Getting Started

Since it was a little hard for me as a physicist to get started at first, and I believe the exact setup for running C++ varies strongly per device, I have taken the time to write down the steps I followed to get the code running on my device. First of all, by choosing Meadowlark for the SLM, we need to use Windows. The most obvious choice for C++ on Windows is Visual Studio, but at the time I did not know this and there happens a lot under the hood, which did not give me a feeling of being in control. Therefore, I installed MSYS2 with MingW64. We start by getting some code running, then we include OpenCL and VkFFT before including the Meadowlark libraries. Then optionally one can add mosquitto, like we did at UvA and finally we will run the whole code. I will try to provide some examples along the way, but let it be clear that I am no expert on this. I just provide information on what worked for me (like any modern day influencer).

Installing MSYS2/MingW64 and first runs

1. I started by installing MSYS2. I used the version of 2023/10/26, but I assume nothing too much will change that removes compatability. I ran the installer and had the default installation folder `C:/msys64/`.
2. Next, we want to install relevant packages for MingW64. I tried to download the whole relevant set of packages using the following command in the *purple* MSYS command prompt:

```
pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

At first, I thought this would be enough, but it did not yet install everything, including gcc/g++ compilers.

3. Thinking something had gone wrong, I tried another general update command in the *purple* MSYS command prompt:

```
pacman -Syu
```

This triggered a whole host of updates and took quite some time to install. Now everything was installed with the exception of the gcc/g++ compiler. To achieve that, I ran the command:

```
pacman -S mingw-w64-x86_64-gcc
```

Now we have installed the relevant packages and the compiler so we can close the command prompt.

4. To test whether our installation has worked, I made a simple `hello.cpp` script, that does nothing else than say "Hello World". The file is in the folder of examples and should be copied to the root folder of MSYS2. In my case that was `C:/msys64/home/iknottn/`, but I assume your username is different. Copy the `hello.cpp` file to that root directory.

Now open the *blue* MINGW64 window. This will use the packages that we just installed in the previous steps. The command prompt should open directly at `~`, which is the root folder that we mentioned above and copied our file to. To compile the file we then run g++:

```
g++ hello.cpp -o hello.exe
```

This should compile and produce an executable `hello.exe` file in the root directory. Note that one could also use the gcc-compiler, but then one has to include the C++ libraries with `-lstdc++` to allow the use of C++

libraries such as <iostream >.

5. Even though compilation went fine for me, I could not run the executable yet because I received and error message that "libstdc++6.dll" was not found. This is a common error message with MingW64 on Windows it seems. The solution is either some static linking of the library and including it in the folder or adding the relevant MingW64 folder to the PATH enviroment variable in Windows. I chose the latter and included `C:/msys64/mingw64/bin/` to the PATH and could run the compiled program.

Including OpenCL to the code

Now that we can compile general C++ files, we need to make sure we can also run GPU code. For general hardware compatability I chose OpenCl.

Note: Programming GPUs is a profession and I am by no means qualified to do this correctly. There are many online resources and examples, such as:

- Examples from the OpenCL-Programming-Guide
- ...

Let's start with installing the thing.

6. The first thing is to get the relevant libraries and header files. For this, I used OpenCL-SDK-Light, although one can also use the seemingly more complete OpenCL-SDK I assume. After downloading and unpacking the `.zip`-file of the OpenCL-SDK-Light, we see two folders: `include` and `lib`. To use these with MingW64, we need to copy the contents of these files to the MingW64 folders.
    1. Copy the contents of `include` to the include-folder. For me this was `C:/msys64/mingw64/include/`.
    2. Likewise, pick the right library (I assume `x86_64` for most systems nowadays) and copy this to the library folder: `C:/msys64/mingw64/lib/`.
7. To test the compilation of OpenCL, I have made a simple script called `opencl_test.cpp`. To compile this, open the *blue* MingW64 terminal and type:

```
g++ opencl_test.cpp -lOpenCl -o opencl_test.exe
```

Note that we added the `-lOpenCl` flag, which tells the compiler to use the static library that we have copied in the previous step. Inside the `.cpp`-file, I have included the header files that we copied. For more examples, see one of the links at the top of this paragraph.

Installing VkFFT

I based the FFT calculation on the VkFFT library. It is really fast and supports many GPU backends, such as CUDA and OpenCL.

8. Download VkFFT from its repository here.
9. Inside the folder you find a folder called `vkFFT` in which there is a header file and another folder containing further headers. Copy both the header file and the folder into the MingW64 `include` folder (i.e. `C:/msys64/mingw64/include/`).

10. Now you should be able to compile with VkFFT. To test, I have made a simple script called `opencl_test_vkfft.cpp` that does the same as the previous OpenCL test but has the added operation of an FFT at the end on the data. This has no meaning, but serves as a test. To compile the script we need to specify for VkFFT that we use OpenCL as a backend, which we do by writing in the *blue* MingW64 terminal:

```
g++ opencl_test_vkfft.cpp -lOpenCl -DVKFFTBACKEND=3 -o opencl_test_vkfft.exe
```

Adding the Meadowlark dependencies

11. First install the Blink SDK that Meadowlark delivered with the SLM. Inside the installation folder should be a number of libraries used for the SDK, as well as examples.
12. Meadowlark has included several `.dll`-files that need to be in the same folder as the executable that one wants to create. These are in the repository of the main code. For simplicity, they are copied into almost all the folders, so that you can compile anywhere. Adjust to your own wishes. You do not need to do anything right now. For compilation, it requires us to add the library so we have to add the `-lBlink_C_wrapper` flag to our compilation command.

Installing MQTT (optional)

At the UvA, we use MQTT for connecting multiple pieces of hardware. The SLM code cannot run on our main Control computer so also this is connected through MQTT. It is thus quite experiment specific what one chooses and it is not necessary to use this, but I will show how I installed it anyway.

13. Download the installer from here. I installed the Files only in the default folder (C:/Program Files/mosquitto).

14. Inside the installation folder there is a folder called `devel`. Copy the `.h`-files inside to the `include` folder and the `.lib`-files to the `lib` folder of MingW64.

15. You should now be able to compile with MQTT as well, but be careful that you do not forget to link the `-lmosquitto` in the compilation. An example command:

```
g++ opencl_test_vkfft_mqtt.cpp -lOpenCl -DVKFFTBACKEND=3 -o opencl_test_vkfft_mqtt.exe -lmosquitto
```

It is necessary to have the flag *behind* the `-o` output tag.

Running the full code

Now we have all the substeps completed, we should be able to run the code. To do so, I installed the repository from our group GitHub. To install it, I cloned the repository into my `msys64/home/iknottn` folder using:

```
git clone https://github.com/StrontiumGroup/SLMSorting.git
```

There is a lot of code and to get a feel for the structure I advise you to read the `ReadMe.md`, or - in a later stage - the actual documentation.

There are four examples provided:

- `example_noslm.cpp`: An example with no SLM commands. This is useful if no SLM is connected or the SDK is not installed on the PC. It sorts a 6x6 array into a 3x3 grid and stores patterns in between in the folder `/masks`. **Note** that you still need the `-lBlink_C_wrapper` tag and the Meadowlark `.dll`-files to compile the program.
- `example_slm.cpp`: Does the same sorting but does not save the masks, but rather displays them on the SLM.
- `example_multipattern_noslm.cpp`: Another example without SLM commands, but this time sorting the 6x6 pattern into a 16-atom circle. It saves the holograms in the `/masks` folder.
- `example_multipattern_slm.cpp`: Does not save but displays the holograms on the SLM.

Now I opened the *blue* MingW64 command prompt and navigated down to `SLMSorting` using:

```
cd SLMSorting
```

and then compiled the an example using:

```
g++ ./examples/example_noslm.cpp ./core/*.cpp -lOpenCl -DVKFFT_BACKEND=3 -I ./core/ -L ./lib/ -lBlink_C_wrapper -o ./bin/example_noslm.exe
```

Many things should look familiar here. What is new is that the files are now ordered into multiple directories, which are included in relative imports. These are important for the libraries and also the final `/bin` directory is important. Inside that directory the Meadowlark libraries are located. Also, we use relative directories in the examples for now.