`ss_core` Walls Documentation

Last Update: 2/13/13

Author: 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

Walls are infinite-mass barriers that can interact with `pkdgrav` particles (but are not affected by particles). Their intended use is to provide confinement for particles in granular dynamics simulations, and optionally to provide an energy input (from oscillation, rotation, etc.). As of this writing, both hard-sphere and soft-sphere wall interactions are supported; the geometry specifications are the same in both cases (see below). The `ssdraw` utility knows about walls and can output appropriate `POV-Ray` source for ray tracing. For more information about the implementation of walls in `pkdgrav`, including mathematical derivations, see Richardson et al. (2011).

2 SPECIFYING WALL PARAMETERS

The parameters that describe the wall geometry are read in by both `pkdgrav` and `ssdraw` from a single text file, the "walls file." The name of the file is specified as `achWallsFile` for `pkdgrav` and as "Wall data file" for `ssdraw`. The specific wall geometry options are detailed in the next section. In addition, `pkdgrav` recognizes two walls-specific parameters: `bWallsEdgeDetect`, which if non-zero instructs the code to check for contact with wall edges (i.e., the one-dimensional edges that border the two-dimensional surfaces of finite walls, viz. either straight-line segments or rings; the calculations can sometimes be expensive, especially for rings—they require solving a quartic equation); and `bWallsSolveQuartic`, which if non-zero instructs the code to include higher-order terms when predicting collisions with particles stuck on rotating cylinders. Also, `ssdraw` accepts "Wall time offset" as a parameter which can be useful to adjust the position of moving walls when drawing. Finally, `ssdraw` accepts the command-line argument "-w", which, if specified, instructs the code to only draw the walls (in which case it is not required to provide a particle data file, but if one or more are in fact provided, they are used as time indicators for the purpose of drawing multiple frames of moving walls).

3 WALLS FILE FORMAT

The walls file is a simple text file consisting of human-readable tokens and data values (real numbers, vectors, or other tokens). The following sections describe each token, the value(s) that can be associated with that token (if any), and the default value(s) assigned if the token is not specified. Note that "loose" formatting is supported: any whitespace is sufficient for demarking tokens and values; redundant whitespace is ignored. Both commas (",") and equals signs ("=") count as whitespace, as do regular spaces, tabs, and new lines. Anything trailing a "#" or "!" is ignored as a comment. So, the user is encouraged to use blank lines, indentation, delimiter characters, and comments to make the walls file as easy to understand (and edit) as possible. Note the walls file is read sequentially from the beginning; any token that changes the current state (i.e., "defaults" or "wall") only applies to subsequent data read in. By default, units are in `pkdgrav` units (i.e., lengths in AU, masses in $M_\odot$, and times in years/2π)—these can be overridden via the "lengthunit," "massunit," and "timeunit" tokens, respectively).

3.1 General Tokens

These tokens affect the overall behavior of the walls file parser. The tokens are not specific to a particular wall.

3.1.1 time [value]

Sets the current time to [value]. Default: 0. Currently ignored. Must be positive.

3.1.2 lengthunit [value]

Sets the current length unit to [value]. Default: 1. Multiplies all subsequent length values read in by [value]. Also used to scale speed values. Must be positive.

3.1.3 massunit [value]

Sets the current mass unit to [value]. Default: 1. Multiplies all subsequent mass values read in by [value]. Must be positive.

3.1.4 timeunit [value]

Sets the current time unit to [value]. Default: 1. Multiplies all subsequent time values read in by [value]. Also used to scale speed and frequency values. Must be positive.

3.1.5 defaults

Indicates that all subsequent token-value(s) sets change the defaults for those tokens to the specified value(s), until the next “wall” token is encountered.

3.1.6 wall

Indicates that a new wall is to be created, with default parameter values. All subsequent token-value(s) sets override the defaults for those tokens to the specified value(s) for this wall, until the next “defaults” or “wall” token is encountered.

3.2 Wall-specific Tokens

These tokens are specific to a wall (or to the default values if a “defaults” token is active). For simplicitly in the following, it is assumed that the parameters for a particular wall are being set, rather than the defaults.

3.2.1 type [value]

Sets the wall to type [value]. Allowed values are: plane, triangle, rectangle, disk, cylinder-infinite, cylinder-finite, and shell. Default: plane. Each type has a specific set of valid parameters that can be set/overridden—see Section 4.

3.2.2 origin [vector]

Sets the wall origin to [vector] (a vector consists of 3 real numbers separated by whitespace). Default: 0,0,0. The meaning of origin depends on the specific wall geometry—see Section 4.

3.2.3 orient [vector]

Sets the wall orientation to [vector]. Default: 0,0,1. Used for all wall types apart from triangle and rectangle. Automatically renormalized to a unit vector (magnitude must be non-zero).

3.2.4 vertex1 [vector]

Sets the location vector of a first vertex relative to the origin to [vector]. Default: 1,0,0. Used together with “vertex2” in place of “orient” for triangle and rectangle types. Magnitude must be non-zero.

3.2.5 vertex2 [vector]

Sets the location vector of a second vertex relative to the origin to [vector]. Default: 0,1,0. Used together with “vertex1” in place of “orient” for triangle and rectangle types. Magnitude must be non-zero.

3.2.6 velocity [vector]

Sets the velocity of the wall to [vector]. Default: 0,0,0 (zero velocity). After a time t the wall origin will be displaced by $\mathbf{v}t$ due to linear motion, where $\mathbf{v}$ is the velocity vector. Can be combined with oscillatory motion.

3.2.7 osc-ampl [value]

Sets the oscillation amplitude (in length units) to [value]. Default: 0 (no oscillation). After a time t the wall origin will be displaced by $A\sin(\omega t)\hat{\mathbf{o}}$ due to oscillatory motion, where A is the oscillation amplitude, ω is the (angular) oscillation frequency, and $\hat{\mathbf{o}}$ is the oscillation vector. Can be combined with linear motion. Note: `pkdgrav` treats oscillatory motion as a series of linear displacements of smoothly varying magnitude. Negative amplitude allowed (equivalent to flipping the oscillation vector).

3.2.8 osc-freq [value]

Sets the oscillation frequency (in radians per time unit) to [value]. Default: 0 (no oscillation). See “osc-ampl”. Note: negative frequency allowed (equivalent to flipping the oscillation vector).

3.2.9 osc-vec [vector]

Sets the oscillation direction to [vector]. Defaults to the default orientation vector (see “orient”). See “osc-ampl”. Automatically renormalized to a unit vector (magnitude must be non-zero).

3.2.10 radius [value]

Sets the wall radius (if applicable) to [value]. Default: 1. Only used for disk, cylinder, and shell wall types. Must be non-negative.

3.2.11 hole-radius [value]

Sets the wall hole radius (if applicable) to [value]. Default: 0. Only used for the disk wall type (the hole origin is coincident with the disk origin). Must be non-negative and less than the disk radius, or zero if the disk radius is zero.

3.2.12 length [value]

Sets the wall length (if applicable) to [value]. Default: 1. Only used for the finite cylinder wall type. Must be non-negative.

3.2.13 taper [value]

Sets the wall taper (if applicable) to [value]. Default: 0. Only used for the finite cylinder wall type. Must be between 0 (no taper) and 1 (maximum taper) inclusive. The radius at the end of the cylinder (in the direction pointed to by "orient") is reduced by a factor equal to the taper (so the cylinder comes to a point if the taper is 1). Currently only implemented in `pkdgrav` for the soft-sphere model.

3.2.14 open-angle [value]

Sets the wall opening angle (if applicable) to [value], in degrees. Default: 0. Only used for the shell wall type. Must be between 0 (no opening) and 180 degrees (maximum opening) inclusive. A hollow half-shell is specified by an opening angle of 90 degrees.

3.2.15 ang-speed [value]

Sets the wall angular speed (if applicable) to [value]. Default: 0. Only used for the infinite plane, disk, cylinder, and shell wall types. The rotation axis is the wall symmetry axis (i.e., along the orientation vector). Negative angular speed allowed (equivalent to flipping the orientation vector). Note: the only effect of angular speed is to change the relative tangential contact velocity when a particle hits the wall; there is no way currently to visualize the rotation with `ssdraw`.

3.2.16 epsn [value]

Sets the normal coefficient of restitution (ε_n) to [value]. Default: 1 (elastic bounce). Must be less than or equal to 1; 0 means the particle sticks on contact; < 0 means the particle is removed from the simulation ("death wall").

3.2.17 epst [value]

Sets the tangential coefficient of restitution (ε_t) to [value]. Default: 1 (no change to relative tangential velocity on contact). Must be between 1 and -1 (reversal of relative tangential velocity) inclusive. A value of 0 means the relative tangential velocity is set to zero on contact.

3.2.18 k_n [value]

Sets the spring constant for the normal spring (k_n) in the dashpot model of the soft-sphere discrete element method (SSDEM) to [value]. Default: 0 (adopt particle value). Must be greater than or equal to 0. A value of 0 means that the k_n for the impacting particle is used. *Important:* The tangential spring's constant (k_t) is automatically set to $\frac{2}{7}k_n$ whenever k_n is changed. To override this behavior, set the value of k_t explicitly *after* setting the value of k_n. This applies equally to setting default values and specific wall values. Only applicable for simulations using SSDEM.

3.2.19 k_t [value]

Sets the spring constant for the tangential spring (k_t) in the SSDEM dashpot model to [value]. Default: 0 (adopt particle value). Must be greater than or equal to 0. Only applicable for simulations using SSDEM. See the "k_n" entry.

3.2.20 mu_s [value]

Sets the sliding friction to [value]. Default: -1 (adopt particle value). Must be greater than or equal to 0 (or equal to -1 for the default behavior). A value of 0 means no sliding friction is applied. Only applicable for simulations using SSDEM. See the "k_n" entry.

3.2.21 mu_r [value]

Sets the rolling friction to [value]. Default: -1 (adopt particle value). Must be greater than or equal to 0 (or -1). A value of 0 means no rolling friction is applied. Only applicable for simulations using SSDEM. See the "mu_s" entry.

3.2.22 mu_t [value]

Sets the twisting friction to [value]. Default: -1 (adopt particle value). Must be greater than or equal to 0 (or -1). A value of 0 means no twisting friction is applied. Only applicable for simulations using SSDEM. See the "mu_s" entry.

3.2.23 color [value]

Sets the color to [value]. Default: 223 (light gray). Must be between 0 and 255 inclusive. The color scheme can be found at the end of the sample `ssdraw.par` file in `ss_core/etc/`. Only used for visualization.

3.2.24 transparency [value]

Sets the transparency to [value]. Default: 0 (opaque). Must be between 0 and 1 *or* 100 (transparent) inclusive. Values above 1 are treated as a percentage and are divided by 100. Only used for visualization.

3.2.25 mass [value]

Sets the wall mass to [value]. Default: infinity (i.e., by default, walls are not affected by particles; if a value is given here, the wall will feel a reaction force when it encounters one or more particles, but presently this force is only felt in the z direction). Must be non-negative. Note walls do not affect other walls.

4 WALL PARAMETERS

The previous section detailed the allowed wall parameters and how they are set/overridden in the walls file. The following describes each wall geometry (type) in more detail, indicating which parameters are applicable to which geometry, and how they are used. Note that the meanings of velocity, osc-ampl, osc-freq, osc-vec, ang-speed, epsn, epst, k_n, k_t, mu_s, mu_t, mu_r, color, and transparency do not depend on the wall type (and all but ang-speed are applicable to any wall type), so they are omitted from the descriptions below.

4.1 plane

This is an infinite plane. Applicable tokens: origin (any point on the plane), orient (the normal to the plane).

4.2 triangle

This is a finite triangle. Not fully implemented. Applicable tokens: origin (the reference vertex), vertex1 (vector from the reference vertex to another vertex), vertex2 (vector from the reference vertex to the remaining vertex). The lengths of vertex1 and vertex2 determine the overall dimensions of the triangle. The relative orientation (cross product) of vertex1 and vertex2 determines the normal to the triangle. The cross product must not be zero.

4.3 rectangle

This is a finite rectangle with square corners. Applicable tokens: origin (the reference vertex), vertex1 (vector from the reference vertex along one side to another vertex), vertex2 (vector from the reference vertex along the other side to another vertex). The lengths of vertex1 and vertex2 determine the overall dimensions of the rectangle. The relative orientation (cross product) of vertex1 and vertex2 determines the normal to the rectangle. The dot product of vertex1 and vertex2 must be zero (indicating perpendicular sides; general parallelograms are not supported).

4.4 disk

This is a circular disk. Applicable tokens: origin (the center of the disk), orient (the normal to the disk), radius, hole-radius. A radius of zero indicates a single point. The hole radius, if specified, must be smaller than the disk radius (unless both the radius and hole radius are zero).

4.5 cylinder-infinite

This is an infinite cylinder (a straight tube). Applicable tokens: origin (any point on the cylinder center axis), orient (the direction of the cylinder axis), radius. A radius of zero indicates an infinite straight line.

4.6 cylinder-finite

This is a finite, open-ended cylinder, with optional taper (to make a cone or frustum). Applicable tokens: origin (the point on the cylinder center axis midway between either end), orient (the direction of the cylinder axis), radius, length, taper. A radius of zero (with length greater than zero) indicates a finite straight line. A length of zero (with radius greater than zero) indicates a ring. If both the radius and length are zero, this indicates a point. The taper, if specified, is in the orientation direction (i.e., the cylinder narrows in that direction).

4.7 shell

This is a shell (hollow sphere), with optional opening angle. Not fully implemented. Applicable tokens: origin (the center of the sphere), orient (the direction from which any opening originates), radius, open-angle. A radius of zero indicates a point, as does an opening angle of 180 degrees.

5 SAMPLE WALLS FILE

Here is a simple example of a walls file:

```
wall type plane
  transparency 1

wall type disk
  origin -1 0 0.2
  orient 0 0 1 # note this is the default
  radius 0.5

wall type cylinder-finite
  origin -0.5 1 0.5
  radius 0.2
  length 0.8

wall type shell
  origin 0.5 1 0.5
  radius 0.3
  open-angle 90

wall type rectangle
  origin 0.5 0 0.2
  vertex1 -0.6 0.6 0
  vertex2 0.6 0.6 0
```

A rendering of this scene is shown in Fig. 1.

6 DRAWING WALLS

Although `ssdraw` provides limited support for rendering walls with line graphics in its native format, the best approach is to specify `POV-Ray` output ("Particle shape 2") and use `povray`

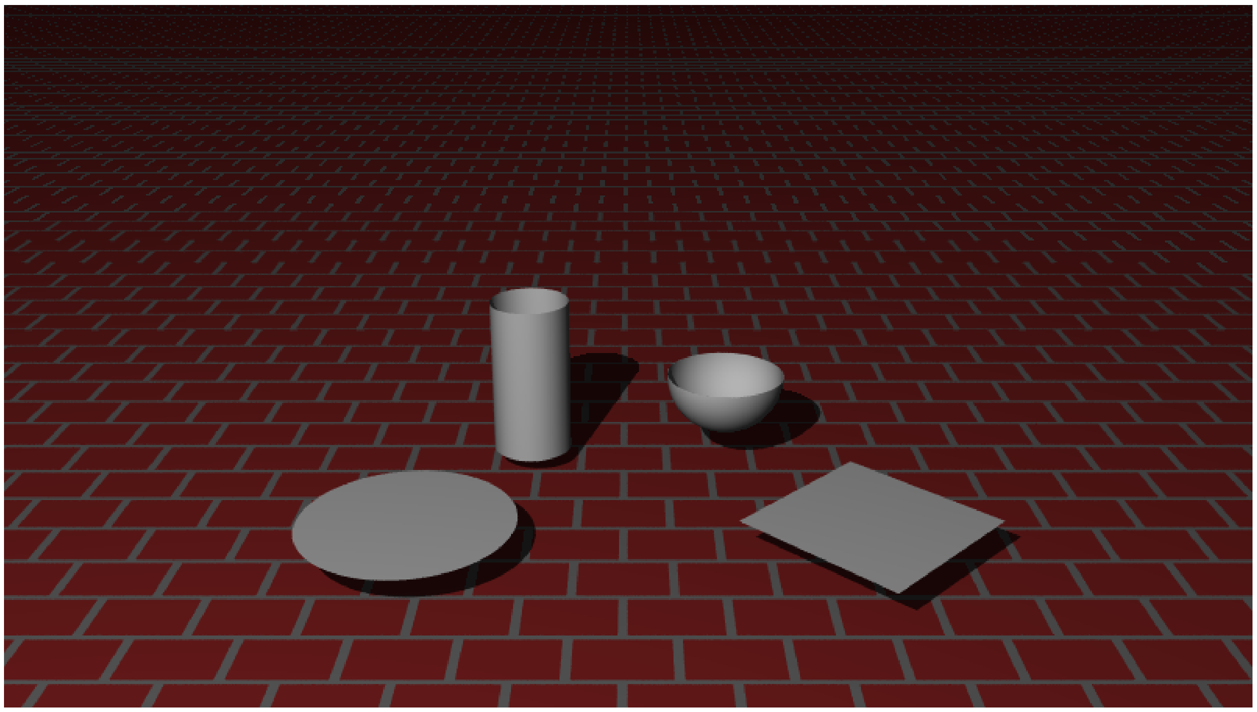

Figure 1: Simple walls scene.

to render the result. As a quick primer, here is the recommended invocation:

```
povray -iFILE -wWIDTH -hHEIGHT +A -D
```

where "FILE" is the name of the `POV-Ray`-format file output by `ssdraw`, and "WIDTH" and "HEIGHT" are the desired dimensions of the output image in pixels (the aspect ratio should match the `ssdraw` "Aspect ratio" parameter). The `POV-Ray` include file specified by `ssdraw`'s "Shape file" should be present in the rendering directory; a sample is provided in `ss_core/etc/povray.inc`.

As always when doing `POV-Ray` rendering, it is important to ensure that all `ssdraw` lengths are mutually consistent, with no values differing by more than a factor of 1 million or so. If small values are present, turn on the `ssdraw` "Renormalize?" option as well (on by default).

The `povray.inc` file contains some simple examples of drawing textures that can be used to liven up a scene. For example, the infinite plane in Fig. 1 was drawn using the "WallBrick" style defined in `povray.inc`. The recommended approach is to make a local copy of `povray.inc` to alter as needed (maybe call it `mypovray.inc` and change the `ssdraw` "Shape file" to match). The defaults are to draw all walls with a "plain" style. To override this, replace the "texture WallPlain" instruction for the corresponding wall type near the end of `mypovray.inc` with the desired style. At present only "WallAgate" and "WallBrick" are provided, but many more options could be added (see the online `POV-Ray` documentation at `povray.org`). Note you can also override the particle drawing style in `povray.inc`.

IMPORTANT: textures applied to walls will only show up if the walls are not opaque (i.e., transparency > 0). This is why the infinite plane in Fig. 1 is assigned a transparency of 1. Experimentation will be required for best results.