SİMÜLASYON VE OYUNLAR İÇİN YÖNTEMSEL ŞEHİR ÜRETİMİ

PROCEDURAL CITY GENERATION FOR SIMULATIONS AND GAMES

BUĞRA YENER ŞAHİNOĞLU

DR. ÖĞR. ÜYESİ UFUK ÇELİKCAN
Tez Danışmanı

Hacettepe Üniversitesi
Lisansüstü Egitim-Öğretim ve Sınav Yönetmeliğinin
Bilgisayar Grafiği Anabilim Dalı için Öngördüğü
YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

2019

BUĞRA YENER ŞAHİNOĞLU'nun hazırladığı SİMÜLASYON VE OYUNLAR İÇİN YÖNTEMSEL ŞEHİR ÜRETİMİ" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR GRAFİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Haşmet GÜRÇAY
Başkan ..……

Dr. Öğr. Üyesi Ufuk ÇELİKCAN
Danışman ..……

Doç. Dr. Nergiz Erçil ÇAĞILTAY
Üye ....……..

Dr. Öğr. Üyesi Serdar ARITAN
Üye ……..

Dr. Öğr. Üyesi Burkay GENÇ
Üye ……..……

Bu tez Hacettepe Üniversitesi Bilişim Enstitüsü tarafından **BİLGİSAYAR GRAFİĞİ YÜKSEK LİSANS TEZİ** olarak ..... / ..... /..... tarihinde onaylanmıştır.

Prof. Dr. Pınar DUYGULU ŞAHİN
Bilişim Enstitüsü Müdürü

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

19 / 09 / 2019

BUĞRA YENER ŞAHİNOĞLU

YAYINLANMA FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan "***Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge***" kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H. Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

□ Enstitü / Fakülte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.

□ Enstitü / Fakülte yönetim kurulu gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren .... ay ertelenmiştir.

□ Tezim ile ilgili gizlilik kararı verilmiştir.

19 / 09 /2019

BUĞRA YENER ŞAHİNOĞLU

ÖZET

SİMÜLASYON VE OYUNLAR İÇİN YÖNTEMSEL ŞEHİR ÜRETİMİ

Buğra Yener ŞAHİNOĞLU

Yüksek Lisans, Bilgisayar Animasyonu ve Oyun Teknolojileri
Tez Danışmanı: Dr. Öğr. Üyesi Ufuk ÇELİKCAN
Eylül 2019

Bu tezin ana amacı simülasyon ve oyunlarda kullanılabilecek yöntemsel üç boyutlu şehirler üretmektir. Üretilen şehir içten dışa doğru zon adı verilen bölgelere ayrılacak, serbest kamera ve üstten bakış kamerası ile görüntülenip gezilebilecektir. Çalışmamızın örneğinde şehir en az üç bölgeye (merkez, merkeze yakın binalar ve banliyö) ayrılıp buna uygun şekilde oluşturulacaktır.

Bu tezde en az girdiyle en gerçekçi şehir modelini üretmek hedeflenmiştir. Çalışmamızın sonucunda çıkardığımız yaklaşımla alınan parametrelere göre yaratılmış ve düz bir araziye konuşlanmış bir şehir üreten bir uygulama geliştirilmiştir. Yaklaşımlarımızı Unity Oyun Motorunda test edecek olsak da geliştirilen yöntem bütün geliştirme ortamlarına uygun adapte olabilir.

Anahtar Kelimeler: yöntemsel üretim, şehir üretimi, şehir zonları üretimi, yol ağları üretimi, parselleme, bina üretimi

ABSTRACT

PROCEDURAL CITY GENERATION FOR SIMULATIONS AND GAMES

Buğra Yener ŞAHİNOĞLU

Master, Computer Animations and Game Technologies
Thesis Advisor: Asst. Prof. Dr. Ufuk ÇELİKCAN
September 2019

Main aim of this thesis is to produce procedurally generated three-dimensional cities that can be used in simulation and games. The produced city will be separated regions that called zones from the inner city to outer city and can be viewed and visited by the freelook and top-down camera. In the case of this thesis, the city will be divided into at least three regions (central, near-central buildings and suburbs) and a city will be created accordingly.

In this thesis, it is aimed to produce the most realistic city model with the least input. As a result of our study, an application has been developed that created a city on a flat terrain based on the given inputs. Although our approaches has been tested in the Unity Game Engine, the developed methods can be adapted to all development environments.

Keywords: procedural, generation, city generation, city zones, road network generation, parcelling, building generation

TEŞEKKÜR

Lisansüstü eğitim sürecim boyunca bu tezin ortaya çıkmasında bana yardımcı olan değerli hocam Dr. Öğr. Üyesi Ufuk ÇELİKCAN'a, bana desteklerini hiçbir zaman esirgemeyen ve her zaman yanımda olan annem Sebiha ŞAHİNOĞLU ve babam Alaeddin ŞAHİNOĞLU'na, her zaman bana moral veren her şeyimi paylaştığım ablam Tuğba ŞAHİNOĞLU'na, bana yol gösteren, akıl veren ve bu mesleğe girmemi sağlayan teyzem Pakize DAĞITAN ve abim olarak gördüğüm eniştem Murat DAĞITAN'a, ve bu süreç boyunca bana destek olan bütün arkadaşlarıma,

Sonsuz Teşekkürler...

Buğra Yener ŞAHİNOĞLU
Eylül, 2019

İÇİNDEKİLER

ÇİZELGELER

ŞEKİLLER

SİMGELER VE KISALTMALAR

Kısaltmalar

OBB	Oriented Boundary Box
AABB	Axis Aligned Boundary Box
BSP	Binary Space Partitioning
L-Sistemi	Lindenmayer Sistemi
IFS	Iterated function systems
2B	2 Boyutlu
3B	3 Boyutlu

1. GİRİŞ

Yöntemsel oluşturma herhangi bir şeyin belirli yaklaşımlar, matematik denklemleri ve ya metodlar kullanılarak oluşturulmasına denmektedir. Yöntemsel oluşturma 70'li yılların başında video oyunları başta olmak üzere bilgisayar üzerinde geliştirilen birçok uygulamada kullanılmaya başlanmıştır. Bilgisayarların ve yöntemsel çalışmaların matematiğe dayalı bir altyapısı olması bu alanda çalışma yapan bilim adamlarının yöntemsel oluşturma için yeni metodlar geliştirip uygulamalar yapabilmesine olanak sağlamıştır.

Bilgisayar dilinde yöntemsel oluşturma birşeyin el ile değil programlanarak oluşturulmasına denilir. 3B modellemeden oyun tasarımına, yapay zeka davranışından animasyona kadar birçok şey yöntemsel olarak oluşturulabilir. Özellikle büyük çapta olan ve üretilmesi saatler alan içeriklerin üretilebilmesi için yöntemsel oluşturma zamandan ve paradan kazanç sağlayan bir yöntem olarak görülebilir.

Yöntemsel oluşturma deterministik ve stokastik diye iki çeşide ayrılabilir. En yaygın yöntemsel oluşturma deterministik oluşturmalardır. Bu oluşturmalarda verilen girdiler daima aynı değeri döndürürler. Bilgisayarların daha kolay anlayıp daha çabuk sonuç ortaya çıkartabildikleri yöntemsel oluşturmalar da aynı zamanda deterministik olanlardır. Stokastik oluşturmalar verilen girdilerin her zaman rastgele sonuçlar döndürdüğü yöntemsel oluşturma çeşitleridir. Çıktılarının tahmin edilemez oluşu bu tarz oluşturma metodlarını daha ilgi çekici kılmaktadır.

Yöntemsel oluşturmanın avantajları:

- Küçük dosya boyutu
- Büyük boyutlarda yapılar oluşturabilme
- Daha az tahmin edilebilir oyun için rastgelelik

olarak gösterilebilir.

{Yöntemsel Oluşturma Teknikleri}
Yöntemsel oluşturma tekniklerinin öncüleri olarak fraktal oluşturmalar gösterilebilir. Fraktallar belirli biçimlerin tekrarlaması ile oluşan yapılardır. Fraktallar matematiksel denklemler aracılığı ile oluşturulabilirler.

Fraktallar doğada sıklıkla gözlemlenmektedir. Kar taneleri, bulutlar, ve çoğu bitki fraktal yapıda bulunmaktadır. Her bir fraktal içerisinde orjinalinin küçük kopyalarını bulundurur. Bu sebeple fraktallar ayrıntılı yapılardır.

Fraktallar kendi içlerinde tekrar eden yapılardan oluştuğundan onları oluşturan algoritmalarda özyinelemeli algoritmalar olmuştur. Algoritmanın yinelenme sayısı ne kadar fazla ise o kadar detaylı fraktallar oluşturulabilir.

Doğal yapılar fraktallar ile oluşturulabilir. Bunun en bilinen örneklerinden birisi Barnsley, M. F tarafından geliştirilen IFS metodu ile oluşturulmuş eğrelti otu fraktalıdır (Şekil 1.1).

Şekil 1.1. Barnsley Eğrelti Otu[1]

Fraktallar ile nehirler, araziler ve yol ağları oluşturulabilir[2,3].

Bir diğer yöntemsel geliştirme tekniği de L-Sistemi adı verilen tekniktir[4]. L-Sistemi Biyolog A. Lindenmayer tarafından biyolojik gelişim için oluşturulmuş bir matematik grameridir. Bu sistem başlangıçta alg gibi basit organizmaların büyüme paternlerini incelemek için geliştirilmiştir.

L-Sistemi ile yapılan çalışmalar bu tekniği zaman içerisinde geliştirerek bu tekniğin farklı alanlarda kullanılmasını sağlamışlardır.

L-Sistemi temel olarak karakter bazlı kural değişimine dayanır. Şekil 1.2'de L-Sistemi tekniğinin çalışma mantığını anlatan bir sözde kod gösterilmiştir. Her bir kural bir karakter dizisi olarak ifade edilir. Bir başlangıç karakteri ve karakterler için aksiyomlar belirlendikten sonra istenilen sayıda iterasyon sağlanarak bir karakter dizisi oluşturulur. En sonunda bu dizi soldan sağ doğru okunarak dizinin tarif ettiği yapıya bakılır.

L-Sisteminin bilgisayar grafiğinde uygulama yollarından biri kaplumbağa grafiği adı verilen yöntemdir. Bu yöntem ile hayali bir kaplumbağa L-Sistemi tekniği ile oluşturulan bir karakter dizisinin tarif ettiği pozisyonlara arkasında bir iz bırakarak sırasıyla gider.

Kaplumbağa grafiği ile uygulanmış bir L-Sistemi tekniği ile 2B bir ağaç oluşturma algoritması Şekil 1.3 ile, bu algoritma sonucu oluşan bir ağaç Şekil 1.4 ile gösterilmiştir.

L-Sistemi Algoritmasının Sözde Kodu

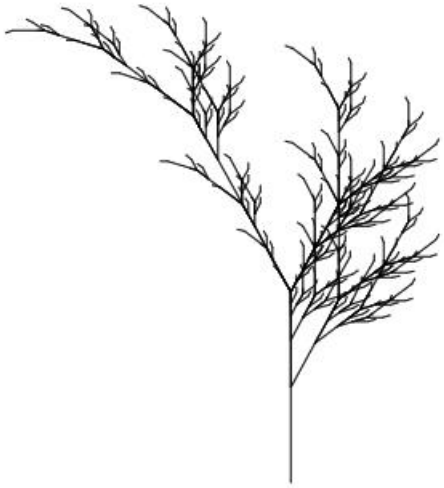
```
1.  Fonksiyon Lsistem(list)
2.      karakter1 için Aksiyom1'i belirle;
3.      karakter2 için Aksiyom2'yi belirle;
4.      Başlangıç karakterini belirle ve list'e ekle
5.      i ← 0;
6.      Döngü(i<belirlenen iterasyon sayısı)
7.          Döngü(her bir karakter için list'i dolaş)
8.              Koşul(karakter=karakter1)
9.                  karakter←Aksiyom1
10.             Koşul Sonu
11.             Koşul(karakter=karakter2)
12.                 karakter←Aksiyom2
13.             Koşul Sonu
14.         Döngü Sonu
15.         i++;
16.     Döngü Sonu
17.     Geri döndür list
18. Fonksiyon Sonu
```

Şekil 1.2. L-Sistemi Algoritmasının Sözde Kodu

Kaplumbağa grafiği L-Sistemi Algoritması

1. Aksiyom kurallarını belirle
2. F: 1 birim ileri
3. +: α derece sola dön
4. -: α derece sağ dön
5. [: bulunan pozisyonu bir yığına ata
6.]: yığına son eklenen pozisyonu döndür
7. Aksiyomları belirle
8. X → F-[[X]+X]+F[+FX]-X;
9. F → FF;
10. w ← X;
11. α ← 22.5°
12. iterasyon ← 5;
13. Aksiyomlara göre döngü sayısı kadar L-Sistemini çalıştır.

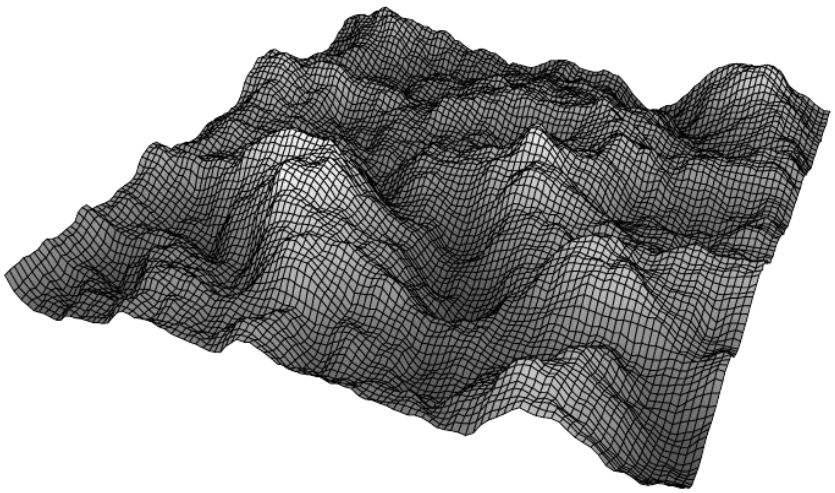
Şekil 1.3. Kaplumbağa grafiği L-Sistemi Algoritması



Şekil 1.4. Kaplumbağa grafiği ile oluşmuş bir ağaç[22]

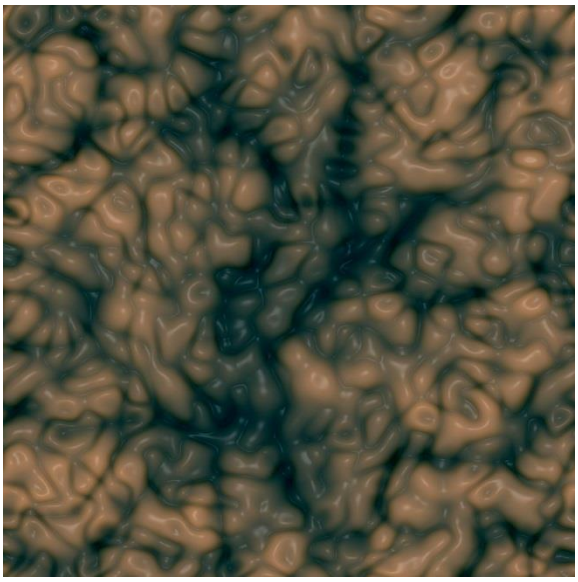
Yöntemsel oluşturmanın başka bir tekniği de gürültü fonksiyonlarından yararlanmaktır.

Gürültü, tipik olarak bir dizi veya matriste tutulan bir rasgele sayı dizisidir. Gürültü fonksiyonları basitçe bu sayı dizisinin herhangi bir yerindeki elemana ulaşmak için dizinin indeks ya da indekslerini parametre olarak alan fonksiyonlara denir.



Şekil 1.5. Perlin gürültüsü ile elde edilmiş bir arazi[7]

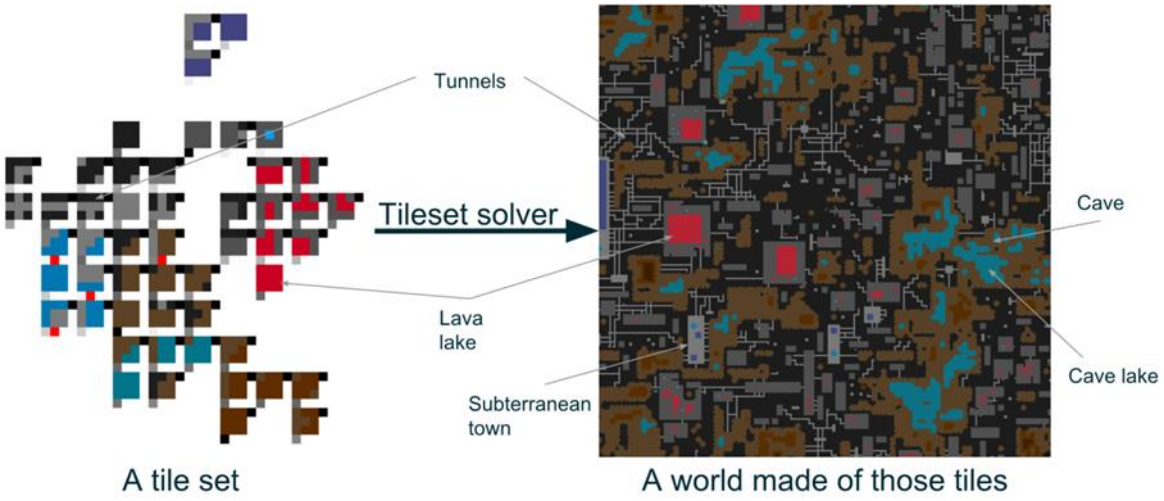
Gürültü fonksiyonlarının en ünlülerinden birisi Perlin gürültü fonksiyonudur. Perlin gürültüsü 2B bir gürültüdür. En başta kaplama oluşturmak için kullanılan (Şekil 1.6) bu fonksiyon, zamanla arazi modellemelerinde (Şekil 1.5) [5] ve hatta hacimsel kaplama oluşturulmasında (Şekil 1.7) [6] kullanılmıştır.



Şekil 1.6. Perlin gürültüsü ile oluşturulmuş bir beyin kaplaması[8]

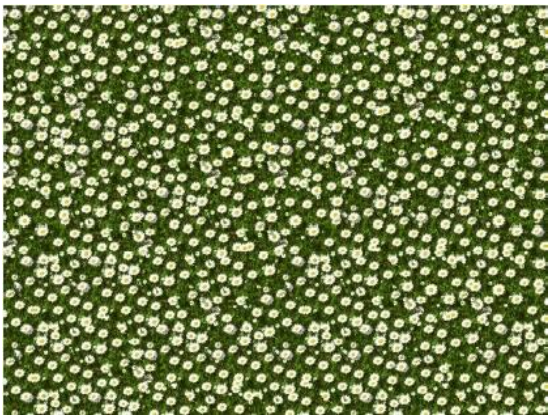
Şekil 1.7. Perlin gürültüsü ile üretilmiş hacimsel kaplama ile kaplanmış bir mermer vazo modeli[23]

Bir başka yöntemsel oluşturma tekniği döşeme tekniğidir. Döşeme tekniği ilk olarak 2B oyunlarda tekrar eden hücreler oluşturup bu hücreler ile haritalar ve level tasarımları yapmak için kullanılmıştır (Şekil 1.8).



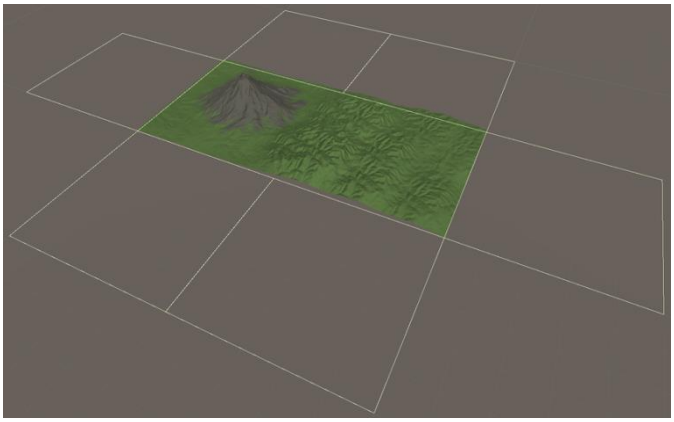
Şekil 1.8. Döşeme tekniği ile harita oluşturma[24]

Döşeme tekniği aynı zamanda detaylı ve çok katmanlı kaplamalar ve resimler oluşturmakta kullanılmıştır. [9] (Şekil 1.9)



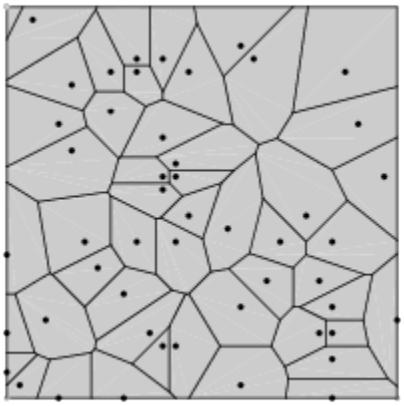
Şekil 1.9. Wang Döşeme tekniği ile oluşturulmuş bir resim[9]

Döşeme tekniği sayesinde ayrıca büyük ölçekli araziler de oluşturulabilmektedir. [10]
Unity oyun motoru son versiyonlarında bu teknikle bağlantılı komşu araziler yapılabilmesine olanak sağlamıştır (Şekil 1.10).



Şekil 1.10. Unity Oyun motorunda kullanılan döşeme arazi tekniği[10]

Bir başka yöntemsel oluşturma tekniği ise Voronoi diyagramı kullanmaktır. Voronoi diyagramı bir alanın dışbükey çokgenlere bölünmesi suretiyle oluşan diyagrama denmektedir (Şekil 1.11).



Şekil 1.11. Bir Voronoi Diyagramı[25]

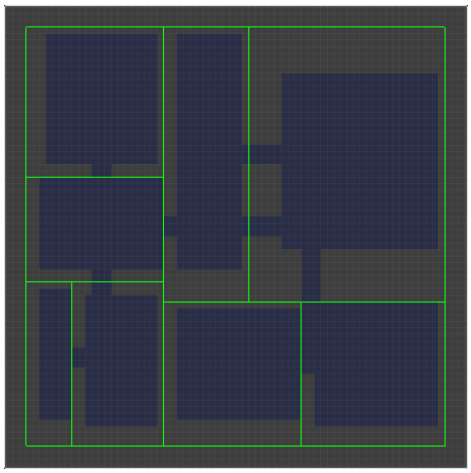
Voronoi diyagramını oluşturan her bir sınır çizgisi komşu noktalar arasında eşit uzaklıkta bulunmaktadır.

Voronoi diyagramını oluşturmak için farklı yöntemler ve parametreler kullanılarak çok çeşitli hücresel desenler oluşturulabilir.

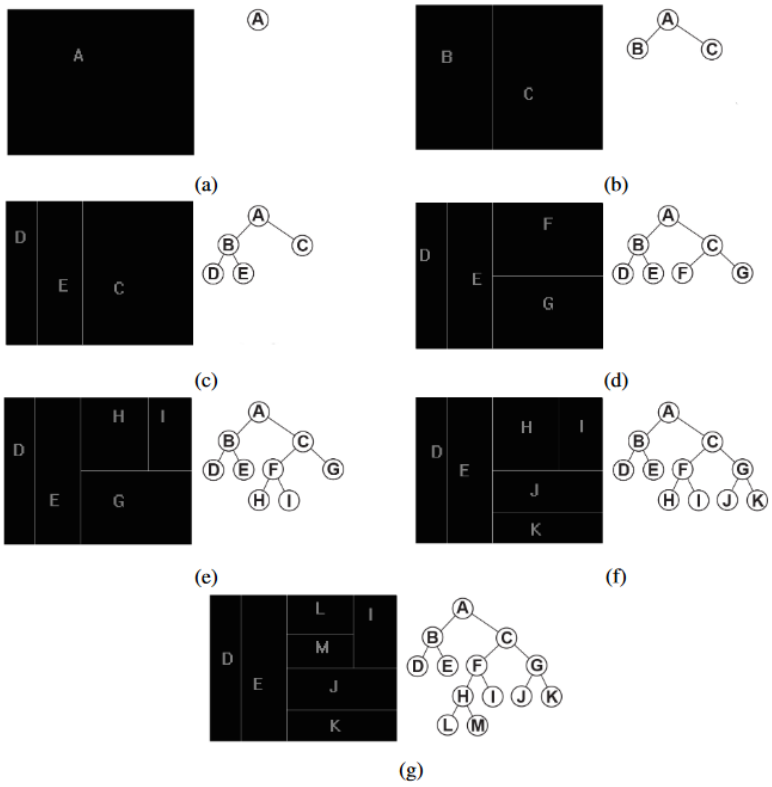
Voronoi diyagramı oluşturmaya yarayan yöntemlerden bazıları; Sweep Line algoritması, Delaunay üçgenselleştirmesi, Worley Algoritması ve Fortune'nun algoritması olarak gösterilebilir.

Yöntemsel oluşturma için kullanılan başka bir teknik ise BSP algoritmasıdır [11]. BSP algoritmasında alan özyinelemeli olarak iki parçaya bölünür. Bu bölümleme işlemleri BSP ağacı adı verilen bir ikili ağaç yapısında gösterilebilir (Şekil 1.13).

BSP algoritması birçok oyunda zindan üretiminde kullanılmaktadır (Şekil 1.12). Bu algoritma ayrıca şehir parsellemesinde, ve statik poligonlar için gerçeğe yakın zamanlı gölge oluşturulmasında da kullanılmıştır[12, 13].



Şekil 1.12. BSP ile zindan oluşturma[26]



Şekil 1.13. BSP ile alan bölümleme ve BSP ağacı[11]

Yöntemsel oluşturma tekniklerinden bilgisayar oyunları da sıkça yararlanmışlardır. Günümüzde rol yapma oyunlarında sıklıkla görülen zindan tasarımları ya da 80'li yılların

popüler oyunlarının birçoğunun level tasarımları yöntemsel oluşturma teknikleri ile geliştirilmiştir.

{Araştırmanın amacı ve Önemi}
Günümüzde şehirler en yaygın yerleşim alanlarını oluşturmaktadır. Bu durum simulasyonlarda ve oyunlarda da kendini göstermektedir. Bu sebeple çeşitli oyunlarda kullanmak için şehirler el ile dizayn edilmektedir. Ancak bu büyük çaplı projeler için bu durum zaman ve para kaybına yol açmaktadır. Bu çalışma ile oyunlar ve simulasyonlar için merkezden banliyolara kadar genişleyen kamu alanlarına ve yol ağına sahip bir şehri yöntemsel olarak oluşturmak için bir algoritma geliştirildi.

Bir şehri yöntemsel olarak oluşturmak kullanıcıların her seferinde farklı yapıdaki şehirlerde, farklı bir deneyim edinmesini sağlayabilir. Ayrıca bilimsel çalışmalar için her seferinde farklı datalar ortaya koyabilir. Yöntemsel olarak üretilecek şehirler, sentetik veri gereksinimlerinde kullanılabilecek şehirlerin sayısını ve çeşitliliğini artırarak bu alandaki çalışmaya katkıda bulunacaktır.

Dünyadaki birçok kentin yerleşim planı olan ızgara planı yapısı, bu şehirlere benzer şehirlerin prosedürel olarak yaratılabileceği düşüncesini kafamızda oluşturdu. Bu nedenle, gerçekçi ve modern şehirler üretebilecek bir yazılım geliştirmek için çeşitli makaleleri ve algoritmaları inceleyerek bu çalışmaya başlanmaya karar verilmiştir.

Bu çalışmanın temel amacı simülasyonda ve oyunlarda kullanılabilecek 3 boyutlu şehirler üretmektir. Üretilen şehir, içeriden dışarıya doğru bölgelere ayrılacak, serbest uçuş kamerası ve yukarıdan bakış kamerası ile izlenebilecek ve ziyaret edilebilecektir.

Bu çalışmada oluşturulan şehirler içerden dışarıya doğru 3 zondan oluşmakta. Şehir merkezi, merkeze yakın binalar ve banliyö. Gösterilen yapı sabit sayıda zonlardan oluşsa da zon sayısı parametrik olarak değiştirilebilecek şekilde kodlanmıştır.

Araştırmamızın bir sonucu olarak, benzer çalışmalarda üretilen şehirlerin, gerçek şehirlerde olduğu gibi tarihi veya doğal gelişimlerinin sonucu oluşan zon adını verdiğimiz katmanlı yapıyı içerecek şekilde üretilmediği görülmektedir. Ender olarak üretilen bölge mantıklı yöntemsel şehir üretim algoritmaları ya bölgeleri merkeze bağlı bir şekilde dağıtmış ya da bir alanı düz bir biçimde bölgelere ayırmıştır. Ve bu olay, oluşan şehirlere yapay bir görünüm kazandırmaktadır. Dünyada iç bölgelerde yer alan şehirler incelendiğinde, kentlerin doğal gelişmeleri nedeniyle kent merkezlerinin içeride oldukları ve dışarıya doğru daha kırsal bir yapıya kaydıkları görülmektedir.

Bu çalışmada Catmull-Rom splinleri ile zone sınırlarının oluşturulması, OBB tabanlı bölümleme, rastgele yürüyüş algoritması, zon tabanlı bölümleme algoritması, ve perlin-gürültüsü algoritması gibi birkaç yöntem geliştirilmiş ve modifiye edilmiştir.

2. LİTERATÜR TARAMASI

Procedural City Layout Generation Based on Urban Land Use Models isimli çalışmada (Groenewegen vd, 2009) şehirler bölgeleri belirli parametreleri sağlayan yerlere yerleştirilmesi suretiyle oluşturulmuştur. Şehir sınırları arazi üzerinde diametrik bir biçimde verildikten sonra merkezden dışarıya doğru ana yollar çizilmiş, rastgele dağılıma göre aday lokasyonlar belirlenip bu aday lokasyonların en iyileri seçilerek bölge lokasyonları oluşturulmuştur. Şehir bölge lokasyonları tarafından parçalara ayrılarak bir Voronoi diyagramı oluşturulmuş ve bu diyagrama bir gürültü fonksiyonu uygulanarak şehre daha gerçekçi bir görünüm kazandırılmıştır[14].

Urban pattern: Layout design by hierarchical domain splitting isimli çalışmada (Yang vd., 2013) sokak ağını ve parsel yerleşimlerini oluşturmak için Akış Çizgisine Dayalı Bölme (Streamline-based Splitting) ve Şablona Dayalı Bölme (Template-based Splitting) algoritmaları birlikte kullanılmıştır. Bu algoritmalarla verilen bir alan önce Akış Çizgisine Dayalı Bölme algoritması kullanarak parçalarına ayrılmış, sonrasında bulunan alanlar Şablona Dayalı Bölme algoritmaları ile bina parsellerine bölünmüştür[15].

Citygen: An interactive system for procedural city generation isimli çalışmada (Kelly vd, 2007) şehir üretimini 3 parçaya bölmüştür. Birincil yol ağının oluşturulması, ikincil yol ağının oluşturulması ve bina üretimi. Yönsüz düzlemsel çizgelerle temsil edilen yollar bitişik listelerle tutulmuştur. Birincil yollar için iki adet çizge oluşturulmuştur. Bu çizgelerden birisi topolojik yapıyı tutarken diğeri arazi üzerindeki asıl yolu tutmaktadır. İkincil yollar için ise birincil yollar kullanılarak şehir hücreleri oluşturulmuş ve bu hücrelerin ağı üzerinde L-Sistemi algoritması kullanılarak hareket edilip yolların büyümesi sağlanmıştır. Binalar ise basit şekillere kaplama ve çıkıntı haritası(Normal Map) atılarak oluşturulmuştur[16].

Modeling of RL-Cities isimli çalışmada (van der Zee vd, 2012), zonlar gerçek şehir haritaları taranarak oluşturulmuş ve yol ağı için L-Sistemi algoritması kullanılmıştır[17].

Template-based generation of road networks for virtual city modeling isimli çalışmada (Sun vd, 2002) yol ağları şablon kullanılarak üretilmiştir. Populasyon tabanlı şablonun kuralları için Voronoi diyagramı kullanılmış, rastal ya da radyal şablonlar için ise L-sistemi kullanılmıştır[18].

Procedural modeling of cities isimli çalışmada (Parish vd., 2001) yol ağları genişletilmiş L-sistemi ve kendinden duyarlı L-Sistemi algoritmaları ile oluşturulmuş, yol ağlarının böldüğü alanları özyinelemeli bir algoritma kullanarak en uzun kenarlarından istenen eşik değerine ulaşıncaya kadar bölmek sureti ile bina alanı parselleri elde edilmiştir. Binaları oluşturmak için de gene L-Sistemi algoritması kullanılmıştır[19].

Procedural generation of parcels in urban modeling isimli çalışmada (Vanegas vd, 2012), şehri parsellerine bölümlemek için birkaç metoddan bahsedilmiştir. Dış iskelete dayalı

bölümleme (Skeleton-based Subdivision) 3 aşamada gerçekleşmiştir. İlk bölünme girdi olarak alınan çokgenin kenarlarına bitişik yüzler alınarak gerçekleştirilmiştir. İlk bölünme ile oluşan diagonal kenarlar dilimleme işlemi ile kaldırılmış oluşan dış şerit ya da dış iskelet eşit parçalara ayrılarak parseller oluşturulmuştur. Bahsedilen başka bir algoritma ise OBB Bölümleme algoritmasıdır. Bu algoritmada alan OBB içerisine alınıp OBB'nin kısa kenarına paralel bir şakilde yarıya bölünmüştür. Bu işlem özyinelemeli bir fonksiyon ile istenildiği kadar tekrarlanıp parsellerin oluşması sağlanmıştır[12].

Instant architecture isimli çalışmada (Vonka vd, 2003), şehirdeki binaların dış yüzeylerini yöntemsel olarak oluşturabilmek için bölünmüş gramer (split grammar) diye adlandırılan bir metod geliştirilmiştir. Bu metodta binanın dış yüzeyine atanacak pencere, balkon kapı gibi yüzeyler bir sözcük ya da harfle ifade edilmiş, Bu yüzeyin bölünmesi ile oluşan elementlere atanmıştır[20].

3. YÖNTEM

Bir şehri yöntemsel olarak oluşturmak 3 parçada incelenebilir.

- Yol ağlarını oluşturmak
- Alanları parsellere bölmek
- Binaları üretmek

Yöntemsel şehir üretiminde bu aşamalar istenildiği zaman istenildiği kadar kullanılabilir.

Tezimizde kullandığımız bir şehri oluşturmak için tasarlanan algoritma 5 parçaya bölünebilir.

- Şehirdeki zonların sınırlarını belirlemek
- Arsaları (adalar, bloklar ve parseller) oluşturmak
- Birincil ve ikincil yolları oluşturmak
- Kamusal alanları oluşturmak
- Binaları üretmek ve eklemek

Bu parçalar birbiri içine geçen farklı metodları barındırmaktadır. Zonların sınırları kapalı Catmull-Rom Splinleri ile oluşturulurken, şehir arsalarının oluşturulması işlemleri algoritmada ardarda olmayan ancak sırayla çalışan 3 metod ile gerçekleşmektedir. Bu metodlar sonucu oluşan arsalara sırasıyla

- Ada
- Blok
- Parsel

adları verilmiştir.

Oluşturulan şehrin birincil yolları çalışma örneğimizdde ızgara modelinde tasarlansa da geliştirilen metodların büyük bir çoğunluğu ,bazı zamanlar ufak modifiyelere ihtiyac duyarak, her türlü alanda çalışabilmektedir.

Algoritmamızın aşamaları aşağıdaki gibidir.

- Zon Sınırlarını Oluşturmak ve Zonları Belirlemek
- Birincil Yol Ağını ve Adaları Oluşturma
- Adaları Zon Bazlı Bölümleme Algoritması Kullanarak Bloklara Bölme
- Kamu Alanlarının Hesaplanması ve Yerleştirilmesi
- Blokları OBB-Rastgele Bölümleme Algoritması ile Parsellerine Bölme
- Şehir Merkezindeki İkincil Yol Ağlarının Çizge Tabanlı Rastgele Yürüyüş Algoritması ile Hesaplanması
- Yöntemsel Bina ve Parsel Modellerinin Oluşturulması
- Hazır Binaların Bina Parsellerine Yerleştirilmesi

3.1. Zon Sınırlarını Oluşturmak ve Zonları Belirlemek

Karasal yapılardaki şehirler incelendiğinde bu şehirlerin içten dışa doğru büyüyen bir yapıda olduğu görülmektedir. Bu yapıda en iç kısımda gökdelenlerden oluşan bir şehir merkezi mevcutken dışa doğru daha müstakil evler yer almaktadır. Bu zonlar şehrin tarihler boyunca büyümesi sonucu oluştuğu için bunların sınırları düzgün bir biçimde oluşmayacaktır. Yaptığımız araştırmalar sonucunda daha önceki makalelerde şehri düzgün bir biçimde belirli alanlara bölen metodlarla karşılaşılsa da, bahsettiğimiz şekilde zon mantığı ile bölge oluşturan algoritmalara rastlanılmamıştır.

Bu tezin çalışmasında zonların sınırlarını oluşturmak için Catmull-Rom splinleri kullanılmıştır. Catmull-Rom splinleri polinomsal eğrilerin uç uca eklenmesiyle oluşan yapılara denmektedir. Catmull-Rom splini elde etmek için kullanılabilecek eğrilerden birisi Hermit eğrisidir. Bir Hermit eğrisini oluşturmak için aşağıda listelenen bilgiler yeterlidir:

p_0 => Eğrinin başlangıç noktası
p_1 => Eğrinin bitiş noktası
v_0 => Eğrinin başlangıç noktasından geçen tanjant vektörü
v_1 => Eğrinin bitiş noktasından geçen tanjant vektörü

Birbiriyle bağlantılı olan $\{v_0, v_1\}$ vektörleri ve $\{p_0, p_1\}$ noktaları kullanılarak oluşturulan Eş. 3.1'deki polinomsal denklem yardımıyla Hermit eğrisi elde edilmektedir. Bu denklemdeki t parametresi 0 ile 1 arasındadır. t=0 başlangıç noktasını verirken, t=1 ise bitiş noktasını vermektedir. Eğri üzerindeki herhangi bir nokta t parametresine 0 ile 1 arası bir değer verilerek bulunabilir.

$$P(t) = (1 - 3t^2 + 2t^3)p_0 + t(1 - t)^2 v_0 + (3t^2 - 2t^3)p_1 - t^2(1 - t)v_1 \qquad (3.1)$$

Hermit eğrilerini birleştirip bir Catmull-Rom splini oluşturmak için birleşim noktalarına teğet geçen birleşim vektörünün hesaplanması gerekmektedir. Birleşim vektörünün hesaplanması Eş. 3.2 ile yapılır. Buradaki τ değeri bağlantının keskinliğini değiştirir.

$$V_i = \tau(p_{i+1} - p_{i-1}) \qquad (3.2)$$

Zonların sınırlarını oluşturmak için Hermit eğrileri kullanılarak kapalı bir Catmull-Rom splini elde edilmiştir. Her bir Hermit eğrisinin başlangıç noktası, başka bir Hermit eğrisinin bitiş noktasına ve bitiş noktası da başka bir Hermit eğrisinin başlangıç noktasına bağlandığı için, başlangıç ve bitiş noktalarından geçen bir tanjant vektörü oluşturmak yerine bütün birleşim noktalarında birer birleşim vektörü oluşturarak Hermit eğrisi için gerekli vektörler elde edilmiştir.

Her bir zon sınırının hesaplanması için, öncelikle zon sınırını oluşturan kapalı Catmull-Rom eğrisinin birleşim noktalarının pozisyonları belirlenmelidir. Her bir birleşim noktasını pozisyonu merkezden belirli minimum ve maksimum iki değer arasında bir uzaklıkta ve

kendisinden önceki birleşim noktası ile merkezde α derecede bir açı yapacak şekilde belirlenir.

Kapalı Catmull-Rom Splinini oluşturan Hermit Eğrilerinin birleşim noktalarının hesaplanması

```
1. birlesimNoktaları[];
2. uzaklik ← 0;
3. radyanDegeri ← 0;
4. i← 0
5. Döngü (i < birlesimNoktaları.uzunlugu)
6.     uzaklik ← RastgeleDeger(Rmin, Rmax);
7.     radyanDegeri ← ((360 / birlesimNoktaları.uzunlugu) * i) * PI / 180;
8.     x ← uzaklik*Cos(radyanDegeri);
9.     z ← uzaklik*Sin(radyanDegeri);
10.    birlesimNoktaları[i] ← Vector3(x,0,z);
11.    i++;
12. Döngü Sonu
```

Şekil 3.1. Kapalı Catmull-Rom Splinini oluşturan Hermit Eğrilerinin birleşim noktalarının hesaplanmasının sözde kodu

Şekil 3.1'de gösterilen sözde kodda bulunan radyanDegeri her bir birleşim noktasının bir önceki kesişim noktası ile merkezde yaptığı α açısının radyan cinsinden gösterilmiş halidir. Algoritma zon sınırı için değişken sayıda birleşim noktası oluşturulmasına izin vermektedir. Ancak bu tezin çalışmasında birleşim noktası sayısı 12 olarak sabitlenmiştir. Böylece iki birleşim noktası arasındaki α açısının değeri 30° olarak belirlenmiştir.
Kapalı Catmull-Rom splinindeki her bir birleşim noktalarından geçen birleşim vektörleri Eş. 3.2 ile hesaplanmıştır. Bu hesaplamanın sözde kodu Şekil 3.2'deki gibidir.

Şekil 3.2'de görüldüğü gibi birleşim vektörlerinin hesaplanmasında kullanılan Eş. 3.2 formülündeki τ(teta) değerine, bağlantı noktalarındaki keskinliğin her iterasyonda değişmesini sağlamak amacı ile, 0.2 ile 0.8 arasında rastgele bir değer verilmiştir.

Bu tezin çalışmasında bir Hermit eğrisi oluşturmak için gerekli birleşim noktaları ve birleşim vektörlerini tutmak için HermiteClass isimli bir sınıf oluşturulmuştur. Her bir Hermit eğrisi için hesaplanan birleşim noktaları sırayla ikişerli gruplar halinde kullanılarak kendileriyle alakalı birleşim vektörleri ile birlikte HermiteClass tipinde nesnelerde tutulmuştur.

Her bir zon sınırı için oluşturulan HermiteClass nesnelerinin listesi, zon değeri ve bu zonun iç ve dış sınırlarını oluşturan her bir Hermit eğrisi üzerinde bulunan noktaların pozisyonlarının listesi ZoneInfo isimli bir sınıfta tutulmuştur. Zonun dış sınırını oluşturan Hermit eğrileri üzerinde bulunan noktaların pozisyonlarının listesinin hesaplanması Şekil 3.3'deki sözde kodda gösterilmiştir.

Kapalı Catmull-Rom Splinini oluşturan Hermit Eğrilerinin birleşim vektörlerinin hesaplanmasını sağlayan sözde kod

```
1.  Fonksiyon BirleşimVektorleriHesapla(birlesimNoktaları[])
2.     birlesimVektorleri[];
3.      teta ← RastgeleDeger (0.2f, 0.8f);
4.      bitisNoktasi ← birlesimNoktaları[1];
5.      baslangicNoktasi ← birlesimNoktaları[birlesimNoktaları.uzunluk – 1];
6.     birlesimVektorleri[0] ← teta * (bitisNoktasi – baslangicNoktasi);
7.      i ← 1;
8.      Döngü (i < birleşimNoktaları.uzunluk-1)
9.          teta ← RastgeleDeger (0.2f, 0.8f);
10.           bitisNoktasi ← birlesimNoktaları[i+1];
11.           baslangicNoktasi ← birlesimNoktaları[i – 1];
12.           birlesimVektorleri[i] ← teta * (bitisNoktasi – baslangicNoktasi);
13.           i++;
14.     Döngü Sonu
15.     teta ← RastgeleDeger (0.2f, 0.8f);
16.     sonIndeks ← birlesimNoktaları.uzunluk – 1;
17.     bitisNoktasi ← birlesimNoktaları[0];
18.     baslangicNoktasi ← birlesimNoktaları[birlesimNoktaları.uzunluk – 2];
19.     birlesimVektorleri[sonIndeks] = teta * (bitisNoktasi – baslangicNoktasi);
20.     Geri Döndür birlesimVektorleri[];
21.  Fonksiyon Sonu
```

Şekil 3.2. Kapalı Catmull-Rom Splinini oluşturan Hermit Eğrilerinin birleşim vektörlerinin hesaplanmasını sağlayan sözde kod

Şekil 3.3'de gösterilen sözde kodda bulunan tPeriod değeri Eş. 3.1 denklemi ile elde edilen Hermit eğrisi üzerindeki noktaların sıklığına etki etmektedir. tPeriod değeri ne kadar küçük ise Hermit eğrisi üzerinden alınan noktaların sayısı o kadar artacak, dolayısıyla eğriyi temsil eden listede tutulan noktalar ile zon sınırı çizimi yapıldığı zaman orjinal Hermit eğrisine daha yakın bir sonuç çıkacaktır. tPeriod değerinin 1'e yakın verilmesi durumunda Hermit noktaları birbirine uzak çıkacak ve bu noktaların arasını doldurmak için ek bir doğru denklemine ihtiyaç duyulacaktır.

Hermit eğrisi üzerinden daha fazla sayıda nokta alınması için tPeriod değeri 0.001 olarak belirlenmiştir. Bu sayede bir Hermit eğrisi üzerindeki 1000 noktanın değeri tutulmuştur. Bir zon sınırı için oluşturulan kapalı Catmull-Rom eğrisinde 12 birleşim noktası belirlendiğinden her bir zon sınırını temsilen 12000 nokta sistemde tutulmaktadır.

Hermit eğrileri üzerinde bulunan noktaların pozisyonlarının listesinin hesaplanmasını sağlayan sözde kod

```
1.  tPeriod ← 0.001f;
2.  Fonksiyon HermitNoktaları(ZoneInfoNesnesi)
3.      Döngü (ZoneInfoNesnesi içindeki HermiteClass listesinde tutulan her bir
        HermiteClass hc objesi için)
4.          t ← 0;
5.          Döngü (t <= 1)
6.              HermitNoktasi ← (1 – 3t^2 + 2t^3)hc.dots[0] + t(1 – t)^2hc.vecs[0] +
                (3t^2 – 2t^3)hc.dots[1] – t^2(1 – t)hc.vecs[1];
7.              ZoneInfoNesnesi.BorderListesi.Ekle(HermitNoktasi);
8.              t ← t + tPeriod;
9.          Döngü Sonu
10.     Döngü Sonu
11. Fonksiyon Sonu
```

Şekil 3.3. Hermit eğrileri üzerinde bulunan noktaların pozisyonlarının listesinin hesaplanmasını sağlayan sözde kod

Zonların sınırlarını ve sınırların içerisinde kalan alanların bilgilerini tutmak için bir adet matris oluşturulmuştur. En dıştaki zonun sınırını oluşturan kapalı Catmull-Rom splininin birleşim noktalarından birbirlerine x ekseni ve z ekseni üzerinde ayrı ayrı en uzak olan noktaların uzaklıkları hesaplanmış ve bu değerler, sınırların matris sınırlarına temasının engellenip şehrin çevresini saran bir boş alan oluşturulması için 2 ile çarpılarak oluşturulacak matrisin boyutları belirlenmiştir.

Matrisi oluşturan her bir hücrede,

- Hücrenin pozisyonu
- Zon değeri,
- Sınır hücresi olup olmadığının bilgisi
- Zon değeri atamasının gerçekleşip gerçekleşmediği bilgisi

tutulmaktadır. Matris üzerindeki her bir hücrenin boyutu 1 br^2 olarak belirlenmiştir.
Matrisin merkezi (0, 0, 0) noktasına konuşlandırılarak, matristeki tüm hücrelerin pozisyonlarının x, y ve z değerlerinin tam sayı olması sağlanmıştır. ZoneInfo sınıfında tutulan sınır noktaları listesindeki noktaların pozisyonlarının x, y ve z değerleri de kendilerine en yakın en büyük tam sayı değerlerine yuvarlanmış ve bu değerler kullanılarak sınırlar matrisin gerekli hücrelerine yerleştirilmiştir. Zon sınırlarını matrise yerleştiren algoritmanın sözde kodu Şekil 3.4'deki gibidir.

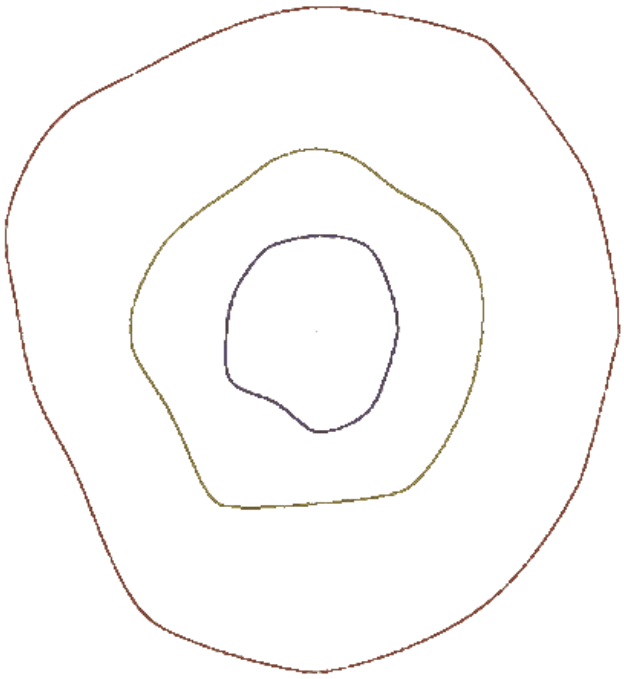
Zon sınırlarını matrise yerleştiren algoritmanın sözde kodu

```
1.  Fonksiyon ZonlarıMatriseEkle(ZoneInfoList, Matris[,])
2.      i←0
3.      Döngü(i< ZoneInfoList.ElemanSayısı)
4.          MatristeSınırOlustur(i, ZoneInfoList[i].ZoneDegeri, ZoneInfoList, Matris);
5.          i++;
6.      Döngü Sonu
7.      Geri Döndür Matris[,]
8.  Fonksiyon Sonu
9. Fonksiyon MatristeSınırOlustur(index, ZoneDegeri, ZoneList, Matris[,])
10.     Döngü (ZoneList[index] objesinin sınır listesinde bulunan her bir
                v vektörü için)
11.         i←AşağıYuvarla(v.x)+AşağıYuvarla(Matris.Eni/2);
12.         j← AşağıYuvarla(v.z)+AşağıYuvarla(Matris.Boyu/2);
13.         IF(i>0 ve j>0)
14.            IF (Matris[i,j].Kontrol = false)
15.               Matris[i,j].ZoneDegeri = ZoneDegeri;
16.               Matris[i,j].Kontrol = true;
17.               Matris[i,j].Sınır = true;
18.            Koşul Sonu
19.         Koşul Sonu
20.     Döngü Sonu
21.     Geri Döndür Matris[,]
22. Fonksiyon Sonu
```

Şekil 3.4. Zon sınırlarını matrise yerleştiren algoritmanın sözde kodu

Oluşan Zon sınırlarının görünümü Şekil 3.5'deki gibidir.

Matrise zon sınırları yerleştirildikten sonraki aşama, matris üzerindeki her hücrenin zon değerlerini belirlemektir. Bunun için önce Flood-Fill algoritması (Şekil 3.6) denenmiştir. Ancak bu algoritmanın özyinelemeli yapısından dolayı yığın aşırı yüklenmiş, bu sebeple sistem hafızası dolarak sistemi kapatmıştır. 4 yönlü yığın kontrollü Flood-Fill algoritması sınır uzaklığı 100 ile 200 birim arasında değişen tek bir zonu doldurmak için kullanıldığında da işlem yaklaşık 90 saniye sürmüştür.



Şekil 3.5. Zon sınırlarının görüntüsü

Özyinelemeli Flood-Fill algoritmasının sözde kodu

```
1.  Fonksiyon FloodFill(Nod, sınırX, sınırY, DegisimDegeri)
2.      IF(Nod.Deger = DegişimDegeri)
3.          Geri Dön
4.      IF(sınırX >= Nod.En veya sınırY >= Node.Boy)
5.          Geri Dön
6.      IF(sınırX < 0 veya sınırY < 0)
7.          Geri Dön
8.      Nod.Deger← DeğişimDeğeri
9.     FloodFill(1 sağdaki nod, sınırX, sınırY, DegisimDegeri);
10.    FloodFill(1 soldaki nod, sınırX, sınırY, DegisimDegeri);
11.    FloodFill(1 yukarıdaki nod, sınırX, sınırY, DegisimDegeri);
12.    FloodFill(1 aşağıdaki nod nod, sınırX, sınırY, DegisimDegeri);
13. Fonksiyon Sonu
```

Şekil 3.6. Özyinelemeli Flood-Fill algoritmasının sözde kodu

Matris üzerindeki hücrelerin zon değerlerini belirlemek için Hücre Ekseni Sınır Kontrolü algoritması (Şekil 3.7) geliştirilmiştir. Bu algoritmada seçilen hücrenin x ve z eksenleri boyunca ilerleyerek zon sınırlarına ya da matris sınırlarına ulaşılıp hücrenin zon değerleri belirlenmiş ve hücrenin kendisi ve eksenlerinden sınırlara doğru giderken uğranılan hücreler belirlenen değerle doldurulmuştur. Eğer hücrenin tüm eksenlerinden zon sınırlarına ulaşıldıysa zon değeri ulaşılan zon sınırlarından en büyük olanın değeri olarak belirlenmiş, en az 1 eksen matris sınırına ulaşıyor ise zon değeri 0 olarak belirlenmiştir.

Hücre Ekseni Sınır Kontrolü algoritması

1. Matrisin seçili hücresinin eksenleri doğrultusunda ilerle
2. 4 taraftan karşılaşılan sınır değerlerinin en büyüğünü ve sınır pozisyonlarını al
3. Eğer herhangi bir taraf matris sınırına ulaşıyosa sınır değerini 0 olarak belirle
4. Seçili hücreden tutulan sınır pozisyonlarına kadar olan eksen üzerindeki hücrelerin değerini sınır değeriyle değiştir ve kontrol edilen her hücrenin kontroledildi değerini true yap
5. Matris üzerinde kontrol edilmemiş hücre kalmayana kadar devam et

Şekil 3.7. Hücre Ekseni Sınır Kontrolü algoritması

3.2. Birincil Yol Ağını ve Adaları Oluşturma

Bir şehri oluşturan en önemli unsurlardan biri de yol ağlarıdır. Modern şehirlerde yol ağları genellikle ızgara modeline uygun olsa da, radyal yol ağları ve ya düzensiz yol ağları da görülebilir.

Algoritmamızda bu modellerden ızgara modeli kullanılmıştır. Şehir yatay ve dikey yollarla bölünmüş. Bu bölünmeden çıkan parçalar adaları oluşturmuştur.

Yolların arasında kalan parçaların boyutları hesaplanırken zon matrisi 6x6'lık parçalara ayrılıp adaların sağ ya da alt sınırına göre bir çarpım değeri belirlenmiştir. Enine bölme sırasında oluşacak adanın alt sınırı en alt ve en üstteki parçalar arasında kalıyorsa çarpım değeri 3, alttan ya da üstten 2. Parçalar arasında kalıyorsa çarpım değeri 2, alttan ve üstten 3. Parçalar arasında kalıyorsa da çarpım değeri 1 olarak belirlenmiştir. Aynı şekilde boyuna bölme sırasında oluşacak adanın sağ sınırı en sağ ve en soldaki parçalar arasında kalıyorsa çarpım değeri 3, sağdan ya da soldan 2. Parçalar arasında kalıyorsa çarpım değeri 2, sağdan ve soldan 3. Parçalar arasında kalıyorsa da çarpım değeri 1 olarak belirlenmiştir. Bu çarpım değeri 40 ile 70 arasında değişen bir tam sayı değeri ile çarpılmış ve böylece yolların yatay ve dikey konumları elde edilmiştir. Bu işlem sayesinde yollar merkeze doğru sıklaşırken merkezden uzaklaştıkça seyrekleşmişlerdir.

Yol çizgileri genişletilmeden yol çizgilerinin arasında kalan alanlar adalar olarak bir listede tutulmuştur. Her bir ada; adanın genişliğini, uzunluğunu, merkez pozisyonunu ve zon matrisi üzerinden AABB ile alınmış matris parçasını tutar.

Adalar dikdörtgen şeklinde olduğundan AABB'leri de kendilerine eşit olmuştur. Ancak Voronoi diyagramı gibi metodlarla oluşturulan yol ağları sonucunda çokgen şeklinde adalar da ortaya çıkabilecektir. Bu durumda adaların sadece köşelerinin konumlarını tutması yeterli olacaktır. Böyle bir durumda ada diktörtgen olmadığı için AABB'si x ekseninde birbirine en uzak köşeler ve z ekseninde birbirine en uzak köşeler alınarak hesaplanacaktır.

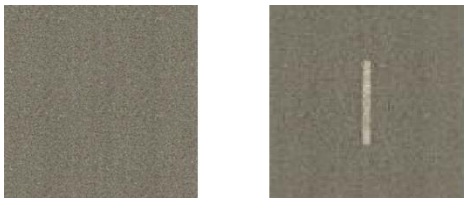
Yolların genişlikleri de gene çarpım değerine bağlı olarak hesaplanmıştır. 1 ile 2 birim arasında rastgele genişlik değeri seçilip konumlarına göre çarpım değeri ile çarpılarak yolun genişliği belirlenmiştir. Yollar her 1 birime 2 şerit sığacak şekilde planlanmıştır. Bu sayede şehrin merkezinde az şeritli yollar bulunurken şehri çevreleyen yolların şerit sayısı artmıştır.

Bir yol enine doğru genişlediğinde yolun sağında bulunan adalar genişleme kadar sağ tarafa, boyuna doğru genişlediğinde de yolun üst tarafında bulunan adalar genişleme kadar üst tarafa doğru itilir.

Şehri enine ve boyuna bölen şeritlerin bilgileri iki farklı listede saklanmıştır. Bu şeritler birbirleri ile 90°'lik açılarla kesişmiş, kesişim noktalarında kalan alanların uzunlukları, genişlikleri ve merkez pozisyonları kavşakları oluşturmak üzere bir listede tutulmuş, bölünen parçaların uzunlukları, genişlikleri ve merkez pozisyonları ise yolları oluşturmak üzere başka bir listede tutulmuştur.

Tutulan yol ve kavşak bilgileri ile diktörtgen model kafesleri oluşturulmuştur. Kavşak modelleri için Şekil 3.8'deki kaplama, model kafesine atanmış ve model kafesinin eni ve boyunun 2 katı kadar tekrarlanmıştır. Uzunlamasına giden yollarda da Şekil 3.8.'deki yol kaplaması, model kafesine atanmış ve sonrasında yarım birim sağ çıkıntı yapacak şekilde model kafesinin eni ve boyunun 2 katı kadar tekrarlanmıştır. Enlemesine giden yollarda çıkıntı üstten yapılmış ve Şekil 3.8.'deki yol kaplamasının 90° döndürülmüş hali kullanılmıştır.

Her kavşağın 4 köşesine trafik akış yönüne bakan trafik lambaları da yarleştilmiştir.



Şekil 3.8. Kavşak ve yol kaplamaları

3.3. Adaları Zon Bazlı Bölümleme Algoritması Kullanarak Bloklara Bölme

Adalar içerisinde birden fazla zonu bulundurabileceğinden binaların yerleştirilecekleri parsellere bölünmeleri durumunda parselin de birden fazla zonda bulunması durumu ortaya çıkacaktır. Bu sorunu ortadan kaldırmak ve her seferinde parsellerin zon değerlerini hesaplamak için bulundukları zon matrisi parçasının hücrelerinde gezinmeyi önlemek için Zon Bazlı Bölümleme Algoritması (Şekil 3.10) geliştirilmiştir.

Zon Bazlı Bölümleme Algoritması bir adanın tuttuğu zon matrisi parçası içerisindeki seçilen zon değerine sahip hücre grubunu AABB kullanarak çıkarmayı sağlamaktadır. Çalışmamızda bu değer en küçük zon değeri olarak belirlenmiştir. Hücre grubunun matris

içerisinde x ekseninde birbirine en uzak noktaları bulunup zon matrisi parçasını bu noktaların pozisyonları boyunca boyuna, z ekseninde birbirine en uzak noktaları bulunup zon matrisi parçasını bu noktaların pozisyonları boyunca enine keserek bölümleme sağlanır.

En küçük zon değerine sahip hücre grubunun AABB'sinin en az bir kenarı zon matrisi parçasının bir kenarına teğet geçeceğinden bu bölümleme sonucu en fazla 3x2 ya da 2x3 şeklinde 6 parça çıkacaktır. En küçük zon değerine sahip hücre grubunun AABB'sinin birbirine dik iki kenarının zon matrisinin kenarlarına teğet olması durumunda 2x2 lik 4 adet parça oluşacaktır.

Oluşan parçaların enlerinin uzunlukları verilen eşik değerinden küçük ise parçalar yataydaki komşuları ile, boylarının uzunlukları verilen eşik değerinden küçük ise parçalar dikeydeki komşuları ile birleşecektir.

AABB ile ayrılan her parçaya ister aynı değerlere göre ister seçilen değerin zıttına göre ayrılacak şekilde istenilen tekrar sayısı kadar Zon Bazlı Bölümleme Algoritması uygulanabilir. Ancak bu durum zonların data kayıplarını azaltsa da, çalışma zamanına negatif bir etki olarak yansıyacaktır. Bu yüzden zonlardaki data kaybı optimal kabul edilerek Zon Bazlı Bölümleme Algoritması adalara sadece bir kez uygulanmıştır.

İlk parselin bölümlenmesi ile ortaya çıkan parçalara "bloklar" denilmiştir. Bloklar bünyelerinde

- Blok eni
- Blok boyu
- Blok zon değeri
- Blok merkez pozisyonu
- Kamu alanı olup olmadığı bilgisi

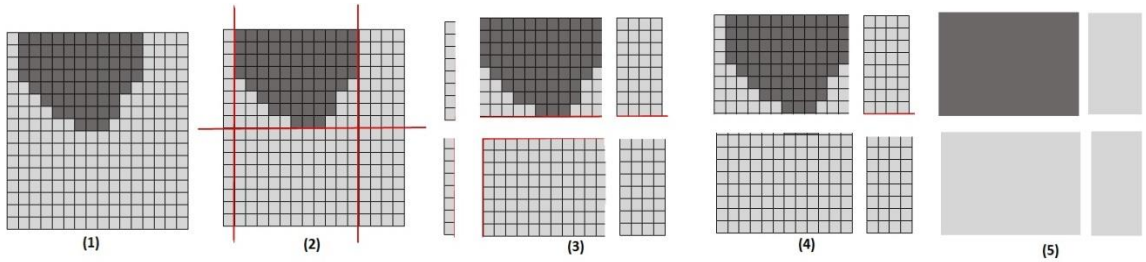
bilgilerini tutar.

Adaların çokgen şeklinde olması durumunda oluşturulan bloklar da çokgen şeklinde olacaktır. Bu durumda ise blok köşelerinin pozisyonları, kamu alanı olup olmadığı bilgisi ile blok zon değerini tutmalıdır.

Bölümleme sonucunda oluşan blokların aralarından 2 şeritli yollar geçirilmiştir. Bu işlem sonucunda oluşan blokların boyutları bir miktar küçültülüp, merkez pozisyonları değiştirilmiştir.

Zon Bazlı Bölümleme Algoritmasıyla oluşan bölümleme sonucunda adaların özdeş bilgileri bloklara aktarılmıştır.

Zon Bazlı Bölümleme Algoritmasının çalışma mantığı Şekil 3.9'de resmedilmiştir. Bölümleme sonucu oluşan parsellerin görüntüsü Şekil 3.11'de gösterildiği gibi olacaktır.

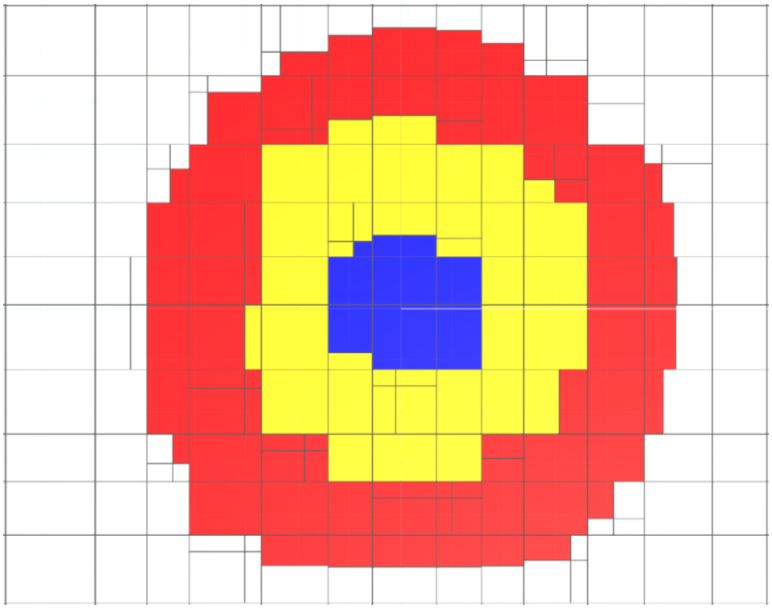


Şekil 3.9. Zon bazlı bölümleme algoritmasının çalışması

Zon Bazlı Bölümleme Algoritması

1. Adanın matrisindeki en küçük zon değerini bul
2. Bu zon değerine ait hücre grubunun x ve z eksenlerindeki birbirlerine en uzak konumlarının indeks değerlerini al (AABB'sinin kenarlarının geldiği pozisyonlar)
3. Bulunan indeks değerleri yardımı ile adayı bölümle
4. Ortaya çıkan blokların boyutları eşik değeri altında ise komşu blok ile birleştir.
5. Ortaya çıkan blok bilgilerini blok objesine aktar.

Şekil 3.10. Zon Bazlı Bölümleme Algoritması



Şekil 3.11. Şehrin bloklara bölünmüş hali. Her renk farkı bir zonu ifade etmektedir.

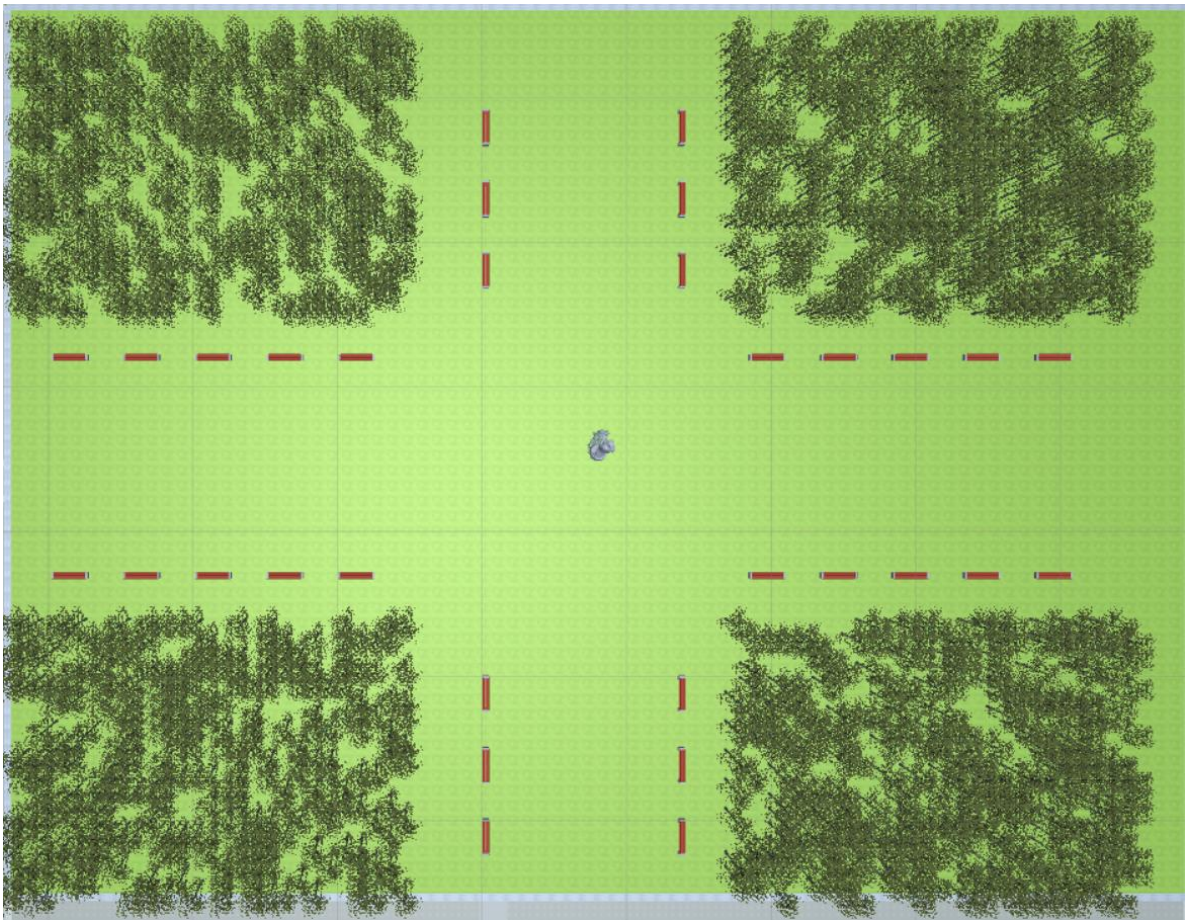
3.4. Kamu Alanlarının Hesaplanması ve Yerleştirilmesi

Kamu alanları şehirlerde önemli yer kaplamaktadır. Yöntemsel şekilde üretilen şehirlerde kamu alanları ya bina parselleri ile ayrılan alanlara konmuş ya da hiç kullanılmamıştır.

Kamu alanları karakollar, okul kampüsleri, hastaneler, parklar gibi alanlara denmektedir.

Kamusal alanlar, bloklara konulmuş böylece bütün kenarlarının yollarla bağlantısı sağlanmıştır.

Şehir merkezinde bulunan bloğa merkez parkı konulmuştur. Merkez parkı tam merkezinde devasa bir heykelin bulunduğu, bu heykele doğru uzanan yolların olduğu, yolların ayırdığı köşelerde ağaç kümeleri bulunan ve bu ağaç kümelerinin içeriye bakan kısmında bankların bulunduğu, çim kaplamasına sahip bir alan üzerinde bulunan bir yapıdır. (Şekil 3.12)



Şekil 3.12. Merkez parkının üstten görünüşü

Şehirdeki diğer parkların yapısı da merkez parkına benzemektedir. Tek farkları merkezlerinde heykel yerine oyun alanı bulundurmalarıdır.

Merkez parkı dışındaki diğer tüm kamu alanlarının koyulacağı bloklar bu kamu alanlarının AABB'lerinin eni ve uzunluğunun sığabileceği blokların herhangi birisinin rastgele seçilmesi ile belirlenmiştir.

Seçili bloklara konulan alanlar eğer yer var ise blok üzerinde kendi ön cephelerine bakan tarafa doğru çekilip arka taraflarına sahalar ve koşu alanları gibi etkinlik alanları eklenmiştir.

Tez çalışmasında kullanılan kamu alanı modelleri aşağıdaki gibidir:

- 1 adet karakol,
- 2 adet okul,
- 1 hastane
- 1 Emniyet müdürlüğü
- Merkez parkı için 1 heykel
- Diğer parklar için 1 oyun alanı
- 3 saha

Bu modeller şehre zonlara göre yerleştirilmiştir. Buna göre zonlar en içten en dışa 1'den 3'e doğru sıralanırsa:

1. **Zonda bulunan kamusal alanlar:** Merkez parkı, Emniyet genel müdürlüğü ve 1 adet okul kampüsü
2. **Zonda bulunan kamusal alanlar:** 1 adet okul kampüsü, 1 adet hastane, 1 adet karakol ve 1 park
3. **Zonda bulunan kamulal alanlar:** 1 adet karakol, 1 adet okul kampüsü ve 2 adet park

olarak belirlenmiştir.

Kullanıcı şu anda dışarıdan kamu alanı ekleyemese de program yeni kamu alanlarının parametre olarak ekleneceği şekilde bir liste tutmaktadır. Bu listenin her bir elemanında alanın adı, kullanılacak model, koyulacağı zon ve kamu alanlarının kaplayacağı alanın uzunluğu ve genişliği yer almaktadır.

3.5. Blokları OBB-Rastgele Bölümleme Algoritması ile Parsellerine Bölme

Yöntemsel şehir üretiminde parselleme denince ilk akla gelen binaların konulacağı alanları oluşturan metodlardır. Bu alanların alabileceği binaların boyutları şehrin görüntüsünü değiştirecektir. İstenen şey bu alanların yollarla bir şekilde bağlantısının olması ve binaların sığabileceği ya da yöntemsel üretilen binaların garip durmayacağı yüzey alanlarına sahip olmalarıdır.

Bu tezin çalışmasında bir parselleme metodu olan OBB bölümleme metodu geliştirilerek OBB-Rastgele bölümleme metodu oluşturulmuştur.

OBB bölümleme bir BSP metodudur. Yani alanları OBB içine alıp sürekli iki parçaya böler.

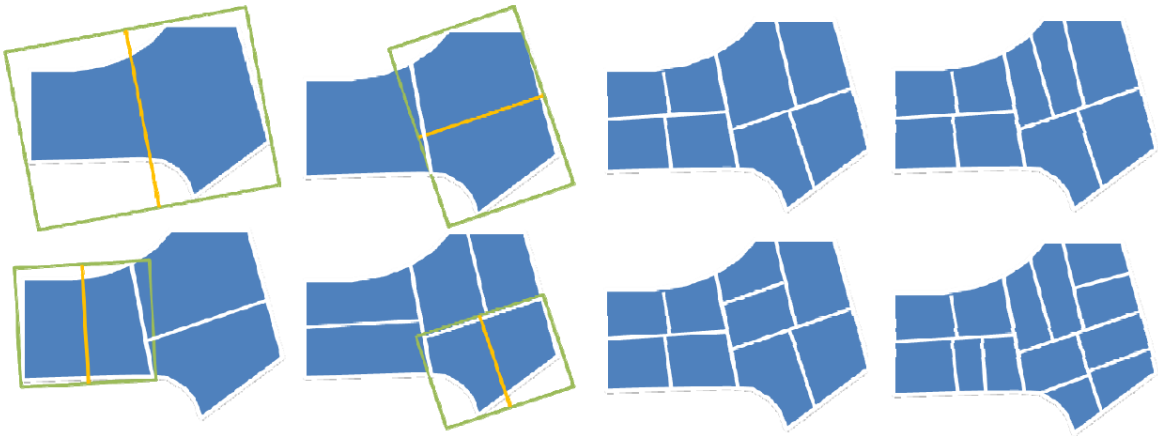
OBB bölümleme metodunun algoritması Şekil 3.13'de gösterildiği gibidir.

OBB bölümleme metodunun algoritması

1. Alanı OBB içine al
2. Alanı OBB'nin en kısa kenarına paralel bir çizgi ile tam ortadan böl
3. Oluşan yeni alanlar için bu işlemi istenilen koşul sağlanana kadar tekrarla

Şekil 3.13. OBB bölümleme metodunun algoritması

Bu algoritma sonucu Şekil 3.14'de gösterilen biçimde parselleme yapılır.



Şekil 3.14. Bir alanı OBB Bölümleme metodu kullanarak parçalarına ayırma[27]

OBB bölümleme metodu deterministik bir metoddur. Yani bir alan ne kadar bölünürse bölünsün tekrar sayısı aynı ise hep aynı şekil ve büyüklükte parsellere ayrılacaktır.

OBB-Rastgele bölümleme algoritması OBB bölümleme metodunun deterministik yapısını değiştirip her seferinde tamamen rastlantısal boyutlarda parseller oluşmasını sağlamak için geliştirilmiştir.

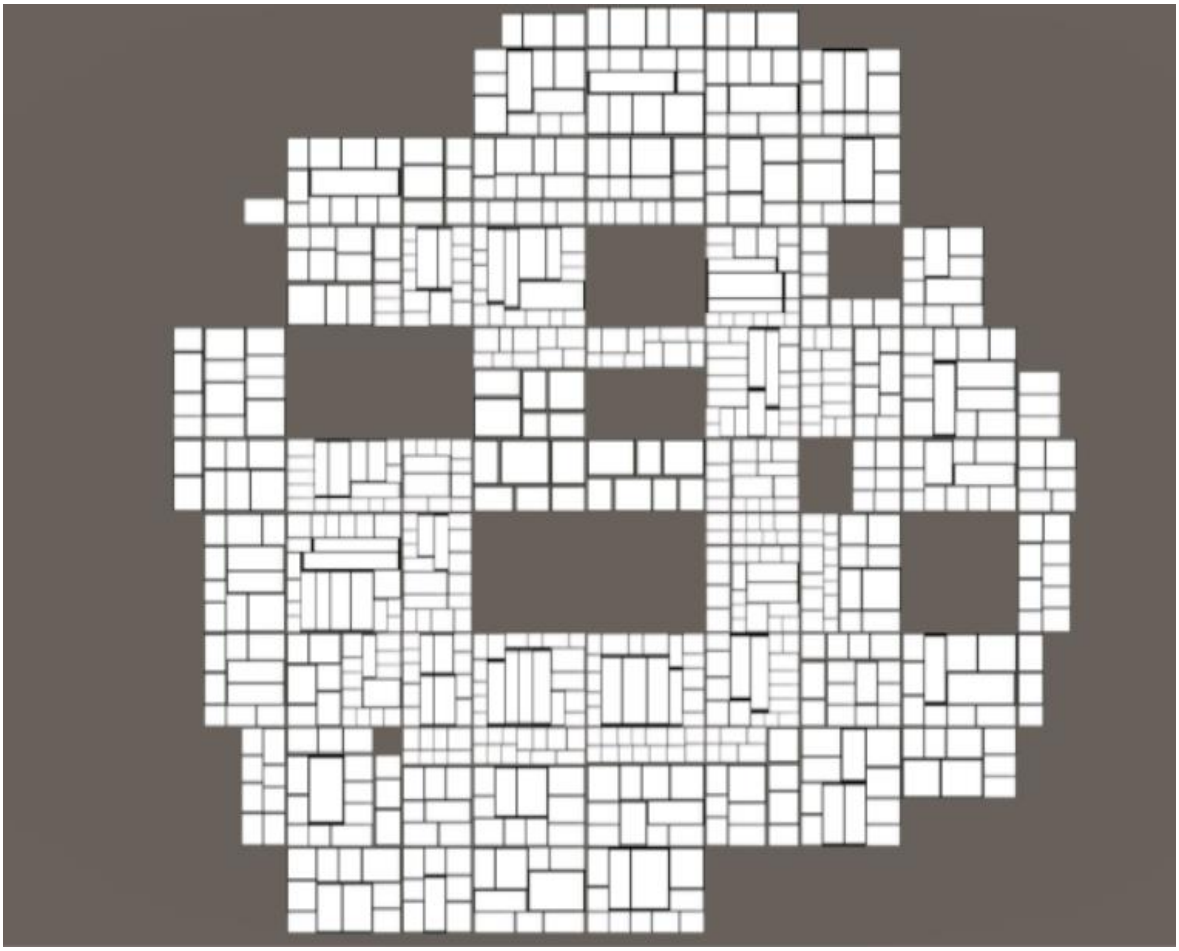
Tez çalışmasında oluşan alan parselleri hali hazırda diktörtgen şeklinde olduğu için bu parsellerin OBB'lerinin hesaplanmasına gerek kalmamıştır. OBB olarak direk kendileri kullanılmıştır.

OBB-Rastgele bölümleme algoritmasının koşulları aşağıdaki gibidir:

- Bölümlemeler kısa kenara paralel bir çizgi ile değil oluşan tüm blok parçaları için sırasıyla yatay ve dikey olarak yapılacaktır.
- Bölümleme bloğu direk yarıya değil rastgele verilen oranlarda iki parçaya ayıracaktır.
- Bölünen parçanın uzunluğu minimum değerden küçük olamaz.
- Bölünen blok parçasının en az bir kenarının yol ile bağlantısı olması şarttır.

- Eğer bölümleme çizgisinin dik kestiği OBB kenarı verilen maksimum değerden büyükse bölümleme kesin gerçekleşir, minimum ve maksimum değer arasında ise bölümleme %50 olasılıkla gerçekleşir, minimum değerine eşit ise bölümleme gerçekleşmez ve eğer gerçekleşmedi ise bi daha o kenara dik doğrultuda bölünme gerçekleşmez.

OBB-Rastgele Bölümleme algoritmasının her adımında oluşan bütün parçalara yatayda ve dikeyde bölünüp bölünmeyeceğini kontrol eden bir deger eklenmiştir. OBB-Rastgele bölümleme algoritmasının sözde kodu Şekil 3.16'da gösterildiği gibidir. Bu bölümleme sonucu şehrin blokları Şekil 3.15'de gösterildiği gibi parsellerine bölünür.



Şekil 3.15. Şehirdeki parsellerin gösterimi

OBB-Rastgele bölümleme metodunun algoritması

1. Alanı OBB içine al
2. x i değerinin 1 olması durumunda enine 0 olması durumunda boyuna bölümleme gerçekleştir.
3. w←Bölünecek elemanın OBB'sinin bölüm çizgisine dik olan kenarının uzunluğu
4. x←0
5. Min ve max değerlerini belirle
6. Alanı ve OBB'sini listeye aktar
7. Döngü (listedeki elemanlar bitene kadar)
8. BolmeKontrolu ← true
9. **IF**(sıradaki eleman enine bölünemiyor ve x = 1 ise)
10. BolmeKontrolu ← false;
11. **Koşul Sonu**
12. **IF**(sıradaki eleman boyuna bölünemiyor ve x = 0 ise)
13. BolmeKontrolu ← false;
14. **Koşul Sonu**
15. **IF**(x=1 ve eleman x sadece eksenine paralel 1 yola sahipse)
16. BolmeKontrolu ← false;
17. elemanın enine ve boyuna bölünme özelliklerini false yap
18. **Koşul Sonu**
19. **IF**(x=0 ve eleman y sadece eksenine paralel 1 yola sahipse)
20. BolmeKontrolu ← false;
21. elemanın enine ve boyuna bölünme özelliklerini false yap
22. **Koşul Sonu**
23. **IF**(w <= 2*min)
24. BolmeKontrolu ← false;
25. **Koşul Sonu**
26. **IF**(min< w ve w< max ve BölmeKontrolu = true ise)
27. rd← RastgeleDeger(0,1);
28. **IF**(rd<=0.5f)
29. BolmeKontrolü ← false
30. **Koşul Sonu**
31. **Koşul Sonu**
32. **IF**(BolmeKontrolu= false ve x = 1)
33. elemanın enine bölünme kontrol özelliğini false yap;
34. **Koşul Sonu**
35. **IF**(BolmeKontrolu= false ve x = 0)
36. elemanın boyuna bölünme kontrol özelliğini false yap;
37. **Koşul Sonu**
38. **IF**(BolmeKontrolu = true)
39. rmax← max
40. **IF**(rmax> w)
41. rmax← w;
42. **Koşul Sonu**
43. s1← RastgeleDeger(min,rmax);
44. s2←w-s1;
45. **IF**(s2<min)
46. s2←min
47. s1←w-s2
48. **Koşul Sonu**
49. Parçayı s1 ve s2 değerlerine gör böl ve her bir parçayı listeye ekle
50. **Koşul Sonu**
51. Eleman hem enine hem de boyuna bölünemiyor ise binaParseliListesine ekle
52. Elemanı listeden çıkar.
53. Döngü Sonu

Şekil 3.16. OBB-Rastgele bölümleme metodunun algoritması

Bir parsel bünyesinde

- Merkez pozisyonunu
- Genişlik ve Uzunluk değerlerini
- Yol geçen kenarlarını temsilen bir 4 elemanlı byte dizisini
- Ve zon değerini

taşır.

Çalışmamızda oluşan bloklar diktörtgen şeklinde olsa da birincil yolları oluşturmak için kullanılan fonksiyonun değilmesi durumunda adalar çokgen şekillerinde oluşabilir. Bu durum domino etkisi yaratarak blokların da şeklini değiştirecektir. Bu durumda OBB-Rastgele bölümlenmesini gerçekleştirmek üzere bir bloğun OBB'si olarak bloğu içine alan minimum dikdörtgen hesaplanmıştır.

Bu hesaplama iki adımda gerçekleşmiştir:

1. Çokgeni içine alan dış bükey kabuğu bulma
2. Bu dış bükey kabuğu kullanarak minimum diktörtgeni bulma

Çokgenin dış bükey kabuğunu bulmak için Şekil 3.17'de sözde kodu gösterilen Graham Scan algoritması kullanılmıştır.

Graham Scan Algoritması

1. En sol alt konumda bulunan noktayı (köşeyi) kabuğun başlangıç noktası say
2. Kalan noktaları başlangıç noktası ve x ekseni arasındaki açıları küçükten büyüğe artacak şekilde sırala
3. Başlangıç noktasını ve sıralanmış noktalardaki ilk iki noktayı bir yığına ekle
4. Yığının en üstündeki elemanı alın ve sıralanmış noktalardaki bir sonraki noktaya gidip yığına ekle
5. Yığının en üstündeki elemanı alıp sıralanmış noktalardaki bir sonraki noktaya git
6. Eğer bu noktaya giderken saat yönünün tersine değil saat yönünde hareket edildi ise yığına eklenen son elemanı çıkar
7. Eger yığının boyutu 3'ten büyükse yığının üstündeki her nokta için 7. Adımı tekrar et
8. Saat yönünün tersine dönüş sağlandığında noktayı yığına ekle
9. Sıralı noktalar bitene kadar işlemlere 6. Adımdan itibaren devam et
10. Bitir

Şekil 3.17. Graham Scan Algoritmasının Sözde Kodu

Şekil 3.17'de gösterilen Graham Scan algoritmasında oluşturulan yığın üzerindeki elemanlar üzerinde sırayla dolaşılıp yığının ilk elemanı son elemanına bağlanırsa dış bükey kabuğu elde edilir.

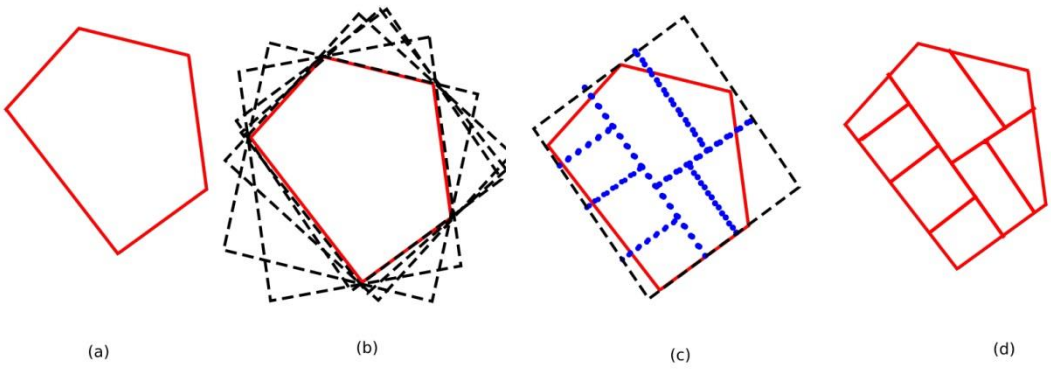
Dış bükey kabuğu bulduktan sonra ikinci aşamada bloğuiçine alan en küçük dikdörtgen hesaplanmıştır. Bu dikdörtgenin en az bir kenarı dış bükey kabuğun bir kenarına teğet olacağından, bu şekilde dış bükey kabuğu saran bütün dikdörtgenler Şekil 3.18 de gösterilen sözde kod yardımı ile hesaplanıp aralarından en küçük alana sahip olan alınmıştır.

Dış Bükey Kabuğu saran dikdörtgen hesaplama

1. Teğet geçen kenar doğrultusunda sonsuz bir çizgi çek
2. Bu çizgi üzerine noktaların dik iz düşümlerini al
3. İz düşümleri birbirine en uzak iki noktayı belirle ve iz düşüm noktalarını bir listeye ekle
4. İz düşümlerinden noktalara doğru uzanan sonsuz birer çizgi çiz
5. Teğet çizgisine dik olarak en uzaktaki noktayı bul
6. bu noktanın iz düşüm çizgilerine dik iz düşüm değerlerini al ve listeye ekle
7. Listeyi geri döndür.

Şekil 3.18. Dış Bükey Kabuğu saran dikdörtgen hesaplama

Bu şekilde en küçük dikdörtgen hesaplandıktan sonra dikdörtgen içerisindeki blok ile birlikte herhangi bir kenarının x ekseni ile yaptığı açı kadar döndürülüp OBB Rastgele bölümleme algoritması uygulandıktan sonra tekrar aynı açı kadar geri döndürülüp bölümlenme tamamlanmıştır. (Şekil 3.19)



Şekil 3.19. Bir çokgen üzerinde OBB-Rastgele bölümleme fonksiyonunun kullanımı (a) Blok belirlenir ve dışbükey kabuğu bulunur(burada blok zaten dışbükey olduğundan hesaplama yapılmamıştır) (b) Dış bükey kabuğu saran diktörtgenlerden en küçüğü seçilir (c) OBB-Rastgele Bölümleme algoritması gerçekleştirilir. (d) Bölümleme sonucu bloğa yansıtılaral parseller oluşturulur.

3.6. Şehir Merkezindeki İkincil Yol Ağlarının Çizge Tabanlı Rastgele Yürüyüş Algoritması ile Hesaplanması

Rastgele yürüyüş algoritması yöntemsel oluşturmada kullanılan bir tekniktir. En ünlü rastgele yürüyüş algoritmalarından birisi Sarhoş Yürüyüşü algoritması oyunlar için yöntemsel zindan sistemi oluşturmakta kullanılmaktadır[21]. Sarhoş Yürüyüşü algoritmasının sözde kodu Şekil 3.20'deki gibidir.

Sarhoş Yürüyüşü Algoritması

1. Dolu bir matris oluşturun
2. Rastgele bir nokta seçin ve boş olarak işaretleyin.
3. Eksen komşularından birini seçin ve onu da boş olarak işaretleyin
4. 4. Adımı boş bir alan gelene kadar tekrarlayın
5. 3. Adımdan itibaren işlemleri istenilen iterasyon kadar tekrarlayın

Şekil 3.20. Sarhoş Yürüyüşü Algoritması

Rastgele yürüyüş algoritması ikincil yol ağını oluşturmak için kullanılan metodun temelini oluşturmuştur. Rastgele yürüyüş algoritmasına benzer biçimde parseller arasında kalan kısımlarda rastgele dolaşarak ikincil yol ağlarını oluşturan bir algoritma geliştirilmiştir.

Bu algoritma için bir bloğun bölünmesi ile oluşan parsellerin arasında kalan kısımlarla bir çizge oluşturulup bu çizge üzerinde gezinildiği için bu algoritmaya Çizge Tabanlı Rastgele Yürüyüş Algoritması denmiştir.

Çizgeler düğümlerden oluşan yapılardır. Her bir düğüm üzerinde:

1. Düğümün bulunduğu pozisyon
2. Komşu düğümlerin çizge üzerindeki indekslerinin listesi
3. Köşeleri oldukları parsellerin liste indekslerini saklayan 4 birimlik dizi {sağ üst, sol üst, sol alt, sağ alt} (eğer belirtilen köşe doğrultusunda bir bina parseli yoksa dizinin o elemanına -1 değeri verilir)
4. Yığılma değeri (Birden fazla komşusu olan düğümlerde bu değer 1, tek komşuya sahip düğümlerde bu değer 0'dır. Her seferinde komşu sayılarının kontrolünü önlemek amacı ile atanmıştır.)

bilgileri bulunur.

Çizgeler doldurulurken önce düğüm pozisyonları ve köşeleri oldukları parseli liste indeks dizisi hesaplanır. Bunun için Şekil 3.21'deki algoritma kullanılmıştır.

Çizge Dolum Algoritması 1. Adım

1. P←Bina parselinin köşe noktasının pozisyonunu
2. **IF**(Aynı pozisyona dair bir bilgi çizge listesinde yoksa)
3. P'yi çizge listesinin düğüm pozisyonu bilgisine ekle
4. **Koşul Sonu**
5. Bina parselinin liste indeksini konumuna göre çizgede tutulan dizide gerekli yere yerleştir.

Şekil 3.21. Çizge Dolum Algoritması 1. Adım

İkinci aşamada çizge üzerindeki düğümlerin komşulukları hesaplanmıştır. Bunun için çizge çizgilerinin eksenlere paralel olmaları şarttır. Çizge üzerindeki düğümlerin komşulukları Şekil 3.22'de gösterilen algoritma ile hesaplanır.

Çizge Dolum Algoritması 2. Adım

1. **Döngü**(Çizge üzerindeki her bir düğüm için)
2. Pozisyon değerinin z'si aynı olan tüm değerlerde zıt taraflarda birbirine en yakın iki düğümün indeksi alınır
3. Pozisyon değerinin x'i aynı olan tüm değerlerde zıt taraflarda birbirine en yakın iki düğümün indeksi alınır
4. **Döngü Sonu**

Şekil 3.22. Çizge Dolum Algoritması 2. Adım

Çizgenin dış kısmını oluşturan bağlantılar üzerinden hali hazırda yollar geçtiğinden bu çizgiler üzerindeki komşuluk bağlantıları silinmelidir. Bunun için şekil de sözde kodu verilen algoritma kullanılmıştır.

Çizge Dolum Algoritması 3. Adım

1. En alt-sol köşedeki düğümü bul
2. Komşularına giden vektörlerle pozitif x'e giden vektör arasında kalan açıları hesapla
3. Açısı en küçük olan düğüm ile komşuluk bağlantılarını sil
4. Yeni düğüm değerini bağlantısı silinen düğüm olarak belirle
5. Başlangıç noktasına gelene kadar 3. Adımdan beri tekrar et

Şekil 3.23. Çizge Dolum Algoritması 3. Adım

Tüm işlemler halledildikten sonra her düğümün komşuluk sayılarına göre yığılma değerleri güncellenir.

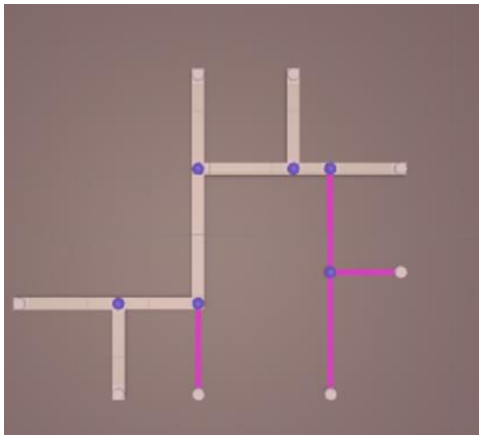
İkincil yol ağını oluşturmak için kullanılan Çizge Tabanlı Rastgele Yürüyüş Algoritması Şekil 3.24'deki gibidir. Bu algoritmayı geliştirmeye L-Sistemini kullanan Kaplumbağa grafiğinin düşünce yapısı da yardımcı olmuştur.

Çizge Tabanlı Rastgele Yürüyüş Algoritması

1. secim←Yığılma değeri 0 olan düğümlerden rastgeli birisinin indexi
2. secimin indexini yol indexi listesine ekle
3. komşuya git
4. komşunun indexini yol indeksi listesine ekle
5. komşunun indexini bir yığına ekle
6. yol indeksi listesinde bulunmayan komşulardan birisine git
7. gidilen komşunun yığın değeri 0 oluncaya ya da gidilecek komşusu kalmayıncaya kadar 5. Adımdan devam et
8. komşunun indeksini yol ağı listesine
9. Yığın dolu ise yığının en üstünde tutulan değeri çıkartıp yol indeksi listesine ekle ve tut
10. Yığın dolu ise ve istenilen iterasyon sayısından daha az döngü sağlanmışsa 7. adıma dön

Şekil 3.24. Çizge Tabanlı Rastgele Yürüyüş Algoritması

İkincil yol ağı görselleştirilirken Şekil 3.24'te oluşturulan yol indeksi listesi kullanılır. Listedeki her elemanın bulundu yerde bir adet kavşak modeli konulmuştur. Art arda olan iki elemanın aralarındaki uzaklıktan yol genişliği çıkarılınca oluşan değer yolların uzunluğunu vermiş. Bu bilgi ile de yollar oluşturulmuştur. (Şekil 3.25)



Şekil 3.25. Çizge tabanlı rastgele yürüyüş algoritması sonucu oluşmuş bir yol ağı. Yol ağı çizge üzerindeki düğümler arasında rastgele oluşturulmuştur.

Yollar genişletilirken bu yollarla bağlantılı parsellerin boyutları küçülecek, merkezlerinin yeri değişecektir. Bu işlem Şekil 3.26'de gösterilen algoritma yardımı ile yapılmıştır.

Bina alan boyutu güncelleme algoritması

1. Yol indeksi listesinde tutulan değerden bir sonraki değere uzanan vektör doğrultusundaki tutulan değere komşu bina parsellerini bu vektörden dik çıkan vektörler doğrultusunda yol genişliğinin yarısı kadar böl ve aynı doğrultuda yol genişliğinin ¼'ü kadar merkez noktalarını kaydır.
2. Sıradaki noktadan yolun herhangi bir indexte aynı eğim doğrultusunda ilerleyip ilerlemediğine bak
3. İlerlemiyorsa aynı eğim doğrultusu çaprazındaki bina parsellerini kontrol et
4. Eğer bir bahsi geçen bina parsellerinden birisinin köşesi değilse diğer komşuyu o bölüme kadar yol genişliğinde büyüt ve yol genişliğinin yarısı kadar kaydır
5. Bir sonra gelecek değer bir önceki değerin aynısı ise bir değer atla
6. Yol indeksi listesindeki değerler bitene kadar devam et

Şekil 3.26. Bina alan boyutu güncelleme

3.7. Yöntemsel Bina ve Parsel Modellerinin Oluşturulması

Binalar şehir üretiminde olmazsa olmaz yapılardır. Bu çalışmada hem hazır modeller kullanılırken hem de yöntemsel üretim teknikleri kullanarak binalar üretilmiştir.

Yöntemsel olarak üretmek üzere 4 tip bina konsepti oluşturulmuştur(Şekil 3.27):

1. Tipik dikdörtgen
2. Katmanlı bir biçimde küçülen
3. Merdiven modeli
4. V şekli

Bina üretimi aynı Citygen: An interactive system for procedural city generation isimli çalışmada olduğu gibi oluşturulan modellere kaplama atayarak oluşturulur. Kaplamalar her bir yüzeyin boyu ve eni oranında tekrarlanmıştır[16].

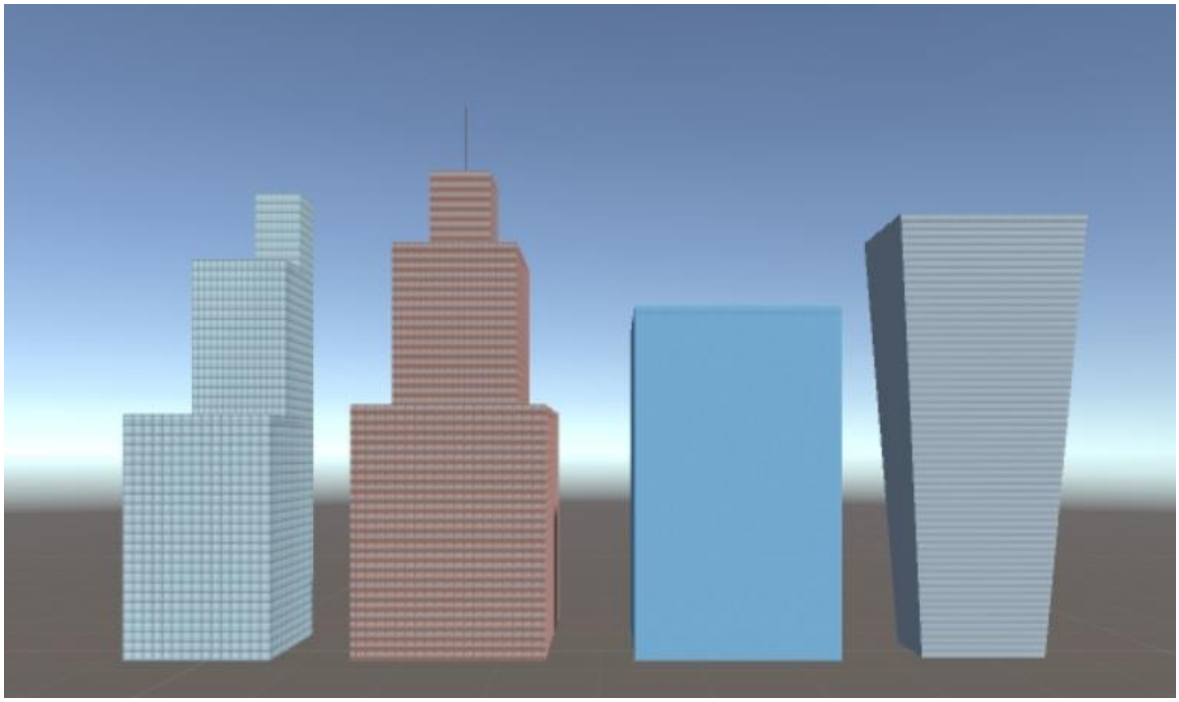
Üretilmesi planlanan bina modelleri binanın boyutuna göre seçilmiştir. Binanın boyutu 40 birim altında olacaksa 1. Tipte binalar, 40 birimden yüksek olacaksa da

- %50 olasılıkla 3. Tipteki
- %30 olasılıkla 2. Tipteki
- %20 olasılıkla 4. Tipteki

binalardan seçilecektir..

Bina yüksekliklerini belirlemek için perlin gürültüsü fonksiyonu kullanılmıştır. Gürültü değeri bina parselinin merkez pozisyonunun x ve z değerlerinin grid boyutuna oranı ile

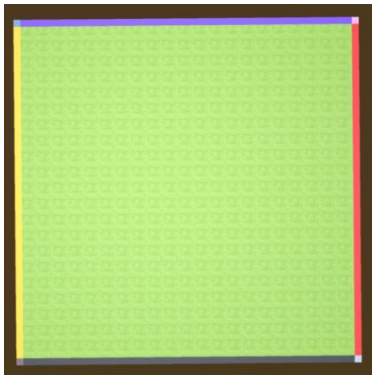
hesaplanmıştır. Bulunan değer 0 ile 1 arasında olduğundan bina boyu minimum 30 birim maksimum 100 birim olacak şekilde boy hesaplamasında kullanılırlar.



Şekil 3.27. Yöntemsel olarak oluşturulmuş bina tipleri. Sağdan sola: Merdiven, Katmanlı bir biçimde küçülen, Normal ve V şaklinde

Binaların, kamusal alanların ve neredeyse bütün herşeyin üstünde duracağı parseller veya bloklar alt tarafı olmayan dikdörtgen kutular şeklinde yöntemsel olarak parsel ya da blok bilgilerini kullanarak üretilmiştir.

Her bir parsel modelinin üst yüzü 9 parçaya ayrılmış. Dışta kalan alanlar kaldırımları oluşturacak büyüklükte küçülmüşlerdir.



Şekil 3.28. Parsel modelinin üst yüzey kaplamasının görünümü

Şekil 3.28'da görüldüğü gibi her bir kaldırıma, köşede, kenarda nerede olursa olsun farklı materyaller atanabilmektedir. Bu sayede istenildiği zaman bu parsel modelleri ile blokların

ya da adaların boyutunda yapılar döşeme tekniği ile doldurulup tek parçaymış görüntüsü verilebilir.

3.8. Hazır Binaların Parsellere Yerleştirilmesi

Binaların hepsini yöntemsel olarak oluşturmak iyi bir yöntem gibi gözükse de bazen kullanıcılar bölünen parsellere kendi binalarını yerleştirmek isteyebilirler.

Bu çalışmada merkez zonu haricindeki zonlarda bulunan parsellere binalar yerleştirilerek şehir oluşturulmuştur. En dıştaki zonda müstakil tipte binalar bulunurken, ortada bulunan zonda apartman daireleri bulunmaktadır.

Dışarıdan alınan binaların hepsinin boyutlandırma standartı farklı olduğu için 2 katlı binaların 10 katlı bina boyutunda olması gibi sorunlarla karşılaşılmıştır. Bu sorunu çözmek için binalar bir adet birim küp referans alınarak yeniden boyutlandırılmış ve bu boyutlanmış binalar kullanılmıştır.

Binaların ön yüzleri yola bakacak şekilde döndürüldükten sonra bina parselinin içine tam sığanlar bir listeye alınır ve bu listeden bu parsele konacak bina rastgele seçilir.

Her binanın girişinin kaldırıma yakın olması istendiğinden binalar kaldırımlara yaklaştırılmışlardır.

Parsellerin dikdörtgen olmaması durumunda binaların yerleştirilmesi için de bu çalışmada uygulanmasa da bir yöntem düşünülmüştür.

Öncelikle bir çokgenin içine sığabilecek en büyük daire elde edilmelidir. Dışbükey çokgenlerde bu işlem nispeten kolay olsa da, iç bükey çokgenlerde biraz daha zorlu olacaktır. Bu durumu aşmak için de içbükey çokgeni en büyük dışbukey çokgen verecek şekilde parçalamak işe yarayacaktır.

Taban köşegeninin yarısı, hesaplanan dairenin yarıçapına eşit ya da küçük olan binalar arasından rastgele seçim yapıp binanın merkezi dairenin merkezine gelecek şekilde parselin üzerine yerleştirilebilir.

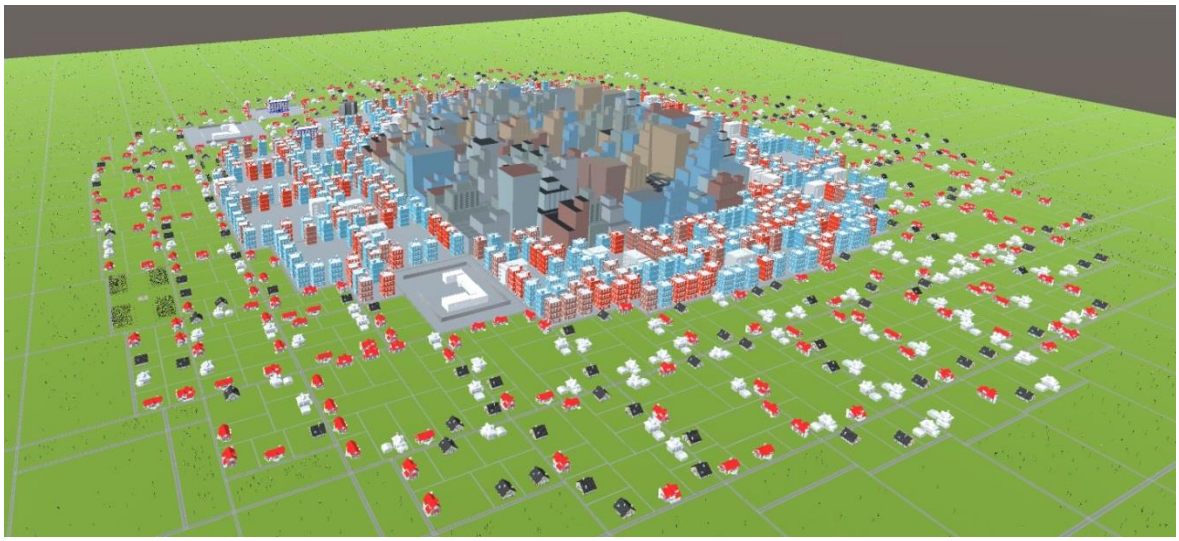
4. SONUÇ

4.1. Sonuç

Yukarıda bahsi geçen algoritmalar kullanılarak yöntemsel şehir üretim aracı geliştirilmiştir. Bu aracın arayüzünden zonların dış sınırlarının uğrayabileceği minimum ve maksimum değerler girilerek bir butona basmak suretiyle birbirinden farklı şehirler oluşturulabilir.

Kullanılan metodlar stokastik oldukları için aynı değerlerin girilmesi durumunda aynı sonuçlar değil rastgele sonuçlar çıkacaktır.

Tez sonucu geliştirilen projeden üretilmiş bir şehrin perspektiften (Şekil 4.1) ve üstten (Şekil 4.2) görünüşü resimlerdeki gibidir.



Şekil 4.1. Şehrin yandan görünüşü

Proje exe formatında build alıp çalıştırıldığında yaklaşık 860 binalık bir şehri 7.2 saniyelik bir sürede oluşturmuştur. Sahnenin FPS oranları da 5 ile 20 arasında değişmektedir.

4.2. Karşılaştırma.

Tezde ortaya çıkan algoritmayı kullanarak şehir üreten uygulamamız Citygen beta sürümü ve CityScape isimli yöntemsel şehir üreten 2 adet uygulama ile karşılaştırışmıştır.

Citygen küçük boyutlardaki şehirleri, çok az olarak üst üste gelen kaplamalarla, tatmin edici bir şekilde oluştursa da; şehir boyu büyüdükçe oluşan şehirde model ağı bozulması, üst üste gelmiş kaplamalar ve oluşmayan şehir bölümleri gibi hatalar gözlemlenmiştir.

CityScape, Manhattan modeli gibi ızgara yol ağına sahip bölgelere sahip olmayan şehirler üretmektedir. Bina modelleri oldukça gerçekçi üretilse de şehir yerleşimi hep aynı olmaktadır. Enine ve boyuna giden yollarla oluşturulmuş ızgara yol ağı arasında kalan her bir alana sadece bir adet bina yerleştirilmiştir. Izgaranın aynı satırındaki binaların taban

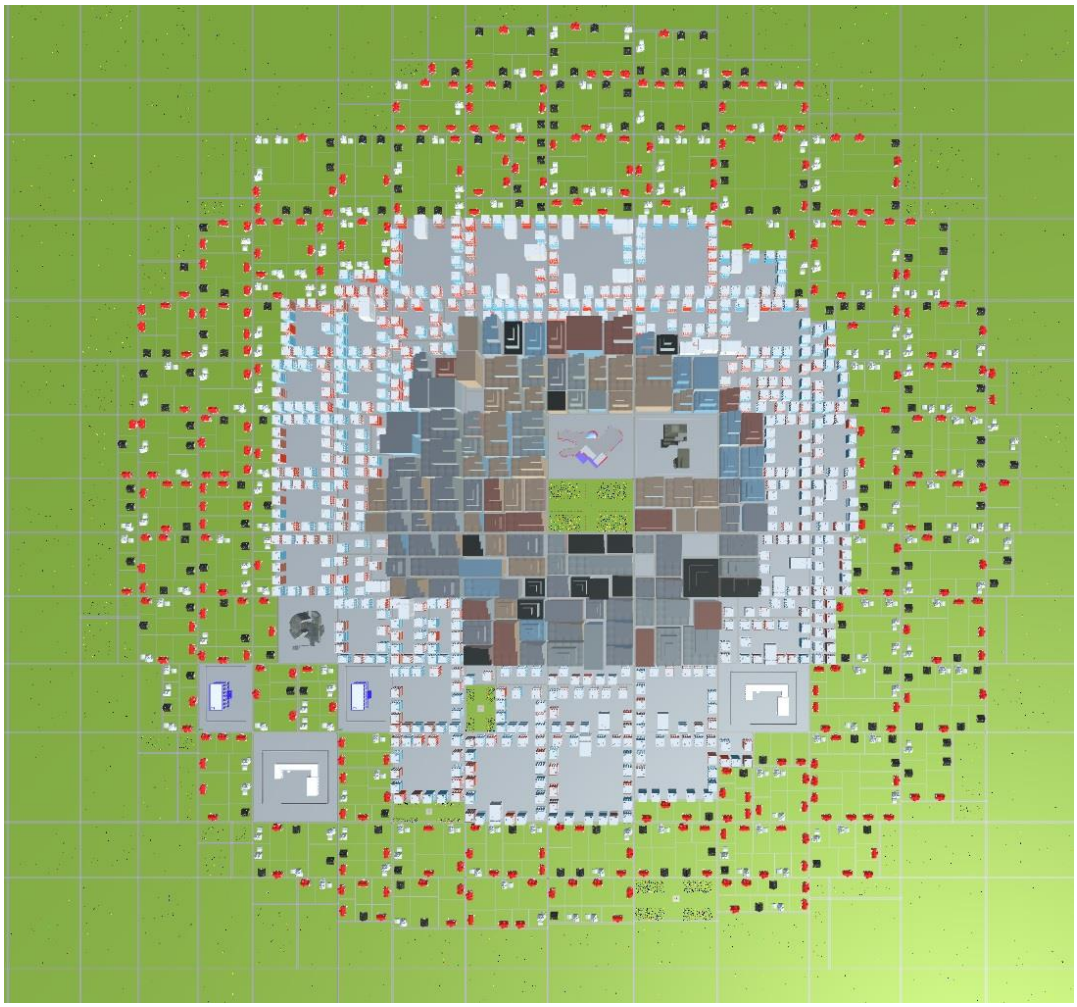
alanlarının uzunlukları ve aynı sütunundaki binaların taban alanlarının genişlikleri aynı olacak şekilde binalar oluşturulmuştur. Binaların tüm kenarlarından yollar geçmektedir. Ve ikincil yol ağı barındırmamaktadır. CityScape'de ayrıca kamu alanları da bulunmamaktadır.

Tezde ortaya çıkarılan projenin üretim hızı ve RAM kullanımı CityScape ile karşılaştırılmıştır. Karşılaştırma sonuçları Çizelge 4.1'de görülebilir.

Çizelge 4.1. CityScape ile geliştirilen programın karşılaştırılması

Araç	**Üretim Hızı**	**Bina Sayısı**	**Ayrılan Hafıza**	**Kullanılan Hafıza**
Tez	18.6 sn	847	1.25 GB	0.98 GB
CityScape	60 sn	840	1.76 GB	1.43 GB

CityScape Unity Editör üzerinde çalıştığı için karşılaştırma sırasında tez projesi de derlenmeden almadan Unity Editörde çalıştırılmıştır.



Şekil 4.2. Şehrin üstten görünüşü

Tüm işlemler 8GB RAM, intel i7 işlemcili, 2GB 128-bit Ekran kartı bulunduran bir bilgisayarda gerçekleştirilmiştir.

4.3. Gelecekteki Çalışmalar

Algoritmamız ilerleyen zamanlarda şehirlerin sadece düz bir yüzey üzerinde değil bir araziye uyumlu oluşacak şekilde üretilmesini sağlayacak biçimde geliştirilebilir. Metropol benzeri büyük şehirlerde olduğu gibi birden fazla merkezi olan zon yapılarından oluşan şehirleri geliştirmek için de düzenlemeler yapılabilir. Ayrıca sahil kenarındaki şehirlerin merkezlerinin sahil kenarında, dağlık alanlara konulmuş şehirlerin merkezlerinin dağın eteklerinde olması gibi coğrafik etmenlere bağlı şehir yapılanmaları üretilen şehirlerin coğrafik konumuna bağlı şehir üretimi gerçekleştirilecek şekilde geliştirilebilir.

5. KAYNAKLAR

[1] Barnsley, M. F., Massopust, P., Strickland, H., & Sloan, A. D. (1987). Fractal modeling of biological structures. *Ann. NY Acad. Sci*, *504*, 179-194.

[2] Prusinkiewicz, P., & Hammel, M. (1993, May). A fractal model of mountains and rivers. In *Graphics Interface* (Vol. 93, pp. 174-180). Canadian Information Processing Society.

[3] Musgrave, F. K., Kolb, C. E., & Mace, R. S. (1989, July). The synthesis and rendering of eroded fractal terrains. In *ACM Siggraph Computer Graphics* (Vol. 23, No. 3, pp. 41-50). ACM.

[4] Prusinkiewicz, P., & Lindenmayer, A. (2012). The algorithmic beauty of plants. Springer Science & Business Media.

[5] Smelik, R. M., De Kraker, K. J., Tutenel, T., Bidarra, R., & Groenewegen, S. A. (2009, June). A survey of procedural methods for terrain modelling. In *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)* (pp. 25-34).

[6] Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D. S., ... & Zwicker, M. (2010, December). A survey of procedural noise functions. In *Computer Graphics Forum* (Vol. 29, No. 8, pp. 2579-2600). Oxford, UK: Blackwell Publishing Ltd.

[7] Anonim, Perlin Noise: Part 2, https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise-terrain-mesh (Erişim Tarihi:2019)

[8] Strandgaard, S., Brain Texture - Perlin Noise, https://www.flickr.com/photos/12739382@N04/2914851309 (Erişim Tarihi:2019)

[9] Cohen, M. F., Shade, J., Hiller, S., & Deussen, O. (2003). Wang tiles for image and texture generation (Vol. 22, No. 3, pp. 287-294). ACM.

[10] Unity, Create Neighbor Terrains, https://docs.unity3d.com/Manual/terrain-CreateNeighborTerrains.html (Erişim Tarihi: 2019)

[11] Shaker, N., Liapis, A., Togelius, J., Lopes, R., & Bidarra, R. (2016). Constructive generation methods for dungeons and levels. In Procedural Content Generation in Games (pp. 31-55). Springer, Cham

[12] Vanegas, C. A., Kelly, T., Weber, B., Halatsch, J., Aliaga, D. G., & Müller, P. (2012, May). Procedural generation of parcels in urban modeling. In Computer graphics forum (Vol. 31, No. 2pt3, pp. 681-690). Oxford, UK: Blackwell Publishing Ltd.

[13] Chin, N., & Feiner, S. (1989). Near real-time shadow generation using BSP trees. ACM SIGGRAPH Computer Graphics, 23(3), 99-106.

[14] Groenewegen, S., Smelik, R. M., de Kraker, K. J., & Bidarra, R. (2009, January). Procedural City Layout Generation Based on Urban Land Use Models. In Eurographics (Short Papers) (pp. 45-48).

[15] Yang, Y. L., Wang, J., Vouga, E., & Wonka, P. (2013). Urban pattern: Layout design by hierarchical domain splitting. ACM Transactions on Graphics (TOG), 32(6), 181.

[16] Kelly, G., & McCabe, H. (2007, November). Citygen: An interactive system for procedural city generation. In Fifth International Conference on Game Design and Technology (pp. 8-16).

[17] van der Zee, A., & de Vries, B. (2012). Modeling of RL-Cities. In 30th International Conference on Education and Research in Computer Aided Architectural Design in Europe (eCAADe 2012) (pp. 375-380). eCAADe and CVUT, Faculty of Architecture.

[18] Sun, J., Yu, X., Baciu, G., & Green, M. (2002, November). Template-based generation of road networks for virtual city modeling. In Proceedings of the ACM symposium on Virtual reality software and technology (pp. 33-40). ACM.

[19] Parish, Y. I., & Müller, P. (2001, August). Procedural modeling of cities. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (pp. 301-308). ACM.

[20] Wonka, P., Wimmer, M., Sillion, F., & Ribarsky, W. (2003). Instant architecture (Vol. 22, No. 3, pp. 669-677). ACM.

[21] Anonim, Drunkard's Walk, http://pcg.wikidot.com/pcg-algorithm:drunkard-walk (Erişim Tarihi: 2019)

[22] Kelly, G., & McCabe, H. (2006). A survey of procedural techniques for city generation. The ITB Journal, 7(2), 5.

[23] Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D. S., ... & Zwicker, M. (2010, December). A survey of procedural noise functions. In Computer Graphics Forum (Vol. 29, No. 8, pp. 2579-2600). Oxford, UK: Blackwell Publishing Ltd.

[24] Dykeman, I., Procedural Worlds from Simple Tiles, https://ijdykeman.github.io/assets/wang_tiles/map_explanation.svg

[25] Anonim, Voronoi Diagram, http://mathworld.wolfram.com/VoronoiDiagram.html

[26] Rodrigues, J., How to procedurally generate a dungeon using the BSP method on Unity, https://bladecast.pro/unity-tutorial/how-to-procedurally-generate-a-dungeon-bsp-method-unity-tilemap

[27] González-Medina, D., Rodríguez-Ruiz, L., & García-Varea, I. (2016). Procedural city generation for robotic simulation. In Robot 2015: Second Iberian Robotics Conference (pp. 707-719). Springer, Cham.