Homework Assignment 2

15-463/663/862, Computational Photography, Fall 2017
Carnegie Mellon University

Due Thursday, Sep. 28, at 11:59pm

The purpose of this assignment is to explore Eulerian video magnification [2] as a tool for revealing temporal image variations that are difficult to visualize, or even see at all. While the assignment describes most of the steps you need to perform, it is highly recommended that you watch this video and read the associated paper describing the technique. There is a "Hints and Information" section at the end of this document that is likely to help.

1. Implement Eulerian Video Magnification (100 points)

Throughout this problem, you will use the videos `face.mp4` and `baby2.mp4` included in the `./data` directory of the homework ZIP archive. You will use these videos as test sequences for your implementation of Eulerian video magnification.

How can you amplify image variations that are hard to see with the naked eye? The insight is that some of these hard-to-see changes occur at particular temporal frequencies that you can augment using simple filters in the frequency domain. For example, to magnify pulse, you can look at pixel variations with frequencies between 0.4 and 4Hz, which correspond to 24 to 240 beats per minute. Given the above insight, Eulerian video magnification is done as a sequence of the following steps:

1. If your video has color, transform it to an appropriate color space.
2. Create a Laplacian pyramid for each video frame.
3. Band-pass filter the time series for each pixel, on all levels of the pyramid.
4. Magnify bands of interest by some scale.
5. Reverse the Laplacian pyramid and undo the color transform to obtain the final output.

After running this pipeline, you get results that, for the `face.mp4` video, should look *something* like what is shown in Figure 1, or the results videos shown in the project website.

The exact result can vary greatly, depending on the choices you make in your implementation of the algorithm. In fact, the hardest part of the process is exactly finding the right parameters to get the desired magnification effect. For example, one can change the size of the Laplacian pyramid, multiply the time series corresponding to the value of a pixel by different scale factors at different levels of the pyramid, or attenuate the magnification when adding the augmented band-passed signals to the original ones. The choice of the band-pass filter (e.g., the range of frequencies it passes/rejects, its order, etc.) can also influence the obtained results. We recommend that you implement your pipeline in such a way that you can easily re-run it with different sets of parameters, multiple times. This parameter exploration is part of the project and is *very time consuming*, especially since you are processing video sequences and not just still images. *So make sure you start the assignment early!*

Initials and color transformation (5 points). Load the video file into Matlab, extract its frames, and convert them to double-precision in the range $[0, 1]$. (See help for `VideoReader` and `GetFrames` about reading video files in Matlab.)

Then, convert each of the frames to the YIQ color space. The YIQ color space is particularly suggested for Eulerian magnification because it allows to easily amplify intensity and chromaticity independently of each other. All subsequent steps in this problem will be performed on each color channel independently. (See help for `rgb2ntsc` and `ntsc2rgb` about converting between RGB and YIQ.)

Laplacian pyramid construction (20 points). The next step towards motion magnification is to construct a Laplacian pyramid *for every frame in the video sequence*. This can be done in many, approximately

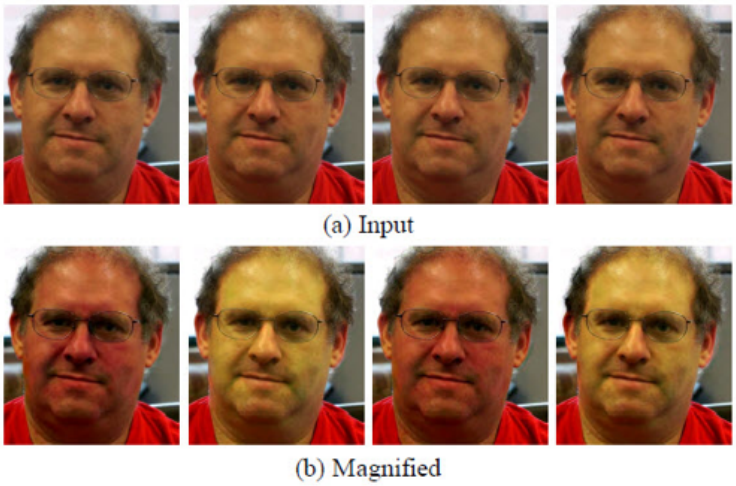

Figure 1: One set of possible results for the `face.mp4` video.

equivalent ways. One was discussed in class: Iterate steps of Gaussian blurring, differencing, and downsampling, then store the image differences and the lowest resolution image. An alternative is to directly use Laplacian-of-Gaussian filtering. You can find more details in the slides for lectures 3 (linear filtering) and lectures 5 (pyramids), the Szeliski chapters referenced as reading at the end of those slides, and the original Laplacian pyramid paper by Burt and Adelson [1].

Whichever method you use, make sure to describe in detail your implementation in your PDF report.

Temporal filtering (40 points). At each spatial level of the Laplacian pyramid, look at the values of each pixel across all video frames. This time series of values is a temporal signal that you can filter like any other signal, as we discussed in class. You will need to apply a band-pass filter to this time series. The choice of the band-pass filter is crucial, and we recommend designing and visualizing the filter using Matlab filter design tools. In order to filter the time series of all pixels fast, you should perform this operation in the frequency domain, since multiplication is faster than convolution. (See Hints and Information.)

You can check the Eulerian video magnification paper for details on the parameters they used on the `face.mp4` and `baby2.mp4` videos. In your PDF report, you should discuss and justify the choices you made for the band-pass filters.

Pixel change magnification (10 points). After extracting the frequency band of interest, you need to amplify it and add the result back to the original signal. You should experiment with different magnification scales.

Image reconstruction (25 points). After amplifying the signals, it is now time to collapse the Laplacian pyramids into a single image per frame. As discussed in class, this will require (among other steps) upsampling images from one level of the pyramid to the next lower one. You can use bilinear interpolation to do this, or any other way you come up with. In your PDF report, make sure to describe how you implement the pyramid inversion.

Once you have inverted the Laplacian pyramid, all that is left to do is convert your processed frames from YIQ back to RGB.

Note that, in addition to implementation of the above, the number of points for this question also includes evaluating your final results and the parameter exploration you performed.

2. Bonus: Capture and motion-magnify your own video(s) (up to 30 points)

Use your own camera (or one that you borrowed) to capture your own video sequences. A phone camera is fine for this. Try to find scenes where there are interesting motion effects to visualize.

Then, apply Eulerian video magnification to the videos you captured. You will need to search for a whole new set of parameters to use when doing motion magnification on your own videos, in order for the final results to look good.

The total number of points you will get from this bonus question will depend on how visually compelling the motion effect you capture and magnify is.

Deliverables

As described on the course website, solutions are submitted through Canvas. Your solution should be an archive (e.g., a ZIP file) that includes the following:

- A PDF report explaining what you did for each problem, including answers to all questions asked throughout Problem 1, as well as any of the bonus problems you choose to do. The report should include any figures and intermediate results that you think may help. Make sure to include explanations of any issues that you may have run into that prevented you from fully solving the assignment, as this will help us determine partial credit. The report should also explain any additional `.MP4` files you include in your solution (see below).
- All of your Matlab code, including code for the bonus problems, as well as a `README` file explaining how to use the code.
- At least two `.MP4` files, showing the final motion-magnified videos you created in Problem 1 for `face.mp4` and `baby2.mp4`. You can also include additional `.MP4` files corresponding to parameter settings (e.g., band-pass filter or Laplacian pyramid settings) other than your final ones, if you think they show something important.
- If you do Bonus Problem 2: The original video files you used, as well as the corresponding `.MP4` files after processing them.

Please organize your solution submission using the following file structure:

```
.zip
├─ .pdf ..... The PDF report.
├─ src/ ..... Contains all Matlab M-files and the README file explaining how to use the code.
└─ data/ ..... Contains all image, video, and other data files.
```

Hints and Information

- To design and visualize the band-pass filter you use in your pipeline, you can use the functions `fdatool` and `fvtool`. You should take a look at this tutorial from Mathworks on how to use these funtions.

  To make the filter design process easier, in the `./src` directory of the homework ZIP archive, we provide you with a `butterworthBandpassFilter` function. This function generates a Butterworth band-pass filter of a particular order. It was generated with `fdatool`, and also uses the function `fdesign.bandpass`, which you can read more about in the Matlab documentation. Use of this filter is optional.

- To convert a one-dimensiona discrete signal (e.g., a temporal signal) to the Fourier domain, you can use the Fast Fourier Transform, which in Matlab is provided by the function `fft`. However, you need to be careful about `fft`'s output format! As explained in this tutorial, the DC component of `fftx = fft(x)`, for `x` a 1D signal, is the first element `fftx(1)` of the array. If `x` has an even number of samples, then the magnitude of the FFT will be symmetric, such that the first `(1+nfft/2)` points are unique, and the rest are symmetrically redundant. In this case, the element `fftx(1+nfft/2)` is the Nyquist frequency component. However, if the number of samples of `x` is odd, the Nyquist frequency component is not evaluated, and the number of unique points is `(nfft+1)/2`.

  If you choose to use the provided `butterworthBandpassFilter` function, you will need to get the frequency components of the filter. This can be done by using function `freqz`, by passing the filter and the length of the output that you want, for example,

```
fftHd = freqz(Hd,NumSamples));
```

As above, be careful about the order in which `freqz` outputs the frequency components.

Credits

Most of the write-up for this assignment came from previous offerings of this class. Video data came from the original Eulerian magnification paper.

References

[1] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[2] H.-Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. T. Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Trans. Graph. (Proceedings SIGGRAPH 2012)*, 31(4), 2012.