Exercises for Advanced Graphics
Lectures 5–7 (2nd part)

Rafał Mantiuk

Lent term 2016/17

1 Models of early stages of visual perception

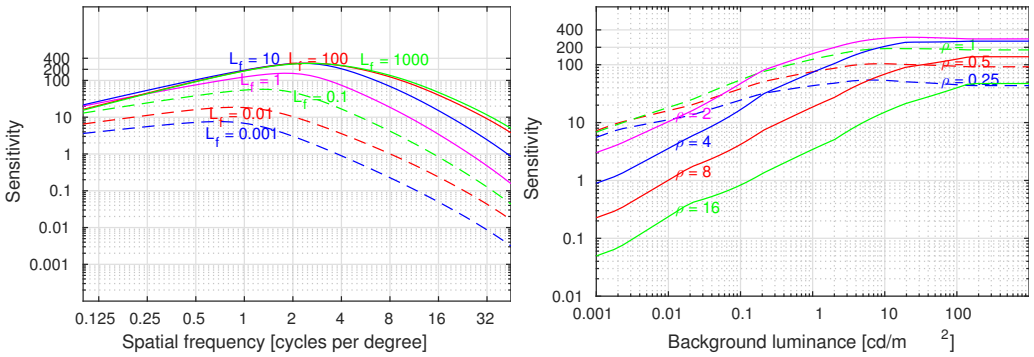

Figure 1: The contrast sensitivity function (CSF), as a function of spatial frequency (left) and background luminance (right).

1. The angular resolution of an image seen on a screen is 32 pixels per visual degree. If you want to draw in that image a sinusoid of the frequency 8 cycles per visual degree, what should be its frequency in cycles per pixel?

2. Given the CSF function in Figure 1, read the data from the plot and answer the questions:

   a) What spatial frequency is the easiest to detect at a uniform field of $0.01\,\mathrm{cd/m^2}$?

b) What is the smallest detectable difference in luminance (in $\mathrm{cd/m^2}$) for 8 cycles per visual degree pattern shown at the background of $1\,\mathrm{cd/m^2}$?

c) What is the highest spatial frequency of a just noticeable sinusoidal pattern of a relative contrast ($\Delta L/L$) of 0.01 (1%) at $100\,\mathrm{cd/m^2}$?

The values read from the plot do not need to be exact.

2 High dynamic range and tone-mapping

The following set of exercises need to be completed either in Matlab or in GNU Octave. To make your work easier, you should use a set of helper functions, which you can download from a github repository `https://github.com/mantiuk/advanced_graphics`, and you can find in the directory `tmo_utils`. The directory also contains an example HDR image "memorial", stored both in Radiance *.hdr* format and as Octave *.mat* file. Use `hdrread` function in Matlab and `load` function in Octave to load the image. Use `imview` function to view (LDR) images.

1. Convert an HDR image to the sRGB color space and display it. The image is likely to come out too bright or too dark. What operation do you need to perform before applying the sRGB non-linearity to get a well exposed image?

   You can use `lin2srgb` function for the color space conversion.

2. Compute the dynamic range of a display used

   a) in a dark room with ambient light illumination $E_{amb} = 0\,\mathrm{lux}$.

   b) in sunlight, under ambient light $E_{amb} = 10\,000\,\mathrm{lux}$

   The parameters of the display are as follows: $\gamma = 2.2$, $L_{peak} = 500\,\mathrm{cd/m^2}$, $L_{black} = 0.5\,\mathrm{cd/m^2}$. The screen reflects back 1% of the light. You can use `gog_fw_display_model` function for this task.

   If you want to improve legibility of a display in sunlight, which display parameters do you need to alter?

3. Transform an HDR image to luminance (use `get_luminance` function), then compress luminance contrast using a power function. Finally, restore colors using the formula:

$$C_{out} = \left(\frac{C_{in}}{L_{in}}\right)^{s} \cdot L_{out}, \tag{1}$$

where C_{in} is the input colour channel (red, green or blue), L_{in} is the input luminance and C_{out} is the resulting colour value. Note that the resulting RGB values are in linear space and still need to be convered to sRGB before they can be displayed. Experiment with different values of s.

4. Write code for a reflectance-illumination decomposition tone-mapping that

   a) uses Gaussian filter

   b) uses Bilateral filter (function `bilateral_fast`)

   Both filters need to operate on an image in the logarithmic domain. You can perform all processing on a gray-scale (luminance only) image, or perform the same operation on each color channel. Employ a simple contrast compression for the tone mapping of the base layer:

$$\log(L'_{base}) = c\left(\log(L_{base}) - \log(L_{white})\right) + \log(L_{white})\,, \tag{2}$$

   where L_{base} is the base (illumination) layer before tone-mapping, L_{white} is the luminance of white, and L'_{base} is the layer after tone-mapping.

3 GPU programming in OpenCL

You are encouraged to experiment with writing a simple OpenCL code. Can you think of a (simple) algoritm that could be efficiently exdcuted on a GPU. To help get you started, we provided a few examples in the github repository `https://github.com/mantiuk/opencl_examples`. All examples are in C++11 and can be compiled on Windows, Linux and Mac using cmake (`https://cmake.org/`).

Note that compiling C++ code with external libraries requires some experience. If you struggle to compile the examples, just spend some time

going through the code. You are not expected to become an expert OpenCL programmer after this course, however, you should know the main concepts and the programming model.