COSC579: Texture

Jeremy Bolton, PhD
Assistant Teaching Professor

GEORGETOWN UNIVERSITY

Outline

I. What is texture?
II. Approaches to specifying or modelling texture
    I. (Matched) Filters
    II. Tuple of features (use HoG, Hough, Edge, …)
III. Texture in frequency domain

- Structure:
  - texture representations
  - texture synthesis
  - shape from texture

Texture

- Patterns of structure from
  - changes in surface albedo (eg printed cloth)
  - changes in surface shape (eg bark)
  - many small surface patches (eg leaves on a bush)
- Hard to define; but texture tells us
  - what a surface is like
  - (sometimes) object identity
  - (sometimes) surface shape

Texture: Core Problems

- Represent complex surface textures to recognize
  - objects
  - materials
  - textures
- Synthesize texture from examples
  - to create big textures for computer graphics
  - to fill in holes in images caused by editing

Texture representation

- Core idea: Textures consist of
  - a set of elements
  - repeated in some way

- Representations
  - identify the elements
  - summarize the repetition

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.*

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

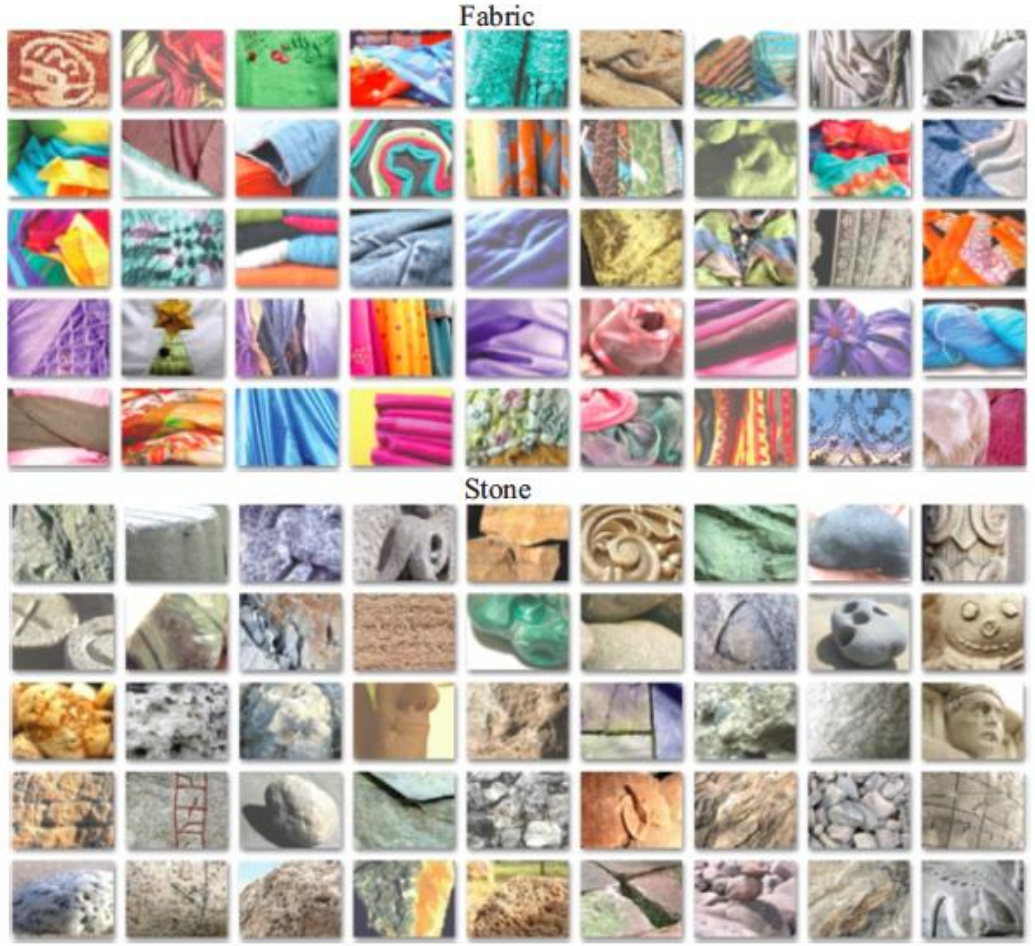


FIGURE 6.2: Typically, different materials display different image textures. These are example images from a collection of 1,000 material images, described in by Sharan *et al.* (2009); there are 100 images in each of the ten categories, including the two categories shown here (fabric and stone). Notice how (a) the textures vary widely, even within a material category; and (b) different materials seem to display quite different textures. *This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at* `http://people.csail.mit.edu/lavanya/research_sharan.html`. *Figure by kind permission of the collectors.*

Different materials tend to have different textures (though these are not the same ideas)

Filter based texture representations

- Choose a set of filters, each representing a pattern element
  - typically a spot and some oriented bars
- Filter the image at a variety of scales
- Rectify the filtered images
  - typically half wave, to avoid averaging contrast reversals
    - eg should not average dark spot on light background, light spot on dark background to zero
- Compute summaries of rectified filtered images
  - eg smoothed average
  - at a variety of scales to capture
    - nearby pattern elements and general picture of pattern elements
- Now describe each pixel by vector of summaries
  - which could be very long

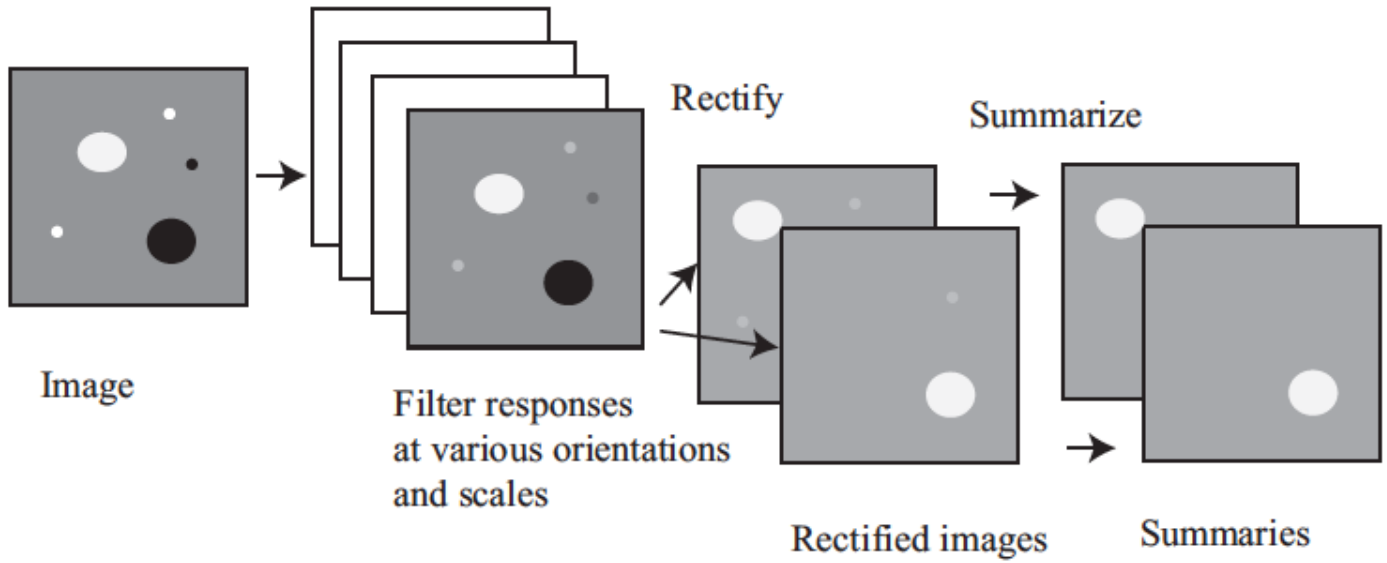


FIGURE 6.3: Local texture representations can be obtained by filtering an image with a set of filters at various scales, and then preparing a summary. Summaries ensure that, at a pixel, we have a representation of what texture appears near that pixel. The filters are typically spots and bars (see Figure 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.

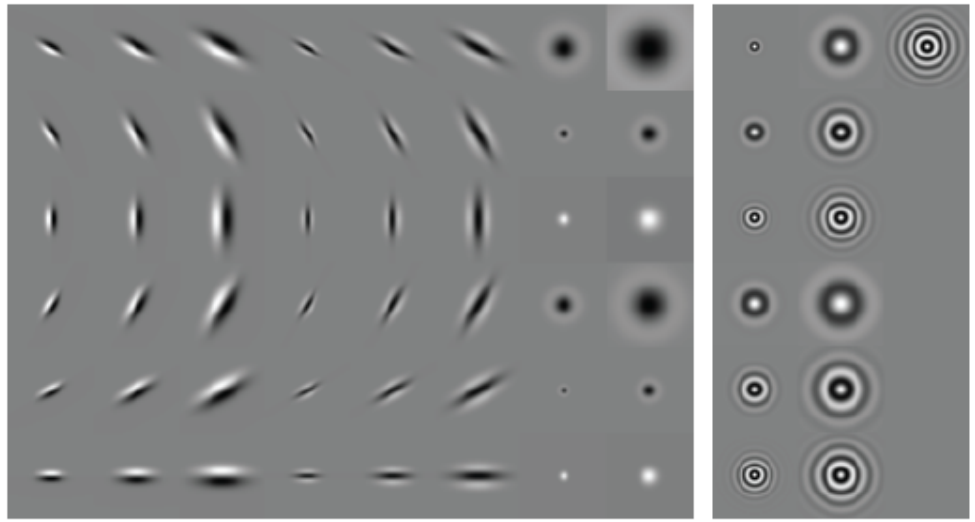


FIGURE 6.4: **Left** shows a set of 48 oriented filters used for expanding images into a series of responses for texture representation. Each filter is shown on its own scale, with zero represented by a mid-gray level, lighter values being positive, and darker values being negative. The left three columns 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.

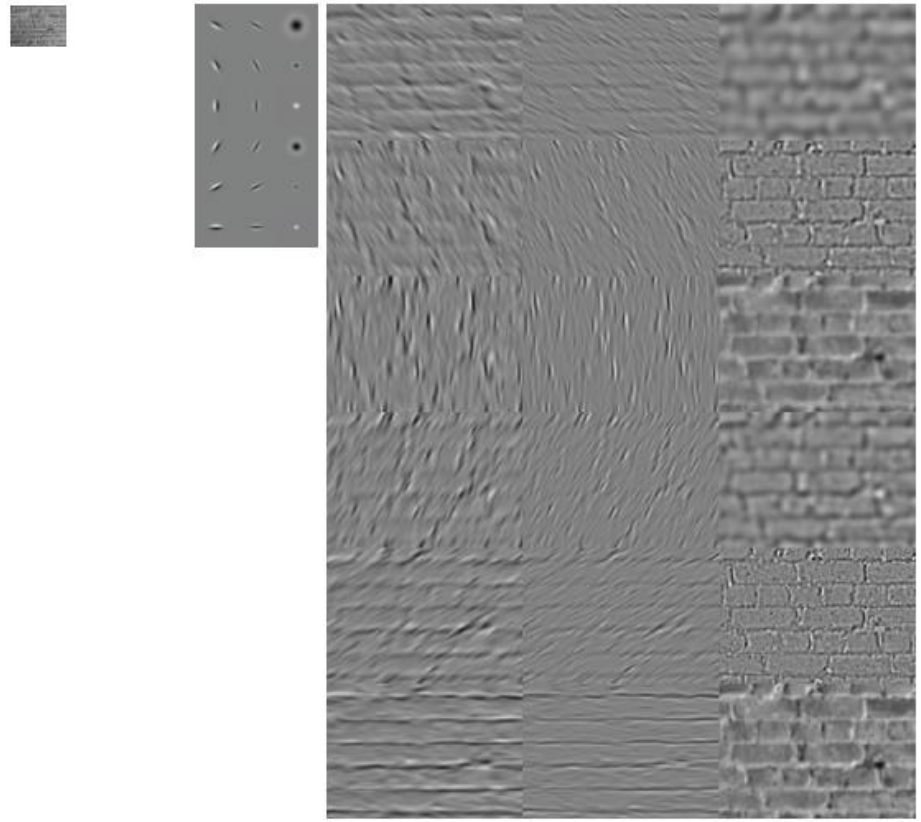


FIGURE 6.5: Filter responses for the oriented filters of Figure 6.4, applied to an image of a wall. At the **center**, we show the filters for reference (but not to scale, because they would be too small to resolve). The responses are laid out in the same way that the filters are (i.e., the response 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.

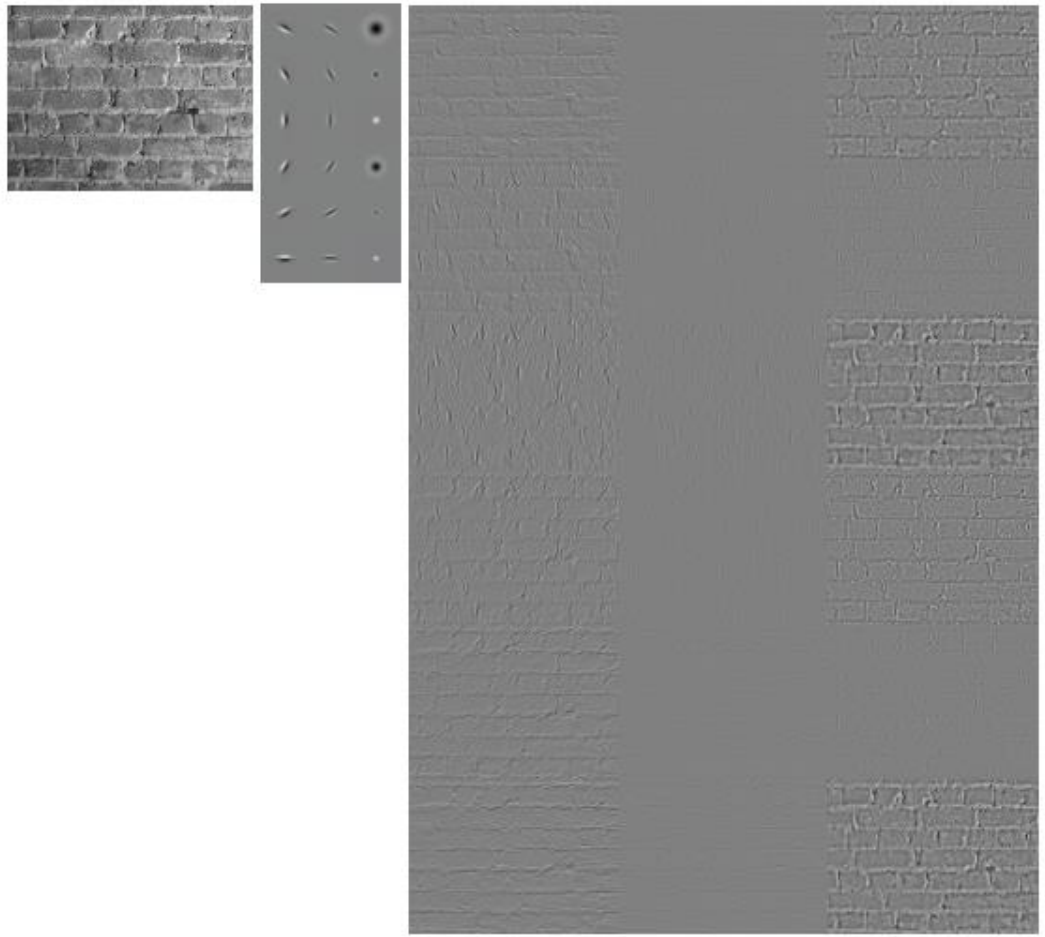


FIGURE 6.6: Filter responses for the oriented filters of Figure 6.4, applied to an image of a wall. At the **center**, we show the filters for reference (not to scale). The responses are laid out in the same way that the filters are (i.e., the response map on the top left corresponds to the filter on 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.

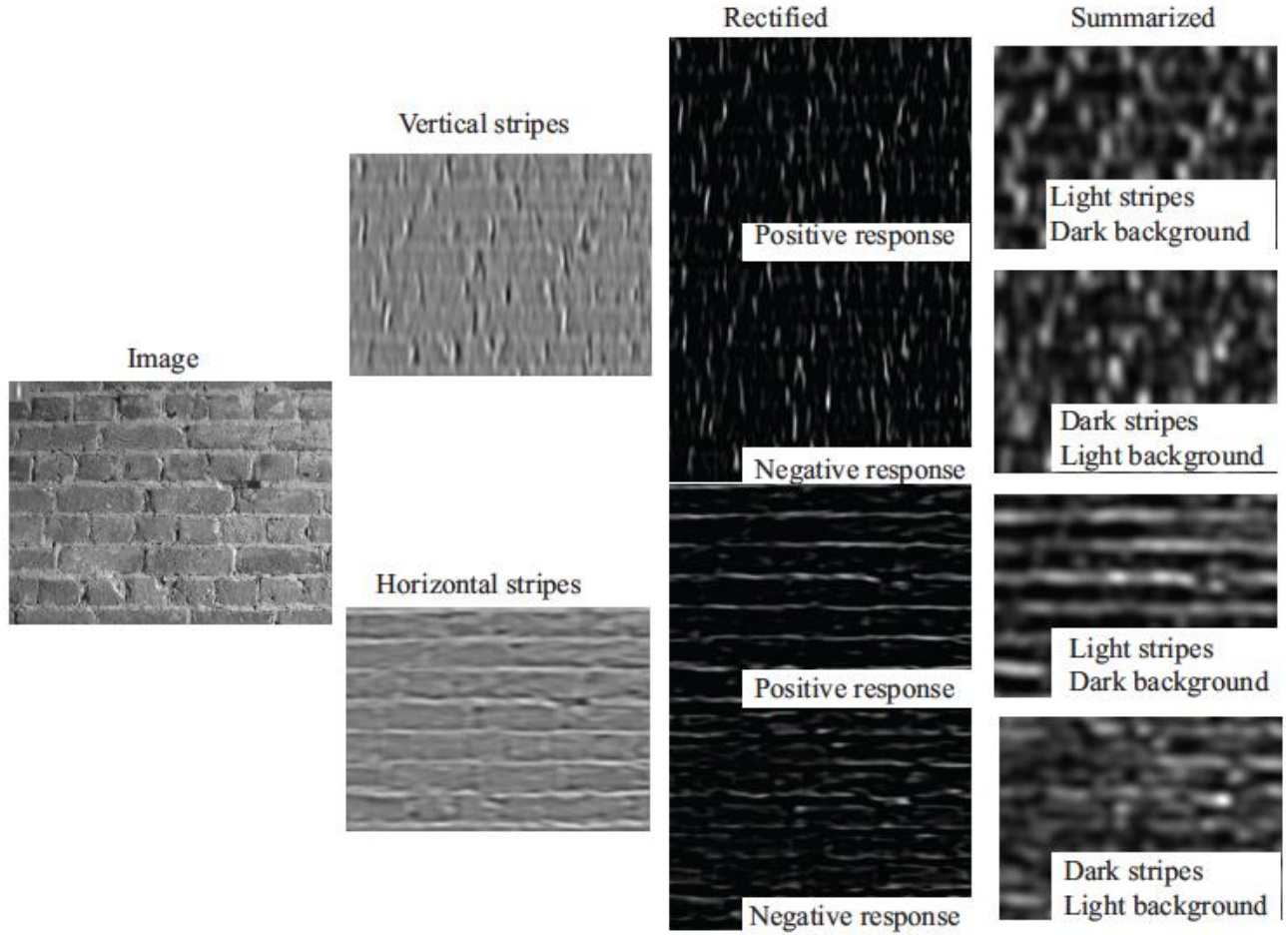


FIGURE 6.7: Filter-based texture representations look for pattern subelements such as oriented bars. The brick image on the **left** is filtered with an oriented bar filter (shown as a tiny inset on the top left of the image at full scale) to detect bars, yielding stripe responses (**center left**; negative 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.

Example based texture representations

- Q: how does one choose the filters?
- Alternative
  - build a vocabulary of pattern elements from pictures
  - describe using this vocabulary

Building a vocabulary

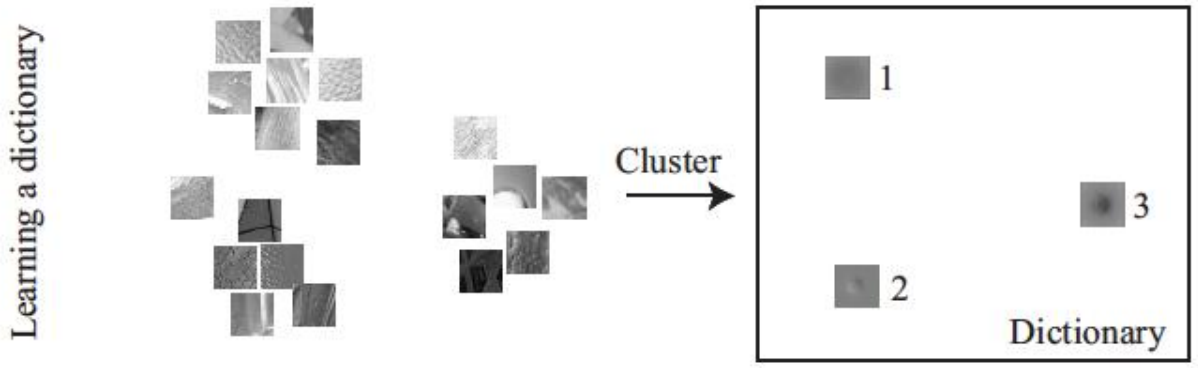


FIGURE 6.8: There are two steps to building a pooled texture representation for a texture in an image domain. First, one builds a dictionary representing the range of possible pattern elements, using a large number of texture patches. This is usually done in advance, using a training data set of some form.

Clustering the examples

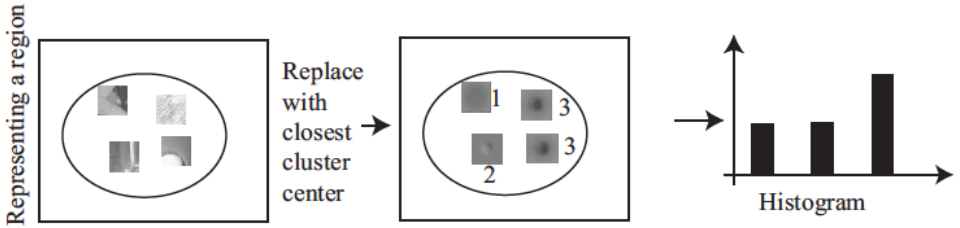
- Natural to use k-means
  - represent patches with
    - intensity vector
    - vector of filter responses over patch

Choose k data points to act as cluster centers
Until the cluster centers change very little
    Allocate each data point to cluster whose center is nearest.
    Now ensure that every cluster has at least
      one data point; one way to do this is by
      supplying empty clusters with a point chosen at random from
      points far from their cluster center.
    Replace the cluster centers with the mean of the elements
      in their clusters.
end

Algorithm 6.3: Clustering by K-Means.

Representing a region

- Vector Quantization
  - Represent a high-dimensional data item with a single number
    - Find the number of the nearest cluster center in dictionary
    - Use that
- Summarize the pattern of patches
  - Cut region into patches
  - Vector quantize - vector quantized image patches often called visual words
  - Build histogram of resulting numbers



GEORGETOWN UNIVERSITY

Build a dictionary:

Collect many training example textures

Construct the vectors x for relevant pixels; these could be a reshaping of a patch around the pixel, a vector of filter outputs computed at the pixel, or the representation of Section 6.1.

Obtain k cluster centers c for these examples

Represent an image domain:

For each relevant pixel i in the image

Compute the vector representation x_i of that pixel

Obtain j, the index of the cluster center c_j closest to that pixel

Insert j into a histogram for that domain

Algorithm 6.2: Texture Representation Using Vector Quantization.

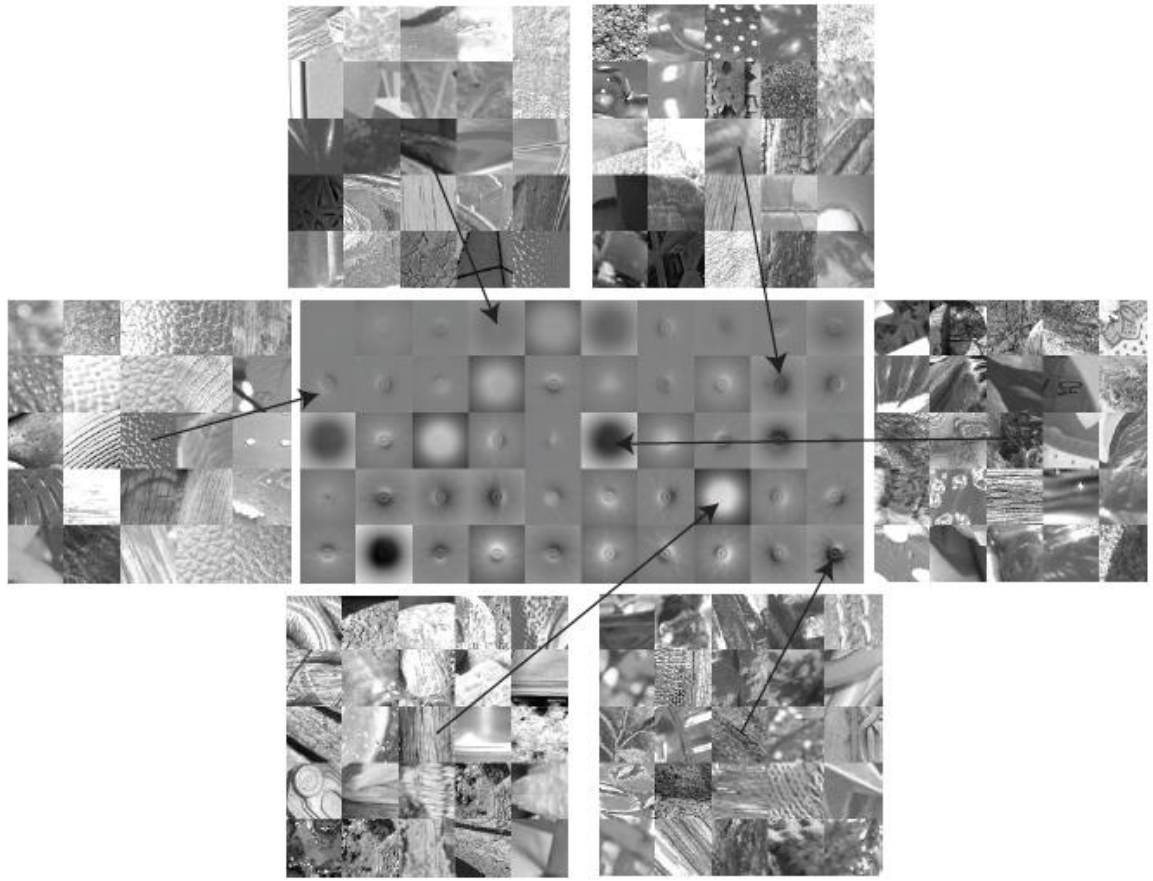


FIGURE 6.9: Pattern elements can be identified by vector quantizing vectors of filter outputs, using k-means. Here we show the top 50 pattern elements (or textons), obtained from all 1,000 images of the collection of material images described in Figure 6.2. These were filtered with the complete set of oriented filters from Figure 6.4. Each subimage here illustrates a cluster center. For each cluster center, we show the linear combination of filter kernels that would result in the set of filter responses represented by the cluster center. For some cluster centers, we show the 25 image patches in the training set whose filter representation is closest to the cluster center. *This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at* `http://people.csail.mit.edu/lavanya/research_sharan.html`. *Figure by kind permission of the collectors.*

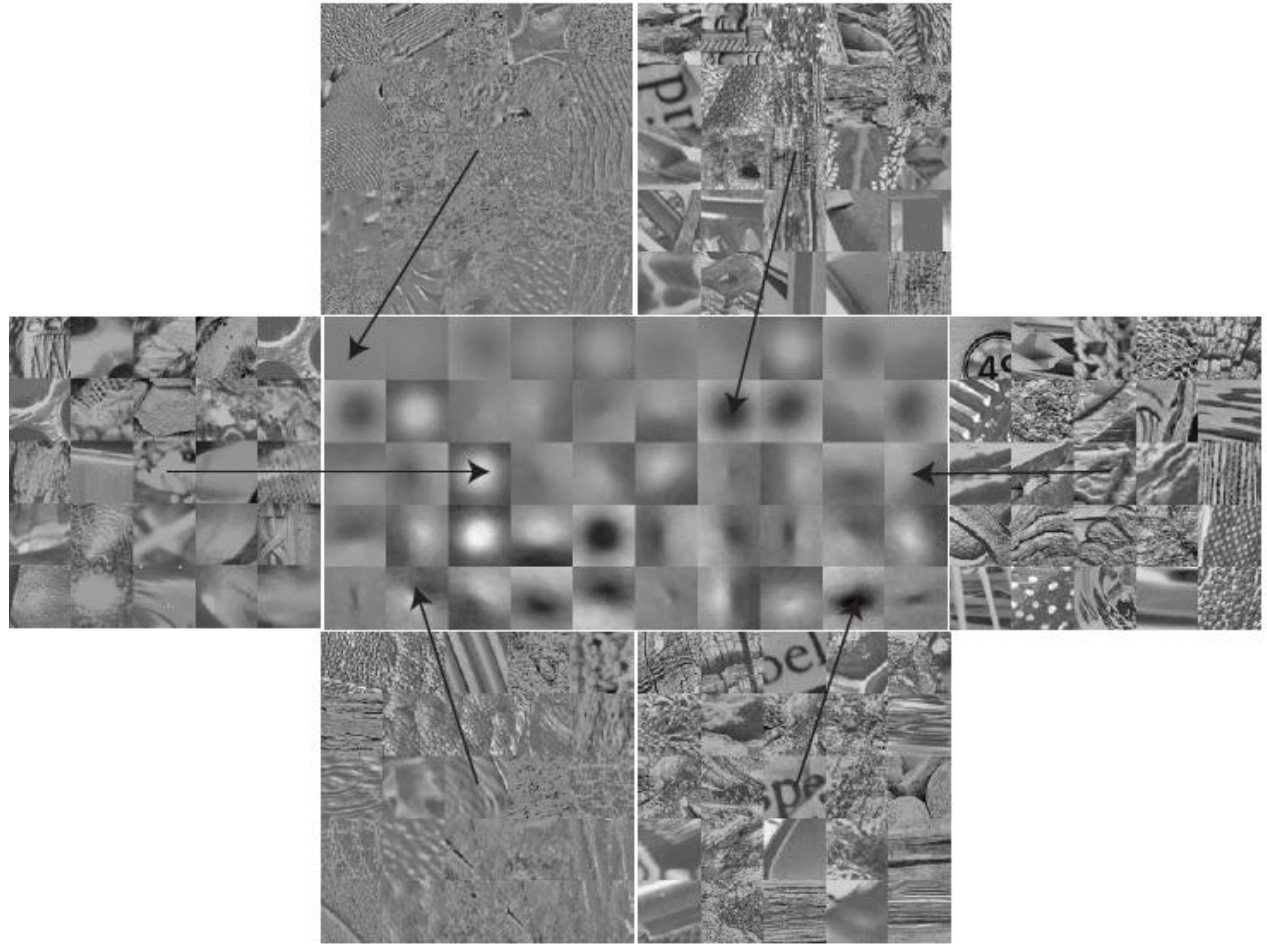


FIGURE 6.10: Pattern elements can also be identified by vector quantizing vectors obtained by reshaping an image window centered on each pixel. Here we show the top 50 pattern elements (or textons), obtained using this strategy from all 1,000 images of the collection of material images described in Figure 6.2. Each subimage here illustrates a cluster center. For some cluster centers, we show the closest 25 image patches. To measure distance, we first subtracted the average image intensity, and we weighted by a Gaussian to ensure that pixels close to the center of the patch were weighted higher than those far from the center. *This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at* `http://people.csail.mit.edu/lavanya/research_sharan.html`. *Figure by kind permission of the collectors.*

Texture representations

- A vector summarizing the trends in pattern elements
  - either overall trend in filter responses
  - or histogram of vector quantized patches

- At a pixel
  - compute representations for domains centered on the pixel

- For a region
  - compute representations the whole region

Texture synthesis

- Problem:
  - Take a small example image of pure texture
  - Use this to produce a large domain of “similar” texture
- Why:
  - Computer graphics demands lots of realistic texture, hard to find
  - Fill in holes in images created by removing objects
- Simple case:
  - Assume we must synthesize a single pixel in a large image
  - Approach:
    - Match the window around that pixel to other windows in the image
    - Choose a value from the matching windows
      - most likely, uniformly and at random

Texture synthesis

- Expand to large images
  - Start: take a piece of the example image
  - Fill in pixels on the boundary
    - But these are missing more than the center
      - Discount missing pixels when matching
  - Each time you fill in a pixel, you can use that to match

```
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.

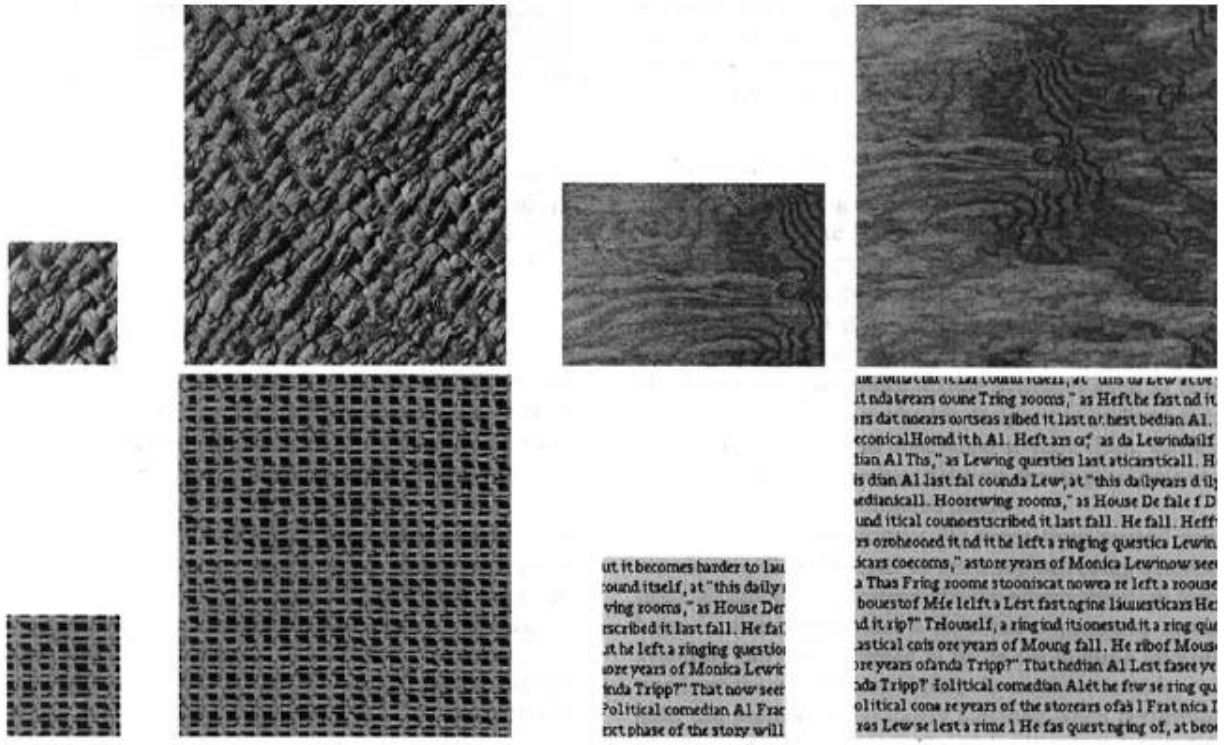


FIGURE 6.11: Efros and Leung (1999) synthesize textures by matching neighborhoods of the image being synthesized to the example image, and then choosing at random amongst the possible values reported by matching neighborhoods (Algorithm 6.4). This means that the algorithm can reproduce complex spatial structures, as these examples indicate. The small block on the **left** is the example texture; the algorithm synthesizes the block on the **right**. Note that the synthesized text looks like text: it appears to be constructed of words of varying lengths that are spaced like text, and each word looks as though it is composed of letters (though this illusion fails as one looks closely). *This figure was originally published as Figure 3 of "Texture Synthesis by Non-parametric Sampling," A. Efros and T.K. Leung, Proc. IEEE ICCV, 1999* © *IEEE, 1999.*

Small blocks are examples, large are synthesized. Notice how (for example) synthesized text looks like actual text.

GEORGETOWN UNIVERSITY

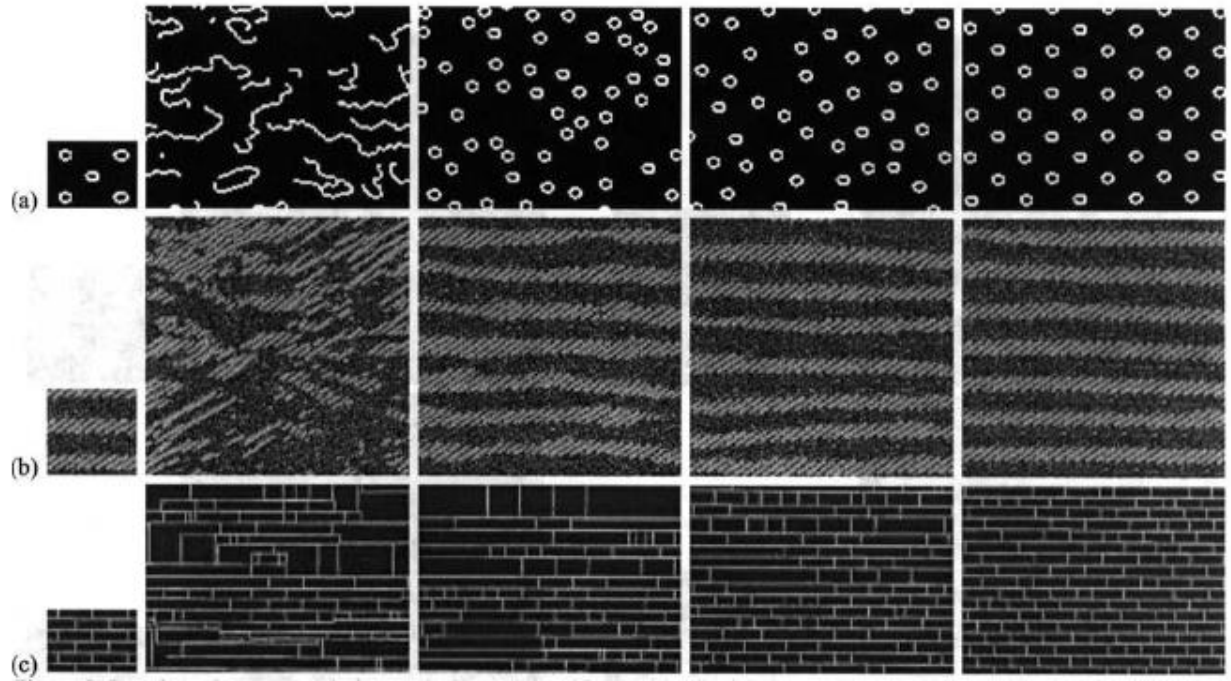


FIGURE 6.12: The size of the image neighborhood to be matched makes a significant difference in Algorithm 6.4. In the figure, the textures at the right are synthesized from the small blocks on the **left**, using neighborhoods that are increasingly large as one moves to the **right**. If very small neighborhoods are matched, then the algorithm cannot capture large-scale effects easily. For example, in the case of the spotty texture, if the neighborhood is too small to capture the spot structure (and so sees only pieces of curve), the algorithm synthesizes a texture consisting of curve segments. As the neighborhood gets larger, the algorithm can capture the spot structure, but not the even spacing. With very large neighborhoods, the spacing is captured as well. *This figure was originally published as Figure 2 of "Texture Synthesis by Non-parametric Sampling," A. Efros and T.K. Leung, Proc. IEEE ICCV, 1999 © IEEE, 1999.*

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.*

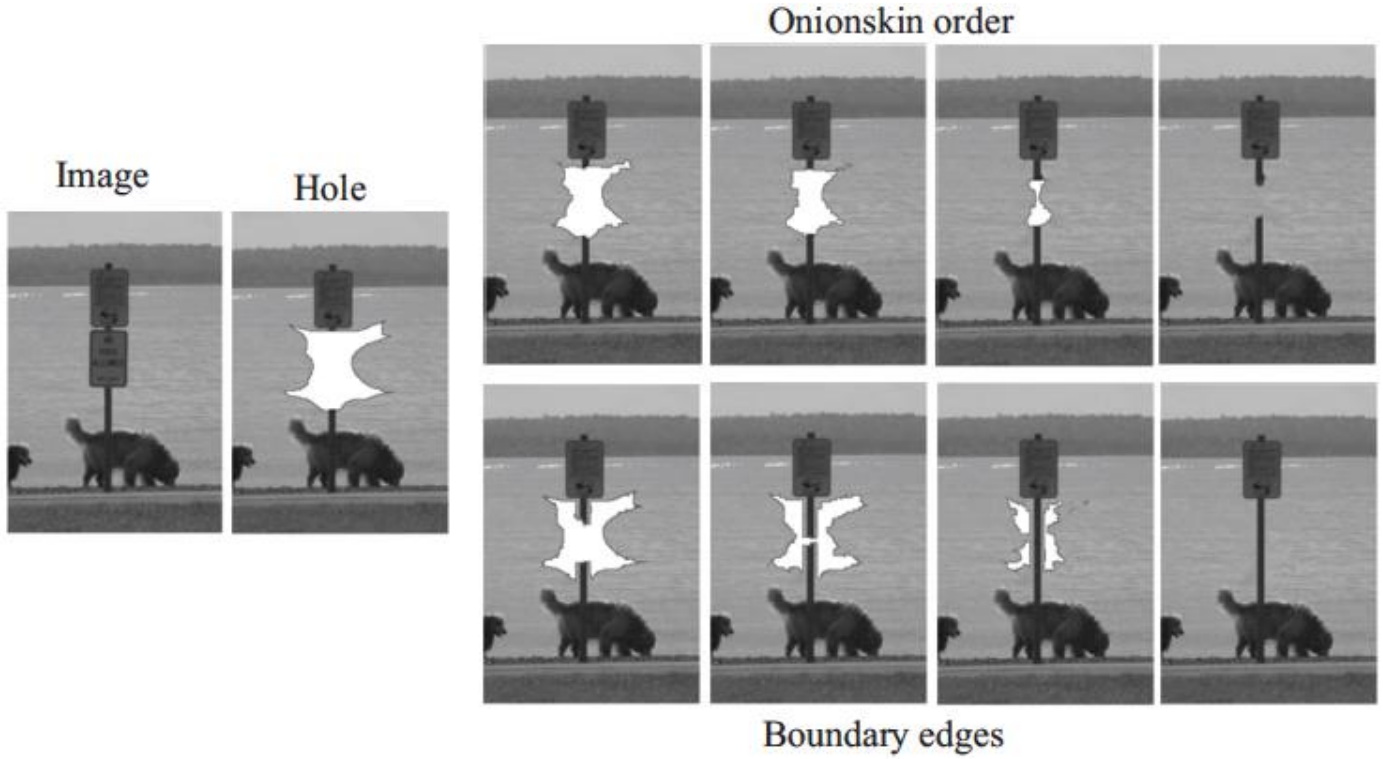
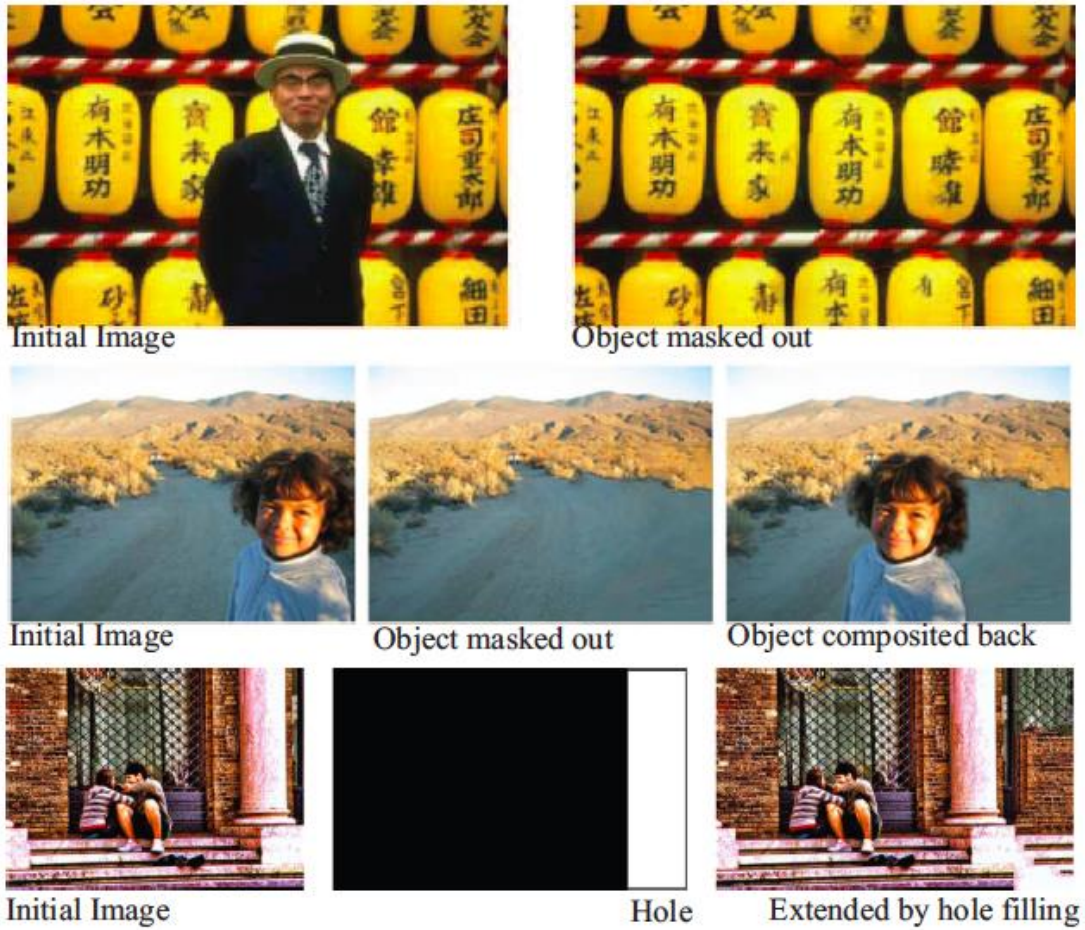


FIGURE 6.14: Texture synthesis methods can fill in holes accurately, but the order in which pixels are synthesized is important. In this figure, we wish to remove the sign, while preserving the signpost. Generally, we want to fill in pixels where most of the neighbors are known first. This yields better 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.*

Shape from texture

- Texture is a powerful shape cue
  - most likely because small pattern elements deform in predictable ways
- Recovering shape from texture
  - Identify repeating pattern elements
  - Determine frontal view
  - From this, determine normal
  - Integrate normals to get surface
- Shape from texture offers information about lighting
  - If pattern elements are repetitions, then
    - brighter (resp. darker) ones receive more (resp. less) light

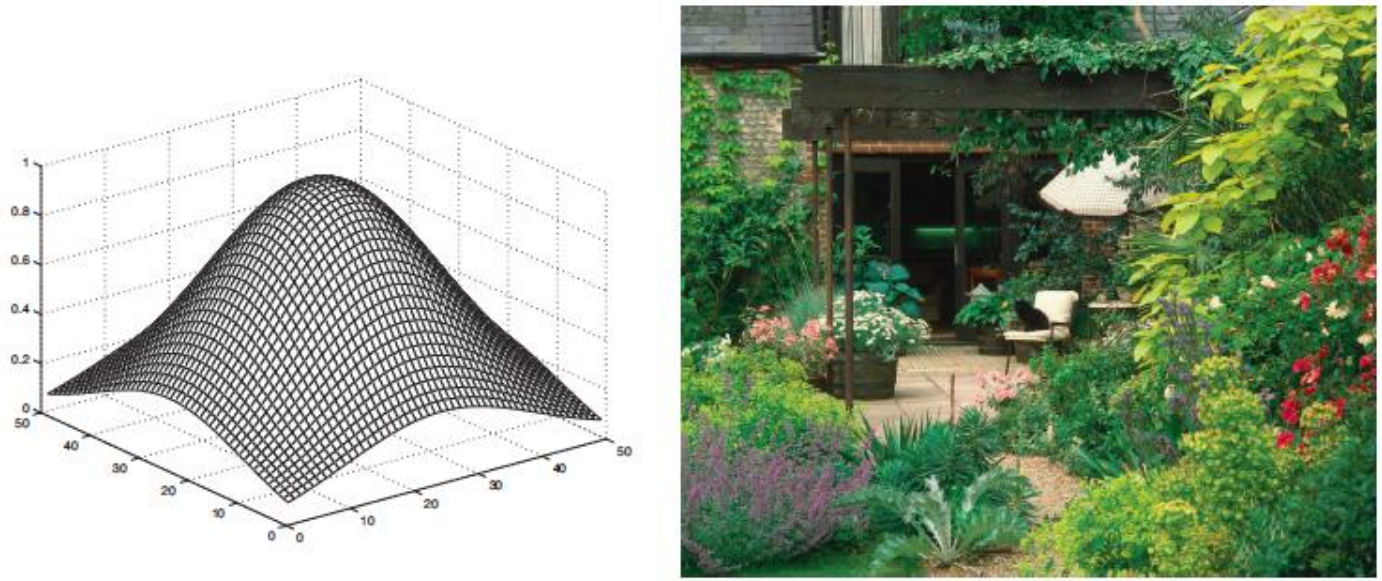
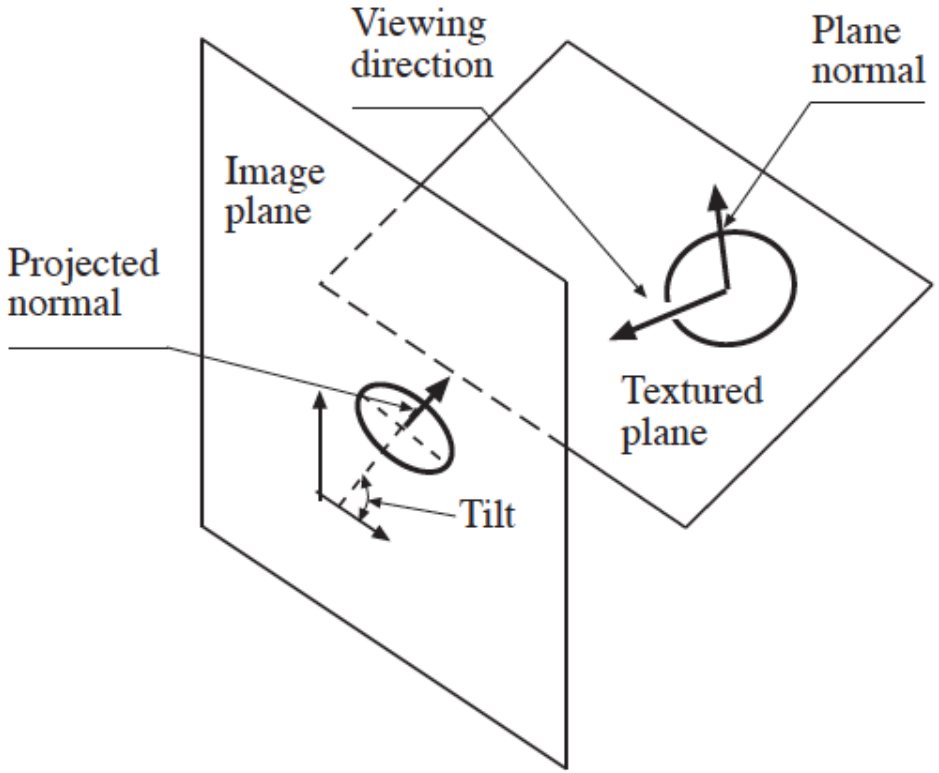
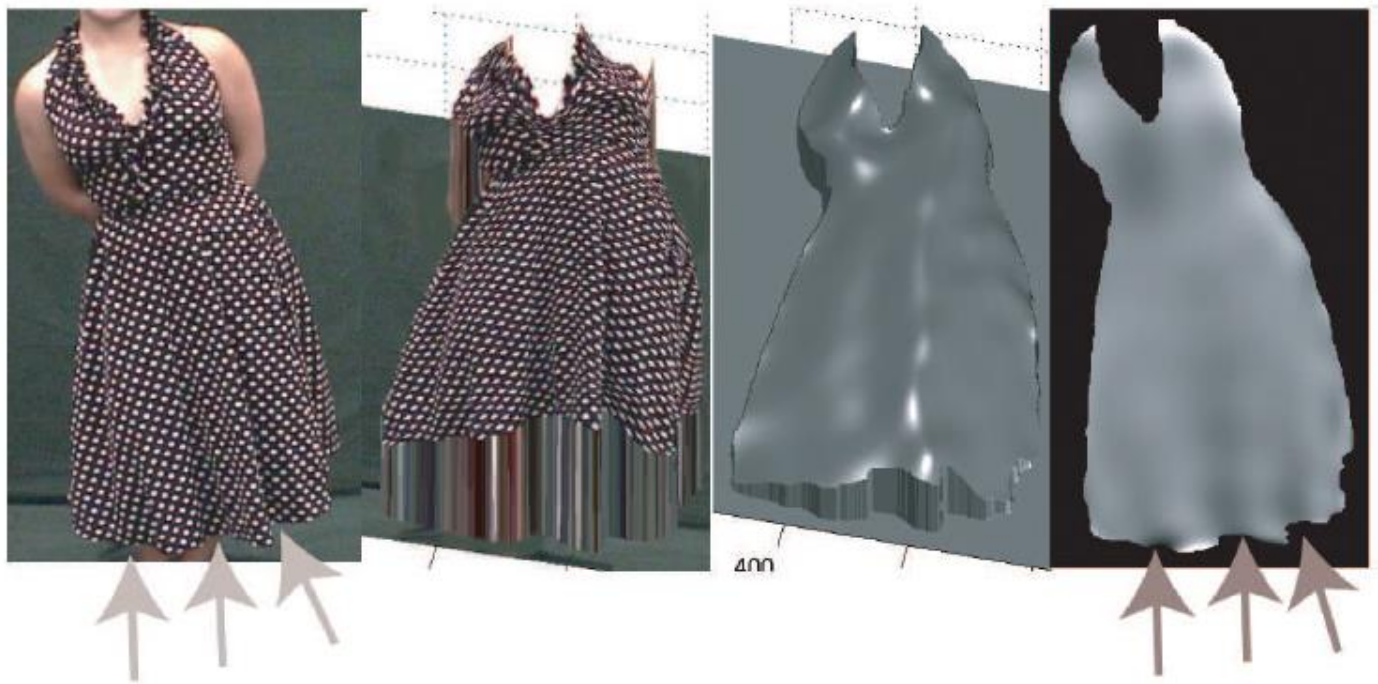


FIGURE 6.19: Humans obtain information about the shape of surfaces in space from the appearance of the texture on the surface. The figure on the **left** shows one common use for this effect; away from the contour regions, our only source of information about the surface depicted is the distortion of the texture on the surface. On the **right**, the texture gives a clear sense of the orientation of the ground plane, how the plants stand out from the path, and how far away the building at the back is. *Geoff Brightling © Dorling Kindersley, used with permission.*



Surface orientation foreshortens texture elements. If you know what the element is, you can figure out the surface orientation, and so the shape.
The trick is recovering the element from repeated examples.

FIGURE 6.20: The orientation of a plane with respect to the camera plane can be given by the slant, which is the angle between the normal of the textured plane and the viewing direction, and the tilt, which is the angle the projected normal makes with the camera coordinate system. The figure illustrates the tilt, and shows a circle projecting to an ellipse. The direction of the minor axis of this image ellipse is the tilt, and the slant is revealed by the aspect ratio of the ellipse. However, the slant is ambiguous because the foreshortening is given by $\cos\sigma$, where σ is the slant angle. There will be two possible values of σ for each foreshortening, so two different slants yield the same ellipse (one is slanted forwards, the other backwards).



If pattern elements are repetitions, then brighter (resp. darker) ones receive more (resp. less) light, so we get an estimate of lighting.

FIGURE 6.21: On the **left**, a textured surface, whose texture is a set of repeated elements, in this case, spots. **Center left**, a reconstruction of the surface, made using texture information alone. This reconstruction has been textured, which hides some of its imperfections. **Center right**, the same reconstruction, now rendered as a slightly glossy gray surface. Because texture elements are repeated, we can assume that if different elements have a significantly different brightness, this is because they experience different illumination. **Right** shows an estimate of the illumination on the surface obtained from this observation. Notice how folds in the dress (arrows) tend to be darker; this is because, for a surface element at the base of a fold, nearby cloth blocks a high percentage of the incident light. *This figure was originally published as Figure 4 of "Recovering Shape and Irradiance Maps from Rich Dense Texton Fields," by A. Lobay and D. Forsyth Proc. IEEE CVPR 2004 © IEEE, 2004.*

Texture: crucial points

- Texture is the result of repetition of pattern elements
- Represent by representing
  - vocabulary of pattern elements
    - either filters or visual words
  - summary of how they repeat
    - local averages of rectified filter responses or histograms
- Good texture synthesis procedures are available
- Texture reveals shape