Seminar presentation

N-body algorithms

Tancred Lindholm, ctlindho@cc.hut.fi

Contents

Table of Figures

1. Introduction

Many physical phenomena directly or indirectly (when solving a discrete version of a continuous problem) involve, or can be simulated with particle systems, where each particle interacts with all other particles according to the laws of physics. Examples include the gravitational interaction among the stars in a galaxy or the Coulomb forces exerted by the atoms in a molecule. The challenge of efficiently carrying out the related calculations is generally known as the *N-body problem*.

Mathematically, the N-body problem can be formulated as

$$U(\mathbf{x}_0) = \sum_i F(\mathbf{x}_0, \mathbf{x}_i) \tag{1}$$

where $U(\mathbf{x_0})$ is a physical quantity at $\mathbf{x}_0$ which can be obtained by summing the pairwise interactions $F(\mathbf{x}_0,\mathbf{x}_i)$ over the particles of the system. For instance, assume a system of N particles, located at $\mathbf{x}_i$ and having a mass of m_i. The gravitational force exerted on a particle $\mathbf{x}$ having a mass m is then expressed as

$$F(\mathbf{x}) = \sum_{i=1}^{N} Gmm_i \frac{\mathbf{x} - \mathbf{x}_i}{\left|\mathbf{x} - \mathbf{x}_i\right|^3} \tag{2}$$

where G is the gravitational constant.

The task of evaluating the function $U(\mathbf{x}_0)$ for all N particles using (1) requires $O(n)$ operations for each particle, resulting in a total complexity of $O(n^2)$. In this paper we will see how this complexity can be reduced to $O(n \log n)$ or $O(n)$ by using efficient methods to approximate the sum in the right hand term of (1), while still preserving such important physical properties as energy and momentum.

2. Example N-body problem: 10 million-star galaxy

The summation techniques presented in this paper may be applied to a wide range of N-body problems. However, in order to clarify it is beneficial to have an actual problem to apply the ideas on. In this case, I have chosen the problem of simulating the movements of the stars of a galaxy. Let's assume that there are about $N:=10$ million starts in the galaxy, although this is clearly much less than "in the real world". Furthermore, for the sake of clarity the simulation will be done in two dimensions.

In the model, each star has the following quantities associated with it:

- mass, m_i
- position, $\mathbf{x}_i$ (depends on time)

Newtonian (let's not allow Einstein to mess things up) physics tells us that for each star

$$\frac{\partial^2}{\partial \mathbf{x}^2} \mathbf{x}_j(t) = \sum_{i=1, j\neq i}^{N} Gm_i \frac{\mathbf{x}_j - \mathbf{x}_i}{\left|\mathbf{x}_j - \mathbf{x}_i\right|^3} \tag{3}$$

Then, for each timestep from time t_k to $t_{k+1}:=\Delta t+t_k$ we need to integrate the right hand term of equation (3) in order to obtain the change in position:

$$\Delta \mathbf{x}_j = \iint_{[t_k, t_{k+1}]} \mathbf{F}(\mathbf{x}_j(t))dtdt \tag{4}$$

where

$$F(\mathbf{x}_j) = \sum_{i=1}^{N} Gm_i \frac{\mathbf{x}_j - \mathbf{x}_i}{\left|\mathbf{x}_j - \mathbf{x}_i\right|^3} \tag{5}$$

(4) is a somewhat difficult integral equation, since $\mathbf{x}_j$ is present on both sides. Also, $\mathbf{x}_i$ is dependent on *t*, which means we have a system of *N* coupled integral equations for each time step.

A discrete version of (4) (which can be obtained by making certain assumptions) has the general form[1]

$$\Delta \mathbf{x}_j = \sum_{i=1}^{k} c_i \mathbf{F}(\mathbf{x}_j(t + h_i)), \quad k < \infty \tag{6}$$

and is thus a linear combination of the function **F** evaluated at different time points; different discrete integration schemes yield different coefficients c_i and h_i. A commonly used integrator is the so-called *Leapfrog* integration scheme.

We can now formulate an algorithm for the simulation:

```
1. Set initial positions
2. for each timestep Δt do
3.   for each particle j do
4.     evaluate F(x_j(t)) at timepoints required by
          the integrator
5.     use the integrator to calculate Δx_j
6.     x_j(t+Δt)=x_j(t)+Δx
7.   endfor
8. endfor
```

The function **F** is of the form (1), and thus the N-body force calculation algorithms presented in this paper can be used to speed up step 4 of the algorithm.

3. The Particle-Particle (PP) method

The method of evaluating the right hand side of (1) directly is generally referred to as the Particle-Particle (PP) method[2]. This brute-force approach is clearly not feasible for large amount of particles due to the $O(n^2)$ time requirement, but can effectively be used for small amounts of particles[3]. As this method does not approximate the sum, the accuracy equals machine precision.

4. Tree codes

Let *a* be the radius of the smallest disc, call it *D*, so that the set of particles $P:=(\mathbf{x}_{i1},..,\mathbf{x}_{iN})$ are inside the disc. Many physical systems have the property that the field $U(\mathbf{x})$ generated by the the particle set *P* may be very complex inside *D*, but smooth ("low on information content") at some distance $c \cdot a$ from *D*. The gravitational force, for instance, has this property. [4]

This observation is used in the so-called *tree code*-approach to the N-body problem: Clusters of particles at enough distance from the "target particle" $\mathbf{x}_0$ of equation 1 are

given a computationally simpler representation in order to speed up summation[5]. This approach can be illustrated by the following example: when calculating the gravitational force exerted by earth on an orbiting satellite, we do not try to sum the gravitational forces of all atoms that constitute planet earth. Instead we approximate the force with that of an infinitesimal particle, which have the same mass as earth and is located at earth's center of mass.

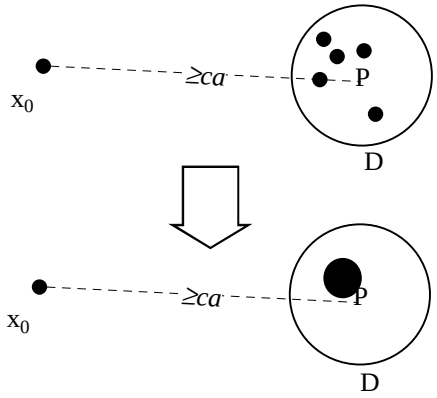

Figure 1 Approximating the set P

The tree-code approach is effective because it can be applied recursively[6]. For instance, in Figure 5, we might also be able to use a subset of $P' \subseteq P$ to approximate the force on the particle down to the left, as the radius a' of the disc D' enclosing P' is less than a, and thus the minimal distance for approximation, ca', is also smaller.

We need a data structure that supports the recursive nature of the idea, in 2D this is a *quadtree* and in 3D an *octree*. Each node in the quadtree is a square, and has four child nodes (unless it is a leaf node), representing a break-up into four smaller squares, ¼ the size of the original square (see Figure 2). An octree is built in a similar manner (each node has 8 children).

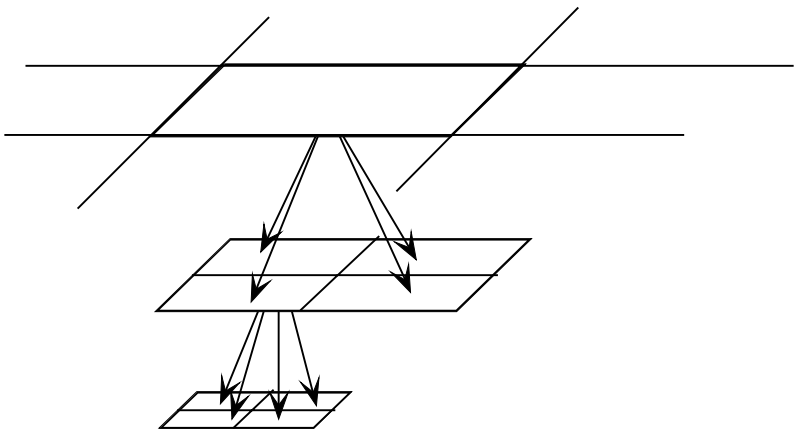

Figure 2 Quadtree

We now construct a quadtree, so that each leaf node contains only 1 particle. This is done by recursively subdividing the computational box; each node is further subdivided if it contains more than one particle.

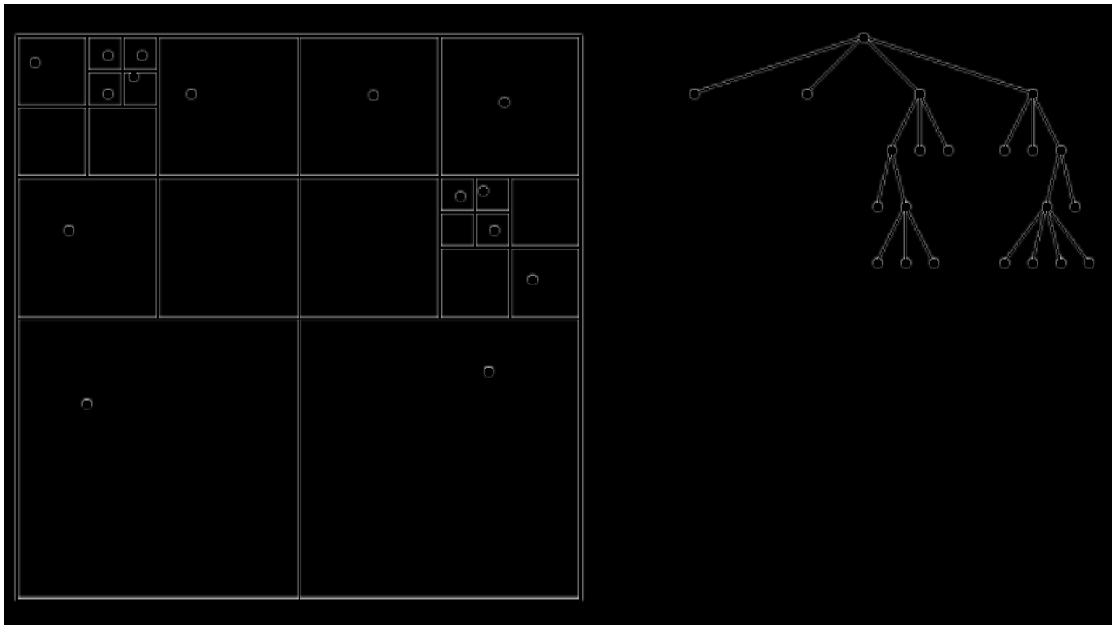

Figure 3 Adaptive quadtree with one particle/leaf. The picture is from [Demmel1]

Assuming the particles are not at arbitrarily small distances from each other (at least the machine precision sets a limit) a quadtree can be built in $O(n \min(b,\log n))$ time, where b is the machine precision[7].

Now assume that the distance between a cluster and a particle must be at least the length of the side of the cluster box in order to obtain an accurate approximation. When calculating the force on a particle $\mathbf{x}_0$, the tree is recursively traversed from the root. At each level, there may be no more than 9 boxes (the ones surrounding the box containing the particle) which need further subdivision, limiting the number of force calculations on the next level to 27 (=2*3*2*3-9, see Figure 4). Thus as each level a maximum of 27 $O(1)$ operations are performed. The depth of the tree is $\min(b,\log(n))$ yielding a total complexity (for all N particles) of $O(n \min(b,\log n))$.

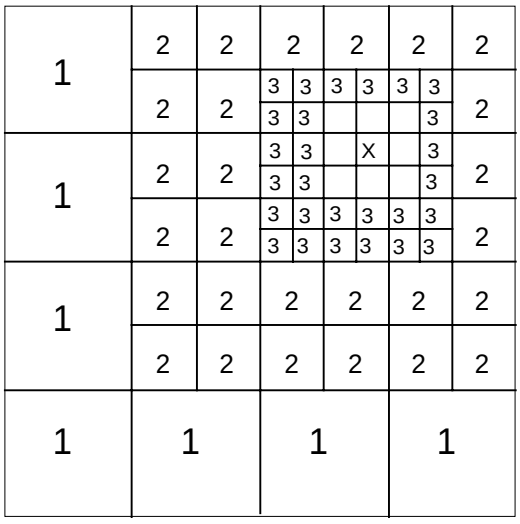

Figure 4 Quadtree nodes calculated at each level for particle x

Tree codes thus reduces the computational complexity from $O(n^2)$ to $O(n \log n)$ or $O(n)$ depending on your point of view - certainly a vast improvement! But as the

saying goes, there's no such thing as a free lunch: tree codes are less accurate than simple PP, and require more auxiliary storage[8].

5. The Barnes-Hut algorithm: applying tree codes.

The Barnes-Hut algorithm is a good example of how to use tree codes in an algorithm. The algorithm was presented in [Barnes&Hut] in 1986, and is widely used in astrophysics. The algorithm has also been successfully parallized.[9] This discussion of the algorithm is mainly based on [Demmel1], which describes Barnes-Hut for a 2D N-body system similar to our example.

The main idea is to approximate long-range forces by aggregating particles into one particle, and using the force exerted by this particle. A quadtree structure (or octree in 3D[10]), as described in the previous section, is used to store the particles. The tree steps of the algorithm are

```
1. Build quadtree as described in section 0
2. Traverse the quadtree from the leaves to the root, computing
   center of mass and total mass for each parent node.
3. For each particle, traverse the tree from the root,
   calculating the force during the traversal
```

Step 2

Step 2 calculates the approximations for the long-range force. The approximation is made by considering several particles as one, with a position equal to the center of mass of the approximated particles, and a mass the sum of the approximated particles' masses. More formally, to find the mass and position associated with a node N:

```
calculate_approxiamtions( N )
  if N is a leaf node
    return; // Node has a (real) particle => has mass & position
  for all children n of N do
    calculate_approximations( n )
  M := 0
  cm := (0,0)
  for all children n of N do
    M := M + mass of n
    cm := cm + mass of n * position of n
  endfor
  cm := 1/M * cm
  mass of N := M
  position of N := cm
end
```

Step 3

Consider the ratio,

$$\theta = \frac{D}{r} \qquad (7)$$

where *D* is the size of the current node (call it A) "box" and *r* is the distance to the center of mass of another node (called B). If this ratio is sufficiently small, we can use the center of mass and mass of B to compute the force in A. If this is not the case, we need to go to the children of B and do the same test. Figure 4 shows this for $\theta = 1.0$; the numbers indicate the relative depth of the nodes. It can clearly be seen that further away from the particle, x, large nodes are used and closer to it smaller. The method is accurate to approximately 1% with $\theta = 1.0$[11]. Expressed in pseudocode

```
treeForce(x, N)
  if N is a leaf or size(N)/|x-N|<θ
   return force(x,N)
  else
   F := 0
   for all children n of N do
     F := F + treeForce(x, n)
   endfor
   return F
  endif
end
```

6. The Particle-Mesh (PM) method

Whereas we have discretized time, position is still continuous. The PM method goes one step further: it effectively discretizes position too[12]. However, before we explore this idea further, we need the concept of potential.

Let us assume that there exists a quantity Φ, which is related to the physical quantity U we are studying according to:

$$U = \nabla\Phi \tag{8}$$

and, furthermore, that

$$\nabla \bullet U = c\rho \tag{9}$$

where ρ is the density function (e.g. mass, charge) obtained from the particle distribution and c is a constant . This leads to the Poisson equation[13]

$$\nabla^2\Phi = c\rho \tag{10}$$

In the sample N-body problem U corresponds to the force, and Φ to the potential energy in the gravitational field generated by the particles. ρ is the mass density (mass/area unit). In the continued discussion of the PM method the quantities used will be these.

The idea of the PM method is that we set up a mesh (grid) over the computational box, and then solve the potential (i.e. Poisson's equation) at the meshpoints. Forces at the meshpoints can then be obtained by calculating the gradient of the potential. To find the force on a particle not located at a meshpoint we can either use the force at the nearest meshpoint, or interpolate the force from the closest meshpoints.[14]

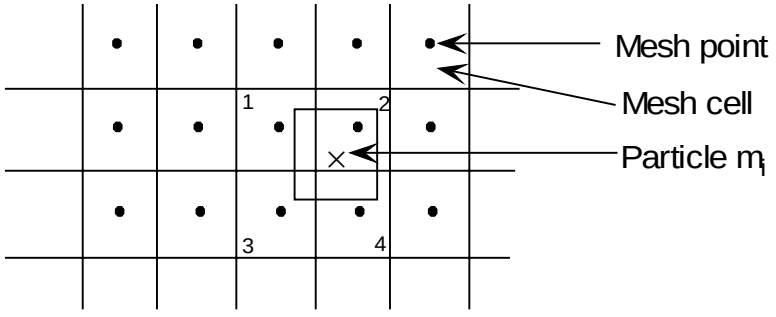

Figure 5 Mesh density function assignment

6.1 Creating the density function ρ

There are several ways of assigning the particle masses to the density function[15] ρ:

Nearest gridpoint (NGP): The mass of each particle m_i is assigned to the gridpoint closest to the mass. In Figure 5, this would mean assigning the entire mass of the particle to cell 2. NGP is also referred to as zero-order interpolation

Cloud-in-Cell (CIC): The mass of each particle is weighted over the four (in 2D) closest cells; the weighting is proportional to the intersection of the "cloud" surrounding the particle and the cell. In Figure 5 almost all mass would be assigned to cell 2, then approximately the same amount to cells 1 and 4, and finally about 1/16 to cell 3. CIC implements first order (linear) interpolation.

Higher order interpolations: The "cloud" (weighting function) around the particle can be made to cover even more cells, resulting in higher order interpolations, e.g. TSC (triangular shaped cloud).

6.2 Calculating the potentials

Now that we have the density function ρ, we can solve (10) in order to obtain the potential at the meshpoints. This is done by rewriting (10) as a system of discrete difference equations, and solving the system. The system can be solved in $O(G \log G)$ time, where G is the number of gridpoints, by using the Fast Fourier Transform.[16]

6.3 Finding the forces

Thus the potential is solved. Calculating the force is easy, we only need to calculate the gradient of the potential by taking the difference between two potential values. Some special considerations regarding the degree of force interpolation must however be taken in order to conserve such physical properties as momentum and energy.[17]

6.4 PM pros and cons

The PM method has a complexity of $O(n)$ with respect to the particles, the slowest steps being the FFT requiring $O(G \log G)$ operations.[18] The cost of the speed is the limited spatial resolution of PM - phenomena at a smaller scale than the mesh spacing are not modelled accurately, i.e. it is only suited for modelling of systems where collisions occur. On the other hand, large-scale phenomena can be shown quite accurately.[19]

7. The Particle-Particle/Particle-Mesh algorithm (P^3M)

The Particle-Particle/Particle-Mesh algorithm is a hybrid algorithm that strives to correct the shortcomings of PM when it comes to modelling of short-range phenomena. The following discussion is based on [Hockney] [20] and applied to our sample N-body problem.

The general idea is to split up the gravitational force in short-range and long-range forces, just as we did in section 0. The PM method is used to determine the long range force, and PP to correct the PM force at short distances.

Let r_e be the distance (typically 2-3 times the size of the mesh cell), below which we want to use the PP method for force calculation. Denote by $\mathbf{x}_0$ the position where the force is to be evaluated. In order to be able to efficiently find the particles $\mathbf{x}_i$ whose distance from $\mathbf{x}_0$ is less than r_e we employ as so-called *chaining mesh* for the particles,

see Figure 6. As can be see from the figure, those $\mathbf{x}_i$ closer to $\mathbf{x}_0$ than r_e must be in the neighbouring 8 chaining mesh cells of $\mathbf{x}_0$'s chaining mesh cell.

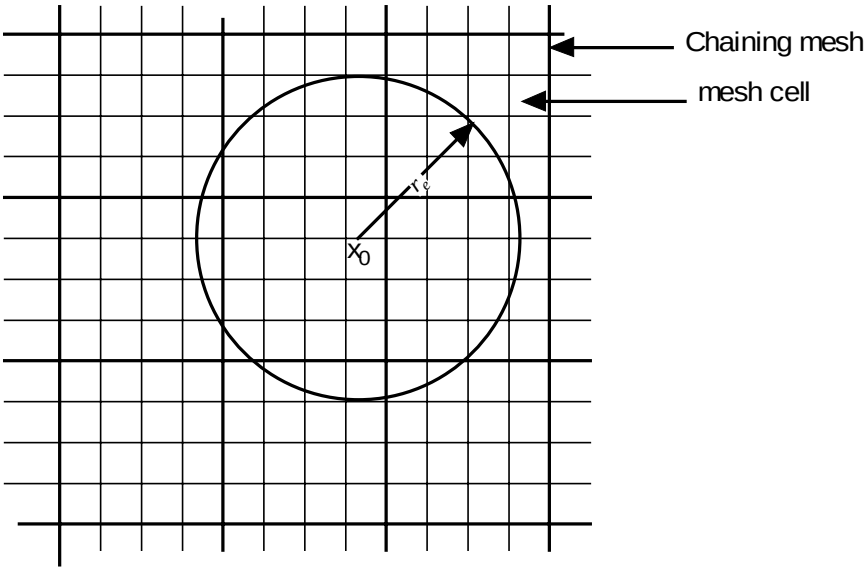

Figure 6 Mesh cells and chaining mesh

Mathematically, we define the correction, or short-range, force $\mathbf{f}^{sr}$ as

$$\mathbf{f}^{tot} = \mathbf{R} + \mathbf{f}^{sr} \Rightarrow \mathbf{f}^{sr} = \mathbf{f}^{tot} - \mathbf{R} \qquad (11)$$

where $\mathbf{f}^{tot}$ is the total force and $\mathbf{R}$ is the force obtained using the PM method.

Suppose we have a particle $\mathbf{x}_i$ closer to than r_e. We'd like to calculate the total force on $\mathbf{x}_0$, using direct PP to correct the force $\mathbf{R}$ obtained by PM calculations. With some lengthy mathematical manipulations it can be shown that PM treats the particle $\mathbf{x}_i$ as a "mass cloud" (the density and shape of which depend on the details of the PM method). Thus, we can correct the PM force by subtracting the force induced by such a "mass cloud" (this force is included in the PM force) and then adding the direct PP interaction force. Mathematically, in the case of pointlike masses

$$\mathbf{f}^{sr} = Gmm_i \frac{\mathbf{x}_0 - \mathbf{x}_i}{\left|\mathbf{x}_0 - \mathbf{x}_i\right|^3} - \mathbf{e}(\mathbf{x}_0, \mathbf{x}_i) \qquad (12)$$

where $\mathbf{e}(\mathbf{x}_0,\mathbf{x}_i)$ is the function for the gravitational force of the "mass cloud" the PM method treats $\mathbf{x}_i$ as. We can now formulate the P^3M algorithm:

```
1.for each particle x do
2.  find force on x=:F using the PM method
3.  locate particles closer than r_e for PP calculation using the
    chaining mesh. Calculate short range force using these
    particles
4.  force on x:= F + short range forces
5. endfor
```

The P^3M method has been used widely in cosmological simulations, and is easy to use when forces can be easily split into short and long-range (i.e. gravity). A problem is that the algorithm easily becomes dominated by the PP phase.[21]

8. Fast multipole method (FMM)

Multipole expansion can be viewed as a tree code, with the following main differences

- Potential instead of force is used
- Multipole expansions of the potential of a box are more accurate than a simple center of mass substitution, but also computationally "heavier". This is compensated by having more than 1 particle per tree leaf.[22]
- FMM only require evaluation of $\mathbf{F}(\mathbf{x}_0,\mathbf{x}_i)$ at leaf nodes, and possesses a "richer analytical structure", meaning it can be adapted to a wider variety of problems[23]

8.1 Potentials and multipole expansion of the gravitational field in 2D

The mathematics associated with FMM is somewhat lengthy, but not excessively advanced. This section will deal with multipole expansions of the gravitational field of our example N-body problem, similar methods are used when expanding other quantities and/or in other dimensionalities[24]. The following presentation of the mathematical ideas is based on [Demmel2].

Recall that the potential of a particle satisfies Poissions equation (equation 10). A solution to (10) for a point mass located at $\mathbf{x}_0$ in 2D is

$$\Phi(\mathbf{x}) = \log(|\mathbf{x} - \mathbf{x}_o|) \qquad (13)$$

Using the complex number $z=a+b\mathbf{i}$ to represent the point $\mathbf{x}=(a,b)$ the potential can be rewritten as the real part of the complex logarithm, which is analytic. Remembering that potentials are additive, the total potential from n particles can be expressed as

$$\Phi(z) = \sum_{i=1}^{n} m_i \log(z - z_i) = \sum_{i=1}^{n} m_i (\log(z) + \log(1 - z_i / z)) \qquad (14)$$

Since $\Phi(z)$ is analytic, it is possible to do a Taylor expansion around origo of $\log(1-z_i/z)$, yielding

$$\Phi(z) = M \log(z) + \sum_{j=1}^{\infty} \alpha_j z^{-j} \qquad (12)$$

where M is the combined mass of all particles and

$$\alpha_j = \sum_{i=1}^{n} \frac{m_i z_i{}^j}{j} \qquad (13)$$

We approximate the potential $\Phi(z)$ by summing a finite number, p, of terms in (12). The error is then proportional to

$$\varepsilon = \left(\frac{\max(|z_i|)}{|z|} \right)^{p+1} \qquad (14)$$

Now, suppose that all z_i lie inside a DD square centered at origin and z is evaluated outside a 3D·3D square centered at origin, then $\varepsilon \propto 2.12^{-p}$ (see Figure 7). We say that z and z_i are well-separated when this condition holds. Also note that the potential is expressed as a power series, and the gradient can thus easily be computed analytically, avoiding further discretization errors (compare to PM!).

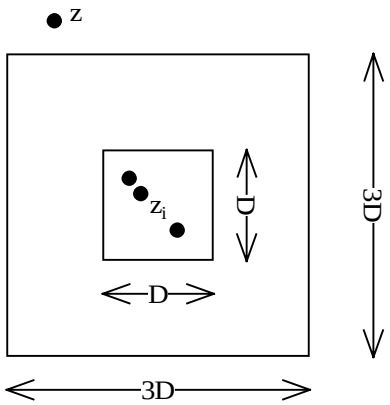

Figure 7 The expansion of the particles z_i converge outside the outer box

Expansion around a point z_c instead of origo is similar. Denote by `outer(M,`α_1`,...,`α_p`,`z_c`)` this expansion.

Similarly, we can do an *inner expansion* which approximates the potential from particles outside a 3D·3D box and is valid in a D·D box in the center of the 3D·3D box. This is denoted by `inner(M,`β_1`,...,`β_p`,`z_c`)`

Furthermore, we'll need functions to translate the center of expansion for the inner and outer expansions. These functions are defined as

`outer(M,`α_1`,...,`α_p`,`z_c`')=outer_shift(outer(M,`α_1`,...,`α_p`,`z_c`),`z_c`')`

and

`inner(M,`β_1`,...,`β_p`,`z_c`')=inner_shift(inner(M,`β_1`,...,`β_p`,`z_c`),`z_c`')`

These functions can be implemented in $O(p^2)$

Finally a conversion function between outer and inner expansions is needed (i.e. when an outer expansion of a box A is valid in the region of another box B, the outer expansion of A can be converted to an inner expansion around B's center). This can also be performed in $O(p^2)$ and is denoted with

`inner(M,`β_1`,...,`β_p`,`z_c`')=convert(outer(M,`α_1`,...,`α_p`,`z_c`),`z_c`')`

To further compress the notation, I'll use only the cell as identifier for an expansion, e.g. the inner expansion of cell A centered at z_c is written as:

`inner(A)`

8.2 The FMM algorithm: Treecodes with a twist

Now that we are done with the math, let's see how all these expansions help us to build a fast and accurate force calculation algorithm. The discussion is based on [Demmel2]

We begin by constructing a quadtree, as in section 0, with the exception that subdivison is not continued until there is 1 particle per cell, but less particles than some limit *s*. Then the quadtree is traversed from the leaves toward the root, computing outer expansions for each quadtree node. Next, the quadtree is traversed from the root to the leaves, computing inner expansions for the nodes (using the outer expansions provided in the previous step). Now, the force on the particles can be calculated by

adding the inner expansion for the node (accounts for the potential for all particles well separated from the node) and the direct sum of potentials from particles in nearby (non well-separated) nodes. A more detailed description follows:

Step 1: Building the quadtree

For simplicity, we well assume that the tree is fully populated, that is each leave is at the same distance from the root (this can be achieved by augmenting the tree with empty leaves). A version of FMM using adaptive trees can be found in [Carrier].

Step 2: Computing `outer()` for each node

Recall that we were able to move the center of expansion with the `outer_shift` function. Now we can move the center of expansion for the child cells $B_1..B_4$ to the center of parent cell A (z_a), and then simply add the coefficients α_j of the shifted expansions to obtain an outer expansion for cell A. The expansions of $B_1..B_4$ converge at distances larger than D from z_a, and thus the criterion of convergence for the outer expansion of A is satisfied.

We start from form the leaves, by calculating their outer expansions directly, and then proceed towards the root, "merging" outer expansions as described above. Note the similarity to the center of mass calculation step in the Barnes&Hut algorithm.

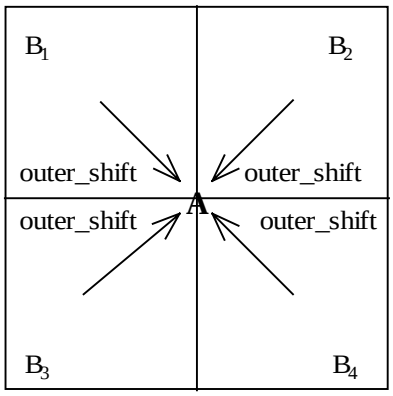

Figure 8 Step 2 of the FMM

Step 3: Computing `inner()` for each node

The idea is that we want to be able to calculate an expansion, which is valid inside a cell A, by using the outer expansions of other cells. Since the outer expansion of a cell B_i only converges (with the desired accuracy) at a distance ≥1.5D from the center of the cell, the immediate neighbour cells of A cannot be used to calculate the inner expansion. More formally, we define the interaction set I of A, in which our particle resides:

```
I(A):={nodes B_i such that B_i is a child of a neighbour of parent(A),
       but B_i is not itself a neighbour of A}
```

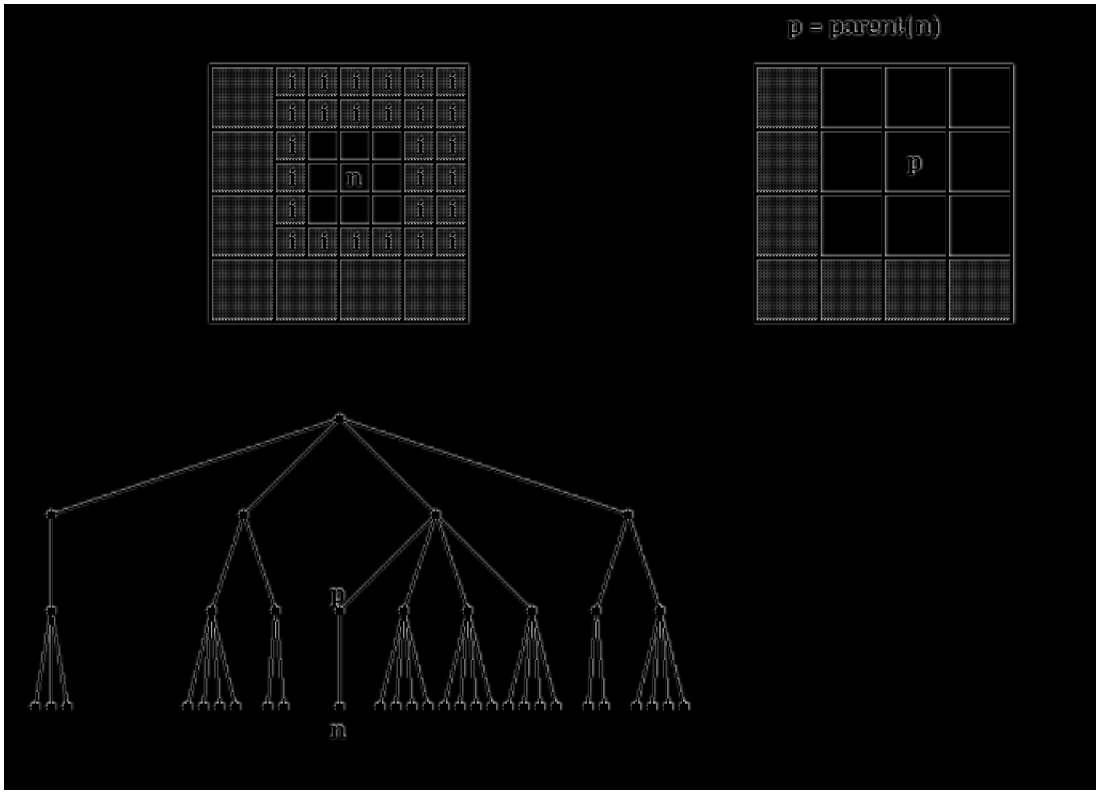

Figure 9 Interaction set for a cell A. From [Demmel2]

The interaction set of A is pictured in Figure 9.

In this stage, we start traversing the tree from the root. At each level, we have an inner expansion which is generated from the interaction set of the previous level (the root does not have any interaction set, and thus it's expansion from "previous levels" is empty). Now, by converting the outer expansions of the interaction set at the current level to inner expansions for A, summing them up, and finally adding the shifted inner expansion of A's parent, we have obtained an inner expansion for A that includes the potential of all cells except the neighbours of A. If A is itself a parent node, we then recurse once more. More formally:

```
Build_inner(A)
  P:=parent(A)
  inner(A):=EMPTY
  for all B_i∈I(A) do
    inner(A):=inner(A)+convert(outer(B_i),A)
  endfor
  inner(A):=inner(A)+inner_shift(inner(P),A)
  for all C:=children of A do
    build_inner(C)
  endfor
end
```

Note the difference from Barnes-Hut: This step does not calculate the force on a single particle, rather the potential of an entire leaf cell.

Step 4: Adding nearest neighbour contributions

For all leaf nodes we calculate the contribution to the potential by direct summation over the particles in the neighbouring cells. The number of particles is limited to *9s*, as we have limited the number of particles in a cell to *s*.

Step 5: Calculating forces[1]

Force calculation is now easy, and requires $O(1)$ time per particle. We iterate over the particles, taking the gradient of the inner expansion of the particle's cell (i.e. the local potential function) at the position of the particle. Since the potential is represented as a power series, taking the gradient analytically is possible.

8.3 FMM usage and efficiency.

The FMM is considered to be faster than Barnes-Hut for a given accuracy; this is due to the more accurate expansion of the potential. Also, the FMM can easily be set up in such a way as to guarantee a given accuracy.[25]

The FMM is usually referred to as an $O(n)$ algorithm[26]. The derivation relies heavily on the assumption that the quadtree would not grow higher as N increases (supposedly limited by the precision p). In effect, this means that the number of particles in neighbouring leaf cells must increase by $O(n)$, contradicting the argument of a constant s, and $O(1)$ time for PP potential calculation. Apparently I am not the only one disputing the way of deriving the complexity, see [Srinivas].

FMM compares favourably with Barnes-Hut in practical applications[27].

9. Other methods

There are a lot more methods used than those presented here. Most of them use some kind of hybrid scheme, or is a refinement of the methods presented in this paper; the web page of [Amara] is an excellent starting point. There are, however, also some methods based on radically different approaches; [Amara] lists the Self-Consistent Field (SCF) method and the Symplectic method.

10. Conclusions

We have seen that a great deal can be done to speed up the cumbersome summing of pairwise interactions in the general N-body problem. Algorithms exist that are, if not $O(n)$ in it's strictest sense, at least $O(n \log n)$, which is a tremendous improvement. PM and P^3M are more suited for large scale phenomena and collisionless systems, whereas Barnes-Hut and FMM are better when detail is important and and/or collisions are possible. FMM is nearly always preferable to Barnes-Hut; since the accuracy can easily be specified and it is faster than Barnes-Hut for a given accuracy.

---

[1] This step is not presented in [Demmel2]

11. References

[Amara] Amara Grap's excellent web page on N-body algorithms, internet, http://www.amara.com/papers/nbody.html

[Barnes&Hut] J. Barnes and P.Hut, A hierarchical O(N log N) force-calculation algorithm, Nature, 1986, 324, pages 446-449

[Carrier] J. Carrier and L. Greengard and V. Rokhlin, A fast adaptive multipole algorithm for particle simulations, SIAM J Sci Stat Comput, 1988, Vol 9, pages 669-686

[Demmel1] Lecure notes by prof James Demmel, internet http://www.cs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

[Demmel2] ibid, internet http://www.cs.berkeley.edu/~demmel/cs267/lecture27/lecture27.html

[Greengard] L.Greengard, Fast algorithms for classical physics, Science, 1994, Vol 265, pages 909-914

[Hockney] R.W. Hockney and J.W. Eastwood, Computer simulation using particles, McGraw-Hill, 1988

[Lecture15] Lecture notes by Tajh Taylor and Guy Blelloch, internet http://www.cs.cmu.edu/~guyb/real-world/nbody/index.html

[Lemmens] Master thesis of Kees Lemmens, internet http://ta.twi.tudelft.nl/DV/Staff/Lemmens/MThesis.TTH/chapter4.html

[Srinivas] Srinivas Aluru, "Greengard's N-Body Algorithm Is Not Order N," SIAM Journal on Scientific Computing, May 1996, Volume 17, Number 3.

[XStar] "The XStar N-Body Solver - Theory of operation" by Wayne Schlitt, internet, http://incolor.inetnebr.com/wayne/nbody/

---

[1] XStar pp. 30

[2] Amara

[3] Amara

[4] Greengard

[5] Greengard

[6] Demmel1

[7] Demmel1

[8] Amara

[9] Amara

[10] Barnes&Hut

[11] Barnes&Hut

[12] Lemmens

[13] Lemmens

[14] Hockney pp 120

[15] Hockney

[16] Lemmens

[17] Hockney pp 127

[18] Amara

[19] Lemmens

[20] Hockney pp 267

[21] Amara

[22] Demmel2

[23] Greengard

[24] Greengard

[25] Amara

[26] Greengard

[27] Lecture15