Texture

- D. Forsythe and J. Ponce
- Computer Vision modern approach
- Chapter 9
- (Slides D. Lowe, UBC)

Texture

- **Key issue**: How do we represent texture?
- **Topics**:
  - Texture segmentation
  - Texture-based matching
  - Texture synthesis
    - Can be based on simpler representations than analysis
  - Shape from texture (we will skip)
Objectives: 1) Discrimination/Analysis

The Goal of Texture Analysis

- **input image**
- **ANALYSIS**
- **“Same” or “different”**
- **True (infinite) texture**
- **generated image**

Compare textures and decide if they’re made of the same “stuff”.

Slide credit: Freeman

---

2) Synthesis

The Goal of Texture Synthesis

- **input image**
- **SYNTHESIS**
- **True (infinite) texture**
- **generated image**

Slide credit: Freeman
Representing textures

Observation: textures are made up of sub-elements, repeated over a region with similar statistical properties

Texture representation:
  – find the sub-elements, and represent their statistics
  • What filters can find the sub-elements?
    – Human vision suggests spots and oriented filters at a variety of different scales
  • What statistics?
    – Mean of each filter response over region
    – Other statistics can also be useful

Human texture perception
Bergen and Adelson, Nature 1988

Learn size-tuned filter responses.

Fig. 2 For new Textures consisting of Xs within a region composed of x's. The textures are placed in random orientations on a randomly perturbed lattice. a The bars of the Xs have the same length as the bars of the L's. The bars of the L's have been lengthened by 25%, and the intensity adjusted for the same mean luminance. Discriminability is reduced. b The bars of the Ls have been lengthened by 25%, and the intensity adjusted so the same mean luminance. Discriminability is improved. Because the responses of a size-tuned mechanism to image b, c, respond to image f, it responds to image e.

Fig. 3c Shows a Texture consisting of Xs within a region composed of x's. The textures are placed in random orientations on a randomly perturbed lattice. a The bars of the Xs have the same length as the bars of the L's. The bars of the L's have been lengthened by 25%, and the intensity adjusted for the same mean luminance. Discriminability is reduced. b The bars of the L's have been lengthened by 25%, and the intensity adjusted so the same mean luminance. Discriminability is improved. Because the responses of a size-tuned mechanism to image b, c, respond to image f, it responds to image e.
Derivative of Gaussian Filters

Measure the image gradient and its direction at different scales (use a pyramid).

- Blurring – to spatially integrate – get the mean
- Classification – to 4 classes – outputs of horizontal, vertical, both or neither are large
Extracting structure with filter banks

• Convolution with a filter – response is strong at places where the image structure looks similar to the filter
• Use multiple filters – to detect different types of image structures
• Commonly used filter bank – combination of bars and spots at different scale and orientation

Add more oriented filters (Malik & Perona, 1990)
Alternative: Gabor filters

**Gabor filters:** Product of a Gaussian with sine or cosine

Top row shows anti-symmetric (or odd) filters, bottom row the symmetric (or even) filters.

No obvious advantage to any one type of oriented filters.
Texture representation

- Given set of responses of filter banks
- Compute some statistics of the distribution of texture elements (e.g. flower field – many yellow spots)
- Zebra – many vertical bars
- Need to choose the scale over which the distribution of filter outputs is computed – see previous example
- What statistics to collect? (e.g. just consider mean of squared outputs)
- Mean and standard deviation of the filter outputs
- What if the outputs are correlated (e.g. spots and bars) … cabbage field example

Application: Texture-based Image Matching

Query image

Ordered list of best matches

Decreasing response vector similarity
Analysis using pyramids

• Choice of scale is important – how large of the neighborhood should I choose

• Analysis and synthesis using oriented pyramids
• Gaussian and Laplacian Pyramid
• Gaussian pyramid highly redundant
• Laplacian pyramid – approximated by differences of Gaussians
• Obtain image from Laplacian Pyramid – recover Gaussian and get the final resolution

The Laplacian Pyramid

• Building a Laplacian pyramid:
  – Create a Gaussian pyramid
  – Take the difference between one Gaussian pyramid level and the next (before sub-sampling)
• Properties
  – Also known as the difference-of-Gaussian function, which is a close approximation to the Laplacian
  – It is a band pass filter - each level represents a different band of spatial frequencies
• Reconstructing the original image:
  – Reconstruct the Gaussian pyramid starting at top layer
Gaussian Pyramid

Laplacian Pyramid

(note top image is from Gaussian)
Oriented pyramids

- Laplacian pyramid is orientation independent
- Apply an oriented filter to determine orientations at each layer
- This represents image information at a particular scale and orientation.
Final texture representation

- Form a Laplacian and oriented pyramid (or equivalent set of responses to filters at different scales and orientations).
- Square the output (makes values positive)
- Average responses over a neighborhood by blurring with a Gaussian
- Take statistics of responses
  - Mean of each filter output
  - Possibly standard deviation of each filter output

Texture Synthesis

- Application
- Texture synthesis for rendering
- How to obtain/generate a texture map reconstruction
- Hole filling
The texture synthesis problem

Generate new examples of a texture.

• **Original approach**: Use the same representation for analysis and synthesis
  
  – This can produce good results for random textures, but fails to account for some regularities

• **Recent approach**: Use an image of the texture as the source of a probability model
  
  – This draws samples directly from the actual texture, so can account for more types of structure
  
  – Very simple to implement
  
  – However, depends on choosing a correct distance parameter
This is like copying, but not just repetition

Efros and Leung method

- For each new pixel $p$ (select $p$ on boundary of texture):
  - Match a window around $p$ to sample texture, and select several closest matches
    - Matching minimizes sum of squared differences of each pixel in the window (Gaussian weighted)
    - Give zero weight to empty pixels in the window
  - Select one of the closest matches at random and use its center value for $p$
  - Size and shape of the neighborhood matter
Initial conditions for growing texture

- If no initial conditions are specified, just pick a patch from the texture at random
- To fill in an empty region within an existing texture:
  - Grow away from pixels that are on the boundary of the existing texture

Window size parameter

[Image of window size parameter with different texture patterns]
More Synthesis Results

Increasing window size
Further issues in texture synthesis

- How to improve efficiency
- Use fast nearest-neighbor search
- How to select region size automatically
- How to edit textures to modify them in natural ways
Texture synthesis

Task:
Make the donkey vanish

Fill in black region using
texture from white box

% Holefill.m
• clear; tic;
% Constants

% Change patchL to change the patch size used (patch size is
% 2*patchL+1).
• patchL = 10;
• patchSize = 2 * patchL + 1;
• randomPatchSD = 1; % Standard deviation for random patch selection
• showResults = 1;
% Read input image

• im = double(imread('donkey.jpg'))/255;
• [imRows, imCols, imBands] = size(im);
% Define hole and texture regions. This will use regions.mat if it exists,
% but otherwise will allow the user to select the regions.
• fid = fopen('regions.mat');
if (fid ~= -1) % file exists read regions from disk
disp('Loading regions');
fclose(fid);
load 'regions.mat' fillRegion textureRegion
else % file does not exist
disp('Select fill region'); % User define fill region
fillRegion = roipoly(im);
disp('Select texture region'); % User define texture region
textureRegion = roipoly(im);
save 'regions.mat' fillRegion textureRegion; % Save regions to disk
end; % else

% Perform the hole filling
while (nFill > 0)
disp(sprintf('Number of pixels remaining = %i', nFill));

% Set TODORegion to pixels on the boundary of fillRegion
TODORegion = fillRegion - imerode(fillRegion, [0,1,0;1,0,1; 0,1,0]);
[TTODO, jTODO] = find(TODORegion);
nTODO = length(iTODO);

while(nTODO > 0)
% Pick a random pixel from the TODORegion
r = rand;
pix = ceil(r * nTODO);
...
% Compute masked SSD of TODOPatch and textureIm
ssdIm = ComputeSSD(TODOPatch, patchL, TODOMask, textureIm, texImRows, texImCols, imBands);
• % Randomized selection of one of the best texture patches
  • [ssdImRows, ssdImCols] = size(ssdIm);
  • [sortVal, sortIndex] = sort(ssdIm(:));
  • \( r = \text{abs}(\text{random('Normal', 0, randomPatchSD})) \);
  • selectIndex = ceil(r);
  • selectIndex = max(1, min(ssdImRows * ssdImCols, selectIndex));

  ...
  • selectPatch = textureIm(iSelectRange, jSelectRange, :);

  % Copy patch into hole
  • imHole = CopyPatch(iPatchCentre, jPatchCentre, imHole, imRows, imCols, imBands, selectPatch, patchL, TODOMask);

  end; % while nTODO > 0

• % Output results
  • if (showResults)
    • figure;
    • imshow(imHole0);
    • hold on;
    • line([jTextureMin, jTextureMax], [iTextureMin, iTextureMin], 'color', 'w');
    • line([jTextureMin, jTextureMax], [iTextureMax, iTextureMax], 'color', 'w');
    • line([jTextureMin, jTextureMin], [iTextureMin, iTextureMax], 'color', 'w');
    • line([jTextureMax, jTextureMax], [iTextureMin, iTextureMax], 'color', 'w');
    • figure; imshow(im);
    • figure;
    • imshow(imHole);
    • end; % if showResults
  • imwrite(imHole, 'holefill.jpg');
  • toc