eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Trabajo Fin de Grado
Grado en Electrónica

# Estudio e implementación del algoritmo Barnes-Hut para el cálculo de la interacción gravitatoria entre N-cuerpos

Autora:
Maia Aguirre Pascual de Zulueta
Director:
Germán Bordel García



Leioa, 16 de Junio de 2020

# Índice

## Abstract

In this work, we provide an original implementation of the Barnes-Hut algorithm in Python 3.7 both in 2D and 3D. This algorithm solves approximately the N-body problem and is well known for achieving order $\mathrm{O}(N\log(N))$ by treating nearby bodies as single individuals when observed from a far enough distance. Besides, we came up with a clever scheme for grouping those bodies: a proof of concept that turned out to perform accurately. Further, we analyzed the validity range of our prototype and correlated it to the direct-sum algorithm $\mathrm{O}(N^2)$ by means of a selected set of test examples. Our implementation of the BH algorithm relies heavily on a deeply nested tree data structure. As such, its manipulation is fundamentelly recursive and highly complex. This is the reason why we additionally have included an extended explanation, with drawings and schemes, of the rather cumbersome bookkeeping strategy involved in its use. Finally, we have uploaded the full code to the GitHub platform and thereby made it publicly available.

## Resumen

Este trabajo proporciona una implementación original del algoritmo Barnes-Hut en Python 3.7, tanto en 2D como en 3D. Este algoritmo resuelve el problema de N-cuerpos de forma aproximada y es conocido por lograr el orden $\mathrm{O}(N\log(N))$ al agrupar los cuerpos cercanos en un solo individuo cuando son observados desde una distancia lo suficientemente lejana. Además, concebimos un esquema original para agrupar esos cuerpos; esta prueba de concepto resultó funcionar con precisión. Asimismo, analizamos el rango de validez de nuestro prototipo y lo contrastamos con el algoritmo de suma directa $\mathrm{O}(N^2)$ mediante un conjunto seleccionado de ejemplos. Nuestra implementación del algoritmo BH depende en gran medida de una estructura de datos de tipo árbol profundamente anidada. Como tal, su manipulación es fundamentalmente recursiva y altamente compleja. Esta es la razón por la que hemos incluido una explicación extendida, con dibujos y esquemas, de la estrategia de contabilidad bastante engorrosa involucrada en su uso. Finalmente, hemos subido el código completo a la plataforma GitHub y, por lo tanto, lo hemos puesto a disposición del público.

# 1. Introducción y objetivos

El objetivo de este trabajo es implementar un prototipo del algoritmo Barnes-Hut (BH) en el lenguaje de programación Python y efectuar un estudio detallado de sus características principales mediante este programa.

Este algoritmo ha tenido históricamente un gran impacto en la simulación física por tener una complejidad computacional de orden $\mathrm{O}(N\log(N))$, permitiendo así que los cálculos del problema de N-cuerpos se escalen a conjuntos de datos mucho más grandes que los que permiten los algoritmos que calculan todos los pares de fuerzas.

Con este fin, empezaremos exponiendo brevemente los conceptos fundamentales de la mecánica Newtoniana ya que estos constituyen la base de la mecánica celeste. Las leyes de la mecánica cuántica y la relatividad general no serán discutidas en este trabajo por ser sus contribuciones de un orden de magnitud despreciable en el estudio de la mecánica celeste convencional. Comprender estas cuestiones resultaría imprescindible en casos extremos como colisiones de agujeros negros, explosión de supernovas, nacimiento del universo etc. que quedan totalmente excluidos de nuestro estudio.

A continuación, detallaremos las estructura tanto del algoritmo Barnes-Hut como del algoritmo clásico de suma directa o fuerza bruta, añadiendo una discretización original del espacio que será de gran utilidad a la hora de implementar el algoritmo BH. Esta clasificación se basa en emplear el sistema octal (base-8) cuando el espacio a categorizar es tridimensional y el sistema cuaternario (base-4) en su análogo bidimensional.

Por último, compararemos el desempeño de ambos algoritmos mediante una selección de problemas bien escogidos con el fin de discriminar la eficacia de los algoritmos bajo la influencia de diferentes parámetros.

Me gustaría recalcar que otro de los objetivos principales de este trabajo ha sido crear un documento que facilite la comprensión de este desconocido y asombroso algoritmo y explique de manera clara, sencilla y accesible las estructuras de datos empleadas.

## 2. Estado del Arte

La mecánica celeste (un término acuñado por Laplace en 1799) es la rama de la astronomía que se ocupa de estudiar los movimientos de los cuerpos celestes bajo la influencia de la gravedad. Su objetivo es calcular estas trayectorias aplicando la ley de gravitación universal de Isaac Newton (1643-1727) [1].

El primero en desarrollar las leyes que rigen las órbitas a partir de observaciones empíricas del movimiento de Marte fue Johannes Kepler (1571-1630) quién apoyó su trabajo en observaciones astronómicas realizadas por Tycho Brahe (1546-1601) [2].

Sin embargo, el inicio de lo que se conoce hoy en día como mecánica celeste no llegó hasta 1687 con la publicación de *Principia* de Isaac Newton que desarrolló su ley de gravitación universal basándose en el trabajo de Kepler, introduciendo la idea de que la fuerza que hace que los objetos caigan con aceleración constante en la Tierra es de la misma naturaleza que la fuerza que mantiene en movimiento los planetas y las estrellas [2].

La mecánica celeste fue posteriormente desarrollada en una ciencia madura por científicos célebres como Euler (1707-1783), Clairaut (1713-1765), D'Alembert (1717-1783), Lagrange (1736-1813), Laplace (1749-1827) y Gauss (1777-1855) [1].

El análisis sobre fuerzas interplanetarias comenzó históricamente con el problema de los dos cuerpos del que conocemos la solución exacta. Sin embargo, en el siglo XIX Henri Poincaré (1854-1912) demostró que el problema de los tres cuerpos no posee una solución general analítica [3]. Por lo tanto, queda totalmente descartado que el problema de N-cuerpos tenga una solución matemáticamente exacta. En consecuencia, los algoritmos que aproximan este problema numéricamente son de gran interés. Discutiremos en seguida la evolución que estos algoritmos han sufrido en cuanto a precisión y rapidez de cómputo a lo largo de los años, pero antes haremos una breve mención al dominio de validez de las leyes Newtonianas.

Hasta el comienzo del siglo XX, se consideraba que la mecánica Newtoniana constituía una descripción completa de todos los tipos de movimiento que ocurren en el universo. Hoy en día, sabemos que esto no es cierto y que el modelo de Newton no es más que una aproximación válida bajo ciertas circunstancias. Concretamente, el modelo falla cuando las velocidades de los objetos son próximas a la velocidad de la luz en el vacío y deberá ser modificado de acuerdo a la *teoría de la relatividad especial* de Einstein (1879-1955). Este tampoco será válido en regiones del espacio que están suficientemente curvadas como para que la geometría Euclidea no sea una buena aproximación y deberá ser sustituido por la *teoría de la relatividad general* de Einstein. Por último, el modelo es inapropiado en escalas atómicas o subatómicas (donde la acción es comparable a la

constante de Planck) y deberá ser reemplazado por la *mecánica cuántica* [1]. En este trabajo obviaremos completamente los efectos relativistas y cuánticos: restringiremos nuestras simulaciones al movimiento de objetos grandes (comparados con el átomo), lentos (comparados con la velocidad de la luz) e ignoraremos los efectos de las masas en la geometría espacio-temporal que lo alejan de la estructura Euclidea. Afortunadamente, la mecánica celeste convencional pertenece a esta categoría.

Conviene añadir que en nuestras simulaciones hemos empleado el sistema por unidad (ver anexo 1). Esto hace que una sola simulación permita representar una variedad infinita de sistemas físicos debido a que la elección de la masa y distancia de base ($M_0$ y $L_0$) es totalmente arbitraria y responde únicamente a la propia conveniencia de cálculo. Sin embargo, es evidente que existen límites físicos; dado que la cantidad de masa que puede ser contenida en un determinado volumen está acotada superiormente; una vez excedido este límite se provoca la aparición de un agujero negro. Este caso queda totalmente excluido de nuestro análisis.

En las últimas décadas el cálculo numérico ha sufrido una enorme evolución: a partir de la década de 1950, la velocidad de cómputo del hardware se ha multiplicado por un factor de $10^9$ y también lo hicieron los mejores algoritmos, generando un aumento combinado en la velocidad de una escala casi incomprensible [4].

En cuanto a los algoritmos que resuelven el problema de N-cuerpos, fue en 1985, un año antes de la publicación del algoritmo BH, cuando se empezó a representar la estructura espacial del universo mediante una estructura de datos de tipo árbol. Asimismo, los intervalos temporales escogidos para la integración serían mucho mayores salvo que dos cuerpos pasaran muy cerca el uno del otro, en cuyo caso los intervalos de tiempo serían extremadamente pequeños para no perder precisión [5]. Insertando estos ahorros a diferentes niveles del diseño, se estimaba que el tiempo de cómputo se vería reducido a $\mathrm{O}\big(N\log(N)\big)$ con lo que por ejemplo, un tiempo de ejecución estimado en 8000 horas se vería reducido a 20 horas [5].

Otros algoritmos en la literatura del año 1985 previos a BH consistían en: (i) representar el problema en un espacio de fase posición-velocidad y transformar el campo de fuerza usando la transformada rápida de Fourier (FFT) [11, 12]. Este algoritmo también es de orden $\mathrm{O}\big(N\log(N)\big)$; (ii) En [13] se sugiere un algoritmo con dos escalas temporales: para cada cuerpo del sistema se mantiene un seguimiento de los conjuntos de cuerpos "cercanos" y "lejanos" a él de manera que los cuerpos "lejanos" serán integrados con pasos temporales mayores que los cuerpos "cercanos". Con una inicialización homogénea de cuerpos este algoritmo tiene una complejidad asintótica de $\mathrm{O}(N^{1.5})$. (iii) La alternativa propuesta en [14] consiste en dividir el universo en celdas y calcular las interacciones cuerpo a cuerpo dentro de las celdas y después, las interacciones celda a celda. Para una distribución

uniforme, la complejidad de este algoritmo es de $\mathrm{O}(N^{4/3})$. Una variante de este método calcula las interacciones celda a celda con FFT, reduciendo la complejidad a $\mathrm{O}\big(N\log(N)\big)$ [5].

Todos estos algoritmos suponen grandes mejoras frente al algoritmo original (la solución trivial o directa) cuando la distribución de los cuerpos en el espacio es uniforme. Sin embargo, la simulación de distribuciones no uniformes tiene complejidad asintótica similar a la del algoritmo original [5] por lo que estas propuestas serán descartadas ya que los cuerpos celestes tienden a agruparse en cúmulos.

En los años siguientes, códigos de orden $\mathrm{O}\big(N\log(N)\big)$ como BH, basados en la estructura árbol, fueron exitosamente empleados para resolver problemas de dinámica de galaxias, formación de galaxias y formación de la estructura cosmológica (Barnes-Hut 1986, Hernquist 1987, Dubinski 1988) [8]. El algoritmo más popular y el que se emplea en la mayoría de estudios es el de Barnes-Hut.

En los años posteriores las mejoras en cálculo llegaron principalmente de la mano de la mejora del hardware (ley de Moore) [9]. A finales del siglo XX, se ha dado un cambio de paradigma en la supercomputación con el paso de máquinas de vectores a las nuevas máquinas masivamente paralelas [8]. Al hablar de computación paralela nos referimos a un grupo de núcleos que resuelven el problema cooperando entre sí. En estos ordenadores, cada procesador opera independientemente, pero la información se propaga velozmente de unos a otros. Es decir, la capacidad de procesamiento de cada CPU ha dejado de incrementar exponencialmente, pero sin embargo, el número de CPUs empleados crece sin parar [7, 10]. Esta metodología requiere un considerable rediseño de los algoritmos [8].

Ha habido varios intentos de paralelizar BH, podemos citar a Hills y Barnes (1987), Makino y Hut (1989) y al más exitoso, Salmon (1990) [8].

Recientemente, la computación paralela se ha universalizado mediante el auge de la tecnología GPU [15]. A raíz de estos recientes desarrollos de hardware se ha realizado la implementación del algoritmo BH enteramente en GPU [16].

El propósito de este trabajo es mucho más modesto. En primer lugar, hemos empleado el lenguaje de programación Python que es no compilado y no está realmente adaptado al cálculo numérico intensivo. Sin embargo, es un lenguaje muy sencillo de programar, su entorno de programación es muy cómodo y cuenta con numerosas librerías.

En segundo lugar, el enfoque de este trabajo y su principal aportación ha consistido en la exploración de una nueva y original clasificación de la discretización del espacio. El programa desarrollado es básicamente un prototipo, una prueba de concepto. Precisamente por esta razón no hemos

intentado paralelizar el algoritmo y el código desarrollado es robusto y fiable pero queda pendiente de optimización.

# 3. Aproximación teórica

## 3.1. Problema gravitacional de N-cuerpos

En apartados anteriores hemos acordado que el problema de N-cuerpos que nos concierne consiste en la predicción de los movimientos individuales de un grupo de N-cuerpos celestes que interactúan entre sí gravitacionalmente [17].

El problema de N-cuerpos en mecánica clásica puede expresarse informalmente de la siguiente manera:

Dadas la posición y velocidad de un conjunto de N-cuerpos celestes en un instante de tiempo determinado, podremos calcular sus fuerzas interactivas y consecuentemente predecir sus posiciones y velocidades para un intervalo de tiempo posterior. Repitiendo este algoritmo iterativamente obtendremos las orbitas de los N-cuerpos celestes.

Formulación general en 3D:

Consideremos N masas puntuales $m_i$ $(1 \leq i \leq N)$ que se mueven bajo la influencia de la atracción gravitatoria mutua. A cada masa $m_i$ le corresponde un vector posición $\boldsymbol{r}_i$. La segunda ley de Newton establece que la masa de un cierto cuerpo por la aceleración que este experimenta es igual a la suma de todas las fuerzas que se ejercen sobre dicho cuerpo:

$$\sum_{j \neq i}^{n} \boldsymbol{F}_{ij} = m_i \frac{d^2 \boldsymbol{r}_i}{dt^2} \tag{1}$$

Por otro lado, la ley de gravedad de Newton establece que la fuerza gravitacional ejercida sobre la masa $m_j$ por la masa $m_i$ viene dada por:

$$\boldsymbol{F}_{ij} = \frac{G m_i m_j}{\left\| \boldsymbol{r}_j - \boldsymbol{r}_i \right\|^3} (\boldsymbol{r}_j - \boldsymbol{r}_i) \tag{2}$$

Siendo $G = 6.674 \cdot 10^{-11} \frac{\mathrm{m}^3}{\mathrm{kg\,s}^2}$ la constante de gravitación universal. Combinando las ecuaciones (1) y (2) deducimos que el conocimiento de las masas nos permite determinar la aceleración instantánea sufrida por cada cuerpo (ec. 3) y, en consecuencia, por integración en el tiempo, sus velocidades y posiciones en cada momento.

$$\boldsymbol{a}_i = \frac{d^2 \boldsymbol{r}_i}{dt^2} = \sum_{j \neq i}^{N} \frac{G m_j}{\left\| \boldsymbol{r}_j - \boldsymbol{r}_i \right\|^3} (\boldsymbol{r}_j - \boldsymbol{r}_i) \tag{3}$$

## 3.2. Cálculo numérico de las fuerzas

El objetivo de este trabajo es contrastar la eficacia de dos algoritmos que resuelven numéricamente el problema de N-cuerpos. Estos algoritmos, que explicaremos en detalle en los apartados consecutivos son: el método clásico o directo que llamaremos algoritmo de Fuerza Bruta (FB) y el algoritmo de Barnes-Hut (BH). Ambos están basados en calcular la fuerza de atracción gravitatoria (ec. 2) experimentada por cada cuerpo del sistema para después, eventualmente, poder determinar sus respectivas trayectorias. Una vez calculadas las aceleraciones (ec. 3) mediante el algoritmo FB o mediante BH, emplearemos el método de integración numérica de Euler para integrar las ecuaciones de movimiento y obtener las trayectorias. Este procedimiento de integración numérica es un método de primer orden, lo que significa que el error local es proporcional al cuadrado del tamaño del paso temporal, y el error global es proporcional al tamaño del mismo paso [19]. Es conocido que este simple método de integración no garantiza que la energía y el momento angular se conserven a largo plazo, pero hemos escogido el método de integración de Euler debido a su sencillez ya que la parte fundamental de este trabajo se centra en la implementación del algoritmo BH para el cálculo aproximado de las fuerzas.

Implementar el algoritmo FB para resolver el problema de N-cuerpos supone un coste computacional de orden $\mathrm{O}(N^2)$. Queda claro que el cálculo directo nos permite una obtención exacta de las fuerzas, pero supone un coste computacional que incrementa rápidamente con N. El objetivo de la resolución numérica del problema de N-cuerpos es estudiar sistemas celestes de miles de cuerpos lo que resulta prohibitivo con un coste cuadrático. Por ejemplo, si $N = 1000$ por cada iteración temporal habría que calcular $\frac{N(N-1)}{2} = 499500$ contribuciones de fuerza.

Nuestro objetivo es rebajar este coste computacional y para ello estudiaremos el algoritmo Barnes-Hut que destaca por ser de orden $\mathrm{O}(N \log N)$. En este caso si $N = 1000$ por cada iteración temporal, habría que calcular un número de contribuciones de fuerza del orden de $3000$.

Aún así, nos detendremos brevemente a explicar cómo ha sido implementado algoritmo FB.

### 3.3. Algoritmo de Fuerza Bruta

La fuerza de interacción gravitatoria que ejerce el cuerpo $i$ sobre el cuerpo $j$ es exactamente la misma (con el signo cambiado) que la que ejerce el cuerpo $j$ sobre el cuerpo $i$. Esto se debe a la antisimetría de la ecuación (2). Con el fin de que el número de veces que evaluemos los pares de fuerzas de interacción entre los cuerpos del sistema sea $\frac{N(N-1)}{2}$ y no $N^2$ debemos dar con un método que recuerde qué pares de fuerzas han sido evaluadas hasta el momento para no tener que recalcularlas.

El algoritmo que he implementado en Python almacenará la información de las fuerzas calculadas en matrices para así poder tener en cuenta el criterio previamente mencionado. Cada celda de la matriz contiene un vector con los componentes $[F_x^{ij}, F_y^{ij}, F_z^{ij}]$ que representa la fuerza de interacción entre los cuerpos representados por la fila $i$ y la columna $j$ en la que se encuentra la celda.

Obviamente, los términos de la diagonal de la matriz no deberán ser calculados. Además, para no calcular términos de fuerza redundantes, solo rellenaremos el triangulo superior de la matriz.

Para calcular la fuerza total a la que cada cuerpo está sometido debemos sumar la contribución de fuerza que el resto de cuerpos contenidos en el sistema ejercen sobre él (ec. 2). Traduciendo esto al lenguaje matricial deberemos:

- Crear un vector $M^0$ de longitud N que contiene la suma vectorial de las columnas de la matriz: $M^0 = [0, \boldsymbol{F}_{12}, \boldsymbol{F}_{13} + \boldsymbol{F}_{23}, \sum_{i=1}^{3} \boldsymbol{F}_{i4}, \dots, \sum_{i=1}^{N-1} \boldsymbol{F}_{iN}]$
- Crear un vector $M^1$ de longitud N que contiene la suma vectorial de las filas de la matriz: $M^1 = [\sum_{j=2}^{N} \boldsymbol{F}_{1j}, \sum_{j=3}^{N} \boldsymbol{F}_{2j}, \dots, \boldsymbol{F}_{N-1N}, 0]$

La fuerza total experimentada por cada cuerpo $k$ está dada por la componente $k$ de la diferencia de los vectores $M^0$ y $M^1$:

$$\boldsymbol{F}_k = \sum_{i=1}^{k-1} \boldsymbol{F}_{ik} - \sum_{j=k+1}^{N} \boldsymbol{F}_{kj}$$

El símbolo negativo se debe a que la matriz de fuerzas es antisimétrica: $\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$

### 3.4. Algoritmo de Barnes-Hut [18]

Para rebajar el coste computacional del problema de N-cuerpos celestes a $\mathrm{O}(N \log N)$ el algoritmo BH usa una estructura de datos de tipo árbol. Estos árboles son estructuras de datos jerárquicas basadas en la subdivisión virtual del espacio en celdas cúbicas en un espacio tridimensional o

cuadradas en un espacio bidimensional. Cada una de estas celdas cúbicas (resp. cuadradas) representa un nodo del árbol y será recursivamente dividida en ocho (resp. cuatro) subceldas siempre que más de un cuerpo se halle ocupando la misma celda. Todas las subceldas o celdas hijas tendrán exactamente la mitad de la longitud, anchura y altura de sus celdas madres. Denominaremos el nodo que contiene el espacio entero como "raíz". Así mismo, denominaremos "hojas" a los nodos que representen la división más pequeña del espacio. Por lo tanto, construiremos el árbol de celdas virtuales mediante (i) el descarte de celdas vacías, (ii) la aceptación de subceldas con un solo ocupante y (iii) la división recursiva de ocupaciones compartidas en subceldas. Por último, es un requisito del algoritmo que el árbol se reconstruya de cero a cada iteración temporal.

Teniendo en cuenta esta estructura de datos, la dinámica del algoritmo se implementa asignando a cada celda no vacía, así como a las celdas de orden superior que contienen más de una partícula, una pseudopartícula que contiene la masa total de la celda ubicada en el centro de masas de todas las partículas que contiene dicha celda. Todas las partículas reales individuales sienten la fuerza de todas las pseudopartículas del sistema que representan celdas lo suficientemente pequeñas y alejadas como para renunciar a la necesidad de una mayor división.

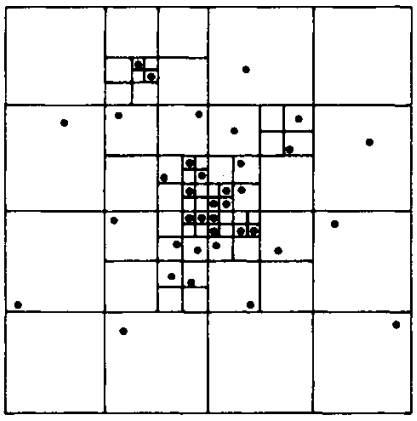

**Figura 1.a: Estructura árbol 2D [18]**

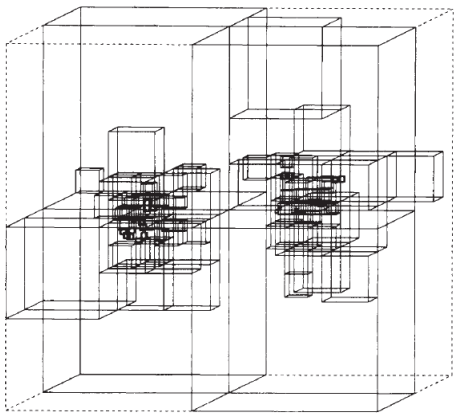

**Figura 1.b: Estructura árbol 3D [18]**

### 3.4.1. Construcción del árbol

El algoritmo original de Josh Barnes y Piet Hut construye la estructura árbol tridimensional de la siguiente manera (la reducción a 2D es trivial):

Empezamos con una celda cúbica que contiene el espacio entero que nos ocupa, es decir el nodo raíz. Uno a uno iremos colocando los cuerpos en la celda raíz. Si dos cuerpos se encuentran en la misma celda esta se ha de dividir en ocho subceldas cúbicas (la primera división ocurre al añadir el

segundo cuerpo al sistema). Cada celda ha de ser representada por una estructura de datos que contenga información sobre las subceldas que derivan de esta (masa total y posición de centro de masas) así como unos punteros a sus celdas hijas (ver figura 2). Cuando los N cuerpos hayan sido introducidos en el sistema, el espacio habrá quedado dividido en células cubicas de diferentes tamaños que contienen un solo cuerpo. Estas celdas portadoras de cuerpos individuales son agrupadas en celdas cúbicas de mayor tamaño que a su vez serán agrupadas en celdas cúbicas de mayor tamaño recursivamente hasta llegar a formar la celda raíz que contiene el sistema entero.

El tamaño medio de una celda portadora de un cuerpo individual es del orden de $\mathrm{O}(\log N)$ y el tiempo requerido para construir el árbol $\mathrm{O}(N \log N)$ [18]. Además, el tiempo requerido para etiquetar las celdas subdivididas con la masa total y la posición del centro de masas de todos los cuerpos que contienen, de modo que la información se propague en un recorrido descendiente desde la raíz hasta los cuerpos individuales es de $\mathrm{O}(N \log N)$ [18].

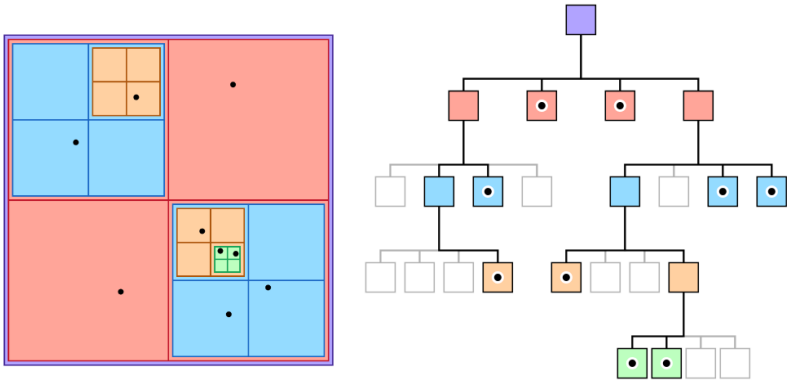

**Figura 2.a: Representación piramidal de la estructura interna del árbol 2D [20]**

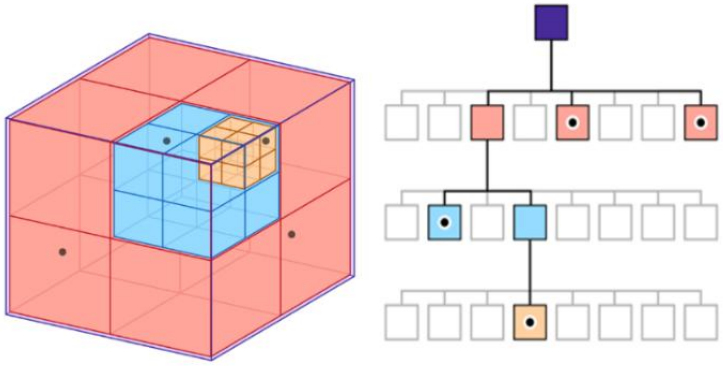

**Figura 2.b: Representación piramidal de la estructura interna del árbol 3D [21]**

### 3.4.2. Calculo de las fuerzas

Habiendo construido un árbol tal, la fuerza ejercida sobre cualquier cuerpo del sistema puede ser aproximada mediante un simple cálculo recursivo. Pongamos por ejemplo que queremos calcular la suma de las fuerzas ejercidas sobre el cuerpo $j$ por el resto de cuerpos del sistema. Para ello, recorreremos el árbol de raíz a hojas y calcularemos la fuerza de interacción gravitacional entre el cuerpo $j$ y las celdas (a partir de los datos masa total de cuerpos que contiene y posición del centro de masas) que no contengan el cuerpo $j$. Definiremos como *l* la longitud de la celda cuya interacción gravitacional con el cuerpo $j$ queremos calcular y *D* la distancia desde el centro de masas de esta celda a la posición del cuerpo $j$. Además, definimos $\theta$ como el parámetro de precisión umbral [18].

Si $l/D < \theta$ , aceptamos la interacción de la celda y el cuerpo $j$ por encontrarse lo "suficientemente lejos" el uno del otro y procedemos a calcular la fuerza de interacción gravitatoria entre el centro de masas de la celda y el cuerpo $j$. De esta forma hemos podido evaluar en un solo cálculo la fuerza de interacción gravitatoria que ejercen sobre el cuerpo $j$ todos los cuerpos ubicados en dicha celda. Este criterio de "lejanía suficiente" se basa en que los cuerpos que componen dicho cúmulo se encuentran a suficiente distancia del cuerpo $j$ como para que el cálculo conjunto (cuerpo $j$ a cúmulo) no suponga una gran pérdida de precisión comparando con lo que supondría el cálculo individualizado (cuerpo a cuerpo) de las fuerzas.

Por el contrario, de no cumplirse el criterio $l/D < \theta$ , recorreremos las ocho subceldas que cuelgan de esta y repetimos el proceso.

Analizaremos las consecuencias de la elección del parámetro $\theta$ sobre la precisión del algoritmo en el apartado 4.1. Por el momento cabe decir que es un parámetro ajustable entre los valores 0 y 1 ($0 < \theta < 1$) y determina la precisión de la simulación. Los valores de $\theta$ próximos a 1 aumentan la velocidad de la simulación, pero disminuyen su precisión y viceversa. Si $\theta = 0$ el algoritmo degenera en el algoritmo de Fuerza Bruta.

## 3.5. Prototipo original del algoritmo Barnes-Hut

A priori, el método propuesto por Barnes y Hut para la construcción del árbol (apartado 3.4.1) puede parecer sencillo de implementar pero esto es engañoso ya que en realidad se trata de un proceso bastante complejo. Además, el artículo original que explica este algoritmo no proporciona pautas especificas sobre cómo ha de ser implementado. Por este motivo, nosotros exploramos la idea de crear la estructura árbol de hojas a raíz, a la inversa del algoritmo BH original que construye el árbol de raíz a hojas. Creemos que esta clasificación a la inversa, como explicaremos a continuación, es

más natural, pero se trata en todo momento de una prueba de concepto. Para ello, empezaremos por discretizar el espacio en celdas de igual tamaño asegurando que no haya más de un cuerpo por celda. A continuación, agruparemos los cuerpos que se hallen en celdas contiguas e iremos guardando la información pertinente en el árbol. Las celdas vacías serán eliminadas y los cuerpos o cúmulos de cuerpos que no sean agrupados con otros cuerpos o cúmulos en celdas de tamaño superior (por ausencia de estos) serán promocionados (ver figura 3).

Otra discrepancia de nuestro algoritmo BH con respecto al original es el empleo de la base binaria para representar las posiciones de las celdas y sus respectivos niveles de profundidad en el árbol. El siguiente apartado se ocupa de estudiar este método detalladamente. Para ello, estudiaremos los casos del espacio unidimensional, bidimensional y tridimensional, de menor a mayor complejidad. Empezaremos analizando el caso más sencillo: el espacio unidimensional.

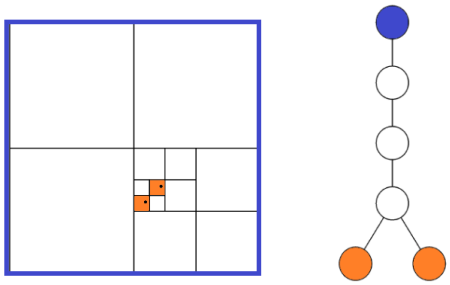

**Figura 3: Representación de la promoción en un árbol de 2D**

### 3.5.1. Clasificación espacial

#### Espacio unidimensional (base-2)

En este primer caso, el espacio que trataremos de clasificar en subdivisiones recursivas es unidimensional. Por lo tanto, estas subdivisiones serán bisecciones de las celdas. Dado que en cada partición crearemos dos celdas nuevas el empleo de la base-2 nos servirá de clasificación automática.

La base-2, como su propio nombre indica, la forman dos dígitos: el 0 y el 1. Nuestro convenio será el siguiente: el dígito 0 representará el lado izquierdo mientras que el dígito 1 representará el derecho. De esta manera, tras haberse efectuado la división de una celda, la subcelda de código “0” será la que ha quedado a la izquierda de la división mientras que la subcelda de código “1” será la que ha quedado a la derecha de la división. Está clasificación de las celdas sirve a cualquier nivel de profundidad del árbol. Concatenando el resultado de cada división (0 o 1) de una celda obtendremos su código de celda completo. El dígito más significativo (el primero por la izquierda) de este código

representará la clasificación obtenida tras la partición de la celda raíz. Así sucesivamente, hasta llegar a la última subdivisión sufrida por la celda. Por ejemplo, el código "1101" pertenece a una celda que ha sido dividida cuatro veces y por lo tanto es de tamaño $2^4$ veces inferior a la celda raíz. Además, en la primera, segunda y cuarta división ha quedado a la derecha de su celda madre mientras que en la tercera división ha quedado clasificada a la izquierda. De este modo, con un simple código queda totalmente trazada la información de las celdas. En los próximos apartados veremos las ventajas de este código a la hora de combinar celdas en celdas de tamaño superior, recordamos que nuestro algoritmo BH es construido de hojas a raíz.

### Espacio bidimensional (base-4)

Procedemos a continuación a estudiar la clasificación del espacio bidimensional, generalizando el método utilizado para tratar el espacio unidimensional. En este caso, el espacio será recursivamente dividido en cuatro celdas del mismo tamaño. Codificar estas subceldas empleando la base-4 supondrá una clasificación automática del espacio, homóloga a la empleada para el espacio unidimensional en base-2.

Empezaremos escribiendo el código en base-2 tanto de la coordenada vertical como de la horizontal correspondientes a la posición de la celda que queremos clasificar. Es decir, convertimos nuestra clasificación bidimensional en dos clasificaciones unidimensionales empleando el procedimiento del apartado anterior. De esta forma, obtendremos dos códigos de celda por celda: uno que representa la posición vertical y otro que representa la posición horizontal, ambos en base-2. A fin de poder traducir estos dos códigos de base-2 a uno solo de base-4 agruparemos por parejas los dígitos de los códigos vertical y horizontal. Esta agrupación se hará por niveles, correspondiendo cada nivel de profundidad en el árbol a la posición que ocupa cada dígito en el código de celda. A fin de cuentas, los códigos de celda de los ejes vertical y horizontal se concatenarán por parejas de derecha a izquierda. Por ejemplo, siendo la coordenada del eje horizontal en binario "011" y la del vertical "101" la coordenada de celda resultante será "01 10 11". Ahora bien, emplear dos dígitos en base-2 es equivalente a emplear un solo dígito en base-4 siendo la correspondencia: $00_2 \longrightarrow 0_4$, $01_2 \longrightarrow 1_4$, $10_2 \longrightarrow 2_4$, $11_2 \longrightarrow 3_4$. Por lo tanto, el código de celda del ejemplo anterior en base-4 resultaría: $(01\ 10\ 11)_2 \longrightarrow (123)_4$. Con el fin de que la división del espacio siga una lógica se hará acorde al esquema de la figura 4.

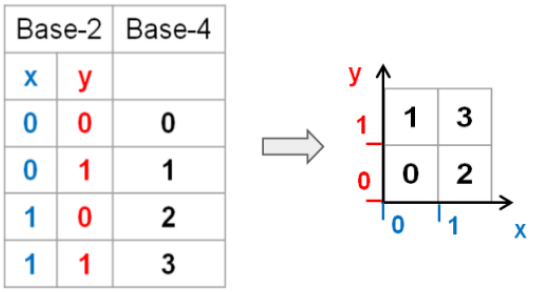


**Figura 4: Esquema de la división del espacio en base-4**

Con el propósito de ofrecer una mejor comprensión de la división y la clasificación del espacio en base-4 un árbol bidimensional de nivel cuatro es ilustrado en la figura 5. En este, el espacio entero (sin dividir) es el nodo cero o raíz del árbol, el color rojo representa los cuatro nodos del nivel uno, el color verde los dieciséis nodos de nivel dos y el color negro los sesenta y cuatro nodos hoja. A modo de ejemplo dispondremos cinco cuerpos (A, B, C, D y E) en el espacio que nos atañe. Cabe recalcar que en nuestro algoritmo el tamaño de cada cuerpo será equivalente al tamaño de celda de los nodos hoja y su posición se toma en referencia al vértice de abajo a la izquierda de dicha celda.

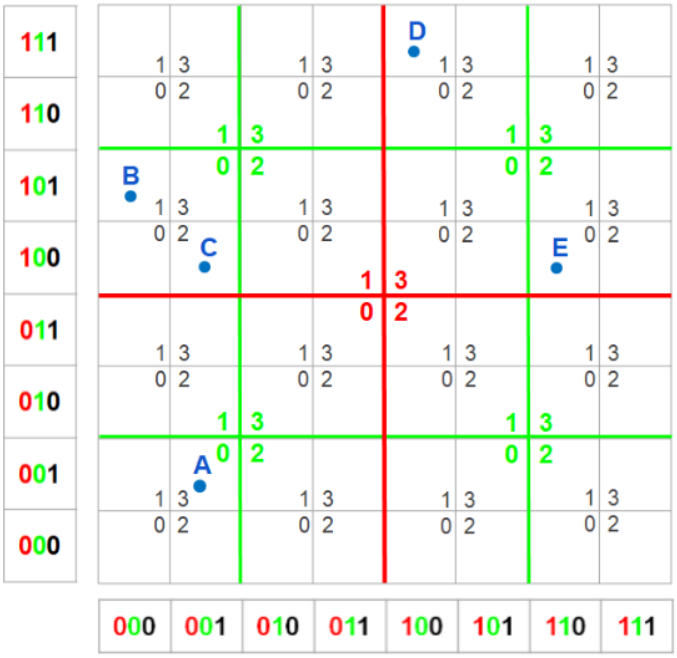


**Figura 5 : Representación en base-4 del árbol de nivel cuatro.**

En el ejemplo que nos concierne las posiciones de los cuerpos A, B, C, D y E en base-2 y base-4 son:

A: (001,001) ⇨ 003

B: (000,101) ⇨ 101

C: (001,100) ⇨ 102

D: (100,111) ⇨ 311

E: (110,100) ⇨ 320

A continuación vamos a reproducir este mismo ejemplo en forma de árbol piramidal en la figura 6. Puede parecer redundante, pero nos ayudará a comprender cómo para combinarse las celdas no vacías en celdas de tamaño mayor deberán perder el dígito menos significativo (el último de la derecha) de sus respectivos códigos de celda. El convenio establecido es que el relleno de color amarillo indicará que dicho nodo está habitado únicamente por un cuerpo, el relleno de color morado, que ese nodo está habitado por dos o más cuerpos y la falta de relleno querrá decir que el respectivo nodo se encuentra vacio. Una vez finalizada la clasificación de los cuerpos en el árbol los nodos vacios serán eliminados. Recordamos que las celdas que contienen más de un cuerpo guardan la información de la masa total de esos cuerpos, de la posición de su centro de masas y punteros a sus celdas hijas.

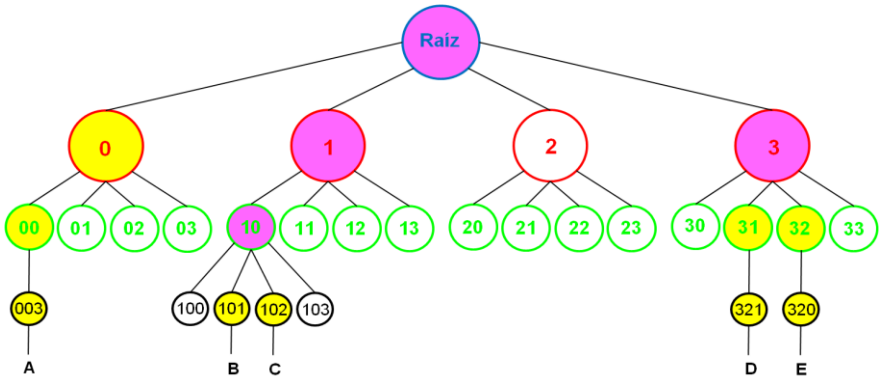


**Figura 6: Representación piramidal del ejemplo de árbol binario de nivel cuatro**

## Espacio tridimensional (base-8)

Esta clasificación es totalmente equivalente a la realizada en el espacio bidimensional en base-4. Para poder generalizar dicha clasificación a un espacio tridimensional debemos añadir al código de cada celda el componente en base-2 correspondiente al eje Z. Esta vez, al concatenar los tres dígitos en base-2 de los ejes X, Y y Z correspondientes a cada nivel del árbol, obtendremos un número de tres

dígitos en base-2. De forma semejante a la traducción de base-2 a base-4, un número de tres cifras en base-2 equivale a un número de una sola cifra en base-8. A fin de que la división del espacio siga una lógica se hará acorde al esquema de la figura 7.

| Base-2 | | | Base-4 |
|---|---|---|---|
| x | y | z | |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

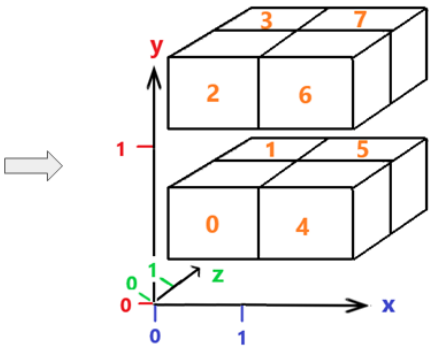


**Figura 7: Esquema de la división del espacio en base-8**

# 4. Comparación de algoritmos

En este apartado vamos a realizar una comparación de los algoritmos de cálculo de fuerza bruta (FB) y Barnes-Hut (BH). Este análisis no lo llevamos a cabo de forma teórica sino práctica mediante el examen de un conjunto pequeño pero representativo de ejemplos bien escogidos que esperamos capturen las diferencias más significativas entre ambos métodos. Debe quedar claro que en realidad analizaremos solamente la eficacia de nuestras propias implementaciones de ambos algoritmos.

Para ello, evaluaremos el número total de veces que cada algoritmo calcula la fuerza de atracción entre cuerpos y el tiempo que es empleado en este proceso. Para poder comparar los resultados obtenidos, la inicialización del sistema en ambos algoritmos (número de cuerpos, sus posiciones y tamaño del espacio) será idéntica (leída de un fichero) así como el número de veces que iteraremos el sistema. Todas las inicializaciones serán creadas aleatoriamente.

Estudiaremos el funcionamiento de los algoritmos para los casos siguientes:

1. Influencia de $theta$ $(\theta)$ (parámetro umbral) en el algoritmo BH.
2. Número de veces que evalúa cada algoritmo la fuerza de interacción gravitatoria.
3. El tiempo de ejecución de cada algoritmo.
4. Proporcionalidad entre el número de iteraciones y fuerzas evaluadas.
5. Configuraciones de cuerpos que favorecen el uso del algoritmo BH sobre FB.

Las características del ordenador en el que ejecutamos los programas son:

- Sistema operativo: Windows 10 Home 64-bit
- Procesador: Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz (4 CPUs), ~2.0GHz
- Memoria: 8192MB RAM

## 4.1. Influencia del parámetro theta

Empezaremos estudiando la influencia que tiene el parámetro umbral $\theta$ en el comportamiento del algoritmo BH. El algoritmo concede la libertad de fijar $\theta$ en cualquier valor comprendido entre $0$ y $1$ $(0 < \theta < 1)$ y este deberá ser ajustado tras haberse realizado un balance de la precisión y rapidez del algoritmo deseadas. Cuanto más próximo a $0$ sea el valor de $\theta$ más pares de fuerzas de interacción gravitatorias evaluaremos (agrupamos menos cuerpos en cúmulos para el cálculo) y por lo tanto, más preciso será el cálculo de estas pero más lenta será su ejecución. Si por el contrario, aproximamos el valor de $\theta$ a $1$, perderemos precisión de cálculo (consideramos más agrupaciones de cuerpos como

cuerpos individuales) pero ganaremos en tiempo de ejecución. Las siguientes figuras ilustran las ganancias en tiempo de ejecución y en número de evaluaciones de fuerza ("*Fuerzas calculadas*") del algoritmo BH de 3D a medida que empleamos valores de $\theta$ que tienden 1 en los casos en los que dejamos iterar 10 y 20 veces un sistema de $N = 100$ cuerpos.

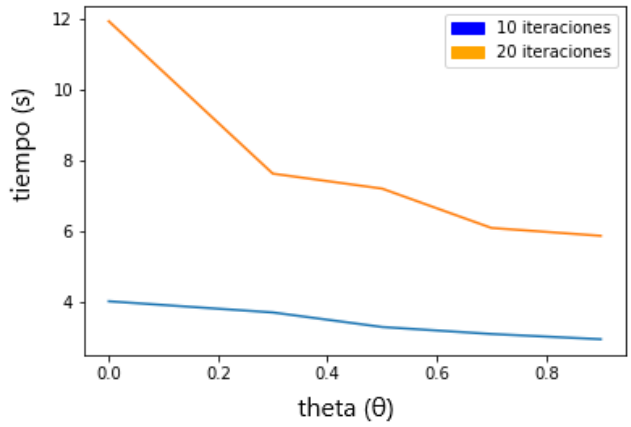


**Figura 8: Influencia de theta sobre el tiempo de ejecución**

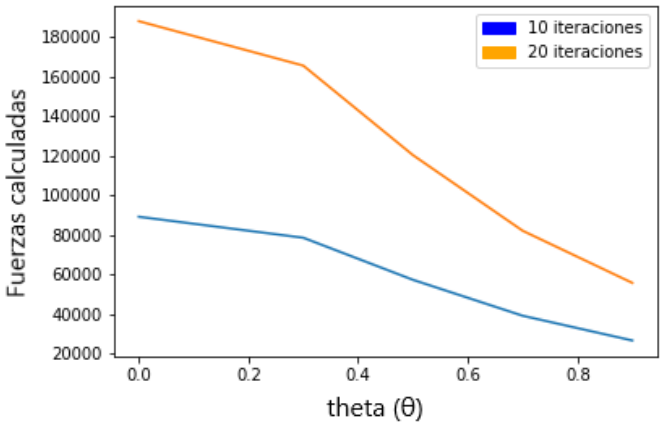


**Figura 9: Influencia de theta sobre el número de fuerzas evaluadas**

Tal y como esperábamos, el número de pares de fuerzas que evalúa el algoritmo decrece drásticamente a medida que $\theta$ crece y por lo tanto, también lo hace el tiempo de ejecución.

El objetivo del próximo estudio será determinar la pérdida de precisión que el aumento de $\theta$ trae consigo. Para ello, cabe recordar que cuando $\theta = 0$ la precisión del algoritmo es la misma que la del algoritmo FB, es decir, los valores de la fuerza ejercida sobre cada cuerpo que obtendremos serán exactos (apartado 3.4.2).

En la figura 10 cotejaremos los valores de las fuerzas (en Newtons) obtenidas al emplear el algoritmo BH para $\theta = 0$ (precisión de FB), con los valores de las fuerzas obtenidas para valores de $\theta$

superiores. Para llevar a cabo esta comparación, tendremos en cuenta únicamente la fuerza $\boldsymbol{F}_x$ total experimentada por un cuerpo concreto que pertenece a un sistema de $N = 500$ cuerpos en 3D. Con el fin de evitar el error que el algoritmo de integración Euler entrañaría, calculamos la fuerza para una única configuración de los cuerpos.

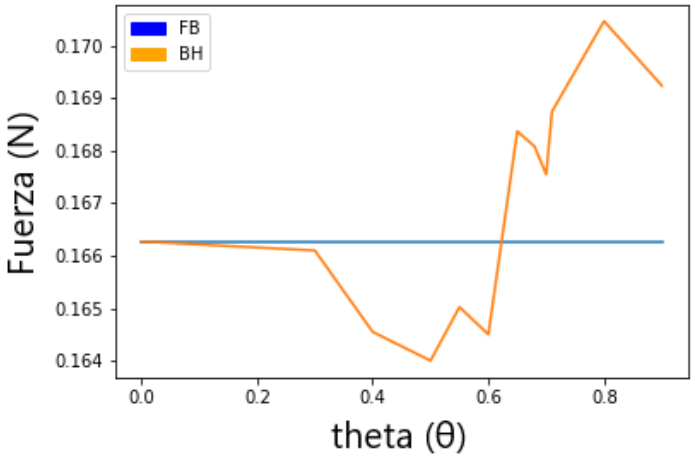


**Figura 10: Influencia de theta sobre la precisión del valor de la fuerza**

La figura 10 muestra como el valor de la fuerza calculada con BH para diferentes valores de $\theta$ se aleja del valor de la fuerza calculada con $\theta = 0$ que corresponde exactamente a la fuerza calculada con FB a medida que $\theta$ aumenta. Dado que el comportamiento oscilatorio de este fenómeno no permite ver esta tendencia con claridad, en la figura 11 ofreceremos un tratamiento diferente de estos mismos datos: calcularemos el valor absoluto de la diferencia. De esta forma, obtendremos el error absoluto (en Newtons) del cálculo de la fuerza y lo dibujaremos para diferentes $\theta$.

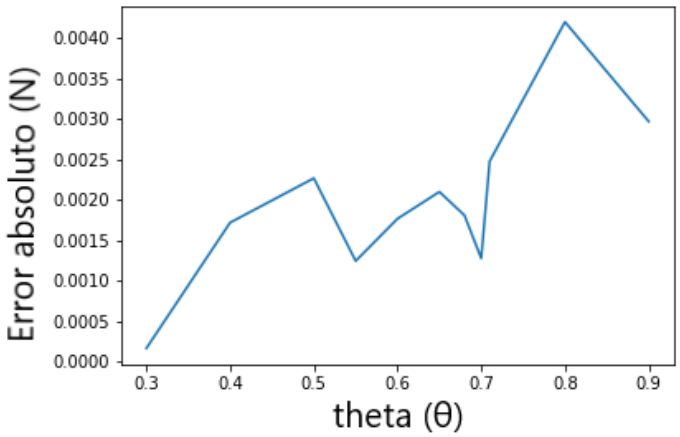


**Figura 11: Influencia de theta sobre la precisión del valor de la fuerza expresada en valor absoluto**

Es normal que el valor del error absoluto oscile ya que hemos empleado relativamente pocos datos para crear esta figura. Aún así, la tendencia creciente de la figura 11 es apreciable y esta indica que cuanto mayor es $\theta$ mayor será la imprecisión del cálculo, como habíamos predicho. Del mismo modo, en la figura 11 apreciamos que las diferencias de precisión que manejamos son muy pequeñas llegando a ser de un 2% en su punto más alejado.

Teniendo en cuenta que la precisión del valor de la fuerza apenas varía cuando $\theta$ aumenta y que las ganancias en número de evaluaciones de fuerza y por ende en tiempo son sustanciales estableceremos $\theta = 0.6$ para todos los análisis que haremos a partir de ahora.

## 4.2. Cálculo de las fuerzas

A continuación, contrastaremos el número de veces que los algoritmos BH y FB (de 3D) evalúan la fuerza. El algoritmo BH emplea el argumento $\theta = 0.6$ y dejaremos evolucionar dinámicamente el sistema dos pasos temporales, que es el mínimo requerido para inicializar el sistema. Además, el número de pasos de integración permitido en el análisis no alterará los resultados obtenidos mientras la distribución de los cuerpos del sistema siga siendo homogénea (apartado 4.4).

Como hemos apuntado previamente, las inicializaciones serán comunes para ambos algoritmos. Estas inicializaciones son aleatorias (generador de números pseudo aleatorios de Python) y emplearemos una nueva configuración cada vez que el número de cuerpos del sistema varíe.

La figura 12 muestra el crecimiento comparativo del número total de evaluaciones de la fuerza de atracción por número total de cuerpos N para ambos algoritmos. Se aprecia claramente que este crecimiento es de tipo cuadrático $\mathrm{O}(N^2)$ para FB mientras que para BH este crecimiento es aproximadamente logarítmico $\mathrm{O}(N \log N)$. Bajo este criterio, el algoritmo BH cumple perfectamente con sus expectativas y a partir de aproximadamente $N \approx 1000$ es claramente ventajoso frente a FB.

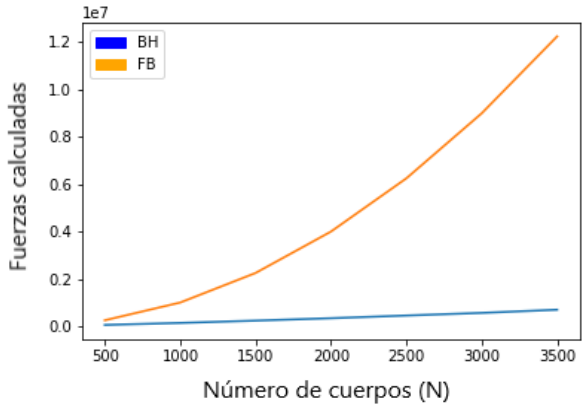


**Figura 12: Número de fuerzas evaluadas por número de cuerpos en 3D, $\theta = 0.6$**

Cuando hablamos de la fuerza de atracción experimentada por cada cuerpo, nos referimos en realidad a una fuerza vectorial que consta de tres (resp. dos) componentes en 3D (resp. 2D). Por lo tanto, cada computo de fuerza corresponde a tres (resp. dos) operaciones en 3D (resp. 2D). Tener en cuenta este factor es indispensable para poder comparar los algoritmos en 2D y 3D.

El algoritmo en 2D es de gran interés ya que las orbitas de los sistemas de cuerpos celestes tienden pasado el tiempo suficiente a alinearse en un mismo plano debido en parte a la necesidad de conservar el momento angular. Ejemplo de ello es el plano de la eclíptica en el sistema solar [28]. Al no tener la capacidad de cálculo computacional suficiente no simularemos este fenómeno. Sin embargo, mostraremos gráficamente la ventaja en cuanto a número de veces que evaluamos la fuerza del algoritmo BH sobre FB en 2D. La figura 13 muestra esta comparación repitiendo el mismo estudio realizado para obtener la figura 12 pero empleando los algoritmos FB y BH de 2D.

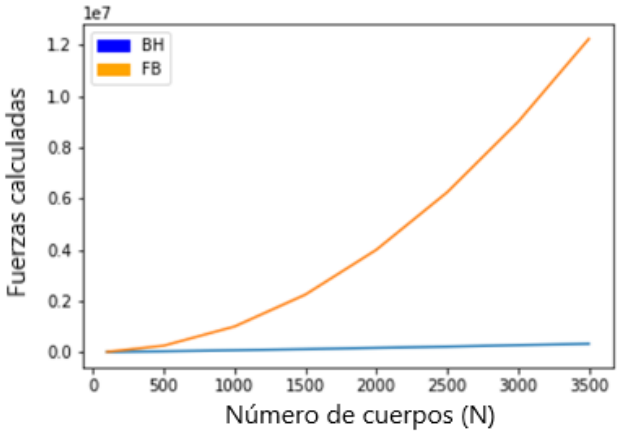


**Figura 13: Número de fuerzas evaluadas por número de cuerpos en 2D, $\theta = 0.6$**

El número total de evaluaciones de la fuerza de atracción del algoritmo FB en 2D sigue siendo $\frac{N(N-1)}{2}$. Por otro lado, en el algoritmo BH de 2D al encontrarse todos los cuerpos del sistema en un mismo plano, las distancias que los separan serán considerablemente menores en comparación a la disposición de un sistema del mismo número de cuerpos en 3D. Por ello, los cuerpos formarán agrupaciones a niveles de profundidad mayores del árbol (celdas de menor tamaño) y el coste computacional del cálculo de las fuerzas se verá reducido.

La representación grafica del número de veces que evaluamos la fuerza en el algoritmo BH de la figura 13 (representación 2D) tiene una pendiente menor (poco apreciable por la no suficientemente fina discretización del eje vertical) que la de la figura 12 (representación 3D), mientras que las curvas

que representan al algoritmo FB en ambas figuras (2D y 3D) son idénticas. La inicialización de las disposición de los cuerpos en el sistema en 2D es la misma que la de 3D exceptuando el eje Z. Anteriormente hemos mencionado que para comparar justamente los algoritmos de 2D y 3D hay que tener en cuenta cada computo de fuerza corresponde a tres operaciones en 3D y a dos operaciones en 2D. Esta comparación está recogida en la figura 14.

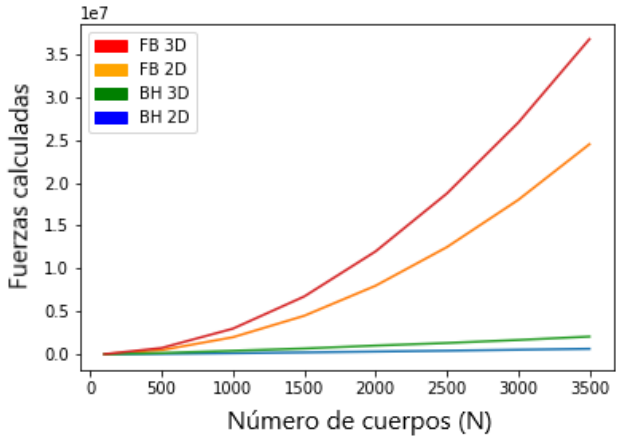


**Figura 14: Número de fuerzas evaluadas por número de cuerpos en 2D y 3D, $\theta = 0.6$**

### 4.3. Tiempo de ejecución

Procedemos a comparar el coste computacional de los algoritmos BH y FB. Es decir, el tiempo que tarda cada uno de estos algoritmos en calcular un mismo número de iteraciones de un sistema idéntico.

Cabe aclarar de antemano, que esta comparación no es del todo justa, debido a que el algoritmo FB ha sido implementado usando la librería NumPy [29] de Python. Esta librería está extremadamente optimizada ya que (i) permite que todos los cálculos estén vectorizados ahorrando así el coste de los ciclos y (ii) está parcialmente implementada en lenguajes compilados como C o FORTRAN [29]. El algoritmo BH por el contrario no permite fácilmente el uso de cálculos vectorizados dada la condición recursiva de la estructura árbol.

Para realizar esta comparativa emplearemos el mismo procedimiento que hemos empleado para obtener las figuras previas: crearemos inicializaciones aleatorias que dejaremos iterar dos veces.

A pesar de la considerable ventaja del algoritmo BH sobre FB obtenida al contabilizar en número de veces que la fuerza es evaluada, el coste computacional del algoritmo BH es superior al de FB. Esto es: a pesar de que con el algoritmo BH haya que calcular un número inferior de fuerzas el tiempo empleado para esta tarea es mayor al que emplea el algoritmo FB. La “lentitud” de nuestra

implementación del algoritmo BH es achacable a que la construcción su árbol es muy costosa computacionalmente.

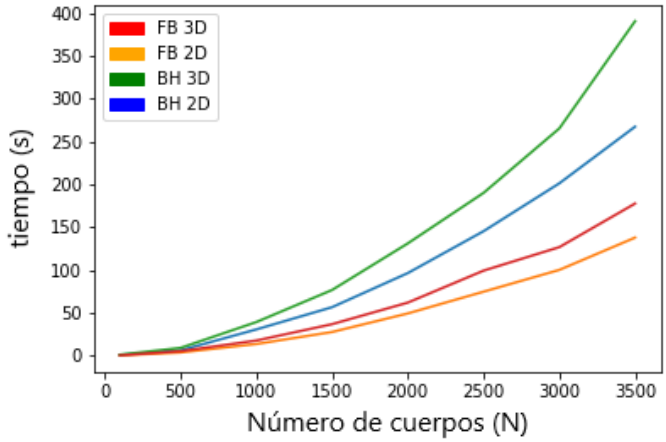


**Figura 15: Tiempo de ejecución por número de cuerpos en 2D y 3D, $\theta = 0.6$**

## 4.4. Proporcionalidad entre el número de iteraciones y fuerzas evaluadas

Los análisis previos han sido realizados dejando iterar (evolucionar dinámicamente) al sistema únicamente dos veces ya que el incremento del tiempo de ejecución y el número de evaluaciones de la fuerza en ambos algoritmos son directamente proporcionales al número de iteraciones del sistema mientras su configuración espacial se mantenga homogénea.

Para comprobar esto, tomaremos como ejemplo una inicialización aleatoria de $N = 100$ cuerpos en la que estudiaremos la influencia del número de iteraciones realizadas sobre el número de fuerzas computadas.

La figura 16 muestra que el número de veces que evaluamos la fuerza crece linealmente con el número de iteraciones del sistema.

Observando la figura 17 apreciamos que el tiempo de ejecución no es una medida precisa para N pequeños a pesar de que atisbamos una tendencia lineal. Además, variará dependiendo de la capacidad del procesador que lo ejecute.

Sin embargo, la figura 16 muestra que el número de veces que evaluamos la fuerza sí es un parámetro estable y crece linealmente con el número de iteraciones del sistema.

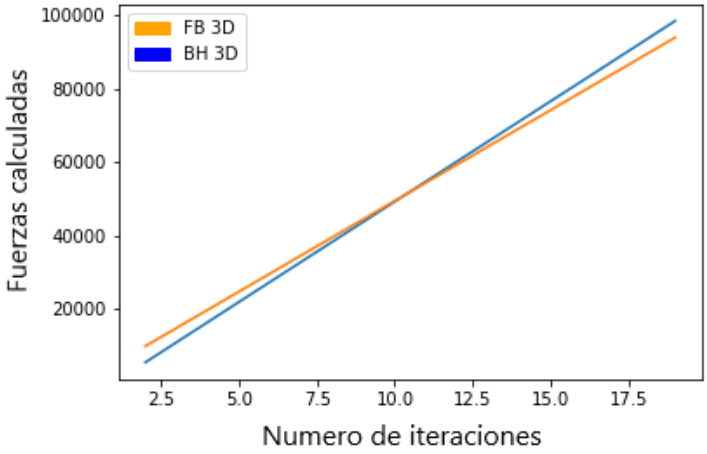


**Figura 16: Número de fuerzas evaluadas por número de iteraciones, $\theta = 0.6$**

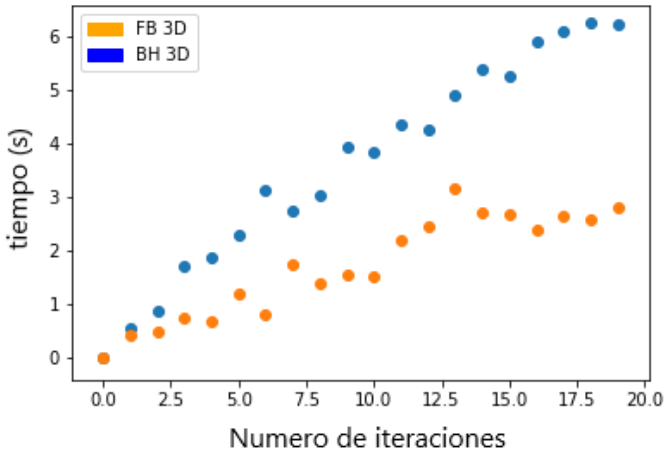


**Figura 17: Tiempo de ejecución por número de iteraciones, $\theta = 0.6$**

## 4.5. Configuraciones que favorecen el algoritmo Barnes-Hut

Para concluir esta comparativa de la eficacia de los algoritmos expondremos una configuración inicial en la que el número de evaluaciones de la fuerza ejecutadas por el algoritmo BH es todavía menor que en el caso de la inicialización aleatoria de este mismo algoritmo, lo cual supone a su vez una mejora temporal en el desempeño del algoritmo BH.

Esta configuración se basa en colocar cúmulos de cuerpos en los ocho vértices del cubo (estamos analizando en algoritmo BH de 3D). Tanto el número de cuerpos que hay en cada vértice como sus respectivas posiciones serán elegidos de forma aleatoria.

En todos los ejemplos que estudiaremos a continuación, el tamaño del espacio total (universo) será constante y de valor $(2^{13})^3$. Así mismo, el volumen de los espacios habilitados en cada vértice para contener cuerpos será $\left(\frac{2^{13}}{8}\right)^3$. Como de costumbre, iteraremos el sistema dos veces en cada caso e iremos incrementando el número de cuerpos iniciales que hay en el sistema.

Por lo tanto, en la figura 18 comparamos el número de evaluaciones de la fuerza de atracción que necesitamos realizar cuando la disposición inicial de los cuerpos es totalmente aleatoria (homogénea en el espacio) y cuando está dispuesta en ocho cúmulos de cuerpos.

Esta configuración inicial formada por cúmulos de cuerpos distanciados entre sí no es un caso meramente ficticio: Nuestro universo está formado por cúmulos de cuerpos celestes alejados entre sí. Debido a la fuerza gravitatoria, los sistemas de cuerpos celestes evolucionan agrupándose entre sí. Una vez más, no disponemos de la capacidad de cálculo suficiente para dejar al sistema evolucionar hasta formar este tipo de agrupaciones. Aun así, podemos afirmar que al modelizar un sistema que represente el universo el algoritmo BH tendrá una ganancia mayor que la obtenida en las figuras 12, 13 y 14 respecto al algoritmo FB.

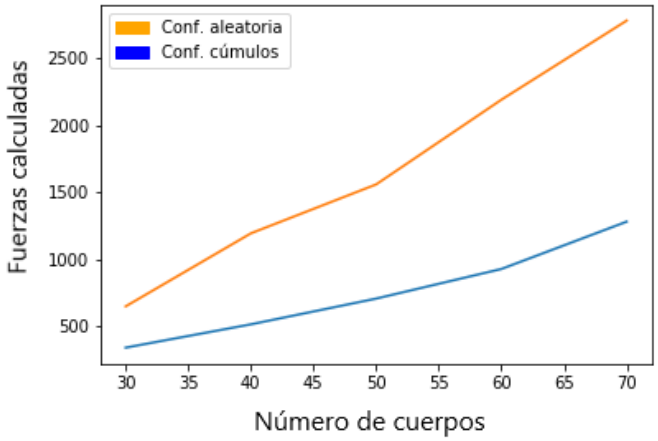


**Figura 18: Número de evaluación de fuerzas que requiere una configuración inicial homogénea frente a una dispuesta en ocho cúmulos de cuerpos.**

# 5. Conclusiones

La realización de este trabajo ha requerido de cuatro etapas:

La primera ha sido realizar un estudio previo sobre las estructuras de datos de tipo árbol y la programación recursiva de las mismas.

La segunda, desarrollar una visión personal del algoritmo BH e idear una implementación original del mismo.

La tercera, crear desde cero un programa que lo implementa y que ha demostrado ser funcionalmente robusto.

En cuarto y último lugar, realizar un estudio detallado de las propiedades del mismo así como un análisis comparativo con el algoritmo FB.

A continuación comentaremos las principales dificultades encontradas en estas etapas:

El algoritmo es engañosamente simple ya que es realmente difícil de implementar, habiéndole dedicado únicamente a la programación más de cien horas.

Python, el lenguaje elegido para la programación, es sencillo y proporciona gran cantidad de librerías pero no es competitivo en cálculo numérico. Por ello, la implementación conseguida aunque válida no es óptima. En este sentido, cabe recordar que nuestro algoritmo BH bate claramente al algoritmo FB en cuanto a evaluaciones de fuerza.

## Valoraciones personales y perspectivas de futuro

Estudiar e implementar el algoritmo BH ha sido muy instructivo, pero veamos aspectos prácticos: si estimamos como de grande debe ser N antes de que la diferencia entre $\frac{N(N-1)}{2}$ y $N\log(N)$ sea lo suficientemente sustancial como para compensar la diferencia de tiempo que conlleva programar ambos algoritmos, encontraremos que el algoritmo FB no está en peligro de volverse obsoleto. Entonces, ¿merece la pena implementar este algoritmo? Por supuesto que sí, pero a nuestro prototipo todavía le quedan unas cuantas mejoras para ser práctico; aquí enumeramos unas cuantas sugerencias que sería interesante explorar en un futuro.

Para empezar, podríamos analizar los cuellos de botella mediante un programa que nos dé el perfil de ejecución (profiler). Las partes de código que se ejecuten intensivamente podrían programarse como llamadas a funciones escritas en un lenguaje de programación compilado (FORTRAN por ejemplo). Asimismo, más adelante podríamos plantearnos el procesado paralelo, la traducción total a

Java, ejecutar el programa en PyPy (compilador de Python optimizado) [24], llevar el código a ejecución GPU, etc.

Por otro lado, el método de integración empleado (Euler) no conserva la energía ni el momento angular. Una mejora considerable sería emplear un algoritmo de integración que garantizase (en la medida de lo posible) la conservación numérica de estas cantidades teóricamente conservadas. Este avance contraerá una importancia capital si aspiramos a que el modelo sea capaz de reproducir los resultados experimentales. Podemos mencionar en este sentido los recientes avances en el desarrollo de integradores simplécticos [23].

En una versión inicial de la implementación del algoritmo, empleamos un espacio continuo (con condiciones de frontera periódicas de tipo Born-von Karman [30]), pero tales condiciones provocaban discontinuidades en el cálculo de la fuerza y tuvo que ser descartada. Queda por lo tanto pendiente analizar la diferencia que supondría emplear un espacio de estas características siempre y cuando consiguiésemos evitar estas discontinuidades.

Como última valoración me gustaría añadir que siendo el problema de N-cuerpos una cuestión tan popular en la comunidad física, me sorprende que este algoritmo que conlleva una mejora de cálculo tan sustancial sea un gran desconocido. Véase el caso de su famosísimo análogo FFT, que es de orden $\mathrm{O}(N\log(N))$ frente a la transformada de Fourier discreta de orden $\mathrm{O}(N^2)$[22]. Entiendo que este desconocimiento se debe a la complejidad inherente de su programación recursiva pero invita a la siguiente reflexión: ¿existirán otras cuestiones a las que este enfoque sea aplicable y no se haga?

# ANEXOS

## Anexo I

## Análisis dimensional

El objetivo de este análisis es poder definir unas unidades normalizadas ("por unidad") que estén adaptadas al problema tratado en cada caso. Mediante una adecuada definición, tales sistemas de unidades normalizadas adaptadas ofrecen las siguientes considerables ventajas:

1. Todas las variables del problema varían en rangos mucho más acotados, lo cual es muy ventajoso desde el punto de vista numérico por ganar en estabilidad y precisión. A este respecto son bien conocidas las limitaciones de las representaciones finitas de números en coma flotante en la máquina. Por ejemplo, la sustracción de dos números reales de gran magnitud es inherentemente imprecisa; también lo son las operaciones básicas con números de muy diferente magnitud.
2. Ventajas conceptuales: por un lado, el manejo de ecuaciones adimensionalizadas permite absorber las constantes arbitrarias de manera que estas toman el valor unidad, que es el más sencillo posible. Por otro lado, los razonamientos y cálculos que se hacen son aplicables inmediatamente a familias enteras de sistemas, sin más que adaptar unas pocas unidades de base. Es decir, mediante este "truco" numérico ahora estamos en condiciones de comparar sistemas muy diferentes que de otro modo serían incomparables.

| | Variables comunes | Nombre | dimensiones | Unidad S.I. |
|---|---|---|---|---|
| Unidades fundamentales | $d, r, x, y, z$ | Longitud, posición, distancia | $L$ | M |
| | $M, m$ | Masa (inercial y gravitacional) | $M$ | Kg |
| | $T, t, \delta t, \frac{2\pi}{\omega}$ | Tiempo, periodo | $T$ | S |
| Unidades derivadas | $v$, c | Velocidad | $V = \frac{L}{T}$ | $\frac{\mathrm{m}}{\mathrm{s}}$ |
| | $a$ | Aceleración | $a = \frac{L}{T^2}$ | $\frac{\mathrm{m}}{\mathrm{s}^2}$ |
| | $F, f$ | Fuerza | $F = \frac{M \cdot L}{T^2}$ | N |
| | $E_c, E_p$ | Energía | $E = \frac{M \cdot L^2}{T^2}$ | J |
| | $J_z$ | Momento angular | $J_z = \frac{M \cdot L^2}{T}$ | $\frac{\mathrm{kg} \cdot \mathrm{m}^2}{\mathrm{s}}$ |

Si elegimos arbitrariamente unas magnitudes de base $\{M_o, L_o, T_o\}$, resulta evidente a la vista de la tabla que para las magnitudes derivadas también pueden definirse magnitudes de base dependientes de las unidades básicas del siguiente modo:

| Velocidad | Aceleración | Fuerza | Energía | Momento angular |
|---|---|---|---|---|
| $V_o = \frac{L_o}{T_o}$ | $a_o = \frac{L_o}{T_o^2}$ | $F_o = \frac{M_o \cdot L_o}{T_o^2}$ | $E_o = \frac{M_o \cdot L_o^2}{T_o^2}$ | $J_z^o = \frac{M_o \cdot L_o^2}{T_o}$ |

Ahora definimos el siguiente convenio notacional: dadas una magnitud $X$, y su magnitud de base correspondiente $X_o$ escribiremos:

$$X_{pu} = \frac{X}{X_o} \Leftrightarrow X = X_{pu} \cdot X_o$$

($pu$, del inglés per unit). Procedamos entonces a adimensionalizar (es decir, a rescribir de forma adimensional) las ecuaciones de Newton de mecánica clásica del problema de interacción gravitatoria (adimensionalizamos las variables y absorbemos las constantes):

$$F = G\,\frac{m \cdot m'}{d^2} = m \cdot a \Leftrightarrow F_{pu} \cdot F_o = G\,\frac{m_{pu} \cdot m'_{pu} \cdot M_o^2}{d_{pu}^2 \cdot L_o^2} = m_{pu} \cdot a_{pu} \cdot M_o \cdot a_o$$

Entonces, la primera parte nos queda:

$$F_{pu} = \left(\frac{G \cdot M_o^2}{F_o \cdot L_o^2}\right) \frac{m_{pu} \cdot m'_{pu}}{d_{pu}^2} = \left(\frac{G \cdot T_o^2 \cdot M_o}{L_o^3}\right) \frac{m_{pu} \cdot m'_{pu}}{d_{pu}^2}$$

Y la segunda:

$$F_{pu} = \left(\frac{M_o \cdot a_o}{F_o}\right) \cdot m_{pu} \cdot a_{pu} = m_{pu} \cdot a_{pu}$$

Esto quiere decir que si en vez de definir $M_o$, $L_o$ y $T_o$ independientemente, imponemos la ecuación adicional

$$\frac{G \cdot T_o^2 \cdot M_o}{L_o^3} = 1$$

podemos hacer "desaparecer" la constante gravitatoria de las ecuaciones. Es decir que trabajaríamos en un sistema de unidades en el que $G^{pu} = 1\ (pu)$ [25]. Por esta razón nosotros tomamos $M_o$ y $L_o$ como magnitudes de base fundamentales (arbitrarias) y obtenemos $T_o$ a partir de ellas:

$$T_o \equiv \sqrt{\frac{L_o^3}{G \cdot M_o}}$$

Por lo que, en adelante, podremos escribir en unidades "naturales":

$$F = \frac{m \cdot m'}{d^2} = m \cdot a$$

Estudiaremos como ejemplo el sistema Sol-Tierra:

Datos físicos

| | | |
|---|---|---|
| Constante de gravitación universal | | $G = 6.6743 \cdot 10^{-11} \frac{\mathrm{m}^3}{\mathrm{kg} \cdot \mathrm{s}^2}$ |
| Velocidad de la luz en el vacío | | $c = 2.99792458 \cdot 10^8 \frac{\mathrm{m}}{\mathrm{s}}$ |
| | Masa | Radio |
| Sol | $M_\odot = 1.989 \cdot 10^{30}\ \mathrm{kg}$ | $R_\odot = 6.964 \cdot 10^8\ \mathrm{m}$ |
| Tierra | $M_\oplus = 5.97 \cdot 10^{24}\ \mathrm{kg}$ | $R_\oplus = 6.371 \cdot 10^6\ \mathrm{m}$ |
| Distancia Sol-Tierra | $d_{\odot\oplus} = 149.6 \cdot 10^9\ \mathrm{m}$ | |

| | | |
|---|---|---|
| Magnitudes de base fundamentales: | $M_o = M_\odot$ | $L_o = R_\odot$ |
| Variables por unidad: | $M_\odot^{pu} = 1\ (pu)$ | $R_\odot^{pu} = 1\ (pu)$ |
| A partir de estos datos (elegidos arbitrariamente) el resto de variables y sus interpretaciones : | | |
| $T_o = \sqrt{\frac{R_\odot^3}{G \cdot M_\odot}} = \sqrt{\frac{R_\odot}{g_\odot}} = 1594.9\ \mathrm{s} \approx 26.9\ \mathrm{min}$ | $\frac{1}{2\pi}$ del periodo de un satelite ecuatorial en vuelo rasante alrededor del sol. En estas unidades $T_{sat}^{pu} = 2\pi\sqrt{\frac{1}{1}} = 6.28\ (pu)$ | |
| $V_o = \frac{L_o}{T_o} = \sqrt{g_\odot R_\odot} = 4.366 \cdot 10^5 \frac{\mathrm{m}}{\mathrm{s}}$ | $\frac{1}{2\pi}$ de la velocidad de dicho satélite | |
| $a_o = \frac{V_o}{T_o} = \frac{G \cdot M_\odot}{R_\odot^2} = g_\odot = 273.8 \frac{\mathrm{m}}{\mathrm{s}^2}$ | Número interpretable como la aceleración de la gravedad en la superficie del sol | |
| $c^{pu} = \frac{c}{V_o} = 686.6\ (pu)$ | Velocidad máxima permitida físicamente | |
| $M_\oplus^{pu} = \frac{M_\oplus}{M_o} = 3 \cdot 10^{-6}\ (pu)$ | Masa de la tierra | |
| $d_{\odot\oplus}^{pu} = \frac{d_{\odot\oplus}}{L_o} = 214.8\ (pu)$ | Distancia sol-tierra | |
| Con este último dato podemos calcular la aceleración de la tierra hacia el sol: $a_{\odot\oplus}^n = \frac{M_\odot^{pu}}{\left(d_{\odot\oplus}^{pu}\right)^2} = \frac{1}{214.8^2} \approx 2.17 \cdot 10^{-5}\ (pu) \approx 0.0059 \frac{\mathrm{m}}{\mathrm{s}^2}$ | | |
| Si suponemos que la tierra describe una órbita circular entorno al sol, esta aceleración es centrípeta y podemos calcular su velocidad orbital: $a_{\odot\oplus}^t = \frac{\left(V_{\odot\oplus}^{pu}\right)^2}{d_{\odot\oplus}^{pu}} = \frac{M_\odot^{pu}}{\left(d_{\odot\oplus}^{pu}\right)^2} \Rightarrow V_{\odot\oplus}^{pu} = \frac{1}{\sqrt{d_{\odot\oplus}^{pu}}} = 0.0682\ (pu) \approx 29.8 \frac{\mathrm{km}}{\mathrm{s}}$ También podemos obtener el valor del periodo de su órbita salvo por una fracción $\frac{1}{2\pi}$ (recordemos | | |

que el análisis dimensional no tiene en cuenta los factores numéricos constantes):

$$T = 2\pi \frac{d_{\odot\oplus}^{pu}}{V} = 2\pi \left(d_{\odot\oplus}^{pu}\right)^{3/2} \approx 19780\ (pu) \approx 365 \text{ días}$$

Esta ecuación expresa de forma muy elegante la 3ª ley de Kepler.

Estas cifras concuerdan perfectamente con datos astronómicos bien conocidos.

## Adaptación a nuestra implementación

Esta correspondencia entre las magnitudes por unidad empleadas en la implementación y las magnitudes que representan los sistemas reales es general y válida para interpretar cualquier sistema. Queda claro que para obtener el valor real de una variable expresada en el sistema por unidad sólo hay que multiplicarla por las magnitudes de base correspondientes. Expondremos a continuación un ejemplo de esta correspondencia tomando como referencia el sistema solar.

Nuestro objetivo es representar un sistema real que comprende masas y distancias acotadas entre los valores

$$10^{22}\ \text{kg} < M < 10^{31}\ \text{kg}$$

$$10^{8}\ \text{m} < L < 10^{13}\ \text{m}$$

Podemos elegir como magnitudes de base

$$M_o = 10^{23}\ \text{kg}, L_o = 10^{8}\ \text{m}$$

Con lo cual para representar las masas reales y las distancias reales emplearíamos los valores por unidad

$$0.1\ < M^{pu} < 10^{8}$$

$$1\ < L^{pu} < 10^{5}$$

En este caso, la discretización mínima del espacio $1\ (pu)$ representaría una distancia de $L_o = 10^8$ m (orden de magnitud del diámetro de la Tierra) y un cuerpo de masa $10\ (pu)$ correspondería a una masa real $M = 10^{24}$ kg (orden de magnitud de la masa terrestre). Así mismo, la distancia $10^{13}$m, que hemos escogido como cota superior, es del orden de magnitud del diámetro del sistema solar. Su representación en el sistema por unidad es $10^5\ (pu)$, distancia que obtendremos empleando un árbol de cómo mínimo $A = 17$ niveles, ya que en ese caso la longitud de cada eje será $2^{17} \sim 1.5 \times 10^5$.

Por otro lado, empleando las formulas del apartado anterior podemos determinar que la magnitud de base temporal correspondiente sería

$$T_o \equiv \sqrt{\frac{L_o^3}{G \cdot M_o}} = 387201.5\ \text{s} = 107.6\ \text{h}$$

En este caso, si el valor del paso temporal elegido para la integración numérica es

$$T^{pu} = 0.1$$

Su equivalente en el sistema real sería

$$T = T^{pu} \times T_o = 10.76\,\mathrm{h}$$

Por lo tanto, cada paso temporal simularía una fracción $\sim\frac{1}{800}$ del periodo de la órbita terrestre alrededor del Sol.

Con una elección diferente de las magnitudes de base, exactamente el mismo cálculo describe un sistema físico diferente.

# Anexo II

## Implementación del algoritmo en Python 3.7

Este documento trata de explicar mi implementación del algoritmo Barnes-Hut (BH). El código fuente podrá encontrarse en la siguiente dirección: https://github.com/maiaagi/prototipoBH. La implementación ha sido escrita en su totalidad en el lenguaje de programación Python 3.7 tanto en dos (2D) como en tres dimensiones (3D). Para que este documento tenga pleno sentido debe ser leído a la vez que los scripts que implementan el algoritmo ya que ha sido concebido de modo que facilite la lectura y comprensión del código sin entrar en explicaciones detalladas del mismo.

La implementación inicial del algoritmo BH la realicé en 2D de una forma "agnóstica de la dimensión" de manera tal que su generalización a 3D fue inmediata. En la Tabla I se recogen los nombres de los ficheros fuente utilizados:

**Tabla 1: ficheros fuente del algoritmo BH en 2D/3D**

| Dimensión | Main | Módulos con funciones auxiliares |
|---|---|---|
| 2D | `Main_2D` | `functions_2D` |
| 3D | `Main_3D` | `functions_3D` |

Además, el fichero `tree_grapher` contiene código que utiliza funciones de la librería graphviz que partiendo de una estructura de datos de tipo árbol genera recursivamente una representación gráfica del mismo.

Los algoritmos Barnes-Hut en 2D y 3D tienen la misma estructura y son prácticamente idénticos salvo por las siguientes diferencias:

1. El algoritmo en dimensión 3 lógicamente maneja una tercera coordenada espacial (la llamada componente "Z").
2. La denominación de las celdas del espacio utiliza números en base 4 en 2D y en base 8 en 3D.

Realmente el cambio de dimensión solamente afecta a unas pocas funciones de los módulos y hay unas mínimas correcciones en los programas principales (Main). Por esta razón resulta conveniente explicar conjuntamente la estructura de ambos algoritmos indicando las diferencias cuando sea necesario. Esto ocurre así porque desde un principio hemos diseñado el programa para que, en la medida de lo posible, sea independiente de la dimensión.

La función `tree_grapher` está explicada al final de este documento.

## Notas previas

Cuando hablamos de cuerpos individuales en el sistema nos referimos a cuerpos formados por una sola “estrella” o “planeta”. El sistema o “universo” se inicializa con N cuerpos individuales.

En Python, un índice que cuente tales cuerpos comenzará en 0 e irá hasta N-1.

Por hipótesis, cuando la evolución dinámica haga que dos cuerpos se encuentren en la misma posición espacial diremos que han chocado. Las colisiones admisibles son completamente inelásticas, por lo que se fusionarán y convertirán en un nuevo cuerpo que consideraremos individual. La masa total del nuevo cuerpo corresponde a la suma de las masas que han colisionado, por lo que la masa total de los cuerpos es una cantidad conservada. Sin embargo, el número de cuerpos decrecerá en una unidad y no se conserva.

El espacio en el que se encuentran los cuerpos está discretizado en celdas cuadradas del mismo tamaño en 2D; correlativamente, en 3D el espacio está discretizado en celdas cúbicas del mismo tamaño.

En nuestro algoritmo el espacio será discretizado recursivamente en un número creciente de celdas de tamaños cada vez más pequeños. La celda que corresponde a la discretización de tamaño mínimo corresponde a un cuerpo individual en caso de estar ocupada. El tamaño del cuerpo equivale al tamaño de estas celdas y cada una de ellas solamente puede contener un cuerpo individual. Todos los cuerpos ocupan el mismo volumen en el espacio independientemente de la masa de cada uno. Lógicamente la densidad másica de cada cuerpo podrá ser diferente.

Los cúmulos de cuerpos o “clusters” son agrupaciones ficticias de cuerpos que se encuentran en la misma celda o nodo, obviamente estas celdas no serán las de tamaño mínimo. Estos clusters son artefactos de cálculo requeridos por el algoritmo BH.

Llamamos niveles a las posibles discretizaciones del espacio (siempre que sean cuadradas en 2D o cúbicas en 3D y todas las celdas sean del mismo tamaño). La discretización correspondiente al número máximo de celdas permitidas (celdas del tamaño de los cuerpos individuales) es la de nivel mayor. Recursivamente se irán combinando estas celdas de tamaño menor y nivel mayor y se irán formando celdas de mayor superficie que corresponderán a niveles inferiores. El menor nivel o nivel cero es aquel que comprende el espacio entero en una sola celda (“El Universo”). Asimismo, el nivel uno es aquel que cuenta con cuatro (resp. ocho) celdas del mismo tamaño en 2D (resp. 3D) y así sucesivamente (ver figura 2).

## Parámetros globales (en Main)

**N:** Número inicial de cuerpos. Se trata de un número natural que se establece una vez, al inicio del cálculo y no varía.

**A:** Número natural (entero positivo) que establece el número máximo de subdivisiones del espacio. En concreto, el espacio 2D (resp. 3D) está subdividido recursivamente en 4 (resp. 8) celdas. El parámetro "A" indica el número máximo de divisiones en 4 (resp. 8) que se consideran en el cálculo. Cada vez que dividimos nuestras celdas en 4 (resp. 8) creamos un nivel mayor (con celdas de tamaño menor). Por lo tanto, el nivel mayor "A" corresponde a las celdas de tamaño menor que comprenden la discretización del espacio siendo el número de nodos de tamaño mínimo $(2^A)^2$ en 2D y $(2^A)^3$ en 3D.

Dicho de otro modo, si $N$ es el número total de celdas distinguibles y $d$ es la dimensión del espacio, entonces $A \approx \frac{\log_2 N}{d} = \log_2 \sqrt[d]{N}$

**theta:** Parámetro ajustable – *fixed accuracy parameter* [18] – . Criterio umbral para determinar si un nodo (correspondiente a todo un conjunto de celdas) está lo suficientemente alejado de un cuerpo concreto y de ser así proceder al cálculo de la fuerza entre ambos. Este parámetro está descrito en el algoritmo Barnes-Hut original.

**zmax, ymax, xmax:** Estos parámetros miden la discretización del espacio por eje; en general tomamos $x_{\max} = y_{\max} = z_{\max} = 2^A - 1$, así como $x_{\min} = y_{\min} = z_{\min} = 0$.

**count:** Contador del número de cuerpos individuales que hay en el sistema. Empieza siendo "N" y disminuye a medida que los cuerpos chocan y se fusionan.

**P:** Diccionario (estructura de datos con llaves "hasheables") ordenado de Python. Se inicializa con la información de los "N" cuerpos iniciales (situados en los nodos del nivel "A"). A medida que creamos la estructura árbol se le añade la información de los nodos superiores.

Información que contiene "P" para cada cuerpo/cúmulo de cuerpos:

"X": posición en eje x "Y": posición en eje y "Z": posición en eje z (solo en 3D)

"VX": velocidad en eje x "VY": velocidad en eje y "VZ": velocidad en eje z (solo en 3D)

"M": masa del cuerpo "cell": código de celda en la que se encuentra en base-4 (2D) o base-8 (3D)

"S": número de cuerpos que contiene. Para los N primeros cuerpos S=1.

"I": Número de identificación tanto de los cuerpos físicos actuales como de los cúmulos de cuerpos ficticios empleados en el cálculo. Se trata de un número natural con el que etiquetamos los cuerpos o

cúmulos asignando a cada uno de ellos un número de identificación único. Los primeros "N" cuerpos individuales que creamos irán etiquetados del 0 al N-1. Las agrupaciones de cuerpos en celdas de mayor tamaño se etiquetarán a partir del número "N". Aunque ligeramente contraintutitiva, esta convención de contar empezando desde 0 es totalmente natural en Python.

"ancs": "Ancestros". Guarda los "I" de los cúmulos de cuerpos que han sido combinados para formar el actual.

"nodes": Guarda los "I" de los "N" cuerpos iniciales que contenga dicho cuerpo/cúmulo.

**P2:** Diccionario ordenado. Este segundo diccionario auxiliar "P2" nos sirve de apoyo para "P" ya que no es posible actualizar/modificar los diccionarios de Python a la vez que los recorremos. Por lo tanto, hacemos una copia profunda `deepcopy()` del diccionario original que de este modo no se verá afectado y modificamos la copia.

**T:** Árbol. Estructura de datos jerárquica a base de diccionarios que contienen otros diccionarios.

Estructura de datos de Python utilizada: diccionario vivificado. Implementamos el método `_missing_` de la clase diccionario necesario para la auto-vivificación. [26]

Cada diccionario anidado tiene la siguiente estructura:

Llave: Etiqueta "I" del cuerpo/cúmulo que es.

Valores:

1. Nivel de profundidad al que pertenece en el árbol "level"
2. Lista de los nodos primarios (de 0 a "N-1") que cuelgan de este cuerpo/cúmulo: "nodes"
3. Tantos diccionarios como ancestros tenga, como máximo 4 en 2D y 8 en 3D.

Ejemplo de un fragmento "T" de un sistema que inicialmente tenía 5 cuerpos (0-4) y tres niveles (0-2):

```
{    "8": {"level": 0,"nodes": [ 2,0,3,4,1],
            "7": {"level": 1,"nodes": [4,1],
                        "4": {"level": 2,"nodes": [4],
                         "1": {"level": 2,"nodes": [1]}},
            "5": {"level": 1,"nodes": [5]},        ...}
```

**L:** Diccionario. Las llaves son el código "I" del cuerpo/cúmulo y los valores el tamaño del nodo en el que se encuentra dicho cuerpo cúmulo: $2^{A-level}$

**I_max:** Índice. El mayor "I". Etiqueta que pertenece al cúmulo de cuerpos del nivel 0, es decir, al cúmulo de la celda de tamaño mayor que es a su vez es el cúmulo de todos los cuerpos del sistema.

**position:** Lista. Contiene las parejas ($r_x$, $r_y$) en 2D o tripletes ($r_x$, $r_y$, $r_z$) en 3D de posiciones de todos los cuerpos individuales (las posiciones de los centros de masas de los cúmulos no nos interesan) después de cada iteración. Lo utilizamos únicamente para plotear.

**cells:** Diccionario. Contiene como llave la etiqueta de cuerpo "I" y como valor el código de celda "cell". Lo utilizaremos para fusionar cuerpos que se encuentren en la misma celda en una misma iteración. Choque inelástico.

Los parámetros **P, P2, T, L, I_max, position** y **cells** son renovados en cada iteración.

## Funciones de los módulos

En el orden en el que aparecen en los módulos auxiliares:

**`pprint(Q):`** serializa la estructura de datos de Python "Q" (por ejemplo, diccionarios, trees, etc.) mediante funciones de la librería json en un formato indentado que resulta más legible. [27]

**`hits(r,v):`** Recibe un string "r" y busca si este está incluido en el vector de strings "v". Devuelve el índice de las posiciones de los elementos de "v" en las que se encuentra "r".

**`p2q_v1(x,y,A):`** Exclusiva del algoritmo en 2D. Llamamos a esta función en cada creación de "P". A partir de las coordenadas "x" e "y" devuelve el código de la celda "cell" en base-4 que tendrá "A+1" dígitos. Contiene cuatro funciones que son creadas en su interior:

**`px(x):`** convierte de base-4 a base-2 (ver figura 5) en eje x mediante la función `replace()` y después de base-2 a base-10 mediante la función `int(,2).`

**`py(y):`** convierte de base-4 a base-2 (ver figura 5) en eje x mediante la función `replace()` y después de base-2 a base-10 mediante la función `int(,2).`

**`v2s(v):`** vector to string.

**`IP(x,y,a):`** Empieza haciendo una división entera entre "a" que es el nivel en el que estamos: `[x//a,y//a]`

A continuación junta ambos componentes convirtiéndolos en un string: `v2s([x//a,y//a])`

Para terminar lo leemos como si fuera en base-2 y lo pasamos a base-10: `int(v2s([x//a,y//a]),2)`

*****`[x//a,y//a]` es un vector con dos componentes de un solo dígito en base-2, al unirlos mediante la función `v2s()` obtenemos un componente de dos dígitos en base-2 que se puede traducir a un

componente de un solo digito de base-4. Mediante la función `int(s,2)` pasamos de base-2 a base-10 pero como nuestro argumento "s" es de dos dígitos en realidad estaremos pasando a base-4.

**ov:** lista de tamaño "A"

El código de celda será una cadena de dígitos en base-4 de longitud proporcional a su nivel de profundidad en la jerarquía del árbol. Es decir, el nodo que contiene a todos los demás tendrá un código de celda de un solo digito (siempre es "0") y los nodos más pequeños (discretización del espacio) tendrán un código de celda de "A+1" dígitos. Los códigos de celda siempre estarán referidos al vértice de abajo izquierda de cada cuadrante.

A medida que bajamos de nivel (aumentamos el tamaño de las celdas) iremos eliminando los dígitos menos significativos (de derecha a izquierda).

Una vez todo definido veamos como llama a estas funciones la función **p2q_v1():**

**ov[0] = IP(x,y,2**(A-1))** : Obtenemos el segundo dígito de la posición del cuerpo en base-4 (el que aparece en rojo en el ejemplo de la figura 5, ya que el primero siempre es cero). `2**(A-1)` es la mitad del espacio entero y por lo tanto `x//2**(A-1)` será 0 o 1. Igual con y. Hemos determinado a qué cuadrante de la primera subdivisión pertenece nuestro cuerpo.

A continuación iteramos entre 1 y "A" ("A"-2 iteraciones) para obtener los dígitos correspondientes a las celdas de niveles superiores (celdas cada vez más pequeñas).

**ov[n] = IP(x - px(os),y - py(os),2**(A-1-n)):** `ov[n]` es el dígito en base-4 del nivel n.

`px(os)` devuelve la coordenada en base-10 (o base-4) del "ov" anterior a este nivel (`ov[n-1]`). `py(os)` igual.

Según subimos de nivel el `2**(A-1-n)` por el que dividimos "x" e "y" va siendo menor y las divisiones `[x//2**(A-1-n),y// 2**(A-1-n)]` dejarán de ser 0 o 1. Para que esto no ocurra restamos a "x" e "y" los valores del cuadrante de un nivel superior, es decir, restar el trozo de espacio que no nos interesa y cuya referencia ya está guardada en `ov[n-1]`. Es imprescindible que las divisiones sean 0 o 1 para poder traducir el código de coordenada a base-4.

Vamos adentrándonos en el árbol haciendo zoom: cada vez conseguimos el digito en base-4 de un nivel más profundo.

**p2o_v1(x,y,z,A):** Exclusiva del algoritmo en 3D. Estructura idéntica a **p2q_v1().** En vez de traducir a base-4 traduciremos a base-8. Al traducir a base-10 tres dígitos concatenados en base-2 en realidad estaremos traduciendo a base-8 (ver figura 7)

**p2q()** y **p2o():** Funciones que sustituyen a **p2q_v1()** y **p2o_v1()** en una versión posterior de la implementación. Estas nuevas versiones son 10 veces más rápidas que sus predecesoras. **p2q_v1()** y **p2o_v1()** son funciones fácilmente generalizables a dimensiones superiores. Sin embargo, al utilizar las nuevas versiones deberemos examinar el "truco" que vamos a explicar a continuación para cada dimensión independientemente.

Convertir las coordenadas de los ejes "x" e "y" a binario es aparentemente un proceso sencillo. En la explicación de **p2q_v1()** hemos visto que si concatenamos los dígitos de las coordenadas binarias de "x" e "y" por parejas (que ocupen la misma posición), obtenemos un numero binario de dos dígitos que con la función `int(,2)` es traducido a base-4.

El truco en 2D es el siguiente:

En vez de concatenar por niveles los dígitos que ocupen la misma posición en la representación binaria de "x" e "y", los sumaremos sustituyendo mediante la función `replace()` los "1" que haya en "x" por "2". Previamente nos habremos encargado de que la representación en binario de "x" e "y" sea de la misma longitud. Ver figura 5 para entender el por qué de este truco. De esta forma, al sumar dígito a dígito las coordenadas en base-2 de "x" y de "y" obtendremos las coordenada del los cuadrantes en base-4.

El truco en 3D es similar:

Traducimos "x", "y" y "z" a binario y sustituiremos los "1" de "y" por "2" y los "1" de "x" por "4". Ver figura 5 para entender el por qué de este truco. Sumamos las coordenadas binarias de los tres vectores dígito a dígito y obtenemos la posición en base-8.

**MASS(Q):** Función opcional. Devuelve la masa total del sistema. Es una función de comprobación (debugging) para asegurar que en todos los niveles del árbol la suma de las masas de todos los planetas es la misma ("ley de conservación de la masa").

**combine(I,*L):** L es un vector de dos o más diccionarios. Cada diccionario contiene la información de un cuerpo o un cúmulo de cuerpos ("P[I]"). La función combina la información de los cuerpos o cúmulos de cuerpos que se encontraban en "L" en un nuevo diccionario "R".

**promotion(C,a):** Su propósito es el de asignar a cada cuerpo/cúmulo de cuerpos la celda de tamaño mayor (nivel menor) en la que este cuerpo/cúmulo se encuentra solo. Las celdas de nivel "a" tienen un código de celda de "a+1" dígitos. Como nuestro objetivo es situar los cuerpos/cúmulos en las celdas de mayor tamaño posible esto equivale a que su código de celda tenga el menor número de dígitos posible. Para ello, vamos eliminando (de derecha a izquierda) los dígitos de los códigos de

celda mientras estos no coincidan con el código de ninguna otra celda que contenga un cuerpo a ese mismo nivel.

Definimos la función **`cindx(n,a)`** en la que "n" es el nivel y "a" el número de decimales y devuelve las posiciones de todos los cuerpos del nivel "n" con el numero de decimales "a".

**`clean(C):`** Función opcional. En una versión posterior de la implementación ha sido eliminada por ralentizar el programa. Después de haber promocionado los cuerpos/cúmulos que se encontraban solos a niveles inferiores (celdas de tamaño mayor) borramos la información de estos cuerpos en niveles superiores por ser redundante.

**`combination(D,a,P,I):`** Dado un árbol "D" un nivel "a" un diccionario "P" y una etiqueta "I" esta función escribe la información de "D" y "P" para el nivel "a" asignando a cada cúmulo de cuerpos creado la etiqueta correspondiente "I". Es decir, actualiza el árbol "D" y el diccionario "P" añadiendo la información de los cúmulos de cuerpos creados a un nivel inferior (nivel de celdas de tamaño mayor dado que creamos el árbol de hojas a raíz).

Definimos la función **`cindx(n,a)`** en la que "n" es el nivel y "a" el numero de decimales y devuelve las posiciones de todos los cuerpos del nivel "n" con el numero de decimales "a".

*En el ejemplo de la figura 5 podemos apreciar que para concretar las posiciones de las celdas de cada nivel necesitamos un número de decimales proporcional al tamaño de las celdas de dicho nivel. El nivel "a" tiene "a+1" decimales.

Es decir, a medida que bajamos de nivel (aumentamos el tamaño de celda) necesitamos un dígito menos para precisar la posición de cada celda. El dígito del que nos deshacemos ese el que precisaba la subdivisión a un nivel más alto: el último digito por la derecha.

"aux" contiene los índices de las celdas que contienen cuerpos de un nivel inferior, es decir, los que tendremos que rellenar combinando cuerpos del nivel superior.

En el nivel "a+1" las celdas tienen "a+2" dígitos, combinaremos en las celdas de nivel "a" aquellos cuerpos cuyos códigos de celda tengan "a+1" dígitos idénticos (todos menos el ultimo de la derecha).

Seleccionamos los cuerpos que se encuentran en la misma celda a un nivel inferior y los combinamos mediante la función **`combine(),`** después actualizamos la información de "D" y de "P" para las siguientes iteraciones.

En cada iteración añadiremos a "D" y "P" la información de un nivel menor. Cabe recalcar que al añadir la información de un nivel menor se añade la información del cúmulo de cuerpos que forman cada uno de los cuerpos que hay en cada una de las divisiones de dicho nivel.

**P_builder(j,v,r,m,A,P2,count,cells):** Esta función actualiza el "P" de los cuerpos individuales a partir de los nuevos datos de posición "r". Es decir, reescribe el "P" de los cuerpos individuales después de que se hayan desplazado en la iteración anterior. Además, combina los cuerpos en caso de que hayan chocado.

**tree_builder(P):** Función **recursiva.** A partir de un "P" que contenga la información de todos los cuerpos individuales y de todos los cúmulos de cuerpos que se hayan formado a todos los niveles construye una estructura anidada de tipo árbol "T" conteniendo exclusivamente información sobre índices y nodos.

**Tree_Renewer(P,N,A):** Dado un número de cuerpos "N", niveles "A" y el diccionario "P" con la información de los "N" cuerpos iniciales exclusivamente, devuelve el árbol "T" y el diccionario "P" con la información de las agrupaciones de cuerpos a todos los niveles además de los de los "N" cuerpos iniciales que ya contenía. Para ello llama a las funciones **promotion(), clean(), combination() y tree_buillder().**

**new_velocity_and_position(a,F,xmax,P):** A partir de la aceleración "F" ("F" es la fuerza gravitatoria total que experimenta el cuerpo dividida por su propia masa) padecida por el cuerpo de etiqueta "a" y las posiciones y velocidades de este en la iteración anterior, calculamos las nuevas velocidades $v_x$ y $v_y$ (resp. $v_x$, $v_y$ y $v_z$) y posiciones $r_x$ y $r_y$ (resp. $r_x$, $r_y$ y $r_z$) del cuerpo de etiqueta "a". "P" es utilizado para leer las velocidades y posiciones del paso anterior de los cuerpos individuales exclusivamente.

El algoritmo de integración utilizado es Euler con intervalo de tiempo o paso t.

Cuando los cuerpos chocan con los límites del espacio cambiamos el sentido de la velocidad perpendicular a la pared en cuestión. Los cuerpos realmente rebotan elásticamente en las paredes del contenedor por lo que podemos asimilar este sistema a un "gas de planetas".

**IvsSize(D,I,L,A):** Función **recursiva** que va recorriendo el árbol "T" y devuelve un diccionario "L". Las llaves del diccionario "L" son las etiquetas "I" de cada cuerpo o cúmulo de cuerpos y los valores el tamaño del nodo correspondiente: $2^{A-level}$.

**BH(D,I,j,F,P,L,N,theta):** Recorremos el árbol "D" de forma **recursiva** leyendo la información de "P". El índice "I" nos permite conectar la información que contienen "D" y "P". "j" es el cuerpo (individual, uno de los N-1 iniciales) del que calculamos la fuerza que ejercen los demás sobre él. En realidad la función devuelve directamente la aceleración aj con el nombre "F" que es la fuerza entre la masa del propio cuerpo "j". "F" tendrá componentes (x, y) en 2D y (x, y, z) en 3D. Para calcular la fuerza recorremos el árbol de raíz a hojas y calculamos la fuerza entre el cuerpo "j" y el nodo "el" en

el que nos encontremos del árbol siempre que se cumpla el criterio de lejanía suficiente de Barnes-Hut: $s/d < theta$

“s”: tamaño de la celda del nodo, información que está almacenada en “L”.

“d”: distancia de cuerpo a nodo.

“theta”: valor umbral del criterio.

Si el criterio se cumple no hace falta seguir recorriendo esa rama del árbol, la contribución a la fuerza total ejercida sobre “j” que efectúan los subnodos de esa rama ya la hemos tenido en cuenta. Recordemos que los nodos superiores del árbol contienen la información de la masa total de sus nodos descendientes así como la posición del centro de masas correspondiente.

G está definido dentro de esta función.

## Programa principal

1) Creamos el “P” para los primeros “N” cuerpos.
2) Creamos el árbol “T” y añadimos a “P” la información de los cúmulos de cuerpos.
3) Creamos el diccionario “L” con la información “l” vs tamaño de celda.
4) Las iteraciones en “w” son iteraciones del sistema entero. Las iteraciones en “j” recorren los cuerpos individuales (“s”= 1) que haya en el sistema.
    a. Para cada cuerpo “j” calculamos la aceleración resultante de la fuerza total ejercida por los demás cuerpos. A partir de esta aceleración calculamos la nueva posición y velocidad del cuerpo.
    b. Añadimos la nueva posición del cuerpo “j” a la lista “position” para poder dibujarla.
    c. Calculamos el “P[“j”]”. Es decir, el fragmento de “P” que corresponde al cuerpo de etiqueta “j”. En caso de haber chocado aplicamos choque inelástico.
5) Una vez hechos los pasos a, b y c para todos los cuerpos individuales pasaremos a la iteración del sistema entero para recalcular el árbol “T” y el diccionario “P” al que añadimos la información de los cúmulos de cuerpos.
6) Recalculamos “L”.
7) Guardamos la información de la iteración y pasamos a la siguiente. Repetir desde el paso 4 hasta el final.

8) A modo de ejemplo para poder usar la función `tree_grapher` guardamos el árbol de la última iteración.

## Esquema árbol

**`tree_grapher`** es un fichero que contiene una única función que es ejecutable desde la línea de comandos.

**`tree_crawler(t,i,G):`** Dibuja y guarda el esquema del árbol de dos o tres dimensiones que tengamos guardado en el fichero save_T.

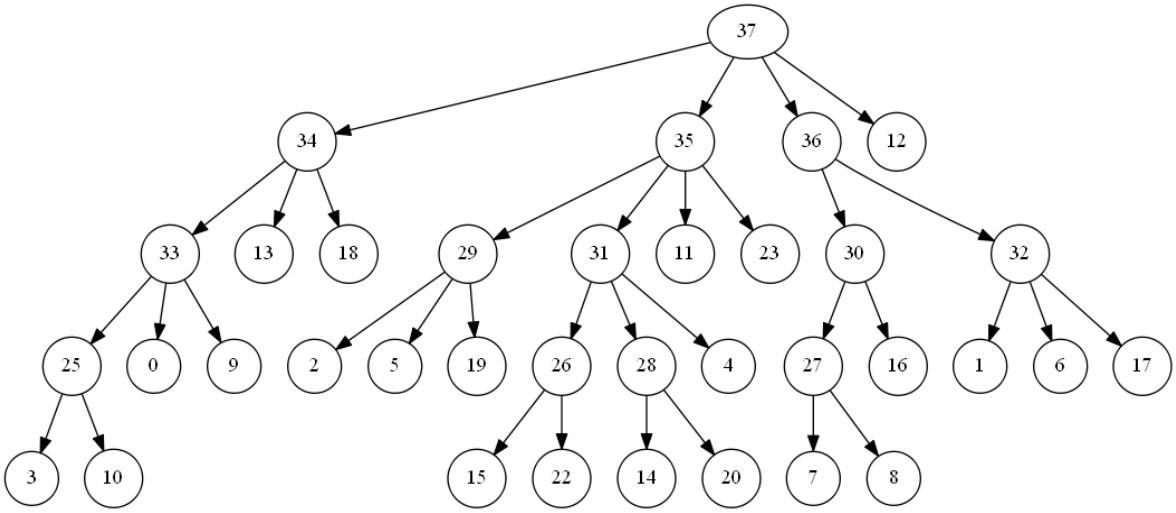


**Figura 19: Grafo de un árbol de 2D de 25 cuerpos individuales obtenido con la function `tree_crawler()`**

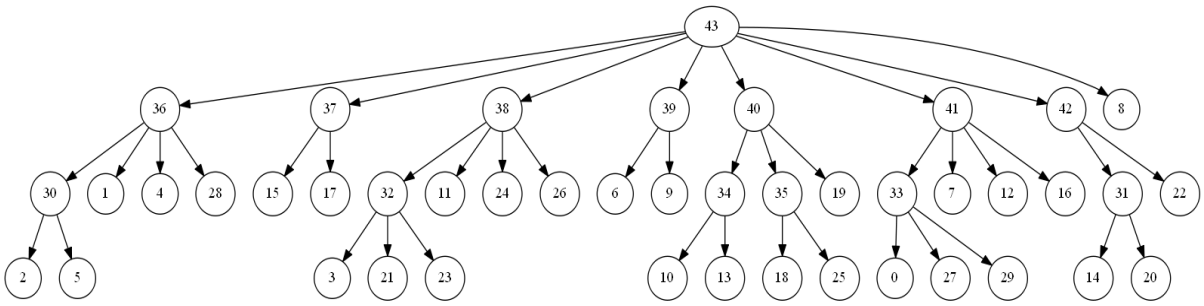


**Figura 20: Grafo de un árbol de 3D de 30 cuerpos individuales obtenido con la function `tree_crawler()`**

## Referencias bibliográficas


[1] R. Fitzpatrick, *An introduction to celestial mechanics*. New York, N.Y: Cambridge University Press, 2012

[2] Forest R. Moulton, *Introduction to Celestial Mechanics*, New York, N.Y: Dover, 1984.

[3] D. Gottlieb and J. Barrow-Green, "Poincare and the Three Body Problem.", *The American Mathematical Monthly*, vol. 106, no. 10, p. 977, 1999. Disponible en: 10.2307/2589771.

[4] T. Gowers, J. Barrow-Green and I. Leader, *The Princeton companion to mathematics*. Princeton, N.J.: Princeton University Press, 2008.

[5] A. Appel, "An Efficient Program for Many-Body Simulation", *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 1, pp. 85-103, 1985. Disponible en: 10.1137/0906008.

[6] Pangfeng Liu and S. Bhatt, "Experiences with parallel N-body simulation", *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 12, pp. 1306-1323, 2000. Disponible en: 10.1109/71.895795.

[7] U. Becciani, V. Antonuccio-Delogu and M. Gambera, "A Modified Parallel Tree Code for N-Body Simulation of the Large-Scale Structure of the Universe", *Journal of Computational Physics*, vol. 163, no. 1, pp. 118-132, 2000. Disponible en: 10.1006/jcph.2000.6557.

[8] J. Dubinski, "A parallel tree code", *New Astronomy*, vol. 1, no. 2, pp. 133-147, 1996. Disponible en: 10.1016/s1384-1076(96)00009-7.

[9] "1965: "Moore's Law" Predicts the Future of Integrated Circuits | The Silicon Engine | Computer History Museum", *Computerhistory.org*, 2020. [Online]. Disponible en: https://www.computerhistory.org/siliconengine/moores-law-predicts-the-future-of-integrated-circuits/. [Consultado: 14- Jun- 2020].

[10] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers (Chapman & Hall/CRC computational science series)*. CRC Press, 2010.

[11] R. Miller and K. Prendergast, "Stellar Dynamics in a Discrete Phase Space", *The Astrophysical Journal*, vol. 151, p. 699, 1968. Disponible en: 10.1086/149469.

[12] R. Miller, K. Prendergast and W. Quirk, "Numerical Experiments on Spiral Structure", *The Astrophysical Journal*, vol. 161, p. 903, 1970. Disponible en: 10.1086/150593.

[13] S. Aarseth, E. Turner and J. Gott, "N-body simulations of galaxy clustering. I - Initial conditions and galaxy collapse times", *The Astrophysical Journal*, vol. 228, p. 664, 1979. Disponible en: 10.1086/156892.

[14] R. Hockney and J. Eastwood, *Computer simulation using particles*. New York: Taylor & Francis, 1988.

[15] [13]"CUDA", *En.wikipedia.org*, 2020. [Online]. Disponible en: https://en.wikipedia.org/wiki/CUDA. [Consultado: 18- Jun- 2020].

[16] M. Burtscher and K. Pingali, "An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm". 2011.

[17] S. Aarseth, *Gravitational N-body simulations*. Cambridge: Cambridge University Press, 2009.

[18] J. Barnes and P. Hut, " A Hierachical *O(N log N)* Force-Calculation Algorithm," *Nature*, vol. 324, 1986.

[19] "Método de Euler", *Es.wikipedia.org*, 2020. [Online]. Disponible en: https://es.wikipedia.org/wiki/M%C3%A9todo_de_Euler. [Consultado: 8- Jun- 2020].

[20] "GKQuadtree - GameplayKit | Apple Developer Documentation", *Developer.apple.com*, 2020. [Online]. Disponible en: https://developer.apple.com/documentation/gameplaykit/gkquadtree. [Consultado: 8- Jun- 2020].

[21] "Octree", *Facebook.com*, 2020. [Online]. Disponible en: https://www.facebook.com/OctTree/photos/a.1590613727628371/1590607487628995 [Consultado: 8- Jun- 2020].

[22] W. Press, *Numerical recipes in FORTRAN*. Cambridge: Cambridge University Press, 1992.

[23] J. Makino, P. Hut, M. Kaplan and H. Saygın, "A time-symmetric block time-step algorithm for N-body simulations", *New Astronomy*, vol. 12, no. 2, pp. 124-133, 2006. Disponible en: 10.1016/j.newast.2006.06.003.

[24] T. Team, "PyPy", *PyPy*, 2020. [Online]. Disponible en: https://www.pypy.org/. [Consultado: 15- Jun- 2020].

[25] "Natural units", *En.wikipedia.org*, 2020. [Online]. Disponible en: https://en.wikipedia.org/wiki/Natural_units. [Consultado: 9- Jun- 2020].

[26] W., Hall, A. and can, p., 2020. *What Is The Best Way To Implement Nested Dictionaries?*. [online] Stack Overflow. Disponible en: <https://stackoverflow.com/questions/635483/what-is-the-best-way-to-implement-nested-dictionaries> [Consultado 24 Mar 2020].

[27] Docs.python.org. 2020. *Json — JSON Encoder And Decoder — Python 3.8.3 Documentation*. [online] Disponible en: <https://docs.python.org/3/library/json.html> [Consultado 4 Abr 2020].

[28] D. Souami and J. Souchay, "The solar system's invariable plane", *Astronomy & Astrophysics*, vol. 543, p. A133, 2012. Disponible en: 10.1051/0004-6361/201219011.

[29] "NumPy", *Numpy.org*, 2020. [Online]. Disponible en: https://numpy.org/. [Consultado: 4- Jun- 2020].

[30] "Born–von Karman boundary condition", *En.wikipedia.org*, 2020. [Online]. Disponible en: https://en.wikipedia.org/wiki/Born%E2%80%93von_Karman_boundary_condition. [Consultado: 18- Jun- 2020].