NEON Intrinsics (V2)

This project gives you experience with SIMD programming.

On the ARM, the SIMD (vector) instruction set is called NEON. In this project you will leverage the NEON instruction set directly **but without using assembly language** per se. Instead, you will use *intrinsics*. An intrinsic is a higher level language passthrough into assembly language.

The idea of this project is to take an RGB image in PPM ASCII format and convert it to black and white according to the Rec. 709 standard (i.e. HDTV). using straight C++ and then again using ARM NEON intrinsics.

Most of this project is given to you. You must use the following files without change:

- `ppm.cpp`
- `ppm.hpp`
- `student_main.cpp`

You will implement the two functions found in `student_implementation.cpp`.

The picture file you will use for the purposes of grading is `storm.ppm`. For test purposes, start out with `rgb.ppm`.

Your project will emit `NEON.ppm` and `CPU.ppm`. These must match each other byte for byte. For testing purposes, run the program with the files indicated above. Both `NEON.ppm` and `CPU.ppm` must also be identical to `storm_out.ppm` or `rgb_out.ppm` depending upon what you used as input.

Consult the man pages for `diff`.

The image files I provide are:

File	Content
rgb.ppm	Test file containing columns of RGB
rgb_out.ppm	Expected results of rgb.ppm
storm.ppm	Test file containing an AI generated image
storm_out.ppm	Expected results of storm.ppm

Images are written in the ASCII variant of ppm so you can open them in your editor to see where you went wrong.

Partnership Rules

All work is solo.

Command Line

Invoke your program with one command line argument, a PPM file. For example:

```
% ./a.out storm.ppm
```

Y

Y is the name given to the black and white component of several color spaces. Given RGB, your program will produce Y according to the weights specified by the REC 709 / HDTV specification. These are:

```
Y = r * 0.3333 + g * 0.6 + b * 0.0666
```

PPM

Consult the source code in `ppm.cpp` to learn the format of PPM.

Compiling

You'll need to compile with something like this:

```
g++ -g -std=c++11 -pthread student_main.cpp student_implementation.cpp ppm.cpp
```

Once things are working, replace the `-g` with `-O3`.

Step 2 - SIMD

See here for a simple concrete example.

SIMD means *single instruction multiple data*. ARM offers the NEON instruction set. These are assembly language instructions that provide many of the same operations as the standard instructions you've already used such as add, subtract, multiply etc.

However, the NEON instructions differ in an important way. They operate over several registers in parallel. In fact, the NEON instruction set adds a **new** set of registers to the ARM CPU designed for this purpose.

Intrinsics are proxies for assembly language instructions that are callable directly from C and C++.

Here is a link to a document describing NEON intrinsics and the whole underlying instruction set architecture. By the way, the only way little ARM machines (like a Chromecast, Firestick, your phone, etc) can process video so well is by virtue of extended instruction sets like NEON.

This might be a better reference.

For example, `vaddvq_f32()` is a proxy for the `FADDP` instruction. This is an intrinsic you might consider as part of your project. Another is `vmulq_f32()`.

To use an intrinsic in C or C++, we need new types. We are using floats, four at a time. The type that represents this is: `float32x4_t`. Having the right type is necessary for type checking purposes but more importantly in this case, for computing the size (four floats).

Setting expectations

Apart from writing all the code given to you, the part that you are responsible for, in my implementation, is about 35 lines.

`storm.ppm`

This image was created by a generative AI using the prompt:

`Storm tossed fishing trawler in a lake in the style of Turner`

Figure 1: storm

The output should be:

`rgb.ppm`

should produce:

Figure 2: output

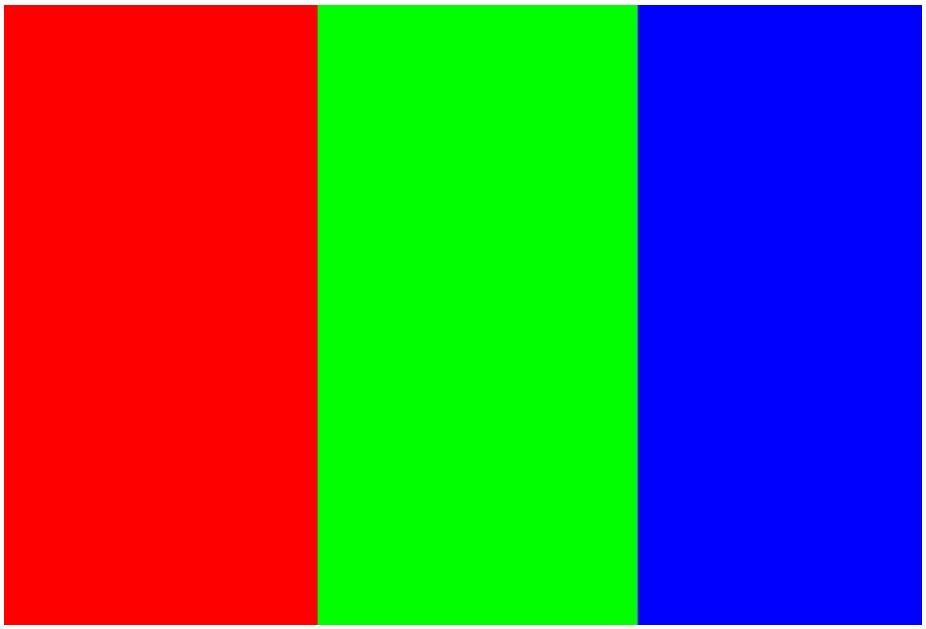

Figure 3: rgb

Figure 4: rgb