PARMEC USER MANUAL

June 1, 2018

Contents

Chapter 1

Installation

Clone parmec sources from GitHub:

```
git clone https://github.com/tkoziara/parmec
```

Enter parmec directory:

```
cd parmec
```

Edit Config.mak file variables:

```
# C++ compiler (ISPC is assumed to be in the PATH; http://ispc.github.io)
CXX=g++

# Python paths
PYTHONINC=-I/usr/include/python2.7
PYTHONLIB=-L/usr/lib -lpython2.7

# HDF5 paths
HDF5INC=-I/usr/include
HDF5LIB=-L/usr/lib -lhdf5 -lhdf5_hl

# Debug version
DEBUG=no
```

Compile sources:

```
make
```

Parmec executable files are:

```
parmec4 (single precision)
parmec8 (double precision)
```

Parmec library files are:

```
libparmec4.a, parmec4.h (single precision library, header)
libparmec8.a, paremc8.h (double precision library, header)
```

To update parmec type:

```
make clean
git pull
make
```

Chapter 2

Running

PARMEC is a command line program. Typical usage:

1. Include parmec directory into your PATH variable.
2. Create a directory where your input file and output files will be stored (e.g. *mkdir test*).
3. Edit your Python input file in this directory (e.g. *test.py*); Chapter 4 documents all input commands.
4. Run PARMEC (e.g. *parmec4 path/to/test/test.py*, or *parmec8 path/to/test/test.py*).
5. Time histories can be generated during analysis using the HISTORY command; see Section 4.20.
6. Upon termination output file(s) is(are) created in the same directory (e.g. *path/to/test/test.dump*); see Section 4.21 and Chapter 5.
7. The output files can be viewed with OVITO, ParaView, or VisIt, as documented in Section 4.21 and Chapter **??**.

Chapter 3

Implementation status

Individual features of parmec which are not implemented yet are marked as (under development). Relatively complex features that have seen little testing are marked as (experimental). Table 3.1 summarizes current status of automatic contact detection.

	SPHERE	MESH	OBSTACLE
SPHERE	OK	N/A	OK
MESH		N/A	N/A
OBSTACLE			N/A

Table 3.1: Current status of automatic contact detection.

Chapter 4

Input commands

PARMEC input language extends Python. Subroutines related to input processing are listed below. **In all cases below, when an object number is returned, indexing starts at 0 and increments on each call**.

4.1 ARGV

List command line arguments.

list = ARGV (| nonparmec)

- **list** - Python list (possibly empty) of command line arguments
- **nonparmec** - optional boolean flag enabling filtering out parmec arguments; default: True

4.2 RESET

Erase all data.

RESET ()

4.3 TSERIES

Create time series: a linear spline based on series of 2-points.

tmsnum = TSERIES (points)

- **tmsnum** - time series number
- **points** - a constant *v0*, or a list *[t0, v0, t1, v1, ...]* or *[[t0,v0], [t1,v1], ...]* or *[(t0,v0), (t1,v1), ...]* of points (where $ti < tj$, when $i < j$), or a path to a file storing pairs of times and values in format:

```
# comment 1 ...
# comment 2 ...
t0 v0
t1 v1
# comment 3 ...
```

```
t2 v2
...
```

4.4 MATERIAL

Create material.

matnum = MATERIAL (density, young, poisson)

- **matnum** - material number
- **density** - mass density
- **young** - Young modulus
- **poisson** - Poisson ratio

4.5 SPHERE

Create a spherical particle.

parnum = SPHERE (center, radius, material, color)

- **parnum** - particle number
- **center** - tuple (x, y, z) defining the center
- **radius** - radius
- **material** - material number
- **color** - positive integer surface color

4.6 MESH

Create a meshed particle.

parnum = MESH (nodes, elements, material, colors)

- **parnum** - particle number
- **nodes** - list of nodes: [*x0, y0, z0, x1, y1, z1, ...*]
- **elements** - list of elements: [*e1, n1, n2, ..., ne1, me1, e2, n1, n2, ..., ne2, me2, ...*], where *e1* is the number of nodes of the first element, *n1, n2, ..., ne1* enumerate the element nodes, and *me1* is the material number. Similarly for the second and all remaining elements. Supported numbers of nodes per element are *4, 5, 6, and 8* for respectively *tetrahedron, pyramid, wedge, and hexahedron*, cf. Figure 4.1.
- **material** - material number
- **colors** - list of positive integer face colors: [*gcolor, f1, n1, n2, ..., nf1, c1, f2, n1, n2, ..., nf2, c2, ...*], where *gcolor* is the global color for all not specified faces, *f1* is the number of nodes in the first specified face, *n1, n2, ..., nf1* enumerate the face nodes, and *c1* is the surface color of that face. Similarly for the second and all remaining faces. If only the global color is required, it can be passed as [*gcolor*] or as *gcolor* alone.

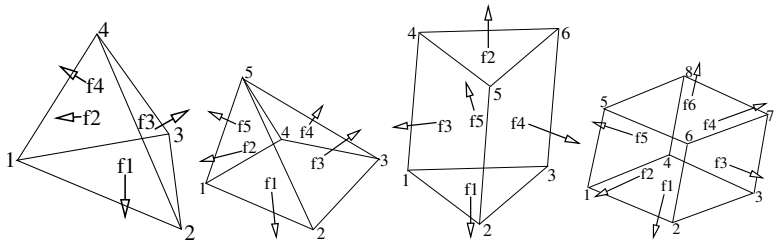

Figure 4.1: Mesh element types in Parmec.

4.7 ANALYTICAL

Create an analytical particle. Analytical particles have no shapes and are not involved in contact.

parnum = ANALYTICAL (| inertia, mass, rotation, position, material, particle)

Note, that all parameters are optional.

- **parnum** - particle number
- **inertia** - inertia tensor passed as a list [*Ixx, Iyy, Izz, Ixy, Ixz, Iyz*]; optional, if **particle** parameter is used; default [*1, 1, 1, 0, 0, 0*]
- **mass** - scalar mass; optional, if **particle** parameter is used; default 1
- **rotation** - optional orientation matrix passed as a list [*e1x, e1y, e1z, e2x, e2y, e2z, e3x, e3y, e3z*], where vectors *e1*, *e2*, *e3* are orthonormal; default [*1, 0, 0, 0, 1, 0, 0, 0, 1*]
- **position** - optional position vector passed as a tuple (*x, y, z*); default (*0, 0, 0*)
- **material** - material number; default *0*
- **particle** - optional; if specified, an existing particle is converted into an analytical particle; its properties are inherited or overwritten, depending on whether any of the **inertia**, **mass**, **rotation**, **position** parameters are used; if initially specified, particle shape is inherited and its animated motion is included into the results

4.8 OBSTACLE

Create an obstacle.

OBSTACLE (triangles, color | point, linear, angular)

- **triangles** - list of triangle tuples [(*t1x1, t1y1, t1z1, t1x2, t1y2, t1z2, t1x3, t1y3, t1z3), (t2x1, t2y1, t2z1, t2x2, t2y2, t2z2, t2x3, t2y3, t2z3), ...*] defining the obstacle
- **color** - positive integer surface color or a list [*color1, color2, ...*] of colors for each individual triangle
- **point** - spatial reference point
- **linear** - linear velocity history callback: $(v_x, v_y, v_z) =$ **linear** (t)
- **angular** - spatial angular velocity history callback: $(\omega_x, \omega_y, \omega_z) =$ **angular** (t)

4.9 SPRING

Create a translational spring constraint. The applied force formula reads

$$\text{force}\,(t) = \text{direction}\,(t) \cdot [\text{spring}\,(\text{stroke}\,(t)) + \text{dashpot}\,(\text{velocity}\,(t)) \cdot |\text{sign}\,(\text{spring}\,(\text{stroke}\,(t)))|]$$

where

$$\text{direction}\,(t) = \begin{cases} \text{d1}\,(t) = (\text{point2}\,(t) - \text{point1}\,(t)) \,/\, |\text{point2}\,(t) - \text{point1}\,(t)| \textbf{ if } \text{geom2} = \text{point2} \\ \text{normal2}\,(t) \textbf{ if } \text{ geom2 } = [\text{point2, normal2}] \\ \text{constant } \textbf{direction} = (d_x, d_y, d_z) \\ \text{d1}\,(t) - \text{d1}\,(t) \cdot (d_x, d_y, d_z) \text{ if } \textbf{planar} \text{ is enabled} \end{cases}$$

$$\text{stroke0} = \text{direction}\,(0) \cdot [\text{point2}\,(0) - \text{point1}\,(0)]$$

$$\text{stroke}\,(t) = \begin{cases} \text{direction}\,(t) \cdot [\text{point2}\,(t) - \text{point1}\,(t)] - \text{stroke0} \textbf{ if } \text{geom2} = \text{point2} \\ \text{normal2}\,(t) \cdot [\text{point1}\,(t) - \text{point2}\,(t)] \textbf{ if } \text{ geom2 } = [\text{point2, normal2}] \end{cases}$$

$$\text{velocity}\,(t) = \text{direction}\,(t) \cdot \frac{d}{dt}\,[\text{point2}\,(t) - \text{point1}\,(t)]$$

$$\text{sign}\,(x) = \begin{cases} -1 \text{ if } x < 0 \\ 0 \text{ if } x = 0 \\ 1 \text{ if } x > 0 \end{cases}$$

The spring (stroke) and dashpot (velocity) relationships are defined by means of lookup tables; force (t) is applied at point2 (t), and $-$force (t) is applied at point1 (t); dashpot force is not applied when spring force is zero.

sprnum = SPRING (part1, point1, part2, geom2, spring | dashpot, direction, planar, unload, ylim, inactive, offset, friction, kskn) (experimental)

- **sprnum** - spring number
- **part1** - first particle number
- **point1** - tuple (x, y, z) defining a point moving with the first particle
- **part2** - second particle number; -1 can be used to indicate a single-particle constraint
- **geom2** - tuple (x, y, z) defining a second point, either moving with the second particle, or a spatial point; alternatively a list storing a point and a normal $[(p_x, p_y, p_z),\ (n_x, n_y, n_z)]$ defining a referential plane, moving with the second particle or spatially fixed; when a plane is defined the spring direction and stroke are calculated from a projection of **point1** onto this plane: in this case the input arguments **direction** and **planar** are ignored
- **spring** - spring force lookup table $[\text{stroke}_1, \text{force}_1, \text{stroke}_2, \text{force}_2, ..., \text{stroke}_n, \text{force}_n]$; used for both loading and unloading when the **unload** table and the **yield** limits are not given
- **dashpot** - optional dashpot force lookup table $[\text{velocity}_1, \text{force}_1, \text{velocity}_2, \text{force}_2, ..., \text{velocity}_m, \text{force}_m]$ or a critical damping ratio from interval $[0, +\infty)$; default: $[-\infty, 0, +\infty, 0]$
- **direction** - optional constant direction (d_x, d_y, d_z)
- **planar** - optional planar spring flag; when 'ON' spring direction

  $$(\text{point2}\,(t) - \text{point1}\,(t)) \,/\, |\text{point2}\,(t) - \text{point1}\,(t)|$$

  is projected onto a plane orthogonal to (d_x, d_y, d_z); default: 'OFF'

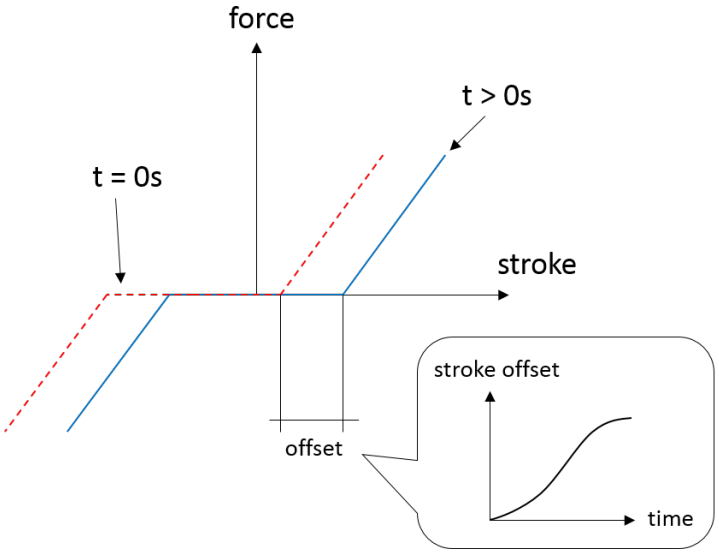

Figure 4.2: The concept of time dependent spring stroke offset.

- **unload** - optional spring unloading lookup table $[\text{stroke}_1, \text{force}_1, \text{stroke}_2, \text{force}_2, ..., \text{stroke}_n, \text{force}_n]$; must be monotonically increasing; default: unspecified
- **ylim** - optional tuple (f_{yc}, f_{yt}) defining the compression, $f_{yc} < 0$, and tension, $f_{yt} > 0$, yield limits; the unloading curve begins to be used once either of these limits is crossed; default: (0, 0)
- **inactive** - optional boolean flag: if *True* create an inactive spring, that can be activated by the UNSPRING command; default: *False*
- **offset** - optional TSERIES number representing a time dependent adjustment applied to all stroke values of the spring curve (dashpot unchanged) as in Figure 4.2; **offset** is applied only prior to yielding for springs with **ylim** and **unload** specified; default: unspecified
- **friction** - optional tangential friction coefficient; default: 0.0
- **kskn** - optional ≥ 0 ratio of normal to tangential spring and dashpot parameters used when **friction** > 0; using **kskn** $= 0$ disables tangential springs/dampers and applies frictional force opposing tangential velocity (in this case frictional sticking and stick-slip transition are not modeled); default: 0.0

4.10 TORSION_SPRING (under development)

Create a torsional spring constraint. The applied torque formula reads

$$\begin{aligned}\mathbf{t} = &\left(\text{kalpha}\left(\alpha\right) + \text{dalpha}\left(\dot{\alpha}\right)\right)\mathbf{x}\\ &+ \left(\text{kbeta}\left(\beta\right) + \text{dbeta}\left(\dot{\beta}\right)\right)\mathbf{z}\\ &+ \left(\text{kgamma}\left(\gamma\right) + \text{dgamma}\left(\dot{\gamma}\right)\right)\mathbf{y}\end{aligned}$$

where the Euler angles α, β, γ are calculated from the co-rotated reference directions $\mathbf{z}$ and $\mathbf{x}$ as follows

$$\mathbf{y} = \mathbf{z} \times \mathbf{x}$$

$$\mathbf{A}_i = \mathbf{\Lambda}_i \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}, \quad i = 1, 2$$

$$\mathbf{B} = \mathbf{A}_2 \mathbf{A}_1^T$$

$$\alpha = \text{atan2}(-B_{23}, B_{22})$$

$$\beta = \text{asin}(B_{21})$$

$$\gamma = \text{atan2}\left(-B_{31}, B_{11}\right)$$

and where $\mathbf{\Lambda}_i$ are rigid rotation matrices of particles 1 and 2, and $\mathbf{B}$ is the matrix of relative rotation from the $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ frame co-rotated with particle 1 into the same frame co-rotated with particle 2. $\mathbf{B}$ is then expressed as a product of Euler rotations

$$\mathbf{B} = \mathbf{A}_\alpha \mathbf{A}_\beta \mathbf{A}_\gamma$$

where γ rotates about $\mathbf{y}$, β rotates about $\mathbf{z}$, and α rotates about $\mathbf{x}$ ($\mathbf{A}_\alpha$, $\mathbf{A}_\beta$, $\mathbf{A}_\gamma$ are the corresponding rotation matrices).

trsnum = TORSION_SPRING (part1, part2, zdir, xdir | kalpha, kbeta, kgamma, dalpha, dbeta, dgamma)

- **trsnum** - torsion spring number
- **part1** - first particle number
- **part2** - second particle number; -1 can be used to indicate a single-particle constraint
- **zdir**, **xdir** - reference direction tuples, $\mathbf{z} = (x_1, y_1, z_1)$ and $\mathbf{x} = (x_2, y_2, z_2)$, co-rotated with the two particles, and used to determine the Euler angles alpha, beta, gamma as described above; these directions must be orthogonal
- **kalpha**, **kbeta**, **kgamma** - spring torque lookup tables $[\text{angle}_1, \text{torque}_1, \text{angle}_2, \text{torque}_2, ...]$, about the three Euler angles; default: $[-\infty, 0, +\infty, 0]$
- **dalpha**, **dbeta**, **dgamma** - dashpot torque lookup tables $[\text{angvel}_1, \text{torque}_1, \text{angvel}_2, \text{torque}_2, ...]$ or a critical damping ratio from interval $[0, +\infty)$; default: $[-\infty, 0, +\infty, 0]$

4.11 UNSPRING (experimental)

UNSPRING (tsprings, msprings, limits | entity, operator, abs, nsteps, nfreq, unload, activate)

Undoes user defined selection of springs (**msprings**) based on the value of spring entities experienced by a different user defined selection of springs (**tsprings**). Modifications to the spring curves occur during a simulation. Undone springs remain in the simulation but generate zero forces.

- **tsprings** - list of unique spring numbers whose spring entities are assessed against a criteria defined by limits; must be nonempty
- **msprings** - list of unique spring numbers which are to be modified if **tsprings** meet the limits criteria (springs defined in **tsprings** are not modified unless also specified in **msprings**); must be nonempty
- **limits** - tuple of (min, max) **tsprings operator entity** limit values which need to be exceeded for **msprings** to be modified; if either value is *None* then no failure limit is assumed e.g. (*None*, max) only has an upper failure limit; also min $<$ max
- **entity** - scalar spring entity string: (spring stroke) 'STROKE', (spring total force) 'F', (spring force without damping) 'SF', (spring total friction force) 'FF', cf. 4.20 and 4.21; default: 'SF'
- **operator** - collective **tsprings** operator string: 'SUM', 'MIN', 'MAX'; default: 'SUM'
- **abs** - boolean, if *True* then spring forces are converted to absolute values before summation of the spring forces; default: *False*
- **nsteps** - int, number of time steps between calls of UNSPRING; default: 1
- **nfreq** - int, number of **nsteps** for which **tsprings** exceed **limits** before **msprings** are modified; default: 1
- **unload** - Python dictionary (i.e. **unload**[*key*] = *value*), where *key* (int) - unique spring number (must be present in **msprings**) and *value* (int) - time series number (TSERIES) defining the unload spring curve; an unloading curve must originate at zero and increase monotonically; once modification is activated, for each spring in **msprings**, the unloading curve is individually applied with a shift specific to the current displacement; both negative and positive displacement increments decrease total spring forces until zero; the spring force remains zero ever after; dashpot force is zero during unloading; default: instantaneous unloading to zero total force
- **activate** - a list of inactive SPRING numbers that will be activated upon complete unloading of all **msprings**; default: empty

By default, modification of **msprings** is based on the sum of the elastic spring force values across all spring numbers defined in **tsprings**. This is a sum of absolute values if **abs** = *True*. Forces in all **tsprings** must exceed the specific min/max values defined in **limits** for the spring curves to be modified (i.e. spring curve modification is an *and* operation, not *or*). For example:

```
tsprings = (1,2)
msprings = (3,4)
limits = (-1.0, 1.0)
UNSPRING(tsprings, msprings, limits)
```

results in the resultant elastic spring force (SF) being assessed against the (-1.0, 1.0) limits. For the spring curves of springs 3 and 4 to be modified, the sum of the forces of springs 1 and 2 must be outside of the (-1.0,1.0) limits for **nfreq** (=1) number of **nsteps** (=1).

4.12 EQM (experimental)

Calculate equivalent point mass from particle inertia and mass properties. Two particles can be passed for relative motion mass. This subroutine can be used to calculate linear stiffness and damping properties for spring based constraints, e.g. stiffness = acceleration $\cdot$ EQM/leeway and damper = damping ratio $\cdot\, 2 \cdot \sqrt{\text{mass} \cdot \text{stiffness}}$ can be used to define spring and dashpot curves as $[-1, -\text{spring}, 1, \text{spring}]$ and $[-1, -\text{damper}, 1, \text{damper}]$ respectively, with the critical time step equal to $\left(2/\sqrt{\text{stiffness} \cdot \text{mass}}\right) \cdot \left(\sqrt{1 + \text{damper}^2} - \text{damper}\right)$.

mass = EQM (part1, point1 | part2, point2, direction) (experimental)

- **mass** - equivalent point mass
- **part1** - particle number
- **point1** - point coordinates
- **part2** - optional second particle number; default: not specified
- **point2** - optional second particle point coordinates; default: not specified
- **direction** - optional direction of motion; default: not specified

4.13 GRANULAR(under development)

Define surface pairing for the granular contact interaction model. Default parameters, for unspecified pairings, are: spring = 0.0, damper = 1.0, friction = (0.0, 0.0), rolling = 0.0, drilling = 0.0, and kskn = 0.5.

GRANULAR (color1, color2, spring | damper, friction, rolling, drilling, kskn)

- **color1** - first color (positive, or color1 = 0 and color2 = 0 to redefine default parameters for unspecified pairings)
- **color2** - second color (positive, or color1 = 0 and color2 = 0 to redefine default parameters for unspecified pairings)
- **spring** - normal spring constant
- **damper** - optional normal damping ratio; default: 1.0
- **friction** - optional Coulomb's friction coefficient; default: 0.0; tuple (μ_s, μ_d) can be used to specify respectively static and dynamic friction coefficients; (experimental)
- **rolling** - optional rolling friction coefficient; default: 0.0; (under development)
- **drilling** - optional drilling friction coefficient; default: 0.0; (under development)
- **kskn** - optional ratio of normal to tangential spring and dashpot parameters; default: 0.5

4.14 RESTRAIN

Restrain particle motion.

RESTRAIN (parnum | linear, angular)

- **parnum** - particle number
- **linear** - list $[x_1, y_1, z_1]$, $[x_1, y_1, z_1, x_2, y_2, z_2]$, or $[x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3]$ defining directions of restrained linear motion; default: $[0, 0, 0]$
- **angular** - list $[x_1, y_1, z_1]$, $[x_1, y_1, z_1, x_2, y_2, z_2]$, or $[x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3]$ defining directions of restrained spatial rotation; default: $[0, 0, 0]$

4.15 PRESCRIBE

Prescribe particle motion. Prescribed motion overwrites this resulting from dynamics and restraints.

PRESCRIBE (parnum | linear, angular, kind)

- **parnum** - particle number
- **linear** - a tuple (i, j, k) of TSERIES numbers, or a callback: (v_x, v_y, v_z) = **linear** (t), defining linear velocity or acceleration history; default: *not prescribed*
- **angular** - a tuple (i, j, k) of TSERIES numbers, or a callback: $(\omega_x, \omega_y, \omega_z)$ = **angular** (t), defining spatial angular velocity or acceleration history; default: *not prescribed*
- **kind** - string 'vv', 'va', 'av', or 'aa' indicating interpretation of respectively **linear** and **angular** time histories as either velocity or acceleration; default: 'vv'

4.16 VELOCITY

Set particle velocity.

VELOCITY (parnum | linear, angular)

- **parnum** - particle number
- **linear** - linear velocity tuple (v_x, v_y, v_z); default: $(0, 0, 0)$ at $t = 0$
- **angular** - angular velocity tuple $(\omega_x, \omega_y, \omega_z)$; default: $(0, 0, 0)$ at $t = 0$

4.17 GRAVITY

Set gravity.

GRAVITY (gx, gy, gz)

- **gx** - constant x float number, or callback $\mathbf{gx}(t)$, or TSERIES number
- **gy** - constant y float number, or callback $\mathbf{gy}(t)$, or TSERIES number
- **gz** - constant z float number, or callback $\mathbf{gz}(t)$, or TSERIES number

4.18 DAMPING

Set global damping, applied as

$$\text{force} = -m \begin{bmatrix} -d_{vx} v_x \\ -d_{vy} v_y \\ -d_{vz} v_z \end{bmatrix}, \ \text{torque} = -\mathbf{\Lambda J \Lambda}^T \begin{bmatrix} -d_{\omega x} \omega_x \\ -d_{\omega y} \omega_y \\ -d_{\omega z} \omega_z \end{bmatrix}$$

where m is scalar mass, v is linear velocity, $\mathbf{\Lambda}$ is the rotation matrix, $\mathbf{J}$ is the referential inertia matrix, and ω is spatial angular velocity.

DAMPING (linear, angular)

- **linear** - linear damping curve callback (d_{vx}, d_{vy}, d_{vz}) = **linear** (t), or a tuple (i, j, k) of TSERIES numbers
- **angular** - angular damping curve callback $(d_{\omega x}, d_{\omega y}, d_{\omega z})$ = **angular** (t), or a tuple (i, j, k) of TSERIES numbers

4.19 CRITICAL

Estimate critical time step. By default this routine returns an estimate in the area of a practical stable step. When run with the optional parameter perspring = n it will return a list of lowest estimates for n individual springs. Similarly, when run with the optional parameter perparticle = n it will return a list of lowest estimates for n individual particles. The actual stable time step may be a factor of $[0.1, 10]$ away of the returned step (estimated), depending on the degree of nonlinearity of the system. See also.

h = CRITICAL (| perspring, perparticle)

- **h** - critical time step (when run without parameters), or a list $[(h_1, i_1, \omega_1, \xi_1), ..., (h_n, i_n, \omega_n, \xi_n)]$ when **perspring** = n or **perparticle** = n is used, where h_k is the per-spring/particle critical time step estimate, i_k is the spring/particle index, ω_k is the maximum spring/particle circular frequency, and ξ_k is the maximum spring/particle damping ratio; when both parameters **perspring** = n and **perparticle** = m are used, a tuple of two corresponding lists is returned $([(h_1, i_1, \omega_1, \xi_1), ..., (h_n, i_n, \omega_n, \xi_n)], [(h_1, i_1, \omega_1, \xi_1), ..., (h_m, i_m, \omega_m, \xi_m)])$
- **perspring** - optional integer n indicating the number of lowest per-spring critical time step estimates; default: undefined
- **perparticle** - optional integer n indicating the number of lowest per-particle critical time step estimates; default: undefined

4.20 HISTORY

Before running a simulation, request time history output; or read history from an existing output file.

list = HISTORY (entity | source, point, h5file, h5last)

- **list** - output time history list (empty upon initial request, populated during simulation)
- **entity** - entity name; global entities: (output time) 'TIME'; particle entities: (position) 'PX', 'PY', 'PZ', '|P|', (displacement) 'DX', 'DY', 'DZ', '|D|', (linear velocity) 'VX', 'VY', 'VZ', '|V|', (angular velocity) 'OX', 'OY', 'OZ', '|O|', (body force) 'FX', 'FY', 'FZ', '|F|', (body torque) 'TX', 'TY', 'TZ', '|T|'; spring entities: (spring length) 'LENGTH', (spring stroke) 'STROKE', (spring total force) 'F', (spring force without damping) 'SF', (spring total friction force) 'FF', (spring state) 'SS', cf. Sec 4.21 for description;
- **source** - particle number i, or a list of particle numbers [i, j, ...], or a spatial sphere defined as tuple (x, y, z, r) (under development), or a spatial box defined as tuple $(x_{\min}, y_{\min}, z_{\min}, x_{\max}, y_{\max}, z_{\max})$ (under development); in case of a list of particle numbers the output entity is averaged over the set of particles; in case of a spatial sphere or box the output entity is averaged over the set of particles passing through it (under development); default: 0 (useful when entity is 'TIME'); spring number or a list of numbers can be used as a source in case of spring entities
- **point** - optional referential point used in case of a single particle source; default: particle mass centre

- **h5file** (experimental) - optional *.h5 file storing existing results; in this case the history is retrieved from this file (if found) or an error message is issued; an appropriate output file needs to be picked depending on the **entity**, cf. Sec 4.21; the output **list** is not populated until **h5last** = *True*; default: not specified
- **h5last** (experimental) - optional boolean flag marking a last call to HISTORY for which the **h5file** argument is used; for faster reading all such histories are populated once HISTORY(..., **h5file** = ..., **h5last** = *True*) is called; default: *False*

4.21 OUTPUT

Before running a simulation, define scalar and/or vector entities included into the output file(s). PARMEC outputs:

- *0.dump (under development) files for spherical particles **not** specified as **a subset** in the OUTPUT command
- *1.dump, *2.dump, ... (under development) files for spherical particles specified as subsets, where numbers 1, 2, ... match consecutive OUTPUT calls
- *0.vtk.* **and/or** (*0.h5, *0.xmf) **and/or** (*0.med) (experimental) files for obstacles and mesh based particles **not** specified as **a subset** in the OUTPUT command
- *1.vtk.*, *2.vtk.*, ... **and/or** (*1.h5, *1.xmf, *2.h5, *2.xmf, ...) **and/or** (*1.med, *2.med...) (experimental) files for mesh based particles specified as subsets, where numbers 1, 2, ... match consecutive OUTPUT calls
- *0rb.vtk.* **and/or** (*0rb.h5, *0rb.xmf) **and/or** (*0rb.med) (under development) files for rigid body data of particles **not** specified as **a subset** in the OUTPUT command
- *1rb.vtk.*, *2rb.vtk.*, ... **and/or** (*1rb.h5, *1rb.xmf, *2rb.h5, *2rb.xmf, ...) **and/or** (*1rb.med, *2rb.med, ...) (under development) files for rigid body data of particles specified as subsets, where numbers 1, 2, ... match consecutive OUTPUT calls
- *0cd.vtk.* **and/or** (*0cd.h5, *0cd.xmf) **and/or** (*0cd.med) (under development) files for contact data including particles **not** specified as **a subset** in the OUTPUT command
- *1cd.vtk.*, *2cd.vtk.*, ... **and/or** (*1cd.h5, *1cd.xmf, *2cd.h5, *2cd.xmf, ...) **and/or** (*1cd.med, *2cd.med, ...) (under development) files for contact data including particles specified as subsets, where numbers 1, 2, ... match consecutive OUTPUT calls
- *0sd.vtk.* **and/or** (*0sd.h5, *0sd.xmf) **and/or** (*0sd.med) (under development) files for spring data including particles **not** specified as **a subset** in the OUTPUT command
- *1sd.vtk.*, *2sd.vtk.*, ... **and/or** (*1sd.h5, *1sd.xmf, *2sd.h5, *2sd.xmf, ...) **and/or** (*1sd.med, *2sd.med, ...) (under development) files for spring data including particles specified as subsets, where numbers 1, 2, ... match consecutive OUTPUT calls

OUTPUT (| entities, subset, mode, format)

- **entities** - list of output entities; default: ['NUMBER', 'COLOR', 'DISPL','LENGTH', 'ORIENT', 'ORIENT1', 'ORIENT2', 'ORIENT3', 'LINVEL', 'ANGVEL', 'FORCE', 'TORQUE', 'F', 'FN', 'FT', 'SF', 'FF', 'SS', 'AREA', 'PAIR'] where:
  - 'NUMBER' - scalar field of particle numbers (modes: 'SPH', 'MESH', 'RB'), or scalar field of spring numbers (modes: 'SD')

  - 'COLOR' - scalar field of surface colors (modes: 'SPH', 'MESH'), or 2-component vector field of contact surface colors (modes: 'CD')
  - 'DISPL' - 3-component vector field of displacements (modes: 'SPH', 'MESH', 'RB'), or scalar field of contact depths (modes: 'CD'), or scalar field of spring strokes (modes: 'SD')
  - 'LENGTH' - scalar field of spring lengths (modes: 'SD')
  - 'ORIENT' - 9-component tensor field representing rigid rotation matrix (modes: 'RB'), or 3-component vector field of spring orientations (modes: 'SD')
  - 'ORIENT1', 'ORIENT2', 'ORIENT3' - three 3-component vector fields representing columns of rigid rotation matrix (orientation vectors) (modes: 'RB')
  - 'LINVEL' - 3-component vector field of linear velocity (modes: 'SPH', 'MESH', 'RB')
  - 'ANGVEL' - 3-component vector field of (spatially constant) angular velocity (modes: 'SPH', 'MESH', 'RB')
  - 'FORCE' - 3-component vector field of (spatially constant) total body force (modes: 'SPH', 'MESH', 'RB')
  - 'TORQUE' - 3-component vector field of (spatially constant) total body torque (modes: 'SPH', 'MESH', 'RB')
  - 'F' - 3-component vector field of total contact forces (modes: 'CD'), or scalar field of total spring forces (modes: 'SD')
  - 'FN' - 3-component vector field of normal contact forces (modes: 'CD')
  - 'FT' - 3-component vector field of tangential contact forces (modes: 'CD')
  - 'SF' - scalar field of spring force magnitude, without dashpot contribution (modes: 'CD', 'SD')
  - 'FF' - scalar field of total friction force magnitude (modes: 'CD', 'SD')
  - 'SS' - scalar field of spring states, where -3.0 denotes a regular spring (always active), -2.0 denotes an active spring, -1.0 denotes a deactivated spring (zero force), and values ≥ 0 denote a spring currently being unloaded (the number itself denotes TSERIES, Sec 4.3, used as an unloading curve) (modes: 'SD')
  - 'AREA' - scalar field of contact area (modes: 'CD')
  - 'PAIR' - 2-component vector field of particle pair numbers (modes: 'CD', 'SD')

- **subset** - optional particle number *i*, or a list of particle numbers [*i, j, ...*], to which this specification is narrowed down

- **mode** - optional output mode or list of output modes: 'SPH' for sphere output, 'MESH' for mesh output, 'RB' for rigid body output, 'CD' for contact data output, 'SD' for spring data output; default: ['SPH', 'MESH', 'RB', 'CD', 'SD']

- **format** - optional output format, e.g. 'DUMP' (experimental/under development) or 'VTK' or 'XDMF' or 'MED' (experimental/under development), or a list, e.g. ['DUMP', 'VTK', 'XDMF', 'MED'], where 'DUMP' is the text based LAMMPS format (experimental/under development), 'VTK' is the text based legacy VTK format, 'XDMF' is the HDF5/XML based XDMF format, and 'MED' is the binary MED format (experimental/under development); default: 'XDMF'

4.22 DEM

Run DEM simulation.

t = DEM (duration, step | interval, prefix, adaptive)

- **t** - simulation runtime in seconds
- **duration** - simulation duration
- **step** - time step; initial if **adaptive** is used or constant otherwise
- **interval** - output interval (default: time step); tuple $(dt_{\text{files}}, dt_{\text{history}})$ can be used to indicate different output frequencies of output files and time histories, respectively; callback functions or TSERIES numbers can also be used, e.g. $dt_{\text{files}} = \text{dt\_fiels}\,(t)$ and $dt_{\text{history}} = \text{tmsnum}$, prescribing variable interval frequencies, depending on current time;
- **prefix** - output file name prefix (default: input file name without the ".py" extension); Note: **prefix** can only change at time 0.0 or after RESET()
- **adaptive** - adaptive time step reduction factor; zero turns off adaptive time stepping, values > 0.0 and ≤ 1.0 turn it on; default: 0.0 (experimental)

Chapter 5

Output files

Currently Parmec supports the following output file formats:

- For spherical particles:
  - LAMMPS *.dump files (experimental/under development); can be viewed with OVITO
- For mesh based particles, for rigid bodies, for contact points, and for springs:
- *.vtk text files; can be viewed with ParaView, OVITO
- *.xmf binary files; can be viewed with ParaView
- *.med binary files (under development); can be viewed with Gmsh, SALOME

See also the OUTPUT command for details on outputted entities and file kinds.

Status (as of commit 5ab402d)

The *.dump file format is partially supported at this point: it can be used to view animated results for spheres, including some of the output entities. The *.vtk and *.xmf file formats are fairly complete and can be used to view motion of particles as well as time histories of all output fields and entities. The *.med file format currently supports only mesh data (it does not include output for rigid bodies, contact points, and springs).