CS 534: Computer Vision
Texture

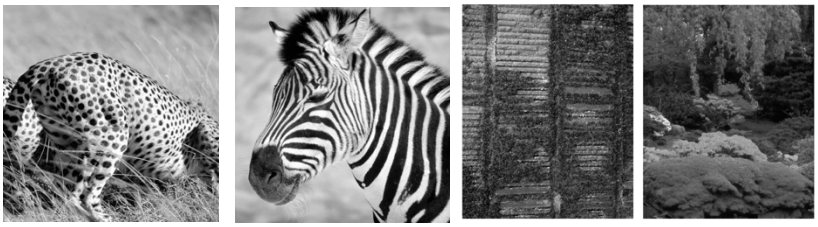
Ahmed Elgammal
Dept of Computer Science
Rutgers University

Outlines

- Finding templates by convolution
- What is Texture
- Co-occurrence matrices for texture
- Spatial Filtering approach
- Multiresolution processing, Gaussian Pyramids and Laplacian Pyramids
- Gabor filters and oriented pyramids
- Texture Synthesis

Texture

- What is texture ? Easy to recognize hard to define
  - Views of large number of small objects: grass, foliage, brush, pebbles, hair
  - Surfaces with patterns: spots, stripes, wood, skin
- Texture consists of organized patterns of quite regular sub-elements.
- Whether an effect is referred to as texture or not depends on the scale at which it is viewed.

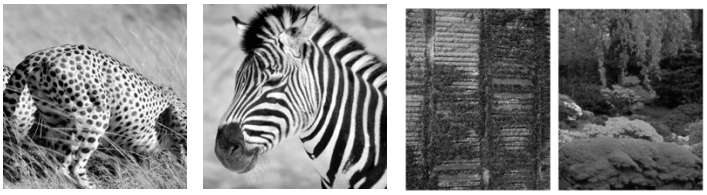


CS 534 – Texture - 3

Texture

Problems related to Texture:

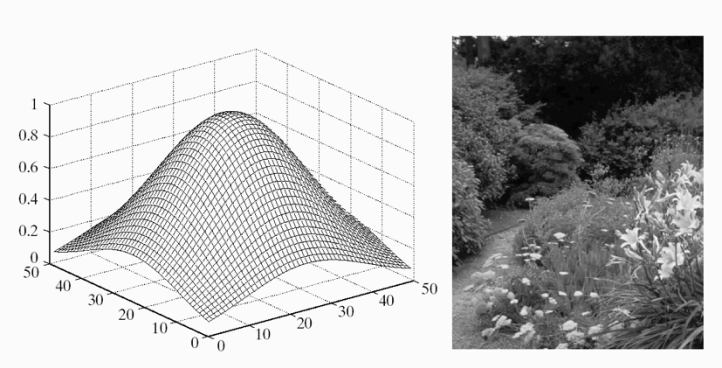
- Texture analysis: how to represent and model texture
- Texture segmentation: segmenting the image into components within which the texture is constant
- Texture synthesis: construct large regions of texture from small example images
- Shape from texture: recovering surface orientation or surface shape from image texture.



CS 534 – Texture - 4

Shape from Texture

- Texture looks different depending on the viewing angle.
- Texture is a good cue for shape and orientation
- Humans are very good at that.



Notice how the change in pattern elements and repetitions is the main difference between different textured surfaces (the plants, the ground, etc.)

FIGURE 6.1: Although texture is difficult to define, it has some important and valuable properties. In this image, there are many repeated elements (some leaves form repeated "spots"; others, and branches, form "bars" at various scales; and so on). Our perception of the material is quite intimately related to the texture (what would the surface feel like if you ran your fingers over it? what is soggy? what is prickly? what is smooth?). Notice how much information you are getting about the type of plants, their shape, the shape of free space, and so on, from the textures. *Geoff Brightling © Dorling Kindersley, used with permission.*

Representing Texture

- What we should look for ?
- Texture consists of organized patterns of quite regular subelements. “Textons”
- Find the subelements, and represent their statistics
- Reason about their spatial layout.
- Problem: There is no known canonical set of textons.

Texture Analysis

Different approaches:

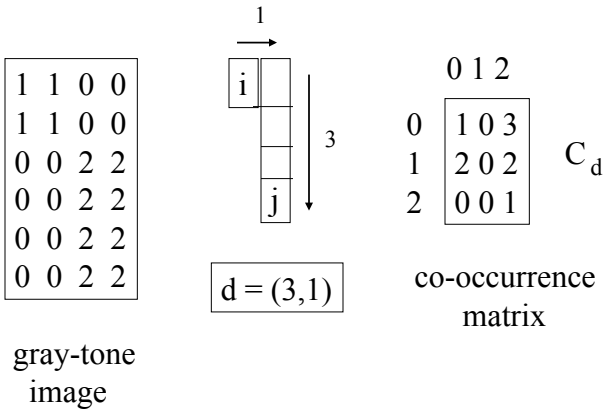
- Co-occurrence matrices (classical)
- Spatial Filtering
- Random Field Models

Co-occurrence Matrix Features

Objective: Capture spatial relations

A co-occurrence matrix is a 2D array C in which

- Both the rows and columns represent a set of possible image values
- $C_d(i,j)$ indicates how many times value i co-occurs with value j in a particular spatial relationship d.
- The spatial relationship is specified by a vector d = (dr,dc).



From C_d we can compute N_d , the normalized co-occurrence matrix, where each value is divided by the sum of all the values.

Co-occurrence Features

From Co-occurrence matrices extract some quantitative features:

$$Energy = \sum_i \sum_j N_d^2(i,j) \quad (7.7)$$

$$Entropy = -\sum_i \sum_j N_d(i,j) log_2 N_d(i,j) \quad (7.8)$$

$$Contrast = \sum_i \sum_j (i-j)^2 N_d(i,j) \quad (7.9)$$

$$Homogeneity = \sum_i \sum_j \frac{N_d(i,j)}{1+|i-j|} \quad (7.10)$$

$$Correlation = \frac{\sum_i \sum_j (i-\mu_i)(j-\mu_j) N_d(i,j)}{\sigma_i \sigma_j} \quad (7.11)$$

where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column

Disadvantages:

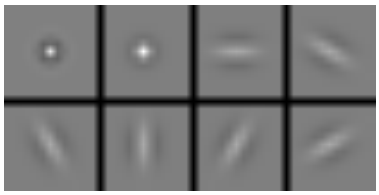
- Computationally expensive
- Sensitive to gray scale distortion (co-occurrence matrices depend on gray values)
- May be useful for fine-grain texture. Not suitable for spatially large textures.

Spatial Filtering Approaches

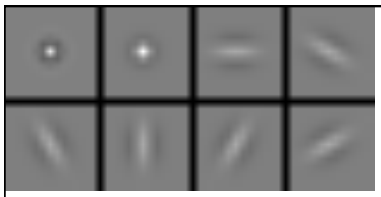
- Look for the subelements
- But what are the subelements, and how do we find them?
- Find subelements by applying filters, looking at the magnitude of the response
- Spots and bars detectors at various scales and orientations.

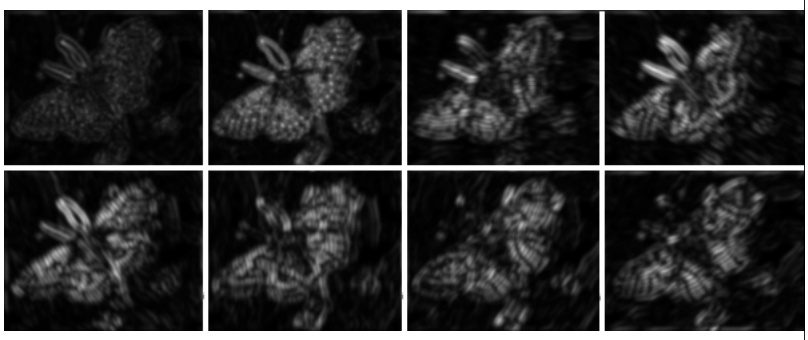
Typically:

- "Spot" filters are Gaussians or weighted sums of concentric Gaussians.
- "Bar" filters are differentiating oriented Gaussians

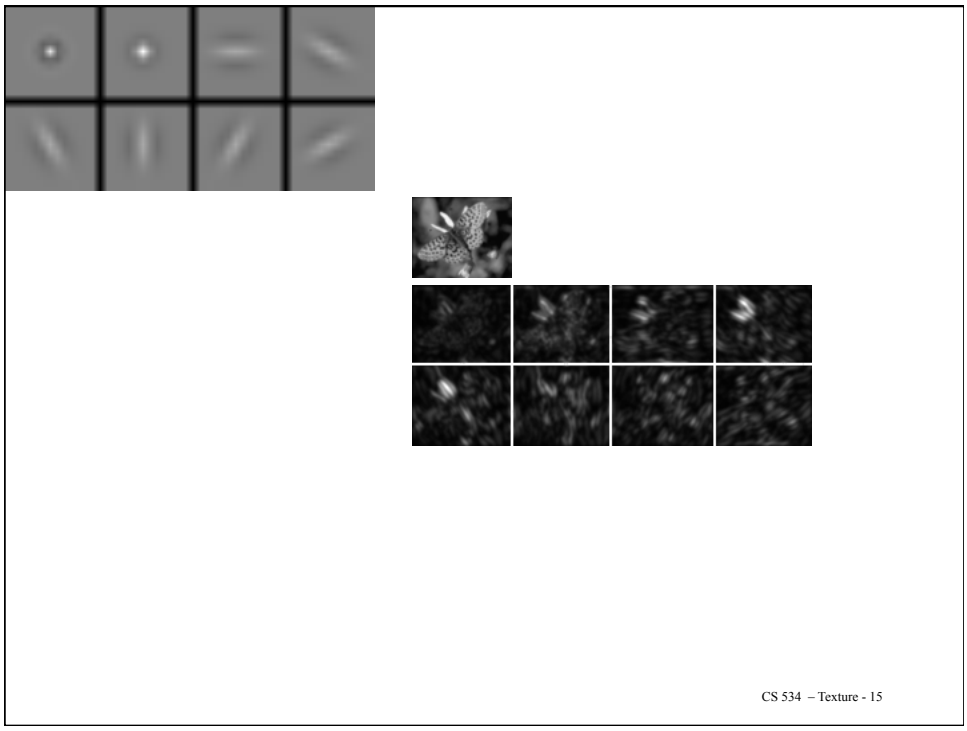


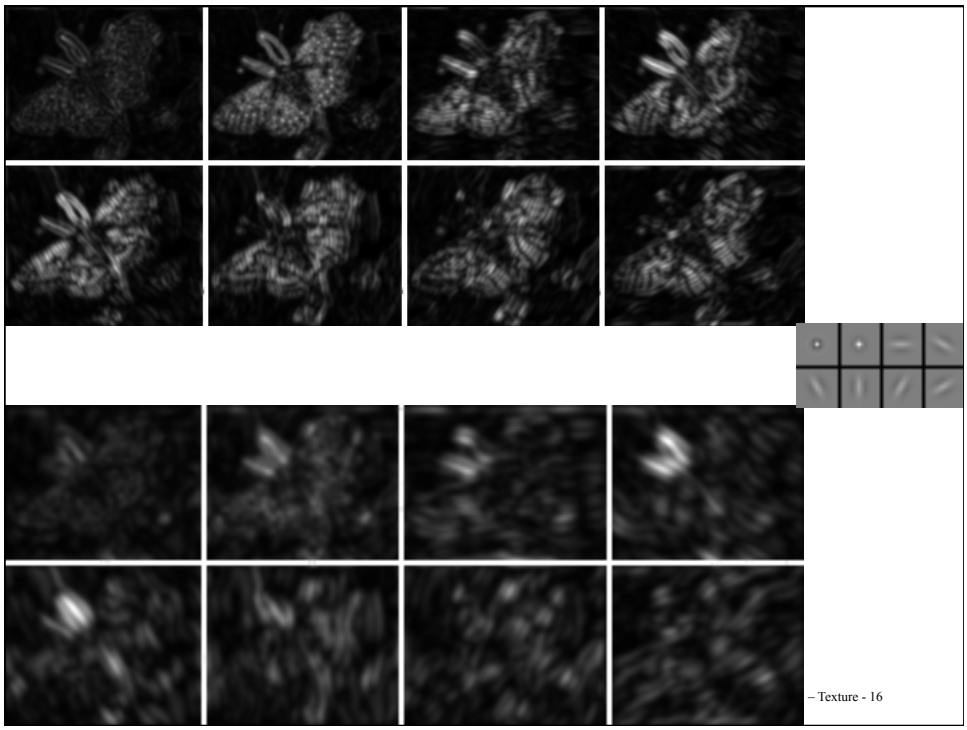
CS 534 – Texture - 13





CS 534 – Texture - 14


CS 534 – Texture - 15


– Texture - 16

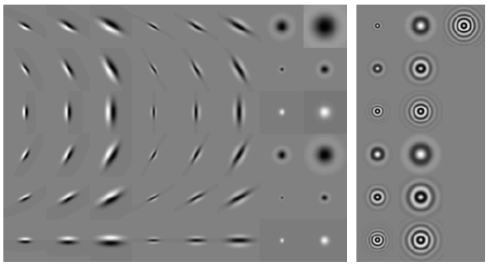
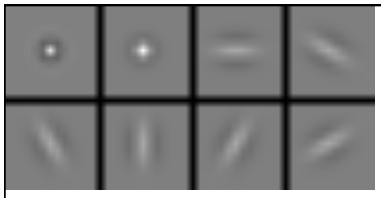
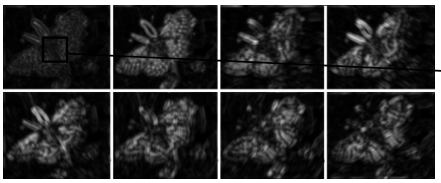


FIGURE 6.4: **Left** shows a set of 48 oriented filters used for expanding images into a series of responses for texture representation. Each filter is shown on its own scale, with zero represented by a mid-gray level, lighter values being positive, and darker values being negative. The left three columns represent edges at three scales and six orientations; the center three columns represent stripes; and the right two represent two classes of spots (with and without contrast at the boundary) at different scales. This is the set of filters used by Leung and Malik (2001). **Right** shows a set of orientation-independent filters, used by Schmid (2001), using the same representation (there are only 13 filters in this set, so there are five empty slots in the image). The orientation-independence property means that these filters look like complicated spots.



Filter bank

Image



Filter responses

Collect Statistics (mean, var, etc.)

Texture Representation

CS 534 – Texture - 18

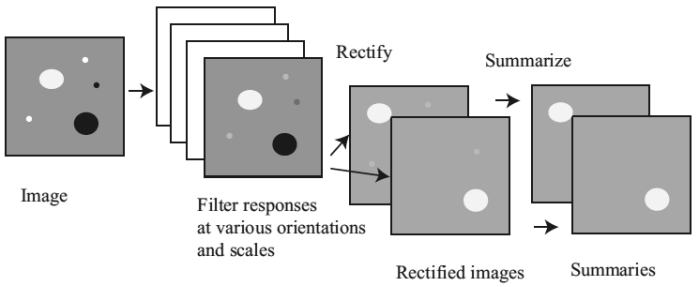


FIGURE 6.3: Local texture representations can be obtained by filtering an image with a set of filters at various scales, and then preparing a summary. Summaries ensure that, at a pixel, we have a representation of what texture appears near that pixel. The filters are typically spots and bars (see Figure 6.4). Filter outputs can be enhanced by rectifying them (so that positive and negative responses do not cancel), then computing a local summary of the rectified filter outputs. Rectifying by taking the absolute value means that we do not distinguish between light spots on a dark background and dark spots on a light background; the alternative, half-wave rectification (described in the text), preserves this distinction at the cost of a fuller representation. One can summarize either by smoothing (which will tend to suppress noise, as in the schematic example above) or by taking the maximum over a neighborhood. Compare this figure to Figure 6.7, which shows a representation for a real image.

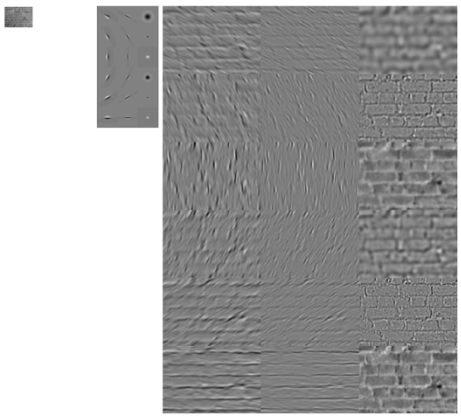


FIGURE 6.5: Filter responses for the oriented filters of Figure 6.4, applied to an image of a wall. At the **center**, we show the filters for reference (but not to scale, because they would be too small to resolve). The responses are laid out in the same way that the filters are (i.e., the response map on the top left corresponds to the filter on the top left, and so on). For reference, we show the image at the **left**. The image of the wall is small, so that the filters respond to structures that are relatively large; compare with Figure 6.6, which shows responses to a larger image of the wall, where the filters respond to smaller structures. These are filters of a fixed size, applied to a small version of the image, and so are equivalent to large-scale filters applied to the original version. Notice the strong response to the vertical and horizontal lines of mortar between the bricks, which are at about the scale of the bar filters. All response values are shown on the same intensity scale: lighter is positive, darker is negative, and mid-gray is zero.

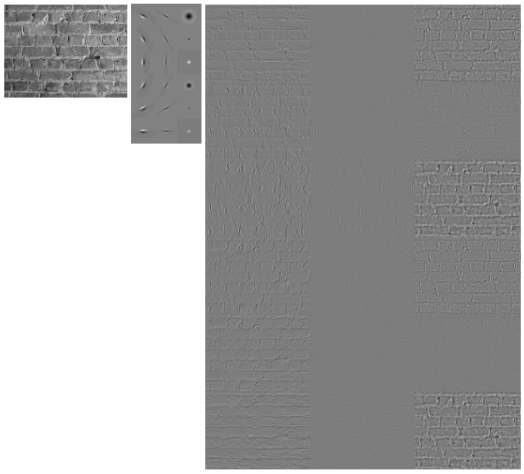


FIGURE 6.6: Filter responses for the oriented filters of Figure 6.4, applied to an image of a wall. At the **center**, we show the filters for reference (not to scale). The responses are laid out in the same way that the filters are (i.e., the response map on the top left corresponds to the filter on the top left, and so on). For reference, we show the image at the **left**. Although there is some response to the vertical and horizontal lines of mortar between the bricks, it is not as strong as the coarse scale (Figure 6.5); there are also quite strong responses to texture on individual bricks. All response values are shown on the same intensity scale: lighter is positive, darker is negative, and mid-gray is zero.

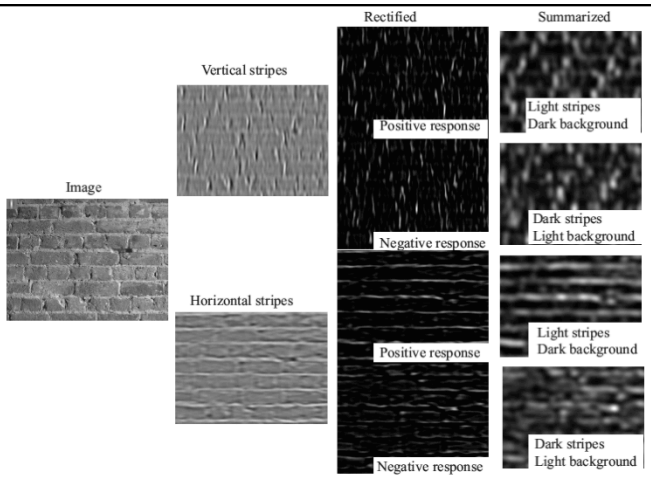
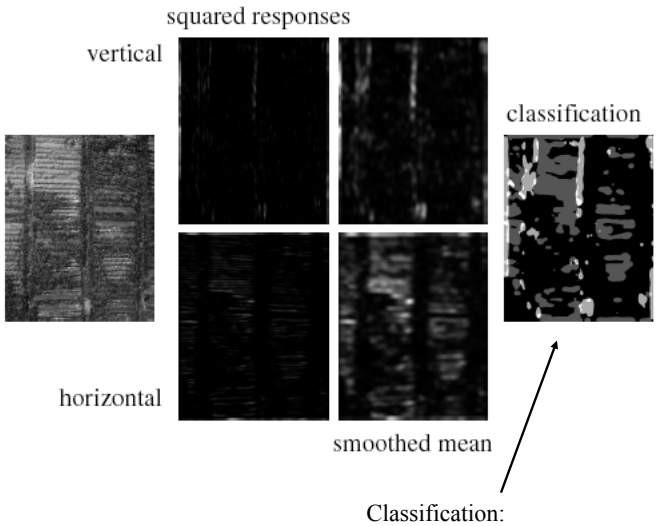


FIGURE 6.7: Filter-based texture representations look for pattern subelements such as oriented bars. The brick image on the **left** is filtered with an oriented bar filter (shown as a tiny inset on the top left of the image at full scale) to detect bars, yielding stripe responses (**center left**; negative is dark, positive is light, mid-gray is zero). These are rectified (here we use half-wave rectification) to yield response maps (**center right**; dark is zero, light is positive). In turn, these are summarized (here we smoothed over a neighborhood twice the filter width) to yield the texture representation on the **right**. In this, pixels that have strong vertical bars nearby are light, and others are dark; there is not much difference between the dark and light vertical structure for this image, but there is a real difference between dark and light horizontal structure.

- How many filters and at what orientations ?
- Filter responses are not unique
- Tradeoff: using more filters leads to a more detailed and more redundant representation of the image
- How to control the amount of redundant information?
- At what scale?
- There are two scales:
  - The scale of the filter
  - The scale over which we consider the distributions of the filters.
- What statistics should be collected from filters responses.



Classification:

• Texture is classified into four categories: vertical, horizontal, both or neither.

Scaled representations: Multiresolution

Use a multiresolution representation (Image Pyramid)

- Search over scale
- Spatial Search
- Feature Tracking

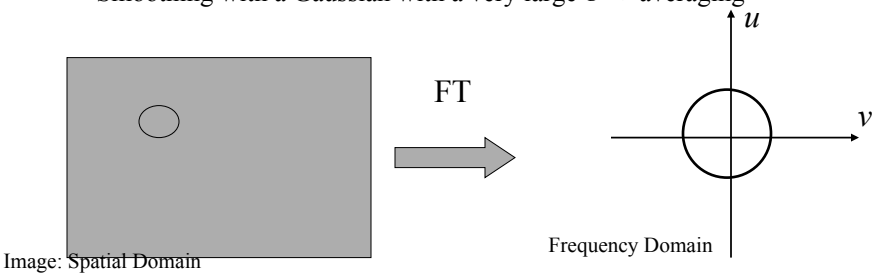
Examples:

- Search for correspondence
  - look at coarse scales, then refine with finer scales
- Edge tracking
  - a "good" edge at a fine scale has parents at a coarser scale
- Control of detail and computational cost in matching
  - e.g. finding stripes
  - terribly important in texture representation

Gaussian Filter and Smoothing

Gaussian Filter is Low-Pass Filter:

- Recall: Convolution in the image domain is equivalent to multiplication in the Frequency domain.
- Recall: FT of a Gaussian with sd=σ is a Gaussian with sd=$1/\sigma$
- Therefore, convolving an image with a Gaussian with sd=σ is equivalent to multiplying it's FT with a Gaussian with sd=$1/\sigma$
- Therefore we will get rid of high frequencies.
- Smoothing with a Gaussian with a very small $\sigma \Rightarrow$ get rid of highest spatial frequencies
- Smoothing with a Gaussian with a very large $\sigma \Rightarrow$ averaging

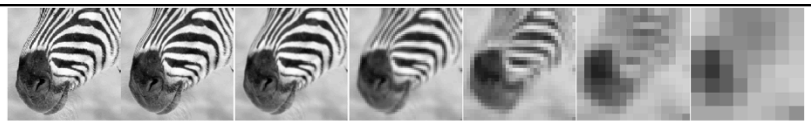


The Gaussian pyramid

- Smooth with gaussians, because
  - a gaussian*gaussian=another gaussian
- Forming a Gaussian Pyramid:
  - Set the finest scale layer to the image
  - For each layer going up (coarser)
    - Obtain this layer by smoothing the previous layer with a Gaussian and subsampling it

$$P_{\text{Gaussian}}(I)_{n+1} = S^{\downarrow}(G_{\sigma} * P_{\text{Gaussian}}(I)_n)$$
$$P_{\text{Gaussian}}(I)_1 = I$$

CS 534 – Texture - 28



512 256 128 64 32 16 8

9

The Laplacian Pyramid

- Gaussians are low pass filters, so response is redundant
- A coarse level layer of the Gaussian pyramid predicts the appearance of the next finer layer
- Laplacian Pyramid
  - preserve differences between upsampled Gaussian pyramid level and Gaussian pyramid level
  - band pass filter - each level represents spatial frequencies (largely) unrepresented at other levels

$$P_{\text{Laplacian}}(I)_m = P_{\text{Gaussian}}(I)_m$$

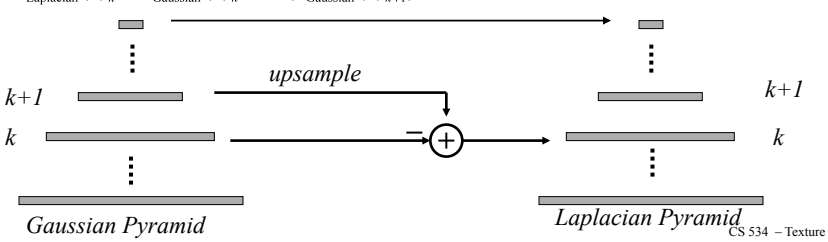
$$P_{\text{Laplacian}}(I)_k = P_{\text{Gaussian}}(I)_k - S^{\uparrow}(P_{\text{Gaussian}}(I)_{k+1})$$

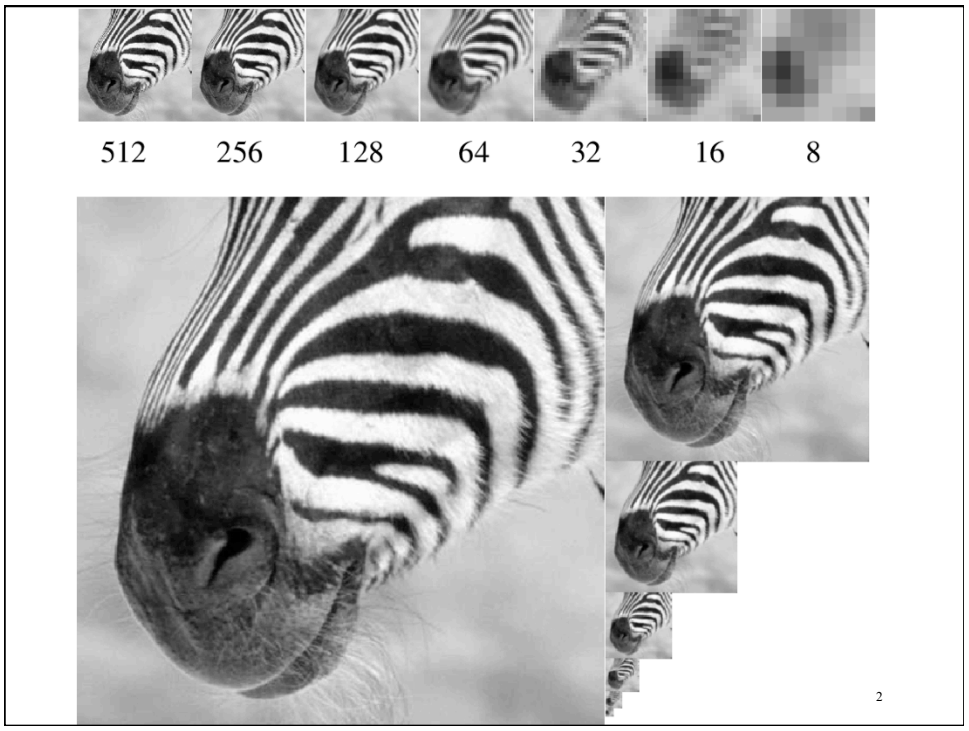
Laplacian Pyramid

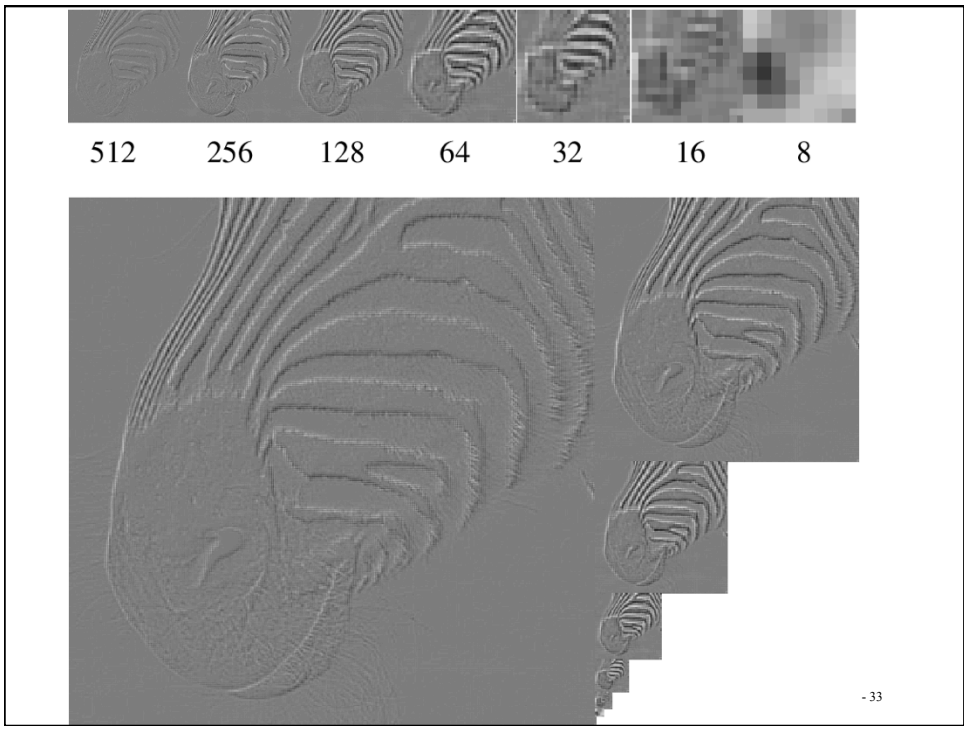
- Building a Laplacian Pyramid:
  - Form a Gaussian pyramid
  - Set the coarsest layer of the Laplacian pyramid to be the coarsest level of the Laplacian pyramid
  - For each layer going from next to coarsest to finest (top to bottom):
    - Obtain this layer by upsampling the coarser layer and subtracting it from this layer of the Gaussian pyramid.

$P_{\text{Laplacian}}(I)_m = P_{\text{Gaussian}}(I)_m$

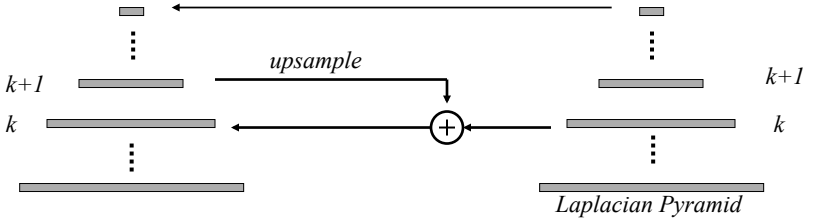
$P_{\text{Laplacian}}(I)_k = P_{\text{Gaussian}}(I)_k - S^{\uparrow}(P_{\text{Gaussian}}(I)_{k+1})$




512
256
128
64
32
16
8
2

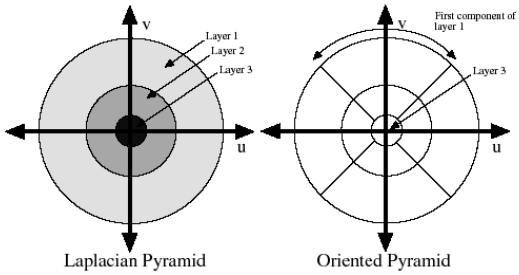

512
256
128
64
32
16
8
- 33

- Synthesis: Obtaining an Image from a Laplacian Pyramid:
  - Start at the coarsest layer
  - For each layer from next to coarsest to finest
    - Upsample the current image and add the current layer to the result



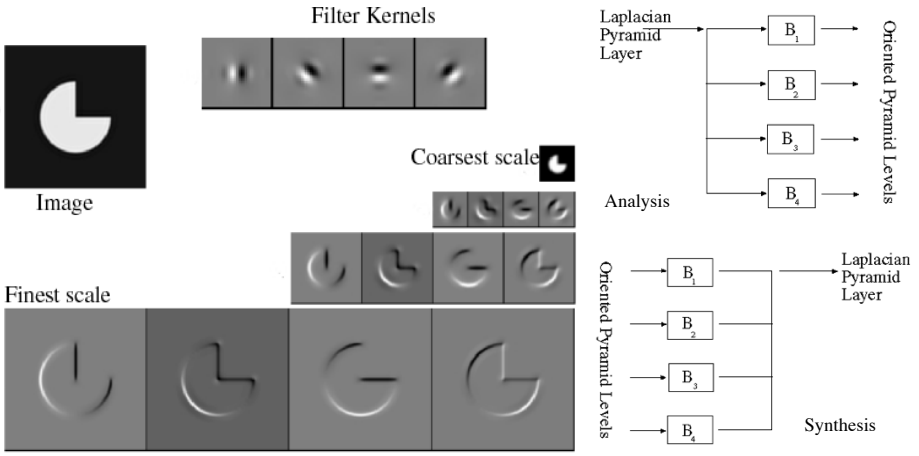
CS 534 – Texture - 34

- Laplacian pyramid layers are band-pass filters.
- Laplacian pyramid is orientation independent
- Apply an oriented filter to determine orientations at each layer
- Look into spatial frequency domain:



CS 534 – Texture - 35

Oriented Pyramids

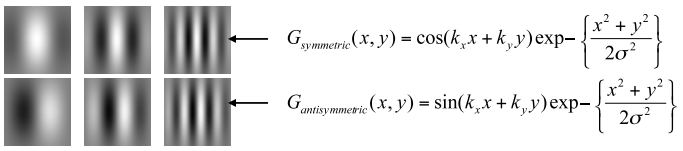


Reprinted from "Shiftable MultiScale Transforms," by Simoncelli et al., IEEE Transactions on Information Theory, 1992, copyright 1992, IEEE

CS 534 – Texture - 36

Gabor Filters

- Fourier coefficients depend on the entire image (Global): We lose spatial information.
- Objective: Local Spatial Frequency Analysis
- Gabor kernels: look like Fourier basis multiplied by a Gaussian
  - The product of a symmetric Gaussian with an oriented sinusoid
  - Gabor filters come in pairs: symmetric and antisymmetric
  - Each pair recover symmetric and antisymmetric components in a particular direction.
  - (k_x,k_y) :the spatial frequency to which the filter responds strongly
  - σ: the scale of the filter. When σ = infinity, similar to FT
- We need to apply a number of Gabor filters at different scales, orientations, and spatial frequencies.



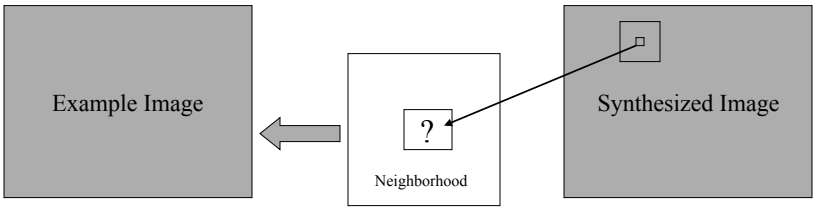
$$G_{symmetric}(x,y) = \cos(k_x x + k_y y)\exp-\left\{\frac{x^2+y^2}{2\sigma^2}\right\}$$

$$G_{antisymmetric}(x,y) = \sin(k_x x + k_y y)\exp-\left\{\frac{x^2+y^2}{2\sigma^2}\right\}$$

CS 534 – Texture - 37

Texture synthesis

- Variety of approaches.
- Example: Synthesis by Sampling Local Models: Efros and Leung 1999 (Nonparametric texture matching)
  - Use image as a source of probability model
  - Choose pixel values by matching neighborhood, then filling in



Find Matching Image Neighborhood and chose value uniformly randomly from these matches

```
Choose a small square of pixels at random from the example image
Insert this square of values into the image to be synthesized
Until each location in the image to be synthesized has a value
  For each unsynthesized location on
    the boundary of the block of synthesized values
    Match the neighborhood of this location to the
      example image, ignoring unsynthesized
      locations in computing the matching score
    Choose a value for this location uniformly and at random
      from the set of values of the corresponding locations in the
      matching neighborhoods
  end
end
```

Algorithm 6.4: Non-parametric Texture Synthesis.

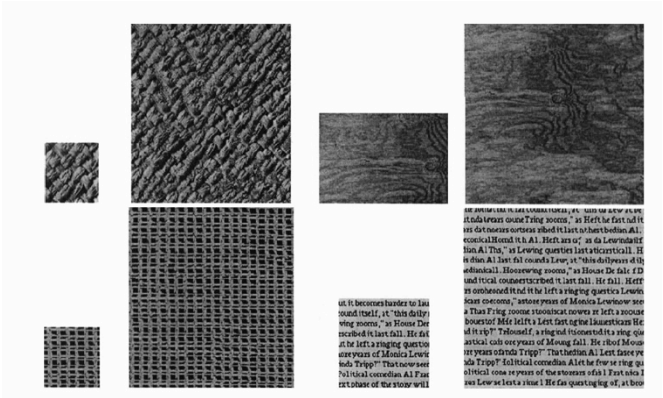


Figure from Texture Synthesis by Non-parametric Sampling, A. Efros and T.K. Leung, Proc. Int. Conf. Computer Vision, 1999 copyright 1999, IEEE

CS 534 – Texture - 43

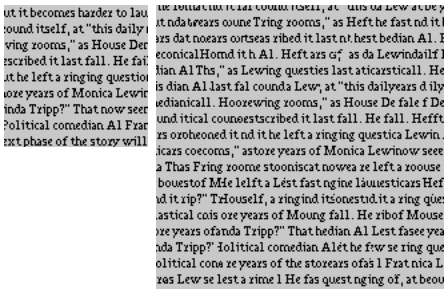
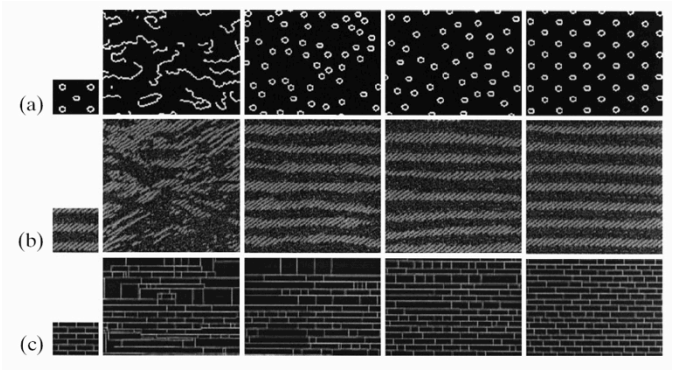


Figure from Texture Synthesis by Non-parametric Sampling, A. Efros and T.K. Leung, Proc. Int. Conf. Computer Vision, 1999 copyright 1999, IEEE

- The size of the image neighborhood to be matched makes a significant difference



Fill in holes by looking for example patches in the image. If needed, rectify faces (lower images)

FIGURE 6.13: If an image contains repeated structure, we have a good chance of finding examples to fill a hole by searching for patches that are compatible with its boundaries. **Top left:** An image with a hole in it (black pixels in a rough pedestrian shape). The pixels on the region outside the hole, but inside the boundary marked on the image, match pixels near the other curve, which represents a potentially good source of hole-filling pixels. **Top right:** The hole filled by placing the patch over the hole, then using a segmentation method (Chapter 9) to choose the right boundary between patch and image. This procedure can work for apparently unpromising images, such as the one on the **bottom left**, an image of the facade of a house, seen at a significant slant. This slant means that distant parts of the facade are severely foreshortened. However, if we rectify the facade using methods from Section 1.3, then there are matching patches. On the **bottom right**, the hole has been filled in using a patch from the rectified image, that is then slanted again. *This figure was originally published as Figures 3 and 6 of "Hole Filling through Photomontage," by M. Wilczkowiak, G. Brostow, B. Tordoff, and R. Cipolla, Proc. BMVC, 2005 and is reproduced by kind permission of the authors.*

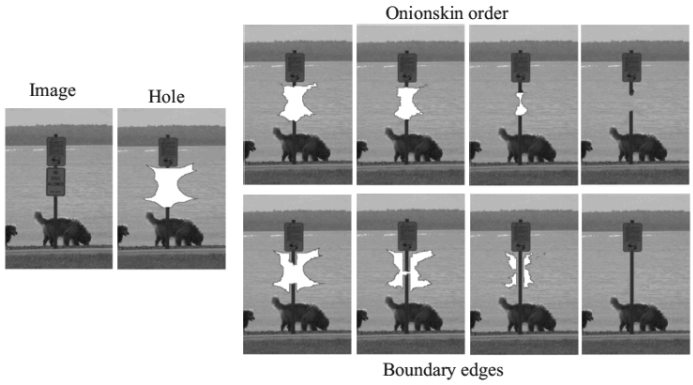
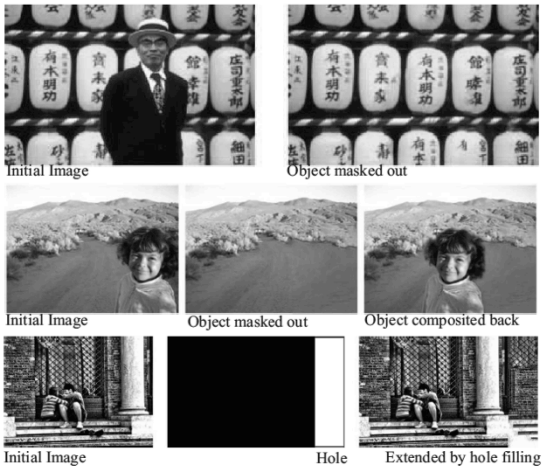


FIGURE 6.14: Texture synthesis methods can fill in holes accurately, but the order in which pixels are synthesized is important. In this figure, we wish to remove the sign, while preserving the signpost. Generally, we want to fill in pixels where most of the neighbors are known first. This yields better matching patches. One way to do so is to fill in from the boundary. However, if we simply work our way inwards (onionskin filling), long scale image structures tend to disappear. It is better to fill in patches close to edges first. *This figure was originally published as Figure 11 of "Region Filling and Object Removal by Exemplar-Based Image Inpainting," by A. Criminisi, P. Perez, and K. Toyama, IEEE Transactions on Image Processing, 2004 © IEEE, 2004.*



State of the art in image fill-in is very good. This uses texture synthesis and other methods.

FIGURE 6.15: Modern hole-filling methods get very good results using a combination of texture synthesis, coherence, and smoothing. Notice the complex, long-scale structure in the background texture for the example on the **top** row. The **center** row shows an example where a subject was removed from the image and replaced in a different place. Finally, the **bottom** row shows the use of hole-filling to resize an image. The white block in the center mask image is the "hole" (i.e., unknown pixels whose values are required to resize the image). This block is filled with a plausible texture. *This figure was originally published as Figures 9 and 15 of "A Comprehensive Framework for Image Inpainting," by A. Bugeau, M. Bertalmío, V. Caselles, and G. Sapiro, Proc. IEEE Transactions on Image Processing, 2010 © IEEE, 2010.*

Sources

- Forsyth and Ponce, Computer Vision a Modern approach (2nd ed): chapter 6.
- L. G. Shapiro and G. C. Stockman "Computer Vision", Prentice Hall 2001.
- R. Gonzalez and R.E. Woods, "Digital Image Processing", 2002.
- Slides by
  - D. Forsyth @ UC Berkeley
  - G.C. Stockman @MSU