Computational Photography
Project 1, Fall 2014

18.09.2014

The goal of this project is to get familiar with Matlab and obtain some first experience in processing color images. The project needs to be turned in by uploading on ILIAS. Submit material as described in the *turn-in instructions* for each part. The deadline is October 2, 10:00.

Your turned in Matlab code needs to be "ready to run". Therefore you should include the input images for your code, and set the path information in your scripts correctly. Further, create a "start file", which is organized in Matlab cells. Each cell contains the call to the functions written for that part of the project.

1. Spanish Castle Illusion (2 points)

In this assignment you will produce a "Spanish Castle" illusion using Matlab. This is an optical illusion involving color perception. Run the first cell of the provided `_start.m`-file to see an example.

Your goal is to write a Matlab function `function [grey inverted] = spanishCastle(img)` that generates a grey and an inverted image to produce the illusion. This function should work as follows:

- `img` represents the input color image as an $m \times n \times 3$ matrix. We assume RGB colors and the range of each color channel is in $[0, 1]$. You can read image files using `imread`, which returns an $m \times n \times 3$ matrix. You will need to convert this matrix to double and scale it to the interval $[0, 1]$ using `im2double`.

- Convert the RGB colors to YUV color space. This is implemented by interpreting the RGB colors of each pixel as a column vector. You multiply the RGB vector by a specific transformation matrix to obtain the corresponding YUV vector. You can find the transformation matrix here `http://en.wikipedia.org/wiki/YUV`. Your implementation should avoid using a for loop over all image pixels. Instead, reorganize the input image such that the conversion can be done with a single matrix product that converts the whole image at once! Hint: Use the Matlab `shiftdim` and `reshape` functions.

- Use the input Y channel to form the `grey` image.

- To obtain the `inverted` image, set the Y channel to 0.6 for all pixels, convert back to RGB, and invert the RGB colors by taking one minus the RGB values. The result is the `inverted` image.

Save the images to file using `imwrite` and test the illusion by flipping between the images as in the demo. The simplest way to flip between the images is to load them both in an image viewer and flip between them using alt-tab on Windows or similar on other systems. Try to find an explanation how the human visual system produces this illusion.

Turn-in: Matlab code, a pdf file containing your own example image and explanation of how the illusion works.

2. Processing RAW Images

Many digital cameras provide the functionality to store the raw sensor data in so-called *RAW* files, which makes it possible to perform all processing using off-camera programs and algorithms. Raw sensor data can be interpreted as *digital negatives* that require many processing steps to produce appealing final images. This is somewhat analogous to developing negatives in film photography. The process can be quite complicated and include many different steps. For an overview, see `http://en.wikipedia.org/wiki/Raw_image_format`. In this assignment you will implement some basic operations to obtain a reasonable image from raw data.

2.1. Bayer Demosaicing using Bilinear Filtering (2 points)

In this assignment you write a Matlab function `function out = demosaicBayer(img)` for Bayer demosaicing using bilinear interpolation. The input `img` to the function is a $m \times n$ 2D matrix that represents a color image using a Bayer pattern. The output is a $m \times n \times 3$ matrix that represents a color image with per pixel RGB values. The conversion process uses bilinear interpolation of the 4 or 2 nearest neighbors. The goal is to implement this without any for loops. This can be achieved as follows:

- Generate three $m \times n$ matrices that represent masks for red, green, and blue pixels. Each mask contain ones at pixels of the corresponding color, and zeros otherwise.
- Extract the red, green, and blue values from the input by element-wise multiplication (using the Matlab `.*` operator) with the masks. This results in three matrices containing only values of the corresponding colors, and zeros otherwise.
- Interpolate the red, green, and blue matrices by convolving each with a 3×3 matrix consisting of ones. Use the Matlab function `conv2` with the parameter `'same'` for the convolution. This operation replaces each value with the sum of the input values in a 3×3 neighborhood. Therefore, it “fills the holes” in the input.
- Normalize the output of the convolution. This is necessary because the number of nonzero input values is not equal in all 3×3 neighborhoods. You can do this by first convolving the grid matrices containing only zeros and ones with the 3×3 convolution matrix. This computes the number of nonzero elements in each neighborhood. Now perform an element-wise division (using the `./` operator) of the outputs of the previous step with the corresponding normalization matrices.

- Assemble the interpolated data in a $m \times n \times 3$ matrix. Note that you want to use the original data where available, and fill in interpolated pixels only where necessary. You can do this again using the masks for each color channel.

Use the `foliage raw.tiff` image we provide. This image contains raw sensor data from a Canon EOS 400D camera. The values have 12 bit precision and are in the range of zero to 4096. Hence you need to scale them to $[0, 1]$ for display in Matlab. Ask us for advice if you want to use your own RAW images. Note that the output colors will not look perfect because they are not standard RGB colors, and there is no gamma correction.

Turn-in: Matlab code, a pdf file containing one or several examples. Show the input as a grey scale image and the corresponding color output.

2.2. Median Filter Demosaicing (2 points)

Implement median filter demosaicing as discussed in class. Median filtering is supported in Matlab by the `medfilt2` function. As before, try to write Matlab code that avoids for loops! Experiment with the `black and white raw.tif` that we provide. Note that this image file has 8 bit precision, so the maximum value per color channel is 255. Compare linear filtering to the technique using the median filter.

Turn-in: Matlab code. A pdf file containing a comparison between demosaicing with linear and median interpolation. The comparison should show an example with color fringing in the linear case, and its absence using median filtering. Use the `black and white raw.tif` image, or optionally your own images.

2.3 Color Balancing (2 points)

Color balance is the global adjustment of the relative intensities of the color channels to render specific colors such that they appear correctly in the output. Since the goal is usually to make sure that neutral colors (grey scales) appear grey, it is also called white balance. Color balancing is necessary because of color constancy, i.e., the fact that human color perception ensures that the perceived color of objects remains relatively constant under varying illumination conditions. A white object under red illumination is captured as a red object by a camera, but is perceived as white to a human observer because of color constancy. White balancing removes the red tint from the captured image to reproduce the white object as it is perceived, which is white instead of red.

Write Matlab functions that implement the following white balancing operations:

- *Grey World Assumption:* This technique assumes that the average color in any image is grey. It transforms the input image to match this assumption. This is achieved by first computing the average color in the image. Assume that the scaling factor for one color, say green, is one. You compute scaling factors for red and blue such that after scaling, the average red and blue will have the same value as the average green, i.e., the average color is a grey level. This means the scaling factor for, e.g., red is *average green of input/average red of input.*

- *Manual White Balance:* Choose a pixel in the image manually that should be white. Compute scaling factors based on this pixel's color.

Compare the two methods on several images of your own choice. Describe cases where each of them would not work well.

Turn-in: Matlab code. A pdf file containing comparisons using several images of your own choice and text describing your observations and explanation.

2.4 Global Contrast Adjustment (2 points)

Contrast adjustment is often performed to increase detail and make images look more vivid. Here you implement linear contrast scaling and gamma transformation.

Linear contrast scaling. Implement a Matlab function `out = linearContrast(img, toMin, toMax)` that changes contrast by linearly scaling image brightness. The function should proceed as follows:

- Assume that the input is an RGB image with color channels in the range $[0, 1]$.
- Transform the image to YUV color space.
- Scale and shift the Y channel such that `toMin` maps to zero and `toMax` maps to one.
- Transform back to RGB colors.

Gamma transformation. Implement a Matlab function `out = gammaTransformation(img, gamma)` that changes contrast by applying a gamma transformation. Transform color images to YUV and operate on the Y channel only as above.

Demonstrate your implementation using several examples and plot (using the Matlab `plot` function) the corresponding transformation curves.

Turn-in: Matlab code. A pdf file containing several images of your own choice and the corresponding curves.

3. Bonus (up to 2 points)

You can earn a bonus point for each of the following two voluntary bonus assignements. If you solve both correctly, you will get two bonus points.

- Write a GUI for one (or more) of the previous assignments. For example for assignment 2.3, allow the user to click on the image to select the pixel which should be white. For assignment 2.4, you could allow the user to change the parameters interactively using sliders. You could also visualize the transformation curves, similar as in Photoshop.
- As mentioned at the end of assignment 2.1, filtering RAW files is not enough to get good looking RGB images. To obtain more convincing results, additionally perform color balancing, color transformation, and gamma correction after the demosaicing step. In assignment 1, 2.2, and 2.3 you already wrote code that can perform these

steps. The color transformation is basically a transformation from the camera's color space to the sRGB color space. To obtain good results, find correct parameters for your camera model from the open source application DCraw. You will need to do some reverse engineering of the code to identify the right parameters.

Turn-in: Matlab code. A pdf file containing a screen shot of the GUI and a short description how to use it respectively containing the used transformation matrix and the parameters of the color balancing and the gamma correction.