The Snowflake-Toolkit Updates

Torbjörn Rathsman

September 30, 2016

The Snowflake-Toolkit is a software toolkit intended to generate shape data for simulating microwave scattering from snow and ice particles. This paper briefly describes changes since the publication of the Master's thesis, roughly in chronological order.

Contents

1. Converting code to a new build system 2

2. Saving simulation state 2

3. Computing D_{max} 2

4. More stop conditions 2

5. Forcing aggregate merge 3

6. Options for controlling overlap 3

7. Non-convex sub-volumes 3

8. Global scaling of prototypes 4

9. Initial work on graupel generator 4

10. Adding Alice Long optIons C++ Extractor 4

11. Empty element for the graph traversing program 4

12. More probability distributions for setting deformation parameters 4

13. Improvements on geometry sampling 5

A. External libraries required by the snowflake toolkit 5

1. Converting code to a new build system

While `wand` has been a great tool for managing semi-large projects, its algorithm does not detect any cyclical dependencies. Also, the codebase, especially the frontend, is quite messy. Moreover, `wand` has no good API for calling the engine without invoking `wand` itself. Also, it has no plug-in framework, and no support for code generation.

With this background, a new tool called `maike`[1] was written. This tool resolves all listed problems, and it also improves compilation times an order of magnitude, as well as diagnostics. For example, it is easy to list all external dependencies by using the option `list-external-targets`[2]. Using this option in the code directory for the snowflake toolkit currently produces the list in appendix A.

Maike requires `libjansson`[1][3]. Therefore, the corresponding development packages are needed to compile and link the `maike` executable. However, the snowflake toolkit has not a dependency to that library itself.

2. Saving simulation state

Since simulations can run for some time, it is useful to be able to save the current state to disk, so the simulation process can be continued later. This new feature is implemented using the `libhdf5` libraries. The choice of HDF5 is motivated by its capabilities to store a large amount data in a machine-friendly format [4]. The simulation state can be reloaded with the `statefile` option.

The statefile is standalone. This means that all external resources are embedded into the statefile. For reference, the path to the external resources are stored also, but no data are read from these files when recovering the state.

It is important to notice that statefiles are not intended to be compatible with other versions of the toolkit. This means that an old simulation cannot be continued with another version of the toolkit than the one that created the file.

3. Computing D_{max}

Previously, the toolkit has only provided the radius of `Solid`s, defined as the largest distance from its centroid to one vertex. Now, $D_{\max}$ is also available, and it is computed from the largest distance between any two vertices.

4. More stop conditions

A desired feature has been the ability to use different stop conditions. With these changes, the aggregate generator can stop based on the following conditions listed in table 1.

Table 1: All stop conditions supported by the aggregate generator
,

Condition	Description
`iterations=N`	Run a fixed number of iterations
`subvols_max=N`	Run until there is an aggregate with `N` elements
`d_max_max=d_max`	Run until there is an aggregate with `d_max` as largest its extent
`v_max=V`	Run until there is an aggregate with volume `V`
`n_dropped_max=N`	Run until the model has dropped `N` particles
`infinity`	Run forever, or until the simulation is stopped externally

5. Forcing aggregate merge

Another addition to the aggregate generator is a force-merging mode. In this mode, a merge event that failed due to an overlap is not rejected directly, but instead a different pair of faces are chosen until the merge succeeds, or until the number of retries left goes to zero. This behaviour is controlled by the `merge-retries` option.

6. Options for controlling overlap

It is now possible to generate aggregates with overlap. The overlap is given relative to the number of sub-volumes in the smallest aggregate. Zero corresponds to no overlap while one corresponds to an overlap everywhere. It is possible to set both lower and an upper limits on the amount of overlap. This is done by the options `overlap-min`, and `overlap-max`, respectively.

The overlap introduces a slight error in the total volume. Currently, the toolkit assumes that the overlapping volume in each overlap is one tenth of the smallest sub-volume in that overlap.

7. Non-convex sub-volumes

The restriction on non-convex sub-volumes makes it hard to use non-convex shapes as crystal prototypes, because it requires that the user splits the prototype into convex sub-volumes before it can be used.

This restriction has now been removed, by using a new hit-test algorithm, based on ray-casting. The test is based on the fact that a ray that begins inside the volume has to cross the boundary an odd number of times.

Another issue with non-convex sub-volumes is the seed for the geometry sampler. With convex sub-volumes, the centroid works as a seed, since it has to be inside the volume. This approach has been replaced by a stochastic approach, where a number

of points inside the bounding box is tried, and the first point that hits the sub-volume will be used as starting point.

8. Global scaling of prototypes

All crystal prototypes now supports a global and isotropic scaling parameter. This makes it possible to have a randomised size, without a random aspect ratio.

9. Initial work on graupel generator

An attempt has been made to implement a graupel generator. While it generates spheres out of spheres, it produces particles with too low effective densities. It is possible to increase the effective density, but it increases the processing time. For graupels, the polygon mesh structure is impractical, and it is probably much faster to model the spheres as a centre and a radius. Then there is no need to track individual faces and vertices.

10. Adding Alice Long optIons C++ Extractor

As more options are added to the programs, maintaining synchronisation between the help text and the actual implementation becomes a heavy work. Therefore, `Alice`[5] has been used instead of GNU `getopts` for processing the command line in two of the programs. The goal is to replace GNU `getopts` everywhere.

Alice improves readability, and maintainability, of the code, but adds pushes the requirement from C++11 to C++14. This means that GCC version 5 or later[6] is required to compile the toolkit.

11. Empty element for the graph traversing program

A `flake` file can now use an empty prototype. This makes it possible to choose starting coordinates explicitly. Also, this makes it possible to use the graph traversing program to assemble an aggregate with only one non-empty prototype.

12. More probability distributions for setting deformation parameters

The aggregate generator now has more choices for probability distributions when for particle deformations. In addition to the Γ distribution, δ, and Exp is available. Also it is possible to use sampled probability distributions stored in tab delimited text files.

The δ distribution—that is, a distribution which always return its mean value—has been present since the first release by detecting a small standard deviation. With this change, the user explicitly states that a δ distribution should be used.

Neither δ or Exp accepts a standard deviation. The former, since it is zero anyway, and the latter, because it is the same as its mean value. A `custom` distribution is parameterised by the data stored in the text file.

13. Improvements on geometry sampling

Sometimes, it is easier to specify the total number of voxels, instead of the number of voxels in each direction. Now, it is possible to specify a desired number of voxels N, and an aspect ratio $r_x : r_y : r_z$, which determines the step size in each direction. The step size in direction i is computed using the expression

$$\mathrm{d}x_i = r_i \left(\frac{V}{N r_x r_y r_z} \right)^{1/3}$$

where V is the volume of the particle.

A. External libraries required by the snowflake toolkit

This is a list of all external libraries used by the toolkit, as reported by `maike`.

- `hdf5`
- `hdf5_cpp`
- `pthread`
- `hdf5_hl_cpp`

References

[1] `https://milasudril.github.io/maike/`

[2] `https://milasudril.github.io/maike/cmdline.html`

[3] `http://www.digip.org/jansson/`

[4] `https://support.hdfgroup.org/HDF5/whatishdf5.html`

[5] `https://github.com/milasudril/alice`

[6] `https://gcc.gnu.org/projects/cxx-status.html`