Lecture 9

Particle Methods

9.1 Reduce and Broadcast: A function viewpoint

[This section is being rewritten with what we hope will be the world's clearest explanation of the fast multipole algorithm. Readers are welcome to take a quick look at this section, or pass to the next section which leads up to the multipole algorithm through the particle method viewpoint]

Imagine we have P processors, and P functions $f_1(z), f_2(z), \ldots, f_P(z)$, one per processor. Our goal is for every processor to know the sum of the functions $f(z) = f_1(z) + \ldots + f_P(z)$. Really this is no different from the reduce and broadcast situation given in the introduction.

As a practical question, how can functions be represented on the computer? Probably we should think of Taylor series or multipole expansion. If all the Taylor series or multipole expansions are centered at the same point, then the function reduction is easy. Simply reduce the corresponding coefficients. If the pairwise sum consists of functions represented using different centers, then a common center must be found and the functions must be transformed to that center before a common sum may be found.

> **Example: Reducing Polynomials** Imagine that processor i contains the polynomial $f_i(z) = (z-i)^3$. The coefficients may be expanded out as $f_i(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3$. Each processor i contains a vector (a_0, a_1, a_2, a_3). The sum of the vectors may be obtained by a usual reduce algorithm on vectors.
>
> An alternative that may seem like too much trouble at first is that every time we make a pairwise sum we shift to a common midpoint (see Figure 9.1).

There is another complication that occurs when we form pairwise sums of functions. If the expansions are multipole or Taylor expansions, we may shift to a new center that is outside the region of convergence. The coefficients may then be meaningless. Numerically, even if we shift towards the boundary of a region of convergence, we may well lose accuracy, especially since most computations choose to fix the number of terms in the expansion to keep.

Difficulties with shifting multipole or Taylor Expansions

The fast multipole algorithm accounts for these difficulties in a fairly simple manner. Instead of computing the sum of the functions all the way up the tree and then broadcasting back, it saves the intermediate partial sums summing them in only when appropriate. The figure below indicates when this is appropriate.

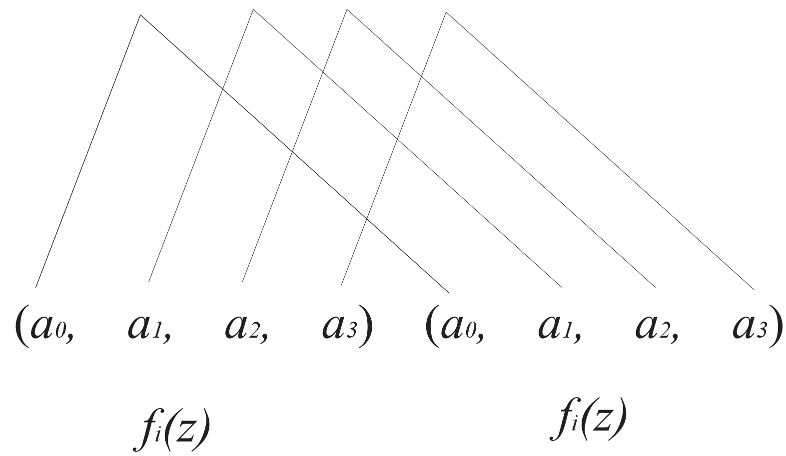

Figure 9.1: Pairwise Sum

9.2 Particle Methods: An Application

Imagine we want to model the basic mechanics of our solar system. We would probably start with the sun, somehow representing its mass, velocity, and position. We might then add each of the nine planets in turn, recording their own masses, velocities, and positions at a point in time. Let's say we add in a couple of hundred of the larger asteroids, and a few of our favorite comets. Now we set the system in motion. Perhaps we would like to know where Pluto will be in a hundred years, or whether a comet will hit us soon. To solve Newton's equations directly with more than even two bodies is intractably difficult. Instead we decide to model the system using discrete time intervals, and computing at each time interval the force that each body exerts on each other, and changing the velocities of the bodies accordingly. This is an example of an N-body problem. To solve the problem in a simple way requires $O(n^2)$ time for each time step. With some considerable effort, we can reduce this to $O(n)$ (using the fast multipole algorithm to be described below). A relatively simple algorithm the Barnes-Hut Algorithm, (to be described below) can compute movement in $O(n\log(n))$ time.

9.3 Outline

- Formulation and applications
- "The easiest part": the Euler method to move bodies.
- Direct methods for force computation.
- Hierarchical methods *(Barnes-Hut, Appel, Greengard and Rohklin)*

9.4 What is N-Body Simulation?

We take n bodies (or particles) with state describing the initial position $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \Re^k$ and initial velocities $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \in \Re^k$.

We want to simulate the evolution of such a system, i.e., to compute the trajectories of each body, under an interactive force: the force exerted on each body by the whole system at a given

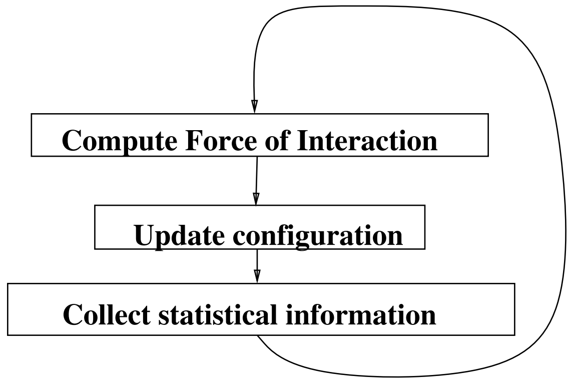

Figure 9.2: Basic Algorithm of N-body Simulation

point. For different applications we will have different interaction forces, such as gravitational or Coulombic forces. We could even use these methods to model spring systems, although the advanced methods, which assume forces decreasing with distance, do not work under these conditions.

9.5 Examples

- **Astrophysics**: The bodies are stars or galaxies, depending on the scale of the simulation. The interactive force is gravity.
- **Plasma Physics**: The basic particles are ions, electrons, etc; the force is Coulombic.
- **Molecular Dynamics**: Particles are atoms or clusters of atoms; the force is electrostatic.
- **Fluid Dynamics**: Vortex method where particle are fluid elements (fluid blobs).

Typically, we call this class of simulation methods, the **particle methods**. In such simulations, it is important that we choose both spatial and temporal scales carefully, in order to minimize running time and maximize accuracy. If we choose a time scale too large, we can lose accuracy in the simulation, and if we choose one too small, the simulations will take too long to run. A simulation of the planets of the solar system will need a much larger timescale than a model of charged ions. Similarly, spatial scale should be chosen to minimize running time and maximize accuracy. For example, in applications in fluid dynamics, molecular level simulations are simply too slow to get useful results in a reasonable period of time. Therefore, researchers use the vortex method where bodies represent large aggregates of smaller particles. Hockney and Eastwood's book **Computer Simulations Using Particles,**, McGraw Hill (1981), explores the applications of particle methods applications, although it is somewhat out of date.

9.6 The Basic Algorithm

Figure 9.2 illustrates the key steps in n-body simulation. The step of collecting statistical information is application dependent, and some of the information gathered at this step may be used during the next time interval.

We will use gravitational forces as an example to present N-body simulation algorithms. Assume there are n bodies with masses $m_1, m_2, \ldots, m_n$, respectively, initially located at $\vec{x_1}, ..., \vec{x_n} \in \Re^3$ with velocity $\vec{v_1}, ..., \vec{v_n}$. The gravitational force exert on the ith body by the jth body is given by

$$\vec{F_{ij}} = G\frac{m_i m_j}{r^2} = G\frac{m_i m_j}{|\vec{x_j} - \vec{x_i}|^3}(\vec{x_j} - \vec{x_i}),$$

where G is the gravitational constant. Thus the total force on the ith body is the vector sum of all these forces and is give by,

$$\vec{F_i} = \sum_{j \neq i} \vec{F_{ij}}.$$

Let $\vec{a_i} = d\vec{v_i}/dt$ be the acceleration of the body i, where where $\vec{v_i} = d\vec{x_i}/dt$. By Newton's second law of motion, we have $\vec{F_i} = m_i\vec{a_i} = m_i d\vec{v_i}/dt$.

In practice, we often find that using a potential function $V = \phi m$ will reduce the labor of the calculation. First, we need to compute the potential due to the N-body system position, i.e. $x_1, \ldots, x_n$, at positions $y_1, \ldots, y_n$.

The total potential is calculated as

$$V_i = \sum_{i,j=1; i \neq j}^{n} \phi(x_i - y_j) m_j \qquad 1 \leq i, j \leq n,$$

where ϕ is the potential due to gravity. This can also be written in the matrix form:

$$V = \begin{pmatrix} 0 & \ldots & (x_i - y_j) \\ \ldots & \ldots & \ldots \\ \phi(x_j - y_i) & \ldots & 0 \end{pmatrix}$$

In $\Re^3$,

$$\phi(x) = \frac{1}{\| x \|}$$

.

In $\Re^2$,

$$\phi(x) = log\| x \|$$

.

The update of particle velocities and positions are in three steps:

1. $F = \pi \cdot m$;
2. $V_{new} = V_{old} + \Delta t \cdot \frac{F}{m}$;
3. $x_{new} = x_{old} + \Delta t \cdot V_{new}$.

The first step is the most expensive part in terms of computational time.

9.6.1 Finite Difference and the Euler Method

In general, the force calculation is the most expensive step for N-body simulations. We will present several algorithms for this later on, but first assume we have already calculated the force $\vec{F_i}$ acting one each body. We can use a numerical method (such as the Euler method) to update the configuration.

To simulate the evolution of an N-body system, we decompose the time interval into discretized time steps: $t_0, t_1, t_2, t_3, ...$ For uniform discretizations, we choose a Δt and let $t_0 = 0$ and $t_k = k\Delta t$. The Euler method approximates the derivative by finite difference.

$$\begin{aligned} \vec{a_i}(t_k) &= \vec{F_i}/m_i = \frac{\vec{v_i}(t_k) - \vec{v_i}(t_k - \Delta t)}{\Delta t} \\ \vec{v_i}(t_k) &= \frac{\vec{x_i}(t_k + \Delta t) - \vec{x_i}(t_k)}{\Delta t}, \end{aligned}$$

where $1 \leq i \leq n$. Therefore,

$$\vec{v_i}(t_k) = \vec{v_i}(t_{k-1}) + \Delta t(\vec{F_i}/m_i) \tag{9.1}$$
$$\vec{x_i}(t_{k+1}) = \vec{x_i}(t_k) + \Delta t \vec{v_i}(t_k). \tag{9.2}$$

From the given initial configuration, we can derive the next time step configuration using the formulae by first finding the force, from which we can derive velocity, and then position, and then force at the next time step.

$$\vec{F_i} \to v_i(t_k) \to x_i(t_k + \Delta t) \to \vec{F_{i+1}}.$$

High order numerical methods can be used here to improve the simulation. In fact, the Euler method that uses uniform time step discretization performs poorly during the simulation when two bodies are very close. We may need to use non-uniform discretization or a sophisticated time scale that may vary for different regions of the N-body system.

In one region of our simulation, for instance, there might be an area where there are few bodies, and each is moving slowly. The positions and velocities of these bodies, then, do not need to be sampled as frequently as in other, higher activity areas, and can be determined by extrapolation. See figure 9.3 for illustration.[1]

How many floating point operations (flops) does each step of the Euler method take? The velocity update (step 1) takes $2n$ floating point multiplications and one addition and the position updating (step 2) takes 1 multiplication and one addition. Thus, each Euler step takes $5n$ floating point operations. In Big-O notation, this is an $O(n)$ time calculation with a constant factor 5.

Notice also, each Euler step can be parallelized without communication overhead. In data parallel style, we can express steps (1) and (2), respectively, as

$$\begin{aligned} V &= V + \Delta t(F/M) \\ X &= X + \Delta t V, \end{aligned}$$

where V is the velocity array; X is the position array; F is the force array; and M is the mass array. V, X, F, M are $3 \times n$ arrays with each column corresponding to a particle. The operator / is the elementwise division.

[1]In figure 9.3 we see an example where we have some close clusters of bodies, and several relatively disconnected bodies. For the purposes of the simulation, we can ignore the movement of relatively isolated bodies for short periods of time and calculate more frames of the proximous bodies. This saves computation time and grants the simulation more accuracy where it is most needed. In many ways these sampling techniques are a temporal analogue of the later discussed Barnes and Hut and Multipole methods.

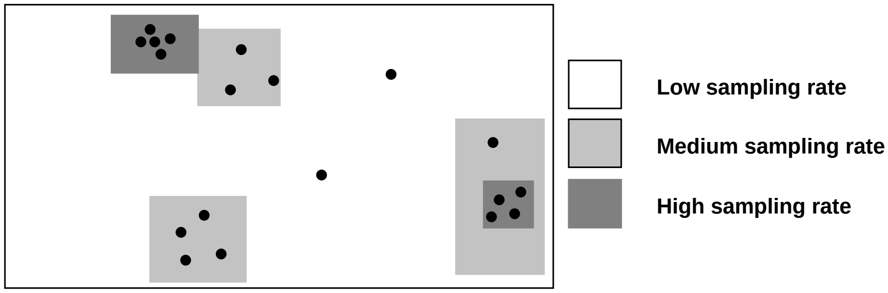

Figure 9.3: Adaptive Sampling Based on Proximity

9.7 Methods for Force Calculation

Computationally, the force calculation is the most time expensive step for N-body simulation. We now discuss some methods for computing forces.

9.7.1 Direct force calculation

The simplest way is to calculate the force directly from the definition.

$$\vec{F_{ij}} = G\frac{m_i m_j}{r^2} = G\frac{m_i m_j}{|\vec{x_j} - \vec{x_i}|^3}(\vec{x_j} - \vec{x_i}),$$

Note that the step for computing $\vec{F_{ij}}$ takes 9 flops. It takes n flops to add $\vec{F_{ij}}$ $(1 \leq j \leq n)$. Since $\vec{F_{ij}} = -\vec{F_{ji}}$, the total number of flops needed is roughly $5n^2$. In Big-O notation, this is an $O(n^2)$ time computation. For large scale simulation (e.g., $n = 100$ million), the direct method is impractical with today's level of computing power.

It is clear, then, that we need more efficient algorithms. The one fact that we have to take advantage of is that in a large system, the effects of individual distant particles on each other may be insignificant and we may be able to disregard them without significant loss of accuracy. Instead we will cluster these particles, and deal with them as though they were one mass. Thus, in order to gain efficiency, we will approximate in space as we did in time by discretizing.

9.7.2 Potential based calculation

For N-body simulations, sometimes it is easier to work with the (gravitational) potential rather than with the force directly. The force can then be calculated as the gradient of the potential.

In three dimensions, the gravitational potential at position $\vec{x}$ defined by n bodies with masses $m_1, ..., m_n$ at position $\vec{x_1}, ..., \vec{x_n}$, respectively is equal to

$$\Phi(\vec{x}) = \sum_{i=1}^{n} G\frac{m_i}{||\vec{x} - \vec{x_i}||}.$$

The force acting on a body with unit mass at position $\vec{x}$ is given by the *gradient* of Φ, i.e.,

$$F = -\nabla\Phi(x).$$

The potential function is a sum of local potential functions

$$\phi(\vec{x}) = \sum_{i=1}^{n} \phi_{\vec{x}_i}(\vec{x}) \tag{9.3}$$

where the local potential functions are given by

$$\phi_{\vec{x}_i}(\vec{x}) = \frac{G * m_i}{||\vec{x} - \vec{x}_i||} \qquad \text{in } \Re^3 \tag{9.4}$$

9.7.3 Poisson Methods

The earlier method from 70s is to use Poisson solver. We work with the gravitational potential field rather than the force field. The observation is that the potential field can be expressed as the solution of a Poisson equation and the force field is the gradient of the potential field.

The gravitational potential at position $\vec{x}$ defined by n bodies with masses $m_1, ..., m_n$ at position $\vec{x_1}, ..., \vec{x_n}$, respectively is equal to

$$\Phi(\vec{x}) = \sum_{i=1}^{n} G \frac{m_i}{|\vec{x} - \vec{x_i}|}.$$

The force acting on a body with unit mass at position $\vec{x}$ is given by the gradient of Φ:

$$\vec{F} = -\nabla\Phi(\vec{x}).$$

So, from Φ we can calculate the force field (by numerical approximation).

The potential field Φ satisfies a Poisson equation:

$$\nabla^2\Phi = \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial z^2} = \rho(x, y, z),$$

where ρ measures the mass distribution can be determined by the configuration of the N-body system. (The function Φ is harmonic away from the bodies and near the bodies, $div\nabla\Phi = \nabla^2\Phi$ is determined by the mass distribution function. So $\rho = 0$ away from bodies).

We can use finite difference methods to solve this type of partial differential equations. In three dimensions, we discretize the domain by a structured grid.

We approximate the Laplace operator ∇^2 by finite difference and obtain from $\nabla^2\Phi = \rho(x, y, z)$ a system of linear equations. Let h denote the grid spacing. We have

$$\begin{aligned}
\Phi(x_i, y_j, z_k) &= \frac{1}{h^2}(\Phi(x_i + h, y_j, z_k) + \Phi(x_i - h, y_j, z_k) + \Phi(x_i, y_j + h, z_k) \\
&\qquad + \Phi(x_i, y_j - h, z_k) + \Phi(x_i, y_j, z_k + h) + \Phi(x_i, y_j, z_k - h) - 6\phi(x_i, y_j, z_k)) \\
&= \rho(x_i, y_j, z_k).
\end{aligned}$$

The resulting linear system is of size equal to the number of grid points chosen. This can be solved using methods such as FFT (fast Fourier transform), SOR (successive overrelaxation), multigrid methods or conjugate gradient. If n bodies give a relatively uniform distribution, then we can use a grid which has about n grid points. The solution can be fairly efficient, especially on parallel machines. For highly non-uniform set of bodies, hybrid methods such as finding the potential induced by bodies within near distance by direct method, and approximate the potential field induced by distant bodies by the solution of a much smaller Poisson equation discretization. More details of these methods can be found in Hockney and Eastwood's book.

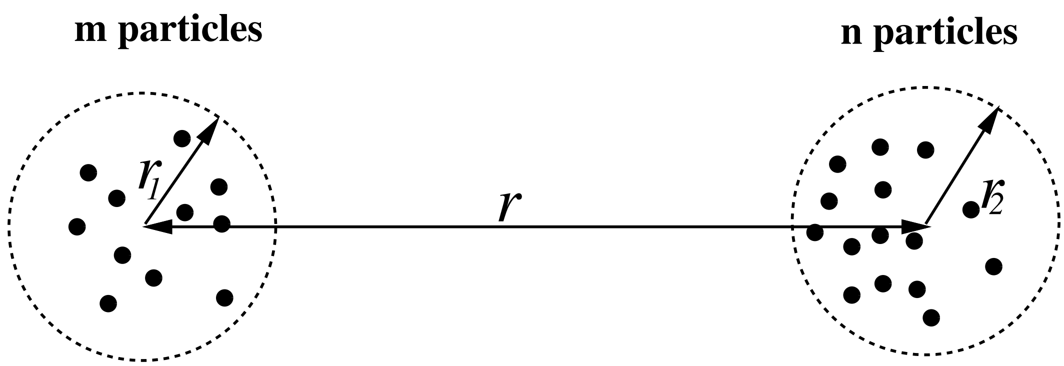

Figure 9.4: Well-separated Clusters

9.7.4 Hierarchical methods

We now discuss several methods which use a hierarchical structure to decompose bodies into clusters. Then the force field is approximated by computing the interaction between bodies and clusters and/or between clusters and clusters. We will refer this class of methods *hierarchical methods or tree-code methods.*

The crux of hierarchical N-body methods is to decompose the potential at a point x, $\phi(x)$, into the sum of two potentials: $\phi_N(x)$, the potential induced by "neighboring" or "near-field" particles; and $\phi_F(x)$, the potential due to "far-field" particles [5, 44]. In hierarchical methods, $\phi_N(x)$ is computed exactly, while $\phi_F(x)$ is computed approximately.

The approximation is based on a notion of well-separated clusters [5, 44]. Suppose we have two clusters A and B, one of m particles and one of n particles, the centers of which are separated by a distance r. See Figure 9.4.

Suppose we want to find the force acting on all bodies in A by those in B and vice versa. A direct force calculation requires $O(mn)$ operations, because for each body in A we need to compute the force induced by every body in B.

Notice that if r is much larger than both r_1 and r_2, then we can simplify the calculation tremendously by replacing B by a larger body at its center of mass and replacing A by a larger body at its center of mass. Let M_A and M_B be the total mass of A and B, respectively. The center of mass c_A and c_B is given by

$$\begin{aligned} c_A &= \frac{\sum_{i\in A} m_i x_i}{M_A} \\ c_B &= \frac{\sum_{j\in B} m_j x_j}{M_B}. \end{aligned}$$

We can approximate the force induced by bodies in B on a body of mass m_x located at position s by viewing B as a single mass M_B at location c_B. That is,

$$F(x) \approx \frac{G m_x M_B (x - c_B)}{||x - c_B||^3}.$$

Such approximation is second order: The relative error introduced by using center of mass is bounded by $(\max(r_1, r_2)/r)^2$. In other words, if $f(x)$ be the true force vector acting on a body at

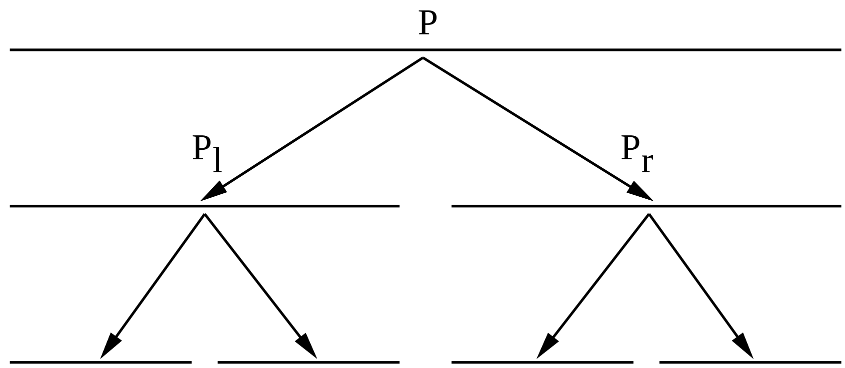

Figure 9.5: Binary Tree (subdivision of a straight line segment)

x, then

$$F(x) = f(x)\left(1 + O\left(\left(\frac{\max(r_1, r_2)}{r}\right)^2\right)\right).$$

This way, we can find all the interaction forces between A and B in $O(n+m)$ time. The force calculations between one m particle will computed separately using a recursive construction. This observation gives birth the idea of hierarchical methods.

We can also describe the method in terms of potentials. If r is much larger than both r_1 and r_2, i.e., A and B are "well-separated", then we can use the pth order multipole expansion (to be given later) to express the pth order approximation of potential due to all particles in B. Let $\Phi_B^p(x)$ denote such a multipole expansion. To (approximately) compute the potential at particles in A, we simply evaluate $\Phi_B^p()$ at each particle in A. Suppose $\Phi_B^p()$ has $g(p,d)$ terms. Using multipole expansion, we reduce the number of operations to $g(p,d)(|A|+|B|)$. The error of the multipole-expansion depends on p and the ratio $\max(r_1, r_2)/r$. We say A and B are *β-well-separated*, for a $\beta > 2$, if $\max(r_1, r_2)/r \le 1/\beta$. As shown in [44], the error of the pth order multipole expansion is bounded by $(1/(\beta-1))^p$.

9.8 Quadtree (2D) and Octtree (3D) : Data Structures for Canonical Clustering

Hierarchical N-body methods use quadtree (for 2D) and octtree (for 3D) to generate a canonical set of boxes to define clusters. The number of boxes is typically linear in the number of particles, i.e., $O(n)$.

Quadtrees and octtrees provide a way of hierarchically decomposing two dimensional and three dimensional space. Consider first the one dimensional example of a straight line segment. One way to introduce clusters is to recursively divide the line as shown in Figure 9.5.

This results in a binary tree[2].

In two dimensions, a *box* decomposition is used to partition the space (Figure 9.6). Note that a box may be regarded as a "product" of two intervals. Each partition has at most one particle in it.

[2]A tree is a graph with a single *root* node and a number of subordinate nodes called *leaves* or *children*. In a binary tree, every node has at most two children.

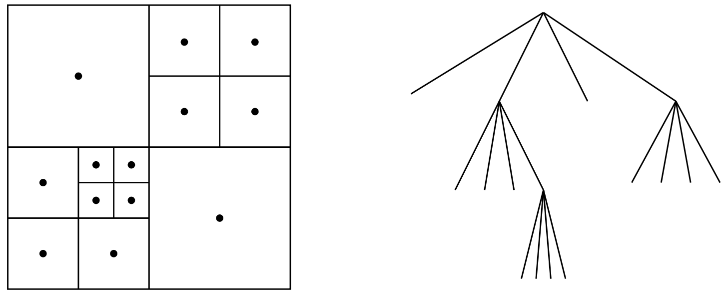

Figure 9.6: Quadtree

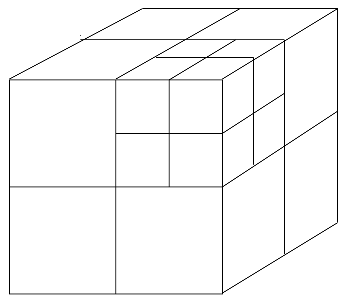

Figure 9.7: Octtree

A *quadtree* [88] is a recursive partition of a region of the plane into axis-aligned squares. One square, the *root*, covers the entire set of particles. It is often chosen to be the smallest (up to a constant factor) square that contains all particles. A square can be divided into four *child* squares, by splitting it with horizontal and vertical line segments through its center. The collection of squares then forms a tree, with smaller squares at lower levels of the tree. The recursive decomposition is often adaptive to the local geometry. The most commonly used termination condition is: the division stops when a box contains less than some constant (typically $m = 100$) number of particles (See Figure 9.6).

In 2D case, the height of the tree is usually $log_2\sqrt{N}$. This is in the order of . The complexity of the problem is $N \cdot O(log(N))$.

Octtree is the three-dimension version of quadtree. The root is a box covering the entire set of particles. Octtree are constructed by recursively and adaptively dividing a box into eight child-boxes, by splitting it with hyperplanes normal to each axes through its center (See Figure 9.7).

9.9 Barnes-Hut Method (1986)

The Barnes-Hut method uses these clustered data structures to represent the bodies in the simulation, and takes advantage of the distant-body simplification mentioned earlier to reduce computational complexity to $O(n \log(n))$.

The method of Barnes and Hut has two steps.

1. **Upward evaluation of center of mass**

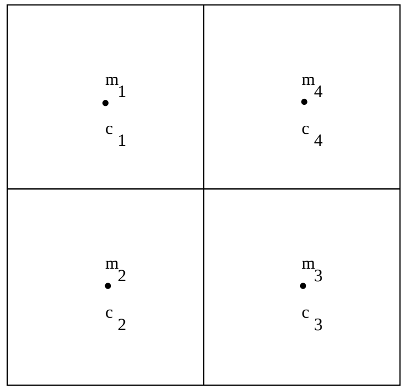

Figure 9.8: Computing the new Center of Mass

Refer to Figure 9.6 for the two dimensional case. Treating each box as a uniform cluster, the center of mass may be hierarchically computed. For example, consider the four boxes shown in Figure 9.8.

The total mass of the system is

$$m = m_1 + m_2 + m_3 + m_4 \tag{9.5}$$

and the center of mass is given by

$$\vec{c} = \frac{m_1\vec{c_1} + m_2\vec{c_2} + m_3\vec{c_3} + m_4\vec{c_4}}{m} \tag{9.6}$$

The total time required to compute the centers of mass at all layers of the quadtree is proportional to the number of nodes, or the number of bodies, whichever is greater, or in Big-O notation, $O(n+v)$, where v is for *vertex*

This result is readily extendible to the three dimensional case.

Using this approximation will lose some accuracy. For instance, in 1D case, consider three particles locate at $x = -1, 0, 1$ with strength $m = 1$. Consider these three particles as a cluster, the total potential is

$$V(x) = \frac{1}{x} + \frac{1}{x-1} + \frac{1}{x+1}.$$

Expand the above equation using Taylor's series,

$$V(x) = \frac{3}{x} + \frac{2}{x^3} + \frac{2}{x^5} + \ldots.$$

. It is seen that high order terms are neglected. This brings the accuracy down when x is close to the origin.

2. **Pushing the particle down the tree**
   Consider the case of the octtree i.e. the three dimensional case. In order to evaluate the potential at a point $\vec{x}_i$, start at the top of the tree and move downwards. At each node, check whether the corresponding box, b, is well separated with respect to $\vec{x}_i$ (Figure 9.9).

   Let the force at point $\vec{x}_i$ due to the cluster b be denoted by $\vec{F}(i, b)$. This force may be calculated using the following algorithm:

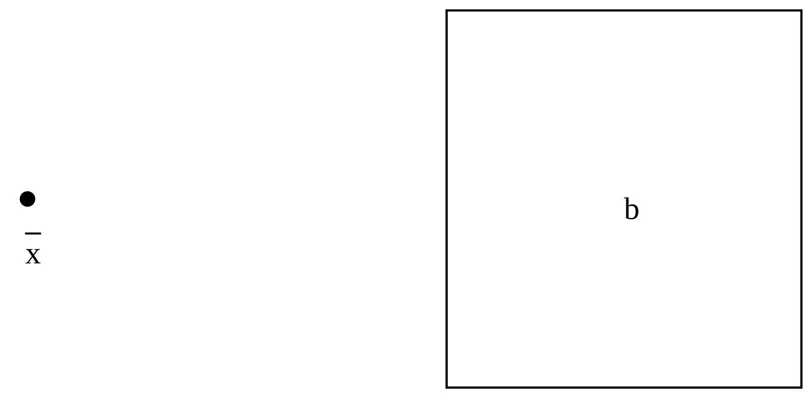

Figure 9.9: Pushing the particle down the tree

- *if b is "far" i.e. well separated from $\vec{x}_i$*, then

$$\vec{F}(\vec{x}_i) := \vec{F}(\vec{x}_i) + \frac{G m_x M_b(\vec{x} - \vec{c_b})}{||\vec{x}_i - \vec{c_b}||^3} \qquad \text{in } \Re^3 \tag{9.7}$$

- *else if b is "close" to $\vec{x}_i$*

$$\begin{aligned} & for\ k = 1\ to\ 8 \\ & \qquad \vec{F}(\vec{x}_i) = \vec{F}(\vec{x}_i) + \vec{F}(i, child(b,k)) \end{aligned} \tag{9.8}$$

$$\tag{9.9}$$

The computational complexity of pushing the particle down the tree has the upper bound $9hn$, where h is the height of the tree and n is the number of particles. (Typically, for more or less uniformly distributed particles, $h = \log_4 n$.)

9.9.1 Approximating potentials

We now rephrase Barnes and Hut scheme in term of potentials. Let

$$\begin{aligned} m_A &= \text{total mass of particles in } \mathbf{A} \\ m_B &= \text{total mass of particles in } \mathbf{B} \\ \vec{c_A} &= \text{center of mass of particles in } \mathbf{A} \\ \vec{c_B} &= \text{center of mass of particles in } \mathbf{B} \end{aligned}$$

The potential at a point $\vec{x}$ due to the cluster $\mathbf{B}$, for example, is given by the following second order approximation:

$$\phi(\vec{x}) \approx \frac{m_B}{||\vec{x} - \vec{c_B}||}(1 + \frac{1}{\delta^2}) \qquad \text{in } \Re^3 \tag{9.10}$$

In other words, each cluster may be regarded as an individual particle when the cluster is sufficiently far away from the evaluation point $\vec{x}$.

A more advanced idea is to keep track of a higher order (Taylor expansion) approximation of the potential function induced by a cluster. Such an idea provides better tradeoff between time required and numerical precision. The following sections provide the two dimensional version of the fast multipole method developed by Greengard and Rokhlin.

The Barnes-Hut method discussed above uses the *particle-cluster* interaction between two well-separated clusters. Greengard and Rokhlin showed that the *cluster-cluster* intersection among well-separated clusters can further improve the hierarchical method. Suppose we have k clusters B_1 ..., B_k that are well-separated from a cluster A. Let $\Phi_i^p()$ be the pth order multipole expansion of B_i. Using particle-cluster interaction to approximate the far-field potential at A, we need to perform $g(p,d)|A|(|B_1|+|B_2|+...+|B_k|)$ operations. Greengard and Rokhlin [44] showed that from $\Phi_i^p()$ we can efficiently compute a Taylor expansion $\Psi_i^p()$ centered at the centroid of A that approximates $\Phi_i^p()$. Such an operation of transforming $\Phi_i^p()$ to $\Psi_i^p()$ is called a *FLIP*. The cluster-cluster interaction first flips $\Phi_i^p()$ to $\Psi_i^p()$; we then compute $\Psi_A^p() = \sum_{i=1}^{k} \Psi_i^p()$ and use $\Psi_A^p()$ to evaluate the potential at each particle in A. This reduces the number of operations to the order of

$$g(p,d)(|A|+|B_1|+|B_2|+...+|B_k|).$$

9.10 Outline

- Introduction
- Multipole Algorithm: An Overview
- Multipole Expansion
- Taylor Expansion
- Operation No. 1 — SHIFT
- Operation No. 2 — FLIP
- Application on Quad Tree
- Expansion from 2-D to 3-D

9.11 Introduction

For N-body simulations, sometimes, it is easier to work with the (gravitational) potential rather than with the force directly. The force can then be calculated as the gradient of the potential.

In two dimensions, the potential function at z_j due to the other bodies is given by

$$\begin{aligned}\phi(z_j) &= \sum_{i=1, i\neq j}^{n} q_i \log(z_j - z_i) \\ &= \sum_{i=1, i\neq j}^{n} \phi_{z_i}(z_j)\end{aligned}$$

with

$$\phi_{z_i}(z) = q_i \log|z - z_i|$$

where $z_1, \ldots, z_n$ the position of particles, and $q_1, \ldots, q_n$ the strength of particles. The potential due to the bodies in the rest of the space is

$$\phi(z) = \sum_{i=1}^{n} q_i \log(z - z_i)$$

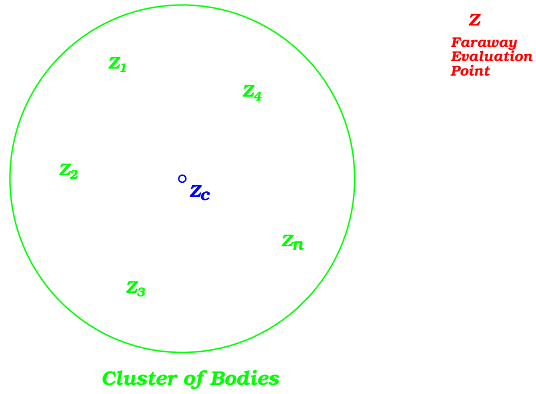

Figure 9.10: Potential of Faraway Particle due to Cluster

which is *singular* at each potential body. (Note: actually the potential is $Re\ \phi(z)$ but we take the complex version for simplicity.)

With the Barnes and Hut scheme in term of potentials, each cluster may be regarded as an individual particle when the cluster is sufficiently far away from the evaluation point. The following sections will provide the details of the fast multipole algorithm developed by Greengard and Rokhlin.

Many people are often mystified why the Green's function is a logarithm in two dimensions, while it is $1/r$ in three dimensions. Actually there is an intuitive explanation. In d dimensions the Green's function is the integral of the force which is proportional $1/r^{d-1}$. To understand the $1/r^{d-1}$ just think that the lines of force are divided "equally" on the sphere of radius r. One might wish to imagine an d dimensional ball with small holes on the boundary filled with d dimensional water. A hose placed at the center will force water to flow out radially at the boundary in a uniform manner. If you prefer, you can imagine 1 ohm resistors arranged in a polar coordinate manner, perhaps with higher density as you move out in the radial direction. Consider the flow of current out of the circle at radius r if there is one input current source at the center.

9.12 Multipole Algorithm: An Overview

There are three important concepts in the multipole algorithm:

- function representations (multipole expansions and Taylor series)
- operators to change representations (SHIFTs and FLIPs)
- the general tree structure of the computation

9.13 Multipole Expansion

The multipole algorithm flips between two point of views, or to be more precise, two representations for the potential function. One of them, which considers the cluster of bodies corresponding to many far away evaluation points, is treated in detail here. This part of the algorithm is often called the *Multipole Expansion*.

In elementary calculus, one learns about Taylor expansions for functions. This power series represents the function perfectly within the radius of convergence. A multipole expansion is also a perfectly valid representation of a function which typically converges *outside* a circle rather than inside. For example, it is easy to show that

$$\begin{aligned}\phi_{z_i}(z) &= q_i \log(z - z_i) \\ &= q_i \log(z - z_c) + \sum_{k=1}^{\infty} -\frac{q_i}{k}\left(\frac{z_i - z_c}{z - z_c}\right)^k\end{aligned}$$

where z_c is any complex number. This series converges in the region $|z - z_c| > |z - z_i|$, i.e., outside of the circle containing the singularity. The formula is particularly useful if $|z - z_c| \gg |z - z_i|$, i.e., if we are far away from the singularity.

Note that

$$\begin{aligned}\phi_{z_i}(z) &= q_i \log(z - z_i) \\ &= q_i \log[(z - z_c) - (z_i - z_c)] \\ &= q_i \left[\log(z - z_c) + \log(1 - \frac{z_i - z_c}{z - z_c})\right]\end{aligned}$$

The result follows from the Taylor series expansion for $\log(1 - x)$. The more terms in the Taylor series that are considered, the higher is the order of the approximation.

By substituting the single potential expansion back into the main equation, we obtain the multipole expansion as following

$$\begin{aligned}\phi(z) &= \sum_{i=1}^{n} \phi_{z_i}(z) \\ &= \sum_{i=1}^{n} q_i \log(z - z_c) + \sum_{i=1}^{n}\sum_{k=1}^{\infty} q_i \left(-\frac{1}{k}\left(\frac{z_i - z_c}{z - z_c}\right)^k\right) \\ &= Q \log(z - z_c) + \sum_{k=1}^{\infty} a_k \left(\frac{1}{z - z_c}\right)^k\end{aligned}$$

where

$$a_k = -\sum_{i=1}^{n} \frac{q_i (z_i - z_c)^k}{k}$$

When we truncate the expansion due to the consideration of computation cost, an error is introduced into the resulting potential. Consider a *p-term* expansion

$$\phi_p(z) = Q \log(z - z_c) + \sum_{k=1}^{p} a_k \frac{1}{(z - z_c)^k}$$

An error bound for this approximation is given by

$$||\phi(z) - \phi_p(z)|| \leq \frac{A}{(|\frac{z - z_c}{r}| - 1)} \left|\frac{r}{z - z_c}\right|^p$$

where r is the radius of the cluster and

$$A = \sum_{i=1}^{n} |q_i|$$

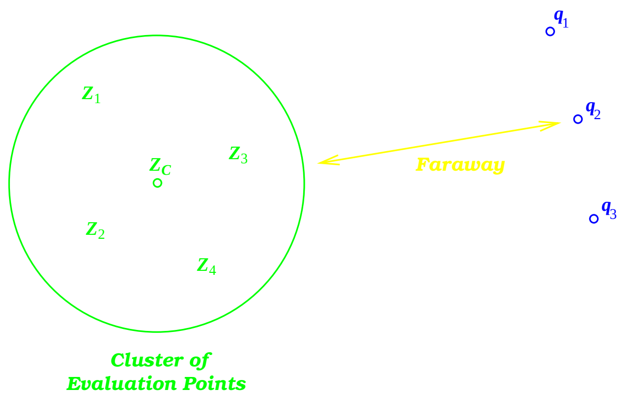

Figure 9.11: Potential of Particle Cluster

This result can be shown as the following

$$
\begin{aligned}
\text{Error} &= \left| \sum_{k=p+1}^{\infty} a_k \frac{1}{(z-z_c)^k} \right| \\
&= \left| - \sum_{k=p+1}^{\infty} \sum_{i=1}^{n} \frac{q_i}{k} \left(\frac{z_i - z_c}{z - z_c} \right)^k \right| \\
&\leq \sum_{k=p+1}^{\infty} \sum_{i=1}^{n} |q_i| \left| \frac{r}{z - z_c} \right|^k \\
&\leq A \sum_{k=p+1}^{\infty} \left| \frac{r}{z - z_c} \right|^k \\
&\leq A \frac{\left| \frac{r}{z-z_c} \right|^{p+1}}{1 - \left| \frac{r}{z-z_c} \right|} \\
&\leq \frac{A}{\left(\left| \frac{z-z_c}{r} \right| - 1 \right)} \left| \frac{r}{z - z_c} \right|^p
\end{aligned}
$$

At this moment, we are able to calculate the potential of each particle due to cluster of far away bodies, through multipole expansion.

9.14 Taylor Expansion

In this section, we will briefly discuss the other point of view for the multipole algorithm, which considers the cluster of evaluation points with respect to many far away bodies. It is called Taylor Expansion. For this expansion, each processor "ownes" the region of the space defined by the cluster of evaluation points, and compute the potential of the cluster through a Taylor series about the center of the cluster z_c.

Generally, the local Taylor expansion for cluster denoted by C (with center z_c) corresponding to some body z has the form

$$\phi_{C,z_c}(z) = \sum_{k=0}^{\infty} b_k(z - z_c)^k$$

Denote $z - z_i = (z - z_c) - (z_i - z_c) = -(z_i - z_c)(1 - \xi)$. Then for z such that $|z - z_c| < \min(z_c, C)$, we have $|\xi| < 1$ and the series $\phi_{C,z_c}(z)$ converge:

$$\begin{aligned}
\phi_{C,z_c}(z) &= \sum_C q_i \log(-(z_i - z_c)) + \sum_C q_i \log(1 - \xi) \\
&= \sum_C q_i \log(-(z_i - z_c)) + \sum_{k=1}^{\infty} \left(\sum_C q_i \right) k^{-1} (z_i - z_c)^{-k} (z - z_c)^k \\
&= b_0 + \sum_{k=1}^{\infty} b_k (z - z_c)^k
\end{aligned}$$

where formulæ for coefficients are

$$b_0 = \sum_C q_i \log(-(z_i - z_c)) \quad \text{and} \quad b_k = k^{-1} \sum_C q_i (z_i - z_c)^{-k} \quad k > 0.$$

Define the *p-order* truncation of local expansion ϕ^p_{C,z_c} as follows

$$\phi^p_{C,z_c}(z) = \sum_{k=0}^{p} b_k(z - z_c)^k.$$

We have error bound

$$\begin{aligned}
\left| \phi_{C,z_c}(z) - \phi^p_{C,z_c}(z) \right| &= \left| \sum_{k=p+1}^{\infty} k^{-1} \sum_C q_i \left(\frac{z - z_c}{z_i - z_c} \right)^k \right| \\
&\leq \frac{1}{p+1} \sum_C |q_i| \sum_{k=p+1}^{\infty} \left| \frac{z - z_c}{\min(z_c, C)} \right|^k = \frac{A}{(1+p)(1-c)} c^{p+1},
\end{aligned}$$

where $A = \sum_C |q_i|$ and $c = |z - z_c| / \min(z_c, C) < 1$.

By now, we can also compute the local potential of the cluster through the Taylor expansion. During the process of deriving the above expansions, it is easy to see that

- Both expansions are singular at the position of any body;
- Multipole expansion is valid outside the cluster under consideration;
- Taylor expansion converges within the space defined the cluster.

At this point, we have finished the basic concepts involved in the multipole algorithm. Next, we will begin to consider some of the operations that could be performed on and between the expansions.

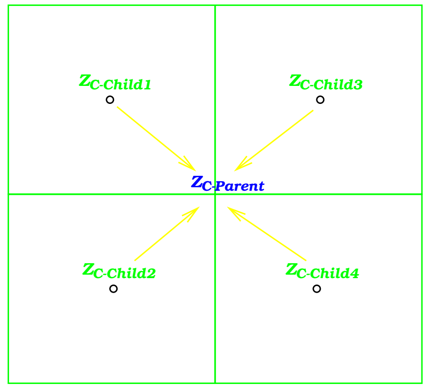

Figure 9.12: SHIFT the Center of Reference

9.15 Operation No.1 — SHIFT

Sometimes, we need to change the location of the center of the reference for the expansion series, either the multipole expansion or the local Taylor expansion. To accomplish this goal, we will perform the SHIFT operation on the expansion series.

For the multipole expansion, consider some far away particle with position z such that both series ϕ_{z_0} and ϕ_{z_1} , corresponding to different center of reference z_0 and z_1, converge: $|z - z_0| > \max(z_0, C)$ and $|z - z_1| > \max(z_1, C)$. Note that

$$z - z_0 = (z - z_1) - (z_0 - z_1) = (z - z_1)\left(1 - \frac{z_0 - z_1}{z - z_1}\right) = (z - z_1)(1 - \xi)$$

for appropriate ξ and if we also assume z sufficiently large to have $|\xi| < 1$, we get identity

$$(1 - \xi)^{-k} = \left(\sum_{l=0}^{\infty} \xi^l\right)^k = \sum_{l=0}^{\infty} \binom{k + l - 1}{l} \xi^l.$$

Now, we can express the SHIFT operation for multipole expansion as

$$\begin{aligned}
\phi_{z_1}(z) &= SHIFT\,(\phi_{z_0}(z), z_0 \Rightarrow z_1) \\
&= SHIFT\left(a_0 \log(z - z_0) + \sum_{k=1}^{\infty} a_k (z - z_0)^{-k}, z_0 \Rightarrow z_1\right) \\
&= a_0 \log(z - z_1) + a_0 \log(1 - \xi) + \sum_{k=1}^{\infty} a_k (1 - \xi)^{-k} (z - z_1)^{-k} \\
&= a_0 \log(z - z_1) - a_0 \sum_{k=1}^{\infty} k^{-1} \xi^k + \sum_{k=1}^{\infty} \sum_{l=1}^{\infty} a_k \binom{k + l - 1}{l} \xi^l (z - z_1)^{-k} \\
&= a_0 \log(z - z_1) + \sum_{l=1}^{\infty} \left(\sum_{k=1}^{l} a_k (z_0 - z_1)^{l-k} \binom{l - 1}{k - 1} - a_0 l^{-1} (z_0 - z_1)^l\right) (z - z_1)^{-l}
\end{aligned}$$

We can represent $\phi_{z_1}(z)$ as a sequence of its coefficients a'_k:

$$a'_0 = a_0 \quad \text{and} \quad a'_l = \sum_{k=1}^{l} a_k (z_0 - z_1)^{l-k} \binom{l - 1}{k - 1} - a_0 l^{-1} (z_0 - z_1)^l \quad l > 0.$$

Note that a'_l depends only on $a_0, a_1, \ldots, a_l$ and not on the higher coefficients. It shows that given $\phi^p_{z_0}$ we can compute $\phi^p_{z_1}$ exactly, that is without any further error! In other words, operators SHIFT and truncation commute on multipolar expansions:

$$SHIFT(\phi^p_{z_0}, z_0 \Rightarrow z_1) = \phi^p_{z_1}.$$

Similarly, we can obtain the SHIFT operation for the local Taylor expansion, by extending the operator on the domain of local expansion, so that $SHIFT(\phi_{C,z_0}, z_0 \Rightarrow z_1)$ produces ϕ_{C,z_1}. Both series converges for z such that $|z - z_0| < \min(z_0, C)$, $|z - z_1| < \min(z_1, C)$. Then

$$\begin{aligned}
\phi_{C,z_1}(z) &= SHIFT\left(\phi_{C,z_0}(z), z_0 \Rightarrow z_1\right) \\
&= \sum_{k=0}^{\infty} b_k((z - z_1) - (z_0 - z_1))^k \\
&= \sum_{k=0}^{\infty} b_k \sum_{l=0}^{\infty} (-1)^{k-l} \binom{k}{l} (z_0 - z_1)^{k-l} (z - z_1)^l \\
&= \sum_{l=0}^{\infty} \left(\sum_{k=l}^{\infty} b_k (-1)^{k-l} \binom{k}{l} (z_0 - z_1)^{k-l} \right) (z - z_1)^l
\end{aligned}$$

Therefore, formula for transformation of coefficients b_k of ϕ_{C,z_0} to b'_l of ϕ_{C,z_1} are

$$b'_l = \sum_{k=l}^{\infty} a_k (-1)^{k-l} \binom{k}{l} (z_0 - z_1)^{k-l}.$$

Notice that in this case, b'_l depends only on the higher coefficients, which means knowledge of the coefficients $b_0, b_1, \ldots, b_p$ from the truncated local expansion in z_0 does *not* suffice to recover the coefficients $b'_0, b'_1, \ldots b'_p$ at another point z_1. We do incur an error by the SHIFT operation applied to truncated local expansion:

$$\begin{aligned}
\left| SHIFT(\phi^p_{C,z_0}, z_0 \Rightarrow z_1) - \phi^p_{C,z_1} \right| &= \left| \sum_{l=0}^{\infty} \left(\sum_{k=p+1}^{\infty} b_k (-1)^{k-l} (z_0 - z_1)^{k-l} \right) (z - z_1)^l \right| \\
&\leq \left| \sum_{k=p+1}^{\infty} b_k (z_1 - z_0)^k \right| \left| \sum_{l=0}^{\infty} \left(\frac{z - z_1}{z_1 - z_0} \right)^l \right| \\
&= \left| \sum_{k=p+1}^{\infty} k^{-1} \sum_{C} q_i \left(\frac{z_1 - z_0}{z_i - z_0} \right)^k \right| \left| \sum_{l=0}^{\infty} \left(\frac{z - z_1}{z_0 - z_1} \right)^l \right| \\
&\leq \frac{A}{(p+1)(1-c)(1-D)} c^{p+1},
\end{aligned}$$

where $A = \sum_C |q_i|$. $c = |z_1 - z_0| / \min(z_0, C)$ and $D = |z - z_1| / |z_0 - z_1|$.

At this moment, we have obtained all the information needed to perform the SHIFT operation for both multipole expansion and local Taylor expansion. Next, we will consider the operation which can transform multipole expansion to local Taylor expansion.

9.16 Operation No.2 — FLIP

At this section, we will introduce the more powerful operation in multipole algorithm, namely the FLIP operation. For now, we will consider only the transformation in the direction from the

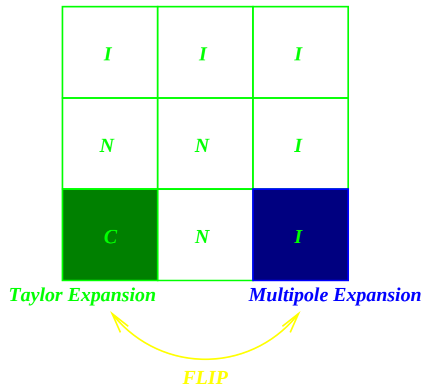

Figure 9.13: FLIP from Multipole to Taylor Expansion

multipole expansion $\phi_{z_0}(z)$ to the local Taylor expansion $\phi_{C,z_1}(z)$, denoted by

$$FLIP(\phi_{z_0}, z_0 \Rightarrow z_1) = \phi_{C,z_1}$$

For $|z - z_0| > \max(z_0, C)$ and $|z - z_1| < \min(z_1, C)$ both series converge. Note that

$$z - z_0 = -(z_0 - z_1)(1 - \frac{z - z_1}{z_0 - z_1}) = -(z_0 - z_1)(1 - \xi)$$

and assume also $|\xi| < 1$. Then,

$$\begin{aligned}
\phi_{z_0}(z) &= a_0 \log(z - z_0) + \sum_{k=1}^{\infty} a_k (z - z_0)^{-k} \\
&= a_0 \log(-(z_0 - z_1)) + a_0 \log(1 - \xi) + \sum_{k=1}^{\infty} a_k (-1)^k (z_0 - z_1)^{-k} (1 - \xi)^{-k} \\
&= a_0 \log(-(z_0 - z_1)) + \sum_{l=1}^{\infty} - a_0 l^{-1} \xi^l + \sum_{k=1}^{\infty} (-1)^k a_k (z_0 - z_1)^{-k} \sum_{l=0}^{\infty} \binom{k+l-1}{l} \xi^l \\
&= \left(a_0 \log(-(z_0 - z_1)) + \sum_{k=1}^{\infty} (-1)^k a_k (z_0 - z_1)^{-k} \right) + \\
&\qquad \sum_{l=1}^{\infty} \left(a_0 l^{-1} (z_0 - z_1)^{-l} + \sum_{k=1}^{\infty} (-1)^k a_k \binom{k+l-1}{l} (z_0 - z_1)^{-(k+l)} \right) (z - z_1)^l.
\end{aligned}$$

Therefore coefficients a_k of ϕ_{z_0} transform to coefficients b_l of ϕ_{C,z_1} by the formula

$$\begin{aligned}
b_0 &= a_0 \log(-(z_0 - z_1)) + \sum_{k=1}^{\infty} (-1)^k a_k (z_0 - z_1)^{-k} \\
b_l &= a_0 l^{-1} (z_0 - z_1)^{-l} + \sum_{k=1}^{\infty} (-1)^k a_k \binom{k+l-1}{l} (z_0 - z_1)^{-(k+l)} \quad l > 0
\end{aligned}$$

Note that FLIP does *not* commute with truncation since one has to know all coefficients $a_0, a_1, \ldots$ to compute $b_0, b_1, \ldots, b_p$ exactly. For more information on the error in case of truncation, see Greengard and Rokhlin (1987).

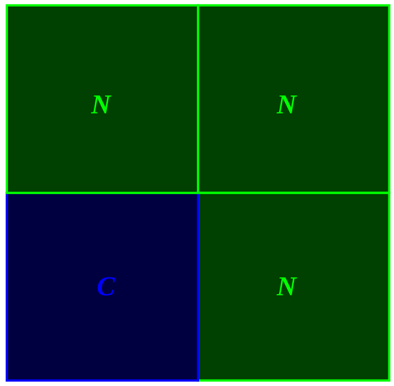

Figure 9.14: First Level of Quad Tree

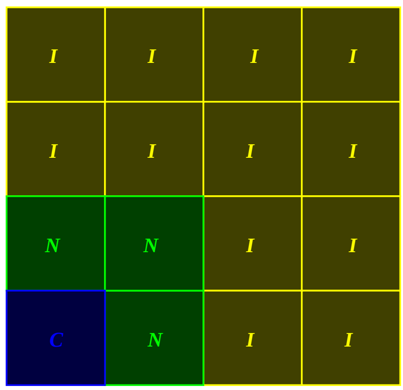

Figure 9.15: Second Level of Quad Tree

9.17 Application on Quad Tree

In this section, we will go through the application of multipole algorithm on quad tree in detail. During the process, we will also look into the two different operations $SHIFT$ and $FLIP$, and gain some experience on how to use them in real situations.

We will start at the lowest level h of the tree. For every node of the tree, it computes the multipole expansion coefficients for the bodies inside, with origin located at the center of the cell. Next, it will shift all of the four centers for the children cells into the center of the parent node, which is at the $h-1$ level, through the SHIFT operation for the multipole expansion. Adding up the coefficients from the four shifted expansion series, the multipole expansion of the whole parent node is obtained. And this SHIFT and ADD process will continue upward for every level of the tree, until the multipole expansion coefficients for each node of the entire tree are stored within that node. The computational complexity for this part is $O(N)$.

Before we go to the next step, some terms have to be defined first.

- NEIGHBOR — a neighbor N to a cell C is defined as any cell which shares either an edge or a corner with C
- INTERACTIVE — an interactive cell I to a cell C is defined as any cell whose parent is a neighbor to parent of C, excluding those which are neighbors to C

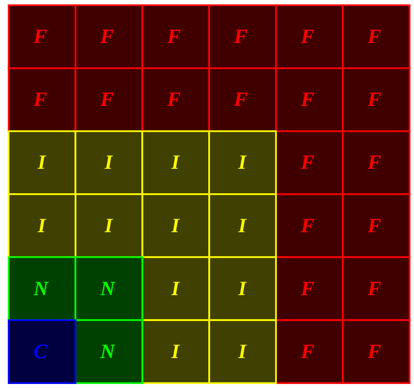

Figure 9.16: Third Level of Quad Tree

- FARAWAY — a faraway cell F to a cell C is defined as any cell which is neither a neighbor nor an interactive to C

Now, we start at the top level of the tree. For each cell C, FLIP the multipole expansion for the interactive cells and combine the resulting local Taylor expansions into one expansion series. After all of the FLIP and COMBINE operations are done, SHIFT the local Taylor expansion from the node in this level to its four children in the next lower level, so that the information is conserved from parent to child. Then go down to the next lower level where the children are. To all of the cells at this level, the faraway field is done (which is the interactive zone at the parent level). So we will concentrate on the interactive zone at this level. Repeat the FLIP operation to all of the interactive cells and add the flipped multipole expansion to the Taylor expansion shifted from parent node. Then repeat the COMBINE and SHIFT operations as before. This entire process will continue from the top level downward until the lowest level of the tree. In the end, add them together when the cells are *close* enough.

9.18 Expansion from 2-D to 3-D

For 2-D N-body simulation, the potential function is given as

$$\phi(z_j) = \sum_{i=1}^{n} q_i \log(z_j - z_i)$$

where $z_1, \ldots, z_n$ the position of particles, and $q_1, \ldots, q_n$ the strength of particles. The corresponding multipole expansion for the cluster centered at z_c is

$$\phi_{z_c}(z) = a_0 \log(z - z_c) + \sum_{k=1}^{\infty} a_k \frac{1}{(z - z_c)^k}$$

The corresponding local Taylor expansion looks like

$$\phi_{C,z_c}(z) = \sum_{k=0}^{\infty} b_k \frac{1}{(z - z_c)^k}$$

In three dimensions, the potential as well as the expansion series become much more complicated. The 3-D potential is given as

$$\Phi(x) = \sum_{i=1}^{n} q_i \frac{1}{||x - x_i||}$$

where $x = f(r, \theta, \phi)$. The corresponding multipole expansion and local Taylor expansion as following

$$\begin{aligned} \Phi_{multipole}(x) &= \sum_{n=0}^{\infty} \frac{1}{r^{n+1}} \sum_{m=-n}^{n} a_n^m Y_n^m(\theta, \phi) \\ \Phi_{Taylor}(x) &= \sum_{n=0}^{\infty} \sum_{m=-n}^{n} r^n b_n^m Y_n^m(\theta, \phi) \end{aligned}$$

where $Y_n^m(\theta, \phi)$ is the Spherical Harmonic function. For a more detailed treatment of 3-D expansions, see Nabors and White (1991).

9.19 Parallel Implementation

In Chapter **??**, we will discussion issues on parallel N-body implementation.