Compact Stars via Smoothed Particle Hydrodynamics with Rotation

Matthew Portman

May 10, 2018

1 Introduction

Smoothed Particle Dynamics is a Lagrangian method of solving the spatial part of PDE's and is often used to model fluid flow. SPH is a mean-field method which uses a function to smooth the contributions between particles/fluid packets also known as a smoothing function/kernel:

$$W = \frac{1}{\pi h^2} e^{-\frac{|r|^2}{2h^2}}; \quad W_{ij} = W(|r_i - r_j|; h) \tag{1}$$

per [3] where h is a smoothing parameter. More specifically, SPH is used to solve the advection equation in space with a time-stepping scheme such as Euler or Leap frog. Various toy models have been developed that explore both 1D and 2D (see [2], [4], and [5]) using SPH with varying levels of complexity. For the scope of this project,we chose the basic SPH equations with considerations for a polytropic equation of state as in Equation 2 and modifiable gravitational parameters in Equation 3 that would simplify the contribution due to gravity ([2] and references inside).

$$P = k\rho^2 \tag{2}$$

$$a_i = -\nu v_i - \lambda x_i - a_i \tag{3}$$

where k, ν,and λ are modifiable parameters to adjust the pressure and acceleration due to gravity [4]. Other methods are available (see [6]) but were outside the scope of this project. We also included rigid and differential rotation as a starting point for future work on the subject via [1]. Please see the code for further references in regards to the specific equations used.

2 Implementation

2.1 Serial

For this project, we developed a serial code which followed the formulation of SPH for stars by Monaghan [3]. Differential rotation was implemented by the equation:

$$\frac{\Omega}{\Omega_c} = \frac{A^2}{A^2 + r^2 sin^2\theta} \tag{4}$$

from [1] where Ω is the angular velocity, Ω_c is the angular velocity at the center, and A is a parameter related to the amount of differential rotation which was chosen and could be modified. Although boundaries are an interesting topic in SPH (see [2]) a boundary condition which acts a spring to return particles inside the radius of the star was used by way of re-normalizing at every time step.

To begin the simulation, the particles were randomly spaced throughout the radius of the star by their distance from the center and by their angular position. They were then stepped forward in time by Euler's method throughout the duration of the simulation. Although we initially chose the Leapfrog method, it proved too difficult to use successfully so we reverted to Euler's method to maintain proper implementation and more reliable results. In initial runs of the data and those used to generate the figures, we used 1000 particles to get an estimation of the expected behavior of the full results and discernible figures. For later runs, we used 10,000 particles to generate a more accurate model of the toy star as recommended by Rosswog [6] while outputting particle positions and their change in thermal energy with time (power) to gauge accuracy of the model.

For every run, we used a time step of 0.04 s, a max time of 10 s or 250 iterations and other parameters per [2]. The rotation parameters were chosen at random. The code was checked for causality violation by calculating the speed of sound. If the speed of sound was greater than the speed of light then causality had been violated and the model was no longer physical.

2.2 Parallelization

To parallelize, we first ran gprof to determine which parts of the code took longest (but forgot to screenshot it). The bulk of the calculations came from determining the acceleration and power which both rely on a function call which analytically calculates the gradient of the smoothing function. Since this is a linear operation, parallelizing is applicable and useful for problem sizes large enough to more accurately model the star. To implement parallelization, we attempted to use a 2D grid of blocks with 1024 threads per block in 1D passing the position array (of all particles) to the GPU and operating on that array as necessary to generate the gradient function.

To run CUDA, we used Tristan's computer with Ubuntu 18.04 and a 1080ti(!). Due to compatibility issues with CUDA and PGI on Tuckoo (and a personal laptop) his computer was the next best bet. The bulk of the code, CUDA included, was written in Fortran with a MATLAB script overhead to graph the output and generate gifs. All output data was written at every time step to three separate .dat files; position, density, and power. These were then used to plot in MATLAB.

Please see code for further details on implementation.

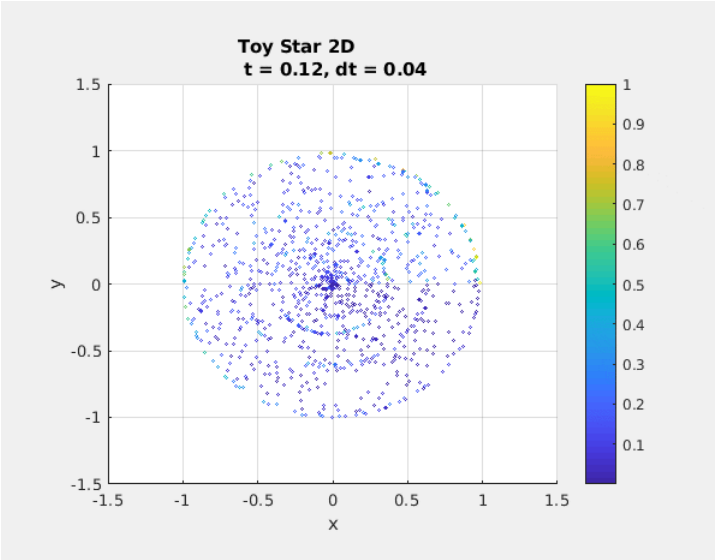


Figure 1: Snapshot of a simulation after a few time steps with power on the color axis.

3 Results and Conclusions

3.1 Serial Results

The gifs of three 1000-particle runs can be found in Google Drive corresponding to runs with/without rotation generated from serial code. Figure 1 is a snapshot a few time steps after the code ran. From the .gifs, it is unclear whether or not rotation plays a noticeable role in the dynamics of the toy star. From the gifs, what may be shock fronts seem to affect the displacement of particles at various radial lengths in the star (more about that later).

3.2 Parallel Results and Speed-up

Although CUDA seemed to speed-up the calculation, the results faulted and the code output non-physical results (Appendix A). The speed-up as compared to serial code can be found in Table 1 which still demonstrates the proficiency of CUDA in large scale calculations. With better implementation, the difference between the CUDA code and serial should become much more drastic and we expect a speed-up of at least 3x. Unfortunately, due to time constraints, we were unable to fix the issue in time but have an idea of where the CUDA code fell through. We plan on fixing the code sometime in the near future with the intention to both cement our understanding of CUDA and potentially use this code for other projects.

Table 1: CUDA vs Serial in seconds with and without writing to files.

Serial w/ write	Serial w/o write	CUDA w/ write	CUDA w/o write
1155	1108	567	709
1180	1113	572	710
1146	1201	560	703
1199	1125	565	709
1176	1102	564	712

4 Further Work and Extensions

As explained above, further work must be done on the CUDA implementation to correct the data being output. Furthermore, the code that calls the CUDA could be optimized to avoid accessing transferring data between the GPU and CPU at every time step which would most likely involve adding a dimension onto some of these arrays so long as the GPU has enough on-board memory. With so many particles, this problem is still ripe for CUDA implementation.

A still-outstanding issue is the balance between the gravitational pull and pressure on the particles inside the star. In the current code, the re-normalization effectively returned particles to within the radius of the star like a spring which acts as a boundary condition. From [2], it should be possible to do without this condition although we found it incredibly difficult to balance the parameters appropriately. Another issue that can be see in the gifs provided above is the shock front development for all stars. According to [3], these shock fronts should be dampened out by rotation which does not appear to be the case. Also according to [3], implementing artificial viscosity is an appropriate means to smooth out the shock fronts and prevent their development which would be

an appropriate addition to this code.

Other extensions include supplementing the code with a better smoothing kernel, better approximations of gravity such as Special or General Relativity, extending to 3D, and implementing a variable smoothing parameter which depends on the position of the particle in the star. In the future, we would like to tackle some of these extensions and apply those to other areas of active research in our research group. It is our hope to be able to use this code as a basis upon which to understand some of the more complex physics inside the core of stars.

This work was supported by a STEM scholarship award funded by the National Science Foundation grant DUE-1259951, PHY-1714068, and the Computational Science Research Center at SDSU.

Thanks to: Dr. Fridolin Weber, Johnny Corbino, CSRC, and SDSU.

References

[1] Y. Eriguchi and E. Muller. "A General Computational Method for Obtaining Equilibria of Self-Gravitating and Rotating Gases". In: *Astronomy and Astrophysics* 146 (1985).

[2] Phillip Mocz. "Smoothed Particle Hydrodynamics: Theory, Implementation, and Application to Toy Stars". In: (Dec. 2011).

[3] J. J. Monaghan. "Smoothed Particle Hydrodynamics". In: *Annual Review of Astronomy and Astrophysics* (1992).

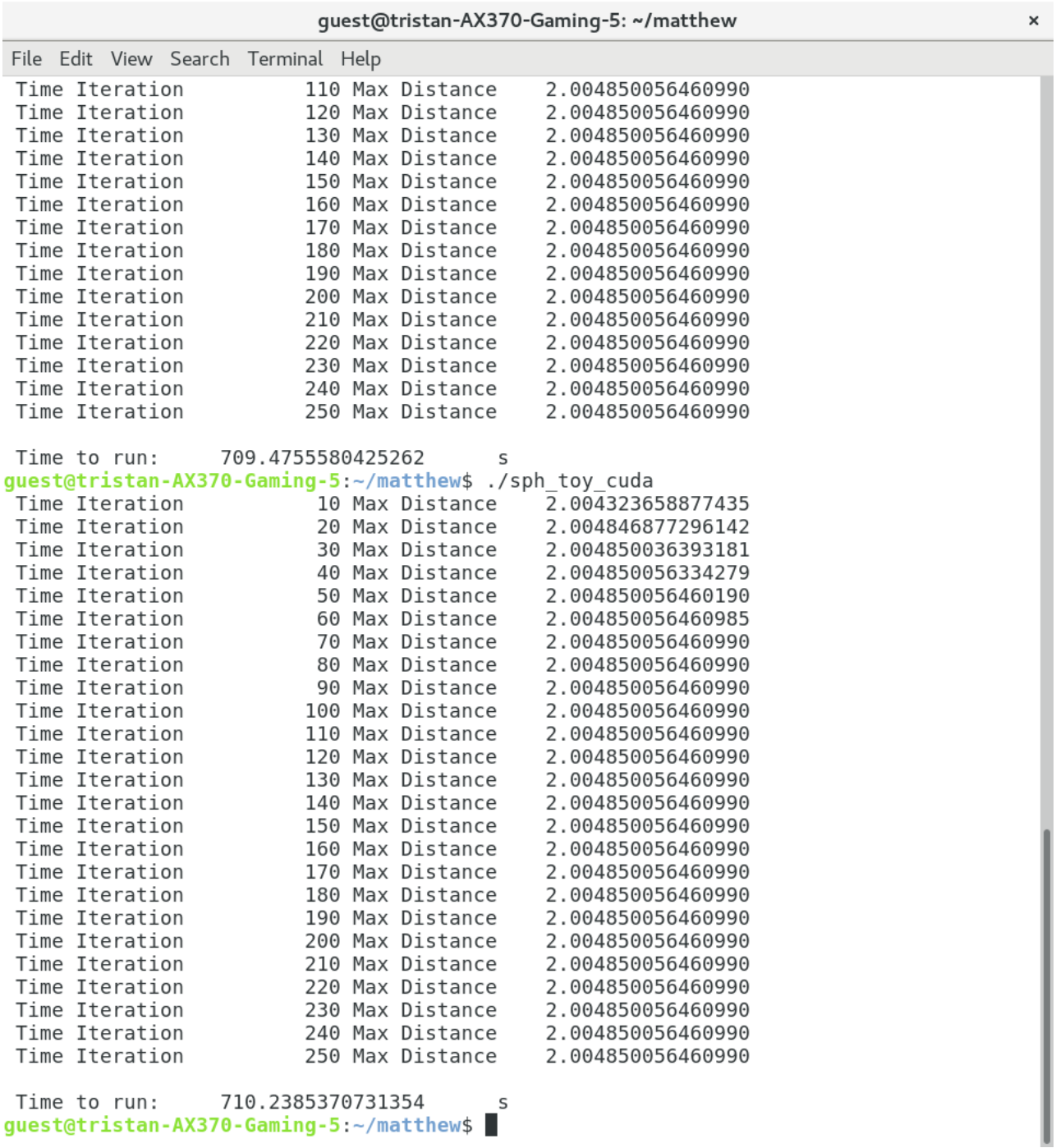
[4] J. J. Monaghan and D. J. Price. "Toy Stars in one dimension". In: *MNRAS* 350 (2004).

[5] J. J. Monaghan and D. J. Price. "Toy Stars in two dimensions". In: *MNRAS* 365 (2005).

[6] S. Rosswog. *SPH Methods in the Modelling of Compact Objects*. 2015.

A Screenshot from Terminal

Figure 2: A screen-shot from running the code with CUDA. The results converge in a way that is non-physical.



For a quick simulation, see Google Drive.