Project 1 Writeup: Hybrid Images

Dustin Cook

Quick Description

- MatLab is what I chose to implement this project.
- The main function is gen_hybrid_image that takes highFreqImg, lowFreqImg, and a method.
- Method can be either FFT or my_filter
- The underlying functions gen_hybrid_FFT and gen_hybrid_myfilter are callable from matlab terminal as well. They are given defaults for kernel and sigma in their code near top of functions.
- A pyramid function is provided to construct an image pyramid. This function is called makePyramid(img, n) where n is the number of smaller images you desire, each one being halved in size.

About hybrid images

The idea behind a hybrid image is to fool the human senses by hiding one image within another. It is similar to a mosaic, as you walk away from the image, more hidden details become apparent to your eyes, and another image appears. How do we do this? We do this with low frequencies and high frequencies. Up close, your eyes catch mostly high frequencies, but as the image becomes smaller, lower image frequencies become apparent. A Hybrid image is taking the low frequencies of one image, and merging them with the high frequiencies of another image, thus creating the mosaic effect. For this project, two methods of merging these frequencies were implemented, one using Fourier transform, another using spatial filtering.

Hybrids using Fourier Transform

When it comes to a 2d image, a Fourier transform is exactly what we are looking for implementing this idea because creating a hybrid image with a Fourier transform is as easy

as computing the transform for both images, taking the middle of the low frequency FFT, and the outer ring of the high frequency FFT, and merging them together. Taking the inverse FFT of this new Fourier transform gives an image that has all of the high frequencies of the high frequency image, and all of the low frequencies of the low frequency image. In order to cut out the center and outer ring, I used MatLabs fspecial gaussian filter to produce a low frequency filter. Depending on the given sigma value, this filter produces a matrix the same dimensions as the image, but after being normalized, contains values between 0-1, with higher values close to the center. A high frequency filter is achieved by taking 1 - (low pass).

Hybrids from the spatial domain

In order to accomplish this, I wrote my own image filter that, given a kernel, performs a convolution on the image. In doing this, an FFT is not required to get the high and low frequencies because this gives you the low frequencies of the image, and as for getting the high frequencies of the desired high frequency image, all you have to do is subtract the low pass filter given by my_imfilter from the original image. Then you can combine these two to achieve a similar effect as you would with the FFT method. Doing it this way does appear to be more natural, but FFT tends to more accurately cut the frequencies. however, in a lot of cases, they tend to appear fairly similar when you adjust the sigma values and kernel. Interesting though, the spatial filter can have a hard time combining some images together, such as the weeping angel (see Figure 2).

Code Example

```
es = imread('Data/einstein.bmp');
mm = imread('Data/marilyn.bmp');
imshow(makePyramid(gen_hybrid_image(es, mm, "FFT"), 4));
% generates a pyramid with 4 images with the generated hybrid
```

Questions

Q1 Convolution is a general purpose filter effect for images. It can serve many purpose such as smoothing or sharpening an image. It requires a kernel, and an image and returns the image modified. Each element is modified by the values of its neighbors.

Q2 Convolution is a linear operation on an image that is the measure of effect of one signal on the other signal. Correlation is the measurement of the similarity between

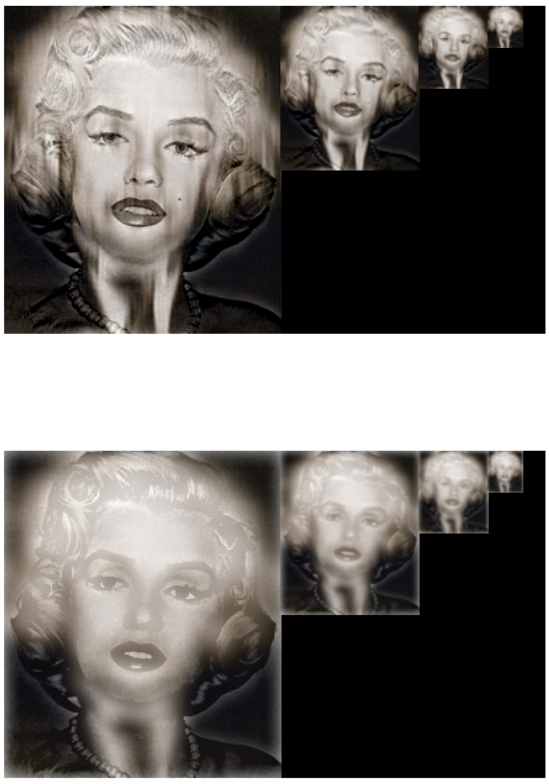

Figure 1: *Above:* FFT result. *Below:* Spatial result.

two signals. They may differ if you are not looking for a low pass filter with your convolution or the kernels are not symmetrical.

Q3 The filter being high or low merely strips out a frequency past a certain threshold. The high pass filter is the same as the image subtracted by its low pass because, you only have frequencies above that threshold remaining. For the low filter, you get a blurred image because there are no strong frequencies to tell your eyes where something begins and ends. As for a high filter image, you get very strong lines but no transitioning, and this can be kind of hard to look at. See (Figure 3).

Q4 In order to view this, I wrote a couple test functions. TestRunner(img, n, small, maxKernel) runs testFilterTime(img, n, small, kernel) where img is an image, n is the number of reductions to make to the image to get to small, small is a vector size 2 that represents the smallest test value for testFilterTime, kernel is the kernel size to test, and maxKernel, the maximum kernel to test. testRunner calls testFilterTime with an

Figure 2: *Above:* FFT result. *Below:* Spatial result.

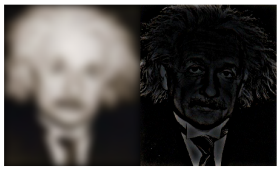

Figure 3: *Left:* Low filter. *Right:* High filter.

increasing kernel size, and collects a vector of times in seconds to run the filter on all image sizes with the kernel goes from largest image to smallest. I then used surf to make a 3d plot of this where z (up) represents time in seconds, x (left) represents image size from small to big, and y (right) represents kernel size. I ran twice with n = 8 and a max kernel of [3 3] and [15 15]. I noticed that the time was largest for the

15x15 tests torwards the max kernel sizes with a smallish image.

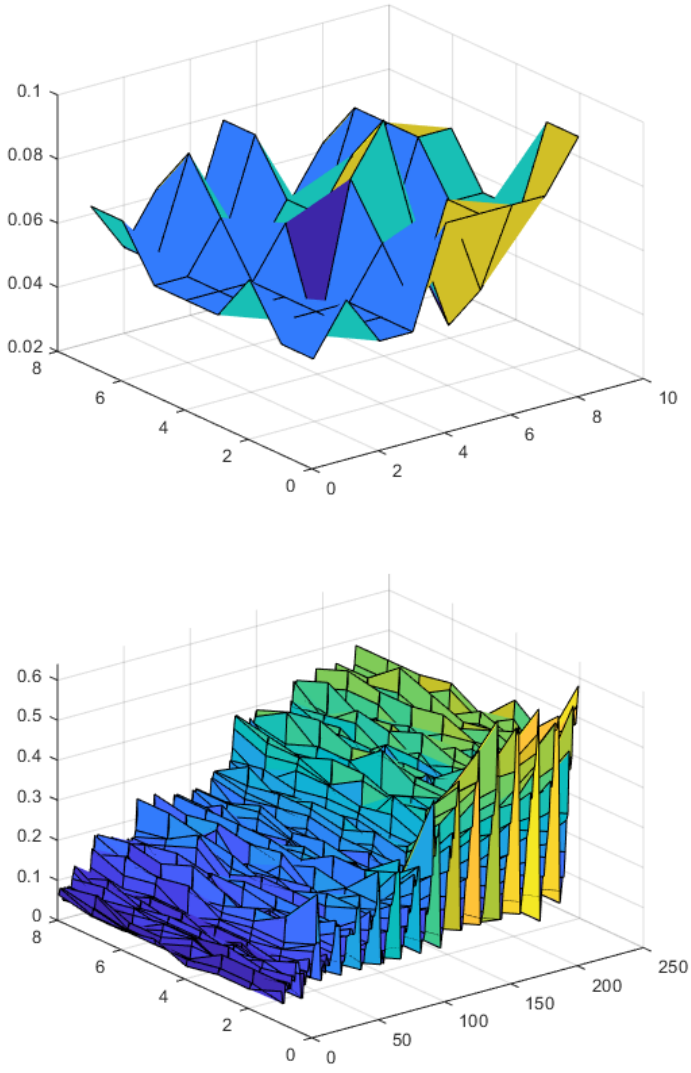

Figure 4: *Above:* Max kernel: 3x3. *Below:* Max kernel 15x15