Image Processing - 67829

Exercise 3: Image Pyramids & Pyramid Blending

Due date: 17/12/2014

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 week 6 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] = GaussianPyramid(im, maxLevels, filterSize)
[pyr, filter] = LaplacianPyramid(im, maxLevels, filterSize)
```

with the following input arguments:
`im` - a grayscale image (e.g. the output of ex1's `imReadAndConvert` with the `representation` set to 1).

`maxLevels` - the maximal number of levels in the resulting pyramid. `filterSize` - the size of the Gaussian filter in each dimension (squared filter) to be used in constructing the pyramid filter.

Comments:

- Both functions should output the resulting pyramid `pyr` as a *cell array* (see Matlab documentation of `cell` command) of grayscale images.
- The functions should also output `filter` which is 1D-row of size `filterSize` 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` should be normalized.
- Note that when performing both the *expand* and *reduce* operations you should convolve with this `filter` twice - once as a row vector and then as a column vector. 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{1}` has the resolution of the given input image `im`.
- The number of levels in the resulting pyramids should be the largest possible s.t. `maxLevels` isn't exceeded and the dimensions of the lowest resolution pyramid level is not smaller than 16×16 pixels. You may assume that the input image dimensions are multiples of $2^{(\texttt{maxLevels}-1)}$.

3.2 Laplacian pyramid reconstruction

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

`img = LaplacianToImage(lpyr, filter, coeffMultVec)`

Comments:

- `lpyr` is the Laplacian pyramid generated by the second function in 3.1.
- `coeffMultVec` is a vector. The vector size is the same as the number of levels in the pyramid `lpyr`. When reconstructing the image `img` you should multiply each level i by its corresponding coefficient `coeffMultVec(`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.

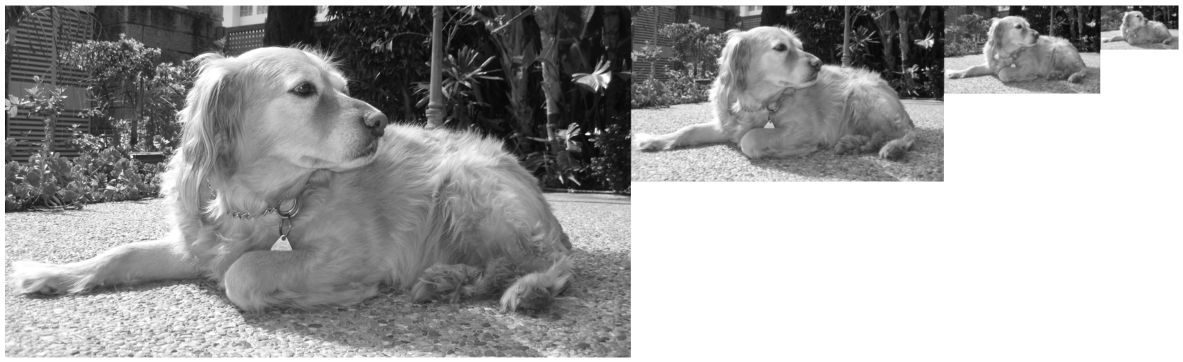
Q: What does it mean to multiply each level in a different value? What do we try to control on?

3.3 Pyramid display

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

```
res = renderPyramid(pyr,levels)
displayPyramid(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 $\leq$ `maxLevels`. `res` is a single white image in which the pyramid levels of the given pyramid `pyr` are stacked horizontally as follows:



Here a 4-level Gaussian pyramid was rendered.

Comments:

- The function `renderPyramid` should only return the big image `res`.
- The function `displayPyramid` should use `renderPyramid` to internally render and then display the stacked pyramid image using `imshow`.

4 Pyramid Blending

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

```
imBlend = pyramidBlending(im1, im2, mask, maxLevels, filterSizeIm, filterSizeMask)
```

where:

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

`mask` - is a binary mask containing 1's and 0's representing which parts of `im1` and `im2` should appear in the resulting `imBlend`.

`maxLevels` - is the `maxLevels` parameter you should use when generating the Gaussian and Laplacian pyramids.

`filterSizeIm` - is the size of the Gaussian filter in each dimension which defining the filter used in the construction of the Laplacian pyramids of `im1` and `im2`.

`filterSizeMask` - is the size of the Gaussian filter in each dimension 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 $2^{(\texttt{maxLevels}-1)}$. Pyramid blending should now be performed as as follows:

1. Construct Laplacian pyramids L_1, L_2 for the input images `im1` and `im2`.
2. Construct a Gaussian pyramid G_m for the provided `mask`.
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}.

Comments:

- Be careful and to use a double matrix for `mask`, since fractional values should appear while constructing the mask's pyramid.
- Pay attention that L_{out} should be reconstructed in each level.

4.1 Your blending examples

You should also submit two scripts called `blendingExample1.m` and `blendingExample2.m`, performing pyramid blending on two sets of image pairs and masks you find nice. 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 `imshow`) the two input images, the mask, and the resulting blended image (therefore each script should open 4 figures). 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.
Students that produce especially impressive and creative blended images may get up to **7 bonus points**.

4.2 Questions

Explain in your `README` file what happens (and why this happens) to the result blending from section 4 image when:

1. Blending is performed with different image filters (`filterSizeIm` = 1,3,5,7...).
2. Blending is performed with a varying number of pyramid levels (`maxLevels` = 1,2,3,4,5,...).

Answer also to the question from section 3.2

5 Tips & Guidelines

- Read image files by calling `imReadAndConvert` that you implemented in exercise 1 (e.g. use this in your `blendingExampleN.m` scripts 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 double.
- 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 `figure` command for this purpose (e.g. `figure, imshow(im1); figure, imshow(im2);` etc.).

- 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 `LaplacianPyramid`, internally use the `GaussianPyramid` 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.

Good luck and enjoy!