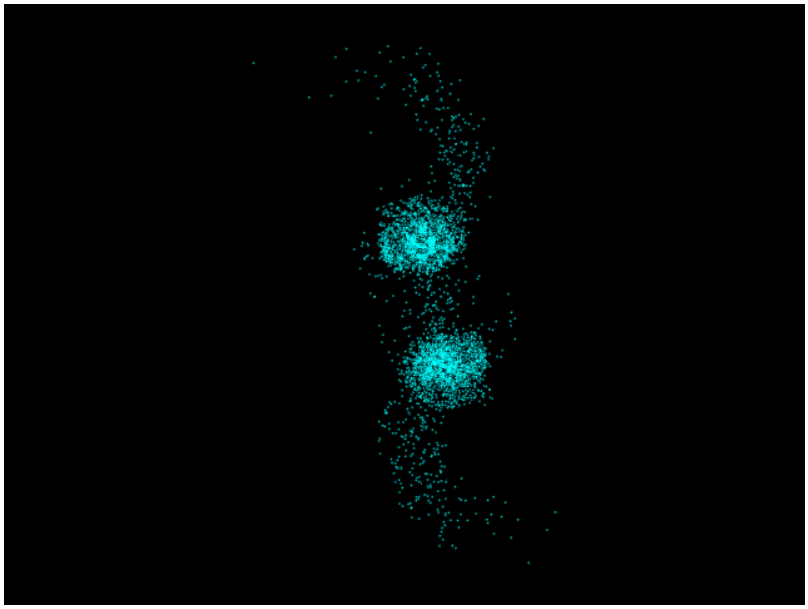

Starscream User's Manual

Jay Jay Billings

August 26, 2008

Contents

Chapter 1

About the Code

This chapter will describe the code in general and provide the user with information on compilation and installation.

1.1 Purpose

Starscream was written to assist users of gravitational N-body codes with creating the initial conditions necessary to simulate galaxy collisions. Particularly, the author hopes that graduate students and researchers alike will find the code useful, either as a tool for simulating galaxies or, at least, as a starting point for reference materials on the subject. The author remembers creating everything from random noise to "cubic" galaxies while working as a graduate student and, somewhat embarrassingly, hopes to flatten the learning curve a bit for anyone in a similar situation.

Furthermore, it may be possible to use the code in a classroom setting to teach undergraduates about galaxy collisions. Since most undergraduates lack the computational skills necessary to write anything greater than "Hello World," linking to a library that does the heavy lifting may make it easier for educators to separate the physics from the computer science.

The code is open-source, released under the GNU General Public License (GPL) v.2, and is free to use in whole or part.

1.2 Compilation and Installation

The Starscream source code is available for download from http://materials.chem.utk.edu/ jaybilly/tools/starscream as a G-zipped tarball. This package can be unzipped by typing at the prompt

tar -xzvf starscream_"release number".tar.gz

where "release number" is the eight digit number of the downloaded version. Change the working directory to the newly created "starscream/" directory.

The next step is to edit the makefile. There are several options that must be edited for the code to properly compile on the target machine:

1.) Change the value of INS_DIR to the preferred install location.
2.) Pick the compiler type by removing the comment in front of COMPILER = "GNU" or COMPILER = "Intel".
3.) Edit the compiler specific section to reflect the proper location of the headers and libraries for the GSL and FFTW libraries.
4.) Set any optimization flags after OPTIMIZE.

After setting these options, type "make" and "make install" to compile and install the library. The installation may be tested by typing "make test" to run the example starscream.c test code and the compile directory may be cleaned by typing "make clean". Users should note that "make test" displays timing information in the last few lines of output and that those lines are not an error message.

If the code compiled fine and "make test" executed without error, then the library is ready for use. Details on linking against the library are provided in the **Quick Start** section, but, if the install directory is in your path, it should be as simple as adding "-lstarsream" to your compile command .

1.2.1 Compilers and Operating Systems

Starscream should compile and work on any GNU+Linux system. The code was developed and has been thoroughly tested with 64-bit Fedora 9 using both the GCC[1] and Intel compilers, versions 4.3.0-8 and 10.1.015, respectively. The author has not tested the code in a Windows environment, but sees no reason why it would not compile with gcc under Cygwin or in a virtualized flavor like Damn Small Linux.

Users should also know that "make install" is a little different depending on the choice of compiler. If Starscream is compiled using gcc, the shared library is created using GNU Libtool. However, when compiled with the Intel compilers, the shared library is created using the standard "ld" command. The author is not sure what, if any, implications this has on the final product, since both versions provide the convenience of a single file and automatic linking to the GSL and FFTW libraries, but thought it necessary to note the difference.

1.2.2 Required Libraries

There are two external libraries required to compile and run Starscream. The first and most important of these libraries is the GNU Scientific Library (GSL), available from http://www.gnu.org/software/gsl/. Starscream uses the GSL for random numbers, numerical differentiation and integration, and some other small mathematical tasks.

The second library is the Fastest Fourier Transform in the West (FFTW), available from http://www.fftw.org/. FFTW is used to compute FFTs for finding the potential due to the disk. In principle, these functions could be replaced with any FFT library, see section 3.1 for example, but FFTW is the fastest *portable* option.

Both of these libraries should be compiled with the "-fPIC" option to create Position Independent Code (PIC). If these codes are not compiled with the "-Starscream uses FFTW version 3, not version 2, and does not use the "–enable-type-prefix" option.

The exact location of the header and libraries for these two codes must be specified in the Starscream makefile in the GSL_INC, FFTW_INC, GSL_LIB, and FFTW_LIB lines. The -I and -L options must be prepended to the include and library paths respectively.

[1]The difference between GCC and gcc is notable; GCC is the GNU Compiler Collection and gcc is the GNU C Compiler.

1.2.3 Optimization

Platform-specific optimization can greatly decrease execution time when using Starscream. Users should consider researching the proper optimization flags for their system and compiling both the required libraries and Starscream with these flags enabled. On the author's development machine, a quad-core AMD Phenom 9850, execution time of the test problem was decreased by a factor of four with compile-time optimization of the GSL and FFTW libraries using gcc. Optimization with the Intel compilers provides a similar, although slightly smaller, speed increase.

The following optimization flags worked well for Starscream, using gcc:

-O3 -msse4a -march=amdfam10

The first option is the standard optimization level tag. The second option is enables the SSE4 instruction set and the last option optimizes the code for the Barcelona family of processors. The last option is hardware-specific and is not portable. With the GSL, FFTW, and Starscream compiled using these tags, the execution time of the test case decreases to approximately six seconds from twenty four.

Further information on optimizations with gcc for AMD processors can be found at AMD Development Central, http://developer.amd.com/Pages/default.aspx.

If users are interested in submitting optimization information to this section, the author will gladly add and credit it.

1.3 Quick Start

This section will introduce users to Starscream by going over the example problem provided in the distribution. This example problem can be found in the file called starscream.c.

The first and most important element of Starscream is the "galaxy" type definition defined in the header file, starcream.h. This structure defines all of the relevant information needed to create a Springel-Hernquist type compound galaxy consisting of a stellar disk and a massive dark matter halo.[2] The starscream header file should be included at the top

[2]See ch. 2 for more on the physics of the model and ch. 3 for a discussion of the missing bulges and gas.

of any program calling starscream functions.

Users may check the starscream.h header file for more information on the galaxy type definition, but for the purposes of getting started, the details are quite inconsequential. First, users should declare some galaxy type variables,

galaxy *galaxy_1, *galaxy_2, *galaxy_3

These galaxy objects do not have to be pointers, but all functions in Starscream are call by reference. Therefore, at the very least, users who declared their galaxy objects as

galaxy galaxy_1, galaxy_2, galaxy_3

will have to pass the galaxy object as, for example, &galaxy_1 instead of galaxy_1. For possibly sadistic reasons, the author decided to use pointers to galaxies in the example.

Users could specify the physical parameters at any point before calling the function that actually creates the galaxy, but the author chose to do this specification right after declaring the galaxy variables:

parts[0] = 10000;
parts[1] = 20000;
m_d = 0.025;
j_d = m_d;
lambda = 0.050;
c = 15.0;
v200 = 1.6E7;
N = 64;
space = 4.0;

These variables are, in order, the number of particles in the stellar disk, the number of particles in the dark matter halo, the mass fraction of the disk relative to the halo, the angular momentum fraction of the disk relative to the halo, the spin parameter, the

halo concentration, the virial radius, the number of points for the potential grid in one Cartesian direction, and the spacing between two consecutive grid points. The first seven parameters are discussed extensively in chapter two and [Springel and White, 1999]. The last two parameters are purely numerical, but choosing a grid that is two small in size or has two big of a distance between grid points could create somewhat unphysical results. The Poisson solver is discussed more in chapter two, but users are warned here to be careful when picking values for N and the spacing between points.

Following the example, the next step is to allocate these pointers using something like,

```
if (!(galaxy_1=calloc(1,sizeof(galaxy)))) {
    fprintf(stderr,"Unable to allocate galaxy. Aborting.");
    return 0;
}
```

for each galaxy.

At this point, users should create the random number environment for Starscream. A global set of random number variables, T and r, are declared in the header file, but since mind reading is not the author's strong suit, users must choose their own random number generator from the GSL. The choice in the example is probably good enough for government work, but users familiar with the GSL may want to change the default random number generator or the seed value. For most users though, these five lines are sufficient:

```
seed = time(NULL);
gsl_rng_env_setup();
T = gsl_rng_default;
r = gsl_rng_alloc(T);
gsl_rng_set(r,seed);
```

After the galaxy pointers have been allocated, the physical parameters defined, and the random number generator initialized, the galaxy may be assembled in memory by calling

create_galaxy(galaxy_1,parts,m_d,j_d,lambda,c,v200,N,space,1);

The extra value of "1" at the end tells the create_galaxy() function to print physical information about the galaxy. Setting this value to 0 will kill the text output. This function allocates stores the physical parameters of this galaxy, uses them to calculate other parameters, and allocates all of the particle storage arrays including space for the positions, velocities, masses, particle ids, and the potential grid.

The next thing users must do depends on what type of system they are creating. If users are creating a new system from scratch, it is necessary to pick particle positions, calculate the potential, and set the particle velocities. This would look something like,

set_galaxy_coords(galaxy_1);
set_galaxy_potential(galaxy_1,0);
set_galaxy_velocity(galaxy_1,1);

The function set_galaxy_coords() picks the particle positions from a distribution using the Metropolis-Hastings algorithm and has no extra options. The function set_galaxy_potential() calculates the potential of a disk using a particle mesh as described in [Hockney and Eastwood, 1981]. This function has an extra option, set to zero here, that will print the values of the potential in the x,y plane to standard out if set to one. The function set_galaxy_velocity() will set the velocity of the particles to the local Kepler velocity if the second option is set to zero. However, if the second option is set to one, as in the example, the local velocity dispersions will be calculated and the velocity will be chosen from a normal distribution.

On the other hand, some users may not want to create new galaxies, but load a previous simulation from ASCII files or a Gadget2 snapshot. In the example, the commented part of the code shows how to load a Gadget2 snapshot using the load_snapshot() function. This is the same function provided by Volker Springel in the Gadget2 distribution, under the Analysis directory, in the read_snapshot.c file for reading Gadget2 snapshots. The only downside to this is that the physical parameters for this galaxy may be somewhat different than those specified and Starscream does not yet include functions to calculate

those quantities from input data.

After creating a galaxy from input parameters or reading a galaxy in from disk, users will presumably want to either put this galaxy on orbit with another galaxy or write it out for a gravitational N-body code. In the example, the author copies the first galaxy into a second galaxy, puts the two galaxies on a prograde parabolic orbit, and creates a "galaxy system" using,

copy_galaxy(galaxy_1,galaxy_2,0);
set_orbit_parabolic(galaxy_1,galaxy_2,galaxy_1->r200,3.5);
create_galaxy_system(galaxy_1,galaxy_2,galaxy_3);

The first function copies galaxy_1 into galaxy_2 completely, making galaxy_2 an exact copy of galaxy_1. The second function takes these two galaxies and edits their position and velocity information so that if a gravitational field was applied, they would begin to orbit each other on a parabolic orbit. The third option in this function is the distance of the galaxies from the center of mass and the third option is the distance of closest approach.

The last function combines the galaxies into one galaxy object. The author finds it best to think of this galaxy system as a combination two galaxies *plus* an orbit because the orbital motion is just as fundamental to a galaxy collision as the galaxies themselves. Citing, once again, the lack of significant mind reading skills, the author cannot predict all possible orbits and, therefore, only provided one orbit definition in Starscream. However, for those interested in a tidal tail show, prograde parabolic orbits are the best, [Toomre and Toomre, 1972].

The galaxy system is written to an initial conditions file compatible with Gadget2 by calling

write_gadget_ics(galaxy_3,"initial_conditions.dat");

where, again, galaxy_3 is the galaxy system and the second option is the filename. There are plenty of functions for writing the galaxy information to standard out if a user is not interested in using Gadget2 and those options are commented out in the example.

Finally, the galaxies must be deallocated to be nice to the memory,

destroy_galaxy(galaxy_1,0);
destroy_galaxy(galaxy_2,0);
destroy_galaxy_system(galaxy_3,0);

Users can compile and link the example code by executing the command

gcc -o starscream starscream.c -lstarscream

or, if the GSL and Starscream are installed in a non-standard location,

**gcc -o starscream -I"gsl_include_directory" starscream.c **
-L"starscream_install_dir" -lstarscream

Executing the sample code will print some physical information to the screen and create a file called initial_conditions.dat that can be run in Gadget2 or viewed with a program like Starsplatter of Ifrit. The galactic system in this case consists of two equal-mass and identical galaxies with ten thousand disk particles and twenty thousand halo particles each that collide on a parabolic orbit. Example videos of this collision can be found at http://www.youtube.com/jayjaybillings.

Chapter 2

About the Physics

This chapter will provide a brief description of the physics in Starscream.

2.1 Introduction

Starscream creates model galaxies using a Springel-Hernquist model with a massive dark matter halo and a stellar disk. Gas and bulge components are not considered. The halo and disk are briefly described below, but detailed information is available in several sources, [Springel et al., 2005, Springel and White, 1999, Hernquist, 1993, Hernquist, 1990].

2.2 Dark Matter Halos

The dark matter halos in Starscream have a Hernquist distribution, [Hernquist, 1990], with density,

$$\rho(r) = \frac{M_{DM}}{2\pi}\frac{a}{r}\frac{1}{(r+a)^3} \tag{2.1}$$

where M_{DM} is the total mass of the dark matter. The "a" parameter depends on the halo scale length, r_s, and halo concentration, $c = r_{200}/r_s$, of the NFW profile, [Springel et al., 2005],

$$a = r_s\sqrt{2[ln(1+c) - c/(1+c)]} \tag{2.2}$$

The mass distribution is described by,

$$M(r) = \frac{M_{DM} r^2}{(r+a)^2} \tag{2.3}$$

and the gravitational potential is known analytically, [Hernquist, 1990],

$$\Phi = -\frac{GM}{r+a} \tag{2.4}$$

The local circular velocity is,

$$v_c^2(r) = \frac{1}{r}\frac{\partial \Phi}{\partial r} = \frac{GM}{r(r+a)^2} \tag{2.5}$$

Starscream calculates the total mass of the dark matter by relating the virial velocity, (user-defined), to the virial mass of an NFW profile, [Springel and White, 1999],

$$M_{200} = \frac{{v_{200}}^3}{10GH} \tag{2.6}$$

and calculates r_{200} by

$$r_{200} = \frac{v_{200}}{10H} \tag{2.7}$$

Since r_s can now be found from the concentration, the "a" parameter can be calculated as well.

2.3 Disks

Disks in Starscream are modeled as in [Springel and White, 1999] with density,

$$\rho_d(R,z) = \frac{\Sigma(R)}{2z_0} sech^2(\frac{z}{z_0}) \tag{2.8}$$

and

$$\Sigma(R) = \Sigma_0 e^{-R/R_d} = \frac{M_d}{2\pi {R_d}^2} e^{-R/R_d} \tag{2.9}$$

M_d is the mass of the disk, R_d is the disk scale length, and z_0 is the disk thickness or "temperature."[1]

Starscream assumes that the local circular velocity of this disk is well approximated by the circular velocity of a purely exponential disk,

$$v_c^2(R) = R\frac{\partial \Phi}{\partial R} = 4\pi G\Sigma_0 \frac{R^2}{4R_d}\left[I_0\left(\frac{R}{2R_d}\right)K_0\left(\frac{R}{2R_d}\right) + I_1\left(\frac{R}{2R_d}\right)K_1\left(\frac{R}{2R_d}\right)\right] \quad (2.10)$$

where I and K are modified regular cylindrical Bessel functions of the first and second kind, respectively.

The disk potential is calculated using a Particle Mesh (PM) method with Cloud-In-Cell density mapping. The PM routine in Starscream follows the work of [Hockney and Eastwood, 1981].

2.4 Combining a Disk and a Halo

Starscream follows [Springel and White, 1999] and uses the analytical work of [Mo et al., 1998] to combine the disk and halo.

Four assumptions are made in this model about the nature of the disk. All disks are approximated as exponential and all observed disks are assumed to be dynamically stable against bar formation. The last two assumptions relate the angular momentum and mass of the halo to the disk,

$$J_d = j_d J_{\mathrm{DM}} \quad (2.11)$$

$$M_d = m_d M_{\mathrm{DM}} \quad (2.12)$$

where j_d and m_d are fractions of the total. [Mo et al., 1998] also specify the total angular momentum of the halo in terms of a spin parameter

$$\lambda = \frac{J_{\mathrm{DM}}|E|^{1/2}}{G{M_{\mathrm{DM}}}^{5/2}} \quad (2.13)$$

[1] Starscream assumes a constant value of $z_0 = 0.2R_d$, but this should be a user-defined parameter. See chapter 3.

where, by assuming the particles are all in circular obits,

$$E = \frac{-G{M_{\rm DM}}^2}{2r_{200}} \cdot f_c \tag{2.14}$$

for

$$f_c = \frac{c}{2}\frac{1 - 1/(1+c)^2 - 2\ln(1+c)/(1+c)}{[c/(1+c) - \ln(1+c)]^2} \tag{2.15}$$

Using the above description of the energy, the spin parameter, and the angular momentum,

$$J_d = M_d R_d v_{200} \int_0^{r_{200}/R_d} e^{-u} u^2 \frac{v_c(R_d u)}{v_{200}} du \tag{2.16}$$

it is possible to write an equation for the disk scale length,

$$R_d = \frac{1}{\sqrt{2}} \left(\frac{j_d}{m_d}\right) \lambda r_{200} {f_c}^{-1/2} f_R(\lambda, c, m_d, j_d) \tag{2.17}$$

for

$$f_R(\lambda, c, m_d, j_d) = 2\left[\int_0^{\infty} e^{-u} u^2 \frac{v_c(R_d u)}{v_{200}} du\right]^{-1} \tag{2.18}$$

The circular velocity of the system is

$$v_c(r) = \sqrt{{v_{c,d}(r)}^2 + {v_{c,\rm DM}(r)}^2} \tag{2.19}$$

In this case, v_{DM} is

$$v_{c,\rm DM} = \sqrt{G(M_f(r) - M_d(r))/r} \tag{2.20}$$

for an adiabatically contracted halo with, [Mo et al., 1998]

$$M_f(r) = M_d(r) + M_{\rm DM}(r_i)(1 - m_d) \tag{2.21}$$

Starscream differs from [Mo et al., 1998] in finding the disk scale length. Instead of iterating through several of the above equations, Starscream uses the fitting function for f_r in [Mo et al., 1998],

$$f_R \sim \left(\frac{(j_d/m_d)\lambda}{0.1}\right)^{-0.06+2.71m_d+0.0047/[(j_d/m_d)\lambda]} . \tag{2.22}$$

$$\cdot(1 - 3m_d + 5.2{m_d}^2) \cdot (1 - 0.019c + 0.00025c^2 + 0.52/c)$$

to make a decent guess for the disk scale length. Using the guess, Starscream iterates between the 2.13 and 2.14 above.

Knowing the disk scale length, the halo "a" parameter, and the user-defined parameters, it is now possible to assemble a model.

2.5 Picking Positions and Velocities

To create a pair of colliding galaxies, Starscream must set the position and velocity of each particle. This is very simple for the position and Starscream manages this by using an implementation of the Metropolis-Hastings algorithm.

On the other hand, picking particle velocities is much more difficult. Starscream will set particle velocities to either the local circular velocity or calculate velocity dispersions and pick velocities from a normal distribution. The first option is left for pedagogical reasons, there may be some people who are interested in the difference, while the second is certainly the much more accurate choice.

The method described in [Hernquist, 1993] and [Springel and White, 1999] is used to calculate the velocity dispersions. The method is a bit too detailed for a small user's manual and users are encouraged to read those references for more information. In short, the vertical velocity dispersion is calculated by,

$$\overline{v_z^2} = \frac{1}{\rho}\int_z^\infty \rho \frac{\partial \Phi}{\partial z} dz \tag{2.23}$$

and

$$\overline{v_z^2} = \overline{v_r^2} \tag{2.24}$$

$$\overline{v_z}^2 = \overline{v_r}^2 = 0 \tag{2.25}$$

For the halo, the azimuthal moment is calculated by,

$$\overline{v_\theta^2} = \overline{v_r^2} + \frac{r}{\rho}\frac{\partial}{\partial r}\rho\overline{v_z^2} + v_c^2 \tag{2.26}$$

and for the disk,

$$\overline{\sigma_\theta^2} = \frac{\overline{\sigma_r^2}}{\gamma} \tag{2.27}$$

where γ is given by the epicycle approximation, cf. [Binney and Tremaine, 2008].

Not mentioned above is the streaming velocity of the halo. For this, Starscream follows [Springel and White, 1999] exactly.

2.6 Sample Parabolic Orbit

Since prograde, parabolic orbits produce very prominent tidal tails, [Toomre and Toomre, 1972], a sample orbit parabolic orbit is included in Starscream. The formulation of the parabolic orbit is based on the description in [van de Kamp, 1964].

A parabolic orbit is an elliptical orbit with eccentricity $e = 1$ and the radius is given by the law of ellipses,

$$r = \frac{p}{1 + cos(\nu)} \tag{2.28}$$

where p is the distance of closest approach and ν is so-called *true anomaly*. The velocity components are, in polar coordinates,

$$v_r = \sqrt{\frac{\mu}{p}} sin(\nu) \tag{2.29}$$

$$v_\theta = \sqrt{\frac{\mu}{p}} (1 + cos(\nu) \tag{2.30}$$

μ is the gravitational parameter,

$$\mu = G(M + m) \tag{2.31}$$

Chapter 3

Known Issues and Future Additions

This chapter will list known issues, unfortunate oversights, and planned additional features for Starscream.

3.1 Possible Crashes

As a new and largely untested code, Starscream may turn out to be error prone. There are several rules of thumb that the author practices to reduce the risk of failure when using a new code:

- Never stray from the function prototypes. If a message about incompatible pointer types appears, then they really are incompatible!
- Run the test case thoroughly and check for "nan","inf", or "0.0e0" in the output.
- At least read the "Quick Start" or "Getting Started" section of the manual. This will save so many headaches!
- If you run into a truly insurmountable problem, contact the author or post to a mailing list.

Users should take special note that sending a galaxy system to any function that requires potential information will probably cause a segfault. Setting particle positions, the

potential, or particle velocities before creating the galaxy will also cause a segfault, (as will deviating from that calling order).

3.2 Physics

There are several issues with the physics in Starscream. The most obvious problem with Starscream is the lack of support for bulge or gas particles. Galactic bulges are not a difficult feature to add because they can use the same distribution function as the halo. Adding gas particles may be much more difficult. Users are referred to the discussion in [Springel et al., 2005]. The author hopes to add these galactic components in the future, but does not plan to do it soon.

Starscream would also benefit by adding more particle distributions from which to draw positions. At the moment, users are limited to a Springel-Hernquist model since that is the model with which the author has the most experience. However, there are many more particle distributions and combinations out there and adding them to Starscream should be a fundamentally simple task.

The author attempted to add OPENMP support to Starscream, but was not wholly pleased with the results and released a serial code instead. Hopefully, future version of Starscream will fully exploit parallelism and not have to rely on compile-time parallization tricks like -march=amdfam10, (although that particular option was very efficient for the author).

Finally, the most unfortunate oversight of the author was neglecting to make the disk "temperature" a user-defined parameter. This oversight will be fixed in the next release.

3.3 Future NVidia CUDA Support

The author is currently planning to release a version of Starscream that utilizes the NVidia Compute Unified Device Architecture (CUDA) to calculate the FFTs used for setting the gravitational potential of the disk. This will allow large, fine potential grids to be used while off-loading the extra computational cost to the user's graphics card. While users will always have the option of using the default FFTW function calls, the author would like to

at least provide CUDA support as an option.

3.4 Reporting Bugs and User Support

The author is very eager to receive bug reports and is willing to offer user support and add features on request.[1]

Users can email any bugs or questions to jayjaybillings@gmail.com.

[1]The author will not offer support to users that break Starscream and ask for it to be fixed.

Chapter 4

Function Descriptions

Starscream contains many different functions for operating on galaxies, some of which may be useful for users. The functions that users may find to be the most useful are listed and described here.

int create_galaxy(galaxy *, int [], double, double, double, double, double, double, int, double, int);

Create_galaxy() should be called to create a galaxy structure in memory. It takes as arguments a galaxy type, nine physical parameters, and output option. The nine physical parameters are, in order, the number of particles in the stellar disk, the number of particles in the dark matter halo, the mass fraction of the disk relative to the halo, the angular momentum fraction of the disk relative to the halo, the spin parameter, the halo concentration, the virial radius, the number of points for the potential grid in one Cartesian direction, and the spacing between two consecutive grid points. The final integer value tells the function whether or not it should print physical information about the system. The function returns zero if everything executed properly.

int create_galaxy_system(galaxy *, galaxy *, galaxy *);

Create_galaxy_system() places two galaxies, the first and second arguments, in the same galaxy object, the third argument. Users should note that if the galaxies are not placed on

the orbit, they will be combined, which may not produce realistic results. This function returns zero on success.

int set_galaxy_coords(galaxy *);

This function must be called to set the particle positions. Positions are drawn from particle distributions using the Metropolis-Hastings algorithm. This function takes a galaxy object as an argument after create_galaxy() has been called for that object. The function returns zero on success.

int set_galaxy_potential(galaxy *, int);

This function calculates the potential due to the disk using the Particle Mesh (PM) method. The first argument must be a galaxy and the second argument is an option to print the potential in the xy-plane on exit, which can be a useful diagnostic. If users plan to set velocities using dispersion, (argument two set to one in set_galaxy_velocity()), set_galaxy_potential should be called before setting the velocity. This function returns zero on success.

int set_galaxy_velocity(galaxy *, int);

Particle velocities are set by calling set_galaxy_velocity(). The first argument is a galaxy object and the second is an integer that sets the type of velocity to be calculated. If the second option is zero, the particle velocities are equal to the local Kepler velocity. This isn't too useful, in reality, but it is left in the code for pedagogical purposes: allocating a galactic disk with Keplerian particle velocities is a great way to see bar formation from instability. Passing a value of one for the second argument will calculate the local velocity dispersions and draw the velocity components from a normal distribution. This function returns zero on success and must be called after the particles positions have been set.

void destroy_galaxy(galaxy *, int); and
void destroy_galaxy_system(galaxy *, int);

Galaxies are destroyed by calling destroy_galaxy() or destroy_galaxy_system() for the particular object. The second option in both functions should be set to zero unless users are trying to debug memory problems. Settings these options to zero will print a message when entering and exiting the function. Destroying the galaxies insures that the memory is properly deallocated before exiting or reuse.

void write_galaxy_position(galaxy *,int);,
void write_galaxy_position_disk(galaxy *);,
and void write_galaxy_position_halo(galaxy *);

These functions will write the particle positions to standard out in XYZ format. The second option on write_galaxy_position() will print "#Disk" and "#Halo" before printing the particle positions for that component. The "#" is a uniquely searchable character for parsing the file and lines beginning with it are ignored altogether by some graphics packages.

void write_galaxy_velocity(galaxy *,int);,
void write_galaxy_velocity_disk(galaxy *);,
and void write_galaxy_velocity_halo(galaxy *);

These functions will write the particle velocities to standard out in XYZ format. The second option on write_galaxy_velocity() will print "#Disk" and "#Halo" before printing the particle velocities for that component. The "#" is a uniquely searchable character for parsing data and lines beginning with it are ignored altogether by some graphics packages.

void write_galaxy_potential(galaxy *);

Write_galaxy_potential() will write the gravitational potential of a galaxy to standard out. A galaxy system should not be sent to this function since galaxy systems do not contain any potential information.

The output format is XYZ,disk potential,halo potential,total potential.

void write_galaxy_rotation_curve(galaxy *);

This function will calculate the local circular velocity out to a distance equal to the virial radius and write this information to standard out.

int write_gadget_ics(galaxy *, char *);

Users should call this function to write a galaxy or galaxy system, the first argument, to a file, the second argument, that is compatible with Gadget2. Write_gadget_ics is a strict inversion of the load_snapshot function. This function returns zero on success.

int load_snapshot(char *, int);

Load_snapshot will read Gadget2 snapshot file and returns zero on success. This routine was take from the "Analysis" directory of the Gadget2 distribution.

int unload_snapshot();

Unload_snapshot must be called immediately after a user finishes working with the information from a particular Gadget2 snapshot file. Starscream does not currently support a Gadget2-style data type, but instead reads snapshot information into a single global structure. This scheme is sufficient to work with a single snapshot, but requires that users unload snapshots from memory if they want to load another.

void copy_galaxy(galaxy *, galaxy *, int);

Copy_galaxy copies argument one into argument two exactly. This function returns zero on success.

void set_orbit_parabolic(galaxy *, galaxy *, double, double);

This function sets arguments one and two on a parabolic orbit with each other. The third argument is the initial separation of the galaxies from the center of mass and the last argument is the distance of closest approach. This function is the only orbit available, to date, in Starscream.

Bibliography

[Binney and Tremaine, 2008] Binney, J. and Tremaine, S. (2008). *Galactic Dynamics*. Princeton University Press, 41 William Street, Princeton, New Jersey.

[Hernquist, 1990] Hernquist, L. (1990). An analytical model for spherical galaxies and bulges. *The Astrophysical Journal*, 356:359–364.

[Hernquist, 1993] Hernquist, L. (1993). N-body realizations of compound galaxies. *The Astrophysical Journal Supplement Series*, 86:389–400.

[Hockney and Eastwood, 1981] Hockney, R. and Eastwood, J. (1981). *Computer Simulations Using Particles*. CRC Press, 6000 Broken Sound Parkway, NW, (Suite 300) Boca Raton, FL.

[Mo et al., 1998] Mo, H., Mao, S., and White, S. D. M. (1998). The formation of galactic disks. *Monthly Notices of the Royal Astronomical Society*, 295:319–336.

[Springel et al., 2005] Springel, V., Di Matteo, T., and Hernquist, L. (2005). Modelling feedback from stars and black holes in galaxy mergers. *Monthly Notices of the Royal Astronomical Society*, 361:776–794.

[Springel and White, 1999] Springel, V. and White, S. D. M. (1999). Tidal tails in cold dark matter cosmologies. *Monthly Notices of the Royal Astronomical Society*, 307:162–178.

[Toomre and Toomre, 1972] Toomre, A. and Toomre, J. (1972). Galactic bridges and tails. *The Astrophysical Journal*, 178:623–666.

[van de Kamp, 1964] van de Kamp, P. (1964). *Elements of Astromechanics.* W. H. Freeman and Company, 660 Market Street, San Francisco 4, California.

Acknowledgments

The author would like to thank Shuhua Liang, Steven Boada, Alex McCaskey, and Mike Guidry for their contributions to Starscream. Shuhua shared his unbelievable intellect and was always available to discuss the finer points of an algorithm or bit of Physics with the author.[1] Steven, Alex, and Mike all provided encouragement and enthusiasm, either by regularly inquiring after the status of the code, spreading AVI movies of the author's galaxy collisions across continents, or trying the authors algorithms in their own projects.

Additionally, the author would like to thank Volker Springel for pointing him in the right direction by recommending [Springel and White, 1999].

This work was not funded.

[1] It may even be the case that Shuhua typed a character or two of the code but decided that programming would ruin his reputation and promptly erased them.

About the Author

Jay Billings holds a Bachelor of Science degree in Physics from Virginia Tech and a Master of Science degree in Astrophysics from the University of Tennessee, where he is currently a member of the Department of Chemisty. He plans to one day enlist in the Army as a Pastrychef.