N-body gravitational simulation using Tree Codes in a video game setting

Martin Paule
School of Design and Informatics
Abertay University
DUNDEE, DD1 1HG, UK

ABSTRACT

Context/Background: Accurately simulating kinematics of N-bodies in space has always inherently been limited by the exponential increase in necessary calculations as more bodies are added.
Throughout history, different algorithms were devised to reduce this computational strain.

Aim: The proposed project aims to develop an application that makes use of such algorithms to efficiently simulate many bodies with as little error as possible.

Method: The main approach this project will take in increasing performance is integrating Tree Codes with the aim of reducing the number of necessary gravitational calculations.
Given time, the application's performance can be further improved by implementing CPU/GPU multithreading.

Results: Following the successful implementation of Tree codes, the performance of the application should be increased notably.
If the parallelisation was to be implemented, further increase in performance is to be expected.

Conclusion: With research and development in this area, this project aims to prove the effectiveness of Tree codes in general, provide additional insight in this area, as well as give an example of how this can be useful in game development.

Keywords

Tree codes, N-body simulation, kinematics,

1. INTRODUCTION

Videogames always strive to simulate a believable and enjoyable environment. There are several factors that attribute to successfully achieving this, such as quality of models or level design. No less important in defining a virtual world are its physics. Despite their importance, dynamic calculations that are done at every frame such as gravitational physics are often simplified to leave more computational power for other features of the application.
Since traditionally, games are set on a solid surface, the gravitational forces of individual small objects are extremely negligible compared to the scene's main gravitational force. Because of this their respective calculations using their forces are traditionally ignored altogether. But even when it comes to the main gravitational force, it is rare to come across realistic equations. This can be attributed to two main reasons;
First reason is purely due to optimisation; this way not only can some calculations be avoided but also certain problematic variables such as the *Gravitational coefficient G* (6.6743×10^{-11}) can be ignored. In the case of *G*, the number is extremely small, commonly leading to data loss upon storing or use.
Secondly, user experience is generally better when the gravity is less strong – people like the carnage it allows, namely in games with a destructible environment (racing games such as *Flatout: Ultimate Carnage (Bugbear Entertainment, 2012)* or open-world like the *Grand Theft Auto V (Rockstar North, 2015)).*
Even in many games set in space, gravity is often simulated through simpler ways – a perfect example of this is *Angry Birds: Space (Rovio Entertainment Corporation, 2012),* where each planet has a collider around it, which acts as an explicit border for the planet's gravitatitonal field. The individual planets don't affect each other and neither do smaller objects.
A good example of an application which does aim to simulate space in a realistic fashion is *Universe Sandbox (Giant Army, 2015).*
While this application has accumulated a lot of additional features and mechanics throughout its years of development (such as temperature and wind on planets, realistic particles, VR, etc), their core mechanics (gravitational kinematics) are the same and their extensive, publicly available development diary as well as a fan-made Wikipedia page provide a lot of insight into how this team of programmers tackled various issues in different stages of development.

This proposed project intends to develop an application that accurately and efficiently simulates the movement of bodies in space.
Due to the nature of space consisting of an extremely large number of bodies with varying masses, to realistically simulate it the program should be able to handle a fairly many bodies with relative ease.
As this project means to showcase how scientific approaches can be adapted in the development of videogames, several basic gameplay elements will also be included to give a brief idea of how this could be realised.
When aiming to simulate a certain phenomenon in an accurate, scientific fashion, proven and established equations should be used for all calculations. Fortunately, all necessary theoretical knowledge on this matter has been known for several centuries, going as far back as Isaac Newton's Laws of Motion, from which all the required formulae can be derived.

2. BACKGROUND

Despite having abundant theoretical knowledge on this matter, directly integrating equations derived from Newton's Laws of Motion into a virtual simulation is proven to be extremely computationally inefficient.
Since each individual object must take all the other objects' forces (no matter how miniscule) into account, with a given number of bodies N, the number of required individual calculations can be calculated as $N \times (N - 1)$.

In programming the number of calculations tends to play a vital role when it comes to the resulting performance.
A widely used way to define the scalability of algorithms is the Big O notation, which directly refers to the computational complexity (in other words, how the number of calculations scales depending on the amount of handled data) of a given problem.
In this case, the computational complexity of directly integrating Newton's formulae is O(N^2), where N, referring to the number of bodies in the simulation, is the handled data.
This result is generally considered to be rather inefficient and it is exactly this computational issue that this project aims to solve by implementing performance improving algorithms.

Over the years, many attempts were made to devise a method which can more efficiently calculate N-body dynamics without becoming too inaccurate.

Presently, there are two major approaches to solve this issue; Tree Codes[1] and by them inspired Fast Multipole Method.
While the Fast Multipole Method is considered to be generally faster, its implementation is rather complex and its hierarchical trees are not adaptive.
For the purpose of this project, Tree codes will be used as they allow for a relatively fast and straightforward implementation, they are highly extendable with adaptive trees and there is much documentation on the algorithm itself, as well as its improvements. Their computational complexity has been defined as O(N log N).

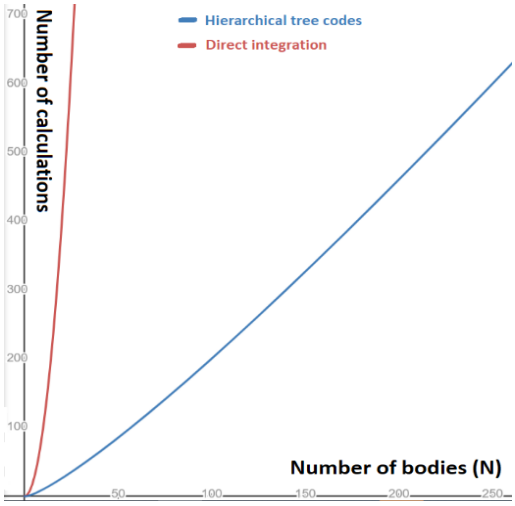

Figure 1: Comparison of computational complexity using Tree codes and using direct integration. Note how much more efficient and scalable the former is.

In its core, the hierarchical Tree Codes work by recursively subdividing the space into 8 cubic cells (Oct-trees) until each object is in a cell alone.
Following this division, when calculating the gravitational forces acting upon a given body, faraway bodies are treated as one force defined by their combined mass and averaged position, whereas nearby objects are calculated directly.
For more details into how this algorithm works please refer to the original article [1].

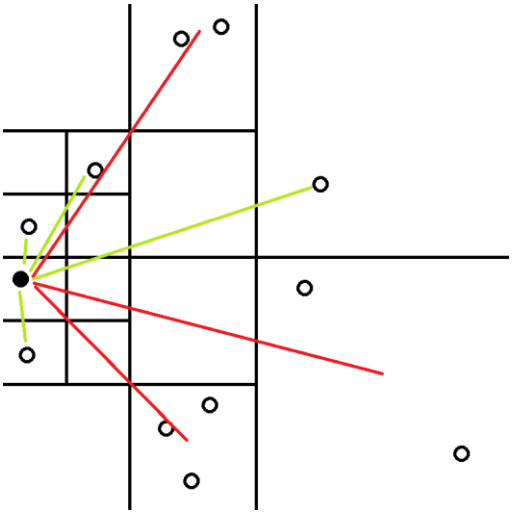
Figure 2: Visual representation of tree codes, simplified in 2D. This figure displays how the gravitational forces acting upon the black object are calculated. All faraway objects' forces are combined and treated as one (red). Nearby objects are calculated normally (green).

This approach was very well received by the scientific community, inspiring new iterations, improvements and implementation for almost 4 decades, such as introducing time step controlling, effectively skipping some calculations of insignificant slow or distant objects [3] or implementing high-end parallelisation [2].
Following its success in simulating bodies in space, other applications for this algorithm have also emerged, such as using oct-trees for calculation of inter-particle electrostatic interactions [4].

It is also worth noting that one of the original authors of this method, Piet Hut, has returned to the topic of N-body Gravitational simulations in 2008 with a very concise and informative paper [5] summarizing all the major new approaches in solving this method, as well as notable improvements onto the original idea using Tree Codes.

3. METHOD

3.1 IDE Choice

Unreal Engine 5 has been chosen as the development IDE for this application.
While there are several environments to simulate this problem with differing focus on science/games, this choice has been inspired by the following factors:

1. Newly introduced Mega Scans in UE5 with their respective pre-defined dynamic tessellation. This would allow for the use of realistic models with efficient long-distance rendering performance.
2. C++, the language native to this IDE, has many math libraries focusing on precise scientific calculations
3. Unreal Engine comes with its own robust system of analytics tools, allowing for reliable and convenient benchmarking.

3.2 Development goals

The workload on the application will be split into several important development goals, each posing its own challenges. These main goals are defined in a relatively chronological fashion as follows:

3.2.1 Framework setup

Firstly, a well-running base program needs to be established.

1. Presently the direct integration of equations derived from Newton's laws is to be used to calculate the movement of the gravitational bodies.
    a. The computational complexity here is O(N^2).
    b. This approach will be preserved for future accuracy testing, even after being replaced by faster algorithms.
2. At this point, most of the UI / models / general code is to be set up and implemented.
3. The UI should be able to display and evaluate the application's current state (meaning performance, attributes of individual objects, accuracy, etc).

3.2.2 Performance improvement

The second part of developing the application will be focused on improving the application's performance.

1. The main approach in improving performance that this project will take is implementing Tree Codes.
    a. As there have been many improvements, iterations and additions to this approach since it was first introduced, it is expected that aspects of the more recent and well-developed works will also be used as a development reference for this project.
    b. In its originally proposed version, the recursive oct-tree division of is done at every frame. Investigation and research will be conducted onto this, as it might be possible to devise a method that only updates changed cells, rather than recalculating the whole structure.
2. A very efficient way of further improving the performance of tree codes is using parallelisation. While there are many works on this topic, such as [2], significant performance improvement can be achieved even without delving extremely deep into this. This is not the main goal of this project, however, therefore this will be treated as additional features to be implemented given enough time.

3.2.2.1 Performance and accuracy

By combining the mass of faraway bodies and averaging their position, performance will increase at the slight expense of accuracy. Throughout the development process efforts will be made to find ways which would reduce the severity of this tradeoff.

This application aims to calculate and display both performance and accuracy.

For the purpose of this project, performance will be mostly considered in terms of what FPS (Frames per second) can this application comfortably achieve and maintain, as well as what is the number of calculations it must compute at every frame.

Accuracy is to be calculated by comparing the positions of gravitational bodies calculated using the Tree Codes with their positions achieved by using direct integration.

At this point it is unclear whether this will be calculated at the end or in real-time. The latter is, however, less likely as that would mean simultaneously simulating the efficient Tree codes along with direct integration, resulting in an application running slower than if only direct integration were to be used.

3.2.3 Implementing Gameplay

3. The final aim in developing this project will focus on implementing gameplay elements.

Since hierarchical Tree Codes are extremely general and transferrable, it is possible to implement this algorithm into a number of space-themed games.

At the present, the exact implementation of gameplay elements is still unclear. Some examples of what gameplay could be implemented are:

    a. User controlling a spaceship that must collect/avoid individual objects (for example pieces of space junk or small asteroids).
    b. Game where the player must launch an object from point A to point B using the present bodies' orbits.
    c. The player controls a black hole that must absorb objects to increase in size while avoiding other, bigger black holes.

4. SUMMARY

Following a successful development, this application will efficiently simulate kinematics of bodies in space, while retaining a fairly accurate structure. By achieving this, the effectiveness of Tree codes in an N-Body problem will once again be confirmed and demonstrated.

It is also expected that resulting application, as well as the recorded data accumulated throughout development, can provide further information and insight into this area of study.

Additionally, by this point it will be possible to evaluate how well this scientific approach fits in a videogame setting, as well as how well Unreal Engine 5 fares in development of highly accurate scientific simulations and what its strengths and weaknesses in this regard are.

5. REFERENCES

[1] Barnes, J., Hut, P. (1986) 'hierarchical O(N log N) force-calculation algorithm'. *Nature* 324, pp. 446–449. https://doi.org/10.1038/324446a0

[2] Winkel, M. et al. (2012) 'A massively parallel, multi-disciplinary Barnes–Hut tree code for extreme-scale N-body simulations', *Computer physics communications*, 183(4), pp. 880–889. doi:10.1016/j.cpc.2011.12.013.

[3] Zhu, Q. (2021) 'A momentum-conserving N-body scheme with individual time steps', *New astronomy*, 85, p. 101481. doi:10.1016/j.newast.2020.101481.

[4] Boutsikakis, P. Fede, O. Simonin (2022) 'Quasi-periodic boundary conditions for hierarchical algorithms used for the calculation of inter-particle electrostatic interactions', *Journal of Computational Physics*, p. 38. doi:10.1016/j.jcp.2022.111686

[5] P. Hut and M. Trenti (2008) 'Gravitational N-Body Simulations', *Scholarpedia*, 3(5):3930, p. 13. arXiv:0806.3950

6. BIBLIOGRAPHY

I. Winkel, M. (2013) 'The Barnes-Hut Tree Algorithm and its highly scalable parallel implementation PEPC', *Julich Supercomputing Centre*, p.27, id: http://hdl.handle.net/2128/5876

II. Brent Shapiro-Albert et al. (2010-present) Universe Sandbox Development Blog Available at: https://universesandbox.com/blog/category/development/ (Accessed: 18 October 2022).