Image Processing - 67829

Exercise 3: Image Pyramids & Pyramid Blending

Due date: 28/12/2016

1 Overview

This exercise deals with image pyramids, low-pass and band-pass filtering, and their application in image blending. In this exercise you will construct Gaussian and Laplacian pyramids, use these to implement pyramid blending, and finally compare the blending results when using different filters in the various *expand* and *reduce* operations.

2 Background

Before you start working on the exercise it is recommended that you review the lecture slides describing image pyramids and pyramid blending.

3 Image Pyramids

3.1 Gaussian & Laplacian pyramid construction

Implement two functions that construct a Gaussian pyramid and a Laplacian pyramid of a given image. The functions should have the following interface:

```
pyr, filter_vec = build_gaussian_pyramid(im, max_levels, filter_size)
pyr, filter_vec = build_laplacian_pyramid(im, max_levels, filter_size)
```

with the following input arguments:

`im` – a grayscale image with double values in $[0, 1]$ (e.g. the output of ex1's `read_image` with the `representation` set to 1).

`max_levels` – the maximal number of levels in the resulting pyramid.

`filter_size` – the size of the Gaussian filter (an odd scalar that represents a squared filter) to be used in constructing the pyramid filter (e.g for `filter_size = 3` you should get $[0.25, 0.5, 0.25]$).

Comments:

- Both functions should output the resulting pyramid `pyr` as a standard python array (i.e. not numpy's array) with maximum length of `max_levels`, where each element of the array is a grayscale image.

- The functions should also output `filter_vec` which is 1D-row of size `filter_size` used for the pyramid construction. This filter should be built using a consequent 1D convolutions of [1 1] with itself in order to derive a row of the binomial coefficients which is a good approximation to the Gaussian profile. The `filter_vec` should be normalized.

- Note that when performing both the expand and reduce operations you should convolve with this `filter_vec` twice - once as a row vector and then as a column vector (for efficiency). Also note that to maintain constant brightness in the expand operation `2*filter` should actually be used in each convolution.

- The pyramid levels should be arranged in order of descending resolution s.t. `pyr[0]` has the resolution of the given input image `im`.

- The number of levels in the resulting pyramids should be the largest possible s.t. `max_levels` isn't exceeded and the minimum dimension (height or width) of the lowest resolution image in the pyramid is not smaller than 16. You may assume that the input image dimensions are multiples of $2^{(\texttt{max\_levels}-1)}$. Finally, you should use the function `scipy.ndimage.filters.convolve` to apply the filter on the image for best results.

- For consistency, you should down-sample an image by taking its even indexes (assuming zero-index and of course after blurring), and up-sample by adding zeros in the odd places.

3.2 Laplacian pyramid reconstruction

You should also implement the reconstruction of an image from its Laplacian Pyramid.

```
img = laplacian_to_image(lpyr, filter_vec, coeff)
```

Comments:

- `lpyr` and `filter_vec` are the Laplacian pyramid and the filter that are generated by the second function in 3.1.

- `coeff` is a vector. The vector size is the same as the number of levels in the pyramid `lpyr`. Before reconstructing the image `img` you should multiply each level i of the laplacian pyramid by its corresponding coefficient `coeff[i]`.

- Notice that only when this vector is all ones we get the original image (up to a negligible floating error, e.g. maximal absolute difference around 10^{-12}). When some values are different than 1 we will get filtering effects.

Q1: What does it mean to multiply each level in a different value? What do we try to control on?

3.3 Pyramid display

To facilitate the display of pyramids, implement the following two functions:

```
res = render_pyramid(pyr, levels)
display_pyramid(pyr, levels)
```

where: `pyr` is either a Gaussian or Laplacian pyramid as defined above. `levels` is the number of levels to present in the `result` $\leq$ `max_levels`. `res` is a single black image in which the pyramid levels of the given pyramid `pyr` are stacked horizontally (after stretching the values to $[0, 1]$) as follows:

Figure 1: Here a 4-level Gaussian pyramid was rendered.

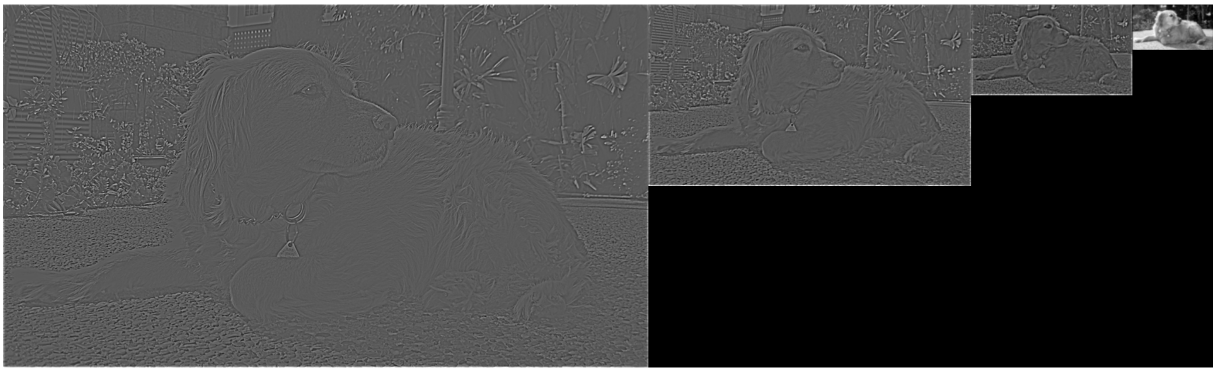

Figure 2: Here a 4-level Laplacian pyramid was rendered.

Comments:

- The function `render_pyramid` should only return the big image `res`.
- The function `display_pyramid` should use `render_pyramid` to internally render and then display the stacked pyramid image using `plt.imshow()`.
- You should stretch the values of each pyramid level to $[0, 1]$ before putting it in the big black image. Note that you should stretch the values of both pyramid types: Gaussian and Laplacian

4 Pyramid Blending

Implement pyramid blending as described in the lecture. The pyramidBlending function should have the following interface:

```
im_blend = pyramid_blending(im1, im2, mask, max_levels, filter_size_im, filter_size_mask)
```

where:

`im1, im2` – are two input grayscale images to be blended.

`mask` – is a boolean (i.e. `dtype == np.bool`) mask containing `True` and `False` representing which parts of `im1` and `im2` should appear in the resulting `im_blend`. Note that a value of `True` corresponds to 1, and `False` corresponds to 0.

`max_levels` – is the `max_levels` parameter you should use when generating the Gaussian and Laplacian pyramids.

`filter_size_im` – is the size of the Gaussian filter (an odd scalar that represents a squared filter) which defining the filter used in the construction of the Laplacian pyramids of `im1` and `im2`.

`filter_size_mask` – is the size of the Gaussian filter(an odd scalar that represents a squared filter) which defining the filter used in the construction of the Gaussian pyramid of `mask`.

Note that `im1, im2` and `mask` should all have the same dimensions and that once again you can assume that image dimensions are multiples of $2^{(\texttt{max\_levels}-1)}$. Pyramid blending should now be performed as as follows:

1. Construct Laplacian pyramids L_1 and L_2 for the input images im1 and im2, respectively.
2. Construct a Gaussian pyramid G_m for the provided mask (convert it first to `np.float32`).
3. Construct the Laplacian pyramid L_{out} of the blended image for each level k by:

$$L_{out}[k] = G_m[k] \cdot L_1[k] + (1 - G_m[k]) \cdot L_2[k]$$

   where $(\cdot)$ denotes pixel-wise multiplication.
4. Reconstruct the resulting blended image from the Laplacian pyramid L_{out} (using ones for coefficients).

Comments:

- Remember to convert the `mask` to double, since fractional values should appear while constructing the `mask`'s pyramid.
- Pay attention that L_{out} should be reconstructed in each level.
- Make sure the output `im_blend` is a valid grayscale image in the range $[0, 1]$, by clipping the result to that range.

4.1 Your blending examples

You should also add two functions:

```
im1, im2, mask, im_blend = blending_example1()
im1, im2, mask, im_blend = blending_example2()
```

These functions will be performing pyramid blending on two sets of image pairs and masks you find nice. Each function should return the two images (`im1` and `im2`), the mask (`mask`) and the resulting blended image (`im_blend`). Don't forget to include these additional 6 image files (in `jpg` format) in your submission for the scripts to function properly. Each script should display (using `plt.imshow()`) the two input images, the mask, and the resulting blended image in a single figure (you can use `plt.subplot()` with 4 quadrants). The examples should present **color images** (RGB). To generate blended RGB images, perform blending on each color channel separately (on red, green and blue) and then combine the results into a single image.

Important: when you load your own images, you must use relative paths together with the this function:

```
def relpath(filename):
    return os.path.join(os.path.dirname(__file__), filename)
```

For example, if you want to load the file `test.jpg` in the subdirectory `externals` of your submission, then call:

```
im = read_image(relpath('externals/test.jpg'), 1)
```

Students that produce especially impressive and creative blended images may get up to **7 bonus points**. Note that if your initial `mask` is not binary you will not get bonus point even if the result is very nice.

4.2 Questions

What happens (and why this happens) to the result blending from section 4 image when:

Q2: Blending is performed with different image filters (`filter_size_im = 1,3,5,7...`).

Q3: Blending is performed with a varying number of pyramid levels (`max_levels = 1,2,3,4,5,...`).

5 Tips & Guidelines

- Read image files by calling `read_image` that you implemented in exercise 1 (e.g. use this in your `blending_exampleN()` functions to load color images into an RGB representation). Do not forget to include this function in your submission (ex3.zip).
- You are free to choose how to treat the image boundaries. In any case, this should not have an influence on the Gaussian pyramid reconstruction and on image blending, since all the differences should be saved in the appropriate levels of the Laplacian pyramid.
- You can assume **legal input** to all functions.
- All input images are represented by a matrix of class `np.float32`, except for the mask which is `np.bool`.
- Display figures only when this was required by the exercise definition. We may reduce points for unnecessary figures since this makes checking your exercise difficult for the grader. Each figure should be displayed in a new window. Use the `plt.figure()` command for this purpose, e.g.:

```
plt.figure()
plt.imshow(im1)
plt.figure()
plt.imshow(im2)
plt.show()
```

- It is recommended to create auxiliary functions such as for the `expand` and `reduce` operations. This will facilitate significant code reuse. Also, in your implementation of `build_laplacian_pyramid`, internally use the `build_gaussian_pyramid` function.
- Avoid unnecessary loops in your code, e.g. for pixel-wise operations such as sampling, expansion, etc. You are allowed to loop over RGB channels though.

6 Submission

Submission instructions may be found in the ”Exercise Guideline‘s” document published on the course web page. Please read and follow them carefully. You should answer the questions in sections 3.2 and 4.2 in answer files named `answer_q1.txt`, `answer_q2.txt`, and `answer_q3.txt`.

Good luck and enjoy!