PRISM

C++/CUDA implementation of compact S-matrix formulism for fast Multislice simulation of electron micrographs. PRISM is capable of producing simulated micrographs with tolerable error compared with the original implementation by Kirkland while providing computational acceleration of over ###x.

Installation

Binary Installers

Links to binary installers should go here in the future

Building PRISM from the source code

PRISM is built on top of CMake, a cross-platform compilation utility that allows a single source tree to be compiled into a variety of formats including UNIX Makefiles, Microsoft Visual Studio projects, Mac OS XCode projects, etc. Only having to maintain one project means PRISM developers spend less time managing multiple code-bases and more time optimizing, debugging, and extending PRISM, resulting in a better end-user experience.

External dependencies

To install PRISM, you must first install Cmake and FFTW.

To accelerate PRISM with CUDA GPUs, you must also install the CUDA toolkit and have a CUDA enabled GPU of compute capability 2.0 or higher. PRISM was developed using CUDA 8.0, but likely works with older versions as well and we welcome feedback from any user who attempts to do so (CUDA 7.0, 7.5 also have been reported to work). *Note: Even if you download a binary version of the GPU codes, you will still need to have the CUDA toolkit installed so that the `cuFFT` libraries can be found at runtime.*

If you are building the GUI from source, you will also need Qt5.

Getting the source code

Once the dependencies are installed get the PRISM source either from compressed source files or directly from Github using `git clone`

Building with CMake from the command line

To build PRISM from the command line with CMake, open a terminal and navigate to the top of the source directory

```
cd /path/to/PRISM/
```

Conventional CMake practice is to use out-of-source builds, which means we will compile the source code into a separate directory. This has a number of advantages including providing the flexibility to build multiple version of PRISM (such as compiling with/without GPU support), and allowing for easier cleanup. First, make a build directory (the name doesn't matter) at the top of the PRISM source tree.

```
mkdir build
cd build
```

Then invoke CMake

```
cmake ../
```

This will generate a Makefile/Visual Studio Project/Xcode project with the necessary dependencies and paths to compile PRISM. The default behavior is to build only the CLI without GPU support. These options can be enabled as described in later sections. Finally, compile and install PRISM with:

```
make
```

For faster compilation, add the `-j` switch to `make` to use multiple threads, for example to compile with 8 threads

```
make -j 8
```

If this succeeds, the executable file `prism` has been built and can be run from within the build directory. To permanently install them, invoke

```
make install
```

which may require `sudo` privileges. This will place the files in `/usr/local/bin` on Unix systems.

Setting CMake options

All aspects of how PRISM is compiled, such as whether or not to include GUI or GPU support, are controlled through CMake variables. There are at least four different ways to adjust these:

If you are using the CMake GUI, then options are turned on/off with check boxes and paths are set by clicking browse and navigating to the file/folder of interest.

If you are using the command line tool, `cmake`, then options are set with the `-D` (Define) flag. For example, to set `My_Variable` to 0 one would add `-DMY_VARIABLE=0` to the call to `cmake` (see the sections on enabling GUI or GPU support for more examples).

There is also the hybrid "command-line GUI" option, `ccmake`, which provides an interactive way to adjust CMake options from the command line.

Finally, you can also directly edit a file called `CMakeCache.txt`. The first time you run CMake for a given project this special file will be created containing all of the option settings. Whenever you generate a project or Makefile, the options will be read for the `CMakeCache.txt` file, so options can be changed here directly.

Note: Any time you change CMake options for a particular project you must regenerate the build files and recompile before the changes will actually take effect

Enabling GPU support

To enable GPU support, set the CMake variable `PRISM_ENABLE_GPU=1`. You must have the CUDA toolkit installed and the appropriate paths setup as described in the CUDA documentation so that CMake may find `nvcc` and the necessary libraries to build/run PRISM.

Enabling the GUI

To build the GUI from source, you must install Qt5 and set the CMake variable `PRISM_ENABLE_GUI=1`. I find that CMake sometimes has trouble automatically finding Qt5, and at configuration time may complain about being unable to find packages such as `Qt5Widgets`. An easy solution is to follow CMake's suggestion and set `CMAKE_PREFIX_PATH=/path/to/Qt5` where `/path/to/Qt5` should be replaced with the path on your machine. For example, on my Macbook with Qt5 installed through Homebrew I might invoke

```
cmake ../ -DPRISM_ENABLE_GUI=1 -DCMAKE_PREFIX_PATH=/usr/local/Cellar/qt5/5.8.0_1
make -j 8
make install
```

Using PRISM from the command line

PRISM contains a command line tool, `prism`, that can be used to run simulations from within a terminal, bash script, etc. Building it requires the CMake variable `PRISM_ENABLE_CLI=1` at compilation time, which is the default behavior.

Options

The following options are available with `prism`, each documented as ***long form (short form)*** *parameters* : description

- --***input-file (-i)*** *filename* : the filename containing the atomic coordinates, which should be a plain text file with comma-separated values in the format x, y, z, Z
- --***output-file(-o)*** *filename* : output filename
- --***interp-factor (-f)*** *number* : PRISM interpolation factor
- --***num-threads (-j)*** *number* : number of CPU threads to use
- --***num-streams (-S)*** *number* : number of CUDA streams to create per GPU
- --***slice-thickness (-s)*** *thickness* : thickness of each slice of projected potential (in Angstroms)
- --***num-gpus (-g)*** *number* : number of GPUs to use. A runtime check is performed to check how many are actually available, and minimum of these two numbers is used.
- --***help(-h)*** : print information about the available options
- --***pixel-size (-p)*** *pixel_size* : size of simulation pixel size
- --***cell-dimension (-c)*** *x y z* : size of sample in x, y, z directions (in Angstroms)
- --***algorithm (-a)*** *p/m* : the simulation algorithm to use, either (p)rism or (m)ultislice
- --***energy (-E)*** *value* : the energy of the electron beam (in keV)
- --***alpha-max (-A)*** *angle* : the maximum probe angle to consider (in mrad)
- --***potential-bound (-P)*** *value* : the maximum radius from the center of each atom to compute the potental (in Angstroms)
- --***also-do-cpu-work (-C)*** *0/1* : boolean value used to determine whether or not to also create CPU workers in addition to GPU ones
- --***force-streaming-mode*** *0/1* : boolean value to force code to use (true) or not use (false) streaming versions of GPU codes. The default behavior is to estimate the needed memory from input parameters and choose automatically.
- --***probe-step (-r)*** *step_size* : step size of the probe (in Angstroms)

- --***num-FP (-F)*** *number* : number of frozen phonon configurations to calculate