EECS6323: Advance Topics in Computer Vision [Assignment3 – Computational Optics]

Assigned: ***March 15th, 2017 (Wednesday)***

Due: ***March 27st, 2013 (Friday 11.59pm, via email)***

Goal of this assignment

1. To implement two (2) "computational illumination" techniques
2. To give you an opportunity to learn vision related techniques, including log-mapping, bi-lateral filtering, edge extraction

IMPLEMENT ANY TWO (2) FROM THREE.

1. Tone Mapping

Write routines to implement the following three (3) tone-mapping approaches:

1) Linear Mapping
   a. Simply scale all radiance values, where min world radiance→0, and max world radiance→255, all values in between are linearly scaled.
      - Use the global min/max (across all 3 RGB channel).

2) Erik Reinhard's simple tone mapping approach based on the log-average luminance and a user supplied key 'a'. This technique is from eq (1) & eq (2) from Reinhard's paper (see link from Readings).

3) Implement a "Local Dodge and Burn" technique. You can implement this anyway you want, but Section 3.2 in Reinhard's paper is a reasonable starting point although a bit complicated. You can implement this exactly like the Tone Reproduction paper or try your own ideas. Essentially you should apply some heuristic to determine a local region to use for local image scaling based on the image content.

Input Images: AtriumNight.pfm, church.pfm, doll_doll.pfm, lips.pfm, rosette.pfm

Output:

inputname_linear.bmp	// Simple linear map.
inputname_key_X.bmp	// X is the user key value; pick the key value you think is best.
inputname_local.bmp	// Your local dodge-burn operator.

For each input image, generate the above three images.

This task should have a total of 15 images output images.

Several images are provided for you to try. All images in portable float map format. This format encodes world radiance (or what Reinhard et al call world luminance $L_w(x,y)$). Maltab `imread` function cannot read this format. Code to read this is provided (getpfmraw.m); try: `I=getpfmraw('filename.pfm')` in matlab. This function returns back an RGB image, size: `Height×Width×3`. By the way, you won't be able to view these as images without some sort of tone-mapping.

2. Low-Noise Ambient Lighting Imaging (Flash/No-Flash Photography)

Implement *Petschnigg et al* SIG'04 "Flash/No-Flash" technique to recover ambient light (i.e. low light) photographs with reduced noise. The main idea in this paper is the use of the joint-bilateral filter, which uses the Flash image as the range weight to filter the No-Flash image.

You should be able to implement this in Matlab or C++, but just be aware that the joint bilateral filtering may be very slow, especially for a large size image. There are several components to this task, the Mask computation (to handle shadows and specularities) may not be easy to compute, but do the best you can. I'm more interested in the core algorithm versus the Masking technique.

There are three pairs of images made available. You should generate the following additional images:

ImageXBase	// Bilateral filter on the ImageX (no flash)
ImageXNR	// Joint-bilateral filter on ImageX (using both inputs)
ImageXFdetail	// Bilateral filter details on ImageX (flash)
ImageXMask	// Shadow and specular mask
ImageXOutput	// The final output

You should have a total of at least 15 images (3 examples x 5 outputs per example), but you may submit even more if you try other examples, which I highly encourage you to try.

You can also try to apply this same procedure, but use the darkflash images from Krishnan's paper. I included a few in the directory. You can find more images here:
http://cs.nyu.edu/~fergus/research/dark_flash.html

3. Multi-Flash Camera

Implement the basic idea in Raskar et al NPR Camera paper from SIG'04. You do not need to implement the Poisson Image Integration technique, but instead focus on extracting the occluding boundary edge maps. *You must also make some simple NPR images, e.g. overlaying your detected edges onto the input image (or MAX image).*

Please try two data-sets:
(1) From Raskar's paper – provided.

(2) Capture your own data, using a spot light source (i.e. a lamp) and move the lamp around. Be sure to keep the camera and object you are capturing very still and only move the light. This will be a reasonable approximation to Raskar's hardware. Note that Raskar gives sample Matlab code in his paper.

SUBMITTING YOUR ASSIGNMENT
Programming Language
You may use any programming language you want. Task 1, 2 and 3 can be done in Matlab, but you are free to write you own code.

What to turn in?
1.) README File

Turn in a short writeup on which tasks you implemented. For each task, provide a short description of your implementation followed by several results. The results can be screen captures in the case of Task 1 (Image Snapping). For each problem I've included some test files, but please try more than these too.

2.) Code and Images

Include your code and output images.

Make the following directory tree structure:

~\assignment3\Readme.{doc,tex,pdf}
~\assignment3\t1\yourfiles
~\assignment3\t3\yourfiles

Your {t1} and {t3} will depend on the tasks you performed. I should be able to run your code. If it is not obvious how to run your code, then describe what I need to do in your README File.

Please double-check your zip file to make sure everything is in order.

Example, if your name is Zhang Bo, then name your submission will be:
zhangbo_assignmen.zip.

How to turn in?

Email me a link to your assignment. *Email your assignment by the due date. Late submission will have points deducted.*