OCTREES, QUADTREES AND BARNES-HUT

LINA PULGARIN DUQUE*

Abstract. This paper surveys the use of Octrees and Quadtrees. Special emphasis is given to the implementation of the Barnes-Hut algorithm and it's use of trees for n-body simulations. Implementation of this is done in Julia code, using Quadtrees.

Key words. Octree, Quadtree, Barnes-Hut, n-body, Julia

1. Introduction.

Tree data structures have a wide range of uses in computer science. The basic concept behind a tree is to systematically and hierarchically divide data based on a chosen quality. For our review of the Barnes-Hut algorithm we will focus on Quadtrees and Octrees that divide spatial data based on the position of particles.

The Barnes-Hut algorithm was developed in 1986 by Josh Barnes & Piet Hut as a solution to the gravitational N-body problem. A brute force implementation for modeling gravitational interactions between n-bodies(i.e. particles) is of order N^2 computation. The implementation of the Barnes-Hut algorithm reduces this to order $NlogN$. [1]

The paper is organized as follows. An explanation of the the tree construction and algorithm implementation is in section 2, a brief explanation of the physics and efficiency of tree construction is presented in section 3, our algorithmic implementation and benchmark results are in section 4, a comparison to competing algorithms and further applications are in section 5 and the conclusions follow in section 6.

2. Tree Construction and Traversion.

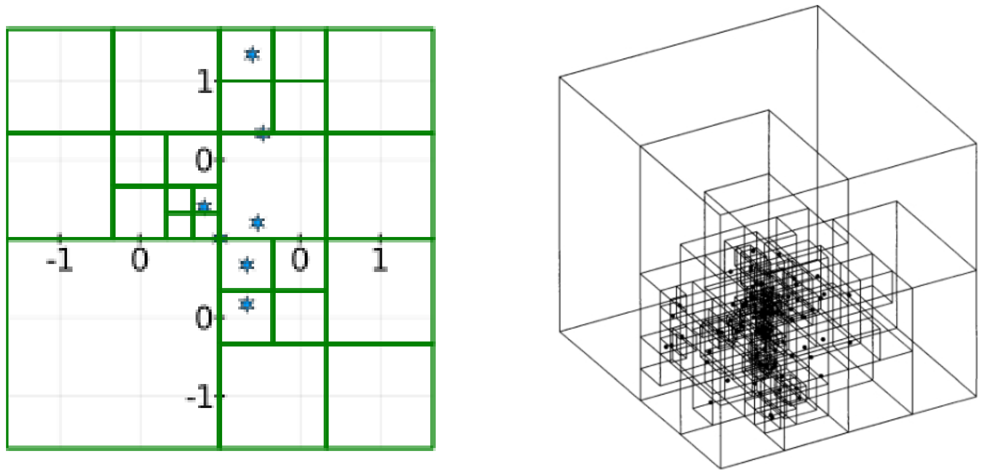
The Barnes-Hut algorithm begins with an empty box, this is the root of our tree. This box can either be 3D or 2D depending on the data that is being simulated. The particles are placed in the box one-by-one. Each box can only have one particle [1]. As particles are being assigned, if there is an existing particle in the box then the box is subdivided into equal quadrants(child boxes) and our outer box becomes a node. For a 3D simulation this subdivision consists of eight child boxes that are split in the x,y,z axes along the midpoint of our root box. For a 2D simulation this is four child boxes along the x, y axes.

Once subdivided, the existing particle finds a home among one of the child boxes and the current particle is recursively placed in one of our newly formed child boxes. This process is done recursively, and the child boxes become nodes as more particles are placed (parents as more children are created, or branches as more leafs are added). Each node corresponds to a volume of space, containing within it the corresponding particles as well as pointers to it's parents and any children [1].

Figure 1a is a visualization of this 2D implementation. Seven particles are placed in our box and subdivided until every box has at least one particle. For our implementation we placed all particles on the edges of our boxes as belonging towards the Northwest direction. In the figure we can see that because the Southwest corner had no particles it was not subdivided beyond the initial subdivision that occurred when the first particle was placed. Figure 1b is from [8] and is an example of the division of space in three dimensions for a star cluster.

Our tree structure can be used to store any information about our particle besides

*For MIT 18:335 Spring 2021.



(a) Division of space in 2D. (b) Division of space in3D [8].

Fig. 1: Fig (a) displays the subdivision of a 2D space with seven particles in the space. For particles on the edge of our subdivision they are placed in the direction of a Northwest box. Fig (b) displays subdivision of a 2D space with a star cluster in the space. Galaxies are often distributed such that the center is significantly denser than it's outer limits [8].

it's location in space. We can store it's mass, velocity, acceleration etc. Starting from the root(the largest box), the number of subdivisions needed to reach a particle(height of the tree) can be estimated from the average size of a cell containing one or more particles [8] or rather it's interparticle spacing [1]. This spacing can be represented by the average volume of the root cell divided by the number of simulation particles N.

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

as motivated by [8] this means that the height of our tree is is $O(log_2 N^{1/3})$, which expanded out,

$$log_2 N^{1/3} = \frac{1}{3log2} logN \simeq logN.$$

Since the tree has N particles (and therefore N leaves) the tree can be constructed in $O(NlogN)$. For an N-body simulation it is also necessary to store in the parent nodes the total mass and center of mass of all the particles it contains. This is done backwards, from the children to parents and can also be done in $O(NlogN)$ time[5].

3. Physics Motivation.

In an N-body simulation the simulation progresses by calculating acceleration of particles at every time step. In a brute force method this is done by calculating the force that each and every particle exerts on each other, this is computationally expensive resulting in $O(N^2)$. The effects of force on acceleration can be reduced down into this form

$$a_i = G \sum_{j \neq i} m_j \frac{r_j - r_i}{|r_j - r_i|^3}$$

and G is the gravitational constant[7].

The brute force algorithm in pseudo-code is as follows

Algorithm 3.1 Brute force N-body

```
for ( n=1:iterations )
    for ( i=1:length(particles) )
        acceleration = 0
        for ( j=1:length(particles) )
            acceleration += CalculateAcceleration(particles[i], particles[j])
        UpdateVelocityAndPosition(particles[i], acceleration)
    UpdateUniverse(particles, positions, velocities)
```

For someone who wants to perform simulations over a long period of time this method is cost prohibitive. Since most stellar and galactical simulations have at least 10^5 particles, there existed plenty of motivation to innovate past the brute force method [8]. One of the reasons that the Barnes-Hut algorithm and a tree structure works well for N-body simulations is that is that if a group of particles are at a $significant$ distance from the particle on which they are exerting a force, then they can be modeled as one massive point particle. The center of mass for the group being used as their point location.

This criterion for a $significant$ distance is intrinsic to the operation of the Barnes-Hut algorithm. It is marked by the ratio of the of the length, s, of the box being evaluated and the radius, d, from the box's center of mass to the current particle. This ratio s/d is compared to a tolerance parameter θ, if $s/d \leq \theta$ then all particles in this box are treated as one, otherwise the operation is repeated on the child boxes until the condition is met or a box with no children i.e. a particle, is found [1]. θ is usually set to 0.5, with $\theta = 0$ forcing the algorithm to break down into the brute force method outlined in Algorithm 3.1. This comparison is done after the tree has been created and when we are advancing the simulation, it does not affect the creation of the tree.

The number of operations required to compute the force on a single particle, using the Barnes-Hut method, was demonstrated by Lars Hernquist in 1988 using geometry [5].

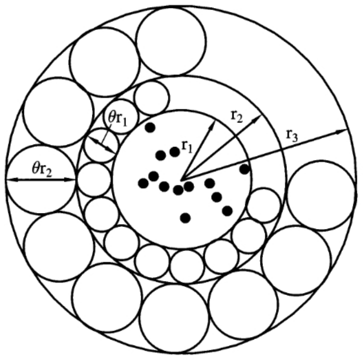


Fig. 2: An interpretation of Hernquist's logic [8].

As in Figure 2 consider a single particle in the center of a homogeneous sphere.

The particles close to the center particle will be contained within a shell of radius r_1. The remaining particles are organized into pseudoparticles(much like our boxes), contained within concentric shells of radius r_i, as r_i(distance from center particle) increases so does the size of the pseudoparticles.

The number of operations i.e. interactions, can be estimated from the number of these pseudo particles. This number of sub units in each shell is given by

$$n_{sub}^{i} \sim \frac{4\pi r_i^3 \theta}{\frac{4}{3}\pi r_i^3 \theta^3/8} = \frac{24}{\theta^2}\ . \quad [5]$$

Therefore total number of interactions is then estimated by $n_{int} \sim 24 n_{sh}/\theta^2 + n_0$, where n_0 is the number of particles in our innermost shell with radius r_1 and n_{sh} is the total number of concentric shells. Through geometric properties of these shells, this formula ends up as

$$n_{int} \sim \frac{24}{\theta^2} \frac{log\left[\theta\left(3N/4\pi\right)^{1/3}\right]}{log\left(1+\theta\right)} + \frac{4\pi}{3\theta^3} \quad [5]$$

and for large N and $\theta > 0$ is dominated by the denominator such that $n_{int} \sim logN/\theta^2$. Thus when using a Barnes-Hut or similar tree algorithm, for a universe containing N particles, the number of operations necessary to compute the force on all N particles scales as $\mathcal{O}\left(NlogN\right)$ [5].

4. Implementation. For our implementation of the Barnes-Hut algorithm we first create the substructure for what we have been referring to as boxes. Our box is created so that we have four replicable children boxes(our subdivisions) and a pointer that indicates a parent or lack-of.

Algorithm 4.1 Structure for Box

```
mutable struct Box
    parent::Union{Box, Bool, Array}
    NW::Union{Box, Array}
    NE::Union{Box, Array}
    SW::Union{Box, Array}
    SE::Union{Box, Array}
```

We take advantage of Julia's parametric composite types as shown in Algorithm 4.1. This structure consists of four subdivisions named after their cardinal direction and a *parent*. They can hold either another *Box* type or an *array*, the parent can also hold a *bool*. As we fill our *Box* structure if a subdivision is necessary(one particle is already present in our Box) then we simply update the corresponding cardinal direction with a new *Box* structure. This subdivision is also referred to by *branches*, the identity of what's in the *parent* field informs as to whether we are in a branch, a node or a leaf. A leaf, can hold whatever information about our particles that we want. As the tree structure is built, parallel structures can also be built to hold any information about our particles that is needed. Since this information, even if it's in a different tree, will have the same directions it is easily reached.

To fill our *Box* structure we use the following pseudo-code,

Algorithm 4.2 Tree Construction (UpdateBranches)

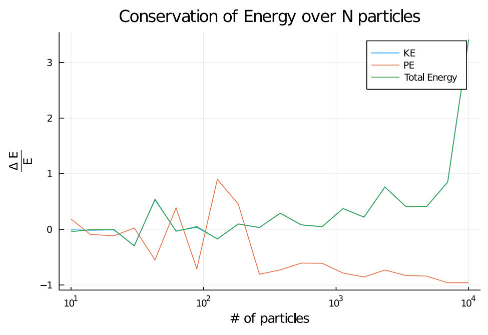
```
input: Three empty instances of Box that correspond to the dimensions of the box,
       the particles and the masses, as well as the universe that is being modeled
for ( i=1:size(univ,1) )
    particle = univ[i]
    pos = particle[i, ipos]
    mass = particle[i, imass]
    isinbox = IsInBox(pos, dimensions)
    if particles.parent == false and isinbox then
        update our particles Box instance to hold the current particle
        update the corresponding mass branch
    else if particles.parent !=false and isinbox then
        if particles.parent!=true then
            ExistingParticle == particles.parent
            SplitBox(dimensions)
            NewDir = FindDirection(ExistingParticle, dimensions.parent)
            particles.NewDir = ExistingParticle
            UpdateBranches(dimensions, particles, masses, particle)

        else
            dir = FindDirection(particle, dimensions.parent
            UpdateBranches(dimensions.dir, particles.dir, masses.dir, particle)
    else
        dir = FindDirection(particle, dimensions.parent)
        UpdateBrances(dimensions.dir, particles.dir, masses.dir, particle)
```

Algorithm 4.2 can be adjusted for 2D or 3D models. It is recursive and the first if statement where the corresponding dimensions, mass and particles boxes are updated is the exit statement. In this implementation we held all of the particle information in the particles box as well as the mass box, so it is possible to eliminate the mass box.

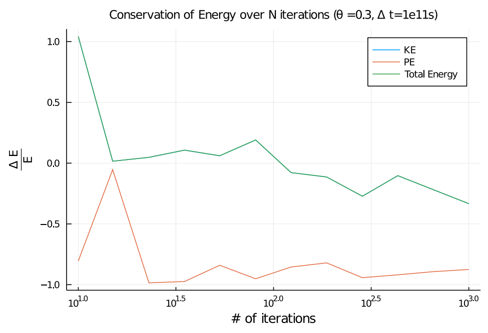
After using Algorithm 4.2 to update all branches, a similar recursive algorithm is used to work backwards and populate the parent node of the mass branch with the information of all the particles that are held in the node. This facilitates our implementation of the Barnes-Hut algorithm where we treat certain boxes as one particle instead of doing calculations on the individual particles. This tree is rebuilt at the beginning of every iteration, since the location of our particles has changed.

To traverse the tree we iterate over the four directions NW, NE, SW, SE. In the Barnes-Hut version of the N-body simulation, instead of calculating and summing all the particle-particle interactions we use the tree to estimate the force on all N particles. We compare the radial distance from the current particle to the four boxes in our tree. While iterating through the children of our tree, and as described in section 2, we use the dimensions of our boxes, as well as the center of mass of our boxes to determine whether we need to iterate into our child boxes.

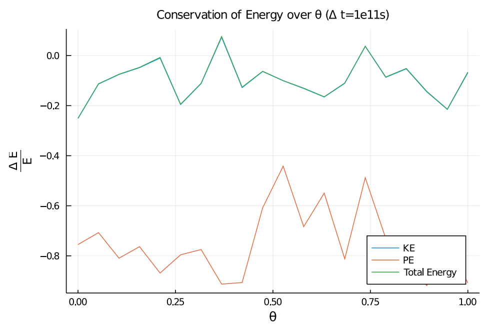
In order to measure to the effectiveness of our algorithm in a quick and effective way we will look at conservation of total energy. This method tends to underestimate errors, but it is a good indication of the health of our implementation [3]. Zadunaisky surveys various methods of evaluating the accuracy of N-body simulations. The "reverse" test, where one integrates the equations of motion backwards to see if the initial



(a)



(b)



(c)

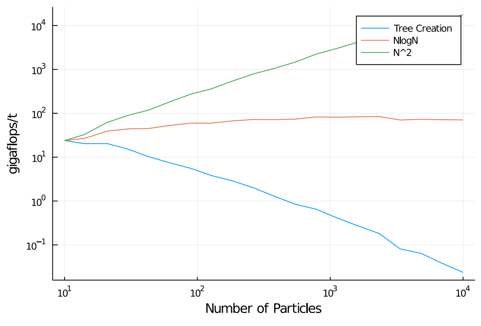
Fig. 3: All figures are made from our Barnes-Hut implementation and feature the relative change of energy on the y-axis. Fig (a) was run with 20 iterations, a θ value of 0.3 and a Δt of 1e11 seconds. Fig (b) was run with $10^{2.5}$ particles, a θ value of 0.3 and Δt of 1e11 seconds. Fig (c) was run with $10^{2.5}$ particles and iterations and Δt of 1e11 seconds.

conditions are reconstructed seems particularly fun [10].

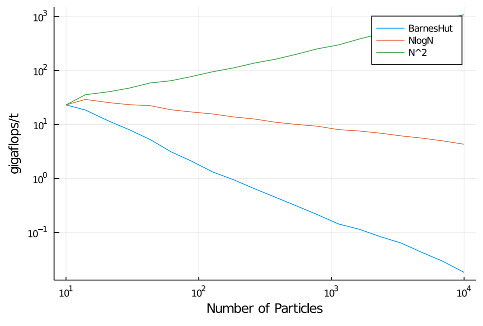
In all of our performance tests we found that Kinetic Energy dominated the relative change in total energy. Since our model is updating the forces at which particles move, and this is where our errors are introduced, this is to be expected. In Figure 3a, we find that the conservation of the total energy of our model begins to break down around 10^4 particles. Since competent models require 10^5 particles this should be refined. However, this might be influenced by the low number of iterations that were chosen for this test- 20.

Looking at Figure 3b we find that increasing the number of iterations increases the relative conservation of energy between the initial and final state of the universe. While comforting, this doesn't speak to the accuracy of the model - so for a serious implementation needs to be taken into consideration with other factors.

Figure 3c shows that the value of θ doesn't seem to have a large impact on conservation. However, since θ is the paramater the controls the amount of pseudoparticles we create, and branches of our tree we have to dive through - it is beneficial in terms of timing, to pick a high theta. In addition to these variables that were looked at, and depending on how many particles a model has, it is also possible to fine tune the Δt along with θ to find an optimal balance between accuracy and efficiency.



(a)



(b)

Fig. 4: These plots represent the benchmark of our implementation. The gigaflops were calculated using the Julia library GFlops.jl and compared to the elapsed time. Fig(a) benchmarks the tree creation process as outlined in Algorithm 4.2 and Fig(b) benchmarks the Barnes-Hut implementation, using θ of 0.3, Δt of 1e11 and a single iteration.

In addition to this, we looked at the performance of both the tree creation and the Barnes-Hut algorithm. We measured the gigaflops rate. The computer we were using was not optimized, but bench marked against N^2 and $NlogN$ in Figure 4 we see that there is room for improvement in our implementation. There is plenty of opportunities to parallelize both the tree creation and the Barnes-Hut algorithm itself.

5. Competing Algorithms and Further Applications. The next best algorithm for N-body simulations builds off the Barnes-Hut algorithm, it is called the Fast Multipole Method(FMM). One of the weaknesses of Barnes-Hut is that as pseudoparticles are formed, information on the spatial distribution of the cells is lost, and this introduces an error [8]. In FMM, a similar tree structure to that of Barnes-Hut is formed but instead of calculating forces, the potential is calculated for every single leaf. When traversing the tree, instead of using θ(center of mass and size of box) only the position and size of the box is used, not the mass.

The trade off between Barnes-Hut and FMM is that the expansion used on the potential can become expensive. However, FMM is a $\mathcal{O}(N)$ algorithm [4] since it computes cell-cell interactions instead of particle-cell. Instead of θ, the number of multipole expansions controls the accuracy of the algorithm [2].

The Parallel Multipole Tree Algorithm(PMTA) is a hybrid between Barnes-Hut and FMM. In PMTA a cell is allowed to have more than one particle. When traversing the tree the multipole expansions are computed only on the box determined to be far enough from the current iterative box, the rest of the children in that box are not visited. This hybrid is thus both dependent on the number of multipole expansions and θ [2].

Within n-body simulations, Barnes-Hut is simple to implement but there is a loss on accuracy. FMM is harder to implement but there is a gain in accuracy and performance. Some of this interaction can be seen in Figure 5 where Hwu [6] benchmarked the performance of tree and FMM methods on both CPUs and GPUs. On a CPU FMM performed better, but on an optimized GPU they performed quite similar. The decision should come down to the time it takes for implementation and accuracy [2].

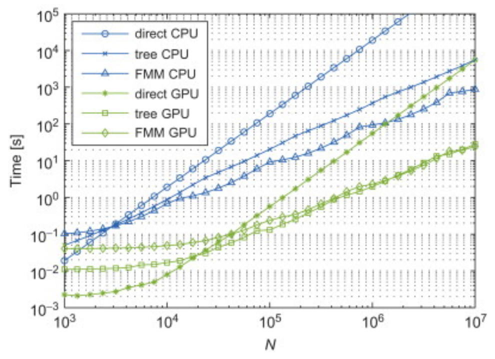


Fig. 5: This plot is taken from Hwu [6] and compares the computational time used by FMM and tree methods(Barnes-Hut) for N-body simulations implemented on CPUs and GPUs. The x-axis is the number of particles in the simulation.

It should also be noted that FMM were voted as one of the best algorithms of the 20th century and has applications far more reaching than Barnes-Hut. At the center of it all though is the Octree[9].

6. Conclusions. The use of Octrees for representation of spatial data is advantageous when dealing with applications that have high levels of interactions between itself. Barnes-Hut algorithm and it's descendants are able to replicate the particle-particle interactions of N-body systems without sacrificing too much accuracy through the use of trees.

The disadvantage to using a tree algorithm is their sensitivity to the the variables that control the creation and traversion of our trees. However much exploration is being done in this area, and there is innovation that can still occur, especially when exploring modern Machine Learning algorithms.

Finally we note that Octrees have a wide range of uses beyond nbody simulations. Their recursive decomposition and representation of data can be used in computer graphics, robotics, computer vision and cartography [9].

7. Acknowledgements. I would like to thank Prof. Steven G. Johnson for the engaging zoom lectures and patience during this pandemic. To better times.

REFERENCES

[1] J. Barnes and P. Hut, *A hierarchical O(N log N) force-calculation algorithm*, , 324 (1986), pp. 446–449, https://doi.org/10.1038/324446a0.
[2] G. Blelloch and G. Narlikar, *A practical comparison of n-body algorithms*, in Parallel Algorithms, Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1997.
[3] T. Boekholt and S. Portegies Zwart, *On the reliability of N-body simulations*, Computational Astrophysics and Cosmology, 2 (2015), 2, p. 2, https://doi.org/10.1186/s40668-014-0005-3, https://arxiv.org/abs/1411.6671.
[4] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, Journal of Computational Physics, 73 (1987), pp. 325–348, https://doi.org/https://doi.org/10.1016/0021-9991(87)90140-9, https://www.sciencedirect.com/science/article/pii/0021999187901409.
[5] L. Hernquist, *Hierarchical n-body methods*, Computer Physics Communications, 48 (1988), pp. 107–115, https://doi.org/https://doi.org/10.1016/0010-4655(88)90028-8, https://

www.sciencedirect.com/science/article/pii/0010465588900288.

[6] W. Hwu, *GPU Computing Gems Emerald Edition*, 01 2011, https://doi.org/10.1016/C2010-0-65709-9.

[7] I. Newton, *Principia mathematica*, Philosophiae Naturalis Principia Mathematica, (1972).

[8] S. Pfalzner and P. Gibbon, *Many-Body Tree Methods in Physics*, Cambridge University Press, 1996, https://doi.org/10.1017/CBO9780511529368.

[9] H. Samet, *The quadtree and related hierarchical data structures*, ACM Comput. Surv., 16 (1984), p. 187–260, https://doi.org/10.1145/356924.356930, https://doi.org/10.1145/356924.356930.

[10] P. E. Zadunaisky, *On the Accuracy in the Numerical Solution of the N-Body Problem*, Celestial Mechanics, 20 (1979), pp. 209–230, https://doi.org/10.1007/BF01371363.