YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

POUL BOURKE ÜÇGENLEME YÖNTEMİ İLE YERSEL LAZER TARAYICI VERİLERİNDE GÜRÜLTÜ ELİMİNASYONUNA YÖNELİK ALGORİTMA GELİŞTİRME

Harita Mühendisi, İrem Füsun TOMURCUK

FBE Harita Mühendisliği Anabilim Dalında Uzaktan Algılama ve Coğrafi Bilgi Sistemleri Programında Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Doç. Dr. Bülent BAYRAM

İSTANBUL, 2010

İÇİNDEKİLER

KISALTMA LİSTESİ

CAD	Bilgisayar Destekli Tasarım (Computer Aided Design)
CAE	Bilgisayar Destekli Mühendislik (Computer Aided Engineering)
CAM	Bilgisayar Destekli Üretim (Computer Aided Manufacturing)
CMM	Koordinat Ölçme Aleti (Coordinat Measuring Machine)
GG	Gabriel Çizgileri
RE	Tersine Mühendislik (Reverse Engineering)
3B	Üç Boyut

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖNSÖZ

Yersel lazer tarayıcılarla elde edilen 3 boyutlu veri, farklı disiplinler tarafından kullanılmaktadır. Günümüzde birçok mühendislik uygulamalarında, tarihi ve kültürel mirasın korunmasında, rölöve ve restorasyon gibi vb. çalışma alanlarında yer alan Lazer teknolojisinde nesne, yansıma yoğunluğu verisi içeren 3 boyutlu nokta verisi olarak elde edilmektedir. Nokta bulutlarının kaydedilmesi, birleştirilmesi, inceltilmesi, nokta boşluklarının doldurulması, filtrelenmesi ile nesnelerin 3 boyutlu modelleri oluşturulmaktadır. Oluşturulan bu modeller üzerinden, mühendislik uygulamaları için gerekli her türlü veriye ulaşılabilmektedir.

Bu çalışmada lazer tarayıcılardan elde edilen bir yüzeye ait nokta bulutu verilerini kullanarak sayısal ortamda 3 boyutlu yüzey modeli oluşturulabilmesi için geliştirilen gürültü elemeye yönelik algoritma anlatılmış olup veri, modellemenin sonraki adımlarına hazır hale getirilmiştir. Yapılan üçgenleme, hesaplanan normal vektörü geliştirilen algoritmaya yönelik kriterlerin ve eşik değerin belirlemesinde yardımcı olmuştur. Yazılım C# dilinde kodlanmıştır.

Tez çalışmalarım sırasında ve yazılım geliştirme aşamalarında gösterdiği sabır ve desteğinden ötürü tez danışmanım Doç. Dr. Bülent Bayram'a, yardım ve sonsuz sevgilerini eksik etmeyen ailem arkadaşlarım ve tüm sevdiklerime çok teşekkür eder daima yanımda benimle olmalarını dilerim.

ÖZET

Günümüz teknolojisinin hızlı değişimini fark etmemek mümkün değildir. Lazer tarama teknolojisi de hızla uygulama alanını genişletmekte, getirdiği kolaylıkları günbegün artmaktadır. Endüstriyel uygulamalar, mimari uygulamalar, obje modelleme, tıbbi görüntüleme, sayısal film yapımı, şekil analizi ve sanal ortam oluşturma gibi nice alana hitap etmiş olan bu teknoloji lazer tarayıcılar tarafından elde edilen verilerin işlenmesi sayesinde oluşmuştur.

Bir ölçme tekniği aracı olan Lazer Tarayıcıları çeşitli objelere ve yüzeylere ilişkin veri toplanabilmektedir. Toplanan bu veriler üç boyutlu(3B) nokta kümeleri gibidirler. Nokta bulutu adı verilen bu kümler tarama esnasında obje yüzeyinden veya dış etken kaynaklı hatalı verilere sahip olabilirler. Bu tip hatalı veriler diğer adıyla gürültüler, yansıma ve saçılma etkisi yüzünden objenin modellenmesini olumsuz yönden etkilerler. İyi bir obje modeli oluşturulabilmesi için bu gürültülerin belirlenip var olan nokta bulutundan çıkarılması gerekmektedir.

Yüksek lisans tezi kapsamında yürütülen çalışmada lazer tarayıcılarından elde edilen bir yüzeyine ait nokta bulutu verisinde gürültüyü tespit edecek özgün bir algoritma geliştirilmesi amaçlanmıştır. Bunun için var olan nokta bulutunun üçgenlenmesi, elde edilen üçgenlerin normal vektörleri yardımıyla düzlemlere ayrılması ve düzlemde yer alan noktaların bir eşik değer karşısında büyüklükleri incelenerek gürültünün belirlenmesi sağlanmıştır. Normal vektörü yardımıyla geliştirilen gürültü eliminasyonu algoritması sayesinde bulunan gürültüler nokta bulutundan arındırılarak modellemenin diğer aşamalarını yapabilen yazılımlara hazır hale getirilmiştir.

Anahtar Kelimeler: Nokta bulutu, gürültü, lazer tarayıcı, üçgenleme, Poul Bourke Üçgenleme Algoritması, gürültü eliminasyonu algoritması.

ALGORITHM DEVELOPMENT FOR NOISE ELIMINATION AT TERRESTRIAL LASER SCANNER DATA WITH POUL BOURKE TRIANGULATION METHOD

ABSTRACT

It is not possible to realize changing of the technology. Laser technology has widen to field of applicaiton and raises convenience from day to day. Industrial applications, architectural applications, object modelling, medical image processing, digital movie making, object extraction, virtual environment creating and so much fields like these, address this technology which is formed by processing of laser scanner datas.

Laser Scanners which is a kind of measuring technique equipment, gather datas concerned with different objects and surfaces. These datas like point clusters with three dimension(3D). These clusters, named Point Cloud, can have some errors caused by object surface or outer factors. This kind of datas, which are named noises, influence object modelling negatively cause by reflection and scattering effects. For making a good object model, these noises must be determined and removed from the point cloud.

In this study, an orginal algorithm development which determine and eliminate noises in the point cloud data that belongs to an object surface, is aimed. For this study, trangulating point cloud, separating triangles to plane by normal vectors and determining noises by comparing value of points that are on the plane with a threshold value, are proved. Noises, which is determined and eliminated from the point cloud by the algorithm with normal vector, have being ready for softwares that can make other steps of modelling.

Key Words: point cloud, noise, laser scanner, trangulation, poul bourke trangulation algorithm, noise elimination algorithm

1. GİRİŞ

Bilgisayar destekli tasarım (Computer Aided Design, CAD) popüler hale geldikçe, tersine mühendislik (Reverse Engineering, RE), 3B bilgisayar destekli tasarım, bilgisayar destekli üretim (Computer Aided Manufacturing, CAM), bilgisayar destekli mühendislik (Computer Aided Engineering, CAE) ve diğer yazılımlarda kullanılmak üzere, var olan parçaların üç boyutlu sanal modellerinin yaratılması kullanılabilir bir yöntem haline gelmiştir. Bir objenin ölçümü ve ardından oluşturulan üç boyutlu modelleme tekniği günümüzde birçok disipline yarar sağlamaktadır. Bu ölçüm, koordinat ölçme makinesi (Coordinat measuring machine, CMM), üç boyutlu üçgenli lazer tarayıcılar, üç boyutlu yapısal ışık sayısallaştırıcılı tarayıcı veya bilgisayarlı tomografi gibi üç boyutlu tarama teknolojileri kullanılarak yapılır. (Dayık ve diğerleri, 2008).

Lazer tarama, geleneksel ölçme teknikleri ile kıyaslandığında üç boyutlu (3B) nokta bilgilerinin çok yüksek hızla elde edildiği bir ölçme tekniğidir. Ölçme alanın 3B nokta bilgileri, nokta dizileri şeklinde yüksek doğrulukta ölçülebilmektedir (Altunbaş ve Yıldız, 2008) . Yersel lazer tarayıcıları yakın geçmiş zamanda dahil şu sıralar çeşitli uygulamalarda kullanılan yeni bir araçtır. Günümüzde mühendislik ölçmelerinin genişleyen dallarında, kültürel mirasın korunmasında, deformasyon ölçmelerinde vb. pek çok uygulamada yaygın olarak kullanılmaktadır. Üç boyutlu (3B) veri işleme ve görselleştirmedeki gelişmeler, taramalar sonucu üretilen büyük miktarlardaki noktaları kullanılabilir hale getirmiştir (Gümüş ve diğerleri, 2009).

Yapılan çalışmanın niteliğine göre binlerce hatta on binlerce üç boyutlu koordinatı bulunan noktalardan oluşan nokta bulutu, alımı yapılan yüzeyi en iyi şekilde temsil etmeli ve ideal olarak modellenmelidir (Gopi ve diğerleri, 2000). Tarama sonunda elde edilen nokta bulutunun var olan objenin sanal bir benzeri haline gelebilmesi için sistematik ya da sistematik olmayan kimi hatalardan arınması, hatalar yüzünden oluşan eksikliklerin tamamlanması ve tamamlananlardan sonra istenen sanal gerçek yüzeyin oluşturulabilmesi için bir takım düzeltme adımlardan geçmesi gerekmektedir.

Lazer tarayıcılardan hassas olarak çeşitli objelere ve yüzeylere ilişkin elde edilen üç boyutlu nokta verileri, her ölçümde olası ihtimali taşıyan hatalar içerebilir. Bu hataların giderilmesi gerçeğe en yakın modelin oluşmasını sağlar (Morita ve diğerleri, 2003).

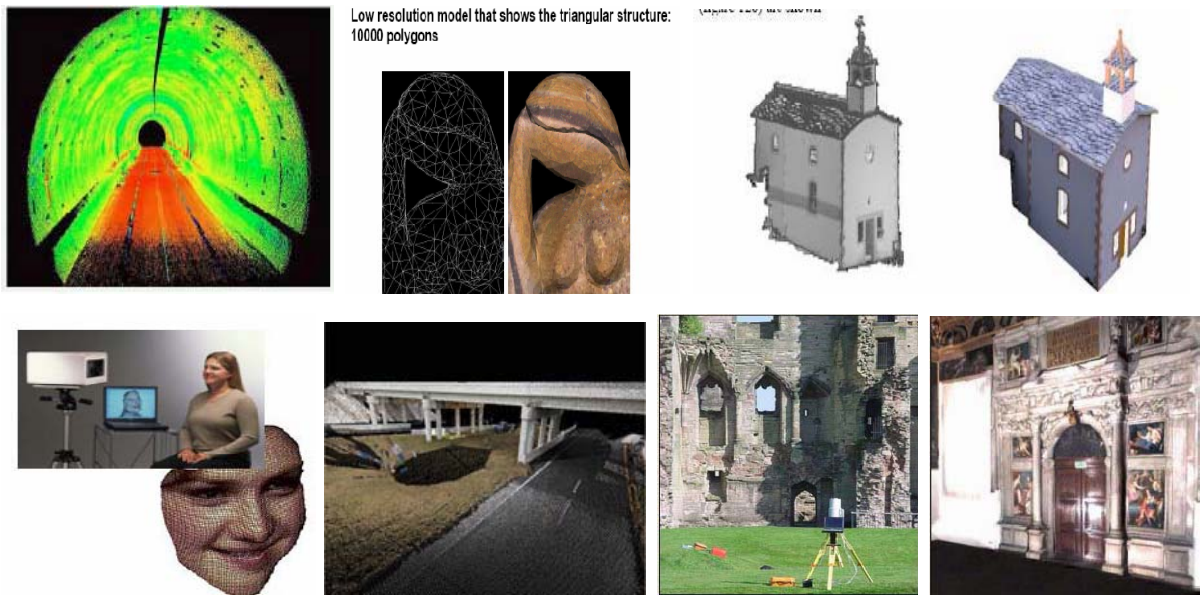
Yüksek lisans tezi, bahsi geçen tarama sonrasında elde edilen nokta bulutunun sahip olduğu ama gerçek modele ulaşmamıza engel olacak olan tarama esnasında obje yüzeyinden veya dış etken kaynaklı hatalı verilerin diğer adıyla gürültülerin tespiti ve bu kümeden arındırılmasını amaçlamaktadır.

Gürültüler, yansıma ve saçılma etkisi yüzünden oluşurlar. Bu, objenin modellenmesini olumsuz yönde etkiler. Modelin yanlış boşlukların oluşmasına, yanlış yüzeylerin belirlemesine ve bu hatalar yüzünden yanlış detayların elde edilmesine neden olur. Sonuçta hedeflenenden farklı bir modele ulaşılır (Morita ve diğerleri, 2003). Bunları engellemek için gürültünün elenmesi gerekmektedir. Gürültü, yapılan gözlemlerden uzaklara sapan, uyuşumsuz veridir. Nokta bulutundaki verilerin birbirleri ile olan ilişkilerince ortaya çıkan bilgiler, gürültüleri elemine etmede yardımcı olurlar (Sotoodeh,2007). Yapılan çalışmalara bakıldığında gürültülerin belirlenebilmesi için gerçekte var olan objeye ait doğruların ölçüt haline geldiği, bunlar sayesinde nokta bulutunda olması ve olmaması gereken noktaların belirlenebildiği görülmüştür. Bu tez çalışmasında da, seçilen Poul Bourke üçgenleme tekniği ve hesaplanan normal vektör yardımıyla bir ölçüt oluşturulması hedeflenmiştir. En az üç noktadan bir düzlemin oluşması noktalar arasındaki en basit ilişkiyi gösterir. Bahsedilen ilişki ile modellerde oluşturulan yüzeylerin en küçük birimi olan üçgenlerden çıkış alan bu ölçüt sayesinde, nokta bulutunda yer alan gürültüyü tespit eden ve var olan nokta bulutundan elimine eden özgün bir algoritma geliştirilmiştir. Bu algoritma C# programla dili ile yazılarak da hayata geçirilmiştir.

2. LAZER TARAMA TEKNOLOJİSİ

Laser İngilizce; Light Amplification by Stimulated Emission of Radiation (uyarılmış ışın salınımıyla ışığın kuvvetlendirilmesi) cümlesindeki kelimelerin baş harflerinin alınmasından türetilmiş bir kelimedir. Bu ışınlardan yararlanılarak laser tarayıcı sistemleri geliştirilmiştir (Demir, 2005).

3B lazer tarama teknolojisi, yeni teknoloji alanında bir ölçme tekniği olarak yeni bir gelişim yönünün önünü açmıştır. Lazer ışını kullanılarak bir nesne veya bir yüzeyin uzaklığını anlamaya yarayan teknoloji olmakla birlikte nesne ya da yüzeye gönderilen lazer darbesinin gönderiliş zamanı ile nesneye çarpıp gelen yansımanın tekrar kaynağa ulaşma vakti arasındaki fark sayesinde uzaklık ölçülür (Kıvılcım,2005).

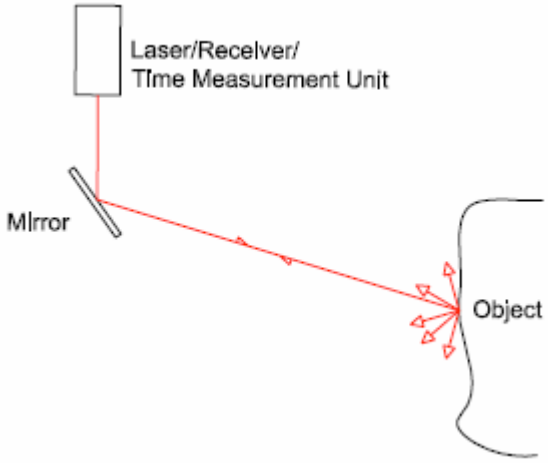


Şekil 2.1 Uygulama alanları

Bina, anıt, gibi mimari eserlerin yanı sıra yol, köprü v.b. mühendislik yapılarının 3 Boyutlu olarak belgelendirilmeleri geleneksel jeodezik ve fotogrametrik yöntemler ile yıllardır yapılmaktadır. Üretilen CAD verileriyle modelden bilgi elde etmek, modelleme ve düzeltmeler yapmak kolaydır. Günümüzde yersel lazer tarayıcı sistemleri ile ölçeğe bağlı kalınmaksızın modelleme yapılabilmektedir (Kıvılcım,2005). Lazer tarayıcılar farklı platformlarda, tıbbi görüntülemeden, endüstriyel tasarıma kadar tarihi ve kültürel eserlerin belgelendirilmesi ve projelendirilmesinde birlikte yersel lazer tarayıcılar geniş bir kullanım alanı bulmaktadır (Şekil 2.1).

Lazer tarayıcılar, kullanıcısı için nesnenin yüzeyine ait noktaların üç boyutlu koordinatlarını toparlayabilen alettir. Başka bir değişle lazer tarayıcılar, nesne yüzeyini tarayarak, otomatik ve sistematik bir desen oluşturan, saniyede yüzlerce hatta binlerce nokta verisi toparlayabilen çok kısa sürede sonuç ürün olarak üç boyutlu koordinat bilgisine ulaşabilinen aletlerdir.

Yersel lazer ölçmelerinde temel büyüklük, alet ve ölçülen nokta arasındaki mesafedir. Lazer mesafe ölçümü için farklı teknikler kullanılmaktadır. Bunlar; üçgenleme, faz farkı ölçümü, ışığın gidiş/dönüş zamanı ölçümü ya da puls metodudur. Yersel lazer tarayıcılarında kısa zaman aralıklarıyla lazer pulslarının gönderilmesi ve ölçülmesi esasına dayanan puls metodu kullanılmaktadır (Altunbaş ve Yıldız, 2008).



Şekil 2.2 Işığın gidiş/dönüş zamanı ölçümü (Gümüş ve Erkaya, 2007)

Etkin bir veri toplama tekniği olan lazer tarayıcılar hem ölçmecilere hem de bu ölçüleri kullananlara büyük kolaylıklar sağlar. Lazer tarama yönteminin avantajları;

- Hızlı ve obje ile temas kurmadan ölçme,

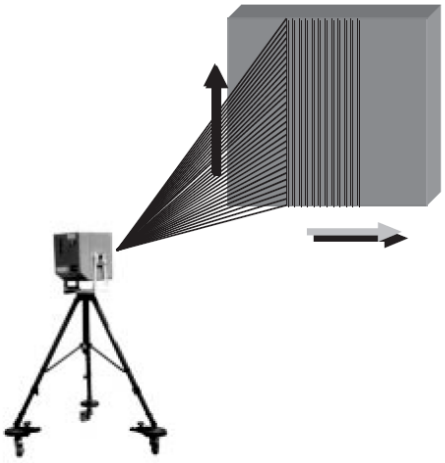
- Aynı ölçme alanı için daha fazla veri toplama,

- Lazer ölçülerinin var olan başka tür ölçülerle kolayca entegrasyonu,

- Daha güvenli veri toplama imkânı,

- Gerçek renkli görüntü üretebilme,

- Ölçme alanının belirli periyotlarla tamamen ölçülebilmesi olarak sıralanabilir (Waggot ve diğerleri, 2005).



Şekil 2.3 Yersel tarayıcının çalışma prensibi (Kıvılcım,2005)

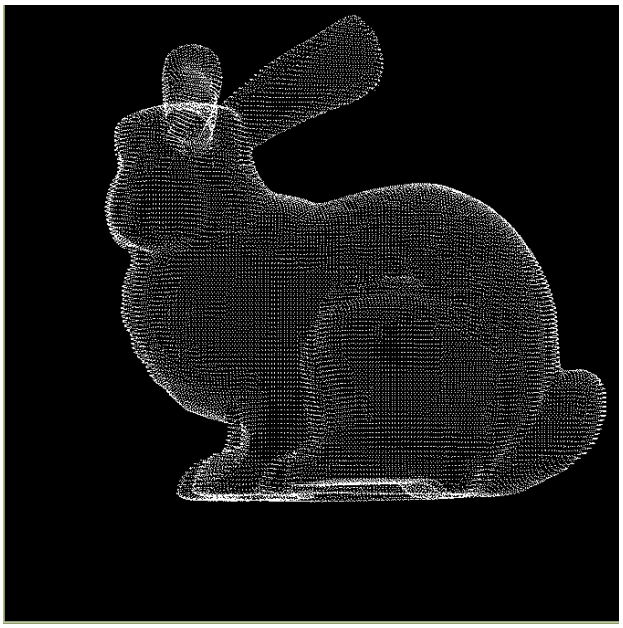
Lazer tarayıcılarla, ölçülecek alanın 3B nokta verileri istenilen aralıklarla çok yüksek hızla ölçülür. Tarayıcıların tarama teknikleri satır tarama ya da sütun tarama şeklinde olabilir (Şekil 2.3). Tarayıcıların çoğu ölçülecek alanın tamamı için satır ya da sütunlar oluşturacak şekilde hareket ederek tarama yapar (Altunbaş ve Yıldız, 2008). Tarama bitiminde ise objeye ait 3B nokta kümesi oluşturulmuş olur.

Şekil 2.4 Nokta bulutu (Hohner, 2009)

3. NOKTA BULUTU

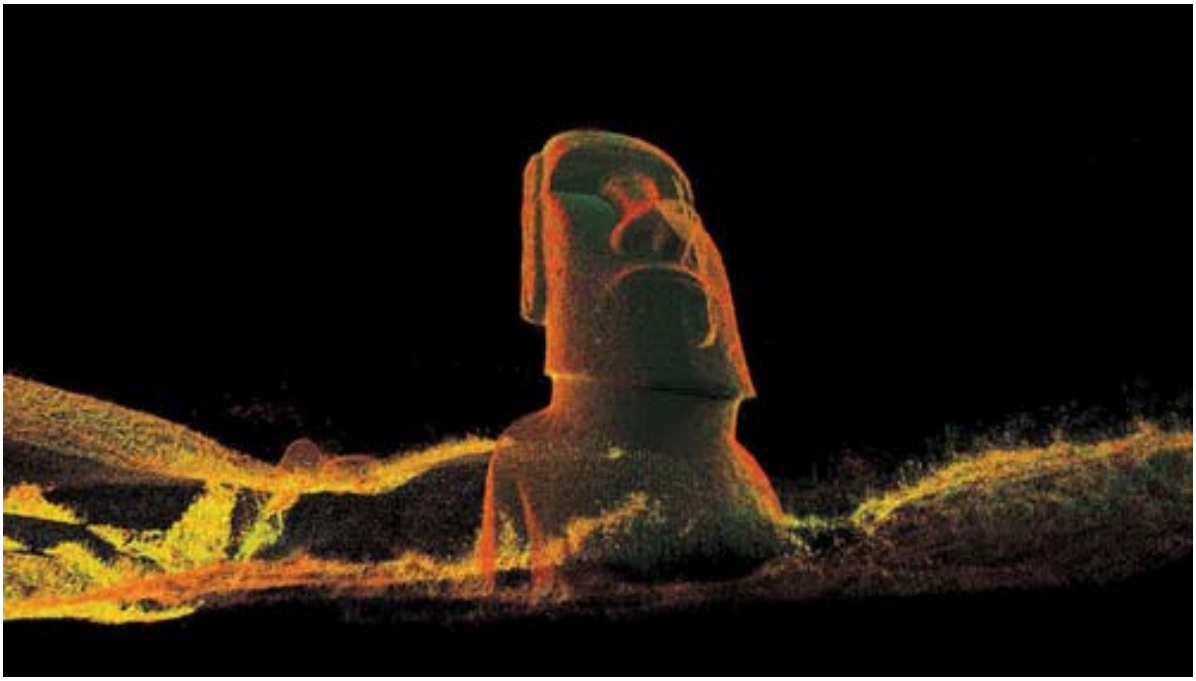
Bir lazer tarayıcıdan gönderilecek her bir lazer ışını ile bir nesnenin lazer tarayıcısına olan mesafesi noktalar kümesi halinde elde edilir. Yersel lazer tarayıcı ile taranacak yüzey üzerindeki bir nokta arasındaki uzunluk, lazer sinyalinin yüzeye gönderilmesi ve yüzeyden geri dönen lazer sinyalinin tespiti arasında geçen zamanın yüksek doğrulukla belirlenmesiyle hesaplanır. Obje, optik mekanik tarayıcılar ile ölçme uzunluğuna bağlı olarak, yatay ve düşey yönlendirmelerle taratılır. Tarama işlemi sonucunda elde edilen, objenin milyonlarca noktadan oluşan detaylı 3B görüntüsünün çıkarılmasını sağlayan, yoğun lazer sinyallerinin oluşturduğu nokta kümelerine nokta bulutu denir (Gümüş ve Erkaya, 2007).

Nokta bulutu, sayısal ortama aktarılan ve birbirine belli aralıklarda bulunan pek çok sayıda noktadan oluşan üç boyutlu bir yapıdır. Bu yapıyı oluşturan ve yapılan çalışmanın niteliğine göre binlerce hatta on binlerce üç boyutlu noktalardan oluşan nokta bulutu, alımı yapılan yüzeyi en iyi şekilde temsil etmelidir.



Şekil 3.1 Nokta bulutu (Yugang Min)

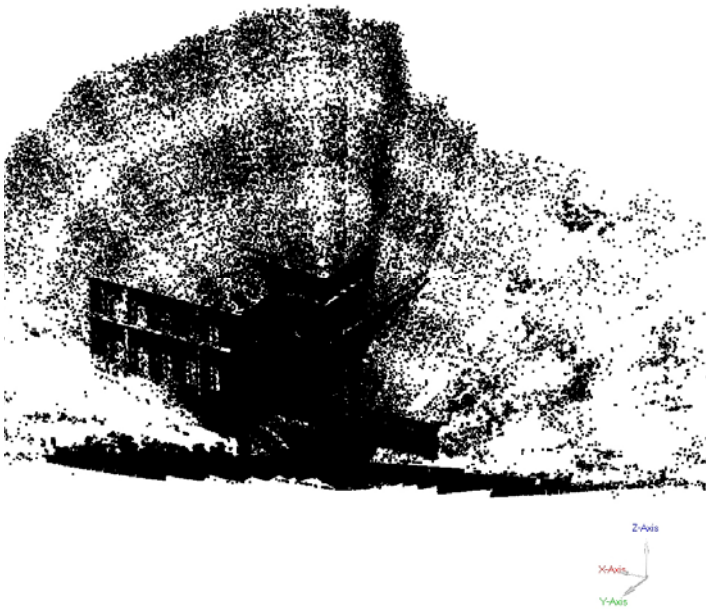
Bir lazer tarayıcı, tarama sonucu taranan objenin yüzeyini oluşturan bir nokta bulutu üretir (Point Cloud). Bu noktalar objenin temel 3B formunu oluşturmakta kullanılır. Her nokta için alınan renk bilgisi de eklenerek dokular yaratılır ve yüzey kaplanır. Gerekiyorsa 3B düzenleme programına aktarılarak (3D Max, LightWave, MAYA, SoftImage vb..) isteğe uygun olarak işlenebilir.



Şekil 3.2 Nokta bulutu (Beesley, 2008)

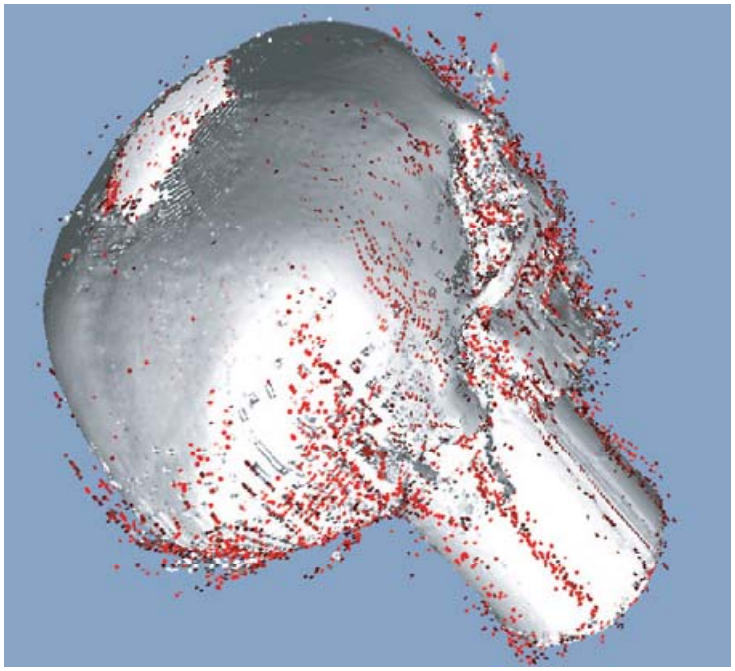
4. GÜRÜLTÜ

Lazer tarayıcılarla belirlenen nokta bulutunda yer alan ama normal olarak sanal objeye ait olmayan noktalardır. Diğer gözlemlerden uzaklara sapan ve başka bir nedenle ve başka bir mekanizma tarafından oluştuğu şüphesi uyandıran ölçüme gürültü denir (Sotoodeh, 2006).



Şekil 4.1 Gürültülü veri (Sotoodeh, 2007)

Şekil 4.2 Gürültülü veri (Bornaz ve Rinaudo, 2004)



Şekil 4.3 Gürültülü veri (Weyrich vd., 2004)

Lazer tarayıcılardan elde edilen veriler yansıma ve saçılma kaynaklı gürültü içermektedir (Şekil 4.3). Alınan noktalar pürüzsüz bir yüzey oluşturacak şekilde değillerdir. Noktalar dağınıktır. Bu nedenle noktaların sadeleştirilmesi ve ideal bir yüzey değerinin belirlenmesi gereklidir.

4.1 Gürültünün Oluşum Nedenleri

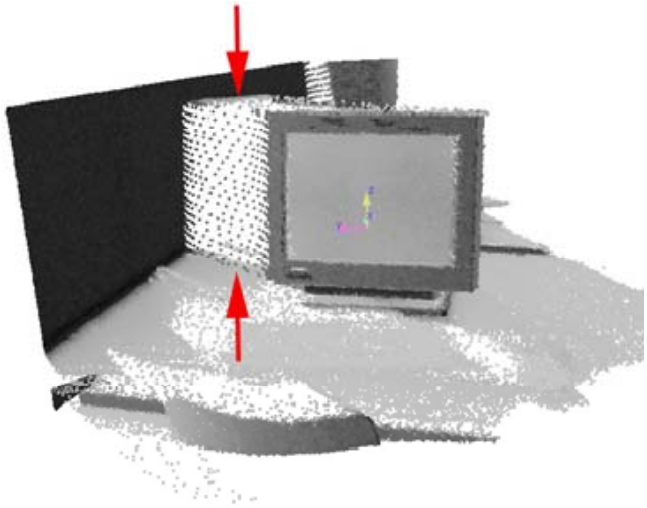
Lazer taramada büyük hatalar ve normal dışı noktalar farklı nedenlerden dolayı oluşurlar. Bunlar genelde ölçümlerden kaynaklanmaktadır. Bu tip hatalar lokal yüzey geometrisine uyum sağlamazlar (Sotoodeh, 2006).

Lazer taramada gürültülü noktaların oluşum sebeplerini üç durumda incelenebilir:

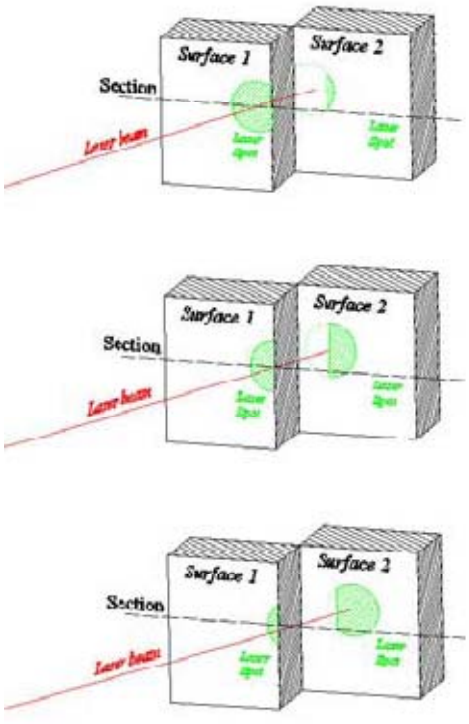
- Kapama sınırları(Boundaries of occlusions)
- Yüzey yansıması (Surface reflectance)
- Çok yollu yansıma(Multi path reflection) (Sotoodeh,2006).

4.1.1 Kapama sınırları (Boundaries of occlusions)

Lazer ışınları bir nokta değildir footprint denilen izlerdir ve bu izler elipstir. Bu ışınlar, aynı ışın doğrultusuna denk gelen iki obje arasında kalan bölüm olan kapatma sınırına çarptığında ışını ikiye böler. Bu oluşan hatalı noktalar uyuşumsuz bir başka değişle gürültüdür (Sotoodeh,2006).



Şekil 4.1.1.1 Kapama sınırı nedeniyle oluşan gürültü (Sotoodeh,2006)

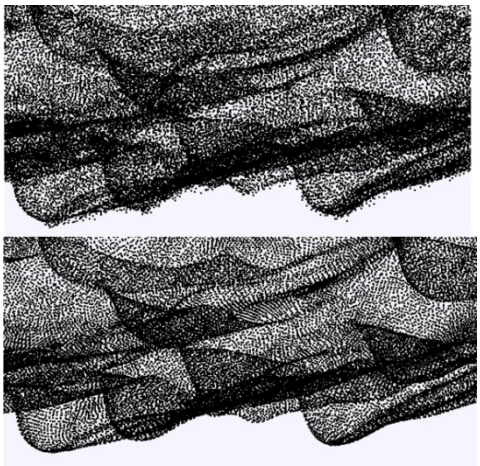


Şekil 4.1.1.2 Kapama sınırında footprint (Sotoodeh,2006)

Ayrılmış bir footprint hatalı bir ölçümün göstergesidir. İki yüzey ve lazer ışının geldiği nokta bu tip bir hatanın oluşumuna sebep vermiş olup asıl objenin sınırını etkilemiştir. Asıl objeyi elde etmek içinse bu noktaların kaldırılması bunun yerine kapama sınırı karşısından taramanın yapılması gerekmektedir. Böylelikle asıl objenin sınırında bulunan verileri koruma altına almış, modellemesi eksiksiz yapılmış olur.

4.1.2 Yüzey yansıması (Surface reflectance)

Kimi ölçüm hataları obje yüzeyinden kaynaklanmaktadır. Bunlar fazla yüksek ya da fazla düşük yansılamalar içerirler. Lazerin geri dönüşünün atışından farklı olması bu tip bir hatanın oluşum göstergesidir. Siyah, cam, parlak metal yüzeyler bu duruma sebep olan yüzeylere örnektir (Sotoodeh,2006).

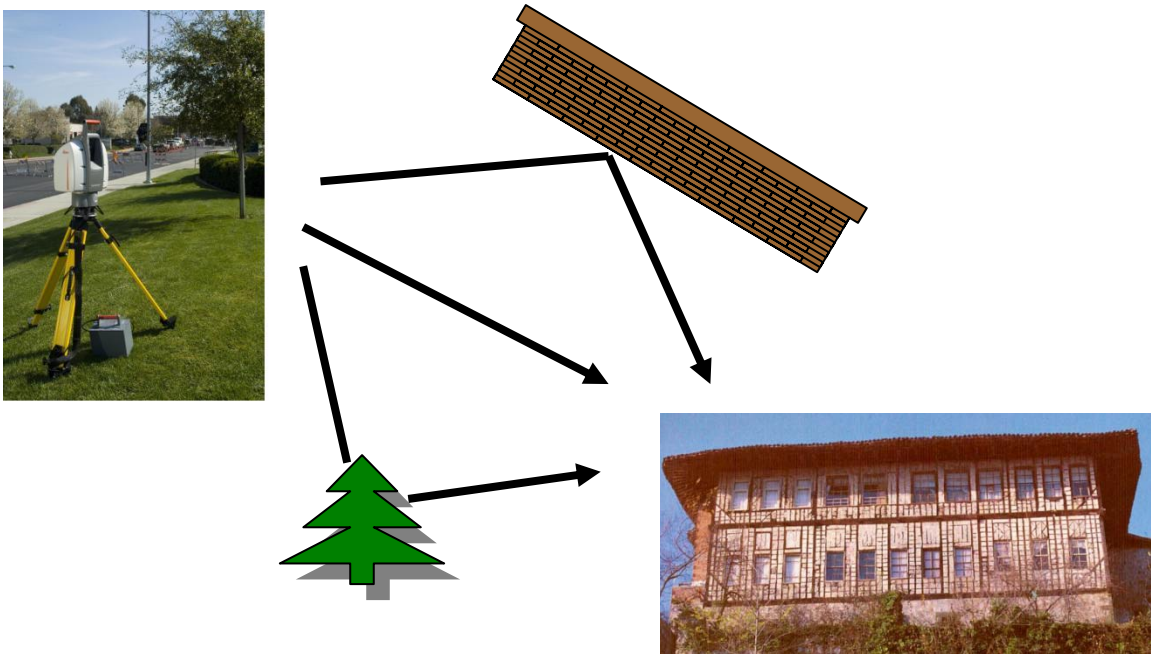


Şekil 4.1.2.1 Metal yüzey yansıması (Laser Design Inc. projeleri, 2008)

4.1.3 Çok yollu yansıma (Multi path reflection)

Lazer ışını küçük bir açı altında yüzeye çarptığında direk tarayıcıya dönmeyebilir dönerken öncelikle yakınlarındaki yüzeye çarpar (Şekil 4.1.3.1). Gönderildiği ve geri döndüğü yer farklı olduğundan ölçüm uyuşumsuz olarak nokta bulutunda yerini alır.

Geri dönemeyen kimi zayıf yansımalar, basit bir ön işlemde ışının fiziksel özellikleri dikkate alınarak belirlenebilir (Bornaz ve Rinaudo, 2004).



Şekil 4.1.3.1 Çok yollu yansıma

4.2 Gürültü Belirleme Yöntemleri

Gürültülerin bir başka değişle uyuşumsuz noktaların belirlenmesi (outlier detection) lazer tarama yöntemiyle elde edilen nokta bulutlarının modellemesi öncesinde yapılması gereken önemli bir adımdır.

Gürültü belirleme yöntemleri:

- Kümeleme (clustering)
- Derinlik tabanlı (depth-based)
- Yayılım tabanlı (distribution-based) yöntemlere dayalıdır (Papadimitriou ve diğerleri, 2002).

Bunlara ek olaraktan yoğunluk tabanlı (density-based) ve uzaklık tabanlı (distance based) yöntemler mevcuttur (Sotoodeh, 2007).

4.2.1 Yayılım tabanlı (distribution-based) yöntem

Stokastik yayılım modellerini benimsemiş belli bir seviyeye göre modelde sapmalar gösteren noktaların belirlenmesidir. Nokta bulutunda nokta değerlerinin yayılımı, objelerin lazer tarayıcısına olan uzaklığı, atış aralığı ve objenin geometrisine bağlıdır (Papadimitriou ve diğerleri, 2002).

4.2.2 Derinlik tabanlı (depth-based) yöntem

Bu yaklaşım hesaplanan geometriye ve n boyutlu konvekslerin farklı katmanlarının hesaplanmasına bağlıdır. Dıştaki katmanda kalan objeler, uyuşumsuz yani gürültü olarak belirlenir (Sotoodeh, 2007).

4.2.3 Kümeleme (clustering) yöntemi

Amaç, elemanların birbirlerine çok benzediği, ancak özellikleri birbirlerinden çok farklı olan kümelerin bulunması ve veri tabanındaki kayıtların bu farklı kümelere (gruplara) bölünmesidir (Sotoodeh, 2007).

4.2.4 Uzaklık tabanlı (distance-based) yöntem

Yaklaşım E.M. Knorr and R.T. Ng tarafından öne sürülmüştür. Veri setindeki en ufak kırılma belirlenen bir değerden büyük ise bu obje bir gürültü olarak belirlenir(Sotoodeh, 2007).

4.2.5 Yoğunluk tabanlı (density-based) yöntem

Komşusunun lokal yoğunluğuna bağlı olarak belirlenen yaklaşımdır (Sotoodeh, 2007).

4.3 Gürültü Belirleme Kriterleri

Gürültü belirlemek için birçok algoritma farklı uygulamalarda mevcuttur. Fakat bu algoritmaların çoğu mevcut veri setlerince o anki amaç doğrultusunda geliştirilmiş algoritmalardır. Üç boyutlu rastlantısal veri setlerine uygun değillerdir.

Bunun yanında nokta bulutunun yüzey sürekliliği, gürültü ve farklı yerel yoğunluklar içerebilmeleri kimi yöntemlere yöneltmiştir. Tüm bunlar uyuşumsuz noktaların ölçeğinin değişebilir kümeler halinde görülebileceğini işaret etmektedir.

Kullanılan tüm veri setleri için en iyi kümelemeyi yapabilen bir algoritma bulunmamaktadır. Çünkü tüm kümeleme algoritmalarının performansları verilerin dağılımına bağlıdır. Ama en iyisini belirlemek kullanıcının elindedir (Durmuş ve İplikçi, 2007).

Kümeleme algoritmalarında amaç, elemanların birbirlerine çok benzediği, ancak özellikleri birbirlerinden çok farklı olan kümelerin bulunması ve veri tabanındaki kayıtların bu farklı kümelere bölünmesidir. Kısaca, aynı kümede bulunan veriler diğer kümelerde bulunan verilere göre birbirlerine daha benzerdirler.

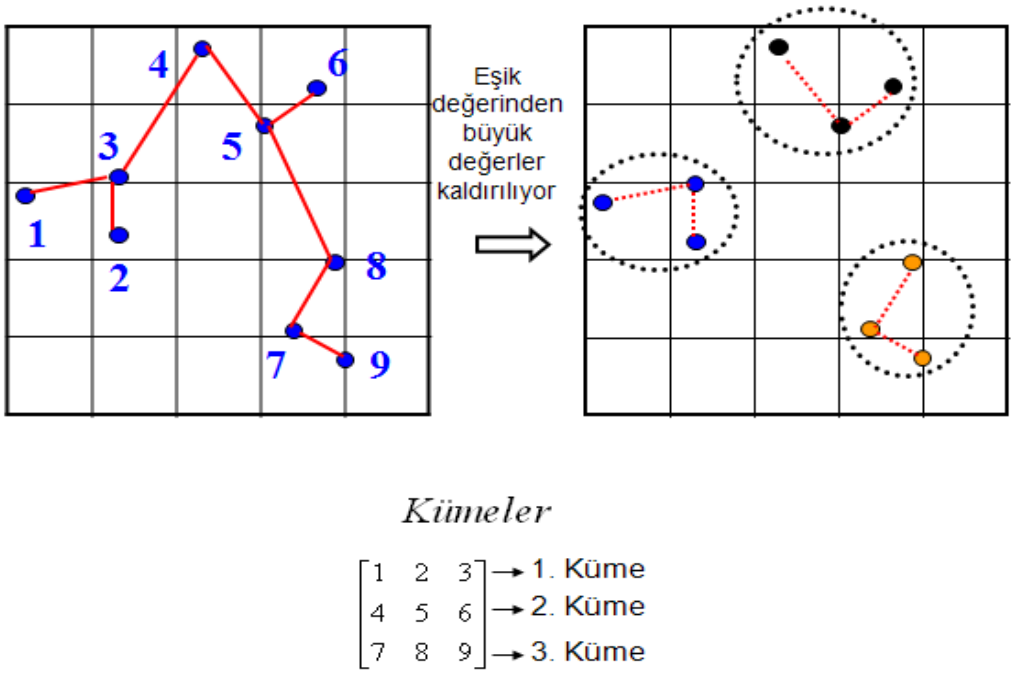
Kümeleme yöntemi sınıflandırma yöntemleri olarak adlandırılırken kontrollü ve kontrolsüz olarak iki ana başlıkta ayrılabilirler(Sotoodeh, 2007).

Kontrollü yaklaşımda farklı sınıflara ait örnek verilere ihtiyaç vardır. Kontrolsüz yaklaşımda ise amaç girdi verisinden sınıflar oluşturarak kümelemektir Ck, K yani sınıf sayısı belirlendikten sonra gruplar oluşturulur. Bu oluşumda işleme alınan ölçümlerin ortak bir niteliği, kümelemenin ölçütünü oluşturmaktır. Örneğin K kadar örneğin yoğunlukları hesaba katılır ve buna göre kümeleme yapılır(Sotoodeh, 2007).

4.3.1 Öklid Minimum Örten Ağaç (Euclidean Minimum Spanning Tree)

Bir kümeleme algoritmasıdır. İki nokta arasındaki uzaklık ağırlık olarak seçilir. Ağırlıklar toplamı en küçük olan ağaç belirlenir.

Eşik değerinden büyük ağırlığa sahip dallar ağaçtan kaldırılır. Eşik değeri yerine uyuşmayan kenar (inconsistent edge) seçimi ile de kümeler belirlenebilir (Durmuş ve İplikçi, 2007).



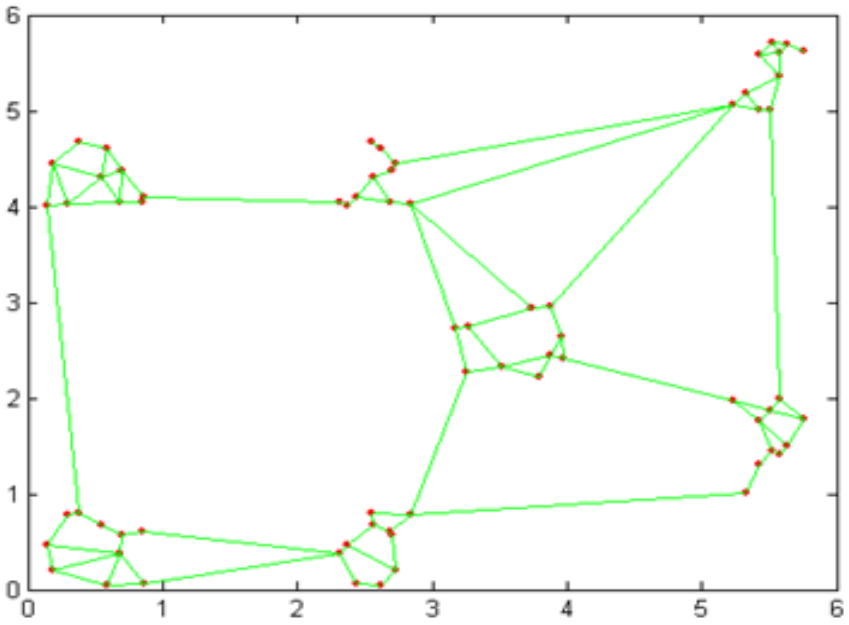
Şekil 4.3.1.1 Öklid Minimum Örten Ağaç algoritması (Durmuş ve İplikçi, 2007)

4.3.2 Gabriel Çizgeleri

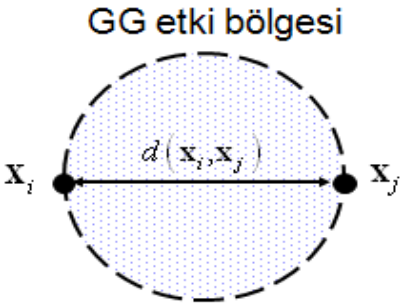
$\mathbf{x}_i$ ve $\mathbf{x}_j$ noktaları dışında hiçbir nokta DISK($\mathbf{x}_i$,$\mathbf{x}_j$)'de bulunmuyorsa, $\mathbf{x}_i$ ve $\mathbf{x}_j$ noktaları oluşturulan çizgede birbirine bağlıdır (Şekil 4.3.2.1). Yani; (1) şartı sağlandığı takdirde noktalar oluşturulan çizgeye dâhil edilir (tüm k değerleri için $k \neq i, k \neq j$).

$$d^2\left(\mathbf{x}_i,\mathbf{x}_j\right) < d^2\left(\mathbf{x}_i,\mathbf{x}_k\right) + d^2\left(\mathbf{x}_j,\mathbf{x}_k\right);\ tüm\ k\ değerleri\ için,\ k \neq i, k \neq j \quad (4.1)$$

(Durmuş ve İplikçi, 2007)



Şekil 4.3.2.1 Gabriel Çizgeleri (G.G) (Durmuş ve İplikçi, 2007)



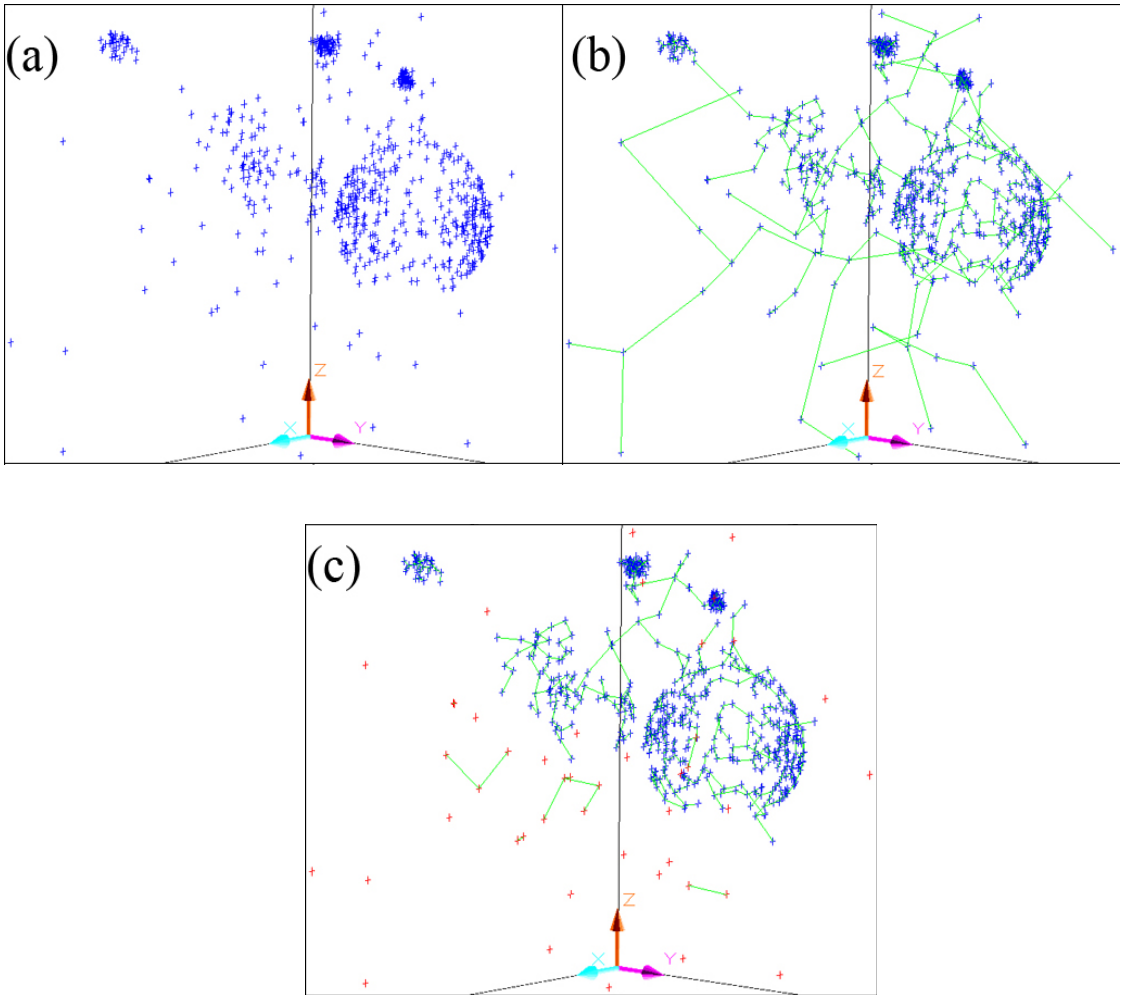
Şekil 4.3.2.2 DISK(X_i,X_j) (Durmuş ve İplikçi, 2007)

Diğer hiçbir nokta DISK(X_i,X_j)'de bulunmuyorsa, X_i ve X_j noktaları GG' de birbirine bağlıdır. DISK, GG' nin etki bölgesidir (Şekil 4.3.2.2).

Sotoodeh tarafından belirtilen bu kümeleme algortimasında ve sonucunda bulunan gürültülü verilerin temizlenmesi iki aşama yer almaktadır.

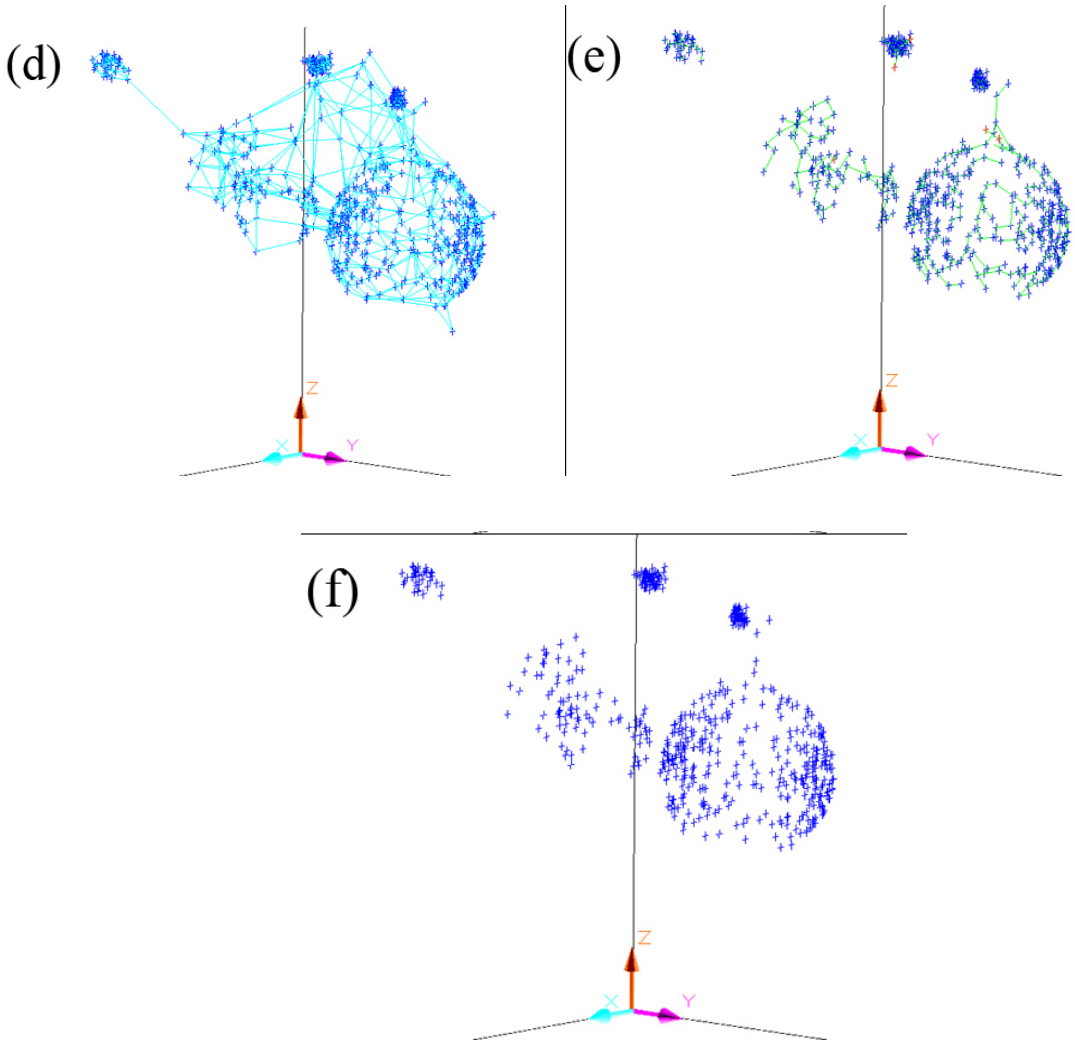
Yoğun nokta bulutundaki gürültülerin sapma aralıklarının modellenmesiyle yola çıkan algoritmada sabit bir aralığa rastlamak mümkün değildir. Bu durumda uygulamada lokal ve global olmak üzere iki belirleyici oluşturulmaktadır. 1.aşama, bir global aralık modellemesinin istatistiksel bilgilerini yakalamaya çalışıp ana kümeden çok uzakta olan noktaları belirlerken 2.aşamada küçük ölçekteki uyuşumsuz noktaları işleme alıp nokta bulutunu sınıflandırmada lokal ölçütler oluşturmaktadır (Sotoodeh, 2007).

Önce modelleme aralıklarının kaba yaklaşıkları bir kümeleme yöntemi olan Öklid Minimum Örten Ağaç (Euclidean Minimum Spanning Tree) ile belirlenir. Bu 1.adımdır(Şekil 4.3.2.3). Ağacın kenarları kenar uzunluğunun istatistiksel analizine göre budanır (Sotoodeh, 2007).



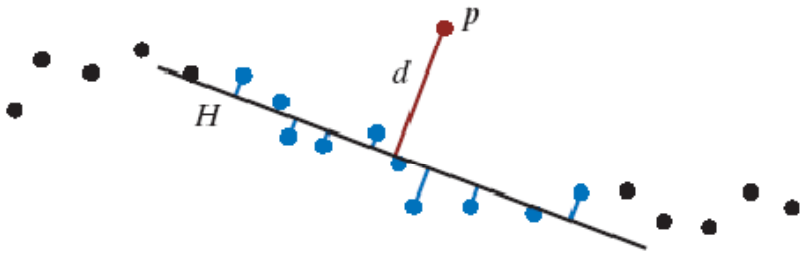
Şekil 4.3.2.3 1.adım a) Nokta bulutu b) Öklid Minimum Örten Ağaç uygulaması c) sınıfların belirlenmesi ve silinmesi gereken veriler (Sotoodeh, 2007)

Algoritmanın 2.aşamada ise her kümenin nokta bulutları için Gabriel Grafiği (GG) oluşturulur, GG' nin kenarlarınca hesaplanan istatistiğe göre her GG' nin uzun kenarı silinir (Şekil 4.3.2.4) (Sotoodeh, 2007).



Şekil 4.3.2.4 2.adım d) Gabriel grafiğinin uygulanması oluşan kenarlar e) 2.aşama sonunda belirlenen uyuşumsuzlar f) Uyuşumsuz noktalardan arınmış veri (Sotoodeh, 2007)

4.3.3 Düzleme oturma kriteri (Plane Fit Criterion)



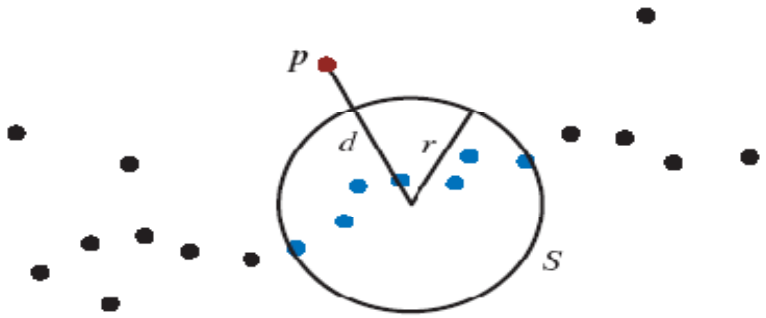
Şekil 4.3.3.1 Düzleme oturma kriteri (Weyrich vd., 2004)

$$\min_{H} \sum_{q \in \mathcal{N}_p} dist(p,H)^2 \tag{4.2}$$

$$\chi_{\mathrm{pl}}(p) = \frac{d}{d + \bar{d}}. \tag{4.3}$$

Bu kriter p noktasının komşularına en yakın düzlemi(H) içerir. P noktasının H düzlemine olan uzaklığı d ve $\bar{d}$ komşuların H düzlemine olan ortalamalarını temsil eder. Yapılan hesaplamalar (4.2)(4.3) belirlenen kriter ile gürültülerin belirlenmesi mümkünleşmiş hale gelir (Weyrich vd., 2004).

4.3.4 Mini Top kriteri (Miniball)

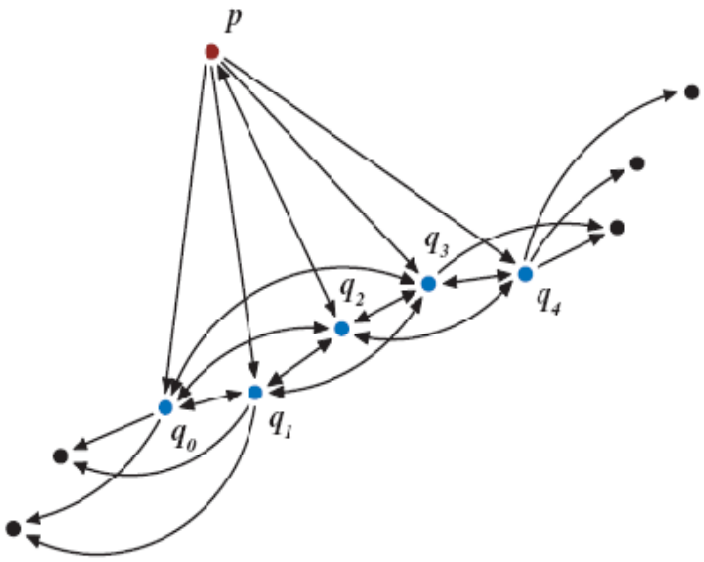


Şekil 4.3.4.1 Mini Top kriteri (Weyrich vd., 2004)

P noktasının belirlenen k kadar en yakın komşuluklarından oluşan bir küre oluşturulur. Bu kriter p noktasının S küresinin karşılaştırılmasıyla oluşur. Komşularının oluşturduğu r çapındaki kürenin merkezinden d kadar uzaklıkta olan noktanın durumu incelenir. Küre dışında kalan nokta gürültü olarak adlandırılır (Weyrich vd., 2004).

$$\chi_{\mathrm{mb}}(p) = \frac{d}{d + 2r/\sqrt{k}}. \tag{4.4}$$

4.3.5 En Yakın Komşu Karşılığı



Şekil 4.3.5.1 En yakın komşu karşılığı (Weyrich vd., 2004)

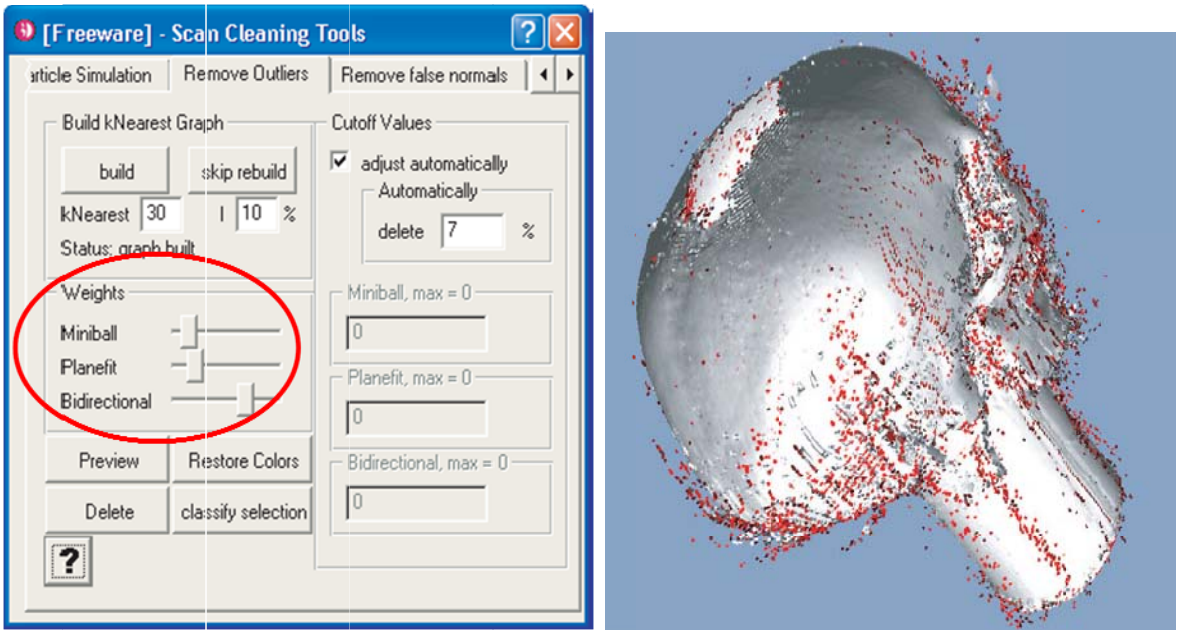
Potansiyel olarak görünen gürültülü noktanın en yakın komşularından uzakta olması bu kriterin en belirgin özelliğidir. Geçerli noktaları q diye adlandırırsak bu durumda uyuşumsuz nokta q komşularının bir parçası ilişkisi nerdeyse hiç olmayan olarak belirlenir (Weyrich vd., 2004).

Nuni(***p***) ={***q*** | ***q*** 2 N***p***, ***p*** /2 N***q***} Nbi(***p***) = {***q*** | ***q*** 2 N***p***, ***p*** 2 N***q***}

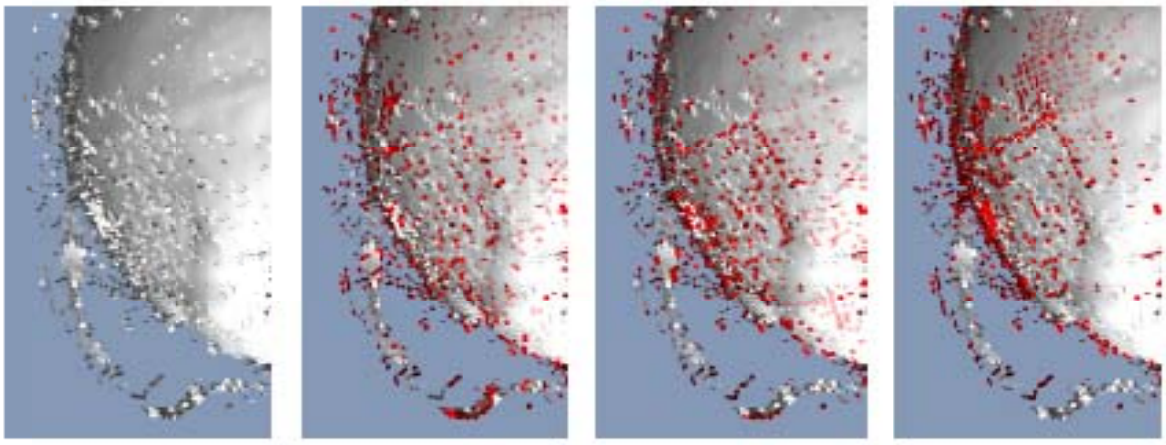
$$\chi_{\mathrm{bi}}(\boldsymbol{p}) = \frac{||\mathcal{N}_{\mathrm{uni}}(\boldsymbol{p})||}{||\mathcal{N}_{\mathrm{bi}}(\boldsymbol{p})|| + ||\mathcal{N}_{\mathrm{uni}}(\boldsymbol{p})||} = \frac{||\mathcal{N}_{\mathrm{uni}}(\boldsymbol{p})||}{k} \tag{4.5}$$

Uyuşumsuz yani gürültülü noktalar tek yönlü ilişki (Nbi) sayısının fazla ilişkisi olan (Nuni) ya da tam tersi en az karşlıklı ilişkisi olan noktalar olarak belirlenir (4.5) (Weyrich vd., 2004).

Konu üzerine Stanford üniversitesi bir araç geliştirmiş olup noktaların belirlenmesi için kümelendirme metotlarından üç yöntemi ele alınmıştır. Oluşturulan sınıflar sonrasında uygulama treshold değeri ise uyuşumsuz noktaları belirlemiştir (Şekil 4.3.5.2) (Weyrich vd., 2004).



Şekil 4.3.5.2 Stanford Ünv. tarafından geliştirilen yazılım (Weyrich vd.,2004)



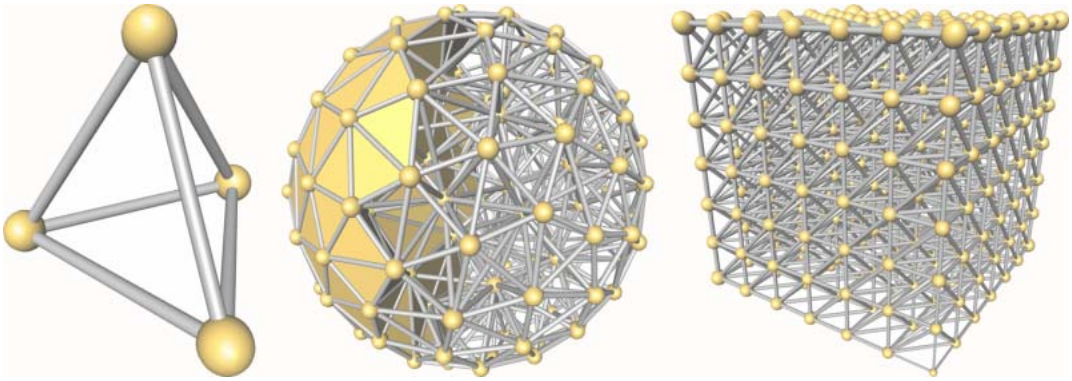
(a) Nokta Bulutu (b) Mini Top (c) Düzleme oturma (d) En Yakın Komşuluk

Şekil 4.3.5.3 Uygulanan kriterler sonucunda belirlenen gürültüler (Weyrich vd., 2004)

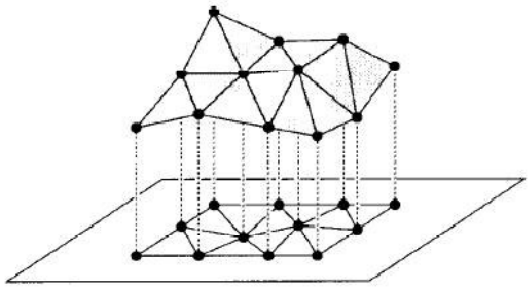
5. ÜÇGENLEME

Konuma bağlı bilginin (yükseklik, jeoit ondülasyonu, gravite değeri vb.) üretiminin ve tüketiminin artması, konumsal bilginin modellenmesini ve gerektiğinde enterpolasyonla ara değer üretilmesini gerekli kılmıştır. Gelişen bilgisayar olanakları bu ihtiyacı daha da kolay karşılanır hale getirmiştir. Fiziksel yeryüzü gibi düzgün olmayan yüzeylerin matematiksel olarak ifadesinde zorluklar vardır. Tam olarak ifade edilebilmesi için yüzeydeki tüm noktaların tanımlı olması gerekir ki bu da pratik olarak mümkün değildir (Yanalak, 2001).

Yüzey tek bir fonksiyonla bütün olarak ifade edilmesiyle yapılabileceği gibi geometrik şekillere bölünerek parça parça ifade edilmesiyle de yapılabilmektedir. Özellikle düzensiz dağılım gösteren noktalara bağlı yüzey modellemesinde, noktaların işlenerek üçgenler ağı oluşturulması (üçgenleme), enterpolasyon işlemi gibi önemli olan konularda sıkça kullanılan bir çözüm yöntemidir



Şekil 5.1 Üç boyutlu üçgenleme (Pion ve Teillaud)



Şekil 5.2 İki boyutlu üçgenleme (Berg ve digerleri, 1997)

Yüzeyi oluşturan üçgenlerin köşe noktaları dayanak noktalarıdır ve her bir dayanak noktası en az bir üçgenin köşe noktasını oluşturur. Üçgenlemenin amacı dayanak noktalarını ilişkilendirmektir. İki, üç ve daha büyük boyutlu uzaylarda gerçekleştirilebilir (Şekil 5.1- 5.2).

5.1 Üçgenleme Yöntemlerinin Sınıflandırılması

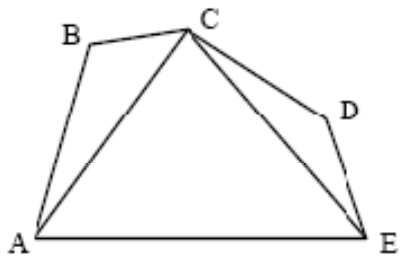
Üçgenleme algoritmaları, hedeflenen amaç ve kullanılan çözüm yöntemi esas alınarak iki farklı şekilde sınıflandırılabilir (Yanalak, 2001) .

5.1.1 Amaca Göre Sınıflandırma

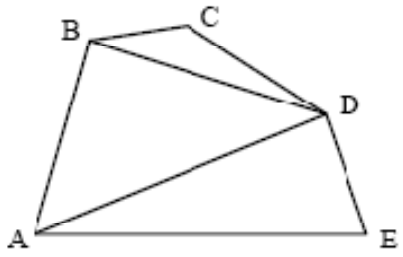
Üçgenlemede sıklıkla kullanılan amaçlar şunlardır(Yanalak, 2001):

- Oluşan üçgenlerin eşkenar üçgenlere en yakın üçgenler olması,
- Oluşan üçgenler ağının kenarları toplamının minimum olması,
- Her bir üçgen oluşturulurken olası kenarlardan en kısa olanının seçilmesidir.

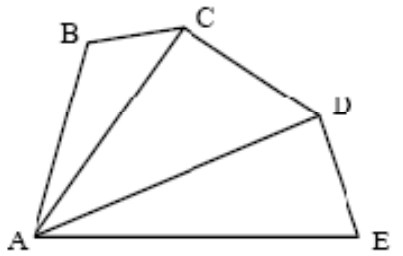
Amaca bağlı olarak, oluşacak üçgenler ağı da farklı olur. Söz konusu amaçlar ve aralarındaki fark basit bir üçgenleme örneği üzerinde kolaylıkla anlaşılabilir (Yanalak, 2001). Şekil 5.1.1.1'de dayanak noktası ve bunların 3 ayrı amaca uygun olarak üçgenlenmiş hali görülmektedir. Her üç üçgenlemede de veri alanını sınırlayan AB, BC, CD, DE ve EA kenarları ortak olarak bulunmaktadır. Kenarlar toplamını minimum yapan üçgenleme AC ve CE kenarlarını kullanmaktadır. Her bir üçgeni oluştururken olası kenarlardan en kısa olan seçilerek yapılan üçgenlemede (Greedy üçgenlemesi), önce BD kenarı, sonra da geriye kalan ve BD' yi kesmeyen AD ve BE kenarlarından, daha kısa olan AD kenarı kullanılmaktadır. "Eşaçılılık" özelliği diye adlandırılan özelliği gerçekleştirmek için, bu özelliği taşıdığı kanıtlanmış olan Delaunay üçgenlemesi kullanılmıştır. Oluşan üçgenler en olası eşkenar üçgenlerdir (eşaçılık özelliği). Birbirlerine uzak olan ve direkt ilişkisi bulunmayan noktalar arasında doğrusal bir ilişki kurulması engelleyen, çevrel çember kriteri bulunduran bir üçgenlemedir. Delaunay üçgenlemesi uygulandığında, AC ve AD kenarları kullanılır. Çünkü üçgenlemenin özelliği gereği, oluşan üçgenlerin çevrel çemberi içerisinde başka dayanak noktası yer almamaktadır (Yanalak,2001).



A(0.0, 0.0) AC =6.20
B(1.4, 4.5) CE =6.45
C(3.8, 4.9)
D(7.15, 3.1)
E(8.0, 0.0)



A(0.0, 0.0) BD =5.92
B(1.4, 4.5) AD =7.79
C(3.8, 4.9)
D(7.15, 3.1)
E(8.0, 0.0)



A(0.0, 0.0) AC =6.20
B(1.4, 4.5) AD =7.79
C(3.8, 4.9)
D(7.15, 3.1)
E(8.0, 0.0)

Şekil 5.1.1.1 Üç Farklı Üçgenleme Örneği (Optimal, Greedy ve Delaunay) (Yanalak,2001)

5.1.2 Çözüm Yöntemine Göre Sınıflandırma

Üçgenleme algoritmaları, dayandıkları çözüm yöntemlerine göre iki genel gruba ayrılabilirler. Bunlardan birincisi, artan yöntemler, ikincisi ise bölüp-birleştiren yöntemlerdir (Yanalak,2001). Artan yöntemler, veri alanının içindeki veya sınırındaki bir dayanak noktasından başlayıp adım adım diğer noktaları ağa katarak üçgenleme işlemini gerçekleştirir. Bölüp-birleştiren yöntemler, son üçgenleme oluşana kadar veri alanını ardışık olarak alt bölgelere ayıran yöntemlerdir(Yanalak,2001).

Bu ayırımın yanı sıra yöntemler, direkt ve iteratif yöntemler olarak da sınıflandırılabilir. Direkt yöntemler üçgenleme işlemini bir kerede tamamlayan algoritmalardır. İteratif yöntemlerde ise ilk olarak keyfi bir üçgenleme oluşturulur, daha sonra seçilen bir optimizasyon kriteri kullanılarak, hiçbir üçgen kenarı değişmeyene kadar üçgenleme iyileştirilir (Yanalak,2001).

Bazı algoritmalarda ise araya sokma yöntemi kullanılır. İlk işlem olarak sadece veri alanını sınırlayan dayanak noktaları kullanılarak üçgenleme işlemi yapılır. İçeride kalan dayanak noktaları sonradan teker teker üçgenlemeye dâhil edilir. Araya sokulan nokta, içine düştüğü üçgenin köşe noktalarıyla birleştirildikten sonra optimizasyon kriteri uygulanır. Değişecek kenarlar ve üçgenler yenilenir. Araya sokma işlemini gerçekleştirmenin ikinci bir yolu da şöyledir: Araya sokulan noktayı hangi üçgenlerin çevrel çemberlerinin içerdiği belirlenir. Bu üçgenlerin kenarları bir listeye yazılır. Listede iki kez geçen kenarlar ve bu kenarlara ait üçgenler iptal edilir. Bu silinen üçgenler yerine, araya sokulan nokta ile listede kalan kenarlar birleştirilerek elde edilecek üçgenler dâhil edilir (Yanalak,2001).

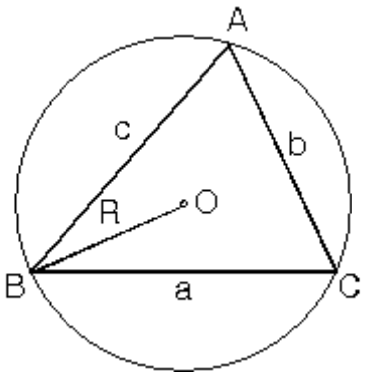
Günümüzde arazi modellemede sıklıkla kullanılan Delaunay üçgenlemesi bu durumda amaca yönelik bir algoritmadır. Çevrel çember kriteri kullanarak arazi modellemede düzensiz dağılım gösteren noktaları Delaunay yöntemiyle üçgenleyen Doğu Avusturalya Üniversitesinden araştırmacı Poul Bourke ‘ün algoritması bir sonraki bölümde detaylı olarak anlatılmıştır.

5.2 Poul Bourke Üçgenleme Algoritması

Yüzey enterpolasyonu için birçok teknik mevcuttur. Bunlardan bazıları komşuluk ilişkili enterpolasyon, 2.dereceden yüzeyler, polinomal enterpolasyon, spline enterpolasyon ve Delaunay üçgenlemesidir. Delaunay üçgenlemesi Voronoi çözümlemesinde olduğu gibi Dirichlet problemiyle yakından ilişkilidir. Dirichlet problemi matematikte, bir integrali kapalı bir çevredeki değerlerle belirleyen problem olarak bilinir. Çözüm, çevredeki değerlerce bulunur (Bourke,1989).

Bu algoritmada kenar ve üçgen sayısı, var olan nokta sayısınca belirlenir. Bunlar kullanıcıdan bağımsız işleyen değerlerdir. Yöntem bir süper üçgen üretmek ile tüm noktaları çevreleyen bir yapay üçgen ile başlamaktadır.

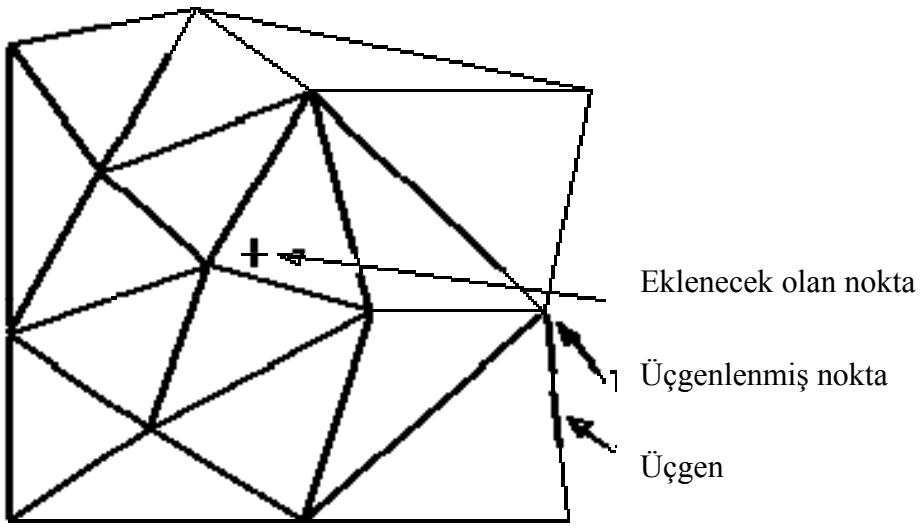
Çevrel çember, düzgün çokgenlerin tüm köşelerinden geçen çemberdir (Şekil 5.2.1). “Doğrusal olmayan üç noktadan bir çember geçebildiği için, her üçgenin çevrel çemberi vardır” şartından dolayı istenen noktanın üçgenlerin çevrel çemberlerinin içinde olup olmadığı kontrol edilir.



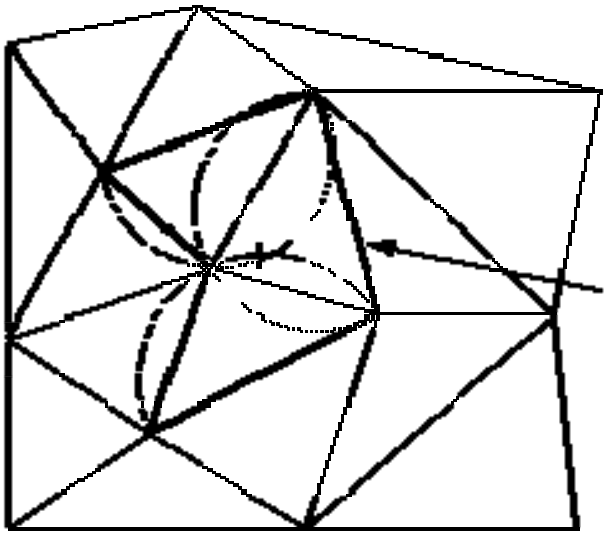
Şekil 5.2.1 Çevrel çember

Her bir üçgenin 3. noktası belirlenirken bu üç noktanın çevrel çemberi içinde bir başka dayanak noktası olup olmadığı kontrol edilir. Çember içerisinde bir başka noktanın olduğu belirlenirse, belirlenen bu nokta yeni oluşacak üçgenin 3. köşesi olmaya aday olur. Aday üçgenin çevrel çemberi içerisinde bir başka nokta olup olmadığı da kontrol edilir. Bu işlem çevrel çember içerisinde nokta kalmayana kadar devam eder ve üçgen oluşur.

Algoritma, tüm noktaları çevreleyen bir yapay üçgen ile başlar. Sonunda ise bu süper üçgenle ortak kenarları olan üçgenler üretilir. Üçgenlemeye giren her nokta bulutu elemanı Şekil 5.2.2 Şekil 5.2.3 ve Şekil 5.2.4 de yer alan işleme tabi tutulur.

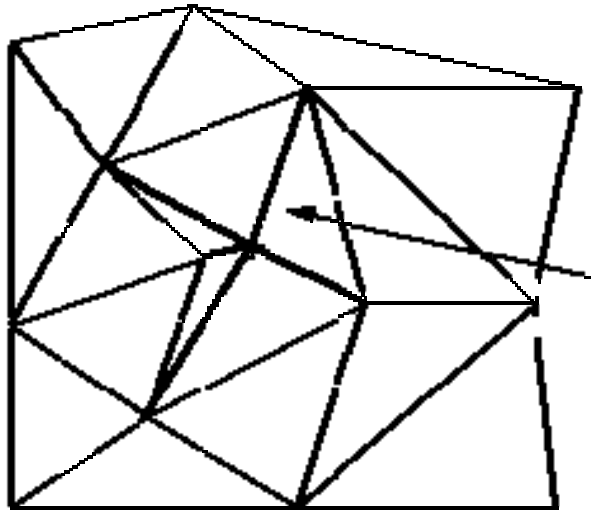


Şekil 5.2.2 Üçgen ağa yeni bir noktanın eklenmesi (Bourke,1989)



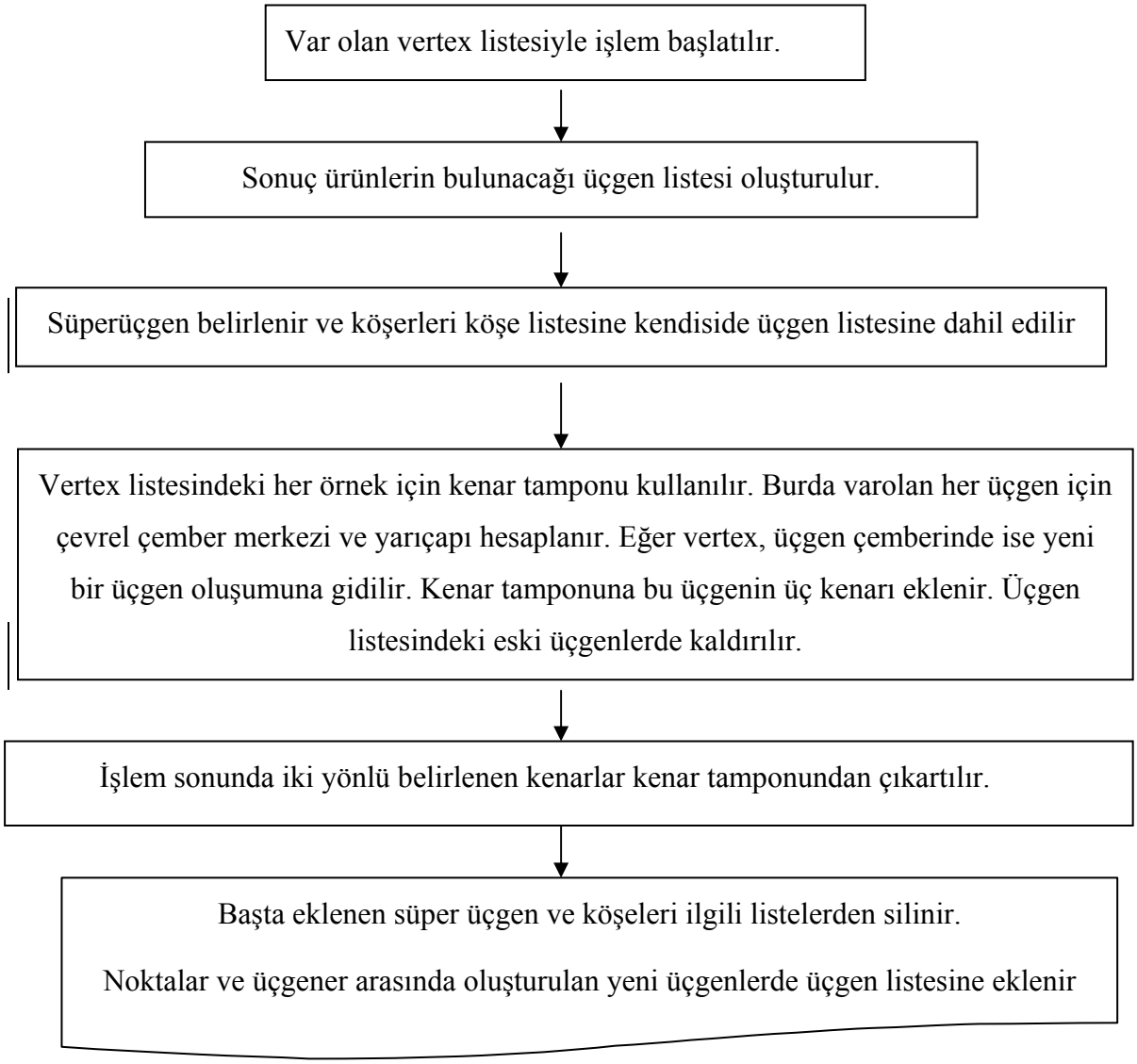
Şekil 5.2.3 Çevrel çember kriteri ile nokta üçgen ağda yerini oluşturur (Bourke,1989)

Yeni eklenen noktayı çevreleyen poligondaki tüm üçgenler silinir. Eklenen nokta ve çevreleyen poligon kenarlarıyla yeni üçgenler oluşturulur.



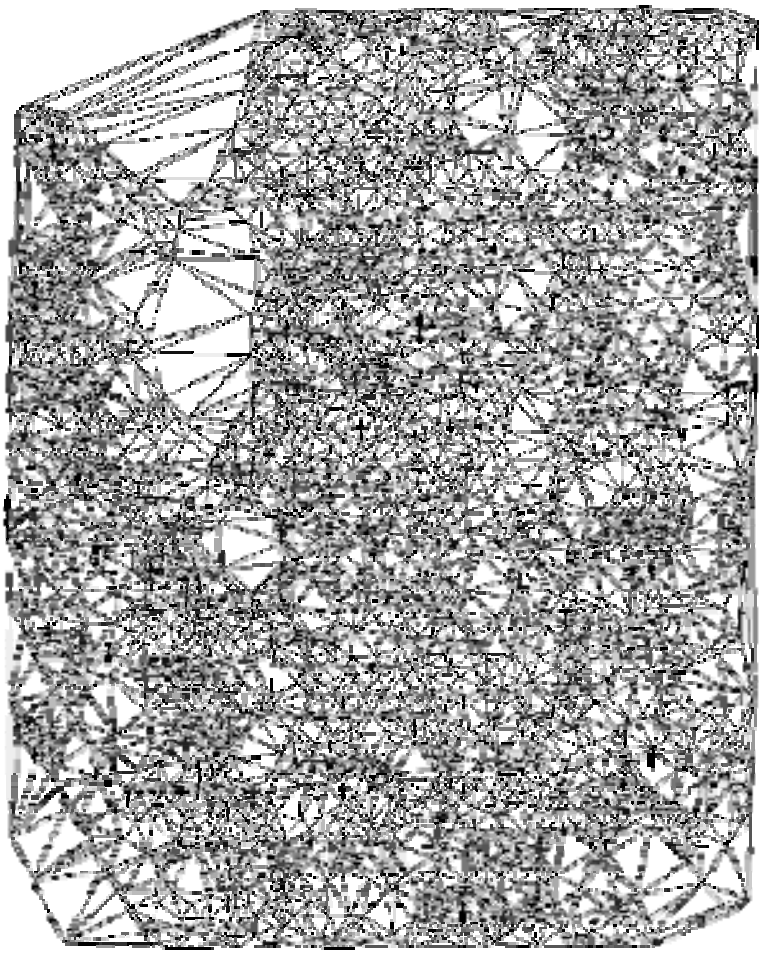
Şekil 5.2.4 Her yeni noktada yinelenen üçgenleme (Bourke,1989)

Üçgenleme algoritması aşağıdaki gibidir (Bourke, 1989).

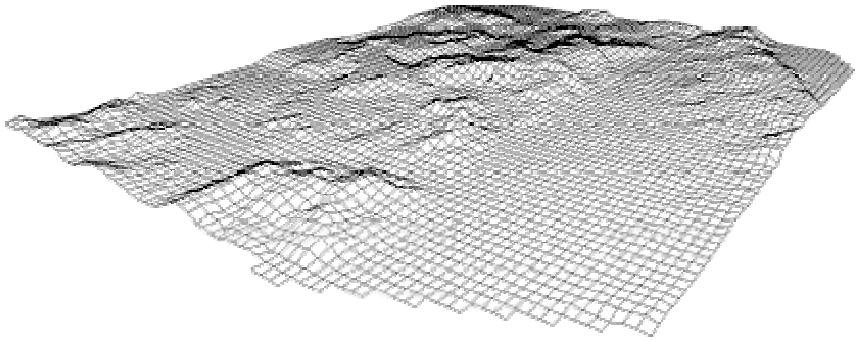


Şekil 5.2.5 Poul Bourke üçgenleme algoritması

Algoritmanın C# dilinde yazılmış kodları Ek1'de bulunmaktadır. Birbirlerine uzak olan ve direkt ilişkisi bulunmayan noktalar arasında doğrusal bir ilişki kurulması engelleyen, var olan nokta listesince üçgenleme yapılmasına izin veren, iteratif özellikli yani seçilen bir optimizasyon kriteri kullanarak hiç bir üçgen kenarı değişmeyene kadar üçgenlemeyi iyileştiren bir yöntem olduğu için de bu tez çalışmasında kullanılmaktadır.



Şekil 5.2.6 Üçgenlenmiş veri (Bourke, 1989)



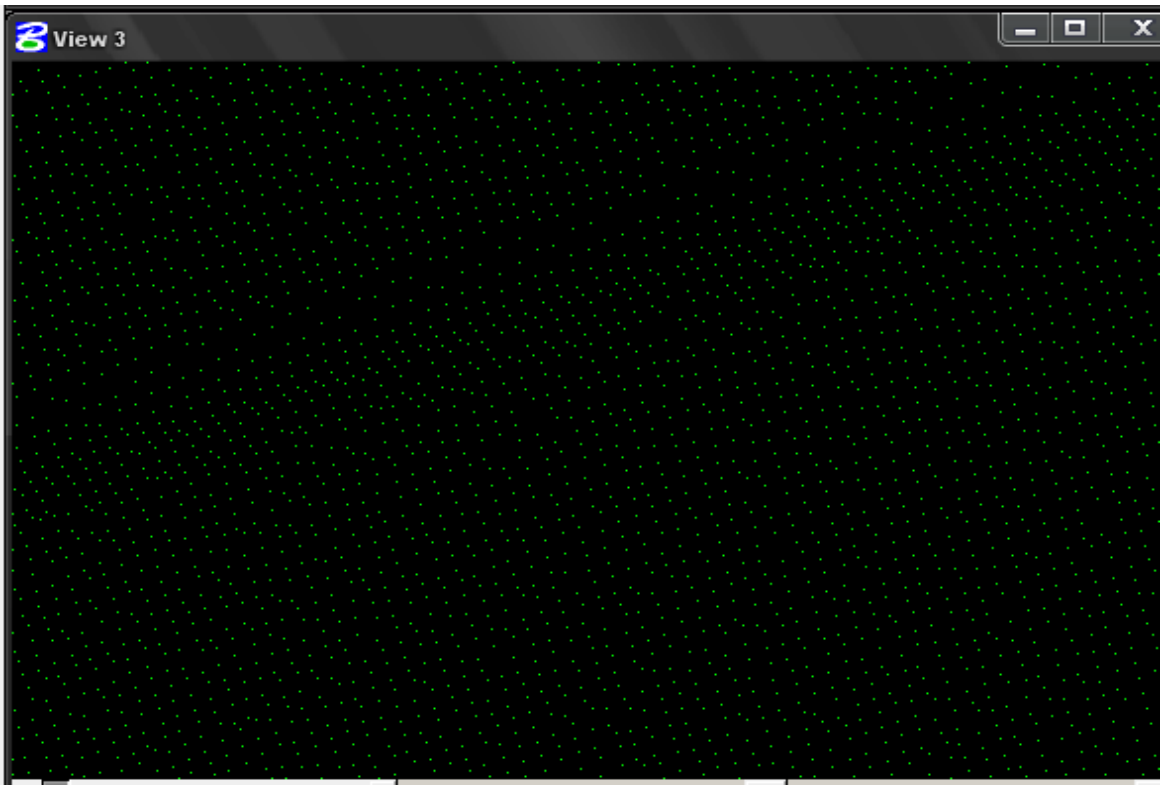
Şekil 5.2.7 Üçgenlenmiş mesh ile belirlenmiş grid yüzey (Bourke, 1989)

6. POUL BOURKE ÜÇGENLEME YÖNTEMİ VE NORMAL VEKTÖRÜ YARDIMIYLA GELİŞTİRİLEN GÜRÜLTÜ ELİMİNASYONU ALGORİTMASI

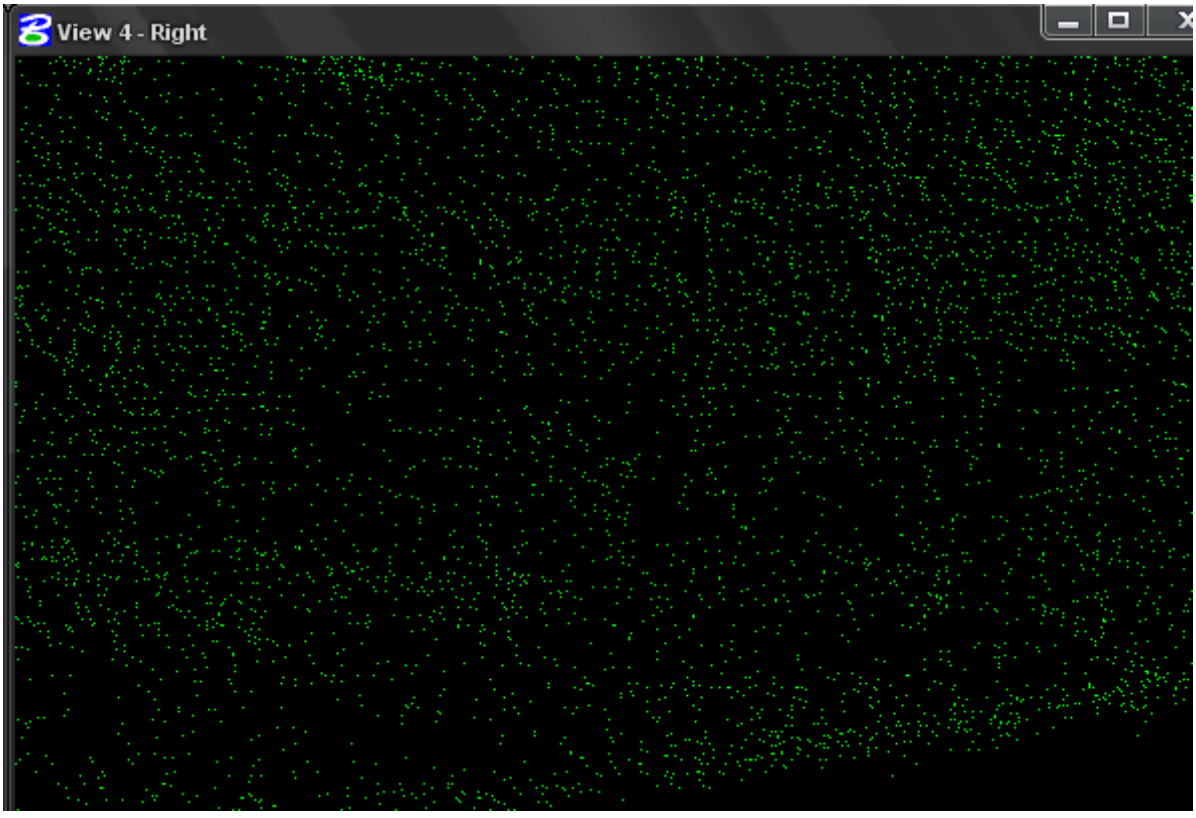
"Teorini mümkün olduğu kadar basit tut, ama daha basit değil." diyen Einstein, bizlere evrenin işleyişinin basit olduğunu ancak bunun her zaman anladığımız biçimde basit olmadığını vurgulamıştır. Bilim, doğa ve teknoloji insan beynin algıladığı kadar basit, anlayamadığı kadar karmaşık olgulardan ibarettir. Karmaşıklık, temel doğrularla düzene girer ve nedenleri belirlenen sorun, basit işlemlerle çözümüne ulaşır. Her şey özünde basitlik içerir detaylar çoğaldıkça da karmaşıklıklar başlar. Temel doğrular ise bütün bunlar hakkında fikir yürütülmesini sağlarlar.

Kusursuz koşullar altında (yansıma ışımanın olmadığı, pürüzsüz yüzeye sahip vb.) taranan bir objeden düzenli bir veri elde edilir. Bu obje tarayıcı tarafından belli aralıklarca atılan noktalar sayesinde gerçeğe yakın, gerçek modele ulaştıracak nokta bulutuna sahip olur. Bulundurdukları gürültüler belli bir sistematik içinde yer alır. Pürüzsüz ve yansıma yapmayan bir yüzeye sahip, multipath (çoklu yansıma) etkisi yaratmayan koşullar altında taranan objeden elde edilen veri ancak tarayıcının yanlış konumlanmış olmasından vb. nedenlerden kaynaklanan hatalar içerir. Bunun dışında taranarak elde edilen her lazer verisi kendi içinde düzensizlikler barındırır.

Yansıma, ışıma, yüzey pürüzlülüğü vb. etkenler nedeniyle gerçekte objeye ait olmayan birçok veri nokta bulutunda yer alır. Detaylar birbirine karışır. Fazla olduğu kadar eksik noktalarda yer alır. Yansımalardan ötürü beklenmedik yerlerde noktalar oluşur. İşte tüm bunları bulunduran lazer tarama berileri düzensiz olarak adlandırılır. Özelliklerinden ötürü bahsi geçen düzenli ve düzensiz nokta bulutunun veri dağılımı Şekil 6.1 ve Şekil 6.2de gösterilmiştir.



Şekil 6.1 Düzenli dağılmış nokta bulutu



Şekil 6.2 Düzensiz dağılmış nokta bulutu

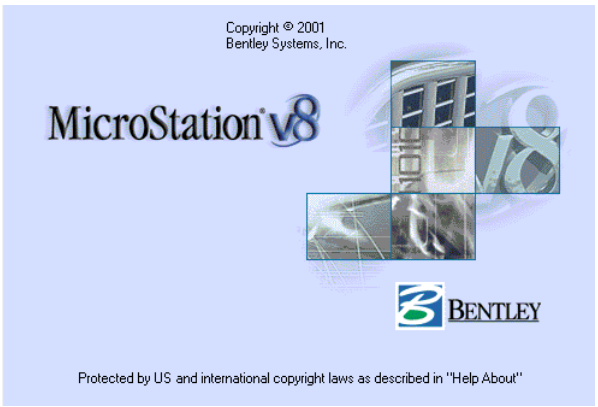
Elde edilen verinin düzensiz oluşu düzene girmeyeceği, sahip olduğu uyuşumsuz noktaların belirlenemeyeceği anlamına gelmez. Bunun için yapılacak olan çalışmada veri mutlaka analizlenmelidir. Objenin şekli önemli olduğu kadar, özellikleri ve taramanın ayrıntıları da yapılacak analizde önem taşımaktadır. Analizler ya ortam koşullarını dikkate alarak ya da var olan veri üzerinden incelemeler yaparak çözüme ulaştırır.

Uyuşumsuz diğer adıyla gürültüleri belirleyen sistemler manuel, yarı otomatik ve otomatik olarak sınıflandırmak mümkündür (Weyrich vd.,2004). Manuel sistemler gözle ayırt edilebilinen kaba hataları belirleyen sistemlerdir. Yarı otomatik ve otomatik sistemler ise var olan gerçek obje ile bağdaşmayan verilerin dışarıdan değerler girilerek yada girilmesine gerek duymadan gözle ayırt edilmesi güç hataları hassasiyet ile tespit eden sistemlerdir (Weyrich vd.,2004). Bu tip sistemler Bölüm 4.3te bahsi geçen çeşitli ölçütlerce geliştirilmiştir. Gürültüler belirlenen ölçütler ile analiz edilerek tespit edilir. Bir başka değişle gürültü, ölçütü sağlamayan nokta bulutu örnekleridir.

Bu bağlamda tez konusu olan gürültüyü elimine eden algoritma basit aksiyomlardan yola çıkarak geliştirilen özgün bir ölçütten oluşur. “En fazla iki tanesi bir doğru üzerinde olan üç noktadan bir düzlem oluşur” aksiyomunun “Aynı yönlü normal vektörüne sahip küçük düzlemler aynı evrensel düzlemin elemanıdır” aksiyomu ile birleşmesi, üçgenleme yapılan nokta bulutunun düzlemlerine ayrılabileceği, aynı düzlemde bulunan nokta bulutu elemanlarının ağırlık merkezinin gürültü elimine etmek için bir ölçüt oluşturabileceği fikrini oluşturmuştur. Bu fikir ile geliştirilen algoritma aşağıdaki işlem akışından ibarettir (Şekil 6.3). İşlem adımları ilerleyen bölümlerde birebir açıklanmış ve örneklendirilmiştir. Örnekleri görselleştirmede yardımcı bir takım programlar kullanılmıştır. Var olan nokta bulutunun, belirlenen gürültülerin ve daha sonra oluşabilecek yüzeyi görebilmek adına Cad programı olan Microstation ve GeoMAGİC modelleme programından yardım alınmıştır (Şekil 6.4).

1	**NOKTA BULUTUNUN ELDE EDİLMESİ**
2	**ÜÇGENLEME**
3	**VERİLERİ DÜZLEMLERİNE AYIRMA**
4	**EŞİK DEĞERİN SİSTEME GİRİLMESİ**
5	**GÜRÜLTÜNÜN TESPİT EDİLMESİ**
6	**GÜRÜLTÜLERİN NOKTA BULUTUNDAN ELİMİNE EDİLMESİ**

Şekil 6.3 İşlem akışı



Şekil 6.4 Verinin görselleştirilmesinde kullanılan programlar

6.1 Nokta Bulutunun Elde Edilmesi

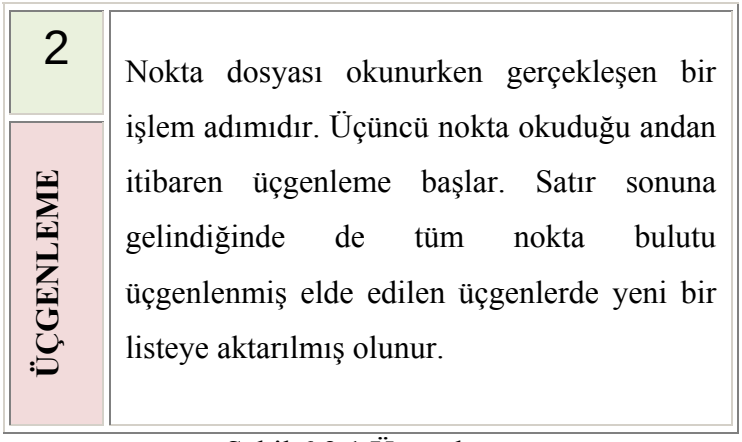
Yersel lazer tarayıcılarından elde edilen her veri yani nokta bulutunda yer alan her noktanın konum belirten x,y,z, yoğunluk belirten RGB (Red-Green-Blue) değerleri bulunur. Geliştirilen algoritma noktanın konum değerlerini dikkate alarak işlem yapmaktadır. İşlem, var olan nokta bulutu dosyasını okuyarak başlar (Şekil 6.1.1). Nokta bulutunun okunabilmesi için de Ek 2 de olduğu gibi ASCII (.txt) formatında olmalıdır.

1	
NOKTA BULUTUNUN ELDE EDİLMESİ	ASCII formatındaki nokta bulutu dosyası açılıp X, Y, Z değerleri satır satır okunur. Okunan her satır oluşturulan Vertex listesinin birer elemanı olarak kaydedilir. Noktaları listeye aktarırken yazım formatına dikkat edilmelidir (nokta, virgül, tab vb.) İşlem sonunda tüm noktalar Vertex listesine alınmış olunur.

Şekil 6.1.1 Nokta bulutu dosyasının okunması

6.2 Üçgenleme

Bölüm 5.2' de tanımlanan çevrel çember kriteri ile üçgenleme yapan Paul Bourke algoritmasına göre içinde nokta bulutu verileri olan Vertex listesi işleme alınır. İşlem sonucunda elde edilen üçgenler bir listede tanımlanır(Şekil 6.2.1). Bu durumda geliştirilen yazılımda hız kazanmak adına noktaları listesine atarken bir yandan da üçgenlemesini yapar. İşlem bitiminde var olan ASCII formatındaki nokta bulutu dosyası okunmuş listeye atılmış, okunurken de üçgenlenmiş ve üçgen listesine eklenmiş olarak hazır halde bulunur. Şekil 6.2.2'de her iki işlem adımının bir arada olduğu kaynak kodlar yer almaktadır.



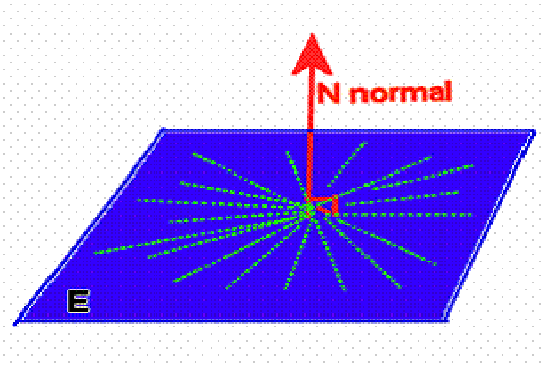
Şekil 6.2.1 Üçgenleme

```
private void Form1_Load(object sender, EventArgs e)
 {
            string[] lineArr;

            openFileDialog1.ShowDialog();
            StreamReader sr;
            char sep;
            char localSep;
            if (Convert.ToDouble("1,1") == 1.1)
            {
                sep = '.';
                localSep = ',';
            }
            else
            {
                sep = ',';
                localSep = '.';
            }

   using (sr = new StreamReader(openFileDialog1.FileName))
   {
    while (!sr.EndOfStream)
       {
          lineArr = sr.ReadLine().Split('\t');

  Vertex[tPoints].x = Convert.ToDouble(lineArr[0].Replace(sep, localSep));
  Vertex[tPoints].y = Convert.ToDouble(lineArr[1].Replace(sep, localSep));
   Vertex[tPoints].z = Convert.ToDouble(lineArr[2].Replace(sep, localSep));
        tPoints++;

        }
if (tPoints > 2)
{
HowMany = Triangulate(ref tPoints);

 }
}
```

Şekil 6.2.2 Nokta bulutu dosyası okunurken üçgenleme yapan kaynak kodlar

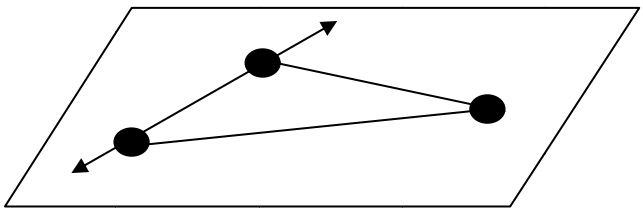
6.3 Verinin Düzlemlere Ayrılması

Düzlem, uzayda bulunan bir doğrunun, yön değiştirmeden ve kendi doğrultusunda olmayan hareketiyle meydana getirdiği kabul edilen yüzeydir (Şekil 6.3.1). Aynı doğru üzerinde olmayan ya da en fazla iki tanesi aynı doğrunun elemanı olan üç nokta bir düzlemi oluşturur (Şekil 6.3.2). Bu aksiyomlardan yola çıkarak üçgenlerin birer düzlem, evrensel düzlemlerinde aynı yönlü üçgenlerin bir araya gelmesinden oluşacağı düşünülmüş ve algoritma bu yönlü geliştirilmiştir(Şekil 6.3.3).

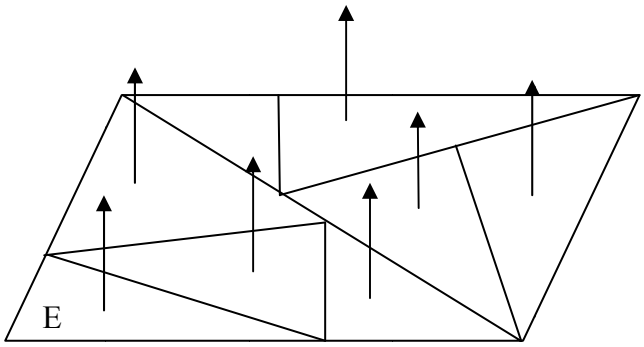


Şekil 6.3.1 Düzlem (Mahir, 1999)

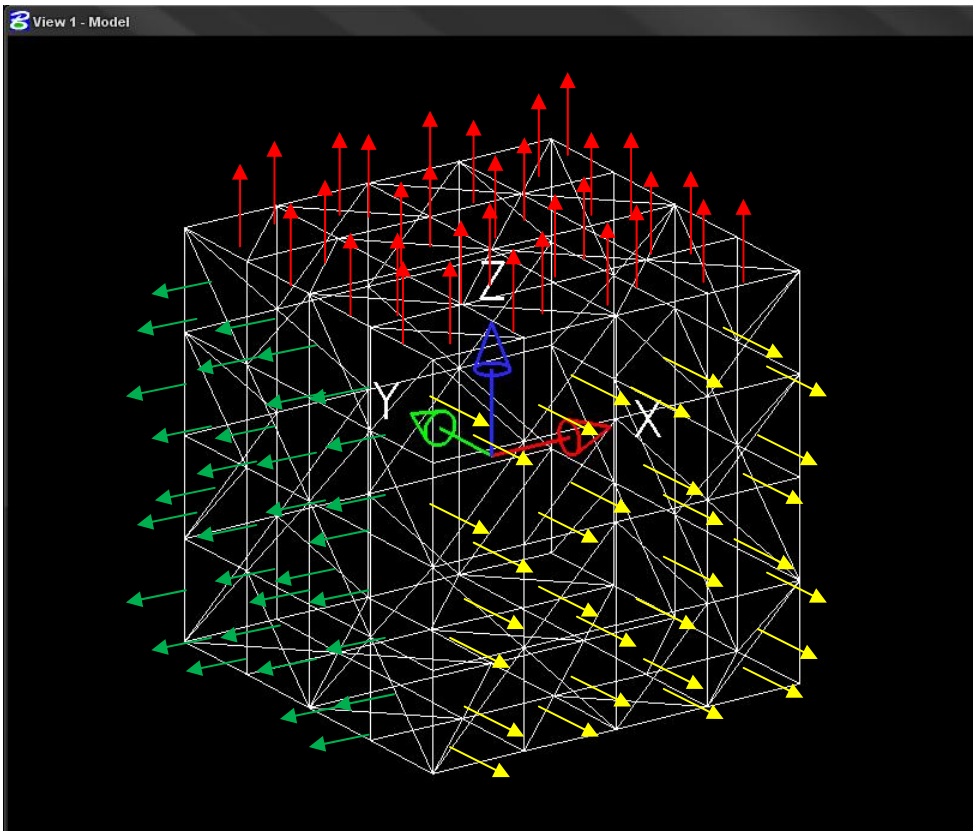
.



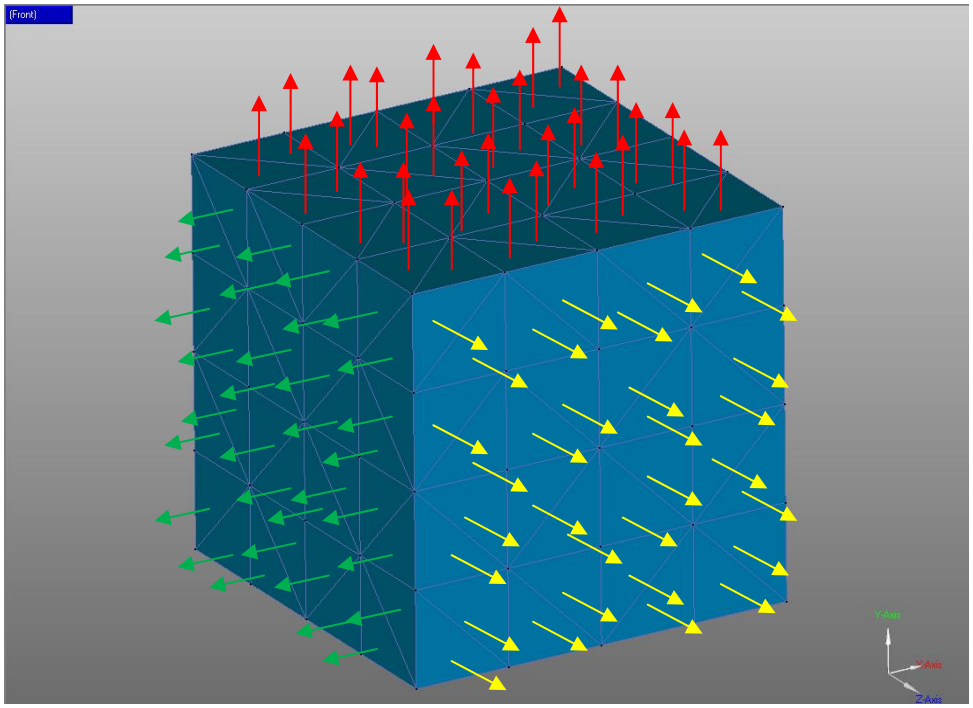
Şekil 6.3.2 En fazla iki tanesi aynı doğrunun elemanı olan üç noktadan oluşan düzlem



Şekil 6.3.3 Evrensel düzleme ait aynı yönlü üçgenler kümesi



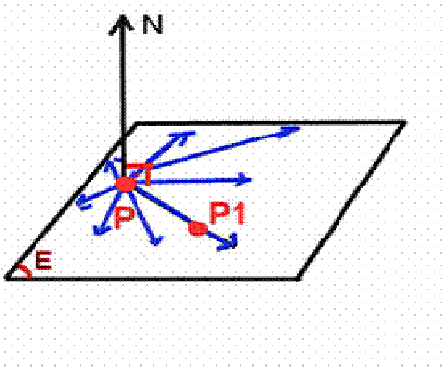
Şekil 6.3.4 Microstationda görselleştirilmiş aynı yönlü üçgenlere sahip küp model



Şekil 6.3.5 GeoMagicte görselleştirilmiş aynı yönlü üçgenlere sahip küp model

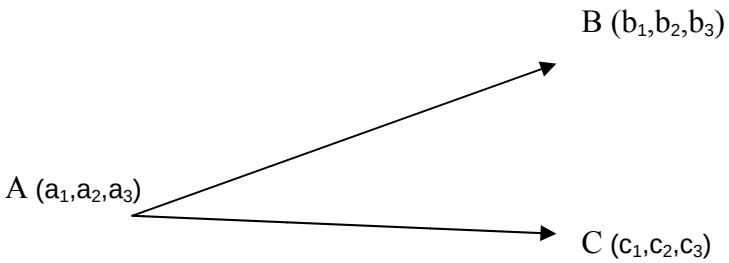
Aynı normal denklemini sağlan üçgenler bir araya geldiklerinde daha anlamlı bir düzlemi oluştururlar (Şekil 6.3.4 - Şekil 6.3.5). Bu anlamlılığı, geliştirilen algoritmada yakalamak adına normal vektörü ve ağırlık nokta hesabından yardım alınmıştır.

6.3.1 Normal Vektör



Şekil 6.3.1.1 Normal Vektörü (Mahir, 1999)

Bir başka adı doğrultu vektörü olarak geçen normal, var olduğu düzleme dik bulunmaktadır. Düzlem normal vektörü ve bir nokta ile tanımlanan bir yapıdır. Bu yapı uzayda da böyle tanımlanır. Uzayda bir A noktası ve bir v vektörü verilsin. Bu durumda normal denklemi (x, y, z) R3 | v. AX = 0'dır.



Şekil 6.3.1.2 Düzlem belirten noktalar (Mahir,1999)

AB ve AC vektörlerini şekildeki gibi oluşturulduğunda uzayda AB x AC vektörü hem AB hem de AC vektörlerine dik bulunur. Bu durumda AB x AC vektörü normal vektör olarak alınabilir. Verilen denklem A noktasından geçen normali ifade eder (Şekil 6.3.1.3).

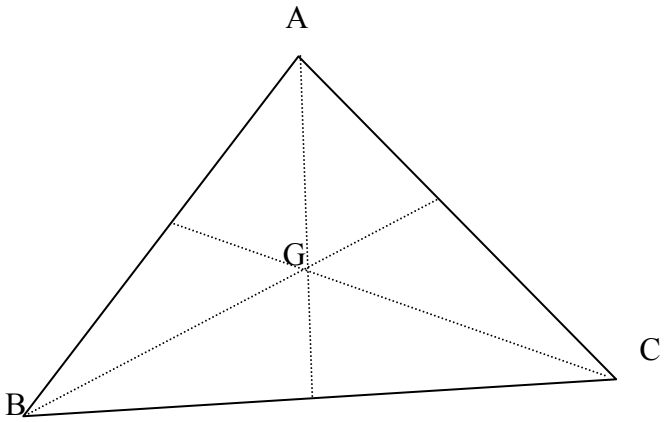
$$\vec{n} = \overrightarrow{AB} \times \overrightarrow{AC} = \begin{vmatrix} e_1 & e_2 & e_3 \\ b_1 - a_1 & b_2 - a_2 & b_3 - a_3 \\ c_1 - a_1 & c_2 - a_2 & c_3 - a_3 \end{vmatrix}$$

$$= \left(\begin{vmatrix} b_2 - a_2 & b_3 - a_3 \\ c_2 - a_2 & c_3 - a_3 \end{vmatrix}, - \begin{vmatrix} b_1 - a_1 & b_3 - a_3 \\ c_1 - a_1 & c_3 - a_3 \end{vmatrix}, \begin{vmatrix} b_1 - a_1 & b_2 - a_2 \\ c_1 - a_1 & c_2 - a_2 \end{vmatrix} \right)$$

Şekil 6.3.1.3 Normal denklemi (Mahir,1999)

6.3.2 Ağırlık Noktası

ABC üçgeninin köşe koordinatları A(x1,y1,z1), B(x2,y2,z2), C(x3,y3,z3) ve ağırlık merkezi G(xG,yG,zG) ise ağırlık merkezi koordinatları:

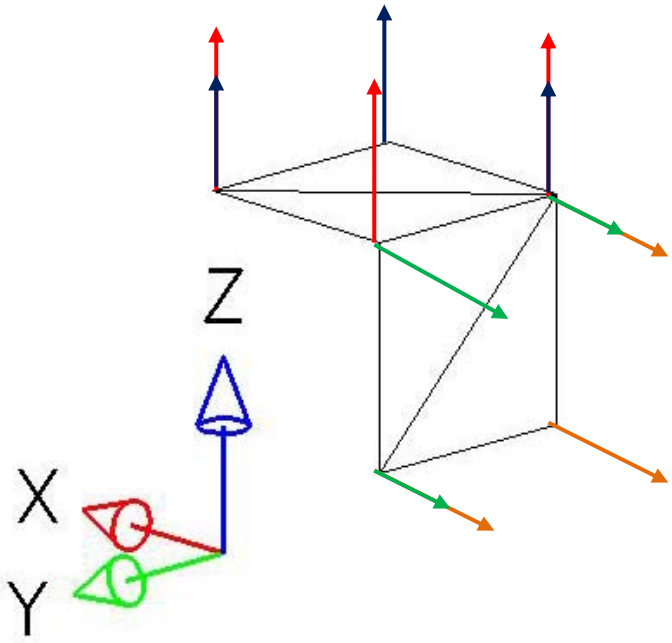


Şekil 6.3.2.1 Ağırlık noktası

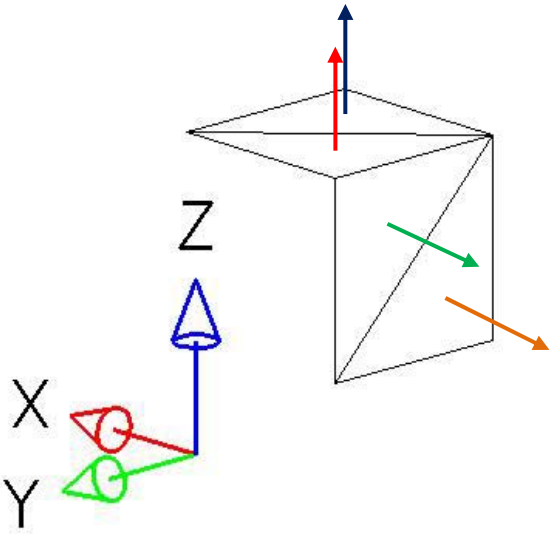
$$xG=(x1+x2+x3)/3 \;,\; yG=(y1+y2+y3)/3,\; zG=(z1+z2+z3)/3 \qquad (6.1)$$

şeklinde hesaplanır.

Ağırlık noktalarından geçen normallerin belirlenmesinin en önemli sebebi oluşabilecek karmaşıklığı engellemek içindir. Şekil 6.3.2.2'de görüldüğü gibi herhangi bir üçgen noktasından noktası olduğu üçgen sayısı kadar normal geçtiğini görmek mümkün. Bunu gidermek için üçgenin anlamlı bir noktasının seçilmesi hem algoritma hem de üçgenin yönelimini fark edebilmesine kolaylık getirmiştir (Şekil 6.3.2.3).

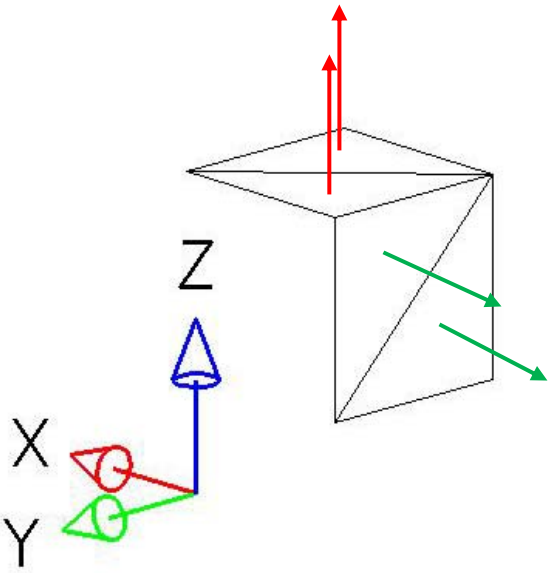


Şekil 6.3.2.2 Bir noktadan birden fazla normal vektörü geçme durumu



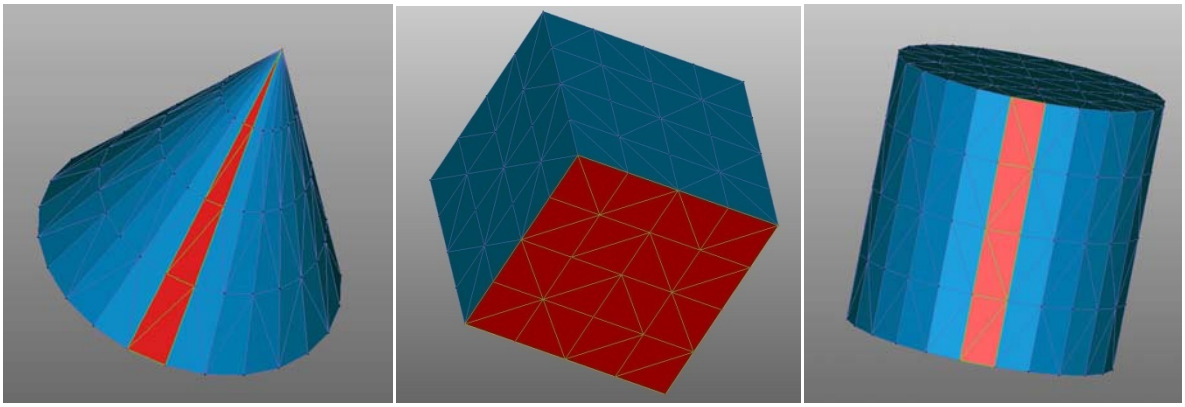
Şekil 6.3.2.3 Ağırlık noktalarından geçen normal vektörü

Aynı normal denklemini sağlayan üçgenler bir düzlemin varlığını işaret ederler (Şekil 6.3.2.4) Geliştirilen algoritma normalleri aynı olan üçgenleri bir araya getirir ve bir sonraki basamak olan gürültü eliminasyonu için ölçüt oluşturur.



Şekil 6.3.2.4 Aynı normal denklemini sağlayan üçgenler

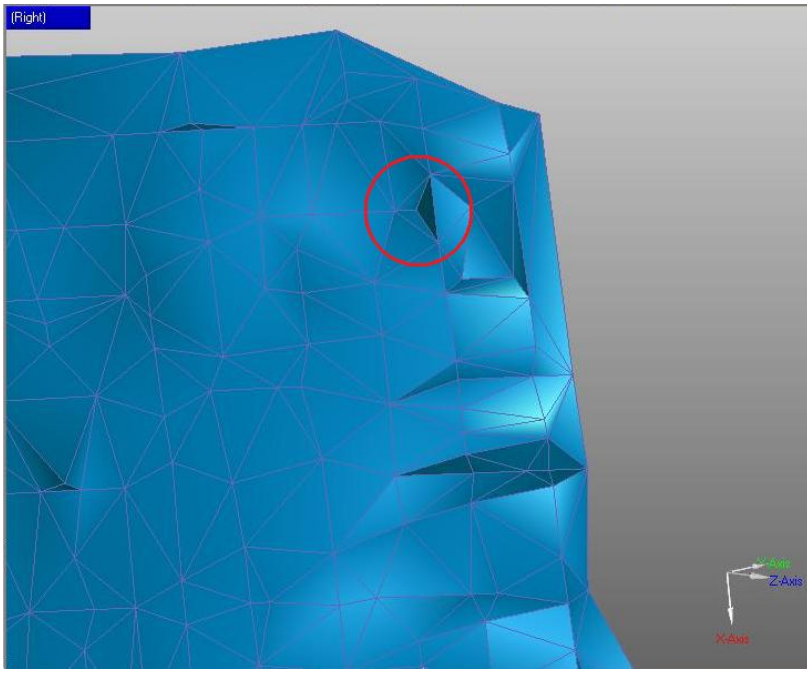
İlkel basit objelerden elde edilen nokta bulutlarında üçgenleme, düzlemlerine ayırma gibi işlemler basit yapıda gelişir. Çünkü kendisi düzenli veri bulundurmaktadır. Aynı yönlü üçgenleri bir arada ve bir düzleme aittir(Şekil 6.3.2.5).



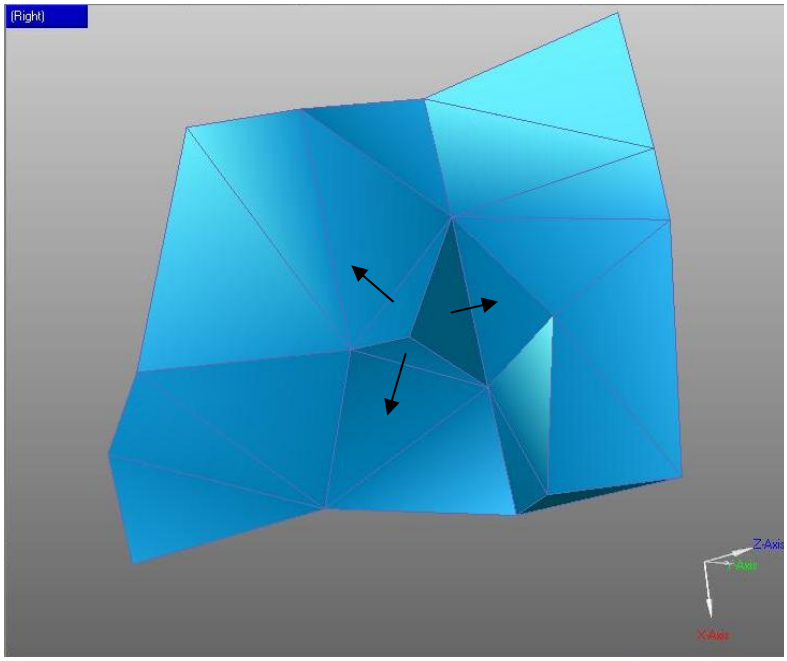
Şekil 6.3.2.5 GeoMagicte görselleştirilmiş ilkel objelerde düzlemler

Düzensiz verilerde durum daha karmaşıktır. Normalleri aynı olan üçgen bulup evrensel bir düzlem oluşturmak düzenli verilerde olduğu kadar kolay olmaz. Düzensiz verileri belli bir düzen içine yerleştirmek ancak verilecek olan ölçüt sayesinde olur. Bu çalışmada sağlanan ölçüt ise normal vektörlerinin aynı yönlü olup bir düzlemi temsil ettiği görüşüdür.

Düzensiz veri, bu bağlamda var olan ölçütü baz aldığında tek üçgenden oluşan düzlemlere de ayrılabilir anlamına gelir. Özellikle de gürültü barındıran verilerde görülür (Şekil 6.3.2.6). Bu durum gürültü belirlemeyi engellemez sadece var olan işlem süresini uzatır.



(a)



(b)

Şekil 6.3.2.6 GeoMagicte görselleştirilmiş üçgen yapısı (a) Düzensiz ve gürültülü verinin üçgen yapısı (b) Gürültü barındıran üçgenlerin normal vektörleri

3	Üçgenleme sonrasında elde edilen listede yer alan her üçgenin ağırlık merkezleri ve bu merkezlerden geçen normal vektörleri belirlenir.
VERİLERİ DÜZLEMLERİNE AYIRMA	İşleme alınacak olan veri sayısının büyük olması nedeniyle algoritmanın zaman ve donanım bakımından zorluk yaşamaması adına normal vektörleri aynı olan yani aynı evrensel düzleme ait olan üçgenler bir sonraki işlem adımında kullanılmak üzere geçici bir listeye aktarılır.

Şekil 6.3.2.7 Verileri düzlemlerine ayırma işlemi

6.4 Eşik Değerin Belirlenmesi

Belirlenecek olan bu değer işleme girdiğinde nokta bulutunda yer alan gürültünün tespitini sağlayacaktır. Bu yüzden belirlenecek olan değer, duyarlılık anlamı taşımalıdır. Göz ardı edilebilecek bir hassasiyet niteliğindeki eşik değer için bir dayanak belirtilmelidir. Çalışmada var olan dayanak, aynı normal denklemini sağladıkları için bir araya gelen üçgenlerin oluşturdukları düzlemin ağırlık merkezidir. Bu merkeze göre belirlenen uzaklık hassasiyeti algoritmanın eşik değerini oluşturur. Bu değer dışarıdan kullanıcı tarafından algoritmaya girilmesi istenir.

4	
EŞİK DEĞERİN SİSTEME GİRİLMESİ	Nokta bulutunda gürültüyü belirleyecek olan eşik değer dışarıdan programa girilir.

Şekil 6.4.1 Eşik değerin girilmesi

6.5 Gürültünün Tespit Edilmesi

Çalışmada girilen eşik değer ve düzlemlerin ağırlık merkezleri sayesinde gürültülü noktalar belirlenir (Şekil 6.5.1). Normal vektörlerinin eşitliği doğrultusunda düzlemlerine ayrılan üçgenler ve onlara ait köşe noktalar, işlem akışına göre bulundukları listenin içinde üç yönde (x,y,z) değerlendirmeye alınırlar. Düzleme ait olan noktalar, ağırlık merkezi ve tanımlanan eşik değer ilişkisi ile gürültü olup olmadıkları araştırılır. Evrensel düzleminde yer alan her nokta için kurulan işlem;

$|\text{Nokta}_{\text{Düzlem}} - \text{Ağırlık merkezi}_{\text{Düzlem}}| > \text{Eşik Değer}$ şeklindedir.

Şayet düzlem içindeki nokta bu mantıksal işlemi doğrulamıyor ise nokta o düzlem için, bir başka değiş ile var olan nokta bulutu için gürültülü veri olarak sayılır.

5	Farklı yönlü her normal vektörü için oluşturulan, içinde aynı evrensel düzlemi temsil eden üçgenler barındıran geçici liste içinde değerlendirmek şartı ile gürültüler belirlenir.
GÜRÜLTÜNÜN TESPİT EDİLMESİ	Aynı evrensel düzleme mensup üçgenlerin köşe noktaları ile düzlemin ağırlık merkezi hesaplanır. Evrensel düzlemin içinde bulunan üçgenlerin köşe noktaları yani başka bir değişle evrensel düzlem içinde yer alan nokta bulutları düzlemin ağırlık merkezi ile karşılaştırılır. Şayet bu karşılaştırma önceden sisteme girilen eşik değer miktarını aşıyorsa değerlendirilen nokta gürültü listesine aktarılır. Nokta $|\text{Nokta}_{\text{Düzlem}} - \text{Ağırlık merkezi}_{\text{Düzlem}}| > \text{Eşik Değer}$ mantıksal işlemi doğrulamıyorsa gürültü listesine aktarılır. İşlem sonunda gürültü listesi ASCII formatında dosyaya aktarılır.

Şekil 6.5.1 Gürültünün tespit edilmesi

Var olan nokta bulutunun düzlemlerine ayrılması, eşik değerinin girilmesi, gürültülerin tespit edilmesi ve gürültü listesinin dosyaya aktarılmasına dair bir kısım kaynak kodlar Şekil 6.5.2 – Şekil 6.5.3 – Şekil 6.5.4 - Şekil 6.5.5'te yer almaktadır.

```
//ağırlık noktasının ve normalin hesaplanması;

            for (int g = 1; g <= HowMany; g++)
            {
Agirliknok[g].x = (Vertex[Triangle[g].vv0].x + Vertex[Triangle[g].vv1].x +
Vertex[Triangle[g].vv2].x) / 3;
Agirliknok[g].y = (Vertex[Triangle[g].vv0].y + Vertex[Triangle[g].vv1].y +
Vertex[Triangle[g].vv2].y) / 3;
Agirliknok[g].z = (Vertex[Triangle[g].vv0].z + Vertex[Triangle[g].vv1].z +
Vertex[Triangle[g].vv2].z) / 3;

            }

            for (int i = 1; i <= HowMany; i++)
            {
normal[i].x = ((Vertex[Triangle[i].vv0].y - Agirliknok[i].y) * (Vertex[Triangle[i].vv1].z -
Agirliknok[i].z)) - ((Vertex[Triangle[i].vv0].z - Agirliknok[i].z) *
(Vertex[Triangle[i].vv1].y - Agirliknok[i].y));


normal[i].y = ((Vertex[Triangle[i].vv0].z - Agirliknok[i].z) * (Vertex[Triangle[i].vv1].x -
Agirliknok[i].x)) - ((Vertex[Triangle[i].vv0].x - Agirliknok[i].x) *
(Vertex[Triangle[i].vv1].z - Agirliknok[i].z));


normal[i].z = ((Vertex[Triangle[i].vv0].x - Agirliknok[i].x) * (Vertex[Triangle[i].vv1].y -
Agirliknok[i].y)) - ((Vertex[Triangle[i].vv0].y - Agirliknok[i].y) *
(Vertex[Triangle[i].vv1].x - Agirliknok[i].x));

             }
```

Şekil 6.5.2 Düzlemlerine ayırma işlemi için gerekli olan ağırlık noktası ve normal vektörü hesabına ait kaynak kod

```
// normalleri aynı olan üçgenlerin vertexlerini kendi içinde tekrar etmicek şekilde toplandığı
adı kova olan bir list tanımlanır. List, düzlemin takendisidir.burda bulunan vertexlerinin
ortalamasını yani düzlemin ağırlık merkezi belirleyecektir.

   List<dVertex> kova = new List<dVertex>();
   List<dVertex> gurultu = new List<dVertex>();
   double tx, ty, tz;
   double ortx, orty, ortz;
   for (int i = 1; i <= HowMany; i++)
     {
     if (i == 1)// giriş yapılır
      {
       kova.Add(Vertex[Triangle[i].vv0]);
       kova.Add(Vertex[Triangle[i].vv1]);
       kova.Add(Vertex[Triangle[i].vv2]);
tx = Vertex[Triangle[i].vv0].x + Vertex[Triangle[i].vv1].x + Vertex[Triangle[i].vv2].x;
ty = Vertex[Triangle[i].vv0].y + Vertex[Triangle[i].vv1].y + Vertex[Triangle[i].vv2].y;
tz = Vertex[Triangle[i].vv0].z + Vertex[Triangle[i].vv1].z + Vertex[Triangle[i].vv2].z;

for (int j = 1; j <= HowMany; j++)//aynı normale sahip diğer ücgenlerin vertexlerini kovaya
atılır
         {if ((normal[i].x == normal[j].x) && (normal[i].y == normal[j].y) && (normal[i].z ==
normal[j].z))
         { int varmi0 = 0;
           int varmi1 = 0;
           int varmi2 = 0;
           for (int k = 0; k < kova.Count; k++)//kovada daha öncesinden o vertex varmı? yokmu?
Kontrolü yapılır şayet yoksa kovaya atılır
 {
 if ((Vertex[Triangle[j].vv0].x == kova[k].x) && (Vertex[Triangle[j].vv0].y == kova[k].y) &&
(Vertex[Triangle[j].vv0].z == kova[k].z))
             {varmi0++;}
if ((Vertex[Triangle[j].vv1].x == kova[k].x) && (Vertex[Triangle[j].vv1].y == kova[k].y) &&
(Vertex[Triangle[j].vv1].z == kova[k].z))
             { varmi1++;}
 if ((Vertex[Triangle[j].vv2].x == kova[k].x) && (Vertex[Triangle[j].vv2].y == kova[k].y) &&
(Vertex[Triangle[j].vv2].z == kova[k].z))
             { varmi2++;}
}
             if (varmi0 == 0)
              {kova.Add(Vertex[Triangle[j].vv0]);
                tx = tx + Vertex[Triangle[j].vv0].x;
                ty = ty + Vertex[Triangle[j].vv0].y;
                tz = tz + Vertex[Triangle[j].vv0].z; }
             if (varmi1 == 0)
                 {kova.Add(Vertex[Triangle[j].vv1]);
                   tx = tx + Vertex[Triangle[j].vv1].x;
                    ty = ty + Vertex[Triangle[j].vv1].y;
                    tz = tz + Vertex[Triangle[j].vv1].z; }
             if (varmi2 == 0)
                   {kova.Add(Vertex[Triangle[j].vv2]);
                     tx = tx + Vertex[Triangle[j].vv2].x;
                     ty = ty + Vertex[Triangle[j].vv2].y;
                     tz = tz + Vertex[Triangle[j].vv2].z; }

}
        }
```

Şekil 6.5.3 Düzlemlerine ayırma işlemine ait kaynak kod

```
//düzlemin ağırlık noktasının hesaplanmasında sıra...ortx,y,z düzlemin ağırlık merkezinin
koordinatlarıdır.
                            ortx = tx / kova.Count;
                            orty = ty / kova.Count;
                            ortz = tz / kova.Count;
//gürültü belirlenirken Ölçü - Gerçek değer= Hata miktarı belirlenmeli..+-değerdedir mutlağı
alınır...kabul edilebilir miktar hata payıda denilebilir! gerçek değerimiz düzleme ait olan
ağırlık noktamızdır..incelik belirtir! Uzaklık hassaiyeti söz konusudur.

    double hatapayı = 200; //girilen eşik değer
    for (int k = 0; k < kova.Count; k++)// nokta 3 akseste incelenir.
    {
     if ((Math.Abs(kova[k].x - ortx) > hatapayı) || (Math.Abs(kova[k].y - orty) > hatapayı) ||
(Math.Abs(kova[k].x - ortx) > hatapayı))
       {
         if (gurultu.Count==0)
           {gurultu.Add(kova[k]); }
         else
         {
          int tekrar = 0;//gurultu kendi içinde tekrar ediyormu?
          for (int f = 0; f < gurultu.Count; f++)
          {if ((kova[k].x == gurultu[f].x) && (kova[k].y == gurultu[f].y) && (kova[k].z ==
gurultu[f].z))
              {tekrar++;}
           }
               if (tekrar == 0)
               {gurultu.Add(kova[k]); }
          }

         }
       }
```

Şekil 6.5.4 Gürültünün tespit edilip listeye alınması işlemine ait kaynak kod

```
StreamWriter sw5 = new StreamWriter("C:\\Users\\USER\\Desktop\\gurultu.txt");

 for (int k = 0; k < gurultu.Count; k++)
 {


  sw5.Write(gurultu[k].x);
  sw5.Write('\t');
  sw5.Write(gurultu[k].y);
  sw5.Write('\t');
  sw5.Write(gurultu[k].z);

  sw5.WriteLine();


  }
  sw5.Close();
```

Şekil 6.5.5 Gürültü listesinin dosyaya aktarılması işlemine ait kaynak kod

6.6 Gürültüden Arınmış Verinin Elde Edilmesi

Başta var olan nokta bulutu verisinden, elde edilen gürültülerin çıkarılması işlemidir (Şekil 6.6.1 - Şekil 6.6.2). Bu noktaların çıkartılması sonucunda temiz ve modellemenin diğer adımlarına hazır bir nokta bulutu elde edinilir. Geliştirilen gürültü eliminasyonu algoritması çıktı ürünü olarak temiz modele ait noktaların koordinatlarını ASCII formatındaki dosyalara aktarır (Şekil 6.6.3).

6	Nokta bulutunun bulunduğu Vertex listesindeki her örneğin gürültü listesinde olup olmadığı kontrol edilir. Listede yer almayan veri model nokta listesine aktarılır. İşlem sonunda elde edilen model listesi gürültüden arındırılmış veri olarak belirlenir. Bu veri işlem sonunda ASCII formatında bir dosyaya yazdırılır.
GÜRÜLTÜLERİN NOKTA BULUTUNDAN ELİMİNE EDİLMESİ	

Şekil 6.6.1 Gürültülerin nokta bulutundan elimine edilmesi

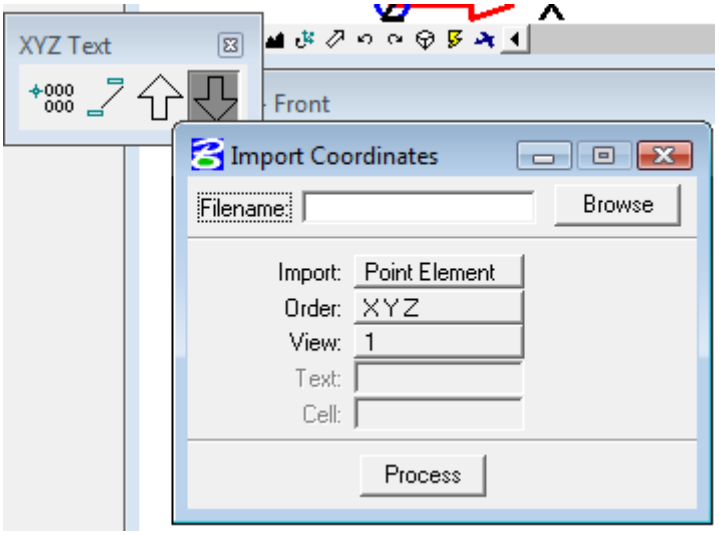
```
//temizlenmiş modele ulaşılmalı!

  List<dVertex> modelnokta = new List<dVertex>();

   for (int m = 0; m < tPoints; m++)
   {
     int varmi=0;
     for (int n =0; n < gurultu.Count;n++)
      {if
((Vertex[m].x==gurultu[n].x)&&(Vertex[m].y==gurultu[n].y)&&(Vertex[m].z==gurultu[n].z))
        {varmi++;}
       }
   if(varmi==0)
        {modelnokta.Add(Vertex[m]); }

   }
```

Şekil 6.6.2 Gürültülerin nokta bulutundan elimine edilmesi işlemine dair kaynak kod

```
StreamWriter sw7 = new StreamWriter("C:\\Users\\USER\\Desktop\\temizmodel.txt");

      for (int k = 0; k < modelnokta.Count; k++)
       {
        sw7.Write(modelnokta[k].x);
        sw7.Write('\t');
        sw7.Write(modelnokta[k].y);
        sw7.Write('\t');
        sw7.Write(modelnokta[k].z);

         sw7.WriteLine();


        }
       sw7.Close();
```

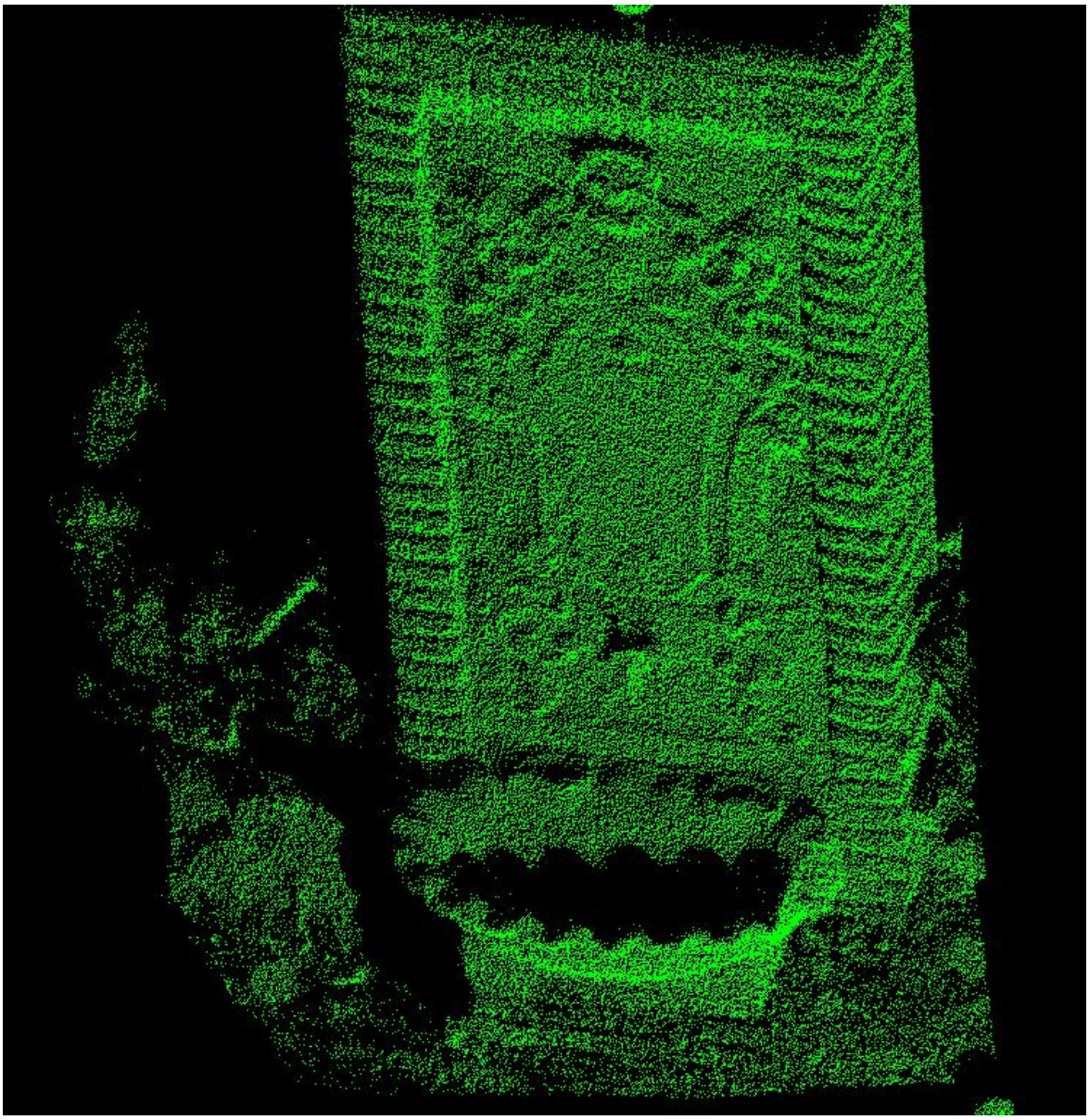
Şekil 6.6.3 Temiz modele ait noktaların koordinatlarının dosyaya yazdırılması işlemi

Çıktı ürün olarak alınan ASCII formatındaki gürültü ve temiz modele ait nokta listelerini görselleştirmek için Microstation programında yaralanılır (Şekil 6.6.4).



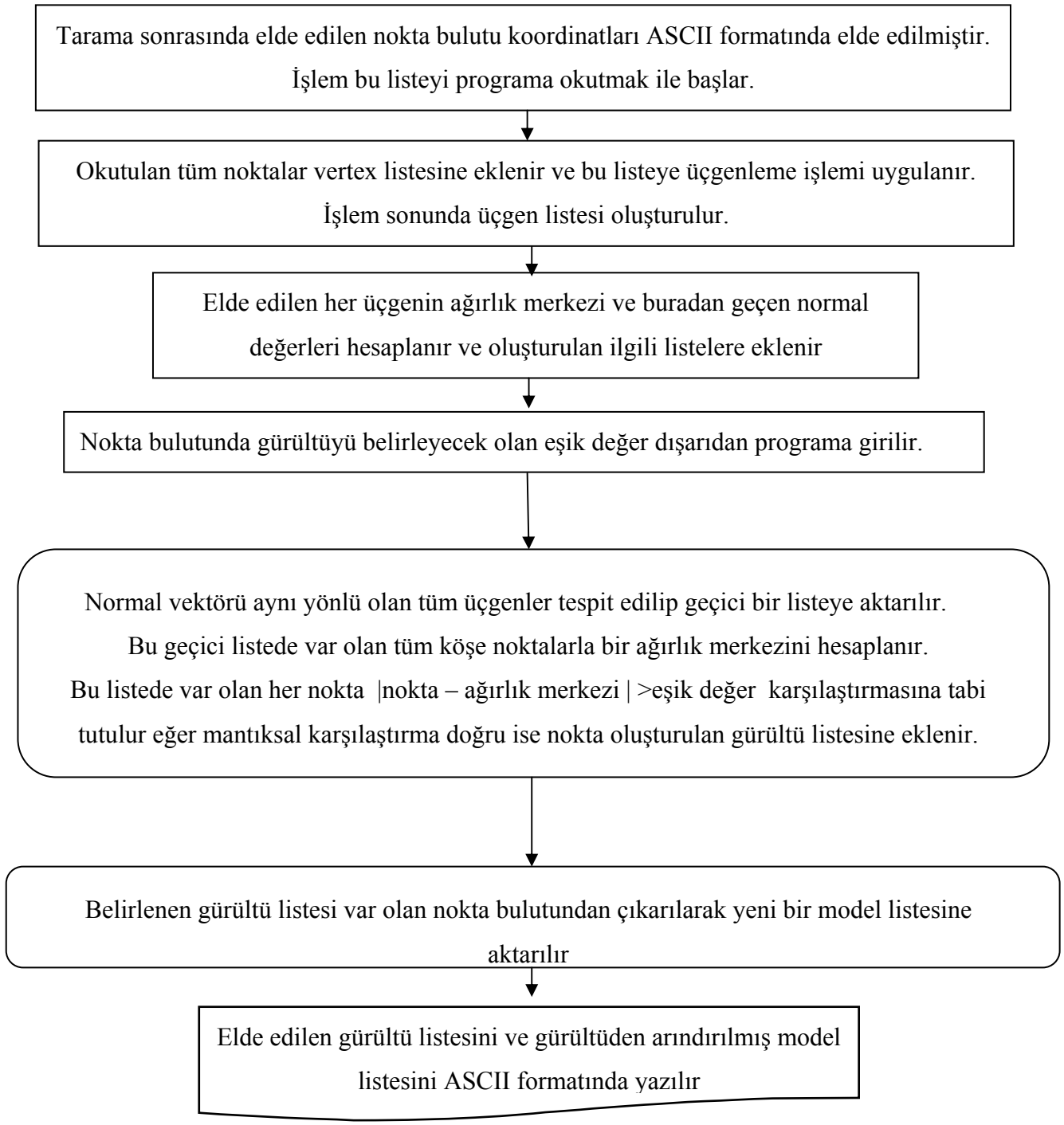
Şekil 6.6.4 Microstation XYZ TEXT

Bunun için yapılması gereken işlem Ek3 de yer alan ASCII formatı gibi bir dosyayı bu araca yerleştirmek ve işletmektir (Şekil 6.6.5).



Şekil 6.6.5 Microstationda görüntülenen nokta bulutu dosyası

"Poul Bourke Üçgenlenme Yöntemi ve Normal Vektörü Yardımıyla Geliştirilen Gürültü Eliminasyonu Algoritması" aşağıda yer alan akış diyagramındaki gibi özetlenebilinir (Şekil 6.6.6).

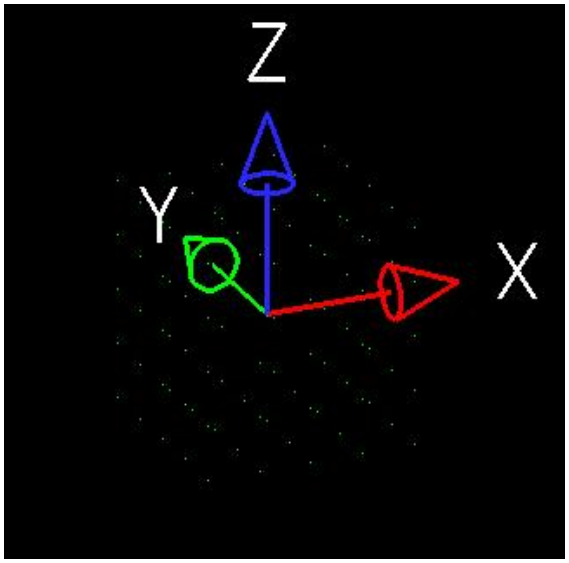


Şekil 6.6.6 Akış diyagramı

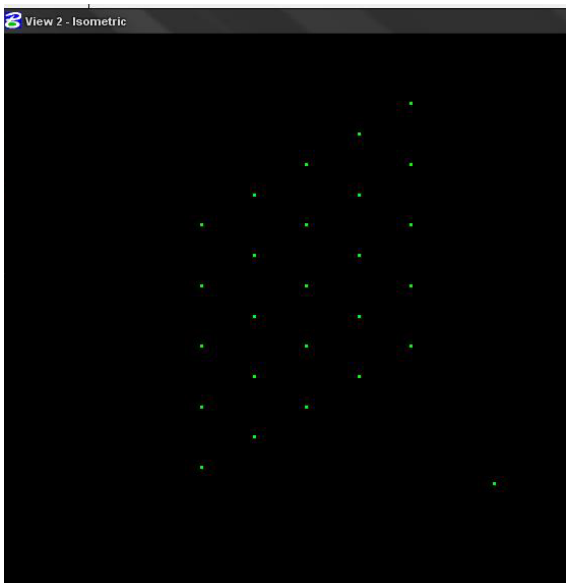
C# programlama dili ile yazılan kod kaynağı tümüyle Ek4'te bulunmaktadır.

7. ÖRNEKLER İLE YAZILIM

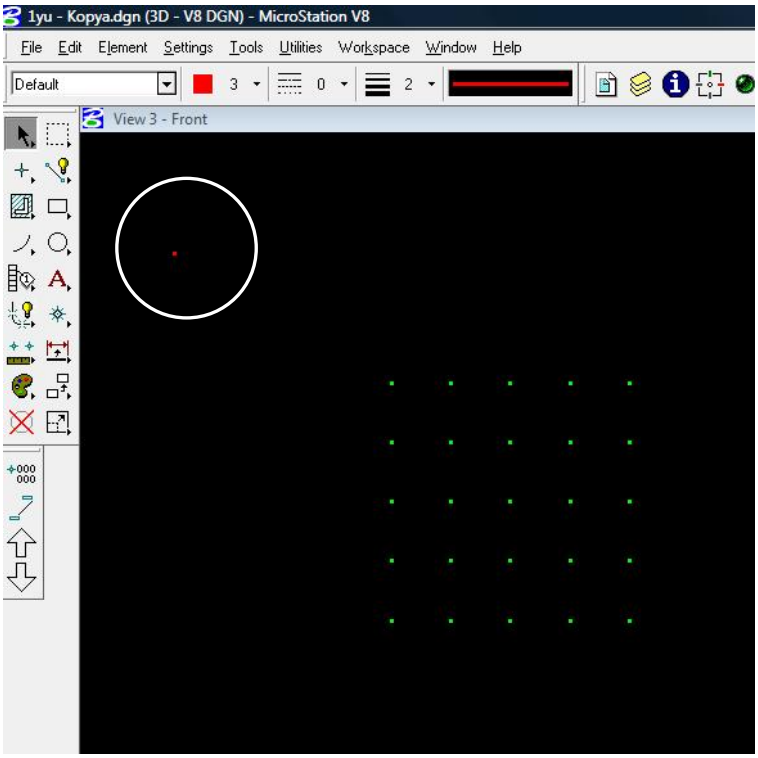
Tez kapsamından geliştirilen yazılım var olan örnekler üzerinden karşılaştırmalar yapılarak kontrol edilmiştir. Kullanılan örnekler milimetrik hassasiyet gerektiren makine parçaları içerdiği gibi yapay oluşturulmuş örneklerde bulundurmaktadır. Tarama verilerinin en önemli özelliği düzensiz oluşlarıdır. Düzenli verilerden oluşan taramalar yansıma yapmayan yüzey kullanılması gibi kendi içinde özel koşullar barındırır. Yapay oluşturulan bu örnekler yazılımın geliştirilmesi için önem taşımaktadır (Şekil 7.1).



Şekil 7.1 Küp

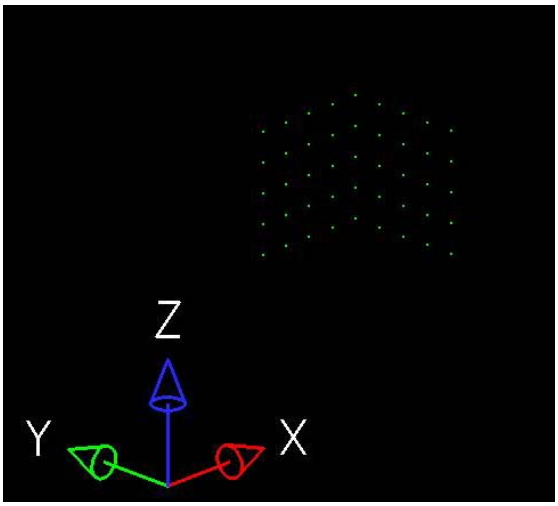


Şekil 7.2 Küpün bir yüzeyi ve yapay gürültü

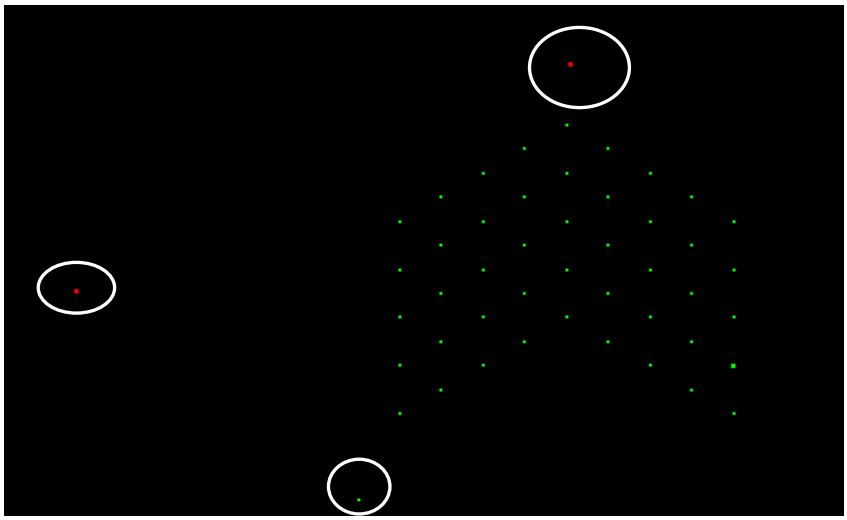


Şekil 7.3 Verilen eşik değer karşısında bulunan gürültü

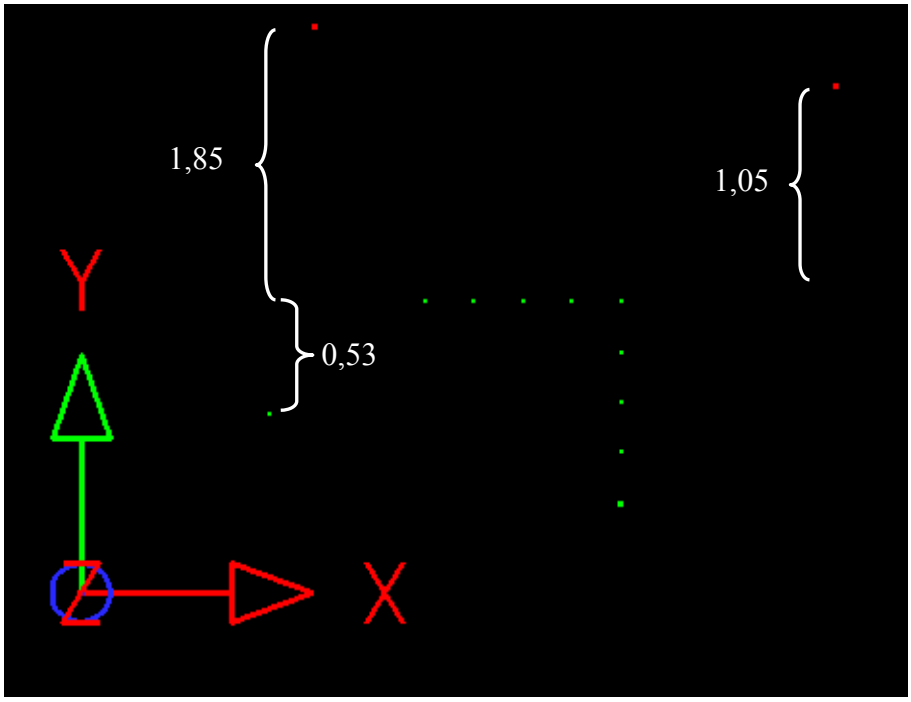
Yapay oluşturulan örnekler yazılımı anlamak için önemli bir çalışmadır. Şekil 7.2 de 0.25 cm aralıklarla atılan noktalara sahip olan tek bir düzlem bulunmakta. Bu düzlemin 2 cm uzaklığına atılan bir nokta, yapıya göre gürültü olarak adlandırılır. Dışarıdan kullanıcı tarafından girilen eşik değer (1cm) gürültü olarak belirlenen yapay oluşumu doğrulanmıştır (Şekil 7.3). Tek yüzeyli yapay bir veri sonrasında yüzey sayısının bir artması (Şekil 7.4), oluşturulan yapay gürültünün belirlenmesini etkilememiştir. Gürültüler programın ayırmış olduğu düzlemlerce ve dışarıdan girilen eşik değerince tespit edilmiştir.



Şekil 7.4 İki yüzeyli yapay model



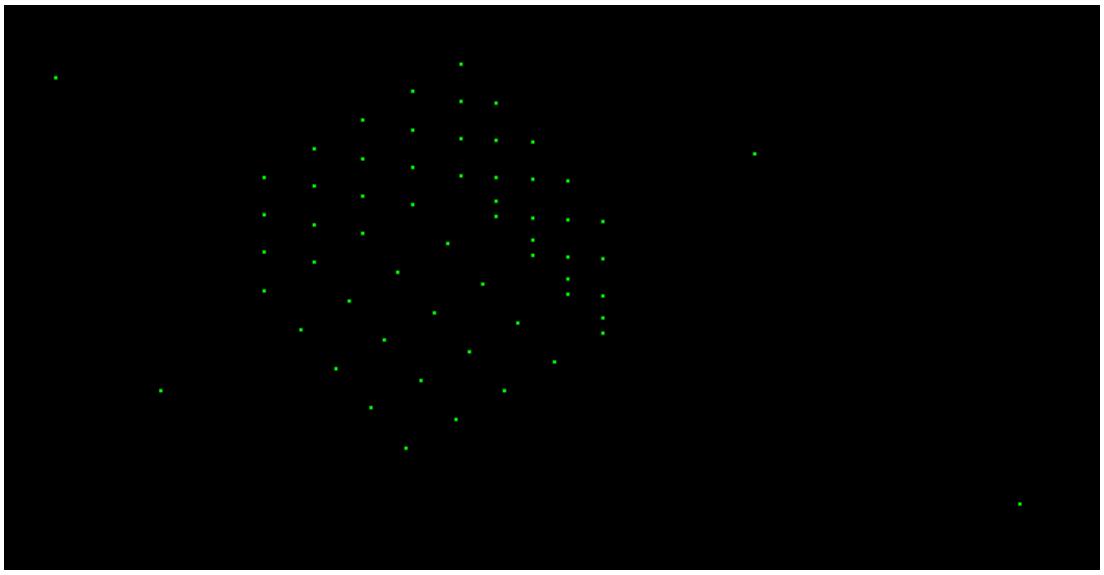
(a)



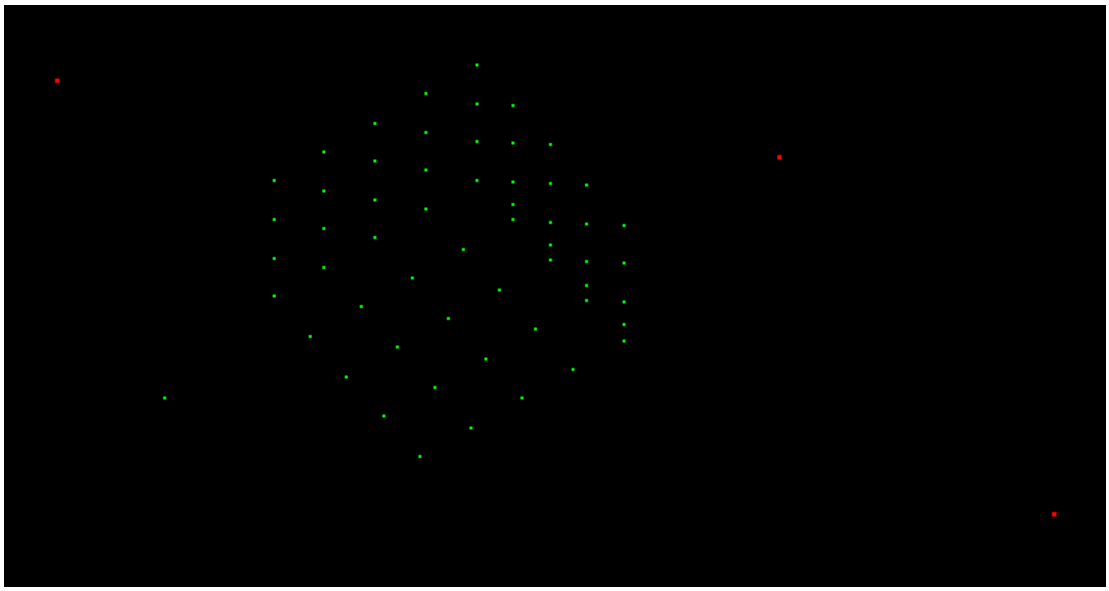
(b)

Şekil 7.5 İki yüzeyli modelde sonuçlar (a) İki yüzeyli modeli yapay gürültüleri (b) eşik değeri 1 cm olan işlem sonucunda tespit edilen 2 gürültünün durumu

Özellikle oluşturulan yapay gürültüler, yapılarında bulundukları üçgenler sayesinde var olan yüzeylerin düzlemlerinde yer alamazlar. Oluşturdukları üçgenlerin diğer 2 köşe noktası mutlaka bu düzlemin elemanları olacaktır. Gürültü olarak belirlenemeyen nokta, en yakın düzlemden 2 nokta ile üçgenleme yapsa dahi ağırlık merkezinin y ekseni üzerindeki değeri 1cm'den azdır. Bu durumda verilen eşik değerine göre nokta modele ait olarak görünür.



(a)

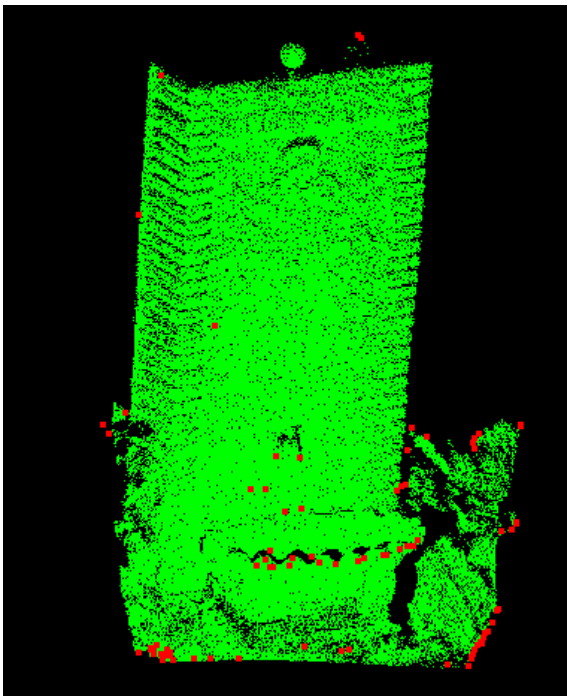


(b)

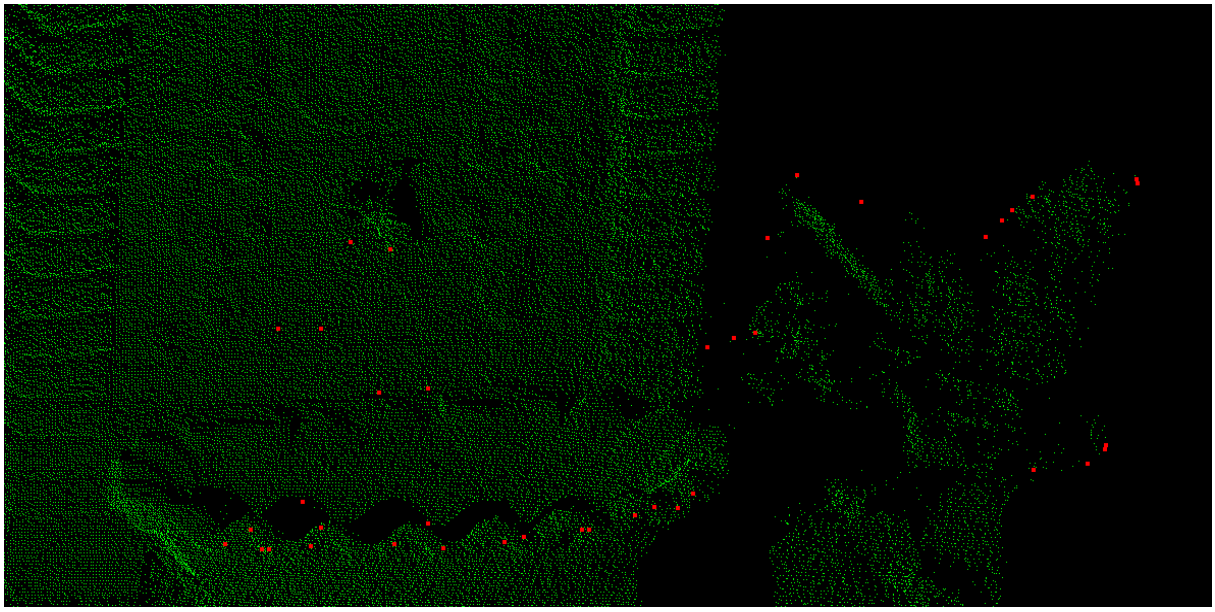
Şekil 7.6 (a) Eşik değeri 1cm olan üç yüzlü bir model ve (b) tespit edilen yapay gürültüleri

Diğer yüzeylerde de olduğu gibi yapay gürültüler belirlenen eşik değeri sayesinde bulunmuştur (Şekil 7.6) . Bulunan gürültüler kırmızı renkte işaretlenmiştir. Belirlenemeyen diğer nokta ise üç yüzeye de yakın mesafede bulunmaktadır. Noktanın yüzeylerden herhangi iki noktası ile üçgen oluşturup, kendi düzlem listesinin içinde hesapladığı ağırlık merkezine uzaklığı 1cm altında olduğu için gürültü listesine alınmamıştır.

Gerçek verilerle çalışmak bahsedilenler kadar basit ve anlaşılır düzeyde değildir. Düzensiz oldukları için birçoğu belirgin bir düzlem içinde yer alamayacaklardır. Bu nedenle üçgenler tek başına kendi düzlemlerini oluşturacaklardır. Şayet uzaklık niteliği taşıyan eşik değerde uygunsuz verilirse yazılım gürültü belirlemede başarılı olamayacaktır (Şekil 7.7).

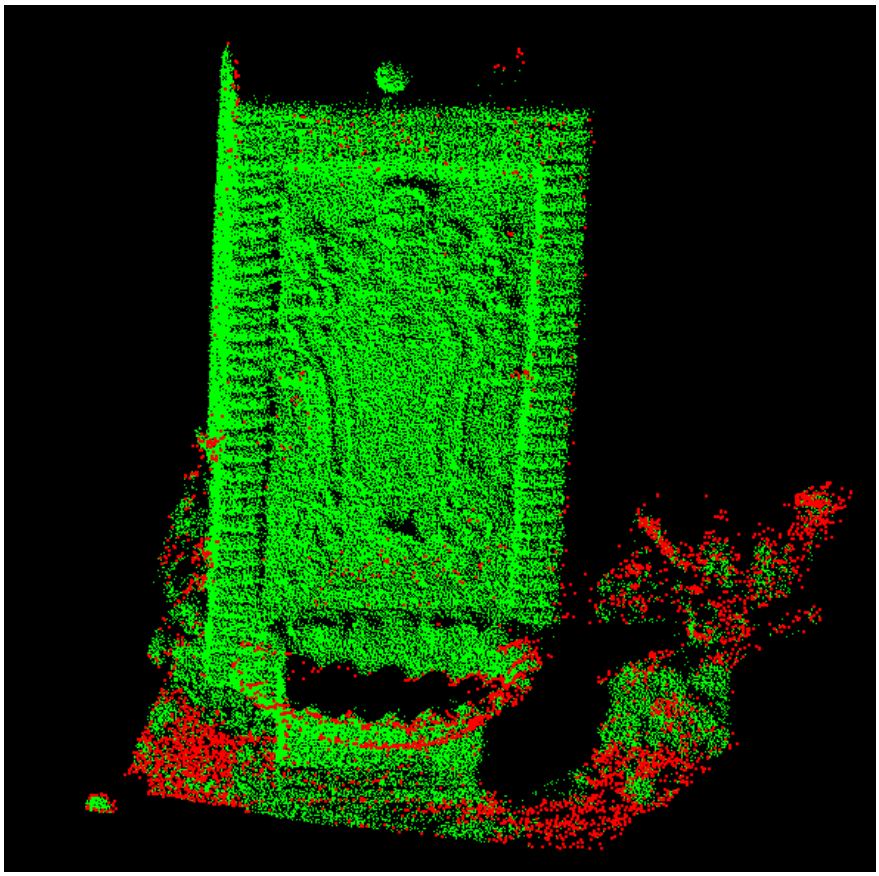


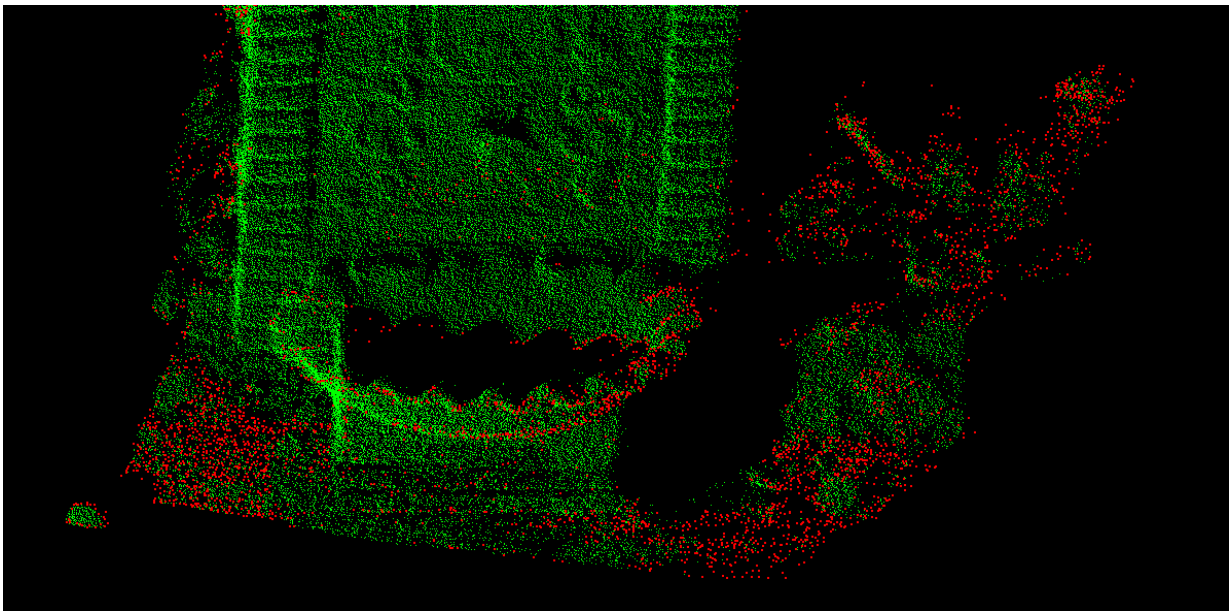
(a)



(b)

Şekil 7.7 (a) Düzensiz veride (mm hassasiyetinde) yüksek eşik değer (70 mm) sonucu (b) 126956 noktadan 87 adet gürültü belirlenmiştir.

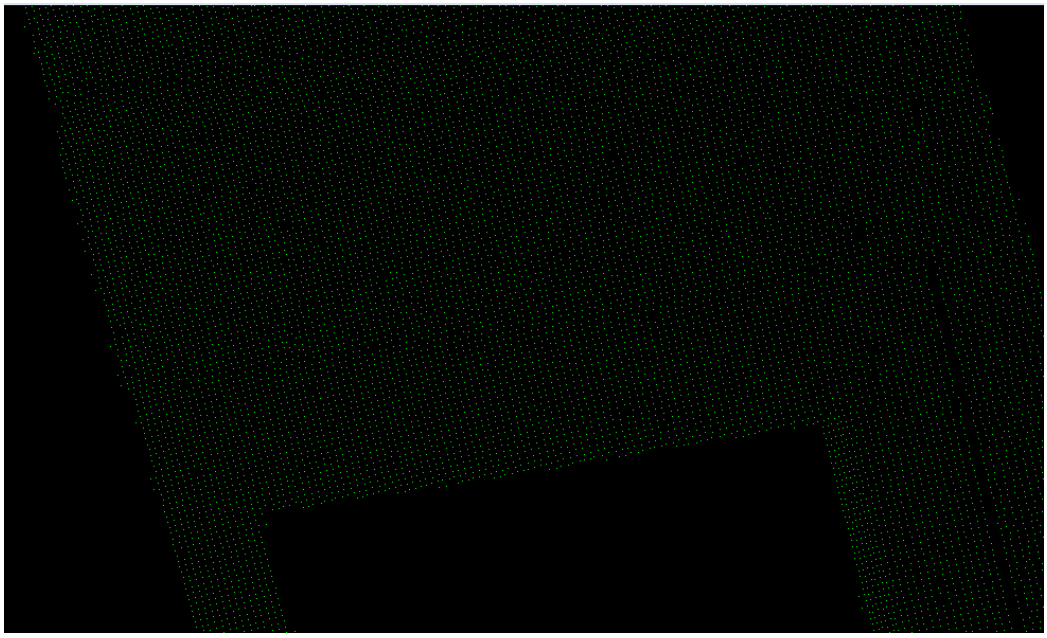




Şekil 7.8 Düzensiz veride (mm hassasiyetinde) düşük eşik değer (5 mm) sonucu: 126956 noktadan 3890 adet gürültü belirlenmiştir.

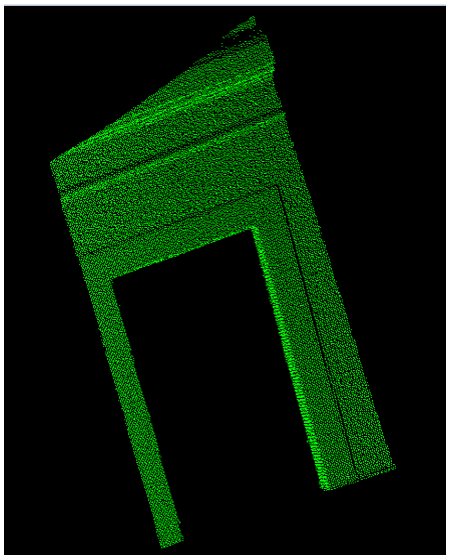
Eşik değerinde yükselen miktar belirlenen gürültü miktarını da arttırmıştır (Şekil 7.8). Bu artış var olan üçgenlerin ve onların oluşturdukları düzlemlerin ağırlık merkezlerine olan yakınlığı da artırmıştır.

Tarama sistemi, aslında veriler ne kadar detaylı, yüzeyi pürüzlü olursa olsun kendi içinde yinede bir düzen barındırmaktadır (Şekil 7.9).



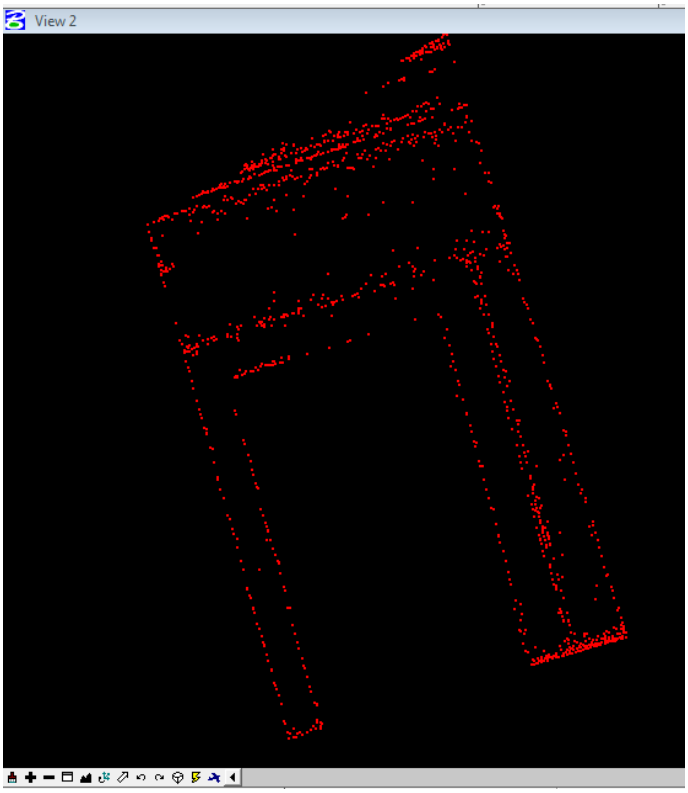
Şekil 7.9 Belli aralıklarda noktaya sahip bir tarama örneği

Bu, düzenli bir tarama örneğinin gürültü barındırmayacağı anlamına gelmez. Aksine gürültüler bu tip verilerde daha belirgin bulunurlar. Özellikle kapama sınırından kaynaklanan gürültülere etkileşimli olarak düzeltilmek, elimine etmek istense de baş edebilmek mümkün değildir. Şekil 7.10' da ki kapı örneği kapanma sınırından kaynaklanan gürültüler içermektedir. Özellikle kapıya ait olan kimi kademelerin geçişleri bu tip gürültülerin oluşmasına birebirdir.

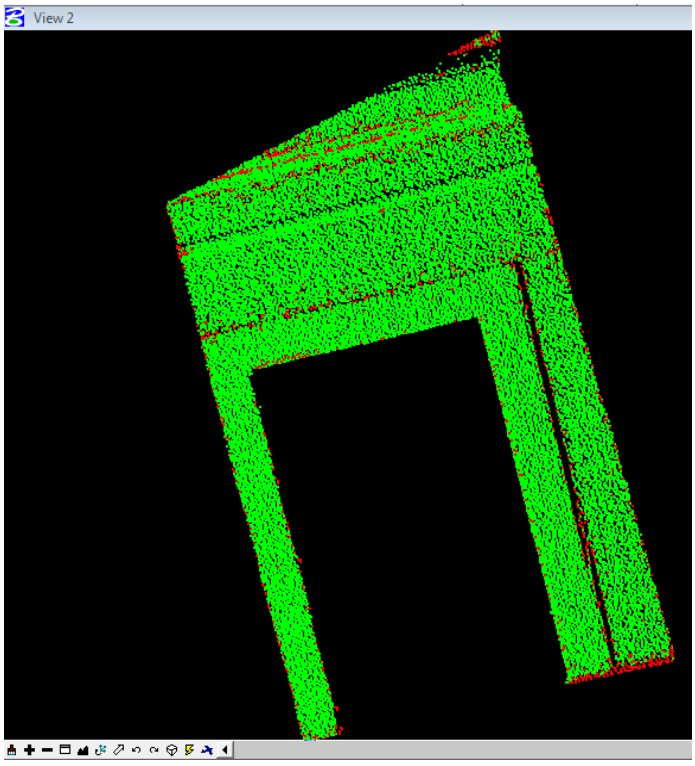


Şekil 7.10 Kapı örneği (25929 nokta)

Bu tip bir örnek işleme alınırsa karşımıza çıkan sonuçlar Şekil 7.11 da ki gibi olur.



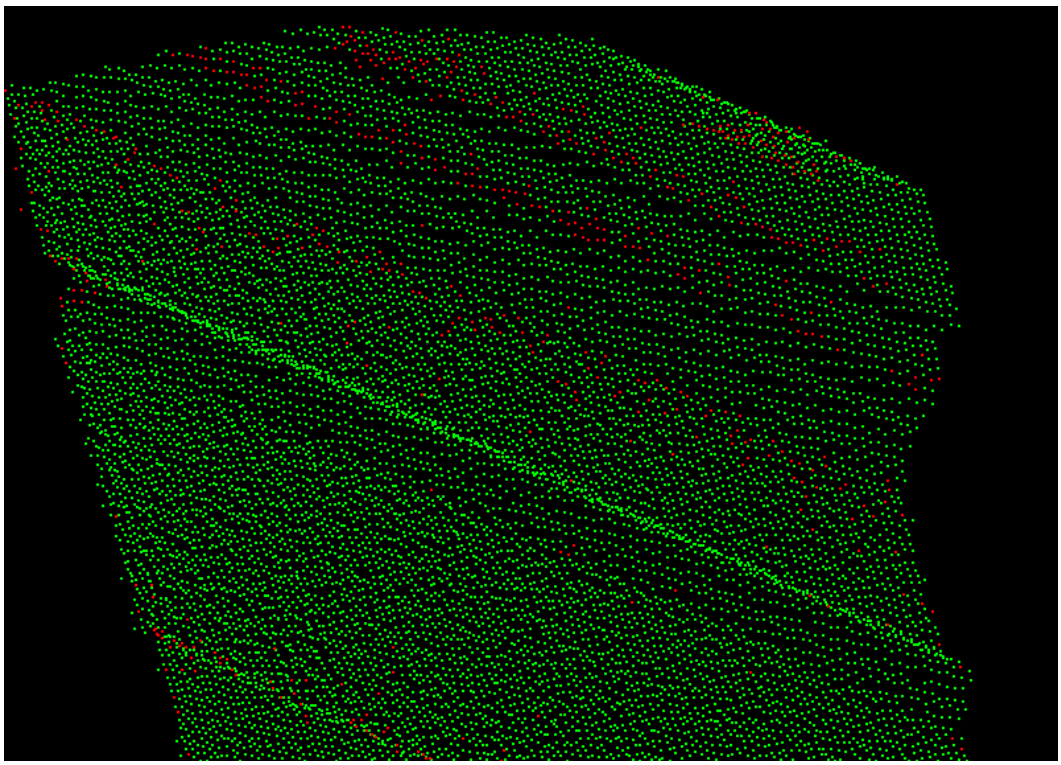
(a)



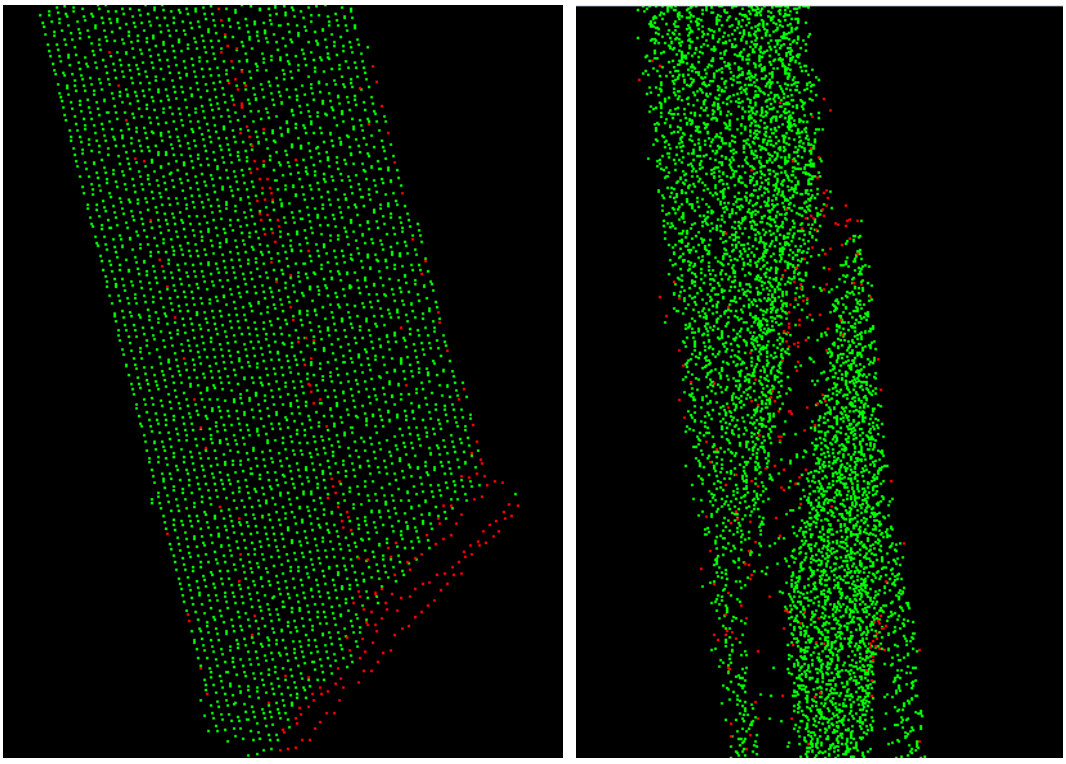
(b)

Şekil 7.11 Kapı örneği sonuçları (a) gürültü (b) orijinal ver ile gürültü

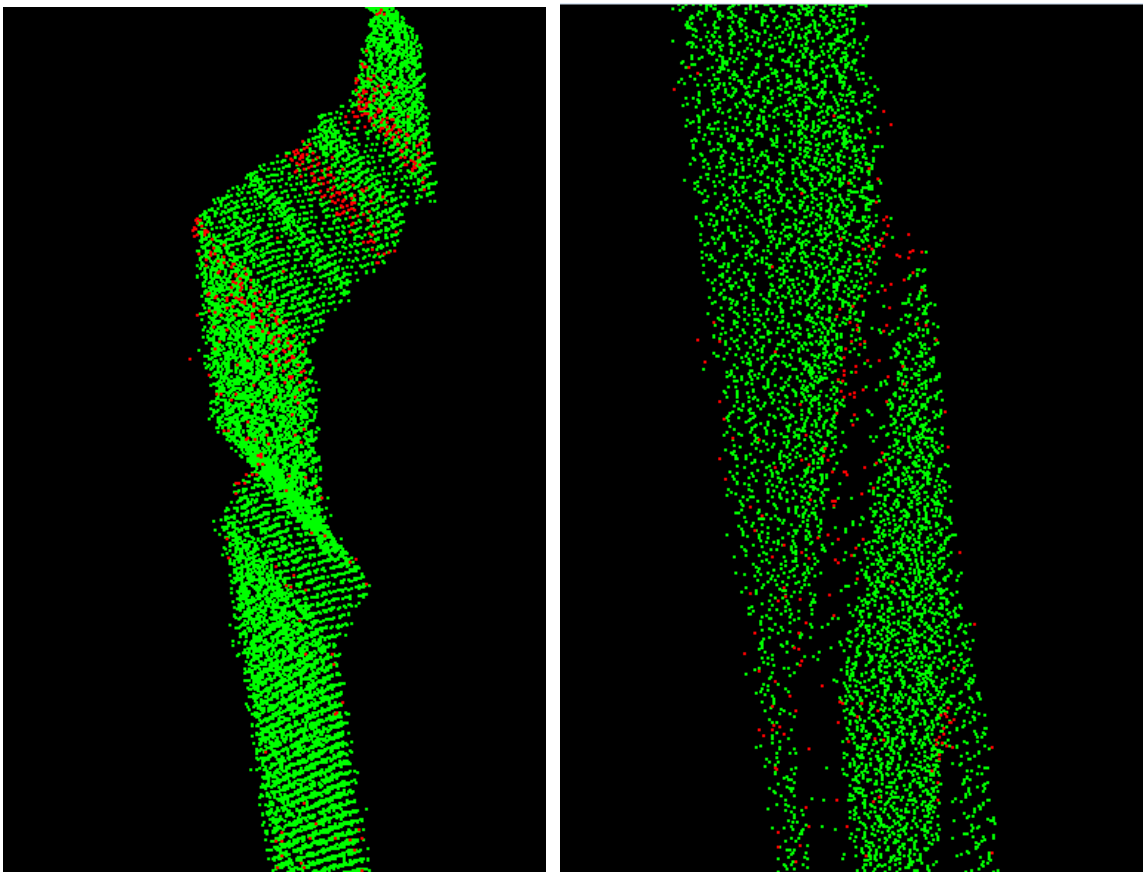
Eşik değer yüksek duyarlılıkta girilince 25929 sayı kadar noktadan 1181 tanesi gürültü olarak listeye alınır (a). Belirlenen bu gürültüler genelde kapama sınırlarından kaynaklanmıştır.



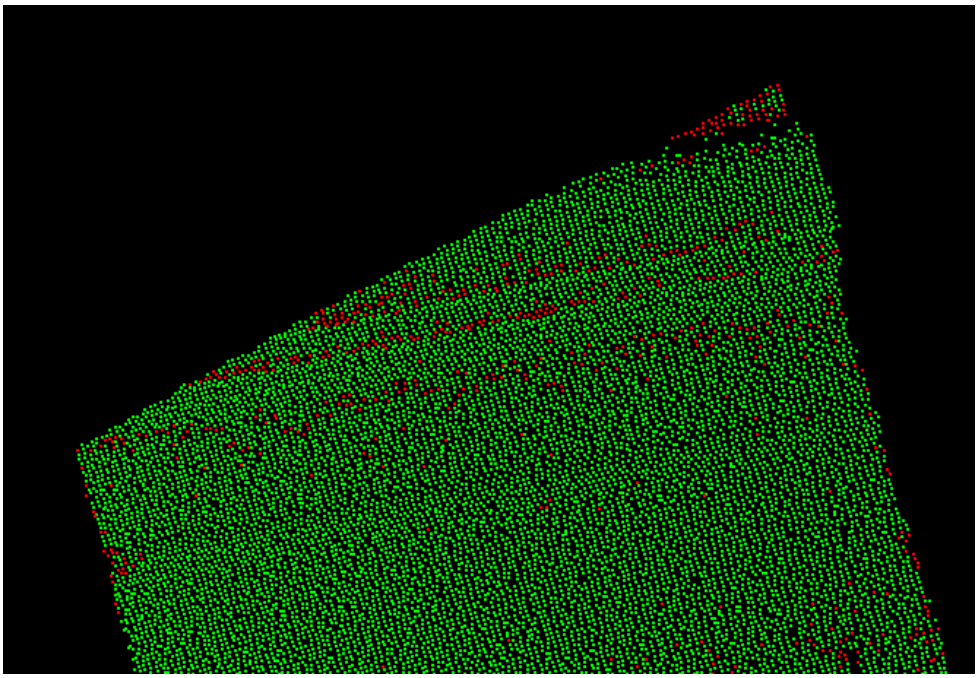
Şekil 7.12 Belirlenen gürültüler



Şekil 7.13 Belirlenen gürültüler

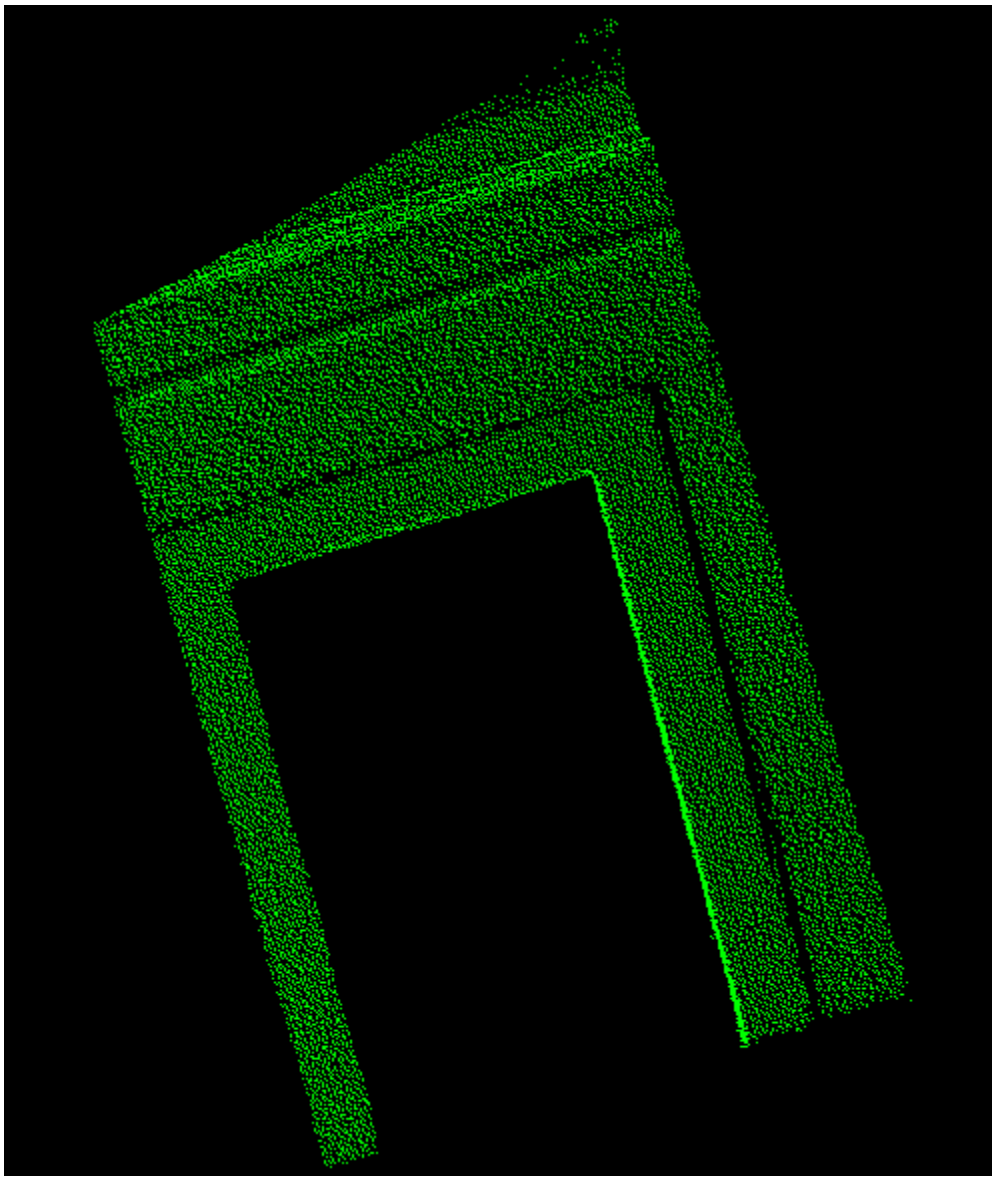


Şekil 7.14 Belirlenen gürültüler

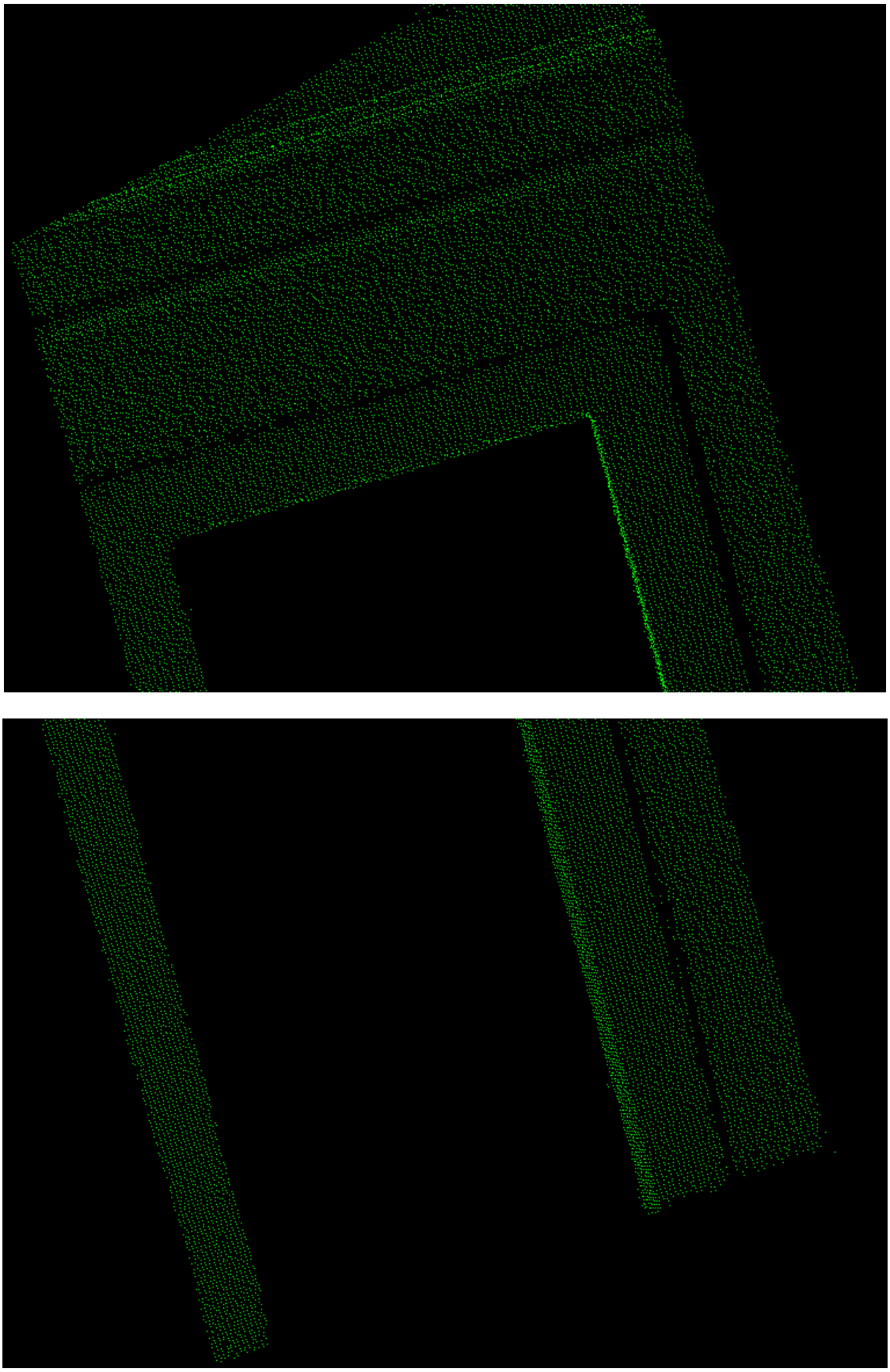


Şekil 7.15 Kapama sınırlarında yer alan ve programca tespit edilen gürültüler

Tez çalışmasında geliştirilen yazılım, gürültülü verileri ASCII formatında verdiği gibi gürültüden arındırılmış verilerin listesini de yine aynı formatta çıktısını oluşturmaktadır (Şekil 7.16).



Şekil 7.16 Gürültüden arındırılmış veri



Şekil 7.17 Temizlenmiş verinin detaylı görünüşleri

25929 örnekli mm hassasiyetinde elde edilen bir nokta bulutu farklı eşik değer miktarları ile işleme sokulduğunda aşağıda yer alan tablo sonuçları ortaya çıkmıştır.

Çizelge 7.1 Nokta bulutunda farklı eşik değerlerde bulunan gürültü miktarları

Hassasiyet	**Belirlenen Gürültü Miktarı**	**Gürültüden Arındırılmış Nokta Bulutu**
1 mm	25341	588
2 mm	18065	7864
3 mm	7100	18829
4 mm	2330	23599
5 mm	1180	24747
6 mm	774	25155
7 mm	583	25346
8 mm	498	25431
9 mm	438	25491
10 mm	388	25541
15 mm	253	25676
30 mm	125	25804
40 mm	91	25838
50 mm	78	25851

8. SONUÇ

Sonuç olarak bu çalışmada yersel lazer tarayıcılar kullanılarak elde edilen nokta bulutundaki tespit edilmesi güç olan yansıma ve saçılmalara ait noktalar, belirlenen ölçütlerce silinerek dosyaya düzenli bir şekilde yeniden kaydedildiği görülmektedir. Geliştirilen ölçüt üçgenleme yaptıktan sonra oluşan üçgenleri, normalleri bakımından kendi içinde düzlemlerine ayırmaktadır. Dışarıdan belirlenen bir eşik değer ile ayrılan düzlemlerin ağırlık merkezinin karşılaştırılması sonucu var olan düzlemler içerisinde uyuşumsuz diğer adı ile gürültülü noktaların bulunup liste haline getirilmesi söz konusudur. Algoritma, düzenli ya da düzensiz veri dağılımına sahip tüm nokta bulutlarında, sistematik ya da kaba hatalı verileri tespit edebildiği görülmüştür. Çalışma, tüme varım bir fikirin var olan tümü incelemesinden oluşmaktadır. Bir başka değişle oluşturulan üçgenler modelin tamamını değerlendirmede yardımcı olmuş ve nokta bulutunun analizlenmesine imkan tanışmıştır.

Geliştirilen algoritma yarı otomatik sistemlerin altında çalışabilen bir yapıdadır. İşleme alınan verinin form ve özelliklerine bakmaksınız analizler. Girdi ürün belirlenen ölçüt, dışarıdan girilen eşik miktarı doğrultusunda gürültüden arındırılmış çıktı ürün haline getirilmektedir. Bunun sonucu olarak geliştirilen bu algoritmayı bir modelleme yazılımında gürültü eliminasyonu yapabilen bir araç olarak görebilmek mümkün olacaktır.

Belirlenen ölçütte aranan normal vektörlerinin birebir aynı yönlü olması şartı, laboratuar ortamında taranmayan aksine dış etkenlere maruz kalan, yüzey özellikleri sebebiyle belli bir düzene sahip olmayan objeye ait nokta bulutu verilerinin bu eşitlik karşısında düzlemlerine ayrılması, işlem hızı açısından güçlük yaratmaktadır. Binlerce farklı normal yönüne sahip üçgenlerin evrensel düzlemlerini bu eşitlik karşısında oluşturulması, algoritmada işlem hızını düşürmektedir. Bu eşitliğin esnek değerler arasında bulunması için yapılacak olan yeni bir çalışma, performansın değişeceğini gösterecektir.

Tespit edilen gürültü listelerinin ve belirlenen temiz modelin görselleştirilmesinde kullanılan yardımcı programlar yerine VRML(Virtual Reality Modelling Language) gibi grafik objelerin internet üzerinden görüntülenebilmesine imkan veren alternatif yollar kullanılması yazılan algoritmanın özgünlüğünü görselliğini kuvvetlendirecektir.

Bundan sonraki aşamada yazılıma yönelik mevcut eksiklikleri giderildikten sonra algortimanın daha da geliştirilebilmesi mümkündür. Örneğin taraması yapılmış ve işlenmek üzere kullanıcıya devredilen veriyi programa öğreterek düzlemlerine ayrılması söz konusu olabilir. Veri hakkında herhangi bir bilgiye sahip olmayan kullanıcı için alternatif bir modelleme örneği yaratılmış olunur. Modelin ana toplanma merkezini baz alarak belli toleranslarla düzlemlerine ayrılması, ayrıca ayrılan bu düzlemlerin komşuluklarını inceleyerek ortak olan noktaları sayesinde kenar çıkarımı algoritmasını oluşturmak mümkün olabilir. Geliştirilebilecek olan kenar algoritması sayesinde var olan yazlım bir adım daha ileriye gidecektir. Bunun yanında işlemler sonrasında oluşturulan 3D sayısal modele, bu modele ait görüntü verisinin de eklenmesiyle daha gerçekçi bir sayısal model oluşturulabilmesi de mümkündür.

Üzerinde çalışılmaya devam edilirse mevcut üç boyutlu sayısal model oluşturma yöntemleri içerisinde alternatif bir yöntem olması olasıdır.

KAYNAKLAR

Altuntaş, C., Yıldız, F., (2008), "Yersel Lazer Tarayıcı Ölçme Prensipleri ve Nokta Bulutlarının Birleştirilmesi", Jeodezi Jeoinformasyon ve Arazi Yönetimi Dergisi 2008/1 sayı 98.

Beesley, C., Balancing Modernization and Preservation on Easter Island, [acronym] magazine, Issue 8, Summer 2008

Berg M.D., Kreveld M.V., Overmars M. ve Schwarzkopf O.,(1997) "Computational Geometry-Algorithms and Applications", Springer-Verlag Berlin Heidelberg, Germany

Bornaz, L., Rinaudo, F., (2004) " Terrestrial Laser Scanner data Processing", Commission V, WG V/4

Bourke,P., "Triangulate Efficient Triangulation Algorithm Suitable for Terrain Modelling", Pan Pacific Computer Conference, Beijing, China, Ocak 1989.

Dayık, M., Kodaloğlu, M., Güler, Ç., Sivrikaya, D., (2008), "3 Boyutlu Vücut Tarama Sistemleri", Tekstil Teknolojileri Elektronik Dergisi 2008 (3) 59 -76

Durmuş, S., İplikçi, S., (2007), "Veri Kümeleme Algoritmalarının Performansları Üzerine Karşılaştırmalı Bir Çalışma", Akademik Bilişim 2007 Dumlupınar Üniversitesi, Kütahya

Demir, N., "Yersel Laser Tarama ve Fotogrametrinin Birlikte Kullanılması", Yüksek Lisans Tezi,YTÜ Fen Bilimleri Enst.,İstanbul, 2005

Gopi M., Krishnan S., Silva C.T., "Surface REconstruction Based On Lower Dimensional Localized Delaunay Triangulation", Eurographics 2000 ,Volume 19 (2000), No. 3

Gümüş, K., Erkaya H., (2007), "Mühendislik Uygulamalarında Kullanılan Yersel Lazer Tarayıcı Sistemler", TMMOB Harita ve Kadastro Mühendisleri Odası 11. Türkiye Harita Bilimsel ve Teknik Kurultayı 2 – 6 Nisan 2007, Ankara.

Gümüş, K., Erkaya, H., Tunalıoğlu, N., (2009) "Yersel Tarama Verilerinde Çevresel ve Objesel Nedenlerden Kaynaklanan Hatalar", TMMOB Harita ve Kadastro Mühendisleri Odası 12. Türkiye Harita Bilimsel ve Teknik Kurultayı 11-15 Mayıs 2009, Ankara

Hohner, N., ClearEdge3D releases EdgeWise Automated Pointcloud to CAD, SparView Vol. 7, No. 11, June 9, 2009

Kıvılcım, Ö. (2005) "Yersel Lazer Tarayıcı ile Üç Boyutlu Modelleme", Yıldız Teknik Üniversitesi Lisans Tezi, İstanbul

Mahir, N., " Vektörler ile Tanımlanan İşlemler", Açık Öğretim Fakültesi Analitik Geometri 4. Ünite 65-84.

Morita, J., Yokoyama, H., Chikatsu, H., (2003) "Efficient Noise Reduction of Laser Scanning Data For Archaeological", The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXIV, Part 5/W12.

Papadimitriou, S., Hiroyuki, K., Gibbons, P. B. and Faloutsos, C., (2002). "Fast outlier detection using the local correlation Integral", Intel Corporation, IRP-TR-02-09.

Pion, S., Teillaud, M., Computational Geometry Algorithms Library Packages IX Triangulations and Delaunay Triangulations Chapter 37.

Sotoodeh, S., (2006), "Outlier Detection in Laser Scanner Point Clouds", Commission V IAPRS Volume XXXVI, Part 5 , WG V/3 Dresden

Sotoodeh, S., (2007), "Hierarchical Clustered Outlier Detection in Laser Scanner Point Clouds", Commission V/3 ,IAPRS Volume XXXVI, Part 3 / W52

Yanalak, M., "Yüzey Modelleme Üçgenleme Yöntemleri", (2001) Harita Dergisi Temmuz sayı 126.

Waggot S.M., Clegg P., Jones R.R. (2005)"Combining Terrestrial Laser Scanning, TK GPS and 3D Visualisation: Application of Optical 3D Measurement in Geological Exploration, 7th Conference on 3D Optical Measurement Techniques, s.on CD, Vienna, Austria.

Weyrich, T., Pauly, M., Keiser, R., Heinzle, S., Scandella, S., Gross, M., (2004), "Post-processing of Scanned 3D Surface Data", Eurographics Symposium on Point-Based Graphics 04

İNTERNETTEN KAYNAKLAR

http://www.matematikci.org

http://www.cs.ucf.edu/~ymin/

http://local.wasp.uwa.edu.au/~pbourke/

http://www.laserdesign.com/

EKLER

Ek 1 Poul Bourke Üçgenlemesine Ait Kaynak Kodlar

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace delaunay_duzlemayir
{
    public partial class Form1 : Form
    //public class Form1 : System.Windows.Forms.Form
    {
        //Points (Vertices)
        public struct dVertex
        {
            public double x;
            public double y;
            public double z;
        }
        //Created Triangles, vv# are the vertex pointers
        public struct dTriangle
        {
            public int vv0;
            public int vv1;
            public int vv2;
        }
        //Set these as applicable
        public int MaxVertices = 500000;
        public int MaxTriangles = 1000000;
        public dVertex[] Vertex;
        public dTriangle[] Triangle;
        public dVertex[] Agirliknok;
        public dVertex[] normal;


        int tPoints; //Variable for total number of points (vertices)
        int HowMany;
        public Form1()
        {
            InitializeComponent();

            // ------------------------------------------------------------
```

```
//Our points
            Vertex = new dVertex[MaxVertices];
 //Our Created Triangles
            Triangle = new dTriangle[MaxTriangles];
//Initiate total points to 1, using base 0 causes problems in the functions
            tPoints = 0;
            Agirliknok = new dVertex[MaxTriangles];
            normal = new dVertex[MaxTriangles];
            // ------------------------------------------------------------

        }

private bool InCircle(ref double xp, ref double yp, ref double x1, ref
double y1, ref double x2, ref double y2, ref double x3, ref double y3, ref
double xc, ref double yc, ref double r)
        {
          //Return TRUE if the point (xp,yp) lies inside the circumcircle
          //made up by points (x1,y1) (x2,y2) (x3,y3)
          //The circumcircle centre is returned in (xc,yc) and the radius r
          //NOTE: A point on the edge is inside the circumcircle
            bool TheResult;

            double eps;
            double m1;
            double m2;
            double mx1;
            double mx2;
            double my1;
            double my2;
            double dx;
            double dy;
            double rsqr;
            double drsqr;

            TheResult = false;
            eps = 0.000001;

      if (System.Math.Abs(y1 - y2) < eps && System.Math.Abs(y2 - y3) < eps)
            {
              //MessageBox.Show("INCIRCUM - F - Points are coincident !!");
                TheResult = false;
                return TheResult;
            }
            if (System.Math.Abs(y2 - y1) < eps)
            {
                m2 = (double)-(x3 - x2) / (y3 - y2);
                mx2 = (double)(x2 + x3) / 2;
                my2 = (double)(y2 + y3) / 2;
                xc = (x2 + x1) / 2;
```

```
                yc = m2 * (xc - mx2) + my2; }
            else if (System.Math.Abs(y3 - y2) < eps)
              {
                  m1 = (double)-(x2 - x1) / (y2 - y1);
                  mx1 = (double)(x1 + x2) / 2;
                  my1 = (double)(y1 + y2) / 2;
                  xc = (x3 + x2) / 2;
                  yc = m1 * (xc - mx1) + my1;
              }
              else
              {
                  m1 = (double)-(x2 - x1) / (y2 - y1);
                  m2 = (double)-(x3 - x2) / (y3 - y2);
                  mx1 = (double)(x1 + x2) / 2;
                  mx2 = (double)(x2 + x3) / 2;
                  my1 = (double)(y1 + y2) / 2;
                  my2 = (double)(y2 + y3) / 2;
                  xc = (m1 * mx1 - m2 * mx2 + my2 - my1) / (m1 - m2);
                  yc = m1 * (xc - mx1) + my1;
              }

              dx = x2 - xc;
              dy = y2 - yc;
              rsqr = dx * dx + dy * dy;
              r = System.Math.Sqrt(rsqr);
              dx = xp - xc;
              dy = yp - yc;
              drsqr = dx * dx + dy * dy;

              if (drsqr <= rsqr)
              {
                  TheResult = true;

              }
              return TheResult;

          }
      private int WhichSide(ref int xp, ref int yp, ref int x1, ref int y1, ref
      int x2, ref int y2)
          {
              // Determines which side of a line the point (xp,yp) lies.
              // The line goes from (x1,y1) to (x2,y2)
              // Returns -1 for a point to the left
              //          0 for a point on the line
              //         +1 for a point to the right
              double equation;
              equation = ((yp - y1) * (x2 - x1)) - ((y2 - y1) * (xp - x1));
              if (equation > 0)
              {
                  return -1;
```

```
        }
        else if (equation == 0)
        {
            return 0;
        }
        else
        {
            return 1;
        }
    }
    public int Triangulate(ref int nvert)
    {
        // Takes as input NVERT vertices in arrays Vertex()
        // Returned is a list of NTRI triangular faces in the array
        // Triangle(). These triangles are arranged in clockwise order.
        bool[] Complete = new bool[MaxTriangles];
        int[,] Edges = new int[3, MaxTriangles * 3];

        int Nedge;

        // For Super Triangle
        double xmin;
        double xmax;
        double ymin;
        double ymax;
        double xmid;
        double ymid;
        double dx;
        double dy;
        double dmax;

        // General Variables
        int i;
        int j;
        int k;
        int ntri;
        bool inc;

        double xc;
        xc = 0;
        double yc;
        yc = 0;
        double r;
        r = 0;
```

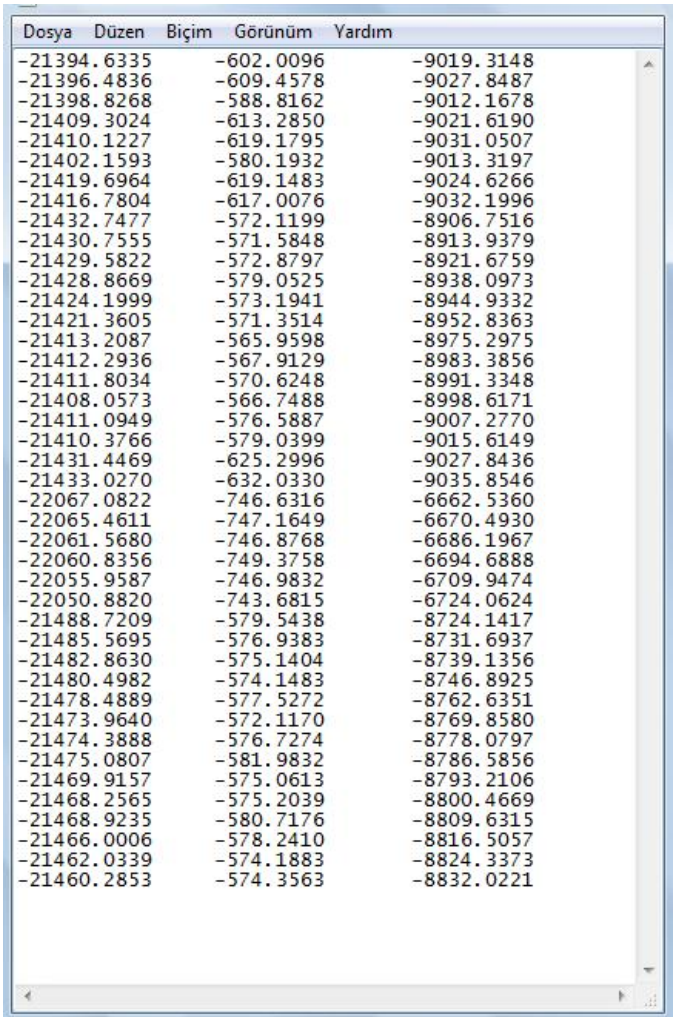
```
// Find the maximum and minimum vertex bounds.
// This is to allow calculation of the bounding triangle
xmin = Vertex[1].x;
ymin = Vertex[1].y;
xmax = xmin;
ymax = ymin;
for (i = 2; i <= nvert; i++)
{
    if (Vertex[i].x < xmin)
    {
        xmin = Vertex[i].x;
    }
    if (Vertex[i].x > xmax)
    {
        xmax = Vertex[i].x;
    }
    if (Vertex[i].y < ymin)
    {
        ymin = Vertex[i].y;
    }
    if (Vertex[i].y > ymax)
    {
        ymax = Vertex[i].y;
    }
}

dx = xmax - xmin;
dy = ymax - ymin;
if (dx > dy)
{
    dmax = dx;
}
else
{
    dmax = dy;
}

xmid = (xmax + xmin) / 2;
ymid = (ymax + ymin) / 2;

// Set up the supertriangle
// This is a triangle which encompasses all the sample points.
// The supertriangle coordinates are added to the end of the
// vertex list. The supertriangle is the first triangle in
// the triangle list.
Vertex[nvert + 1].x = (int)(xmid - 2 * dmax);
Vertex[nvert + 1].y = (int)(ymid - dmax);
Vertex[nvert + 2].x = (int)xmid;
Vertex[nvert + 2].y = (int)(ymid + 2 * dmax);
```

```
            Vertex[nvert + 3].x = (int)(xmid + 2 * dmax);
            Vertex[nvert + 3].y = (int)(ymid - dmax);
            Triangle[1].vv0 = nvert + 1;
            Triangle[1].vv1 = nvert + 2;
            Triangle[1].vv2 = nvert + 3;
            Complete[1] = false;
            ntri = 1;
            // Include each point one at a time into the existing mesh
            for (i = 1; i <= nvert; i++)
            {

                Nedge = 0;
                // Set up the edge buffer.
                // If the point (Vertex(i).x,Vertex(i).y) lies inside the
circumcircle then the
                // three edges of that triangle are added to the edge
buffer.
                j = 0;
                do
                {
                    j = j + 1;
                    if (Complete[j] != true)
                    {
                        inc = InCircle(ref Vertex[i].x, ref Vertex[i].y,
ref Vertex[Triangle[j].vv0].x, ref Vertex[Triangle[j].vv0].y, ref
Vertex[Triangle[j].vv1].x, ref Vertex[Triangle[j].vv1].y, ref
Vertex[Triangle[j].vv2].x, ref Vertex[Triangle[j].vv2].y, ref xc, ref yc,
ref r);
                        // Include this if points are sorted by X
                        // If (xc + r) < Vertex(i).x Then
                        // complete(j) = True
                        // Else
                        if (inc)
                        {
                            Edges[1, Nedge + 1] = Triangle[j].vv0;
                            Edges[2, Nedge + 1] = Triangle[j].vv1;
                            Edges[1, Nedge + 2] = Triangle[j].vv1;
                            Edges[2, Nedge + 2] = Triangle[j].vv2;
                            Edges[1, Nedge + 3] = Triangle[j].vv2;
                            Edges[2, Nedge + 3] = Triangle[j].vv0;
                            Nedge = Nedge + 3;
                            Triangle[j].vv0 = Triangle[ntri].vv0;
                            Triangle[j].vv1 = Triangle[ntri].vv1;
                            Triangle[j].vv2 = Triangle[ntri].vv2;
                            Complete[j] = Complete[ntri];
                            j = j - 1;
                            ntri = ntri - 1;
                        }
                    }
                } while (j < ntri);
```

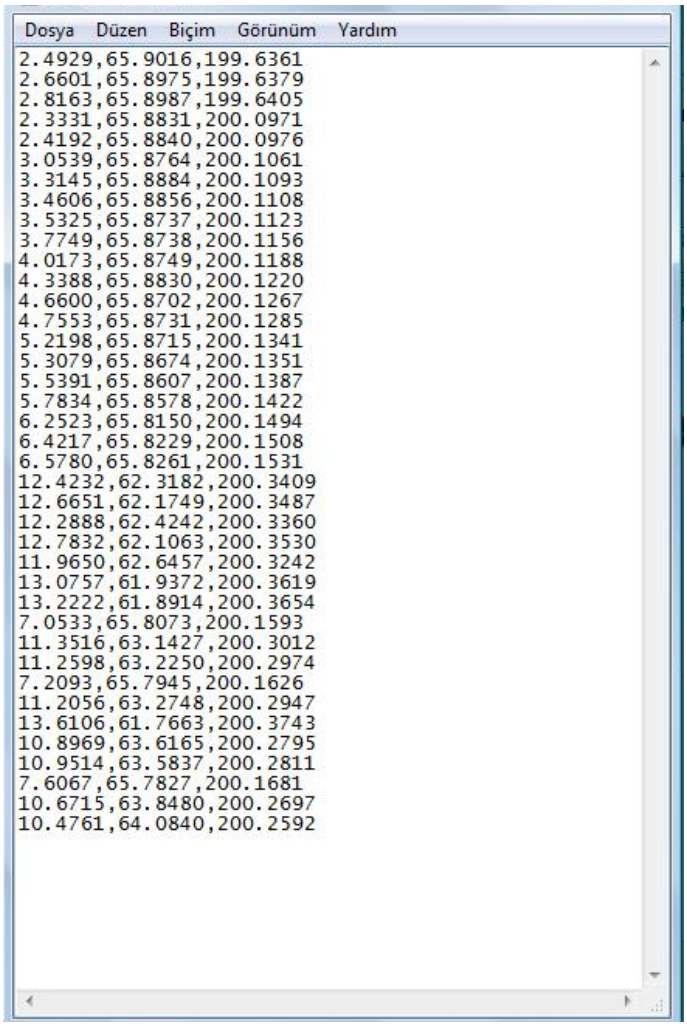
```
            // Tag multiple edges
            // Note: if all triangles are specified anticlockwise then all
            // interior edges are opposite pointing in direction.
               for (j = 1; j <= Nedge - 1; j++)
               {
                   if (Edges[1, j] != 0 && Edges[2, j] != 0)
                   {
                       for (k = j + 1; k <= Nedge; k++)
                       {
                           if (Edges[1, k] != 0 && Edges[2, k] != 0)
                           {
                               if (Edges[1, j] == Edges[2, k])
                               {
                                   if (Edges[2, j] == Edges[1, k])
                                   {
                                       Edges[1, j] = 0;
                                       Edges[2, j] = 0;
                                       Edges[1, k] = 0;
                                       Edges[2, k] = 0;
                                   }
                               }
                           }
                       }
                   }
               }
               // Form new triangles for the current point
               // Skipping over any tagged edges.
               // All edges are arranged in clockwise order.
               for (j = 1; j <= Nedge; j++)
               {
                   if (Edges[1, j] != 0 && Edges[2, j] != 0)
                   {
                       ntri = ntri + 1;
                       Triangle[ntri].vv0 = Edges[1, j];
                       Triangle[ntri].vv1 = Edges[2, j];
                       Triangle[ntri].vv2 = i;
                       Complete[ntri] = false;
                   }
               }
           }
       // Remove triangles with supertriangle vertices
       // These are triangles which have a vertex number greater than NVERT
           i = 0;
           do
           {
               i = i + 1;
               if (Triangle[i].vv0 > nvert || Triangle[i].vv1 > nvert ||
Triangle[i].vv2 > nvert)
               {
```

```
                Triangle[i].vv0 = Triangle[ntri].vv0;
                Triangle[i].vv1 = Triangle[ntri].vv1;
                Triangle[i].vv2 = Triangle[ntri].vv2;
                i = i - 1;
                ntri = ntri - 1;
            }
        } while (i < ntri);

        return ntri;

    }
```

Ek 2 Girdi Verisi Formatı



Ek 3 Microstation Programında Görüntülemek İçin Gerekli Olan Veri Formatı



Ek 4 Kaynak Kodlar

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;


namespace delaunay_duzlemayir
{
    public partial class Form1 : Form
    //public class Form1 : System.Windows.Forms.Form
    {
        //Points (Vertices)
        public struct dVertex
        {
            public double x;
            public double y;
            public double z;
        }
        //Created Triangles, vv# are the vertex pointers
        public struct dTriangle
        {
            public int vv0;
            public int vv1;
            public int vv2;
        }
        //Set these as applicable
        public int MaxVertices = 500000;
        public int MaxTriangles = 1000000;
        public dVertex[] Vertex;
        public dTriangle[] Triangle;
        public dVertex[] Agirliknok;
        public dVertex[] normal;


        int tPoints; //Variable for total number of points (vertices)
        int HowMany;
        public Form1()
        {
            InitializeComponent();

            // ------------------------------------------------------------
```

```
//Our points
            Vertex = new dVertex[MaxVertices];
 //Our Created Triangles
            Triangle = new dTriangle[MaxTriangles];
//Initiate total points to 1, using base 0 causes problems in the functions
            tPoints = 0;
            Agirliknok = new dVertex[MaxTriangles];
            normal = new dVertex[MaxTriangles];
            // ------------------------------------------------------------

        }
//poul bourke çalışması
private bool InCircle(ref double xp, ref double yp, ref double x1, ref
private int WhichSide(ref int xp, ref int yp, ref int x1, ref int y1, ref
int x2, ref int y2)
public int Triangulate(ref int nvert)

        private void Form1_Load(object sender, EventArgs e)
        {
            string[] lineArr;

            openFileDialog1.ShowDialog();
            //MessageBox.Show(openFileDialog1.FileName);
            StreamReader sr;
            char sep;
            char localSep;
            if (Convert.ToDouble("1,1") == 1.1)
            {
                sep = '.';
                localSep = ',';
            }
            else
            {
                sep = ',';
                localSep = '.';
            }

            using (sr = new StreamReader(openFileDialog1.FileName))
            {
                while (!sr.EndOfStream)
                {

                    lineArr = sr.ReadLine().Split('\t');
                    //dVertex a = new dVertex();
                    //nokta a = new nokta();
                    Vertex[tPoints].x =
Convert.ToDouble(lineArr[0].Replace(sep, localSep));
                    Vertex[tPoints].y =
Convert.ToDouble(lineArr[1].Replace(sep, localSep));
                    Vertex[tPoints].z =
```

```
Convert.ToDouble(lineArr[2].Replace(sep, localSep));
                    tPoints++;
                }
                 if (tPoints > 2)
                {
                    HowMany = Triangulate(ref tPoints);

                }

            }

//ağırlık noktasının ve normalin hesaplanması;
            for (int g = 1; g <= HowMany; g++)
            {
Agirliknok[g].x = (Vertex[Triangle[g].vv0].x + Vertex[Triangle[g].vv1].x +
Vertex[Triangle[g].vv2].x) / 3;
Agirliknok[g].y = (Vertex[Triangle[g].vv0].y + Vertex[Triangle[g].vv1].y +
Vertex[Triangle[g].vv2].y) / 3;
Agirliknok[g].z = (Vertex[Triangle[g].vv0].z + Vertex[Triangle[g].vv1].z +
Vertex[Triangle[g].vv2].z) / 3;

            }

            for (int i = 1; i <= HowMany; i++)
            {
                normal[i].x = ((Vertex[Triangle[i].vv0].y -
Agirliknok[i].y) * (Vertex[Triangle[i].vv1].z - Agirliknok[i].z)) -
((Vertex[Triangle[i].vv0].z - Agirliknok[i].z) * (Vertex[Triangle[i].vv1].y
- Agirliknok[i].y));
                normal[i].y = ((Vertex[Triangle[i].vv0].z -
Agirliknok[i].z) * (Vertex[Triangle[i].vv1].x - Agirliknok[i].x)) -
((Vertex[Triangle[i].vv0].x - Agirliknok[i].x) * (Vertex[Triangle[i].vv1].z
- Agirliknok[i].z));
                normal[i].z = ((Vertex[Triangle[i].vv0].x -
Agirliknok[i].x) * (Vertex[Triangle[i].vv1].y - Agirliknok[i].y)) -
((Vertex[Triangle[i].vv0].y - Agirliknok[i].y) * (Vertex[Triangle[i].vv1].x
- Agirliknok[i].x));


            }

// normalleri aynı olan üçgenlerin vertexlerini kendi içinde tekrar etmicek
şekilde toplandığı adı kova olan bir list tanımlanır.
// List, düzlemin ta kendisidir.burda bulunan vertexlerinin ortalamasını
yani düzlemin ağırlık merkezi belirleyecektir.


   List<dVertex> kova = new List<dVertex>();
   List<dVertex> gurultu = new List<dVertex>();
   double tx, ty, tz;
   double ortx, orty, ortz;
```

```
for (int i = 1; i <= HowMany; i++)
     {

     if (i == 1)// giriş yapılır
      {
       kova.Add(Vertex[Triangle[i].vv0]);
       kova.Add(Vertex[Triangle[i].vv1]);
       kova.Add(Vertex[Triangle[i].vv2]);

        tx = Vertex[Triangle[i].vv0].x + Vertex[Triangle[i].vv1].x +
Vertex[Triangle[i].vv2].x;
        ty = Vertex[Triangle[i].vv0].y + Vertex[Triangle[i].vv1].y +
Vertex[Triangle[i].vv2].y;
        tz = Vertex[Triangle[i].vv0].z + Vertex[Triangle[i].vv1].z +
Vertex[Triangle[i].vv2].z;

        for (int j = 1; j <= HowMany; j++)//aynı normale sahip diğer
ücgenlerin vertexlerini kovaya atılır
         {
           if ((normal[i].x == normal[j].x) && (normal[i].y == normal[j].y)
&& (normal[i].z == normal[j].z))
         {
           int varmi0 = 0;
           int varmi1 = 0;
           int varmi2 = 0;
           for (int k = 0; k < kova.Count; k++)//kovada daha öncesinden o
vertex varmı? yokmu?- yoksa kovaya ekleme yap
            {
              if ((Vertex[Triangle[j].vv0].x == kova[k].x) &&
(Vertex[Triangle[j].vv0].y == kova[k].y) && (Vertex[Triangle[j].vv0].z ==
kova[k].z))
             {
               varmi0++;
             }
              if ((Vertex[Triangle[j].vv1].x == kova[k].x) &&
(Vertex[Triangle[j].vv1].y == kova[k].y) && (Vertex[Triangle[j].vv1].z ==
kova[k].z))
             {
               varmi1++;
             }
             if ((Vertex[Triangle[j].vv2].x == kova[k].x) &&
(Vertex[Triangle[j].vv2].y == kova[k].y) && (Vertex[Triangle[j].vv2].z ==
kova[k].z))
             {
              varmi2++;
             }

            }
```

```
            if (varmi0 == 0)
             {
               kova.Add(Vertex[Triangle[j].vv0]);
               tx = tx + Vertex[Triangle[j].vv0].x;
               ty = ty + Vertex[Triangle[j].vv0].y;
               tz = tz + Vertex[Triangle[j].vv0].z;
             }
               if (varmi1 == 0)
               {
                 kova.Add(Vertex[Triangle[j].vv1]);
                 tx = tx + Vertex[Triangle[j].vv1].x;
                  ty = ty + Vertex[Triangle[j].vv1].y;
                  tz = tz + Vertex[Triangle[j].vv1].z;
                 }
                 if (varmi2 == 0)
                 {
                   kova.Add(Vertex[Triangle[j].vv2]);
                   tx = tx + Vertex[Triangle[j].vv2].x;
                   ty = ty + Vertex[Triangle[j].vv2].y;
                   tz = tz + Vertex[Triangle[j].vv2].z;
                 }

               }
             }

//düzlemin ağırlık noktasının hesaplanmasında sıra...ortx,y,z düzlemin
ağırlık merkezinin koordinatlarıdır.
                          ortx = tx / kova.Count;
                          orty = ty / kova.Count;
                          ortz = tz / kova.Count;
//gürültü belirlenirken Ölçü - Gerçek değer= Hata miktarı belirlenmeli..+-
değerdedir mutlağı alınır...kabul edilebilir miktar hata payıda
denilebilir! gerçek değerimiz düzleme ait olan ağırlık noktamızdır..incelik
belirtir! Uzaklık hassaiyeti söz konusudur.

    double hatapayı = 200;
    for (int k = 0; k < kova.Count; k++)
    {
     if ((Math.Abs(kova[k].x - ortx) > hatapayı) || (Math.Abs(kova[k].y -
orty) > hatapayı) || (Math.Abs(kova[k].x - ortx) > hatapayı))
       {
         if (gurultu.Count==0)
           {
            gurultu.Add(kova[k]);
           }
```

```
else
          {
          int tekrar = 0;//gurultu kendi içinde tekrar ediyormu?
           for (int f = 0; f < gurultu.Count; f++)
           {
            if ((kova[k].x == gurultu[f].x) && (kova[k].y == gurultu[f].y)
&& (kova[k].z == gurultu[f].z))
               {tekrar++;}
            }
                if (tekrar == 0)
                {
                 gurultu.Add(kova[k]);
                 }
                }

                }
              }
          }
       else
      {
        int kontrol = 0;

       for (int t = 0; t < i; t++)// o normal için daha önce bir işlem
yapılıp yapılmadığını öğrenilir
       {
        if ((normal[i].x == normal[t].x) && (normal[i].y == normal[t].y) &&
(normal[i].z == normal[t].z))
       {
        kontrol++;
       }
       }
       if (kontrol == 0)// yeni bir normal ile işlem başlıyor.
       {
           kova.Add(Vertex[Triangle[i].vv0]);
           kova.Add(Vertex[Triangle[i].vv1]);
           kova.Add(Vertex[Triangle[i].vv2]);

       tx = Vertex[Triangle[i].vv0].x + Vertex[Triangle[i].vv1].x +
Vertex[Triangle[i].vv2].x;
       ty = Vertex[Triangle[i].vv0].y + Vertex[Triangle[i].vv1].y +
Vertex[Triangle[i].vv2].y;
       tz = Vertex[Triangle[i].vv0].z + Vertex[Triangle[i].vv1].z +
Vertex[Triangle[i].vv2].z;

            for (int j = 1; j <= HowMany; j++)
            {
              if ((normal[i].x == normal[j].x) && (normal[i].y ==
normal[j].y) && (normal[i].z == normal[j].z))
               {
                 int varmi0 = 0;
                 int varmi1 = 0;
```

```
                int varmi2 = 0;
for (int k = 0; k < kova.Count; k++)//kovada daha öncesinden o vertex
varmı? yokmu?- yoksa kovaya ekle
   {
    if ((Vertex[Triangle[j].vv0].x == kova[k].x) &&
(Vertex[Triangle[j].vv0].y == kova[k].y) && (Vertex[Triangle[j].vv0].z ==
kova[k].z))
    {
     varmi0++;
     }
     if ((Vertex[Triangle[j].vv1].x == kova[k].x) &&
(Vertex[Triangle[j].vv1].y == kova[k].y) && (Vertex[Triangle[j].vv1].z ==
kova[k].z))
      {
       varmi1++;
       }
     if ((Vertex[Triangle[j].vv2].x == kova[k].x) &&
(Vertex[Triangle[j].vv2].y == kova[k].y) && (Vertex[Triangle[j].vv2].z ==
kova[k].z))
       {
        varmi2++;
       }

    }
    if (varmi0 == 0)
    {
         kova.Add(Vertex[Triangle[j].vv0]);
         tx = tx + Vertex[Triangle[j].vv0].x;
         ty = ty + Vertex[Triangle[j].vv0].y;
         tz = tz + Vertex[Triangle[j].vv0].z;
    }
    if (varmi1 == 0)
     {
        kova.Add(Vertex[Triangle[j].vv1]);
        tx = tx + Vertex[Triangle[j].vv1].x;
        ty = ty + Vertex[Triangle[j].vv1].y;
        tz = tz + Vertex[Triangle[j].vv1].z;
      }
    if (varmi2 == 0)
    {
        kova.Add(Vertex[Triangle[j].vv2]);
        tx = tx + Vertex[Triangle[j].vv2].x;
        ty = ty + Vertex[Triangle[j].vv2].y;
        tz = tz + Vertex[Triangle[j].vv2].z;
     }


   }

 }
```

```
//düzlemin ağırlık noktasının hesaplanmasında sıra...ortx,y,z düzlemin
ağırlık noktasının koordinatlarıdır.
           ortx = tx / kova.Count;
           orty = ty / kova.Count;
           ortz = tz / kova.Count;

   double hatapayı = 200; // eşik değeri olarak dışardan belirlenen değer
     for (int k = 0; k < kova.Count; k++)
     {
      if ((Math.Abs(kova[k].x - ortx) > hatapayı) || (Math.Abs(kova[k].y -
orty) > hatapayı) || (Math.Abs(kova[k].x - ortx) > hatapayı))
      {
         if (gurultu.Count == 0)
           {
             gurultu.Add(kova[k]);
           }
         else
           {
             int tekrar = 0;//gurultu kendi içinde tekrar ediyormu?
             for (int f = 0; f < gurultu.Count; f++)
             {
                if ((kova[k].x == gurultu[f].x) && (kova[k].y ==
gurultu[f].y) && (kova[k].z == gurultu[f].z))
                {
                 tekrar++;
                 }
              }
             if (tekrar == 0)
             {
               gurultu.Add(kova[k]);
             }
            }

          }
       }
   }

}
//ayrıca döngü i değiştirmeye geçerken kova boşaltılmalı!

kova.Clear();
}
StreamWriter sw5 = new
StreamWriter("C:\\Users\\USER\\Desktop\\gurultu.txt");

 for (int k = 0; k < gurultu.Count; k++)
 {


  sw5.Write(gurultu[k].x);
```

```
  sw5.Write('\t');
  sw5.Write(gurultu[k].y);
  sw5.Write('\t');
  sw5.Write(gurultu[k].z);

  sw5.WriteLine();

  }
  sw5.Close();
//temizlenmiş modele ulaşmamız gerekli şimdide!

  List<dVertex> modelnokta = new List<dVertex>();

   for (int m = 0; m < tPoints; m++)
   {
     int varmi=0;
     for (int n =0; n < gurultu.Count;n++)
      {
       if
((Vertex[m].x==gurultu[n].x)&&(Vertex[m].y==gurultu[n].y)&&(Vertex[m].z==gu
rultu[n].z))
        {
          varmi++;
          }
       }

     if(varmi==0)
        {
          modelnokta.Add(Vertex[m]);
        }
     }

     StreamWriter sw7 = new
StreamWriter("C:\\Users\\USER\\Desktop\\temizmodel.txt");

     for (int k = 0; k < modelnokta.Count; k++)
      {
       sw7.Write(modelnokta[k].x);
       sw7.Write('\t');
       sw7.Write(modelnokta[k].y);
       sw7.Write('\t');
       sw7.Write(modelnokta[k].z);

        sw7.WriteLine();


       }
      sw7.Close();
}
}}
```

ÖZGEÇMİŞ

Doğum tarihi	26.11.1983	
Doğum yeri	İskenderun	
Lise	1997-2001	Yalova Lisesi (Y.D.A)
Lisans	2002-2007	Yıldız Üniversitesi İnşaat Fak. Harita Mühendisliği Bölümü

Çalıştığı kurum(lar)

2007-2008	NİK İnşaat Ltd. Şti.
2008-Devam ediyor	İSTANBUL İl Özel İdaresi Etüt Proje Müdürlüğü