KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ZAMANLA DEĞİŞEN HACİMSEL VERİLERİN DAĞITIK SİSTEMLER YARDIMIYLA GÖRSELLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisi Mehmet Bilsay KARADENİZ

TEMMUZ 2006
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI

ZAMANLA DEĞİŞEN HACİMSEL VERİLERİN DAĞITIK SİSTEMLER YARDIMIYLA GÖRSELLEŞTİRİLMESİ

Bilgisayar Mühendisi Mehmet Bilsay KARADENİZ

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsünce
"Bilgisayar Yüksek Mühendisi"
Unvanı Verilmesi İçin Kabul Edilen Tezdir.

Tezin Enstitüye Verildiği Tarih : 09.06.2006
Tezin Savunma Tarihi : 14.07.2006

Tez Danışmanı : Yrd. Doç. Dr. Cemal KÖSE
Jüri Üyesi : Prof. Dr. Rıfat YAZICI
Jüri Üyesi : Yrd. Doç. Dr. Ali GANGAL

Enstitü Müdürü : Prof. Dr. Emin Zeki BAŞKENT

Trabzon 2006

ÖNSÖZ

Zamanla değişen hacimsel verilerin görselleştirilmesi büyük miktarlarda hesaplama ve bellek gerektirmektedir. “Zamanla Değişen Hacimsel Verilerin Dağıtık Sistemler Yardımıyla Görselleştirilmesi” adlı bu çalışmada, küçük boyutlu bilgisayar kümelerinin zamanla değişen hacimsel verilerin görselleştirilmesinde sağlayabileceği avantajlar araştırılmıştır.

Çalışmam süresince bilgi, görüş ve önerileriyle bana yardımcı olan çok değerli danışman hocam sayın Yrd. Doç. Dr. Cemal KÖSE’ ye teşekkürü bir borç bilir şükranlarımı sunarım.

Tez çalışmam sürensince sabır, destek ve sevgileriyle hep yanımda olan aileme ve nişanlıma teşekkür eder saygılarımı sunarım.

Mehmet Bilsay KARADENİZ

Trabzon 2006

İÇİNDEKİLER

Sayfa No

ÖZET

Zamanla değişen hacimsel verilerin görselleştirilmesi büyük miktarlarda hesaplama ve bellek gerektirmektedir. Paralel sistemler bu tür görselleştirmeleri kabul edilebilir süreler içinde yapabilme potansiyelini sunmaktadır. Ağ ve bilgisayar teknolojilerindeki gelişmeler, bilgisayar kümelerini sadece büyük firmaların değil aynı zamanda üniversiteler, küçük araştırma merkezleri gibi görece küçük işletmelerin kullanımına da açmıştır.

Bu tezde küçük bilgisayar kümelerinin zamanla değişen hacimsel verilerin paralel görselleştirilmesinde sağlayabileceği imkanlar incelenmiştir.

Tezde zamanla değişen hacimsel verilenin görselleştirilmesi için paralel splatting algoritması kullanılan iki adet yük dengelemesi yaklaşımı gerçeklenmiştir. Bu çalışmada, paralel hesaplamada standart olan, Mesaj Geçme Arayüzü (MPI) uygulamaların geliştirilmesinde temel araç olarak kullanılmıştır.

Görselleştirmede kullanılan zamanla değişen hacimsel veri Mandelbrot kümesidir. Bu kümesi, dört boyutlu karmaşık sayılar (quaternion) kullanılarak hesaplanmıştır.

Geliştirilen uygulamayla, bilgisayar sayısı; bilgisayar başına düşen süreç sayısı; nesne uzayı dilim sayısı; Gauss maske boyu; yüzey normali hesaplama hacmi; hacimsel verinin boyutu ve yük dengeleme algoritmasının performans üzerindeki etkileri incelenerek sonuçlar değerlendirilmiştir.

Yapılan çalışma sonunda özellikle dengesiz dağılan yüklerde statik yük dağıtımı yapmanın performansı olumuz etkilediği görülmüştür. Buna ek olarak voksel sayısı arttıkça sistem verimi düşmektedir. Ayrıca bilgisayar başına düşen optimum süreç sayısının iki olduğu gözlemlenmiştir.

Anahtar Sözcükler: Hacim Görselleştirme, Paralel Splatting, Zamanla Değişen Hacimsel Veriler, Yük Dengeleme, Dağıtık Sitemler, Mandelbrot Kümesi.

SUMMARY

Time-Varying Volume Data Visualization on Distributed Systems

Time-varying volume data visualization is a computationally intensive process and requires large amount of memory. Parallel processing offers the potential for achieving the visualization in reasonable times. The advances in network and computer technology reduced the cost of the computer clusters and made them available to small companies, research labs and universities.

This thesis discusses the potentials offered by small clusters in parallel visualization of time-varying volume data.

Two load balancing approaches are implemented for an efficient parallel splatting. This parallel splatting approach exploits time and object space-slicing technique. Here, MPI is used for message passing interface. This standard message-passing interface makes it easy for developers to concentrate on parallel programming, not to low level communication details.

The Mandelbrot set is used as time-varying volume data in visualization. This 4D Mandelbrot sets are calculated in the quaternion.

By employing our approach, the effects of the number of computers, processes per computer, volume slices and voxels, Gauss mask size, surface normal extraction volume size, and load balancing algorithms on the performance are examined and the results are evaluated.

As a result of this study it is seen that; especially static load distribution method has negative effects on the performance for unbalanced loads. In addition to this, as the problem size increases the efficiency of the system decreases. Also the optimum number of processes per computers is found to be two.

Keywords: Volume Rendering, Parallel Splatting, Time-Varying Volume Data, Load Balancing, Distributed Systems, Mandelbrot Set.

ŞEKİLLER DİZİNİ

Sayfa No

TABLOLAR DİZİNİ

Sayfa No

SEMBOLLER DİZİNİ

a	: Başlangıç noktası
amb	: Ambient renk değeri
$\alpha(P,R)$	: (P,R) pikseline eklenecek Alpha değeri
$\alpha_i(P,R)$	: (P,R) pikselinin i. örnekte sahip olduğu Alpha değer
$\alpha_{i+1}(P,R)$	: (P,R) pikselinin i. örnek eklendikten sonraki Alpha değeri
b	: İlerleme vektörü
$C(P,R)$	: (P,R) pikseline eklenecek renk değeri
$C_i(P,R)$	: (P,R) pikselinin i. örnekte sahip olduğu değer
$C_{i+1}(P,R)$	: (P,R) pikselinin i. örnek eklendikten sonraki değeri
dif	: Diffuse renk değeri
$\partial x, \partial y, \partial z$	: x, y, z eksenleri arasındaki örnekleme mesafeleri
$\partial \eta$	: Yoğunluk farkı
G	: Gerekli ağ bağlantısı sayısı
Gbps	: Gigabit per second
k_a	: Ambient katsayısı
k_d	: Diffuse katsayısı
k_s	: Specular katsayısı
$\vec{K}$	: Işık yansıma vektörü
I_a	: Ambient renk
I_d	: Diffuse rengi
I_s	: Specular renk
$\vec{L}$	: Işığın geliş vektörü
MB	: Mega Byte
MIMD	: Multiple Instruction Multiple Data
MISD	: Multiple Instruction Single Data
MPI	: Message Passing Interface

$\eta(n)$	: Yoğunluk değeri
n	: Bilgisayar sayısı
$\vec{N}$	: Yüzey normali vektörü
PVM	: Parallel Virtual Machine
Q	: Quarternion (Dört boyutlu karmaşık sayı)
r	: Eş yüzey noktası
$\vec{R}$	: Gözlemci bakış vektörü
$spec$	: Specular renk değeri
SIMD	: Single Instruction Multiple Data
SISD	: Single Instruction Single Data
SPMD	: Single Program Multiple Data
τ	: b vektörü yönündeki ilerleme miktarı
$veri$	: Üç boyutlu hacimsel veri
$(\Delta x, \Delta y, \Delta z)$	: Voksel içinde eksenler yönündeki örnekleme aralıkları
(n_x, n_y, n_z)	: Voksel küpü içindeki herhangi bir noktanın koordinatları
(N_x, N_y, N_z)	: Yüzey normali vektörü
(x_0, y_0, z_0)	: Vokselin referans başlangıç konumu

GENEL BİLGİLER

1.1. Giriş

Günümüzde bilgisayar kümeleri (cluster) birçok alanda kullanılabilmektedir. Özellikle, küresel hava tahmini, deprem benzetimleri (simülasyonları), dünya çekirdeği benzetimleri, kanser tedavisi için protein yapılarının incelenmesi gibi, büyük bilimsel simülasyonlar kabul edilebilir sürelerde ancak dağıtık sistemler yardımıyla gerçeklenebilmektedir.

Bilimsel benzetimlerde elde edilen verinin insanlar tarafından daha kolay incelenmesi ve anlaşılabilmesi için görselleştirme teknikleri kullanılmalıdır. Genellikle, bilimsel benzetimler sonucunda ortaya çıkan, noktasal verilerin geometrik yapıdan yoksun olmaları görselleştirilmelerini zorlaştırmaktadır.

Noktalardan oluşan hacimsel verilerin görselleştirilmesi için birçok yöntem önerilmiştir. Fakat bu yöntemler zamanla değişen hacimsel verilerin görselleştirilmesi söz konusu olduğunda çok yavaş kalmaktadırlar. Hacimsel veriyi sınıflandırmak, görselleştirme algoritması için gerekecek çeşitli parametreleri hesaplamak gibi ön işlemler de göz önüne alındığında görselleştirme çok uzun zaman alan bir işlem haline gelir.

Günümüzde zamanla değişen hacimsel verilerin gerçek zamanlı yada kabul edilebilir sürelerde görselleştirilmesi ancak paralel sistemler yardımıyla mümkündür.

Bu çalışmada zamanla değişen hacimsel verilerin dağıtık sistemler yardımıyla dengeli olarak görselleştirilmesi amaçlanmıştır. Veri kaynağı olarak dört boyutlu Mandelbrot kaotik-fraktal cismi kullanılmıştır. Bu dört boyutlu fraktal cisim zamanla değişen üç boyutlu bir yapı olarak ele alınmıştır. Fraktal olması nedeniyle özellikle bilimsel benzetimler sonucu oluşabilecek birçok görsel yapıyı içermektedir. Mandelbrot fraktalının kullanılmasının bir diğer nedeni ise standart olarak üretilebilmesidir. Bu sayede benzer çalışmalarda aynı veriler kullanılarak karşılaştırmalı sonuçlar elde edilebilir.

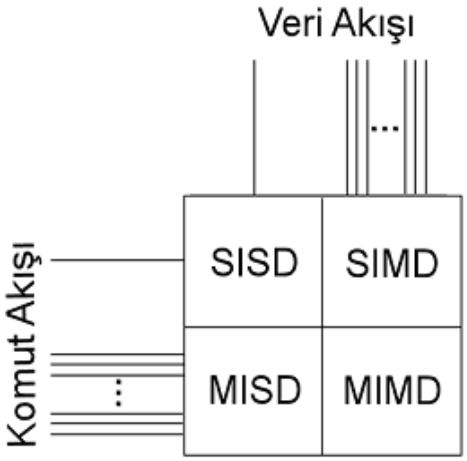
Çalışmada sekiz adet işçi ve bir adet yük dağıtıcı-sonuç toplayıcı olarak dokuz adet bilgisayar kullanılmıştır. Kullanılan sistem bilgisayar kümesi olarak küçük bir küme oluşturuyor olsa da sistem hızlanmalarını temel standart değerler ölçüsünde ölçmeye yeterlidir.

1.2. Paralel Bilgisayar Sistemleri

1.2.1 Paralel Bilgisayar Sistemlerinin Sınıflandırılması

1.2.1.1 Flynn Sınıflandırması

Flynn' e [1] göre bilgisayarlar dört ayrı grupta sınıflandırılabilir. SISD (Single Instruction-Single Data), MISD (Multiple Instruction-Single Data), SIMD (Single Instruction-Multiple Data) ve MIMD (Multiple Instruction-Multiple Data) (Şekil 1.1).



Şekil 1.1. Flynn'e göre bilgisayarların sınıflandırılması [2]

Flynn bu sınıflandırmada komutlarla, komutların üzerinde işlem yaptığı veriler arasındaki ilişkiyi temel almıştır.

SISD sistemler tek komut, tek veri mantığına göre çalışmaktadırlar. Bu tip sistemlerde bir komut bir veri üzerinde işlem yapmaktadır.

MISD sistemlerde ise birçok komut tek bir mantıksal veri yolu üzerinde işlem yapmaktadır. MISD sistemlerin kullanım alanı çok kısıtlı olduğundan pratikte kullanılmamaktadırlar. Ancak bazı yedeklemeli sistemler bu tip çalışmadan yarar sağlayabilmektedir.

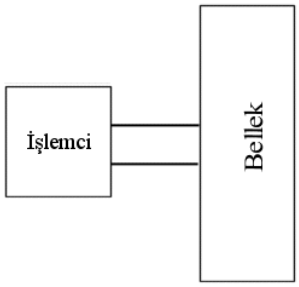
SIMD sistemler bir komutu birçok veri üzerinde işleten sistemlerdir. Birçok bilimsel hesaplamada kullanılabilmektedirler. Örneğin iki adet dizinin toplanıp üçüncü bir diziye yazılması gibi vektör hesaplaması gerektiren durumlarda çok kullanışlıdırlar. Bu nedenle

vektör bilgisayarların temel çalışma mantığını oluşturmaktadırlar.

MIMD sistemler ise birçok komutu birçok veri üzerinde işletebilen sistemlerdirler. Birbirinden bağımsız paralel işlemler gerçekleştirebildiklerinden çok kullanışlıdırlar. Diğer sınıflandırmalara ait bütün çalışma modellerinin özelliklerine sahip olduklarından, bu tür bir sistem diğerleri gibi kullanılabilir. Bu nedenle diğer sistemlerin sahip olduğu bütün avantajlara sahiptirler. Ancak maliyetleri diğer sistemlerden fazla olmaktadır.

1.2.1.2. Tek İşlemcili Bilgisayarlar

Tipik bir bilgisayar sisteminde merkezi bir işlemci birimi ve bu birimin eriştiği bellek bulunmaktadır (Şekil1.2).



Şekil 1.2. Tipik bir bilgisayar sistemi

Tek işlemcili bu sistemler günlük kullanımda çok başarılı olmakla birlikte özellikle büyük miktarda hesaplamanın gerektiği özel durumlara uygun değildir. Düşük maliyete sahip bu sistemler geliştirilerek paralel sistemler oluşturulabilmektedir.

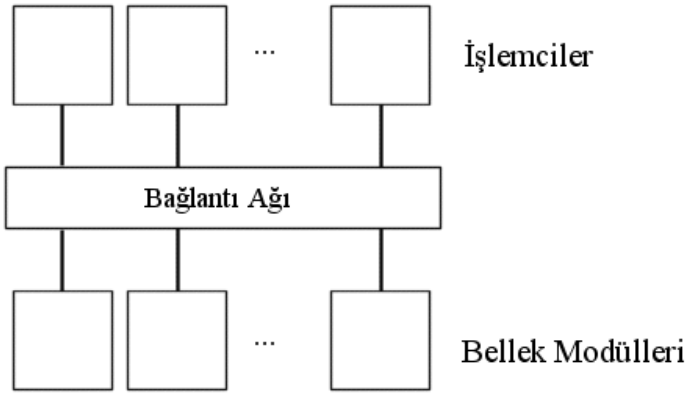
1.2.1.3 Paylaşımlı Bellekli Çok İşlemcili Sistemler

Tek işlemcili sistemleri geliştirmenin en iyi basit yolu ortak belleğe erişen birçok işlemci kullanmaktır (Şekil 1.3). Bu sayede her işlemci aynı belleğe erişebilecektir [3].

Bu tür sistemlerde genellikle işlemciler bir ağ yapısı üzerinden belleğe erişerek sistem belleğini ortak kullanırlar.

Program çalıştırma mantığı ise her işlemci için ortak bir program ve bu programların kullanacağı verilerin yüklenmesi şeklindedir. Programlama arayüzü ya yeni bir paralel

programlama dili geliştirerek yada başka bir dile yapılan ek kütüphanelerle sağlanabilir.



Şekil 1.3. Paylaşımlı bellekli çok işlemcili sistem

Bu tür sistemleri kullanmak programcılık açısından kolaydır. Mantıksal olarak çok basit çalışması ve herkesin ortak bellek uzayını paylaşması bir çok haberleşme ve eşleme (senkronizasyon) işlemini kolaylaştırmaktadır.

Paylaşımlı bellek kullanılması nedeniyle bu tip sistemlerin yüksek performanslı belleklere ihtiyaçları vardır. Birçok işlemcinin aynı anda belleğe erişmeye çalışması bellek performansını düşürerek sistemin çalışma hızını ve ölçeklenebilirliliğini önemli ölçüde etkileyecektir. Bellek alt yapısı olarak çok hızlı ve bant genişliği çok yüksek sistemler oluşturmak zor olduğundan işlemci sayısı arttıkça bellek bant genişliği arttırılamamakta ve sistem verimsiz hale gelmektedir.

Bellek bant genişliği arttırılamadığı durumlarda işlemcilere hızlı önbellekler eklenerek performans artışı sağlanabilir. Büyük önbellekli işlemciler daha az bellek erişimi yapmaya ihtiyaç duyacağından bellek bant genişliği daha optimum şekilde kullanılabilecektir.

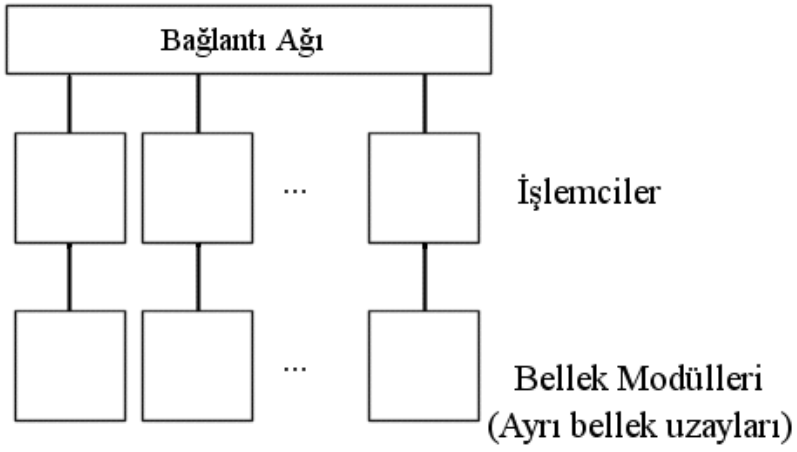
Bütün bu avantajlarına rağmen önbellek kullanımı, önbellekteki verilerin tutarlılığı ile ilgili problemler ortaya çıkarmaktadır. Bir işlemci önbelleğindeki veriyi değiştirdiğinde veya belleğe yazma işlemi yaptığında diğerlerinin de bundan haberdar olmasını sağlamak zorunda kalmaktadır. Bu ise sistemin performansını azaltmakta ve bu iş için kullanılacak donanımın eklenmesini gerektirmektedir.

Bir diğer dezavantaj ise sisteme eklenecek ek bellek ve donanımların maliyetidir. Bütün bunlar göz önüne alındığında optimum işlemci sayısı genellikle 4-8 arasında sınırlı

kalmaktadır. Daha fazla işlemci eklemek maliyet açısından büyük yük getirmektedir. Bu nedenlerden dolayı bu tür sistemlerin kullanımı küçük ölçeklerle sınırlıdır.

1.2.1.4. Dağıtık Bellekli Sistemler

Dağıtık bellekli sistemler paylaşımlı bellekli sistemlerden farklı olarak birçok tekil bilgisayar sisteminin birleşmesiyle oluşturulurlar. Paylaşımlı bellekli sistemlerde bulunan bellek erişim sistemi ortadan kaldırılarak, işlemcilerin birbirleriyle haberleşmesi için bir ağ sistemi oluşturulmuştur (Şekil1.4). Bu sayede her işlemci kendi belleğine erişir ve diğerleriyle haberleşebilir.



Şekil 1.4. Dağıtık bellekli sistem

Böyle bir sistemde her işlemci diğer işlemcilerin kullanamayacağı bir belleğe sahip olmaktadır. Bütün işlemciler kendi başlarına hareket ederler, ancak birbirleriyle haberleşebildiklerinden çeşitli mesajlarla veri gönderip alabilmeleri mümkündür.

Bu tür sistemlere eklenecek her bilgisayar kendi belleğine sahip olacağından toplam bellek bant genişliği sistem büyüdükçe artmaktadır. Ancak programlama ve işlemcileri bir birine bağlayan ağ yapısı ile ilgili çeşitli problemler ortaya çıkabilmektedir.

Programlama açısından bu tür sistemleri iyi organize etmek gerekmektedir. Çözülecek problem mümkün olduğunca bir birinden bağımsız parçalara bölünmeli ve haberleşme en aza indirilmelidir. Fakat bu her zaman kolay olmamaktadır. Haberleşmenin yoğun olduğu durumlarda bu tür sistemler istenilen performans artışını vermemekte, bazı

durumlarda performans kaybına neden olmaktadırlar.

İşlemcilerde koşan bütün süreçlerin birbirinden bağımsız olması bu tür sistemlerin programlamasında kullanılacak yeni programlama arayüzlerinin oluşturulmasını gerektirmektedir. Sistemlerin programlanmasında kullanılacak olan arayüz yeni bir programlama dili olarak tasarlanabileceği gibi mevcut programlama dillerine bir ek kütüphane olarak da eklenebilir. Programcı açısından bakıldığında yapılacak haberleşmenin de organize edilmesi gerekecektir. Verilerin ortak bir alanda tutulmaması nedeniyle her süreç kendini hesaplaması için gerekli olan veriyi mesaj olarak alacak ve kendi belleğine kopyalayacaktır. Bu nedenle ortak bilgilere ihtiyaç duyulduğu durumlarda belek kullanımı artacak ve verim düşecektir.

Özellikle haberleşmenin organizasyonu programın tasarımının daha uzun sürede yapılmasına ve oluşabilecek problemlerin daha zor ayıklanmasına neden olacaktır. Bu tür programlarda hata ayıklamak oldukça zor olmakta ve özel bazı araçlar gerektirmektedir.

Bütün bu olumsuzluklara rağmen dağıtık bellekli sistemler birçok durumda oldukça performanslı olarak çalışabilmektedirler. Özellikle haberleşmenin fazla gerekmediği, daha çok hesaplama yükünün ön planda olduğu durumlarda bu tür sistemler oldukça ölçeklenebilir olmaktadırlar.

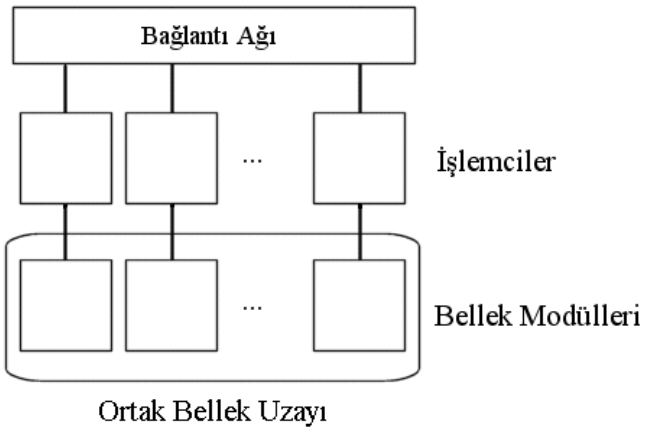
1.2.1.5. Dağıtık-Paylaşımlı Bellekli Sistemler

Paylaşımlı bellekli sistemlerin programlanmasının daha kolay olması nedeniyle tercih edilmesi yeni bir kavramın tartışılmasına neden olmuştur. Dağıtık-paylaşımlı bellekli sistemler kavramı, paylaşımlı bellekli sistemlerin programlama kolaylıklarına sahip olmak ve aynı zamanda sistem ölçeklenebilirliğini korumak için ortaya atılmıştır.

Dağıtık-paylaşımlı bellekli sistemler dağıtık bellekli sistemlere oldukça benzemektedir. İşlemciler yine kendi belleklerine erişebilmektedirler fakat çalışma alt yapısı Şekil 1.5’ deki gibi değiştirilerek diğer işlemcilerin belleklerine de aynı bellek uzayındaymış gibi erişim imkanı sağlanmıştır. Bu sayede bütün işlemciler sanki tek bir bellek uzayında koşuyor gibi işlem yapabilirler.

Belleğin dağıtık olması nedeniyle yerel bellekteki verilere erişim hızlı olurken uzaktaki işlemcinin belleğine erişmek yavaş olacaktır. Fakat programlama açısından bakıldığında aynen paylaşımlı bellekli sistemlerde olduğu gibi programlama yapmak

mümkün hale gelmiştir. Ayrıca ölçeklenebilirlilik korunmuş olacağından performans da fazla olacaktır.



Şekil 1.5. Dağınık-paylaşımlı bellekli sistem

Bütün avantajlarına rağmen bu tür sistemlerin bazı problemleri olmaktadır. Her şeyden önce uzaktaki belleğe erişmeye çalışırken hangi belleğin hangi işlemcide olduğunun bilinmesi gerekecektir. Bu problem genellikle işlemcilerde bulunan bir erişim tablosu yardımıyla aşılmaktadır. Bu tür sistemlerde bir diğer problem ise gerekli donanımsal alt yapının pahalı olmasıdır. Sistem dağıtık bellekli sistemler gibi çalıştığı için mesaj alt yapısını programcıdan saklayabilmek için özel donanımlar tasarlanması gerekecektir.

Bu tür sistemlerde önbellek kullanılması büyük hız avantajına sahip olmakla birlikte, bazı problemleri de beraberinde getirmektedir. Önbellek, sistemin uzaktaki belleği tekrar tekrar istemesini önleyerek performansı arttıracak fakat önbelleğe yazılan bir verinin değişmesi durumunda bütün diğer işlemcilerin haberdar edilmesi gerekecektir. Bu da donanımsal olarak ek maliyet anlamına gelmektedir.

1.2.2. Paralel Bilgisayarlarda Ağ Topolojileri

1.2.2.1. Noktadan Nokataya Tamamen Bağlı Bağlantı

Bu topolojide bütün bilgisayarlar diğer bütün bilgisayarlarla haberleşmek için

adanmış ağ arayüzlerine sahiptirler (Şekil 1.6). Bu tip bağlantının avantajı bütün bilgisayarların istedikleri bilgisayarla istedikleri zaman iletişim kurabilmesidir. En önemli dezavantajı ise sistemin ölçeklenebilir olmamasıdır. Sisteme giren her bilgisayar diğer bütün bilgisayarlarla bağlantıya sahip olacağından 8-10 bilgisayardan sonra sistem ölçeklenebilir olmaktan çıkacaktır.

Sistemde gerekli olacak network bağlantısı sayısı aşağıda verilmiştir.

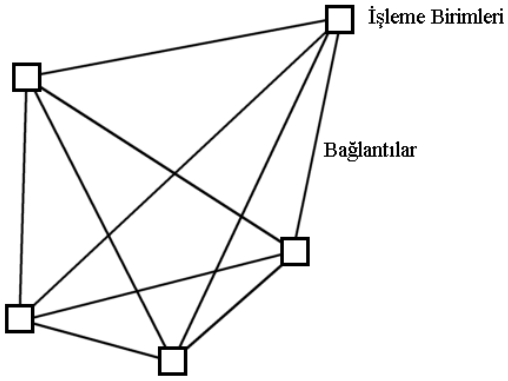
$$G = (n \times (n-1))/2 \qquad (1)$$

Burada;

n : bilgisayar sayısı,

G : gerekli ağ bağlantısı sayısıdır.

8 bilgisayar için böyle bir topoloji kurulmak istense 28 ağ bağlantısına sahip olmak gerekecektir. Bu sayı arttıkça ağ kurmak imkansızlaşacaktır.

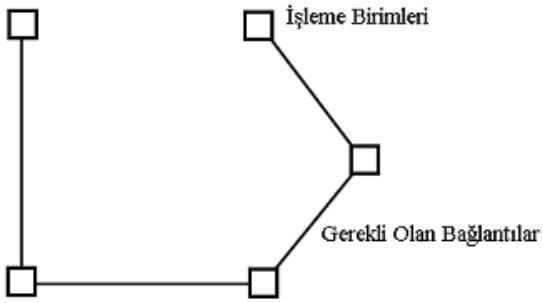


Şekil 1.6. Noktadan noktaya tamamen bağlı bilgisayarlar

1.2.2.2. Çizgi Şeklinde Bağlantı (Line)

Bu tip bağlantılarda her bilgisayar diğer iki bilgisayarla bağlantı kurmakla birlikte ağ yapısı iki uçtan açıktır (Şekil 1.7).

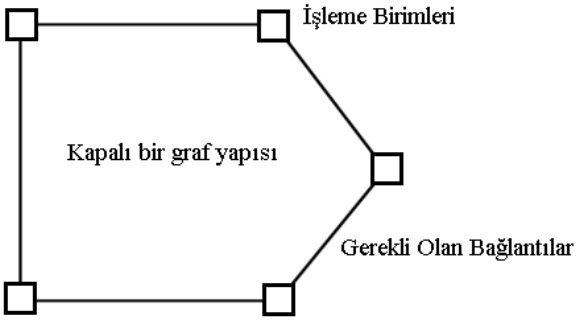
Gerekli olan bağlantı sayısı az olmakla birlikte bilgisayarlardan birinin arızalanması ağda ciddi problemlere neden olacaktır. Aynı zamanda tek hattın kullanılması nedeniyle iletişim gecikmesi artacak ve bant genişliği sınırlanacaktır.



Şekil 1.7. Çizgi halinde bağlantıya sahip bilgisayarlar

1.2.2.3. Dairesel Bağlantı (Ring)

Bu tip bağlantılarda her bilgisayar diğer iki bilgisayarla bağlantı kurmaktadır (Şekil 1.8). Dairesel bağlantı esasen özel bir tip çizgi halinde bağlantıdır. Tek fark ağın iki ucunun da birbirine bağlanmasıdır. Paket sonlandırması yeni bir sorun olarak ortaya çıkmakla birlikte erişilebilirlik ve güvenirlilik artmıştır.

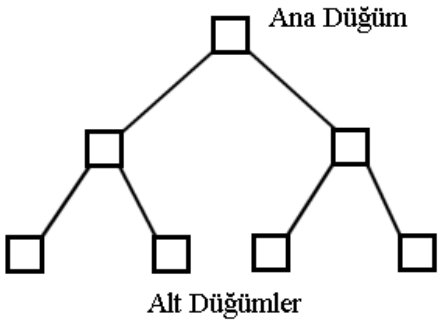


Şekil 1.8. Dairesel bağlantıya sahip bilgisayarlar

Bu tip ağlar çizgi şeklinde bağlantıya sahip ağlarla birçok ortak özelliği paylaşmaktadırlar. Örneğin birinci bilgisayarın ağın ortasındaki bilgisayara erişmesi çok uzun zaman alırken yanındaki bilgisayarlara erişmesi daha çabuk olacaktır. Topolojideki bir diğer zayıflık ise bir bilgisayarın arızalanması durumunda ağın kullanımının kısıtlanması ve ağ performansının düşmesidir. Bir birinden uzak iki bilgisayarın arızalanması durumunda ise ağın bir kısmı diğerinde ayrılacak ve ağ kullanılamaz hale gelecektir.

1.2.2.4. Ağaç Yapısı

Bu tip ağlar genellikle böl-yönet problemleri için uygundurlar. Bilgisayarlar birbirlerine ağaç yapısıyla bağlanırlar (Şekil 1.9).



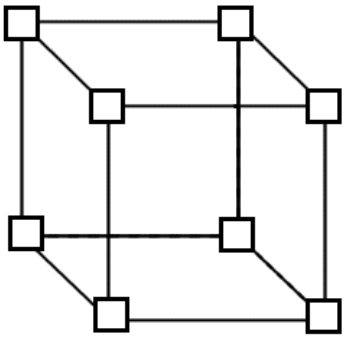
Şekil 1.9. İkili ağaç yapısı halinde birbirine bağlanmış bilgisayarlar

Kullanım alanının kısıtlı olması ve sadece belirli problemlerde iyi sonuç vermesi dışında bu ağlarda oluşabilecek olumsuz durumlardan en önemlisi ağın üst kesiminde bulunan bilgisayarlara ilerlendikçe bant genişliğinin yetmemeye başlamasıdır. Birçok bilgisayarın ağacın üst düğümlerinden geçecek şekilde haberleşme yapması durumunda sistem performansı oldukça düşecektir.

1.2.2.5. N-Boyutlu Kübik Ağ

Ağ bağlantı sayısının optimize edilmesinde kullanılabilecek bir diğer bağlantı şekliyse N-boyutlu kübik ağ yapısıdır (Şekil 1.10). Bu tip ağlar nispeten az bağlantıyla yüksek hızlara çıkmaya olanak sağlarlar. Boyut arttırıldıkça ağ bağlantısı sayınsının artması nedeniyle ölçeklenebilirliği azalsa da diğer kısıtlı ölçeklenebilir ağ topolojilerinden daha iyi sonuçlar vermektedir.

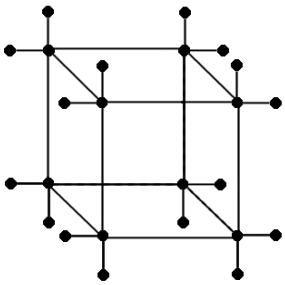
Bu tip ağlarda bir düğümden diğer düğüme giden en kısa yol mümkün olduğunca küçük tutulmaya çalışılmıştır. Bu sayede ağ gecikmesi ve ağ bağlantı sayısı optimum şekilde ayarlanmıştır.



Şekil 1.10. 3-Boyutlu kübik topoloji

1.2.2.6. Karışık Ağ Topolojileri

Birçok ağ topolojisinin çeşitli avantajları ve dezavantajları bulunmaktadır. Bu nedenle birçok ağ yapısı uygun şekilde birleştirilerek farklı tipte ağ topolojileri yaratılabilir. Örneğin bir ağaç yapısı bir küple birleştirilerek az sayıda bağlantıyla yüksek performans elde edilmeye çalışılabilir (Şekil 1.11).



Şekil 1.11. Her düğümü ağaç yapısında olan bir 3-boyutlu kübik ağ

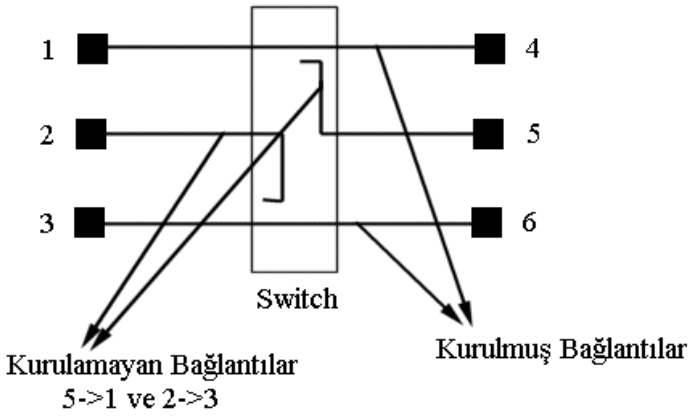
1.2.2.7. Switch ve Hub Arasındaki Farklar

Switch ile hub arasındaki temel fark; hublarda gönderilen verinin bağlı olan bütün bilgisayarlara gitmesi, switchlerde ise sadece gönderilen bilgisayarın veriyi almasıdır.

Switchler verileri sadece gidecekleri bilgisayara gönderdiklerinden ağda giden verileri değerlendirmeleri gerekmektedir. Bu nedenle ek donanıma ihtiyaç duymaktadırlar. Ancak son dönemlerde ağ teknolojilerinin gelişmesiyle switchler daha çok kullanılmaya başlanmıştır. Switchler yönetilebilir ve yönetilemeyen olmak üzere iki gruba ayrılırlar, ek

maliyet nedeniyle yönetilebilir switchler pahalı olmaktadırlar.

Switchlerin temel avantajı aynı anda bağlantı kurmak isteyen farklı bilgisayarların bağlantı kurmalarına izin vermesidir. Bu sayede aynı anda bir çok bağlantı kurulabilmektedir. Ancak bağlantı halinde olan bilgisayarlara erişim olmamaktadır (Şekil 1.12).



Şekil 1.12. Switch ile bağlantı

Hubların kullanım alanları ise son derece kısıtlıdır. Özellikle gönderilen verinin bütün bilgisayarlara gitmesi aynı anda farklı bilgisayarların bağlantı kurmasını engellemektedir. Ancak bu dezavantaj ağda broadcast (herkese duyurma) yapıldığında bir avantaj olmaktadır.

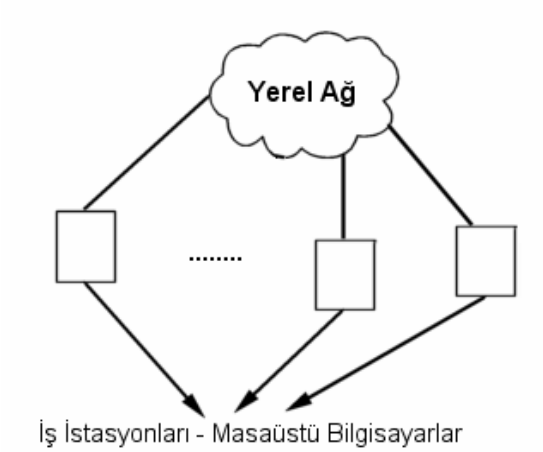
1.2.3. Bilgisayar Kümeleri (Cluster)

Bilgisayar kümeleri (cluster) birçok bilgisayarın bir ağ ile birbirine bağlanmasıyla oluşturulan sistemlerdir (Şekil 1.13).

Özellikle internetin yaygınlaşması ve iş yerindeki bilgisayarların ağlar yardımıyla birbirlerine bağlanması paralel hesaplamada kullanılabilecek yeni bir fırsat oluşturmuştur. Ağ arayüzlerinin sık kullanılması zamanla donanım maliyetini düşürerek bilgisayar kümleriyle hesaplama yapılmasının önünü açmıştır.

Pahalı olmayan kişisel bilgisayarların oluşturduğu ağlar, paralel hesaplamalarda birçok avantaj sağlamaktadır. Özellikle bu tür bilgisayarların pahalı olmaması, sistemin

dağınık yapısı nedeniyle kolaylıkla bilgisayar eklenip çıkarılabilmesi gibi avantajlar bugün birçok süper bilgisayara alternatif olarak bilgisayar kümelerinin tercih edilmesine neden olmaktadır.



Şekil 1.13. Tipik bir bilgisayar kümesi yapısı

Bilgisayar kümelerinin bir diğer avantajı ise alt yapısının temellerinin hazır olmasıdır. Bu tür sistemlerde mevcut yazılımlar ya hiç değiştirilmeden yada çok az bir değişiklik yapılarak çalıştırılabilmektedir. Sisteme yeni bilgisayarlar eklendiğinde, eğer yazılan program uygunsa, hemen performans artışı elde edilebilmektedir. Uygun olmayan birçok program ise ufak bazı değişikliklerle yeni sistemde kolayca çalışabilmektedir.

Bilgisayar kümelerinde sistem performansının arttırılması için sistemi yenilemek gerekmemektedir. Sisteme eklenen her bilgisayarlarla sistem performansı arttırılabilmektedir. Bugünün kaynaklarıyla oluşturulan bir bilgisayar kümesi ileride daha hızlı bilgisayarlar eklenmesiyle daha da hızlanacak ve bilgisayarların kullanım ömürleri arttırılmış olacaktır. Bu durum, çok hızlı gelişen bilgisayar endüstrisinin etkisiyle kısa sürede demode olan bilgisayar sistemlerinin ekonomik olarak kullanılabilecekleri zamanı arttırarak maliyeti düşürecektir.

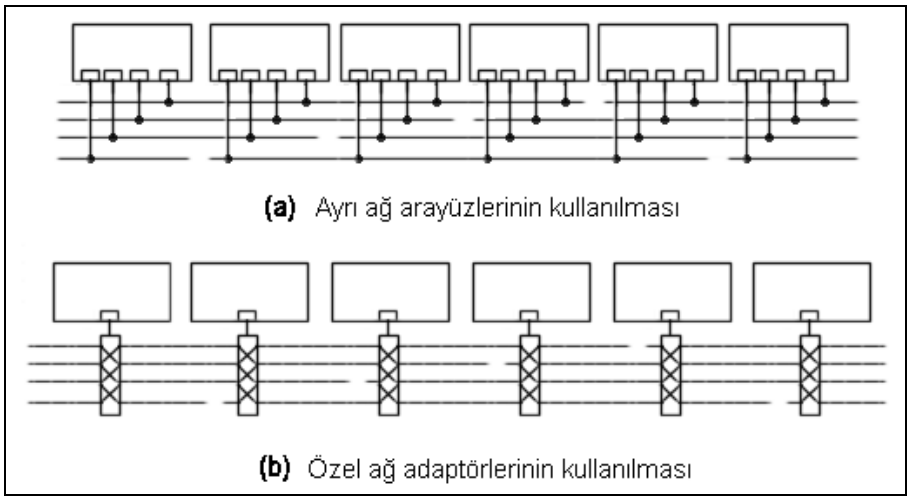
Bugün ağ standardı olarak Ethernet standardını kullanmaktayız. Yakın zamana kadar çok ucuza erişilebilen 100 megabitlik Ethernet arayüzleri standart olarak bilgisayarlarla beraber gelmekteydi. Günümüzde ise giderek ucuzlayan 1 gigabitlik Ethernet arayüzleri daha çok kullanılmaktadır.

İletişim protokolü olarak ise internetin yaygınlığı nedeniyle IP protokolü kullanılmaktadır. Bazı durumlarda UDP/IP protokolü kullanılmakla beraber özellikle daha

güvenilir olması nedeniyle TCP/IP protokolü de tercih edilmektedir.

Bilgisayar kümeleri bir çok ağ topolojisiyle birlikte oluşturulabilmektedir. Çeşitli switchler ve hublar kullanılarak istenilen ağ topolojisi kurulabilmekte, performans-maliyet analizi kolaylıkla yapılabilmektedir.

Bilgisayar kümelerinde dikkat edilmesi gereken iki nokta mevcuttur. Bunlardan ilki sistemde bant genişliği darboğazı yaşanmaması için gerekli ağ topolojisinin oluşturulması gerekliliğidir. Bir çok bilgisayarın aynı anda aynı bilgisayara veri göndermesi ağda bant genişliğini kısıtlayacağından mümkün olduğunca, fazla iletişim halinde olan, bilgisayarların bant genişlikleri arttırılmalıdır. Bu birden çok ağ arayüzü kullanılarak (Şekil 1.14(a)) yada tek ağ arayüzü ve özel adaptörler kullanarak (Şekil 1.14(b)) sağlanabilir. Ayrıca ağda oluşabilecek gereksiz trafiğin engellenmesi içinde ağ bölümlere ayrılabilir (Şekil 1.14).



Şekil 1.14. Çoklu hat kullanarak ağ performansının iyileştirilmesi [3]

İkinci olarak kullanılacak ağ arayüzlerinin bilgisayar alt yapısına uygun şekilde seçilmesi gerekmektedir. Kabloların kalitesine ve bilgisayarda ağ arayüzlerinin takılı bulunduğu arayüz biriminin bant genişliğinin yeterli olmasına dikkat edilmelidir.

Her ne kadar genel amaçlı ağlar kısıtlı bant genişliği ve yüksek gecikmeleri nedeniyle büyük miktarda verilerin gönderilmesini engelliyorsa da, bilgisayar kümesi sistemleri giderek daha çok kullanılmakta ve giderek günümüzün süper bilgisayar standardı olmaktadırlar.

Bilgisayar kümeleri genellikle dağıtık bellekli sistemler gibi çalışmakla birlikte bazı

işletim sistemi yamalarının yardımıyla dağıtık-paylaşımlı bellekli sistemler gibi de çalışabilmektedirler.

Dağıtık bellekli sistemlerde genellikle PVM yada MPI programlama arayüzleri kullanılmaktadır.

1.2.4. Bilgisayar Kümelerinde Kullanılan Programlama Arayüzleri

1.2.4.1. PVM (Parallel Virtual Machine)

PVM Unix ve Windows makinelerin birlikte bir ağ üzerinde çalışarak tek bir bilgisayarmış gibi hareket etmesini sağlamayı amaçlayan bir programlama arayüzüdür [4]. Genellikle problem parçalara ayrılarak FORTRAN yada C de çeşitli makinelerde çalışabilecek şekilde derlenir. Daha sonra programı koşturmadan önce programın çalışacağı makinelerin bir listesi hazırlanır ve PVM' i oluşturacak makine tanımlanmış olur. Bir diğer yöntem ise önce bir süreç çalıştırıp daha sonra PVM kontrol komutları kullanılarak diğer süreçleri çalıştırmaktır.

PVM' de süreç sayısı ile işlemci sayısının birbirine eşit olmasına gerek yoktur. PVM servisi süreçleri otomatik olarak işlemcilere dağıtmaktadır. Buna rağmen programcı istediği makinelere istediği işleri dağıtabilmektedir. Bu tip bir çalışma özellikle programlar deneme aşamasındayken çok yararlıdır. Program bilgisayar kümesinde çalıştırılmadan önce tek bilgisayarda çok süreçli şekilde çalıştırılarak koddaki hataların kısa sürede bulunması sağlanılabilir. Aynı zamanda bilgisayar kümesine fiziksel olarak erişilemediği durumlarda da bu tür bir çalışma stili çok yararlı olmaktadır.

PVM' de süreçler pvm_spawn() fonksiyonuyla yaratılır. Programlar genellikle bir yönetici ve bu yöneticinin etrafında çalışan işçiler şeklinde dizayn edilmektedir. Bu sayede ilk olarak yönetici çalıştırılır daha sonra yönetici süreç pvm_spawn() fonksiyonuyla diğer süreçleri çalıştırır. PVM' de aynı zamanda pvm_spawn() ile çalıştırılan süreçler tekrar pvm_spawn() fonksiyonunu çağırarak yeni süreçler oluşturabilirler.

Süreçler başlatıldıktan sonra pvm_mytid() fonksiyonuyla süreç numaralarını alabilirler. Bu fonksiyon aynı zamanda süreç grubuna üye olunmasını da sağlayan bir fonksiyondur. Fakat çoğu zaman bu fonksiyonu çağırmaya gerek yoktur, çünkü pvm_spawn() fonksiyonuyla oluşturulan her süreç otomatik olarak üye olmaktadır.

PVM programları bittiklerinde pvm_end() fonksiyonunu çağırarak mevcut süreç

grubuna üyeliklerini sonlandırmalıdırlar.

İletişim için bir çok gönderme alma fonksiyonu vardır. PVM' de bütün gönderme işlemleri bloklanmasız olmaktadır. Alma işlemleri ise bloklanmalı yada bloklanmasız olabilmektedir. Gönderme ve alma işlemleri genellikle mesajlaşma tamponuyla olmaktadır. Bir gönderme işleminde mesaj, gönderme tamponuna yazılır ve PVM servisinin bunu göndermesi istenir. Daha sonra mesaj karşıya gittiğinde alma tamponuna yazılarak bir alma işleminin gerçekleşmesi için uygun hale gelir. PVM aynı zamanda mesaj türlerinin belirlenmesi için her mesaja belli bir mesaj etiketi (message tag) verilmesini ister. Bu sayede gelen mesajın ne tür bir mesaj olduğunu anlaşılabilir.

PVM' de pvm_send() ve pvm_receive() fonksiyonları, en temel, alma gönderme fonksiyonlarıdır. Eğer gönderilecek veri dizi halindeyse pvm_psend(), pvm_receive() fonksiyonları kullanılabilir. Bir diğer gönderme-alma şekli ise paketleyerek gönderme-almadır. Bu tip işlemlerde önce PVM' de bulunan komutlarla bir gönderme belleğine paketleme işlemi yapılarak çeşitli tipte veriler yazılır. Daha sonra alıcı tarafından bu paketlenmiş veriler yine PVM' in sunduğu paket açma fonksiyonlarıyla beraber açılarak kullanılır. Bu işlem karmaşık veri yapıları gönderilmek istendiğinde programcıya biraz zorluk çıkarmakla birlikte, özellikle C' de structure tipindeki yapıların gönderilmesine olanak sağlamaktadır.

PVM aynı zamanda broadcast (herkese duyurma), multicast (bir gruba duyurma), scatter (dağıtma), gather (toplama) ve reduce (azaltma) gibi işlemleri yapmak için de çeşitli fonksiyonlara sahiptir.

1.2.4.2. MPI (Message Passing Interface)

MPI programlama Arayüzü PVM programlama arayüzüne oldukça benzemektedir. Bir çok özelliği ortak olmakla birlikte MPI bazı farklı gelişmiş özelliklere sahiptir. Aynı zamanda MPI in tasarlamasındaki en büyük amaç bütün makinelerde çalışabilecek ortak bir arayüzün ortaya çıkarılmasıdır. MPI sadece standartları belirlemektedir. Bu standarda uyan bütün arayüzler MPI' ı çalıştırabilmektedirler. Bu sayede platformdan bağımsız kod yazmak mümkün hale gelmiştir.

Uzun süren tartışmalar sonucunda oluşan MPI arayüzü birçok yeteneği bünyesinde bulundurmaktadır. Veri mesajlaşmasında çok geniş bir alma-gönderme fonksiyonlar topluluğuna sahiptir. Bu sayede birçok amaç için kullanılabilmektedir. MPI 1 standardında

120 den fazla fonksiyon bulunmaktadır. Bu geniş arayüzü PVM in kullanılmasındaki bazı zorlukları gidermiştir.

MPI da süreç çalıştırma yöntemi tamamen tasarıma bağlanmıştır. Standart bir süreç çalıştırma mekanizması bulunmamaktadır. Fakat genellikle PVM' deki gibi bir bilgisayar listesi oluşturulup belli bir sürecin koşturulması sağlanmaktadır. Aynı zamanda bir çok MPI implementasyonu standart olarak komut satırından süreçler çalıştırabilmektedir. MPI' ın standartlarında süreçlerin hangi bilgisayarlara dağıtılacağını belirlemek için bir yol bulunmamaktadır. Bu mekanizma genellikle ya otomatik olarak yada belli bazı kurallara bağlı olarak işlemektedir. Fakat bir çok MPI arayüzünün kaynak kodu mevcut olduğundan istenilen durumlarda özel olarak süreç yaratılmasını sağlayan algoritmalar uygulanabilir.

MPI programları çalıştırıldıklarında MPI_Init() fonksiyonunu çağırmalıdırlar. Bu sayede çeşitli iç ayarlamalar yapılabilmektedir. Program bittiğinde ise MPI_Finalize() fonksiyonu çağırılmalıdır. Bu sayede MPI sevisi kaynakları sorunsuz bir şekilde kullanabilir.

MPI' da gönderme-alma işlemleri yapılırken iletişimciler kullanılır. Bir MPI programı çalıştırıldığında otomatik olarak bütün çalışan süreçleri kapsayan MPI_COMM_WORLD iletişimcisine bağlıdır. Bu iletişimci bütün süreçleri kapsadığından her sürece mesaj göndermek için uygundur. Programcının isteğine bağlı olarak yeni iletişimciler oluşturulabilir. İletişimciler sayesinde gönderilen mesajların nerelere gönderilip nerelere gönderilmeyeceği denetlenebilir. Aynı zamanda mesaj türü denetimi için PVM' e benzer şekilde mesaj belirleyici etiketler kullanılmaktadır.

Çalışan her süreç 0 ile n-1 arasında bir süreç numarasına sahip olur. Bu numara sayesinde belli işler denetlenebilir.

MPI' ın en büyük özelliklerinden biri tek sürecin koşulabilmesidir. Bu modele SPMD (Single Program Multiple Data) ismi verilmektedir. Bu çalışma modelinde her süreç bir diğerinin aynıdır. Bu nedenle yönetici ve işçi işlevlerini sağlayacak kod aynı programın içine yazılır ve her sürecin kendi numarasına bakarak ne iş yapması gerektiğini anlaması beklenir. Programı bu şekilde tasarlamak ufak tefek zorluklar içerse de süreçleri kontrol etmek oldukça kolay olacaktır.

MPI kodları genellikle basit bir kalıba oturmaktadırlar. MPI programlarının kodları genel olarak aşağıdaki gibidir.

MPI_Programı()

```
1:      …
2:      MPI_Init(&argv,&argc);
3:      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
4:      if ( rank = = 0 )
5:              Yönetici();    //Yönetici fonksiyon
6:      else
7:              Isci();        //İşçi fonksiyon
8:      ...
9:      MPI_Finalize();
10:     …
```

Bu tür bir yaklaşımın bir diğer avantajı ise her sürece aynı komut satırı argümanlarını vermenin mümkün olmasıdır.

Burada uygulanan tek süreç-çoklu veri yaklaşımının belki de tek dezavantajı her sürecin ayrı bir işçi fonksiyon gerektirmesi durumunda belleğin optimum kullanılamayacağı gerçeğidir. Fakat bu tür bir durumun genelde karşımıza çıkmamaktadır. Ayrıca gereken fonksiyonlar dinamik kütüphanelerle yüklenerek kullanılabilmektedir. Bu sayede bu tür bir bellek israfının önüne geçilmiş olur.

Mesaj geçme sistemleri arasında en hatasız çalışan arayüzlerden biri MPI alma-gönderme arayüzüdür. MPI mesaj alma-gönderme arayüzü tasarım bakımından eşleşmemesi geren alma-gönderme işlemlerinin olmasına izin vermez. Bu özelliğin sağlanmasında mesaj etiketleri ve iletişimcilerin payı büyüktür.

MPI veri alma-gönderme fonksiyonları çeşitli özelliklere sahip olabilmektedir. Standart olarak tanımlanan fonksiyonlar MPI_Send() ve MPI_Recv() fonksiyonlarıdır. MPI_Send() fonksiyonu parametre olarak verinin adresini, kaç veri yollanacağını, veri tipini, verinin gideceği sürecin numarasını, mesaj etiketini ve mesajın iletileceği iletişimciyi almaktadır (MPI_Send(adres, adet, tip, kime, etiket, iletişimci)). MPI_Recv() fonksiyonu ise parametre olarak verinin adresini, alınacak en fazla veri miktarını, veri tipini, verinin geleceği sürecin numarasını, mesaj etiketini, iletişimciyi ve iletişimle ilgili bilgileri döndüreceği bir durum değişkeni almaktadır (MPI_Recv(adres, maksimum_adet, tip, kimden, etiket, iletişimci, durum)). Burada dikkat edilmesi gereken MPI' ın aldığı verilerin boyutunu bilmemesi sadece alabileceği en fazla verinin boyutunu bilmesi. Ne kadar verinin alındığının öğrenilmesi fonksiyon çağırıldıktan sonra parametre olarak adresi

gönderilen durum değişkenine bakılarak bulunabilmektedir. Bu sayede sabit uzunluklu olmayan verilerin alınabilmesine imkan sağlanmıştır. Bir alma fonksiyonunun gerçekleşebilmesi için , parametrelerine uygun verinin başka bir süreç tarafından kendine gönderilmiş olması gerekmektedir. Eşleşme ancak iki fonksiyonunda parametrelerinin bir birine eşit yada uyumlu olması durumunda gerçekleşebilmektedir.

MPI arayüzünde standart, tamponlu (buffered), senkron (synchronous), ve hazır (ready) olmak üzere dört çeşit gönderme fonksiyonu mevcuttur.

Standart modda, gönderme işleminden önce herhangi bir eşleşen alma fonksiyonunun çağırılmış olması gerekmemektedir. Bu fonksiyon için gereken tamponlamanın nasıl yapılması gerektiği MPI standardı tarafından belirtilmediğinden bu tamamen üretici tarafından belirlenir.

Tamponlu modda gönderme işlemi karşı tarafta eşleşen bir alma işlemi olmadan başlayıp bitebilir. Fakat bu işlemi yapabilmek için gerekli olan tamponun MPI_Buffer_attach() fonksiyonuyla sağlanması gerekmektedir. Gönderme işlemi tamamlandığında MPI_Buffer_detach() fonksiyonu kullanılarak işlemde kullanılan tampon başka işlemlerde kullanılmak için boşaltılabilir.

Senkron göndermede ise, alma yada gönderme fonksiyonları birbirlerinden önce başlayabilirler fakat ancak ve ancak eşleştiklerinde sonlanırlar.

Son olarak hazır modunda verinin gönderilebilmesi için gönderilen mesajla uyumlu bir alma işleminin yapılmış olması gerekmektedir. Aksi halde bir hata oluşturulur.

Kural olarak her gönderme fonksiyonu parametreleri uygun olduğunda her alma fonksiyonuyla çalışabilmektedir. Ayrıca fonksiyonlar bloklanmalı yada bloklanmasız olarak da çalışabilmektedirler.

Bu fonksiyonlara ek olarak MPI bazı özel gönderim fonksiyonlarına da sahiptir. MPI_Bcast(), MPI_Gather(), MPI_Scatter(), MPI_Alltoall(), MPI_Reduce(), MPI_Reduce_scatter(), MPI_Scan() gibi çok çeşitli ve çok kullanışlı fonksiyonlar MPI' da standart olarak bulunaktadır. Bu fonksiyonların çeşitli modlarda çalışanları da ayrıca mevcuttur.

MPI' da senkronizasyon için bariyer kullanılmaktadır. MPI_Barrier() fonksiyonu belli bir iletişimciye bağlı olan bütün süreçler bu çağrıya gelmedikleri sürece , bu çağrıyı yapmış olan süreçlerin beklemesini sağlar. Bu sayede bütün süreçler belli bir noktada birbirlerini beklemek zorunda kalırlar ve senkronize olurlar. Bu fonksiyon genellikle bir işlemin yapılmasından önce bütün süreçlerin beklenmesi için kullanılmaktadır.

1.2.5. Etkin Paralel Hesaplama için Yük Dengeleme

Yük dengelemenin amacı verilen bir işin işlemcilere dengeli bir şekilde dağıtılarak mümkün olan en kısa sürede işin bitirilmesidir. İş parçacıklarının ne kadar zaman alacağı önceden bilinemediği yada işlemci hızlarının eşit dağılıma sahip olmadığı durumlarda sistem performansını arttırmak için yük dengeleme yapılmalıdır.

Yük dengeleme statik ve dinamik olarak yapılabilmektedir.

1.2.5.1 Statik Yük Dengeleme

Süreçler başlamadan önce durağan olarak yapılan yük dengelemeye statik yük dengeleme denir. Genellikle optimizasyon teknikleri kullanılarak yapılan statik yük dengeleme bir tarifeleme problemi olarak ele alınabilir. Bu tür yük dengelemelerde genellikle problem parçacıklarının bağımlılık analizleri yapılarak işlemcilere eşit yük dağıtılmaya çalışılır.

Statik yük dengelemede kullanılabilecek bir çok algoritma mevcuttur. Raound-Robin, genetik algoritmalar ve rasgele algoritmalar gibi algoritmalar kullanılarak yük dengelemeleri yapılabilmektedir.

Statik ağ yapılarına sahip sistemler ağ yapılarına bağlı olarak statik yük dengeleme algoritmalarında başarısız olabilirler. Özellikle iş dağıtımları ve haberleşme kalıpları ağ bağlantı yapılarına uymadığında performans kaybı olmaktadır. Bu tür durumlarda iyi performans elde edebilmek için ağ bağlantı yapısını haberleşme kalıplarına uydurmak gerekmektedir. Bu ise yine bir optimizasyon problemi olarak ortaya çıkar.

Bütün şartların olumlu olması durumunda bile ağ gecikmeleri gibi gecikmeler tam olarak tahmin edilemeyeceğinden bazen yük dengelemede elde edilebilecek en iyi performansın altında kalınabilmektedir.

1.2.5.2. Dinamik Yük Dengeleme

Dinamik yük dengelemede iş parçacıkları işçi süreçlere süreçlerin koşması esnasında atanmaktadır. Yük dengeleme, işi biten işlemcilerin tekrar iş istemesi mantığına dayanır.

Bu tip yük dengelemede dikkat edilmesi gereken en önemli unsur iş parçacıklarının çok küçük yada çok az olmamasıdır. İş parçacıklarının çok küçük olması durumunda ağ bant genişliği ve gecikmeleri problem yaratırken, çok az olması durumunda ise iş parçacıklarının düzgün dağılmamaları riski ortaya çıkmaktadır. Özellikle son işler işlenirken, süreçlerden bazılarının işlerinin erken bitmesi durumunda yük dağılımı bozulacağından performans kaybı yaşanabilmektedir.

Dinamik yük dengeleme merkezi ve dağıtık olarak ikiye ayrılabilir.

1.2.5.2.1 Merkezi Dinamik Yük Dengeleme

Merkezi dinamik yük dengelemede bir yönetici süreç yapılması gereken iş parçacıklarını tutarken, işçi olarak çalışan süreçler yönetici süreçten iş isterler. İşi biten her işçi süreç, sonuçları yöneticiye iletir ve yeni iş ister. Bu tür çalışma mantığına genellikle işlemci çiftliği modeli denmektedir.

Bu yöntemin bir çok avantajı bulunmaktadır. İş işleme sırasında yeni süreçler sisteme katılabilmektedir. Çalışma esnasında yeni işçi eklemek yada çıkmak problem olmadığından anında performans ayarlaması yapılabilmektedir. İşlere öncelik atanması yeni iş kuyruklarının eklenmesiyle mümkün olabilmektedir.

Bu tür yük dengelemenin bir çok avantajı olmasına rağmen özellikle işleri dağıtma ve sonuçları toplama işlemlerinin uzun süreceği durumlarda performans kayıplarına sebep olabilmektedir.

1.2.5.2.2 Dağıtılmış Dinamik Yük Dengeleme

Yükün tek bir merkezi süreçte dengelenmesinin performans kayıplarına neden olduğu durumlarda dağıtık dinamik yük dengeleme kullanılabilmektedir. Bu algoritmada yükün dengelenmesi birden çok süreç tarafından yapılmaktadır. Yükler ya tamamen dağıtık olarak bir çok süreç tarafından ayarlanmakta yada yine merkezi bir süreç tarafından bir çok alt yönetici yönlendirilmektedir. Bu sayede yükün dağıtması işlemi de yük dengelemeye tabi tutulmuş olur.

1.3. Hacimsel Verilerin Görselleştirilmesi

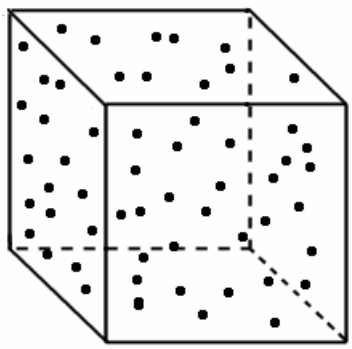
1.3.1. Hacimsel Veriler

Hacimsel veriler üç boyutlu uzayda noktalar ve bu noktaların yoğunluk değerlerinden oluşan verilerdir. Genellik CT, MR gibi tıbbi aletler ve bir çok yüzey tarayıcısı tarafından elde edilen veriler hacimsel yoğunluk olarak elde edilmektedir.

Bu verilerle çalışmanın en büyük zorluğu verinin belli bir geometrik şekle sahip olmamasıdır. Veriler üç boyutlu uzayda noktasal yoğunluklar olarak elde edildiği için alışılagelmiş geometrik primitiflerin kullanılması mümkün olmaz. Ayrıca verinin doğası gereği çoğu zaman hacimsel bir niteliğe sahip olması da, elde edilen verinin geometrik bir şekilmiş gibi düşünülmemesi gerektiği sonucunu ortaya çıkarmaktadır.

Elde edilen veriler sunuluş biçimine göre iki temel şekle ayrılabilir.

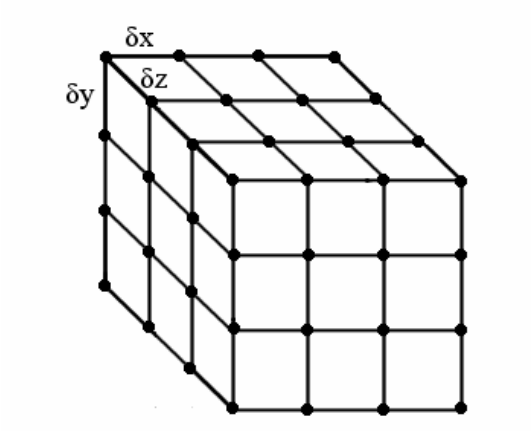
İlk olarak veri düzensiz noktalardan oluşan yoğunluklar kümesi şeklinde olabilir (Şekil 1.15). Bu tip veriler genellikle tıbbi görüntülemeler sonucu ortaya çıkabilmekle beraber, belli bir hareketli noktanın simülasyonu sonucu da ortaya çıkabilmektedir. Böyle verilerle çalışmak son derece zor olmaktadır. Veriler bazen belli bir bölgede toplanmış yada belli bir alanda kümelenmiş olabilir. Düzensiz hacimsel verileri düzgün bir şekilde görselleştirmek çok büyük çaba gerektirmektedir. Hiçbir düzenliliği bulunmayan noktalar kümesini görselleştirmek çok zor olduğundan bu veriler genellikle interpolasyon yardımıyla daha düzenli bir şekle indirgenirler. Bu indirgeme sonucu düzenli bir ağ yapısına kavuşan veriyi görselleştirmek daha kolay olmaktadır.



Şekil 1.15. Kübik bir hacim içerisinde dağınık veriler

İkinci olarak düzenli lineer yapılara sahip verilerden bahsedilebilir. En çok kullanılan düzenli veriler karesel veya dikdörtgensel düzenli prizma yapısına sahip verilerdir.

Dikdörtgensel veriler üç eksen arasındaki örnekleme mesafesi aynı olmayan fakat sabit olan verilerdir (Şekil 1.16). Bir diğer deyişle vokseller düzenli bir grid oluşturmakta fakat aralarındaki mesafeler eşit olmayabilmektedir. Eksenler yönündeki farkların $\partial x, \partial y, \partial z$ olduğu düşünülürse, bu tip gridler için $\partial x \neq \partial y \neq \partial z$ yazılabilir.



Şekil 1.16. 3x3x3'lük dikdörtgensel veri

Bu tip veriler genellikle uzayda bir dikdörtgen prizması şeklinde düzenli aralıklarla birbirinden ayrılan bir ağ yapısına sahiptir Bu verileri işlemek ve görselleştirmek oldukça kolaydır. Hemen hemen bütün görselleştirme algoritmaları bu tip verilerle çalışabilmektedir.

Özel bir dikdörtgensel durum ise karesel durumdur. Eksenler yönünde veri noktaları arasındaki mesafe aynı olan ve genellikle kübik olarak düzenlenen verilerdir. Bu tip veriler işlemesi ve görselleştirilmesi en kolay verilerdir. Bütün yönlerdeki aralıklar sabit olduğundan eksensel olarak kullanılan bütün oranlar da birbiriyle aynı olacaktır. Bu sayede bir çok oranlama işlemine gerek kalmamaktadır. Oysa veri dikdörtgensel olduğunda görselleştirmede kullanılan bazı interpolasyonlar için gereken süre artmaktadır.

Her iki tip veri de eksenler yönündeki örneklenmiş nokta sayıları bakımından farklılık gösterebilirler. Örneğin bir eksende 128 örnek bulunurken diğer eksende 256 örnek bulunabilir. Böyle veriler genellikle görselleştirme algoritması üzerinde fazla etki yapmamaktadırlar.

Bir diğer veri tipi ise küresel olarak elde edilmiş veridir. Küresel veya silindirik yapıya sahip bu tip veriler genellikle tıbbi görüntüleme sistemlerinde kullanılmaktadırlar. Bu verilerin görselleştirilmesinde farklı yöntemler kullanılabilir.

1.3.2. Hacimsel Verinin Görselleştirilmesinde Kullanılan Yöntemler

1.3.2.1. Marching Cubes

Bu yöntem ilk olarak Lorensen ve Cline [5] tarafından ortaya atılmış ve çeşitli şekillerde geliştirilmiştir ve başka bir çok algoritmaya temel olmuştur [6,7]. Yöntemin temelinde verileri geometrik primitiflere çevirme mantığı vardır.

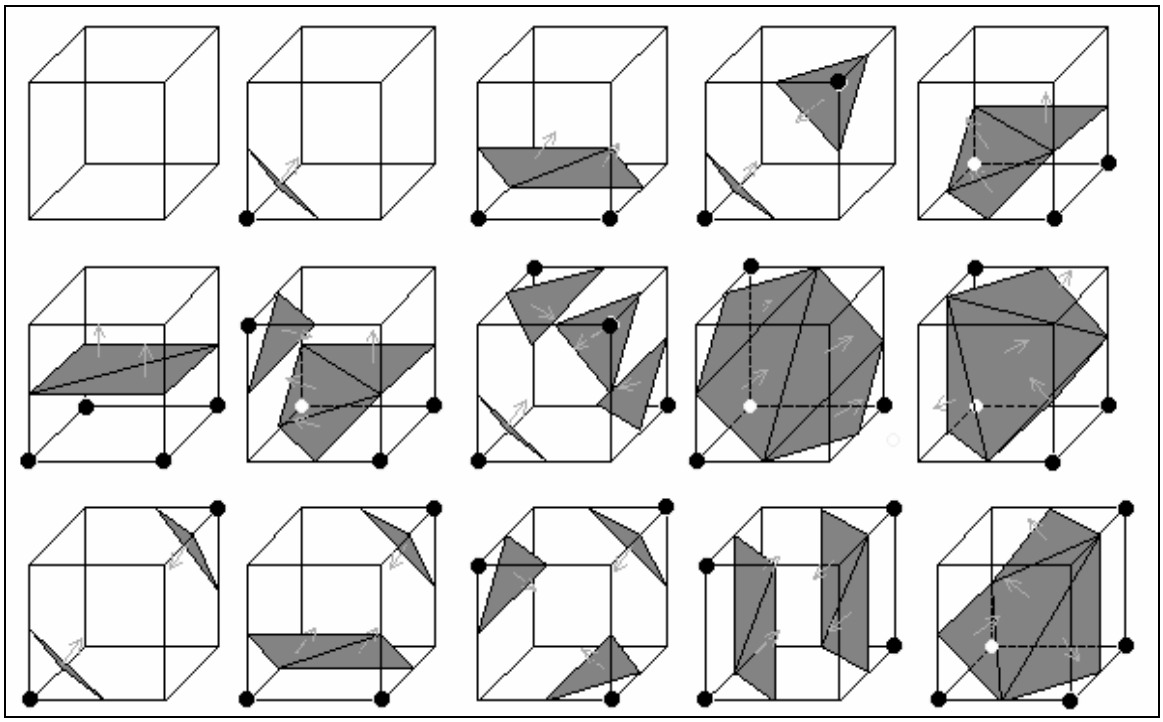
Orijinal yöntem ikili veriler üzerinde çalışmaktadır. Bir voksel ele alınarak vokselin köşelerinde bulunan ve cisme ait (1) veya ait olmayan (0) verilerine bakarak voksel içerisinde çeşitli geometrik primitiflerin olduğu varsayımında bulunulur. Daha sonra her vokselde bulunan bu üçgensel primitifler normal bir cisimmiş gibi grafik kartında görselleştirilebilir.

Bir vokselin 8 köşesi olduğundan ve her köşe var yada yok olarak işaretlendiğinden 256 farklı şekil çıkabilmektedir. Fakat bu şekillerden bir çoğu birbirinin simetriği olduğundan aslında 15 farklı sınıfta şekil yeterli olmaktadır (Şekil 1.17). Diğer şekiller bu şekiller döndürülerek elde edilebilmektedir.

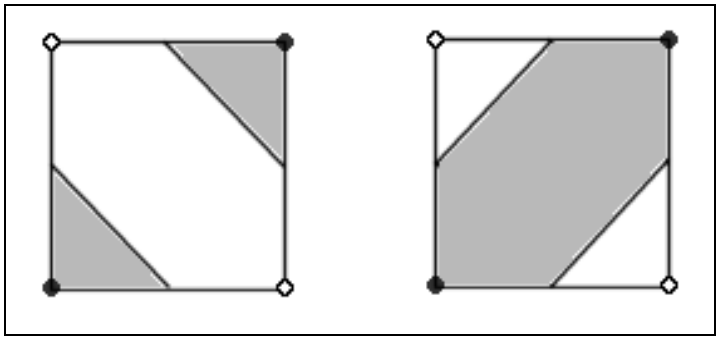
Bu algoritmada eldeki voksellere bağlı olarak bazı belirsiz durumlar oluşmaktadır. Bu nedenle bazen bitişik çıkması gereken bölgeler ayrı, ayrı çıkması gereken bölgelerde bitişik çıkabilmektedir. Problemi iki boyutlu düzlemde ele alırsak köşegenler üzerinde karşılıklı iki nokta cisme ait değilse, arada kalan bölgenin durumu belirsiz olur.(Şekil1.18) Bunu gidermek için çeşitli uygulamalara yönelik olarak değişik algoritmalar kullanılmaktadır.

Bu yöntem temel olarak ikili şekilde çalışsa da daha sonra bazı geliştirmeler yapılarak çoklu yüzeyli verileri işlemesi mümkün hale gelmiştir. Böylece iki veya daha çok farklı yoğunlukta cismin görselleşmesi mümkün olmuştur.

Yapılan diğer geliştirmeler ise genellikle algoritmanın hızlandırılması, oluşturulacak poligon sayısının azaltılması, üzerine olmuştur [9].



Şekil 1.17. Marchin-Cubes algoritmasında kullanılan küpler [5,8]



Şekil 1.18. Marching-Cubes algoritmasında oluşabilecek belirsiz durumların iki boyutlu düzlemde gösterimi

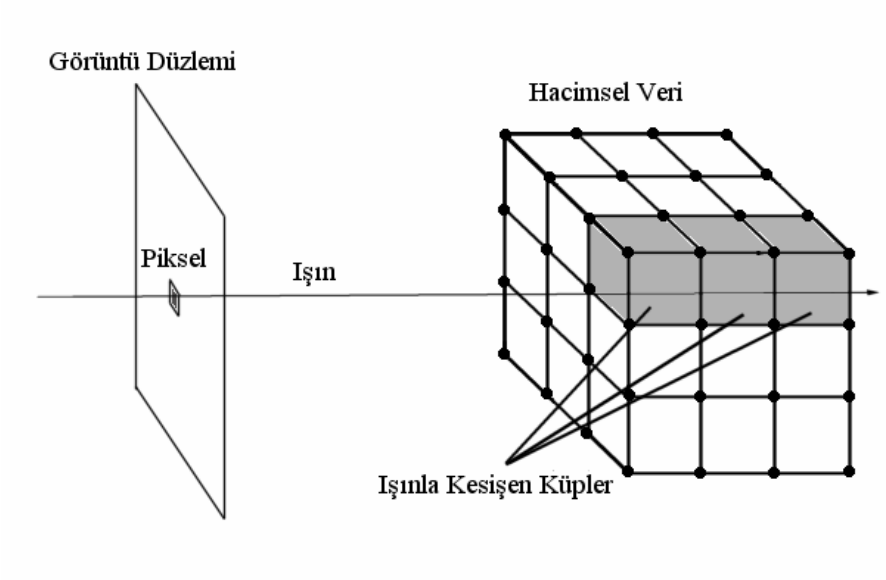
Bu algoritma her ne kadar iyi sonuçlar üretse de gaz benzeri kitleleri elde etmek pek mümkün olmamakta ve zamanla değişen verilerde avantajını kaybetmektedir. Kullanım alanı daha çok bir defa üretilip daha sonra çeşitli kamera açılarıyla incelenecek olan verilerin görselleştirilmesidir. Zamanla değişen verilerde her karede tekrar tekrar poligonize etme işlemi uygulanacağı için zaman kaybı oluşacaktır. Sonuç olarak işlemin en pahalı kısmı olan poligonize etme işlemi bir kez yapılıp daha sonra tekrar tekrar aynı model kullanılacağında oluşmayan problemler, veriler zamanla değiştiğinde ortaya çıkacaktır.

1.3.2.2. Işın Çizme Algoritmasıyla Eş-Yüzey Çıkarma

Algoritma projeksiyon düzleminden ışın atma mantığına dayanmaktadır. Resmin her pikseli için göz noktasından yaptığımız projeksiyona uygun olarak ışın çizilir. Daha sonra bu ışının hacmi tanımlayan prizmaya girip girmediği test edilir. Eğer girmiyorsa arka plan rengi ekrana basılır. Aksi halde ışın prizma içinde takip edilerek girdiği her vokselde istenilen yüzeye çarpıp çarpmadığı kontrol edilir (Şekil 1.19). Eğer ışın istenilen yoğunluk değerine sahip bir yüzey içeren vokselden geçerse noktanın vokselin neresinde kaldığı bulunur ve bu noktanın yüzey normali hesaplanır. Yüzey normali hesaplanan nokta ışıklandırılarak piksel rengi ayarlanır. Ekrandaki bütün pikseller için bu işlemi yapıldığında hacimsel veri içindeki belli bir yüzeyin görüntüsü elde edilmiş olur.

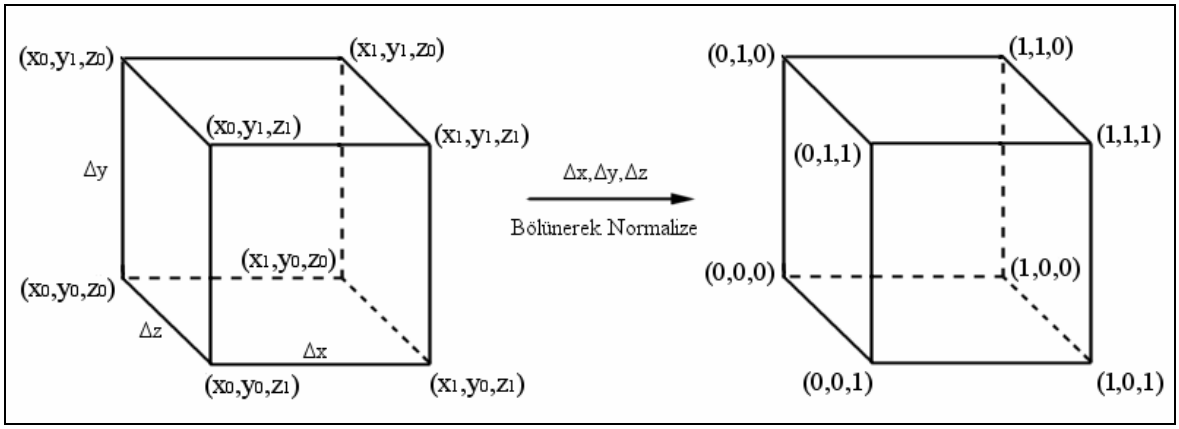
Bu algoritma çok bilinmekle ve uygulaması kolay olmakla birlikte çok hesaplama gerektirmektedir [10]. Işınların geçtiği küplerdeki değerler, polinomsal fonksiyonların çözülmesi ve trilineer interpolasyon yapılmasını gerektirmektedir.

Bu yöntemin matematiksel temeli ve analitik çözümleri bir çok kaynakta anlatılmış ve tartışılmıştır [11,12].



Şekil 1.19. Işın çizme algoritması.

Yöntemde ilk olarak vokselimiz 0-1 aralığına gelecek şekilde normalize edilir. Normalize işlemi için küpün en alt köşesi olan (x_0, y_0, z_0) noktası istenen $n = (n_x, n_y, n_z)$ noktasından çıkardıktan sonra $(\Delta x, \Delta y, \Delta z)$ farklarına bölünür (Şekil 1.20).



Şekil 1.20. Küpçüklerin normalize edilmesi.

Bu normalize edilmiş küpün içinde her hangi bir $n=(n_x,n_y,n_z)$ noktasının hangi yoğunluk değerine sahip olduğu çeşitli bağıntılarla hesaplanabilmekle beraber en çok kullanılan interpolasyon tekniği tri-lineer interpolasyondur.

Bu interpolasyon yöntemine göre n noktasının η(n) yoğunluk değeri aşağıdaki şekilde bulunabilmektedir.

$$\eta(n)=(1-n_x)\cdot(1-n_y)\cdot(1-n_z)\cdot n_{000}+(1-n_x)\cdot(1-n_y)\cdot(n_z)\cdot n_{001}+ (1-n_x)\cdot(n_y)\cdot(1-n_z)\cdot n_{010}+\ldots+(n_x)\cdot(n_y)\cdot(n_z)\cdot n_{111} \quad (2)$$

Bu bağıntı,

$$s_0(a)=1-a \quad (3)$$

$$s_1(a)=a \quad (4)$$

şeklinde iki fonksiyon tanımlanarak interpolasyon bağıntısı

$$\eta(n)=\sum_{i,j,k\in\{0,1\}} s_i(n_x)\cdot s_j(n_y)\cdot s_k(n_z)\cdot n_{ijk} \quad (5)$$

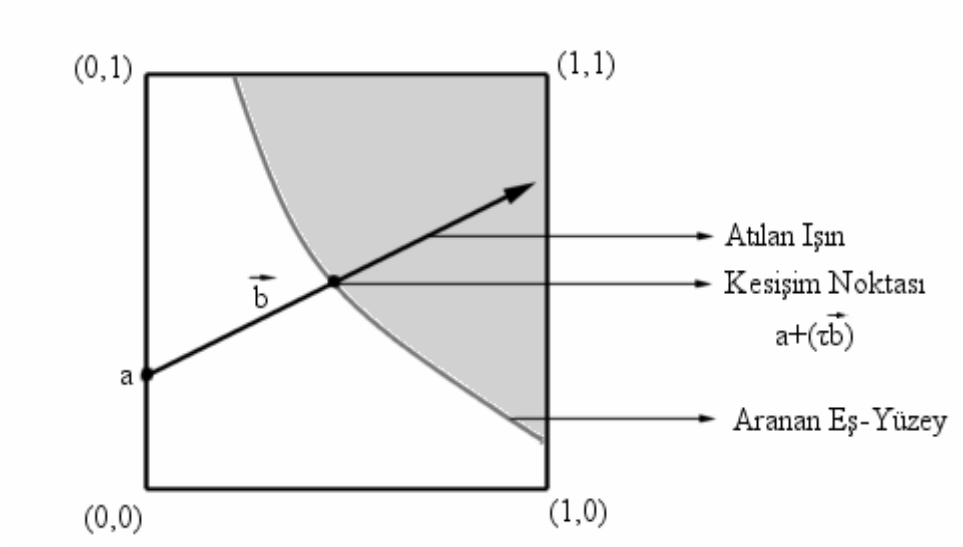
şeklinde yazılabilir. Bu bağıntı birim küp içinde bulunan bir $n=(n_x,n_y,n_z)$ noktasının yoğunluk değerini trilineer iterpolasyon yaparak vermektedir Bulunması istenen ise verilen

$$r = a + \tau \cdot \vec{b} \tag{6}$$

elde edilir. Bu bağıntı, a noktasından başlayan ve b vektörü yönünde ilerleyen τ parametresine bağlı bir bağıntıdır. Bu bağıntıyla trilineer interpolasyon bağıntısı bir araya getirilerek

$$\eta(r) = \sum_{i,j,k\in\{0,1\}} (s_i(a_0) + \tau \cdot s_i(b_0)) \cdot (s_j(a_1) + \tau \cdot s_j(b_1)) \cdot (s_k(a_2) + \tau \cdot s_k(b_2)) \cdot n_{ijk} \tag{7}$$

elde edilir. (7) bağıntısı sadece τ parametresine bağlı bir üçüncü dereceden polinom verir. Üçüncü dereceden bir polinom fonksiyonun analitik olarak çözümüne yönelik bir çok yöntem mevcuttur [13]. Bağıntı çözüldüğünde polinomun köklerinden küpün dışında bulunan noktaları işaret edenler atıldığında, istenilen η(n) değerini veren τ değeri veya değerleri elde edilmiş olur. Bağıntı (6) da yerine koyduğumuz τ değeri küp içindeki hangi noktanın istenilen yüzey yoğunluğuna sahip olduğu bilgisini verir (Şekil 1.21). İstenilen değerin bulunamaması durumunda ise ışını ilerleterek bir sonraki küpten devam etmek gerekecektir.



Şekil 1.21. Eş-yüzey bulmak için ışın çizme.

Eş-yüzey bulunduğunda ışıklandırma için yüzey normalinin belirlenmesi gerekecektir. Bunu yapmak için bir çok yol önerilmiştir [11,12]. Bu yollardan en çok

bilineni temel eğim (gradient) vektörü hesaplamaktır. Belli bir küpün içinde bulunan $n=(n_x,n_y,n_z)$ noktasındaki yüzey normali küpün köşelerinden yaralanılarak nokta yoğunluk farklarının x, y, z yönündeki eğimlerinden bulunur. Yüzey normali bulunurken trilineer interpolasyon bağıntısından yararlanılır.

$$\overrightarrow{N}=\vec{\Delta}\eta=\left(\frac{\partial\eta}{\partial x},\frac{\partial\eta}{\partial z},\frac{\partial\eta}{\partial z}\right)=\left(N_x,N_y,N_z\right) \tag{8}$$

$$N_x=\sum_{i,j,k\in\{0,1\}}(-1)^{i+1}\cdot s_j(\eta_y)\cdot s_k(\eta_z)\cdot\eta_{ijk} \tag{9}$$

$$N_y=\sum_{i,j,k\in\{0,1\}}(-1)^{j+1}\cdot s_i(\eta_x)\cdot s_k(\eta_z)\cdot\eta_{ijk} \tag{10}$$

$$N_z=\sum_{i,j,k\in\{0,1\}}(-1)^{k+1}\cdot s_i(\eta_x)\cdot s_j(\eta_y)\cdot\eta_{ijk} \tag{11}$$

Yüzey normali bulunduktan sonra ışıklandırma bağıntıları kullanılarak gerekli ışıklandırma yapılabilir.

1.3.2.3. Işın Atma (Ray Casting)

Hacimsel verilerin görselleştirilmesinde çok kullanılan bir diğer yöntem de ışın atma yöntemidir. Gaz benzeri hacimlerin görselleştirilmesi için uygun olduğu kadar eş-yüzey bulma algoritmalarıyla kullanımı da mümkündür.

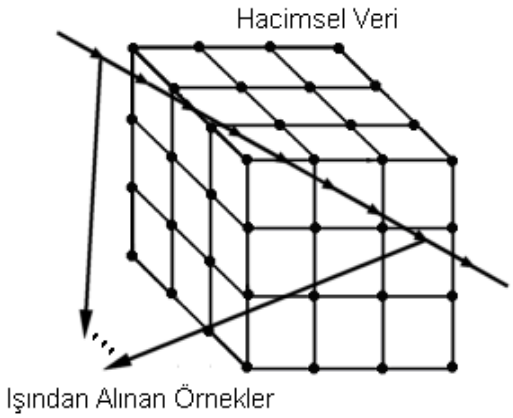
Bu yöntemde ilk olarak görüntü düzleminden hacimsel veriye bir ışın atılır. Daha sonra bu ışının takip ettiği yol üzerinde Δt aralıklarıyla örnekleme yapılır [11,14] (Şekil 1.22). Bulunan bu örnekler kullanıcının belirleyeceği transfer fonksiyonları yardımıyla renk ve geçirgenlik değerlerine dönüştürülür. Bu yönteme göre elde edilen renk değerlerini geçirgenlikleriyle (Alpha) beraber önden arkaya yada arkadan öne doğru ekleyerek ışının temsil ettiği pikseldeki renk elde edilir.

Bu algoritma birçok örnekleme gerektireceğinden yavaş olmaktadır. Algoritmayı hızlandırmak için birçok yöntem önerilmiştir [15-18].

İlk olarak ilgilenilmeyen, geçirgenlik değeri 0 olan, veri küpçüklerini tespit edilip ışının bu küplerden hızlı bir şekilde geçmesi ve örnek almaması sağlanabilir.

Bir diğer yapılabilecek geliştirme erken ışın sonlandırmasıdır [16]. Bu yöntemde

ışının takip ettiği yol üzerinde geçirgenliği nedeniyle arkasındaki piksellerin rengini geçirmeyecek bir örnek alındığında, artık ışının takip edilmesine gerek kalmamaktadır. Benzer şekilde ışın önden arkaya örnekleniyorsa geçirgenlik kat sayısı artık attaki örneklerin geçirilemeyeceği kadar büyükse yine örnekleme durdurularak ışının erken sonlandırılması sağlanabilir.



Şekil 1.22. Işın atma

Bu algoritmada arkadan öne doğru renklerin toplanması aşağıdaki şekilde olmaktadır.

$$C_{i+1}(P,R) = (\alpha(P,R) \times C(P,R)) + (1 - \alpha(P,R)) \times C_i(P,R) \quad (12)$$

Burada;

$C_i(P,R)$ pikselin i. örnekte sahip olduğu renk değeri,

$C_{i+1}(P,R)$ pikselin i. örnek eklendikten sonra alacağı renk,

$\alpha(P,R)$ i. örneğin geçirgenlik değeri (Alpha),

$C(P,R)$ i. örneğin renk değeridir.

Bağıntıdan da anlaşılabileceği gibi geçirgenlik değeri 1 olan bir örnek arkasındaki örneklerin görünmesini engelleyecektir. Bunu göz önünde bulundurarak geçirgenlik değeri 1 olan örnek görüldüğünde ışın sonlandırılıp geriye doğru renklerin eklenmesi ve piksel renginin elde edilmesi işlemi yapılabilir.

Önden arkaya ilerlendiğinde ise Bağıntı 12'nin bir benzeri kullanılabilir. Farklı olarak pikselin sadece rengi değil aynı zamanda geçirgenlik değeri de toplanır .

$$C_{i+1}(P,R) = C_i(P,R) + (1 - \alpha_i(P,R)) \times \alpha(P,R) \times C(P,R) \quad (14)$$

$$\alpha_{i+1}(P,R) = \alpha_i(P,R) + (1 - \alpha_i(P,R)) \times \alpha(P,R) \quad (15)$$

Burada,

$\alpha(P,R)$ i. örneğin geçirgenlik değeri (Alpha),

$C(P,R)$ i. örneğin renk değeri,

$C_i(P,R)$ pikselin i. örnek eklenirken sahip olduğu renk,

$\alpha_i(P,R)$ pikselin i. örnekte sahip olduğu geçirgenlik değeri,

$C_{i+1}(P,R)$ pikselin i. örnekleme sonunda alacağı renk

$\alpha_{i+1}(P,R)$ pikselin i. örnek sonunda sahip olacağı geçirgenlik değeridir.

$\alpha_i(P,R)$ değeri başlangıçta 0 olarak alınır ve ilerlenen her örnekle giderek 1 değerine yaklaşır. Bu değer 1'e çok yakın yada eşit bir değere geldiğinde ışın sonlandırılabilir. Ayrıca $C_i(P,R)$ renk değeri de başlangıçta 0 olarak alınmalıdır.

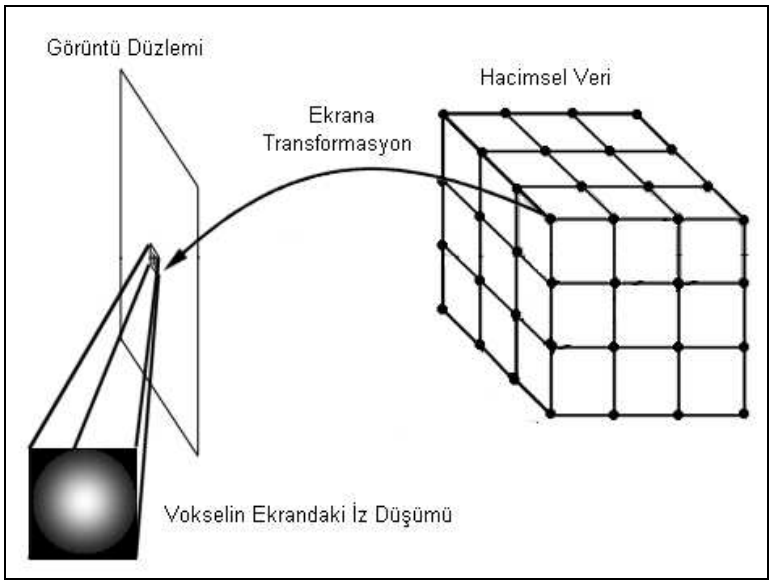
Bu ışın atma yöntemi farklı şekillerde uygulanabildiğinden hem eş yüzey çıkartma hem de gaz benzeri hacimlerin görselleştirilmesinde kullanılabilmektedir. Böylece melez bir görüntü elde edilebilir.

1.3.2.4. Splatting

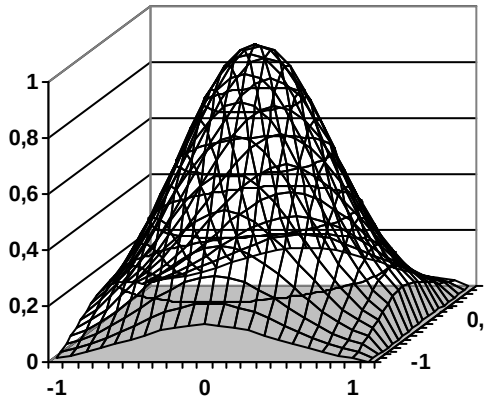
Bu teknik ışın atma tekniğinden farklı olarak cisim uzayının görüntü uzayına transfer edilmesi mantığına dayanmaktadır. Yöntem temel olarak bir vokselin ekranda nerede görülmesi gerektiğini bulur ve gerekli renklendirme işlemini yapar [19] (Şekil 1.23).

Teknik ilk olarak vokseli ekranda belir bir alana iz düşürür. Daha sonra bu alanı bir transfer fonksiyonuyla çarpar. Genellikle transfer fonksiyonu olarak gauss ağırlıklandırma fonksiyonu kullanılır (Şekil 1.24). Böylece her vokselin ekran üzerinde bulunan piksellerdeki ağırlıkları bulunur.

Voksellerin ağırlıklandırılmasından sonra görüntü düzlemine en paralel olan düzlemden önden arkaya veya arkadan öne doğru pikseller toplanır. Bu görselleştirme işlemi dilim dilim yapıldığından, ilk önce bütün bir dilim geçici bir görüntü düzlemine yazılarak renkler birleştirilebilir. Daha sonra bu geçici düzlemden ana resim düzlemine önden-arkaya yada arkadan-öne birleştirme işlemi yapılarak görüntü elde edilir.



Şekil 1.23. Splatting



Şekil 1.24. 2 boyutlu Gauss fonksiyonu (standart sapma = 0.5, değerler 1' e normalize edilmiştir)

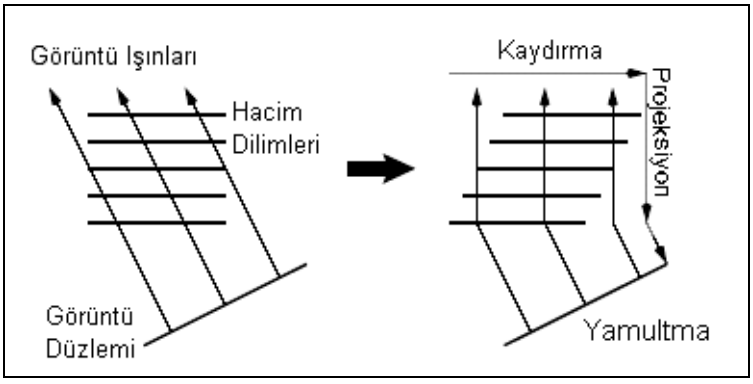
Bu yöntemde genellikle ekrana transfer edilecek voksellerin ekranda ne kadarlık bir yer kapladığının belirlenmesi gerekmektedir. Bir voksel ekranda az yer kaplarsa resim kesik kesik çıkacaktır, aksi durumda, olması gerekenden daha büyük olduğunda, resim bulanıklaşmaya başlayacaktır [19]. Bu nedenle vokselin ekranda ne kadarlık bir alan kaplayacağını belirleyen transfer çekirdeğinin büyüklüğü ekran pikselinin uzaydaki büyüklüğü kadar veya biraz daha fazla olarak seçilmelidir. Bu sayede hem ekranda kesikli görüntüler elde edilmesinin önüne geçilir hem de resimde oluşabilecek istenmeyen yumuşamalara izin verilmemiş olur.

Splatting tekniği üzerinde çok çalışılmış bir tekniktir. Teknik zaman içinde iyileştirilmiş ve çeşitli şekillerde hızlandırılmıştır [20-25]. Özellikle video kartlarının gelişmesi ve 3 boyutlu yeteneklerin artması sayesinde splatting tekniğine benzer teknikler grafik işlemcisinden ve belleğinden daha çok yararlanma imkanı bulmuşlardır [26]. 3 boyutlu dokular sayesinde bir çok işlem çok daha kolay yapılabilir olmuş ve büyük miktarda hız artışı sağlanmıştır. Ayrıca ekrana en paralel olan düzlemden yapılan birleştirme işlemi yerine, hacimsel verinden görüntü düzlemine paralel kesitler olacak şekilde yeni kesitler oluşturularak birleştirme işlemi yapma, oluşabilecek bir çok görüntü problemini ortadan kaldırmaktadır [27].

1.3.2.5. Kaydırma-Yamultma (Shear-Warp)

Bu teknik hem görüntü hem de cisim uzayında gerçekleştirilmektedir [28]. Algoritmik açıdan melez bir teknik olduğu söylenebilir. Bu teknikte görüntü düzleminin ve projeksiyonun özelliklerine göre eldeki hacimsel veri üzerinde ilk olarak üç boyutlu kaydırma daha sonra iki boyutlu yamultma işlemleri yapılır. Bu kaydırma ve yamultma işlemleri sonunda veriler doğru şekilde görselleştirilir (Şekil 1.25).

Bu teknik genellikle en hızlı teknik olmasına rağmen detaylı şekiller içeren verilerde görüntü bozulmalarına yol açabilmektedir. Ayrıca görüntü kalitesi açısından diğer tekniklere göre biraz zayıf kalmaktadır [10].



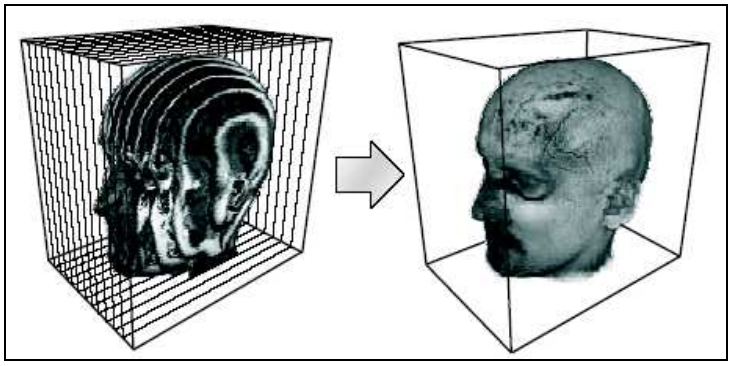
Şekil 1.25. Kaydırma ve Yamultma [28]

1.3.2.6. 2 ve 3 Boyutlu Dokularla Hacimsel Görselleştirme

Bu yöntemin en büyük avantajı bilgisayarın grafik işleri için özelleşmiş olan grafik

kartını kullanmasıdır. Bu sayede çok büyük hız artışları sağlanabilir. Özellikle küçük ve orta boy hacimsel verilerin görselleştirilmesi tek bilgisayarda gerçek zamanlı görüntü (30 kare/saniye) verebilmektedir. Fakat bu yöntemle oluşturulan görüntüler genellikle diğer yöntemlerle oluşturulan görüntülerden daha kötü görünmektedir.[10]

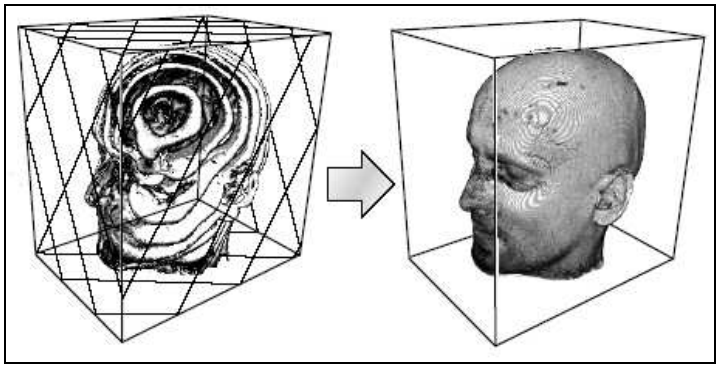
Temel olarak 2 çeşit yöntem kullanılmaktadır. Birinci yöntemde veriler dilim dilim grafik kartına doku olarak verilirler. Daha sonra bu dilimler arka arkaya gelen dikdörtgenler şeklinde ekrana yazdırılırlar (Şekil 1.26).



Şekil 1.26. İki boyutlu dokular kullanarak görselleştirme [29]

Bu yöntem daha çok sadece 2 boyutlu dokuları tutabilen sistemlerde kullanılmaktadır. 3 boyutlu dokuları destekleyen grafik kartlarında ise daha gelişmiş bir teknik uygulanabilmektedir.

Üç boyutlu dokular sayesinde dilimleri arka arkaya gelen ve ekrana en paralel olan dilimler şeklinde yazmak yerine, ekrana paralel düşecek şekilde doku içerisinden dilimler seçip bu dilimler arka arkaya yazdırılır (Şekil 1.27). Bu sayede daha kaliteli bir görüntü elde etmek mümkün olmaktadır.



Şekil 1.27. Üç boyutlu dokular kullanarak görselleştirme [29]

1.3.2.7. En Büyük Parlaklığın Haritalanması (Maximum Intensity Mapping)

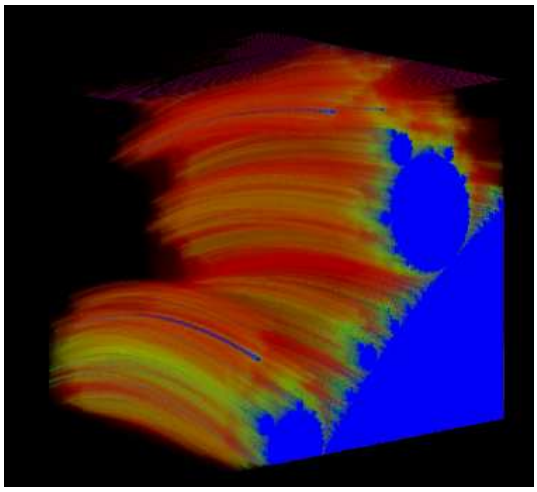
En büyük parlaklığın haritalanması yöntemi genellikle X-ışını taramalarında kullanılan bir yöntemdir. Bir çok farklı şekilde gerçeklenebilmesine rağmen en doğal yöntem ekrandan gönderilen bir ışının izlenerek geçtiği yol üzerindeki en parlak değerin ekrandaki piksele atanmasıdır. Bu sayede o ışının izlediği yol üzerindeki en parlak değer görüntülenmiş olur.

Bu yöntem doğası gereği daha çok tıbbi uygulamalarda kullanılmakla birlikte, hacimdeki bazı dağılım özelliklerinin görülmesini gerektiren uygulamalarda da kullanılabilir.

1.3.3 Phong Aydınlatma Modeli

Özellikle yüzey içeren hacimsel verilerde aydınlatma yapılması gerekmektedir. Aydınlatma yapılmadan, sadece renk ve geçirgenlik değerleri göz önüne alınarak yapılan görselleştirmeler genellikle istenilen sonucu vermemektedir (Şekil 1.28).

Aydınlatma yapmak için bir çok model kullanılmakla birlikte en çok kullanılan aydınlatma modeli Phong aydınlatma modelidir.



Şekil 1.28. Mandelbrot kümesinin bir parçası

Phong aydınlatma modeline göre [30] bir cismin rengini belirlerken üç bileşen kullanılmaktadır;

- Ambient,
- Diffuse,
- Specular.

Aydınlatma genellikle ışık kaynağından gelen ışınlarla yapıldığından, cismin doğrudan ışık almayan bölgeleri siyah görünecektir. Gerçek hayatta ise cismin doğrudan ışık almayan kısımları çevreden yansıyan ışığın etkisiyle aydınlanmaktadır. Bu etkiyi yaratabilmek için Phong aydınlatma modelinde bütün cisim noktalarına sabit bir renk değeri verilmektedir. Bu sabit bileşen ambient bileşen olarak bilinmektedir. Belli bir sabit katsayı cismin rengiyle çarpılarak bütün cisim piksellerine eklenir.

$$amb = k_a \times I_a \tag{16}$$

Phong aydınlatma modelinin en önemli kısmı diffuse aydınlatma bileşenidir. Bu bileşen Lambert yasasını temel alarak cismin ana aydınlatma bileşenini vermektedir. Lambert yasasına göre ışık cisme ne kadar dik gelirse cisim o kadar parlak gözükmektedir. Aydınlatma modeline diffuse bileşeni eklemek için ilk olarak ışığın geliş vektörü ve yüzeyin yüzey normali bilinmelidir. Daha sonra bu bileşenler arasındaki açı hesaplanıp, ışığın rengi ve diffuse katsayısı ile çarpılarak diffuse bileşen bulunmaktadır.

$$dif = \overrightarrow{N} \otimes \overrightarrow{L}^{T} \times k_d \times I_d \tag{17}$$

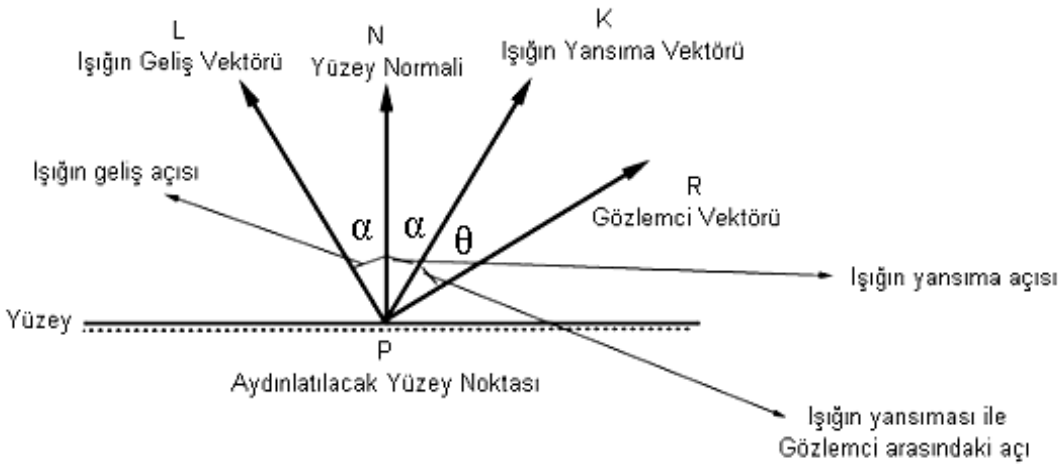
N yüzey normali vektörü ile L ışık vektörünün transpozu çarpıldığında iki vektör arasındaki açının kosinüsü elde edilmektedir. Her iki vektörün de bire normalize edilmesi gerektiği unutulmamalıdır. Aksi halde bu iki vektörün çarpımından sonra çıkan sonucu her iki vektörün uzunluğunun çarpımına bölmek gerekecektir.

Diffuse aydınlatma ana aydınlatma bileşeni olmasına rağmen gerçek hayattaki görüntüleri tek başına verememektedir. Bunun en önemli nedeni gerçek hayatta cisimlerden yansıya ışığın bakış açısına göre parlamasıdır. Bu özellik cisimlerin aynasal özelliği olarak bilinmektedir. Bu tür bir aynasal etkiyi elde etmek için specular aydınlatma kullanılmaktadır. Bu aydınlatma bileşeni ışığın yansıma vektörü ile gözlemcinin bakış vektörü arasındaki açının kosinüsünü kullanmaktadır.

$$spec = (\overrightarrow{K} \otimes \overrightarrow{R}^{T})^{n} \times k_{s} \times I_{s} \quad (18)$$

Diffuse bileşenden farklı olarak yansıma vektörü ile gözlemci vektörünün arasındaki açının kosinüsünün belli bir üssü alınmaktadır. Bu sayede aynasal parlamanın cismin üzerine ne kadar yayılacağı kontrol edilebilmektedir. n değeri küçük seçilirse parlama yayvan olacaktır, büyük seçildiği durumlarda ise çok küçük bir alanda parlama oluşacaktır.

Phong Aydınlatma modelinde kullanılan vektörler Şekil 1.29'da verilmiştir.



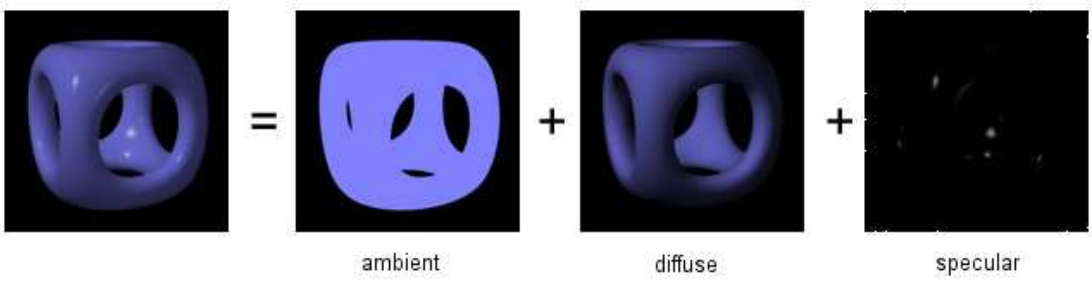
Şekil 1.29. Phong aydınlatma modelinde kullanılan vektörler.

Bütün bileşenler hesaplandıktan sonra hepsi birlikte toplanılarak piksel rengi elde edilir.

$$P = amb + dif + spec \quad (19)$$

Bir çok ışık kaynağı söz konusu olduğunda ambient bileşenler toplanmaz, fakat diffuse ve specular bileşenler toplanarak sonuç rengi elde edilir.

Phong aydınlatma modeli bu üç bileşen sayesinde az miktarda hesap yaparak gerçekçi görüntüler elde edilmesini sağlamaktadır (Şekil 1.30).



Şekil 1.30. Phong aydınlatmada etkili olan bileşenler [31]

1.4. Quaternionda Mandelbrot Kümesi

1.4.1 Mandelbrot Kümesi

Gaston Julia' nın 20. yüzyılın başlarında yaptığı rasyonel polinomal fonksiyonların iterasyonlarıyla ilgili çalışmalar yüzyılın ikinci yarısında Benoit Mandelbrot tarafından yeniden ele alınıp bilgisayar grafiklerinde ve matematiğin yeni bir dalı olan fraktal geometride yeni bir çığır açmış ; gündeme oturmasını ve insanların dikkatini çekmesini sağlamıştır.

Mandelbrot ilginç resimler elde etmek ve işin estetik tarafıyla birlikte matematiksel kaosun kullanışsız olduğunu düşünen klasik yaklaşıma karşı çıkarak kaosun hemen her yerde ortaya çıkabilen bir olgu olduğunu anlatmak için fraktalları kullanmıştır.

Bu fraktallardan en çok bilinenlerinden biri olan Mandelbrot kümesi :

$$f(z) = z^2 + c, \tag{20}$$

bağıntısıyla ifade edilen bir kümedir. Bağıntıyla oynanarak daha değişik şekliler de elde etmek mümkündür. İki boyutlu karmaşık sayılarda Mandelbrot kümesi iterasyonlar yapılarak kolaylıkla hesaplanabilmektedir. Küme başlangıç olarak z = 0 değeriyle c noktası için hesaplanır ve çıkan fonksiyon değeri bir sonraki fonksiyon hesabında z parametresi olarak kullanılır.

$$z = 0,\ f(z) = z^2 + c,\ \ f(f(z)) = f(z)^2 + c\ ,\ \ldots \tag{21}$$

Bu iterasyonlar sonucunda nokta ya sonsuza gidecek yada gitmeyerek garip şekilde 0 etrafında gezinecektir. Fakat bir noktanın kaç iterasyon sonucu sonsuza gidip gitmeyeceği net bir şekilde bilinemez. Fonksiyon değerinin sonsuza gidip gitmeyeceğini hesaplamanın tek yolu iterasyonları hesaplamaktır. Sonsuza kadar bu işlemin devam ettirilmesi mümkün değildir. Fonksiyon değerinin sıfırdan belli bir uzaklıktan daha uzak kalması durumunda sonsuza gideceği kesin olarak söylenebilir. Bu nedenle belli bir iterasyon sayısına kadar hesaplamaya devam edilip , bu iterasyonlar sonunda belli bir değeri aşmayan noktaların sonsuza gitmeyecekleri varsayımında bulunulabilir. Yapılan iterasyon ne kadar fazlaysa kümenin sınırları o kadar kesin olarak belirlenecek fakat hiçbir zaman bilinemeyecektir.

Bir boyutlu uzayda bu fonksiyon çok açık bir şekilde ilginç bir sonuçlar vermemektedir. Fonksiyonu asıl ilginç kılan çok boyutlu karmaşık sayılar kullanıldığında ortaya çıkan ilginç sınırı belirsiz şekildir. İki boyutlu düzlemde [-2,-1.5] – [1,1.5] karesel alanında kalan bölgedeki noktalar incelendiğinde en ünlü fraktallardan biriyle karşılaşılmış olur. Bu fraktalı görselleştirmek için genellikle sonsuza giden noktaların kaçıncı iterasyonda sonsuza gittikleri bilgisi kullanılır. Bu bilgi yardımıyla belli iterasyon sayısında sonsuza gideceği hesaplanan noktalar belli bir renge sahip olurlar.

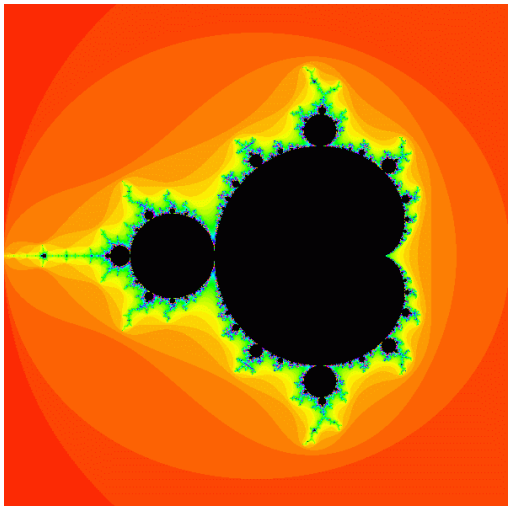
Bir diğer ilginç nokta ise sınır bölgelerde şeklin detayının kaybolmamasıdır. İterasyon sayısı arttırıldığı sürece şekil ne kadar büyütülürse büyütülsün detay kaybolmayacaktır.

Mandelbrot kümesi hesaplanırken birkaç noktaya dikkat edilmesi hesaplamalarda hız kazanılması ve hesaplamanın doğru yapılması açısından önemlidir. Dikkat edilmesi gereken iki nokta vardır :

1. İterasyon sonucu ortaya çıkan nokta orijinden 2 ‘ den daha uzakta kalıyorsa bu nokta sonsuza gidecek demektir. Bu aşamadan sonra hesaplama yapılmasına gerek yoktur. Nokta iterasyon numarası kadar iterasyonda sonsuza gitmiş olarak işaretlenebilir.
2. Karmaşık sayının orijinden uzaklığı (büyüklüğü, mutlak değeri) hesaplanırken karekök alma işlemi yapmak gerekir. Ancak bunu yapmak yerine karekök almadan sonucun 4’ten büyük olup olmadığına da bakılabilir. Bu sayede hesaplama zamanından tasarruf edilmiş olur.

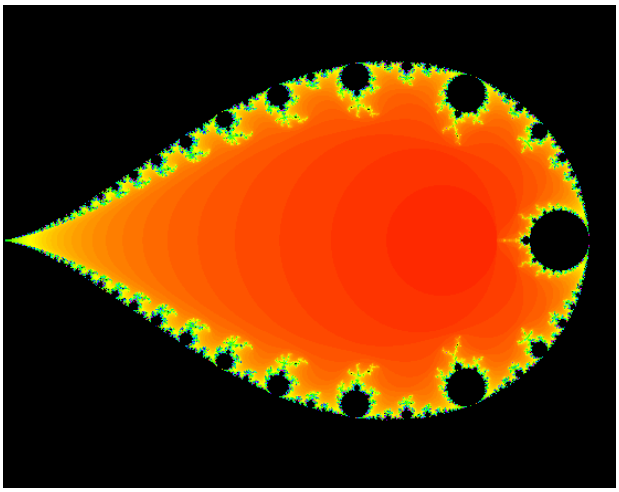
Ayrıca hız kazanmak için çok sık kullanılan bir teknik de hesaplamayı tam sayılarla ölçeklendirerek yapmaktır. Ek olarak günümüz işlemcilerinin bir çoğunda buluna vektörel hesaplama komutlarını kullanarak da hız kazancı sağlanması mümkündür.

Fonksiyon basit olmasına rağmen ortaya karmaşık sonuçlar çıkmaktadır (Şekil 1.31).



Şekil 1.31. $f(z) = z^2 + c$

Ayrıca küçük parametre değişiklikleri yapılarak çok daha farklı ve değişik görünüme sahip olan fraktal şekiller de elde etmek mümkün olmaktadır (Şekil 1.32).



Şekil 1.32. $f(z) = z^2 - 1/c$

1.4.2 Quaternionlar (4 Boyutlu Karmaşık Sayılar)

Temel olarak quaternionlar dört boyutlu karmaşık sayılardır. İki boyutlu karmaşık sayılara benzer şekilde tanımlanırlar.

$$Q = q_0 + q_1 \cdot i + q_2 \cdot j + q_3 \cdot k \qquad (22)$$

Dört adet bileşene sahip olduklarından her bileşenin çarpımı farklı etkilere sahip olmaktadır. Toplama ve çıkarma işlemi aynen karmaşık sayılarda olduğu gibi bileşenlerin çıkarılması yada toplanmasıyla yapılmaktadır. Çarpma yaparken ise her bileşen diğer sayıdaki bütün bileşenlerle çarpılmaktadır. Bu çapma işlemi sırasında bazı kurallara dikkat etmek gerekir. Çarpanların oluşturacakları kombinasyonlara göre çıkacak sonuçlar Tablo 1.'de verilmiştir.

Tablo 1. Dört boyutlu karmaşık sayı bileşenlerinin çarpım sonuçları [32]

	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

Quaternionlar genellikle bilgisayar grafiklerinde üç boyutlu homojen koordinat sistemindeki noktalar için kullanılırlar. Bu sayede ölçekleme ve öteleme işlemlerini matris çarpımlarıyla yapmak mümkün hale gelir.

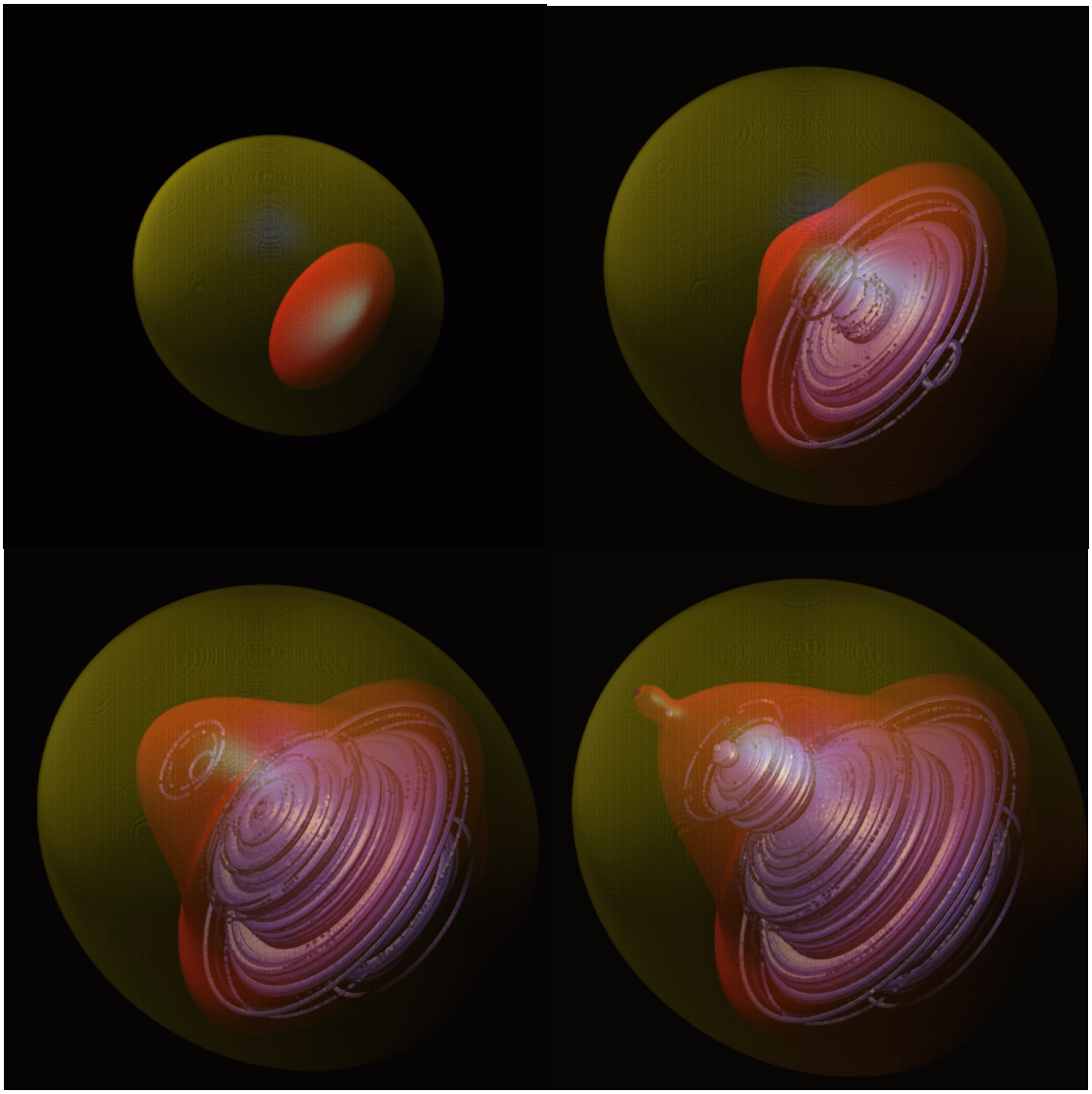
Bir diğer kullanım alanı ise zamanda değişen 3 boyutlu cisimleri tanımlamaktır. Dört bileşenden herhangi biri zaman diğer üç bileşen ise mekansal boyut olarak alınır. Her ne kadar rasgele seçim mümkün olsa da genellikle k katsayısıyla temsil edilen sonuncu terim zaman boyutu olarak ele alınmaktadır. Bu sayede art arda sıralanan dört boyutlu noktaların aynı zaman dilimine ait oldukları sadece sonuncu parametreye bakılarak anlaşılabilir. Bu zorunlu olmamakla birlikte insan alışkanlıkları nedeniyle daha az hataya neden olacağından tercih edilmektedir.

1.4.3 Quaterniyonda Mandelbrot Kümesinin Hesaplanması

Zamanla değişen 3 boyutlu bir görüntü elde edilmek isteniyorsa iki boyutlu Mandelbrot fraktalı dört boyutlu qutternionlarla hesaplanabilir. Dört boyutlu karmaşık sayılarda bir bileşen sabit olarak zamanı temsil edecek şekilde alınır. Diğer bileşenler de üç

boyutlu bir Mandelbrot kümesi verecek şekilde ayarlanabilir.

Bu tür bir işlem yapıldığında zaman k parametresi olarak alınabilir, 0 noktası en ilginç nokta olmaktadır. ±1,2 değerine doğru geri yada ileri ilerlendiğinde şekil giderek kaybolmakta ve sonunda yok olmaktadır. En ilginç zamanla değişen veriler gerçel parametrenin [-2.0,0.9], i parametresinin [-1.28,1.28], j parametresinin [-1.28,1.28] ve k parametresinin [-1.28,1.28] olduğu aralıktır. Bu aralıkta yapılacak incelemeler görsel olarak en ilginç ve kayda değer sonuçları vermektedir (Şekil 1.33).



Şekil 1.33. Mandelbrot kümesinin zamanla değişimi.

Bu kümeyi hesaplamak için takip edilmesi gereken algoritma aşağıda yalancı kodda verilmiştir.

```
DortBoyutluMandel()
1:      FLOAT x , y,z,t, q1, q2,q3,q4, gecici;
2:      INT IterasyonNo;
3:      For( t = AltLimitT ; t < ÜstLimitT ; t = t + Zaman_Aralık )
4:      For( x = AltLimitX ; x < ÜstLimitX ; x = x + X_Aralık )
5:      For( y = AltLimitY ; y < ÜstLimitY ; y = y + Y_Aralık )
6:      For( z = AltLimitZ ; z < ÜstLimitZ ; z = z + Z_Aralık )
7:      {
8:        q1=0; q2=0;  q3=0;q4=0;q5=0;
9:       For(IterasyonNo=0; IterasyonNo < MaksimumIterasyon; IterasyonNo++ )
10:      {
11:            gecici=2*q1;
12:            q1=q1*q1-q2*q2-q3*q3-q4*q4+x;
13:            q2=gecici*q2+y;
14:            q3=gecici*q3+z;
15:            q4=gecici*q4+t;
16:             If (q1*q1+q2*q2+q3*q3+q4+q4 > 4.0 )
17:            {
18:                  Plot( x , y , z , t , IterasyonNo );
19:                  Break;
20:            }
21:       }
22:       Plot( x , y , z , t , IterasyonNo );
23:     }
```

2. YAPILAN ÇALIŞMALAR

2.1. Bilgisayar Kümesi Oluşturmak İçin Kullanılan Donanımlar ve Yazılımlar

Çalışmalarda her biri ; Pentium IV 3.2 GHz işlemeci, 512 MB RAM, 128 MB hafızalı ATI Radeon RX300 ekran kartı ve 1Gbps Ethernet kartına sahip dokuz adet sistem kullanılmıştır. Donanımsal olarak birbirlerinin aynı olan bu sistemler işlem gücü bakımından eşittirler. Bu durum, sistemlerin hız farklılıklarının, yük dengeleme sırasında asimetri oluşumunun nedeni olmayacakları anlamına gelmektedir.

Sistemler donanımsal olarak kişisel bilgisayarlara benzediklerinden fiyatları normal bir ev-ofis bilgisayarıyla aynıdır. Bu nedenle bilgisayar kümeleri oluştururken bu tür bilgisayarların kullanımı oldukça ucuz olmaktadır.

Ağ alt yapısı olarak 24 portlu 1 GigaBit switch kullanılmıştır. Bütün bilgisayarların bağlanması için uygun port sayısına sahip olması nedeniyle araya giren ek switchlerden kaynaklanan gecikmeler olmaz. Çok sayıda bilgisayarın birbirine bağlanması gerektiğinde bu tür switchler yeterince port içermediklerinden ek switchler kullanmak gerekir. Kullanılan her ek switch diğer switchlere bağlanırken belli bir süre gecikme olmasına neden olur. Ayrıca switchler arasındaki bağlantı sayısının çeşitliliği ve bağlantıların bant genişliği de önem kazanacaktır. Bu nedenle büyük yapılar kurulurken ihtiyaçlar iyi belirlenmeli, bant genişliği ve gecikme faktörü göz önünde bulundurulmalıdır. Aksi halde hem işletim maliyeti artacak hem de bilgisayarların işlem gücü yeterince verimli kullanılamayacaktır

Kullanılan bilgisayarlarda işletim sistemi olarak Windows XP Professional kullanılmaktadır. Ayrıca mesaj geçme arayüzü olarak MPI 1.2 standartlarında [33] yazılmış olan MPICH NT 1.2.5 yazılımı kullanılmıştır. Bu MPI implementasyonu hem açık kaynak kodlu olması hem de kolay kurulum ve kullanımı nedeniyle tercih edilmiştir. Bu yazılım standart kurulum paketiyle sorunsuz olarak kurulabilmektedir. Kurulum paketi; gerekli yazılım geliştirme kütüphaneleri, örnek programları ve Windows için MPI servisini otomatik olarak kurmaktadır. MPICH sisteme kurulduğundan “mpd” servis programıyla bir Windows servisi olarak hizmet vermektedir.

MPICH implementasyonunun süreç çalıştırma mekanizması her bilgisayarda verilen yolda çalıştırılabilir bir program dosyasının var olmasını gerektirmektedir. Ayrıca

koşturulan programın kullanacağı bütün dosyaların da mevcut süreçleri koşan bilgisayarlar tarafından erişilebilir olması gerekmektedir.

Her defasında bütün bilgisayarlara tekrar tekrar program dosyalarını yüklemek zor olacağından program dosyalarının konulabileceği ortak bir ağ klasörü oluşturulmuştur. Bu ağ klasörü sayesinde bütün bilgisayarlar ağ üzerinden bağlantı kurup mevcut dosyalara erişebilmektedirler. Ayrıca bu ağ dizinin bütün bilgisayarlara belli bir sabit disk olarak bağlanması kullanım rahatlığı açısından önemlidir. Ancak büyük ağlar için her bilgisayarda teker teker bu işi yapmak zor olacağından ağ diziniyle yetinilebilir.

Windows NT temelinde bir sistem olan Windows XP çok kullanıcılı bir sistem olduğundan MPI çalışabilmek için bir kullanıcı hesabına gereksinim duymaktadır. Bu nedenle her bilgisayarda MPI programlarını çalıştıracak ve tümünde aynı adı alacak şekilde bir kullanıcı hesabı açılmıştır. Bu hesabın şifreleri aynı olup MPICH tarafından her program çalıştırıldığında istenmektedir. Fakat, her defasında kullanıcı ismi ve şifre girilmek istenmiyorsa, MPICH' ın sunduğu kayıt programı sayesinde Windows kayıt defterine bu bilgiler yazılarak programın otomatik kullanıcı ismi ve şifre girmesi sağlanabilmektedir.

MPICH çalışacağı bilgisayarları seçmek için ya kendine ait görsel işletim programını, yada komut satırından parametre alan "mpirun" programını kullanmaktadır. En sorunsuz kullanım grafiksel arayüz sağlayan programla olmasına rağmen bu çalışmada, programın çalışmasında daha çok söz sahibi olunacak, komut satırı programının kullanılması tercih edilmiştir.

Kullanılan MPICH versiyonu yapılandırma dosyasıyla çalışmaya müsaade etmektedir. Bu yapılandırma dosyasına MPI ile ilgili hemen hemen tüm parametreler yazılarak "mpirun" programının bu yapılandırma dosyasını işletmesi sağlanabilmektedir. Bu dosyanın içinde genellikle çalıştırılacak programın yolu, argüman listesi, çevresel değişkenler ve programın hangi makinede kaç süreç koşturacağına ilişkin bilgiler liste halinde bulunmaktadır. Bu dosyanın bir örneği aşağıda verilmiştir:

```
1:    exe \\bilgisayar1\paylasim\program.exe
2:    args -d 3 3
3:    env MPICH_SINGLETHREAD=1
4:    hosts
5:    bilgisayar1 2
```

6: bilgisayar2 1

7: bilgisayar3 2

8: bilgisayar4 3

MPICH, koşulacak süreçlerin numarasını yapılandırma dosyasındaki sıraya göre dağıtmaktadır. Bu sayede kaçıncı sürecin hangi bilgisayarda koşacağı kolaylıkla denetlenebilmektedir. Ancak, yapılandırma dosyası kullanılmadığında, MPICH süreç numaralarını Round-Robin algoritmasına göre dağıttığından, bilgisayar sayısıyla süreç sayısının uyumlu olmasına dikkat edilmelidir.

2.2. Paralel Görselleştirme Uygulamasının Geliştirilmesi

Uygulamadaki temel amaç zamanla değişen üç boyutlu Mandelbrot kümesini dağıtık olarak hesaplamak ve görselleştirmektir. Mandelbrot kümesi [-1.28,0.0] zaman aralığında, dört boyutlu karmaşık sayının en son parametresi zaman olacak şekilde, hesaplanmıştır. Karmaşık sayının gerçel kısmı, i ve j parametreleri sırasıyla x, y ve z eksenlerine karşılık düşecek şekilde; [-2.1,0.9], [-1.5,1.5] ve [-1.5,1.5] aralığında alınmıştır. Verilen parametrelere göre bu aralıklarda istenen sayıda örnek alınabilmektedir.

Uygulamanın mümkün olduğunca modüler ve birbirinden bağımsız parçalar halinde yazılmasına dikkat edilmiştir.

2.2.1. Yönetici Süreçler ve Yük Dengeleme Yaklaşımları

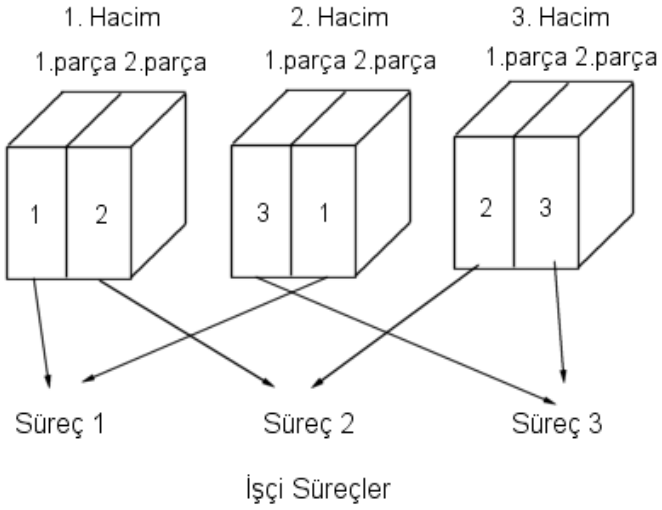
Uygulamada merkezi dinamik yük dengeleme kullanılmıştır. Sıfır numaralı süreç yük dağıtıcı ve sonuç toplayıcı olarak hizmet vermektedir. Uygulamada her ne kadar dinamik yük dengeleyici kullanılmış olsa da iş dağıtma algoritmasını değiştirerek statik yük dengelemeye benzer yük dengelemesi yapabilmenin mümkün olduğu unutulmamalıdır.

Yük dengeleyici süreçler işçi süreçlere dört adet dört bytelık işaretli tam sayı göndermektedirler. Bu sayılardan ilki zamanda toplam kaç adet örnek alınacağını göstermektedir. Bir diğer deyişle bu sayı belirlenen [-1.28,0.0] zaman aralığında kaç adet kare istendiğini belirtmektedir. İkinci değer hesaplama yapılacak zamanın örnek indeksidir. Bu değer hangi zamanda hesaplama yapılacağını göstermektedir. Üçüncü

parametre zaman dilimine ait üç boyutlu hacmin kaç parçaya ayrıldığını belirtmektedir. Dördüncü parametre ise hesaplanacak dilimin indisini vermektedir. Bu iş paketleri sayesinde sadece zamanda bölümleme veya hem zamanda hem de mekanda bölümleme mümkün olmuştur.

İşlerin bittiği işçi süreçlere -1 parametreleri gönderilerek bildirilmektedir.

İş dağıtma algoritması olarak iki adet algoritma gerçeklenmiştir. İlk algoritma işlemci çiftliği modeline uygun olarak işi biten işçiye sıradaki iş paketini göndermektedir. İkinci algoritma ise daha çok statik algoritmadaki yaklaşıma benzer şekilde davranarak süreç sırasına göre iş ataması yapmaktadır (Şekil 2.1). Böylelikle sıraya göre iş atamanın performans açısından değerlendirmesini yapmak mümkün olmuştur.



Şekil 2.1. Süreç numarasına göre iş dağıtma.

İşlerin sonucu olarak geriye gönderilen değer 1 bytelık işaretsiz tam sayıdır. İşçi süreçten okunan değer 1 ise algoritmaya göre sıradaki iş paketi işçiye atanır. Bu değer 2 olduğunda görselleştirme yapıldığı anlamına gelmektedir. Görselleştirilen verinin sonucu olan resim okunduktan sonra, gerekiyorsa hacim parçacıklarına ait resimler birleştirilip sabit diske yazılarak, yeni iş ataması gerçekleşmektedir. Hacim parçacıklarına ait resimler bilgisayarın belleğinde bütün hacim parçacıkları bitene kadar beklemektedir. Bu nedenle yönetici programın hafıza tüketimi koşma esnasında çeşitlilik gösterebilmektedir.

2.2.2. İşçi Süreçlerin Yapısı

Uygulamada ilk süreç hariç bütün süreçler işçi süreçler olmaktadır. İşçi süreçler programa verilen parametreleri ve yönetici süreçten aldıkları iş paketlerin kullanarak nasıl bir algoritma uygulayacaklarını belirlerler. İşçi süreçler dört adet fonksiyonu gerçekleyebilmektedirler. İlk olarak Mandelbrot kümesini hesaplayabilmektedirler. İkinci olarak hesaplanmış Mandelbrot kümesini sabit diske yazabilmektedirler. Üçüncü olarak sabit diskten Mandelbrot kümesini okuyabilmektedirler. Son olarak ise hafızada bulunan Mandelbrot kümesini görselleştirebilmektedirler.

Programa komut satırından verilen parametreler ve alınan iş paketlerine göre Mandelbrot kümesinin hesaplanması gereken kısımları kolaylıkla bulunabilmektedir. Hesaplanan kısımların sabit diske yazılması veya belli bir kısmın sabit diskten okunması gibi işlemler de kolaylıkla yapılabilmektedir. Eldeki verinin görselleştirilmesi ise daha karmaşık işlemler gerektirmektedir.

İşçi süreçler bu dört fonksiyonu kullanarak dört farklı çalışma algoritmasına sahip olabilmektedirler. İlk olarak sadece Mandelbrot kümesi hesaplanabilmektedir. İkinci olarak Mandelbrot kümesi hesaplanıp sabit diske yazılabilmektedir. Üçüncü olarak Mandelbrot kümesi hesaplanıp görselleştirilebilmektedir. Son olarak ise Mandelbrot kümesi sabit diskten okunarak görselleştirilebilmektedir.

İşçi süreçler görselleştirme yapmadıkları sürece geriye sadece 1 bytelık işaretsiz tam sayı olarak 1 döndürürler. Görselleştirme yapılan durumlarda 2 değeri gönderildikten sonra görselleştirme sonucu oluşan resim geriye gönderilir.

Bu dört algoritma sayesinde sabit disk erişimlerinin ve hesaplama yoğunluğunun sistem üzerindeki etkileri görülebilmektedir.

2.2.3. Zamanla Değişen Hacimsel Verinin Paralel Görselleştirilmesi

Zamanla değişen hacimsel verilerin görselleştirilmesi tek bir hacimsel verinin görselleştirilmesinden farklı yaklaşımlar gerektirmektedir. Bu konuda bir çok çalışma yapılmıştır [34-42].

Geliştirilen uygulamada görselleştirme algoritması olarak splatting tekniği kullanılmıştır. Görselleştirme algoritması eldeki veriyi ilk olarak ön işlemden geçirmekte daha sonra görselleştirmektedir.

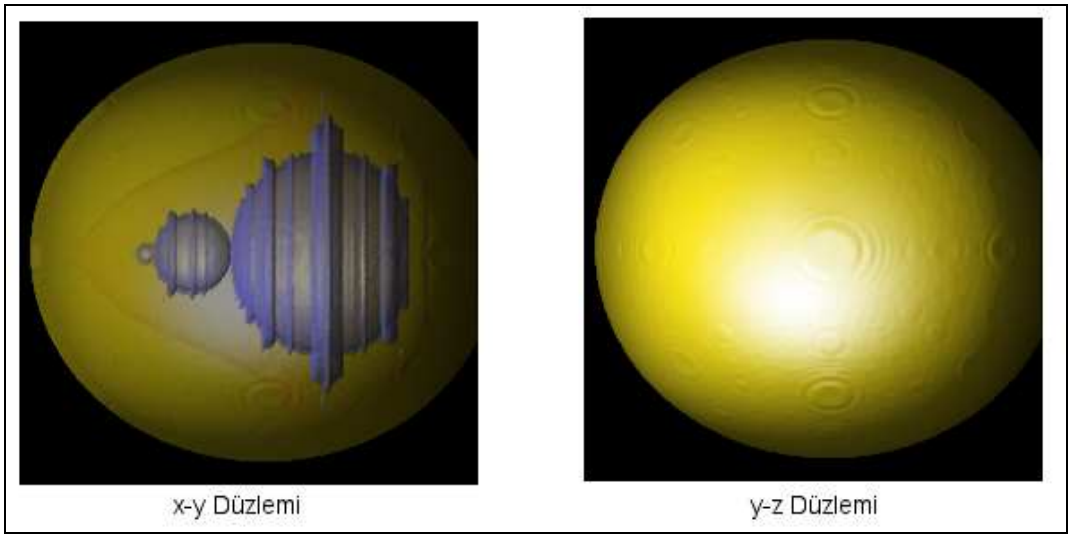
Paralelleştirme ise veri uzayını x-y düzleminde parçalara ayırarak yapılmıştır. Görüntü düzlemi bölünerek yapılan görselleştirmeler, görselleştirme esnasında gereken verinin başka bilgisayarlardan alınmasını gerektirecektir. Oysa veri uzayı bölündüğünde her işçi kendi hesapladığı bölgeyi görselleştirebilir. Veri uzayını bölümleyen algoritmalar daha az ağ haberleşmesi yapmakta ve bu sayede genellikle daha hızlı olmaktadırlar [42].

Bilindiği gibi splatting tekniği hacimdeki voksellerin ekrana düşürülmesi mantığına dayanmaktadır. Doğal olarak ek bir işlem yapılmadığı sürece genellikle bir eş yüzey bulunması söz konusu değildir. Bu nedenle Mandelbrot kümesi verisi görselleştirilmeden önce içindeki veriler dört farklı yoğunluk değeri kullanılarak üç eş yüzeye ayrılmaktadır. Bu uygulamada görselleştirme aşamasından önce eldeki veri Mandelbrot kümesinin 2, 4 ve 255 eş yüzeylerini ayıracak şekilde gruplanmıştır. Mandelbrot kümesinde bu eş yüzeyler kolaylıkla çıkarılabilmektedirler. Eş yüzey için yüksek değerlere çıkıldıkça, bu değerlere ait az sayıda nokta olduğundan eş yüzey bulmak zorlaşmaktadır. Bu nedenle bu tür yüzeylerin görselleştirilmesi isteniyorsa tri-lineer interpolasyonla bu yüzeylere ait yeni noktalar oluşturabilmek mümkündür.

Veri gruplandıktan sonra görselleştirilme yapılacak resimler için bellekte yer ayrılır. Bu resimler RGBA formatında piksel başına dört byte veri tutmaktadır. Bu bileşenler ilk değer olarak 0 değerini almaktadırlar. Resimlerin boyutu ise verinin boyutuyla aynı olarak seçilmiştir.

Verilen verinin hacmin neresinde bulunduğu alınan ve komut satırından girilen parametrelere bakılarak bulunur. Görselleştirme işlemi gözlemciye en yakın düzlemden en uzak düzleme, önden arkaya, doğru yapılır (Bağıntı 14-15).

Görselleştirmeyi sadece gerekli pikseller için yapmak gerekmektedir. Bu nedenle çeşitli noktaların elemesi yapılmaktadır. İlk olarak görselleştirilecek noktanın istenilen eş yüzey yoğunluklarından birine sahip olup olmadığı kontrol edilir. Bu sayede istenmeyen noktalar görselleştirilmez. Bu aşamadan geçen noktalar y ve z eksenleri etrafında 45 derece döndürülür. Splattingde görselleştirmeye bakış açısına en dik olan düzlemden başlamak gerekir, aksi halde görselleştirme sonucunda çeşitli bozulmalarla karşılaşılabilir (Şekil 2.2). Bu nedenle bu döndürme açısında görselleştirme y-z düzlemi kullanılarak yapılmıştır.



Şekil 2.2. Döndürülmemiş Mandelbrot kümesinde yanlış ve doğru düzlemlerde splatting

Bu döndürme işlemi sonrasında paralel projeksiyon yapıldığı için noktanın, x ve y koordinatlarına bakılarak, ekran içinde olup olmadığı kontrol edilir. Böylece ekranda gözükmeyecek verilerin görselleştirilmesi için işlem yapılmaz. Daha sonra noktanın etrafındaki komşularına bakılarak, noktanın hacmin içinde kalıp kalmadığı kontrol edilir. Nokta eğer yüzeyi oluşturan noktalardan biriyse, nokta için yüzey normali hesaplanır. Bu hesap yoğunluk farklarına bakılarak yapılmaktadır (Bağıntı 23-26).

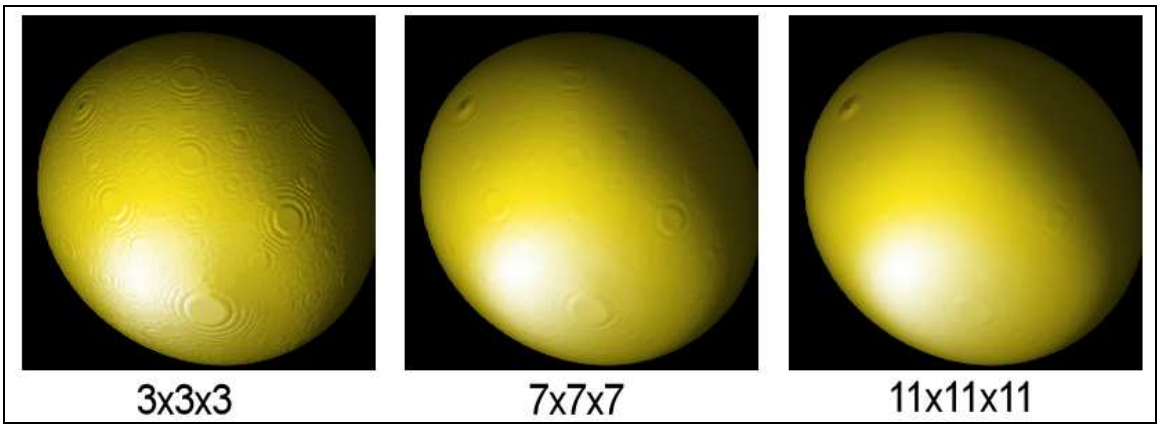
$$N_x = \sum_{r=-n}^{-1}\sum_{i=-n}^{n}\sum_{j=-n}^{n}\left(veri[x][y][z] - veri[x+r][y+i][z+j]\right)/|r| + \sum_{r=1}^{n}\sum_{i=-n}^{n}\sum_{j=-n}^{n}\left(veri[x+r][y+i][z+j] - veri[x][y][z]\right)/r \quad (23)$$

$$N_y = \sum_{r=-n}^{n}\sum_{i=-n}^{-1}\sum_{j=-n}^{n}\left(veri[x][y][z] - veri[x+r][y+i][z+j]\right)/|i| + \sum_{r=-n}^{n}\sum_{i=1}^{n}\sum_{j=-n}^{n}\left(veri[x+r][y+i][z+j] - veri[x][y][z]\right)/i \quad (24)$$

$$N_z = \sum_{r=-n}^{n}\sum_{i=-n}^{n}\sum_{j=-n}^{-1}\left(veri[x][y][z] - veri[x+r][y+i][z+j]\right)/|j| + \sum_{r=-n}^{n}\sum_{i=-n}^{n}\sum_{j=1}^{n}\left(veri[x+r][y+i][z+j] - veri[x][y][z]\right)/i \quad (25)$$

$$\vec{N} = (N_x, N_y, N_z) \quad (26)$$

Burada herhangi bir (x,y,z) noktası için (2n+1) boyuta sahip kübik bir hacimde yüzey normali hesaplanmaktadır. Bu hesaplama yapılırken yoğunluk farklarının noktanın yerel konumuna olan uzaklığına bakılmaktadır. Yüzey normalinin ne kadar bir alandan hesaplanacağı kullanıcı tarafından programa parametre olarak verilmektedir. Bu değer büyük seçildiğinde görüntü yumuşak ve az detaylı çıkmakta aynı zamanda hesaplama süresi uzamaktadır; küçük seçildiğinde ise görüntü keskin ve detaylı çıkmakta, hesaplama süresi azalmaktadır (Şekil 2.3).

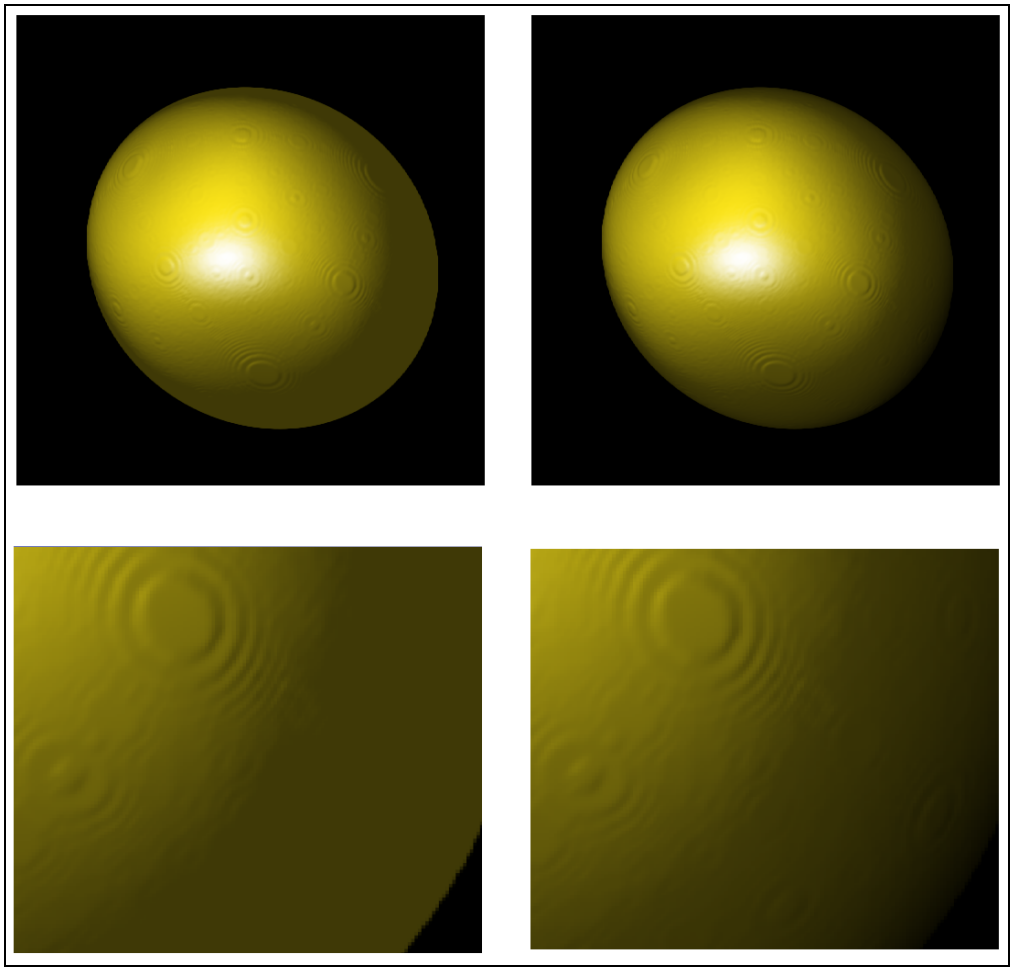


Şekil 2.3. Yüzey normali bulurken kullanılan hacmin görüntüdeki etkisi

Yüzey normali bulunduktan sonra aydınlatma için gereken renk değerleri Phong aydınlatma modeline benzer şekilde bulunmaktadır. Bilindiği gibi Phong aydınlatma modelinde üç adet bileşen bulunmaktadır. Specular ve diffuse bileşenler aydınlatmada kullanılan açılarının 90 dereceden fazla olması durumunda kaybolmaktadırlar. Cismin bütününde etkili olan ambient aydınlatma sayesinde bu iki bileşenin de etkisiz olduğu bölgeler siyah çıkmak yerine belli bir sabit değeri almaktadırlar. Bunun sonucunda cismin dış hatları rahatlıkla seçilebilecek hale gelmektedir. Ancak cismin karanlıkta kalan kısmının geometrik yapısını görmek mümkün olmamaktadır (Şekil 2.4). Bu durumu gidermek için bir çok yöntem önerilmiştir [43]. Örneğin gözlemcinin baktığı noktada sabit bir ışık kaynağı olduğu farz edilebilmektedir. Ancak bu yöntem yeni bir ışık kaynağı için hesaplama gerektireceğinden yavaş olacaktır.

Geliştirilen uygulamada mümkün olan minimum performans kaybıyla, karanlıkta kalan bölgelerin geometrik yapısını incelemek için diffuse aydınlatmada kullanılan açının kosinüsü kullanılmıştır. Bilindiği gibi diffuse aydınlatma açısının kosinüsünün sıfırdan

büyük olması durumunda diffuse aydınlatma yapılabilmektedir. Bu değerin negatif olması durumunda yalnızca ambient aydınlatma yapılmaktadır. Bu negatif değer yüzeyin geometrisi hakkında bilgi verdiğinden karanlıkta kalan bölgeleri aydınlatmada kullanılabilir.



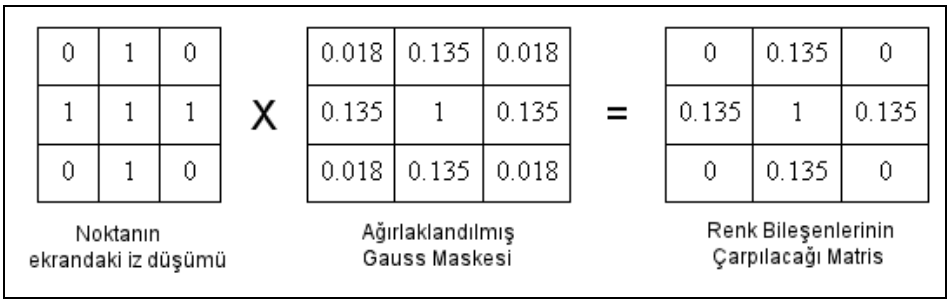
Şekil 2.4. Phong aydınlatma modeli ve kullanılan aydınlatma modeli

Uygulamada kullanılan ambient ışık, ışık alan bölgelerde Phong modeline göre davranmaktadır. Işık almayan, diffuse açısının kosinüsü negatif olan, durumlarda ise diffuse aydınlatmada kullanılan açının kosinüsü 1 ile toplanarak yeni bir katsayı elde edilir. Bu katsayı daha sonra ambient ışıklandırmada kullanılan katsayıyla çarpılarak yeni bir ambient ışıklandırma katsayısı elde edilir. Böylece cismin karanlıkta kalan kısmının geometrik yapısı görülerek daha gerçekçi bir sonuç elde edilebilir (Şekil 2.4). Bu tekniğin tek olumsuz tarafı karanlıkta kalan bölgelerin parlaklıklarının bir miktar azalmasıdır. Bu etkiden kurtulmak için daha büyük ambient aydınlatma katsayıları kullanılabilir.

Aydınlatma sonucu yüzey rengi bulunduktan sonra, yine kullanıcının dışarıdan parametre olarak girdiği, noktanın ekrandaki izdüşüm alanı belirlenir. Bu alan, değerleri 1'e normalize edilmiş, standart sapması her iki yönde de 0.5 olan ve (1,1)-(-1,-1) aralığında hesaplanan, Gauss maskesiyle çarpılır. Bire normalize edilmiş Gauss maskesini elde etmek için kullanılan bağıntı aşağıda verilmiştir.

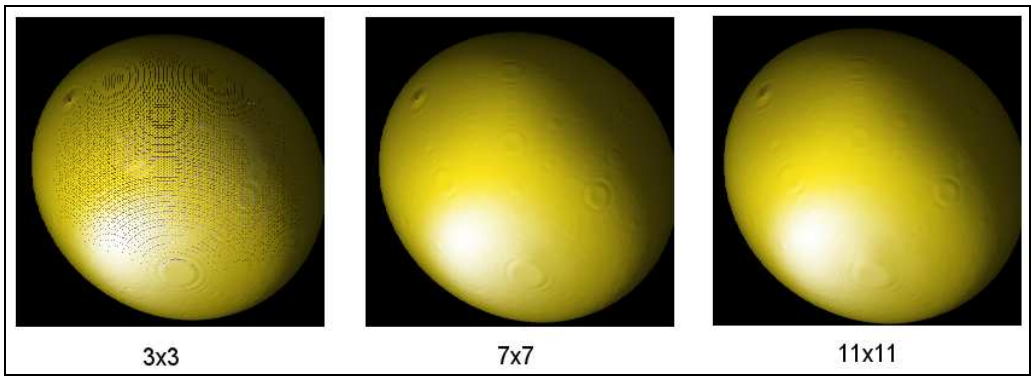
$$G_{x,y} = e^{-\left(x^2+y^2\right)/2\sigma^2} \tag{27}$$

Elde edilen bu maske cismin geçirgenlik katsayısı dahil bütün renk değerleriyle ayrı ayrı çarpılarak önden arkaya renk toplama bağıntıları ile (Bağıntı14-15) ekrana eklenir. 1 yarı çaplı, 3x3 lük bir maskenin nasıl elde edildiği Şekil 2.5'de verilmiştir.



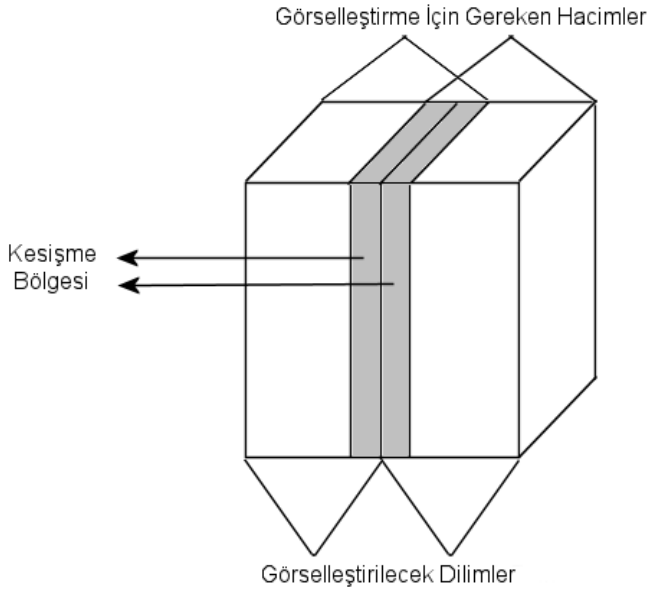
Şekil 2.5. Renk bileşenlerinin çarpılacağı matrisin elde edilmesi.

Noktanın ekrandaki izdüşümünün çapı çok büyük olması durumunda resim aşırı yumuşayarak bozulur, çok küçük olması durumunda ise noktaların arasında boşluklar kalır [19] (Şekil 2.6).



Şekil 2.6. Çeşitli maske büyüklüklerinin etkisi.

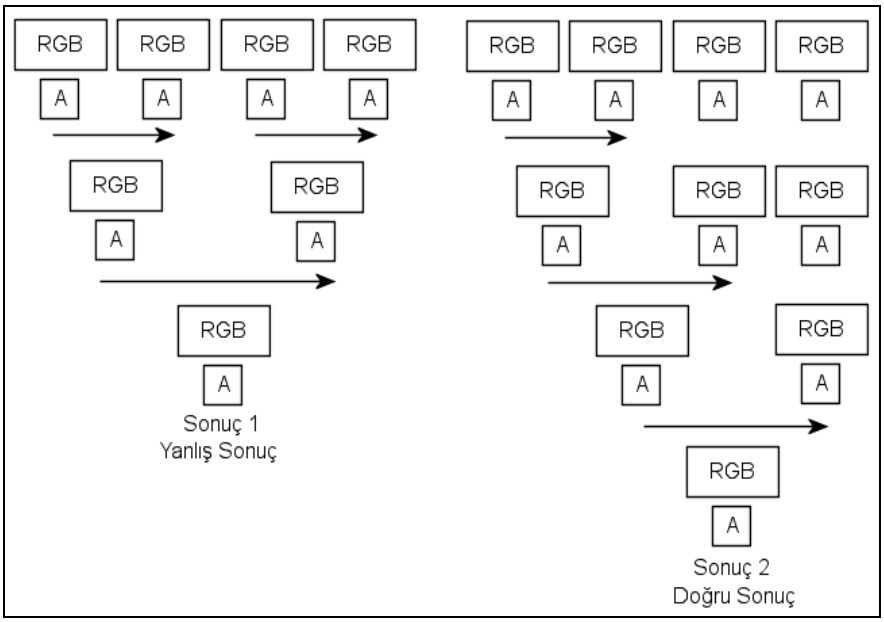
Üç boyutlu hacimler dilimlere ayrılmadığında görselleştirme algoritması sorunsuz şekilde çalışmaktadır. Fakat, üç boyutlu hacimlerin de dilimlere ayrılması durumunda her dilim kendi içinde görselleştirileceğinden sonuç resminin elde edilmesi sırasında bazı sorunlar çıkabilmektedir. Aynı zamanda yüzey normallerinin belli bir hacimden hesaplandığı düşünüldüğünde belli alanların ya iki kere hesaplanması yada sabit diskten ek olarak okunması gerekecektir. Bu çalışmada görselleştirme yapılacak dilimin sadece başlangıç, sadece bitiş yada hem başlangıç hem bitişinden itibaren yüzey normali çapı kadar daha y-z düzlemi hesaplanmış yada okunmuştur (Şekil 2.7).



Şekil 2.7. Yüzey normali hesaplanması için gereken ek bölgeler.

Görselleştirilen dilim resimlerinin birleştirilmesi için çeşitli yöntemler bulunmaktadır [42,44]. Özellikle saydam cisimlerde cismin niteliğine ve bakış açısına göre çeşitli problemler oluşabilmektedir.

Bu resimlerin birleştirilmesinde önden arkaya toplama işlemi yapmak doğru sonuç vermemektedir. Bunun nedeni önden arkaya toplanarak elde edilen dilim resimlerinin hepsinin ilk olarak sıfır olan renk ve geçirgenlik değerine sahip resimlerden başlamış olmasıdır. Bir diğer deyişle dört rengi önden arkaya toplamakla ikili gruplar halinde toplayarak, bu iki toplamanın sonucunu üçüncü bir önden arkaya toplama işlemine tabi tutmak aynı sonucu vermemektedir (Şekil 2.8).



Şekil 2.8. Doğru ve yanlış önden arkaya toplama.

Doğru sonucun bulunabilmesi için, önden arkaya doğru toplamanın sıfır renk ve geçirgenlik değerinden başladığı göz önüne alınarak, P rengi ve A geçirgenlik katsayısına sahip bir resimden başlandığında elde edilecek sonucun nasıl fark edeceğinin düşünülmesi gerekmektedir.

Bu durum göz önüne alındığında doğru toplamayı yapacak bağıntılar şu şekilde olmalıdır:

$$C_{i+1}(P,R) = C_i(P,R) + C(P,R) \times (1 - \alpha(P,R)) \quad (28)$$

$$\alpha_{i+1}(P,R) = (1 - \alpha(P,R)) \times \alpha_i(P,R) + \alpha(P,R) \quad (29)$$

Bu bağıntılarda;

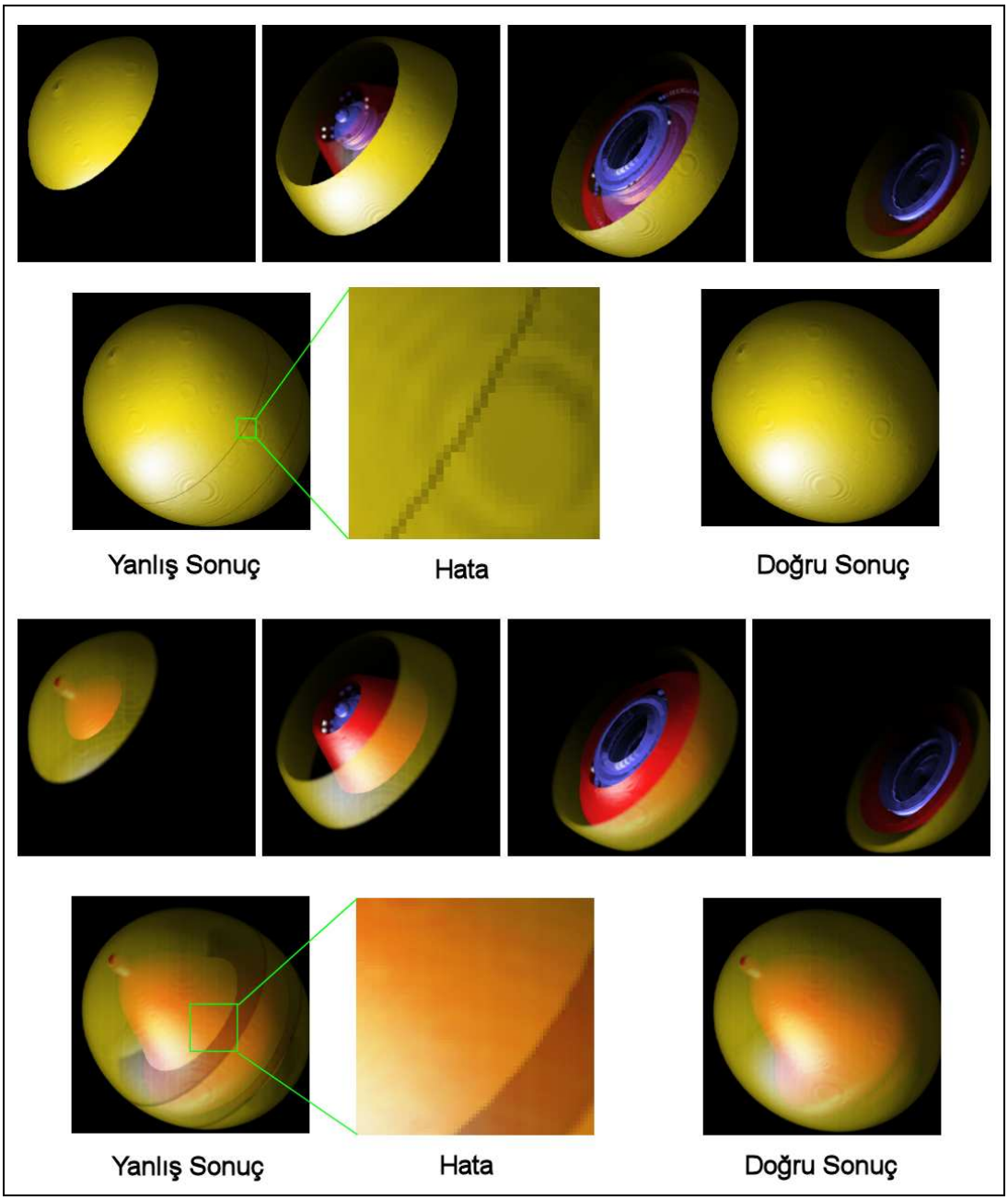
$C_i(P,R), C_{i+1}(P,R)$ i. örneği eklemeden önceki ve sonraki renk değerleri,

$\alpha_i(P,R), \alpha_{i+1}(P,R)$ i. örnek toplanmadan önceki ve sonraki geçirgenlik değerleri,

$C(P,R)$ piksele eklenecek n. renk,

$\alpha(P,R)$ eklenecek rengin geçirgenlik değeridir.

Bu bağıntı kullanılarak sonuç resimleri önden arkaya toplanabilmektedir. Normal önden arkaya toplama işlemiyle oluşabilecek hatalar ve düzeltilmiş önden arkaya toplama işlemi sonucunda elde edilen doğru resimler Şekil 2.9’ da verilmiştir.



Şekil 2.9. Saydam olmayan yüzeylerde hatalı toplama.

3. BULGULAR VE TARTIŞMA

3.1. Geliştirilen Paralel Görselleştirme Uygulamasının Performans Değerlendirmesi

Uygulama, bilgisayar sayısı; bilgisayar başına düşen süreç sayısı; hacim dilim sayısı; Gauss maske boyu; yüzey normali hesaplama hacmi; hacim içindeki örnekleme sayısı ve yük dengeleme algoritması bakımından, incelenerek her bileşenin sistem performansına ve dengesine etkisi ortaya çıkarılmaya çalışılmıştır. Deneylerde aksi belirtilmediği sürece yüzey normali bulma hacmi 3x3x3, Gauss maskesi boyu 7x7, zaman dilim sayısı 128, hacim dilim sayısı 1, bilgisayar başına düşen süreç sayısı 2 olarak alınmıştır.

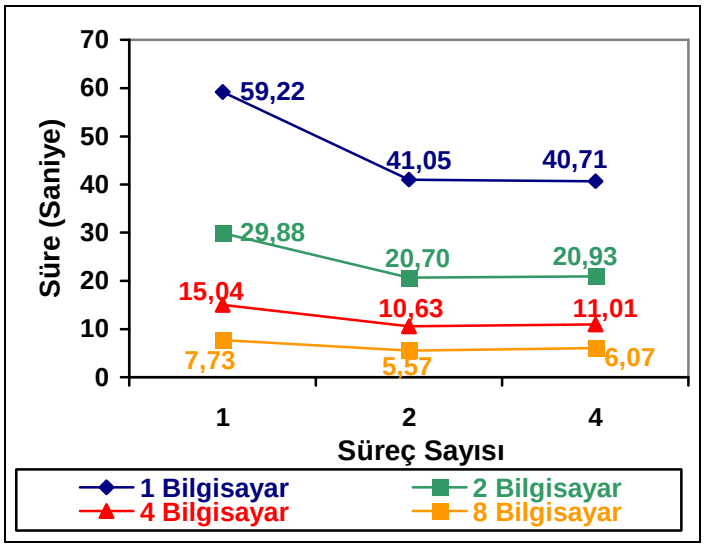
3.1.1. Zamanla Değişen Verinin Hesaplanarak Görselleştirilmesi

Statik ve dinamik iş dağıtımı yapılarak, bilgisayar sayısının sistem performansı üzerinde ne gibi etkileri olduğu incelenmiştir. Bu amaçla zamanla değişen 128x128x128x128 voksellik, 2568x256x256x128 voksellik ve 512x512x512x128 voksellik Mandelbrot kümeleri hesaplanarak görselleştirilmiştir.

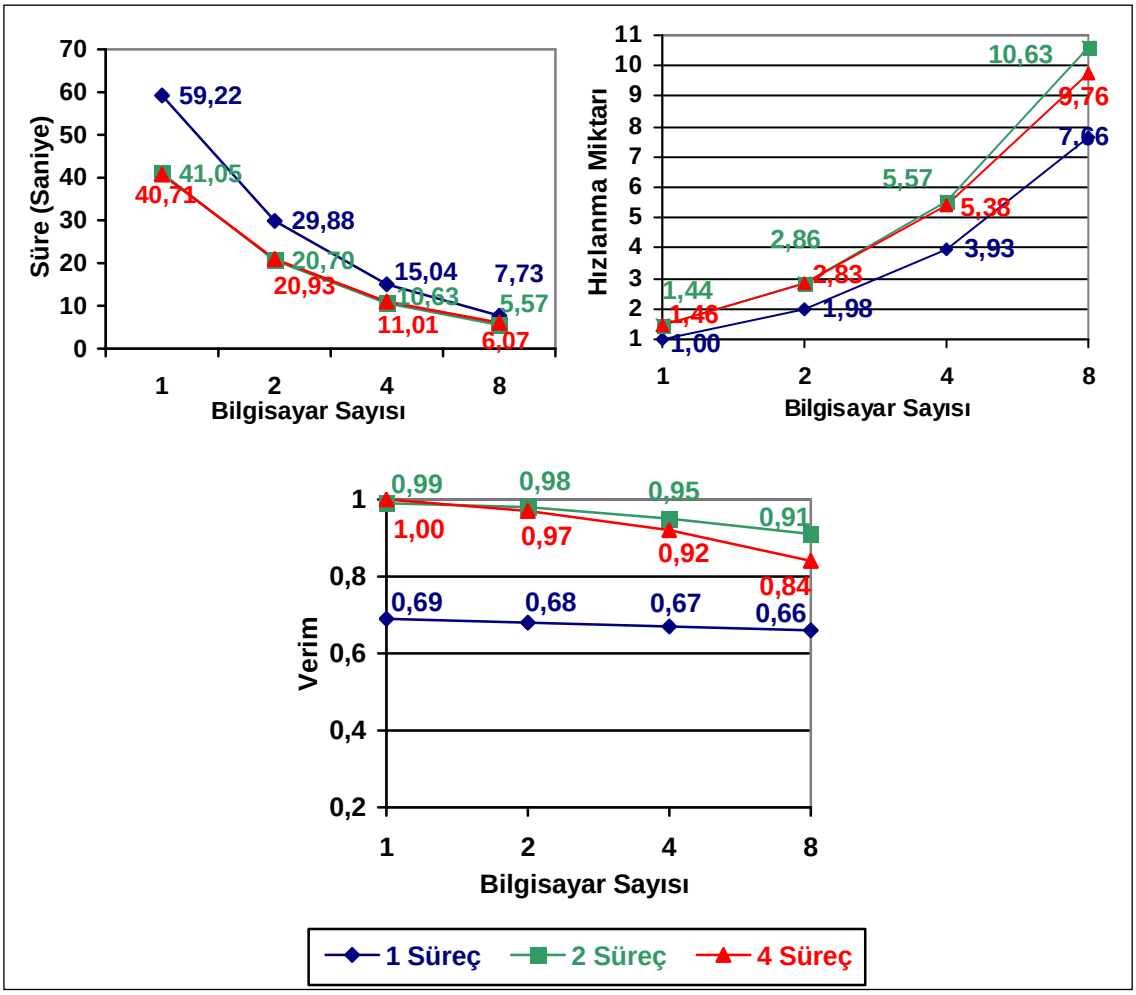
3.1.1.1. Statik Yük Dağıtımı Kullanılarak Hesaplama ve Görselleştirme

İlk olarak 128x128x128 voksellik Mandelbrot kümesinin hesaplanarak görselleştirilmesi incelenmiştir. Sıralı iş dağıtımı yapılarak, bilgisayar sayısının sistem performansı üzerinde ne gibi etkileri olduğu incelenmiştir. Statik yük dağıtımı ile 1, 2, 4 ve 8 bilgisayar kullanılması ve bilgisayar başına 1, 2 ve 4 süreç düşmesi durumunda performansın nasıl etkilendiği Şekil 3.1 ve Şekil 3.2 ’de verilmiştir.

Şekillerden de görülebileceği gibi bilgisayar sayısı arttırıldığında ilk önce performans doğrusal hızlanma miktarı kadar artmaktadır ancak bilgisayar sayısı artırılmaya devam ettirildiğinde hızlanma artış miktarı azalmaktadır. Ayrıca süreç sayısının artmasıyla bu durum daha belirgin bir hal almaktadır.



Şekil 3.1. 128x128x128 voksellik hesaplamanın süreç – süre grafiği

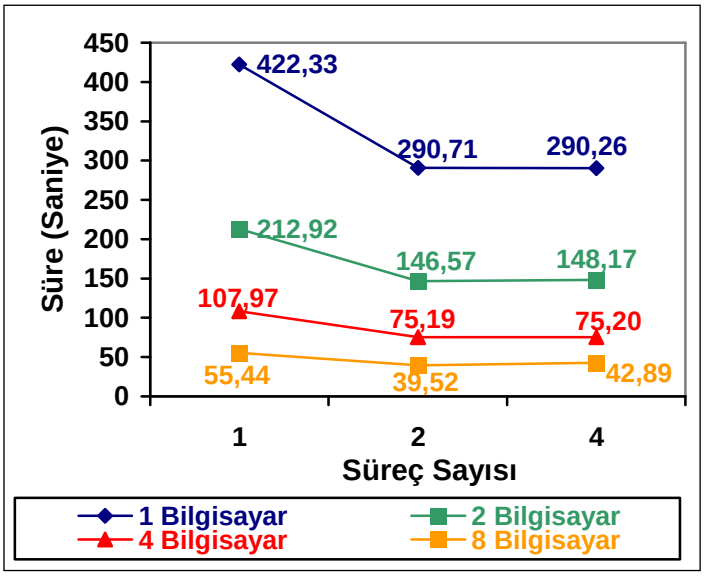


Şekil 3.2. 128x128x128 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

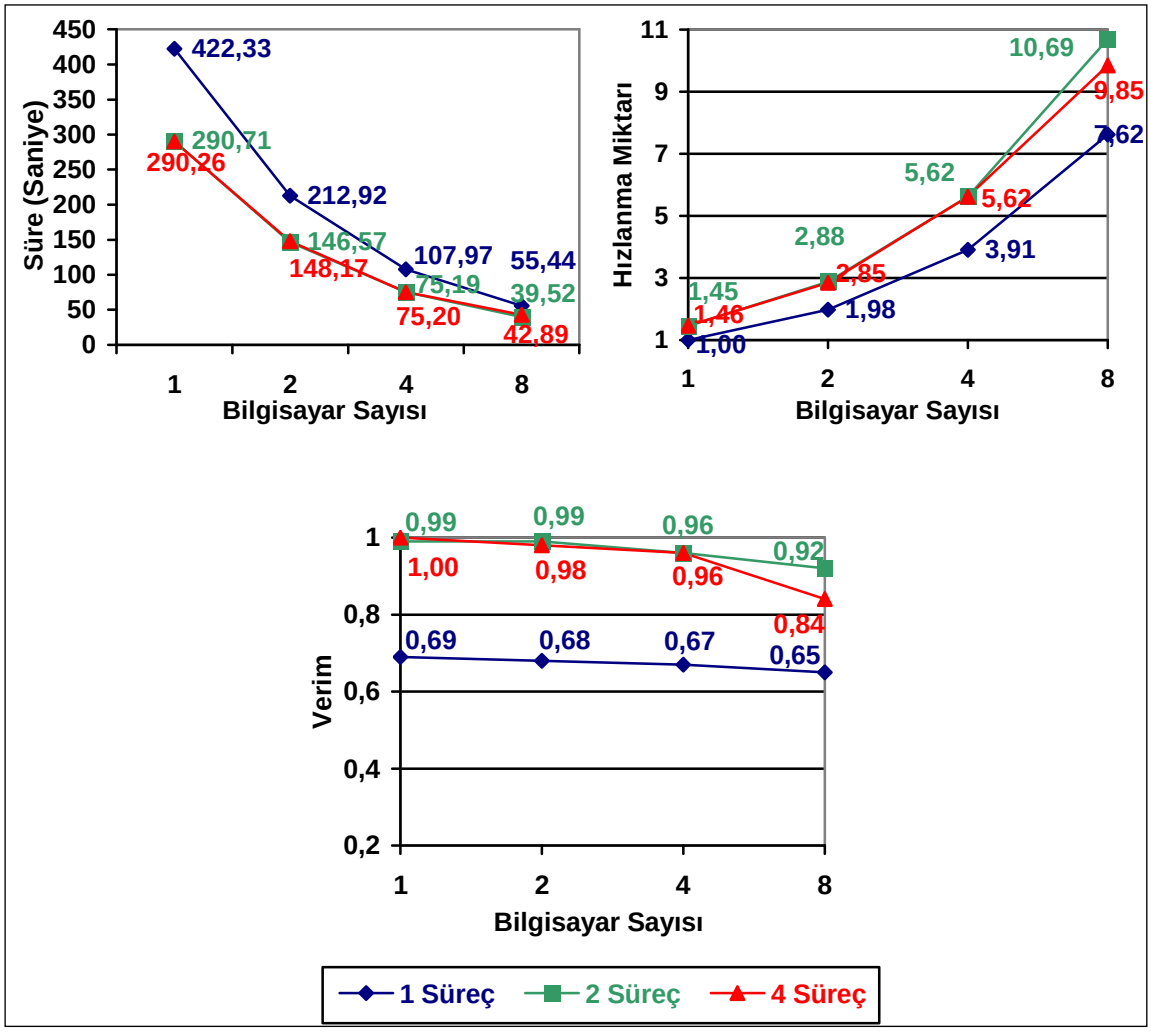
Deneylerde bir diğer dikkat çekici nokta, süreç sayısının birden ikiye çıkarılması durumunda elde edilen %25 civarındaki performans artışıdır. Bunun nedeni işlemcilerin Hyper-Threading destekli oluşudur. Bu teknolojide işlemciler iki işlemci gibi davranmaktadırlar. Bu teknoloji iki işlemci kullanmak yerine işlemcinin boşta kalan işlem birimlerini kullanılmaktadır. Bu nedenle aynı işlerin yapılacağı durumlarda performans artışı sınırlı kalabilmektedir. Ayrıca bellek bant genişliği sabit kaldığından yeterli işlem birimi boşta kalsa da performans artışı sağlanamayabilir. Ek olarak farklı bellek bölgelerinin kullanımı sonucu önbellek içeriğinin devamlı olarak değişmesi durumunda performans azalması yaşanabilmektedir. Sonuç olarak açıklanan nedenlerden ötürü performans artışı pratik olarak iki katına çıkmamaktadır.

Süreç sayısı ikiden dörde çıkarıldığında tek bilgisayarla az miktarda performans artışı elde edilmiş, fakat bilgisayar sayısı arttıkça hızlanma miktarında azalma görülmüştür. Bunun nedeni ise çok sayıda sürecin aynı anda sisteme sonuç göndermesidir. Gönderilen sonuçlar dengeli olarak gelmediğinden yük dengeleyici sabit disk erişimi yaparken süreçler iş almak için boşta beklemektedirler.

İkinci olarak yine aynı şartlar altında 256x256x256 voksellik Mandelbrot kümesi hesaplanarak görselleştirilmiştir. Elde edilen sonuçlar Şekil 3.3 ve Şekil 3.4 ’de verilmiştir.



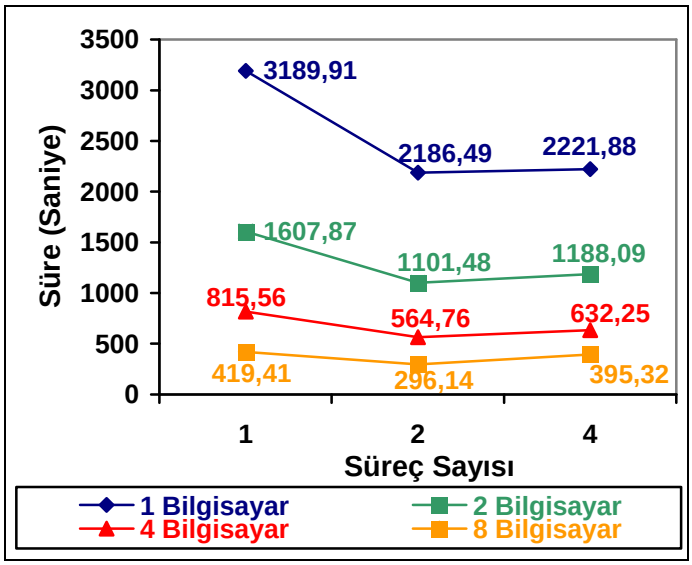
Şekil 3.3. 256x256x256 voksellik hesaplamanın süreç – süre grafiği



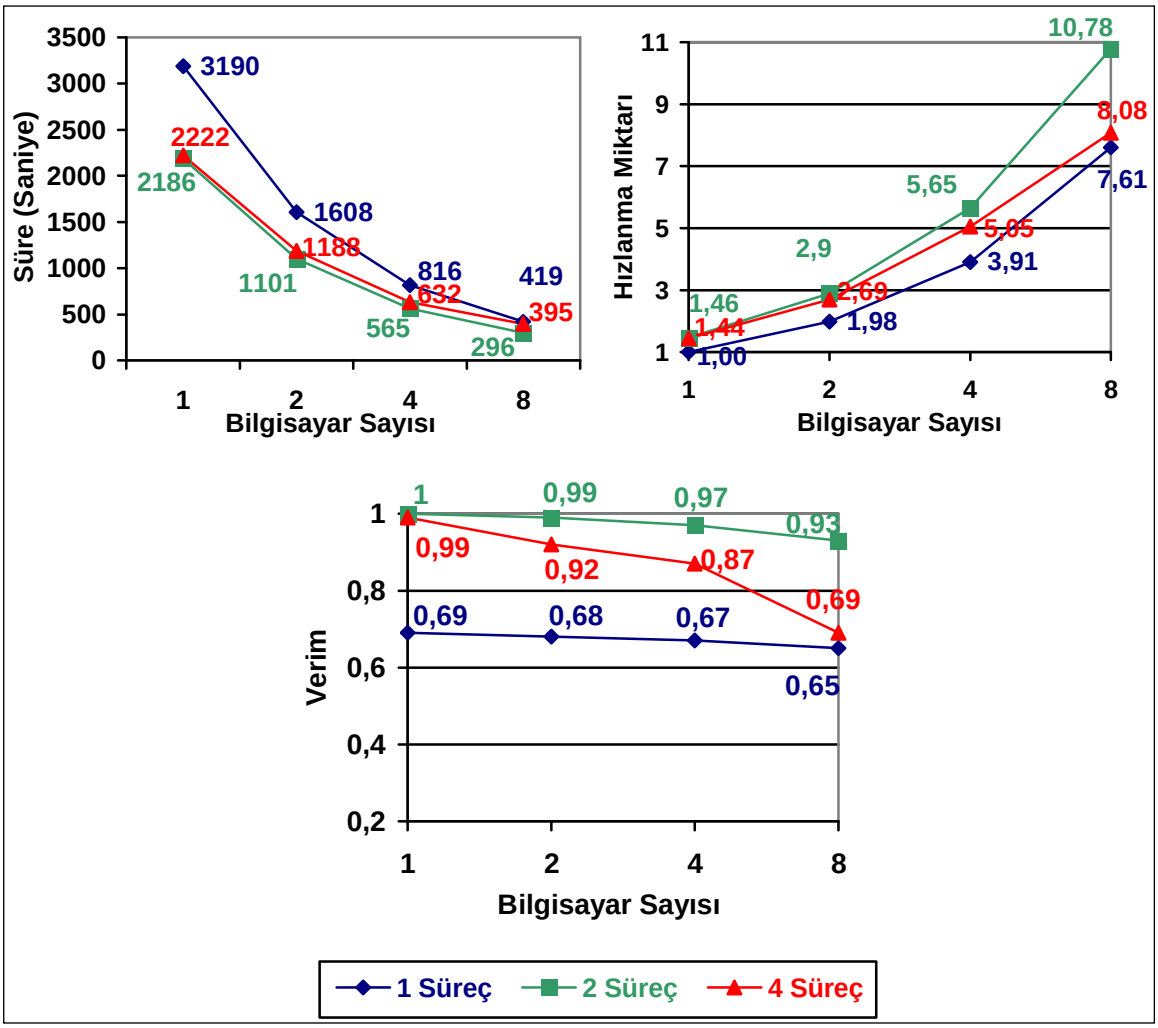
Şekil 3.4. 256x256x256 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

256x256x256 voksellik hacmin görselleştirilmesi, 128x128x128 voksellik hacmin görselleştirilmesiyle benzerlikler göstermektedir. Bu görselleştirmede resim boyu büyüdüğü için ağ haberleşmesi bir miktar artmış olsa da bu durum sonuçlarda belirgin bir farklılık oluşturmamıştır.

Son olarak 512x512x512 voksellik Mandelbrot kümesi görselleştirilerek yük dengeleme algoritmasının davranışlarının işlem yoğun durumlarda ne gibi özellikler gösterdiği test edilmiştir. Süreç sayısının bir , iki ve dört olması durumlarında elde edilen sonuçlar, Şekil 3.5 ve Şekil 3.6'da verilmiştir.



Şekil 3.5. 512x512x512 voksellik hesaplamanın süreç – süre grafiği



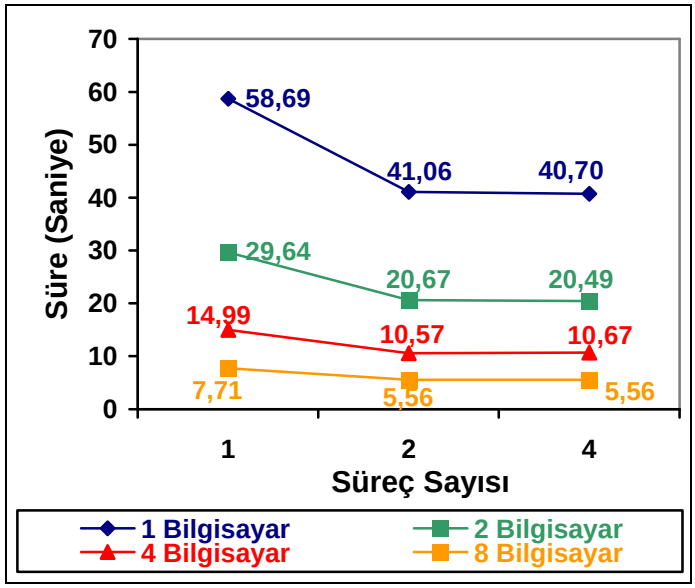
Şekil 3.6. 512x512x512 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

Şekillerde görüldüğü gibi bir ve iki süreç koşturulduğunda daha önceki deneylere benzer davranışlar görülmekle birlikte süreç sayısı dörde çıkarıldığında hızlanma artış miktarında bir azalma oluşmaktadır. Bu azalmanın nedeni sistem belleklerinin ihtiyaç duyulan bellekten daha az olmasıdır. Her örnek bir bytelık değerler halinde tutulduğundan 512x512x512 voksellik bir hacmin görselleştirilmesi için yaklaşık olarak 128 MB hafıza gerektirmektedir. Dört süreç çalıştığı düşünüldüğünde gerekli belleğin 512 MB olduğu görülür. Bilgisayarlarda çalışan diğer yazılımların da hafıza gerektirdiği düşünülürse, sistemlerin aktif olarak sabit diskten sanal bellek kullandıkları sonucu çıkarılabilir. Bu nedenle her bilgisayarda dört süreç çalıştırmak performans kaybına neden olmaktadır.

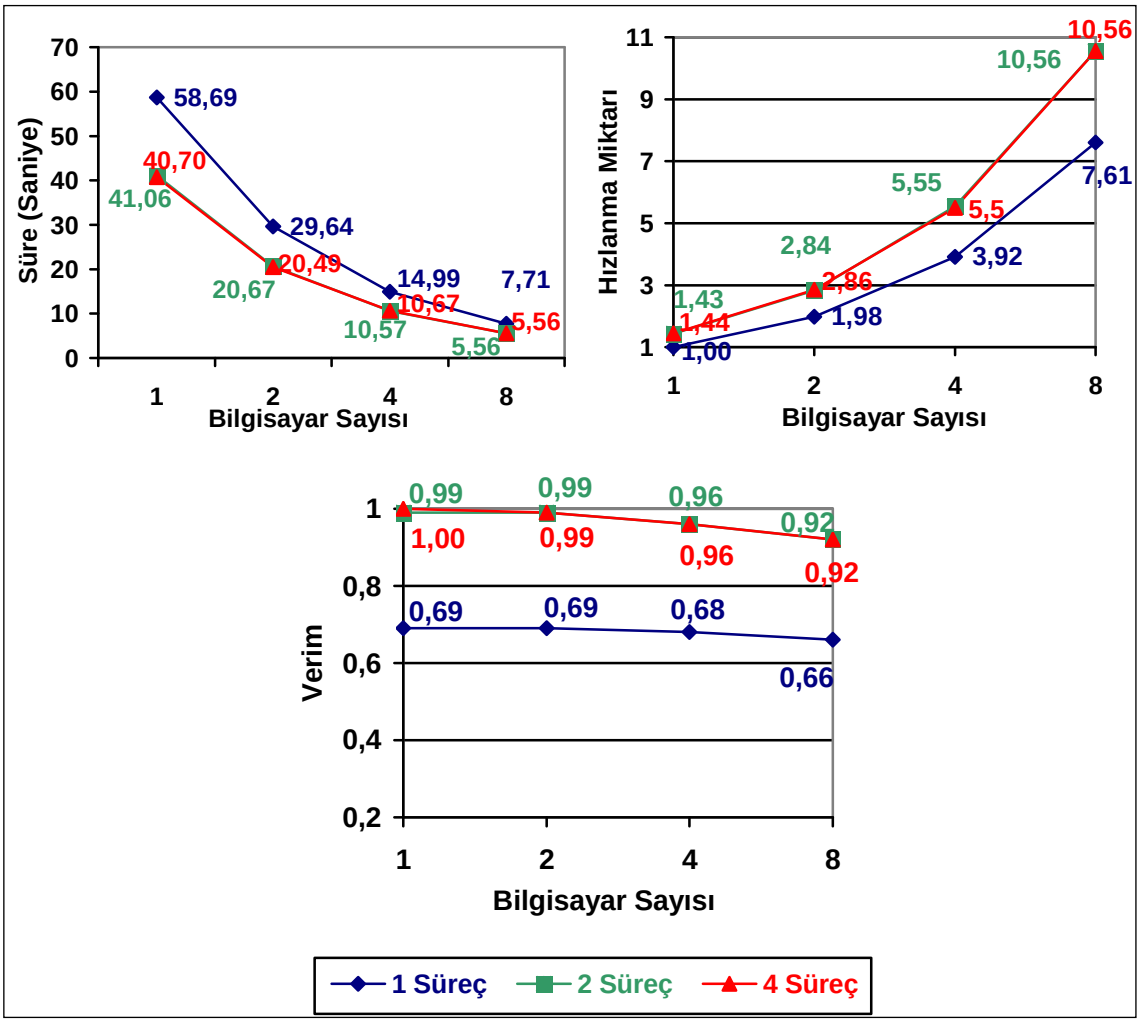
3.1.1.2. Dinamik Yük Dağıtımı Kullanılarak Hesaplama ve Görselleştirme

Yukarıda statik yük dağıtımı yapılması durumunda elde edilen sonuçların daha iyi anlaşılabilmesi ve dinamik olarak yük dağıtımının farklarını görebilmek için aynı deneyler dinamik yük dengelemesi ile tekrarlanmıştır.

128x128x128 voksellik hacmin görselleştirilmesi durumunu yansıtan sonuçlar Şekil 3.7 ve Şekil 3.8‘de verilmiştir.



Şekil 3.7. 128x128x128 voksellik hesaplamanın süreç – süre grafiği

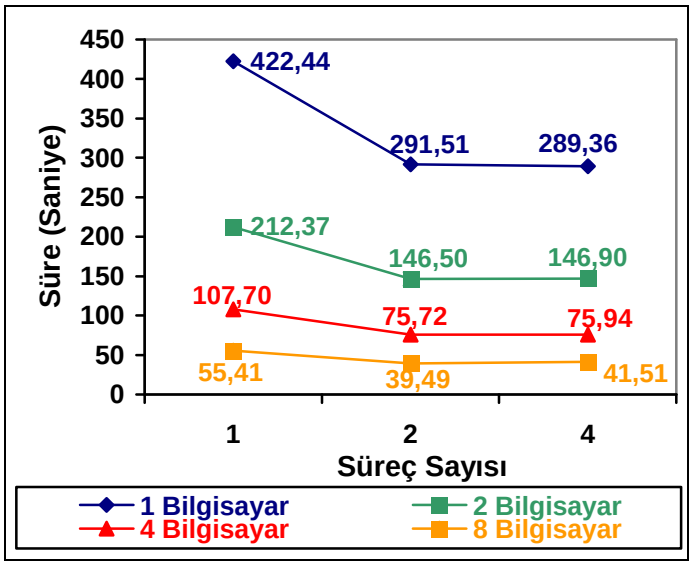


Şekil 3.8. 128x128x128 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

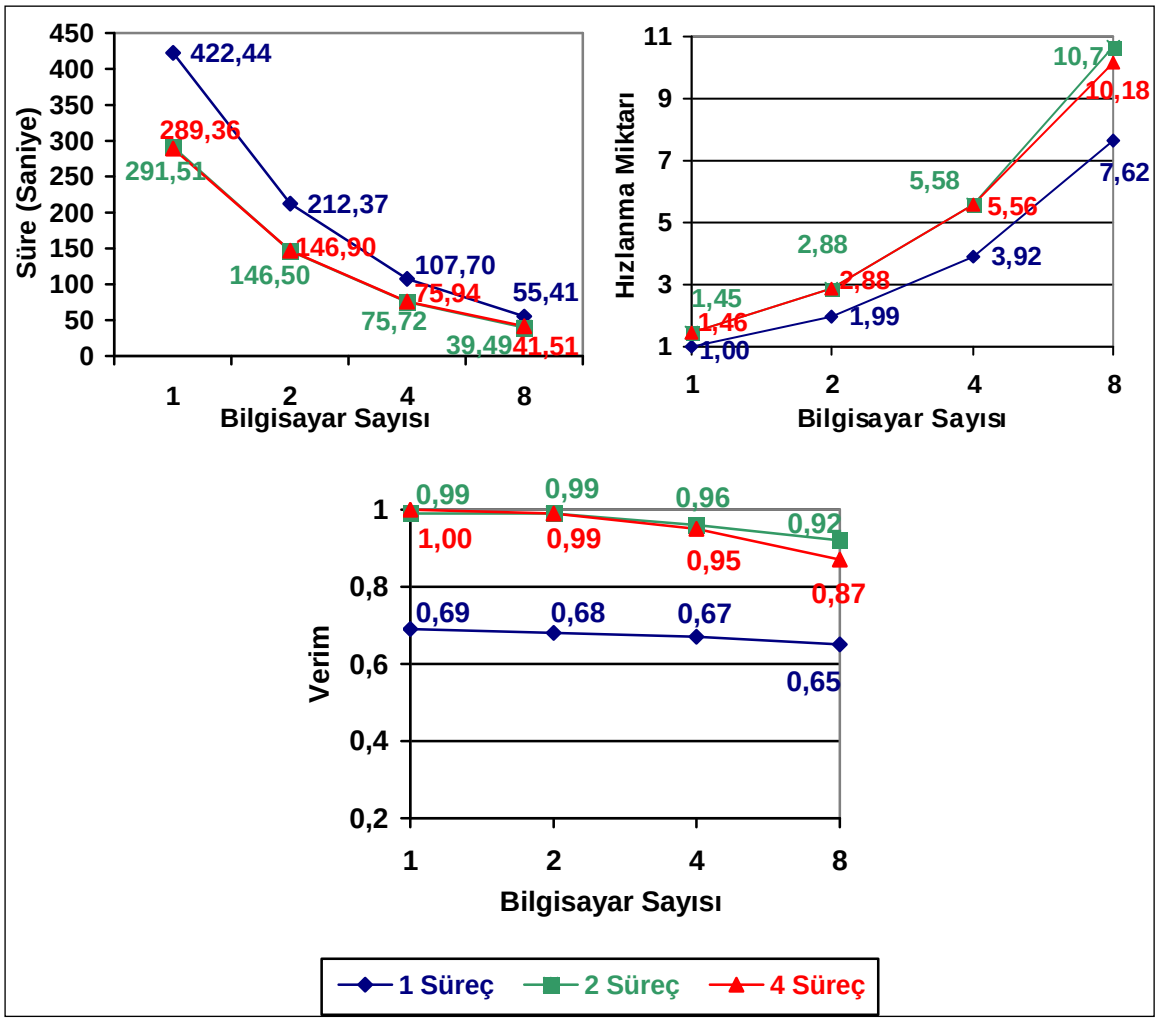
Bu durumda süreler benzer olmakla birlikte dinamik olarak yük dağıtan algoritmanın, statik yük dağıtanla karşılaştırıldığında daha hızlı olduğu görülmektedir. Özellikle bilgisayar başına dört süreç koşturulduğunda statik yük dağıtma algoritmasından elde edilen sonuçlardan daha iyi sonuçlar elde edilmiştir.

Orta boyutlu zamanla değişen verilerde algoritmanın etkinliğini ölçmek için 256x256x256 voksellik Mandelbrot kümesinin hesaplanarak görselleştirilmesi incelenmiştir. Elde edilen performans değerleri Şekil 3.9 ve Şekil 3.10‘ da verilmiştir.

Şekillerden de görülebileceği gibi özellikle dört süreç çalıştırılırken ortaya çıkan hızlanma miktarındaki azalma dinamik yük dağıtan algoritmada statik olanla kıyaslandığında daha az görülmektedir. Bunun dışında, çoğunlukla, statik yük dengeleme mantığına göre çalışan algoritmayla benzer sonuçlar elde edilmiştir.

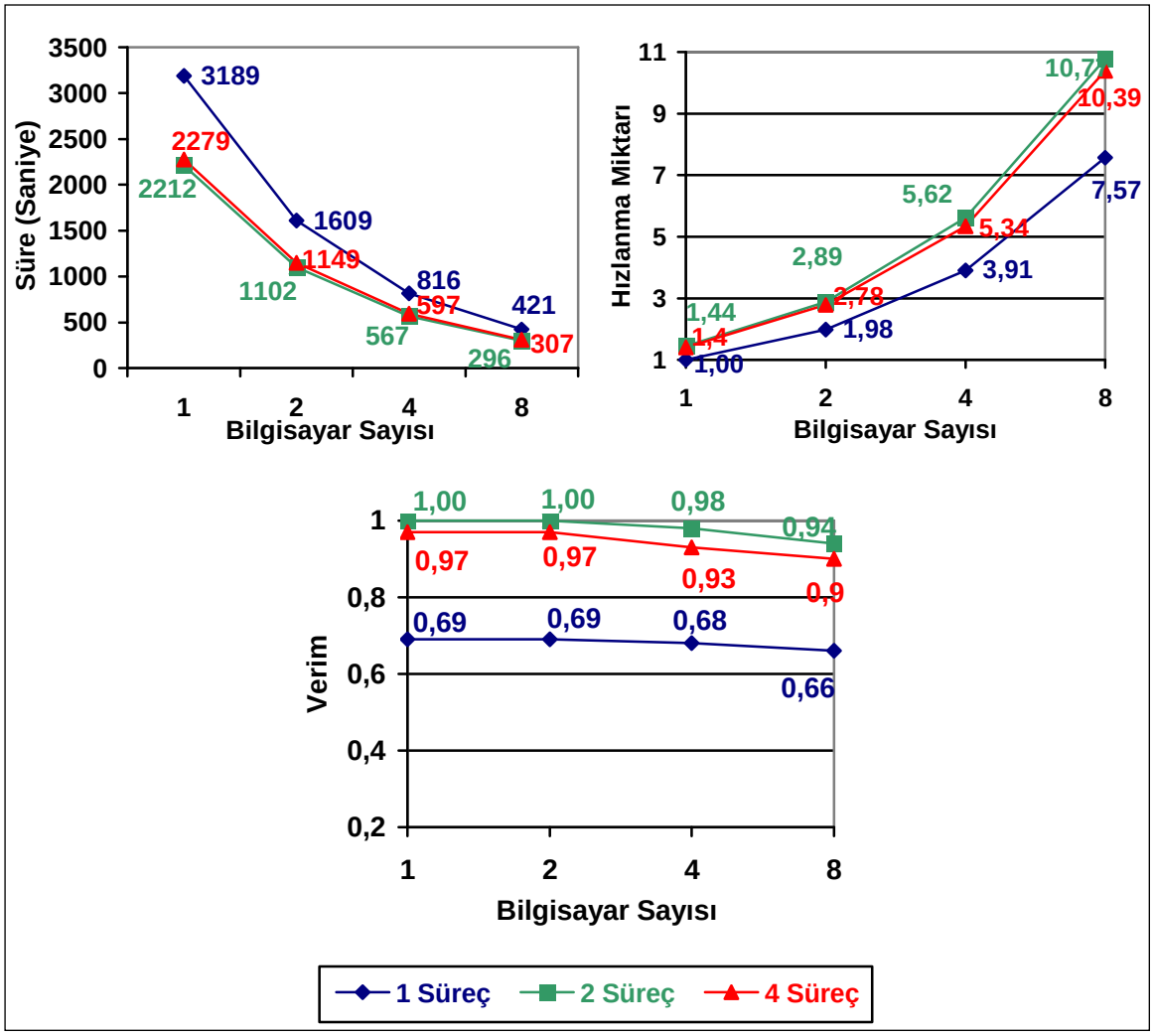


Şekil 3.9. 256x256x256 voksellik hesaplamanın süreç – süre grafiği



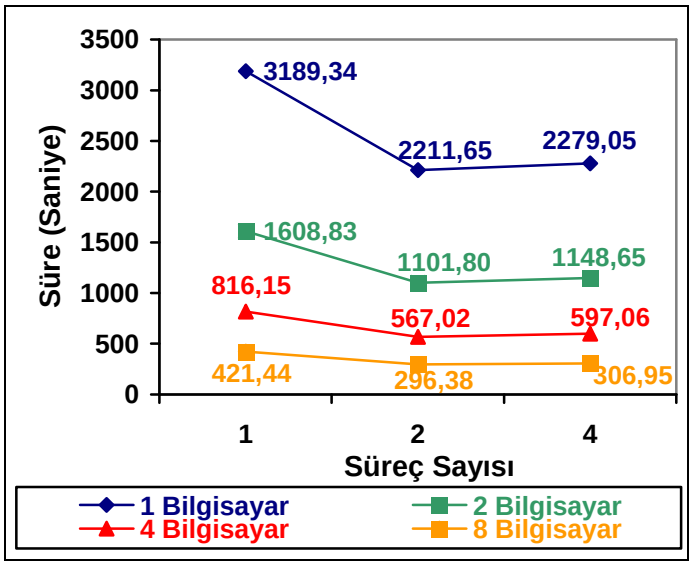
Şekil 3.10. 256x256x256 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

Büyük boyutlu zamanla değişen verilerde algoritmanın etkinliğini ölçmek için 512x512x512 voksellik Mandelbrot kümesi hesaplanarak görselleştirilmiştir. Bu durumu yansıtan performans değerleri Şekil 3.11 ve Şekil 3.12'de verilmiştir.



Şekil 3.11. 512x512x512 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

512x512x512 voksellik kümenin görselleştirilmesi statik yük dağıtımıyla elde edilen sonuçlarla karşılaştırıldığında belirgin şekilde daha iyi performans değerleri elde edilmiştir. Bütün diğer şartların aynı kaldığı düşünüldüğünde aradaki farkın dinamik yük dağıtımı algoritmasının daha dengeli yük dağıtımı yapmasından kaynaklandığı söylenebilir.



Şekil 3.12. 512x512x512 voksellik hesaplamanın süreç – süre grafiği

3.1.2. Hacim Dilim Sayısının Zamanla Değişen Hacimsel Verinin Görselleştirilmesindeki Etkisi

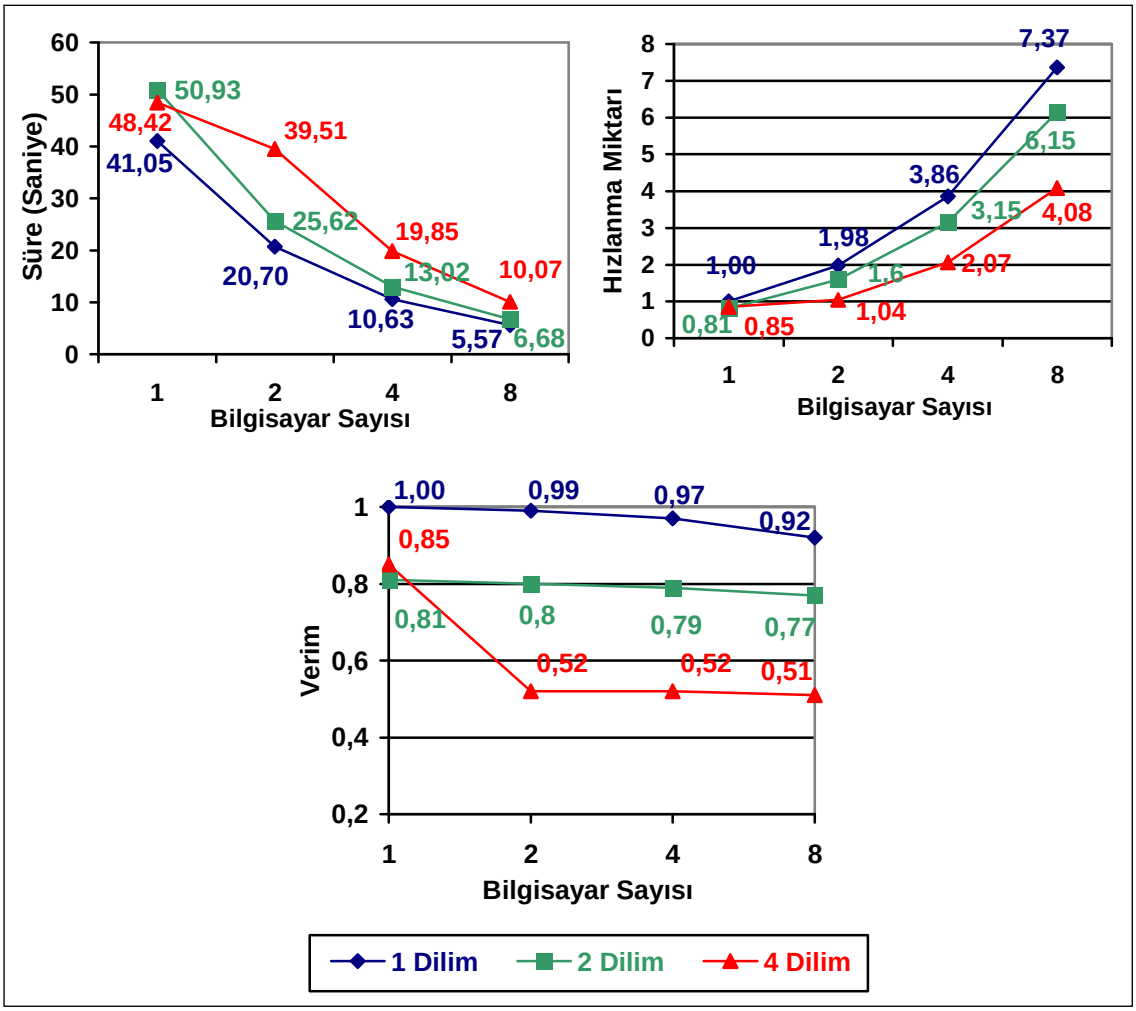
Algoritma farklarının daha iyi anlaşılabilmesi için üç boyutlu hacmi de parçalara bölerek hesaplamak gerekmektedir. Hesaplamalarda, genellikle iyi sonuç verdiği görülen bilgisayar başına iki adet süreç modeli kullanılması daha uygun olacaktır.

Daha önce yapılan görselleştirmeler tek dilimde hesaplandığından, hesaplamaların tek dilim için tekrar edilmesine gerek kalmamıştır. Bu nedenle karşılaştırma yapılırken daha önce iki süreç için elde edilen hesaplamalar kullanılmıştır.

3.1.2.1. Statik Yük Dağıtımı Kullanılarak Hesaplama ve Görselleştirme

Hacimsel olarak 128x128x128 vokselden oluşan Mandelbrot kümesi görselleştirilen ilk zamanla değişen hacimsel veridir. Bu duruma ait performans verileri Şekil 3.13'de verilmiştir.

Hacimler bölünerek elde edilen süreçler hacim bölünmeden elde edilen sürelerle karşılaştırıldığında belirgin bir yavaşlama olduğu görülmektedir. Aynı zamanda özellikle veri dört dilime ayrıldığında performans kazancı büyük oranda azalmaktadır.

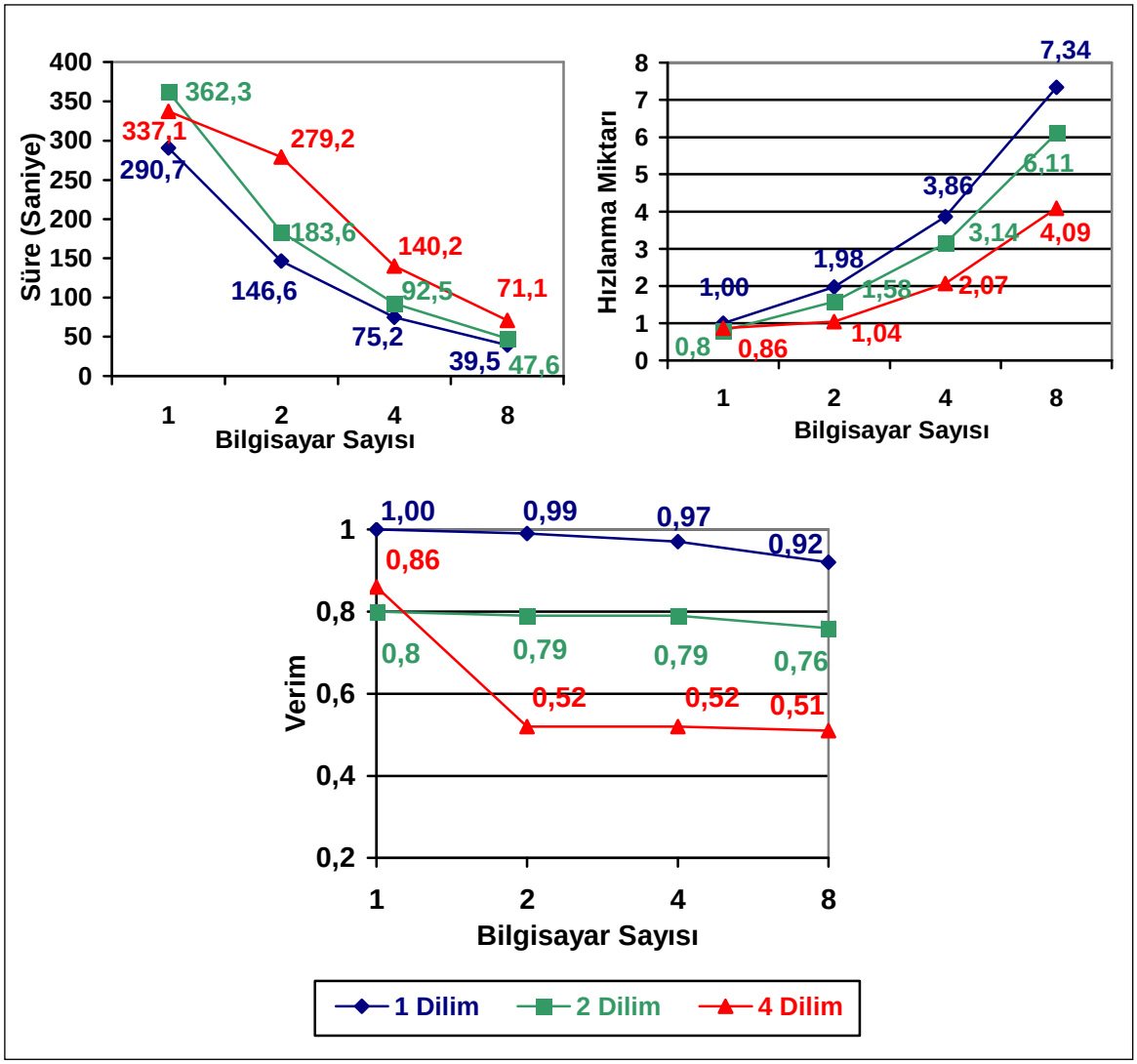


Şekil 3.13. 128x128x128 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

Dört dilime ayrılmış bir verinin tek bir bilgisayarda işlenmesi ek iş yükü dışında performansı etkilememektedir. Ancak bilgisayar sayısı arttıkça iş dengesiz dağılmış ve hızlanma miktarı azalmıştır.

Statik yük dengeleme algoritması bütün deneylerde büyük performans kayıplarına neden olmuştur. Bunun nedeni dilimlere ayırma işlemiyle artan iş dengesizliği ve yüzey normali hesaplamak için kullanılacak hacmin ek olarak hesaplanmasıdır.

Orta ölçekli zamanla değişen hacimsel verilerin dilimlenerek hesaplanıp görselleştirilmesinin etkilerini inceleyebilmek için 256x256x256 vokselden oluşan Mandelbrot kümesi dilimlere ayrılarak hesaplanıp görselleştirilmiştir. Sonuçlar Şekil 3.14 ‘de verilmiştir.

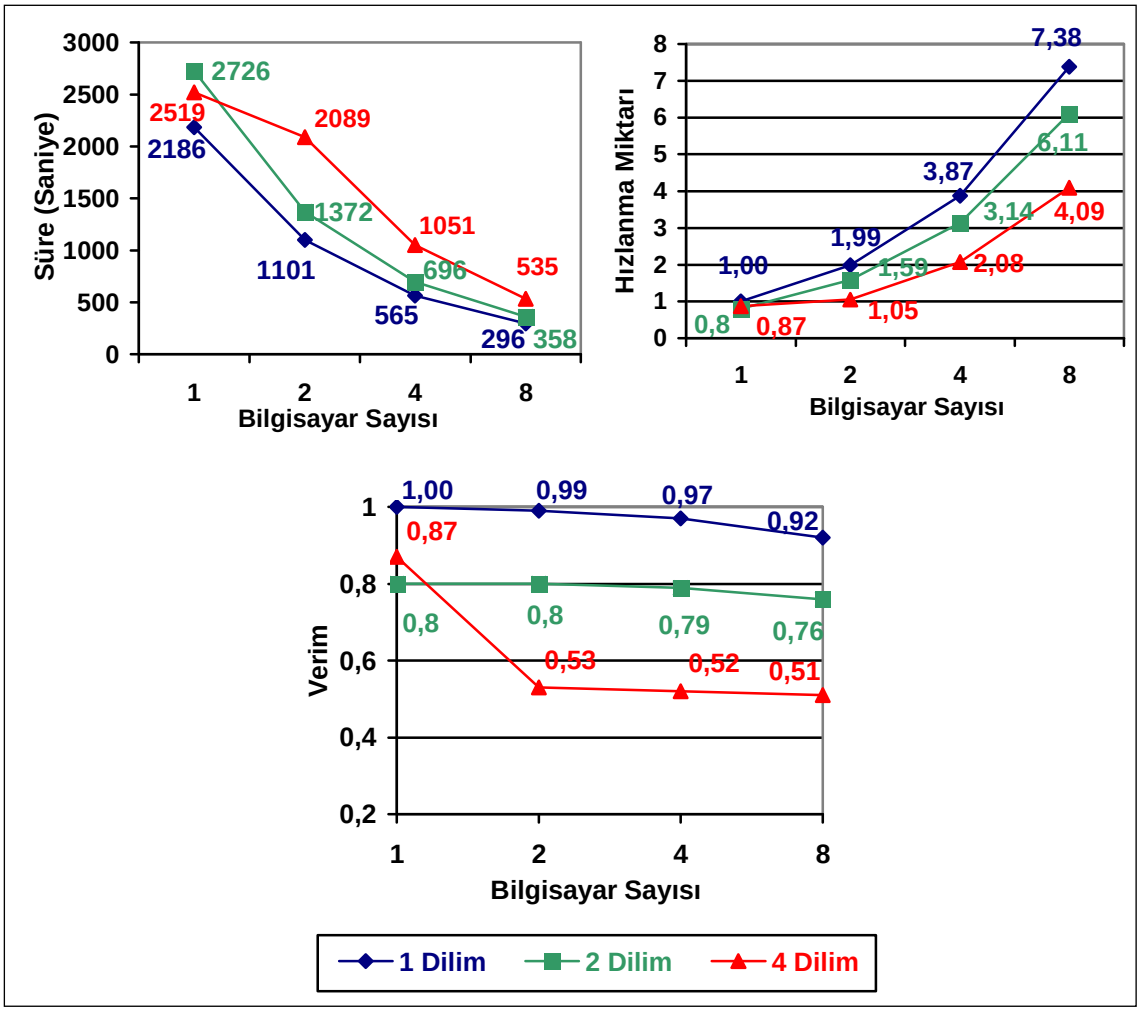


Şekil 3.14. 256x256x256 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

256x256x256 voksellik hacim dilimlenerek elde edilen performans sonuçları 128x128x128 voksellik görselleştirme deneyine benzerdir. Orta boyutlu bu veride algoritmanın etkisi çok daha belirgin olarak görülebilmektedir.

Son olarak daha büyük boyutlu 512x512x512 voksellik Mandelbrot kümesi hesaplanarak görselleştirilmiştir. Sonuçlar Şekil 3.15 ’de verilmiştir.

512x512x512 voksellik Mandelbrot kümesinin dilimlere ayrılarak görselleştirilmesi deneyinde, daha önce gerçekleştirilen 128x128x128 ve 256x256x256 voksellik hacimlerin hesaplanarak görselleştirilmesi deneylerinde elde edilenlere benzer sonuçlar elde edilmiştir.



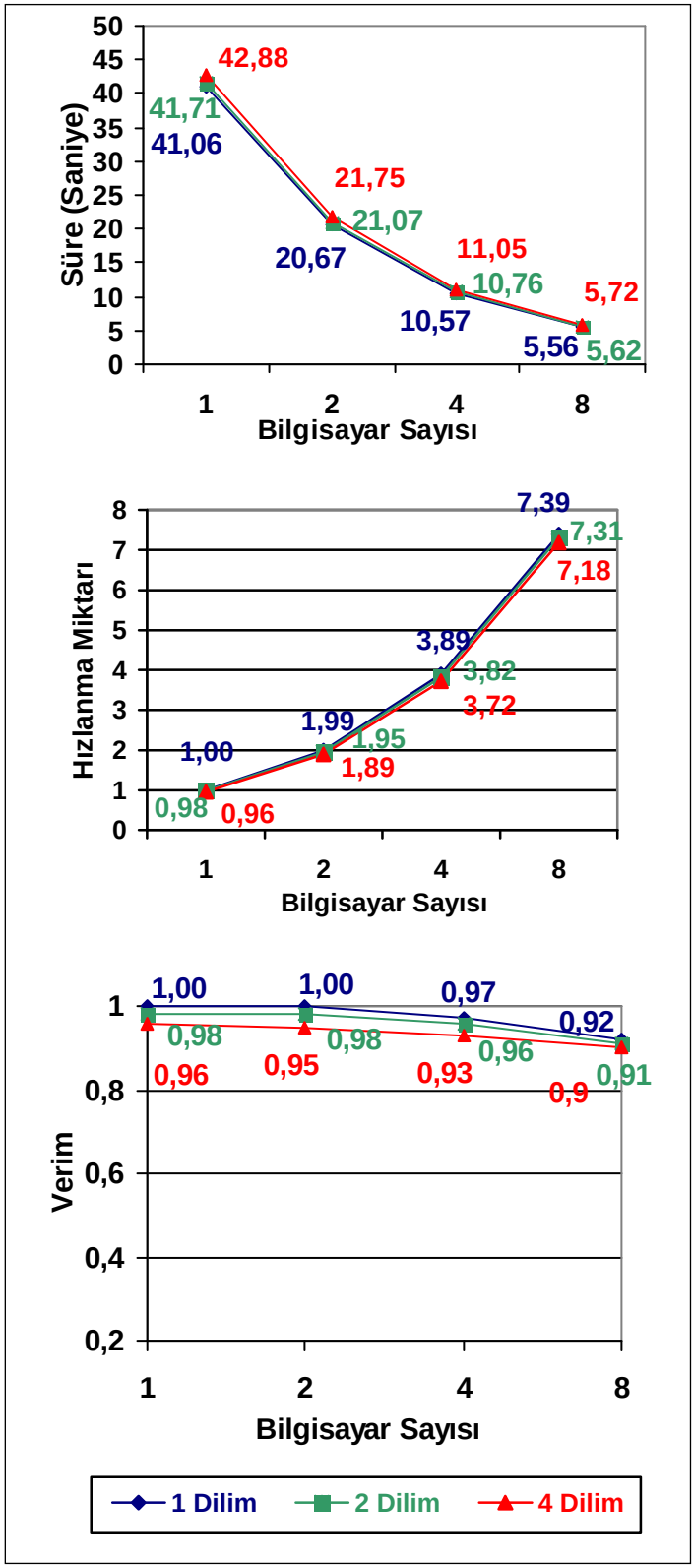
Şekil 3.15. 512x512x512 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

3.1.2.2. Dinamik Yük Dağıtımı Kullanılarak Hesaplama ve Görselleştirme

Dinamik yük dengeleme algoritmasının sonuçlarına bakılarak performans kaybının ne kadarının yük dengesizliğinden kaynaklandığı anlaşılabilir. Bu nedenle deneylerin dinamik yük dengeleme için gerçekleştirilmesi gerekmektedir.

İlk deneylerde küçük ölçekli 128x128x128 voksellik Mandelbrot kümesinin dilimlere ayrılarak hesaplanıp görselleştirilmesi incelenmiştir. Sonuçlar hacim dilim sayısının bir, iki ve dört olması durumları için Şekil 3.16'da verilmiştir.

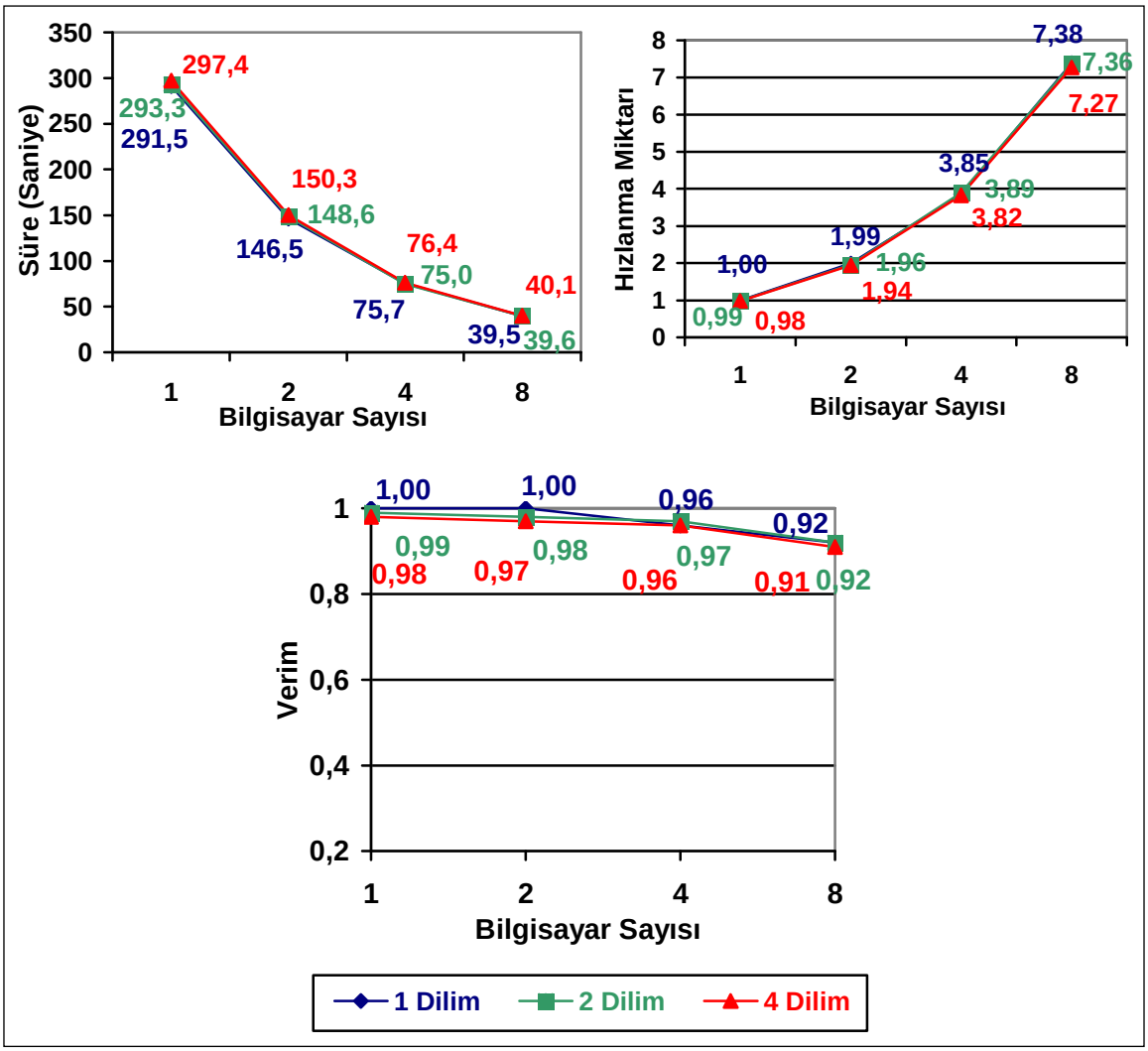
Dinamik yük dengeleme algoritması 128x128x128 voksellik hacmi yaklaşık olarak tek dilimlik hesap sürelerinde görselleştirmiştir. Ancak az miktarda da olsa yavaşlama görülmektedir. Bu yavaşlamanın temel nedeni ek hesaplama ve haberleşme sürelerine bağlanabilir.



Şekil 3.16. 128x128x128 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

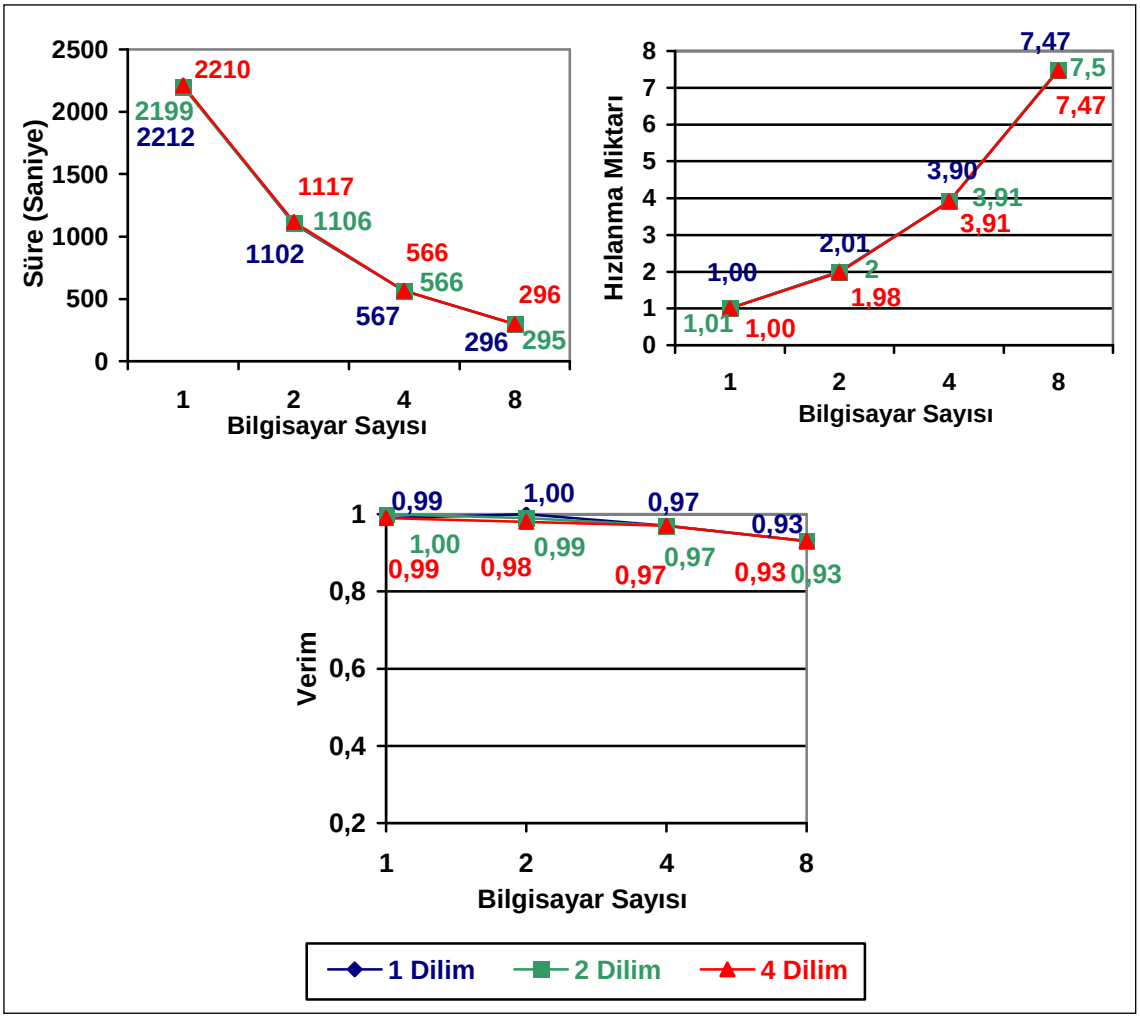
Bu bölümde ikinci olarak 256x256x256 voksellik Mandelbrot kümesi hesaplanarak görselleştirilmiştir. Bu durumu yansıtan sonuçlar hacim dilim sayıları bir, iki ve dört için , sırasıyla , Şekil 3.17'de verilmiştir.

Grafiklerden de görüldüğü gibi 256x256x256 voksellik kümenin görselleştirilmesi az miktarda zaman artışına neden olmaktadır. Bu zaman artışının nedeni de artan haberleşme ve daha önemlisi hesaplanan ek bölgelerdir.



Şekil 3.17. 256x256x256 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

Son olarak 512x512x512 voksellik Mandelbrot kümesi görselleştirilmiştir. Sonuçlar Şekil 3.18'de görülmektedir.

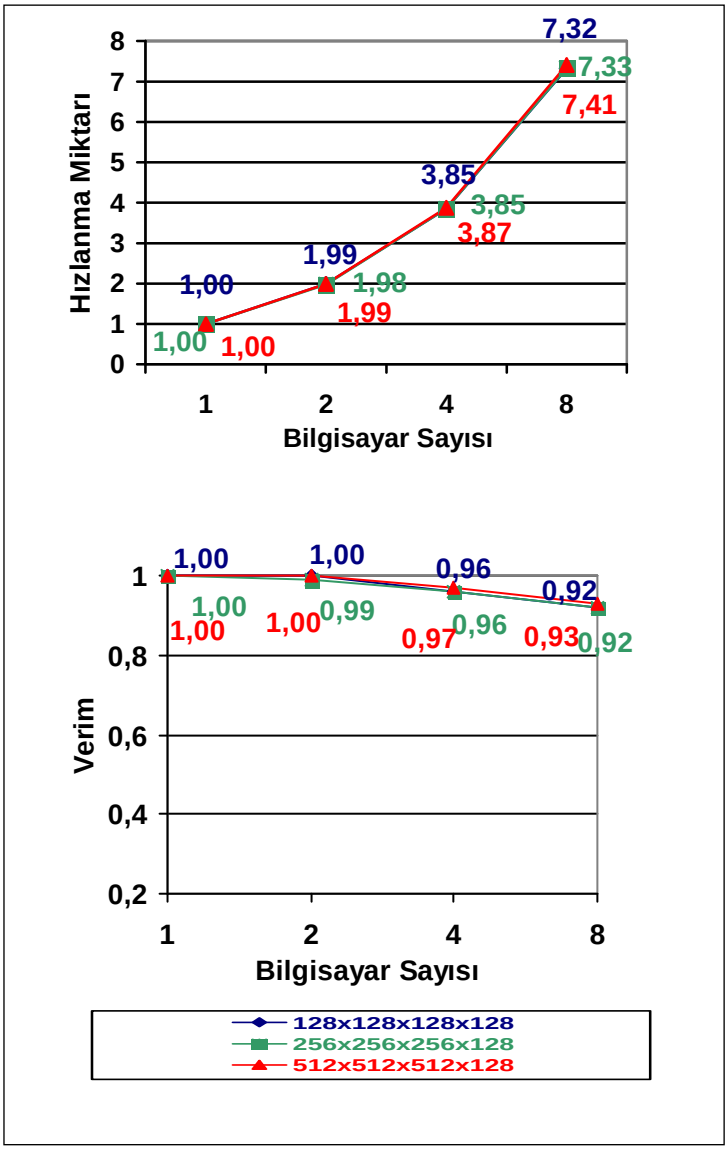


Şekil 3.18. 512x512x512 voksellik hesaplamanın süre, hızlanma ve verim grafikleri

512x512x512 voksellik küme dilimlenerek hesaplandığında performansta belirgin bir fark gözlenmemekte, hatta hacmin iki dilime ayrılarak görselleştirilmesinde az da olsa hızlanma görülmektedir. Bu hızlanmanın nedeni daha dengeli yük dağılımıyla birlikte veri miktarı büyüdükçe yüzey normali hesaplanacak hacim için yapılacak hesaplama miktarının oransal olarak azalmasıdır.

3.1.3. En İyi Performans Değerlerinin İncelenmesi

Hacim dilim sayısı ve bilgisayar başına düşen süreç sayısı değiştirilerek yapılan deneylerden, her hacim ve bilgisayar kategorisi için, elde edilen en iyi sonuçlar incelenerek verim ve hızlanma miktarları elde edilmiştir (Şekil 3.19).



Şekil 3.19. En iyi performans değerleriyle elde edilen hızlanma ve verim grafikleri

En iyi değerler incelendiğinde 512x512x512 voksellik zamanla değişen verinin hesaplanmasında verim ve hızlanma miktarı, daha küçük hacimlerle kıyaslandığında, bir miktar artış göstermektedir.

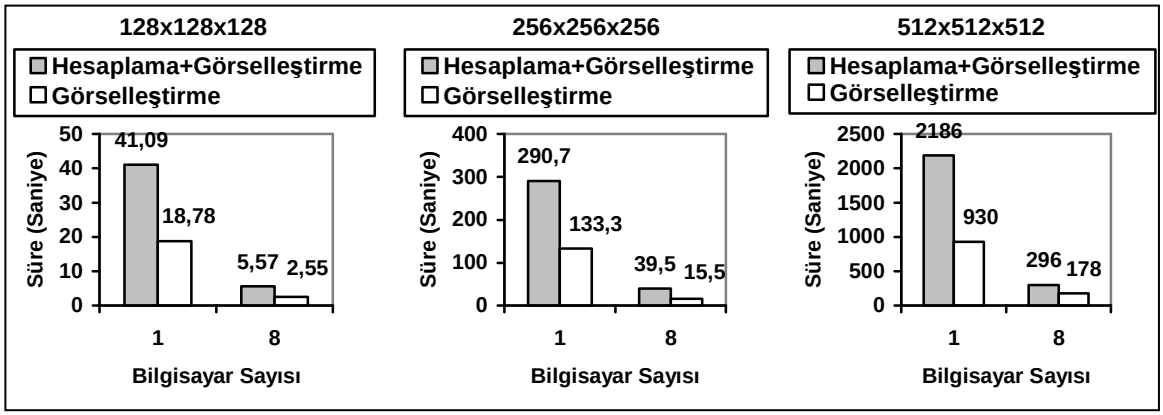
3.1.4. Zamanla Değişen Hacimsel Verinin Sabit Diskten Okunarak Görselleştirilmesi

Uygulama performansını değerlendirmenin bir diğer yolu da Mandelbrot kümesini sabit diskten okuyarak görselleştirmektir. Böylece Mandelbrot kümesinin görselleştirilmesi

ve hesaplanması için harcanan süreler bir miktar belirginleşecektir. Ancak bu durumda sabit disk erişimlerinin de performansı etkileyeceği göz önüne alınmalıdır.

Deneyler 128x128x128, 256x256x256 ve 512x512x512 voksellik Mandelbrot kümeleri dilimlere ayrılmadan, bilgisayar başına iki süreç düşecek şekilde 1 ve 8 bilgisayar üzerinde, statik ve dinamik yük dağıtma yaklaşımlarıyla gerçekleştirilmiştir.

İlk olarak statik yük dengeleme algoritması 128x128x128, 256x256x256 ve 512x512x512 voksellik Mandelbrot kümesi sadece görselleştirilmiş ve hem hesaplanmış hem de görselleştirilmiş olan sonuçlarla karşılaştırılmıştır. Elde edilen sonuçlar Şekil 3.20 ‘de verilmiştir.

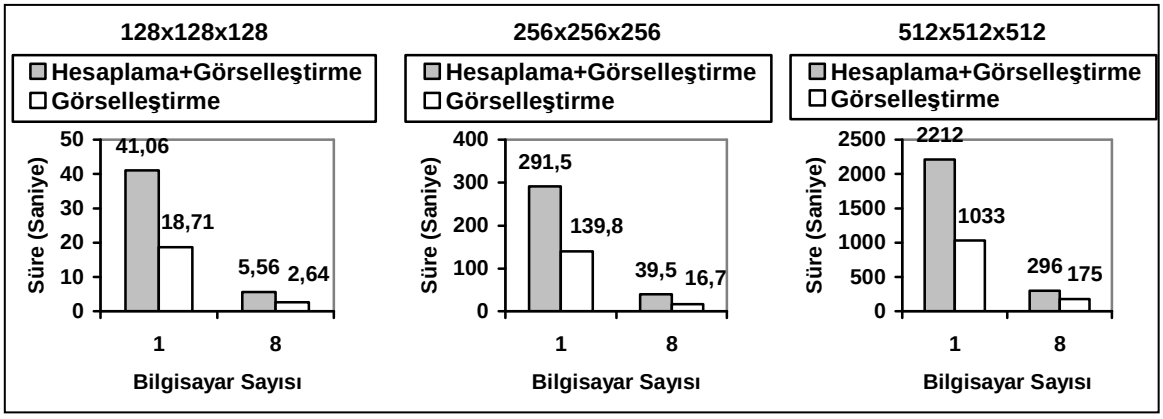


Şekil 3.20. Statik yük dağıtımıyla zamanla değişen hacimsel verinin sabit diskten okunarak görselleştirilmesi

Tek bilgisayarda elde edilen sonuçlar Mandelbrot kümesinin hesaplanmasının toplam sürenin yaklaşık olarak % 53-58‘i kadar olduğunu göstermektedir. Ancak sekiz bilgisayar kullanılarak yapılan deneylerde sonuçlar farklılık göstermiştir. 128x128x128 voksellik hacimde sabit diskten okunarak görselleştirme ve hesaplanarak görselleştirme yaklaşımlarında hızlanma miktarları oransal olarak sabit kalırken, 256x256x256 voksellik hacimde doğrusal hızlanmadan daha iyi bir hızlanma görülmüştür. Ancak 512x512x512 voksellik kümenin görselleştirilmesi hızlanama olarak düşük performans kaydetmiş ve kümenin hesaplama süresinin görselleştirme süresine oranı düşmüştür.

Bu dalgalanmaların en önemli nedeni sabit disk erişimidir. Sabit disk erişimleri sistemler arasında ve erişim düzenlerine bağlı olarak değişmektedir. Bunun sonucunda bazı erişimler daha yavaş, bazı erişimler ise daha hızlı olmaktadır.

İkinci olarak aynı deneyler dinamik yük dağıtımı yaklaşımı için tekrarlanmış ve sonuçlar yine dinamik yük dağıtımı kullanılarak hesaplanıp görselleştirilen Mandelbrot kümesi deneyleriyle karşılaştırılmıştır. Elde edilen sonuçlar Şekil 3.21' de sunulmuştur.



Şekil 3.21. Dinamik yük dağıtımıyla zamanla değişen hacimsel verinin sabit diskten okunarak görselleştirilmesi

Dinamik yük dağıtımı yaklaşımı da statik yük dağıtımı yaklaşımına benzer sonuçlar üretmiştir.

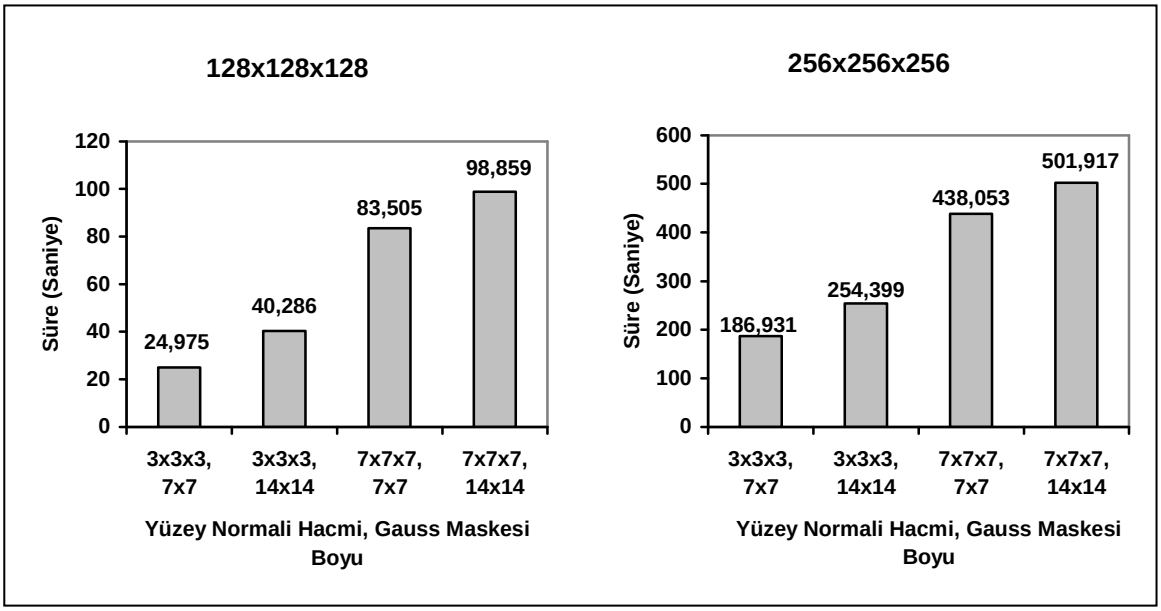
Zamanla değişen hacimsel verilerin sabit diskten okunarak görselleştirilmesindeki sabit disk erişimlerinin etkisi yapılan bu deneylerde daha net bir şekilde görülmüştür. En yavaş sistemle en hızlı sistem arasındaki fark verinin sabit disk üzerindeki dağılımına bağlı olarak yaklaşık %30 kadar olabilmektedir. Bu durum göz önüne alındığında sonuçların genel disk erişiminden ayrı tutularak değerlendirilmemesi gerektiği söylenebilir.

3.1.5. Paralel Splatting Algoritmasında Kullanılan Parametrelerin Görselleştirme Performansına Etkileri

Deneylerde son olarak görselleştirme parametrelerinin görselleştirme zamanını nasıl etkilediği incelenmiştir.

Paralel splatting algoritmasında kullanılan parametrelerinden Gauss maskesi büyüklüğü ve yüzey normali hesaplama hacminin görselleştirme algoritması üzerindeki performans etkilerini belirlemek için 128x128x128, 256x256x256 voksellik Mandelbrot kümeleri sabit diskten okunup görselleştirilmiştir. Deneyler Gauss maskesi boyu 7x7, 15x15 ve yüzey normali bulma hacmi 3x3x3 ve 7x7x7 olacak şekilde tek bilgisayarda tek

süreç koşularak yapılmıştır. Yük dengeleyici yaklaşım olarak dinamik yük dağıtımı kullanılmıştır. Elde edilen Performans değerleri Şekil 3.22'de sunulmuştur.



Şekil 3.22. 128x128x128 ve 256x256x256 voksellik kümelerin değişik Gauss maskesi ve yüzey normali hesaplama hacmi parametreleriyle görselleştirilmesi

128x128x128 voksellik hacmin görselleştirilmesinde, parametre değişikliklerinin çok daha belirgin etkileri olduğu söylenebilir. 256x256x256 voksellik hacimde en kısa görselleştirme süresi ile en uzun görselleştirme süresi arasında 2.7 katlık süre farkı bulunurken, 128x128x128 voksellik hacmin görselleştirilmesinde bu süre farkı 4 kat kadar olmaktadır. Bu da görselleştirme hacmi büyüdükçe hesaplama parametrelerinin etkisinin azaldığını göstermektedir. Bu nedenle parametreler arasındaki fark hacim büyüdükçe oransal olarak azalmıştır.

4. SONUÇLAR

Yapılan çalışmada zamanla değişen hacimsel veriler dağıtık sistemler yardımıyla yük dengelemeli olarak görselleştirilmiştir. Görselleştirme tekniği olarak paralelleştirilmiş splatting tekniği kullanılmıştır. Splatting tekniğinin paralel bir şekilde gerçeklenmesi ek işlemler gerektirmektedir.

Bu çalışmada çeşitli yük dengeleme teknikleri yanında görselleştirme parametrelerinin de görselleştirme işlemi üzerindeki etkileri incelenmiştir.

Gerçeklenen iki yük dengeleme algoritmasının, zamanla değişen Mandelbrot kümesi parçalara ayrılmadığında, benzer hızlanma miktarlarına sahip olduğu görülmüştür. Ancak veriler dilimlere ayrıldığında statik yük dengelemenin dinamik yük dengelemeye oranla çok yavaş kaldığı ve şartlara bağlı olarak çok değişik sonuçlar verebileceği görülmüştür.

Elde edilen bir diğer sonuç HyperThreading gibi teknolojilerin paralelselleştirilen algoritmalarda önemli ölçüde hız avantajı sağladığıdır.

Sadece görselleştirme işleminde ise sabit disk performansının çok önemli olduğu sonucuna varılmıştır. Yapılan deneylerde aynı özelliklere sahip iki bilgisayarın performans farkının %30 sevilerinde olabileceği görülmüştür. Bu nedenle sabit diskteki verilerin birleştirilerek erişim için optimize edilmesi gerekmektedir.

Bir diğer incelenen konu ise görselleştirme parametrelerinin (Gauss maskesi, yüzey normali hesaplama hacmi) işlem miktarına olan katkılarıdır. Görselleştirme parametrelerinin büyük seçilmesi durumunda performansın önemli ölçüde düşebileceği görülmüştür. Bu yavaşlama küçük boyutlu verilerde azami olmaktadır. Veri boyu büyüdükçe parametrelerin etkisi azalmaktadır.

5. ÖNERİLER

Görselleştirme algoritmasında kullanılan bütün parametrelerin karşılaştırmalı olarak incelenmesi gerekmektedir. Uzun süre alan bir çok deneyin yapılmasını gerektiren bu çalışmaların sadece bir kısmı tamamlanabilmiştir. Bu nedenle göreceli karşılaştırmalar yapılarak çeşitli sonuçlara varılması daha uygun olacaktır.

Görüntüleri zamanda dengeli olarak üretebilecek dinamik yük dengeleme algoritmasını kullanan yük dengeleme algoritmaları geliştirilmelidir. Dağıtılmış dinamik yük dengelemenin ne gibi imkanlar sağlayabileceği ayrıca incelenmelidir.

Diğer görselleştirme teknikleri kullanılarak zamanla değişen hacimsel verilerin görselleştirilmesi için en uygun çözümün hangi yöntem olduğunun incelenmesi gerekmektedir.

Cisim uzayını bölümleme ve görüntü uzayını bölümleme yaklaşımlarının, incelerek yük dengeleme ve haberleşmede ne gibi etkileri olduğu gösterilmelidir.

Büyük ölçekli sistemlerde üretilen büyük miktardaki verinin ağ üzerinde dağıtık olarak depolanması yada tek bir merkezi veri depolama biriminin kullanılması gerekmektedir. Bu gibi durumlarda veri sıkıştırma algoritmalarının, ağ haberleşmeleri göz önüne alınarak, kullanılması durumunda performansa etkilerinin neler olabileceği incelenmelidir.

5. KAYNAKLAR

1. Flynn, M.J., Very High-Speed Computing Systems, Proceeding of the IEEE, 54,12 (1966) 1901-1909.

2. Chalmers A. ve Tidmus J., Practical Parallel Processing: An Introduction to Problem Solving in Parallel, International Thomson Computer Pres, London, 1996.

3. Wilkinson, B. ve Allen M., Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 1st Edition, Prentice Hall, New Jersey, 1998.

4. http://www.csm.ornl.gov/pvm/pvm_home.html, PVM Parallel Virtual Machine, 15 Haziran 2005.

5. Lorensen, W.E. ve Cline, H.E., Marching Cubes: A High Resolution 3D Surface Construction Algorithm, Computer Graphics, 21, 4 (1987) 163-169.

6. Guarnieri,A., Vettore, A. ve Pontin, M., A Volumetric Approach for 3D Surface Reconstruction, CIPA 2005 XX. International Symposium, Eylül 2005, Torino, Bildiriler Kitabı, 831-836.

7. Ho, C., Wu, F., Chen, B., Chuang, Y., ve Ouhyoung, M., Cubical Marching Squares: Adaptive Feature Preserving Surface Extraction from Volume Data, Computer Graphics Forum, 24, 3 (2005) 537-545.

8. http://www.essi.fr/~lingrand/MarchingCubes/algo.html, The Marching Cubes, 16 Nisan 2006

9. Shekhar, R., Fayyad, E., Yagel, R. ve Cornhill, J.F., Octree-Based Decimation of Marching Cubes Surfaces, Seventh IEEE Visualization 1996 (VIS'96), Kasım1996, California, Bildiriler Kitabı, 335-342.

10. Meißner, M., Huang, J., Bartz, D., Mueller, K. ve Crawfis, R., A Practical Evaluation of Popular Volume Rendering Algorithms, The 2000 IEEE Symposium on Volume Visualization, Ekim 2000, Utah, Bildiriler Kitabı, 81-90.

11. Marmitt, G., Kleer, A., Wald, I., Friedrich, H., ve Slusallek, P., Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing, Vision, Modeling and Visualization 2004 (VMV-04), Kasım 2004, California, Bildiriler Kitabı, 429-435.

12. Agron, P., Isosurface Viewing in Volume Rendering, Course paper for CSE564 (Scientific Visualization), Computer Science Department, State University at Stony Brook, İlkbahar 2002.

13. Schwarze, J., Graphics Gems, Academic Pres, San Diego,1990.

14. Levoy, M., Display of Surfaces from Volume Data. IEEE Computer Graphics and Applications, 8, 3 (1988) 29–37.

15. Yagel, R. ve Shi, Z., Accelerating Volume Animation by Space-Leaping, IEEE Visualization 1993 Conference, Ekim 1993, California, Bildiriler Kitabı, 62–69.

16. Levoy, M., Efficient Ray Tracing of Volume Data, ACM Trans. on Computer Graphics, 9, 3 (1990) 245–261.

17. Ke, H. R. ve Cahng, R. C., Ray Cast Volume Rendering Accelerated by Incremental Trilinear Interpolation and Cell Templates, Visual Computer,11,11 (1995) 297-308.

18. Lakare, S., Ray Based Exploration of Volumetric Data, Doktora Tezi, Stony Brook University, New York, 2004.

19. Westover, L., Footprint Evaluation for Volume Rendering, ACM SIGGRAPH Computer Graphics, 24, 4 (1990) 367-376.

20. Zwicker, M., Pfister, H., Baar, J. ve Gross, M., EWA volume splatting, The Conference on Visualization '01 IEEE, Ekim 2001, California, Bildiriler Kitabı, 29-36.

21. Chen, W., Ren, L., Zwicker, M., ve Pfister, H., Hardware-Accelerated Adaptive EWA Volume Splatting, IEEE Visualization 2004, Eylül 2004, Texas, Bildiriler Kitabı, 67-74.

22. Zhang, C. ve Crawfis, R., Volumetric Shadows Using Splatting, The Conference on Visualization 2002 IEEE, Eylül 2002, Massachusetts, Bildiriler Kitabı, 85-92 .

23. Neophytou, N. ve Mueller, K., Post-Convolved Splatting, Symposium on Data Visualization, Mayıs 2003, Fransa, Bildiriler Kitabı, 223-230.

24. Mueller, K. ve Yagel, R., Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach, The 7th Conference on Visualization '96, Eylül 1996, California, Bildiriler Kitabı, 65-72.

25. Chaudhary, G., Ramaswamy, L.ve Farias, R., RZSweep: A New Hardware-Assisted Volume-Rendering Technique for Regular Datasets, XVI. Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003), Eylül 2003, Brezilya, Bildiriler Kitabı, 99-106.

26. Neophytou, N. ve Mueller, K., GPU Accelerated Image Aligned Splatting, International Workshop on Volume Graphics, Haziran 2005, ABD, Bildiriler Kitabı, 197-205.

27. Mueller, K. ve Crawfis, R., Eliminating Popping Artifacts in Sheet Buffer-Based Splatting, Ninth IEEE Visualization, Ekim1998, ABD, Bildiriler Kitabı, 239–246.

28. Lacroute, P., Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation, Technical Report, Stanford University, 1995.

29. Rezk, C. ve Erlangen, S., Volume Rendering Techniques for General Purpose Graphics Hardware, Doktora Tezi, Der Technischen Fakult, Universit¨at Erlangen-Nurnberg zur Erlangung des Grades DOKTOR-INGENIEUR, Almanya, 2001.

30. Phong, B. T., Illumination for Computer Generated Pictures, Communications of the ACM, 18, 6 (1975) 311-317.

31. http://en.wikipedia.org/wiki/Phong_reflection_model, Phong Reflection Model, 21 Şubat 2006.

32. http://mathworld.wolfram.com/Quaternion.html, Quaternion, 20 Haziran 2005.

33. http://www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-2.0/node28.htm, Version 1.2 of MPI, 30 Mart 2005.

34. Kniss, J., McCormick, P., McPherson, A., Ahrens, J., Painter, J., Keahey, A. ve Hansen, C.. Interactive Texture-Based Volume Rendering for Large Data Sets. IEEE Computer Graph. Applications, 21, 4 (2001) 52-61.

35. Chen, Y., Cohen, J. ve Kumar, S., On the Visualization of Time-Varying Structured Grids Using a 3D Warp Texture, The Conference on Visualization '04, Eylül 2004, Texas, Bildiriler Kitabı, 17-18.

36. Ma, K. ve Parker, S., Massively Parallel Software Rendering for Visualizing Large-Scale Data Sets, IEEE Computer Graphics and Applications, 21, 4 (2001) 72-83.

37. Liao, S., Lin, C., Chung, Y. ve Lai, J.Z.C., A Differential Volume Rendering Method with Second-Order Difference for Time-Varying Volume Data, Journal of Visual Languages and Computing,14, (2003) 233–254.

38. Tost, D., Grau, S., Ferré, M. ve Puig, A., Ray-Casting Time-Varying Volume Data Sets with Frame-to-Frame Coherence, Conference of Visualization and Data Analysis, Ocak 2006, ABD, Bildiriler Kitabı, Cilt 6060, 1-10 .

39. Park, S., Bajaj, C., ve Ihm, I., Visualization of Very Large Oceanography Time-Varying Volume Datasets, International Conference on Computational Science, Haziran 2004, Polonya, Bildiriler Kitabı, 419-426.

40. Kusurkar, Y. ve D. Silver, Visualizing Time Varying Distributed Datasets, Visualization Development Environments 2000, Nisan 2000, New Jersey, Bildiriler Kitabı, 97-104.

41. Reinhard, E., Hansen, C. ve Parker, S., Interactive Ray Tracing of Time Varying Data, The Fourth Eurographics Workshop on Parallel Graphics and Visualization, Eylül 2002, Almanya, ACM International Conference Proceeding Series Bildiriler Kitabı, Cilt 20, 77-82.

42. Li, P.P., Whitman, S., Mendoza, R. ve Tsaio, J., ParVox: A Parallel Splatting Volume Rendering System for Distributed Visualization, IEEE Symposium on Parallel Rendering, Ekim 1997, ABD, Bildiriler Kitabı, 7-14.

43. Çakır, Ö., Yerel Ağdaki Kişisel Bilgisayarlarla Paralel Işin İzleme, Yüksek Lisans Tezi, KTÜ Fen Bil. Enst., Trabzon, 2004

44. Ma, K., Painter, J.S., Hansen, C D. ve Krogh, M.F., Parallel Volume Rendering Using Binary-Swap Compositing, IEEE Computer Graphics and Applications, 14, 4 (1994) 59-68.

ÖZGEÇMİŞ

1981 yılında İngiltere'nin Leicester şehrinde doğmuştur. İlköğrenimini Trabzon Mimar Sinan İlkokulu, orta ve lise öğrenimini Kanuni Anadolu Lisesi'nde tamamladı. 1999 yılında Karadeniz Teknik Üniversitesi Mühendislik Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümünde lisans eğitimine başladı ve 2003 yılında bu bölümden sınıf birincisi olarak mezun oldu. Aynı yıl Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı'nda Yüksek Lisans Programı'na başladı. 2003 yılından beri aynı enstitüde araştırma görevlisi olarak çalışmaktadır.