mesh_illustris

Release 0.2.dev

Bill Chen

Apr 20, 2023

CONTENTS

`mesh_illustris` is a toolkit for analyzing Illustris (and also IllustrisTNG) data with mesh. The goal of `mesh_illustris` is to **quickly** load a subset (e.g., a box or sphere) of particles/cells with **minimal** amount of memory. *Get started* now!

CHAPTER

ONE

INTRO

Illustris is a **huge** suite of simulations with the moving mesh code, `AREPO`. Illustris applies the FoF and `Subfind` algorithms to identify halos/subhalos and store particles accordingly. Therefore, we can easily load a subset of particles belonging to the same halo/subhalo. This method works for most cases but still fails when

1. two halos largely overlap,
2. there are too many "fuzz" particles which don't belong to any halo,
3. we want to trace the evolution of a particle, but it "jumps" to another halo at some time, etc.

Therefore, we develope the `mesh_illustris` toolkit to enable loading Illustris like mesh-based simulations. `mesh_illustris` splits the entire volume into a 3D mesh and index each particle/cell according to its location in the mesh. This method allows us to **quickly** load a subset (e.g., a box or sphere) of particles/cells with **minimal** amount of memory.

CHAPTER

TWO

GET STARTED

2.1 Install

The prerequisites of `mesh_illustris` are

```
python >= 3.8
numpy >= 1.18
h5py >= 2.10
numba >= 0.50
```

Lower versions may also work (and higher versions may not work). Please raise an issue if it doesn't work for you. Next, the `mesh_illustris` package can be easily installed with `pip`:

```
$ pip install mesh_illustris
```

Alternatively, you can `git clone` the source package from GitHub:

```
$ git clone https://github.com/EnthalpyBill/mesh_illustris.git
```

To build and install `mesh_illustris`, `cd` the folder and `pip install` it:

```
$ cd mesh_illustris/
$ pip install -e .
```

The `-e` command allows you to make changes to the code.

2.2 Usage

To use the package, just import it as

```
>>> import mesh_illustris as mi
```

To start with, let's load the last snapshot of the simulation:

```
>>> base = "/your/base/path/output"
>>> partType = ["gas"]
>>> d = mi.load(base, snapNum=99, partType=partType)
```

Now, a `Dataset` object is created. Let's load a 100 kpc/h box in the simulation:

```
>>> boundary = np.array([[0,0,0],[100,100,100]])
>>> fields = ["Coordinates", "Masses"]
>>> data = d.box("c", boundary=boundary, partType=partType, fields=fields)
```

The method `box()` automatically start a pre-indexing process if it has not been done before. Once the pre-indexing is complete, several index files will be created at `base`. It may take time to generate such files, but once generated, `mesh_illustris` will skip pre-indexing for the next time. If you want to save the index file to a different location, just specify the path to `load()` with the argument `index_path`.

CHAPTER

THREE

COOKBOOK

In this page, we show some example scripts written with `mesh_illustris`. Feel free to contribute to this cookbook.

3.1 Load a box from TNG50-4-Subbox0

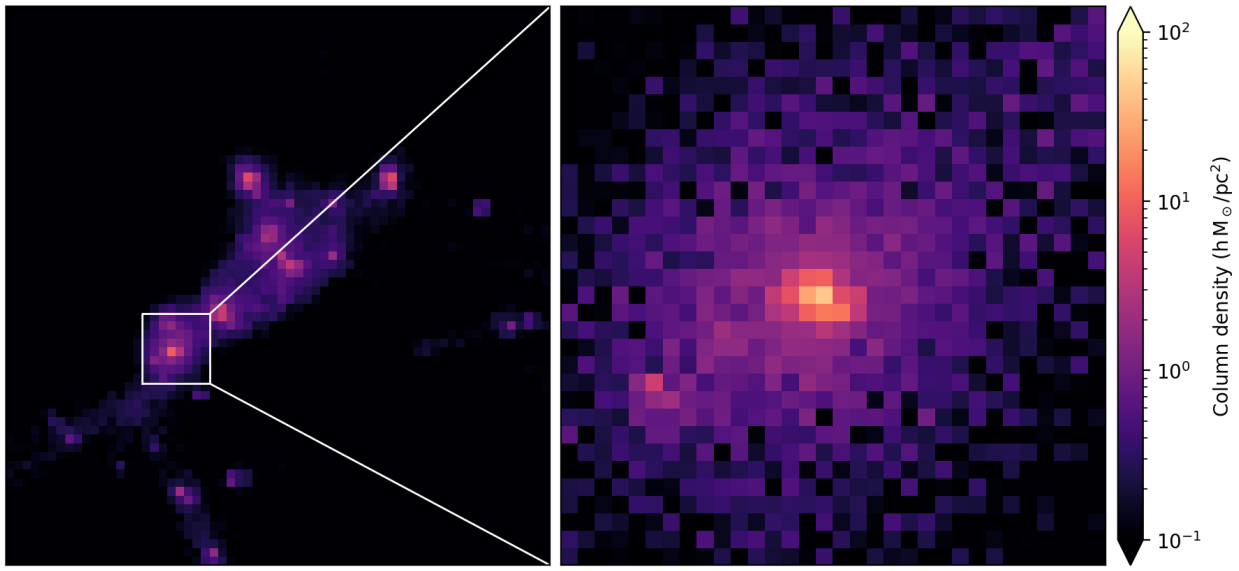
TNG50-4-Subbox0 is a subbox of the TNG50 simulation. The center of this subbox is (26, 10, 26.5) Mpc/h, and the side length is 4 Mpc/h. We first load the entire subbox with the `box()` method of the `Dataset` object. Next, we laod a smaller 500 kpc/h box containing a Milky-Way mass galaxy also with the `box()` method. Finally, we make projection plots of gas for both boxes.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Copyright (c) 2021 Bill Chen
# License: MIT (see LICENSE)

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import copy

import mesh_illustris as mi

my_magma = copy.copy(mpl.cm.get_cmap('magma'))
my_magma.set_bad(my_magma(-1))

basePath = "TNG50-4-Subbox0/output" # may alter
d = mi.load(basePath, 2332, ["gas"])

# The center of TNG50-4-Subbox0 is (26, 10, 26.5) Mpc/h
# The side length of TNG50-4-Subbox0 is 4 Mpc/h
boundary = np.array([[24.0, 8.0, 24.5], [28.0, 12.0, 28.5]]) # in Mpc/h
# Note, the internal length unit of TNG is kpc/h
box = d.box(boundary*1000, ["gas"], ["Coordinates", "Masses"])

fig, [ax0, ax1] = plt.subplots(1, 2, figsize=(9,4))
fig.subplots_adjust(wspace=0.02)

x = box["gas"]["Coordinates"][:,0] / 1000 # in Mpc/h
y = box["gas"]["Coordinates"][:,1] / 1000 # in Mpc/h
weights = box["gas"]["Masses"] / (4/64)**2 / 1e2 # in h Msun/pc^2
ax0.hist2d(x, y, norm=mpl.colors.LogNorm(), weights=weights,
        range=[[24.0, 28.0], [8.0, 12.0]], bins=64, cmap=my_magma,
```

(continues on next page)

(continued from previous page)

```
        vmax=1e2, vmin=1e-1)

ax0.plot([25, 25.5, 25.5, 25,25], [9.3, 9.3, 9.8, 9.8, 9.3], c="w", lw=1)
ax0.plot([25.5, 28.0], [9.8, 12.0], c="w", lw=1)
ax0.plot([25.5, 28.0], [9.3, 8.0], c="w", lw=1)

ax0.set_xticks([])
ax0.set_yticks([])

boundary = np.array([[25.0, 9.3, 26.25], [25.5, 9.8, 26.75]]) # in Mpc/h
box = d.box(boundary*1000, ["gas"], ["Coordinates", "Masses"])

x = box["gas"]["Coordinates"][:,0] / 1000 # in Mpc/h
y = box["gas"]["Coordinates"][:,1] / 1000 # in Mpc/h
weights = box["gas"]["Masses"] / (0.5/32)**2 / 1e2 # in h Msun/pc^2
h = ax1.hist2d(x, y, norm=mpl.colors.LogNorm(), weights=weights,
        range=[[25.0, 25.5], [9.3, 9.8]], bins=32, cmap=my_magma,
        vmax=1e2, vmin=1e-1)

ax1.set_xticks([])
ax1.set_yticks([])

plt.subplots_adjust(bottom=0, right=1, top=1)
cax = plt.axes([1.01, 0, 0.02, 1])
cbar = plt.colorbar(h[3], cax=cax, extend="both")
cbar.set_label(r"Column density $({\rm h\,M_\odot/pc^2})$")

plt.savefig('figures/load_box.png', bbox_inches ='tight',
        pad_inches=0.05, dpi=200)
```



CHAPTER

FOUR

CONTRIBUTE

4.1 How to contribute

`mesh_illustris` is distributed in GitHub. Feel free to dive in and raise an issue. If you want to modify the codes, please *run tests* (as well as write new tests) before submitting a pull request.

4.2 Maintainers of `mesh_illustris`

Author:

- @ybillchen (Bill Chen)

Maintainers:

- @ybillchen (Bill Chen)

4.3 License

`mesh_illustris` is available at GitHub under the MIT license.

CHAPTER

FIVE

TEST GUIDE

Performing tests is extremely important in producing robust codes. The test framework of `mesh_illustris` is `pytest`. Since we don't need `pytest` for common use, it's not included in `install_requires` of `setup.py`. So, before moving on, please manually install `pytest` if it's not installed:

```
$ pip install pytest
```

There are two ways to run tests. First, we can simply run

```
$ pytest
```

under the source code path of `mesh_illustris` (with `__init__.py` inside). Run `pytest -v` instead if you want more information. See `pytest`'s documentation for more details.

The second way doesn't require `cd` the source code path. After installing `mesh_illustris`, we can run tests by

```
>>> import mesh_illustris as mi
>>> mi.test()
```

Similarly, replace `mi.test` with `mi.test(["-v"])` if you want more information.

If you get everything passed, congratulations! Skipped cases are also OK. However, if you see errors, we'd strongly suggest you to slow down and check what's wrong with the code. Raise an issue if it can't be solved!

CHAPTER

SIX

MODULE API

6.1 mesh_illustris.core module

core codes of the mesh_illustris package.

class mesh_illustris.core.**Dataset**(*datasets*, *n_chunk*)
Bases: object

Dataset class stores a snapshot of simulation.

property datasets
Chunks that store the snapshot of simulation.

Type list of SingleDataset

property n_chunk
Number of chunks.

Type int

box(*boundary*, *partType*, *fields*, *mdi=None*, *float32=False*)
Load a sub-box of data.

Parameters

- **boundary** (*numpy.ndarray of scalar*) – Boundary of the box, with shape of (3, 2).
- **partType** (*str or list of str*) – Particle types to be loaded.
- **fields** (*str or list of str*) – Particle fields to be loaded.
- **mdi** (*None or list of int, default to None*) – sub-indeces to be loaded. None to load all.
- **float32** (*bool, default to False*) – Whether to use float32 or not.

Returns Sub-box of data.

Return type dict

sphere(*center*, *radius*, *partType*, *fields*, *mdi=None*, *float32=False*)
Load a sub-sphere of data.

Parameters

- **center** (*numpy.ndarray of scalar*) – Center of the sphere, with shape of (3,).
- **radius** (*scalar*) – Radius of the sphere.
- **partType** (*str or list of str*) – Particle types to be loaded.

- **fields** (*str or list of str*) – Particle fields to be loaded.
- **mdi** (*None or list of int, default to None*) – sub-indeces to be loaded. None to load all.
- **float32** (*bool, default to False*) – Whether to use float32 or not.

Returns Sub-sphere of data.

Return type dict

class mesh_illustris.core.**SingleDataset**(*fn*, *partType*, *depth=8*, *index_path=None*)

Bases: object

SingleDataset class stores a chunck of snapshot.

property fn

File name to be loaded.

Type str

property partType

Particle types to be loaded.

Type str or list of str

property box_size

Box size of the simulation.

Type scalar

property index

Newly generated or cached index of the Dataset.

Type dict

box(*boundary*, *partType*, *fields*, *mdi=None*, *float32=True*, *method='outer'*)

Slicing method to load a sub-box of data.

Note: The current version only support loading the outer or inner box of the sub-box. Loading the exact sub-box is not supported.

Parameters

- **boundary** (*numpy.ndarray of scalar*) – Boundary of the box, with shape of (3, 2).
- **partType** (*str or list of str*) – Particle types to be loaded.
- **fields** (*str or list of str*) – Particle fields to be loaded.
- **mdi** (*None or list of int, default to None*) – sub-indeces to be loaded. None to load all.
- **float32** (*bool, default to False*) – Whether to use float32 or not.
- **method** (*str, default to "outer"*) – How to load the box, must be "outer" or "exact" or "inner".

Returns Sub-box of data.

Return type dict

sphere(*center*, *radius*, *partType*, *fields*, *mdi=None*, *method='outer'*)

Slicing method to load a sub-sphere of data.

Note: This function is not supported in the current version

Parameters

- **center** (*numpy.ndarray of scalar*) – Center of the sphere, with shape of (3,).
- **radius** (*scalar*) – Radius of the sphere.
- **partType** (*str or list of str*) – Particle types to be loaded.
- **fields** (*str or list of str*) – Particle fields to be loaded.
- **mdi** (*None or list of int, default to None*) – sub-indeces to be loaded. None to load all.
- **float32** (*bool, default to False*) – Whether to use float32 or not.
- **method** (*str, default to "outer"*) – How to load the box, must be "outer" or "exact" or "inner".

6.2 mesh_illustris.il_util module

il_util module defines some commonly used functions for Illustris.

mesh_illustris.il_util.**loadFile**(*fn*, *partType*, *fields=None*, *mdi=None*, *float32=True*, *index=None*)

Load a subset of particles/cells in one chunk file. This function applies numpy.memmap to minimize memory usage.

Parameters

- **fn** (*str*) – File name to be loaded.
- **partType** (*str or list of str*) – Particle types to be loaded.
- **fields** (*str or list of str*) – Particle fields to be loaded.
- **mdi** (*None or list of int, default to None*) – sub-indeces to be loaded. None to load all.
- **float32** (*bool, default to False*) – Whether to use float32 or not.
- **index** (*list of list of int*) – List of Fancy indices for slicing.

Returns Entire or subset of data, depending on whether index == None.

Return type dict

mesh_illustris.il_util.**partTypeNum**(*partType*)

Map common names to numeric particle types.

Parameters **partType** (*str*) – Common names of particle type.

Returns Numeric particle type.

Return type int

mesh_illustris.il_util.**snapPath**(*basePath*, *snapNum*, *chunkNum=0*)

Return path to a chunk file of snapshot.

Parameters

- **basePath** (*str*) – Base path of the simulation data. This path usually ends with "output".
- **snapNum** (*int*) – Number of the snapshot.

- **chunkNum** (*int, default to 0*) – Number of the chunk.

Returns Path to a chunk file.

Return type str

6.3 mesh_illustris.loader module

loader module defines a convenient load() function to load snapshots in Illustris or IllustrisTNG.

mesh_illustris.loader.**load**(*basePath*, *snapNum*, *partType*, *depth=8*, *index_path=None*)

Function to load snapshots in Illustris or IllustrisTNG.

Parameters

- **basePath** (*str*) – Base path of the simulation data. This path usually ends with "output".
- **snapNum** (*int*) – Number of the snapshot.
- **partType** (*str or list of str*) – Particle types to be loaded.
- **depth** (*int, default to 8*) – Depth of mesh. For example, depth = 8 corresponds to the mesh dimension of (2^8, 2^8, 2^8).
- **index_path** (*str*) – Path to store the index files. None to store with the data.

Returns Structured data.

Return type *Dataset*

6.4 mesh_illustris.mesh module

mesh module defines the Mesh to tessellate the entire volume.

class mesh_illustris.mesh.**Mesh**(*pos*, *length*, *offset*, *boundary*, *depth*)

Bases: object

Mesh class tessellates the entire volume.

property boundary

Boundary of the box, with shape of (3, 2).

Type numpy.ndarray of scalar

property depth

Depth of Mesh. For example, depth = 8 corresponds to the Mesh dimension of (2^8, 2^8, 2^8)

Type int, default to 8

build()

Build index for the points according to the Mesh. The indexing process produces a "rand" and a "mark" variables, which link the index of each point to its location in the Mesh.

Returns (rank, mark).

Return type tuple of numpy.ndarray of int

PYTHON MODULE INDEX

m

INDEX