Lecture

2

Basic Principles of the Hierarchical Tree Method

2.1 Tree Construction

We have seen in the preceding chapter that in grid-based codes the particles interact via some averaged density distribution. This enables one to calculate the influence of a number of particles represented by a cell on its neighbouring cells. Problems occur if the density contrast in the simulation becomes very large or the geometry of the problem is very complex.

So why does one bother with a grid at all and not just calculate the inter-particle forces? The answer is simply that the computational effort involved quite dramatically limits the number of particles that can be simulated. Particularly with $1/r$-type potentials, calculating each particle–particle interaction requires an unnecessary amount of work because the individual contributions of distant particles is small. On the other hand, gridless codes cannot distinguish between near-neighbours and more distant particles; each particle is given the same weighting.

Ideally, the calculation would be performed without a grid in the usual sense, but with some division of the physical space that maintains a relationship between each particle and its neighbours. The force could then be calculated by direct integration while combining increasingly large groups of particles at larger distances. Barnes and Hut (1986) observed that this works in the same way that humans interact with neighbouring individuals, more distant villages, and larger states and countries. A resident of Lower-Wobbleton, Kent, England, is unlikely to undertake a trip to Oberfriedrichsheim, Bavaria, Germany, for a beer and to catch up on the local gossip.

Independently, in the early 1980s, several workers attempted to implement this kind of hierarchical grouping in N-body codes (Appel 1985, Jernighan 1985, Porter 1985). Although these early hierarchical codes had a nominal $N \log N$ dependence of the computation time, additional errors were introduced

that were hard to analyse because of the arbitrary structure of the tree. Complicated bookkeeping was required to 'reconnect' groups of near neighbours, with the result that the $N \log N$ scaling could only be conjectured.

Barnes and Hut (1986) introduced a scheme which avoided these complications. Its systematic division of the physical space has since become the basis of most hierarchical tree codes and their refinements. These codes are sometimes described as octagonal tree codes to distinguish them from so-called binary tree codes. Press (1986) and Benz (1988) have developed binary trees, based on nearest neighbour pairs, and Jernighan and Porter (1989) devised an integration method in which the particles and the nodes of the tree are the basic dynamical units. Although such binary trees might reflect the structure of the system more closely, the Barnes and Hut method is by far the most commonly used method due to its conceptual simplicity and straightforward tree construction. Therefore, we will restrict our discussion to 'oct-trees' throughout this book.

In their original work, Barnes and Hut (1986) began with an empty cubical cell that was big enough to contain the whole system of particles. Particles are placed one by one into this 'root' cell. If any two particles fall into the same cell (which happens as soon as the second particle has been loaded into the root cell), the cell is divided into daughter cells having exactly half the length, breadth, and width of their parent.

This means that in 3D, the system is split into eight pieces. If the two particles are still in the same daughter cell, this cell is recursively subdivided in the same way until the particles sit in different boxes. Then, the next particle is loaded and the same procedure starts again, except that the starting point is the level just above the root cell (because the latter has already been subdivided). When all N particles have been loaded, the system space will have been partitioned into a number of cubical cells of different sizes, with at most one particle per cell.

For numerical reasons which will be discussed later, most algorithms do not start with an empty box, but with the root cell containing all particles of the simulation (Hernquist 1988). As before, this cell is then divided into its eight daughter cells. For each cell one asks the question: How many particles are there in the cell – 0, 1, or > 1?

- If the cell is empty, this cell is ignored.
- If there is one particle in the cell, this is stored as a 'leaf' node in the tree structure.
- If there are more particles in a cell, this cell is stored as a 'twig' node and subdivided.

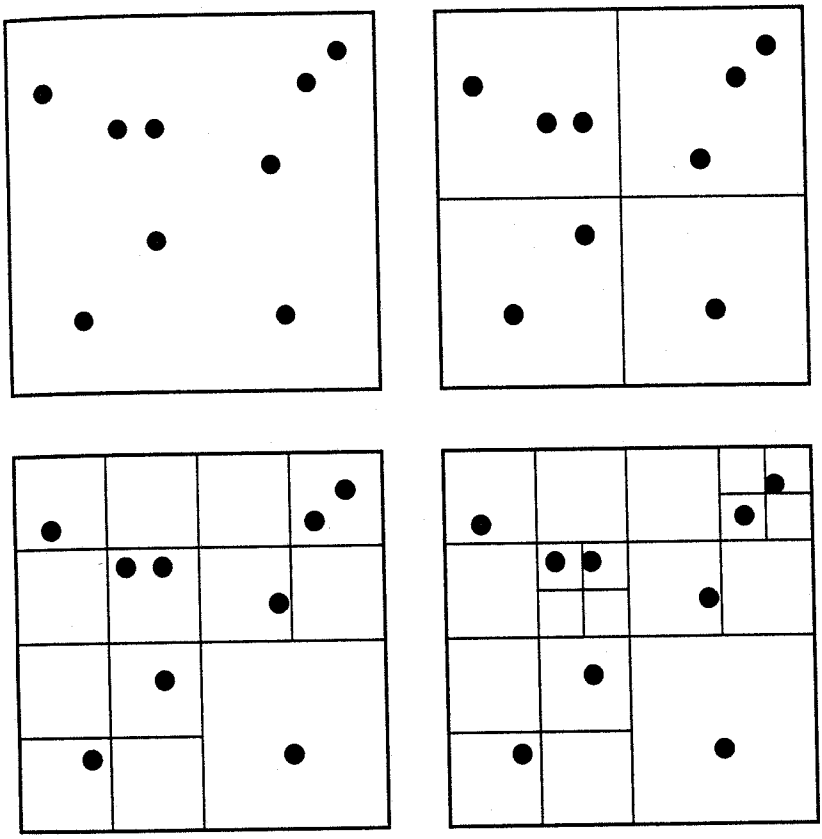


Fig. 2.1. Step-by-step division of space for a simple 2-D particle distribution.

The subdivision process continues until there are no cells with more than one particle left, which ultimately leads to an identical structure as the Barnes–Hut (BH) algorithm. Figure 2.1 shows the end-result of the space-division. For ease of illustration, a 2-dimensional example has been chosen; this means that one has always four daughter cells instead of eight.

Figure 2.2 shows the end-product of the division of space in three dimensions for the example of a star cluster (galaxy). Galaxies usually have a steep gradient in the particle distribution; they may be very dense at the centre relative to their outer reaches. While this is a source of difficulty for simulations with grid-based codes like PIC, it can be seen that the subdivision of space by the tree method *automatically* adjusts to the particle distribution.

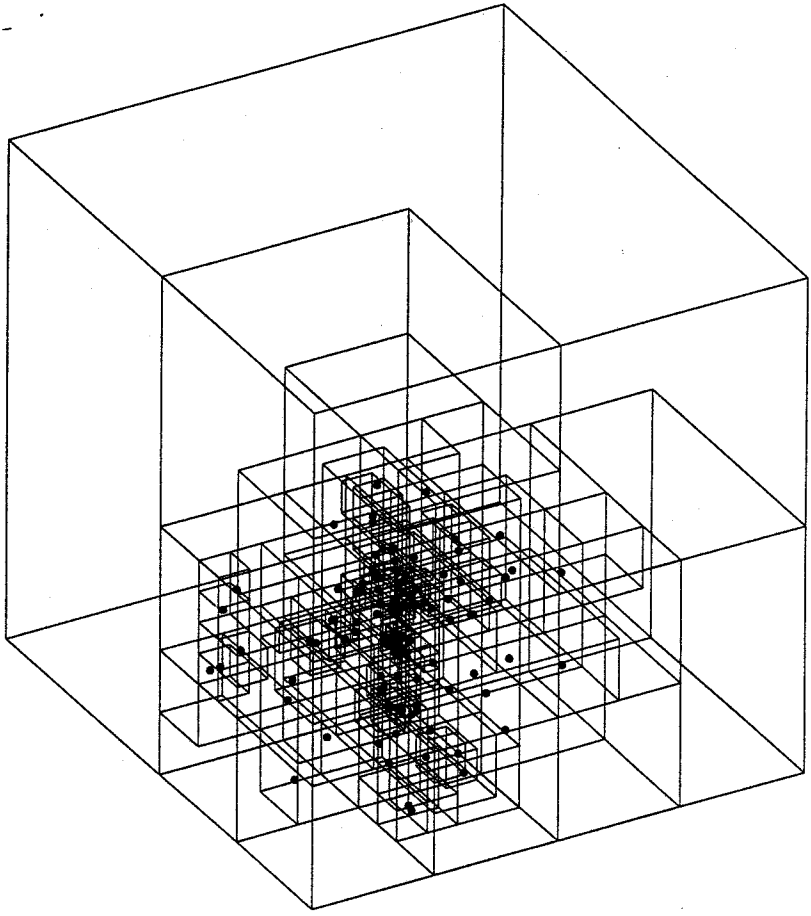


Fig. 2.2. Example of the division of space in three dimensions for a star cluster.

However, the division of space just described is not used like a grid, but as a bookkeeping structure for the tree. Figure 2.3 illustrates this relationship explicitly. The 'root' is the basic cell to start from, which contains all the particles of the simulation. At each division step the tree data structure is augmented with the next level in the hierarchy. Each node in the tree is associated with a cubic volume of space containing a given number of particles; empty cells are not stored.

The root cell is the biggest 'twig'. In our example in Fig. 2.3, the first division would lead to four nonempty cells:

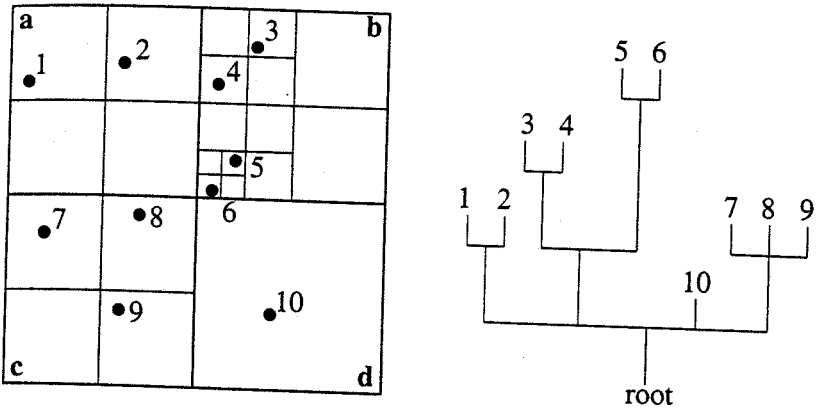


Fig. 2.3. Relationship between spatial division and the resulting tree structure.

- Cell **a** contains particles 1 and 2.
- Cell **b** contains particles 3, 4, 5, 6.
- Cell **c** contains particles 7, 8, 9.
- Cell **d** contains particle 10.

For the root, an identifier as twig as well as the number of nonempty daughter cells (here four daughters) have to be stored. Cell **d** contains only one particle, which means it is a leaf and needs no further processing. By contrast, cells **a**, **b**, and **c** are twigs, and are subdivided further.

For the leaf, an identifier as a leaf, a pointer to the parent cell (here the root), and the particle label have to be stored. This label will be the link to the physical quantities of the particle, like its position, mass, and charge. For a twig, the same information as for the root has to be stored *plus* a pointer to its parent cell. Having accumulated this information, the first level of division is finished. At the next level, the division of cell **a** leads to the creation of leaves 1 and 2 and the division of cell **c** leads to leaves 7, 8, and 9, whereas the division of cell **b** leads to two new twigs. Continuing this process, we finally end up with only leaves at the ends of the twigs. The whole data structure of the tree is thus in place.

Figure 2.4 shows a flowchart of the division process and the tree building. To actually convert this algorithm into a computer code, a number of arrays are necessary to define the tree structure with its cross references. In order to avoid being too language-specific, we will not show any explicit coding at this point,

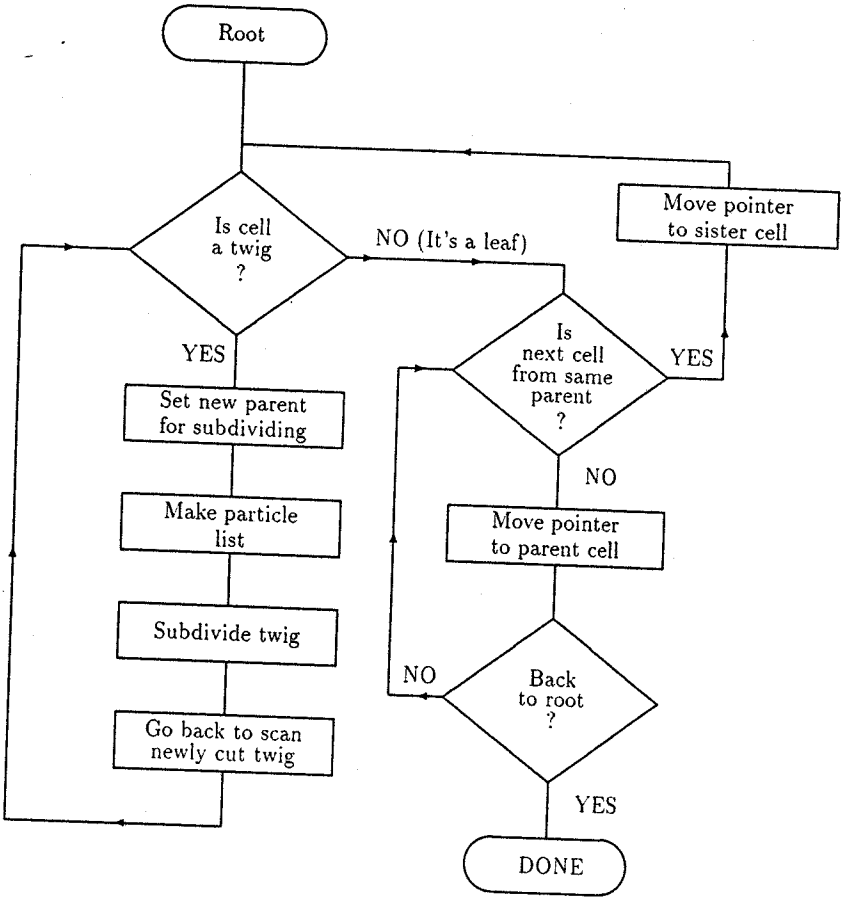


Fig. 2.4. Flowchart of tree building.

but it is perhaps instructive to consider the most important arrays for the example in Fig. 2.3. As the tree is built, each node is given a label `itwig` = (-1, -2, -3, ...) for the twigs and `ileaf` = (1, 2, 3, 4, ...) for the leaves. The root has a label 0. A pointer provides a link between the node number and the position in the array. Each node's parent and level of refinement is also stored. For each twig node, it is also useful to store the position of the 1st daughter node and the number of daughter nodes. For example, the command 'subdivide twig' in Fig. 2.4 actually refers to a subroutine containing the following operations:

```
SUBDIVIDE(cut-node,pointer)

        iparent = cut-node
        ipoi = pointer
        ilev = level(iparent) + 1
        1st_dau(iparent) = ipoi

        count particles in subcells
        n_dau(iparent) = # new daughter nodes

        for each twig-node do:
            itwig = itwig - 1
            node(ipoi) = itwig
            parent(ipoi) = iparent
            level(itwig) = ilev
            ipoi = ipoi + 1
        end do

        for each leaf-node do:
            ileaf = ileaf + 1
            node(ipoi) = ileaf
            parent(ipoi) = iparent
            plabel(ileaf) = particle #
            level(ileaf) = ilev
            store position and moments of particle
            ipoi = ipoi + 1
        end do
```

Before entering the SUBDIVIDE routine, the variable `pointer` is set at the end of the tree. On exit, it will point to the first node (twig or leaf) created in that subdivision. If this node is a twig, SUBDIVIDE is called again, otherwise we check for 'sister' nodes or descend back to the root. The arrays for the completed tree corresponding to Fig. 2.3 are shown in Table 2.1. Each of the intermediate horizontal lines in the table marks the beginning of a subdivision, that is: the point where SUBDIVIDE is called. The arrays `1st_dau` and `n_dau` are filled in on the next level up, after the node has been subdivided.

A rough estimate of how many divisions are needed to reach a typical cell, starting from the root, can be obtained from the average size of a cell containing one or more particles. The average volume of such a cell is the volume of the

Table 2.1. *Tree arrays corresponding to Fig. 2.3*

pointer	level	node	parent	1st_dau	n_dau	plabel
1	0	0	0	2	4	–
2	1	-1	0	6	2	–
3	1	-2	0	8	2	–
4	1	-3	0	15	3	–
5	1	1	0	–	–	10
6	2	2	-1	–	–	1
7	2	3	-1	–	–	2
8	2	-4	-2	10	2	–
9	2	-5	-2	12	1	–
10	3	4	-4	–	–	3
11	3	5	-4	–	–	4
12	3	-6	-5	13	1	–
13	4	6	-6	–	–	6
14	4	7	-6	–	–	5
15	2	8	-3	–	–	7
16	2	9	-3	–	–	8
17	2	10	-3	–	–	9

root cell V divided by the number of simulation particles N. Moreover, the average length of a cell is a power of $V^{1/3}/2$. Therefore,

$$\left(\frac{1}{N}\right)^{1/3} = \left(\frac{1}{2}\right)^{x},$$

which means that the height x of the tree is of the order $\log_2 N^{1/3}$. This is equivalent to

$$\log_2 N^{1/3} = \frac{1}{3\log 2}\log N \simeq \log N. \tag{2.1}$$

Starting from the root, an average of $O(\log N)$ divisions are necessary to reach a given leaf. The tree contains N leaves, therefore the time required to construct the tree is $O(N \log N)$.

As shown in the second loop of the SUBDIVIDE routine, an identifier (i.e., a numerical label) of the particle is stored for every leaf, which provides the link to the physical properties of the particle such as position, mass, and charge. Once the tree structure is completed, equivalents of these quantities, like the

centre of mass, the sum of the masses, and the sum of the charges, still have to be calculated and stored for the twigs as well, because this information will be needed for the force calculation. This 'loading' of twig-nodes can be performed by propagating information down the tree from individual particles (leaves) towards the root. The tree structure can be used to find out which leaves and twigs belong to a given parent. The total mass is simply calculated by a sum over the masses of the daughter cells

$$m_{parent} = \sum_i m_i (daughters) \tag{2.2}$$

and the centre of mass by the sum

$$\mathbf{r}_{parent} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i}. \tag{2.3}$$

Once these quantities have been defined for a twig-node, we can use them instead of the particle distribution inside it, that is, it can be regarded as a 'pseudoparticle'. Proceeding down the tree, the total mass and the centre of mass of the pseudoparticle can be used to define the pseudoparticles on the next lower level of division, as shown in the following MOMENTS routine.

MOMENTS

```
ilevel = levmax - 1
itwig = -ntwig

for each level do:
    repeat for each node on same level:
        pointer = ipoint(itwig)
        nbuds = ndau(pointer)
        point1 = 1st_dau(pointer)
        zero moment sums:
        M(itwig) = 0
        r_com(itwig) = 0

        do i = 1,nbuds
            point_dau = point1 + i - 1
            inode = node(point_dau)
            sum mass and centre of mass:
            M(itwig) = M(itwig) + M(inode)
            r_com(itwig) = r_com(itwig) + r_c(inode)
        end do
```

```
            r_com(itwig) = r_com(itwig)/M(itwig)

            itwig = itwig + 1
        until level done
        ilevel = ilevel + 1
    end do
```

The variable definitions are the same as in Table 2.1. The simple loop over the daughter nodes is made possible because during the tree construction they are stored sequentially (see routine SUBDIVIDE). Eventually one reaches the root cell, the total mass of which is the same as the mass of the whole system. This is an excellent check to see that the tree structure is correct! Figure 2.5 illustrates graphically how the MOMENTS routine is used to calculate the total mass (denoted by the size of the symbol) and the centre of mass from the particles (black) through all levels for the pseudoparticles (grey).

2.2 Force Calculation

As we have seen, the tree structure needs a computational time of $O(N \log N)$ to build it. This structure is now used to perform the force calculation, which, as we will see, also requires a time $O(N \log N)$. The basic idea is to include contributions from near particles by a direct sum whereas the influence of remote particles is taken into account only by including larger cells, that is, pseudoparticles, representing many individual particles. This means that the advantages of the particle–particle technique are retained, but the computation time can be reduced significantly, especially for large numbers of particles (see Chapter 4). In contrast to grid-based codes, this time saving is achieved not by compromising the spatial resolution and/or imposing geometrical restrictions, but by introducing approximations into the calculation of the potential. This procedure is physically well motivated. For example, the dynamics of our solar system is insensitive to the detailed mass distribution of each planet. Furthermore, in all N-body simulations errors occur due to round-off, truncation, and discreteness effects, which makes it unreasonable to compute the potential field to extremely high precision. It is sufficient to require that the error in approximating the potential by the tree algorithm should be of the same order as these numerical errors. In dynamical applications one can often relax this requirement further because the errors introduced by finite timesteps in the integration scheme tend to dominate anyway.

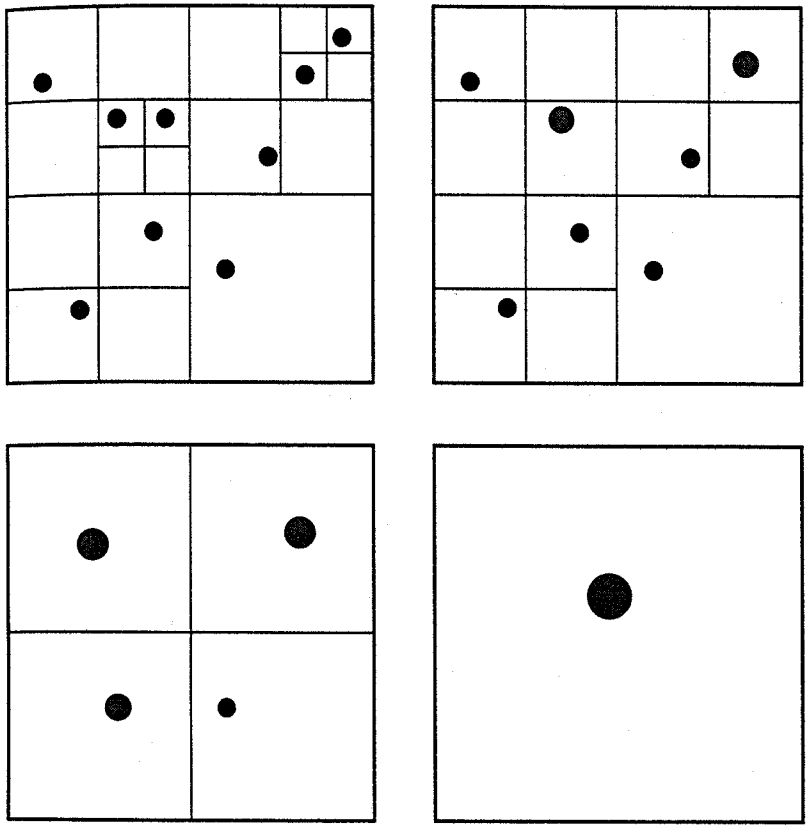


Fig. 2.5. Total mass and centre of mass from the particles (black) for the pseudoparticles (grey) at the different levels of the tree construction. The size of the symbol is proportional to the mass of the pseudoparticle.

The tree structure provides the means to distinguish between close particles and distant particles without actually calculating the distance between every particle. The force between near particles is calculated directly whereas more distant particles are grouped together to pseudoparticles. How do we actually decide when to group?

There is actually some flexibility in the way this is done, but we will start with the simplest method, introduced by Barnes and Hut (1986). Alternatives to the basic 's/d' criterion have recently been proposed by Salmon and Warren (1994), who demonstrate some potential problems with this choice by calculating 'worst-case' errors for various scenarios. A more detailed discussion of alternative criteria is deferred to Section 4.4.

For each particle the force calculation begins at the root of the tree. The 'size' of the current node (or twig), s, is compared with the distance from the particle, d. Figure 2.6 illustrates this comparison. If the relation

$$s/d \leq \theta \tag{2.4}$$

is fulfilled, where θ is a fixed tolerance parameter, then the internal structure of the pseudoparticle is ignored and its force contribution is added to the cumulative total for that particle. Otherwise, this node is resolved into its daughter nodes, each of which is recursively examined according to (2.4) and, if necessary, subdivided. Figure 2.7 shows a flowchart which demonstrates how the tree structure is used to calculate the forces. The node is subdivided by continuing the ascent through the tree until *either* the tolerance criterion is satisfied *or* a leaf-node is reached.

For nonzero θ the time required by the CPU (central processing unit) to compute the force on a single particle scales as $O(N \log N)$. Hernquist (1988) demonstrated this by using a simplified geometry. Consider a single particle at the centre of a homogeneous spherical particle distribution, with the number density n being

$$n = N \Big/ \left(\frac{4\pi}{3} R^3\right),$$

where R is the radius of the whole system. Organising these particles in a hierarchical structure of cells, there will be an inner sphere where the force of each particle on the centre particle is directly calculated. The surrounding concentric shells will contain progressively larger pseudoparticles. Figure 2.8 illustrates this situation. The total number of interactions n_{int} can be estimated by

$$n_{int} = n_o + \sum_{shells} n^i_{sub},$$

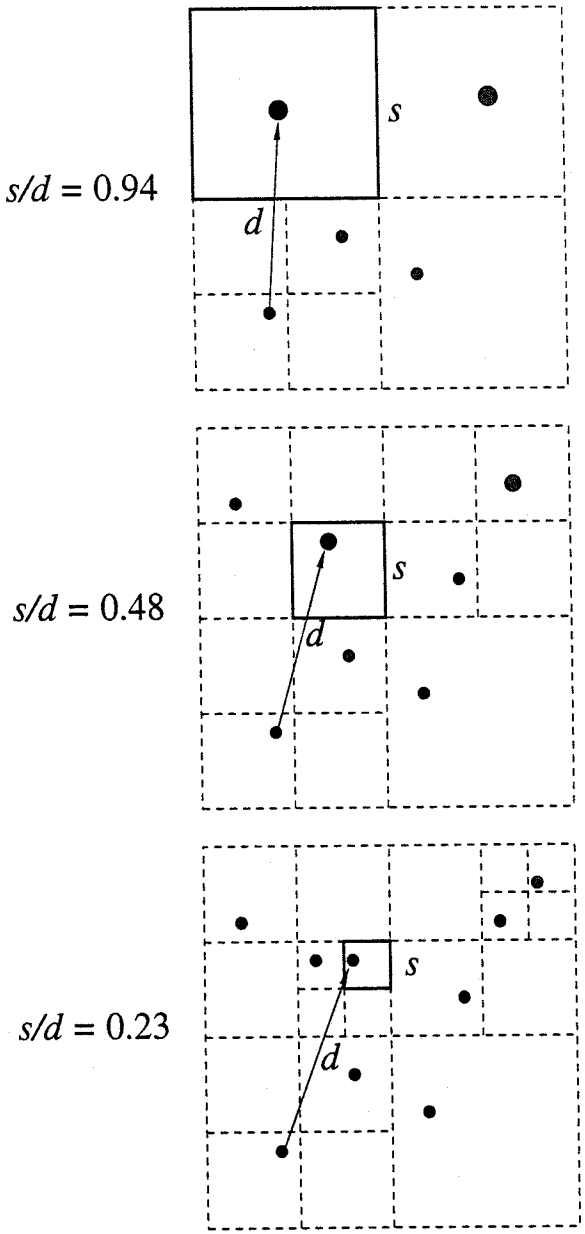
where n_o is the number of direct interactions and n_{sub} is the number of subunits in each shell, which is roughly given by

$$n^i_{sub} \sim \frac{4\pi r_i^3 \theta}{\frac{4\pi}{3} r_i^3 \theta^3 / 8} = \frac{24}{\theta^2},$$

therefore

$$n_{int} \sim n_o + \frac{24}{\theta^2} n_{sh},$$

where n_{sh} is the total number of shells. The inner sphere, where only direct interactions are considered, has a radius of $r_1 = n^{-1/3}/\theta$, which leads to the



$\theta = \frac{s}{d - l_{CM}}$

corrected

Fig. 2.6. The relation s/d for different levels of the tree.

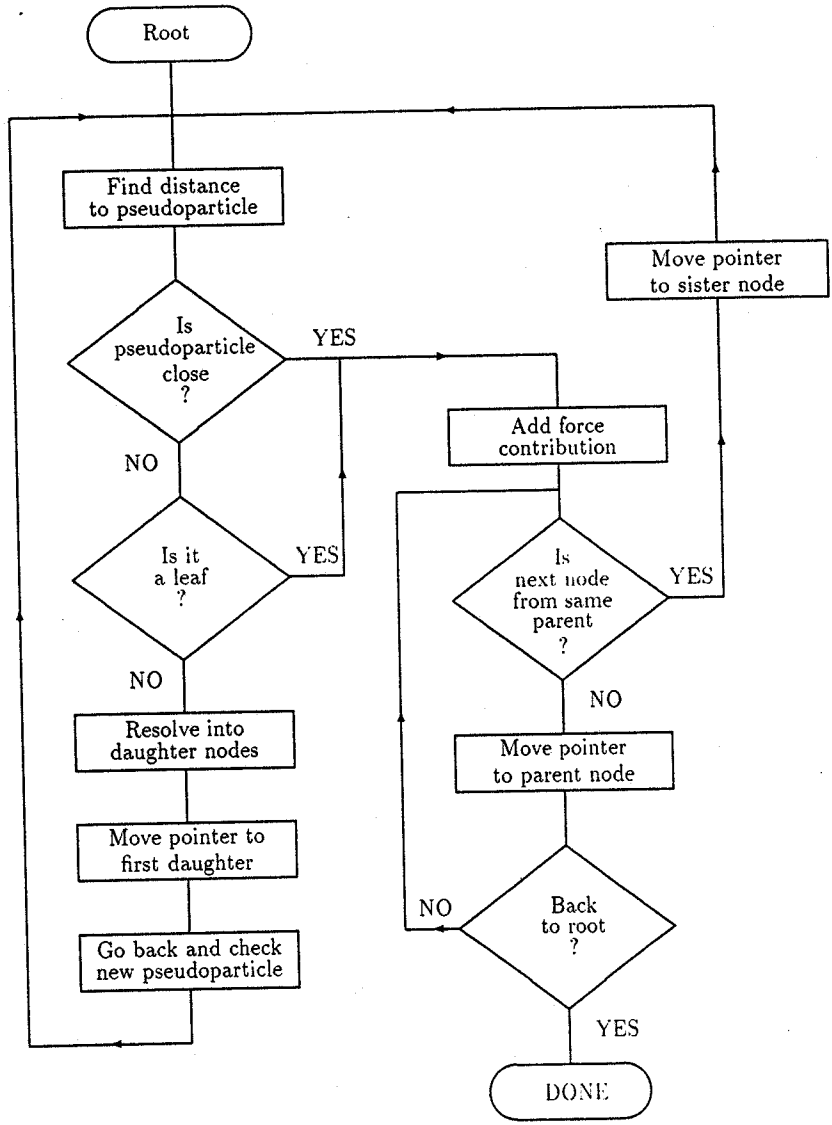


Fig. 2.7. Flowchart of the force calculation.

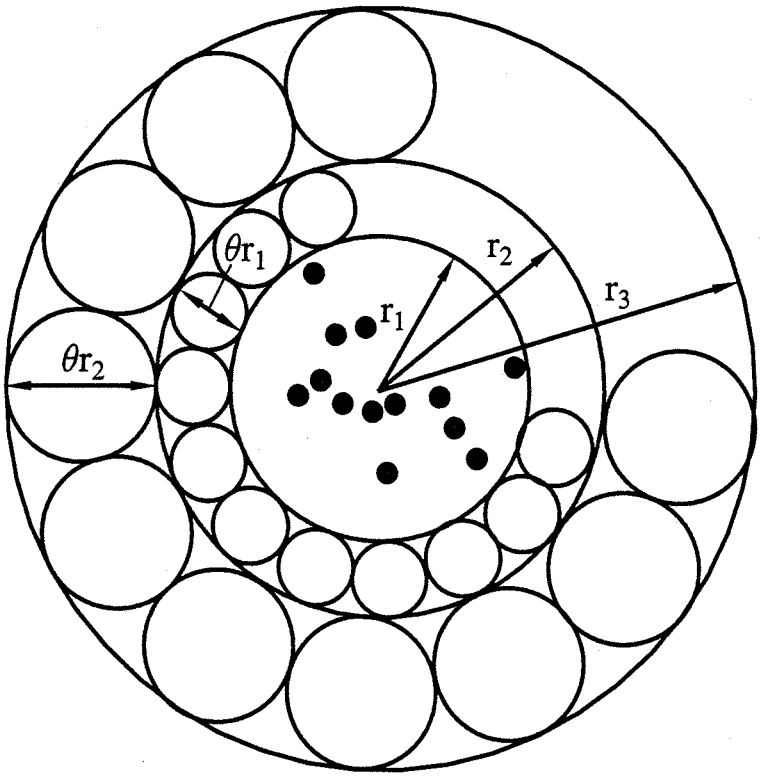


Fig. 2.8. Schematic of the hierarchical tree structure according to Hernquist.

number of interactions:

$$n_o = N\left(\frac{r_1}{R}\right)^3 = n\frac{4\pi}{3}r_1^3 = \frac{4\pi}{3}\frac{1}{\theta^3}.$$

Now the radii of the outer shells are $r_i = (1+\theta)^{i-1}r_1$, implying

$$\frac{r_n}{r_1} = (1+\theta)^{n_{sh}-1},$$

which gives the number of shells as

$$\begin{aligned} n_{sh} &= 1 + \log_{(1+\theta)}\frac{r_n}{r_1} \\ &= 1 + \log_{(1+\theta)}\frac{R}{(1+\theta)r_1} \\ &= 1 + \log_{(1+\theta)}\left(\frac{1}{1+\theta}\frac{\theta}{(3N/4\pi)^{-1/3}}\right), \end{aligned}$$

where we have used $R = r_n(1+\theta)$ to get the last expression. Therefore, the total number of interactions is given by

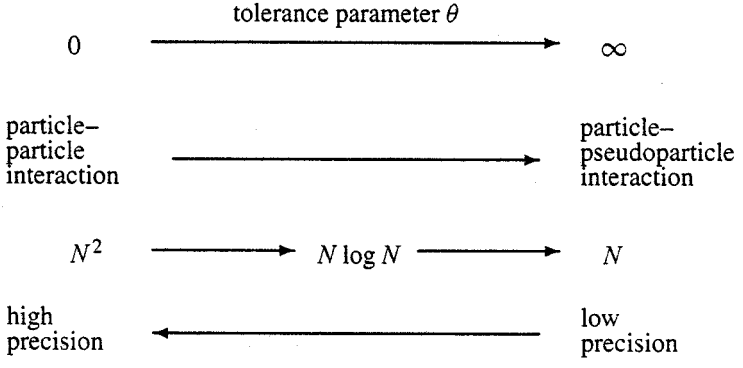
$$n_{int} \sim \frac{24}{\theta^2} \frac{\log\left(\theta(3N/4\pi)^{1/3}\right)}{\log(1+\theta)} + \frac{4\pi}{3}\frac{1}{\theta^3}. \tag{2.5}$$

For large N, it follows that $\theta \geq (4\pi/3N)^{1/3}$; this means that the dominant behaviour for $\theta > 0$ is

$$n_{int} \sim \log N/\theta^2 \tag{2.6}$$

and the time required to calculate the force on a given particle is $O(\log N)$, which means the number of operations to compute the force on all N bodies will scale as $O(N \log N)$.

In contrast to the idealised picture in Fig. 2.8, the actual interaction list resembles one of the examples shown in Fig. 2.9. As this illustrates, the computation time not only is a function of the simulation size but also depends strongly on the choice of the tolerance parameter θ. The case $\theta = 0$ is equivalent to computing all particle–particle interactions, which is exact but rather time-consuming because the operation count is again $O(N^2)$. This is in fact slower than traditional PP because one has to build the tree and walk all the way up to the leaves during the force calculation. Choosing the other extreme, $\theta \to \infty$, would produce a very low spatial resolution with the force being calculated only between the individual particle and large pseudoparticles, which would be very quick but extremely inaccurate. This conflict in the choice of θ can be summarised as follows.



One would really like a rapid force calculation *and* high precision. In fact, a compromise of $\theta \sim 0.1 - 1.0$, depending on the problem, proves to be a practical choice. Fortunately, there is an additional way of improving the accuracy without the need of a higher resolution (smaller θ), which is to include

the multipole moments of the pseudoparticles, rather than treating them as point masses (or charges).

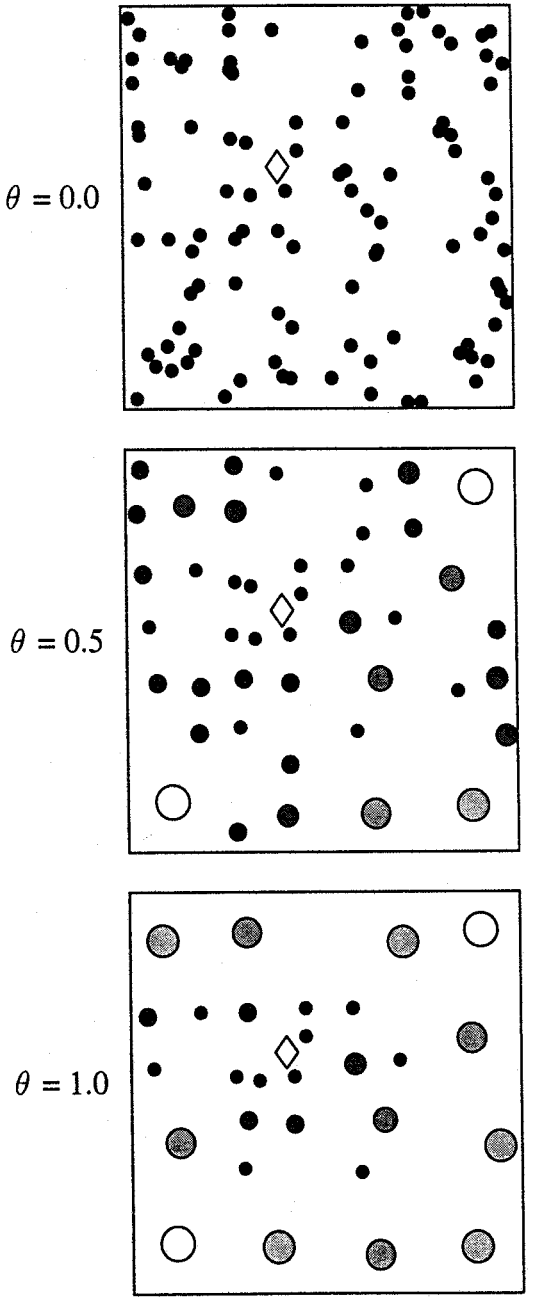


Fig. 2.9. Interaction list for a given particle (◇) for different values of the tolerance parameter θ.

masses (or charges).

LECTURE

2.3 Multipole Expansion

In the analysis that follows in this section, we consider an N-body system which is characterised simply by the masses and positions of its N particles and has a $1/r$-potential. In the tree algorithm, distant particles are grouped to form a pseudoparticle with a mass equal to the sum of its constituent masses, and a position equal to the centre of mass of the group. This means a loss of information about the spatial distribution of the masses within the cell and leads to an error in the force calculation. One can recover the spatial information by performing a multipole expansion of the particle distribution in the cell. The force of the pseudoparticle on the individual particle is then given by

$$\begin{aligned} \mathbf{F}(\mathbf{R}-\mathbf{r}_i) \simeq\ & \mathbf{F}(\mathbf{R}) && \text{Monopole} \\ & -\mathbf{r}_i \nabla \mathbf{F}(\mathbf{R}) && \text{Dipole} \\ & +\tfrac{1}{2}\mathbf{r}_i\mathbf{r}_i :: \nabla\nabla\mathbf{F}(\mathbf{R}) && \text{Quadrupole} \\ & +\ldots, \end{aligned}$$

where $\mathbf{R}$ is the vector from the particle to the centre of mass and $\mathbf{r}_i$ is the vector from the particle to an individual particle in the cell (see Fig. 2.10). If this multipole expansion is carried out to a high enough order, it will contain the total information of the particle distribution in the cell. There are more recently developed codes based on a 'Fast Multipole Method' (FMM) which exploit this fact. They use large pseudoparticles and perform the multipole expansion to a high order (typically 10–20) and represent a most elegant refinement of the tree code, which, depending on context and accuracy requirements, starts to become economic for $N \gtrsim 10^4$–10^5.

The maximum size of a pseudoparticle in a 3-dimensional code is 1/8 of the total system volume. In practice, the size of the cell occupied by a pseudoparticle is determined by the choice of θ. For potentials falling off as $1/r$ or faster, higher moments contribute increasingly less to the force. In the range of $\theta \sim 0.1 - 1.0$ it turns out that including the dipole and quadrupole moments of the pseudoparticles gives acceptable accuracy in the force calculation for typical dynamical applications. Doing the tree construction by subdividing cubic cells fixes us to a Cartesian coordinate system and the multipole expansion is done in Cartesian coordinates too. Unfortunately, this is rather clumsy in comparison to the usual expansion in Legendre polynominals found in Jackson

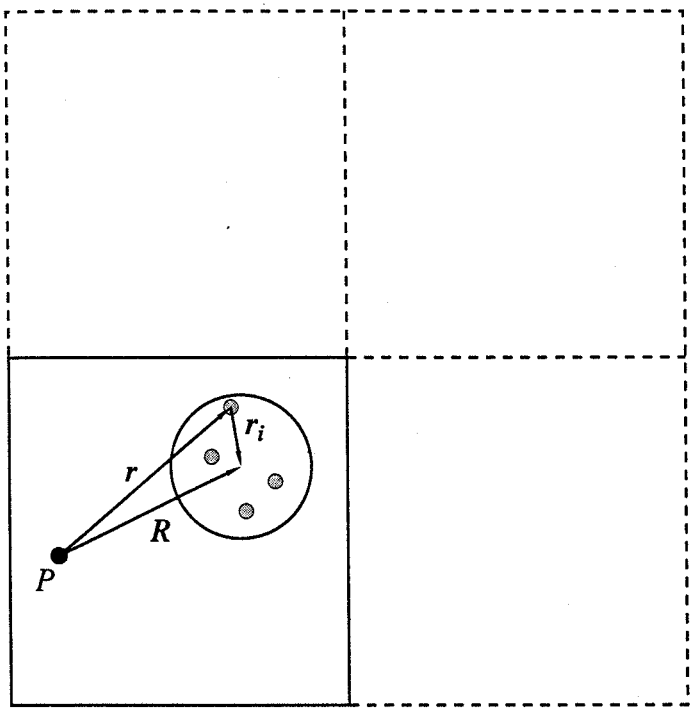


Fig. 2.10. **R** is the vector from the individual particle (P) to the centre of mass of the pseudoparticle (large circle), **r** is the vector to a single particle of this pseudoparticle, and $\mathbf{r}_i$ is the vector from a member of the pseudoparticle to the centre of mass.

(1975). Different formulations for computing higher order moments in 2- and 3-dimensional FMM codes are given in Chapter 7.

The potential at the origin Φ_o due to the pseudoparticle is the sum of the potentials Φ_i due to the particles in the cell,

$$\Phi(\mathbf{R}) = \sum_i \Phi_i(\mathbf{R} - \mathbf{r}_i),$$

where $\mathbf{r}_i$ is the vector from the particle to the centre of mass and the origin is, for simplicity, the individual particle on which the force of the pseudoparticle is calculated. Here we consider a $1/r$-potential, therefore

$$\begin{aligned}\Phi_i(\mathbf{R} - \mathbf{r}_i) &= -\frac{m_i}{|\mathbf{R} - \mathbf{r}_i|} \\ &= -\frac{m_i x_i}{\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2}}.\end{aligned}$$

The multipole expansion of the potential up to the quadrupole moment is given by

$$\begin{aligned}\Phi(\mathbf{R}) &= -\sum_i m_i \left[1 - \mathbf{r}_i \frac{\partial}{\partial \mathbf{r}} + \frac{1}{2}\mathbf{r}_i\,\mathbf{r}_i \frac{\partial}{\partial \mathbf{r}}\frac{\partial}{\partial \mathbf{r}} + \ldots\right]\frac{1}{R} \\ &= -\sum_i m_i \left[1 - x_i\frac{\partial}{\partial x} - y_i\frac{\partial}{\partial y} - z_i\frac{\partial}{\partial z}\right. \\ &\qquad + \frac{1}{2}x_i^2\frac{\partial}{\partial x}\frac{\partial}{\partial x} + \frac{1}{2}y_i^2\frac{\partial}{\partial y}\frac{\partial}{\partial y} + \frac{1}{2}z_i^2\frac{\partial}{\partial z}\frac{\partial}{\partial z} \\ &\qquad + \frac{1}{2}x_i y_i\left(\frac{\partial}{\partial x}\frac{\partial}{\partial y} + \frac{\partial}{\partial y}\frac{\partial}{\partial x}\right) \\ &\qquad + \frac{1}{2}y_i z_i\left(\frac{\partial}{\partial y}\frac{\partial}{\partial z} + \frac{\partial}{\partial z}\frac{\partial}{\partial y}\right) \\ &\qquad \left. + \frac{1}{2}x_i z_i\left(\frac{\partial}{\partial x}\frac{\partial}{\partial z} + \frac{\partial}{\partial z}\frac{\partial}{\partial x}\right)\right]\frac{1}{R}.\end{aligned}$$

There are in principle three kinds of derivatives

$$\begin{aligned}\frac{\partial}{\partial x}\frac{1}{R} &= -\frac{x}{R^3}, \\ \frac{\partial}{\partial x}\frac{\partial}{\partial x}\frac{1}{R} &= \frac{3x^2}{R^5} - \frac{1}{R^3}, \\ \frac{\partial}{\partial x}\frac{\partial}{\partial y}\frac{1}{R} &= \frac{3xy}{R^5},\end{aligned}$$

and the other coordinates can be obtained in the same way. The multipole expansion of the pseudoparticle potential is then

$$\begin{aligned}\Phi(\mathbf{R}) = -\sum_i m_i \Bigg[& \frac{1}{R} + x_i\frac{x}{R^3} + y_i\frac{y}{R^3} + z_i\frac{z}{R^3} \\ & + \frac{1}{2}x_i^2\left(-\frac{1}{R^3} + \frac{3x^2}{R^5}\right) + \frac{1}{2}y_i^2\left(-\frac{1}{R^3} + \frac{3y^2}{R^5}\right) \\ & + \frac{1}{2}z_i^2\left(-\frac{1}{R^3} + \frac{3z^2}{R^5}\right) \\ & + x_i y_i\left(\frac{3xy}{R^5}\right) + y_i z_i\left(\frac{3yz}{R^5}\right) + x_i z_i\left(\frac{3xz}{R^5}\right)\Bigg].\end{aligned} \tag{2.7}$$

The force on the individual particle by the pseudoparticle is given as the derivative of the potential

$$\mathbf{F}(\mathbf{R}) = -m_p \frac{\partial}{\partial \mathbf{r}} \sum_i \Phi_i. \tag{2.8}$$

Using Eq. 2.7 we can obtain the field by performing derivatives of the form:

Monopole (M):
$$\frac{\partial}{\partial x}\frac{1}{R} = -\frac{x}{R^3}.$$

Dipole (D):
$$\frac{\partial}{\partial x}\frac{x}{R^3} = \frac{1}{R^3} - \frac{3x^2}{R^5},$$
$$\frac{\partial}{\partial x}\frac{y}{R^3} = \frac{-3xy}{R^5}.$$

Quadrupole (Q):
$$\frac{\partial}{\partial x}\left(\frac{1}{R^3} - \frac{3x^2}{R^5}\right) = -\frac{9x}{R^5} + \frac{15x^3}{R^7},$$
$$\frac{\partial}{\partial x}\frac{-3xy}{R^5} = \frac{-3y}{R^5} + \frac{15x^2y}{R^7},$$
$$\frac{\partial}{\partial x}\frac{-3yz}{R^5} = \frac{15xyz}{R^7},$$
$$\frac{\partial}{\partial x}\left(\frac{1}{R^3} - \frac{3y^2}{R^5}\right) = \frac{-3x}{R^5} + \frac{15xz^2}{R^7}.$$

The x-component of the force vector is then given by:

$-F_x =$

M:
$$\frac{x}{R^3}\sum_i m_i.$$

D:
$$-\left(\frac{1}{R^3} - \frac{3x^2}{R^5}\right)\cdot\sum_i m_i x_i + \frac{3xy}{R^5}\cdot\sum_i m_i y_i + \frac{3xz}{R^5}\cdot\sum_i m_i z_i.$$

Q:
$$+\left(\frac{15x^3}{R^7} - \frac{9x}{R^5}\right)\cdot\frac{1}{2}\sum_i m_i x_i^2 + \left(\frac{15xy^2}{R^7} - \frac{3x}{R^5}\right)\cdot\frac{1}{2}\sum_i m_i y_i^2$$
$$+\left(\frac{15xz^2}{R^7} - \frac{3x}{R^5}\right)\cdot\frac{1}{2}\sum_i m_i z_i^2 + \left(\frac{15x^2y}{R^7} - \frac{3y}{R^5}\right)\cdot\sum_i m_i x_i y_i$$
$$+\left(\frac{15x^2z}{R^7} - \frac{3z}{R^5}\right)\cdot\sum_i m_i x_i z_i + \left(\frac{15xyz}{R^7}\right)\cdot\sum_i m_i y_i z_i.$$

Equivalent expressions for the y- and z-components of the force can be easily obtained by cyclic rotation. Appendix 1 shows the multipole expansion for the somewhat simpler 2-dimensional case.

To keep the computational effort to a minimum it would be useful to utilize the information stored for the daughter nodes to obtain information about the parent nodes. The force expression contains sums such as: $M = \sum_i m_i$, the dipole moment $\mathbf{D} = \sum_i m_i \mathbf{r}_i$, and the quadrupole moments $Q_{xx} = \sum_i m_i x_i^2$, $Q_{xy} = \sum_i m_i x_i y_i$, and so forth. In general, the multipole moments depend on the choice of the origin of the coordinate system, but the value of the first non-vanishing moment is independent of this choice (Jackson 1975). This means that the monopole moment is independent of this choice, and therefore the total mass of the parent cell can simply be obtained by summing over the masses of the daughter cells.

The calculation of the dipole and quadrupole moments of the parent cell from the moments of the daughter cell is a bit more difficult because the moments depend on the choice of the origin. Initially, the moments are calculated relative to the centre of mass, defined by

$$\mathbf{r}_{cm} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i}.$$

As Fig. 2.11 illustrates, the centres of mass are different for the daughter cells and the parent cell. The difference in the x-component is given by

$$x_{sd} = x_{cm}(daughter) - x_{cm}(parent),$$

where d indicates the different daughters. For example, for daughter 1 in Fig. 2.11, we have:

$$x_{s1} = X_1 - X_0.$$

This means that the moments of the daughter cell have to be calculated relative to a new origin, which is equivalent to a shift by x_{sd}. Each individual x_i in the sum of one daughter is shifted by the same vector x_{sd}, meaning: $x_i(\text{new}) = x_i - x_{sd}$. The dipole moment relative to the new origin can be obtained from the dipole moment relative to the original origin by making this substitution:

$$\begin{aligned} D_x^{daughter}(\text{new}) &= \sum_i m_i (x_i - x_{sd}) \\ &= \sum_i m_i x_i - \sum_i m_i x_{sd} \\ &= \sum_i m_i x_i - x_{sd} \sum_i m_i. \end{aligned}$$

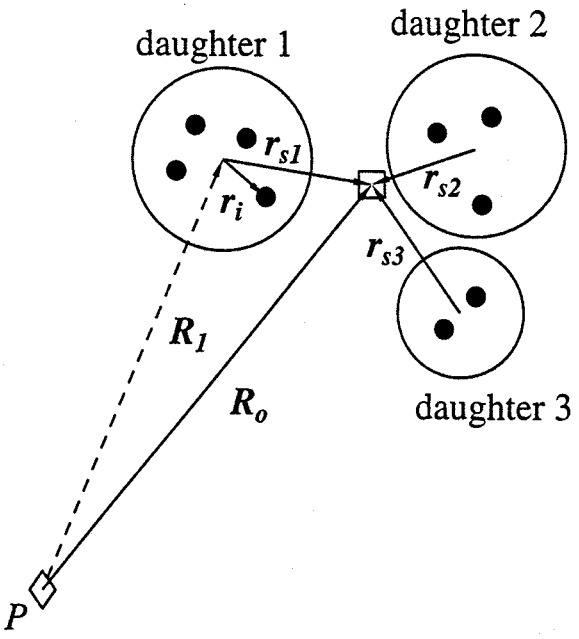


Fig. 2.11. Origin shift for the multipole calculation: the circles symbolize the pseudoparticles; $\mathbf{r}_i$ is the vector from a member of one of the pseudoparticles to the centre of mass of this pseudoparticle and $\mathbf{r}_{si}$ is the shifting vector to the new origin (□), which is the centre of mass of the daughters (pseudoparticles).

The same procedure applies for the quadrupole moments, in which case we have:

$$Q_{xx}^{daughter}(\text{new}) = \sum_i m_i (x_i - x_{sd})^2$$
$$= \sum_i m_i x_i^2 - 2x_{sd} \sum_i m_i x_i + x_{sd}^2 \sum_i m_i ,$$

and

$$Q_{xy}^{daughter}(\text{new}) = \sum_i m_i (x_i - x_{sd})(y_i - y_{sd})$$
$$= \sum_i m_i x_i y_i - x_{sd} \sum_i m_i y_i$$
$$- y_{sd} \sum_i m_i x_i + x_{sd} y_{sd} \sum_i m_i .$$

This means that all moments of the parent can be obtained from the knowledge of the daughters' moments and the shifting vectors $\mathbf{r}_{sd}$. The dipole moment of the parent cell is therefore:

$$D_x^{parent} = \sum_d \left(\sum_i m_i x_i - x_{sd} \sum_i m_i \right)$$
$$= \sum_d \left(D_x^d - x_{sd} M^d \right). \tag{2.9}$$

Similarly, the quadrupole moments of the parent cell can be found from:

$$Q_{xx}^{parent} = \sum_d \left(\sum_i m_i x_i^2 - 2x_{sd} \sum_i m_i x_i + x_{sd}^2 \sum_i m_i \right)$$
$$= \sum_d \left(Q_{xx}^d - 2x_{sd} D_x^d + x_{sd}^2 M^d \right), \tag{2.10}$$

$$Q_{xy}^{parent} = \sum_d \left(\sum_i m_i x_i y_i - x_{sd} \sum_i m_i y_i \right.$$
$$\left. + y_{sd} \sum_i m_i x_i + x_{sd} y_{sd} \sum_i m_i \right)$$
$$= \sum_d \left(Q_{xy}^d - x_{sd} D_y^d - y_{sd} D_x^d + x_{sd} y_{sd} M^d \right). \tag{2.11}$$

How is this implemented in the code? Starting at the highest level of the tree, the sums $\sum_i m_i$, $\sum_i m_i x_i$, $\sum_i m_i x_i^2$, and $\sum_i m_i x_i y_i$ and the equivalent ones for the other space directions are calculated and stored for every pseudoparticle (twig node). The multipole moments for the next level down (i.e., for the parent nodes) are then evaluated *at their respective centres of mass* using the shifting formulae (2.9)–(2.11). In practice, this means adding another loop over the daughter nodes to the MOMENTS routine described earlier.

```
do i = 1, nbuds
    point_dau = point1 + i - 1
    inode = node(point_dau)

    shift vector
    r_s = r_com(itwig) - r_com(inode)
```

sum moments
$\mathbf{D}$(itwig) = $\mathbf{D}$(itwig)
+ $\mathbf{D}$(inode) − $\mathbf{r}_s \times Q$(inode)
... etc.
end do

This procedure is continued until the root node is reached, which will then contain a (second-order) expansion for the whole system. The moments are later used to evaluate the dipole and quadrupole corrections in the force calculation.

For problems in which the forces are gravitational, the dipole moments vanish relative to the centre of mass, so that we have only monopole and quadrupole contributions to the force calculation. We will see later that this does not happen for electrostatic forces. Table 2.2 shows how including the moment terms in the force calculation improves the accuracy. Or, seen from a different point of view, the same level of accuracy in the force computation is achieved more efficiently if quadrupole moments are included rather than using a smaller θ in the monopole version. This is due to the fact that, as (2.5) shows, the number of interactions varies as θ^{-3}, and so increases rapidly with decreasing θ.

The error introduced into the force calculation as a result of the truncated multipole expansion increases monotonically with θ. For open systems the error relative to a direct particle–particle calculation is typically $\lesssim 0.1\%$ for $\theta = 1$. It increases rapidly for larger θ, which suggests that $\theta = 1$ may be a practical upper limit for open systems.

A factor of $\sim 1/\theta$ improvement of the relative force error $\Delta F/F$ is expected for each new added term in the expansion (McMillan & Aarseth 1993). However, it would be wrong to conclude that adding higher order multipole moments is always more effective than choosing a smaller θ. The evaluation time of the higher moments (2^l) of the multipole expansion has an $O(l^2)$ dependence. So, at some point, improving the accuracy by reducing θ costs less than including higher moments (Makino 1990a).

Table 2.2. *Force on a randomly chosen particle from an ion beam simulation*

		Monopole tree		Dipole tree	
N	PP code	$\Theta = 1.0$	$\Theta = 0.3$	$\Theta = 1.0$	$\Theta = 0.3$
100	0.5284	0.5141	0.5273	0.5271	0.5287
500	0.6911	0.7152	0.6932	0.6959	0.6912
1,500	0.8951	0.9139	0.8958	0.8965	0.8951

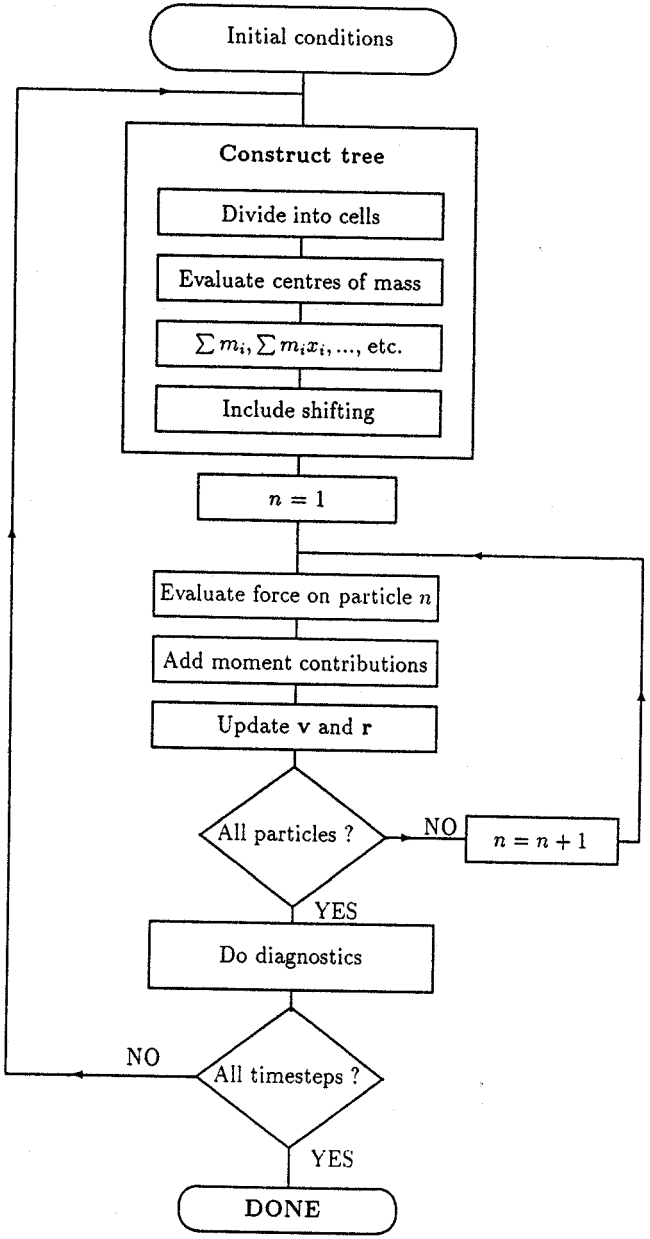


Fig. 2.12. Flowchart of a hierarchical tree code for a dynamic N-body problem.

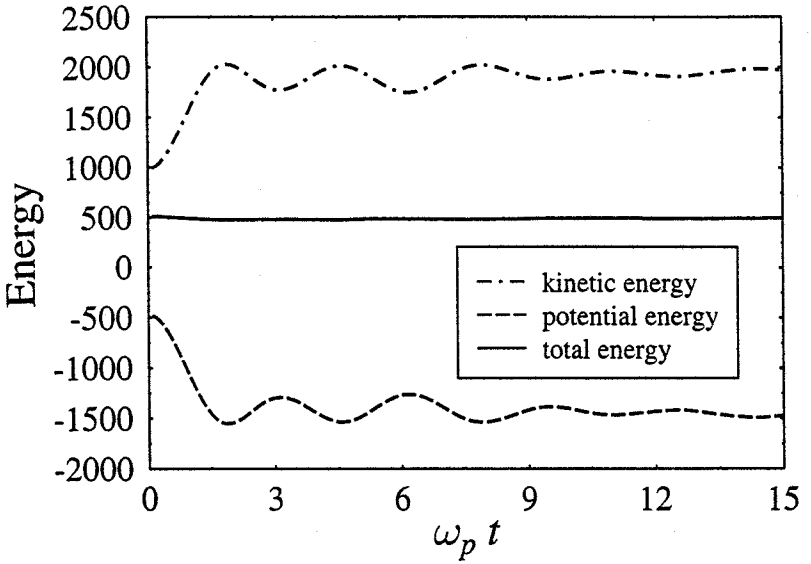


Fig. 2.13. Energy conservation for a plasma system approaching equilibrium over 10,000 timesteps.

2.4 Dynamics

Up to now we have considered only a single timestep of the dynamical system. In contrast to first hierarchical codes by Appel (1985), Jernighan (1985), and Porter (1985), recent codes mainly reconstruct the tree ab initio every timestep. This sounds like a lot of computational effort, but in fact only a small fraction of time is needed in generating the tree structure, which is usually of the order of a few percent for $\theta \lesssim 1$ of the force calculation. The whole code is summarised by the flowchart of Fig. 2.12.

The approximation of the potential and the force by the tree structure influences the dynamical evolution of representative systems only slightly. Tests for open systems show that the energy conservation is only weakly violated, and the departure from exact conservation seems to be comparable to those errors typically tolerated from time integration errors. These errors are only weakly correlated from one timestep to the next, which leads to a fluctuation around a mean value rather than a steady growth or decay. Figure 2.13 shows an example of the energy conservation for a dense plasma system over 10,000 timesteps. The fluctuation in total energy is less than 0.5%.

Because most of the integrations are treated as particle–pseudoparticle interactions, the force calculation is not symmetric. This means that momentum

and angular momentum are not exactly conserved. Empirical tests by Hernquist (1987) indicate that the departures are small for $\theta \lesssim 1$. In particular, the non-reciprocity in the force-law causes the centre of mass to drift, an effect which can be minimised by taking $\theta \leq 0.5$ and by increasing the number of particles.

Even in its most basic form, the hierarchical tree code is a very promising simulation tool and is already commonly used for the treatment of N-body problems. It can be implemented in a variety of programming languages like C, PASCAL, LISP, and FORTRAN. The advantage of C, PASCAL, and LISP is that they allow recursive function calls and permit a close correspondence between the tree structure and the program coding.

In a later chapter we will show how the computational performance of tree codes can be improved. On the software side, this means higher order integration schemes, vectorisation and introduction of individual timesteps. On the hardware side, there have been suggestions to design computers which are especially adapted to the tree structure. The tree algorithm can also be used on a parallel machine: Each particle has its own interaction list which can be summed independently.