3D Visualization of Early Cosmic Structure Formation

Andrea Husseiniova

Code Documentation

Institute for Astronomy
University of Edinburgh
United Kingdom
July 2019

Contents

1 Code Description

The code provided produces 3D animations of dark matter density distributions. It is written in such a way that it can take any dataset and produce a 3D animation of any requested field.

The code is run by running the command `python 3DVisualization.py`. The required packages for this program are `numpy`, `matplotlib`, `scipy` and `yt`.

To customize the program and its inputs, one needs to `Define parameters in CODE`. These are all programmed into the main function at the top of the script. All parameter descriptions and their recommended values are shown in section 2.

The end product from the script is an `.mp4` file containing the animation defined by the parameters. For animations including rotations, the animation also shows its rotational velocity in multiples of the speed of light. The velocity calculation is done assuming the box size is in Mpc, however this can be changed in the `calc_velocity` method by changing the `to_meters` variable.

It can be the case, as described in section 3, that the program must be run on Cuillin due to computational demand. So far, creating an `.mp4` file on Cuillin is not possible, therefore the user is also provided with an `Animation.py` script, which is run after generating all the frames on Cuillin. To run this script, it is necessary to provide the file path to the images created by changing the `image_path` variable, the number of `frames` created and the name of the animation (`anim_name`). The command to run this script is simply `python Animation.py`. To generate frames only on Cuillin, without the actual animation, please comment out the `make_anim` method in the main function.

2 Parameters

Descriptions of all parameters is given in the code itself and for clarity summarized in Table 1. The parameters are organized alphabetically. The user is not required to specify all the parameters available, most of them are provided with a default value, which was deemed appropriate and tested on various simulations.

Although the program poses no requirements on the user in terms of parameter specification, in Table 2 is a list of recommended values, which have proven to produce effective and smooth animations.

parameter	description	required format
`angle`	angle to rotate camera by over the course of the whole animation (multiple of pi)	integer
`anim_center_req`	which point to center animation at, options: max (string), box_middle (string), arbitrary (array), halo id (int)	string / array / int
`anim_name`	name of animation file produced at the end (without extension)	string in quotes
`anim_type`	animation type: zoom, rotate, move or rotzoom (rotate and zoom)	string in quotes
`data_path`	pathway to where data is saved	string in quotes
`dims`	resolution of each frame (in pixels)	integer
`fps`	frames per second in animation	integer
`frames`	how many frames to generate for animation	integer
`magnif`	total magnification to achieve by end of animation wrt current state	float
`program_path`	pathway to where program is saved	string in quotes
`rockstar_path`	pathway to rockstar catalog	string in quotes
`s_clip`	sigma_clip: how much to enhance regions of high density, higher number = less enhancement	float

Table 1: Table of all parameters used in the program, their description and format in alphabetical order.

parameter	recommended	range
`frames`	100	50+
`magnif`	25.0	10.0 - 50.0
`dims`	128	64 - 512
`fps`	10	5+
`s_clip`	5.0	4.0 - 9.0

Table 2: Table of recommended parameter values and suitable ranges of values for each parameter.

3 Computational Demand

This code can be quite computationally demanding depending on the parameters set, therefore a table of expected demands and corresponding parameter values was devised. This information is based on producing an animation from a dataset from a particle based simulation using a 100 Mpc box with 256^3 particles in it. Table 3 shows the RAM demand for a given value of `dims` which corresponds to the number of pixels along each axis. The table also states whether the program with such value is recommended to be run at the local Linux machines.

Another feature to consider when determining the computational demand, is the type of animation requested (`zoom, rotate, move`). The `zoom` animation has proven to be much slower and

dims	RAM requirement (GB)	run on local machines?
32	<<15	yes
64	<15	yes
128	</∼15	can fail
256	16	no
512	42	no
1024	74	no

Table 3: Table of dimension (pixels) and corresponding RAM requirements. Third column shows whether the program with such parameter will run on the local Linux machines with 15GB of RAM.

more computationally demanding than the other two, therefore running these animations on Cuillin is recommended.

4 Using Rockstar

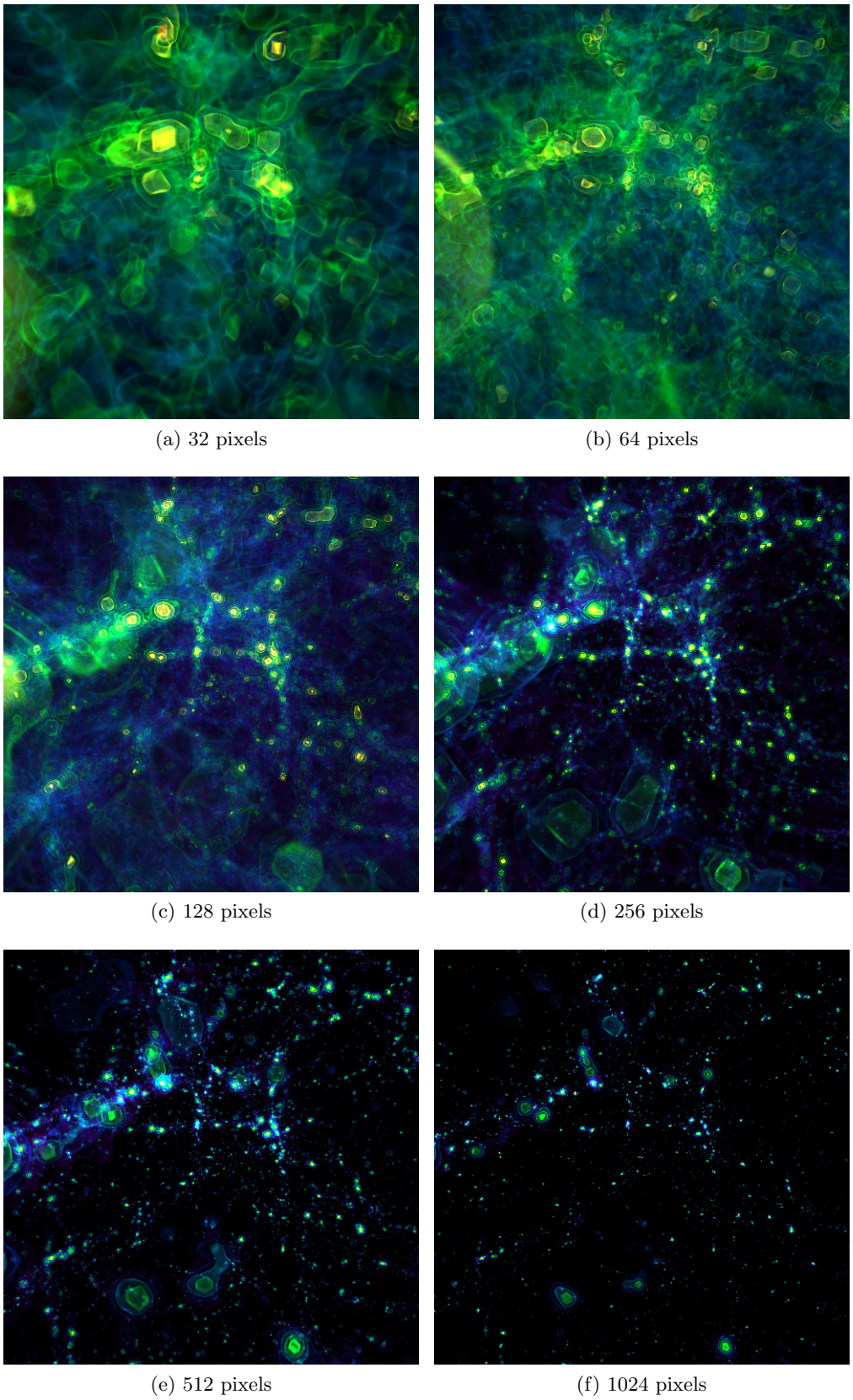
The `3DVisualization.py` program also enables the user to center an animation on any halo in the Rockstar catalog and zoom in on it, rotate around it or move the camera to it. This is done by providing a pathway to the Rockstar catalog and the `.bin` file corresponding to the animated snapshot. The program then takes in the `halo_id` of any given halo as the `anim_center_req`. The user can choose any of the 4 types of animations available and it will be centered on the halo requested. For halos it is recommended to try out the `rotzoom` animation which rotates around the halo while zooming in on it.

5 Sample Frames

This section shows some sample plots, which may be useful in determining the `dims` and the `sigma_clip`. These plots were made using a dataset from a particle based simulation with 256^3 particles in a 100 Mpc box.

Firstly we look at the resolution of the images. Figure 1 shows the same image at multiple resolutions corresponding to different values of the `dims` parameter. All of these plots were produced using `s_clip` = 5.0. It is apparent that higher resolution frames might require a lower `s_clip` to enhance the higher density regions. All plots are displayed from the same angle apart from the 1024 pixel frame. This is because the last frame was generated as a part of a rotation animation and its rendering cannot be redone without accessing the computational cluster Cuillin.

To demonstrate how the value of `s_clip` influences the 3D plots, Figure 2 shows the 128 pixel plot in values ranging from 2.0 to 8.0 including no `s_clip` specified.

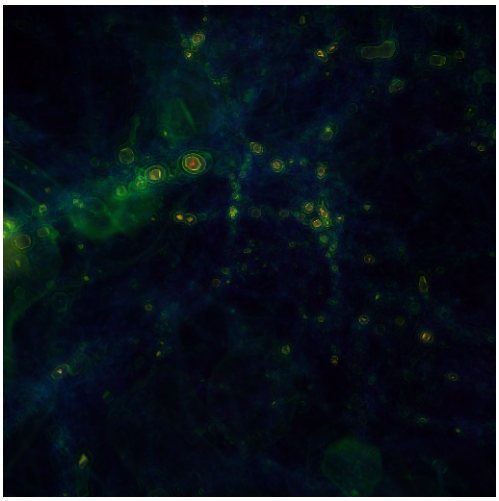


(a) 32 pixels (b) 64 pixels

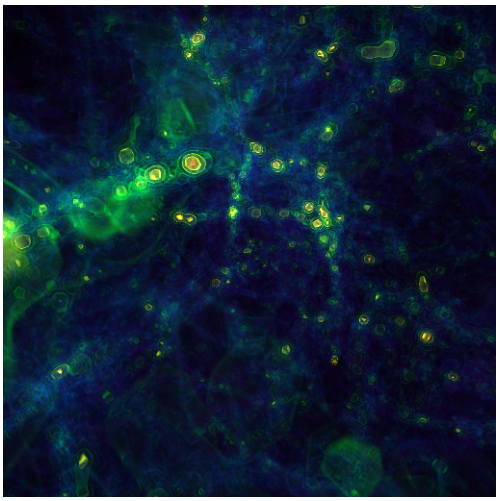
(c) 128 pixels (d) 256 pixels

(e) 512 pixels (f) 1024 pixels

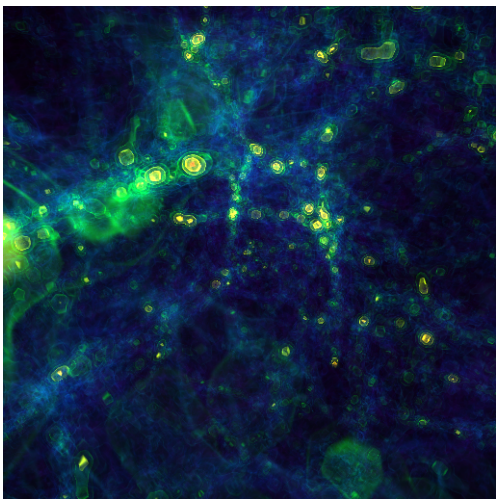
Figure 1: Set of 3D plots corresponding to `dims` = 32, 64, 128, 256, 512 and 1024 respectively. The value of `dims` corresponds to the number of pixels along each axis.



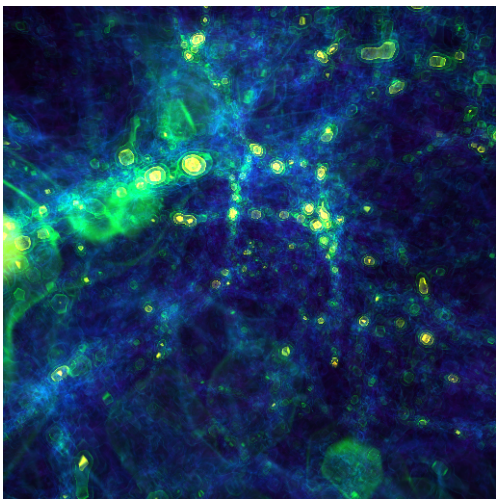
(a) sigma clip = None



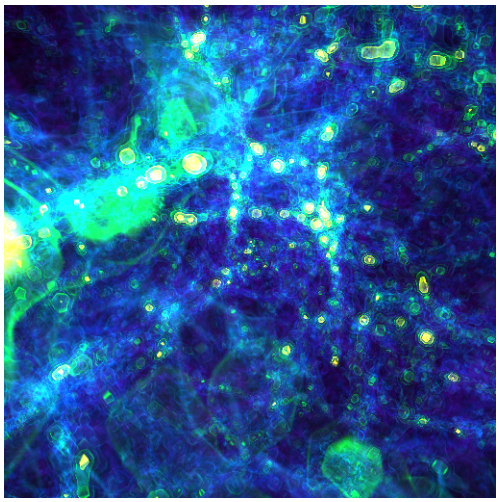
(b) sigma clip = 8.0



(c) sigma clip = 6.0



(d) sigma clip = 4.0



(e) sigma clip = 2.0

Figure 2: Set of 3D plots (`dims` = 128) corresponding to `s_clip` = None, 8.0, 6.0, 4.0 and 2.0 respectively. The value of `s_clip` changes how enhanced high density regions are. Higher `s_clip` means less enhancement.