T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

HİYERARŞİK N-CİSİM ALGORİTMALARININ PERFORMANS KARŞILAŞTIRILMASI VE VERİ YAPILARININ İNCELENMESİ

A. Burak İNNER

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

TEZ DANIŞMANI
Yrd.Doç.Dr.F. Erdoğan SEVİLGEN

GEBZE
2004

Burak İNNER'in tez çalışması, G.Y.T.E. Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 15.7.2006 tarih ve .2006/29 sayılı kararıyla oluşturulan jüri tarafından Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS Tezi olarak kabul edilmiştir.

JÜRİ

ÜYE (Tez Danışmanı) : Yrd. Doç. Dr. F. Erdoğan SEVİLGEN

ÜYE : Yrd. Doç. Dr. Mehmet GÖKTÜRK

ÜYE : Yrd. Doç. Dr. İlyas KANDEMİR

ONAY

G.Y.T.E. Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun .11.11.2006..... tarih ve .2006/42.. sayılı kararı.

Enstitü Müdürü

Yrd. Doç. Dr. Ahmet ÇAPOĞLU
GYTE Mühendislik ve Fen Dil.
Enstitüsü Müdür Yardımcısı

ÖZET

Barnes-Hut ve Hızlı çok-kutup (Fast Multipole Method), N-cisim problemine etkin bir çözüm getiren metotlardır. Her iki metod da cisimleri hiyerarşik olarak gruplayan ağaç yapıları (üç boyutta sekizli ağaç gibi) kullanmakta ve cisim-cisim etkileşimi yerine cisim-grup veya grup-grup etkileşimleri kullanarak hızlı çözümler sunmaktadırlar. Fakat sekizli ağacın yüksekliği ve düğüm sayısı cisimlerin dağılımlarına bağlıdır. Cisimlerin dağılımları düzenli olduğunda O(logN) yüksekliğinde ve O(N) düğüm sayısına sahip ağaç elde edilir fakat düzensiz dağılımlar için ağacın yüksekliği ve düğüm sayısı için bir üst limit vermek mümkün değildir. Her iki algoritmanın çalışma zamanı da ağacın yüksekliğinin ve düğüm sayısının bir fonksiyonu olduğundan algoritmaların çalışma süresi cisimlerin dağılımına bağlıdır. Bu algoritmaları dağılımdan bağımsız hale getirmek için sıkıştırılmış sekizli ağaç (compressed octree) veri yapısı önerilmiştir. Sıkıştırılmış sekizli ağacın yüksekliği ve düğüm sayısı hem düzenli dağılım hem de düzensiz dağılımlar için O(N)'dir. Sıkıştırılmış ağaç yapısının kullanımının düzensiz dağılımlar için performans artışı sağladığı teorik olarak gösterilmiştir.

Bu çalışmada sekizli ağaç yapısı ve sıkıştırılmış sekizli ağaç yapıları kullanılarak Barnes-Hut ve Hızlı çok-kutup algoritmaları gerçeklenmiştir. Her iki veri yapısı kullanılarak algoritmaların performansları, çalışma zamanları, doğruluk oranları gibi konularda karşılaştırılmalar yapılmış ve teorik sonuçların pratik etkileri gözlemlenmiştir.

Karşılaştırmaların doğru sonuçları vermesi açısından her iki algoritma da aynı ana çatı altında gerçeklenmiştir. Bu ana çatı cisimler ile ilgili bilgileri ve ağaç yapılarını içermektedir. Algoritmaların genel işleyişindeki benzerlikler böyle bir ana çatı oluşturmayı kolaylaştırmaktadır.

Algoritmaların gerçeklenmesinde yerçekimsel kuvvetler kullanılmıştır. Fakat hazırlanan yazılım başka kuvvet çeşitleri içinde genellenebilecek bir yapıdadır.

SUMMARY

Barnes-Hut and Fast Multipole Method use hierarchical clustering of the bodies and effectively solve the N-body problem. Both algorithms use hierarchical tree structures (such as octree for 3D) for clustering and provide fast solutions by employing particle-cluster or cluster-cluster interactions instead of particle-particle interactions. The height and the number of nodes of the tree structure used depend on the distribution of bodies. Using octree data structure for uniform distribution the height of the tree and the number of nodes are bounded by O(logN) and O(N) respectively but for a non-uniform distribution it's hard to give such bounds. So, the running times of algorithms not only depend upon the number of bodies in the problem but also depend upon the distribution. To rectify this problem, the utilization of compressed hyperoctrees is proposed. For a compressed hyperoctree, the height of the tree and the number of nodes are bounded by O(N) for all distributions. Especially for non-uniform distributions, it has been theoretically proved that compressed hyperoctrees provide better performance.

In this thesis, we implement the Barnes-Hut and Fast Multipole Method algorithms using both octree and compressed octree data structures. We evaluate the performance of both algorithms with respect to the running time and accuracy. We examine the practical effects of the theoretical results.

For an accurate comparison we implement both algorithms with a general framework. In this framework, we store the data about the particles and tree data structure. Similarities between the operations performed by each algorithm make it easier to implement such a framework.

We use the gravitational problem while implementing the algorithms but the adaptation of our implementations for various force calculations can be obtained easily.

TEŞEKKÜR

Bu çalışmanın her safhasında özveri ile yaklaşan danışmanım Yrd. Doç. Dr. Fatih Erdoğan SEVİLGEN'e (GYTE), bu noktaya gelişimde her zaman yanımda olan aileme, maddi ve manevi desteğini esirgemeyen eşime teşekkürü bir borç bilirim.

İÇİNDEKİLER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

HÇA : Hızlı Çok-Kutup Algoritması

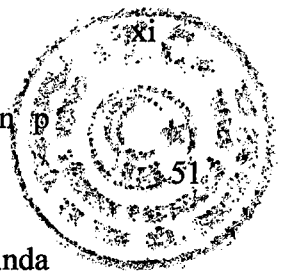
BHA : Barnes-Hut Algoritması

FMM : Fast Multipole Method

Θ: Teta parametresi

p: Çok-kutup açılımlarında alınan terim sayısı

ŞEKİLLER DİZİNİ



ALGORİTMALAR DİZİNİ

1. GİRİŞ

Klasik N -cisim problemi, N tane cisimden oluşan bir sistemde, her bir cismin üzerindeki diğer tüm cisimlerin etkilerini ve bu etkilerden dolayı oluşan hareketin hesaplanmasını içerir. Pek çok bilimsel çalışma birbirlerini etkileyen cisimlerden oluşan böyle fiziksel sistemlerin davranışının incelenmesi konusunu ele alır. Farklı fiziksel sistemlerde cisimler arasındaki etkileşim kütle çekim kuvveti veya elektrostatik kuvveti gibi farklı farklı olabilir ama genel olarak kuvvet cisimler arasındaki uzaklığın karesi ile ters orantılıdır yani ters kare kuralını sağlamaktadır. N-cisim problemi ile astrofizik, 3-boyutlu kapasite [12], plazma fiziği [13], akışkanlar mekaniği [7] ve ısı potansiyeli [8] gibi pek çok alanlarda karşılaşılmaktadır. Geleneksel fiziksel sistem uygulamaları dışında bilgisayar grafiklerinde [11] ve eliptik parçalı diferansiyel denklemlerin çözümünde de [3] benzer yöntemler kullanılmıştır.

Dört ve daha fazla cisim bulunan ve ters kare kuralını sağlayan N-cisim problemleri için bilinen genel bir analitik çözüm yoktur. Bu durumda, istenilen zaman sonunda cisimlerin konumlarının ve hızlarının bulunabilmesi için küçük zaman adımlarında simülasyon çalıştırılması popüler bir yaklaşımdır. Her bir adımda, cisimlerin her birine etkiyen kuvvet ve bu kuvvetin etkisiyle cisimlerin hareketleri hesaplanır. Bir adımda cisimlere etkiyen kuvvetin hesaplanmasının en basit yöntemi doğrudan hesaplama yöntemidir. Bu yöntem, cisimler arasındaki kuvvetleri tek tek hesapladığı için cisim-cisim etkileşimi olarak da bilinir. N tane cisim bulunan bir sistemde, doğrudan hesaplama yöntemi yardımıyla hesaplama yapıldığında, toplam $N(N-1)$ tane cisim-cisim etkileşimi yani kuvvet hesabı yapılmalıdır. Bu sebeple N sayısı büyüdüğünde gereken işlem sayısı çok hızlı artmakta ve simülasyon çok uzun zaman almaktadır. Örneğin, 10^6 tane cisim söz konusu olduğunda bile bir adım sonraki kuvvetlerin hesaplanması için saatlerce hesap yapılması gerekmektedir.

Genel olarak, fiziksel sistem simülasyonunun anlamlı sonuçlar vermesi için simülasyonun çok fazla sayıda cisim içeren sistemler için yapılması gerekir. Böyle

sistemlerin simülasyonunda, doğrudan hesaplama yöntemiyle yapılan hesaplamalar çok uzun süreceği için hesaplama zamanını azaltacak algoritmalar geliştirilmiştir. Bu çalışmada ilgili algoritmalardan, bir adım için kuvvet hesaplamasını (sistemi oluşturan cisimler düzenli dağıldığında) $O(N \log N)$ zamanda yapabilen Barnes-Hut [1] ve $O(N)$ zamanda yapabilen Hızlı çok-kutup [2][3] algoritmaları ve bu algoritmaların kullandığı veri yapıları incelenmiştir.

Bu algoritmalarda kuvvet hesabı, her bir cisme etkiyen kuvvet üç kısma ayrılarak yapılır:

$$F_{toplam\_etki} = F_{yakın\_cisimler} + F_{uzaktaki\_cisimler} + F_{dış\_etkiler} \quad (1)$$

Eğer varsa, dış etkilerden meydana gelen kuvvetin hesabı, her bir cisim için birbirinden bağımsız olarak yapılabilir. Yakın cisimlerin etkileri doğrudan hesaplama yöntemi ile bulunur. Uzaktaki cisimlerin etkilerinin hesabı yapılırken ise kuvvet çeşitli yaklaşık hesaplama yöntemleri kullanılarak hesaplanır. Bir cismin diğerine uyguladığı kuvvetin ve bu yöntemlerden doğacak hatanın, cisimlerin aralarındaki uzaklıkla ters orantılı olması nedeniyle, birbirinden yeteri kadar uzak olan cisimler için hatanın küçük kalması sağlanabilmektedir. Fakat aynı yöntemler yakın cisimler için de kullanıldığında hata çok daha büyük olmaktadır.

Barnes-Hut ve Hızlı çok-kutup algoritmaları birbirlerinden yeterince uzakta bulunan cisim grupları arasındaki etkileşimler için farklı yöntemler kullanırlar. Barnes-Hut algoritmasında uzaktaki cisim gruplarının tek bir sanal cisim olduğu varsayılır. Bu sanal cismin kütlesi, gruplanmış cisimlerin toplam kütlesi ve konumu, gruplanmış cisimlerin kütle merkezidir. Hızlı çok-kutup algoritmasında ise uzaktaki cisim gruplarının etkisinin hesaplanması için Taylor serisi gibi kuvvet serileri kullanılır. Hızlı çok-kutup algoritmasını Barnes-Hut algoritmasından ayıran en önemli özellik, birbirinden uzakta bulunan grupların etkileşiminin hesaplanmasını sağlayacak açılımlardır. Barnes-Hut algoritması etkileşimlerin yaklaşık olarak hesaplamasından dolayı oluşacak hata için, önceden belirlenen bir parametre sayesinde üst sınır ve tahmini bir hata verebilmektedir. Hızlı çok-kutup algoritması

ise işlem sonundaki hatayı sınırlandırabilmekte ve istenilen hata miktarına göre hesaplama imkanı sunmaktadır.

Barnes-Hut ve Hızlı çok-kutup algoritmaları cisimleri gruplandırmak için hiyerarşik ağaç veri yapıları kullanırlar. Bu veri yapısı iki boyutta dörtlü ağaç (quadtree), üç boyutta ise sekizli ağaç (octree) olarak adlandırılır. Hiyerarşik ağaç veri yapılarını oluşturmak için cisimlerin bulundukları uzay hiyerarşik olarak iki boyutta dört, üç boyutta sekiz eşit parçaya bölünür. Bir parça içinde birden -veya önceden belirlenmiş sabit bir sayıda cisimden- fazla cisim bulunması durumda bu parça tekrar küçük parçalara bölünür ve bu parçalar ağacın düğümleri ve alt-düğümlerini oluşturur. Cisim dağılımları düzenli olduğunda dengeli bir ağaç veri yapısı elde edilebilir ve etkin bir hesaplama yapılabilir. Fakat düzensiz dağılım durumunda ağaçta bulunan düğüm sayısı ve ağacın yüksekliği için bir üst sınır vermek mümkün olmadığı gibi algoritmaların çalışma zamanları için de üst sınır vermek de mümkün değildir. Yani algoritmaların çalışma zamanları sadece cisimlerin sayısına değil aynı zamanda cisimlerin dağılımlarına da bağlıdır. Böyle algoritmalara dağılıma bağlı algoritmalar denilmektedir. Dağılıma bağlı algoritmaların analizleri zorlaşmaktadır ve analizler genellikle en iyi çalışma zamanını veren düzenli dağılımlar için yapılmaktadır.

Cisim dağılımlarının düzensiz olduğu durumlarda ağacın yüksekliği dengesiz olarak artmakta ve ağaç üzerinde ikiye dallanma olmadan bulunan düğümler olabilmektedir. Bu tür yolları sıkıştırarak ağacın yüksekliğinin cisim sayısı ile lineer olarak artmasını sağlayan veri yapıları geliştirilmiştir. Bunlardan sıkıştırılmış sekizli ağaç yapısı [14] düzensiz dağılımlar olduğunda bile ağaçtaki toplam düğüm sayısının ve ağacın yüksekliğinin en fazla $O(N)$ olmasını sağlayabilmektedir. Bu ağaç yapısında bir düğüm için büyük hücre ve küçük hücre diye iki değer tutulmaktadır. Büyük hücre, ağaç üzerinde ikiye dallanma olmadan bulunan bir yolda en üstteki düğümün bilgilerini, küçük hücre ise en alttaki düğüme karşılık gelen düğümün bilgilerini saklamak için kullanılmaktadır. Böylece sıkıştırma işlemi sonunda herhangi bir bilgi kaybı olmadan hesaplamalar yapılabilmektedir.

Bu çalışmada Barnes-Hut ve Hızlı çok-kutup algoritmaları sekizli ağaç veri yapısı kullanılarak C++ programlama dili ile gerçeklenmiş daha sonra bu algoritmalar sıkıştırılmış sekizli ağaç yapısı ile gerçeklenerek çalışma zamanları, bellek karmaşıklığı ve doğruluk oranları karşılaştırılmıştır. Algoritmalar düzenli ve düzensiz dağılımlar için çalıştırılarak sonuçlar incelenmiştir.

2. N-CİSİM ALGORİTMALARI

Popüler N-cisim yaklaşımlarından biri şöyledir; Simülasyon küçük zaman adımlarında çalıştırılarak her bir adımda önce tüm cisimlerin birbirlerine uyguladığı kuvvetler hesaplanır daha sonra bu kuvvetler yardımıyla hızlar ve konumlar hesaplanarak bir sonraki zaman adımına geçilir. Kuvvetten konumlara geçerken kullanılan integrasyon yöntemine bağlı olarak küçükte olsa bir hata yapılmaktadır. Zaman aralığının çok küçük seçilmesi yapılan hata miktarını azaltacak fakat simülasyon zamanının artmasına sebep olacaktır. Değişik N -cisim uygulamaları için hesaplanacak olan kuvvetlerin formülleri farklı olsa bile simülasyon için kullanılacak temel adımlar şunlardır :

Her bir Δt küçük zaman adımı için

Kuvvet Hesaplaması yap

Bir integrasyon yöntemi ile konum ve hıza geç

Örnek olarak yerçekimsel kuvvet hesabını ele alalım. Kütleleri m_a ve m_b olan 2 gezegenin yerçekimi kuvveti Newton'un evrensel çekim kanununa göre

$$F = -G\frac{m_a m_b}{r^2} \tag{2}$$

formülü yardımıyla hesaplanabilir. Burada G yerçekimi sabitini, r gezegenler arasındaki uzaklığı göstermektedir. Bu şekilde her gezegenin bir diğerine uyguladığı kuvvet hesaplanabilir. Gezegenler arasındaki kuvvet, aradaki uzaklığın karesi ile ters orantılıdır. Gezegen, bulunan bu bileşke kuvvetine bağlı olarak Newton'un 2. yasası olan

$$F = ma = m\frac{dv}{dt} = m\frac{d^2x}{dt^2}$$

formülüne göre ivme (a) kazanmaktadır. Burada v hızı, x konumu, t zamanı, a ivmeyi göstermektedir. Bu diferansiyel denklem Euler, Leapfrog vb. gibi bir integrasyon yöntemi kullanarak çözülebilir.

Euler yönteminde cisimler üzerindeki kuvvetlerin Δt zaman aralığında sabit olduğu kabul edilir. Gerçekte bu küçük zaman aralığında cisim üzerine etki eden kuvvetler, zaman aralığı boyunca değişecektir çünkü aralarındaki uzaklıklar değişmektedir. Bu nedenle çok küçük de olsa hata yapılacaktır. Zaman aralığının çok küçük seçilmesi, işlem zamanını artıracak fakat hatayı azaltacaktır. Zaman aralıklarının çok büyük seçilmesi durumda ise simülasyon zamanı kısalacak fakat daha fazla hata yapılacaktır. Bu durumda m kütleli cisim için kuvvet hesabı şu şekilde yapılabilir:

$$F = ma = \frac{m(v^{t+\Delta t} - v^t)}{\Delta t} \tag{4}$$

$$v^{t+\Delta t} = v^t + \frac{F\Delta t}{m} \tag{5}$$

Burada $v^{t+\Delta t}$, $t+\Delta t$ zamanındaki hızı, v^t, t zamanındaki hızı göstermektedir. Cismin yeni hızı hesaplandıktan sonra

$$x^{t+\Delta t} - x^t = v^t \Delta t \tag{6}$$

formülü kullanılarak $t+\Delta t$ zamandaki konumu hesaplanabilir.

Hız ve pozisyon hesaplamasında hatayı azaltmak için alternatif olarak "leap-frog" hesaplama yöntemini kullanabilir. Bu yöntemde Δt zaman aralığındaki hızın ortalaması alınarak cismin bu zaman aralığında bu hızla gittiği düşünülmektedir.

Leap-frog yöntemi fazla hesaplama yapılmasını gerektirmemektedir. Hızların $t+\frac{\Delta t}{2}$, $t+\frac{3\Delta t}{2}$, $t+\frac{5\Delta t}{2}$ zamanlarında, konumların t, $t+\Delta t$, $t+2\Delta t$, $t+3\Delta t$ zamanlarındaki değerleri kullanılarak hesaplamalar yapılması yeterli olmaktadır. Böylece zaman aralığının tam ortasındaki hızlar alınmış olur.

$$F^t = \frac{m(v^{t+\frac{\Delta t}{2}} - v^{t-\frac{\Delta t}{2}})}{\Delta t} \tag{7}$$

$$x^{t+\Delta t} - x^t = v^{t+\frac{\Delta t}{2}} \Delta t \tag{8}$$

İstenilen zaman sonunda hesaplamalardan dolayı yapılan hatanın miktarı hakkında fikir vermesi açısından enerjinin korunumu prensibinden yararlanılabilir. Başlangıç konumlarındaki enerji ile son konumdaki enerji arasındaki fark karşılaştırılır. Enerji hesaplaması yapılırken potansiyel ve kinetik enerji ayrı ayrı hesaplanarak toplanır. Potansiyel enerji ve kinetik enerji sırasıyla:

$$E_{kinetik} = \sum_{i=1}^{N} \frac{1}{2} m_i v_i^2 \tag{9}$$

$$E_{potansiyel} = \sum_{i=1}^{N} \sum_{j \neq i}^{N} \frac{m_i m_j}{|r_i - r_j|} \tag{10}$$

formülleri ile hesaplanabilir. Kinetik enerji hesaplaması tüm cisimler için birbirinden bağımsız yapılabilir. Potansiyel enerji hesaplaması yapılabilmesi diğer cisimlerin de kullanılması gerekmektedir. Bu nedenle potansiyel hesabı cisim-cisim hesaplamasında olduğu gibi $O(N^2)$ tane hesaplama yapılmasını gerektirmektedir.

2.1. Kuvvet Hesaplama Yöntemleri

N-cisim yöntemleri, kuvvet hesaplamada kullandıkları yöntemler nedeniyle birbirlerinden ayrılmaktadır. Bu yöntemler kuvvet hesaplama işlemini daha kısa zamanda yapabilmek için çeşitli yöntemler kullanmaktadırlar fakat kuvvet hesapladıktan sonra bir integrasyon yöntemi yardımıyla aynı şekilde hızlara ve konumlara geçerler. Bu bölümde kuvvet hesaplamada kullanılan yöntemlerden doğrudan hesaplama yöntemi ve hiyerarşik hesaplama yöntemlerinden Barnes-Hut ve Hızlı çok-kutup yöntemleri incelenecektir.

2.2. Cisim–Cisim Hesaplama Yöntemi

Bu kuvvet hesaplama yönteminde her bir adım için tüm cisimlerin birbirlerine uyguladığı kuvvetler ayrı ayrı hesaplandığı için cisim-cisim hesaplama yöntemi olarak adlandırılmıştır. Bir anlık kuvvet için tam sonucu veren algoritmadır fakat bu hesaplama için toplam $O(N^2)$ hesaplama yapılması gerekir. Yani kuvvet hesabı için gerekli süre cisim sayısının karesi ile orantılıdır. Bu yöntem cisim sayısının çok büyük olduğu problemlerde çok fazla süre almaktadır. Çok fazla sayıda cisim içeren simülasyonlar için uygun bir yöntem değildir.

İki cisim arasındaki kuvvet her bir cisim için bir kere hesaplanarak, zıt yönlü kuvvet alınabilir. Böylece işlem sayısı yarıya indirilmiş olur fakat çalışma zamanı hala çok yüksek olmaktadır. Cisim-cisim hesaplama yönteminin algoritması Algoritma 2.1'de verilmiştir.

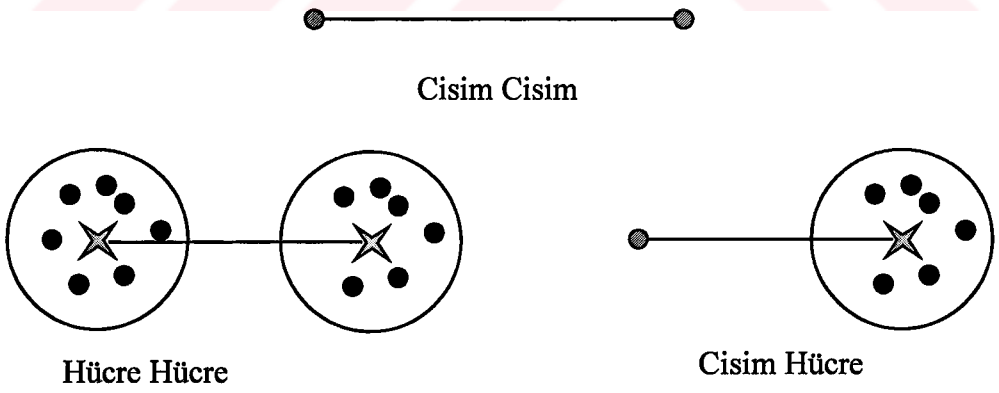
```
Algoritma Cisim-cisim

Girdi: Kuvvet, Kuvvet dizisi
 N, Cisim Sayısı
Çıktı: Kuvvet, Hesaplanmış kuvvetler

For each Δt
  For i from 1 to N do
Kuvvet[i] = 0;
For j from 1 to i-1 do
  Aradaki_kuvvet=i ve j cisimleri arasındaki kuvvet;
  Kuvvet[i] += aradaki_kuvvet;
  Kuvvet[j] -= aradaki_kuvvet;
End For j;
  End For i;
  For i from 1 to N do
Hız[i] = Kuvvet[i] kullanarak hesapla;
Konum[i] = Kuvvet[i] kullanarak hesapla;
  End For i;
End For each Δt;
```

Algoritma 2.1 Cisim-cisim Algoritması

2.3. Hiyerarşik Ağaç Yöntemleri

Hiyerarşik ağaç yönteminde bir cismin diğerine uyguladığı kuvvetin cisimler arasındaki uzaklığın karesi ile ters orantılı olma özelliğinden yararlanılmıştır. Cisim-cisim etkileşimi yapmak yerine hiyerarşik ağaç algoritmaları, birbirlerinden uzakta bulunan cisimleri Şekil 2.1'deki gibi gruplayarak bu gruplanmış cisimlerle etkileşim yapmaktır. Düzenli dağılmış cisimler için hiyerarşik ağaç yöntemlerinden Hızlı çok-kutup algoritması $O(N)$ ve Barnes-Hut algoritması $O(N \log N)$ zamanda kuvvet hesabı yapabilmektedir. Düzenli dağılmış cisimler için bu yöntemlerde kullanılan ağaçtaki düğüm sayısının $O(N)$ olması ve ağacın dengeli olması, hesaplama sayısının az olmasını sağlamaktadır. Düzensiz dağılımlarda ise ağaç dengesiz olabilmekte ve ağaçtaki düğüm sayısı için bir sınır vermek mümkün olmamaktadır. Yapılan işlem sayısı ve çalışma zamanı içinde üst sınır vermek mümkün değildir.

Cisim grupları ile etkileşim yapılmasından dolayı algoritmalar çok küçük de olsa hatalı sonuçlar üretmiş olacaklardır. Bu hata gruplanmış cisimler ile etkileşim yapılan cisimler arasındaki uzaklığa bağlıdır ve kullanılan algoritmaya göre sınırlandırılabilmektedir.



Şekil 2.1 Cisim-cisim, cisim-hücre ve hücre-hücre Etkileşimi

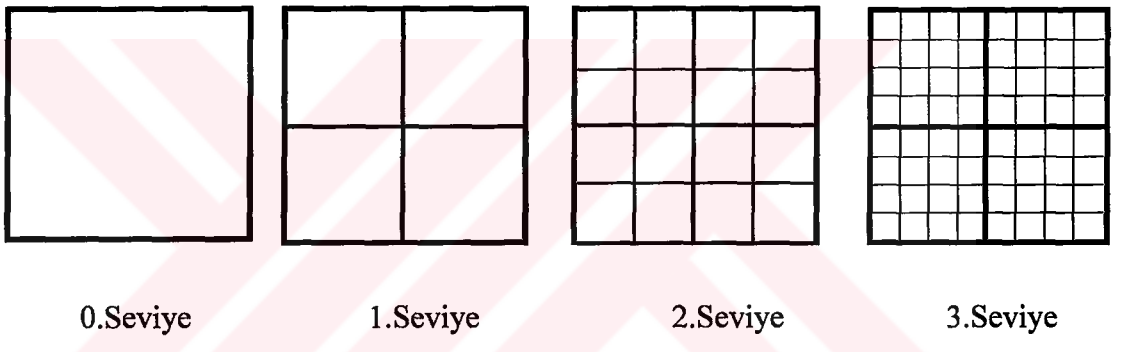
Cisimlerin hiyerarşik olarak gruplanması için gruplanacak cisimleri içine alan bir daire (üç boyutta küre) düşünülür. Eğer bu daireyi içine alan bir kare düşünülür ve buna göre işlem yapılırsa hesaplamalar ve gruplandırmalar daha kolay olacaktır. Çünkü karelerin tekrar kareler şeklinde bölümlenmesi kolay olduğu halde daire için aynı şeyi söylemek mümkün değildir. Cisimlerin hiyerarşik olarak gruplandırması tüm cisimleri içine alan en büyük karenin hiyerarşik olarak bölümlenmesi ile yapılır. Hiyerarşik olarak bölümlenmiş gruplar bir veri yapısında saklanır. (Bkz. Bölüm 3)

Barnes-Hut ve Hızlı çok-kutup algoritması benzer ağaç yapılarını kullandığı için önce ağaç veri yapısı anlatılacak sonra algoritmalar anlatılırken veri yapılarının farklılıklarından bahsedilecektir. Her iki algoritma da her bir zaman adımında ağaç veri yapısını oluşturup işlemlerini bu ağaç veri yapısı yardımıyla gerçekleştirir. Bir sonraki zaman adımında ağaç veri yapısını tekrar oluşturulur.

Cisimleri hiyerarşik olarak gruplayarak, bu grupları ağaç veri yapısında saklayan ve aralarındaki etkileşimi bu ağaç veri yapısı yardımıyla hesaplayan ilk uygulama Andrew A. Appel [6] tarafından yazılmıştır. Bu algoritmada birbirlerine yakın olan cisimler gruplanarak, uzaktaki cisimlerle etkileşim yapılmıştır. Fakat gruplandırma işleminde kullanılacak geometrik şekillerin düzensiz boyutta ve şekillerde olmasından dolayı algoritmanın analizi ve gerçeklenmesi oldukça zordur. Ağaç yapısı oluşturulduktan sonra, bu yapı üzerinden yapılan işlemler her bir cismin kuvvet hesabı için gerekli işlemleri (dengeli bir ağaç yapısı elde edildiğinde) $O(N)$'den ortalama $O(\log N)$'e indirilmiştir. Fakat bu hız artışı düzensiz boyutlardaki şekillerden dolayı analiz edilmesi zor hataları da beraberinde getirmiştir.

3. AĞAÇ VERİ YAPISI

Barnes-Hut [1] ve Hızlı çok-kutup [2] algoritmaları benzer ağaç yapılarını kullanırlar. Bu yöntemlerde sistemdeki tüm cisimleri içine alan, iki boyutta bir kare, üç boyutta ise bir küp seçilir. Bu kare (küp) hiyerarşik olarak iki boyutlu sistem için dört, üç boyutlu sistem için sekiz eşit parçaya bölünür (Bkz. Şekil 3.1). Bu bölme işlemi hiyerarşik olarak her bir karede tek bir cisim kalana kadar yapılır. Bu işlem sonunda elde edilen karelerin ağaç veri yapısında her bir kare bir düğümde olacak şekilde ifade edilir. İki boyutlu sistemler için kullanılan ağaç veri yapısı dörtlü ağaç (quadtree), üç boyutlu sistemler için kullanılan ağaç veri yapısı ise sekizli ağaç (octree) olarak adlandırılır.



Şekil 3.1 İki boyutta hiyerarşik dilimleme

Ağaç yapısında kullanılan terimler açıklanırken anlatım kolaylığı açısından iki boyutlu problem ele alınacaktır. Dörtlü ağaç veri yapısında kök düğüm (root node) tüm sistemi içine alacak büyüklükte olan kareyi (hücreyi) temsil etmektedir. Bir düğüme karşılık gelen kare, her bir boyutta iki eşit parçaya bölünerek, dört alt-düğüm (çocuk düğüm) ve dört alt-kare (alt-hücre) elde edilir. Dörde bölünen kareye ebeveyn, bölümlenmiş olan bu dört küçük karenin her birine ise çocuk kare denir. Dörtlü ağaç yapısında bir ebeveyn düğümün en fazla 4 çocuk düğümü bulunabilir. Hiyerarşik olarak bölümleme işleminde kareler birbirlerini kesmezler. İçinde cisim bulunmayan düğümler ağaç yapısında yer almazlar. Bir cisim aynı seviyede bulunan karelerden sadece bir tanesinin içinde bulunabilir. İçinde bir tane cisim bulunan

kareler ağaç yapısında en alt seviyedeki düğümlerdir ve bu düğümlere yaprak düğüm denir.

Aynı seviyede bulunan bir karenin merkezi ile aralarındaki uzaklık karenin bir kenar uzunluğu kadar olan kareler komşu kare olarak adlandırılır.

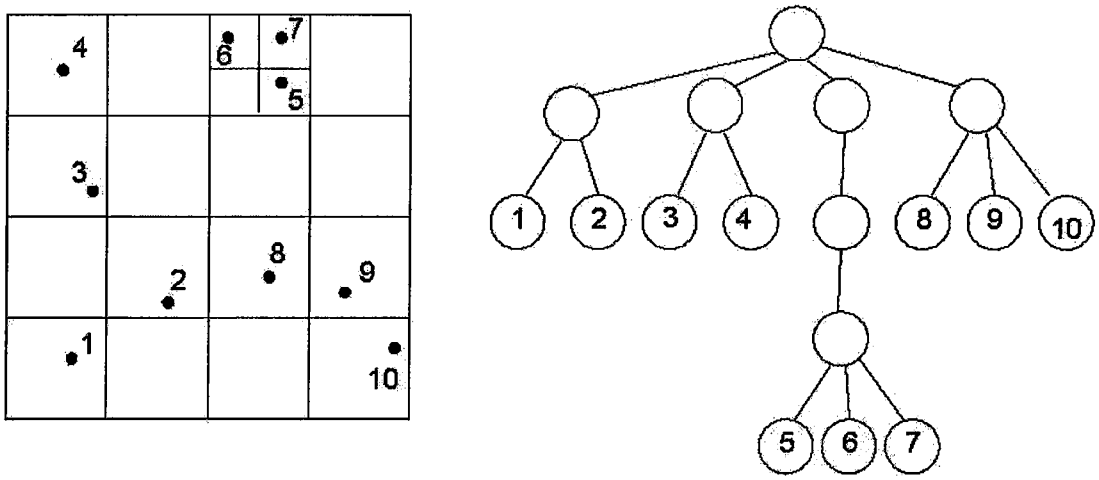
Seviye numaralandırmaları bu konuda yayınlanmış diğer makalelerde olduğu gibi sıfırdan başlatılmıştır. Bölümleme işlemi tamamlandığında 0. seviyede 4^0, 1. seviyede 4^1 ve l. seviyede en fazla 4^l tane kare elde edilir. Seviyesi l olan bir karenin (l+1). seviyede en fazla dört tane çocuk karesi vardır.

Bu tez boyunca kare ve hücre, alt-kare ve çocuk kare, alt-düğüm ve çocuk düğüm kelimeleri eşanlamlı olarak kullanılmaktadır.

3.1. Ağaç Yapısının Oluşturulması :

Ağaç veri yapısının oluşturulması için pek çok yöntem önerilmiştir. Bunlardan aşağıdan yukarıya ağaç oluşturma yönteminde birbirlerine yakın olan cisimler gruplanır ve bu gruplar hiyerarşik olarak tekrar tekrar gruplandırılarak işleme devam edilir. Yukarıdan aşağıya ağacı oluştururken, bütün cisimler öncelikle kök kareye yerleştirilir ve rekürsif olarak her hücrede bir cisim kalana kadar kareler bölümlenir. Bu yöntemde, kareye cisim yerleştirilir ve işlem sonunda karede bulunan cisim sayısı kontrol edilir. Eğer 1'den fazla cisim varsa, tek bir cisim kalana kadar bu kare dört alt kareye bölünür ve cisimler uygun karelere yerleştirilir. Bölümleme sonunda oluşan kareler ağaç veri yapısında kök düğümden başlanarak saklanır (Bkz. Şekil 3.2). Eğer 1 tane cisim bulunma şartı sağlanıyorsa bu kareyi tekrar bölümlemeye gerek kalmamıştır. İşlem sonunda karede cisim yoksa bu kare işleme alınmaz ve ağaç veri yapısında bu kareye karşılık gelen bir düğüm bulunmaz. Bu çalışmada gerçeklemesi daha kolay olan yukarıdan aşağıya ağaç oluşturma yöntemi kullanılmıştır.

Yaprak düğümlerde 1'den fazla cisim saklanacak şekilde algoritmalar genelleştirilebilir. Özellikle Hızlı çok-kutup algoritmasında bu sayının 1'den büyük olması algoritmanın çalışma zamanını ve yapılan hatayı düşürmektedir.



Şekil 3.2 İki boyutta hiyerarşik bölümleme ve dörtlü ağaç yapısı

Cisimlerin düzgün dağılması ağaç veri yapısının da dengeli olmasını sağlamaktadır. Her bir bölümleme sonucunda ebeveyn karede bulunan cisimler 4 çocuk kareye –yaklaşık- eşit bir şekilde dağılacaktır. Toplam cisim sayısı N, en alt seviyeye h dersek bu seviyede toplam 4^h tane kare olacaktır. Düzenli dağılmış bir sistem olduğunu kabul ettiğimizden N tane cisim 4^h tane kareye yerleştirilmiştir. Bu durumda h yaklaşık olarak $\log N$'dir. Sisteme bir cisim ekleyebilmek için ağaç üzerinde en fazla h seviye aşağı inilir. Bu durumda bir cismin ağaç üzerindeki yerini belirleyebilmek için $O(\log N)$ işlem yapılması gerekir. Tüm cisimlerin ağaç üzerine yerleştirilmesi $O(N \log N)$ işlemde tamamlanacaktır.

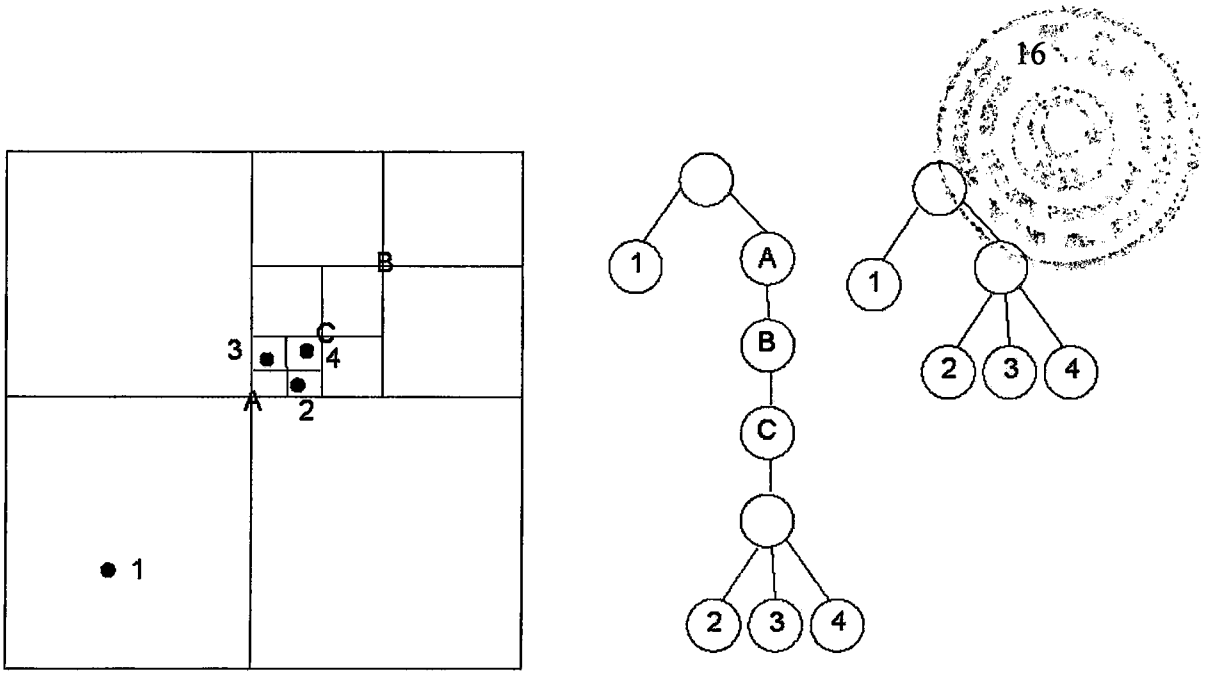
3.2. Sıkıştırılmış Sekizli Ağaç yapısı

Cisimlerin düzensiz bir dağılımı ağacın olmasına sebep olacaktır. Bu dengesizlik nedeniyle ağacın yüksekliği ve ağaçtaki düğüm sayısı için bir sınır vermek mümkün olmayacak, algoritmanın çalışma süresi de bunlara bağlı olarak $O(N \log N)$ olmayacaktır. Cisimlerin dağılımları düzensiz olduğunda sekizli ve

dörtlü ağaç yapısının üzerinden yapılan işlemler etkinliğini yitirmektedir. Cisimler ağaca yerleştirilirken bir kare içinde bulunan cisimler birbirinden farklı karelere düşene kadar özyineli (rekürsif) olarak kare, alt-karelerine bölümlenir. Düzensiz dağılımlarda cisimler pek çok bölümleme işlemi sonunda ancak farklı karelere ayrılabilirler. Bu ağaç yapısında dallanma olmayan uzun yollar (path) (Bkz. Şekil 3.3) olmasına neden olur. Barnes-Hut ve Hızlı çok-kutup algoritmaları bu tür veri yapısı kullandığından, cisimlerin düzensiz dağılımı algoritmaların performanslarını ve bellek karmaşıklıklarını olumsuz etkilemektedir. Cisimlerin dağılımları algoritmaların çalışma zamanlarını etkiliyorsa bu algoritmalara dağılıma bağlı algoritmalar denir. Eğer bir algoritmanın çalışma zamanı cisimlerin dağılımına bağlı değilse dağılımdan bağımsız algoritma olarak adlandırılır.

Böyle bir yol üzerindeki hücrelerin boyutları farklı olmasına rağmen hepsi aynı cisimleri içine almakta ve bu cisimlerle ilgili bilgileri tutmaktadır. Düzensiz dağılmış sistemlerde, dallanma olmayan uzun bir yol üzerindeki düğümleri hiçbir bilgi kaybına uğratmadan tek bir düğüme indirgeyebiliriz. Bu şekilde sekizli ağaçda bulunan her dallanma olmayan yolun sıkıştırılması sonucu elde edilen ağaç yapısına sıkıştırılmış sekizli ağaç denir. Bu sıkıştırma işlemi ilk olarak tüm-yakın komşular (all-nearest neighbour) problemi için Clarkson [24] tarafından ortaya atılmıştır. Clarkson N tane d-boyutlu nokta için $O(c^d NlogN)$ beklenen zamanda (c sabit bir sayıdır) ağaç oluşturan rastgele algoritma yayınlamıştır. Daha sonra Sevilgen [14] hem çalışma süresini O(dNlogN) zamana indiren hem de rasgele olamayan 3 değişik algoritma önermiştir. Bu tezde ilgili algoritmalardan aşağıdan-yukarıya ağaç oluşturulması anlatılmış ve gerçeklenmiştir.

Sıkıştırılmış ağaç yapısını oluşturabilmek için verilen iki hücreyi içine alacak kök düğümün en küçük-alt-hücresini sabit zamanda bulacak en küçük hücre tekniği kullanılmıştır. Cisimler sıfır uzunluğunda hücreler olarak düşünüleceğinden aynı teknik cisimler için de uygulanabilmektedir.



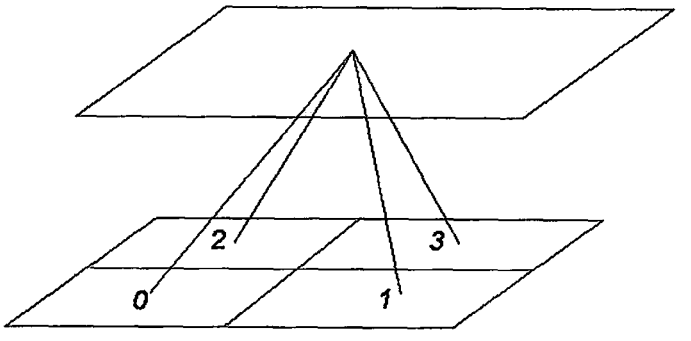
Şekil 3.3 Cisimlerin hiyerarşik bölümlenmesi, dallanma olmadan inilen yol (path) ve sıkıştırılmış dörtlü ağaç yapısı

Bu teknik ağaçtaki iki hücrenin en alt seviyeli ortak ebeveyn düğümünü bulmak için kullanılmaktadır. Bu tez kapsamında yazılan uygulamalarda bir ebeveyn hücrenin alt-hücreleri Şekil 3.4'deki gibi numaralandırılmıştır. Numaralandırma ebeveyn hücrenin merkezine bakılarak yapılmıştır. Bu numaralandırmaya göre eğer alt-hücrenin merkezinin konumu, ebeveyn hücrenin merkezinin konumundan d boyutunda büyük ise numaralandırmaya 2^d ilave edilir aksi durumda herhangi bir değer ilave edilmez.

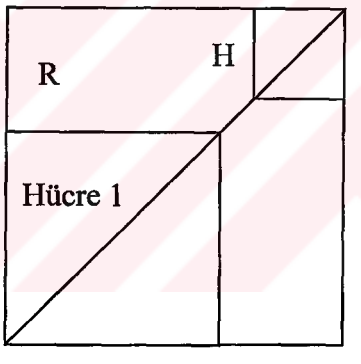
En küçük hücre tekniğinde, öncelikle verilen iki hücreyi de içine alacak en küçük dikdörtgenler prizması bulunur (Bkz. Şekil 3.5). Daha sonra bu dikdörtgeni içine alacak küçük hücreyi bulan algoritma, Algoritma 3.1 olarak verilmiştir.

Sıkıştırılmış ağaç veri yapısında dallanma olmadan bulunan bir yol üzerindeki düğümlerin yerine tek bir düğüm olması sağlamaktadır. Bu nedenle her bir düğüm, en az iki çocuk düğümü bulunan bir düğüm veya bir yaprak düğümdür. Sıkıştırılmış ağaç yapısında her bir v düğümü için iki hücre tanımlanmıştır: Dallanma olmayan bir yolun ilk düğümüne karşılık gelen kareye büyük hücre ve yol üzerinde bulunan son düğüme karşılık gelen kareye küçük hücre denir. Büyük hücre L(v), küçük hücre ise

S(v) ile gösterilir. Herhangi bir düğüm için küçük hücre, büyük hücre içinde bulunan tüm cisimleri içine alan en küçük alt-karedir. Bu sebeple, herhangi bir v iç düğümün küçük hücresi dört eşit alt-hücreye bölünürse boş olmayan en az iki alt-hücre olmalıdır. Bu bölümleme sonundaki boş olmayan her bir C alt-hücre için küçük hücresi C olan bir çocuk düğüm vardır. Yaprak düğümün en küçük hücresi cismin konumunda sıfır uzunluğunda bir hücredir.



Şekil 3.4 Bir hücrenin çocuk hücrelerinin numaralandırması



Şekil 3.5 İki hücrenin köşeleri birleştirilerek oluşturulan R hücresi

Ağacın aşağıdan yukarıya oluşturulması için öncelikle cisimlerin sıralanması gereklidir. Bu sıra aynı zamanda cisimler ağaç yapısında soldan sağa bulunma sırasıdır. Cisimlerin sıralanması için öncelikle iki cismin nasıl karşılaştırılabileceği anlatılacaktır: Verilen p_1 ve p_2 cisimleri için, bunları içine alan en küçük hücre Algoritma 3.1'deki teknik yardımıyla hesaplanır. Hesaplanan bu hücreyi alt-hücrelerine böldüğümüzde p_1 ve p_2 cismi farklı alt-hücrelere düşeceklerdir. Bu alt-hücrelerin numaralandırmasıyla cisimler sıralanmış olur.

Algoritma En_küçük_hücre

Girdi R, iki hücreyi içine alan dikdörtgenler prizmasının koordinatları
Çıktı: l_c, en küçük hücrenin bir kenarı
KH, en küçük hücrenin konumu

For each i boyutu

min = R'nin i boyutundaki en küçük koordinatı

max = R'nin i boyutundaki en büyük koordinatı

$$t = \left\lceil \log_2 \frac{1}{x_{max} - x_{min}} \right\rceil$$

$$a_1 = \left\lceil \frac{2^t x_{min}}{1} \right\rceil \text{ ve } a_2 = \left\lceil \frac{2^t x_{max}}{1} \right\rceil$$

Eğer $a_1 = a_2$ eşit ise

$$a' = a_2$$

Değilse

a'= a_1 ve a_2'den çift olan tamsayıdır

$$s = t - \log_2(1 + \{a' \oplus (a' - 1)\}) + 1$$

k = s değerlerinden en küçüğü

$$l_c = \frac{1}{2^{k-1}}$$

For each i

KH i boyutu için konumu = $\left\lfloor \frac{min}{l_c} \right\rfloor + \frac{l_c}{2}$

End for i;

Algoritma 3.1 **En küçük hücre bulma tekniği**

Verilen N cisim için ağaç şu şekilde oluşturulur: Tüm cisimleri içine alabilecek en küçük kare kök hücre olarak belirlenir. Cisimler optimum bir sıralama algoritması yardımıyla $O(dN\log N)$ zamanda sıralanır. Sıralanmış cisimler teker teker yerleştirilerek ağaç yapısı oluşturulur. Ağaca cisimlerin yerleştirilmesi sırasında en son yerleştirilen düğümün kaydı tutulur. Bir sonraki yerleştirilecek cismin p olduğunu düşünelim. En son eklenen düğümden başlanarak kök düğüme doğru $p\in L(v)$ olacak ilk düğüm, v düğümü bulunur. Bu noktada iki ihtimal vardır:

Eğer p, S(v) içinde değilse, p (L(v)- S(v)) içindedir. p ve S(v)'yi içine alan en küçük hücre L(v)'nin bir alt-hücresidir ve p ile S(v) bu en küçük hücrenin farklı alt-hücreleri içinde yeralır. Bu durumda yeni bir u düğümü v ile ebeveyni arasında oluşturulur ve p u düğümünün bir çocuğu olarak eklenir.

Eğer p, S(v) içinde ise, p v'nin çocuk düğümü olarak ilave edilir.

Ağacı oluşturmak için tek bir cismin yerleştirilmesi $O(1)$ zamanda olmaz fakat tüm N tane cismin yerleştirilmesi için toplam $O(N)$ zamana ihtiyaç vardır. Cisimlerin sıralanması ile beraber ağacın oluşturulması $O(N\log N)$ zaman almaktadır. Ağacın oluşturulma zamanı cisimleri sıralamak için kullanılan algoritmanın zamanı ile belirlenir. Performans yüzünden hızlı-sıralama (quick-sort) algoritması tercih edilmelidir fakat ikinci adımdan itibaren cisimlerin sıralaması çok fazla değişmeyeceğinden quicksort yerine insertion-sort algoritması kullanılırsa daha iyi çalışma zamanı elde edilebilir..

Algoritma Ağaç_Oluşturma
Girdi N tane cisim **Çıktı** Sıralanmış N tane cisim
Tüm cisimleri içine alacak kök düğümü bul. Cisimleri sıkıştırılmış ağaç yapısına göre sırala. Sıralanmış cisimlerden ilk p cismi için *v* düğümünü oluştur. Eklenecek cisim olduğu sürece devam et Sıradaki p cismi için yeni bir *w* düğümü oluştur p, L(*v*) içinde olmadığı sürece devam et *v*=ebevyn(*v*) IF p, S(*v*) içinde değil ise P ve S(*v*)'yi içine alacak küçük düğüm C'nin L(*v*)'sini hesapla *V* düğümü yerinde *u* düğümünü oluştur. L(*u*)= L(*v*); S(*u*)=C L(*v*)ve L(*w*) C'nin alt-hücreleri olarak ata. *v* ve *w* düğümlerini *u* düğümünün çocuk-düğümleri olarak ata. ELSE IF p∈S(*v*)ise L(*w*) p'yi içine alan S(*v*)'nin alt-hücresi yap. *w* düğümünü *v* düğümünün çocuk-düğümü olarak ata. v=w;

Algoritma 3.2 Aşağıdan yukarıya ağaç oluşturma algoritması

4. BARNES-HUT ALGORİTMASI

Barnes-Hut algoritması 1985 yılında Appel [6] tarafından yayınlanan algoritmayı temel alarak 1986 yılında [1] yayınlanmıştır. Barnes-Hut, Appel'in algoritmasında olduğu gibi her bir cismin üzerindeki kuvvet hesabını düzenli bir dağılım olduğunda ortalama O(logN) zamanda yapabilen bir algoritma geliştirmiştir. Barnes-Hut'un algoritmasının Appel'in algoritmasına göre sağladığı avantajları, ağaç yapısındaki karışıklıkların engellemesi ve hata için üst sınır verip ortalama bir hata hesabı yapmayı sağlamasıdır.

Barnes-Hut algoritması cisimleri sekizli ağaç yapısı kullanarak gruplandırmakta ve ağaç yapısındaki her kare için karenin içinde bulunan cisimlerin toplam kütlelerini ve kütle merkezlerini hesaplamaktadır. Bir karenin içindeki bütün cisimler bu kütle merkezinde toplanmış toplam kütle kadar büyüklükte sanal bir cisim olarak kabul edilir. Kütle merkezindeki bu sanal cisim ile yeterince uzakta olan cisimlerle etkileşim yapılır. Buna cisim-küme etkileşimi denir. Böylece kare içindeki cisimlerin her birisi ile etkileşim yapmak yerine tek bir cisim ile etkileşim yapılarak işlem sayısı azaltılmış olur. Bu hız artışı beraberinde küçük de olsa hatayı da getirecektir. Bu hatanın küçük kalabilmesi için cisim ve kümenin birbirlerinden yeterince_uzakta bulunmaları gerekir. Cisim ile kümenin kütle merkezi arasındaki uzaklık olan d, karenin bir kenarının uzunluğu olan l 'den büyük olmalıdır. Barnes-Hut l/d oranını bir Θ parametresi ile kontrol etmektedir. Cisim ile küme, l/d oranı Θ parametresinden büyük ise yeterince_uzakta kabul edilir. Θ parametresi 0 ile 1 arasında olmalıdır ve genellikle 0.5 seçilir. Çok küçük Θ parametresi seçilmesi durumunda algoritma $O(N^2)$ 'ye yaklaşır çünkü yapılan cisim-küme etkileşim sayısı azalacaktır. Θ parametresinin 1'e yaklaşması durumda ise yapılan hata büyük olmaktadır. Kuvvet uzaklığın karesi r^2 ile ters orantılı olduğundan yakın cisimlerin etkisi daha büyüktür ve yakın cisimleri için r değerinin yerine kütle merkezini uzaklığının yani d'nin alınması daha fazla hataya sebep olmaktadır.

Barnes-Hut algoritması 3 ana adımdan oluşur [1] :

- AğacıOluştur()
  Sekizli (octtree) ağaç yapısını hiyerarşik olarak oluştur. (Bkz. Bölüm 3.1)

- AşağıdanYukarıya()
  Yapraklardan kök düğüme doğru gidilerek her bir kümenin kütle merkezleri ve toplam kütlelerini hesapla.

- YukarıdanAşağıya()
  Her bir cisme etki eden kuvveti ağaç üzerinde yukarıdan aşağıya (kök düğümden yapraklara) doğru hesapla.

4.1. Aşağıdan-Yukarıya:

Cisim küme etkileşiminin yapılabilmesi için gerekli olan bilgiler kümenin kütle merkezinin koordinatları ve küme merkezindeki sanal cismin kütlesidir. Bu adımda tüm düğümler için bu bilgiler hesaplanır. Hesaplama işlemi için önce yaprak düğümlere inilir. Yaprak düğümlerde birden fazla cisim varsa bunların kütle merkezi ve toplam kütleleri hesaplanır. Bu bilgiler ağaç yapısında saklanır. Hesaplanmış bu bilgiler kullanılarak daha yukarıdaki kümelerin toplam kütleleri ve kütle merkezlerini hesaplanır. Bu hesaplama ağacın kök-sonra dolaşımı (post-oder traversal) kullanılarak yapılabilir.

4.2. Yukarıdan Aşağıya:

Bu adımda tüm cisimlerin kuvvetleri ağaç veri yapısı yardımıyla hesaplanır. Her bir cisim için ağaç üzerinde kök düğümden başlanıp yaprak düğümlere inilir. Her bir düğüm ile cismin yeterince_uzakta olma şartı kontrol edilir. Eğer bu şart sağlanıyorsa cisim ile bu düğümün cisim-küme etkileşimi hesaplanır ve bu düğümün alt düğümlerine inilmez. Eğer şart sağlanmıyorsa bu düğümün alt düğümlerinin her birisi için tekrar bu şart kontrol edilir ve işleme devam edilir. Yaprak düğüme kadar inildiği halde, yeterince_uzakta şartı herhangi bir yaprak düğüm için sağlanmıyorsa, bu yaprak içinde bulunan cisimlerle cisim-cisim etkileşimi yapılması gerekmektedir.

Bir cisim üzerindeki kuvveti hesaplayabilmek için ağaç üzerinde kök düğümden başlanarak en alttaki düğüme kadar inilir. Eğer cisim ile bu düğüm arasında yeterince_uzakta şartı sağlanıyorsa cisim ile küme etkileşimi yapılır.

Bölüm 3.1'de belirtildiği gibi cisimler düzgün olarak dağıldığında ağacın yüksekliği $O(N \log N)$'dir. Böylece bir cisim diğer $N-1$ tane cisim ile doğrudan hesaplama yapmak yerine ağaç üzerinde en fazla $O(\log N)$ seviye aşağıya inerek etkileşim yapılabilir. Düzenli bir dağılım olduğunda cisimlerin yakın komşu sayısının az, uzak komşu sayısının çok olacaktır.

Ağaç oluşturması $O(N \log N)$ zamanda, yukarıdan aşağıya ortalama $O(N)$ zamanda ve kuvvet hesabı $O(N \log N)$ zamanda yapılacaktır. Bu durumda toplam çalışma zamanı

$$O(N \log N + N + N \log N) = O(N \log N)$$

olacaktır.

Kuvvetler hesaplandıktan sonra cisimler yeni konumlarına konumlandırılır ve bir sonraki zaman aralığına geçilir. Ağaç tekrar oluşturulur ve tüm hesaplamalar tekrar yapılır.

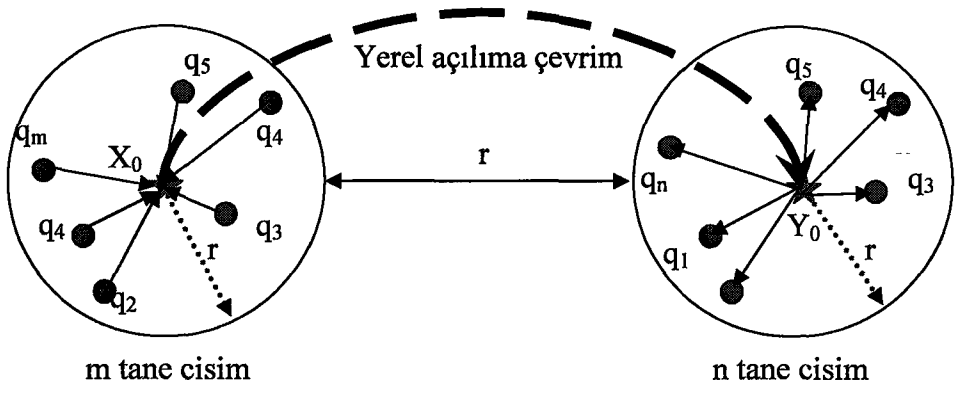
5. HIZLI ÇOK-KUTUP ALGORİTMASI

Hızlı çok-kutup Algoritması (HÇA) Greengard ve Rokhlin tarafından geliştirilmiştir [2][3]. 20. yüzyılın en popüler 10 algoritması arasında gösterilmektedir. HÇA cisim-küme etkileşimi yapmak yerine küme-küme (Bkz. Şekil 2.1) etkileşimi yapar. Küme-küme etkileşimi yaparken Taylor serisi açılımı yardımıyla potansiyelleri hesaplayarak kuvvetleri yaklaşık olarak hesaplar. Algoritma düzenli dağılmış N tane cismin bulunduğu bir sistemde cisimler arasındaki etkileşimleri (kuvvet hesabı) $O(N)$ zamanda yapmaktadır.

Hızlı çok-kutup algoritmasında iki temel seri açılım kullanılır: Bir daire içinde bulunan cisimlerin, bu dairenin dışında (yeterince uzakta) bulunan cisimler üzerine uyguladığı kuvvetin hesaplamasını sağlayan Taylor serisi benzeri açılıma çok-kutup açılımı denir. Bu açılım bir kere hesaplanarak, istenilen noktadaki cisimler için hesaplamalar bu seri açılımı yardımıyla yapılabilir. Bu açılımın tersi de mümkündür. Bir dairenin dışında yeterince uzakta bulunan tüm cisimlerin dairenin içindeki cisimler üzerine uyguladığı kuvveti hesaplamayı sağlayacak seri açılımına yerel açılım denir. Hızlı çok-kutup algoritmasındaki hız artışı bu yerel açılımlar yardımıyla sağlanmaktadır. Algoritmada kullanılan seri açılımlar, sonsuz sayıda terim içermektedir. Fakat pratikte, açılımı bilgisayar yardımıyla hesaplayabilmek için p tane terim alınır ve p terimden sonsuza kadar olan terimler toplamı hatayı göstermektedir. Bu sebeple p sayısı büyük seçilerek daha doğru sonuçlar elde edilebilir.

HÇA[2] 2-boyutlu uzayda, düzenli dağılmış cisimler için yüksek doğruluk oranlarında etkin çözüm sağlamaktadır fakat düzensiz dağılımlar olduğunda çalışma zamanı artmaktadır. 3-boyutlu uzayda ise seri açılımları üzerindeki işlemler [18], arttığından algoritma etkinliğini yitirmektedir. Cheng [19] ise düzenli ve düzensiz dağılmış cisimler için üç boyutlu uzayda düzlem dalgası denklemleri (plane wave equations) kullanarak etkin çözüm sağlayan matematiksel dönüşümler önermiştir. Fakat bu algoritmanın gerçeklenmesi oldukça zor ve karmaşıktır.

Hızlı çok-kutup algoritması hiyerarşik olarak sistemi karelere bölerek, bunları ağaç veri yapısında saklar ve hesaplamaları bu ağaç yapısı yardımıyla yapar. Hesaplamalar için hiyerarşik olarak bölümlenmiş karelerin merkezleri kullanılır. Çok-kutup açılımının sağladığı hız artışı ve algoritmanın ana hatları incelenecektir:



Şekil 5.1 Çok kutup açılımının merkezinin kaydırılması ve yerel açılıma çevrilmesi

Şekil 5.1'de görüldüğü gibi X_0 merkezli r yarıçaplı X dairesi içinde q_1, q_2, q_3,..., q_m yüklere sahip m tane cisim, Y_0 merkezli r yarıçaplı Y daire içinde q_1, q_2, q_3,..., q_n yüklere sahip n tane cisim olsun. X ve Y kümelerinin yeterince uzakta olduğunu söyleyebilmek için x_0 ve y_0 merkezleri arasındaki uzaklığın 3r'den büyük eşit olması gerekir.

X kümesindeki cisimlerin Y kümesindeki cisimlere uyguladığı potansiyeller doğrudan hesaplanmak istenirse toplam $O(nm)$ işlem yapılması gereklidir.

Hızlı çok-kutup yöntemi kullanılarak X daire içindeki q_1, q_2, q_3,..., q_m yüklerinin p terimli çok-kutup açılımının katsayıları hesaplanabilir. Bu işlem m tane cisim için mp işlem gerektirir. Bu açılımı kullanılarak Y kümesinin merkezi Y_0 noktasındaki yerel açılım elde edilir (Bkz. Bölüm 5.5.1). Bu yerel açılım elde edildikten sonra Y kümesi içindeki tüm cisimlerin potansiyelleri bu yerel açılımdan hesaplanabilir. Bu hesaplama için toplam $O(np)$ işlem gereklidir. Dönüşüm işlemi

dışında tüm işlemler için toplam $O(mp+np)$ işlem gereklidir. m ve n sayısı çok büyüdüğünde $O(mp+np)$ işlem $O(nm)$ 'den çok daha kısa zaman alacaktır.

Hızlı çok-kutup algoritmasının çalışma zamanı olan $O(N)$ karmaşıklığın içindeki sabit sayı oldukça büyüktür. Bu nedenle Hızlı çok-kutup Algoritması düşük cisim sayısı veya düşük doğruluk oranlarında Barnes-Hut algoritmasına göre oldukça yavaştır. Ancak cisim sayısı büyük olduğunda veya yüksek doğruluk oranı istendiğinde Hızlı çok-kutup Algoritması tercih edilmelidir.

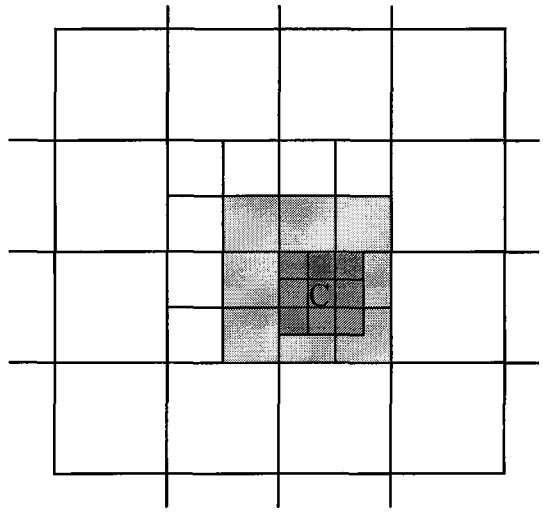
Hızlı çok-kutup algoritmasının Barnes-Hut algoritmasından farklı yönleri şunlardır [17]:

Barnes-Hut algoritmasındaki gibi her cisim üzerindeki kuvvetler değil potansiyeller hesaplanır. Vektörel bir büyüklük olan kuvvet yerine skaler bir büyüklük olan potansiyel kullanımı ile hesaplamalar kolaylaşmaktadır. Potansiyel daha sonra kuvvete çevrilmektedir.

Küme merkezi ve kütle bilgileri yerine seri açılımlar kullanılır. Bu açılımda daha fazla terim alarak daha hassas hesaplamalar yapılabilmektedir. Daha fazla terim alma, işlem zamanını ve bellek kullanımını artıracaktır.

Barnes-Hut algoritmasından farklı olarak Hızlı çok-kutup Algoritması, etkileşime girecek kümelerin sayısını 2 boyutta 27, 3 boyutta 128 sayıları ile sınırlandırmıştır. [2].

Hızlı çok-kutup Algoritması anlatılırken elektrostatik problem ele alınacaktır. Fakat yerçekimsel kuvvet hesabı veya elektrostatik problem için aynı formüller kullanılabilir. Sadece kuvvetin yönünde değişiklik olmaktadır.



Şekil 5.2 İki boyutta C karesinin komşuları (koyu-gri) ve etkileşim listesi(açık-gri)

5.1. Tanımlar

Bu bölümde Hızlı çok-kutup yönteminde kullanılan tanımlardan bahsedilecektir. Algoritmanın anlaşılmasında en önemli nokta yeterince uzakta (well seperate) kriterinin anlaşılmasıdır. Bitişik ve aynı boyutta olan iki kare komşu kare olarak adlandırılır (Bkz. Şekil 5.2). Bir C karesinin yakın komşuları iki boyutta 8, üç boyutta 27 tanedir. C karesinin ebeveyn karesinin yakın komşularının çocuklarından, C ile yakın komşu olmayan kareler, etkileşim listesindeki karelerdir (Bkz. Şekil 5.2).

5.2. Hızlı Çok-Kutup Algoritması

Hızlı çok-kutup algoritması temelde Barnes-Hut algoritmasının adımları ile aynı adımlara sahiptir.

- AğacıOluştur()
  Sekizli (octtree) ağaç yapısını hiyerarşik olarak oluştur. (Bkz. Bölüm 3.1)

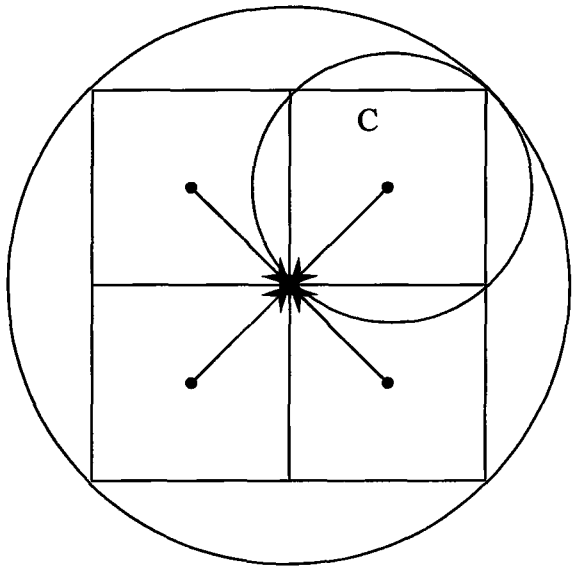
- AşağıdanYukarıya()
  Yapraklardan kök düğüme doğru gidilerek her bir kümenin çok-kutup açılımları hesaplanarak, ebeveyne kaydırılır.

- YukarıdanAşağıya()
  Çok-kutup açılımları yerel açılımlara çevrilerek, çocuk kümelere kaydırılır.

- KuvvetHesapla()
  Yerel açılımlar ve yakın komşular için doğrudan hesaplama yapılarak kuvvet hesaplanır.

Ağaç üzerinde, aşağıdan yukarıya hesaplama adımında karelerin merkezlerine göre çok-kutup açılımları hesaplanır. Çok-kutup açılımını φ ile gösterelim. Önce tüm yaprak düğümlerdeki cisimler kullanılarak çok-kutup açılımı hesaplanır. (Daha önce belirtildiği gibi çok-kutup açılımları karenin merkezine göre verilmektedir.) Sonra bu düğümlerin ebeveyn düğümleri için çok-kutup açılımları hiyerarşik olarak hesaplanır. Bu hesaplama işlemi için her bir çocuk düğümün çok-kutup açılımının merkezi, ebeveyn düğümün merkezine kaydırılır ve ebeveynin φ değerine ilave edilir (Bkz. Şekil 5.3). Bu işleme kök düğüme kadar devam edilir.

Yukarıdan aşağıya inilen adımda ise yerel açılımlar hesaplanır. Yerel açılımı Ψ ile gösterelim. Yerel açılımın hesaplanması için karenin etkileşim listesinde bulunan karelerdeki φ çok-kutup açılımları, yerel açılıma çevrilerek Ψ açılımına ilave edilir. Yakında bulunan kareler, yerel açılım hesaplanırken bu hesaplamaya ilave edilmez. Tüm sistemi içine alan bir kök düğüm düşündüğümüz için bu karenin yerel açılımından bahsedemeyiz çünkü bu kök düğümün etkileşim listesi boştur. Kök düğümün alt düğümlerinin de yakın komşularından başka düğümler bulunmadığından bu alt düğümlerinde etkileşim listesi boştur. Yerel açılım hesaplaması işlemi ağacın 2.seviyesinden başlar. Bir seviyedeki tüm düğümler için yerel açılımlar hesaplanır ve bu yerel açılımlar çocuk düğümlerin merkezlerine kaydırılır. Bir kare için yerel açılım hesabı şu şekilde yapılır:



Şekil 5.3 Çok-kutup açılımının merkezinin kaydırılması

A karesinin yerel açılımını hesapladığımızı düşünelim. Bu A karesinin ebeveyni B karesi olsun. B'nin yerel açılımı, tüm çocuk karelerin merkezine kaydırılır. Böylece A karesi için yerel açılım elde edilmiş oldu fakat bu yerel açılım, B karesinin yeterince uzakta bulunan tüm cisimlerin A karesi içindeki cisimlere olan potansiyel etkisini içermektedir. Fakat A karesinin etkileşim listesinde B karesinin yerel açılımda olmayan kareler vardır. Bu etkileşim listesindeki tüm karelerdeki çok-kutup açılımlar, A karesinin yerel açılımı olarak çevrilir ve B karesinden gelen yerel açılıma eklenir.

Yukarıdan aşağıya hesaplama sonunda, yaprak düğümler de dahil her bir düğüm için yerel açılım hesaplanmış olur. Yaprak düğümlerdeki yerel açılımlar bu düğümde bulunan cisimlerin konumlarına kaydırılarak, cisimler için potansiyel değerleri hesaplanır. Yalnız bu potansiyel, sadece cismin içinde bulunduğu yaprak düğümden yeterince_uzaktaki düğümlerde bulunan cisimlerin etkilerini içermektedir. Yaprak düğümün yakın komşularında bulunan cisimlerin etkilerini içermemektedir. Yaprak düğümün yakın komşularında bulunan cisimlerin etkileri ise cisim-cisim etkileşimi kullanılarak hesaplanır. Böylelikle her bir cisim için toplam potansiyel

bulunmuş olur. Daha sonra potansiyel önce kuvvete çevrilir, daha sonra bu kuvvet kullanılarak cisimlerin hızları ve konumları değiştirilir.

5.3. Matematiksel Altyapı[22]

Üç boyutta $C_0=(x_0,y_0,z_0)\in R^3$ noktasındaki q yüklü bir cismin C_0 dışında herhangi bir C=(x,y,z) noktasındaki potansiyeli ve elektrostatik alanı sırasıyla

$$\Phi = \frac{1}{\tilde{r}} \tag{11}$$

$$\vec{E} = -\nabla\Phi = \left(\frac{x-x_0}{\tilde{r}^3}, \frac{y-y_0}{\tilde{r}^3}, \frac{z-z_0}{\tilde{r}^3} \right) \tag{12}$$

olarak tanımlanmıştır [22][19]. Burada $\tilde{r}$, C_0 ile C arasındaki uzaklığı göstermektedir. C noktasındaki potansiyel hesabını yapmamızı sağlayacak bir seri açılımı elde edilecektir. Kaynak noktalar dışında potansiyel Φ harmoniktir yani Laplace denklemini sağlar :

$$\nabla^2\Phi = \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial z^2} = 0 \tag{13}$$

C noktasının küresel koordinatı (r,θ,φ) ve C_0 noktasının küresel koordinatı (ρ,α,β) olsun. C ve C_0 vektörleri arasındaki açının γ olduğunu kabul edelim. Kosinüs teoreminden dolayı :

$$\tilde{r}^2 = r^2 + \rho^2 - 2r\rho\cos\gamma \tag{14}$$

$$\cos\gamma = \cos\theta\cos\alpha + \sin\theta\sin\alpha\cos(\phi-\beta) \tag{15}$$

elde edilir. Böylece

$$\frac{1}{\tilde{r}} = \frac{1}{r\sqrt{1-2\frac{\rho}{r}\cos\gamma+\frac{p^2}{r^2}}} = \frac{1}{r\sqrt{1-2u\mu+\mu^2}} \tag{16}$$

$$\mu = \frac{\rho}{r} \quad \text{ve} \quad u = cos\gamma \tag{17}$$

yazılabilir. μ<1 için μ kuvvet serisi şeklinde açabilir :

$$\frac{1}{\sqrt{1-2u\mu+\mu^2}} = \sum_{n=0}^{\infty} P_n(u)\mu^n \tag{18}$$

$$P_0(u)=1 \ , \ P_1(u)=u \ , \ P_2(u)=\frac{3}{2}(u^2-\frac{1}{3}) \ ,..... \tag{19}$$

Genel olarak $P_n(u)$ n. dereceden Legendre polinomudur. Açılımımız şu şekle gelmiştir:

$$\frac{1}{\tilde{r}} = \sum_{n=0}^{\infty} \frac{\rho^n}{r^{n+1}} P_n(u) \tag{20}$$

Fakat (20)'deki formül istenilen şekilde değildir çünkü u parametresi hem kaynak noktalara hem de hedef noktalara bağlıdır. Sadece hedef noktalara bağlı bir açılım elde edebilirse, bu açılım bir kere hesaplanarak, istenilen noktalardaki potansiyelleri hesaplamak için kullanılabilir. Bunun için önce Laplace denklemini küresel koordinatlarda yazalım :

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial\Phi}{\partial r}\right)+\frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Phi}{\partial\theta}\right)+\frac{1}{r^2\sin^2\theta}\frac{\partial^2\Phi}{\partial\phi^2}=0 \tag{21}$$

Bu denklemin değişkenlerin ayrılarak standart çözümü terimlerinin küresel harmonik olarak bilinen seri açılımının elde edilmesini sağlar :

$$\Phi = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \left(L_n^m r^n + \frac{M_n^m}{r^{n+1}} \right) Y_n^m(\theta,\phi) \tag{22}$$

$Y_n^m(\theta,\phi)r^n$ terimleri n. dereceden küresel harmonik olarak, $Y_n^m(\theta,\phi)/r^{n+1}$ (-n-1). derecen küresel harmonik, L_n^m ve M_n^m katsayıları ise açılımın momentleri olarak bilinir.

Küresel harmonikler $1/r$ nin parçalı türevinin terimleri, $\sqrt{(2n+1)/4\pi}$ normalizasyon faktörü ihmal edildiğinde :

$$Y_n^m(\theta,\phi) = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} . P_n^{|m|}(\cos\theta) e^{im\phi} \tag{23}$$

elde edilir. P_n^m özel fonksiyonu Legendre fonksiyonu olarak adlandırılır ve Rodrigues formülü ile tanımlanır:

$$P_n^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_n(x) \tag{24}$$

5.4. Çok-Kutup Açılımı

C ve Q noktaları küresel koordinatlarda sırası ile (r,θ,φ) ve (ρ,α,β) bulunsun ve aralarındaki açı γ olsun. Bu durumda

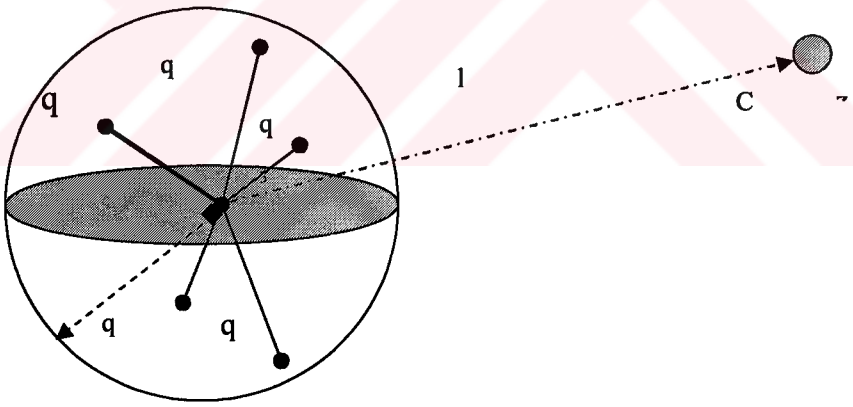
$$P_n(\cos\gamma) = \sum_{m=-n}^{n} Y_n^{-m}(\alpha,\beta).Y_n^m(\theta,\phi) \qquad (25)$$

ile hesaplanır. Şekil 5.4'daki gibi α yarıçaplı bir daire içinde bulunan k tane $\{q_i, i=1,\ldots,k\}$ yüklü cismin $\{Q_i=(\rho_i,\alpha_i,\beta_i), i=1,..,k\}$ noktalarında olduğunu kabul edelim. Böylece $r>\alpha$ şartını sağlayan herhangi bir $C=(r,\theta,\varphi)\in R^3$ noktası için $\phi(P)$ potansiyeli

$$\phi(P) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}}.Y_n^m(\theta,\phi) \qquad (26)$$

$$M_n^m = \sum_{i=1}^{k} q_i.\rho_i^n.Y_n^{-m}(\alpha_i,\beta_i) \qquad (27)$$

şeklinde yazılır.



Şekil 5.4 Çok-kutup Açılımı

Ayrıca seri açılımda alınan terim sayısını gösteren herhangi bir $p\geq 1$ değeri için

$$\left|\phi(P)-\sum_{n=0}^{p}\sum_{m=-n}^{n}\frac{M_n^m}{r^{n+1}}.Y_n^m(\theta,\phi)\right|\le\frac{A}{r-a}\left(\frac{a}{r}\right)^{p+1} \quad (28)$$

$$A=\sum_{i=1}^{k}|q_i| \quad (29)$$

şeklinde hata sınırlandırılmış olur.

Böylece Şekil 5.4'deki gibi belli bir r yarıçapı içinde bulunan n tane cisim için bu dairenin dışında istenilen herhangi bir noktadaki potansiyeli hesaplamamıza yarayacak bir seri açılımı elde edilmiş oldu. Serinin p terimden sonrası alınmadığı için yaklaşık bir çözüm elde edilecek fakat yapılacak bu hata önceden hesaplanabilecektir. Bu bize istenilen duyarlılığa (doğruluğa) göre p 'yi belirleme fırsatı sağlamaktadır. Seride bulunan M_n^m ifadesini incelediğimizde bu ifadenin sadece kaynak noktalara bağlı olduğunu görürüz. Bu katsayılar bir kere hesaplanarak tüm hedef noktalar için kullanılabilir. [2].

5.5. Dönüşüm Operatörleri Ve Hata Sınırları

Yukarıda anlatıldığı gibi çok-kutup açılımı katsayıları sadece kaynak noktalardaki cisimlere bağlıdır. Yani istenilen bir hedef nokta için potansiyel hesabı yapılabilir. Fakat Hızlı çok-kutup algoritmasında asıl elde edilmek istenen açılım yerel açılımdır. Bu bölümde Hızlı çok-kutup algoritmasında kullanılan çok-kutup açılımının merkezinin başka bir noktaya kaydırılması, çok-kutup açılımının yerel açılıma çevrilmesi ve yerel açılımının merkezinin başka bir noktaya alınması için gerekli dönüşüm işlemleri anlatılacaktır.

5.5.1. Çok-kutup Açılımının Merkezinin Kaydırılması

Q=(ρ,α,β) merkezli α yarıçaplı D dairesi içindeki l tane $q_1,q_2,\ldots,q_l$ yüklü cisimler ve D dairesi dışındaki C=(r,θ,φ) noktaları için bu cisimlerden dolayı oluşan potansiyel hesabını veren çok-kutup açılımı:

$$\Phi(P)=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{O_n^m}{r^{n+1}}.Y_n^m(\theta,\phi) \tag{32}$$

burada C-Q=(r',θ',φ')'dir. Böylece merkezi orijinde olan $(\alpha+\rho)$ yarıçaplı D_1 dairesi dışındaki herhangi bir C=(r,θ,φ) noktası için

$$\Phi(P)=\sum_{i=0}^{\infty}\sum_{k=-i}^{i}\frac{M_i^k}{r^{i+1}}.y_i^k(\theta,\phi) \tag{33}$$

şeklinde tanımlanır. Burada

$$M_i^k=\sum_{n=0}^{i}\sum_{m=-n}^{n}\frac{O_{i-n}^{k-m}.i^{|k|-|m|-|k-m|}.A_{i-n}^{k-m}.\rho^n.Y_n^{-m}(\alpha,\beta)}{A_i^k} \tag{34}$$

$$A_n^m=\frac{(-1)^n}{\sqrt{(n-m)!(n+m)!}} \tag{35}$$

şeklinde tanımlanmıştır. (34) yardımıyla O_k^j çok-kutup açılımı katsayıları M_i^k katsayılarına çeviren lineer operatör tanımlanmış oldu. Herhangi p≥1 değeri için hata şu şekilde sınırlandırılmış olur:

$$\left|\Phi(P)-\sum_{i=0}^{p}\sum_{k=-i}^{i}\frac{M_i^k}{r^{i+1}}.Y_i^k(\theta,\phi)\right|\le\left(\frac{\sum_{i=1}^{l}|q_i|}{r-(a+\rho)}\right)\left(\frac{a+\rho}{r}\right)^{p+1} \quad (36)$$

5.5.2. Çok-kutup Açılımının Yerel Açılıma Çevrilmesi

Q=(ρ,α,β) Merkezli α yarıçaplı D_Q dairesi içindeki l tane $q_1,q_2,\ldots,q_l$ yüklü cisimler ve c>1 şartı ile ρ>(c+1)α olduğunu kabul edelim. Orijin merkezli α yarıçaplı D_0 dairesi içinde yakınsayacak çok-kutup açılımı (32)'deki formüldür.D_0 içinde, $q_1,q_2,\ldots,q_l$ yüklerinden dolayı oluşan potansiyeli veren yerel açılım:

$$\Phi(P)=\sum_{i=0}^{\infty}\sum_{k=-j}^{j}L_j^k.Y_j^k(\theta,\phi).r^j \quad (37)$$

$$L_j^k=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{O_n^m.i^{|k-m|-|k|-|m|}.A_n^m.A_j^k.Y_{j+n}^{m-k}(\alpha,\beta)}{(-1)^n A_{j+n}^{m-k}.\rho^{j+n+1}} \quad (38)$$

A_n^m (35)'daki gibi tanımlanmıştır. Herhangi p≥1 için hata şu şekilde sınırlandırılmış olur:

$$\left|\Phi(P)-\sum_{i=0}^{p}\sum_{k=-i}^{i}L_i^k.Y_i^k(\theta,\phi).r^{i+1}\right|\le\left(\frac{\sum_{i=1}^{l}|q_i|}{ca-a}\right)\left(\frac{1}{c}\right)^{p+1} \quad (39)$$

(38) yardımıyla O_k^j çok-kutup açılımı katsayıları L_j^k yerel açılım katsayılarına çeviren lineer operatör tanımlanmış oldu.

5.5.3. Yerel Açılımın Kaydırılması

Merkezi Q=(ρ,α,β) olan yerel açılım

$$\Phi(P) = \sum_{n=0}^{p} \sum_{m=-n}^{n} O_n^m . Y_n^m(\theta', \phi') . r'^n \tag{40}$$

Olsun burada P=(r,θ,φ) ve P-Q=(r',θ',φ') göstermektedir. Böylece

$$\Phi(P) = \sum_{i=0}^{p} \sum_{k=-j}^{j} L_j^k . Y_j^k(\theta, \phi) . r^j \tag{41}$$

$$L_j^k = \sum_{n=j}^{p} \sum_{m=-n}^{n} \frac{O_n^m . i^{|m|-|m-k|-|k|} . A_{n-j}^{m-k} . A_j^k . Y_{n-j}^{m-k}(\alpha, \beta) . \rho^{n-j}}{(-1)^{n+j} . A_n^m} \tag{42}$$

hesaplanabilir ve A_n^m (35)'daki gibi tanımlanmıştır. (42) yardımıyla O_k^j yerel açılım katsayılarını L_j^k yerel açılım katsayılarına çeviren lineer operatör tanımlanmış oldu.

6. GERÇEKLEMENİN DETAYLARI

Bu tez çalışması kapsamında tüm kodlar C/C++ dili kullanılarak yazılmıştır. Barnes-Hut[1] ve Hızlı çok-kutup[2] algoritmaları yayınlandıkları makalelere bağlı kalınarak yerçekimsel kuvvet (gravitational forces) hesabı göz önüne alınarak gerçeklenmiştir. Bu iki algoritma karşılaştırılacağı için kodlama yapılırken aynı sınıf yapısı kullanılmış, farklılıklar fonksiyonlar içerisinde yazılarak genel bir çatı elde edilmiştir. Bu bölümde gerçekleme ile ilgili yapı detaylı olarak anlatılacaktır.

Cisim-cisim, Barnes-Hut ve Hızlı çok-kutup gerçeklemelerinde kuvvet hesaplandıktan sonra hızlara ve konumlara geçmeyi sağlayan integrasyon yöntemi her üç algoritma içinde aynıdır. Yerçekimsel N-cisim probleminin çözümünde kullanılabilecek integrasyon yöntemlerinden euler yöntemi, leapfrog yöntemi ve hermite integrasyon yöntemleri [25] ve [26]'da detaylı olarak anlatılmıştır. Bu tez çalışması için integrasyon yöntemlerinden basit ama etkili bir yöntem olan leap-frog yöntemi kullanılmıştır.

Cisimlerin konumları, hızları ve kütleleri ile ilgili bilgiler dosyada belli bir standart içinde saklanmıştır (Bkz. Bölüm 7). Düzenli dağılmış cisimler ve düzensiz dağılmış cisimler için değerler üretilerek, tüm algoritmalar bu bilgiler kullanılarak çalıştırılmıştır. Düzenli olmayan dağılımlar için plummer modele [25] göre cisim dağılımları üretilmiştir.

Vektörel büyüklük olan cisimlerin konumları, hızları ve kuvvetleri için vektor isimli bir sınıf tanımlanmış ve tüm algoritmalarda bu sınıf yapısı kullanılmıştır. Bu sınıf sayesinde iki boyutlu veya üç boyutlu problemler üzerinde hesaplamalar yapılabilmektedir.

Cisimler için *particle* isimli bir sınıf yapısı tanımlanmıştır. Cisimler üzerindeki tüm işlemler için bu sınıf yapısı ile gerçekleştirilmiştir. Yapı şu şekilde tanımlanmıştır:

```
class particle {
  double data;
  vektor pos;
  vektor vel;
  vektor force;
}
```

6.1. Ağaç Veriyapısı

Barnes-Hut ve Hızlı çok-kutup algoritmaları 3-boyutlu yerçekimsel kuvvet hesabı yapacak şekilde sekizli ağaç veri yapısı ile gerçeklenmiştir fakat boyut sayısının 2 olması durumunda da çalışabilecek bir yapı sunulmuştur.

Ağaç yapısı için *tree* isimli bir sınıf (class) oluşturulmuştur. Bu sınıf yapısı ağacın kök (root) düğüme işaret eden bir işaretçiye sahiptir ve ağaç üzerindeki her işlem bu sınıf yapısından gerçekleştirilmektedir. Ağaçtaki her bir düğüm için *Node* isimli bir yapı kullanılmıştır. *tree* sınıf yapısının üye değişkenlerini şu şekilde tanımlanmıştır :

```
class tree {
  Node *root;
  particle *treeparticle;
  double dt;
  Node *Nodelist;
  int particleLEAF;
}
```

Bu sınıf yapısı içinde root kök düğüme işaret eder. Cisimlerin bir dizi içerisinde saklandığı kabul edildiğinden, ağaç üzerinden cisimlerin bilgilerine ulaşmak için treeparticle isimli işaretçi kullanılmaktadır. Cisimlerin üzerlerine etki

eden kuvvetler hesaplandıktan sonra konumlara geçebilmek için kullanılan zaman aralığı için dt değişkeni kullanılmaktadır. Ağaç yapısı her bir adımda yeniden oluşturulduğundan dolayı her bir adımda dinamik bellek tahsisatı yapmak yerine belli sayısı (genellikle $3/2\,N$) düğüm için bellekten tahsisat yapılarak bu değişkenler kullanılmıştır. Buradaki düğümler yeterli gelmediğinde bu liste genişletilmektedir. Yapraklarda bulunan cisim sayısı particleLEAF değişkeninde saklanmaktadır.

Ağaç üzerindeki her bir düğümün ve yaprakların bilgileri için *Node* isimli bir yapı(struct) şu şekilde tanımlanmıştır:

```
struct Node{
  _type Type;
  vektor geocenter;
  double rsize;
  Node* child[NSUB];
  Node* parent;
};
```

Bu yapı içindeki Type değişkeni, bu yapının *Node* (düğüm) veya *Leaf* (yaprak) olduğunu göstermektedir. Bir düğümün alt-düğümlere *child* işaretçisi ile erişilmektedir ve *NSUB* çocuk sayısını göstermektedir. Bu sayı üç boyutlu problemlerle uğraşıldığında sekiz olmaktadır. Bir düğüme karşılık gelen hücrenin merkezinin koordinatı için *geocenter*, bu hücrenin bir kenar uzunluğu için *rsize* değişkenleri kullanılmıştır. Her bir düğümün ebeveyn düğümüne, *parent* işaretçisi yardımıyla ulaşılabilmektedir.

Algoritmalar her bir adımda ağaç yapısını yeniden oluşturmaktadır. Her bir adım için dinamik bellekten tahsisatı yapmak yerine bir NodeList dizisi oluşturulmuştur. Bu dizi kullanılarak ağaç yapısı oluşturulmaya çalışılmış dizi boyutu yetersiz kaldığında dizi genişletilmiştir. Yazılan mynew fonksiyonu ile bu dizi üzerinden bir düğümün adresi geri döndürülmektedir. Dizi üzerinden doğrudan işlem yapılmasına izin verilmez. Bunun yerine kürsor (cursor) yapısı kullanılmıştır. Yapılan deneylerde genelde düğüm sayısının cisim sayısının 3/2 katından fazla

olduğu gözlemlenmiştir. Fakat bu sayı yapraklardaki cisim sayısı ve cisimlerin dağılımlarına bağlı olarak değişmektedir.

Bir cismin veya düğümün, ebeveyn düğümün hangi numaralı (Bkz. Şekil 3.4) alt-hücresine yerleştirileceğine karar vermek *whichchild* isimli bir fonksiyon yazılmıştır:

```
for ( int k=0; k< NDIM; k++ )
     if (particle[k] >= root->geocenter[ k ] )
      childnumber += 1<<k;
```

Her bir boyutta (k) cisim veya hücrenin merkezi, ebeveyn hücrenin merkezinden büyükse 2^k değeri toplanarak sonuç (childnumber) geri döndürülür. Bu fonksiyon ağaca yerleştirme yapılırken ve sıkıştırılmış ağaç yapısı kullanıldığında cisimleri sıralamak için kullanılmaktadır. Fonksiyon hem iki boyut, hem de üç boyut için problemsiz olarak çalışabilmektedir.

6.2. Barnes-Hut Algoritması

Barnes-Hut algoritmasının en temel adımları *tree* sınıfı içersinde

AğacıOluştur(loadparticles)

AşağıdanYukarıya(upwardpass)

YukarıdanAşağıya(downwardpass)

KuvvetleriHesapla(calculateforces)

Konumlarıgüncelle(updateVelPos)

şeklinde tanımlanmıştır.

Ağaca cisimleri yerleştirebilmek için önce tüm cisimleri içine alabilecek en küçük boyutlu hücrenin merkezi ve bir kenar uzunluğu belirlenir. Daha sonra *AğacıOluştur* fonksiyonu çağrılır. Daha sonra bu kök düğüme cisimler teker teker özyineli (rekürsif) olarak yerleştirilir. Yerleştirme yapılırken hangi alt-hücreye yerleştirme yapılacağı *whichchild* fonksiyonu yardımıyla belirlenir. Bu fonksiyon sonunda dönen numaraya göre düğümün, çocuk düğümüne bakılır. Çocuk düğüm boş veya bir yaprak düğüm olana kadar özyineli olarak fonksiyon tekrar çağrılır. Bu çocuk düğüm boşsa yeni bir yaprak düğüm oluşturulur. Eğer bir yaprak hücreye daha önceden bir cisim yerleştirilmişse (veya *particleLEAF* sayısı kadar cisim yerleştirilmişse) önce *Leaf2Node* fonksiyonu çağrılarak yaprak düğüm bir iç düğüme çevirilir (internal Node) yapısına çevrilir ve cisim uygun alt düğüme yerleştirilir. Yaprak düğüm bir iç düğüme çevrildiğinde bu yaprak düğüm içerisindeki düğümler, tekrar bu iç düğüm içerisine yerleştirilmelidir. Cisimlerin dağılımı düzenli olduğunda bir cismi ağaca yerleştirmek için ortalama $O(\log N)$ seviye inilmesi yeterli olmaktadır. Fakat cisim dağılımları düzensiz olduğunda yükseklik artmaktadır.

AşağıdanYukarıya fonksiyonu içinde yapraklardan başlanarak kök düğüme doğru çıkılarak kütle merkezlerinin koordinatları ve toplam kütleler hesaplanır. Bu işlem özyineli olarak çocuk düğümler önce ebeveyn düğümler sonra (post-order traversal) [27] ziyaret edilecek şekilde gerçeklenmiştir. Bir düğümün kütle merkezinin hesaplanabilmesi için öncelikle bu düğümün tüm çocuk düğümlerinin kütle merkezlerinin hesaplanmış olması gerekir.

Cisimlerin kuvvet hesapları ağaç yapısı üzerinden *KuvvetleriHesapla* fonksiyonu yardımıyla hesaplanmaktadır. Her bir cisim için kök düğümünden başlanarak yaprak düğümlere doğru inilir ve cisim ile düğümlerin yeterince uzakta olup olmadığı kontrol edilir. Bu işlem için *wellseperate* fonksiyonu yazılmıştır. Bu fonksiyon içinde cisim ile düğümün merkezi arasındaki uzaklığın, düğüme karşılık gelen hücrenin bir kenarına oranının teta parametresinden büyük olup olmadığına bakılır. Eğer cisim, düğümden yeterince uzakta ise cisim-küme etkileşimi yapılmak üzere *particlecell* fonksiyonu çağrılır. Cisim yeterince uzakta değilse bu düğümün her bir alt-düğümü özyineli olarak kontrol edilir. Eğer bir yaprak düğüme ulaşılmışsa

bu yaprak düğüm içindeki tüm cisimler ile cisim-cisim etkileşimi yapılmalıdır ve bunun için *directforcecalc* fonksiyonu kullanılır.

Tüm cisimler için kuvvetler hesaplandıktan sonra *KonumlarıGüncelle* fonksiyonu çağrılarak her bir cisim için kuvvetten konumlara geçilir. Bu işlem için *particle* sınıfı içindeki *updatevelpos* fonksiyonu kullanılır.

6.3. Hızlı Çok-Kutup Yöntemi

Hızlı çok-kutup yönteminin gerçeklenmesinde çok-kutup ve yerel açılımlar için kullanılacak tüm matematiksel hesaplamalar DPMTA (Distributed Parallel Multipole Tree Algorithm) [34] kodundan alınmıştır. DPMTA kondun en son sürümlerinde moleküler dinamiğinde kullanılan bazı hızlandırma teknikleri [35] gerçeklenmiştir. Fakat orijinal HÇA [2] makalesine göre gerçeklenme yapılmak istendiğinden kodun 2.0.4 sürümündeki matematiksel hesaplama rutinleri alınmıştır.

Barnes-Hut algoritmasının temel adımları Hızlı çok-kutup içinde aynı şekilde kullanılmıştır. Ağacın oluşturulması Barnes-Hut ile aynı şekilde gerçekleştirilmektedir. (Bkz. Bölüm 6.2)

Aşağıdan yukarıya adımında (*upwardpass*) ağaç üzerinde çok-kutup açılımlarının hesaplamaları yapılmaktadır. Bu hesaplamalar için DPMTA kodundan alınmış *AddMultipoleC* ve Calc_M2M isimli iki fonksiyon kullanılmıştır. Bunlardan *AddMultipoleC* yardımıyla yapraklarda bulunan cisimlerin yaprak düğümün merkezindeki çok-kutup açılımlarının hesaplanması yapılmaktadır. Yani *AddMultipoleC* fonksiyonu bir cismin bağlı bulunduğu yaprak düğümün çok-kutup açılımına ilave edilmesini sağlar. Tüm yapraklar için çok-kutup açılımları hesaplanınca *Calc_M2M* fonksiyonu yardımıyla hesaplanmış çok-kutup açılımları ebeveyn hücrenin merkezine kaydırılır (Bkz. Bölüm 5.5). Bu fonksiyon ebeveyn ve çocuk düğümün çok-kutup açılımlarını parametre olarak alır ve çocuğun merkezini ebeveyn düğümün merkezindeki çok-kutup açılımına kaydırır.

Yukarıdan aşağıya (*downwardpass*) ağaç üzerinde hesaplama işleminde, hızlı çok-kutup açılımlarının yerel açılımlara çevrilmesi işlemi gerçekleştirilmektedir. Bu hesaplamalar için DPMTA kodundan alınmış *Calc_M2L* ve *Calc_L2L* fonksiyonlar kullanılmıştır Çok-kutup açılımının yerel açılıma çevrilebilmesi için öncelikle etkileşim listesindeki düğümlerin belirlenmiş olması gerekmektedir. Etkileşim listesinin belirlenebilmesi için de ebeveyn hücrelerin yakın komşularının bilinmesi gerekmektedir. Bu amaçla *Node* yapısı içinde bir düğümün yakın komşularının listesi *nearneighbour* işaretçisi yardımıyla saklanmıştır. Kök düğümün çok-kutup ve yerel açılımı olmaz. Yerel açılımlar ancak 2.seviyeden itibaren geçerli olmaktadır. Öncelikle 1.seviyedeki düğümler için yakın komşular listesi oluşturulmaktadır. Her bir düğüm kendisinin yakın komşusu olarak kabul edilip *nearneighbour*'a eklenmiştir. Sonra 2.seviyeden itibaren hesaplamalar şu şekilde yapılmaktadır:

Bir düğüm, ebeveyn düğümünün (*parent*) yakın komşuları listesindeki (*nearneighbour*) düğümlerin tüm çocuk düğümleri ile etkileştirilmeye çalışılır. Burada iki durum vardır.

Düğümler birbirinden yeterince uzakta ise uzaktaki düğümün çok-kutup açılımı *Calc_M2L* fonksiyonu yardımıyla yerel açılıma çevrilir.

Düğümler birbirine yakınsa bu durumda düğüm yakın komşular listesine eklenir.

Düğümlerin yeterince uzakta olup olmadığını anlamak için *welllseperate* fonksiyonu kullanılmaktadır. Orijinal HÇA [2] yönteminde aynı seviyede bulunan düğümlere göre işlem yapılmaktadır. Yakın komşular belirlenirken bir hücrenin yanındaki hücrelerle beraber iki yanındaki (yanındakinin yanındaki) hücrelerde yakın komşular olarak seçilmektedir. Fakat bunun yerine sadece yanındaki hücreler yakın komşular olarak belirlenerek işlem yapılarak [21] elde edilen düşük doğruluk oranını, seri açılımdan daha fazla terim alarak giderilmesi tavsiye edilmiştir. Böylece bir hız artışı sağlanmıştır. Ayrıca bir düğümün tüm çocuk düğümleri yeterince uzakta ise bu durumda bu çocuk hücrelerin ebeveyni ile etkileşim yapmakta hız artışı sağlamıştır [21] fakat hata miktarı artacaktır.

Etileşim listesindeki düğümlerin çok-kutup açılımlarının yerel açılıma çevrilmesinden sonra bu düğümün tüm çocuk düğümlerinin merkezlerine hesaplanmış olan bu yerel açılım *Calc_L2L* fonksiyonu ile kaydırılmaktadır (Bkz. Bölüm 5.5). Daha sonra bu çocuk düğümlerin her birisi için etkileşim listesi oluşturularak yerel açılımlar hesaplanır ve bunlarında çocuk düğümlere kaydırılır.

Tüm yerel açılımlar hesaplandığında ise yaprak düğümlerdeki cisimlerin konumlarına bu yerel açılımlar kaydırılarak kuvvet hesaplanır. Bu işlem için DPMTA kodundan alınan *Force_C* fonksiyonu kullanılmaktadır. Yerel açılım ile hesaplanması mümkün olmayan yakın komşulardaki cisimler için doğrudan hesaplama yapılması gerekmektedir. Yakın komşuların listesi kullanılarak yapraklardaki her bir cisim için doğrudan kuvvet hesabı yapılarak, bu kuvvetlerden konumlara geçilir.

6.4. Sıkıştırılmış Barnes-Hut ve FMM

Bölüm 3.2 'de anlatıldığı şekilde önce cisimler sıralanarak daha sonra ağaç veri yapısı oluşturulmaktadır. İlk adımda quicksort algoritması kullanılabilir fakat daha sonraki adımlarda cisimlerin konumlarında çok büyük bir değişiklik olmayacağından insertionsort algoritması kullanılırsa daha iyi çalışma zamanı elde edilecektir. Sıralama işlemi için *whichchild* fonksiyonu kullanılmaktadır. İki hücrenin küçük hücresi (smallcell) hesaplanır. Bu iki hücre hesaplanmış küçük hücrenin hangi numaralı alt-hücresi olduğuna bakılarak sıralama işlemi yapılmaktadır.

Bu veri yapısı gerçeklendiğinde daha önceki algoritmalardaki ağaç ve düğüm yapısı aynen alınarak bazı değişkenler ilave edilmiştir. Düğüm yapısında büyük hücre için *largecell*, küçük hücre için *smallcell* değişkenleri tanımlanmıştır. Ağaca yerleştirme işlemi algoritma 3.2'deki gibi gerçeklenmiştir (Bkz. Bölüm 3.2).

Sıkıştırılmış ağaç veri yapısında ağaç oluşturulması dışında Barnes-Hut ve HÇA üzerinde fazla bir değişliğe gerek kalmamaktadır. Fakat tüm hesaplamalarda hücrenin merkezi yerine bu hücrenin küçük hücresinin (smallcell) merkezi kullanılmaktadır.

7. DENEYSEL SONUÇLAR

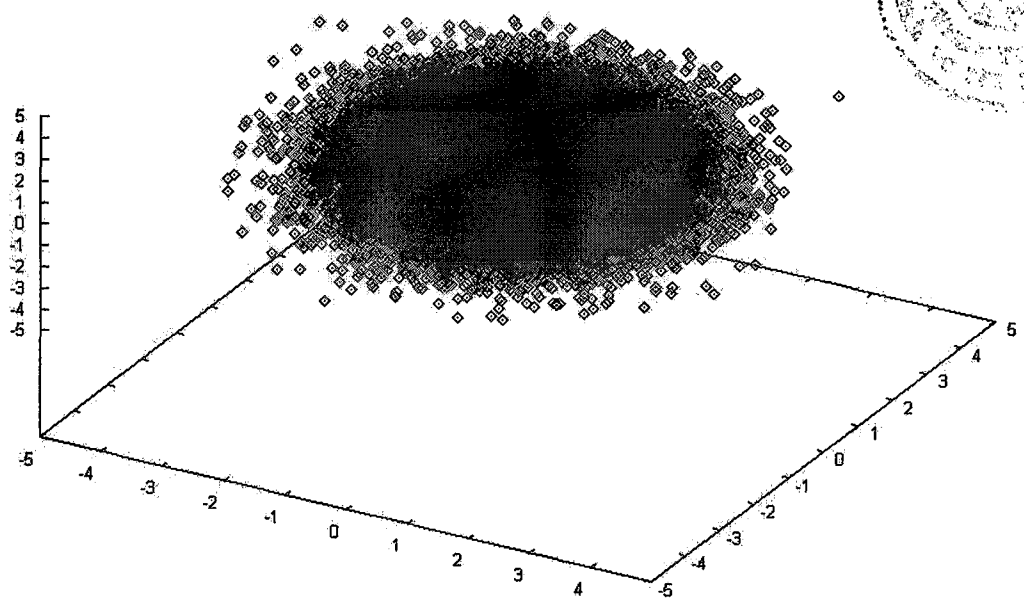
Bu tez kapsamda yapılan gerçeklemeler için veri dosyası, veri dosyasından okuma ve veri dosyasına yazma için standart bir format oluşturulmuş ve bu rutinler Barnes-Hut ve Hızlı çok-kutup gerçeklemelerinde aynen kullanılmıştır.

Öncelikle veri dosyası standardından bahsedelim. Dosyanın ilk satırında cisim sayısını gösteren N değeri, ikinci satırında simülasyona başlama zamanı, üçüncü satırında simülasyonun sona ereceği zaman ve dördüncü satırında simülasyonun adım süresi verileri bulunmaktadır. Beşinci satırdan itibaren her bir satırda bir cisme ait tüm bilgiler bulunmaktadır. Her bir cisim için satırda bulunan veriler sırasıyla kütle, konum ve hızdır. Kütle elektrostatik problem düşünüldüğünde yük olacaktır. Konumlar ve hızlar vektörel büyüklükler olduğundan bu veriler her üç boyut içinde yazılmıştır. Yani kütleden sonra x konumu, y konumu, z konumu, x düzlemindeki hızı, y düzlemindeki hızı, z düzlemindeki hızı şeklinde veriler reel sayı olarak kaydedilmiştir. Veri dosyasından cisimlerle ilgili bilginin bulunduğu satırdan bir örnek aşağıda verilmiştir:

1 0.28616333 0.639923 0.0158386 0.247 0.247 0.247

Rastgele sayı üreticisi kullanılarak cisimlerin konumları 0 ile 1 arasında rastgele üretilmiş ve düzenli dağılmış cisimler elde edilmiştir. Düzensiz dağılmış cisimler plummer modeline göre [25] üretilmiştir. Elde edilen cisimlerin dağılımının görselleştirilmesinde gnuplot isimli uygulamadan yararlanılmıştır. Gnuplot yardımıyla çizdirilmiş bir ekran görüntüsü Şekil 7.1'de verilmiştir.

Deneyler üzerinde Windows XP işletim sistemi yüklü, Pentium 4 2.8 işlemcili ve 1 GB belleği bulunan bir bilgisayarda yapılmıştır. Kodlar Visual C++ ile yazılmış ve mingw (gnuc derleyicisinin windows'a tanışınmış hali) derleyicisi ile derlenip çalıştırılmıştır. Kodlar Linux işletim sistemi üzerinde de çalıştırılabilecek haldedir. Çalışma Zamanlarının tamamı saniye olarak verilmiştir. Aksi belirtilmediği sürecek yapraklardaki cisimler 1 olarak kabul edilmiştir.



Şekil 7.1 Düzensiz Dağılmış Cisimlerin Gnuplot ile üç boyutlu görünümü

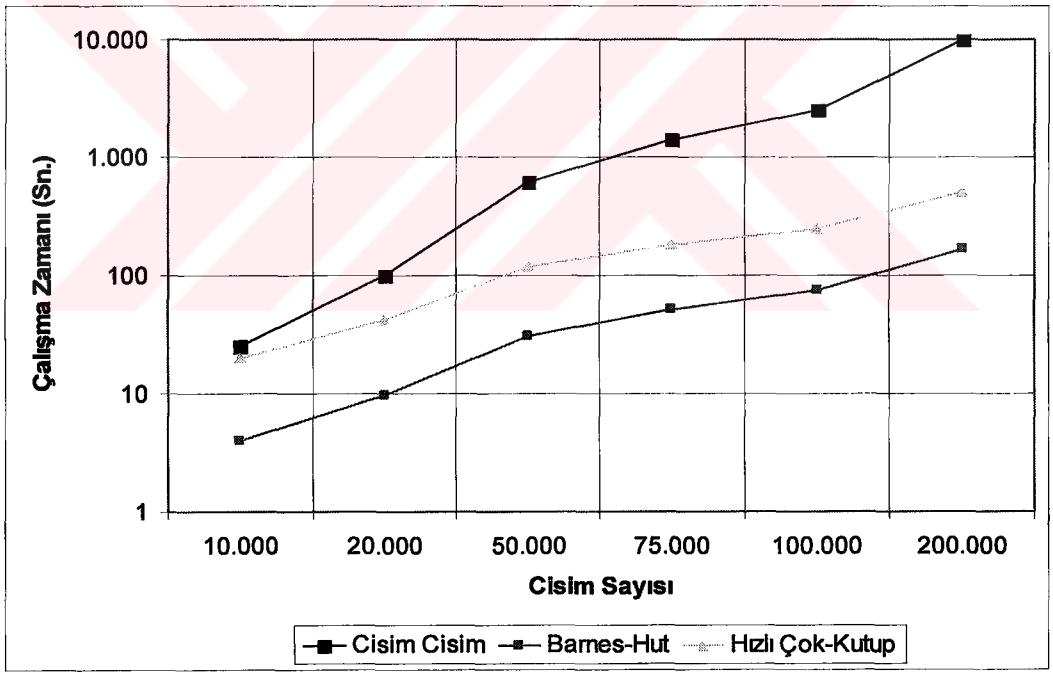
Çeşitli sayılarda ve dağılımlarda cisimler alınarak, yazılan cisim-cisim algoritması çalıştırılmış, bir adım sonundaki kuvvetler hesaplanarak kaydedilmiştir. Bu tez çalışması kapsamında yazılan tüm kodlar için leap-frog integrasyon yöntemi kullanılmıştır. Hesaplanmış olan bu kuvvetlerden konumlara geçilirken kullanılan integrasyon yöntemine bağlı olarak küçükte olsa hata olmaktadır. Bu hata hakkında fikir vermesi açısından enerjinin korunumu prensibinden yararlanılabilir. Fakat bu tez çalışmasında kuvvet hesaplama işlemini kısaltmayı amaçlayan iki algoritma karşılaştırılacağı için bir adım sonraki kuvvetler karşılaştırılmıştır. Cisim-cisim algoritması ile bir adım sonunda elde edilen kuvvetler doğru kabul edilip, Barnes-Hut ve HÇA algoritmalarından elde edilen kuvvetler ile farkı alınarak ortalama hata hesaplanmıştır. N tane cisim için ortalama hata hesaplanırken :

$$\text{Ort\_Hata} = \frac{1}{N}\left[\sum_{i=1}^{N}\left(\frac{f_{hiyerarşik} - f_{cisim-cisim}}{f_{cisim-cisim}}\right)^2\right]^{1/2}$$

formülü kullanılmıştır[33]. Burada $f_{hiyerarşik}$, Barnes-Hut veya HÇA ile hesaplanan kuvvetleri, $f_{cisim-cisim}$ cisim-cisim ile hesaplanan kuvvetleri göstermektedir.

Deneyler için 10.000, 20.000, 50.000, 75.000, 100.000 ve 200.000 ve 1.000.000 tane düzenli dağılmış cisim için oluşturulmuş dosya kullanılmıştır. Öncelikle cisim-cisim ve hiyerarşik ağaç yöntemleri karşılaştırılmıştır. $O(N^2)$'lik çalışma zamanına sahip cisim-cisim gerçeklemesi tüm cisim dosyaları ile çalıştırılmıştır. Cisim sayısı on kat arttığında çalışma zamanının yüz kat arttığı gözlemlenmiştir.

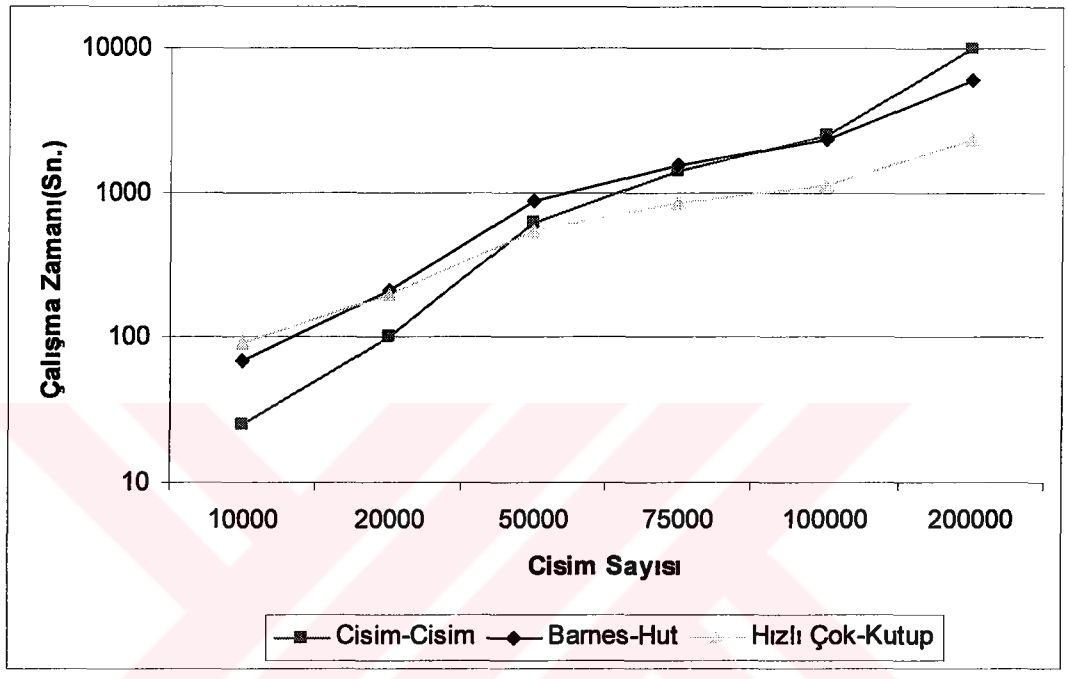
Şekil 7.2'de Barnes-Hut için teta değeri 0.5, Hızlı çok-kutup algoritması ise p değeri 3 alınarak düşük doğruluk oranı için (ortalama hata yaklaşık 10^{-3}) her üç algoritmanın çalışma zamanları karşılaştırılmıştır.



Şekil 7.2 Cisim-cisim, BHA ve HÇA düşük doğruluk oranında cisim sayısı ve çalışma zamanı değişimi

Şekil 7.3'de ise Barnes-Hut için teta değeri 0.1, Hızlı çok-kutup algoritması ise p değeri 7 alınarak yüksek doğruluk oranı için (ortalama hata yaklaşık 10^{-5}) her üç algoritmanın çalışma zamanları karşılaştırılmıştır. Şekil 7.2 ve Şekil 7.3'de cisim-

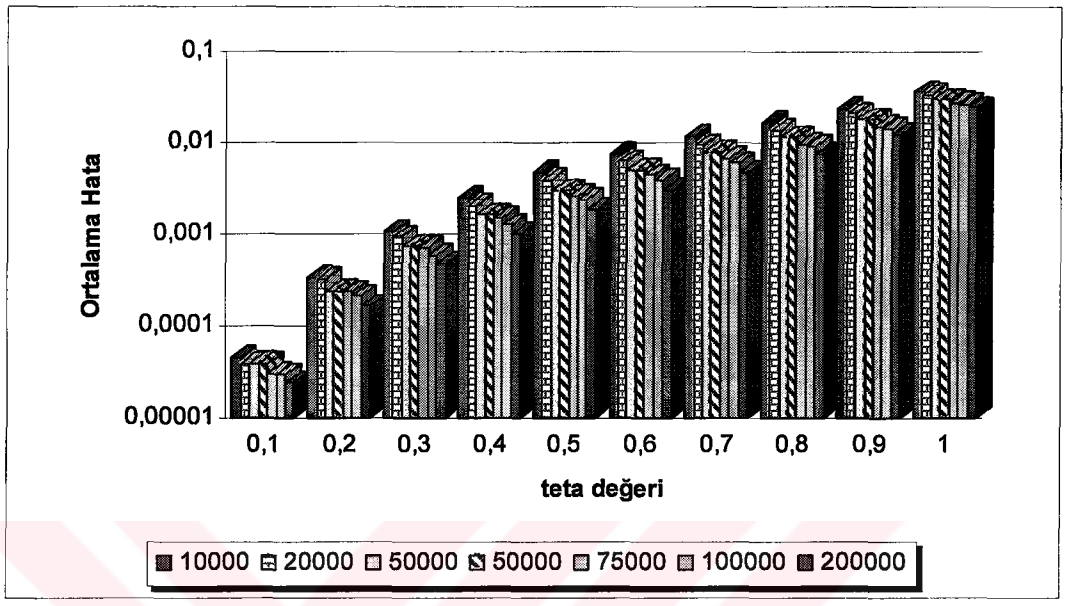
cisim, Barnes-Hut ve HÇA gerçeklemelerinin çalıştırıldığında elde edilen çalışma zamanları logaritmik artışla verilmiştir. Bu iki şekilden de görüldüğü gibi düşük doğruluk oranlarında Barnes-Hut algoritması oldukça iyi sonuçlar vermiştir. HÇA ise ancak yüksek doğruluk oranında ve yüksek sayıda cisim için simülasyon yapıldığında etkinliğini göstermektedir.



Şekil 7.3 Cisim-cisim, BHA ve HÇA yüksek doğruluk oranında cisim sayısı ve çalışma zamanı değişimi

Barnes-Hut algoritması cisim sayılarının ve teta değişiminin ortalama hataya etkisi Şekil 7.4'de incelenmiştir. Bu grafikte ortalama hata ekseni logaritmik olarak verilmiştir. Teta değerinin 0.1'e yaklaştıkça hata oranındaki düşüş şekilde görülmektedir. Fakat teta'nın 0.1 olarak seçildiği durumda (10^5 tane cisimden fazla sayıda cisimler için) Barnes-Hut algoritması, cisim-cisim algoritmasından daha uzun sürede sonucu vermektedir. Bu sonucun alınmasında iki önemli etken vardır. Birincisi cisim-grup etkileşiminin sayısının çok düşük olması ve ağaç üzerinde kök düğümden başlanarak yapraklara kadar gezinilmesidir. İkinci etken ise cisim-cisim algoritmasında bir cismin başka bir cisme olan etkisi hesaplandıktan sonra, zıt yönlü kuvvet alınarak hesaplama yapılmasına gerek kalmamıştır. Barnes-Hut algoritması

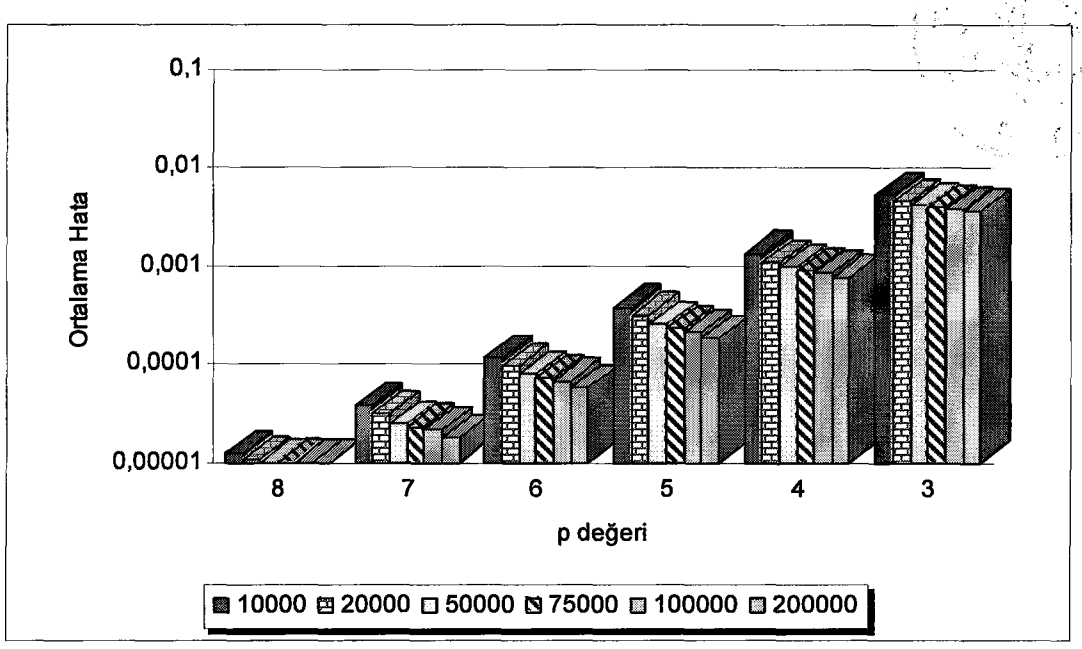
ise her bir cisim için etkileri tek tek hesaplamaktadır. Bu nedenlerden dolayı cisim sayısı az olduğu ve yüksek doğruluk oranı istendiğinde Barnes-Hut algoritması uygun değildir.



Şekil 7.4 Barnes-Hut Algoritmasının düzenli dağılmış cisimler için teta değerleri ile ortalama hatanın cisim sayısı ile değişim grafiği

Hızlı çok-kutup algoritması için cisim sayısının ve p değerinin değişmesinin ortalama hataya etkisi Şekil 7.5'de incelenmiştir. Hata oranı düştüğünde Hızlı çok-kutup algoritması daha kısa zamanda çalışmaktadır. Bu grafikte görüldüğü gibi p değerinin hata üzerine etkisi, Barnes-Hut algoritmasında tetanın değişiminin hata etkisinden oldukça fazladır.

Yapraklarda bulunan cisim sayısının artırılması durumda algoritmaların çalışma zamanları ve doğruluk oranları üzerindeki değişim incelenmiştir. Yapraklarda bulunan cisim sayısı artırıldığında ağacın yüksekliğinin azalmaya başladığı böylece ağaç üzerinde gezinme işlemlerinin azaldığı gözlemlenmiştir. Ayrıca cisim-cisim etkileşimlerinin sayısı ve doğruluk oranları artmıştır.

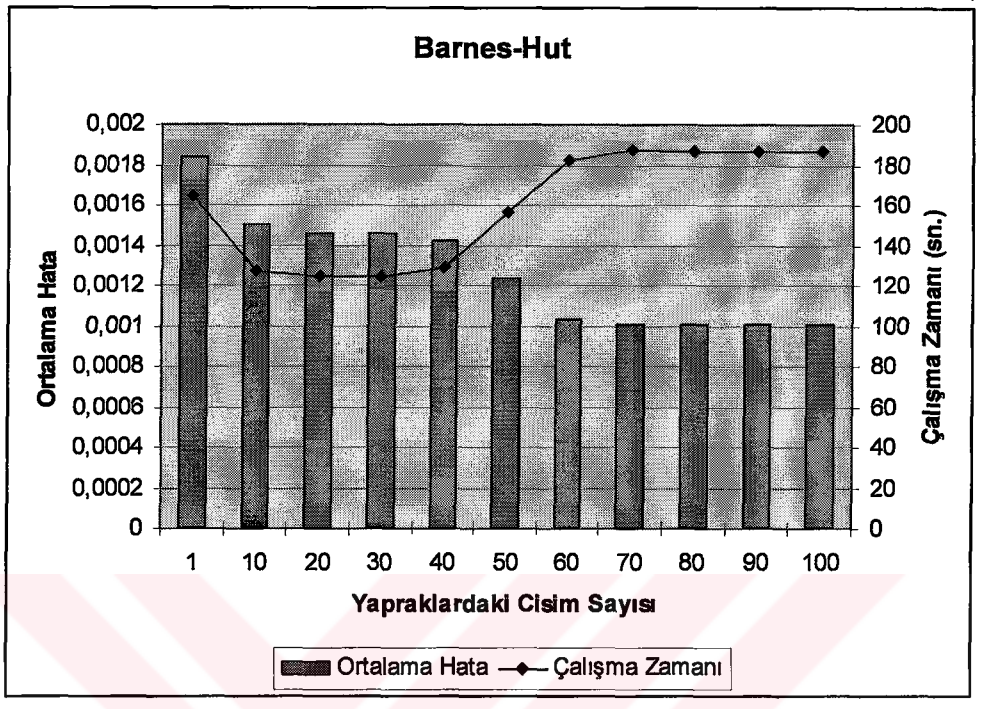


Şekil 7.5 Hızlı çok-kutup algoritmasında düzenli dağılmış cisimler için p değeri ile Ortalama Hata değişimi

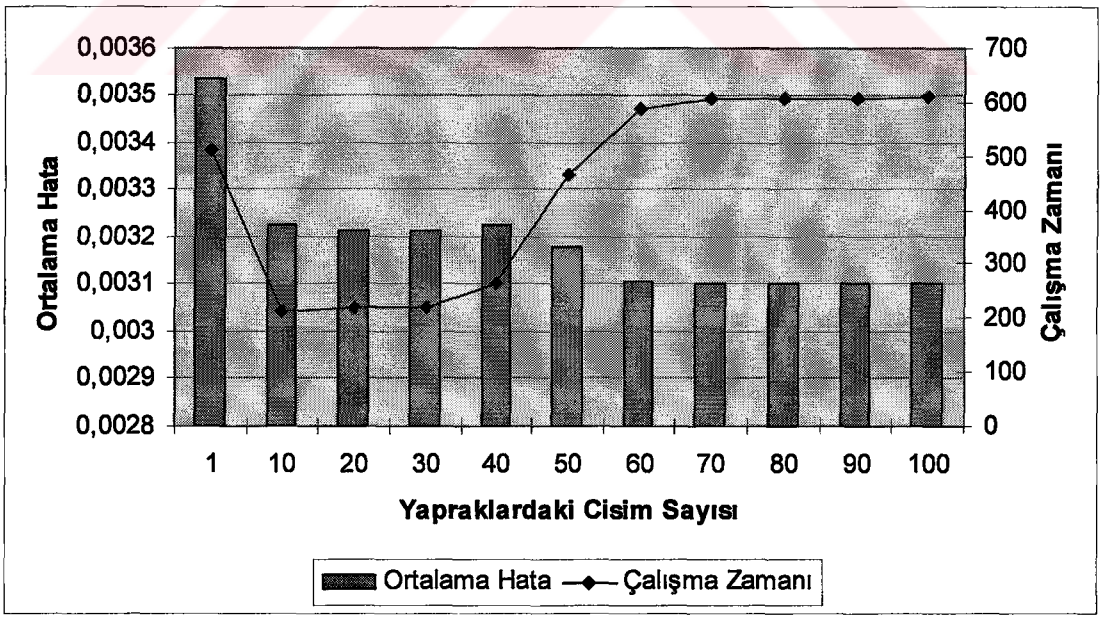
Yapraklardaki cisim sayısının değişimini gözlemleyebilmek için düzenli dağılmış 200.000 tane cisim için Barnes-Hut algoritması 0.5 Teta değeri ile çalıştırılmıştır. Elde edilen grafik Şekil 7.6'da verilmiştir. Şekil incelendiğinde yapraklardaki cisim sayısının artması 40 değerine kadar çalışma zamanını azaltmakta ve doğruluk oranını arttırmaktadır. Fakat 40 değerinden sonra çalışma zamanı artmaya başlamış, ortalama hata düşmeye devam etmiştir. Yapraklardaki cisim sayısının 60'dan büyük olması durumda ise çalışma zamanı ve ortalama hatada büyük değişiklikler gözlemlenmemiştir.

Hızlı çok-kutup algoritmasında ise yapraklardaki cisim sayısının algoritmanın doğruluk oranına etkisinin çok daha büyük olduğu gözlemlenmiştir. Bu algoritma da 200.000 tane cisim, p değeri 3 olarak seçilerek çalıştırılmış ve elde edilen grafik Şekil 7.7'de verilmiştir. Bu şekil incelendiğinde HÇA için yapraklardaki cisim sayısının bir olmasının bu algoritma için uygun olmadığı gözlemlenmiştir. Çünkü yapraklardaki cisim sayısının arttırılması ile ortalama hata ve çalışma zamanı Barnes-Hut algoritmasına göre oldukça fazla düşmüştür. Bu algoritma için

yapraklardaki cisim sayısının 10-40 arasında alınması durumunda hem düşük hata hem de düşük çalışma zamanı elde edilmiştir



Şekil 7.6 Düzenli dağılmış 200.000 cisim için BHA'da yapraklardaki cisim sayısının ortalama hata ve çalışma zamanı üzerine etkisi (Teta=0.5)



Şekil 7.7 Düzenli dağılmış 200.000 tane cisim için HÇA'da yapraklardaki cisim sayısının ortalama hata ve çalışma zamanı üzerine etkisi (p=3)

Düzenli dağılımlar için etkin çözüm üreten algoritmaların düzensiz dağılımlar için nasıl bir performans sergilediği incelenmiştir. Sıkıştırılmış ağaç veri yapısı kullanılarak her iki algoritma gerçeklenmiştir. Fakat sıkıştırılmış HÇA için beklenenden çok daha fazla hata elde edildiğinden bu algoritma için sadece çalışma zamanı verilmiştir.Öncelikle sıkıştırılmış ağaç veri yapısı ile gerçeklenen Barnes-Hut algoritmasının düzensiz dağılımlar için çalışma zamanı ve ortalama hata değişimi incelenecek daha sonra sıkıştırılmış ağaç ile sekizli ağaç yapıları karşılaştırılacaktır.

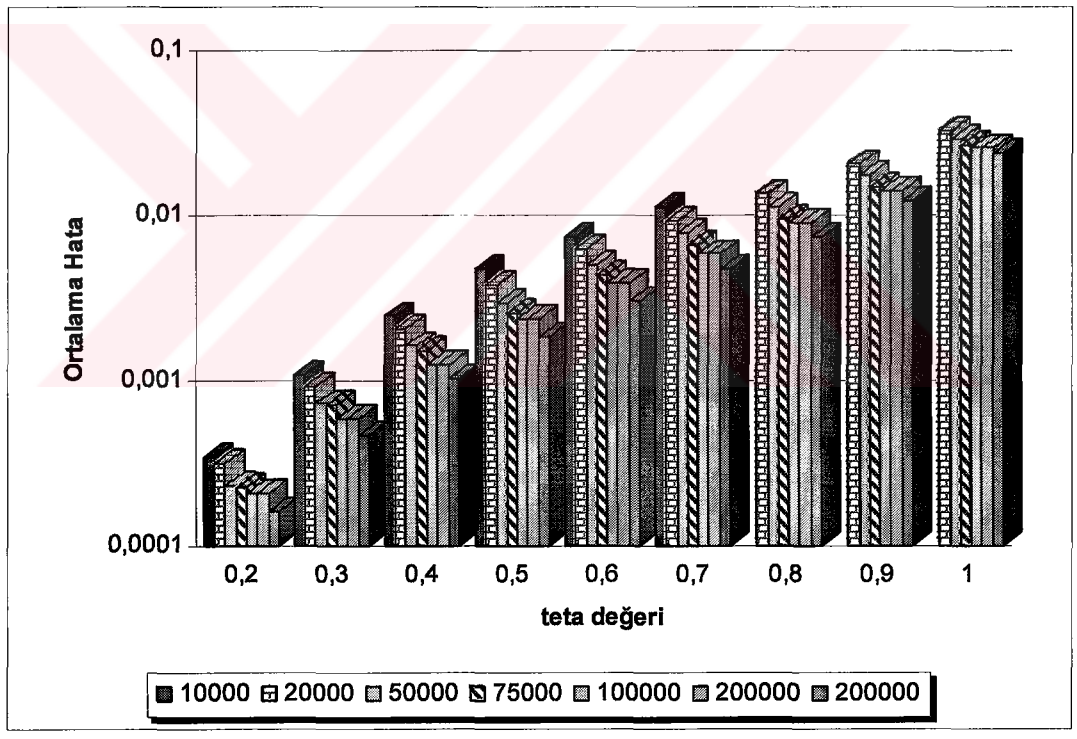
Şekil 7.8'de düzensiz dağılmış 100.000 tane cisim için sıkıştırılmış Barnes-Hut gerçeklemesinin çalıştırıldığında teta değişimi ile ortalama hatanın değişiminin grafiği verilmiştir. Sıkıştırılmış ağaç veri yapısı kullanılarak gerçeklenen Barnes-Hut algoritmasında çok-ufakta olsa bir hata düşüşü gözlemlenmiştir fakat asıl sağladığı kazanım çalışma zamanlarında gözlemlenmiştir.

Düzenli dağılmış cisimler ile düzensiz dağılmış cisimlerin algoritmaların hata oranları ve çalışma zamanlarının karşılaştırılması için 100.000 tane cisim kullanılarak gerçeklemeler çalıştırılmıştır. Gerçeklemeler yapraklardaki cisim sayısı bir seçilerek çalıştırılmıştır. Elde edilen ortalama hata grafiği Şekil 7.9'da, çalışma zamanı grafiği Şekil 7.10'da verilmiştir.

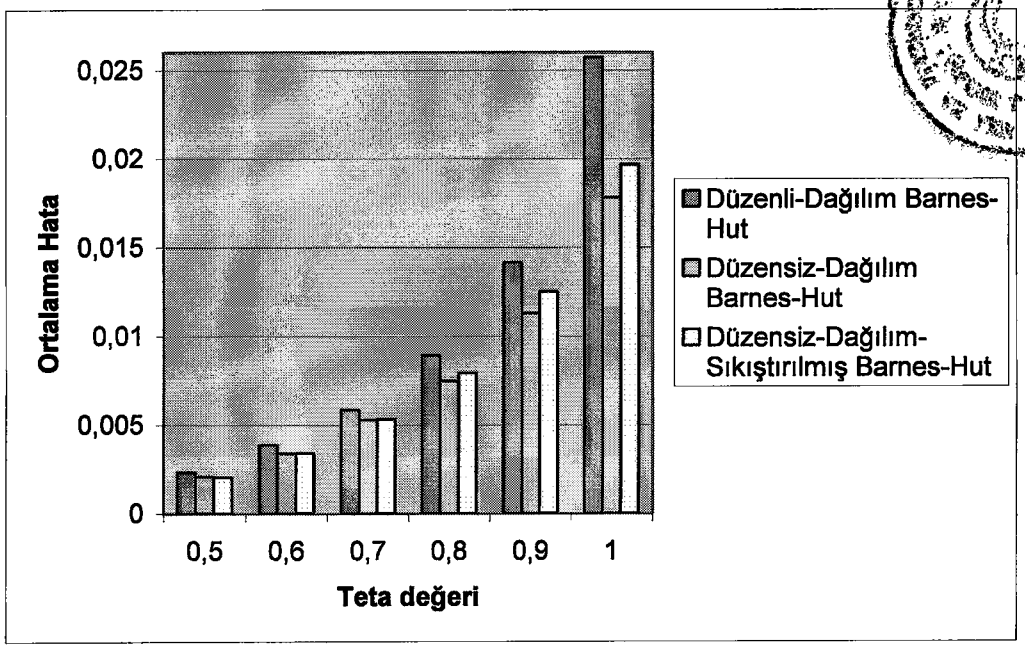
Şekil 7.9'da hata oranları incelendiğinde cisim dağılımlarının ortalama hataya etkisi görülmektedir. Düzensiz dağılmış cisimler için algoritma daha iyi bir doğruluk oranı vermiştir. Ağaç yapısının dengesiz olması ve cisim-cisim etkileşiminin sayısının artması bu hata oranların nedenlerinden olabilir. Sıkıştırılmış ağaç veri yapısının kullanımında elde edilen ortalama hata farkı tetanın 0.7'den küçük değerleri için ortadan kalkmıştır. Genellikle bu algoritma için teta değeri 0.7'den küçük değerler alındığından bu değerden büyük değerler için elde edilen fark ihmal edilebilir kaldı ki çalışma zamanları incelendiğinde sıkıştırılmış ağaç veri yapısı için tetanın 0.7'den küçük alınmasının zorunlu olduğu görülmektedir.

Şekil 7.10'da ise sıkıştırılmış ve sekizli ağaç veriyapısının farkının çalışma zamanı olarak grafiği görülmektedir. Düzenli dağılmış Barnes-Hut algoritması sekizli ağaç yapısı ile oldukça etkin bir çözüm vermektedir. Fakat sekizli ağaç yapısı

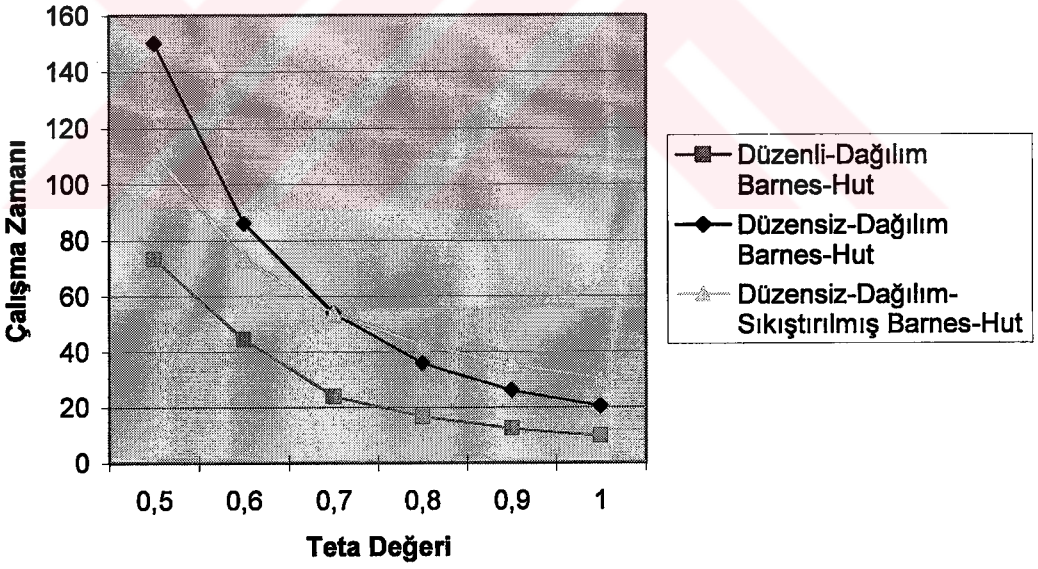
kullanılarak düzensiz cisim dağılımları için problem çözüldüğünde bu etkinlik yitirilmektedir. Sıkıştırılmış ağaç veri yapısında ağaç oluşturulurken sıralama işlemi gerçekleştirilmektedir. Bu nedenle cisimlerin sıralanması toplam çalışma zamanını etkilemektedir. Sıkıştırılmış ağaç veri yapısının düzensiz dağılımlar için çalışma zamanı , sekizli ağaç yapısının düzenli dağılımlar için çalışma zamanının belli bir değer ötelenmiş hali gibi gözlemlenmiştir. Bu ötelemenin nedeni cisimlerin sıralanması için gerekli zamandır. Bu zaman quick-sort algoritması kullanıldığında elde edilmiştir fakat simülasyonlar bir adım değil pek çok adımlar için çalıştırılmaktadır. Ve bu adımlarda cisimlerin ağaç üzeirndeki konumlarında çok büyük değişiklikler olmamaktadır. Birinci adımdan sonraki adımlar için insertion-sort algoritması kullanılarak sıralama işlemi için harcanan zaman minimuma indirilebilmektedir.



Şekil 7.8 Sıkıştırılmış Barnes-Hut algoritmasının cisim sayısı, teta ve ortalama hata değişim grafiği



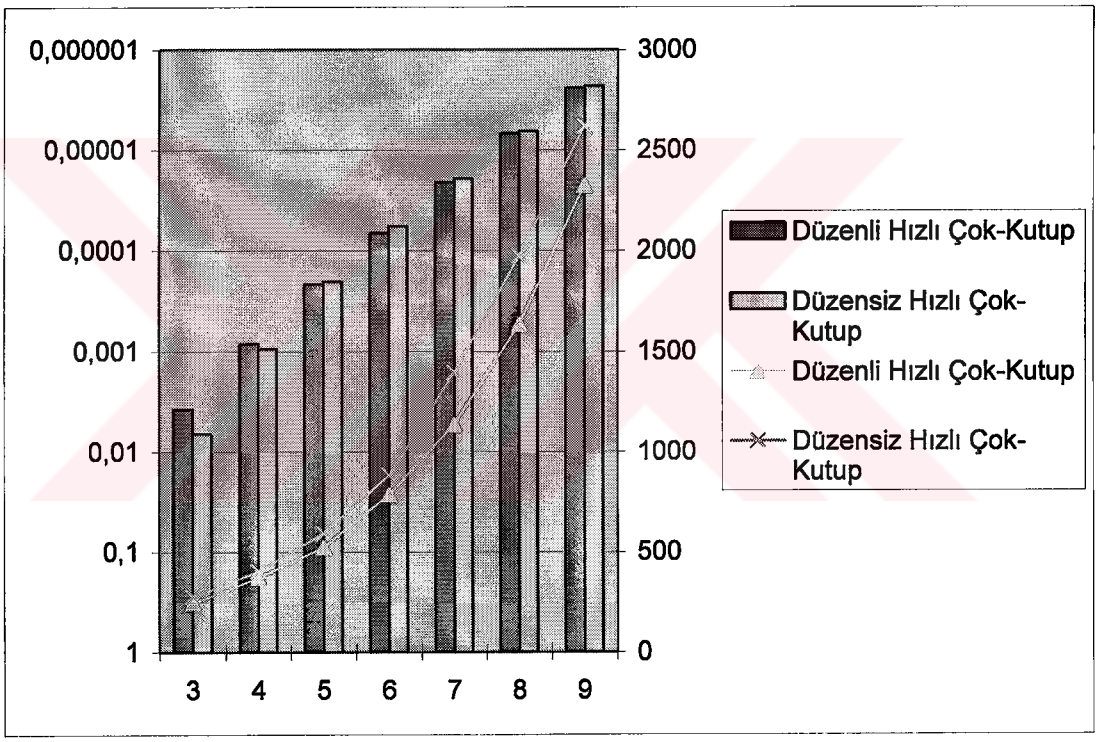
Şekil 7.9 Barnes-Hut ve Sıkıştırılmış Barnes-Hut algoritmaları için Dağılımların ve Tetanın Ortalama Hata üzerine etkisi (yapraklardaki cisim sayısı 1)



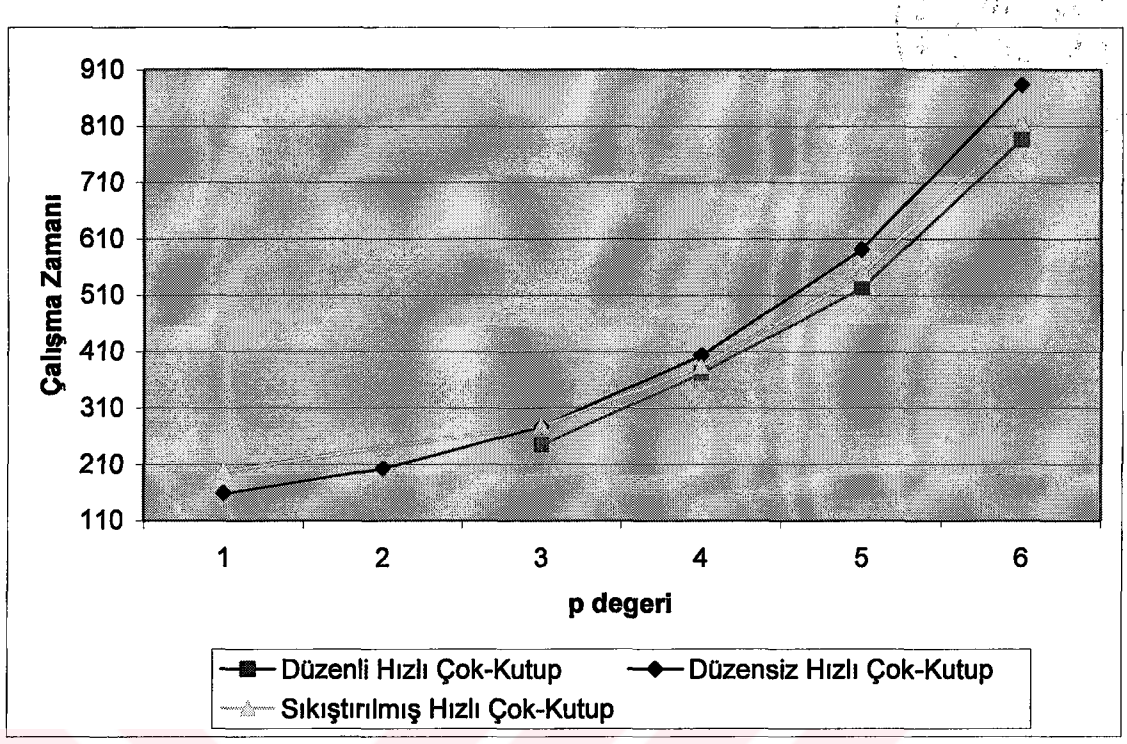
Şekil 7.10 Barnes-Hut ve Sıkıştırılmış Barnes-Hut algoritmaları için Dağılımların ve Tetanın Çalışma Zamanı üzerine etkisi (yapraklardaki cisim sayısı 1)

Hızlı çok-kutup algoritması, 100.000 tane düzenli ve düzensiz dağılmış cisim için çalıştırılarak elde edilen grafik Şekil 7.11'de verilmiştir. Şekil 7.11'de ortalama hata bar grafikleri ile çalışma zamanları ise çizgi ile belirtilmiştir. Düzensiz cisim dağılımlarında HÇA çalışma zamanının arttığı gözlemlenmiştir.

Sıkıştırılmış ağaç yapısının HÇA algoritmasına olan etkisi ancak çalışma zamanı olarak verilmiştir. Daha öncede belirtildiği gibi bu algoritma için sıkıştırılmış ağaç veri yapısı ile elde edilen ortalama hata oranı beklenenin üstünde çıkmıştır. Sıkıştırılmış ağaç veri yapısı ile gerçeklenen Hızlı çok-kutup algoritması 100.000 tane cisim için çalıştırılmış ve çalışma zamanı grafiği Şekil 7.12 'de verilmiştir.



Şekil 7.11 Düzenli ve düzensiz dağılmış cisimler için HÇA'nın hata ve çalışma zamanı



Şekil 7.12 Sıkıştırılmış HÇA'nın plummer modele göre dağılmış 100.000 tane cisim için p değeri ve çalışma zamanı

8. SONUÇ

Yapılan deneyler sonunda düşük doğruluk oranında (ortalama 10^{-3}) Barnes-hut algoritmasının hızlı çok-kutup algoritmasından daha hızlı çalıştığı görülmüştür. Düzensiz cisim dağılımlarında sekizli ağaç yapısı ile gerçeklenen her iki algoritmada da çalışma zamanlarında artış gözlemlenmiştir.artmaktadır.

Yüksek doğruluk oranı istendiğinde sekizli ağaç yapısı ile gerçeklenmiş Hızlı çok-kutup algoritması ($N > 2.10^4$ olmak şartıyla) Barnes-Hut algoritmasından düzenli dağılmış cisimler için daha hızlı çalışmaktadır. Barnes-Hut algoritması fazla sayıda cisim (10^{-5}'den büyük değerler) için çok yüksek doğruluk oranı (ortalama 10^{-5}) için çalıştırıldığında cisim-cisim algoritmasından bile daha yavaş çalışmaktadır.

Sıkıştırılmış sekizli ağaç veri yapısı kullanılarak Barnes-Hut algoritması düzensiz dağılımlar için çalıştırıldığında standart sekizli ağaç yapısına göre daha hızlı çalıştığı görülmüştür. Fakat sıkıştırılmış Barnes-Hut algoritmasında cisimler ağaca yerleştirilmeden önce sıralanmaktadır. Sıralama işlemi nedeniyle teta'nın 0.7'den büyük değerlerinde sıkıştırılmış Barnes-hut algoritması sekizli ağaç ile gerçeklenmiş BHA'dan daha yavaş çalışmaktadır. Teta'nın 0.7'den düşük değerleri için sıkıştırılmış Barnes-hut algoritması hem daha hızlı çalışmakta hem de daha kesin sonuçlar vermektedir. İkinci adımdan itibaren kullanılan sıralama algoritması insertion-sort olarak değiştirildiğinde cisimlerin sıralanmasından dolayı doğan bu fark çok daha azalacaktır.

Yapraklardaki cisim sayısının artmasıyla algoritmaların doğruluk oranlarında bir iyileşme görülmüştür. Yapraklardaki cisim sayısının büyük olması ağacın yüksekliğinin düşük olmasına ve cisim-cisim etkileşiminin sayısının artmasına neden olmaktadır. Yapılan deneylerde genellik yapraklardaki cisim sayısının 20-40 arasında alınması hem çalışma zamanı hem de doğruluk oranı olarak optimum sonuçlar vermiştir.

KAYNAKLAR

1. **J. BARNES AND P. HUT,** "A Hierarchical O(N Log N) Force-Calculation Algorithm", Nature 324, 446-449 (1986).

2. **L. GREENGARD AND V. ROKHLIN**, "A Fast Algorithm For Particle Simulations", Journal Of Computational Physics Vol. 73, Number 2, December (1987), pages 325–348.

3. **V. ROKHLIN**, "Rapid Solution of Integral Equations of Classical Potential Theory", J. Comp. Phys. v. 60

4. **J. Barnes**, "A Modified Tree Code: Don't Laugh; It Runs", J. Comput. Phys., 87 (1990) 161-170

5. **SRINIVAS ALURU, JOHN GUSTAFSON, G.M. PRABHU,** "Truly Distribution-Independent Algorithms for the N-body Problem"

6. **W. APPEL**. "An Efficient Program for Many-body Simulation", Siam. J. Sci. Stat. Comput. 6, 85–103 (1985).

7. **C.R.ANDERSON, C. Greengard, L. Greengard, and V. Rokhlin**. "On The Accurate Calculation Of Vertex Shadding". Physics of Fluids (Fluid Dynamics), 2(6), 1990.

8. **L. GREENGARD AND . STRAIN**. "A Fast Algorithm For The Evaluation Of Heat Potentials". Technical Report YALEU/DCS/RR-700, Yale University, May 1989.

9. **S. Bhatt, M. Chen, C.Y. Len and P. Liu**, "Abstractions For Parallel N-Body Simulations", Tech. Rep. DCS/TR-895, Yale University, 1992

10. **L. GREENGARD**. "Potential Flow In Channels". SIAM Journal on Scientific and Statistical Computing, 11(4):603 620, 1990.

11. **P.HANRAHAN, D. SALTZMAN AND L.AUPPERLE.** "A Rapid Hierarchical Radiosity Algorithm", Computer Graphics, 25(4):197-206,July 1991

12. **K. NABORS AND J. WHITE.** "Fastcap: A Multipole Accelerated 3-D Capacitance Extraction Program", IEEE; Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10(11):1447-59, 1991.

13. **J. AMBROSIANO, L. GREENGARD, AND V.ROKHLIN.** Gridless plasma simulations via the fast multipole method. In ICS 88, Third International Conference on Supercomputing. Proceedings, Supercomputing '88, volume 1.Int. Supercomputing Inst., 1988

14. **FATIH ERDOĞAN SEVİLGEN,** Distribution Independent Spatial Data Structures and Algorithms with Applications, Ph.D thesis, Syracuse University, 2000

15. **S. ALURU AND F. SEVILGEN,** "Dynamic Compressed Hyperoctrees With Applications To N-Body Problem", Proc. Foundations of Software Technology and Theoretical Computer Science (1999) 21-33.

16. **H. SAMET,** "The Quadtree And Related Hierarchical Data Structures", ACM Computing syrveys, 16(2) (1984) 187-260

17. **JIM DEMMEL,** "Applications of Parallel Computers Lecture Notes", http.cs.berkeley.edu\~demmel\cs267\

18. **L. GREENGARD AND V.ROKHLIN,** "A new version of the fast multipole method for the Laplace equation in three dimensions", Acta Numerica 6,229 (1997)

19. **H. CHENG, L. GREENGARD AND V. ROKHLIN,** "A fast adaptive multipole algorithm in three dimensions", Journal of Computational Physics 155,468-498(1999)

20. **JAMES F. LEATHRUM**, "Parallel Fast Multipole Algorithm"

21. **F. ZHAO** "A O(n) algorithm for the tree dimensional N-body simulations", Technical report , MIT, 1987

22. **R.BEATSON, L. GREENGARD**, "A Short Course on Fast Multipole Method" www.math.canterbury.ac.nz/~mathrkb/ pdfs/beatson+greengard

23. **KEES LEMMENS**, "An Investigation, Implementation And Comparison Of 3 Important Simulation Techniques: Particle-Particle, Particle-Mesh And Tree-Code", Master Thesis, University of Technology Delft, 1997

24. **K. L. CLARKSON**, "Fast Algorithms For The All Nearest Neighbours Problem", Proc. Foundations of Computer Science (1983) 226-232

25. **PIET HUT, JUN MAKİNO**, "The Art of Computational Science", http://www.artcompsci.org

26. **AMARA LYNN GRAPS**, "Investigating the Motions and Energies of Ions Confined in a Uniform Magnetic Field", MS Physics Thesis, San Jose State University, 1991

27. **B. Flaming,** "Practical Data Structures in C++"

28. **PANGFENG LIU**, "The Parallel Implementation of N-body Algorithms", DIMACS Technical Report 94-27 May (1994)

29. **J.K. Salmon**, "Parallel Hierarchical N-Body Methods", Ph.D. Thesis, California Institute of Technology, 1990

30. **M.S. Warren and J.K. Salmon**, "Astrophysical N-Body Simulations Using Hierarchical Tree Data Structures", Proc. Supercomputing '92 (1992) 570- 576

31. **M.S. Warren and J.K. Salmon**, "A Parallel Hashed Oct-Tree N-Body Algorithm", Proc. Supercomputing '93 (1993) 1-12

32. **J.P. Singh**, "Parallel hierarchical N-body methods and their implications for multiprocessors", Ph.D. thesis, Stanford University, 1993

33. **Guy Blelloch and Girija Narlikar,** "A Practical Comparison of N-Body Algorithms", Parallel Algorithms. Series in Discrete Mathematics and Theoretical Computer Science, Volume 30, 1997

34. **William T. Rankin**, "Efficient Parallel Implementations of Multipole Based N-Body Algorithms", Ph.D. Thesis, Duke University, 1997

35. **W. D. Elliott and J. A. Board,** "Fast Multipole Algorithm For The Lennardjones Potential", Technical Report TR94-005, Duke University, Department of Electrical Engineering, August 1994

ÖZGEÇMIŞ

1979 yılında Erzincan'da doğdu. Orta öğrenimini Erzincan Anadolu Lisesi'nde tamamladı. Lise eğitimini Erzincan Fen Lisesi'nde başladıktan sonra son sınıfta geçiş yaptığı Özel Otlukbeli Lisesi'nde ikincilikle tamamladı. Kocaeli Üniversitesi Mühendislik Fakültesi Bilgisayar Bilimleri Mühendisliği Bölümü'nden 2000 yılında mezun oldu. 2001 yılında Gebze Yüksek Teknoloji Enstitüsünde yüksek lisans eğitimine başladı. Halen Kocaeli Üniversitesi Teknik Eğitim Fakültesi Bilgisayar Öğretmenliği Bölümü'nde araştırma görevlisi olarak çalışmaktadır.