School of Physics and Astronomy

Senior Honours Project

Visualising the Cosmic Web and Dark Matter Halos

Faidra Antoniadou
April 2022

Abstract

Cosmological N-body dark matter simulations have facilitated da capo analyses of the formation of structure in the Universe in the past few decades. Small density fluctuations produced just after the Big Bang proliferated due to gravity, resulting in the formation of galaxies, enclosed within dark matter halos, in the cosmic web. The need for tools to investigate time-dependent particle-based simulations has grown as a result of these calculations.

This paper presents a framework that integrates three ways to visualise and make movies of dark matter halos, utilising the `yt` open source software. These include following halos during their evolution over time, navigating between neighbouring halos and detecting and visualising flybys. The new methods we present are well-adapted for visualising the cosmic web and can probe a deeper understanding of the cosmic web's complex nature. The framework in itself accepts multiple parameters as user input and is therefore user-friendly.

Declaration

I declare that this report is my own work.

Supervisors: Dr. B. D. Smith and Prof. S. Khochfar 10 Weeks

Contents

1 Introduction

Just after inflation, the Universe's dark and baryonic matter density functions resembled realisations of nearly constant, isotropic and stationary Gaussian random fields [1]. These fields evolved into the greatly nonuniform, matter and galaxy distribution that exists today solely due to the influence of gravity [2, 3, 4].

The cosmic web is one of the most striking features of the distribution of galaxies and dark matter on the largest scales in the Universe. The late-time Universe consists of two components: voids, and the cosmic web, which can be further partitioned into sheets, (galaxy) clusters, rich and poor, and filaments [5]. The web-like structure is mostly driven by dark matter and within the filaments we find dark matter within which galaxies are enclosed. Voids are largely devoid of galaxies and encompass the volume in between [6, 3, 7, 8]. The study of the cosmic web has focused primarily on the identification of such features, and on understanding the environmental effects on galaxy formation and halo assembly. As such, a variety of different methods have been devised to classify the cosmic web - depending on the data at hand, be it numerical simulations, large sky surveys or otherwise [8].

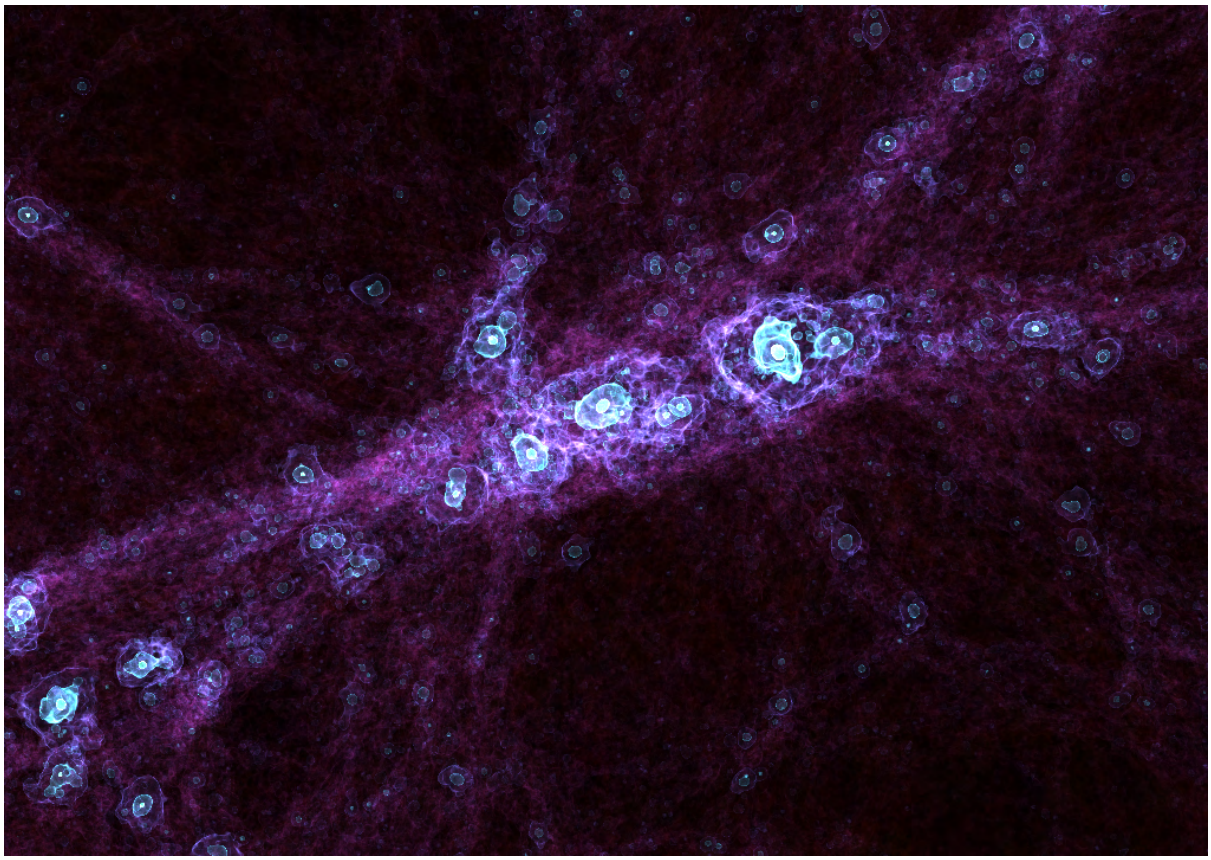

Figure 1: Snapshot of an image rendered using `yt`. The frame spans 50 Mpc and is centred on a halo with mass 2.662×10^{14} $M_{\odot}$.

In addition, the large scale structure we observe in the distribution of galaxies is thought to be a consequence of the gravitational instability of a medium that was relatively smooth

in the early Universe, with the nonlinear patterns we see having evolved from small density fluctuations superposed on a homogeneous and isotropic background [3]. Overdense regions subsequently collapsed into dark matter halos which then begin accreting mass and undergoing mergers, as described by the hierarchical model [9].

Every galaxy forms within a dark matter halo, according to our current understanding of galaxy formation [10]. Consequently, researching the behaviour and evolution of dark matter can help us better comprehend those of galaxies. This is because the birth and evolution of galaxies are linked to the growth of the halos in which they form [11].

Simulations are widely used in cosmology for a number of tasks: i) the interpretation of observations in terms of the underlying physics and cosmological parameters; ii) the testing and aiding the development of perturbative approaches and analytic models for structure formation; iii) the production of reliable input (training) data for data-driven approaches and emulators; iv) the creation of mock universes for current and future large-scale extragalactic cosmological surveys, from which we can quantify statistical and systematic errors; v) the study of the importance of various aspects of the cosmological model and physical processes, and determining their observability [12]. There are various numerical methods – grid or particle-based – to perform such simulations [13].

Technological advancements in high performance computing has allowed cosmologists to model the formation of the cosmos, with the use of visual analysis to explore and analyse complex, large scale datasets, being widely used in research. Previous works have visualised, for example, the Millennium Simulation Project's (see Figure 2) 10 billion dark matter particles model of the expansion of the universe.

Structure formation information in modern cosmological simulations spans many orders of magnitude in physical scale. One current difficulty is to visualise this in a straightforward and easy-to-use format for analysis.

In this project, we work with multiple terrabyte data from a state-of-the-art cosmological dark matter simulation. The aim is to visualise the evolution and formation of the cosmic web. We focus on visualising the structure formation of dark matter halos from our simulation, through movies (see Figure 24 for instance). At the same time, we highlight the physical properties of individually formed objects in the simulation. This can then be used to investigate the assembly history of dark matter halos like the one surrounding our own galaxy. Various software packages for visualisation are available, however, the raw simulation data needs to be prepared in a way it can be used by them. In this study, we employ the use of `yt` (available at `http://yt-project.org/`) an open source, community-developed astrophysical analysis and visualisation toolkit [15] which is mainly written in Python[1] with several core routines written in C for speed enhancement.

We aim to describe an approach to visualising various aspects of cosmic web structure and its evolution through time. This leads us to discuss the formation and patterns of different structures composing the cosmic web. The contributions of this paper are encompassed in the framework we describe which allows for:

- The user to render and visualise individual dark matter halos and particularly ones

[1] `http://www.python.org/`

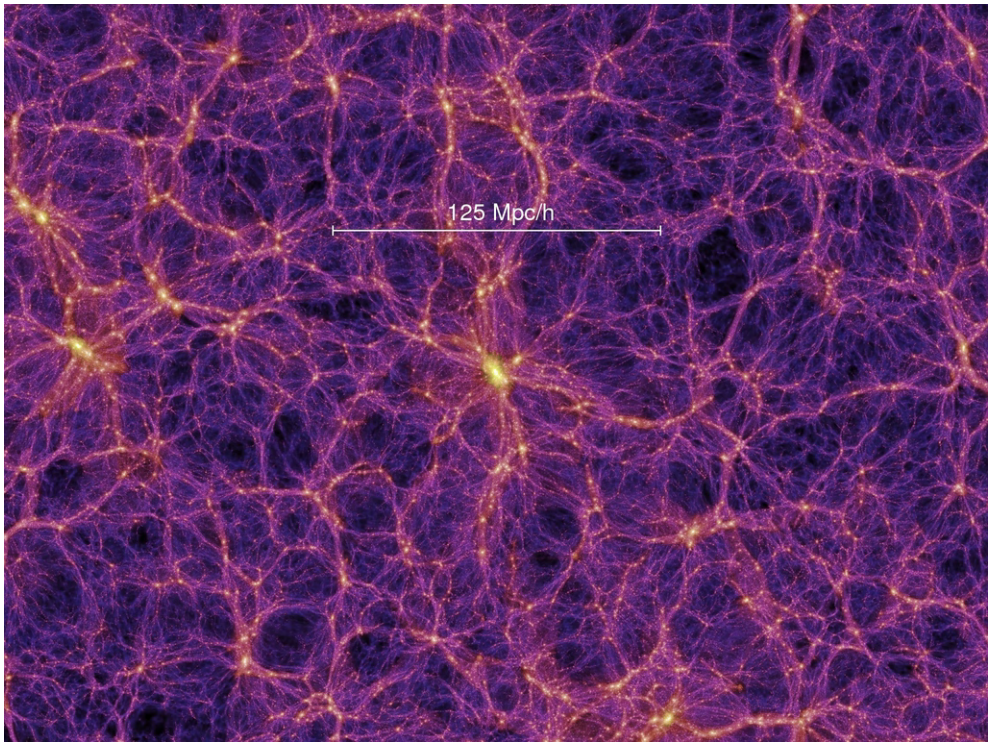

Figure 2: The Millennium Run used more than 10 billion particles to trace the evolution of the matter distribution in a cubic region of the Universe over 2 billion light-years on a side. The figure gives the dark matter distribution in the simulation on Mpc scales.
Source: Adapted from [14]

with chosen distinctive properties. The user may choose to apply transformations such as rotations, zooming and translations. The output is multiple image files that can be compiled into a movie of the transformation.

- A method to produce smoother halo paths is also provided. This is useful in the case where the centre of gravity of a given halo moves between different discrete redshifts shift of the camera focus is abrupt - for instance, during major mergers.
- Tracks halos with chosen distinctive properties and visualises their evolution over time.
- Navigation between individual halos with the functionality to choose the length of the path in units of the primary halo's virial radius. This allows for the visualisation of various structures, such as filaments.
- The detection and visualisation of flyby interactions within halo systems where the user chooses the mass of the most massive (primary) halo in the interaction.

In Section 2 we briefly summarize the N-body simulation we use, as well as the basic visualisation and rendering techniques we employ. The methods for identifying and tracking halos and creating camera are described Section 2.3. Our visualisation results and observations are presented in Section 3. Finally, Section 4 summarises the main conclusions that we draw from our work as well as potential future improvements.

2 N-Body Simulation & Visualisation Methods

2.1 The Simulation and Dataset

In this report, we develop visualisations from covering grids containing 1024^3 cells. The original simulations are composed of 2048^3 particles which were then deposited onto these grids to produce that number of cells. They contain a 3D grid and deposited mass/ density field of particles into the point in the grid where those particles would be located.

This was done using particle-based numerical simulations ran on GADGET ((**GA**laxies with **D**ark matter and **G**as int**E**rac**T**)) which can be used both for studies of isolated self-gravitating systems including gas, or for cosmological N-body simulations [16]. We use these in combination with merger tree and dark matter halo catalogues, to produce movies.

There are in total 102 grid snapshots, each corresponding to different cosmic time frames. Figure 3 illustrates the structure of the dataset in terms of redshifts and cosmic time in Gyr.

The simulation parameters are: Hubble constant $H_0 = 69.5$ km s^{-1} Mpc^{-1}, and cosmological parameters: mass (baryonic and dark matter) density $\Omega_m = 0.285$, dark energy density $\Omega_\Lambda = 0.715$, and radiation (photons and relativistic neutrinos) density $\Omega_{rad} = 0$.

Every `yt` dataset has a "code" unit system that corresponds to the unit system in which the data is stored on a disc [15]. In our case, each cubic grid has width 100 code units, corresponding to ~ 144 Mpc.

2.2 Visualisation Techniques

2.2.1 Slices & Projections

`yt` provides many tools for visualising data, such as slice plots, wherein a 3-D volume (or any of the data objects) is sliced by a plane to return the 2-D field data intersected by that plane. Additionally, `yt` may generate line queries (i.e. rays) of a single line intersecting a 3-D dataset. Projection plots are generated by projecting a 3-D volume into 2-D either by summing or integrating a field along each pixel's line of sight. Slices, projections, and rays can be made to align with the primary axes of the simulation (e.g. x, y, z) or at any arbitrary angle throughout the volume. Examples of these are shown in Figure 4.

Slices are generally quicker than projections because they only need to read and process a slice through the dataset. These are useful to produce visualisations with annotations which we employ in our flyby visualisation method in Section 3.5.

2.2.2 Volume Rendering

A more advanced visualisation functionality in `yt` includes creating photorealistic isocontour images of your data called volume renderings which we use in subsequent sections, to create movies.

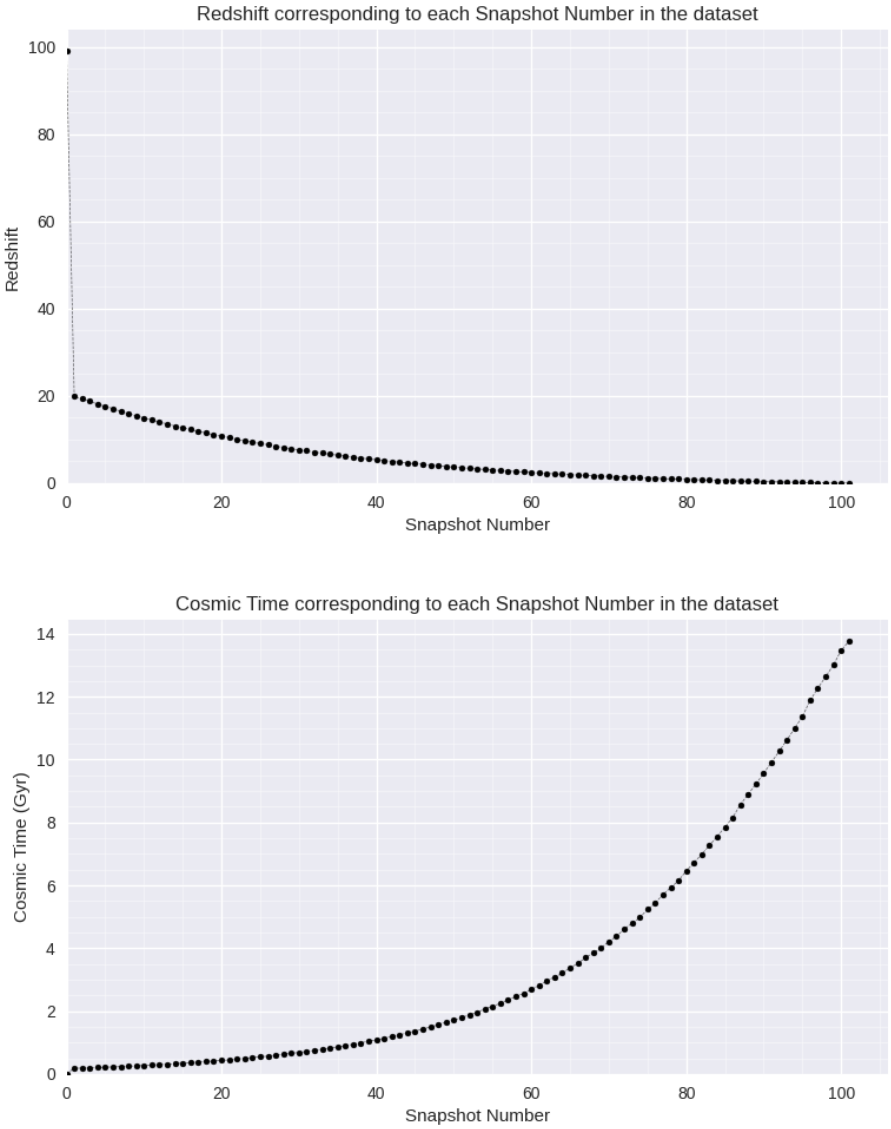

Figure 3: Dataset structure. The graph at the top shows the redshift that corresponds to each snapshot number, while the graph at the bottom shows the respective cosmic time in Gyr for each snapshot number. As shown, the snapshot with the highest number (101) corresponds to the most recent cosmic time frame.

The provided `Camera` interface allows for camera paths, zooming, stereoscopic rendering and easier access to the underlying vector plane where 3-D geometric objects of individual datasets (volume sources) are placed into the scene for rendering. Subsequently, to map the simulation field values to colour, brightness, and opacity in the resulting rendered image, we use a transfer function. This describes how rays that pass through the domain of a volume source. The x-dimension typically represents field values in the underlying dataset to which we want our rendering to be sensitive. The y-dimension consists of 4 channels for red, green, blue, and alpha (opacity). A transfer function starts with all zeros for its y-dimension values, implying that rays traversing the volume source will not show up at all in the final image. However, we can add features to the transfer function

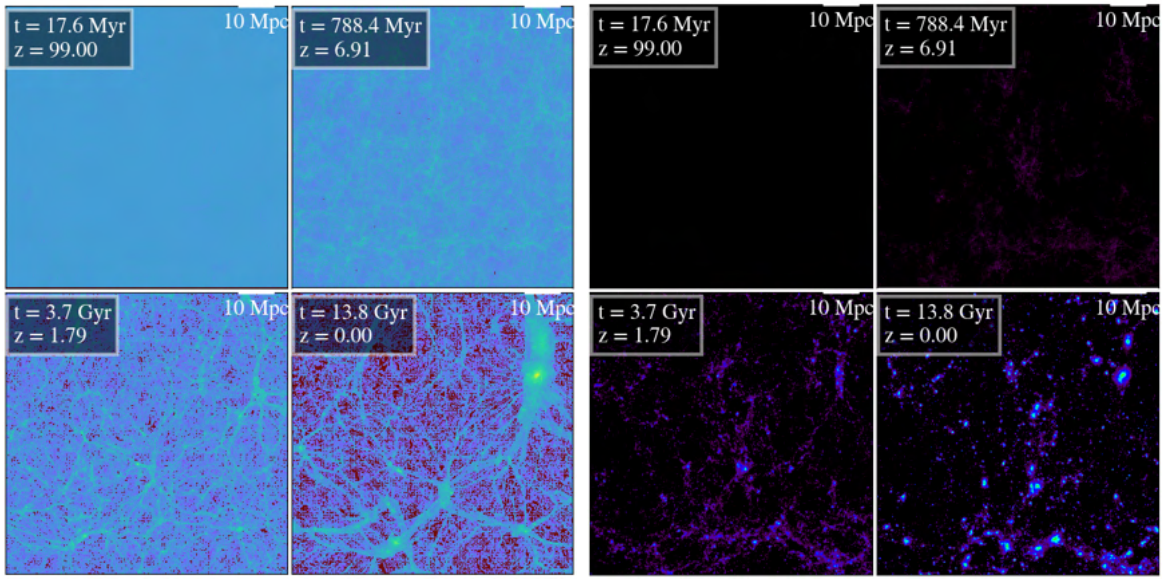

Figure 4: On the left-hand side a multiplot of four slices (in four different snapshots) is shown, while on the right-hand side a multiplot of four projections of the same snapshots is displayed. Slices are centred on the grid's origin and all plots have a width of 80 Mpc. The redshifts corresponding to the snapshots are labelled on the top left corner of each image. Different colour maps are used for each figure in order to best visualise the distribution of dark matter.

that will highlight certain field values in your rendering.

Colour transfer functions convert dataset field values to rendered ray colours, brightnesses, and opacities (emission and absorption red, green, blue, α space (rgba)). Adding discrete features to the transfer function produces isocontours in the field data, which is useful for visualising nested structures in a simulation. Furthermore, continuous features can be applied to the transfer function. Equally spaced isocontours can then be added along with the transfer function, sampling a colourmap to determine the colours of the layers [15]. Appealing transfer functions can be created modified using defaults for our datasets using the `TransferFunctionHelper` instance of [15]. It is the easiest way to construct and customize a colour transfer function for a volume rendering.

Transfer functions that sample colour maps automatically as well as one that provides off-axis line integrals are available together with a transfer function whose colours correspond to Johnson filter-convolved Planck emission with approximate scattering terms, as in [17].

Figure 5 shows an example of a rendered halo using a different number of colour layers in the transfer function.

A plot of the transfer function, along with the cumulative distribution function (CDF) of the density field to see how the transfer function corresponds to structure in the CDF, is illustrated in Figure 6.

Alternatively, we may visualise transfer functions as shown in Figure 7. This illustrates the simulation grid at redshift $z = 0$ with annotations including the simulation time, an axis triad indicating the direction of the coordinate system, and the transfer function on

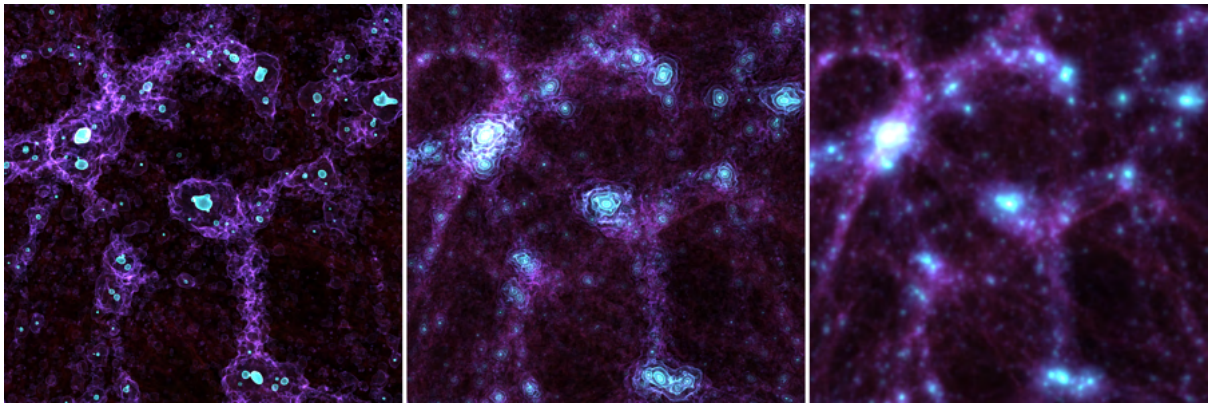

Figure 5: The figure shows a dark matter halo with mass 3.11×10^{14} $M_{\odot}$ (namely, the 7th most massive halo in the simulation) rendered using 4, 10 and 100 colour layers (from left to right, respectively) in the transfer function. In this project we use 10 layers to maintain a balance of isocontour contrast and smoothness.

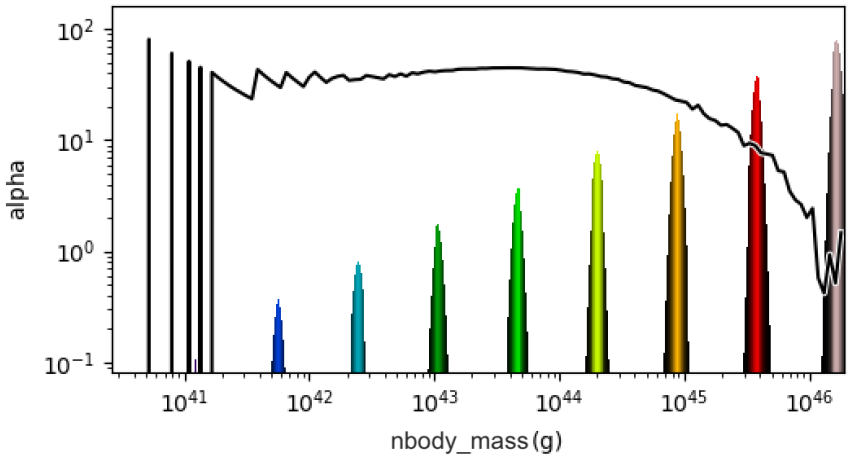

Figure 6: Plot of the transfer function used in rendering the halo shown in Figure 5, using 10 colour layers. Isocontours are placed on the values shown on the horizontal axis.

a volume rendering. The corresponding transfer function plot is shown in Figure 9.

When saving our rendered images we apply sigma clipping to eliminate very bright pixels, producing an image with better contrast. Image values greater than the value of the sigma clipping number multiplied by the standard deviation added to the mean of the image will be clipped before saving. In our visualisations, we set this equal to 4. In addition, we let opacity be calculated on a channel-by-channel basis (useful for opaque renderings as it makes under-dense regions appear opaque).

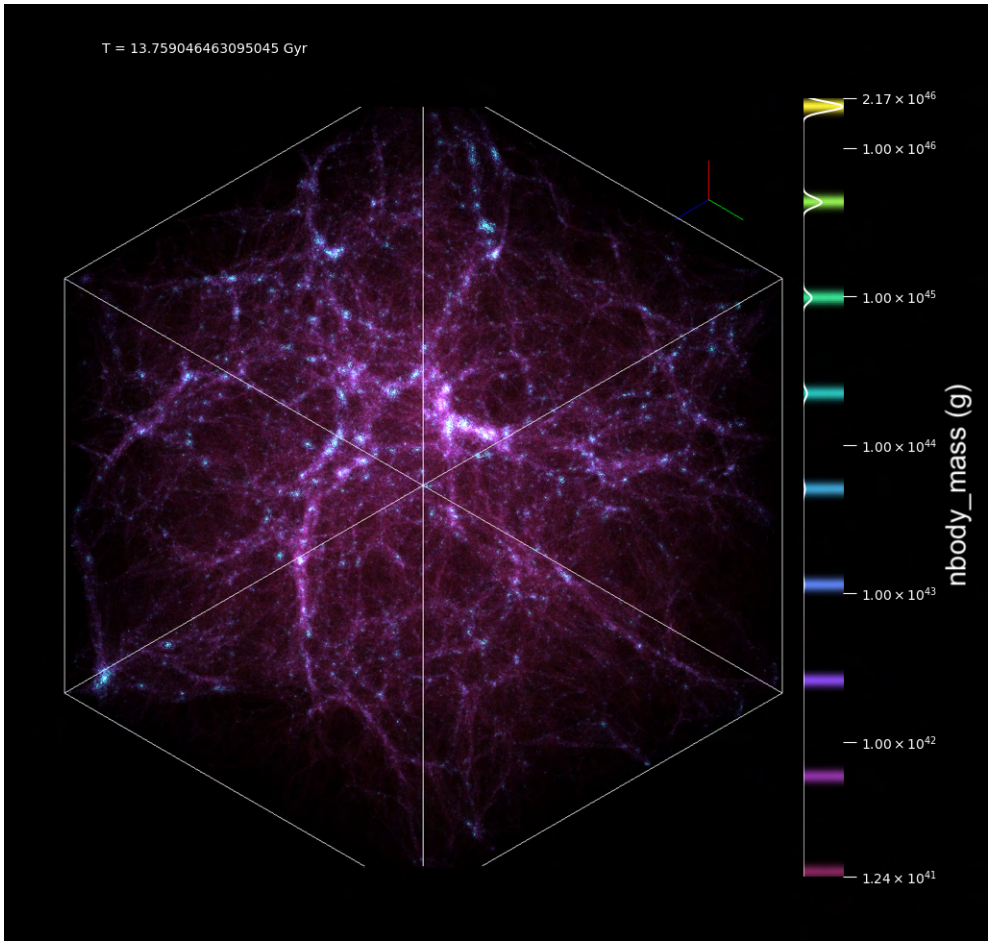

Figure 7: The figure shows the simulation grid at redshift $z = 0$ with annotations such as the simulation time, $T \approx 13.76$ Gyr, the grid boundaries, an axis triad indicating the direction of the Cartesian coordinate system, and the transfer function on a volume rendering. The coloured lines indicate the placing of the isocontours.

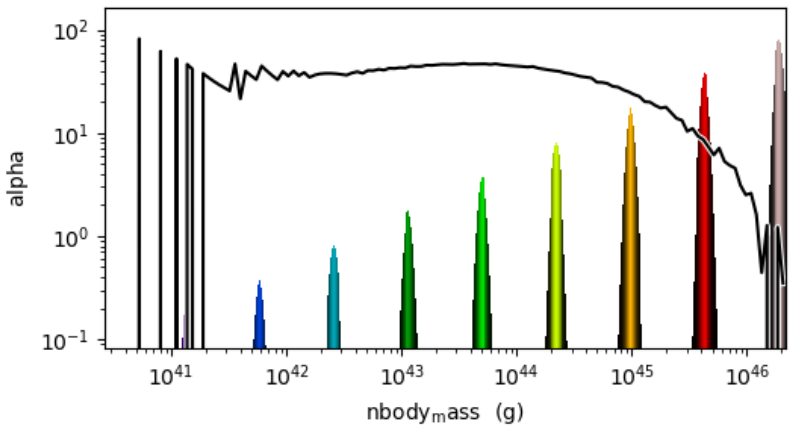

Figure 8: Plot of the transfer function used in rendering the grid box (at $z = 0$) shown in Figure 7 5, using 10 colour layers.

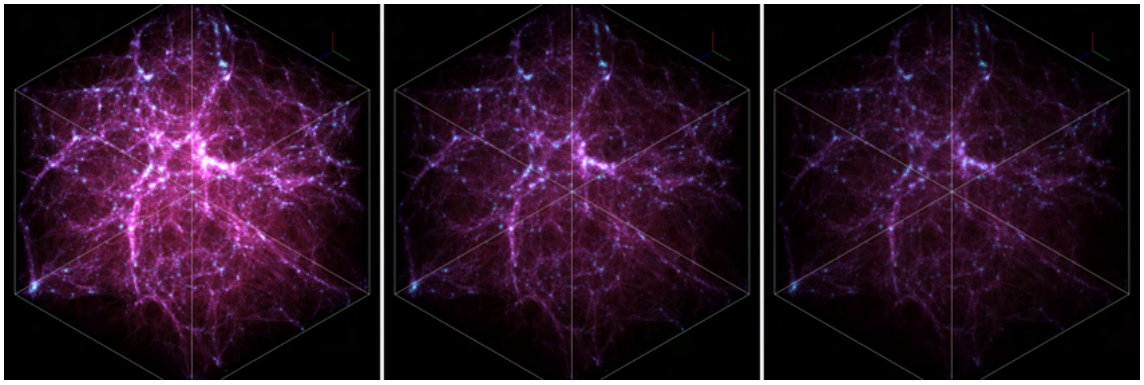

Figure 9: Grid box plotted using different sigma clipping values: 2, 4 and 6 from left to right, respectively. Contrast decreases with higher sigma clipping values, as shown.

.

2.3 Methods for Dark Matter Halo Movies

In order to visualise and perform analyses on dark matter halos we use the `ytree` software package [18] which is an extension of the `yt` analysis toolkit. The merger tree was created using the `consistent-trees` [19] merger tree code, loaded by `ytree`, and visualised with `pydot` [20] and `graphviz` [21]. A visualisation of it is shown in Figure 10.

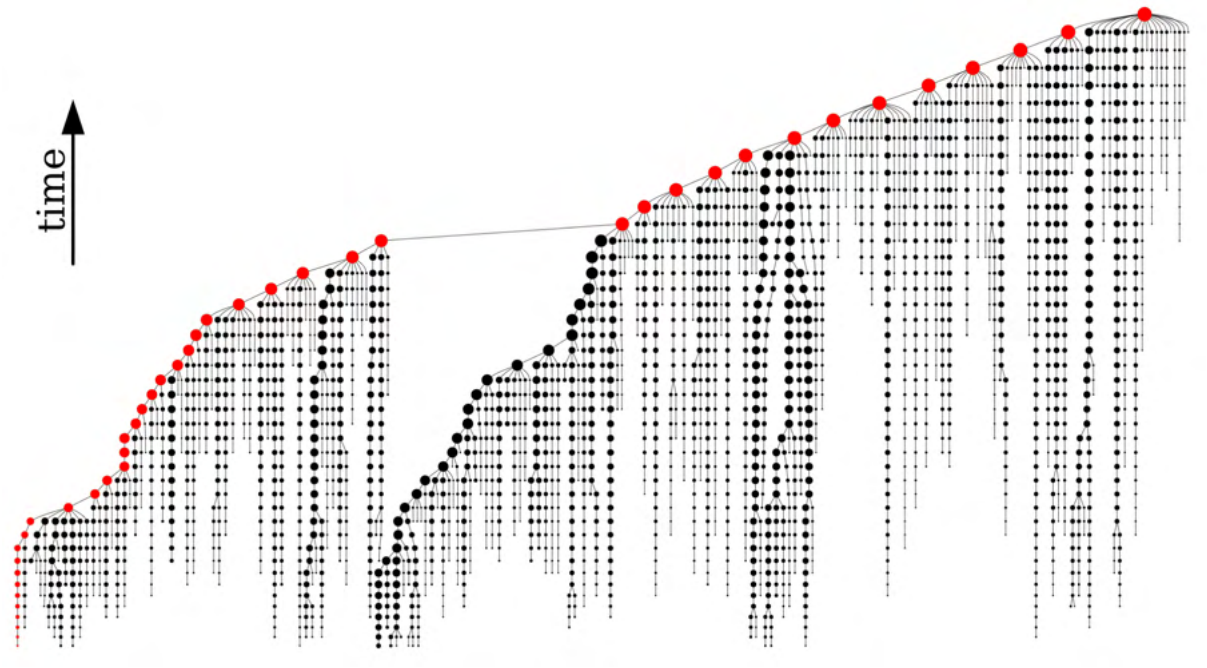

Figure 10: A visualisation of a merger tree. Circles represents halos and lines link them to their ancestors (downward direction) and descendants (upward direction). The diameter of each circle is proportional to the mass of the halo it represents. The line of most massive progenitors of the halo of interest is indicated by the red circles.
Source: Adapted from [22]

2.4 Halo Identification

A major part of our framework is based on allowing the user to choose the halo of interest, based on distinctive properties. The code, which is freely available in this GitHub repository [23], is organised into four classes. The parent class consists of methods that perform operations onto the chosen halo. Specific examples are mentioned in subsequent sections of this paper.

This main class first prompts the user to enter the property of the halo they are interested in, the order of the halo relative to the halo possessing the maximum of that quantity (e.g. the 3rd most massive halo or the halo with the 70th largest angular momentum magnitude). This is done upon initialisation. It then loads a dataset from the disk (via `yt`'s generic load command) followed by merger tree data, using `ytree`, from our halo catalogue.

```
import yt
import ytree
import numpy as np

class ExploreHalo:
    def __init__(self, fnumber=101):
        self.quantity = str(input('Quantity of interest: '))
        self.how_large = int(input(f'Order to largest {self.quantity} (e.g. 1 is
    ↪ maximum and 2 is second to largest): '))
        self.ds = yt.load(f"../dm_gadget/covering_grids/snapshot_{fnumber:03}
    ↪ _covering_grid.h5")
        self.a = ytree.load("../dm_gadget/mergertree_h5/rockstar/rockstar.h5")
        self.field = ("grid", "nbody_mass")
```

The halo is then identified, and returned as an `Arbor` object, via the `load_halo` method, so that its properties can be used in subsequent operations. We use the NumPy [24] package extensively throughout the code.

```
def load_halo(self):
    values = self.a[self.quantity].tolist()
    values.sort()

    halo_index = np.where(self.a[self.quantity] == values[-self.how_large])[0][0]
    my_tree = self.a[halo_index]

    return my_tree
```

2.5 Creating Camera Paths

In terms of Tracking a specific halo through different redshifts involves setting the focus of the camera to the centre of the halo. In the case where a halo undergoes a merger or experiences a shift in its centre of gravity by a noticeable distance between two snapshots. The effect this has on a time series movie that tracks the specific halo is an abrupt (visually

discontinuous) movement of the camera between two frames. In order to resolve this, we use a method to create camera paths.

After executing halo selection or having an array of the nodes to be included in the movie, we obtain their positions. The array of positions is the array we aim to make smoother. We then apply the 1-D convolutional Savitzky-Golay filter built into `SciPy` [25]. The filter uses convolution and the linear least squares method to fit consequtive subsets of nearby data points to a low-degree polynomial in 1-D. The least-squares equations can be solved analytically when the data points are evenly spaced, in the form of "convolution coefficients". These can be applied to all data subsets to create smoothed signal estimates (or derivatives of the smoothed signal) at the central point of each sub-set [26, 27, 28].

When applying the filter in our case, we use a polynomial of degree 8, an odd window size (fraction of the length of the positions array), applied in all three dimensions. This is determined by experimenting with different values and seeing which produced the best result in visualisations. Finally, we loop through this array to set the focus of the camera in each consecutive frame.

Figure 20 uses an example of the path of a specific halo through time to demonstrate that using the Savitzky-Golay algorithm resulted in a smoother camera path in all three dimensions. A limitation of this method is that it does not automatically decide the optimal window size and polynomial degree to be used in the filter. This requires a few manual attempts instead.

This method is used throughout our code, in various methods of our class, such as when tracking halos through time and tracking flybys of halos with chosen properties. It can thus be used for snapshots corresponding to specific, or different redshifts and can thus have many applications.

Along with the halo properties, the path that the camera has to follow, are the two most vital inputs in our framework. As explained in subsequent sections, these are the building blocks that we will use to produce visualisations of different phenomena. These can be accompanied by other basic camera operations, such as the orientation of the camera, frame width, etc., which can also be adjusted by the user.

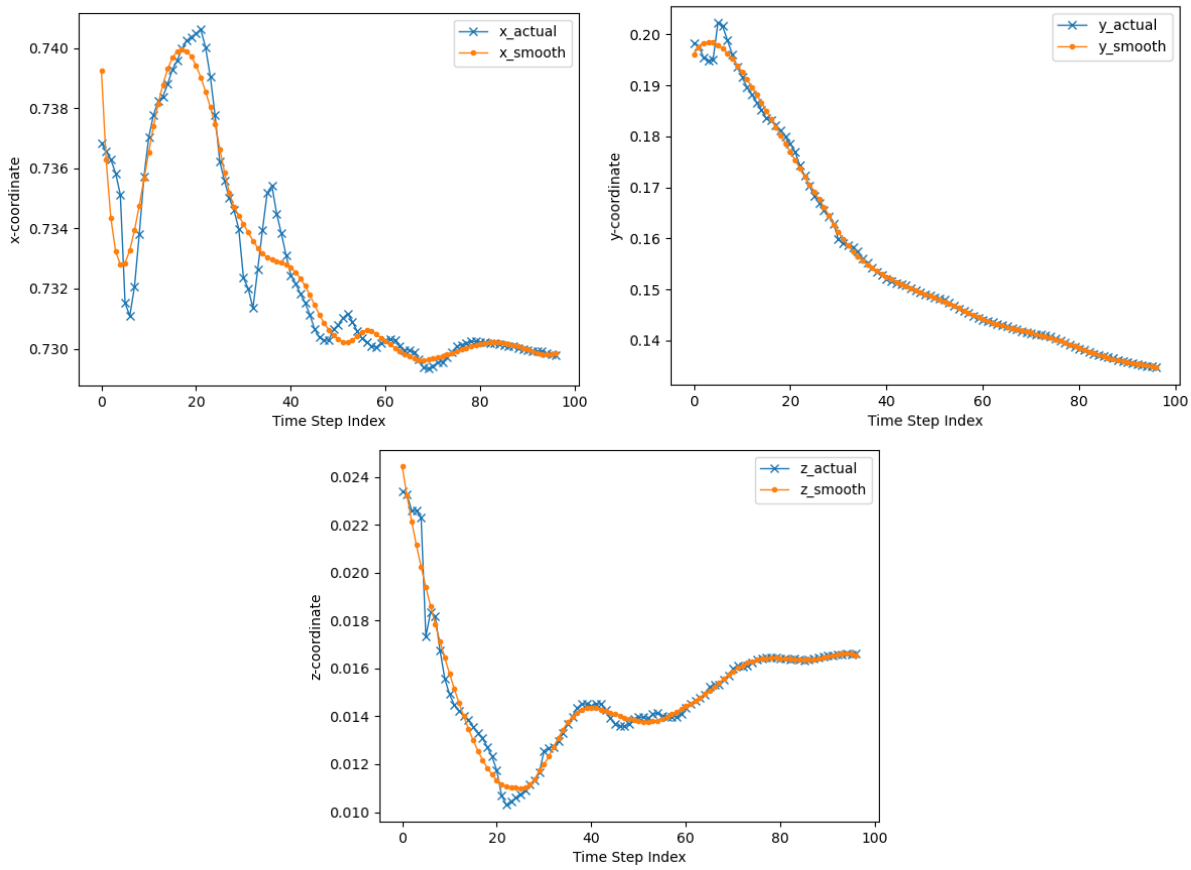

Figure 11: Comparison of the path the camera would take without the smoothing filter ('$[x, y, z]$_actual') to the smoothed path after the filter is applied ('$[x, y, z]$_smooth') in the time series movie of the most massive halo. The polynomial used to produce the path is of degree 6 and an odd window size (fraction of the length of the positions array) is applied in all three dimensions. This uses the path of the most massive halo (with mass 6.901×10^{14} $M_\odot$) in our simulation, through time.

3 Results

3.1 Rotations

Once the halo is loaded as described in Section 2.4, we are able to render and visualise it using the `Camera` interface. An example of a basic operation of the interface is a rotation, as shown in Figure 12.

Since we are dealing with 3-D data, rotations are useful to reveal the morphology of the halo the camera is focused on, as well as objects found behind it, giving a sense of depth to the frame.

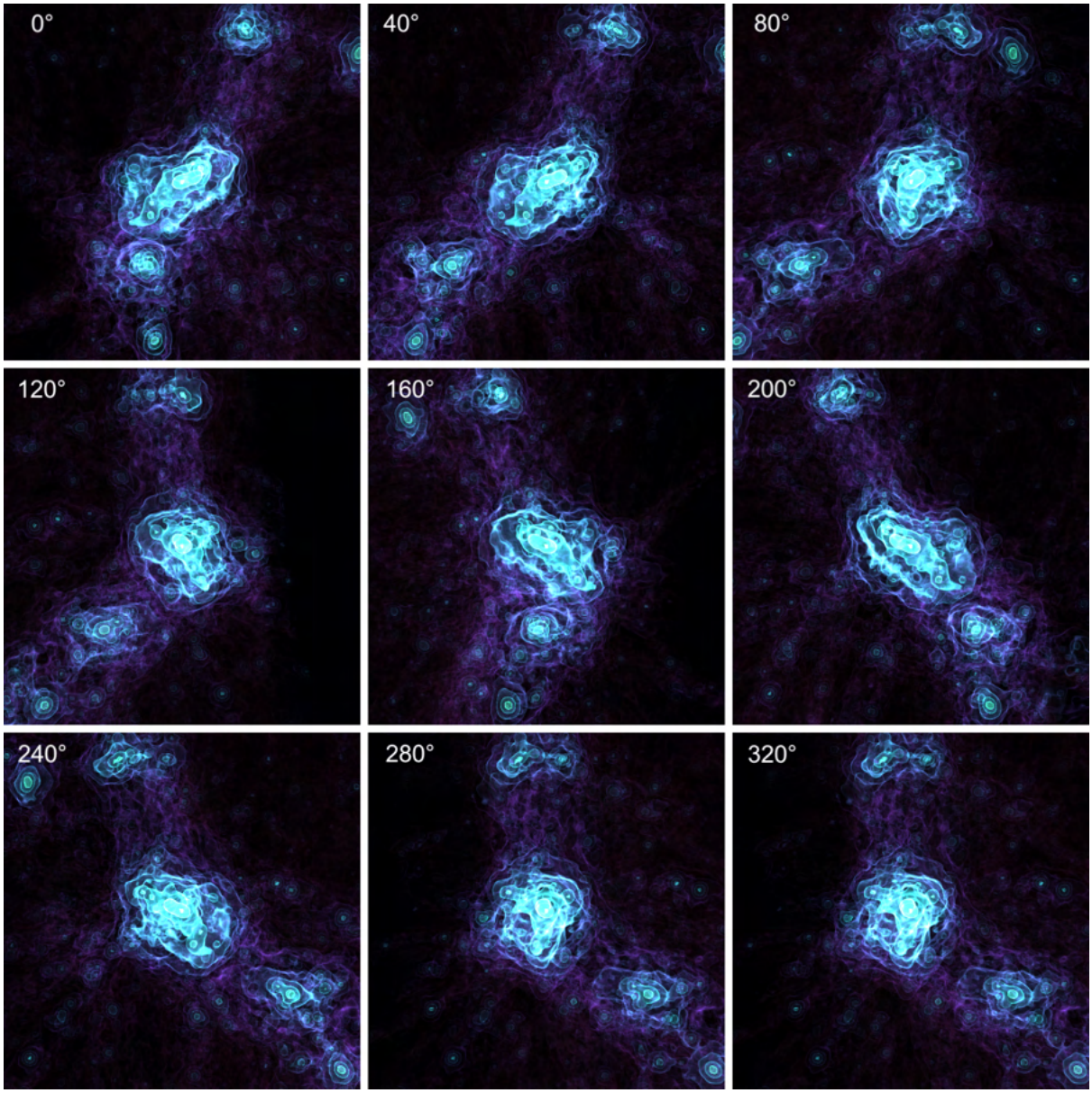

Figure 12: Basic rotation of the halo with the highest mass (6.901×10^{14} $M_\odot$) in the simulation. The image shows screenshots of the movie at different angles of rotation.

3.2 Zooming In

Another example of a basic camera operation of the interface is zooming, as shown in Figure 13. Zooming in on halos with distinctive properties - chosen by the user - is adapted to our framework. A visual consequence of zooming in is that isocontours in the image become more distinguishable. Hence, the closer the camera is to an object, the higher is the number of colours layers required to make the image look smooth. The method used here is for the width of the frame to be adjusted by a factor every time the scene renders for a particular halo. In our example, we focus the camera on a halo of mass 2.109×10^{14} $M_\odot$ and every frame is half the width (scale factor of 1/2) of the previous one. That is, we produce 3 frames of width 40, 20 and 10 Mpc. When making smooth

movies of zooming, we use a smaller factor. For instance, we can use a scale factor equal to 0.99 to produce 200 frames and a smooth movie starting at 40 Mpc and finishing at about 5 Mpc (at 20 frames per second, that would produce a movie with duration 10 seconds).

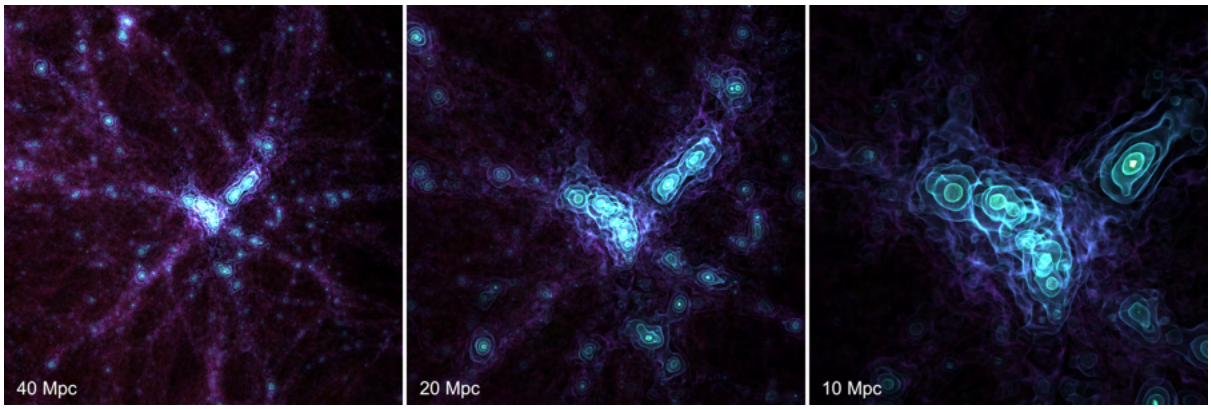

Figure 13: Basic zooming on a halo of mass 2.109×10^{14} $M_\odot$ (13^{th} most massive halo). The image shows screenshots of the movie at different frame widths. When zooming, a scale factor of 1/2 is used, such that the width of the frame changes from 40 to 20 and finally to 10 Mpc, producing the series of images above.

3.3 Tracking Halos Through Time

Using our halo selection framework, we produce a time-series movie of the most massive halo's progenitors, as shown in Figure 14. Each grid in the dataset corresponds to a different cosmic time-frame. Using `ytree` allows us to obtain properties of a halo's main progenitors, including position, mass, etc. Thus, we iterate through all snapshots, loading the grid, focusing our `Camera` object at the centre of the halo's progenitors and rendering. The class method used to find progenitors' properties, and hence implement this, is provided in Appendix A.

This yields a movie of the halo of interest through time, as for instance in Figure 14. We may use the Savitzky-Golay filter described in Section 2.5 to obtain a smooth camera movement throughout the movie.

The visualisation allows us to observe the matter accretion (and mass increase) of the halo of interest with time. Figure 15 shows a plot of the mass and virial radius as functions of redshift and is consistent with the seemingly increasing mass and virial radius of the halo as seen in the visualisation. There is also a jump in both the mass and virial radius in the last recent redshifts, which is consistent with the change in the appearance of the halo in the last two, bottom right, images in the figure. In combination with the spike in the position of the halo in Figure 20 in all axes, this can be potentially attributed to a merger.

A similar effect is observed with most massive halos. The plot below demonstrates the distribution of the mass of the 15,000 most massive halos in the simulation as a function of redshift.

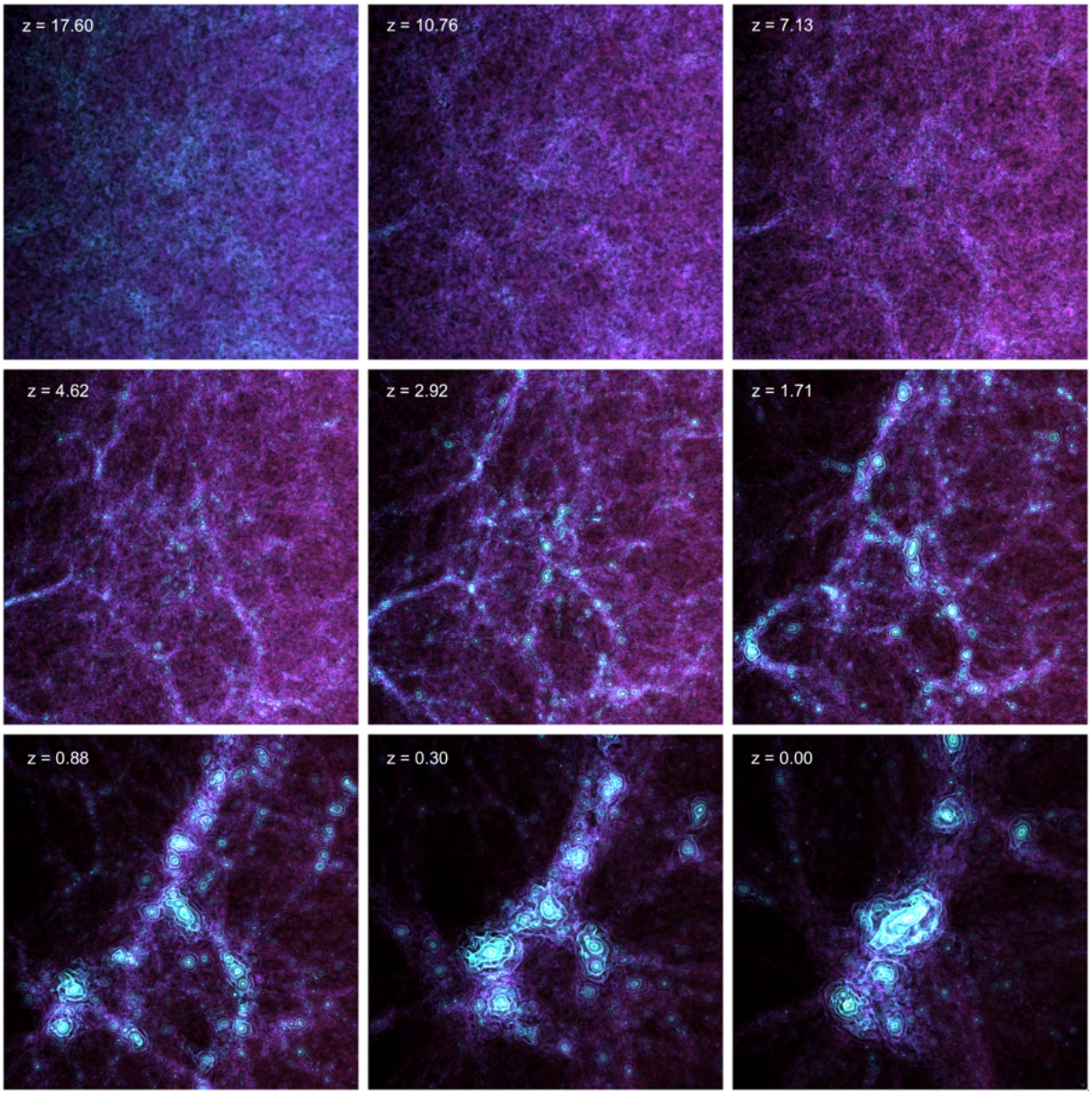

Figure 14: Snapshots from a movie tracking the halo with the highest mass (6.901×10^{14} $M_\odot$) through time and displaying its main progenitors in each image's centre. Redshift decreases from top to bottom and from left to right and its value is annotated on each image. As expected, the halo merges with other halos and its mass increases at an increasing rate through time. The halo's virial radius at $z = 0.0$ is ~ 18.2 kpc.

As expected, for the more massive halos, mass tends to increase with decreasing redshift and, for redshifts $z \lesssim 2.5$, most of them possess a mass in the range $10^{11} - 10^{12}$ $M_\odot$.

Our code could be significantly be improved with better speed. As every frame exists in a different grid snapshot, the interface has to render to construct a different object. As we are dealing with multiple-Gigabyte data, the process of producing this movie is slow. This can be improved by using parallelism strategies used by `yt` [15].

Along with mass, various properties of dark matter halos change with time, as the mor-

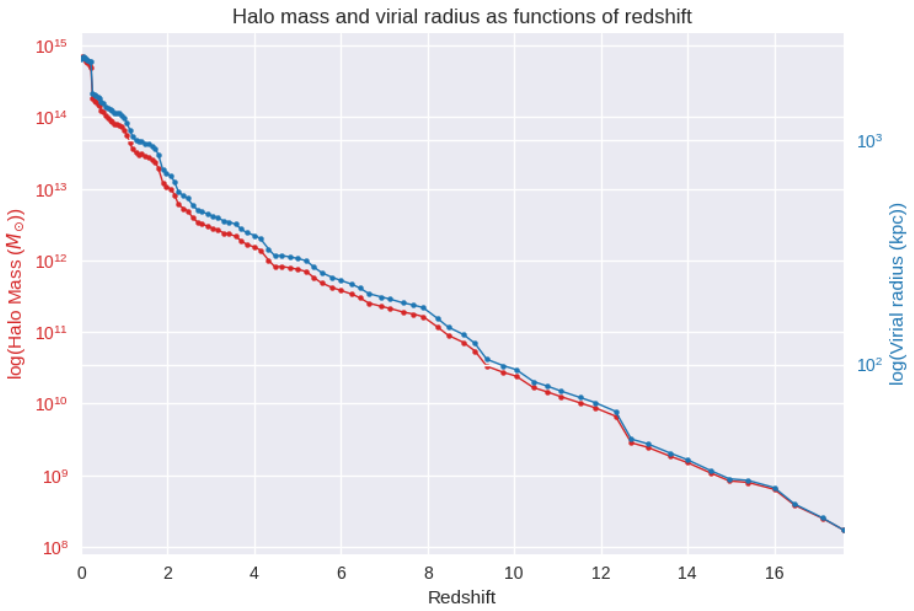

Figure 15: Halo mass and virial radius as functions of redshift for the halo with mass 6.901×10^{14} $M_\odot$. The graph is consistent with the increasing mass and virial radius over time of the halo as seen in the visualisation in Figure 14. There is sharp increase in both the mass and virial radius in the last recent redshifts ($z \lesssim 0.5$), which is consistent with the change in appearance of the halo in the last two, bottom right, images of the figure

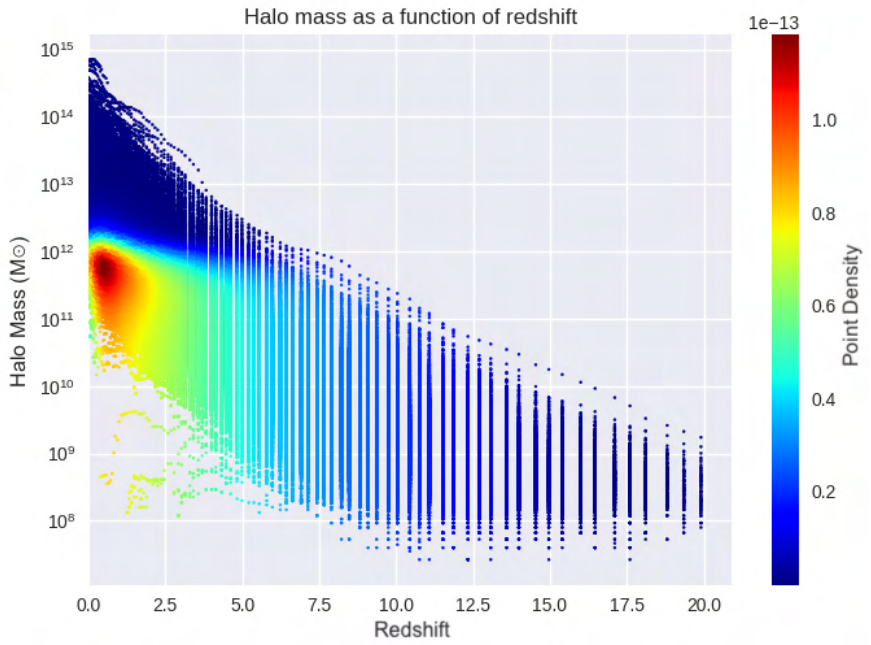

Figure 16: Halo mass as a function of time for the (currently) 15,000 most massive halos. The distribution of halos shows that the majority of the most massive halos possess a mass in the range $10^{11} - 10^{12}$ $M_\odot$ for redshifts $z \lesssim 2.5$.

phology and evolution of the cosmos have high complexity. Using this method to visualise such changes at a glance, gives us a better understanding and intuition of how these changes take place.

3.4 Navigating Through Neighbouring Halos

A crucial, predicted observable of the standard model of cosmology is Universe's baryon content. Baryons make up around 5% of the total energy density in the Universe, according to measurements of the abundances of light elements made in primordial nucleosynthesis and the acoustic peaks in the power spectrum of the cosmic microwave background (CMB) [29].

Only $\sim$ 10% of the total baryon content in the local universe can be accounted for by observations of galaxies and galaxy clusters. According to cosmological simulations, the "missing baryons" are dispersed across the cosmic web's filamentary structures, forming a low-density gas with temperatures of $10^5 - 10^7$ K [30, 31, 32].

Furthermore, it was found that with the exception of the most massive halos, the baryon fraction lies below the expected universal level [33]. Resolving the apparent discrepancy in baryonic mass between the standard cosmological model and galaxy formation theories would thus require for the "missing baryons" to be located.

By the Λ cold dark matter (ΛCDM) model, the matter distribution of the Universe will form a web-like pattern, with galaxies and galaxy clusters embedded in the web's knots and linked by large-scale filamentary and sheet-like structures. Observations and simulations agree in that a large fraction of baryons should be located in filaments and sheets outside of gravitationally bound halos [34, 35, 32]. The baryons are expected to be in a diffuse "warm-hot" state, with low over-densities ($\delta \sim 10$) and temperatures between $10^5 - 10^7$ K [36].

The underlying multi-scale nature of the cosmic web's spatial pattern adds to its complexity, as it includes objects with a vast range of spatial scales and densities [37]. In the study of hierarchical structure development, the physical connection between the growth and features of gravitationally collapsed dark matter halos and the cosmic web environment in which these halos exist is an interesting and complex subject [38, 39, 40, 3, 41, 42].

In this study we implement a new way to visualise the cosmic web environment between neighbouring halos of specific properties, using an approach that has not been implemented elsewhere. The steps followed by our code are as follows:

1. Use the target halo's position as the starting point and initial focus of the camera.
2. Search for halos of a mass comparable to that of the target halo, to increase the chance of seeing filaments and nodes when moving.
3. Calculate the distance between each halo that fulfils the criteria (comparable mass to that of the target halo).
4. Choose the halo (neighbouring halo) that is at a distance closest to the scale we chose (e.g. at a distance equal to twice the diameter of the target halo). This distance can be adjusted, in units of the target halo's diameter, by the user.

5. Take the position of the neighbouring halo to be the final position of the camera when creating a movie.

6. Choose the number and width of the frames to render for the movie.

We could alternatively search the arbor for halos that are found at a range of distances we choose from the target halo and then choose the one with the highest mass in step 4. This could potentially improve the speed of the algorithm.

The code used to find the final position of the camera is included in Appendix B.1. Following this, we create an array of evenly spaced points between the position of the target halo and that of the neighbouring halo. Snapshots of movies we have produced are shown in Figures 17 and 18. An additional example is provided in Appendix B.2.

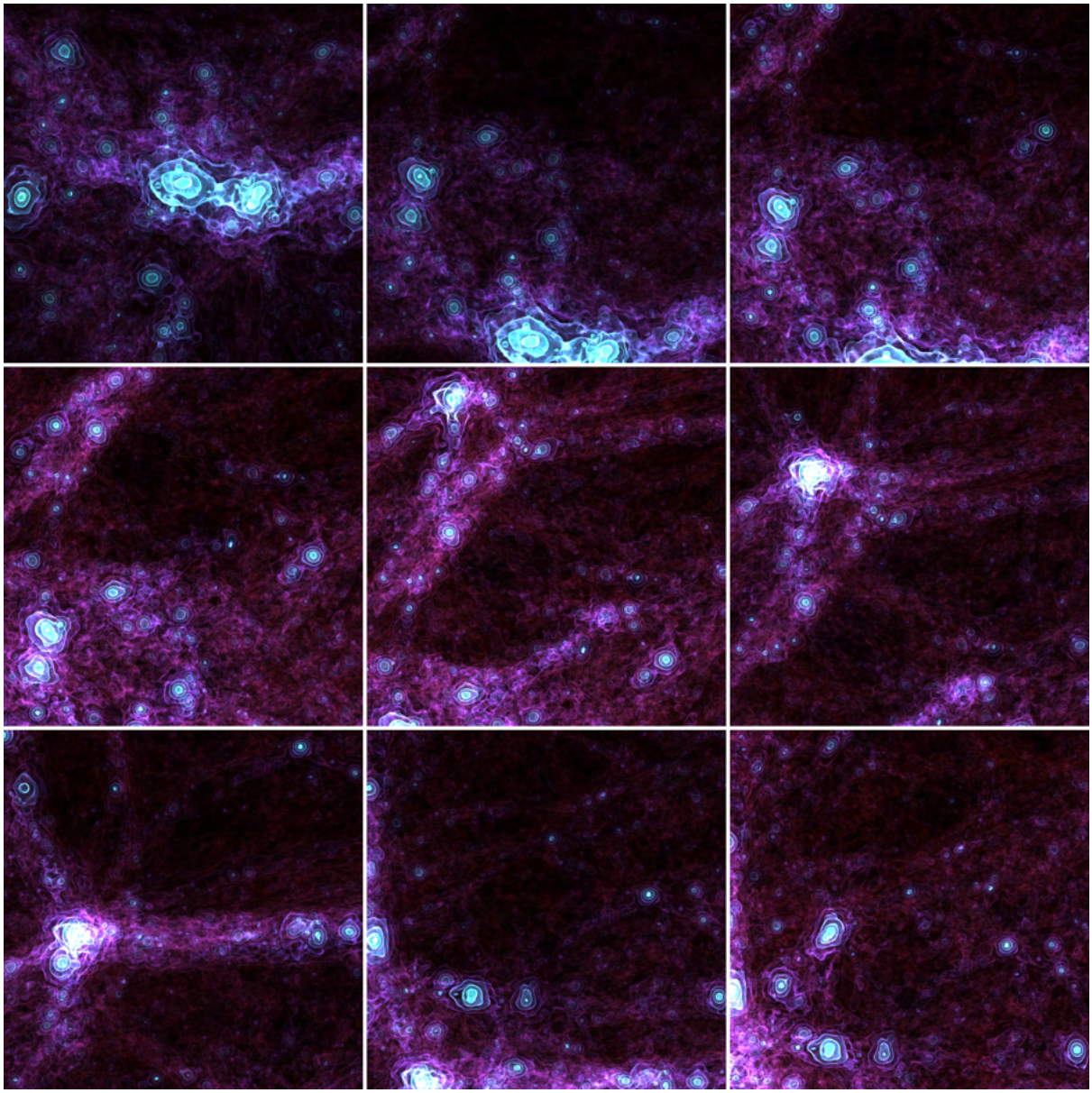

Figure 17: Snapshots from a movie where the camera navigates from a halo of mass 1.59×10^{14} $M_{\odot}$ and virial radius ~ 1.42 Mpc to one that resides 17 Mpc away from the halo's centre. Filaments, sheets and clusters can be seen as the camera moves through the grid. The neighbouring halo (at the final focus point of the camera) is not shown.

We also explored the option of placing the camera within the target halo and moving it through the volume while it faces the direction in which the neighbouring halo is found. Although the movie yielded some interesting frames (Figure 19) containing filaments, sheets, etc., it is not possible to follow the path that connected the two halos and hence visualise their distribution along the path.

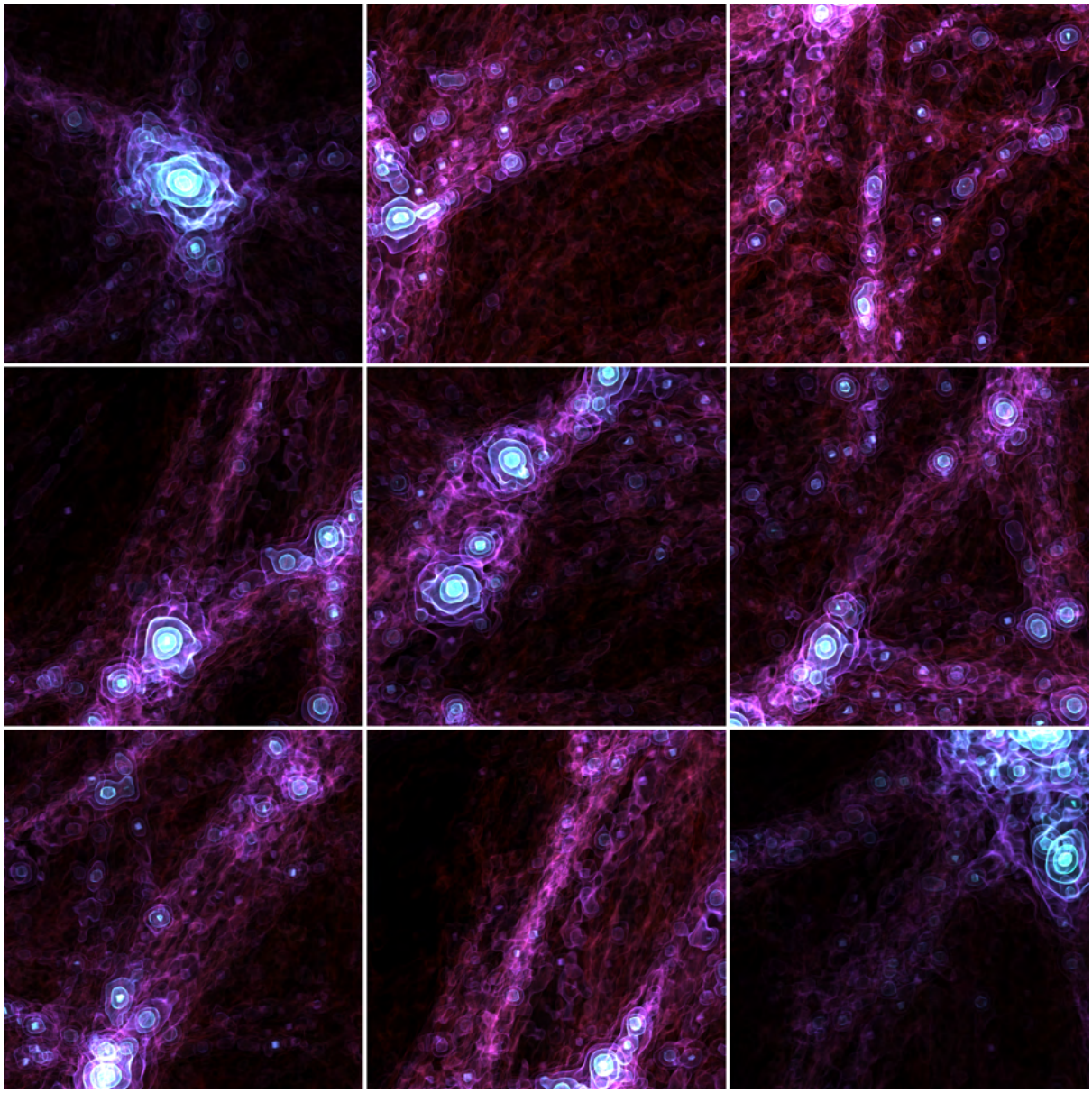

Figure 18: Snapshots from a movie where the camera navigates between a halo of mass 5.929×10^{13} $M_\odot$ to one that is 40 Mpc away. Filaments, voids and nodes can be seen as the camera moves through the grid. The neighbouring halo is seen on the upper right edge of the bottom right image in the Figure.

An improvement would be to integrate a way of following a path of maximum dark matter density to construct a path along filaments instead of a linear one. This would focus the camera on filaments with the highest density and never voids. Overall, the visualisation technique we use is a novel capability that is not generally used in the field. Thereby, this can probe the adoption of visualisation techniques to deepen our understanding of cosmic web structure, and potentially its evolution.

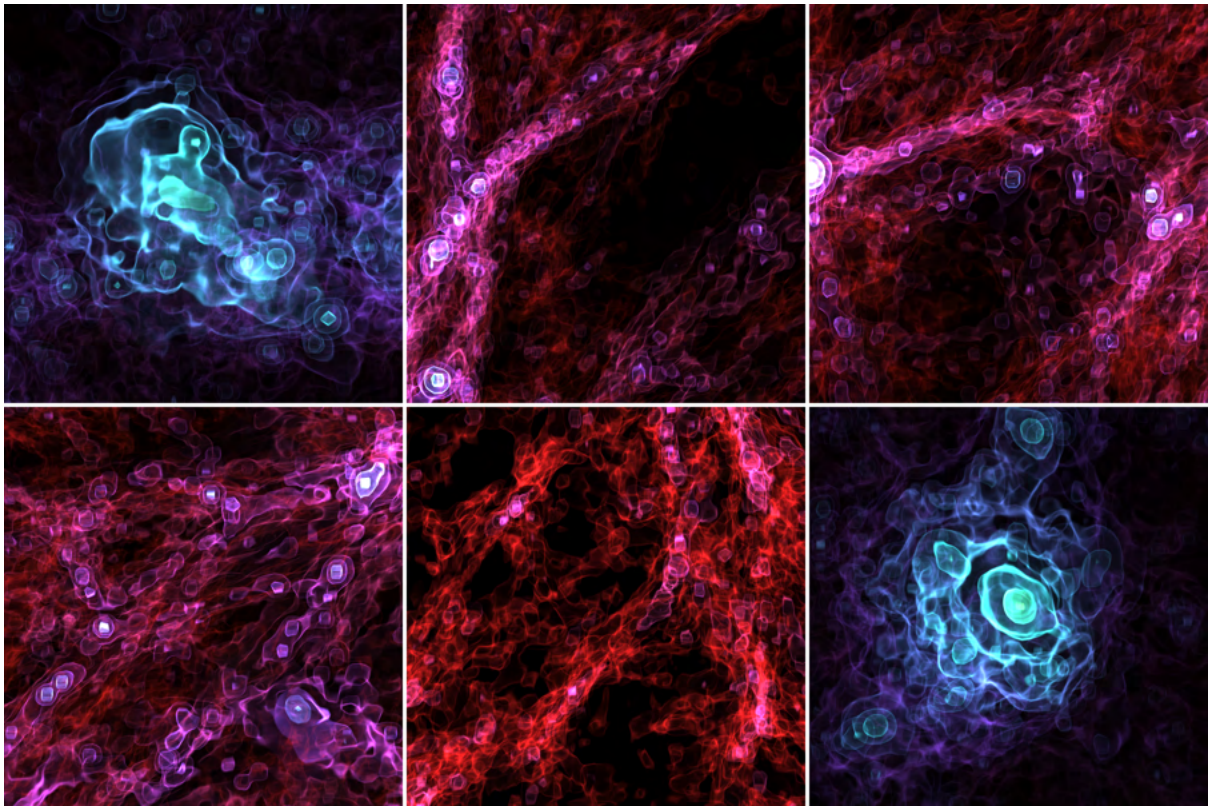

Figure 19: Snapshots from a movie where the camera navigates from a halo of mass $\sim 3.11 \times 10^{14}$ $M_\odot$ to a halo residing 1.81 Mpc away. Filaments, voids and nodes can be seen as the camera moves through the grid, though it is not possible to see how these structures are distributed between the halos.

3.5 Detecting Dark Matter Flybys

Flybys occur when two independent galaxy halos interpenetrate but detach at a later time; this can cause a rapid and significant perturbation in each galaxy, unlike galaxy mergers, which combine two galaxies into one remnant [43].

High-mass halos are constantly disrupted by flybys, preventing them from reaching a dynamical equilibrium. In particular, a study discovered that Milky Way type halos, regardless of mass ratio, have the same number of flybys as mergers for $z \lesssim 2$. Some indication that at high redshift, $z \gtrsim 14$, flybys are as common as mergers were also found. Their findings showed that close flybys played a key role in the creation of the first dark matter halos and their galaxies, and that they continue to influence galaxy evolution at the present epoch [43].

Due to gravitational effects, dark matter halos tend to cluster, making their environment extremely dense at high redshifts, and flyby interactions between halos are frequent at lower redshifts [44, 45]. Two halos can alter their angular momentum in close encounters via a process similar to that which causes planets and moons to tidally lock. Figure 21 depicts two halos passing by each other, each of which can convert its orbital angular momentum to intrinsic angular momentum.

We know that once a halo is formed, its angular momentum is conserved unless it merges

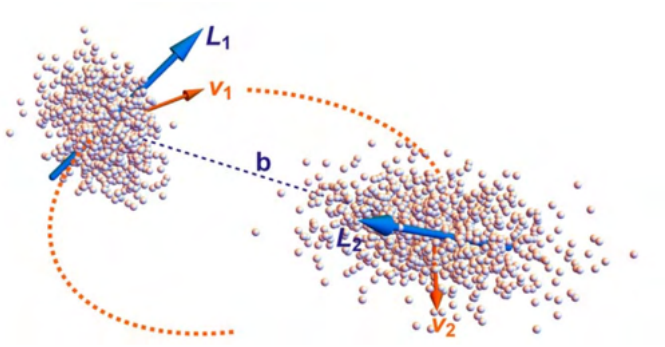

Figure 20: Two halos passing by each other can transfer their orbital angular momentum to intrinsic angular momentum.
Adapted from: [46]

with another halo or a flyby halo. This affects its angular momentum via the tidal locking effect [46] and hence at the early stages of structure development, the angular momentum of halos may be explained entirely by tidal torque theory (TTT). However, TTT's predictions for the alignments is becomes invalid as time passes and flyby incidents become more prevalent.

This can be attributed to their higher susceptibility to the tidal-locking effect compared to other halos. At lower redshifts, the alignment of the fast halos[2] must be the most prominent, but it is less so at higher redshifts. Flyby interactions become crucial in this context because it is probable that fast halos have undergone a flyby, where they acquired their high speed [47].

A recent study focused on another consequence of flyby interactions: the dark matter mass loss of the secondary, intruder galaxy. The results showed that lost mass follows an exponential decay law with impact parameter, so it would be fairly safe to assume that close or strong flybys could lead to the formation of ultra-diffuse and dark matter deficient galaxies with just the right impact parameters [44].

These consequences of flyby interactions attests their pivotal role in our understanding of dark matter and galaxy evolution.

In this study we utilise the properties of our simulation, `yt` and `ytree` to develop flyby identification algorithm and facilitate the visualisation analysis of such effects.

We begin by considering the gravitational two-body problem, which can be solved analytically using the average velocity and mass inside the virial radii of the two interacting halos through transitioning to the equivalent one-body problem of the reduced mass system [48]. By the equation of motion:

$$\mu \ddot{\boldsymbol{r}} = -G\frac{M_h M_s}{r^2}\frac{\boldsymbol{r}}{r} \tag{1}$$

with M_h and M_s being the mass inside the virial radius of the more (primary) and less

[2]Fast halos are those with velocities larger than twice the median [47].

(secondary) massive of the interacting halos, respectively. The reduced mass, μ, is defined by:

$$\mu \equiv \frac{M_h M_s}{M_h + M_s} \tag{2}$$

Equation 1 describes the motion of a fictitious particle of mass μ moving around a point at a distance $r = |\boldsymbol{r_h} - \boldsymbol{r_s}|$. We calculate the total energy and orbital angular momentum to describe the particle's orbit:

$$E = \frac{1}{2}\mu \dot{r}^2 - G\frac{M_h M_s}{r} \qquad \text{and} \qquad L = \mu \boldsymbol{r} \times \dot{\boldsymbol{r}} \tag{3}$$

respectively. If a halo pair has a negative energy, the encounter is classified as a flyby, while a positive energy characterises two halos that are gravitationally bound and that will subsequently merge [49, 50].

We generally classify a flyby using spatial and dynamical information. The distinction between isolated hosts and subhalos of dark matter halos is critical for understanding structure formation and the galaxy–halo relationship. Subhalos are most commonly defined as residing within a spherical overdensity boundary, such as the virial radius. Thus, we limit our search to halos that existed outside the boundary either within a few Gyr before and or after the interaction occurred. As a result, we eliminate subhalos and potential mergers from our analysis. We also require the total energy of the system as in Equation 3 to be negative, so as to eliminate the possibility of the halos being bound and thus merger events. To make sure that the flyby is not actually a merger of a bound halo system, we also check that the separation of the two halos increased again after the flyby interaction. We do this by looking for minima in the magnitude of the distances between the main halo and the flyby over cosmic time (i.e., require that the distance between the two halos at $z = 0$ is larger than the minimum distance at any other point in time).

An additional constraint we may impose is for the minimum distance between them to be smaller than the sum of their radii so that the halos inter-penetrate but detach at a later time without merging. In addition, spurious interactions (e.g. grazing flybys) could be eliminated by only choosing the flybys that have a duration longer than half the crossing time, for a more specific approach. Another potential improvement would be to investigate flybys based on the circularity, ϵ, of their orbits which can be calculated from the orbit's eccentricity, e, given by [51]:

$$e = \sqrt{1 + \frac{2EL^2}{\mu(GM_h M_s)^2}}. \tag{4}$$

Orbital circularity is defined as the ratio of the orbital angular momentum to that of the circular orbit with the same energy and can also be related to eccentricity. The circularity then takes the form:

$$\epsilon = \sqrt{1 - e^2} \tag{5}$$

which is another parameter commonly used to describe bound orbits with $E < 0$ [50].

Once our framework identifies a flyby, it produces a time series projection movie of the flyby interaction and annotates the image to indicate the secondary and primary halo, as in Figure 21. It then plots the path of the secondary halo relative to the main halo (while accounting for boundary conditions) as shown in Figure 22.

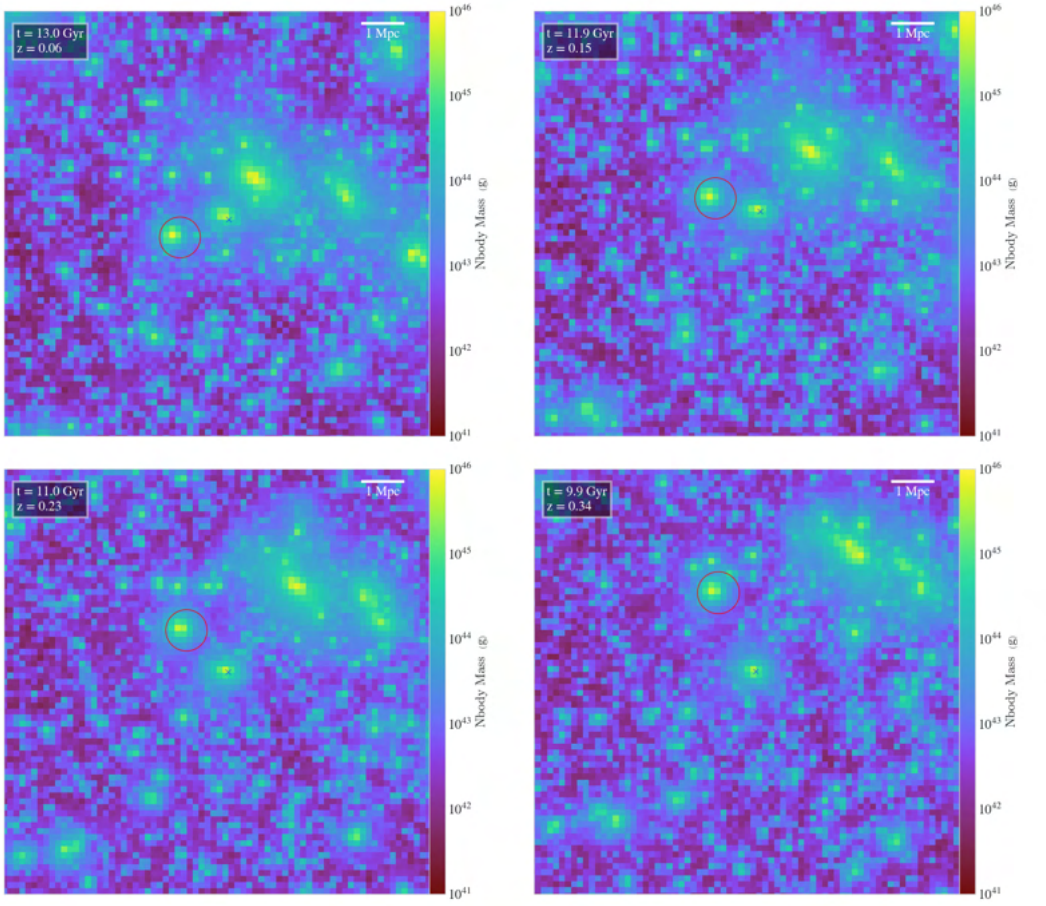

Figure 21: The flyby is circled in red while the primary halo is marked with a blue cross. The images show the interaction, with redshift increasing from left to right and top to bottom.

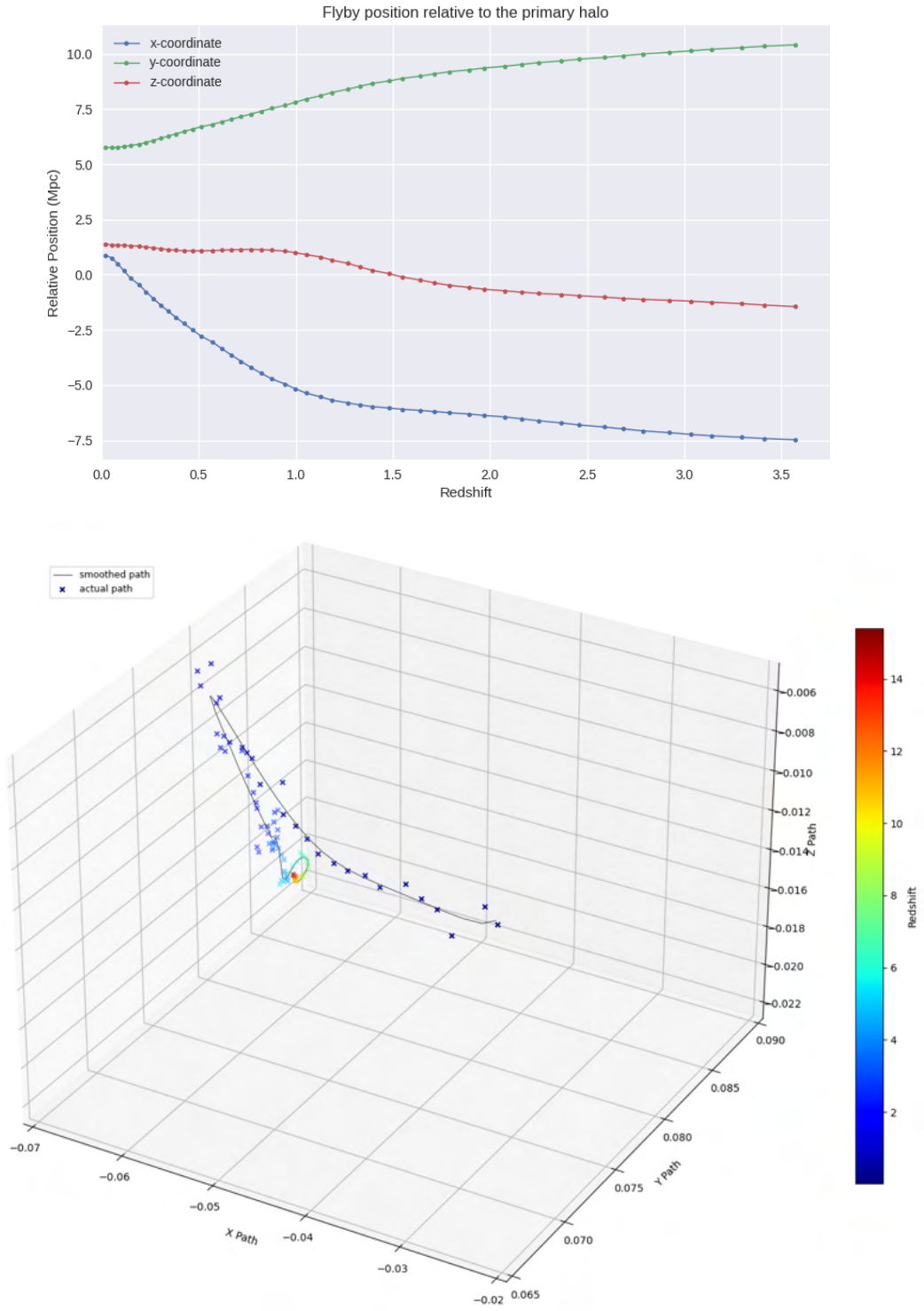

Figure 22: a) The plot at the top shows the position of the flyby relative to the primary halo in each dimension separately. the interaction took place throughout $z \lesssim 2.0$.
b) The figure at the bottom shows a 3-D visualisation of the flyby halo's position relative to the primary halo in the Cartesian system. The units used are unitary units (where the grid box is normalised and has sides of unit length). The squares represent the actual path followed by the flyby relative to the halo while the line plots the smoothed path using the method described in Section 2.5.

The magnitude of the angular momentum of the primary halo in this case varied as shown in Figure 23. It should be noted that this change in momentum could be attributed to many interactions including possible mergers, however, the specific flyby possesses a mass comparable to that of the primary halo. Thereby, the interaction most likely had a significant contribution to this effect.

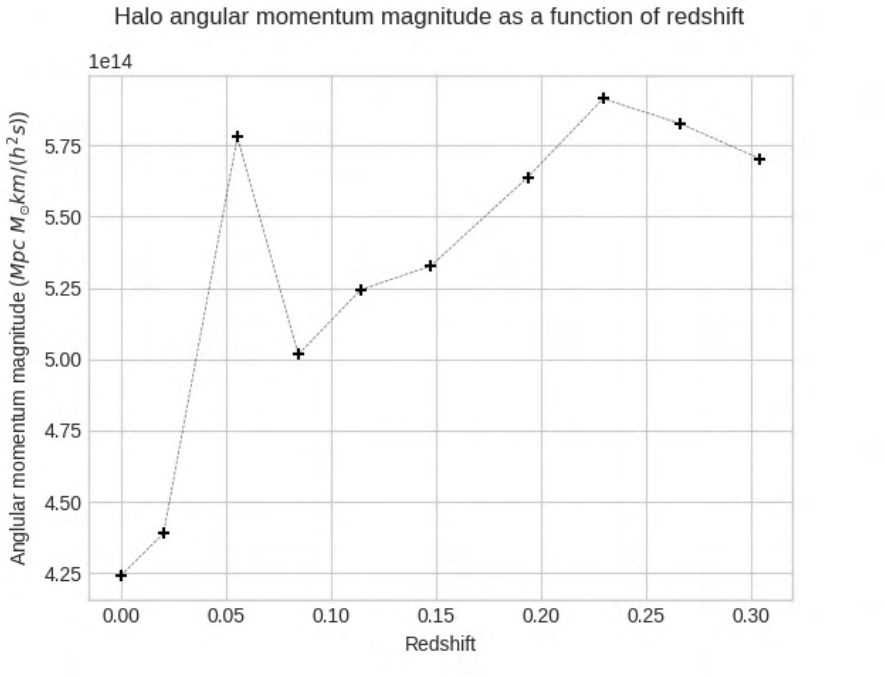

Figure 23: Angular momentum magnitude of the main halo during the flyby interaction. in between the redshifts when the interaction occurred, a rapid increase is observed.

Overall, this method proved effective in identifying halos and uses constraints that satisfy the physical requirements for the interaction to be classified as a flyby. Further development of this method could include distinguishing between different types of flybys and perhaps integrating mergers in the analysis.

4 Discussion and Conclusions

Visualisation is one of the most effective techniques for understanding and presenting a scientific dataset, allowing users to comprehend its basic features and properties ostensively. In order to fulfil such goals, appropriate graphical solutions must be developed and implemented. Such solutions can vary depending on the intended audience, for example, whether it is professionals or the general public. Furthermore, different application fields may have unique requirements, and different datasets may have unique characteristics that must be addressed.

The main methods of our software include identifying halo properties based on user input and constructing a path for the `Camera` interface to navigate in and focus on. From there, we develop functionalities such as navigating between neighbouring halos, following halos through their evolution in time and detecting flybys.

Our method is effective in making movies of halos' evolution over time and several basic camera operations such as rotations and zooming. It is capable of displaying the distribution of structures (sheets, filaments, clusters) in the cosmic web and it accepts many parameters as user input. Hence, it requires little technical knowledge from the user's side. In addition, it effectively identifies flybys, though it can significantly benefit from a boost in processing speed.

Like many of the software tools used to analyse scientific data, developed today, our tool faces the challenge of vast data volumes. Visualisation software must be able to handle gigabytes of data at once while delivering a satisfactory outcome with a reasonable speed. To achieve the best possible results, the algorithm must be tailored to the data. In order to handle such large volumes of data processing efforts, 64bit architectures, big memory, and multiprocessor systems must be supported and utilised. In this instance, the challenge of dealing with large data volumes can also be significantly improved by modifying the software to use `yt`'s parallelisation strategies.

As a result of the amplified complexity of the physical systems described by modern cosmological simulations, visualisations of them are fascinating to the public and have thus grown ever so close to the beauty of real observations. Apart from reflecting the intriguing nature of astronomy, visualisations provide a fresh glimpse into the physical large-scale features of the cosmic web, including voids, filaments and halos.

Acknowledgements

Faidra thanks Dr. Britton D. Smith and Professor Sadegh Khochfar for providing valuable advice and guidance throughout the development of this project. Thanks to the University of Edinburgh School of Physics and Astronomy and the Senior Honours Project course staff for making this project possible.

References

[1] Linde, A. Inflationary Cosmology. In Lemoine, M., Martin, J. & Peter, P. (eds.) *Inflationary Cosmology*, vol. 738, 1 (2008).

[2] Gao, L., Springel, V. & White, S. D. M. The age dependence of halo clustering. **363**, L66–L70 (2005). astro-ph/0506510.

[3] Bond, J. R., Kofman, L. & Pogosyan, D. How filaments of galaxies are woven into the cosmic web. **380**, 603–606 (1996). astro-ph/9512141.

[4] White, S. D. M., Frenk, C. S., Davis, M. & Efstathiou, G. Clusters, Filaments, and Voids in a Universe Dominated by Cold Dark Matter. **313**, 505 (1987).

[5] Oei, M. S. S. L. *et al.* Filamentary Baryons and Where to Find Them: A forecast of synchrotron radiation from merger and accretion shocks in the local Cosmic Web. *arXiv e-prints* arXiv:2203.05365 (2022). 2203.05365.

[6] Bardeen, J. M., Bond, J. R., Kaiser, N. & Szalay, A. S. The Statistics of Peaks of Gaussian Random Fields. **304**, 15 (1986).

[7] Forero-Romero, J. E., Hoffman, Y., Gottlöber, S., Klypin, A. & Yepes, G. A dynamical classification of the cosmic web. **396**, 1815–1824 (2009). 0809.4135.

[8] Libeskind, N. I. *et al.* Tracing the cosmic web. **473**, 1195–1217 (2018). 1705.03021.

[9] Codis, S. *et al.* Connecting the cosmic web to the spin of dark haloes: implications for galaxy formation. **427**, 3320–3336 (2012). 1201.5794.

[10] Ade, P. A. R. *et al.* Planck 2015 results XIII. Cosmological parameters. *ASTRONOMY & ASTROPHYSICS* **594** (2016).

[11] Wechsler, R. H. & Tinker, J. L. The Connection Between Galaxies and Their Dark Matter Halos. **56**, 435–487 (2018). 1804.03097.

[12] Angulo, R. E. & Hahn, O. Large-scale dark matter simulations. *Living Reviews in Computational Astrophysics* **8** (2022).

[13] Dolag, K., Reinecke, M., Gheller, C. & Imboden, S. Splotch: visualizing cosmological simulations. *New Journal of Physics* **10**, 125006 (2008).

[14] Croton, D. J. *et al.* The many lives of active galactic nuclei: cooling flows, black holes and the luminosities and colours of galaxies. **365**, 11–28 (2006). astro-ph/0508046.

[15] Turk, M. J. *et al.* yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data. *The Astrophysical Journal Supplement Series* **192**, 9 (2011). 1011.3514.

[16] Springel, V., Yoshida, N. & White, S. D. M. GADGET: a code for collisionless and gasdynamical cosmological simulations. **6**, 79–117 (2001). astro-ph/0003162.

[17] Kaehler, R. *et al.* Eurographics/ieee vgtc workshop on volume graphics (2006).

[18] Smith, B. & Lang, M. Ytree: Merger-Tree Toolkit (2018).

[19] Behroozi, P. S. *et al.* Gravitationally Consistent Halo Catalogs and Merger Trees for Precision Cosmology. **763**, 18 (2013). 1110.4370.

[20] Pydot. Pydot. *GitHub repository* (2008). URL https://github.com/pydot/pydot. GitHub.

[21] Gansner, E. R. & North, S. C. An open graph visualization system and its applications to software engineering. *Software: practice and experience* **30**, 1203–1233 (2000).

[22] Smith, B. D. & Lang, M. ytree: A python package for analyzing merger trees. *Journal of Open Source Software* **4**, 1881 (2019). URL https://doi.org/10.21105/joss.01881.

[23] Antoniadou, F. Visualizing the Cosmic Web and Dark Matter Haloes.

[24] Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020). URL https://doi.org/10.1038/s41586-020-2649-2.

[25] Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261–272 (2020).

[26] Schafer, R. W. What is a savitzky-golay filter? [lecture notes]. *IEEE Signal Processing Magazine* **28**, 111–117 (2011).

[27] Press, W. H. & Teukolsky, S. A. Savitzky-golay smoothing filters. *Computers in Physics* **4**, 669–672 (1990).

[28] Savitzky, A. & Golay, M. J. E. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry.* **36** (1964).

[29] Planck Collaboration *et al.* Planck 2015 results. XIII. Cosmological parameters. **594**, A13 (2016). 1502.01589.

[30] Persic, M. & Salucci, P. The baryon content of the universe. **258**, 14P–18P (1992). astro-ph/0502178.

[31] Fukugita, M. & Peebles, P. J. E. The Cosmic Energy Inventory. **616**, 643–668 (2004). astro-ph/0406095.

[32] Shull, J. M., Smith, B. D. & Danforth, C. W. The Baryon Census in a Multiphase Intergalactic Medium: 30% of the Baryons May Still be Missing. **759**, 23 (2012). 1112.2706.

[33] Marshall, M. A., Mutch, S. J., Qin, Y., Poole, G. B. & Wyithe, J. S. B. Dark-ages reionization and galaxy formation simulation - XVII. Sizes, angular momenta, and morphologies of high-redshift galaxies. **488**, 1941–1959 (2019). 1904.01619.

[34] Cen, R. & Ostriker, J. P. Where Are the Baryons? **514**, 1–6 (1999). astro-ph/9806281.

[35] Penton, S. V., Stocke, J. T. & Shull, J. M. The Local Lyα Forest. IV. Space Telescope Imaging Spectrograph G140M Spectra and Results on the Distribution and Baryon Content of H I Absorbers. **152**, 29–62 (2004). astro-ph/0401036.

[36] de Graaff, A., Cai, Y.-C., Heymans, C. & Peacock, J. A. Probing the missing baryons with the Sunyaev-Zel'dovich effect from filaments. **624**, A48 (2019). 1709.10378.

[37] van de Weygaert, R. Voids and the Cosmic Web: cosmic depression & spatial complexity. In van de Weygaert, R., Shandarin, S., Saar, E. & Einasto, J. (eds.) *The Zeldovich Universe: Genesis and Growth of the Cosmic Web*, vol. 308, 493–523 (2016). 1611.01222.

[38] White, S. D. M. & Silk, J. The growth of aspherical structure in the universe: Is the Local Supercluster an unusual system? **231**, 1–9 (1979).

[39] Eisenstein, D. J. & Loeb, A. An Analytical Model for the Triaxial Collapse of Cosmological Perturbations. **439**, 520 (1995). astro-ph/9405012.

[40] Bond, J. R. & Myers, S. T. The Peak-Patch Picture of Cosmic Catalogs. I. Algorithms. **103**, 1 (1996).

[41] Giuricin, G., Mezzetti, M. & Salucci, P. Observational cosmology: The development of galaxy systems. *Observational Cosmology: The Development of Galaxy Systems* **176** (1999).

[42] Sheth, R. K. & Tormen, G. Large-scale bias and the peak background split. **308**, 119–126 (1999). astro-ph/9901122.

[43] Sinha, M. & Holley-Bockelmann, K. A First Look at Galaxy Flyby Interactions. I. Characterizing the Frequency of Flybys in a Cosmological Context. **751**, 17 (2012). 1103.1675.

[44] Mitrašinović, A. Dark matter mass loss in galaxy flybys: dependence on impact parameter. *arXiv e-prints* arXiv:2203.08665 (2022). 2203.08665.

[45] L'Huillier, B., Park, C. & Kim, J. The ecology of dark matter haloes -I. The rates and types of halo interactions. **451**, 527–538 (2015). 1505.00788.

[46] Ebrahimian, E. & Abolhasani, A. A. Dynamical Tidal Locking Theory: A New Source of the Spin of Dark Matter Halos. **912**, 57 (2021). 2007.13148.

[47] Ebrahimian, E. & Abolhasani, A. Spin alignment of dark matter halos: Fast halos. *The Astrophysical Journal* **926**, 200 (2022). URL https://doi.org/10.3847/1538-4357/ac497c.

[48] Goldstein, H., Poole, C. & Safko, J. *Classical mechanics* (2002).

[49] An, S.-H., Kim, J. H., Yun, K., Kim, J. & Yoon, S.-J. Fly-By Encounters Between Dark Matter Halos in Cosmological Simulations. *Publication of Korean Astronomical Society* **30**, 331–333 (2015).

[50] Khochfar, S. & Burkert, A. Orbital parameters of merging dark matter halos. **445**, 403–412 (2006). astro-ph/0309611.

[51] Abraham, R. & Marsden, J. E. *Foundations of Mechanics* (1978).

Appendices

A Identifying halo properties

```
def get_properties(self):
    # gets target halo and its progenitors' properties

    my_tree = self.load_halo()

    halo_position = np.array(my_tree["position"].to("unitary"))
    halo_radius = my_tree["virial_radius"].to("unitary").value
    prog_redshifts = my_tree["prog", "redshift"]
    prog_snaps = my_tree["prog", "Snap_idx"]

    # get progenitor positions
    prog_positions = np.array(my_tree["prog", "position"].to("unitary"))
    halo_mass = my_tree["mass"].to('Msun').value
    print(f"Halo mass: {my_tree['mass'].to('Msun')}")
    print(f"Halo radius: {my_tree['virial_radius']}")

    return halo_position, halo_radius, halo_mass, prog_snaps, prog_positions
```

B Finding nearest neighbours

B.1 Code to identify neighbour

```
def get_neighbour(self, how_far=5.):
    '''
    Gets neighbouring halo based on criteria
    This can be modified to use a sphere object instead since it is specific to one halo
    '''
    initial_position, _, _, prog_snaps, _ = self.get_properties()
    self.how_far = float(input("How far should I search (in units of the halo's diameter)? "))

    # empty list for halo distances
    distances = []
    positions = []

    for halo in a.select_halos("(tree['forest', 'mass'].to('Msun') > upper_bound_mass) & (tree['forest', 'mass'].to('Msun') > lower_bound_mass)"):        # exclude the most massive halo
        pair_pos = halo["position"].to("unitary")

        # calculate distance to target halo
        distance = np.linalg.norm(np.array(pair_pos) - np.array(initial_position))
        positions.append(np.array(pair_pos))
            distances.append(distance)

        if len(distances) == 0:
            print('No haloes found.')

        # now we want to choose a halo that is neither too close nor too far from the target
        idx = (np.abs(distances - self.how_far * np.array(self.width))).argmin()
        final_position = positions[idx]

        return initial_position, prog_snaps, final_position
```

B.2 Movie Example

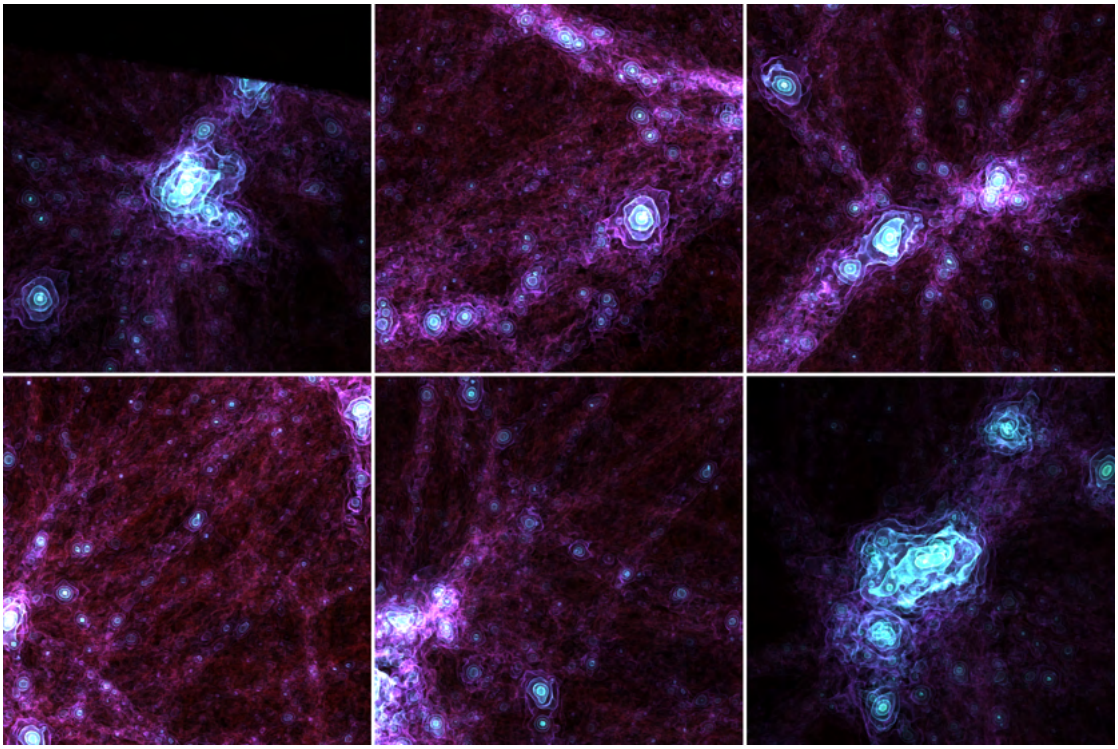

Figure 24: Snapshots from navigation starting from a halo of mass 1.927×10^{14} $M_\odot$ to a halo found 336 kpc away which is also the most massive one, with mass 6.901×10^{14} $M_\odot$.

C Flyby Detection Algorithm

```
G_value = 6.67e-11

def make_halo_region(self, how_far=10.):
    '''
    Create data object that contains halo region
    '''
    initial_position, initial_radius, _, _, _ = self.get_properties()
    ds = self.a.ytds
    sphere = ds.sphere(initial_position, (initial_radius*how_far, "unitary"))
    center = initial_position

    return sphere


def get_flybys_energy(self):

    my_tree = self.load_halo()
    sphere = self.make_halo_region()

    initial_position, initial_radius, primary_mass, _, prog_positions= self.
    ↪ get_properties()
    primary_radii = (my_tree["prog", "virial_radius"].to("unitary")).value
    primary_mass = my_tree["mass"].value

    found = 0           # to keep track of identified flybys
    my_momenta = my_tree["prog", "angular_momentum_magnitude"]
    main_vel = my_tree['velocity'].to('unitary/s').value


    for node in self.a.get_nodes_from_selection(sphere):
        node_redshifts = node["prog", "redshift"]
        node_prog_positions = np.array(node["prog", "position"].to("unitary"))
        node_radii = node["prog", "virial_radius"].to("unitary").value
        node_snaps = node["prog", "Snap_idx"]
        node_momenta = node["prog", "angular_momentum_magnitude"].value
        node_mass = node["mass"].value
        node_vel = node['velocity'].to('unitary/s').value
        node_posn = np.array(node['position'].to("unitary"))
        vec = initial_position - node_posn

        abs_dist = np.linalg.norm(np.subtract(np.mod(np.add(vec, np.multiply(1,
    ↪  1/2)), 1),
                                                np.multiply(1, 1/2)))

        distances, secondary_momenta, secondary_x, secondary_y, secondary_z =
    ↪ [], [], [], [], []           # can probably be simplified
        main_radii, main_momenta, radii, redshifts, positions, snaps = [], [],
    ↪ [], [], [], []
        primary_x, primary_y, primary_z = [], [], []


        # energy calculation
        G =  self.ds.quan(G_value, 'm**3 * kg ** (-1) * s ** (-2)').to('unitary
    ↪ **3 * Msun ** (-1) * s ** (-2)')
        mu = (primary_mass * node_mass) / (node_mass + primary_mass)
        GPE = G.value * primary_mass * node_mass / (abs_dist)
        KE = 1/2 * mu * ((main_vel - node_vel).dot(main_vel - node_vel))
```

```
        E = KE - GPE

        if (KE < GPE and node['mass'].value/primary_mass >= 0.01 and node['mass
    ↪ '].value <= primary_mass):

            # find distance between progenitors of node and primary halo for as
    ↪  long as the node existed
            for i, (main_pos, node_pos) in enumerate(zip(prog_positions,
    ↪ node_prog_positions)):
                # let us account for periodic boundary conditions:
                vec = main_pos - node_pos
                box_size = 1
                abs_dist = np.linalg.norm(np.subtract(np.mod(np.add(vec, np.
    ↪ multiply(box_size, box_size/2)), box_size),
                                                      np.multiply(box_size,
    ↪ box_size/2)))

                distances.append(abs_dist)
                positions.append(node_pos)
                snaps.append(node_snaps[i])
                radii.append(node_radii[i])
                secondary_momenta.append(node_momenta[i])
                main_momenta.append(my_momenta[i])

                main_radii.append(primary_radii[i])
                redshifts.append(node_redshifts[i])

                # for pandas dataframe purposes:

                secondary_x.append(node_pos[0])
                secondary_y.append(node_pos[1])
                secondary_z.append(node_pos[2])

                primary_x.append(main_pos[0])
                primary_y.append(main_pos[1])
                primary_z.append(main_pos[2])

            if distances[0] > np.min(distances) and np.min(distances) <=
    ↪ initial_radius+radii[0]:
                print('Suspected flyby found!')

                df = pd.DataFrame({"x": primary_x, "y": primary_y, "z":
    ↪ primary_z, "x'": secondary_x, "y'": secondary_y, "z'": secondary_z,
                                   "Distances" : distances, "Snapshot" : snaps, "
    ↪ Flyby Radius": radii, "Primary Radius": main_radii,
                                   "Redshift" : redshifts, "Primary AngMom":
    ↪ main_momenta, "Secondary AngMom": secondary_momenta})

                name = f'{node}_{self.quantity}={self.how_large}_energy_dist.
    ↪ csv'
                print(f"Saving as {name}")

                df.to_csv(name, index=False)
                    # break
```