SSDEM Documentation

Last Update: 7/7/20

Started By: Derek C. Richardson

Contact Info:
Department of Astronomy
University of Maryland
College Park MD 20742
Tel: 301-405-8786
E-mail: `dcr@astro.umd.edu`

Contents

1 OVERVIEW

This is brief and incomplete documentation of the soft-sphere discrete element method (SSDEM) implementation in `pkdgrav`.

1.1 Other Documentation

For documentation of walls in `pkdgrav`, see `walls.pdf`.

For documentation of `ssio.py`, a utility for reading and writing binary `ss` files in `python`, see `ssio_py.pdf`.

There is an SSDEM tutorial at `https://goo.gl/NIYJ5z` (or email `dcr@astro.umd.edu`).

2 USING A FIXED SEARCH BALL

This section describes how to properly use the `DEM_FIXED_BALL` compilation option, which can significantly reduce the time required to run a simulation (depending on the circumstances). We first give an introduction to the function and behavior of a search ball, then give a description of the default dynamically generated search ball, and, finally, describe the different fixed search ball options.

If you already know which search ball option you want to use, simply read the first three lines of the sub-section of that option to understand what changes are required for proper use.

WARNING: Problems can occur with certain scenarios using a fixed search ball, such as a rubble pile asteroid flying by a planet. In this case, the search ball will be huge (the size of the planet), so all the smaller particles will include each other in their search balls. This can cause a parallel run to crash without warning due to a memory overflow. In those cases, if the large particle is not intended to collide with the smaller ones, consider setting its radius to something much smaller.

2.1 Introduction

In `pkdgrav`, each particle maintains a list of its nearest neighbors in order to check for potential colliders. Rather than performing a collision check for each other particle in a simulation, which would take N^2 time, a particle only performs a collision check on its nearest neighbors. The number of nearest neighbors that a particle has to search for in order to effectively determine whether it is in overlap with another particle largely depends on its size with respect to other particles. For equal-sized particles, the theoretical maximum number of neighbors that a single particle may be in contact with is 12. (Note: this is only true if the particles are "just touching" and not overlapping significantly.) In general, for a size distribution of particles, the maximum number of overlaps that a single particle may experience is given by,

$$\mathrm{MAX\_NUM\_OVERLAPS\_PER\_PARTICLE} = 8 + 4\left(\frac{R_{\max}}{R_{\min}}\right)^2, \tag{1}$$

where $R_{\max}$ and $R_{\min}$ are the maximum and minimum particle radii in the simulation, respectively. The value of `MAX_NUM_OVERLAPS_PER_PARTICLE` is defined in the header file `dem.h`.

Of vital importance to detecting neighbors, and therefore particle contacts, is the use of a search ball by each particle. The search ball of a particle defines the limited region around it where it will look for nearest neighbors. In the default implementations of `pkdgrav` and `SSDEM`, the search ball is dynamically constructed each time step. Years of practical use of the codes have shown that this might not be the most efficient way to find nearest neighbors. Therefore, methods that generate a fixed search ball for each particle make the code run faster as the search ball does not need to be re-constructed each time-step. Three different fixed search ball strategies were implemented in order to speed up processing time and ensure proper neighbor finding for certain `pkdgrav` scenarios. We outline these methods below, and best practices for using them. First, we provide more context by reviewing the default dynamic search ball method.

2.2 Dynamic Search Ball

For the default compilation of SSDEM, a particle's search ball is dynamically set. The generation of the search ball is linked with the partitioning of space into sub-domains and the building of the k-d tree. A particle's search ball is thusly defined by the local number density of particles.

For this dynamically generated search ball, the number of nearest neighbors that each particle needs to keep track of is set by the `nSmooth` parameter defined in the run-time parameter file (`ss.par`). The value of `nSmooth` should be set, similar to Eq. (1), as,

$$\text{nSmooth} = 8 + 4\left(\frac{R_{\max}}{R_{\min}}\right)^2. \tag{2}$$

This ensures that a small particle is able to detect the presence of a larger particle in its vicinity. However, for a dynamic search ball, `MAX_NUM_OVERLAPS_PER_PARTICLE` can be set to the theoretical limit of 12 if the particles in the input file are size-sorted in *increasing* order. Size-sorting in increasing order allows the smallest particle to first resolve any overlap conditions and apply the necessary forces on itself and its larger collisional partners (by applying Newtown's second law of motion). The collision algorithm then progresses to the next largest particle. This next particle only looks for collisions with particles larger than itself (since any collision with smaller particles have already been calculated); hence, it too only needs to look for 12 or fewer contacts. The algorithm progresses through the list of particles. The largest particle need not look for contacts as the forces from smaller particles have already been calculated.

The advantage of using this search ball option is that it is general, and its low memory use (only 12 colliders ever need to be stored into memory). The disadvantage to this method is that the search ball needs to be regenerated every time step, and, for a very large size difference in particles, each particle will have to loop through `nSmooth` nearest neighbors to check for overlaps. These two factors require a lot of processing time, and poorly scale with total number of particles.

2.3 Fixed Search Ball

By compiling `pkdgrav` with the `DEM_FIXED_BALL` compilation option turned on, each particle will have a fixed search ball to look for nearest neighbors. The most obvious advantage of a fixed search ball is that processors spend less time re-computing search balls each time step. The cost of using a fixed search ball is larger memory requirements for certain cases. There are three options for defining the size of the fixed search ball, controlled by the `iFixedBallOption` defined in `ss.par`. These options are described below.

2.3.1 Maximum Diameter

To use this option:

1. Set `iFixedBallOption` to 0 in ss.par.
2. Size-sort particles in increasing order.
3. Set `MAX_NUM_OVERLAPS_PER_PARTICLE` = 12.

This option sets the radius of each search ball to the diameter of the largest particle. This option is optimal for equal-size particles, or a size distribution of particles with a relatively small size ratio ($\frac{R_{\max}}{R_{\min}} \lesssim 3$). The advantage of this option is the speed-up in having a fixed search ball, while maintaining a relatively small memory requirement.

2.3.2 Particle Radius + Mean Radius

To use this option:

1. Set `iFixedBallOption` to 1 in ss.par.
2. Size-sort particles in decreasing order.
3. Use Eq. (1) to set `MAX_NUM_OVERLAPS_PER_PARTICLE`.

With this option, the radius of each particle's search ball is set to its own radius plus the mean radius of all particles. This allows each particle to have a unique search ball that depends on its own size. This option is especially useful for scenarios where a single very large particle interacts with a large number of small particles (of roughly equal size). This allows the large particle to efficiently detect all the small particles as nearest neighbors, while also limiting the size of the small particles search ball. This reduces the number of nearest neighbors that each particle needs to check for overlaps. However, the disadvantage is the large memory requirement. Each particle needs to carry around an overlap list that can contain `MAX_NUM_OVERLAPS_PER_PARTICLE` number of particle IDs.

2.3.3 Particle Radius + Next-largest Particle Radius

To use this option:

1. Set `iFixedBallOption` to 2 in ss.par.

2. Size-sort particles in decreasing order.

3. Use Eq. (1) to set `MAX_NUM_OVERLAPS_PER_PARTICLE`.

With this option, the radius of each particle's search ball is set to its own radius plus the radius of the next-largest particle. This option is useful when there is a large and continuous size distribution of particles, or in the case of a single large intruder interacting with a continuous size distribution of particles. This optimizes the number of nearest neighbors a particle needs to check for overlaps. The disadvantage is the large memory requirements.

2.4 Sorting Particles by Radius

Several utilities order particles in `ss` files by radius. The `ssgen` utility generates initial conditions that are automatically saved in increasing radius order (or in decreasing order with the `-v` option). The `rpx` utility can be used to sort an existing `ss` file by invoking the `-s` option (use `rpx -h` to get a list of options). Otherwise, particle data can be altered manually by converting to `bt` files (see tutorial) or by using the `ssio.py` script (see documentation).

2.5 Search Ball Fudge Factor

As a precaution, the routine that sets the search ball (`dem.c:pkdDEMSetBall()`) includes a fudge factor "`epsilon`" (currently set to `0.1`) that is used when calculating the search ball radius. Mathematically, this is how it is applied for the three cases:

$$\begin{aligned}
0: \; R_{\rm ball} &= R_{\rm max} + (1+\epsilon)R_{\rm max} \\
1: \; R_{\rm ball} &= R_i + (1+\epsilon)\bar{R} \\
2: \; R_{\rm ball} &= R_i + (1+\epsilon)R_{i+1}
\end{aligned}$$

(In the last case, it is assumed the file is in decreasing size order; R_{N+1} is just set to R_N.)

The utility `ssn`, which generates neighbor lists from `ss` files, includes the three fixed-search-ball options above and has a configurable fudge factor. The utility can also be used to flag particle overlaps that are detected within the search radius.

3 RESTARTS WITH SSDEM

In SSDEM, particles keep track of overlaps (contacts) with other particles (and walls, if applicable), but this information is not stored in `ss` files or checkpoints. Instead, special `.dem` files are generated every full `ss` output, and these can be used for clean restarts. (In fact, checkpointing is disabled altogether when running `pkdgrav` with SSDEM.)

To restart a simulation from a full `ss` output with a corresponding `.dem` file, use the following procedure:

1. Set `achInFile` in `ss.par` (or equivalent `pkdgrav` parameter file) to the `ss` output that will be used, e.g., `ss.01000`.

2. Change `iStartStep` to the timestep of the output (1000 in this example).

3. Set `bReadDEMData` to 1.

4. Rerun with the `+overwrite` option.

Note that for this procedure the following preprocessor macros cannot be changed between restarts:

- `USE_WALLS`.
- `USE_DEM_ROTATION_DASHPOT`.
- `MAX_NUM_OVERLAPS_PER_PARTICLE` (in `dem.h`).
- `MAX_NUM_OVERLAPS_PER_PARTICLE_FOR_WALLS` (in `dem.h`).

Generally speaking, any changes to run parameters should also be considered carefully to avoid any inconsistencies on restarts. This includes changing the number or order of walls, if applicable.

The frequency of outputting `.dem` files is set by the `iOutInterval` parameter. The `.dem` files are big, so you can limit how many are kept using the `nDEMOutputs` parameter (the default is 1, corresponding to keeping only the latest `.dem` file).

3.1 Format of `.dem` Files

A `.dem` file is a binary file consisting of a header with the following ordered information:

- Time (double).
- Number of particles (integer).
- `MAX_NUM_OVERLAPS_PER_PARTICLE` (integer).
- `MAX_NUM_OVERLAPS_PER_PARTICLE_FOR_WALLS` (integer; -1 if `WALLS` is not defined).
- Rotation dashpot flag (integer; 1 if `DEM_ROTATION_DASHPOT` is defined, 0 otherwise).

This is followed by an ordered data record, one for each particle, of format:

- Predicted velocity (vector).
- Predicted spin (vector).
- `MAX_NUM_OVERLAPS_PER_PARTICLE` repetitions of:
  - `iOrder` of overlapping particle (integer; -1 if none).
  - Tangential displacement (shear) from initial contact (vector).
  - Previous contact normal (vector).
  - If `DEM_ROTATION_DASHPOT` defined:
    - Rolling angular displacement (vector).

        * Twisting scalar displacement (double).
    - Overlap counter (4-byte integer).

- If `WALLS` defined, `MAX_NUM_OVERLAPS_PER_PARTICLE_FOR_WALLS` repetitions of:
    - ID of overlapping wall (integer; -1 if none).
    - Tangential displacement (shear) from initial contact (vector).
    - Previous contact normal (vector).
    - If `DEM_ROTATION_DASHPOT` defined:
        * Rolling angular displacement (vector).
        * Twisting scalar displacement (double).
    - Overlap counter (4-byte integer).

Note this information is also given in the comments of `src/pkdgrav/dem.h` related to `DEMHEAD_SIZE` and `DEMDATA_SIZE`.

4 DIAGNOSTIC OUTPUTS

Currently there are 2 sets of diagnostics that can be generated for SSDEM runs.

4.1 SSDEM Statistics

At a frequency set by the `iDEMStatsInterval` parameter, basic SSDEM statistics can be output in `.demstats` files. These are text files that contain a header describing their contents; in brief, the data stored include histograms of particle overlaps, overlap orientations ("$\cos\alpha$"), and tangential spring **S** magnitudes. If `sm` and `ffmpeg` available, the file `demstats.sm` can be used to visualize the data; copy the file (from the `pkdgrav etc` directory) to the working directory and run the `demstatsanim` script. This will generate snapshots of the histograms and stitch them together into 3 corresponding movies (`demstats_ohist.mp4`, `demstats_ohist.mp4`, and `demstats_ohist.mp4`, respectively).

Important: accumulated histogram data is lost on a restart. Also, only particle-particle data is stored, not particle-wall.

4.2 SSDEM Forces

If `USE_DEM_DIAG` is defined in `Makefile.in`, at every full `ss` output a `.demdiag` file will be generated. This is a text file with the following format:

```
N
OP [ OW ]
i1 Fnx Fny Fnz Ftx Fty Ftz i2 ... [ w1 Fnx Fny Fnz Ftx Fty Ftz w2 ... ]
i1 ...
...
```

The first line gives the number of particles (N). MAX_NUM_OVERLAPS_PER_PARTICLE is encoded as OP on the second line, along with OW for MAX_NUM_OVERLAPS_PER_PARTICLE_FOR_WALLS, if applicable, in case these values change. Each subsequent line (one per particle) gives a list of overlaps for each particle: i1, i2, etc., are the iOrder numbers of the overlapped particles for that particle while Fn and Ft are the x, y, z components of the normal and tangential forces due to that overlap, respectively; w1, w2, etc., are the wall id numbers (starting from 0) for wall overlaps (if USE_WALLS is defined in Makefile.in), with corresponding force components as for particles. There are OP particle overlap entries per particle and OW wall overlap entries per particle (see dem.h).

Important: particles only store overlap information for particles of higher iOrder number. For example, if particles 3 and 5 are overlapping, only particle 3 stores the information. As a result, only particle 3 in the .demdiag file will contain information on the overlap with particle 5. It is up to the user to reconstruct the overlap data for particle 5 due to particle 3 (which will be just the negative of the force components for particle 3 due to particle 5, by Newton's 3$^{\text{rd}}$ law).

5 TRACKING SPHERE ORIENTATIONS

If pkdgrav is compiled with the DEM_TRACK_ORIENT option, the orientations of spherical particles will be tracked explicitly and stored in files for analysis and plotting. If no initial orientations are specified (see bReadOrient parameter below), all particles are assumed to start with orientations aligned with the space axes. Currently particles in aggregates or stuck to walls will not have orientations updated, and runs using SLIDING_PATCH will not do the update properly (i.e., the orientations will not account correctly for the rotating frame). Orientations are output to binary ".ori" files at both the full output (double precision) and reduced output (single precision) frequency. The utilities ori2ort and ort2ori are provided to convert between binary (.ori) and text (.ort) representations. The .ort files consist of 1 particle per line in the following format:

$$p_{1,x} \quad p_{1,y} \quad p_{1,z} \quad p_{2,x} \quad p_{2,y} \quad p_{2,z}$$

where $\hat{\mathbf{p}}_i = (p_{i,x}, p_{i,y}, p_{i,z})$ is principal axis i of the orientation matrix, a unit vector. Only $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{p}}_2$ are stored because $\hat{\mathbf{p}}_3 = \hat{\mathbf{p}}_1 \times \hat{\mathbf{p}}_2$ by virtue of the orientation matrix being orthonormal. It might be possible to omit one more element, but the added overhead of reconstruction does not make this worthwhile.

Because the particles have spherical symmetry, the Euler equations[1] are particularly simple to solve, with spin vectors updated during leapfrog kicks (as they were before this new feature) and orientations updated during drifts (with a call to dem.c:demUpdateOrient()). The torques that are used to update the spins are computed as normal in the DEM routines along with other forces.

[1]http://www.astro.umd.edu/~dcr/reprints/richardson_icarus115,320.pdf

5.1 Initial Conditions and Restarts

If the `bReadOrient` parameter has a non-zero value, `pkdgrav` will read the `.ori` file associated with `achInFile` on start-up (e.g., if the input file is `ss.12345`, `pkdgrav` will look for `ss.12345.ori`). This file must be in standard format (not reduced). This can be used to give the particles a particular orientation at the start (see the format above), or for restarts (Section 3).

5.2 Drawing Sphere Orientations

If invoked with the `-o` option, `ssdraw` will read the orientation (`.ori`) files associated with each `ss` file and apply the appropriately rotated `StyleOrient` texture (in `povray.inc`). This only works for `POV-Ray` output. A few patterns are provided in the template `povray.inc` file (located in the `pkdgrav etc` folder), though currently only 1 pattern can be chosen at a time, which is applied to all particles. Transparency specified in the pattern pigment can be used to bring out the underlying particle color. To make a movie with particle orientations, use `mkmov.py -o`.

5.3 Future Updates

1. Add support for `AGGS`, sticky walls, and `SLIDING_PATCH`.
   - For `AGGS`, the particle orientation could be transformed to the aggregate body frame and frozen in until the inertia tensor changes (due to adding or removing of particles). Transform back to the space frame for output.
   - For sticky walls, the particle orientation for reactive walls could be treated the same as for `AGGS`, while for non-reactive walls the needed transformation can be computed within `ssdraw`.
   - For `SLIDING_PATCH`, it is necessary to account for the rotating frame. Code to do that for the particle spins is already in `pkdgrav`; that code could be leveraged to also update the particle orientations, in principle.

6 OTHER SSDEM NOTES

6.1 Softening

In its original incarnation, `pkdgrav` used softening to prevent interparticle forces from getting arbitrarily large during a close encounter. With `COLLISIONS`, the softening radius is set equal to the particle radius, largely because the two concepts are considered mutually exclusive so the same particle data structure element could be used for both (see the `RADIUS` macro in `collision.h`). In `HSDEM`, this works well because particles are not supposed to interpenetrate. But in `SSDEM`, particles are allowed to interpenetrate, and this determines the restoring and friction forces. This means that in simulations where particles experience self-gravity (say a self-gravitating rubble pile), gravity forces will be softened when particles overlap. In practice this should be a small effect, since overlaps are meant to be kept below

1% of the smallest particle radius, but if this condition is relaxed, or extreme fidelity is needed for some reason, the softening would need to be taken into account (and may necessitate a new scheme, such as making the particle radius twice the softening radius, etc.).