TÜRK HAVA KURUMU ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Donanımsal Mozaikleştirme Yöntemiyle Arazi Görselleştirmede Noktasal Görünürlük Testiyle Detay Seviyesi Belirleme

YÜKSEK LİSANS TEZİ

Mehmet Ömer Özek

Elektrik ve Bilgisayar Mühendisliği Anabilim Dalı

Elektrik ve Bilgisayar Mühendisliği Programı

EYLÜL 2016

TÜRK HAVA KURUMU ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Donanımsal Mozaikleştirme Yöntemiyle Arazi Görselleştirmede Noktasal Görünürlük Testiyle Detay Seviyesi Belirleme

YÜKSEK LİSANS TEZİ

Mehmet Ömer Özek
1403610006

Elektrik ve Bilgisayar Mühendisliği Anabilim Dalı

Elektrik ve Bilgisayar Mühendisliği Programı

Tez Danışmanı : Yrd. Doç. Dr. Engin Demir

Türk Hava Kurumu Üniversitesi Fen Bilimleri Enstitüsü'nün 1403610006 numaralı Yüksek Lisans öğrencisi, Mehmet Ömer Özek, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı Donanımsal Mozaikleştirme Yöntemiyle Arazi Görselleştirmede Noktasal Görünürlük Testiyle Detay Seviyesi Belirleme başlıklı tezini, aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : Yrd. Doç. Dr. Engin DEMİR .....
Türk Hava Kurumu Üniversitesi

Jüri Üyeleri : Yrd. Doç. Dr. Engin DEMİR .....
Türk Hava Kurumu Üniversitesi

Yrd. Doç. Dr. Erhan MENGÜŞOĞLU .....
Türk Hava Kurumu Üniversitesi

Yrd. Doç. Dr. Kasım Murat KARAKAYA .....
Atılım Üniversitesi

Tez Savunma Tarihi : 7 Eylül 2016

TÜRK HAVA KURUMU ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜ'NE

Yüksek Lisans Tezi olarak sunduğum, Donanımsal Mozaikleştirme Yöntemiyle Arazi Görselleştirmede Noktasal Görünürlük Testiyle Detay Seviyesi Belirleme adlı çalışmamın, tarafımdan akademik etik ve kurallara aykırı düşecek bir yardıma başvurmaksızın yazıldığını ve yararlandığım kaynakların kaynakçada gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve bunu onurumla doğrularım.

07.09.2016
Mehmet Ömer Özek

TEŞEKKÜRLER

Yüksek Lisans tez çalışma sürecinde beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan desteğini ve yardımını hiçbir zaman esirgemeyen tez danışmanım değerli Yrd. Doç. Dr. Engin DEMİR'e teşekkürlerimi sunarım.

Hayatım boyunca her zaman yanımda olan, maddi ve manevi desteklerini hiçbir zaman esirgemeyen aileme teşekkürlerimi, sevgi ve saygılarımı sunarım.

Eylül, 2016 **Mehmet Ömer Özek**

İÇİNDEKİLER

TABLOLAR

ŞEKİLLER

TERİMLER

Application : Uygulama
Block : Blok
Buffer : Tampon
Building cListGPU : Merkezi İşlemci Birimi Listesi Sınıfı Oluşturma
Camera Position : Kamera Pozisyonu
CPU : Merkezi İşlemci Birimi
CUDA : Tümleşik Hesaplama Donanım Mimarisi
CUDA Array : CUDA Dizisi
CUDA Linear Memory : CUDA Sıralı Hafıza
CUDA Space : CUDA Alanı
CudaMemcpy : CUDA Hafıza Kopyalama
cuGLMapBufferObject : Tampon Nesnesini Eşleştir
cuGLRegisterBufferObject : Tampon Nesnesi Bağını Kur
cuGLUnregisterBufferObject : Tampon Nesnesi Bağını Kaldır
DEM : Sayısal Yükseklik Haritası
Depth Buffer : Derinlik Tamponu
Distance : Uzaklık
Domain Shader : Bölge Akış Programı
Execute Kernels : Çekirdekleri Çalıştırma
Fragment Shader : Ekran Noktası Akış Programı
Frame Buffer : Çizim Çerçevesi Tamponu
Frustum Culling : Bakış Hunisi Filtreleme
Generated Primitives : Oluşturulmuş Temel Bileşenler
Geometry Shader : Geometri Akış Programı
glReadPixels : Ekran Noktası Değerlerini Oku
GPU : Grafik İşlem Birimi
Hardware Tessellation : Donanımsal Mozaikleştirme

Hull Shader : Kabuk Akış Programı
Hysteresis : Histerezis
Image Plane Tiling : Resim Düzlem Bölümleme
Inner Tessellation Level : İç Mozaikleştirme Katsayısı
Input Patch : Girdi Parçaları
Instances Of Objects : Nesnelerin Örnekleri
LOD : Detay Seviyesi
LOD Estimation : Detay Seviyesi Kestirimi
Navigation Speed : Gezinme Hızı
Non-uniform Terrain Patches : Dağınık Dağılımlı Arazi Parçaları
Object : Nesne
Observer : Gözlemci
Occlusion Query : Örtme Sorgusu
Offset : Referans Noktasından Uzaklık
OpenGL : Açık Grafik Kütüphanesi
Output Patch : Çıktı Parçaları
Output Vertices : Çıktı Köşe Noktaları
Outter Tessellation Level : Dış Mozaikleştirme Katsayısı
PBO : Ekran Noktası Tampon Nesnesi
Pixel Based LOD : Noktasal Görünürlük Tabanlı LOD
Primitive Assembly : Temel Birleştirici Akış Programı
Quad Tree : Dörtlü Ağaç
Radial LOD : Dairesel LOD
Rasterization : Resimleştirme Akış Programı
BGRA : Mavi Yeşil Kırmızı Saydamlık
Read Pixel BGRA : BGRA Formatında Ekran Noktası Okuma
Register PBO : PBO Bağını Kur
Render To Texture : Resim Dokusuna Çizim Yapmak
Rendering Space : Çizim Alanı
Root Node : Ata Ağaç Düğümü

Screen : Ekran

Shader Storage Buffer Object : Akış Programı Tampon Nesnesi

TCS : Mozaikleştirme Kontrol Akış Programı

Terrain Mesh Algorithm : Arazi Doku Algoritması

Terrain VBO : Arazi Köşe Noktası Tampon Nesnesi

TES : Mozaikleştirme Oluşum Akış Programı

Tessellation Control Shader : Mozaikleştirme Kontrol Akış Programı

Tessellation Evaluation Shader : Mozaikleştirme Oluşum Akış Programı

Tessellation Factor : Mozaikleştirme Çarpanı

Tessellation Level : Mozaikleştirme Seviyesi

Tessellator : Mozaikleştirici Akış Programı

Texture : Resim Dokusu

Thread : İş Parçası

Tile Load Map : Parça Yük Haritası

Tiles Requested : İstekte Bulunulan Parçaları

TPG : Mozaikleştirme Temel Oluşturucu

Uniform Terrain Patches : Eşit Dağılımlı Arazi Parçaları

Unique Id : Tekil Belirteç

Unregister PBO : PBO Bağını Kaldır

Vertex Shader : Köşe Noktası Akış Programı

View : Bakış

View Point : Bakış Noktası

ÖZET

Donanımsal Mozaikleştirme Yöntemiyle Arazi Görselleştirmede Noktasal Görünürlük Testiyle Detay Seviyesi Belirleme

ÖZEK, Mehmet Ömer
Yüksek Lisans, Elektrik ve Bilgisayar Mühendisliği Anabilim Dalı
Tez Danışmanı: Yrd. Doç. Dr. Engin DEMİR
Eylül-2016, 52 sayfa

Arazi görselleştirmenin en kolay yolu yükseklik haritasındaki her bir yüksekliğin üç boyutlu bir noktaya tekabül ettiği yüzeyin tümüyle çizilmesi yaklaşımıdır. Bu yaklaşım ile büyük yükseklik haritalarının gerçek zamanlı olarak çizilmesi mümkün olmamaktadır. Bu nedenle detay seviyesi (İng. LOD) algoritmaları yükseklik haritası verilerinin bir alt kümesini kullanarak ekrandaki üçgen dağılımlarının görece eşit olması için farklı karmaşıklıkta bir yüzey üretir.

Arazi görselleştirmesinde LOD belirlenmesi görselleştirmenin kalitesini ve çizimin hızını direk etkilemektedir. Düşük detay seviyesi çizim hızını artırırken görselleştirme kalitesini düşürmektedir. Buna karşılık yüksek LOD da tam tersi bir etki göstermektedir. Arazi görselleştirmesinin kaliteli olması ile çizim işleminin hızlı olması arasındaki dengeyi sağlamada LOD belirleme algoritması kritik bir önem kazanmaktadır.

Grafik İşlemci Birimi'nin (İng. GPU) hesaplama gücünün Merkezi İşlemci Birimi'ne (İng. CPU) göre çok daha hızlı gelişmesinden beri arazi görselleştirme algoritmaları ekran kartını mümkün olduğunca çok kullanır hale geldiler. Son zamanlarda geliştirilen algoritmalardan biri de ekran kartında donanımsal mozaikleştirme özelliğini kullanan sürekli görüş bağımlı uyumsal detay seviyesi algoritmasıdır. Bu çalışmada donanımsal mozaikleştirme kullanan sürekli görüş bağımlı uyumsal detay seviyesi algoritmasına ilave geliştirmeler yapılmıştır. İlk metot var olan bir yöntem olan ekran kartında çalışan örtme sorgusu yardımıyla nokta tabanlı detay seviyesinin belirlenmesidir. Arazi parçalarında ekrana çizilmeyenlerin

filtrelenmesi sayesinde çizim hızlandırılmıştır. İkinci metot ise bu çalışma kapsamında OpenGL ve CUDA beraber kullanılarak yeni olarak geliştirilen örtme sorgusu yardımıyla nokta tabanlı detay seviyesinin belirlenmesidir. Üçüncü metot ise ikinci metoda ilave olarak örtme sorgulamasındaki sorgulanan her bir arazi parçasının dört alt parçaya ayrılması yardımıyla nokta tabanlı detay seviyesi belirlenmesidir. Bu metotla ilk iki metottan farklı olarak sadece bir kısmı gözüken arazi parçalarının detay seviyesinin daha kaliteli bir şekilde belirlenmesinin sağlanması hedeflenmiştir.

Anahtar Kelimeler: Arazi Görselleştirme, Donanımsal Mozaikleştirme, Detay Seviyesi Belirleme, Örtme Sorgusu, Sürekli Görüş Bağımlı Uyumsal Detay Seviyesi

ABSTRACT

Determining Pixel-Based Level of Detail on Hardware Tessellated Terrain Rendering

ÖZEK, Mehmet Ömer

Master, Department of Electrical and Computer Engineering

Thessis Supervisor: Yrd. Doç. Dr. Engin, DEMİR

September–2016, 52 page

The easy way to render a terrain is the brute force approach that every height in the heightmap data is represented with one vertex in the surface. For bigger heightmap data, this is not feasible in real time terrain rendering. Thus level of detail (LOD) algorithms produce a mesh of different complexity so that the on-screen triangle distribution is relatively equal while using only a subset of heightmap data.

Determining LOD for terrain rendering directly affects rendering quality and performance. Low LOD reduces the visualization quality while increasing the drawing speed.However high LOD affects vice versa. Determining LOD plays critical role in balancing with visualization quality and drawing speed.

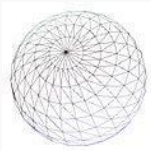
Since the GPU's processing power has been improving much faster than the CPU's. the terrain rendering algorithms have changed to use the graphics hardware as much as possible. One of the recently developed GPU-based LOD algorithm is Continuous View Dependent Adaptive LOD using hardware tessellation. In this context, Continuous View Dependent Adaptive LOD using hardware tessellation is enhanced using addinitial mothods. First method is determining pixel-based LOD using occlusion query. Because of hidden surface culling, render time is reduced. Second method is determining pixel-based LOD using occlusion query that newly developed by using OpenGL and CUDA interoperability. Third method is extension of second method that includes quad query for each terrain node. Third method aims to increase rendering quality of partial rendered terrain node.

Keywords: Terrain Rendering, Hardware Tessellation, Determining Level of Detail, Occlusion Query, Continuous View Dependent Adaptive Level of Detail

BİRİNCİ BÖLÜM

GİRİŞ

Arazi görselleştirilmesi coğrafi bilgi sistemleri, oyunlar ve eğitim simülatörleri gibi dış dünya görselleştirmesinin yapıldığı çoğu uygulamada kullanılmaktadır. Arazi görselleştirilmesinin hem detaylı hem de gerçek zamanlı yapılması kullanıcıların gerçeklik algısını olumlu yönde etkilemektedir. Arazinin tümünün detaylı çizilmesi performans kaygılarından dolayı mümkün olmadığından kullanıcının gerçeklik algısını minimum etkileyecek şekilde arazinin farklı detay seviyelerinde (İng. LOD) çizilmesi için algoritmalar geliştirilmiştir. Örnek olarak Şekil 1.1'de farklı detay seviyelerinde küre çizimleri gösterilmektedir. Ekranda çok uzakta olan bir küreyi 140 nokta ile çizerken çok yakında olan bir küreyi 5500 nokta ile çizerek gerçeklik algısını kaybetmeden daha performanslı çizim gerçekleştirilmiş olmaktadır.

Resim					
Nokta	~5500	~2880	~1580	670	140

Şekil 1.1: Farklı detay seviyelerinde küre çizimleri [26]

Çizimin hem kalitesini hem de performansını LOD algoritması direk etkilemektedir. Bu çalışmanın amacı arazi görselleştirilmesinin performansını ve

gerçeklik algısını direk etkileyen LOD algoritmasını geliştirmektir. Bu kapsamda literatürdeki ekran kartının donanımını etkin bir şekilde kullanan güncel bir LOD algoritmasının hem var olan hem de bu tez kapsamında yeni geliştirilen yöntemlerle performansının artırılması hedeflenmiştir.

Bu çalışmada son zamanlarda geliştirilen çoğunlukla ekran kartı üzerinde çalışan donanımsal mozaikleştirmeyi kullanan sürekli görüş bağımlı uyumsal detay seviyesi(Continuous View Dependent Adaptive LOD) algoritmasının üç farklı yöntem kullanılarak hızlandırılması ve çizim kalitesinin artırılması önerilmiştir.

Bu metotlardan ilki ekran kartı üzerindeki grafik işlemci birimi (İng. GPU) üzerinde paralel bir şekilde çalışan örtme sorgu (İng. Occlusion query) kullanılarak hem ekranda çizilmeyen arazi parçaları daha az parçaya ayrılmış ve çok düşük çözünürlükte çizilmesi sağlanmış. Hem de detay seviyesi belirlemede arazi parçalarının ekranda görünen nokta sayıları kullanarak gerçeklik algısı iyileştirilmiştir.

İkinci metotta ise örtme sorgusu ile aynı işi yapan, OpenGL ve CUDA beraber kullanılarak tamamen ekran kartı üzerinde çalışan yeni bir örtme sorgusu geliştirilmiştir. Bu sayede detay seviyesi belirleme için harcanan süre düşürülmüştür.

Üçüncü metot kapsamında ise ikinci metotta geliştirilen yeni örtme sorgusu, her bir arazi parçası için 4 farklı bölgesi için de ayrı sonuç hesaplanacak şekilde değiştirilmiştir. Böylelikle arazi parçasının bir kısmı gözükmesi durumunda bu parça için belirlenen detay seviyesinin daha gerçekçi olması sağlanmıştır.

İkinci bölümde temel bilgilere yer verilecektir. Üçüncü bölümde literatür özeti aktarılacaktır. Dördüncü bölümde materyal ve yöntem anlatılacaktır. Beşinci bölümde deneysel çalışmalar anlatılacaktır. Altıncı bölümde deneyler ve sonuçlar anlatılacaktır. Yedinci ve son bölümde ise sonuçlar ve önerilere yer verilecektir.

İKİNCİ BÖLÜM

TEMEL BİLGİLER

Temel bilgiler kısımda tez kapsamında kullanılan teknolojiler anlatılmıştır. Böylelikle tezde kullanılan teknolojilerin daha iyi anlaşılacağı öngörülmektedir. Şekillerin içerisindeki ingilizce kelimelerin anlamları terimler kısmında açıklanmaktadır.

2.1 OpenGL

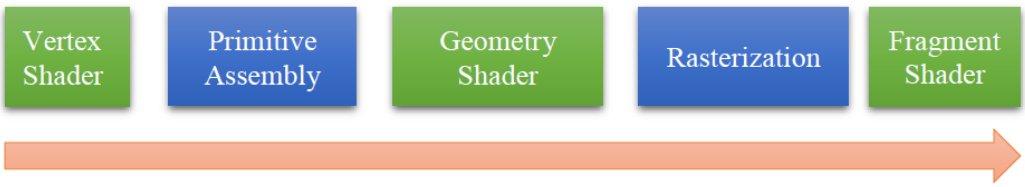
OpenGL (İngilizce: Open Graphics Library, Türkçe: Açık Grafik Kütüphanesi), gelişmiş donanım desteğini kullanarak hem iki hem de üç boyutlu grafikleri ekrana çizmek için kullanılan ücretsiz bir grafik uygulama geliştirme arabirimidir. Windows, Linux, MacOS ve Solaris gibi birçok işletim sisteminde yaygın olarak ve Playstation 3 başta olmak üzere bazı oyun konsollarınca desteklenir. Donanım tarafında ise SGI, ATI, Nvidia veya Intel gibi büyük üreticiler her ekran kartında OpenGL desteği sunar [22].

Aşağıdaki sebeplerden dolayı tez kapsamında OpenGL kullanımı tercih edilmiştir.

- Taşınabilirlik
- Platform bağımsızlık
- Pencere yöneticisinden bağımsızlık
- Birçok programlama dilinde kullanılabilirlik

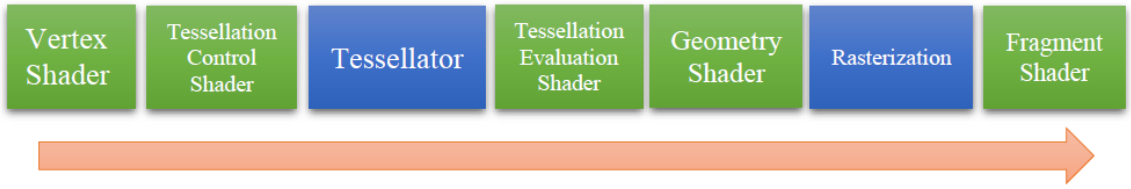
2.2 OpenGL çizim akışı

OpenGL 4.0 öncesi ekran kartında çizim akışındaki programlanabilen kısımlar Şekil 2.1'de [18] çizim akışı; köşe noktası akış programı, temel birleştirici akış programı, geometri akış programı, resimleştirme akış programı ve ekran noktası akış programından oluşmaktadır. Yeşil kutular programlanabilenleri göstermektedir.



Şekil 2.1: OpenGL 4 öncesi çizim akışı [18]

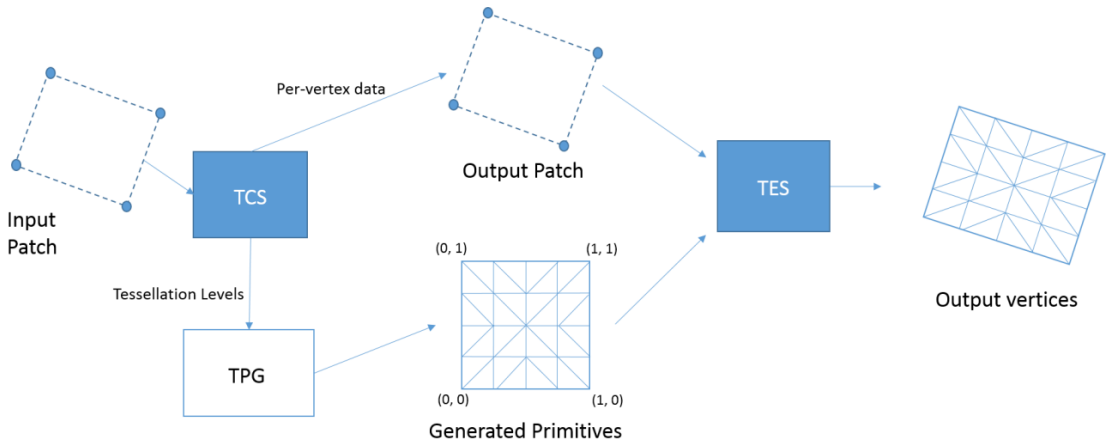
Fakat OpenGL 4 ile akışta programlanabilen kısımlar artmış ve böylelikle donanımsal mozaikleştirme kabiliyeti eklenmiştir. Şekil 2.2'de çizim akışı; köşe noktası akış programı, mozaikleştirme kontrol akış programı, mozaikleştirici, mozaikleştirme oluşum akış programı, geometri akış programı, resimleştirme akış programı ve ekran noktası akış programından oluşmaktadır



Şekil 2.2: OpenGL 4 sonrası çizim akışı [18]

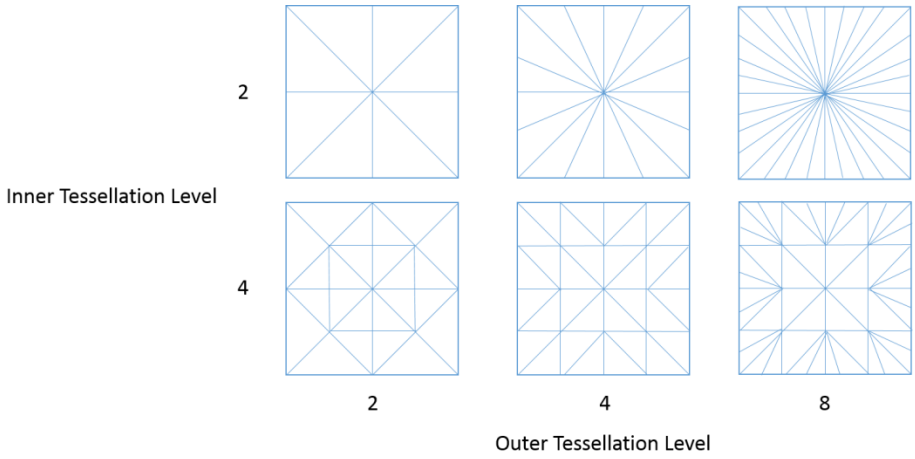
2.3 OpenGL donanımsal mozaikleştirme

Donanımsal mozaikleştirme özelliği DirectX 11 [16] ile 2011'in başında, OpenGL 4.0 [17] ile 2010 yılında gelmiştir. Şekil 2.3'de mozaikleştirme kontrol akış programı girdi olarak aldığı noktaları çoklanmış noktalar halinde mozaikleştirme oluşum akış programına gönderir. Mozaikleştirme oluşum akış programı da aldığı diğer parametreler göre noktalara son halini verir.



Şekil 2.3: Farklı mozaikleştirme seviyeleri görüntüsü [20]

Aşağıdaki Şekil 2.4'de örnek olarak farklı iç ve dış mozaikleştirme katsayısı ile oluşan şekiller [20] gösterilmektedir.



Şekil 2.4: Farklı mozaikleştirme seviyeleri görüntüsü [20]

2.4 OpenGL akış programı tampon nesnesi

Akış programı tampon nesnesi 2012 yılında yayınlanan Open 4.3 ile gelmiştir. Bu yüzden tez kapsamında yeni geliştirilen algoritma OpenGL 4.3 ve üzeri ekran kartlarında çalışabilecektir.

Akış programı tampon nesnesinin normal tampon nesnesinde ana farkları [19] :

- Akış programı tampon nesnesi çok büyük olabilir. Normal tampon nesnesi 16KB'a kadar desteklerken akış programı tampon nesnesi en az 16MB'ye kadar desteklemektir.
- Akış programı tampon nesnesindeki veriler atomik olarak yazılabilir. Verinin atomik olmasına gerek yok sadece erişim metotları atomik olması yeterlidir.
- Akış programı tampon nesnesi değişken boylu olabilmekteyken normal tampon nesnesi sabit boyutta olması gerekmektedir.
- Akış programı tampon nesnesindeki verilere erişim doku verisine erişim gibi olmaktayken normal tampon nesnesindeki verilere erişim içsel akış programı hafızasına erişim gibi olduğundan biraz daha yavaştır.

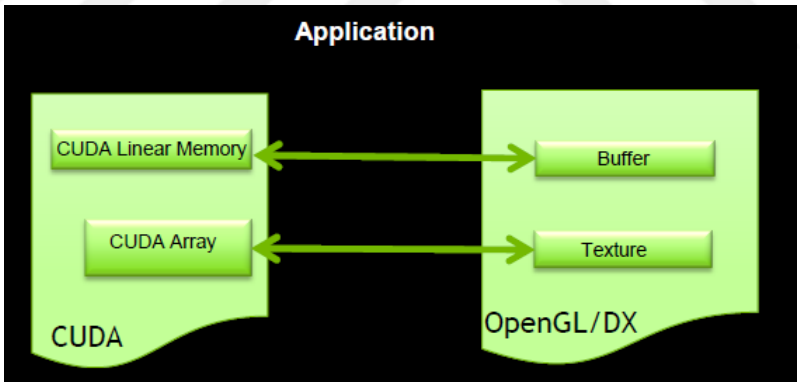
2.5 CUDA

CUDA, NVIDIA'nın GPU (grafik işlem birimi) gücünü kullanarak hesaplama performansında büyük ölçüde artışlara olanak veren paralel hesaplama mimarisidir [23].

GPU hızlandırmalı hesaplama, bir grafik işlem biriminin (GPU) bilimsel, analitik, mühendislik, tüketici ve kurumsal uygulamaları hızlandırmak için bir CPU ile birlikte kullanılmasıdır. 2007 yılında öncülüğünü NVIDIA'nın yaptığı GPU' hızlandırıcılar, artık dünya genelinde devlet laboratuvarları, üniversiteler, kurumlar, küçük ve orta işletmelerin enerji verimli veri merkezlerini güçlendirmektedir. GPU'lar otomobillerden cep telefonlarına ve tabletlere, insansız hava araçlarından robotlara, çeşitli platformlardaki uygulamaları hızlandırıyor [24].

2.6 OpenGL ve CUDA beraber çalışabilirliği

CUDA, Nvidia firması tarafından GPU programlama platformu olarak geliştirildi. CUDA 5.0 ile çizim kütüphanesi olan OpenGL ile beraber çalışabilmesi kapsamında aynı GPU hafıza alanlarının birlikte kullanımına yönelik metotlar geliştirildi [21]. Şekil 2.5'de CUDA ve OpenGL GPU hafıza paylaşımı temsili olarak gösterilmektedir.



Şekil 2.5: CUDA ve OpenGL GPU hafıza paylaşımı [21]

2.7 Noktasal görünürlük testi

Çizim yapılacak nesnenin başka bir nesne arkasında tamamen kalıp kalmadığını sorma işlemine örtme sorgusu denmektedir. Bu sorgu sayesinde örtülen nesne çizim işlemine gönderilmeyerek çizim yükü düşürülmüş olmaktadır.

Örtülenin çizilmemesi yöntemlerinden biri çevrimdışı olarak görünen görünmeyen nesnelerin tespit edilmesine dayanmaktadır. Bu yöntem birçok kullanıcı etkileşimli uygulamalar için uygun değildir [8].

Bir diğer yöntem de hücre tabanlı görünürlük testi yapmak. Bu kapsamda sahne hücrelere bölünür ve bağlantı noktaları ile bağlanır. Bunu odaların kapılar ve pencereler ile bir birine bağlanması şeklinde düşünebiliriz. Bu işlem çevrim dışı yapılacağı gibi sahnede gezinme sırasında da yapılabilir [8].

Çevrim içi örtme sorgusu yöntemi daha genel ve dinamik sahneler için daha uygundur. Tipik çevrim için örtme sorgusu hesaplama yükünü azaltmak için resim tabanlı olarak çalışır. Bu yöntem için CPU yerine GPU kullanmak daha efektiftir çünkü resimleştirme işinde GPU özelleşmiştir [8].

Tez çalışması kapsamında arazi görselleştiriminde sürekli kullanıcı ile etkileşim halinde olunduğundan dolayı çevrim içi örtme sorgusu yöntemi benimsenmiştir.

Opengl çizim kütüphanesine örtme sorgusu desteği ilk olarak NV_occlusion_query daha sonrada ARB_occlusion_query fonksiyonları ile desteklenmiştir. Direct3D çizim kütüphanesi ile ise D3DQUERYTYPE_OCCLUSION eklentisi ile gerçekleştirilmiştir. Bu sayede çizim yapılacak nesnenin ekranda kaç noktasının gözüktüğü sorgulanabilir hale gelmiştir [9]. Genel kullanım akışı şu şekildedir:

1. Örtme sorgusu ilklendirilir.
2. Çizimin hızlı yapılması için ekran ve derinlik tamponları kapatılır ayrıca çizimi güzelleştirmek için kullanılan ilave özellikler kapatılır.
3. Test edilecek nesnenin daha az karmaşık bir hali çizilir. Genelde çevreleyen kutusu çizilir.
4. Örtme sorgusu bitirilir.
5. Örtme sorgusu sonucu alınır.
6. Eğer örtme sorgusu sonucu belli bir eşikten büyükse sıfır gibi nesne çizilir aksi halde ise çizilmez.

ÜÇÜNCÜ BÖLÜM

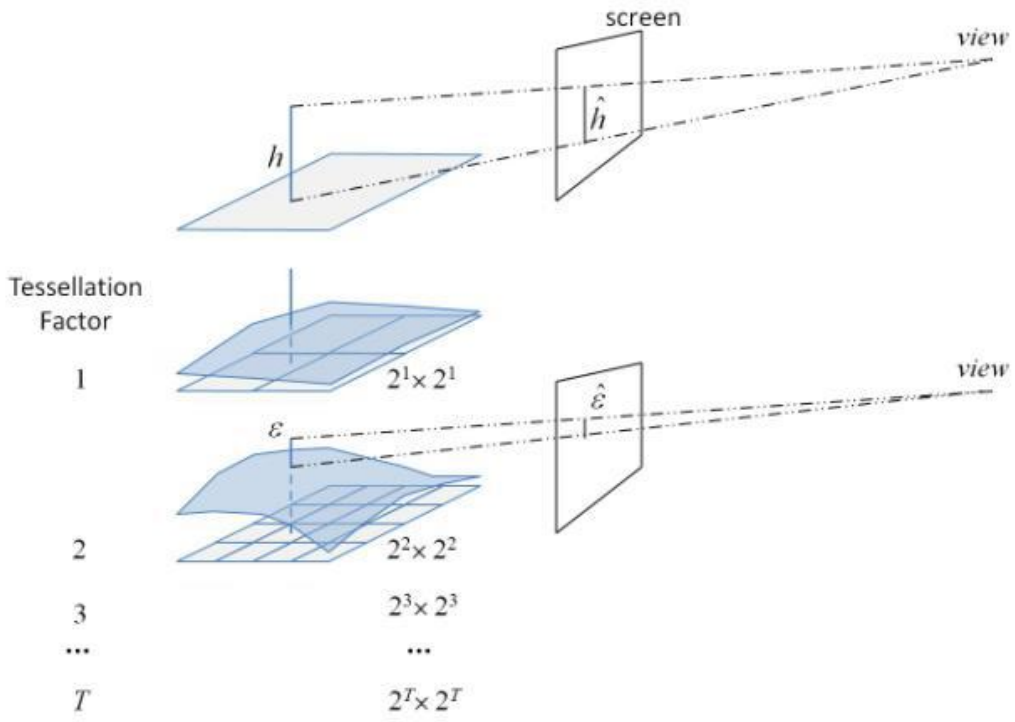
LİTERATÜR ÖZETİ

Literatür özetinin ilk kısımda da son zamanlarda geliştirilen GPU tabanlı mozaikleştirmeyi kullanarak arazi görselleştirmesi yapan algoritmalar kronolojik olarak anlatılmaktadır. Böylelikle tezde kullanılan algoritmanın gelişim süreçleri daha iyi anlaşılması amaçlanmaktadır. İkinci kısımda ise noktasal görünürlük tabanlı LOD yöntemleri anlatılmaktadır. Bu şekilde tez kapsamında kullanılan ve geliştirilen noktasal görünürlük tabanlı LOD algoritmaları daha iyi anlaşılacağı öngörülmektedir.

3.1 Donanımsal Mozaikleştirme İle Arazi Görselleştirme

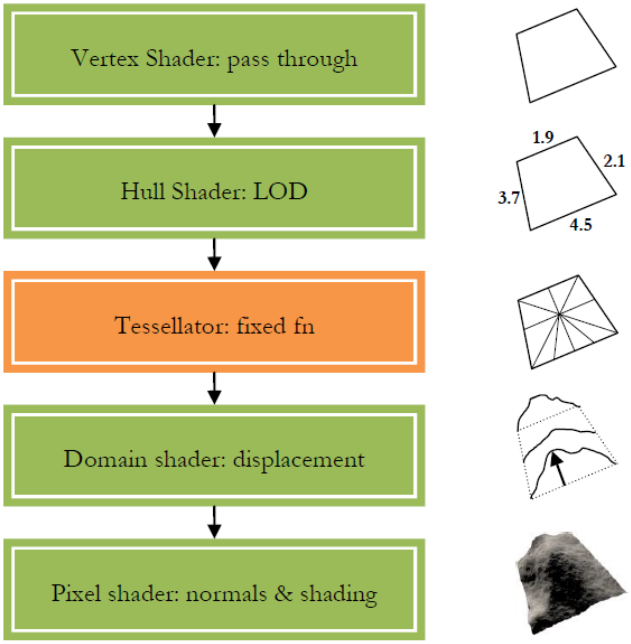
Grafik İşlemci Birimi (GPU)'inde çalışan mozaikleştirme teknolojisi geliştirildikten sonra bu özelliği kullanarak arazi görselleştirmesine yönelik birçok makale yayınlanmıştır.

2010 yılında Valdetaro [1] donanımsal mozaiklemeyi kullanan sürekli görüş bağımlı detay seviyesini geliştirerek arazi görselleştirme tekniğini önermiştir. Detay seviyesi belirlemede sadece arazinin kameraya uzaklığını değil aynı zamanda görünen arazideki yükseklik farklarını da dikkate alınmıştır. Şekil 3.1'de kabul edilebilir hata payına erişen en küçük mozaikleştirme faktörünün bulunması aşamalarla gösterilmiştir. Bu yaklaşım olumsuz tarafı sadece tekdüze boyutlu arazi parçalarını kullandığından dolayı herhangi bir hiyerarşik arazi veri yapısına uyarlanabilir değildir. Bu yüzden çok büyük arazi görselleştirilmesi kapsamında uygun bir algoritma değildir.



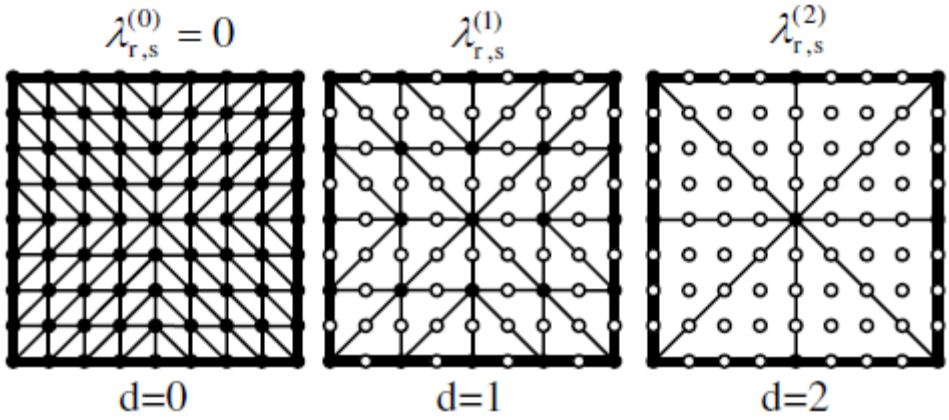
Şekil 3.1: Minimum mozaikleştirme faktörü belirleme [1]

2011 yılında Cantlay [2] DirectX 11'in mozaikleştirme özelliğini kullanarak arazi görselleştirmesi yapan bir teknik gerçekleştirmiştir. CPU tabanlı algoritmalara göre kayda değer bir geliştirme sağlandığını göstermiştir. Şekil 3.2'de çizim program akış hattındaki aşamalardan geçişlerde gerçekleşen işlemler gösterilmektedir.



Şekil 3.2: Çizim program akış hattı aşamalarında uygulanan işlemler [2]

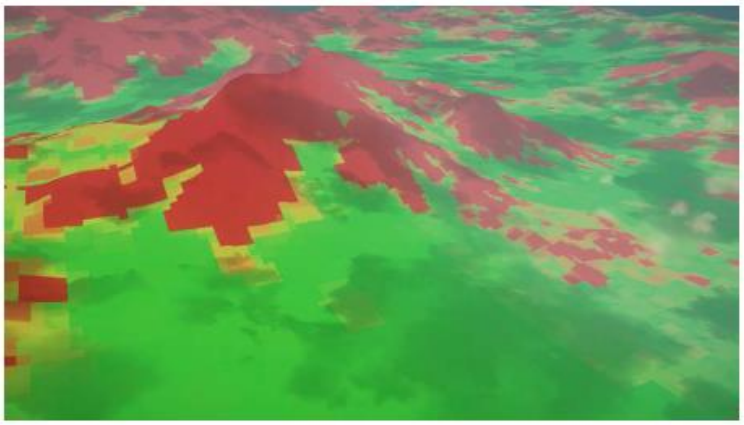
2011 yılında geliştirilen diğer bir algoritma ise Egor Yusov [3] tümüyle mozaikleme birimi tarafından uyumsal bir şekilde üçgenleştirmeye dayalı yeni arazi görselleştirme yaklaşımıdır. Şekil 3.3'de örnek olarak 9x9 bir mozaikleştirilecek bloğun üçgenleştirilmesi gösterilmektedir. Komşu arazi parçaları arasında oluşan boşlukları yok etmek için de kenarlar boyunca T. Ulrich [4] tarafından geliştirilen dikey örtme yöntemi kullanılmıştır.



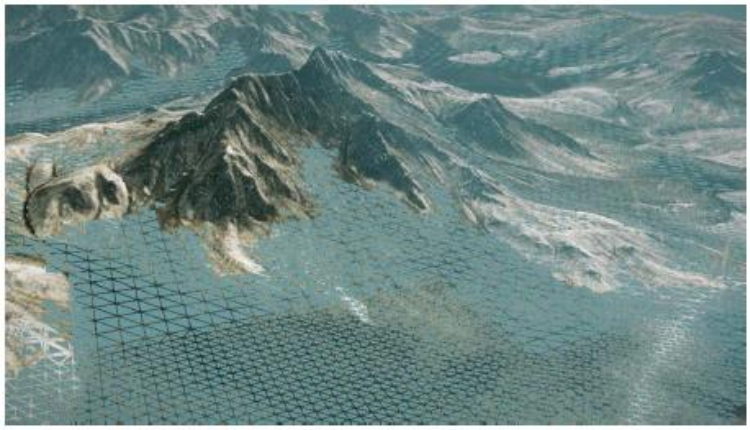
Şekil 3.3: 9x9 mozaikleştirilecek bloğun üçgenleştirilmesi [3]

2012 yılında Albert Cervin [5] adaptasyonlu donanımsal arazi mozaikleştirme yoğunluk şemasını geliştirmiştir. Bu şema yardımıyla çevrimdışı hesaplamalarla yoğunluk haritası oluşturulmuştur. Fakat GPU mozaikleştirme işlemi dörtgen yerine üçgenler için uygulandığından çok efektif değildir. Şekil 3.4'de yoğunluk şeması uygulanmadan mozaikleştirilmiş arazinin tel görünümdeki hali gösterilmektedir. Şekil 3.5'de ise arazi üzerindeki yoğunluk şeması görselleştirilmektedir. Şekil 3.6'da ise yoğunluk şeması uygulanmış tel görünümdeki arazi görselleştirilmektedir.

Şekil 3.4: Yoğunluk şeması uygulanmamış tel arazi gösterimi [5]

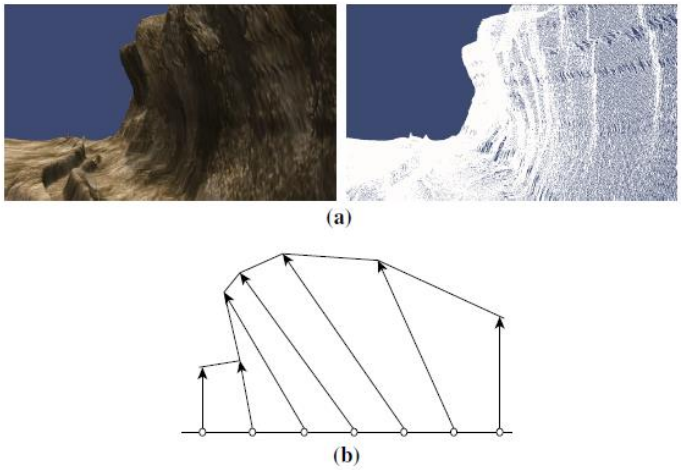


Şekil 3.5: Yoğunluk şeması gösterimi [5]



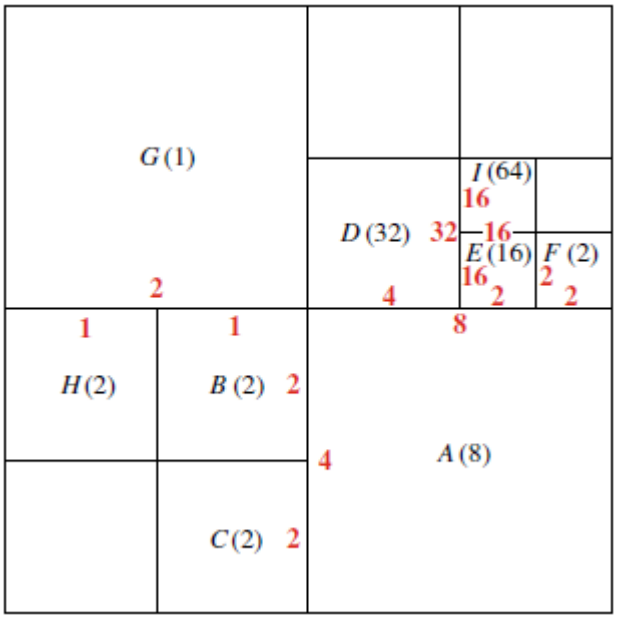
Şekil 3.6: Yoğunluk şeması uygulanmış tel arazi gösterimi [5]

2015 yılında Hyeong Yeop [6] GPU mozaikleştirme kullanarak o zamana kadar önerilmemiş bir arazi görselleştirme algoritması önermiştir. Sadece yükseklik haritası değil aynı zamanda geometrik resimleri de kullanmıştır. Bu sayede arazi yüzey özelliklerinin efektif bir şekilde açılmasında yardımcı olmuştur. Şekil 3.7'de dik yükselen bir arazinin uyumlandırılması gösterilmektedir.



Şekil 3.7: Dörtlü ağaç oluşturulurken dış mozaikleştirme katsayısı belirleme [6]

Kenar boşluklarını gidermek için ilk önce içsel mozaikleştirme seviyesi kullanarak gideren bir yaklaşımı geliştirilmiş ardından da kenar mozaikleştirme katsayısını hesaplamak için gerçekleştirimi kolay olmayan karışık bir akış geliştirilmiştir. Şekil 3.8'de dörtlü ağaç veriyapısı oluşturulurken CPU'da iç mozaikleştirme katsayısı ekran hata katsayısı hesaplanarak elde edilmesi gösterilmiştir. Parantez içerisindeki rakam iç mozaikleştirme katsayısı iken kırmızı ile yazılmış rakam ise dış mozaikleştirme katsayısıdır.



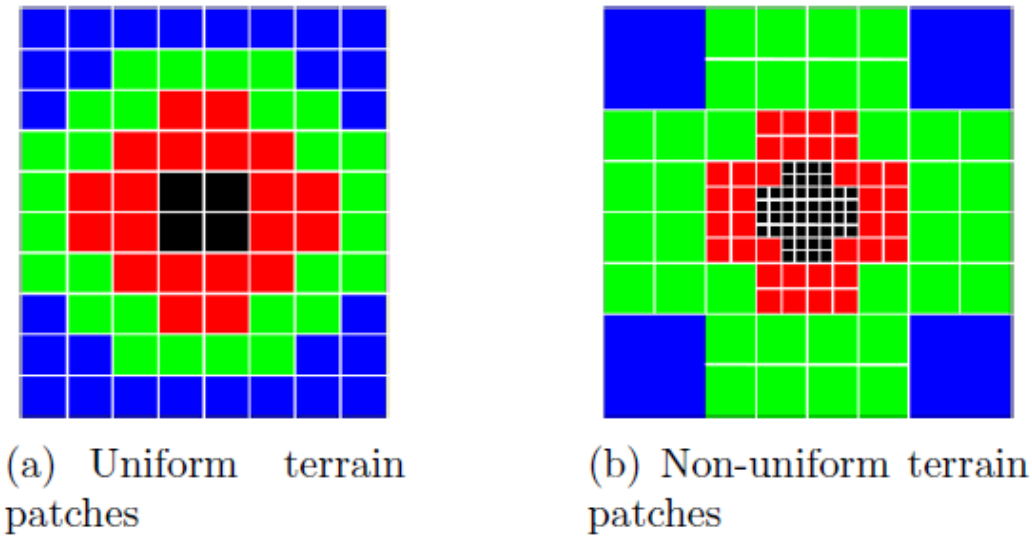
Şekil 3.8: Dörtlü ağaç oluşturulurken dış mozaikleştirme katsayısı belirleme [6]

Dış mozaikleştirme katsayısı, komşu hücre iç mozaikleştirme katsayısı ile diyagonal uzunluğuna orantılı olarak hesaplanmaktadır. Formülleştirmek gerekirse bir hücrenin komşu kenarının uzunluğu ile iç mozaikleştirme katsayısının çarpımı, komşu hücrenin komşu kenar uzunluğu ile komşu hücrenin iç mozaikleştirme katsayısının çarpımına eşit olması gerekmektedir.

2015 yılının sonlarına doğru Dong Wang [7] GPU mozaikleştirme yardımıyla uyumsal LOD arazi görselleştirmesini önermiştir. Tez kapsamında bu makale temel alınarak geliştirilen koda ilave metotlar eklenmiştir. Kameraya uzaklığa göre Hyeong Yeop [6]'un yaklaşımındaki gibi CPU'da dörtlü ağaç veriyapısı oluşturulmuştur. Daha sonra dörtlü ağaç veriyapısındaki parçalar GPU mozaikleştirme kullanılarak detaylandırılmaktadır. Kenar boşlukları gidermek için dış mozaikleştirme katsayılarını komşu arazi parçalarında aynı olacak şekilde hesaplayarak çok rahat bir şekilde bu problemin üstesinden gelmektedir. İkinci olarak önerdiği donanımsal mozaikleştirme iç ve dış katsayısını ekran kartında hesaplamaya yöneliktir. Fakat tez çalışması sırasında bu algoritmada bazı durumlarda nadirde olsa boşluklar oluşabildiği gözlemlenmiştir. Tez kapsamında boşluk oluşma durumları tespit edilmiş ve giderilmiştir.

Detay seviyesi belirleme yöntemi iki kısımdan oluşmaktadır birincisi CPU tabanlı dörtlü ağaç veriyapısında detay seviyesini belirlemeye yöneliktir. İkinci kısım ise CPU'da belirlenen arazi parçasının donanımsal mozaikleştirme yöntemiyle daha detaylı halde çizilmesi şeklindedir.

GPU tabanlı detay seviyesi belirleme işlemini de iki farklı yöntem de denenmiş ve performans kıyaslamaları yapılmıştır. Fakat makalede üçgen sayıları ile çizim performansı kıyaslanmış. Hâlbuki donanımsal mozaikleştirme yöntemi ile ne kadar çok üçgeni bu şekilde çizerseniz o kadar üçgen çizme performansınız artar fakat bunun ekrandaki görüntünün iyileşmesinde bir faydası olmayabilir. Tez kapsamında özellikle bu kısım iyileştirilerek ciddi oranda performans artışı sağlanmıştır. Çünkü ekranda gözükmeyen arazi parçalarının çizilmesinin hiçbir manası yoktur. Eşit dağılımlı dörtlü ağaç veriyapısında arazi parçaları eşit şekilde bölünürken detay seviyesi uzaklıkla orantılı belirlenmektedir. Eşit olmayan dağılımlı dörtlü ağaç veriyapısında arazi parçaları eşit şekilde bölünmez onun yerine uzaklıkla orantılı şekilde bölünür. Şekil 3.9'da eşit olan ve eşit olmayan dörtlü ağaç detaylandırması gösterilmektedir.



(a) Uniform terrain patches

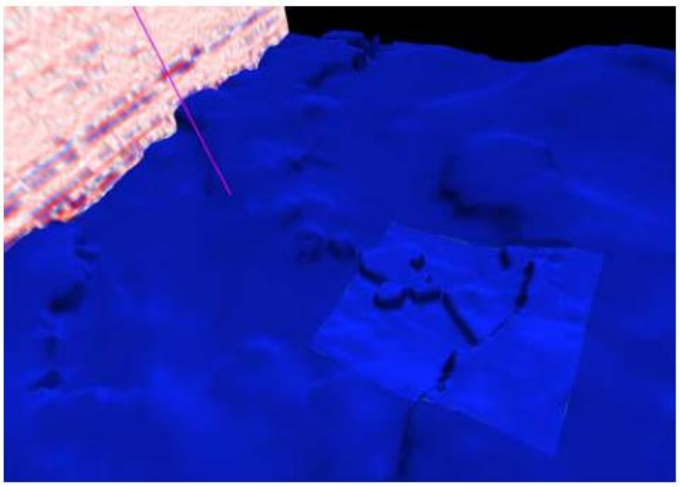
(b) Non-uniform terrain patches

Şekil 3.9: Eşit olan ve eşit olmayan dörtlü ağaç detaylandırması [7]

3.2 Noktasal Görünürlük Tabanlı LOD

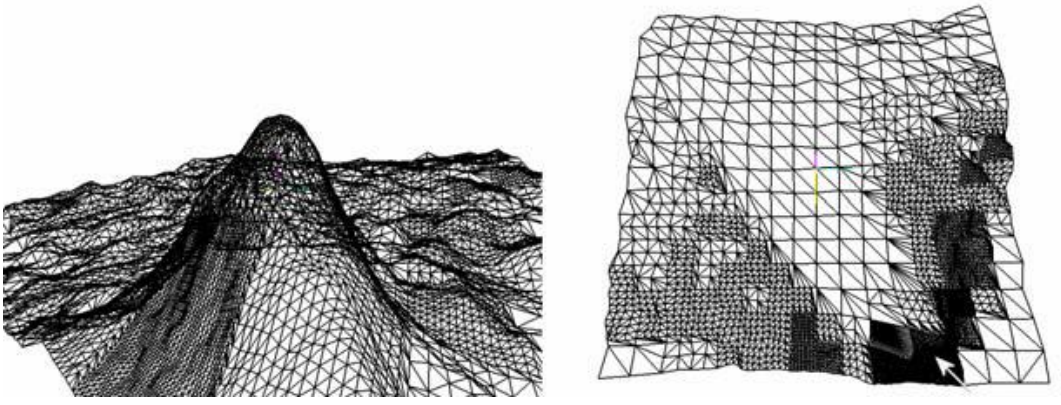
Noktasal görünürlük testi ile detay seviyesi belirlemeye yönelik birçok makale yayınlanmıştır.

2004 yılında John Plate [10] noktasal görünürlük testi kullanarak üç aşamalı çizim yaparak performans artışı sağlamıştır. Bu yöntemin bir avantajı da çizilecek nesneleri sıralamaya gerek olmamasıdır. İlk aşamada ışık, gölgelendirme ve dokulaştırma kapatılmış tüm nesneleri düşük çözünürlükte çizilmiş. İkinci aşamada her bir nesne için örtme sorgusu başlatıp tekrar düşük çözünürlükte çizimlerini yapıp örtme sorgusunu bitirilmiştir. Üçüncü aşamada ise sorgu sonuçlarına bakarak daha detaylı çizim yapıp yapmamaya karar verilmiştir. Burada makalesinde neredeyse fark edilmeyen problemler dışında görüntüde bozulmalar olmuyor dese de derinlik tampon savaşı olması kuvvetle muhtemeldir. Çünkü hem düşük çözünürlüklü çizim yapmakta eğer görünüyorsa da daha detaylısını üstüne çizmektedir. Şekil 3.10'da görüldüğü gibi az detaylı çizim üzerine detaylı çizimin belli problemi gözükmektedir.



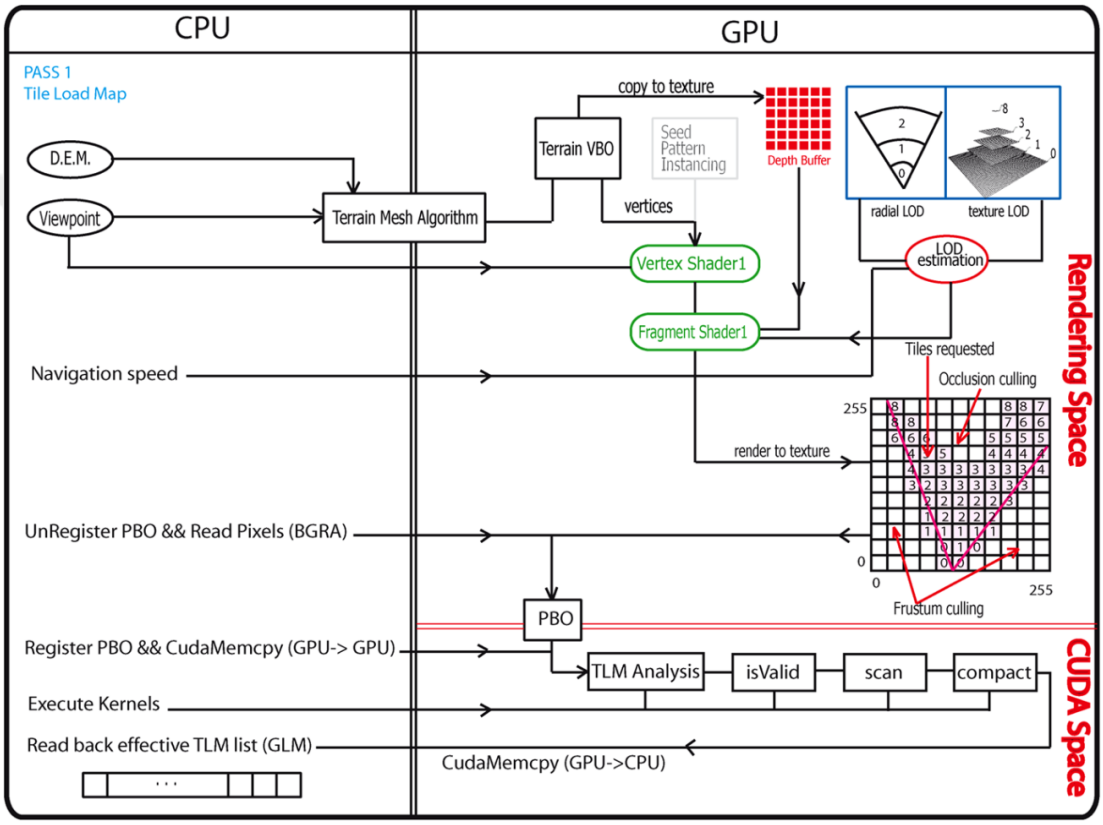
Şekil 3.10: Az detaylı çizim üzerine detaylı çizim [10]

2006 yılında benzer yöntemi kullanan Nick Gebbie [11] programlanabilen grafik donanımı kullanarak hızlı ve gerçekçi küresel dünya çizimi önerilmiştir. Optimizasyon amaçlı örtme sorgusunun normal akışındaki gibi arazi parçalarının çevreleyen kutuları ile önce test yapmak yerine normal çizimi yaparken her bir parsel için örtme sorgusu da çalıştırılmıştır. Böylelikle hem çizim yapıp hem de her bir arazi parçasının kaç noktasının gözüktüğü anlaşılmıştır. Bunun dezavantajı ise çizim yapmadan önce hangi parselin görünür olduğunun bilinmiyor olmasıdır. Görünürlük bilgisini diğer çizimde kullanarak gözükmeyen parselleri az detaylı çizip görünür olanları görünme oranında detaylı çizilmiştir. Görünmeyeni az detaylı çizmesinin sebebi kamera hareket ettikten sonraki ilk çizimde o parselin gözükebilir olmasından kaynaklanmaktadır. Ayrıca kabarma efektini yani bir parselin birden çok detaylı olmaması içinde nokta tabanlı bir yaklaşım geliştirmiştir. Şekil 3.11'de dağın arkasında kalan bir alanın detaylandırılması gösterilmektedir.



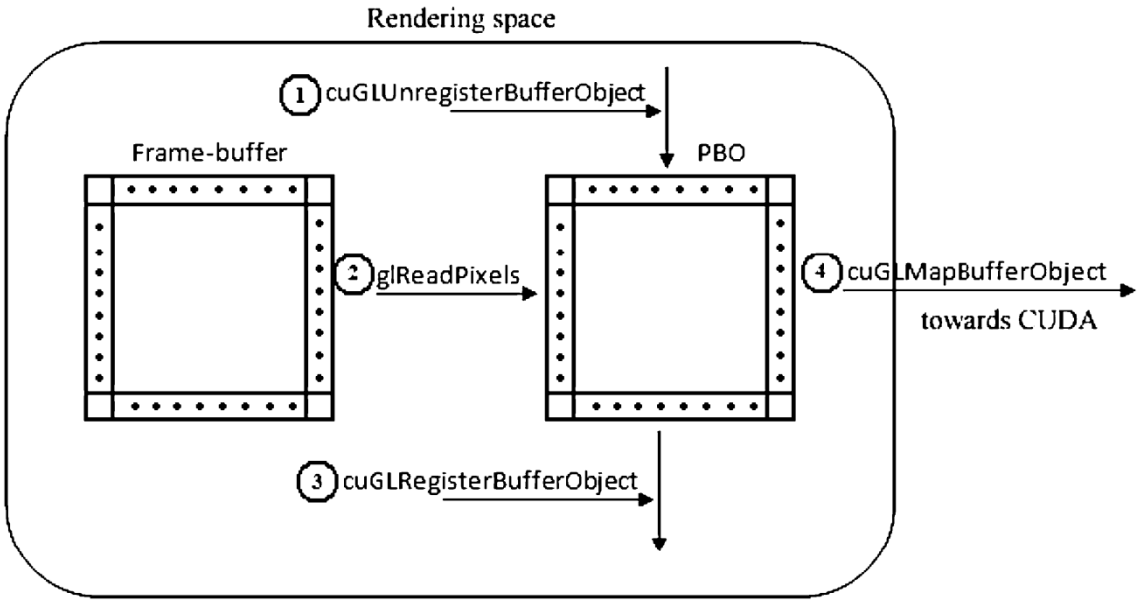
Şekil 3.11: Dağın arkasındaki parsellerin tel görünümü [11]

2008 yılında CUDA-OpenGL birlikte çalışabilirliğini de kullanan makalelerden biri olan Yacine Amara [12] Arazi görselleştiriminde GPU parça yük harita teorisi ve uygulamasını önermiştir. Parça yük haritasını GPU'da hesaplayarak kullanmaktadır. Bu teorinin tek dezavantajı dört aşamadan oluşmasıdır. Tüm araziyi sabit büyüklükte küçük parçalara bölerek bu parçalara tekil numara vermektedir. Birinci aşama sadece detay seviyesi belirlemeye yöneliktir. Parselleri kamera pozisyonuna göre 256x256'lık bir tampona çizerek arazi parsellerinin görünürlük haritasını Şekil 3.12'deki gibi oluşturmaktadır.



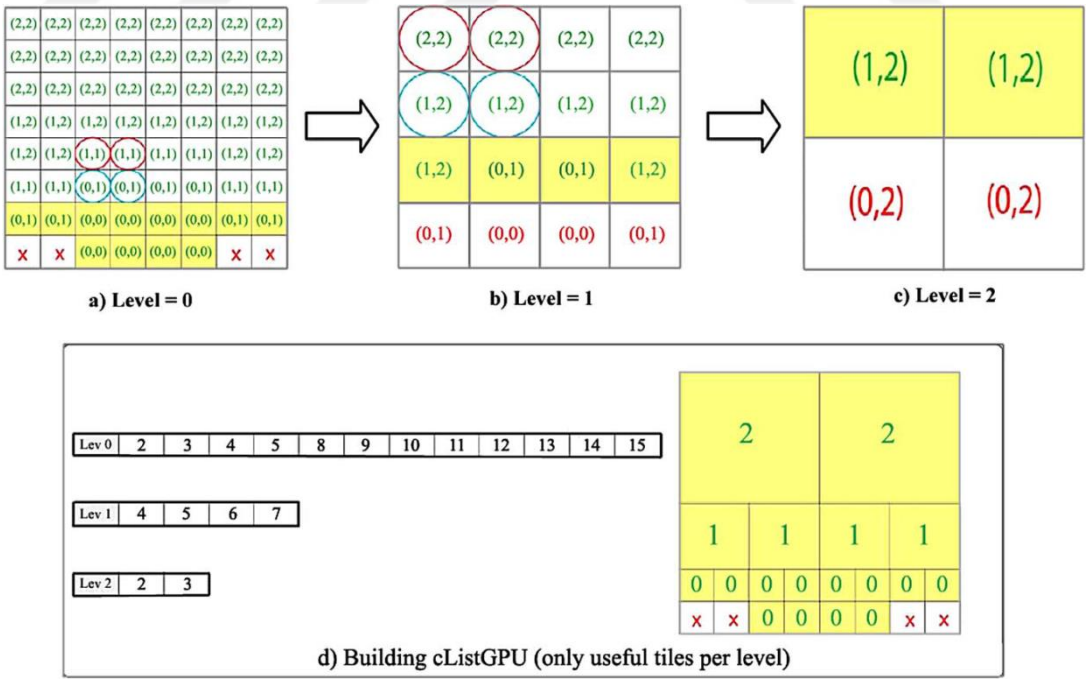
Şekil 3.12: Parça yük haritası ilk çizim aşaması [12]

Daha sonra OpenGL ve CUDA beraber çalışabilirlik kullanılarak arazi parsellerinin görünürlük haritası OpenGL çizimi bitirilerek Şekil 3.13'deki gibi CUDA'ya parametre olarak geçilir.



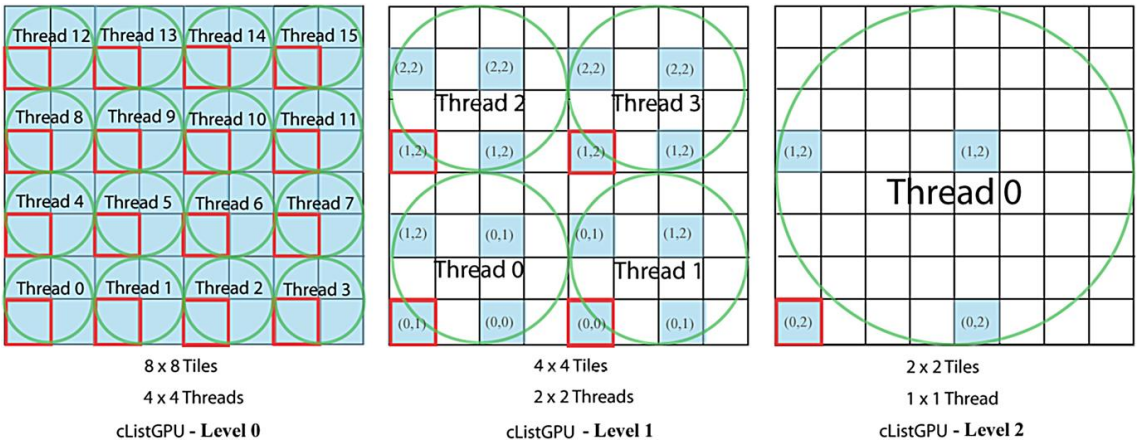
Şekil 3.13: Çizim uzayından CUDA'ya tampon transferi [12]

Şekil 3.14'de örnek olarak 8x8 bir parselde detay seviyesi belirleme işlemleri gösterilmektedir. Parantez içindeki ilk parametre en düşük detay seviyesini gösterirken ikinci parametre ise en yüksek detay seviyesini göstermektedir.



Şekil 3.14: 8x8 parsel için LOD hesaplama [12]

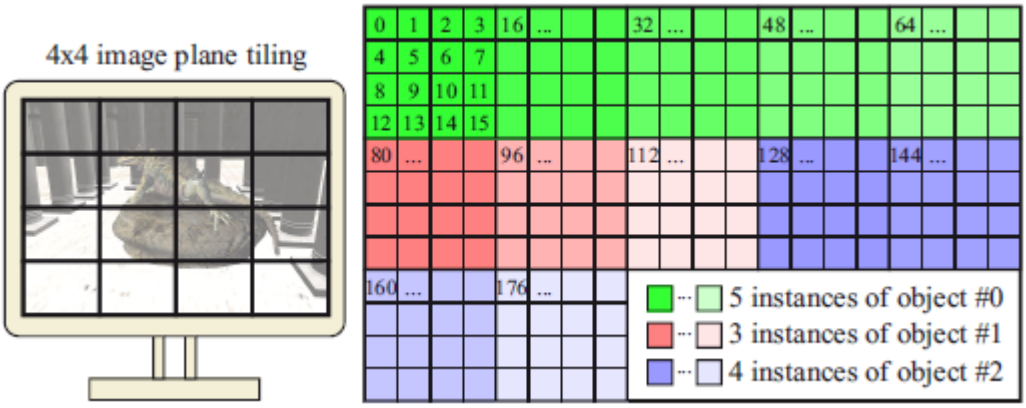
Detay seviyesi belirlenirken Şekil 3.15'deki gibi CUDA programı yardımıyla paralelde birçok iş parçası ile seviye seviye hesaplanması gösterilmektedir.



Şekil 3.15: 8x8 parsel için CUDA iş parçası dağılımı [12]

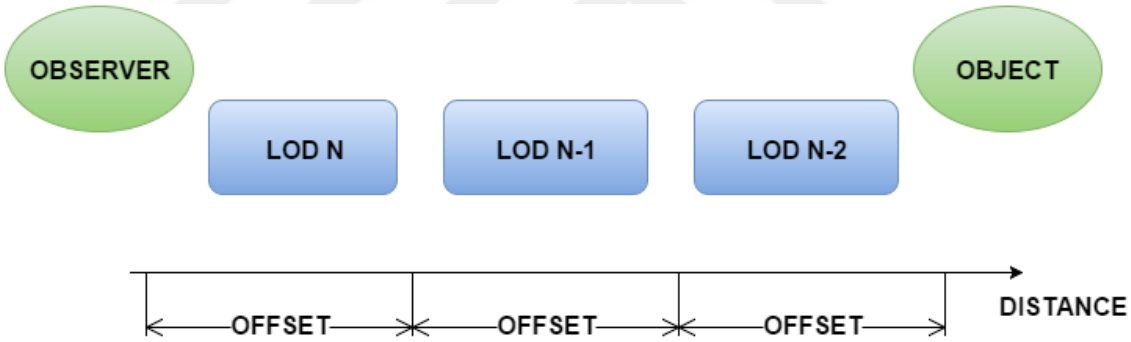
Diğer üç aşama burada anlatılmayacaktır fakat okunmak istenirse Yacine Amara [12]'den bakılabilir. CUDA aslında binlerce işi paralelde performanslı bir şekilde hesaplamak için özelleşmiştir. Burada OpenGL ve CUDA beraber çalışabilirliğinin getirdiği fazladan bir maliyet vardır. Ve kullanılan CUDA iş parçası sayısına baktığımızda buradaki hesaplama süresinin görece daha az olması beklenir. Fakat makalede beraber çalışabilirlik süresi ölçülmemiş sadece CUDA hesaplama süresi ölçülmüştür. Bu detay seviyesi hesaplama süresinin makaledeki gibi çok az olmayacağı kanaatini oluşturmaktadır.

2009 yılında Thomas Engelhardt [13] GPU'da tanecikli görünürlük sorgulama yöntemini önermiştir. Ana amacı donanımsal örtme sorgusundan farklı olarak alt parçaların görünürlüğünü de aynı anda hesaplayıp ekran kartındaki hesaplamalarda direk erişebilmektir. Bu kapsamda iki metot geliştirdi. İlki sorgulanacak nesneleri renkli bir dokuya çizerek ait olduğu noktaları saymaya dayanır. Bunun için toplam bölge tablosu ve kolay erişebilmek için küçük hiyerarşik haritalama yapısı kullanılmıştır. İkinci metotta ise renk yerine her bölge için ayrı hesaplanan bir tekil belirteç kullanılmıştır. Bölgelere ayırma işlemleri CPU'da yapılırken toplam bölge tablosu oluşturma işlemi de GPU'da yapılmaktadır. Daha sonra bu tablo çizim yapılırken kullanılmaktadır. Şekil 3.16'da hiyerarşik nesne tamponundan oluşturulan iki boyutlu histogram tablosu gösterilmektedir.



Şekil 3.16: 3 farklı nesneden oluşan 12 nesne örneği histogramı [13]

2016 yılında Szymon Jabłonski [14] Seyrek hacimsel nokta sekizli ağacı için sürekli detay seviyesinin gerçek zamanlı çizimi yaklaşımını önermiştir. Uygulama alanı arazi görselleştiriminden farklı da olsa benzer yaklaşımın kullanılabilir olmasından dolayı önemlidir. Şekil 3.17'de makale kapsamında CPU'da hesaplanan sürekli LOD algoritmasındaki detay seviyesi değişimi gösterilmektedir.



Şekil 3.17: Sürekli LOD de detay seviyesi değişimi [14]

Nokta dolum oranı tabanlı detay seviyesi yaklaşımında görselleştirme algoritmasına göre ya GPU hesaplama programlanabilir işlemcisinde ya da GPU sorgusu ile kaç noktanın çizildiği hesaplanır. Makalede CUDA ile de hesaplamaların yapılabileceği fakat GPU hesaplama programlanabilir işlemcisi ile yapıldığı belirtilmektedir. Hesaplama tamponu olarak RG32UI kullanılmıştır. Paralel küçültme yaklaşımıyla 1280x720 çizim tamponu 80x45'e düşürülerek CPU'da kullanılmak üzere transfer edilmektedir.

DÖRDÜNCÜ BÖLÜM

MATERYAL VE YÖNTEM

Bu çalışmanın ana amacı arazi görselleştiriminde çizim kalitesinden ödün vermeden daha hızlı çizim işleminin yapılmasıdır. Bu kapsamda literatürde bilinen en son geliştirilen algoritmalar incelenerek arazi görselleştiriminde çizim hızının artırılmasına yönelik yeni önerilerde bulunulmuştur.

4.1 Arazi Görselleştiriminde Kullanılan Veriler

Çizim kapsamında kullanılan tüm ham veriler uygulama açılır açılmaz direk ekran kartı hafızasına gönderilmekte ve böylelikle normal hafıza ile ekran kartı hafızası arasında büyük veri transferleri yapılmayarak ciddi oranda veri transferi maliyetinden kurtulmuş olunmaktadır. Hafızaya sığmayacak büyüklükteki veriler için sayfalama yapma işlemi kapsam dışı tutulmuştur. Bununla alakalı öneriler kısmında daha detaylı değinilecektir.

Testlerin yapıldığı bölge olarak büyük kanyon seçilmiştir. Aşağıda kullanılan ham veriler ve boyutları yer almaktadır.

- Yükseklik verisi (7,68 MB) tga formatında 4097x2049 noktadan oluşmaktadır.
- Yüzey normalleri verisi (11,4 MB) tga formatında 4097x2049 noktadan oluşmaktadır.
- Toprak dokusu (12,0 MB) tga formatında 2048x2048 noktadan oluşmaktadır.
- Çimen dokusu (6,43 MB) tga formatında 1500x1500 noktadan oluşmaktadır.
- Kar dokusu (3,00 MB) tga formatında 1024x1024 noktadan oluşmaktadır.
- Gökyüzü küp dokusu (54,0 MB) tga formatında 6x1024x024 noktadan oluşmaktadır.

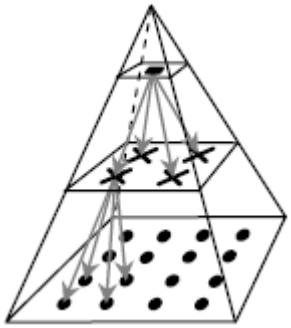
4.2 OpenGL Çizim Akış Programı

Uygulamada ekran kartında çalışacak kod parçacıkları da bir dosyada tutulmakta ve uygulama açılır açılmaz çalışma zamanında yüklenip ekran kartı uygulaması olarak derlenmektedir. Ekran kartı uygulamaları ve kod parçacıkları şu şekildedir:

1. Arazi çizim uygulaması
    a. Köşe noktası akış programı
    b. Mozaikleştirme kontrol akış programı
    c. Mozaikleştirme oluşum akış programı
    d. Geometri akış programı
    e. Ekran noktası akış programı
2. Gökyüzü kutusu çizim uygulaması
    a. Köşe noktası akış programı
    b. Ekran noktası akış programı

4.3 Dörtlü Ağaç Veriyapısı

Tez kapsamında kullanılan yapıların başında arazi verilerinin hiyerarşik bir şekilde tutulduğu dörtlü ağaç yapısı gelmektedir. Şekil 4.1'de çoklu çözünürlüklü dörtlü ağaç veriyapısı gösterilmektedir [15]. Çözünürlük sadece mozaikleştirme katsayısını tutmak şeklinde olmaktadır.



Şekil 4.1: Dörtlü ağaç veriyapısı [15]

Bir dörtlü ağaç parçası aşağıdaki bilgilerden oluşmaktadır:

- Dörtlü ağaç parçası tekil belirteci
- Orta nokta koordinatı
- Boyut bilgileri
- İç mozaikleştirme katsayısı
- Dış mozaikleştirme katsayısı

- Ağaç parçası yaprak mı?
- Ata ve çocuk ağaç parçası işaretçisi

Ağaç parselleri çizilmeden önce orta noktasının kameraya uzaklığına göre sıralanıp yakından uzağa doğru çizilmeleri sağlanmaktadır.

4.4 Çizim akışı

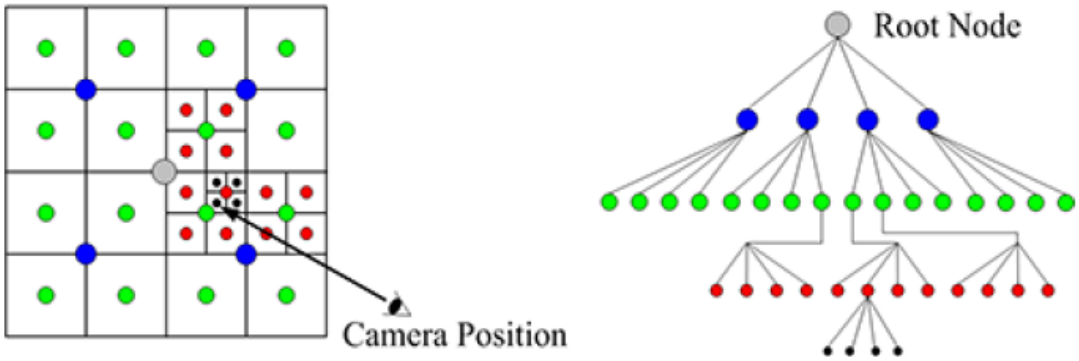
Uygulamanın ana akışı aşağıda anlatıldığı gibidir. Bu akışa tez kapsamında önerilen yöntemlerde ilaveler gelecektir. Bu akış temel model olarak kabul edilmiştir.

1. İlklendirme işlemleri
    a. Arazi verilerini yükle
    b. Ekran kartı akış programlarını yükle
    c. Ekran kartı akış programlarını derle
    d. Arazi verilerini ekran kartına yükle
2. Çizim işlemleri
    a. Dörtlü ağaç veriyapısını LOD algoritması ile oluştur/güncelle.
    b. Arazi parçalarının iç mozaikleştirme katsayılarını hesapla
    c. Arazi parçalarının dış mozaikleştirme katsayılarını hesapla
    d. Arazi parçalarını çiz

4.5 Yöntem-1 Uyumsal Detay Seviyesiyle LOD Belirleme

Donanımsal Mozaikleştirme İle Arazi Görselleştirme kısmında anlatılan Dong Wang [7] GPU mozaikleştirme yardımıyla uyumsal LOD arazi görselleştirmesi yöntemidir. İki aşamalı detay seviyesi belirleme yöntemini kullanmıştır.

CPU detay seviyesi belirleme kapsamında dörtlü ağaçtaki parçanın daha detaylı bölünme koşulu, kameraya olan uzaklığının arazi parçasının diyagonal uzunluğuna oranından belli bir sabitten daha küçük olmasıdır. Şekil 4.2'de [7] dörtlü ağaç veriyapısının kamera pozisyonuna göre nasıl bölümlendirildiği gösterilmektedir.



Şekil 4.2: Kamera pozisyonuna göre dörtlü ağaç veri yapısı [7]

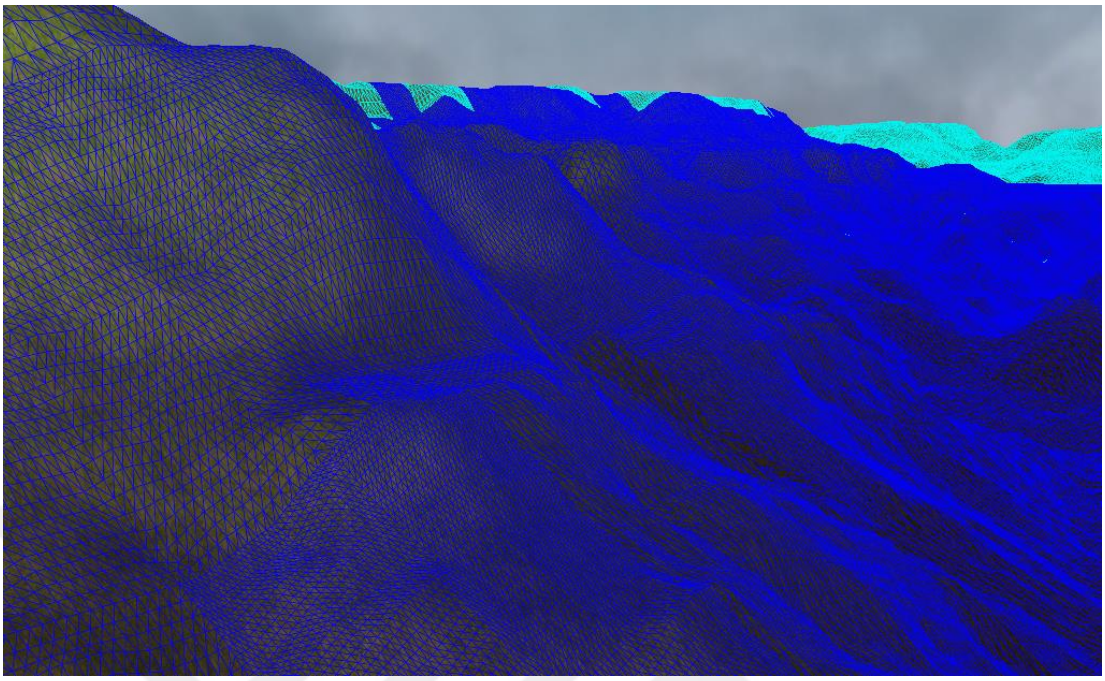
Bu oran küçüldükçe arazi parçaları daha büyük hale gelmekte iken bu oran büyüdüğünde ise arazi parçaları daha küçük hale gelmektedir. Arazi parçalarının küçük olması daha kaliteli LOD belirlememize imkân verirken performansı olumsuz yönde etkilemektedir.

GPU detay seviyesi belirleme kapsamında mozaikleştirme detay seviyesi belirleme yöntemi ise kamera uzaklık yöntemidir [20]. Böylelikle ekrana yakın arazi parçaları daha detaylı çizilirken ekrandan uzak olan arazi parçaları daha az detaylı çizilmektedir.

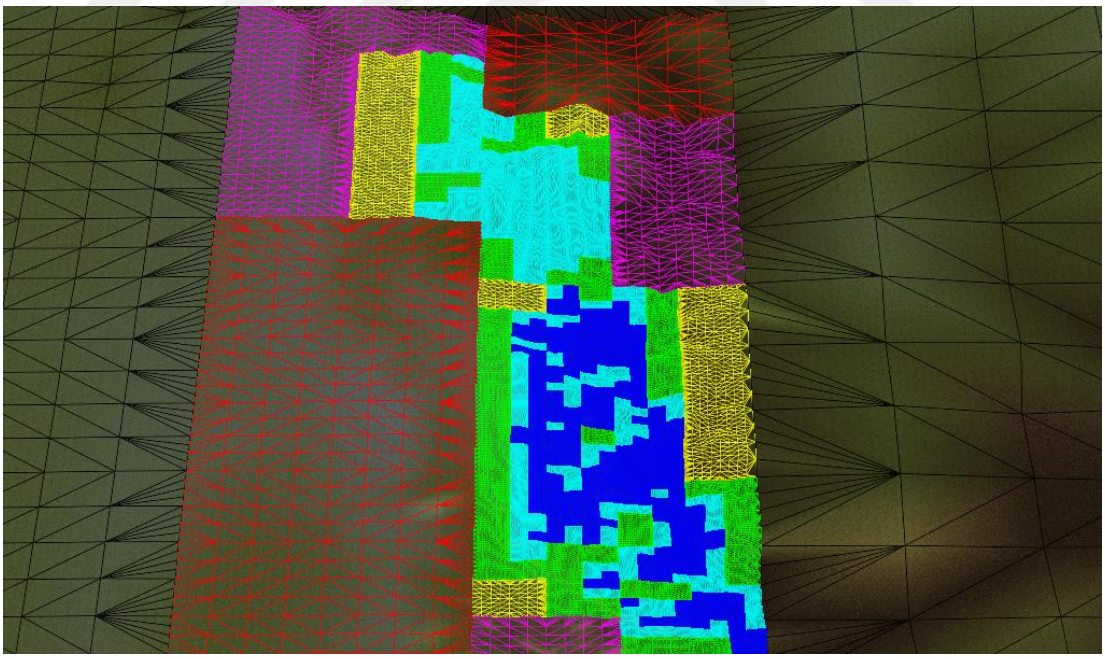
4.6 Yöntem-2 Örtme Sorgusu ile LOD Belirleme

Birinci yöntemden farklı olarak CPU tabanlı detay seviyesi belirlemede ilave olarak görünürlükte eklenmiştir. Böylelikle ekranda gözükmeyen parseller CPU detay seviyesi belirlemede daha az parçalara ayrılması sağlanmış bu sayede ekran kartında daha az arazi parseli çizilmiş olmuştur. Ayrıca GPU detay seviyesi belirlemede de ekranda görünmeyen arazi parçaları daha az detaylı olarak çizilmiş buradan da performans artışı sağlanmıştır. Şekil 4.3'de örtme sorgusu yöntemiyle tel arazi görüntüsünde neredeyse arazi parçalarının ekranda tamamı detaylı çizilirken aslında Şekil 4.4'de görüldüğü gibi ekranda çizilmeyen çoğu parsel dörtlü ağaçta daha detaylı bölünmediğinden aslında çoğu arazi parçasının az detaylı çizildiği anlaşılmaktadır. Burada dikkat edilmesi gereken ekranda gözükmeyen parselin hiç çizilmemesi yerine az detaylı olarak çizilmeye devam edilmesi. Çünkü bu yöntemde örtme sorgusunun performanslı ve gerçek geometrik şekillerle yapılmasının başarımı daha çok artırdığından dolayı çizim ile beraber paralelde örtme sorguları da ekran kartından hesaplanmaktadır. Bir önceki çizimin görünürlük verileri kullanıldığı için kamera

pozisyonu değişimlerinde ekranda gözükmeyen arazi parçalarının olmaması adına bu yöntem benimsenmiştir.



Şekil 4.3: Örtme sorgusu yöntemiyle tel arazi görünümü



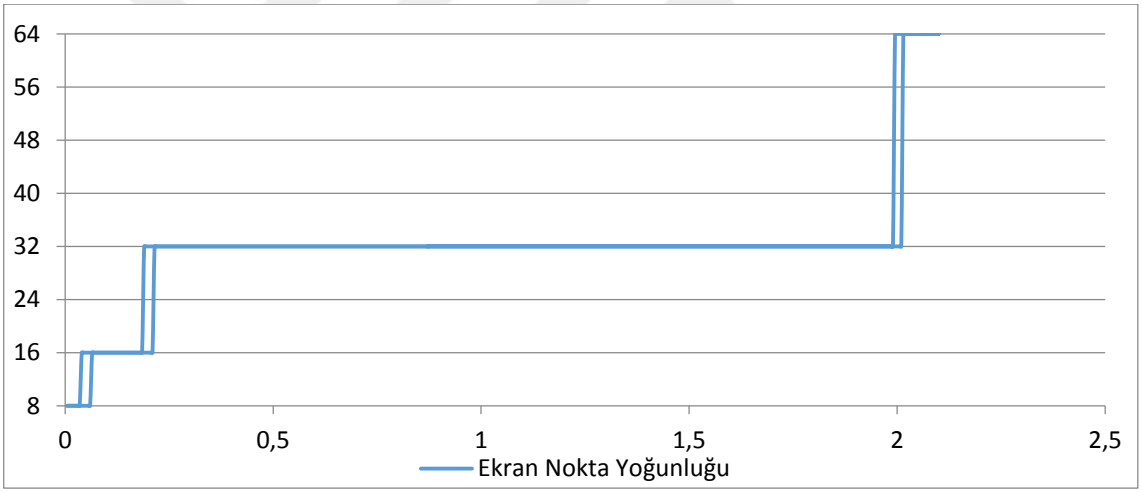
Şekil 4.4: Örtme sorgusu yöntemiyle kuş bakışı tel arazi görünümü

GPU detay seviyesi belirleme kapsamında bir önceki çizimdeki görünürlük verileri ile iç mozaikleştirme katsayısı hesaplanır. Ekranda arazi parçalarının

görünürlüklerini hesaplamak için OpenGL' in örtme sorgusu metodu kullanılmıştır. Her parça çizilmeden önce o parçaya özgü örtme sorgusu başlatılıp ilgili parsel çizilip daha sonra örtme sorgusu bitirilmektedir. Ardından diğer arazi parseline geçilerek hem örtme sorgularının paralel çalışması sağlanmış hem de çizim işlemi yapılırken aynı anda sorgu da hesaplandığından örtme sorgusu maliyeti düşürülmüştür. Tüm çizim ve sorgulama işlemleri bitirildikten sonra sorgu sonuçları alınıp arazi parçası görünürlük tablosuna sonuçlar işlenir. Daha sonra her bir arazi parçası için ekran noktası yoğunluk değeri aşağıda verilen formüle göre hesaplanır.

Ekran noktası yoğunluk değeri = toplam görünen nokta sayısı / yüzey alanı

Bu değere göre iç mozaikleştirme katsayısı belirlenirken, ekran noktası yoğunluğunda sürekli küçük değişiklikler olabileceğinden dolayı histerezis bir yapı önerilmektedir. Böylelikle iç mozaikleştirme önceki değerine göre daha tutarlı ve göze hoş gelecek şekilde değişimi sağlanmıştır. Aşağıda ekran noktası yoğunluk değerine göre iç mozaikleştirme histerezis değişimi Şekil 4.5'de gösterilmektedir.



Şekil 4.5: Ekran noktası yoğunluk değeri iç mozaikleştirme katsayısı histerezis değişimi

4.7 Yöntem-3 Ekran Noktası Akış Programı ve CUDA ile tekli LOD Belirleme

CPU ve GPU detay seviyesi belirleme yöntemi birebir ikinci yöntem ile aynı olsa da tek fark ekran noktası görünürlük hesaplamasıdır.

Bu yöntemde ekran noktası akış programında görünen nokta sayılarını hesaplamak için sayma sayısı tipinde veri dizisi barındıran akış programı tampon

nesnesi [19] kullanılmıştır. Akış programı tampon nesnesinin kullanılmasının üç sebebi vardır. Birincisi yazma işleminin atomik yapılması gerekmektedir. İkinci sebebi çalışma zamanında verinin boyutunun değişebilir olmasıdır. Üçüncü sebep ise verinin boyutu normal tampon nesnesi maksimum kapasitesini geçebilmesidir.

Bu yöntemde kullanılan akış programı tampon nesnesinin görünümü Tablo 1.1'deki gibidir. Her bir parsel için derleme esnasında belirlenen sayıda (bu örnekte 128) ekranda görülen noktaların sayısını tutmak için alan bulunur. Bunun ana sebebi ekran noktası akış programından buraya erişerek ekranda görünen nokta sayısını atomik artırmak gerektiğinden eğer burada tek bir sayı olursa paralelde yapılan her şey buradaki artırma işlemine takılır ve sıralı ilerler bu da ciddi anlamda performans kaybına yol açar. Onun yerine her bir parselin her bir üçgeni çizilirken belirlenen belirtecin ilgili parseldeki sayı adedi ile modül alınıp atomik artırılırsa çizimdeki performans düşüşü yaşanmaz. Parsel numaraları çizim sırasına göre arttırılır. Uygulamada ilave olarak tutulan arazi parça belirtecine karşılık parsel çizim numarası eşlemi bulunmaktadır.

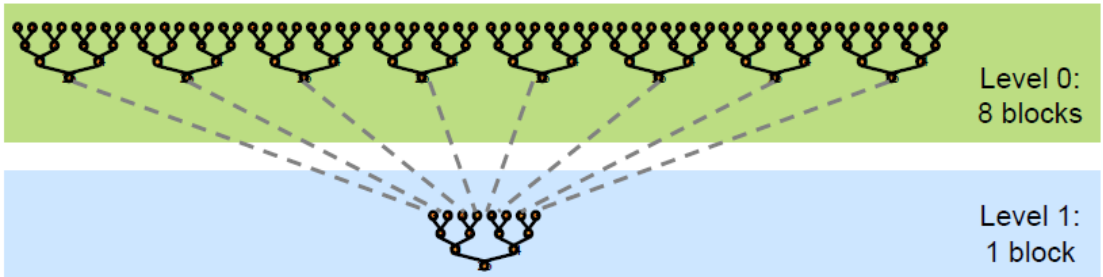
Tablo 1.1: Akış programı tampon nesnesi.

ID	Sayı	Parsel
0	0	Parça #1
1	0	
…	…	
127	0	
128	0	Parça #2
129	0	
…	…	
255	0	
256	0	Parça #3
257	0	
…	…	
383	0	
…	…	…

Böylelikle ekran noktası akış programında ilgili arazi parçasının alt bölgelerinin görünürlükleri akış programı tampon nesnesine yazılmaktadır.

Ekran kartındaki çizimlerle beraber oluşturulan akış programı tampon nesnesindeki her bir sayıyı toplamak gerekmektedir. Bu kapsamda ekran kartından bu kadar büyük bir veriyi transfer etmek yerine buradaki rakamları da ekran kartında toplayıp sadece parsel adedi kadar sayı transfer edilmesi için ekran kartı programlama dili kullanılması gerekmektedir. Bu kapsamda birden çok alternatif olsa da gerek böceklerden arındırılmasının kolay olması gerekse yazımında C'ye yakın bir dili olmasından dolayı CUDA ekran kartı programlama aracı kullanılmıştır.

Akış programı tampon nesnesindeki verileri toplamak için Mark Harris'in [25] geliştirdiği CUDA'da optimize paralel toplama işlemi yöntemi kullanılmıştır. Şekil 4.6'da örnek bir CUDA paralel toplama akışı gösterilmektedir.



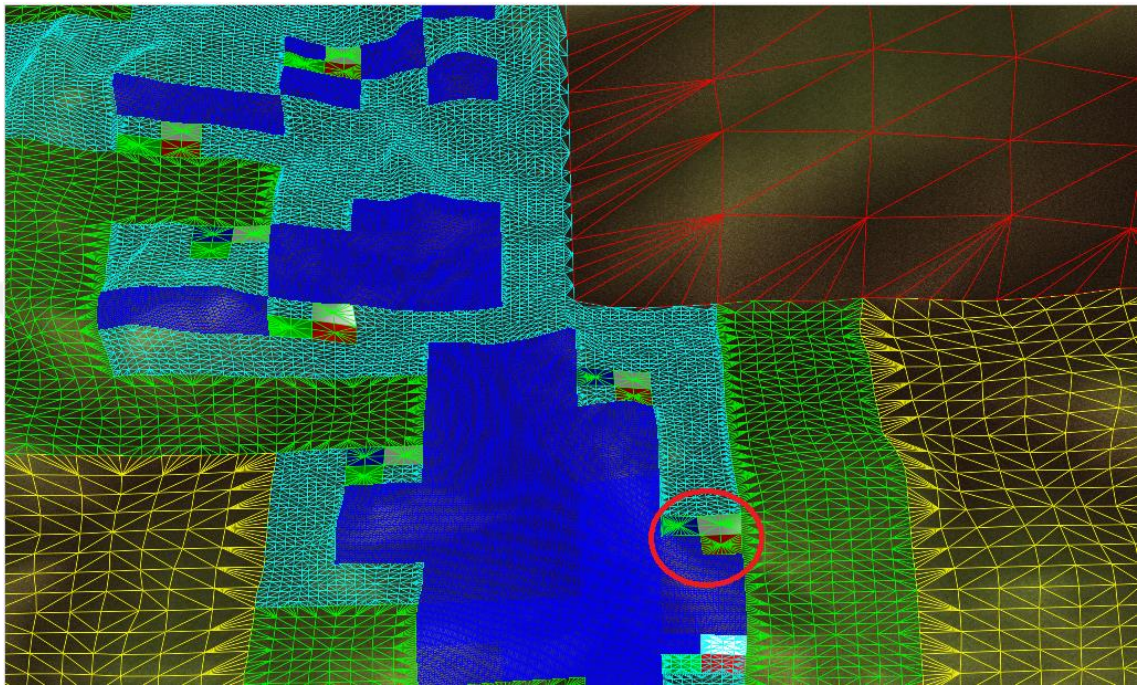
Şekil 4.6: CUDA paralel toplama işlemi akışı [25]

Daha sonra toplanan görünürlük değerleri CUDA hafızasından kopyalanarak uygulama hafızasına taşınır. Uygulama hafızasına kopyalama işleminin hızlı olması adına işletim sisteminin sayfalamasını saf dışı bırakan gene CUDA'nın hafızası kullanılmıştır. Buradaki değerler alınarak arazi görünürlük tablosuna işlenir ve ikinci yöntemdeki gibi arazi parçası ekran noktası yoğunluğuna göre iç mozaikleştirme katsayısı belirlenir.

4.8 Yöntem-4 Ekran Noktası Akış Programı ve CUDA ile dörtlü LOD Belirleme

Bu yöntemi üçüncü yöntemden ayıran ana unsur her bir arazi parçası için bir değil dört alt parçası için de görünürlük testinin yapılıyor olmasıdır. Böylelikle eğer bir alt parçası gözüküyorsa ekran noktası yoğunluğunun daha tutarlı hesaplanması adına hesaplanan değer dört ile çarpılmaktadır. Burada ana amaç performanstan çok ekranda nokta tabanlı detay seviyesi belirlemedeki başarımı artırmaya yöneliktir. Şekil

4.7’de kırmızı çember ile gösterilen kısımda sadece köşesi gözüken arazi parçasının gözüken parçası daha detaylı çizilirken gözükmeyen kısımları daha az detaylı çizilmektedir. Bu yöntemle ekranda gözüken üçgen sayısı artarken toplamda çizilen üçgen sayısı azalmaktadır. Yalnız performans olarak üçüncü yöntemden biraz daha kötü olması öngörülüyor çünkü çizilen üçgen sayısı az olsa da arazi parsel sayısı arttığından dolayı ekran kartına çizim için gidilme sayısı artmıştır.



Şekil 4.7: Dörtlü görünürlük sorgulu arazi parçası tel çizimi

4.9 Minimum donanım gereksinimi

Tüm yöntemlerde kullanılan donanımsal mozaikleştirme yöntemi OpenGL 4.0 ve üzerine ihtiyaç duymaktadır. Bununla beraber Yöntem 3 ve Yöntem 4’de önerilen akış programı tampon nesnesi minimum OpenGL 4.3 gerektirdiği için ekran kartının donanımsal olarak OpenGL 4.3 veya üstü bir versiyonu destekliyor olması gerekmektedir. Uygulama çalışırken 4 bölümünde anlatılan veriler kapsamında ekran kartındaki hafızaya yüklenecek verilerin büyüklüğü yaklaşık 100MB olacağı öngörülmektedir bu yüzden ekran kartı ve sistem hafızası kapsamında bir kısıt öngörülmemektedir.

BEŞİNCİ BÖLÜM

DENEYSEL ÇALIŞMALAR

Donanımsal mozaikleştirme yöntemiyle arazi görselleştirmesi kapsamında önerilen detay seviyesi belirleme yöntemlerinin performans ölçümlerinin nasıl yapılacağı anlatılacaktır.

5.1 Testlerin Koşacağı Bilgisayar

Uygulamaların çalışacağı donanımı belirlemesi kapsamında 4.9'de anlatılan minimum gereksinimleri sağlaması en önemli kriterdi. Bu kapsamda seçilen testlerin koşacağı bilgisayar aşağıdaki donanıma ve yazılama sahiptir.

- Windows 8.1 Pro 64bit
- Intel Core i5-4200H CPU @ 2.80 GHz
- 12 GB RAM
- NVIDIA GeForce GTX 950M
- Visual Studio 2013
- C++
- CUDA 7.5

5.2 Testlerin Koşacağı Arazi

Testlerin koşacağı arazi 4'de belirtildiği gibi büyük kanyon (80km x 40 km) seçilmiştir. Fakat bu arazi çok büyük olduğu için testler kapsamında tüm arazi gezilemeyeceği için testler kapsamında küçük bölümleri gezilmiştir. Bu kapsamda büyük kanyonda iki ana senaryo belirlenmiştir. Birinci senaryoda bakış açısına daha çok arazi girecek şekilde ilerleyerek performans ölçümleri alınmıştır. İkinci senaryoda ise arazi üzerindeki bir tepeye doğru ilerlenerek bakış açısına giren arazi azalmakta daha sonra tepeden uzaklaşılarak tekrar bakış açısına daha fazla arazinin girmesi sağlanmıştır.

Birinci senaryoda daha az örtülen arazi parçası varken ikinci senaryonun ortalarında neredeyse arazinin tamamı örtülmekte ve ekrana arazinin küçük bir parçası çizilmektedir. Böylelikle denenen metotlar arasındaki farklar, iki farklı senaryoda nasıl değişiklik gösterileceği tespit edilmeye çalışılmıştır.

5.3 Test Senaryolarının oluşturulması

Tez kapsamında geliştirilen uygulamaya arazi üzerinde gezinirken her bir çizim öncesi kamera pozisyonunun kaydedilebilmesi özelliği eklenmiştir. Kayıt işlemini başlatmak ve bitirmek için bir kısa yol tuşu atanmıştır. Böylelikle testlerin koşacağı iki senaryoya da uygun bölgeler tespit edilerek kayıt işlemi başlatılmış arazi üzerinde gezinilip her bir çizim öncesi kamera pozisyonu harici bir dosyaya kayıt edilmiştir.

5.4 Testler

Yöntemler arasındaki performans farklılıkların tutarlı bir şekilde tespit edilebilmesi için deneylerin kontrollü bir şekilde yapılmasına önem gösterilmiştir. Yani her bir farklı deneyde sadece bir parametre değiştirilerek diğer parametrelerin aynı kalması sağlandı. Böylelikle değiştirilen parametrenin deneyi nasıl değiştirdiği anlaşılmaya çalışılmıştır. Deneyler kapsamında değiştirilecek parametreler; iç mozaikleştirme katsayısı, dörtlü ağaç parçası bölme katsayısı ve Yöntem-3 ile Yöntem-4'de kullanılan akış noktası tampon nesnesi büyüklüğü parametresidir.

5.4.1 İç mozaikleştirme testi

GPU'daki detay seviyesi belirlemede iç mozaikleştirme katsayısı büyük önem taşımaktadır. Ekran noktası yoğunluğuna göre belirlenecek detay seviyesinde bu çarpan kullanılmaktadır. Testler kapsamında varsayılan değeri 32 olarak belirlenip eğer bu değer 16 veya 64 olsaydı neler olacağına bakılmıştır. İç mozaikleştirme katsayısı arttıkça ekranda çizilen arazi parçalarının detayı artmakta azaldıkça da azalmaktadır.

5.4.2 Dörtlü ağaç parçası bölme testi

CPU'daki detay seviyesi belirlemede dörtlü ağaç parçası bölme katsayısı büyük önem taşımaktadır. Kameradan uzaklaşıldıkça görünen arazi parçalarının daha detaylı olup olmayacağı kararında bakılan katsayıdır. İlgili katsayı arttıkça arazi parçalarının

sayısı artmakta azaldıkça da azalmaktadır. Testler kapsamında varsayılan değeri 10 olarak belirlenip eğer bu değer 5 veya 15 olsaydı neler olacağına bakılmıştır. Arazi parçalarının sayısının artması hem çizim için GPU'ya gidilme sayını arttırdığından hem de örtme sorgusu yapma sayısını artırdığından performansa olumsuz etkileri olacağı öngörülmektedir. Bununla beraber parsellerin küçülmesinden dolayı ekranda çizilen parsel sayısı artsa da çizilen üçgen sayılarının azalmasından kaynaklanan bir performans artışı da beklenmektedir. Deneyler sonucunda nasıl bir etkisi olduğu daha detaylı değerlendirilecektir.

5.4.3 Akış programı tampon nesnesi büyüklük katsayısı testi

Yöntem-3 ve Yöntem-4'de kullanılan akış programı tampon nesnesinin büyüklüğünün performansa etkisinin olup olmadığının anlaşılması için testlere ihtiyaç duyulmuştur. Testler kapsamında varsayılan değeri 32 olarak belirlenip eğer bu değer 16 veya 64 olsaydı neler olacağına bakılmıştır. Testler öncesinde ekran noktası çizim programı akışında atomik yazma işlemini hızlandırması kapsamında bu değerin olabildiğince büyümesi gerekirken CUDA tarafında bunun okunup CPU hafızasına yazılması kapsamında da bu değerin olabildiğince küçük olması gerekmektedir. Bu test ile ideal değeri tespit edilmeye çalışılmıştır.

5.5 Testlerin Başlatılması

Testlerin çalıştırılması kapsamında işlemleri otomatikleştirme kapsamında komut dosyası kullanılmıştır. Her bir senaryo kapsamında ilgili senaryoya ait kamera pozisyonları dosyası değiştirilerek komut dosyası çalıştırılmaktadır. Komut dosyasında farklı parametrelerle uygulama çalıştırma komutları yer almaktadır. Uygulama çalıştırılırken geçilen dört parametre bulunmaktadır. Birinci parametre yöntemi ifade etmektedir. Yöntem-1, Yöntem-2, Yöntem-3 ve Yöntem-4 için sıfırdan dörtte numaralar kullanılmıştır. İkinci parametre iç mozaikleştirme katsayısını belirtmektedir. Senaryolar kapsamında 16, 32 ve 64 değerleri ile çalıştırıldı. Varsayılan değeri 32 olarak belirlendi. Üçüncü parametre dörtlü ağaç parçası bölme katsayısını belirtmektedir. Senaryolar kapsamında 5, 10 ve 15 değerleri ile çalıştırıldı. Varsayılan değeri 10 olarak belirlendi. Dördüncü parametre ise akış programı tampon nesnesinin büyüklüğünü belirtmektedir. Senaryolar kapsamında 16, 32 ve 64 değerleri ile çalıştırıldı. Varsayılan değeri 32 olarak belirlendi.

5.6 Testlerin Tutarlılığının Sağlanması

Testler sırasında performans ölçümleri alınırken testleri etkileyecek dış etkenler minimize edilmeye çalışılmıştır. Bu kapsamda ağ iletişimini tamamen kapatmak için uçak modu açılmıştır. Arka planda çalışan uygulamalar kapatılmıştır. Ekran kartında enerji koruma ayarlarında performans öncelikli hale getirilmiştir. Ekran kartından optimizasyonları engellemek adına her bir çizimde kameranın sürekli hareket halinde olmasına özen gösterilmiştir. Tüm bunlara ilave olarak tüm koşumlar 30 kere tekrarlanarak performans sonuçları bu 30 koşumun ortalaması olarak belirlenmiştir.

5.7 Testlerin Sonuçları

Her bir koşum sonrası performans sonuçları uygulama çalışırken parametre olarak verilen yöntem numarası, iç mozaikleştirme katsayısı, dörtlü ağaç parçası bölme katsayısı ve akış programı tampon nesnesi büyüklüğü değerlerini içeren bir dosya adıyla kayıt edilmektedir. Her bir satırda aşağıdaki üç bilgiyi içermektedir.

- Çizim süresi (milisaniye cinsinden)
- Çizilen arazi parçası sayısı
- Çizilen üçgen sayısı

ALTINCI BÖLÜM

DENEYLER VE SONUÇLAR

Bu bölümde Yöntem-1, Yöntem-2, Yöntem-3 ve Yöntem-4 kullanılarak iki farklı senaryo kapsamında yapılan testler ve elde edilen sonuçları değerlendirilecektir. Testler kapsamında değiştirilmeyen parametreler için hep varsayılan değerler kullanılmıştır. İç mozaikleştirme katsayısı için 32, dörtlü ağaç parçası bölme katsayısı olarak 10 ve akış programı tampon nesnesi büyüklüğü de 32 olarak kabul edilmiştir.

6.1. Engelsiz Arazi Senaryosu Deneyleri

Bu bölümde engelsiz arazi senaryosu kapsamında yapılan testler ve sonuçlarına yer verilecektir.

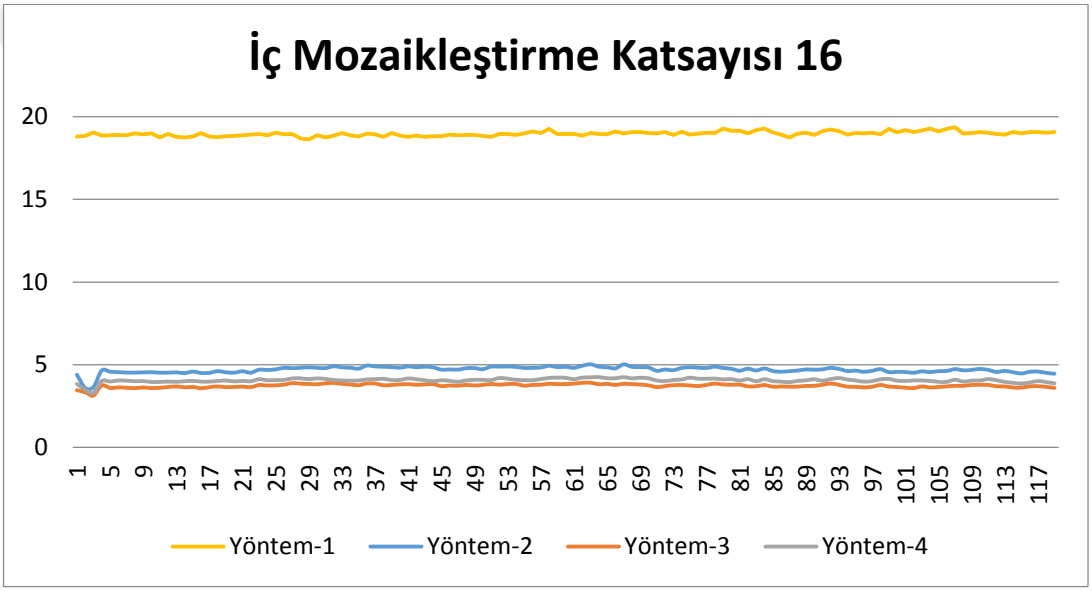
6.1.1. Engelsiz Arazi Senaryosu İç Mozaikleştirme Katsayısı

İç mozaikleştirme katsayı değeri değiştirilerek diğer parametreler için varsayılan değerler kullanılmıştır.

İç mozaikleştirme katsayısı 16 olarak kabul edilerek 4 yöntem için elde edilen sonuçlar Şekil 6.1'de ve Tablo 6.1'de gösterilmektedir. Yöntem-2 ortalamada Yöntem-1'den 4,0 kat daha performanslı çıkarken, Yöntem-4 ise Yöntem-2'den %15,6 daha performanslı, Yöntem-3'de Yöntem-4'den %8,8 daha performanslı olmaktadır. İç mozaikleştirme katsayısı düşük seçildiğinde çizim için harcanan süre az olduğundan örtme sorgu süreleri ciddi önem kazanmaktadır.

İç mozaikleştirme katsayısı 32 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.2'de ve Tablo 6.2'de gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-4 ortalamada Yöntem-2'den %6,9 daha performanslı iken Yöntem-3'de Yöntem-4'den %3,4 daha performanslı olmaktadır.

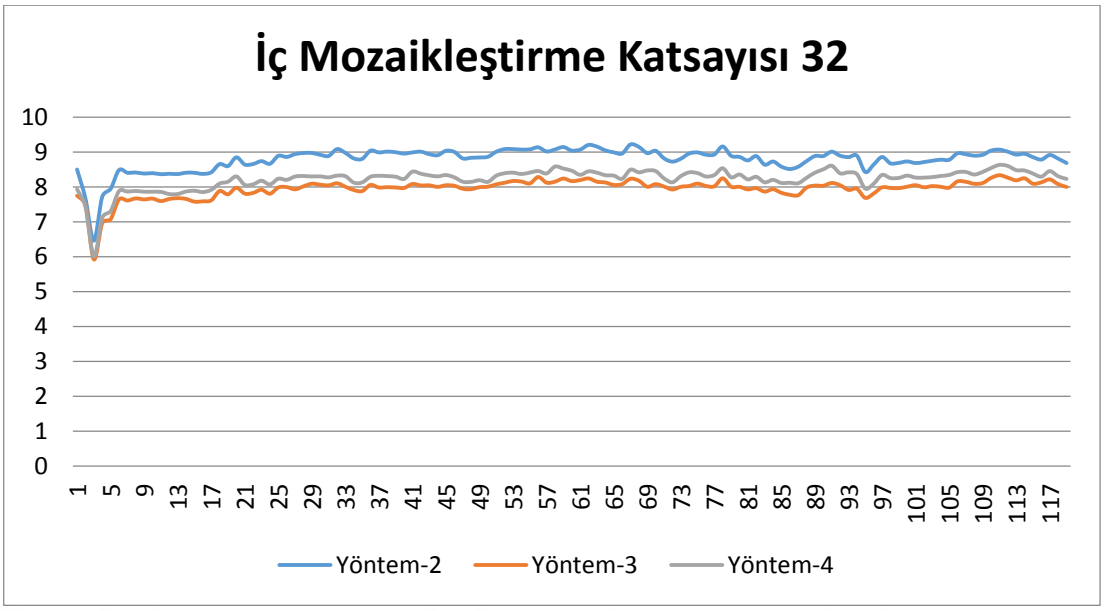
İç mozaikleştirme katsayısı 64 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.3'de ve Tablo 6.3'de gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-3 ortalamada Yöntem-2'den %1,5 daha performanslı iken Yöntem-4'de Yöntem-3'den %1,8 daha performanslı olmaktadır. Burada dikkat çeken şey ilk defa Yöntem-4 Yöntem-3'den daha performanslı çıkmıştır. Çünkü iç mozaikleştirme katsayısı 64 demek çizim işleminin çok uzun süre alması demek oluyor bu da Yöntem-4 sayesinde daha az üçgen çizilmesinin performansa katkısının daha hissedilir olmasına yol açmaktadır. Ayrıca çizimin çok uzun süre almasından dolayı da performans farkları ciddi manada kapanmıştır.



Şekil 6.1: Engelsiz Arazi Senaryosu İç Mozaikleştirme Katsayısı 16

Tablo 6.1: Engelsiz Arazi Senaryosu İç Mozaikleştirme 16 Ortalama Değerleri.

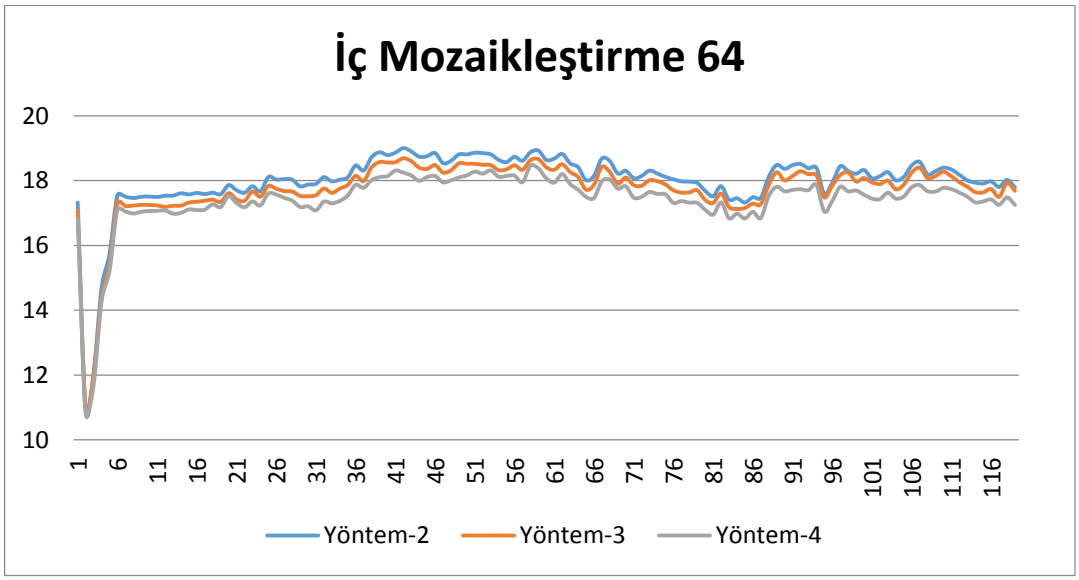
	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	18,97	2505	1282648
Yöntem-2	4,70	478	126032
Yöntem-3	3,73	478	126032
Yöntem-4	4,06	618	125221



Şekil 6.2: Engelsiz Arazi Senaryosu İç Mozaikleştirme Katsayısı 32

Tablo 6.2: Engelsiz Arazi Senaryosu İç Mozaikleştirme 32 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	8,80	482	505129
Yöntem-3	7,96	482	505129
Yöntem-4	8,23	628	500207



Şekil 6.3: Engelsiz Arazi Senaryosu İç Mozaikleştirme 64

Tablo 6.3: Engelsiz Arazi Senaryosu İç Mozaikleştirme 64 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	17,99	482	2026373
Yöntem-3	17,73	482	2026373
Yöntem-4	17,42	634	1991280

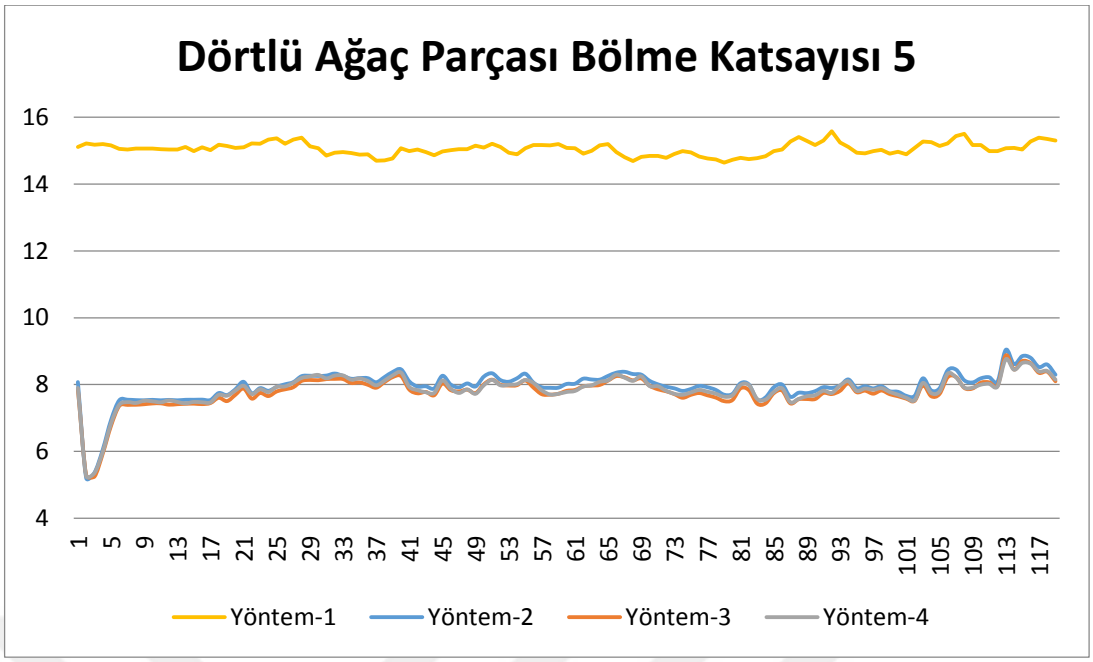
6.1.2. Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı

Dörtlü ağaç parçası bölme katsayı değeri değiştirilerek diğer parametreler için varsayılan değerler kullanılmıştır.

Dörtlü ağaç parçası bölme katsayı 5 olarak kabul edilerek 4 yöntem için elde edilen sonuçlar Şekil 6.4'de ve Tablo 6.4'de gösterilmektedir. Yöntem-2 ortalamada Yöntem-1'den 1,9 kat daha performanslı çıkarken Yöntem-4 ise Yöntem-2'den %1,3 daha performanslı, Yöntem-3'de Yöntem-4'den %0,7 daha performanslı olmaktadır. Dörtlü ağaç parçası bölme katsayısı düşük seçildiğinde oluşan dörtlü ağaç parçası az olduğundan bu da örtme sorgusu yapılacak parsel sayısının az olmasına yol açmaktadır. Bu da yeni geliştirilen Yöntem-3 ve Yöntem-4'ün performans artışlarının çok az olmasına yol açmaktadır.

Dörtlü ağaç parçası bölme katsayısı 10 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.5'de ve Tablo 6.5'de gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-4 ortalamada Yöntem-2'den %6,9 daha performanslı iken Yöntem-3'de Yöntem-4'den %3,4 daha performanslı olmaktadır.

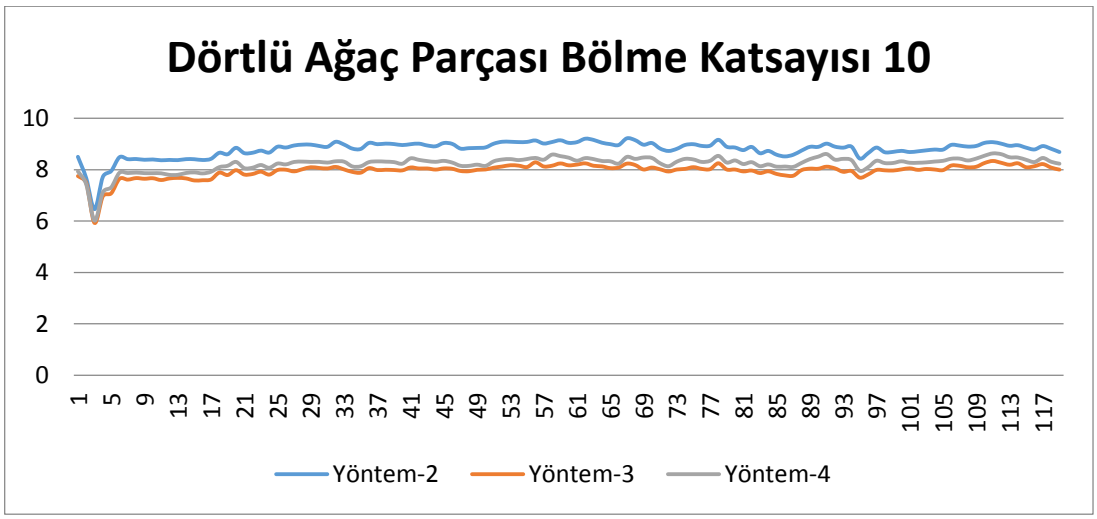
Dörtlü ağaç parçası bölme katsayısı 15 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.6'da ve Tablo 6.6'da gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-4 ortalamada Yöntem-2'den %12,9 daha performanslı iken Yöntem-3'de Yöntem-4'den %4,8 daha performanslı olmaktadır. Ağaç parçası sayısı çok fazla olduğundan dolayı yeni geliştirilen Yöntem-3 ve Yöntem-4'ün performans artışları daha belirgin olmaktadır.



Şekil 6.4: Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 5

Tablo 6.4: Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 5 Ortalama Değerleri.

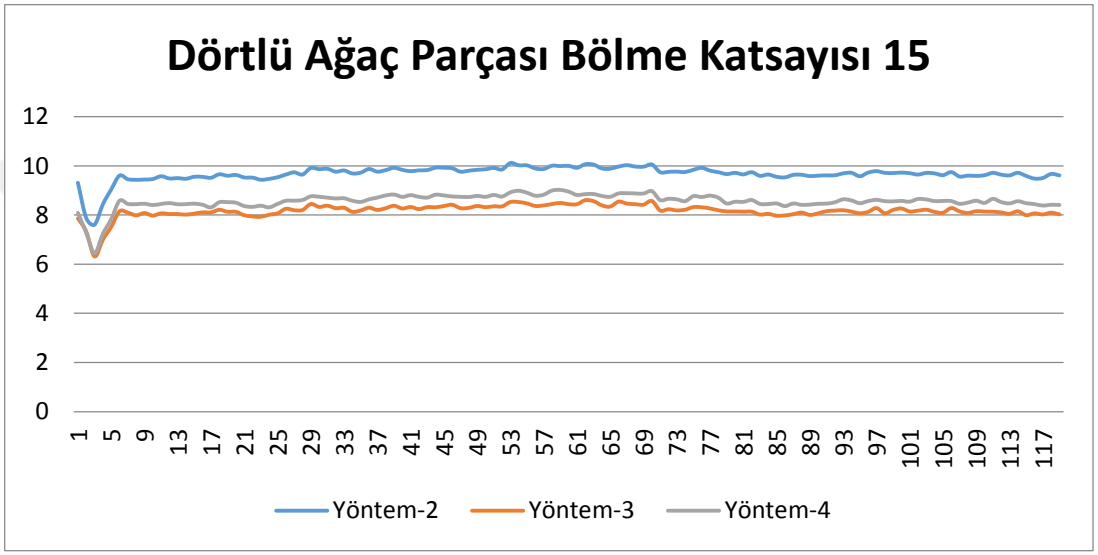
	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	15,06	763	1562939
Yöntem-2	7,94	191	551657
Yöntem-3	7,78	191	551657
Yöntem-4	7,84	230	547037



Şekil 6.5: Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 10

Tablo 6.5: Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 10 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	8,80	482	505129
Yöntem-3	7,96	482	505129
Yöntem-4	8,23	628	500207



Şekil 6.6: Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 15

Tablo 6.6: Engelsiz Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 15 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	9,68	816	430846
Yöntem-3	8,17	816	430846
Yöntem-4	8,57	1036	422945

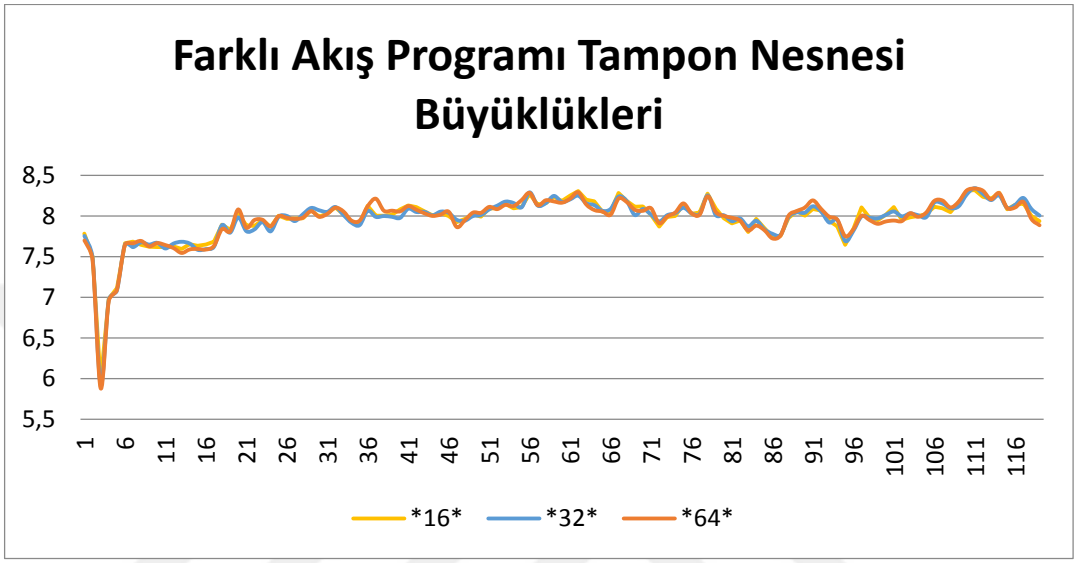
6.1.3. Engelsiz Arazi Senaryosu Akış Programı Tampon Nesnesi Büyüklüğü

Akış programı tampon nesnesi büyüklüğü değiştirilerek diğer parametreler için varsayılan değerler kullanılmıştır.

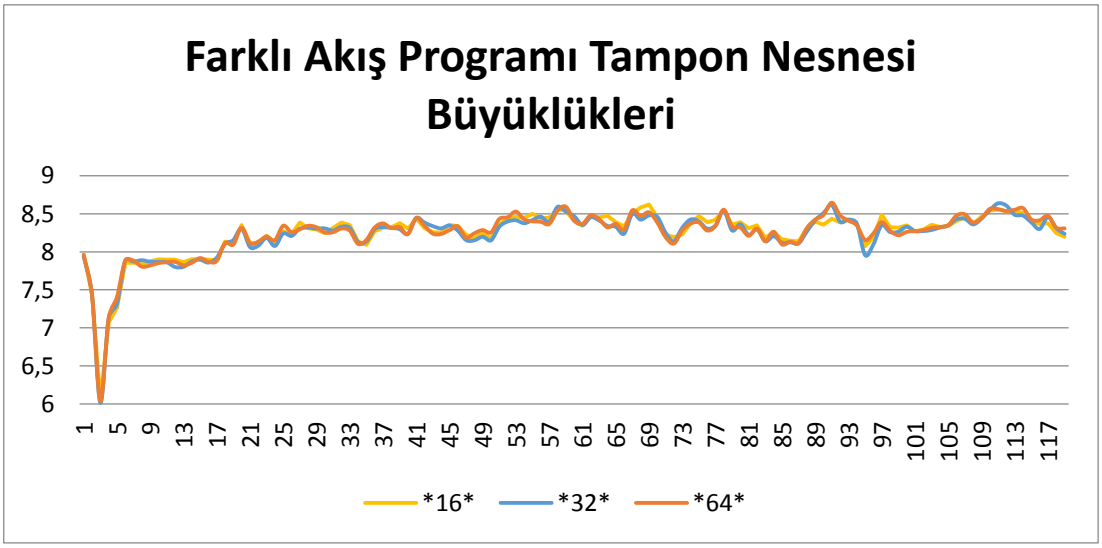
Yöntem-3 kullanılarak akış programı tampon nesnesi büyüklükleri değiştirilerek performans kıyaslaması yapılmıştır. Şekil 6.7'de bu kıyaslama sonucu

gösterilmektedir. Elde edilen sonuçlar ışığında akış programı tampon nesnesi büyüklüğü 32, 16 ve 64 arasında ciddi farklar gözlemlenememiştir.

Yöntem-3 kullanılarak akış programı tampon nesnesi büyüklükleri değiştirilerek performans kıyaslaması yapılmıştır. Şekil 6.8'de bu kıyaslama sonucu gösterilmektedir. Elde edilen sonuçlar ışığında akış programı tampon nesnesi büyüklüğü 32, 16 ve 64 arasında ciddi farklar gözlemlenememiştir.



Şekil 6.7: Engelsiz Arazi Senaryosu Yöntem-3 İle Farklı Akış Programı Tampon Nesnesi Büyüklükleri



Şekil 6.8: Engelsiz Arazi Senaryosu Yöntem-4 İle Farklı Akış Programı Tampon Nesnesi Büyüklükleri

6.2. Engelli Arazi Senaryosu Deneyleri

Bu bölümde engelli arazi senaryosu kapsamında yapılan kontrollü deneyler ve sonuçlarına yer verilecektir.

6.2.1. Engelli Arazi Senaryosu İç Mozaikleştirme Katsayısı

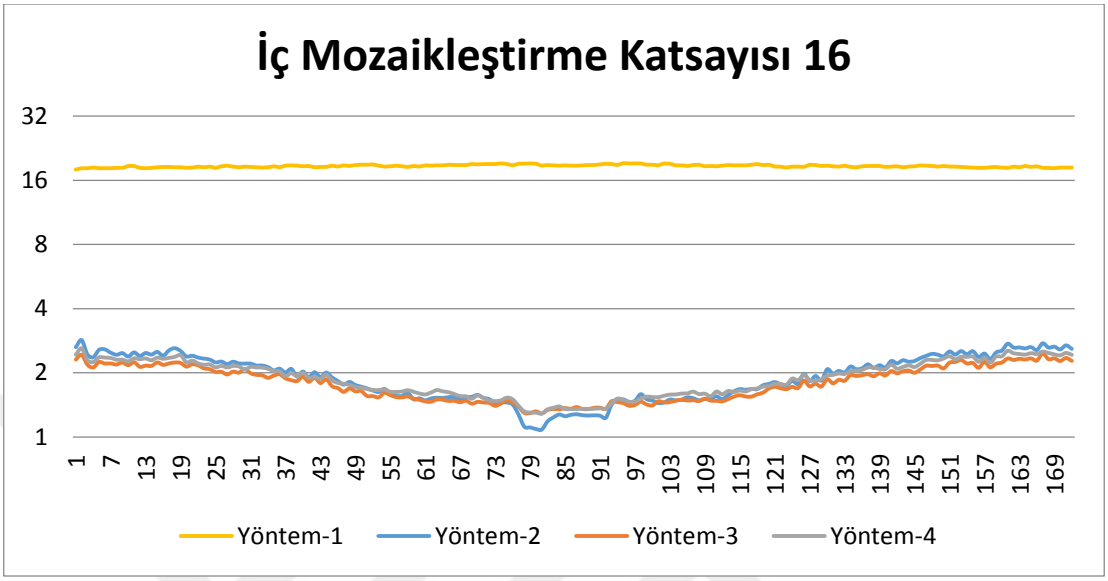
İç mozaikleştirme katsayı değeri değiştirilerek diğer parametreler için varsayılan değerler kullanılmıştır.

İç mozaikleştirme katsayısı 16 olarak kabul edilerek 4 yöntem için elde edilen sonuçlar Şekil 6.9'da ve Tablo 6.7'de gösterilmektedir. Yöntem-2 ortalamada Yöntem-1'den 9,6 kat daha performanslı çıkarken Yöntem-4 ise Yöntem-2'den %1,7 daha performanslı, Yöntem-3'de Yöntem-4'den %5,6 daha performanslı olmaktadır. İç mozaikleştirme katsayısı düşük seçildiğinde çizim için harcanan süre az olduğundan örtme sorgu süreleri ciddi önem kazanmaktadır. Ayrıca bir engel olduğunda ekrana çizilen parsel sayısında önemli miktarda düşme olduğundan Şekil 6.9'da 80. çerçevede Yöntem-2 ilk defa yeni geliştirilen yöntemlerden daha performanslı çıkmıştır. Bunun ana sebebi de CUDA işlemlerinin belli bir minimum maliyetinin olmasından kaynaklanmaktadır.

İç mozaikleştirme katsayısı 32 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.10'da ve Tablo 6.8'de gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-4 ortalamada Yöntem-2'den %1,7 daha performanslı iken Yöntem-3'de Yöntem-4'den %3,3 daha performanslı olmaktadır. Buradaki değerler de engelsiz arazi senaryosuna göre yeni geliştirilen yöntemlerin performans artışlarında bir azalma söz konusudur. Bunun ana sebebi de engel varken ekrana çizilen arazi parseli az olduğundan performans artışını olumsuz etkilemektedir.

İç mozaikleştirme katsayısı 64 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.11'de ve Tablo 6.9'da gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-3 ortalamada Yöntem-2'den %0,8 daha performanslı iken Yöntem-4'de Yöntem-3'den çok çok az da olsa performanslı olmaktadır. Burada dikkat çeken şey ilk defa Yöntem-4 Yöntem-3'den daha performanslı çıkmıştır çünkü iç mozaikleştirme katsayısı 64 demek çizim işleminin çok uzun süre alması demek oluyor bu da Yöntem-4 sayesinde daha az üçgen çizilmesinin performansa olan katkısının daha hissedilir olmasına yol

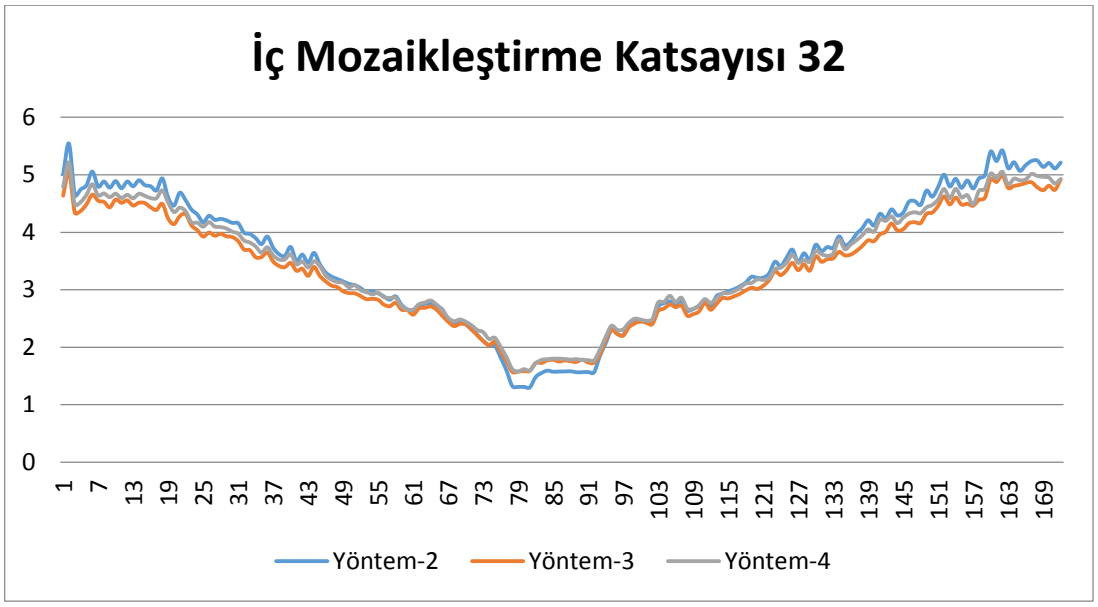
açmaktadır. Ayrıca çizimin çok uzun süre almasından dolayı da performans farkları ciddi manada kapanmıştır. Ayrıca engel olduğundan dolayı da farklar engelsiz arazi senaryosuna göre yeni geliştirilen yöntemlerin performans artışlarının iyice düşmesine yol açmıştır.



Şekil 6.9: Engelli Arazi Senaryosu İç Mozaikleştirme Katsayısı 16

Tablo 6.7: Engelli Arazi Senaryosu İç Mozaikleştirme Katsayısı 16 Ortalama Değerleri.

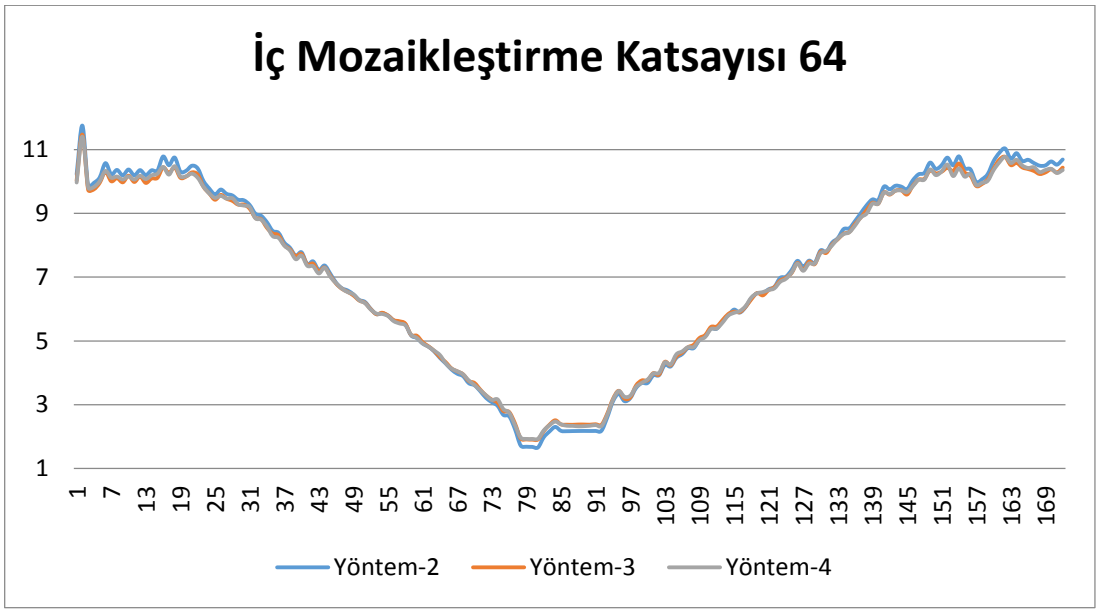
	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	18,67	2399	1228398
Yöntem-2	1,94	196	38941
Yöntem-3	1,81	196	38941
Yöntem-4	1,91	231	38618



Şekil 6.10: Engelli Arazi Senaryosu İç Mozaikleştirme Katsayısı 32

Tablo 6.8: Engelli Arazi Senaryosu İç Mozaikleştirme Katsayısı 32 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	3,54	208	159141
Yöntem-3	3,37	208	159141
Yöntem-4	3,48	244	157161



Şekil 6.11: Engelli Arazi Senaryosu İç Mozaikleştirme 64

Tablo 6.9: Engelli Arazi Senaryosu İç Mozaikleştirme Katsayısı 64 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	7,16	208	638197
Yöntem-3	7,11	208	638197
Yöntem-4	7,10	244	630960

6.2.2. Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı

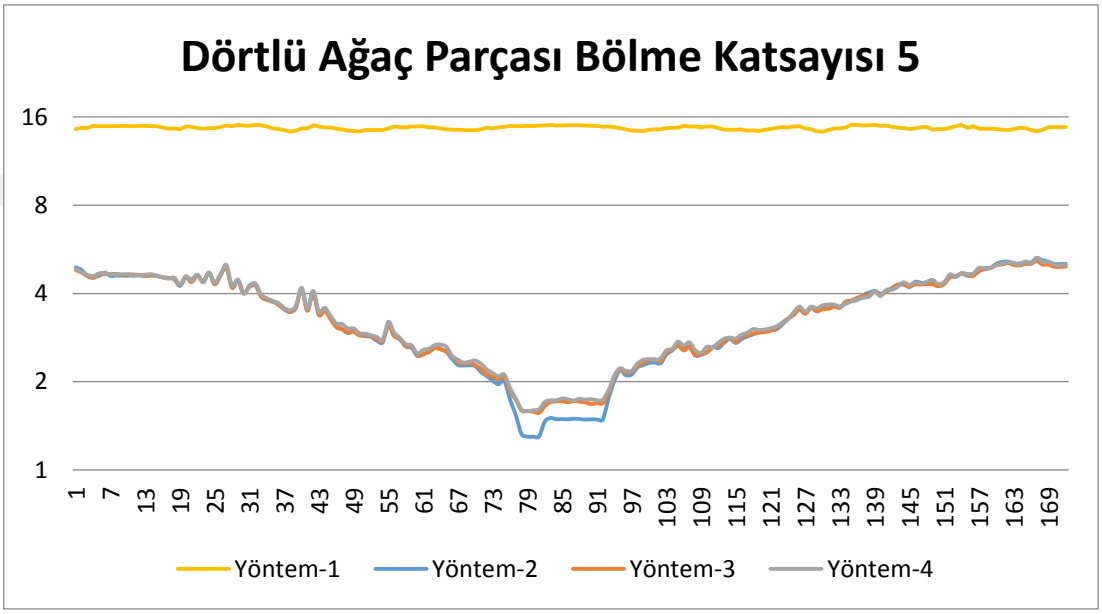
Dörtlü ağaç parçası bölme katsayı değeri değiştirilerek diğer parametreler için varsayılan değerler kullanılmıştır.

Dörtlü ağaç parçası bölme katsayı 5 olarak kabul edilerek 4 yöntem için elde edilen sonuçlar Şekil 6.12'de ve Tablo 6.10'da gösterilmektedir. Yöntem-4 ortalamada Yöntem-1'den 4,2 kat daha performanslı çıkarken Yöntem-3 ise Yöntem-4'den %1,2 daha performanslı, Yöntem-2'de Yöntem-3'den %0,5 daha performanslı olmaktadır. Dörtlü ağaç parçası bölme katsayısı düşük seçildiğinde oluşan dörtlü ağaç parçası az olduğundan bu da örtme sorgusu yapılacak parsel sayısının az olmasına yol açmaktadır. Bu da yeni geliştirilen Yöntem-3 ve Yöntem-4'ün performans artışlarının çok az olmasına yol açmaktadır. Bunlara ilave olarak engelli senaryodaki deneyin ortasında çok az arazi parçası çizilmesi ile CUDA'nın minimum maliyetinden dolayı tüm testler arasında ilk defa ortalamada çok az da olsa yeni geliştirilen Yöntem-3 ve Yöntem-4 Yöntem-2'nin arkasında kalmıştır. Tabi %0,7 bir fark belli iki üç parametrenin bir araya gelmesiyle olduğundan dolayı ihmal edilebilir.

Dörtlü ağaç parçası bölme katsayısı 10 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.13'de ve Tablo 6.11'de gösterilmiştir. Yöntem-1 değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-4 ortalamada Yöntem-2'den %1,8 daha performanslı iken Yöntem-3'de Yöntem-4'den %3,3 daha performanslı olmaktadır. Buradaki değerler de engelsiz arazi senaryosuna göre yeni geliştirilen yöntemlerin performans artışlarında bir azalma söz konusudur. Bunun ana sebebi de engel varken ekranda çizilen arazi parseli az olduğundan performans artışını olumsuz etkilemektedir.

Dörtlü ağaç parçası bölme katsayısı 15 olarak kabul edilerek 3 yöntem için elde edilen sonuçlar Şekil 6.14'de ve Tablo 6.12'de gösterilmiştir. Yöntem-1

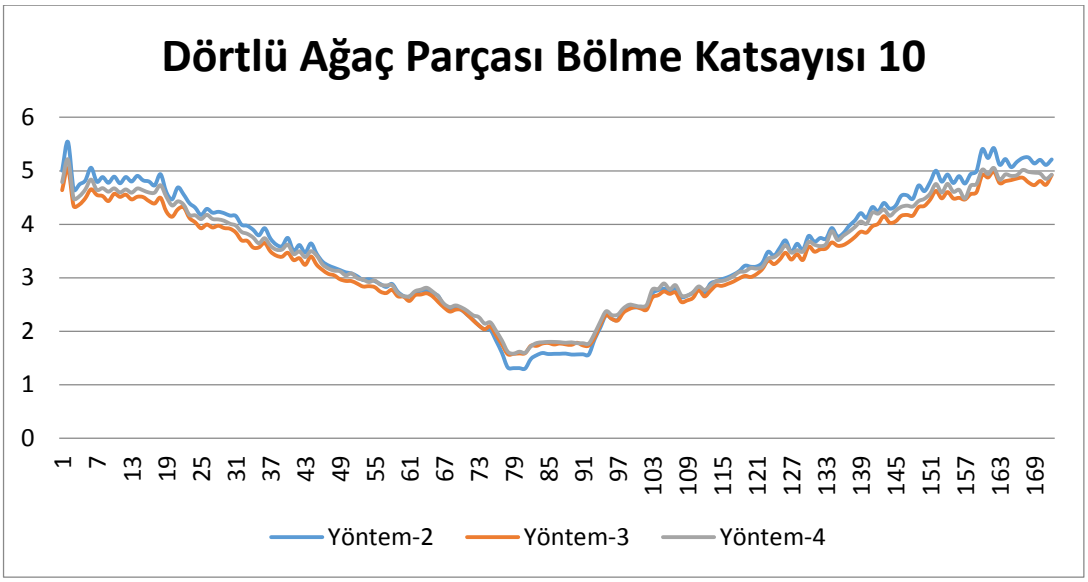
değerlendirme dışında tutulmuştur çünkü performansı diğer yöntemlerden çok fazla kötü olduğu için diğer yöntemler arasındaki farkların daha iyi gözlemlenmesi sağlanmıştır. Yöntem-4 ortalamada Yöntem-2'den %6,8 daha performanslı iken Yöntem-3'de Yöntem-4'den %3,0 daha performanslı olmaktadır. Ağaç parçası sayısı çok fazla olduğundan dolayı yeni geliştirilen Yöntem-3 ve Yöntem-4'ün performans artışları daha belirgin olmaktadır. Ancak engelsiz arazi senaryosuna görece yeni geliştirilen Yöntem-3 ve Yöntem-4'ün performansı o kadar artmadığı değerlendirilmektedir.



Şekil 6.12: Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 5

Tablo 6.10: Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 5 Ortalama Değerleri.

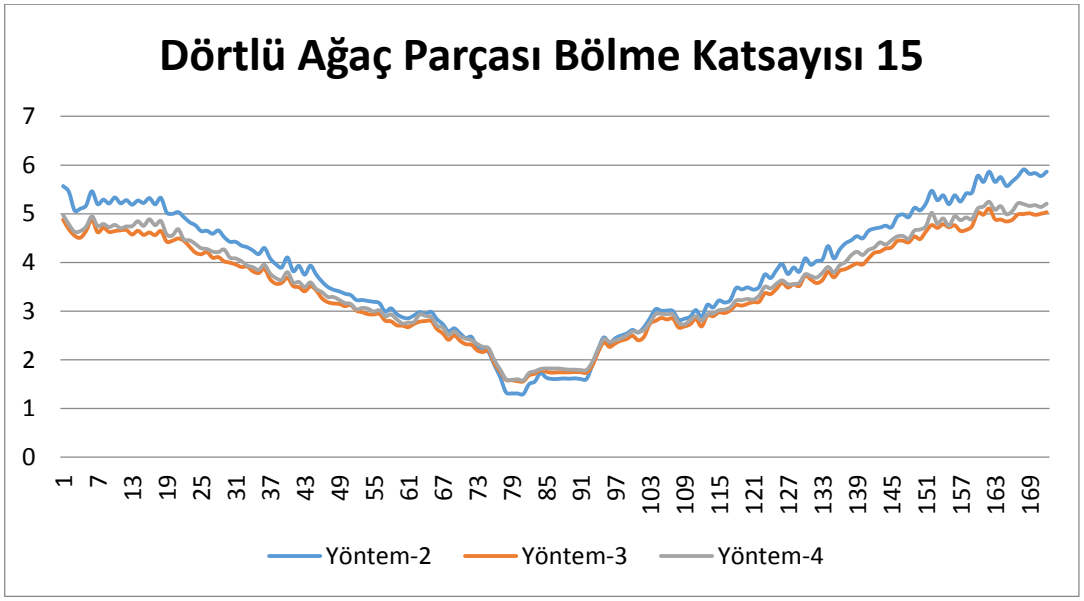
	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	14,70	750	1536990
Yöntem-2	3,41	114	191276
Yöntem-3	3,43	114	191276
Yöntem-4	3,47	138	188486



Şekil 6.13: Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 10

Tablo 6.11: Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 10 Ortalama Değerleri.

	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	3,54	208	159141
Yöntem-3	3,37	208	159141
Yöntem-4	3,48	244	157161



Şekil 6.14: Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 15

Tablo 6.12: Engelli Arazi Senaryosu Dörtlü Ağaç Parçası Bölme Katsayısı 15 Ortalama Değerleri.

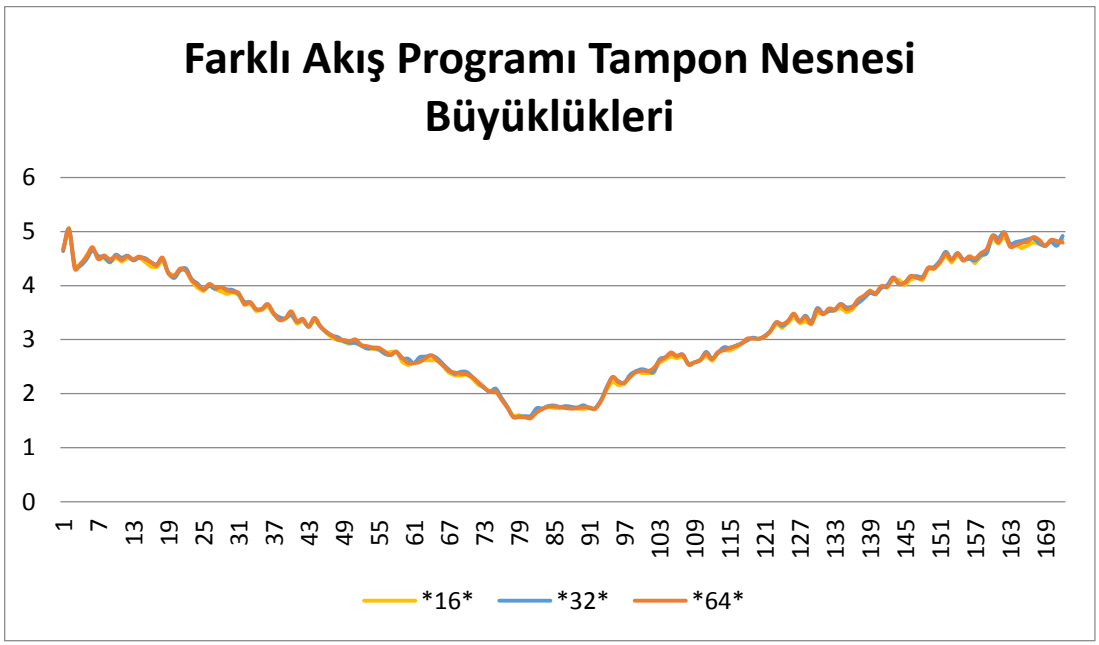
	Çizim Süresi (ms)	Arazi Parça Sayısı	Üçgen Sayısı
Yöntem-1	UD	UD	UD
Yöntem-2	3,83	271	153230
Yöntem-3	3,48	271	153230
Yöntem-4	3,59	316	151470

6.2.3. Engelli Arazi Senaryosu Akış Programı Tampon Nesnesi Büyüklüğü

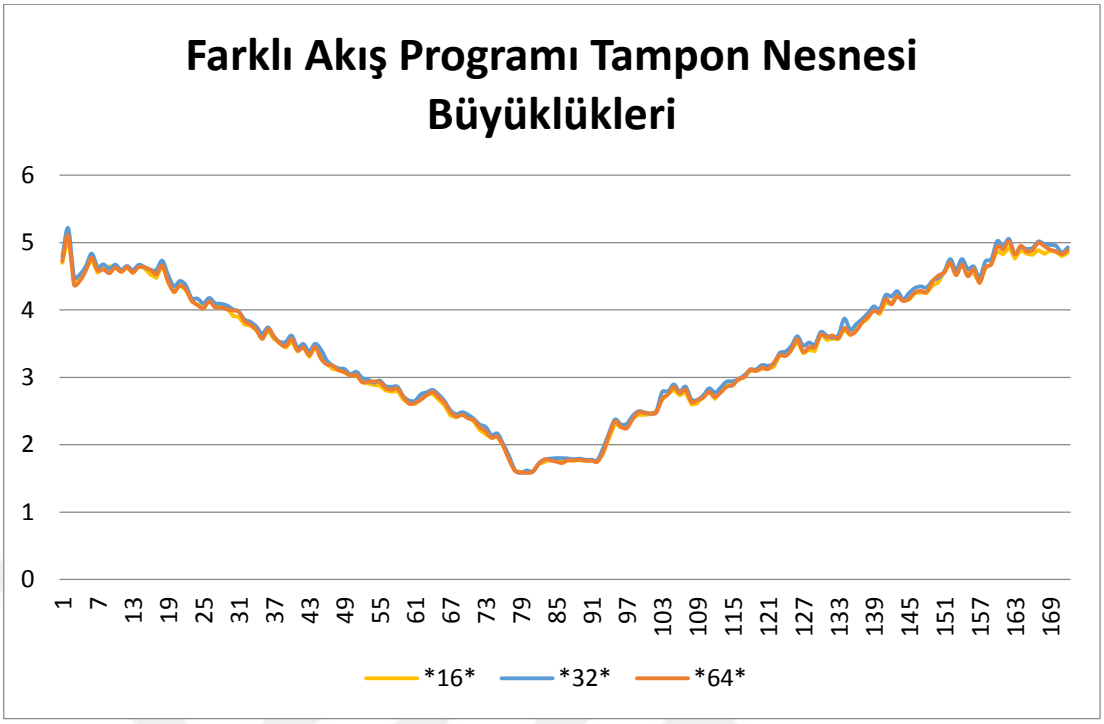
Akış programı tampon nesnesi büyüklüğü değiştirilerek diğer parametreler için varsayılan değerler kullanılmıştır.

Yöntem-3 kullanılarak akış programı tampon nesnesi büyüklükleri değiştirilerek performans kıyaslaması yapılmıştır. Şekil 6.15'de bu kıyaslama sonucu gösterilmektedir. Elde edilen sonuçlar ışığında akış programı tampon nesnesi büyüklüğü 32, 16 ve 64 arasında ciddi farklar gözlemlenememiştir.

Yöntem-3 kullanılarak akış programı tampon nesnesi büyüklükleri değiştirilerek performans kıyaslaması yapılmıştır. Şekil 6.16'da bu kıyaslama sonucu gösterilmektedir. Elde edilen sonuçlar ışığında akış programı tampon nesnesi büyüklüğü 32, 16 ve 64 arasında ciddi farklar gözlemlenememiştir.



Şekil 6.15: Engelli Arazi Senaryosu Yöntem-3 İle Farklı Akış Programı Tampon Nesnesi Büyüklükleri



Şekil 6.16: Engelli Arazi Senaryosu Yöntem-4 İle Farklı Akış Programı Tampon Nesnesi Büyüklükleri

YEDİNCİ BÖLÜM

SONUÇLAR VE ÖNERİLER

Bu tez çalışmasında donanımsal mozaikleştirme yöntemiyle arazi görselleştirmede kullanılan 4 detay seviyesi belirleme yönteminin, iki ana senaryo kapsamında farklı iç mozaikleştirme katsayısı, dörtlü ağaç parçası bölme katsayısı ve akış programı tampon nesnesi büyüklükleri kullanarak performansları gözlemlenmiştir.

Elde edilen veriler ışığında temel model olarak alınan uyumsal detay seviyesi ile LOD belirleme (Yöntem-1), ekranda görselleştirilmeyen arazi parçalarını filtrelememesinden dolayı testler sırasında aşırı yavaş olduğu gözlemlenmiştir. Dong Wang [7] makalesinde Yöntem-1'in performansını çizim yapılan üçgen sayısının saniyedeki çizim sayısına oranını vererek diğer yöntemlere göre performans iyileştirmesi elde edildiğini duyurmuştur. Fakat tez kapsamında yapılan testlerle bu kapsamda yapılan bir performans artışının gerçekçi olmadığı ortaya konulmuştur. Örtme sorgusu ile LOD belirleme (Yöntem-2), Yöntem-1'deki filtreleme olmamasından kaynaklanan yavaşlığı gidererek ekranda daha fazla üçgen görünmesine rağmen çizim performansı ortalama 4 kat arttırdığı gözlemlenmiştir.

Tez kapsamında farklı LOD yöntemleri ile kıyaslaması yapılmamıştır. Bunun ana sebebi farklı LOD belirleme algoritmalarında ekranda çizilen üçgen sayısı farklılaştığı için performans kıyaslamaları çok tutarlı olmamaktadır.

Yeni geliştirilen ekran noktası akış programı ve CUDA ile tekli LOD belirleme (Yöntem-3) ve ekran noktası akış programı ve CUDA ile dörtlü LOD belirleme (Yöntem-4)'ün klasik örtme sorgusu ile detay seviyesi belirleme yöntemine göre daha performanslı olduğu gözlemlenmiştir. Yöntem-4 genelde Yöntem-3'den daha yavaş Yöntem-2'den de daha hızlı olduğu gözlemlenmiştir. Yöntem-4'ün Yöntem-3'den ana farklı her bir arazi parçası için bir değil dört sorgu yapmasıdır. Performansı düşük olsa

da aslında köşesi görünen bazı parselleri daha detaylı çizdiğinden dolayı tercih edilebilir.

İç mozaikleştirme katsayısının yüksek olması ekranda çizilen bir arazi parçasının daha detaylı olması manasına geldiğinden, yöntemler arasındaki farkların kapanmaya başlandığı gözlemlendi. Çünkü çizim maliyetlerinin yüksek olması noktasal örtme sorgusundan elde edilen performans artışının daha az gözlemlenmesine yol açtığı öngörülmektedir. Dörtlü ağaç parçası bölme katsayısı eğer büyükse daha fazla arazi parçasının oluşacağı bunun da daha fazla arazi parçası noktasal görünürlük testi olduğundan dolayı Yöntem-3 ve Yöntem-4'ün daha performanslı çıkmasına yol açtığı öngörülmektedir.

Yöntem-3 ve Yöntem-4 kapsamında kullanılan akış programı tampon nesnesinin büyüklüğünün farklılaşması ile alakalı deneylerde seçilen farklı büyüklüklerin sonuca ciddi etkisinin olmadığı gözlemlendi. Çünkü sayının azlığı ekran noktası akış programında atomik işlemlerin çoğalmasına, dolayısıyla ilave bir yavaşlığa sebep olurken bununla beraber CUDA tarafındaki işlem ve hafıza transferinin az olmasından dolayı bir hızlanmaya yol açmaktadır. Bu yüzden deneyler arasında ciddi farklar gözlemlenmediği düşünülmektedir.

Engelli arazi senaryosunda Yöntem-3 ve Yöntem-4'ün deneylerin ortasında neredeyse hiç arazi parçasının çizilmemesinden dolayı performans artışı sağlayamamış hatta CUDA'nın minimum maliyetlerinden dolayı Yöntem-2'inin bile geresinde kalmışlardır. Bu kapsamda aslında parametrik bir şekilde Yöntem-3 ve Yöntem-4'ün daha az performanslı olacağın öngörülen ekranda çizilen arazi parçası sayısının miktarına göre Yöntem-2'ye geçilebilir böylelikle ekranda çok az arazi parçası varken performans kayıpları da yaşanmamış olacağı öngörülmektedir.

Testler sırasında çok nadirde olsa uzaktaki bazı zirve arazi parçalarının detay seviyesi değişiminin fark edilebildiği gözlemlendi. Bunu gidermek adına diğer detay seviyesi belirleme algoritmalarında kullanılan yükseklik tabanlı ekran hata payı hesaplamalarındaki yöntem ile noktasal görünürlük testiyle detay seviyesi belirleme yönteminin karma bir şekilde birleştirilebileceği öngörülmektedir.

KAYNAKLAR

[1] A. Valdetaro, G. Nunes, A. Raposo, B. Feijo, and R. de Toledo, "LOD terrain rendering by local parallel processing on GPU", in: Games and Digital Entertainment (SBGAMES), 2010 Brazilian Symposium on, pp. 182-188.

[2] Iain Cantlay, "DirectX 11 terrain Tessellation", Nvidia whitepaper, 2011.

[3] Egor Yusov, Maxim Shevtsov, "High-performance terrain rendering using hardware tessellation", Journal of WSCG, 19: 3 (2011) 85-82.

[4] Thatcher Ulrich, "Rendering massive terrains using chunked level of detail control", Course at SIGGRAPH 2002, 2002.

[5] Albert Cervin, "Adaptive Hardware-accelerated Terrain Tessellation", M. S. Thesis, Department of Science and Technology, Linkoping University, Sweden, 2012. 11.

[6] HyeongYeop Kang, Hanyoung Jang, Chang-Sik Cho, et al, "Multi-resolution terrain rendering with GPU Tessellation", The Visual Computer, 31: 4 (2015) 455-469.

[7] Dong Wang, Qingsheng Zhu, Yi Xia, Dan Liu, Wei Li, Ling Zhang, "Adaptive LOD Terrain Rendering with GPU Tessellation", Journal of Computational Information Systems 11: 20 (2015)

[8] Daniel Cohen-Or, Yiorgos L. Chrysanthou, Cláudio T. Silva, and Fredo Durand, "A survey of visibility for walkthrough applications", in Visualization and Computer Graphics, IEEE Transactions on 9.3 (2003): 412-431.

[9] Sekulic, D. 2004. "Efficient occlusion culling. In GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics", Pearson Higher Education.

[10] John Plate, Anselm Grundhoefer, Benjamin Schmidt, Bernd Froehlich, "Occlusion Culling for Sub-Surface Models in Geo-Scientific Applications", Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2004)

[11] Nick Gebbie and Mike Bailey. "Fast Realistic Rendering of Global Worlds Using Programmable Graphics Hardware." Journal of Game Development 1:4 (2006), 5-28.

[12] Yacine Amara, Xavier Marsault, “A GPU Tile-Load-Map architecture for terrain rendering: theory and applications”, In The Visual Computer (VC) 25(8), pp.805-824 (2009)

[13] Thomas Engelhardt and Carsten Dachsbacher. 2009. “Granular visibility queries on the GPU”. InProceedings of the 2009 symposium on Interactive 3D graphics and games (I3D '09). ACM, New York, NY, USA, 161-167.

[14] Szymon Jabłonski, Tomasz Martyn, “Real-Time Rendering of Continuous Levels of Detail for Sparse Voxel Octrees”, WSCG 2016 - 24th WSCG Conference on Computer Graphics, Visualization and Computer Vision 2016

[15] Egor Yusov, Vadim Turlapov, “GPU-optimized efficient quad-tree based progressive multiresolution model for interactive large scale terrain rendering”, in: Conference Proceedings of the 17th International Conference on Computer Graphics and Vision (GraphiCon 07), 2007, pp. 23-27.

[16] https://en.wikipedia.org/wiki/DirectX/ (01.01.2016)

[17] https://www.opengl.org/wiki/History_of_OpenGL/ (01.01.2016)

[18] Corey Barnard , “Applying Tessellation to Clipmap Terrain Rendering”, COSC460 Research Project Report , 2014

[19] https://www.opengl.org/wiki/Shader_Storage_Buffer_Object/ (01.01.2016)

[20] D. Wolff, “OpenGL 4 Shading Language Cookbook”, Birmingham B3 2PB, UK: Packt Publishing Ltd., 2013.

[21] J. Stam, “What Every CUDA Programmer Should Know About OpenGL”, in GPU Technology Conference, San Jose, CA, October 1, 2009

[22] https://tr.wikipedia.org/wiki/OpenGL/ (01.01.2016)

[23] http://www.nvidia.com.tr/object/cuda-parallel-computing-tr.html (01.01.2016)

[24] http://www.nvidia.com.tr/object/gpu-computing-tr.html (01.01.2016)

[25] M. Harris, “Optimizing Parallel Reduction in CUDA”, NVIDIA Developer Technology (2007).

[26] https://en.wikipedia.org/wiki/Level_of_detail/ (01.01.2016)

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı: Mehmet Ömer Özek

Uyruğu: T.C.

Doğum Yeri ve Tarihi: Fethiye - 01.10.1981

Medeni Hali: Evli

Adres: Akpınar Mh. 848. Sk. No: 3 Daire: 15 Çankaya/ANKARA

E-Posta Adresi: mozek@havelsan.com.tr

İletişim (Telefon) : 5331397068

EĞİTİM

Lise : Fatih Sultan Lisesi (Nazilli) - 1999

Lisans : Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği (İzmir) - 2005

Yüksek Lisans : Türk Hava Kurumu Üniversitesi Elektrik ve Bilgisayar Mühendisliği (Ankara) – Devam Ediyor

MESLEKİ DENEYİM

HAVELSAN, Ankara

Kıdemli Yazılım Mühendisi, Aralık 2006 – Devam Ediyor

YABANCI DİL

İngilizce