PMPP WiSe 2024 - Exercise 1

Johannes S. Mueller-Roemer and Sebastian Besler

TECHNISCHE UNIVERSITÄT DARMSTADT

WiSe 2024 – v1.25 (2023/07/21)
Sheet 1

Task 1.1: Setup and execution

- **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].
- **Upload the framework (from your own machine):** `scp Framework.tar` *`TU-ID@loginnode`*`:Framework.tar`
- **Load cuda:** `module load cuda`
- **Extract framework:** `tar -xvf Framework.tar`
- **Setup build dir:** `mkdir build` and `cd build`
- **Configure build:** `cmake ..`
- **Build:** `make`
- **Run:** `sbatch run.sh`
- **List jobs** You may list your currently enqueued jobs using `squeue`
- **Cancel job** You may cancel your currently enqueued jobs using `scancel` *`JOBID`*

Task 1.2: Memory management

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

1.2a)

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

1.2b)

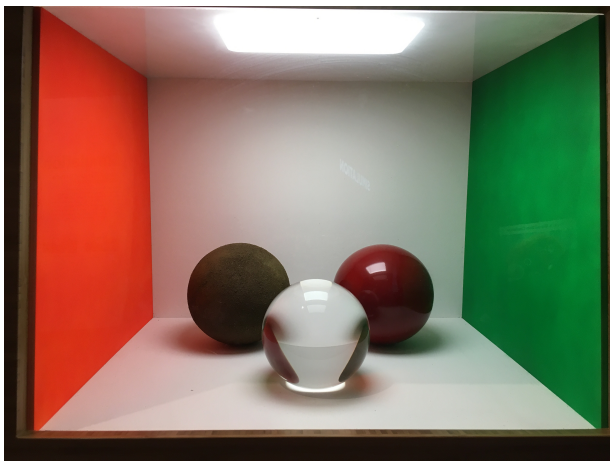
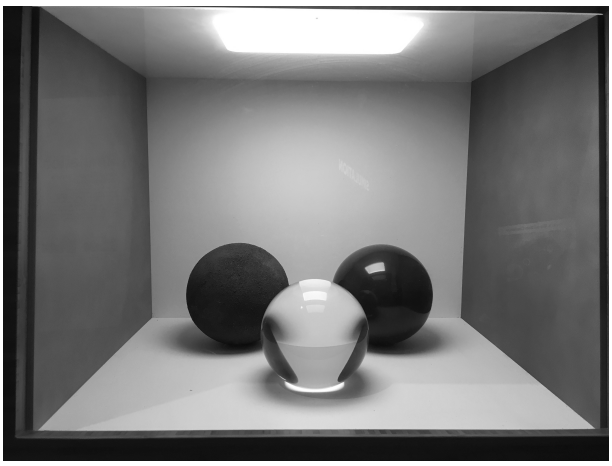
Implement copying matrix data from/to the device in the functions

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

[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: ␣␣␣␣␣␣␣␣

Task 1.3: Color image to grayscale image

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 1.4: Convolution

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 1.5: Histogram

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()`