CHAPTER

NINE

DUMP STYLES

9.1 dump command

9.2 dump vtk command

9.3 dump h5md command

9.4 dump molfile command

9.5 dump netcdf command

9.6 dump image command

9.7 dump movie command

9.8 dump atom/adios command

9.9 dump custom/adios command

9.10 dump cfg/uef command

9.10.1 Syntax

```
dump ID group-ID style N file attribute1 attribute2 ...
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be dumped
- style = *atom* or *atom/adios* or *atom/gz* or *atom/zstd* or *cfg* or *cfg/gz* or *cfg/zstd* or *cfg/uef* or *custom* or *custom/gz* or *custom/zstd* or *custom/adios* or *dcd* or *grid* or *grid/vtk* or *h5md* or *image* or *local* or *local/gz* or *local/zstd* or *molfile* or *movie* or *netcdf* or *netcdf/mpiio* or *vtk* or *xtc* or *xyz* or *xyz/gz* or *xyz/zstd* or *yaml*
- N = dump on timesteps which are multiples of N

- file = name of file to write dump info to
- attribute1,attribute2,… = list of attributes for a particular style

  atom attributes = none
  atom/adios attributes = none, discussed on dump atom/adios page
  atom/gz attributes = none
  atom/zstd attributes = none
  cfg attributes = same as custom attributes, see below
  cfg/gz attributes = same as custom attributes, see below
  cfg/zstd attributes = same as custom attributes, see below
  cfg/uef attributes = same as custom attributes, discussed on dump cfg/uef page
  custom, custom/gz, custom/zstd attributes = see below
  custom/adios attributes = same as custom attributes, discussed on dump custom/adios page
  dcd attributes = none
  h5md attributes = discussed on dump h5md page
  grid attributes = see below
  grid/vtk attributes = see below
  image attributes = discussed on dump image page
  local, local/gz, local/zstd attributes = see below
  molfile attributes = discussed on dump molfile page
  movie attributes = discussed on dump image page
  netcdf attributes = discussed on dump netcdf page
  netcdf/mpiio attributes = discussed on dump netcdf page
  vtk attributes = same as custom attributes, see below, also dump vtk page
  xtc attributes = none
  xyz attributes = none
  xyz/gz attributes = none
  xyz/zstd attributes = none
  yaml attributes = same as custom attributes, see below

- *custom* or *custom/gz* or *custom/zstd* or *cfg* or *cfg/gz* or *cfg/zstd* or *cfg/uef* or *netcdf* or *netcdf/mpiio* or *yaml* attributes:

```
possible attributes = id, mol, proc, procp1, type, element, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu,
                      xsu, ysu, zsu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz, mu,
                      radius, diameter, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz, tqx, tqy, tqz,
                      c_ID, c_ID[I], f_ID, f_ID[I], v_name,
                      i_name, d_name, i2_name[I], d2_name[I]
```

  id = atom ID
  mol = molecule ID
  proc = ID of processor that owns atom
  procp1 = ID+1 of processor that owns atom
  type = atom type
  typelabel = atom type label
  element = name of atom element, as defined by dump_modify command
  mass = atom mass
  x,y,z = unscaled atom coordinates
  xs,ys,zs = scaled atom coordinates
  xu,yu,zu = unwrapped atom coordinates
  xsu,ysu,zsu = scaled unwrapped atom coordinates

```
ix,iy,iz = box image that the atom is in
vx,vy,vz = atom velocities
fx,fy,fz = forces on atoms
q = atom charge
mux,muy,muz = orientation of dipole moment of atom
mu = magnitude of dipole moment of atom
radius,diameter = radius, diameter of spherical particle
omegax,omegay,omegaz = angular velocity of spherical particle
angmomx,angmomy,angmomz = angular momentum of aspherical particle
tqx,tqy,tqz = torque on finite-size particles
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID, I can include wildcard␣
↪(see below)
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID, I can include wildcard (see␣
↪below)
v_name = per-atom vector calculated by an atom-style variable with name
i_name = custom integer vector with name
d_name = custom floating point vector with name
i2_name[I] = Ith column of custom integer array with name, I can include wildcard (see below)
d2_name[I] = Ith column of custom floating point vector with name, I can include wildcard (see␣
↪below)
```

- *local* or *local/gz* or *local/zstd* attributes:

```
possible attributes = index, c_ID, c_ID[I], f_ID, f_ID[I]
  index = enumeration of local values
  c_ID = local vector calculated by a compute with ID
  c_ID[I] = Ith column of local array calculated by a compute with ID, I can include wildcard (see␣
↪below)
  f_ID = local vector calculated by a fix with ID
  f_ID[I] = Ith column of local array calculated by a fix with ID, I can include wildcard (see below)
```

- *grid* or *grid/vtk* attributes:

```
possible attributes = c_ID:gname:dname, c_ID:gname:dname[I], f_ID:gname:dname, f_
↪ID:gname:dname[I]
  gname = name of grid defined by compute or fix
  dname = name of data field defined by compute or fix
  c_ID = per-grid vector calculated by a compute with ID
  c_ID[I] = Ith column of per-grid array calculated by a compute with ID, I can include wildcard␣
↪(see below)
  f_ID = per-grid vector calculated by a fix with ID
  f_ID[I] = Ith column of per-grid array calculated by a fix with ID, I can include wildcard (see␣
↪below)
```

9.10.2 Examples

```
dump myDump all atom 100 dump.lammpstrj
dump myDump all atom/gz 100 dump.atom.gz
dump myDump all atom/zstd 100 dump.atom.zst
dump 2 subgroup atom 50 dump.run.bin
dump 4a all custom 100 dump.myforce.* id type x y vx fx
dump 4a all custom 100 dump.myvel.lammpsbin id type x y z vx vy vz
dump 4b flow custom 100 dump.%.myforce id type c_myF[3] v_ke
dump 4b flow custom 100 dump.%.myforce id type c_myF[*] v_ke
dump 2 inner cfg 10 dump.snap.*.cfg mass type xs ys zs vx vy vz
dump snap all cfg 100 dump.config.*.cfg mass type xs ys zs id type c_Stress[2]
dump 1 all xtc 1000 file.xtc
```

9.10.3 Description

Dump a snapshot of quantities to one or more files once every *N* timesteps in one of several styles. The timesteps on which dump output is written can also be controlled by a variable. See the *dump_modify every* command.

Almost all the styles output per-atom data, i.e. one or more values per atom. The exceptions are as follows. The *local* styles output one or more values per bond (angle, dihedral, improper) or per pair of interacting atoms (force or neighbor interactions). The *grid* styles output one or more values per grid cell, which are produced by other commands which overlay the simulation domain with a regular grid. See the *Howto grid* doc page for details. The *image* style renders a JPG, PNG, or PPM image file of the system for each snapshot, while the *movie* style combines and compresses the series of images into a movie file; both styles are discussed in detail on the *dump image* page.

Only information for atoms in the specified group is dumped. The *dump_modify thresh and region and refresh* commands can also alter what atoms are included. Not all styles support these options; see details on the *dump_modify* doc page.

As described below, the filename determines the kind of output: text or binary or gzipped, one big file or one per timestep, one file for all the processors or multiple smaller files.

Note

Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box. Re-neighbor timesteps will not typically coincide with the timesteps dump snapshots are written. See the *dump_modify pbc* command if you wish to force coordinates to be strictly inside the simulation box.

Note

Unless the *dump_modify sort* option is invoked, the lines of atom or grid information written to dump files (typically one line per atom or grid cell) will be in an indeterminate order for each snapshot. This is even true when running on a single processor, if the *atom_modify sort* option is on, which it is by default. In this case atoms are re-ordered periodically during a simulation, due to spatial sorting. It is also true when running in parallel, because data for a single snapshot is collected from multiple processors, each of which owns a subset of the atoms.

Warning

Without either including atom IDs or using the *dump_modify sort* option, it is impossible for visualization programs (e.g. OVITO or VMD) or analysis tools to assign data in different frames consistently to the same atom. This can lead to incorrect visualizations or results. LAMMPS will print a warning in such cases.

For the *atom*, *custom*, *cfg*, *grid*, and *local* styles, sorting is off by default. For the *dcd*, *grid/vtk*, *xtc*, *xyz*, and *molfile* styles, sorting by atom ID or grid ID is on by default. See the *dump_modify* page for details.

The *style* keyword determines what kind of data is written to the dump file(s) and in what format.

Note that *atom*, *custom*, *dcd*, *xtc*, *xyz*, and *yaml* style dump files can be read directly by VMD, a popular tool for visualizing and analyzing trajectories from atomic and molecular systems. For reading *netcdf* style dump files, the netcdf plugin needs to be recompiled from source using a NetCDF version compatible with the one used by LAMMPS. The bundled plugin binary uses a very old version of NetCDF that is not compatible with LAMMPS.

Likewise the OVITO visualization package, popular for materials modeling, can read the *atom*, *custom*, *local*, *xtc*, *cfg*, *netcdf*, and *xyz* style atom dump files directly. With version 3.8 and above, OVITO can also read and visualize *grid* style dump files with grid cell data, including iso-surface images of the grid cell values.

Note that settings made via the *dump_modify* command can also alter the format of individual values and content of the dump file itself. This includes the precision of values output to text-based dump files which is controlled by the *dump_modify format* command and its options.

Format of native LAMMPS format dump files:

The *atom*, *custom*, *grid*, and *local* styles create files in a simple LAMMPS-specific text format that is mostly self-explanatory when viewing a dump file. Many post-processing tools either included with LAMMPS or third-party tools can read this format, as does the *rerun* command. See tools described on the *Tools* doc page for examples, including Pizza.py.

For all these styles, the dimensions of the simulation box are included in each snapshot. The simulation box in LAMMPS can be defined in one of 3 ways: orthogonal, restricted triclinic, and general triclinic. See the *Howto triclinic* doc page for a detailed description of all 3 options.

For an orthogonal simulation box the box information is formatted as:

```
ITEM: BOX BOUNDS xx yy zz
xlo xhi
ylo yhi
zlo zhi
```

where xlo,xhi are the maximum extents of the simulation box in the x-dimension, and similarly for y and z. The "xx yy zz" terms are six characters that encode the style of boundary for each of the six simulation box boundaries (xlo,xhi; ylo,yhi; and zlo,zhi). Each of the six characters is one of *p* (periodic), *f* (fixed), *s* (shrink wrap), or *m* (shrink wrapped with a minimum value). See the *boundary* command for details.

For a restricted triclinic simulation box, an orthogonal bounding box which encloses the restricted triclinic simulation box is output, along with the three tilt factors (*xy*, *xz*, *yz*) of the triclinic box, formatted as follows:

```
ITEM: BOX BOUNDS xy xz yz xx yy zz
xlo_bound xhi_bound xy
ylo_bound yhi_bound xz
zlo_bound zhi_bound yz
```

The presence of the text "xy xz yz" in the ITEM line indicates that the three tilt factors will be included on each of the three following lines. This bounding box is convenient for many visualization programs. The meaning of the six character flags for "xx yy zz" is the same as above.

Note that the first two numbers on each line are now xlo_bound instead of xlo, etc. because they represent a bounding box. See the *Howto triclinic* page for a geometric description of triclinic boxes, as defined by LAMMPS, simple formulas for how the six bounding box extents (xlo_bound, xhi_bound, etc.) are calculated from the triclinic parameters, and how to transform those parameters to and from other commonly used triclinic representations.

For a general triclinic simulation box, see the "General triclinic" section below for a description of the ITEM: BOX BOUNDS format as well as how per-atom coordinates and per-atom vector quantities are output.

The *atom* and *custom* styles output a "ITEM: NUMBER OF ATOMS" line with the count of atoms in the snapshot. Likewise they output an "ITEM: ATOMS" line which includes column descriptors for the per-atom lines that follow. For example, the descriptors would be "id type xs ys zs" for the default *atom* style, and would be the atom attributes you specify in the dump command for the *custom* style. Each subsequent line will list the data for a single atom.

For style *atom*, atom coordinates are written to the file, along with the atom ID and atom type. By default, atom coords are written in a scaled format (from 0 to 1). That is, an *x* value of 0.25 means the atom is at a location 1/4 of the distance from *xlo* to *xhi* of the box boundaries. The format can be changed to unscaled coords via the *dump_modify* settings. Image flags can also be added for each atom via dump_modify.

Style *custom* allows you to specify a list of atom attributes to be written to the dump file for each atom. Possible attributes are listed above and will appear in the order specified. You cannot specify a quantity that is not defined for a particular simulation—such as *q* for atom style *bond*, since that atom style does not assign charges. Dumps occur at the very end of a timestep, so atom attributes will include effects due to fixes that are applied during the timestep. An explanation of the possible dump custom attributes is given below.

Added in version 22Dec2022.

For style *grid* the dimension of the simulation domain and size of the Nx by Ny by Nz grid that overlays the simulation domain are also output with each snapshot:

```
ITEM: DIMENSION
dim
ITEM: GRID SIZE
nx ny nz
```

The value dim will be 2 or 3 for 2d or 3d simulations. It is included so that post-processing tools like OVITO, which can visualize grid-based quantities know how to draw each grid cell. The grid size will match the input script parameters for grid(s) created by the computes or fixes which are referenced by the the dump command. For 2d simulations (and grids), nz will always be 1.

There will also be an "ITEM: GRID DATA" line which includes column descriptors for the per grid cell data. Each subsequent line (Nx * Ny * Nz lines) will list the data for a single grid cell. If grid cell IDs are included in the output via the *compute property/grid* command, then the IDs will range from 1 to N = Nx*Ny*Nz. The ordering of IDs is with the x index varying fastest, then the y index, and the z index varying slowest.

For style *local*, local output generated by *computes* and *fixes* is used to generate lines of output that is written to the dump file. This local data is typically calculated by each processor based on the atoms it owns, but there may be zero or more entities per atom (e.g., a list of bond distances). An explanation of the possible dump local attributes is given below. Note that by using input from the *compute property/local* command with dump local, it is possible to generate information on bonds, angles, etc. that can be cut and pasted directly into a data file read by the *read_data* command.

Dump files in other popular formats:

Note

This section only discusses file formats relevant to this doc page. The top of this page has links to other dump commands (with their own pages) which write files in additional popular formats.

Style *cfg* has the same command syntax as style *custom* and writes extended CFG format files, as used by the AtomEye visualization package. Since the extended CFG format uses a single snapshot of the system per file, a wildcard "*" must be included in the filename, as discussed below. The list of atom attributes for style *cfg* must begin with either "mass type xs ys zs" or "mass type xsu ysu zsu" since these quantities are needed to write the CFG files in the appropriate format (though the "mass" and "type" fields do not appear explicitly in the file). Any remaining attributes will be stored as "auxiliary properties" in the CFG files. Note that you will typically want to use the *dump_modify element* command with CFG-formatted files, to associate element names with atom types, so that AtomEye can render atoms appropriately. When unwrapped coordinates *xsu*, *ysu*, and *zsu* are requested, the nominal AtomEye periodic cell dimensions are expanded by a large factor UNWRAPEXPAND = 10.0, which ensures atoms that are displayed correctly for up to UNWRAPEXPAND/2 periodic boundary crossings in any direction. Beyond this, AtomEye will rewrap the unwrapped coordinates. The expansion causes the atoms to be drawn farther away from the viewer, but it is easy to zoom the atoms closer, and the interatomic distances are unaffected.

The *dcd* style writes DCD files, a standard atomic trajectory format used by the CHARMM, NAMD, and XPlor molecular dynamics packages. DCD files are binary and thus may not be portable to different machines. The number of atoms per snapshot cannot change with the *dcd* style. The *unwrap* option of the *dump_modify* command allows DCD coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *xtc* style writes XTC files, a compressed trajectory format used by the GROMACS molecular dynamics package, and described here. The precision used in XTC files can be adjusted via the *dump_modify* command. The default value of 1000 means that coordinates are stored to 1/1000 nanometer accuracy. XTC files are portable binary files written in the NFS XDR data format, so that any machine which supports XDR should be able to read them. The number of atoms per snapshot cannot change with the *xtc* style. The *unwrap* option of the *dump_modify* command allows XTC coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *xyz* style writes XYZ files, which is a simple text-based coordinate format that many codes can read. Specifically it has a line with the number of atoms, then a comment line that is usually ignored followed by one line per atom with the atom type and the *x*-, *y*-, and *z*-coordinate of that atom. You can use the *dump_modify element* option to change the output from using the (numerical) atom type to an element name (or some other label). This option will help many visualization programs to guess bonds and colors. You can use the *dump_modify types labels* option to replace numeric atom types with *type labels*.

Added in version 22Dec2022.

The *grid/vtk* style writes VTK files for grid data on a regular rectilinear grid. Its content is conceptually similar to that of the text file produced by the *grid* style, except that it in an XML-based format which visualization programs which support the VTK format can read, e.g. the ParaView tool. For this style, there can only be 1 or 3 per grid cell attributes specified. If it is a single value, it is a scalar quantity. If 3 values are specified it is encoded in the VTK file as a vector quantity (for each grid cell). The filename for this style must include a "*" wildcard character to produce one file per snapshot; see details below.

Added in version 4May2022.

Dump style *yaml* has the same command syntax as style *custom* and writes YAML format files that can be easily parsed by a variety of data processing tools and programming languages. Each timestep will be written as a YAML "document" (i.e., starts with "—" and ends with "…"). The style supports writing one file per timestep through the "*" wildcard but not multi-processor outputs with the "%" token in the filename. In addition to per-atom data, *thermo* data can be included in the *yaml* style dump file using the *dump_modify thermo yes*. The data included in the dump file uses the "thermo" tag and is otherwise identical to data specified by the *thermo_style* command.

Below is an example for a YAML format dump created by the following commands.

```
dump out all yaml 100 dump.yaml id type x y z vx vy vz ix iy iz
dump_modify out time yes units yes thermo yes format 1 %5d format "% 10.6e"
```

The tags "time", "units", and "thermo" are optional and enabled by the dump_modify command. The list under the "box" tag has three lines for orthogonal boxes and four lines for triclinic boxes, where the first three are the box boundaries and the fourth the three tilt factors (*xy*, *xz*, *yz*). The "thermo" data follows the format of the *yaml* thermo style. The "keywords" tag lists the per-atom properties contained in the "data" columns, which contain a list with one line per atom. The keywords may be renamed using the dump_modify command same as for the *custom* dump style.

```
---
creator: LAMMPS
timestep: 0
units: lj
time: 0
natoms: 4000
boundary: [ p, p, p, p, p, p, ]
thermo:
  - keywords: [ Step, Temp, E_pair, E_mol, TotEng, Press, ]
  - data: [ 0, 0, -27093.472213010766, 0, 0, 0, ]
box:
  - [ 0, 16.795961913825074 ]
  - [ 0, 16.795961913825074 ]
  - [ 0, 16.795961913825074 ]
  - [ 0, 0, 0 ]
keywords: [ id, type, x, y, z, vx, vy, vz, ix, iy, iz,  ]
data:
  - [     1 , 1 ,  0.000000e+00 ,  0.000000e+00 ,  0.000000e+00 ,  -1.841579e-01 , -9.710036e-01 , -2.
→934617e+00 , 0 , 0 , 0, ]
  - [     2 , 1 ,  8.397981e-01 ,  8.397981e-01 ,  0.000000e+00 ,  -1.799591e+00 ,  2.127197e+00 ,  2.
→298572e+00 , 0 , 0 , 0, ]
  - [     3 , 1 ,  8.397981e-01 ,  0.000000e+00 ,  8.397981e-01 ,  -1.807682e+00 , -9.585130e-01 ,  1.
→605884e+00 , 0 , 0 , 0, ]

  [...]
...
---
timestep: 100
units: lj
time: 0.5

  [...]

...
```

Frequency of dump output:

Dumps are performed on timesteps that are a multiple of *N* (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of *N*. This behavior can be changed via the *dump_modify first* command, which can also be useful if the dump command is invoked after a minimization ended on an arbitrary timestep.

The value of *N* can be changed between runs by using the *dump_modify every* command (not allowed for *dcd* style). The *dump_modify every* command also allows a variable to be used to determine the sequence of timesteps on which dump files are written. In this mode a dump on the first timestep of a run will also not be written unless the *dump_modify first* command is used.

If you instead want to dump snapshots based on simulation time (in time units of the *units command* command), the *dump_modify every/time* command can be used. This can be useful when the timestep size varies during a simulation run, e.g. by use of the *fix dt/reset* command.

Dump filenames:

The specified dump filename determines how the dump file(s) is written. The default is to write one large text file, which is opened when the dump command is invoked and closed when an *undump* command is used or when LAMMPS exits. For the *dcd* and *xtc* styles, this is a single large binary file.

Many of the styles allow dump filenames to contain either or both of two wildcard characters. If a "*" character appears in the filename, then one file per snapshot is written and the "*" character is replaced with the timestep value. For example, tmp.dump.* becomes tmp.dump.0, tmp.dump.10000, tmp.dump.20000, etc. This option is not available for the *dcd* and *xtc* styles. Note that the *dump_modify pad* command can be used to ensure all timestep numbers are the same length (e.g., 00010), which can make it easier to read a series of dump files in order with some post-processing tools.

If a "%" character appears in the filename, then each of P processors writes a portion of the dump file, and the "%" character is replaced with the processor ID from 0 to $P-1$. For example, tmp.dump.% becomes tmp.dump.0, tmp.dump.1, … tmp.dump.:math:*P-1*, etc. This creates smaller files and can be a fast mode of output on parallel machines that support parallel I/O for output. This option is **not** available for the *dcd*, *xtc*, *xyz*, *grid/vtk*, and *yaml* styles.

By default, *P* is the the number of processors, meaning one file per processor, but *P* can be set to a smaller value via the *nfile* or *fileper* keywords of the *dump_modify* command. These options can be the most efficient way of writing out dump files when running on large numbers of processors.

Note that using the "*" and "%" characters together can produce a large number of small dump files!

Deprecated since version 21Nov2023.

The MPIIO package and the the corresponding "/mpiio" dump styles, except for the unrelated "netcdf/mpiio" style were removed from LAMMPS.

Compression of dump file data:

If the specified filename ends with ".bin" or ".lammpsbin", the dump file (or files, if "*" or "%" is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster. Of course, when post-processing, you will need to convert it back to text format (see the *binary2txt tool*) or write your own code to read the binary file. The format of the binary file can be understood by looking at the `tools/binary2txt.cpp` file. This option is only available for the *atom* and *custom* styles.

If the filename ends with ".gz", the dump file (or files, if "*" or "%" is also used) is written in gzipped format. A gzipped dump file will be about 3× smaller than the text version, but will also take longer to write. This option is not available for the *dcd* and *xtc* styles.

Note that styles that end with *gz* are identical in command syntax to the corresponding styles without "gz", however, they generate compressed files using the zlib library. Thus the filename suffix ".gz" is mandatory. This is an alternative approach to writing compressed files via a pipe, as done by the regular dump styles, which may be required on clusters where the interface to the high-speed network disallows using the fork() library call (which is needed for a pipe). For the remainder of this page, you should thus consider the *atom* and *atom/gz* styles (etc.) to be inter-changeable, with the exception of the required filename suffix.

Similarly, styles that end with *zstd* are identical to the gz styles, but use the Zstd compression library instead and require a ".zst" suffix. See the *dump_modify* page for details on how to control the compression level in both variants.

General triclinic simulation box output for the *atom* and *custom* styles:

As mentioned above, the simulation box can be defined as a general triclinic box, which means that 3 arbitrary box edge vectors **A**, **B**, **C** can be specified. See the *Howto triclinic* doc page for a detailed description of general triclinic boxes.

This option is provided as a convenience for users who may be converting data from solid-state crystallographic representations or from DFT codes for input to LAMMPS. However, as explained on the *Howto_triclinic* doc page, internally, LAMMPS only uses restricted triclinic simulation boxes. This means the box and per-atom information (e.g. coordinates, velocities) LAMMPS stores are converted (rotated) from general to restricted triclinic form when the system is created.

For dump output, if the *dump_modify triclinic/general* command is used, the box description and per-atom coordinates and other per-atom vectors will be converted (rotated) from restricted to general form when each dump file snapshots is output. This option can only be used if the simulation box was initially created as general triclinic. If the option is not used, and the simulation box is general triclinic, then the dump file snapshots will reflect the internal restricted triclinic geometry.

The dump_modify triclinic/general option affects 3 aspects of the dump file output.

First, the format for the BOX BOUNDS is as follows

```
ITEM: BOX BOUNDS abc origin
ax ay az originx
bx by bz originy
cx cy cz originz
```

where the **A** edge vector of the box is (ax,ay,az) and similarly for **B** and **C**. The origin of all 3 edge vectors is (originx, originy, originz).

Second, the coordinates of each atom are converted (rotated) so that the atom is inside (or near) the general triclinic box defined by the **A**, **B**, **C** edge vectors. For style *atom*, this only alters output for unscaled atom coords, via the *dump_modify scaled no* setting. For style *custom*, this alters output for either unscaled or unwrapped output of atom coords, via the *x,y,z* or *xu,yu,zu* attributes. For output of scaled atom coords by both styles, there is no difference between restricted and general triclinic values.

Third, the output for any attribute of the *custom* style which represents a per-atom vector quantity will be converted (rotated) to be oriented consistent with the general triclinic box and its orientation relative to the standard xyz coordinate axes.

This applies to the following *custom* style attributes:

- vx,vy,vz = atom velocities
- fx,fy,fz = forces on atoms
- mux,muy,muz = orientation of dipole moment of atom
- omegax,omegay,omegaz = angular velocity of spherical particle
- angmomx,angmomy,angmomz = angular momentum of aspherical particle
- tqx,tqy,tqz = torque on finite-size particles

For example, if the velocity of an atom in a restricted triclinic box is along the x-axis, then it will be output for a general triclinic box as a vector along the **A** edge vector of the box.

Note

For style *custom*, the *dump_modify thresh* command may access per-atom attributes either directly or indirectly through a compute or variable. If the attribute is an atom coordinate or one of the vectors mentioned above, its value will *NOT* be a general triclinic (rotated) value. Rather it will be a restricted triclinic value.

Arguments for different styles:

The sections below describe per-atom, local, and per grid cell attributes which can be used as arguments to the various styles.

Note that in the discussion below, for styles which can reference values from a compute or fix or custom atom property, like the *custom*, *cfg*, *grid* or *local* styles, the bracketed index *i* can be specified using a wildcard asterisk with the index to effectively specify multiple values. This takes the form "*" or "*n" or "m*" or "m*n". If *N* is the number of columns in the array, then an asterisk with no numeric values means all column indices from 1 to *N*. A leading asterisk means all indices from 1 to n (inclusive). A trailing asterisk means all indices from m to *N* (inclusive). A middle asterisk means all indices from m to n (inclusive).

Using a wildcard is the same as if the individual columns of the array had been listed one by one. For example, these two dump commands are equivalent, since the *compute stress/atom* command creates a per-atom array with six columns:

```
compute myPress all stress/atom NULL
dump 2 all custom 100 tmp.dump id myPress[*]
dump 2 all custom 100 tmp.dump id myPress[1] myPress[2] myPress[3] &
                                  myPress[4] myPress[5] myPress[6]
```

Per-atom attributes used as arguments to the *custom* and *cfg* styles:

The *id*, *mol*, *proc*, *procp1*, *type*, *typelabel*, *element*, *mass*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, *q* attributes are self-explanatory.

Id is the atom ID. *Mol* is the molecule ID, included in the data file for molecular systems. *Proc* is the ID of the processor (0 to $N_{\mathrm{procs}} - 1$) that currently owns the atom. *Procp1* is the proc ID+1, which can be convenient in place of a *type* attribute (1 to N_{types}) for coloring atoms in a visualization program. *Type* is the atom type (1 to N_{types}). *Typelabel* is the atom *type label*. *Element* is typically the chemical name of an element, which you must assign to each type via the *dump_modify element* command. More generally, it can be any string you wish to associated with an atom type. *Mass* is the atom mass. The quantities *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, and *q* are components of atom velocity and force and atomic charge.

There are several options for outputting atom coordinates. The *x*, *y*, and *z* attributes write atom coordinates "unscaled", in the appropriate distance *units* (Å, σ, etc.). Use *xs*, *ys*, and *zs* if you want the coordinates "scaled" to the box size so that each value is 0.0 to 1.0. If the simulation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. The actual unscaled (x, y, z) coordinate is $x_s a + y_s b + z_s c$, where (a, b, c) are the non-orthogonal vectors of the simulation box edges, as discussed on the *Howto triclinic* page.

Use *xu*, *yu*, and *zu* if you want the coordinates "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that using *xu*, *yu*, and *zu* means that the coordinate values may be far outside the box bounds printed with the snapshot. Using *xsu*, *ysu*, and *zsu* is similar to using *xu*, *yu*, and *zu*, except that the unwrapped coordinates are scaled by the box size. Atoms that have passed through a periodic boundary will have the corresponding coordinate increased or decreased by 1.0.

The image flags can be printed directly using the *ix*, *iy*, and *iz* attributes. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation.

The *mux*, *muy*, and *muz* attributes are specific to dipolar systems defined with an atom style of *dipole*. They give the orientation of the atom's point dipole moment. The *mu* attribute gives the magnitude of the atom's dipole moment.

The *radius* and *diameter* attributes are specific to spherical particles that have a finite size, such as those defined with an atom style of *sphere*.

The *omegax*, *omegay*, and *omegaz* attributes are specific to finite-size spherical particles that have an angular velocity. Only certain atom styles, such as *sphere*, define this quantity.

The *angmomx*, *angmomy*, and *angmomz* attributes are specific to finite-size aspherical particles that have an angular momentum. Only the *ellipsoid* atom style defines this quantity.

The *tqx*, *tqy*, and *tqz* attributes are for finite-size particles that can sustain a rotational torque due to interactions with other particles.

The *c_ID* and *c_ID[I]* attributes allow per-atom vectors or arrays calculated by a *compute* to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the *compute* command for details. There are computes for calculating the per-atom energy, stress, centro-symmetry parameter, and coordination number of individual atoms.

Note that computes which calculate global or local quantities, as opposed to per-atom quantities, cannot be output in a dump custom command. Instead, global quantities can be output by the *thermo_style custom* command, and local quantities can be output by the dump local command.

If *c_ID* is used as a attribute, then the per-atom vector calculated by the compute is printed. If *c_ID[i]* is used, then *i* must be in the range from 1 to *M*, which will print the *i*th column of the per-atom array with *M* columns calculated by the compute. See the discussion above for how *i* can be specified with a wildcard asterisk to effectively specify multiple values.

The *f_ID* and *f_ID[I]* attributes allow vector or array per-atom quantities calculated by a *fix* to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script. The *fix ave/atom* command is one that calculates per-atom quantities. Since it can time-average per-atom quantities produced by any *compute*, *fix*, or atom-style *variable*, this allows those time-averaged results to be written to a dump file.

If *f_ID* is used as a attribute, then the per-atom vector calculated by the fix is printed. If *f_ID[i]* is used, then *i* must be in the range from 1 to *M*, which will print the *i*th column of the per-atom array with *M* columns calculated by the fix. See the discussion above for how *i* can be specified with a wildcard asterisk to effectively specify multiple values.

The *v_name* attribute allows per-atom vectors calculated by a *variable* to be output. The name in the attribute should be replaced by the actual name of the variable that has been defined previously in the input script. Only an atom-style variable can be referenced, since it is the only style that generates per-atom values. Variables of style *atom* can reference individual atom attributes, per-atom attributes, thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of creating quantities to output to a dump file.

The *i_name*, *d_name*, *i2_name*, *d2_name* attributes refer to custom per-atom integer and floating-point vectors or arrays that have been added via the *fix property/atom* command. When that command is used specific names are given to each attribute which are the "name" portion of these keywords. For arrays *i2_name* and *d2_name*, the column of the array must also be included following the name in brackets (e.g., d2_xyz[i], i2_mySpin[i], where *i* is in the range from 1 to *M*, where *M* is the number of columns in the custom array). See the discussion above for how *i* can be specified with a wildcard asterisk to effectively specify multiple values.

See the *Modify* page for information on how to add new compute and fix styles to LAMMPS to calculate per-atom quantities which could then be output into dump files.

Attributes used as arguments to the *local* style:

The *index* attribute can be used to generate an index number from 1 to N for each line written into the dump file, where N is the total number of local datums from all processors, or lines of output that will appear in the snapshot. Note that because data from different processors depend on what atoms they currently own, and atoms migrate between processor, there is no guarantee that the same index will be used for the same info (e.g. a particular bond) in successive snapshots.

The *c_ID* and *c_ID[I]* attributes allow local vectors or arrays calculated by a *compute* to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the *compute* command for details. There are computes for calculating local information such as indices, types, and energies for bonds and angles.

Note that computes which calculate global or per-atom quantities, as opposed to local quantities, cannot be output in a dump local command. Instead, global quantities can be output by the *thermo_style custom* command, and per-atom quantities can be output by the dump custom command.

If *c_ID* is used as a attribute, then the local vector calculated by the compute is printed. If *c_ID[I]* is used, then I must be in the range from 1-M, which will print the Ith column of the local array with M columns calculated by the compute. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

The *f_ID* and *f_ID[I]* attributes allow local vectors or arrays calculated by a *fix* to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script.

If *f_ID* is used as a attribute, then the local vector calculated by the fix is printed. If *f_ID[I]* is used, then I must be in the range from 1-M, which will print the Ith column of the local with M columns calculated by the fix. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

Here is an example of how to dump bond info for a system, including the distance and energy of each bond:

```
compute 1 all property/local batom1 batom2 btype
compute 2 all bond/local dist eng
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_2[1] c_2[2]
```

Attributes used as arguments to the *grid* and *grid/vtk* styles:

The attributes that begin with *c_ID* and *f_ID* both take colon-separated fields *gname* and *dname*. These refer to a grid name and data field name which is defined by the compute or fix. Note that a compute or fix can define one or more grids (of different sizes) and one or more data fields for each of those grids. The sizes of all grids output in a single dump grid command must be the same.

The *c_ID:gname:dname* and *c_ID:gname:dname[I]* attributes allow per-grid vectors or arrays calculated by a *compute* to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script.

If *c_ID:gname:dname* is used as a attribute, then the per-grid vector calculated by the compute is printed. If *c_ID:gname:dname[I]* is used, then I must be in the range from 1-M, which will print the Ith column of the per-grid array with M columns calculated by the compute. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

The *f_ID:gname:dname* and *f_ID:gname:dname[I]* attributes allow per-grid vectors or arrays calculated by a *fix* to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script.

If *f_ID:gname:dname* is used as a attribute, then the per-grid vector calculated by the fix is printed. If *f_ID:gname:dname[I]* is used, then I must be in the range from 1-M, which will print the Ith column of the per-grid with M columns calculated by the fix. See the discussion above for how I can be specified with a wildcard asterisk to effectively specify multiple values.

9.10.4 Restrictions

To write gzipped dump files, you must either compile LAMMPS with the -DLAMMPS_GZIP option or use the styles from the COMPRESS package. See the *Build settings* page for details.

While a dump command is active (i.e., has not been stopped by using the *undump command*), no commands may be used that will change the timestep (e.g., *reset_timestep*). LAMMPS will terminate with an error otherwise.

The *atom/gz*, *cfg/gz*, *custom/gz*, and *xyz/gz* styles are part of the COMPRESS package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

The *xtc*, *dcd*, and *yaml* styles are part of the EXTRA-DUMP package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

9.10.5 Related commands

dump atom/adios, *dump custom/adios*, *dump cfg/uef*, *dump h5md*, *dump image*, *dump molfile*, *dump netcdf*, *dump netcdf/mpiio*, *dump_modify*, *undump*, *write_dump*

9.10.6 Default

The defaults for the *image* and *movie* styles are listed on the *dump image* page.

9.11 dump atom/adios command

9.12 dump custom/adios command

9.12.1 Syntax

```
dump ID group-ID atom/adios N file.bp
dump ID group-ID custom/adios N file.bp args
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- adios = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the *dump* doc page)
- N = dump every this many timesteps
- file.bp = name of file/stream to write to
- args = same options as in *dump custom* command

9.12.2 Examples

```
dump adios1 all atom/adios   100 atoms.bp
dump 4a     all custom/adios 100 dump_adios.bp id v_p x y z
dump 2 subgroup custom/adios 100 dump_adios.bp mass type xs ys zs vx vy vz
```

9.12.3 Description

Dump a snapshot of atom coordinates every *N* timesteps in the ADIOS-based "BP" file format, or using different I/O solutions in ADIOS, to a stream that can be read on-line by another program. ADIOS-BP files are binary, portable, and self-describing.

Note

To be able to use ADIOS, a file adios2_config.xml with specific configuration settings is expected in the current working directory. If the file is not present, LAMMPS will try to create a minimal default file. Please refer to the ADIOS documentation for details on how to adjust this file for optimal performance and desired features.

Use from write_dump:

It is possible to use these dump styles with the *write_dump* command. In this case, the sub-intervals must not be set at all. The write_dump command can be used to create a new file at each individual dump.

```
dump 4     all atom/adios 100 dump.bp
write_dump all atom/adios singledump.bp
```

9.12.4 Restrictions

The number of atoms per snapshot **can** change with the adios style. When using the ADIOS tool 'bpls' to list the content of a .bp file, bpls will print __ for the size of the output table indicating that its size is changing every step.

The *atom/adios* and *custom/adios* dump styles are part of the ADIOS package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

9.12.5 Related commands

dump, *dump_modify*, *undump*

9.13 dump cfg/uef command

9.13.1 Syntax

```
dump ID group-ID cfg/uef N file mass type xs ys zs args
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be dumped
- N = dump every this many timesteps
- file = name of file to write dump info to

  args = same as args for dump custom

9.13.2 Examples

```
dump 1 all cfg/uef 10 dump.*.cfg mass type xs ys zs
dump 2 all cfg/uef 100 dump.*.cfg mass type xs ys zs id c_stress
```

9.13.3 Description

This command is used to dump atomic coordinates in the reference frame of the applied flow field when *fix nvt/uef* or *fix npt/uef* is used. Only the atomic coordinates and frame-invariant scalar quantities will be in the flow frame. If velocities are selected as output, for example, they will not be in the same reference frame as the atomic positions.

9.13.4 Restrictions

This fix is part of the UEF package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

This command can only be used when *fix nvt/uef* or *fix npt/uef* is active.

9.13.5 Related commands

dump, *fix nvt/uef*

9.13.6 Default

none

9.14 dump h5md command

9.14.1 Syntax

```
dump ID group-ID h5md N file.h5 args
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- *h5md* = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the *dump* doc page)
- N = dump every this many timesteps
- file.h5 = name of file to write to
- args = *position* options or *image* or *velocity* options or *force* options or *species* options or *file_from* ID or *box* value or *create_group* value or *author* value = list of data elements to dump, with their dump "sub-intervals"

  ```
  position options
  image
  velocity options
  force options
  species options
  file_from ID = do not open a new file, re-use the already opened file from dump ID
  box value = yes or no
  create_group value = yes or no
  author value = quoted string
  ```

Note that at least one element must be specified and that *image* may only be present if *position* is specified first.

For the elements *position*, *velocity*, *force* and *species*, a sub-interval may be specified to write the data only every N_element iterations of the dump (i.e. every N*N_element time steps). This is specified by this option directly following the element declaration:

```
options = every N_element
```

9.14.2 Examples

```
dump h5md1 all h5md 100 dump_h5md.h5 position image
dump h5md1 all h5md 100 dump_h5md.h5 position velocity every 10
dump h5md1 all h5md 100 dump_h5md.h5 velocity author "John Doe"
```

9.14.3 Description

Dump a snapshot of atom coordinates every N timesteps in the HDF5 based H5MD file format *(de Buyl)*. HDF5 files are binary, portable and self-describing. This dump style will write only one file, on the root node.

Several dumps may write to the same file, by using file_from and referring to a previously defined dump. Several groups may also be stored within the same file by defining several dumps. A dump that refers (via *file_from*) to an already open dump ID and that concerns another particle group must specify *create_group yes*.

Each data element is written every N*N_element steps. For *image*, no sub-interval is needed as it must be present at the same interval as *position*. *image* must be given after *position* in any case. The box information (edges in each dimension) is stored at the same interval than the *position* element, if present. Else it is stored every N steps.

Note

Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

Use from write_dump:

It is possible to use this dump style with the *write_dump* command. In this case, the sub-intervals must not be set at all. The write_dump command can be used either to create a new file or to add current data to an existing dump file by using the *file_from* keyword.

Typically, the *species* data is fixed. The following two commands store the position data every 100 timesteps, with the image data, and store once the species data in the same file.

```
dump h5md1 all h5md 100 dump.h5 position image
write_dump all h5md dump.h5 file_from h5md1 species
```

9.14.4 Restrictions

The number of atoms per snapshot cannot change with the h5md style. The position data is stored wrapped (box boundaries not enforced, see note above). Only orthogonal domains are currently supported. This is a limitation of the present dump h5md command and not of H5MD itself.

The *h5md* dump style is part of the H5MD package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info. It also requires (i) building the ch5md library provided with LAMMPS (See the *Build package* page for more info.) and (ii) having the HDF5 library installed (C bindings are sufficient) on your system. The library ch5md is compiled with the h5cc wrapper provided by the HDF5 library.

9.14.5 Related commands

dump, *dump_modify*, *undump*

(de Buyl) de Buyl, Colberg and Hofling, H5MD: A structured, efficient, and portable file format for molecular data, Comp. Phys. Comm. 185(6), 1546-1553 (2014) - [arXiv:1308.6382].

9.15 dump image command

9.16 dump movie command

(see below for *dump_modify options* specific to dump image/movie)

9.16.1 Syntax

```
dump ID group-ID style N file color diameter keyword value ...
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- style = *image* or *movie* = style of dump command (other styles such as *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the *dump* doc page)
- N = dump every this many timesteps
- file = name of file to write image to
- color = atom attribute that determines color of each atom
- diameter = atom attribute that determines size of each atom
- zero or more keyword/value pairs may be appended
- keyword = *atom* or *adiam* or *bond* or *grid* or *line* or *tri* or *body* or *fix* or *size* or *view* or *center* or *up* or *zoom* or *box* or *axes* or *subbox* or *shiny* or *fsaa* or *ssao*

```
atom = yes or no = do or do not draw atoms
adiam size = numeric value for atom diameter (distance units)
bond values = color width = color and width of bonds
  color = atom or type or none
  width = number or atom or type or none
    number = numeric value for bond width (distance units)
grid = per-grid value to use when coloring each grid cell
  per-grid value = c_ID:gname:dname, c_ID:gname:dname[I], f_ID:gname:dname, f_
↪ID:gname:dname[I]
    gname = name of grid defined by compute or fix
    dname = name of data field defined by compute or fix
    c_ID = per-grid vector calculated by a compute with ID
    c_ID[I] = Ith column of per-grid array calculated by a compute with ID
    f_ID = per-grid vector calculated by a fix with ID
    f_ID[I] = Ith column of per-grid array calculated by a fix with ID
line = color width
  color = type
  width = numeric value for line width (distance units)
tri = color tflag width
  color = type
  tflag = 1 for just triangle, 2 for just tri edges, 3 for both
  width = numeric value for tringle edge width (distance units)
body = color bflag1 bflag2
  color = type
  bflag1,bflag2 = 2 numeric flags to affect how bodies are drawn
fix = fixID color fflag1 fflag2
  fixID = ID of fix that generates objects to draw
  color = type
  fflag1,fflag2 = 2 numeric flags to affect how fix objects are drawn
size values = width height = size of images
  width = width of image in # of pixels
  height = height of image in # of pixels
view values = theta phi = view of simulation box
  theta = view angle from +z axis (degrees)
```

```
    phi = azimuthal view angle (degrees)
    theta or phi can be a variable (see below)
  center values = flag Cx Cy Cz = center point of image
    flag = s for static, d for dynamic
    Cx,Cy,Cz = center point of image as fraction of box dimension (0.5 = center of box)
    Cx,Cy,Cz can be variables (see below)
  up values = Ux Uy Uz = direction that is "up" in image
    Ux,Uy,Uz = components of up vector
    Ux,Uy,Uz can be variables (see below)
  zoom value = zfactor = size that simulation box appears in image
    zfactor = scale image size by factor > 1 to enlarge, factor < 1 to shrink
    zfactor can be a variable (see below)
  box values = yes/no diam = draw outline of simulation box
    yes/no = do or do not draw simulation box lines
    diam = diameter of box lines as fraction of shortest box length
  axes values = axes length diam = draw xyz axes
    axes = yes or no = do or do not draw xyz axes lines next to simulation box
    length = length of axes lines as fraction of respective box lengths
    diam = diameter of axes lines as fraction of shortest box length
  subbox values = lines diam = draw outline of processor subdomains
    lines = yes or no = do or do not draw subdomain lines
    diam = diameter of subdomain lines as fraction of shortest box length
  shiny value = sfactor = shinyness of spheres and cylinders
    sfactor = shinyness of spheres and cylinders from 0.0 to 1.0
  fsaa arg = yes/no
    yes/no = do or do not apply anti-aliasing
  ssao value = shading seed dfactor = SSAO depth shading
    shading = yes or no = turn depth shading on/off
    seed = random # seed (positive integer)
    dfactor = strength of shading from 0.0 to 1.0
```

9.17 dump_modify options for dump image/movie

9.17.1 Syntax

```
dump_modify dump-ID keyword values ...
```

- these keywords apply only to the *image* and *movie* styles and are documented on this page
- keyword = *acolor* or *adiam* or *amap* or *gmap* or *backcolor* or *bcolor* or *bdiam* or *bitrate* or *boxcolor* or *color* or *framerate* or *gmap*
- see the *dump modify* doc page for more general keywords

```
  acolor args = type color
    type = atom type (numeric or type label) or range of numeric types (see below)
    color = name of color or color1/color2/...
  adiam args = type diam
    type = atom type (numeric or type label) or range of numeric types (see below)
    diam = diameter of atoms of that type (distance units)
  amap args = lo hi style delta N entry1 entry2 ... entryN
    lo = number or min = lower bound of range of color map
    hi = number or max = upper bound of range of color map
    style = 2 letters = c or d or s plus a or f
      c for continuous
      d for discrete
      s for sequential
      a for absolute
      f for fractional
    delta = binsize (only used for style s, otherwise ignored)
      binsize = range is divided into bins of this width
    N = # of subsequent entries
    entry = value color (for continuous style)
      value = number or min or max = single value within range
      color = name of color used for that value
    entry = lo hi color (for discrete style)
      lo/hi = number or min or max = lower/upper bound of subset of range
      color = name of color used for that subset of values
    entry = color (for sequential style)
      color = name of color used for a bin of values
  backcolor arg = color
    color = name of color for background
  bcolor args = type color
    type = bond type (numeric or type label) or range of numeric types (see below)
    color = name of color or color1/color2/...
  bdiam args = type diam
    type = bond type (numeric or type label) or range of numeric types (see below)
    diam = diameter of bonds of that type (distance units)
  bitrate arg = rate
    rate = target bitrate for movie in kbps
  boxcolor arg = color
    color = name of color for simulation box lines and processor subdomain lines
  color args = name R G B
    name = name of color
    R,G,B = red/green/blue numeric values from 0.0 to 1.0
  framerate arg = fps
    fps = frames per second for movie
  gmap args = identical to amap args
```

9.17.2 Examples

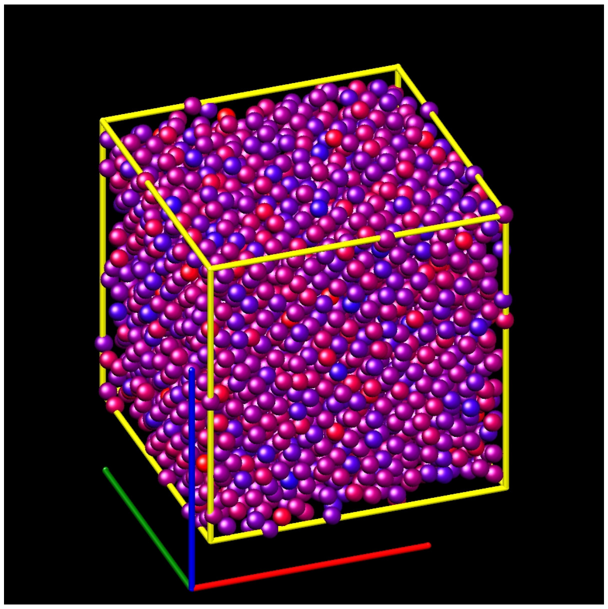
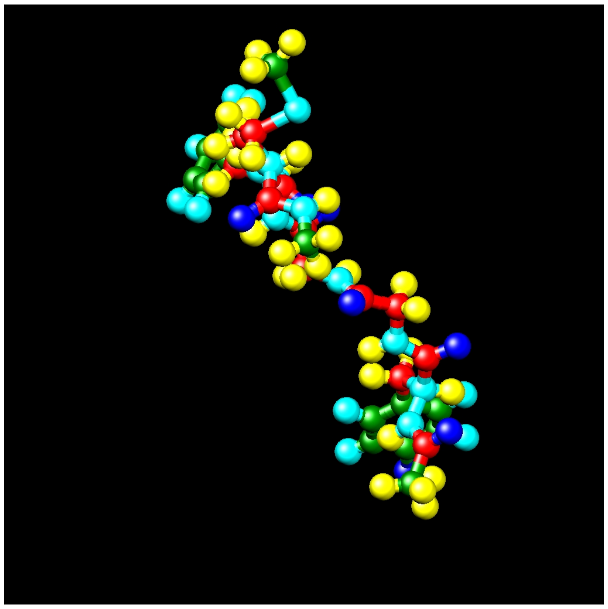
```
dump d0 all image 100 dump.*.jpg type type
dump d1 mobile image 500 snap.*.png element element ssao yes 4539 0.6
dump d2 all image 200 img-*.ppm type type zoom 2.5 adiam 1.5 size 1280 720
dump m0 all movie 1000 movie.mpg type type size 640 480
dump m1 all movie 1000 movie.avi type type size 640 480
dump m2 all movie 100 movie.m4v type type zoom 1.8 adiam v_value size 1280 720

dump_modify 1 amap min max cf 0.0 3 min green 0.5 yellow max blue boxcolor red

labelmap atom 1 C 2 H 3 O 4 N
dump_modify 1 acolor C gray acolor H white acolor O red acolor N blue
```

9.17.3 Description

Dump a high-quality rendered image of the atom configuration every *N* timesteps and save the images either as a sequence of JPEG or PNG or PPM files, or as a single movie file. The options for this command as well as the *dump_modify* command control what is included in the image or movie and how it appears. A series of such images can easily be manually converted into an animated movie of your simulation or the process can be automated without writing the intermediate files using the dump movie style; see further details below. Other dump styles store snapshots of numerical data associated with atoms in various formats, as discussed on the *dump* doc page.

Note that a set of images or a movie can be made after a simulation has been run, using the *rerun* command to read snapshots from an existing dump file, and using these dump commands in the rerun script to generate the images/movie.

Here are two sample images, rendered as 1024 $\times$ 1024 JPEG files.

Only atoms in the specified group are rendered in the image. The *dump_modify region and thresh* commands can also alter what atoms are included in the image. The filename suffix determines whether a JPEG, PNG, or PPM file is created with the *image* dump style. If the suffix is ".jpg" or ".jpeg", then a JPEG format file is created, if the suffix is ".png", then a PNG format is created, else a PPM (aka NETPBM) format file is created. The JPEG and PNG files are binary; PPM has a text mode header followed by binary data. JPEG images have lossy compression, PNG has lossless compression, and PPM files are uncompressed but can be compressed with gzip, if LAMMPS has been compiled with -DLAMMPS_GZIP and a ".gz" suffix is used.

Similarly, the format of the resulting movie is chosen with the *movie* dump style. This is handled by the underlying FFmpeg converter and thus details have to be looked up in the FFmpeg documentation. Typical examples are: .avi, .mpg, .m4v, .mp4, .mkv, .flv, .mov, .gif Additional settings of the movie compression like *bitrate* and *framerate* can be set using the dump_modify command as described below.

To write out JPEG and PNG format files, you must build LAMMPS with support for the corresponding JPEG or PNG library. To convert images into movies, LAMMPS has to be compiled with the -DLAMMPS_FFMPEG flag. See the *Build settings* page for details.

Note

Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom in the image may be slightly outside the simulation box.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the *dump_modify first* command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the *dump_modify every* command.

Dump *image* filenames must contain a wildcard character "*" so that one image file per snapshot is written. The "*" character is replaced with the timestep value. For example, tmp.dump.*.jpg becomes tmp.dump.0.jpg, tmp.dump.10000.jpg, tmp.dump.20000.jpg, etc. Note that the *dump_modify pad* command can be used to ensure all timestep numbers are the same length (e.g., 00010), which can make it easier to convert a series of images into a movie in the correct ordering.

Dump *movie* filenames on the other hand, must not have any wildcard character since only one file combining all images into a single movie will be written by the movie encoder.

The *color* and *diameter* settings determine the color and size of atoms rendered in the image. They can be any atom attribute defined for the *dump custom* command, including *type* and *element*. This includes per-atom quantities calculated by a *compute*, *fix*, or *variable*, which are prefixed by "c_", "f_", or "v_", respectively. Note that the *diameter* setting can be overridden with a numeric value applied to all atoms by the optional *adiam* keyword.

If *type* is specified for the *color* setting, then the color of each atom is determined by its atom type. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. This mapping can be changed by the "dump_modify acolor" command, as described below.

If *type* is specified for the *diameter* setting then the diameter of each atom is determined by its atom type. By default all types have diameter 1.0. This mapping can be changed by the "dump_modify adiam" command, as described below.

If *element* is specified for the *color* and/or *diameter* setting, then the color and/or diameter of each atom is determined by which element it is, which in turn is specified by the element-to-type mapping specified by the "dump_modify element" command, as described below. By default every atom type is C (carbon). Every element has a color and diameter associated with it, which is the same as the colors and sizes used by the AtomEye visualization package.

If other atom attributes are used for the *color* or *diameter* settings, they are interpreted in the following way.

If "vx", for example, is used as the *color* setting, then the color of the atom will depend on the x-component of its velocity. The association of a per-atom value with a specific color is determined by a "color map", which can be specified via the dump_modify amap command, as described below. The basic idea is that the atom-attribute will be within a range of values, and every value within the range is mapped to a specific color. Depending on how the color map is defined, that mapping can take place via interpolation so that a value of -3.2 is halfway between "red" and "blue", or discretely so that the value of -3.2 is "orange".

If "vx", for example, is used as the *diameter* setting, then the atom will be rendered using the x-component of its velocity as the diameter. If the per-atom value <= 0.0, them the atom will not be drawn. Note that finite-size spherical particles, as defined by *atom_style sphere* define a per-particle radius or diameter, which can be used as the *diameter* setting.

The various keywords listed above control how the image is rendered. As listed below, all of the keywords have defaults, most of which you will likely not need to change. As described below, the dump modify command also has options specific to the dump image style, particularly for assigning colors to atoms, bonds, and other image features.

The *atom* keyword allow you to turn off the drawing of all atoms, if the specified value is *no*. Note that this will not turn off the drawing of particles that are represented as lines, triangles, or bodies, as discussed below. These particles can be drawn separately if the *line*, *tri*, or *body* keywords are used.

The *adiam* keyword allows you to override the *diameter* setting to set a single numeric *size*. All atoms will be drawn with that diameter, e.g. 1.5, which is in whatever distance *units* the input script defines, e.g. Angstroms.

The *bond* keyword allows to you to alter how bonds are drawn. A bond is only drawn if both atoms in the bond are being drawn due to being in the specified group and due to other selection criteria (e.g. region, threshold settings of the *dump_modify* command). By default, bonds are drawn if they are defined in the input data file as read by the *read_data* command. Using *none* for both the bond *color* and *width* value will turn off the drawing of all bonds.

If *atom* is specified for the bond *color* value, then each bond is drawn in 2 halves, with the color of each half being the color of the atom at that end of the bond.

If *type* is specified for the *color* value, then the color of each bond is determined by its bond type. By default the mapping of bond types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for bond types > 6. This mapping can be changed by the "dump_modify bcolor" command, as described below.

The bond *width* value can be a numeric value or *atom* or *type* (or *none* as indicated above).

If a numeric value is specified, then all bonds will be drawn as cylinders with that diameter, e.g. 1.0, which is in whatever distance *units* the input script defines, e.g. Angstroms.

If *atom* is specified for the *width* value, then each bond will be drawn with a width corresponding to the minimum diameter of the two atoms in the bond.

If *type* is specified for the *width* value then the diameter of each bond is determined by its bond type. By default all types have diameter 0.5. This mapping can be changed by the "dump_modify bdiam" command, as described below.

The *line* keyword can be used when *atom_style line* is used to define particles as line segments, and will draw them as lines. If this keyword is not used, such particles will be drawn as spheres, the same as if they were regular atoms. The only setting currently allowed for the *color* value is *type*, which will color the lines according to the atom type of the particle. By default the mapping of types to colors is as follows:

- type 1 = red

- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. There is not yet an option to change this via the dump_modify command.

The line *width* can only be a numeric value, which specifies that all lines will be drawn as cylinders with that diameter, e.g. 1.0, which is in whatever distance *units* the input script defines, e.g. Angstroms.

The *tri* keyword can be used when *atom_style tri* is used to define particles as triangles, and will draw them as triangles or edges (3 lines) or both, depending on the setting for *tflag*. If edges are drawn, the *width* setting determines the diameters of the line segments. If this keyword is not used, triangle particles will be drawn as spheres, the same as if they were regular atoms. The only setting currently allowed for the *color* value is *type*, which will color the triangles according to the atom type of the particle. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. There is not yet an option to change this via the dump_modify command.

The *body* keyword can be used when *atom_style body* is used to define body particles with internal state (e.g. sub-particles), and will drawn them in a manner specific to the body style. If this keyword is not used, such particles will be drawn as spheres, the same as if they were regular atoms.

The *Howto body* page describes the body styles LAMMPS currently supports, and provides more details as to the kind of body particles they represent and how they are drawn by this dump image command. For all the body styles, individual atoms can be either a body particle or a usual point (non-body) particle. Non-body particles will be drawn the same way they would be as a regular atom. The *bflag1* and *bflag2* settings are numerical values which are passed to the body style to affect how the drawing of a body particle is done. See the *Howto body* page for a description of what these parameters mean for each body style.

The only setting currently allowed for the *color* value is *type*, which will color the body particles according to the atom type of the particle. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. There is not yet an option to change this via the dump_modify command.

The *fix* keyword can be used with a *fix* that produces objects to be drawn.

The *fflag1* and *fflag2* settings are numerical values which are passed to the fix to affect how the drawing of its objects is done. See the individual fix page for a description of what these parameters mean for a particular fix.

The only setting currently allowed for the *color* value is *type*, which will color the fix objects according to their type. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. There is not yet an option to change this via the dump_modify command.

The *size* keyword sets the width and height of the created images, i.e. the number of pixels in each direction.

The *view*, *center*, *up*, and *zoom* values determine how 3d simulation space is mapped to the 2d plane of the image. Basically they control how the simulation box appears in the image.

All of the *view*, *center*, *up*, and *zoom* values can be specified as numeric quantities, whose meaning is explained below. Any of them can also be specified as an *equal-style variable*, by using v_name as the value, where "name" is the variable name. In this case the variable will be evaluated on the timestep each image is created to create a new value. If the equal-style variable is time-dependent, this is a means of changing the way the simulation box appears from image to image, effectively doing a pan or fly-by view of your simulation.

The *view* keyword determines the viewpoint from which the simulation box is viewed, looking towards the *center* point. The *theta* value is the vertical angle from the +z axis, and must be an angle from 0 to 180 degrees. The *phi* value is an azimuthal angle around the z axis and can be positive or negative. A value of 0.0 is a view along the +x axis, towards the *center* point. If *theta* or *phi* are specified via variables, then the variable values should be in degrees.

The *center* keyword determines the point in simulation space that will be at the center of the image. *Cx*, *Cy*, and *Cz* are specified as fractions of the box dimensions, so that (0.5,0.5,0.5) is the center of the simulation box. These values do not have to be between 0.0 and 1.0, if you want the simulation box to be offset from the center of the image. Note, however, that if you choose strange values for *Cx*, *Cy*, or *Cz* you may get a blank image. Internally, *Cx*, *Cy*, and *Cz* are converted into a point in simulation space. If *flag* is set to "s" for static, then this conversion is done once, at the time the dump command is issued. If *flag* is set to "d" for dynamic then the conversion is performed every time a new image is created. If the box size or shape is changing, this will adjust the center point in simulation space.

The *up* keyword determines what direction in simulation space will be "up" in the image. Internally it is stored as a vector that is in the plane perpendicular to the view vector implied by the *theta* and *pni* values, and which is also in the plane defined by the view vector and user-specified up vector. Thus this internal vector is computed from the user-specified *up* vector as

$$\text{up\_internal} = \text{view cross (up cross view)}$$

This means the only restriction on the specified *up* vector is that it cannot be parallel to the *view* vector, implied by the *theta* and *phi* values.

The *zoom* keyword scales the size of the simulation box as it appears in the image. The default *zfactor* value of 1 should display an image mostly filled by the atoms in the simulation box. A *zfactor* > 1 will make the simulation box larger; a *zfactor* < 1 will make it smaller. *Zfactor* must be a value > 0.0.

The *box* keyword determines if and how the simulation box boundaries are rendered as thin cylinders in the image. If *no* is set, then the box boundaries are not drawn and the *diam* setting is ignored. If *yes* is set, the 12 edges of the box are drawn, with a diameter that is a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d). The color of the box boundaries can be set with the "dump_modify boxcolor" command.

The *axes* keyword determines if and how the coordinate axes are rendered as thin cylinders in the image. If *no* is set, then the axes are not drawn and the *length* and *diam* settings are ignored. If *yes* is set, 3 thin cylinders are drawn to represent the x,y,z axes in colors red,green,blue. The origin of these cylinders will be offset from the lower left corner of the box by 10%. The *length* setting determines how long the cylinders will be as a fraction of the respective box lengths. The *diam* setting determines their thickness as a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d).

The *subbox* keyword determines if and how processor subdomain boundaries are rendered as thin cylinders in the image. If *no* is set (default), then the subdomain boundaries are not drawn and the *diam* setting is ignored. If *yes* is set, the 12 edges of each processor subdomain are drawn, with a diameter that is a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d). The color of the subdomain boundaries can be set with the "dump_modify boxcolor" command.

The *shiny* keyword determines how shiny the objects rendered in the image will appear. The *sfactor* value must be a value 0.0 <= *sfactor* <= 1.0, where *sfactor* = 1 is a highly reflective surface and *sfactor* = 0 is a rough non-shiny surface.

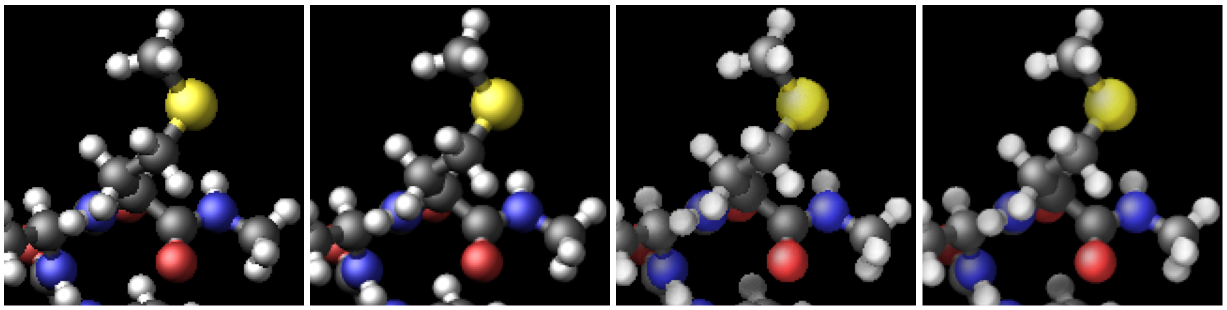
Added in version 21Nov2023.

The *fsaa* keyword can be used with the dump image command to improve the image quality by enabling full scene anti-aliasing. Internally the image is rendered at twice the width and height and then scaled down by computing the average of each 2x2 block of pixels to produce a single pixel in the final image at the original size. This produces images with smoother, less ragged edges. The application of this algorithm can increase the cost of computing the image by about 3x or more.

The *ssao* keyword turns on/off a screen space ambient occlusion (SSAO) model for depth shading. If *yes* is set, then atoms further away from the viewer are darkened via a randomized process, which is perceived as depth. The strength of the effect can be scaled by the *dfactor* parameter. If *no* is set, no depth shading is performed. The calculation of this effect can increase the cost of computing the image substantially by 5x or more, especially with larger images. When used in combination with the *fsaa* keyword the computational cost of depth shading is particularly large.

9.17.4 Image Quality Settings

The two keywords *fsaa* and *ssao* can be used to improve the image quality at the expense of additional computational cost to render the images. The images below show from left to right the same render with default settings, with *fsaa* added, with *ssao* added, and with both keywords added.



A series of JPEG, PNG, or PPM images can be converted into a movie file and then played as a movie using commonly available tools. Using dump style *movie* automates this step and avoids the intermediate step of writing (many) image snapshot file. But LAMMPS has to be compiled with -DLAMMPS_FFMPEG and an FFmpeg executable have to be installed.

To manually convert JPEG, PNG or PPM files into an animated GIF or MPEG or other movie file you can use:

- a) Use the ImageMagick convert program.

  ```
  convert *.jpg foo.gif
  convert -loop 1 *.ppm foo.mpg
  ```

  Animated GIF files from ImageMagick are not optimized. You can use a program like gifsicle to optimize and thus massively shrink them. MPEG files created by ImageMagick are in MPEG-1 format with a rather inefficient compression and low quality compared to more modern compression styles like MPEG-4, H.264, VP8, VP9, H.265 and so on.

- b) Use QuickTime.

  Select "Open Image Sequence" under the File menu Load the images into QuickTime to animate them Select "Export" under the File menu Save the movie as a QuickTime movie (*.mov) or in another format. QuickTime can generate very high quality and efficiently compressed movie files. Some of the supported formats require to buy a license and some are not readable on all platforms until specific runtime libraries are installed.

- c) Use FFmpeg

  FFmpeg is a command line tool that is available on many platforms and allows extremely flexible encoding and decoding of movies.

  ```
  cat snap.*.jpg | ffmpeg -y -f image2pipe -c:v mjpeg -i - -b:v 2000k movie.m4v
  cat snap.*.ppm | ffmpeg -y -f image2pipe -c:v ppm -i - -b:v 2400k movie.avi
  ```

  Front ends for FFmpeg exist for multiple platforms. For more information see the FFmpeg homepage

---

Play the movie:

- a) Use your browser to view an animated GIF movie.

  Select "Open File" under the File menu Load the animated GIF file

- b) Use the freely available mplayer or ffplay tool to view a movie. Both are available for multiple OSes and support a large variety of file formats and decoders.

  ```
  mplayer foo.mpg
  ffplay bar.avi
  ```

- c) Use the Pizza.py animate tool, which works directly on a series of image files.

```
a = animate("foo*.jpg")
```

- d) QuickTime and other Windows- or macOS-based media players can obviously play movie files directly. Similarly for corresponding tools bundled with Linux desktop environments. However, due to licensing issues with some file formats, the formats may require installing additional libraries, purchasing a license, or may not be supported.

9.17.5 Dump_modify keywords for dump image and dump movie

The following dump_modify keywords apply only to the dump image and dump movie styles. Any keyword that works with dump image also works with dump movie, since the movie is simply a collection of images. Some of the keywords only affect the dump movie style. The descriptions give details.

The *acolor* keyword can be used with the dump image command, when its atom color setting is *type*, to set the color that atoms of each type will be drawn in the image.

The specified *type* should be a type label or integer from 1 to Ntypes = the number of atom types. For numeric types, a wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the "dump_modify color" command, as described below. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified atom types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified atom types.

The *adiam* keyword can be used with the dump image command, when its atom diameter setting is *type*, to set the size that atoms of each type will be drawn in the image. The specified *type* should be a type label or integer from 1 to Ntypes. As with the *acolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of numeric atom types. The specified *diam* is the size in whatever distance *units* the input script is using, e.g. Angstroms.

The *amap* keyword can be used with the dump image command, with its *atom* keyword, when its atom setting is an atom-attribute, to setup a color map. The color map is used to assign a specific RGB (red/green/blue) color value to an individual atom when it is drawn, based on the atom's attribute, which is a numeric value, e.g. its x-component of velocity if the atom-attribute "vx" was specified.

The basic idea of a color map is that the atom-attribute will be within a range of values, and that range is associated with a series of colors (e.g. red, blue, green). An atom's specific value (vx = -3.2) can then mapped to the series of colors (e.g. halfway between red and blue), and a specific color is determined via an interpolation procedure.

There are many possible options for the color map, enabled by the *amap* keyword. Here are the details.

The *lo* and *hi* settings determine the range of values allowed for the atom attribute. If numeric values are used for *lo* and/or *hi*, then values that are lower/higher than that value are set to the value. I.e. the range is static. If *lo* is specified as *min* or *hi* as *max* then the range is dynamic, and the lower and/or upper bound will be calculated each time an image is drawn, based on the set of atoms being visualized.

The *style* setting is two letters, such as "ca". The first letter is either "c" for continuous, "d" for discrete, or "s" for sequential. The second letter is either "a" for absolute, or "f" for fractional.

A continuous color map is one in which the color changes continuously from value to value within the range. A discrete color map is one in which discrete colors are assigned to sub-ranges of values within the range. A sequential color map is one in which discrete colors are assigned to a sequence of sub-ranges of values covering the entire range.

An absolute color map is one in which the values to which colors are assigned are specified explicitly as values within the range. A fractional color map is one in which the values to which colors are assigned are specified as a fractional portion of the range. For example if the range is from -10.0 to 10.0, and the color red is to be assigned to atoms with a value of 5.0, then for an absolute color map the number 5.0 would be used. But for a fractional map, the number 0.75 would be used since 5.0 is 3/4 of the way from -10.0 to 10.0.

The *delta* setting must be specified for all styles, but is only used for the sequential style; otherwise the value is ignored. It specifies the bin size to use within the range for assigning consecutive colors to. For example, if the range is from -10.0 to 10.0 and a *delta* of 1.0 is used, then 20 colors will be assigned to the range. The first will be from $-10.0 \leq \mathrm{color1} < -9.0$, then second from $-9.0 \leq color2 < -8.0$, etc.

The *N* setting is how many entries follow. The format of the entries depends on whether the color map style is continuous, discrete or sequential. In all cases the *color* setting can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

For continuous color maps, each entry has a *value* and a *color*. The *value* is either a number within the range of values or *min* or *max*. The *value* of the first entry must be *min* and the *value* of the last entry must be *max*. Any entries in between must have increasing values. Note that numeric values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value *X* of its atom attribute. *X* will fall between 2 of the entry values. The color of the atom is linearly interpolated (in each of the RGB values) between the 2 colors associated with those entries. For example, if $X = -5.0$ and the two surrounding entries are "red" at -10.0 and "blue" at 0.0, then the atom's color will be halfway between "red" and "blue", which happens to be "purple".

For discrete color maps, each entry has a *lo* and *hi* value and a *color*. The *lo* and *hi* settings are either numbers within the range of values or *lo* can be *min* or *hi* can be *max*. The *lo* and *hi* settings of the last entry must be *min* and *max*. Other entries can have any *lo* and *hi* values and the sub-ranges of different values can overlap. Note that numeric *lo* and *hi* values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The entries are scanned from first to last. The first time that *lo* <= X <= *hi*, X is assigned the color associated with that entry. You can think of the last entry as assigning a default color (since it will always be matched by X), and the earlier entries as colors that override the default. Also note that no interpolation of a color RGB is done. All atoms will be drawn with one of the colors in the list of entries.

For sequential color maps, each entry has only a *color*. Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The range is partitioned into N bins of width *binsize*. Thus X will fall in a specific bin from 1 to N, say the Mth bin. If it falls on a boundary between 2 bins, it is considered to be in the higher of the 2 bins. Each bin is assigned a color from the E entries. If E < N, then the colors are repeated. For example if 2 entries with colors red and green are specified, then the odd numbered bins will be red and the even bins green. The color of the atom is the color of its bin. Note that the sequential color map is really a shorthand way of defining a discrete color map without having to specify where all the bin boundaries are.

Here is an example of using a sequential color map to color all the atoms in individual molecules with a different color. See the examples/pour/in.pour.2d.molecule input script for an example of how this is used.

```
variable        colors string &
               "red green blue yellow white &
```

(continues on next page)

(continued from previous page)

```
              purple pink orange lime gray"
variable        mol atom mol%10
dump            1 all image 250 image.*.jpg v_mol type &
              zoom 1.6 adiam 1.5
dump_modify     1 pad 5 amap 0 10 sa 1 10 ${colors}
```

In this case, 10 colors are defined, and molecule IDs are mapped to one of the colors, even if there are 1000s of molecules.

The *backcolor* sets the background color of the images. The color name can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

The *bcolor* keyword can be used with the dump image command, with its *bond* keyword, when its color setting is *type*, to set the color that bonds of each type will be drawn in the image.

The specified *type* should be a type label or integer from 1 to *N*, where *N* is the number of bond types. For numeric types, a wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of bond types. This takes the form "*" or "*n" or "m*" or "m*n". If *N* is the number of bond types, then an asterisk with no numerical values means all types from 1 to *N*. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from m to *N* (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character (e.g., red/green/blue). In the former case, that color is assigned to all the specified bond types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified bond types.

The *bdiam* keyword can be used with the dump image command, with its *bond* keyword, when its *diam* setting is *type*, to set the diameter that bonds of each type will be drawn in the image. The specified *type* should be a type label or integer from 1 to Nbondtypes. As with the *bcolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of numeric bond types. The specified *diam* is the size in whatever distance *units* you are using (e.g., Angstroms).

The *bitrate* keyword can be used with the *dump movie* command to define the size of the resulting movie file and its quality via setting how many kbits per second are to be used for the movie file. Higher bitrates require less compression and will result in higher quality movies. The quality is also determined by the compression format and encoder. The default setting is 2000 kbit/s, which will result in average quality with older compression formats.

Note

Not all movie file formats supported by dump movie allow the bitrate to be set. If not, the setting is silently ignored.

The *boxcolor* keyword sets the color of the simulation box drawn around the atoms in each image as well as the color of processor subdomain boundaries. See the "dump image box" command for how to specify that a box be drawn via the *box* keyword, and the subdomain boundaries via the *subbox* keyword. The color name can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

The *color* keyword allows definition of a new color name, in addition to the 140-predefined colors (see below), and associates three red/green/blue RGB values with that color name. The color name can then be used with any other dump_modify keyword that takes a color name as a value. The RGB values should each be floating point values between 0.0 and 1.0 inclusive.

When a color name is converted to RGB values, the user-defined color names are searched first, then the 140 pre-defined color names. This means you can also use the *color* keyword to overwrite one of the pre-defined color names with new RBG values.

The *framerate* keyword can be used with the *dump movie* command to define the duration of the resulting movie file. Movie files written by the dump *movie* command have a default frame rate of 24 frames per second and the images generated will be converted at that rate. Thus a sequence of 1000 dump images will result in a movie of about 42 seconds. To make a movie run longer you can either generate images more frequently or lower the frame rate. To speed a movie up, you can do the inverse. Using a frame rate higher than 24 is not recommended, as it will result in simply dropping the rendered images. It is more efficient to dump images less frequently.

The *gmap* keyword can be used with the dump image command, with its *grid* keyword, to setup a color map. The color map is used to assign a specific RGB (red/green/blue) color value to an individual grid cell when it is drawn, based on the grid cell value, which is a numeric quantity specified with the *grid* keyword.

The arguments for the *gmap* keyword are identical to those for the *amap* keyword (for atom coloring) described above.

9.17.6 Restrictions

To write JPEG images, you must use the -DLAMMPS_JPEG switch when building LAMMPS and link with a JPEG library. To write PNG images, you must use the -DLAMMPS_PNG switch when building LAMMPS and link with a PNG library.

To write *movie* dumps, you must use the -DLAMMPS_FFMPEG switch when building LAMMPS and have the FFmpeg executable available on the machine where LAMMPS is being run. Typically its name is lowercase (i.e., “ffmpeg”).

See the *Build settings* page for details.

Note that since FFmpeg is run as an external program via a pipe, LAMMPS has limited control over its execution and no knowledge about errors and warnings printed by it. Those warnings and error messages will be printed to the screen only. Due to the way image data are communicated to FFmpeg, it will often print the message

```
pipe:: Input/output error
```

which can be safely ignored. Other warnings and errors have to be addressed according to the FFmpeg documentation. One known issue is that certain movie file formats (e.g., MPEG level 1 and 2 format streams) have video bandwidth limits that can be crossed when rendering too large of image sizes. Typical warnings look like this:

```
[mpeg @ 0x98b5e0] packet too large, ignoring buffer limits to mux it
[mpeg @ 0x98b5e0] buffer underflow st=0 bufi=281407 size=285018
[mpeg @ 0x98b5e0] buffer underflow st=0 bufi=283448 size=285018
```

In this case it is recommended either to reduce the size of the image or to encode in a different format that is also supported by your copy of FFmpeg and which does not have this limitation (e.g., .avi, .mkv, mp4).

9.17.7 Related commands

dump, *dump_modify*, *undump*

9.17.8 Default

The defaults for the dump image and dump movie keywords are as follows:

- adiam = not specified (use diameter setting)
- atom = yes
- bond = none none (if no bonds in system)
- bond = atom 0.5 (if bonds in system)
- size = 512 512
- view = 60 30 (for 3d)
- view = 0 0 (for 2d)
- center = s 0.5 0.5 0.5
- up = 0 0 1 (for 3d)
- up = 0 1 0 (for 2d)
- zoom = 1.0
- box = yes 0.02
- axes = no 0.0 0.0
- subbox no 0.0
- shiny = 1.0
- ssao = no

The defaults for the dump_modify keywords specific to dump image and dump movie are as follows:

- acolor = * red/green/blue/yellow/aqua/cyan
- adiam = * 1.0
- amap = min max cf 0.0 2 min blue max red
- backcolor = black
- bcolor = * red/green/blue/yellow/aqua/cyan
- bdiam = * 0.5
- bitrate = 2000
- boxcolor = yellow
- color = 140 color names are pre-defined as listed below
- framerate = 24
- fsaa = no
- gmap = min max cf 0.0 2 min blue max red

These are the standard 109 element names that LAMMPS pre-defines for use with the dump image and dump_modify commands.

- 1-10 = “H”, “He”, “Li”, “Be”, “B”, “C”, “N”, “O”, “F”, “Ne”
- 11-20 = “Na”, “Mg”, “Al”, “Si”, “P”, “S”, “Cl”, “Ar”, “K”, “Ca”
- 21-30 = “Sc”, “Ti”, “V”, “Cr”, “Mn”, “Fe”, “Co”, “Ni”, “Cu”, “Zn”
- 31-40 = “Ga”, “Ge”, “As”, “Se”, “Br”, “Kr”, “Rb”, “Sr”, “Y”, “Zr”
- 41-50 = “Nb”, “Mo”, “Tc”, “Ru”, “Rh”, “Pd”, “Ag”, “Cd”, “In”, “Sn”
- 51-60 = “Sb”, “Te”, “I”, “Xe”, “Cs”, “Ba”, “La”, “Ce”, “Pr”, “Nd”
- 61-70 = “Pm”, “Sm”, “Eu”, “Gd”, “Tb”, “Dy”, “Ho”, “Er”, “Tm”, “Yb”
- 71-80 = “Lu”, “Hf”, “Ta”, “W”, “Re”, “Os”, “Ir”, “Pt”, “Au”, “Hg”
- 81-90 = “Tl”, “Pb”, “Bi”, “Po”, “At”, “Rn”, “Fr”, “Ra”, “Ac”, “Th”
- 91-100 = “Pa”, “U”, “Np”, “Pu”, “Am”, “Cm”, “Bk”, “Cf”, “Es”, “Fm”
- 101-109 = “Md”, “No”, “Lr”, “Rf”, “Db”, “Sg”, “Bh”, “Hs”, “Mt”

These are the 140 colors that LAMMPS pre-defines for use with the dump image and dump_modify commands. Additional colors can be defined with the dump_modify color command. The 3 numbers listed for each name are the RGB (red/green/blue) values. Divide each value by 255 to get the equivalent 0.0 to 1.0 value.

aliceblue = 240, 248, 255	antiquewhite = 250, 235, 215	aqua = 0, 255, 255	aquamarine = 127, 255, 212	azure = 240, 255, 255
beige = 245, 245, 220	bisque = 255, 228, 196	black = 0, 0, 0	blanchedalmond = 255, 255, 205	blue = 0, 0, 255
blueviolet = 138, 43, 226	brown = 165, 42, 42	burlywood = 222, 184, 135	cadetblue = 95, 158, 160	chartreuse = 127, 255, 0
chocolate = 210, 105, 30	coral = 255, 127, 80	cornflowerblue = 100, 149, 237	cornsilk = 255, 248, 220	crimson = 220, 20, 60
cyan = 0, 255, 255	darkblue = 0, 0, 139	darkcyan = 0, 139, 139	darkgoldenrod = 184, 134, 11	darkgray = 169, 169, 169
darkgreen = 0, 100, 0	darkkhaki = 189, 183, 107	darkmagenta = 139, 0, 139	darkolivegreen = 85, 107, 47	darkorange = 255, 140, 0
darkorchid = 153, 50, 204	darkred = 139, 0, 0	darksalmon = 233, 150, 122	darkseagreen = 143, 188, 143	darkslateblue = 72, 61, 139
darkslategray = 47, 79, 79	darkturquoise = 0, 206, 209	darkviolet = 148, 0, 211	deeppink = 255, 20, 147	deepskyblue = 0, 191, 255
dimgray = 105, 105, 105	dodgerblue = 30, 144, 255	firebrick = 178, 34, 34	floralwhite = 255, 250, 240	forestgreen = 34, 139, 34
fuchsia = 255, 0, 255	gainsboro = 220, 220, 220	ghostwhite = 248, 248, 255	gold = 255, 215, 0	goldenrod = 218, 165, 32
gray = 128, 128, 128	green = 0, 128, 0	greenyellow = 173, 255, 47	honeydew = 240, 255, 240	hotpink = 255, 105, 180
indianred = 205, 92, 92	indigo = 75, 0, 130	ivory = 255, 240, 240	khaki = 240, 230, 140	lavender = 230, 230, 250
lavenderblush = 255, 240, 245	lawngreen = 124, 252, 0	lemonchiffon = 255, 250, 205	lightblue = 173, 216, 230	lightcoral = 240, 128, 128
lightcyan = 224, 255, 255	lightgoldenrodyellow = 250, 250, 210	lightgreen = 144, 238, 144	lightgrey = 211, 211, 211	lightpink = 255, 182, 193
lightsalmon = 255, 160, 122	lightseagreen = 32, 178, 170	lightskyblue = 135, 206, 250	lightslategray = 119, 136, 153	lightsteelblue = 176, 196, 222
lightyellow = 255, 255, 224	lime = 0, 255, 0	limegreen = 50, 205, 50	linen = 250, 240, 230	magenta = 255, 0, 255
maroon = 128, 0, 0	mediumaquamarine = 102, 205, 170	mediumblue = 0, 0, 205	mediumorchid = 186, 85, 211	mediumpurple = 147, 112, 219
mediumseagreen = 60, 179, 113	mediumslateblue = 123, 104, 238	mediumspringgreen = 0, 250, 154	mediumturquoise = 72, 209, 204	mediumvioletred = 199, 21, 133
midnightblue = 25, 25, 112	mintcream = 245, 255, 250	mistyrose = 255, 228, 225	moccasin = 255, 228, 181	navajowhite = 255, 222, 173
navy = 0, 0, 128	oldlace = 253, 245, 230	olive = 128, 128, 0	olivedrab = 107, 142, 35	orange = 255, 165, 0
orangered = 255, 69, 0	orchid = 218, 112, 214	palegoldenrod = 238, 232, 170	palegreen = 152, 251, 152	paleturquoise = 175, 238, 238
palevioletred = 219, 112, 147	papayawhip = 255, 239, 213	peachpuff = 255, 239, 213	peru = 205, 133, 63	pink = 255, 192, 203
plum = 221, 160, 221	powderblue = 176, 224, 230	purple = 128, 0, 128	red = 255, 0, 0	rosybrown = 188, 143, 143
royalblue = 65, 105, 225	saddlebrown = 139, 69, 19	salmon = 250, 128, 114	sandybrown = 244, 164, 96	seagreen = 46, 139, 87
seashell = 255, 245, 238	sienna = 160, 82, 45	silver = 192, 192, 192	skyblue = 135, 206, 235	slateblue = 106, 90, 205
slategray = 112, 128, 144	snow = 255, 250, 250	springgreen = 0, 255, 127	steelblue = 70, 130, 180	tan = 210, 180, 140
teal = 0, 128, 128	thistle = 216, 191, 216	tomato = 253, 99, 71	turquoise = 64, 224, 208	violet = 238, 130, 238
wheat = 245, 222, 179	white = 255, 255, 255	whitesmoke = 245, 245, 245	yellow = 255, 255, 0	yellowgreen = 154, 205, 50

9.18 dump_modify command

9.19 dump_modify command for image/movie options

9.19.1 Syntax

```
dump_modify dump-ID keyword values ...
```

- dump-ID = ID of dump to modify
- one or more keyword/value pairs may be appended
- these keywords apply to various dump styles
- keyword = *append* or *at* or *balance* or *buffer* or *colname* or *delay* or *element* or *every* or *every/time* or *fileper* or *first* or *flush* or *format* or *header* or *image* or *label* or *maxfiles* or *nfile* or *pad* or *pbc* or *precision* or *region* or *refresh* or *scale* or *sfactor* or *skip* or *sort* or *tfactor* or *thermo* or *thresh* or *time* or *triclinic/general* or *types* or *units* or *unwrap*

```
append arg = yes or no
at arg = N
  N = index of frame written upon first dump
balance arg = yes or no
buffer arg = yes or no
colname values =  ID string, or default
  string = new column header name
  ID = integer from 1 to N, or integer from -1 to -N, where N = # of quantities being output
       or a custom dump keyword or reference to compute, fix, property or variable.
delay arg = Dstep
  Dstep = delay output until this timestep
element args = E1 E2 ... EN, where N = # of atom types
  E1,...,EN = element name (e.g., C or Fe or Ga)
every arg = N
  N = dump on timesteps which are a multiple of N
  N can be a variable (see below)
every/time arg = Delta
  Delta = dump once every Delta interval of simulation time (time units)
  Delta can be a variable (see below)
fileper arg = Np
  Np = write one file for every this many processors
first arg = yes or no
flush arg = yes or no
format args = line string, int string, float string, ID string, or none
  string = C-style format string
  ID = integer from 1 to N, or integer from -1 to -N, where N = # of quantities being output
       or a custom dump keyword or reference to compute, fix, property or variable.
header arg = yes or no
  yes to write the header
  no to not write the header
image arg = yes or no
label arg = string
  string = character string (e.g., BONDS) to use in header of dump local file
maxfiles arg = Fmax
  Fmax = keep only the most recent Fmax snapshots (one snapshot per file)
```

```
nfile arg = Nf
  Nf = write this many files, one from each of Nf processors
pad arg = Nchar = # of characters to convert timestep to
pbc arg = yes or no = remap atoms via periodic boundary conditions
precision arg = power-of-10 value from 10 to 1000000
region arg = region-ID or "none"
refresh arg = c_ID = compute ID that supports a refresh operation
scale arg = yes or no
sfactor arg = coordinate scaling factor (> 0.0)
skip arg = v_name
  v_name = variable with name which evaluates to non-zero (skip) or 0
sort arg = off or id or N or -N
   off = no sorting of per-atom lines within a snapshot
   id = sort per-atom lines by atom ID
   N = sort per-atom lines in ascending order by the Nth column
   -N = sort per-atom lines in descending order by the Nth column
tfactor arg = time scaling factor (> 0.0)
thermo arg = yes or no
thresh args = attribute operator value
  attribute = same attributes (x,fy,etotal,sxx,etc) used by dump custom style
  operator = "<" or "<=" or ">" or ">=" or "==" or "!=" or "|^"
  value = numeric value to compare to, or LAST
  these 3 args can be replaced by the word "none" to turn off thresholding
time arg = yes or no
triclinic/general arg = yes or no
types value = numeric or labels
units arg = yes or no
unwrap arg = yes or no
```

- these keywords apply only to the *image* and *movie* *styles*
- keyword = *acolor* or *adiam* or *amap* or *backcolor* or *bcolor* or *bdiam* or *boxcolor* or *color* or *bitrate* or *framerate*

  ```
  see the dump image doc page for details
  ```

- these keywords apply only to the */gz* and */zstd* dump styles
- keyword = *compression_level*

  ```
  compression_level args = level
    level = integer specifying the compression level that should be used (see below for supported levels)
  ```

- these keywords apply only to the */zstd* dump styles
- keyword = *checksum*

  ```
  checksum args = yes or no (add checksum at end of zst file)
  ```

- these keywords apply only to the vtk* dump style
- keyword = *binary*

  ```
  binary args = yes or no (select between binary and text mode VTK files)
  ```

9.19.2 Examples

```
dump_modify 1 format line "%d %d %20.15g %g %g" scale yes
dump_modify 1 format float %20.15g scale yes
dump_modify myDump image yes scale no flush yes
dump_modify 1 region mySphere thresh x < 0.0 thresh fx >= 3.2
dump_modify xtcdump precision 10000 sfactor 0.1
dump_modify 1 every 1000 nfile 20
dump_modify 1 every v_myVar
```

9.19.3 Description

Modify the parameters of a previously defined dump command. Not all parameters are relevant to all dump styles.

Unless otherwise noted, the following keywords apply to all the various dump styles, including the *dump image* and *dump movie* styles.

The *append* keyword applies to all dump styles except *cfg* and *xtc* and *dcd*. It also applies only to text output files, not to binary or gzipped or image/movie files. If specified as *yes*, then dump snapshots are appended to the end of an existing dump file. If specified as *no*, then a new dump file will be created which will overwrite an existing file with the same name.

The *at* keyword only applies to the *netcdf* dump style. It can only be used if the *append yes* keyword is also used. The *N* argument is the index of which frame to append to. A negative value can be specified for *N*, which means a frame counted from the end of the file. The *at* keyword can only be used if the dump_modify command is before the first command that causes dump snapshots to be output (e.g., a *run* or *minimize* command). Once the dump file has been opened, this keyword has no further effect.

The *buffer* keyword applies only to dump styles *atom*, *cfg*, *custom*, *local*, and *xyz*. It also applies only to text output files, not to binary or gzipped files. If specified as *yes*, which is the default, then each processor writes its output into an internal text buffer, which is then sent to the processor(s) which perform file writes, and written by those processors(s) as one large chunk of text. If specified as *no*, each processor sends its per-atom data in binary format to the processor(s) which perform file wirtes, and those processor(s) format and write it line by line into the output file.

The buffering mode is typically faster since each processor does the relatively expensive task of formatting the output for its own atoms. However it requires about twice the memory (per processor) for the extra buffering.

Added in version 4May2022.

The *colname* keyword can be used to change the default header keyword for dump styles: *atom*, *custom*, *cfg*, and *local* and their compressed, ADIOS variants. The setting for *ID string* replaces the default text with the provided string. *ID* can be a positive integer when it represents the column number counting from the left, a negative integer when it represents the column number from the right (i.e. -1 is the last column/keyword), or a custom dump keyword (or compute, fix, property, or variable reference) and then it replaces the string for that specific keyword. For *atom* dump styles only the keywords "id", "type", "x", "y", "z", "ix", "iy", "iz" can be accessed via string regardless of whether scaled or unwrapped coordinates were enabled or disabled, and it always assumes 8 columns for indexing regardless of whether image flags are enabled or not. For dump style *cfg* only changes to the "auxiliary" keywords (6th or later keyword) will become visible.

The *colname* keyword can be used multiple times. If multiple *colname* settings refer to the same keyword, the last setting has precedence. A setting of *default* clears all previous settings, reverting all values to their default names. Using the *scale* or *image* keyword will also reset all header keywords to their default values.

The *delay* keyword applies to all dump styles. No snapshots will be output until the specified *Dstep* timestep or later. Specifying *Dstep* < 0 is the same as turning off the delay setting. This is a way to turn off unwanted output early in a simulation, for example, during an equilibration phase.

The *element* keyword applies only to the dump *cfg*, *xyz*, and *image* styles. It associates element names (e.g., H, C, Fe) with LAMMPS atom types. See the list of element names at the bottom of this page.

In the case of dump *cfg*, this allows the AtomEye visualization package to read the dump file and render atoms with the appropriate size and color.

In the case of dump *image*, the output images will follow the same AtomEye convention. An element name is specified for each atom type (1 to Ntype) in the simulation. The same element name can be given to multiple atom types.

In the case of *xyz* format dumps, there are no restrictions to what label can be used as an element name. Any white-space separated text will be accepted.

The *every* keyword can be used with any dump style except the *dcd* and *xtc* styles. It specifies that the output of dump snapshots will now be performed on timesteps which are a multiple of a new *N* value, This overrides the dump frequency originally specified by the *dump* command.

The *every* keyword can be specified in one of two ways. It can be a numeric value in which case it must be > 0. Or it can be an *equal-style variable*, which should be specified as v_name, where name is the variable name.

In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a dump snapshot will be written out. On that timestep the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for *equal-style variables*, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for *equal-style variables*. Also see the next() function, which allows use of a file-style variable which reads successive values from a file, each time the variable is evaluated. Used with the *every* keyword, if the file contains a list of ascending timesteps, you can output snapshots whenever you wish.

Note that when using the variable option with the *every* keyword, you need to use the *first* option if you want an initial snapshot written to the dump file. The *every* keyword cannot be used with the dump *dcd* style.

For example, the following commands will write snapshots at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s first yes
```

The following commands would write snapshots at the timesteps listed in file tmp.times:

```
variable        f file tmp.times
variable        s equal next(f)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s
```

Note

When using a file-style variable with the *every* keyword, the file of timesteps must list a first timestep that is beyond the current timestep (e.g., it cannot be 0). And it must list one or more timesteps beyond the length of the run you perform. This is because the dump command will generate an error if the next timestep it reads from the file is not a value greater than the current timestep. Thus if you wanted output on steps 0,15,100 of a 100-timestep run, the file should contain the values 15,100,101 and you should also use the dump_modify first command. Any final value > 100 could be used in place of 101.

Added in version 7Jan2022.

The *every/time* keyword can be used with any dump style except the *dcd* and *xtc* styles. It changes the frequency of dump snapshots from being based on the current timestep to being determined by elapsed simulation time, i.e. in time units of the *units* command, and specifies *Delta* for the interval between snapshots. This can be useful when the timestep size varies during a simulation run, e.g. by use of the *fix dt/reset* command. The default is to perform output on timesteps which a multiples of specified timestep value *N*; see the *every* keyword.

The *every/time* keyword can be used with any dump style except the *dcd* and *xtc* styles. It does two things. It specifies that the interval between dump snapshots will be set in simulation time (i.e. in time units of the *units* command). This can be useful when the timestep size varies during a simulation run (e.g., by use of the *fix dt/reset* command). The default is to specify the interval in timesteps; see the *every* keyword. The *every/time* command also sets the interval value.

Note

If you wish dump styles *atom*, *custom*, *local*, or *xyz* to include the simulation time as a field in the header portion of each snapshot, you also need to use the dump_modify *time* keyword with a setting of *yes*. See its documentation below.

Note that since snapshots are output on simulation steps, each snapshot will be written on the first timestep whose associated simulation time is >= the exact snapshot time value.

As with the *every* option, the *Delta* value can be specified in one of two ways. It can be a numeric value in which case it must be > 0.0. Or it can be an *equal-style variable*, which should be specified as v_name, where name is the variable name.

In this case, the variable is evaluated at the beginning of a run to determine the next simulation time at which a dump snapshot will be written out. On that timestep the variable will be evaluated again to determine the next simulation time, etc. Thus the variable should return values in time units. Note the current timestep or simulation time can be used in an *equal-style variables* since they are both thermodynamic keywords. Also see the next() function, which allows use of a file-style variable which reads successive values from a file, each time the variable is evaluated. Used with the *every/time* keyword, if the file contains a list of ascending simulation times, you can output snapshots whenever you wish.

Note that when using the variable option with the *every/time* keyword, you need to use the *first* option if you want an initial snapshot written to the dump file. The *every/time* keyword cannot be used with the dump *dcd* style.

For example, the following commands will write snapshots at successive simulation times which grow by a factor of 1.5 with each interval. The dt value used in the variable is to avoid a zero result when the initial simulation time is 0.0.

```
variable        increase equal 1.5*(time+dt)
dump             1 all atom 100 tmp.dump
dump_modify      1 every/time v_increase first yes
```

The following commands would write snapshots at the times listed in file tmp.times:

```
variable        f file tmp.times
variable        s equal next(f)
dump             1 all atom 100 tmp.dump
dump_modify      1 every/time v_s
```

Note

When using a file-style variable with the *every/time* keyword, the file of timesteps must list a first time that is beyond the time associated with the current timestep (e.g., it cannot be 0.0). And it must list one or more times beyond the length of the run you perform. This is because the dump command will generate an error if the next time it reads from the file is not a value greater than the current time. Thus if you wanted output at times 0,15,100 of a run of length 100 in simulation time, the file should contain the values 15,100,101 and you should also use the dump_modify first command. Any final value > 100 could be used in place of 101.

The *first* keyword determines whether a dump snapshot is written on the very first timestep after the dump command is invoked. This will always occur if the current timestep is a multiple of N, the frequency specified in the *dump* command or *dump_modify every* command, including timestep 0. It will also always occur if the current simulation time is a multiple of *Delta*, the time interval specified in the *dump_modify every/time* command.

But if this is not the case, a dump snapshot will only be written if the setting of this keyword is *yes*. If it is *no*, which is the default, then it will not be written.

Note that if the argument to the *dump_modify every dump_modify every/time* commands is a variable and not a numeric value, then specifying *first yes* is the only way to write a dump snapshot on the first timestep after the dump command is invoked.

The *flush* keyword determines whether a flush operation is invoked after a dump snapshot is written to the dump file. A flush ensures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes. Flushes cannot be performed with dump style *xtc*.

The *format* keyword can be used to change the default numeric format output by the text-based dump styles: *atom*, *local*, *custom*, *cfg*, and *xyz* styles. Only the *line* or *none* options can be used with the *atom* and *xyz* styles.

All the specified format strings are C-style formats, such as used by the C/C++ printf() command. The *line* keyword takes a single argument which is the format string for an entire line of output for each atom (do not include a trailing "n"), with *N* fields, which you must enclose in quotes if there is more than one field. The *int* and *float* keywords take a single format argument and are applied to all integer or floating-point quantities output. The setting for *M string* also takes a single format argument which is used for the *M*th value output in each line (e.g., the fifth column is output in high precision by "format 5 %20.15g").

Note

When using the *line* keyword for the *cfg* style, the first two fields (atom ID and type) are not actually written into the CFG file, however you must include formats for them in the format string.

The *format* keyword can be used multiple times. The precedence is that for each value in a line of output, the *M* format (if specified) is used, else the *int* or *float* setting (if specified) is used, else the *line* setting (if specified) for that value is used, else the default setting is used. A setting of *none* clears all previous settings, reverting all values to their default format.

Note

Atom and molecule IDs are stored internally as 4-byte or 8-byte signed integers, depending on how LAMMPS was compiled. When specifying the *format int* option you can use a "%d"-style format identifier in the format string and LAMMPS will convert this to the corresponding 8-byte form if it is needed when outputting those values. However, when specifying the *line* option or *format M string* option for those values, you should specify a format string appropriate for an 8-byte signed integer (e.g., one with "%ld") if LAMMPS was compiled with the -DLAMMPS_BIGBIG option for 8-byte IDs.

Note

Any value written to a text-based dump file that is a per-atom quantity calculated by a *compute* or *fix* is stored internally as a floating-point value. If the value is actually an integer and you wish it to appear in the text dump file as a (large) integer, then you need to use an appropriate format. For example, these commands:

```
compute     1 all property/local batom1 batom2
dump        1 all local 100 tmp.bonds index c_1[1] c_1[2]
dump_modify 1 format line "%d %0.0f %0.0f"
```

will output the two atom IDs for atoms in each bond as integers. If the dump_modify command were omitted, they would appear as floating-point values, assuming they were large integers (more than six digits). The "index" keyword should use the "%d" format since it is not generated by a compute or fix, and is stored internally as an integer.

The *fileper* keyword is documented below with the *nfile* keyword.

The *header* keyword toggles whether the dump file will include a header. Excluding a header will reduce the size of the dump file for data produced by *pair tracker* or *bpm bond styles* which may not require the information typically written to the header.

The *image* keyword applies only to the dump *atom* style. If the image value is *yes*, three flags are appended to each atom's coords which are the absolute box image of the atom in each dimension. For example, an *x* image flag of −2 with a normalized coord of 0.5 means the atom is in the center of the box, but has passed through the box boundary twice and is really two box lengths to the left of its current coordinate. Note that for dump style *custom* these various values can be printed in the dump file by using the appropriate atom attributes in the dump command itself. Using this keyword will reset all custom header names set with *dump_modify colname* to their respective default values.

The *label* keyword applies only to the dump *local* style. When it writes local information, such as bond or angle topology to a dump file, it will use the specified *label* to format the header. By default this includes two lines:

```
ITEM: NUMBER OF ENTRIES
ITEM: ENTRIES ...
```

The word “ENTRIES” will be replaced with the string specified (e.g., BONDS or ANGLES).

The *maxfiles* keyword can only be used when a ‘*’ wildcard is included in the dump file name (i.e., when writing a new file(s) for each snapshot). The specified *Fmax* is how many snapshots will be kept. Once this number is reached, the file(s) containing the oldest snapshot is deleted before a new dump file is written. If the specified Fmax ≤ 0, then all files are retained.

This can be useful for debugging, especially if you do not know on what timestep something bad will happen (e.g., when LAMMPS will exit with an error). You can dump every time step and limit the number of dump files produced, even if you run for thousands of steps.

The *nfile* or *fileper* keywords can be used in conjunction with the “%” wildcard character in the specified dump file name, for all dump styles except the *dcd*, *image*, *movie*, *xtc*, and *xyz* styles (for which “%” is not allowed). As explained on the dump command doc page, the “%” character causes the dump file to be written in pieces, one piece for each of *P* processors. By default, *P* is the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set *P* to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets *P* to the specified N_f value. For example, if $N_f = 4$, and the simulation is running on 100 processors, four files will be written by processors 0, 25, 50, and 75. Each will collect information from itself and the next 24 processors and write it to a dump file.

For the *fileper* keyword, the specified value of N_p means write one file for every N_p processors. For example, if $N_p = 4$, every fourth processor (0, 4, 8, 12, etc.) will collect information from itself and the next three processors and write it to a dump file.

The *pad* keyword only applies when the dump filename is specified with a wildcard “*” character which becomes the timestep. If *pad* is 0, which is the default, the timestep is converted into a string of unpadded length (e.g., 100 or 12000 or 2000000). When *pad* is specified with *Nchar* > 0, the string is padded with leading zeroes so they are all the same length = *Nchar*. For example, pad 7 would yield 0000100, 0012000, 2000000. This can be useful so that post-processing programs can easily read the files in ascending timestep order.

The *pbc* keyword applies to all the dump styles. As explained on the dump doc page, atom coordinates in a dump file may be slightly outside the simulation box. This is because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, which will not typically coincide with the timesteps dump snapshots are written. If the setting of this keyword is set to *yes*, then all atoms will be remapped to the periodic box before the snapshot is written, then restored to their original position. If it is set to *no* they will not be. The *no* setting is the default because it requires no extra computation.

The *precision* keyword only applies to the dump *xtc* style. A specified value of *N* means that coordinates are stored to $1/N$ nanometer accuracy (e.g., for *N* = 1000, the coordinates are written to $1/1000$ nanometer accuracy).

The *refresh* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. It allows an “incremental” dump file to be written, by refreshing a compute that is used as a threshold for determining which atoms are included in a dump snapshot. The specified *c_ID* gives the ID of the compute. It is prefixed by “c_” to indicate a compute, which is the only current option. At some point, other options may be added (e.g., fixes or variables).

Note

This keyword can only be specified once for a dump. Refreshes of multiple computes cannot yet be performed.

The definition and motivation of an incremental dump file is as follows. Instead of outputting all atoms at each snapshot (with some associated values), you may only wish to output the subset of atoms with a value that has changed in some way compared to the value the last time that atom was output. In some scenarios this can result in a dramatically smaller dump file. If desired, by post-processing the sequence of snapshots, the values for all atoms at all timesteps can be inferred.

A concrete example is a simulation of atom diffusion in a solid, represented as atoms on a lattice. Diffusive hops are rare. Imagine that when a hop occurs an atom moves more than a distance *Dhop*. For any snapshot we only want to output atoms that have hopped since the last snapshot. This can be accomplished with something the following commands:

```
variable        Dhop equal 0.6
variable        check atom "c_dsp[4] > v_Dhop"
compute         dsp all displace/atom refresh check
dump            1 all custom 20 tmp.dump id type x y z
dump_modify     1 append yes thresh c_dsp[4] > ${Dhop} refresh c_dsp
```

The *compute displace/atom* command calculates the displacement of each atom from its reference position. The "4" index is the scalar displacement; 1, 2, and 3 are the *xyz* components of the displacement. The *dump_modify thresh* command will cause only atoms that have displaced more than 0.6 Å to be output on a given snapshot (assuming metal units). However, note that when an atom is output, we also need to update the reference position for that atom to its new coordinates. So that it will not be output in every snapshot thereafter. That reference position is stored by *compute displace/atom*. So the dump_modify *refresh* option triggers a call to compute displace/atom at the end of every dump to perform that update. The *refresh check* option shown as part of the *compute displace/atom* command enables the compute to respond to the call from the dump command, and update the appropriate reference positions. This is done be defining an *atom-style variable*, *check* in this example, which calculates a Boolean value (0 or 1) for each atom, based on the same criterion used by dump_modify thresh.

See the *compute displace/atom* command for more details, including an example of how to produce output that includes an initial snapshot with the reference position of all atoms.

Note that only computes with a *refresh* option will work with dump_modify refresh. See individual compute doc pages for details. Currently, only compute displace/atom supports this option. Others may be added at some point. If you use a compute that does not support refresh operations, LAMMPS will not complain; dump_modify refresh will simply do nothing.

The *region* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. If specified, only atoms in the region will be written to the dump file or included in the image/movie. Only one region can be applied as a filter (the last one specified). See the *region* command for more details. Note that a region can be defined as the "inside" or "outside" of a geometric shape, and it can be the "union" or "intersection" of a series of simpler regions.

The *scale* keyword applies only to the dump *atom* style. A scale value of *yes* means atom coords are written in normalized units from 0.0 to 1.0 in each box dimension. If the simulation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. A value of *no* means they are written in absolute distance units (e.g., Å or σ). Using this keyword will reset all custom header names set with *dump_modify colname* to their respective default values.

The *sfactor* and *tfactor* keywords only apply to the dump *xtc* style. They allow customization of the unit conversion factors used when writing to XTC files. By default, they are initialized for whatever *units* style is being used, to write out coordinates in nanometers and time in picoseconds. For example, for *real* units, LAMMPS defines *sfactor* = 0.1 and *tfactor* = 0.001, since the Å and fs used by *real* units are 0.1 nm and 0.001 ps, respectively. If you are using a units system with distance and time units far from nm and ps, you may wish to write XTC files with different units, since the compression algorithm used in XTC files is most effective when the typical magnitude of position data is between 10.0 and 0.1.

Added in version 15Sep2022.

The *skip* keyword can be used with all dump styles. It allows a dump snapshot to be skipped (not written to the dump file), if a condition is met. The condition is computed by an *equal-style variable*, which should be specified as v_name, where name is the variable name. If the variable evaluation returns a non-zero value, then the dump snapshot is skipped. If it returns zero, the dump proceeds as usual. Note that *equal-style variable* can contain Boolean operators which effectively evaluate as a true (non-zero) or false (zero) result.

The *skip* keyword can be useful for debugging purposes, e.g. to dump only on a particular timestep. Or to limit output to conditions of interest, e.g. only when the force on some atom exceeds a threshold value.

The *sort* keyword determines whether lines of per-atom output in a snapshot are sorted or not. A sort value of *off* means they will typically be written in indeterminate order, either in serial or parallel. This is the case even in serial if the *atom_modify sort* option is turned on, which it is by default, to improve performance. A sort value of *id* means sort the output by atom ID. A sort value of N or $-N$ means sort the output by the value in the Nth column of per-atom info in either ascending or descending order.

The dump *local* style cannot be sorted by atom ID, since there are typically multiple lines of output per atom. Some dump styles, such as *dcd* and *xtc*, require sorting by atom ID to format the output file correctly. If multiple processors are writing the dump file, via the "%" wildcard in the dump filename and the *nfile* or *fileper* keywords are set to non-default values (i.e., the number of dump file pieces is not equal to the number of procs), then sorting cannot be performed.

In a parallel run, the per-processor dump file pieces can have significant imbalance in number of lines of per-atom info. The *balance* keyword determines whether the number of lines in each processor snapshot are balanced to be nearly the same. A balance value of *no* means no balancing will be done, while *yes* means balancing will be performed. This balancing preserves dump sorting order. For a serial run, this option is ignored since the output is already balanced.

Note

Unless it is required by the dump style, sorting dump file output requires extra overhead in terms of CPU and communication cost, as well as memory, versus unsorted output.

The *thermo* keyword only applies the dump styles *netcdf* and *yaml*. It triggers writing of *thermo* information to the dump file alongside per-atom data. The values included in the dump file are cached values from the last thermo output and include the exact same the values as specified by the *thermo_style* command. Because these are cached values, they are only up-to-date when dump output is on a timestep that also has thermo output. Dump style *yaml* will skip thermo output on incompatible steps.

The *thresh* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. Multiple thresholds can be specified. Specifying *none* turns off all threshold criteria. If thresholds are specified, only atoms whose attributes meet all the threshold criteria are written to the dump file or included in the image. The possible attributes that can be tested for are the same as those that can be specified in the *dump custom* command, with the exception of the *element* attribute, since it is not a numeric value. Note that a different attributes can be used than those output by the *dump custom* command. For example, you can output the coordinates and stress of atoms whose energy is above some threshold.

If an atom-style variable is used as the attribute, then it can produce continuous numeric values or effective Boolean 0/1 values, which may be useful for the comparison operator. Boolean values can be generated by variable formulas that use comparison or Boolean math operators or special functions like gmask() and rmask() and grmask(). See the *variable* command page for details.

The specified value must be a simple numeric value or the word LAST. If LAST is used, it refers to the value of the attribute the last time the dump command was invoked to produce a snapshot. This is a way to only dump atoms whose attribute has changed (or not changed). Three examples follow.

```
dump_modify ... thresh ix != LAST
```

This will dump atoms which have crossed the periodic *x* boundary of the simulation box since the last dump. (Note that atoms that crossed once and then crossed back between the two dump timesteps would not be included.)

```
region foo sphere 10 20 10 15
variable inregion atom rmask(foo)
dump_modify ... thresh v_inregion |^ LAST
```

This will dump atoms which crossed the boundary of the spherical region since the last dump.

```
variable charge atom "(q > 0.5) || (q < -0.5)"
dump_modify ... thresh v_charge |^ LAST
```

This will dump atoms whose charge has changed from an absolute value less than $\frac{1}{2}$ to greater than $\frac{1}{2}$ (or vice versa) since the last dump (e.g., due to reactions and subsequent charge equilibration in a reactive force field).

The choice of operators listed above are the usual comparison operators. The XOR operation (exclusive or) is also included as "|^". In this context, XOR means that if either the attribute or value is 0.0 and the other is non-zero, then the result is "true" and the threshold criterion is met. Otherwise it is not met.

Note

For style *custom*, the *triclinic/general* keyword can alter dump output for general triclinic simulation boxes and their atoms. See the *dump* command for details of how this changes the format of dump file snapshots. The thresh keyword may access per-atom attributes either directly or indirectly through a compute or variable. If the attribute is an atom coordinate or a per-atom vector (such as velocity, force, or dipole moment), its value will *NOT* be a general triclinic (rotated) value. Rather it will be a restricted triclinic value.

The *time* keyword only applies to the dump *atom*, *custom*, *local*, and *xyz* styles (and their COMPRESS package versions *atom/gz*, *custom/gz* and *local/gz*). For the first three styles, if set to *yes*, each frame will will contain two extra lines before the "ITEM: TIMESTEP" entry:

```
ITEM: TIME
<elapsed time>
```

For the *xyz* style, the simulation time is included on the same line as the timestep value.

This will output the current elapsed simulation time in current time units equivalent to the *thermo keyword time*. This is to simplify post-processing of trajectories using a variable time step (e.g., when using *fix dt/reset*). The default setting is *no*.

The *types* keyword applies only to the dump xyz style. If this keyword is used with a value of *numeric*, then numeric atom types are printed in the xyz file (default). If the value *labels* is specified, then *type labels* are printed for atom types.

The *triclinic/general* keyword only applies to the dump *atom* and *custom* styles. It can only be used with a value of *yes* if the simulation box was created as a general triclinic box. See the *Howto_triclinic* doc page for a detailed explanation of orthogonal, restricted triclinic, and general triclinic simulation boxes.

If this keyword is used with a value of *yes*, the box information at the beginning of each snapshot will include information about the 3 arbitrary edge vectors **A**, **B**, **C** that define the general triclinic box as well as their origin. The format is described on the *dump* doc page.

The coordinates of each atom will likewise be output as values in (or near) the general triclinic box. Likewise, per-atom vector quantities such as velocity, omega, dipole moment, etc will have orientations consistent with the general triclinic box, meaning they will be rotated relative to the standard xyz coordinate axes. See the *dump* doc page for a full list of which dump attributes this affects.

The *units* keyword only applies to the dump *atom*, *custom*, and *local* styles (and their COMPRESS package versions *atom/gz*, *custom/gz* and *local/gz*). If set to *yes*, each individual dump file will contain two extra lines at the very beginning with:

ITEM: UNITS
<units style>

This will output the current selected *units* style to the dump file and thus allows visualization and post-processing tools to determine the choice of units of the data in the dump file. The default setting is *no*.

The *unwrap* keyword only applies to the dump *dcd* and *xtc* styles. If set to *yes*, coordinates will be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *COMPRESS package* offers both GZ and Zstd compression variants of styles atom, custom, local, cfg, and xyz. When using these styles the compression level can be controlled by the compression_level keyword. File names with these styles have to end in either .gz or .zst.

GZ supports compression levels from -1 (default), 0 (no compression), and 1 to 9, 9 being the best compression. The COMPRESS /gz styles use 9 as default compression level.

Zstd offers a wider range of compression levels, including negative levels that sacrifice compression for performance. 0 is the default, positive levels are 1 to 22, with 22 being the most expensive compression. Zstd promises higher compression/decompression speeds for similar compression ratios. For more details see *https://facebook.github.io/zstd/*.

In addition, Zstd compressed files can include a checksum of the entire contents. The Zstd enabled dump styles enable this feature by default and it can be disabled with the checksum keyword.

The *VTK package* offers writing dump files in VTK file formats that can be read by a variety of visualization tools based on the VTK library. These VTK files follow naming conventions that collide with the LAMMPS convention to append ".bin" to a file name in order to switch to a binary output. Thus for *vtk style dumps* the dump_modify command supports the keyword *binary* which selects between generating text mode and binary style VTK files.

9.19.4 Restrictions

Not all *dump_modify* options can be applied to all dump styles. Details are in the discussions of the individual options.

9.19.5 Related commands

dump, *dump image*, *undump*

9.19.6 Default

The option defaults are

- append = no
- balance = no
- buffer = yes for dump styles *atom*, *custom*, *loca*, and *xyz*
- element = "C" for every atom type
- every = whatever it was set to via the *dump* command
- fileper = # of processors
- first = no
- flush = yes
- format = %d and %g for each integer or floating point value
- image = no
- label = ENTRIES
- maxfiles = -1
- nfile = 1
- pad = 0
- pbc = no
- precision = 1000
- region = none
- scale = yes
- sort = off for dump styles *atom*, *custom*, *cfg*, and *local*
- sort = id for dump styles *dcd*, *xtc*, and *xyz*
- thresh = none
- time = no
- triclinic/general = no
- types = numeric
- units = no
- unwrap = no
- compression_level = 9 (gz variants)

- compression_level = 0 (zstd variants)
- checksum = yes (zstd variants)

9.20 dump molfile command

9.20.1 Syntax

```
dump ID group-ID molfile N file format path
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- molfile = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the *dump* doc page)
- N = dump every this many timesteps
- file = name of file to write to
- format = file format to be used
- path = file path with plugins (optional)

9.20.2 Examples

```
dump mf1 all molfile 10 melt1.xml hoomd
dump mf2 all molfile 10 melt2-*.pdb pdb .
dump mf3 all molfile 50 melt3.xyz xyz .:/home/akohlmey/vmd/plugins/LINUX/molfile
```

9.20.3 Description

Dump a snapshot of atom coordinates and selected additional quantities to one or more files every N timesteps in one of several formats. Only information for atoms in the specified group is dumped. This specific dump style uses molfile plugins that are bundled with the VMD molecular visualization and analysis program.

Unless the filename contains a * character, the output will be written to one single file with the specified format. Otherwise there will be one file per snapshot and the * will be replaced by the time step number when the snapshot is written.

Note

Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

The molfile plugin API has a few restrictions that have to be honored by this dump style: the number of atoms must not change, the atoms must be sorted, outside of the coordinates no change in atom properties (like type, mass, charge) will be recorded.

The *format* keyword determines what format is used to write out the dump. For this to work, LAMMPS must be able to find and load a compatible molfile plugin that supports this format. Settings made via the *dump_modify* command can alter per atom properties like element names.

The *path* keyword determines which in directories. This is a "path" like other search paths, i.e. it can contain multiple directories separated by a colon (or semicolon on Windows). This keyword is optional and default to ".", the current directory.

The *unwrap* option of the *dump_modify* command allows coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the *dump_modify first* command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the *dump_modify every* command. The *dump_modify every* command also allows a variable to be used to determine the sequence of timesteps on which dump files are written.

9.20.4 Restrictions

The *molfile* dump style is part of the MOLFILE package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

Molfile plugins provide a consistent programming interface to read and write file formats commonly used in molecular simulations. The MOLFILE package only provides the interface code, not the plugins. These can be obtained from a VMD installation which has to match the platform that you are using to compile LAMMPS for. By adding plugins to VMD, support for new file formats can be added to LAMMPS (or VMD or other programs that use them) without having to re-compile the application itself. The plugins are installed in the directory: <VMDHOME>/plugins/<VMDARCH>/molfile

Note

while the programming interface (API) to the plugins is backward compatible, the binary interface (ABI) has been changing over time, so it is necessary to compile this package with the plugin header files from VMD that match the binary plugins. These header files in the directory: <VMDHOME>/plugins/include For convenience, the package ships with a set of header files that are compatible with VMD 1.9 and 1.9.1 (June 2012)

9.20.5 Related commands

dump, *dump_modify*, *undump*

9.20.6 Default

The default path is ".". All other properties have to be specified.

9.21 dump netcdf command

9.22 dump netcdf/mpiio command

9.22.1 Syntax

```
dump ID group-ID netcdf N file args
dump ID group-ID netcdf/mpiio N file args
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- *netcdf* or *netcdf/mpiio* = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the *dump* doc page)
- N = dump every this many timesteps
- file = name of file to write dump info to
- args = list of atom attributes, same as for *dump_style custom*

9.22.2 Examples

```
dump 1 all netcdf 100 traj.nc type x y z vx vy vz
dump_modify 1 append yes at -1 thermo yes
dump 1 all netcdf/mpiio 1000 traj.nc id type x y z
dump 1 all netcdf 1000 traj.*.nc id type x y z
```

9.22.3 Description

Dump a snapshot of atom coordinates every N timesteps in Amber-style NetCDF file format. NetCDF files are binary, portable and self-describing. This dump style will write only one file on the root node. The dump style *netcdf* uses the standard NetCDF library. All data is collected on one processor and then written to the dump file. Dump style *netcdf/mpiio* uses the parallel NetCDF library and MPI-IO to write to the dump file in parallel; it has better performance on a larger number of processors. Note that style *netcdf* outputs all atoms sorted by atom tag while style *netcdf/mpiio* outputs atoms in order of their MPI rank.

NetCDF files can be directly visualized via the following tools:

- Ovito (https://www.ovito.org/). Ovito supports the AMBER convention and all extensions of this dump style.
- VMD (https://www.ks.uiuc.edu/Research/vmd/).

In addition to per-atom data, *thermo* data can be included in the dump file. The data included in the dump file is identical to the data specified by *thermo_style*.

9.22.4 Restrictions

The *netcdf* and *netcdf/mpiio* dump styles are part of the NETCDF package. They are only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

The *netcdf* and *netcdf/mpiio* dump styles currently cannot dump string properties or properties from variables.

9.22.5 Related commands

dump, *dump_modify*, *undump*

9.23 dump vtk command

9.23.1 Syntax

```
dump ID group-ID vtk N file args
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be dumped
- vtk = style of dump command (other styles such as *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the *dump* doc page)
- N = dump every this many timesteps
- file = name of file to write dump info to
- args = same as arguments for *dump_style custom*

9.23.2 Examples

```
dump dmpvtk all vtk 100 dump*.myforce.vtk id type vx fx
dump dmpvtp flow vtk 100 dump*.%.displace.vtp id type c_myD[1] c_myD[2] c_myD[3] v_ke
```

9.23.3 Description

Dump a snapshot of atom quantities to one or more files every *N* timesteps in a format readable by the VTK visualization toolkit or other visualization tools that use it, such as ParaView. The time steps on which dump output is written can also be controlled by a variable; see the *dump_modify every* command for details.

This dump style is similar to *dump_style custom* but uses the VTK library to write data to VTK simple legacy or XML format, depending on the filename extension specified for the dump file. This can be either **.vtk* for the legacy format or **.vtp* and **.vtu*, respectively, for XML format; see the VTK homepage for a detailed description of these formats.

Since this naming convention conflicts with the way binary output is usually specified (see below), the *dump_modify binary* command allows setting of a binary option for this dump style explicitly.

Only information for atoms in the specified group is dumped. The *dump_modify thresh and region* commands can also alter what atoms are included; see details below.

As described below, special characters ("*", "%") in the filename determine the kind of output.

Warning

Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

Warning

Unless the *dump_modify sort* option is invoked, the lines of atom information written to dump files will be in an indeterminate order for each snapshot. This is even true when running on a single processor, if the *atom_modify sort* option is on, which it is by default. In this case atoms are re-ordered periodically during a simulation, due to spatial sorting. It is also true when running in parallel, because data for a single snapshot is collected from multiple processors, each of which owns a subset of the atoms.

For the *vtk* style, sorting is off by default. See the *dump_modify* page for details.

The dimensions of the simulation box are written to a separate file for each snapshot (either in legacy VTK or XML format depending on the format of the main dump file) with the suffix *_boundingBox* appended to the given dump filename.

For an orthogonal simulation box this information is saved as a rectilinear grid (legacy .vtk or .vtr XML format).

Triclinic simulation boxes (non-orthogonal) are saved as hexahedrons in either legacy .vtk or .vtu XML format.

Style *vtk* allows you to specify a list of atom attributes to be written to the dump file for each atom. The list of possible attributes is the same as for the *dump_style custom* command; see its documentation page for a listing and an explanation of each attribute.

Note

Since position data is required to write VTK files the atom attributes "x y z" do not have to be specified explicitly; they will be included in the dump file regardless. Also, in contrast to the *custom* style, the specified *vtk* attributes are rearranged to ensure correct ordering of vector components (except for computes and fixes - these have to be given in the right order) and duplicate entries are removed.

The VTK format uses a single snapshot of the system per file, thus a wildcard "*" must be included in the filename, as discussed below. Otherwise the dump files will get overwritten with the new snapshot each time.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the *dump_modify first* command, which can also be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the *dump_modify every* command. The *dump_modify* *every* command also allows a variable to be used to determine the sequence of timesteps on which dump files are written. In this mode a dump on the first timestep of a run will also not be written unless the *dump_modify first* command is used.

Dump filenames can contain two wildcard characters. If a "*" character appears in the filename, then one file per snapshot is written and the "*" character is replaced with the timestep value. For example, tmp.dump*.vtk becomes tmp.dump0.vtk, tmp.dump10000.vtk, tmp.dump20000.vtk, etc. Note that the *dump_modify pad* command can be used to ensure all timestep numbers are the same length (e.g. 00010), which can make it easier to read a series of dump files in order with some post-processing tools.

If a "%" character appears in the filename, then each of P processors writes a portion of the dump file, and the "%" character is replaced with the processor ID from 0 to P-1 preceded by an underscore character. For example, tmp.dump%.vtp becomes tmp.dump_0.vtp, tmp.dump_1.vtp, … tmp.dump_P-1.vtp, etc. This creates smaller files and can be a fast mode of output on parallel machines that support parallel I/O for output.

By default, P = the number of processors meaning one file per processor, but P can be set to a smaller value via the *nfile* or *fileper* keywords of the *dump_modify* command. These options can be the most efficient way of writing out dump files when running on large numbers of processors.

For the legacy VTK format "%" is ignored and P = 1, i.e., only processor 0 does write files.

Note that using the "*" and "%" characters together can produce a large number of small dump files!

If *dump_modify binary* is used, the dump file (or files, if "*" or "%" is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster.

9.23.4 Restrictions

The *vtk* style does not support writing of gzipped dump files.

The *vtk* dump style is part of the VTK package. It is only enabled if LAMMPS was built with that package. See the *Build package* page for more info.

To use this dump style, you also must link to the VTK library. See the info in lib/vtk/README and ensure the Makefile.lammps file in that directory is appropriate for your machine.

The *vtk* dump style supports neither buffering or custom format strings.

9.23.5 Related commands

dump, *dump image*, *dump_modify*, *undump*

9.23.6 Default

By default, files are written in ASCII format. If the file extension is not one of .vtk, .vtp or .vtu, the legacy VTK file format is used.