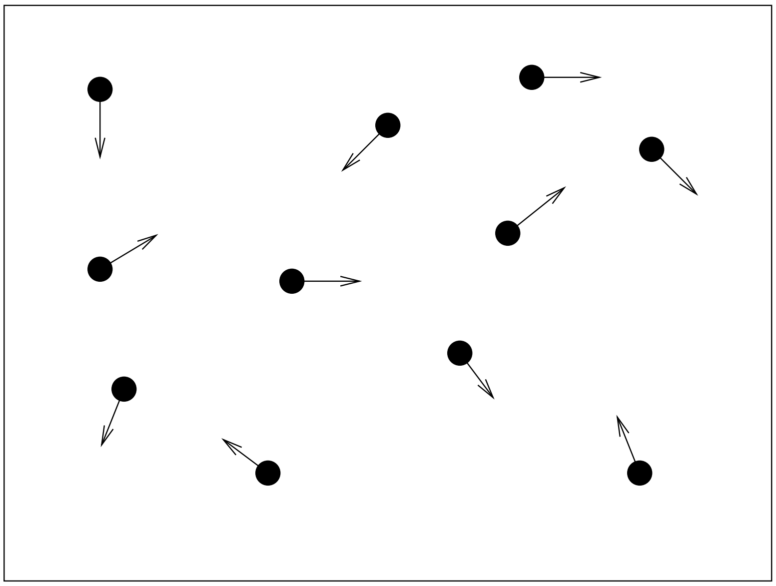
Notes by Tajh Taylor

- Delaunay Triangulation - more applications
- N-Body Problems
  1. Introduction
     - Basic Assumptions
     - Applications
     - History of Algorithms
  2. Mesh-based Algorithms
     - Particle mesh (using FFT)
  3. Tree-based Algorithms
     - Barnes-Hut Algorithm
     - Fast Multipole Method

1 Delaunay Triangulation - more applications

- Studying structure of galaxies
  use Euclidean minimum spanning tree to study clustering characteristics
- Prediction of protein folding
  run statistics on triangulated protein structures
- Studies of structural damage in metals
  e.g. study growth of "voids"
- Animal and plant ecology
- Human space patterns
  "central space" theory

2 N-Body Introduction



Particles in motion

The goal of N-Body problems is to determine the motion over time of bodies (particles) due to forces from other bodies. Typical calculated forces include electrostatic force and gravity. The basic method used for solving the problem is to loop forever, stepping discretely through time and doing the following at each timestep:

- Update positions using velocities ($\vec{x}_{i+1} = \vec{x}_i + \Delta t \vec{v}_i$)
- **Calculate forces** $\vec{F}$
- Update velocities ($\vec{v}_{i+1} = \vec{v}_i + \frac{1}{m} \Delta t \vec{F}_i$)

Our focus will be on methods for calculating the forces. Note that it is possible to use multiple resolutions for time, i.e. different Δt for different particles, but we will only consider uniform Δt.

2.1 Basic Assumptions

We will consider the particles to be in three-dimensional space, and we will use the following definitions:

$$
\begin{array}{llcll}
\text{Potential} & \phi(\vec{x}) & \propto & \frac{1}{r} \text{ (for 1 body)} & \\
\text{Coulomb Force} & \vec{F} & = & m_2 \nabla \phi(\vec{x}) & \propto \frac{1}{r^2}
\end{array}
$$

In general the potential $\phi(\vec{x})$ is a continuous function over space, and if it is known everywhere, we can calculate forces on all the bodies by taking the gradient.

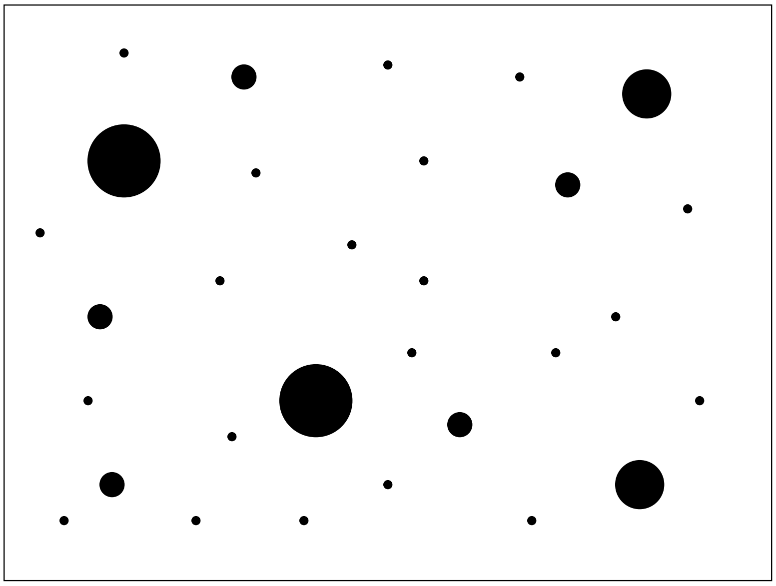


Figure 84: An area over which the potential $\phi(\vec{x})$ is defined. The balls represent different sized particles.

2.2 Applications

Astronomy : Formation of galaxies

- Questions:
  Is the universe expanding?
  How much "dark matter" is out there?
- Forces are gravitational
- State of the art simulations:
  $\approx 10^7$ bodies running for 10^3 steps.
  1 week on 500 processors (1994).

Biology/Chemistry: Molecular Dynamics

- Electrostatic + other forces
  (Bond forces, Lennard-Jones potential)
- State of the art (protein folding):
  10^6 bodies for 10^4 timesteps
- Note that it is easier to use a cutoff radius for limiting electrostatic calculations, because distant charges cancel each other out.

Others

- Plasma Physics

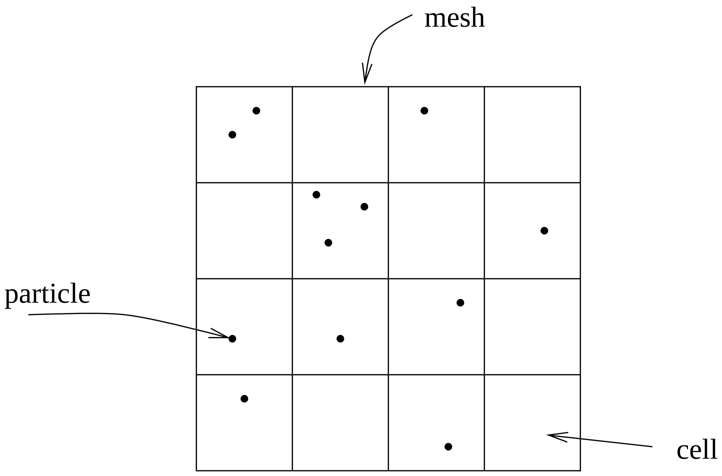


Figure 85: Mesh-based N-body methods

- Partial Differential Equations
- Fluid Dynamics – "vortex particles"

2.3 History of Algorithms

2.3.1 Particle-Particle (the naive method)

This method simply calculates the force between all pairs of particles.

- The time required to calculate all interactions is $O(n^2)$.
- This is inefficient for systems with 10^6+ bodies.

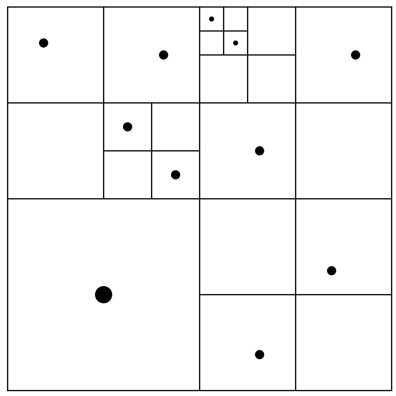
2.3.2 Mesh Based Methods

Particle Mesh (PM): Breaks space into a uniform mesh and places each particle in the appropriate cell of the mesh. Uses a discrete Fourier transform (DFT) to solve a partial differential equation (PDE) over the mesh.

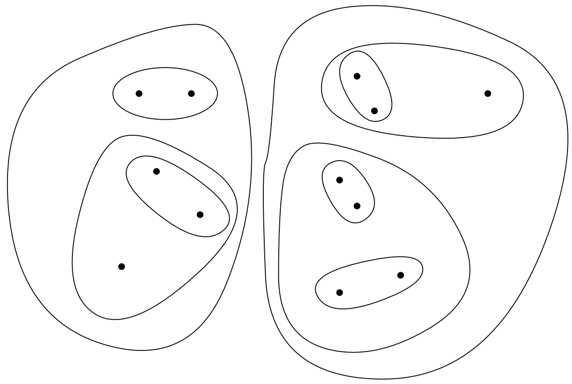
- Invented in the 1970s.
- For m cells, $O(m \log m)$ time is required.

Particle-Particle Particle-Mesh (P3M): A combination of direct particle-particle interactions for close particles and the particle-mesh method for distant particles.

- Invented in the early 1980s.
- Provides better accuracy for the same time as PM.



(a) A quadtree in a Top-down method



(b) Nearest neighbor groupings in a bottom-up method

Figure 86: Tree-based N-body methods

Nested-Grid Particle-Mesh (NGPM): Uses a nested mesh with different "resolutions".

- Invented around 1988.
- Is related to multigrid methods.

2.3.3 Tree Based Methods

Top-down particle-cell: Divides space into a quadtree (2 dimensional) or octree (3 dimensional). When a cell (inner node of the quad or octtree) is far enough away from a particle we can calculate the force to the center of mass of the cell instead of having to calculate the force to each particle in the cell. For example, in Figure 86 (a) the particle in the upper left could group the two particles in the lower right together and calculate the force to their center-of-mass.

- Appel (1985) produced the first tree-based solution.
- Barnes-Hut refined it in 1986.
- For n particles, $O(n \log n)$ time is required (under certain assumptions).

Bottom-up particle-cell: Similar to the top-down method but the tree is created bottom-up by grouping neighboring particles or cells (see Figure 86 (b)).

- Press produced an algorithm in 1986.
- $O(n \log n)$ time is required (under certain assumptions).

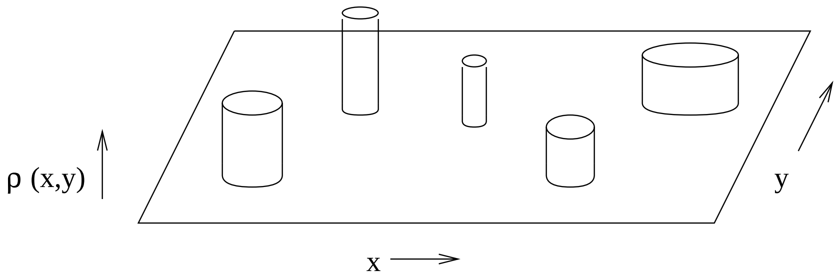


Figure 87: Visualization of the mass density function.

Cell-cell (Fast multipole method, FMM): Also uses a tree but allows for "cell-cell interactions" as well as "particle-cell interactions".

- Incorporates four main ideas: multipole expansions, error bounds on dropping terms, dual expansion, and translation of expansions.
- These are the fastest known methods for solving N-Body problems.
- Done by Greengard in 1988.
- $O(n)$ time is required for uniform distributions. Non-uniform distributions may increase the required time.

Tree-code particle-mesh: Mixes ideas from particle-mesh methods and tree-based methods.

- Work by Xi in 1989.

3 Mesh-based Algorithms

The goal is to solve the following PDE (Poisson's equation)

$$\nabla^2\phi(\vec{x}) = 4\pi G\rho(\vec{x})$$

where $\rho(\vec{x})$ is the "mass density" (Figure 87 shows an example). The solution to this equation gives the potential over space, which can then be used to calculate the force on each particle. The Poisson equation is the same as used for heat flow, or fluid flow without compression or viscosity. Poisson's equation can be solved by various methods, including spectral methods (DFT), finite element, or finite volume methods. Here we will discuss using spectral methods and the FFT, an approach that has been used extensively in astronomical n-body problem.

Particle mesh (using FFT)

We are given the density function $\rho(\vec{x})$ and want to solve for the potential $\phi(\vec{x})$ in the equation $\nabla^2\phi(\vec{x}) = 4\pi G\rho(\vec{x})$. We first discretize space by breaking it into a uniform mesh. Solving for the potential on this mesh will give us an approximate answer to the continuous solution and as long as the mesh is “fine enough” the approximation will be reasonably good. We will discuss how fine it has to be later.

For simplicity let's start by assuming 1-dimensional space, and let's consider the Discrete Fourier transform of the density function $\rho(x)$ (assuming x goes from 0 to $n-1$ in discrete steps of 1). The DFT is

$$\theta_j = \sum_{x=0}^{n-1} \rho(x) e^{2\pi i x j/n}$$

which transforms our density function into a set of frequency components (i is the imaginary number $\sqrt{-1}$). Since Poisson's equation is linear we can solve for each frequency component separately. Lets use the notation $w_j = 2\pi j/n$ and consider the j^{th} frequency component of the density function: $f_j(x) = \theta_j e^{i w_j x}$. If we plug this component into Poisson's equation, we get

$$\nabla^2 \phi_j(x) = 4\pi G \theta_j e^{i w_j x}$$

The solution to this equation is

$$\phi_j(x) = -4\pi G \theta_j \frac{1}{w_j^2} e^{i w_j x} = -4\pi G \frac{1}{w_j^2} f_j(x)$$

since $\nabla^2 e^{i w_j x} = -w_j^2 e^{i w_j x}$. In other words, to solve for the j_{th} frequency component of the potential ϕ we simply multiply the j_{th} frequency component of the density function by $-4\pi G \frac{1}{w_j^2}$.

We can similarly solve for each of the other frequency components of the potential, then add up the solutions to get the solution to our initial problem. This is equivalent to convolving the frequency component j with $1/w_j^2$ and then transforming back to coordinate space, which can be done with a second DFT. This idea can be applied in multiple dimensions by taking a multidimensional DFT. In this case the solution for a particular component with frequencies w_j, w_k and w_l would be

$$\phi_{jkl}(\vec{x}) = -4\pi G \frac{1}{w_j^2 + w_k^2 + w_l^2} f_{jkl}(\vec{x})$$

where j, k and l are the frequency components in the x, y and z dimensions respectively.

So the overall algorithm in three dimensions has five steps:

- Break space into discrete uniform mesh of points.
- Assign each particle to a point (or set of points) to generate ρ for each point.
- Do a 3-D FFT on the mesh.
- Convolve the results with the sum of inverse square frequencies.
- Do an inverse FFT back to the original mesh points.

This will give $\phi(\vec{x})$ over the mesh from which we can derive the forces. For m mesh points the algorithm will take $O(m \log m)$ time.

The problems with this method is that in order to get reasonable accuracy there should be no more than one particle per cell, and ideally there should be a few cells separating each particle. This works fine if the particles are uniformly distributed, but if the particles are nonuniformly distributed the simulation would require a number of cells that is very much larger than the number of particle. This is because the mesh has to fine enough to accommodate the smallest distance between particles. For sufficiently nonuniform points the number of mesh points required could become greater than n^2 and therefore run even slower than the naive all-pair interactions.

4 Tree-based Algorithms

The approach taken by tree based methods is to approximate far away particles by some sort of center of mass rather than by direct interactions.[1] The further away you are from a set of points, the more you can group together and consider as a center-of-mass. These "centers of mass" are then organized hierarchically (in a tree) so that the approaches can determine which sets of points to approximate by their centers of mass. The root of the tree will include all the particles, and at each node of the tree the children will partition the set of particles of that node, usually by partitioning the space taken by that node. The closer you are to the points, the further down in the tree you will need to go so that you will consider smaller sets of particles.

We will call a set of particles grouped together with a center of mass, a *cell*. In the algorithms we will consider three types of interactions, particle-particle (direct interactions between two particles, particle-cell (interaction between a particle and the center of mass of a cell), and cell-cell (interactions between two centers-of-mass). These are shown in Figure 88.

In this class we will only consider top-down tree-based algorithms.

4.1 Barnes-Hut Algorithm

The first tree-based algorithm we consider is the Barnes-Hut algorithm, which use particle-particle and particle-cell interactions, but no cell-cell interactions. It consists of building a tree using an oct-tree in 3 dimensions or a quad-tree in 2 dimensions, and then using the tree to calculate the force on each particle. Each division evenly splits up the space so, for example, in a quadtree each step will consists of dividing a square into 4 squares. Note that this will not necessarily evenly divide the particles, so the tree in general will not be balanced. The algorithm for building the tree works top down and is defined by the following code

[1] Center of mass is used informally here since later we will show that some of the algorithms approximate a set of particles by more complicated functions.

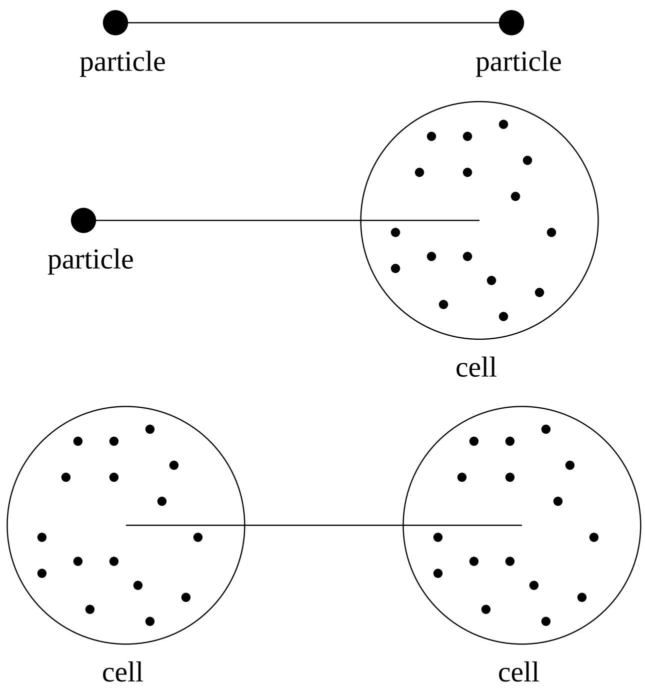


Figure 88: Particle-particle, particle-cell, and cell-cell interactions.

function Build_Tree(P)
  // P is a set of points along with a bounding box
  if Particle_Count(P) $\leq \tau$ // where τ is an integer threshold ($\tau \geq 1$)
    return P
  else
    $(P_1, P_2, P_3, \ldots)$ = partition(P)   // the quadtree or octree division (divides space evenly)
    a = Make_Empty_Tree()
    for $\pi \in (P_1, P_2, P_3, \ldots)$
      a = Add_Child(a, build_tree(π))
    return a

Figure 89 gives an example of a set of points in 2-dimensions along with the tree built by this code. Once the tree has been built the force algorithm is quite simple and is given in Figure 90. The **well_separated** function tells when to approximate the force for a group of particles using the center of mass for the group (see Figure 91). θ is a user-specified parameter that controls the accuracy of the results. A smaller θ will increase accuracy but also increase runtime, and larger θ will decrease accuracy and runtime.

For "reasonable" distributions each particle interacts with a constant number of cells on each level of the tree, and there are $O(\log n)$ levels. Therefore each particle takes $O(\log n)$ time and the total time for the Barnes-Hut algorithm is $O(n \log n)$. One can show, for example, that for uniform distributions in 3-d:

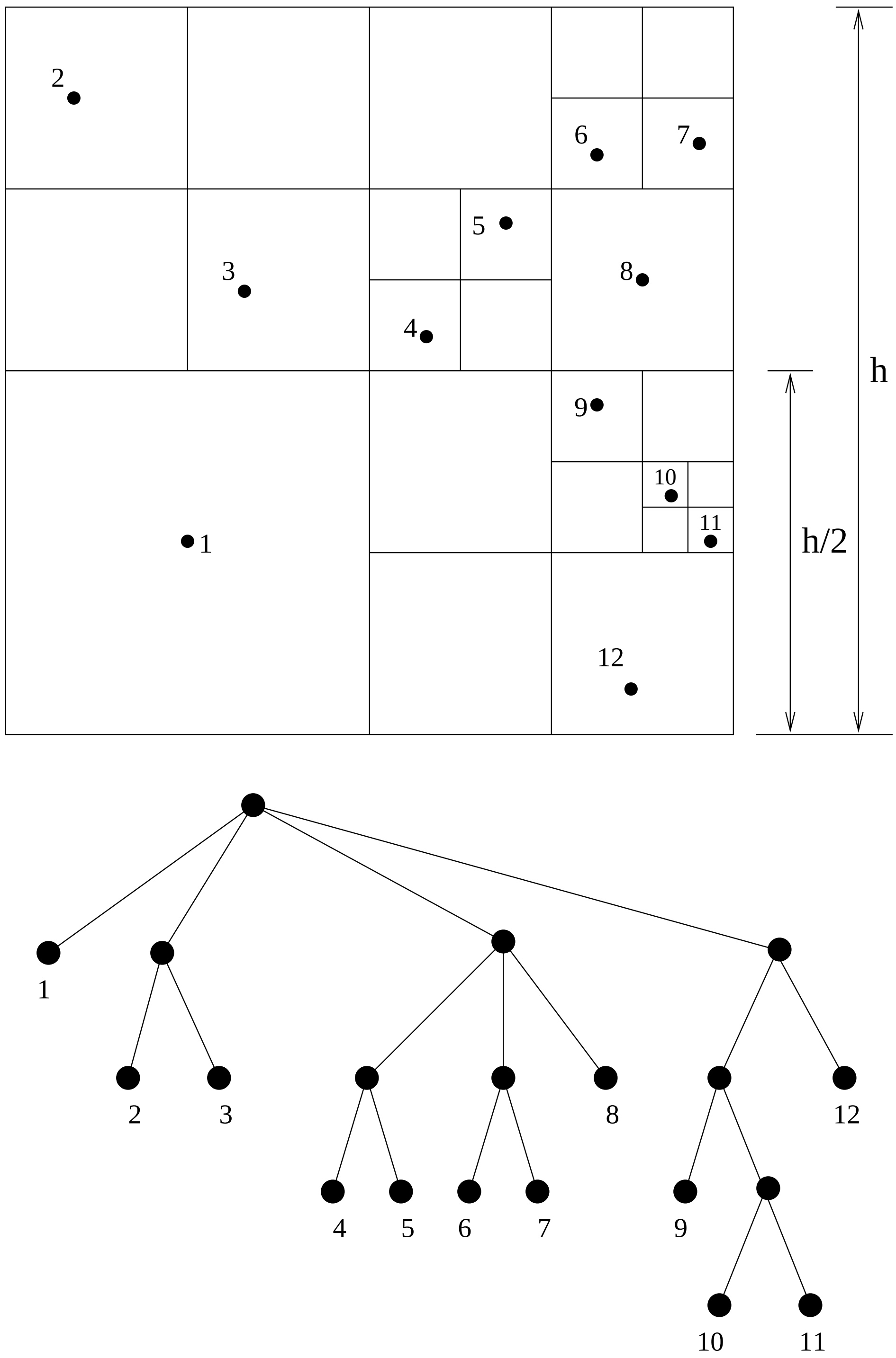


Figure 89: An example of a quadtree partition of space and the corresponding tree used in the Barnes Hut algorithm (assumes $\tau = 1$).

```
function Barnes_Hut(P)
    T = Build_Tree(P)
    r = root of T
    for p ∈ P
        F⃗_p = Calculate_Force(p, r)

function Calculate_Force(p, c)
    // p is a particle
    // c is a node of the tree, which is either a particle or cell
    if Is_Particle(c)
        if p = c return 0              // don't include the point itself
        else return Force(p, c)    // particle-particle force calculation
    if Well_Separated(p, c)
        return Force(p, Centermass(c))    // particle-cell force calculation
    F⃗ = 0
    for c' ∈ children of c
        F⃗ += Calculate_Force(p, c')
    return F⃗

function Well_Separated(p, c)
    return s(c) < θ · d(p, Center(c))
```

Figure 90: Code for the Barnes-Hut algorithm. Assumes the leaves of the tree are individual particles, *i.e.*, $\tau = 1$

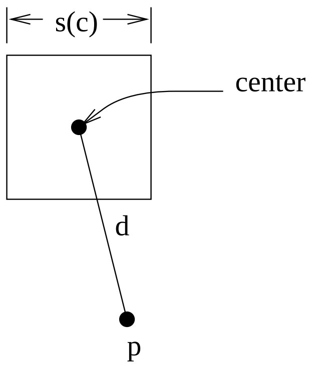


Figure 91: Well_Separated calculation on a cell c and particle p.

$$\text{Total Interactions: } I(n, \theta) \approx \frac{28\pi}{3\theta^3} n \log_8(n)$$

For $\theta = .3$, which is typically required for good accuracy, this gives $\approx 350n \lg n$. Using a very rough calculation (we are dropping some lower order terms) this outperforms calculating all pairs interactions when $350n \lg n \leq n^2/2$, which is when $n > 10000$. At $\theta = .5$, which is the about the highest used, the tradeoff is at around 2000 particles. Barnes-Hut is therefore not useful for a small number of particles. It, however, can do much better than the all-pairs algorithm for larger n.

Exercise: For a million particles and $\theta = .3$ about how much faster is Barnes-Hut over using all pairs?

The Barnes-Hut algorithm is highly parallelizable since the loop over the particles once the tree is built is completely parallel, and it is not too hard to build the tree in parallel. There have been many parallel implementations of the Barnes-Hut algorithm.

4.2 Fast Multipole Method

This method was pioneered by Greengard and Rokhlyn in 1987, and it won an ACM Thesis Award for Greengard. It was the first linear-time algorithm for N-Body problems (for uniform or near uniform points). They were also the first to have proven error bounds.

The method has four main ideas in it, each of which will be covered in more detail:

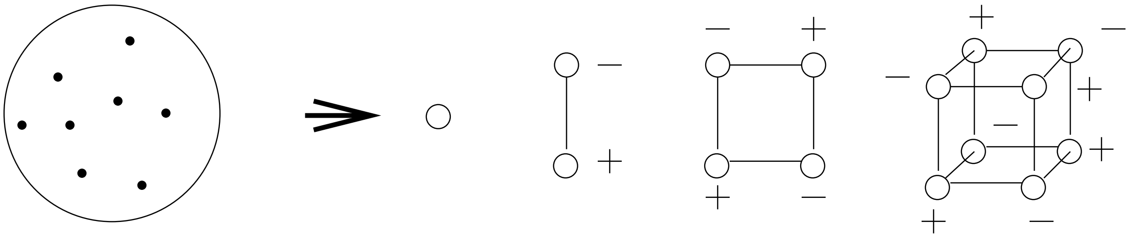
- Multipole expansions
- Error bounds on dropping terms
- Dual local expansion
- Translation of expansions

For nonuniform points, FMM can take $O(n \log n)$ time to find a set of "well-separated" points, but the force calculation is still $O(n)$.

Idea 1: Multipole expansions

For any cluster of particles, their net force can be approximated using a multipole expansion (see Figure 92) which is taken around a fixed point. The first term of the expansion is often called the monopole term, the next group of 3 terms are called the dipole terms, and the next group of 5 independent terms are called the quadrapole terms.

The multipole expansion is analogous to a Taylor-series expansion, but in 3d. Like the Taylor-series expansion the earlier terms have a larger magnitude when far enough from the origin, and the function can be approximated by dropping higher order terms. In the case of multipole-expansions the force and potential due to higher-order terms drops off faster as a function of r (see the last two rows Figure 92), which is what allows us to drop these terms far from the center.

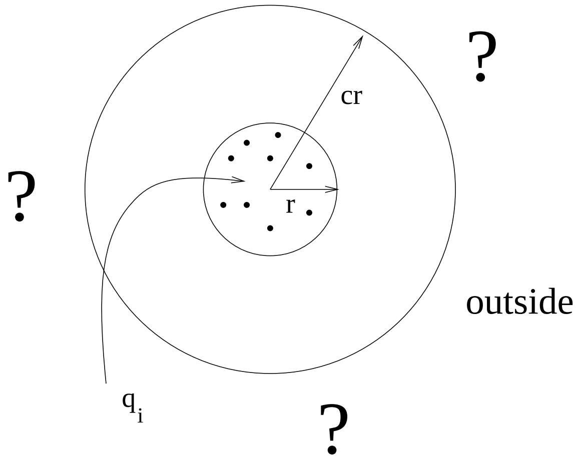


p	1	2	3	4
Terms	1	3	5	7
Potential	$\frac{1}{r}$	$\frac{1}{r^2}$	$\frac{1}{r^3}$	$\frac{1}{r^4}$
Force	$\frac{1}{r^2}$	$\frac{1}{r^3}$	$\frac{1}{r^4}$	$\frac{1}{r^5}$

Figure 92: Multipole expansion.

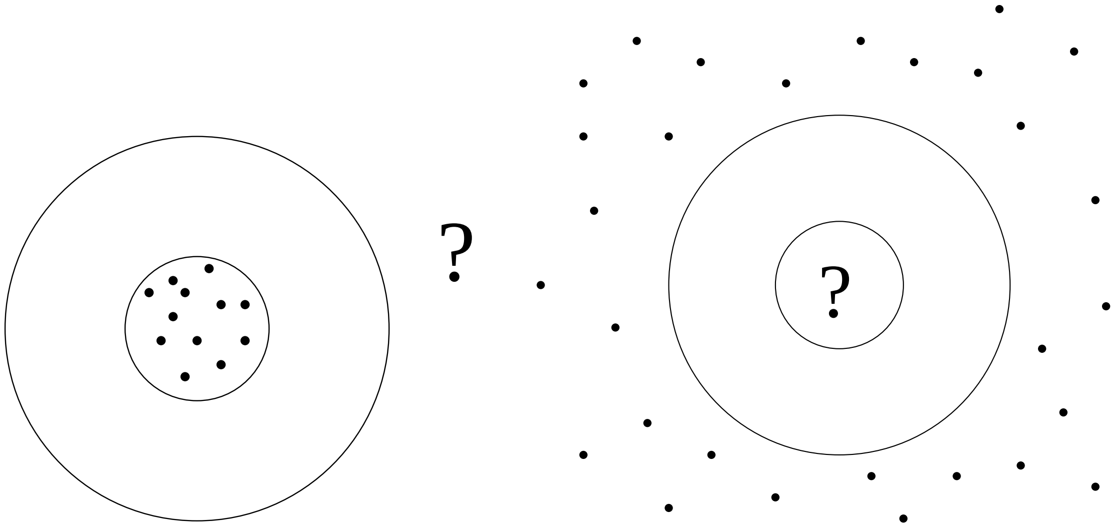
Idea 2: Error Bounds

For distributions with only positive sources (*e.g.*, gravitation) we can bound the ratio of the force due to higher terms of the potential relative to the first term, and therefore bound the error due to dropping terms after a given p. This is possible at a sufficient distance from the particles since the higher terms diminish faster than the first term (see the third row in Figure 92). To see this consider the following diagram:



Lets define $M = \sum q_i$ as the sum of the masses inside the sphere defined by r. A lower bound on the force due to the monopole of these masses (first term of the expansion) on a mass m_1 on the outer sphere is

$$F_1 \geq \frac{m_1 M}{(((c+1)r)^2)}$$



(a) multipole expansion approximation (b) local expansion approximation

Figure 93: Multipole (outer) and local (inner) expansions.

Similarly an upper bound on the force due to the other multipole terms on m_1 is

$$F_p \leq \frac{r^{p-1} m_1 M}{((c-1)r)^{p+1}} \,.$$

With these bounds we can then bound the ratio of the contribution of the p^{th} term to the first term by

$$E_p = \frac{F_p}{F_1} \leq \frac{(c+1)^2}{(c-1)^{p+1}}$$

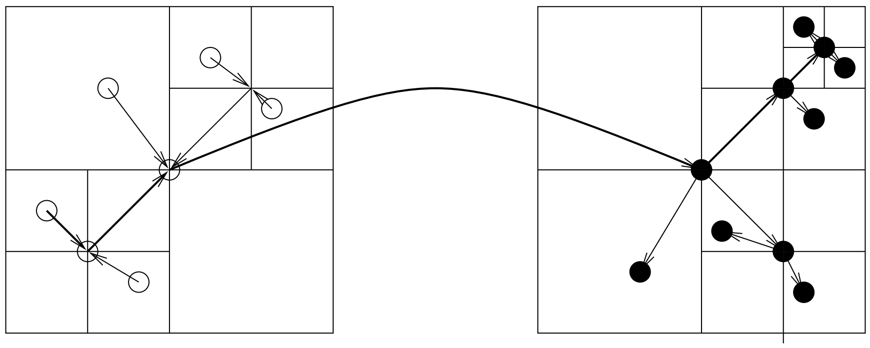
Assuming c is greater than 2, this bound decreases exponentially with p and allows us to bound the number of terms p we need for a given accuracy (we can actually get tighter bounds if we are more careful). It also shows how having larger c gives better error bounds. As we go outside the outer sphere our approximation can only get better.

Idea 3: Dual Expansion

Our previous multipole expansion assumed we know the sources inside an inner sphere and are trying to approximate the potential (or forces) due to them outside an outer sphere (see Figure 93(a)). There is also a dual expansion in which the sources are known outside the outer sphere and we want an approximation the potential contribution due to them inside the inner sphere (see Figure 93(b)). These dual expansions turn out to be very important in the Fast Multipole Method. We will refer to them as the *local* or *inner* expansion. We will refer to our original expansion as the *multipole* or *outer* expansion.

Idea 4: Translation of Expansions

The final idea is that the expansions can be translated from one center to another. Furthermore we can translate between the two different kinds of expansions. This is illustrated in



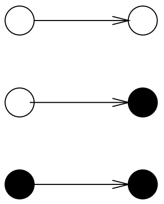


Figure 94: Example translations

Figure 94. In this figure we are trying to approximate the forces due to the particles in the left box on the particles in the right box. We first combine the outer (multipole) expansions on the left in a tree like fashion by translating each expansion to its parent in the tree and summing the children into the parent. We then translate the total outer expansion to an inner (local) expansion in the right box. Finally we translate this inner expansion down the tree to each of the locations on the right. The running time for each translation is dependent on the number of terms. With a naive implementation each translation can run in $O(p^4)$ time but this can be reduced to $O(p^2 \log p^2)$ with a FFT-style translation.

The translation of the outer expansion to an inner expansion in another box is called a cell-cell interaction. This is because we are effectively having the cells interact rather than having particles interact directly (note that in the example of Figure 94 no particle on the right interacts directly with a particle on the left).

Why cell-cell interactions?

To motivate why cell-cell interactions and both forms of expansions are important consider Figure 95. In the example the goal is to determine the potential (or force) on each particle due to all other particles outside of its own cell. To do this we will use expansions for each cell. We first consider the cost (number of interactions) when using just particle-cell interactions and then show that we can do much better by using cell-cell interactions.

Particle-Cell Cost

- $\sqrt{n} \times \sqrt{n}$ to calculate multipole expansions within each cell each cell
- $n \times \sqrt{n}$ for particle-cell interactions (each particle has to look at the other $\sqrt{n}$ cells.

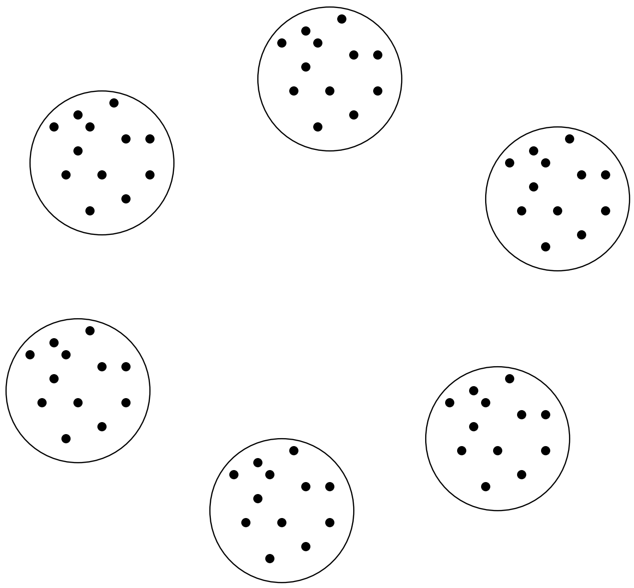


Figure 95: Example cell-cell interactions. Assume there are n total particles with $\sqrt{n}$ particles within each cell (circle) and that the cells are well separated from each other.

- Total $= O(n^{3/2})$

Cell-Cell Cost

- $\sqrt{n} \times \sqrt{n}$ to calculate multipole expansions within each cell
- $\sqrt{n} \times \sqrt{n}$ for outer-inner (multipole-local) translations — every cell translates to every other cell and we add up the contribution at each cell
- $\sqrt{n} \times \sqrt{n}$ to translate the local expansion from the center-of-mass of each cell to each of its $\sqrt{n}$ elements.
- Total $= O(n)$

FMM: Using a tree

We now consider putting the ideas together into the Fast Multiple Algorithm. There are three phases, each which uses a tree.

- Bottom up: forming multipole expansions
  - outer-outer translation from children to parents
- Top down: forming local expansions
  - inner-inner translation from parent to child
  - outer-inner translation from interaction list

- Direct interactions with neighbors at leafs

For a parameter k and for cells on the same level (i.e. of the same resolution), we make the following definitions.

Definition: two cells are *near neighbors* of each other if they are within k cells of each other. Note that two cells with touching corners are considered to have a distance of one.

Definition: two cells are *well separated* if they are not near neighbors.

Definition: a cell c_2 is in c_1's *interaction list* if

1. well-separated(c_1, c_2) and
2. near-neighbor$(p(c_1), p(c_2))$

where $p(c)$ indicates the parent of c.

In practice $k = 2$, and in 3-dimensional space the size of the interaction lists is then 875. In general it is not hard to show that the size of an interaction list is $(2^d - 1)(2k + 1)$.

Total Time (for a uniform distribution)

Assume $k = 2$. Place p^2 particles per leaf cell (n/p^2 leaf cells).

Assuming translations take p^4 time:

$$\begin{array}{ll}
\text{Leaf multipoles} & p^4\left(\dfrac{n}{p^2}\right) \\
\text{Up Tree} & \displaystyle\sum_{l=1}^{d} 8^{l+1}p^4 \\
\text{Down Tree} & \displaystyle\sum_{l=1}^{d}(875 \cdot 8^l p^4 + 8^l p^4) \\
\text{Local expansion} & p^4\left(\dfrac{n}{p^2}\right) \\
\text{Total Time} & = O(np^2) \\
\text{Error Bound} & \approx 2.6^{-p}
\end{array}$$

There is a clear tradeoff between time and accuracy.

Figure 96 shows a comparison of runtimes for the various algorithms for two different accuracies. The BH (rect.) is a variant of Barnes-Hut that uses multipole expansions in rectilinear coordinates. The BH (spher.) is a a variant of Barnes-Hut that uses multipole expansions in spherical coordinates. The PMTA is an algorithms that combines the ideas of Barnes-Hut and FMM. Comparisons are given for sets of particles that all have the same charge (chargeless) and for particles which have both positive and negative charges (charged).

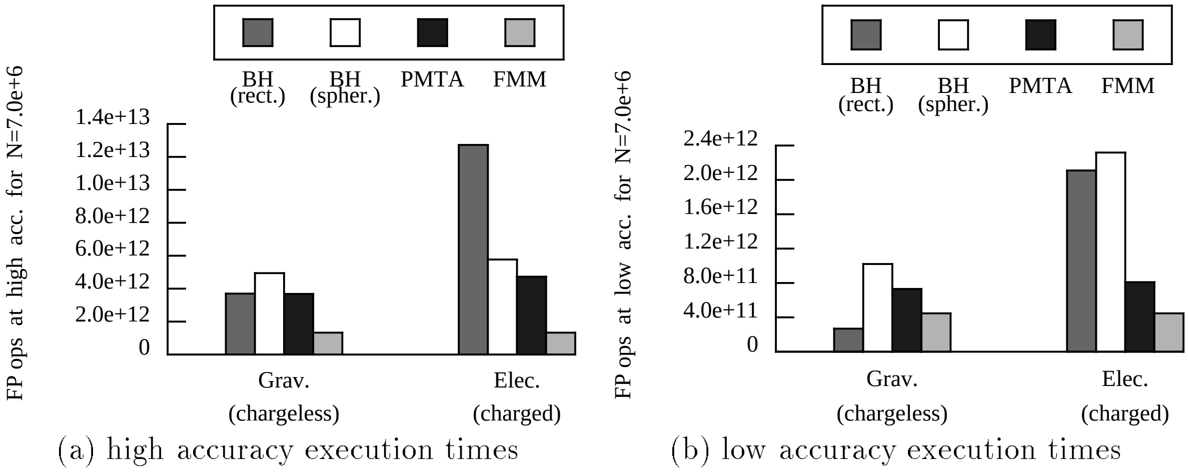


Figure 96: Runtimes for the various algorithms.