EG UK Computer Graphics & Visual Computing (2023)
D. Hunter and P. Vangorp (Editors)

Using The Barnes-Hut Approximation For Fast N-Body Simulations In Computer Graphics

Peter Dravecky and Ian Stephenson

Bournemouth University

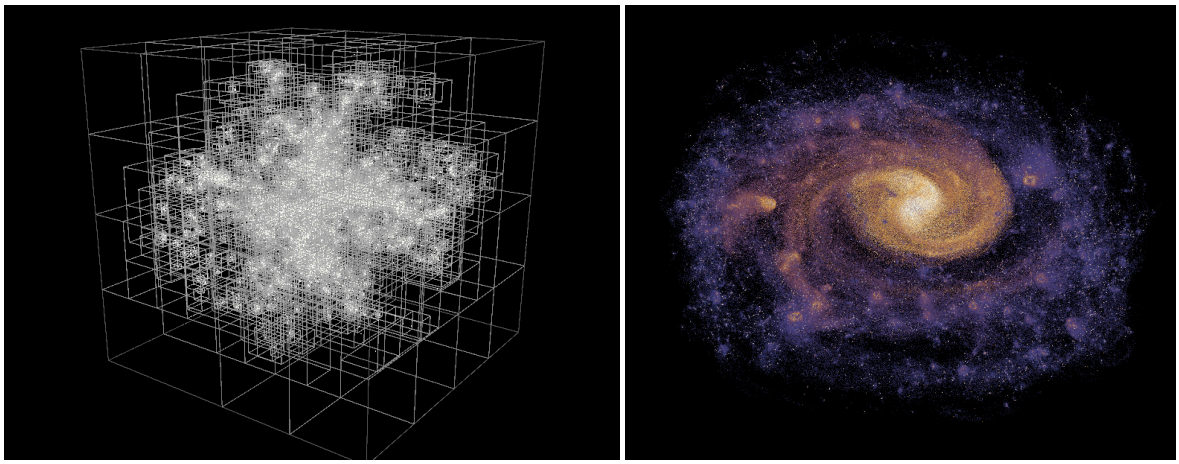

Figure 1: *a) Octree Visualisation b) Simulation Results*

Abstract

Particle systems in CG often encounter performance issues when all the particles rely on mutual influence, producing an $O(N^2)$ performance. The Barnes-Hut approximation is used in the field of astrophysics to provide sufficiently accurate results in $O(Nlog(N))$ time. Here we explore a hardware accelerated implementation of this algorithm, implemented within SideFX Houdini — the commercial tool typically used for particle work in film. We are able to demonstrate a workflow with integrates into the existing artist friendly environment, with performance improved by orders of magnitudes for typically large simulations, and negligible visual change in results.

CCS Concepts

• Computing methodologies → Scientific visualization; Massively parallel and high-performance simulations; Massively parallel algorithms; • Applied computing → Media arts;

1. Introduction

Particle simulations simulating mutual influence on each other, often called n-body systems, are nothing new for the VFX industry, being used in anything from flocking to fluid dynamics. Nowadays, these methods often rely on a mix of Eulerian and Lagrangian methods to achieve adequate quality to performance ratio, particle-only methods being considered too demanding in large scale simulations due to the complexity of the base N-Body algorithm. Although performant, these simulations can be considered inaccurate in some niché cases, like fluid simulations with a high Reynolds number. [Cas88]. These specific cases are common in the field of cosmological simulations (the simulation of dark matter and baryons on cosmological scales is almost entirely inviscous), where different methods have been researched to achieve efficient particle-only approaches. Although the performance boost they offer makes large simulations viable for use by 3D artists on standard hardware, there are no public implementations of them in any popular 3D software.

DOI: 10.2312/cgvc.20231197

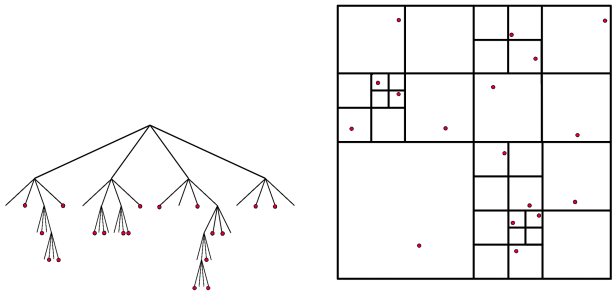

Figure 2: *Tree-Code Method Visual Examples (red dots = particles)*

In this paper we explore a version of *The Barnes-Hut Approximation* [JB86], which decreases the complexity of the original algorithm from $O(N^2)$ to $O(Nlog(N))$, while also using hardware acceleration to achieve best performance possible for a particle-only method. We implement this method in an artist-friendly way into SideFX Houdini, a popular 3D software often used for procedural geometry and visual effects, improving the performance of the implementation to the original algorithm while offering more artistic control over the final result.

2. Background

2.1. Gravitational N-Body Simulation

The dynamics used in the cosmological context for a system of N particles interacting gravitationally is typically an alternative formulation of Newton's law of universal gravitation, which accounts for the solid angle effect [HE88]:

$$\vec{F}_i = -\sum_{j \neq i} \frac{G m_i m_j (\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j|^2 + \varepsilon^2}$$

Where $\vec{F}_i$ is the force acting on a particle i with mass m_i and a position vector $\vec{r}_i$, with j representing each iterated particle. This equation also contains a softening length $\varepsilon > 0$, which avoids the gravitational force rising infinitely when particles get too close to each other.

2.2. Barnes-Hut Approximation

Among the different methods for approximating the base n-body algorithm (referred to as Particle-Particle (PP)), The Barnes-Hut Approximation is a Tree Code (TC) method. In the tree-code method, particles are grouped into clusters or nodes, and the forces between nodes are approximated using a multi-pole expansion. The tree structure is constructed dynamically, with nodes further away from a given particle being merged together until a predetermined level of accuracy is reached (figure 2). This tree hierarchy in 3-dimensional simulations is referred to as an octree, since every 3-dimensional section is separated by subdivision into 8 smaller parts. This approach reduces the number of force calculations needed, improving the computational efficiency of the simulation. [JB86]

In the Barnes-Hut approximation, the level of accuracy is set by a ratio of the cell's width and the distance between the point and the cell. If this ratio falls below a custom threshold (often referred to as **theta**), we treat the cell as a center of force and don't traverse down the octree any further.

Other notable methods include Particle-Mesh Method (PM), Fast Multipole Method (FMM), Self-consistent Field (SCF) or the Symplectic Method. [TH08] The FMM and the Barnes-Hut generally offer the biggest performance increase over the PP method [BN97], thought the performance boost largely depends on specific implementation techniques, hardware used, and the scale and purpose of the simulation.

2.3. Parallelisation

Burtscher and Pingali propose a parallelized version of the algorithm [BP11], which allows a significant performance boost specifically when running the parallel processes on the GPU. It efficiently gets rid of the recursive nature of the tree-building algorithm and deliberately exploits some of the architectural features of the GPU to speed-up the computation even more. Since it performs the entire computation on the GPU, the exchange of data between the CPU and the GPU is only performed at the start and the end of the algorithm, with most buffers not having to transfer any data from or to the CPU at all, which offers a significant performance boost.

2.4. Houdini

SideFX Houdini is a popular software used in the 3D industry for achieving large scale visual effects, such as procedural geometry, fluid and cloth dynamics or muscle simulation. It offers unprecedented artistic control and editability by using a node-based approach, allowing the user both the option to use pre-made, more abstract subnetworks, while also allowing scripting and the creation of your own nodes using various programming languages.

3. Implementation

Implementation of the BH algorythm consists of four steps:

1. Calculate the bounds of the simulation.
2. Construct a top-down octree hierarchy which connects all the bodies within the simulation.
3. Calculate the center of mass and total mass of each node in the octree.
4. Calculate the forces acting on each body with the help of the octree.

Each of these is implemented as a seperate Houdini OpenCL node, based on Burtscher and Pingali's approach (figure 3).

3.1. Houdini

3.2. Bouding Box Calculation

The first kernel performs the bounding box calculation, which is done by simple comparison of all position values. To optimize this, each work-group first performs a reduction operation between all its work-items to find the local bounds. When all the work-groups

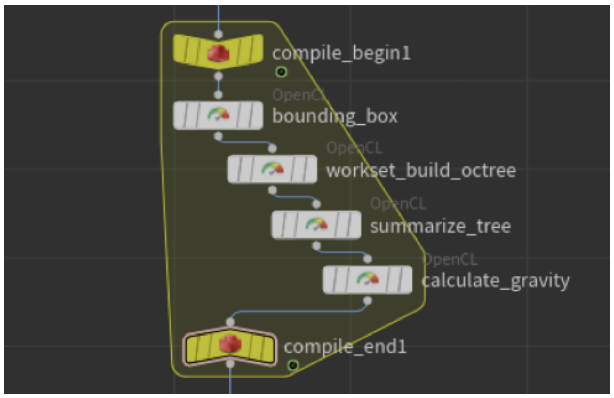

Figure 3: *Houdini node setup*

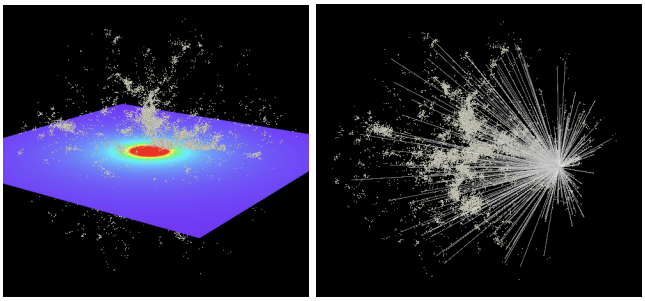

Figure 4: *Houdini Visulisations*

have finished, the last one to finish then computes the global maximum and minimum. To accurately pick the work-group which finishes last, an integer attribute is used, which is atomically incremented every time a work-group finishes. When this number equals the total number of work-groups, we know that we are in the final work-group. Side Note: This process might be more efficient by counting down to zero instead of counting up to the number of work-groups, allowing us to have one less global variable.

3.3. Octree Generation

3.3.1. The Base Algorithm

To parallelize the recursive nature of an octree construction, the cells are allocated to a 1-dimensional array representing the octree in a round-robin fashion.

Each work-item travels down the octree according to the particle position until it finds an unoccupied cell (-1).

When it gets to an unoccupied cell it tries to lock it by atomically writing a lock value (-2). If it succeeds, the work-item inserts a new body. The work-items which fail retry within the while loop until they succeed. To prevent swamping the main memory with while loop iterations, each failed work-item waits until the work-items that succeeded in the current iteration are finished.

If the work-item finds a cell that's occupied already, it locks the cell and inserts new cells into the octree until the two bodies each belong to a separate cell. To know where in the 1-D array a new cell is created, a global variable keeps track of the bottom index, which is atomically decremented every time a new cell is created.

3.3.2. Implementing Maximum Depth

In an optimal situation there would never be more cells in the octree than there are particles in the simulation due to its hierarchical nature – this is unfortunately not the case when the particles are spread out unevenly. Since the particles in a base n-body simulation are collisionless, a situation may arise when 2 particles have the same position vector – which would make the kernel function run until it passes the bottom of the array and crashes. To mitigate this, a maximum depth parameter was implemented. Whenever a work-item reaches an occupied cell which is at the maximum depth, instead of subdividing the cell further, it sums the masses of the body and the cell and computes the new center of gravity.

3.4. N-Body Octree Summarization

The 3rd kernel traverses the octree from bottom up. For each cell, it computes its center of mass and the total mass of all the bodies included in the cell. All cells start with negative mass, which makes it easy to check which cells have been calculated. When a work-item is processing a cell, whose children cells haven't been calculated yet, it waits for them to be ready in a similar fashion to the locking mechanism used in the 2nd kernel.

3.5. Force Calculation

At the start of the final kernel, the 1st work-item in each work-group pre-computes the maximum radius for each level in the octree according to theta, the accuracy ratio set at the start of the simulation - with each level the radius threshold gets 2x smaller. Subsequently, each work-item representing one body traverses down the octree and computes its squared distance to each cell. If this cell falls within the precalculated radius, it continues down, if not, it calculates cell's gravitational potential on the body.

The implemented algorithm works as a DOP network sub-solver node containing four OpenCL nodes bound in a compile block - each OpenCL node in Houdini represents one kernel function, and the compile blocks allows them to share data blocks sent to the GPU, preventing any unnecessary communication between the CPU and the GPU.

This implementation is then abstracted away by an user interface that allows the user to change virtually any parameter that gets passed down to the simulation, the base Barnes-Hut Approximation being complemented by additional parameters like Maximum Depth and Relative Mass, the latter preserving the same total mass of the simulation independent of particle count, allowing the artist to block out the simulation with a smaller particle count, a technique often use in prototyping visual effects.

Additional guide geometry has also been created to provide a visual representation of some harder-to-grasp concepts of the algorithm, like the function of the theta parameter show in in figure 4 or the octree visualization in figure 1.

No. of Points	Timer per step (ms)					
	P-P	Barnes-Hut				
		$\theta = 1.5$	$\theta = 1$	$\theta = 0.5$	$\theta = 0.25$	$\theta = 0.125$
5,000	9.84	34	37.1	38.82	39	43.25
15,000	17	35.84	41	41.23	43	44
50,000	23.95	42.73	43.27	44.95	49.07	53.37
150,000	108	58	59.02	66	84.77	130
500,000	1,225	122.6	141.4	200.6	346.1	699.11
1,500,000	9,886	293.6	341.1	503.3	890.8	1,761
5,000,000	119,836	941	1,186	1,970	3,872	8,332

Table 1: *Performance comparison table*

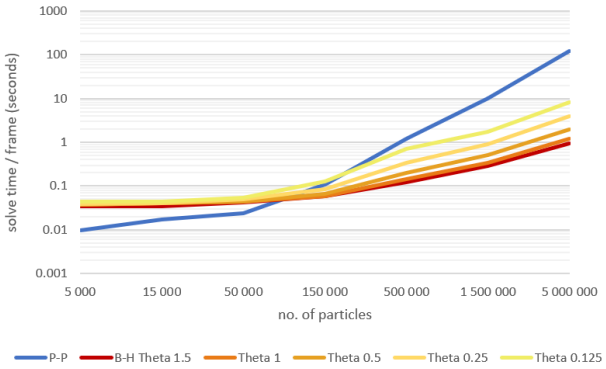

Figure 5: *Performance comparison graph*

4. Results

4.1. Performance

A simple particle-particle implementation was also created in Houdini and compared to the Barnes-Hut approximation. Results are show in figure 5 and table 1

Up until 50,000 particles, the PP algorithm performs better than the implemented Barnes-Hut algorithm, as expected as there is an overhead in creating the octree. Above this threshold, the performance signifigantly improves, its solve time being 20-times faster at 1.5 million particles and about 100-times faster at 5 million. This makes the Barnes-Hut algorithm preferred choice for particle simulations with $N > 50,000$.

4.2. Effect of Theta

The performance difference between various Theta values was also measured, with the performance increasing further as this parameter is increased. In terms of accuracy, the visual difference between the various approximations was negligible. A theta of 0.5 has been often recommended to provide good performance/accuracy ratio for scientific applications, and can be relaxed further for visual use. [Hee17]

4.3. Artistic Controls

In terms of artistic control, the n-body algorithm is implemented as a subsolver node – this allows it to interact with any other forces and control nodes already included inside HoudiniFX. This, together with the parameter menu to control the smoothing and the scale of the simulation offers great artistic control. Most of the experiments included some sort of repulsive force to keep particles from merging in the center of the simulation – This was either POP Axis Force or POP Attract. These settings combined with changing the world scale produced quick simulations of singular star clusters or entire networks of galaxies, as seen in fig 1

5. Conclusions

Our implementation offers an significant performance boost compared to the base algorithm. It also provides artistic control over the result and the option to combine it with other useful tools inside the Houdini environment.

References

[BN97] BLELLOCH G., NARLIKAR G.: A Practical Comparison of *n*-body Algorithms. In *Parallel Algorithms*, Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1997. 2

[BP11] BURTSCHER M., PINGALI K.: An efficient CUDA Implementation of the Tree-based Barnes Hut N-Body Algorithm. 2

[Cas88] CASULLI V.: Eulerian-lagrangian methods for the navier-stokes equations at high reynolds number. *International Journal for Numerical Methods in Fluids 8*, 10 (1988), 1349–1360. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.1650081016, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.1650081016, doi:https://doi.org/10.1002/fld.1650081016. 1

[HE88] HOCKNEY R. W., EASTWOOD J. W.: *Computer Simulation Using Particles*. CRC Press, 1988. 2

[Hee17] HEER J.: Interactive explanation of the Barnes-Hut approximation. https://github.com/jheer/barnes-hut, 2017. 4

[JB86] JOSH BARNES P. H.: A hierarchical o(n log n) force-calculation algorithm. *Nature 324*, 6096 (Dec. 1986), 446–449. doi:10.1038/324446a0. 2

[TH08] TRENTI M., HUT P.: N-body simulations (gravitational). *Scholarpedia 3*, 5 (2008), 3930. revision #91544. doi:10.4249/scholarpedia.3930. 2