T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ A.B.D.
BİLGİSAYAR MÜHENDİSLİĞİ

REST MVC UYGULAYARAK ŞEHİR NESNELERİNİN WebGL İLE GÖRSELLEŞTİRİLMESİ

Yüksek Lisans Tezi

Tezi Hazırlayan: **Seyfullah TIKIÇ**

İSTANBUL, 2014

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ A.B.D.
BİLGİSAYAR MÜHENDİSLİĞİ

REST MVC UYGULAYARAK ŞEHİR NESNELERİNİN WebGL İLE GÖRSELLEŞTİRİLMESİ

Yüksek Lisans Tezi

Tezi Hazırlayan:

Seyfullah TIKIÇ

Öğrenci No :

110820013

Danışman :

Yrd. Doç Ümit Işıkdağ

İSTANBUL, 2014

YEMİN METNİ

Yüksek lisans tezi olarak sunduğum "REST MVC uygulayarak şehir nesnelerinin WebGL ile görselleştirilmesi" başlıklı bu çalışmanın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını belirtir ve bunu onurumla doğrularım. 24/02/2014

Aday : Seyfullah Tıkız

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ SAVUNMA SINAVI SONUÇ TUTANAĞI

Beykent Üniversitesi
Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Aşağıda tez adı belirtilen yüksek lisans öğrencisi 110820113 no'lu Seyfullah Tuka'in 18/02/2014 tarihinde yapılan tez savunma sınavı[1] sonucunda 50 dakika süreyle sunduğu ve savunduğu tezi hakkında[2] oybirliğiyle, KABUL kararı verilmiştir.

Bilgilerinize saygılarımızla arz ederiz.

Anabilim Dalı : Bilgisayar Mühendisliği
Programı : Bilgisayar Mühendisliği Yüksek Lisans
Tez Başlığı[3] : REST MVC Uygulayarak Şehir Meslelerinin WEBGL ile Görselleştirilmesi

---

Tez Sınav Jürisi	Öğretim Üyesi	İmza
Danışman	: Yrd.Doc.Dr.Umit ISIKDAG	
Üye	: Yrd. Doç. Dr. Ediz ŞAYKOL	
Üye	: Yrd. Doç. Dr. Turhan KARAGÜLER	

[1] Jüri üyeleri söz konusu tezin kendilerine teslim edildiği tarihten itibaren en geç bir ay içinde toplanarak öğrenciyi tez savunma sınavına alır. Belirlenen günde yapılamayan jüri toplantısı, katılanların hazırladığı bir tutanakla enstitü yönetimine bildirilir. Bu durumda jüri en geç onbeş gün içinde toplanarak adayı tez savunma sınavına alır. Tez savunma sınav süresi en az 45 dakikadır. Yüksek lisans tez savunma sınavı, tez çalışmasının sunulması ve bunu izleyen soru-yanıt bölümlerinden oluşur ve dinleyiciye açıktır. (Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-3)

[2] Tez sınavının tamamlanmasından sonra jüri, tez hakkında "kabul", "düzeltme" veya "red" kararı verir. Jüri başkanı, jüri üyelerince imzalanmış sınav tutanağını, tez sınavını izleyen üç gün içinde ilgili enstitü yönetimine teslim eder. Tezi başarısız bulunan öğrencinin Enstitü ile ilişiği kesilir. Tezi hakkında düzeltme kararı verilen öğrenci en geç üç ay içinde gerekli düzeltmeleri yaparak ve yönetmelikte belirtilen usullere uygun olarak tezini aynı jüri önünde yeniden savunur. Bu savunma sınavında da tezi kabul edilmeyen öğrencinin enstitü ile ilişiği kesilir.(Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-4)

[3] İleridedoğabilecekaksaklıklarınengellenmesiiçintezinbaşlığınınyazılmasıgerekmektedir.

REST MVC UYGULAYARAK ŞEHİR NESNELERİNİN WebGL İLE GÖRSELLEŞTİRİLMESİ

Tezi Hazırlayan : Seyfullah Tıkız

Özet

Bu tez çalışmasında Html 5 ile gelen Canvas kontrolü üzerine geliştirilen WebGL teknolojisi ile tarayıcı içinde üç boyutlu çizim yaparak şehir nesneleri üzerinde gezinme üzerinde durulmaktadır. Bu teknoloji sayesinde bilgisayara herhangi bir program yüklemeden tarayıcı içinde harita çizme imkanı sağlanmaktadır. Sunucu ile hızlı haberleşebilmek için REST servisleri ve JSON veri yapısı kullanılmaktadır. Veritabanı olarak da Oracle Spatial ürünü denenmektedir.

Anahtar Kelimeler : WebGL, Oracle Spatial, Rest

A RESTISH IMPLEMENTATION FOR VISUALISING CITY OBJECTS USING WebGL

Thesis Prepared By : Seyfullah Tıkız

Summary

In this thesis work navigating on city objects that are drawn as three dimensions in Canvas control that has come with Html 5 is emphasized. Thanks to this technology there is no need to install any program to draw a map in a browser. To communicate with server in a fast way, REST services are implemented and JSON data type is used. Oracle Spatial is experimented as database.

Keywords : WebGL, Oracle Spatial, Rest

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

KISALTMALAR

3B :Üç Boyutlu
API :Application Programming Interface(Uygulama Programlama Arayüzü)
BG :Bilgisayar Grafikleri
BDT :Bilgisayar Destekli Tasarım
CAD :Computer Aided Design (Bilgisayar Destekli Tasarım)
GIS :Coğrafi Bilgi Sistemi
CSS :Cascading Style Sheets (Basamaklı Biçim Sayfaları)
DAM :Dijital Arazi Modeli
DOM :Document Object Model (Döküman Nesne Modeli)
GL :Graphics Library (Grafik Kütüphanesi)
GIS :Geographic information system(Coğrafi Bilgi Sistemi)
GML3 :Geography Markup Language(Coğrafya Biçimlendirme Dili)
HTML :HyperText Markup Language (Zengin Metin İşaret Dili)
HTTP :Hyper-Text Transfer Protocol (Hiper Metin Transfer Protokolü)
İGS :İleri Görselleştirme Sistemleri
İVA :İntegraf Voksel Analiz
JSON :Javascript Object Notation (Javascript Nesne Gösterimi)
MVC :Model-View-Controller (örnek, görünüm, denetçi)
REST :Representational State Transfer (Temsili Durum Transferi)
SOAP :Simple Object Access Protocol (Basit Nesne Erişim Protokolü)
URI :Uniform resource identifier (Tekbiçimli kaynak tanımlayıcı)
VTYS :Veritabanı Yönetim Sistemi
WebGL :Web Graphic Library (Web Grafik Kütüphanesi)
W3C :World Wide Web Consortium
WHATWG :Web Hypertext Application Technology Working Group
XML :Extensible Markup Language (Genişletilebilir İşaretleme Dili)

TABLO LİSTESİ

1 GİRİŞ

Var olan sistemlerde şehir nesnelerinin görselleştirilmesi masaüstü uygulamalar ile sağlanmaktadır [1]. Bu bilgisayara ayrı bir kurulum gerektirmektedir ve kurulum ile ilgili donanımsal gereklilikler ve güvenlik ile ilgili gereksinimlerin karşılanması için bir çalışma ihtiyacını doğurmaktadır. Ayrıca üç boyutlu şehir nesnesi bilgileri çeşitli veri kaynaklarında tutulabilmekte ama bunların uzaysal kavramlara göre sorgulanması ve performansı çeşitli zorluklar içermektedir [Ek 1]. Çoğu uygulamada kullanıcı arayüzü ve veritabanı birbirine sıkı bir şekilde bağlanmış durumdadır ve bu farklı veritabanı veya kullanıcı arayüzü kullanmayı zorlaştırmaktadır.

Üç boyutlu verinin kullanıcı makinesine ulaşması için kullanılan ağ protokolleri ve mesajlaşma sistemi genelde uygulamaya özel olduğundan farklı uygulamalar ile entegrasyonu zorlaşmaktadır ve bunlar gereksiz paket boyutlarına neden olarak ağ trafiğini olumsuz etkileyebilmektedir.

1.1 Amacı

Bu çalışma ile web arayüzü kullanılarak uzaysal verinin HTML 5 ile gelen yeni Canvas elementi üzerine geliştirilen WebGL uygulama çatısı sayesinde bu teknolojiyi destekleyen bir tarayıcıda her hangi bir program veya eklenti kurmaksızın üç boyutlu çizim yapılmasını sağlamak üzere bir çalışma yapılacak ve bu çalışma canlı bir yazılım uygulaması üzerinde gösterilecektir.

Burada şehir nesnesi bilgileri Oracle veritabanı üzerine bir seçenek olarak gelen Oracle Spatial ile üç boyutlu verilerin depolanması ve uzaysal sorgular ile performanslı bir şekilde veriye ulaşılması sağlanacaktır.

Araştırmada web katmanı öncesi REST servisleri kullanılarak farklı platformlar ve uygulamalar için de HTTP protokolünde veri sunulması sağlanacaktır.

REST web servisleri kullanarak standart olarak kabul görmüş bir uygulama çatısı kullanılmış olacak bunun sağladığı fayda ile var olan bir çok programlama dilinde hazırlanmış REST kütüphaneleri ile hızlı uygulama geliştirme sağlanabilecektir. Buna ilave olarak REST teknolojisi öncesinde gelmiş olan SOAP

teknolojisinde kullanılan bir çok mesaj paketleme aracını kullanmadığında ve sadece XML veri yapısı yerine JSON veri yapısını da desteklediğinden paket boyutlarında ciddi indirimler sağlamakta ve daha hızlı ağ iletişimi sağlanabilmektedir.

1.2 Önemi

Bu çalışma ile 3B şehir nesnelerinin görüntülenmesi uygulamalarının masaüstü yapılması zorunluluğu ortadan kalkmış olabilecek, herhangi bir program veya eklenti kurulmadan bu tip uygulamalar bilgisayarda çalıştırılıyor olacaktır.

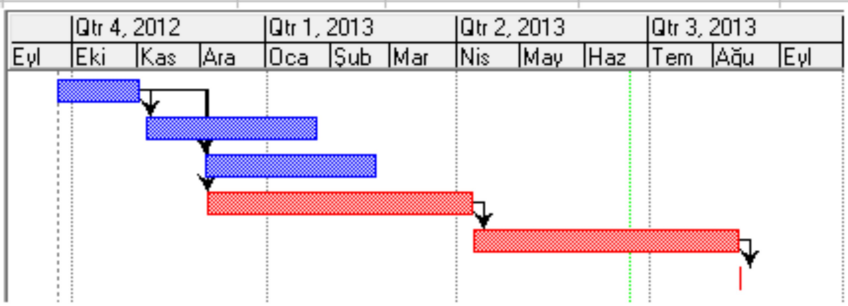
Servis odaklı bir yaklaşım ile sunulan bu veriler aynı zamanda performanslı bir şekilde de sorgulanabilecektir.

1.3 Kapsamı

Bu çalışma ile web ortamında 3B harita ile şehir nesneleri görselleştirilmeye çalışılacaktır. Görselleştirilen nesneler üzerinde dolaşım imkanı ve farklı açılardan görüntüleme imkanı sağlanacaktır. Böylece aynı sahne farklı açılardan görüntülenebilecektir.

1.4 Araştırma Takvimi

	Name	Duration	Start	Finish	Predecessors
1	Konu Seçimi	30 days	24.09.2012 08:00	02.11.2012 17:00	
2	Literatür Taraması	60 days	05.11.2012 08:00	25.01.2013 17:00	1
3	Teknoloji Seçimi	60 days	03.12.2012 08:00	22.02.2013 17:00	1
4	Yazılım Geliştirme	90 days	04.12.2012 09:00	09.04.2013 09:00	1
5	Tez Yazımı	90 days	09.04.2013 09:00	13.08.2013 09:00	4
6	Tez Sunumu	1 day?	13.08.2013 09:00	14.08.2013 09:00	5



***Şekil 1**: Araştırma Takvimi*

1.5 Araştırmanın Modeli

Araştırma tarama ve deney modelindedir. Araştırmada çeşitli doktora tezleri, yazılım üreticileri dökümantasyonları taranmıştır. Bu taramalar sonucunda örnek bir uygulama ortaya konmuştur.

1.6 Sınırlılıklar

Bu araştırma WebGL, REST ve MVC kullarak web sayfası üzerinde Oracle Spatial verisinin 3B gösterilmesi ile sınırlıdır.

1.7 Verilerin Toplanması

Verilerin toplanması sırasında akademik ulusal ve uluslarası kütüphaneler, Beykent Üniversitesinin üye olduğu uluslararası elektronik kütüphaneler ve buralardaki doktora tezleri, akademik makaleler, çalışmada kullanılan teknolojileri sağlayan yazılım üreticilerinin teknik dökümantasyonları, ilgili teknolojiler için oluşturulmuş öğretici siteler kullanılmıştır.

1.8 Veri Toplama Araçları

Veri toplama araçları olarak yazılı kaynaklar kullanılmıştır.

1.9 Veri Toplama Araçlarının Geçerliliği ve Güvenilirliği

Çalışma sırasında toplanan veriler ile yapılan örnek uygulama bir bilgisayar programı olduğu için aynı bilgisayar yapılandırması ile aynı tarayıcı ile her zaman aynı sonucu verecektir.

2 3B SAYISAL KENT MODELLEME

2.1 Şehir Modelleme

2007 yıllarında çoğu belediye 3B şehir modelleri hazırlamaya karar veriyordu. Genelde bundaki temel amaç şehir planlama süreçlerine yardımcı olmaktı. Buna rağmen çoğu zaman bu sanal sahnelerin görselleştirilmesi ile sınırlıydı. İlk nedeni o zamanlar ticari 3B coğrafi bilgi sistemlerinin olmayışıdır. Bundan dolayı şehir modelleri CAD sistemleri veya sınırlı modelleme yeteneklerine sahip görselleştirme yazılımı ile uygulanıyordu. İkinci nedeni ise o yılda 3B şehir modelleme standardı yoktu. Sadece bir kaç çok işlevli ve çok ölçekli modelleme, depolama ve analiz gerçekleştirilmişti. Bu uygulamaya özel 3B modeller için desen sağlayan bir temel şema oluşturmaktadır. Bu gerçek dünya nesneleri geometrik, topolojik ve tematik (yani konumsal olmayan) özelliklere sahip özellikleri ile nasıl temsil edildiği gösterilmiştir. Burada çok ölçekli temsil sorunu ile başa çıkmak için uğraşıyoruz. Özellikleri ve geometri arasında özel bir seviye detay ilişkisi farklı ölçeklerde 3 boyutlu modeller arasında mekansal tutarlılığı sağlamak için tanıtıldı. Ayrıca sayısal arazi modeli ile yüzey altındaki özellikleri entegrasyonu ile ilgili konular ele alınmıştır. Son olarak, sistem düzeyinde birlikte çalışabilirliğin GML3 için önerilen model haritalama ile elde edilebilir olduğu gösterilmiş, mekansal veri temsili ve değişimi için yeni bir standart OpenGIS Konsorsiyumu tarafından geliştirilmiş.

Büyüyen mekansal veri altyapısı bağlamında bir önemli konu sistemleri ve verinin (Groot ve McLaughlin, 2000) birlikte çalışabilirliğidir. Farklı sistemler, biçimleri ve kavramsal modelleri farklı yerlerde mevcut dağıtılmış mekansal bilgilerin hemen birleşmesini önler. Örneğin bir afet yönetim sistemi birden fazla şehri etkileyen seli izleyebilmek için her belediyenin mekansal verisine erişime sahip olması gerekir. Ayrıca, sistem tehlike altında olan binaları belirlemek için her veri kümesini yorumlamasını bilmelidir. Farklı mevcut modellerin çeşitliliği nedeniyle bu yaklaşım açısından olanaksızdır. Genel olarak, bu problem farklı makamlardan veya veri sağlayıcılardan gelen büyük alanların analizinde veya görşelleştirilmesinde kullanılan 3 boyutlu modelleri entegre etmeyi gerektiren tüm uygulama senaryolarında ortaya çıkar. Bishr'e göre, birlikte çalışabilirlik sorunları anlamsal,

şema ve sözdizimsel düzeyde (Bishr, 1998) ortak modelleri ve yapıları belirterek aşılabilir.

"Kuzey Rhine-Westphalia coğrafik bilgi altyapısı" (GDI NRW) girişiminde "3B" Özel İlgi Grubunda (SIG 3D) belediyeler, bilim adamları ve yazılım geliştiriciler 3B şehir modelleri için birleşik bir yaklaşım geliştirme üzerine çalışmaktadır. Bu çalışmada sunulan modelleme şemaları bu grup içindeki istişareler sonucu geliştirilmiştir. Amaç, analiz ve simülasyon amaçlı olarak görüntülenmesi için kullanılabilecek bir çok fonksiyonlu 3B şehir modeli elde etmektir. Bu 3B kadastro veya tesis yönetimi gibi özel uygulamaları bunun üstüne inşa edebilmek için temel bir model olmalıdır.

Genel olarak, bir mekansal temel model mekansal bütünlüğünü korumak için araçlar sunmalıdır. Böylece sadece geometrisi değil aynı zamanda topolojik açıdan göz önünde bulundurulmalıdır. Dahası model gerçek dünya nesnelerinin tematik ve konumsal yapılarına bakarak farklılaştırılmış temsillerine olanak sağlamalıdır. Bu genelleşme ve uzmanlaşma ile karmaşık nesne türlerinin tanımını içerir. Örneğin, bir belediye bir yandan tam bir küçük ölçekli, kaba ve diğer yandan da seçilen binaların ayrıntılı temsilleri olan 3 boyutlu kent modeline sahip olabilir. 3 boyutlu kent modelleri için tipik olan bir sorun kullanıcılar verimli görünüm için farklı ölçeklerde nesneleri karıştırmak veya analiz görevleri için en yüksek detaylı gösterimi seçmek isteyebilir. (Koninger & Bartel, 1998; Coors & Flick, 1998; Zipf & Schilling, 2003). Her iki durumda da hiçbir mekansal nesne iki veya daha fazla kabul edilmemeli ve sayılmamalıdır. Otomatik tutarlılık denetimi etkinleştirmek için, farklı ölçeklerde bir nesnenin mekansal temsilleri arasındaki ilişkilerin bariz yapılması gerekir.

Yüzeydeki nesnelere ek olarak, şehirler tipik olarak yer altı yapılarına, mesela yaya altgeçitleri, tüneller, metro, sahiptir, ancak bunlar teoride ve 3B coğrafik bilgi sistemlerinde ve 3B şehir modellerinde ihmal edilmişti. Yaya gezinmesi ve afet yönetimi veya yüksek detaylı görselleştirme gibi yeni uygulamalar için yer altı yapıları da gereklidir. Burada asıl sorun yer altı konumsal nesneler, dijital arazi modeli ve yer üstü nesneler arasındaki arayüzdür.

Sistem düzeyinde birlikte çalışabilirlik elde etmek için, sözdizimsel uyumluluk elde edilmelidir. OpenGIS konsorsiyumu Geometrik İşaretleme Dili GML3'ü önerdi.

Bu 2B, 2.5B ve 3B konumsal nesneleri kodlamak için en önemli standart olacaktır [2].

2.2 3B GIS

Yeni nesil Coğrafik Bilgi Sistemi (GIS) konumsal veri modelleme için yeni bir yöntem gerektirmektedir. Bu yeni nesil GIS'e 3B GIS denmektedir. Temel olarak yeni bir dijital model geliştirilmeli veya kurulmalıdır. Hayat kalitesini yükseltmek veya tehlike ve afetleri önlemek için dijital hesaplama teknolojisini sömürmek dünya ve çevresinin dijital şekilde bir modelinin oluşturulmasını gerektirir. Böyle bir model, karmaşık gerçekliğin basitleştirilmiş tanımı, rahatlıkla kullanılabilir, depolanabilir, yönetilebilir, sürdürülebilir, dağıtılabilir ve taşınabilir. Bir dijital model gerçekliğin konumsal ve konumsal olmayan taraflarını içerir ve ilgili parçalar arasında işlem ve iletişim için temel sağlar. Bir model nesneleri nesne veya nesne kümesi olarak ayırır, incelemedeki gerçekliğin öğelerini içerir. Konumsal tarafları şekil, boyut ve yer ile ilgili geometrik özelliklerdir. Konumsal olmayan tarafları isim, renk, işlev, fiyat, sahiplik ve bunun gibi genellikle tematik özellikler olarak değinilir. Gerçekliğin konumsal tarafları iyi ve ekonomik olarak grafik biçiminde sunulabilir, buna rağmen konumsal olmayan taraflar çoğu durumda metin olarak sunulabilir. Grafik sunum gerçekteki durumu hızlı anlamaya olanak sağlar, yüksek seviyeli soyutlamaya veya komşu ilişkilerini anlamaya olanak verir, buna karşın metinsel sunum grafik olarak anlatılamayan durumlara daha uygundur. Bir dijital model bu iki sunumu ilişkilendirmeye kabiliyetli olmalıdır. Böyle bir modeli gerçekliğin yapay inşası olarak oluşturmak hesaplama ortamında hem bilgisayar grafiklerini (BG) (Sutherland, 1963, 1970; Foley et al., 1992; Watt, 1993), hem de veritabanı yönetimini (VTYS) kullanmayı gerektirir. Coğrafik bilgi sistemleri (Burrough, 1986; Maguire et al., 1991) ve bilgisayar destekli tasarım (BDT) böyle aletlere örnektir. GIS ve BDT arasındaki temel fark konumsal olmayan taraflardan ziyade konumsal tarafları ele alış biçimidir.

Coğrafik Bilgi Sistemleri coğrafi bilgiyi yakalama, depolama, işleme ve analiz etme için güçlü araçlar sağlar. Bu araç araştırmacılar, haritacılar, tepeden fotoğraf çekenler, inşaat mühendisleri, fiziksel planlamacılar (kentsel ve kırsal), kırsal ve kentsel geliştiriciler, jeologlar, vb çeşitli coğrafya ile ilgili uzmanlar tarafından kullanılmaktadır. Karmaşık konumsal verinin ilişkisel veritabanından

ziyade nesne tabanlı olarak ele alınması önerilmektedir.

2B GIS için özellikle coğrafya biliminde limitler olduğu Ones (1989), Raper ve Kelk (1991), Rongxing Li (1994), Houlding (1994), Bonham-Carter (1996), ve Wei Guo (1996) tarafından belirtilmiştir. Bahsedilen limitler veri boyutluluğu ve veri yapıları ile ilgilidir. Tek değerli z-koordinatlı nokta gibi veri x, y, koordinatlarında temsil edilebilir, ancak cevher gövdesi gibi çoklu z değerlerinde yetersiz kalmaktadır (Bonham-Carter, 1996; Raper and Kelk, 1991). 3B GIS kurmada büyük bir engel Jones'un (1989) ve Rongxing Li'nin (1994) bildirdiğine göre yetersiz konum veri yapısı ile ilişkilidir. Bu iki yazar 3 boyutlu veri için veri yapılandırma sorununa karşın voksel veri yapıları önermiş, fakat yapı üzerinde gerçek işlevsel sistem geliştirilmemiştir. Sorun coğrafik mantık alanında Houlding (1994) tarafından da belirtilmiştir. Doğru sunumlar ve konumsal bilgi, mesela yüzey altı 3B nesneler için, 2B sistemler ile elde edilemez. 3B görselleştirme araçları tek başına (İleri Görselleştirme Sistemleri (İGS), İntegraf Voksel Analiz (İVA) ve diğer Dijital Arazi Modeli (DAM) paketleri) böyle bir veriyi doğru şekilde istendiği gibi yönetemez. Mesela Wei Guo (1996) 3B binaların modellemesini Molenaar'ın (1992) önceki veri yapılarını kullanarak ilişkisel veritabanı ortamında 3B görselleştirme aracı olarak AutoCad ile yapmayı test etti. Literatürde var olan GIS'lerin çoğu 2B konumsal veri için yeterli olduğu fakat 3B konumsal veri ve daha fazlası için zorluk yaşadığı, bundan dolayı var olan sistemlere en azından üçüncü bir boyut (3B) ile bir eklenti yapılması gerektiği GIS araştırmacıları tarafından ortak bir öneri olagelmiştir.

2.3 3B GIS İhtiyacı

Biz 3B bir dünyada yaşamaktayız. Yeryüzü bilimadamları ve mühendisleri uzun zamandır kroki ve çizimlerle gerçekliğin 3B konumsal yanlarını grafiksel olarak ifade etmenin yollarını arayageldiler. Gerçekliğin 3B grafik tanımı yeni bir şey değil. Perspektif görünümde gerçekliğin 3B tanımı görüş yeri ile değişir, böylece bunun oluşturulması oldukça yorucudur. Geleneksel haritalar bunun üzerinden ortogonal izdüşümler ile gelmişlerdir. Buna rağmen çok sınırlı bir 3B hissi verebilmiştir.

Bu geleneksel çizimler ve haritalar konumsal açıklamaları 3B'dan 2B'a indirgemektedir. Buna rağmen hesaplama teknolojisini kullanarak gerçeklik hakkındaki bilgi 3B modelleme denen işlem ile 3B dijital modele aktarılabilir.

Gerçekliğin 3B tanımı görüş açısından bağımsızdır. İncelemedeki gerçekliğin yeterli şekilde anlaşılabilmesi için bir çok yönden ele alınması gerekir. Jeoloji (Carlson, 1987; Bak ve Mill, 1989; Jones, 1989; Youngman, 1989; Raper ve Kelk, 1991), hidroloji (Turner, 1989), inşaat mühendisliği (Petrie ve Kennie, 1990), çevre mühendisliği (Smith ve Paradis, 1989), peyzaj mimarlığı (Batten, 1989), arkeoloji, meteoroloji (Slingerland ve Keen, 1990), cevher arama (Slides 1992), 3B kent haritalama (Shibasaki et al., 1990; Shibasaki ve Shaobo, 1992) disiplinlerinden uzmanlar 3B modellemeyi kendi işlerini verimli bir şekilde bitirmek için kullandılar.

Bir 3B model eldeki işi yapmak için gerekli sistemin temelidir. Scott (1994) Bak ve Mil'in (1984), Fisher'in (1993), Kavouras ve Masry'ın (1987), Raper'ın (1989), Raper ve Kelk'in (1991) çalışmalarını 3B modellemeden gereken işlevleri sağlamak için özetledi. Bu çalışmalar farklı kaynaklardan 3B model oluşturmak için bir araç oluşturmalı, mevcut modellerin bakımına olanak vermeli, 3B görselleştirmeyi kolaylaştırmalı; mesela gizli çizgi/yüzey kaldırılması, yüzey aydınlatması, doku eşlemesi ile ortografik, perspektif veya steryo görüntüye olanak vermeli; hacim, yüzey alanı, ağırlık merkezi, makul yol hesaplamalarına konumsal ve konumsal olmayan arama ve sorgulama için konumsal analize olanak vermelidir.

BDT, araba, makine, uçak, uzay araçları, yapı endüstrisi ve mimari tasarımda kullanılan 3B tipik bir BG aracıdır. BDT modelin geometrik yönüne ve 3B görselleştirmesine odaklanır. Bir örnek olarak gizli çizgi ve yüzey gidermesi, yüzey aydınlatması, ışın izleme ve doku eşlemesi ile perspektif görünüm verilebilir. Sorun ise BDT'ın yukarda sayılan disiplinlerdeki tüm görevleri destekleyip destekleyemeyeceği kısmında ortaya çıkmaktadır. BDT, 3B modelleme ve işlevselliği gerektiren yeryüzü bilimlerindeki işler için kullanma girişimlerinde bulunuldu. Buna rağmen BDT bu işleri gerçekleştirmek için aşağıdaki nedenlerden ötürü uygun sayılamaz.

- BDT, insan yapımı iyi tanımlanmış şekilleri, büyüklükleri, konumsal ilişkileri ve tematik özelliklerin tasarımındaki sorunları çözmek için geliştirilmiştir. BDT, veri yapılandırması veya iyi tanımlanmamış şekiller, boyutlar, konumsal ilişkiler ve tematik sorunlar ile başa çıkabilecek araçlara sahip değildir. Ne konumsal ilişkileri analiz edebilir, ne de ayrık veri kümeleri ve GIS'de karşılaşılan belirsizliklerle başa çıkabilir. Mesela BDT yeryüzü

bilimindeki analizlerde önemli olan nesneler arasındaki komşuluk ilişkisini güvenilir bir şekilde yönetemez, çünkü bu ilişkiler tasarımda önemli görülmez.

- Mesela bina gibi bir nesneyi tasarlamak öznel bir meseledir. Nesnelerin tüm yönleri ve onlar arasındaki ilişkiler bir insan tarafından karar verilmek durumundadır; otomatikleştirilebilecek çok az şey vardır. Yeryüzü bilimi uygulamaları var olan nesneleri şekilleri, boyutları ve aralarındaki ilişki ile insanın kontrolü dışında modelleme arayışı içindedir. Burada çok fazla sayıda nesne olduğu için otomasyon istenmektedir. Konumsal analiz için önemli bazı ilişkiler otomatik olarak oluşturulmalı. BDT genellikle böyle bir otomasyon için bir işlev sunmamaktadır.

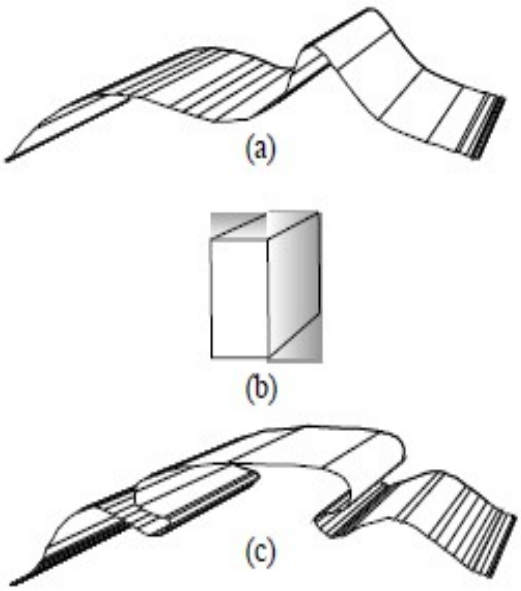
- BDT nesne tanımına 3B'dan başlar. Nesneler 2B komponentlere ayrıldığında, aralarındaki ilişki bilinir. Yeryüzü uygulamalarında gerçekliğin tipik modelleri, genellikle 2B'dur ve uygulama görünümü, mevcut araçlar ve bilgi ile yönetilir. Komponentler sonradan birleştirilmeli ve aralarındaki ilişkiler sonraki aşamalarda ortaya çıkarılmalıdır. Bu, BDT için ayrık bileşenlerin arasındaki ilişkiyi çözebilecek yeterli aracı olmadığı için çok zordur.

- BDT küp, silindir veya küre gibi basit şekilleri birleştirerek karmaşık nesneleri oluşturur. Şekil değiştirme, bileşim, kesişim gibi işlemleri böyle bileşenlere uygulayıp karışık nesneler elde etme zaten bilinen mekanizmalardır. Yeryüzü bilimi uygulamaları böyle karışık nesnelere genelde bütün olarak yaklaşmaktadır. Ana bileşenlerine ayırmak BDT'nin aksine tersine mühendislik gibidir. BDT tarafından kullanılan modelleme yaklaşımı bundan dolayı yeryüzü bilimi uygulamaları için her zaman uygun olmayabilir. İnsan yapımı nesnelerden ziyade karmaşık gerçekliği ifade etmek için daha düşük seviyedeki nokta ve çizgi gibi geometrik bileşenlere ihtiyaç vardır.

Bu geometrik bileşenler aynı zamanda BDT'nin sağlayamadığı ilişkili işlemleri de belirler.

Yeryüzü uygulamaları için daha uygun bir araç 3B modelleme kabiliyetine

sahip GIS olabilir, yani 3B GIS. 2007 yılında 3B modelleme kabiliyetine sahip, yukardaki listedeki tüm işlevleri sağlayabilen ticari bir araç olarak mevcut değildi. Çoğu GIS kendi geometrik modelleme sistemlerini 2B ile sınırlandırmaktadır, böylece BDT ile sağlanan 3B sunum, analiz ve görselleştirme mümkün olmamaktadır. Üçüncü boyutu modellemek için en büyük çaba arazi kabartması ve dijital arazi modellemesi (DAM) sunumunda bulunabilir. DAM, perspektif görünümde kabartma ile ilgili eğim, bakı, yükseklik bölgesi, görünürlük, kesme ve dolgu hacmi, yüzey alanı ve 3B görselleştirme için konumsal analizlere kolaylık sağlar. Buna rağmen, DAM'ın temeli tüm planimetrik yerler için tek yükseklik değerli sürekli bir yüzeydir. DAM, 3B bir katı nesneyi veya belirli bir planimetrik yerde çok yükseklik değerine sahip bir yüzey için kullanılamaz.



Şekil 2: Tek değerli yüzey (a) 3B katı nesne (b) ve çok değerli yüzey (c)

Tarama temelli sistemlere her ne kadar 3B GIS'leri gözüyle bakılabilirse de orjinal veri kümesindeki mevcut gerçeklikteki malumatı koruyamayabilir. Bu malumat çözünürlük ve yeniden örneklendirmedeki sorunlardan dolayı kaybolabilir. Bir çare olarak orjinal veri kümesi modelden ayrı olarak depolanmalıdır, mesela, şunlar için:

- eğer sonuç uygun olmayan matematiksel tanımdan dolayı yetersiz olursa modeli tekrar oluşturmak için
- farklı bir çözünürlükte başka bir model oluşturmak için
- yeni bir model oluşturmak için başka bir veri kümesi ile birleştirme sırasında
- bir modelin referansı veya kanıtı olarak arşivlemek için

Bu etkinlikler gelecekte kullanım için orjinal verinin uygun bir yapıda saklanmasının gerekli olduğu anlamına gelmektedir. Her bir veri öğesine veri hakkında gerekli bilgi eklenmeli. DAM'nde mesela, bir çizginin kesme çizgisi

olduğu bilgisi tutulmalıdır, çünkü bu enterpolasyonda bir etkiye neden olacaktır. Benzer şekilde veri kullanma stratejilerini etkileyen başka bilgiler de eklenmelidir.

Ne BDT, ne de GIS şu anda yeryüzü bilimleri için gereksinimleri karşılayamadığından daha fazla 3B GIS araştırma ve geliştirmesi uygun görünmektedir [3].

3 TEKNOLOJİ ANALİZİ

Bu kısımda çalışmanın daha iyi anlaşılması için temel bazı tanımlara yer verilecektir.

3.1 HTML 5

- Yeni özellikler HTML, CSS, DOM ve JavaScript üzerine kurulmalı.
- Harici eklentilere olan ihtiyaç azaltılmalı. (Flash gibi)
- Daha iyi hata yönetimi
- Kodlama yerine daha fazla işaretleme dili
- HTML5 aygıt bağımsız olmalı
- Geliştirme süreci halka açık olmalı [2]

Aşağıda en küçük bir HTML 5 dosyası oluşturmak için gerekli kod verilmiştir.

```
<!DOCTYPE html>
<html>
<head>
<title>Döküman başlığı</title>
</head>
<body>
  Döküman içeriği.....
</body>
</html>
```
[4]

HTML 5 ile bir çok kullanılmayan veya amacı dışında kullanılan özellik kaldırılmış, yeniden yazılmış ve yeni öğeler eklenmiştir.

Bu çalışmada HTML5 ile gelen canvas öğesi üzerinde durulacaktır.

<canvas> öğesi web sayfası üzerinde o anda kodlama ile (genellikle javascript, bu projede WebGL ile) grafik çizmek için kullanılır.

Kırmızı dikdörtgen, eğimli dikdörtgem, çok renkli dikdörtgen ve bir kaç renkli yazı canvas öğesi üzerine çizilebilir.

<canvas> öğesi grafikler için sadece bir kaptır. Asıl çizimi yapmak için kodlama yapmak gerekir.

Canvas yol, kutu, karakter çizmek ve resim eklemek için bir çok metoda sahiptir.

Aşağıdaki kod ile web sayfasındaki canvas öğesi içine kırmızı bir dikdörtgen çizilebilir.

```
<!DOCTYPE html>

<html>

<body>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #c3c3c3;">

Your browser does not support the HTML5 canvas tag.

</canvas>

<script>

var c=document.getElementById("myCanvas");

var ctx=c.getContext("2d");

ctx.fillStyle="#FF0000";

ctx.fillRect(0,0,150,75);

</script>

</body>

</html>
```

Aşağıdaki gibi açıklayabiliriz:

Önce, <canvas> öğesini bul:

```
var c=document.getElementById("myCanvas");
```

Sonra, onun getContext() metodunu çağır. ("2d" metnini bu metoda geçmek gerekir.):

```
var ctx=c.getContext("2d");
```

getContext("2d") nesnesi yol, kutu, daire, metin resim ve daha fazlasını çizmek için bir çok özellik ve metodlar barındıran HTML5 dahili nesnesidir.

Sonraki iki satır kırmızı bir dikdörtgen çizer:

```
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
```

fillStyle özelliği CSS rengi, bir eğim veya bir desen olabilir. Varsayılan fillStyle değeri #000000 (siyah)'dır.

fillRect(*x,y,genişlik,yükseklik*) metodu şu anki doldurma sitili ile bir dikdörtgen çizer [5].

3.2 SOAP

SOAP uygulamaların HTTP üzerinden veri alışverişi yapabilmesi için XML tabanlı basit bir protokoldür. Başka bir deyişle SOAP bir Web Servise ulaşmak için bir prokoldür [6].

SOAP Yapı Taşları :

Bir SOAP mesajı aşağıdaki öğeleri içeren sıradan bir XML dökümanıdır:

- XML dökümanını bir SOAP mesajı olarak tanıtan bir Envelope (zarf) öğesi
- Başlık bilgisini içeren bir Header (başlık) öğesi
- Çağrı ve cevap bilgisini içeren bir Body (gövde) öğesi
- Hata ve durum bilgisini içeren bir Fault (hata) öğesi

Yukarıda tanımlanan tüm öğeler SOAP zarfı için varsayılan şu isim uzayında tanımlanmıştır:

http://www.w3.org/2001/12/soap-envelope

ve SOAP kodlama ve veri tipleri için varsayılan isim uzayı şudur:

http://www.w3.org/2001/12/soap-encoding

Sözdizimi Kuralları

Bazı önemli sözdizimi kuralları şunlardır:

- Bir SOAP mesajı XML ile kodlanmalıdır
- Bir SOAP mesajı SOAP zarf isim uzayını kullanmalıdır

•Bir SOAP mesajı SOAP kodlama isim uzayını kullanmalıdır
•Bir SOAP mesajı DTD referansı içermemelidir
•Bir SOAP mesajı XML işleme komutları içermemelidir

SOAP Mesaj İskelet Yapısı

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

[7]

3.3 REST

REST tabiri ilk önce 2000 yılında Roy Fielding tarafından onun doktora çalışmasında ortaya çıkmıştır [8]. REST, WWW gibi dağıtık sistemler için bir yazılım mimarisidir. REST'in başlıca özellikleri şunlardır:

- İ*stemci- Sunucu* : Bir arayüz istemci ile ve sunucu birbirinden ayrılır. Böylece istemci veri depolama ile ilgilenmez, bu şekilde istemci kodun taşınabilirliği artar. Sunucu da istemci arayüzü ile ilgilenmez, bu şekilde daha basit ve daha ölçeklenebilir olurlar. İstemci ile sunucu arasındaki arayüz değişmedikçe birbirlerinden bağımsız geliştirilebilir veya tamamen değiştirilebilirler.

- *Durumsuz* : İletişim doğası gereği durum bilgisini taşımamalıdır, istemciden sunucuya giden her bir istek içinde isteği anlamak için tüm bilgiyi

barındırmalıdır ve sunucuda tutulan bağlam bilgisinden faydalanamaz. Oturum bilgisi tamamen istemcide tutulur.

- *Önbellek* : Durum bilgisinin sürekli taşınması gerektiği için oluşan fazla ağ trafiğini azaltmak için istemci tarafında önbellek kullanmak gerekmektedir. Önbellek kısıtı cevabın içindeki verinin açık veya gizli olarak önbelleklenebileceği veya önbelleklenemeyeceğinin belirtilmesini gerektirir. Eğer cevap önbelleğe alınabilirse, istemci önbellekleme ile aldığı veriyi daha sonra tekrar kullanma yetkisine sahip olur.

- *Tekdüze arayüz* : İstemciler ve sunucular arasındaki tekdüze arayüz mimariyi basitleştirir ve ayrıştırır, böylece tüm parçalar ayrı geliştirilebilir.

- *Katmanlı sistem* : Ara sunucular sistem ölçeklenebilirliğini yük-dengelemesi ve paylaşımlı önbellekler ile arttırır, hatta güvenlik politikalarını da zorlayabilir. Ancak katmanlar arası beklemelere de neden olup bir miktar yavaşlığa neden olabilir.

- *Çalıştırılabilir kod (seçimlik)*: Sunucular geçici olarak istemcideki işlevleri çalıştırılabilir kod göndererek arttırabilir veya değiştirebilir. Bunlar arasında java applet'ler, javascript sayılabilir.

HTTP'nin istek metodları isimli bir işlemler sözlüğü vardır. GET, POST, PUT, PATCH, DELETE işlemlerini ve başka HTTP'nin var olan özelliklerini kullanır.

Öte yandan SOAP RPC ise her bir uygulamanın yeni, uygulama bazlı işlemlerini tanımlamasını önerir. Mesela :

kullacılarıGetir()

suTarihtenSonrakiKullanicilariGetir(date denBeri)

siparisiKaydet(string musteriNo, string siparisNo)

Bu şekilde tekerleği yeniden keşfedercesine bir işlemler sözlüğü oluşturmak ise HTTP ile gelen yetkilendirme, önbellekleme, içerik-tipi sorgulama özelliklerinden faydalanamamaya neden olur. SOAP, HTTP geleneklerine uymadığı

için TCP, adlandırılmış kanallar, mesaj kuyrukları vb yapılarda da iyi bir şekilde çalışır [8][9].

RESTful Web API

Bir RESTful web API'si HTTP ve REST prensipleri uygulayarak oluşturulan bir web API'sidir. Bu kaynaklardan oluşan bir derlemedir ve dört görünüşü vardır:

- web API için temel URI, mesela

  http://example.com/resources/

- web API tarafından desteklenen internet medya tipi. Bu genellikle JSON'dır ama başka geçerli hipermetin standardı olan internet medya tipleri de olabilir

- HTTP metodlarını destekleyen web API işlemleri (mesela, GET, PUT, POST, DELETE)

- API hipermetin tipinde olmalıdır.

Aşağıdaki tablo web API uygulaması için sık kullanılan HTTP metodlarını gösterir.

Kaynak	**GET**	**PUT**	**POST**	**DELETE**
Toplama URI'ı, mesela http://example.com/resources/	URI'ları ve istenirse listenin detay üyelerini de **listele.**	Tüm listeyi başka bir liste ile değiştir.	Listede yeni bir kayıt **oluştur**. Yeni oluşan kayıdın URI'ı otomatik olarak atanır ve genellikle işlem tarafından döndürülür.	Tüm listeyi siler.
Öğe URI'ı, mesela http://example.com/resources/item17/	Listenin belirtilen üyesinin temsilini uygun medya tipinde belirtildiği şekilde **al.**	Listenin belirtilen elemanını değiştir veya yoksa, oluştur.	Genellikle kullanılmaz. Listede belirtilen öğeye bir liste gibi davranıp, kendi yetkisinde yeni bir kayıt oluşturur.	Listenin belirtilen öğesini siler.

Tablo 1: RESTful Web API HTTP metodları

[9]

SOAP temelli web servislerinin aksine RESTful web API'leri için bir resmi standart yoktur [10]. Çünkü REST mimari bir sitildir, SOAP ise bir protokoldür. REST bir standart olmamasına rağmen Web gibi RESTful uygulamalar HTTP, URI, XML gibi standartlar kullanır.

Görüldüğü üzere SOAP çoğu durumda gereksiz bir çok xml etiketi bulundurmakta ve mesajı büyütmektedir, REST ise sadece URI ve HTTP işlemleri kullanarak istek yapabilmekte, sunucudan cevabı ise XML biçimindeki veriye göre oldukça küçük JSON biçiminde veri olarak alabilmektedir. HTTP üzerinde çalışırken basit işlemler için REST ağ trafiğinden önemli ölçüde tasarruf ve hız artışı sağlar. Bundan dolayı bu çalışmada RESTful web servisleri kullanılacaktır.

3.4 MVC

MVC, model-view-controller (örnek, görünüm, denetçi) , paradigmasında kullanıcı girişi, dış dünyayı örnekleme ve kullanıcıya görsel geri besleme açıkça ayrılmıştır ve her biri kendi işi için geliştirilmiş üç çeşit nesne tarafından ele alınmıştır. Görünüm uygulamaya ayrılan grafiği ve/veya metin çıktısını yönetir. Denetçi kullanıcının fare ve klavye girişlerini karşılar ve örneği ve/veya görünümü uygun olduğu şekilde karşılar. Son olarak, örnek ise uygulamanın davranışını ve verisini yönetir ve onun durumunu genelde görünümden gelen sorgulamaya ve genelde denetçiden gelen durumu değiştirmeye yönelik komutlara cevap verir [11].

3.5 ASP.NET Web API

ASP.NET Web API tarayıcılar ve mobil cihazlar da dahil geniş bir istemci kitlesine hitap eden HTTP servislerinin geliştirilmesini kolaylaştıran bir uygulama çatısıdır. ASP.NET Web API, .NET Framework üzerinde RESTful uygulamalar geliştirmek için ideal bir platformdur. ASP.NET Web API ile uygulama geliştirmek için http://www.asp.net/web-api sitesi referans olarak kullanılabilir.

3.6 Oracle Spatial Database Option

Oracle Spatial uzaysal verinin hızlı ve verimli bir şekilde saklanması, erişilmesi analiz edilmesi için bütünleşik işlemlerden oluşur. Uzaysal veri gerçek veya kavramsal nesnelerin temel konum özelliklerini temsil eder.

Oracle Spatial, ya da Spatial da denir, Oracle veritabanında uzaysal özelliklerin depolanması, elde edilmesi, güncellenmesi ve sorgulanması için bir SQL şeması ve fonksiyonları sağlar. Spatial aşağıdaki bileşenleri içerir:

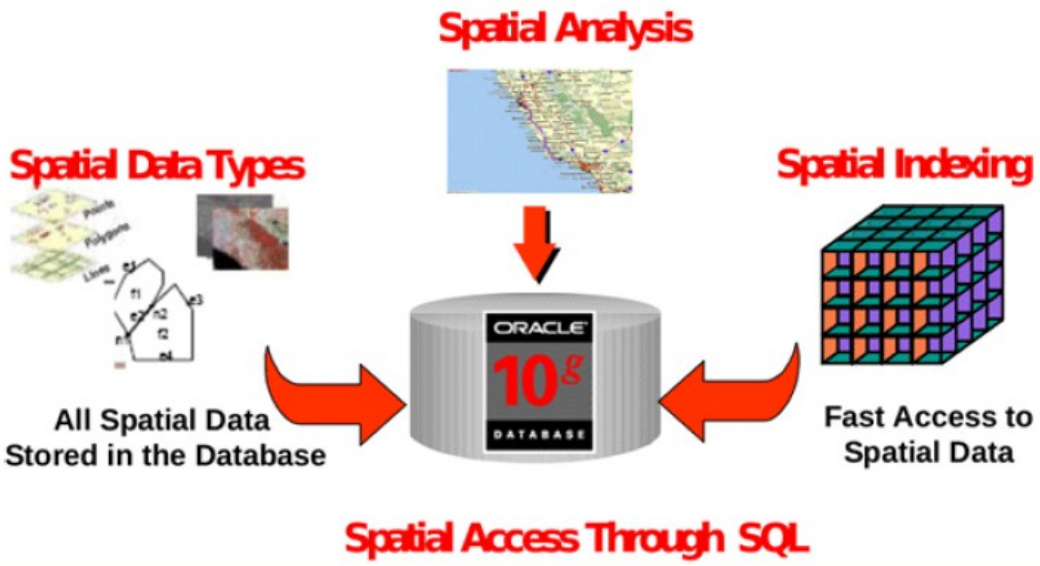
- Desteklenen geometri veri tiplerinin depolama, sözdizimi ve anlamını tanımlayan bir şema (MDSYS)

- Bir uzaysal veri indeksleme mekanizması

- Alanla ilgili sorgular, uzaysal birleşme sorguları ve diğer uzaysal analiz işlemleri için bir dizi operatör ve işlemler

- Yönetimsel araçlar

Uzaysal bir özelliğin uzaysal bir bileşeni bir koordinat sisteminde onun geometrik temsilidir. Buna geometri denir [12].



Şekil 3: Oracle Spatial'da Uzaysal Veri Tiplerinin, Analizin ve İndeksleme Yapılması

Oracle Spatial'ın bileşenleri şunlardır :

- MDSYS isimli ve geometrik veri tiplerinin saklanması, sözdizimi ve tanımlarını öngören Oracle şeması.

- Mekansal indeksleme sistemi.

- Komşuluk ve yakınlık analizi, kesişme, içerme vb. Muhtelif mekansal

sorguların yapılabilmesini sağlayan özel coğrafi fonksiyonlar, prosedür ve operatörler.

- Yönetimsel araçlar (utility& tuning)
- Topoloji veri modeli
- Network veri modeli
- GeoRaster özelliği
- 3D nesneler için veri tipleri ve operatörler
- Geocoder
- Routing motoru
- OpenGeospatial Consortium(OGC) uyumlu Web Servisleri

Oracle Spatial'ın faydaları şu şekilde belirtilebilir.

- Mükerrer mimariler azaltılır, tüm veriler aynı şekilde depolanır.
- Tüm veriler (metin, harita, multimedya) aynı yerde depolanır.
- SQL kullanılır, mekansal veri için yeni dillere ihtiyaç yoktur.
- SDO_GEOMETRY veri tipi OGC ve SQL veri tipleriyle denktir.
- Evrensel bir standarttır; önemli GIS yazılımları ve araçları tarafından desteklenir, esneklik sağlar.
- Güçlü özellikler içerir: Ölçeklenebilirlik, bütünlük, güvenlik, kurtarma.
- Mekansal verinin kullanımı ve bakımı için özel araçlar veya yetenekler gerektirmez.
- Grid computing özelliğinden faydalanılabilir.

Oracle Spatial veya Locator kurulunca MDSYS (Multi Dimensional SYS) şeması oluşturulur. Bu şema mekansal tip, paket, fonksiyon, prosedür ve meta veriye sahiptir. Oracle'daki standart SYS kullanıcısına benzer ve admin hakkına sahiptir.

Oracle Spatial, Oracle Enterprise Edition'da opsiyonel bir bileşendir. Ayrıca lisans ücreti gerektirir. Standard Edition and Standard Edition One sürümlerinde yoktur. Bu 2 sürümde Oracle Locator bileşeni ücretsiz olarak mevcuttur. Oracle Spatial'ın Oracle Locator'a göre aşağıdaki şekilde görüldüğü üzere bazı üstünlükleri vardır.

Şekil 4: Oracle Locator ve Oracle Spatial Karşılaştırması

Oracle Spatial'da vektör veriler için MDSYS.SDO_GEOMETRY veri tipi kullanılır. Bu yapı aşağıdaki gibidir.

SDO_GTYPE	NUMBER
SDO_SRID	NUMBER
SDO_POINT	SDO_POINT_TYPE
SDO_ELEM_INFO	SDO_ELEM_INFO_ARRAY
SDO_ORDINATES	SDO_ORDINATE_ARRAY

SDO_GTYPE : Geometrinin tipini belirten özelliktir. Geometry Object Model for the OGIS Simple Features for SQL spesifikasyonu ile uyumludur. DLLT formatındadır (D: Dimension(Boyut) L:Linear Referencing(Lineer Referans Sistemi) TT:Geometry Type(Geometri Tipi)). Geçerli SDO_GTYPE değerleri şunlardır aşağıdaki gibidir.

DL00:Bilinmeyen geometri

DL01:Nokta

DL02:Çizgi veya yay

DL03:Poligon veya yüzey

DL04:Koleksiyon

DL05:Çoklu nokta

DL06:Çoklu çizgi veya yay

DL07:Çoklu poligon veya yüzey

DL08:Katı cisim

DL09:Çoklu katı cisim

Örnek : 2003, iki boyutlu poligon, 3008, üç boyutlu katı cisim gibi.

SDO_SRID : Geometrinin ait olduğu koordinat sistemini belirten özelliktir (Spatial Reference System) Eğer bu değer NULL ise hiç bir koordinat sistemi ile ilişkilendirilmez, aksi halse SDO_COORD_REF_SYS tablosundaki SRID kolonundaki değerlerden birini almalıdır. Bir geometri sütunundaki tüm geometrilerin aynı SDO_SRID değerine sahip olması gerekir. Türkiye'nin koordinat sistemi 4327 no'lu kayıttır.

	SRID	COORD_REF_SYS_NAME	COORD_REF_SYS_KIND	COORD_SYS_ID
1	1	Sinusoidal (WGS 84)	PROJECTED	4400
2	2	Unit Sphere	GEOGRAPHIC2D	6405
3	2000	Anguilla 1957 / British West Indies Grid	PROJECTED	4400
4	2001	Antigua 1943 / British West Indies Grid	PROJECTED	4400
5	2002	Dominica 1945 / British West Indies Grid	PROJECTED	4400
6	2003	Grenada 1953 / British West Indies Grid	PROJECTED	4400
7	2004	Montserrat 58 / British West Indies Grid	PROJECTED	4400

Şekil 5: SDO_COORD_REF_SYS tablosundan bir görünüm

SDO_POINT : SDO_POINT_TYPE nesne tipi kullanılarak tanımlanır. X, Y, Z özniteliklerine sahiptir. Eğer SDO_ELEM_INFO ve SDO_ORDINATES dizileri null ise ve SDO_POINT özniteliği null değilse, verilen X, Y, Z değerleri bir nokta geometrinin koordinatları olarak düşünülür. Optimize edilmiş bir nesne tipi olduğundan, sadece noktalardan oluşan katmanlarda kullanılması tavsiye edilmektedir.

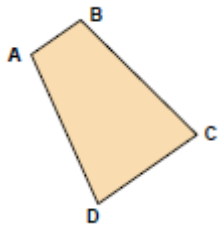
SDO_ELEM_INFO: Geometrinin ordinat değerlerinin nasıl yorunlanacağını belirtir. Değişken uzunluklu sayı dizisi olarak tanımlanır. Yapı: SDO_ELEM_INFO:

(1, Etype, Interpretation) Örnek: Aşağıdaki poligonun temsili:

SDO_ORDINATES: (Xa, Ya, Xb, Yb, Xc, Yc, Xd, Yd, Xa, Ya)

SDO_ELEM_INFO: (1, 2003, 1)

Burada 2003 Poligon, 1 ise Basit poligon olduğunu belirtir.



Şekil 6: Basit Poligon

SDO_ORDINATES: Bir geometrinin sınırlarını oluşturan koordinat değerlerini tutar. Değişken uzunluklu sayı dizisi olarak tanımlanır. Bu dizi daima SDO_ELEM_INFO dizisi ile birlikte kullanılmalıdır. Dizideki değerler boyuta göre sıralıdır.
Örnek: 4 köşeli ve 2 boyutlu poligonun temsili:
{X1,Y1, X2, Y2, X3, Y3, X4, Y4, X1, Y1}
Örnek: 4 köşeli ve 3 boyutlu poligonun temsili:
X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4, X1, Y1, Z1}

Tabloya dikdörtgen nesnesi eklemek için örnek SQL cümleciği

```
INSERT INTO cola_markets VALUES(
1,
'cola_a',
SDO_GEOMETRY(
2003, --İki-boyutlu poligon
NULL,--SRID
NULL,--SDO_POINT
SDO_ELEM_INFO_ARRAY(1,1003,3), --Bir dikdörtgen
SDO_ORDINATE_ARRAY(1,1, 5,7) --Kartezyen koordinatlarda dikdörtgen tanımlamak için iki köşe (sol alt ve sağ üst) yeterli
));
```

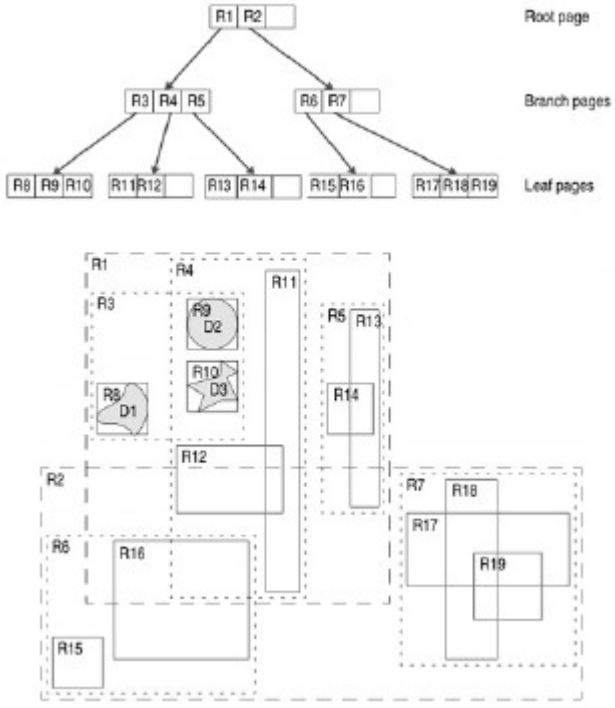
Mekansal İndeksleme, mekansal sorgu performansını arttırmak için kullanılır. Oracle Spatial'da R-Tree indeksleme kullanılır. 2D veri için Minimum Bounding Rectangles (MBRs) 3D veri için Minimum Bounding Volumes (MBVs) kullanılır. İndeksleme 2, 3 ve 4 boyut için yapılabilir.

R-Tree İndeksleme

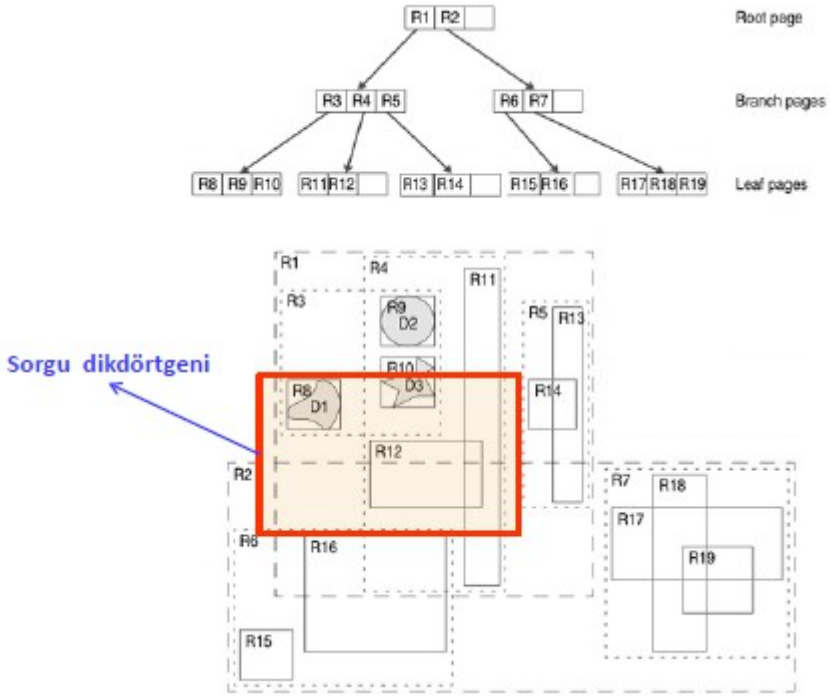
R-Tree: Mekansal erişim metotları için kullanılan ağaç veri yapısıdır. İsimdeki "R" Rectangle dan gelir. (Rectangle-Tree) 1984 yılında Antonin Guttman tarafından ortaya kondu. Bilimsel araştırmalarda ve gerçek hayatta önemli kullanım alanları buldu.

R-Tree İndeks Yapısı

Her geometrik nesne bir OBJECT_ID, Minimum Bounding Rectangle (MBR) ve öznitelik bilgileri ile temsil edilir. Bu şekilde uzay bir nesneler topluluğu olarak düşünülebilir. Bu temsil sadece iki boyut için geçerli değildir, n-boyuta genişletilebilir. Bir sorgu yeni bir dikdörtgen olarak temsil edilebilir. Burada sorgu, verilen sorgu dikdörtgeniyle çakışan tüm nesneleri bulma anlamına gelmektedir. R-Tree, bu şekilde istenen nesneleri hızlıca bulmada kullanılan bir mekansal indeksleme tekniğidir. Ana fikir ise birbirine yakın nesneleri gruplandırmak ve dalları bu gruplardan oluşan bir ağaç yapısı oluşturmaktır.



Şekil 7: R-Tree İndeks Yapısı



Şekil 8: R-Tree Sorguları

Arama ağacın kök nodundan başlar. Her nod bir dikdörtgen (MBR) kümesi ve alt nodlar için referans bilgisi tutar. Her yaprak, içerdiği geometrilere ait dikdörtgenleri (MBR) tutar. Bir nodun içindeki her bir dikdörtgenin sorgu dikdörtgeni ile çakışıp çakışmadığına bakılır. Eğer çakışma varsa, ilgili alt nod da aranır. Arama bu şekilde özyinelemeli (recursive) olarak ve tüm çakışan nodlar bulunana dek sürer. Bir yaprak noda ulaşılınca, içerilen dikdörtgenlerin sorgu dikdörtgeni ile çakışıp çakışmadığına bakılır; bunların içinde nesneler varsa bu nesneler sonuç kümesine eklenir [15].

3.7 WebGL

WebGL bir platform bağımsız, telifi ücretsiz web standardı, HTML5 Canvas Döküman Nesne Modeli arayüzleri ile ortaya çıkan OPENGL ES 2.0'a dayanan düşük seviyeli 3B grafik API'sidir. OPENGL ES 2.0'a aşina olanlar WebGL'i GLSL kullanan gölge tabanlı bir API olarak fark edeceklerdir. WebGL, OPENGL ES 2.0 tanımlamasına geliştiricilerin Javascript gibi yetersiz bellekle yönetilen diller için verilen tavizlerle beraber çok benzer.

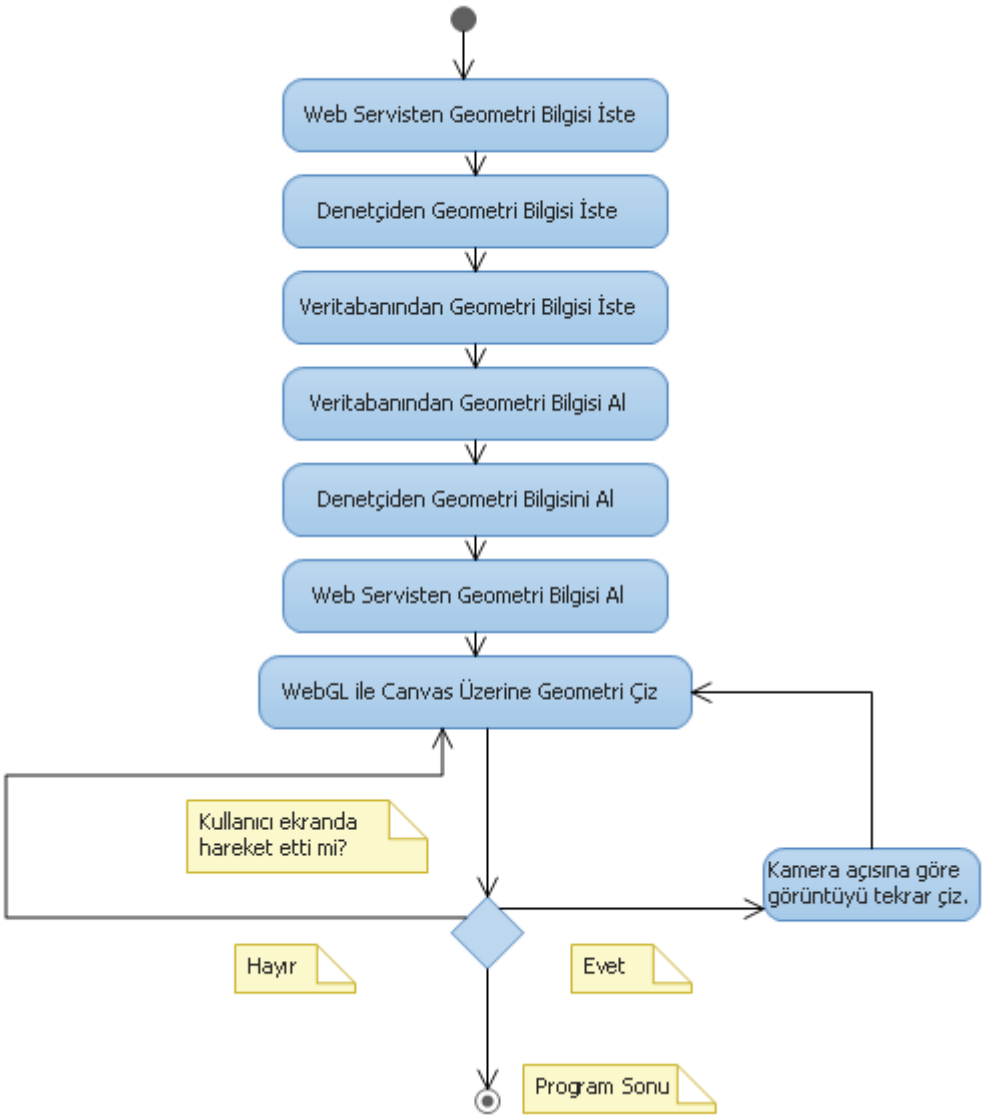
WebGL ile 3B uygulamalar eklentisiz olarak tarayıcıda görüntülenebilmektedir. Büyük tarayıcı satıcıları Apple(Safari), Google(Chrome), Mozilla(Firefox) ve Opera (Opera) WebGL çalışma grubunun üyeleridir [13].

4 SİSTEM ANALİZİ VE TASARIMI

Bu kısımda çalışmanın analizi, tasarımı, geliştirilmesi, testi konularına yer verilecektir.

4.1 Sistem Analizi ve Tasarımı

Sistemin aktivite diyagramı aşağıdaki gibidir.

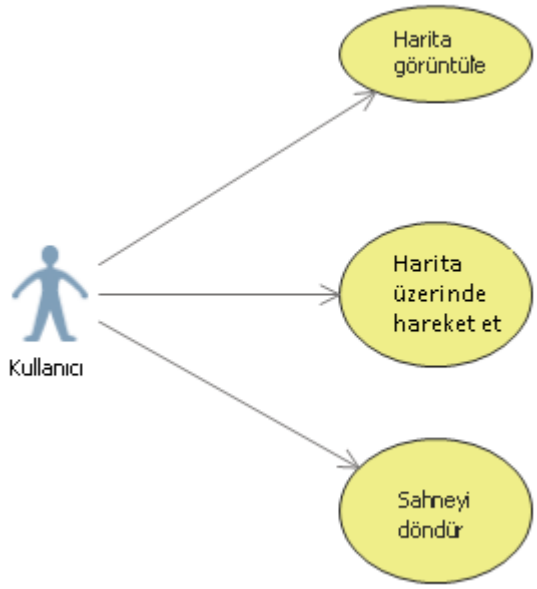


Şekil 9: Aktivite Diyagramı

Şekilden görüldüğü üzere HTML sayfa üzerinden javascript ile REST web servis çağrısı yapılır, web servis denetçiden bir metod çağrısı yaparak geometri

bilgisini ister, denetçi veritabanından sql ile geometri bilgisini ister, buradan alınan cevap denetçiye verilir, denetçi aldığı veriyi web servise iletir, web servis de cevabı JSON olarak javascript motoruna döndürür, alınan geometri bilgisi ve raster veri de WebGL ile HTML5 canvas üzerine çizdirilir.

Sistemin kullanım durumu diyagramı aşağıdaki gibidir. Kullanıcının uygulama üzerinde yapabileceği işlemler yukarıdaki kullanım durumunda belirtilmiştir. Buna göre kullanıcı haritayı görüntüleyebilir. Harita üzerinde hareket edebilir ve sahneyi döndürebilir. Bu durumlarda çizim işlemleri için WebGL tarafında otomatik olarak matris hesaplamaları yapılır ve yeni görünüm elde edilir.



Şekil 10: Kullanım Durumu Diyagramı

Oracle Database 11g2 Release 2 Installer Kurulumu

Bu kurulumda aşağıdaki adımlar izlenmiştir.

Configure Security Options seçeneğinde istenirse Oracle'dan güvenlik ile ilgili düzenli bülten almak için e-posta belirtilebilir. Aynı zamanda My Oracle Support ile güvenlik güncellemeleri alınmak istenirse ilgili seçenek işaretlenip My Oracle Support Password girilebilir. Installation Options'da Oracle'ın ilk sefer kurulacağını varsayarak Create and configure a database seçeneği seçilir.

System Class adımında Desktop Class seçilmesi bu çalışma için yeterlidir. Uygulamayı bir dizüstü veya masaüstü bilgisayara kurunca bu seçenek seçilir ve böylece bir başlangıç veritabanı oluşturulur ve daha az ayarlama yapma ihtiyacı gerektirir.

Typical Installation adımında Oracle Base, Software Location, Database file location seçenekleri için uygun klasörler seçilir. Database edition olarak Enterprise Edition seçilmelidir, Oracle Spatial sadece bu edisyonda seçilebilmektedir. Character Set seçeneğinde sadece Latin harfleri ile çalışılacaksa Default seçmek yeterlidir, aksi halde Unicode seçilmeli. Global database name yazılır. Oracle'daki yönetici işlemleri için Administrative Password ve Confirm Password alanları aynı olarak ve Oracle güvenli şifre politikasına uygun olarak girilir.

Summary ekranında yapılacak kurulumun özellikleri listelenir. Install Product adımında kurulum yapılır. Finish adımında ise kurulumun durumu hakkında bilgi verilir [16].

Oracle Client Installer Kurulumu

Bu kurulumda kurulu olan bir Oracle Server'a erişebilme için gerekli işlemler yapılmaktadır. Select Installation Type adımında Custom seçerek ayrı ayrı istediğimiz komponentlerin kurulmasını sağlanabilir. Select Product Languages adımında uygulamada kullanılacak diller seçilebilir.

Security Installation Location adımında Oracle yazılımı ve ayarlarla ilgili dosyaların yerleştirileceği yol belirtilir. Bu yola Oracle base directory denir. Oracle yazılım dosyalarını tutmak için de Software Location belirtilir. Perform Prerequisite Checks adımında varsa gerekli öngereksinimler gösterilir, bunları tamamladıktan sonra tekrar denetleme yaptırıp bu adım geçilebilir.

Available Product Components adımında bu çalışma için Oracle Database Utilities, Oracle Net, Oracle SQL Developer, Oracle Data Providers for .NET ve Oracle Providers for ASP.NET yeterlidir. Sonrasında Bitir tuşuna basıp kurulumun tamamlanması beklenir. Oracle SQL Developer için JDK 7 veya daha üstü gereklidir. Oracle SQL Developer'ı ilk çalıştırma sırasında Java Virtual Machine için java.exe'nin yolu sorulur. Bunu bir sefer girmek yeterli olacaktır. Oracle Spatial

geliştirme için bu çalışmada Oracle SQL Developer programı kullanıldı.

Oracle Spatial'ı kurmak windows işletim sisteminde kurmak için şu adımlar izlenir. Oracle'ın kurulu olduğu [Sürücü]\app\ [Kullanıcı_Adı]\product\11.2.0\dbhome_1\md\admin klasöründe command promt'da açılır. Sqlplus.exe çalıştırılır. Kullanıcı adı ve şifre girildikten sonra şu komutlar çalıştırılır.

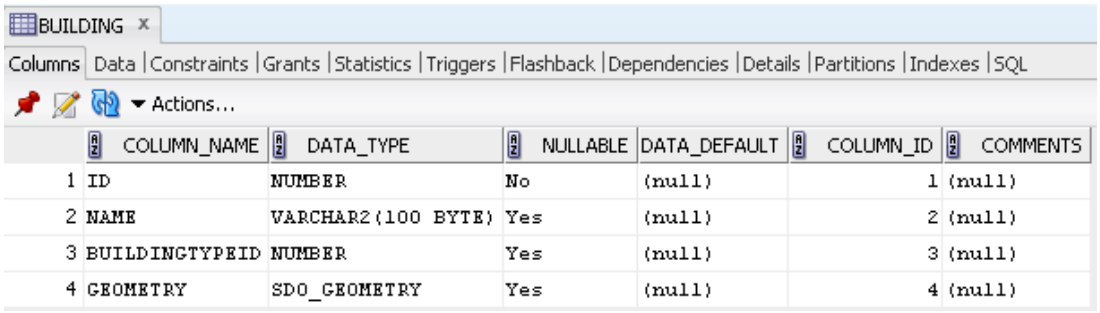
```
SQL> connect / as sysdba
```

```
SQL> create user mdsys identified by secret
 2  default tablespace sysaux
 3  account lock
 4  password expire;
SQL> @?/md/admin/mdprivs.sql
SQL> @?/md/admin/catmd
```

Spatial ile ilgili daha fazla bilgiye `$ORACLE_HOME/md/doc/README.txt` dosyasından ulaşılabilir.

```
SQL> select parameter, value
 2  from v$option
 3  where parameter like 'Spatial%';
PARAMETER            VALUE
-------------------- -----
Spatial              TRUE
```

Bina tablosunun veritabanı şeması aşağıdaki gibidir. Basit bir bina kaydında NUMBER tipinde benzersiz bir Id alanı, VARCHAR2(100) tipinde bina ismi, NUMBER tipinde bina tipi bilgisi ve MDSYS.SDO_GEOMETRY tipinde binanın şeklini belirten geometri tipinde bir veri bulunmaktadır.


BUILDING

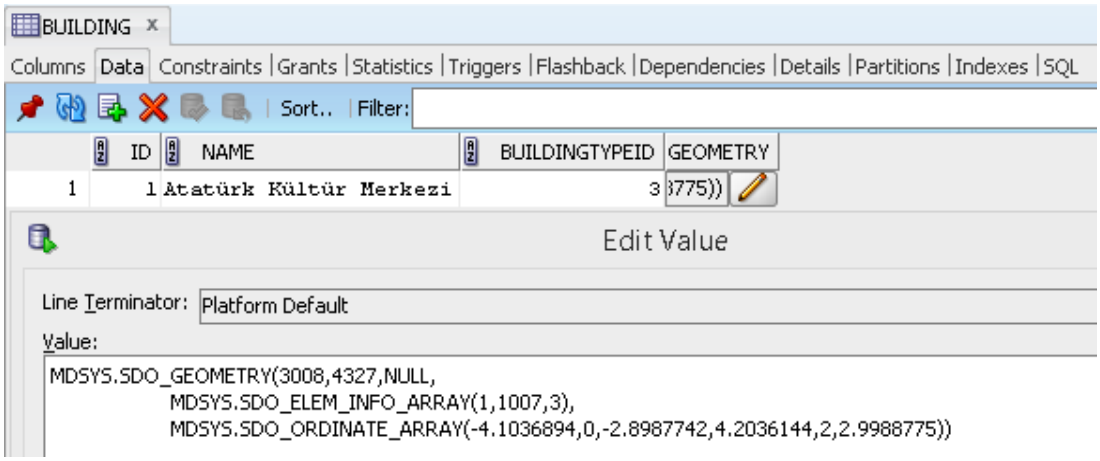
Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER	No	(null)	1	(null)
2	NAME	VARCHAR2(100 BYTE)	Yes	(null)	2	(null)
3	BUILDINGTYPEID	NUMBER	Yes	(null)	3	(null)
4	GEOMETRY	SDO_GEOMETRY	Yes	(null)	4	(null)

Şekil 11: Building Tablosu Şeması

İlk olarak bina tablosuna aşağıdaki gibi bir bina kaydı atılması gerekmektedir.


BUILDING

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	ID	NAME	BUILDINGTYPEID	GEOMETRY
1	1	Atatürk Kültür Merkezi	3	3775))

Edit Value

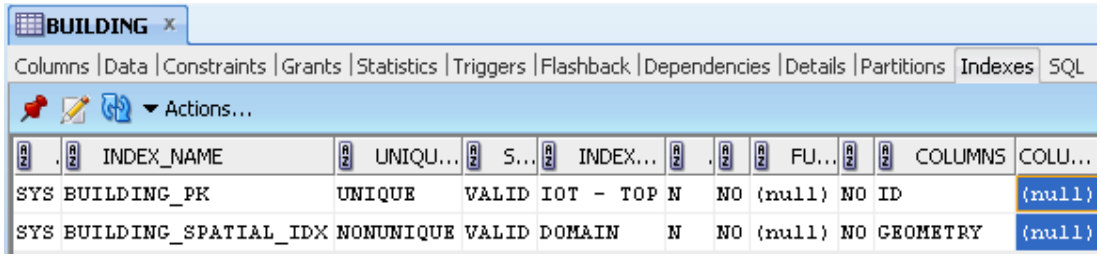
Line Terminator: Platform Default

Value:

```
MDSYS.SDO_GEOMETRY(3008,4327,NULL,
                   MDSYS.SDO_ELEM_INFO_ARRAY(1,1007,3),
                   MDSYS.SDO_ORDINATE_ARRAY(-4.1036894,0,-2.8987742,4.2036144,2,2.9988775))
```

Şekil 12: Building Tablosu Verisi

Sonrasında Oracle Spatial veritabanında bu tabloya IOT-TOP tipinde tekil bir birincil anahtar indeksi ve DOMAIN tipinde tekil olmayan bir Spatial indeks eklenir. Spatial indeks sayesinde bu sütun üzerinde uzaysal sorgulamalar yapılabilmektedir.


BUILDING

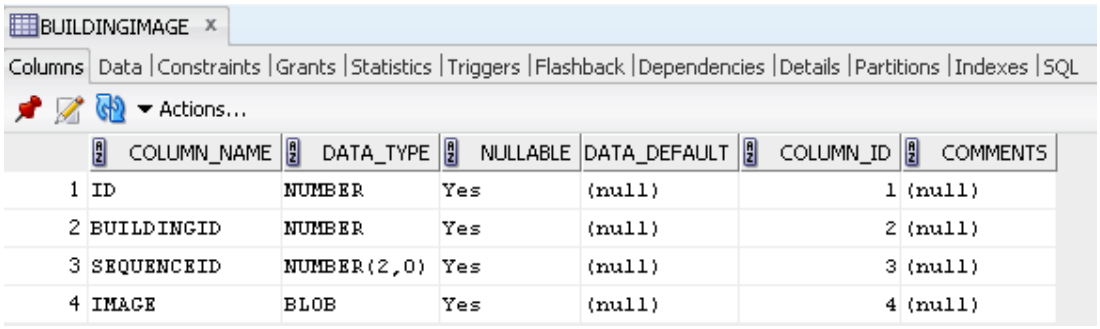
Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	INDEX_NAME	UNIQU...	S...	INDEX...	...	FU...	...	COLUMNS	COLU...
SYS	BUILDING_PK	UNIQUE	VALID	IOT - TOP	N	NO	(null)	NO	ID	(null)
SYS	BUILDING_SPATIAL_IDX	NONUNIQUE	VALID	DOMAIN	N	NO	(null)	NO	GEOMETRY	(null)

Şekil 13: Building Tablosu indeksleri

BuildingImage tablosu veritabanı şeması aşağıdaki gibidir. Bu tabloda benzersiz NUMBER tipinde bir Id sütunu, NUMBER tipinde Building tablosu ile ilişki kuran BuildingId alanı, NUMBER(2,0) tipinde binanın kaçıncı sıradaki açıdan görüntü olduğunu belirten SequenceId sütunu ve BLOB tipinde bina resmini tutan Image sütunu bulunmaktadır.


BUILDINGIMAGE

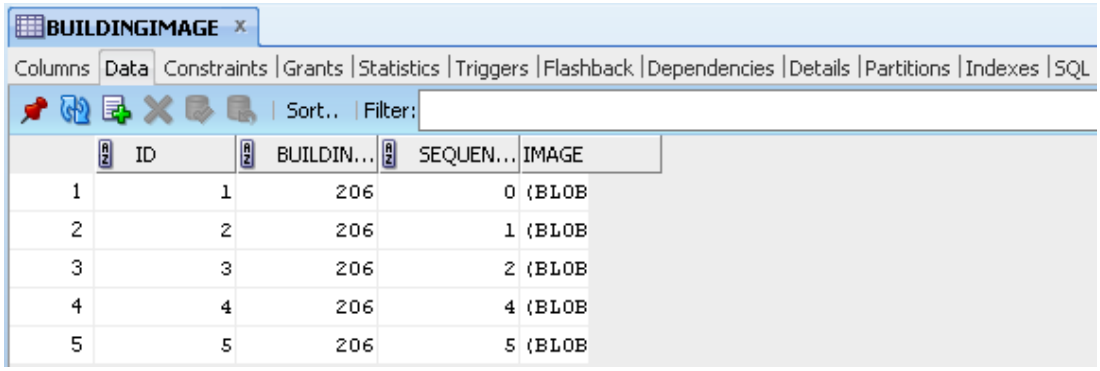
Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER	Yes	(null)	1	(null)
2	BUILDINGID	NUMBER	Yes	(null)	2	(null)
3	SEQUENCEID	NUMBER(2,0)	Yes	(null)	3	(null)
4	IMAGE	BLOB	Yes	(null)	4	(null)

Şekil 14: BuildingImage Tablosu Şeması

Aşağıdaki şekilde BuildingImage tablosundaki veriler görünmektedir.


BUILDINGIMAGE

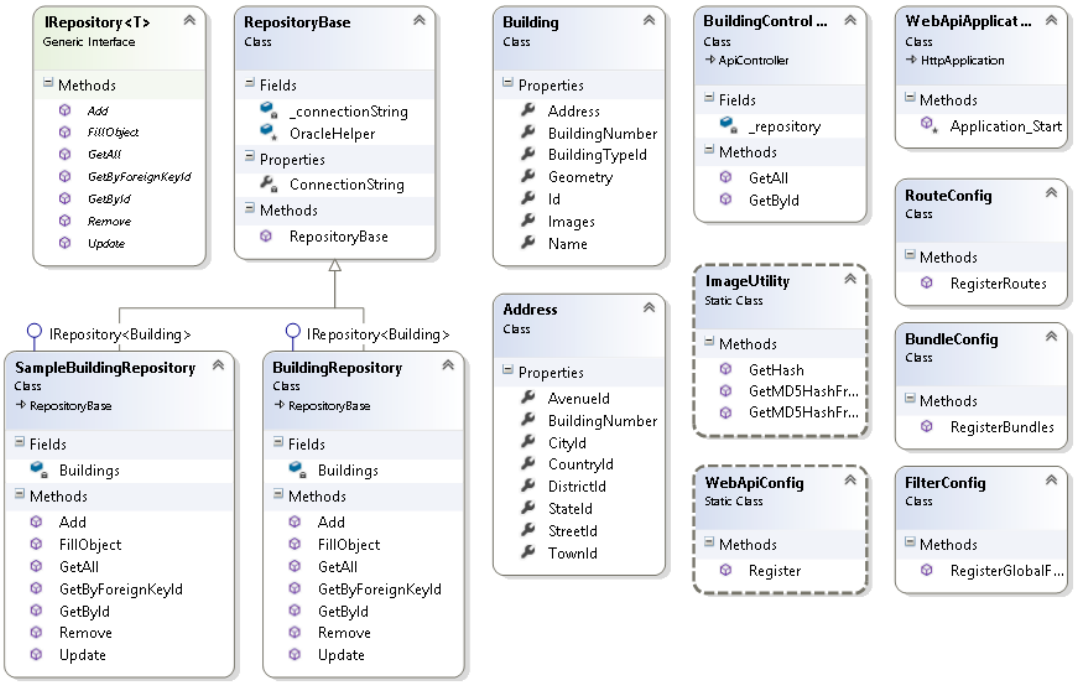
Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	ID	BUILDIN...	SEQUEN...	IMAGE
1	1	206	0	(BLOB
2	2	206	1	(BLOB
3	3	206	2	(BLOB
4	4	206	4	(BLOB
5	5	206	5	(BLOB

Şekil 15: BuildingImage Tablosu Verisi

NetSdoGeometry projesi sınıf diyagramı aşağıdaki gibidir.



Şekil 16 : NetSdoGeometry projesi Sınıf Diyagramı

OracleCustomTypeBase sınıfı bağlantı sağlamak için kullanılabilmektedir. SdoGeometry ve SdoPoint sınıfları bu sınıftan türemiştir. SdoGeometry sınıfı Oracle Spatial veritabanında tutulan uzaysal verinin sınıf yapısı halindedir. SdoPoint sınıfı Oracle Spatial'da nokta yapısındaki veriyi işleyebilmek için kullanılan sınıftır. OracleArrayTypeFactoryBase sınıfı MDSYS.SDO_ELEM_INFO_ARRAY Oracle Spatial veri tipini temsil eden ElemArrayFactory sınıfına ve MDSYS.SDO_ORDINATE_ARRAY Oracle Spatial veri tipini temsil eden OrdinatesArrayFactory sınıfına temel olmaktadır. OracleObjectColumns sınıfı ise .Net verilerini Oracle Spatial veri tiplerine eşlemek için kullanılabilir.Aşağıda bu sınıftaki class, metod, property, alan ve enum'ların açıklaması vardır.

NetSdoGeometry Assembly
OracleHelper Class

OracleHelper	Veritabanı ile ilgili işlemlerin yapıldığı yardımcı sınıftır.

Fields

strConnectionString	Bağlantı satırı
oConnection	Oracle veritabanına bağlantı nesnesi
oCommand	Oracle veritabanı üzerinde işlem yapan komut nesnesi

iTimeOut	Zaman aşımı süresi (saniye cinsinden)

Methods

#ctor	Yapıcı metod, bağlantı satırı web.config'den alınır ve komut nesnesi bağlantı için hazır hale getirilir
#ctor(System.String)	Yapıcı metod, bağlantı satırı parametreden alınır ve komut nesnesi bağlantı için hazır hale getirilir
Dispose	Kullanımına gerek kalmayan bağlantı kapatılır ve bellekten atılır, komut nesnesi bellekten atılır.
CloseConnection	Bağlantı kapalı değilse kapatılır.
ExecuteScalarByCommand(System.String)	Sayısal bir değer dönen komut çalıştırır
ExecuteNonQueryByCommand(System.String)	Sonuç dönmeyen bir komut çalıştırır
ExecuteNonQueryBySQL(System.String)	Sayısal değer dönmeyen sql çalıştırır
GetDatasetByCommand(System.String)	Sp ile bir veri kümesi döndürür
GetDatasetBySQL(System.String)	Sql ile bir veri kümesi döndürür
GetReaderBySQL(System.String)	Sql ile bir OracleDataReader nesnesi döndürür
GetReaderBySQLGeo(System.String)	Sql ile MDSYS.SDO_GEOMETRY tipinde parametre alır ve OracleDataReader tipinde veri döner
GetReaderByCmd(System.String)	SP komutu ile OracleDataReader nesnesi döner
Client.OracleDbType)	Komut nesnesine parametre ekler
OracleDbType,System.Int32)	Komut nesnesine belirtilen büyüklükte parametre ekler
String,System.Object)	Belirtilen komut parametresine değer atar

ExpectedType Enum

ExpectedType	Veritabanından gelmesi beklenen tip

Fields

StringType	Yazı tipi
NumberType	Sayı tipi
DateType	Tarih tipi
BooleanType	Evet/hayır tipi
ImageType	Görüntü tipi

OracleArrayTypeFactoryBase Class

OracleArrayTypeFactoryBase	Oracle Spatial veritabanındaki dizileri ifade

	etmek için temel sınıf olarak kullanılır.
Methods	
CreateArray(System.Int32)	Belirtilen sayıda öğeye sahip dizin oluşturur
CreateStatusArray(System.Int32)	Belirtilen sayıda öğeye sahip OracleUdtStatus tipinde dizin oluşturur

OracleCustomTypeBase Class

OracleCustomTypeBase	Geometri tipleri için temel sınıf
Fields	
errorMessageHead	Hata mesajı
connection	Bağlantı nesnesi
pUdt	Udt nesnesi için göstergeç
isNull	Null kontrolü alanı
Methods	
CreateObject	Yeni bir nesne oluşturup döner
SetConnectionAndPointer((OracleConnection, System.IntPtr)	Bağlantı ve göstergeci ayarlar
MapFromCustomObject	.Net sınıfı veri yapısından Oracle Spatial veritipine eşlemeyi sağlar.
MapToCustomObject	Oracle Spatial veri yapısından .Net sınıfı veritipine eşlemeyi sağlar.
`FromCustomObject(` OracleConnection,System.IntPtr)	Özel bir nesneden eşlemeyi sağlar
`ToCustomObject(` OracleConnection,System.IntPtr)	Özel bir nesneye eşlemeyi sağlar
String,System.Object)	Belirli bir hücreye sütun adına göre Oracle değeri atamayı sağlar
Int32,System.Object)	Belirli bir hücreye sütun sırasına göre Oracle değeri atamayı sağlar
GetValue<U>(System.String)	Belirli bir Oracle verisini sütun adına göre almayı sağlar
GetValue<U>(System.Int32)	Belirli bir Oracle verisini sütun sırasına göre almayı sağlar
Properties	
IsNull	Null kontrolü property'si
Null	Nesne için NULL değer dönen property

SdoGeometry Class

SdoGeometry	MDSYS.SDO_GEOMETRY Oracle Spatial veri tipinin karşılığı olan .Net sınıfı

Methods

MapFromCustomObject	Özel bir nesneden veri eşlemesini sağlar
MapToCustomObject	Özel bir nesneye veri eşlemesini sağlar

Fields

Dimensionality	Boyut bilgisi
LRS	Lineer Referans Sistemi
GeometryType	Geometri tipi

Methods

PropertiesFromGTYPE	GTYPE'dan özellikleri okumak için kullanılır
PropertiesToGTYPE	Özelliklerden GTYPE verisi oluşturmak için kullanılır.
ToString	Bu nesnenin özelliklerini göstermek için kullanılır

Properties

Sdo_gtype	Boyut bilgisi
Sdo_srid	Koordinat sistemi belirtmek için kullanılır.
Sdo_point	Nokta bilgisi
ElemArray	Öğe dizini
OrdinatesArray	Ordinat dizisi
AsText	Bu nesnenin özelliklerini göstermek için kullanılır

OracleObjectColumns Class

OracleObjectColumns	Oracle Spatial veri hücresinin sütunları

ElemArrayFactory Class

ElemArrayFactory	MDSYS.SDO_ELEM_INFO_ARRAY veri tipine karşılık gelen property

OrdinatesArrayFactory Class

OrdinatesArrayFactory	MDSYS.SDO_ORDINATE_ARRAY veri tipine karşılık gelen property

SdoGeometryTypes Class

SdoGeometryTypes	Oracle Spatial verisi tanımlarken kullanılan alt tipleri belirtir

ETYPE_SIMPLE Enum

ETYPE_SIMPLE	Basit geometri tiplerini belirlemek için kullanılır

Fields

POINT	Nokta
LINE	Çizgi
POLYGON	Poligon
POLYGON_EXTERIOR	Poligon dışı
POLYGON_INTERIOR	Poligon içi

ETYPE_COMPOUND Enum

ETYPE_COMPOUND	İle seviye geometri tiplerini belirlemek için kullanılır

Fields

FOURDIGIT	4 basamak
POLYGON_EXTERIOR	Poligon dışı
POLYGON_INTERIOR	Poligon içi

GTYPE Enum

GTYPE	Basit geometri tiplerini belirlemek için kullanılır

Fields

UNKNOWN_GEOMETRY	Bilinmeyen geometri
POINT	Nokta
LINE	Çizgi
CURVE	Yay
POLYGON	Poligon
COLLECTION	Koleksiyon
MULTIPOINT	Çoklu nokta
MULTILINE	Çoklu çizgi
MULTICURVE	Çoklu yay
MULTIPOLYGON	Çoklu poligon

DIMENSION Enum

DIMENSION	Geometrinin kaç boyutlu olduğunu belirtir

Fields

DIM2D	2 boyutlu
DIM3D	3 boyutlu
LRS_DIM3	3 boyutlu LRS
LRS_DIM4	4 boyutlu LRS

SdoPoint Class

SdoPoint	Oracle Spatial veritabanındaki Sdo_point tipini .Net tarafında kullanabilmeyi sağlar.

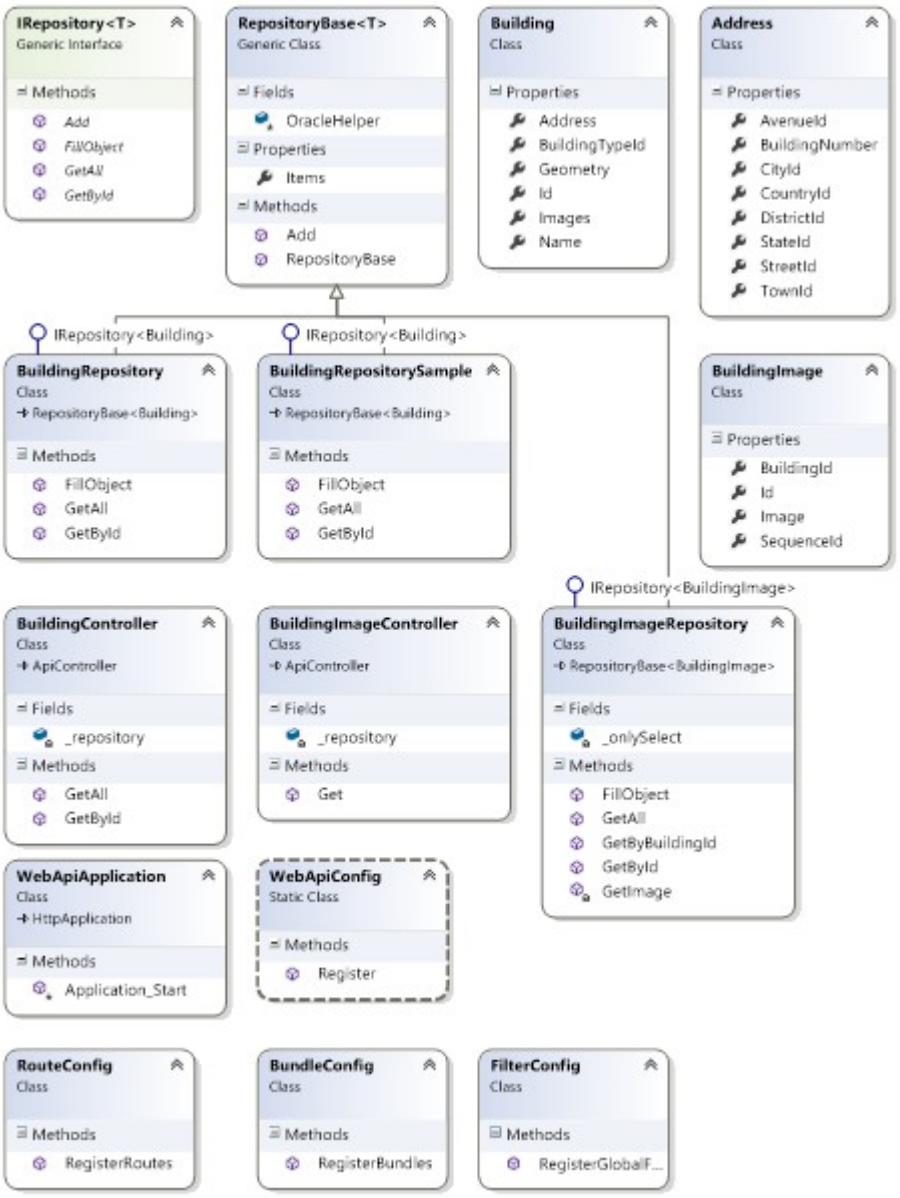
Methods

MapFromCustomObject	.Net SdoPoint sınıfı veri yapısından Oracle Spatial sdo_point veritipine eşlemeyi sağlar.
MapToCustomObject	Oracle Spatial sdo_point veri yapısından .Net SdoPoint sınıfına eşlemeyi sağlar.

Properties

X	X koornidatı
Y	Y koordinatı
Z	Z koordinatı

UrbanGeneration projesi sınıf diyagramı aşağıdaki gibidir.



Şekil 16: UrbanGeneration projesi sınıf diyagramı

IRepository arayüzü veritabanı işlemlerini yapabilmek için jenerik bir arayüzdür. RepositoryBase sınıfı veritabanı işlemleri yapabilmek için kullanılan jenerik bir temel sınıftır. BuildingRepository, BuildingRepositorySample ve BuildingImageRepository sınıfları RepositoryBase sınıfından türemiştir ve IRepository arayüzünün metodlarını uygulamaktadır. Building, Address, BuildingImage sınıfları veri tutma ve katmanlar arası veri taşıma işlemi yapmaktadır. BuildingController ve BuildingImageController sınıfları Repository sınıflarını çağırır ve oluşan sonuçları REST Web servisleri ile HTML katmanına iletir. Aşağıda bu sınıftaki class, metod, property, alan ve enum'ların açıklaması vardır.

UrbanRegeneration Assembly

BundleConfig Class

BundleConfig	Javascript dosyalarını birleştirip kullanmayı sağlar

Methods

Optimization.BundleCollection)	For more information on Bundling, visit http://go.microsoft.com/fwlink/?LinkId=254725

FilterConfig Class

FilterConfig

Methods

GlobalFilterCollection

RouteConfig Class

RouteConfig	İstek yönlendirme ayarlamalarını yapar

Methods

RouteCollection	Register Routes

WebApiConfig Class

WebApiConfig	Web API ayarlamalarını yapar

Methods

HttpConfiguration	Web API ayarlamalarını yapar

BuildingController Class

BuildingController	APIController'dan türeyen bu sınıf bina denetçisi olarak REST çağrılarını karşılamaktadır.

Fields

_repository	Sınıf verilerini getirmede kullanılır

Methods

GetAll	Tüm bina verilerini getirir
GetById(System.Int32)	Belirtilen id'ye göre bina verisini getirir

BuildingImageController Class

BuildingImageController	Bina resmi sınıfı

Fields

_repository	Bina resim veri işlemlerini yapan alan

Methods

Get	Tüm bina resimlerini getirir.

BuildingImage Class

BuildingImage	Bina resim bilgisi
Properties	
Id	Benzersiz numara
BuildingId	Bina no
SequenceId	Sıra numarası
Image	Resim yolu

BuildingImageRepository Class

BuildingImageRepository	Bina resim veritabanı işlemlerini yapar

RepositoryBase Class

RepositoryBase	Veritabanı işlemleri yapan sınıflar için temel sınıf
Fields	
OracleHelper	Oracle yardımcı sınıf property'si
Methods	
#ctor	Yapıcı metod, oracle yardımcı sınıfının numunesini oluşturur
Add	Listeye öğe ekle
Properties	
Items	Öğe listesi

IRepository Class

IRepository	Veri katmanı arayüzü
Methods	
GetAll	Tüm verileri getir
GetById(System.Int32)	Id'ye göre veri getir
IRepository.Add(`0)	Ekleme işlemi yapar
FillObject(System.String)	Verilen sql'e göre nesnelerin özelliklerini doldurur
Fields	
_onlySelect	Select cümlesi
Methods	
GetAll	Tüm bina resimlerini getirir
GetById(System.Int32)	Id'ye göre belirli bir bina resmini getirir
GetByBuildingId(System.Int32)	Bir bina Id'ye göre bina resimlerini getirir
FillObject(System.String)	Verilen sql'e göre nesne özelliklerini doldurur
BuildingImage,System.Byte[])	Bina nesnesinden resim yolunu alır.

BuildingRepositorySample Class

BuildingRepositorySample	Veritabanı bağımsız statik test repository'si

Methods

GetAll	Test verisi getir
GetById(System.Int32)	Id'ye göre test verisi getir
FillObject(System.String)	Uygulanmadı

WebApiApplication Class

WebApiApplication	Web Api Application

Methods

Application_Start	Application Start işlemleri

Address Class

Address	Adres bilgisi

Properties

CountryId	Ülke no
StateId	Ülke/eyalet no
CityId	Şehir no
TownId	İlçe no
DistrictId	Bölge no
AvenueId	Bulvar/cadde no
StreetId	Sokak no
BuildingNumber	Bina no

Building Class

Building	Bina

Properties

Id	Benzersiz bina numarası
Name	Bina ismi
BuildingTypeId	Bina tipi
Address	Adres bilgisi
Geometry	Geometri bilgisi
Images	Resimler

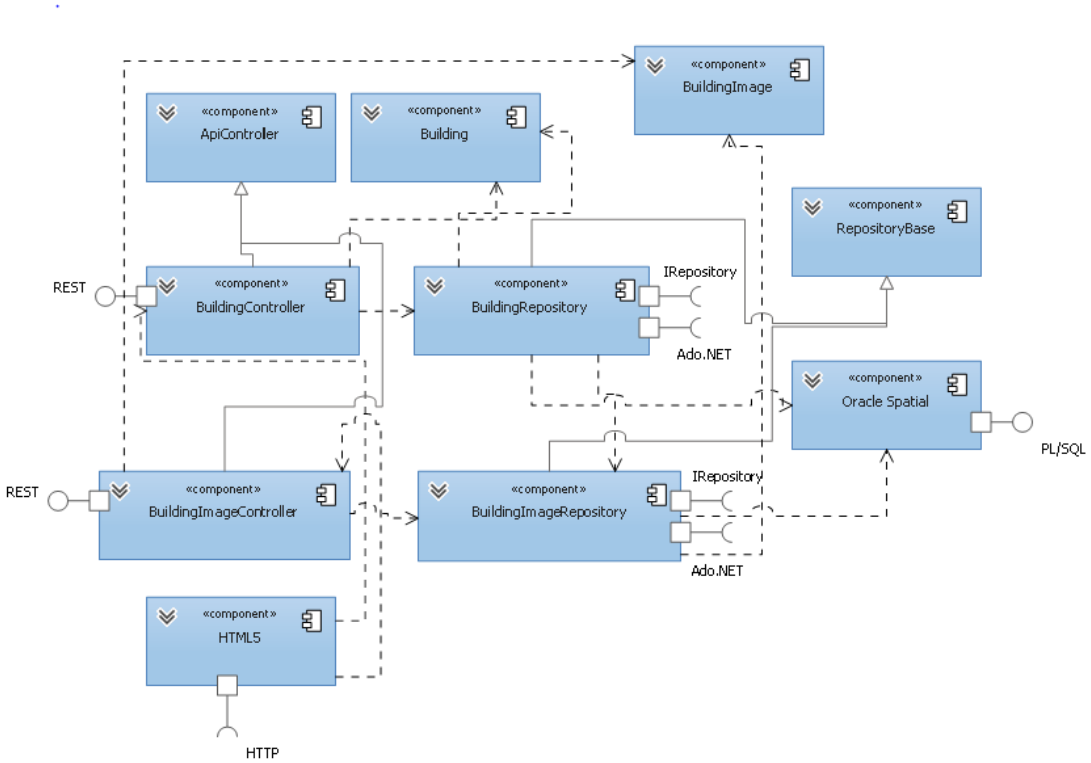
BuildingRepository Class

BuildingRepository	Bina nesnesi için veritabanı işlemleri yapar

Methods

GetAll	Tüm bina listesini getirir
GetById(System.Int32)	Belirli bir binayı Id bilgisine göre getirir
FillObject(System.String)	Verilen sql'e göre nesnelerin özelliklerini doldurur

Sistemin komponent diyagramı aşağıdaki gibidir. Burada kesikli çizgiler bağımlılığı ifade eder, yani bir öğenin başka bir elemana bağlı olduğunu belirtir. Düz çizgisi olup ucu üçgen olan çizgiler genellemeyi ifade eder, yani bir öğenin başka bir öğeden özellik ve davranış miras aldığını belirtir. Ucu hilal şeklinde olan çizgi de gerekli bağımlılığı ifade eder, yani başka öğelerde belirtilen metodlara ve servislere ulaşmak için bir giriş belirtir. HTML5 sayfası HTTP arayüzü ile BuildingController denetçisinin sunduğu REST arayüzünü kullanır ve bu şekilde JSON veri alır. BuildingController ve BuildingImageController komponentleri ApiController komponentine bağımlıdır, bu sınıf sayesinde REST web servisleri sunabilme özelliği kazanırlar. BuildingController sınıfı BuildingRepository sınıfına, BuildingImageController sınıfı da BuildingImageRepository sınıfına bağımlıdır, RepositoryBase'den türeyen bu sınıflar veritabanı işlemleri için ADO.NET arayüzü ile Oracle Spatial komponent'inin sunduğu PL/SQL arayüzü ile işlem yaparlar. BuildingController ve BuildingRepository sınıfları Building sınıfına, BuildingImageController ve BuildingImageRepository sınıfları da BuildingImage sınıflarına bağımlıdır [17].

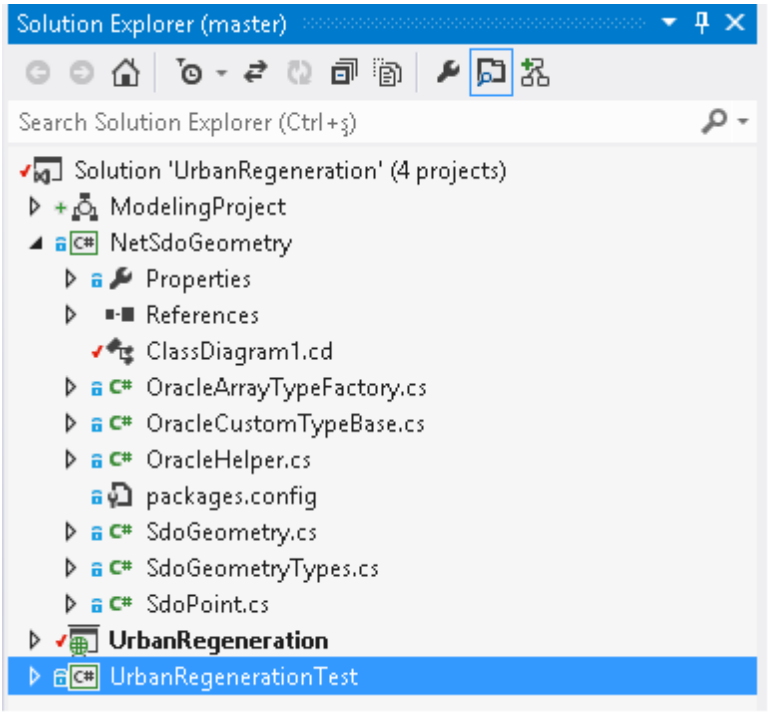


Şekil 17: Komponent Diyagramı

4.2 Geliştirme

Geliştirme ortamı olarak Visual Studio 2012 ve Oracle SQL Developer kullanıldı. Visual Studio ile ASP.NET MVC yapısı üzerine web sitesi tasarlandı. ASP.NET Web API ile REST web servislerinin kullanımı mümkün oldu. Web Sitesi diyagramlarda görüldüğü üzere katmanlı bir mimaride kurgulandı. Oracle Spatial ile veri tabanı uzaysal veri yapısı işlemleri yapıldı, diğer verisel bilgiler de Oracle veritabanı üzerine konumlandırıldı.

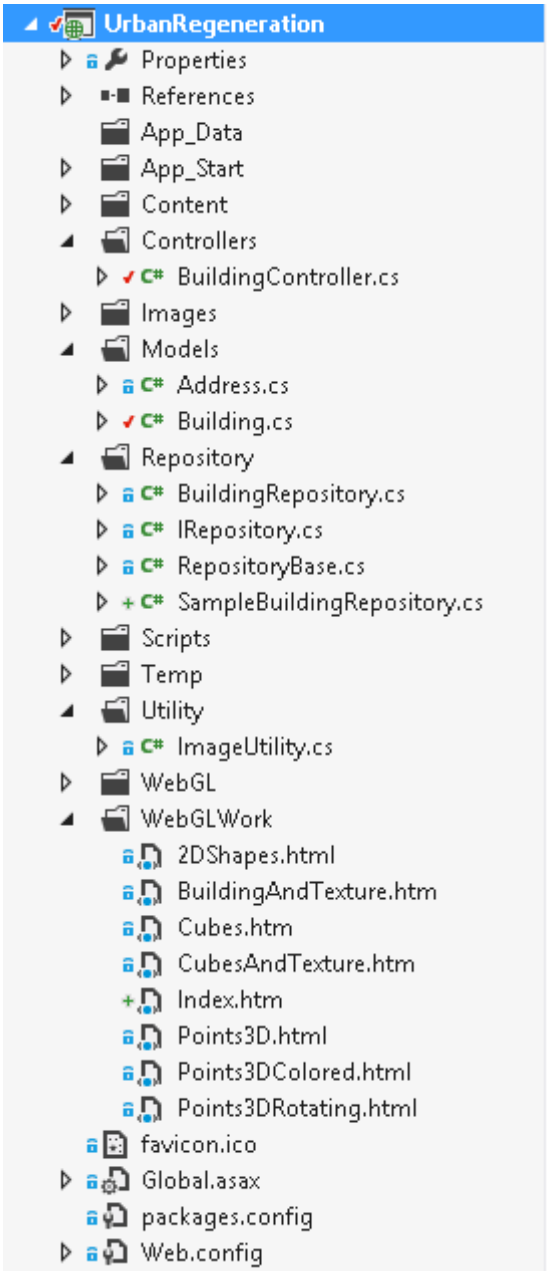
UrbanGeneration çözümü katmanlı bir yapıda yazılmıştır. Şekil 7'de görüldüğü üzere NetSdoGeometry sınıf kütüphanesi uygulamanın çalışması için gerekli temel sınıfları içermektedir.



Şekil 18: NetSdoGeometry Sınıf kütüphanesi

HTML dosya, javascript dosyalar, resimler, denetçiler, model sınıfları ve veri tutan sınıflar UrbanGeneration ASP.NET Web API projesinde tutulmaktadır.

Models klasöründe bulunan Address ve Building sınıfları sadece veri barındırma işlemi yapmaktadır, yani sadece property'leri olan sınıflardır. Veritabanından veri alınırken ve veritabanına yazılırken bu veri taşıyan sınıflar kullanılır.



Şekil 19: UrbanRegeneration Projesi

Repository klasöründeki jenerik IRepository arayüzü ASP.NET Web API'da REST servisi açabilmemizi sağlamaktadır. Bunun içinde uzaysal veri işlemek için gerekli metod imzaları bulunmaktadır. RepositoryBase sınıfında bağlantı cümlesi property'si ve bağlantı kurmak için bir property bulunmaktadır. BuildingRepository sınıfı RepositoryBase sınıfından türemiştir ve IRepository arayüzündeki metodları veritabanı işlemlerini yaparak uygulamaktadır.

Controllers klasöründeki BuildingController sınıfı REST servisi sunabilmemiz için APIController sınıfından türetilmiştir. İçinde IRepository tipinde bir alan ve bu

alanı kullanan veri işlemi yapıp Building sınıfını JSON veri olarak dönen bir metod vardır.

Oracle'a erişebilmek için Oracle'ın kurulu olduğu [Sürücü]:\app\[Kullanıcı_Adı]\product\11.2.0\client_1\odp.net\bin\4\Oracle.DataAccess.dll dosyası NetSdoGeometry, UrbanGeneration ve UrbanGenerationTest sınıflarına referans olarak eklendi, aksi halde uyumsuz bir dll'den dolayı Oracle'a erişimde sorunlar meydana gelmektedir.

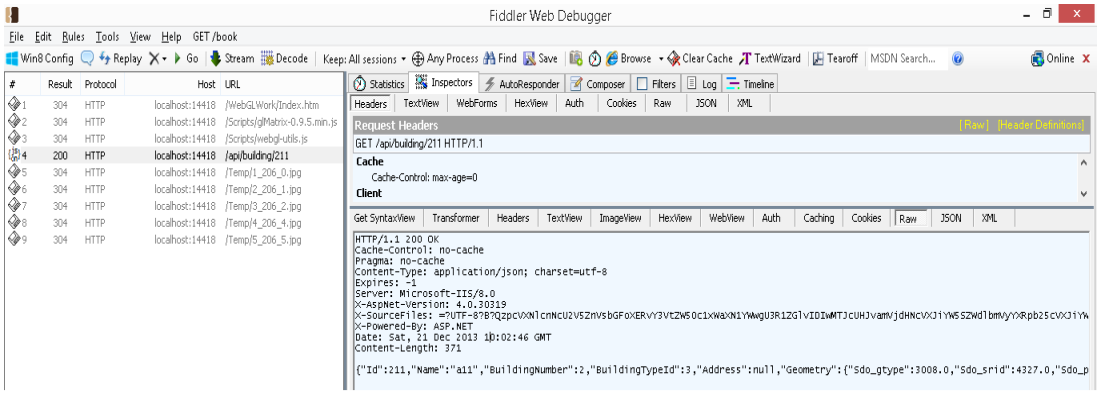
Web.config dosyasında `<add name="OracleConnectionString" connectionString="DATA SOURCE=ORCL;PERSIST SECURITY INFO=True;USER ID=spatial;PASSWORD=spatial" />` Oracle bağlantı cümlesi bulunmaktadır. Packages.config dosyasında da bu projede kullanılan Framework'lerin bir listesi bulunmaktadır.

Uygulamanın görsel olan kısmı ise WebGLWork klasöründeki Index.htm dosyasıdır. Bu dosyada HTML kontrol olarak önemli olan HTML5 ile gelen bir Canvas kontrolü bulunmaktadır.

Bu dosyanın body tag'ında onload event'inde webGLStart metodu çağrılır. Bu metodda javascript ile HTML kısmıdaki canvas kontrolü elde edilir. Bu canvas kontrolü ile eğer browser WebGL'i destekliyorsa bir WebGL context'i oluşturulur, değilse uygun bir hata mesajı verilir. Sonrasında initShaders metodu çağrılır. Bu metod ile gölgeleme için gerekli köşe, desen ve matris bilgileri ayarlanır. Sonrasında çağrılan initBuffers metodu ile yüzeyin olduğu düzlem belirtilir.

Bundan sonra ise onload metodunda initWorldObjects fonksiyonu çağrılır. Bu fonksiyon içinde aşağıdaki çağrı yapılır ve aşağıdaki sonuç alınır, ancak bu sonuç gerçekte fiddler ile bakıldığında XML yerine JSON veri olarak dönülür. ASP.NET Web API ile REST çağrısının sonucunun XML, JSON, HTML veya başka özel bir biçimde verilebilmesi sağlanabilmektedir.

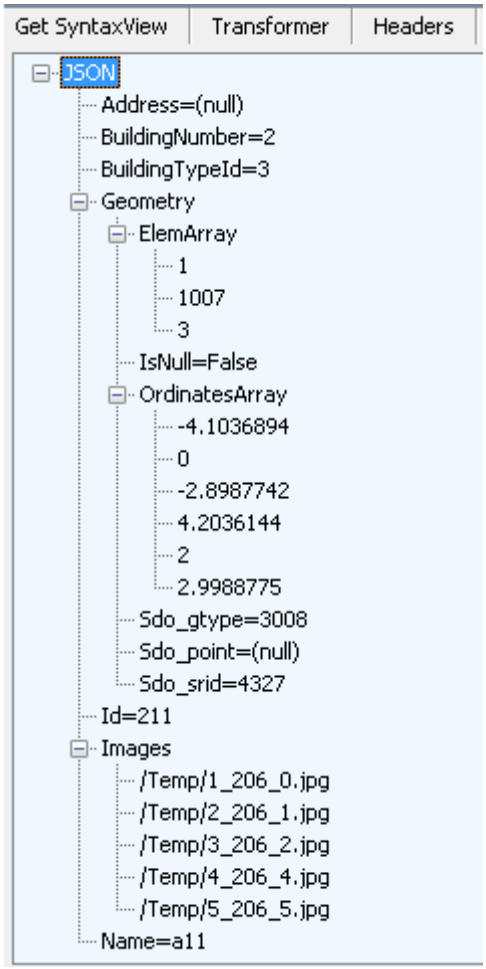
Şekil 20: REST Çağrısı XML Sonucu



Şekil 21: REST Çağrısı Fiddler Raw JSON görünümü

Aşağıda yukarıdaki yazıları açıkça görebilirsiniz.

HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.0
X-AspNet-Version: 4.0.30319
X-SourceFiles: =?UTF-8?B?
QzpcVXNlcnNcU2V5ZnVsbGFoXERvY3VtZW50c1xWaXN1YWwgU3R1ZGlvID
IwMTJcUHJvamVjdHNcVXJiYW5SZWdlbmVyYXRpb25cVXJiYW5SZWdlbmVy
YXRpb25cYXBpXGJ1aWxkaW5nXDIxMQ==?=
X-Powered-By: ASP.NET
Date: Sat, 21 Dec 2013 10:02:46 GMT
Content-Length: 371
{"Id":211,"Name":"a11","BuildingNumber":2,"BuildingTypeId":3,"Address":null,"
Geometry":{"Sdo_gtype":3008.0,"Sdo_srid":4327.0,"Sdo_point":null,"ElemArray":
[1.0,1007.0,3.0],"OrdinatesArray":[-4.10368940,0.0,-
2.89877420,4.20361440,2.0,2.99887750],"IsNull":false},"Images":
["/Temp/1_206_0.jpg","/Temp/2_206_1.jpg","/Temp/3_206_2.jpg","/Temp/4_206_4.
jpg","/Temp/5_206_5.jpg"]}

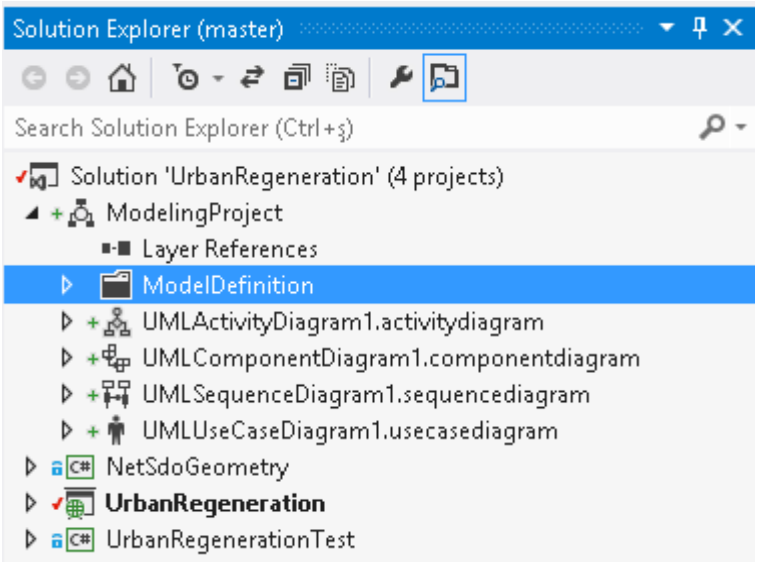


Şekil 22: REST Çağrısı Fiddler JSON görünümü

Aynı initWorldObjects fonksiyonu içinde alınan JSON bilgileri işlenir. Bu JSON bilgileri içinde görüldüğü üzere adres, bina numarası, bina tipi, geometri bilgisi, id bilgisi, binanın çevresini gösteren resim bilgileri ve isim bulunmaktadır. Bu bilgiler ile sahne çizimi gerçekleştirilir.

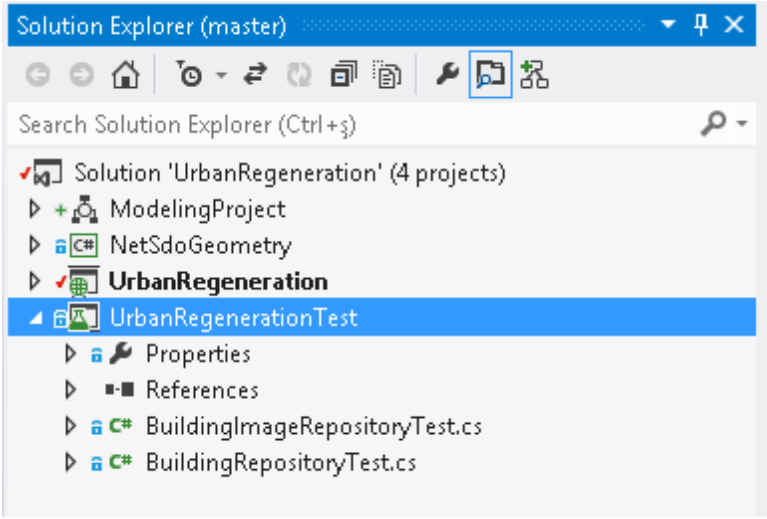
WebGLStart fonksiyonunda klavye tuşlarına basılmasına ve fare hareketlerine göre sahne gezinme imkanı sağlanmaktadır. En sonundaki tick metodu ise ekranı sürekli tazelemek ile görevlidir.

Aşağıda ModellingProject projesi bulunmaktadır. Her bir dosyada adı geçen ve bu çalışmada da var olan UML şemaları bulunmaktadır. Visual Studio 2012 kendi içinde UML şemaları tasarlamaya imkan sunmaktadır.



Şekil 23: ModellingProject Projesi

Aşağıdaki şekilde UrbanRegenerationTest projesi bulunmaktadır. Burada dosyalar BuildingController ve BuildingImageController dosyalarındaki Get metodlarını test etmek için birim testi yöntemi kullanılmaktadır.



Şekil 24: UrbanRegenerationTest projesi

UrbanRegenerationTest Assembly

BuildingImageRepositoryTest Class

BuildingImageRepositoryTest BuildingImageRepository sınıfı için Test sınıfı

Methods

Get Get metodu testi

BuildingRepositoryTest Class

BuildingRepositoryTest BuildingRepository Test sınıfı

Methods

Get	Get metodu testi

4.3 WebGL ile Kodlama

WebGL'i windows bir makinede çalıştırmak için Microsoft sitesinden ücretsiz olarak indirilebilen Microsoft DirectX runtime programının kurulu olması gerekmektedir. Grafik kartının son sürücü güncellemesinin olduğu kontrol edilmelidir.

Tarayıcı seçimi

İnternet Explorer 11'de Web GL sınırlı olarak desteklenmektedir.

Firefox için versiyon 4 veya üstü olmalıdır.

Chrome için versiyon 10 veya üstü olmalıdır.

Safari için Mac bilgisayarlar için OS X 10.7 WebGL desteği sunar ancak varsayılan olarak bu özellik kapalıdır. Bu desteği açmak için geliştirici menüsünden "WebGL'i Etkinleştir" seçeneğine tıklayarak bu seçeneği etkinleştirmek gerekmektedir.

Bundan sonra WebGL uygulamalarına gözatılabilir. Tarayıcınızda etkin olam özelliklere şu linkten erişilebilir. WebGL Raporu :

http://webglreport.sourceforge.net/

WebGL'de ilk olarak HTML sayfa içinde bir canvas kontrolü tanımlanmalı ve body etiketinin onLoad fonksiyonunda webGL'i başlatacak bir fonksiyon çağrılmalıdır. Örnek bir kod aşağıdaki gibi olabilir.

```
<body onload="webGLStart();">
     <canvas id="mycanvas" style="border: none;" width="500"
height="500"></canvas>
</body>
```

Bu sayfanın body etiketinde yer alan WebGL'in çalışması için gerekli tüm koddur (Geri kalanı başka amaçlar için kullanılmaktadır). Canvas öğesi 3B grafiğin barındığı yerdir ve HTML5 ile gelen bir özelliktir. 2B ve (WebGL ile) 3B JavaScript ile çizilen öğelerin web sayfalarında gösterilmesine olanak sağlar. Canvas etiketinde sadece sayfa yapısı ile ilgili bazı basit ayarlamalar yapılır ve geri kalan WebGL ayarlama işlemi webGLStart diye bir fonksiyona bırakılır. Bu fonksiyona bakılırsa:

```
function webGLStart() {
    var canvas = document.getElementById("mycanvas");
    initGL(canvas);
    initShaders();
```

```
        initBuffers();

        initWorldObjects();

        gl.clearColor(0.0, 0.75, 1, 1.0);//Gök mavisi
        gl.enable(gl.DEPTH_TEST);

        document.onkeydown = handleKeyDown;
        document.onkeyup = handleKeyUp;
        document.onmouseup = handleMouseUp;
        document.onmousemove = handleMouseMove;
        canvas.onmousedown = handleMouseDown;
        canvas.oncontextmenu = handleContextMenu;
        if (canvas.addEventListener)
            canvas.addEventListener('DOMMouseScroll', handleMouseWheel,
    false);

        tick();
    }
```

WebGL ve gölgelendiricileri başlatır, bunu 3B nesneleri çizmek için kullanılacak canvas öğesini parametre olarak fonksiyona geçerek yapar. Sonra initBuffers fonksiyonunu kullanarak çizilecek geometri bilgisini tutan tamponları başlatır. InitWorldObjects fonksiyonu ile çizilecek nesnelerin bilgisi alınır. Sonra bazı basit WebGL ayarlaması yapılır, yani canvas'ı temizleyip arka planın gök mavisi olması gerektiği ayarlanır. Bundan da sonra derinlik testi yapılır, böylece başka şeylerin arkasında olanların ön taraftaki şeylerin arkasında kalan kısımları gizlenir. Sonra klavyedeki yön tuşlarına basıldığında ve fare ile işlem yapıldığında görünümün nasıl değişmesi gerektiğini belirten olaylar için yapılacak işlemler belirtilir. En son olarak da tick fonksiyonu çağrılır, böylece tamponlardaki veri kullanılarak belirtilen grafik bilgisi çizdirilir.

```
    var gl;
    function initGL(canvas) {
        try {
            gl = canvas.getContext("experimental-webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
        } catch (e) {
        }
        if (!gl) {
            alert("Could not initialise WebGL, sorry :-(");
        }
    }
```

gl değişkeni bir çeşit WebGL bağlamıdır. Bu fonksiyon da bunu "experimental-webgl" ismi ile elde etmektedir. Bu isim bir süre sonra webgl olarak değişecektir.

```
    var mvMatrix = mat4.create();
    var pMatrix = mat4.create();
```

Burada mvMatrix olarak tanımlanan global değişken model-görünümü tutacak ve pMatrix olarak tanımlanan global değişken de projeksiyon matrisini tutacak ve onları kullanmak için boş (hepsi sıfır) olan matrislere dönüştürür. Cisimleri etrafta döndürme işlemi model-görünüm matrisi ile uzaktaki cisimleri yakındaki cisimlere göre orantısal olarak daha küçük gösterme işlemi projeksiyon matrisi ile yapılır. Mat4.persective fonksiyonu görünüm oranı ve görüş alanı ile matrisi perspektif için istenen değerlerle doldurur.

Yukarıda görüldüğü üzere initShaders fonksiyonu çağrılmaktadır. Gölgelendirici kavramı 3B grafik tarihinde ilk önceleri sisteme sahnenin çizilmeden önce nasıl gölgelendirileceğini, renklendirileceğini belirten kod olarak tarif edilirdi. Buna rağmen zamanla kapsam olarak genişledi ve şu an sahne çizilmeden önce sahnedeki her bir noktayı belirtebilen bir kod olarak tanımlanabilir. Bu aslında çok kullanışlıdır çünkü grafik kartında çalışır, böylece yaptıkları şeyi gerçekten çok hızlı yapar ve yaptıkları dönüşüm çok uygun olur.

Gölgelendiriciler ile grafik kartında çalışan WebGL sistemi elde edilir, bu sistem ile model-görünüm matrisi ve projeksiyon matrisi her bir noktayı ve her bir köşeyi görece yavaş JavaScript ile taşımadan sahneye uygulanmaktadır. Hatırlanacağı üzere webGLStart initShaders fonksiyonunu çağırmıştı, adım adım gidilirse :

```
var shaderProgram;
function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");
    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }
    gl.useProgram(shaderProgram);
    shaderProgram.vertexPositionAttribute =
gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
    shaderProgram.textureCoordAttribute =
gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
"uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
"uMVMatrix");
    shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram,
"uSampler");
    shaderProgram.useTexturesUniform =
```

```
    gl.getUniformLocation(shaderProgram, "uUseTextures");
    }
```

Görüldüğü üzere getShader fonksiyonu iki şeyi elde etmek için kullanılmaktadır; “fragment shader” (parça gölgelendirici) ve “vertex shader” (köşe gölgelendirici) ve sonra ikisini de “program” denen bir WebGL nesnesine bağlanmaktadır. Bir program sistemin WebGL tarafında yaşayan bir parçasıdır; grafik kartında çalışan bir şeyi tanımlamak olarak görülebilir. Bir miktar gölgelendirici onunla ilişkilendirilebilir, her biri bir kod parçası olarak bu programda görülebilir; özellikle, her bir program bir parça ve bir köşe gölgelendiricisi tutabilir. Fonksiyon bir kere programı ayarlayıp gölgelendiricileri eklediğinde, “attribute” a (özelliğe) bir referans elde eder, bu özellik program nesnesinde vertexPositionAttribute denen köşe bilgisini tutmak için bir alanı barındırır. JavaScript'in herhangi bir nesnede herhangi bir alan tutma isteğinden faydalanılır; program nesnesi aslında vertexPositionAttribute diye bir alana sahip değildir, ama bu iki değeri bir arada tutmada uygun bir yöntemdir, böylece özellik programın yeni bir alanı yapılır. Aynı şekilde desen bilgisi için aTextureCoord alanı tanımlanır ve değerlerin bir dizi olarak verileceği belirtilir. VertexPositionAttribute özelliği ile tamponu daha sonra ilişkilendirir. gl.enableVertexAttribArray fonksiyonu ile özellikler için WebGL'e değerlerin bir dizi ile verileceği belirtilir. Daha sonra da hangi uniform değişkenin shaderProgram üzerindeki hangi alana aktarılacağı belirtilir.

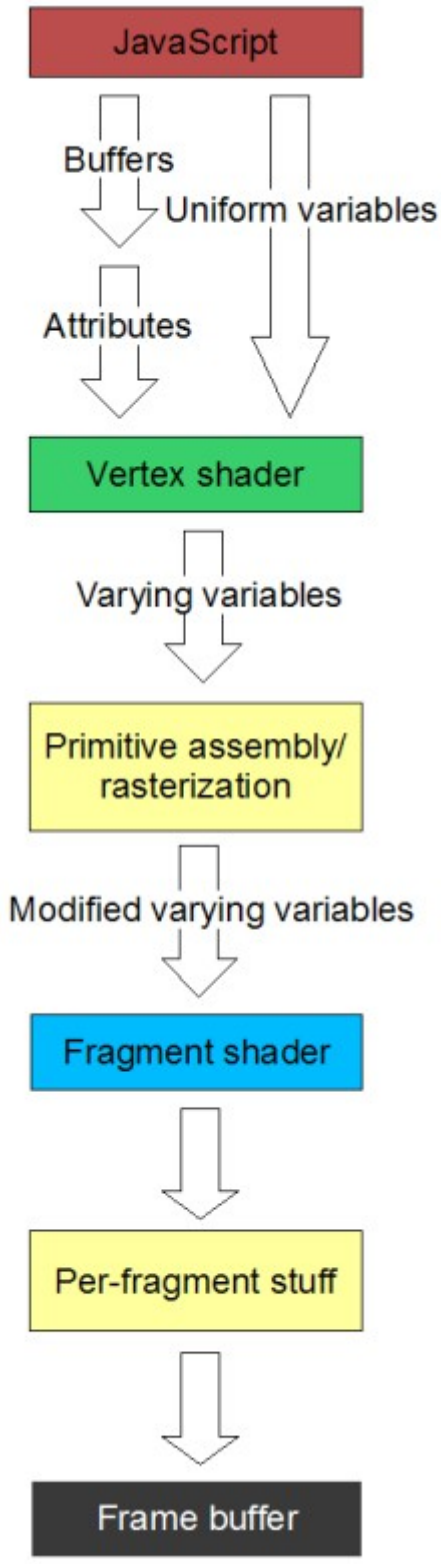
initShaders fonksiyonu program için iki değer daha alır, iki şeyin yerine uniform değişken denir. Özellik gibi bu da uygun olduğu için program nesnesinde tutulur. GetShader fonksiyonuna bakıldığında :

```
function getShader(gl, id) {
    var shaderScript = document.getElementById(id);
    if (!shaderScript) {
        return null;
    }
    var str = "";
    var k = shaderScript.firstChild;
    while (k) {
        if (k.nodeType == 3) {
            str += k.textContent;
        }

        k = k.nextSibling;
    }
    var shader;
    if (shaderScript.type == "x-shader/x-fragment") {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    } else if (shaderScript.type == "x-shader/x-vertex") {
        shader = gl.createShader(gl.VERTEX_SHADER);
```

```
        } else {
            return null;
        }
        gl.shaderSource(shader, str);
        gl.compileShader(shader);
        if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
            alert(gl.getShaderInfoLog(shader));
            return null;
        }
        return shader;
    }
```

Burada yapılan tüm işlem HTML sayfasında parametre geçilen ID ile aynı isimli bir öğeye bakıp, içeriği dışarı çekip, tipine göre parça veya köşe gölgelendiricisi oluşturup, sonra da WebGL'e geçip grafik kartında kullanılabilmesi için gerekli şekilde derlenmesini sağlamaktır. Sonra kod tüm hatalar ile ilgilenir ve işlem tamamlanır. Gölgelendiriciler aslında JavaScript içinde tanımlanabilir ve böylece HTML kısmını kirletmez ancak bu şekilde yaparak daha kolay okunması sağlanır, yani web sayfasında sanki JavaScript gibi bir script olarak tanımlanır.



Yandaki diyagram, drawScene metodunda JavaScript fonksiyonlarına geçilen verinin nasıl ekrandaki WebGL canvas'ında görüntülenen piksellere dönüştüğünü açıklamaktadır. DrawArrays gibi bir fonksiyon çağrıldığında WebGL özellikler şeklinde verilen veriyi (köşe gibi tamponları) ve uniform değişkenleri (projeksiyon matrisi ve model-görünüm matrisi gibi) işler ve bunu köşe gölgelendiriciye geçer:

Köşe gölgelendiriciyi her bir köşe için bir kere çağırır, her seferinde köşe için özellikler uygun şekilde ayarlanır; uniform değişkenleri de geçilir ama adından anlaşılacağı üzere onlar çağrımdan çağrıma değişmezler. Köşe gölgelendiriciler bu veri ile işlem yapar ve sonucunu değişen değişkenlere koyar, yani burada model-görünüm matrisine ve projeksiyon matrisine bir işlem uygular ve köşeler perspektif içinde olur ve şimdiki model-görünüm durumuna göre taşınır. Birkaç tane değişen değişkeni sonuç olarak verebilir; bir tane zorunlu olan ise gl_Position'dır ki bu, gölgelendirici köşe ile ilgili işlemi bitirdikten sonra köşenin koordinatlarını tutar.

Köşe gölgelendiricinin işi bittikten sonra WebGL 3B resmi 2B resme çevirmek için gerekli sihirli işlemi yapar ve sonra her bir piksel için parça gölgelendiriciyi bir kere çağırır. Yani parça gölgelendiriciyi köşesi olmayan her bir piksel için çağıracağı anlamına gelir. Bunlar için köşeler arasındaki noktalar lineer enterpolasyon denilen bir işlem ile kenarın görünür olması için

doldurulur. Parça gölgelendiricinin amacı her bir enterpolasyona uğramış noktanın rengini döndürmektir ve bunu gl_FragColor denen bir değişen değişken ile yapar.

Parça gölgeleme yapıldığında sonuçları WebGL tarafından biraz da karmaşıklaştırılır ve çerçeve tamponuna (frame buffer) konur ki bu da en sonunda ekranda görüntülenir.

Köşe gölgelendiriciden yer dışında bir kaç tane değişen değişken daha çıkarılabilir ve parça gölgelendiricide elde edilebilir. Böylece renk köşe gölgelendiriciye geçilir, sonra parça gölgelendiricinin elde edeceği bir değişen değişkene konur. Köşe gölgelendirici tarafından değer atanan tüm değişen değişkenler köşeler arasında parçalar üretirken lineer olarak enterpolasyona uğrar. Gölgelendirici koduna bakılırsa:

```
<script id="shader-fs" type="x-shader/x-fragment">
precision mediump float;
varying vec2 vTextureCoord;
varying vec3 vLightWeighting;
varying vec4 vColor;
uniform bool uUseTextures;
uniform sampler2D uSampler;
void main(void) {
    vec4 fragmentColor;
    if (uUseTextures)
        fragmentColor = texture2D(uSampler, vec2(vTextureCoord.s,
vTextureCoord.t));
    else
        fragmentColor = vec4(.45, .3, .07, 1.0);//Toprak rengi
    gl_FragColor = vec4(fragmentColor.rgb * vLightWeighting,
fragmentColor.a);
}
</script>
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;
attribute vec4 aVertexColor;
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
uniform bool uUseLighting;
varying vec2 vTextureCoord;
varying vec3 vLightWeighting;
varying vec4 vColor;
void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;
    if (!uUseLighting)
        vLightWeighting = vec3(1.0, 1.0, 1.0);
    vColor = aVertexColor;
}
</script>
```

Bunlar hakkında hatırlanması gereken ilk şey çok benzese de JavaScript olarak

yazılmadığıdır. Aslında GLSL diye C'ye ve JavaScript'e çok benzeyen özel bir gölgelendirme dili ile yazılır [14].

İlki, parça gölgelendirici; grafik kartına kayan-noktalı sayılarda ne kadar duyarlı olacağını belirten zorunlu bir koda sahiptir (mediump orta seviyeli duyarlılık tüm WebGL cihazlarında çalışması için iyidir, highhp yüksek seviyeli duyarlılık tüm mobil cihazlarda çalışmaz), daha sonra desen bilgisi sağlamamışsa sonradan çizilen herşeyin toprak renginde çizilmesi sağlanır. Desen, ışık şiddeti ve renk değişen değişkenleri ile parçalar için enterpole edilmiş değerler alınır. Numunelendirici WebGL'in deseni sunma biçimidir. Gölgelendiricinin yaptığı iş koordinatları kullanarak uygun rengi bulmaktır. Desenler geleneksel olarak koordinatları belirtmek için x ve y yerine s ve t kullanır, gölgelendirici dili de bu lakapları destekler, kolaylıkla vTextureCoord.x ve vTextureCoord.y olarak kullanılabilir. uUseTextures da çizim yapıldığı sırada desen kullanılıp kullanılmayacağını belirtir.

İkinci gölgelendirici, köşe gölgelendiricisi köşe ile ilgili hemen her şeyi yapan grafik kartı kodudur. Bununla ilişkili olarak uMVMatrix ve uPMatrix diye iki tane uniform değişkeni vardır. Uniform değişkenler gölgelendirici dışından erişilebildiği, aslında onu içeren program dışından erişilebildiği için kullanışlıdır, çünkü model-görünüm ve projeksiyon matrisleri değerleri onlara atanır. Nesne tabanlı mantığında gölgelendiricinin programı bir nesne olarak uniform değişkenler de alan olarak düşünülebilir. Burada desen ve renk girdi değişkenlerden çıktı değişken değişkenlere aktarılır. Işıklandırma kullanılmadığı durumda varsayılan olarak sahne tam aydınlık olarak ayarlanır.

Gölgelendirici her bir köşe için çağrılır ve köşenin yeri gölgelendirici koduna aVertexPosition olarak geçilir, vertexPositionAttribute özelliğinin drawScene metodunda kullanımı sayesinde, özellik ve tampon ilişkilendirilir. Gölgelendiricinin ana kısmındaki küçük kodda köşenin yeri ile model-görünüm ve projeksiyon matrisleri çarpılır ve çıkan sonuç köşenin yeni yeri olarak atanır.

Böylece webGLStart initShaders'ı çağırır, o da getShader'i web sayfasındaki script'lerden parça ve köşe gölgelendiricilerini yüklemek için kullanır, böylece derlenebilirler ve WebGL'e geçilirler ve 3B sahne oluşturulurken kullanılırlar.

```
function setMatrixUniforms() {
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}
```

setMatrixUniforms fonksiyonu ile initShaders ile elde edilen model-görünüm matrisini ve projeksiyon matrisini temsil eden uniform'lara olan referanslar kullanılarak JavaScript stilindeki matrislerden WebGL'e değerler gönderilir.

```
var squareVertexPositionBuffer;      //Zemin için koordinatları belirtir.
var solidVertexPositionBuffer;       //Katı cisim için yer bilgisini tutar
var solidVertexColorBuffer;          //Katı cisim için renk bilgisini tutar
var solidVertexTextureCoordBuffer;   //Katı cisim için desen bilgisini
tutar
var solidVertexIndexBuffer;          //Katı cisim için köşe indeks
bilgisini tutar
var solidTextureCoords = [//Desen için 2B yer bilgisi 0,0 sol alt, 1,1 sağ
üstü ifade eder
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,// Front face
    1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0,// Back face
    0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0,// Top face
    1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,// Bottom face
    1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0,// Right face
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,// Left face
];
var solidVertexIndices = [//Katı cisim için çizilecek köşelerin üçgen
oluşturup //ışıklandırma yönünü belirtmek için üçlü gruplar olarak sırası
    0, 1, 2, 0, 2, 3,    // Front face
    4, 5, 6, 4, 6, 7,    // Back face
    8, 9, 10, 8, 10, 11,  // Top face
    12, 13, 14, 12, 14, 15, // Bottom face
    16, 17, 18, 16, 18, 19, // Right face
    20, 21, 22, 20, 22, 23  // Left face
];
function initBuffers() {
    //Zeminin çizilmesi için gereken yer bilgileri
    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    vertices = [
                1000.0, 0.0, 1000.0,
                -1000.0, 0.0, 1000.0,
                1000.0, 0.0, -1000.0,
                -1000.0, 0.0, -1000.0
    ];

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;

    //Zeminin çizilmesi için gereken renk bilgileri
    squareVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    colors = [];
    colors = colors.concat([0.2, 0.4, 0.6, 1.0]);
    colors = colors.concat([0.6, 0.8, 1.0, 1.0]);
    colors = colors.concat([1.0, 0.2, 0.4, 1.0]);
    colors = colors.concat([0.4, 0.6, 0.8, 1.0]);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
```

```
        squareVertexColorBuffer.itemSize = 4;
        squareVertexColorBuffer.numItems = 4;

        //Katı cismin çizilmesi için gereken desen bilgileri
        solidVertexTextureCoordBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, solidVertexTextureCoordBuffer);
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(solidTextureCoords),
    gl.STATIC_DRAW);
        solidVertexTextureCoordBuffer.itemSize = 2;
        solidVertexTextureCoordBuffer.numItems = 24;
        //Katı cismin çizilmesi için gereken köşe indeks bilgileri
        solidVertexIndexBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, solidVertexIndexBuffer);
        gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new
    Uint16Array(solidVertexIndices), gl.STATIC_DRAW);
        solidVertexIndexBuffer.itemSize = 1;
        solidVertexIndexBuffer.numItems = 36;
    }
```

gl.createBuffer çağrımı ile katı cisim için köşelerin 3B uzaydaki yerleri, renkleri, desenleri ve köşe indeksleri için tampon bilgisi oluşturulur. Tampon aslında grafik kartındaki hafızadadır; köşelerin ilgili bilgilerini başlangıç kodunda tampona koyarak, sahne çizileceği zaman WebGL'e "daha önceden söylenen şeyleri çiz" denebilir ve kod çok verimli hale getirilebilir. Küçük modellerde bu bilgileri grafik kartına koymada fazla maliyet olmamasına rağmen onbinlerce köşeye sahip büyük modellerle çalışırken işlemleri bu şekilde yapmak çok verimli olur. Daha sonra gl.bindBuffer fonksiyon çağrımları ile bundan sonra gelecek olan tüm işlemler bu fonksiyon çağrımındaki tampon değişkeni üzerinde yapılacak denir. Daima bir "şimdiki tampon" kavramı vardır, ve fonksiyonlar her seferinde belirtilmesine gerek olmaksızın sürekli onun üzerinde işlem yapar. Sonrasında gl.bufferData fonksiyonu JavaScript tipinde dizileri alıp burada Float32Array veya Uint16Array tipinde WebGL'e yükler. Tampon ile ilgili daha sonra itemSize ve numItems diye iki yeni özellik daha tanımlanır. JavaScript bir özelliği açıkça belirtmek zorunda değildir; yani bir nesnenin bir özelliğine atama yapıldığında bu özellik daha önce tanımlanmamış olsa dahi o an tanımlanmış olur. Burada tanımlanan özellikler: zemin köşe yer bilgisi için x, y, z olarak 3 ayrı koordinata sahip 4 köşe olduğunu, zemin renk bilgisi için rbga(red, green, blue, alpha. Alfa bir solukluk ölçülüdür, 0 tamamen saydam, 1 tamamen donuk, ışık geçirmezdir.) olarak 4 ayrı renk öğesi ile toplamda 4 ayrı renk olduğunu, köşe desen bilgisi için 2 ayrı öğe (0,0 sol, alt veya 1,1 sağ, üst gibi) olarak toplamda 24 ayrı desen bilgisi olduğu ve bir boyutlu 36 köşe indeks numarası olduğunu belirtmektedir.

```
function initWorldObjects() {
    if (window.XMLHttpRequest) {// code for IE7+, Firefox, Chrome, Opera,
Safari
        xmlhttp = new XMLHttpRequest();
    }
    else {// code for IE6, IE5
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.open("GET", apiPath + "building", false);
    xmlhttp.send();
    if (xmlhttp.status != 200)
        return;
    var jsonResponse = JSON.parse(xmlhttp.response);
    if (jsonResponse != null) {
        for (var c = 0; c < jsonResponse.length; c++) {
            var item = jsonResponse[c];
            if (item.Geometry != null && item.Geometry.Sdo_gtype == 3008){
                for (var i = 0; i < item.Images.length ; i++)
                    textures[i] = initTexture(item.Images[i]);
                PrepareSolid(item.Geometry.OrdinatesArray);
            }
        }
    }
}
```

initWorldObjects fonksiyonunda window.XMLHttpRequest javascript özelliği desteklenen yeni tarayıcılar için XMLHttpRequest, diğer IE5 ve IE6 için ActiveXObject("Microsoft.XMLHTTP") nesnesi ile yeni bir xmlhttp nesnesi alınır. Sonrasında "http://localhost:14418/api/building" gibi bir adrese bir istek gönderilir. İstek sonucu başarılı ise dönen JSON sonucu ayrıştırılır. Burada dolu bir veri gelirse, dizi şeklinde gelen bu verinin her bir öğesi için bir veri olduğu ve geometri veri tipinin katı cisim olduğu kontrol edilir. Sonrasında bu öğenin resimleri sırayla desen bilgilerinin tutulduğu diziye aktarılır. Sonrasında da PrepareSolid fonksiyonu çağrılır.

```
var solids = [];
function PrepareSolid(geo) {
    var x0 = geo[0];
    var y0 = geo[1];
    var z0 = geo[2];
    var x1 = geo[3];
    var y1 = geo[4];
    var z1 = geo[5];
    solids.push(new Solid(x0, y0, z0, x1, y1, z1));
}
```

Bu fonksiyonda solids isimli katı cisimlerin bulunduğu listeye öğe eklenir.

```
var solidVertices = [//Katı cisim için köşenin x, y, z ortamında hangi
alana düştüğünü belirtir
    -1.0, -1.0, 1.0,  1.0, -1.0, 1.0,  1.0, 1.0, 1.0,  -1.0, 1.0, 1.0,  //
Front face
    -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, //
Back face
    -1.0, 1.0, -1.0,  -1.0, 1.0, 1.0,  1.0, 1.0, 1.0,  1.0, 1.0, -1.0,  //
```

```
Top face
    -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, //
Bottom face
    1.0, -1.0, -1.0,  1.0, 1.0, -1.0,  1.0, 1.0, 1.0,  1.0, -1.0, 1.0,  //
Right face
    -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0  //
Left face
];
```

Bu şekilde solidVertices dizisi tanımlanır, her bir üçlü değer x, y, z değerlerine karşılık gelir, her bir dört tane üçlü değer de bir yüzeyi tanımlar, bu şekilde altı yüzeyi olan bir katı cismin yüzeyleri belirtilmiş olur.

```
function Solid(x, y, z, x1, y1, z1) {
    this.x = x;//mix x
    this.y = y;//min y
    this.z = z;//min z

    this.vertices = [];
    for (var i = 0; i < solidVertices.length; i++) {
        if (i % 3 == 0) {//x değerini ata
            if (solidVertices[i] == -1)
                this.vertices.push(x);
            else
                this.vertices.push(x1);
        }
        if (i % 3 == 1) {//y değerini ata
            if (solidVertices[i] == -1)
                this.vertices.push(y);
            else
                this.vertices.push(y1)
        }
        if (i % 3 == 2) {//z değerini ata
            if (solidVertices[i] == -1)
                this.vertices.push(z);
            else
                this.vertices.push(z1);
        }
    }
    this.solidVertexPositionBuffers = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, this.solidVertexPositionBuffers);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.vertices),
gl.STATIC_DRAW);
    this.solidVertexPositionBuffers.itemSize = 3;
    this.solidVertexPositionBuffers.numItems = 24;
}
```

Buradaki Solid fonksiyonu JavaScript dilinde tanımlanmış bir yapıcı metoddur. Bu metoda cismin yerini tanımlamak için en düşük ve en yüksek x,y,z değerleri parametre olarak geçilir. X, y, z, vertices ve solidVertexPositionBuffers alanları yazıldığı satırlarda JavaScript dili sayesinde tanımlanmış olurlar. This.vertices dizisine değer atanması gereken yüzeydeki her bir köşe için koordinatlar katı cismin merkezi 0, 0, 0 olarak düşünülüp x, y, z değerlerinin sadece negatif veya pozitif işaretleri değiştirilerek diziye değer atanır. This.solidVertexPositionBuffers alanı

olarak yeni bir tampon oluşturulur, bu tampona yeni oluşturulan this.solidVertexPositionBuffers verisi dizi olarak bağlanır ve çizilmek üzere hazırlanır. Her bir yer bilgisi üç öğeden oluşur, toplam öğe sayısı da sekiz köşeyi ifade etmek üzere 24dür. Nesnemizin köşe yerlerini grafik kartına yüklemek için yapılması gerekenler bunlardır.

```
function tick() {
    requestAnimFrame(tick);
    handleKeys();
    drawScene();
}
```

İlk satır yeniden sahnenin çizilmesi gerektiğinde tekrar çağrılmasını ayarlar. RequestAnimFrame script etiketinde eklenen webgl-utils.js içinde gelen Google tarafından sağlanan bir fonksiyondur. Bu tarayıcı bağımsız bir şekilde tarayıcıya WebGL sahnesinin yenilenmesi gerekip gerekmediğini sormayı sağlar. Bu işlemi WebGL'i destekleyen tüm tarayıcılarda yapacak fonksiyonlar vardır, ancak bunlar farklı isimler altında bulunmaktadır (Mesela Firefox mozRequestAnimationFrame isimli bir fonksiyona sahipken, Chrome ve Safari ise webkitRequestAnimationFrame isimli bir fonksiyona sahiptir). Gelecekte hepsinin requestAnimationFrame isimli bir fonksiyon kullanılması beklenmektedir. O zamana kadar Google WebGL fonksiyonları bu işlemi yapmak için kullanılabilir.

RequestAnimFrame fonksiyonu ile benzer etki JavaScript'e drawScene fonksiyonu düzenli olarak çağrıltılarak da alınabilir, mesela JavaScript ile gelen setInterval fonksiyonunu kullanarak olduğu gibi. Bir çok ilk çıkan WebGL kodlarında bu iş görüyordu, ancak WebGL çalıştıran bir kaç tane tarayıcı penceresi olmaya başladıktan sonra bu bir sorun olmaya başladı. Çünkü setInterval fonksiyonu ile zamanlanan fonksiyonlar çalıştığı tarayıcı penceresinin o an aktif olup olmadığına bakmıyordu. Bu yüzden sadece aktif pencerede WebGL çalıştırmak için requestAnimFrame fonksiyonu oluşturuldu.

```
function handleKeyDown(event) {
    currentlyPressedKeys[event.keyCode] = true;
}
function handleKeyUp(event) {
    currentlyPressedKeys[event.keyCode] = false;
}
var XOffset = 4;    //Kamera X yeri, başlangıçta orijin noktasının 4 birim
sağında yerleştir.
var YOffset = -2;   //Kamera Y yeri, başlangıçta orijin noktasının 2 birim
üstünde yerleştir.
var ZOffset = -17;  //Kamera Z yeri, başlangıçta orijin noktasının 17
```

```
birim gerisinde yerleştir.
var ZoomStep = 0.25;//X, Y, Z koordinatlarında yaklaşma adımı
function handleKeys() {
    if (currentlyPressedKeys[90] || currentlyPressedKeys[122])// Z || z
        ZoomStep /= 1.4;//Yaklaşma adımını küçült
    if (currentlyPressedKeys[88] || currentlyPressedKeys[120])// X || x
        ZoomStep *= 1.4;//Yaklaşma adımını büyüt
    if (currentlyPressedKeys[40] || currentlyPressedKeys[83]) // Down || S
        ZOffset -= ZoomStep;//Kamerayı uzaklaştır
    if (currentlyPressedKeys[38] || currentlyPressedKeys[87]) // Up || W
        ZOffset += ZoomStep;//Kamerayı yaklaştır
    if (currentlyPressedKeys[37] || currentlyPressedKeys[65]) // Left || A

        XOffset += ZoomStep;//Kamerayı sola kaydır
    if (currentlyPressedKeys[39] || currentlyPressedKeys[68]) // Right ||
D
        XOffset -= ZoomStep;//Kamerayı sağa kaydır
    if (currentlyPressedKeys[34] && YOffset < 0)              // Page Down

    {
        YOffset += ZoomStep;//Kamerayı aşağı indir
        if (YOffset > 0)//Yer seviyesinden aşağı inme
            YOffset = 0;
    }
    if (currentlyPressedKeys[33])                             // Page Up
        YOffset -= ZoomStep;//Kamerayı yukarı kaldır
    if (currentlyPressedKeys[81] ||currentlyPressedKeys[113]) // Q || q
        RotateX(lastMouseX + ZoomStep, ZoomStep);//Kamerayı sola döndür
    if (currentlyPressedKeys[69] || currentlyPressedKeys[101])// E || e
        RotateX(lastMouseX + ZoomStep, -ZoomStep);//Kamerayı sağa döndür
}
```

handleKeyDown ve handleKeyUp fonksiyonları ile currentlyPressedKeys adlı şu an basılı olan tuşlar belirtilir. XOffset, YOffset, ZOffset, ZoomStep değişkenleri ile kameranın yeri ve bakış açısı ayarlanmaktadır. HandleKeys fonksiyonunda, fonksiyon içinde hangi tuşa basınca kameranın hangi yönde ve ne birimde hareket edeceği belirtilmektedir.

```
var newRotationMatrix = mat4.create();//4x4 her öğesi 0 olan bir matris
oluşturur
var rotationMatrix = mat4.create();//4x4 her öğesi 0 olan bir matris
oluşturur
mat4.identity(rotationMatrix);//mat4 nesnesine kimlik matrisini atar
var clientX = null;
var clientY = null;

function RotateX(newX, deltaX) {
    mat4.identity(newRotationMatrix);
    //newRotationMatrix matrisini X ekseni etrafında belirtilen radyan
derece kadar döndür. Sonuçta çıkan matris değerini newRotationMatrix
matrisine yaz
    mat4.rotateY(newRotationMatrix, degToRad(deltaX));
    //newRotationMatrix ile rotationMatrix matrislerini çarp, sonucu
rotationMatrix matrisine yaz
    mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
    clientX = newX;//Kamera X koordinatını ayarla
}
```

```
function RotateY(newY, deltaY) {
    mat4.identity(newRotationMatrix);
    //newRotationMatrix matrisini Y ekseni etrafında belirtilen radyan
derece kadar döndür. Sonuçta çıkan matris değerini newRotationMatrix
matrisine yaz
    mat4.rotateX(newRotationMatrix, degToRad(deltaY));
    //newRotationMatrix ile rotationMatrix matrislerini çarp, sonucu
rotationMatrix matrisine yaz
    mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
    clientY = newY;//Kamera Y koordinatını ayarla
}

function degToRad(degrees) {
    return degrees * Math.PI / 180;
}
```

newRotationMatrix değişkeni o anda yapılan dönme miktarını hesaplamak için kullanılan matristir. RotationMatrix cismin son haldeki toplam dönme durumunu tutan matristir ve program başladığında kimlik matrisi olarak atanır. RotateX ve RotateY fonksiyonlarında kameranın yeni X veya Y konumu ve önceki hale göre yaptığı newX veya newY fark değerleri alınır. İlk olarak newRotationMatrix matrisine kimlik matrisi olarak değer atanır, sonrasında mat4.rotate[X-Y] fonksiyonu ile matris dönme işlemi hesaplanır, burada dönme derecesi delta[X-Y] radyan cinsinden belirtilir. Bundan sonra mat4.multiply fonksiyonu ile newRotationMatrix ve rotationMatrix matrisleri çarpılıp sonucu rotationMatrix matrisine yazılır. En son olarak da kameranın yeni X veya Y koordinatı ayarlanır. Bu şekilde cisim döndürülmüş olur. Burada degToRad fonksiyonu derece cinsinden aldığı değeri radyan cinsinden dereceye dönüştürür.

```
//Farenin sağ tuşuna basılınca sağ tuş menüsünü gösterme
function handleContextMenu(event) {
    return false;
}
var mouseDown = false;//Fare tuşlarına basılıp basılmadığını tutar
function handleMouseDown(event) {
    mouseDown = true;
    clientX = event.clientX;//Ekranda farenin tıklandığı X konumu
    clientY = event.clientY;//Ekranda farenin tıklandığı Y konumu
}
function handleMouseUp(event) {
    mouseDown = false;
}
function handleMouseMove(event) {
    if (!mouseDown) {
        return;
    }
    var newX = event.clientX;//Ekranda farenin tıklandığı X konumu
    var deltaX = newX - clientX;
    var newY = event.clientY;//Ekranda farenin tıklandığı Y konumu
    var deltaY = newY - clientY;
    if (event.button == 0) {//Sol tuş
        XOffset += deltaX / 100;//Kamerayı sola veya sağa kaydır
        YOffset -= deltaY / 100;//Kamerayı yukarı veya aşağı kaydır
```

```
            if (YOffset > 0)//Kamera zemin hattından daha aşağı geçemesin
                YOffset = 0;
        }
        else if (event.button == 2) {//Sağ tuş
            RotateX(newX, deltaX);
        }
    }
```

Yukarıdaki fonksiyonlar ile fare ile yapılan işlemler kontrol altına alınır. Fare sağ tuşa basılınca her hangi bir menü çıkmasının önüne geçilir. Farenin basılı olup olmadığı handleMouseDown ve handleMouseUp fonksiyonları ile kontrol edilir. HandleMouseMove fonksiyonu ile fareye basılı olduğu durumda farenin hareketi miktarınca ekranda kamera koordinatı tekrar ayarlanır.

DrawScene fonksiyonunda bu tamponlar kullanılarak gördüğümüz asıl resim çizilir. Adım adım ilerlenirse:

```
    function drawScene() {
        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
```

Bu ilk adım WebGL'e viewPort fonksiyonunu kullanan canvas'a boyut hakkında bilgi vermek içindir; fonksiyon canvas'ın boyutunu çizim yapmadan önce bilmelidir. Sonra, çizim yapmak için hazırlık yaparken canvas temizlenir:

```
      gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

ve sonra:

```
      mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 360.0,
    pMatrix);
```

Sahnenin görülmesi istenen perspektifi ayarlanır. Varsayılan olarak WebGL uzak ve yakın nesneleri aynı yakınlıkta çizer (ortografik projeksiyon denen 3B bir stil). Uzaktaki nesneleri küçük kılmak için kullanılan perspektiften bahsedilmesi gerekir. Bu sahne için dikey görüş alanının 45° olduğu, canvas'ın genişliğin yüksekliğe oranı ve sadece görüş noktasına 0.1 birimden uzak ve 100 birimden yakın nesnelerin görülmesinin istendiği belirtilir.

Görüldüğü gibi bu perspektif tanımı mat4 isimli bir modülden bir fonksiyon kullanmaktadır ve pMatrix isminde bir değişken içermektedir. Perspektif ayarlandıktan sonra bazı nesneler çizilebilir:

```
        mat4.identity(mvMatrix);
```

İlk adım 3B sahnenin ortasına “taşıma” işlemidir. OpenGL'de bir sahne çizilirken her bir nesneyi “şimdiki” yerde ve “şimdiki” doğrultuda –mesela “20 birim ileri taşı, 32 derece döndür, sonra robotu çiz”- çizilmesi gerektiği söylenir. Bu, “robotu çiz” kodu

bir fonksiyon yapılabildiği için kullanışlıdır ve daha sonra taşıma/yönlendirme ayarları değiştirilerek belirtilen robot kolayca taşınabilir.

Şu anki yer ve şu anki yönlenme bir matriste tutulur; matrisler bir yerden bir yere yer değiştirmeyi ifade etmek, yönlenmeyi belirtmek ve başka geometrik dönüşümleri belirtmek için kullanılabilir. 3B bir uzayda herhangi bir sayıdaki dönüşüm 4x4 bir matris ile (3x3 değil) belirtilebilir; ilk önce kimlik matrisi ile başlanır -yani her hangi bir şey yapmayan dönüşüm matrisi- sonra ilk dönüşümü temsil eden matris ile çarpılır, sonra ikinci dönüşümü temsil eden matris ile çarpılır ve daha sonra benzer şekilde devam edilir. Sonuçta ortaya çıkan matris tüm dönüşümleri bir seferde sunar. Bu şekilde taşıma/yönlenme durumunu belirten matrise model-görünüm matrisi denir, mvMatrix değişkeni model-görünüm matrisini tutar, mat4.identity fonksiyonu onu kimlik matrisine atar ve taşımaları ve yönlendirmeleri onunla çarpabilmek için hazır hale getirir. Veya başka bir deyişle bizi 3B dünyayı çizmeye başlayabilmek için bir başlangıç noktasına götürür.

Ama WebGL ise OpenGL gibi grafik kütüphanesinde bu şekilde bir yapı bulundurmamaktadır. Bunun yerine aynı etkiyi elde etmek için bu çalışmada Brandon Jone'un glMatrix kütüphanesi ve bazı WebGL kolaylıkları kullanılacaktır.

Şehir nesnesini sahneye çizecek koda bakılırsa :

```
mat4.translate(mvMatrix, [XOffset, YOffset, ZOffset]);
```

mvMatrix matrisini kimlik matrisi olarak taşıdıktan sonra x, y, z koordinatlarında nesne istenen konuma taşınır, yani pozitif X değerleri ekranın sağına negatif X değerleri ekranın soluna, pozitif Y değerleri yukarı, negatif Y değerleri aşağı, negatif Z değerleri kameradan uzağa doğru anlamına gelmektedir. mat4.translate verilen matrisi verilen taşıma matrisi ile çarp anlamına gelmektedir.

```
mat4.multiply(mvMatrix, rotationMatrix);
```

Daha sonra nesne verilen döndürme matrisi ile döndürülür.

```
gl.uniform1i(shaderProgram.useTexturesUniform, true);//Desen bilgisi
kullanmaya başla
```

Desen çizmeye başlarken şu an desen kullanmaya başlandığı belirtilir. Bu desenler ile binanın cephe resimleri giydirilecek.

```
for (var i in solids)
 solids[i].drawThis();
```

Bu komutlar ile katı cisimler listesindeki nesneler teker teker çizdirilir. Burada nesne tabanlı bir yöntem olarak o anki cismin drawThis fonksiyonu çağrılarak cismin çizdirilmesi sağlanır.

```
Solid.prototype.drawThis = function () {
    mvPushMatrix();
    mat4.translate(mvMatrix, [this.x, this.y, -this.z]);
    gl.bindBuffer(gl.ARRAY_BUFFER, this.solidVertexPositionBuffers);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
this.solidVertexPositionBuffers.itemSize, gl.FLOAT, false, 0, 0);
    gl.bindBuffer(gl.ARRAY_BUFFER, solidVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
solidVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
    DrawTexture(gl.TEXTURE0, 0);
    DrawTexture(gl.TEXTURE1, 1);
    DrawTexture(gl.TEXTURE2, 2);
    DrawTexture(gl.TEXTURE3, 3);
    DrawTexture(gl.TEXTURE4, 4);
    mvPopMatrix();
};
var mvMatrixStack = [];//Matrisleri tutan dizi
function mvPushMatrix() {//Matris yığın dizisine bir model-görünüm matrisi
kopyası ekler
 var copy = mat4.create();
 mat4.set(mvMatrix, copy);
 mvMatrixStack.push(copy);
}

function mvPopMatrix() {//Matris yığın dizisinden bir matris alıp siler ve
şu anki model-görünüm matrisi yapar
 if (mvMatrixStack.length == 0) {
 throw "Invalid popMatrix!";
 }
 mvMatrix = mvMatrixStack.pop();
}
```

Burada mvPushMatrix fonksiyonu şu anki çizdirme yerini bir diziye koyup orada bekletir. Bundan sonra yapılan çizme işlemlerinden sonra mvPopMatrix fonksiyonu çağrılarak çizdirme işlemlerinden önceki çizdirme pozisyona gidilir, böylece başlangıç çizdirme pozisyonuna bağlı olarak ifade edilen çizdirme işlemleri düzgün bir şekilde yapılabilir. Aksi takdirde eğer bu fonksiyonlar olmasaydı, çizdirme fonksiyonu son çizimden sonra kalan yerden başlayarak yapılırdı ve iç içe geçmiş çizimler meydana gelirdi. Mat4.translate ile matris istenen şekilde taşınır. gl.bindBuffer ile katı cisim köşe bilgileri yüklenir. Sonrasında gl.vertexAttribPointer fonksiyonu ile köşe bilgisi dizisinin boyutu belirtilir. Bundan sonra cismin desen bilgileri de yüklenir. DrawTexture fonksiyonu ile her bir kenar desen bilgisi çizdirilir.

```
    function DrawTexture(textureN, index) {
        gl.activeTexture(textureN);
        gl.bindTexture(gl.TEXTURE_2D, textures[index]);
        gl.uniform1i(shaderProgram.samplerUniform, index);
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, solidVertexIndexBuffer);
        setMatrixUniforms();
        if (index > 2) index++;//Zeminde olması gereken desen bilgisi anlamlı
    olmadığı için kullanılmamaktadır.
        gl.drawElements(gl.TRIANGLES, 6, gl.UNSIGNED_SHORT, index * 12);
    }
```

DrawTexture fonksiyonuna desen bilgisi ve desenin dizide kaçıncı indekste olduğu bilgisi verilir. WebGL gl.drawElements fonksiyonu çağrıldığında 32 ayrı desen bilgisi kullanabilir ve desenler texture0'dan texture31'e kadar değer alabilir. gl.activateTexture ile verilen desen bilgisi üzerinde çalışıldığı belirtilir. gl.bindTexture ile desenler dizindeki belirtilen indeksteki desen bilgisi şu anki desen bilgisi olarak belirtilmiş olur. gl.uniform1i fonksiyonu ile gölgelendiriciye belirtilen indeksteki desenin kullanıldığı bildirilir. gl.bindBuffer fonksiyonu cisim köşe indeks tamponundaki değerleri kullanarak köşe yerlerini kullanarak desen çizdirmeyi sağlar. gl.drawElements fonksiyonu kullanılarak 6 tane köşe kullanıp her bir yüz için 2 tane üçgen çizilmektedir. DrawScene metodu daha sonra şu şekilde devam eder.

```
    gl.uniform1i(shaderProgram.useTexturesUniform, false);//Desen bilgisi
kullanma
    //Zemini çiz
    mvPushMatrix();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
    mvPopMatrix();
```

Cisimler çizildikten sonra desen bilgisinin kullanılmaması belirtilir. Zemini çizmek için kare köşe yer bilgileri tampona bağlanır, dizideki eleman sayısı belirtilir, kare köşe renk bilgisi tampona bağlanır, dizideki eleman sayısı belirtilir, sonrasında gl.drawArrays fonksiyonu ile çizim sağlanır.

```
    function initTexture(fileName) {
        var texture = gl.createTexture();//Yeni bir desen nesnesi oluştur
        texture.image = new Image();//Desen bilgisi için yeni bir resim
```

```
    nesnesi oluştur
        texture.image.onload = function () {//Resim yüklenince çalışacak
    işlemler
            handleLoadedTexture(texture)
        }
        texture.image.src = fileName;//Desen resim yolu
        return texture;
    }
    function handleLoadedTexture(texture) {
        gl.bindTexture(gl.TEXTURE_2D, texture);//Deseni bağla
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);//Dikey olarak ters çevir
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
    texture.image);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
        gl.bindTexture(gl.TEXTURE_2D, null);//Desen bağını kopar
    }
```

initTexture fonksiyonu verilen dosya adı ile desen bilgisi oluşturmayı sağlar. Bunu yaparken gl.createTexture fonksiyonunu kullanarak yeni bir desen oluşturur, bu desen için bir resim nesnesi oluşturur, bu resim tamamen yüklendiği zaman çağrılacak bir olay oluşturur ve sonrasında da resim yolunu belirtir. Resim yükleme işlemi asenkron yapılır, yani resim yolu belirtildikten sonra ayrı bir Thread açılır ve bu arkaplanda çalışan iş sayesinde resim yüklenir. Resim yükleme işlemi bittiğinde handleLoadedTexture fonksiyonu çağrılır.

Burada yapılan ilk iş WebGL'e desen bilgisinin “şu anki” desen bilgisi olduğunu belirtmektir. Buradaki bindbuffer fonksiyonu daha önceki kullanımlar ile benzerdir. Sonra WebGL'e yüklenen tüm resimlerin dikey olarak ters çevrilmesi gerektiği söylenir. Bu, koordinatlarda bir fark olduğu için yapılır; buradaki desen koordinatlarını normalde matematikte kullanılan koordinatlar gibi kullanabilmek için yapılır, yukarı doğru gittikçe artan şekilde koordinatların kullanılması istenir. Bu da köşeleri belirtirken kullanılan X, Y, Z koordinatları ile tutarlıdır. Buna karşın çoğu bilgisayar grafikleri sistemi – mesela GIF biçimi – dikey eksende aşağı doğru gittikçe değer olarak artar. Yatay eksen tüm koordinat sistemlerinde aynıdır.

Sonraki adımda texImage2D fonksiyonu kullanılarak henüz yüklenmiş resmi desenin grafik kartındaki yerine yükler. Sırasıyla parametreler ne tür resim yüklendiği, ayrıntı seviyesi, grafik kartında ne türde tutulması gerektiği, her bir “kanalın” (yani, kırmızı, yeşil ve mavi'yi tutmak için kullanılan veri tipi) boyutu ve resmin kendisi.

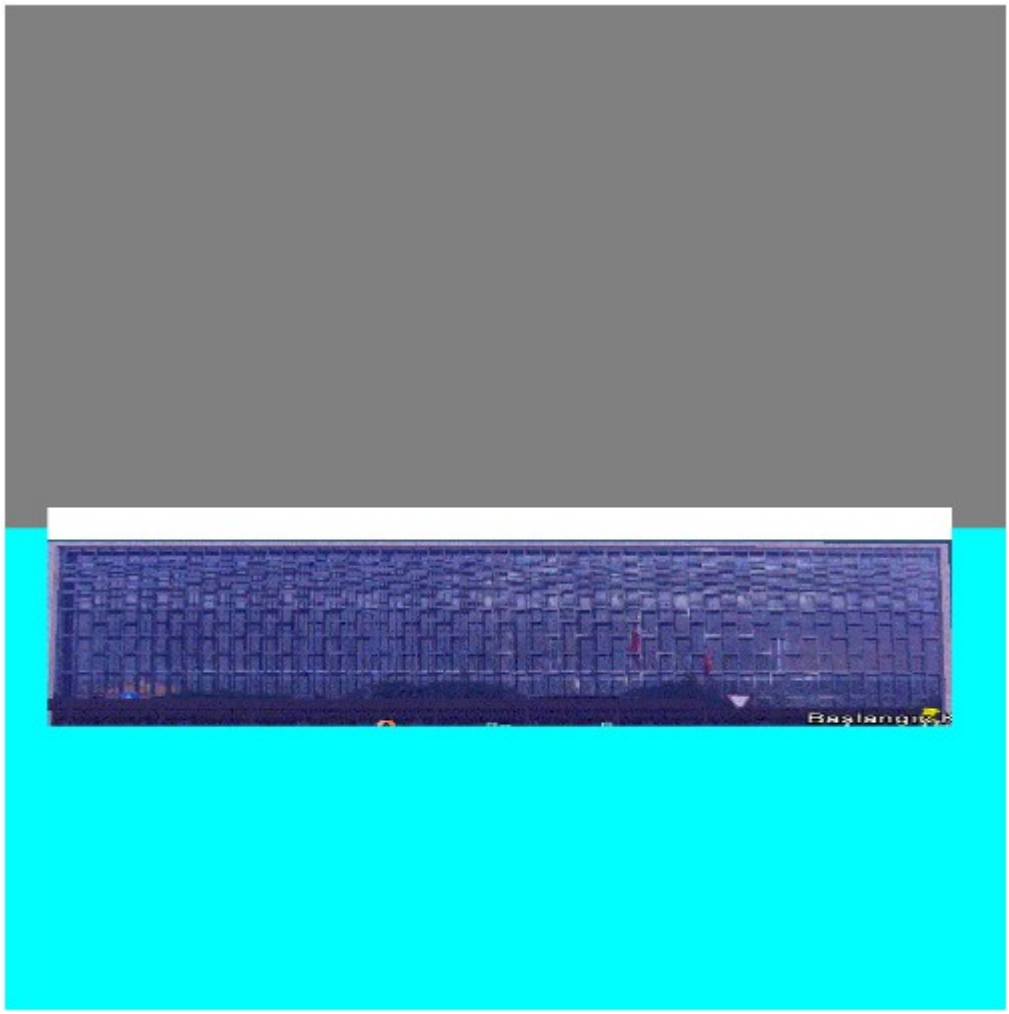
Sonraki iki satırda desen için özel ölçeklendirme parametreleri belirtilir. İlki WebGL'e resme göre ekranda büyük bir alanı kaplarken ne yapılacağını söyler; yani

nasıl ölçekleyeceğinin ipucunu verir. İkincisi ise resim büyük ama ekrandaki alan küçük olduğu zaman nasıl ölçekleme yapacağının ipucunu verir. Belirtilebilecek bir çok ölçeklendirme ipucu vardır; mesela NEAREST Orijinal resmi olduğu gibi kullanmayı belirtir, yaklaştıkça bloklu bir şekilde görünür. Buna rağmen yavaş makinelerde bile çok hızlı olması avantajına sahiptir. Burada NEAREST kullanılmamasının sebebi bu ipucunda desenin görüntülenmemesidir WebGL, OpenGL ES 2.0 niteliklerine uyumludur. İkinin katı olmayan desenler (2x2, 4x4, 8x8 vb boyutlu) desen filtrelemenin mips'e ihtiyacı olmadığı ve desen kaydırmasının CLAMP_TO_EDGE'e ayarlanmamış olduğu durumlarda gösterilmez. Desenin varsayılan parametrelerinde mips'e ihtiyaç vardır ve kaydırmaya ayarlanmıştır, bundan dolayı desen parametreleri düzgün bir şekilde ayarlanmalıdır; yani burada CLAMP_TO_EDGE'e ayarlanmalıdır. Çalışma sırasında bunu keşfetmek önemli bir zaman aldı. S ve T sırasıyla X ve Y koordinatlarını temsil eder. Bu da bittiğinde şu anki desene null değeri verilir; bu kesin olarak gerekli değildir, ama iyi bir alışkanlıktır; bir çeşit temizleme işlemidir.

4.3 Kullanıcı Testi

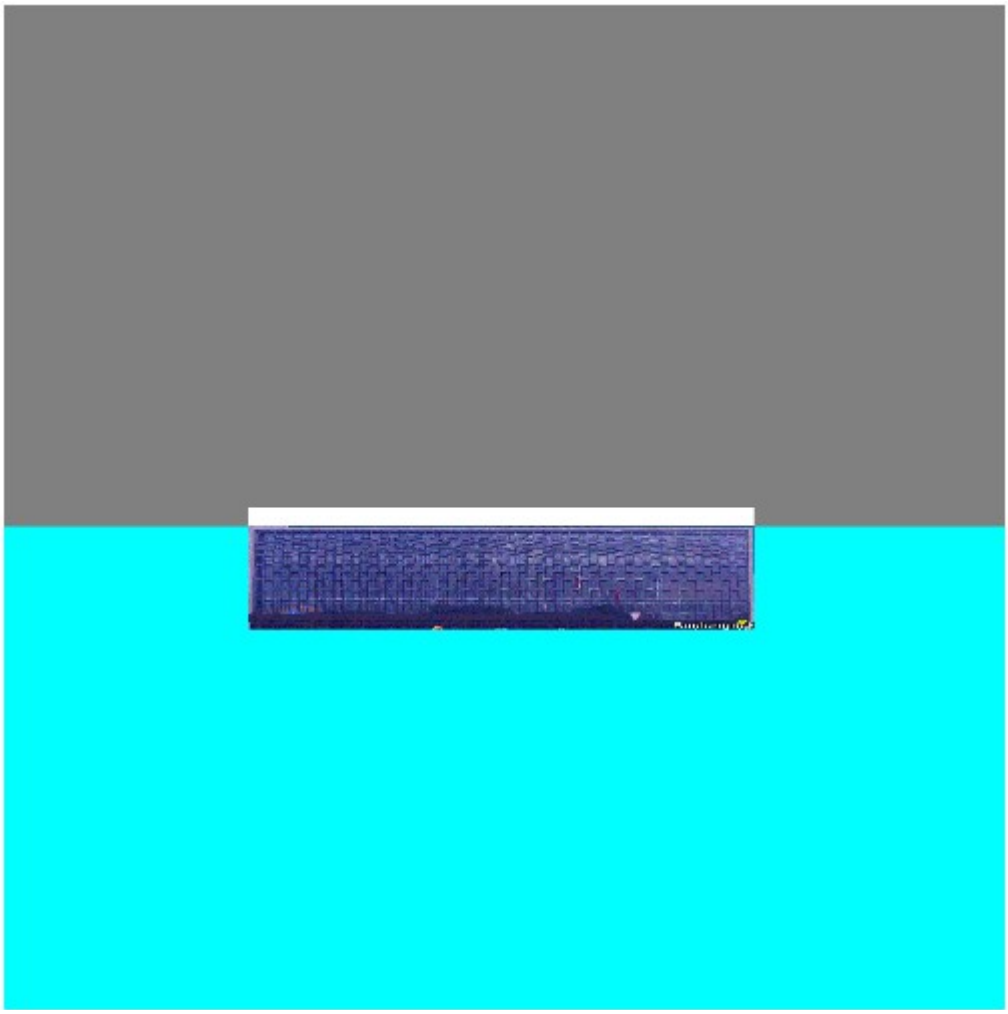
Kullanıcı testi bilinen harita uygulamalarında yapıldığı üzere fare ve klavye ile yapılabilmektedir. Bunlar ile harita üzerinde yakınlaşma, uzaklaşma, sağa, sola, ileri geri, yukarı aşağı komutları verilebilmektedir. Küçük bir alan için tüm şehir nesneleri önceden yüklenmektedir ve WebGL ile matris hesaplamaları yapılarak yeni görünüm kullanıcıya sunulmaktadır.

Uygulama ilk açıldığında aşağıdaki görünüm oluşmaktadır. Gökyüzü, yer ve Atatürk kültür merkezini temsil eden bir binaya ön cepheden bir bakış.



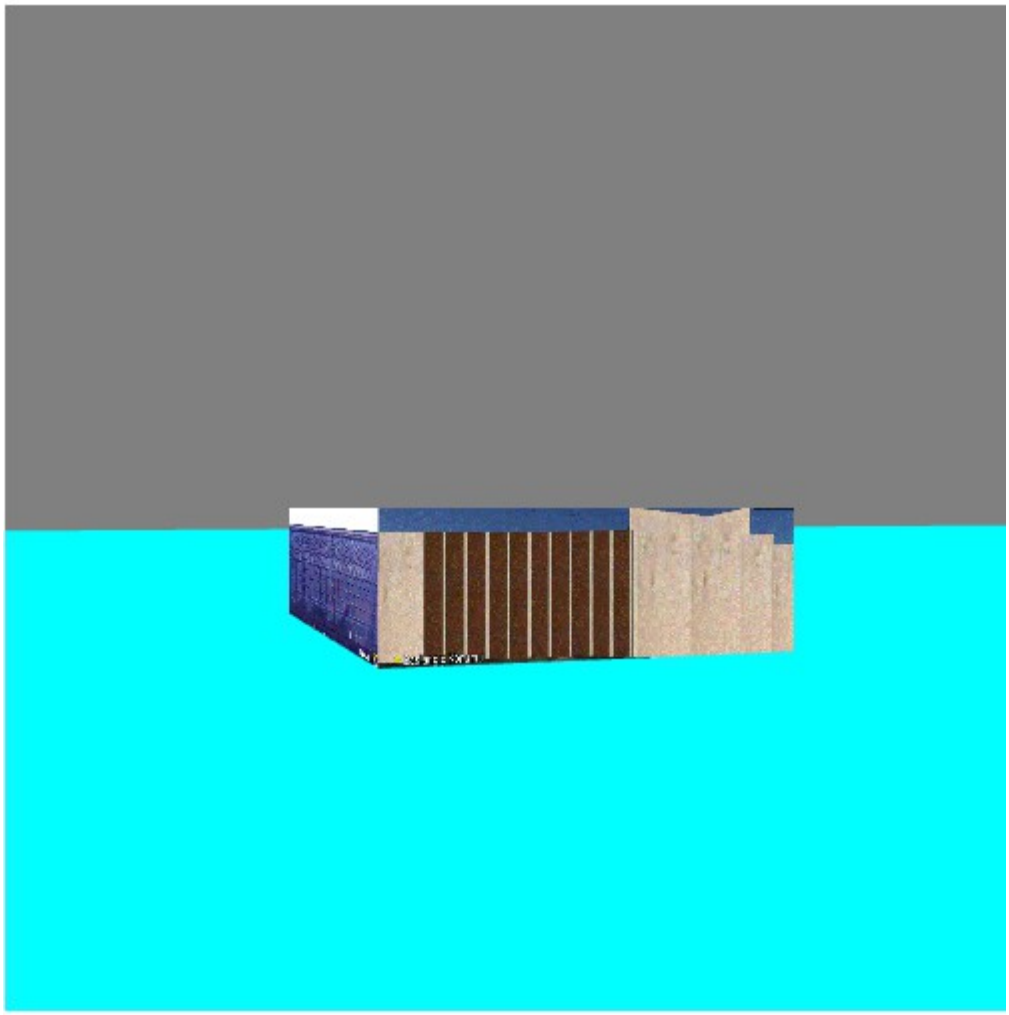
Şekil 25: Uygulama açıldığındaki ekran görüntüsü

Ekranda klavyeyi kullanarak bir miktar uzaklaşıldığında aşağıdaki ekran görüntüsü elde edilmektedir.



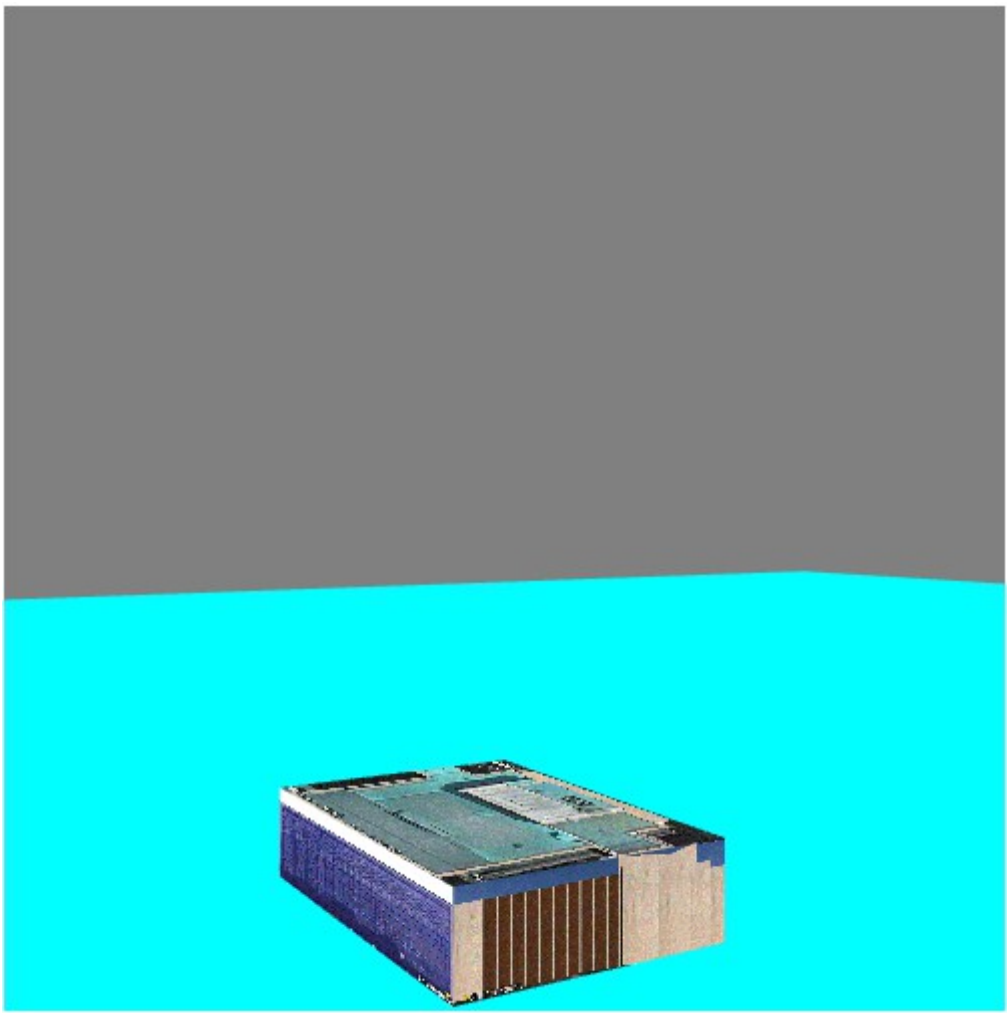
Şekil 26: Uygulamada nesneden uzaklaştığındaki görünüm

Uygulamada hem biraz uzaklaşıp hem de sağa kaydırma sonucu elde edilen görüntü aşağıdaki gibi olmaktadır.



Şekil 27: Uygulamada sahneyi sağa döndürünce oluşan görünüm

Uygulamadaki sahnede hem geri gidip hem sağa kayıp hem de biraz yukarı hareket edildiğinde aşağıdaki görünüm elde edilmektedir.



Şekil 28: Uygulamada binaya 3 cepheden çapraz bakan bir görünüm

4.4 Yeni Bir Bina Ekleme

Uygulamaya yeni bir bina eklemek için binanın köşe koordinatlarına ve cephe resimlerine ihtiyaç vardır. Bu bilgi elde yoksa Google Earth gibi bir uygulama ile harita üzerinden bulunabilir. Bu bilgi ile Building ve BuildingImage tablolarına kayıt eklenmesi gerekmektedir. Aşağıda bu uygulamadaki bina için eklenen bilgiler görünmektedir. Atatürk Kültür Merkezi binası için 3008 katı cisim tipindeki, 4327 Türkiye koordinat sistemindeki bina geometri bilgisi eklenmiştir. Sonrasında da bu binanın ön, arka, sol, sağ ve üst cepheleri için beş ayrı resim çekilmiş ve BuildingImage tablosuna BLOB veri tipinde kaydedilmiştir.

```
Insert into SYS.BUILDING (ID,NAME,BUILDINGTYPEID,GEOMETRY) values
(1,'Atatürk Kültür Merkezi','3',
MDSYS.SDO_GEOMETRY(3008,4327,NULL,MDSYS.SDO_ELEM_INFO_ARR
AY(1, 1007, 3),MDSYS.SDO_ORDINATE_ARRAY(-4.10368940, 0, -2.89877420,
4.20361440, 2, 2.99887750 )));

CREATE OR REPLACE DIRECTORY AKM_DIR as

'C:\Users\Seyfullah\Documents\Visual Studio
2012\Projects\UrbanRegeneration\UrbanRegeneration\Images\AKM_images';

DECLARE v_src_loc BFILE := BFILENAME('AKM_DIR', 'front.gif');
   v_amount  INTEGER;
   v_b       BLOB;
 BEGIN
   DBMS_LOB.OPEN(v_src_loc, DBMS_LOB.LOB_READONLY);
   v_amount := DBMS_LOB.GETLENGTH(v_src_loc);
   INSERT INTO SYS.BuildingImage VALUES (1, 206, 0, EMPTY_BLOB())
RETURNING IMAGE INTO v_b;
   DBMS_LOB.LOADFROMFILE(v_b, v_src_loc, v_amount);
   DBMS_LOB.CLOSE(v_src_loc);
   v_src_loc  := BFILENAME('AKM_DIR', 'back.gif');
```

```
   DBMS_LOB.OPEN(v_src_loc, DBMS_LOB.LOB_READONLY);
   v_amount := DBMS_LOB.GETLENGTH(v_src_loc);    INSERT INTO
SYS.BuildingImage VALUES (2, 206, 1, EMPTY_BLOB()) RETURNING IMAGE
INTO v_b;

   DBMS_LOB.LOADFROMFILE(v_b, v_src_loc, v_amount);
   DBMS_LOB.CLOSE(v_src_loc);
   v_src_loc  := BFILENAME('AKM_DIR', 'top.gif');

   DBMS_LOB.OPEN(v_src_loc, DBMS_LOB.LOB_READONLY);
   v_amount := DBMS_LOB.GETLENGTH(v_src_loc);
   INSERT INTO SYS.BuildingImage VALUES (3, 206, 2, EMPTY_BLOB())
RETURNING IMAGE INTO v_b;
   DBMS_LOB.LOADFROMFILE(v_b, v_src_loc, v_amount);
   DBMS_LOB.CLOSE(v_src_loc);
    v_src_loc  := BFILENAME('AKM_DIR', 'right.gif');

   DBMS_LOB.OPEN(v_src_loc, DBMS_LOB.LOB_READONLY);
   v_amount := DBMS_LOB.GETLENGTH(v_src_loc);
   INSERT INTO SYS.BuildingImage VALUES (4, 206, 4, EMPTY_BLOB())
RETURNING IMAGE INTO v_b;
   DBMS_LOB.LOADFROMFILE(v_b, v_src_loc, v_amount);
   DBMS_LOB.CLOSE(v_src_loc);
   v_src_loc := BFILENAME('AKM_DIR', 'left.gif');

   DBMS_LOB.OPEN(v_src_loc, DBMS_LOB.LOB_READONLY);
   v_amount := DBMS_LOB.GETLENGTH(v_src_loc);
   INSERT INTO SYS.BuildingImage VALUES (5, 206, 5, EMPTY_BLOB())
RETURNING IMAGE INTO v_b;
   DBMS_LOB.LOADFROMFILE(v_b, v_src_loc, v_amount);
   DBMS_LOB.CLOSE(v_src_loc);
 END;

 /

commit;
```

5 SONUÇ

Bu çalışmada REST MVC yapısı ile şehir modellerinin yeni bir teknoloji olan HTML5 üzerinde Canvas üzerine WebGL kullanarak çizilmesi üzerine çalışılmıştır. Veritabanı olarak Oracle Spatial kullanılmıştır. Verilerin ağ üzerinde hızlı iletilmesi için XML yerine JSON veri tipi seçilmiştir. Bu teknolojiler sayesinde bilgisayara bir program kurmadan tarayıcı üzerinde platform bağımsız olarak şehir üzerinde gezinebilme imkanı sağlanmıştır.

Var olan literatürde bu teknolojileri bir arada kullanan bir çalışma görülmemiştir. Bu anlamda bu çalışma bir ilk olma özelliği barındırmaktadır.

Bu çalışmanın devamı olarak CityGML3 standartlarına göre daha üst LOD seviyelerinde görselleştirme üzerine çalışılabilir. Böylece şehir nesneleri üzerinde ayrıntılı görünüm elde edilebilir ve kullanıcı deneyimi arttırılabilir. Şu anki çalışmada tüm şehir nesneleri bir defada yüklenmesi üzerine bir kurgu yapılmıştır. Daha büyük alanlardaki bir çok nesne için sadece belirli bir alandaki nesneler önce yüklenir, farklı bir alana geçildiğinde oradaki nesneler yüklenecek şekilde bir tasarım yapılabilir.

Bu çalışma sayesinde belediyeler, su, elektrik, telefon, internet, yol gibi altyapı sağlayıcıları için şebeke tasarımlarını bir tarayıcı üzerinden sunmak mümkün hale gelmektedir. Çok küçük veri paketleri ile nesnelerin görselleştirilmesi ve bunlar içinde gezinebilme imkanı oluşmaktadır. Bununla beraber verilerin ISO standartlarında tutulması ve varolan kaynaklardan sisteme alınıp kısa bir sürede kullanılabilmesine olanak sağlamaktadır.

KAYNAKLAR

[1] http://www.vagueware.com/top-10-architectural-design-software-for-budding-architects/ Top 10 Architectural Design Software for Budding Architects, Alınma Tarihi, 24.02.2014

[2] http://masters.donntu.edu.ua/2009/ggeo/grigoriva/library/article7.htm , TOWARDS UNIFIED 3D CITY MODELS, 20.07.2013

[3] Spatial Data Modelling for 3D GIS A. Abdul-Rahman, M. Pilouk

[4] http://www.w3schools.com/html/html5_intro.asp HTML5 Introduction,30.05.2013

[5] http://www.w3schools.com/html/html5_canvas.asp HTML5 Canvas, 30.05.2013

[6] http://www.w3schools.com/soap/ SOAP Tutorial, 30.05.2013

[7] http://www.w3schools.com/soap/soap_syntax.asp SOAP Syntax, 30.05.2013

[8] Fielding, Roy Thomas (2000), Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine

[9] https://en.wikipedia.org/wiki/Representational_state_transfer Representational State Transfer, 01.06.2013

[10] Elkstein, M. What is REST?., What is REST, 01.06.2013

[11] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html , Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC), 01.06.2013

[12]http://docs.oracle.com/cd/B10500_01/appdev.920/a96630/sdo_intro.htm#sthref45 , Oracle® Spatial User's Guide and Reference Release 9.2, 07.06.2013

[13] http://www.khronos.org/webgl/ , OpenGL ES 2.0 for the Web, 07.06.2013

[14] http://www.learningwebgl.com , Learning Web GL, 07.06.2013

[15] Oracle Spatial Giriş, Mustafa Özçetin (Uzman Araştırmacı)

[16] http://qdosmsq.dunbar-it.co.uk/blog/2012/05/installing-locator-or-spatial-on-11g/, alınma tarihi 29.12.2013

[17] http://www.ibm.com/developerworks/rational/library/dec04/bell/ UML basics: The component diagram, alınma tarihi 31.12.2013

EKLER

Ek 1 – SPATIO-SEMANTIC COHERENCE IN THE INTEGRATION OF 3D CITY MODELS

Alexandra Stadler, Thomas H. Kolbe

Artan sayıdaki uygulamalar 3B geometrik bilgiye dayanmaktadır. 3B geometriye ek olarak, bu uygulamalar özellikle karmaşık anlambilim bilgisi de gerektirmektedir. Uzaysal veri altyapıları bağlamında gerekli veri dağıtık kaynaklardan sağlanır ve konu ile alakalı olarak ve uzaysal olarak parçalıdır. 3B nesneleri basitçe birleştirmek kaçınılmaz olarak çatlaklara, tutarsızlıklara neden olur.Bu tutarsızlıklara örnek olarak bir dijital arazi modeli (DAM) ve 3B bina modellerinin farklı kaynaklardan birleştirilmesi sonucu ortaya çıkan uçan veya batan binalar verilebilir. Laurini'ye (1998) göre bu tutarsızlıkların nedeni katman parçalanmasıdır. Bu tanım farklı özellik sınıflarını içeren aynı bölgeyi kapsayan veri kümelerinin birleştirilmesi sırasında oluşan hataları açıklar. Buna karşın bölgesel parçalanma uzaysal olarak ayrık bölgeleri içeren veri kümelerinin birleştirilmesi sırasında oluşan hatalar ile uğraşır. Burada, tipik tutarsızlıklar sınırlarda üst üste çakışır veya aralarında boşluk oluşur, belediye sınırlarındaki DAM'leri buna örnek verilebilir.

1. TANITIM

Sanal 3B şehir modelleri gürültü haritalama, eğitim simülasyonu, felaket yönetimi, mimari ve şehir planlama gibi çevre simülasyonları ile alakalı artan sayıdaki görevleri gerçekleştirmek için uygulanmaktadır. (Shiode, 2001; Döllner et al., 2006) 3B geometri ve görünüm bilgisine ilave olarak bu uygulamalar karışık mantıksal bilgiye ihtiyaç duyar. Buna rağmen, ihtiyaç duyulan veri farklı kaynaklardan ve sıkça konu ile alakalı ve uzaysal olarak parçalıdır. Bundan dolayı verilen bir coğrafi bölge verisi kalite ve modellenen anlam yönlerinden farklılık gösterir. Bu durum internette bir çok coğrafik bilgi kaynaklarına erişim sağlayan uzaysal veri altyapıları bağlamında oldukça yaygındır.

Direkt 3B nsneleri birleştirmek kaçınılmaz olarak çatlaklara, ayrışmalara veya ayrıntı seviyesinde tutarsızlıklara neden olacaktır. Bu tutarsızlıklar için başta gelen

örnekler arasında bir dijital arazi modeli (DAM) ve 3B bina modelleri farklı kaynaklardan birleştiğinde 'uçan' veya 'batan' binaların olması gösterilebilir. Laurini'ye (1998) göre bu tip tutarsızlıklar katman parçalanmalarına aittir. Bu terim farklı özellik sınıflarına sahip farklı veri kümelerinin aynı alanı gösterdiğinde oluşan hataları belirtir. Buna karşın bölgesel parçalanma aynı özellik sınıflarına sahip farklı bölgelere ait farklı veri kümelerinin birleştirilmesi üzerine çalışır. Burada, tipik tutarsızlıklar sınırlarda üst üste çakışır veya aralarında boşluk oluşur, belediye sınırlarındaki DAM'leri buna örnek verilebilir.

Kusursuz bir veri entegrasyonu için yöntemlerin olması gereklidir. Şimdiye kadar veri entegrasyonu veri sağlayıcıları (haritalama ajansları vb.) tarafından yapılıyordu. Veri kümelerinin tutarlılığı kapsamlı olarak ve çoğu zaman elle uyumlulaştırılarak sağlanmıştır. İnsanlar geometriyi tanır ve kendiliğinden anlambilimi ilişkilendirir. Yerel tutarsızlıkları çözmek için bu ek bilgi doğal bir akla yatkınlık ile birlikte uygulanır. Web servisleri (Web Harita Servisi, Web Özellik Servisi, Groot ve McLaughlin'e bakınız) gibi yeni teknolojiler bağlamında kullanıcı bir çok uzaysal veri kaynağına kolaylıkla erişebilir ve bu bilgileri birleştirebilir. Sonuç olarak veri entegrasyonu veri sağlayıcılar tarafından sağlanamaz ve kullanıcının kendisi tutarlılıktan emin olmalıdır ki bu da büyük bir dezavantajdır.

Dağıtık veri kümelerinin anında birleştirilebilmesi için otomatik veri entegrasyonu için yöntemler geliştirilmelidir. Bu yöntemler değişik durumların mantıklı yorumlamasında insan yeteneklerine güvenemeyeceğinden, özellik anlambilimleri açıkça belirtilmelidir. Bu anlambilimleri işlemi iki yaklaşımla yapılabilir, ya kendiliğinden bilinen kuralları kullanarak veya açıkça verilen bağlantıları kullanarak (bağlantı noktaları, arazi kesişim çizgileri gibi). Anlamlandırma katmanında ne kadar fazla bilgi sağlanırsa geometik entegrasyonda o kadar az belirsizlik kalacaktır. Bu bilgilere dayanarak, birbirinden farklı içerikler için uyarlanabilir uyum süreçleri oluşturulabilir. Mesela bir kapı üzerinde durması için bir zemine ihtiyaç duyar. Böylece bir bina ve arazi birleştirildiğinde kapı olarak işaretlenen poligonun alt kenarının bir merdivene veya araziye değmesi gerekmektedir.Buna ek olarak anlam bilgisi geometrik ayarlamaların kısıtları hakkında da bilgi verir, mesela bir şehir mobilyasının taşınma işlemi sokağın taşınmasından çok daha olasıdır.

Bu yazıda 3B şehir modellerinin anlamsal ve uzaysal bilgilerinin yanı sıra uzaysal-anlambilim tutarlılık olarak anılan kendi yazışma yapısı analiz edilmektedir. Mevcut verinin kendi yapısal bölümlemesinde farklılık göstermesinden dolayı uzaysal ve anlambilim karmaşıklığın ortak senaryolarını 3B şehir modellerinin sunumu ve aktarılması için tasarlanmış bir geometrik uzaysal veri modeli olan CityGML bağlamında tanımlayacağız. İkinci bölüm, şehir modelleri hakkında genel olarak ve seçilen yönlerde bir kanı verecektir. Üçüncü bölüm, uzaysal-anlambilim uyumu açıklar ve yukarda belirtilen farklı uzaysal ve anlambilim karmaşıklığındaki senaryoları tanımlar. Sonra dördüncü bölüm veri doğrulama ve entegrasyonun faydalarını anlatır. Beşinci bölümde ilgili çalışma açıklanır ve son olarak altıncı bölümde yazı kısa bir özet ve öngörü ile bitirilir.

2. 3B Şehir Modelleri ve CityGML

Sanal 3B şehir modelleri, dünya yüzeyinin ve şehir alanlarına ait ilişkili nesnelerin dijital sunumudur. Bunlar, sonrasında ayrıntılı 3B şehir modellerine olan isteği oluşturan geniş kapsamlı uygulamaların hayata geçmesine olanak sağlar. Böyle modeller şehir nesnelerinin karmaşıklığını ve aralarındaki ikişkiyi yansıtmalıdır. Bir çok yönü kapsayan, gösteren veri, bir çok kaynaktan hali hazırda elde edilebilmektedir. Uygun bir kullanım için bu veri kümeleri uygulamaya özel modellere uyarlanmalıdır.

Görselleştirme, yüksek kalitede grafik sunuma ihtiyaç duyar ve mümkün olduğunca gerçekçi olmalıdır. Bu amaç için görünüm bilgisi ile birlikte geometri yeterlidir. Buna karşın, gürültü haritalama veya felaket yönetimi (Kolbe ve diğerleri 2005) gibi mühendislik uygulamaları karmaşık sorgulamalar ile çalışmaktadır ve ayrıntılı anlambilim ile analiz yapar. Mesela gürültü haritalamada ek olarak yol kaplama ve duvarların ses yalıtkanlığı gibi akustik veriler yüksek çözünürlüklü gürültü kirliliği haritalarının oluşturulmasına olanak sağlar (Czerwinski).

Uygulama katmanına dayanarak, uygun analizleri gerçekleştirebilmek için anlambilime ihtiyaç vardır. Heterojen ortamlarda birlikte çalışılabilirliği arttırmak için uzaysal ve anlambilim verisinden oluşan şehir modelleri için standartlaşmış veri

değiştokuş yöntemleri gereklidir.

2.1 CityGML

CityGML, sanal 3B şehir modellerinin depolanması ve değiştokuşu için XML tabanlı bir açık veri modelidir. GML3 için bir uygulama şeması olarak benimsenmiştir, Açık Geometrik Uzaysal Konsorsiyumu (OGC) tarafından uzaysal veri değiştokuşu için genişleyebilir uluslararası standart olarak yayımlanmıştır. Ana fikir temel özellik sınıfları, nitelikleri ve 3B şehir modelleri ontolojisine göre topolojik, anlam ve görüntü özellikleri dikkate alınarak ortak bir tanım elde etmektir (Gröger ve diğerleri, 2006). Bu farklı uygulama alanlarında aynı veri kümesinin tekrar kullanımını sağlayarak maliyet-etkin sürdürülebilir bakım için önemlidir.

Modelleme ilkesi bir özellik sınıflandırmasına ve anlambilim ve uzaysal olarak ayrıştırma işlemine dayanır. Bu ayrıştırmalar aşağıda belirtilen iki hiyerarşik yapıyı oluşturur. (Kolve ve Gröger, 2003'e bakınız)

CityGML'in anlambilim modeli binalar, DAM'leri, su kaynakları, ulaşım, bitki örtüsü, şehir mobilyası gibi 3B şehir modellerinde önemli özelliklerin sınıf tanımlamalarını içerir. Şekil 1 binaları tanımlamak için kullanılan anlambilimin küçük bir kısmını gösterir. Tüm sınıflar, uzaysal nesnelerin sunumunda ve bir araya getirilmesinde ISO 19109 ve GML3'de belirtilen temel sınıf olan "Özellik (Feature)" sınıfından türemiştir. Özellik, GML3'de eşleşen özelliklerin özelliklerine karşı gelen veritipleri (ISO/FDIS 19109, 2005) ile uzaysal ve uzaysal olmayan niteliklere sahiptir.

Şekil 1'den aşağıdaki çıkarımlar elde edilebilir.

- Bir bina özyinelemeli olarak bina parçalarından oluşur.

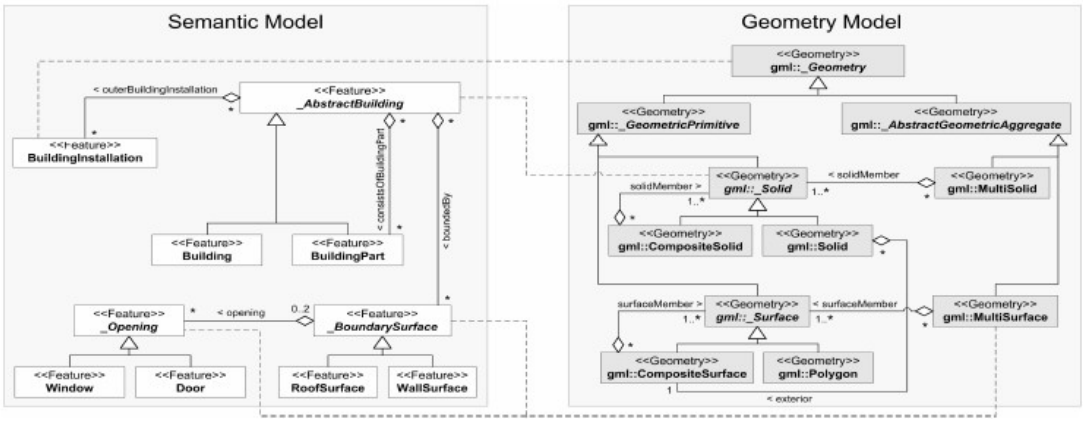
- Bir bina pencere ve kapıya sahip duvar, çatı gibi bir çok yüzey tarafından sınırlandırılabilir.

- Bir bina dış bina nesnelerine sahip olabilir.

- Hem anlambilim, hem geometri modelleri bir çok seviyede toplanmaya izin verir.

CityGML'in uzaysal özellikleri ISO 19107 standardı "Uzaysal Şema"'ya (Herrin, 2001) dayanan GML3'ün geometri modeli nesneleri tarafından sunulur. Aslında CityGML, GML3 geometri paketinin yalnızca bir kısmını kullanır.

GML3 geometri modeli ilkel öğelerini barındırır. Her bir boyut için, farklı bağlantı gereksinimlerini karşılamak için toplama veya karmaşık geometrileri oluşturabilirler. Toplama geometriler ilkellerin keyfi derlemesi olmasına rağmen, karmaşık geometriler sadece kendi sınırları boyunca topolojik olarak ilkelleri sunar.



Şekil 29: CityGML'de topoloji açıkça belirtilebilir. Uzayın her bir parçası ayrı olarak sadece bir kere modellenebilir ve sonra aynı geometriyi içeren tüm özellikler tarafından referans edilebilir. Bundan dolayı tekrarlamadan kaçınılabilir ve parçalar arasındaki açık topolojik ilişkiler korunabilir.

Dahası, Ayrıntı Seviyeleri (Levels Of Detail (LOD)) desteklenmektedir. Bir veri kümesinde aynı nesne 5 ayrık şekile kadar ve iyi tanımlı LOD olarak sade DAM'lerinden iç yapılarla beraber mimari modellere kadar aynı anda sunulabilir. Bu sadece belirli bir aralıktaki LOD'lar için geçerli özellik sınıfları sayesinde elde edilebilir. Mesela bina özellik sınıfı LOD 1'den 4'e kadar geçerlidir, buna karşın sınır yüzeyi özellği LOD 2'den 4'e kadar geçerlidir.

Böylece CityGML 3B şehir modellerini geometriye ve anlambilime bağlı olarak çeşitli karmaşıklık seviyelerinde sunmaya yeteneklidir. Bu da CityGML'in sunulabilir veri ve uygulamalar bağlamında bir alışveriş biçimi olması için esnek olmasını sağlamaktadır.

3. UZAYSAL-ANLAMBİLİM UYUMLULUĞU

ISO 191xx standartları ailesini geometrik uzaysal ile uğraşırken izlerken geometri ve anlambilim olarak iki ayrı yapı olduğu görülür. Yukarıda anlatıldığı gibi CityGML'de bu durum özellik ve geometri tiplerinin toplama hiyerarşileri olarak görülmüştür. Birbiri ile alakalı nesneleri birbirine ilişkilendirerek anlambilim ve geometri modellemesindeki uyumluluk temin edilebilir. Bu terimin anlamı aşağıda ayrıntılı şekilde açıklanacaktır.

Genelde uyumluluk, uyumun kalitesi veya durumu olarak ifade edilir, mantıksal, düzenli ve estetik olarak parçaların tutarlı ilişkileridir.

Bu tanıma göre geometri uzaysal bağlamında uyum, uzaysal ve anlambilim varlıkların tutarlı ilişkilerini tanımlar. Bu ilişkiler sadece yapısal benzerlik sırasında kurulabilen biçimde tanınır. Böylece, diğer bir ifadeyle eğer anlambilim ve geometri toplaması aynı yapıyı gösteriyorsa bunlar uyumlu olarak kabul edilir. Sadece bundan sonra anlambilim ve uzaysal bilgi iki açık fayda ile ilintili kullanılabilir :

- Geometrik nesneler ne olduklarını "bilirler".

- Anlambilim varlıkları nerede olduklarını ve uzaysal uzantılarının ne olduklarını “bilirler”.

Matematiksel bir mantıkta uzaysal ve anlambilim analizinde yapısal benzerlik iki yapı arasındaki tek biçimlilik ile ifade edililir. Temel olarak, somut model örneklerinden daha fazla birleştirme ilişkisi geometri hiyerarşisinden anlambilim hiyerarşisine (ve tersi) ilişkilendirilebilir, yüksekliği uyumun derecesini belirtir. Bu ilkeye dayalı belirli bir nicel ölçünün türevi bu çalışmanın konusudur.

Aşağıda anlambilim ve uzaysal karmaşıklığına göre 3B şehir modellerinin farklı durumlarını inceleyip, ayrıştıracağız. Bu daha çok geometrinin yapısal parçalarının anlamlı alt parçalara bölünmesi, bir hiyerarşi ve topolojik ilişkiler tanımlamasını ifade eder. Benzer bir şekilde anlambilim karmaşıklığı anlambilim bilgisinin yapısal alt bölümlere ayrılmasını ifade eder. Şekil 2'den 6'ya kadar olanlar

duvarlardan, pencerlerden kapı, çatı ve merdivenden oluşan bir binayı belirtir. Aşağıdaki resimler tüm örneklerde aynı şekilde görsel görünümü göstermektedir, ortadaki ve sağ taraftaki ağaçlar anlambilim ve uzaysal yapıları tanımlamak için göstermektedir. Kesikli çizgiler birbirine karşı gelen varlıklar ve yapıları eşlemek için kullanılmaktadır.

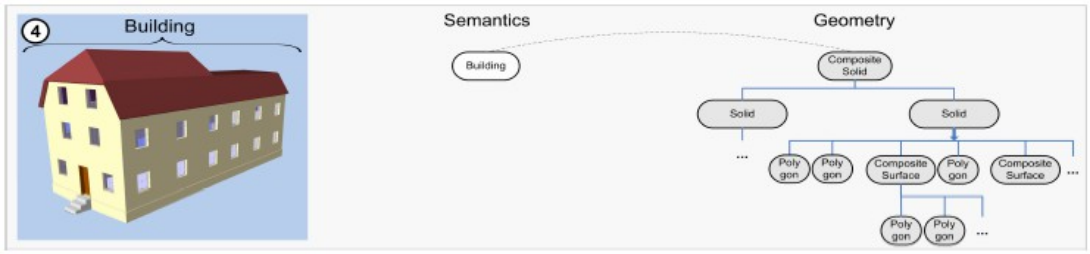
Durum 1 : Sadece geometri, anlambilim yok.

İlk durum (bkz 2. şekil) VRML, X3D, KML, U3D veya eski CAD geometri biçimleri gibi tipik modelleri açıklar. Bu 3B modeller genelde sahne grafiklerinde kullanılan yapılandırılmış geometrilerden oluşmaktadır (cf. Foley et al, 1995). Bunlar genellikle bilgisayar grafiklerinde ve BDT'da kullanılan 3B modelleme araçlarının tipik çıktılarıdır. Anlambilim nesne bilgisi içermediğinden herhangi bir uyum yoktur, çoğu zaman nesne Id'si bile yoktur.

Durum 2 : Sadece anlambilim, geometri yok. Bu durum oldukça sıra dışıdır. Şehir modeli belirli geometri uzaysal özelliklere sahiptir ama geometri bilinmemektedir veya erişlebilir değildir. Bu tip veri ekonomik, muhasebe veya tesis yönetim verilerinden elde edilmiş olabilir. Bu, gökten çekilen ve arazi lazer ve görüntü verisinden otomatik olarak şehir nesnelerinin yeniden yapılandırılmasındaki hipotezlerin üretilmesinde kullanılabilir. (bkz. Fischer gibi, 1998, Brenner, 2003)

Şekil 30: Yapılandırılmamış geometri ile basit nesne (Durum 3).

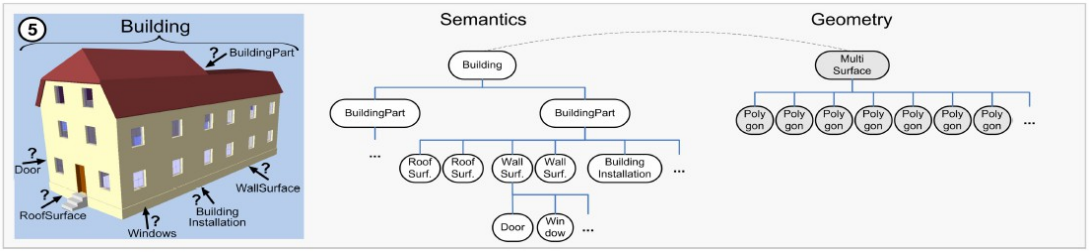
Durum 3: Yapılandırılmamış geometri ile basit nesneler (şekil 3) Nesneler geometrik özellikler ile sunulur. Her bir özellik yapılandırılmamış 3B yüzeylerden ve belki de bir dizi sayısal, uzaysal olmayan özellikleri olan bir uzaysal niteliğe sahiptir. Bu model, eğer geometri tarafından tanımlanan şekilbilim basitse yüksek derecede uyumlu, yapılandırılmamış geometri karmaşık bir şekli tanımlıyorsa az uyumlu olarak belirtilmelidir. Tipik uygulamaları Çok yama şekil dosyaları olarak ifade edilen modellerdir (cf. ESRI 1998).



Şekil 31: Karmaşık yapılandırılmış geometri ile basit nesne(Durum 4).

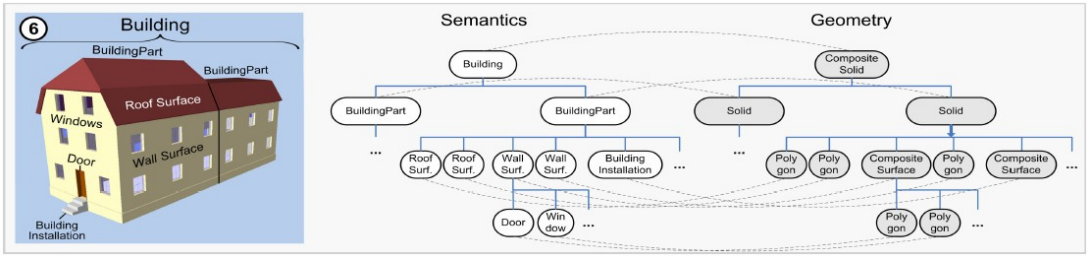
Durum 4: Yapılandırılmış geometri ile basit nesneler (şekil 4) Bu durumda, geometri ayrıntılı olmakla kalmayıp, aynı zamanda uzaysal ayrışımına göre yapılandırılmıştır. Buna rağmen, anlambilime gelince bir binanın varlığından daha fazla bir şey ifade edilmemiştir. Öyleki, alt geometriler arasında ilişkiler kurulamaz ve eksik anlambilim parçaları düşük seviyede bir uyumluluğa neden olurlar. Bu tip modeller fotogrametri nesne çıkarma araçları ile üretilebilir (cf. Gülch ve Müşer, 2001) veya lazer tarayıcı nokta bulutları veya üçgenleştirmeler gibi yapılandırılmamış geometrilerden elde edilen ilkel geometrilerden otomatik yeniden yapılandırmada üretilebilir. (cf. Marhall ve arkadaşları).

Durum 5: Yapılandırılmamış geometri ile karmaşık nesneler (şekil 5) Anlambilim ayrıntılıdır, yani binanın konu ile ilgili parçalarının neler olduğu bilinmektedir. Geometrinin ayrıntılı ancak yapılandırılmamış olmasından dolayı farklı toplama seviyelerinde anlambilim ve geometri ilişkisi kurulamaz. Bundan dolayı uyum, sadece 3B yüzeyler kümesi tarafından tamamen uzaysal olarak sunulabildiği ölçüde olduğu varsayılır. Böyle modeller Endüstri Temel Sınıfları (IFC, Adachie ve diğerleri, 2003; IFC hakkında daha fazla bilgi için beşinci bölüme bakınız.) veya tesis yönetim sistemleri gibi 3B Bina Bilgi Modellerinden (BIM) türemiş olabilir. Karmaşık uzaysal yapı, genellikle Yapıcı Katı Geometriye (CFG, bkz Foley ve diğerleri, 1995) bir dizi 3B sınır yüzeylerine dönüştürülmüştür.



Şekil 32: Karmaşık ve ayrıntılı anlambilim yapısına sahip nesne, ancak yapılandırılmamış ve ilişkilendirilmemiş geometri (Durum 5).

Durum 6: Yapılandırılmış geometri ile karmaşık nesneler. Hem anlambilim modeli hem de geometri karmaşık bir toplama olarak verilir. Tüm anlambilim bileşenleri geometrik bileşenler ile hiyerarşide aynı seviyede ilişkili olursa, yapı tamamen uyumlu varsayılır (şekil 12). Bu modeller hem anlambilimsel hem de geometrik olarak zengin olduğundan ve yapısal olarak izomorf olduğundan dolayı en yüksek seviyede yapısal kaliteye neden olur. Böyle modeller IFC gibi Bina Bilgi Modeli ile gelişmiş analiz ve dönüşümler ile elde edilebilir. Benner ve arkadaşları 2005 yılında anlambilimin ve IFC modelinin CSG geometrisinin uzaysal-anlambilim uyumlu B-Rep sunumuna CityGML'de nasıl eşleneceğini göstermiştir. Ayrıca binaların otomatik olarak çıkarılması bu kalite seviyesini hedeflemektedir (Fischer ve arkadaşları, 1998, Brenner, 2003).



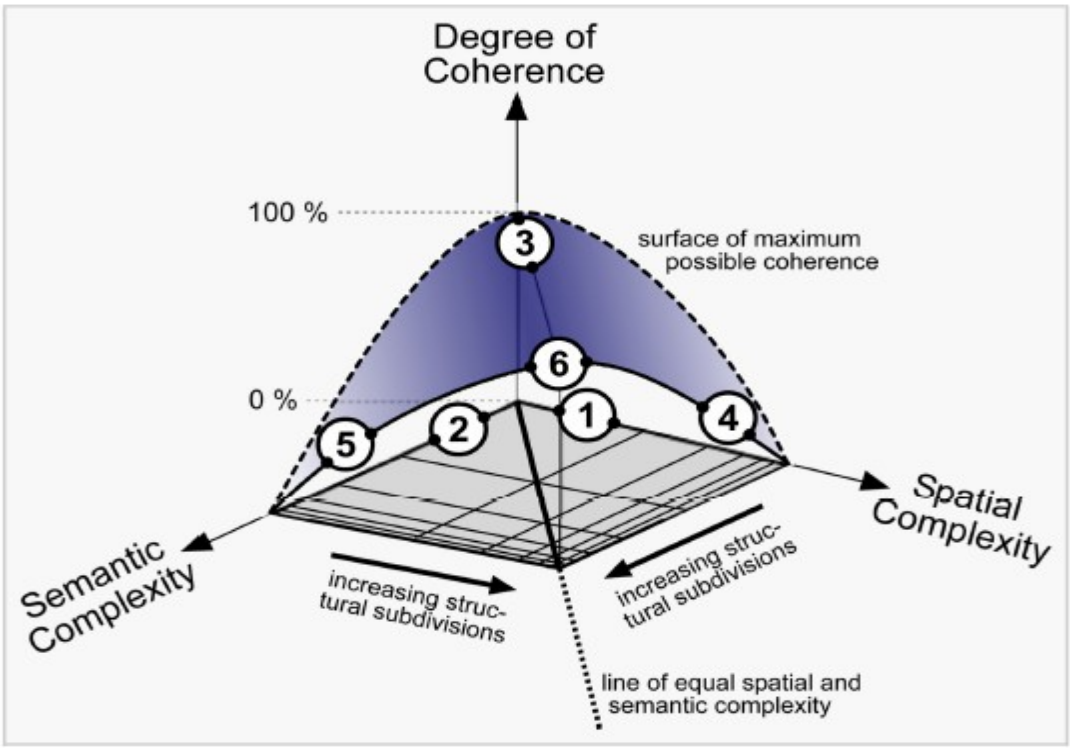
Şekil 33: Uzaysal-anlambilim olarak tamamen uyumlu karmaşık nesne yapısı (Durum 6).

Farklı veri kalitelerinde şehir modeli sunumunu gerçekleştirebilmek için CityGML durum 2'den durum 6'ya kadar olan durumları destekler. Durum 1 desteklenmez, çünkü CityGML ISO 19019 modelini temel alır ve bundan dolayı her zaman geometrileri anlambilim nesnelerine eşlemeye ihtiyaç duyar. Böylece modeller ilk başta yapılandırılmamış ve uyumsuz olarak sunulabilir ve adım adım elle ve otomatik niteliklendirme süreçleri ile iyileştirilebilir.

Yukarıdaki tanımlardan uzaysal ve anlambilim karmaşıklıktan öte iki binanın

uyumluluğu 3B şehir modellerinin önemli bir kalite yönüdür. Uyumun derecesini ölçmek için saptanabilen uygunluk miktarına göre hem anlambilim hem uzaysal altbölümlerin benzerliğini yansıttığı dikkate almalıyız. Bu miktar varlıkların miktarınca normalleştirilmeli, çünkü basit yapılandırılmış veri, karmaşık yapılandırılmış veri kadar uyumlu olabilir.

Eğer uzaysal ve anlambilim karmaşıklığı uyum derecesinde eşlenmişse (her biri bir eksende), bunlar bir uyum uzayına yayılırlar. Tüm özellik örnekleri bu uzayda kendi somut uzaysal ve anlambilim özelliklerine göre konumlanabilir. Tam uyumun sadece eşit uzaysal ve anlambilim karmaşıklığında elde edilebildiğinden dolayı, sonuç olarak çıkan maksimum mümkün olan uyumluluk yüzeyi bir yay gibidir. Şekil 7 geometri, anlambilim ve bunların uyumunu göstermektedir. Arkadaki kesikli çizgi uyum anlambilim ve geometri için tanımlanmadığı için açık bir aralık anlamına gelmektedir. 1'den 6'ya kadar olan etiketler durum 1'den 6'ya kadar olan konumları belirtir.



Şekil 34: Uzaysal ve anlambilim karmaşıklığına göre mümkün olan uyumluluk. Numaralar 1'den 6'ya kadar olan durumlara karşılık gelir.

ÖZGEÇMİŞ

31 Temmuz 1983 tarihi, Ordu İli Merkez ilçesi doğumluyum. İlk ve orta okulu Ordu'da okuduktan sonra, liseyi Adana'da, sonrasında Eskişehir Anadolu Üniversitesi Sivil Havacılık Yüksekokulu Uçak Gövde Motor Bakım lisans programını 2006 yılında ve Anadolu Üniversitesi İşletme Fakültesi, İşletme Bölümünü 2008 yılında bitirdim. Askerlikten gözden dolayı muafım. 2007 yılında Eskişehir'de 6 ay çalıştıktan sonra İstanbul'da farklı özel şirketlerde çalıştım. Halen (2013 Temmuz) İstanbul'da özel bir bankada yazılım uzmanı olarak çalışıyorum. 2011 yılında Beykent Üniversitesi Bilgisayar Mühendisliğinde yüksek lisans eğitimine başladım.

Özel ilgi alanlarım veritabanı, web ve masaüstü uygulamalarıdır.

İngilizceyi çok iyi derecede, Almanca, İspanyolca, Arapça dillerini de başlangıç seviyesinde bilmekteyim. Evli ve bir çocuk babasıyım.

Aday: Seyfullah Tıkız