

# AstroBlend: An Astrophysical Visualization Package for Blender

J. P. Naiman

*Harvard-Smithsonian Center for Astrophysics, The Institute for Theory and Computation,  
60 Garden Street, Cambridge, MA 02138, USA*

---

## Abstract

The rapid growth in scale and complexity of both computational and observational astrophysics over the past decade necessitates efficient and intuitive methods for examining and visualizing large datasets. Here, I present *AstroBlend*, an open-source Python library for use within the three dimensional modeling software, *Blender*. While *Blender* has been a popular open-source software among animators and visual effects artists, in recent years it has also become a tool for visualizing astrophysical datasets. *AstroBlend* combines the three dimensional capabilities of *Blender* with the analysis tools of the widely used astrophysical toolset, *yt*, to afford both computational and observational astrophysicists the ability to simultaneously analyze their data and create informative and appealing visualizations. The introduction of this package includes a description of features, work flow, and various example visualizations. A website - [www.astroblend.com](http://www.astroblend.com) - has been developed which includes tutorials, and a gallery of example images and movies, along with links to downloadable data, three dimensional artistic models, and various other resources.

*Keywords:* miscellaneous, methods: numerical

---

## 1. Introduction

With the growth of multi-dimensional astrophysical datasets from simulations and observations, the need for greater interaction with such complex systems has also increased [1, 2, 3, 4, 5]. Not only do larger simulations and observation programs necessitate efficient ways to investigate their data [2, 6, 3], but comparison between the outputs of multiple astrophysical codes and observational surveys requires a cross platform application capable of processing numerous data formats [2, 7].

The need for multidimensional data analysis is twofold. Collapsing multidimensional data into two dimensions for the purposes of a plot in a paper generally necessitates a loss of information. This loss can seek to illuminate the salient physical principles within the data, however this is often not the case [4]. In addition, the need to efficiently interact with multidimensional data will only

grow with new and more complex astrophysical datasets as automatic analysis packages are not always reliable [8, 9].

In this new large dataset paradigm, development of both efficient means of visualization along side quantitative analysis is required - both methods of analyzing the data are complementary [10]. Recent innovations in the computer graphics and gaming industry can be exploited for this multi-pronged approach to data investigation [5, 11]. In addition to increasing the rate of information exchange within scientific collaborations, as technological advancements in graphics and gaming progress, innovative methods for discussing scientific concepts with the general public can be developed [12, 13, 14, 15, 16].

In the past several years there has been a growing interest in the use of the three dimensional modeling and special effects software *Blender* as a scientific visualization and analysis tool. *Blender* is currently being utilized in fields as diverse as biology in order to study proteins [17] and to visualize planetary surface topology with data from Mars, Earth and the Moon [18]. In astronomy, the work of [19, 11] serves as an introduction to many of *Blender*'s features for the field, and focuses mainly on methods to generate physically accurate artistic models of astrophysical objects. While [19, 11] provides initial methods for viewing observational and simulation data, the *FRELLED*<sup>1</sup> package provides a detailed interface for uploading and interacting with multidimensional observational and simulated datasets [5]. *FRELLED* is styled for the extraction of HI sources, which makes use of *Blender*'s volume rendering capabilities and includes methods to record source information, decreasing extraction times by more than an order of magnitude. While initially developed as predominantly a FITS viewer, it provides methods to upload and interact with data from Adaptive Mesh Refinement (AMR) and Smooth Particle Hydrodynamics (SPH) simulations.

In this paper, I introduce *AstroBlend*, a Python package which provides complimentary capabilities to these other methods. Not only does *AstroBlend* allow users to compare data from multiple sources (simulations and observations), it is designed to lower the barrier to entry for interaction and visualization of astrophysical datasets in a three dimensional context. In this first release, there is no support for the volumetric interaction with data as provided with the *FRELLED* package, however it is being developed for subsequent releases. While other visualization tools have been developed for the analysis of large datasets (ParaView [20], VisIt [21]), they lack easy integration between datasets and physically motivated artistic models. As a predominately three dimensional modeling software, *Blender* provides access to a wealth of artistic representations of scientific concepts. In addition, by building a *yt* [2] backend into *Blender*, we allow for both analysis and visualization to be produced in tandem with the support of the full set of tools available in *yt*. The focus of this paper will be on the visualization capabilities of *AstroBlend* in *Blender*.

The organization of this paper is as follows: after a brief review of the

---

<sup>1</sup><http://www.rhysy.net/frelled.html>

*Blender* software and GUI in section 2, an example workflow including the import of astrophysical data and image rendering is presented in section 3. The different methods for importing datasets are discussed in section 4. Options for manipulation of lighting and camera positioning are summarized in sections 5 and 6, respectively. Several three dimensional annotations are discussed in section 7. Preliminary camera motion options for the creation of movies are shown in section 8. The paper concludes with several example renders and interactive three dimensional models in section 9 and a discussion of future plans in section 10.

## 2. Blender

*Blender* is an open source three dimensional modeling and animation software used by millions<sup>2</sup> of visual effects artists, animators and an increasing number of astrophysicists [22, 19, 5, 11]. The software is cross platform - packages for Linux, Windows and Mac computers are provided on the download website<sup>3</sup>. In addition to the rendering of images from three dimensional models, *Blender* provides a game engine for interactive graphics along with support for animating two and three dimensional characters. Many other tools are present in *Blender* including methods for rendering complex surfaces and volumes (with emission, transmission, absorption, reflective and texture components), compositing images, stereoscopic support for three dimensional graphics, and the ability to export models and lighting in a variety of formats for importing into other 3D modeling and gaming software like the Unity<sup>4</sup> game engine (Wavefront (OBJ) formatting) or 3D printing software (STereoLithography (STL) formatting). In addition, *Blender* includes some prescriptions for modeling physical phenomena, including fluids using the Lattice Boltzmann Method [23], smoke as particles based on [24], and rigid body mechanics using Newtonian physics. Here, we will focus predominately on the rendering of images, specifically from three dimensional astrophysical models and datasets.

Rendering an image in *Blender* consists of supplying a rendering engine with defined light sources to calculate light transmission, absorption, reflection, and shadows generated by objects placed in *Blender*'s three dimensional space. An example of an object and a light source placed in the three dimensional space is shown in Figure 1. Here, *Blender*'s highly customizable interface has been modified to include a view of the interactive three dimensional space in the 3D View panel, a Python Console, and a UV Image Editor panel. With this setup, when the **camera** button in the Properties Panel is selected (highlighted with a green circle in Figure 1), followed by clicking the **Render** button (highlighted with the blue circle in Figure 1), *Blender* uses its Internal Renderer to calculate the path of light from the lamp in the 3D View panel (shown as two concentric

---

<sup>2</sup><http://www.blender.org/about/website/statistics/>

<sup>3</sup><http://www.blender.org/download/>

<sup>4</sup>Unity download and information can be found here: <https://unity3d.com/>

black circles) as it bounces off the cube and into the camera (shown as the pyramid with a black arrow on top in the 3D View panel). The resulting image is shown in the UV Image Editor panel of Figure 1. One can interact with the objects in the 3D View panel directly - moving the cube, lamp, or camera with the mouse and re-rendering will generate a different image in the UV Image Editor. Rendering an image internally without generating an external file can be done through the default Internal Renderer as shown in Figure 1, or through the newly developed Cycles Renderer<sup>5</sup>, which creates more photo realistic images than the default internal renderer and allows for both GPU and CPU rendering. Both the default Internal and Cycles render engines can provide realtime, interactive preview renderings of objects in the 3D View. In addition to providing render previews either in the UV Image Editor, or in real time interactively, the two internal renders can generate images to be saved as external images in a variety of formats (png, jpg, etc). While the example shown here uses lamps to light the scene which can often enhance visualizations, more physically realistic lighting from the data itself can be utilized to illuminate the scene as discussed in section 3.

One can also pass data from *Blender* to one of the many external renderers such as Mitsuba<sup>6</sup>, LuxCore<sup>7</sup>, and Aqsis<sup>8</sup>, among others. Each of these external renders can produce images of higher quality than *Blender*'s default internal renderer, with trade-offs in user interface, speed, and accuracy<sup>9</sup>. The images shown in this paper are produced using only *Blender*'s Internal Renderer, however Cycles rendering will be supported in a future release of the *AstroBlend* library.

*AstroBlend* simplifies the importing process of astrophysical datasets and generating of visualization and analysis plots in *Blender*. The library makes heavy use of *Blender*'s Python API<sup>10</sup> to generate scripts which can either be called externally or pasted into the Python Console shown in Figure 1. Note that while *Blender* is a powerful and widely used tool, it isn't strictly a tool used by astronomers - *Blender* is generally used for low resolution models (low polygon-count surfaces) with physical accuracy modeled by "painting on" textures, while astronomers produce high resolution simulations (resulting in high polygon-count surfaces and particle clouds). In addition, an animator typically generates one three dimensional model which is then manipulated as the scene is animated, while astronomers instead represent the animation of a scene with stepping through collections of time series simulation files. This distinction manifests

---

<sup>5</sup>Comparison between the Internal and Cycles Renderers can be found here:  
[https://c\(cookie\).com/image/blender-internal-vs-cycles/](https://c(cookie).com/image/blender-internal-vs-cycles/)

<sup>6</sup><https://www.mitsuba-renderer.org/>

<sup>7</sup><http://www.luxrender.net/wiki/LuxCore>

<sup>8</sup><http://www.aqsis.org/>

<sup>9</sup>For a comparison of three of the most popular external renderers see  
<https://bartoszstyperek.wordpress.com/2015/02/14/luxcore-vs-mitsuba-vs-cycles-round-3/>

<sup>10</sup>[https://www.blender.org/api/blender\\_python\\_api\\_2\\_76\\_2/](https://www.blender.org/api/blender_python_api_2_76_2/)

itself in the way we manipulate our data in *Blender* - particularly in how objects are loaded and deleted. These subtleties are outlined below in section 3. Finally, there are many other features in *Blender* such as keyframing (an animation technique), and UV unwrapping (a modeling technique) that may be useful in the future, but will not be discussed in the context of this paper (see: [11] for details on these processes).

### 3. Example Workflow

The pathway from an astrophysical dataset to a rendered image and/or three dimensional model revolves around first loading the data in a format that is understood by *Blender*, defining the lighting source(s), placing the objects in the three dimensional space, and then exporting a rendered image(s) or a three dimensional model. The workflow of a *Blender* session is highly variable between users and projects. Alternate workflows to that described here are demonstrated in the extensive introductions of [19, 11]. For the purposes of this paper, the workflow will typically be composed of the six following steps, an example of which can be found in Figure 2.

- **Load 3D Models:** The first task is to load data files from one or multiple simulations. One can either load pre-generated surfaces or particle clouds, or rely on the integrated *yt* [2] backend to generate these objects. The Load command in the *AstroBlend* library deciphers which type of file to import and is discussed more in section 4. In the example code of Figure 2 the file that is loaded is a set of OBJ-formatted surfaces from AMR data of an isolated galaxy simulation [25] pre-generated with *yt*<sup>11</sup>. OBJ, or Wavefront, files are a data format used to represent objects in space as lists of the vertex locations, colors, and lighting options of individual polygons which, when combined, define a three dimensional surface. The surfaces stored in the OBJ file and loaded into *Blender* in Figure 2 are two isodensity surfaces, colored by temperature. In practice, one can load any series of surfaces in this manner including a time series.
- **Pick Lighting Source:** Currently, the supported sources of light are either *Blender*'s lamps which provide external sources of light, or the loaded objects themselves. While point clouds which are used to represent particles in SPH data are generated with some emissivity, one can set the emissivity of a surface based on combinations of physical parameters used to extract the isosurface and its color map. In Figure 2, the lighting is set to be solely due to the emissivity of the surface - in this example, it is a combination of the density (the isosurface value) and the temperature (isosurface color),  $\epsilon \propto \rho^2 T^{1/2}$ . Note that for *Blender* to be able to import the emissivity of this externally generated OBJ file, slight modifications

---

<sup>11</sup>See <http://blog.yt-project.org/post/objexporter/> for more details on exporting OBJ-formatted surfaces

to *Blender*'s OBJ importer are necessary. The few lines of code which need to be changed are discussed more fully on the AstroBlend website's "Getting Started" page<sup>12</sup>.

- **Arrange the Scene:** Once objects are loaded and lighting sources are determined, one must place the camera, simulation utilizingobject and other three dimensional models in the scene. In Figure 2, the three dimensional surfaces are automatically placed at the origin, and the camera, once initialized, is placed at 6 Blender Units (BU) along the negative x-axis, (-6, 0, 0), pointing at the center of our simulation object, (0,0,0). More information about the camera setup is discussed in section 6. The lamp and cube from Figure 1 are also deleted with the `science.delete_object` command. One can move objects in the scene with commands in the Python console, or with a combination of the mouse and keyboard. Here, direct interaction with the Python console is assumed and a full discussion of keyboard commands is relegated to the *AstroBlend* website<sup>12</sup>.
- **Render an Image:** A quick render to *Blender*'s Image Editor as depicted in Figure 1 can be produced by pressing the `camera` and `Render` buttons, or render to a PNG image as shown at the end of the code segment in Figure 2. To render to a file, *AstroBlend* requires a directory and a base naming structure for the image files and produces each rendered image tagged with a number. Each time an image is rendered with this render object, the number tag of output image is increased by 1. For example, the first render from the code in Figure 2 would be "galSurfs\_0000.png", and if `render.render()` is called again, the next image would be stored as "galSurfs\_0001.png". This naming structure provides for easy combining of image files into movies externally from *Blender*.
- **Export 3D Objects:** Before moving onto a new model, one can also export isosurfaces or save entire blend files to be later uploaded to various websites dedicated to the distribution of three dimensional models (see section 9.2 for some examples).
- **Delete the Model:** When one is done with a model, it is important to fully remove the object from the 3D viewer and memory. This is accomplished by using the `science.delete_object(name_or_object)` where `name_or_object` is either the name of the object, or the object variable itself. As *Blender* is not designed for the rapid generation and removal of large polygon-count meshes, `science.delete_object` scripts delete the object from both the current 3D scene and memory as well ("unlinking" the object from the scene in standard *Blender* nomenclature). While this command is not explicitly shown for the isosurfaces in Figure 2, example usages of `science.delete_object` are shown in the removal of the cube

---

<sup>12</sup><http://www.astroblend.com/getstarted.html>

and lamp objects in the arrangement of the scene. This method should be utilized when scripting animations of large datasets.

It is worth noting that many of these operations can be done utilizing the GUI and mouse in addition to using *AstroBlend* in the Python Console<sup>13</sup>.

#### 4. Data Import

In *AstroBlend*, all data files can be loaded with the same load command, `science.Load`. This command uses the file extension, or in the case of direct access to simulation files with *yt*, the *yt* simulation type flag, to determine how to load and display the model data. Each source of model data has different parameters which can be used to augment what surface (for AMR data) or point cloud (SPH data) is displayed in *Blender*'s 3D viewer.

##### 4.1. Direct access with *yt*

Currently, most of the widely used AMR codes supported by *yt* allow for isosurface models with *AstroBlend*, and many SPH codes supported by *yt* can be uploaded as point clouds. *yt*, the widely used open source, parallel, visualization and analysis Python package, provides a wealth of tools to view and analyze data including volume rendering, projections, halo finding, isocontour generation, two dimensional histograms and integration of variables along user supplied paths. At present *AstroBlend* mainly exploits *yt*'s ability as a data reader for both AMR and SPH codes, and its isocontour generation capabilities. In addition, *AstroBlend* makes use of *yt*'s color maps and color mapping functions to rebin data values to one of the 256 indices in a specified *yt* color map when generating surfaces or point clouds in *Blender*<sup>14</sup>.

Table 1 lists the full range of code support available in both *yt* and *AstroBlend*. For many simulation data types where *yt* does not support the generation of surfaces or point clouds, one can still load the files in *Blender*'s Python console, generate all supported *yt* analysis plots, including slice, projection and phase plots and save these files as pngs from the command line.

As code support is being updated constantly, a more up to date list of supported codes can be found on the *AstroBlend* website<sup>15</sup>. The parameters used in generating AMR surfaces and SPH point clouds determine the shape and color range of the displayed three dimensional models.

---

<sup>13</sup>Several useful tutorials about setting up a test model and render using the GUI can be found here: [https://en.wikibooks.org/wiki/Blender\\_3D:\\_Noob\\_to\\_Pro/Quickie\\_Model](https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Quickie_Model) and here: [https://en.wikibooks.org/wiki/Blender\\_3D:\\_Noob\\_to\\_Pro/Quickie\\_Render](https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Quickie_Render)

<sup>14</sup>More on *yt* color maps here: <http://yt-project.org/doc/visualizing/colormaps/index.html>

<sup>15</sup>Up to date list of *yt* supported codes is in “Current Code Support” section here: [http://www.astroblend.com/tutorials/tutorial\\_callingytdirectly.html](http://www.astroblend.com/tutorials/tutorial_callingytdirectly.html)

#### 4.1.1. Adaptive Mesh Refinement Data - Surfaces

To upload an isosurface directly from an AMR simulation output file using *yt* as a backend, only slight modifications to the `science.Load` command from that listed in the code of Figure 2 are necessary. Specifically, one needs to pass information regarding which variable will be used to generate the isosurface, what variable to color the isosurface by, and how to make the surface emissive if the light source in the scene is not a lamp (more detail on lighting options in section 5). Figure 3 depicts several surfaces derived from an Enzo [25] simulation along with their generating code.

In addition to specifying the isocontour parameter (density) and the color scheme (temperature), the call to `Load` specifies the emissivity parameter of each face as a *yt* derived field from the combination of the isosurface generating and color parameters - in this example, proportional to the bremsstrahlung emission of the gas ( $\epsilon \propto \rho^2 T^{1/2}$ ) in this example. Here, *yt* is used to query the data, generate the isosurface vertices and color and emissivity maps, which are then passed to *Blender* which then generates the colored meshes in the three dimensional space from the vertex and color index lists. Note that when generating isosurfaces directly with *yt* the physical parameters (scale, location, etc) of each surface can be manipulated independently, whereas in the case of the surfaces generated in *yt* externally from *Blender* of Figure 2 all surfaces are loaded as a single mesh object. However, if a Wavefront file generating program labels each surface independently in the OBJ file each surface can be uploaded as an independently named object.

#### 4.1.2. Smooth Particle Hydrodynamics Data - Point Clouds

Direct access to SPH data is also available using *yt* as a backend in *Blender*. An example calling sequence for generating a point cloud from *Tipsy* [26] simulation data along with a generated image is shown in Figure 4. To create point clouds in a memory efficient manner, *AstroBlend* generates meshes with each vertex located at the position of an SPH particle. By using a halo shader to create a point at the location of each vertex a point cloud can be rendered without the need to generate a sphere at each SPH location, a procedure which is highly memory intensive and slows render times considerably as discussed in [19]. To simulate a range of point cloud colors, *AstroBlend* creates 256 separate meshes based on bins in the parameters `color_field` and `color_map`. In Figure 4, the point clouds are specified to be colored by the gas temperature using the Rainbow colormap. Each vertex of each mesh is then rendered with the halo shader, with the size of the halo determined by the `halo_size` parameter. The further away the camera is set from the point clouds, the larger the `halo_size` parameter needs to be to render each SPH particle.

In addition, while smaller halos in dense regions can be beneficial for visualization aesthetics, they can cause the gas to appear diffuse in less dense regions. In future releases of *AstroBlend*, more freedom will be given to the user to modify the halo sizes of individual particles. The related parameter `clip_begin`, shown in the example code of Figure 4, which determines the smallest object the camera will render can be lowered to insure both foreground and background

SPH halos are adequately rendered. Both this parameter and the `halo_size` parameter can be modified to optimize the final visualization.

#### 4.2. External File Formats

Beyond the use of *yt* as a backend for querying data and generating surfaces, *AstroBlend* supports two main external file formats for reading in simulation data - OBJ files and particle text files. An example of reading in an OBJ surface pre-generated externally with *yt* was shown in Figure 2.

Additionally, when SPH simulation data is not supported for direct access with *yt*, the output simulation data files can be converted to text particle lists for import with *AstroBlend*. For this option, *AstroBlend* expects particle data in the following form: (column 1) the particle identifier (or a place holder if there is none), (columns 2-4) the x/y/z coordinates of the particle, (5) a number specifying the “type” of particle. For example, if one had formatted data from a simulation with gas particles (particle type 0) and star particles (particle type 1), the `Load` command would be as follows:

```
myobject = science.Load(txtfile, scale = scale,
                       halo_sizes = halo_sizes, particle_num=2,
                       particle_colors=colors)
```

where `txtfile` is the name of the particle text file, `scale` is a tuple giving the x/y/z rescaling of the imported particle data set, `halo_sizes` gives the halo size of each of the `particle_num` particles, and `particle_colors` is a `particle_num` sized list of RGB tuples defining each particle group’s color. A more detailed description of importing data through these external formats is relegated to the online tutorials found on [www.astroblend.com](http://www.astroblend.com).

## 5. Lighting Options

Once AMR surfaces or SPH point clouds have been loaded into *Blender*, the source of light for the render must be selected. Two options exist for illuminating scenes - the emissivity of the point cloud and surfaces themselves or external lamps. Both methods described here are meant to create photorealistic and not necessarily physically accurate images, and should be used for illustrative visualizations only.

### 5.1. Available Lamps

*Blender* provides several external lighting options to mimic spotlights, area lights, lamp light and more. In *AstroBlend*, the currently support lamp types are `POINT` and `SUN`. Both `POINTS`, which are isotropic point light sources similar to the light produced from a naked light bulb, and `SUNs`, which simulate far away point sources of light are initialized in the same way. The following code snippet initializes a `SUN` type lamp and places it at x/y/z coordinates (5,0,6) BU:

```
light = science.Lighting(lightning_type='SUN')
light.location = (5, 0, 6)
```

Other lamps not yet supported by *AstroBlend* can be added through the use of *Blender*'s GUI<sup>16</sup>.

### 5.2. Surface and Point Cloud Emission

Emissive objects in *AstroBlend* come in two forms - glowing vertices from a halo shader for SPH point clouds (as depicted in Figure 4), and emission from the faces of an AMR isosurface (as shown in Figures 2 and 3). While halo shaders are emissive<sup>17</sup> but do not cast light into the scene, an emissive isosurface can illuminate other objects in the scene. In either case, emissive lighting must be selected (`science.Lighting(lightning_type = 'EMISSION')`) in order to view the particles or surfaces in a render.

Mesh face emission is calculated with *Blender*'s implemented “Approximate Ambient Occlusion” method. In brief, this method approximates transmission and emission from different points on a mesh based on how much an emissive disk placed at each vertex would be occulted by other vertex's disks<sup>18</sup>. While the level of emissivity of the SPH point clouds is fixed, each mesh face can have different emissivities based on their physical properties, as depicted in Figures 2 and 3. In Figure 2 the emissivity of each surface is calculated externally in *yt* and read in with the other parameters of the OBJ file using the modified OBJ importer in *Blender* discussed in section 3. When using *yt* within *Blender*, the emissivity of the sources is added as a derived field using *yt*'s standard prescription. An example of this can be found in the code example of Figure 3.

## 6. Camera Options

In addition to placing lamps and selecting sources of light, to render an image one must select a camera location and pointing. *Blender* allows the user to move and rotate the camera directly in the 3D Viewer with the mouse<sup>19</sup>. *AstroBlend* allows for manipulation of the Camera object and its pointing directly through the Python Console by parenting the Camera object to an “Empty” mesh, as shown in Figure 5. To initialize the camera with *AstroBlend* as shown in Figure 5 the following single command is used:

```
cam = science.Camera()
```

---

<sup>16</sup>See [https://en.wikibooks.org/wiki/Blender\\_3D:\\_Noob\\_to\\_Pro/Understanding\\_Blender\\_Lights](https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Understanding_Blender_Lights) for a discussion of *Blender*'s many available lamps.

<sup>17</sup>For details on halo shaders in *Blender* see: [https://www.blender.org/manual/render/blender\\_render/materials/special\\_effects/halo.html](https://www.blender.org/manual/render/blender_render/materials/special_effects/halo.html)

<sup>18</sup>More information on this method and theory papers found here: <https://peach.blender.org/2008/01/approximate-ambient-occlusion/>

<sup>19</sup>Many tutorials exist about manipulation of the camera. One such example is [http://wiki.blender.org/index.php/Doc:2.4/Manual/3D\\_interaction/Navigating/Camera\\_View](http://wiki.blender.org/index.php/Doc:2.4/Manual/3D_interaction/Navigating/Camera_View)

By parenting to a non-renderable mesh, the camera is easily pointed at an object by placing the empty mesh at the location of the object as shown in Figure 5, or by simply using the command:

```
cam.location = (cam_x, cam_y, cam_z)
cam.pointing = (object_x, object_y, object_z)
```

where `(cam_x, cam_y, cam_z)` and `(object_x, object_y, object_z)` are the x/y/z coordinates of the camera and the object of interest, respectively.

## 7. Three Dimensional Annotations

Before rendering a final image consisting of loaded models, a placed camera and lights, the scene can be annotated with several simple three dimensional models. *AstroBlend* currently supports annotations with colored spheres, arrows, and text. While spheres are adirectional, pointings of both arrows and text are supported by the same method outlined in section 6 to enable camera pointing - these objects are parented to empty (non-renderable) meshes, and by moving these empty-meshes, the objects can be pointed. Figure 6 shows these three objects in *Blender*'s 3D viewer. The code in the python console depicts the addition of a green sphere without shading (a sphere without depth in the render), an arrow with shading enabled, and a simple text statement. All objects can be moved with their `.location` parameters, and the text and arrow objects can be pointed in different directions with their `.pointing` parameters.

Note in the top right Outliner panel of Figure 6 there are three objects associated with the text object. The `CenterOf` object is an invisible empty-mesh at the center of the text object. In order for the pointing empty-mesh to parent to the center of the text object, and not its leading side, an extra empty-mesh locked to the center of the text object is necessary. In practice this requires the moving of any text to be performed by selecting the `CenterOf` object, and not the text object itself. More information on this topic can be found in the sixth tutorial on the *AstroBlend* website<sup>20</sup>.

## 8. Complex Camera Motions for Movies

The rendering of single images, either in *Blender*'s UV Image Viewer (as depicted with two button presses in Figure 1) or to a file (as shown in the code section of Figure 2) have been used as examples in the discussions of camera placement, lighting and annotation in the previous sections. However, *AstroBlend* also supports the rendering of multiple images with a moving camera, which can be combined externally from *Blender* into a movie. Currently supported camera motions include Zoom, Rotate and Bezier curves.

---

<sup>20</sup>[http://www.astroblend.com/tutorials/tutorial\\_simpleUseThings.html](http://www.astroblend.com/tutorials/tutorial_simpleUseThings.html)

### 8.1. Zoom

The Zoom motion simply moves the camera along the line connecting the camera location to its pointing by a certain factor. For example, to zoom the camera from 10 BU to 5 BU along the x axis one could use the sample code:

```
render_directory = '/Users/jillnaiman/renders/'
render_name = 'zooms_'
cam.location = (10, 0, 0)
cam.pointing = (0,0,0)
movie = science.Movies(cam, render_directory, render_name,
                      render_type = 'Zoom', render_steps = 60,
                      zoom_factor = 0.5)
```

where `render_directory` is the directory where the renders will be stored and `render_name` is the base name of the renders. The number of renders in the movie, given by `render_steps = 60` indicates there will be 60 files in the render directory with names from `zooms_000.png` to `zooms_059.png`.

### 8.2. Rotation

The Rotate camera motion rotates and zooms the camera about a fixed location. The calling sequence is similar to the Zoom motion, but instead of specifying the factor by which to zoom, one specifies the rotation angles:

```
cam.location = (10, 10, 10)
cam.pointing = (3, 0, 0)
render_name = 'rotation_'
movie = science.Movies(cam, render_directory, render_name,
                      render_type = 'Rotation', render_steps = 60,
                      radius_end = 2.0, theta_end = 0.1,
                      phi_end = 90., use_current_start_angles=True,
                      use_current_radius = True)
```

The above section of code rotates the camera around the point (3,0,0) and zooms into a radius of 2.0 BU in 60 steps. Each render is stored in `render_directory` as an image from `rotation_000.png` to `rotation_059.png`. The keywords `use_current_start_angles` and `use_current_radius` indicate that this movie will start with the current camera radius and rotation angles. If these keywords are set to `False` parameters defining the beginning radius and angles, `radius_start`, `theta_start` and `phi_start`, must be specified.

### 8.3. Bezier Paths

The final available camera motion, Bezier, interpolates the camera locations and pointings from given lists using bezier curves for a smooth, complex camera motion. An example calling sequence uses three pointings and locations to constrain the camera motion along an interpolated 3D path:

```
locs = [[10,10,10],
        [10,-10,10],
        [10,-10,-10]]
```

```

pts = [[0,3,0], [0,-3,0], [0,3,0]]
movie = science.Movies(cam, render_directory, 'bezier_',
                      render_type = 'Bezier',
                      render_steps = 60,
                      bezier_locations=locs,
                      bezier_pointings=pts)

```

Using the above code, the camera will smoothly move thorough the points `locs` while the camera's pointings are interpolated from the points `pts`.

A full discussion of these three camera motions with examples and movies is relegated to the seventh tutorial on the *AstroBlend* website<sup>21</sup>.

## 9. Example Visualizations

The following section contains examples of *AstroBlend*'s visualization capabilities. As these capabilities are constantly being extended, refer to the website's gallery and tutorials for further examples. In all cases, references to timing are from the performance on a iMac quad core, 3.5 GHz Intel Core i7, 32 GB RAM desktop with a NVIDIA GeForce GTX 780M 4096 MB GPU.

### 9.1. SPH Galaxy Merger Simulation: A Simple Movie from Snapshot Files

The snapshot files from the Gadget-2 [27] galaxy merger simulations of [28, 29] are rendered in series to generate a movie of the gas and stellar properties during a typical merger. Figure 7 shows several frames of such a movie featured in [30].

Each render consists of approximately  $1.6 \times 10^6$  particles, represented by the individual halo shaders as discussed in 4.1.2. Here, yellow particles are gas, red particles denote the original bulge and disk stars, and newly formed stars are blue. The central massive black hole particles and dark matter halo particles are hidden from view in these movies to highlight the trajectories of the gas and stellar particles during the merger.

The code used to generate the movie consists of first importing each sequential Gadget-2 snapshot file as a formatted particle text file with the `Load` command and painting each particle with its specified color, determined by its particle type. After an image is rendered to a PNG file, the particle cloud is deleted with `science.delete_object` before loading the next snapshot file. In this example, the camera is stationary and the motion of the galaxies comes from the simulation itself.

This method of loading, rendering, and then deleting the data from each file serves to limit the memory usage of *Blender*, as apposed to loading the full set of datafiles to memory and animating their visibility in time. An individual cycle of loading a data snapshot and rendering an image takes < 20 s. However,

---

<sup>21</sup>[http://www.astroblend.com/tutorials/tutorial\\_cameraMotions.html](http://www.astroblend.com/tutorials/tutorial_cameraMotions.html)

repeated loading and rendering of data and images results in a slowdown of each loading processes such that the resultant movie in this example movie took approximately 7 hours to generate. This bug is likely due to imperfect memory reallocation and will be addressed in a future release of *AstroBlend*.

A full, high resolution, movie of this simulation is available on the *AstroBlend* Youtube site<sup>22</sup>.

### 9.2. Side by Side Isosurfaces: Export Data to A Variety of 3D Websites

In addition to producing movies from simulation snapshot files, several three dimensional data products can be formatted in *Blender* for export to a variety of new model sharing websites. Figure 8 shows a rendered image of one such data product - three isosurfaces of decreasing density, colored by temperature. This figure is a modified version of Figure 3 in Soares-Furtado et al. (in prep) which depicts the gas structure created by the evolved stellar population in the present day globular cluster M15, representing  $\approx 10^8$  cells of a Cartesian geometry FLASH [31] simulation.

While Figure 8 illustrates an essential figure in a stellar winds paper (Soares-Furtado et al., in prep), this particular visualization also exemplifies how to make visualizations accessible across various forms of media. As an example, these isosurfaces and annotations have been published to the Sketchfab website allowing for realtime interaction with the model in three dimensions<sup>23</sup>. Sketchfab allows for the publication and distribution of interactive models through embedded interfaces on a website or blog and downloadable OBJ files for other users. To share an isosurface with collaborators or the general public, one can directly upload *Blender* files, or export OBJ files using the `science.scienceutils.export_obj` command. While the loading of the data in this example is relatively fast (1-30 seconds for each isosurface), the exporting as an OBJ file took longer than rendering (10 seconds to render, 1 minute to export). More details about this example can be found in the fourth tutorial on [www.astroblend.com](http://www.astroblend.com)<sup>24</sup>.

The largest of the three isosurfaces depicted in Figure 8 has also been made available as a 3D-printable file on Thingiverse<sup>25</sup>, a repository for freely available three dimensional printing model files. OBJ files exported from *Blender* can be uploaded into many open source 3D printing software packages, which can then be configured to print on specific 3D printers [13, 14, 16]. In Figure 9, the original isosurface is shown alongside a photograph of its 3D-printed version. Uploading isosurfaces to 3D-printing distribution sites allows the scientist to print their models themselves, or make them available to the public for artistic

---

<sup>22</sup><https://youtu.be/8Snhd4B9SEc>

<sup>23</sup>Main Sketchfab website: <https://sketchfab.com/>, specific winds model: <https://skfb.ly/IwyK>

<sup>24</sup>More information about direct interaction with data through *yt* can be found on [http://www.astroblend.com/tutorials/tutorial\\_callingytdirectly.html](http://www.astroblend.com/tutorials/tutorial_callingytdirectly.html)

<sup>25</sup>Main Thingiverse website: <https://www.thingiverse.com/>, specific winds model: <http://www.thingiverse.com/thing:1145716>

or educational purposes, as exemplified by the use of such surfaces to make “visualizations” for the blind<sup>26</sup>.

### 9.3. Including Artistic Models in your Visualization

As *Blender* is first and foremost a 3D modeling and graphics software package, this last example visualization touches briefly on *AstroBlend*'s capabilities to combine three dimensional artistic models with astrophysical data and simulation snapshot files.

Figure 10 shows how one can combine an artistic three dimensional model with scientific data. Here, the user can download a pre-made model of a physical object as a *Blender* file and upload text or simulation data using the Python API and *AstroBlend*. In the right panel of Figure 10, the positions and names of several Milky Way dwarf galaxies are shown with an artistic model of the Milky Way, to scale. This model Milky Way is part of a downloadable Youtube tutorial on making composite objects with particles and vortices in *Blender*<sup>27</sup>. A modified version of this model can be viewed interactively on Sketchfab as well<sup>28</sup>. The left panel of Figure 10 shows the model galaxy and the bowshock around a simulated dwarf galaxy as it is ram pressured stripped during its incorporation in the Milky Way's halo. Here, the artistic galaxy is used as a point of reference for the size of the dwarf galaxy and its bowshock. Both of these images took less than 5 seconds to load and render.

By using this three dimensional artistic model of a galaxy, one can quickly understand the scale of observed substructures in our Galaxy or place simulation data done in a small scale box into the larger astrophysical context of the Milky Way.

## 10. Summary and Future Plans

As simulated and observed datasets increase in size, scientists need improved tools to analyze and visualize multidimensional datasets. In this work I introduce *AstroBlend*, a Python library for use in the 3D animation software *Blender*. *AstroBlend* relies on the isosurface mesh creation, point cloud generate, complex camera motions, lighting and texturing available in *Blender* to visualize both AMR and SPH simulation data. The use of *yt* as a backend in *AstroBlend* to generate surface and particle visualizations from direct interaction with simulation data files is discussed. In the final section of this paper, several examples of the creation of movies, three dimensional data projects, and the integration of 3D artistic models with astrophysical datasets are discussed. These examples serve as a basis with which the reader can begin to explore options for generating their own interactive visualizations for papers and press releases.

---

<sup>26</sup>Downloadable 3D printable models for the blind: <http://3dprint.com/72633/think3d-devnar-for-the-blind/>

<sup>27</sup>Model from the YouTube tutorial (<https://youtu.be/8aVOQw-BIf0>) of Thomas Piemon-teze, tomwalks.deviantart.com

<sup>28</sup><https://skfb.ly/IqAF>

Future versions of *AstroBlend* will permit the user to do on-the-fly data analysis using *yt* and make further use of *Blender*'s volume rendering capabilities and external rendering engines. Currently the development version supports a GUI for *AstroBlend* features and the display of *yt* analysis plots within *Blender* and some preliminary volume rendering. Once these updates have been pushed to the stable version of *AstroBlend*, further future updates will include the Cycles rendering engine and Cycles point density support for SPH simulations.

The reader is encouraged to visit the AstroBlend website [www.astroblend.com](http://www.astroblend.com) for further information, resources, and tutorials.

The author would like to acknowledge illuminating discussions with Matthew Turk, Morgan MacLeod, Melinda Soares-Furtado and Enrico Ramirez-Ruiz. In addition, the thorough comments of one anonymous reviewer and the extensive input of another, Dr. Rhys Taylor were invaluable. This work is supported by an NSF grant AST-1402480.

Table 1: *AstroBlend* Code Support for Direct Data Access with *yt*

Code	Code/Format Type	AB Surface Support	AB Point Cloud Support
FLASH	AMR	Y	NA
Enzo	AMR	Y	NA
Athena	AMR	Y	NA
Artio	AMR	N <sup>1</sup>	NA
Fits	FITS	N <sup>2</sup>	NA
GDF	Grid Data	N <sup>2</sup>	NA
MOAB	AMR	N (partial loading) <sup>2</sup>	NA
SPH Text Files	Text Formatting	NA	Y <sup>1</sup>
Tipsy	SPH	NA	Y
Gadget	SPH	NA	Y

<sup>1</sup>See section 4.2 for more details.

<sup>2</sup>One can load and make plots in *Blender* via *yt* with this simulation type.

## References

- [1] D. G. Barnes, C. J. Fluke, Incorporating interactive three-dimensional graphics in astronomy research papers, NA13 (2008) 599–605. [arXiv:0709.2734](https://arxiv.org/abs/0709.2734), doi:10.1016/j.newast.2008.03.008.
- [2] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, M. L. Norman, yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data, ApJS 192 (2011) 9. [arXiv:1011.3514](https://arxiv.org/abs/1011.3514), doi:10.1088/0067-0049/192/1/9.
- [3] D. Punzo, J. M. van der Hulst, J. B. T. M. Roerdink, T. A. Oosterloo, M. Ramatsoku, M. A. W. Verheijen, The role of 3-D interactive visualization in blind surveys of H I in galaxies, Astronomy and Computing 12 (2015) 86–99. [arXiv:1505.06976](https://arxiv.org/abs/1505.06976), doi:10.1016/j.ascom.2015.05.004.
- [4] F. P. A. Vogt, C. I. Owen, L. Verdes-Montenegro, S. Borthakur, Advanced Data Visualization in Astrophysics: the X3D Pathway, ArXiv e-prints [arXiv:1510.02796](https://arxiv.org/abs/1510.02796).
- [5] R. Taylor, FRELLED : A Realtime Volumetric Data Viewer For Astronomers, ArXiv e-prints [arXiv:1510.03589](https://arxiv.org/abs/1510.03589).
- [6] M. J. Turk, How to Scale a Code in the Human Dimension, ArXiv e-prints [arXiv:1301.7064](https://arxiv.org/abs/1301.7064).
- [7] J.-h. Kim, T. Abel, O. Agertz, G. L. Bryan, D. Ceverino, C. Christensen, C. Conroy, A. Dekel, N. Y. Gnedin, N. J. Goldbaum, J. Guedes, O. Hahn,

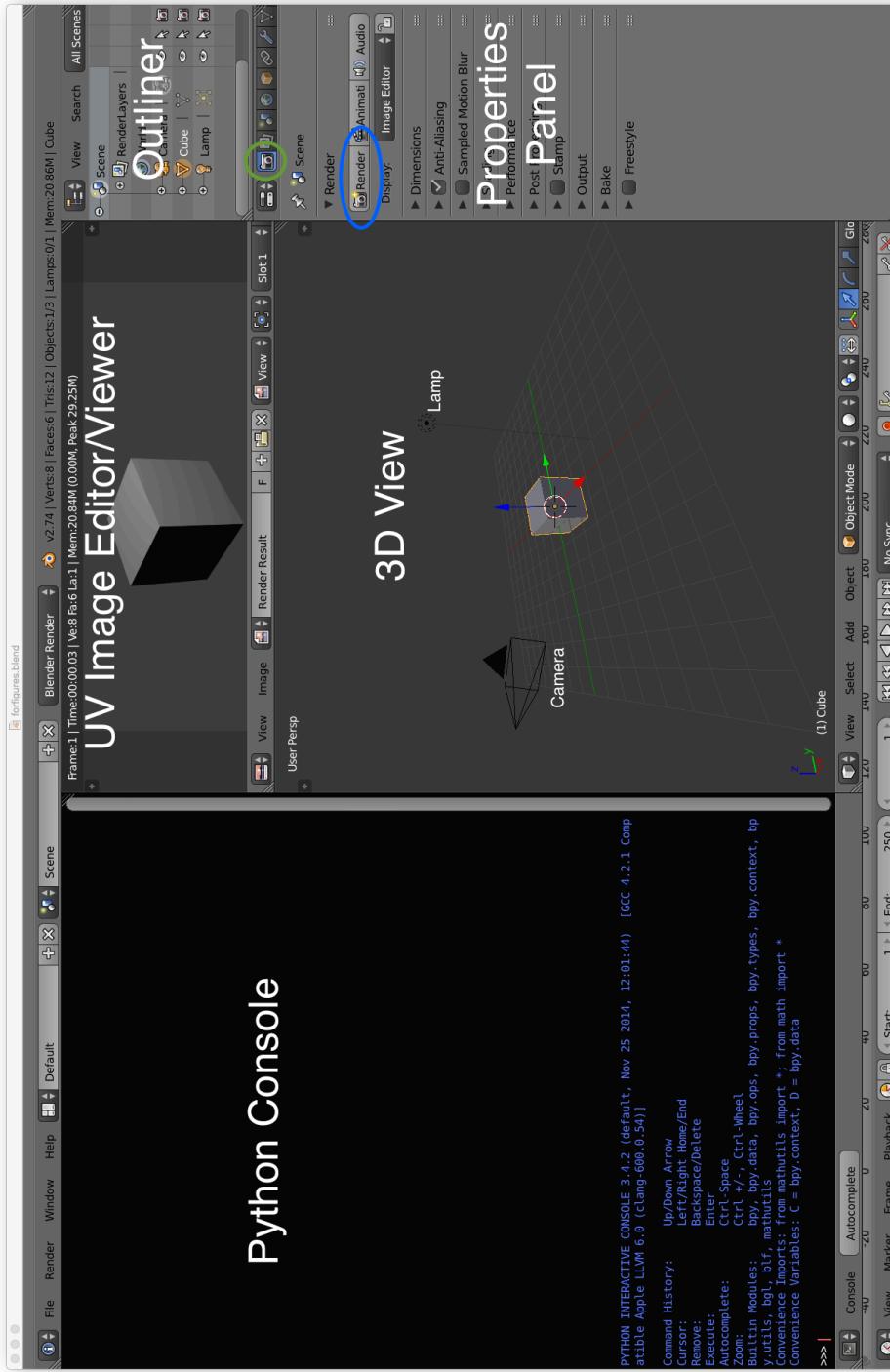


Figure 1: A typical window setup for *Blender* involves utilizing *Blender's* customizable GUI. On the left, the Python Console can be utilized to interact with the three dimensional objects and renders via the Python scripting language. The 3D View panel shows the objects in the three dimensional space - in this example, a cube, a camera and a lamp. These three objects are also listed in the Outliner panel which summarizes the objects active in the 3D View panel. The UV Image Editor (or Viewer) shows the result of a preview render created by pressing the camera button in the Properties Panel (highlighted with a green circle) followed by the Render button (Properties Panel, highlighted with a blue circle). In addition to rendering buttons the Properties Panel also contains information about objects' materials, textures and other properties.

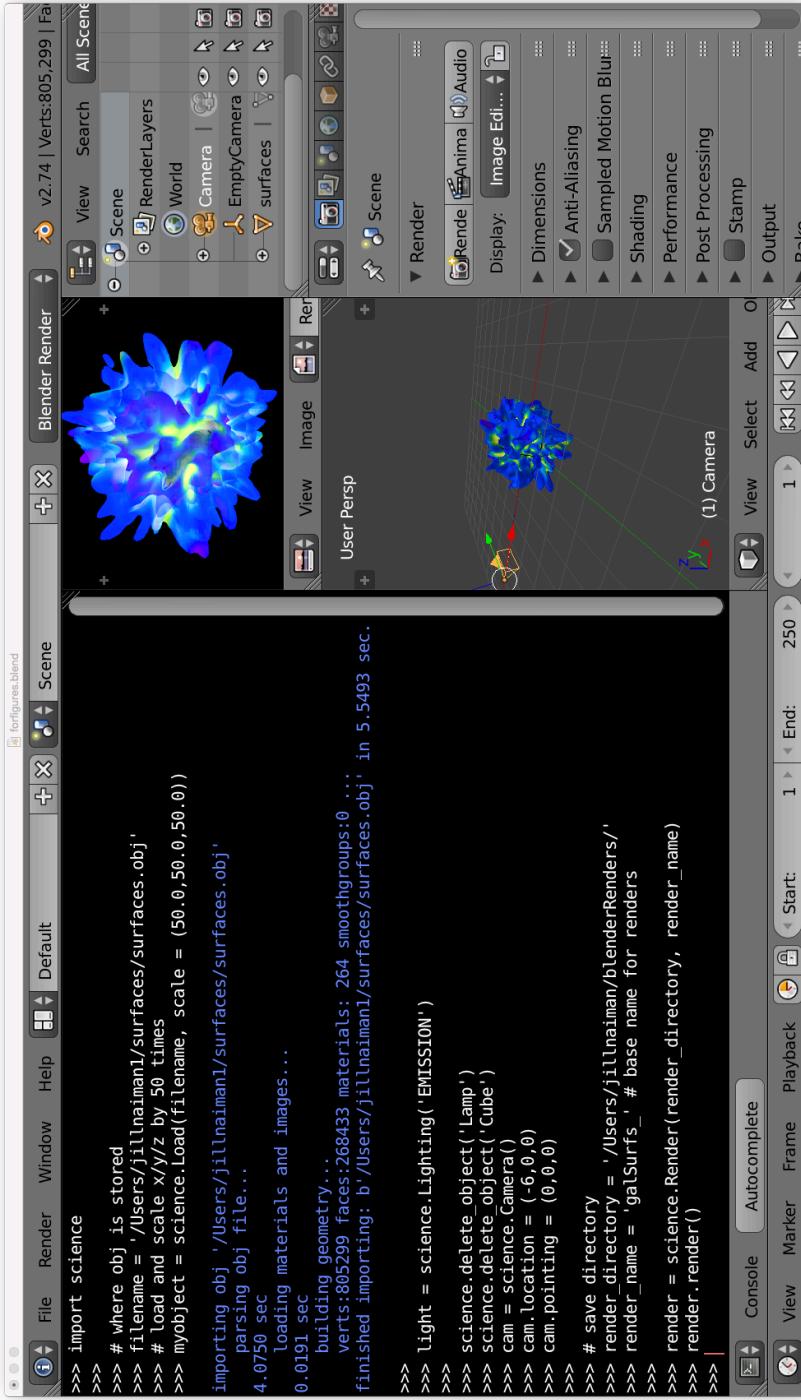


Figure 2: Generating an image from pre-generated set of isodensity surfaces with the isolated galaxy test simulation of [25]. The two surfaces are colored by temperature and each face of each surface has an emissivity proportional to its bremsstrahlung emission  $\epsilon \propto \rho^2 T^{1/2}$ . In the Python Console (left panel) the code for loading the surfaces (stored in the file surfaces.obj), setting up the lighting and camera and rendering the image is shown in white, while the blue text shows the outputs from *Blender* during this process. The 3D View panel (bottom middle) shows the resulting orientation of the surface and camera in the three dimensional space, while the UV Image Editor (top middle panel) shows the rendered image. The Outliner panel lists the objects in the 3D View panel which include the loaded isodensity surfaces (surfaces), the camera (Camera) and the empty mesh used to control the camera's pointing (EmptyCamera).

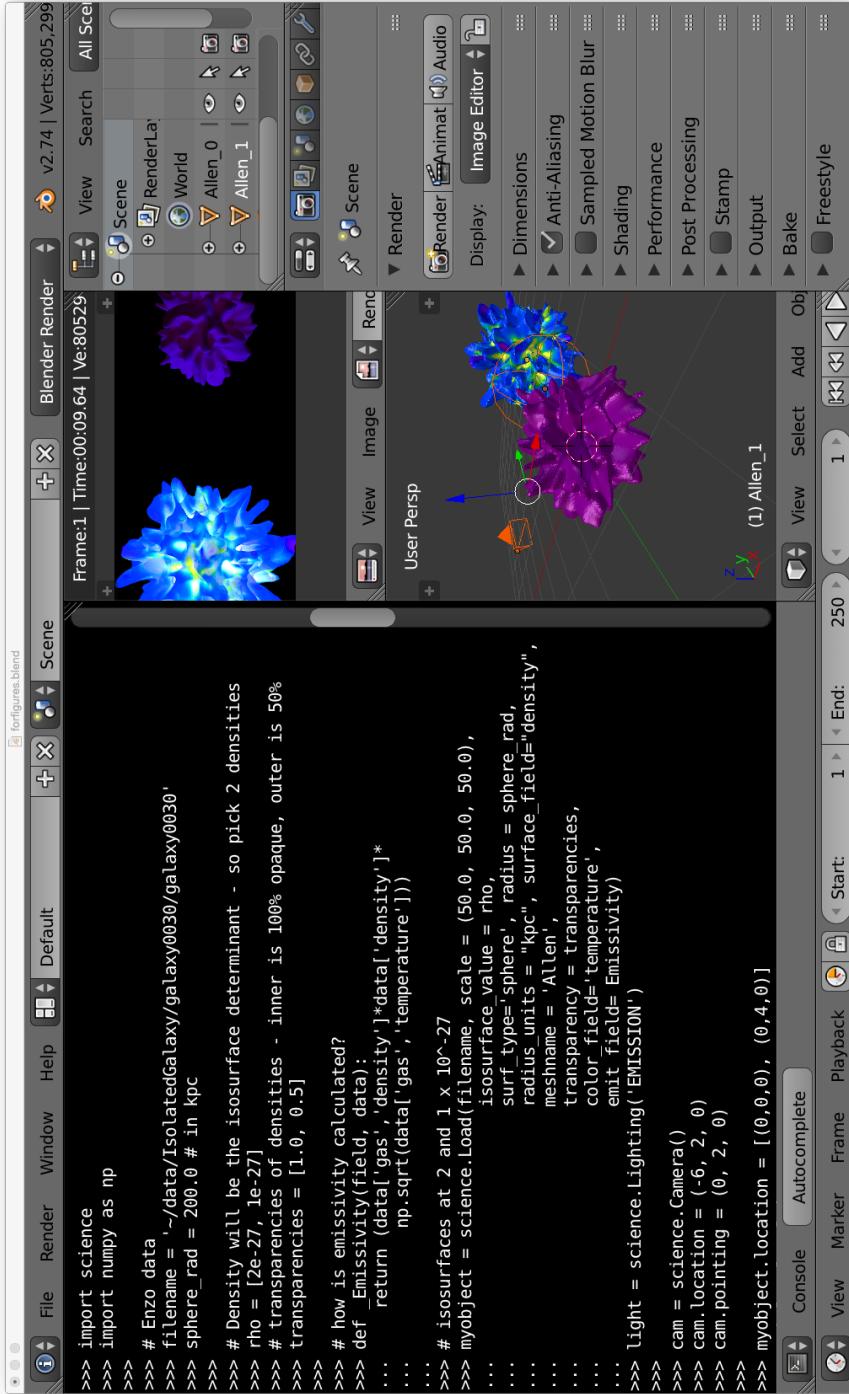
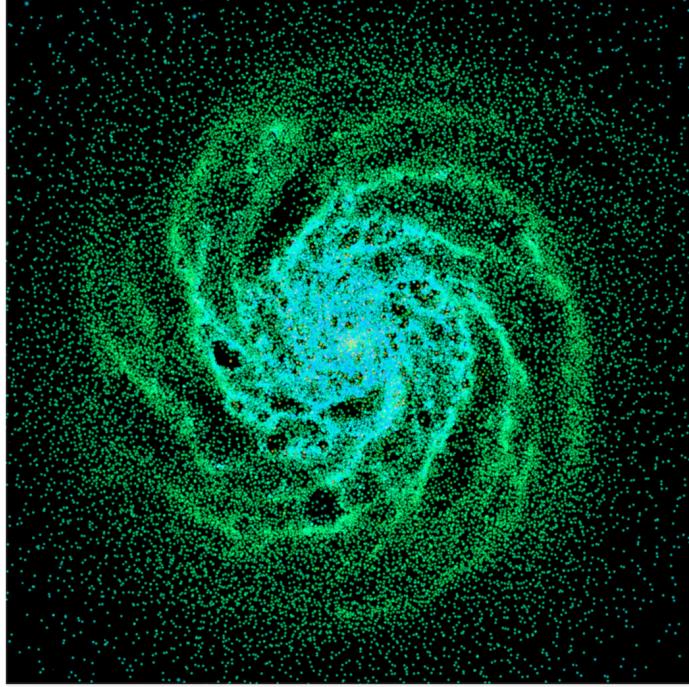


Figure 3: Direct generation of isodensity surfaces from the isolated galaxy simulation of [25] with  $yt$ . Here,  $yt[2]$  is called during the Load command to create isocontours of the variable `surface.field` (density), at values given by  $\rho$  ( $2 \times 10^{-27}$  and  $10^{-27} \text{ g/cm}^3$ ), colored by `color.field` (temperature), with the emissivity of each face of the surface calculated by the function `Emissivity` ( $\epsilon \propto \rho^2 T^{1/2}$ ). The code in the Python Console (left panel) results in an image similar to that shown in Figure 2, however, here the two isodensity surfaces are separated in the three dimensional space (bottom middle panel) for separate viewing in the final command (top middle panel) with the `myobject.location` command.



```

import science

filename = '/Users/jillnaiman/data/TipsyGalaxy/galaxy.00300'

color_field = ('Gas', 'Temperature') # color by temp
color_log = True # take log of temp for mapping colors
color_map = 'Rainbow'

# these two things play off eachother!
# larger halo size is needed when cam is far away
halo_size = 0.108 # need to play with this and cam distance
set_cam = (0,0,70)

scale = [(1.0, 1.0, 1.0)]

cam = science.Camera()
cam.location = set_cam
cam.clip_begin = 0.0001

lighting = science.Lighting('EMISSION')

myobject = science.Load(filename, scale=scale, halo_sizes = halo_size,
                       color_field = color_field, color_map = color_map,
                       color_log = color_log, n_ref=8)

```

Figure 4: The generation of a point cloud from the data of [32]. Once again, *yt* is used to facilitate direct interaction with the Tipsy [26] dataset through the `Load` command and its parameters. Here, the different colors of the point cloud halo shaders are based on the temperature of each particle. The camera is set above the galaxy along the z axis, and the clipping is lowered to be able to resolve the small halo shader particles. The parameter `n_ref`, which controls the level of oct-tree refinement, is needed to use *yt* as a backend loader of this data.

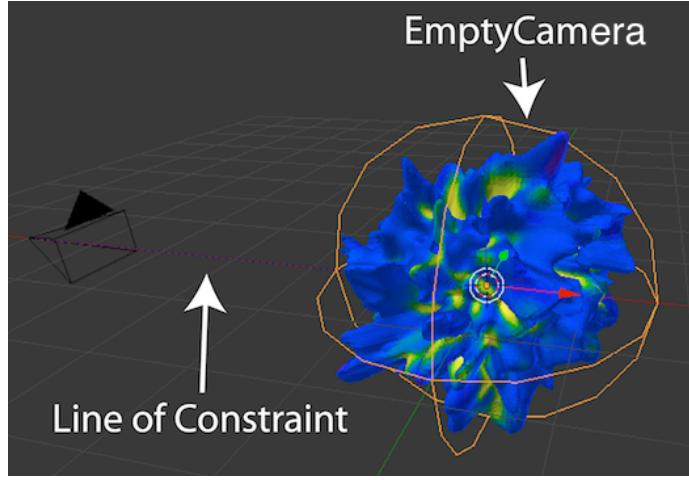


Figure 5: Depicted is a close up of *Blender*'s 3D View containing a camera and its empty pointing mesh object as well as the isodensity surfaces generated in Figures 2 and 3. The pointing of the camera is parented to an empty mesh for ease of control and is initiated with a call of `cam = science.Camera()`. In this way, one can center the camera's pointing on an object - the isodensity surfaces from Figures 2 and 3 in this example. The camera's location is modified by setting `cam.location = (x,y,z)` and its pointing with `cam.pointing = (px, py, pz)`.

- A. Hobbs, P. F. Hopkins, C. B. Hummels, F. Iannuzzi, D. Keres, A. Klypin, A. V. Kravtsov, M. R. Krumholz, M. Kuhlen, S. N. Leitner, P. Madau, L. Mayer, C. E. Moody, K. Nagamine, M. L. Norman, J. Onorbe, B. W. O'Shea, A. Pillepich, J. R. Primack, T. Quinn, J. I. Read, B. E. Robertson, M. Rocha, D. H. Rudd, S. Shen, B. D. Smith, A. S. Szalay, R. Teyssier, R. Thompson, K. Todoroki, M. J. Turk, J. W. Wadsley, J. H. Wise, A. Zolotov, t. AGORA Collaboration29, The AGORA High-resolution Galaxy Simulations Comparison Project, *ApJS* 210 (2014) 14. [arXiv:1308.2669](https://arxiv.org/abs/1308.2669), [doi:10.1088/0067-0049/210/1/14](https://doi.org/10.1088/0067-0049/210/1/14).
- [8] M. T. Whiting, DUCHAMP: a 3D source finder for spectral-line data, *MNRAS* 421 (2012) 3242–3256. [arXiv:1201.2710](https://arxiv.org/abs/1201.2710), [doi:10.1111/j.1365-2966.2012.20548.x](https://doi.org/10.1111/j.1365-2966.2012.20548.x).
  - [9] R. Taylor, J. I. Davies, R. Auld, R. F. Minchin, R. Smith, The Arecibo Galaxy Environment Survey - VI. The Virgo cluster (II), *MNRAS* 428 (2013) 459–469. [arXiv:1209.4338](https://arxiv.org/abs/1209.4338), [doi:10.1093/mnras/sts042](https://doi.org/10.1093/mnras/sts042).
  - [10] A. A. Goodman, Principles of high-dimensional data visualization in astronomy, *Astronomische Nachrichten* 333 (2012) 505. [arXiv:1205.4747](https://arxiv.org/abs/1205.4747), [doi:10.1002/asna.201211705](https://doi.org/10.1002/asna.201211705).
  - [11] B. R. Kent, 3D Scientific Visualization with Blender, Morgan & Claypool Publishers, 2015. [doi:9781627056120](https://doi.org/10.21437/9781627056120).

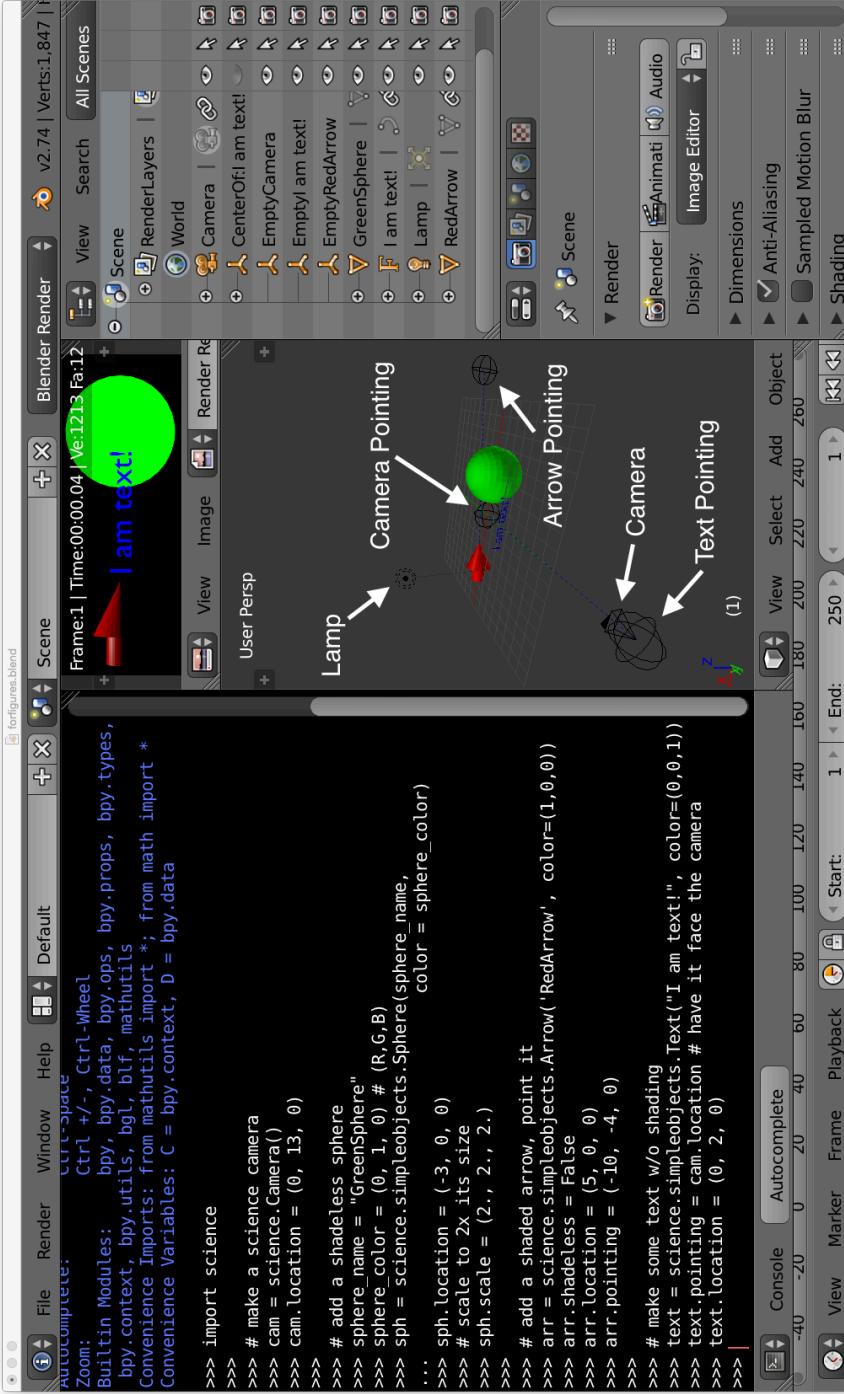


Figure 6: Simple three dimensional objects are accessible through the python interface with the code shown in the Python Console (left panel). The code snippet generates the three objects and various assorted empty meshes shown in the 3D View (bottom middle panel) and summarized in the Outliner (right upper panel). The arrow object includes both the three dimensional model, and its empty pointing mesh labeled RedArrow and EmptyRedArrow in the Outliner, respectively. The text object has three meshes associated with it - the blue text object (`I am text!` in the Outliner), the empty pointing mesh (`EmptyIf:I am text!`) and the center empty mesh (`CenterIf:I am text!`). The text empty pointing mesh controls where the text is pointing while the center empty mesh controls the position of the text - see text for further details.

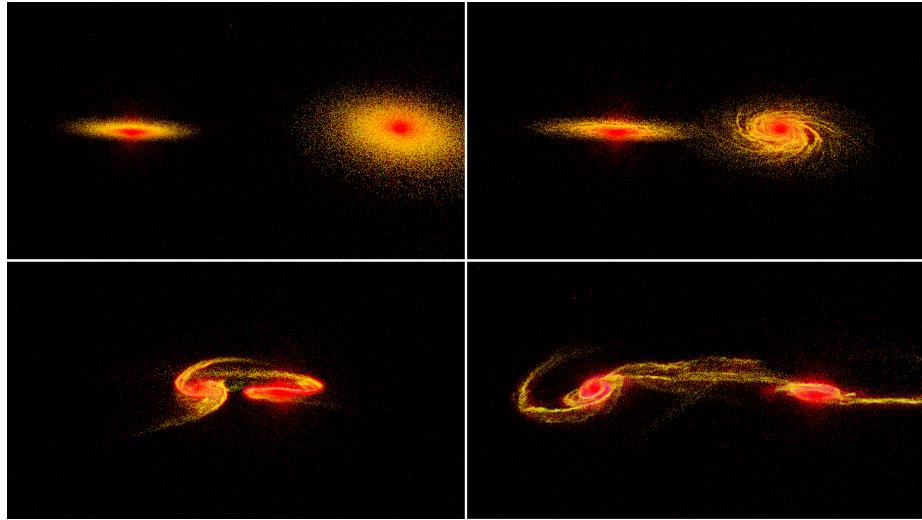


Figure 7: Four renders from a movie generated in conjunction with the paper [30] show different stages of a galaxy merger. The particle colors represent pre-merger stars (red), gas (yellow), and newly formed stars (blue) as well as black hole sinks (green, not visible). The full movie can be viewed at <https://www.youtube.com/watch?v=8Snhd4B9SEc>

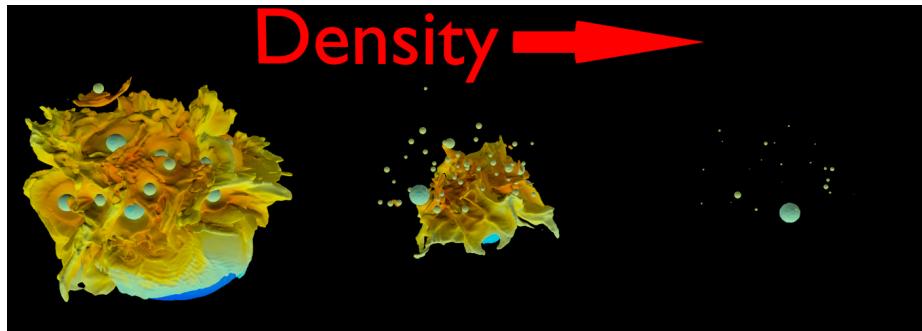


Figure 8: Several isodensity contours created in conjunction with (Soares-Furtado et. al, in prep.) show the gas structures surrounding stellar winds in a present day M15 globular cluster analog. The surfaces are colored by temperature with the cold gas residing in the areas directly surrounding the stars (blue) and the hot gas forming in the shocked colliding stellar winds (red). As the density of each isocontour increases (from left to right), we are probing the structures which closely surround each individual star. This model is available interactively here: <https://skfb.ly/IwyK>.

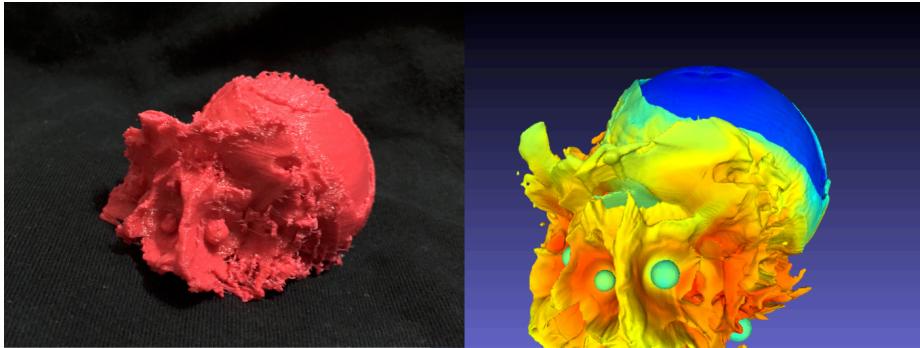


Figure 9: Isocontours generated with *AstroBlend* can be used in 3D printing. The iso-density contour of the stellar wind structures modeled in Soares-Furtado et. al, (in prep.) on the right is available as a 3D printable stl file as shown on the left, on Thingiverse: <http://www.thingiverse.com/thing:1145716>.

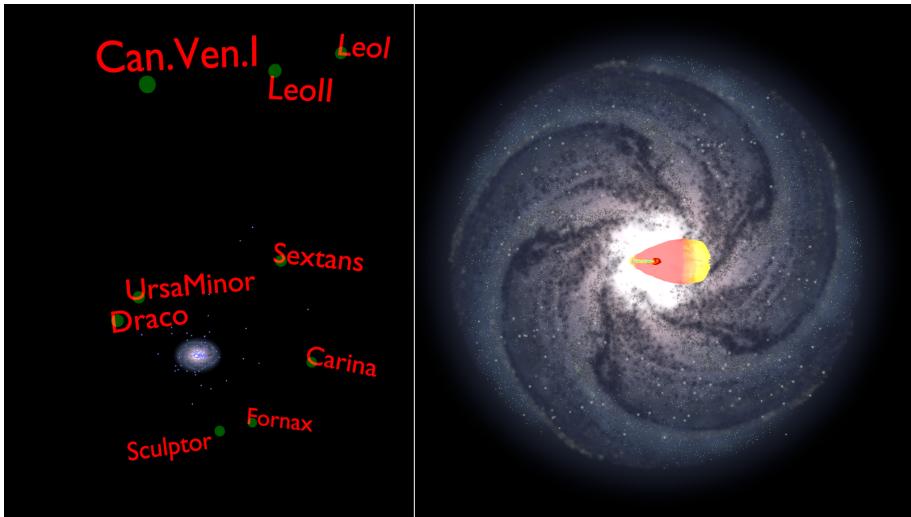


Figure 10: The use of an artistic three dimensional model alongside astrophysical data can create context for simulation and observational data. On the right a model galaxy is combined with the observed positions of Milky Way globular clusters (small blue spheres) from [33] and dwarf galaxies (labeled, larger green spheres) tabulated in [34]. A modified version of the right hand panel can be found on Sketchfab: <https://skfb.ly/IqAF>.

- [12] F. P. A. Vogt, L. J. Shingles, Augmented Reality in astrophysics, APSS 347 (2013) 47–60. [arXiv:1305.5534](https://arxiv.org/abs/1305.5534), [doi:10.1007/s10509-013-1499-x](https://doi.org/10.1007/s10509-013-1499-x).
- [13] W. Steffen, M. Teodoro, T. I. Madura, J. H. Groh, T. R. Gull, A. Mehner, M. F. Corcoran, A. Damineli, K. Hamaguchi, The three-dimensional structure of the Eta Carinae Homunculus, MNRAS 442 (2014) 3316–3328. [arXiv:1407.4096](https://arxiv.org/abs/1407.4096), [doi:10.1093/mnras/stu1088](https://doi.org/10.1093/mnras/stu1088).
- [14] F. P. A. Vogt, M. A. Dopita, L. J. Kewley, R. S. Sutherland, J. Scharwächter, H. M. Basurah, A. Ali, M. A. Amer, Galaxy Emission Line Classification Using Three-dimensional Line Ratio Diagrams, ApJ 793 (2014) 127. [arXiv:1406.5186](https://arxiv.org/abs/1406.5186), [doi:10.1088/0004-637X/793/2/127](https://doi.org/10.1088/0004-637X/793/2/127).
- [15] J. Brown, M. G. Knepley, B. F. Smith, Run-time extensibility and librariization of simulation software, ArXiv e-prints [arXiv:1407.2905](https://arxiv.org/abs/1407.2905).
- [16] T. I. Madura, W. Steffen, N. Clementel, T. R. Gull, 3D Printing Meets Astrophysics: A New Way to Visualize and Communicate Science, IAU General Assembly 22 (2015) 55804.
- [17] M. F. Zini, Y. Porozov, R. M. Andrei, T. Loni, C. Caudai, M. Zoppè, BioBlender: Fast and Efficient All Atom Morphing of Proteins Using Blender Game Engine, ArXiv e-prints [arXiv:1009.4801](https://arxiv.org/abs/1009.4801).
- [18] I. V. Florinsky, S. V. Filippov, Development of virtual morphometric globes using Blender, ArXiv e-prints [arXiv:1512.08511](https://arxiv.org/abs/1512.08511).
- [19] B. R. Kent, Visualizing Astronomical Data with Blender, PASP 125 (2013) 731–748. [arXiv:1306.3481](https://arxiv.org/abs/1306.3481), [doi:10.1086/671412](https://doi.org/10.1086/671412).
- [20] J. Ahrens, B. Geveci, C. Law, ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook [arXiv:NA](https://arxiv.org/abs/NA).
- [21] G. H. Weber, S. Ahern, E. W. Bethel, S. Borovikov, H. R. Childs, E. Deines, C. Garth, H. Hagen, B. Hamann, K. I. Joy, D. Martin, J. Meredith, D. Prabhat, D. Pugmire, O. Rubel, B. Van Straalen, K. Wu, Recent Advances in VisIt: AMR Streamlines and Query-Driven Visualization, Numerical Modeling of Space Plasma Flows: Astronum-2009 (Astronomical Society of the Pacific Conference Series, 3185E, 2010, 429:329-334) [arXiv:NA](https://arxiv.org/abs/NA).
- [22] M. A. Aragon-Calvo, S. F. Shandarin, A. Szalay, Geometry of the Cosmic Web: Minkowski Functionals from the Delaunay Tessellation, ArXiv e-prints [arXiv:1006.4178](https://arxiv.org/abs/1006.4178).
- [23] S. Chen, G. D. Doolen, Lattice Boltzmann Method for Fluid Flows, Annual Review of Fluid Mechanics 30 (1998) 329–364. [doi:10.1146/annurev.fluid.30.1.329](https://doi.org/10.1146/annurev.fluid.30.1.329).
- [24] J. J. Monaghan, Smoothed particle hydrodynamics, ARA&A 30 (1992) 543–574. [doi:10.1146/annurev.aa.30.090192.002551](https://doi.org/10.1146/annurev.aa.30.090192.002551).

- [25] B. W. O’Shea, G. Bryan, J. Bordner, M. L. Norman, T. Abel, R. Harkness, A. Krutsuk, Introducing Enzo, an AMR Cosmology Application, ArXiv Astrophysics e-prints [arXiv:astro-ph/0403044](https://arxiv.org/abs/astro-ph/0403044).
- [26] N-Body Shop, TIPSY: Code for Display and Analysis of N-body Simulations, Astrophysics Source Code Library (Nov. 2011). [arXiv:1111.015](https://arxiv.org/abs/1111.015).
- [27] V. Springel, The cosmological simulation code GADGET-2, MNRAS 364 (2005) 1105–1134. [arXiv:astro-ph/0505010](https://arxiv.org/abs/astro-ph/0505010), doi:[10.1111/j.1365-2966.2005.09655.x](https://doi.org/10.1111/j.1365-2966.2005.09655.x).
- [28] J. Debuhr, E. Quataert, C.-P. Ma, The growth of massive black holes in galaxy merger simulations with feedback by radiation pressure, MNRAS 412 (2011) 1341–1360. [arXiv:1006.3312](https://arxiv.org/abs/1006.3312), doi:[10.1111/j.1365-2966.2010.17992.x](https://doi.org/10.1111/j.1365-2966.2010.17992.x).
- [29] J. Debuhr, E. Quataert, C.-P. Ma, Galaxy-scale outflows driven by active galactic nuclei, MNRAS 420 (2012) 2221–2231. [arXiv:1107.5579](https://arxiv.org/abs/1107.5579), doi:[10.1111/j.1365-2966.2011.20187.x](https://doi.org/10.1111/j.1365-2966.2011.20187.x).
- [30] J. P. Naiman, E. Ramirez-Ruiz, J. Debuhr, C.-P. Ma, The Role of Nuclear Star Clusters in Enhancing Supermassive Black Hole Feeding Rates During Galaxy Mergers, ApJ 803 (2015) 81. [arXiv:1410.7381](https://arxiv.org/abs/1410.7381), doi:[10.1088/0004-637X/803/2/81](https://doi.org/10.1088/0004-637X/803/2/81).
- [31] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, H. Tufo, FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes, ApJS 131 (2000) 273–334. doi:[10.1086/317361](https://doi.org/10.1086/317361).
- [32] B. W. Keller, J. Wadsley, S. M. Benincasa, H. M. P. Couchman, A superbubble feedback model for galaxy simulations, MNRAS 442 (2014) 3013–3025. [arXiv:1405.2625](https://arxiv.org/abs/1405.2625), doi:[10.1093/mnras/stu1058](https://doi.org/10.1093/mnras/stu1058).
- [33] W. E. Harris, A Catalog of Parameters for Globular Clusters in the Milky Way, AJ 112 (1996) 1487. doi:[10.1086/118116](https://doi.org/10.1086/118116).
- [34] A. J. Deason, V. Belokurov, N. W. Evans, Rotation of halo populations in the Milky Way and M31, MNRAS 411 (2011) 1480–1494. [arXiv:1008.3067](https://arxiv.org/abs/1008.3067), doi:[10.1111/j.1365-2966.2010.17785.x](https://doi.org/10.1111/j.1365-2966.2010.17785.x).