FleCSPH notes

Hyun Lim, Julien Loiseau

August 7, 2018

1 FleCSI

FleCSI is a compile-time configurable framework designed to support multi-physics application development. As such, FleCSI provides a very general set of infrastructure design patterns that can be specialized and extended to suit the needs of a broad variety of solver and data requirements. FleCSI currently supports multi-dimensional mesh topology, geometry, and adjacency information, as well as n-dimensional hashed-tree data structures, graph partitioning interfaces, and dependency closures.
FleCSI introduces a functional programming model with control, execution, and data abstractions that are consistent both with MPI and with state-of-the-art, task-based runtimes such as Legion, HPX and Charm++. The abstraction layer insulates developers from the underlying runtime, while allowing support for multiple runtime systems including conventional models like asynchronous MPI.
The intent is to provide developers with a concrete set of user-friendly programming tools that can be used now, while allowing flexibility in choosing runtime implementations and optimization that can be applied to future architectures and runtimes.
FleCSI's control and execution models provide formal nomenclature for describing poorly understood concepts such as kernels and tasks. FleCSI's data model provides a low-buy-in approach that makes it an attractive option for many application projects, as developers are not locked into particular layouts or data structure representations.
FleCSI currently provides a parallel but not distributed implementation of binary, quad- and octree topologies. Domain decomposition is implemented using space-filling curves, such as Morton ordering curve.
At current stage, FleCSI framework requires implementation of a *driver* and a *specialization_driver*. The role of the *specialization_driver* is to provide the data model and its parallel distribution. Currently, FleCSI does not has this feature fully implemented, so we provide it. The next step will be to incorporate it directly from FleCSPH to FleCSI as we reach a good level of performance. The *driver* represents the general execution of the resolution without worrying of the data locality and communications. As FleCSI is a code in development, its structure may change in the future and we keep track of these changes in FleCSPH.

2 Domain and tree construction

In the previous section we described the FleCSI framework. This part gives details on the domain decomposition using octree, tree construction and search algorithm.

2.1 Domain decomposition

In the current version of FleCSI the domain decomposition is done using the Morton ordering. It allows to describe, sort and distribute particles based on a unique value – key.
In principle, various space-filling curves can be used, with their own advantages and flaws:

- Morton ordering: interlace the bits of X, Y and Z positions to create the key. It is very simple to compute, but the curve suffers discontinuous jumps.
- Hilbert-Peano: interlace the bits, but also add rotations. In this ordering, data locality in memory guarantees locality in space.
- Other space-filling curves: hexagonal space filling curves, ...?

This first implementation is based on the Morton ordering which is used during several steps:

- The distribution part, to be able to split the particles between the processes providing a good locality in the data.
- The tree construction and search.

3 Binary, Quad- and Octrees

4 General algorithm

The main distributed algorithm is presented in algorithm 1

Algorithm 1 Main algorithm

```
 1: procedure SPECIALIZATION_DRIVER(input parameter file p)
 2:     Read parameter file p
 3:     Set simulation parameters
 4:     Distributed read input data files, specified in parameter file
 5:     Set additional parameters specified in input data files
 6:     while iterations do
 7:         Distribute the particles using distributed quick sort          ▷ Using Morton keys
 8:         Compute total range
 9:         Generate the local tree
10:         Share branches
11:         Compute ghosts particles
12:         Update ghosts data
13:         Do physics
14:         Update ghosts data
15:         Do physics
16:         Periodic analysis and output
17:     end while
18: end procedure
```

In the current version the 7 is based on a distributed quick sort algorithm. Each process sends a sample of its keys to the master (or submaster for larger cases). We have set this to 256 Kb of key data per process, but it can be increased for larger simulations. After receiving sample keys, the master determines the general ordering for all the processes and shares the pivots. Then each process sorts its local keys and, in a global communication step, the particles are distributed to the process on which they belong. The advantage is that it is a quick distribution algorithm, but it can lead to bad load balancing.

- The ordering may not be perfect in terms of the number of particles per processes. But by changing the amount of data exchanged to the master can lead to better affectation (?)
- The load balancing also depends on the number of neighbors of each particles. If a particle is located in a poor area with large space between the particles this can lead to bad load balancing too.

After the sorting step the local tree can be created on each process. To be able to look for the ghosts and shared particles we need to share some information with the neighbors on the tree. At line (10), the algorithm searches for the neighboring branches, which are affecting the local particles. We compute the global bounding box of each processes and based on this information each process can then compute the affecting branches to share from its local tree. This new information is then added to the local tree by considering *NON_LOCAL* particles. This data structure does just contain the position and mass of the distant particle.
The branch sharing allows to compute the ghosts for this step. Each process performs a local search in the tree and computes the required ghosts particles (the *NON_LOCAL* bodies). Those data for shared and ghosts are stored and are use to share the complete particle information when 12 is invoked. As the ghosts data remain the same within an iteration, the 12 can be used several times to update local information on remote particles.

5 I/O

Large-scale simulations require an efficient, parallel and distributed I/O. We base this first implementation on HDF5 file structure with H5Part and H5Hut. The I/O was developed at LANL and provides a simple interface to read and write data in H5Part format. The first requirement is to allow the user to work directly with the Paraview visualization tool and splash[1].

6 SPH Formulation in FleCSPH

Smoothed particle hydrodynamics (SPH) is an explicit numerical meshfree method that solves partial differential equations (PDE) of hydrodynamics by discretizing the flow with a set of fluid elements called particle. The main SPH formula to interpolate a quantity $A(\vec{r})$, which is specified by its values on a set of particles $A_b \equiv A(\vec{r}_b)$, is as follows (Rosswog 2009):

$$A(\vec{r}) \simeq \sum_{b\in\Omega(\vec{r})} V_b A_b W(|\vec{r}-\vec{r}_b|, h) \tag{1}$$

where W is a smoothing kernel, h is the smoothing length (hydro interaction range) at a position $\vec{r}$, and V_b is a volume element, usually $V_b = m_b/\rho_b$.
In comparison with Eulerian methods, SPH has several advantages. It can easily adapt to complex geometries, naturally handle low density regions, and does not require low-density floor to handle vacuum. It conserves mass by construction, and can be easily made to conserve linear momentum, angular momentum, and energy up to roundoff. Another advantage of using SPH is its exact advection of fluid properties. Furthermore, the same tree which is used to find particle neighbors, can be employed for computing Newtonian gravitational forces.
Shortcomings of SPH are its convergence which is restricted to low-order, and high sensitivity to the initial particles distribution. Also, SPH struggles with resolving turbulence-dominated flows and requires special care when handling high gradients, such as shocks and stellar surface.
The starting point at a continuum limit is Euler ideal fluid equations in the Lagrangian formulation, expressing conservation equations of mass and linear momentum:

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\vec{v}, \tag{2}$$

$$\frac{du}{dt} = \left(\frac{P}{\rho^2}\right)\frac{d\rho}{dt}, \tag{3}$$

$$\frac{d\vec{v}}{dt} = -\frac{\nabla P}{\rho} + \vec{g}, \tag{4}$$

$$\tag{5}$$

where $d/dt = \partial_t + \vec{v}\cdot\nabla$ and $\vec{g}$ is a gravitational acceleration. The latter may be due to interparticle gravitational interaction, an external gravitational field, or both.

6.1 Basic formulation

In its simplest form, SPH discretization uses the volume element $V_b = m_b/\rho_b$, a constant smoothing length h, and artificial viscosity term Π_{ab}:

$$\rho_a = \sum_b m_b W_{ab}, \tag{6}$$

$$\frac{du_a}{dt} = \sum_b m_b\left(\frac{P_a}{\rho_a^2} + \frac{1}{2}\Pi_{ab}\right)\vec{v}_{ab}\cdot\nabla_a W_{ab}, \tag{7}$$

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b\left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} + \Pi_{ab}\right)\nabla_a W_{ab} + \vec{g}_a, \tag{8}$$

[1]Describe it! url too

where $W_{ab} = W(|\vec{r}_a - \vec{r}_b|, h)$. The viscous stress tensor Π_{ab} may be defined in different ways; currently we have adopted the following form:

$$\Pi_{ab} = \begin{cases} \frac{-\alpha \bar{c}_{ab}\mu_{ab} + \beta\mu_{ab}^2}{\bar{\rho}_{ab}} & \text{for } \vec{r}_{ab}\cdot\vec{v}_{ab} < 0, \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where the following quantities are defined:

$$\mu_{ab} = \frac{\bar{h}_{ab}\vec{r}_{ab}\cdot\vec{v}_{ab}}{|\vec{r}_{ab}|^2 + \epsilon\bar{h}_{ab}^2}, \tag{10}$$

$$\vec{r}_{ab} = \vec{r}_a - \vec{r}_b, \qquad \vec{v}_{ab} = \vec{v}_a - \vec{v}_b, \tag{11}$$

$$\text{average speed of sound:} \qquad \bar{c}_{ab} = (c_a + c_b)/2, \tag{12}$$

$$\text{average density:} \qquad \bar{\rho}_{ab} = (\rho_a + \rho_b)/2, \tag{13}$$

$$\text{average smoothing length:} \qquad \bar{h}_{ab} = (h_a + h_b)/2, \tag{14}$$

$$\tag{15}$$

In the basic formulation with a constant smoothing length, $\bar{h}_{ab} \equiv h$. The values of ϵ, α, and β control the strength and application of artificial viscosity. Their default values: $\epsilon = 0.01$, $\alpha = 1.0$, and $\beta = 2.0$.
The quantity $\bar{c}_{ab}$ is a speed of sound, averaged between particles a and b: $\bar{c}_{ab} = (c_a + c_b)/2$. It is computed as usual:

$$c_a = \sqrt{\left(\frac{\partial P}{\partial \rho}\right)_{S,a}}, \tag{16}$$

where the partial derivative is taken under constraint of a constant entropy S, with thermodynamic conditions at particle a. E.g., the following (Newton-Laplace) equation,

$$c_a = \sqrt{\frac{\Gamma P_a}{\rho_a}} \tag{17}$$

holds both for polytropic and ideal fluid equations of state (see Section **??**).
If $\vec{g} = 0$, then the basic formulation, when integrated with a symplectic integrator, conserves energy, momentum and angular momentum *exactly* (e.g. Rosswog 2009, Section 2.4).

6.2 Kernels

There are many choices for selecting a smoothing kernel. In the code, the kernel choice is specified by a `sph_kernel` parameter, and the smoothing length is defined to be equal to the kernel support radius. Note that some works define smoothing length differently, such that the kernel support radius becomes a multiple of smoothing length. For example, for a cubic spline kernel it is twice the h. We do not make such distinction; all kernels below must satisfy the following normalization condition:

$$\iiint_{S_D(h)} W(\vec{r}, h) d^D\vec{r} = 1, \tag{18}$$

where the integration is performed over the D-dimensional volume of a sphere $S_D(h)$ of radius h.
All the currently implemented kernels possess spherical symmetry, which makes it easy to impose exact conservation of linear momentum. For such kernels, the gradients can be computed as follows:

$$\nabla_a W_{ab} \equiv \nabla_a W(|\vec{r}_a - \vec{r}_b|, h_a) \tag{19}$$

$$= \frac{dW}{dr}\vec{\varepsilon}_{ab}, \tag{20}$$

where $\vec{\varepsilon}_{ab} \equiv \vec{r}_{ab}/|\vec{r}_{ab}|$ is a unit vector in the direction from particle b to particle a. It is therefore sufficient to implement dW/dr for each kernel.
In the formulae below, we define $q \equiv |\vec{r}|/h$.

6.2.1 `sph_kernel = "cubic spline"`

The simplest (but *not* the best one) is a Monaghan's cubic spline kernel:

$$W(\vec{r}, h) = \frac{\sigma_D}{h^D} \begin{cases} 1 - 6q^2 + 6q^3 & \text{if } 0 \le q \le 1/2, \\ 2(1-q)^3 & \text{if } 1/2 \le q \le 1, \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

where $q = r/h$, D is the number of dimensions and σ_D is a normalization constant:

$$\sigma_D = \left\{ \frac{4}{3}, \frac{40}{7\pi}, \frac{8}{\pi} \right\} \text{ in 1D, 2D and 3D resp.} \tag{22}$$

Radial derivative of the cubic spline kernel is:

$$\frac{dW}{dr} = \frac{\sigma_D}{h^{D+1}} \begin{cases} -6q(2-3q) & \text{if } 0 \le q \le 1/2, \\ -6(1-q)^2 & \text{if } 1/2 \le q \le 1, \\ 0 & \text{otherwise.} \end{cases} \tag{23}$$

7 Applications

7.1 Sod Shock Tube

The Sod shock tube is the test consists of a one-dimensional Riemann problem with the following initial parameters

$$(\rho, v, p)_{t=0} = \begin{cases} (1.0, 0.0, 1.0) & \text{if} 0 < x \le 0.5 \\ (0.125, 0.0, 0.1) & \text{if} 0.5 < x < 1.0 \end{cases} \tag{24}$$

This link shows some references and values for Sod shock tube problem that also includes boundary and jump conditions(`http://www.phys.lsu.edu/~tohline/PHYS7412/sod.html`).
Also, we would like to re-generate the shock test result from Rosswog's paper. In that paper, he shows the result of a 2D relativistic shock tube test where the left state is given by $[P, v_x, v_y, N]_L = [40/3, 0, 0, 10]$ and the right state by $[P, v_x, v_y, N]_R = [10^{-6}, 0, 0, 1]$ with $\Gamma = 5/3$
In our code, we use below parameters to get results

7.2 Sedov Blast Wave

A blast wave is the pressure and flow resulting from the deposition of a large amount of energy in a small very localized volume. This is another great test problem for computational fluid dynamics field.
There are different version of blast wave test but we consider the analytic solution for a point explosion is given by Sedov, making the assumption that the atmospheric pressure relative to the pressure insider the explosion negligible. The position of the shock as a function of time t, relative to the initiation of the explosion, is given by

$$R(t) = \left(\frac{et^2}{\rho_0} \right)^{\frac{1}{\delta+2}} \tag{25}$$

with $\delta = 2$ and $\delta = 3$ for cylindrical and spherical geometry respectively. The initial density ρ_0 whereas e is a dimensionless energy. Right behind the shock we ahve the following properties

$$\rho_2 = \frac{\Gamma+1}{\Gamma-1}\rho_0 P_2 = \frac{2}{\Gamma+1}\rho_0 w^2 v_2 = \frac{2}{\Gamma+1} w \tag{26}$$

where the shock velocity is

$$w(t) = \frac{dR}{dt} = \frac{2}{\delta+2}\frac{R(t)}{t} \tag{27}$$

In numerical simulations, energy deposition in a single point is difficult to achieve. A solution to the problem is to make use of the bursting balloon analogue. Rather than depositing the total energy in a single point, the energy is released into a balloon of finite volume V

$$e = \frac{(P - P_0)V}{\Gamma - 1} \tag{28}$$

The energy release in a balloon of radius r_0 raises the pressure to the value

$$P = \frac{3(\Gamma - 1)e}{(\delta + 1)\pi r_0^{\delta}} \tag{29}$$

Here, we test 2D blast wave test. In this simulation, we use ideal gas EOS with $\Gamma = 5/3$ and we are assuming that the undistributed area is at rest with a pressure $P_0 = 1.0^{-5}$. The density is constant ρ_0, also in the pressurized region.

7.3 Equations of State

To understand the inner property of stars, one needs to find the equation which describes the relation between the pressure of matter and its density, temperature and other compositions such that

$$P = P(\rho, T, Y_e, ...) \tag{30}$$

First, we consider analytic equations of state that are relevant for binary neutron stars

7.3.1 Ideal Gas

Ideal gas equation of state is

$$P(\rho, u) = (\Gamma - 1)\rho u \tag{31}$$

where Γ is the adiabatic index of the gas. For a monatomic gas, we set $\Gamma = 5/3$. For another test such as sod tube, people use $\Gamma = 1.4$

7.3.2 Piecewise Polytrope

For more relevant simulation, we choose piecewise polytropic EOS.(Ideal gas EOS is still good for many simple test cases like Sod tube) In our case, we assume constant entropy so that many thermodynamic situations can be approximated as polytropes or piecewise functions made up of polytropes.
For neutron star case, we assume degenerated Fermi gas of neutrons then polytropic constant for a non-relativistic degenerated neutron gas is

$$K_0 = \frac{(3\pi^2)^{2/3}\hbar^2}{5m_n^{8/3}} \tag{32}$$

where m_n is the mass of a proton and $\hbar$ is a Planck constant. For polytropic index, we set

$$\gamma_0 = \frac{5}{3} \tag{33}$$

In the relativistic case,

$$\gamma_1 = \frac{5}{2} \tag{34}$$

Then, piecewise polytrope EOS is

$$P(\rho) = \begin{cases} K_0\rho^{\gamma_0} & \text{if} \rho \leq \rho_0 \\ \frac{K_0\rho_0^{\gamma_0}}{\rho_0^{\gamma_1}}\rho^{\gamma_1} & \text{if} \rho > \rho_0 \end{cases} \tag{35}$$

where $\rho_0 = 5 \times 10^{14} g/cm^3$. We can combine the piecewise polytropic EOS with ideal gas to attain an EOS valid at both low and high densities. For more realistic studies, we need to consider different types of analytic EOSs such as Maxwell-Boltzmann and Helmholtz EOSs. Also, we will put the functionality that can control tabulated EOS.

7.3.3 Zero Temperature Equations of State

Another interesting problem using SPH is the double white dwarf (DWD) simulations for studying possible progenitors to type Ia supernovae. Here, we use zero temperature equations of state (ZTWD) as a variation of the self consistent field technique. In ZTWD, the electron degeneracy pressure P varies with the mass density ρ according to the relation

$$P = A\left[x(2x^2-3)(x^2+1)^{1/2} + 3\sinh^{-1} x\right] \tag{36}$$

where the dimensionless parameter

$$x \equiv \left(\frac{\rho}{B}\right)^{1/3} \tag{37}$$

and the constant A and B are

$$A \equiv \frac{\pi m_e^4 c^5}{3h^3} = 6.00288 \times 10^{22}\,\mathrm{dynes\,cm^{-2}} \tag{38}$$

$$\frac{B}{\mu_e} \equiv \frac{8\pi m_p}{3}\left(\frac{m_e c}{h}\right)^3 = 9.81011 \times 10^5\,\mathrm{g\,cm^{-3}} \tag{39}$$

8 Time Integration Scheme

8.1 Leap-Frog Method

Leap-frog time integration is frequently used in particle simulation. The name comes that the velocities are updated on half steps and the positions on integer steps so the two leap over each other. After computing accelerations, one step takes the form

$$v^{i+1/2} = v^{i-1/2} + a^i \Delta t \tag{40}$$

$$r^{i+1} = r^i + v^{i+1/2}\Delta t \tag{41}$$

For v^{i+1}, a common approximation in SPH is to assume that the velocity at the current time plays a minor role in the computation of the acceleration (i.e. velocity changes are small with each time step) and then the following approximation can be made

$$v^{i+1} = \frac{1}{2}(v^{i-1/2} + v^{i+1/2}) \tag{42}$$

At the first step, we only have initial velocity so we need to follow below routing

$$v^{1/2} = v^0 + a^0 \Delta t/2 \tag{43}$$

$$r^1 = r^0 + v^{1/2}\Delta t \tag{44}$$

The time step is adaptive and determined with $\Delta t = Min(\Delta t_1, \Delta t_2)$:

$$\Delta t_1 = k\mathrm{Min}_i\left(\frac{h_i}{c_i + 1.2\alpha c_i + 1.2\beta \mathrm{Max}_j \mu_{ij}}\right) \text{ with } k \approx 0.1$$

$$\Delta t_2 = \mathrm{Min}_i \sqrt{\left(\frac{h_i}{|\dot{\vec{v_i}}|}\right)}$$

9 Solving Lane-Emden Equation

We need to determine the density function based on the radius.
As we consider the star as a polytropic fluid, we use the equation of Lane-Emden which is a form of the Poisson equation:

$$\frac{d^2\theta}{d\xi^2} + \frac{2}{\xi}\frac{d\theta}{d\xi} + \theta^n = 0 \tag{45}$$

With ξ and θ two dimensionless variables. There is only exact solutions for a polytropic index $n = 0.5$, 1 and 2. In our work we use a polytropic index of 1 which can correspond to a NS simulation.
For $n = 1$ the solution of equation 38 is:

$$\theta(\xi) = \frac{sin(\xi)}{\xi} \tag{46}$$

We note $\xi_1 = \pi$, the first value of ξ as $\theta(\xi) = 0$. $\theta(\xi)$ is also defined as:

$$\theta(\xi) = \left(\frac{\rho(\xi)}{\rho_c}\right)^{\frac{1}{n}} = \frac{\rho(\xi)}{\rho_c} \tag{47}$$

With ρ_c the internal density of the star and ρ the density at a determined radius. ξ is defined as:

$$\xi = Ar = \sqrt{\frac{4\pi G}{K(n+1)}\rho_c^{(n-1)/n}} \times r = \sqrt{\frac{2\pi G}{K}} \times r \text{ (for } n = 1)$$

With K a proportionality constant.
From the previous equations we can write the stellar radius R as:

$$R = \sqrt{\frac{K(n+1)}{4\pi G}}\rho_c^{(1-n)/2}\xi_1 = \sqrt{\frac{K}{2\pi G}} \times \xi_1 \tag{48}$$

(We note that for $n = 1$ the radius does not depend of the central density.)
If, for example, we use dimensionless units as $G = R = M = 1$ (for the other results we use CGS with $G = 6.674 \times 10^{-8} cm^3 g^{-1} s^{-2}$) We can compute K as:

$$K = \frac{R^2 2\pi G}{\xi_1^2} \tag{49}$$

	NS_1	NS_2	NS_3	NS_4
Radius (cm)	$R = G = M = 1$	1500000	1400000	960000
K	0.636619	95598.00	83576.48	39156.94

Then we deduce the density function of r as :

$$\rho(\xi) = \frac{sin(A \times r)}{A \times r} \times \rho_c \text{ with } A = \sqrt{\frac{2\pi G}{K}}$$

As we know the total Mass M, the radius R and the gravitational constant G we can compute the central density as:

$$\rho_c = \frac{MA^3}{4\pi(sin(AR) - ARcos(AR))}$$

Then we normalize the results to fit $R = M = G = 1$: $K' = K/(R^2 G)$, $m_i' = m_i/M$, $h_i' = h_i/R$, $\vec{x_i}' = \vec{x_i}/R$

9.1 Gravitational force

The self-gravity $\vec{F_i}^{Grav}$ for each particles:

$$\vec{F_i}^{Grav} = \sum_j G\frac{m_i m_j}{(|\vec{r_j} - \vec{r_i}|)^3}\vec{r_{ij}} \tag{50}$$

In this part we will need Fast Multiple Method (FMM) for the computation to avoid $O(N^2)$ complexity.

10 Resolution order

The resolution is done in this order:

```
Load data from file
while TotalTime not reached do
    Move particles
    Apply rotation with defined angular velocity
    Compute density $\rho$
    Compute pressure $P$ and sound speed $c$
    Compute $\vec{F}_{hydro}$
    Compute $\vec{F}_{grav}$
    if relaxation then
        Compute $\vec{F}_{roche}$ or $\vec{F}_{rot}$
    end if
    Compute acceleration $\vec{a}$ with the equation of motion
    Compute $\Delta t$
    Compute new velocity from new acceleration
    if output step then
        output data to file
    end if
    TotalTime $\leftarrow$ TotalTime + $\Delta t$
end while
```

11 Hydrostatic Equilibrium

The initial data are based on a cubic lattice within a sphere of radius R. The density function, based on radius, $\rho(\vec{r})$ is known using the result of the Lane-Emden equation (we use polytropic index $n = 1$ here). The mass associate to each particle i of the total N particles:

$$m_i = \frac{\rho(\vec{r_i})}{n_r} \text{ with } n_r = \frac{3N}{4\pi R^3}$$

The smoothing length is define constant and the same for all particles for all the simulation:

$$h = \frac{1}{2}\sqrt{\frac{3N_N}{4\pi n}}$$

Here we choose N_N, the average number of neighbors, to be 100.

11.1 Roche lobe problem

To create Hydrostatic Equilibrium Models we use a different equation of motion. This version use Roche Lobe:

$$\frac{d\vec{v_i}}{dt} = \frac{\vec{F}_i^{Grav}}{m_i} + \frac{\vec{F}_i^{Hydro}}{m_i} + \vec{F}_i^{Roche} - \frac{\vec{v_i}}{t_{relax}} \tag{51}$$

With $t_{relax} \leq t_{osc} \sim (G\rho)^{-1/2}$ and where $\vec{F}_i^{Roche}$ is:

$$\vec{F}_i^{Roche} = \mu(3+q)x_i\hat{\vec{x}} + \mu q y_i \hat{\vec{y}} - \mu z_i \hat{\vec{z}}$$

With μ to be determined (for us $\mu = 0.069$) and $q = \frac{M'}{M} = 1$ as the two polytropes have the same total mass. This is apply to each star to get the equilibrium and the simulate the tidal effect.

11.2 Darwin problem

This is the way we use to generate the final simulation. The equation of motion for the relaxation is now:

$$\frac{d\vec{v_i}}{dt} = \frac{\vec{F}_i^{Grav}}{m_i} + \frac{\vec{F}_i^{Hydro}}{m_i} + \vec{F}_i^{Rot} - \frac{\vec{v_i}}{t_{relax}} \tag{52}$$

With t_{relax} same as before and $\vec{F}_{Rot}$ defined by:

$$\vec{F}_{Rot} = \Omega^2(x_i\vec{x} + y_i\vec{y}) \tag{53}$$

With $\Omega = \sqrt{\frac{G(M+M')}{a^3}}$.
Or $L_z = Q_{zz}\Omega$ and $Q_{zz} = \sum_i(x_i^2 + y_i^2)$. At $t = 0$ we compute the total angular moment L_z which stay constant. Using it during the relaxation we can compute Ω as: $\Omega = \frac{L_z}{Q_{zz}}$ just by recomputing Q_{zz}.
Here the scheme is in N^2 but just for the relaxation step.
For this relaxation we use two stars generated as before, applying equation of motion 45. Using a as the distance between the two polytropes (Here $a = 2.9$ for $R = 1$) and $\vec{x}$ going for the center of the first to the second star, and $\vec{z}$ is like the rotation vector.

12 Gravitation computation

The algorithm is presented in algorithm 2. In this algorithm, the *macangle* is the angle of the Multipole Acceptance Criterion. The acceleration is, for a center of mass c, the sum of contributions from the local particles and the distant cells with:

$$\vec{f_c}(\vec{r_c}) = -\sum_p \frac{m_p.(\vec{r_c} - \vec{r_p})}{|\vec{r_c} - \vec{r_p}|^3} \tag{54}$$

With p the particle inside this cell and *cell* the cells that are accepted with the MAC. Here we directly consider the gravitational acceleration, we don't take in account the mass of the center of mass c. And $G = 1$ in our context.
The acceleration at a point from this center of mass is based on taylor series:

$$\vec{f}(\vec{r}) = \vec{f_c}(\vec{r_c}) + ||\frac{\partial\vec{f_c}}{\partial\vec{r_c}}|| \cdot (\vec{r} - \vec{r_c}) + \frac{1}{2}(\vec{r} - \vec{r_c})^\intercal \cdot ||\frac{\partial\vec{f_c}}{\partial\vec{r_c}\partial\vec{r_c}}|| \cdot (\vec{r} - \vec{r_c}) \tag{55}$$

The Jacobi matrix $||\frac{\partial\vec{f_c}}{\partial\vec{r}}||$ is then:

$$-\sum_p \frac{m_p}{|\vec{r_c} - \vec{r_p}|^3}\begin{bmatrix} 1 - \frac{3(x_c-x_p)(x_c-x_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(y_c-y_p)(x_c-x_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(z_c-z_p)(x_c-x_p)}{|\vec{r_c}-\vec{r_p}|^2} \\ -\frac{3(x_c-x_p)(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} & 1 - \frac{3(y_c-y_p)(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(z_c-z_p)(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} \\ -\frac{3(x_c-x_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & -\frac{3(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & 1 - \frac{3(z_c-z_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} \end{bmatrix} \tag{56}$$

$$||\frac{\partial f_c^i}{\partial r_c^j}|| = -\sum_p \frac{m_p}{|\vec{r_c} - \vec{r_p}|^3}\left[\delta_{ij} - \frac{3.(r_c^i - r_p^i)(r_c^j - r_p^j)}{|\vec{r_c} - \vec{r_p}|^2}\right] \tag{57}$$

With δ_{ij} the identity matrix with $\delta_{ij} = 1$ if $i = j$ where i, j runs spatial index from 1 to 3. For example, $r^1 = x$, $r^2 = y$, and $r^3 = z$ as usual sense. (We do not consider covariant form of this because we are not considering spacetime).
The Hessian matrix $||\frac{\partial\vec{f_c}}{\partial\vec{r_c}\partial\vec{r_c}}||$ is then:

$$||\frac{\partial^2 f_c^x}{\partial r_c^i \partial r_c^j}|| = -\sum_p \frac{3m_p}{|\vec{r_c} - \vec{r_p}|^5}\begin{bmatrix} \frac{5(x_c-x_p)^3}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) & \frac{5(x_c-x_p)^2(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) & \frac{5(x_c-x_p)^2(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_c - z_p) \\ \frac{5(x_c-x_p)^2(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) & \frac{5(x_c-x_p)(y_c-y_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) & \frac{5(x_c-x_p)(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} \\ \frac{5(x_c-x_p)^2(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_c - z_p) & \frac{5(x_c-x_p)(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & \frac{5(x_c-x_p)(z_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) \end{bmatrix} \tag{58}$$

$$||\frac{\partial^2 f_c^y}{\partial r_c^i \partial r_c^j}|| = -\sum_p \frac{3m_p}{|\vec{r_c} - \vec{r_p}|^5} \begin{bmatrix} \frac{5(x_c-x_p)^2(y_c-y_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) & \frac{5(x_c-x_p)(y_c-y_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) & \frac{5(x_c-x_p)(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} \\ \frac{5(x_c-x_p)(y_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) & \frac{5(y_c-y_p)^3}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) & \frac{5(y_c-y_p)^2(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_c - z_p) \\ \frac{5(x_c-x_p)(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & \frac{5(y_c-y_p)^2(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_c - z_p) & \frac{5(y_c-y_p)(z_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) \end{bmatrix} \tag{59}$$

$$||\frac{\partial f_c^z}{\partial r_c^i \partial r_c^j}|| = -\sum_p \frac{3m_p}{|\vec{r_c} - \vec{r_p}|^5} \begin{bmatrix} \frac{5(x_c-x_p)^2(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_c - z_p) & \frac{5(x_c-x_p)(y_c-y_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & \frac{5(x_c-x_p)(z_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) \\ \frac{5(x_c-x_p)(y_c-z_p)(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} & \frac{5(y_c-y_p)^2(z_c-z_p)}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_r - z_p) & \frac{5(y_c-y_p)(z_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) \\ \frac{5(x_c-x_p)(z_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(x_c - x_p) & \frac{5(y_c-y_p)(z_c-z_p)^2}{|\vec{r_c}-\vec{r_p}|^2} - 3(y_c - y_p) & \frac{5(z_c-z_p)^3}{|\vec{r_c}-\vec{r_p}|^2} - 3(z_c - z_p) \end{bmatrix} \tag{60}$$

$$||\frac{\partial^2 f_c^i}{\partial r_c^j \partial r_c^k}|| = -\sum_p \frac{3m_p}{|\vec{r_c} - \vec{r_p}|^5} \left[\frac{5(r_c^i - r_p^i)(r_c^j - r_p^j)(r_c^k - r_p^k)}{|\vec{r_c} - \vec{r_p}|^2} - \frac{3}{w} \left(\delta_{ij}(r_c^k - r_p^k) + \delta_{jk}(r_c^i - r_p^i) + \delta_{ik}(r_c^j - r_p^j) \right) \right] \tag{61}$$

where $w = \delta_{ij} + \delta_{jk} + \delta_{jk} + \epsilon_{ijk}$ and ϵ_{ijk} is 3D Levi-Civita symbol. We add Levi-Civita symbol to avoid zero in denominator Here again, latin indices i, j, and k indicates spatial components

Algorithm 2 Gravitation computation

```
1: procedure TREE_TRAVERSAL_GRAV(branch $sink$)
2:     if $sink.mass < M_{cellmax}$ then                ▷ Another choice criterion can be use
3:         $\vec{f_c} \leftarrow \vec{0}$
4:         $\frac{\partial \vec{f_c}}{\partial \vec{r}} \leftarrow \vec{0}$
5:         TREE_TRAVERSAL_C2C($sink$,$tree.root$,$\vec{f_c}$, $\frac{\partial \vec{f_c}}{\partial \vec{r}}$)    ▷ Compute $\vec{f_c}$ and $\frac{\partial \vec{f_c}}{\partial \vec{r}}$ using MAC
6:         SINK_TRAVERSAL_C2P($sink$, $\vec{f_c}$, $\frac{\partial \vec{f_c}}{\partial \vec{r}}$)    ▷ Expand to the particles below
7:     else
8:         for All children $c$ of $sink$ do
9:             TREE_TRAVERSAL_GRAV($c$)
10:        end for
11:    end if
12: end procedure
13:
14: function MAC(branch $sink$,branch $source$,double $macangle$)
15:     $d_{max} \leftarrow source.radius \times 2$
16:     $dist \leftarrow distance(sink.position, source.position)$
17:     return $d_{max}/dist < macangle$
18: end function
19:
20: procedure TREE_TRAVERSAL_C2C(branch $sink$, branch $source$, acceleration $\vec{f_c}$)
21:     if MAC($sink, source, macangle$) then
22:         $\vec{f_c} \leftarrow \vec{f_c} + (-\frac{source.mass \times (sink.position - source.position)}{|sink.position - source.position|^3})$
23:         $\frac{\partial \vec{f_c}}{\partial \vec{r}} \leftarrow ...$
24:     else
25:         if $source.is\_leaf()$ then
26:             for All particles $p$ of $source$ do
27:                 $\vec{f_c} \leftarrow \vec{f_c} + (-\frac{p.mass \times (sink.position - p.position)}{|sink.position - p.position|^3})$
28:                 $\frac{\partial \vec{f_c}}{\partial \vec{r}} \leftarrow ...$
29:             end for
30:         else
31:             for All children $c$ of $source$ do
32:                 TREE_TRAVERSAL_C2C($sink, c, \vec{f_c}$)
33:             end for
34:         end if
35:     end if
36: end procedure
37:
38: procedure TREE_TRAVERSAL_C2P(branch $current$, acceleration $\vec{f_c}$)
39:     if $current.is\_leaf()$ then
40:         for All particle $p$ of $current$ do
41:             $p.grav \leftarrow \vec{f_c} + \frac{\delta \vec{f_c}}{\delta current.position}.(p.position - current.position) + ...$
42:         end for
43:     else
44:         for All children $c$ of $current$ do
45:             TREE_TRAVERSAL_C2P($c$, $\vec{f_c}$, $\frac{\partial \vec{f_c}}{\partial \vec{r_c}}$)
46:         end for
47:     end if
48: end procedure
```