

**CMPEN/EE 454 Computer Vision I**  
**Project Assignment 1**  
**Due Monday October 7**

**An Application of Image Pyramid Representations**

**Introduction and outcomes**

In this assignment you will learn how to generate multi-scale pyramid representations of images and use them in a computer graphics application, namely blending portions of two images together. Specific tasks will include generating a multi-scale Gaussian pyramid, generating from that a Laplacian of Gaussian pyramid (more precisely, it will be a Difference of Gaussian pyramid that approximates the LoG pyramid), choosing two images to blend together and generating a binary mask saying which pixels should come from which image, and finally, doing the blend using the Gaussian and LoG pyramids computed for each image.

**Generating a Multi-scale Gaussian Pyramid**

In Lecture 8 on Pyramids and Scale Space, we introduced three basic routines for working with Gaussian pyramids. These are:

- 1) blur – convolving an image with a 2D Gaussian filter
- 2) downsample – reduce size of an image by two in the row and col dimensions
- 3) upsample – increase size of image by two in row and col dimensions

Robert Collins  
CSE486

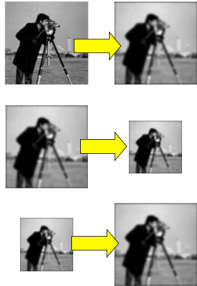
**Working with a Gaussian Pyramid**

**Basic Functions:**

Blur (convolve with Gaussian to smooth image)

DownSample (reduce image size by half)

Upsample (double image size)

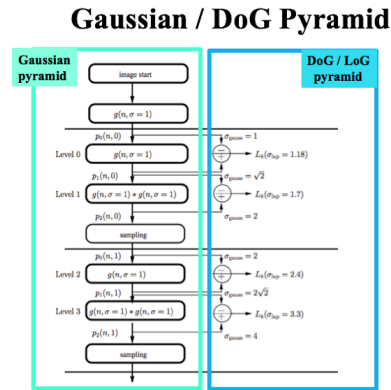
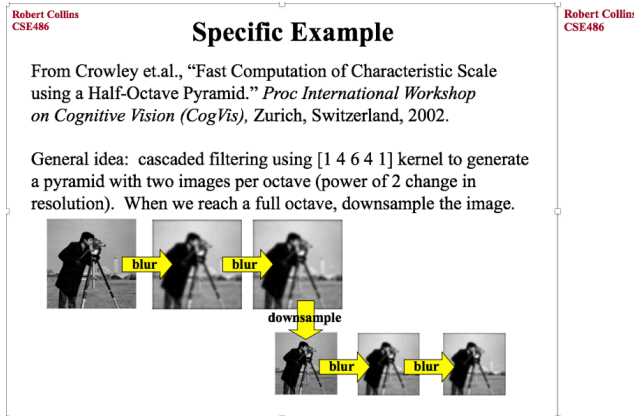


The diagram shows three rows of image transformations. Each row consists of a small input image, a yellow arrow pointing to the right, and a larger output image. The first row is labeled 'Blur' and shows a slightly more smoothed version of the input image. The second row is labeled 'DownSample' and shows a smaller version of the input image. The third row is labeled 'Upsample' and shows a larger version of the input image.

See slides in lecture 8 for more details on implementing each function.

To generate our Gaussian pyramid, we will follow a strategy in the paper [Crowley et.al. 2002], which generates a pyramid based on a Gaussian blur filter with standard deviation 1. When writing your 2D blur function, recall that the 2D Gaussian filter is separable, so you can blur along rows and then cols with a 1D filter. Also recall a simple approximation to a 1D Gaussian filter with  $\sigma=1$  is given by  $[1\ 4\ 6\ 4\ 1]/16$ .

The Crowley paper algorithm is illustrated in the figure below.



Note that, for a given image scale (size) there are three images at that scale. In the flowchart, for example, they are labeled  $p_0, p_1, p_2$ . Image  $p_1$  is produced by blurring  $p_0$  once with the Gaussian filter  $g$ , that is  $p_1 = g * p_0$ . The next image  $p_2$  is produced by blurring  $p_1$  TWICE with that Gaussian,  $p_2 = g * g * p_1$ . We discussed in class how cascading Gaussians in this way serves to double the blur factor sigma between incoming  $p_0$  and outgoing  $p_2$ . At that point,  $p_2$  is downsampled by two, becoming  $p_0$  for the next “octave” of the pyramid representation.

When to stop blurring and downsampling is up to you. I would test the size of the image after downsampling and stop if either the number of rows or number of cols falls below some size value  $k$ . If you are fancy, I imagine you could determine ahead of time how many downsample operations it will take to get to size  $k$ , based on some function involving log base 2.

By the way, what we are describing works for single channel (grey scale) images. If you want to use the algorithm for **three-channel RGB color images**, you will run the algorithm three times to generate three pyramids, one for red channel, one for green, and one for blue.

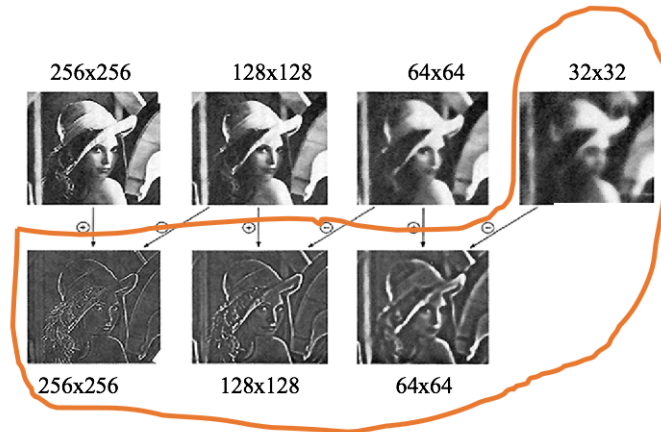
## Generating a Multi-scale Laplacian Pyramid

Our multi-scale Laplacian pyramid representation gets generated as a byproduct of the Gaussian pyramid. Referring to the right-hand side the flowchart shown in the picture above, we end up with two Laplacian images for each octave, generated approximately as the difference of two Gaussians. The first one is generated as  $p_1 - p_0$ , the second is generated as  $p_2 - p_1$ . As the size of the Gaussian-blurred images gets smaller and smaller, the generated Laplacian images will also get smaller and smaller, thus also forming a pyramid.

## An Efficient Image Coding

Turning our attention to another paper [Burt and Adelson, 1983a] we learn that we can decompose the content of an image into a small Gaussian-blurred image together with a progressively larger set of Laplacian of Gaussian images (see figure below). In a signal processing sense, the Gaussian is a low-pass filter (only low frequency, slowly varying information is kept), and Laplacian of Gaussian is a band-pass filter (a narrow range of image frequencies are kept), so the representation depicted is decomposing the original image into a range of frequencies, stored as separate images, that can be pieced back together to regenerate a

full frequency approximation of the original image. We are going to leverage this image decomposition method to do image blending.



In the context of the pyramids we discussed generating above, the small Gaussian filtered image is the last subsampled Gaussian we generate before deciding to stop because number of rows or cols is now less than  $k$  pixels (recall our discussion above). The Laplacian images are the set of difference of Gaussian images running down the right-hand side of the Crowley flowchart (unlike the picture from the Burt and Adelson paper, we will have TWO Laplacian images for each image size).

Given this representation, we should be able to generate a good approximation to the original image by starting with the small, low-resolution Gaussian image (in this example it is  $32 \times 32$ ), upsampling it (to  $64 \times 64$ ), subtracting from that the two Laplacian images at the next highest image scale ( $64 \times 64$ ), upsampling that result (to  $128 \times 128$ ), and so on, until we reach the original image size and have incorporated information from the Laplacian image(s) at that size.

**Milestone:** To make sure your implementation is on the right track, decompose an image into the representation described above, which will first involve generating the Gaussian and Laplacian pyramids. Then, starting with the lowest resolution Gaussian image, work your way back towards higher resolutions by upsampling and subtracting Laplacians, until you recover an approximation of the original image. Compare and see how close your recovered image is to the original image. They may not be exactly the same, but as long as they are similar, you can proceed. Can you think of any modifications to the Crowley pyramid generation and/or Burt and Adelson upsampling algorithms that will make the recovered image exactly the same as the original image you started with? Hint: Think about how to recover the small amount of information that is lost during a downsampling operation.

### Image Blending Application

OK, now for some fun. You will need to find or generate two images that are the same size, and that have some parts of one image that you would like to replace with pixel values in the corresponding location of the other image. In addition to the two images, you need to generate a binary mask image (0 and 1) that tells which pixels to take from which image (e.g. 0 pixels

come from image A and 1 pixels come from image B). You can generate a binary mask however you like. In matlab, the `roipoly` function is useful for this task. Or you use a “paint” program to draw a region by hand.

Generate Gaussian and Laplacian pyramids for both of the images being used as source images for blending. Also generate a Gaussian pyramid for the binary mask image (make sure you convert the binary mask image to floating point first because you want to be able to produce numbers between 0 and 1). Values in this latter Gaussian pyramid will be used as weights for blending each level of the Laplacian pyramid images.

To generate a blended image, let  $G_A$  and  $G_B$  be the two lowest resolution images from the Gaussian pyramids for A and B. Combine them by weighted blending using the lowest resolution image  $W$  from the Gaussian pyramid generated from the binary mask image:

$$\text{Blended } G = G_A .* (1-W) + G_B .* W$$

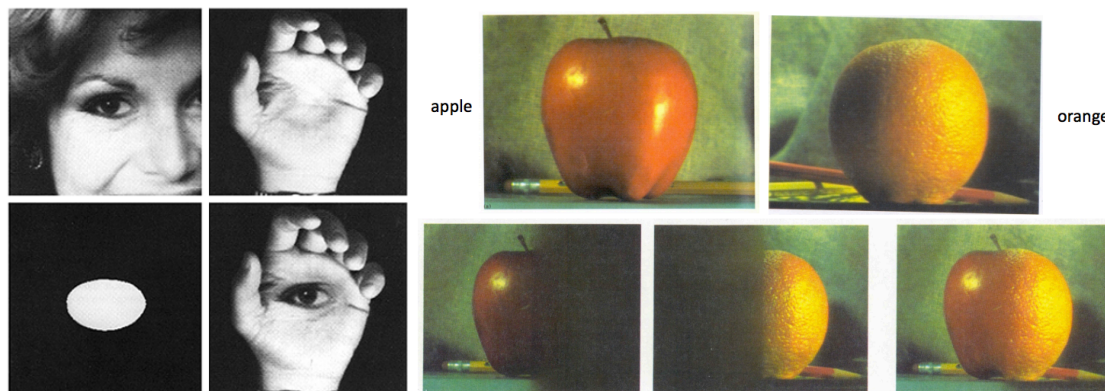
where we have used matlab’s “`.*`” notation to mean pixelwise multiplication.

Likewise, at every level of the two Laplacian pyramids, if we have Laplacian images  $L_A$  and  $L_B$ , combine them also, using the weight image  $W$  taken from the corresponding level of the Gaussian pyramid for the binary mask:

$$\text{Blended } L = L_A .* (1-W) + L_B .* W$$

In this way you not only will produce a blended low resolution image, but also a blended Laplacian pyramid to go along with it. Now, regenerate a full sized image from this set of blended images, by repeatedly upsampling and subtracting Laplacians in the same manner as you did before for a single image. The result should be a nicely blended composite of your two original images! This multi-scale approach to blending was first described in [Burt and Adelson, 1983b].

Two suggestive examples are shown below



## What to Hand In

Half of your grade will be based on submitting a runnable program, and the other half will be based on a written report discussing your program, design decisions, experimental observations and results. The report should include at least the following things:

- a) Summarize in your own words what you think the project was about. What were the tasks you performed; what did you expect to achieve?
- b) Present an outline of the procedural approach along with a flowchart showing the flow of control and subroutine structure of your Matlab code. Explain any design decisions you had to make. For example, what data structure did you use to represent Gaussian and Laplacian pyramids? Did you make any modifications to the basic algorithms presented in the Crowley or the Burt and Adelson papers?
- c) Image coding milestone. What do you observe about the behavior of your code for pyramid decomposition and image regeneration? Does it seem to work the way you think it should? For a sample image, can you regenerate the original image after decomposing it into a low-res Gaussian and a Laplacian pyramid? Show some intermediate results of the process, such as images from different levels of the Gaussian pyramid and the Laplacian pyramid.
- d) Experimental results. Show examples output by your blending application – the images chosen, the binary region mask, and the resulting blended image. Do some examples work better than others? How did you choose images to use for the blending application? How did you generate a binary mask image? Did you develop any rules of thumb by trial and error to tell you what to look for when choosing images that would be good to blend?
- e) Document what each team member did to contribute to the project. It is OK if you divide up the labor into different tasks. This is also where you let us know if any of your teammates were slacking off on this project.

## References

[Burt and Adelson 1983a]

Burt and Adelson, “The Laplacian Pyramid as a Compact Image Code.” *IEEE Transactions on Communication*, COM-31:532-540, 1983.

[Burt and Adelson 1983b]

Burt and Adelson, “A Multiresolution Spline with Application to Image Mosaics.” *ACM Transactions on Graphics*, Volume 2, pp. 217-236, 1983.

[Crowley et.al. 2002]

Crowley et.al., “Fast Computation of Characteristic Scale using a Half-Octave Pyramid.” *Proc International Workshop on Cognitive Vision (CogVis)*, Zurich, Switzerland, 2002.

