PMPP WiSe 2025 - Exercise 1

Johannes S. Mueller-Roemer and Sebastian Besler

TECHNISCHE UNIVERSITÄT DARMSTADT

WiSe 2025 – v1.00 (October 27, 2025)
Sheet 1

1. Grading

In order to get the bonus of $\frac{1}{3}$, you will need to achieve 75% of the total points of both exercises. In total there are 16 points that can be achieved, exercise 1 and 2 give up to 8 each. The respective points of each task are given in the headline.

2. Working with the Cluster

- **Download:** Acquire the framework from moodle
- **Login:** Use your TU-ID to gain access to the cluster. The compute nodes can't be accessed directly. Use one of the login nodes[1]
  Our compute node runs RedHat 8.10, which means you will need to compile the code on one of the RedHat 8.10 login nodes: `lcluster7` or `lcluster19`
  me@home: `ssh <TU-ID>@lcluster<X>.hrz.tu-darmstadt.de`
- **Upload the framework (from your own machine):**
  me@home: `scp pmpp_ex<num>.tar.gz \`
  `<TU-ID>@lcluster<X>.hrz.tu-darmstadt.de:~/pmpp_ex<num>.tar.gz`
- **Load cuda:**
  me@cluster: `module load cuda/12.5 gcc/13.1.0`
- **Extract framework:**
  me@cluster: `tar -xzvf pmpp_ex<num>.tar.gz` and `cd pmpp_ex<num>`
- **Setup build dir:**
  me@cluster: `mkdir build`
  me@cluster: `cd build`
- **Configure build:**
  me@cluster: `cmake -DCMAKE_BUILD_TYPE=<Debug or Release> ..`
- **Build:**
  me@cluster: `make`
- **Run:**
  me@cluster: `sbatch run.sh`
- **List jobs** You may list your currently enqueued jobs using:
  me@cluster: `squeue`
- **Cancel job** You may cancel your currently enqueued jobs using:
  me@cluster: `scancel <job_id>`

[1] List of login nodes: https://www.hrz.tu-darmstadt.de/hlr/betrieb_hlr/hardware_hlr_1/aktuell_verfuegbar_hlr/index.en.jsp

Exercise1
LastName, FirstName: _____ EnrollmentID: ␣␣␣␣␣␣␣␣

3. Submission

Submit your solution via moodle. Please provide a zipped tar archive of the code (**without** the `out` or `build` directories).

- **Create archive:**
  me@cluster: `tar -czvf pmpp_ex<num>.tar.gz pmpp_ex<num>`
- **Upload:** Upload your solution to moodle

4. Implementation

Please do not change the paths of the images or move the code within the directory structure. Grading will be performed automatically, if your code doesn't compile or the images can't be found, the task will be graded with 0 points.

Task 1.1: Memory management (2 Points)

The framework includes the RAII wrappers shown in the lecture in `src/pointer.h`.

1.1a) Allocation

Implement the creation of empty matrices in the functions

- `make_cpu_matrix` in `src/image.cpp`
- `make_gpu_matrix` in `src/image.cpp`

Note: Use the wrappers to allocate memory

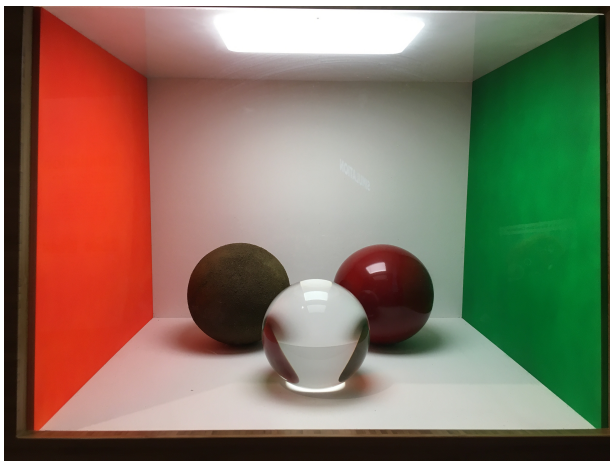
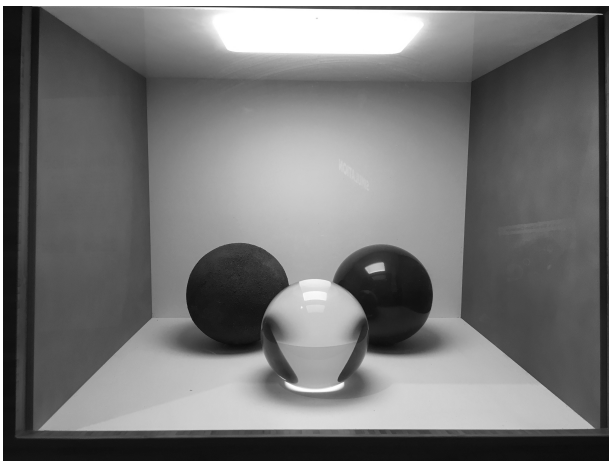
1.1b) Transfers

Implement copying matrix data from/to the device in the functions

- `to_gpu` in `src/image.cpp`
- `to_cpu` in `src/image.cpp`

Exercise1
LastName, FirstName: _____ EnrollmentID: ␣␣␣␣␣␣␣␣

Task 1.2: Color image to grayscale image (2 Points)

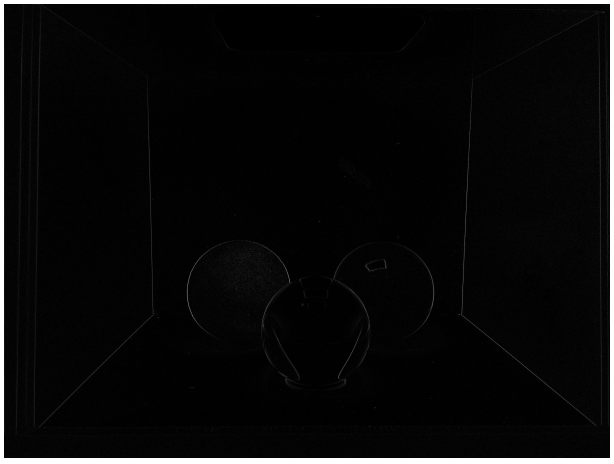



Write a kernel to convert color image data (given as RGB values) to a grayscale image (also using RGB values) and apply it to the example image[2]. Compute the grayscale image using the luminance of each pixel $Y = 0.2126 * r + 0.7152 * g + 0.0722 * b$
Implement the conversion kernel `gray_scale_kernel` in `src/filtering.cu`.
Note: Converting a single pixel per thread is sufficient
Note: Extract the color channels from the tuple using bit masks and shifts:

```
unsigned char r = pixel & 0xff;
unsigned char g = (pixel >> 8) & 0xff;
unsigned char b = (pixel >> 16) & 0xff;
```

Note: Remember to set the values for all color channels

Task 1.3: Convolution (2 Points)



Common image filtering operations, such as Gaussian Blur, apply a matrix of weights to a subset of pixels to compute the new value of a pixel. These are called *convolutions*, the matrix of weights is called *filter kernel*. Implement GPU-parallel filtering of images given a filter kernel in `convolution_kernel` in `src/filtering.cu`.
Note: The filter kernel is applied to each pixel and its neighborhood to compute the value of the new pixel
Note: The filter kernel is applied to each color channel individually

[2]Original image by info2learn, https://en.wikipedia.org/wiki/File:Cornell_Box_with_3_balls_of_different_materials.jpg, licensed under the Creative Commons Attribution 3.0 Unported license: https://creativecommons.org/licenses/by/3.0/deed.en, Converted to Netpbm

Note: The filter kernel may produce negative values and values above/below ± 255. Use $\frac{|value|}{2}$ if `use_abs_values` is `true`

Task 1.4: Histogram (2 Points)

A histogram of an image shows the total number of pixels within a given range (the framework uses 32 bins). Implement parallel computation of the histogram of a given image. Implement the kernel `histogram_kernel` in `src/filtering.cu`.
Note: There are several ways to achieve the correct result
Note: Take race conditions, etc. into consideration and properly synchronize your code
Suggestion: One possible solution uses a 2D array in shared memory and aggregates the results

- Allocate a 2D array in shared memory (One row per thread in block, one column per bin)
- Compute the histogram of one column of the image per thread, store the values in the array
- Sum the bins of the block using one thread per bin (if there are fewer threads than bins, use a loop)
- Add the result to the histogram in global memory using `atomicAdd()`