

Handed out: February 21, 2008

Due on: March 4, 2008

Problem Set #2

Problem 1: Image Pyramids

This problem uses pyramid image processing. Download and install the matlabPyrTools from <http://www.cns.nyu.edu/~eero/software.html>. When forming pyramid decompositions for this problem, you may always use the default decomposition filters. For this problem submit your MATLAB code and include a printout.

Subjectively, our visual world appears to us to be high resolution everywhere. However, we have much higher spatial resolution in the center of our field of view than in the periphery. In this problem, we will synthesize an image approximating our visual resolution as a function of eccentricity. Figure 1 shows a plot of the minimum angle resolvable as a function of the visual eccentricity. The visual eccentricity is measured in degrees away from the center of fixation. (From Rodieck, "The First Steps in Seeing", Sinauer, 1998).

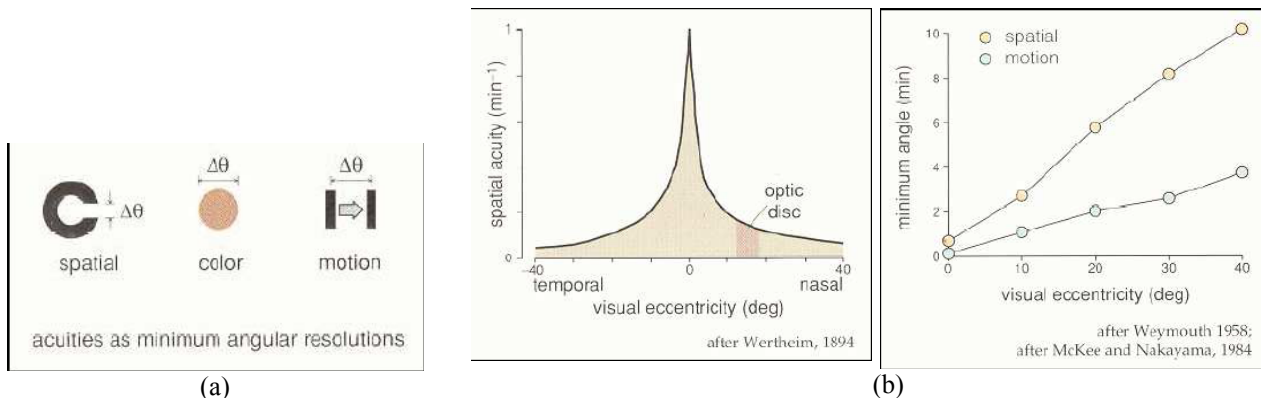


Figure 1: (a) Measures of acuity. (b) Plots of eccentricity versus acuity

Approximate acuity, a , in minutes of arc (60 minutes to a degree) as a function of eccentricity, e , in degrees, by the expression,

$$a = 0.23e + 0.7 \tag{1}$$

We will create an image with the effective spacing of the pixels equal to the angular size of the acuity limit. In the figure, that limit is defined as the white space between two ends of a circle. Adjacent black, white, black pixels could approximately represent that circle opening if the pixel spacing were equal to the angular size of the acuity limit. Assume that the image (or monitor) is square, and that you view it from a distance of two times the length of one side of the image. Where convenient, you may assume angles are small enough so that $\tan(\theta) \approx \theta$.

- (a) How many evenly spaced pixels per side does the image need to have in order that the highest resolution part of the image has one pixel per length of finest acuity? Assume that the highest resolution image point lies at half the maximum acuity, where maximum acuity is as specified by (1).
- (b) Let the upper left corner of the image be (0,0), and the right and bottom edges of the picture be at a distance 1 from this corner. Assume that the upper left corner is the center of fixation. What effective pixel spacing, as a function of these units, causes the pixel spacing to equal the spatial acuity for the corresponding eccentricity?
- (c) We can approximate images of this resolution by using a Gaussian pyramid, which generates images at different numbers of pixel samples, dividing the number of pixels by two at each level of the Gaussian pyramid. Start from an image at the full resolution of part (a). Each pyramid level increases the effective size of its pixels by a factor of two in each dimension. As a function of the coordinate system used in (b), by how many factors of two should the

resolution of the original image be reduced as a function of position in the image in order to simulate the human visual acuity, assuming the viewer stares at the upper left corner of the image?

- (d) The expression in (c) involves fractional pyramid levels. We can visually approximate images at those intermediate resolution levels by linearly interpolating between our Gaussian pyramid levels. On the class web site is a 2000x2000 image, which should be more than enough pixels for you. Crop that image to the desired resolution such that the upper left corner will be at half the maximum visual acuity, when viewed from 2 picture lengths away. Use the Gaussian pyramid to create an image that simulates the fall-off in visual acuity, assume the fixation point is at the upper left corner. At any given pixel, determine the coefficients for interpolating between images by linearly interpolating the corresponding pixel dimensions.

Hint: You will want to use the upBlur function to transform the Gaussian pyramid levels to all have the same number of pixels. Assume that a pyramid level after upBlur has effectively the same number of pixels (in terms of picture content) as the original pyramid band before the upBlur operation. That is a reasonable approximation (take a probability course for the details that we're glossing over here).

Problem 2: Noise Removal

We study noise removal using a wavelet/QMF pyramid.

- (a) Download the three "training set" images from the web site. Convert each to a QMF pyramid. For the subband selecting vertical oriented structure, at the second highest resolution level, plot the histogram of the coefficient values, for each of the 3 training set images. Do these images share the subband histograms described in class?
- (b) Add independent Gaussian random noise to each pixel of each of the training images (mean zero, standard deviation 10). Reform the QMF pyramids, and plot the histograms for the same subbands used in (a). Discuss any changes.
- (c) For each image, make a scatter plot plotting noisy versus clean pixel values. Make similar scatter plots of noisy versus clean subband coefficients, for the single subband used in part (a). Discuss how these plots show that one representation is better than the other for noise removal.

The last two parts of this problem are potentially time-consuming. They demonstrate how the statistical observations above could be transformed into an image denoising algorithm.

- (d) Find a parametric form which estimates the clean subband coefficient value as a function of noisy input value. The values of the free parameters could in general depend on the original image statistics, the noise level, and the pyramid band. The original image statistics and the noise level may be estimated from the noisy image. However, for this exercise, we'll assume the fit is only a function of the pyramid band. Matlab's `fminsearch` function may be used to find the parameters which minimize the squared error from the observed relationships in the noisy and clean training images (such as the scatter plots of (c)). Combining the data for the 3 images, determine the parameters of your parametric form for each of the 3 subbands (vertical, horizontal, and diagonal) over the 4 finest pyramid levels.
- (e) Download the noisy test image, which is made from a clean image corrupted by noise at the same level as (b). Use the parametric forms from (d) to estimate a denoised image.

Problem 3: Texture Synthesis

In this problem you will implement the Efros and Leung algorithm for texture synthesis discussed in Section 9.3 of Forsyth and Ponce. In addition to reading the textbook you may also find it helpful to visit Efros' texture synthesis website: <http://www.cs.berkeley.edu/~efros/research/synthesis.html>, from which many of the implementation details described below can be found.

As discussed in class, the Efros and Leung algorithm synthesizes a new texture by performing an exhaustive search of a source texture for each synthesized pixel in the target image, in which sum-of-squared differences (SSD) is used to associate similar image patches in the source image with that of the target. The algorithm is initialized by randomly selecting a 3x3 patch from the source texture and placing it in the center of the target texture. The boundaries of this patch are then recursively filled until all pixels in the target image have been considered.

Implement the Efros and Leung algorithm as the following MATLAB function:

```
synthIm = SynthTexture(sample; w; s)
```

where `sample` is the source texture image, `w` is the width of the search window, and `s=[ht wt]` specifies the height and width of the target image `synthIm`. As described above, this algorithm will create a new target texture image, initialized with a 3×3 patch from the source image. It will then grow this patch to fill the entire image. As discussed in the textbook, when growing the image unfilled pixels along the boundary of the block of synthesized values are considered at each iteration of the algorithm. A useful technique for recovering the location of these pixels in MATLAB is using *dilation*, a morphological operation that expands image regions (it performs the opposite function of the *erode* operation from the previous problem set). Use MATLAB's `imdilate` and `find` routines to recover the unfilled pixel locations along the boundary of the synthesized block in the target image. In addition to the above function we ask you to write a subroutine that for a given pixel in the target image, returns a list of possible candidate matches in the source texture along with their corresponding SSD errors. We ask this function to have the following syntax:

```
[bestMatches; errors] = FindMatches(template; sample; G)
```

where `bestMatches` is the list of possible candidate matches with corresponding SSD errors specified by `errors`. `template` is the $w \times w$ image template associated with a pixel of the target image, `sample` is the source texture image, and `G` is a 2D Gaussian mask discussed below. This routine is called by `SynthTexture` and a pixel value is randomly selected from `bestMatches` to synthesize a pixel of the target image. To form `bestMatches` accept all pixel locations whose SSD error values are less than the minimum SSD value times $(1+\epsilon)$. To avoid randomly selecting a match with unusually large error, also check that the error of the randomly selected match is below a threshold δ . Efros and Leung use threshold values of $\epsilon = 0.1$ and $\delta = 0.3$.

Note `template` can have values that have not yet been filled in by the image growing routine. Mask the template image such that these values are not considered when computing SSD. Efros and Leung suggest using the following image mask:

```
Mask = G .* validMask
```

where `validMask` is a square mask of width w that is 1 where `template` is filled, 0 otherwise and `G` is a 2D zero-mean Gaussian with variance $\sigma = w/6.4$ sampled on a $w \times w$ grid centered about its mean. `G` can be pre-computed using MATLAB's `fspecial` routine. The purpose of the Gaussian is to downweight pixels that are farther from the center of the template. Also, make sure to normalize the mask such that its elements sum to 1. Test and run your implementation using the grayscale source texture image `rings.jpg`, with window widths of $w = 5; 7; 13$, `s=[100 100]` and an initial starting seed of $(x; y) = (4; 32)$. Explain the algorithm's performance with respect to window size. For a given window size, if you re-run the algorithm with the same starting seed do you get the same result? Why or why not? Is this true for all window sizes?

Please include the synthesized textures that correspond to each window size along with answers to the above questions and a printout of your code in your writeup. Also, submit the files `synthtexture-[last name].m` and `findmatches-[last name].m` by email to cse527_submit@cs.sunysb.edu