Geo1000 – Python Programming for Geomatics – Assignment 4

Assignment 4

Deadline: Friday, October 25, 2024, 18h00 CEST

1 Introduction

In this assignment you are asked to make the output functions of a n-body simulation. The outputs will be the 3D point coordinates of the bodies for each time step simulated. They can be visualized in QGIS (at least, the 2D projections). You will do so, by starting from both a Python script and a C++ program.

This assignment is *preferably* made in groups of 2 (enroll with your group or individually in Brightspace), and your mark will count for your own final grade of the course. Helping each other is fine. However, make sure that your implementation is your *own*.

2 Learning objectives

The goal of the assignment is that a student after this assignment:

- knows how to set up and use a C++ development environment (build, run, difference between debug and release builds)
- gets more familiar with the simlarities/differences of/between Python and C++

3 N-Body simulation

The following description is taken from: http://www.new-npac.org/.

> "The goal of N-Body problems is to determine the motion over time of bodies (particles) due to forces from other bodies. Typical calculated forces include electrostatic force and gravity. The basic method used for solving the problem is to loop forever, stepping discretely through time and doing the following at each timestep:
>
> - Update positions using velocities ($\overline{x}_{i+1} = \overline{x}_i + \Delta(t)\overline{v}_i$)
> - Calculate forces $\overline{F}_i$
> - Update velocities ($\overline{v}_{i+1} = \overline{v}_i + (1/m)\Delta(t)\overline{F}_i$)

[...] Note that it is possible to use multiple resolutions for time, i.e. different $\Delta(t)$ for different particles, but we will only consider uniform $\Delta(t)$.''

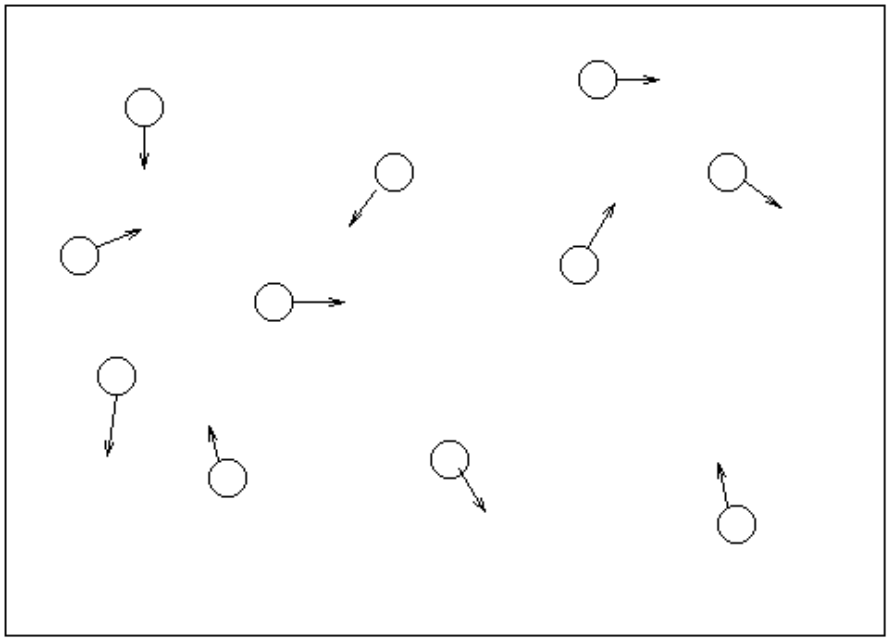

Figure 1: Particles in Motion

4 Preparation

Before starting on this assignment, make sure to read the online book 'C++ for Python Programmers' (go over at least Chapters 1–7) at https://runestone.academy/runestone/books/published/cpp4python/index.html Also, make sure to answer all interactive questions in these Chapters (to test yourself if you've mastered the material).

4.1 C++ development environment

Make sure you have an environment for programming with Python and C++:

- Install a C++ Compiler for your platform
- Install Clion

These steps are documented in the manual for installing CLion prepared by AdHok. You find the correct manual at: https://adhok.bk.tudelft.nl/manuals Please, let us know (e.g. in your report) if certain steps are missing or unclear (this way we can improve the manuals for next year). In case you get stuck, also the developer of CLion (JetBrains) has extensive documentation online: https://www.jetbrains.com/help/clion/installation-guide.html

4.2 Making your first steps with C++

After you have installed CLion, download the code from Brightspace, which was discussed during the lectures on C++. Use it to get familiar with your newly installed development environment: Try to compile and run (some of) the sample programs, in both debug as well as release mode (for more information about this, check: https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html#profiles). Also, intentionally introduce some errors in the code and see what happens (e.g. remove a semi-colon at the end of a statement, or put a too big number to be represented for a data type).

Make a screenshot of CLion showing you trying to compile a C++ program with such an error present. Include it in your report and indicate what you did wrong and what needs to be done to correct the error.

5 Changes you have to make to the Python / C++ code

Once you are (a bit) familiar with CLion, download the `.zip` file from Brightspace. Place the source code from the `.zip` file in a new CLion project title `nbody`.

Now, first get familiar with CLion and the code given to you. See how to execute the C++ code in CLion (build and run). Also, set up the Python interpreter you want to use (as CLion can also be used to develop with Python). Make sure to include both a Debug and a Release option for the build.

As the programs require a program argument from the terminal (to specify how many times to iterate in the simulation), you will have to set this up in the dialog for running the program in CLion: https://www.jetbrains.com/help/clion/run-debug-configuration.html#envvars-progargs (Hint: step 3 - Give an integer in the Program arguments field).

In the terminal you can also pass a program argument. An example, for Python where the script is called nbody.py and the program argument is 1000 (assuming your current working directory is the same as where the script resides):

```
python3 nbody.py 1000
```

and for a compiled C++ version:

```
./nbody.exe 1000
```

Try to understand what happens in the n-body simulation (although it is not necessary for the assignment to grasp all the math behind the simulation).

Adapt both the Python *and* the C++ code to output a CSV file, that stores (at least) the 3D position of the bodies in the simulation for each time step at which the simulation is calculated. Per time step per body (Sun, Earth, ...) a line should be present in the CSV file with on every line:

> name of the body, position x, position y, position z

Make sure there also is a header line in the output files you write with descriptive column names. The delimiter you should use is a semi column (`;`). It should be possible to load the CSV files produced by both your Python and

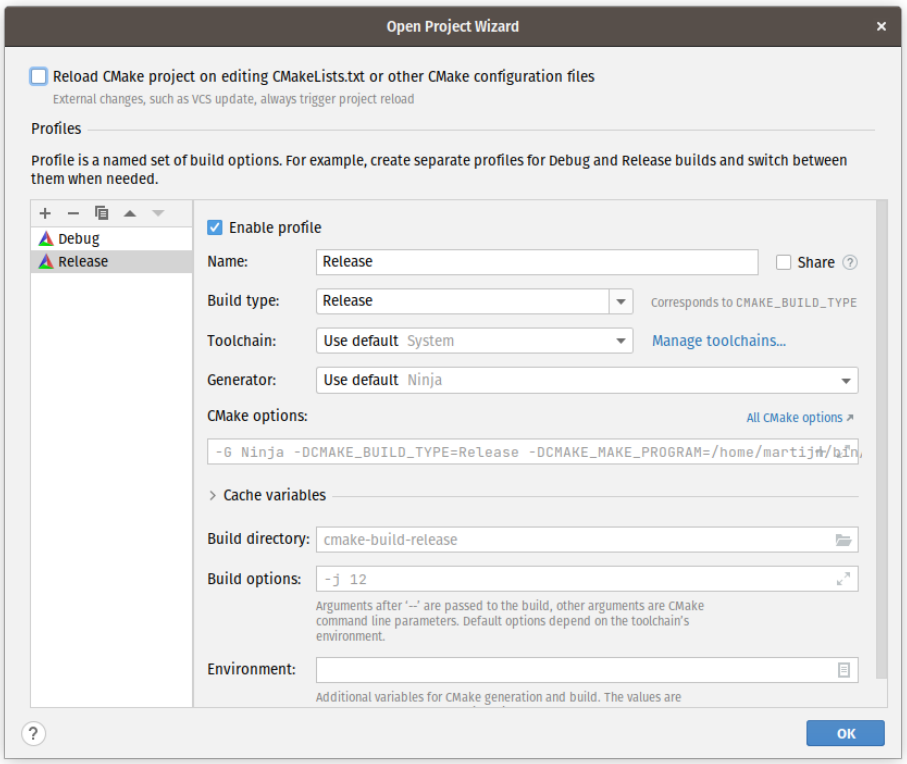

Figure 2: Both Debug and Release builds should be enabled in CLion

C++ program in QGIS. Note, you should put some screenshots of QGIS (after loading the produced CSVs) into your report (see §7).

If you want, you can make the output more 'fancy', e.g. you can output more attributes as well (e.g. components of the velocity vector, or the mass of each body), or you can add a boolean program argument that can enable or disable writing the output file.

6 Benchmark

Give an indication of the runtime of both of your programs. The number of simulation steps should be varied in 5 sizes: n = 5'000, 50'000, 500'000, 5'000'000 and 50'000'000 iterations (or some other incrementing steps, if it turns out to be too demanding to run this amount of iterations on your laptop). For the C++ program: Compile your C++ code both in Debug, as well as in Release mode and include both the timings in your report.

Note: You could write a small Python script that invokes the other programs. More details on how this can be done can be found in the Python book in Section 14.8, or using os.system, together with: time.perf_counter.

7 What to hand in?

Put the modified code in a `.zip` file and upload to Brightspace. If you make scripts for running the benchmarks and plotting the results, also include these. Do *not* include Python environments, C++ build folders (like `cmake-build-debug` or `cmake-build-release`) or compiled, binary C++ executables in your zip.

Next to the code, hand in a report (as PDF file) on Brightspace that contains:

- Your name(s) and student number(s)
- The number of words your report uses.
- A paragraph (max. 200 words) explaining how the Python and C++ programs are different (How is the information in the simulation represented: Which data types do the programs use to represent the data for the simulation?)
- A paragraph reflection on how you went about solving the task (max. 400 words). Which steps did you take? How did you measure the runtime? How did what you learn about Python help you for developing the C++ solution? Did you get stuck? Which sources did you consult when you got stuck? Did you expect the results you obtained? Etc.
- Timings for Python, C++ Debug and C++ Release, in a table and visualised in a chart (x-axis: instance size/y-axis: run time). Advice: Use matplotlib to produce the chart.
- A short description of the hardware you run the simulation on (CPU, memory, operating system).
- A screenshot of a failed compilation of one of the sample programs discussed in class (cf. §4.2).
- One (or more, e.g. 1 overview and 1 close-up) screenshot of QGIS, where you have loaded a CSV file your code did produce for 5'000 iterations. If

you have a recent version of QGIS, you can try to use the 3D map view as well.

8 Grading

You can get 100 points for this assignment. The aspects that we will look at are: Your (modified) code (40 points) and your report (60 points).

Note that if you submit your assignment after the deadline, some points will be removed. For the first day that a submission is late, 10 points will be removed before marking. For every day after that, another 20 points will be removed.

Appendix

C++ reference

The go-to website for C++: https://en.cppreference.com/w/. On Brightspace you can find more references to good C++ books with (much) more on the C++ programming language.

CMake

CMake helps defining the steps for building software (i.e. the steps the compiler needs to take, while compiling your C++ source code). For this, a text file named `CMakeLists.txt` has to exist within your project. In the project you get from us, this text file is present. The important line in that file is:

```
add_executable(nbody nbody.cpp)
```

This instructs CMake to use the compiler to build a C++ Executable `nbody`, based on the source code from `nbody.cpp`. You can read more about it at: https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html