

CS601: Software Development for Scientific Computing

Autumn 2021

Week11:

Particle Methods, Tree-based codes

Course Progress..



- Last topic - Computation on unstructured grids
 - Finite Element Method (FEM)
- Coming Next
 - Particle (Simulation) Methods
 - Tree-based codes

Course Progress..

- Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)

<i>Motif</i>	Embed	Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser	<i>Motif</i>					Desktop	Games	DB	ML	HPC	Medicine	Music	Speech	CBIR	Browser									
	1 Finite State Mach.	Hot	Hot	Warm	Hot	Warm						Hot	9 N-Body	Hot			Hot											Hot							
2 Combinational	Hot			Warm	Warm						Hot	10 MapReduce	Warm			Hot	Hot	Hot						Warm	Hot		Warm								
3 Graph Traversal	Hot	Hot	Hot	Hot	Hot		Hot		Hot		Warm	11 Backtrack/B&B			Hot	Hot						Hot													
4 Structured Grid	Hot	Hot	Hot		Hot	Hot	Hot		Hot	Hot	Hot	12 Graphical Models			Hot	Hot						Hot													
5 Dense Matrix	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	13 Unstructured Grid		Hot	Hot	Hot	Hot																		
6 Sparse Matrix	Hot	Hot	Hot		Hot	Hot	Hot	Hot	Hot		Hot	<i>Temperature Chart of Need</i>					DB = database																		
7 Spectral (FFT)	Hot		Hot		Hot	Hot		Hot	Hot	Warm	Hot	Hot	Hot	Warm	Med	Cool	ML = machine learning																		
8 Dynamic Prog	Hot			Hot	Hot				Hot	Hot	Hot	Hot	Hot	Hot	Hot	Hot	HPC = High Perf. Comp.																		

Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

 Seen earlier
 Next..

Particle (Simulation) Methods

- N-Body Simulation – Problem

System of N-bodies (e.g. galaxies, stars, atoms, light rays etc.) interacting with each other continuously

- Problem:

- Compute force acting on a body due to all other bodies in the system
- Determine position, velocity, at various times for each body

- Objective:

- Determine the (approximate) evolution of a system of bodies interacting with each other simultaneously

Particle (Simulation) Methods

- N-Body Simulation - Examples

- Astrophysical simulation: E.g. each body is a star/galaxy

https://commons.wikimedia.org/w/index.php?title=File%3AGalaxy_collision.ogv

- Graphics: E.g. each body is a ray of light emanating from the light source.

<https://www.fxguide.com/fxfeatured/brave-new-hair/>



- Here each body is a point on a strand of hair

N-Body Simulation

- All-pairs Method
 - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
 - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
 - Examples:
 - Barnes-Hut
 - Fast Multipole Method

N-Body Simulation


- Three fundamental simulation approaches
 - Particle-Particle (PP)
 - Particle-Mesh (PM)
 - Particle-Particle-Particle-Mesh (P3M)
- Hybrid approaches
 - Nested Grid Particle Scheme
 - Tree Codes
 - Tree Code Particle Mesh (TPM)
- Self Consistent Field (SCF), Smoothed-Particle Hydrodynamics (SPH), Symplectic etc.

Particle-Particle method

- Simplest. Adopts an all-pairs approach.
- State of the system at time t given by particle positions $x_i(t)$ and velocity $v_i(t)$ for $i=1$ to N

$$\{x_i(t), v_i(t); i = 1, N\}$$

– Steps:

- 
1. Compute forces
 2. Integrate equations of motion
 3. Update time counter

Each iteration updates $x_i(t)$ and $v_i(t)$ to compute $x_i(t + \Delta t)$ and $v_i(t + \Delta t)$

Particle-Particle Method

1. Compute forces

```
//initialize forces
```

```
for i=1 to N
```

```
   $F_i = 0$ 
```

```
//Accumulate forces
```

```
for i=1 to N-1
```

```
  for j=i+1 to N
```

```
     $F_i = F_i + F_{ij}$  ←  $F_{ij}$  is the force on particle i due to particle j
```

```
     $F_j = F_j - F_{ij}$ 
```

Typically: $F_i = F_{\text{external}} + F_{\text{nearest_neighbor}} + F_{\text{N-Body}}$

Particle-Particle Method

2. Integrate equations of motion

for $i=1$ to N

$$v_i^{new} = v_i^{old} + \frac{F_i}{m_i} \Delta t \quad // \text{using } a=F/m \text{ and } v=u+at$$

$$x_i^{new} = x_i^{old} + v_i \Delta t$$

3. Update time counter

$$t^{new} = t^{old} + \Delta t$$

Particle-Particle Method

```
t=0
while(t<tfinal) {
//initialize forces
    for i=1 to N
        Fi = 0
//Accumulate forces
    for i=1 to N-1
        for j=i+1 to N
            F[i] = F[i] + Fij
            F[j] = F[j] - Fij
//Integrate equations of motion
    for i=1 to N
         $v_i^{new} = v_i^{old} + \frac{F_i}{m_i} \Delta t$  //using a=F/m and v=u+at
         $x_i^{new} = x_i^{old} + v_i \Delta t$ 
// Update time counter
        t = t + Δt
}
```

Particle-Particle Method

- Costs (CPU operations)?

```
t=0
while(t<tfinal) {
  //initialize forces
  for i=1 to N
    Fi = 0
  //Accumulate forces
  for i=1 to N-1
    for j=i+1 to N
      F[i] = F[i] + Fij
      F[j] = F[j] - Fij
  //Integrate equations of motion
  for i=1 to N
    vinew = viold +  $\frac{F_i}{m_i} \Delta t$  //using a=F/m and v=u+at
    xinew = xiold + vi Δt
  // Update time counter
  t = t + Δt
}
```

Particle-Particle Method

- Experimental results (then):
 - Intel Delta = 1992 supercomputer, 512 Intel i860s
 - 17 million particles, 600 time steps, 24 hours elapsed time
 - M. Warren and J. Salmon
 - *Gordon Bell Prize at Supercomputing 1992*
 - Sustained 5.2 Gigaflops = 44K Flops/particle/time step
 - 1% accuracy
 - *Direct method (17 Flops/particle/time step) at 5.2 Gflops would have taken 18 years, 6570 times longer*

Particle-Particle Method

- Experimental results (now):

Vortex particle simulation of turbulence

- Cluster of 256 NVIDIA GeForce 8800 GPUs

- 16.8 million particles

- T. Hamada, R. Yokota, K. Nitadori, T. Narumi, K. Yasuki et al

- *Gordon Bell Prize for Price/Performance at Supercomputing 2009*

- Sustained 20 Teraflops, or \$8/Gigaflop

Particle-Particle (PP) Method

- Discussion
 - Simple/trivial to program
 - High computational cost
 - Useful when number of particles are small (few thousands) and
 - We are interested in close-range dynamics when the particles in the range contribute *significantly* to forces
 - Constant time step must be replaced with variable time steps and numerical integration schemes for close-range interactions

Particle-Mesh (PM) Method

- Now think of Force as a *Field* i.e. a quantity that is pervading all space rather than arising out of (and concentrated at) the particle.

- **Steps** (e.g. body of electrically charged particles):

1. Assign *charge* to *mesh*

2. Solve the Field Potential equation on the mesh

$$\nabla^2 \phi = -\rho / \epsilon_0 \quad \longleftarrow \text{What kind of PDE is involved?}$$

3. Compute forces from the mesh-defined potential and interpolate forces at particle positions

4. Integrate forces to get particle positions and velocities

5. Update time counter

} Same as PP

Particle-Mesh (PM) Method

1. Assign *charge* to *mesh*

- Create a mesh of points for each particle. Assign “charge” to mesh points
 - Many schemes exist. "Nearest-Grid-Point" (NGP) PM method assigns charge densities to mesh points. The charge densities are computed as: total amount of "charge" in the cell surrounding the mesh point, divided by the cell volume.
 - Rarely used. Drawback: it gives forces that are discontinuous in value.
 - “Smoothness” is incorporated to ensure continuity of the derivatives.
- Computational Cost? Assume NGP.

Particle-Mesh (PM) Method

2. Solve the Field Potential equation on the mesh

$$\nabla^2 \phi = -\rho / \epsilon_0$$

ϕ is electrostatic potential and ρ is the charge density.

The Poisson's equation is solved using finite difference approximations.

- Cost of solving the equation depends on ???

Particle-Mesh (PM) Method

- Computational Costs?
 - Cost for steps 1, 3, and 4: $O(N)$
 - Cost for step 2: depends on number of mesh points
 - Cost for step 5: constant

Particle-Mesh (PM) Method

- Discussion

- PM is much faster than PP but less accurate

- Computation cost

$$\alpha N + \beta N_{meshpoints}$$

- Source of errors:

- 1) Replacing the charge with charge densities on the mesh

- 2) Truncation error due to finite differences

- 3) Interpolation error,

- 4) Integration error (when numerical integration schemes used)

- Unsuitable for “close-encounters”

Exercise

- Assume $\alpha=20$, $\beta = N^3 \log_2 N^3$ (for $N \times N \times N$ mesh)

Give an estimate of time taken by PP and PM methods for:

- Number of particles = 10^5
- Number of mesh points = 32
- $\approx 1\mu s$ per pair-wise force calculation

N-Body Simulation

- Three fundamental simulation approaches
 - Particle-Particle (PP)
 - Particle-Mesh (PM)
 - Particle-Particle-Particle-Mesh (P3M)
- Hybrid approaches
 - Nested Grid Particle Scheme
 - Tree Codes
 - Tree Code Particle Mesh (TPM)
- Self Consistent Field (SCF), Smoothed-Particle Hydrodynamics (SPH), Symplectic etc.

Particle-Particle-Particle-Mesh (P3M)

- Combine the advantages of PP and PM
 - PP:
 - Can be used with smaller systems with long-range forces and
 - Large systems with few near-neighbor/long-range forces
 - PM:
 - Fast but can only handle smoothly varying forces
- F_i is split into $F_i^{shortrange} + F_i^{mesh}$
 - $F_i^{shortrange}$, fast-varying force, calculation done using PP (assumption: $F_i^{shortrange}$ is non-zero for only a few particles)
 - F_i^{mesh} , slow-varying force, calculation done using PM (assumption: F_i^{mesh} is sufficiently smooth to be calculated over a mesh)

N-Body Simulation

- All-pairs Method
 - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
 - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
 - Examples:
 - Barnes-Hut
 - Fast Multipole Method

Tree Codes

$$F_i = F_{\text{external}} + F_{\text{nearest_neighbor}} + F_{\text{N-Body}}$$

- F_{external} can be computed for each body independently. $O(N)$
- $F_{\text{nearest_neighbor}}$ involve computations corresponding to few nearest neighbors. $O(N)$
- $F_{\text{N-Body}}$ require all-to-all computations. Most expensive. $O(N^2)$ if computed using all-pairs approach.

for(i = 1 to N)

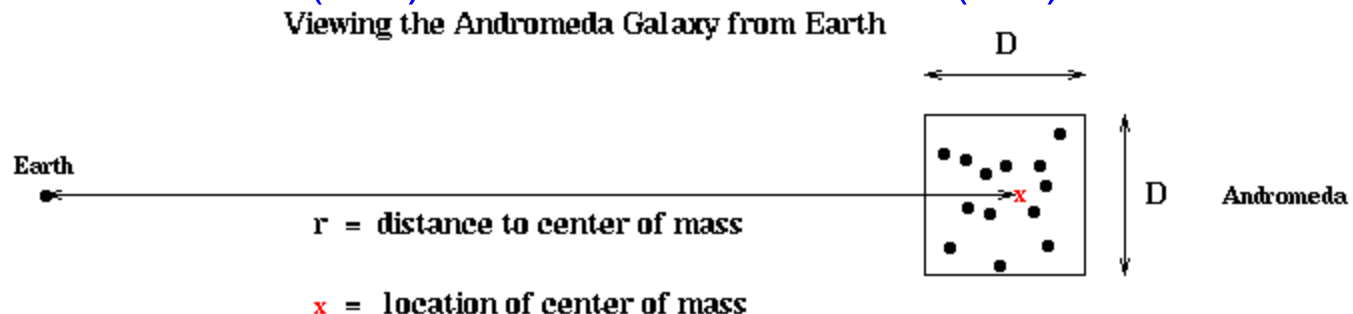
$$F_i = \sum_{i \neq j} F_{ij} \quad F_{ij} = \text{force on } i \text{ from } j$$

$$F_{ij} = c \cdot v / \|v\|^3 \text{ in 3D, } F_{ij} = c \cdot v / \|v\|^2 \text{ in 2D}$$

v = vector from particle i to particle j , $\|v\|$ = length of v , c = product of masses or charges

Tree Codes: Divide-Conquer Approach

- Consider computing force on earth due to all celestial bodies
 - Look at the night sky. Number of terms in $\sum_{i \neq j} F_{ij}$ is greater than the number of visible stars
 - One “star” could really be the Andromeda galaxy, which contains billions of real stars. *Seems like a lot more work than we thought ...*
 - Idea: Ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass (TM)



- Require that D/r be “small enough” (D = size of box containing Andromeda, r = distance of CM to Earth).

Idea is not new. Newton approximated earth and falling apple by CM

Tree Codes: Divide-Conquer Approach

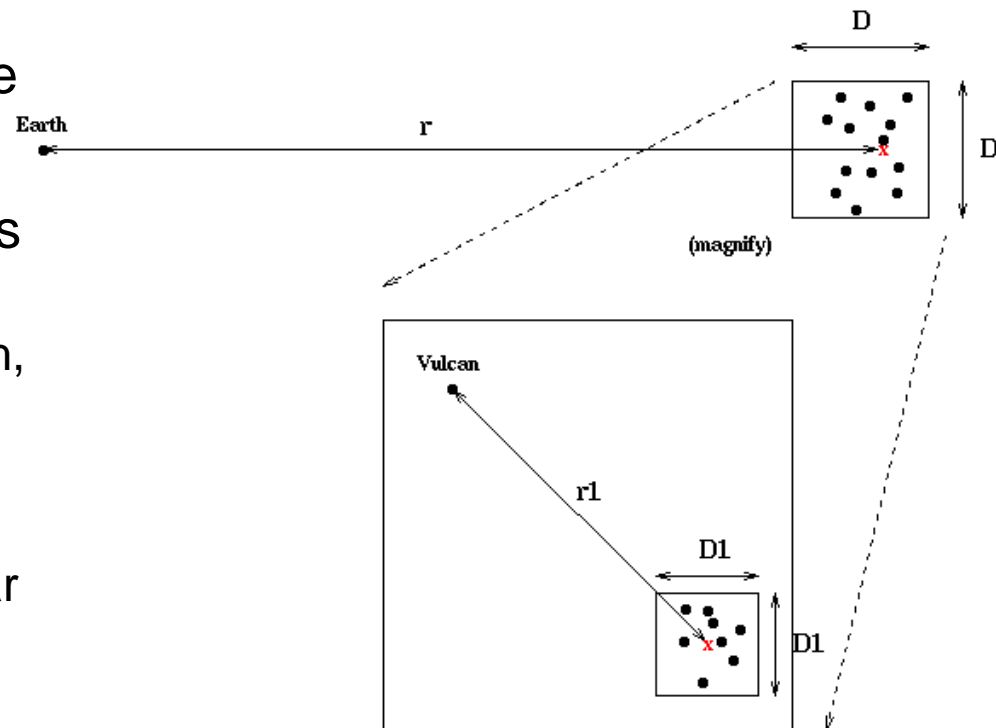
- New idea: recursively divide the box.

– If you are in Andromeda, Milky Way (the galaxy we are part of) could appear like a white dot. So, can be approximated by a point mass.

– Within Andromeda, picture repeats itself

- As long as D_1/r_1 is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
- If you are on Vulcan, another solar system in Andromeda can be a white dot.
- Boxes nest in boxes recursively

Replacing Clusters by their Centers of Mass Recursively



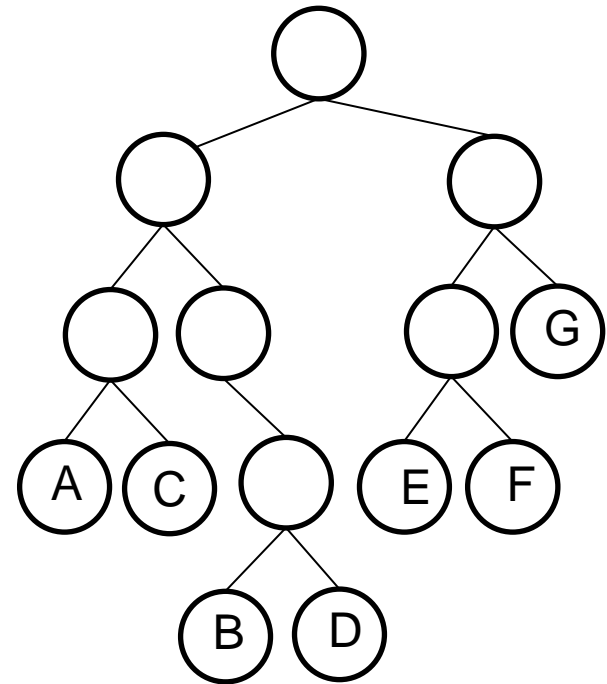
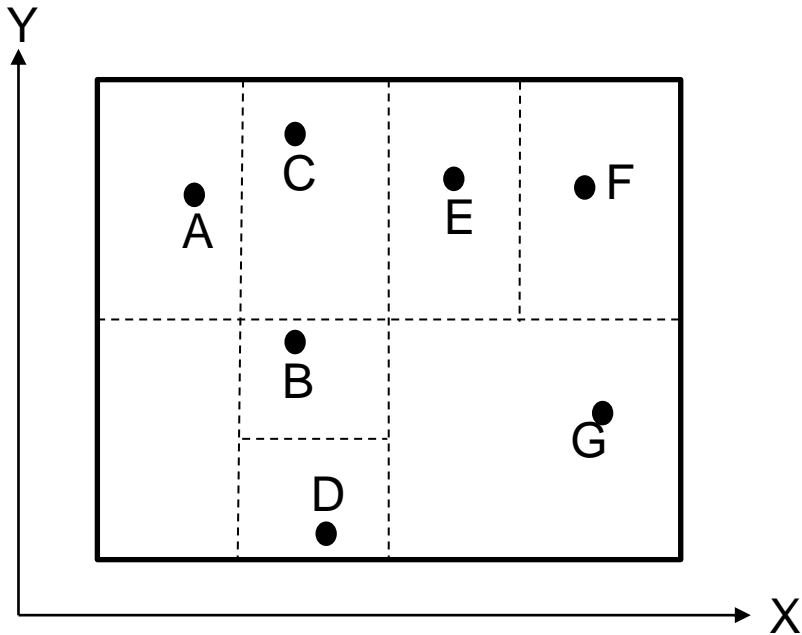
Tree Codes: Divide-Conquer Approach

- Data structures needed:
 - Quad-trees
 - Octrees

Background – metric trees

e.g. K-dimensional (kd-), Vantage Point (vp-), quad-trees, octrees, ball-trees

2-dimensional space of points Binary kd-tree, 1 point /leaf cell

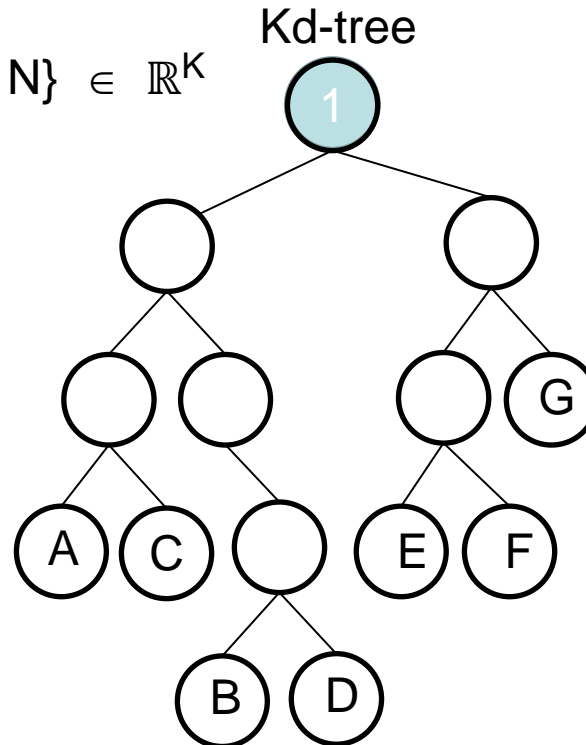


Background - metric trees

Typical use: traverse the tree (often repeatedly), truncate the traversal at some intermediate node if a domain-specific criteria is not met.

Input points = $\{1, 2, \dots, N\} \in \mathbb{R}^K$

Cost ???

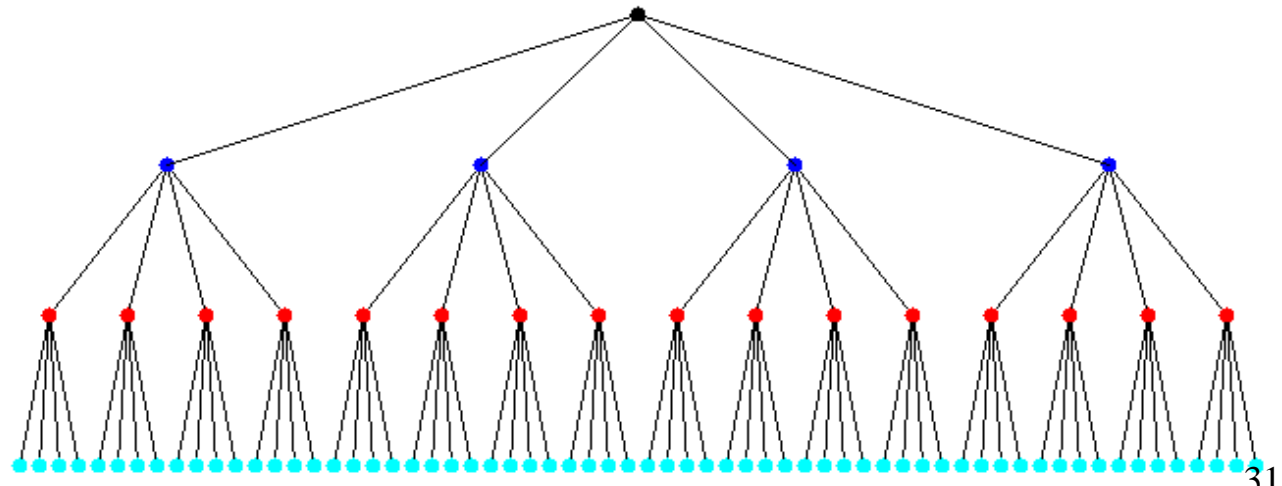
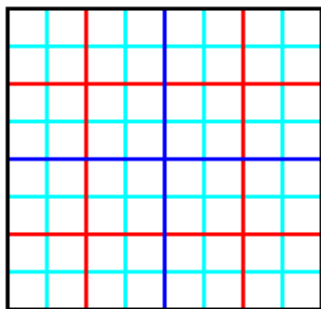


E.g. Does the distance from CM to me $< D/r$?

Quad Tree

- Data structure to subdivide the plane
 - Nodes can contain coordinates of center of box, side length.
 - Eventually also coordinates of CM, total mass, etc.
- In a **complete** quad tree, each non-leaf node has 4 children

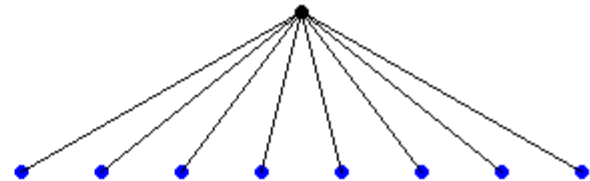
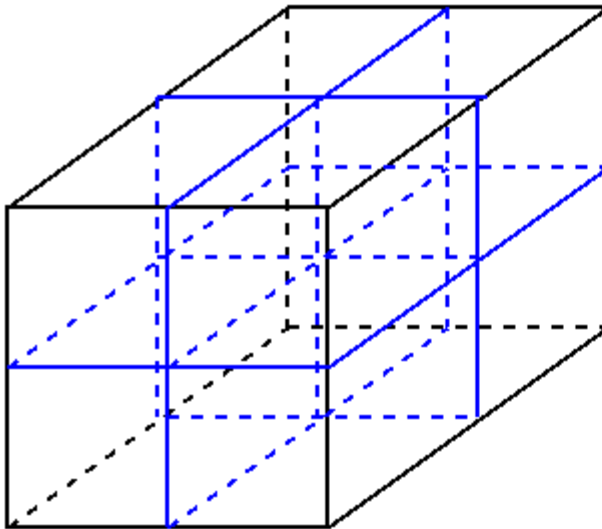
A Complete Quadtree with 4 Levels



Octree or Oct Tree

- Similar data structure for subdividing 3D space

2 Levels of an Octree

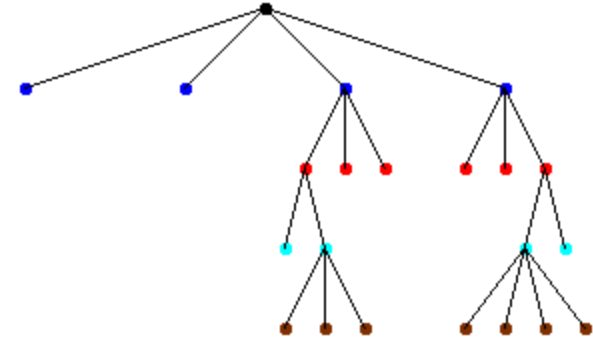
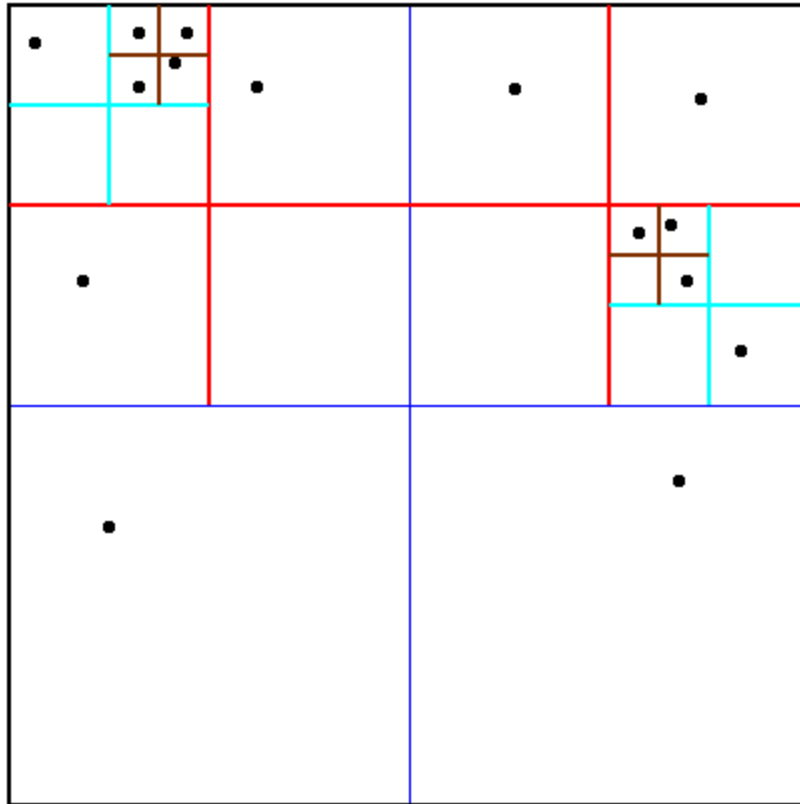


Using Quad Tree and Octree

- Begin by constructing a tree to hold all the particles
 - Interesting cases have nonuniformly distributed particles
 - In a complete tree most nodes would be empty, a waste of space and time
 - **Adaptive** Quad (Oct) Tree only subdivides space where particles are located
- For each particle, traverse the tree to compute force on it

Using Quad Tree and Octree

Adaptive quadtree where no square contains more than 1 particle



Child nodes enumerated counterclockwise from SW corner, empty ones excluded

- In practice, have $q > 1$ particles/square; tuning parameter (code to build data structure on hidden slide)