Exercises for the Lecture: "Architecture and Programming Models for GPUs and Coprocessors" Exercise Sheet № 4

Priv.-Doz. Dr. rer. nat. Stefan Zellmann

WS 2021/22

4 Pointer Jumping and Ray Tracing

Ex. 4.1

The *prefix sum* of the sequence $(a_n) = a_1, a_2, ..., a_n$ is defined as

$$
\begin{aligned}
p_1 &= a_1 \\
p_2 &= a_1 + a_2 \\
p_3 &= a_1 + a_2 + a_3 \\
p_n &= a_1 + ... + a_n
\end{aligned}
$$

a.) Develop a *serial* $O(n)$ algorithm that computes the prefix sum given an array with n elements. The input array can be reused to store the result.

b.) The parallel algorithm Pointer Jumping from the lecture can also be used to compute prefix sums. Adapt the algorithm accordingly. Just as in the lecture, the algorithm is passed an array as input whose entries are the predecessors in a rooted tree. Your adapted version of the algorithm will also be passed another array that assigns each rooted tree a sequence (a_n). The algorithm is supposed to compute the prefix sums of those sequences. If, say, the input was the following:

	0	1	2	3	4	5	6	7	8	9
S	0	0	1	2	3	5	5	6	7	8
(a_n)	1	1	1	1	1	2	2	2	2	2

the output would be:

	0	1	2	3	4	5	6	7	8	9
S	0	0	0	0	0	5	5	5	5	5
(p_n)	1	1	2	3	4	2	2	4	6	8

Is this method to compute prefix sums cost efficient?

Ex. 4.2

The C++ template program accompanying this exercise includes a tiny 2D ray tracer that you are supposed to extend.
In order to compile the template program, you need the cross-platform build tool *CMake* (`https://cmake.org/`). CMake allows us to manage projects that contain multiple *compilation units* that are later linked to a single executable. Typical CMake projects should be built "out-of-source": the binaries and temporary files reside in a different folder than the source code. For that, create a folder (for example as a subfolder of the template program's top-level folder) and call that folder `build` (that's just a convention, not a requirement). Switch to that folder and call the CMake command line program from there and pass the source folder (the one that contains the file `CMakeLists.txt`) as a command line argument. With a Unix shell, the following commands can be used:

```
mkdir build
cd build
cmake ..
```

CMake allows us to set platform-specific parameters via the *CMake cache*. The cache can for example be adjusted via the command line tool `ccmake`, or by directly editing the file `CMakeCache.txt` in the `build` folder. The parameters can also be passed directly to the `cmake` command line program, e.g.:

```
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_FLAGS="-std=c++11"
```

a.)
Extend the 2D ray tracer from the template. For that, you should implement the refraction method from the lecture by filling in the function `refract()`. The template program defines a tiny 2D geometry with different properties w.r.t. refraction. The `.pnm` image files output by the program will allow you to debug your implementation.

b.)
Also handle the case of total internal reflection (TIR). That case should be recognized by your `refract()` function that should return a 0-vector in the case of TIR. If `refract()` returns a 0-vector, the ray tracer will assume that TIR occurred and will call the function `reflect()` to generate a new direction vector. Also implement the function `reflect()`. With the given input geometry and primary rays, what needs to be changed in the template so that TIR can even occur?

Remark: When implementing those two functions, it should not be necessary to use any actual trigonometric functions. For that, recall that the light/surface interaction models we discussed in the lecture operate on unit vectors: normals, light and viewing vectors all point away from the surface and have length 1.

The exercise sheet will be discussed on January 26, 2022.