SPHGR Documentation

Release 1.0

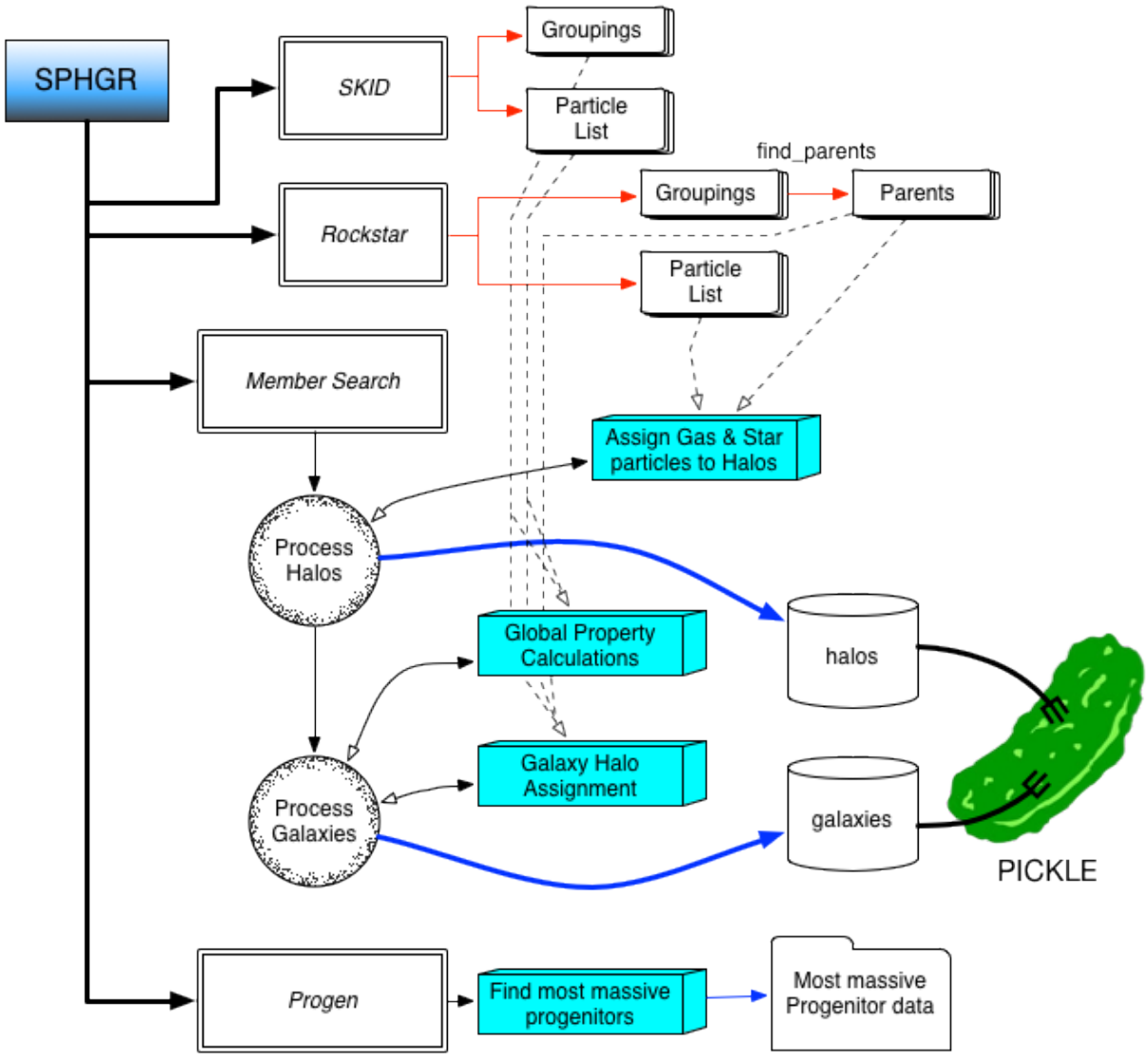
Robert Thompson

August 03, 2015

Contents

This is a multi-purpose data reduction suite designed to read and then process `GADGET`, `TIPSY`, or `HDF5` binary outputs from `GADGET`. Then general idea is to be a one-stop-shop to simulation analysis and data access. The resulting data is saved to the disk in distinct object files which allows for easy examination/plotting. The general operation of the code looks something like this:

1. run galaxy finder
2. run DM halo finder
3. run `membersearch`:
   - assign gas & star particles to halos
   - assign DM particles to halos
   - calculate galaxy global properties
   - assign galaxies to DM halos
   - determine central/satellite
4. find most-massive progenitors of each galaxy
5. run `LOSER` to determine magnitudes


SPHGR
SKID
Groupings
Particle List
find_parents
Rockstar
Groupings
Parents
Particle List
Member Search
Assign Gas & Star particles to Halos
Process Halos
Global Property Calculations
halos
Galaxy Halo Assignment
Process Galaxies
galaxies
PICKLE
Progen
Find most massive progenitors
Most massive Progenitor data

CHAPTER 1

Contents

1.1 Getting Started

1.1.1 Requirements

- **python**
  - numpy
  - scipy
  - cython
  - cPickle
  - h5py
  - pygadgetreader - (*custom*)
  - mpi4py - (*optional*)
  - python-vtk - (*optional*, needed for visualization)
- **DM-halo group finder** - (see *Plugin Support*)
- **Baryonic group finder** - (see *Plugin Support*)
- **MPI** - (*optional*, http://www.open-mpi.org)
- **HDF5** - (*optional*, http://www.hdfgroup.org/HDF5)

Plugin Support

`SPHGR` is configured by default to work with `Rockstar-Galaxies` (DM-halo groups incl. gas&stars) and `SKID` (baryonic groupings). You must download a custom version of RS-G whose link is below.

1.1.2 Compilation

In this section I will briefly describe how to configure your system for a default `SPHGR` run. This assumes you are using `Rockstar-Galaxies` & `SKID` as your DM+Baryon group finders. But first, `python`!

python

You will need a working `python` installation that includes `numpy`, `cython`, `scipy`, and `cPickle`. Most all-in-one distros (EPD/anaconda) should come with most of these components. You can also install `mpi4py` if you want to run the data analysis in parallel.

pyGadgetReader

`SPHGR` reads gadget files via `pygadgetreader`; this is a custom `python` backend that allows you to easily read in `Gadget` files. It currently supports `Gadget` type 1, `TIPSY`, and `HDF5` binary files. Download pygadgetreader from bitbucket and compile & install via the following commands:

```
> python setup.py install
```

See the ecnlosed readme within `pygadgetreader` for further details or installation tips/suggestions.

ROCKSTAR-GALAXIES

Currently, you must download a modified version which allows for `SPHGR` to do it's job much quicker. The custom fork can be found via this bitbucket link. We then need to compile both `rockstar-galaxies` and the `parents` utility via:

```
> make
> make parents
```

HDF5: edit the `Makefile` and ensure that you specify your `HDF5 include` and `lib` paths in the `HDF5_FLAGS` variable. Then compile in the following manner:

```
> make with_hdf5
> make parents
```

SKID

In order to use `SKID` you must download the customized version found here; it has been modified to read Gadget type1-binaries and HDF5 files directly. Modify the `Makefile` and comment/uncomment `-DGADGET` and/or `-DHDF5` depending on what file types you will be using. The compilation should then be as simple as:

```
> make
```

HDF5: ensure that the `HDF5INCL` and `HDF5LIB` directories within the `Makefile` point to your `HDF5` installtion before compilation.

1.1.3 Configuration

Before we move on to script configurations, we need to acomplish two tasks:

1. copy over template scripts

   I have written a small script in the root directory called `setup_templates.py`. Run this once, and it will put the user configurable `configs/config.py`, `configs/sim_config.py`, and `reduction.py` scripts in place. Further updates to these files will be located in the `configs/templates` directory.

2. build the `cython` modules.

   I have included a script called `make_extensions.sh` in the root directory of `SPHGR`. If cython is properly installed and up to date this should run without a hitch from the suite's root directory.

Now we can start configuring the configuration files located in the configs folder. The ones we will be focusing on is `configs/config.py` & `configs/sim_config.py`.

configs/config.py

This file contains global parameters related to running the code. The first few variables should be fairly self explanatory. But the most important ones are the variables SKID, RS, RS_FP, MPI, and PY. Edit these so that each points directly to the executable of the respective program.

note: If you have both `MPI` and `mpi4py` set USE_MPI=1 to allow for parallel analysis.

configs/sim_config.py

This file is slightly tricky; it holds the information needed to find your snapshots. The code allows for the customization of how your snaps are named and stored. The first thing you'll want to edit is the `SIMBASEDIR` variable which farther down in the file; this tells the code where to start looking. `SNAPDIR` dictates how your snapshot directory is formatted, and `SNAPNAME` dictates how your snapshot files are formatted. The easiest way to illustrate how this works is by example. If we have the following configuration:

```
SIMBASEDIR    = '/Users/bob'
SNAPDIR       = '%s/%s'
SNAPDIR_OPTS  = '(%s,%s)' % (DIR_OPTS.get(0), DIR_OPTS.get(1))
SNAPNAME      = '%s/snap_%s_%03d'
SNAPNAME_OPTS = '(%s,%s,%s)' % (SNAP_OPTS.get(0),SNAP_OPTS.get(1),SNAP_OPTS.get(2))
```

then when the code is executed the `SNAPDIR` and `SNAPNAME` variables will be defined like so:

```
SNAPDIR  = '/Users/bob/%s'    % (SNAPPRE)
SNAPNAME = '%s/snap_%s_%03d' % (SNAPDIR,SNAPPRE,SNAPNUM)
```

You will define `SNAPPRE` and `SNAPNUM` later when actually executing the code so don't worry about these for now. What you will notice is that I set *SNAPDIR='%s/%s'* meaning that it will be composed of two strings (string/string). These strings are set via `SNAPDIR_OPTS`, the `.get(N)` values are taken from the dict at the top of the `configs/sim_config.py` file:

```
SNAP_OPTS = {0:'SNAPDIRS[i]', 1:'SNAPPRE', 2:'SNAPNUMS[j]'}
DIR_OPTS  = {0:'SIMBASEDIR', 1:'SNAPPRE', 2:'DIRPRES[i]'}
```

These dictionaries use the `.get()` function to return the string for a given index. I know this is a bit confusing at first, but it was the only way I could implement it so that your snapshot locations and names were completely customizable. It may take some tinkering, but once you set it up you should be good to go.

The `DIRPRES` variable allows for you to stick snapshots with the same `SNAPPRE` under different subdirectories. Below is additional explanations for each property within the above dict:

- **SNAP_OPTS:**
  0. SNAPDIRS[i] = represents snapshot directories
  1. SNAPPRE = snap prefix, usually N256L16 or similar (specified later)
  2. SNAPNUMS[j] = snapshot numbers (specified later)

- **DIR_OPTS:**
    0. SIMBASEDIR = base snapshot directory - specified in sim_config.py
    1. SNAPPRE = snap prefix, usually N256L16 or similar (specified later)
    2. DIRPRES[i] = prefix for subdirectories

Next we need to define `SNAPPRE` and `DIRPRES`. The first should be fairly self explanatory, the second specifies any sub directories that may exist within your snapshot directory. If you do not have any subdirectories under your snapshot folder simply set:

```
DIRPRES = ['.']
```

Last is the parameters *ZOOM_RES* and *ZOOM_LOWRES**. The first sets the effective resolutuion of your simulation if it is detected to be a zoom; don't worry about this for cosmological boxes. The second tells `SPHGR` which particles to consider low-resolution elements. The number is calculated via 2^particleType+2^particleType, so if particle types 2,3,&5 are low-resolution particles we would set this value to 2^2+2^3+2^5=44.

EXAMPLE

Let's say you are working with a simulation snapshot whose full path is the following:

```
/home/rthompson/N512L32/snap_N512L32_050
```

This is what the `configs/sim_config.py` variables should look like for this particular sim:

```
SIMBASEDIR    = '/home/rthompson'
SNAPDIR       = '%s/%s'
SNAPDIR_OPTS  = '(%s,%s)' % (DIR_OPTS.get(0), DIR_OPTS.get(1))
SNAPNAME      = '%s/snap_%s_%03d'
SNAPNAME_OPTS = '(%s,%s,%s)' % (SNAP_OPTS.get(0),SNAP_OPTS.get(1),SNAP_OPTS.get(2))
SNAPPRE       = ['N512L32']
DIRPRES       = ['.']
```

reduction.py

Once `configs/config.py` and `configs/sim_config.py` are edited appropriately we need to setup the reduction script parameters.

- *BEG/END:* this determines what snap numbers the code will analyze. When working with a linear series of snapshots, say 0-100, set BEG=0, END=100. If you are working with a single snapshot set `BEG=[N]` and `END` will be ignored. Lastly, if your snapshots are *not* linearly spaced then you can set `BEG=[a,c,k,l]` and it will again ignore the value specified for `END`.
- *SKIPRAN:* code will skip the analysis if the result already exists. 0=do not skip, 1=skip
- *PROMPT:* code will prompt if files are not found, useful to disable if submitting jobs.
- *RUN_XXX:* allows you to switch on/off specific analysis processes. 0=do not run, 1=run.
    - *SKID* - galaxy group finding
    - *ROCKSTAR* - halo finding
    - *MEMBERSEARCH* - main analysis
    - *PROGEN* - most massive progenitor search
    - *LOSER* - Romeel's LOSER code
- *MPI_NP:* Sets the number of processors to use for parallel analysis via `mpi4py`.

- *OMP_NP:* Number of threads to spawn for misc. calculations

NOTE: Only use PROGEN if you are planning on examining a sequence of snapshots, it requires more than one to work as it leapfrogs backwards looking for the previous sequential snapshot (SNAPNUM-1).

1.1.4 Running

Once the above steps are complete you can execute the reduction script via:

```
> python reduction.py
```

and `python` should take care of the rest. Once this process is complete most of the analysis should take place via scripts located in the `analysis` directory.

1.2 Plot Routines

`progenner.`**`galaxy_pool_helper`**(*args*)

`progenner.`**`getGalaxyProgens`**(*obj*, *galID*)

`progenner.`**`getHaloProgens`**(*obj*, *halID*)

`progenner.`**`halo_pool_helper`**(*args*)

`progenner.`**`multiProgenInit`**(*objs*, *SN1*, *SN2*, *IDs*, *progenType*, *NPROCS=None*, *printFreq=60.0*, *parentsOnly=1*)

`progenner.`**`progenInit`**(*objs*, *SN1*, *SN2*, *IDs*, *progenType*, *printer=1*, *parentsOnly=1*)

Function used to initialize progen data.

Parameters **objs** : list

List of SPHGR objects

SN1 : int

Starting snapnum

SN2 : int

Ending snapnum (typically 0)

IDs : list

List of galaxy or halo indexes of interest

progenType : string

specify ‘galaxy’ or ‘halo’

Notes

This function does not return anything. However, it does add the following information to the root of your object (data not saved to disk):

•obj.galaxies[n].progen_z

•obj.galaxies[n].progen_indexes

•obj.galaxies[n].progen_central

- obj.galaxies[n].progen_cmx
- obj.galaxies[n].progen_cmy
- obj.galaxies[n].progen_cmz
- obj.galaxies[n].progen_HMR
- obj.galaxies[n].progen_FMR
- obj.galaxies[n].progen_SFR
- obj.galaxies[n].progen_gas_mass
- obj.galaxies[n].progen_stellar_mass
- obj.galaxies[n].progen_total_mass
- obj.galaxies[n].progen_halo_mass
- obj.galaxies[n].progen_halo_gmass
- obj.galaxies[n].progen_halo_smass
- obj.galaxies[n].progen_gHMR
- obj.galaxies[n].progen_gFMR
- obj.galaxies[n].progen_sHMR
- obj.galaxies[n].progen_h2HMR
- obj.galaxies[n].progen_indexes2
- obj.galaxies[n].progen_central2
- obj.galaxies[n].progen_cmx2
- obj.galaxies[n].progen_cmy2
- obj.galaxies[n].progen_cmz2
- obj.galaxies[n].progen_HMR2
- obj.galaxies[n].progen_FMR2
- obj.galaxies[n].progen_SFR2
- obj.galaxies[n].progen_gas_mass2
- obj.galaxies[n].progen_stellar_mass2
- obj.galaxies[n].progen_total_mass2
- obj.galaxies[n].progen_halo_mass2
- obj.galaxies[n].progen_halo_gmass2
- obj.galaxies[n].progen_halo_smass2

- obj.halos[n].progen_z = z
- obj.halos[n].progen_r = r
- obj.halos[n].progen_m = m

Examples

```
>>> import initSnap as iS
>>> pList = iS.initSnap(SNAPNUM1,SNAPNUM1)
>>> sims  = []
>>> for i in range(0,len(pList)):
>>>    sims.append(iS.loadPickle(pList[i]))
>>> progenInit(sims,SNAPNUM1,SNAPNUM2,'galaxy')
```

class progenner.**progenPlot**(*objs*, *galIDs*, *COLORS*, *LABELS*)

Bases: object

Class to assist in the plotting of PROGEN data.

Parameters **objs** : list of SPHGR objects

galIDs : list of galaxy IDs to plot

COLORS : list of colors

LABELS : list of labels

Methods

makeplot(objs, galIDs, COLORS, LABELS)

makeplot(*objs*, *galIDs*, *COLORS*, *LABELS*)

plt

progenner.**query_progens**(*obj*, *snaps*, *progenType*)

Function to check for the exitstance of PROGEN files on disk.

Parameters **obj** : SPHGR object

snaps : numpy_array

array of snapshots going from SN2–>SN1+1

See also:

progenInit

Examples

```
>>> for i in range(0,len(objs)):
>>>     query_progens(objs[i],snaps)
```

1.3 Pickle Contents

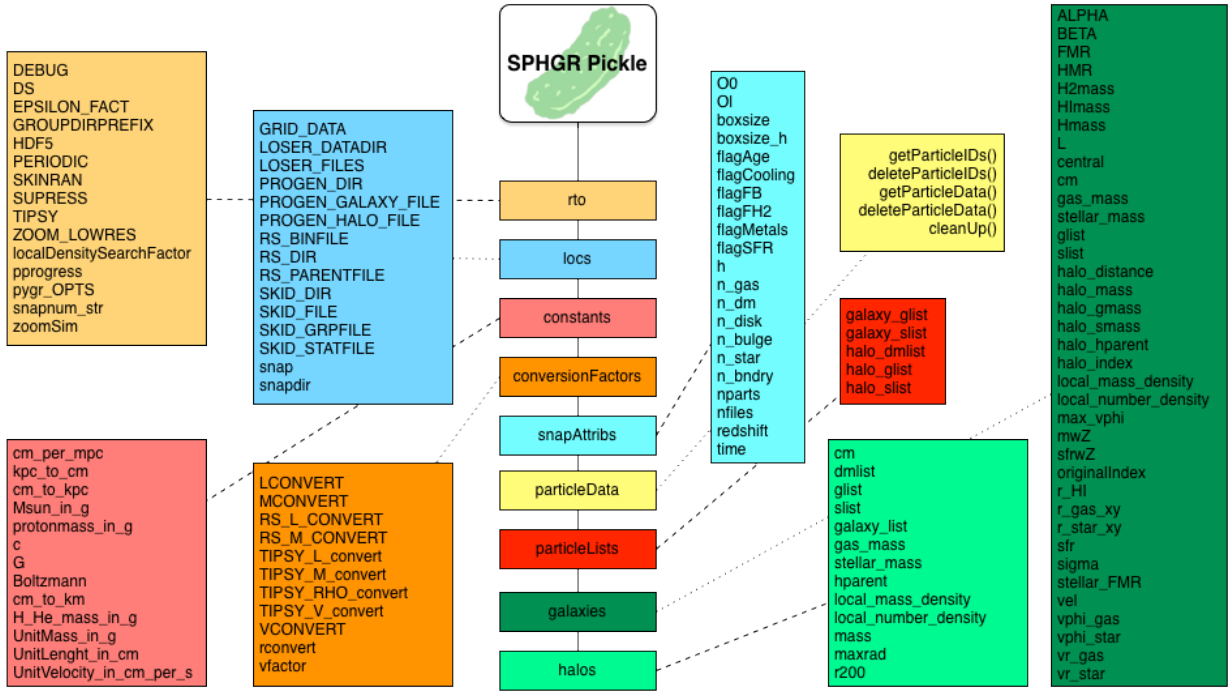
SPHGR stores data in a pickle file. This file is a serialization of objects similar to Java's JSON. This lets us store data in a way where upon reading it back in, all of the objects and functions associated with said objects are restored. The contents of the SPHGR pickle file contains a number of *containers*, which hold other objects; basically a way of organizing the data. The general idea is to have a hierarchy of data that looks something like this:

```
PICKLE
- container1
  - data
- container2
  - data
...
```

Which can then be accessed very easily. To get data within the first container you would refer to it like so:

```
> PICKLE.container1.data
```

These objects are very similar to a common `struct`, but are more flexible in that they can also contain `functions` or `methods`. The hierarchy of an `SPHGR` pickle file look something like this:



1.3.1 Global Attributes & Containers

note: – – denotes a container

`pickleFile.:`

```
- - rto                 # run-time-options
- - constants           # universal constants
- - conversionFactors   # conversion factors
- - locs                # file locations
- - snapAttribs         # snapshot info
- - particleLists       # holds methods to read in particle lists
- - particleData        # holds methods to read in particle data
- - halos               # list of all halo objects
- -  parent_halos       # list of all parent halo objects
- -  sub_halos          # list of all sub-halo objects
- - galaxies            # list of all galaxy objects
- -  central_galaxies   # list of all central galaxy objects
- -  satellite_galaxies # list of all satellite galaxy objects
```

```
- halo_index_translator #
- n_side                # number of particles per side (N^1/3)
- nparts                # list of particle numbers (types 0-5)
- h                     # Hubble parameter
- boxsize               # boxsize
- boxsize_h             # boxsize/h
- redshift              # redshift
- time                  # scale factor
- num_halos             # number of halos
- num_galaxies          # number of galaxies
- snapdir               # snapshot directory
- snapname              # snapshot base name (excluding directory)
- snapnum               # snapshot number
- snap                  # full snap path
- pickle                # path and name of pickle file
- pprogress             # run time parameter
```

rto

pickleFile.rto.:

```
- DEBUG
- DS
- EPSILON_FACT
- GROUPDIRPREFIX
- HDF5
- TIPSY
- PERIODIC
- SKIPRAN
- SUPRESS
- zoomSim
- ZOOM_LOWRES
- localDensitySearchFactor
- pprogress
- pygr_OPTS
- snapnum_str
```

locs

pickleFile.locs.:

```
- GRID_DATA
- LOSER_DATADIR
- LOSER_FILES
- PROGEN_DIR
- PROGEN_GALAXY_FILE
- PROGEN_HALO_FILE
- RS_BINFILE
- RS_DIR
- RS_PARENTFILE
- SKID_DIR
- SKID_FILE
- SKID_GRPFILE
- SKID_STATFILE
- snap
- snapdir
```

constants

pickleFile.constants.:

```
- cm_per_mpc
- kpc_to_cm
- cm_to_kpc
- Msun_in_g
- protonmass_in_g
- c
- G
- Boltzmann
- cm_to_km
- H_He_mass_in_g
- UnitMass_in_g
- UnitLength_in_cm
- UnitVelocity_in_cm_per_s
```

conversionFactors

pickleFile.conversionFactors.:

```
- LCONVERT              # --> kpc
- MCONVERT              # --> Msun
- rconvert              # length conversion factor [1/(1+z)]
- vfactor               # velocity conversion factor [sqrt(a)]
- VCONVERT              # --> km/s
- RS_L_CONVERT          # RS length conversion factor --> Mpc/h
- RS_M_CONVERT          # RS mass conversion factor --> Msun/h
- TIPSY_L_convert       # TIPSY length conversion factor
- TIPSY_V_convert       # TIPSY velocity conversion factor
- TIPSY_M_convert       # TIPSY mass conversion factor
- TIPSY_RHO_convert     # TIPSY density conversion factor
```

snapAttribs

pickleFile.snapAttribs.:

```
- DM_ONLY               # set to 1 if this is a DM only sim
- redshift              # redshift
- time                  # scale factor
- h                     # hubble parameter
- O0                    # matter density
- Ol                    # vacuum energy density
- boxsize               # size of the box (no little h)
- boxsize_h             # size of the box (little h divided out)
- n_gas                 # number of type [0] gas particles
- n_star                # number of type [4] star particles
- n_dm                  # number of type [1] dm particles
- n_bulge               # number of type [3] bulge particles
- n_disk                # number of type [2] disk particles
- n_bndry               # number of type [5] bndry particles
- nparts                # list - [n_gas,n_dm,n_bulge,n_disk,n_star,n_bndry]
- nfiles                # number of files for this snapshot
- flagSFR               # SFR flag
- flagFB                # FB flag
```

```
  - flagCooling          # Cooling flag
  - flagAge              # Stellar Age flag
  - flagFH2              # fH2 flag
```

particleData

pickleFile.particleData.:

```
- getParticleIDs()        # gives access to particle IDs
  - gIDs
  - sIDs
  - dmIDs
- getParticleData()       # gives access to particle data
  - gpos
  - gvel
  - gmass
  - grho
  - gTemp
  - gsfr
  - gHSML
  - gnh
  - gne
  - gZ
  - gfH2
  - zarray
  - spos
  - svel
  - smass
  - sZ
  - sage
  - dmmass
  - dmpos
  - dmvel
- deleteParticleIDs()     # removes particle IDs from memory
- deleteParticleData()    # removes particle data from memory
- cleanUp()               # removes PIDs and particle data
```

particleLists

pickleFile.particleLists.:

```
- getParticleLists()      # gives access to particle lists
  - halo_glist            # halo index membership for all gas
  - halo_slist            # halo index membership for all star
  - halo_dmlist           # halo index membership for all DM
  - halo_lrlist           # halo index membership for all lowres
  - galaxy_glist          # galaxy index membership for all gas
  - galaxy_slist          # galaxy index membership for all star
- deleteParticleLists()   # removes particle lists from memory
```

galaxies (`galaxy` class)

pickleFile.galaxies[N].:

```
- index                 # current index
- cm                    # x,y,z center-of-mass
- vel                   # x,y,z center-of-mass velocities
- HMR                   # half-mass-radius  (SKID)
- FMR                   # full-mass-radius  (SKID)
- stellar_FMR           # maximum stellar radius
- gas_mass              # total gas mass
- stellar_mass          # total stellar mass
- total_mass            # gas + stellar
- HImass                # HI mass
- H2mass                # H2 mass
- Hmass                 # H  mass
- HMR_cv                # circular velocity at HMR (SKID)
- FMR_cv                # circular velocity at FMR (SKID)
- max_cv                # maximum circular velocity
- max_cv_r              # radius at maximum vphi
- sigma1d               # 1D velocity dispersion
- central              # -1=unassigned, 0=satellite, 1=central
- originalIndex         # original SKID index
- sfr                   # total SFR
- Z                     # SFR-weighted metallicity
- mwZ                   # MASS-weighted metallicity
- [glist]               # indexes of gas particles that belong to this gal
- [slist]               # indexes of star particles that belong to this gal
- L                     # Lx,Ly,Lz angular momentum vector
- ALPHA                 # first rotation angle to align L with z-axis
- BETA                  # second rotation angle to align L with z-axis
- local_mass_density    # local mass density of galaxies
- local_number_density  # local number density of galaxies
- halo                  # parent :class:`halo` object
- halo_index            # index of corresponding RS halo
- halo_distance         # distance between galaxy CoM and halo CoM
- halo_dm_mass          # total DM mass within the halo
- halo_gmass            # total gas mass within the halo
- halo_smass            # total stellar mass within the halo
- halo_total_mass       # halo_gas + halo_star + halo_dm mass
- halo_mass             # total DM mass within the halo
- halo_isparent         # -1=parent, anything else is substructure
```

halos (`halo` class)

pickleFile.halos[N].:

```
- index                 # current index
- cm                    # x,y,z center-of-mass
- r200                  # radius
- maxrad                # maximum radius
- L                     # Lx,Ly,Lz angular momentum vector
- hparent               # -1=parent, anything else is substructure
- particle_count        # number of DM particles
- [galaxy_list]         # list of galaxy indexes contained within this halo
- [sub_halo_list]       # list of subhalo indexes contained
- [glist]               # indexes of gas particles that belong to this halo
- [slist]               # indexes of star particles that belong to this halo
- [dmlist]              # indexes of dm particles that belong to this halo
- [lrlist]              # list of low-res particles (if present)
- mass                  # virial mass given by RS
```

```
    - dm_mass                # total DM mass within the halo
    - gas_mass               # total gas mass within the halo
    - stellar_mass           # total stellar mass within the halo
    - total_mass             # gas + star + dm mass
    - local_mass_density     # local mass density of halos
    - local_number_density   # local number density of halos
    - sfr                    # total SFR (all gas within)
    - Z                      # SFR-weighted metallicity (all gas within)
    - mwZ                    # MASS-weighted metallicity (all gas within)
    - energy                 #
    - spin                   #
    - vmax                   #
    - rvmax                  #
    - vrms                   #
    - originalIndex          #
    - sub_halos              # list of sub-halo objects
    - galaxies               # list of contained galaxy objects
```

CHAPTER 2

Code Reference

2.1 initSnap Module

2.2 snapData Module

2.3 Containers

class particleData.**ParticleData**(*obj*)
Bases: object

Class container for holding particle data.

Methods

cleanUp()	
deleteAttribute(attrib)	
deleteParticleData()	Function to delete particle data from the object for storing
deleteParticleIDs()	
getParticleData([GAS, STAR, DM, DISK, ...])	
getParticleIDs([GAS, STAR, DM])	
getRawParticleData()	

cleanUp()

deleteAttribute(*attrib*)

deleteParticleData()
Function to delete particle data from the object for storing

deleteParticleIDs()

getParticleData(*GAS=1*, *STAR=1*, *DM=1*, *DISK=0*, *BULGE=0*, *BNDRY=0*, *LOWRES=0*, *GRHO_UNITS=1*, *METALARRAY=0*, *CONSERVE=0*, *VEL=1*, *NE=1*)

getParticleIDs(*GAS=0*, *STAR=1*, *DM=0*)

getRawParticleData()

class conversion_factors.**ConversionFactors**(*obj*)
Bases: object

class locations.**FileLocations**(*snapdir*, *snap*)
Bases: object

class particleLists.**ParticleLists**(*obj*)
Bases: object

Methods

deleteAttribute(attrib)
deleteParticleLists()
getParticleLists()
saveParticleLists()

deleteAttribute(*attrib*)

deleteParticleLists()

getParticleLists()

saveParticleLists()

class runTimeOpts.**RunTimeOPTS**
Bases: object

class snapAttribs.**SnapAttribs**(*obj*)
Bases: object

Methods

selfAssign(obj)

selfAssign(*obj*)

2.4 Classes

class halo.**Halo**(*cmx*, *cmy*, *cmz*, *r200*, *mass*, *hparent*, *Lx*, *Ly*, *Lz*)

Methods

deleteAttribute(attrib)

deleteAttribute(*attrib*)

class galaxy.**Color**(*name*, *wavelength_range*, *index*)
Bases: object

Methods

flux2mag(FLUX)	
mag2flux(MAG)	

eflux

flux

flux2mag(*FLUX*)

mag2flux(*MAG*)

class galaxy.**Galaxy**(*cmx*, *cmy*, *cmz*, *cmvx*, *cmvy*, *cmvz*, *gas_mass*, *stellar_mass*, *HMR*, *FMR*)
Bases: object

Methods

calculate_HMR(obj, mode)	
deleteAttribute(attrib)	
getProgenGalaxy([makeCopy])	
rotator(obj[, DM])	

calculate_HMR(*obj*, *mode*)

deleteAttribute(*attrib*)

getProgenGalaxy(*makeCopy=1*)

rotator(*obj*, *DM=0*)

2.5 Processing

#.. image:: images/sphgr_gridsearch.png

2.6 Plugins

Out of the box *SPHGR* has plugins for Rockstar as the halo group finder, and SKID as the baryonic group finder. You are more than welcome to build plugins for different programs, just follow the format you see in the plugins directory.

2.6.1 Rockstar

rockstar_init.**initRSAttribs**(*obj*)

rockstar_reader.**read_rockstar**(*obj*)

2.6.2 SKID

skid_init.**initSKIDAttribs**(*obj*)

skid_reader.**iter_loadtxt**(*filename*, *skiprows=0*)
function to load large single-column text files quickly

skid_reader.**read_skid**(*obj*)

2.6.3 LOSER

loser_init.**initLOSERAttribs**(*obj*, *LOSER_viewdir*)

loser_reader.**read_LOSER_fullsim**(*obj*)

2.7 Misc Modules

class visualizer.**vtk_render**
Bases: object

Class to render particle data with VTK.

Once this class is created you can easily add 'actors' to the scene via the enclosed methods.

Examples

```
>>> import visualizer as viz
>>>
>>> obj  = viz.vtk_render()
>>> obj.point_render(pos[:,0],pos[:,1],pos[:,2],
                     color=[0.2,0.3,0.4],opacity=0.8)
>>> obj.sphere(x,y,z,10,color=[1,0,0],alpha=0.2)
>>> obj.renderthis()
```

Methods

Keypress(obj, event)	function to perform actions on keypress
PRINT()	
arrow(p1, p2[, shaftR, tipR, tipL, balls, ...])	Draw an arrow in the render window.
create_helpers()	creates helper text for keyboard shortcuts
cube(center, size[, color])	Render a cube in the render window.
label(cmx, cmy, cmz, text[, textcolor, ...])	Create label within the VTK object.
line(p1, p2[, color])	Render a line in the render window
makebutton()	
optsWindow()	
placetext(text[, size, fontsize, loc, textcolor])	Place text in one of the corners of the render window.
pointRender(pos[, color, opacity, alpha, ...])	
point_render(x, y, z[, color, opacity, ...])	Adds points to the rendering.
print_camera_settings()	function to print camera setting to screen
processrender(xsize, ysize, offscreen)	function to setup VTK render windows and data
quit()	
renderthis([xsize, ysize, bgcolor, ...])	Renders the scene.
resetSTLcounter()	

Continued on next page

Table 2.7 – continued from previous page

save_frame()	function to save current render
savethis(filename)	
setup_axislook(axis)	
setup_camera(hrotate, vrotate, focalpoint)	
setup_helper()	setup helper text
setup_orientmarker(bgcolor)	
sphere(x, y, z, r[, color, opacity, res])	Add a sphere to the rendering.
switch_help(switch)	function to turn help on or off entirely
toggle_actors()	
toggle_help([manual])	function to toggle helper text on/off
volume_render(grid)	Volume rendering currently bugged and unsupported.

Keypress(*obj*, *event*)
function to perform actions on keypress

PRINT()

arrow(*p1, p2, shaftR=0.01, tipR=0.05, tipL=0.2, balls=0, ballcolor=[1.0, 1.0, 0.0], ballradius=1, color=[1, 1, 1], stl='*')
Draw an arrow in the render window.

Parameters **p1** : numpy_array

base of the arrow location

p2 : numpy_array

tip of the arrow location

create_helpers()
creates helper text for keyboard shortcuts

cube(*center, size, color=[1, 1, 1]*)
Render a cube in the render window.

label(*cmx, cmy, cmz, text, textcolor=[1, 1, 1], textfontsize=12, labelboxcolor=[0, 0, 0], labelbox=1, labelboxopacity=0.8*)
Create label within the VTK object.

Parameters **cmx** : float

x-coordinate of the label

cmy : float

y-coordinate of the label

cmz : float

z-coordinate of the label

text : string

what to put in the label

textcolor : tuple (default=[1,1,1])

color of the text

textfontsize : int (default=12)

size of the text

labelboxcolor : tuple (default=[0,0,0])

color of the box enclosing the text

labelbox : int (default=1)

enclose the text in a box?

labelboxopacity : float (default=0.8)

opacity of the enclosing box

line(*p1, p2, color=[1, 1, 1]*)
Render a line in the render window

makebutton()

optsWindow()

placetext(*text, size=800, fontsize=18, loc=1, textcolor=[1, 1, 1]*)
Place text in one of the corners of the render window.

Parameters **text** : string

text to stick in the corner

loc : int (default=1)

which corner?

textcolor : tuple (default=[1,1,1])

what color should the text be?

pointRender(*pos, color=[1, 1, 1], opacity=1, alpha=1, size=800, singlepoint=0, psize=1*)

point_render(*x, y, z, color=[1, 1, 1], opacity=1, alpha=1, size=800, singlepoint=0, psize=1*)
Adds points to the rendering.

Parameters **x** : numpy_array or float

x-coordinates of particles to render

y : numpy_array or float

y-coordinates of particles to render

z : numpy_array or float

z-coordinates of particles to render

color : numpy_array or tuple (default=[1,1,1])

color of particles. if array is passed then it must be the same size as the x/y/z array with each element corresponding to the color of each particle (ie color[i]=[r,g,b])

opacity : float (default=1)

opacity of said particles

alpha : float (default=1)

interchangable with opacity

singlepoint : int (default=0)

if rendering a single point then this must be set to 1

psize : int (default=1)

size of particles?

print_camera_settings()
function to print camera setting to screen

processrender(*xsize*, *ysize*, *offscreen*)
function to setup VTK render windows and data

quit()

renderthis(*xsize=800, ysize=800, bgcolor=[0, 0, 0], orientmarker=1, focalpoint=[], zoom=0, axis=3, hrotate=0, vrotate=0, savefile=None, filetype='png', get_cam_pos=0, set_cam_pos=[], helptext=1, stl=0*)
Renders the scene.

resetSTLcounter()

save_frame()
function to save current render

savethis(*filename*)

setup_axislook(*axis*)

setup_camera(*hrotate*, *vrotate*, *focalpoint*)

setup_helper()
setup helper text

setup_orientmarker(*bgcolor*)

sphere(*x, y, z, r, color=[1, 1, 1], opacity=1, res=12*)
Add a sphere to the rendering.

Parameters **x** : float

x-coordinate of the sphere center

y : float

y-coordinate of the sphere center

z : float

z-coordinate of the sphere center

r : float

radius of the sphere

color : tuple (default=[1,1,1])

opacity : float (default=1)

res : int (default=12)

resolutuion of the sphere

switch_help(*switch*)
function to turn help on or off entirely

toggle_actors()

toggle_help(*manual=None*)
function to toggle helper text on/off

volume_render(*grid*)
Volume rendering currently bugged and unsupported.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

c

g

h

l

p

r

s

v

c

g

h

l

p

r

s

v

Q

R

S

T

V