Selected Methods in N-body Simulations

Aleksy Bałaziński

June 5, 2025

Contents

1 Introduction

The dominant force over large distances is the gravitational force. The force exerted on a body with mass m_2 at the point $\mathbf{x}_2$ by a body with mass m_1 located at $\mathbf{x}_1$ can be expressed by the relation

$$\mathbf{F} = -G\frac{m_1 m_2}{|\mathbf{x}_{21}|^3}\mathbf{x}_{21} \tag{1}$$

where G is the gravitational constant $6.674 \times 10^{-11}\mathrm{m}^3\mathrm{kg}^{-1}\mathrm{s}^{-2}$ and $\mathbf{x}_{21} = \mathbf{x}_2 - \mathbf{x}_1$. Therefore, the evolution of a system of N bodies is described by N equations

$$\ddot{\mathbf{x}}_i = -G\sum_{j\neq i}\frac{m_j}{|\mathbf{x}_{ij}|^3}\mathbf{x}_{ij}. \tag{2}$$

for each $i = 1, \ldots, N$. Direct application of Equation 2 is the basis of the so-called *particle-particle* method. The method is characterized by $O(N^2)$ time complexity (more precisely, it requires $(N-1)N/2$ operations if Newton's 3rd law is used in the computation). Assuming that 100ns are required to perform the floating-point operations under the summation symbol, $N = 30,000$, and 150 iterations, the simulation would take approximately 2 hours to complete. Therefore, it is evident that more efficient algorithms are needed to make simulations of this scale feasible.

The *particle-mesh* (PM) technique, introduced around 1985 by Hockney and Eastwood, was an early improvement over the PP method. In the PM approach, the space is divided into a rectangular grid (or mesh) of cells. Each cell is assigned a portion of the mass of nearby particles, creating a density distribution $\rho(\mathbf{x})$. The relation between the density and gravitational potential ϕ, in the form of Poisson's equation

$$\nabla^2\phi = 4\pi G\rho, \tag{3}$$

is then used to obtain the potential at each cell center. The gravitational field $\mathbf{g}$ can then be calculated as $\mathbf{g} = -\nabla\phi$. Since $\mathbf{g}$ equals the acceleration due to gravity, we get $\ddot{\mathbf{x}}_i = \mathbf{g}(\mathbf{x}_i)$.

The drawback of the PM method is its poor modeling of forces over short distances. Eastwood and Hockney proposed a remedy for this problem: the *particle-particle-particle-mesh* method (or P^3M in short). In the P^3M method, the force on the i-th particle is split into two components: *short-range* and *long-range* force. The long-range force is calculated using the PM method, whereas the short-range force can be found by direct summation of the forces due to nearby particles.

The computational complexity of the PM and P^3M methods depends on the implementation of the potential solver used to calculate ϕ from Equation 3. For instance, if a fast Fourier transform is used, then the complexity of the PM algorithm is $O(N + N_g^3 \log N_g)$, where N_g is the number of cells in a single dimension of the grid (note it is linear in N). For the P^3M method, the worst-case scenario happens when all particles are clustered closely together, which causes the short-range $O(N^2)$ correction part to become dominant.

2 Particle-mesh method

The particle-mesh method can be described as the following sequence of four steps:

1. Assign masses to mesh points,
2. Solve the field equation (Equation 3) on the mesh,
3. Calculate the field strength at mesh-points,
4. Find forces applied to individual particles by interpolation.

In this section, each of these steps will be described in more detail.

2.1 Mass assignment

The specifics of assigning mass from particles to mesh points depend on the density profile (or *shape*) associated with the particles. In general, the particles need not be represented as idealized dimensionless points; indeed, it is possible to construct a hierarchy of shapes, where each successive member covers a larger number of mesh points and whose application leads to smaller numerical errors.

An infinite hierarchy of shapes with this property, as described by Hockney and Eastwood in [7], can be generated by successive convolutions with the "top-hat" function Π, defined as

$$\Pi(x) = \begin{cases} 1, & |x| < \frac{1}{2} \\ \frac{1}{2}, & |x| = 1 \\ 0, & \text{otherwise.} \end{cases}$$

The three most popular assignment schemes that hail from this family (and the ones implemented in our program) are the *nearest grid point* (NGP), *cloud in cell* (CIC), and *triangular shaped cloud* (TSC)

schemes, with shapes S given by

$$S_{\mathrm{NGP}} = \delta(x), \quad S_{\mathrm{CIC}} = \delta(x) * \frac{1}{H}\Pi\left(\frac{x}{H}\right) = \frac{1}{H}\Pi\left(\frac{x}{H}\right), \quad S_{\mathrm{TSC}} = \frac{1}{H}\Pi\left(\frac{x}{H}\right) * \frac{1}{H}\Pi\left(\frac{x}{H}\right) = \frac{1}{H}\Lambda\left(\frac{x}{H}\right),$$

where Λ is the triangle function

$$\Lambda(x) = \begin{cases} 1 - |x|, & |x| < 1 \\ 0, & \text{otherwise.} \end{cases}$$

For illustrative purposes, the shape S_{CIC} is depicted in Figure 1.

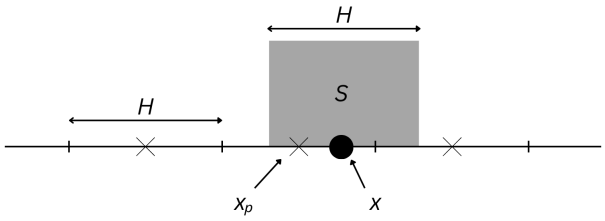


Figure 1: The CIC shape centered at x (particle position). The particle is within cell x_p, however, the cell x_{p+1} gets non-zero density contribution from the particle.

In the one-dimensional case, the fraction of mass W_p assigned to mesh-point p from particle at position x is given by

$$W(x - x_p) = W_p(x) = \int_{x_p - H/2}^{x_p + H/2} S(x' - x)dx'.$$

A simple rule for relating the assignment function W defined above with shape S can be found by noticing that

$$W(x) = \int_{-H/2}^{H/2} S(x' - x)dx' = \int_{-\infty}^{\infty} \Pi\left(\frac{x'}{H}\right) S(x' - x)dx' = \Pi\left(\frac{x}{H}\right) * S(x).$$

This implies that

$$W_{\mathrm{NGP}}(x) = \Pi\left(\frac{x}{H}\right), \quad W_{\mathrm{CIC}}(x) = \Lambda\left(\frac{x}{H}\right), \quad W_{\mathrm{TSC}}(x) = \Pi\left(\frac{x}{H}\right) * \frac{1}{H}\Lambda\left(\frac{x}{H}\right) = (\Pi * \Lambda)\left(\frac{x}{H}\right). \quad (4)$$

Splitting the domain of integration in the expression for W_{TSC} into five disjoint intervals shows that

$$(\Pi * \Lambda)(x) = \begin{cases} \frac{1}{8}(3 - 2|x|)^2, & \frac{1}{2} \le |x| < \frac{3}{2} \\ \frac{3}{4} - x^2, & |x| < \frac{1}{2} \\ 0, & \text{otherwise.} \end{cases}$$

Two- and three-dimensional versions of the assignment functions in Equation 4 are products of the assignment functions in each dimension. For example, the three-dimensional assignment function W is

$$W(\mathbf{x}) = W(x)W(y)W(z).$$

Hence, the mass assigned at mesh-point at $\mathbf{x_p}$ is

$$m(\mathbf{x_p}) = \sum_i m_i W_{\mathbf{p}}(\mathbf{x}_i),$$

or, in terms of density ρ,

$$\rho(\mathbf{x_p}) = \frac{1}{V}\sum_i m_i W_{\mathbf{p}}(\mathbf{x}_i), \tag{5}$$

where $V = H^3$ is the volume of a cell and i indexes the particles.

Obviously, Equation 5 is not suitable for direct application in the actual algorithm. Instead, we iterate over all particles, identify the parent cell $\mathbf{p}$ of each particle (and its neighborhood) and update ρ. This process is illustrated in Algorithm 1. The set $\mathcal{C}_S(\mathbf{x}_i)$ of cells that have to be considered while

Algorithm 1 Density assignment algorithm

1: **for each** particle i **do**
2: **for each** cell $\mathbf{q}$ in $\mathcal{C}_S(\mathbf{x}_i)$ **do**
3: $\rho(\mathbf{x_q}) \leftarrow \rho(\mathbf{x_q}) + m_i W(\mathbf{x}_i - \mathbf{x_q})/V$

assigning density from the i-th particle, depends on the shape S of the particle. Specifically, we have $\mathcal{C}_{\mathrm{NGP}}(\mathbf{x}) = \{[\mathbf{x}/H]\}$, $\mathcal{C}_{\mathrm{CIC}}(\mathbf{x}) = \{\lfloor \mathbf{x}/H \rfloor + \mathbf{t} \mid t_i = 0, 1\}$, and $\mathcal{C}_{\mathrm{TSC}}(\mathbf{x}) = \{[\mathbf{x}/H] + \mathbf{t} \mid t_i = -1, 0, 1\}$. It follows that $|\mathcal{C}_{\mathrm{NGP}}(\mathbf{x})| = 1$, $|\mathcal{C}_{\mathrm{CIC}}(\mathbf{x})| = 8$, and $|\mathcal{C}_{\mathrm{TSC}}(\mathbf{x})| = 27$ which illustrates the increasing computational cost resulting from using higher-order assignment schemes. We note that Algorithm 1 can be parallelized if atomic increments are used in line 3.

2.2 Solving the field equation

The Poisson equation (Equation 3) can be restated in integral form

$$\phi(\mathbf{x}) = \int \mathcal{G}(\mathbf{x} - \mathbf{x}')\rho(\mathbf{x}')dV',$$

which has the following discrete analogue

$$\phi(\mathbf{x_p}) = V\sum_{\mathbf{p}'} \mathcal{G}(\mathbf{x_p} - \mathbf{x_{p'}})\rho(\mathbf{x_{p'}}), \tag{6}$$

where $\mathcal{G}$ is the Green's function (potential due to unit mass). The right-hand side of Equation 6 is a convolution sum that runs over a finite set of mesh points. If we assume periodic boundary conditions, we can apply the discrete Fourier transform to both sides and use the convolution theorem to conclude that[1]

$$\hat{\phi}(\mathbf{k}) = \hat{\mathcal{G}}(\mathbf{k})\hat{\rho}(\mathbf{k}). \tag{8}$$

[1] In this work, the Hockney & Eastwood definition of DFT is used, i.e.

$$D(x_p) = \frac{1}{L}\sum_{l=0}^{N-1} \hat{D}(k)e^{ikx_p}, \quad \hat{D}(k) = H\sum_{p=0}^{N-1} D(x_p)e^{-ikx_p},$$

where $x_p = pH$. The conversion between this form and another popular definition,

$$\widetilde{D_H}(k) = \sum_{p=0}^{N-1} D_H(p)e^{-i2\pi kp/N}, \tag{7}$$

is given by

$$\widetilde{D_H}(k) = \frac{1}{H}\hat{D}\left(\frac{2\pi}{NH}k\right),$$

where $D_H(p) = D(pH)$.

An approximation to $\hat{\mathcal{G}}$ can be found using a discretized version of the Laplacian in Equation 6. Specifically, for a 7-point stencil,

$$\begin{aligned} 4\pi G\rho(\mathbf{x}_{ijk}) = &\frac{\phi(\mathbf{x}_{i-1,j,k}) - 2\phi(\mathbf{x}_{ijk}) + \phi(\mathbf{x}_{i+1,j,k})}{H^2} \\ &+ \frac{\phi(\mathbf{x}_{i,j-1,k}) - 2\phi(\mathbf{x}_{ijk}) + \phi(\mathbf{x}_{i,j+1,k})}{H^2} \\ &+ \frac{\phi(\mathbf{x}_{i,j,k-1}) - 2\phi(\mathbf{x}_{ijk}) + \phi(\mathbf{x}_{i,j,k+1})}{H^2}. \end{aligned}$$

Applying the discrete Fourier transform to both sides and using the shift theorem we get

$$\begin{aligned} 4\pi G\hat{\rho}(\mathbf{k}) &= \frac{1}{H^2}\sum_{i=1}^{3}\left(e^{-iHk_i} + e^{iHk_i} - 2\right)\hat{\phi}(\mathbf{k}) \\ &= \frac{1}{H^2}\sum_{i=1}^{3}\left(e^{iHk_i/2} - e^{-iHk_i/2}\right)^2\hat{\phi}(\mathbf{k}) \\ &= -\frac{4}{H^2}\sum_{i=1}^{3}\sin^2\left(\frac{Hk_i}{2}\right)\hat{\phi}(\mathbf{k}). \end{aligned}$$

and hence

$$\hat{\phi}(\mathbf{k}) = -4\pi G\underbrace{\frac{(H/2)^2}{\sin^2(Hk_1/2) + \sin^2(Hk_2/2) + \sin^2(Hk_3/2)}}_{\hat{\mathcal{G}}(\mathbf{k})}\hat{\rho}(\mathbf{k}), \tag{9}$$

where $\hat{\mathcal{G}}$ can be identified by comparison with Equation 8. It is worth noting that the constant multiplier $(-4\pi G)$ is often left out of $\hat{\mathcal{G}}$ (this is the convention used in [7]). In the implementation, values of $\hat{\mathcal{G}}$ are computed only once and saved for future look-up.

In the one-dimensional case, the above discussion can be easily rephrased in terms of a diagonalization problem [4]. In one dimension, the Poisson equation is

$$\frac{d^2\phi}{dx^2} = \rho. \tag{10}$$

The interval $[a, b]$ on which we wish to find ϕ is assumed to be discretized into N points $x_j = a + jH$, where $H = (b-a)/(N-1)$ and $j = 0, \ldots, N-1$. Furthermore, we assume periodic boundary conditions so that $\phi(x_{-1}) = \phi(x_{N-1})$ and $\phi(x_N) = \phi(x_0)$. The approximation of Δ at x_j is

$$(\Delta_H\phi)_j \equiv \frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{H^2}.$$

Thus, the discrete version of Equation 10 reads $(\Delta_H\phi)_j = \rho_j$ or

$$\frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{H^2} = \rho_j,$$

where $\rho_j = \rho(x_j)$. This gives a system of N equations (one per each sampled value of ρ) with N unknowns (values of ϕ) of the following form

$$\underbrace{\frac{1}{H^2}\begin{bmatrix} 2 & -1 & & & -1 \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ -1 & & & -1 & 2 \end{bmatrix}}_{\Delta_H}\underbrace{\begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \end{bmatrix}}_{\bar{\phi}} = -\underbrace{\begin{bmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \end{bmatrix}}_{\bar{\rho}}. \tag{11}$$

Next we define $\bar{\psi}(k) \in \mathbb{C}^N$ by $\psi_j(k) \equiv e^{i2\pi jk/N} = \omega^{jk}$ for $j = 0, \ldots, N-1$. The j-th row of $H^2\Delta_H\bar{\psi}$ equals

$$\begin{aligned}(H^2\Delta_H\bar{\psi})_j &= -\psi_{j-1} + 2\psi_j - \psi_{j+1} = -\omega^{(j-1)k} + 2\omega^{jk} - \omega^{(j+1)k} \\ &= -\omega^{jk}(\omega^{-k} - 2 + \omega^k) = -\omega^{jk}(\omega^{k/2} - \omega^{-k/2})^2 \\ &= 4\omega^{jk}\sin^2\left(\frac{\pi k}{N}\right).\end{aligned}$$

Hence we have $(H^2\Delta_H\bar{\psi})_j = 4\sin^2(\pi k/N)\psi_j$ which implies

$$\Delta_H\bar{\psi}(k) = \frac{4}{H^2}\sin^2\left(\frac{\pi k}{N}\right)\bar{\psi}(k).$$

This results tells us that for any value of k, $\bar{\psi}(k)$ is an eigenvector of Δ_H with eigenvalue $(4/H^2)\sin^2(\pi k/N)$. For $k = 0, \ldots, N-1$ we get N linearly independent eigenvectors $\{\bar{\psi}(k)\}$ which necessarily form the basis of $\mathbb{C}^N$. This implies that Δ_H is diagonalizable. The matrix F of its eigenvectors is given by $F_{jk} = \omega^{jk}$, and the inverse is easily verified to be $F^{-1} = (1/N)F^\dagger$. The eigendecomposition of Δ_H is therefore

$$\Delta_H = F\Lambda F^{-1},$$

where $\Lambda = \mathrm{diag}((4/H^2)\sin^2(\pi k/N))$. Substitution into Equation 11 yields $F\Lambda F^{-1}\bar{\phi} = -\bar{\rho}$ or

$$F^{-1}\bar{\phi} = -\Lambda^{-1}F^{-1}\bar{\rho}.$$

Evaluating the left-hand side leads to the conclusion that $F^{-1}\bar{\phi}$ is nothing else but the DFT of ϕ. Indeed,

$$(F^{-1}\phi)_k = \frac{1}{N}(F^\dagger\phi)_k = \frac{1}{N}\sum_{j=0}^{N-1}F^\dagger_{kj}\phi_j = \frac{1}{N}\sum_{j=0}^{N-1}\omega^{-kj}\phi_j = \frac{1}{N}\sum_{j=0}^{N-1}e^{-2\pi ijk/N}\phi_j$$

which is exactly the discrete Fourier transform of ϕ (using the standard definition of the DFT). Thus we see that the DFT of the Green's function derived in Equation 9 (or rather the one-dimensional variant thereof) is the k-th eigenfunction of the discretized Laplacian (with slight differences due to a different definition of the DFT being used there). By following the line of reasoning presented above, we observe a deep connection between the DFT and the eigendecomposition of the discretized Laplace operator.

A natural question that arises in the context of this discussion is whether a similar relation holds in the continuous case. The answer turns out to be positive; assuming free-space boundary conditions, the complex exponential $e^{2\pi i\mathbf{k}\cdot\mathbf{x}}$ (kernel of the inverse Fourier transform) is an eigenfunction of the Laplace operator (with eigenvalue $-(2\pi k)^2$) and the Fourier transform allows for a "diagonalization" of the Laplacian [4]. More precisely, we have

$$\Delta = F\Lambda F^{-1},$$

where F is the inverse FT, and $\Lambda = -(2\pi k)^2$. Applying this result to the Poisson equation yields $F^{-1}\phi = \Lambda^{-1}F^{-1}\rho$ or

$$\hat{\phi} = -\frac{1}{4\pi^2k^2}\hat{\rho}, \tag{12}$$

where the circumflex denotes the standard Fourier transform and $k = |\mathbf{k}|$. The function $-(2\pi k)^2$ is sometimes used instead of the Green's function $\mathcal{G}$ defined in Equation 9. This approach is the basis of

the "poor man's Poisson solver" (as dubbed by [7]). While implementing it, we have to take into account that in the standard definition of the DFT nonnegative frequencies correspond to $0 \leq k \leq N/2$ and negative frequencies correspond to $N/2 + 1 \leq k \leq N - 1$ (see [10], pp. 607–608). This necessitates the following "index wrapping":

$$k_i = \begin{cases} i & \text{if } i \leq \frac{N}{2} \\ i - N & \text{if } i > \frac{N}{2} \end{cases},$$

where i is the index of the gridpoint.

2.3 Field strength calculation

The strength $\mathbf{g}$ of the gravitational field at mesh-point $\mathbf{x_p}$ can be approximated using a central difference. Our implementation currently supports two types of finite differences, described below.

The two-point finite difference operator $\mathbf{D}$, whose x component is given by

$$D_x(\phi)(\mathbf{x_p}) = \frac{\phi(\mathbf{x}_{i+1,j,k}) - \phi(\mathbf{x}_{i-1,j,k})}{2H}$$

(and analogously for the y and z components), is second order accurate.

The fourth-order accurate finite difference is given by

$$D_x(\phi)(\mathbf{x_p}) = \alpha \frac{\phi(\mathbf{x}_{i+1,j,k}) - \phi(\mathbf{x}_{i-1,j,k})}{2H} + (1 - \alpha) \frac{\phi(\mathbf{x}_{i+2,j,k}) - \phi(\mathbf{x}_{i-2,j,k})}{4H},$$

where $\alpha = 4/3$.

The difference between the accuracy of both methods is illustrated in Figure 2. The figure also provides insight into how the error depends on the value of parameter α.

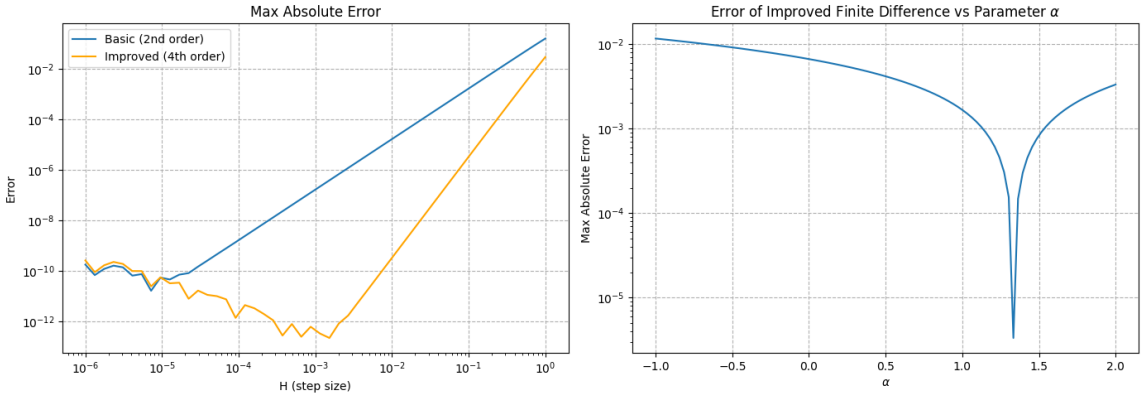


Figure 2: Left pane: Approximation error for second-order and fourth-order schemes. For small values of H, round-off errors dominate. Right pane: Approximation error vs. α in the improved finite difference scheme ($H = 0.1$). Note that the scheme is fourth-order accurate only for $\alpha = 4/3$ (cusp in the graph).

We can alternatively define the finite difference operators in terms of the delta function to get rid of the dependence on the differenced function. (Technically, the resulting quantities are functions rather than operators.) Consider for example the two-point finite difference in Equation 13. The definition can

be generalized beyond mesh points by letting

$$D_j(\phi)(\mathbf{x}) = -\frac{\phi(\mathbf{x}+H\mathbf{e}_j)-\phi(\mathbf{x}-H\mathbf{e}_j)}{2H} = -\int\left[\frac{\delta(\mathbf{x}+H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-H\mathbf{e}_j-\mathbf{x}')}{2H}\right]\phi(\mathbf{x}')d\mathbf{x}',$$

where $\mathbf{e}_j$ is the j-th standard basis vector. This motivates us to define

$$D_j(\mathbf{x}) = \frac{\delta(\mathbf{x}+H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-H\mathbf{e}_j-\mathbf{x}')}{2H} \tag{13}$$

and

$$D_j(\mathbf{x}) = \frac{\delta(\mathbf{x}+H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-H\mathbf{e}_j-\mathbf{x}')}{2H} + (1-\alpha)\frac{\delta(\mathbf{x}+2H\mathbf{e}_j-\mathbf{x}')-\delta(\mathbf{x}-2H\mathbf{e}_j-\mathbf{x}')}{4H} \tag{14}$$

as the two-point as four-point finite difference operators respectively.

If ϕ denotes the gravitational potential, then the field $\mathbf{g}$ is approximated at mesh point $\mathbf{x_p}$ as

$$\mathbf{g}(\mathbf{x_p}) = -\mathbf{D}(\phi)(\mathbf{x_p}).$$

2.4 Interpolation

The value of the field strength $\mathbf{g}(\mathbf{x})$ at the position particle's position $\mathbf{x}$ is calculated by interpolating the values of $\mathbf{g}$ from the neighboring mesh-points. Formally,

$$\mathbf{g}(\mathbf{x}) = \sum_{\mathbf{p}} W(\mathbf{x}-\mathbf{x_p})\mathbf{g}(\mathbf{x_p}).$$

In practice, there is no need to sum over all mesh points. Instead, we use an algorithm analogous to Algorithm 1 to only include the cells with non-zero contribution to the sum. The method is illustrated in Algorithm 2. It is important to note that in order to retain correct physical behavior, the interpolation

Algorithm 2 Field strength interpolation

1: **for each** particle i **do**
2: **for each** cell $\mathbf{q}$ in $\mathcal{C}_S(\mathbf{x}_i)$ **do**
3: $\mathbf{g}(\mathbf{x}_i) \leftarrow \sum_{\mathbf{q}} W(\mathbf{x}_i-\mathbf{x_q})\mathbf{g}(\mathbf{x_q})$

and mass assignment schemes must use the same shape to represent the particles. The procedure in Algorithm 2 is trivially parallelized by converting the sequential loop into a parallel one.

The procedures of density assignment and interpolation presented in Algorithm 1 and Algorithm 2 are high level description. More concrete formulations suitable for direct use in an implementation are given in [7] and [8].

2.5 Code units

Implementation of the PM (and P^3M) methods can be simplified by switching to a system of dimensionless units, often called *code units*. The natural units of time and length in a PM simulation are H and DT, respectively. Hence, length in a PM code is conveniently expressed in terms of multiples of H, and similarly time intervals are given as a multiple of DT, i.e. the conversion relations are

$$x' = \frac{x}{H} \quad \text{and} \quad t' = \frac{t}{DT}.$$

From there, it follows that

$$v' = \frac{DT}{H} v \quad \text{and} \quad a' = \frac{DT^2}{H} a.$$

The expected relation $\mathbf{g}' = -\nabla' \phi'$ leads to the definition $\phi' = (DT^2/H^2)\phi$. By stipulating that we have $\nabla'^2 \phi = \rho'$, we get $\rho' = DT^2 \cdot 4\pi G\rho$, $m' = (DT^2 \cdot 4\pi G/H^3)m$, and $G' = 1/(4\pi)$.

2.6 Properties of the calculated field

The accuracy of a simulation using a PM code depends on a variety of factors. The user can decide to use any combination of the following elements that parameterize the program:

- mass assignment and interpolation scheme: NGP, CIC, TSC, or possibly any other scheme from the infinite hierarchy described in subsection 2.1;
- convolution with the Green's function derived from the discretized Laplacian (Equation 9) or the "poor man's solver" (Equation 12);
- gradient approximation: two-point or four-point finite difference;
- grid resolution: number of meshpoints in each dimension.

In this subsection we analyze these choices have in from two different angles. We first focus on the properties of the field produced by a single source which gives insight into the behavior of the simulation on the basis of pair-wise interparticle interaction. Then, we analyze the global error in force calculation by comparing the PM-calculated forces acting on all particles in a typical simulation with forces produced by the PP method.

2.6.1 Local picture

The field produced by the PM method is neither homogenous nor isotropic. Anisotropy can be observed by measuring the field generated by a particle in two different directions. In Figure 3, the field strength calculated using the PM method (with the Green's function derived from discrete Laplacian) due to a single source at $x = H$ is shown in two variants: when measured along the x-axis (blue line) and the $x = y$ line (orange line). The difference between these two graphs illustrates anisotropy of the calculated field. The figure also shows the field strength measured along the x-axis when the source was shifted by $H/2$ in the $-x$ direction (green line). Its deviation from the case when the source was placed at $x = H$ (the blue graph) exemplifies inhomogeneity of the field computed using the PM method. As expected, the PM-calculated approximation gets better with increasing distance from the source; the inverse-square law is reproduced accurately for $r \gtrsim 4H$.

The single-source case analysis can be extended by considering the effect of choice of finite difference and mass assignment schemes on the relative error between the actual and approximated field strength. Figure 4 shows a comparison of (a) the relative error as a function of distance from the source for PM with second-order and fourth-order finite differences, and (b) the field strength computed using different mass assignment schemes discussed in subsection 2.1. The conclusion that can be drawn from Figure 4

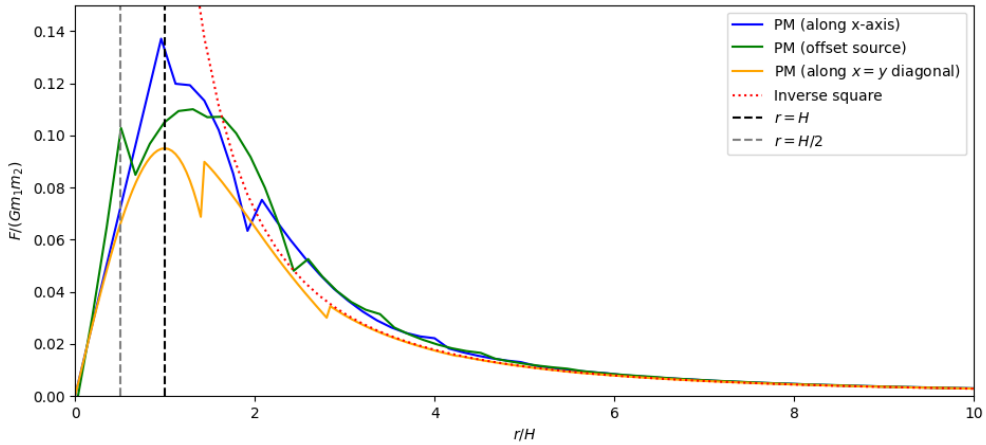
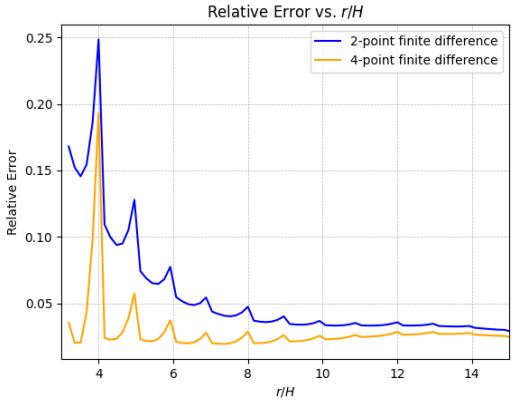
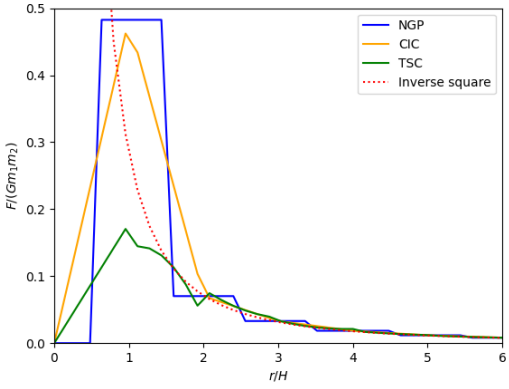


Figure 3: Anisotropy and inhomogeneity of the field as calculated by the PM method (TSC assignment, second order finite difference).



(a) Relative error of field strength approximation in different PM variants.



(b) Field strength calculated with the PM method using different assignment schemes.

Figure 4: Comparison of PM method performance. (a) Relative error due to finite difference accuracy. (b) Effect of mass assignment schemes on computed field strength (4-point finite difference).

is that, as expected, the TSC mass assignment scheme and the four-point finite difference yield best results.

The field produced by the PM method using the "poor man's" potential solver is similar to the variant employing the discretized Laplacian Green's function described above.

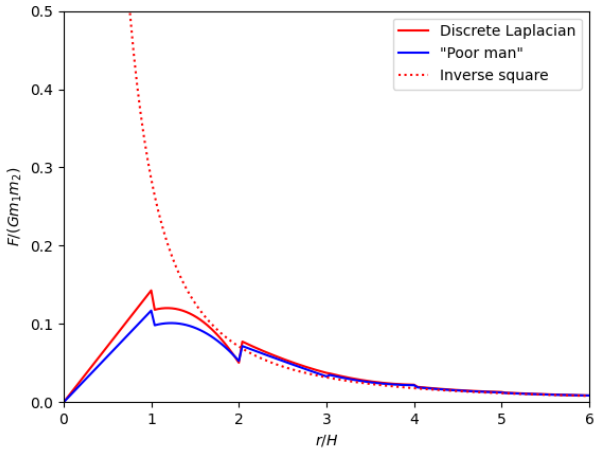


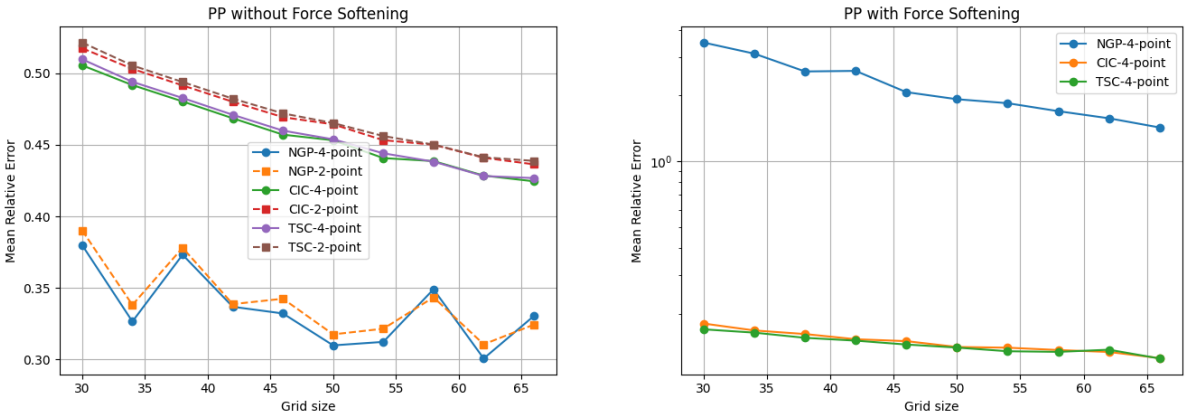
Figure 5: Field obtained potential calculated by the "Poor man's Poisson solver" vs. when using Equation 9.

2.6.2 Global picture

To calculate the global approximation error of a PM simulation, we calculated the average of relative differences over all particles ($N = 10{,}000$ in this case) between the forces produced by the PM method and the exact result obtained using the PP method. More precisely, we the error was defined as

$$\frac{1}{N}\sum_i \frac{|\mathbf{F}_i^{\text{approx.}} - \mathbf{F}_i^{\text{PP}}|}{|\mathbf{F}_i^{\text{PP}}|}. \tag{15}$$

The result, for different grid resolutions, is shown in Figure 6 ($30 \leq N_g \leq 70$). For the purposes of this test, the particle positions were generated according to the uniformly decreasing density disk model. As can be seen in Figure 6a, for each of the assignment schemes, the error is minimized for the four-point difference, as expected. A more surprising result is that the errors are seemingly greatest for the TSC assignment scheme. This is easily explained by considering the graphs in Figure 4b. As can be seen there, the TSC assignment scheme gives raise to a force which accurately reproduces the inverse-square law force but at the same time it severely underestimates the force close to the source. This behavior is expected since it stems from the mass of any individual particle being spread between multiple cells. The CIC and NGP schemes on the other hand, do not reproduce the actual force with the same level of accuracy but the underestimation close the source is not as severe as in the case of the TSC scheme. In some applications, for instance in the context of a galaxy simulation, this is a favorable feature of the TSC scheme. The number of stars in a galaxy can be in the order of a trillion [17], whereas the number of particles in our tests is in the order of tens of thousands, giving an average of around $10^{12}/10^4 = 10^8$



(a) PP calculation without force softening.

(b) PP calculation with force softening.

Figure 6: Average force approximation error in the PM method (discrete Laplacian solver).

of stars per particle. As can be seen, it would be a serious mistake to model the force due to a single particle by the force due to a dimensionless point mass. In such a case it is a standard practice to use a softened gravitational force instead of the one given in Equation 1 [3]. By using the TSC scheme, force softening is automatically included in the calculation. It can be argued that obtaining the smoothness properties enjoyed by the higher-order assignment schemes should be treated as tangential to softening the force. It is for this reason, that Hockeney and Eastwood introduce the notion of *force sharpening* as an additional step in the PM method in [7]. In this modified version of the PM method, the Poisson equation takes the form

$$\nabla^2 \phi_p = \rho_p^*,$$

where ρ_p^* is obtained from the density ρ_p by solving the following system

$$1 + \frac{\rho_{p+1}^* - 2\rho_p^* + \rho_{p-1}^*}{8} = \rho_p,$$

where p indexes the mesh points. In this work we do not investigate this matter further nor do we implement this modification in our program.

Now we return to the test whose results are presented in Figure 6. In the light of the above discussion, it is more reasonable to compare the quality of the results of the PM method with a *softened gravitational force* defined in [19] as

$$\mathbf{F}_{ij}^{\text{soft}} = -G\frac{m_i m_j}{(r_{ij}^2 + \epsilon^2)^{3/2}}\mathbf{r}_{ij}. \tag{16}$$

The relative difference between the softened-forces and the PM-calculated ones is shown in Figure 6b. The *softening length* in the PP method was set at $\epsilon = H$ in each run of the test, so that it roughly matches the softening induced by the PM method (cf. Figure 4b).

2.7 Implementation

2.7.1 CPU variant

The most computationally involved step of the PM method is the happens in the potential solver. As explained in subsection 2.2, in our implementation we solve the Poisson equation for the potential using the convolution method. This involves taking the DFT of the grid-defined density and later the inverse DFT of the product of transformed Green's function and the transformed density. Since the number of grid points in a typical grid can be of the order of a million, choosing the right implementation of the FFT is of paramount importance. We tried incorporating a handful of popular C++ FFT libraries into the program and out of those considered, FFTW (`https://www.fftw.org/`) and kissfft (`https://github.com/mborgerding/kissfft`) provided best performance. The running times for FFT and inverse FFT stages of the algorithm using each of the libraries are shown in Figure 1. As we can see,

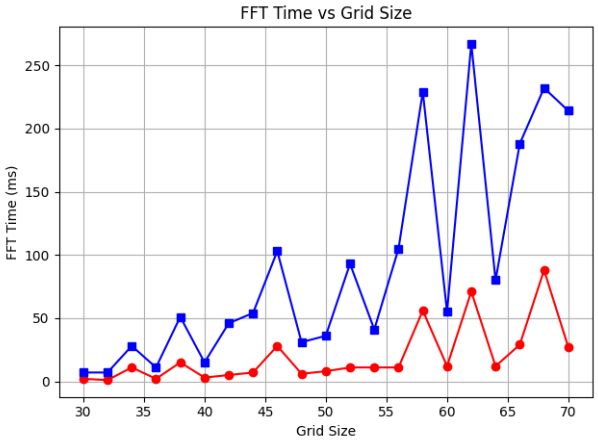


Figure 7: Time needed to perform the DFT and inverse DFT using kissft and FFTW libraries. The total grid size in the test was $(2N_g) \times (2N_g) \times N_g$, where N_g is labeled as "Grid Size" on the horizontal axis in the graph.

FFTW considerably outperformed kissfft, with the former achieving an average five-fold speedup over the latter. Interestingly, most spikes in the running time, taking place for certain grid sizes

As noted in subsection 2.1, parallelization of the mass assignment step which involves "spreading" the mass due to a particle over nearby cells, requires synchronization to avoid data races. The above requirement can be fulfilled by using mutexes or atomic operations. Traditionally, atomic RMW operations were supported by the C++ standard library only for the integer types, however recently C++20 introduced atomic RMW operations on floating point numbers (see `https://en.cppreference.com/w/cpp/atomic/atomic.html`). We conducted a simple benchmark on Intel Core i7-9750H CPU (three threads concurrently incrementing a global variable) to compare the performance of both approaches. The benchmark indicated that guarding the critical section with a mutex was around two times faster than using atomic operations. However, the same test run on a system with a different processor indicated that using atomic counter offers better performance, which shows that more careful benchmarking would be needed to reach a definite conclusion.

2.7.2 GPU variant

As explained before, the mass assignment, field calculation at mesh points, and interpolation steps of the PM method are embarrassingly parallel. For this reason rewriting the PM code for the GPU does not pose much difficulty. In our implementation we used CUDA C++ language extension to implement the method on the GPU. A thorough introduction to the arcana of programming NVIDIA GPU devices is given in [9]. In this work, we only remark on the aspects directly related to our use case.

The memory hierarchy of a GPU (also referred to as the *device*) is organized into three main levels:

1. global memory (accessible by all threads as well as *host* (CPU); high latency),
2. shared memory (accessible by threads in the same threadblock typically up to 48 KB; very low latency),
3. local storage (per thread storage in the form of registers managed by the compiler).

We recognized that one step of the PM method that could potentially benefit from transferring data to shared memory is the gradient calculation step. For example, if the 2-point finite difference approximation to the Laplacian is used, the potential from six cells needs to be fetched from memory. At the same time, there is an overlap between the potential data read by threads working on adjacent mesh cells. Thus, theoretically, fetching the potential data into shared memory could have positive impact on performance. However, our tests for the 2-point approximation, performed at early stages of the development of the program did not indicate significant improvement in the running time of the algorithm and this optimization path was not explored further.

A major performance bottleneck in the GPU implementation is transferring the data between host and device. These data transfers are necessary if one wishes to record the state of the simulation after each timestep. If only the final state of the simulation is of interest, then the PM method can execute entirely on the device transferring the data to the host only once after the final iteration of the simulation. This is the scenario that we assumed for the purpose of comparing the CPU and GPU implementation. The results of the test for different particle numbers are shown in Figure 8. In the test, the computational domain was split into $64 \times 64 \times 32$ cells, and the simulation was run for 200 iterations. Both plots in the figure exemplify linear complexity of the PM method, with the GPU implementation providing 1200% speedup over the CPU version.

3 Particle-particle particle-mesh method

The P^{3}M algorithm is a hybrid method: Forces between distant particles are calculated using the PM method, whereas, for particles lying closely together, the PP method is used. The total force applied to particle i is

$$\mathbf{F}_i^{\mathrm{SR}} + \mathbf{F}_i = \sum_{j \neq i} (\mathbf{f}_{ij}^{\mathrm{tot}} - \mathbf{R}_{ij}) + \mathbf{F}_i, \tag{17}$$

where $\mathbf{F}_i \approx \sum_{j \neq i} \mathbf{R}_{ij}$ is the force computed using the PM method and $\mathbf{R}_{ij} = \mathbf{R}(\mathbf{x}_i - \mathbf{x}_j)$ is a prescribed *reference force*. The reference force is defined as the force between two particle-clouds, i.e. each particle

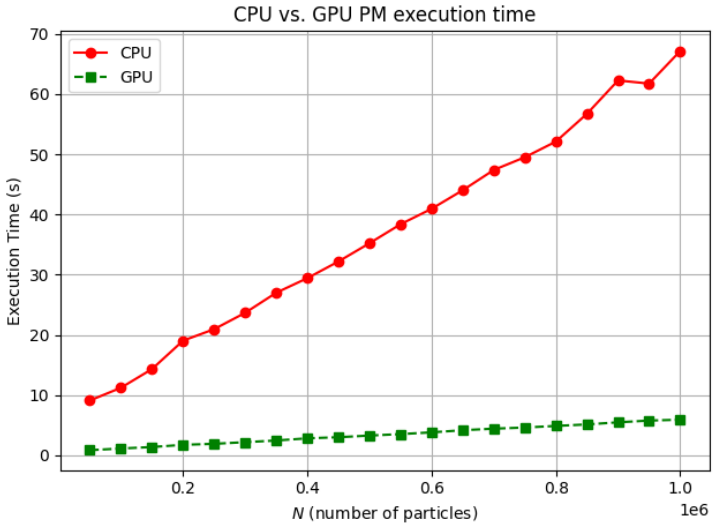


Figure 8: PM method: CPU and GPU implementation running time comparison.

is represented by a sphere with diameter a and a given density profile. The two examples of reference forces described in [7] are

$$R(r) = Gm_1m_2 \times \begin{cases} \frac{1}{35a^2}(224\xi - 224\xi^3 + 70\xi^4 + 48\xi^5 - 21\xi^6), & 0 \leq \xi \leq 1 \\ \frac{1}{35a^2}(12/\xi^2 - 224 + 896\xi - 840\xi^2 + 224\xi^3 + 70\xi^4 - 48\xi^5 + 7\xi^6), & 1 < \xi \leq 2 \\ \frac{1}{r^2}, & \xi > 2 \end{cases}$$

where $\xi = 2r/a$ for a sphere with uniformly decreasing density (shape S_2) and

$$R(r) = Gm_1m_2 \times \begin{cases} \frac{1}{a^2}(8r/a - 9r^2/a^2 + 2r^4/a^4), & r < a \\ \frac{1}{r^2}, \end{cases} \tag{18}$$

for a solid sphere (shape S_1). The reference force vector lies along the line joining the two bodies.

3.1 Optimal Green's function

As it is apparent from Equation 17, the method's validity depends on how well the reference force is approximated by the mesh force. The average deviation between the two forces can be minimized by a suitable choice of Green's function given (in k space) by

$$\hat{\mathcal{G}}(\mathbf{k}) = \frac{\hat{\mathbf{D}}(\mathbf{k}) \cdot \sum_{\mathbf{n}} \hat{U}^2(\mathbf{k_n}) \hat{\mathbf{R}}(\mathbf{k_n})}{|\hat{\mathbf{D}}(\mathbf{k})|^2 \left[\sum_{\mathbf{n}} \hat{U}^2(\mathbf{k_n})\right]^2}. \tag{19}$$

The derivation is mathematically involved and is detailed in [7]. We now proceed to examine the terms appearing in Equation 19.

3.1.1 Finite difference operator

The Fourier transform $\hat{\mathbf{D}}$ of the two-point finite difference operator defined in Equation 13 has the components

$$\begin{aligned}\hat{D}_j(\mathbf{k}) &= \int D_j(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}} d\mathbf{x} \\ &= \frac{1}{2H}\left(\int \delta(\mathbf{x}+H\mathbf{e}_j)e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x} - \int \delta(\mathbf{x}-H\mathbf{e}_j)e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}\right) \\ &= \frac{1}{2H}(e^{ik_jH} - e^{-ik_jH}) = \frac{i\sin(k_jH)}{H}.\end{aligned}$$

Similarly, for the four-point finite difference (Equation 14) we have

$$\hat{D}_j = \alpha\frac{i\sin k_jH}{H} + (1-\alpha)\frac{i\sin 2k_jH}{2H},$$

where $j = 1, 2, 3$.

3.1.2 Assignment function

The quantity $\hat{U}$ is defined as $\hat{W}/V$. For the mass assignment scheme hierarchy described in subsection 2.1 we have

$$\hat{U}(\mathbf{k}) = \left(\prod_{i=1}^{3}\frac{\sin(k_iH/2)}{k_iH/2}\right)^p,$$

where $p = 1, 2, 3, \ldots$ with $p = 1$ corresponding to NGP assignment, etc. In particular, for the TSC assignment scheme, the *alias sum*[2]

$$\sum_{\mathbf{n}}\hat{U}^2(\mathbf{k_n}) \equiv \sum_{\mathbf{n}}\hat{U}^2\left(\mathbf{k}+\mathbf{n}\frac{2\pi}{H}\right)$$

can be rewritten as

$$\begin{aligned}\sum_{\mathbf{n}}\hat{U}^2\left(\mathbf{k}+\mathbf{n}\frac{2\pi}{H}\right) &= \sum_{\mathbf{n}}\prod_{i=1}^{3}\left[\frac{\sin(k_iH/2+n_i\pi)}{k_iH/2+n_i\pi}\right]^6 \\ &= \prod_{i=1}^{3}\sum_{n_i}\left[\frac{\sin(k_iH/2+n_i\pi)}{k_iH/2+n_i\pi}\right]^6 \\ &= \prod_{i=1}^{3}\left[\sin^6\left(\frac{k_iH}{2}\right)\sum_{n_i}\frac{1}{(k_iH/2+n_i\pi)^6}\right]\end{aligned}$$

Using the the partial fractions expansion of the cotangent function [2],

$$\frac{(-1)^s}{s!}\frac{d^s}{dx^s}\cot x = \sum_{n=-\infty}^{\infty}\frac{1}{(x-n\pi)^{s+1}},$$

we can simplify the sum over n_i to

$$\frac{-1}{5!}\frac{d^5}{dx^5}\cot\left(\frac{k_iH}{2}\right) = 1 - \sin^2\frac{k_iH}{2} + \frac{2}{15}\sin^4\frac{k_iH}{2}.$$

[2]To get the alias sums compatible with the DFT definition given in Equation 7, one has to compute

$$\sum_{\mathbf{n}}\tilde{U}^2(\mathbf{k_n}) \equiv \sum_{\mathbf{n}}\tilde{U}^2(\mathbf{k}+\mathbf{n}N) = \frac{1}{H}\sum_{\mathbf{n}}\hat{U}^2\left(\mathbf{k}\frac{2\pi}{NH}+\mathbf{n}\frac{2\pi}{H}\right)$$

instead.

Hence,

$$\sum_{\mathbf{n}} \hat{U}^2_{\mathrm{TSC}}(\mathbf{k_n}) = \prod_{i=1}^{3} \left(1 - \sin^2 \frac{k_i H}{2} + \frac{2}{15} \sin^4 \frac{k_i H}{2}\right).$$

Using the same approach, we can obtain similar results for the CIC and NGP schemes, namely

$$\sum_{\mathbf{n}} \hat{U}^2_{\mathrm{CIC}} = \frac{1}{27} \prod_{i=1}^{3} \left(1 + 2\cos^2 \frac{k_i H}{2}\right) \quad \text{and} \quad \sum_{\mathbf{n}} \hat{U}^2_{\mathrm{NGP}} = 1.$$

3.1.3 Reference force

The quantity $\hat{\mathbf{R}}$, the transformed reference force, is related to the shape S of the particle-cloud by

$$\hat{\mathbf{R}}(\mathbf{k}) = \frac{i\mathbf{k}\hat{S}^2(k)}{k^2}, \tag{20}$$

where $k = |\mathbf{k}|$. This can be shown by recalling that $\mathbf{R}(\mathbf{x}_i - \mathbf{x}_j)$ is the force applied to cloud i due to cloud j. Hence $\mathbf{R}(\mathbf{x})$ is the force on cloud centered at $\mathbf{x}$ due to a cloud at the origin. The force acting on mass element $d\mathbf{x}'S(\mathbf{x}' - \mathbf{x})$ of the cloud centered at $\mathbf{x}$ is therefore

$$-Gd\mathbf{x}'S(\mathbf{x}' - \mathbf{x}) \int d\mathbf{x}''S(\mathbf{x}'')\frac{\mathbf{x}' - \mathbf{x}''}{|\mathbf{x}' - \mathbf{x}''|^3}$$

and the total force is

$$\mathbf{R}(\mathbf{x}) = -G \int d\mathbf{x}'S(\mathbf{x}' - \mathbf{x}) \int d\mathbf{x}''S(\mathbf{x}'')\frac{\mathbf{x}' - \mathbf{x}''}{|\mathbf{x}' - \mathbf{x}''|^3}. \tag{21}$$

The expression on the right-hand side of Equation 21 is a double convolution, $\mathbf{R} = -S * (S * \mathbf{g})$, where $\mathbf{g}(\mathbf{x}) = \mathbf{x}/|\mathbf{x}|^3$. The x-coordinate of $\mathbf{g}$ is $x/|\mathbf{x}|^3$ which coincides with $\partial h/\partial x$ for $h(\mathbf{x}) = -1/|\mathbf{x}|$. The Fourier transform of h is given in [5] (p. 363):

$$\hat{h}(\mathbf{k}) = \frac{4\pi}{|\mathbf{k}|^2}.$$

The formula for the transform of a derivative yields

$$\hat{g}_x(\mathbf{k}) = \frac{4\pi i k_x}{|\mathbf{k}|^2}.$$

Applying the convolution theorem twice and factoring the constant $(-4\pi G)$ out of the Green's function in Equation 19 leaves us with the desired Equation 20.

Since the shapes S are spherically symmetric, the calculation of $\hat{S}$ (appearing in Equation 20) can be simplified. The Fourier transform of S is

$$\hat{S}(\mathbf{k}) = \int S(\mathbf{x})e^{-i\mathbf{k}\cdot\mathbf{x}}d\mathbf{x}.$$

Observe that $\mathbf{k} \cdot \mathbf{x} = kr\cos\theta$, where θ is the angle between $\mathbf{k}$ and $\mathbf{x}$, and $r = |\mathbf{x}|$. Since S is invariant under rotations, θ can be chosen to be the angle between $\mathbf{x}$ and the z-axis. Thus, the integral, rewritten in spherical coordinates becomes

$$\hat{S}(k) = \int_0^{2\pi} d\phi \int_0^{\pi} d\theta \int_0^{\infty} drS(r)e^{-ikr\cos\theta}r^2\sin\theta = 2\pi \int_0^{\infty} r^2 dr \int_0^{\pi} d\theta e^{-ikr\cos\theta}\sin\theta.$$

The θ-integral upon the substitution $-kr\cos\theta \to u$ becomes

$$\frac{1}{kr}\int_{-kr}^{kr} e^{iu}du = \frac{1}{kr}\int_{-kr}^{kr}(\cos u + i\sin u)du = \frac{2\sin kr}{kr}$$

and hence

$$\hat{S}(k) = 4\pi \int_0^\infty r^2 S(r) \frac{\sin kr}{kr} dr.$$

This integral, evaluated for the S_1 and S_2 shapes respectively, gives

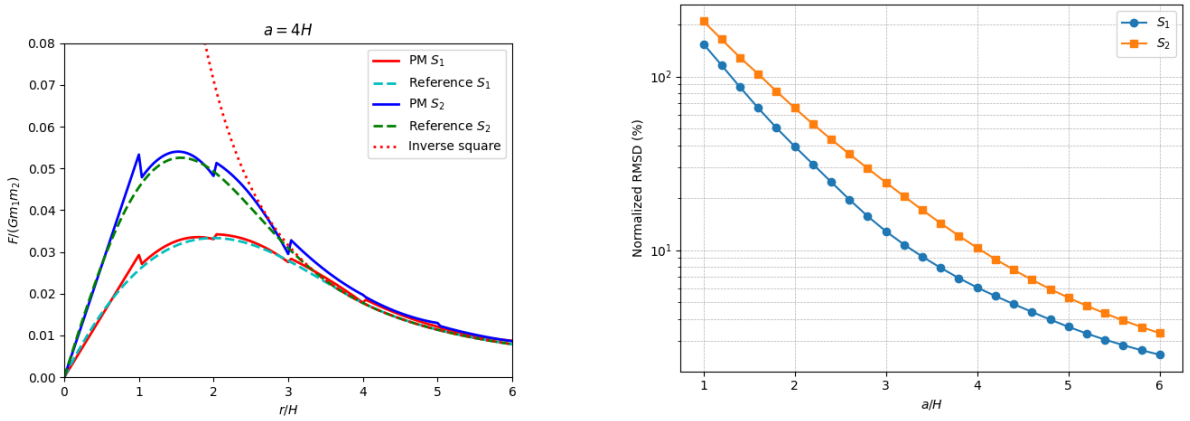
$$\hat{S}_1(k) = \frac{3}{(ka/2)^3} \left(\sin \frac{ka}{2} - \frac{ka}{2} \cos \frac{ka}{2} \right)$$

and

$$\hat{S}_2(k) = \frac{12}{(ka/2)^4} \left(2 - 2\cos \frac{ka}{2} - \frac{ka}{2} \sin \frac{ka}{2} \right).$$

Finally, we note that the infinite sum in the numerator of Equation 19 does not have a closed form but this does not pose a problem since the summand decays rapidly with $\mathbf{n}$ moving further away from zero.

The result of applying the optimal Green's function in Equation 8 is illustrated in Figure 9a. As can be seen, the PM force closely follows the reference force, especially for distances $r > a$, where the reference force nearly matches the inverse-square law. It is worth noting that for r slightly smaller than a, the reference force and its mesh-based PM approximation, still provide a good fit to the inverse-square behavior. This observation allows the cutoff radius r_e, which defines the boundary for direct summation, to be chosen smaller than a (e.g., $r_e = 0.7a$), improving performance without sacrificing accuracy.

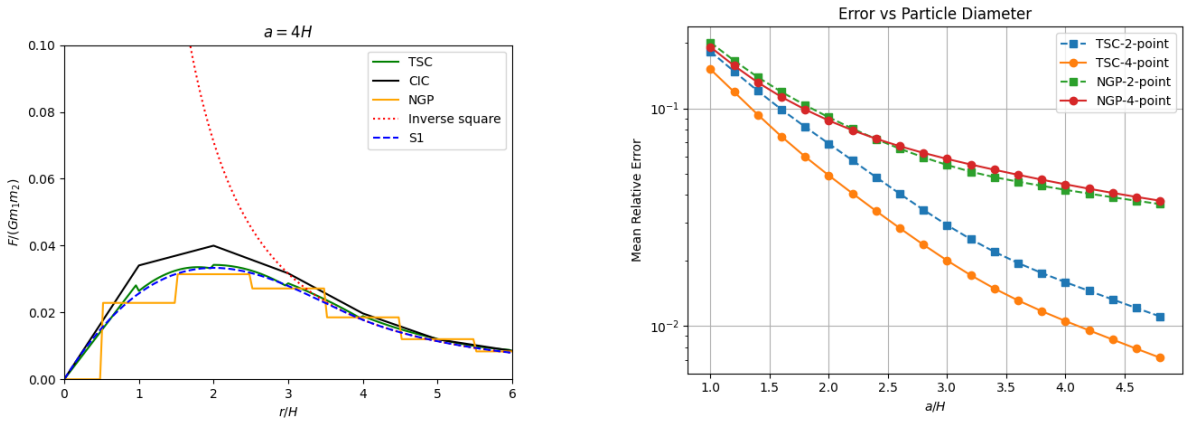


(a) Approximate vs. reference force.

(b) Mean-normalized RMSD error

Figure 9: Comparison of PM approximation using S_1 and S_2 shape functions. The mesh approximation to the reference force was computed using the PM method with a TSC assignment scheme, two-point finite difference, and Green's function optimal for each shape.

The P^{3}M method offers a high degree of flexibility in terms of specifying the "building blocks" of the Green's function in Equation 19. For example, we can be interested in learning how the choice of the assignment scheme influences the shape of the PM-approximated reference force. The result of a test which addresses this issue is shown in Figure 10. Figure 10a shows that the general characteristic of the field strength approximated by the PM method is similar to what we have already seen in Figure 4b. Namely, the NGP schemes yields an unnatural, jagged shape. The shape produced using the CIC shape matches the S_1 reference force more closely, although it overestimates the field strength values close to



(a) Approximate vs. reference force.

(b) Mean relative error.

Figure 10: Comparison of PM approximation using different assignment and interpolation functions. In the test we set $N = 10{,}000$ and used $64 \times 64 \times 32$ grid.

the source. The most accurate approximation is obtained using the TSC scheme (this is the same shape we have already seen in Figure 9a).

The situation portrayed in Figure 10a gives insight into a field due to a single particle. In a typical situation, we are interested in simulating many thousands of particles and thus the averaged approximation error becomes of interest. Just as in the case of the PM method, the global error was measured in terms of mean relative error given by Equation 15. This time, the error was plotted as a function of the particle diameter for TSC and NGP schemes both in two variants: using second-order and fourth-order Laplacian approximation. The result of this test is shown in Figure 10b. The figure illustrates that the error decays rapidly with increasing particle diameter when the TSC scheme is used. It is worth noting that the fourth-order accurate finite difference yields significant improvement in accuracy for the TSC scheme-based P3M method, allowing the error to drop below 1% mark for $a \approx 4H$.

To measure the quality of PM-approximation of a given reference force, we used mean-normalized RMSD error, i.e. we calculated

$$\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(F^{\mathrm{PM}}(r_i) - R(r_i))^2}}{\frac{1}{n}\sum_{i=1}^{n} R(r_i)}$$

in a system with a single unit mass located on the x-axis. The forces $F^{\mathrm{PM}}(r_i)$ were measured along the x-axis in equal-length intervals and the test was run for increasing values of particle diameter a. The result of the test is shown in Figure 9b. Clearly, the approximation error becomes smaller for increasing values of a. Since r_e is proportional to a, this implies that there is a tradeoff between accuracy and the execution time. Increasing a means that r_e should increase as well which forces more particles into short-range correction regions. If the system under consideration has uniform density, we note that for any given particle, the short-range correction is concerned with $\sim r_e^3$ particles in its vicinity which translates into $\sim r_e^3$ operations. The relation between cutoff radius size r_e and execution time is shown in Figure 11. As we can see, the execution time very strongly depends on r_e and this explains why choosing r_e slightly even slightly less than a can lead to significant improvements in performance. Another conclusion that can be

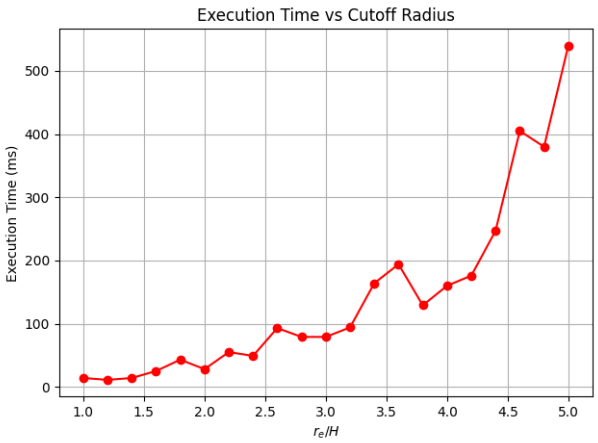


Figure 11: Execution time for different setting of cutoff radius r_e in the P^{3}M method (single step, $N = 10{,}000$). The times shown in the graph do not include tabulating the Green's function and the PM step.

drawn from Figure 9b is that the S_2-based approximation exhibits higher mean-normalized RMSD error compared to S_1. However, as we remarked previously, it allows for choosing $r_e \lesssim a$ without significant loss of accuracy. Therefore, the choice between S_1 and S_2 shapes involves a trade-off between accuracy near the source and overall error, making the optimal choice context-dependent.

3.2 Identifying close pairs of particles

In the P^{3}M method, in addition to the mesh used in the PM algorithm (the "potential mesh"), a second mesh (the *chaining mesh*) is used. The chaining mesh is sparser than the potential mesh. Its sole purpose is to partition the space into cells so that particles "close" to the ones in a given cell can be found efficiently. In this context, two particles are considered to lie close to one another if their separation is less than the cutoff radius.

The number of chaining mesh cells in a single dimension is given by $M_i = \lfloor L_i/r_e \rfloor$, where L_i is the side length of the computational box ($i = 1, 2, 3$). This implies that the side length of a chaining mesh cell is $\mathrm{HC}_i = L_i/M_i \geq r_e$. Thus, for every particle i in a given cell $\mathbf{p}$, it is sufficient to search through the immediate neighborhood of $\mathbf{p}$ to find all the particles within the cutoff radius from i.

In our program, the chaining mesh is implemented as a *head-of-chain* (HOC) array, depicted in Figure 12. The basic version of the HOC array is very cheap to build. Given a particle at point $\mathbf{x} = (x, y, z)$, we can determine in which chaining mesh cell $\mathbf{c}$ it lies by simply performing integer division on each of its coordinates, i.e. $\mathbf{c} = (x/\mathrm{HC}_1, y/\mathrm{HC}_2, z/\mathrm{HC}_3)$. Then, to get a HOC table index i corresponding to $\mathbf{c}$, standard index flattening is used. Subsequently, a new linked-list node is allocated and inserted at the beginning of the list HOC[i]. Clearly, the whole process is linear in the number of particles and the array can be constructed anew at each time step without degrading performance. In our implementation, additional savings are made by preallocating a memory pool large enough to store N nodes of the linked lists and reusing it for the HOC array initialization.

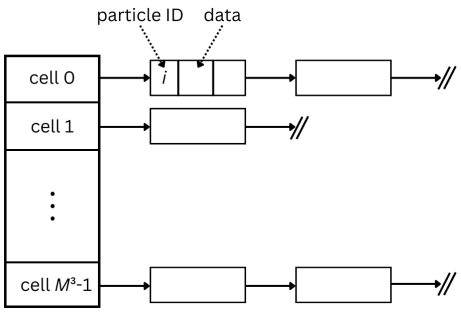


Figure 12: Head-of-chain data structure used for mapping particles to their parent cells in the chaining mesh. Here $M_1 = M_2 = M_3 = M$.

A potential optimization discussed in [7] involves sorting each linked list by a selected particle coordinate, such as the y-coordinate. This ordering enables early termination of the direct summation loop when the condition $y_i - y_j > r_e$ is met, effectively reducing unnecessary pairwise computations as particle i "sweeps through" a cell containing particles j. Our experiments indicate that constructing the linked lists in sorted order is substantially more expensive than the unsorted variant. This is to be expected; in the worst-case scenario where all particles are in a single chaining-mesh cell, and are inserted into the list in decreasing order of y coordinates, the complexity is $O(N^2)$. For a system of 50,000 particles, incorporating y-sorting increased the HOC (head-of-chain) construction time from approximately 180 milliseconds to over 13,000 milliseconds, a 70-fold slowdown. Despite this added cost, we observed no significant performance improvement in the short-range correction phase of the computation.

3.3 Short-range correction

The short-range correction, which takes place immediately after the mesh forces are found using the PM method, is at the heart of the P^{3}M algorithm. Since it scales with a square of the number of particles in each neighborhood, further optimizations are highly desirable.

By Newton's 3rd law, $\mathbf{f}_{ji}^{\mathrm{SR}} = -\mathbf{f}_{ij}^{\mathrm{SR}}$, which allows us to do the calculation of the short-range inter-particle force for any pair (i, j) of particles only once, leading to the reduction of the total running time by half. Informally, the particle i updates its total short-range force $\mathbf{F}_i^{\mathrm{SR}}$ as well as the total short-range force $\mathbf{F}_j^{\mathrm{SR}}$ of its neighbor j. To avoid double-counting, the particle i residing in cell $\mathbf{q}$ has to look for its neighbors in a subset $\mathcal{N}$ of the immediate neighborhood of $\mathbf{q}$. More specifically, define

$$\mathcal{N}(\mathbf{q} = (q_1, q_2, q_3)) = \{(q_1 + t, q_2 - 1, q_3 + s), (q_1 + s, q_2, q_3 - 1), (q_1 - 1, q_2, q_3) \mid s, t = -1, 0, 1\}. \quad (22)$$

Thus $|\mathcal{N}| = 13$, which is half of the size of the immediate neighborhood. The set $\mathcal{N}$ given in Equation 22 is not easily to illustrate. Since for the purposes of the later discussion, it is beneficial to have a clear picture in mind, we depict its two-dimensional analog in Figure 13.

The short-range correction part of the P^{3}M method is shown in Algorithm 3. For each chaining mesh cell $\mathbf{q}$, we compute the pair-wise interactions between the particles in $\mathbf{q}$ and the particles in the chaining mesh cells $\mathbf{q}_n$ from the reduced neighborhood $\mathcal{N}(\mathbf{q})$ *plus* $\mathbf{q}$. The inclusion of $\mathbf{q}$ allows us to take into account forces between particles within $\mathbf{q}$.

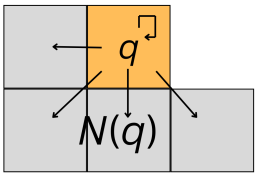


Figure 13: Set $\mathcal{N}$ in two dimensions.

Algorithm 3 Short-range correction

```
1: for each chaining cell q do
2:     for each q_n ∈ N(q) ∪ {q} do
3:         for each i ∈ HOC(q) do
4:             for each j ∈ HOC(q_n) do
5:                 if |y_i − y_j| > r_e then
6:                     break
7:                 UpdateShortRange(i, j, q, q_n)
```

The UpdateShortRange procedure is defined in Algorithm 4. In line 2, we exclude self-forces.

Algorithm 4 Updating short-range forces

```
1: procedure UpdateShortRange(i, j, q, q_n)
2:     if i = j then return
3:     r_ij ← r_i − r_j
4:     if |r_ij|^2 > r_e^2 then return
5:     r_ij ← |r_ij|
6:     r̂_ij ← r_ij / r_ij
7:     R_ij ← −m_i m_j R(r_ij) r̂_ij
8:     f^tot ← −G m_i m_j / r_ij^2 r̂_ij
9:     f_ij^SR ← f^tot − R_ij
10:    f_ji^SR ← −f_ij^SR
11:    F_i^SR ← F_i^SR + f_ij^SR
12:    if q_n ≠ q then                ▷ Avoid double-counting in the parent cell
13:        F_j^SR ← F_j^SR + f_ji^SR
```

The check in line 4 assures that the correction happens only for particles with separation less than the cutoff radius r_e. The pair-wise short-range force is calculated in lines 5–10, with the calculation in line 10 exploiting the Newton's third law, as described previously. The accumulation of total short-range force for a given particle takes place in line 11. The short-range force is also added to the total short-range force of the other particle in the pair (particle j) but only if $\mathbf{q}_n \neq \mathbf{q}$. This stipulation is crucial, as otherwise we would be double counting the forces between particles within $\mathbf{q}$.

As suggested in [7], the computational burden of operations in lines 5–8 in Algorithm 4 can be greatly reduced by storing the values of $f^{\mathrm{SR}}(r)/r = (f^{\mathrm{tot}}(r) - R(r))/r$ in a lookup table T at uniform intervals

Δ^2 of $[0, r_e^2]$ and interpolating them later. The schematic depiction of the interpolation is shown in Figure 14. If we define $\xi = r^2/\Delta^2$ and $t = \lfloor \xi \rfloor$, then

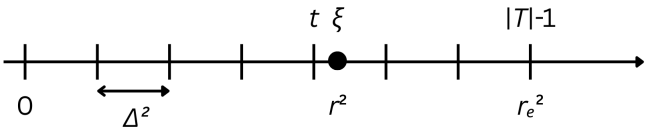


Figure 14: Interpolation of short-range force values.

$$\frac{f^{\mathrm{SR}}(r)}{r} \approx T[t]\,(1-(\xi-t)) + T[t+1](\xi-t) = T[t] + (\xi-t)(T[t+1]-T[t]).$$

The value $\mathbf{f}_{ij}^{\mathrm{SR}}$ can then be obtained by multiplying the interpolated quantity $f^{\mathrm{SR}}(r_{ij})/r_{ij}$ by $Gm_im_j\mathbf{r}_{ij}$, eliminating the use of the square root operations and reducing the total number of floating-point operations to just four.

In our implementation, the procedure outlined inAlgorithm 3 is parallelized by splitting the work done in the outmost loop between some number of threads. In doing so, extra care has to be taken to avoid data races. A thread t that is currently processing cell $\mathbf{p}$ and its neighbors (we say that t is *assigned* to $\mathbf{p}$) may "clash" with a different thread assigned to a nearby cell $\mathbf{q}$ (because possibly $\mathbf{p} \in \mathcal{N}(\mathbf{q})$). However, by the construction of the set $\mathcal{N}$, it is possible to split the short-range force into 14 parts, each of which is accessed by only one thread. For example, consider a particle i in cell $\mathbf{p} = (p_1, p_2, p_3)$ (in other words, $\mathbf{p}$ is the parent cell of i). If thread t is currently assigned to this cell, t will update the part of $\mathbf{F}_i^{\mathrm{SR}}$ corresponding to updates of i coming from within the same cell as the parent cell of i. Possibly at the same time, thread t' assigned to cell $\mathbf{q} = (p_1 + 1, p_2, p_3)$ will update a different part of $\mathbf{F}_i^{\mathrm{SR}}$, i.e. the part corresponding to updates of i coming from the cell "to the right" of the parent cell of i. Since only one thread is responsible for updates to particle i coming "from the right," (or any other "direction") no data races can occur. This approach presents two major drawbacks. First, it significantly increases memory usage, requiring storage for an additional $13N$ three-dimensional vectors. Second, it offers no guarantee of uniform workload distribution across threads. This imbalance, combined with the substantial variation in operations performed by individual threads, leads to severe thread divergence, rendering the parallelization scheme unsuitable for GPU execution.

3.4 Force approximation error and performance

The "local picture" of the errors of force approximation in the $\mathrm{P^3M}$ method has already been outlined in Figure 9. Now we turn to investigate how the choice of the particle shape (S_1 or S_2) and the size of the particle diameter influences the global force approximation error. The error is defined exactly the same as in the case of the PM method, i.e. we use Equation 15.

In our test, the grid covering the computational region had dimensions of $64 \times 64 \times 32$ and $N = 10{,}000$ particles were used. The result of the test is shown in Figure 15. The figure clearly illustrates that the error drops with increasing the size of the particle diameter a. This is an expected result, since with increasing a, the P3M method calculates more interparticle forces through exact direct summation.

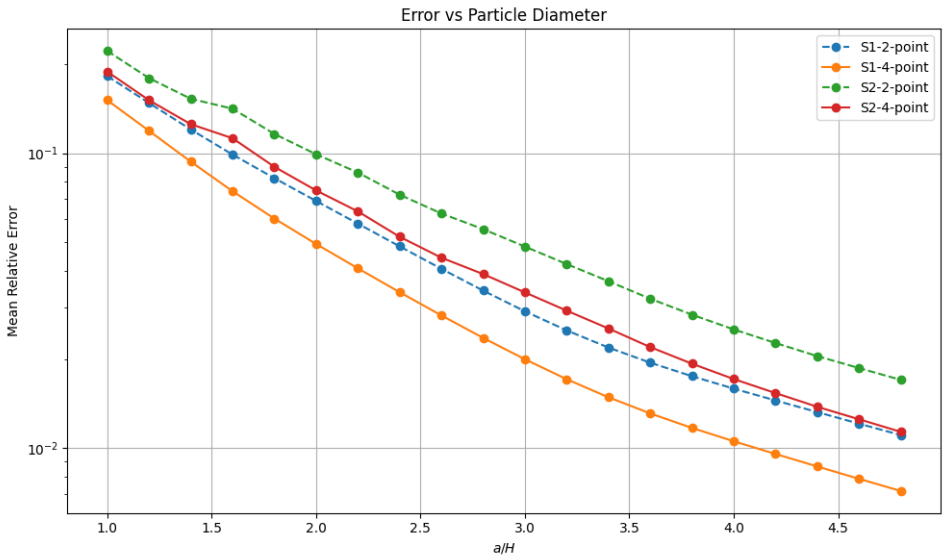


Figure 15: Mean relative error in P3M approximation.

Moreover, as we could see in Figure 9b, the PM approximation to the reference force gets better with increasing particle diameter. These two effects combined are responsible for the reduction of the global error.

For both the S_1 and S_2 shapes, the error was minimized when the fourth-order finite difference was used for gradient calculation, as one could obviously expect. The test indicated superiority of the S_1 shape, with the difference between the two becoming more significant for bigger relative values of a.

4 Barnes-Hut algorithm

The idea behind Barnes-Hut algorithm differs substantially from previously described mesh-based methods. The algorithm deals with gravitational forces directly instead of deriving them from the mesh-defined potential, as was the case with the PM and P^3M methods. Significant reduction of time complexity, from quadratic to $O(N \log N)$, is achieved by approximating the potential due to "far enough" groups of particles by the initial terms of its multipole expansion [13]. The grouping of particles is hierarchical in nature and is thus best understood as a tree. The entire set of particles comprises the top-level group, represented by the root of the tree; the eight children of the root node are representative of groups of particles residing in each of the octants of the computational domain, etc. The process of subdividing the space into eight smaller volumes at each node continues recursively until there is only one or zero particles left in a given volume. Nodes which satisfy this condition are the leafs of tree and are sometimes called the *external nodes*. The remaining nodes, each of which has eight children, are called *internal nodes*.

In the basic variant of the algorithm, the potential due to a group of particles is approximated using only the monopole term with respect to the center of mass of the group, i.e.

$$\phi_{\text{mon}}(r) = -\frac{GM}{r},$$

where M is the group's total mass. Since the potential is expanded about the center of mass, the

dipole moment $\mathbf{p} = \sum_i m_i \mathbf{r}_i$ vanishes. Hence, the next possible improvement comes from including the quadrupole term

$$\phi_{\text{quad}}(r) = -\frac{G}{2r^5}\mathbf{r} \cdot (\mathbf{Q}\mathbf{r}),$$

where $\mathbf{Q}$ is the quadrupole moment tensor defined as

$$Q_{ij} = \sum_k (3r_{ki}r_{kj} - 3r_k^2\delta_{ij})m_k.$$

In theory, we could keep on adding more terms to improve the quality of the approximation. In our implementation however, we restrict ourselves to the quadrupole term.

4.1 Building the tree

The data structure that fits the description given in the introduction is called an *octree*. An internal node of the octree stores the COM vector, the total mass of the group it represents, and the quadrupole tensor, whereas an external node stores a reference to the actual particles (or is empty if no particle was found in its associated volume). The recursive procedure of building the tree is shown in Algorithm 5. The quadrupole moment tensor for each node is calculated once the whole tree is already built. The

Algorithm 5 Insert a particle into the Barnes-Hut tree

```
1: function INSERT(n, p)
2:     if n is an internal node then
3:         Update n.COM and total mass n.M of n with p
4:         INSERT(child of n that should contain p, p)
5:     else if n is empty then
6:         Assign p to n
7:     else                                          ▷ Occupied external node
8:         Subdivide n into child nodes
9:         Move existing particle p' in n into child that should contain p'
10:        Update center of mass and total mass of n with p and p'
11:        INSERT(child of n that should contain p, p)
```

recursive relation used in this calculation is given in [6] and reads

$$\mathbf{Q} = \sum_{\text{child } c} \mathbf{Q}_c + \sum_{\text{child } c} m_c(3\mathbf{R}_c \otimes \mathbf{R}_c - R_c^2\mathbf{I}),$$

where $\mathbf{R}_c = \mathbf{x}_c^{\text{COM}} - \mathbf{x}^{\text{COM}}$ is the displacement vector from the COM of child c to the COM of the parent, $\mathbf{I}$ is the identity matrix, and $\otimes$ denotes the outer product.

For reasons that will become apparent later, it can be beneficial to separate the COM calculation from the tree creation part. In such a case, the COM is calculated recursively using the relation

$$\mathbf{x}^{\text{COM}} = \frac{\sum_{\text{child } c} m_c \mathbf{x}_c^{\text{COM}}}{\sum_{\text{child } c} m_c} \tag{23}$$

after the tree has already been built (and obviously before the quadrupole moment calculation).

4.2 Acceleration calculation

In the Barnes-Hut algorithm, the net acceleration of a particle p is calculated by summing the contributions from single particles or groups of particles while traversing the tree. The decision whether the acceleration can be approximated using the information stored in an internal node n depends on the relative distance from p to n.COM (the center of mass of group represented by n). The distance is relative to the *width* H of the node, i.e. the side length of the cubical volume encompassed by the node. More concretely, the approximation takes place if $n.H/|n.\text{COM} - p.\mathbf{x}| < \theta$, where θ is the so-called *opening angle*. In the extreme case when θ is set to zero, no approximations take place, and the algorithm reduces to the PP method. The procedure described above is illustrated in Algorithm 6. In the implementation,

Algorithm 6 Compute gravitational force on a particle using Barnes-Hut approximation

```
1:  function FindAcceleration(n, p, θ)
2:      if n is an external node then
3:          if n contains a particle q ≠ p then
4:              p.a ← p.a + GravitySoft(q.x, q.m, p.x)/p.mass
5:          return
6:      if n.H/|n.COM − p.x| < θ then
7:          p.a ← p.a + Gravity(n.COM, n.M, p.x)
8:          return
9:      for each child n_c of n do
10:         FindAcceleration(n_c, p, θ)
```

the GravitySoft function calculates the gravitational force softened by ϵ, i.e. it returns the value given by Equation 16. The pairwise potential energy associated with this force is given by

$$\Phi_{ij}^{\text{soft}} = -\frac{Gm_i m_j}{\sqrt{r_{ij}^2 + \epsilon^2}}. \tag{24}$$

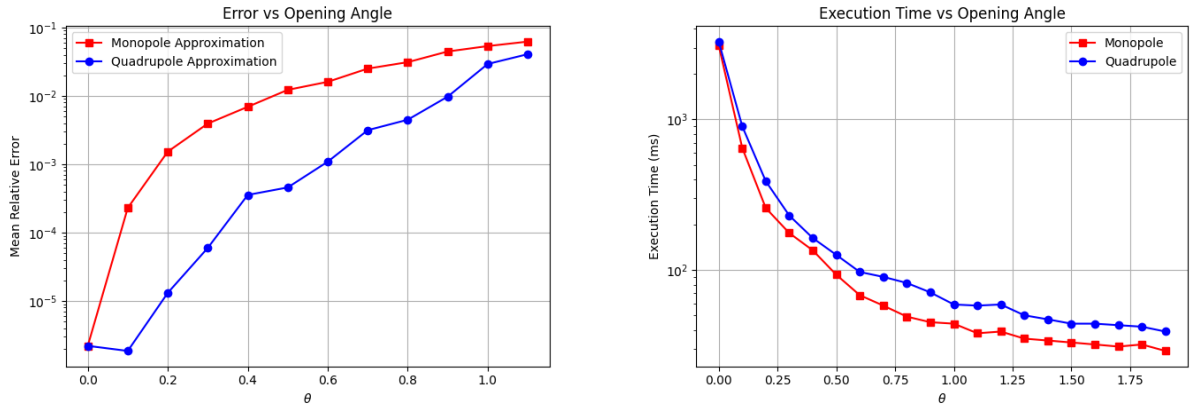
The Gravity function returns the approximation (up to the quadrupole term) of the acceleration due to a group of particles represented by a given node, i.e.

$$\mathbf{a} = -GM\frac{\mathbf{r}}{r^3} + \frac{G}{r^5}\mathbf{Q}\mathbf{r} - \frac{5G}{2}(\mathbf{r}\cdot(\mathbf{Q}\mathbf{r}))\frac{\mathbf{r}}{r^7}$$

(see [6]).

One possible way to quantify the quality of approximation for a given value of θ is to consider the relative error of calculated force. We set the same initial conditions of the system for both the PP direct summation method and the Barnes-Hut algorithm, compute the deviation of Barnes-Hut forces from PP forces acting on each particle, and take the average over all particles. In other words, the error calculated is given by the Equation 15. The dependence of the error on the opening angle θ and the corresponding execution time are shown in Figure 16. The figure includes plots for two cases: when only the monopole term is used in the Barnes-Hut approximation, and when both the monopole and quadrupole terms are included. As can be seen, the quadrupole-based algorithm exhibits significantly improved error scaling

with increasing θ. Naturally, this raises the question of the additional computational cost incurred by the inclusion of the quadrupole term. Our tests showed that this impact is minimal. The results (for $N = 10{,}000$ particles and $0 \leq \theta \leq 2$) support this observation. Both tests described above were



(a) Force approximation error.

(b) Execution time.

Figure 16: Comparison of error and execution time in the Barnes-Hut algorithm using monopole and quadrupole approximations.

conducted on a uniform disk particle distribution.

We note that direct calculation of total potential energy is infeasible as $O(N^2)$ operations would be required. Instead, we use an approximation based on the values stored in the tree. The approximate value of the potential energy is accumulated for each particle using a procedure analogous to force calculation. Indeed, the only difference between the two is the replacement of gravitational force calculation in Algorithm 6 with potential energy calculation according to Equation 24.

It is also noteworthy that the procedure outlined in Algorithm 6 is embarrassingly parallel. In our CPU implementation, the workload is split between an arbitrary number of threads on particle-by-particle basis.

4.3 Accelerating tree construction

Unlike the force calculation part which is easily parallelized, the tree construction step is inherently sequential. An approach to parallel tree construction outlined in [14] involves splitting the set of particles into load-balanced spatial groups which are then used to construct separate trees (one per thread), with the final step being the merge of the created trees. The merge step is highly involved and beyond the intended scope of this thesis. For this reason, we propose an alternative approach to accelerating the tree construction which does not involve building the tree in parallel.

Our strategy, based on the ideas put forth in [14], stems from an observation that a potential bottleneck in Algorithm 5 is related to irregular accesses to the nodes of the tree. Inserting two particles separated by a large spatial distance leads to visiting nodes along two completely different paths in the tree which results in a huge number of cache misses. A solution to this problem is to sort the list of particles in such a way that subsequent particles on the list (typically) lie close to each other in the

physical space. An ordering with this property can be generated by numbering the particles based on their position along a chosen *space-filling curve*.

A space filling curve can be thought of as the limit of an infinite sequence of curves which "fill" the space without "holes" [16]. In the limit, the curve reaches every point of the space, but due to practical constraints we can only deal with an approximation, i.e. some member of the limiting sequence. In our implementation, we are working with a *Z-order curve*, also called a *Morton space-filling curve*, so it will in the focus of our discussion. To make the visualization simpler, assume that the computational domain is two-dimensional and coarsely divided into 16 cells (particles within the same cells are ordered arbitrarily). Then, the Z-order curve that covers all 16 cells is shown in Figure 17. The ordering of

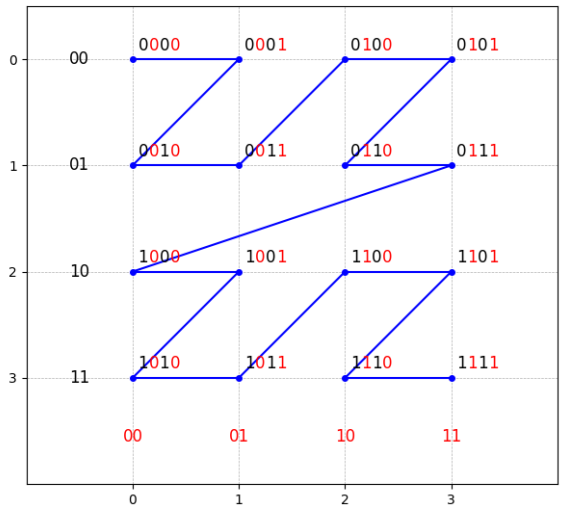


Figure 17: Z-order curve.

the points along the curve is obtained by assigning each cell a *Z-value*, whose binary representation is calculated by interleaving the bits of the x and y coordinates. This is illustrated in Figure 17 by coloring the x- and y-coordinate bits red and black respectively.

In a standard setting, the coordinates of a particle are given by real numbers and not integers however. In order to establish a mapping between the two, we have to first decide on the intended resolution of the grid. In our implementation, we use the 32-bit integer type to represent the Z-values, which means that each coordinate should be represented by a 10-bit integer number ($3 \times 10 = 30 \leq 32$). Thus, the mapping from real-valued coordinate of a particle to an integer one is given by

$$\left\lfloor \frac{(x - \text{low}.x)}{H} \times (2^{10} - 1) \right\rfloor \in [0, 2^{10}),$$

where low and H define the computational domain as

$$\text{domain} = [\text{low}.x, \text{low}.x + H] \times [\text{low}.y, \text{low}.y + H] \times [\text{low}.z, \text{low}.z + H].$$

In our proposed optimization, we make the following changes into the tree construction algorithm (Algorithm 5):

1. The insertion does no longer start at the root of the tree for each particle but at the last inserted to node,

2. The COM calculation is deferred until the tree construction is finished and carried out using Equation 23.

The second point is a direct consequence of the first one; if the insertion of a particle does not start at the root but at some other node n, deep in the tree, the nodes lying above n will not be updated so their COM and total mass values will be incorrect. The first point, however, requires more explanation. Since the particles are z-ordered, subsequent inserts into the tree will result in traversing similar paths. Because of that, the point of insertion can be found more efficiently by starting the traversal from the last seen node and backtracking up the tree until a valid insertion point is found. The decision whether to stop backtracking at a given node n is made on the condition that the particle to be inserted is inside a cube represented by n. When the condition is met, the standard insertion procedure takes place starting at n.

The asymptotic time complexity of the modified tree construction algorithm is the same as for the standard one. Sorting the particles requires $O(N \log N)$ operations but inserting a particle into the tree starting at the last-seen node is an $O(1)$ operation. The last claim may raise objections, since some number of backtrack steps is expected on each insert. To verify it, we measured the average number of backtracks per particle in typical simulations and found that it did not exceed 2.5 (this can be compared with the number of backtracks required when the particles are not z-ordered; in such a case, we found that in the tested scenarios, on average more than 8 backtracks were typically needed). Even though the modified algorithm still has the $O(N \log N)$ time complexity, it leads to more predictable memory access patterns which results in noticeably improved performance. The tree construction time as a function of N is shown in Figure 18. For the purposes of generating the graphs shown there, the tree construction time

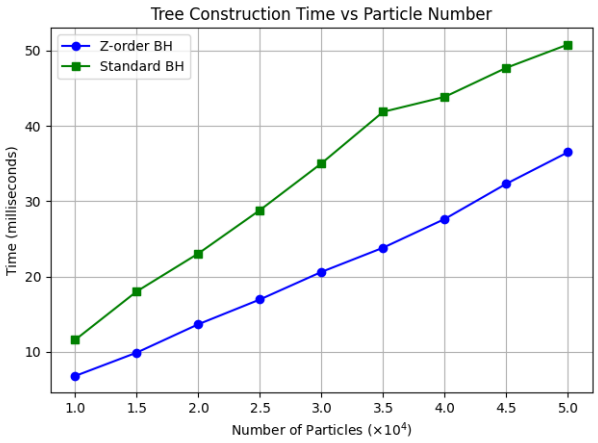


Figure 18: Tree construction time averaged over the number iterations of the simulation.

was measured in a spiral galaxy simulation. As can be seen in the figure, the improved algorithm allows for up to 40% speedup over the standard version. In our tests, we used the standard library procedure `std::sort` with the parallel execution policy, which offered a slight speedup over the sequential variant.

5 Time integration

In the previous sections we described various methods of calculating forces applied to particles in the simulation. Once these forces are found, the evolution of the system in time can be tracked by integrating Newton's 2nd law of motion,

$$\ddot{\mathbf{x}}_i = \frac{\mathbf{F}_i}{m_i}. \tag{25}$$

5.1 Euler's method

Possibly, the simplest numerical method that could be used is Euler's method described by the update rules

$$\begin{aligned} \mathbf{v}_i^{(k+1)} &= \mathbf{v}_i^{(k)} + \mathrm{DT}\frac{\mathbf{F}_i^{(k+1)}}{m_i}, \\ \mathbf{x}_i^{(k+1)} &= \mathbf{x}_i^{(k)} + \mathrm{DT}\mathbf{v}_i^{(k)}. \end{aligned} \tag{26}$$

The method defined in Equation 26 is not suitable for physical simulations, however. Its shortcomings are best illustrated by an example of an undamped pendulum of length l in gravitational field of magnitude g. Although it is a very simple system, it illustrates the numerical challenges faced in gravitational simulations over long timescales, particularly the issue of energy preservation.

The motion of the pendulum is governed by the differential equation

$$\ddot{\theta} = -\frac{g}{l}\sin\theta,$$

and its kinetic and potential energy are given by $\mathrm{KE} = (1/2)ml^2\dot{\theta}$ and $\mathrm{PE} = -mgl\cos\theta$ respectively. As

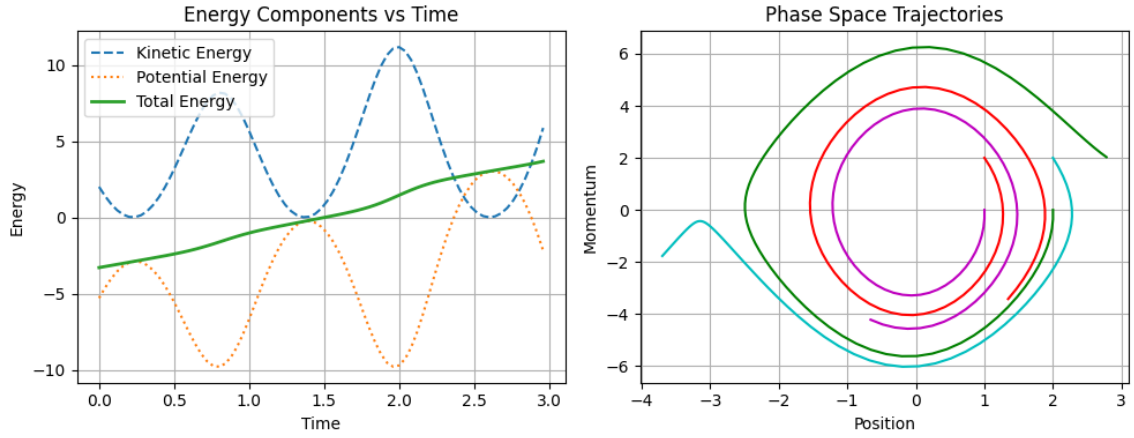


Figure 19: Behavior of Euler's method: lack of conservation of energy and phase space trajectories spiraling out.

shown in Figure 19, Euler's method fails to conserve total energy (PE + KE) and produces trajectories in phase space that are not closed, contrary to expectations for periodic systems. Additionally, the evolution of an area element in phase space violates Liouville's theorem, as described in [11], making the method unsuitable for long-term physical simulations (see Figure 20).

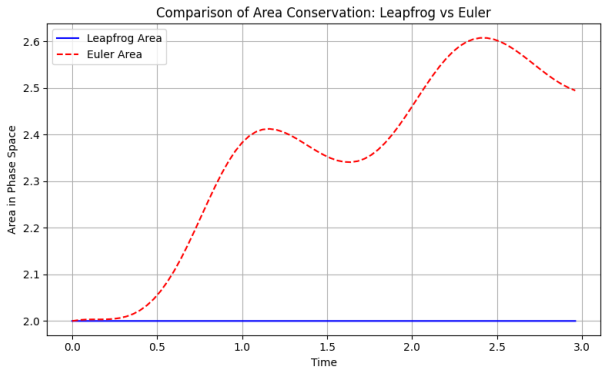


Figure 20: Area in phase space over time. Violation of Liouville's theorem by Euler's method.

5.2 Leapfrog algorithm

The leapfrog algorithm, given by the update rule [18]

$$\begin{aligned} \mathbf{v}_i^{(1/2)} &= \mathbf{v}_i^{(0)} + \frac{1}{2}\mathrm{DT}\frac{\mathbf{F}_i^{(0)}}{m_i^{(0)}}, \\ \mathbf{x}_i^{(k+1)} &= \mathbf{x}_i^{(k)} + \mathrm{DT}\mathbf{v}_i^{(k+1/2)}, \\ \mathbf{v}_i^{(k+3/2)} &= \mathbf{v}_i^{(k+1/2)} + \mathrm{DT}\frac{\mathbf{F}_i^{(k+1)}}{m_i}. \end{aligned} \tag{27}$$

is the preferred way of integrating Equation 25. When applied to the same pendulum system, it conserves energy much more faithfully and preserves the area in phase space, consistent with Liouville's theorem (see Figure 21 and Figure 20). Given its simplicity and excellent long-term energy behavior, we adopt

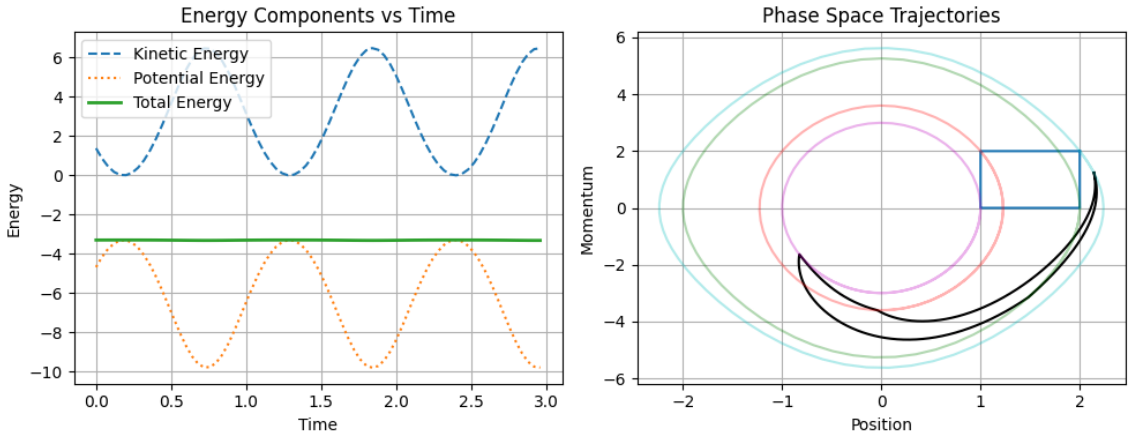


Figure 21: Behavior of the leapfrog algorithm: conservation of energy and phase space trajectories forming closed loops. Evolution of an area element in phase space is shown on the right-hand side: blue rectangle – initial conditions for many copies of the system; black distorted quadrilateral – their state by the end of the simulation.

the leapfrog algorithm to integrate Newton's equations in our program.

6 Galaxy model

The model of a galaxy used as a test bed for the implementation is a simple one. The galaxy is assumed to comprise only two parts: a thin disk and a spherically symmetric halo. The disk comprises a large number of particles, each representing some number of stars. The halo is simulated as a fixed external gravitational field. The schematic illustration of the model is shown in Figure 22.

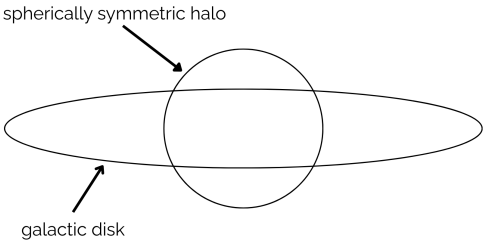


Figure 22: Spiral galaxy model (thin disk and spherical halo).

6.1 Disk

The disk particles are sampled from a radial distribution

$$p(r) = \frac{3}{\pi R_D^2}\left(1 - \frac{r}{R_D}\right),\ z = 0,$$

where R_D is the radius of the disk and $r \leq R_D$. The cumulative distribution function is therefore

$$F(r,\phi) = \int_0^{\phi}\int_0^{r} p(r')r'dr'd\phi' = \frac{\phi}{2\pi R_D^3}(3R_D r^2 - 2r^3)$$

and the marginal CDFs are

$$F_R(r) = F(r, 2\pi) = \frac{1}{R_D^3}(3R_D r^2 - 2r^3) \quad \text{and} \quad F_\Phi(\phi) = F(R_D, \phi) = \frac{\phi}{2\pi}.$$

Now we use inverse transform sampling to generate initial positions (r, ϕ) for the particles, i.e. $\phi = 2\pi u$ and r is given implicitly by $h(r) \equiv 2r^3 - 3R_D r^2 + uR_D^3 = 0$ with $u \sim U(0,1)$. A straightforward calculation shows that $dh/dr < 0$ for $0 < r < R_D$ and $h(0)h(R_D) < 0$ implying that h has exactly one zero between 0 and R_D (which can be found for example using Newton's method).

Strength of the gravitational field $\mathbf{g}_D$ due to the disk at point $\mathbf{x}_0$ lying in the disk is

$$\mathbf{g}_D = G\int_0^{2\pi}\int_0^{R_D} \sigma(r)\frac{\mathbf{x} - \mathbf{x}_0}{|\mathbf{x} - \mathbf{x}_0|^3} r dr d\phi,$$

where $\sigma(r) = \sigma_0(1 - r/R_D)$ describes the density profile of the disk for $r \leq R_D$. If M_D is the total mass of the disk, then $\sigma_0 = 3M_D/(\pi R_D^2)$. By symmetry, the point $\mathbf{x}_0$ may be chosen to lie on the x-axis, i.e. $\mathbf{x}_0 = (-x_0, 0)$, so that $\mathbf{x} - \mathbf{x}_0 = (x_0 + r\cos\phi, r\sin\phi)$. Letting $\bar{r} = r/R_D$ and $\bar{x}_0 = x_0/R_D$, the integral becomes

$$\mathbf{g}_D = G\sigma_0\int_0^{2\pi}\int_0^{1}(1 - \bar{r})\frac{(\bar{x}_0 + \bar{r}\cos\phi, \bar{r}\sin\phi)}{|(\bar{x}_0 + \bar{r}\cos\phi, \bar{r}\sin\phi)|^3}\bar{r}d\bar{r}d\phi.$$

By symmetry $g_{D,y} = 0$ and thus the radial component of the field $\mathbf{g}_D$ at distance $R\bar{x}_0$ from the center is

$$g_{D,r} = -|\mathbf{g}_D| = -G\sigma_0 \int_0^{2\pi} \int_0^1 (1-\bar{r}) \frac{\bar{x}_0 + \bar{r}\cos\phi}{(\bar{x}_0^2 + \bar{r}^2 + 2\bar{x}_0\bar{r}\cos\phi)^{3/2}} \bar{r} d\bar{r} d\phi. \tag{28}$$

If the disk had constant density, $\mathbf{g}_D$ could be expressed in terms of elliptic integrals [15]. However, to the best of the author's knowledge, the integral in Equation 28 cannot be further simplified. For this reason, a crude approximation with a quadratic function is used: $g_{D,r} \approx a(r-h)^2 + k$, where the values $k = 2.5$ and $h = 0.66$ (the maximum of $g_{D,r}$ and the argument thereof) were estimated based on the graph of $g_{D,r}$ (see Figure 23). The value of $a = -k/h^2$ can be found by setting $g_{D,r}(0) = 0$ in the approximate formula.

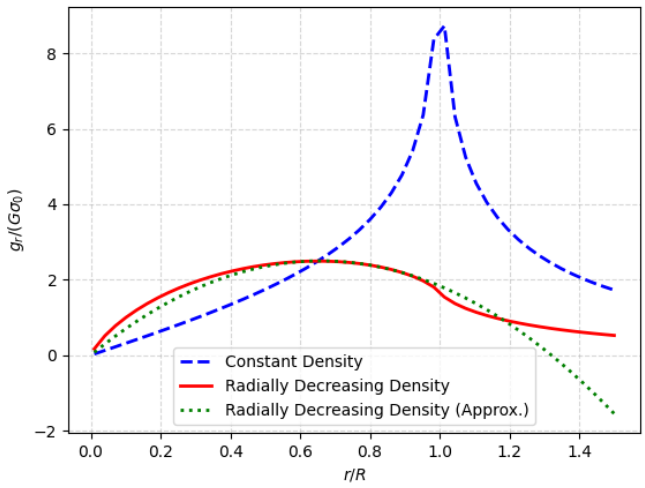


Figure 23: Magnitude of the radial component of the field strength due to a disk. The peak at $r = R$ for the constant density disk is in fact infinite.

6.2 Halo

The density profile of the halo is analogous to the one used for the disk, save for the fact it is 3-dimensional, i.e.

$$\rho(r) = \begin{cases} \rho_0 \left(1 - \frac{r}{R_H}\right), & r \le R_H \\ 0, & \text{otherwise,} \end{cases}$$

where R_H is the radius of the halo. If we let M_H be the mass of the halo, then $\rho_0 = 3M_H/(\pi R_H^3)$. Application of Gauss's law shows that we have

$$g_{H,r} = -GM_H \times \begin{cases} \frac{r}{R_H^3}\left(4 - \frac{3r}{R_H}\right), & r \le R_H \\ \frac{1}{r^2}, & \text{otherwise.} \end{cases}$$

6.3 Initial conditions

The total field $\mathbf{g} = \mathbf{g}_D + \mathbf{g}_H$ is used to find initial velocities for the particles with initial positions $(x, y, 0)$. The formula for the centripetal force yields

$$\frac{v^2}{r} = -g_r$$

and thus

$$\mathbf{v} = \left(-v\frac{y}{r}, v\frac{x}{r}, 0\right)$$

with $v = \sqrt{-rg_r}$ for counter-clockwise rotation.

6.4 Disk with a hole

While testing the implemented methods, we observed that for modeling systems comprising multiple galaxies it may be beneficial to replace the fixed halo with a single massive particle. In such case, the simulation turns out to stable if there are no particles in close vicinity to the galaxy center. This means that the galactic disk ought to have a hole of radius r_0. Using an approach analogous to the one used in subsection 6.1, we obtain the following relations that can be used for sampling points (ϕ, r) in polar coordinates

$$\phi = 2\pi u \quad \text{and} \quad 2(r^3 - r_0^3) - 3R_D(r^2 - r_0^2) + (R_D - r_0)^2(2r_0 + R_D)u = 0,$$

where $u \sim U(0, 1)$.

Getting rid of the external halo field simplifies the treatment of complex systems significantly, as we do not need to keep track of the movement of the halo. The massive particle in the galaxy center, which effectively replaces it, can be treated just as any other particle in the simulation.

7 Globular cluster model

A globular cluster is a formation comprising a large number stars, closely packed in a spherically symmetric form [12]. In this work, we decided to use the implemented methods to simulate such structure using the Plummer model. The model was chosen due to its simplicity and abundance of dedicated resources.

In the Plummer model, the density of a cluster is given by

$$\rho(r) = \frac{3M}{4\pi a^2}\left(1 + \frac{r^2}{a^2}\right)^{-5/2}, \tag{29}$$

where the parameter a controls the spread of the distribution (the size of the cluster core) [1]. Hence, the particles are sampled from a distribution with PDF $p(r) = \rho(r)/M$. Similarly to the galaxy model, we use inverse transform sampling to initialize the positions of the particles. The marginal CDFs are easily calculated as

$$F_R(r) = \frac{r^3}{a^3}\left(1 + \frac{r^2}{a^2}\right)^{-3/2}, \quad F_\Theta(\theta) = \frac{1}{2}(1 - \cos\theta), \quad F_\Phi(\phi) = \frac{\phi}{2\pi},$$

which means that given a random variable $u \sim U(0,1)$, we can generate random points with spherical coordinates

$$r = a(u^{-2/3} - 1)^{-1/2}, \quad \theta = \cos^{-1}(1 - 2u), \quad \phi = 2\pi u, \tag{30}$$

consistent with Equation 29 (assuming equal mass of all particles).

In the Plummer model, the potential is given by

$$\phi(r) = -\frac{GM}{\sqrt{r^2 + a^2}}.$$

This allows us to find the escape velocity v_e at distance r from the center. Using the law of conservation of energy, we get

$$\frac{1}{2}v_e + \phi(r) = 0 \Rightarrow v_e = \sqrt{-2\phi(r)}.$$

For any r, the probability distribution of $q \equiv v/v_e$ is given by [1]

$$g(q) = Nq^2(1 - q^2)^{7/2}, \tag{31}$$

where N is a normalization constant, which can be calculated to be $N = 512/(7\pi)$. The CDF of the distribution in Equation 31, $G(q) = \int_0^q g(q')dq'$, can be determined using symbolic integration. The algebraic expression that one obtains in this way is lengthy and we will not cite it here. The random values of q, consistent with the PDF in Equation 31, are again obtained using inverse transform sampling; the equation $G(q) = u$ is solved for q by finding the roots of $G(q) - u$ using Newton's method. Finally, the magnitude v of the velocity vector $\mathbf{v}$ is set to $v = qv_e$. The direction of $\mathbf{v} = (v_x, v_y, v_z)$ is chosen uniformly at random, i.e.

$$v_x = v\sin\theta\cos\phi, \quad v_y = v\sin\theta\sin\phi, \quad v_z = v\cos\theta,$$

where θ and ϕ are generated in the same way as in Equation 30.

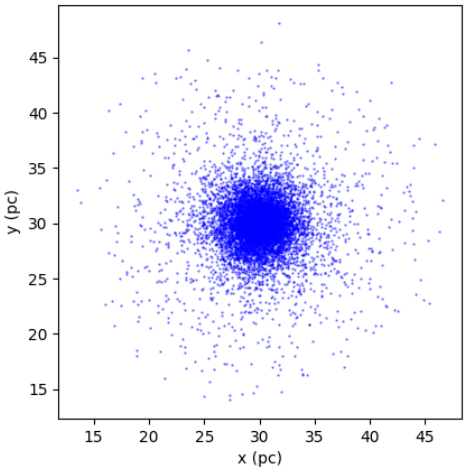
The comparison of particle positions generated using the aforementioned method with an example of a real-world globular cluster is shown in Figure 24.

8 Results

The parameters used in the simulation of a spiral galaxy are shown in Table 1. The galaxy is simulated as an isolated system, however, in deriving Equation 8, periodic boundary conditions were assumed. The simplest way (and the one used) to obtain a free-space solution from the PM method is to extend the computational domain twice in every dimension and fill the space unused in mass distribution with zeros. The total size of the potential mesh used was $128 \times 128 \times 64$ with the region of interest occupying a box of size $60\,\text{kpc} \times 60\,\text{kpc} \times 30\,\text{kpc}$ located in a $64 \times 64 \times 32$ octant of the mesh.

8.1 Particle-mesh method

In the PM method, $N = 50,000$ particles were used. Cell size H and time-step length were set to $60/64 = 0.9375$ kpc, and 1 Myr respectively. The system's evolution over 200 Myr is shown in Figure 25.



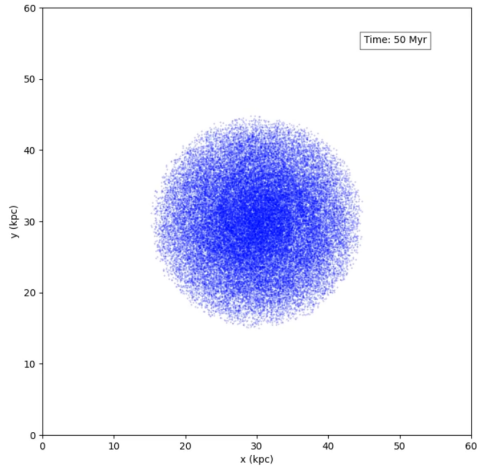
(a) Generated data ($M = 10^6 M_\odot$, $a = 2$ pc, $N = 10,000$).

(b) Real-world globular cluster (Messier 13). Credit: Giuseppe Donatiello

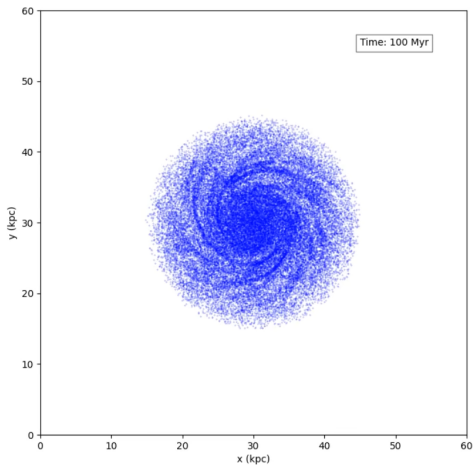
Figure 24: The particle positions generated using the described model compared to a real-world globular cluster.

Parameter	**Value**
Halo radius	3 kpc
Halo mass	$60 \times 10^9 M_\odot$
Disk radius	15 kpc
Disk mass	$15 \times 10^9 M_\odot$
Disk thickness	0.3 kpc
Disk density profile	Uniformly decreasing
Mass assignment scheme	TSC
Finite difference	Two-point
Time integration method	Leapfrog

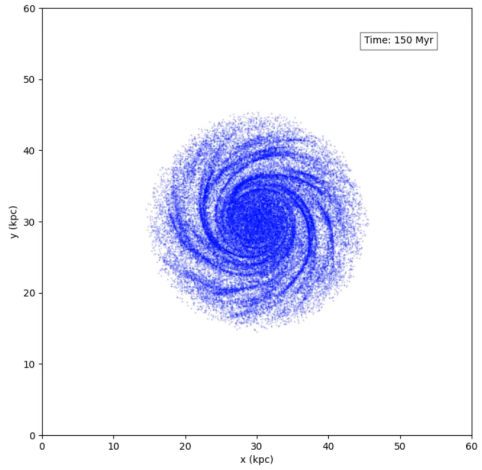
Table 1: Galaxy model parameters used in the simulation.



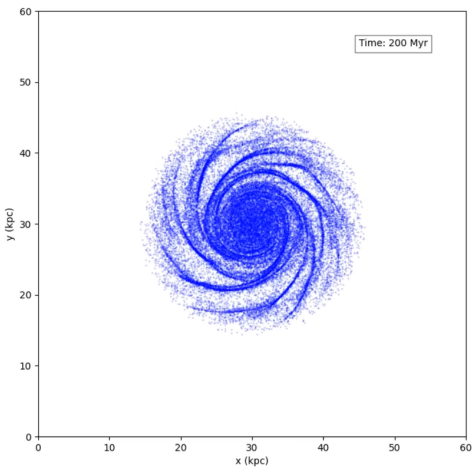
(a) $t = 50\,\mathrm{Myr}$



(b) $t = 100\,\mathrm{Myr}$



(c) $t = 150\,\mathrm{Myr}$



(d) $t = 200\,\mathrm{Myr}$

Figure 25: Evolution of a spiral galaxy as predicted by the PM method.

During the simulation, total energy $E = \mathrm{KE} + \mathrm{PE}$, angular momentum $\mathbf{l}$, and the z-component of the momentum vector $\mathbf{p}$ should stay constant. The x- and y-components of momentum change due to the presence of an external gravitational field (representing the halo). We can verify if this variation satisfies the expected relation

$$\dot{\mathbf{p}} = \mathbf{F}^{\mathrm{ext}} \tag{32}$$

by finding the initial total momentum $\mathbf{p}(t = 0)$ and incrementing the value of $\mathbf{p}$ in each time-step by $\mathbf{F}^{\mathrm{ext}}\mathrm{DT}$.

The exact calculation of the potential energy [11] using the formula

$$\mathrm{PE} = -\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{Gm_im_j}{r_{ij}}$$

is computationally infeasible considering the $O(N^2)$ cost. An approximation based on the potential values at mesh points,

$$\mathrm{PE} \approx \frac{V}{2}\sum_{\mathbf{p}}\rho(\mathbf{x_p})\phi(\mathbf{x_p}),$$

is used instead (for derivation refer to [7]).

8.2 Particle-particle particle-mesh method

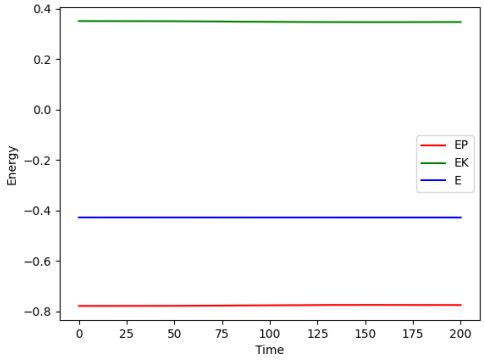
The $\mathrm{P^3M}$ based simulation uses the same parameters as the PM method. The reference force was calculated using the S_1 shape formula (Equation 18) with particle diameter $a = 3H$. The cutoff radius was set to $r_e = 0.7a$. One extra free parameter is the *softening length* ϵ which modifies the universal law of gravitation so that division by zero can be avoided, i.e. the modified law is

$$F_{ij}^{\mathrm{soft}}(r) = \frac{Gm_im_j}{r_{ij}^2 + \epsilon^2}.$$

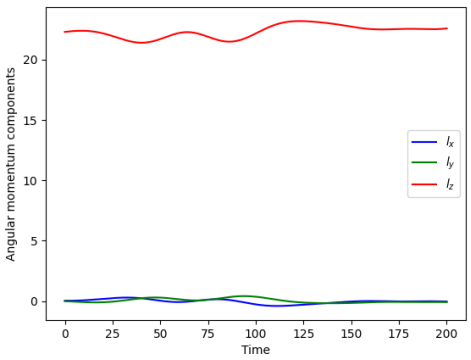
In the simulation, ϵ was set arbitrarily to 1.5 kpc. The system's evolution is presented in Figure 27. Graphs of energy, angular momentum, and momentum components vs. time are shown in Figure 28.

8.3 Performance analysis

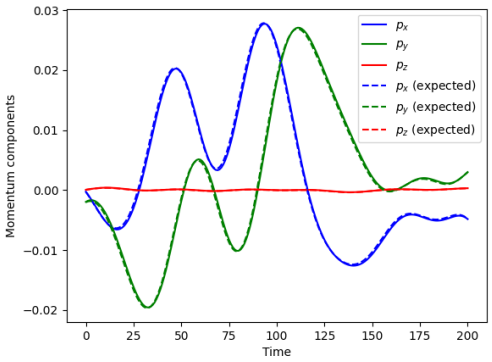
The PM and $\mathrm{P^3M}$ methods were implemented exactly as described in the previous sections. The PM method was developed for both CPU and GPU architectures, using C++ and CUDA C++, respectively. The implementation relies on external libraries for fast Fourier transform computations: FFTW for the CPU version and cuFFT for the GPU version. A performance comparison of the PM method was conducted with N ranging between 50,000 and 1,000,000 over 200 iterations (TSC mass assignment and two-point finite difference). The tests were run on a system equipped with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz and an NVIDIA GeForce GTX 1650 GPU. For the purposes of performance evaluation, the parts of the code responsible for diagnostics collection (energy, momentum, etc.) were switched off. Since disk IO (saving simulation state) was the most time-consuming part of both the CPU and GPU implementation, only the final state of the simulation was saved in these tests. This means that data transfers from device to host were also not taken into account. The results of the test are displayed in Figure 8.



(a) Energy

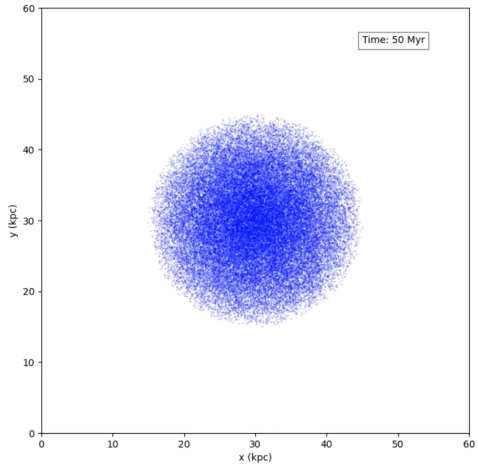


(b) Angular momentum

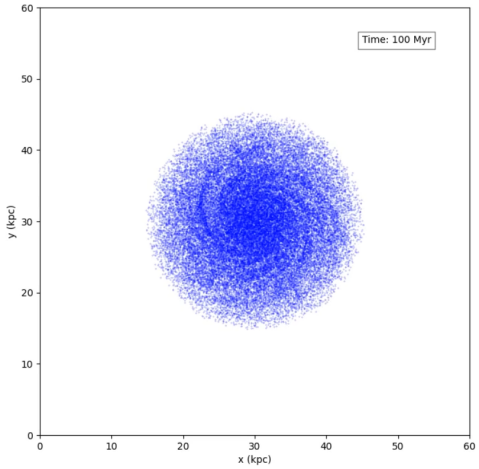


(c) Momentum; broken lines represent the expected momentum following Equation 32

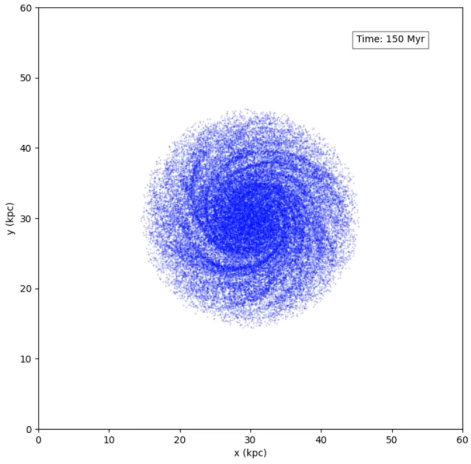
Figure 26: Fundamental physical quantities describing the system over time in the PM simulation. Time is in Myr and the quantities are expressed in units consistent with Table 1



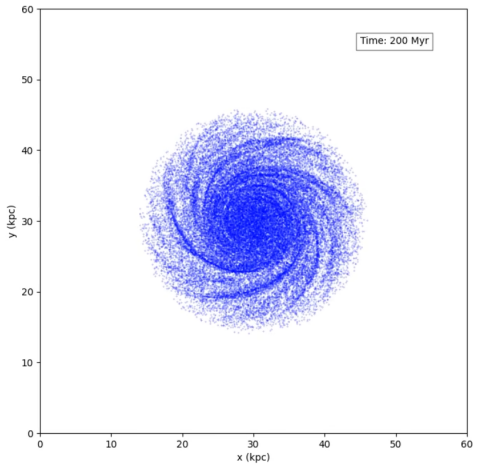
(a) $t = 50\,\mathrm{Myr}$



(b) $t = 100\,\mathrm{Myr}$

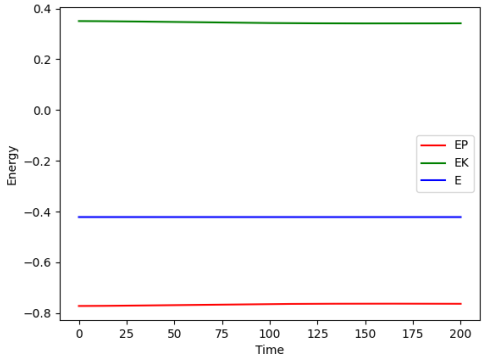


(c) $t = 150\,\mathrm{Myr}$

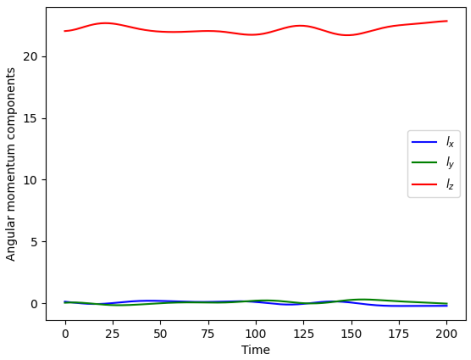


(d) $t = 200\,\mathrm{Myr}$

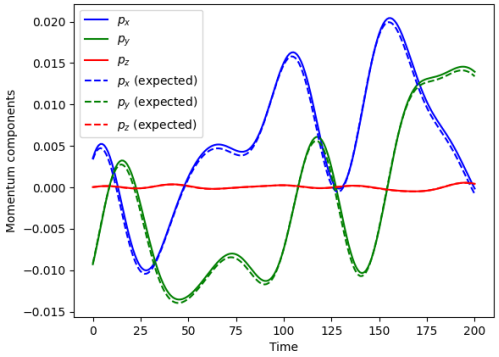
Figure 27: Evolution of a spiral galaxy as predicted by the $\mathrm{P^3M}$ method.



(a) Energy



(b) Angular momentum



(c) Momentum; broken lines represent the expected momentum following Equation 32

Figure 28: Fundamental physical quantities describing the system over time in the P^3M simulation. Time is in Myr and the quantities are expressed in units consistent with Table 1

For the P^3M method, performance was measured using $N = 50,000$ particles on a $128 \times 128 \times 64$ mesh with the TSC assignment scheme. The total runtime was approximately 1 minute and 30 seconds, with the time distribution among key algorithm components as follows:

- HOC table initialization: 12%
- Short-range force calculations: 80%
- PM step: 7.5%

The code is available at `https://github.com/AleksyBalazinski/ParticleSimulation` under the MIT license.

References

[1] S. J. Aarseth, M. Henon, and R. Wielen. A comparison of numerical methods for the study of star cluster dynamics. *Astronomy and Astrophysics*, 37(1):183–187, December 1974.

[2] Martin Aigner and Günter M. Ziegler. *Proofs from THE BOOK*. Springer, Berlin, Germany, 6 edition, 2018.

[3] E. Athanassoula, E. Fady, J. C. Lambert, and A. Bosma. Optimal softening for force calculations in collisionless n-body simulations. *Monthly Notices of the Royal Astronomical Society*, 314(3):475–488, 05 2000.

[4] Laurent Demanet, Jacob White, and Richard Zhang. Fourier-based fast methods for ordinary differential equations. `https://web.archive.org/web/20240802094957/https://math.mit.edu/icg/resources/teaching/18.336-spring2013/Notes_Set2.pdf`, February 2013. Lecture notes for 18.336/6.335: Fast Methods for Partial Differential and Integral Equations, MIT, February 7–14.

[5] I. M. Gelfand and G. E. Shilov. *Generalized Functions, Vol. 1: Properties and Operations*. Academic Press, 1964. Originally published in Russian; this volume introduces the theory of generalized functions and includes topics such as Fourier transforms, homogeneous distributions, and distributions on submanifolds.

[6] Lars Hernquist. Performance characteristics of tree codes. *Astrophysical Journal Supplement Series*, 64:715, August 1987.

[7] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. CRC Press, 1st edition, 1988.

[8] Andrey Kravtsov. Writing a pm code, March 2002. Accessed: 2025-04-03.

[9] NVIDIA Corporation. *CUDA C++ Programming Guide*. NVIDIA Corporation, release 12.9 edition, may 2025. Last accessed May 29, 2025.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3rd edition, 2007.

[11] John R. Taylor. *Classical Mechanics*. University Science Books, 2005.

[12] The Editors of Encyclopaedia Britannica. Globular cluster. `https://www.britannica.com/science/globular-cluster`, January 2024. Encyclopaedia Britannica.

[13] M. Trenti and P. Hut. Gravitational n-body simulations, 2008.

[14] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Supercomputing '93, pages 12–21, New York, NY, USA, 1993. Association for Computing Machinery.

[15] J. Weiss. Certain aspects of the gravitational field of a disk. *Applied Mathematics*, 9:1360–1377, 2018.

[16] Eric W. Weisstein. Plane-filling function. `https://mathworld.wolfram.com/Plane-FillingFunction.html`, n.d. From MathWorld–A Wolfram Web Resource.

[17] Kelly Young. Andromeda galaxy hosts a trillion stars. *New Scientist*, May 2006. Accessed: 2025-05-25.

[18] Peter Young. Leapfrog method and other "symplectic" algorithms for integrating newton's laws of motion, 2019. Physics 115/242 Course Notes, UC San Diego.

[19] Tianchi Zhang, Shihong Liao, Ming Li, and Liang Gao. The optimal gravitational softening length for cosmological n-body simulations. *Monthly Notices of the Royal Astronomical Society*, 487(1):1227–1232, May 2019.