Sink Particles In Torch
Last updated: March 24, 2020

1 What Are Sink Particles?

Sink particles are computational objects used in hydrodynamical simulations where excessive refinement is computationally prohibitive. Sinks eliminate the ability to evolve the dense gas dynamics within their volume thereby saving computation time, and negating the need for further refinement (which would decrease the simulation timestep via the Courant condition). Sinks are an incredibly useful technique in astrophysical computations as simulations often model physical processes on scales ranging many orders of magnitude.

Sinks were first designed for SPH simulations of accreting primordial binary systems (Bate 1995), were modified to operate in Cartesian adaptive mesh refinement codes (Krumholz 2004), and then enhanced and included in the FLASH magnetohydrodynamics software suite (Federrath 2010).

The evolution and development of the sink particle as a computational tool is certainly an interesting trail to follow and worth the deep dive into the literature, but here I will focus on how sinks operate within FLASH and what additions have been made in Torch (Wall 2019). Their main function in Torch is to eliminate the need for MHD calculations when the Jean's criterion is satisfied at the maximum level of refinement, accrete infalling gas, and introduce star particles (AMUSE) to the simulation space.

2 Setup Parameters

In the flash.par parameter file, three variables are of significant importance when it comes to the formation of sinks.

2.1 Sink Size

First, `sink_accretion_radius` sets the physical size of the sink as well as the sink creation test volume V. The accretion radius is set such that the width of the sink particle corresponds to the smallest resolvable Jean's Length Λ_J.

According to the grid hydrodynamical analysis by Truelove et al. (1997), a Jean's Length must be resolved by a minimum of 4 grid cells to ensure that the gas will continue collapsing. Krumholz et al. (2004) adapted this method for AMR capable codes and set the sink diameter to $5\Delta x$ where Δx is the highest refinement cell width. Federrath et al. (2010) continues this convention for their implementation in FLASH. All told, the `sink_accretion_radius` parameter is conventionally set to $2.5\Delta x$.

2.2 Density Threshold

With our assertion that the sink particle be just able to encapsulate the smallest resolvable Jean's length, we can then manipulate the equation for the Jeans length (Jeans 1908; Equation 1) to solve for the corresponding density:

$$\rho_{res} = \frac{\pi c_s^2}{G\lambda_J^2} = \frac{\pi c_s^2}{G(2 * 2.5\Delta x)^2} \tag{1}$$

Each timestep, FLASH will check each highest refinement cell against this parameter. If the gas density in a cell exceeds ρ_{thresh} then the series of steps and checks described in Section 3 are initiated.

Within an already existing sink particle, each cell within the sink volume is again checked against the density threshold parameter. If the value is exceeded, the excess mass is accreted as described in Section 4.

2.3 Refinement Criteria

In flash.par, `refine_var_1` sets the field variable on which the grid AMR capabilities are activated (such as "dens", "pres", "temp"). A field value of "none" still results in (what I believe to be) AMR activation on the Jean's Length (i.e. if a cell's density/size corresponds to a Jean's length smaller than the cell itself, the cell is refined).

I find "none" to function well while also limiting the amount of the simulation space that is overrefined while having no interesting behavior. For example, refining on "dens" highly refines the boundary of the CNM and WNM which we are not necessarily interested in (especially not in the first Myrs).

3 Rules of Sink Creation in Torch

Firstly, the sink method (Federrath 2010) iterates over all cells in the simulation space that exist at the highest level of refinement checking density of each. If a cell's density exceeds the set density threshold, a spherical test volume V with radius r_{acc} centered around the cell and a series of checks is initiated:

1. Are all cells within control volume V at the highest level of refinement?
2. Is the divergence of gas in V negative?
3. Is the central cell at a gravitational potential minimum?
4. Is gas within V Jean's unstable ($E_{grav} > 2E_{therm}$)?
5. Is the gas bound?
6. Is V entirely outside r_{acc} of other sinks?

The order of these checks does not matter, but are arranged such that the computationally least expensive checks are done first. If any check is failed, the sink creation process is halted. If all checks are passed, mass is removed from all cells within V until each has a density equal to or less than the density threshold. The sink particle is then placed at the central cell and all removed gas mass is assigned to the sink particle. In addition to the conservation of mass during the formation, the conservation of linear and angular momentum is also upheld: the momentum of the removed gas is summed and assigned to the sink.

At this point, the sink behaves as a Lagrangian particle: permitted to move around the simulation space (always being centered at a single cell) and can be acted upon by the gravity of surrounding gas.

4 Rules of Sink Accretion in Torch

Once created, the mass of the sink itself and the gas mass within the cells on which the sink sits results in infalling gas, increasing the gas density within the cells in and around the sink. If any gas density within the sink's r_{acc} exceeds the density threshold, the sink will accrete the excess gas if brief set of checks is passed:

1. Is the excess mass ΔM bound to the sink particle?
2. Is the radial velocity of ΔM negative with respect to the sink center?

Of course, the same conservation rules apply as with the sink creation process. The accreted gas mass and momentum is added to the sink particle.

5 Listing and Placement of Stars

Upon creation, a sink particle will have a star list generated and assigned to it. The list is derived from Poisson sampling the Initial Mass Function (IMF; Kroupa 2002) of the total gas cloud. For a $10^4 M_\odot$ cloud, a sink will have a list with around 10.000 stars in the 0.08–0.10$M_\odot$ range and a handful of $100M_\odot$+ stars. The list of stars is then randomized. If the sink exceeds the mass of the first star on the list, that mass is removed from the sink and a star particle is placed inside of the sink volume. This mass condition is then checked for the next star on the list and repeated until the sink cannot form the next star. Then, the process repeats after the accretion step in subsequent time-steps.

Star placement is handled in `src/torch_sf.py`. As of commit `a6c07ed`, a star particle is placed inside of the corresponding sink volume following an isothermal distribution. The star particle is then assigned a velocity derived from a normal distribution around the parent sink's velocity with a scale set to the sound speed of the gas $c_s = \sqrt{P/\rho}$ inside of the sink. Excerpts of the position and velocity methods can be seen below. This is what was used in Wall et al. 2019 and Wall et al. 2020 in prep.

```
def random_three_vector(n=1):
    """
    Generates a random 3D unit vector (direction) with a uniform spherical distribution
    Algo from http://stackoverflow.com/questions/5408276/python-uniform-spherical-distribution
    """
    three_vector = np.zeros((n,3))

    phi = np.random.uniform(0,np.pi*2,n)
    costheta = np.random.uniform(-1,1,n)

    theta = np.arccos( costheta )
    three_vector[:,0] = np.sin( theta) * np.cos( phi )
    three_vector[:,1] = np.sin( theta) * np.sin( phi )
    three_vector[:,2] = np.cos( theta )
    return three_vector

formed_stars = True

# Isothermal spherical distribution.
star.position = sink_pos + sink_rad*np.random.rand(nnew,1)*random_three_vector(nnew)
# Gaussian distribution satisfying <vx**2> = sink_cs**2
# so that stars' specific energy 1/2 <v**2> = (3/2)*sink_cs**2
# matches gas specific energy P/rho/(gamma-1) for gamma=5/3
# with cs = sqrt(P/rho) from Particles_sinkCreateAccrete.F90
star.velocity = sink_vel + (np.random.normal(scale=sink_cs.value_in(units.cm/units.s), size=(nnew,3)) |
    units.cm/units.s)
```

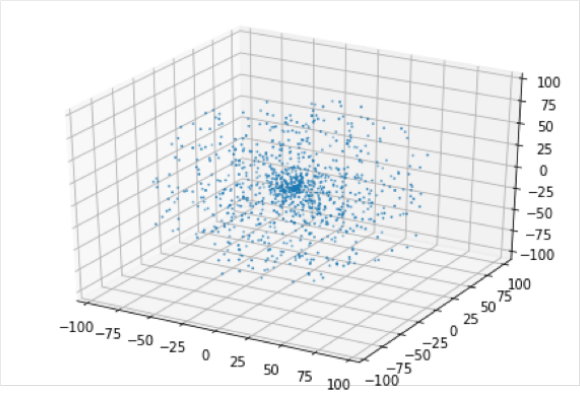

Figure 1. Visual guide for 1000 particles placed in an isothermal distribution. Since a Torch sink particle only creates a few stars at once and is free to move from it's original position (to other cell centers), it is unlikely a distribution like this will spawn into the simulation space.

An important point is star particle are Lagrangian particles but are unconfined to specific positions within a cell (unlike a sink particle). In addition, a star's velocity through the computational space is not at all tied to the Courant condition and so stars are allowed to move supersonically and cross multiple cells in a single computation time-step. See Section 6 for further discussion.

6 Discussion of Methods

6.1 Star Particle Placement, Velocity, and the Courant Condition

6.1.1 Star Particle Placement

There are some considerations needed in how stars are placed in Torch. The isothermal distribution assumes that the region in the sink with the highest rate of star formation is directly in the center. However, the purpose with the sink particle is to denote a region where further gravitational collapse in known to occur, but the specific dynamics

and further fragmentation are not followed/resolved. Therefore, it is reasonable to give to-be-made stars an equal probability of being placed anywhere in the sink rather than oversampling the central region (**SCL to be tested - 03/23/20**).

6.1.2 Star Velocity

Current (March 2020) implementations of Torch use the gas sound (c_s) as the variance in star particle speed. c_s data is determined within and passed from the sink particle module in FLASH. This method of assigning star velocities functions well for early times in an embedded cluster's life where the gas being accreted into the sink is cool, dense 10K gas. However, if/when a massive star forms, the radiative feedback will quickly increase the temperature of the gas in and around the sink, causing c_s and the velocity of any subsequently formed stars to also increase unrealistically. Dr. Joshua Wall implemented a quick fix for this (which I believe is turned 'on' by default) that calculates c_s by only iterating over cells within 2 radii of the sink that have gas temperatures of <100K. A more robust fix would be to use the gas virial velocity as the stellar velocity spread. This can be accomplished by determining the total gas mass a sink is sitting on: $V_{vir} \equiv \left(\frac{3GM}{5R}\right)^{1/2}$ (**SCL TODO 03/23/20**).

6.1.3 Star Particles and the Courant Condition

Star particles and Lagrangian particles and are free to move about the simulation space without being confined to any aspect of the Grid. This fact has a couple implications. Firstly, the movement of stars is unconstrained by the Courant condition, permitting stars to be supersonic. While this is not necessarily an unphysical occurrence, this means that stars are allowed to pass through multiple gas cells within a single system evolution timestep which could lead to significant error in star-gas/gas-star gravitational interactions especially at the highest levels of refinement.

Secondly, refinement regions are not determined by the presence of star particles. If gas-star drag is a behavior of interest, it can only ever be resolved at the highest levels of refinement. Of course, all of the stars are formed within sink particles which are necessarily at the highest grid refinement level, so this may only apply to outlier stars with an outlier radial velocity away from the sink.