SPH Analysis Code - Manual and Methods for Estimating Melting and Mixing

Roshan Mehta
Advisor: Miki Nakajima
University of Rochester

January 7, 2026

1 Overview

This document explains the methodology behind the smoothed particle hydrodynamics (SPH) analysis code provided in this repository folder as well as how to use it step-by-step. SPH is a Lagrangian method of simulating particle systems as fluids. Here, it treats the collision between two planets as such, and generates data frames that can be analyzed for various properties post-impact. The provided code allows users to estimate the amount of target mantle melting that occurs and determine how much of the impactor's core mixes with the target body's mantle.

To begin, the user must first have data to analyze. This analysis code is programmed to run off of the Nakajima Lab's SPH code at the University of Rochester (`https://github.com/NatsukiHosono/FDPS_SPH`). For user convience, this code has been used to generate a low-resolution canonical impact scenario; one sample timestep of this simulation has been linked in the `SampleData.txt` file in the repository for those who wish to try analyzing it. While the programs are designed to run for data frames generated using the Nakajima Lab's SPH code, many of the core ideas used can be adapted to other SPH codes with minor reformatting.

The programs are all based in Python 3 and use the Matplotlib, NumPy, SciPy, and PyVista libraries. Once all of these installations are complete, the user can begin analyzing their data.

2 Sorting Particle Data

The first step in estimating the amount of melting and mixing that occurs is sorting the particle data into planetary, disk, and escaping material. To do this, navigate to the `SortParticles.py` file. At the top of the document, the program requires seven user inputs: `path, outputpath, outputnumber, ncores, xlim, percentile,` and `iterations`.

The input, `path`, represents the system file path to the folder the SPH data is stored in (i.e., `/home/[user]/collision_data/`). Similarly, the `outputpath` represents the system file path to where the sorted data should be stored. It is recommended to make `outputpath` the same as `path` so that all data is stored in the same place. Both inputs should be entered as Python strings.

The SPH code outputs files that are titled `results.XXXXX_YYYYY_ZZZZZ.dat`, where `XXXXX` is the `outputnumber`, `YYYYY` is the number of cores or processors used in the simulation, and `ZZZZZ` is the file number ranging from 0 to one less than the total number of cores. The `ncores` input should be equal to `YYYYY`. For example, the canonical impact data provided has files that are titled `results.00740_00300_ZZZZZ.dat`. Therefore, if analyzing this data, the user should enter `outputnumber = 740` and `ncores = 300` into the program.

The inputs `xlim, percentile,` and `iterations`, are specific to how the program works. Once the code is run, the program first reads through all the data and centers the particles at the main body's center of mass. It then plots the distribution of particle distances from the center of mass in a histogram. `xlim` controls the cutoff distance for the x-axis on the histogram, and should be entered in units of 10^6 m. The

code will then prompt the user to double-click on a point in the histogram which represents an initial guess for the planetary radius. For collisions with no resulting disk, this estimate is easy. However, if there is significant debris, it can be difficult to determine an initial guess. A good place to double-click is at the base of the main particle distribution, as shown in Figure 1.

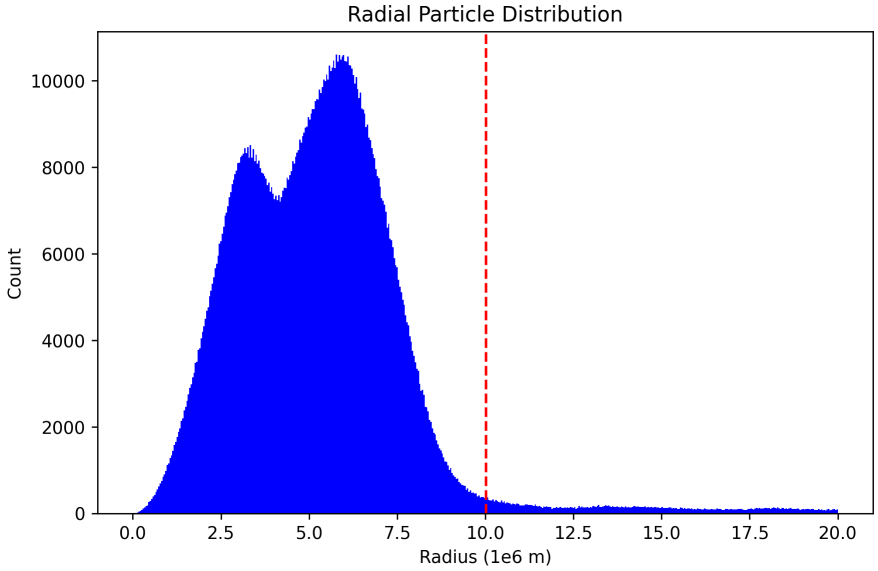


Figure 1: A histogram generated for a collision that has significant disk material, as seen by the trailing right end of the histogram. A good initial guess for the radius is at the base of the main distribution, represented by the red line.

The program then proceeds to determine which particles are part of the planet, which are part of the orbiting disk, and which escape the main body entirely using an iterative scheme proposed by Canup (2004). First, an initial value for the main body's mass, M, is calculated by summing the mass of each particle within the initial planetary radius. Then, each individual particle is tracked using classical orbital dynamics. Its energy is calculated with

$$E = \frac{1}{2}v^2 - \frac{GM}{r}, \tag{1}$$

where v and r are the particle's velocity and radial distance from the center of mass, respectively, and G is the gravitational constant. The particle's periapsis is also calculated with

$$r_p = a(1 - \varepsilon), \tag{2}$$

where a and ε are the orbital semi-major axis and eccentricity, respectively. These are calculated with

$$a = -\frac{GM}{2E} \tag{3}$$

$$\varepsilon = \sqrt{1 + \frac{2El^2}{(GM)^2}}, \tag{4}$$

with l being the particle's specific angular momentum. The program then determines if each particle is escaping or bound based on if its energy is positive or negative, respectively. Of those that are bound, a particle is considered as planet material if its periapsis is less than or equal to the initial guess for the planetary radius, and disk material otherwise. A better estimate for the planetary radius can then be obtained by taking a percentile measure of the planet particles' distances from the center of mass. This is what the input, `percentile`, does. For example, if `percentile = 99`, the program will take the 99^{th} percentile of the planet particles' radial distances as a better estimate for the main body's radius. The

default value of 99 tends to work well, but can be adjusted if necessary. This entire process of tracking the particles' orbits and sorting the particles is then repeated on a second iteration using this new radius value. This again returns a better estimate which can be used for a third iteration, and so on. Ideally, the iterations continue until the radius and mass values converge, which can be seen in the program's print statements. The user can control how many times this process occurs with the `iterations` input (for example, setting `iterations = 10` means the process will repeat 10 times). For systems with no disk, usually three to five iterations is sufficient to achieve convergence. However, for collisions with significant orbiting debris, five to ten iterations may be necessary.

One important note is that this process is not designed for oblique impacts at low velocities (which is about equal to the system escape velocity). This is because these impacts often begin with an initial grazing collision, with the majority of the impactor staying intact and re-colliding with the planet later on. If the sorting program is run before this second collision occurs, the radius value will expand until it encompasses the orbiting impactor. This happens because its periapsis is less than the planetary radius, since it will eventually re-collide with the target. This, of course, gives a nonphysical value for the planetary radius that cannot be used. Additionally, it might not be ideal to run the simulation until this second collision occurs, as this can take several hours to happen. Even if this is feasible to simulate, the effect of numerical viscosity becomes non-negligible near 25 hours of simulated time, meaning this option is not optimal (Canup, 2004). Instead, for these types of collisions, it is best to set `iterations` to `0`. This makes it so that the final radius used is the same as the initial one the user inputs as a guess.

Once the program finishes running, it will output one file titled `SortedData_XXXXX.txt` to the specified `outputpath`, where `XXXXX` is the specified `outputnumber`. The file header is formatted as follows:

```
<simulation time (s)>
<total number of particles>
<planetary radius (m)>
<planetary mass (kg)>
<x center of mass (m)> <y center of mass (m)> <z center of mass (m)>
```

After the header is the particle data. Each line represents a different particle, and is formatted as follows:

```
<id> <tag> <mass (kg)> <x position (m)> <y position (m)> <z position (m)>
<x velocity (m/s)> <y velocity (m/s)> <z velocity (m/s)> <density kg/m^3)>
<internal energy (J)> <pressure (Pa)> <potential energy (J)> <entropy (J/K)>
<temperature (K)> <fate>
```

The particle data is formatted the same way that it is outputted from the SPH code. The only change is that after `temperature`, a new data category, `fate`, is added. This value can be 0, 1, or 2, which indicates that the particle is planet, disk, or escaping material, respectively.

Now that the data has finished sorting, the amount of melting and mixing that occurs can be analyzed.

3 Analyzing Melting

The melting code provided allows users to estimate how much of the planetary mantle is melted during impact, and provides a cross section plot to visualize the melt mass.

To begin, navigate to the `AnalyzeMelting.py` file. This program prompts the user for several inputs: `path, outputpath, outputnumber, gamma, angle, axesdim,` and `thickness`. The input, `path`, refers to where the `SortedData_XXXXX.txt` file is stored from Section 2, and the input, `outputpath`, is where the output plot will be stored. Both should be entered as Python strings. The input, `outputnumber`, is defined as it is in Section 2. The inputs `gamma` and `angle` are needed for the melting plot title, and represent the impactor-to-total mass ratio and the collision angle, in degrees, respectively. The inputs, `axesdim` and `thickness`, are needed for generating the melting plot, and respectively represent the length of each axis and the thickness of the cross section, both in units of 10^6 m.

The program will proceed by calculating a melting temperature, T_m for each gravitationally bound silicate particle. This criteria is defined by Rubie, et al. (2015) as

$$T_m = \begin{cases} 1874 + 55.43P - 1.74P^2 + 0.0193P^3, & P \leq 24 \text{ GPa} \\ 1249 + 58.28P - 0.395P^2 + 0.0011P^3, & P > 24 \text{ GPa} \end{cases}, \tag{5}$$

where P is the particle's pressure in GPa. If a particle's temperature is greater than its melting temperature, it is considered molten. Otherwise, it is considered solid. Partial melting is not considered. The program will finish by saving a plot to the specified `outputpath`, titled `Melting_XXXXX.png`, where `XXXXX` is the specified `outputnumber`. In the plot, blue particles represent solid material, while red particles represent molten material (see Figure 2 for reference). The program will also print the final melting percentage in the terminal.

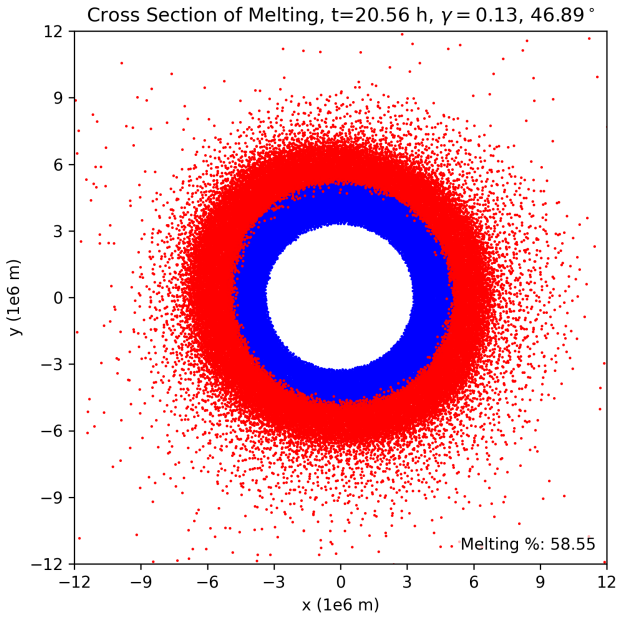


Figure 2: The melt-mass plot generated for the sample canonical impact data provided. Red particles represent molten material, while blue particles represent solid material. The melt-mass percentage is shown in the bottom right.

4 Analyzing Mixing

The mixing code allows users to determine how much of the impactor's core merges with the target's mantle post-collision. It outputs a mixing percentage, as well as two visualization plots; one is a projection of particles onto the x-y plane, and the other is an animated 3D render that shows the mixing distribution.

To begin, navigate to the `AnalyzeMixing.py` file. The program prompts the user for eight inputs: `path`, `outputpath`, `outputnumber`, `gamma`, `angle`, `axesdim`, `cameraposition`, `elevation`, and `settled`. The first six are defined as they are in Section 3. The input, `cameraposition`, is a tuple and refers to the viewing position for the 3D render along the x, y, and z axes. The x and y coordinates are in units of 10^6 m, and the z should be entered as 0 (e.g., (100,100,0)). The input, `elevation`, refers to the angle of the camera above the x-y plane. Lastly, `settled` refers to if the main body has reached a steady state, and if there is

no second collision that will occur (you will know if there will be a second collision if the radius expands to a nonphysical value in the sorting stage; see section 2). If the body is in a steady state with no incoming second collision, set `settled` to `True`. Otherwise, set it to `False`.

If `settled = True`, the program estimates the amount of mixing by sorting through each iron particle and determining how many other iron particles are within 0.05 times the planetary radius from it using a SciPy tree. This is known as the number of "neighbors" that particle has. After sorting through this data, the code will generate a histogram of the distribution of neighbors for each iron particle. There should be a main distribution off the right. This represents iron particles in the resulting body's core, as these particles will have a large number of neighbors. Additionally, there will likely be a spike towards the left. This represents parts of the impactor core that have mixed with the target mantle, as they will likely have far fewer iron neighbors. The program will ask the user to double-click on a point in the histogram representing a cutoff value that will be used to analyze mixing. Specifically, if a non-escaping impactor core particle has fewer neighbors than the cutoff, it will be considered as mixed with the target mantle. Otherwise, it will be considered as mixed with the target core. A good point to choose is right before the main distribution, as this is the natural separation between the mantle and core mergers (see Figure 3 for reference). The plot also has zoom and pan options in the bottom left of the window; it is important to use these in order to zoom in on the histogram and get more accurate inputs.

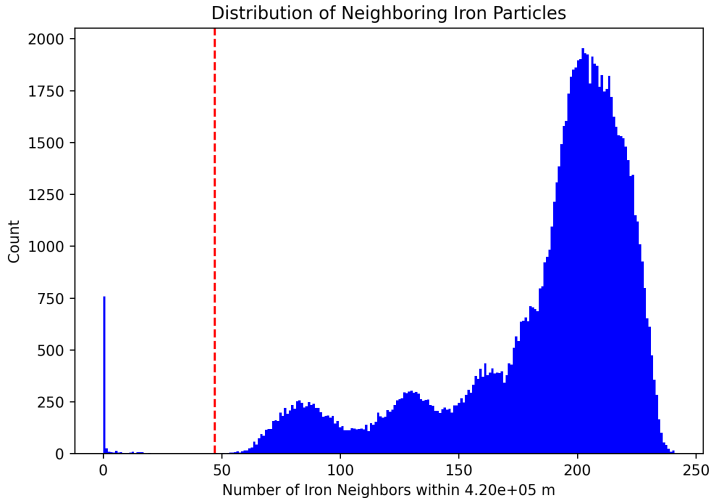


Figure 3: The histogram of neighboring iron particles for the sample canonical impact data provided. The main distribution off to the right represents the core of the main body, while the spike towards the left represents the impactor core particles that have merged with the target mantle. A good point to choose as the cutoff is at the start of the main distribution.

After this, the code will sort through the gravitationally bound impactor core particles and determine which ones have a number of neighbors above versus below the chosen cutoff value.

The above methodology works very well when the body is settled. However, if `settled = False`, this approach can be inaccurate. This is because if a large part of the impactor is still gravitationally bound to the planet, it can still have the majority of its iron core. The code will then consider these iron particles in the mixing calculations, even though it doesn't know their final fate after the second collision settles. A better approach may therefore be to analyze the mixing after the first collision only. To do this, set `settled = False`. The code will show a distribution of the iron and silicate particles in the main body at the specified `outputnumber`. It will then prompt the user to double-click on a point in the histogram that represents the core-mantle-boundary (CMB). A good point to click on is right where the main distribution of iron falls off (see figure 4 for reference[1]). The distributions are plotted semi-transparently to make finding the CMB easier. However, in most of these types of collisions, there will not be as much overlap as is shown in Figure

[1]Note: Figures 4 and 5 are not plots of the sample canonical impact data, as the canonical scenario is not a highly oblique impact that needs `settled` to be set to `False`.

4 (the main reason the plot in Figure 4 was chosen for this document is because the exaggerated overlapping region makes it clear where the optimal place to double-click is in general).

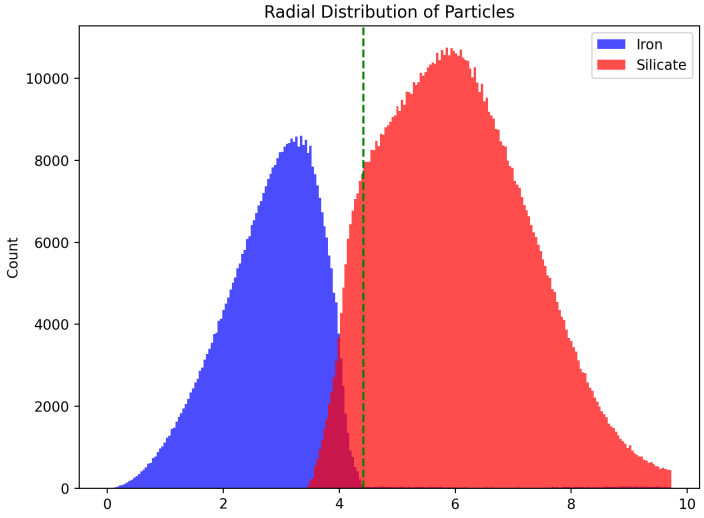


Figure 4: The distribution of iron (blue) and silicate (red) particles in the main body. The best place to double-click on to represent the CMB is right where the iron distribution falls off.

After exiting the plot, the code will generate a second histogram that shows the radial spread of *all* iron particles. Two main distributions should appear; the one on the left represents the target core and the one on the right represents the orbiting impactor core. The user should double-click on a point in the histogram that lies just before the impactor distribution (see figure 5 for reference). After exiting this second plot, the code will then determine how many impactor core particles lie between this point and the CMB. These will be considered as particles that have mixed with the target mantle.

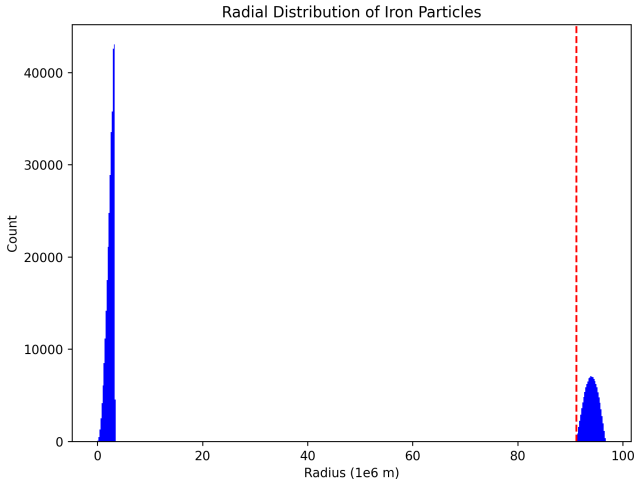
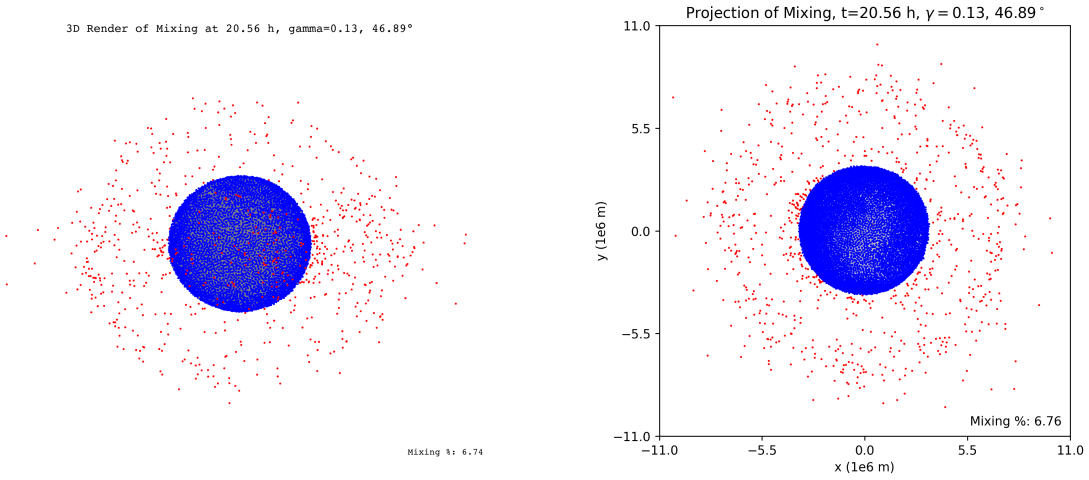


Figure 5: The radial distribution of iron particles. The best place to double-click to represent the radial cutoff is just before the distribution of orbiting impactor iron.

For both the CMB and radial cutoff histogram plots, be sure to use the zoom and pan options to zoom into the figure and ensure you are clicking on a point that truly represents the start or end of a distribution.

Regardless of what `settled` equals, the code will then show a projection onto the x-y plane of the results, along with an interactive 3D render. The projection will be titled `MixingProj_XXXXX.png` and will be saved to the specified `outputpath`, where `XXXXX` represents the specified `outputnumber`. The code will then ask the user if they would like to generate a 3D animation of the results. If they enter `Yes` in the terminal, the program will save a rotating GIF file titled `Mixing3D_XXXXX.gif` to the `outputpath`, with `XXXXX` once again being the `outputnumber`. This animation will show a comprehensive visualization of the mixing distribution. Examples of these plots are shown in Figure 6. In both, only iron is plotted; blue particles represent impactor core particles that have sunk to the target core and red ones represent those that have mixed with the target mantle. The target core is plotted in gray. However, in some cases it may not be visible if there is a coating of the impactor core particles surrounding it. In addition, the code will output the final mixing percentage in the terminal.



(a) **(b)**

Figure 6: A screenshot of the 3D render animation (a) and the projection (b) of mixing for the sample canonical impact data provided. Blue particles represent impactor core particles that have sunk to the target core, while red ones represent those that have merged with the target mantle. The merging fraction is represented by the percentage at the bottom right of each plot.

5 References

All programs and data visualized in this project were created and generated by Roshan Mehta under the supervision of Miki Nakajima at the University of Rochester. Data was generated using the Nakajima Lab's SPH code (`https://github.com/NatsukiHosono/FDPS_SPH`) on BlueHive, the University of Rochester's supercomputer. Additional references are listed below.

[1] Canup, R. M. 2004, Simulations of a late lunar-forming impact, Icarus, 168, 433–456, `https://doi.org/10.1016/j.icarus.2003.09.028`

[2] Marchi, S., Canup, R. M., & Walker, R. J. 2018, Heterogeneous delivery of silicate and metal to the Earth by large planetesimals, Nat. Geosci., 11, 77–81, `https://doi.org/10.1038/s41561-017-0022-3`

[3] Nakajima, M., Golabek, G. J., Wünnemann, K., Rubie, D. C., Burger, C., Melosh, H. J., Jacobson, S. A., Manske, L., & Hull, S. D. 2021, Scaling laws for the geometry of an impact-induced magma ocean, Earth Planet. Sci. Lett., 568, 116983, `https://doi.org/10.1016/j.epsl.2021.116983`