DTU Electro
Department of Electrical and Photonics Engineering

34269 Exercise on Introduction to digital image processing
(Part 1)

Fundamental manipulations of digital images *(see useful functions p3)*

1. **Visualization** on *goldhill.png* and *PVPanel_electroluminescence.tif*: Read and display both images (for the PV image, define min and max for visualization). Display one of the images as a wireframe mesh. Display the most significant bitplane of one image, then the two most significant bitplanes, …

2. **Intensity transformation** on *goldhill.png*
   a. Write a function that stretches the values of an image between a low and high threshold. Visualize the result for various threshold values (e.g. try the 0.1 and 0.9 percentiles) with the histograms of original and modified images.
   b. Isolate part of the image (e.g. the white walls in the goldhill image) by thresholding the original image via conditional indexing. Is intensity transformation good enough for segmenting part of an image?

3. **Problem solving** on operaen.jpg and skuespilhuset.png: you missed the sunset while visiting Wonderful Copenhagen. Transform the images operaen.jpg and skuespilhuset.png to make them look like they were acquired at sunset. Write your own function to match the histogram of pictures of cities at sunset for that (NYCsunset and PamplonaSunset) on each color channel independently.

HDR reconstruction by multi-exposure fusion *(see useful functions p3)*

Aim: Using the 16 Memorial images with various exposure times, find the response curve of the camera using Debevec's method [1] and reconstruct the radiance map (HDR image).

Principle: A pixel response *Zij* (i is pixel index and j exposure)depends only on irradiance *Ei* and exposure time *tj* via a non linear function *f* *Zij = f(Ei x tj)*

By defining $g = \ln(f^{-1})$, we can write $g(Zij) = \ln(Ei) + \ln(tj)$

The aim becomes to estimate g, which is done by minimizing the following cost function:

$$\mathcal{O} = \sum_{i=1}^{N} \sum_{j=1}^{P} [g(Z_{ij}) - \ln E_i - \ln \Delta t_j]^2 + \lambda \sum_{z=Z_{min}+1}^{Z_{max}-1} g''(z)^2 \quad (1)\ \text{(Eq. (3) from [1])}$$

Where the term with the second derivative is used to force the response to be smooth (the "smoothness" is controlled by λ).

The reconstruction of the radiance map can be done using all exposures:

$$\ln(E_i) = \frac{1}{P} \sum_{j=1}^{P} \left(g(Z_{ij}) - \ln(t_j) \right) \quad (2)\ \text{(based on Eq.6 from [1])}$$

Process (for each color channel separately):

- ***Response curve recovery:*** Select a random subset of the pixels of the image (>100). Using the gsolve function to solve the equation system from (1) (use λ=3 as starting point), find the response curve g and plot it for each color channel
- ***HDR reconstruction***: Reconstruct the HDR radiance map lE by using (2). A loop over all exposures is needed, but to avoid looping through all pixels within the image, at each exposure you can go through the 256 intensity values and using conditional indexing to modify all the pixels having that precise value.

Reference [1] Paul E. Debevec and Jitendra Malik. Recovering High Dynamic Range Radiance Maps from Photographs, in *SIGGRAPH 97*

Useful functions for Part 1 and Part 2

Purpose	Matlab	Python

Load an image	`imread`	`import imageio.v3 as iio`
		`img=iio.imread('imagepath')`
		For greyscale images check the dimension, if iio reads a greyscale image with 3 channels, extract the first channel in a new variable and work with it

Display an image	`imshow` (use parameter [low high] for range)	`import matplotlib.pyplot as plt` Create figure with `fig, ax = plt.subplots()` Display image with `ax.imshow(img)`
Display settings	Axes og color coding are defined via `axis` og `colormap`.	Via `imshow` parameters, `e.g. cmap='gray', vmin=0, vmax=255` Through functions setting values for axes object `(e.g. ax.set_xlim())`
Create 2D grid	`meshgrid` Plot with `mesh`	`np.meshgrid` Plot with `ax.plot_surface()`
Separate bitplanes of an image	`dec2bin, str2num/bin2dec` and `reshape`	`np.unpackbits` with adding an empty axis on your image (with `img[:, :, np.newaxis]`) and using this as dimension to unpack (passing it as argument `axis=`) or use quantization (= division/rounding/multiplication)
Reshape an array	`Reshape`	`np.reshape` (in NumPy)

Compute the frequency of occurrence of pixel values	`Histcounts`
	(remember to specify the bin edges or the number of bins)
		`Without plot np.histogram`
		`With plot ax.hist or`
		`matplotlib.pyplot.hist`
		In both cases, remember to define bins, e.g. by using `np.arange()`

Plot histograms	`Bar`	`matplotlib.pyplot.hist` `matplotlib.pyplot.bar`
Perform histogram equalization	`histeq`	skimage.exposure.equalize_hist in Scikit-image or cv2.equalizeHist in OpenCV
Slicing/striding arrays	link	Numpy link
Conditional indexing for vector/matrix	link	NumPy link
Random numbers	randperm	`np.random.default_rng`, `np.arange` and `np.random.shuffle`
Use SVD to solve (1)	gsolve.m (in zip file of the exercise	gsolve.py (in zip file of the exercise, it is a simplified version of Debevec97)