can have different uses. A couple of examples:
  - 3D estimation using stereo.
  - Pose estimation: estimating the 3D orientation of an object.
  - Image stitching: Combining multiple overlapping images into a single image.

Credit: images downloaded from https://www.cs.ubc.ca/~lowe/keypoints/

Finding Correspondences

- A first approach for finding pixel correspondences:
  - For each pixel in one image:
    - Compute the feature vector corresponding to that pixel.
    - Find the best match for that feature vector in the other image.
    - If the matching score is better than a threshold, consider it a "match".

Finding Correspondences

- A first approach for finding pixel correspondences:
  - For each pixel in one image:
    - Compute the **feature vector** corresponding to that pixel.
    - Find the **best match** for that feature vector in the other image.
    - If the matching score is better than a threshold, consider it a "match".
- Black boxes that we need to specify:
  - How do we define feature vectors? We will study several methods.
  - How do we compute matching scores between two vectors?

Local Feature: Color

- The most “local” of all local features is just the color of the pixel.
- The feature vector F(i,j) is just the RGB values of the image at pixel (i,j).
- Usually, this is not very useful.
  - Typically, for any pixel in one image, there will be several pixels matching its color in the other image.
- This is too “local” to be useful.
  - A single pixel does not contain much information.

Local Feature: Image Window

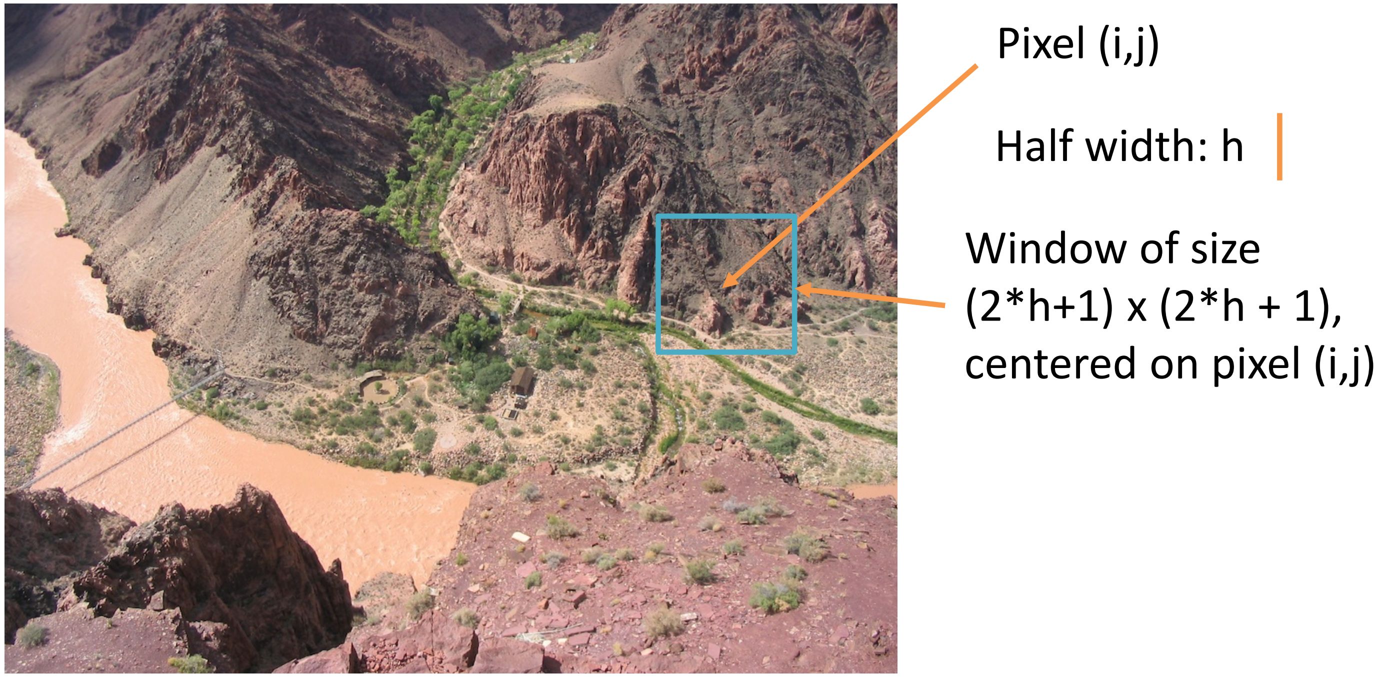
- We can simply extract a window around the pixel.
- We can define a window “half width” h.
- Then, given an image G and a pixel location (i,j):

Pixel (i,j)

Half width: h

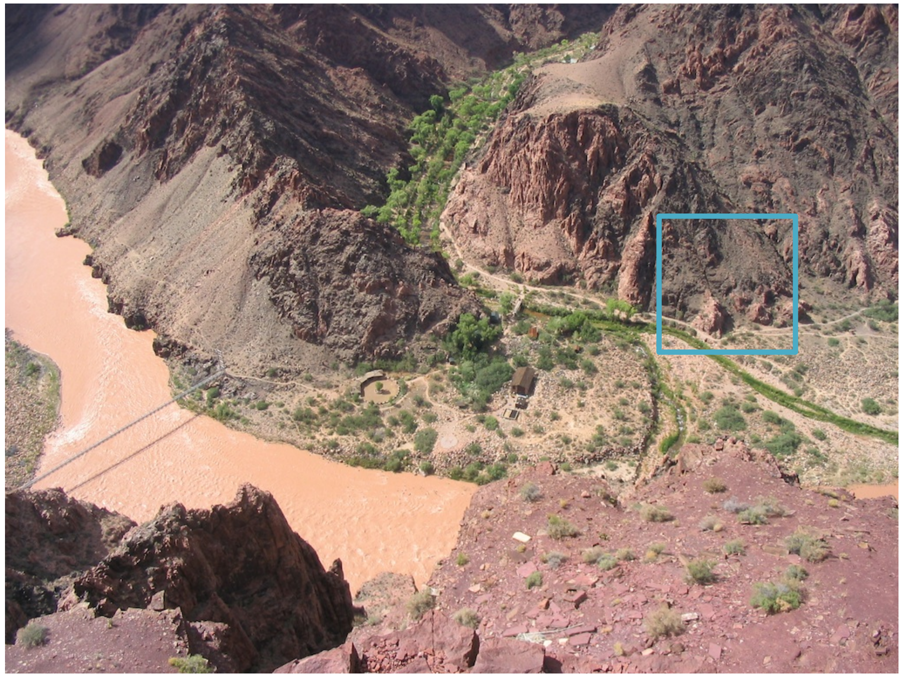
Local Feature: Image Window

- We can simply extract a window around the pixel.
- We must choose a window “half width” h.
- Then, given an image G and a pixel location (i,j):
  F(i,j) = G((i-h):(i+h),(j-h,j+h))
- F(i,j) can be vectorized to a vector of dimensions $(2h+1)^2$.



Local Feature: Image Window

- We want to find the best match of F(i,j) in the second image.
- How? We can use any variation of template matching.
  - Sum-of-squared differences, with or without: brightness and contrast normalizations, multiscale search, multiple orientation search.
  - Obviously, each variation will give different results.



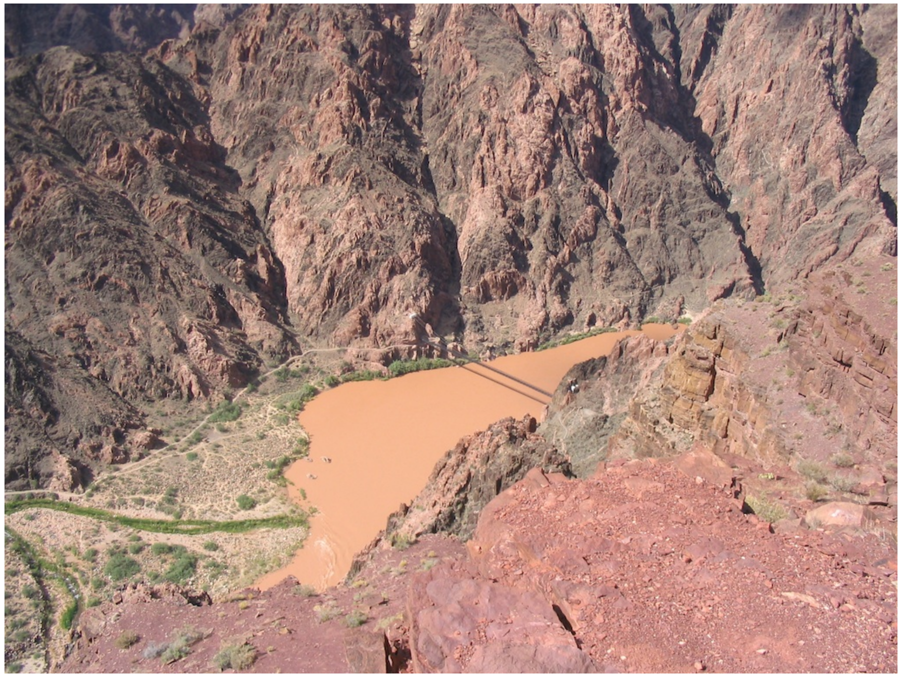
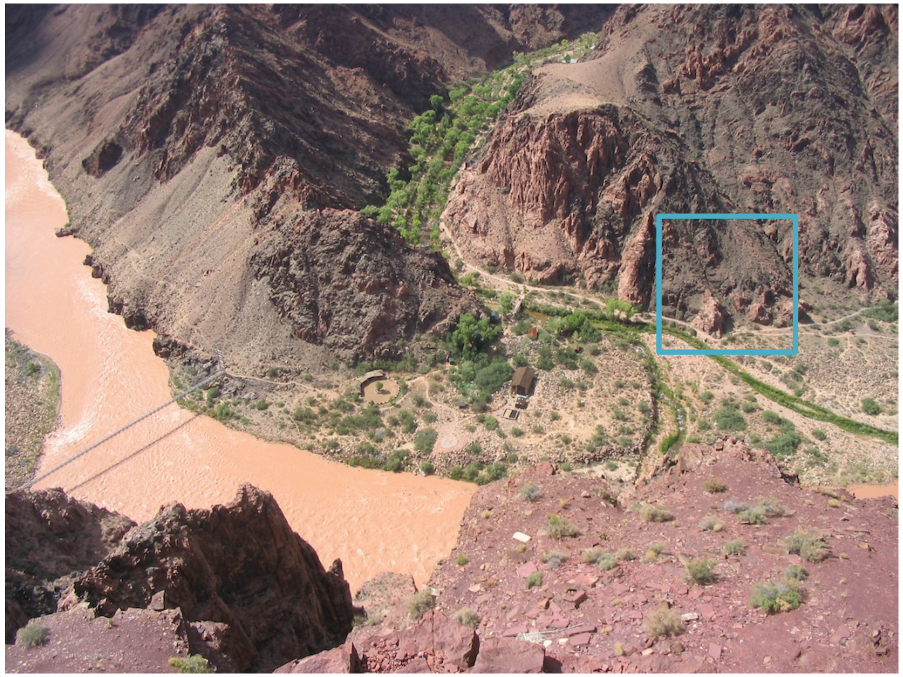
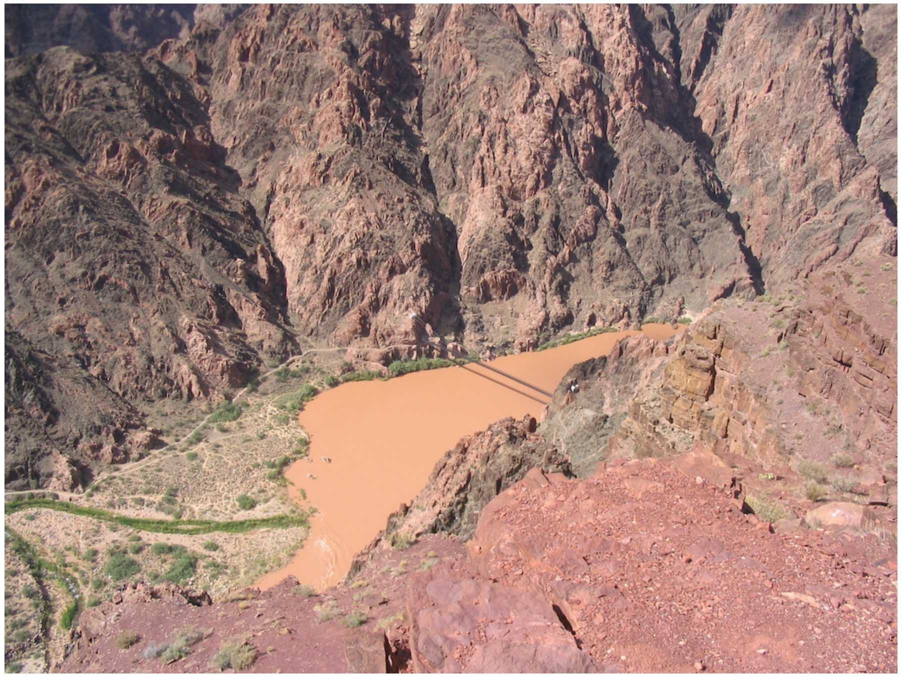


Image Regions as Features: Drawbacks

- Using image regions as features is easy to implement, but:
  - Changes in 3D orientation lead to bad matching scores.
  - Searching the second image at all locations, multiple scales, multiple orientations, can be slow, especially if repeat this process for thousands of templates extracted from different pixels in the first image.





Shape Context

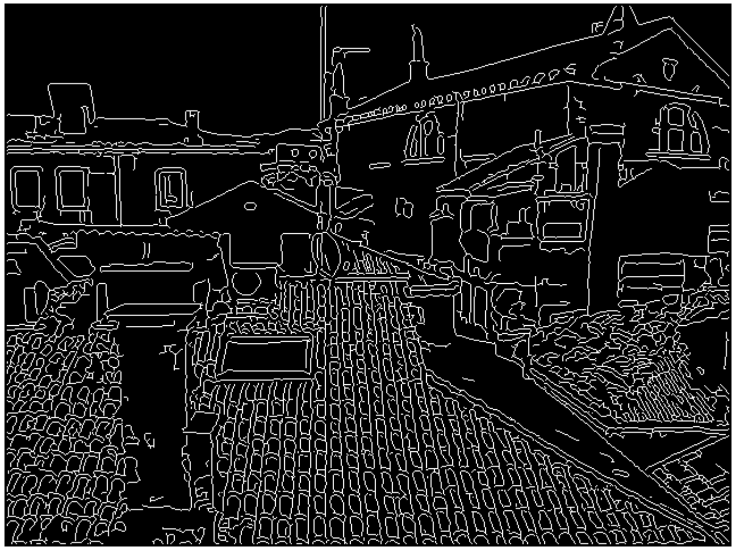
- Another type of local feature is **shape context**.
  - Reference: Belongie, Serge, Jitendra Malik, and Jan Puzicha. "Shape matching and object recognition using shape contexts." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 4 (2002): 509-522.
- Key idea:
  - Compute edge image (for example, using Canny detector).
  - Given an edge pixel (i, j), compute a histogram describing the locations of nearby edge pixels.

Shape Context

- First step: detect edges.
  - You decide what method and what parameters to use.

Credit: downloaded from
https://www.cs.ubc.ca/~lowe/keypoints/

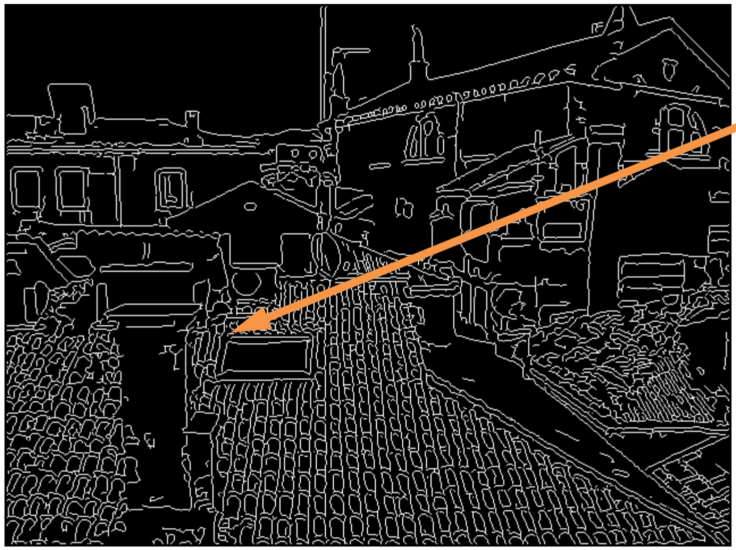
Image **`roofs1`**


Result of **`canny(roofs1,5)`**

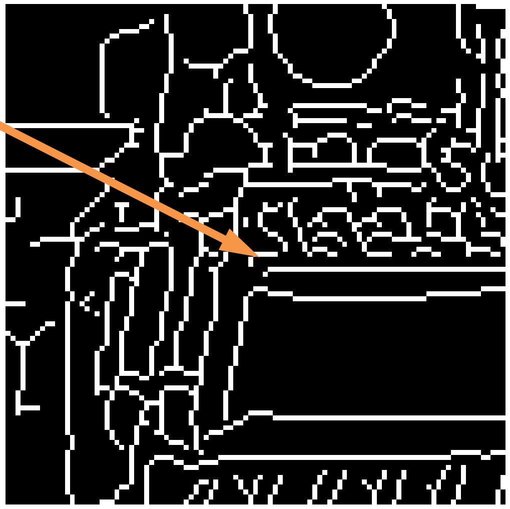
Shape Context

- Second step: given a pixel location (i,j), build the histogram of edge locations.
- Example: we pick location (291, 198).
  - i is row index, j is column index.

Pixel (291, 198)



Result of `canny(roofs1,5)`

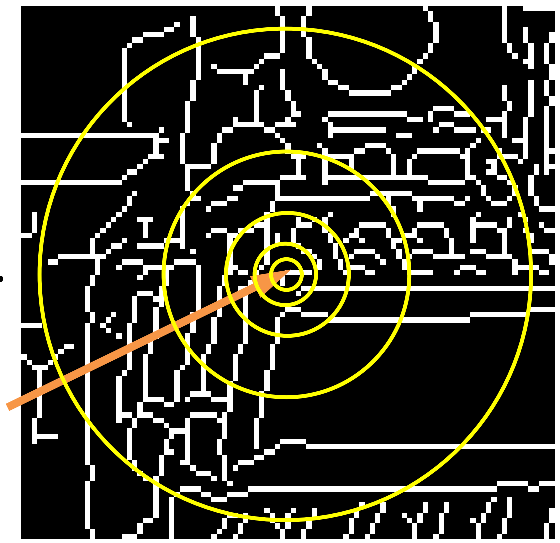


Zoom of 101x101 window centered at pixel (291, 198).

Shape Context - Histogram

- We divide the space around the selected pixel (291, 198) into 5 bins, based on distances to the selected pixel.
  - The innermost bin B_1 covers radius r_1 around the selected pixel.
    - r_1 is a parameter we choose.
  - For m = 2, 3, 4, 5, define radius $r_m = 2 * r_{m-1}$
  - For m = 2, 3, 4, 5, bin B_i covers the region between r_{m-1} and r_m.

Zoom of 101x101 window centered at pixel (291, 198).

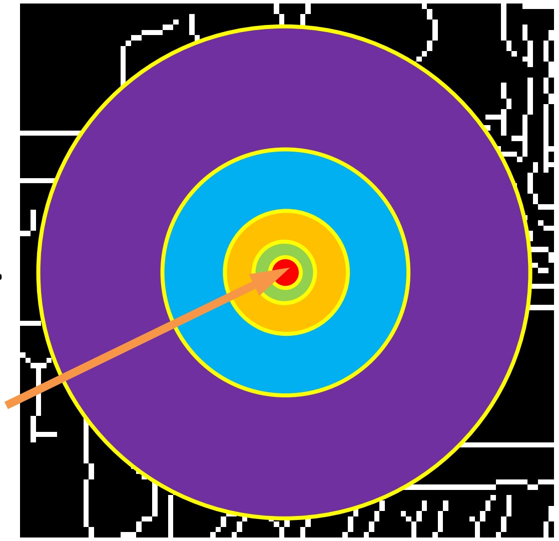


Shape Context - Histogram

- We divide the space around the selected pixel (291, 198) into 5 bins, based on distances to the selected pixel.
  - The innermost bin B_1 covers radius r_1 around the selected pixel.
    - r_1 is a parameter we choose.
  - For m = 2, 3, 4, 5, define radius $r_m = 2 * r_{m-1}$
  - For m = 2, 3, 4, 5, bin B_i covers the region between r_{m-1} and r_m.
- Visualization:
  - Red: region of B_1.
  - Green: region of B_2.
  - Orange: region of B_3.
  - Blue: region of B_4.
  - Purple: region of B_5.

Zoom of 101x101 window centered at pixel (291, 198).

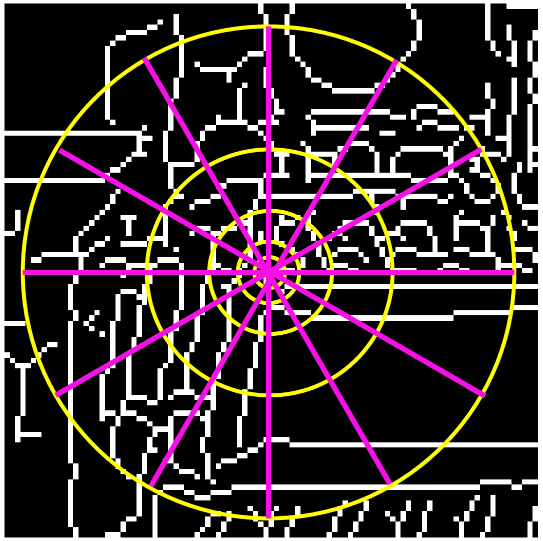
Pixel (291, 198)



Shape Context - Histogram

- We also divide the space within radius r_5 from the selected location in a different way, pizza-slice style.
  - We divide this space into 12 bins C_n, based on orientation of the line connecting each pixel to the selected location.
  - Each bin C_n includes pixels (i', j') such that the line connecting (i', j') to the center has orientation between 30*(n-1) degrees and 30*n degrees.

Zoom of 101x101 window centered at pixel (291, 198).

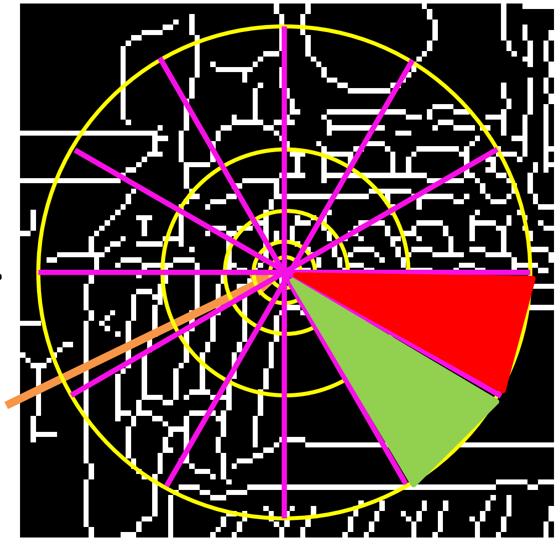


Shape Context - Histogram

- We also divide the space within radius r_5 from the selected location in a different way, pizza-slice style.
  - We divide this space into 12 bins C_n, based on orientation of the line connecting each pixel to the selected location.
  - Each bin C_n includes pixels (i', j') such that the line connecting (i', j') to the center has orientation between 30*(n-1) degrees and 30*n degrees.
- Example:
  - C_1, shown in red:
    - [0, 30) degrees.
  - C_2, shown in green:
    - [30, 60) degrees.

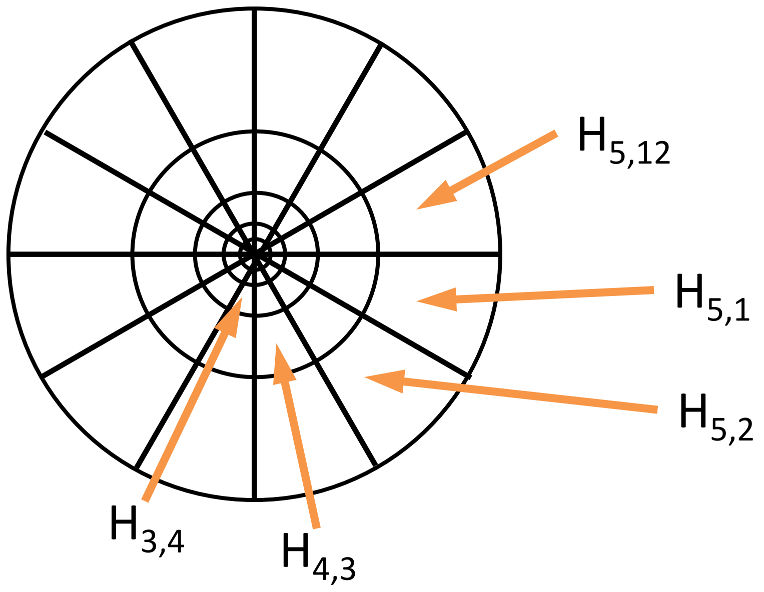
Zoom of 101x101 window centered at pixel (291, 198).

Pixel (291, 198)



Shape Context - Histogram

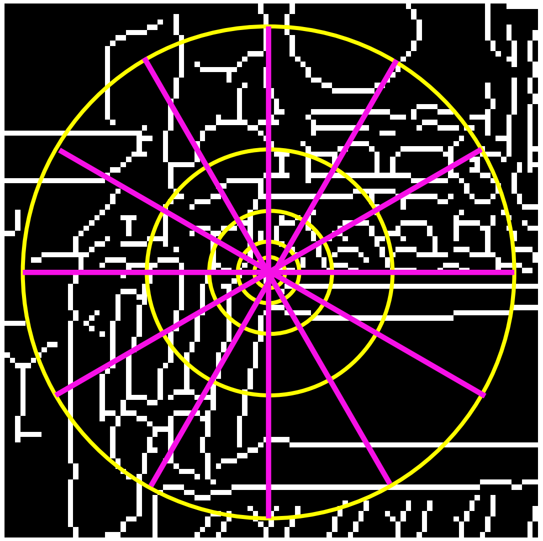
- We define a 60-bin histogram H, where:
  - Each bin $H_{m,n}$ is the intersection between bin B_m and bin C_n.
- So, effectively we divide the area within distance r_5 from the central pixel into 60 regions, each corresponding to a bin $H_{m,n}$.



Shape Context - Histogram

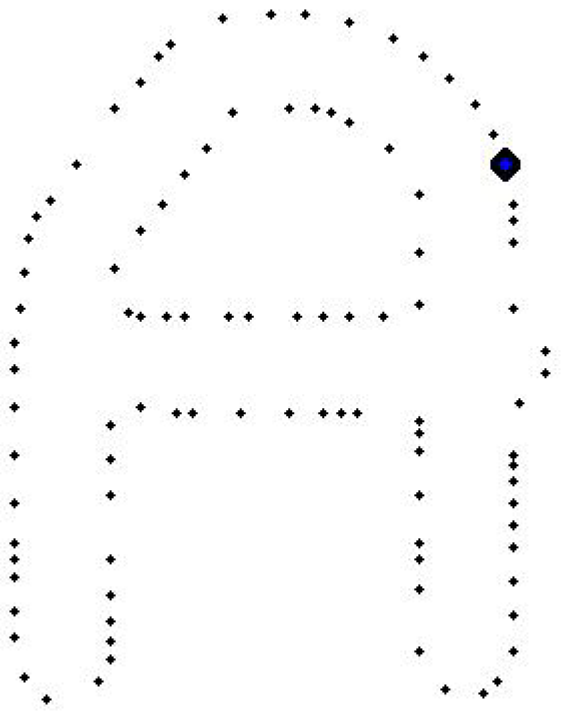
- We define a 60-bin histogram H, where:
  - Each bin $H_{m,n}$ is the intersection between bin B_m and bin C_n.
- So, effectively we divide the area within distance r_5 from the central pixel into 60 regions, each corresponding to a bin $H_{m,n}$.
- The value we store at $H_{m,n}$ is the number of edge pixels in the corresponding region.
- This histogram H is the **shape context feature** for pixel (291, 198).

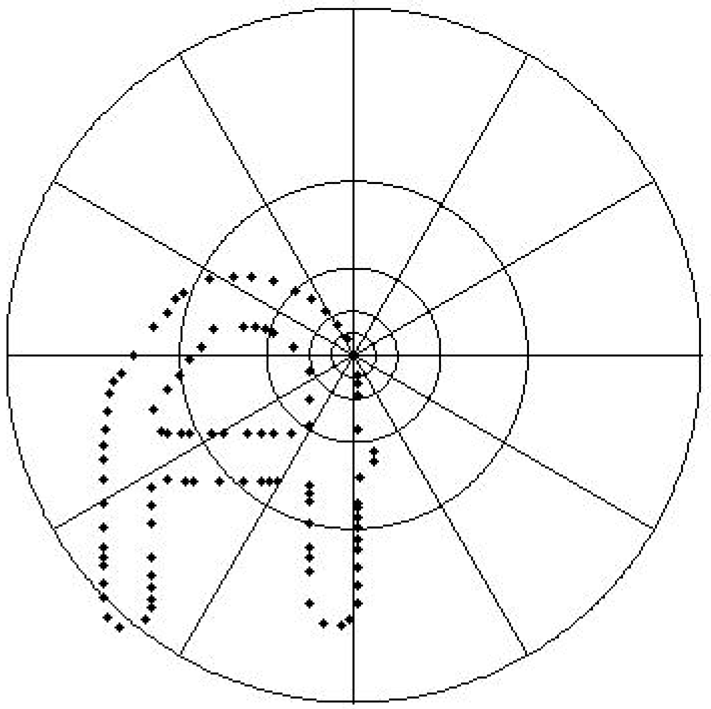
Zoom of 101x101 window centered at pixel (291, 198).



Shape Context - Histogram

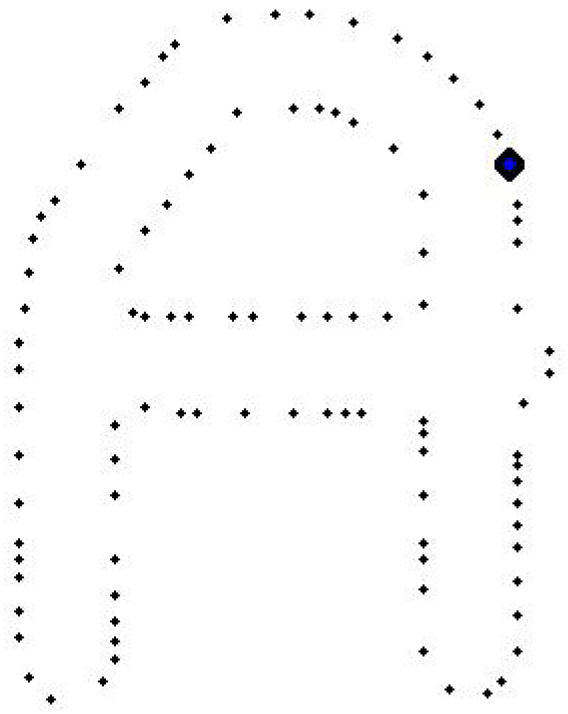
- Here is an image from the Wikipedia article on Shape Context, that helps visualize the definition.

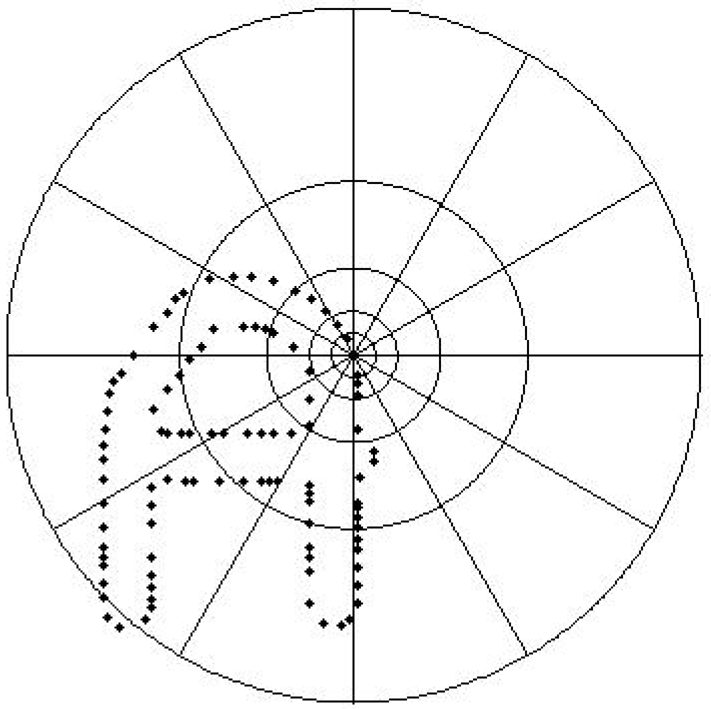




Shape Context - Histogram

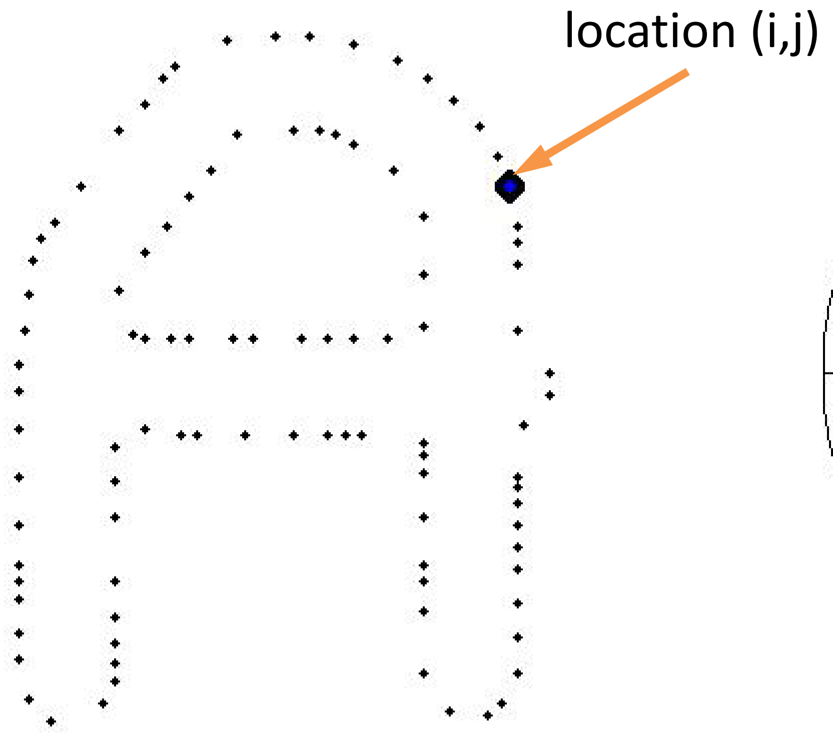
- On the left, you see the edge pixel locations for some image.

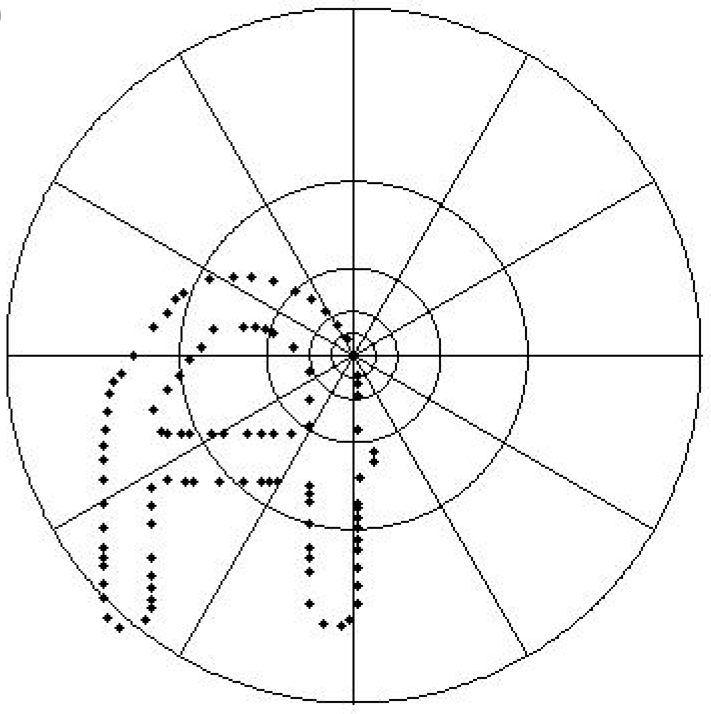




Shape Context - Histogram

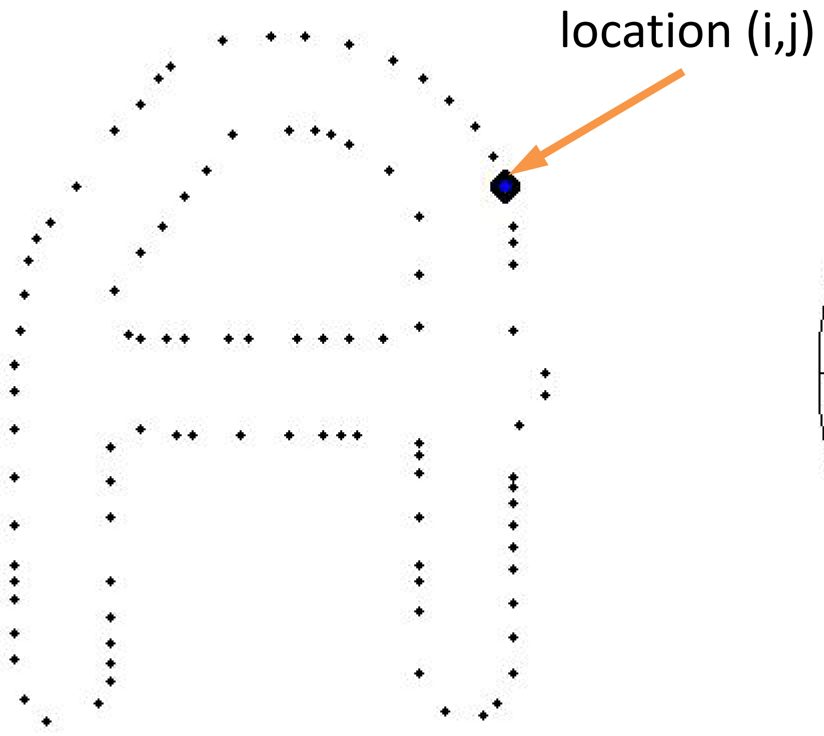
- We pick a location (i,j), for which we want to compute the shape context feature vector.

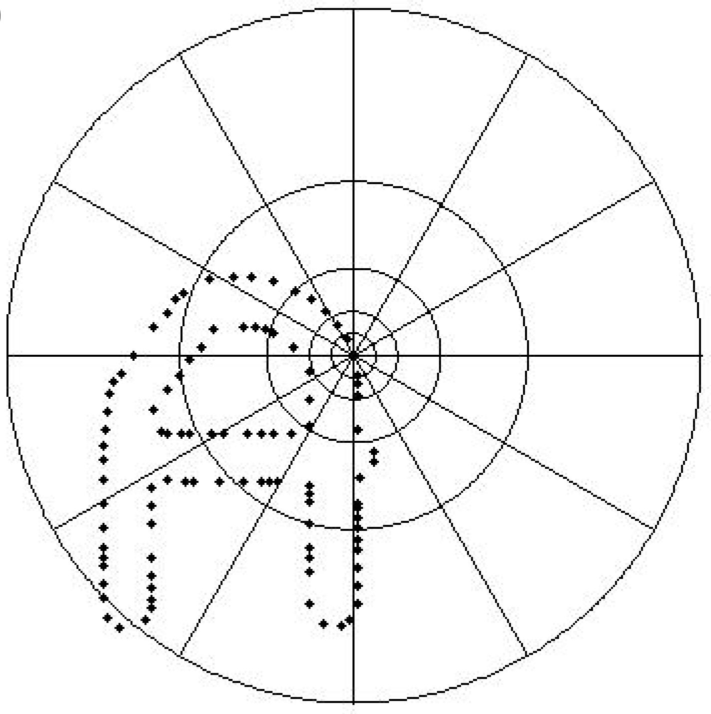




Shape Context - Histogram

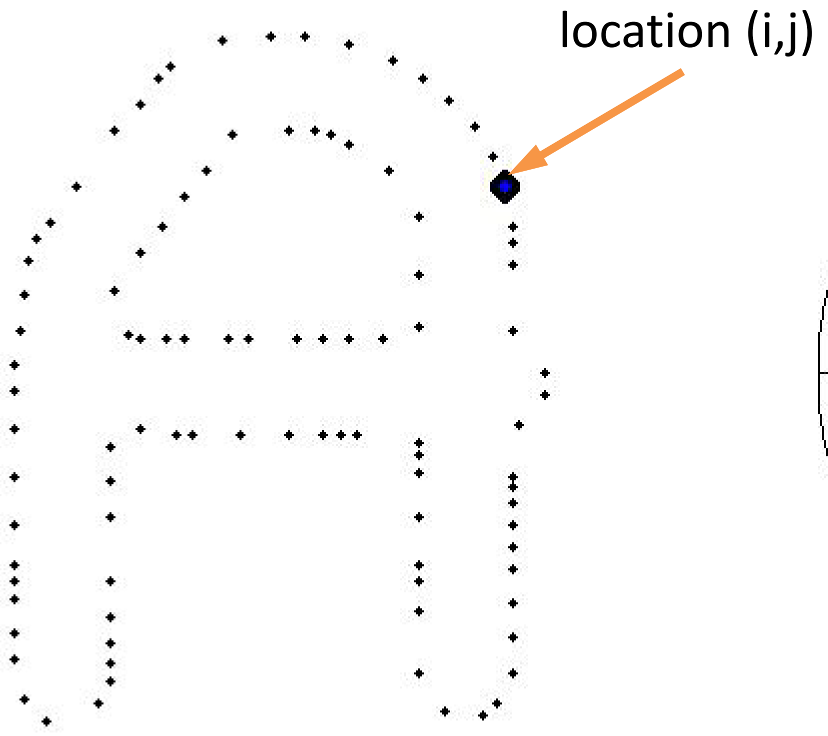
- Each bin $H_{m,n}$ in the shape context histogram corresponds to a cell in the log-polar grid.
  - We store in $H_{m,n}$ the number of edge pixels that fall in the corresponding cell.

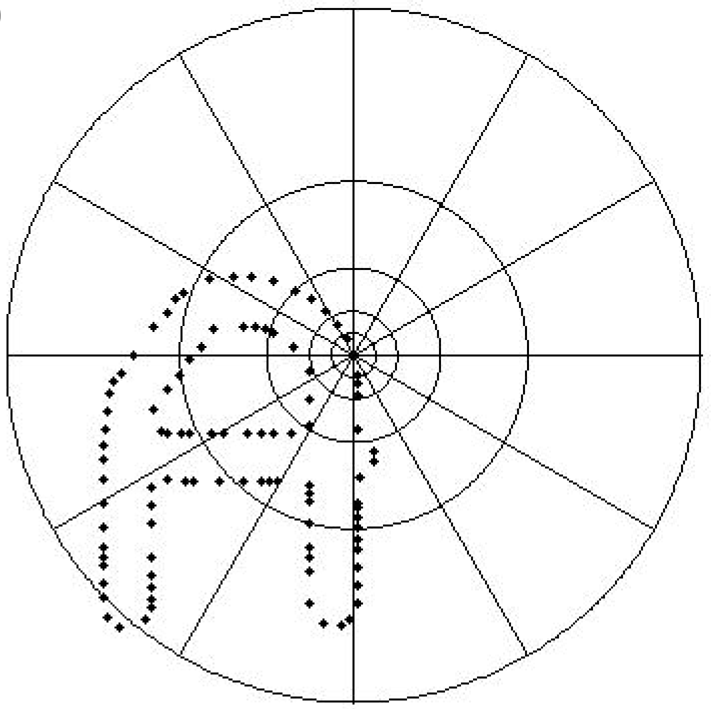




Shape Context - Histogram

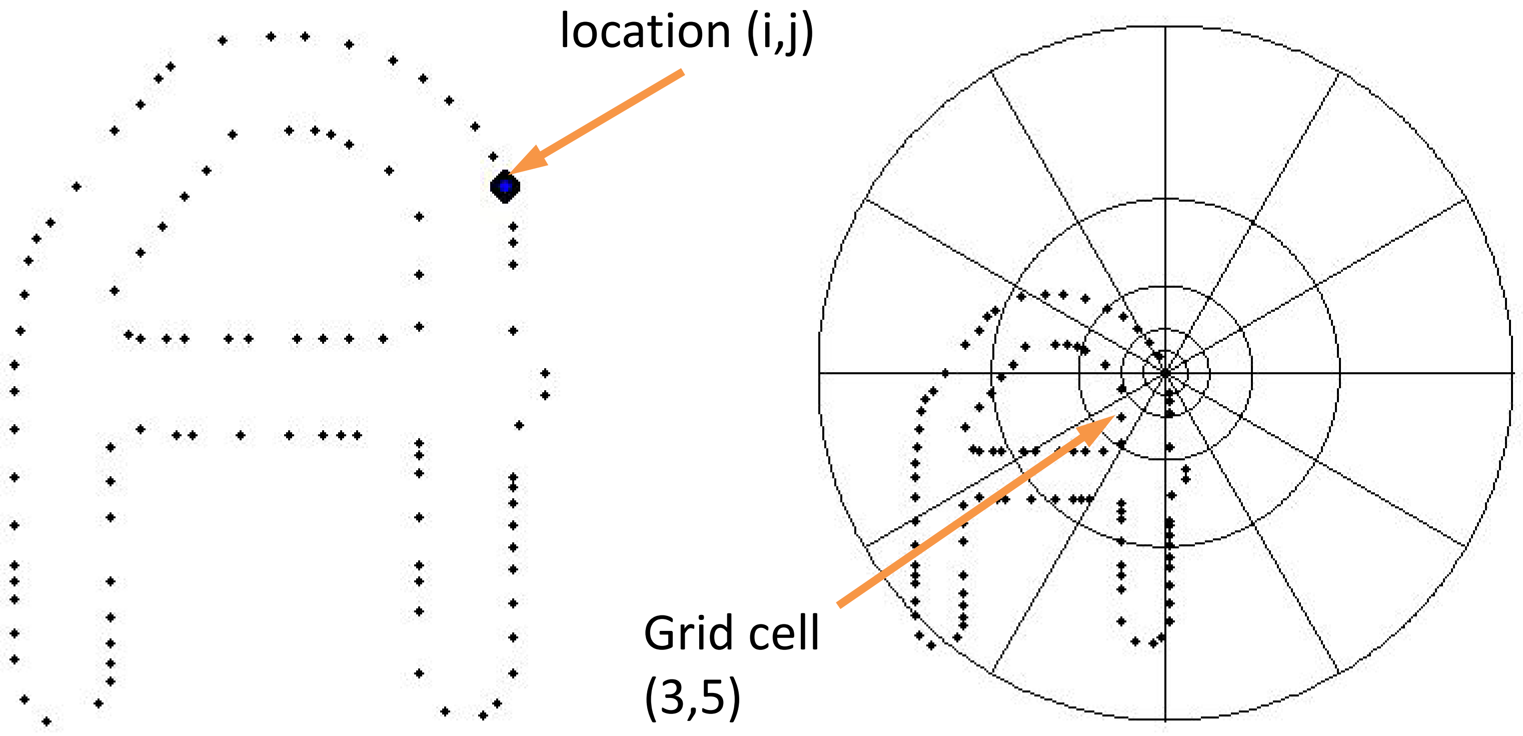
- For example, what value do we store in bin $H_{3,5}$?
  - Which cell corresponds to bin $H_{3,5}$?





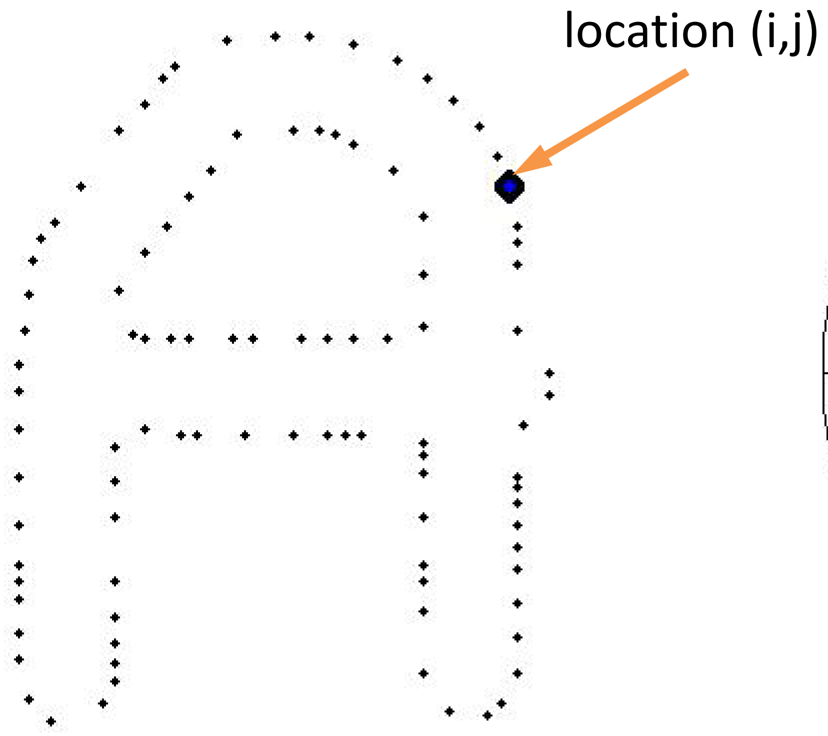
Shape Context - Histogram

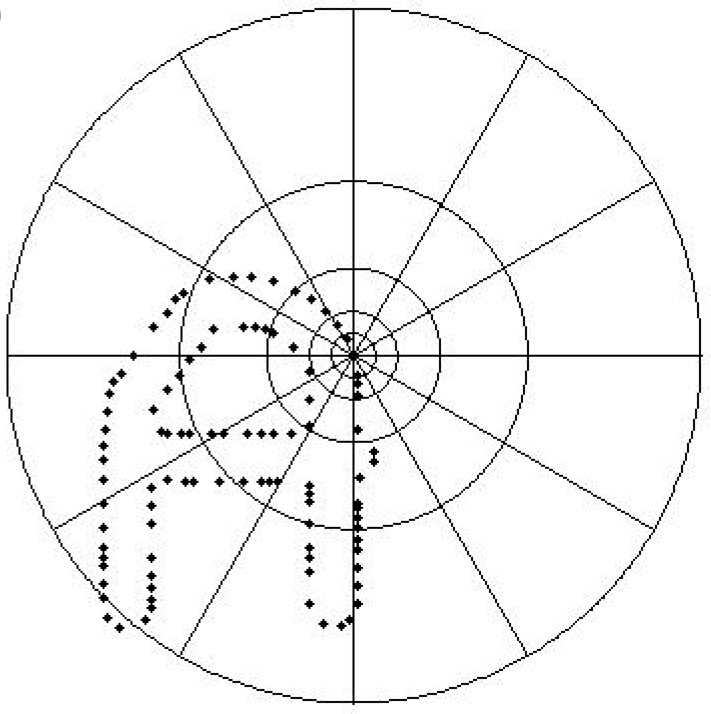
- For example, what value do we store in bin $H_{3,5}$?
  - Corresponds to third innermost ring, fifth pizza slice.
  - There are two edge pixels there, so $H_{3,5} = 2$.



Shape Context - Histogram

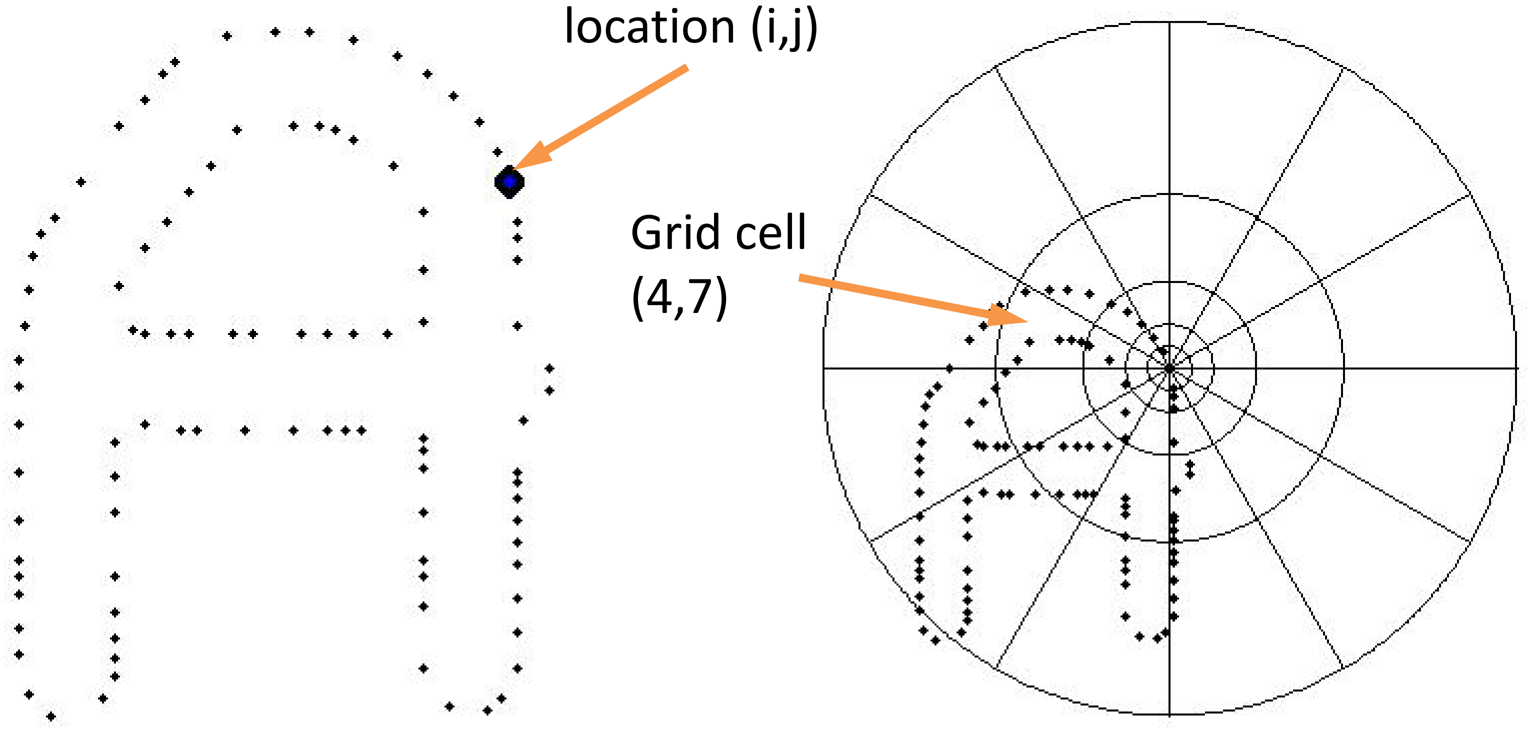
- Second example: what value do we store in bin $H_{4,7}$?
  - Which cell corresponds to bin $H_{4,7}$?





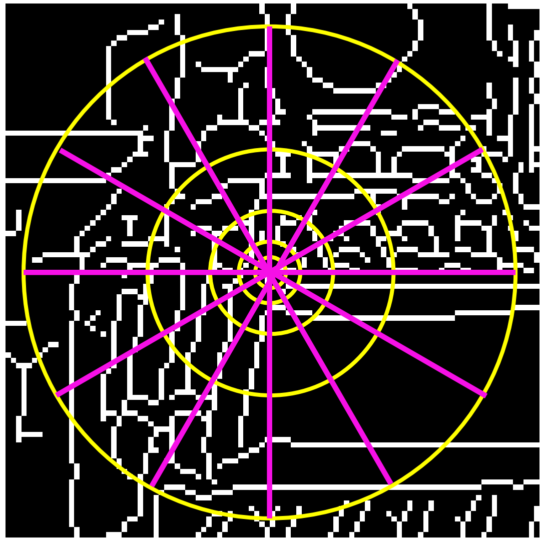
Shape Context - Histogram

- Second example: what value do we store in bin $H_{4,7}$?
  - Corresponds to fourth innermost ring, seventh pizza slice.
  - There are six edge pixels there, so $H_{4,7} = 6$.



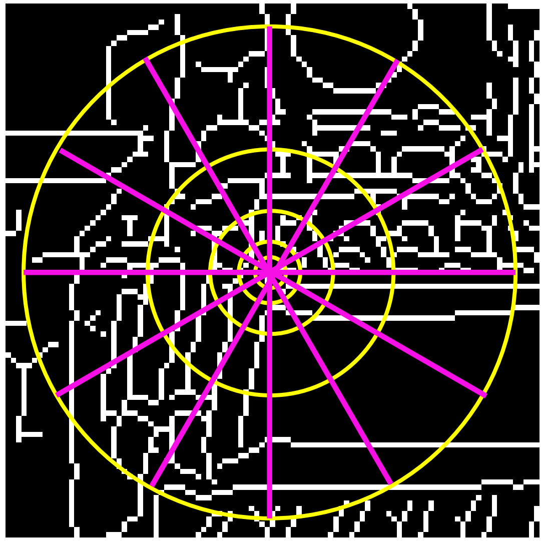
Shape Context - Histogram

- We denote by H(E, i, j) the shape context histogram that we extract from pixel location (i,j) of edge image E.
  - We denote by $H_{m,n}(E, i, j)$ the value that we store in bin (m,n) of histogram H(E, i, j).
  - m specifies the circular ring, n specifies the pizza slice.
- For any location (i,j) of image E, we can repeat the same process to obtain a shape context histogram.
- The shape context histogram can be vectorized to a 60-dimensional vector.



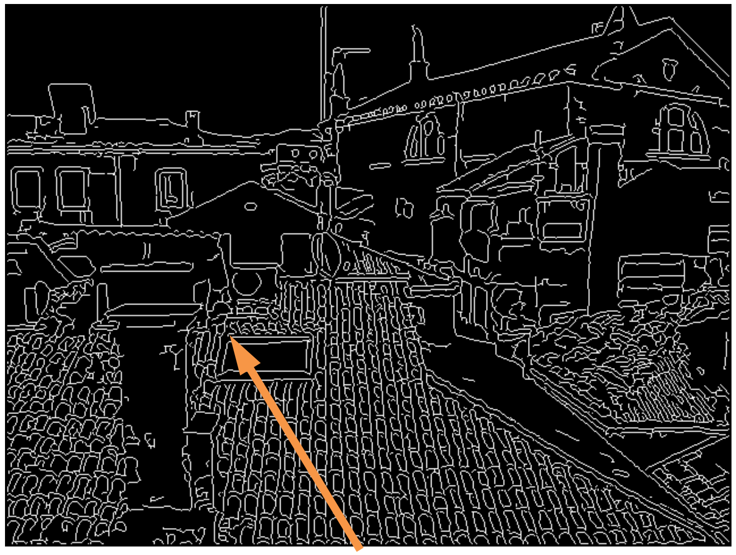
Shape Context - Histogram

- How many shape context histograms can we extract from an image?
  - In principle, we can extract one histogram for each pixel.
  - However, this is usually unnecessary, and leads to too much computational time.
- We usually compute shape context features only at edge pixel locations.
  - We can use all edge pixels.
  - We can randomly select some edge pixels.
  - We can select some edge pixels in a deterministic way (more details in a few slides, when we talk about interest point detection).

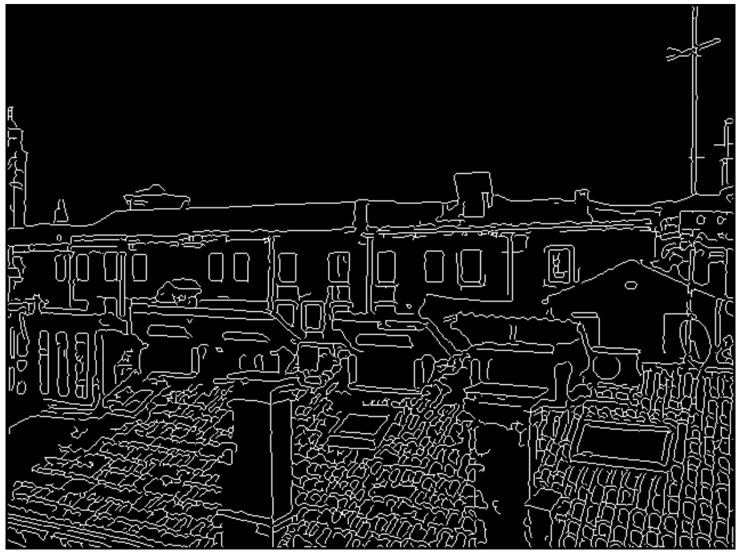


Finding Matches

- Suppose we extracted the shape context feature vector $H(E_1, 291, 198)$ for pixel location 291, 198 on the left image.
- Now we want to find the best match on the right image.
- To do that, first need to compute shape context feature vectors for pixel locations on the right image.



Pixel (291, 198) of image E_1.



Chi-Squared Distance for Histograms

- Now, for every shape context histogram H(E$_2$,i,j) extracted from the second image, we need to estimate how similar it is to H(E$_1$, 291, 198).
- We do that using the **chi-squared measure**.
- Let H and H' be two shape context histograms. In our example:
  - $H = H(E_1, 291, 198)$.
  - $H' = H(E_2, i, j)$.
- We define the chi-squared distance between H and H' as:

$$C(H, H') = \frac{1}{2} \sum_{m=1}^{5} \sum_{n=1}^{12} \frac{\left(H_{m,n} - H'_{m,n}\right)^2}{H_{m,n} + H'_{m,n}}$$

Chi-Squared vs Euclidean Distance

- Euclidean distance: $\sqrt{\sum_{m=1}^{5}\sum_{n=1}^{12}\left(H_{m,n} - H'_{m,n}\right)^2}$
- Chi-squared distance: $\frac{1}{2}\sum_{m=1}^{5}\sum_{n=1}^{12}\frac{\left(H_{m,n} - H'_{m,n}\right)^2}{H_{m,n} + H'_{m,n}}$
- What is the intuition for preferring the chi-squared distance for comparing shape context histograms?
  – In general, the chi-squared distance is often used to measure the difference between histograms. Why?

Chi-Squared vs Euclidean Distance

- Euclidean distance: $\sqrt{\sum_{d=1}^{D}(V_d - V'_d)^2}$

- Chi-squared distance: $\frac{1}{2}\sum_{d=1}^{D}\frac{\left(V_d-V'_d\right)^2}{V_d+V'_d}$

- For simplicity, instead of looking at 60-dimensional shape context histograms, let’s look at four-dimensional vectors.

$$v\ = (22,55,0,42)$$
$$v' = (11,50,5,84)$$

Chi-Squared vs Euclidean Distance

$$v \ = (22,55,0,42)$$
$$v' = (11,50,5,84)$$

- Euclidean distance:

$$\sqrt{\sum_{d=1}^{D}(V_d - V'_d)^2}$$

- Under Euclidean distance, dimension 2 and dimension 3 contribute equally.
  - Dimension 2: $(55\text{-}50)^2 = 5^2 = 25$.
  - Dimension 3: $(0\text{-}5)^2 = 5^2 = 25$.

- Chi-squared distance:

$$\frac{1}{2}\sum_{d=1}^{D}\frac{\left(V_d - V'_d\right)^2}{V_d + V'_d}$$

- Under chi-squared distance, dimension 2 contributes much less than dimension 3.
  - Dimension 2: $\frac{(55-50)^2}{55+50} = \frac{25}{105} = 0.24$
  - Dimension 3: $\frac{(0-5)^2}{0+5} = \frac{25}{5} = 5$

Chi-Squared vs Euclidean Distance

- From the point of view of the Euclidean distance, the distance from 50 to 55 is the same as the distance from 0 to 5.
- However, if we use the chi-squared distance, we think of each dimension as the bin of a histogram, measuring the **frequency** of an event.
  - In shape context histograms, each bin measures the number of edge pixels in a region, which can be thought of as the frequency of getting an edge pixel in a specific region.
- In terms of frequencies:
  - 50 and 55 are rather similar, they differ from each other by about 10%. So, the event happened 10% more frequently in one case than in another case.
  - 0 and 5 are much more different. 0 means that an event NEVER happens, 5 means that an event happened 5 times. So, an event happened infinitely more times in one case than in another case.

Accuracy of Frequency Estimation

- An example of how we evaluate the accuracy of frequency estimation:
- Suppose that, based on historical data, we predict that there will be 50 snowfall events during a specific three-month period, and we get 55 snowfall events.
  - Intuitively, our initial estimate seems reasonably accurate.
- Now, suppose that we predict that there will be 0 snowfall events during that period, and instead there are five such events.
  - This difference between prediction and outcome seems more significant.

Chi-Squared vs Euclidean Distance

- At the end of the day, the most convincing argument that one distance is better than another (in a specific setting) is experimental results.
- Later, we will use specific datasets to get quantitative results comparing various approaches.
- Those quantitative results will be useful for answering a lot of different questions, such as:
  - Are shape context features more useful than templates of local regions?
  - Is it better to compare shape context features using the chi-squared distance or the Euclidean distance?

Orientation Histograms

- Another type of local feature is orientation histograms.
  - Reference: Freeman, William T., and Michal Roth. "Orientation histograms for hand gesture recognition." In *International workshop on automatic face and gesture recognition*, vol. 12, pp. 296-301. 1995.
- Key idea: for each pixel location (i,j), we build the histogram of gradient orientations of all pixels in a square window W centered at (i,j).

Orientation Histograms

- To compute orientation histograms, we need to compute the gradient orientation of every pixel in the image.
- How do we do that?
- Review: the gradient orientation at pixel (i,j) is equal to atan2(gradient_dy(i,j), gradient_dx(i,j)), where:
  - gradient_dy is the result of imfilter(image, [-1, 0, 1]’)
  - gradient_dx is the result of imfilter(image, [-1, 0, 1])

Orientation Histograms - Options

- Based on the previous description, what parameters and design options do we need to specify in order to get a specific implementation?

Orientation Histograms - Options

- Parameters and design choices:
  - Do angles range from 0 to 180 or from 0 to 360 degrees?
  - B: number of bins for the histogram.
    - Common choices: 8-36 bins if angles range from 0 to 360 degrees.
  - h: half-width of window W we extract around (i,j).
  - Do all pixels get equal weight? Alternative options:
    - Pixels with gradient norms below a threshold get weight 0.
    - The weight of each pixel is proportional to the norm of its gradient.
    - Pixels farther from the center (i,j) get lower weights.
  - How do we compare orientation histograms to each other?
    - Euclidean distance or chi-squared distance can be used.

Differences from Shape Context

- What are the pros and cons of orientation histograms compared to shape context?

Differences from Shape Context

- What are the pros and cons of orientation histograms compared to shape context?
- Sensitivity to edge detection results:
  - Shape context depends on edge detection.
  - Edge detection results can be different depending on camera settings, brightness, contrast, etc.
    - Canny edge detection requires us to choose thresholds, and the same thresholds can lead to different results if there are changes in camera settings, brightness, contrast.
  - Gradient orientations are more stable with respect to such changes.

Differences from Shape Context

- What are the pros and cons of orientation histograms compared to shape context?
- Preserving spatial information:
  - In shape context, the location of edge pixels matters.
    - A pixel with edges mainly above it has a shape context histogram that is very dissimilar from a pixel with edges mainly below it.
  - In orientation histograms, location information is not preserved.
    - As a result, different-looking regions get similar histograms.
  - Therefore, shape context preserves spatial information better.

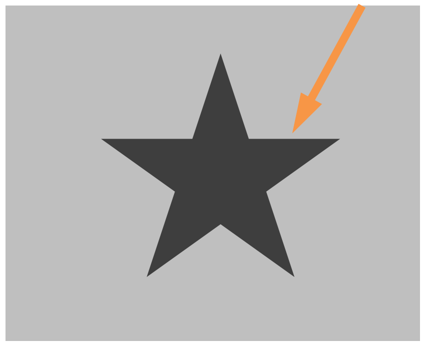
Range of Angles

- In some cases, we prefer to consider gradient orientations as ranging from 0 to 180 degrees.
- In other cases, we consider gradient orientations as ranging from 0 to 360 degrees.
- What are the practical implications?
  - In what situations would we prefer one choice over the other?

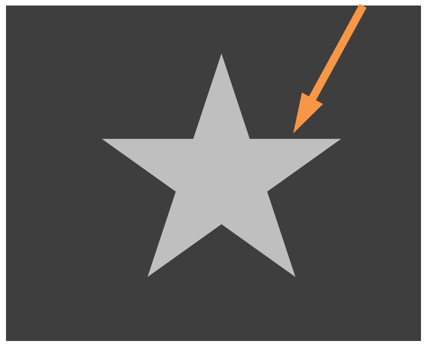
Range of Angles

- Consider these two images, and the indicated pixel correspondences.
- Intuitively, P_1 and P_2 are at corresponding positions.
- If we are building a detector of star shapes, we want the features extracted from P_1 and P_2 to be similar.
  - This way we can detect darker star shapes in brighter background, and brighter star shapes in darker background.

Pixel P_1 of image I_1.



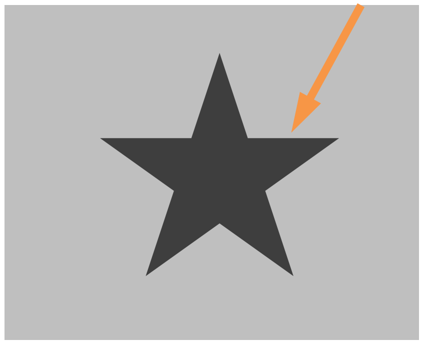
Pixel P_2 of image I_2.



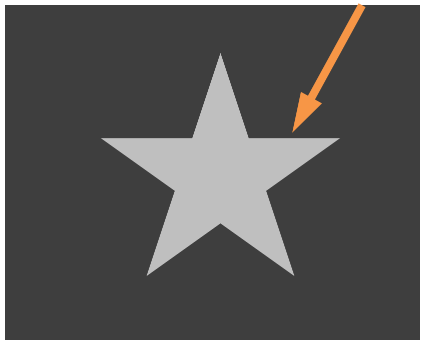
Range of Angles

- The gradient orientations at P_1 and P_2 differ by 180 degrees.
  - Which way is the gradient pointing at P_1?
  - Which way is the gradient pointing at P_2?

Pixel P_1 of image I_1.



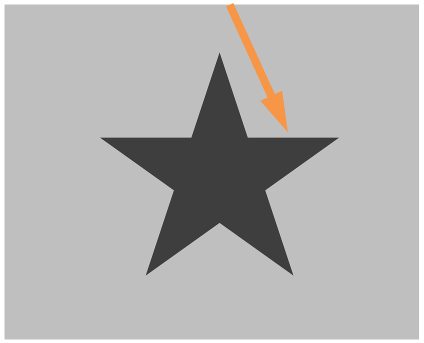
Pixel P_2 of image I_2.



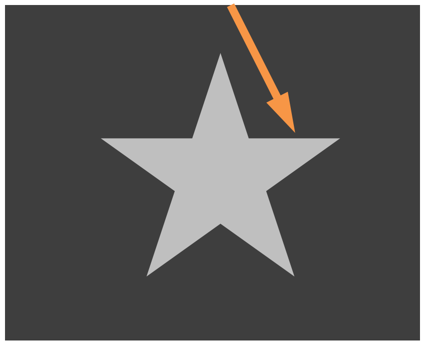
Range of Angles

- The gradient points towards the direction where the intensity increases the fastest.
  - At P_1, intensity increases upwards, so the gradient points up (orientation: 270 degrees).
  - At P_1, intensity increases downwards, so the gradient points down (orientation: 90 degrees).
- If we treat 270 degrees as 90 degrees, then the two orientations match.

Pixel P_1 of image I_1.



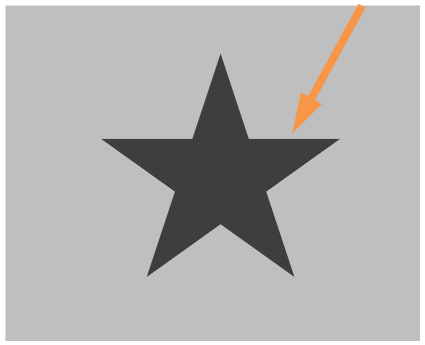
Pixel P_2 of image I_2.



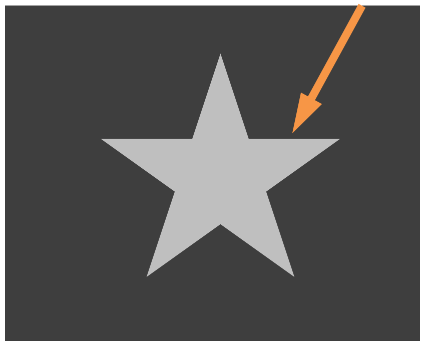
Range of Angles

- So, in general, for detection of specific shapes, we want to define angle values as ranging between 0 and 180 degrees.

Pixel P_1 of image I_1.



Pixel P_2 of image I_2.



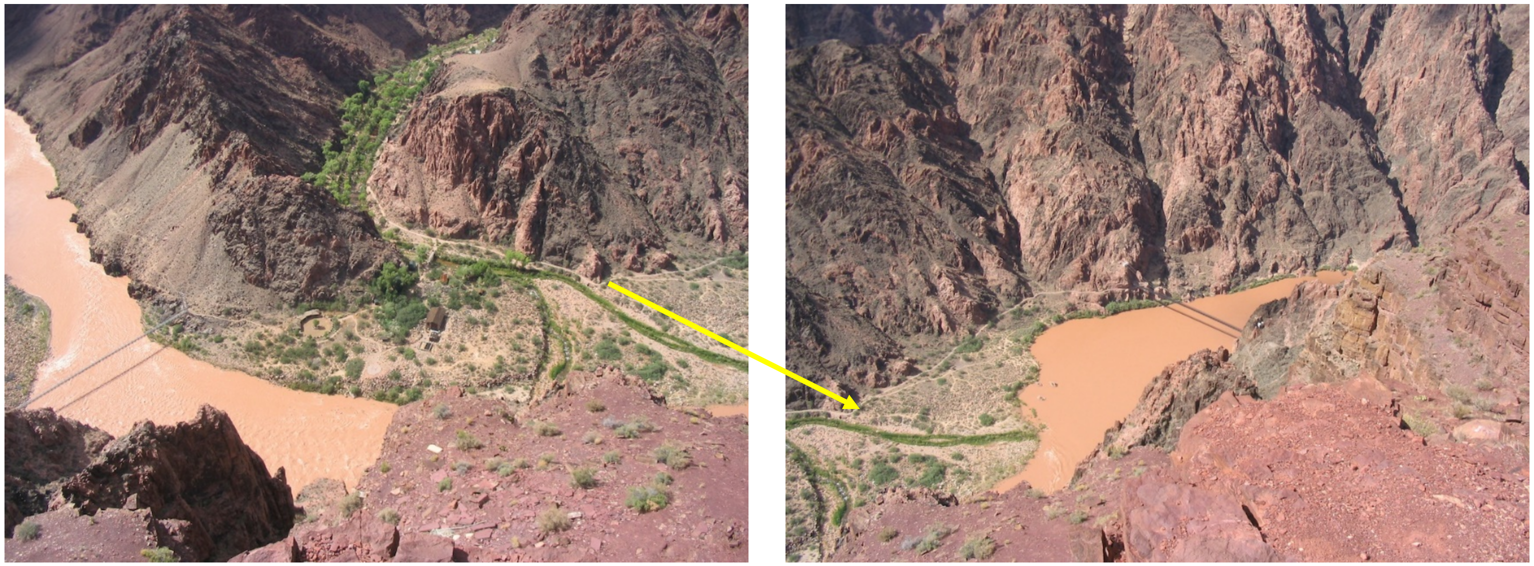
Range of Angles

- Now, consider the problem of pixel correspondences from a stereo pair of cameras.
  - Assume that the x, y, z axes of both cameras are parallel to each other.
- Here, a difference of 180 degrees in gradient orientation should indicate a strong mismatch.
  - So, we should use angle values that range from 0 to 360 degrees.

Credit: images downloaded from http://www.aenigmatis.com/3d-stereo/stereoscopic.htm

Range of Angles

- Note that in the images below, matching locations do have different gradient orientations, because the camera viewpoints are rotated with respect to each other.
- We will see later how SIFT features still benefit from using angle values from 0 to 360 degrees in this case.
  - In SIFT features, each region is rotated in such a way that matching regions (usually) get matching gradient orientations.



Histograms of Oriented Gradients

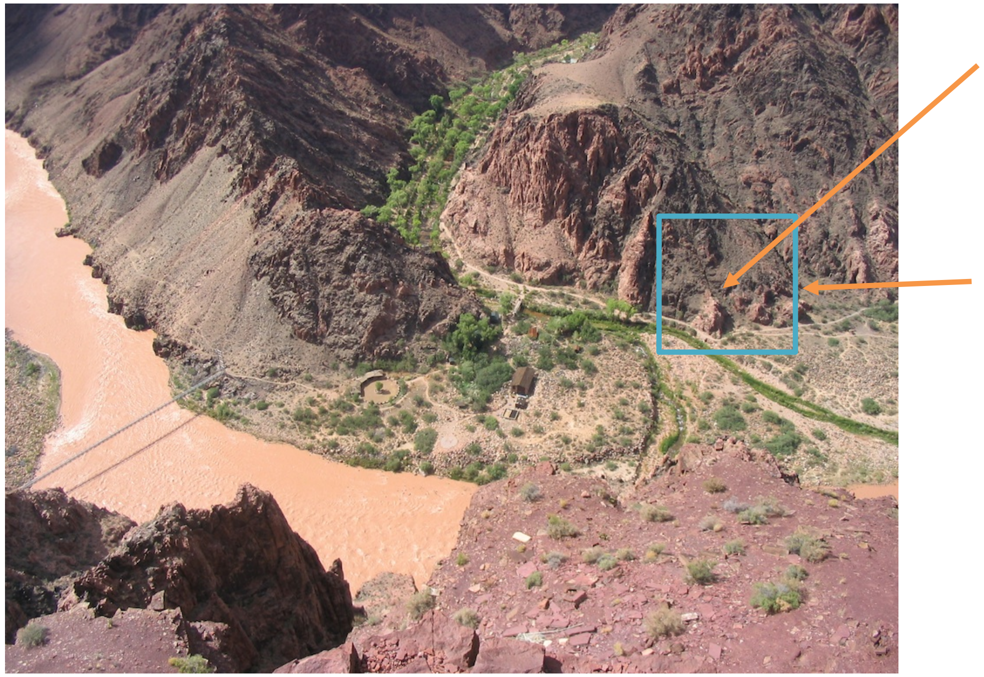
- Reference:
  - Dalal, Navneet, and Triggs, Bill. "Histograms of oriented gradients for human detection." In *International Conference on Computer Vision & Pattern Recognition (CVPR)*, vol. 1, pp. 886-893. IEEE Computer Society, 2005.
- Histograms of Oriented Gradients (HOG features, or HOGs) build on top of orientation histograms.
  - One goal is to also preserve some spatial information, which orientation histograms disregard entirely.
  - Additional postprocessing and normalization of histograms lead to improved accuracy.

Histograms of Oriented Gradients

- Suppose we are given an image I, and a pixel location (i,j).
- We want to compute the HOG feature for that pixel.
- The main operations can be described as a sequence of four steps.

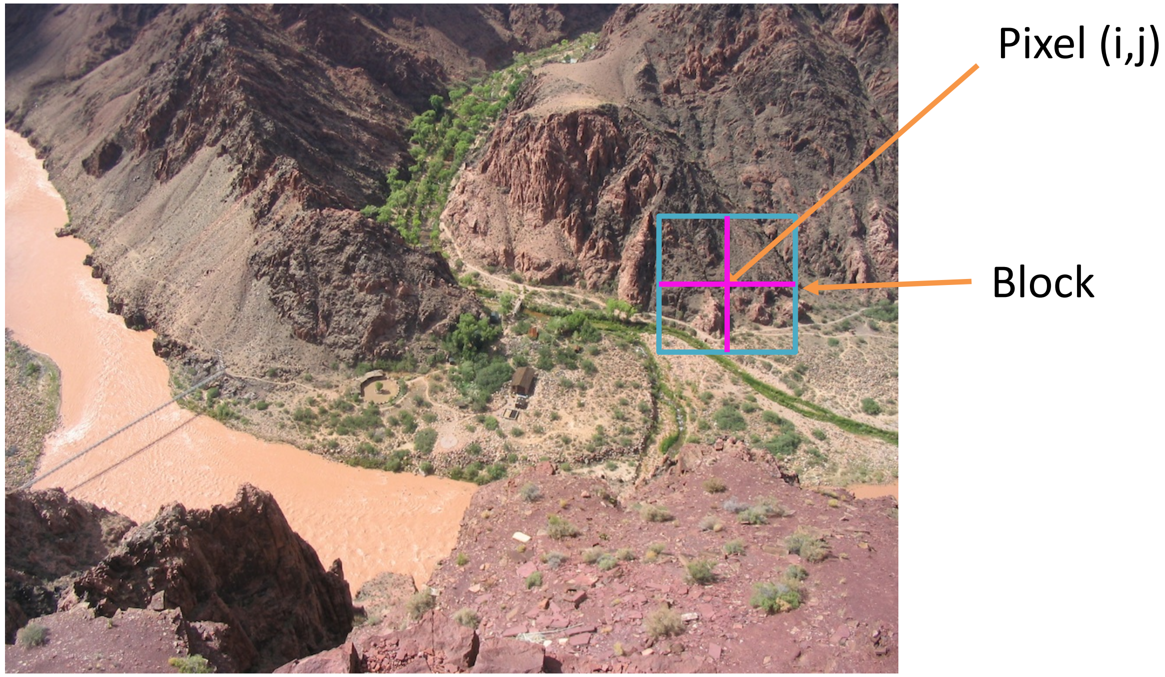
Histograms of Oriented Gradients

- Step 1: Extract a square window (called "block") of some size.



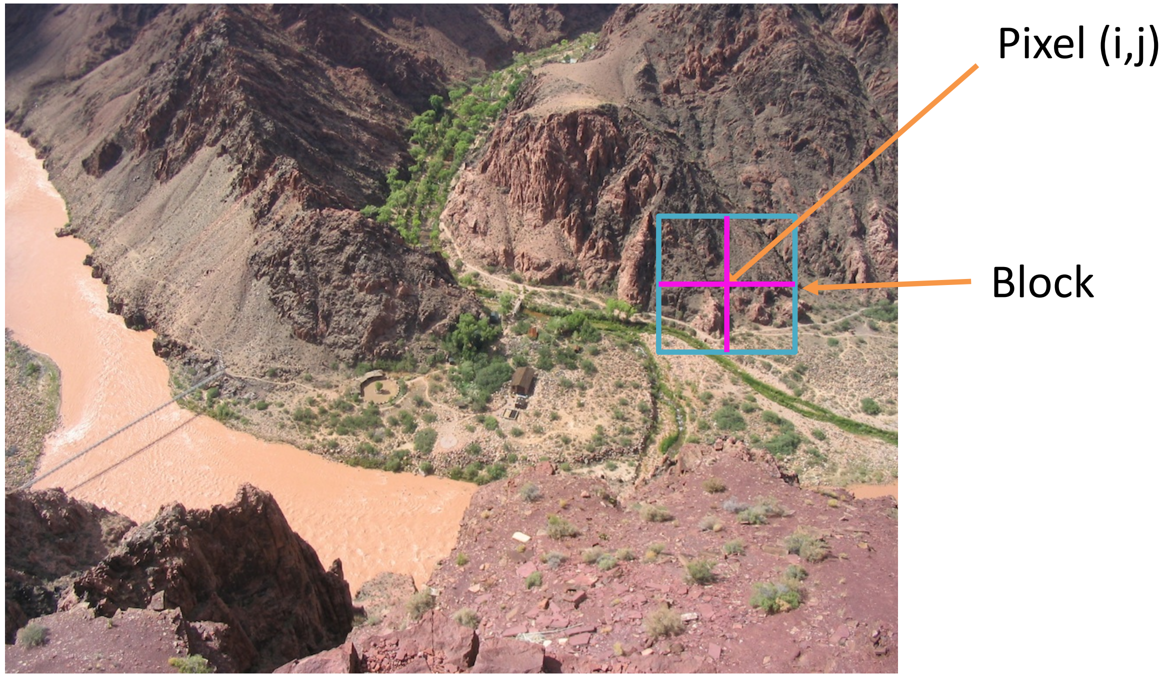
Histograms of Oriented Gradients

- Step 1: Extract a square window (called “block”) of some size.
- Step 2: Divide block into a square grid of sub-blocks (called “cells”) (2x2 grid in our example, resulting in four cells).



Histograms of Oriented Gradients

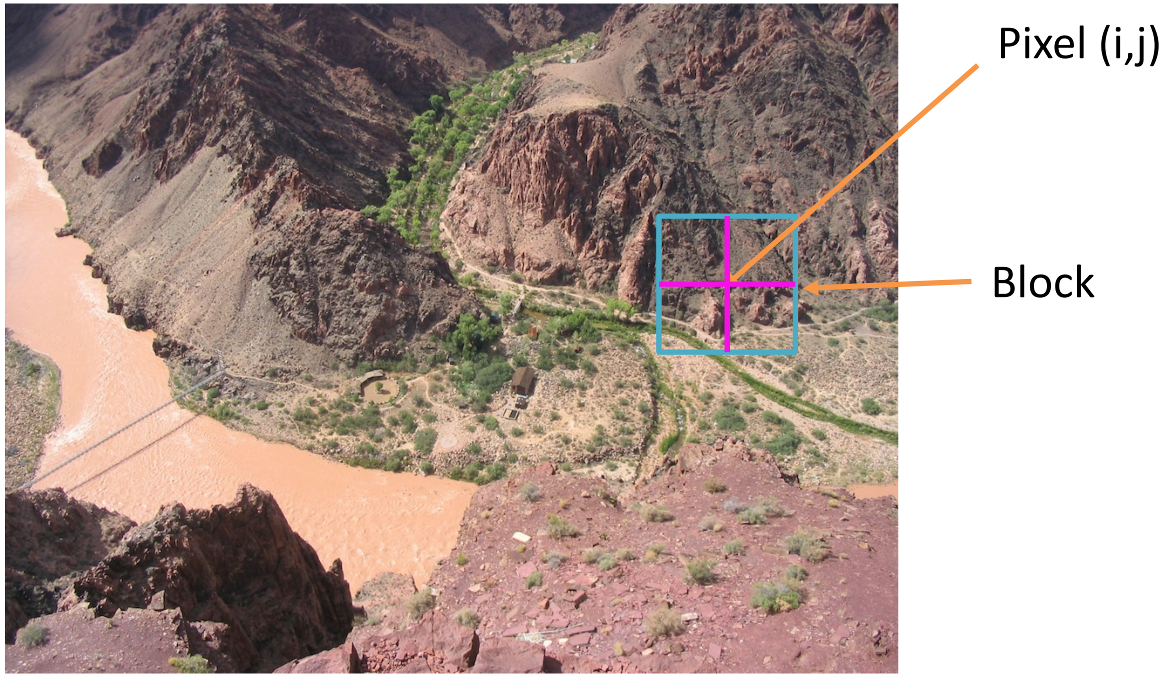
- Step 1: Extract a square window (called “block”) of some size.
- Step 2: Divide block into a square grid of sub-blocks (called “cells”) (2x2 grid in our example, resulting in four cells).
- Step 3: Compute orientation histogram of each cell.
- Step 4: Concatenate the four histograms.



Histograms of Oriented Gradients

Let vector **v** the be concatenation of the four histograms from step 4.

- Step 5: normalize **v**. Here we have three options for how to do it:
  - Option 1: Divide **v** by its Euclidean norm.
  - Option 2: Divide **v** by its L_1 norm (the L_1 norm is the sum of all absolute values of v).



Histograms of Oriented Gradients

Let vector **v** the be concatenation of the four histograms from step 4.

- Step 5: normalize **v**. Here we have three options for how to do it:
  - Option 3:
    - Divide **v** by its Euclidean norm.
    - In the resulting vector, clip any value over 0.2 (i.e., for any value over 0.2, replace it with 0.2).
    - Then, renormalize the resulting vector by dividing again by its Euclidean norm.

- Here is the Matlab code for option 3:

```
v = v / norm(v);  % Divide by Euclidean norm
v(v > 0.2) = 0.2; % Clip values over 0.2
v = v / norm(v);  % Divide again by Euclidean norm
```

Summary of HOG Computation

- Step 1: Extract a square window (called “block”) of some size around the pixel location of interest.
- Step 2: Divide block into a square grid of sub-blocks (called “cells”) (2x2 grid in our example, resulting in four cells).
- Step 3: Compute orientation histogram of each cell.
- Step 4: Concatenate the four histograms.
- Step 5: normalize **v** using one of the three options described previously.
- The HOG feature is the result of step 5.

Histograms of Oriented Gradients

- Based on the previous description, what parameters and design options do we need to specify in order to get a specific implementation?

Histograms of Oriented Gradients

- Parameters and design options:
  - Angles range from 0 to 180 or from 0 to 360 degrees?
    - In the Dalal & Triggs paper, a range of 0 to 180 degrees is used, and HOGs are used for detection of pedestrians.
  - Number of orientation bins.
    - Usually 9 bins, each bin covering 20 degrees.
  - Cell size.
    - Cells of size 8x8 pixels are often used.
  - Block size.
    - Blocks of size 2x2 cells (16x16 pixels) are often used.
- Usually a HOG feature has 36 dimensions.
  - 4 cells * 9 orientation bins.

HOGs vs. Orientation Histograms

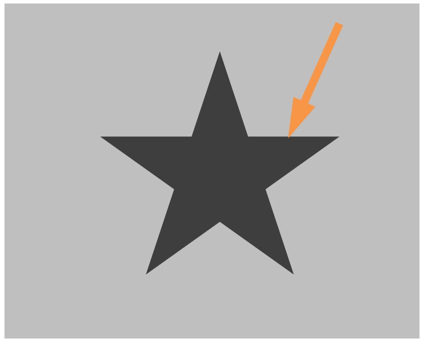
- Why would HOGs work better than orientation histograms?

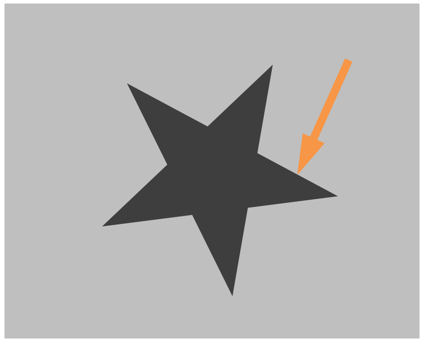
HOGs vs. Orientation Histograms

- Key difference:
  - The orientation histogram is extracted from a single block.
  - The HOG approach breaks up that block into (usually four) cells.
    - The orientation histogram of each cell is computed separately.
    - The histograms of all cells are concatenated.
  - Thus, HOG features preserve some spatial information, whereas orientation histograms do not preserve any.

Orientation Invariance

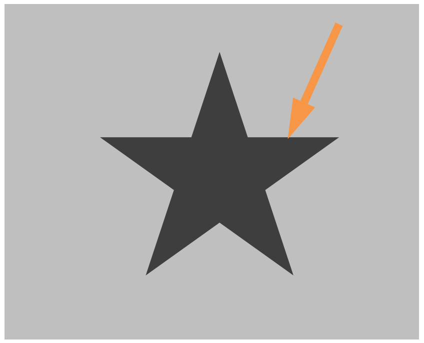
- Consider these features that we have talked about so far:
  - Shape context.
  - Orientation histograms.
  - HOGs.
- Is any of them orientation invariant?
  - In other words: does the feature vector of a certain pixel change if we rotate the image around the image plane?

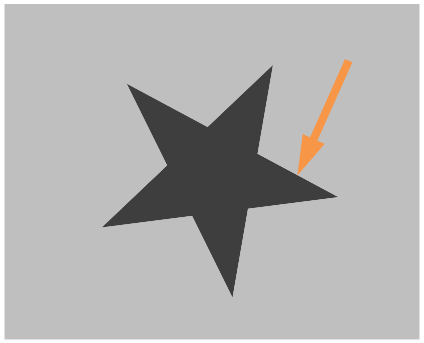




Orientation Invariance

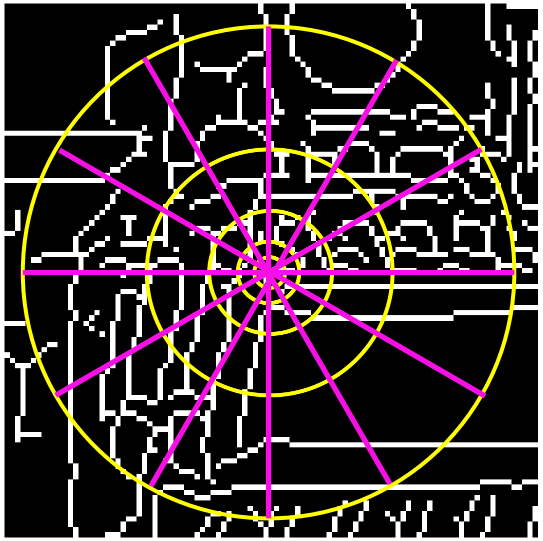
- All these types of features change if the image gets rotated.
- An orientation-invariant feature would not change.
- Can you think of any way to define orientation-invariant features?





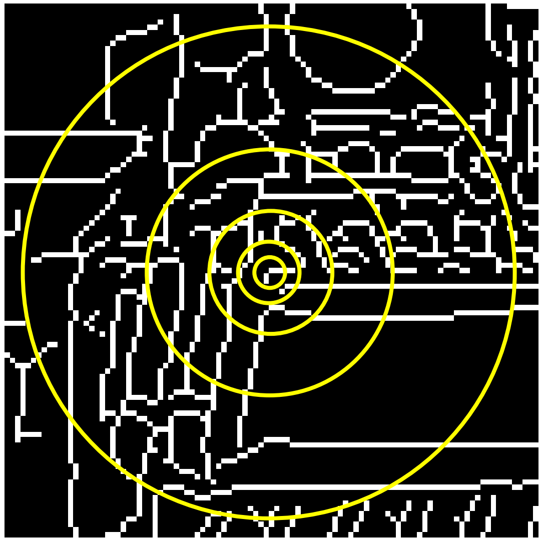
Orientation Invariance

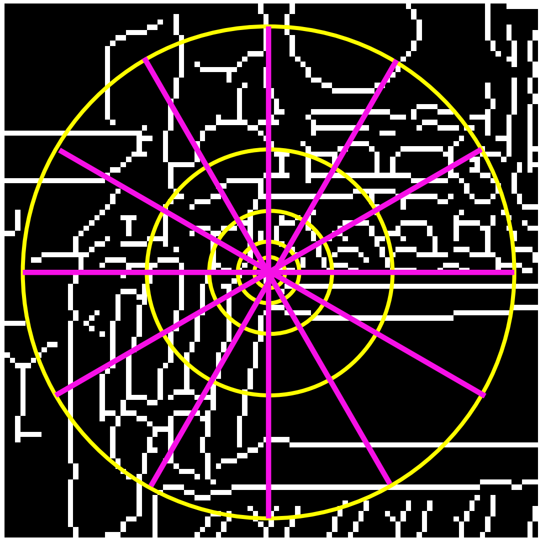
- Consider the shape context feature.
- It divides the image region around pixel (i,j) into a log-polar grid of 60 cells.
- Obviously, rotating the image around (i,j) would shift the edge pixels, and change the counts stored at the bins.



Orientation Invariance

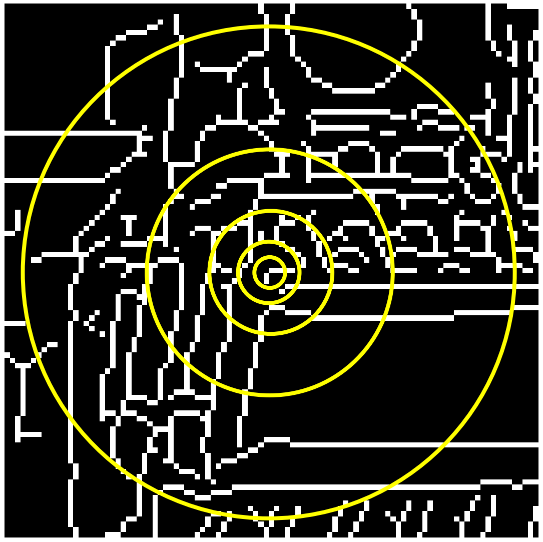
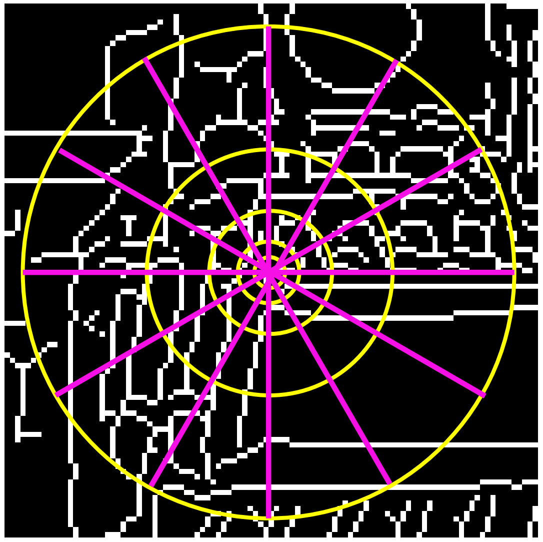
- One approach for orientation invariance would be to get rid of the pizza-slice division of the region.
- Now we only have a 5-cell grid, that we can use to define a 5-dimensional histogram.
- Rotating the image around (i,j) would not change the count of each bin.





Orientation Invariance

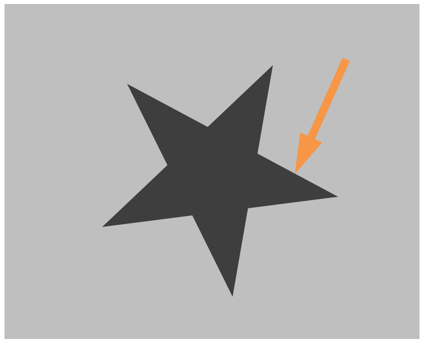
- This five-dimensional variation of shape context is an example of a orientation invariant feature.
- Note that "orientation invariance" should not be interpreted as saying that the feature vectors would be numerically identical regardless of the orientation.
    - There will be small differences due to rounding and interpolation, since rotations map integer pixel coordinates to non-integer values.

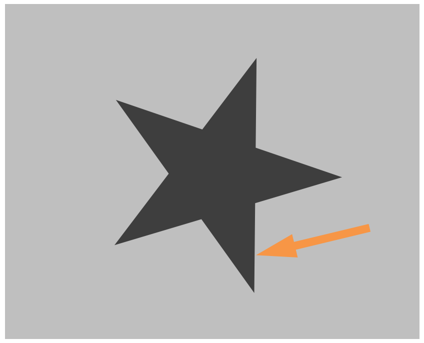
Picking a Dominant Orientation

- One approach for achieving orientation invariance:
  - For each pixel (i,j) where we want to extract a local feature vector F(i,j):
    - Identify a "dominant" orientation for pixel (i,j), or for the region around (i,j).
    - Create a rotated copy of the region around (i,j), so that the "dominant" orientation becomes 0 (parallel to the x-axis).
    - Extract F(i,j) from the rotated copy.

Original image



Rotated image



Picking a Dominant Orientation

- Key idea: if the image get rotated:
  - The region around (i,j) looks different.
  - The rotated copy of the region should still look the same.

Image 1

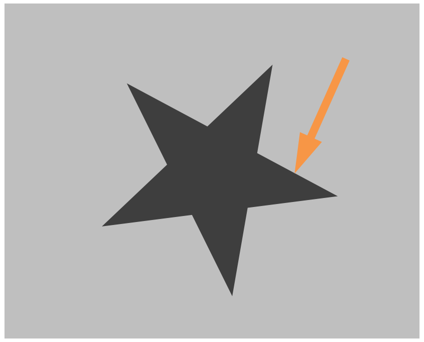
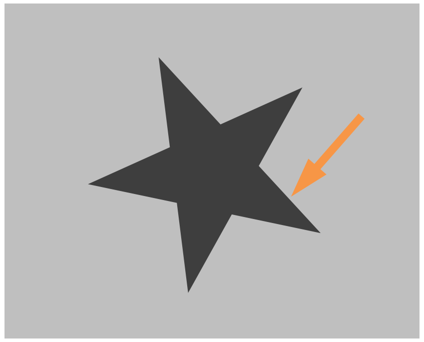
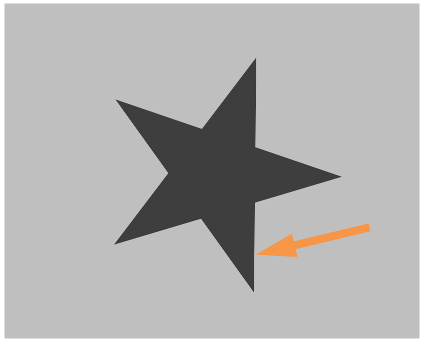


Image 2 (rotated version of image 1).



Rotated version of both images so that local orientation at (i,j) is zero degrees.



Picking a Dominant Orientation

- How do we pick a "dominant" orientation?

Image 1

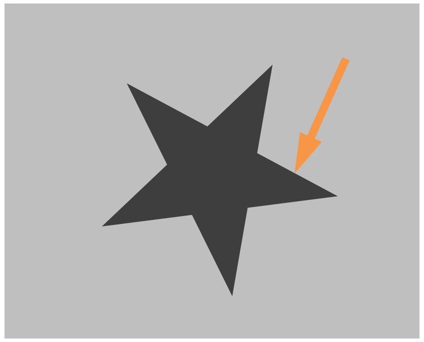
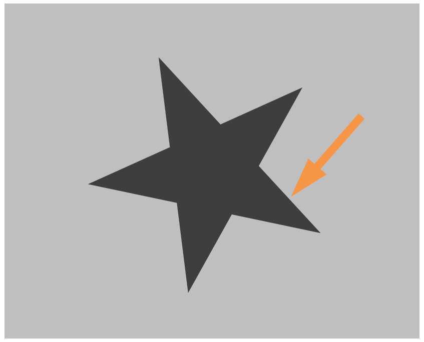
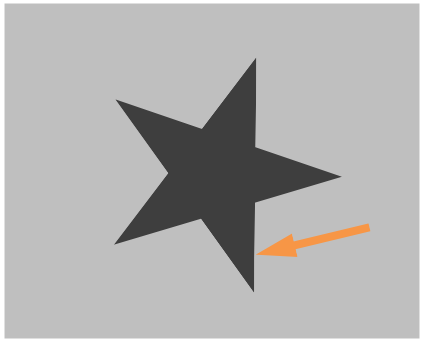


Image 2 (rotated version of image 1).



Rotated version of both images so that local orientation at (i,j) is zero degrees.



Picking a Dominant Orientation

- How do we pick a "dominant" orientation?
- One approach: use the gradient orientation at pixel (i,j).
- Second approach: look at the histogram of orientations around (i,j), and pick the orientation with the highest count.

Image 1

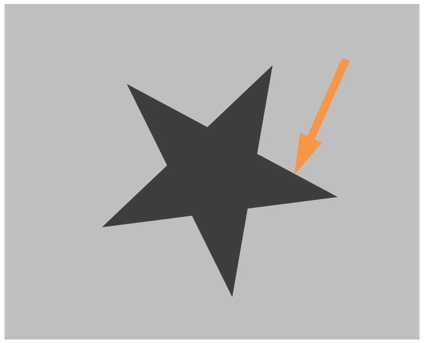
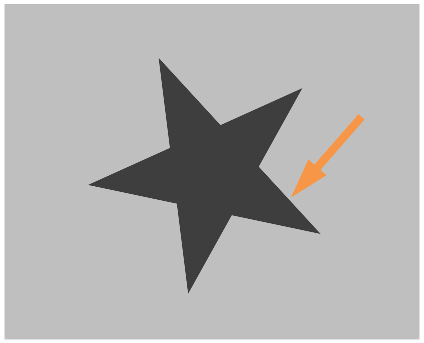
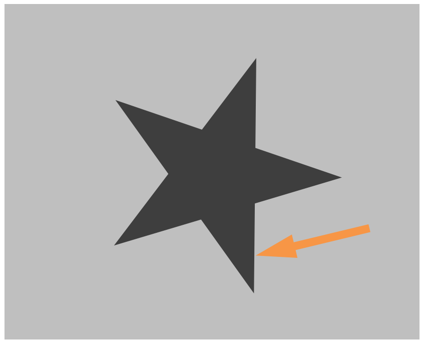


Image 2 (rotated version of image 1).



Rotated version of both images so that local orientation at (i,j) is zero degrees.



Picking a Dominant Orientation

- What are possible pitfalls of these approaches?

Image 1

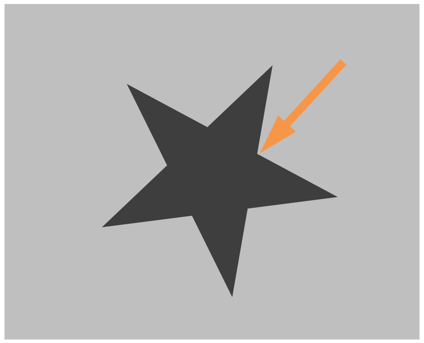
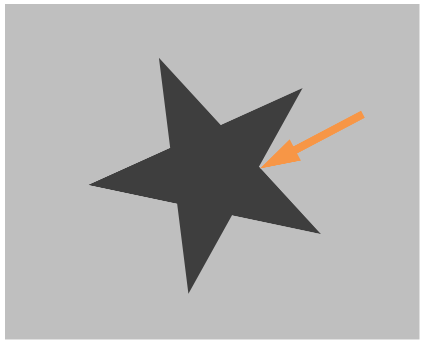


Image 2 (rotated version of image 1).



Picking a Dominant Orientation

- What are possible pitfalls of these approaches?
- There are pixels where there is no “dominant” orientation.
- In those cases, the orientation that we pick in image 1 may not match the orientation we pick in image 2.
- Consider the pixels shown in the two images below.
  - Those pixels should match each other.
  - However, they represent corners, there is no visually dominant orientation.

Image 1

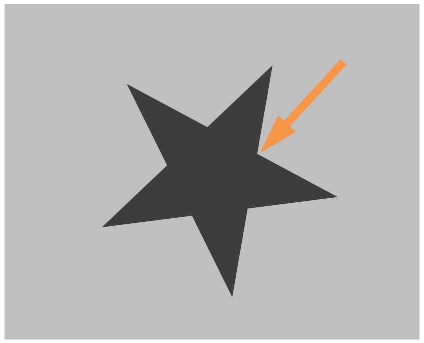
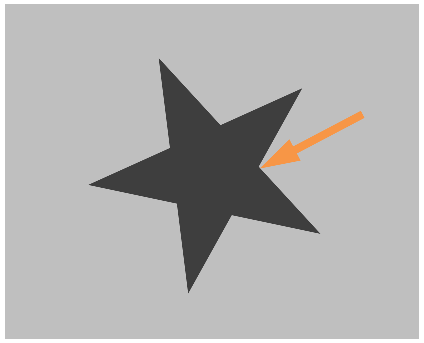


Image 2 (rotated version of image 1).

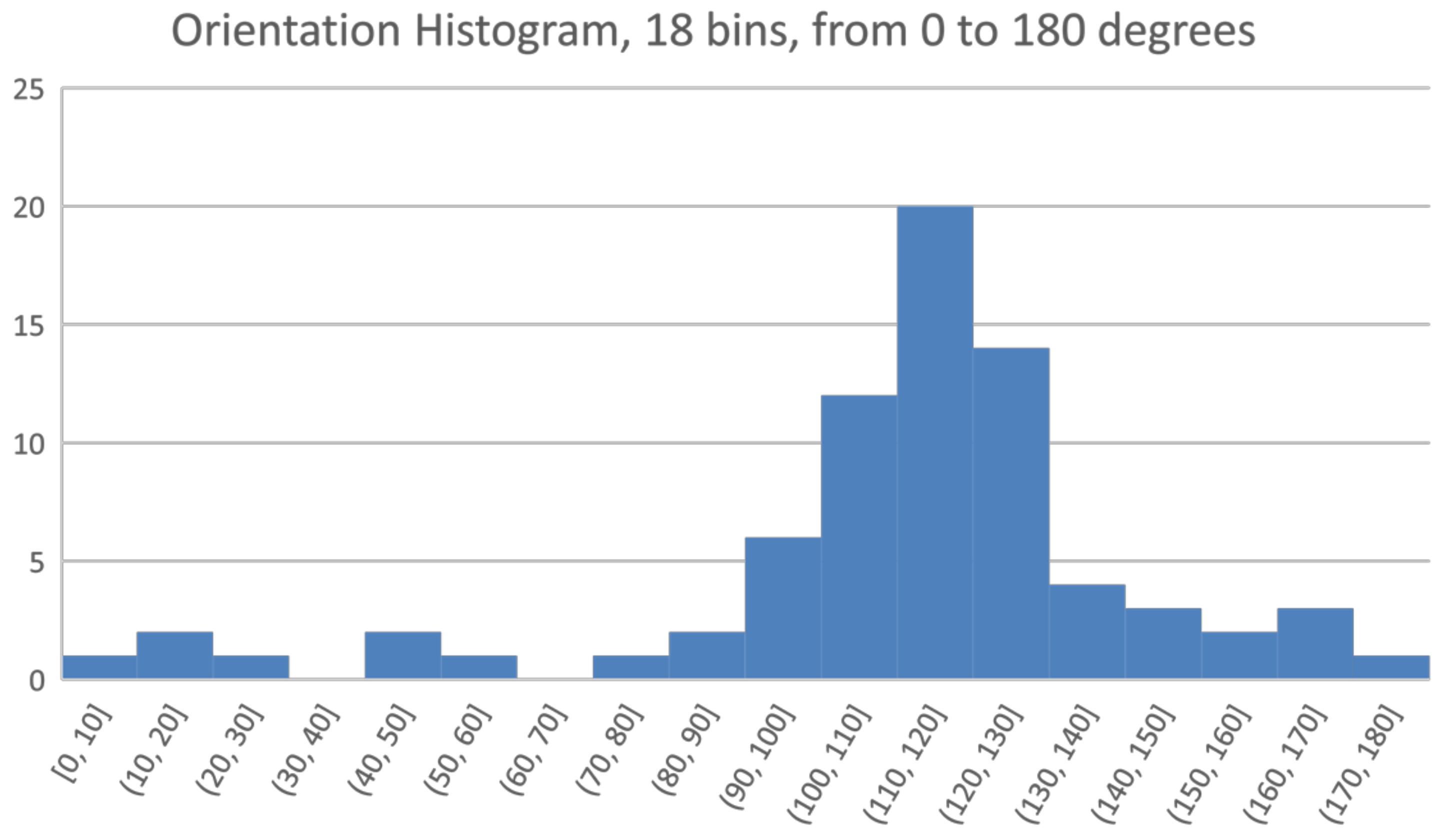


Multiple Dominant Orientations

- Instead of picking a single dominant orientation, we can pick multiple dominant orientations, if needed.
- Picking dominant orientations for pixel (i,j):
  - Construct orientation histogram for region around (i,j).
  - Find bin B_{max} with highest value.
  - Define S = set of bins whose values are both local maxima and within 80% of the value of B_{max}.
  - S is a set of bins. All orientations corresponding to those bins can be treated as “dominant” orientations.
- If S has only one bin, we get one dominant orientation.
- If S has multiple bins, we get multiple orientations.

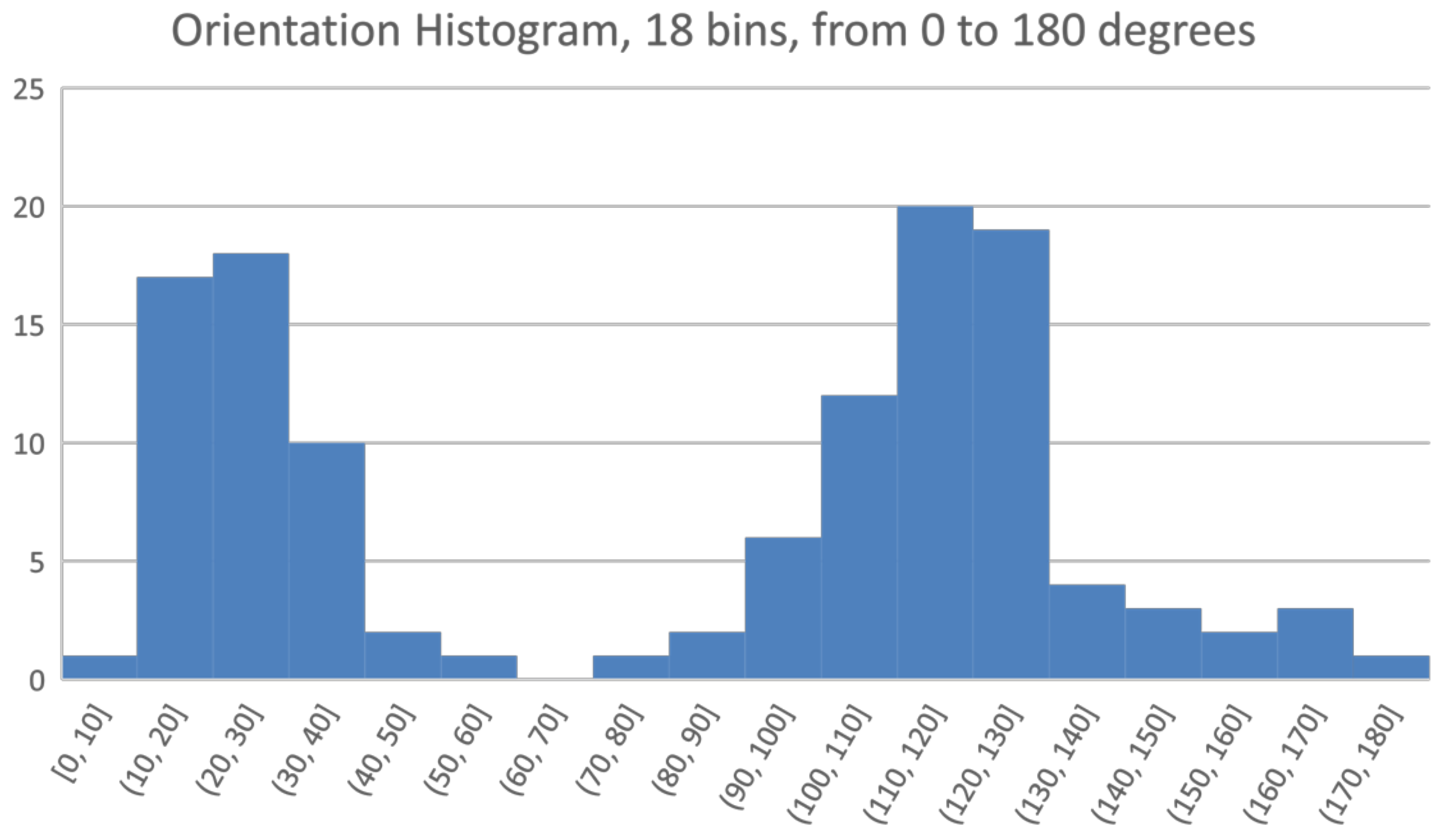
Example

- Here is an example with a clear dominant peak, at range (110,120) degrees.



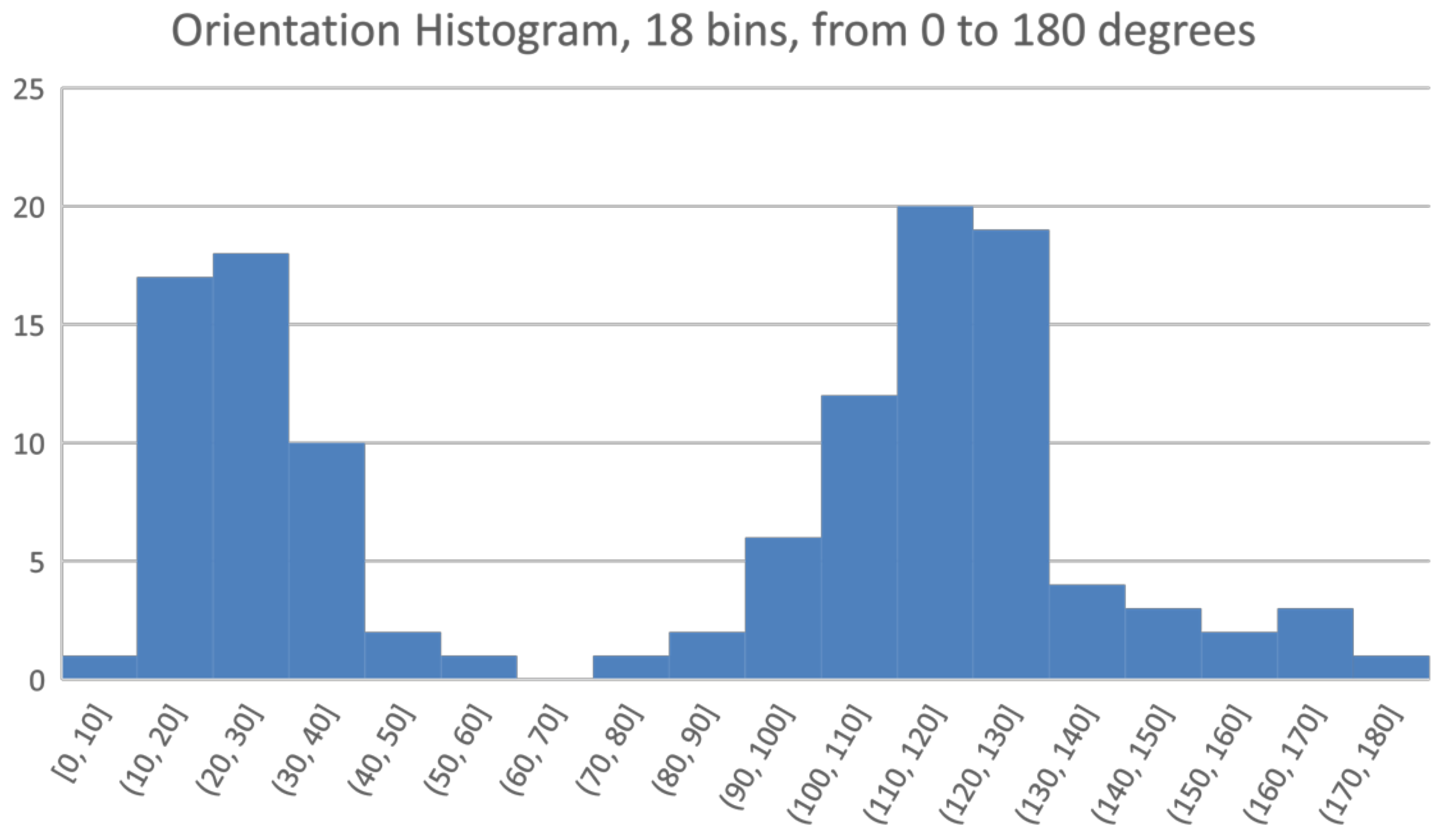
Example

- Here is a more complicated example.
- Which peaks do we keep as dominant peaks?



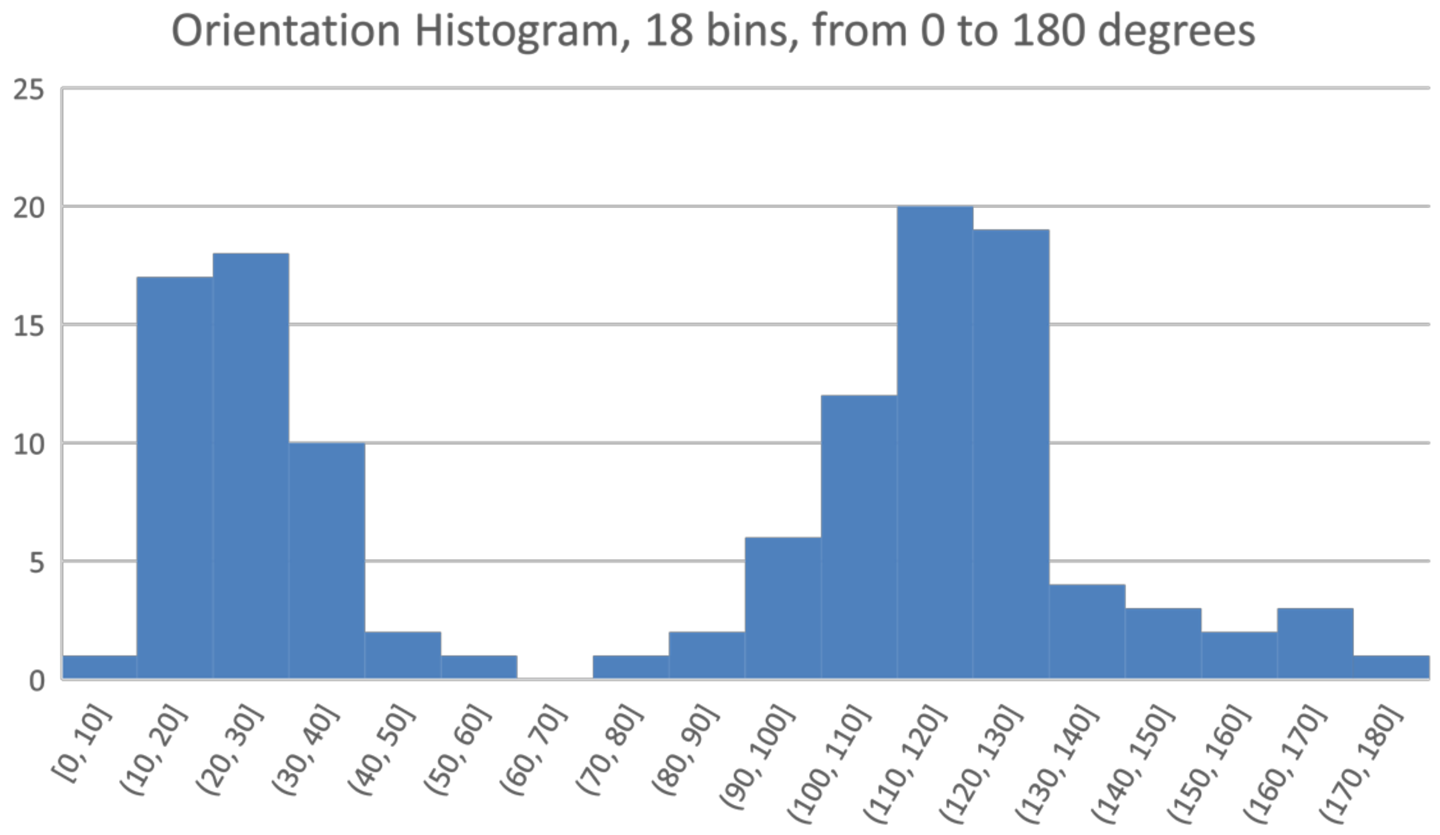
Example

- Step 1: find max value. 20 votes for (110,120) bin.



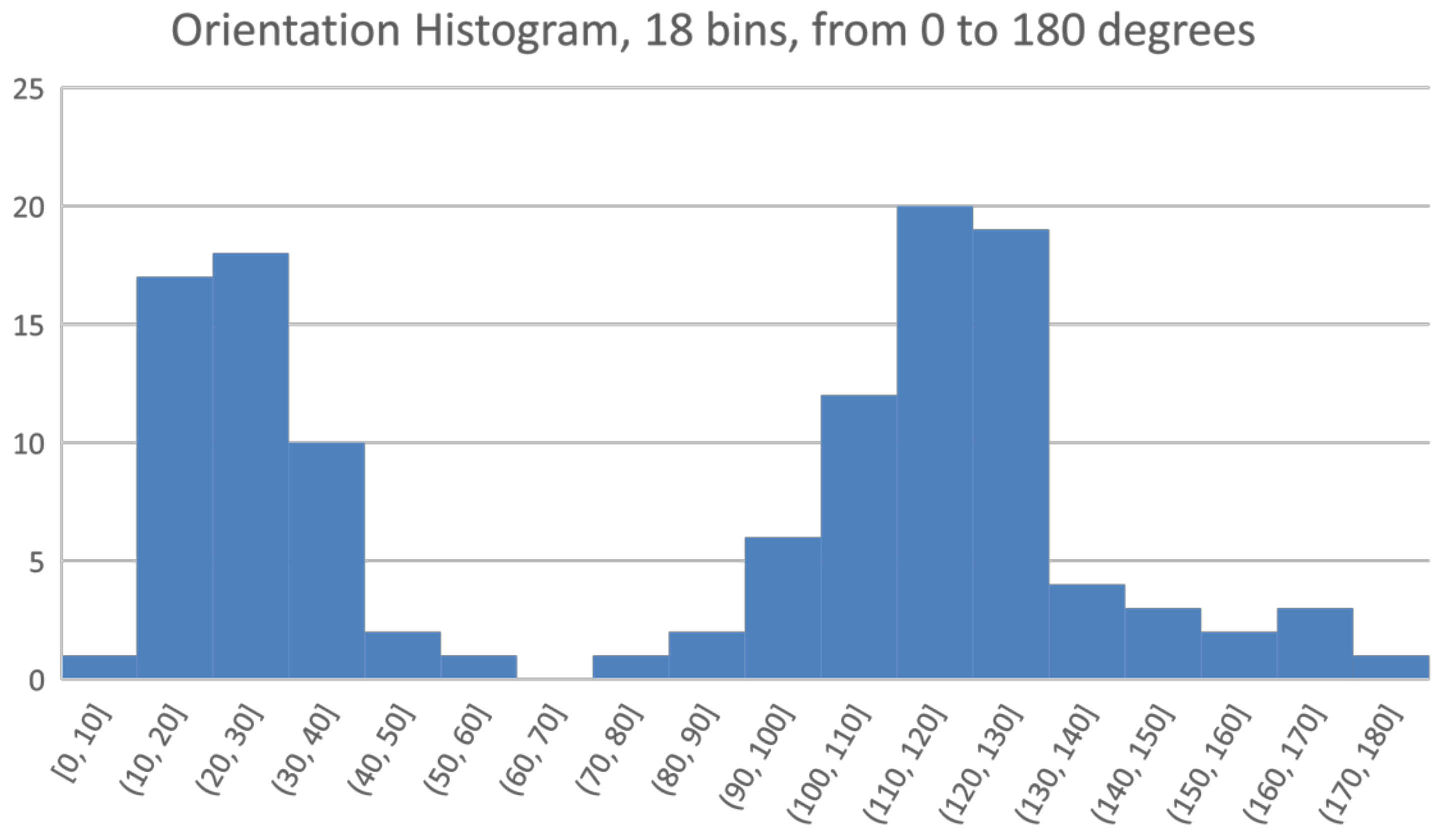
Example

- Step 2: find all bins with values ≥ 80% of the max value.
  - Bins with 16 or more votes: (10,20), (20,30), (110,120), (120,130).



Example

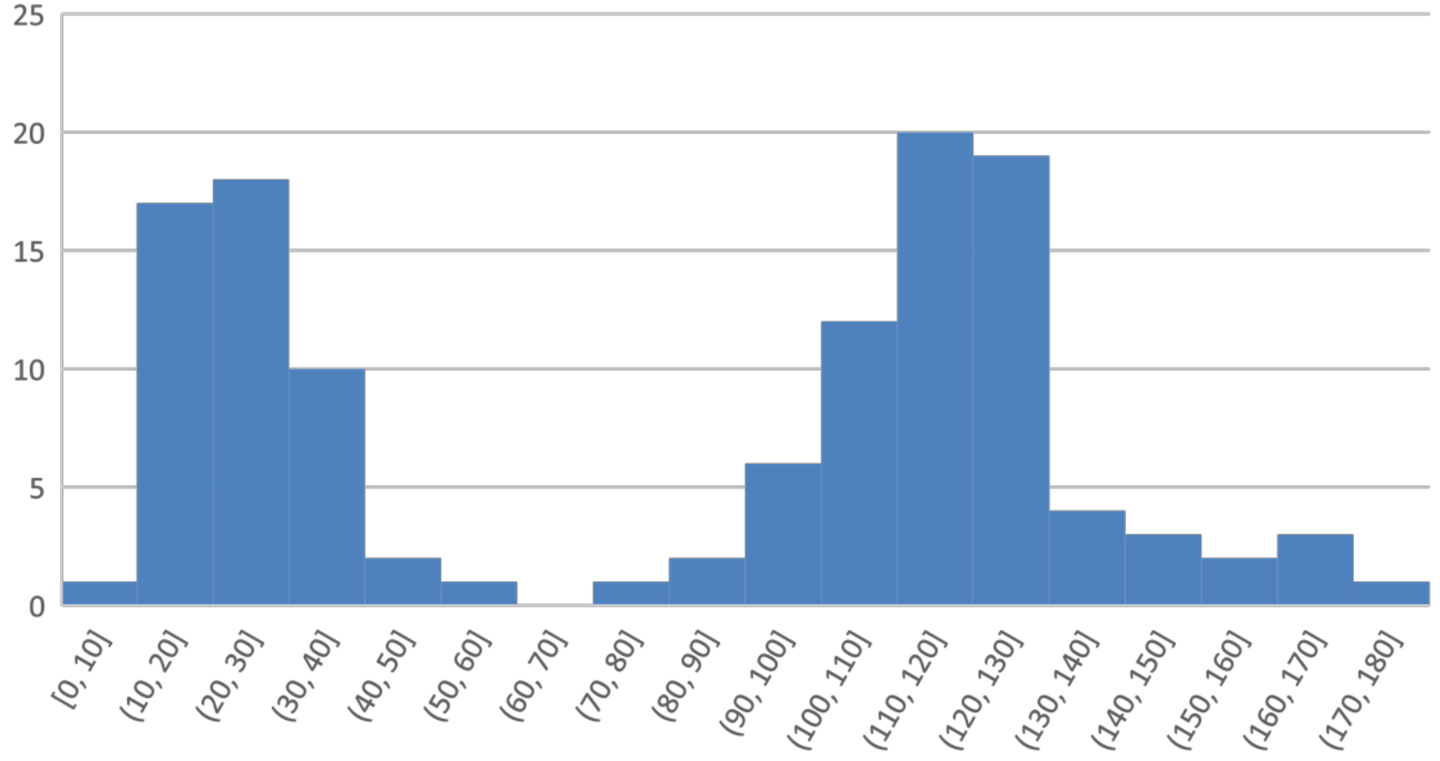
- Step 3: Out of all bins found in Step 2, keep local maxima.
  - Bins (20,30) and (110,120) are local maxima, and we keep them.
  - Bins (10,20) and (120,130) are not local maxima, and we discard them.



Regarding Local Maxima

- A local maximum is simply a value that is higher than its two neighboring values.
  - Bins (10,20) and (120,130) have at least one neighbor with higher value.

Orientation Histogram, 18 bins, from 0 to 180 degrees

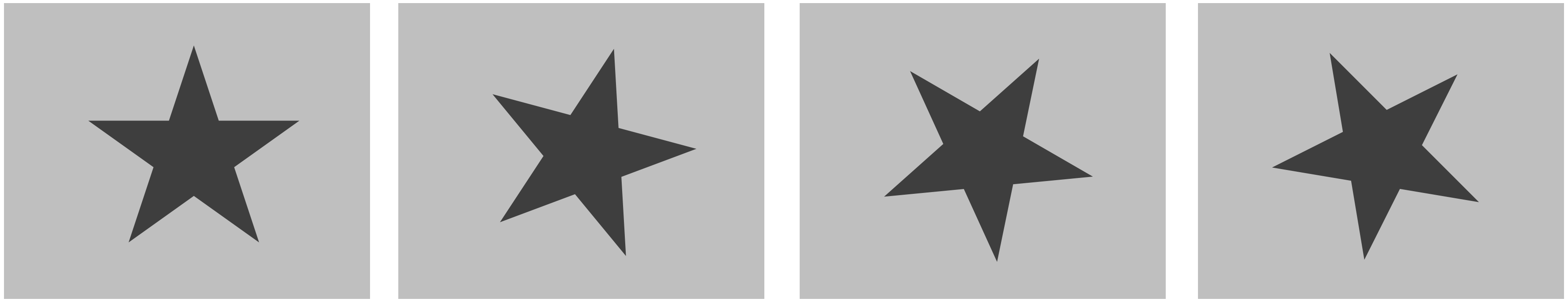


Multiple Dominant Orientations

- Orientation invariant feature extraction for pixel (i,j):
  - Construct orientation histogram for region around (i,j).
  - Find bin B_{max} with highest value.
  - Find set of bins S whose values are within 80% of the value of B_{max}.
  - For each orientation θ corresponding to a bin in S:
    - Get a rotated copy of region around (i,j), by rotating the region by –θ degrees.
    - Extract feature vector F(i,j) from the rotated region.
- Note that, if S has multiple bins, this means that we extract multiple feature vectors from location (i,j).

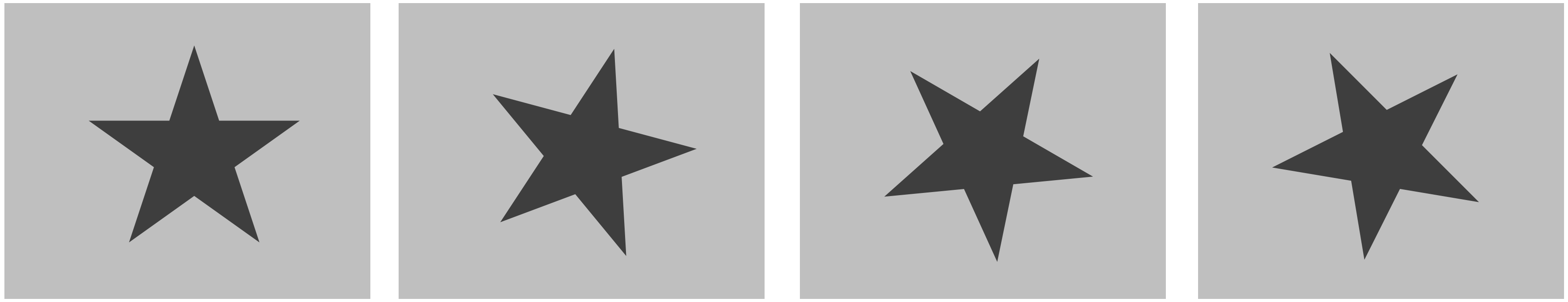
Comparison to Multi-Orientation Search

- An alternative would be to use the same approach we would use for multi-orientation template search:
  - Rotate the image in multiple ways, and extract feature vectors from all the rotated versions.
- How does that alternative compare to rotating each region based on the dominant orientation(s)?



Comparison to Multi-Orientation Search

- Rotating the whole image in multiple ways leads to more feature vectors being defined.
  - One for every combination (i,j,θ) of pixel location and orientation.
- Finding the dominant orientation(s) in practice leads to fewer feature vectors being defined.
  - Usually, most pixels have a single dominant orientation.



Feature Matching: Complexity

- Suppose that we want to use feature matching to find pixel correspondences, for 3D stereo-based estimation.
- What is the time complexity of that?

Feature Matching: Complexity

- Suppose that we want to use feature matching to find pixel correspondences, for 3D stereo-based estimation.
- What is the time complexity of that?
- The time complexity depends on exactly how we do the matching.
  - If we compare every feature vector in one image to every feature vector in the other image, we get time complexity that is quadratic to the number of features.
  - If we have some constraints (for example, looking for matches on an epipolar line, for stereo), we can do better.
- It is useful to have in mind that, sometimes, feature matching time is quadratic to the number of features.

Feature Matching: Complexity

- Since time complexity is (sometimes) quadratic to the number of features, reducing the number of features can lead to significantly faster running times.
- This is why extracting features at dominant orientations is preferable to extracting features at a pre-defined large number of image orientations.

Choosing Feature Locations

- Another design choice that determines the number of extracted feature vectors (and thus the running time of feature matching):
  - The number of locations (i,j) where we extract feature vectors.
- So, which pixels should we extract feature vectors from?
  - One approach: all pixels (leads to much slower running times).
  - Second approach: all edge pixels.
    - If 10% of image pixels are edge pixels, quadratic-time feature matching becomes how much faster?

Feature Matching: Complexity

- Another design choice that determines the number of extracted feature vectors (and thus the running time of feature matching):
  - The number of locations (i,j) where we extract feature vectors.
- So, which pixels should we extract feature vectors from?
  - One approach: all pixels (leads to much slower running times).
  - Second approach: all edge pixels.
    - If 10% of image pixels are edge pixels, quadratic-time feature matching becomes 100 times faster.

Feature Matching: Complexity

- Another design choice that determines the number of extracted feature vectors (and thus the running time of feature matching):
  - The number of locations (i,j) where we extract feature vectors.
- So, which pixels should we extract feature vectors from?
  - One approach: all pixels (leads to much slower running times).
  - Second approach: all edge pixels.
    - If 10% of image pixels are edge pixels, quadratic-time feature matching becomes 100 times faster.
  - Third approach: **keypoint detection**.
    - Find pixels whose feature vectors are more likely to be stable and lead to good matches in other images.

Keypoints

- Terminology: you will see terms such as “interest points”, “corners”, “keypoints”.
  - For the context of choosing locations from where we will extract feature vectors, those three terms are mostly interchangeable.

Image **`roofs1`**

Image **`roofs2`**

Keypoints

- Main idea: we want to identify points whose feature vectors are the most informative.
- What makes a feature vector informative?
  - Does not change match in different images.
  - Does not give good matching scores with unrelated feature vectors.

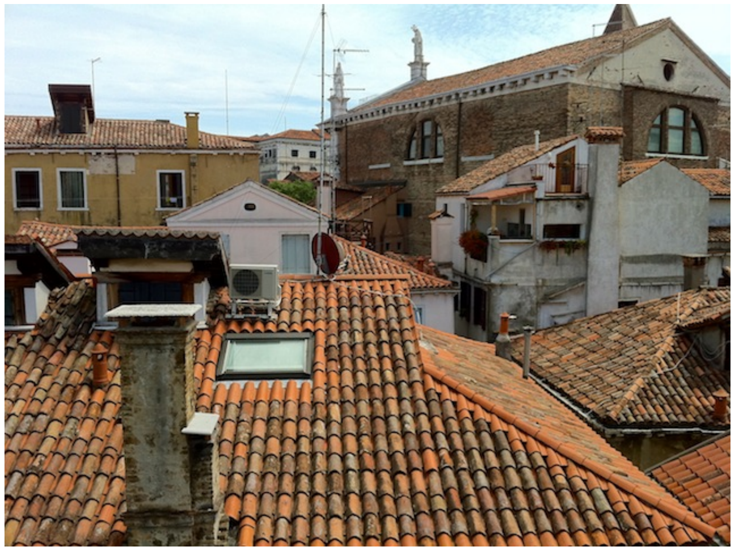


Image **`roofs1`**

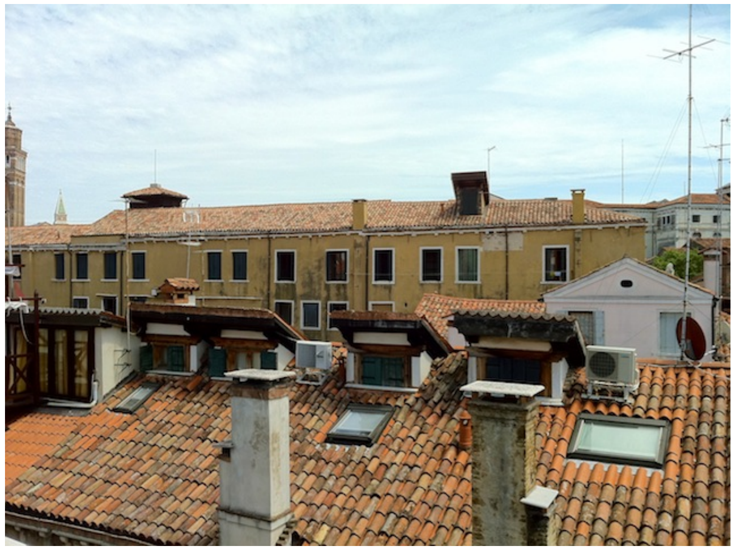


Image **`roofs2`**

Flat Regions Are Not "Interesting"

- Some image regions are "flat": they are composed of pixels mostly identical to each other.
  - Example: look at the indicated yellow regions.

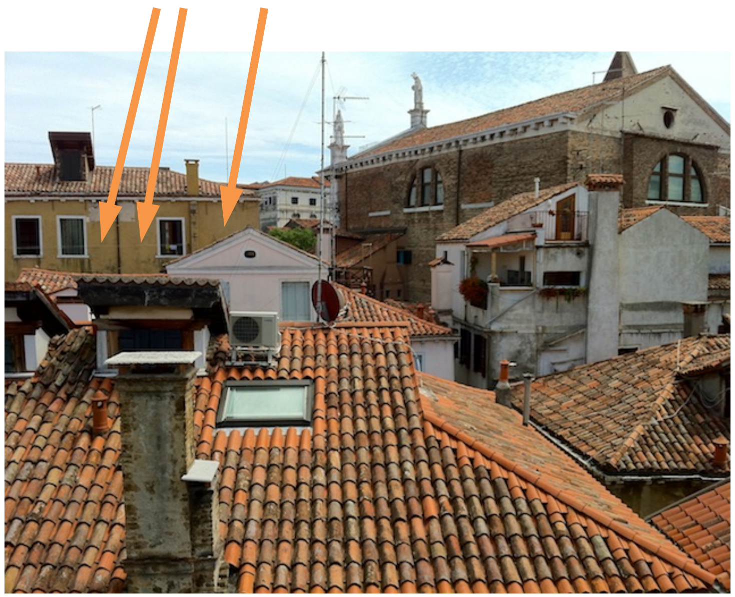


Image **`roofs1`**

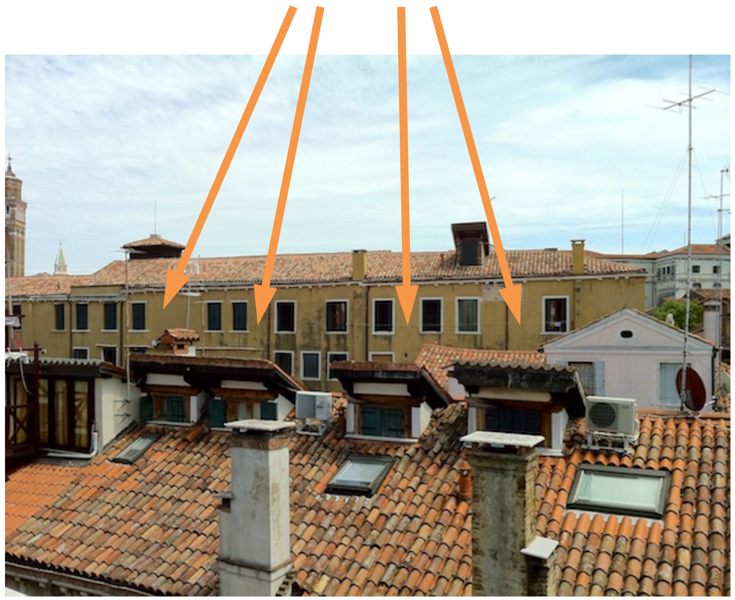


Image **`roofs2`**

Flat Regions Are Not "Interesting"

- Pixels within such regions get similar feature vectors.
- Feature vectors from such regions get lots of similar matches in the other image.
- Thus, these pixels are not useful for finding correspondences.

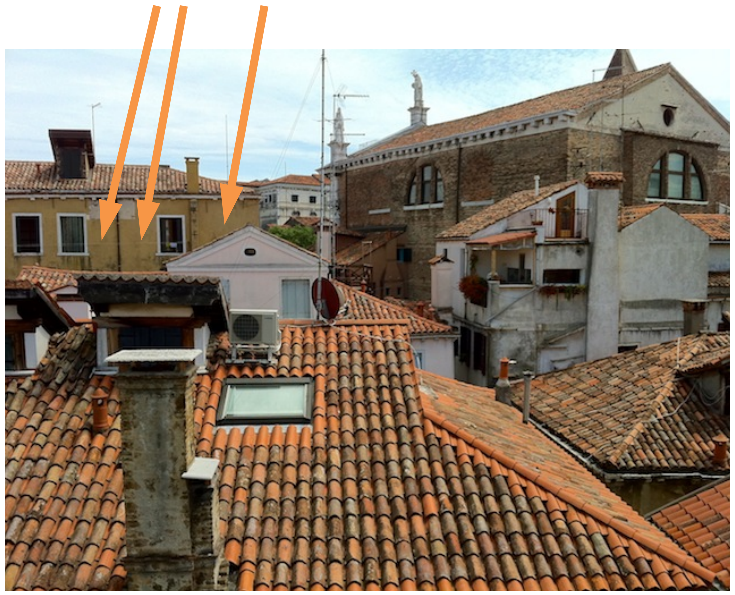


Image **`roofs1`**

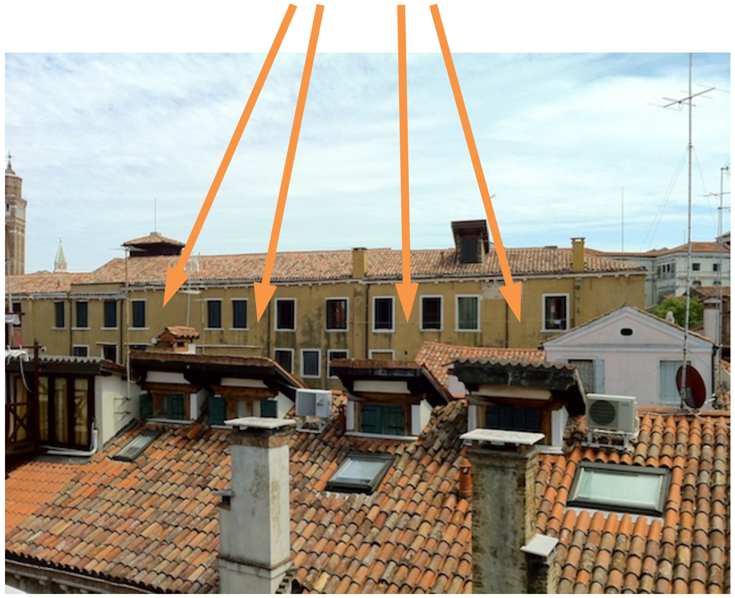


Image **`roofs2`**

Flat Regions Are Not “Interesting”

- How can we identify and reject such pixel locations?

“Flat” yellow regions

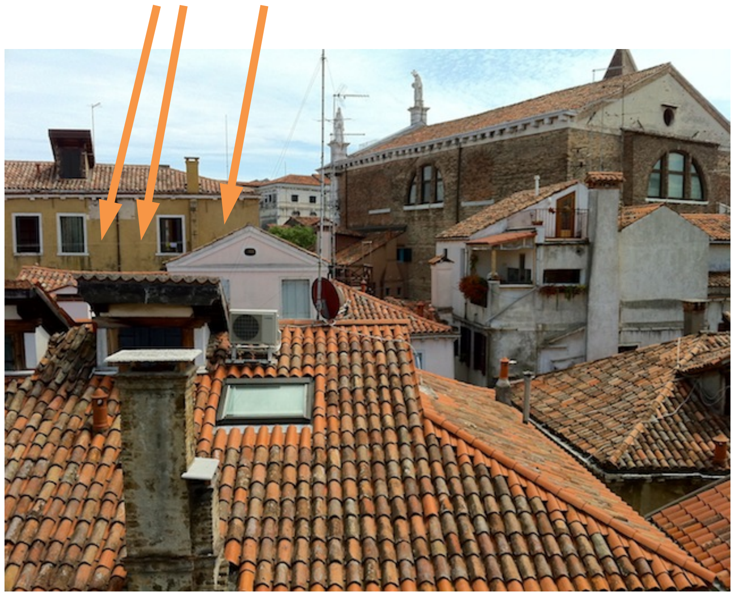


Image **`roofs1`**

“Flat” yellow regions

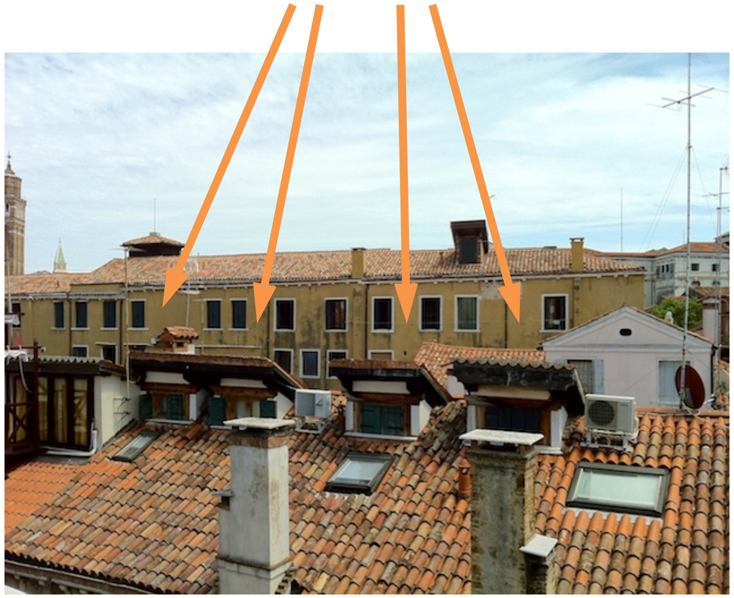


Image **`roofs2`**

Flat Regions Are Not “Interesting”

- How can we identify and reject such pixel locations?
- They have low gradient magnitudes.
- So, if we discard pixels with low gradient magnitudes, what are we left with?

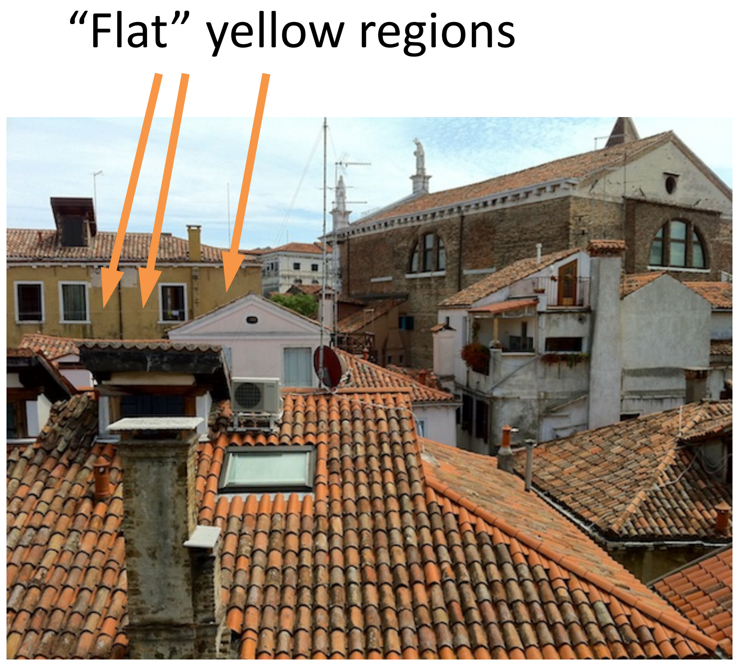


Image **`roofs1`**

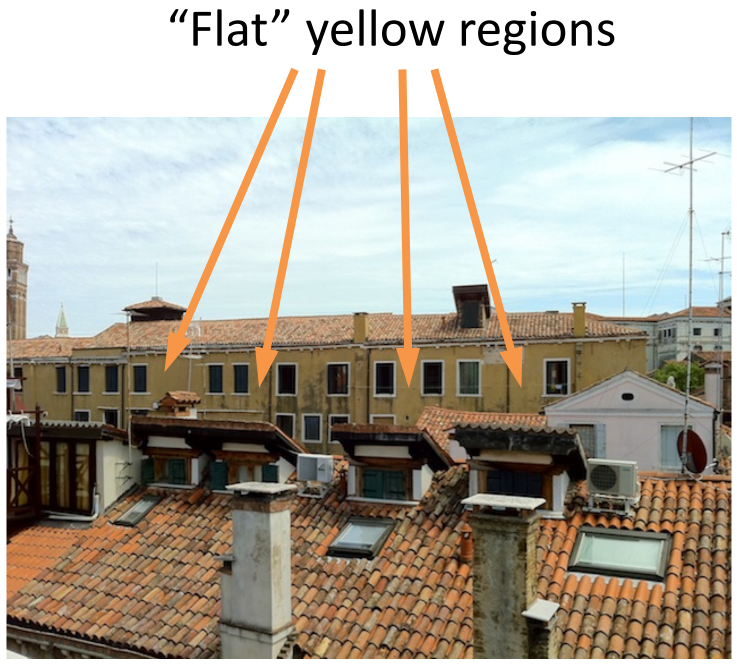


Image **`roofs2`**

Flat Regions Are Not “Interesting”

- So, it looks like a good idea to only consider edge pixels, where gradient magnitudes exceed a threshold.
- However, not all edges are equally useful (for the purposes of finding correspondences), regardless of their gradient norm.

“Flat” yellow regions

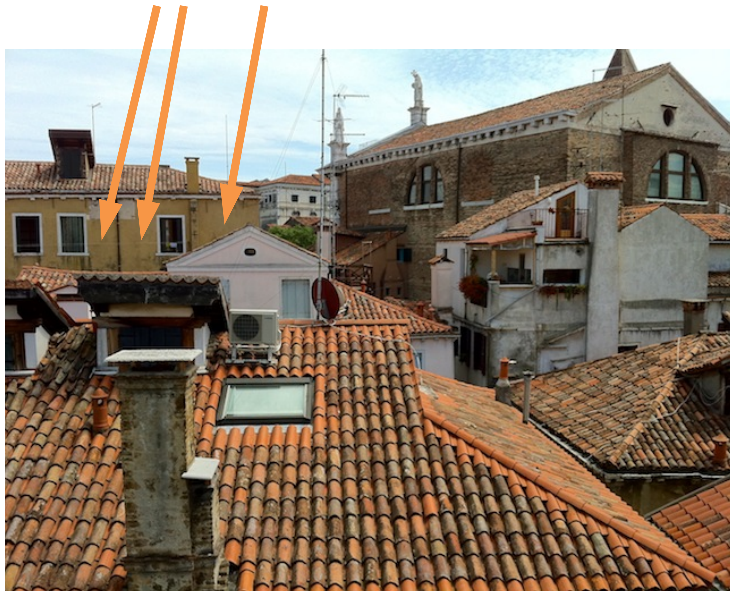


Image **`roofs1`**

“Flat” yellow regions

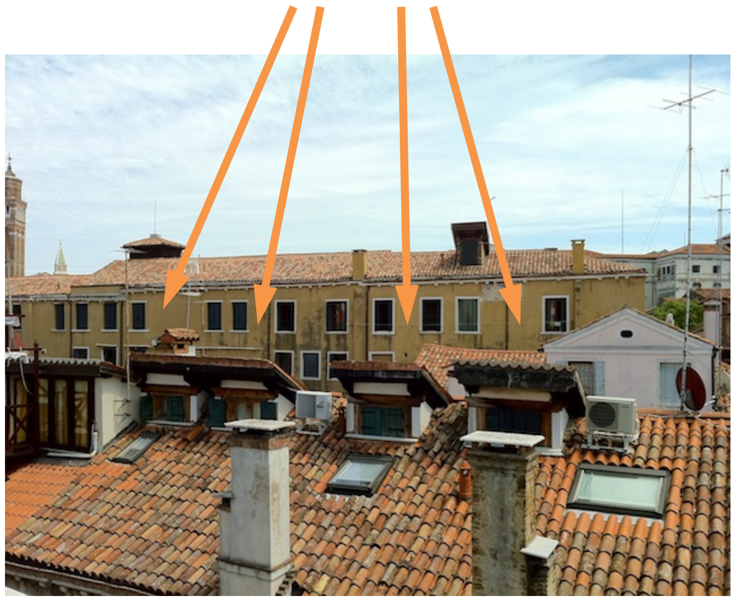
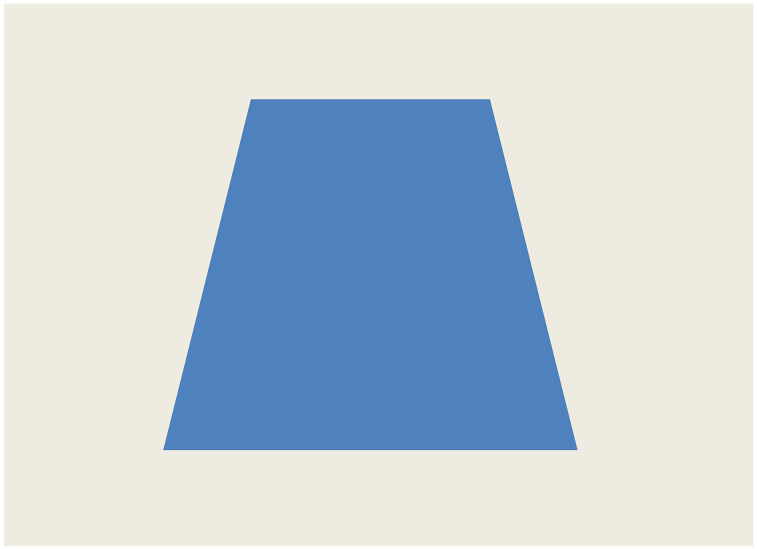


Image **`roofs2`**

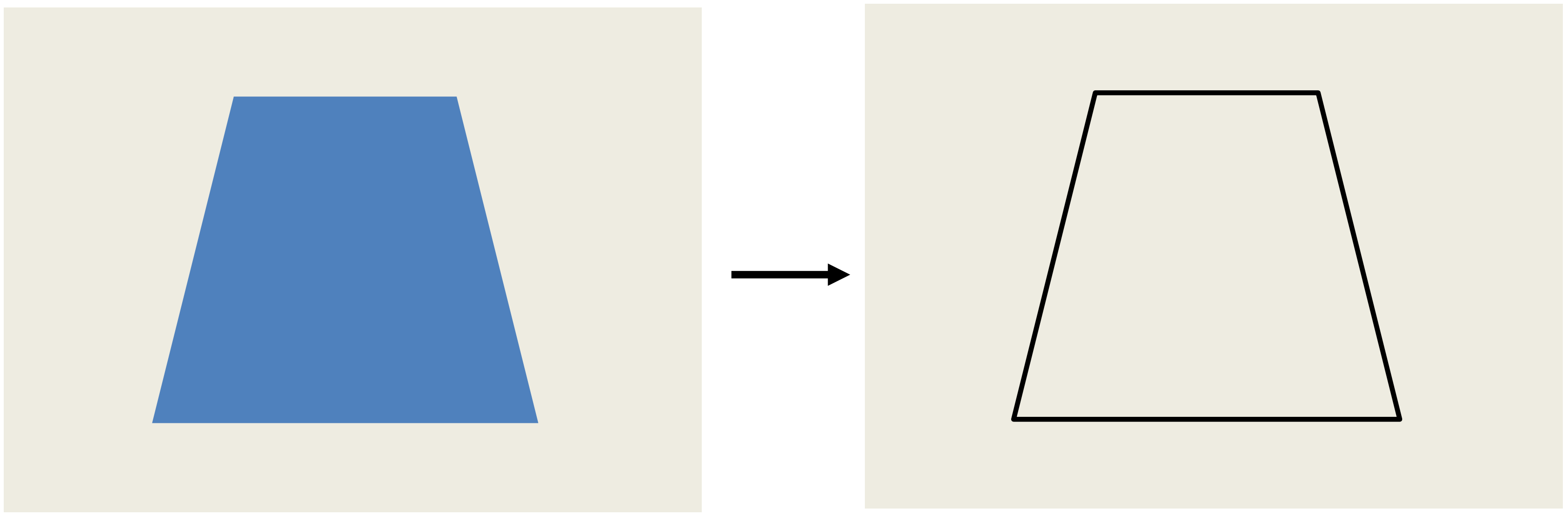
Stable Feature Locations

- Consider this image, showing a trapezoid.



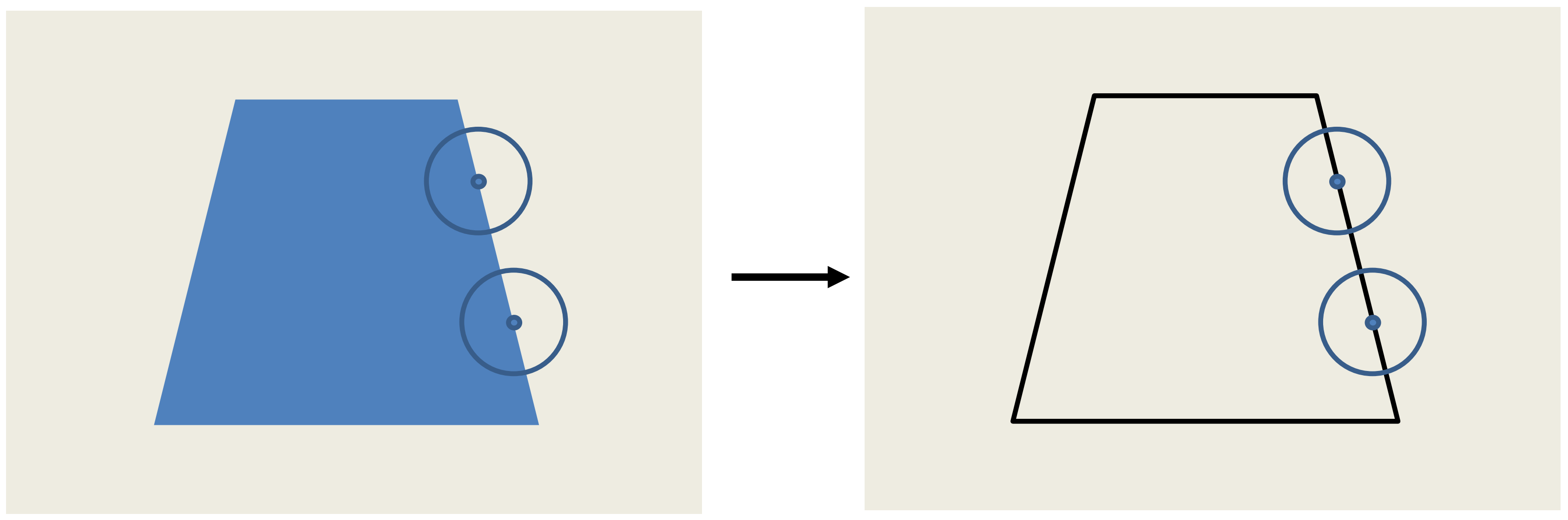
Stable Feature Locations

- We detect edge pixels. These are supposed to be good locations for extracting features. However…



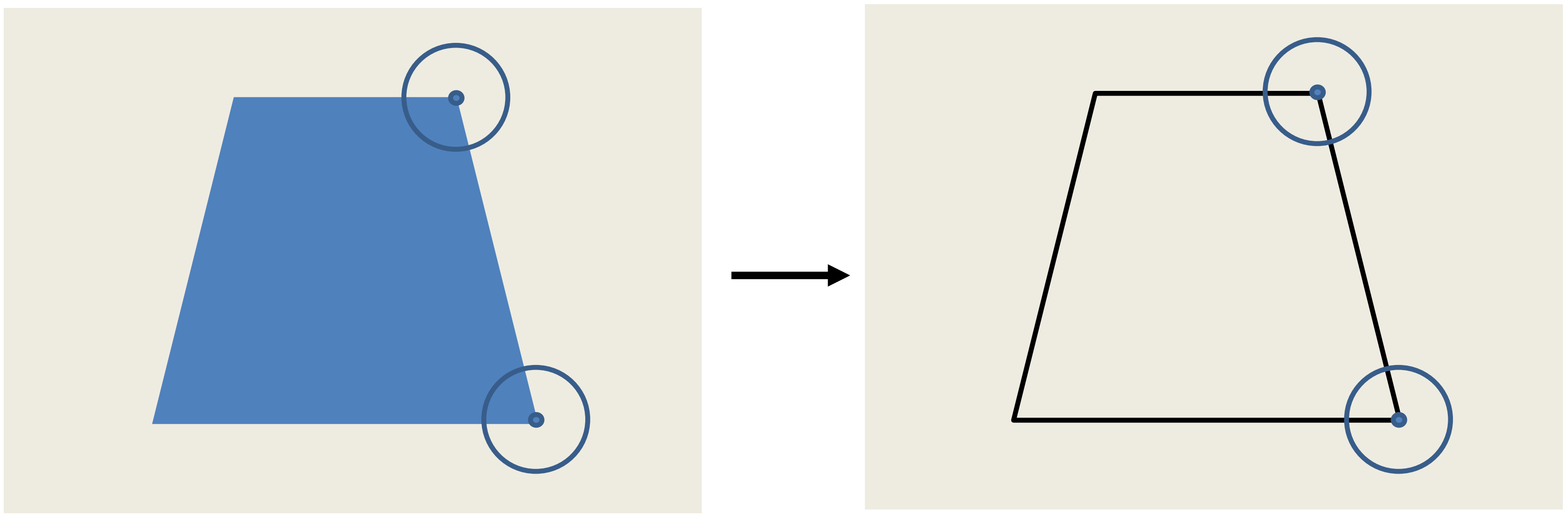
Stable Feature Locations

- We detect edge pixels. These are supposed to be good locations for extracting features. However…
- We note that many neighboring edge pixels have similar surrounding regions.
  - Moving along the edge does not change appearance much.



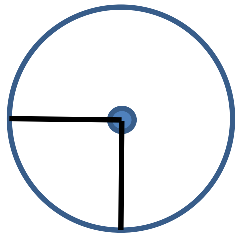
Stable Feature Locations

- What edge pixel locations have the most distinctive appearance?
- Corners!
  - The appearance of a corner region (with a corner at the center) is distinctive.
  - As we move away from the corner in any direction, appearance changes fast.



Appearance of a Corner

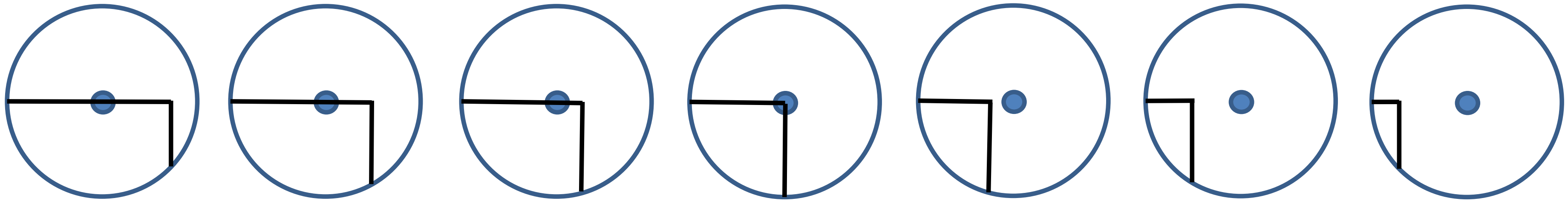
- Suppose that (i,j) is the vertex of a 90-degree corner.
- Here you see how the local region looks if we center at pixel (i,j).



center:
(i,j)

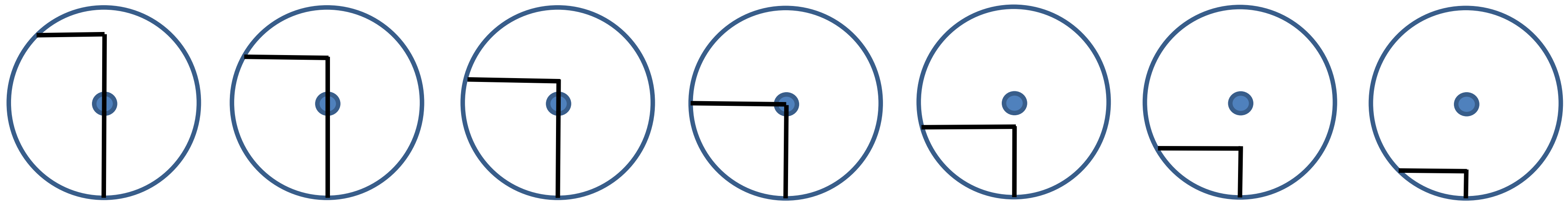
Appearance of a Corner

- Suppose that (i,j) is the vertex of a 90-degree corner.
- Here you see how the local region looks if we center at pixel (i,j).
- Here you see other local regions, centered at pixels on the same **row** as (i,j), and shifted by 1, 2, or 3 pixels horizontally.
- Clearly, the local region appearance changes as the center changes.
  - No other local region looks the same as the one centered at (i,j).



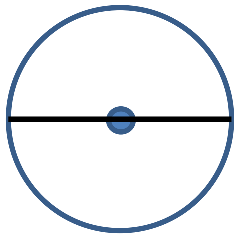
Appearance of a Corner

- Here we see local regions centered at pixels on the same **column** as (i,j), shifted by 1, 2, or 3 pixels verticall.
- Again, the local region appearance changes as the center changes.
  - No other local region looks the same as the one centered at (i,j).



Appearance of a Straight Edge

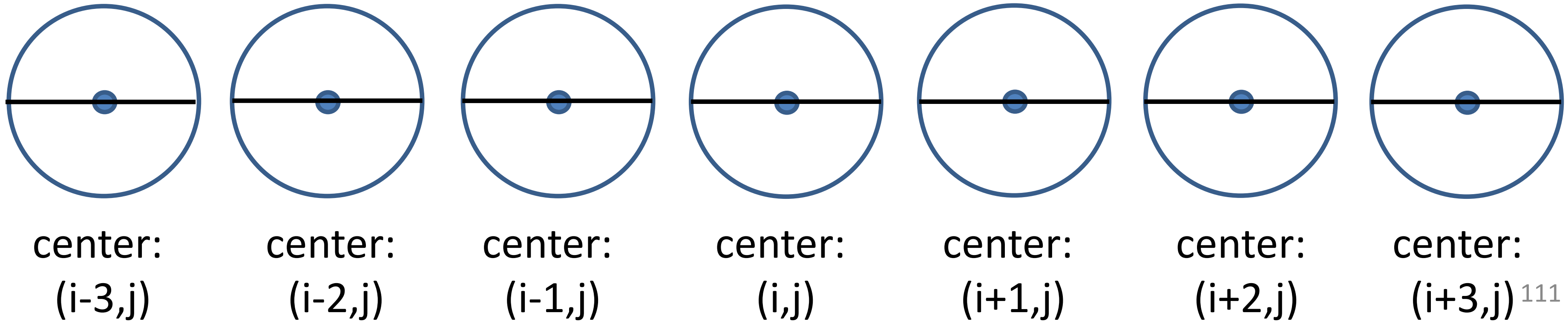
- Now, suppose that (i,j) is the vertex of a long straight horizontal edge.
- Here you see how the local region looks if we center at pixel (i,j).



center:
(i,j)

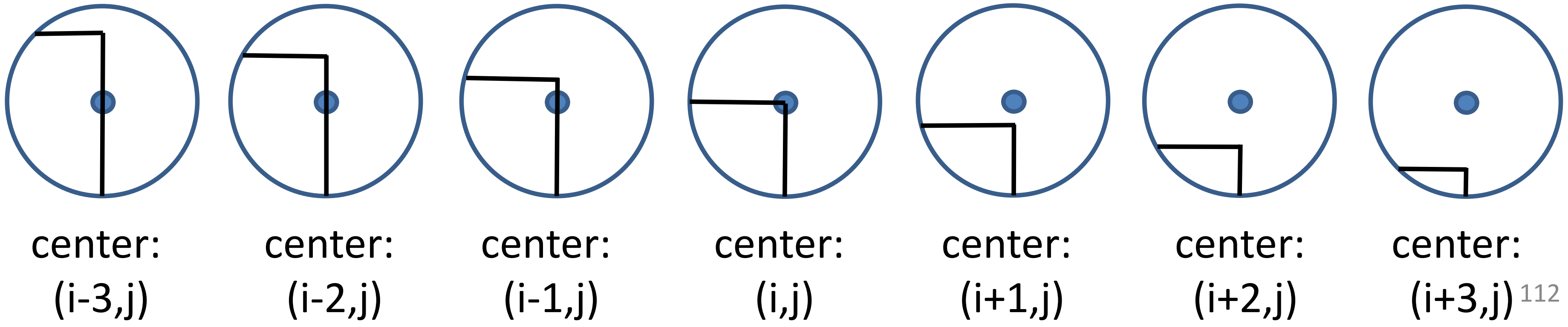
Appearance of a Straight Edge

- Now, suppose that (i,j) is the vertex of a long straight horizontal edge.
- Here you see how the local region looks if we center at pixel (i,j).
- Now we see other local regions, centered at pixels on the same **row** as (i,j), shifted by 1, 2, or 3 pixels horizontally from (i,j).
- Clearly, the local region appearance does **not** change.
  - The other local regions look the same as the one centered at (i,j).



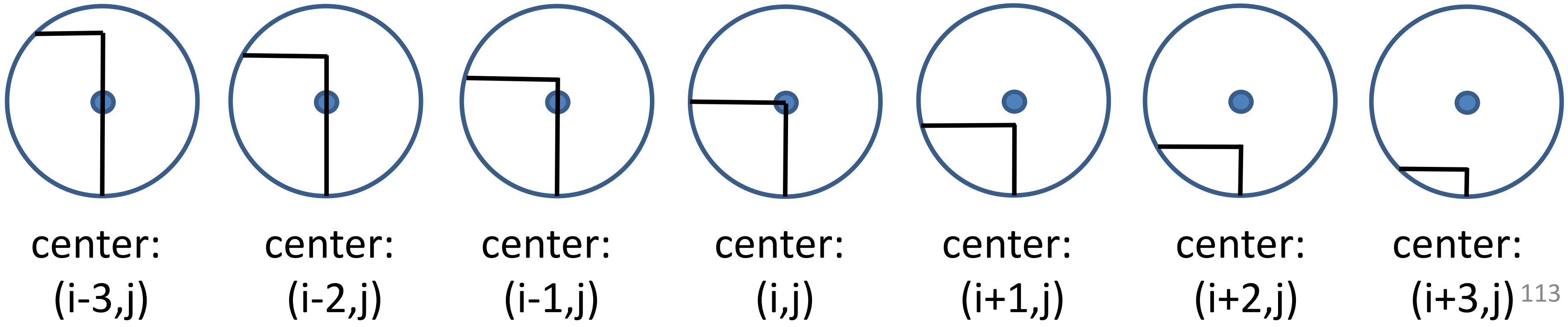
Advantages of Corners

- Obviously, these examples are contrived.
- We ignore other edge pixels that may be present in the region.
- However, this example shows the advantages of extracting features from corner pixels.
  - Features extracted from corner pixels are more distinctive.
  - They are expected to match well with features extracted from corresponding pixels in the other image.
  - They are expected to match poorly with pixels that are nearby but not exactly on the right spot.



Advantages of Corners

- In contrast, features extracted from flat regions cannot be localized very accurately.
  - They tend to match a relatively large region of pixels in the other image.
- Features extracted from straight edges can also be difficult to localize accurately.
  - Other pixels along the direction of the edge produce similar-looking regions and similar-looking feature vectors.



Harris Corner Detection

- How do we detect corner pixels?
- Harris corner detection is a popular approach.
- The Wikipedia article on this method is a useful reference:

  https://en.wikipedia.org/wiki/Harris_Corner_Detector

- The Wikipedia article derives a proof that corner pixels detected with this method are pixels with relatively distinctive regions (i.e., that look different than regions centered at neighboring pixels).

Harris Corner Detection: Derivatives

- To do Harris corner detection, we need to compute second-order image derivatives.
- As a reminder, what are first-order image derivatives?
  - I_x = imfilter(image, [-1, 0, 1], ’same’);
  - I_y = imfilter(image, [-1, 0, 1]’, ’same’);
- Second-order derivatives can be obtained from first-order derivatives:
  - I_{xx} = imfilter(I_x, [-1, 0, 1], ’same’);
  - I_{xy} = I_{yx} = imfilter(I_x, [-1, 0, 1]’, ’same’)
    = imfilter(I_y, [-1, 0, 1], ’same’)
  - I_{yy} = imfilter(I_y, [-1, 0, 1]’, ’same’);

Harris Corner Detection: Window

- To compute scores of "cornerness", we need to choose a parameter *h*, that specifies the half-width of the window we consider around each pixel.
- So, given a pixel location (i,j), the window W we consider is the set of all pixels in the rectangle:
  - From row i-h to row i+h.
  - From column j-h to column j+h.
- Then, for pixel (i,j), we define a matrix M as follows:

$$M = \sum_{(i',j')\in W} \begin{bmatrix} I_{xx}(i',j') & I_{xy}(i',j') \\ I_{xy}(i',j') & I_{yy}(i',j') \end{bmatrix}$$

Harris Corner Detection: Score

- Then, the score we give to location (i,j) is: $\frac{\det(M)}{\text{trace}(M)}$, where:
  - For simplicity, we write M as: $M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$
  - $\det(M)$ is the determinant of matrix M:

$$\det(M) = m_{11}m_{22} - m_{12}m_{21}$$

  - $\text{trace}(M)$ is the trace of matrix M:

$$\text{trace}(M) = m_{11} + m_{22}$$

Harris Corner Detection: Score

- An alternative score that is often used is:

$$\det(M) - \mathrm{k} * \mathrm{trace}(M)$$

where k is a manually picked constant (usually k is between 0.04 and 0.06).

Non-Maxima Suppression

- As we do in edge detection, we do non-maxima suppression before we produce the final output.
- For non-maxima suppression, we simply suppress any score that is not higher than its eight neighbors.
- The final result corresponds to pixels that:
  - Survived non-maxima suppression.
  - Got scores higher than a threshold (which we get to pick).

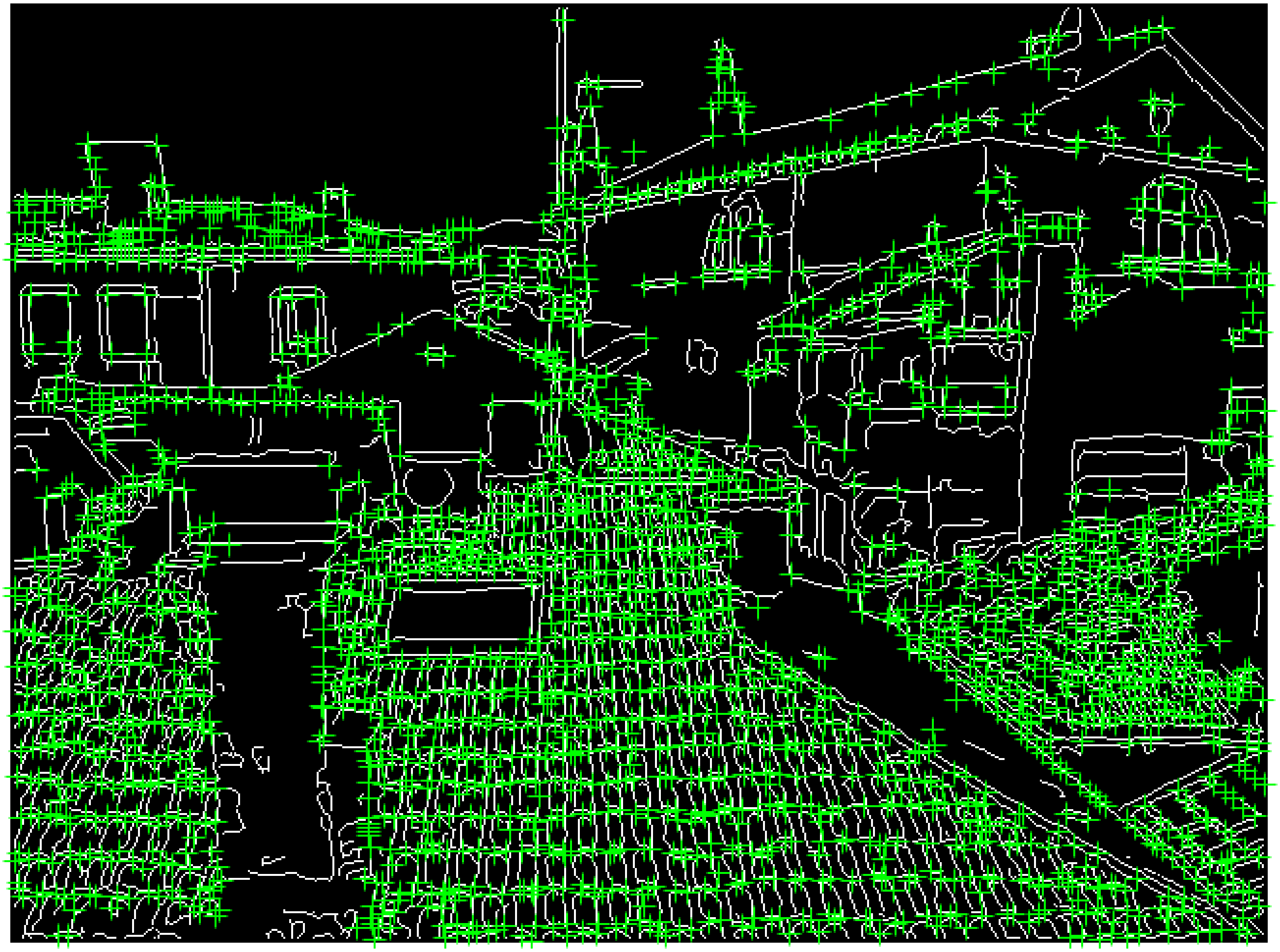
Using Matlab Implementation

- Harris corner detection is implemented in Matlab.
  - **`detectHarrisFeatures`** function in the computer vision toolbox.
- Try this code to see corner points superimposed on edge pixels:

```
a = read_gray('roofs1.jpg');
b = canny(a, 5);
c = detectHarrisFeatures(a); % built-in function
figure(1);
imshow(b, []);
hold on;
plot(c);
```

Using Matlab Implementation

- This is the result:



Using Matlab Implementation

```
a = read_gray('roofs1.jpg');
b = canny(a, 5);
c = detectHarrisFeatures(a); % built-in function
corners = c.Location;
```

- The above code shows how to get the actual corner locations.
- The `c.Location` variable is a two-column matrix.
  - Each row is of the format [x, y], containing the column and row of a corner pixel.
- Note: pixel co-ordinates are floats, not integers.
  - Interpolation was used to get subpixel accuracy.

Using Matlab Implementation

```
a = read_gray('roofs1.jpg');
b = canny(a, 5);
c = detectHarrisFeatures(a); % built-in function
corners = c.Location;
scores = c.Metric;
```

- The above code shows how to get the score of each corner location.
- The `c.Metric` variable is a one-column matrix.
  - `c.Metric(i)` contains the score for the corner whose location is stored on the i-th row of `c.Location`.

Scale Invariance

- So far, we have covered these topics regarding local features:
  - Different types: shape context, orientation histograms, HOGs.
  - Orientation invariance: rotate each local region based on dominant gradient orientation at that region.
  - Keypoint selection: corner pixels provide more stable features, that can be used to establish more accurate correspondences between images.
- There is one last topic to address: scale invariance.
  - How can we extract feature vectors that are (at least mostly) invariant to changes in image scale?

Alternative: Multi-Scale Search

- Instead of trying to define scale-invariant features, we could simply do multi-scale search.
  - For image 1, extract features at some specific scale.
  - For image 2, extract features at many different scales.
  - Match all features of image 1 with all features of image 2.
- Disadvantages:
  - We extract more features from image 2, compared to single-scale search, and thus feature matching takes longer.
  - More features mean more opportunities for false matches.
    - In pattern recognition this is always an issue: the more candidate matches we have, the more likely the system is to identify wrong matches.

Scale-Invariant Features

- By defining scale-invariant features, we achieve a reasonable compromise, compared to the extremes of single-scale search and multi-scale search.
- We extract more features than we would for single-scale search, but fewer features than we would for multi-scale search.
- This way:
  - Feature matching is faster compared to multi-scale search.
  - There are fewer candidates for false matches, compared to multi-scale search.

How to Achieve Scale Invariance

- How did we achieve orientation invariance?
  - For each region, we looked for “special” orientations.
  - This leads to usually one “special” orientation per region, occasionally more.
  - We extract one feature vector for each of those “special orientations”.
- We get fewer feature vectors than we would if we simply extracted feature vectors for all orientations.
- At the same time, if the region looks similar (but rotated) in another image, the feature vectors for those “special” orientations should be similar.
- We follow a similar approach for scale invariance: for each region, we look for some “special” scales.
- Key question: how do we find “special” scales?

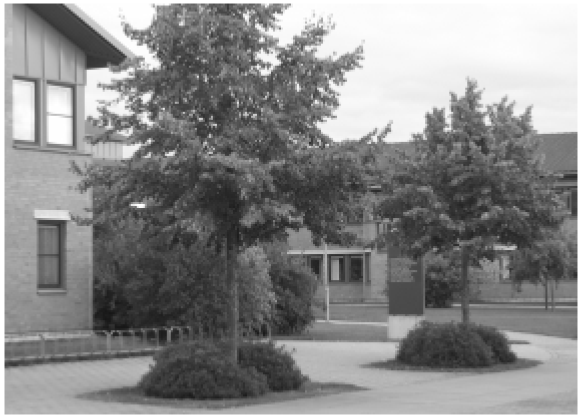
Scale Space

- We have seen before that we can blur an image by convolving it with a Gaussian kernel.
  - The higher the standard deviation of the Gaussian kernel, the more blurry the result looks.
- Suppose that we are given a grayscale image I.
- We denote by G_σ the Gaussian kernel with standard deviation σ.
- We denote the convolution of I by G_σ as I * G_σ.
- Then, the **scale-space** representation L of image I is defined as: $L(\sigma) = \mathrm{I} * \mathrm{G}_\sigma$.
- We denote by $L(i, j, \sigma)$ the value of $L(\sigma)$ at pixel (i,j).

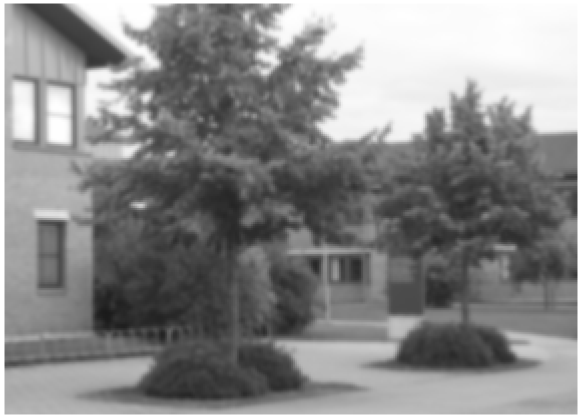
Scale Space: Example

- An example of the scale space representation of an image, from Wikipedia:

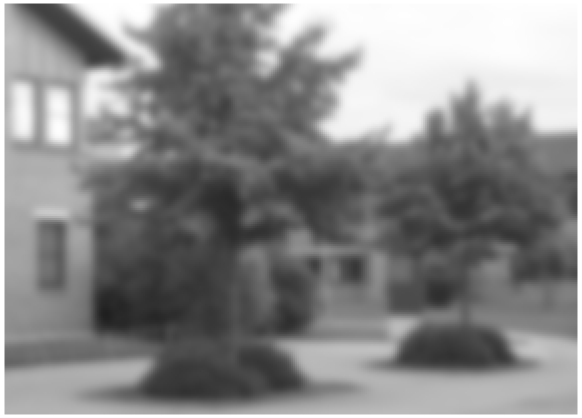
σ=0.5 (original image)

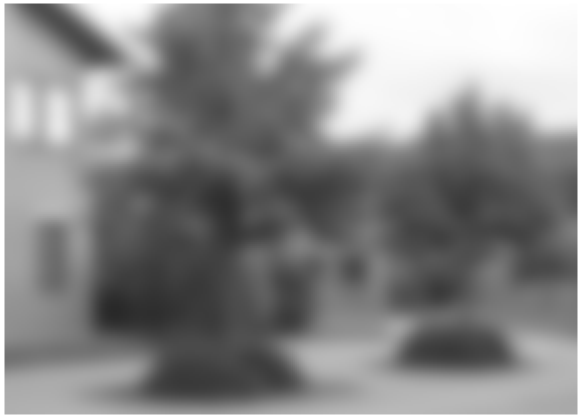


σ=1



σ=2





σ=4

σ=8

σ=16

Standard Deviation and Image Size

- Blurring an image discards information.
  - The higher the standard deviation σ of the Gaussian filter, the more information we discard.
- Reducing the image size also discards information.
  - The more we reduce the image size, the more information we discard.
- From the field of signal processing, it turns out that these two operations correspond to each other.
  - Original size: $\sigma = 0.5$ (this is just an assumption).
  - Half size: $\sigma = 1$.
  - Quarter size: $\sigma = 2$.
  - Eighth size: $\sigma = 4$… (and so on).

Standard Deviation and Image Size

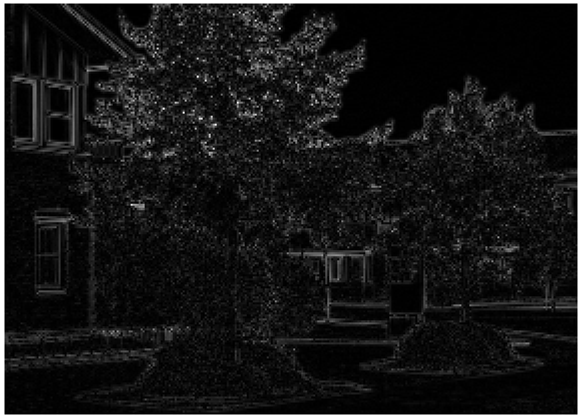
- From the field of signal processing, it turns out that these two operations correspond to each other.
  - Original size: σ = 0.5 (this is just an assumption).
  - Half size: σ = 1.
  - Quarter size: σ = 2.
  - Eighth size: σ = 4… (and so on).
- That is why we are thinking of different standard deviations corresponding to different image scales.
  - That is why we use the term “scale space” for the images we define as $L(\sigma) = \mathrm{I} * \mathrm{G}_{\sigma}$, using different values of σ.

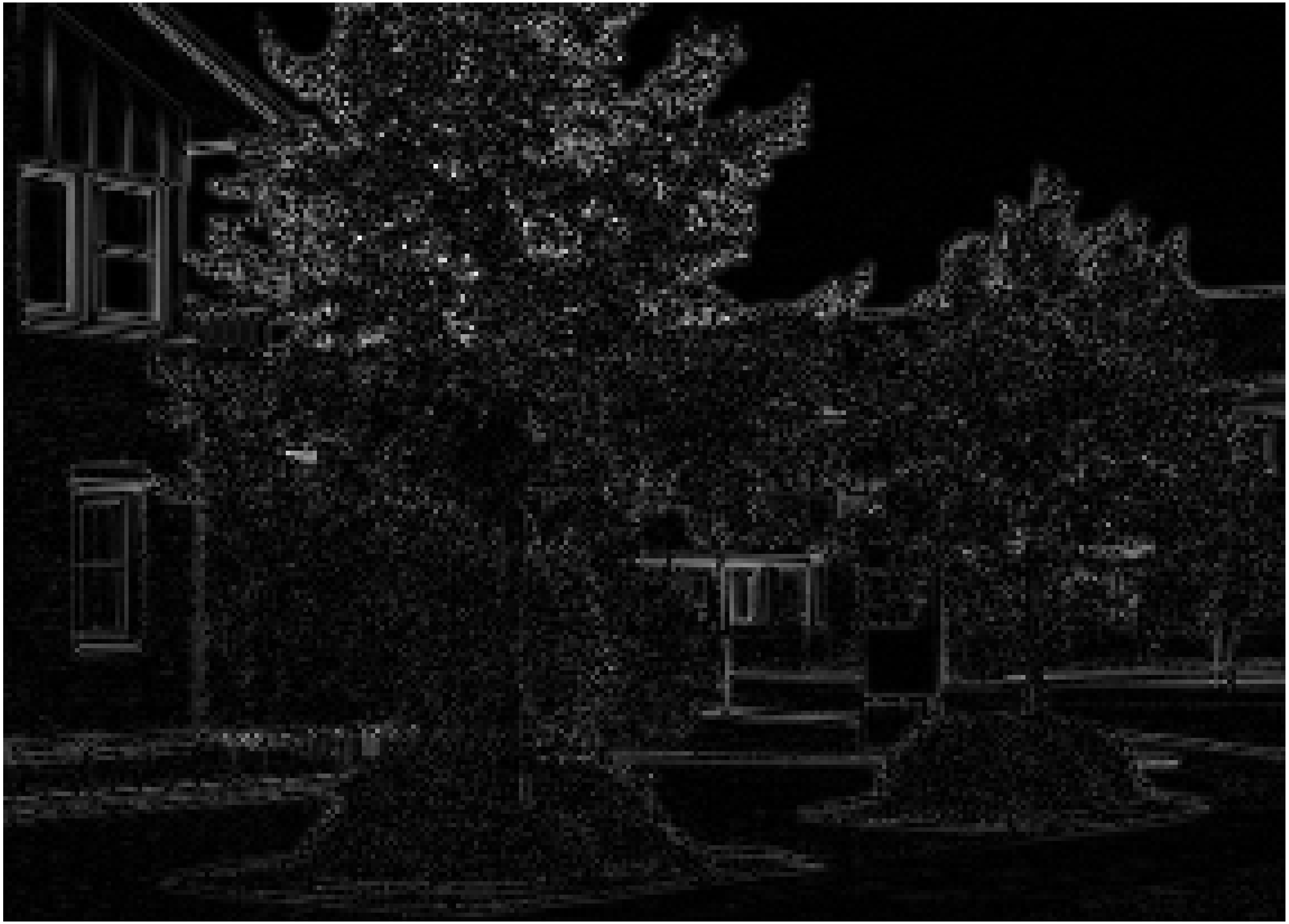
Differences of Gaussians

- As we saw a couple of slides ago:
  - The **scale-space** representation L of image I is defined as: $L(\sigma) = \mathrm{I} * \mathrm{G}_{\sigma}$.
- Then, we can define a difference-of-Gaussians image $D(\sigma_1, \sigma_2) = L(\sigma_2) - L(\sigma_1)$.
- In $D(\sigma_1, \sigma_2)$, what do high absolute values mean?

img1 (σ=0.5)

img2 (σ=1)

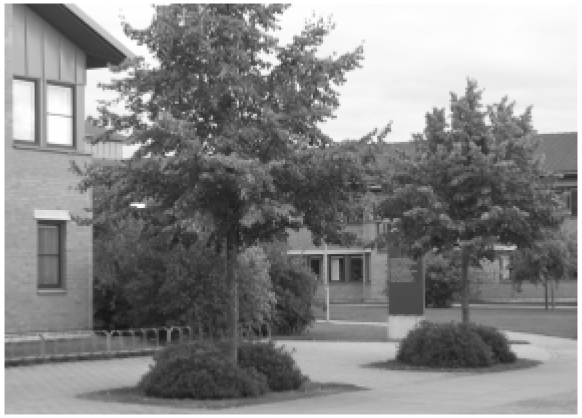

abs($D(0.5,1)$)

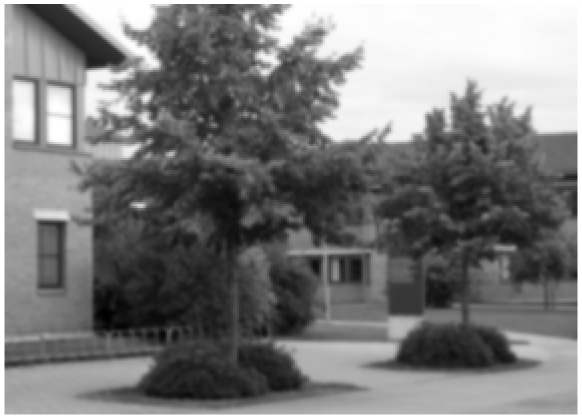


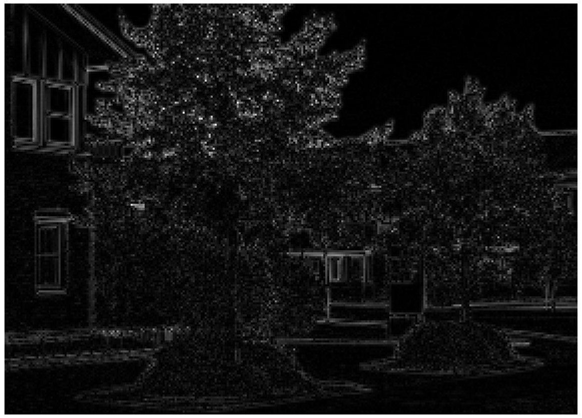
$\mathrm{abs}(D(0.5,1))$

Differences of Gaussians

- We assume that $\sigma_1 < \sigma_2$.
  - Then, $L(\sigma_2)$ is a blurrier version of $L(\sigma_1)$.
- In $D(\sigma_1, \sigma_2)$, high absolute values correspond to areas with high gradient norms in $L(\sigma_1)$.
  - Why?


img1 (σ=0.5)


img2 (σ=1)

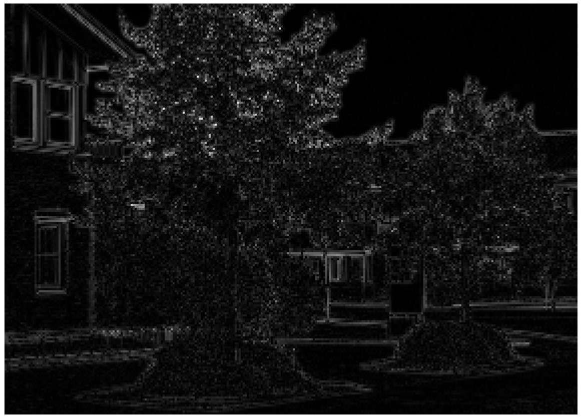

abs($D(0.5,1)$)

Differences of Gaussians

- If a region is flat in $L(\sigma_1)$, it will remain flat in $L(\sigma_2)$.
  - Therefore, $L(\sigma_1)$ and $L(\sigma_2)$ will look similar in that region.
  - Therefore, the values for that region in $D(\sigma_1, \sigma_2)$ will be close to 0.
- If a region has high gradient norms in $L(\sigma_1)$, it means that there are sudden changes in neighboring values in that region.
  - Those changes get blurred and become smoother in $L(\sigma_2)$.
  - Thus, the region looks different in $L(\sigma_2)$ than in $L(\sigma_1)$.
  - Therefore, the corresponding values in $D(\sigma_1, \sigma_2)$ will be far from 0.

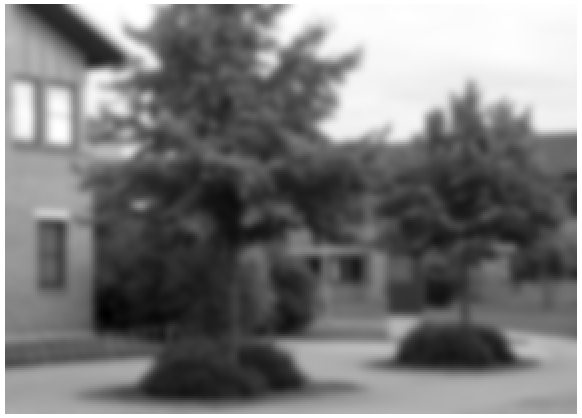
img1 (σ=0.5)

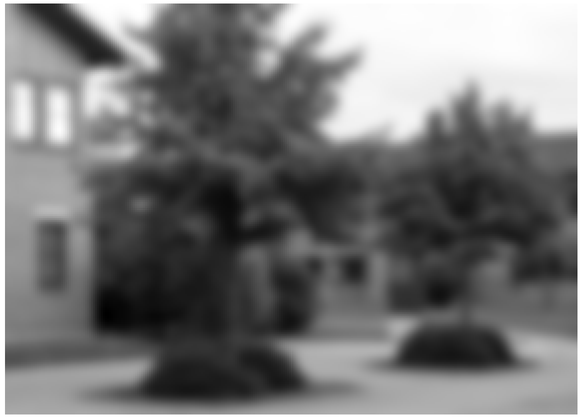
img2 (σ=1)

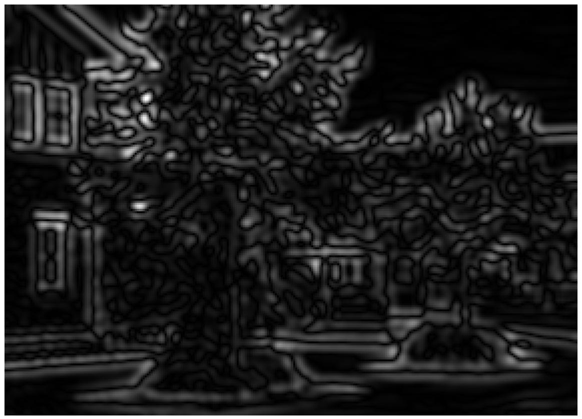

abs($D(0.5,1)$)

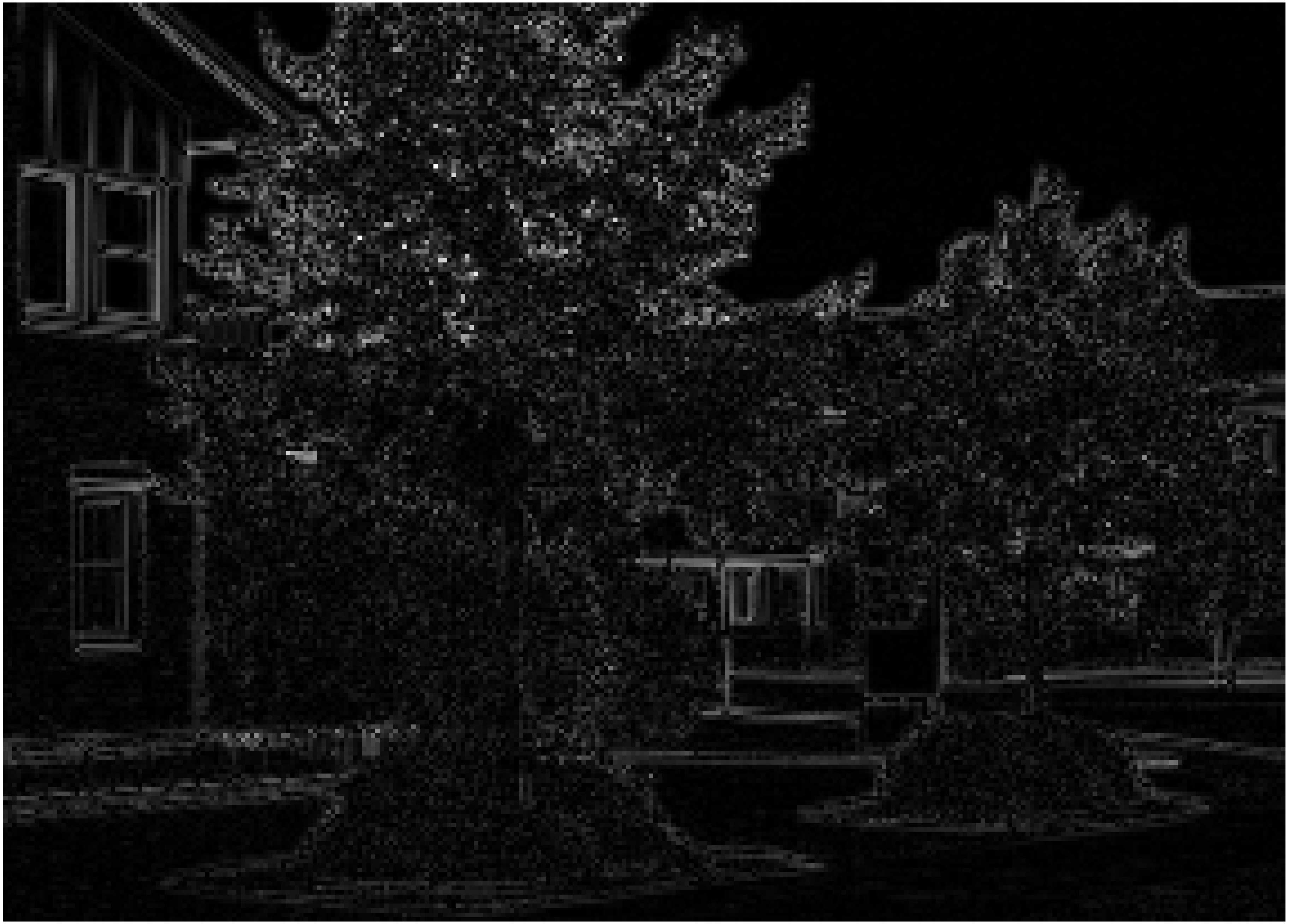
Differences of Gaussians

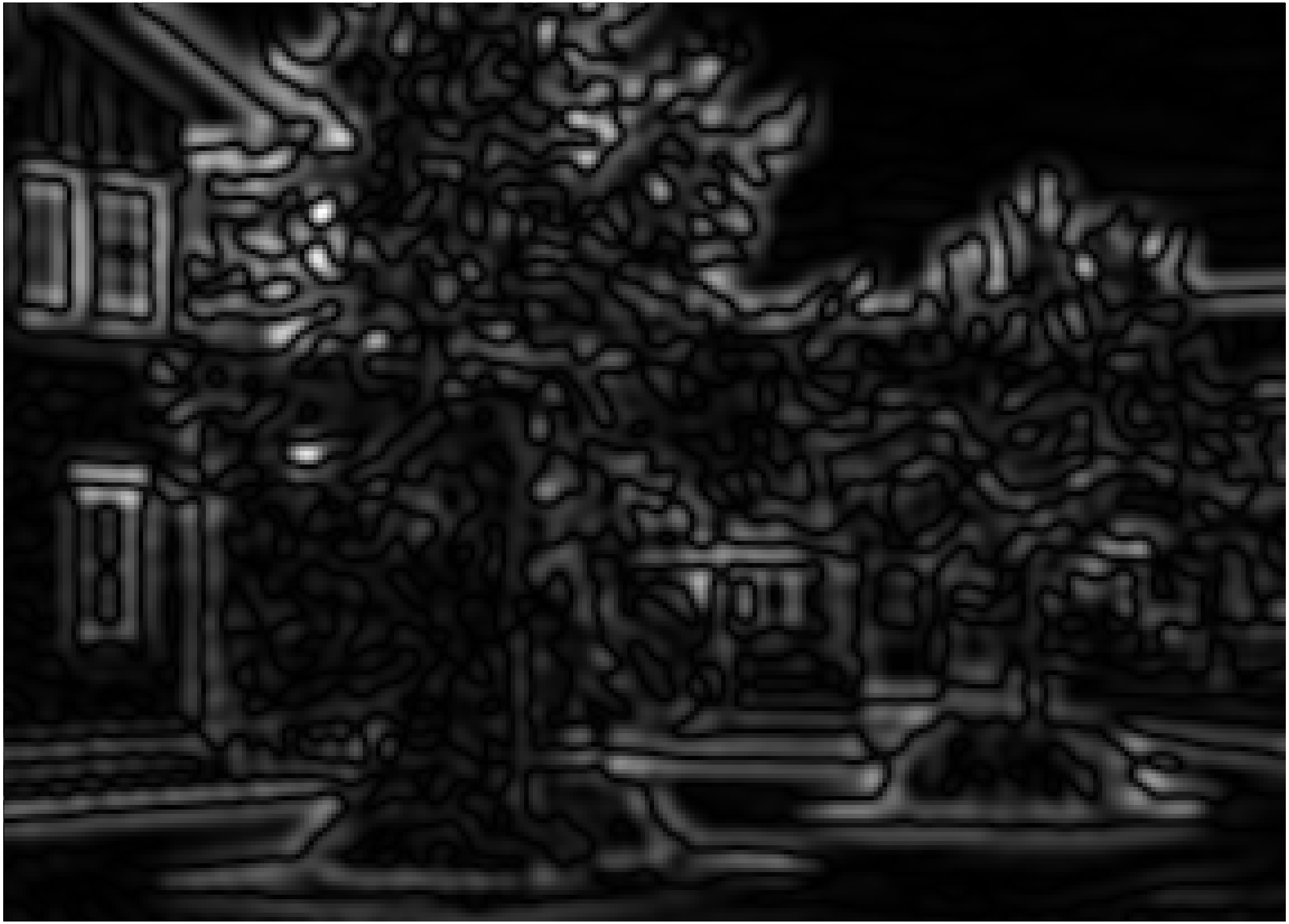
- If we increase σ_1 and σ_2, then $D(\sigma_1, \sigma_2)$ is the difference of blurrier versions of the original image.
- Then, small-scale details (like intensity differences in the tree regions) have been more blurred.
- High absolute values of $D(\sigma_1, \sigma_2)$ correspond to larger-scale changes in the original image (like transitions from tree regions to sky region).


img1 (σ=2)


img2 (σ=3)


abs($D(2,3)$)


abs($D(0.5,1)$)


abs($D(2,3)$)

Derivative in Scale Direction

- Consider $D(\sigma - \varepsilon, \sigma + \varepsilon)$, where ε is a relatively small value.
- Let (i,j) be a pixel location.
- We denote by $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$ the value of $D(\sigma, \sigma + \varepsilon)$ at pixel location (i,j).
- This value can be thought of as the derivative of $L(i, j, \sigma)$ in the direction of σ. Why?

Derivative in Scale Direction

- Consider $D(\sigma - \varepsilon, \sigma + \varepsilon)$, where ε is a relatively small value.
- Let (i,j) be a pixel location.
- We denote by $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$ the value of $D(\sigma, \sigma + \varepsilon)$ at pixel location (i,j).
- This value can be thought of as the derivative of $L(i, j, \sigma)$ in the direction of σ. Why?
- Derivative in x direction: $L(i, j + 1, \sigma) - L(i, j - 1, \sigma)$.
- Derivative in y direction: $L(i + 1, j, \sigma) - L(i - 1, j, \sigma)$.
- Derivative in σ direction $L(i, j, \sigma + \varepsilon) - L(i, j, \sigma - \varepsilon)$.

Derivative in Scale Direction

- Consider $D(\sigma - \varepsilon, \sigma + \varepsilon)$, where ε is a relatively small value.
- Let (i,j) be a pixel location.
- We denote by $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$ the value of $D(\sigma, \sigma + \varepsilon)$ at pixel location (i,j).
- This value can be thought of as the derivative of $L(i, j, \sigma)$ in the direction of σ. Why?
- Derivative in x direction: $L(i, j + 1, \sigma) - L(i, j - 1, \sigma)$.
- Derivative in y direction: $L(i + 1, j, \sigma) - L(i - 1, j, \sigma)$.
- Derivative in σ direction $L(i, j, \sigma + \varepsilon) - L(i, j, \sigma - \varepsilon)$.
- Note: by definition, $D(\sigma - \varepsilon, \sigma + \varepsilon) = L(\sigma + \varepsilon) - L(\sigma - \varepsilon)$.
- Therefore, the derivative in σ direction can be written as $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$.

Derivative in Scale Direction

As we just saw:

- Derivative in x direction: $L(i, j + 1, \sigma) - L(i, j - 1, \sigma)$.
- Derivative in y direction: $L(i + 1, j, \sigma) - L(i - 1, j, \sigma)$.
- Derivative in σ direction: $L(i, j, \sigma + \varepsilon) - L(i, j, \sigma - \varepsilon)$.

Question:

- Why do we use differences of 1 for the x and y directions, and differences of ε (and not 1) for the σ direction?

Derivative in Scale Direction

As we just saw:

- Derivative in x direction: $L(i, j + 1, \sigma) - L(i, j - 1, \sigma)$.
- Derivative in y direction: $L(i + 1, j, \sigma) - L(i - 1, j, \sigma)$.
- Derivative in σ direction: $L(i, j, \sigma + \varepsilon) - L(i, j, \sigma - \varepsilon)$.

Question:

- Why do we use differences of 1 for the x and y directions, and differences of ε (and not 1) for the σ direction?
  - Because pixel locations are integers, whereas σ can be any real number.

Finding Special Scales for (i,j)

- To achieve orientation invariance, we found a way to define some characteristic orientation (or orientations) for the region centered at (i,j).
  - Even if the image gets rotated, those characteristic orientations allow us to rotate the region back to a canonical orientation.
- Now we will define some characteristic scales for the region centered at (i,j).
  - Even if the image gets scaled (up or down), those characteristic scales allow us to scale the region back to a canonical scale.

Finding Special Scales for (i,j)

- The special scales for the region centered at (i,j) are simply σ values that produce local extrema (i.e., local minima and local maxima) of $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$.
  - We are talking about local extrema in the direction of σ.
  - For example, $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$ is a local maximum in the direction of σ if it is greater than $D(i, j, \sigma' - \varepsilon, \sigma' + \varepsilon)$ for all σ' in some local vicinity of σ.

Finding Special Scales for (i,j)

- The special scales for the region centered at (i,j) are simply σ values that produce local extrema (i.e., local minima and local maxima) of $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$.
  - We are talking about local extrema in the direction of σ.
  - For example, $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$ is a local maximum in the direction of σ if it is greater than $D(i, j, \sigma' - \varepsilon, \sigma' + \varepsilon)$ for all σ' in some local vicinity of σ.
- Question: how do we search for such local extrema?
  - Values σ are real numbes. We can not check all possible values. Therefore we …???

Finding Special Scales for (i,j)

- The special scales for the region centered at (i,j) are simply σ values that produce local extrema (i.e., local minima and local maxima) of $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$.
  - We are talking about local extrema in the direction of σ.
  - For example, $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$ is a local maximum in the direction of σ if it is greater than $D(i, j, \sigma' - \varepsilon, \sigma' + \varepsilon)$ for all σ' in some local vicinity of σ.
- Question: how do we search for such local extrema?
  - Values σ are real numbes. We can not check all possible values. Therefore we sample values of σ.

Sampling Values of σ

- A common approach is to define a sequence $\sigma_0, \sigma_1, \sigma_2, \ldots$ as follows:
  - Assume that the original image corresponds to $\sigma = 0.5$.
  - So, we define $\sigma_0 = 0.5$.
  - Pick a number s so that the value of σ doubles every s samples.
    - So, $\sigma_s = 1, \sigma_{2s} = 2, \sigma_{3s} = 4, \sigma_{4s} = 8$, and so on.
    - For example, if $s = 3$, we get: $\sigma_3 = 1, \sigma_6 = 2, \sigma_9 = 4, \sigma_{12} = 8$.
  - We pick the other values so that sequence σ_i increases exponentially: $\sigma_n = k * \sigma_{n-1}$.
  - In order for the sequence to double every s steps, it has to be that $k = 2^{\frac{1}{s}}$.

Sampling Values of σ

- We said before that the special scales for the region centered at (i,j) are σ values that produce local extrema of $D(i, j, \sigma - \varepsilon, \sigma + \varepsilon)$.
- Once we define our sequence of samples $\sigma_0, \sigma_1, \sigma_2, \ldots$, we find such extrema by simply finding local extrema in the sequence

$$D(i, j, \sigma_0, \sigma_1), D(i, j, \sigma_1, \sigma_2), D(i, j, \sigma_2, \sigma_3), \ldots$$

SIFT Features, Brief Overview

- SIFT features are a very popular method for extracting local features from an image.
  - Reference: Lowe, David G. "Distinctive image features from scale-invariant keypoints." *International Journal of Computer Vision,* 60, no. 2 (2004): 91-110.
- We will not describe the full method in this class.
- However, it is based on the concepts we have discussed:
  - Identify keypoints in the image.
  - For each keypoint, identify dominant orientations.
  - For each keypoint, identify scales as local extrema in scale space.
  - For each keypoint, at each of its dominant orientations and scales:
    - Extract a feature vector from the region around that keypoint, after we rotate and scale that region based on the chosen orientation and scale.

SIFT Features, Brief Overview

- The feature vector that is extracted by SIFT is pretty similar to a HOG feature vector, i.e., it is a concatenation of orientation histograms.
  - There are some minor differences in the implementation, that are described in the paper.