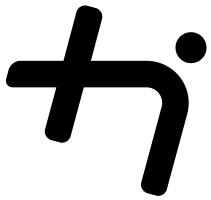
Technische Hochschule Ingolstadt
Prof. Dr.-Ing. Richard Membarth



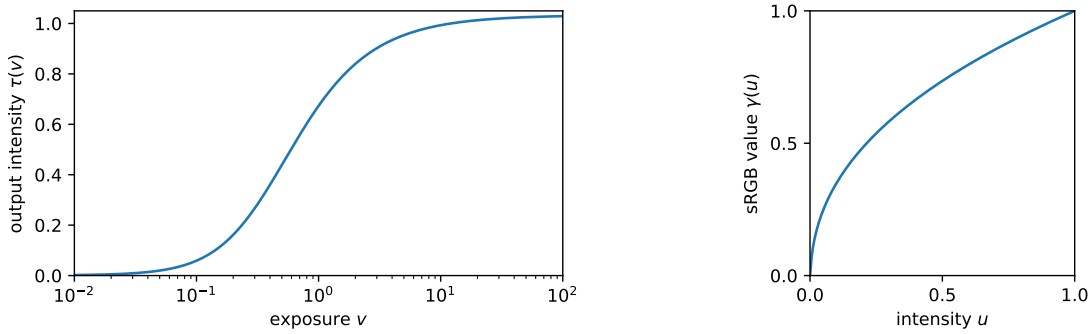
26.04.2023

GPU Programming
Assignment 4

Submission deadline for the exercises: 15.05.2023

HDR Tone Mapping

The goal of this exercise is to implement a simple transformation that takes high-dynamic-range (HDR) image data as input, maps the color of each input pixel to a corresponding low-dynamic-range (LDR) color according to a given tone mapping function, and then writes out the resulting LDR image encoded as 8-bit sRGB data suitable for display on a regular display device.
The term *dynamic range* describes the range of luminance in a scene, or the range of luminance values that an imaging system can capture, an image processing system can process, or a display system can reproduce. Luminance in real-world scenes can span a vast range of values with the ratio between the brightest and darkest points often exceeding ten orders of magnitude. Human eyes as well as film and modern digital cameras are capable of capturing images of such scenes in a way that preserves a significant part of the dynamic range in the



(a) the tone mapping operator

(b) sRGB gamma function

Figure 1: (a) The tone mapping function $\tau(v)$ used to map high-dynamic range input color values to low-dynamic-range output intensities for display. (b) The sRGB gamma function $\gamma(u)$ used to encode linear output intensities into sRGB color values. Note how the steeper slope in the area of lower intensities results in a larger part of the output domain being spent on encoding values from this range of inputs.

resulting image information. Light transport simulation allows for photorealistic imagery to be generated computationally.

While it is possible nowadays to capture, generate, and process HDR image data representative of the real world, commodity display hardware is still limited in the range of luminance that can be reproduced. Thus, to actually display such an HDR image requires that the HDR image information be reduced to a lower dynamic range, ideally in a way that preserves as much detail as possible without altering the overall appearance of the image too much. This process of mapping HDR color data to LDR color for display is called *tone mapping*.

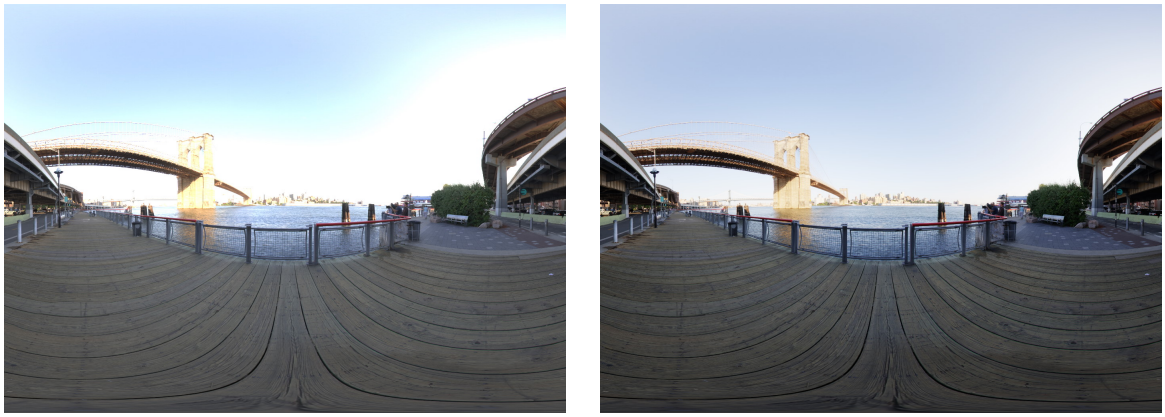
In general, a tone mapping function maps an exposure value in the range of $[0, \infty)$ to an output intensity in the range $[0, 1]$ where 0 represents the lowest and 1 the highest luminance the target display device can reproduce. For the purpose of this exercise, we will be using the function

$$\tau(v) = \frac{v \cdot (0.9036 \cdot v + 0.018)}{v \cdot (0.8748 \cdot v + 0.354) + 0.14} \tag{1}$$

as our tone mapping operator where v denotes the input exposure value and $\tau(v)$ yields the corresponding output intensity[1]. As can be seen in Figure 1a, this mapping works by "compressing" the parts of the input domain that are increasingly far away from an exposure value of 1 into increasingly smaller parts of the output domain. Such an S-shaped mapping function is designed to mimic the response curve of film. Figure 2 shows the effect of tone mapping.

To actually display the LDR output image, the output intensities obtained via the tone mapping function have to be encoded into some color representation that is understood by the target display system. The most-common encoding in use today is the sRGB color space. To transform our desired output intensity into its corresponding sRGB value, we apply the

[1]This function is a rational function approximation derived by Krzysztof Narkowicz for an Academy Color Encoding System (ACES) tone mapping operator.



(a) no tone mapping (b) tone mapping

Figure 2: Comparison to illustrate the effect of tone mapping. (a) No tone mapping. Color values beyond the $[0, 1]$ range are simply clipped. (b) Tone mapping. Note how detail is visible even in extremely bright parts of the image such as the sky and skyline in the background.

sRGB gamma function

$$\gamma(u) = \begin{cases} 12.92 \cdot u & u \leq 0.0031308 \\ 1.055 \cdot u^{\frac{1}{2.4}} - 0.055 & \text{otherwise.} \end{cases} \tag{2}$$

The purpose of this non-linear distortion is to allocate the limited number of bits available to encode color values in a way that accounts for the fact that the human eye is more sensitive to differences between darker colors than between bright colors (see Figure 1b).
Putting it all together, given an HDR input color value c and exposure e, we can compute the LDR output color value as

$$c' = \gamma(\tau(c \cdot e)). \tag{3}$$

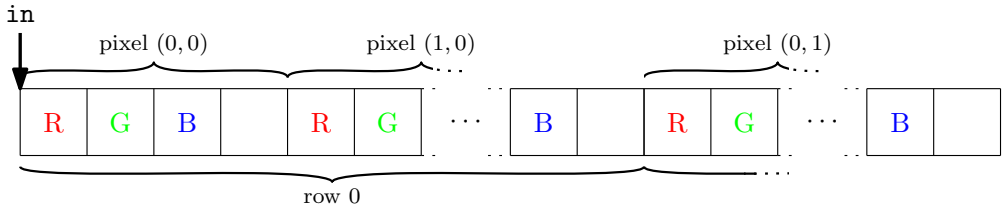
As a last step, the output color c' has to be encoded into an 8-bit representation by scaling such that values in the range $[0, 1]$ are mapped to values in the range $[0, 255]$ and then rounding to the nearest integer.

HDR Pipeline Task 1: Tone Mapping

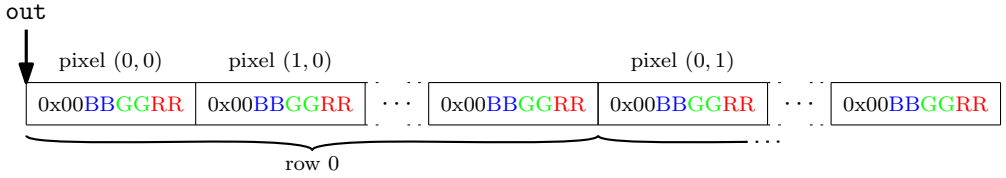
In «`src/hdr_pipeline.cu`», implement the function

```
void tonemap(uint32_t* out,
             const float* in,
             int width, int height,
             float exposure,
             float brightpass_threshold)
```

such that it transforms the HDR image data pointed to by `in` into an LDR image following the steps described above. `in` points to a block of device memory that contains `height` rows of `width` input pixels, each pixel consisting of four values of type `float` holding the red, green, and blue color values of the pixel in that order. The last `float` serves as padding and is to be skipped. `out` points to a block of device memory into which the 8-bit sRGB output image



(a) input image data layout



(b) output image data layout

Figure 3: Memory layout of the input and output image data. (a) The HDR input image data is stored as a sequence of objects of type `float` where each four consecutive `float`s hold the red, green, and blue color values of a pixel respectively. The last `float` in each pixel serves as padding and is to be skipped. Image data is stored in a row-wise fashion top down and left to right, that is, starting with the pixels of the top row in left-to-right order followed by the pixels of the second-to-top row, and so on. (b) The LDR output image data is stored as a sequence of 32-bit unsigned integers also in top down and left to right order. Each 32-bit unsigned integer corresponds to one pixel with the lowest 8-bits holding the red, the next-lowest 8-bits holding the green, and the next 8-bits after that holding the blue value.

is to be written. It consists of `height` rows of `width` output pixels, each output pixel holding a 32-bit unsigned integer value where bits 0–7 correspond to the red, bits 8–15 to the green, and bits 16–23 to the blue channel. See Figure 3 for an illustration of the input and output image data layouts. `exposure` corresponds to the exposure parameter e in Equation (3). The parameter `brightpass_threshold` will only be relevant for a later assignment and can be ignored for now.

Sources Extraction: First, download the sources («`hdr-pipeline-task1.tar.gz`»), assets/images («`assets.tar.gz`»), and reference images («`assets-ref-task1.tar.gz`») for the HDR pipeline project. Create a directory for the HDR pipeline project and extract the files there;

```
mkdir hdr-pipeline
cd hdr-pipeline
tar xvfz hdr-pipeline-task1.tar.gz
tar xvfz assets.tar.gz
tar xvfz assets-ref-task1.tar.gz
```

Building: We use CMake to build the project. For this, we create a separate «`build`» directory and configure the project for `Release` mode:

```
mkdir build
```

```
cd build
cmake ../hdr-pipeline-task1 -DCMAKE_BUILD_TYPE=Release
make
```

Running: After building the project, the «`hdr_pipeline`» binary will be present in the «`bin`» directory. The binary expects an input image in `hdr` format and will apply the tone mapping and write the output image as `png` file.

```
./bin/hdr_pipeline ../assets/LA_Downtown_Helipad_GoldenHour_3k.hdr
eog LA_Downtown_Helipad_GoldenHour_3k.png
```