İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

NOKTA BULUTLARININ WEB ÜZERİNDE ÜÇ BOYUTLU GÖRSELLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Arife MUTLU

Bilişim Enstitüsü

Coğrafi Bilgi Teknolojileri

Tez Danışmanı: Dr. Öğretim Üyesi Caner Güney

2018

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

NOKTA BULUTLARININ WEB ÜZERİNDE ÜÇ BOYUTLU GÖRSELLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Arife MUTLU

Bilişim Enstitüsü

Coğrafi Bilgi Teknolojileri

Tez Danışmanı: Dr. Öğretim Üyesi Caner Güney

2018

İTÜ, Fen Bilimleri Enstitüsü'nün 706141003 numaralı Yüksek Lisans öğrencisi Arife Mutlu, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "NOKTA BULUTLARININ WEB ORTAMINDA ÜÇ BOYUTLU GÖRSELLEŞTİRİLMESİ" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Dr. Öğr. Üyesi Caner GÜNEY** .....
İstanbul Teknik Üniversitesi

Jüri Üyeleri : **Prof. Dr. Necla ULUĞTEKİN** .....
İstanbul Teknik Üniversitesi

Dr. Öğr. Üyesi Melis UZAR .....
Yıldız Teknik Üniversitesi

Teslim Tarihi : 4 Mayıs 2018
Savunma Tarihi : 6 Haziran 2018

ÖNSÖZ

Tez sürecim boyunca sabır gösteren ve destek olan danışman hocam Yrd. Doç. Dr. Caner Güney'e ve benden yardımlarını esirgemeyen sayın Şakir Çağlar Toklu'ya teşekkürü bir borç bilirim.

Mayıs 2018 Arife Mutlu

İÇİNDEKİLER

Sayfa

KISALTMALAR

3B : 3 Boyutlu
3D : 3 Dimensional
API : Application Programming Interface
BIM : Building Information Modeling
BLOB : Binary Large Object
CAD : Computer-Aided Design
CAM : Computer-Aided Manufacturing
CityGML : City Geography Markup Language
CPU : Central Processing Unit
CSS : Cascading Style Sheets
DBMS : Database Management System
DTM : Digital Terrain Model
GDAL : Geospatial Data Abstraction Library
GIS : Geospatial Information Systems
GML : Geography Markup Language
GPU : Graphics Processing Unit
HTML : Hyper Text Markup Language
IMU : Inertial Measurement Unit
JSON : Javascript Object Notation
LAS : Local Authority Services
LAZ : LASzip
LIDAR : Light Detection and Ranging
NoSQL : Not Only SQL Database
OGC : Open Geospatial Consortium
OpenGL : Open Graphic Library
PCL : Point Cloud Library
PDAL : Point Data Abstraction Library
RDBMS : Relational Database Management System
RGB : Red, Green, Blue
SDK : Software Development Kit
SQL : Structured Query Language
TIN : Triangular Irregular Network
VR : Virtual Reality
WEBGL : Web Graphics Library

XML : Extensible Markup Language

ŞEKİL LİSTESİ

Sayfa

NOKTA BULUTLARININ WEB ÜZERİNDE ÜÇ BOYUTLU GÖRSELLEŞTİRİLMESİ

ÖZET

Hem açık alanlarda hem kapalı alanlarda nesnelerin yüzeylerinin ve/veya içinde bulundukları ortamların hava, kara ve su üzerindeki platformlardan sabit ya da hareketli biçimde elde edilen lazer veya görüntü kaynaklı nokta bulutu veri üretimi üstel olarak artmaktadır. Faz temelli, uçuş zamanlı, yapılandırılmış ışık (*structured light*), görüntü çifti ile 3 boyutlu (3B) görüntü (*stereo-imagery*) vb. ölçme teknikleri ile hızlı biçimde elde edilebilen, tarama alanı ve tarama yoğunluğuna bağlı olarak milyonlarca veya milyarlarca noktadan oluşan büyük miktardaki nokta verileri gerek kamunun gerekse özel sektörün ve girişimcilerin farklı uygulamalarında yoğun olarak yer almaktadır. Nokta verisi ne kadar yoğun olursa o düzeyde detay ortaya çıkacak ve nesnelerin/ortamın 3B geometrik modellenmesi için daha zengin veri kullanılabilecektir. Geniş bir uygulama yelpazesinde yer alan nokta bulutları tekil bir nesneden 3B kent modellerine ve robotik uygulamalara kadar farklı birçok alanda kullanılmaktadır. Lazer tarayıcı veya kamera eklenmiş bir İHA filosu yüksek çözünürlüklü 3B nokta bulutu verilerini hızlı bir biçimde üretebilmektedir ya da tek kameralı veya çift kameralı bir akıllı telefonla 3B nokta bulutu veri setleri kolaylıkla üretilmektedir. Gezgin robotlar yüksek kalite 3B nokta bulutları ile içinde bulundukları ortamı tanımlayabilmektedir. Hızla gelişen nokta bulutu üretim teknolojileri ve bu teknolojilerin artan örnekleme hızları (*sampling rates*) ile farklı uygulamalarda milyarlarca nokta üretilebilmektedir.

Hızlı bir biçimde üretilebilen yüksek çözünürlüklü ve doğruluklu büyük miktardaki nokta bulutu verisinin yönetimi, depolanması, değerlendirilmesi, görselleştirilmesi, semantik özelliklerinin kullanılması gibi temel konular üzerinde yoğunlaşılması gereken alanlar arasında yer almaktadır.

Üretilen nokta bulutlarının büyük boyutta veriler olması nedeniyle ortalama bir bilgisayarda bile çalışılamayan nokta bulutlarının web üzerinde görselleştirilmesi problem olan bir konudur. Bu nedenle nokta bulutlarının değerlendirilmesi ve görselleştirilmesi genellikle iş istasyonları ya da yüksek kapasiteli sunucular üzerinde gerçekleştirilmektedir. Bunun dışında nokta yoğunluğunu azaltma (*subsampling*) uygulanan bir teknik olmasına rağmen gerçek durumu değiştirdiği için olumsuz yönleri de bulunmaktadır. Ancak bulut hesaplamadaki, farklı veri depolama yöntemlerindeki gelişmeler ve WebGL teknolojisinin büyük rağbet görmesi vb. gelişmeler nokta bulutlarının kullanılma alışkanlıklarını da değiştirmektedir.

SUMMARY

In both open and closed areas, Laser or image-based point cloud data production, which is obtained in fixed or moving form over the surfaces of objects and / or the media in which they are located, is exponentially increasing from platforms on air, land and water. Phase-based, flight-time, structured light, image pair and 3-D (stereo-imagery) large quantities of millions or billions of points, which can be obtained quickly by measuring techniques, depending on the scanning area and scanning density, are heavily involved in the different applications of the public sector, private sector and entrepreneurs. The more intense the point data, the more detail will be revealed at that level and the richer data will be available for 3D geometric modeling of the objects / environment. In a wide range of applications, point clouds are used in many areas ranging from a single object to 3D urban models and robotic applications. A HRA fleet with a Laser scanner or camera can quickly produce high resolution 3D point cloud data, or 3D point cloud data sets are easily generated with a single camera or dual camera smartphone. With high quality 3D point clouds, mobile robots can identify the environment they are in. Billions of points can be produced in different applications with the rapidly developing point cloud production technologies and the increasing sampling rates of these technologies.

It is one of the areas of study that needs to focus on basic issues such as management, storage, evaluation, visualization and use of semantic features of high-resolution and accurate large point cloud data that can be produced quickly, and effective and automated solutions based on robotic field should be developed.

Visualization of point clouds, which can not be worked on even an average computer, is a problem because the generated point cloud is given in large size. For this reason, the evaluation and visualization of point clouds is usually performed on workstations or high-capacity servers. It is also a technique applied to subsampling of spot density, but it has negative aspects as it changes the actual situation. However, cloud computing, the development of different data storage methods and the popularity of WebGL technology and so on. developments also change the habits of consuming point clouds.

1. NOKTA BULUTU VERİLERİ

1.1 Nokta Bulutu Tanımı

Gerçek nesnelerinin 3 boyutlu (3B) modellerinin oluşturulmasında, bu nesnelere ait özellik çıkarımlarında, nesne tanıma ve nesne bulma gibi birçok uygulamada 3B nokta bulutları (point clouds) etkin olarak kullanılmaktadır. Nokta bulutları yeryüzünde bulunan fiziksel nesnelerin dijital ortamda üç boyutlu bir koordinat sisteminde yeniden yapılandırılmaları için kullanılan 3B gösterimlerdir. Nokta bulutları genel olarak LAS (Local Authority Services), XYZ gibi açık formatların yanında üretici firmaların kendi binary formatlarında dosya yapısında tutulmaktadır.

Nokta bulutları lazer tarayıcıların gönderdikleri lazer sinyalinin nesnelerin yüzeyinde yansıması ve yansıyan sinyalin kaydedilmesi ile üretilen çok sayıdaki noktalardır. Lazer ışını nesnelerin dış yüzeyinden yansımasından dolayı nesnenin yüzeyine ilişkin geometrik bilgiler üretilirken, nesnelerin görünmeyen iç kısımlarına ilişkin bilgi üretilememektedir.

Nokta bulutu gösteriminde bulunan noktalar birbirinden bağımsız ve ilişkisizdir. Bu nedenle farklı uygulamalarda nokta bulutları kullanılmak istendiğinde nokta bulutu verisinin düzenlenmesi gerekmektedir. Uygulamada bu düzenleme işi genel olarak örgü (mesh) yapısı kullanılarak gerçekleştirilmektedir. Böylece nesneler, nokta bulutunda bulunan noktalar birleştirilerek üçgenler, çizgiler ve yüzeyler gibi geometrik yapılar oluşturulmakta ve en sonunda bu geometrik yapılar da ilgili nesneyi göstermektedir.

Nokta bulutu vektör tabanlı bir yapı olup bu vektör yapısında her bir noktanın XYZ koordinatları ve eğer mevcutsa yoğunluk (intensity), renk gibi diğer özellikler tutulabilmektedir. Nokta bulutunun çözünürlüğü bir diğer ifadeyle içerinde bulunan nokta sayısı ne kadar yüksekse tarama yapılan ortama ilişkin o kadar yüksek çözünürlükte detay ortaya çıkarılabilir. Nokta bulutuna ilişkin detay oranı arttıkça ortamın ve içerisinde bulunan nesnelerin 3B modelleri o derece gerçeğe yakın biçimde oluşturulabilecektir. Ancak veri yoğunluğunun artması nokta bulutu verilerinin yönetimini ve işlenmesi

güçleştirmektedir. Bu nedenle bazı uygulamalarda nokta bulutu verisi raster veriye dönüştürülerek kullanılmaktadır. Verinin boyutunu düşürmek için raster yapısına dönüşüm sırasında nokta sıklığı azaltılmaktadır.

1.2 Nokta Bulutu Elde Etme Yöntemleri

Farklı kaynaklardan nokta bulutu elde etme giderek yaygınlaşmaktadır. Sözü edilen kaynaklara aşağıdaki teknikler örnek olarak gösterilebilir:

- Derinlik Kameralarcı (RGB-D)
- Stereo Kameralar
- Fotogrametri
- Yersel lazer Tarama (TLS)
- Havadan lazer Tarama (LIDAR)

Geleneksel ölçme teknikleri ile karşılaştırıldığında lazer tarama tekniği 3B nokta bilgilerini en hızlı üreten tekniktir. Lazer tarama tekniği kullanılarak amaca uygun nokta sıklığıyla ölçülecek alanın 3B nokta bilgileri hızlı bir biçimde üretilebilmektedir. Tarayıcıların tarama teknikleri üretici firmanın belirlediği yöntemle gerçekleştirilmektedir. Genel olarak tarayıcılar satır tarama veya sütun tarama yaparak tarama işini gerçekleştirmektedir.

Lazer ölçmelerinde temel büyüklük cihazla hedef arasındaki mesafedir. Bu uzaklığın belirlenmesi için farklı teknikler kullanılabilir. Bu tekniklere örnek olarak faz farkının belirlenmesi, elektromagnetik dalganın gidiş/dönüş zamanının belirlenmesi, üçgenleme vb. gösterilebilir.

1.2.1 Lazer Tarayıcıların Ölçme Prensibi

Lazer tarayıcılar, ölçülecek ortamı veya nesneyi yatay ve düşey yönde belirli bir açı altında nokta dizileri şeklinde tarayarak nokta bulutu halinde görüntülenmesini sağlar. Her lazer noktası için tarayıcı alet merkezli kutupsal koordinatlar ölçülür. Bunlar, ölçülen noktaya olan eğik uzaklık, ölçme doğrultusunun X ekseni ile yatay düzlemde yaptığı açı ve ölçme doğrultusunun yatay düzlemde yaptığı eğim açısıdır. Aynı zamanda ölçülen yüzeyin yapısına ve ölçme uzaklığına bağlı olarak dönen sinyalin yoğunluğu da ölçülerek kaydedilir. (Altuntaş & Yıldız, 2008)

Lazer ölçme sistemi gönderici (transmitter), alıcı (receiver), kontrol ünitesi ve tarayıcı aynasından (scanning mirror) oluşur. Uçuş şeridi üzerinde hareket eden hava aracında bulunan lazer tarama ünitesinin LIDAR (Light Detection and Ranging) sensörü, tarayıcı aynası ile lazer ışınını gönderir, lazer ışının tarama açısını ölçer ve her bir ışının eğim açısını tarama açısı ve IMU (Inertial Measurement Unit) verileriyle beraber kaydeder. Ayrıca lazer ışınının gidiş dönüş arasındaki süre de kaydedilir. Sistemin bir diğer bileşeni olan IMU, X,Y,Z koordinat değerleri "roll, pitch, heading" değerlerine karşılık gelir ve koordinat düzlemindeki açısal sapmaları ifade eder. (Altuntaş & Yıldız, 2008)

Görüntü tabanlı nokta bulutu üretimi maliyet açısından lazer tabanlı nokta üretminden çok daha avantajlıdır. Öyle ki akıllı telefonlarda bulunan tek kameralardan ya da yeni çıkan modellerde bulunan çift kameradan elde edilen görüntülerden nokta bulutu üretilebilir. Hava fotogrametrisi ve/veya İHA (İnsansız Hava Aracı) fotogrametrisinden elde edilen görüntülerden oluşturulan nokta bulutları da farklı uygulamalarda yoğun olarak kullanılmaktadır.

1.2.2 Lazer Tarayıcıların Doğruluğu

Lazer tabanlı ölçme tekniği de diğer joedezik ölçme tekniklerinde olduğu gibi bazı hata kaynaklarına sahiptir. Rastlantısal hata kaynaklarına lazer dalgasının yansıma hatası ve lazer dalgasının kalınlığı örnek olarak gösterilebilir. Taranan noktanın doğruluğunu tarayıcının açısal doğruluğu, mesafe doğruluğu, çözünürlüğü, kenar etkisi yani ışın kalınlığı, yansıyan sinyal gücü ile sıcaklık, atmosferik koşullar etkilemektedir. (Altuntaş & Yıldız, 2008)

Farklı istasyon noktalarında gerçekleştirilen lazer taramalarıyla her biri farklı bir lokal koordinat sisteminde bulunan nokta bulutları üretilmektedir. Farklı lokal koordinat

sisteminde bulunan bu nokta bulutları tek bir lokal koordinat sisteminde bütünleştirilmektedir (registration/alignment). Sözü edilen lokal koordinat sisteminin jeodezik koordinat sistemine dönüştürülmesi için her iki sistemde koordinat değerleri bilinen ortak noktalar kullanılmaktadır. Bu ortak noktaların her bir sistemdeki koordinat değerlerinde bulunan hatalar da nokta bulutunun dış doğruluğuna etki eden hata kaynaklarıdır.

Bindirmeli en az iki resim kullanılarak, resimlerdeki ortak piksellerin eşleşmesi ve önceden bilinen ya da analiz sırasında hesaplanan bir takım parametreler sayesinde hesaplanan derinlik ile 3B nokta bulutu üretilebilir. Fotogrametri ile 3B nokta bulutu elde etmenin doğruluğu ortamın ışığına, kameranın iyi olmasına, pozlamanın iyi olmasına, mekan dokusuna ve bilgisayar kapasitesine bağlı olarak değişmektedir.

1.3 Nokta Bulutu Verilerinin Kullanım Alanları

Nokta bulutları ile gerçek ortamda bulunan fiziksel nesnelerin tek bir koordinat sisteminde dijital ortamda 3B gösterilebilmesi farklı birçok uygulama alanı için değerli bir ürün olarak görülmesine neden olmuştur. Uygulama alanlarına kent modelleri (CityGML, City Geography Markup Language), bina modelleri (BIM), medikal görüntüleme, tarihi eserlerin röleveleri ve restorasyonları, yer altı ve yer üstü madencilik uygulamaları, 3B oyun endüstrisindeki kullanım, sanal gerçeklik (VR, Virtual Reality) örnek olarak gösterilebilir. Diğer bir ifadeyle 3B modellemenin ve 3B görselleştirmenin kullanıldığı birçok alanda nokta bulutları etkin olarak kullanılmaktadır.

Nokta bulutu verilerinin avantajları hızlı render edilmesi, mekanı ya da alanı tam temsil etmesi ve alanın bütününün işlenmesini mümkün kılması ve hızlı dönüşümler olarak sayılabilir.

Nokta bulutu verilerinin dezavantajı ise veri boyutunun yüksek olması ve bunun yüksek bellek tüketimine ve donanım ihtiyacına neden olup yüksek hesaplama maliyeti ortaya çıkarmasıdır.

1.4 Tezin Amacı

Lazer tarama teknoljisi ve 3B nokta bulutu üretimi çok geniş ve derin bir konudur. Lazer tarama tekniği ve teknolojisi her ne kadar derin bir konu olsa da uygulamada

farklı disiplinlerce farklı amaçlar doğrultusunda yaygın olarak kullanılmaktadır. Bu nedenle tez çalışması kapsamında hem nokta bulutu veri üreticileri hem de farklı alanlarda bulunan farklı disiplinden kullanıcılar için nokta bulutlarının internet ortamında gösterimi, etkileşimi ve analizi için bir bütüncül bir çatkı (framework) önerilmiştir. Sözü edilen bütüncül çatkıda odak noktası nokta bulutu verisinin üretimi değil üretildikten sonra farklı aşamalarda etkin ve esnek kullanımıdır.

Bu nedenle tez çalışmasının ikinci bölümünde nokta bulutu verilerinin yönetimi konusu üzerinden durulmuştur. Eğer nokta bulutu verileri amaca uygun nitelikte yönetilemezse nokta bulutu verilerinin web üzerinden sunumu da başarılı olamayacaktır. Üçüncü bölümde nokta bulutunun web üzerinden 3B gösterimi için kullanılan açık kaynaklı çözümlerden bahsedilecektir. Dördüncü bölümde nokta bulutlarının nokta gösterimi dışında 3B modelleme ile gösterimi konusundaki çalışmalara yer verilecektir. Beşinci bölüm de nokta bulutu verilerinin sanal gerçeklik ortamlarında nasıl kullanılabileceği üzerinde durulacaktır. Altıncı bölümde görselleştirme için kullanılan uygulamardaki sistem mimarisi ve alt yapı üzerinde durulacak. Yedinci bölümde görselleştirilen nokta bulutu verilerinin robotik uygulamalarda nasıl etkin kullanılabileceği gezgin robot güzergah/yörünge planlama konusu üzerinden açıklanacaktır. Yedinci bölümde çalşmanın sonuçları paylaşılacaktır.

2. NOKTA BULUTU VERİLERİNİN YÖNETİMİ

Nokta bulutlarının web üzerinde görselleştirilmesi ve üzerinde analiz yapılabilmesi için öncelikle nokta bulutlarının yönetilebilmesi gerekmektedir.

2.1 Nokta Bulutu Veri Yönetimi

2.1.1 Bölütleme (Segmentasyon)

3B nokta bulutu segmentasyonu, nokta bulutlarını çok sayıda homojen bölgeye ayırma işlemidir, aynı bölgedeki noktalar aynı özelliklere sahip olacak şekilde gruplanmaktadır. Noktasal bulut kümesi göz önüne alındığında, segmentasyon işleminin amacı, benzer karakteristiklere sahip noktaları küme ederek anlamlı homojen bölgelere dönüştürmektir. Segmentasyon işlemi alandaki nesneleri bulma ve tanıma, sınıflandırma ve özellik çıkarımı gibi çeşitli yönlerden analiz etme konusunda yardımcı olmaktadır. (Richter, Behrens , & Döllner, 2013)

Segmentasyon işlemi büyük veri işleme, eşit olmayan yoğunluk ve nokta bulutu verilerinin net olmayan yapısından dolayı oldukça zorlayıcıdır.

İleri teknoloji lazer tarama cihazları saniyede milyonlarca veri noktası üretir ve bu nedenle segmentasyon için kullanılacak yöntemlerin bu veri hacmi ile başa çıkabilmesi gerekmektedir. Diğer yandan nesnelerin sadece yüzey bilgisi elde edildiğinden nesne ile ilgili veri net değildir.

Bu problemler akıllı araçlar, otonom haritalama ve navigasyon gibi bir çok robotik uygulamalar ile çözülebilir hale gelmektedir. Bu işlem için çeşitli algoritmalar kullanılmaktadır. Basit ve sıkça görülen yapılar içeren (cadde ve bina cepheleri gibi) nokta bulutu verilerinin segmentasyonu için kural tabanlı (rule-base) segmentasyon yöntemi kullanılırken, gürültü, düzensiz yoğunluk ve tıkanıklıklar nedeniyle daha karmaşık hale gelen nokta bulutu verileri için öğrenme tabanlı (learning-based) segmentasyon yöntemi kullanılmaktadır. (Babahajiani, fan, Kämäräinen2 , & Gabbouj2, 2017)

2.1.2 Sınıflandırma (Classification)

Nokta bulutu verilerini sınıflandırmak için birçok yöntem kullanılmaktadır. Bunlar yükseklik farkı, eğim tabanlı, morfolojik ve çoklu ölçekleme olarak ayrılabilir. En çok

kullanılan yöntemlerden eğim tabanlı kategorisinde “Axelsson algoritması” ve “Vosselman algoritması” bulunmaktadır.

Axelsson algoritmasına göre sınıflandırma, kullanıcının tanımladığı eğim eşiği ve minumum mesafe gibi parametrelere bağlıdır. “Adaptive TIN” filtreleme algoritması düğüm (seed) noktaları olarak tanımlanan noktalarla TIN (Triangular Irregular Network) oluşturur. Bu filtreleme algoritması belirlenen eşik parametrelerine bağlı olarak iteratif olarak her seferinde TIN yüzeyine nokta ekler. Filtreleme tüm noktalar sınıflandırılana kadar devam eder. (Civelekoğlu, 2015)

Vosselman ise yükseklik farkından yola çıkarak filtreleme metodu üretmiştir, iki nokta arasındaki kabul edilebilir yükseklik farkı noktaların arasındaki mesafe fonksiyonu olarak tanımlanır. Yükseklik farkından yararlanılarak geliştirilen Kraus ve Pfeifer algoritması da nokta bulutu için geliştirilen bir başka yaklaşımdır. Bu algoritmada LIDAR nokta verilerini sınıflandırmak için noktaları, yüzeyde olan ve olmayan noktalar olarak ayırarak enterpolasyon ve filtreleme yöntemi üretilmiştir. Kraus ve Pfeirer LIDAR nokta bulutu ile, yaklaşık yeryüzünü enterpole etmektedir. Geriye kalan artık noktalar (yüzeyden diğer LIDAR noktalarına olan mesafe) ağırlık fonsiyonuna eklenip hesaplanmaktadır. Bu algoritma ile ağırlık değerleri atama ve yeni yüzey hesaplama işlemleri iteratif olarak maksimum iterasyon sayısına ulaşılana kadar yapılır. (Civelekoğlu, 2015)

2.2 Nokta Bulutu Veri Yapıları

Nokta bulutunda bulunan düzensiz noktaların uygun bir veri yapısı kullanılarak yeniden düzenlenmesi gerekmektedir. Çoklu çözünürlüklü veri yapısı bu iş için kullanılmaktadır. Nokta bulutu verilerinin kaydedilmesi ve görselleştirilmesi (visualization) büyük miktarda 3B verinin işlenmesini ve bu verilerin yönetilmesini gerektirir. Birden çok poligondan oluşan belli bir uygulama için bir veri yapısı genel nesneleri tam temsil etmelidir, nesnelerin kombinasyonları birbirleri ile ilişki olmalıdır, hızlı render edilebilmeli, hafıza kapasitesi yüksek olmalı, hızlı mekansal arama yapabilmelidir. Bu işlemleri yapanbilmesi için aşağıdaki veri yapıları kullanılmaktadır.

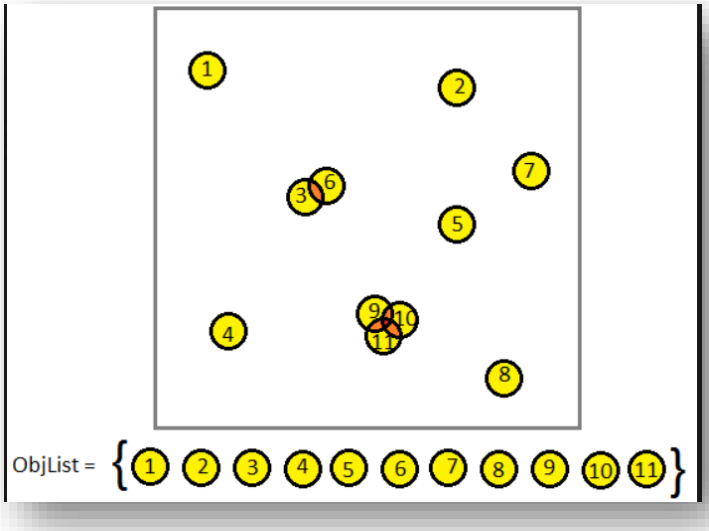
2.2.1 Octree

Octree veri yapısı katı volümetrik nesneleri temsil etmek için kullanılır. Bir boyutlu binarytree ve iki boyutlu quadtree'nin 3B için genelleştirilmiş halidir. Octree 3B nokta bulutu verisinin depolanmasına uygun formatta veri yapısıdır. Octree veri yapısında her düğüm 8 altuzaya bölünmüştür. Her altuzay boş, tam veya daha da bölünmüştür. Bir nesne yeterince doğru temsil edildiğinde alt bölüm durur. Bir octree, her hücredeki kalan nesne sayısı önceden belirlenmiş bir eşiğin altına düşene kadar alanı sekiz hücrelere yinelemeli olarak bölerek ve ya maksimum ağaç derinliğine ulaştırarak oluşturulur. (Mahtani, Sánchez , Fernández , & Martinez, 2016)

Bir sekizli gösterimin hiyerarşik yapısını modellemek için alan, istenilen nesneyi barındıracak kadar büyük bir küp olarak düşünülebilir. Bu kübü barındıran alanı evren ya da çalışma yapılan alan olarak adlandırılabilir. Sekizlik evren modellemek istenilen nesnenin boyutu kadar büyük bir alanı sekiz küp şeklinde bölünmesiyle elde edilir ve her oktan tek bir özniteliğe sahip olana kadar veya bir çözünürlüğe erişene kadar tekrar tekrar altuzaya (suboctantlara) ayrılır. Burada evrenin, yani çalışma yapılan alanın 2n*2n*2n voksellerden oluştuğu varsayılmaktadır. Bir octree'nin kökü, nesnenin kendisine karşılık gelir ve bir octree içindeki her düğüm ya sekizinci dereceden bir yaprak düğüm veya gri bir düğümdür. (Mahtani, Sánchez , Fernández , & Martinez, 2016)

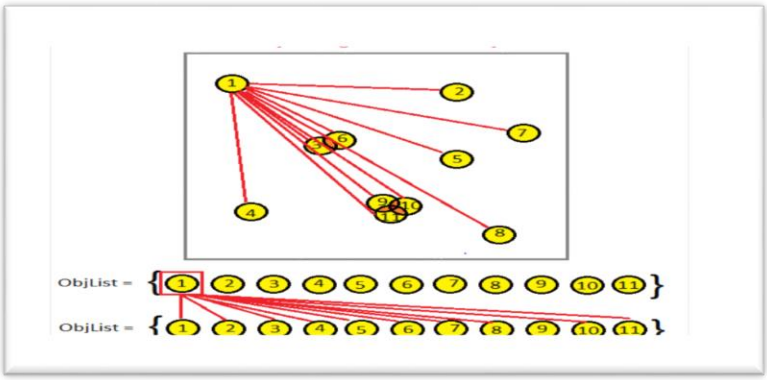
Bir yaprak düğüm düzgün özniteliğe sahip düğümdür ve karşılık gelen oktantının temsil edilen nesnenin içinde veya tamamen dışında olup olmadığına bağlı olarak siyah veya beyaz olarak yazılabilir ve gri bir düğüm düzgün olmayan bir sekizgendir.

Nokta bulutu verileri birbirinden bağımsız verilerdir ve bu verileri kullanırken ve görselleştirirken bu verileri birbirleri ile ilişkilendirmek gerekmektedir. Görselleştirme sırasında kullanılan noktalar birbirleri ile kesişebilir ya da birbirleri ile bağlantılı olabilir. Şekil 2.1'de görüldüğü gibi bir alan temsil edilirken birden fazla nokta bu alanı temsil edebilir ya da başka alanlar ile bağlantı oluşturabilir.



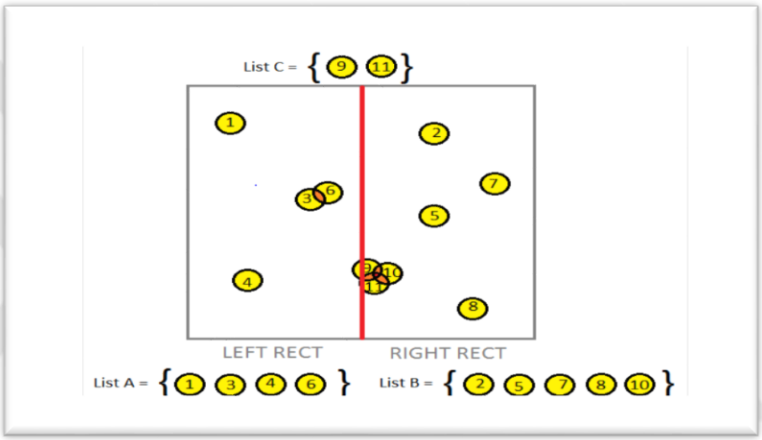
Şekil 2.1 Nokta bulutlarının birbirleri ile ilişkisi (Nevala, 2014)

Şekil 2.2' de görüldüğü gibi noktalar keşişebilir veya üst üste binebilir. Böyle alanlarda alanı daha iyi temsil edebilmek için bağlantılı ve kesişen objelerin bulunması gerekmektedir. Bu ilişkiyi sağlamak için bu nokta bulutu verilerinin konumları birbirleri ile karşılaştırılır. Eğer konumları eşit ya da kesişiyor ise bağlantılı nesnelerdir denilebilir. Bu da yazılımsal olarak basit bir döngü ve if koşulu ile yapılır. Fakat bu döngü bütün noktaları dolaşarak hepsinde teker teker kontrol yaparken Şekil.3' de görüldüğü gibi çok fazla işlem gerektirir ve bu işlemler zaman alır. Dolayısıyla görüntü oluşana kadar çok fazla zaman geçer. Yani görüntünün render edilmesi uzun sürer (Nevala, 2014).



Şekil 2.2 Nokta bulutlarının birbirleri ile karşılaştırılması (Nevala, 2014)

Bu sorunu çözebilmek için çeşitli algoritmalar kullanılır. Bu algoritmaların temelinde çarpışmalar için kontrol edilecek nesne sayısını azaltmak vardır. Mekansal bölme algoritmaları, çarpışmaları algılamak için çalışma ve nesne kontrolü zamanı probleminin çözümüne yönelik kolaylık sağlamaktadır. Belirli bir alanda bulunan tüm nesneler belli mekânsal konuma sahiptir. Şekil 2.3'de belirli bir alanda yer alan nesneler görünmektedir. Bu alanın sol tarafında kalan nesneler için List A, sağ tarafında kalan nesneler için List B ve her iki taraf ile ortada kalan nesneler için List C oluşturulur (Nevala, 2014).



Şekil 2.3 Belirli bir alanda yer alan nesneler (Nevala, 2014)

Liste A'daki tüm nesnelerin Liste B'deki herhangi bir nesneyle hiçbir zaman kesişmeyeceği bilinmektedir, bu nedenle çarpışma kontrollerinin sayısının neredeyse yarısını ortadan kaldırılabilir. Liste C'de hala A ya da B listesindeki nesnelere dokunabilecek nesneler vardır, bu nedenle Liste C'deki tüm nesneleri Listeler A, B ve C'deki tüm nesnelere karşı kontrol edilmelidir. Alan daha küçük ve daha küçük parçalara bölmeye devam edilirse, gerekli çarpışma kontrol sayısını her defasında yarıya kadar azaltılabilir. Mekansal bölümlemenin arkasındaki genel fikir budur. Bir dünyayı ağaç benzeri bir veri yapısına (Binary Tree, QuadTree, Kd Tree, OctTrees, vb.) ayırmanın birçok yolu vardır. Bir alanı karelere ayırmak, her kareyi gerçekleştirmesi gereken çarpışma kontrollerinin sayısını azaltmak anlamına gelmektedir. Eğer bir bölge nesne içermiyorsa o bölgeyi ağaca yani bölümlemeye dahil

etmeye gerek yoktur. Bu şekilde sonsuz bölümlemenin önüne geçilmiş olunmaktadır (Nevala, 2014).

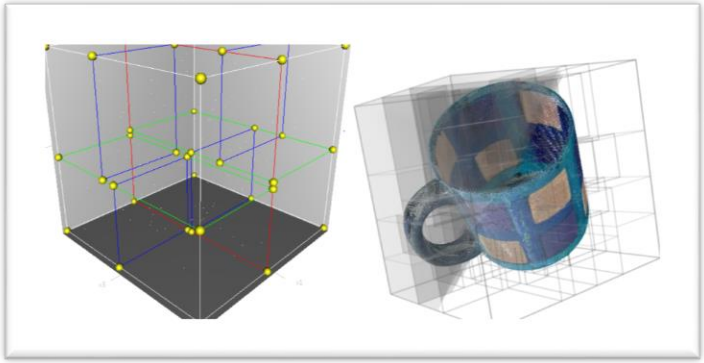
2.2.2 Kd Tree

Kd Tree k boyutlarına sahip bir alanda bazı noktaların düzenlenmesi için kullanılan veri yapısıdır. Kd Tree menzil ve en yakın komşu aramaları için çok kullanışlıdır.

Kd Tree de x ve y koordinat değerlerine göre alandaki noktaları parçalara bölerek bir ağaç oluşturulur. Kd ağaçları oluştururken binary search tree (ikili arama ağacı) kullanılır (Ooi, 2017).

Normal bir binary search algoritması büyüklük küçüklük durumuna göre çalışırken burada koordinatların büyüklük küçüklük değerine bakarak ağaç oluşturulur. Fakat bu yapılırken bir seviyede x koordinatı bir seviyede y koordinatı dikkate alınır. Kök (root) seviyede x koordinatına bakılarak ağaç oluşturulur. Kd Tree alanın bir bölgesini yinelemeli olarak bölümlere ayırır ve ağacın her seviyesinde bir ikili boşluk bölümü oluşturur. 3B Kd Tree ilk bölme kök hücresini iki alt hücreye ayırır, bunların her biri daha sonra iki alt hücreye bölünür. Son olarak, bu dörtlerin her biri iki alt hücreye bölünür (Ooi, 2017).

Genel olarak, bir Kd ağacı, k boyutlarında çok boyutlu bir arama ağacıdır. Kd Trees, koordinat sistemi eksenlerine dik olan bölme düzlemlerini kullanır. Bu nedenle, bir nokta kümesi eksenle hizalanmış kesişmeyen kübik bölgelere bölünür. N noktaları için üç boyutlu kd ağacı n yapraklı bir ikili ağaç olduğundan, O (n) depolamasını kullanır ve O (n log n) yapım süresine sahiptir (Goldberg, 2017). Şekil 2.4’ te Kd Tree veri yapısı gösterilmektedir.



Şekil 2.4 Kd Tree Veri Yapısı (Goldberg, 2017)

2.2.3 QuadTree

QuadTrees iki boyutlu uzayda noktaların verilerini verimli bir şekilde saklamak için kullanılan ağaçlardır. Bu ağaçta her düğümün en fazla dört alt kümesi (children) vardır. İki boyutlu bir alanda dörtlü oluşturmak için; alan dört kutuya bölünür. Bir kutuda bir veya daha fazla nokta varsa, bir alt nesne oluşur ve kutunun iki boyutlu alanı saklanır. Bir kutu herhangi bir nokta içermiyorsa bunun için alt küme oluşturmaya gerek yoktur. Her bir alt küme için işlem tekrarlanır.

Quadtress görüntü sıklaştırılmasında kullanılır. Her düğüm, çocukların (children) her birinin ortalama rengini içerir. Ağaç ne kadar derin olursa görüntünün detayı o kadar fazla olur. Verilen koordinata en yakın koordinatı bulmak için quadtrees kullanılır (Yin, 2009).

2.2.4 Google's MegaTree

MegaTree, milyarlarca nokta tutacak mekansal veri yapısı içeren octree veri yapısıdır.

Megatree, büyük çekirdek dışı octree' lerde nokta bulutlarını depolamak, sıkıştırılmış nokta akışlarını istemcilere akıtmak ve bir web tarayıcısındaki nokta bulutlarını görüntülemek için paketler içerir. Octree milyarlarca nokta içerir. Bir hadoop kümesinde ve yerel diskte depolanabilir. Düşük istemciler için yüksek oranda sıkılaştırılmış noktalar içerir ve web tarayıcısında milyarlarca noktanın görüntülenmesini sağlar.

2.3 Grafik Kütüphaneler

2.3.1 PCL

PCL (Point Cloud Library) 2B ya da 3B görüntü ve nokta işleme için kullanılan açık kaynak kodlu görüntü işleme kütüphanesidir. C++ ve Python ile geliştirilmiştir. PCL kütüphanesi, filtreleme (filters), özellik tahmini (feature estimation), anahtar noktalar (keypoints), kayıt (registiration), Kd Tree, Octree, bölütleme (segmentasyon), örnek fikir birliği (sample consensur), yüzey (surface), aralık resmi (range image) görselleştirme (Visualization) gibi birçok algoritma ve ayrı kütüphane içerir. PCL çapraz platformdur ve Linux, MacOs, Windows ve Android/IOS işletim sistemleri üzerinde derlenebilir. PCL'deki temel veri türü nokta bulutudur. Bir nokta bulutu C++

sınıfının bir şablonudur. Bu şablon genişlik (width), yükseklik (height), noktalar (points), yoğunluk (dense), alıcı merkezi (sensor_origin), alıcı yönü (sensor_orientation) gibi özellikler içerir. Bu özellikler aşağıdaki şekilde açıklanabilir (Goldberg, Tutorials, 2014):

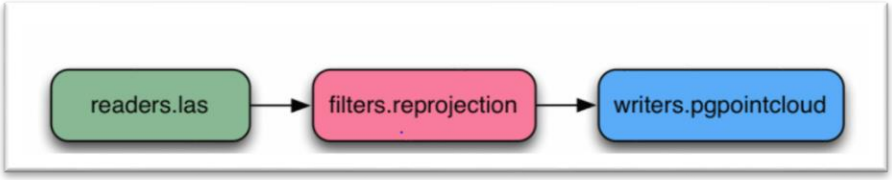
1. Width, int tipinde bir veridir. Organize edilmemiş (unorganized) veri setlerinde noktaların sayısıdır. Organize edilmiş (organized) veri setlerinde bir satırdaki noktaların sayısıdır.
2. Height, int tipinde bir veridir. Organize edilmemiş (unorganized) veri setlerinde 1. satırdaki değer iken organize edilmiş veri setlerinde satırların toplam sayısıdır.
3. Points, nokta tipinde bir veridir. Nokta tipindeki tüm noktaların saklandığı veri dizisini içerir.
4. Is_dense, boolen tipinde bir veridir. Veri setindeki bütün noktaların değeri (XYZ) varsa true, noktalardan bazılarının değeri yoksa false değerini alır.
5. Sensor_orientation, sensör elde etme pozu. Opsiyonel olarak kullanılır (Goldberg, Tutorials, 2014).

PCL'deki nokta tipi birincil nokta veri türüdür ve her bir nokta ögesinin ne tuttuğunu tanımlar. Bu kütüphane içinde tanımlı birçok nokta türlerini barındırır. Bu veri türleri PCL'de uygulanan tüm algoritma ve yöntemleri kullanmak için kütüphane içinde tanımlı nokta tipleridir. Bununla birlikte kullanıcının yeni tür tanımlama isteklerine karşı yeni nokta tipi ekleme özeliği de vardır. PCL'de tanımlı olan nokta tipleri aşağıdaki gibidir:

- PointXYZ- float x, y, z
- PointXYZI – float x, y, z, intensity
- PointXYZRGB – float x, y, z, rgb
- PointXYZRGBA – float x, y, z uint32_t rgba
- Normal – float normal[3], curvature
- PointNormal – float x, y, z normal[3], curvature
- Histogram – float histogram[N] vb. ‘dır (Goldberg, Tutorials, 2014).

2.3.2 PDAL

PDAL (Point Data Abstraction Library), nokta bulutu verilerini çevirmek ve değiştirmek için kullanılan C++ kütüphanesidir. Bir diğer ifade ile nokta veri soyutlama kütüphanesidir. Nokta bulutu verilerinin tercüme edilmesi ve işlenmesi için kullanılan bir uygulamadır. Kütüphanedeki araçların çoğu LIDAR verisi için olsa da sadece LIDAR verisi ile sınırlı değildir. PDAL, nokta bulutları üzerideki işlemleri aşama aşama boru hattı (pipeline) halinde oluşturulmasını sağlar. Bu boru hattı JSON ya da uygun API kullanılarak oluşturulabilir. PDAL yaklaşımı, her biri belirli işlevleri kapsayan bir dizi bileşeni zincirlemektir. Bileşenler, yeniden kullanım, birleştirme ve ayrılma işlemlerine olanak sağlamakatdır. Örneğin, LAS reader aşamasından, reprojection (veri işleme) aşamasından ve postgresql’e yazma aşamalarından oluşan bir iş akışı oluşturulmak istendiğinde bu işlemi gerçekleştirmek için tek parçalı özel bir program yazmak yerine, dinamik olarak bir dizi adım ya da işlem olarak oluşturmak daha hızlı bir yaklaşım sunar (Bell, Chambers, Butler, & Gerlek, 2018). Şekil 2.5’ te basit bir PDAL boru hattı (pipeline) gösterilmektedir.



Şekil 2.5 Basit bir PDAL boru hattı (Bell, Chambers, Butler, & Gerlek, 2018)

PDAL, filtreleme (filtering), kırpma (clipping), döşeme (tiling), işlem hattına dönüştürme (transforming) ve gerektiğinde yeniden kullanma gibi işlemler için ara aşamalar oluşturabilir. Bu iş akışı JSON formatta tanımlanabilir ve üzerinde düzenleme yapılabilir. Şekil 2.6' da PDAL boru hattının JSON olarak gösterim kodu gösterilmektedir.

```
{
  "pipeline":[
    {
      "type":"readers.las",
      "filename":"input.las"
    },
    {
      "type":"filters.reprojection",
      "out_srs":"EPSG:3857"
    },
    {
      "type":"writers.pgpointcloud",
      "connection":"host='localhost' dbname='lidar' user='hobu'",
      "table":"output",
      "srid":"3857"
    }
  ]
}
```

Şekil 2.6 PDAL boru hattının JSON olarak gösterim kodu (Bell, Chambers, Butler, & Gerlek, 2018)

PDAL, nokta bulutu veri çeviri iş akışlarında kullanışlı bir yöntemdir. Kullanıcıların birçok veri formatı sorunu için içeriğe soyut bir API sağlayarak veriye algoritma uygulamasına izin verir. PDAL, kullanıcı dostu bir arayüze sahip değildir. Filtreleme (Filters), okuma (Readers) ve yazma (Writer) işlemleri için kullanıcı bilgisine ihtiyaç duymaktadır bu durum da kaynak kodu okumayı gerektirmektedir. Aktif geliştirmeye açık kütüphane olduğu için istenilen şekilde düzenleme yapılabilir. PDAL, kullanıcıların nokta bulutu işleme iş akışlarını koordine etmesini ve oluşturmasını sağlayan bir dizi uygulama sağlamaktadır.

Bu uygulamalar ile, veri kümesi hakkında bilgi yazdırılabilir, bir nokta bulutu biçiminden diğerine veri çevirisi yapılabilir, DTM oluşturulabilir, nokta bulutundaki

gürültü kaldırılabilir, koordinat dönüşümü yapabilir (Bell, Chambers, Butler, & Gerlek, 2018).

2.3.3 GDAL

GDAL (Geospatial Data Abstraction Library), OSGeo (Open Source Geospatial Foundation) tarafından açık kaynak lisansı altında yayınlanan raster ve vektör coğrafi veri formatları için bir çevirmen kütüphanesidir. C/C++ da geliştirilen mekansal veri soyutlama kütüphanesidir. Bir kütüphane olarak, desteklenen tüm formatlar için çağrı uygulamasına tek bir raster soyut veri modeli ve tek vektör soyut veri modeli sunmaktadır. Sayısız raster formatın okunmasını, yazılmasını, tercüme edilmesini destekleyen GDAL bileşeni ve vektör veri formatının okunmasını, yazılmasını, tercüme edilmesini destekleyen ORG bileşeni olarak GDAL kütüphanesinde iki bileşen bulunmaktadır. GDAL, rasterleri ve geometrileri (nokta, çizgi, poligon) destekler. Yaygın olarak kullanılan mekansal operasyonları (union, intersection, join, clipping, contouring) gerçekleştirir. Çok sayıda farklı sürücüyü destekler (rasterler için veri formatları ve geometriler için veri formatları gibi) (Rouault, 2018).

Bir veri kümesi, GDAL veri sınıfı tarafından temsil edilir ve hepsinde ortak özellik bulunan ilgili raster bantların bir araya getirilmesiyle oluşur. Özellikle bir veri kümesi, tüm bantlar için geçerli olan raster boyutuna (piksel ve çizgiler halinde) sahiptir. Bir GDAL veri kümesi hepsi aynı alana ait olan ve aynı çözünürlüğe sahip olan raster bantların listesidir. Ayrıca metadata, koordinat sistemi, raster büyüklüğü ve diğer çeşitli bilgilere de sahiptir (Rouault, 2018).

Vektör verilerinde GDAL bir dizi OGR layer (katman) sınıfını temsil eder. Bu genellikle bir tek dosya ya da dosya setleri ya da veri tabanı şeklinde temsil edilir. Bu veri seti çağırıldığında kendisi değil referansı geri döner. GDAL veri seti soyut bir temel sınıftır. Uygulanan her dosya formatı sürücüsü için bir alt sınıfın uygulanması beklenir. GDAL veri seti nesneleri, normal olarak doğrudan değil bir GDAL Driver (sürücü) yardımıyla desteklenir.

Geometri sınıfları model vektör verilerini kapsar, ayrıca geometri işlemleri sağlar ve ikili formata (binary) çevrilir. Bu sınıfların içeriği,

- Geometri, sınıfı opengis vektör data modelini kapsar.
- Konumsal Referans (Spatial referance), projeksiyon ve datumu içerir.
- Özellik (Feature), öznitelik ve geometri özelliklerini içerir.
- Sınıf Tanımlama Özellikleri, ilgili özellik grupları için şema oluşturur.
- Katman, soyut temel sınıftır, veri setlerinin özelliklerini temsil eder
- Veri setleri, soyut temel sınıftır, dosya veritabanı bir ya da daha fazla obje ile temsil edilir.
- Drivers, belirli bir formata dönüşümü temsil eder, GdalDriverManager ile yönetilir (Rouault, 2018).

2.3.4 OPENGL

OpenGL, gelişmiş donanım desteğini kullanarak hem 2B hemde 3B grafikleri ekrana çizmek için kullanılan ücretsiz bir grafik uygulama geliştirme ara birimidir. Programlama dili değildir. OpenGL grafik donanımı için tasarlanmış bir yazılımdır. OpenGL, taşınabilir, etkileşimli 2B ve 3B grafik uygulamaları geliştirmek için önde gelen bir ortamdır. OpenGL endüstrinin en yaygın kullanılan ve desteklenen grafik uygulama programlama ara birimidir. Geniş bir dizi oluşturma, doku eşleme, özel efekt ve diğer güçlü görselleştirme işlevlerini birleştirerek uygulama geliştirmeyi hızlandırır.

3B animasyonlardan CAD'ye görsel simülasyondan maksimum performans gerektiren herhangi bir görsel bilgi işlem uygulaması, yüksek kaliteli, yüksek performanslı OpenGL kullanılarak oluşturulur. Bu yetenekler, yayıncı, CAD/CAM, eğlence, tıbbi görüntüleme ve sanal gerçeklik gibi farklı alanlarda ki geliştiricilere 2B ve 3B grafikler üretme ve görüntüleme olanağı sağlar. Platform ve işletim sistemi bağımsızdır, OpenGL, Linux, Windows, Mac OS gibi işletim sistemlerinde çalışır. OpenGL kullanan bir programı işletim sisteminde çalıştırmak için öncelikle işletim sisteminde programın çalışırken kullanacağı işlevleri içeren çalışma anı kütüphanesi (runtime-library) bulunması gerekmektedir. OpenGL kullanılarak yazılmış programlar Win32, MacOS ve X-Window pencere yöneticilerinde çalışmaktadır. OpenGL ile yapılan

uygulama düşük maliyetli bilgisayarlardan süper bilgisayarlara kadar her şeyde tutarlı ekran sonuçları vereceği için yaygın olarak kullanılmaktadır. OpenGL bir grafik API'si olduğu ve kendi başına bir platform olmadığı için, çalışmak için bir dil gerektirir. Birçok programlama dili de burada kullanılabilir. Ada, C, C++, C#, Fortran, Python, Perl ve Java gibi programlama dilleri kullanılarak OpenGL kütüphanesinden faydalanabilir (Luten, 2014).

OpenGL platformdan bağımsız olduğu için bazı işlemler bu kütüphane ile yapılamaz. Örneğin kullanıcıdan veri almak, bir pencere çizdirmek gibi işler hep kullanılan pencere yöneticisi ve işletim sistemine bağlıdır. Bu yüzden OpenGL 'in bu durumlarda platforma bağımlı olduğu düşünülebilir. Çünkü penceresini her pencere yöneticisinde farklı çizdirecek bir canlandırma programı yazmak demek her bilgisayarda çalışacak ayrı pencere kodu yazmak demektir. Bu gibi durumlarda platform bağımsızlığı için GLUT, SDL, SFML ve GLFW gibi kütüphaneler kullanılır. (Luten, 2014)

GLUT, bir çok işletim sistemine aktarılmış bir kütüphanedir. Amacı OpenGL bağlamı oluşturulmaya, pencere parametrelerini tanımlamaya ve ihtiyaç duyulan tüm kullanıcı girişlerinin yönetilmesine izin vermektir. Programların pencerelerini oluşturmak, klavye ve fareden veri almak gibi ihtiyaçları karşılamaktadır. GLUT olmadan da OpenGL programlama yapılabilir, örneğin Linux'ta kullanılan X-Window sistemin kendi işlevleri kullanılarak pencere çizdirebilir fakat bu kod sadece X-Window'da çalışır. Kod Windows'a götürülüp derlendiğinde aynı özellikleri sağlamayan sistemler olduğu için çalışmamaktadır.

OpenGL nesne hakkında bilgi saklamaz sadece yazılan komutlarla nesneyi oluşturur. OpenGL 'i kullanmanın genel yolu, çizilmesi gereken herşeyi çizmek ve daha sonra bu görüntüyü arabellek yardımıyla göstermektir. Resmi güncellenmek istendiğinde, yalnızca resmin bir bölümünde güncelleme istense bile herşey tekrar çizilir. Ekranda hareket eden nesnelere animasyon uygulanmak istendiğinde, ekranı sürekli olarak temizleyen ve yeniden çizen bir döngü kullanılır.

Ada, C, C++, C# (SharpGL adı verilen sınıflar sayesinde), Fortran, Python, Perl ve Java programlama dilleri kullanılarak OpenGL kitaplığından faydalanılabilir. Yüksek seviye komutlar içermez OpenGL 'de yalnızca en temel nesneler (nokta, çizgi, çokgen) bulunur. Bunlarla kompleks modeller (molekül, uçak, ev, araba vb.) oluşturulur.

Tipik bir program birçok çağrıda bulunur. Bunlardan bir kısmı programcı, bir kısmı işletim sistemi ve bir kısmı da programlama dilinin kendi kütüphaneleri tarafından yapılmaktadır. Windows uygulamaları, çıktıları ekranda göstermek için Grafik cihaz arayüzü denilen bir Windows API kullanırlar. Bu grafik cihaz arayüzü (GDI), pencereye yazılar yazmak, çizgiler çizmek için kullanılır (Vries, 2014).

2.3.5 WEBGL

WebGL (Web Grafik Kütüphanesi) herhangi bir eklenti kullanmadan, uyumlu bütün tarayıcılarda 3B ve 2B grafikler oluşturabilen platform bağımsız ve ücretsiz bir Javascript API'sidir. Khoronos Group tarafından geliştirilmiştir. Sistem, ekran kartının özelliklerini kullanmakta ve Javascript ara birimi üzerinde çalışmaktadır. WebGL, Javascript aracılığı ile OpenGL ES 2.0 desteği saylayarak çalışmakta olup, bu sayede eklentiye ihtiyaç duymamaktadır. OpenGL veya OpenGL ES destekleyen herhangi bir internet tarayıcısı 3B grafikleri desteklemektedir. Sistem HTML5 standartlarında açıklanan canvas elementini kullanmaktadır. Crome, Mozilla, Safari, Opera, Internet Explorer 11 ve üzeri tarayıcıları tarafından desteklenmektedir. WebGL, genel bir ifade ile rasterleştirme motorudur. Geliştirilen koda bağlı olarak noktaları, çizgileri ve üçgenleri çizer. WebGL bilgisayarda GPU üzerinde çalışır. GPU üzerinde çalışan kodun yazılması geliştiriciye kalmıştır. Bu kod fonksiyon çiftlerinden oluşur. Bu fonksiyonlar köşe gölgelendiricisi (vertex shader) ve parça gölgelendirici (fragment shader) olarak adlandırılır ve her biri çok açık bir şekilde yazılan C/C++ dilindeki GLSL olarak adlandırılır (GL Shader Language). Birlikte eşleştiklerinde program olurlar. WebGL' de nesneler her biri bir konumu ve rengi olan köşe kümeleri kullanılarak oluşturulur (Greggman, 2018).

WebGL idealde makinenin grafik hesaplama ve bellek donanımının Merkezi İşlem Biriminden (MİB/CPU) ayrıldığı donanım mimarileri için tasarlanmıştır. Grafik İşlem Birimi (GİB/GPU) programın birçok kopyasını eş zamanlı koşturabilecek şekilde tasarlanmıştır. Normal bir CPU' da çalışan programlardan farklı olarak küçük ve basittirler. Bu tarz bir mimaride ortaya çıkabilecek en büyük sorun 3B grafiksel uygulamalar içi CPU ile GPU arasında ortaya çıkacak haberleşme problemidir. WebGL in temel fikri de burada oluşmaktadır. Amaç iki birim arasında gerçekleşen haberleşme yükünü azaltmaktır. Bu işlem, komutları ve grafik kaynaklarını (çizim nesnesi, köşe nokta bilgileri) sürekli parçalı olarak CPU GPU arasında göndermek yerine gerekli tüm grafiksel veriler GPU üzerine bir defada kopyalanır. Kümeler

halinde gerçekleştirilen bu işlem ile haberleşme azalır ve CPU bağımsız grafik işlemleri gerçekleştirilir (Greggman, 2018).

WebGL uygulamaları 3B grafikler oluşturmak için canvas elementi, Javascript betik dilini ve GLSL shading, OpenGL tarayıcı dilini kullanmaktadır. Canvas elementi web sayfalarında çizim işlemlerinin gerçekleştirileceği alanın belirlenmesi ve erişimi için kullanılır. HTML5 ile tanıtılan canvas elementi ve canvas API kullanılarak WebGL kullanılmadan 2B çizimler de gerçekleştirilebilir.

WebGL ile çizim işlemleri GLSL shading dili ile shaderlar kullanılarak gerçekleştirilir. Shaderlar programlanabilir GPU'ların ortaya çıkması ile kullanılmaya başlanmıştır, vertex ve piksellere hükmetmek (programlanabilir hale getirmek) üzere yazılan küçük program parçalarıdır. Shaderlar ayrıca CPU ve GPU arasındaki iletişimi minimize etmek içinde kullanılır ve GPU üzerinde yoğun paralellikte çalışan küçük program parçalarının yazılmasına olanak sağlamaktadır.

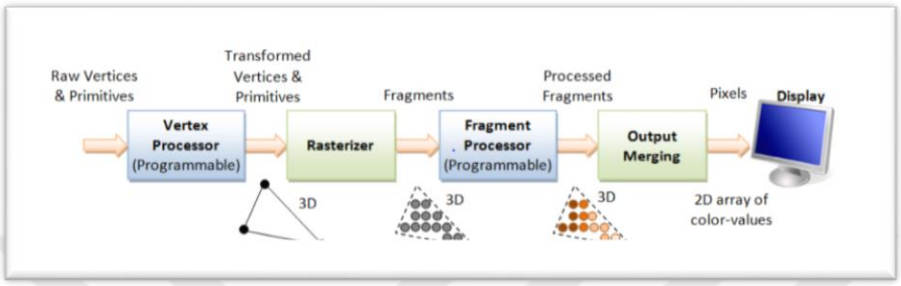
2.3.5.1 Vertex Shader

Bir köşe gölgelendiricinin görevi, köşe pozisyonlarını hesaplamaktır. Vertex shaderlar vertex (poligonların köşe noktaları) özelliklerinin tanımlanması ve işlenmesini sağlarlar (mesh koordinatlarının set edilmesi). Bu işlemler rendering sürecinde çalıştırılacak olan main() fonksiyonları içinde gerçekleştirilir. Vertex shaderlar, WebGL rendering boru hattının (pipeline) ilk aşamasını oluşturur ve her bir vertex üzerinde bir dizi matematiksel işlem gerçekleştirilir. Örneğin, vertex konumlarının ekran konumuna dönüşümü, görselleştirme için gerekli koordinatların üretilmesi, vertexlerin ışıklandırma ortamındaki renk değerlerinin hesaplanması işlemleri gibi. Bu fonksiyon çıktısının pozisyonunu baz alarak WebGL, noktaları çizgileri veya üçgenleri içeren çeşitli kökleri (primitive) rasterleştirebilir. Bu kökleri rasterleştirirken, bir parça gölgelendirici olarak adlandırılan ikinci bir kullanıcı tarafından sağlanan fragment shader fonksiyonunu her bir piksel için çağırır.

2.3.5.2 Fragment Shader

Rasterization sonrasında ilkel geometrik şeklin (üçgen vb.) kapladığı alan piksel boyutundaki fragmentlere parçalanır. Her bir fragment konum, derinlik değeri bilgisi ve renk, texture koordinatı gibi parametreler içerebilir. Diğer bir deyişle zaman içindeki ışıklandırma, transformasyon vb. değişimler sonucunda oluşacak olası yeni

pikselin hesaplanması fragmentlar ile gerçekleştirilir. Bu parça gölgelendiricinin (fragment shader) görevi, o anda çizilen ilkel geometrik şeklin her bir pikseli için bir renk ve derinlik bilgisini hesaplamaktır. Bu işlemlerden sonra fragmentler ekran üzerinde görüntülenmek için frame buffer'a gönderilir. Tüm WebGL API' sinin tamamı bu fonksiyonların birlikte çalışmasını ayarlamak ile ilgilidir (Cozzi, 2016).



Şekil 2.7 Vertex ve fragment shader kullanılarak gerçekleştirilen boru hattı (Vries, 2014)

Vertex ve fragment shader kullanılarak gerçekleştirilen basit bir 3B grafik rendering boru hattı (pipeline) şekil 2.7 'de gösterilmiştir. Bu iş akışında ki işlemlerin detaylı açıklaması aşağıdaki gibidir:

Vertexlerin işlenmesi: Modeli oluşturan her bir bağımsız vertexler üzerinde işlemler vertex shaderlar ile programlanır.

Rasterization: Her bir geometrik şeklin fragmentlere ayrılması işlemidir. Vertexlerin kapladığı alandaki piksel değerlerinin belirlenmesidir.

Fragmentlerin işlenmesi: Birbirinden bağımsız her bir fragmentin ortama göre ilgili renk değerlerinin fragment shaderlarla hesaplanmasıdır.

Çıktıların Birleştirilmesi: Tüm ilkel çizim nesnelerine ait 3B uzaydaki fragmentler kullanıcı ekranında 2B renk-piksel bilgisine dönüştürülür.

Bu işlevlerin erişmesini istediğimiz herhangi bir veri GPU' ya sağlandığında iş akışı tamamlanmış olur.

Shaderların veri alabilmesi için dört yol vardır.

1. Öznitelikler ve Tamponlar (Attributes and Buffers)

Tamponlar, GPU' ya yüklenen ikili verilerin dizisidir. Konumları, normları, doku koordinatları, köşe renkleri içerir. İçine istenilen herşey konulabilir. Öznitelikler, tamponların verileri nasıl çekeceğini ve bunları vertex shaderlara nasıl sağlanacağını belirtmek için kullanılır.

2. Uniformlar

Shaderlar programını çalıştırmadan önce belirlenen etkin global değişkenlerdir.

3. Dokular (Textures)

Dokular, shader programına rastgele erişebilen verilerin dizisidir. Bir dokuya yerleştirilen en yaygın şey görüntü verileridir, ancak dokular sadece veridir.

4. Değişkenler (Varyings)

Değişkenler, vertex shadercının verileri bir fregmant shadera iletmesinin bir yoludur. İşlenmekte olana, noktalara, çizgilere veya üçgenlere bağlı olarak, vertex shedar yürütürken, bir vertex shader tarafından değişen değerler üzerinde ayarlanan değerler enterpole edilir (Greggman, 2018).

2.4 Nokta Bulutu Verilerinin Depolanması

Nokta bulutu verileri karmaşık (multi-dimensional), büyük ölçüde (milyon ve milyarlarca) verilerdir. Bu verilere erişim ve bu verileri analiz etmek özel işlemler gerektirir. Artan LIDAR tarama yoğunlukları, büyük hacimli nokta verileri için yeni veri yönetimi zorlukları getirmektedir. Nokta bulutu elde etmekten analize, metedata erişim ve yönetimine kadar geçen süre, diğer coğrafi konum ve öznitelik verileriyle birleştirme, çoklu kullanıcı erişimi ve güvenlik, büyük ölçekli LIDAR verilerinin yönetilmesindeki zorluklardan bazılarıdır. Nokta bulutu verileri raster verilere benzemektedir. Her iki veri türü genellikle oldukça büyüktür ve statiktir. Çoklu kullanıcılı bir bağlamda (contex) noktaların ayrı ayrı güncellenmesi eklenmesi ve silinmesi çok gerekli bir özellik olmadığı için veritabanı yönetim sistemi (DBMS) işlem desteğine ihtiyaç duyulmamaktadır. Bu nedenden dolayı nokta bulutu veri yönetimi için belirli dosya tabanlı çözümler kullanılabilir. LAZ, LAS gibi. Fakat bu durumda veri kümeleri gittikçe büyüme gösterir. Daha fazla verinin yönetimi daha da zorlaşmaktadır. Bu nedenle büyük ölçekteki verilerin depolanması için veri tabanı yönetim sistemi kullanılması gerekmektedir. Örnegin 60.000 LAZ dosyasında

saklanan ve dağıtılan (distributed) verilerin içinden basit bir seçim yapmak etkili bir yöntem değildir. Böyle bir sistemde basit bir noktayı seçmek için veri tabanı yönetim sistemini (DBMS) kullanmak daha kolay ve etkili sonuçlar vermektedir. Ayrıca, belirli bir dosya tabanlı çözüm bir çok yönüyle iyi olmasına rağmen farklı veri kümesine ve türüne sahip verileri yönetim verileri, vektör veriler, tarama veriler, geçici veriler gibi desteklememektedir. Bu nedenle, vektör ve nokta bulutu verilerini birleştirip, sorgulanmak istendiğinde, sorgulamalarda standart ve genel bir veri tabanı yönetim sistemi (DBMS) çözümü tercih edilmektedir. Büyük hacimli nokta bulutu verileri göz önünde bulundurulduğunda, hem veri depolama, aktarma boyutu hemde hız açısından (yüklenme ve sorgu) performans önemlidir. Etkili bir uygulama yapmak için çeşitli teknikler/teoriler kullanılmaya başlanmış ve Oracle Spatial ve PostgreSQL veri tabanlarında SDO_PC ve PCPATCH gibi veri türleri oluşturulmuştur.

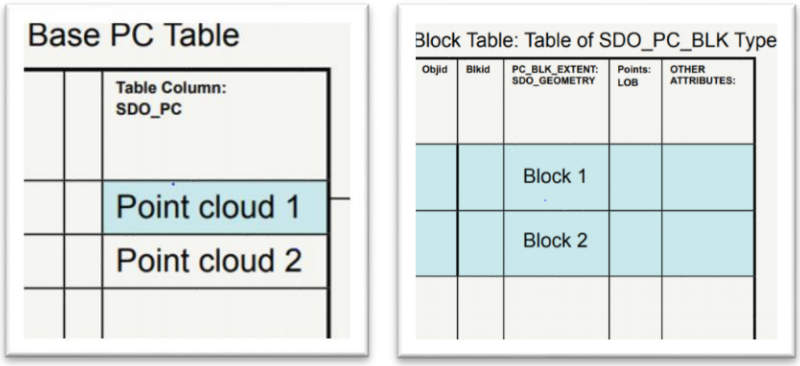
2.4.1 RDBMS ile Depolama (Oracle Spatial ve PostGIS)

RDBMS, ilişkisel veritabanı, organize edilmiş verilerin tablolarda saklanması ve bu tablolar arasında kurulan bağ ile oluşan veritabanı çeşididir. Bu modelde veri tabanı tablolar halinde tutulur. Her tablo satır ve sütunlardan oluşur. Tablodaki kolonlar başka tablolardaki kolonlar ile ilişkilidir . Oracle Spatial, vektörel ve raster tabanlı coğrafi veriler, topoloji ve ağ modelleri de dahil olmak üzere yer küre üzerindeki tüm konum bilgilerini yönetmek için oluşturulmuş bir veritabanı teknolojisidir. En genel anlamda Oracle Veri tabanına mekansal yetenekler ilave eden bir Oracle opsiyonudur. Teknik olarak, çok boyutlu verilerin veya nokta, çizgi, poligon gibi coğrafi şekillerin oracle veri tabanında saklanması, sorgulanması, güncellenmesi gibi işlemlerin yapılabilmesini sağlayan, özel bir SQL şeması ve özel SQL fonksiyonlarını içeren oracle teknolojisidir (Sharma, 2002).

Geometrik veri tiplerin saklanması ve tanımlanmasını sağlayan oracle şeması mekansal indeksleme, komşuluk ve yakınlık analizi, mekansal sorguların yapılabilmesini sağlayan özel coğrafi fonksiyonlar olmak üzre 4 bileşenden oluşur.

Mekansal veri modeli, elemanlar, geometriler ve katmanlardan (layer) oluşan hiyerarşik bir yapıdır. Mekansal tabakalar, sırasıyla elemanlardan oluşan geometriler topluluğudur. Aynı karakteristiğe sahip geometrilerin bir araya getirilmesiyle oluşmaktadır. Bir geometri homojen ya da heterojen elemanlar topluluğu olabilir.

Oracle Spatial, LIDAR taramalarının oluşturduğu çok büyük miktardaki veriyi yönetmek için SPO_PC adında yeni bir veri türü kullanır. SDO_PC nokta bulutu verilerini verimli bir şekilde yönetmek için tasarlanmıştır. Oracle'da tek bir nokta bulutu nesnesinde 4.10^{18} noktaya kadar veri saklanabilir. Noktaları sabit boyutlu bloklara ayırmak ve blokları ayrı bir blok tabloda satırlar halinde saklamak, bu ölçeklenebilir depolamayı sağlar. Blok kapasitesi, toplam boyut sayısı gibi meta veriler ve blok tablo bilgisi SDO_PC sütununda saklanır. Veri noktaları konumsal bölümleme teknikleri kullanarak alt kümelere bölünmüştür, her alt küme ilişkili blok tablosunun puan sütununda saklanır. Blok tablosundaki extent sütunu her bir alt kümenin kapsamını saklar ve konumsal olarak indekslenir. Bu extent sütunu iki aşamalı bir fitre işleminde verimli sorgulama işlemine izin vermektedir. Max-res (maksimum resolution), min-res (minumum resolution) nokta bulutlarının birden fazla çözünürlükte saklanmasını sağlar. Çoklu çözünürlüklü nokta bulutları bir bakış noktasına yakınlığa dayalı olarak erişmeleri gereken (daha yüksek çözünürlük için daha yakın ve daha az ayrıntı için daha uzak) bazı uygulamalarda kullanışlıdır. Bir nokta bulutunun bu bölümlenmiş veya bloke edilmiş depolanması ölçeklenebilirliği sağlar (Sharma, 2002). Şekil 2.8' de nokta bulutunun birden çok bloğa ayrılmış hali gösterilmektedir.



Şekil 2.8 Nokta Bulutunun birden çok bloğa ayrılmış hali. (Sharma, 2002)

Her blok ObjId ve BlkId tarafından özel olarak tanımlanmıştır. Lobdaki her nokta isteğe bağlı ayrı bir tabloda saklanan özniteliklere sahip olabilir.

Oracle, SDO_PC türünde bir PL/SQL sorgusu sağlamaktadır. Bu sorgu, mekansal (spatial) bir pencereyi ve bu pencere ile herhangi bir etkileşime sahip tüm nokları tanımlar. Bu mekansal pencereye ek olarak, çözünürlük aralığını karşılayan blokları tanımlamak için bir çözünürlük aralığı da sunmaktadır. Bir nokta bulutu, ilgili blok tabloda birden çok satır olarak saklanır. Her satır o satırdaki noktaların kapsamını temsil etmek üzere, SDO_GEOMETRY türünde bir blk-extent sütunu içerir. Bu sütun Oracle R-tree veri yapısı kullanılarak indekslenir.

PostgreSQL, tarafından geliştirilen pgpointcloud uzantısı üzerinden nokta bulutu desteği sağlar. Geliştirmenin arkasındaki fikir, noktaların satır başına PostGIS nokta olarak saklanmaması, parçalar (patches) halinde depolanabilmesidir. Oracle'daki blokların karşılığı şeklinde düşünülebilir. (Ramsey, 2017).

Satır başına bir nokta depolanamaz çünkü milyarlarca satırlık bir tablo pratikte kullanmak için çok büyüktür, tablo boyutu çok büyük olacak ve buna bağlı olarak indeks çok büyük olacaktır. Bu nedenle noktalar her biri yüzlerce noktadan oluşan parçalar halinde düzenlenmesi gerekmektedir. Bu milyonlarca satırlık tablo, daha fazla işlenebilir olan on milyonluk bir tablo haline dönüştürülür (Ramsey, 2017).

Bu nedenle PC Point ve PC Patch veri tipleri geliştirilmiştir. Veriler bayt diziler halinde paketlendiğinden, bu paketlemenin nasıl yapıldığının bir açıklaması bir XML şema belgesiyle açıklanmaktadır. Bu belge, nokta bulutlarının çok boyutluluğunu dikkate almak için kullanılır. Nokta bulutunun veritabanına gerçek yüklenmesi, iyi bilinen ikili (WKB) nesnelerden veya pgpointcloud için PDAL sürücüsünden yararlanılarak gerçekleştirilebilir. Noktalar ve parçalar ortak bir tanımlayıcı kullanılarak birbirine bağlanır. Her bir şema belgesi, her bir şema ve konumsal referans sistemine benzersiz (unique) bir nokta bulutu kimliği (pcid) atanan pointcloud-formats tablosunda bir satırda saklanır.

Point cloud format tablosu bahsedilen noktalar için şema bilgilerini tutar. Bir nokta veya parça oluşturmadan önce, tutulacak boyutları tanımlamak için bir şemaya ihtiyaç vardır. Şema oluşturduktan sonra tablo oluşturulabilir. Veri tabanına veri girmek için çoklu giriş formatlarını, LAS dosyaları ve çıktı biçimlerini, PostgreSQL Point Cloud

işlem adımlarında PDAL açık kaynak kodlu LIDAR işleme aracı kullanılır. Nokta bulutu verileri için kullanılan eklentiler sadece PgSQL 9.1 ve üstü versiyonlar için desteklenir (Ramsey, 2017).

2.4.2 NoSQL ile Depolama

NoSQL veri tabanları büyük verileri kontrol edebilme, ölçekleyebilme, veri formatları yönetebilme, sorgulara daha hızlı cevap alabilme, eş zamanlı ekleme ve güncelleme işlemlerini gerçekleyebilme, açık kaynak kodu geliştirmeyi sağlayabilmektedir. İlişkisel veri tabanları işlem (transaction) tabanlı çalışan sistemler olup bu işlemlerin düzgün çalışmasını ve veri bütünlüğünü sağlamaktadır. NoSQL sistemler bu kurallara uymamaktadır. İlişkisel veri tabanı yönetim sistemlerinde veriler tablolarda bulunurken, NoSQL sistemler sabit tablo tanımlarına bağlı değildir. İlişkisel veri tabanlarında veri normalizasyon kurallarına göre tutulurken kullanırken NoSQL veritabanlarında denormalize olarak tutulur. Bu şekilde tutulma veriye erişimi kolaylaştırdığı için sistemin performans kaybı olmaz (Amirian, Pouria, & Anahid, 2013).

NoSQL veritabanlarında en yaygın kullanılan veri tabanları, anahtar-değer (key-value), belge, tablo ya da sütun ve grafiklerdir.

Anahtar-değer veritabanı, en basit NoSQL veritabanı türüdür. Bu tip veri tabanı anahtarlar kullanarak şema içermeyen verileri depolar. Anahtar genellikle bir string veri tipidir, ve saklanan değerler, önceden tanımlanmış bir şema olmazsızın int, string veri tipi veya bir BLOB (binary Large Object) gibi herhangi bir geçerli tip olabilir. Depolanan verilere erişmek için basit bir API sağlar.

Anahtar-değer veri tabanları, coğrafi veriyi depolamak için kullanılabilir, ancak karmaşıklığı özellikle poligon aramalarını engellemektedir. Bu nedenle performans arttırmak için mekansal indekleme yapılması gerekmektedir. Çoğu durumlarda ilişkisel veri tabanından daha düşük performans sağlayabilir. Büyük miktardaki veriyi eklemek silmek için ideal bir yöntemdir, fakat mekansal aramalar ve analizler için iyi bir çözüm değildir.

Belge veritabanı, veritabanının değerlerini anladığı bir anahtar-değer veritabanıdır. Başka bir değiş ile, veri tabanındaki değerler XML, JSON ya da BSON (binary Json) gibi önceden tanımlanmış formatlara dayanmaktadır. Bu özelliği, anahtar-değer veritabanına göre bir çok avantaj sağlamaktadır. Uygulama katmanına çok fazla

mantık koymak yerine, bir çok işlem veritabanının kendisi tarafından yapılabilir. Bu tür bir NoSQL veritabanındaki sorgular oldukça esnektir ve bazı belge veritabanlarının kendi sorgu dilleri vardır. Anahtar-değer veritabanlarına benzer şekilde, veri eklemek için önceden tanımlanmış bir şemaya uymaya gerek yoktur. Transaction işlemler bir belge veya belge parçası için geçerlidir (Amirian, Pouria, & Anahid, 2013).

Belge veritabanları, coğrafi veriyi anahtar-değer veritabanlarından daha etkin bir şekilde yönetmek için kullanılmaktadır. Belge veritabanındaki coğrafi veri esnek sorgu kullanılarak alınabileceğinden, çeşitli kullanım durumlarında coğrafi veri depolamak ve yönetmek için kullanılabilir. Aslında bir çok belge veritabanı coğrafi olarak ya da eklentiler yoluyla coğrafi verileri desteklemektedir. İlişkilendirme ve birleştirme (join) işlemleri bu tarz veritabalarında kullanılmadığı için CBS iş akışlarında da genellikle ilişkiler ve birleştirmeler kullanıldığı için belirli türdeki belge veritabanlarının bu özellikleri desteklemesi gerekmektedir. Yerel XML veritabanları, önceden tanımlanmış XML şemalarının kümesine uymaya yönelik yapılandırılmıştır. Coğrafi işaretleme dili (GML), coğrafi verilerin depolanması, modellenmesi ve değiştirilmesi için standart bir mekanizma olarak, XML tabanlı bir dildir ve XML veri tabanları GML formatında coğrafi veri yönetimi için kullanılmaktadır.

LIDAR nokta bulutlarının dosya merkezli saklanması için ana fikir, büyük miktarda LIDAR verilerinin genellikle terabayt boyutunun tek dosyaları yerine büyük dosya koleksiyonları olarak sunulmasıdır.

2.4.3 3B Nokta Bulutu Verilerinin İndekslenmesi

Veri tabanı indekslemesi, veri alma işlemlerinin hızını artıran bir veri yapısıdır. Sık kullanılan sorgular için yüksek performanslı okuma işlemleri için indeksler gerekmektedir.

İndeks kullanılmazsa sorgu sonucuna ulaşmak için tüm veri tabanı tabloları taranır. Bu işlem de çok fazla veri için yapıldığında oldukça yavaş sonuç getirmektedir. Ancak indeks kullanıldığında, veritabanındaki tüm kayıtları taramak zorunda kalmadan kesin kayıtlara ulaşıldığından veri okuma performansını artırır.

Mekansal verilerin çok boyutlu olması sebebi ile tek boyutlu normal indeks yapısının kullanımı bu tür verilerde uygun olmamaktadır. Çok büyük boyutta mekansal veriler üzerinde sorgulama yapıldığında, doğru indeks kullanımı verimli sorgulamaların

yapılmasında büyük etkendir. İndexleme işlemleri için çeşitli yöntemler kullanılmaktadır.

En iyi bilinen tek boyutlu indeksler, Bayer ve McCreight (2002) tarafından icat edilen B-Tree' dir. B-Tree en az iki boyuta sahip olan konumsal nesneler için uygun değildir. Yıllar boyunca pek çok indeksleme tekniği geliştirilmiştir. Bunlardan en önemlisi konumsal yakın nesnelerin gruplaşmasını ve konumsal nesnelerin minumum sınırlayıcı diktörtgeni ile temsil edildiği R-Tree ve Q-Tree'dir. Oracle veri tabanı, konumsal (spatial) veriler üzerinde R-Tree ve Q-Tree indeks kullanımını desteklemektedir.

R-Tree İndex Yapısı

R-Tree indeksler B-Tree indeks yapısından türetilmiştir. Konumsal veriler için yapılan performans testlerinde büyük oranda başarılı olmuş, en yaygın kullanılan algoritmadır. Oracle R-Tree indeks ağaç yapısını veritabanında "MDRT" ile başlayan veritabanı tablolarında tutmaktadır. Bu tablolar veritabanında başka bir yere taşınmamalı ya da değiştirilmemelidir.

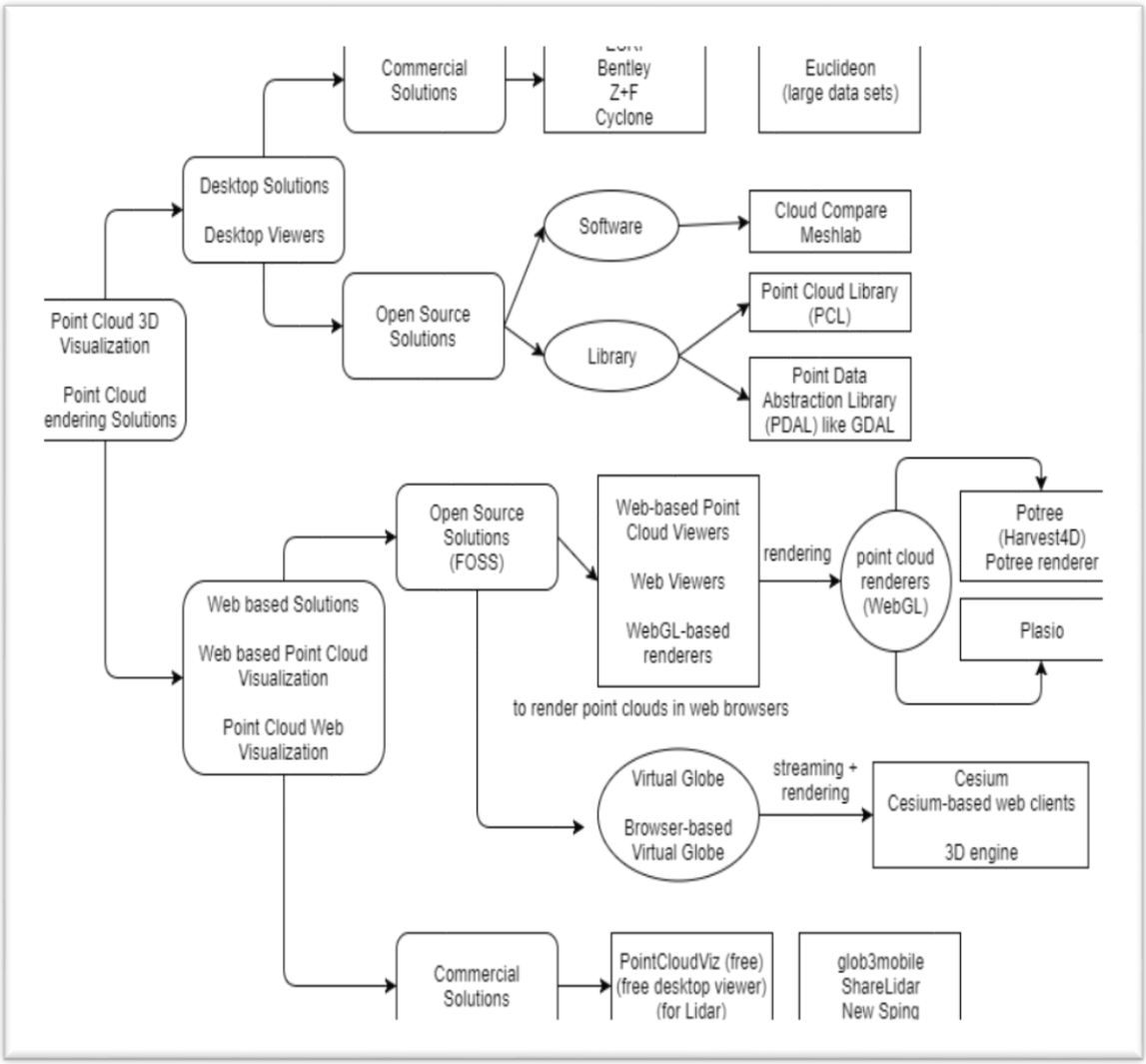
Q-Tree ve R-Tree bazı yönleri ile birbirinden farklıdır. Q-Tree yapıda geometriler, her biri aynı boyuttaki tile'lar ile ifade edilirken, R-Tree yapıda Minumum Bounding Rectangle' lar ile tanımlanmaktadır. Tile yapısı ile Q-Tree, join işlemlerinde çok başarılıdır. Özellikle belirlenen geometri için en yakın komşuları bulma sorgusu R-Tree indeks yapılarında çok daha hızlıdır. R-Tree bütün yer kürenin indekslenmesi için uygunken Q-Tree uygun değildir. Çok fazla güncelleme işlemleri yapıldığında R-Tree ağaç yapısında bozulmalar olabilmekte ve performansın iyileştirilmesi için indeks'in yeniden düzenlenmesi gerekirken, Q-Tree indeks performansı etkilememektedir. Önemli olan, bu indeksleme mekanizmalarının hangisinin yapılacak çalışma için uygun olduğudur (Boehm, 2016).

3. WEB ÜZERİNDE GÖRSELLEŞTİRME

Lazer tarayıcılar veya fotogrametri gibi 3B tarama teknolojileri genellikle yüz milyonlarca veya milyarlarca noktayı aşan miktarda veri üretir. Nokta verilerinin doğası gereği, basit modelleri bile doğru bir şekilde temsil etmek için çok sayıda noktaya ihtiyaç vardır. Nokta bulutu verilerinin görselleştirilmesi, genellikle bunları işlemek zorunda olan sistemlerin ana belleğinden daha büyük olan boyutlarından dolayı karmaşıktır. Bu nedenle görselleştirme için yüksek kapasiteli sunucular kullanılır. İşlenecek olan elde edilen nokta bulutu, sonraki veri analizinin, 3B örgü modellemesinin ve 3B görselleştirmenin temelidir. Bu nedenle, veri kalitesi ve işlem planının hızlı bir şekilde değerlendirilmesi için eklenti kurma zorunluluğu olmadan, kullanıcı dostu nokta bulutu görselleştirme aracı kullanılmalıdır (Martinez, 2016).

3.1 Nokta Bulutu Verilerini Web Ortamında Görselleştirme Yöntemleri

Nokta bulutu görselleştirmesi Scanopy, Arena4D, Geoverse gibi masaüstü çözümler ile sınırlı iken WebGL teknolojisinin gelişmesiyle çeşitli web renderer'lar bu işlemler için kullanılmaya başlanmıştır. WebGL ile web tarayıcıları üzerinden 3B içerik sunumu, ek yazılım yüklemeye gerek kalmadan geniş bir kitle ile paylaşmak popüler ve kullanışlı hale gelmektedir. WebGL, tüm büyük tarayıcılar, masaüstü ve hatta mobil cihazlarda yerel olarak desteklenen bir standart haline gelmiştir. 3B nokta bulutlarının web üzerinde 3B görselleştirilmesi için Potree, Plasio, CesiumJS, Octree, ThreeJS gibi WebGL tabanlı uygulamalar kullanılmaktadır. Şekil 3.1' de nokta bulutlarının görselleştirilmesine ilişkin kullanılan yöntem ve araçlar gösterilmektedir.



Şekil 3.1 Nokta bulutu görselleştirme yöntem ve araçları

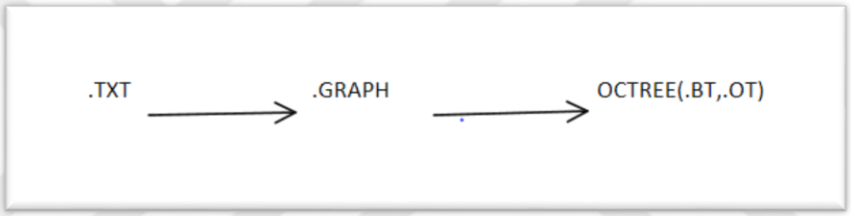
3.1.1 Octomap

Octamap kütüphanesi, özellikle robotik için uygun C++ dilinde veri yapıları ve haritalama algoritmaları sağlayan bir 3B doluluk grid harita yaklaşımı uygulamasıdır. Robotik uygulamalarda genellikle boş, dolu ve henüz keşfedilmemiş alanların modellendiği olasılık tabanlı gösterim kullanılmaktadır. Robotun navigasyonu için bu mekansal gösterim üzerinde engellerden sakınan yol planlama algoritmaları kullanılmaktadır. Ancak yalnız 3B nokta bulutları kullanılarak, sözü edilen olasılık tabanlı modellemedeki engellerin olmadığı alan ile keşfedilmemiş olan arasındaki fark ayırt edilemez. Bu nedenle octree mekansal veri yapısı ile olasılık tabanlı doluluk kestiriminin birleştirildiği Octomap yaklaşımı 3B hacimsel gösterim için

kullanılmaktadır. Birleştirilmiş nokta bulutları voksellere dayalı olarak ortamın 3B modellenmesini sağlar. Octamap kendi alt klasörlerinde iki ayrı kütüphaneden oluşmaktadır. Octamap gerçek kütüphane, octovis görselleştirme kütüphanesidir. Octovis QT ve libQGLViever tabanlı Octomap kütüphanesi için görselleştirme aracıdır. Octovis ubuntu işletim sistemi üzerinde önceden oluşturulmuş paketleri kullanarak koşturulabilir (Hornung, Wurm, Bennewitz, Stachniss , & Burgard, 2013).

Octovis Kullanarak Nokta Bulutu Verilerinin Görselleştirilmesi

Octovis ubuntu üzeride hazır paket halinde geldiği için Linux işletim sistemli bir bilgisayarda bu uygulama terminal penceresine "octovis" yazılarak çalıştırılır. Octamap octree veri tipini kullandığı için octovis kullanarak veri görselleştirilebilmesi için bu verilerin öncelikle octree (.ot, .bt) formatına donüştürülmesi gerekmektedir.



Şekil 3.2 .txt uzantılı nokta bulutu verileri octree veri yapısına dönüştürülmesi

Bunun için var olan .txt uzantılı veriler öncelikle .graph formatına daha sonra octree formatına dönüştürülür. Öncelikle verilerin içinde bulunduğu .txt uzantılı dosyanın bulunduğu klasörde terminal penceresi açılır ve .txt formatından .graph formatına veri dönüştürmek için "*log2graph input.txt output.graph*" komutu yazılır. Input.txt, .txt formatındaki verilerin bulunduğu dosyanın adı. output.graph, .graph formatında oluşacak yeni dosyanın adıdır. Daha sonra oluşan output.graph veri tipinden .bt, .ot uzantılı veri elde etmek için terminale "*graph2tree -i output.graph -o output.bt -res 01*" komutu yazılır. Output.graph, graph formatında dönüştürülecek veriler, output.bt, .bt formatında yeni oluşacak dosyanın adıdır. Bu işlemlerden sonra veriler octree formatına getirilmiş olur ve octovis görselleştirme uygulamasında direk kullanılabilir (Hornung, Wurm, Bennewitz, Stachniss , & Burgard, 2013). Şekil 3.3' de .ot, .bt formatına dönüştürülmüş XYZ formatındaki nokta bulutu verilerinin octovis görselleştirme kütüphanesi kullanarak webde görselleştirilmiş hali gösterilmektedir.

Şekil 3.3 Octomap kullanarak 3B nokta bulutunun görselleştirilmesi

3.1.2 Potree

Potree, Bilgisayar Grafikleri ve Algoritmalar Enstitüsü tarafından geliştirilen geniş nokta bulutları için ücretsiz açık kaynaklı bir nokta bulutu renderer'dır. Potree kullanarak yüksek çözünürlüklü render işlemi ve noktaların RGB, intensity, sınıflandırma gibi özellikleri görselleştirilebilmektedir. Potree büyük veri kümeleriyle başa çıkabilmek için çok çözünürlüklü bir octree veri yapısı kullanır. Bununla birlikte, çok çözünürlüklü octree veri yapısını oluşturmak için gereken hesaplama miktarından dolayı kullanılabilirliği bir kaç milyar nokta ile sınırlıdır. Potree web browser'larda (Crome, Firefox, Safari, Edge) eklentiye gerek olmadan kullanılabilir. LAZ, LAS data tipleri için data dönüşümüne ihtiyaç duymaktadır. LAZ, çoğu nokta bulutu uygulaması tarafından desteklenen, yaygın olarak kullanılan bir nokta bulutu biçimidir. Potree uygulamasında tüm veri ekrana yüklenmemektedir. Uzaktayken düşük çözünürlük yakındayken daha yüksek çözünürlük olacak şekilde veriler yüklenir. Uygulama arayüzü üzerinden mesafe, alan ve kesit ölçüleri gerçekleştrilmektedir.

Potree Converter, nokta bulutu dosyalarının giriş setini (LAS, LAZ, PTX, PLY) gibi Potree uygulamasının ihtiyaç duyduğu gerekli çoklu çözünürlüklü octree veri yapısına dönüştürmek için kullanılır. Octree' nin her düğümü bir dosyada saklanır. Octree

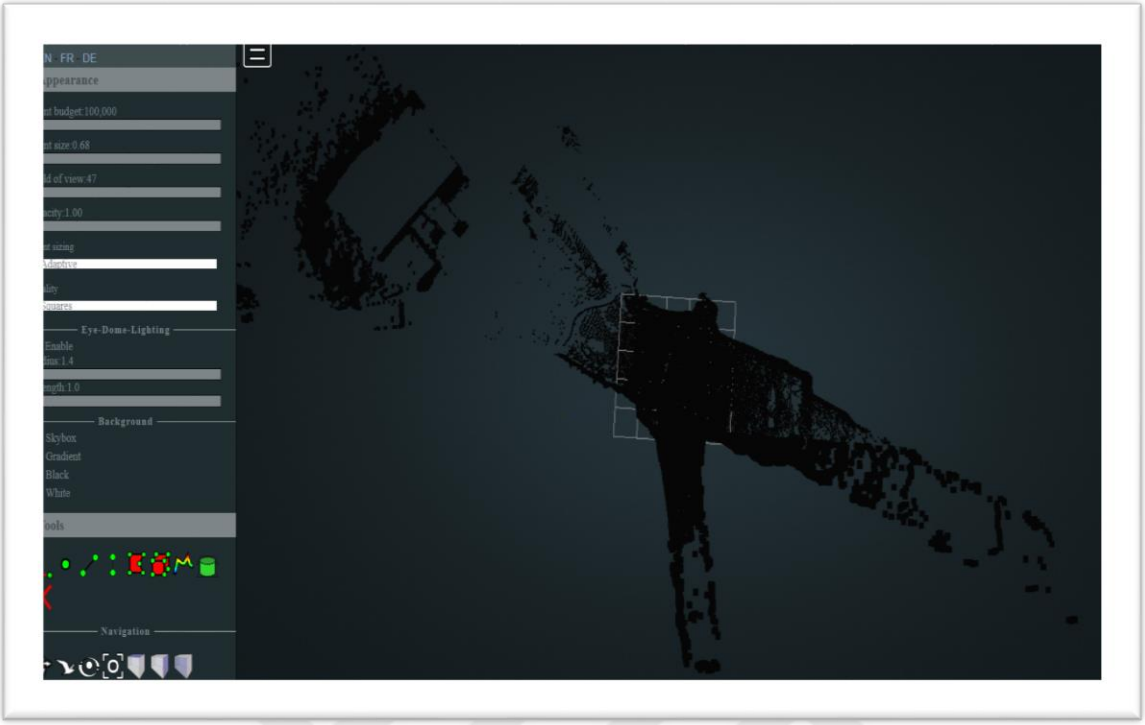
dosyasının biçimi, belirli ikili dosya formatı (binary) olan LAS veya LAZ olabilir. Görüntü yükleme süresini azaltmak için, tüm octree hiyerarşisi yardımcı çoklu dosyalarda saklanır. Octree'nin oluşturulmasını sağlayan çeşitli giriş parametreleri vardır. Boşluk (space) parametresi çözünürlüğü tanımlar, oluşturulan octree'nin kök seviyesindeki noktalar arasındaki minumum mesafeyi belirtir. Nokta – düğüm hesaplaması boşluk parametresini kullanarak gerçekleştirilir. Potree Converter, her noktanın hangi seviyede ve depoda saklanması gerektiğini hesaplar (Schütz, 2014).

Potree Kullanarak Nokta Bulutu Verilerinin Görselleştirilmesi

Potree uygulaması ile web üzerinde nokta bulutlarını görselleştirmek için öncelikle Potree Converter açık kaynak kod kullarak LAZ dosyasını Potree uygulamasının kullanabileceği dosya formatına dönüştürülmesi gerekmektedir.

Bunun için *https://github.com/potree/PotreeConverter* adresinden Potree Converter açık kaynak kodlu PotreeConverter uygulaması indirilir. LAZ verisi bu dosyaların içine yerleştirilir. Uygulama dosyasının içinde "*PotreeConverter.exe Scan1.LAZ -o scan1 --generate-page scan1*" komutu cmd penceresinde çalıştırılır. Bu komut LAZ formatında bulunan veriyi okur ve bu veri için scan1 dosyasını oluşturur. Bu dosya içinde octree veri yapısına dönüşmüş datalar ve uygulamada çalıştırmak üzere gerekli javascript dosyaları bulunmaktadır (Schütz, 2014).

Elde edilen bu verileri Potree uygulamasında açmak için *https://github.com/potree/potree* adresinden açık kaynak kodlu Potree uygulaması indirilir. Bu klasör içinde cmd'ye *gulp webserver* komutu yazılarak dosyalar çalıştırılır. Bu komut http://localhost:8080 portu üzerinde görselleştirme çıktılarını sunar. Oluşturulan bu Scan1 dosyası bu dosyalardan examples'ın altına koyulur. Buradan LAZ dosyasının 3B görüntüsünü elde edilmiş olur (Schütz, Potree, 2014). Şekil 3.4' de nokta bulutu verilerinin potreeconverter kullanılarak potree uygulaması üzerinde görselleştirilmiş hali gösterilmektedir.

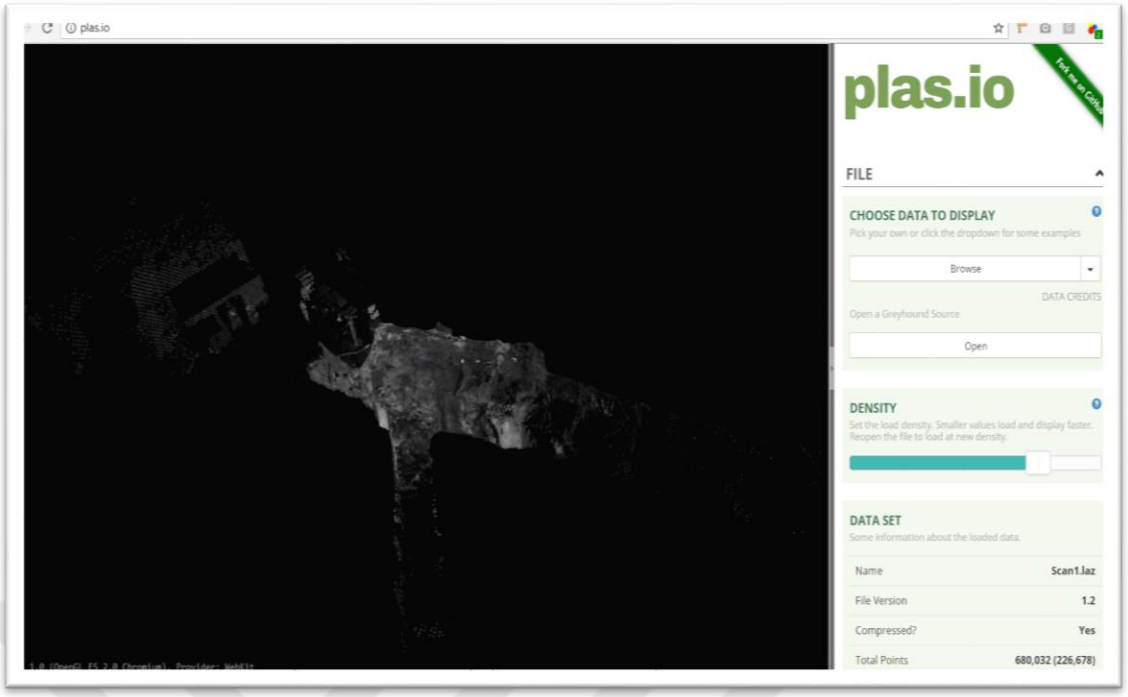


Şekil 3.4 Potree uygulaması kullanılarak 3B nokta bulutunun görselleştirilmesi

3.1.3 Plasio

Plasio, Uday Verma ve Howard Butler tarafından geliştirilen tarayıcıda nokta bulutu verilerini görselleştirme uygulamasıdır. Özellikle ASPRS, LAS formatındaki nokta bulutu veri yapısını destekler. Şu anda sadece Chrome' da çalışmaktadır. Bu uygulamayı kullanmak için hiç bir eklentiye ihtiyaç yoktur. LIDAR verileri doğrudan uygulamaya sürüklenip bırakıldığında görselleştirilmektedir. Açık kaynak kodlu bir uygulamadır. Bu uygulama ile noktaların yoğunluk eşiği ayarlanabilir. Sınıflandırma yaparken nokta bulutunun rengi değiştirilebilir.

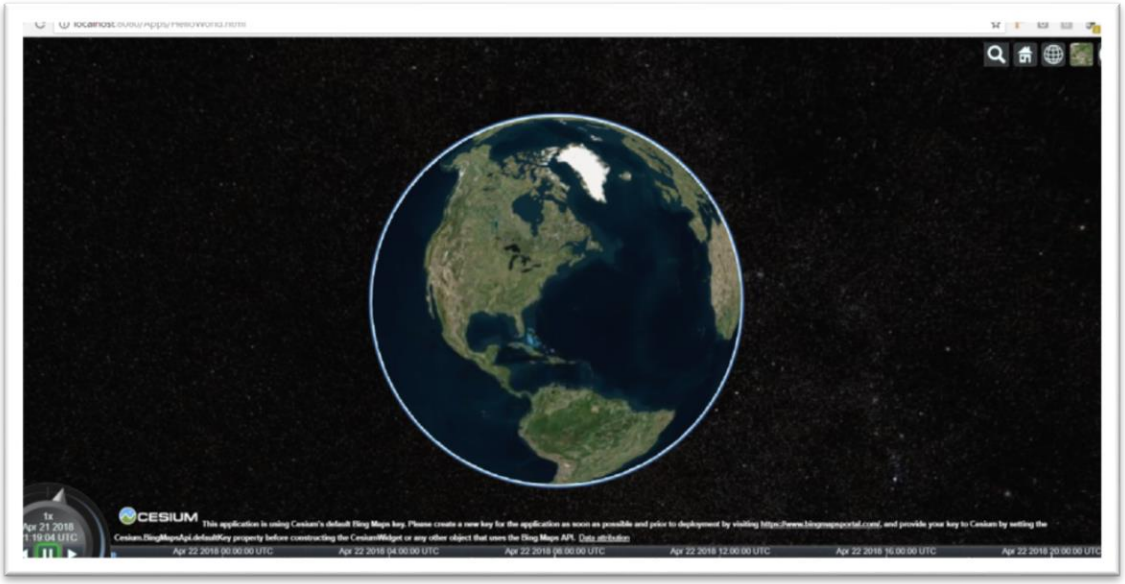
Bu uygulamayı çevrimiçi (online) olarak kullanabilirken açık kaynak kodlu olduğu için *https://github.com/verma/plasio* adresinden indirilip çalıştırılabilir. Bunun için node.js javascript kütüphanesi kullanılır. Uygulamanın bulunduğu klasörde cmd penceresine npm install gulp komutu yazılarak çalıştırılır ve gulp paketleri bilgisayara yüklenir. Sonrasında "*npm install --save-dev browserify-shim*" komutu çalıştırılır, gerekli paketler tekrar kurulur. Sonrasında gulp develop komutu çalıştırılır ve localhost://8080 portu ile tarayıcıda çalıştırılır (Verma, 2018). Şekil 3.5' de plasio uygulaması kullanılarak web ortamında nokta bulutu verisinin görselleştirilmiş hali bulunmaktadır.



Şekil 3.5 Plasio uygulaması kullanılarak 3B nokta bulutunun görselleştirilmesi

3.1.4 Cesium JS

WebGL ile geliştirme maliyetli ve karmaşıktır. Bu nedenle WebGL üzerine CBS ile ilgili özellikleri gerçekleyen bir kütüphane arayışı oluşmuş ve bu arayışın sonunda Cesium ortaya çıkmıştır. Cesium, 2B ve 3B haritalar yaratmak için JavaScript ve WebGL tabanlı, tarayıcıda eklenti gerektirmeyen bir kütüphanedir. Açık kaynaklı ve Apache 2.0 lisansına sahip olduğu için kullanılması ücretsizdir. Cesium, Patrick Cozzi liderliğinde bir ekip tarafından geliştirilmektedir [21]. Cesium, WebGL üzerine javaScript kütüphaneleri ile geliştirilen, web tarayıcı üzerinde nokta bulutlarını ve 3B modelleri yüksek performanslı olarak gösteren bir sanal küre uygulamasıdır. Cesium projesinin en önemli yönlerinden biri 3B katmanları (tile) desteklemesidir. Ayrıca stream ve render işlerini yüksek performanslı olarak yapmaktadır. Cesium arazisine (terrain) ve görüntü akışına kavramsal olarak benzeyen bir teknik kullanarak, 3B Tilelar, etkileşimli olarak görüntülenmesi imkansız olan binalar veri setleri, CAD modelleri ve nokta bulutları gibi devasa modelleri görüntülemeyi mümkün kılar. Cesium js açık kaynak kodlu uygulamadır. Uygulamayı çalıştırmak için node.js javascript kütüphanesi gerekmektedir. Uygulamanın bulunduğu dosyanın içinde cmd penceresinde npm install komutu ile node.js kurulur. Sonrasında node server.js komutu çalıştırılır ve http://localhost:8080 portu ile Cesium.js çalıştırılır (Hlushko, 2018). Şekil 3.6‘ Cesium js’ in sunduğu sanal küre görünmektedir.



Şekil 3.6 Cesium js' in sunduğu sanal küre

3.1.5 Wrld3D

Wrld3D, 3B haritalar, yüksek kaliteli coğrafi veriler kullanılarak oluşturulur, böylece 3B görselleştirmeler oluşturabilir, simülasyonlar çalıştırabilir ve dinamik konum tabanlı deneyimler ve oyunlar geliştirilebilir. Geliştiriciler için sanal dünyaları, akıllı binaları ve daha fazlasını hızlı bir şekilde oluşturması için güçlü dijital harita oluşturma ortamı sunar. Açık kaynak kod sunar ve istenilen şekilde gelişime açıktır (Jenks, 2018).

Wrld3D kullanarak 3B görselleştirme yapmak için Wrld3D web sitesinden ücretsiz kayıt olunması gerekmektedir. Kayıt olduktan sonra web ve mobil ortamda geliştirme yapmak için API-KEY oluşturulmaktadır. Wrld3D ile 3B uygulamalar geliştirilebileceği gibi bina içi (indoor) görselleştirmede de kullanılabilir. Wrld.js API'si web sayfasında 3B haritaları görselleştirmek için kullanılır ve Leaflet.js tabanlıdır. Standart leaflet.js fonksiyonları normal olarak kullanılabilirken 3B alanda çalışılabilmesi için L.Wrld.map fonksiyonu ile tarayıcılarda 3B interaktif haritalar basit bir şekilde üretilebilir. Wrld3D API'sini kullanarak basit bir 3B harita üretmek için visual studio ortamında bir proje geliştirildi ve leaflet.js ve wrld.js javascript kütüphaneleri kullanıldı. 3B haritayı tarayıcıda göstermek için wrld3d sitesinden alınan API-KEY şekil.16' da ki gibi koda eklendi, gerekli HTML elementi ve script tag'i kullanarak 3B görselleştirme için basit bir uygulama geliştirildi. Bu uygulama wrld.js'nin desteklediği tüm arayüz özelliklerini kullanabilir ve interaktif 3B haritalar üretilebilir (Jenks, 2018). Şekil 3.7' de Wrld3D javascrip kütüphanesi kullanılarak yazılan yavascript kodu görünmektedir.

```
<div id="map" style="height: 550px;width: 100%"></div>
<script src="https://cdn-webgl.wrld3d.com/wrldjs/dist/latest/wrld.js"></script>
<script>
    L.Wrld.map("map", "5ee57a6238f9c677d802db5dc04f1cbf");
</script>
```

Şekil 3.7 Wrld3D JS javascript Kodu (Jenks, 2018)

Şekil 3.8 Wrld js kullanılarak 3B görselleştirme

Şekil 3.8'de Wrld3D javascript kütüphanesi kullanılarak web ortamında 3B harita harita görselleştirilmesi için oluşturulan uygulama gösterilmektedir.

Wrld3D, ESRI Shapefiles ve GML gibi vektor verileri ve GeoTIFF gibi raster veriler dahil olmak üzere çeşitli veri formatlarını desteklemektedir. Veri iş akışı esnektir bu nedenle formatlar giriş için hızla desteklenir. Yüksek önem taşıyan alanlarda yani detaya ihtiyaç olunan haritalar, yüksek kaliteli görsel ipuçları sağlamak için elle düzenlenmesi gerekmektedir.

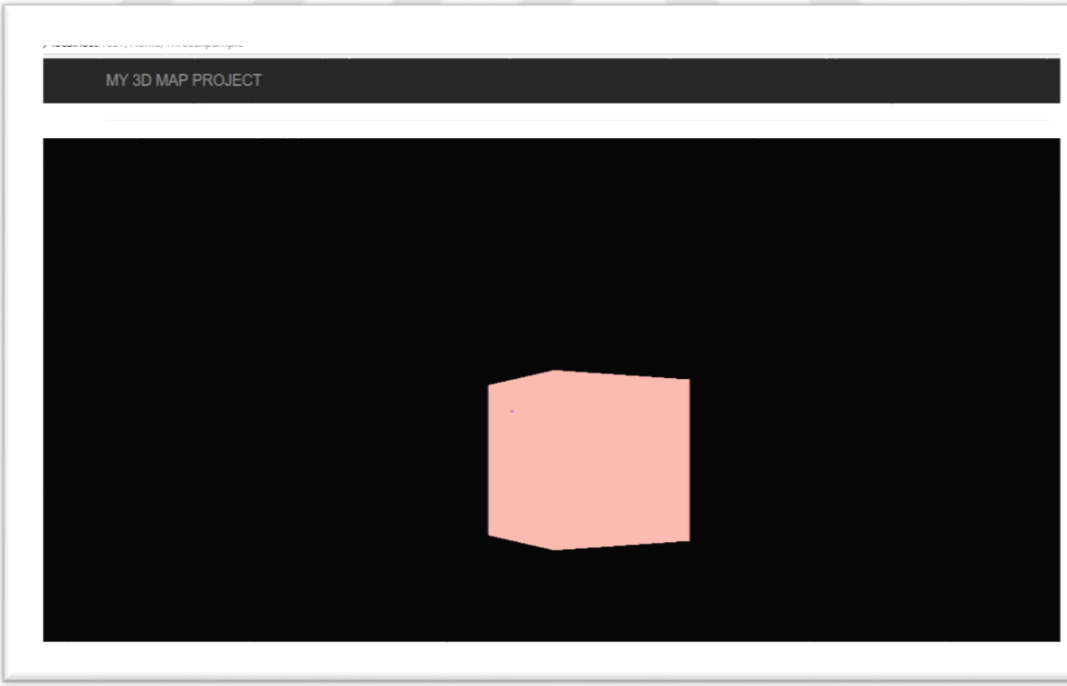
3.1.6 Three JS

Threejs WebGL tabanlı 3B model oluşturma kütüphanesidir. Bir eklentiye ihtiyaç duymadan tarayıcıda görüntülenen karmaşık 3B bilgisayar animasyonları geliştirmeye

olanak sağlar. Threejs karmaşık 3B görüntüleri sadece bir kaç satır kod ile kolayca oluşturabilir ve web sayfasına gömebilmektedir. Açık kaynak kodludur ve DOM (Document Object Model) ile entegre çalışır. WebGl'in bir çok detayından soyutlayarak hızlı ve kolay uygulama geliştirmeye olanak sağlar. Three.js ile uygulama geliştirmek için http://threejs.org/ adresinden açık kaynak kod kullanılır. "var scene = yeni THREE.Scene ();" komutu ile bir web ekranı oluşturulur ve bu ekran, geometrilerin, ışıkların, kameraların yerleşeceği konteynır gibidir. Web sayfasında gösterilmek istenen geometriler bu ekrana eklenir. "var camera = new THREE.PerspectiveCamera (45, window.innerWidth / window.innerHeight, 0.1, 1000);" komutu ile kamera kurulur, kamera görüntünün görüntülendiği göz anlamına gelmektedir. Konumlandırmanın ardından, kameranın sahnenin o kısmına nerede çekildiğini gösterir. Kameranın özellikleri: FOV - görüş alanı, kamera tarafından görülebilen ve ekranda görüntülenen alan. Aspect –Yatay görüş alanının dikey görüş alanına oranı. Near – Kameranın sahne nesnelerini oluşturmaya başlayacağı mesafedir. Far – Bu mesafeden sonra yerleştirilen nesneler oluşturulmayacaktır. Kamerayı oluşturduktan sonra doğru görüntüyü yakalamak için konumlandırılması gerekmektedir."*camera.position.set (x, y, z);"* komutu ile konumlandırılır. Ekran ve kamera hazır olduğunda sayfaya yerleştirilir. Bu görev WebGLRenderer tarafından yapılır. Render oluşturduktan sonra DOM (Document Object Model)'a body elementi aracılığı ile eklenir. Bu Three.js'nin ekran oluşturmak için kullanılacak body elementinin içinde bir canvas elementi oluşturmasını sağlar. Sonrasında ekranda görünecek geometri oluşturulur (Mr.doob, 2018). Şekil 3.9' da Threejs javascript kütüphanesi kullanılarak, web ekranında dönen küp elde etmek için yazılan javascript kodları gösterilmektedir. Şekil 3.10' da bu kodların sonucunda web ekranında oluşan dönen küp gösterilmektedir.

```
<body>

</body>
<script src="~/Scripts/three.min.js"></script>
<script>
    var scene = new THREE.Scene();
    var camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 0.1, 1000);
    camera.position.set(0,0,5);
    var renderer = new THREE.WebGLRenderer();
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.body.appendChild(renderer.domElement);
    var geometry = new THREE.BoxGeometry(1, 1, 1);
    var material = new THREE.MeshBasicMaterial({ color: 0xf9b8ae });
    var cube = new THREE.Mesh(geometry, material);
    scene.add(cube);
    function animate() {
        requestAnimationFrame(animate);
        cube.rotation.y += 0.01;
        renderer.render(scene, camera);
    }
    animate();

```

Şekil 3.9 Web ekranında küp oluşturmak için yazılan javascript kodları



Şekil 3.10 Threejs ile web ekranında dönen küp

4. NOKTA BULUTU VERİLERİNİN MODELLENMESİ

İnsanların ve robotların (yapay zeka) 3B nokta verisi ile etkileşime geçebilmeleri için farklı görselleştirme teknikleri ortaya çıkmaktadır. Nokta bulutları web üzerinde gerçek zamanlı olarak iki biçimde render edilebilir. Birincisi nokta bulutu nokta olarak gösterilir. İkincisi ise nokta bulutu mesh olarak render edilebilir. Her iki yöntemde de çoklu çözünürlüklü veri yapısı kullanılmaktadır. Nokta bulutu verisi ile dokuyla kaplı otomatik 3B model oluşturma önemli bir çalışma konusudur. Ancak model büyüdükçe meshler arasındaki bağlantıları yönetmekte güçleşmektedir. Nokta bulutu verisi ile otomatik 3B model oluşturma işlemi geometri temelli modelleme ve yüzey ağı modelleme (meshing) olmak üzere iki farklı yaklaşımla gerçekleştirilebilir.

4.1 Nokta Olarak Gösterim

Geometri temelli modelleme yöntemi genel olarak verilerin düzenlenmesi, bölütleme (segmentation) aşaması, yüzey yakalama ve 3B model oluşturma aşamalarından oluşmaktadır. Nokta bulutunun bölütleme işleminde nokta bulutu içerisindeki noktalar sahip oldukları geometrik özelliklere bağlı olarak gruplandırılıp sınıflandırılabilmektedir (Ponchio, 2008).

4.2 Mesh Olarak Gösterim

3B noktalar genellikle düzensizdir ve bunları 3B uygulamalarda kullanmak, örneklenen yüzeye en iyi yaklaşan çokgen (triangular) bir ağın hesaplanmasını gerektirmektedir. Bu bağlantı yapısının nokta kümesiyle ilişkilendirilmesi anlamına gelmektedir. Mesh, genellikle bir yüzeyin parçalı doğrusal bir uyarımı temsil eden poligonal yüzler tarafından bağlanmış köşelerin toplanması olarak tanımlanır. 3B noka bulutlarını birbirine bağlamak (meshing) için bir çok algoritma kullanılır. Algoritmalar karmaşıklık, sağlamlık, tekrarlanabilirlik açısından farklılık gösterir.

4.3 Mesh için Kullanılan Yöntemler

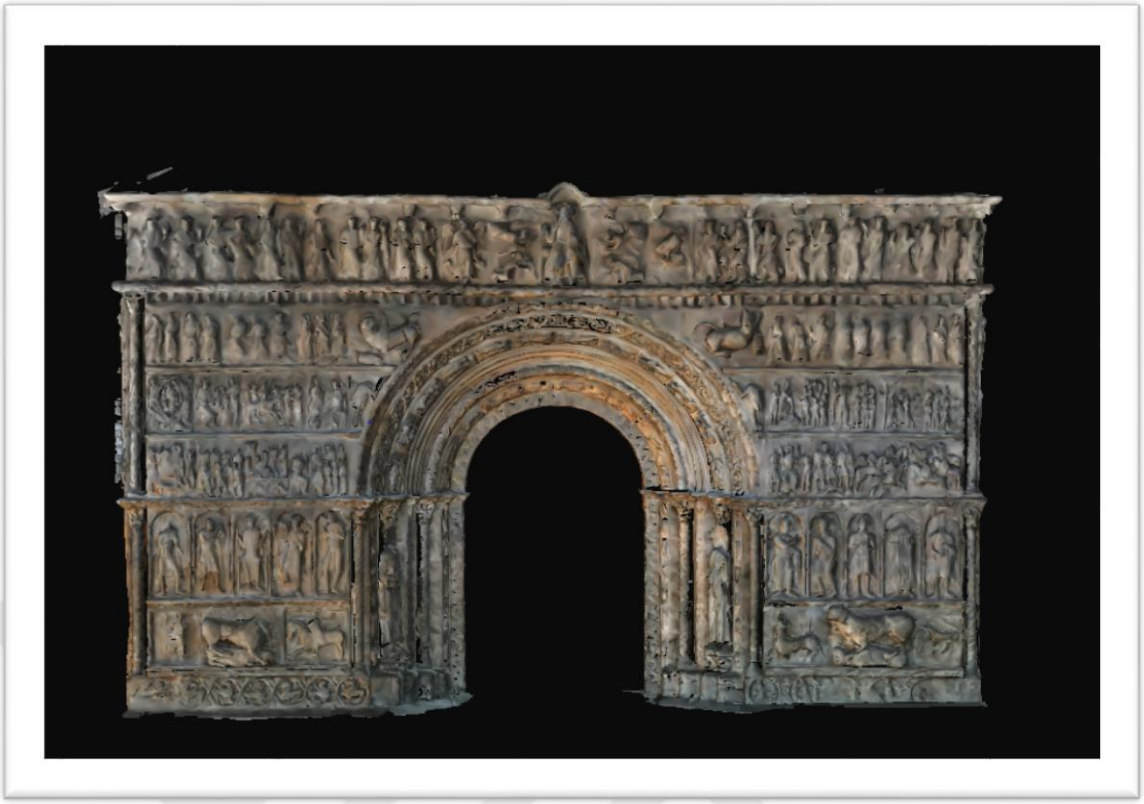
4.3.1 NexusJS

Nexusjs, OpenGL ’de büyük 3B modellerin görselleştirilmesi ve çok bileşenli mesh yapısının oluşturulması için kullanılan javascript kütüphanesidir. Çoklu çözünürlüklü

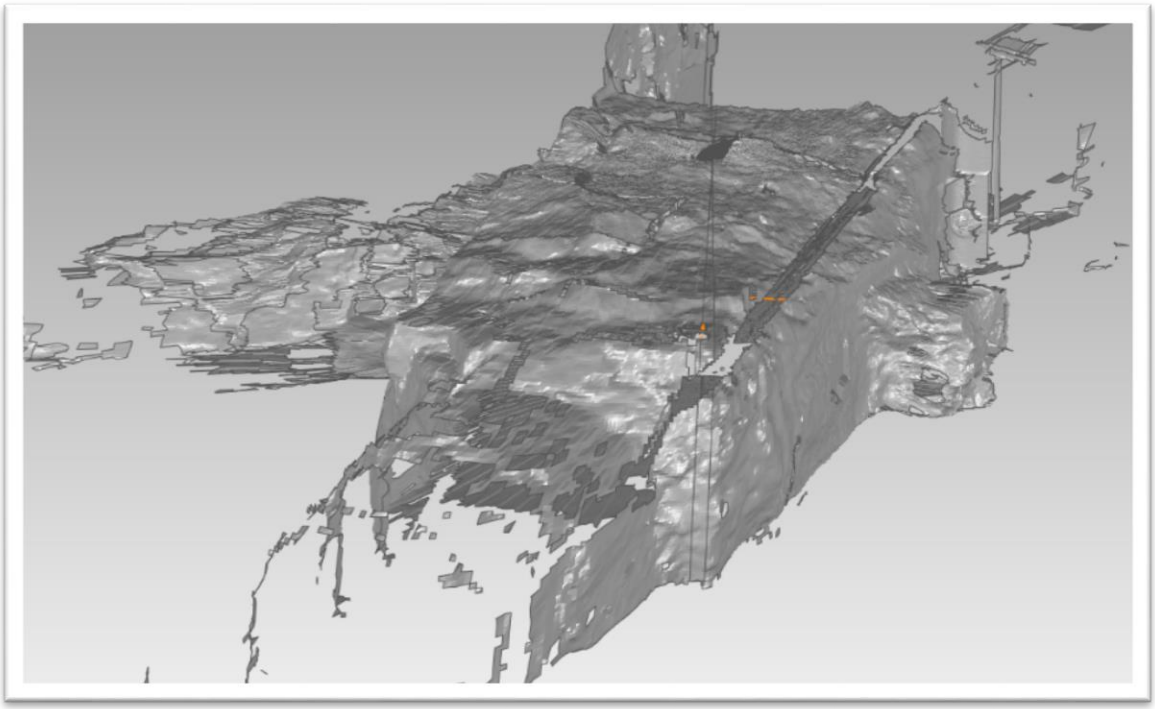
bir yapı sunmaktadır. 3B modeller çözünürlüğe izin veren bir veri yapısına dönüştürülür. Çok büyük veri yapılarını desteklemektedir. Nexus nokta bulutları ve mesh'leri eğim özelliklerine göre destekler. Nexus, toplu bir çok bileşenli örgü yapısının oluşturulması ve görselleştirilmesi için bir yazılım paketi sağlamaktadır. Bu tür veri yapısının ana amacı, büyük üçgen meshleri veya nokta bulutlarının verimli bir şekilde çoğaltılmasını sağlamaktır. Bu kütüphanenin ana özellikleri şunlardır: çekirdek dışı etkileşimli görselleştirme, modellerin çoklu görünümü, http akışı, sıkıştırma, köşe başına renk, opensg Nexus düğümü. Bu çok işlevli kütüphane, WebGL'de geometriyi uygun bir çözünürlükle verilen nokta görünümünde çizmek için tasarlanmıştır (Horman, 2018). Şekil 4.1' de nexusjs kullanılarak ekranda 3B görüntü elde etmek için kullanılan javascript kodu gösterilmektedir. Şekil 4.1' de bu kod kullanılarak oluşturulmuş 3B görüntü gösterilmektedir. Şekil 4.3' de de bir galerinin GIS ortamında 3B mesh modeli gösterilmektedir.

```
var nexus = new Nexus.Instance(gl);  //webgl context is needed
nexus.open(url);
nexus.onLoad = function() {
        var mesh = nexus.mesh;
        //center and scale for the model
        var scale = 1/mesh.sphere.radius;
        var position = mesh.sphere.center;
        //information about components of the nexus are accessible
        var hasNormals = mesh.vertex.normal;
        var hasColors = mesh.vertex.color;
        var hasUv = mesh.vertex.texCoord;
        var hasFaces = mesh.face.index;

        nexus.onUpdate();
};
//when additional geometry is avalable call your rendering function
nexus.onUpdate = function() {
        render();

        //setup a program (for example in three.js)
        var program = renderer.context.getParameter(gl.CURRENT_PROGRAM);

        //tell nexus to which attribute location binds it's attribute
        nexus.attributes['position'] = renderer.context.getAttribLocation(program, "position");
        nexus.attributes['normal'] = renderer.context.getAttribLocation(program, "normal");
        nexus.attributes['color'] = renderer.context.getAttribLocation(program, "color");
        nexus.attributes['uv'] = renderer.context.getAttribLocation(program, "uv");

        nexus.updateView([0, 0, width, height], projectionMatrix, modelViewMatrix);
        nexus.render;
}
```

Şekil 4.2 Nexus js kullanılarak ekranda 3B görüntü elde etmek için kullanılan javascript kodu



Şekil 4.3 Nexus js kullanılarak elde edilen 3B görüntü



Şekil 4.4 Bir galerinin GIS ortamında 3B mesh modeli

5. NOKTA BULUTU VERİLERİ İLE SANAL GERÇEKLİK

3B modeller ile sanal gerçeklik ilgilenilen alanları anlamak için çok kullanışlı bir yöntemdir. Kentsel 3B modelleme, eğlence ve ticari uygulamalarda, google earth ve canlı arama haritaları gibi coğrafi görüntü kütüphaneleri ile oluşturulan uygulamalar ile popüler olarak kullanılmaktadır. 3B modeller, şehir planlaması ve simülasyonu, keşif verilerinin yorumlanması ve gerçek zamanlı acil durum müdahalesi gibi uygulamalar için öneme sahiptir. Sanal gerçeklik ile ilgili olarak, günümüzde elle sanal ortamlar geliştirmek yerine, gerçek ortamlar modellenir. Örneğin, evlerin 3B modelleri, emlak piyasası için sanal yürüyüşlere izin verebilir. Müzeler ve tarihi siteler eğitici ve eğlenceli olacak sanal turlar oluşturabilir (Zhihan, 2011).

Nokta bulutu verilerinden sanal gerçeklik modeli oluşturmak için bu veriler genellikle üçgen şeklindeki yüzeylere dönüştürülür ve dünya yüzeyinin sanal gerçeklik modellerini oluşturmak için yüksek çözünürlüklü dijital fotografların örtülmesi için kullanılır. Modelin kalitesi ve doğruluğu için üçgenleme adımı kritiktir. Sanal gerçeklik modelinin kalitesi noktaların yoğunluğuna ve üçgenlemede kullanılan tarama açısına bağlıdır. LIDAR taramalarında ortaya çıkan sonuç yoğunluğu optimaldir ve her türlü üçgen yüzey, alım gölgelerinden ve minumum uzun üçgenlerden kaynaklanan minumum deliklere sahiptir.

Fotograflar ve LIDAR taramalar, standart bir web görüntüleyicide görüntülenebilen bir sanal gerçeklik modeli oluşturmak için birleştirilir. Bu adım üçgenleme işlemi için önemlidir, çünkü yeniden düzenleme çekim yönüne denk olan bu projeksiyona bağlıdır. Düzensiz nokta bulutu daha sonra ağırlıklı ters mesafe algoritması kullanılarak oluşturulan bu sözde ızgara(pseudo-2D) üzerine yeniden regüle edilir ve üçgen ağ (mesh) oluşturulur. Bu üçgensel yüzey çeşitli yazılımlar kullanarak (Qslim) optimize edilir, üçgen sayısı azaltılır. Son olarak, bu çıktılar 3B özellikli bir web tarayıcıda görüntülenmek için standart bir sanal gerçeklik modelleme dili (VRML) ‘ye dönüştürülür. (Zhihan, 2011)

5.1 WebVR

WebVR ilk olarak 2014 yılında Mozilla çalışanlarından Vladimir tarafından tasarlanmıştır. Mart 2016’da Mozilla VR ekibi ve Google Crome ekibi, WebVR API için 1.0 sürümünü yayınlamıştır. WebVR API’si, WebGL’yi gerekli kamera ayarları ve cihaz etkileşimleri (kontroller ve bakış açısı) ile kullarak web uygulamalarının sanal gerçeklikte içerik sunmasına izin veren arayüzler (VR Ekranı, VR pozu) sağlamaktadır. WebVR uygulamaları veya oyunları geliştirmek için özel yazılım geliştirme kitine (SDK) ihtiyaç duymamaktadır. HTML5, CSS3 ve JavaScript

kullanarak simüle edilmiş ortamlar oluşturulmasına olanak sağlamaktadır. API, coğrafi konum API'si gibi müdahaleci Web API'sine benzeyen belirli bir yolu takip edecek şekilde tasarlanmıştır. Gerekli adımlar aşağıdaki gibidir.

Kullanılabilir VR cihazları listelenir. İstenilen cihazın uygulamanın ihtiyaç duyduğu sunum modlarını destekleyip desteklemediği kontrol edilir. Destekliyorsa, uygulama kullanıcıya VR işlevini bildirir. Kullanıcı, VR moduna girmek istediğini belirten bir eylem gerçekleştirir. VR içeriğini sunmak için bir VR oturumu istenir. VR cihazında görüntülenecek grafik kareler üreten bir işlem döngüsü başlar. Kullanıcı, VR modundan çıkmak istediğini belirtene kadar kareler üretmeye devam edilir. VR oturumu sonlanır (Gilbertson, 2018).

Sanal gerçeklik (VR) teknolojisi, gerçek dünyanın bilgisayar ile 3B bir sanal alan oluşturması ve üretmesi için kullanılır. WebVR, 3B modelleri doğrudan tarayıcıda göstermeye olanak sağlamaktadır. Sanal ortam genellikle gerçek bir ortamın kopyasıdır ve 3B ayarlar (derinlik algısı gibi), sesler ve kullanıcıların kendisiyle etkileşim kurmasına izin veren konsollar gibi araçlar kullanılır. Bir kullanıcının hareketi ya kafa üstü aygıtı kullanarak ya da hareket algılama sensötleri kullanılarak izlenebilir. WebVR API, WebVR'yi sanal gerçeklik deneyimlerini oluşturmak, bunlara erişmek ve paylaşmak, bağlantı linklerinin paylaşılması ile, her yerde erişilebilir hale getirmektedir. WebVR'nin en önemli özelliklerinden biri herhangi bir cihazdan neredeyse anında erişilebilir bir deneyim oluşturmasıdır (Gilbertson, 2018).

5.2 Google Tango

Google Tango en temel biçimde, cihazlara çevremizi nasıl yaptığımıza benzer bir şekilde görme ve anlama yeteneği vermektedir. Bu çeşitli bilgi kaynaklarını birbirine bağlayabilen ve hepsini anlayabilen özel bir sensör seti ve işlemci oluşturmak anlamına gelmektedir. Tango teknolojisinde üç ana bölüm vardır. İlk olarak hareket izleme teknolojisi. Bir hareket izleme kamerası, 3B derinlik sensörü, ivme ölçer, barometre, jiroskop ve GPS kullanarak, Tango uyumlu bir cihaz nerede olduğunu ve belirli bir alan veya alanda nasıl hareket ettiğini ve hangi yöne baktığını anlatabilir. Ardından bu teknoloji ikinci ve üçüncü özellikler olan derinlik algısı ve alan öğrenimi ile birleştirilir. Tango, hareket edilen alanı büyük bir hassasiyetle anlayabilir. Yani, dar bir koridordan geçerken ve bir köşeyi dönerken, gelinen yer ve nereye gidildiği izlenebilir. Duvarın nerede olduğunu ve bilirli nesnelere ne kadar yakın olunduğu söylenebilir. Derinlik algısı içinde belkide daha etkileyici olan, bir nesnenin küçük ve yakın ya da uzak ve büyük olup olmadığını anlayabilmesidir. Nesne ile nesnenin arasındaki mesafe, nesnenin boyutu ve bölgedeki diğer ögelere göre nerede durduğu

hesaplanması gerekmektedir. Böyle işlemler için Tango'nun özellikleri yeterli değildir (Betters & Bunton, 2017).

Bir tabloyu bir uç ile diğeri arasında birleştirme çizgisi çizmek için artırılmış gerçeklik kullanılarak Google Tango teknolojisi ile ölçülebilmektedir. Bu işlemi yaparken hareket izleme ve alan öğrenme teknolojisi sayesinde cihazı sabit tutmaya gerek yoktur. Bir nesne ne kadar uzakta olursa olsun veya ölçüm sırsında hareket edilse bile fark edilir ve ölçülür. Yüzeyler algılanabildiği ve açıları ölçülebildiği için, sanal nesneleri yerleştirmenin nasıl ve hangi açıyla olacağıda bilinir. Tango iç mekanların doğru bir şekilde haritalanması için kullanılmaktadır ve potansiyel olarak Google Haritalar'ın bilgisini binaların içine ulaştıracak şekilde genişletilmeye devam etmektedir (Betters & Bunton, 2017).

6. SİSTEM MİMARİSİ

Sistem arka tarafta bir veri tabanından, istemci tarafında web tabanlı bir API'den ve orta katmanda istemci ve veritabanı arasındaki ilişkiyi kuran bir web servisinden oluşmaktadır. Nokta bulutu eğer klasik ilişkisel veritabanı yaklaşımında tutulmak istenirse tablonun her bir satırı bir noktaya ait olup tablonun üç sütunu noktanın koordinat değerlerini (X,Y,Z), bir sütunu intensity değerini ve eğer varsa üç sütunu da renk değerlerini (R,G,B) gösterecektir. Her ne kadar klasik ilişkisel veritabanı yaklaşımı milyonlarca satır noktayı depolayabilse de performans düşüklüğünden ve zaman zaman projeye bağlı olarak nokta sayısının birkaç milyarı bulmasından dolayı nokta bulutları için klasik ilişkisel veritabanı yaklaşımı kullanılmamaktadır. Ama yine de Oracle/OracleSpatial ve Postgres/PostGIS tarafında nokta bulutundaki her bir noktayı bir satırdaki bir nesne olarak kabul eden çalışmalar bulunmaktadır. PostGIS tarafındaki yaklaşım Point Cloud Patch (PcPatch) olup en fazla 600 noktalı pathcler oluşturup depolamaktadır. Hem görece az sayıda noktayı patch içinde tutması ve gerçek geometriye ulaşmak gerektiğinde patchleri açarak klasik GIS noktasına dönüştürmesi ek bir işlem gerektirdiğinden uygulamalarda rağbet görmemektedir. Nokta bulutu veri seti genellikle dosya tabanlı (native file system) olarak depolanır. Nokta bulutu verisindeki nokta sayısı arttıkça dosyanın da boyutu artmaktadır. Tiling denilen işlemle tek bir dosya farklı mekansal alanları gösteren birçok dosyaya ayrılır. Bir dosya da sözü edilen tileların metaverisini tutmaktadır. Doküman yönelimli NoSQL veritabanı yaklamışı ile her bir tile (dosya) bir doküman olarak depolanabilir. MongoDB, couchbase, clusterpoint gibi NoSQL veritabanları bu iş için kullanılabilir ve tileların sınırlayan kutuları (bounding boxes) mekansal olarak sorgulanabilir. Bu doküman (ya da tileların sınırlayıcı kutuları) örneğin MongoDB'de JSON gösterimine benzeyen depolama için optimize edilmiş BSON (Binary JSON) nesnesi ile gösterilir.

Ayrıca petabytes düzeyindeki verileri dağıtık veritabanı sistemi biçiminde tutabilen ve veritabanındaki satır sayısının sınırsız olduğu Google's Bigtable'da nokta bulutlarının depolanmasında kullanılabilir. Google's Bigtable ve benzeri veritabanı yaklaşımlarında octree yerine diğer bir mekansal veri yapısı olan megatree kullanılmaktadır. Bu veri yapısı milyarlarca noktayı depolayabilecek yapıdadır (Boehm, 2016).

7. GEZİNGE PLANLAMASI

Gezinge planlaması (Pathfinding), bir başlangıç ve bir hedef noktası arasında optimal bir yol hesaplamaktır. Tamamen 3B ortamlarda uygun yollar planlamak oldukça zor bir sorundur. Mobil robotik bağlamında navigasyonun görevi rahat bir şekilde bir rota veya bir yörüngenin gidişatını belirleme ve sürdürme süreci olarak tanımlanabilir. Navigasyon için hakim olan çerçeve dört bileşen üzerine inşa edilmiştir. Yerelleştirme, haritalama, hareket/ yol planlaması ve takip yolu. Eş zamanlı lokalizasyon ve haritalama (SLAM) algoritmalarının altı serbestlik derecesinin ilerlemesiyle, navigasyon sırasında doğru üç boyutlu haritaların çevrimiçi hale gelmesi sağlanmaktadır. Böylece 3B ortamlarda planlama yapmak daha uygulanabilir ve giderek daha önemli bir görev haline geliyor.

7.1 Octomap ile Gezinge Planlaması

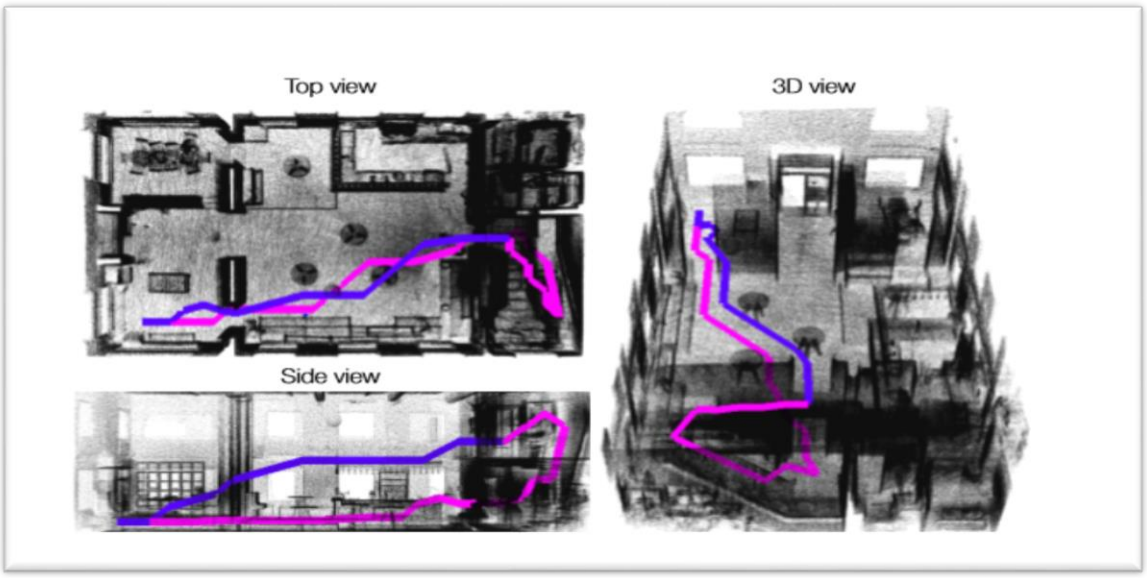
3B yol bulma örneklerinin ortak noktası, belirli bir geometriye sahip bir nesnenin, bir başlangıç ve hedef noktası arasında en uygun çarpışma serbest yolunu bulması gerektiğidir. Bunu yönetmek için nesnenin geometrisini ve çevrenin bir modelinin bilinmesi gerekmektedir. Ya da nokta bulutu kullanılması gerekir. Ancak, tek başına nokta bulutu ortamda rota bulmak için yeterli bilgi vermez. Bunun için boş (pointless) alan gereklidir. Boş alan bir nokta bulutunu bölümlere ayırarak işgal edilen alanlardan üretilmektedir. Octree veri yapısı alanın verimli bir şekilde yapılandırılmasına olanak sağlamaktadır. Büyük boş alan octree'deki büyük bir düğümle temsil edilebilir. Bu büyük boş düğümler otree'deki düğüm sayısını azaltır. Bu yol bulma octree'nin avantajlı bir özelliğidir. Büyük düğümler, düğüm sayısını ve daha sonra yol bulmada keşfetme olasılıklarını azaltır. (Botea, Müller, & Schaeffer , 2007)

7.2 Octree Veri Yapısında Gezinge Planlama Yöntemleri

Yol bulma, başlangıç ve hedef düğüm arasındaki en uygun yolun hesaplanmasıdır. En uygun yol bunun en kısa olması gerektiği anlamına gelmemektedir. Bir çok parametre en uygun terimini tanımlayabilir. Bunun için çeşitli algoritmalar Dijkstra algoritması, A* algortiması gibi kullanılır. Dijkstra algoritması, minumum seyahat maliyetini arayarak bir başlangıç ve bir hedef düğüm arasındaki en uygun yolu hesaplar. Algoritma, hareketin tüm bitişik düğümlere göre hesaplandığı ve açık bir listeye yerleştirildiği bir başlangıç düğümü ile başlar. En düşük maliyete sahip olan düğüm işaretlenmiştir. A* algoritması, Dijkstra algoritmasının bir uzantısıdır. A* algoritması, işaretli düğümünden, hedef düğüme olan maliyetin yaklaşık bir yaklaşımı olan deneyimsel (heuristic) bir maliyet getirmektedir. Deneyimsel maliyet nedeniyle,

başlangıç düğümü ve hedef düğümü arasında olmayan tüm düğümler, olmayan düğümlere kıyasla daha yüksek bir deneyimsel mailyete sahiptir. Bu nedenle algoritmanın hızı ağ boyutuna bağlı değildir, güzergahın uzunluğuna bağlıdır (Rodenberg, 2016).

Herman [1986] octree gösteriminde iki yol bulma yönteminin kombinasyonunu sunmaktadır. A* algoritmasıyle birlikte bir tepe tırmanma (hill climbing) yöntemini birleştirir. Tepe tırmanışı bir yolu hızlıca hesaplayabilir, çünkü bir sonraki düğüme karar vermek için yalnızca bitişik düğümlere olan mesafeleri kullanır. Bunula birlikte, bu yöntem "U" şekilli engellere takılır. Böyle durumda engelden kaçınılıncaya kadar bir A* algoritması kullanılır. Bu yöntem hızlıdır, ancak rotadan uzaklaşma eğilimi vardır. Voros [2001] quadtree ve octree gösterimde bir yolu hesaplamak için bir mesafe haritası ile birlikte tepe tırmanma yöntemini kullanır. Voksel yaklaşımı yerine octree gösterimi kullanarak bellek talebi ve yol bulma süresi azalır. Uzak haritanın dezavantajı , her bir farklı hedef düğüm için belirli bir mesafe haritasın ihtiyaç bulunmasıdır. Xu [2015] bir octree gösteriminde bir A* yol bulma yöntemini açıklamıştır. Mevcut düğümün komşuları, hangi düğümlerin iki oktanın uzunluğunun toplamının yarısını aşmayan bir mesafeye sahip olduğunu kontrol ederek bulunur. Bu hesaplamalı pahalı bir işlemdir. Broersen [2016] bir A* algoritmasına dayalı basit bir yol bulma algoritması oluşturmuştur. Rota, octree'deki boş alan üzerinden hesaplanır**.** Yol bulma yöntemi, ortak bir yüzü paylaşıyorlarsa iki düğümün komşular olarak kabul etmeye dayanır. Komşular küçük, büyük ve eşit olabilir. Şekil 4.4' de iç alanda pathfinding gösterilmektedir (Botea, Müller, & Schaeffer , 2007).



Şekil 4.5 İç alanda pathfinding (Rodenberg, 2016)

8. SONUÇLAR

Tez kapsamında nokta bulutu verilerinin Octomap, Potree, Plasio gibi uygulamalar kullanarak web üzerinde farklı yöntemlerle görselleştirilebileceği görülmüştür. Octomap ve Potree, octree veri yapısını desteklerken, plasio uyglaması direk XYZ formatındaki veriyi görselleştirebilmektedir. Bunun yanında octree veri formatı görselleştirme yapmak için oldukça uygun veri yapısı olduğundan bu uygulamalar web üzerinde daha hızlı render edilebilir. İhtiyaca, veri yoğunluğuna bağlı olarak görselleştirme için uygun uygulamalar kullanılabilir. Bu uygulamalar nokta bulutlarının standart desktop uygulamalar dışında da görselleştirilebileceğini göstermektedir. Bu sayede tek bir bilgisayara bağlı kalınmadan browser üzerinden tüm bilgisayarlarda nokta bulutlarının görselleştirilmesi sağlanabilmektedir. Ancak nokta bulutundaki nokta sayısı çok fazla olduğunda web üzerinde görselleştirme etkin olarak yapılamamaktadır. Çok fazla nokta olması büyük verinin işlenmesi, yönetilmesi gibi sorunlarını da beraberinde getirmektedir. Örneğin Potree çözümünün kullanımı birkaç milyar nokta ile sınırlıdır. Bu nedenle büyük nokta bulutlarının (*massive point clouds*) görselleştirilmesinde farklı yaklaşımlarında kullanılması gerekmektedir.

Nokta bulutlarının depolanması ve görselleştirilmesine ilişkin standart geliştirme çalışmaları başlamış olsa da uygulamada kabul gören bir yaklaşım henüz geliştirilememiştir. Halihazırda yürütülen standart geliştirme çalışmalarına OGC (*Open Geospatial Consortium*) ve OSGEO verilebilir. OGC nokta bulutu verilerinin işlenmesinde ve paylaşılmasında bir nokta bulutu çalışma grubu oluşturmuştur (Point Cloud Domain Working Group (DWG)). ISO TC211 (*ISO technical committee for Geographic information/Geomatics*) OGC ile bu konular üzerine işbirliği geliştirme üzerine çalışmaktadır. OSGEO PointDown initiative ise web üzerinden nokta bulutu verilerinin servis edilmesine ilişkin çalışmalar yürütmektedir.

Gelecek çalışmalar için nokta bulutlarının etkin görselleştirilmesinde GPU yeteneklerinin daha etkin kullanılmasına çalışılacak ve VR/AR uygulaması ile bütünleştirilerek gerçeklik hissi arttırılmaya çalışılacaktır.

9. KAYNAKÇA

Altuntaş, C., & Yıldız, F. (2008). Yersel lazer Tarayıcı Ölçme Prensibleri ve Nokta Bulutlarının Birleştirilmesi. *Hkm*.

Amirian, Pouria, & Anahid, A. (2013). NoSQL storage and management of geospatial data with emphasis on serving geospatial data using standard geospatial web services.

Babahajiani, P., fan, L., Kämäräinen2 , J.-K., & Gabbouj2, M. (2017). Urban 3D segmentation and modelling from street view images and LiDAR point clouds. *Machine Vision and Applications* .

Bell, A., Chambers, B., Butler, H., & Gerlek, M. (2018). *PDAL - Point Data Abstraction Library.* PDAL. adresinden alındı

Betters, E., & Bunton, C. (2017). Google Tango explained.

Boehm, J. (2016). File-centric Organization of large LiDAR Point Clouds in a Big .

Botea, A., Müller, M., & Schaeffer , J. (2007). Near Optimal Hierarchical Path-Finding.

Civelekoğlu, B. (2015). Hava lidar verilerinin sınıflandırılması ve orman ağaçlarına ait özniteliklerin değerlendirilmesi istanbul belgrad ormanı örneği . *Ytü*.

Cozzi, P. (2016). *WebGL Insights.*

Gilbertson, D. (2018). *WebVR API.* Mozilla: https://developer.mozilla.org adresinden alındı

Goldberg, K. (2014). *Tutorials.* Point Cloud Library. adresinden alındı

Goldberg, K. (2017). *Module kdtree.* Point Cloud Library: http://www.pointclouds.org adresinden alındı

Greggman. (2018). *WebGL Fundamentals.* WebGL: https://webglfundamentals.org/ adresinden alındı

Hlushko, E. (2018). *Cesium js Getting Started.* Cesiumjs: https://cesiumjs.org/ adresinden alındı

Horman, K. (2018). *Nexus: adaptive 3D.* GitHup: http://vcg.isti.cnr.it/nexus/ adresinden alındı

Hornung, A., Wurm, K., Bennewitz, M., Stachniss , C., & Burgard, W. (2013). OctoMap: AnEfficientProbabilistic3DMappingFrameworkBasedonOctrees.

Jenks, T. (2018). *Wrld.js Documentation for 3D Maps Anywhere.* Wrld.js: https://www.wrld3d.com/ adresinden alındı

Luten, E. (2014). *OpenGLBook.com.* OpenGl: http://openglbook.com adresinden alındı

Mahtani, A., Sánchez , L., Fernández , E., & Martinez, A. (2016). *Effective Robotics Programming with ROS.*

Martinez, R. (2016). Taming the beast: Free and open-source massive point.

Mr.doob. (2018). *JavaScript 3D library.* GitHup: https://threejs.org/ adresinden alındı

Nevala, E. (2014). *Introduction to Octrees.* Gamedev.net: https://www.gamedev.net adresinden alındı

Ooi, B. C. (2017). Spatial kd-Tree: A Data Structure for Geographic Database. B. C. Ooi içinde, *Informatik-Fachberichte* . Point Cloud Library (PCL) . adresinden alındı

Ponchio, F. (2008). Multiresolution structures for interactive visualization of .

Ramsey, P. (2017). LIDAR in PostgreSQL with PointCloud.

Richter, R., Behrens , M., & Döllner, J. (2013). Object class segmentation of massive 3D point clouds of urban areas using point cloud topology. *International Journal of Remote Sensing*.

Rodenberg, O. (2016). The effect of a* pathfındıng characterıstıcs on the path length and performance ın an octree representatıon of an ındoor poınt cloud.

Rouault, E. (2018). *GDAL - Geospatial Data Abstraction Library.* GDAL. adresinden alındı

Schütz, M. (2014). *Potree.* GitHup: https://github.com/potree/potree adresinden alındı

Schütz, M. (2014). *Potree Converter.* GitHup: https://github.com/potree/PotreeConverter adresinden alındı

Sharma, D. J. (2002). Oracle Spatial. D. J. Sharma içinde, *Oracle.*

Verma, U. (2018). *Plasio.* GitHup: https://github.com/verma/plasio adresinden alındı

Vries, j. d. (2014). *learnopengl.com.* OpenGl. adresinden alındı

Yin, X. (2009). Quadtree Representation & Compression of Spatial Data.

Zhihan, L. (2011). WebVR - Web Virtual Reality Engine Based. China : Jounal of Networks.

ÖZGEÇMİŞ

Ad Soyad: ARİFE MUTLU

Doğum Yeri ve Tarihi: Zonguldak - 15.04.1991

Adres: Sarıyer - İstanbul

E-posta: arife.mutlu@outlook.com

Lisans Üniversite: Yıldız Teknik Üniversitesi – Harita Mühendisliği

Yüksel Lisans Üniversite: İstanbul Teknik Üniversitesi – Coğrafi Bilgi Teknolojileri

İş Tecrübesi: Bilge Adam Bilişim Grubu – Yazılım Uzmanı (2015- Devam)